Contenido de la documentación de Python

- Qué hay de nuevo en Python
  - What’s new in Python 3.14
    - Summary – Release highlights
    - New features
      - **PEP 649** & **PEP 749**: Deferred evaluation of annotations
      - **PEP 734**: Multiple interpreters in the standard library
      - **PEP 750**: Template string literals
      - **PEP 768**: Safe external debugger interface
      - A new type of interpreter
      - Free-threaded mode improvements
      - Improved error messages
      - **PEP 784**: Zstandard support in the standard library
      - Asyncio introspection capabilities
      - Concurrent safe warnings control
    - Other language changes
      - Built-ins
      - Command line and environment
      - PEP 758: Allow `except` and `except*` expressions without brackets
      - PEP 765: Control flow in `finally` blocks
      - Incremental garbage collection
      - Default interactive shell
    - New modules
    - Improved modules

- argparse
- ast
- asyncio
- calendar
- concurrent.futures
- configparser
- contextvars
- ctypes
- curses
- datetime
- decimal
- difflib
- dis
- errno
- faulthandler
- fnmatch
- fractions
- functools
- getopt
- getpass
- graphlib
- heapq
- hmac
- http
- imaplib
- inspect
- io
- json
- linecache
- logging.handlers
- math
- mimetypes
- multiprocessing
- operator
- os
- os.path
- pathlib
- pdb
- pickle
- platform

  - pydoc
  - re
  - socket
  - ssl
  - struct
  - symtable
  - sys
  - sys.monitoring
  - sysconfig
  - tarfile
  - threading
  - tkinter
  - turtle
  - types
  - typing
  - unicodedata
  - unittest
  - urllib
  - uuid
  - webbrowser
  - zipfile
- Optimizations
  - asyncio
  - base64
  - bdb
  - difflib
  - gc
  - io
  - pathlib
  - pdb
  - textwrap
  - uuid
  - zlib
- Removed
  - argparse
  - ast
  - asyncio

    - email
    - importlib.abc
    - itertools
    - pathlib
    - pkgutil
    - pty
    - sqlite3
    - urllib
- Deprecated
    - New deprecations
    - Pending removal in Python 3.15
    - Pending removal in Python 3.16
    - Pending removal in Python 3.17
    - Pending removal in Python 3.18
    - Pending removal in Python 3.19
    - Pending removal in future versions
- CPython bytecode changes
    - Pseudo-instructions
- C API changes
    - Python configuration C API
    - New features in the C API
    - Limited C API changes
    - Removed C APIs
    - Deprecated C APIs
        - Pending removal in Python 3.15
        - Pending removal in Python 3.16
        - Pending removal in Python 3.18
        - Pending removal in future versions
- Build changes
    - **`build-details.json`**
    - Discontinuation of PGP signatures
    - Free-threaded Python is officially

        - supported
        - Binary releases for the experimental just-in-time compiler
    - Porting to Python 3.14
        - Changes in the Python API
        - Changes in annotations (**PEP 649** and **PEP 749**)
            - Implications for annotated code
            - Implications for readers of `__annotations__`
            - Related changes
            - `from __future__ import annotations`
        - Changes in the C API
    - Notable changes in 3.14.1
- What’s New In Python 3.13
    - Summary – Release Highlights
    - New Features
        - A better interactive interpreter
        - Improved error messages
        - Free-threaded CPython
        - An experimental just-in-time (JIT) compiler
        - Defined mutation semantics for `locals()`
        - Support for mobile platforms
    - Other Language Changes
    - New Modules
    - Improved Modules
        - argparse
        - array
        - ast
        - asyncio
        - base64

- compileall
- concurrent.futures
- configparser
- copy
- ctypes
- dbm
- dis
- doctest
- email
- enum
- fractions
- glob
- importlib
- io
- ipaddress
- itertools
- marshal
- math
- mimetypes
- mmap
- multiprocessing
- os
- os.path
- pathlib
- pdb
- queue
- random
- re
- shutil
- site
- sqlite3
- ssl
- statistics
- subprocess
- sys
- tempfile
- time
- tkinter
- traceback
- types

    - typing
    - unicodedata
    - venv
    - warnings
    - xml
    - zipimport

- Optimizations
- Removed Modules And APIs

    - PEP 594: Remove «dead batteries» from the standard library
    - 2to3
    - builtins
    - configparser
    - importlib.metadata
    - locale
    - opcode
    - optparse
    - pathlib
    - re
    - tkinter.tix
    - turtle
    - typing
    - unittest
    - urllib
    - webbrowser

- New Deprecations

    - Pending removal in Python 3.14
    - Pending removal in Python 3.15
    - Pending removal in Python 3.16
    - Pending removal in Python 3.17
    - Pending removal in Python 3.18
    - Pending removal in Python 3.19
    - Pending removal in future versions

- CPython Bytecode Changes
- C API Changes

            - New Features
            - Changed C APIs
            - Limited C API Changes
            - Removed C APIs
            - Deprecated C APIs
                - Pending removal in Python 3.14
                - Pending removal in Python 3.15
                - Pending removal in Python 3.16
                - Pending removal in Python 3.18
                - Pending removal in future versions
        - Build Changes
        - Porting to Python 3.13
            - Changes in the Python API
            - Changes in the C API
        - Regression Test Changes
    - Novedades de Python 3.12
        - Resumen: aspectos destacados de la versión
        - Nuevas características
            - PEP 695: Sintaxis de parámetro de tipo
            - PEP 701: Formalización sintáctica de cadenas f
            - PEP 684: un GIL por intérprete
            - PEP 669: Monitoreo de bajo impacto para CPython
            - PEP 688: Hacer accesible el protocolo de búfer en Python
            - PEP 709: Integración de la comprensión
            - Mensajes de error mejorados
        - Nuevas funciones relacionadas con las sugerencias de escritura
            - PEP 692: uso de **`TypedDict`** para una escritura **`**kwargs`** más precisa

    - PEP 698: Decorador override para escritura estática

- Otros cambios del lenguaje
- Nuevos módulos
- Módulos mejorados

    - array
    - asincio
    - calendar
    - csv
    - dis
    - fractions
    - importlib.resources
    - inspect
    - itertools
    - math
    - os
    - os.path
    - pathlib
    - platform
    - pbd
    - random
    - shutil
    - sqlite3
    - statistics
    - sys
    - tempfile
    - threading
    - tkinter
    - tokenize
    - types
    - typing
    - unicodedata
    - unittest
    - uuid

- Optimizaciones
- Cambios en el código de bytes de CPython
- Demostraciones y herramientas

- Obsoleto
  - Pending removal in Python 3.13
  - Pending removal in Python 3.14
  - Pending removal in Python 3.15
  - Pending removal in Python 3.16
  - Pending removal in Python 3.17
  - Pending removal in future versions
- Eliminado
  - asynchat y asyncore
  - configparser
  - distutils
  - ensurepip
  - enum
  - ftplib
  - gzip
  - hashlib
  - importlib
  - imp
  - io
  - locale
  - smtpd
  - sqlite3
  - ssl
  - unittest
  - webbrowser
  - xml.etree.ElementTree
  - zipimport
  - Otros
- Portar a Python 3.12
  - Cambios en la API de Python
- Cambios de compilación
- Cambios en la API de C
  - Nuevas características
  - Portar a Python 3.12

        - Obsoleto
            - Pending removal in Python 3.14
            - Pending removal in Python 3.15
            - Pending removal in Python 3.16
            - Pending removal in future versions
        - Eliminado
- Qué hay de nuevo en Python 3.11
    - Resumen – Aspectos destacados de la versión
    - Nuevas características
        - PEP 657: Ubicaciones de errores detallados en rastreos
        - PEP 654: Grupos de excepción y **`except*`**
        - PEP 678: Las excepciones se pueden enriquecer con notas
        - Mejoras en el iniciador de Windows **`py.exe`**
    - Nuevas funciones relacionadas con las sugerencias de tipo
        - PEP 646: Genéricos Variádicos
        - PEP 655: Marcado de elementos **`TypedDict`** individuales como requeridos o no requeridos
        - PEP 673: tipo **`Self`**
        - PEP 675: tipo de cadena literal arbitraria
        - PEP 681: Transformaciones de clases de datos
        - PEP 563 puede no ser el futuro
    - Otros cambios de idioma
    - Otros cambios en la implementación de CPython
    - Nuevos Módulos
    - Módulos mejorados
        - asíncio

    - contextlib
    - clases de datos
    - fecha y hora
    - enumeración
    - fcntl
    - fracciones
    - herramientas funcionales
    - gzip
    - hashlib
    - IDLE y libre de inactividad
    - inspeccionar
    - lugar
    - Inicio sesión
    - Matemáticas
    - operador
    - sistema operativo
    - rutalib
    - re
    - cerrar
    - enchufe
    - sqlite3
    - cuerda
    - sistema
    - configuración del sistema
    - archivo temporal
    - enhebrar
    - tiempo
    - tkinter
    - rastrear
    - mecanografía
    - unicodedata
    - prueba de unidad
    - venv
    - advertencias
    - archivo zip

- Optimizaciones
- CPython más rápido

    - Inicio más rápido

        - Importaciones congeladas / Objetos de código estático

    - Tiempo de ejecución más rápido

        - Marcos de Python más baratos y perezosos
        - Llamadas a funciones de Python en línea
        - PEP 659: Intérprete Adaptativo Especializado

    - Varios
    - Preguntas más frecuentes

        - ¿Cómo debo escribir mi código para utilizar estas aceleraciones?
        - ¿CPython 3.11 usará más memoria?
        - No veo ninguna aceleración en mi carga de trabajo. ¿Por qué?
        - ¿Existe un compilador JIT?

    - Sobre

- Cambios en el código de bytes de CPython

    - Nuevos códigos de operación
    - Códigos de operación reemplazados
    - Códigos de operación cambiados/eliminados

- Obsoleto

    - Idioma/Construidos
    - Módulos
    - Biblioteca estándar

- Eliminación pendiente en Python 3.12
- Remoto
- Migración a Python 3.11
- Construir cambios
- Cambios en la API de C

        - Nuevas características
        - Migración a Python 3.11
        - Obsoleto
        - Eliminación pendiente en Python 3.12
        - Remoto
    - Notable changes in 3.11.4
        - tarfile
    - Notable changes in 3.11.5
        - OpenSSL
- Qué hay de nuevo en Python 3.10
    - Resumen: aspectos destacados de la versión
    - Nuevas características
        - Administradores de contexto entre paréntesis
        - Mejores mensajes de error
            - SyntaxErrors
            - Errores de sangría
            - AttributeErrors
            - NameErrors
        - PEP 626: números de línea precisos para depuración y otras herramientas
        - PEP 634: Coincidencia de patrones estructurales
            - Sintaxis y operaciones
            - Enfoque declarativo
            - Patrón simple: coincidir con un literal
                - Comportamiento sin el comodín
            - Patrones con un literal y una variable

    - Patrones y clases
      - Patrones con parámetros posicionales
    - Patrones anidados
    - Patrones complejos y el comodín
    - Guardia
    - Otras características clave
  - Opción opcional **`EncodingWarning`** y **`encoding="locale"`**
- Nuevas funciones relacionadas con las sugerencias de tipos
  - PEP 604: Nuevo tipo de operador unión
  - PEP 612: Variables de especificación de parámetros
  - PEP 613: TypeAlias
  - PEP 647: protectores de tipo definidos por el usuario
- Otros cambios de idioma
- Nuevos módulos
- Módulos mejorados
  - asyncio
  - argumentar
  - formación
  - asynchat, asyncore, smtpd
  - base64
  - bdb
  - bisecar
  - códecs
  - colecciones.abc
  - contextlib
  - maldiciones
  - clases de datos
    - __slots__

    - Campos solo de palabras clave

- distutils
- doctest
- codificaciones
- enum
- entrada de archivo
- manipulador de faltas
- gc
- glob
- hashlib
- hmac
- IDLE e idlelib
- importlib.metadata
- inspeccionar
- itertools
- caché de línea
- os
- os.path
- pathlib
- plataforma
- pprint
- py_compile
- pyclbr
- dejar de lado
- estadísticas
- sitio
- enchufe
- ssl
- sqlite3
- sys
- _hilo
- enhebrar
- rastrear
- tipos
- mecanografía
- prueba de unidad
- urllib.parse
- xml
- zipimport

        - Optimizaciones
        - Obsoleto
        - Eliminado
        - Portar a Python 3.10
            - Cambios en la sintaxis de Python
            - Cambios en la API de Python
            - Cambios en la API de C
        - Cambios en el código de bytes de CPython
        - Construir cambios
        - Cambios en la API de C
            - PEP 652: Mantenimiento del ABI estable
            - Nuevas características
            - Portar a Python 3.10
            - Obsoleto
            - Eliminado
        - Característica de seguridad notable en 3.10.7
        - Característica de seguridad notable en 3.10.8
        - Cambios notables en 3.10.12
            - archivo tar
    - Qué hay de nuevo en Python 3.9
        - Resumen: aspectos destacados de la versión
        - Debe verificar DeprecationWarning en su código
        - Nuevas características
            - Operadores de combinación y actualización de diccionarios
            - Nuevos métodos de cadena para eliminar prefijos y sufijos
            - Tipos genéricos de sugerencia en colecciones estándar
            - Nuevo analizador
        - Otros cambios de idioma
        - Nuevos módulos

    - zoneinfo
    - Graphlib
- Módulos mejorados
    - ast
    - asyncio
    - compilar todo
    - Futuros concurrentes
    - maldiciones
    - fecha y hora
    - distutils
    - fcntl
    - ftplib
    - GC
    - hashlib
    - http
    - IDLE e idlelib
    - imaplib
    - importlib
    - inspeccionar
    - dirección IP
    - Matemáticas
    - multiprocesamiento
    - nntplib
    - os
    - Pathlib
    - pdb
    - poplib
    - pprint
    - Pydoc
    - aleatorio
    - señal
    - smtplib
    - enchufe
    - tiempo
    - sys
    - tracemalloc
    - mecanografía
    - unicodedata

    - venv
    - xml
  - Optimizaciones
  - Obsoleto
  - Remoto
  - Portar a Python 3.9
    - Cambios en la API de Python
    - Cambios en la API de C
    - Cambios en el código de bytes de CPython
  - Construir cambios
  - Cambios en la API de C
    - Nuevas características
    - Portar a Python 3.9
    - Remoto
  - Cambios notables en Python 3.9.1
    - mecanografía
    - Compatibilidad con macOS 11.0 (Big Sur) y Apple Silicon Mac
  - Cambios notables en Python 3.9.2
    - colecciones.abc
    - urllib.parse
  - Notable changes in Python 3.9.3
  - Notable changes in Python 3.9.5
    - urllib.parse
  - Notable security feature in 3.9.14
  - Notable changes in 3.9.17
    - tarfile
- Qué hay de nuevo en Python 3.8

- Resumen – Aspectos destacados de la versión
- Nuevas características
    - Expresiones de asignación
    - Parámetros solo posicionales
    - Caché del sistema de archivos paralelo para archivos de bytecode compilados
    - La compilación de depuración usa la misma ABI que la compilación de lanzamiento
    - los f-strings soportan `=` para expresiones autodocumentadas y depuración
    - PEP 578: Ganchos de auditoría en tiempo de ejecución de Python
    - PEP 587: Configuración de inicialización de Python
    - PEP 590: Vectorcall: un protocolo de llamada rápida para CPython
    - Protocolo 5 de Pickle con búferes de datos fuera de banda
- Otros cambios en el lenguaje
- Nuevos módulos
- Módulos mejorados
    - ast
    - asyncio
    - builtins
    - collections
    - cProfile
    - csv
    - curses
    - ctypes
    - datetime
    - functools
    - gc
    - gettext
    - gzip
    - IDLE e idlelib
    - inspect
    - io

    - itertools
    - json.tool
    - logging
    - math
    - mmap
    - multiprocessing
    - os
    - os.path
    - pathlib
    - pickle
    - plistlib
    - pprint
    - py_compile
    - shlex
    - shutil
    - socket
    - ssl
    - statistics
    - sys
    - tarfile
    - threading
    - tokenize
    - tkinter
    - time
    - typing
    - unicodedata
    - unittest
    - venv
    - weakref
    - xml
    - xmlrpc

- Optimizaciones
- Cambios en la compilación y la API de C
- Obsoleto
- APIs y características eliminadas
- Portando a Python 3.8

    - Cambios en el comportamiento de Python
    - Cambios en la API de Python

        - Cambios en la API de C
        - Cambios en el bytecode de CPython
        - Demos y herramientas
    - Cambios notables en Python 3.8.1
    - Notable changes in Python 3.8.2
    - Notable changes in Python 3.8.3
    - Cambios notables en Python 3.8.8
    - Notable changes in Python 3.8.9
    - Notable changes in Python 3.8.10
        - macOS 11.0 (Big Sur) and Apple Silicon Mac support
    - Notable changes in Python 3.8.10
        - urllib.parse
    - Cambios notables en Python 3.8.1
        - Cambios en la API de Python
    - Notable security feature in 3.8.14
    - Notable changes in 3.8.17
        - tarfile
- Qué hay de nuevo en Python 3.7
    - Resumen – Lanzamientos Destacados
    - Nuevas Características
        - PEP 563: Evaluación pospuesta de anotaciones
        - PEP 538: Coerción de configuración regional del Legado de C
        - PEP 540: Modo de tiempo de ejecución UTF-8 forzado
        - PEP 553: Incorporada en `breakpoint()`
        - PEP 539: Nueva API C para almacenamiento local de subprocesos
        - PEP 562: Personalización del acceso a los

  atributos del módulo
  - PEP 564: Nuevas funciones de tiempo con resolución de nanosegundos
  - PEP 565: Mostrar *DeprecationWarning* en **`__main__`**
  - PEP 560: Soporte básico para el módulo de **`typing`** y tipos genéricos
  - PEP 552: Archivos .pyc basados en hash
  - PEP 545: Traducciones de Documentaciones de Python
  - Modo de desarrollo de Python (-X dev)

- Otros cambios en el lenguaje
- Nuevos módulos

  - *contextvars*
  - *dataclasses*
  - *importlib.resources*

- Módulos mejorados

  - *argparse*
  - *asyncio*
  - *binascii*
  - *calendar*
  - *collections*
  - *compileall*
  - *concurrent.futures*
  - *contextlib*
  - *cProfile*
  - *crypt*
  - *datetime*
  - *dbm*
  - *decimal*
  - *dis*
  - *distutils*
  - *enum*
  - *functools*
  - *gc*
  - *hmac*
  - *http.client*

- *http.server*
- *idlelib and IDLE*
- *importlib*
- *io*
- *ipaddress*
- *itertools*
- *locale*
- *logging*
- *math*
- *mimetypes*
- *msilib*
- *multiprocessing*
- *os*
- *pathlib*
- *pdb*
- *py_compile*
- *pydoc*
- *queue*
- *re*
- *signal*
- *socket*
- *socketserver*
- *sqlite3*
- *ssl*
- *string*
- *subprocess*
- *sys*
- *time*
- *tkinter*
- *tracemalloc*
- *types*
- *unicodedata*
- *unittest*
- *unittest.mock*
- *urllib.parse*
- *uu*
- *uuid*
- *warnings*
- xml
- *xml.etree*

    - *xmlrpc.server*
    - *zipapp*
    - *zipfile*

- Cambios en la API C
- Construir cambios
- Optimizaciones
- Otros cambios de implementación de CPython
- Comportamiento obsoleto de Python
- Módulos, funciones y métodos de Python obsoletos

    - *aifc*
    - *asyncio*
    - *collections*
    - *dbm*
    - *enum*
    - *gettext*
    - *importlib*
    - *locale*
    - *macpath*
    - *threading*
    - *socket*
    - *ssl*
    - *sunau*
    - *sys*
    - *wave*

- Funciones y tipos obsoletos de la API C
- Eliminación de soporte de plataforma
- Eliminaciones de API y funciones
- Eliminaciones de módulos
- Cambios solo en Windows
- Portando a Python 3.7

    - Cambios en el comportamiento de Python
    - Cambios en la API Python
    - Cambios en la API de C
    - Cambios en el código de bytes de CPython
    - Cambios solo en Windows
    - Otra implementación de cambios en

        CPython

    - Cambios notables en Python 3.7.1
    - Cambios notables en Python 3.7.2
    - Cambios notables en Python 3.7.6
    - Cambios notables en Python 3.7.10
    - Cambios notables en Python 3.7.11
    - Cambios notables en Python 3.7.14

- Qué hay de nuevo en Python 3.6

    - Resumen: aspectos destacados de la versión
    - Nuevas características

        - PEP 498: Literales de cadena formateados
        - PEP 526: Sintaxis para anotaciones de variables
        - PEP 515: subrayados en literales numéricos
        - PEP 525: Generadores asíncronos
        - PEP 530: Comprensiones asincrónicas
        - PEP 487: personalización más sencilla de la creación de clases
        - PEP 487: Mejoras en el protocolo descriptor
        - PEP 519: Agregar un protocolo de ruta del sistema de archivos
        - PEP 495: desambiguación de la hora local
        - PEP 529: cambie la codificación del sistema de archivos de Windows a UTF-8
        - PEP 528: cambie la codificación de la consola de Windows a UTF-8
        - PEP 520: Conservación del orden de definición de atributos de clase
        - PEP 468: Conservación del orden de los argumentos de las palabras clave
        - Nueva implementación de dict
        - PEP 523: Agregar una API de evaluación de marcos a CPython
        - Variable de entorno PYTHONMALLOC
        - Soporte de sondeo DTrace y SystemTap

- Otros cambios de idioma
- Nuevos módulos

    - misterios

- Módulos mejorados

    - formación
    - ast
    - asyncio
    - binascii
    - cmath
    - colecciones
    - concurrent.futures
    - contextlib
    - fecha y hora
    - decimal
    - distutils
    - email
    - codificaciones
    - enumeración
    - manipulador de faltas
    - entrada de archivo
    - hashlib
    - http.client
    - idlelib y IDLE
    - importlib
    - inspeccionar
    - json
    - logging
    - math
    - multiprocesamiento
    - os
    - pathlib
    - pdb
    - pepinillo
    - pepinillos
    - pydoc
    - aleatorio
    - re

    - readline
    - rlcompleter
    - shlex
    - sitio
    - sqlite3
    - enchufe
    - servidor de sockets
    - ssl
    - Estadísticas
    - estructura
    - subproceso
    - sys
    - telnetlib
    - tiempo
    - cronométralo
    - tkinter
    - rastrear
    - tracemalloc
    - mecanografía
    - unicodedata
    - unittest.mock
    - urllib.request
    - urllib.robotparser
    - venv
    - advertencias
    - winreg
    - winsonido
    - xmlrpc.client
    - archivo zip
    - zlib
- Optimizaciones
- Cambios en la API de Build y C
- Otras mejoras
- Obsoleto
    - Nuevas palabras clave
    - Comportamiento de Python obsoleto
    - Módulos, funciones y métodos de Python obsoletos

    - asynchat
    - asyncore
    - dbm
    - distutils
    - grp
    - importlib
    - os
    - re
    - ssl
    - tkinter
    - venv
  - xml
  - Funciones y tipos obsoletos de la API de C
  - Opciones de compilación obsoletas
- Remoto
  - Eliminaciones de API y funciones
- Portar a Python 3.6
  - Cambios en el comportamiento del comando “python”
  - Cambios en la API de Python
  - Cambios en la API de C
  - Cambios en el código de bytes de CPython
- Cambios notables en Python 3.6.2
  - Nuevo objetivo de compilación **`make regen-all`**
  - Eliminación del objetivo de compilación **`make touch`**
- Cambios notables en Python 3.6.4
- Cambios notables en Python 3.6.5
- Cambios notables en Python 3.6.7
- Cambios notables en Python 3.6.10
- Cambios notables en Python 3.6.13
- Cambios notables en Python 3.6.14

- Qué hay de nuevo en Python 3.5
    - Resumen – Aspectos destacados de la versión
    - Nuevas características
        - PEP 492 - Corrutinas con sintaxis async y await
        - PEP 465 - Un operador infijo dedicado para la multiplicación de matrices
        - PEP 448 - Generalizaciones de desembalaje adicionales
        - PEP 461 - soporte de formateo porcentual para bytes y bytearray
        - PEP 484 - Indicador de tipos
        - PEP 471 - Función os.scandir() – un iterador de directorio mejor y más rápido
        - PEP 475: Reintentar las llamadas al sistema que fallan con EINTR
        - PEP 479: Cambiar el gestor de StopIteration dentro de generadores
        - PEP 485: Una función para probar la igualdad aproximada
        - PEP 486: Hacer que el launcher de Python sea consciente de los entornos virtuales
        - PEP 488: Eliminación de archivos PYO
        - PEP 489: Inicialización del módulo de extensión multifase
    - Otros cambios en el lenguaje
    - Nuevos módulos
        - typing
        - zipapp
    - Módulos mejorados
        - argparse
        - asyncio
        - bz2
        - cgi
        - cmath

- code
- collections
- collections.abc
- compileall
- concurrent.futures
- configparser
- contextlib
- csv
- curses
- dbm
- difflib
- distutils
- doctest
- email
- enum
- faulthandler
- functools
- glob
- gzip
- heapq
- http
- http.client
- idlelib e IDLE
- imaplib
- imghdr
- importlib
- inspect
- io
- ipaddress
- json
- linecache
- locale
- logging
- lzma
- math
- multiprocessing
- operator
- os
- pathlib
- pickle

  - poplib
  - re
  - readline
  - selectors
  - shutil
  - signal
  - smtpd
  - smtplib
  - sndhdr
  - socket
  - ssl
    - Soporte de memoria BIO
    - Soporte de negociación de protocolo de capa de aplicación
    - Otros cambios
  - sqlite3
  - subprocess
  - sys
  - sysconfig
  - tarfile
  - threading
  - time
  - timeit
  - tkinter
  - traceback
  - types
  - unicodedata
  - unittest
  - unittest.mock
  - urllib
  - wsgiref
  - xmlrpc
  - xml.sax
  - zipfile
- Otros cambios a nivel de módulo
- Optimizaciones
- Cambios en la compilación y la API de C

    - Obsoleto
        - Nuevas palabras clave
        - Comportamiento obsoleto de Python
        - Sistemas operativos no compatibles
        - Módulos, funciones y métodos obsoletos de Python
    - Eliminado
        - APIs y características eliminadas
    - Portando a Python 3.5
        - Cambios en el comportamiento de Python
        - Cambios en la API de Python
        - Cambios en la API de C
    - Cambios notables en Python 3.5.4
        - Nuevo objetivo de compilación **`make regen-all`**
        - Eliminación del objetivo de compilación **`make touch`**
- Qué hay de nuevo en Python 3.4
    - Resumen – Puntos destacados del lanzamiento
    - Nuevas Funciones
        - PEP 453: Arranque explícito de PIP en instalaciones de Python
            - Puesta en marcha de pip por defecto
            - Cambios en la documentación
        - PEP 446: Los descriptores de archivos recién creados no son heredables
        - Mejoras en el manejo de códecs
        - PEP 451: Un tipo ModuleSpec para el sistema de importación
        - Otros cambios de lenguaje

- Nuevos Módulos
    - asyncio
    - ensurepip
    - enum
    - pathlib
    - selectores
    - estadísticas
    - tracemalloc
- Módulos mejorados
    - abc
    - aifc
    - argparse
    - audioop
    - base64
    - colecciones
    - colorsys
    - contextlib
    - dbm
    - dis
    - doctest
    - email
    - archivoecmp
    - functools
    - gc
    - glob
    - hashlib
    - hmac
    - html
    - http
    - idlelib y IDLE
    - importlib
    - inspeccionar
    - dirección IP
    - logging
    - mariscal
    - mmap
    - multiprocesamiento

  - operator
  - os
  - pdb
  - pepinillo
  - plistlib
  - poplib
  - pprint
  - pty
  - pydoc
  - re
  - recurso
  - seleccionar
  - estante
  - shutil
  - smtpd
  - smtplib
  - enchufe
  - sqlite3
  - ssl
  - stat
  - struct
  - subproceso
  - sunau
  - sys
  - tarfile
  - textwrap
  - roscado
  - traceback
  - tipos
  - urllib
  - unittest
  - venv
  - onda
  - weakref
  - xml.etree
  - archivo zip
- Cambios en la implementación de CPython
  - PEP 445: Personalización de los

    Asignadores de Memoria de CPython
    - PEP 442: Finalización segura de objetos
    - PEP 456: Algoritmo Hash seguro e intercambiable
    - PEP 436: Clínica de Argumentación
    - Otros cambios en la API de construcción y C
    - Otras mejoras
    - Optimizaciones significativas
  - Obsoleto
    - Desapariciones en la API de Python
    - Funciones obsoletas
  - Removido
    - Sistemas operativos que ya no son compatibles
    - Eliminación de la API y de las funciones
    - Limpieza del código
  - Adaptación a Python 3.4
    - Cambios en el comportamiento del comando “python”
    - Cambios en la API de Python
    - Cambios en la API de C
  - Cambiado en 3.4.3
    - PEP 476: Habilitar la verificación de certificados por defecto para los clientes http stdlib
- Qué hay de nuevo en Python 3.3
  - Resumen – Aspectos destacados de la versión
  - PEP 405: Entornos virtuales
  - PEP 420: Paquetes para espacios de nombres implícitos
  - PEP 3118: Nueva implementación de vista de

memoria y en la documentación del protocolo del buffer

    - Características
    - Cambios en la API

- PEP 393: Representación flexible de cadenas de caracteres

    - Funcionalidad
    - Rendimiento y uso de recursos

- PEP 397: Lanzador de python para windows
- PEP 3151: Reelaborando de la jerarquía de excepciones de IO y OS
- PEP 380: Sintaxis para delegar en un subgenerador
- PEP 409: Suprimir el contexto de excepción
- PEP 414: Literales *Unicode* explícitos
- PEP 3155: Nombres calificados para clases y funciones
- PEP 412: Diccionario de intercambio de claves
- PEP 362: Objeto de firma de función
- PEP 421: Agregar sys.implementation

    - SimpleNamespace

- Usar importlib como implementación de Import

    - Nuevas APIs
    - Cambios visibles

- Otros cambios de idioma
- Un bloqueo de importación más detallado
- Funciones y tipos incorporados
- Nuevos módulos

    - faulthandler
    - ipaddress
    - lzma

- Módulos mejorados

- abc
- array
- base64
- binascii
- bz2
- códecs
- colecciones
- contextlib
- crypt
- curses
- datetime
- decimal

    - Características
    - Cambios en la API

- email

    - Marco de políticas
    - Política provisional con nueva API de encabezado
    - Otros cambios en la API

- ftplib
- functools
- gc
- hmac
- http
- html
- imaplib
- inspect
- io
- itertools
- logging
- math
- mmap
- multiprocesamiento
- nntplib
- os
- pdb
- pickle

    - pydoc
    - re
    - sched
    - select
    - shlex
    - shutil
    - signal
    - smtpd
    - smtplib
    - socket
    - socketserver
    - sqlite3
    - ssl
    - stat
    - struct
    - subprocess
    - sys
    - tarfile
    - tempfile
    - textwrap
    - threading
    - time
    - types
    - unittest
    - urllib
    - webbrowser
    - xml.etree.ElementTree
    - zlib

- Optimizaciones
- Construcción y cambios en la API de C
- Obsoleto

    - Sistemas operativos no compatibles
    - Módulos , funciones y métodos obsoletos en Python
    - Funciones y tipos obsoletos en la API de C
    - Características obsoletas

- Migración a Python 3.3

        - Portando código Python
        - Portando código C
        - Construyendo extensiones C
        - Cambios en el conmutador de línea de comandos
- Qué hay de nuevo en Python 3.2
    - PEP 384: Definición de un ABI estable
    - PEP 389: Módulo de análisis sintáctico (*Parser*) de línea de comandos Argparse
    - PEP 391: Configuración basada en diccionario para Logging
    - PEP 3148: El módulo **`concurrent.futures`**
    - PEP 3147: Directorios del repositorio de PYC
    - PEP 3149: Archivos .so con etiquetado de versión para ABI
    - PEP 3333: Interfaz de puerta de enlace del servidor web Python v1.0.1
    - Otros cambios de lenguaje
    - Módulos nuevos, mejorados y obsoletos
        - email
        - elementtree
        - functools
        - itertools
        - collections
        - threading
        - datetime and time
        - math
        - abc
        - io
        - reprlib
        - logging
        - csv
        - contextlib
        - decimal and fractions
        - ftp
        - popen
        - select

    - gzip y zipfile
    - tarfile
    - hashlib
    - ast
    - os
    - shutil
    - sqlite3
    - html
    - socket
    - ssl
    - nntp
    - certificados
    - imaplib
    - http.client
    - unittest
    - random
    - poplib
    - asyncore
    - tempfile
    - inspect
    - pydoc
    - dis
    - dbm
    - ctypes
    - site
    - sysconfig
    - pdb
    - configparser
    - urllib.parse
    - mailbox
    - turtledemo
- Multi-threading
- Optimizations
- Unicode
- Codecs
- Documentación
- IDLE
- Repositorio de código
- Cambios en la API de construcción y C

        - Portar a Python 3.2

    - Qué hay de nuevo en Python 3.1

        - PEP 372: Diccionarios ordenados
        - PEP 378: Especificador de formato para el separador de miles
        - Otros cambios del lenguaje
        - Módulos nuevos, mejorados y obsoletos
        - Optimizaciones
        - IDLE
        - Cambios en la compilación y la API de C
        - Portando a Python 3.1

    - Qué hay de nuevo en Python 3.0

        - Escollos comunes

            - *Print* es una función
            - Vistas e iteradores en lugar de listas
            - Comparaciones de ordenamiento
            - Enteros
            - Texto vs. datos en lugar de unicode vs. 8 bits

        - Descripción general de los cambios de sintaxis

            - Nueva sintaxis
            - Sintaxis modificada
            - Sintaxis eliminada

        - Cambios ya presentes en Python 2.6
        - Cambios de biblioteca
        - **PEP 3101**: Un nuevo enfoque al formateo de cadena de caracteres
        - Cambios a excepciones
        - Otros cambios diversos

            - Operadores y métodos especiales
            - Incorporados

        - Construcción y cambios a la API de C

    - Rendimiento
    - Migración a Python 3.0

- Qué hay de nuevo en Python 2.7

    - El futuro de Python 2.x
    - Cambios en el manejo de las advertencias de desuso
    - Características de Python 3.1
    - PEP 372: Adición de un diccionario ordenado a las colecciones
    - PEP 378: Especificador de formato para separador de miles
    - PEP 389: El módulo argparse para el análisis de líneas de comando
    - PEP 391: Configuración basada en diccionarios para el registro
    - PEP 3106: Vistas de diccionario
    - PEP 3137: El objeto memoryview
    - Otros cambios de lenguaje

        - Cambios en el intérprete
        - Optimizaciones

    - Módulos nuevos y mejorados

        - Nuevo módulo: importlib
        - Nuevo módulo: sysconfig
        - ttk: Widgets temáticos para Tk
        - Módulo actualizado: unittest
        - Módulo actualizado: ElementTree 1.3

    - Cambios en la API de construcción y C

        - Cápsulas
        - Cambios específicos en los puertos: Windows
        - Cambios específicos en los puertos: Mac OS X
        - Cambios específicos en los puertos: FreeBSD

        - Otros cambios y correcciones
        - Adaptación a Python 2.7
        - Nuevas funciones añadidas a las versiones de mantenimiento de Python 2.7
            - Dos nuevas variables de entorno para el modo de depuración
            - PEP 434: Excepción de mejora de IDLE para todas las ramas
            - PEP 466: Mejoras en la seguridad de la red para Python 2.7
            - PEP 477: Backport ensurepip (PEP 453) a Python 2.7
                - Puesta en marcha de pip por defecto
                - Cambios en la documentación
            - PEP 476: Habilitar la verificación de certificados por defecto para los clientes http stdlib
            - PEP 493: Herramientas de migración de verificación HTTPS para Python 2.7
            - Nuevo objetivo de construcción **`make regen-all`**
            - Eliminación del objetivo de construcción **`make touch`**
        - Agradecimientos
    - Qué hay de nuevo en Python 2.6
        - Python 3.0
        - Cambios en el proceso de desarrollo
            - Nuevo seguidor de incidentes: Roundup
            - Nuevo formato de documentación: texto reestructurado con Sphinx
        - PEP 343: La sentencia *"with"*
            - Escribiendo gestores de contexto

    - El módulo contextlib

- PEP 366: Importaciones relativas explícitas desde un módulo principal
- PEP 370: Directorio de **`site-packages`** por usuario
- PEP 371: El paquete **`multiprocessing`**
- PEP 3101: Formateo avanzado de cadena de caracteres
- PEP 3105: **`print`** como función
- PEP 3110: Cambios en el manejo de excepciones
- PEP 3112: Literales de bytes
- PEP 3116: Nueva biblioteca de E/S
- PEP 3118: Protocolo revisado de la memoria intermedia
- PEP 3119: Clases base abstractas
- PEP 3127: Soporte y sintaxis de literales enteros
- PEP 3129: Decoradores de clase
- PEP 3141: Una jerarquía de tipos para los números

    - El módulo **`fractions`**

- Otros cambios lingüísticos

    - Optimizaciones
    - Cambios de intérprete

- Módulos nuevos y mejorados

    - El módulo **`ast`**
    - The **`future_builtins`** module
    - El módulo **`json`**: Notación de objetos de JavaScript
    - El módulo **`plistlib`**: Un analizador de listas de propiedades
    - mejoras en ctypes
    - Mejora de la compatibilidad con SSL

- Cancelaciones y eliminaciones
- Cambios en la API de construcción y C

        - Cambios específicos en los puertos: Windows
        - Cambios específicos en los puertos: Mac OS X
        - Cambios específicos en los puertos: IRIX

    - Adaptación a Python 2.6
    - Agradecimientos

- Qué hay de nuevo en Python 2.5

    - PEP 308: Expresiones condicionales
    - PEP 309: Aplicación parcial de funciones
    - PEP 314: Metadatos para paquetes de software Python v1.1
    - PEP 328: Importaciones absolutas y relativas
    - PEP 338: Ejecutando Módulos como Scripts
    - PEP 341: Try/except/finally unificados
    - PEP 342: Nuevas funciones del generador
    - PEP 343: La declaración «con

        - Redacción de Gestores de Contexto
        - El módulo contextlib

    - PEP 352: Las excepciones como clases de nuevo estilo
    - PEP 353: Uso de ssize_t como tipo de índice
    - PEP 357: El método "_index_"
    - Otros cambios lingüísticos

        - Cambios en el intérprete interactivo
        - Optimizaciones

    - Módulos nuevos, mejorados y eliminados

        - El paquete ctypes
        - El paquete ElementTree
        - El paquete hashlib
        - El paquete sqlite3
        - El paquete wsgiref

    - Cambios en la API de construcción y C
        - Cambios específicos en los puertos
    - Adaptación a Python 2.5
    - Agradecimientos
- Qué hay de nuevo en Python 2.4
    - PEP 218: Objetos conjunto integrados
    - PEP 237: Unificando enteros largos y enteros
    - PEP 289: Expresiones generadoras
    - PEP 292: Sustituciones simples de cadenas de caracteres
    - PEP 318: Decoradores para funciones y métodos
    - PEP 322: Iteración inversa
    - PEP 324: Nuevo módulo de subproceso
    - PEP 327: Tipo de dato decimal
        - ¿Por qué se necesita Decimal?
        - El tipo de clase `Decimal`
        - El tipo de clase `Context`
    - PEP 328: Importaciones multilínea
    - PEP 331: Conversiones locales-independientes números flotantes/cadenas de texto
    - Otros cambios en el lenguaje
        - Optimizaciones
    - Módulos nuevos, mejorados y obsoletos
        - cookielib
        - doctest
    - Cambios en la API de Build y C
        - Cambios específicos del puerto
    - Portar a Python 2.4
    - Agradecimientos

- Qué hay de nuevo en Python 2.3
    - PEP 218: Un tipo de datos de conjunto estándar
    - PEP 255: Generadores simples
    - PEP 263: Codificación del código fuente
    - PEP 273: Importar módulos desde archivos ZIP
    - PEP 277: Soporte de nombres de archivo Unicode para Windows NT
    - PEP 278: Soporte universal de nuevas líneas
    - PEP 279: enumerate()
    - PEP 282: El paquete de registro
    - PEP 285: Un tipo booleano
    - PEP 293: Llamadas de retorno para el manejo de errores del códec
    - PEP 301: Índice de paquetes y metadatos para Distutils
    - PEP 302: Nuevos ganchos de importación
    - PEP 305: Archivos separados por comas
    - PEP 307: Mejoras en Pickle
    - Rebanadas ampliadas
    - Otros cambios en el lenguaje
        - Cambios en las cadenas de texto
        - Optimizaciones
    - Módulos nuevos, mejorados y obsoletos
        - Tipo de fecha / hora
        - El módulo optparse
    - Pymalloc: un asignador de objetos especializado
    - Cambios en la API de Build y C
        - Cambios específicos del puerto
    - Otros cambios y correcciones
    - Portar a Python 2.3
    - Agradecimientos
- Qué hay de nuevo en Python 2.2

    - Introducción
    - PEPs 252 y 253: Cambios de tipo y clase
        - Clases antiguas y nuevas
        - Descriptores
        - Herencia múltiple: la regla del diamante
        - Acceso a atributos
        - Enlaces relacionados
    - PEP 234: Iteradores
    - PEP 255: Generadores simples
    - PEP 237: Unificación de enteros largos y enteros
    - PEP 238: Cambio del operador de división
    - Cambios en Unicode
    - PEP 227: Ámbitos anidados
    - Módulos nuevos y mejorados
    - Cambios y correcciones en el intérprete
    - Otros cambios y correcciones
    - Agradecimientos
- Qué hay de nuevo en Python 2.1
    - Introducción
    - PEP 227: Ámbitos anidados
    - PEP 236: Directivas __future__
    - PEP 207: Comparaciones Enriquecidas
    - PEP 230: Marco de advertencia
    - PEP 229: Sistema de construcción nuevo
    - PEP 205: Referencias débiles
    - PEP 232: Atributos de la función
    - PEP 235: Importación de módulos en plataformas que no distinguen entre mayúsculas y minúsculas
    - PEP 217: Gancho de pantalla interactivo
    - PEP 208: Nuevo modelo de coerción
    - PEP 241: Metadatos en paquetes de Python
    - Módulos nuevos y mejorados
    - Otros cambios y correcciones
    - Agradecimientos
- Qué hay de nuevo en Python 2.0

    - Introducción
    - ¿Qué pasa con Python 1.6?
    - Nuevo proceso de desarrollo
    - Unicode
    - Comprensión de listas
    - Asignación aumentada
    - Métodos de cadena de caracteres
    - Recogida de basura de los ciclos
    - Otros cambios en el núcleo
        - Cambios menores del lenguaje
        - Cambios en las funciones incorporadas
    - Adaptación a la versión 2.0
    - Extensión/Incorporación de cambios
    - Distutils: Facilitando la instalación de módulos
    - Módulos XML
        - Soporte de SAX2
        - Soporte DOM
        - Relación con PyXML
    - Cambios en los módulos
    - Nuevos módulos
    - Mejoras en IDLE
    - Módulos eliminados y obsoletos
    - Agradecimientos
- Registro de cambios
    - Python next
        - Windows
        - Security
        - Library
        - Documentation
        - Core and Builtins
        - Build
    - Python 3.14.4 final

  - Security
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - macOS
  - C API

- Python 3.14.3 final

  - Windows
  - Tools/Demos
  - Tests
  - Security
  - Library
  - IDLE
  - Documentation
  - Core and Builtins
  - C API
  - Build

- Python 3.14.2 final

  - Security
  - Library
  - Core and Builtins
  - Library

- Python 3.14.1 final

  - Windows
  - Tools/Demos
  - Tests
  - Security
  - Library
  - IDLE
  - Documentation
  - Core and Builtins
  - Library

    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - C API
    - Build

- Python 3.14.0 final

    - macOS
    - Windows
    - Tools/Demos
    - Security
    - Library

- Python 3.14.0 release candidate 3

    - Windows
    - Tools/Demos
    - Security
    - Library
    - Core and Builtins

- Python 3.14.0 release candidate 2

    - macOS
    - Windows
    - Library
    - Documentation
    - Core and Builtins
    - C API
    - Build

- Python 3.14.0 release candidate 1

    - Tools/Demos
    - Security
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - C API
    - Build
- Python 3.14.0 beta 4
    - Tools/Demos
    - Tests
    - Security
    - Library
    - Documentation
    - Core and Builtins
    - C API
    - Build
- Python 3.14.0 beta 3
    - Windows
    - Tests
    - Security
    - Library
    - Documentation
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - C API
    - Build
- Python 3.14.0 beta 2

    - Windows
    - Tools/Demos
    - Tests
    - Security
    - Library
    - Core and Builtins
    - C API
    - Build
- Python 3.14.0 beta 1
    - Windows
    - Tools/Demos
    - Tests
    - Security
    - Library
    - IDLE
    - Documentation
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - C API
    - Build
- Python 3.14.0 alpha 7
    - macOS
    - Windows
    - Tools/Demos
    - Tests
    - Security
    - Library
    - Documentation
    - Core and Builtins
    - Library
    - Core and Builtins
    - C API
    - Build

- Python 3.14.0 alpha 6
    - macOS
    - Windows
    - Tools/Demos
    - Tests
    - Security
    - Library
    - Documentation
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - C API
    - Build
- Python 3.14.0 alpha 5
    - macOS
    - Tools/Demos
    - Tests
    - Security
    - Library
    - IDLE
    - Documentation
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - C API
    - Build
- Python 3.14.0 alpha 4
    - macOS
    - Tools/Demos
    - Tests
    - Library

    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - C API
    - Build
- Python 3.14.0 alpha 3
    - Windows
    - Tools/Demos
    - Tests
    - Security
    - Library
    - Documentation
    - Core and Builtins
    - Library
    - Core and Builtins
    - C API
    - Build
- Python 3.14.0 alpha 2
    - Windows
    - Tools/Demos
    - Tests
    - Security
    - Library
    - Documentation
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - C API
    - Build
- Python 3.14.0 alpha 1

    - macOS
    - Windows
    - Tools/Demos
    - Tests
    - Security
    - Library
    - IDLE
    - Documentation
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - C API
    - Build
- Python 3.13.0 beta 1
    - Security
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Documentation
    - Build
    - Windows
    - macOS
    - IDLE
    - C API

- Python 3.13.0 alpha 6
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - C API
- Python 3.13.0 alpha 5
    - Security
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - Tools/Demos
    - C API
- Python 3.13.0 alpha 4
    - Security
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE

  - Tools/Demos
  - C API
- Python 3.13.0 alpha 3
  - Security
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - macOS
  - IDLE
  - C API
- Python 3.13.0 alpha 2
  - Core and Builtins
  - Library
  - Tests
  - Build
  - Windows
  - macOS
  - IDLE
  - Tools/Demos
  - C API
- Python 3.13.0 alpha 1
  - Security
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - macOS
  - IDLE
  - Tools/Demos
  - C API

- Python 3.12.0 beta 1
    - Security
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - Tools/Demos
    - C API
- Python 3.12.0 alpha 7
    - Core and Builtins
    - Library
    - Core and Builtins
    - Build
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - Tools/Demos
    - C API
- Python 3.12.0 alpha 6
    - Security
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows

    - macOS
    - C API
- Python 3.12.0 alpha 5
    - Security
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
- Python 3.12.0 alpha 4
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - Tools/Demos
    - C API
- Python 3.12.0 alpha 3
    - Security
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS

    - Tools/Demos
    - C API

- Python 3.12.0 alpha 2

    - Security
    - Core and Builtins
    - Build
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - C API

- Python 3.12.0 alpha 1

    - Security
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Documentation

  - Tests
  - Build
  - Windows
  - macOS
  - IDLE
  - Tools/Demos
  - C API

- Python 3.11.0 beta 1

  - Security
  - Core and Builtins
  - Library
  - Core and Builtins
  - Build
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - macOS
  - Tools/Demos
  - C API

- Python 3.11.0 alpha 7

  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - macOS
  - Tools/Demos
  - C API

- Python 3.11.0 alpha 6
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - IDLE
  - C API
- Python 3.11.0 alpha 5
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Build
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - macOS
  - IDLE
  - C API
- Python 3.11.0 alpha 4
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins

    - Library
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - C API

- Python 3.11.0 alpha 3

    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - C API

- Python 3.11.0 alpha 2

    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - C API

- Python 3.11.0 alpha 1

    - Security
    - Core and Builtins
    - Library
    - Core and Builtins

  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - macOS
  - IDLE
  - Tools/Demos
  - C API
- Python 3.10.0 beta 1
  - Security
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - macOS

  - IDLE
  - C API
- Python 3.10.0 alpha 7
  - Security
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - IDLE
  - C API
- Python 3.10.0 alpha 6
  - Security
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - macOS
  - IDLE
  - C API
- Python 3.10.0 alpha 5
  - Security
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library

    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - C API

- Python 3.10.0 alpha 4

    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - macOS
    - Tools/Demos
    - C API

- Python 3.10.0 alpha 3

    - Security
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - Tools/Demos

  - C API

- Python 3.10.0 alpha 2

  - Security
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - macOS
  - IDLE
  - C API

- Python 3.10.0 alpha 1

  - Security
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins

  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - macOS
  - IDLE
  - C API

- Python 3.9.0 beta 1

  - Security
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - macOS
  - Tools/Demos
  - C API

- Python 3.9.0 alpha 6

  - Security
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - macOS
  - IDLE

    - Tools/Demos
    - C API
- Python 3.9.0 alpha 5
    - Security
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - Tools/Demos
    - C API
- Python 3.9.0 alpha 4
    - Security
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - IDLE
    - C API
- Python 3.9.0 alpha 3
    - Core and Builtins
    - Library
    - Documentation
    - Build
    - IDLE
    - C API
- Python 3.9.0 alpha 2
    - Security

    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - C API
- Python 3.9.0 alpha 1
    - Security
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - Tools/Demos
    - C API
- Python 3.8.0 beta 1
    - Security
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - Tools/Demos
    - C API
- Python 3.8.0 alpha 4
    - Security

    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - Tools/Demos
    - C API

- Python 3.8.0 alpha 3

    - Security
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - IDLE
    - Tools/Demos
    - C API

- Python 3.8.0 alpha 2

    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Windows
    - IDLE

- Python 3.8.0 alpha 1

    - Security
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build

    - Windows
    - macOS
    - IDLE
    - Tools/Demos
    - C API

- Python 3.7.0 final

    - Library
    - C API

- Python 3.7.0 release candidate 1

    - Core and Builtins
    - Library
    - Documentation
    - Build
    - Windows
    - IDLE

- Python 3.7.0 beta 5

    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - macOS
    - IDLE

- Python 3.7.0 beta 4

    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - Tools/Demos

- Python 3.7.0 beta 3
    - Security
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - Tools/Demos
    - C API
- Python 3.7.0 beta 2
    - Security
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - Tools/Demos
- Python 3.7.0 beta 1
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - C API
- Python 3.7.0 alpha 4
    - Core and Builtins

  - Library
  - Documentation
  - Tests
  - Windows
  - Tools/Demos
  - C API

- Python 3.7.0 alpha 3

  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - macOS
  - IDLE
  - Tools/Demos
  - C API

- Python 3.7.0 alpha 2

  - Core and Builtins
  - Library
  - Documentation
  - Build
  - IDLE
  - C API

- Python 3.7.0 alpha 1

  - Security
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
  - Windows
  - IDLE
  - Tools/Demos
  - C API

- Python 3.6.6 final
- Python 3.6.6 release candidate 1
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - Tools/Demos
    - C API
- Python 3.6.5 final
    - Tests
    - Build
- Python 3.6.5 release candidate 1
    - Security
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - macOS
    - IDLE
    - Tools/Demos
    - C API
- Python 3.6.4 final
- Python 3.6.4 release candidate 1
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build

    - Windows
    - macOS
    - IDLE
    - Tools/Demos
    - C API

- Python 3.6.3 final

    - Library
    - Build

- Python 3.6.3 release candidate 1

    - Security
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - IDLE
    - Tools/Demos

- Python 3.6.2 final
- Python 3.6.2 release candidate 2

    - Security

- Python 3.6.2 release candidate 1

    - Security
    - Core and Builtins
    - Library
    - IDLE
    - C API
    - Build
    - Documentation
    - Tools/Demos
    - Tests
    - Windows

- Python 3.6.1 final
  - Core and Builtins
  - Build
- Python 3.6.1 release candidate 1
  - Core and Builtins
  - Library
  - IDLE
  - Windows
  - C API
  - Documentation
  - Tests
  - Build
- Python 3.6.0 final
- Python 3.6.0 release candidate 2
  - Core and Builtins
  - Tools/Demos
  - Windows
  - Build
- Python 3.6.0 release candidate 1
  - Core and Builtins
  - Library
  - C API
  - Documentation
  - Tools/Demos
- Python 3.6.0 beta 4
  - Core and Builtins
  - Library
  - Documentation
  - Tests
  - Build
- Python 3.6.0 beta 3

  - Core and Builtins
  - Library
  - Windows
  - Build
  - Tests

- Python 3.6.0 beta 2

  - Core and Builtins
  - Library
  - Windows
  - C API
  - Build
  - Tests

- Python 3.6.0 beta 1

  - Core and Builtins
  - Library
  - IDLE
  - C API
  - Tests
  - Build
  - Tools/Demos
  - Windows

- Python 3.6.0 alpha 4

  - Core and Builtins
  - Library
  - IDLE
  - Tests
  - Windows
  - Build

- Python 3.6.0 alpha 3

  - Security
  - Core and Builtins
  - Library
  - IDLE

    - C API
    - Build
    - Tools/Demos
    - Documentation
    - Tests

- Python 3.6.0 alpha 2

    - Security
    - Core and Builtins
    - Library
    - IDLE
    - Documentation
    - Tests
    - Windows
    - Build
    - C API
    - Tools/Demos

- Python 3.6.0 alpha 1

    - Security
    - Core and Builtins
    - Library
    - IDLE
    - Documentation
    - Tests
    - Build
    - Windows
    - Tools/Demos
    - C API

- Python 3.5.5 final
- Python 3.5.5 release candidate 1

    - Security
    - Core and Builtins
    - Library

- Python 3.5.4 final

    - Library

- Python 3.5.4 release candidate 1

    - Security
    - Core and Builtins
    - Library
    - Documentation
    - Tests
    - Build
    - Windows
    - C API

- Python 3.5.3 final
- Python 3.5.3 release candidate 1

    - Security
    - Core and Builtins
    - Library
    - IDLE
    - C API
    - Documentation
    - Tests
    - Tools/Demos
    - Windows
    - Build

- Python 3.5.2 final

    - Core and Builtins
    - Tests
    - IDLE

- Python 3.5.2 release candidate 1

    - Security
    - Core and Builtins
    - Library
    - IDLE
    - Documentation
    - Tests

    - Build
    - Windows
    - Tools/Demos

- Python 3.5.1 final

    - Core and Builtins
    - Windows

- Python 3.5.1 release candidate 1

    - Core and Builtins
    - Library
    - IDLE
    - Documentation
    - Tests
    - Build
    - Windows
    - Tools/Demos

- Python 3.5.0 final

    - Build

- Python 3.5.0 release candidate 4

    - Library
    - Build

- Python 3.5.0 release candidate 3

    - Core and Builtins
    - Library

- Python 3.5.0 release candidate 2

    - Core and Builtins
    - Library

- Python 3.5.0 release candidate 1

    - Core and Builtins

  - Library
  - IDLE
  - Documentation
  - Tests

- Python 3.5.0 beta 4

  - Core and Builtins
  - Library
  - Build

- Python 3.5.0 beta 3

  - Core and Builtins
  - Library
  - Tests
  - Documentation
  - Build

- Python 3.5.0 beta 2

  - Core and Builtins
  - Library

- Python 3.5.0 beta 1

  - Core and Builtins
  - Library
  - IDLE
  - Tests
  - Documentation
  - Tools/Demos

- Python 3.5.0 alpha 4

  - Core and Builtins
  - Library
  - Build
  - Tests
  - Tools/Demos
  - C API

    - Python 3.5.0 alpha 3
        - Core and Builtins
        - Library
        - Build
        - Tests
        - Tools/Demos
    - Python 3.5.0 alpha 2
        - Core and Builtins
        - Library
        - Build
        - C API
        - Windows
    - Python 3.5.0 alpha 1
        - Core and Builtins
        - Library
        - IDLE
        - Build
        - C API
        - Documentation
        - Tests
        - Tools/Demos
        - Windows
- El tutorial de Python
  - 1. Abriendo el apetito
  - 2. Usando el intérprete de Python
    - 2.1. Invocar el intérprete
      - 2.1.1. Paso de argumentos
      - 2.1.2. Modo interactivo
    - 2.2. El intérprete y su entorno
      - 2.2.1. Codificación del código fuente

- 3. Una introducción informal a Python
    - 3.1. Usando Python como una calculadora
        - 3.1.1. Números
        - 3.1.2. Texto
        - 3.1.3. Listas
    - 3.2. Primeros pasos hacia la programación
- 4. Más herramientas para control de flujo
    - 4.1. La sentencia `if`
    - 4.2. La sentencia `for`
    - 4.3. La función `range()`
    - 4.4. Las sentencias `break` y `continue`
    - 4.5. Cláusulas `else` en bucles
    - 4.6. La sentencia `pass`
    - 4.7. La sentencia `match`
    - 4.8. Definir funciones
    - 4.9. Más sobre definición de funciones
        - 4.9.1. Argumentos con valores por omisión
        - 4.9.2. Palabras claves como argumentos
        - 4.9.3. Parámetros especiales
            - 4.9.3.1. Argumentos posicionales o de palabras claves
            - 4.9.3.2. Parámetros únicamente posicionales
            - 4.9.3.3. Argumentos únicamente de palabras clave
            - 4.9.3.4. Ejemplos de Funciones
            - 4.9.3.5. Resumen
        - 4.9.4. Listas de argumentos arbitrarios
        - 4.9.5. Desempaquetando una lista de argumentos
        - 4.9.6. Expresiones lambda
        - 4.9.7. Cadenas de texto de documentación
        - 4.9.8. Anotación de funciones

    - 4.10. Intermezzo: Estilo de programación
- 5. Estructuras de datos
    - 5.1. Más sobre listas
        - 5.1.1. Usar listas como pilas
        - 5.1.2. Usar listas como colas
        - 5.1.3. Comprensión de listas
        - 5.1.4. Listas por comprensión anidadas
    - 5.2. La instrucción `del`
    - 5.3. Tuplas y secuencias
    - 5.4. Conjuntos
    - 5.5. Diccionarios
    - 5.6. Técnicas de iteración
    - 5.7. Más acerca de condiciones
    - 5.8. Comparando secuencias y otros tipos
- 6. Módulos
    - 6.1. Más sobre los módulos
        - 6.1.1. Ejecutar módulos como scripts
        - 6.1.2. El camino de búsqueda de los módulos
        - 6.1.3. Archivos «compilados» de Python
    - 6.2. Módulos estándar
    - 6.3. La función `dir()`
    - 6.4. Paquetes
        - 6.4.1. Importar * desde un paquete
        - 6.4.2. Referencias internas en paquetes
        - 6.4.3. Paquetes en múltiples directorios
- 7. Entrada y salida
    - 7.1. Formateo elegante de la salida
        - 7.1.1. Formatear cadenas literales
        - 7.1.2. El método format() de cadenas

        - 7.1.3. Formateo manual de cadenas
        - 7.1.4. Viejo formateo de cadenas
    - 7.2. Leyendo y escribiendo archivos
        - 7.2.1. Métodos de los objetos Archivo
        - 7.2.2. Guardar datos estructurados con `json`
- 8. Errores y excepciones
    - 8.1. Errores de sintaxis
    - 8.2. Excepciones
    - 8.3. Gestionando excepciones
    - 8.4. Lanzando excepciones
    - 8.5. Encadenamiento de excepciones
    - 8.6. Excepciones definidas por el usuario
    - 8.7. Definiendo acciones de limpieza
    - 8.8. Acciones predefinidas de limpieza
    - 8.9. Lanzando y gestionando múltiples excepciones no relacionadas
    - 8.10. Enriqueciendo excepciones con notas
- 9. Clases
    - 9.1. Unas palabras sobre nombres y objetos
    - 9.2. Ámbitos y espacios de nombres en Python
        - 9.2.1. Ejemplo de ámbitos y espacios de nombre
    - 9.3. Un primer vistazo a las clases
        - 9.3.1. Sintaxis de definición de clases
        - 9.3.2. Objetos clase
        - 9.3.3. Objetos instancia
        - 9.3.4. Objetos método
        - 9.3.5. Variables de clase y de instancia
    - 9.4. Algunas observaciones
    - 9.5. Herencia

        - 9.5.1. Herencia múltiple
    - 9.6. Variables privadas
    - 9.7. Detalles y Cuestiones Varias
    - 9.8. Iteradores
    - 9.9. Generadores
    - 9.10. Expresiones generadoras
- 10. Brief tour of the standard library
    - 10.1. Operating system interface
    - 10.2. File wildcards
    - 10.3. Command-line arguments
    - 10.4. Error output redirection and program termination
    - 10.5. String pattern matching
    - 10.6. Matemáticas
    - 10.7. Internet access
    - 10.8. Dates and times
    - 10.9. Data compression
    - 10.10. Performance measurement
    - 10.11. Quality control
    - 10.12. Batteries included
- 11. Brief tour of the standard library — part II
    - 11.1. Output formatting
    - 11.2. Plantillas
    - 11.3. Working with binary data record layouts
    - 11.4. Multi-hilos
    - 11.5. Registrando
    - 11.6. Weak references
    - 11.7. Tools for working with lists
    - 11.8. Decimal floating-point arithmetic
- 12. Entornos virtuales y paquetes
    - 12.1. Introducción
    - 12.2. Creando entornos virtuales
    - 12.3. Manejando paquetes con pip

  - 13. ¿Y ahora qué?
  - 14. Edición de entrada interactiva y sustitución de historial
    - 14.1. Autocompletado con tab e historial de edición
    - 14.2. Alternativas al intérprete interactivo
  - 15. Floating-Point Arithmetic: Issues and Limitations
    - 15.1. Error de Representación
  - 16. Apéndice
    - 16.1. Modo interactivo
      - 16.1.1. Manejo de errores
      - 16.1.2. Programas ejecutables de Python
      - 16.1.3. El archivo de inicio interactivo
      - 16.1.4. Los módulos de customización
- Configuración y uso de Python
  - 1. Línea de comandos y entorno
    - 1.1. Línea de comando
      - 1.1.1. Opciones de interfaz
      - 1.1.2. Opciones genéricas
      - 1.1.3. Opciones diversas
      - 1.1.4. Controlling color
    - 1.2. Variables de entorno
      - 1.2.1. Variables de modo de depuración
  - 2. Uso de Python en plataformas Unix
    - 2.1. Obteniendo e instalando la última versión de Python
      - 2.1.1. En Linux

      - 2.1.1.1. Installing IDLE
    - 2.1.2. En FreeBSD y OpenBSD
  - 2.2. Construyendo Python
  - 2.3. Rutas y archivos relacionados con Python
  - 2.4. Miscelánea
  - 2.5. OpenSSL personalizado
- 3. Configurar Python
  - 3.1. Requisitos de compilación
    - 3.1.1. Requirements for optional modules
  - 3.2. Archivos generados
    - 3.2.1. configure script
  - 3.3. Configurar opciones
    - 3.3.1. Opciones generales
    - 3.3.2. C compiler options
    - 3.3.3. Opciones del enlazador
    - 3.3.4. Options for third-party dependencies
    - 3.3.5. Opciones de WebAssembly
    - 3.3.6. Opciones de instalación
    - 3.3.7. Opciones de desempeño
    - 3.3.8. Compilación de depuración de Python
    - 3.3.9. Opciones de depuración
    - 3.3.10. Opciones del enlazador
    - 3.3.11. Opciones de bibliotecas
    - 3.3.12. Opciones de seguridad
    - 3.3.13. Opciones macOS
    - 3.3.14. iOS Options
    - 3.3.15. Opciones de compilación cruzada
  - 3.4. Sistema de compilación Python
    - 3.4.1. Archivos principales del sistema de compilación

    - 3.4.2. Pasos principales de compilación
    - 3.4.3. Objetivos principales de Makefile
      - 3.4.3.1. make
      - 3.4.3.2. make platform
      - 3.4.3.3. make profile-opt
      - 3.4.3.4. make clean
      - 3.4.3.5. make distclean
      - 3.4.3.6. make install
      - 3.4.3.7. make test
      - 3.4.3.8. make ci
      - 3.4.3.9. make buildbottest
      - 3.4.3.10. make regen-all
    - 3.4.4. Extensiones C
  - 3.5. Banderas de compilador y vinculación
    - 3.5.1. Banderas del preprocesador
    - 3.5.2. Banderas del compilador
    - 3.5.3. Banderas de vinculación
- 4. Uso de Python en Windows
  - 4.1. Python install manager
    - 4.1.1. Installation
    - 4.1.2. Basic use
    - 4.1.3. Command help
    - 4.1.4. Listing runtimes
    - 4.1.5. Installing runtimes
    - 4.1.6. Offline installs
    - 4.1.7. Uninstalling runtimes
    - 4.1.8. Configuration
    - 4.1.9. Shebang lines
    - 4.1.10. Advanced installation
    - 4.1.11. Administrative configuration
    - 4.1.12. Installing free-threaded binaries
    - 4.1.13. Index signatures
    - 4.1.14. Troubleshooting

- 4.2. El paquete incrustable
    - 4.2.1. Python application
    - 4.2.2. Incrustar Python
- 4.3. El paquete de nuget.org
    - 4.3.1. Free-threaded packages
- 4.4. Distribuciones alternativas
- 4.5. Supported Windows versions
- 4.6. Removing the MAX_PATH limitation
- 4.7. Modo UTF-8
- 4.8. Encontrar módulos
- 4.9. Módulos adicionales
    - 4.9.1. PyWin32
    - 4.9.2. cx_Freeze
- 4.10. Compilar Python en Windows
- 4.11. The full installer (deprecated)
    - 4.11.1. Pasos para la instalación
    - 4.11.2. Removing the MAX_PATH limitation
    - 4.11.3. Installing without UI
    - 4.11.4. Installing without downloading
    - 4.11.5. Modificar una instalación
    - 4.11.6. Installing free-threaded binaries
- 4.12. Python launcher for Windows (deprecated)
    - 4.12.1. Comenzar
        - 4.12.1.1. Desde la línea de comandos
        - 4.12.1.2. Entornos virtuales
        - 4.12.1.3. Desde un script
        - 4.12.1.4. Desde asociaciones de archivos
    - 4.12.2. Shebang lines
    - 4.12.3. Argumentos en líneas shebang

    - 4.12.4. Personalización
        - 4.12.4.1. Personalización con archivos INI
        - 4.12.4.2. Personalizar las versiones de Python predeterminadas
    - 4.12.5. Diagnóstico
    - 4.12.6. Dry run
    - 4.12.7. Instalación bajo demanda
    - 4.12.8. Códigos de retorno
- 5. Using Python on macOS
  - 5.1. Using Python for macOS from **`python.org`**
    - 5.1.1. Installation steps
    - 5.1.2. Cómo ejecutar un *script* de Python
  - 5.2. Alternative Distributions
  - 5.3. Instalación de paquetes adicionales de Python
  - 5.4. GUI Programming
  - 5.5. Advanced Topics
    - 5.5.1. Installing Free-threaded Binaries
    - 5.5.2. Installing using the command line
    - 5.5.3. Distributing Python Applications
    - 5.5.4. App Store Compliance
  - 5.6. Otros recursos
- 6. Using Python on Android
  - 6.1. Adding Python to an Android app
  - 6.2. Building a Python package for Android
- 7. Using Python on iOS
  - 7.1. Python at runtime on iOS
    - 7.1.1. iOS version compatibility

      - 7.1.2. Platform identification
      - 7.1.3. Standard library availability
      - 7.1.4. Binary extension modules
      - 7.1.5. Compiler stub binaries
    - 7.2. Installing Python on iOS
      - 7.2.1. Tools for building iOS apps
      - 7.2.2. Adding Python to an iOS project
      - 7.2.3. Testing a Python package
    - 7.3. App Store Compliance
      - 7.3.1. Incompatible code in the standard library
      - 7.3.2. Privacy manifests
  - 8. Editores e IDEs
    - 8.1. IDLE — Python editor and shell
    - 8.2. Other Editors and IDEs
- Referencia del Lenguaje Python
  - 1. Introducción
    - 1.1. Implementaciones alternativas
    - 1.2. Notación
      - 1.2.1. Lexical and Syntactic definitions
  - 2. Análisis léxico
    - 2.1. Estructura de línea
      - 2.1.1. Líneas lógicas
      - 2.1.2. Líneas físicas
      - 2.1.3. Comentarios
      - 2.1.4. Declaración de Codificación
      - 2.1.5. Unión explícita de líneas
      - 2.1.6. Unión implícita de líneas
      - 2.1.7. Líneas en blanco

    - 2.1.8. Sangría
    - 2.1.9. Espacios en blanco entre tokens
    - 2.1.10. End marker

- 2.2. Otros tokens
- 2.3. Names (identifiers and keywords)

    - 2.3.1. Palabras clave
    - 2.3.2. Palabras clave suaves
    - 2.3.3. Clases reservadas de identificadores
    - 2.3.4. Non-ASCII characters in names

- 2.4. Literales
- 2.5. Literales de cadenas y bytes

    - 2.5.1. Triple-quoted strings
    - 2.5.2. String prefixes
    - 2.5.3. Formal grammar
    - 2.5.4. Secuencias de escape

        - 2.5.4.1. Ignored end of line
        - 2.5.4.2. Escaped characters
        - 2.5.4.3. Octal character
        - 2.5.4.4. Hexadecimal character
        - 2.5.4.5. Named Unicode character
        - 2.5.4.6. Hexadecimal Unicode characters
        - 2.5.4.7. Unrecognized escape sequences

    - 2.5.5. Bytes literals
    - 2.5.6. Raw string literals
    - 2.5.7. f-strings
    - 2.5.8. t-strings
    - 2.5.9. Formal grammar for f-strings

- 2.6. Literales numéricos

    - 2.6.1. Literales enteros
    - 2.6.2. Literales de punto flotante
    - 2.6.3. Literales imaginarios

        - 2.7. Operators and delimiters
    - 3. Modelo de datos
        - 3.1. Objetos, valores y tipos
        - 3.2. Jerarquía de tipos estándar
            - 3.2.1. None
            - 3.2.2. NotImplemented
            - 3.2.3. Elipsis
            - 3.2.4. **`numbers.Number`**
                - 3.2.4.1. **`numbers.Integral`**
                - 3.2.4.2. **`numbers.Real`** (**`float`**)
                - 3.2.4.3. **`numbers.Complex`** (**`complex`**)
            - 3.2.5. Secuencias
                - 3.2.5.1. Secuencias inmutables
                - 3.2.5.2. Secuencias mutables
            - 3.2.6. Tipos de conjuntos
            - 3.2.7. Mapeos
                - 3.2.7.1. Diccionarios
            - 3.2.8. Tipos invocables
                - 3.2.8.1. Funciones definidas por el usuario
                    - 3.2.8.1.1. Atributos especiales de solo lectura
                    - 3.2.8.1.2. Atributos especiales de escritura
                - 3.2.8.2. Métodos de instancia
                - 3.2.8.3. Funciones generadoras
                - 3.2.8.4. Funciones de corrutina
                - 3.2.8.5. Funciones generadoras asincrónicas

        - 3.2.8.6. Funciones incorporadas
        - 3.2.8.7. Métodos incorporados
        - 3.2.8.8. Clases
        - 3.2.8.9. Instancias de clases
    - 3.2.9. Módulos
        - 3.2.9.1. Import-related attributes on module objects
        - 3.2.9.2. Other writable attributes on module objects
        - 3.2.9.3. Module dictionaries
    - 3.2.10. Clases personalizadas
        - 3.2.10.1. Atributos especiales
        - 3.2.10.2. Métodos especiales
    - 3.2.11. Instancias de clase
        - 3.2.11.1. Atributos especiales
    - 3.2.12. Objetos E/S (también conocidos como objetos de archivo)
    - 3.2.13. Tipos internos
        - 3.2.13.1. Objetos de código
            - 3.2.13.1.1. Atributos especiales de solo lectura
            - 3.2.13.1.2. Métodos en objetos de código
        - 3.2.13.2. Objetos de marco
            - 3.2.13.2.1. Atributos especiales de solo lectura
            - 3.2.13.2.2. Atributos especiales de escritura
            - 3.2.13.2.3. Métodos de objeto de marco

    - 3.2.13.3. Objetos de seguimiento de pila (traceback)
    - 3.2.13.4. Objetos de segmento (Slice objects)
    - 3.2.13.5. Objetos de método estático
    - 3.2.13.6. Objetos de método de clase

- 3.3. Nombres especiales de método

  - 3.3.1. Personalización básica
  - 3.3.2. Personalizando acceso a atributos

    - 3.3.2.1. Personalizando acceso a atributos de módulo
    - 3.3.2.2. Implementando descriptores
    - 3.3.2.3. Invocando descriptores
    - 3.3.2.4. __slots__

  - 3.3.3. Personalización de creación de clases

    - 3.3.3.1. Metaclases
    - 3.3.3.2. Resolviendo entradas de la Orden de Resolución de Métodos (MRU)
    - 3.3.3.3. Determinando la metaclase adecuada
    - 3.3.3.4. Preparando el espacio de nombres de la clase
    - 3.3.3.5. Ejecutando el cuerpo de la clase
    - 3.3.3.6. Creando el objeto de clase
    - 3.3.3.7. Usos para metaclases

  - 3.3.4. Personalizando revisiones de instancia y subclase
  - 3.3.5. Emulando tipos genéricos

    - 3.3.5.1. El propósito de *__class_getitem__*
    - 3.3.5.2. *__class_getitem__* frente a *__getitem__*

        - 3.3.6. Emulando objetos que se pueden llamar
        - 3.3.7. Emulando tipos de contenedores
        - 3.3.8. Emulando tipos numéricos
        - 3.3.9. Gestores de Contexto en la Declaración *with*
        - 3.3.10. Personalización de argumentos posicionales en la coincidencia de patrones de clase
        - 3.3.11. Emulando tipos de búfer
        - 3.3.12. Annotations
        - 3.3.13. Búsqueda de método especial

    - 3.4. Corrutinas

        - 3.4.1. Objetos esperables
        - 3.4.2. Objetos de corrutina
        - 3.4.3. Iteradores asíncronos
        - 3.4.4. Gestores de contexto asíncronos

- 4. Modelo de ejecución

    - 4.1. Estructura de un programa
    - 4.2. Nombres y vínculos

        - 4.2.1. Vinculación de nombres
        - 4.2.2. Resolución de nombres
        - 4.2.3. Ámbitos de anotación
        - 4.2.4. Evaluación perezosa
        - 4.2.5. Integraciones y ejecución restringida
        - 4.2.6. Interacción con funcionalidades dinámicas

    - 4.3. Excepciones
    - 4.4. Runtime Components

        - 4.4.1. General Computing Model
        - 4.4.2. Python Runtime Model

- 5. El sistema de importación

    - 5.1. **`importlib`**
    - 5.2. Paquetes
        - 5.2.1. Paquetes regulares
        - 5.2.2. Paquetes de espacio de nombres
    - 5.3. Buscando
        - 5.3.1. La caché del módulo
        - 5.3.2. Buscadores y cargadores
        - 5.3.3. Ganchos de importación
        - 5.3.4. La meta ruta (*path*)
    - 5.4. Cargando
        - 5.4.1. Cargadores
        - 5.4.2. Submódulos
        - 5.4.3. Module specs
        - 5.4.4. __path__ attributes on modules
        - 5.4.5. Representación (*Reprs*) de módulos
        - 5.4.6. Invalidación del código de bytes en caché
    - 5.5. El buscador basado en rutas
        - 5.5.1. Buscadores de entradas de ruta
        - 5.5.2. Buscadores de entradas de ruta
    - 5.6. Reemplazando el sistema de importación estándar
    - 5.7. Paquete Importaciones relativas
    - 5.8. Consideraciones especiales para __main__
        - 5.8.1. __main__.__spec__
    - 5.9. Referencias
- ○ 6. Expresiones
    - 6.1. Conversiones aritméticas
    - 6.2. Átomos

    - 6.2.1. Built-in constants
    - 6.2.2. Identificadores (Nombres)
        - 6.2.2.1. Alteración de nombre privado
    - 6.2.3. Literales
        - 6.2.3.1. Literals and object identity
        - 6.2.3.2. String literal concatenation
    - 6.2.4. Formas entre paréntesis
    - 6.2.5. Despliegues para listas, conjuntos y diccionarios
    - 6.2.6. Despliegues de lista
    - 6.2.7. Despliegues de conjuntos
    - 6.2.8. Despliegues de diccionario
    - 6.2.9. Expresiones de generador
    - 6.2.10. Expresiones *yield*
        - 6.2.10.1. Métodos generador-iterador
        - 6.2.10.2. Ejemplos
        - 6.2.10.3. Funciones generadoras asincrónicas
        - 6.2.10.4. Métodos asincrónicos de generador-iterador
- 6.3. Primarios
    - 6.3.1. Referencias de atributos
    - 6.3.2. Subscriptions and slicings
        - 6.3.2.1. Segmentos
        - 6.3.2.2. Comma-separated subscripts
        - 6.3.2.3. «Starred» subscriptions
        - 6.3.2.4. Formal subscription grammar
    - 6.3.3. Invocaciones
- 6.4. Expresión await
- 6.5. El operador de potencia

    - 6.6. Aritmética unaria y operaciones bit a bit
    - 6.7. Operaciones aritméticas binarias
    - 6.8. Operaciones de desplazamiento
    - 6.9. Operaciones bit a bit binarias
    - 6.10. Comparaciones
        - 6.10.1. Comparaciones de valor
        - 6.10.2. Operaciones de prueba de membresía
        - 6.10.3. Comparaciones de identidad
    - 6.11. Operaciones booleanas
    - 6.12. Expresiones de asignación
    - 6.13. Expresiones condicionales
    - 6.14. Lambdas
    - 6.15. Listas de expresiones
    - 6.16. Orden de evaluación
    - 6.17. Prioridad de operador
- 7. Declaraciones simples
    - 7.1. Declaraciones de tipo expresión
    - 7.2. Declaraciones de asignación
        - 7.2.1. Declaraciones de asignación aumentada
        - 7.2.2. Declaraciones de asignación anotadas
    - 7.3. La declaración **`assert`**
    - 7.4. La declaración **`pass`**
    - 7.5. La declaración **`del`**
    - 7.6. La declaración **`return`**
    - 7.7. La declaración **`yield`**
    - 7.8. La declaración **`raise`**
    - 7.9. La declaración **`break`**
    - 7.10. La declaración **`continue`**
    - 7.11. La declaración **`import`**
        - 7.11.1. Declaraciones Futuras

    - 7.12. La declaración `global`
    - 7.13. La declaración `nonlocal`
    - 7.14. The `type` statement

- 8. Sentencias compuestas

    - 8.1. La sentencia `if`
    - 8.2. La sentencia `while`
    - 8.3. La sentencia `for`
    - 8.4. La sentencia `try`

        - 8.4.1. Cláusula `except`
        - 8.4.2. Cláusula `except*`
        - 8.4.3. La sentencia `else`
        - 8.4.4. Cláusula `finally`

    - 8.5. La sentencia `with`
    - 8.6. La sentencia `match`

        - 8.6.1. Resumen
        - 8.6.2. Protecciones
        - 8.6.3. Bloques de Casos Irrefutables
        - 8.6.4. Patrones

            - 8.6.4.1. Patrones OR
            - 8.6.4.2. patrones AS
            - 8.6.4.3. Patrones literales
            - 8.6.4.4. Patrones de captura
            - 8.6.4.5. Patrones comodín
            - 8.6.4.6. Patrones de valor
            - 8.6.4.7. Patrones de grupo
            - 8.6.4.8. Patrones de secuencia
            - 8.6.4.9. Patrones de mapeo
            - 8.6.4.10. Patrones de clase

    - 8.7. Definiciones de funciones
    - 8.8. Definiciones de clase
    - 8.9. Corrutinas

        - 8.9.1. Definición de la función corrutina
        - 8.9.2. La sentencia `async for`

            - 8.9.3. La sentencia `async with`
        - 8.10. Listas de tipo parámetro
            - 8.10.1. Funciones genéricas
            - 8.10.2. Clases genéricas
            - 8.10.3. Alias de tipo genérico
        - 8.11. Annotations
    - 9. Componentes de nivel superior
        - 9.1. Programas completos de Python
        - 9.2. Entrada de archivo
        - 9.3. Entrada interactiva
        - 9.4. Entrada de expresión
    - 10. Especificación completa de la gramática
- La biblioteca estándar de Python
    - Introducción
        - Notas sobre la disponibilidad
            - Plataformas WebAssembly
            - Mobile platforms
    - Funciones incorporadas
    - Constantes incorporadas
        - Constantes agregadas por el módulo `site`
    - Tipos integrados
        - Evaluar como valor verdadero/falso
        - Operaciones booleanas — `and`, `or`, `not`
        - Comparaciones
        - Tipos numéricos — `int`, `float`, `complex`
            - Operaciones de bits en números enteros
            - Métodos adicionales de los enteros

    - Métodos adicionales de float
    - Additional Methods on Complex
    - Calculo del *hash* de tipos numéricos
- Tipo Booleano — `bool`
- Tipos de iteradores
    - Tipos generador
- Tipos secuencia — `list`, `tuple`, `range`
    - Operaciones comunes de las secuencias
    - Tipos de secuencia inmutables
    - Tipos de secuencia mutables
    - Listas
    - Tuplas
    - Rangos
- Text and Binary Sequence Type Methods Summary
- Cadenas de caracteres — `str`
    - Métodos de las cadenas de caracteres
    - Formatted String Literals (f-strings)
        - Debug specifier
        - Conversion specifier
        - Format specifier
    - Template String Literals (t-strings)
    - Formateo de cadenas al estilo `*printf*`
- Tipos de secuencias binarias — `bytes`, `bytearray` y `memoryview`
    - Objetos de tipo Bytes
    - Objetos de tipo *Bytearray*
    - Operaciones de *bytes* y *bytearray*
    - Usando el formateo tipo `printf` con bytes
    - Vistas de memoria
- Conjuntos — `set`, `frozenset`

  - Tipos mapa — **`dict`**
    - Objetos tipos vista de diccionario
  - Tipos gestores de contexto
  - Tipos de anotaciones de type — alias genérico, Union
    - Tipo Alias Genérico
      - Clases genéricas estándar
      - Atributos especiales de los objetos **`GenericAlias`**
    - Tipo de conversión
  - Otros tipos predefinidos
    - Módulos
    - Clases e instancias de clase
    - Funciones
    - Métodos
    - Objetos código
    - Objetos Tipo
    - El objeto nulo (*Null*)
    - El objeto puntos suspensivos (*Ellipsis*)
    - El objeto *NotImplemented*
    - Objetos internos
  - Atributos especiales
  - Limitación de longitud de conversión de cadena de tipo entero
    - APIs afectadas
    - Configuración del límite
    - Configuración recomendada
- Excepciones incorporadas
  - Contexto de una excepción
  - Heredando de excepciones incorporadas
  - Clases base

        - Excepciones específicas
            - Excepciones del sistema operativo
        - Advertencias
        - Grupos de excepciones
        - Jerarquía de excepción
    - Thread Safety Guarantees
        - Thread safety levels
            - Incompatible
            - Compatible
            - Safe on distinct objects
            - Safe on shared objects
            - Atomic
        - Thread safety for list objects
        - Thread safety for dict objects
        - Thread safety for set objects
        - Thread safety for bytearray objects
        - Thread safety for memoryview objects
    - Servicios de procesamiento de texto
        - `string` — Common string operations
            - Constantes de cadenas
            - Custom string formatting
            - Format string syntax
                - Format specification mini-language
                - Ejemplos de formateo
            - Template strings ($-strings)
            - Funciones de ayuda
        - `string.templatelib` — Support for template string literals
            - Template strings

    - Types
    - Helper functions

- **`re`** — Regular expression operations

    - Sintaxis de expresiones regulares
    - Contenidos del módulo

        - Indicadores
        - Funciones
        - Excepciones

    - Objetos expresión regular
    - Objetos de coincidencia
    - Ejemplos de expresiones regulares

        - Buscando un par
        - Simular scanf()
        - search() vs. match()
        - Haciendo una guía telefónica
        - Mungear texto
        - Encontrar todos los adverbios
        - Encontrar todos los adverbios y sus posiciones
        - Notación de cadena *raw*
        - Escribir un Tokenizador

- **`difflib`** — Helpers for computing deltas

    - SequenceMatcher objects
    - SequenceMatcher examples
    - Differ objects
    - Differ example
    - Una interfaz de línea de comandos para **`difflib`**
    - ndiff example

- **`textwrap`** — Text wrapping and filling
- **`unicodedata`** — Unicode Database
- **`stringprep`** — Internet String Preparation
- **`readline`** — GNU readline interface

        - Archivo de inicio
        - Búfer de línea
        - Archivo de historial
        - Lista del historial
        - Ganchos (*hooks*) de inicialización
        - Terminación
        - Ejemplo
    - **`rlcompleter`** — Completion function for GNU readline
- Servicios de datos binarios
    - **`struct`** — Interpret bytes as packed binary data
        - Funciones y excepciones
        - Cadenas de formato
            - Orden de bytes, tamaño y alineación
            - Caracteres de formato
            - Ejemplos
        - Aplicaciones
            - Formatos nativos
            - Formatos estándar
        - Clases
    - **`codecs`** — Codec registry and base classes
        - Clases Base de Códec
            - Manejadores de errores
            - Codificación y decodificación sin estado
            - Codificación y decodificación incrementales
                - Objetos IncrementalEncoder
                - Objetos IncrementalDecoder

      - Codificación y decodificación de flujos
        - Objetos StreamWriter
        - Objetos StreamReader
        - Objetos StreamReaderWriter
        - Objetos StreamRecoder
    - Codificaciones y Unicode
    - Codificaciones estándar
    - Codificaciones específicas de Python
      - Codificaciones de texto
      - Transformaciones Binarias
      - Standalone Codec Functions
      - Transformaciones de texto
    - `encodings` — Encodings package
    - `encodings.idna` — Internationalized Domain Names in Applications
    - `encodings.mbcs` — Windows ANSI codepage
    - `encodings.utf_8_sig` — UTF-8 codec with BOM signature
- Tipos de datos
  - `datetime` — Basic date and time types
    - Aware and naive objects
    - Constantes
    - Available types
      - Common properties
      - Determining if an object is aware or naive
    - `timedelta` objects
      - Examples of usage: `timedelta`
    - `date` objects

    - Examples of usage: `date`
  - `datetime` objects
    - Examples of usage: `datetime`
  - `time` objects
    - Examples of usage: `time`
  - `tzinfo` objects
  - `timezone` objects
  - `strftime()` and `strptime()` behavior
    - `strftime()` and `strptime()` format codes
    - Technical detail
- `zoneinfo` — IANA time zone support
  - Usando `ZoneInfo`
  - Fuentes de datos
    - Configurando los orígenes de datos
      - Configuración en tiempo de compilación
      - Configuración del entorno
      - Configuración de tiempo de ejecución
  - La clase `ZoneInfo`
    - Representaciones de cadenas
    - Serialización de Pickle
  - Funciones
  - Globales
  - Excepciones y advertencias
- `calendar` — General calendar-related functions

    - Command-line usage
- `collections` — Container datatypes
    - Objetos `ChainMap`
        - Ejemplos y recetas `ChainMap`
    - Objetos `Counter`
    - Objetos `deque`
        - Recetas `deque`
    - Objetos `defaultdict`
        - Ejemplos `defaultdict`
    - `namedtuple()` Funciones *Factory* para Tuplas y Campos con Nombres
    - Objetos `OrderedDict`
        - Ejemplos y recetas `OrderedDict`
    - Objetos `UserDict`
    - Objetos `UserList`
    - Objetos `UserString`
- `collections.abc` — Abstract Base Classes for Containers
    - Colecciones clases base abstractas
    - Colecciones Clases base abstractas - Descripciones detalladas
    - Ejemplos y Recetas
- `heapq` — Heap queue algorithm
    - Ejemplos Básicos
    - Other Applications
    - Notas de Aplicación de la Cola de Prioridades
    - Teoría

- **`bisect`** — Array bisection algorithm
  - Notas de rendimiento
  - Búsqueda en listas ordenadas
  - Ejemplos
- **`array`** — Efficient arrays of numeric values
- **`weakref`** — Weak references
  - Objetos de referencias débiles
  - Ejemplo
  - Objetos finalizadores
  - Comparando finalizadores con los métodos **`__del__()`**
- **`types`** — Dynamic type creation and names for built-in types
  - Creación dinámica de tipos
  - Tipos de Intérpretes Estándar
  - Clases y funciones de utilidad adicionales
  - Funciones de utilidad de corutina
- **`copy`** — Shallow and deep copy operations
- **`pprint`** — Data pretty printer
  - Functions
  - Objetos *PrettyPrinter*
  - Ejemplo
- **`reprlib`** — Alternate **`repr()`** implementation
  - Objetos Repr
  - Subclasificando Objetos Repr
- **`enum`** — Support for enumerations
  - Module contents
  - Data types
    - Nombres soportados **`__dunder__`**
    - Nombres **`_sunder_`** compatibles

        - Utilities and decorators
        - Notas
    - **`graphlib`** — Functionality to operate with graph-like structures
        - Excepciones
- Módulos numéricos y matemáticos
    - **`numbers`** — Numeric abstract base classes
        - La torre numérica
        - Notes for type implementers
            - Agregar más *ABCs* numéricos
            - Implementar operaciones aritméticas
    - **`math`** — Mathematical functions
        - Number-theoretic functions
        - Floating point arithmetic
        - Floating point manipulation functions
        - Power, exponential and logarithmic functions
        - Summation and product functions
        - Conversión angular
        - Funciones trigonométricas
        - Funciones hiperbólicas
        - Funciones especiales
        - Constantes
    - **`cmath`** — Mathematical functions for complex numbers
        - Conversión a y desde coordenadas polares
        - Funciones logarítmicas y de potencias
        - Funciones trigonométricas
        - Funciones hiperbólicas
        - Funciones de clasificación
        - Constantes

- **`decimal`** — Decimal fixed-point and floating-point arithmetic
  - Quick-start tutorial
  - Objetos Decimal
    - Operandos lógicos
  - Objetos context
  - Constantes
  - Modos de redondeo
  - Señales
  - Floating-point notes
    - Mitigación del error de redondeo usando mayor precisión
    - Valores especiales
  - Trabajando con hilos
  - Casos prácticos
  - Preguntas frecuentes sobre decimal
- **`fractions`** — Rational numbers
- **`random`** — Generate pseudo-random numbers
  - Funciones de contabilidad
  - Funciones para los bytes
  - Funciones para enteros
  - Funciones para secuencias
  - Discrete distributions
  - Distribuciones para los nombres reales
  - Generador alternativo
  - Notas sobre la Reproducibilidad
  - Ejemplos
  - Recetas
  - Command-line usage
  - Command-line example
- **`statistics`** — Mathematical statistics functions
  - Promedios y medidas de tendencia central

    - Medidas de dispersión
    - Estadísticas para relaciones entre dos entradas
    - Detalles de las funciones
    - Excepciones
    - Objetos **`NormalDist`**
    - Examples and Recipes
      - Problemas de probabilidad clásicos
      - Entradas de Monte Carlo para simulaciones
      - Aproximación de la distribución binomial
      - Clasificador bayesiano ingenuo
- Módulos de programación funcional
  - **`itertools`** — Functions creating iterators for efficient looping
    - Itertool Functions
    - Fórmulas con itertools
  - **`functools`** — Higher-order functions and operations on callable objects
    - **`partial`** Objetos
  - **`operator`** — Standard operators as functions
    - Asignación de operadores a funciones
    - Operadores *In-place*
- Acceso a archivos y directorios
  - **`pathlib`** — Rutas de sistemas orientada a objetos
    - Uso básico
    - Excepciones
    - Rutas puras

        - Propiedades generales
        - Operadores
        - Acceso a partes individuales
        - Métodos y propiedades
    - Rutas concretas
        - Análisis y generación de URI
        - Ampliando y resolviendo caminos
        - Consulta de tipo de archivo y estado
        - Lectura y escritura de archivos
        - Directorios de lectura
        - Creando archivos y directorios
        - Copying, moving and deleting
        - Permisos y propiedad
    - Lenguaje de patrones
    - Comparación con el módulo `glob`
    - Comparación con los módulos `os` y `os.path`
        - Herramientas correspondientes
    - Protocols
- `os.path` — Common pathname manipulations
- `stat` — Interpreting `stat()` results
- `filecmp` — File and Directory Comparisons
    - La clase `dircmp`
- `tempfile` — Generate temporary files and directories
    - Ejemplos
    - Funciones y variables deprecadas
- `glob` — Unix style pathname pattern expansion
    - Examples
- `fnmatch` — Unix filename pattern matching

  - `linecache` — Random access to text lines
  - `shutil` — High-level file operations
    - Operaciones de directorios y archivos
      - Operaciones de copia eficientes dependientes de la plataforma
      - ejemplo de *copytree*
      - ejemplo de rmtree
    - Operaciones de archivado
      - Ejemplo de archivado
      - Ejemplo de archivado con *base_dir*
    - Consulta el tamaño de la terminal de salida
- Persistencia de datos
  - `pickle` — Python object serialization
    - Relación con otros módulos de Python
      - Comparación con `marshal`
      - Comparación con `json`
    - Formato de flujo de datos
    - Interfaz del módulo
    - ¿Qué se puede serializar (pickled) y deserializar (unpickled) con *pickle*?
    - *Pickling* de Instancias de clases
      - Persistencia de objetos externos
      - Tablas de despacho
      - Manejo de objetos con estado
    - Reducción personalizada para tipos, funciones y otros objetos
    - Búferes fuera de banda
      - API de proveedor
      - API de consumidor

        - Ejemplo

    - Restricción de globals
    - Performance
    - Ejemplos
    - Command-line interface

- **`copyreg`** — Register **`pickle`** support functions

    - Ejemplo

- **`shelve`** — Python object persistence

    - Restricciones
    - Ejemplo

- **`marshal`** — Internal Python object serialization
- **`dbm`** — Interfaces to Unix «databases»

    - **`dbm.sqlite3`** — SQLite backend for dbm
    - **`dbm.gnu`** — GNU database manager
    - **`dbm.ndbm`** — New Database Manager
    - **`dbm.dumb`** — Portable DBM implementation

- **`sqlite3`** — DB-API 2.0 interfaz para bases de datos SQLite

    - Tutorial
    - Referencia

        - Funciones del módulo
        - Constantes del módulo
        - Objetos de conexión
        - Objetos cursor
        - Objetos fila (*Row*)
        - Objetos fila (*Row*)
        - Objetos PrepareProtocol
        - Excepciones
        - SQLite y tipos de Python
        - Adaptadores y convertidores predeterminados (en desuso)

        - Interfaz de línea de comandos
    - Guías prácticas
        - Cómo usar marcadores de posición para vincular valores en consultas SQL
        - Cómo adaptar tipos de Python personalizados a valores de SQLite
            - Cómo escribir objetos adaptables
            - Como registrar un adaptador invocable
        - Como convertir valores SQLite a tipos de Python personalizados
        - Ejemplos para adaptadores y convertidores
        - Cómo utilizar los métodos de acceso directo de conexión
        - Como usar la conexión con un administrador de contexto
        - Como trabajar con URIs SQLite
        - Cómo crear y utilizar fábricas de filas
        - Cómo manejar codificaciones de texto que no sean UTF-8
    - Explicación
        - Control transaccional
            - Control de transacciones mediante el atributo **`autocommit`**
            - Control de transacciones mediante el atributo **`isolation_level`**
- ○ Compresión de datos y archivado

- The `compression` package
- `compression.zstd` — Compression compatible with the Zstandard format
    - Exceptions
    - Reading and writing compressed files
    - Compressing and decompressing data in memory
    - Zstandard dictionaries
    - Advanced parameter control
    - Miscellaneous
    - Examples
- `zlib` — Compression compatible with **gzip**
- `gzip` — Support for **gzip** files
    - Ejemplos de uso
    - Command-line interface
        - Command-line options
- `bz2` — Support for **bzip2** compression
    - (De)compresión de archivos
    - (De)compresión incremental
    - (Des)comprimir en un solo paso
    - Ejemplos de uso
- `lzma` — Compression using the LZMA algorithm
    - Leyendo y escribiendo ficheros comprimidos
    - Comprimiendo y descomprimiendo datos en memoria
    - Misceláneas
    - Especificando cadenas de filtro personalizadas
    - Ejemplos
- `zipfile` — Work with ZIP archives
    - ZipFile objects

        - Path objects
        - PyZipFile objects
        - ZipInfo objects
        - Command-line interface
            - Opciones de línea de comando
        - Problemas de descompresión
            - Del archivo mismo
            - File system limitations
            - Limitaciones de recursos
            - Interrupción
            - Comportamientos predeterminados de extracción
    - `tarfile` — Read and write tar archive files
        - Objetos *TarFile*
        - Objetos TarInfo
        - Extraction filters
            - Default named filters
            - Filter errors
            - Hints for further verification
            - Supporting older Python versions
            - Stateful extraction filter example
        - Interfaz de línea de comandos
            - Opciones de línea de comandos
        - Ejemplos
            - Reading examples
            - Writing examples
        - Formatos tar con soporte
        - Problemas unicode
- Formatos de archivo

  - **`csv`** — CSV File Reading and Writing
    - Contenidos del módulo
    - Dialectos y parámetros de formato
    - Objetos *Reader*
    - Objetos *Writer*
    - Ejemplos
  - **`configparser`** — Configuration file parser
    - Inicio Rápido
    - Tipos de Datos Soportados
    - Valores de contingencia
    - Estructura soportada para el archivo ini
    - Unnamed Sections
    - Interpolación de valores
    - Acceso por protocolo de mapeo
    - Personalizando el comportamiento del parser
    - Ejemplos de la API heredada
    - Objetos ConfigParser
    - Objetos RawConfigParser
    - Excepciones
  - **`tomllib`** — Parse TOML files
    - Ejemplos
    - Tabla de conversión
  - **`netrc`** — netrc file processing
    - Objetos netrc
  - **`plistlib`** — Generate and parse Apple **`.plist`** files
    - Ejemplos
- Servicios criptográficos
  - **`hashlib`** — Secure hashes and message digests

    - Algoritmos de hash
    - Usage
    - Constructors
    - Attributes
    - Hash Objects
    - Resúmenes SHAKE de largo variable
    - Cifrado de archivos
    - Derivación de clave
    - BLAKE2
      - Creando objetos hash
      - Constantes
      - Ejemplos
        - Cifrado simple
        - Usar diferentes tamaños de resumen
        - Cifrado de clave
        - Cifrado aleatorio
        - Personalización
        - Modo de árbol
      - Créditos
  - **`hmac`** — Keyed-Hashing for Message Authentication
  - **`secrets`** — Generate secure random numbers for managing secrets
    - Números aleatorios
    - Generando tokens
      - ¿Cuántos bytes deben tener los tokens?
    - Otras funciones
    - Recetas y mejores prácticas
- Servicios genéricos del sistema operativo
  - **`os`** — Interfaces misceláneas del sistema

operativo

- Nombres de archivos, argumentos de la línea de comandos y variables de entorno
- Modo Python UTF-8
- Parámetros de proceso
- Creación de objetos de tipo archivo
- Operaciones de descriptores de archivos
    - Consultando las dimensiones de una terminal
    - Herencia de los descriptores de archivos
- Archivos y directorios
    - Descriptores de archivos del temporizador
    - Atributos extendidos de Linux
- Gestión de proceso
- Interfaz al planificador
- Información miscelánea del sistema
- Números al azar

- `io` — Core tools for working with streams
    - Resumen
        - E/S Texto
        - E/S Binaria
        - E/S sin formato
    - Codificación de texto
        - EncodingWarning opcional
    - Interfaz de módulo de alto nivel
    - Jerarquía de clases
        - Clases base E/S
        - Archivo sin formato E/S

        - *Streams* almacenados (búfer)
        - E/S Texto
    - Static Typing
    - Rendimiento
        - E/S Binaria
        - E/S Texto
        - Multihilo
        - Reentrada
- **`time`** — Time access and conversions
    - Las Funciones
    - Constantes de ID de reloj
    - Constantes de zona horaria
- **`logging`** — Logging facility for Python
    - Objetos logger
    - Niveles de logging
    - Gestor de objetos
    - Objetos formateadores
    - Filtro de Objetos
    - Objetos LogRecord
    - Atributos LogRecord
    - Objetos LoggerAdapter
    - Seguridad del hilo
    - Funciones a nivel de módulo
    - Atributos a nivel de módulo
    - Integración con el módulo de advertencias
- **`logging.config`** — Logging configuration
    - Funciones de configuración
    - Consideraciones de seguridad
    - Esquema del diccionario de configuración
        - Detalles del esquema del diccionario
        - Configuración incremental
        - Conexiones de objeto

        - Objetos definidos por el usuario
        - Handler configuration order
        - Acceso a objetos externos
        - Acceso a objetos internos
        - Resolución de importación e importadores personalizados
        - Configuring QueueHandler and QueueListener

    - Formato de archivo de configuración

- **`logging.handlers`** — Logging handlers

    - StreamHandler
    - FileHandler
    - NullHandler
    - WatchedFileHandler
    - BaseRotatingHandler
    - RotatingFileHandler
    - TimedRotatingFileHandler
    - SocketHandler
    - DatagramHandler
    - Gestor *SysLog* (*SysLogHandler*)
    - Gestor de eventos *NTELog* (NTEventLogHandler)
    - SMTPHandler
    - MemoryHandler
    - HTTPHandler
    - QueueHandler
    - QueueListener

- **`platform`** — Access to underlying platform's identifying data

    - Cross platform
    - Java platform
    - Windows platform
    - macOS platform
    - iOS platform
    - Unix platforms
    - Linux platforms

    - Android platform
    - Command-line usage
- **`errno`** — Standard errno system symbols
- **`ctypes`** — A foreign function library for Python
    - tutorial de ctypes
        - Carga de bibliotecas de enlaces dinámicos
        - Acceder a las funciones de los dll cargados
        - Funciones de llamada
        - Tipos de datos fundamentales
        - Funciones de llamada, continuación
        - Calling variadic functions
        - Funciones de llamada con sus propios tipos de datos personalizados
        - Especificar los tipos de argumentos requeridos (prototipos de funciones)
        - Tipos de retorno
        - Pasar los punteros (o: pasar los parámetros por referencia)
        - Estructuras y uniones
        - Structure/union layout, alignment and byte order
        - Campos de bits en estructuras y uniones
        - Arreglos
        - Punteros
        - Thread safety without the GIL
        - Conversiones de tipos
        - Tipos incompletos
        - Funciones de retrollamadas (*callback*)
        - Acceder a los valores exportados de los dlls
        - Sorpresas
        - Tipos de datos de tamaño variable
    - referencia ctypes

        - Encontrar bibliotecas compartidas
        - Listing loaded shared libraries
        - Cargando bibliotecas compartidas
        - Funciones foráneas
        - Prototipos de funciones
        - Funciones de utilidad
        - Tipos de datos
        - Tipos de datos fundamentales
        - Tipos de datos estructurados
        - Arreglos y punteros
        - Exceptions
- Command-line interface libraries
    - **`argparse`** — Parser for command-line options, arguments and subcommands
        - Objetos *ArgumentParser*
            - *prog*
            - uso
            - *description*
            - *epilog*
            - *parents*
            - *formatter_class*
            - *prefix_chars*
            - *fromfile_prefix_chars*
            - *argument_default*
            - *allow_abbrev*
            - *conflict_handler*
            - *add_help*
            - exit_on_error
            - suggest_on_error
            - color
        - El método *add_argument()*
            - *name or flags*
            - *action*
            - *nargs*
            - *const*

  - *default*
  - *type*
  - *choices*
  - *required*
  - *help*
  - *metavar*
  - *dest*
  - deprecated
  - Las clases *Action*

- El método *parse_args()*

  - Sintaxis del valor de la opción
  - Argumentos no válidos
  - Argumentos conteniendo `-`
  - Abreviaturas de los argumentos (coincidencia de prefijos)
  - Más allá de **`sys.argv`**
  - El objeto *Namespace*

- Otras utilidades

  - Subcommands
  - Objetos *FileType*
  - Grupos de argumentos
  - Exclusión mutua
  - Valores por defecto del analizador
  - Mostrando la ayuda
  - Análisis parcial
  - Personalizando el análisis de archivos
  - Métodos de salida
  - Análisis entremezclado
  - Registering custom types or actions

- Excepciones

  - Tutorial de *argparse*

    - Conceptos
    - Las bases

      - Introducción a los argumentos posicionales
      - Introducción a los argumentos opcionales

        - Opciones cortas

      - Combinar argumentos opcionales y posicionales
      - Un poco mas avanzado

        - Especificando argumentos ambiguos
        - Opciones conflictivas

      - Cómo traducir la salida de argparse
      - Custom type converters
      - Conclusión

    - Migrating **`optparse`** code to **`argparse`**

- **`optparse`** — Parser for command line options

  - Choosing an argument parsing library
  - Introduction
  - Contexto

    - Terminología
    - ¿Qué finalidad tienen las opciones?
    - ¿Qué finalidad tienen los argumentos posicionales?

  - Tutorial

    - Comprendiendo las acciones de opción
    - La acción store
    - Manejo de opciones booleanas (flags)
    - Otras acciones
    - Valores por defecto

    - Generando ayuda
        - Agrupando opciones
    - Imprimir una cadena de caracteres con la versión del programa
    - How `optparse` handles errors
    - Reuniendo todas las piezas
- Guía de referencia
    - Creando el analizador sintáctico (parser)
    - Completando el analizador con opciones
    - Definiendo las opciones
    - Atributos de opción
    - Acciones de opción estándares
    - Tipos de opción estándares
    - Analizando los argumentos
    - Consultar y manipular el analizador de opciones
    - Conflictos entre opciones
    - Limpieza
    - Otros métodos
- Retrollamadas de opción
    - Definición de una opción con retrollamada
    - Cómo son invocadas las retrollamadas
    - Lanzando errores en una retrollamada
    - Ejemplo de retrollamada 1: una retrollamada trivial
    - Ejemplo de retrollamada 2: comprobar el orden de las opciones
    - Ejemplo de retrollamada 3: comprobar el orden de las opciones (generalizado)

        - Ejemplo de retrollamada 4: comprobar una condición arbitraria
        - Ejemplo de retrollamada 5: argumentos fijos
        - Ejemplo de retrollamada 6: argumentos variables
    - Extending `optparse`
        - Agregando nuevos tipos
        - Agregando nuevas acciones
    - Excepciones
- `getpass` — Portable password input
- `fileinput` — Iterate over lines from multiple input streams
- `curses` — Terminal handling for character-cell displays
    - Funciones
    - Objetos de ventana
    - Constantes
- `curses.textpad` — Text input widget for curses programs
    - Objeto de caja de texto
- `curses.ascii` — Utilities for ASCII characters
- `curses.panel` — A panel stack extension for curses
    - Funciones
    - Objetos de panel
- `cmd` — Support for line-oriented command interpreters
    - Objetos Cmd
    - Ejemplo Cmd

- Ejecución concurrente
  - `threading` — Thread-based parallelism
    - Introduction
    - GIL and performance considerations
    - Reference
      - Thread-local data
      - Thread objects
      - Lock objects
      - RLock objects
      - Condition objects
      - Semaphore objects
      - `Semaphore` example
      - Event objects
      - Timer objects
      - Barrier objects
    - Uso de *locks*, condiciones y semáforos en la declaración `with`
  - `multiprocessing` — Process-based parallelism
    - Introducción
      - La clase `Process`
      - Contextos y métodos de inicio
      - Intercambiando objetos entre procesos
      - Sincronización entre procesos
      - Compartiendo estado entre procesos
      - Usando una piscina de trabajadores (*pool of workers*)
    - Referencia
      - Global start method
      - `Process` y excepciones
      - Tuberías (*Pipes*) y Colas (*Queues*)
      - Miscelánea

    - Objetos de conexión *Connection Objects*
    - Primitivas de sincronización (*Synchronization primitives*)
    - Objetos compartidos `ctypes`
      - The `multiprocessing.sharedctypes` module
    - Administradores (*Managers*)
      - Administradores customizables (*Customized managers*)
      - Utilizando un administrador remoto
    - Objetos Proxy (*Proxy Objects*)
      - Limpieza (*Cleanup*)
    - Piscinas de procesos (*Process Pools*)
    - Oyentes y clientes (*Listeners and Clients*)
      - Formatos de dirección (*Address formats*)
    - Llaves de autentificación
    - *Logging*
    - The `multiprocessing.dummy` module
  - Pautas de programación
    - Todos los métodos de inicio
    - Los métodos de inicio *spawn* y *forkserver*
  - Ejemplos
- `multiprocessing.shared_memory` — Shared

memory for direct access across processes
- El paquete `concurrent`
- `concurrent.futures` — Launching parallel tasks
    - Objetos ejecutores
    - ThreadPoolExecutor
        - Ejemplo de ThreadPoolExecutor
    - InterpreterPoolExecutor
    - ProcessPoolExecutor
        - Ejemplo de *ProcessPoolExecutor*
    - Objetos futuro
    - Funciones del módulo
    - Clases de Excepciones
- `concurrent.interpreters` — Multiple interpreters in the same process
    - Key details
    - Introduction
        - Multiple Interpreters and Isolation
        - Running in an Interpreter
        - Concurrency and Parallelism
        - Communication Between Interpreters
        - «Sharing» Objects
    - Reference
        - Interpreter objects
        - Exceptions
        - Communicating Between Interpreters
    - Basic usage
- `subprocess` — Subprocess management
    - Using the `subprocess` Module

            - Argumentos frecuentemente empleados
            - El constructor de Popen
            - Excepciones

        - Consideraciones sobre seguridad
        - Objetos Popen
        - Elementos auxiliares de Popen en Windows

            - Constantes de Windows

        - Antigua API de alto nivel
        - Replacing Older Functions with the **`subprocess`** Module

            - Cómo reemplazar la sustitución de órdenes de **/bin/sh**
            - Cómo remplazar los flujos de la shell
            - Cómo reemplazar **`os.system()`**
            - Cómo reemplazar la familia **`os.spawn`**
            - Replacing **`os.popen()`**

        - Funciones de llamada a la shell de retrocompatibilidad
        - Notas

            - Timeout Behavior
            - Cómo convertir una secuencia de argumentos a una cadena en Windows
            - Disable use of **`posix_spawn()`**

    - **`sched`** — Event scheduler

        - Objetos de **`Scheduler`**

    - **`queue`** — A synchronized queue class

        - Objetos de la cola

            - Waiting for task completion

      - Terminating queues
    - Objetos de cola simple
  - **`contextvars`** — Context Variables
    - variables de contexto
    - Gestión de contexto manual
    - Soporte asyncio
  - **`_thread`** — Low-level threading API
- Comunicación en redes y entre procesos
  - **`asyncio`** — Asynchronous I/O
    - Runners
      - Ejecutando un programa asyncio
      - Gestor de contexto del runner
      - Manejando interrupciones de teclado
    - Coroutines and tasks
      - Corrutinas
      - Esperables
      - Creating tasks
      - Task cancellation
      - Task groups
        - Terminating a task group
      - Durmiendo
      - Running tasks concurrently
      - Eager task factory
      - Shielding from cancellation
      - Tiempo agotado
      - Waiting primitives
      - Running in threads
      - Scheduling from other threads
      - Introspección
      - Task object

- Streams
    - StreamReader
    - StreamWriter
    - Ejemplos
        - Cliente eco TCP usando *streams*
        - Servidor eco TCP usando *streams*
        - Obtener encabezados HTTP
        - Registrar un socket abierto para esperar datos usando *streams*
- Primitivas de sincronización
    - Lock
    - Event
    - Condition
    - Semaphore
    - BoundedSemaphore
    - Barrera
- Sub-procesos
    - Creando sub-procesos
    - Constantes
    - Interactuando con Subprocesos
        - Subprocesos y Hilos
        - Ejemplos
- Colas
    - Cola
    - Cola de prioridad
    - Cola UEPA (o *LIFO* en inglés)
    - Excepciones
    - Ejemplos
- Excepciones
- Call Graph Introspection

        - Low level utility functions

    - Event loop

        - Event loop methods

            - Iniciar y para el bucle
            - Programación de llamadas de retorno
            - Planificando llamadas retardadas
            - Creating futures and tasks
            - Abriendo conexiones de red
            - Creando servidores de red
            - Transfiriendo archivos
            - TLS upgrade
            - Viendo descriptores de archivos
            - Trabajar con objetos sockets directamente
            - DNS
            - Trabajando con tuberías
            - Señales Unix
            - Ejecutando código en un hilos o grupos de procesos
            - Error handling API
            - Habilitando el modo depuración
            - Running subprocesses

        - Callback handles
        - Server objects
        - Event loop implementations
        - Examples

            - Hola Mundo con call_soon()
            - Muestra la fecha actual con call_later()
            - Mirar un descriptor de archivo para leer eventos
            - Establece los gestores de señal

        para SIGINT y SIGTERM

- Futures

    - Funciones Future
    - Objeto Future

- Transportes y protocolos

    - Transportes

        - Jerarquía de transportes
        - Transporte base
        - Transportes de solo lectura
        - Transportes de solo escritura
        - Transportes de datagramas
        - Transportes de subprocesos

    - Protocolos

        - Protocolos base
        - Protocolo base
        - Protocolos de *streaming*
        - Protocolos de *streaming* mediante búfer
        - Protocolos de datagramas
        - Protocolos de subprocesos

    - Ejemplos

        - Servidor de eco TCP
        - Cliente de eco TCP
        - Servidor de eco UDP
        - Cliente de eco UDP
        - Conectando sockets existentes
        - *loop.subprocess_exec()* y *SubprocessProtocol*

- Políticas

    - Obteniendo y configurando la política

    - Objetos de política
    - Personalizar Políticas

- Soporte de plataforma

    - Todas las plataformas
    - Windows

        - Soporte de sub-procesos en Windows

    - macOS

- Extensión

    - Escribir un bucle de eventos personalizado
    - Constructores privados Future y Task
    - Soporte de por vida de tareas

- Índice de API de alto nivel

    - Tareas
    - Colas
    - Subprocesos
    - Flujos
    - Sincronización
    - Excepciones

- Índice de API de bajo nivel

    - Obtención del bucle de eventos
    - Métodos del bucle de eventos
    - Transportes
    - Protocolos
    - Políticas de bucle de eventos

- Desarrollando con asyncio

    - Modo depuración
    - Concurrencia y multihilo
    - Ejecutando código bloqueante

    - Logueando
    - Detectar corrutinas no esperadas
    - Detectar excepciones nunca recuperadas
    - Asynchronous generators best practices
      - Close asynchronous generators explicitly
      - Create asynchronous generators only when the event loop is running
      - Avoid concurrent iteration and closure of the same generator
- **`socket`** — Low-level networking interface
  - Familias Socket
  - Contenido del módulo
    - Excepciones
    - Constantes
    - Funciones
      - Creación de sockets
      - Otras funciones
  - Objetos Socket
  - Notas sobre los tiempos de espera del socket
    - Tiempos de espera y el método **`connect`**
    - Tiempos de espera y el método **`accept`**
  - Ejemplo
- **`ssl`** — TLS/SSL wrapper for socket objects
  - Functions, constants, and exceptions

        - Creación de sockets
        - Creación de contexto
        - Excepciones
        - Generación aleatoria
        - Gestión de certificados
        - Constantes

    - SSL sockets
    - SSL contexts
    - Certificados

        - Cadenas de certificados
        - Certificados CA
        - Clave y certificado combinados
        - Certificados auto-firmados

    - Ejemplos

        - Pruebas de compatibilidad con SSL
        - Operación del lado del cliente
        - Operación del lado del servidor

    - Notas sobre los sockets no bloqueantes
    - Memory BIO support
    - Sesión SSL
    - Consideraciones de seguridad

        - Los mejores valores por defecto
        - Ajustes manuales

            - Verificación de certificados
            - Versiones del protocolo
            - Selección de cifrado

        - Multiprocesamiento

    - TLS 1.3

- **`select`** — Waiting for I/O completion

    - Objetos de sondeo **`/dev/poll`**
    - Objetos de sondeo de *Edge* y *Level Trigger*

(*epoll*)
    - Sondeo de objetos
    - Objetos Kqueue
    - Objetos Kevent
  - **`selectors`** — High-level I/O multiplexing
    - Introducción
    - Clases
    - Ejemplos
  - **`signal`** — Set handlers for asynchronous events
    - Reglas generales
      - Ejecución de los gestores de señales de Python
      - Señales e hilos
    - Contenidos del módulo
    - Ejemplos
    - Nota sobre SIGPIPE
    - Nota sobre Manejadores de Señales y Excepciones
  - **`mmap`** — Memory-mapped file support
    - Constantes MADV_*
    - Constantes MAP_*
- Manejo de datos de internet
  - **`email`** — An email and MIME handling package
    - **`email.message`**: Representing an email message
    - **`email.parser`**: Parsing email messages
      - API *FeedParser*
      - API *Parser*
      - Notas adicionales

    - **`email.generator`**: Generating MIME documents
    - **`email.policy`**: Policy Objects
    - **`email.errors`**: Exception and Defect classes
    - **`email.headerregistry`**: Custom Header Objects
    - **`email.contentmanager`**: Managing MIME Content
        - Instancias gestoras de contenido
    - **`email`**: Ejemplos
    - **`email.message.Message`**: Representar un mensaje de correo electrónico usando la API **`compat32`**
    - **`email.mime`**: Creating email and MIME objects from scratch
    - **`email.header`**: Internationalized headers
    - **`email.charset`**: Representing character sets
    - **`email.encoders`**: Encoders
    - **`email.utils`**: Miscellaneous utilities
    - **`email.iterators`**: Iterators
- **`json`** — JSON encoder and decoder
    - Uso básico
    - Codificadores y Decodificadores
    - Excepciones
    - Cumplimiento e interoperabilidad estándar
        - Codificaciones de caracteres
        - Valores de número infinito y NaN
        - Nombres repetidos dentro de un objeto
        - Valores de nivel superior No-Objeto , No-Arreglo
        - Limitaciones de la implementación
    - Command-line interface

        - Command-line options
- `mailbox` — Manipulate mailboxes in various formats
    - `Mailbox` objects
        - `Maildir` objects
        - `mbox` objects
        - `MH` objects
        - `Babyl` objects
        - `MMDF` objects
    - `Message` objects
        - `MaildirMessage` objects
        - `mboxMessage` objects
        - `MHMessage` objects
        - `BabylMessage` objects
        - `MMDFMessage` objects
    - Excepciones
    - Ejemplos
- `mimetypes` — Map filenames to MIME types
    - MimeTypes objects
    - Command-line usage
    - Command-line example
- `base64` — Base16, Base32, Base64, Base85 Data Encodings
    - RFC 4648 Encodings
    - Base85 Encodings
    - Legacy Interface
    - Consideraciones de Seguridad
- `binascii` — Convert between binary and ASCII
- `quopri` — Encode and decode MIME quoted-printable data

- Herramientas Para Procesar Formatos de Marcado Estructurado
  - **`html`** — HyperText Markup Language support
  - **`html.parser`** — Simple HTML and XHTML parser
    - Aplicación ejemplo de un analizador sintáctico (*parser*) de HTML
    - Métodos **`HTMLParser`**
    - Ejemplos
  - **`html.entities`** — Definitions of HTML general entities
  - Módulos de procesamiento XML
    - XML security
  - **`xml.etree.ElementTree`** — The ElementTree XML API
    - Tutorial
      - Árbol y elementos XML
      - Procesando XML
      - API de consulta para un procesamiento no bloqueante
      - Encontrando elementos interesantes
      - Modificando un archivo XML
      - Construyendo documentos XML
      - Procesando XML con espacio de nombres
    - Soporte de XPath
      - Ejemplo
      - Sintaxis XPath soportada
    - Referencia
      - Funciones

    - Soporte de XInclude
        - Ejemplo
    - Referencia
        - Funciones
        - Objetos Element
        - Objetos ElementTree
        - Objetos QName
        - Objetos TreeBuilder
        - Objetos XMLParser
        - Objetos XMLPullParser
        - Excepciones
- **`xml.dom`** — The Document Object Model API
    - Contenido del módulo
    - Objetos en el DOM
        - Objetos *DOMImplementation*
        - Objetos nodo
        - Objetos *NodeList*
        - Objetos *DocumentType*
        - Objetos documento
        - Objetos elemento
        - Objetos atributo
        - Objetos *NamedNodeMap*
        - Objetos comentario
        - Objetos Texto y *CDATASection*
        - Objetos *ProcessingInstruction*
        - Excepciones
    - Conformidad
        - Mapeo de tipos
        - Métodos de acceso (*accessor*)
- **`xml.dom.minidom`** — Minimal DOM implementation
    - Objetos del DOM

    - Ejemplo de DOM
    - minidom y el estándar DOM
  - **`xml.dom.pulldom`** — Support for building partial DOM trees
    - Objetos DOMEventStream
  - **`xml.sax`** — Support for SAX2 parsers
    - Objetos SAXException
  - **`xml.sax.handler`** — Base classes for SAX handlers
    - Objetos ContentHandler
    - Objetos DTDHandler
    - Objetos EntityResolver
    - Objetos ErrorHandler
    - Objetos DTDHandler
  - **`xml.sax.saxutils`** — SAX Utilities
  - **`xml.sax.xmlreader`** — Interface for XML parsers
    - Objetos XMLReader
    - Objetos IncrementalParser
    - Objetos localizadores
    - Objetos InputSource
    - La Interfaz **`Attributes`**
    - La Interfaz **`AttributesNS`**
  - **`xml.parsers.expat`** — Fast XML parsing using Expat
    - Objetos XMLParser
    - Excepciones de ExpatError
    - Ejemplo
    - Descripciones del modelo de contenido
    - Constantes de error de expansión
- Protocolos y soporte de Internet

- **`webbrowser`** — Convenient web-browser controller
    - Command-line interface
    - Browser controller objects
- **`wsgiref`** — WSGI Utilities and Reference Implementation
    - **`wsgiref.util`** – WSGI environment utilities
    - **`wsgiref.headers`** – WSGI response header tools
    - **`wsgiref.simple_server`** – a simple WSGI HTTP server
    - **`wsgiref.validate`** — WSGI conformance checker
    - **`wsgiref.handlers`** – server/gateway base classes
    - **`wsgiref.types`** – WSGI types for static type checking
    - Ejemplos
- **`urllib`** — URL handling modules
- **`urllib.request`** — Extensible library for opening URLs
    - Objetos Request
    - Objetos OpenerDirector
    - Objetos BaseHandler
    - Objetos HTTPRedirectHandler
    - Objetos HTTPCookieProcessor
    - Objetos ProxyHandler
    - Objetos HTTPPasswordMgr
    - Objetos HTTPPasswordMgrWithPriorAuth
    - Objetos AbstractBasicAuthHandler
    - Objetos HTTPBasicAuthHandler
    - Objetos ProxyBasicAuthHandler
    - Objetos AbstractDigestAuthHandler
    - Objetos HTTPDigestAuthHandler
    - Objetos ProxyDigestAuthHandler

    - Objetos HTTPHandler
    - Objetos HTTPSHandler
    - Objetos FileHandler
    - Objetos DataHandler
    - Objetos FTPHandler
    - Objetos CacheFTPHandler
    - Objetos UnknownHandler
    - Objetos HTTPErrorProcessor
    - Ejemplos
    - Interfaz heredada
    - **`urllib.request`** Restrictions

- **`urllib.response`** — Response classes used by urllib
- **`urllib.parse`** — Parse URLs into components

    - Análisis de URL
    - Análisis de seguridad de URL
    - Análisis de bytes codificados ASCII
    - Resultados del análisis estructurado
    - Cita de URL

- **`urllib.error`** — Exception classes raised by urllib.request
- **`urllib.robotparser`** — Parser for robots.txt
- **`http`** — HTTP modules

    - Códigos de estado HTTP
    - Categoría de estado HTTP
    - Métodos HTTP

- **`http.client`** — HTTP protocol client

    - Objetos de **`HTTPConnection`**
    - Objetos de **`HTTPResponse`**
    - Ejemplos
    - Objetos de **`HTTPMessage`**

- **`ftplib`** — FTP protocol client

    - Reference

        - FTP objects
        - FTP_TLS objects
        - Module variables
- **`poplib`** — POP3 protocol client
    - Objetos POP3
    - Ejemplo POP3
- **`imaplib`** — IMAP4 protocol client
    - Objetos de IMAP4
    - Ejemplo IMAP4
- **`smtplib`** — SMTP protocol client
    - Objetos SMTP
    - Ejemplo SMTP
- **`uuid`** — UUID objects according to **RFC 9562**
    - Uso de la línea de comandos
    - Ejemplo
    - Ejemplo de línea de comandos
- **`socketserver`** — A framework for network servers
    - Notas de creación del servidor
    - Objetos de servidor
    - Solicitar objetos de controlador
    - Ejemplos
        - **`socketserver.TCPServer`** Ejemplo
        - **`socketserver.UDPServer`** Ejemplo
        - Mixins asincrónicos
- **`http.server`** — HTTP servers
    - Command-line interface

    - Security considerations
- **`http.cookies`** — HTTP state management
    - Objetos de cookie
    - Objetos Morsel
    - Ejemplo
- **`http.cookiejar`** — Cookie handling for HTTP clients
    - Objetos CookieJar y FileCookieJar
    - Subclases FileCookieJar y co-operación con navegadores web
    - Objetos CookiePolicy
    - Objetos DefaultCookiePolicy
    - Objetos Cookie
    - Ejemplos
- **`xmlrpc`** — Módulos XMLRPC para cliente y servidor
- **`xmlrpc.client`** — XML-RPC client access
    - Objetos *ServerProxy*
    - Objetos *DateTime*
    - Objetos binarios
    - Objetos Faults
    - Objetos ProtocolError
    - Objetos MultiCall
    - Funciones de Conveniencia
    - Ejemplo de uso de cliente
    - Ejemplo de uso de cliente y servidor
- **`xmlrpc.server`** — Basic XML-RPC servers
    - SimpleXMLRPCServer objects
        - SimpleXMLRPCServer example
    - CGIXMLRPCRequestHandler
    - Documentando el servidor XMLRPC
    - DocXMLRPCServer objects

        - DocCGIXMLRPCRequestHandler
    - `ipaddress` — IPv4/IPv6 manipulation library
        - Funciones de fábrica de conveniencia
        - Direcciones IP
            - Objetos de dirección
            - Conversión a cadenas y enteros
            - Operadores
                - Operadores de comparación
                - Operadores aritméticos
        - Definiciones de red IP
            - Prefijo, máscara de red y máscara de host
            - Objetos de red
            - Operadores
                - Operadores lógicos
                - Iteración
                - Redes como contenedores de direcciones
        - Objetos de interfaz
            - Operadores
                - Operadores lógicos
        - Otras funciones de nivel de módulo
        - Excepciones personalizadas
- Servicios Multimedia
    - `wave` — Read and write WAV files
        - Los objetos *Wave_read*
        - Los objetos *Wave_write*

        - **`colorsys`** — Conversions between color systems
    - Internacionalización
        - **`gettext`** — Multilingual internationalization services
            - GNU API **gettext**
            - API basada en clases
                - La clase **`NullTranslations`**
                - La clase **`GNUTranslations`**
                - Soporte de catálogo de mensajes de Solaris
                - El constructor del catálogo
            - Internacionalizando sus programas y módulos
                - Agregar configuración regional a su módulo
                - Agregar configuración regional a su aplicación
                - Cambiar idiomas sobre la marcha
                - Traducciones diferidas
            - Agradecimientos
        - **`locale`** — Internationalization services
            - Segundo plano, detalles, indicaciones, consejos y advertencias
            - Locale names
            - Para escritores de extensión y programas que incrustan Python
            - Acceso a los catálogos de mensajes
    - Graphical user interfaces with Tk
        - **`tkinter`** — Python interface to Tcl/Tk
            - Arquitectura

    - Módulos Tkinter
    - Guía de supervivencia de Tkinter
        - Un programa simple de Hola Mundo
        - Conceptos importantes de Tk
        - Entendiendo como funcionan los empaquetadores de Tcl/Tk
        - ¿Cómo lo hago?, ¿Cómo funciona?
        - Navegando en el manual de referencia de Tcl/Tk
    - Modelo de subprocesamiento
    - Guía práctica
        - Configuración de opciones
        - El empaquetador
        - Opciones del empaquetador
        - Asociación de variables de widget
        - El gestor de ventanas
        - Tipos de datos de opciones Tk
        - Enlaces y eventos
        - El parámetro índice
        - Imágenes
    - Gestor de archivos
- **`tkinter.colorchooser`** — Color choosing dialog
- **`tkinter.font`** — Tkinter font wrapper
- Diálogos tkinter
    - **`tkinter.simpledialog`** — Standard Tkinter input dialogs
    - **`tkinter.filedialog`** — File selection dialogs
        - Diálogos nativos de carga/guardado
    - **`tkinter.commondialog`** — Dialog window templates

- **`tkinter.messagebox`** — Tkinter message prompts
- **`tkinter.scrolledtext`** — Scrolled Text Widget
- **`tkinter.dnd`** — Drag and drop support
- **`tkinter.ttk`** — Tk themed widgets
    - Uso de Ttk
    - Ttk widgets
    - Widget
        - Opciones estándar
        - Opciones de widgets desplegables
        - Opciones de etiqueta
        - Opciones de compatibilidad
        - Estados del widget
        - ttk.Widget
    - Combobox
        - Opciones
        - Eventos virtuales
        - ttk.Combobox
    - Spinbox
        - Opciones
        - Eventos virtuales
        - ttk.Spinbox
    - Notebook
        - Opciones
        - Opciones de pestañas
        - Identificadores de pestañas
        - Eventos virtuales
        - ttk.Notebook
    - Progressbar
        - Opciones
        - ttk.Progressbar

    - Separator
        - Opciones
    - Sizegrip
        - Notas específicas por plataforma
        - Errores detectados
    - Treeview
        - Opciones
        - Opciones de elementos
        - Opciones de etiqueta
        - Identificadores de columna
        - Eventos virtuales
        - ttk.Treeview
    - Ttk Styling
        - Diseños
- IDLE — Python editor and shell
    - Menús
        - Menú de archivo (Shell y Editor)
        - Menú editar (Shell y Editor)
        - Menú de formato (solo ventana del Editor)
        - Menú ejecutar (solo ventana Editor)
        - Menú de shell (solo ventana de shell)
        - Menú de depuración (solo ventana de shell)
        - Menú de opciones (Shell y editor)
        - Menú de ventana (shell y editor)
        - Menú de ayuda (shell y editor)
        - Menús contextuales
    - Edición y navegación
        - Ventana del editor

            - Atajos de teclado
            - Indentación automática
            - Buscar y reemplazar
            - Terminaciones
            - Sugerencias de llamada
            - Format block
            - Contexto del código
            - Ventana de Shell
            - Colores del texto

        - Inicio y ejecución de código

            - Command-line usage
            - Error de inicio
            - Ejecutando código del usuario
            - Salida del usuario en consola
            - Desarrollando aplicaciones tkinter
            - Ejecutando sin un subproceso

        - Ayuda y preferencias

            - Recursos de ayuda
            - Preferencias de configuración
            - IDLE en macOS
            - Extensiones

        - idlelib — implementation of IDLE application

    - `turtle` — Turtle graphics

        - Introducción
        - Empezar
        - Tutorial

            - Iniciando un entorno para tortugas
            - Dibujo básico

                - Control del lápiz
                - La posición de la tortuga

            - Creando patrones algorítmicos

- Cómo…
    - Empiece lo antes posible
    - Automatically begin and end filling
    - Utilice el espacio de nombres del módulo **`turtle`**
    - Utilice gráficos de tortugas en un guión
    - Utilice gráficos de tortuga orientados a objetos

- Referencia gráficos de tortugas
    - Métodos Turtle
    - Métodos de TurtleScreen/Screen

- Métodos de *RawTurtle/Turtle* Y sus correspondientes funciones
    - Movimiento de Turtle
    - Mostrar el estado de la tortuga
    - Configuración de las medidas
    - Control del lápiz
        - Estado de dibujo
        - Control del color
        - Relleno
        - Más controles de dibujo
    - Estado de la Tortuga
        - Visibilidad
        - Apariencia
    - Usando eventos
    - Métodos especiales de *Turtle*
    - Formas compuestas

- Métodos de *TurtleScreen/Screen* y sus correspondientes funciones
    - Control de ventana

        - Control de animación
        - Usando eventos de pantalla
        - Métodos de entrada
        - Configuración y métodos especiales
        - Métodos específicos de *Screen*, no heredados de *TurtleScreen*
      - Clases públicas
      - Explicación
      - Ayuda y configuración
        - Cómo usar la ayuda
        - Traducción de cadenas de documentos a diferentes idiomas
        - Cómo configurar Screen and Turtles
      - **`turtledemo`** — Demo scripts
      - Cambios desde Python 2.6
      - Cambios desde Python 3.0
  - Herramientas de desarrollo
    - **`typing`** — Support for type hints
      - Especificación del sistema de tipos de Python
      - Alias de tipo
      - NewType
      - Anotaciones en objetos invocables
      - Genéricos
      - Anotaciones en tuplas
      - El tipo de objetos de clase
      - Anotación de generadores y corrutinas
      - Tipos genéricos definidos por el usuario
      - El tipo **`Any`**
      - Subtipado nominal vs estructural
      - Contenido del módulo
        - Primitivos especiales de tipado
          - Tipos especiales

            - Formas especiales
            - Creación de tipos genéricos y alias de tipos
            - Otras directivas especiales
        - Protocolos
        - ABCs and Protocols for working with I/O
        - Funciones y decoradores
        - Ayudas de introspección
        - Constantes
        - Alias obsoletos
            - Alias de tipos integrados
            - Alias de tipos en **`collections`**
            - Alias a otros tipos concretos
            - Alias de ABCs de contenedores en **`collections.abc`**
            - Alias para ABCs asíncronos en **`collections.abc`**
            - Alias a otros ABCs en **`collections.abc`**
            - Alias de ABCs **`contextlib`**
    - Línea de tiempo de obsolescencia de características principales
- **`pydoc`** — Documentation generator and online help system
- Modo de desarrollo de Python
    - Efectos del modo de desarrollo de Python
    - Ejemplo de ResourceWarning
    - Ejemplo de error de descriptor de archivo incorrecto
- **`doctest`** — Test interactive Python examples
    - Uso simple: verificar ejemplos en docstrings

    - Uso Simple: Verificar ejemplos en un Archivo de Texto
    - Command-line Usage
    - Cómo funciona
        - ¿Qué docstrings son examinados?
        - ¿Cómo se reconocen los ejemplos de docstring?
        - ¿Cuál es el contexto de ejecución?
        - ¿Y las excepciones?
        - Banderas de Opción
        - Directivas
        - Advertencias
    - API básica
    - API de unittest
    - API avanzada
        - Objetos DocTest
        - Objetos *Example*
        - Objetos *DocTestFinder*
        - Objetos *DocTestParser*
        - TestResults objects
        - Objetos *DocTestRunner*
        - Objetos *OutputChecker*
    - Depuración
    - Plataforma improvisada
- **`unittest`** — Unit testing framework
    - Ejemplo sencillo
    - Interfaz de línea de comandos
        - Opciones de la línea de órdenes
    - Descubrimiento de pruebas
    - Organización del código de pruebas
    - Reutilización de código de prueba anterior
    - Omitir tests y fallos esperados
    - Distinguiendo iteraciones de tests

  empleando subtests
  - Clases y funciones
    - Casos de test
    - Agrupando tests
    - Cargando y ejecutando tests
      - load_tests protocolo
  - Instalaciones para clases y módulos
    - setUpClass y tearDownClass
    - setUpModule y tearDownModule
  - Manejo de señales
- **`unittest.mock`** — mock object library
  - Guía rápida
  - La clase Mock
    - Llamar a los objetos simulados
    - Eliminar atributos
    - Los nombres de los objetos simulados y el atributo *name*
    - Adjuntar objetos simulados como atributos
  - Parcheadores
    - patch
    - patch.object
    - patch.dict
    - patch.multiple
    - métodos start y stop de patch
    - parchear objetos incorporados (builtins)
    - TEST_PREFIX
    - Anidando decoradores patch
    - Dónde parchear
    - Parcheando descriptores y objetos proxy

    - MagicMock y el soporte de métodos mágicos
        - Simular métodos mágicos
        - Magic Mock
    - Ayudantes
        - sentinel
        - DEFAULT
        - call
        - create_autospec
        - ANY
        - FILTER_DIR
        - mock_open
        - Autoespecificación
        - Sellar objetos simulados
    - Order of precedence of **`side_effect`**, **`return_value`** and *wraps*
- **`unittest.mock`** — getting started
    - Usando mock
        - Mock patching methods
        - Mock for method calls on an object
        - Mocking classes
        - Nombrando tus mocks
        - Tracking all calls
        - Setting return values and attributes
        - Generar excepciones con mocks
        - Funciones de efectos secundarios e iterables
        - Iteradores asincrónicos de Mocking
        - El gestor de contexto asincrónico de Mocking
        - Creating a mock from an existing object
        - Uso de side_effect para devolver el contenido por archivo

        - Patch decorators
        - Further examples
            - Mocking de llamadas encadenadas
            - Mocking parcial
            - Mocking a generator method
            - Aplicar el mismo parche a cada método de prueba
            - Mocking unbound methods
            - Comprobación de varias llamadas con mock
            - Copiando con argumentos mutables
            - Nesting patches
            - Mocking a un diccionario usando MagickMock
            - Mock de subclases y sus atributos
            - Importaciones de Mocking con patch.dict
            - Seguimiento del orden de las llamadas y de las aserciones de llamadas menos detalladas
            - Coincidencia de argumentos más compleja
    - **`test`** — Regression tests package for Python
        - Writing Unit Tests for the **`test`** package
        - Ejecución de pruebas utilizando la interfaz de línea de comandos
    - **`test.support`** — Utilities for the Python test suite
    - **`test.support.socket_helper`** — Utilities for socket tests
    - **`test.support.script_helper`** — Utilities for the Python execution tests
    - **`test.support.bytecode_helper`** — Support tools for testing correct bytecode generation
    - **`test.support.threading_helper`** — Utilities for threading tests
    - **`test.support.os_helper`** — Utilities for os

    tests
  - `test.support.import_helper` — Utilities for import tests
  - `test.support.warnings_helper` — Utilities for warnings tests

- Depuración y perfilado

  - Tabla de auditoría de eventos
  - `bdb` — Debugger framework
  - `faulthandler` — Dump the Python traceback

    - Volcar el rastreo
    - Dumping the C stack

      - C Stack Compatibility

    - Estado del gestor de fallos
    - Volcar los rastreos después de un tiempo de espera
    - Volcar el rastreo en una señal del usuario
    - Problema con descriptores de archivo
    - Ejemplo

  - `pdb` — The Python Debugger

    - Command-line interface
    - Debugger commands

  - Los perfiladores de Python

    - Introducción a los perfiladores
    - Manual instantáneo de usuario
    - `profile` and `cProfile` Module Reference
    - La clase `Stats`
    - ¿Qué es el perfil determinista?
    - Limitaciones
    - Calibración
    - Usando un temporizador personalizado

  - `timeit` — Measure execution time of small code

snippets
    - Ejemplos básicos
    - Interfaz de Python
    - Interfaz de línea de comandos
    - Ejemplos
- **`trace`** — Trace or track Python statement execution
    - Uso de la línea de comandos
        - Opciones principales
        - Modificadores
        - Filtros
    - Interfaz programática
- **`tracemalloc`** — Trace memory allocations
    - Ejemplos
        - Mostrar los 10 principales
        - Calcula las diferencias
        - Consigue el seguimiento del bloque de memoria
        - **`Los 10 más bonitos`**
            - Graba los tamaños actual y máximo de todos los bloques de memoria rastreados
    - API
        - Funciones
        - Filtro de dominio
        - Filtro
        - Cuadro
        - Captura instantánea
        - Estadística
        - StatisticDiff
        - Rastro

            - Seguimiento
  - Empaquetado y distribución de software
    - **`ensurepip`** — Bootstrapping the **`pip`** installer
      - Command-line interface
      - API del módulo
    - **`venv`** — Creation of virtual environments
      - Creación de entornos virtuales
      - Cómo funcionan los venvs
      - API
      - Un ejemplo de la extensión de **`EnvBuilder`**
    - **`zipapp`** — Manage executable Python zip archives
      - Ejemplo básico
      - Interfaz de línea de comando
      - API de Python
      - Ejemplos
      - Especificar el intérprete
      - Creando aplicaciones independientes con zipapp
        - Advertencias
      - El formato de archivado Zip de aplicaciones Python
  - Servicios en tiempo de ejecución de Python
    - **`sys`** — System-specific parameters and functions
    - **`sys.monitoring`** — Execution event monitoring
      - Identificadores de herramientas
        - Registro y uso de herramientas

    - Eventos
        - Eventos locales
        - Deprecated event
        - Eventos auxiliares
        - Otros eventos
        - El evento STOP_ITERATION
    - Activar y desactivar eventos
        - Configuración de eventos globalmente
        - Eventos por objeto de código
        - Deshabilitando eventos
    - Registrando funciones de retrollamada
        - Argumentos de la función de retrollamada
- **`sysconfig`** — Provide access to Python’s configuration information
    - Variables de configuración
    - Rutas de instalación
    - User scheme
        - **`posix_user`**
        - **`nt_user`**
        - **`osx_framework_user`**
    - Home scheme
        - **`posix_home`**
    - Prefix scheme
        - **`posix_prefix`**
        - **`nt`**
    - Installation path functions
    - Otras funciones

  - Command-line usage
- `builtins` — Built-in objects
- `__main__` — Top-level code environment
  - `__name__ == '__main__'`
    - ¿Qué es el «entorno de código de máximo nivel»?
    - Uso idiomático
    - Consideraciones de empaquetado
  - `__main__.py` en paquetes de Python
    - Uso idiomático
  - `import __main__`
- `warnings` — Warning control
  - Categorías de advertencia
  - El filtro de advertencias
    - Repeated Warning Suppression Criteria
    - Descripción de los filtros de advertencia
    - Filtro de advertencia predeterminado
    - Anulando el filtro por defecto
  - Eliminación temporal de las advertencias
  - Advertencias de prueba
  - Actualización del código para las nuevas versiones de las dependencias
  - Funciones disponibles
  - Gestores de contexto disponibles
  - Concurrent safety of Context Managers
- `dataclasses` — Data Classes
  - Contenidos del módulo
  - Procesamiento posterior a la inicialización

  - Variables de clase
  - Variable de solo inicialización
  - Instancias congeladas
  - Herencia
  - Re-ordering of keyword-only parameters in `__init__()`
  - Funciones fábrica por defecto
  - Valores por defecto mutables
  - Campos tipo descriptor
- `contextlib` — Utilidades para declaraciones de contexto `with`
  - Utilidades
  - Ejemplos y recetas
    - Apoyando un número variable de gestores de contexto
    - Capturando excepciones de los métodos `__enter__`
    - Limpieza en una implementación `__enter__`
    - Reemplazar cualquier uso de `try-finally` y marcar variables
    - Usar un gestor de contexto como decorador de funciones
  - Gestores de contexto de uso único, reutilizables y reentrantes
    - Gestores contextuales reentrantes
    - Gestores contextuales reutilizables
- `abc` — Abstract Base Classes
- `atexit` — Exit handlers
  - `atexit` Example
- `traceback` — Print or retrieve a stack traceback
  - Module-Level Functions

    - `TracebackException` Objects
    - `StackSummary` Objects
    - `FrameSummary` Objects
    - Examples of Using the Module-Level Functions
    - Examples of Using `TracebackException`
- `__future__` — Future statement definitions
    - Module Contents
- `gc` — Garbage Collector interface
- `inspect` — Inspect live objects
    - Tipos y miembros
    - Recuperar el código fuente
    - Introspección de los invocables con el objeto Signature
    - Clases y funciones
    - La pila del interprete
    - Obteniendo atributos estáticamente
    - Current State of Generators, Coroutines, and Asynchronous Generators
    - Objetos de código Bit Flags
    - Buffer flags
    - Command-line interface
- `annotationlib` — Functionality for introspecting annotations
    - Annotation semantics
    - Classes
    - Functions
    - Recipes
        - Using annotations in a metaclass
    - Limitations of the `STRING` format
    - Limitations of the `FORWARDREF` format
    - Security implications of introspecting

annotations

- `site` — Site-specific configuration hook
    - `sitecustomize`
    - `usercustomize`
    - Configuración de *Readline*
    - Contenido del módulo
    - Command-line interface

○ Intérpretes de Python personalizados

- `code` — Interpreter base classes
    - Objetos de intérprete interactivo
    - Objetos de consola interactiva
- `codeop` — Compile Python code

○ Importando módulos

- `zipimport` — Import modules from Zip archives
    - Objetos zipimporter
    - Ejemplos
- `pkgutil` — Package extension utility
- `modulefinder` — Find modules used by a script
    - Ejemplo de uso de `ModuleFinder`
- `runpy` — Localización y ejecución de módulos Python
- `importlib` — La implementación de `import`
    - Introducción
    - Funciones
    - `importlib.abc` – Abstract base classes related to import
    - `importlib.machinery` – Importers and path hooks
    - `importlib.util` – Utility code for

importers

  - Ejemplos
    - Importar programáticamente
    - Comprobando si se puede importar un módulo
    - Importar un archivo fuente directamente
    - Implementar importaciones diferidas
    - Configurar un importador
    - Aproximando **`importlib.import_module()`**
- **`importlib.resources`** – Package resource reading, opening and access
  - Functional API
- **`importlib.resources.abc`** – Abstract base classes for resources
- **`importlib.metadata`** – Acceso a los metadatos de los paquetes
  - Descripción general
  - API funcional
    - Puntos de entrada
    - Metadatos de distribución
    - Versiones de distribución
    - Archivos de distribución
    - Requerimientos de la distribución
    - Mapeo de paquetes de importación a distribución
  - Distribuciones
  - Distribution Discovery
  - Implementing Custom Providers
    - Example
- La inicialización de la ruta de búsqueda de

      - módulo de `sys.path`
        - Virtual Environments
        - Archivos _pth
        - Python embebido
- Servicios del lenguaje Python
  - `ast` — Abstract syntax trees
    - Abstract grammar
    - Clases nodo
      - Nodos raíz
      - Literales
      - Variables
      - Expresiones
        - Subindexado
        - Comprensiones
      - Declaraciones
        - Importaciones
      - Control de flujo
      - La coincidencia de patrones
      - Type annotations
      - Tipos de parámetro
      - Definiciones de función y clase
      - Async y await
    - `ast` helpers
    - Compiler flags
    - Command-line usage
  - `symtable` — Access to the compiler's symbol tables
    - Generando tablas de símbolos
    - Examinando la tabla de símbolos
    - Command-Line Usage

  - `token` — Constants used with Python parse trees
  - `keyword` — Testing for Python keywords
  - `tokenize` — Tokenizer for Python source
    - Convirtiendo la entrada en *tokens*
    - Uso como línea de comandos
    - Ejemplos
  - `tabnanny` — Detection of ambiguous indentation
  - `pyclbr` — Python module browser support
    - Objetos Function
    - Objetos Class
  - `py_compile` — Compile Python source files
    - Interfaz de línea de comandos
  - `compileall` — Byte-compile Python libraries
    - Uso de la línea de comandos
    - Funciones públicas
  - `dis` — Desensamblador para bytecode de Python
    - Interfaz de línea de comandos
    - Análisis de bytecode
    - Funciones de análisis
    - Instrucciones bytecode de Python
    - Colecciones opcode
  - `pickletools` — Tools for pickle developers
    - Command-line usage
      - Command-line options
    - Programmatic interface
- Servicios Específicos para MS Windows

  - `msvcrt` — Useful routines from the MS VC++ runtime
    - Operaciones con archivos
    - Consola E/S
    - Otras funciones
  - `winreg` — Windows registry access
    - Funciones
    - Constantes
      - HKEY_* Constantes
      - Access Rights
        - Específico de 64 bits
      - Tipos de valor
    - Objetos de control del registro
  - `winsound` — Sound-playing interface for Windows
- Unix-specific services
  - `shlex` — Simple lexical analysis
    - objetos `shlex`
    - Reglas de análisis
    - Compatibilidad mejorada con intérprete de comandos
  - `posix` — The most common POSIX system calls
    - Soporte de archivos grandes
    - Contenido notable del módulo
  - `pwd` — The password database
  - `grp` — The group database
  - `termios` — POSIX style tty control

      - Ejemplo
    - `tty` — Terminal control functions
    - `pty` — Pseudo-terminal utilities
      - Ejemplo
    - `fcntl` — The `fcntl` and `ioctl` system calls
    - `resource` — Resource usage information
      - Límites de recursos
      - Utilización de recursos
    - `syslog` — Unix syslog library routines
      - Ejemplos
        - Ejemplo sencillo
  - Modules command-line interface (CLI)
  - Módulos reemplazados
    - `getopt` — C-style parser for command line options
  - Removed Modules
  - Consideraciones de seguridad
- Ampliación e incrustación del intérprete de Python
  - Herramientas de terceros recomendadas
  - Crear extensiones sin herramientas de terceros
    - 1. Extendiendo Python con C o C++
      - 1.1. Un ejemplo simple
      - 1.2. Intermezzo: errores y excepciones
      - 1.3. De vuelta al ejemplo
      - 1.4. La tabla de métodos del módulo y la función de inicialización
      - 1.5. Compilación y enlazamiento
      - 1.6. Llamando funciones Python desde C

  - 1.7. Extracción de parámetros en funciones de extensión
  - 1.8. Parámetros de palabras clave para funciones de extensión
  - 1.9. Construyendo valores arbitrarios
  - 1.10. Conteo de referencias
    - 1.10.1. Conteo de referencias en Python
    - 1.10.2. Reglas de propiedad
    - 1.10.3. Hielo delgado
    - 1.10.4. Punteros NULL
  - 1.11. Escribiendo extensiones en C++
  - 1.12. Proporcionar una API C para un módulo de extensión
- 2. Definición de tipos de extensión: Tutorial
  - 2.1. Lo básico
  - 2.2. Agregar datos y métodos al ejemplo básico
  - 2.3. Proporcionar un control más preciso sobre los atributos de datos
  - 2.4. Apoyo a la recolección de basura cíclica
  - 2.5. Subclases de otros tipos
- 3. Definición de tipos de extensión: temas variados
  - 3.1. Finalización y desasignación
  - 3.2. Presentación de objetos
  - 3.3. Gestión de atributos
    - 3.3.1. Gestión de atributos genéricos
    - 3.3.2. Gestión de atributos específicos de tipo
  - 3.4. Comparación de Objetos
  - 3.5. Soporte de protocolo abstracto

      - 3.6. Soporte de referencia débil
      - 3.7. Más Sugerencias
    - 4. Construyendo extensiones C y C++
      - 4.1. Building C and C++ Extensions with setuptools
    - 5. Creación de extensiones C y C++ en Windows
      - 5.1. Un enfoque de libro de cocina
      - 5.2. Diferencias entre Unix y Windows
      - 5.3. Usar DLL en la práctica
  - Incrustar el tiempo de ejecución de CPython en una aplicación más grande
    - 1. Incrustando Python en otra aplicación
      - 1.1. Incrustación de muy alto nivel
      - 1.2. Más allá de la incrustación de muy alto nivel: una visión general
      - 1.3. Incrustación pura
      - 1.4. Extendiendo Python incrustado
      - 1.5. Incrustando Python en C++
      - 1.6. Compilar y enlazar bajo sistemas tipo Unix
- Python/C API reference manual
  - Introducción
    - Language version compatibility
    - Estándares de codificación
    - Archivos de cabecera (*Include*)
    - Macros útiles
      - Docstring macros
      - General utility macros
        - Numeric utilities
        - Assertion utilities

        - Type size utilities
        - Macro definition utilities
      - Declaration utilities
      - Outdated macros
    - Objetos, tipos y conteos de referencias
      - Conteo de Referencias
        - Detalles del conteo de referencia
      - Tipos
    - Excepciones
    - Integración de Python
    - Depuración de compilaciones
    - Recommended third party tools
  - Estabilidad de la API en C
    - Unstable C API
    - Interfaz binaria de aplicación estable
      - Limited C API
      - Stable ABI
      - Alcance y rendimiento de la API limitada
      - Advertencias de la API limitada
    - Consideraciones de la plataforma
    - Contenido de la API limitada
  - La capa de muy alto nivel
    - Available start symbols
    - Stack Effects
  - Conteo de referencias
  - Manejo de excepciones
    - Impresión y limpieza
    - Lanzando excepciones

    - Emitir advertencias
    - Consultando el indicador de error
    - Manejo de señal
    - Clases de excepción
    - Objetos excepción
    - Objetos unicode de excepción
    - Control de recursión
    - Exception and warning types
        - Exception types
        - OSError aliases
        - Warning types
    - Tracebacks
- Defining extension modules
    - Multiple module instances
    - Initialization function
    - Multi-phase initialization
    - Legacy single-phase initialization
- Utilidades
    - Utilidades del sistema operativo
    - Funciones del Sistema
    - Control de procesos
    - Importando módulos
    - Soporte de empaquetado (*marshalling*) de datos
    - Analizando argumentos y construyendo valores
        - Analizando argumentos
            - Cadena de caracteres y búferes
            - Números
            - Otros objetos
            - Funciones API
        - Construyendo valores
    - Conversión y formato de cadenas de caracteres
    - Character classification and conversion

  - PyHash API
  - Reflexión
  - Registro de códec y funciones de soporte
    - API de búsqueda de códec
    - API de registro para controladores de errores de codificación Unicode
    - Codec utility variables
  - PyTime C API
    - Types
    - Clock Functions
    - Raw Clock Functions
    - Conversion functions
  - Soporte para Mapeo Perf
- Capa de objetos abstractos
  - Protocolo de objeto
  - Protocolo de llamada
    - El protocolo *tp_call*
    - El protocolo vectorcall
      - Control de recursión
      - API de soporte para vectorcall
    - API para invocar objetos
    - API de soporte de llamadas
  - Protocolo de números
  - Protocolo de secuencia
  - Protocolo de mapeo
  - Protocolo iterador
  - Protocolo búfer
    - Estructura de búfer
    - Tipos de solicitud búfer
      - campos independientes de solicitud

      - formato de sólo lectura
      - formas, *strides*, *suboffsets*
      - solicitudes de contigüidad
      - solicitudes compuestas
    - Arreglos complejos
      - Estilo NumPy: forma y *strides*
      - Estilo PIL: forma, *strides* y *suboffsets*
    - Funciones relacionadas a búfer
- Capa de objetos concretos
  - Objetos fundamentales
    - Objetos tipo
      - Crear tipos asignados en montículo (*heap*)
    - El objeto **`None`**
  - Objetos numéricos
    - Objetos enteros
      - Export API
      - PyLongWriter API
      - Deprecated API
    - Objetos booleanos
    - Floating-Point Objects
      - Funciones de empaquetado y desempaquetado
        - Funciones de Empaquetado
        - Funciones de Desempaquetado
    - Objetos de números complejos
      - Números complejos como estructuras

C
    - Números complejos como objetos de Python
- Objetos de secuencia
  - Objetos bytes
  - Objetos de arreglos de bytes (*bytearrays*)
    - Macros de verificación de tipos
    - Funciones API directas
    - Macros
  - Objetos y códecs unicode
    - Objetos unicode
      - Tipo unicode
      - Propiedades de caracteres Unicode
      - Creando y accediendo a cadenas de caracteres Unicode
      - Codificación regional
      - Codificación del sistema de archivos
      - soporte wchar_t
    - Códecs incorporados
      - Códecs genéricos
      - Códecs UTF-8
      - Códecs UTF-32
      - Códecs UTF-16
      - Códecs UTF-7
      - Códecs Unicode escapado
      - Códecs para Unicode escapado en bruto
      - Códecs Latin-1
      - Códecs ASCII
      - Códecs de mapa de caracteres
      - Códecs MBCS para Windows

        - Métodos y funciones de ranura (*Slot*)
        - PyUnicodeWriter
        - Deprecated API

    - Objetos tupla
    - Objetos de secuencia de estructura
    - Objetos lista

- Objetos contenedor

    - Objetos diccionario

        - Dictionary View Objects
        - Ordered Dictionaries

    - Objetos conjunto

        - Deprecated API

- Objetos de función

    - Objetos función
    - Objetos de método de instancia
    - Objetos método
    - Objetos celda
    - Objetos código
    - Code Object Flags
    - Extra information

- Otros objetos

    - File objects

        - Soft-deprecated API

    - Objetos módulo
    - Module definitions

        - Module slots

    - Creating extension modules dynamically
    - Funciones de soporte

        - Module lookup (single-phase initialization)

    - Objetos iteradores

        - Range Objects
        - Builtin Iterator Types
        - Other Iterator Objects

    - Objetos descriptores

        - Built-in descriptors

    - Objetos rebanada (*slice*)

        - Objeto elipsis

    - Objetos de vista de memoria (*MemoryView*)
    - Pickle buffer objects
    - Objetos de referencia débil
    - Cápsulas
    - Frame objects

        - Frame locals proxies
        - Legacy local variable APIs
        - Internal frames

    - Objetos generadores

        - Asynchronous Generator Objects
        - Deprecated API

    - Objetos corrutina
    - Objetos de variables de contexto
    - Objetos para indicaciones de tipado

- C API for extension modules

    - Curses C API
    - Internal data
    - Objetos *DateTime*
    - Internal data

- Interpreter initialization and finalization
    - Before Python initialization
    - Global configuration variables
    - Initializing and finalizing the interpreter
    - Cautions regarding runtime finalization
    - Process-wide parameters
- Thread states and the global interpreter lock
    - Detaching the thread state from extension code
        - APIs
    - Non-Python created threads
    - Legacy API
    - Cautions about fork()
    - High-level APIs
    - GIL-state APIs
    - Low-level APIs
- Asynchronous notifications
- Operating system thread APIs
- Synchronization primitives
    - Python critical section API
    - Legacy locking APIs
- Thread-local storage support
    - Thread-specific storage API
    - Dynamic allocation
    - Methods
    - Legacy APIs
- Multiple interpreters in a Python process
    - A per-interpreter GIL
    - Bugs and caveats
    - High-level APIs
    - Low-level APIs
    - Advanced debugger support

- Profiling and tracing
- Reference tracing
- Configuración de inicialización de Python
    - PyInitConfig C API
        - Ejemplo
        - Create Config
        - Error Handling
        - Get Options
        - Set Options
        - Module
        - Initialize Python
    - Configuration Options
    - Runtime Python configuration API
    - PyConfig C API
        - Ejemplo
        - PyWideStringList
        - PyStatus
        - PyPreConfig
        - Preinicialización de Python con PyPreConfig
        - PyConfig
        - Inicialización con PyConfig
        - Configuración aislada
        - Configuración de Python
        - Configuración de la ruta de Python
    - Py_GetArgcArgv()
    - Delaying main module execution
- Gestión de la memoria
    - Visión general
    - Dominios del asignador
    - Interfaz de memoria sin procesar
    - Interfaz de memoria
        - Deprecated aliases

    - Asignadores de objetos
    - Asignadores de memoria predeterminados
    - Personalizar asignadores de memoria
    - Configurar enlaces para detectar errores en las funciones del asignador de memoria de Python
    - El asignador pymalloc
        - Personalizar asignador de arena de pymalloc
    - El asignador mimalloc
    - tracemalloc C API
    - Ejemplos
- Soporte de implementación de objetos
    - Allocating objects on the heap
        - Soft-deprecated aliases
    - Object Life Cycle
        - Life Events
        - Cyclic Isolate Destruction
        - Functions
    - Estructuras de objetos comunes
        - Tipos objeto base y macros
        - Implementando funciones y métodos
        - Acceder a atributos de tipos de extensión
            - Member flags
            - Member types
            - Defining Getters and Setters
    - Type Object Structures
        - Referencia rápida
            - «ranuras *tp*» (*tp slots*)
            - sub-ranuras (*sub-slots*)

            - ranura de *typedefs*
        - Definición de **`PyTypeObject`**
        - Ranuras (*Slots*) **`PyObject`**
        - Ranuras **`PyVarObject`**
        - Ranuras **`PyTypeObject`**
        - Tipos estáticos
        - Tipos Heap
        - Estructuras de objetos de números
        - Estructuras de objetos mapeo
        - Estructuras de objetos secuencia
        - Estructuras de objetos búfer
        - Estructuras de objetos asíncronos
        - Tipo Ranura *typedefs*
        - Ejemplos
      - Apoyo a la recolección de basura cíclica
        - Controlar el estado del recolector de basura
        - Consultar el estado del recolector de basura
  - Versiones de API y ABI
    - Build-time version constants
    - Run-time version
    - Bit-packing macros
  - Monitoring C API
  - Generating Execution Events
    - Managing the Monitoring State
- Installing Python modules
  - Palabras clave
  - Uso básico
  - ¿Cómo…
    - … instalo paquetes solamente para el usuario actual?
    - … instalo paquetes científicos de Python?
    - … trabajo con múltiples versiones de Python

        instaladas en paralelo?
    - Problemas de instalación comunes
        - Instalando en el Python del sistema bajo Linux
        - Pip no está instalado
        - Instalando extensiones binarias
- Comos (*HOWTOs*) de Python
- Preguntas más frecuentes de Python
    - Preguntas frecuentes generales sobre Python
        - Información general
        - Python en el mundo real
    - Preguntas frecuentes de programación
        - General questions
        - Core language
        - Números y cadenas
        - Rendimiento
        - Sequences (tuples/lists)
        - Objetos
        - Módulos
    - Preguntas frecuentes sobre diseño e historia
        - ¿Por qué Python usa indentación para agrupar declaraciones?
        - ¿Por qué obtengo resultados extraños con operaciones aritméticas simples?
        - ¿Por qué los cálculos de punto flotante son tan inexactos?
        - ¿Por qué las cadenas de caracteres de Python son inmutables?
        - ¿Por qué debe usarse "self" explícitamente en las definiciones y llamadas de métodos?
        - ¿Por qué no puedo usar una tarea en una expresión?
        - ¿Por qué Python usa métodos para alguna

funcionalidad (por ejemplo, list.index()) pero funciones para otra (por ejemplo, len(list))?

- ¿Por qué join() es un método de cadena de caracteres en lugar de un método de lista o tupla?
- ¿Qué tan rápido van las excepciones?
- ¿Por qué no hay un *switch* o una declaración *case* en Python?
- ¿No puede emular hilos en el intérprete en lugar de confiar en una implementación de hilos específica del sistema operativo?
- ¿Por qué las expresiones lambda no pueden contener sentencias?
- ¿Se puede compilar Python en código máquina, C o algún otro lenguaje?
- ¿Cómo gestiona Python la memoria?
- ¿Por qué CPython no utiliza un esquema de recolección de basura más tradicional?
- ¿Por qué no se libera toda la memoria cuando sale CPython?
- ¿Por qué hay tipos de datos separados de tuplas y listas?
- ¿Cómo se implementan las listas en Python?
- ¿Cómo se implementan los diccionarios en CPython?
- ¿Por qué las claves del diccionario deben ser inmutables?
- ¿Por qué list.sort() no retorna la lista ordenada?
- ¿Cómo se especifica y aplica una especificación de interfaz en Python?
- ¿Por qué no hay goto?
- ¿Por qué las cadenas de caracteres sin formato (r-strings) no pueden terminar con una barra diagonal inversa?
- ¿Por qué Python no tiene una declaración «with» para las asignaciones de atributos?
- ¿Por qué los generadores no admiten la declaración with?
- ¿Por qué se requieren dos puntos para las declaraciones if/while/def/class?
- ¿Por qué Python permite comas al final de las

    listas y tuplas?

- Preguntas frecuentes sobre bibliotecas y extensiones
    - Cuestiones generales sobre bibliotecas
    - Tareas comunes
    - Hilos
    - Entrada y salida
    - Programación de Redes/Internet
    - Bases de datos
    - Matemáticas y numérica

- Extendiendo/Embebiendo FAQ
    - ¿Puedo crear mis propias funciones en C?
    - ¿Puedo crear mis propias funciones en C++?
    - Escribir en C es difícil; ¿no hay otra alternativa?
    - ¿Cómo puedo ejecutar declaraciones arbitrarias de Python desde C?
    - ¿Cómo puedo evaluar una expresión arbitraria de Python desde C?
    - ¿Cómo extraigo valores C de un objeto Python?
    - ¿Cómo utilizo Py_BuildValue() para crear una tupla de un tamaño arbitrario?
    - ¿Cómo puedo llamar un método de un objeto desde C?
    - ¿Cómo obtengo la salida de PyErr_Print() (o cualquier cosa que se imprime a stdout/stderr)?
    - ¿Cómo accedo al módulo escrito en Python desde C?
    - ¿Cómo hago una interface a objetos C++ desde Python?
    - He agregado un módulo usando el archivo de configuración y el *make* falla. ¿Porque?
    - ¿Cómo puedo depurar una extensión?
    - Quiero compilar un módulo Python en mi sistema Linux, pero me faltan algunos archivos . ¿Por qué?
    - ¿Cómo digo «entrada incompleta» desde «entrada inválida»?
    - ¿Cómo encuentro símbolos g++ __builtin_new o

    _pure_virtual?
    - ¿Puedo crear una clase objeto con algunos métodos implementado en C y otros en Python (por ejemplo a través de la herencia)?
  - Preguntas frecuentes sobre Python en Windows
    - ¿Cómo ejecutar un programa Python en Windows?
    - ¿Cómo hacer que los scripts de Python sean ejecutables?
    - ¿Por qué Python tarda en comenzar?
    - ¿Cómo hacer un ejecutable a partir de un script de Python?
    - ¿Es un archivo **`*.pyd`** lo mismo que una DLL?
    - ¿Cómo puedo integrar Python en una aplicación de Windows?
    - ¿Cómo puedo evitar que mi editor inserte pestañas en mi archivo fuente de Python?
    - ¿Cómo verifico una pulsación de tecla sin bloquearla?
    - ¿Cómo resuelvo el error de api-ms-win-crt-runtime-l1-1-0.dll no encontrado?
  - Preguntas frecuentes sobre la Interfaz Gráfica de Usuario (*GUI*)
    - Preguntas generales de la GUI
    - ¿Qué conjuntos de herramientas de GUI existen para Python?
    - Preguntas de Tkinter
  - «¿Por qué está Python instalado en mi ordenador?» FAQ
    - ¿Qué es Python?
    - ¿Por qué Python está instalado en mi máquina?
    - ¿Puedo eliminar Python?
- Desusos

    - Pending removal in Python 3.15
    - Pending removal in Python 3.16
    - Pending removal in Python 3.17
    - Pending removal in Python 3.18
    - Pending removal in Python 3.19
    - Pending removal in future versions
    - C API deprecations
        - Pending removal in Python 3.15
        - Pending removal in Python 3.18
        - Pending removal in future versions
- Glosario
- About this documentation
    - Contributors to the Python documentation
- Lidiar con errores
    - Documentación de errores
    - Utilizar el issue tracker de Python
    - Para empezar a contribuir en Python
- Derechos de autor
- Historia y Licencia
    - Historia del software
    - Términos y condiciones para acceder o usar Python
        - PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2
        - ACUERDO DE LICENCIA DE BEOPEN.COM PARA PYTHON 2.0
        - ACUERDO DE LICENCIA CNRI PARA PYTHON 1.6.1
        - ACUERDO DE LICENCIA CWI PARA PYTHON 0.9.0 HASTA 1.2
        - ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON DOCUMENTATION
    - Licencias y reconocimientos para software incorporado

- Mersenne Twister
- Sockets
- Servicios de socket asincrónicos
- Gestión de cookies
- Seguimiento de ejecución
- funciones UUencode y UUdecode
- Llamadas a procedimientos remotos XML
- test_epoll
- Seleccionar kqueue
- SipHash24
- strtod y dtoa
- OpenSSL
- expat
- libffi
- zlib
- cfuhash
- libmpdec
- Conjunto de pruebas W3C C14N
- mimalloc
- asyncio
- Global Unbounded Sequences (GUS)
- Zstandard bindings

Qué hay de nuevo en Python

La serie «Qué hay de nuevo en Python» da un vistazo a los cambios más importantes entre las versiones de Python. Son de «lectura obligatoria» para cualquier persona que quiera estar al día después de un nuevo lanzamiento.

- What’s new in Python 3.14
    - Summary – Release highlights
    - New features
    - Other language changes
    - New modules
    - Improved modules
    - Optimizations
    - Removed
    - Deprecated
    - CPython bytecode changes
    - C API changes
    - Build changes
    - Porting to Python 3.14
    - Notable changes in 3.14.1
- What’s New In Python 3.13
    - Summary – Release Highlights
    - New Features
    - Other Language Changes
    - New Modules
    - Improved Modules
    - Optimizations
    - Removed Modules And APIs
    - New Deprecations
    - CPython Bytecode Changes
    - C API Changes
    - Build Changes
    - Porting to Python 3.13

  - Regression Test Changes
- Novedades de Python 3.12
  - Resumen: aspectos destacados de la versión
  - Nuevas características
  - Nuevas funciones relacionadas con las sugerencias de escritura
  - Otros cambios del lenguaje
  - Nuevos módulos
  - Módulos mejorados
  - Optimizaciones
  - Cambios en el código de bytes de CPython
  - Demostraciones y herramientas
  - Obsoleto
  - Eliminado
  - Portar a Python 3.12
  - Cambios de compilación
  - Cambios en la API de C
- Qué hay de nuevo en Python 3.11
  - Resumen – Aspectos destacados de la versión
  - Nuevas características
  - Nuevas funciones relacionadas con las sugerencias de tipo
  - Otros cambios de idioma
  - Otros cambios en la implementación de CPython
  - Nuevos Módulos
  - Módulos mejorados
  - Optimizaciones
  - CPython más rápido
  - Cambios en el código de bytes de CPython
  - Obsoleto
  - Eliminación pendiente en Python 3.12
  - Remoto
  - Migración a Python 3.11
  - Construir cambios
  - Cambios en la API de C
  - Notable changes in 3.11.4
  - Notable changes in 3.11.5

- Qué hay de nuevo en Python 3.10
    - Resumen: aspectos destacados de la versión
    - Nuevas características
    - Nuevas funciones relacionadas con las sugerencias de tipos
    - Otros cambios de idioma
    - Nuevos módulos
    - Módulos mejorados
    - Optimizaciones
    - Obsoleto
    - Eliminado
    - Portar a Python 3.10
    - Cambios en el código de bytes de CPython
    - Construir cambios
    - Cambios en la API de C
    - Característica de seguridad notable en 3.10.7
    - Característica de seguridad notable en 3.10.8
    - Cambios notables en 3.10.12
- Qué hay de nuevo en Python 3.9
    - Resumen: aspectos destacados de la versión
    - Debe verificar DeprecationWarning en su código
    - Nuevas características
    - Otros cambios de idioma
    - Nuevos módulos
    - Módulos mejorados
    - Optimizaciones
    - Obsoleto
    - Remoto
    - Portar a Python 3.9
    - Construir cambios
    - Cambios en la API de C
    - Cambios notables en Python 3.9.1
    - Cambios notables en Python 3.9.2
    - Notable changes in Python 3.9.3
    - Notable changes in Python 3.9.5
    - Notable security feature in 3.9.14
    - Notable changes in 3.9.17

- Qué hay de nuevo en Python 3.8
    - Resumen – Aspectos destacados de la versión
    - Nuevas características
    - Otros cambios en el lenguaje
    - Nuevos módulos
    - Módulos mejorados
    - Optimizaciones
    - Cambios en la compilación y la API de C
    - Obsoleto
    - APIs y características eliminadas
    - Portando a Python 3.8
    - Cambios notables en Python 3.8.1
    - Notable changes in Python 3.8.2
    - Notable changes in Python 3.8.3
    - Cambios notables en Python 3.8.8
    - Notable changes in Python 3.8.9
    - Notable changes in Python 3.8.10
    - Notable changes in Python 3.8.10
    - Cambios notables en Python 3.8.1
    - Notable security feature in 3.8.14
    - Notable changes in 3.8.17
- Qué hay de nuevo en Python 3.7
    - Resumen – Lanzamientos Destacados
    - Nuevas Características
    - Otros cambios en el lenguaje
    - Nuevos módulos
    - Módulos mejorados
    - Cambios en la API C
    - Construir cambios
    - Optimizaciones
    - Otros cambios de implementación de CPython
    - Comportamiento obsoleto de Python
    - Módulos, funciones y métodos de Python obsoletos
    - Funciones y tipos obsoletos de la API C
    - Eliminación de soporte de plataforma
    - Eliminaciones de API y funciones
    - Eliminaciones de módulos

    - Cambios solo en Windows
    - Portando a Python 3.7
    - Cambios notables en Python 3.7.1
    - Cambios notables en Python 3.7.2
    - Cambios notables en Python 3.7.6
    - Cambios notables en Python 3.7.10
    - Cambios notables en Python 3.7.11
    - Cambios notables en Python 3.7.14

- Qué hay de nuevo en Python 3.6

    - Resumen: aspectos destacados de la versión
    - Nuevas características
    - Otros cambios de idioma
    - Nuevos módulos
    - Módulos mejorados
    - Optimizaciones
    - Cambios en la API de Build y C
    - Otras mejoras
    - Obsoleto
    - Remoto
    - Portar a Python 3.6
    - Cambios notables en Python 3.6.2
    - Cambios notables en Python 3.6.4
    - Cambios notables en Python 3.6.5
    - Cambios notables en Python 3.6.7
    - Cambios notables en Python 3.6.10
    - Cambios notables en Python 3.6.13
    - Cambios notables en Python 3.6.14

- Qué hay de nuevo en Python 3.5

    - Resumen – Aspectos destacados de la versión
    - Nuevas características
    - Otros cambios en el lenguaje
    - Nuevos módulos
    - Módulos mejorados
    - Otros cambios a nivel de módulo
    - Optimizaciones
    - Cambios en la compilación y la API de C
    - Obsoleto

  - Eliminado
  - Portando a Python 3.5
  - Cambios notables en Python 3.5.4

- Qué hay de nuevo en Python 3.4

  - Resumen – Puntos destacados del lanzamiento
  - Nuevas Funciones
  - Nuevos Módulos
  - Módulos mejorados
  - Cambios en la implementación de CPython
  - Obsoleto
  - Removido
  - Adaptación a Python 3.4
  - Cambiado en 3.4.3

- Qué hay de nuevo en Python 3.3

  - Resumen – Aspectos destacados de la versión
  - PEP 405: Entornos virtuales
  - PEP 420: Paquetes para espacios de nombres implícitos
  - PEP 3118: Nueva implementación de vista de memoria y en la documentación del protocolo del buffer
  - PEP 393: Representación flexible de cadenas de caracteres
  - PEP 397: Lanzador de python para windows
  - PEP 3151: Reelaborando de la jerarquía de excepciones de IO y OS
  - PEP 380: Sintaxis para delegar en un subgenerador
  - PEP 409: Suprimir el contexto de excepción
  - PEP 414: Literales *Unicode* explícitos
  - PEP 3155: Nombres calificados para clases y funciones
  - PEP 412: Diccionario de intercambio de claves
  - PEP 362: Objeto de firma de función
  - PEP 421: Agregar sys.implementation
  - Usar importlib como implementación de Import
  - Otros cambios de idioma
  - Un bloqueo de importación más detallado
  - Funciones y tipos incorporados
  - Nuevos módulos
  - Módulos mejorados

  - Optimizaciones
  - Construcción y cambios en la API de C
  - Obsoleto
  - Migración a Python 3.3

- Qué hay de nuevo en Python 3.2

  - PEP 384: Definición de un ABI estable
  - PEP 389: Módulo de análisis sintáctico (*Parser*) de línea de comandos Argparse
  - PEP 391: Configuración basada en diccionario para Logging
  - PEP 3148: El módulo **`concurrent.futures`**
  - PEP 3147: Directorios del repositorio de PYC
  - PEP 3149: Archivos .so con etiquetado de versión para ABI
  - PEP 3333: Interfaz de puerta de enlace del servidor web Python v1.0.1
  - Otros cambios de lenguaje
  - Módulos nuevos, mejorados y obsoletos
  - Multi-threading
  - Optimizations
  - Unicode
  - Codecs
  - Documentación
  - IDLE
  - Repositorio de código
  - Cambios en la API de construcción y C
  - Portar a Python 3.2

- Qué hay de nuevo en Python 3.1

  - PEP 372: Diccionarios ordenados
  - PEP 378: Especificador de formato para el separador de miles
  - Otros cambios del lenguaje
  - Módulos nuevos, mejorados y obsoletos
  - Optimizaciones
  - IDLE
  - Cambios en la compilación y la API de C
  - Portando a Python 3.1

- Qué hay de nuevo en Python 3.0
    - Escollos comunes
    - Descripción general de los cambios de sintaxis
    - Cambios ya presentes en Python 2.6
    - Cambios de biblioteca
    - **PEP 3101**: Un nuevo enfoque al formateo de cadena de caracteres
    - Cambios a excepciones
    - Otros cambios diversos
    - Construcción y cambios a la API de C
    - Rendimiento
    - Migración a Python 3.0
- Qué hay de nuevo en Python 2.7
    - El futuro de Python 2.x
    - Cambios en el manejo de las advertencias de desuso
    - Características de Python 3.1
    - PEP 372: Adición de un diccionario ordenado a las colecciones
    - PEP 378: Especificador de formato para separador de miles
    - PEP 389: El módulo argparse para el análisis de líneas de comando
    - PEP 391: Configuración basada en diccionarios para el registro
    - PEP 3106: Vistas de diccionario
    - PEP 3137: El objeto memoryview
    - Otros cambios de lenguaje
    - Módulos nuevos y mejorados
    - Cambios en la API de construcción y C
    - Otros cambios y correcciones
    - Adaptación a Python 2.7
    - Nuevas funciones añadidas a las versiones de mantenimiento de Python 2.7
    - Agradecimientos
- Qué hay de nuevo en Python 2.6
    - Python 3.0

  - Cambios en el proceso de desarrollo
  - PEP 343: La sentencia *“with”*
  - PEP 366: Importaciones relativas explícitas desde un módulo principal
  - PEP 370: Directorio de **`site-packages`** por usuario
  - PEP 371: El paquete **`multiprocessing`**
  - PEP 3101: Formateo avanzado de cadena de caracteres
  - PEP 3105: **`print`** como función
  - PEP 3110: Cambios en el manejo de excepciones
  - PEP 3112: Literales de bytes
  - PEP 3116: Nueva biblioteca de E/S
  - PEP 3118: Protocolo revisado de la memoria intermedia
  - PEP 3119: Clases base abstractas
  - PEP 3127: Soporte y sintaxis de literales enteros
  - PEP 3129: Decoradores de clase
  - PEP 3141: Una jerarquía de tipos para los números
  - Otros cambios lingüísticos
  - Módulos nuevos y mejorados
  - Cancelaciones y eliminaciones
  - Cambios en la API de construcción y C
  - Adaptación a Python 2.6
  - Agradecimientos

- Qué hay de nuevo en Python 2.5

  - PEP 308: Expresiones condicionales
  - PEP 309: Aplicación parcial de funciones
  - PEP 314: Metadatos para paquetes de software Python v1.1
  - PEP 328: Importaciones absolutas y relativas
  - PEP 338: Ejecutando Módulos como Scripts
  - PEP 341: Try/except/finally unificados
  - PEP 342: Nuevas funciones del generador
  - PEP 343: La declaración «con
  - PEP 352: Las excepciones como clases de nuevo estilo
  - PEP 353: Uso de ssize_t como tipo de índice
  - PEP 357: El método “__index__”
  - Otros cambios lingüísticos
  - Módulos nuevos, mejorados y eliminados
  - Cambios en la API de construcción y C

  - Adaptación a Python 2.5
  - Agradecimientos

- Qué hay de nuevo en Python 2.4

  - PEP 218: Objetos conjunto integrados
  - PEP 237: Unificando enteros largos y enteros
  - PEP 289: Expresiones generadoras
  - PEP 292: Sustituciones simples de cadenas de caracteres
  - PEP 318: Decoradores para funciones y métodos
  - PEP 322: Iteración inversa
  - PEP 324: Nuevo módulo de subproceso
  - PEP 327: Tipo de dato decimal
  - PEP 328: Importaciones multilínea
  - PEP 331: Conversiones locales-independientes números flotantes/cadenas de texto
  - Otros cambios en el lenguaje
  - Módulos nuevos, mejorados y obsoletos
  - Cambios en la API de Build y C
  - Portar a Python 2.4
  - Agradecimientos

- Qué hay de nuevo en Python 2.3

  - PEP 218: Un tipo de datos de conjunto estándar
  - PEP 255: Generadores simples
  - PEP 263: Codificación del código fuente
  - PEP 273: Importar módulos desde archivos ZIP
  - PEP 277: Soporte de nombres de archivo Unicode para Windows NT
  - PEP 278: Soporte universal de nuevas líneas
  - PEP 279: enumerate()
  - PEP 282: El paquete de registro
  - PEP 285: Un tipo booleano
  - PEP 293: Llamadas de retorno para el manejo de errores del códec
  - PEP 301: Índice de paquetes y metadatos para Distutils
  - PEP 302: Nuevos ganchos de importación
  - PEP 305: Archivos separados por comas
  - PEP 307: Mejoras en Pickle

  - Rebanadas ampliadas
  - Otros cambios en el lenguaje
  - Módulos nuevos, mejorados y obsoletos
  - Pymalloc: un asignador de objetos especializado
  - Cambios en la API de Build y C
  - Otros cambios y correcciones
  - Portar a Python 2.3
  - Agradecimientos

- Qué hay de nuevo en Python 2.2

  - Introducción
  - PEPs 252 y 253: Cambios de tipo y clase
  - PEP 234: Iteradores
  - PEP 255: Generadores simples
  - PEP 237: Unificación de enteros largos y enteros
  - PEP 238: Cambio del operador de división
  - Cambios en Unicode
  - PEP 227: Ámbitos anidados
  - Módulos nuevos y mejorados
  - Cambios y correcciones en el intérprete
  - Otros cambios y correcciones
  - Agradecimientos

- Qué hay de nuevo en Python 2.1

  - Introducción
  - PEP 227: Ámbitos anidados
  - PEP 236: Directivas __future__
  - PEP 207: Comparaciones Enriquecidas
  - PEP 230: Marco de advertencia
  - PEP 229: Sistema de construcción nuevo
  - PEP 205: Referencias débiles
  - PEP 232: Atributos de la función
  - PEP 235: Importación de módulos en plataformas que no distinguen entre mayúsculas y minúsculas
  - PEP 217: Gancho de pantalla interactivo
  - PEP 208: Nuevo modelo de coerción
  - PEP 241: Metadatos en paquetes de Python
  - Módulos nuevos y mejorados
  - Otros cambios y correcciones

  - Agradecimientos
- Qué hay de nuevo en Python 2.0
  - Introducción
  - ¿Qué pasa con Python 1.6?
  - Nuevo proceso de desarrollo
  - Unicode
  - Comprensión de listas
  - Asignación aumentada
  - Métodos de cadena de caracteres
  - Recogida de basura de los ciclos
  - Otros cambios en el núcleo
  - Adaptación a la versión 2.0
  - Extensión/Incorporación de cambios
  - Distutils: Facilitando la instalación de módulos
  - Módulos XML
  - Cambios en los módulos
  - Nuevos módulos
  - Mejoras en IDLE
  - Módulos eliminados y obsoletos
  - Agradecimientos

The «Changelog» is an HTML version of the file built [https://pypi.org/project/blurb/] from the contents of the Misc/NEWS.d [https://github.com/python/cpython/tree/3.14/Misc/NEWS.d] directory tree, which contains *all* nontrivial changes to Python for the current version.

- Registro de cambios
  - Python next
  - Python 3.14.4 final
  - Python 3.14.3 final
  - Python 3.14.2 final
  - Python 3.14.1 final
  - Python 3.14.0 final
  - Python 3.14.0 release candidate 3
  - Python 3.14.0 release candidate 2
  - Python 3.14.0 release candidate 1
  - Python 3.14.0 beta 4
  - Python 3.14.0 beta 3

- Python 3.14.0 beta 2
- Python 3.14.0 beta 1
- Python 3.14.0 alpha 7
- Python 3.14.0 alpha 6
- Python 3.14.0 alpha 5
- Python 3.14.0 alpha 4
- Python 3.14.0 alpha 3
- Python 3.14.0 alpha 2
- Python 3.14.0 alpha 1
- Python 3.13.0 beta 1
- Python 3.13.0 alpha 6
- Python 3.13.0 alpha 5
- Python 3.13.0 alpha 4
- Python 3.13.0 alpha 3
- Python 3.13.0 alpha 2
- Python 3.13.0 alpha 1
- Python 3.12.0 beta 1
- Python 3.12.0 alpha 7
- Python 3.12.0 alpha 6
- Python 3.12.0 alpha 5
- Python 3.12.0 alpha 4
- Python 3.12.0 alpha 3
- Python 3.12.0 alpha 2
- Python 3.12.0 alpha 1
- Python 3.11.0 beta 1
- Python 3.11.0 alpha 7
- Python 3.11.0 alpha 6
- Python 3.11.0 alpha 5
- Python 3.11.0 alpha 4
- Python 3.11.0 alpha 3
- Python 3.11.0 alpha 2
- Python 3.11.0 alpha 1
- Python 3.10.0 beta 1
- Python 3.10.0 alpha 7
- Python 3.10.0 alpha 6
- Python 3.10.0 alpha 5
- Python 3.10.0 alpha 4
- Python 3.10.0 alpha 3
- Python 3.10.0 alpha 2
- Python 3.10.0 alpha 1

- Python 3.9.0 beta 1
- Python 3.9.0 alpha 6
- Python 3.9.0 alpha 5
- Python 3.9.0 alpha 4
- Python 3.9.0 alpha 3
- Python 3.9.0 alpha 2
- Python 3.9.0 alpha 1
- Python 3.8.0 beta 1
- Python 3.8.0 alpha 4
- Python 3.8.0 alpha 3
- Python 3.8.0 alpha 2
- Python 3.8.0 alpha 1
- Python 3.7.0 final
- Python 3.7.0 release candidate 1
- Python 3.7.0 beta 5
- Python 3.7.0 beta 4
- Python 3.7.0 beta 3
- Python 3.7.0 beta 2
- Python 3.7.0 beta 1
- Python 3.7.0 alpha 4
- Python 3.7.0 alpha 3
- Python 3.7.0 alpha 2
- Python 3.7.0 alpha 1
- Python 3.6.6 final
- Python 3.6.6 release candidate 1
- Python 3.6.5 final
- Python 3.6.5 release candidate 1
- Python 3.6.4 final
- Python 3.6.4 release candidate 1
- Python 3.6.3 final
- Python 3.6.3 release candidate 1
- Python 3.6.2 final
- Python 3.6.2 release candidate 2
- Python 3.6.2 release candidate 1
- Python 3.6.1 final
- Python 3.6.1 release candidate 1
- Python 3.6.0 final
- Python 3.6.0 release candidate 2
- Python 3.6.0 release candidate 1
- Python 3.6.0 beta 4

- Python 3.6.0 beta 3
- Python 3.6.0 beta 2
- Python 3.6.0 beta 1
- Python 3.6.0 alpha 4
- Python 3.6.0 alpha 3
- Python 3.6.0 alpha 2
- Python 3.6.0 alpha 1
- Python 3.5.5 final
- Python 3.5.5 release candidate 1
- Python 3.5.4 final
- Python 3.5.4 release candidate 1
- Python 3.5.3 final
- Python 3.5.3 release candidate 1
- Python 3.5.2 final
- Python 3.5.2 release candidate 1
- Python 3.5.1 final
- Python 3.5.1 release candidate 1
- Python 3.5.0 final
- Python 3.5.0 release candidate 4
- Python 3.5.0 release candidate 3
- Python 3.5.0 release candidate 2
- Python 3.5.0 release candidate 1
- Python 3.5.0 beta 4
- Python 3.5.0 beta 3
- Python 3.5.0 beta 2
- Python 3.5.0 beta 1
- Python 3.5.0 alpha 4
- Python 3.5.0 alpha 3
- Python 3.5.0 alpha 2
- Python 3.5.0 alpha 1

What’s new in Python 3.14

Editors:

Adam Turner and Hugo van Kemenade

This article explains the new features in Python 3.14, compared to 3.13. Python 3.14 was released on 7 October 2025. For full details, see the changelog.

Ver también

PEP 745 [https://peps.python.org/pep-0745/] – Python 3.14 release schedule

Summary – Release highlights

Python 3.14 is the latest stable release of the Python programming language, with a mix of changes to the language, the implementation, and the standard library. The biggest changes include template string literals, deferred evaluation of annotations, and support for subinterpreters in the standard library.

The library changes include significantly improved capabilities for introspection in asyncio, support for Zstandard via a new **`compression.zstd`** module, syntax highlighting in the REPL, as well as the usual deprecations and removals, and improvements in user-friendliness and correctness.

This article doesn’t attempt to provide a complete specification of all new features, but instead gives a convenient overview. For full details refer to the documentation, such as the Library Reference and Language Reference. To understand the complete implementation and design rationale for a change, refer to the PEP for a particular new feature; but note that PEPs usually are not kept up-to-date once a feature has been fully implemented. See Porting to Python 3.14 for guidance on upgrading from earlier versions of Python.

Interpreter improvements:

- **PEP 649** [https://peps.python.org/pep-0649/] and **PEP 749** [https://peps.python.org/pep-0749/]: Deferred evaluation of annotations
- **PEP 734** [https://peps.python.org/pep-0734/]: Multiple interpreters in the standard library
- **PEP 750** [https://peps.python.org/pep-0750/]: Template strings
- **PEP 758** [https://peps.python.org/pep-0758/]: Allow except and except* expressions without brackets
- **PEP 765** [https://peps.python.org/pep-0765/]: Control flow in finally blocks
- **PEP 768** [https://peps.python.org/pep-0768/]: Safe external debugger interface for CPython
- A new type of interpreter
- Free-threaded mode improvements
- Improved error messages
- Incremental garbage collection

Significant improvements in the standard library:

- **PEP 784** [https://peps.python.org/pep-0784/]: Zstandard support in the standard library
- Asyncio introspection capabilities
- Concurrent safe warnings control
- Syntax highlighting in the default interactive shell, and color output in several standard library CLIs

C API improvements:

- **PEP 741** [https://peps.python.org/pep-0741/]: Python configuration C API

Platform support:

- **PEP 776** [https://peps.python.org/pep-0776/]: Emscripten is now an officially supported platform, at **tier 3** [https://peps.python.org/pep-0011/#tier-3].

Release changes:

- **PEP 779** [https://peps.python.org/pep-0779/]: Free-threaded Python is officially supported
- **PEP 761** [https://peps.python.org/pep-0761/]: PGP signatures have been discontinued for official releases
- Windows and macOS binary releases now support the experimental just-in-time compiler
- Binary releases for Android are now provided

New features

PEP 649 [https://peps.python.org/pep-0649/] & PEP 749 [https://peps.python.org/pep-0749/]: Deferred evaluation of annotations

The annotations on functions, classes, and modules are no longer evaluated eagerly. Instead, annotations are stored in special-purpose annotate functions and evaluated only when necessary (except if `from __future__ import annotations` is used).

This change is designed to improve performance and usability of annotations in Python in most circumstances. The runtime cost for defining annotations is minimized, but it remains possible to introspect annotations at runtime. It is no longer necessary to enclose annotations in strings if they contain forward references.

The new **`annotationlib`** module provides tools for inspecting deferred annotations. Annotations may be evaluated in the **`VALUE`** format (which evaluates annotations to runtime values, similar to the behavior in earlier Python versions), the **`FORWARDREF`** format (which replaces undefined names with special markers), and the **`STRING`** format (which returns annotations as strings).

This example shows how these formats behave:

```
>>> from annotationlib import get_annotations, Format
>>> def func(arg: Undefined):
...     pass
>>> get_annotations(func, format=Format.VALUE)
```

```
Traceback (most recent call last):
  ...
NameError: name 'Undefined' is not defined
>>> get_annotations(func, format=Format.FORWARDREF)
{'arg': ForwardRef('Undefined', owner=<function func at
>>> get_annotations(func, format=Format.STRING)
{'arg': 'Undefined'}
```

The porting section contains guidance on changes that may be needed due to these changes, though in the majority of cases, code will continue working as-is.

(Contributed by Jelle Zijlstra in **PEP 749** [https://peps.python.org/pep-0749/] and gh-119180 [https://github.com/python/cpython/issues/119180]; **PEP 649** [https://peps.python.org/pep-0649/] was written by Larry Hastings.)

Ver también

PEP 649 [https://peps.python.org/pep-0649/]
Deferred Evaluation Of Annotations Using Descriptors

PEP 749 [https://peps.python.org/pep-0749/]
Implementing PEP 649

PEP 734 [https://peps.python.org/pep-0734/]: Multiple interpreters in the standard library

The CPython runtime supports running multiple copies of Python in the same process simultaneously and has done so for over 20 years. Each of these separate copies is called an “interpreter”. However, the feature had been available only through the C-API.

That limitation is removed in Python 3.14, with the new `concurrent.interpreters` module.

There are at least two notable reasons why using multiple interpreters has significant benefits:

- they support a new (to Python), human-friendly concurrency

model
- true multi-core parallelism

For some use cases, concurrency in software improves efficiency and can simplify design, at a high level. At the same time, implementing and maintaining all but the simplest concurrency is often a struggle for the human brain. That especially applies to plain threads (for example, `threading`), where all memory is shared between all threads.

With multiple isolated interpreters, you can take advantage of a class of concurrency models, like Communicating Sequential Processes (CSP) or the actor model, that have found success in other programming languages, like Smalltalk, Erlang, Haskell, and Go. Think of multiple interpreters as threads but with opt-in sharing.

Regarding multi-core parallelism: as of Python 3.12, interpreters are now sufficiently isolated from one another to be used in parallel (see **PEP 684** [https://peps.python.org/pep-0684/]). This unlocks a variety of CPU-intensive use cases for Python that were limited by the GIL.

Using multiple interpreters is similar in many ways to `multiprocessing`, in that they both provide isolated logical «processes» that can run in parallel, with no sharing by default. However, when using multiple interpreters, an application will use fewer system resources and will operate more efficiently (since it stays within the same process). Think of multiple interpreters as having the isolation of processes with the efficiency of threads.

While the feature has been around for decades, multiple interpreters have not been used widely, due to low awareness and the lack of a standard library module. Consequently, they currently have several notable limitations, which are expected to improve significantly now that the feature is going mainstream.

Current limitations:

- starting each interpreter has not been optimized yet
- each interpreter uses more memory than necessary (work continues on extensive internal sharing between interpreters)
- there aren’t many options *yet* for truly sharing objects or

other data between interpreters (other than `memoryview`)
- many third-party extension modules on PyPI are not yet compatible with multiple interpreters (all standard library extension modules *are* compatible)
- the approach to writing applications that use multiple isolated interpreters is mostly unfamiliar to Python users, for now

The impact of these limitations will depend on future CPython improvements, how interpreters are used, and what the community solves through PyPI packages. Depending on the use case, the limitations may not have much impact, so try it out!

Furthermore, future CPython releases will reduce or eliminate overhead and provide utilities that are less appropriate on PyPI. In the meantime, most of the limitations can also be addressed through extension modules, meaning PyPI packages can fill any gap for 3.14, and even back to 3.12 where interpreters were finally properly isolated and stopped sharing the GIL. Likewise, libraries on PyPI are expected to emerge for high-level abstractions on top of interpreters.

Regarding extension modules, work is in progress to update some PyPI projects, as well as tools like Cython, pybind11, nanobind, and PyO3. The steps for isolating an extension module are found at Aislamiento de módulos de extensión. Isolating a module has a lot of overlap with what is required to support free-threading, so the ongoing work in the community in that area will help accelerate support for multiple interpreters.

Also added in 3.14: concurrent.futures.InterpreterPoolExecutor.

(Contributed by Eric Snow in gh-134939 [https://github.com/python/cpython/issues/134939].)

Ver también

PEP 734 [https://peps.python.org/pep-0734/]

PEP 750 [https://peps.python.org/pep-0750/]: Template

string literals

Template strings are a new mechanism for custom string processing. They share the familiar syntax of f-strings but, unlike f-strings, return an object representing the static and interpolated parts of the string, instead of a simple **`str`**.

To write a t-string, use a `'t'` prefix instead of an `'f'`:

```
>>> variety = 'Stilton'
>>> template = t'Try some {variety} cheese!'
>>> type(template)
<class 'string.templatelib.Template'>
```

`Template` objects provide access to the static and interpolated (in curly braces) parts of a string *before* they are combined. Iterate over **`Template`** instances to access their parts in order:

```
>>> list(template)
['Try some ', Interpolation('Stilton', 'variety', None,
```

It's easy to write (or call) code to process **`Template`** instances. For example, here's a function that renders static parts lowercase and **`Interpolation`** instances uppercase:

```
from string.templatelib import Interpolation

def lower_upper(template):
    """Render static parts lowercase and interpolations
    parts = []
    for part in template:
        if isinstance(part, Interpolation):
            parts.append(str(part.value).upper())
        else:
            parts.append(part.lower())
    return ''.join(parts)

name = 'Wenslydale'
template = t'Mister {name}'
assert lower_upper(template) == 'mister WENSLYDALE'
```

Because **`Template`** instances distinguish between static strings and interpolations at runtime, they can be useful for sanitising user input. Writing a **`html()`** function that escapes user input in HTML is an exercise left to the reader! Template processing code can provide improved flexibility. For instance, a more advanced **`html()`** function could accept a **`dict`** of HTML attributes directly in the template:

```
attributes = {'src': 'limburger.jpg', 'alt': 'lovely che
template = t'<img {attributes}>'
assert html(template) == '<img src="limburger.jpg" alt="
```

Of course, template processing code does not need to return a string-like result. An even *more* advanced **`html()`** could return a custom type representing a DOM-like structure.

With t-strings in place, developers can write systems that sanitise SQL, make safe shell operations, improve logging, tackle modern ideas in web development (HTML, CSS, and so on), and implement lightweight custom business DSLs.

(Contributed by Jim Baker, Guido van Rossum, Paul Everitt, Koudai Aono, Lysandros Nikolaou, Dave Peck, Adam Turner, Jelle Zijlstra, Bénédikt Tran, and Pablo Galindo Salgado in gh-132661 [https://github.com/python/cpython/issues/132661].)

Ver también

PEP 750 [https://peps.python.org/pep-0750/].

PEP 768 [https://peps.python.org/pep-0768/]: Safe external debugger interface

Python 3.14 introduces a zero-overhead debugging interface that allows debuggers and profilers to safely attach to running Python processes without stopping or restarting them. This is a significant enhancement to Python's debugging capabilities, meaning that unsafe alternatives are no longer required.

The new interface provides safe execution points for attaching debugger code without modifying the interpreter's normal execution path or adding any overhead at runtime. Due to this, tools can now inspect and interact with Python applications in real-time, which is a crucial capability for high-availability systems and production environments.

For convenience, this interface is implemented in the `sys.remote_exec()` function. For example:

```
import sys
from tempfile import NamedTemporaryFile

with NamedTemporaryFile(mode='w', suffix='.py', delete=F
    script_path = f.name
    f.write(f'import my_debugger; my_debugger.connect({c

# Execute in process with PID 1234
print('Behold! An offering:')
sys.remote_exec(1234, script_path)
```

This function allows sending Python code to be executed in a target process at the next safe execution point. However, tool authors can also implement the protocol directly as described in the PEP, which details the underlying mechanisms used to safely attach to running processes.

The debugging interface has been carefully designed with security in mind and includes several mechanisms to control access:

- A `PYTHON_DISABLE_REMOTE_DEBUG` environment variable.
- A `-X disable-remote-debug` command-line option.
- A `--without-remote-debug` configure flag to completely disable the feature at build time.

(Contributed by Pablo Galindo Salgado, Matt Wozniski, and Ivona Stojanovic in gh-131591 [https://github.com/python/cpython/issues/131591].)

Ver también

PEP 768 [https://peps.python.org/pep-0768/].

A new type of interpreter

A new type of interpreter has been added to CPython. It uses tail calls between small C functions that implement individual Python opcodes, rather than one large C `case` statement. For certain newer compilers, this interpreter provides significantly better performance. Preliminary benchmarks suggest a geometric mean of 3-5% faster on the standard `pyperformance` benchmark suite, depending on platform and architecture. The baseline is Python 3.14 built with Clang 19, without this new interpreter.

This interpreter currently only works with Clang 19 and newer on x86-64 and AArch64 architectures. However, a future release of GCC is expected to support this as well.

This feature is opt-in for now. Enabling profile-guided optimization is highly recommended when using the new interpreter as it is the only configuration that has been tested and validated for improved performance. For further information, see **`--with-tail-call-interp`**.

Nota

This is not to be confused with tail call optimization [https://en.wikipedia.org/wiki/Tail_call] of Python functions, which is currently not implemented in CPython.

This new interpreter type is an internal implementation detail of the CPython interpreter. It doesn’t change the visible behavior of Python programs at all. It can improve their performance, but doesn’t change anything else.

(Contributed by Ken Jin in gh-128563 [https://github.com/python/cpython/issues/128563], with ideas on how to implement this in CPython by Mark Shannon, Garrett Gu, Haoran Xu, and Josh Haberman.)

Free-threaded mode improvements

CPython's free-threaded mode (**PEP 703** [https://peps.python.org/pep-0703/]), initially added in 3.13, has been significantly improved in Python 3.14. The implementation described in PEP 703 has been finished, including C API changes, and temporary workarounds in the interpreter were replaced with more permanent solutions. The specializing adaptive interpreter (**PEP 659** [https://peps.python.org/pep-0659/]) is now enabled in free-threaded mode, which along with many other optimizations greatly improves its performance. The performance penalty on single-threaded code in free-threaded mode is now roughly 5-10%, depending on the platform and C compiler used.

From Python 3.14, when compiling extension modules for the free-threaded build of CPython on Windows, the preprocessor variable `Py_GIL_DISABLED` now needs to be specified by the build backend, as it will no longer be determined automatically by the C compiler. For a running interpreter, the setting that was used at compile time can be found using `sysconfig.get_config_var()`.

The new `-X context_aware_warnings` flag controls if concurrent safe warnings control is enabled. The flag defaults to true for the free-threaded build and false for the GIL-enabled build.

A new `thread_inherit_context` flag has been added, which if enabled means that threads created with `threading.Thread` start with a copy of the `Context()` of the caller of `start()`. Most significantly, this makes the warning filtering context established by `catch_warnings` be «inherited» by threads (or asyncio tasks) started within that context. It also affects other modules that use context variables, such as the `decimal` context manager. This flag defaults to true for the free-threaded build and false for the GIL-enabled build.

(Contributed by Sam Gross, Matt Page, Neil Schemenauer, Thomas Wouters, Donghee Na, Kirill Podoprigora, Ken Jin, Itamar Oren, Brett Simmers, Dino Viehland, Nathan Goldbaum, Ralf Gommers, Lysandros Nikolaou, Kumar Aditya, Edgar Margffoy, and many others. Some of these contributors are employed by Meta, which has continued to provide significant engineering resources to support this project.)

Improved error messages

- The interpreter now provides helpful suggestions when it detects typos in Python keywords. When a word that closely resembles a Python keyword is encountered, the interpreter will suggest the correct keyword in the error message. This feature helps programmers quickly identify and fix common typing mistakes. For example:

  ```
  >>> whille True:
  ...     pass
  Traceback (most recent call last):
    File "<stdin>", line 1
      whille True:
      ^^^^^^
  SyntaxError: invalid syntax. Did you mean 'while'?
  ```

  While the feature focuses on the most common cases, some variations of misspellings may still result in regular syntax errors. (Contributed by Pablo Galindo in gh-132449 [https://github.com/python/cpython/issues/132449].)

- **`elif`** statements that follow an **`else`** block now have a specific error message. (Contributed by Steele Farnsworth in gh-129902 [https://github.com/python/cpython/issues/129902].)

  ```
  >>> if who == "me":
  ...     print("It's me!")
  ... else:
  ...     print("It's not me!")
  ... elif who is None:
  ...     print("Who is it?")
  File "<stdin>", line 5
    elif who is None:
    ^^^^
  SyntaxError: 'elif' block follows an 'else' block
  ```

- If a statement is passed to the Expresiones condicionales after `else`, or one of `pass`, `break`, or `continue` is passed before `if`, then the error message highlights where the `expression` is required. (Contributed by Sergey Miryanov in gh-129515 [https://github.com/python/cpython/issues/129515].)

```
>>> x = 1 if True else pass
Traceback (most recent call last):
  File "<string>", line 1
    x = 1 if True else pass
                       ^^^^
SyntaxError: expected expression after 'else', but

>>> x = continue if True else break
Traceback (most recent call last):
  File "<string>", line 1
    x = continue if True else break
        ^^^^^^^^
SyntaxError: expected expression before 'if', but s
```

- When incorrectly closed strings are detected, the error message suggests that the string may be intended to be part of the string. (Contributed by Pablo Galindo in gh-88535 [https://github.com/python/cpython/issues/88535].)

```
>>> "The interesting object "The important object"
Traceback (most recent call last):
SyntaxError: invalid syntax. Is this intended to be
```

- When strings have incompatible prefixes, the error now shows which prefixes are incompatible. (Contributed by Nikita Sobolev in gh-133197 [https://github.com/python/cpython/issues/133197].)

```
>>> ub'abc'
  File "<python-input-0>", line 1
    ub'abc'
    ^^
SyntaxError: 'u' and 'b' prefixes are incompatible
```

- Improved error messages when using `as` with incompatible targets in:
  - ○ Imports: `import ... as ...`
  - ○ From imports: `from ... import ... as ...`
  - ○ Except handlers: `except ... as ...`
  - ○ Pattern-match cases: `case ... as ...`

  (Contributed by Nikita Sobolev in gh-123539 [https://github.com/python/cpython/issues/123539], gh-123562 [https://github.com/python/cpython/issues/123562], and gh-123440 [https://github.com/python/cpython/issues/123440].)

- Improved error message when trying to add an instance of an unhashable type to a **`dict`** or **`set`**. (Contributed by CF Bolz-Tereick and Victor Stinner in gh-132828 [https://github.com/python/cpython/issues/132828].)

  ```
  >>> s = set()
  >>> s.add({'pages': 12, 'grade': 'A'})
  Traceback (most recent call last):
    File "<python-input-1>", line 1, in <module>
      s.add({'pages': 12, 'grade': 'A'})
      ~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  TypeError: cannot use 'dict' as a set element (unha
  >>> d = {}
  >>> l = [1, 2, 3]
  >>> d[l] = 12
  Traceback (most recent call last):
    File "<python-input-4>", line 1, in <module>
      d[l] = 12
      ~^^^
  TypeError: cannot use 'list' as a dict key (unhasha
  ```

- Improved error message when an object supporting the synchronous context manager protocol is entered using **`async with`** instead of **`with`**, and vice versa for the asynchronous context manager protocol. (Contributed by Bénédikt Tran in gh-128398 [https://github.com/python/cpython/issues/128398].)

PEP 784 [https://peps.python.org/pep-0784/]: Zstandard support in the standard library

The new `compression` package contains modules `compression.lzma`, `compression.bz2`, `compression.gzip` and `compression.zlib` which re-export the `lzma`, `bz2`, `gzip` and `zlib` modules respectively. The new import names under `compression` are the preferred names for importing these compression modules from Python 3.14. However, the existing modules names have not been deprecated. Any deprecation or removal of the existing compression modules will occur no sooner than five years after the release of 3.14.

The new `compression.zstd` module provides compression and decompression APIs for the Zstandard format via bindings to Meta's zstd library [https://facebook.github.io/zstd/]. Zstandard is a widely adopted, highly efficient, and fast compression format. In addition to the APIs introduced in `compression.zstd`, support for reading and writing Zstandard compressed archives has been added to the `tarfile`, `zipfile`, and `shutil` modules.

Here's an example of using the new module to compress some data:

```
from compression import zstd
import math

data = str(math.pi).encode() * 20
compressed = zstd.compress(data)
ratio = len(compressed) / len(data)
print(f"Achieved compression ratio of {ratio}")
```

As can be seen, the API is similar to the APIs of the `lzma` and `bz2` modules.

(Contributed by Emma Harper Smith, Adam Turner, Gregory P. Smith, Tomas Roun, Victor Stinner, and Rogdham in gh-132983 [https://github.com/python/cpython/issues/132983].)

Ver también

PEP 784 [https://peps.python.org/pep-0784/].

Asyncio introspection capabilities

Added a new command-line interface to inspect running Python processes using asynchronous tasks, available via `python -m asyncio ps PID` or `python -m asyncio pstree PID`.

The `ps` subcommand inspects the given process ID (PID) and displays information about currently running asyncio tasks. It outputs a task table: a flat listing of all tasks, their names, their coroutine stacks, and which tasks are awaiting them.

The `pstree` subcommand fetches the same information, but instead renders a visual async call tree, showing coroutine relationships in a hierarchical format. This command is particularly useful for debugging long-running or stuck asynchronous programs. It can help developers quickly identify where a program is blocked, what tasks are pending, and how coroutines are chained together.

For example given this code:

```
import asyncio

async def play_track(track):
    await asyncio.sleep(5)
    print(f'♪ Finished: {track}')

async def play_album(name, tracks):
    async with asyncio.TaskGroup() as tg:
        for track in tracks:
            tg.create_task(play_track(track), name=track

async def main():
    async with asyncio.TaskGroup() as tg:
        tg.create_task(
          play_album('Sundowning', ['TNDNBTG', 'Levitate
          name='Sundowning')
        tg.create_task(
            play_album('TMBTE', ['DYWTYLM', 'Aqua Regia'])
            name='TMBTE')

if __name__ == '__main__':
    asyncio.run(main())
```

Executing the new tool on the running process will yield a table like this:

```
python -m asyncio ps 12345

tid        task id              task name            cor
--------------------------------------------------------
1935500    0x7fc930c18050       Task-1               Tas
1935500    0x7fc930c18230       Sundowning           Tas
1935500    0x7fc93173fa50       TMBTE                Tas
1935500    0x7fc93173fdf0       TNDNBTG              sle
1935500    0x7fc930d32510       Levitate             sle
1935500    0x7fc930d32890       DYWTYLM              sle
1935500    0x7fc93161ec30       Aqua Regia           sle
```

or a tree like this:

```
python -m asyncio pstree 12345

    (T) Task-1
         main example.py:13
             TaskGroup.__aexit__ Lib/asyncio/taskgroups.
                 TaskGroup._aexit Lib/asyncio/taskgroups
                    (T) Sundowning
                         album example.py:8
                             TaskGroup.__aexit__ Lib/asy
                                 TaskGroup._aexit Lib/as
                                    (T) TNDNBTG
                                         play example.py
                                             sleep Lib/a
                                    (T) Levitate
                                         play example.py
                                             sleep Lib/a
                    (T) TMBTE
                         album example.py:8
                             TaskGroup.__aexit__ Lib/asy
                                 TaskGroup._aexit Lib/as
                                    (T) DYWTYLM
                                         play example.py
                                             sleep Lib/a
                                    (T) Aqua Regia
                                         play example.py
                                             sleep Lib/a
```

If a cycle is detected in the async await graph (which could indicate a programming issue), the tool raises an error and lists the cycle paths that prevent tree construction:

```
python -m asyncio pstree 12345

ERROR: await-graph contains cycles - cannot print a tree

cycle: Task-2 → Task-3 → Task-2
```

(Contributed by Pablo Galindo, Łukasz Langa, Yury Selivanov, and Marta Gomez Macias in gh-91048 [https://github.com/python/cpython/issues/91048].)

Concurrent safe warnings control

The **`warnings.catch_warnings`** context manager will now optionally use a context variable for warning filters. This is enabled by setting the **`context_aware_warnings`** flag, either with the `-X` command-line option or an environment variable. This gives predictable warnings control when using **`catch_warnings`** combined with multiple threads or asynchronous tasks. The flag defaults to true for the free-threaded build and false for the GIL-enabled build.

(Contributed by Neil Schemenauer and Kumar Aditya in gh-130010 [https://github.com/python/cpython/issues/130010].)

Other language changes

- All Windows code pages are now supported as “cpXXX” codecs on Windows. (Contributed by Serhiy Storchaka in gh-123803 [https://github.com/python/cpython/issues/123803].)
- Implement mixed-mode arithmetic rules combining real and complex numbers as specified by the C standard since C99. (Contributed by Sergey B Kirpichev in gh-69639 [https://github.com/python/cpython/issues/69639].)
- More syntax errors are now detected regardless of optimisation and the `-O` command-line option. This includes writes to `__debug__`, incorrect use of `await`, and asynchronous comprehensions outside asynchronous functions. For example, `python -O -c 'assert (__debug__ := 1)'` or `python -O -c 'assert await 1'` now produce `SyntaxError`s. (Contributed by Irit Katriel and Jelle Zijlstra in gh-122245 [https://github.com/python/cpython/issues/122245] & gh-121637 [https://github.com/python/cpython/issues/121637].)
- When subclassing a pure C type, the C slots for the new type are no longer replaced with a wrapped version on class creation if they are not explicitly overridden in the subclass. (Contributed by Tomasz Pytel in gh-132284 [https://github.com/python/cpython/issues/132284].)

Built-ins

- The `bytes.fromhex()` and `bytearray.fromhex()` methods now accept ASCII `bytes` and bytes-like objects. (Contributed by Daniel Pope in gh-129349 [https://github.com/python/cpython/issues/129349].)
- Add class methods `float.from_number()` and `complex.from_number()` to convert a number to `float` or `complex` type correspondingly. They raise a `TypeError` if the argument is not a real number. (Contributed by Serhiy Storchaka in gh-84978 [https://github.com/python/cpython/issues/84978].)
- Support underscore and comma as thousands separators in the fractional part for floating-point presentation types of the

new-style string formatting (with `format()` or f-strings). (Contributed by Sergey B Kirpichev in gh-87790 [https://github.com/python/cpython/issues/87790].)

- The `int()` function no longer delegates to `__trunc__()`. Classes that want to support conversion to `int()` must implement either `__int__()` or `__index__()`. (Contributed by Mark Dickinson in gh-119743 [https://github.com/python/cpython/issues/119743].)
- The `map()` function now has an optional keyword-only *strict* flag like `zip()` to check that all the iterables are of equal length. (Contributed by Wannes Boeykens in gh-119793 [https://github.com/python/cpython/issues/119793].)
- The `memoryview` type now supports subscription, making it a generic type. (Contributed by Brian Schubert in gh-126012 [https://github.com/python/cpython/issues/126012].)
- Using `NotImplemented` in a boolean context will now raise a `TypeError`. This has raised a `DeprecationWarning` since Python 3.9. (Contributed by Jelle Zijlstra in gh-118767 [https://github.com/python/cpython/issues/118767].)
- Three-argument `pow()` now tries calling `__rpow__()` if necessary. Previously it was only called in two-argument `pow()` and the binary power operator. (Contributed by Serhiy Storchaka in gh-130104 [https://github.com/python/cpython/issues/130104].)
- `super` objects are now `copyable` and `pickleable`. (Contributed by Serhiy Storchaka in gh-125767 [https://github.com/python/cpython/issues/125767].)

Command line and environment

- The import time flag can now track modules that are already loaded ("cached"), via the new `-X importtime=2`. When such a module is imported, the `self` and `cumulative` times are replaced by the string `cached`.

  Values above `2` for `-X importtime` are now reserved for future use.

  (Contributed by Noah Kim and Adam Turner in gh-118655 [https://github.com/python/cpython/issues/118655].)

- The command-line option `-c` now automatically dedents its code argument before execution. The auto-dedentation behavior mirrors `textwrap.dedent()`. (Contributed by Jon Crall and Steven Sun in gh-103998 [https://github.com/python/cpython/issues/103998].)

- `-J` is no longer a reserved flag for Jython [https://www.jython.org/], and now has no special meaning. (Contributed by Adam Turner in gh-133336 [https://github.com/python/cpython/issues/133336].)

PEP 758: Allow `except` and `except*` expressions without brackets

The `except` and `except*` expressions now allow brackets to be omitted when there are multiple exception types and the `as` clause is not used. For example:

```
try:
    connect_to_server()
except TimeoutError, ConnectionRefusedError:
    print('The network has ceased to be!')
```

(Contributed by Pablo Galindo and Brett Cannon in **PEP 758** [https://peps.python.org/pep-0758/] and gh-131831 [https://github.com/python/cpython/issues/131831].)

PEP 765: Control flow in `finally` blocks

The compiler now emits a `SyntaxWarning` when a `return`, `break`, or `continue` statement have the effect of leaving a `finally` block. This change is specified in **PEP 765** [https://peps.python.org/pep-0765/].

In situations where this change is inconvenient (such as those where the warnings are redundant due to code linting), the warning filter can be used to turn off all syntax warnings by adding `ignore::SyntaxWarning` as a filter. This can be specified in combination with a filter that converts other warnings to errors (for example, passing `-Werror -Wignore::SyntaxWarning` as CLI options, or setting `PYTHONWARNINGS=error,ignore::SyntaxWarning`).

Note that applying such a filter at runtime using the **`warnings`** module will only suppress the warning in code that is compiled *after* the filter is adjusted. Code that is compiled prior to the filter adjustment (for example, when a module is imported) will still emit the syntax warning.

(Contributed by Irit Katriel in gh-130080 [https://github.com/python/cpython/issues/130080].)

Incremental garbage collection

The cycle garbage collector is now incremental. This means that maximum pause times are reduced by an order of magnitude or more for larger heaps.

There are now only two generations: young and old. When **`gc.collect()`** is not called directly, the GC is invoked a little less frequently. When invoked, it collects the young generation and an increment of the old generation, instead of collecting one or more generations.

The behavior of **`gc.collect()`** changes slightly:

- `gc.collect(1)`: Performs an increment of garbage collection, rather than collecting generation 1.
- Other calls to **`gc.collect()`** are unchanged.

(Contributed by Mark Shannon in gh-108362 [https://github.com/python/cpython/issues/108362].)

Default interactive shell

- The default interactive shell now highlights Python syntax. The feature is enabled by default, save if **`PYTHON_BASIC_REPL`** or any other environment variable that disables colour is set. See Controlling color for details.

  The default color theme for syntax highlighting strives for good contrast and exclusively uses the 4-bit VGA standard ANSI color codes for maximum compatibility. The theme can be customized using an experimental API **`_colorize.set_theme()`**. This can be called interactively or in the **`PYTHONSTARTUP`** script. Note that this function has no stability guarantees, and may change or be removed.

(Contributed by Łukasz Langa in gh-131507 [https://github.com/python/cpython/issues/131507].)

- The default interactive shell now supports import auto-completion. This means that typing `import co` and pressing `<Tab>` will suggest modules starting with `co`. Similarly, typing `from concurrent import i` will suggest submodules of `concurrent` starting with `i`. Note that autocompletion of module attributes is not currently supported. (Contributed by Tomas Roun in gh-69605 [https://github.com/python/cpython/issues/69605].)

New modules

- **`annotationlib`**: For introspecting annotations. See PEP 749 for more details. (Contributed by Jelle Zijlstra in gh-119180 [https://github.com/python/cpython/issues/119180].)
- **`compression`** (including **`compression.zstd`**): A package for compression-related modules, including a new module to support the Zstandard compression format. See PEP 784 for more details. (Contributed by Emma Harper Smith, Adam Turner, Gregory P. Smith, Tomas Roun, Victor Stinner, and Rogdham in gh-132983 [https://github.com/python/cpython/issues/132983].)
- **`concurrent.interpreters`**: Support for multiple interpreters in the standard library. See PEP 734 for more details. (Contributed by Eric Snow in gh-134939 [https://github.com/python/cpython/issues/134939].)
- **`string.templatelib`**: Support for template string literals (t-strings). See PEP 750 for more details. (Contributed by Jim Baker, Guido van Rossum, Paul Everitt, Koudai Aono, Lysandros Nikolaou, Dave Peck, Adam Turner, Jelle Zijlstra, Bénédikt Tran, and Pablo Galindo Salgado in gh-132661

[https://github.com/python/cpython/issues/132661].)

Improved modules

argparse

- The default value of the program name for `argparse.ArgumentParser` now reflects the way the Python interpreter was instructed to find the `__main__` module code. (Contributed by Serhiy Storchaka and Alyssa Coghlan in gh-66436 [https://github.com/python/cpython/issues/66436].)
- Introduced the optional *suggest_on_error* parameter to `argparse.ArgumentParser`, enabling suggestions for argument choices and subparser names if mistyped by the user. (Contributed by Savannah Ostrowski in gh-124456 [https://github.com/python/cpython/issues/124456].)
- Enable color for help text, which can be disabled with the optional *color* parameter to `argparse.ArgumentParser`. This can also be controlled by environment variables. (Contributed by Hugo van Kemenade in gh-130645 [https://github.com/python/cpython/issues/130645].)

ast

- Add `compare()`, a function for comparing two ASTs. (Contributed by Batuhan Taskaya and Jeremy Hylton in gh-60191 [https://github.com/python/cpython/issues/60191].)
- Add support for `copy.replace()` for AST nodes. (Contributed by Bénédikt Tran in gh-121141 [https://github.com/python/cpython/issues/121141].)
- Docstrings are now removed from an optimized AST in optimization level 2. (Contributed by Irit Katriel in gh-123958 [https://github.com/python/cpython/issues/123958].)
- The `repr()` output for AST nodes now includes more information. (Contributed by Tomas Roun in gh-116022 [https://github.com/python/cpython/issues/116022].)
- When called with an AST as input, the `parse()` function now always verifies that the root node type is appropriate.

(Contributed by Irit Katriel in gh-130139 [https://github.com/python/cpython/issues/130139].)

- Add new options to the command-line interface: `--feature-version`, `--optimize`, and `--show-empty`. (Contributed by Semyon Moroz in gh-133367 [https://github.com/python/cpython/issues/133367].)

asyncio

- The function and methods named **`create_task()`** now take an arbitrary list of keyword arguments. All keyword arguments are passed to the `Task` constructor or the custom task factory. (See `set_task_factory()` for details.) The `name` and `context` keyword arguments are no longer special; the name should now be set using the `name` keyword argument of the factory, and `context` may be `None`.

  This affects the following function and methods: `asyncio.create_task()`, `asyncio.loop.create_task()`, `asyncio.TaskGroup.create_task()`.

  (Contributed by Thomas Grainger in gh-128307 [https://github.com/python/cpython/issues/128307].)

- There are two new utility functions for introspecting and printing a program's call graph: `capture_call_graph()` and `print_call_graph()`. See Asyncio introspection capabilities for more details. (Contributed by Yury Selivanov, Pablo Galindo Salgado, and Łukasz Langa in gh-91048 [https://github.com/python/cpython/issues/91048].)

calendar

- By default, today's date is highlighted in color in `calendar`'s command-line text output. This can be controlled by environment variables. (Contributed by Hugo van Kemenade in gh-128317 [https://github.com/python/cpython/issues/128317].)

concurrent.futures

- Add a new executor class, `InterpreterPoolExecutor`, which exposes multiple Python interpreters in the same process ("subinterpreters") to Python code. This uses a pool of independent Python interpreters to execute calls asynchronously.

  This is separate from the new `interpreters` module introduced by PEP 734. (Contributed by Eric Snow in gh-124548 [https://github.com/python/cpython/issues/124548].)

- On Unix platforms other than macOS, "forkserver" is now the default start method for `ProcessPoolExecutor` (replacing "fork"). This change does not affect Windows or macOS, where "spawn" remains the default start method.

  If the threading incompatible *fork* method is required, you must explicitly request it by supplying a multiprocessing context *mp_context* to `ProcessPoolExecutor`.

  See forkserver restrictions for information and differences with the *fork* method and how this change may affect existing code with mutable global shared variables and/or shared objects that can not be automatically `pickled`.

  (Contributed by Gregory P. Smith in gh-84559 [https://github.com/python/cpython/issues/84559].)

- Add two new methods to `ProcessPoolExecutor`, `terminate_workers()` and `kill_workers()`, as ways to terminate or kill all living worker processes in the given pool. (Contributed by Charles Machalow in gh-130849 [https://github.com/python/cpython/issues/130849].)

- Add the optional *buffersize* parameter to `Executor.map` to limit the number of submitted tasks whose results have not yet been yielded. If the buffer is full, iteration over the *iterables* pauses until a result is yielded from the buffer. (Contributed by Enzo Bonnal and Josh Rosenberg in gh-74028 [https://github.com/python/cpython/issues/74028].)

configparser

- `configparser` will no longer write config files it cannot read, to improve security. Attempting to `write()` keys containing delimiters or beginning with the section header pattern will raise an `InvalidWriteError`. (Contributed by Jacob Lincoln in gh-129270 [https://github.com/python/cpython/issues/129270].)

contextvars

- Support the context manager protocol for `Token` objects. (Contributed by Andrew Svetlov in gh-129889 [https://github.com/python/cpython/issues/129889].)

ctypes

- The layout of bit fields in `Structure` and `Union` objects is now a closer match to platform defaults (GCC/Clang or MSVC). In particular, fields no longer overlap. (Contributed by Matthias Görgens in gh-97702 [https://github.com/python/cpython/issues/97702].)
- The `Structure._layout_` class attribute can now be set to help match a non-default ABI. (Contributed by Petr Viktorin in gh-97702 [https://github.com/python/cpython/issues/97702].)
- The class of `Structure`/`Union` field descriptors is now available as `CField`, and has new attributes to aid debugging and introspection. (Contributed by Petr Viktorin in gh-128715 [https://github.com/python/cpython/issues/128715].)
- On Windows, the `COMError` exception is now public. (Contributed by Jun Komoda in gh-126686 [https://github.com/python/cpython/issues/126686].)
- On Windows, the `CopyComPointer()` function is now public. (Contributed by Jun Komoda in gh-127275 [https://github.com/python/cpython/issues/127275].)
- Add `memoryview_at()`, a function to create a `memoryview` object that refers to the supplied pointer and length. This works like `ctypes.string_at()` except it avoids a buffer copy, and is typically useful when implementing pure Python callback functions that are passed

dynamically-sized buffers. (Contributed by Rian Hunter in gh-112018 [https://github.com/python/cpython/issues/112018].)

- Complex types, **`c_float_complex`**, **`c_double_complex`**, and **`c_longdouble_complex`**, are now available if both the compiler and the `libffi` library support complex C types. (Contributed by Sergey B Kirpichev in gh-61103 [https://github.com/python/cpython/issues/61103].)
- Add **`ctypes.util.dllist()`** for listing the shared libraries loaded by the current process. (Contributed by Brian Ward in gh-119349 [https://github.com/python/cpython/issues/119349].)
- Move **`ctypes.POINTER()`** types cache from a global internal cache (`_pointer_type_cache`) to the **`_CData.__pointer_type__`** attribute of the corresponding **`ctypes`** types. This will stop the cache from growing without limits in some situations. (Contributed by Sergey Miryanov in gh-100926 [https://github.com/python/cpython/issues/100926].)
- The **`py_object`** type now supports subscription, making it a generic type. (Contributed by Brian Schubert in gh-132168 [https://github.com/python/cpython/issues/132168].)
- **`ctypes`** now supports free-threading builds. (Contributed by Kumar Aditya and Peter Bierma in gh-127945 [https://github.com/python/cpython/issues/127945].)

curses

- Add the **`assume_default_colors()`** function, a refinement of the **`use_default_colors()`** function which allows changing the color pair `0`. (Contributed by Serhiy Storchaka in gh-133139 [https://github.com/python/cpython/issues/133139].)

datetime

- Add the **`strptime()`** method to the **`datetime.date`** and **`datetime.time`** classes. (Contributed by Wannes Boeykens in gh-41431 [https://github.com/python/cpython/issues/41431].)

decimal

- Add `Decimal.from_number()` as an alternative constructor for `Decimal`. (Contributed by Serhiy Storchaka in gh-121798 [https://github.com/python/cpython/issues/121798].)
- Expose `IEEEContext()` to support creation of contexts corresponding to the IEEE 754 (2008) decimal interchange formats. (Contributed by Sergey B Kirpichev in gh-53032 [https://github.com/python/cpython/issues/53032].)

difflib

- Comparison pages with highlighted changes generated by the `HtmlDiff` class now support "dark mode". (Contributed by Jiahao Li in gh-129939 [https://github.com/python/cpython/issues/129939].)

dis

- Add support for rendering full source location information of `instructions`, rather than only the line number. This feature is added to the following interfaces via the *show_positions* keyword argument:

    ○ `dis.Bytecode`
    ○ `dis.dis()`
    ○ `dis.distb()`
    ○ `dis.disassemble()`

  This feature is also exposed via `dis --show-positions`. (Contributed by Bénédikt Tran in gh-123165 [https://github.com/python/cpython/issues/123165].)

- Add the `dis --specialized` command-line option to show specialized bytecode. (Contributed by Bénédikt Tran in gh-127413 [https://github.com/python/cpython/issues/127413].)

errno

- Add the `EHWPOISON` error code constant. (Contributed by James Roy in gh-126585 [https://github.com/python/cpython/issues/126585].)

faulthandler

- Add support for printing the C stack trace on systems that support it via the new `dump_c_stack()` function or via the *c_stack* argument in `faulthandler.enable()`. (Contributed by Peter Bierma in gh-127604 [https://github.com/python/cpython/issues/127604].)

fnmatch

- Add `filterfalse()`, a function to reject names matching a given pattern. (Contributed by Bénédikt Tran in gh-74598 [https://github.com/python/cpython/issues/74598].)

fractions

- A `Fraction` object may now be constructed from any object with the `as_integer_ratio()` method. (Contributed by Serhiy Storchaka in gh-82017 [https://github.com/python/cpython/issues/82017].)
- Add `Fraction.from_number()` as an alternative constructor for `Fraction`. (Contributed by Serhiy Storchaka in gh-121797 [https://github.com/python/cpython/issues/121797].)

functools

- Add the `Placeholder` sentinel. This may be used with the `partial()` or `partialmethod()` functions to reserve a place for positional arguments in the returned partial object. (Contributed by Dominykas Grigonis in gh-119127 [https://github.com/python/cpython/issues/119127].)
- Allow the *initial* parameter of `reduce()` to be passed as a keyword argument. (Contributed by Sayandip Dutta in gh-125916 [https://github.com/python/cpython/issues/125916].)

getopt

- Add support for options with optional arguments. (Contributed by Serhiy Storchaka in gh-126374 [https://

github.com/python/cpython/issues/126374].)

- Add support for returning intermixed options and non-option arguments in order. (Contributed by Serhiy Storchaka in gh-126390 [https://github.com/python/cpython/issues/126390].)

getpass

- Support keyboard feedback in the `getpass()` function via the keyword-only optional argument *echo_char*. Placeholder characters are rendered whenever a character is entered, and removed when a character is deleted. (Contributed by Semyon Moroz in gh-77065 [https://github.com/python/cpython/issues/77065].)

graphlib

- Allow `TopologicalSorter.prepare()` to be called more than once as long as sorting has not started. (Contributed by Daniel Pope in gh-130914 [https://github.com/python/cpython/issues/130914].)

heapq

- The **heapq** module has improved support for working with max-heaps, via the following new functions:
    - ○ `heapify_max()`
    - ○ `heappush_max()`
    - ○ `heappop_max()`
    - ○ `heapreplace_max()`
    - ○ `heappushpop_max()`

hmac

- Add a built-in implementation for HMAC (**RFC 2104** [https://datatracker.ietf.org/doc/html/rfc2104.html]) using formally verified code from the HACL* [https://github.com/hacl-star/hacl-star/] project. This implementation is used as a fallback when the OpenSSL implementation of HMAC is not available. (Contributed by Bénédikt Tran in gh-99108 [https://github.com/

python/cpython/issues/99108].)

http

- Directory lists and error pages generated by the **`http.server`** module allow the browser to apply its default dark mode. (Contributed by Yorik Hansen in gh-123430 [https://github.com/python/cpython/issues/123430].)

- The **`http.server`** module now supports serving over HTTPS using the **`http.server.HTTPSServer`** class. This functionality is exposed by the command-line interface (`python -m http.server`) through the following options:

  ○ **`--tls-cert <path>`**: Path to the TLS certificate file.
  ○ **`--tls-key <path>`**: Optional path to the private key file.
  ○ **`--tls-password-file <path>`**: Optional path to the password file for the private key.

  (Contributed by Semyon Moroz in gh-85162 [https://github.com/python/cpython/issues/85162].)

imaplib

- Add **`IMAP4.idle()`**, implementing the IMAP4 `IDLE` command as defined in **RFC 2177** [https://datatracker.ietf.org/doc/html/rfc2177.html]. (Contributed by Forest in gh-55454 [https://github.com/python/cpython/issues/55454].)

inspect

- **`signature()`** takes a new argument *annotation_format* to control the **`annotationlib.Format`** used for representing annotations. (Contributed by Jelle Zijlstra in gh-101552 [https://github.com/python/cpython/issues/101552].)
- **`Signature.format()`** takes a new argument *unquote_annotations*. If true, string annotations are displayed without surrounding quotes. (Contributed by Jelle Zijlstra in gh-101552 [https://github.com/python/cpython/issues/101552].)

- Add function `ispackage()` to determine whether an object is a package or not. (Contributed by Zhikang Yan in gh-125634 [https://github.com/python/cpython/issues/125634].)

io

- Reading text from a non-blocking stream with `read` may now raise a `BlockingIOError` if the operation cannot immediately return bytes. (Contributed by Giovanni Siragusa in gh-109523 [https://github.com/python/cpython/issues/109523].)
- Add the `Reader` and `Writer` protocols as simpler alternatives to the pseudo-protocols `typing.IO`, `typing.TextIO`, and `typing.BinaryIO`. (Contributed by Sebastian Rittau in gh-127648 [https://github.com/python/cpython/issues/127648].)

json

- Add exception notes for JSON serialization errors that allow identifying the source of the error. (Contributed by Serhiy Storchaka in gh-122163 [https://github.com/python/cpython/issues/122163].)
- Allow using the `json` module as a script using the `-m` switch: **python -m json**. This is now preferred to **python -m json.tool**, which is soft deprecated. See the JSON command-line interface documentation. (Contributed by Trey Hunner in gh-122873 [https://github.com/python/cpython/issues/122873].)
- By default, the output of the JSON command-line interface is highlighted in color. This can be controlled by environment variables. (Contributed by Tomas Roun in gh-131952 [https://github.com/python/cpython/issues/131952].)

linecache

- `getline()` can now retrieve source code for frozen modules. (Contributed by Tian Gao in gh-131638 [https://github.com/python/cpython/issues/131638].)

logging.handlers

- **`QueueListener`** objects now support the context manager protocol. (Contributed by Charles Machalow in gh-132106 [https://github.com/python/cpython/issues/132106].)
- **`QueueListener.start`** now raises a **`RuntimeError`** if the listener is already started. (Contributed by Charles Machalow in gh-132106 [https://github.com/python/cpython/issues/132106].)

math

- Added more detailed error messages for domain errors in the module. (Contributed by Charlie Zhao and Sergey B Kirpichev in gh-101410 [https://github.com/python/cpython/issues/101410].)

mimetypes

- Add a public command-line for the module, invoked via **python -m mimetypes**. (Contributed by Oleg Iarygin and Hugo van Kemenade in gh-93096 [https://github.com/python/cpython/issues/93096].)

- Add several new MIME types based on RFCs and common usage:

  Microsoft and RFC 8081 [https://datatracker.ietf.org/doc/html/rfc8081.html] MIME types for fonts

  ○ Embedded OpenType: `application/vnd.ms-fontobject`
  ○ OpenType Layout (OTF) `font/otf`
  ○ TrueType: `font/ttf`
  ○ WOFF 1.0 `font/woff`
  ○ WOFF 2.0 `font/woff2`

  RFC 9559 [https://datatracker.ietf.org/doc/html/rfc9559.html] MIME types for Matroska audiovisual data container structures

  ○ audio with no video: `audio/matroska` (`.mka`)

- ○ video: `video/matroska` (`.mkv`)
- ○ stereoscopic video: `video/matroska-3d` (`.mk3d`)

Images with RFCs

- ○ **RFC 1494** [https://datatracker.ietf.org/doc/html/rfc1494.html]: CCITT Group 3 (`.g3`)
- ○ **RFC 3362** [https://datatracker.ietf.org/doc/html/rfc3362.html]: Real-time Facsimile, T.38 (`.t38`)
- ○ **RFC 3745** [https://datatracker.ietf.org/doc/html/rfc3745.html]: JPEG 2000 (`.jp2`), extension (`.jpx`) and compound (`.jpm`)
- ○ **RFC 3950** [https://datatracker.ietf.org/doc/html/rfc3950.html]: Tag Image File Format Fax eXtended, TIFF-FX (`.tfx`)
- ○ **RFC 4047** [https://datatracker.ietf.org/doc/html/rfc4047.html]: Flexible Image Transport System (`.fits`)
- ○ **RFC 7903** [https://datatracker.ietf.org/doc/html/rfc7903.html]: Enhanced Metafile (`.emf`) and Windows Metafile (`.wmf`)

Other MIME type additions and changes

- ○ **RFC 2361** [https://datatracker.ietf.org/doc/html/rfc2361.html]: Change type for `.avi` to `video/vnd.avi` and for `.wav` to `audio/vnd.wave`
- ○ **RFC 4337** [https://datatracker.ietf.org/doc/html/rfc4337.html]: Add MPEG-4 `audio/mp4` (`.m4a`)
- ○ **RFC 5334** [https://datatracker.ietf.org/doc/html/rfc5334.html]: Add Ogg media (`.oga`, `.ogg` and `.ogx`)
- ○ **RFC 6713** [https://datatracker.ietf.org/doc/html/rfc6713.html]: Add gzip `application/gzip` (`.gz`)
- ○ **RFC 9639** [https://datatracker.ietf.org/doc/html/rfc9639.html]: Add FLAC `audio/flac` (`.flac`)
- ○ **RFC 9512** [https://datatracker.ietf.org/doc/html/rfc9512.html] `application/yaml` MIME type for YAML files (`.yaml` and `.yml`)
- ○ Add 7z `application/x-7z-compressed` (`.7z`)
- ○ Add Android Package `application/`

    `vnd.android.package-archive` (`.apk`) when not strict
  - Add deb `application/x-debian-package` (`.deb`)
  - Add glTF binary `model/gltf-binary` (`.glb`)
  - Add glTF JSON/ASCII `model/gltf+json` (`.gltf`)
  - Add M4V `video/x-m4v` (`.m4v`)
  - Add PHP `application/x-httpd-php` (`.php`)
  - Add RAR `application/vnd.rar` (`.rar`)
  - Add RPM `application/x-rpm` (`.rpm`)
  - Add STL `model/stl` (`.stl`)
  - Add Windows Media Video `video/x-ms-wmv` (`.wmv`)
  - De facto: Add WebM `audio/webm` (`.weba`)
  - ECMA-376 [https://ecma-international.org/publications-and-standards/standards/ecma-376/]: Add `.docx`, `.pptx` and `.xlsx` types
  - OASIS [https://docs.oasis-open.org/office/v1.2/cs01/OpenDocument-v1.2-cs01-part1.html#Appendix_C]: Add OpenDocument `.odg`, `.odp`, `.ods` and `.odt` types
  - W3C [https://www.w3.org/TR/epub-33/#app-media-type]: Add EPUB `application/epub+zip` (`.epub`)

  (Contributed by Sahil Prajapati and Hugo van Kemenade in gh-84852 [https://github.com/python/cpython/issues/84852], by Sasha «Nelie» Chernykh and Hugo van Kemenade in gh-132056 [https://github.com/python/cpython/issues/132056], and by Hugo van Kemenade in gh-89416 [https://github.com/python/cpython/issues/89416], gh-85957 [https://github.com/python/cpython/issues/85957], and gh-129965 [https://github.com/python/cpython/issues/129965].)

multiprocessing

- On Unix platforms other than macOS, "forkserver" is now the default start method (replacing "fork"). This change does not affect Windows or macOS, where "spawn" remains the default start method.

  If the threading incompatible *fork* method is required, you must explicitly request it via a context from **`get_context()`** (preferred) or change the default via `set_start_method()`.

See forkserver restrictions for information and differences with the *fork* method and how this change may affect existing code with mutable global shared variables and/or shared objects that can not be automatically `pickled`.

(Contributed by Gregory P. Smith in gh-84559 [https://github.com/python/cpython/issues/84559].)

- `multiprocessing`'s `'forkserver'` start method now authenticates its control socket to avoid solely relying on filesystem permissions to restrict what other processes could cause the forkserver to spawn workers and run code. (Contributed by Gregory P. Smith for gh-97514 [https://github.com/python/cpython/issues/97514].)
- The multiprocessing proxy objects for *list* and *dict* types gain previously overlooked missing methods:
    - `clear()` and `copy()` for proxies of `list`
    - `fromkeys()`, `reversed(d)`, `d | {}`, `{} | d`, `d |= {'b': 2}` for proxies of `dict`

  (Contributed by Roy Hyunjin Han for gh-103134 [https://github.com/python/cpython/issues/103134].)
- Add support for shared `set` objects via `SyncManager.set()`. The `set()` in `Manager()` method is now available. (Contributed by Mingyu Park in gh-129949 [https://github.com/python/cpython/issues/129949].)
- Add the `interrupt()` to `multiprocessing.Process` objects, which terminates the child process by sending `SIGINT`. This enables `finally` clauses to print a stack trace for the terminated process. (Contributed by Artem Pulkin in gh-131913 [https://github.com/python/cpython/issues/131913].)

operator

- Add **`is_none()`** and **`is_not_none()`** as a pair of functions, such that `operator.is_none(obj)` is equivalent to `obj is None` and `operator.is_not_none(obj)` is equivalent to `obj is not None`. (Contributed by Raymond Hettinger and Nico Mexis in gh-115808 [https://github.com/python/cpython/issues/115808].)

os

- Add the **`reload_environ()`** function to update **`os.environ`** and **`os.environb`** with changes to the environment made by **`os.putenv()`**, by **`os.unsetenv()`**, or made outside Python in the same process. (Contributed by Victor Stinner in gh-120057 [https://github.com/python/cpython/issues/120057].)
- Add the **`SCHED_DEADLINE`** and **`SCHED_NORMAL`** constants to the **`os`** module. (Contributed by James Roy in gh-127688 [https://github.com/python/cpython/issues/127688].)
- Add the **`readinto()`** function to read into a buffer object from a file descriptor. (Contributed by Cody Maloney in gh-129205 [https://github.com/python/cpython/issues/129205].)

os.path

- The *strict* parameter to **`realpath()`** accepts a new value, **`ALLOW_MISSING`**. If used, errors other than **`FileNotFoundError`** will be re-raised; the resulting path can be missing but it will be free of symlinks. (Contributed by Petr Viktorin for **CVE 2025-4517** [https://www.cve.org/CVERecord?id=CVE-2025-4517].)

pathlib

- Add methods to **`pathlib.Path`** to recursively copy or move files and directories:
    - **`copy()`** copies a file or directory tree to a destination.
    - **`copy_into()`** copies *into* a destination directory.
    - **`move()`** moves a file or directory tree to a destination.

  - `move_into()` moves *into* a destination directory.

  (Contributed by Barney Gale in gh-73991 [https://github.com/python/cpython/issues/73991].)

- Add the `info` attribute, which stores an object implementing the new `pathlib.types.PathInfo` protocol. The object supports querying the file type and internally caching `stat()` results. Path objects generated by `iterdir()` are initialized with file type information gleaned from scanning the parent directory. (Contributed by Barney Gale in gh-125413 [https://github.com/python/cpython/issues/125413].)

pdb

- The `pdb` module now supports remote attaching to a running Python process using a new `-p PID` command-line option:

  ```
  python -m pdb -p 1234
  ```

  This will connect to the Python process with the given PID and allow you to debug it interactively. Notice that due to how the Python interpreter works attaching to a remote process that is blocked in a system call or waiting for I/O will only work once the next bytecode instruction is executed or when the process receives a signal.

  This feature uses PEP 768 and the new `sys.remote_exec()` function to attach to the remote process and send the PDB commands to it.

  (Contributed by Matt Wozniski and Pablo Galindo in gh-131591 [https://github.com/python/cpython/issues/131591].)

- Hardcoded breakpoints (`breakpoint()` and `set_trace()`) now reuse the most recent `Pdb` instance that calls `set_trace()`, instead of creating a new one each time. As a result, all the instance specific data like `display` and `commands` are preserved across hardcoded breakpoints. (Contributed by Tian Gao in gh-121450 [https://github.com/python/cpython/issues/121450].)

- Add a new argument *mode* to **`pdb.Pdb`**. Disable the `restart` command when **`pdb`** is in `inline` mode. (Contributed by Tian Gao in gh-123757 [https://github.com/python/cpython/issues/123757].)

- A confirmation prompt will be shown when the user tries to quit **`pdb`** in `inline` mode. `y`, `Y`, `<Enter>` or `EOF` will confirm the quit and call **`sys.exit()`**, instead of raising **`bdb.BdbQuit`**. (Contributed by Tian Gao in gh-124704 [https://github.com/python/cpython/issues/124704].)

- Inline breakpoints like **`breakpoint()`** or **`pdb.set_trace()`** will always stop the program at calling frame, ignoring the `skip` pattern (if any). (Contributed by Tian Gao in gh-130493 [https://github.com/python/cpython/issues/130493].)

- `<tab>` at the beginning of the line in **`pdb`** multi-line input will fill in a 4-space indentation now, instead of inserting a `\t` character. (Contributed by Tian Gao in gh-130471 [https://github.com/python/cpython/issues/130471].)

- Auto-indent is introduced in **`pdb`** multi-line input. It will either keep the indentation of the last line or insert a 4-space indentation when it detects a new code block. (Contributed by Tian Gao in gh-133350 [https://github.com/python/cpython/issues/133350].)

- `$_asynctask` is added to access the current asyncio task if applicable. (Contributed by Tian Gao in gh-124367 [https://github.com/python/cpython/issues/124367].)

- **`pdb.set_trace_async()`** is added to support debugging asyncio coroutines. **`await`** statements are supported with this function. (Contributed by Tian Gao in gh-132576 [https://github.com/python/cpython/issues/132576].)

- Source code displayed in **`pdb`** will be syntax-highlighted. This feature can be controlled using the same methods as the default interactive shell, in addition to the newly added `colorize` argument of **`pdb.Pdb`**. (Contributed by Tian Gao and Łukasz Langa in gh-133355 [https://github.com/python/cpython/issues/133355].)

pickle

- Set the default protocol version on the **`pickle`** module to 5. For more details, see pickle protocols.
- Add exception notes for pickle serialization errors that allow identifying the source of the error. (Contributed by Serhiy Storchaka in gh-122213 [https://github.com/python/cpython/issues/122213].)

platform

- Add **`invalidate_caches()`**, a function to invalidate cached results in the **`platform`** module. (Contributed by Bénédikt Tran in gh-122549 [https://github.com/python/cpython/issues/122549].)

pydoc

- Annotations in help output are now usually displayed in a format closer to that in the original source. (Contributed by Jelle Zijlstra in gh-101552 [https://github.com/python/cpython/issues/101552].)

re

- Support `\z` as a synonym for `\Z` in **`regular expressions`**. It is interpreted unambiguously in many other regular expression engines, unlike `\Z`, which has subtly different behavior. (Contributed by Serhiy Storchaka in gh-133306 [https://github.com/python/cpython/issues/133306].)
- `\B` in **`regular expression`** now matches the empty input string, meaning that it is now always the opposite of `\b`. (Contributed by Serhiy Storchaka in gh-124130 [https://github.com/python/cpython/issues/124130].)

socket

- Improve and fix support for Bluetooth sockets.
    - Fix support of Bluetooth sockets on NetBSD and DragonFly BSD. (Contributed by Serhiy Storchaka in gh-132429 [https://github.com/python/cpython/issues/132429].)
    - Fix support for `BTPROTO_HCI` on FreeBSD. (Contributed by Victor Stinner in gh-111178 [https://github.com/python/cpython/issues/111178].)
    - Add support for `BTPROTO_SCO` on FreeBSD. (Contributed by Serhiy Storchaka in gh-85302 [https://github.com/python/cpython/issues/85302].)
    - Add support for *cid* and *bdaddr_type* in the address for `BTPROTO_L2CAP` on FreeBSD. (Contributed by Serhiy Storchaka in gh-132429 [https://github.com/python/cpython/issues/132429].)
    - Add support for *channel* in the address for `BTPROTO_HCI` on Linux. (Contributed by Serhiy Storchaka in gh-70145 [https://github.com/python/cpython/issues/70145].)
    - Accept an integer as the address for `BTPROTO_HCI` on Linux. (Contributed by Serhiy Storchaka in gh-132099 [https://github.com/python/cpython/issues/132099].)
    - Return *cid* in `getsockname()` for `BTPROTO_L2CAP`. (Contributed by Serhiy Storchaka in gh-132429 [https://github.com/python/cpython/issues/132429].)
    - Add many new constants. (Contributed by Serhiy Storchaka in gh-132734 [https://github.com/python/cpython/issues/132734].)

ssl

- Indicate through the `HAS_PHA` Boolean whether the `ssl` module supports TLSv1.3 post-handshake client authentication (PHA). (Contributed by Will Childs-Klein in gh-128036 [https://github.com/python/cpython/issues/128036].)

struct

- Support the float complex and double complex C types in the `struct` module (formatting characters `'F'` and `'D'` respectively). (Contributed by Sergey B Kirpichev in

gh-121249 [https://github.com/python/cpython/issues/121249].)

symtable

- Expose the following `Symbol` methods:
  - `is_comp_cell()`
  - `is_comp_iter()`
  - `is_free_class()`

  (Contributed by Bénédikt Tran in gh-120029 [https://github.com/python/cpython/issues/120029].)

sys

- The previously undocumented special function `sys.getobjects()`, which only exists in specialized builds of Python, may now return objects from other interpreters than the one it's called in. (Contributed by Eric Snow in gh-125286 [https://github.com/python/cpython/issues/125286].)
- Add `sys._is_immortal()` for determining if an object is immortal. (Contributed by Peter Bierma in gh-128509 [https://github.com/python/cpython/issues/128509].)
- On FreeBSD, `sys.platform` no longer contains the major version number. It is always `'freebsd'`, instead of `'freebsd13'` or `'freebsd14'`. (Contributed by Michael Osipov in gh-129393 [https://github.com/python/cpython/issues/129393].)
- Raise `DeprecationWarning` for `sys._clear_type_cache()`. This function was deprecated in Python 3.13 but it didn't raise a runtime warning.
- Add `sys.remote_exec()` to implement the new external debugger interface. See PEP 768 for details. (Contributed by Pablo Galindo Salgado, Matt Wozniski, and Ivona Stojanovic in gh-131591 [https://github.com/python/cpython/issues/131591].)
- Add the `sys._jit` namespace, containing utilities for introspecting just-in-time compilation. (Contributed by Brandt Bucher in gh-133231 [https://github.com/python/cpython/issues/133231].)

sys.monitoring

- Add two new monitoring events, `BRANCH_LEFT` and `BRANCH_RIGHT`. These replace and deprecate the `BRANCH` event. (Contributed by Mark Shannon in gh-122548 [https://github.com/python/cpython/issues/122548].)

sysconfig

- Add `ABIFLAGS` key to `get_config_vars()` on Windows. (Contributed by Xuehai Pan in gh-131799 [https://github.com/python/cpython/issues/131799].)

tarfile

- `data_filter()` now normalizes symbolic link targets in order to avoid path traversal attacks. (Contributed by Petr Viktorin in gh-127987 [https://github.com/python/cpython/issues/127987] and **CVE 2025-4138** [https://www.cve.org/CVERecord?id=CVE-2025-4138].)
- `extractall()` now skips fixing up directory attributes when a directory was removed or replaced by another kind of file. (Contributed by Petr Viktorin in gh-127987 [https://github.com/python/cpython/issues/127987] and **CVE 2024-12718** [https://www.cve.org/CVERecord?id=CVE-2024-12718].)
- `extract()` and `extractall()` now (re-)apply the extraction filter when substituting a link (hard or symbolic) with a copy of another archive member, and when fixing up directory attributes. The former raises a new exception, `LinkFallbackError`. (Contributed by Petr Viktorin for **CVE 2025-4330** [https://www.cve.org/CVERecord?id=CVE-2025-4330] and **CVE 2024-12718** [https://www.cve.org/CVERecord?id=CVE-2024-12718].)
- `extract()` and `extractall()` no longer extract rejected members when `errorlevel()` is zero. (Contributed by Matt Prodani and Petr Viktorin in gh-112887 [https://github.com/python/cpython/issues/112887] and **CVE 2025-4435** [https://www.cve.org/CVERecord?id=CVE-2025-4435].)

threading

- `threading.Thread.start()` now sets the operating system thread name to `threading.Thread.name`. (Contributed by Victor Stinner in gh-59705 [https://github.com/python/cpython/issues/59705].)

tkinter

- Make `tkinter` widget methods **`after()`** and **`after_idle()`** accept keyword arguments. (Contributed by Zhikang Yan in gh-126899 [https://github.com/python/cpython/issues/126899].)
- Add ability to specify a name for **`tkinter.OptionMenu`** and **`tkinter.ttk.OptionMenu`**. (Contributed by Zhikang Yan in gh-130482 [https://github.com/python/cpython/issues/130482].)

turtle

- Add context managers for `turtle.fill()`, `turtle.poly()`, and `turtle.no_animation()`. (Contributed by Marie Roald and Yngve Mardal Moe in gh-126350 [https://github.com/python/cpython/issues/126350].)

types

- `types.UnionType` is now an alias for `typing.Union`. See below for more details. (Contributed by Jelle Zijlstra in gh-105499 [https://github.com/python/cpython/issues/105499].)

typing

- The `types.UnionType` and `typing.Union` types are now aliases for each other, meaning that both old-style unions (created with `Union[int, str]`) and new-style unions (`int | str`) now create instances of the same runtime type. This unifies the behavior between the two syntaxes, but leads to some differences in behavior that may affect users who introspect types at runtime:

- Both syntaxes for creating a union now produce the same string representation in **`repr()`**. For example, `repr(Union[int, str])` is now `"int | str"` instead of `"typing.Union[int, str]"`.
- Unions created using the old syntax are no longer cached. Previously, running `Union[int, str]` multiple times would return the same object (`Union[int, str] is Union[int, str]` would be `True`), but now it will return two different objects. Use `==` to compare unions for equality, not `is`. New-style unions have never been cached this way. This change could increase memory usage for some programs that use a large number of unions created by subscripting `typing.Union`. However, several factors offset this cost: unions used in annotations are no longer evaluated by default in Python 3.14 because of **PEP 649** [https://peps.python.org/pep-0649/]; an instance of **`types.UnionType`** is itself much smaller than the object returned by `Union[]` was on prior Python versions; and removing the cache also saves some space. It is therefore unlikely that this change will cause a significant increase in memory usage for most users.
- Previously, old-style unions were implemented using the private class `typing._UnionGenericAlias`. This class is no longer needed for the implementation, but it has been retained for backward compatibility, with removal scheduled for Python 3.17. Users should use documented introspection helpers like **`get_origin()`** and **`typing.get_args()`** instead of relying on private implementation details.
- It is now possible to use **`typing.Union`** itself in **`isinstance()`** checks. For example, `isinstance(int | str, typing.Union)` will return `True`; previously this raised **`TypeError`**.
- The **`__args__`** attribute of **`typing.Union`** objects is no longer writable.
- It is no longer possible to set any attributes on **`Union`**

  objects. This only ever worked for dunder attributes on previous versions, was never documented to work, and was subtly broken in many cases.

  (Contributed by Jelle Zijlstra in gh-105499 [https://github.com/python/cpython/issues/105499].)

- `TypeAliasType` now supports star unpacking.

unicodedata

- The Unicode database has been updated to Unicode 16.0.0.

unittest

- `unittest` output is now colored by default. This can be controlled by environment variables. (Contributed by Hugo van Kemenade in gh-127221 [https://github.com/python/cpython/issues/127221].)
- unittest discovery supports namespace package as start directory again. It was removed in Python 3.11. (Contributed by Jacob Walls in gh-80958 [https://github.com/python/cpython/issues/80958].)
- A number of new methods were added in the `TestCase` class that provide more specialized tests.
  - `assertHasAttr()` and `assertNotHasAttr()` check whether the object has a particular attribute.
  - `assertIsSubclass()` and `assertNotIsSubclass()` check whether the object is a subclass of a particular class, or of one of a tuple of classes.
  - `assertStartsWith()`, `assertNotStartsWith()`, `assertEndsWith()` and `assertNotEndsWith()` check whether the Unicode or byte string starts or ends with particular strings.

  (Contributed by Serhiy Storchaka in gh-71339 [https://github.com/python/cpython/issues/71339].)

urllib

- Upgrade HTTP digest authentication algorithm for **`urllib.request`** by supporting SHA-256 digest authentication as specified in **RFC 7616** [https://datatracker.ietf.org/doc/html/rfc7616.html]. (Contributed by Calvin Bui in gh-128193 [https://github.com/python/cpython/issues/128193].)

- Improve ergonomics and standards compliance when parsing and emitting `file:` URLs.

  In **`url2pathname()`**:

  - ○ Accept a complete URL when the new *require_scheme* argument is set to true.
  - ○ Discard URL authority if it matches the local hostname.
  - ○ Discard URL authority if it resolves to a local IP address when the new *resolve_host* argument is set to true.
  - ○ Discard URL query and fragment components.
  - ○ Raise **`URLError`** if a URL authority isn't local, except on Windows where we return a UNC path as before.

  In **`pathname2url()`**:

  - ○ Return a complete URL when the new *add_scheme* argument is set to true.
  - ○ Include an empty URL authority when a path begins with a slash. For example, the path `/etc/hosts` is converted to the URL `///etc/hosts`.

  On Windows, drive letters are no longer converted to uppercase, and `:` characters not following a drive letter no longer cause an **`OSError`** exception to be raised.

  (Contributed by Barney Gale in gh-125866 [https://github.com/python/cpython/issues/125866].)

uuid

- Add support for UUID versions 6, 7, and 8 via **`uuid6()`**, `uuid7()`, and `uuid8()` respectively, as specified in **RFC 9562** [https://datatracker.ietf.org/doc/html/rfc9562.html]. (Contributed by Bénédikt Tran in gh-89083 [https://github.com/python/cpython/issues/89083].)

- `NIL` and `MAX` are now available to represent the Nil and Max UUID formats as defined by **RFC 9562** [https://datatracker.ietf.org/doc/html/rfc9562.html]. (Contributed by Nick Pope in gh-128427 [https://github.com/python/cpython/issues/128427].)
- Allow generating multiple UUIDs simultaneously on the command-line via `python -m uuid --count`. (Contributed by Simon Legner in gh-131236 [https://github.com/python/cpython/issues/131236].)

webbrowser

- Names in the `BROWSER` environment variable can now refer to already registered browsers for the `webbrowser` module, instead of always generating a new browser command.

  This makes it possible to set `BROWSER` to the value of one of the supported browsers on macOS.

zipfile

- Added `ZipInfo._for_archive`, a method to resolve suitable defaults for a `ZipInfo` object as used by `ZipFile.writestr`. (Contributed by Bénédikt Tran in gh-123424 [https://github.com/python/cpython/issues/123424].)
- `ZipFile.writestr()` now respects the `SOURCE_DATE_EPOCH` environment variable in order to better support reproducible builds. (Contributed by Jiahao Li in gh-91279 [https://github.com/python/cpython/issues/91279].)

Optimizations

- The import time for several standard library modules has been improved, including `annotationlib`, `ast`, `asyncio`, `base64`, `cmd`, `csv`, `gettext`, `importlib.util`, `locale`, `mimetypes`, `optparse`, `pickle`, `pprint`, `pstats`, `shlex`, `socket`, `string`, `subprocess`, `threading`, `tomllib`, `types`, and `zipfile`.

(Contributed by Adam Turner, Bénédikt Tran, Chris Markiewicz, Eli Schwartz, Hugo van Kemenade, Jelle Zijlstra, and others in gh-118761 [https://github.com/python/cpython/issues/118761].)

- The interpreter now avoids some reference count modifications internally when it's safe to do so. This can lead to different values being returned from `sys.getrefcount()` and `Py_REFCNT()` compared to previous versions of Python. See below for details.

asyncio

- Standard benchmark results have improved by 10-20% following the implementation of a new per-thread doubly linked list for `native tasks`, also reducing memory usage. This enables external introspection tools such as python -m asyncio pstree to introspect the call graph of asyncio tasks running in all threads. (Contributed by Kumar Aditya in gh-107803 [https://github.com/python/cpython/issues/107803].)
- The module now has first class support for free-threading builds. This enables parallel execution of multiple event loops across different threads, scaling linearly with the number of threads. (Contributed by Kumar Aditya in gh-128002 [https://github.com/python/cpython/issues/128002].)

base64

- `b16decode()` is now up to six times faster. (Contributed by Bénédikt Tran, Chris Markiewicz, and Adam Turner in gh-118761 [https://github.com/python/cpython/issues/118761].)

bdb

- The basic debugger now has a `sys.monitoring`-based backend, which can be selected via the passing `'monitoring'` to the `Bdb` class's new *backend* parameter. (Contributed by Tian Gao in gh-124533 [https://github.com/python/cpython/issues/124533].)

difflib

- The `IS_LINE_JUNK()` function is now up to twice as fast. (Contributed by Adam Turner and Semyon Moroz in gh-130167 [https://github.com/python/cpython/issues/130167].)

gc

- The new incremental garbage collector means that maximum pause times are reduced by an order of magnitude or more for larger heaps.

  Because of this optimization, the meaning of the results of `get_threshold()` and `set_threshold()` have changed, along with `get_count()` and `get_stats()`.

  - For backwards compatibility, `get_threshold()` continues to return a three-item tuple. The first value is the threshold for young collections, as before; the second value determines the rate at which the old collection is scanned (the default is 10, and higher values mean that the old collection is scanned more slowly). The third value is now meaningless and is always zero.
  - `set_threshold()` now ignores any items after the second.
  - `get_count()` and `get_stats()` continue to return the same format of results. The only difference is that instead of the results referring to the young, aging and old generations, the results refer to the young generation and the aging and collecting spaces of the old generation.

  In summary, code that attempted to manipulate the behavior of the cycle GC may not work exactly as intended, but it is very unlikely to be harmful. All other code will work just fine.

(Contributed by Mark Shannon in gh-108362 [https://github.com/python/cpython/issues/108362].)

io

- Opening and reading files now executes fewer system calls. Reading a small operating system cached file in full is up to 15% faster. (Contributed by Cody Maloney and Victor Stinner in gh-120754 [https://github.com/python/cpython/issues/120754] and gh-90102 [https://github.com/python/cpython/issues/90102].)

pathlib

- `Path.read_bytes` now uses unbuffered mode to open files, which is between 9% and 17% faster to read in full. (Contributed by Cody Maloney in gh-120754 [https://github.com/python/cpython/issues/120754].)

pdb

- `pdb` now supports two backends, based on either `sys.settrace()` or `sys.monitoring`. Using the pdb CLI or `breakpoint()` will always use the `sys.monitoring` backend. Explicitly instantiating `pdb.Pdb` and its derived classes will use the `sys.settrace()` backend by default, which is configurable. (Contributed by Tian Gao in gh-124533 [https://github.com/python/cpython/issues/124533].)

textwrap

- Optimize the `dedent()` function, improving performance by an average of 2.4x, with larger improvements for bigger inputs, and fix a bug with incomplete normalization of blank lines with whitespace characters other than space and tab.

uuid

- **`uuid3()`** and **`uuid5()`** are now both roughly 40% faster for 16-byte names and 20% faster for 1024-byte names. Performance for longer names remains unchanged. (Contributed by Bénédikt Tran in gh-128150 [https://github.com/python/cpython/issues/128150].)
- **`uuid4()`** is now c. 30% faster. (Contributed by Bénédikt Tran in gh-128150 [https://github.com/python/cpython/issues/128150].)

zlib

- On Windows, zlib-ng [https://github.com/zlib-ng/zlib-ng] is now used as the implementation of the **`zlib`** module in the default binaries. There are no known incompatibilities between `zlib-ng` and the previously-used `zlib` implementation. This should result in better performance at all compression levels.

  It is worth noting that `zlib.Z_BEST_SPEED` (`1`) may result in significantly less compression than the previous implementation, whilst also significantly reducing the time taken to compress.

  (Contributed by Steve Dower in gh-91349 [https://github.com/python/cpython/issues/91349].)

Removed

argparse

- Remove the *type*, *choices*, and *metavar* parameters of **`BooleanOptionalAction`**. These have been deprecated since Python 3.12. (Contributed by Nikita Sobolev in gh-118805 [https://github.com/python/cpython/issues/118805].)
- Calling **`add_argument_group()`** on an argument group now raises a **`ValueError`**. Similarly, **`add_argument_group()`** or **`add_mutually_exclusive_group()`** on a mutually exclusive group now both raise **`ValueError`**s. This “nesting” was never supported, often failed to work correctly,

and was unintentionally exposed through inheritance. This functionality has been deprecated since Python 3.11. (Contributed by Savannah Ostrowski in gh-127186 [https://github.com/python/cpython/issues/127186].)

ast

- Remove the following classes, which have been deprecated aliases of **`Constant`** since Python 3.8 and have emitted deprecation warnings since Python 3.12:

  ○ **`Bytes`**
  ○ **`Ellipsis`**
  ○ **`NameConstant`**
  ○ **`Num`**
  ○ **`Str`**

  As a consequence of these removals, user-defined `visit_Num`, `visit_Str`, `visit_Bytes`, `visit_NameConstant` and `visit_Ellipsis` methods on custom **`NodeVisitor`** subclasses will no longer be called when the **`NodeVisitor`** subclass is visiting an AST. Define a `visit_Constant` method instead.

  (Contributed by Alex Waygood in gh-119562 [https://github.com/python/cpython/issues/119562].)

- Remove the following deprecated properties on **`ast.Constant`**, which were present for compatibility with the now-removed AST classes:

  ○ **`Constant.n`**
  ○ **`Constant.s`**

  Use **`Constant.value`** instead. (Contributed by Alex Waygood in gh-119562 [https://github.com/python/cpython/issues/119562].)

asyncio

- Remove the following classes, methods, and functions, which have been deprecated since Python 3.12:

- ○ **`AbstractChildWatcher`**
- ○ **`FastChildWatcher`**
- ○ **`MultiLoopChildWatcher`**
- ○ **`PidfdChildWatcher`**
- ○ **`SafeChildWatcher`**
- ○ **`ThreadedChildWatcher`**
- ○ **`AbstractEventLoopPolicy.get_child_watcher()`**
- ○ **`AbstractEventLoopPolicy.set_child_watcher()`**
- ○ **`get_child_watcher()`**
- ○ **`set_child_watcher()`**

(Contributed by Kumar Aditya in gh-120804 [https://github.com/python/cpython/issues/120804].)

- **`asyncio.get_event_loop()`** now raises a **`RuntimeError`** if there is no current event loop, and no longer implicitly creates an event loop.

  (Contributed by Kumar Aditya in gh-126353 [https://github.com/python/cpython/issues/126353].)

  There's a few patterns that use **`asyncio.get_event_loop()`**, most of them can be replaced with **`asyncio.run()`**.

  If you're running an async function, simply use **`asyncio.run()`**.

  Before:

```
async def main():
    ...


loop = asyncio.get_event_loop()
try:
    loop.run_until_complete(main())
finally:
    loop.close()
```

After:

```
async def main():
    ...

asyncio.run(main())
```

If you need to start something, for example, a server listening on a socket and then run forever, use **`asyncio.run()`** and an **`asyncio.Event`**.

Before:

```
def start_server(loop): ...

loop = asyncio.get_event_loop()
try:
    start_server(loop)
    loop.run_forever()
finally:
    loop.close()
```

After:

```
def start_server(loop): ...

async def main():
    start_server(asyncio.get_running_loop())
    await asyncio.Event().wait()

asyncio.run(main())
```

If you need to run something in an event loop, then run some blocking code around it, use **`asyncio.Runner`**.

Before:

```
async def operation_one(): ...
def blocking_code(): ...
async def operation_two(): ...
```

```
loop = asyncio.get_event_loop()
try:
    loop.run_until_complete(operation_one())
    blocking_code()
    loop.run_until_complete(operation_two())
finally:
    loop.close()
```

After:

```
async def operation_one(): ...
def blocking_code(): ...
async def operation_two(): ...

with asyncio.Runner() as runner:
    runner.run(operation_one())
    blocking_code()
    runner.run(operation_two())
```

email

- Remove `email.utils.localtime()`"s *isdst* parameter, which was deprecated in and has been ignored since Python 3.12. (Contributed by Hugo van Kemenade in gh-118798 [https://github.com/python/cpython/issues/118798].)

importlib.abc

- Remove deprecated `importlib.abc` classes:
    - ○ **`ResourceReader`** (use `TraversableResources`)
    - ○ **`Traversable`** (use `Traversable`)
    - ○ **`TraversableResources`** (use `TraversableResources`)

  (Contributed by Jason R. Coombs and Hugo van Kemenade in gh-93963 [https://github.com/python/cpython/issues/93963].)

itertools

- Remove support for copy, deepcopy, and pickle operations from `itertools` iterators. These have emitted a `DeprecationWarning` since Python 3.12. (Contributed by Raymond Hettinger in gh-101588 [https://github.com/python/cpython/issues/101588].)

pathlib

- Remove support for passing additional keyword arguments to `Path`. In previous versions, any such arguments are ignored. (Contributed by Barney Gale in gh-74033 [https://github.com/python/cpython/issues/74033].)
- Remove support for passing additional positional arguments to `PurePath.relative_to()` and `is_relative_to()`. In previous versions, any such arguments are joined onto *other*. (Contributed by Barney Gale in gh-78707 [https://github.com/python/cpython/issues/78707].)

pkgutil

- Remove the `get_loader()` and `find_loader()` functions, which have been deprecated since Python 3.12. (Contributed by Bénédikt Tran in gh-97850 [https://github.com/python/cpython/issues/97850].)

pty

- Remove the `master_open()` and `slave_open()` functions, which have been deprecated since Python 3.12. Use `pty.openpty()` instead. (Contributed by Nikita Sobolev in gh-118824 [https://github.com/python/cpython/issues/118824].)

sqlite3

- Remove `version` and `version_info` from the `sqlite3` module; use `sqlite_version` and `sqlite_version_info` for the actual version number of the runtime SQLite library. (Contributed by Hugo van

Kemenade in gh-118924 [https://github.com/python/cpython/issues/118924].)

- Using a sequence of parameters with named placeholders now raises a **`ProgrammingError`**, having been deprecated since Python 3.12. (Contributed by Erlend E. Aasland in gh-118928 [https://github.com/python/cpython/issues/118928] and gh-101693 [https://github.com/python/cpython/issues/101693].)

urllib

- Remove the **`Quoter`** class from **`urllib.parse`**, which has been deprecated since Python 3.11. (Contributed by Nikita Sobolev in gh-118827 [https://github.com/python/cpython/issues/118827].)

- Remove the **`URLopener`** and **`FancyURLopener`** classes from **`urllib.request`**, which have been deprecated since Python 3.3.

  `myopener.open()` can be replaced with **`urlopen()`**. `myopener.retrieve()` can be replaced with **`urlretrieve()`**. Customisations to the opener classes can be replaced by passing customized handlers to **`build_opener()`**. (Contributed by Barney Gale in gh-84850 [https://github.com/python/cpython/issues/84850].)

Deprecated

New deprecations

- Passing a complex number as the *real* or *imag* argument in the **`complex()`** constructor is now deprecated; complex numbers should only be passed as a single positional argument. (Contributed by Serhiy Storchaka in gh-109218 [https://github.com/python/cpython/issues/109218].)

- **`argparse`**:

  - Passing the undocumented keyword argument *prefix_chars* to the **`add_argument_group()`** method is now deprecated. (Contributed by Savannah Ostrowski in gh-125563 [https://github.com/python/cpython/issues/125563].)

  - Deprecated the `argparse.FileType` type converter. Anything relating to resource management should be handled downstream, after the arguments have been parsed. (Contributed by Serhiy Storchaka in gh-58032 [https://github.com/python/cpython/issues/58032].)

- `asyncio`:

  - The **`asyncio.iscoroutinefunction()`** is now deprecated and will be removed in Python 3.16; use `inspect.iscoroutinefunction()` instead. (Contributed by Jiahao Li and Kumar Aditya in gh-122875 [https://github.com/python/cpython/issues/122875].)

  - The `asyncio` policy system is deprecated and will be removed in Python 3.16. In particular, the following classes and functions are deprecated:

    - `asyncio.AbstractEventLoopPolicy`
    - `asyncio.DefaultEventLoopPolicy`
    - `asyncio.WindowsSelectorEventLoopPolicy`
    - `asyncio.WindowsProactorEventLoopPolicy`
    - `asyncio.get_event_loop_policy()`
    - `asyncio.set_event_loop_policy()`

    Users should use `asyncio.run()` or `asyncio.Runner` with the *loop_factory* argument to use the desired event loop implementation.

    For example, to use `asyncio.SelectorEventLoop` on Windows:

```
import asyncio

async def main():
    ...

asyncio.run(main(), loop_factory=asyncio.Select
```

(Contributed by Kumar Aditya in gh-127949 [https://github.com/python/cpython/issues/127949].)

- `codecs`: The `codecs.open()` function is now deprecated, and will be removed in a future version of Python. Use `open()` instead. (Contributed by Inada Naoki in gh-133036 [https://github.com/python/cpython/issues/133036].)
- `ctypes`:
  - On non-Windows platforms, setting `Structure._pack_` to use a MSVC-compatible default memory layout is now deprecated in favor of setting `Structure._layout_` to `'ms'`, and will be removed in Python 3.19. (Contributed by Petr Viktorin in gh-131747 [https://github.com/python/cpython/issues/131747].)
  - Calling `ctypes.POINTER()` on a string is now deprecated. Use incomplete types for self-referential structures. Also, the internal `ctypes._pointer_type_cache` is deprecated. See **`ctypes.POINTER()`** for updated implementation details. (Contributed by Sergey Myrianov in gh-100926 [https://github.com/python/cpython/issues/100926].)
- `functools`: Calling the Python implementation of `functools.reduce()` with *function* or *sequence* as keyword arguments is now deprecated; the parameters will be made positional-only in Python 3.16. (Contributed by Kirill Podoprigora in gh-121676 [https://github.com/python/cpython/issues/121676].)
- `logging`: Support for custom logging handlers with the *strm* argument is now deprecated and scheduled for removal in Python 3.16. Define handlers with the *stream* argument instead. (Contributed by Mariusz Felisiak in gh-115032 [https://github.com/python/cpython/issues/115032].)
- `mimetypes`: Valid extensions are either empty or must start with “.” for `mimetypes.MimeTypes.add_type()`. Undotted extensions are deprecated and will raise a

`ValueError` in Python 3.16. (Contributed by Hugo van Kemenade in gh-75223 [https://github.com/python/cpython/issues/75223].)

- **`nturl2path`**: This module is now deprecated. Call `urllib.request.url2pathname()` and `pathname2url()` instead. (Contributed by Barney Gale in gh-125866 [https://github.com/python/cpython/issues/125866].)

- **`os`**: The `os.popen()` and `os.spawn*` functions are now soft deprecated. They should no longer be used to write new code. The `subprocess` module is recommended instead. (Contributed by Victor Stinner in gh-120743 [https://github.com/python/cpython/issues/120743].)

- **`pathlib`**: `pathlib.PurePath.as_uri()` is now deprecated and scheduled for removal in Python 3.19. Use `pathlib.Path.as_uri()` instead. (Contributed by Barney Gale in gh-123599 [https://github.com/python/cpython/issues/123599].)

- **`pdb`**: The undocumented `pdb.Pdb.curframe_locals` attribute is now a deprecated read-only property, which will be removed in a future version of Python. The low overhead dynamic frame locals access added in Python 3.13 by **PEP 667** [https://peps.python.org/pep-0667/] means the frame locals cache reference previously stored in this attribute is no longer needed. Derived debuggers should access `pdb.Pdb.curframe.f_locals` directly in Python 3.13 and later versions. (Contributed by Tian Gao in gh-124369 [https://github.com/python/cpython/issues/124369] and gh-125951 [https://github.com/python/cpython/issues/125951].)

- **`symtable`**: Deprecate `symtable.Class.get_methods()` due to the lack of interest, scheduled for removal in Python 3.16. (Contributed by Bénédikt Tran in gh-119698 [https://github.com/python/cpython/issues/119698].)

- **`tkinter`**: The `tkinter.Variable` methods `trace_variable()`, `trace_vdelete()` and `trace_vinfo()` are now deprecated. Use `trace_add()`,

`trace_remove()` and **`trace_info()`** instead. (Contributed by Serhiy Storchaka in gh-120220 [https://github.com/python/cpython/issues/120220].)

- **`urllib.parse`**: Accepting objects with false values (like `0` and `[]`) except empty strings, bytes-like objects and `None` in **`parse_qsl()`** and **`parse_qs()`** is now deprecated. (Contributed by Serhiy Storchaka in gh-116897 [https://github.com/python/cpython/issues/116897].)

Pending removal in Python 3.15

- The import system:
    - ○ Setting **`__cached__`** on a module while failing to set **`__spec__.cached`** is deprecated. In Python 3.15, **`__cached__`** will cease to be set or take into consideration by the import system or standard library. (gh-97879 [https://github.com/python/cpython/issues/97879])
    - ○ Setting **`__package__`** on a module while failing to set **`__spec__.parent`** is deprecated. In Python 3.15, **`__package__`** will cease to be set or take into consideration by the import system or standard library. (gh-97879 [https://github.com/python/cpython/issues/97879])
- **`ctypes`**:
    - ○ The undocumented **`ctypes.SetPointerType()`** function has been deprecated since Python 3.13.
- **`http.server`**:
    - ○ The obsolete and rarely used **`CGIHTTPRequestHandler`** has been deprecated since Python 3.13. No direct replacement exists. *Anything* is better than CGI to interface a web server with a request handler.
    - ○ The **`--cgi`** flag to the **python -m http.server** command-line interface has been deprecated since Python 3.13.
- **`importlib`**:
    - ○ `load_module()` method: use `exec_module()` instead.
- **`locale`**:
    - ○ The **`getdefaultlocale()`** function has been

deprecated since Python 3.11. Its removal was originally planned for Python 3.13 (gh-90817 [https://github.com/python/cpython/issues/90817]), but has been postponed to Python 3.15. Use `getlocale()`, `setlocale()`, and `getencoding()` instead. (Contributed by Hugo van Kemenade in gh-111187 [https://github.com/python/cpython/issues/111187].)

- `pathlib`:
    - `PurePath.is_reserved()` has been deprecated since Python 3.13. Use `os.path.isreserved()` to detect reserved paths on Windows.
- `platform`:
    - `java_ver()` has been deprecated since Python 3.13. This function is only useful for Jython support, has a confusing API, and is largely untested.
- `sysconfig`:
    - The *check_home* argument of `sysconfig.is_python_build()` has been deprecated since Python 3.12.
- `threading`:
    - `RLock()` will take no arguments in Python 3.15. Passing any arguments has been deprecated since Python 3.14, as the Python version does not permit any arguments, but the C version allows any number of positional or keyword arguments, ignoring every argument.
- `types`:
    - `types.CodeType`: Accessing `co_lnotab` was deprecated in **PEP 626** [https://peps.python.org/pep-0626/] since 3.10 and was planned to be removed in 3.12, but it only got a proper `DeprecationWarning` in 3.12. May be removed in 3.15. (Contributed by Nikita Sobolev in gh-101866 [https://github.com/python/cpython/issues/101866].)
- `typing`:
    - The undocumented keyword argument syntax for creating `NamedTuple` classes (for example, `Point = NamedTuple("Point", x=int, y=int)`) has been deprecated since Python 3.13. Use the class-based syntax or the functional syntax instead.

  - When using the functional syntax of `TypedDict`s, failing to pass a value to the *fields* parameter (`TD = TypedDict("TD")`) or passing `None` (`TD = TypedDict("TD", None)`) has been deprecated since Python 3.13. Use `class TD(TypedDict): pass` or `TD = TypedDict("TD", {})` to create a TypedDict with zero field.
  - The `typing.no_type_check_decorator()` decorator function has been deprecated since Python 3.13. After eight years in the `typing` module, it has yet to be supported by any major type checker.
- `wave`:
  - The `getmark()`, `setmark()`, and `getmarkers()` methods of the `Wave_read` and `Wave_write` classes have been deprecated since Python 3.13.
- `zipimport`:
  - `load_module()` has been deprecated since Python 3.10. Use `exec_module()` instead. (Contributed by Jiahao Li in gh-125746 [https://github.com/python/cpython/issues/125746].)

Pending removal in Python 3.16

- The import system:
  - Setting `__loader__` on a module while failing to set `__spec__.loader` is deprecated. In Python 3.16, `__loader__` will cease to be set or taken into consideration by the import system or the standard library.
- `array`:
  - The `'u'` format code (`wchar_t`) has been deprecated in documentation since Python 3.3 and at runtime since Python 3.13. Use the `'w'` format code (`Py_UCS4`) for Unicode characters instead.
- `asyncio`:
  - `asyncio.iscoroutinefunction()` is deprecated

and will be removed in Python 3.16; use **`inspect.iscoroutinefunction()`** instead. (Contributed by Jiahao Li and Kumar Aditya in gh-122875 [https://github.com/python/cpython/issues/122875].)

  - **`asyncio`** policy system is deprecated and will be removed in Python 3.16. In particular, the following classes and functions are deprecated:
    - **`asyncio.AbstractEventLoopPolicy`**
    - **`asyncio.DefaultEventLoopPolicy`**
    - **`asyncio.WindowsSelectorEventLoopPolicy`**
    - **`asyncio.WindowsProactorEventLoopPolicy`**
    - **`asyncio.get_event_loop_policy()`**
    - **`asyncio.set_event_loop_policy()`**

    Users should use **`asyncio.run()`** or **`asyncio.Runner`** with *loop_factory* to use the desired event loop implementation.

    For example, to use **`asyncio.SelectorEventLoop`** on Windows:

    ```
    import asyncio

    async def main():
        ...

    asyncio.run(main(), loop_factory=asyncio.Select
    ```

    (Contributed by Kumar Aditya in gh-127949 [https://github.com/python/cpython/issues/127949].)

- **`builtins`**:
  - Bitwise inversion on boolean types, `~True` or `~False` has been deprecated since Python 3.12, as it produces surprising and unintuitive results (`-2` and `-1`). Use `not x` instead for the logical negation of a Boolean. In the rare case that you need the bitwise inversion of the underlying integer, convert to `int` explicitly

  (`~int(x)`).

- **`functools`**:
    - Calling the Python implementation of `functools.reduce()` with *function* or *sequence* as keyword arguments has been deprecated since Python 3.14.
- **`logging`**:

  Support for custom logging handlers with the *strm* argument is deprecated and scheduled for removal in Python 3.16. Define handlers with the *stream* argument instead. (Contributed by Mariusz Felisiak in gh-115032 [https://github.com/python/cpython/issues/115032].)
- **`mimetypes`**:
    - Valid extensions start with a "." or are empty for `mimetypes.MimeTypes.add_type()`. Undotted extensions are deprecated and will raise a `ValueError` in Python 3.16. (Contributed by Hugo van Kemenade in gh-75223 [https://github.com/python/cpython/issues/75223].)
- **`shutil`**:
    - The **`ExecError`** exception has been deprecated since Python 3.14. It has not been used by any function in **`shutil`** since Python 3.4, and is now an alias of `RuntimeError`.
- **`symtable`**:
    - The `Class.get_methods` method has been deprecated since Python 3.14.
- **`sys`**:
    - The `_enablelegacywindowsfsencoding()` function has been deprecated since Python 3.13. Use

the `PYTHONLEGACYWINDOWSFSENCODING` environment variable instead.

- `sysconfig`:
  - The **`sysconfig.expand_makefile_vars()`** function has been deprecated since Python 3.14. Use the `vars` argument of `sysconfig.get_paths()` instead.
- `tarfile`:
  - The undocumented and unused **`TarFile.tarfile`** attribute has been deprecated since Python 3.13.

Pending removal in Python 3.17

- `collections.abc`:
  - `collections.abc.ByteString` is scheduled for removal in Python 3.17.

    Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use `Buffer` or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).

    `ByteString` was originally intended to be an abstract class that would serve as a supertype of both `bytes` and `bytearray`. However, since the ABC never had any methods, knowing that an object was an instance of **`ByteString`** never actually told you anything useful about the object. Other common buffer types such as `memoryview` were also never understood as subtypes of **`ByteString`** (either at runtime or by static type checkers).

    See PEP 688 [https://peps.python.org/pep-0688/#current-options] for more details. (Contributed by Shantanu Jain in gh-91896 [https://github.com/python/cpython/issues/91896].)

- `typing`:
  - Before Python 3.14, old-style unions were implemented using the private class `typing._UnionGenericAlias`. This class is no longer needed for the implementation, but it has been retained for backward compatibility, with removal scheduled for Python 3.17. Users should use documented introspection helpers like `typing.get_origin()` and `typing.get_args()` instead of relying on private implementation details.
  - `typing.ByteString`, deprecated since Python 3.9, is scheduled for removal in Python 3.17.

    Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use `Buffer` or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).

    `ByteString` was originally intended to be an abstract class that would serve as a supertype of both `bytes` and `bytearray`. However, since the ABC never had any methods, knowing that an object was an instance of **`ByteString`** never actually told you anything useful about the object. Other common buffer types such as `memoryview` were also never understood as subtypes of **`ByteString`** (either at runtime or by static type checkers).

    See PEP 688 [https://peps.python.org/pep-0688/#current-options] for more details. (Contributed by Shantanu Jain in gh-91896 [https://github.com/python/cpython/issues/91896].)

Pending removal in Python 3.18

- **`decimal`**:
    - The non-standard and undocumented **`Decimal`** format specifier `'N'`, which is only supported in the **`decimal`** module’s C implementation, has been deprecated since Python 3.13. (Contributed by Serhiy Storchaka in gh-89902 [https://github.com/python/cpython/issues/89902].)

Pending removal in Python 3.19

- **`ctypes`**:
    - Implicitly switching to the MSVC-compatible struct layout by setting **`_pack_`** but not **`_layout_`** on non-Windows platforms.

Pending removal in future versions

The following APIs will be removed in the future, although there is currently no date scheduled for their removal.

- **`argparse`**:
    - Nesting argument groups and nesting mutually exclusive groups are deprecated.
    - Passing the undocumented keyword argument *prefix_chars* to **`add_argument_group()`** is now deprecated.
    - The **`argparse.FileType`** type converter is deprecated.
- **`builtins`**:
    - Generators: `throw(type, exc, tb)` and `athrow(type, exc, tb)` signature is deprecated: use `throw(exc)` and `athrow(exc)` instead, the single argument signature.
    - Currently Python accepts numeric literals immediately followed by keywords, for example `0in x`, `1or x`, `0if 1else 2`. It allows confusing and ambiguous expressions like `[0x1for x in y]` (which can be interpreted as `[0x1 for x in y]` or `[0x1f or x in y]`). A syntax warning is raised if the numeric literal is immediately followed by one of keywords

    `and`, **`else`**, **`for`**, **`if`**, **`in`**, **`is`** and **`or`**. In a future release it will be changed to a syntax error. (gh-87999 [https://github.com/python/cpython/issues/87999])
  - Support for `__index__()` and `__int__()` method returning non-int type: these methods will be required to return an instance of a strict subclass of **`int`**.
  - Support for `__float__()` method returning a strict subclass of **`float`**: these methods will be required to return an instance of **`float`**.
  - Support for `__complex__()` method returning a strict subclass of **`complex`**: these methods will be required to return an instance of **`complex`**.
  - Passing a complex number as the *real* or *imag* argument in the **`complex()`** constructor is now deprecated; it should only be passed as a single positional argument. (Contributed by Serhiy Storchaka in gh-109218 [https://github.com/python/cpython/issues/109218].)
- **`calendar`**: `calendar.January` and `calendar.February` constants are deprecated and replaced by **`calendar.JANUARY`** and **`calendar.FEBRUARY`**. (Contributed by Prince Roshan in gh-103636 [https://github.com/python/cpython/issues/103636].)
- **`codecs`**: use **`open()`** instead of **`codecs.open()`**. (gh-133038 [https://github.com/python/cpython/issues/133038])
- **`codeobject.co_lnotab`**: use the **`codeobject.co_lines()`** method instead.
- **`datetime`**:
  - **`utcnow()`**: use `datetime.datetime.now(tz=datetime.UTC)`.
  - **`utcfromtimestamp()`**: use `datetime.datetime.fromtimestamp(timestamp, tz=datetime.UTC)`.
- **`gettext`**: Plural value must be an integer.
- **`importlib`**:
  - **`cache_from_source()`** *debug_override* parameter is deprecated: use the *optimization* parameter instead.
- **`importlib.metadata`**:
  - `EntryPoints` tuple interface.
  - Implicit `None` on return values.
- **`logging`**: the `warn()` method has been deprecated since

Python 3.3, use `warning()` instead.

- `mailbox`: Use of StringIO input and text mode is deprecated, use BytesIO and binary mode instead.
- `os`: Calling `os.register_at_fork()` in multi-threaded process.
- **`pydoc.ErrorDuringImport`**: A tuple value for *exc_info* parameter is deprecated, use an exception instance.
- `re`: More strict rules are now applied for numerical group references and group names in regular expressions. Only sequence of ASCII digits is now accepted as a numerical reference. The group name in bytes patterns and replacement strings can now only contain ASCII letters and digits and underscore. (Contributed by Serhiy Storchaka in gh-91760 [https://github.com/python/cpython/issues/91760].)
- **`sre_compile`**, **`sre_constants`** and **`sre_parse`** modules.
- `shutil`: `rmtree()`"s *onerror* parameter is deprecated in Python 3.12; use the *onexc* parameter instead.
- `ssl` options and protocols:
    - `ssl.SSLContext` without protocol argument is deprecated.
    - `ssl.SSLContext`: `set_npn_protocols()` and **`selected_npn_protocol()`** are deprecated: use ALPN instead.
    - `ssl.OP_NO_SSL*` options
    - `ssl.OP_NO_TLS*` options
    - `ssl.PROTOCOL_SSLv3`
    - `ssl.PROTOCOL_TLS`
    - `ssl.PROTOCOL_TLSv1`
    - `ssl.PROTOCOL_TLSv1_1`
    - `ssl.PROTOCOL_TLSv1_2`
    - `ssl.TLSVersion.SSLv3`
    - `ssl.TLSVersion.TLSv1`
    - `ssl.TLSVersion.TLSv1_1`
- `threading` methods:
    - **`threading.Condition.notifyAll()`**: use `notify_all()`.
    - **`threading.Event.isSet()`**: use `is_set()`.
    - **`threading.Thread.isDaemon()`**, `threading.Thread.setDaemon()`: use `threading.Thread.daemon` attribute.

- ○ **`threading.Thread.getName()`**, **`threading.Thread.setName()`**: use **`threading.Thread.name`** attribute.
  - ○ **`threading.currentThread()`**: use **`threading.current_thread()`**.
  - ○ **`threading.activeCount()`**: use **`threading.active_count()`**.
- **`typing.Text`** (gh-92332 [https://github.com/python/cpython/issues/92332]).
- The internal class `typing._UnionGenericAlias` is no longer used to implement **`typing.Union`**. To preserve compatibility with users using this private class, a compatibility shim will be provided until at least Python 3.17. (Contributed by Jelle Zijlstra in gh-105499 [https://github.com/python/cpython/issues/105499].)
- **`unittest.IsolatedAsyncioTestCase`**: it is deprecated to return a value that is not `None` from a test case.
- **`urllib.parse`** deprecated functions: **`urlparse()`** instead
  - ○ `splitattr()`
  - ○ `splithost()`
  - ○ `splitnport()`
  - ○ `splitpasswd()`
  - ○ `splitport()`
  - ○ `splitquery()`
  - ○ `splittag()`
  - ○ `splittype()`
  - ○ `splituser()`
  - ○ `splitvalue()`
  - ○ `to_bytes()`
- **`wsgiref`**: `SimpleHandler.stdout.write()` should not do partial writes.
- **`xml.etree.ElementTree`**: Testing the truth value of an **`Element`** is deprecated. In a future release it will always return `True`. Prefer explicit `len(elem)` or `elem is not None` tests instead.
- **`sys._clear_type_cache()`** is deprecated: use **`sys._clear_internal_caches()`** instead.

CPython bytecode changes

- Replaced the opcode **`BINARY_SUBSCR`** by the **`BINARY_OP`** opcode with the `NB_SUBSCR` oparg. (Contributed by Irit Katriel in gh-100239 [https://github.com/python/cpython/issues/100239].)
- Add the **`BUILD_INTERPOLATION`** and **`BUILD_TEMPLATE`** opcodes to construct new **`Interpolation`** and **`Template`** instances, respectively. (Contributed by Lysandros Nikolaou and others in gh-132661 [https://github.com/python/cpython/issues/132661]; see also PEP 750: Template strings).
- Remove the **`BUILD_CONST_KEY_MAP`** opcode. Use **`BUILD_MAP`** instead. (Contributed by Mark Shannon in gh-122160 [https://github.com/python/cpython/issues/122160].)
- Replace the **`LOAD_ASSERTION_ERROR`** opcode with **`LOAD_COMMON_CONSTANT`** and add support for loading **`NotImplementedError`**.
- Add the **`LOAD_FAST_BORROW`** and **`LOAD_FAST_BORROW_LOAD_FAST_BORROW`** opcodes to reduce reference counting overhead when the interpreter can prove that the reference in the frame outlives the reference loaded onto the stack. (Contributed by Matt Page in gh-130704 [https://github.com/python/cpython/issues/130704].)
- Add the **`LOAD_SMALL_INT`** opcode, which pushes a small integer equal to the `oparg` to the stack. The **`RETURN_CONST`** opcode is removed as it is no longer used. (Contributed by Mark Shannon in gh-125837 [https://github.com/python/cpython/issues/125837].)
- Add the new **`LOAD_SPECIAL`** instruction. Generate code for **`with`** and **`async with`** statements using the new instruction. Removed the **`BEFORE_WITH`** and **`BEFORE_ASYNC_WITH`** instructions. (Contributed by Mark Shannon in gh-120507 [https://github.com/python/cpython/issues/120507].)
- Add the **`POP_ITER`** opcode to support “virtual” iterators. (Contributed by Mark Shannon in gh-132554 [https://github.com/python/cpython/issues/132554].)

Pseudo-instructions

- Add the **`ANNOTATIONS_PLACEHOLDER`** pseudo instruction to support partially executed module-level annotations with deferred evaluation of annotations. (Contributed by Jelle Zijlstra in gh-130907 [https://github.com/python/cpython/issues/130907].)

- Add the **`BINARY_OP_EXTEND`** pseudo instruction, which executes a pair of functions (guard and specialization functions) accessed from the inline cache. (Contributed by Irit Katriel in gh-100239 [https://github.com/python/cpython/issues/100239].)
- Add three specializations for **`CALL_KW`**; **`CALL_KW_PY`** for calls to Python functions, **`CALL_KW_BOUND_METHOD`** for calls to bound methods, and **`CALL_KW_NON_PY`** for all other calls. (Contributed by Mark Shannon in gh-118093 [https://github.com/python/cpython/issues/118093].)
- Add the **`JUMP_IF_TRUE`** and **`JUMP_IF_FALSE`** pseudo instructions, conditional jumps which do not impact the stack. Replaced by the sequence `COPY 1`, `TO_BOOL`, `POP_JUMP_IF_TRUE/FALSE`. (Contributed by Irit Katriel in gh-124285 [https://github.com/python/cpython/issues/124285].)
- Add the **`LOAD_CONST_MORTAL`** pseudo instruction. (Contributed by Mark Shannon in gh-128685 [https://github.com/python/cpython/issues/128685].)
- Add the **`LOAD_CONST_IMMORTAL`** pseudo instruction, which does the same as **`LOAD_CONST`**, but is more efficient for immortal objects. (Contributed by Mark Shannon in gh-125837 [https://github.com/python/cpython/issues/125837].)
- Add the **`NOT_TAKEN`** pseudo instruction, used by **`sys.monitoring`** to record branch events (such as **`BRANCH_LEFT`**). (Contributed by Mark Shannon in gh-122548 [https://github.com/python/cpython/issues/122548].)

C API changes

Python configuration C API

Add a PyInitConfig C API to configure the Python initialization without relying on C structures and the ability to make ABI-compatible changes in the future.

Complete the **PEP 587** [https://peps.python.org/pep-0587/] PyConfig C

API by adding `PyInitConfig_AddModule()` which can be used to add a built-in extension module; a feature previously referred to as the «inittab».

Add `PyConfig_Get()` and `PyConfig_Set()` functions to get and set the current runtime configuration.

PEP 587 [https://peps.python.org/pep-0587/] "Python Initialization Configuration" unified all the ways to configure Python's initialization. This PEP also unifies the configuration of Python's preinitialization and initialization in a single API. Moreover, this PEP only provides a single choice to embed Python, instead of having two "Python" and "Isolated" choices (PEP 587), to further simplify the API.

The lower level PEP 587 PyConfig API remains available for use cases with an intentionally higher level of coupling to CPython implementation details (such as emulating the full functionality of CPython's CLI, including its configuration mechanisms).

(Contributed by Victor Stinner in gh-107954 [https://github.com/python/cpython/issues/107954].)

Ver también

PEP 741 [https://peps.python.org/pep-0741/] and **PEP 587** [https://peps.python.org/pep-0587/]

New features in the C API

- Add `Py_PACK_VERSION()` and `Py_PACK_FULL_VERSION()`, two new macros for bit-packing Python version numbers. This is useful for comparisons with `Py_Version` or `PY_VERSION_HEX`. (Contributed by Petr Viktorin in gh-128629 [https://github.com/python/cpython/issues/128629].)
- Add `PyBytes_Join(sep, iterable)` function, similar to `sep.join(iterable)` in Python. (Contributed by Victor Stinner in gh-121645 [https://github.com/python/cpython/

issues/121645].)

- Add functions to manipulate the configuration of the current runtime Python interpreter (PEP 741: Python configuration C API):

  - ○ `PyConfig_Get()`
  - ○ `PyConfig_GetInt()`
  - ○ `PyConfig_Set()`
  - ○ `PyConfig_Names()`

  (Contributed by Victor Stinner in gh-107954 [https://github.com/python/cpython/issues/107954].)

- Add functions to configure Python initialization (PEP 741: Python configuration C API):

  - ○ `Py_InitializeFromInitConfig()`
  - ○ `PyInitConfig_AddModule()`
  - ○ `PyInitConfig_Create()`
  - ○ `PyInitConfig_Free()`
  - ○ `PyInitConfig_FreeStrList()`
  - ○ `PyInitConfig_GetError()`
  - ○ `PyInitConfig_GetExitCode()`
  - ○ `PyInitConfig_GetInt()`
  - ○ `PyInitConfig_GetStr()`
  - ○ `PyInitConfig_GetStrList()`
  - ○ `PyInitConfig_HasOption()`
  - ○ `PyInitConfig_SetInt()`
  - ○ `PyInitConfig_SetStr()`
  - ○ `PyInitConfig_SetStrList()`

  (Contributed by Victor Stinner in gh-107954 [https://github.com/python/cpython/issues/107954].)

- Add `Py_fopen()` function to open a file. This works similarly to the standard C `fopen()` function, instead accepting a Python object for the *path* parameter and setting an exception on error. The corresponding new `Py_fclose()` function should be used to close a file. (Contributed by Victor Stinner in gh-127350 [https://github.com/

python/cpython/issues/127350].)

- Add `Py_HashBuffer()` to compute and return the hash value of a buffer. (Contributed by Antoine Pitrou and Victor Stinner in gh-122854 [https://github.com/python/cpython/issues/122854].)

- Add `PyImport_ImportModuleAttr()` and `PyImport_ImportModuleAttrString()` helper functions to import a module and get an attribute of the module. (Contributed by Victor Stinner in gh-128911 [https://github.com/python/cpython/issues/128911].)

- Add `PyIter_NextItem()` to replace `PyIter_Next()`, which has an ambiguous return value. (Contributed by Irit Katriel and Erlend Aasland in gh-105201 [https://github.com/python/cpython/issues/105201].)

- Add `PyLong_GetSign()` function to get the sign of `int` objects. (Contributed by Sergey B Kirpichev in gh-116560 [https://github.com/python/cpython/issues/116560].)

- Add `PyLong_IsPositive()`, `PyLong_IsNegative()` and `PyLong_IsZero()` for checking if `PyLongObject` is positive, negative, or zero, respectively. (Contributed by James Roy and Sergey B Kirpichev in gh-126061 [https://github.com/python/cpython/issues/126061].)

- Add new functions to convert C `<stdint.h>` numbers to/from Python `int` objects:

    ○ `PyLong_AsInt32()`
    ○ `PyLong_AsInt64()`
    ○ `PyLong_AsUInt32()`
    ○ `PyLong_AsUInt64()`
    ○ `PyLong_FromInt32()`
    ○ `PyLong_FromInt64()`
    ○ `PyLong_FromUInt32()`
    ○ `PyLong_FromUInt64()`

  (Contributed by Victor Stinner in gh-120389 [https://github.com/

python/cpython/issues/120389].)

- Add a new import and export API for Python `int` objects (**PEP 757** [https://peps.python.org/pep-0757/]):

  - `PyLong_GetNativeLayout()`
  - `PyLong_Export()`
  - `PyLong_FreeExport()`
  - `PyLongWriter_Create()`
  - `PyLongWriter_Finish()`
  - `PyLongWriter_Discard()`

  (Contributed by Sergey B Kirpichev and Victor Stinner in gh-102471 [https://github.com/python/cpython/issues/102471].)

- Add `PyMonitoring_FireBranchLeftEvent()` and `PyMonitoring_FireBranchRightEvent()` for generating `BRANCH_LEFT` and `BRANCH_RIGHT` events, respectively. (Contributed by Mark Shannon in gh-122548 [https://github.com/python/cpython/issues/122548].)

- Add `PyType_Freeze()` function to make a type immutable. (Contributed by Victor Stinner in gh-121654 [https://github.com/python/cpython/issues/121654].)

- Add `PyType_GetBaseByToken()` and `Py_tp_token` slot for easier superclass identification, which attempts to resolve the type checking issue mentioned in **PEP 630** [https://peps.python.org/pep-0630/#type-checking]. (Contributed in gh-124153 [https://github.com/python/cpython/issues/124153].)

- Add a new `PyUnicode_Equal()` function to test if two strings are equal. The function is also added to the Limited C API. (Contributed by Victor Stinner in gh-124502 [https://github.com/python/cpython/issues/124502].)

- Add a new `PyUnicodeWriter` API to create a Python `str` object, with the following functions:

  - `PyUnicodeWriter_Create()`
  - `PyUnicodeWriter_DecodeUTF8Stateful()`

  - `PyUnicodeWriter_Discard()`
  - `PyUnicodeWriter_Finish()`
  - `PyUnicodeWriter_Format()`
  - `PyUnicodeWriter_WriteASCII()`
  - `PyUnicodeWriter_WriteChar()`
  - `PyUnicodeWriter_WriteRepr()`
  - `PyUnicodeWriter_WriteStr()`
  - `PyUnicodeWriter_WriteSubstring()`
  - `PyUnicodeWriter_WriteUCS4()`
  - `PyUnicodeWriter_WriteUTF8()`
  - `PyUnicodeWriter_WriteWideChar()`

  (Contributed by Victor Stinner in gh-119182 [https://github.com/python/cpython/issues/119182].)

- The `k` and `K` formats in `PyArg_ParseTuple()` and similar functions now use `__index__()` if available, like all other integer formats. (Contributed by Serhiy Storchaka in gh-112068 [https://github.com/python/cpython/issues/112068].)

- Add support for a new `p` format unit in `Py_BuildValue()` that produces a Python `bool` object from a C integer. (Contributed by Pablo Galindo in bpo-45325 [https://bugs.python.org/issue?@action=redirect&bpo=45325].)

- Add `PyUnstable_IsImmortal()` for determining if an object is immortal, for debugging purposes. (Contributed by Peter Bierma in gh-128509 [https://github.com/python/cpython/issues/128509].)

- Add `PyUnstable_Object_EnableDeferredRefcount()` for enabling deferred reference counting, as outlined in **PEP 703** [https://peps.python.org/pep-0703/].

- Add `PyUnstable_Object_IsUniquelyReferenced()` as a replacement for `Py_REFCNT(op) == 1` on free threaded builds. (Contributed by Peter Bierma in gh-133140 [https://github.com/python/cpython/issues/133140].)

- Add `PyUnstable_Object_IsUniqueReferencedTemporary()`

to determine if an object is a unique temporary object on the interpreter's operand stack. This can be used in some cases as a replacement for checking if **`Py_REFCNT()`** is `1` for Python objects passed as arguments to C API functions. (Contributed by Sam Gross in gh-133164 [https://github.com/python/cpython/issues/133164].)

Limited C API changes

- In the limited C API version 3.14 and newer, **`Py_TYPE()`** and **`Py_REFCNT()`** are now implemented as an opaque function call to hide implementation details. (Contributed by Victor Stinner in gh-120600 [https://github.com/python/cpython/issues/120600] and gh-124127 [https://github.com/python/cpython/issues/124127].)
- Remove the **`PySequence_Fast_GET_SIZE`**, **`PySequence_Fast_GET_ITEM`**, and **`PySequence_Fast_ITEMS`** macros from the limited C API, since they have always been broken in the limited C API. (Contributed by Victor Stinner in gh-91417 [https://github.com/python/cpython/issues/91417].)

Removed C APIs

- Creating **`immutable types`** with mutable bases was deprecated in Python 3.12, and now raises a **`TypeError`**. (Contributed by Nikita Sobolev in gh-119775 [https://github.com/python/cpython/issues/119775].)
- Remove `PyDictObject.ma_version_tag` member, which was deprecated in Python 3.12. Use the **`PyDict_AddWatcher()`** API instead. (Contributed by Sam Gross in gh-124296 [https://github.com/python/cpython/issues/124296].)
- Remove the private `_Py_InitializeMain()` function. It was a provisional API added to Python 3.8 by **PEP 587** [https://peps.python.org/pep-0587/]. (Contributed by Victor Stinner in gh-129033 [https://github.com/python/cpython/issues/129033].)
- Remove the undocumented APIs **`Py_C_RECURSION_LIMIT`** and **`PyThreadState.c_recursion_remaining`**. These were added in 3.13 and have been removed without

deprecation. Use `Py_EnterRecursiveCall()` to guard against runaway recursion in C code. (Removed by Petr Viktorin in gh-133079 [https://github.com/python/cpython/issues/133079], see also gh-130396 [https://github.com/python/cpython/issues/130396].)

Deprecated C APIs

- The `Py_HUGE_VAL` macro is now soft deprecated. Use `Py_INFINITY` instead. (Contributed by Sergey B Kirpichev in gh-120026 [https://github.com/python/cpython/issues/120026].)
- The `Py_IS_NAN`, `Py_IS_INFINITY`, and `Py_IS_FINITE` macros are now soft deprecated. Use `isnan`, `isinf` and `isfinite` instead, available from `math.h` since C99. (Contributed by Sergey B Kirpichev in gh-119613 [https://github.com/python/cpython/issues/119613].)
- Non-tuple sequences are now deprecated as argument for the `(items)` format unit in `PyArg_ParseTuple()` and other argument parsing functions if *items* contains format units which store a borrowed buffer or a borrowed reference. (Contributed by Serhiy Storchaka in gh-50333 [https://github.com/python/cpython/issues/50333].)
- The `_PyMonitoring_FireBranchEvent` function is now deprecated and should be replaced with calls to `PyMonitoring_FireBranchLeftEvent()` and `PyMonitoring_FireBranchRightEvent()`.
- The previously undocumented function `PySequence_In()` is now soft deprecated. Use `PySequence_Contains()` instead. (Contributed by Yuki Kobayashi in gh-127896 [https://github.com/python/cpython/issues/127896].)

Pending removal in Python 3.15

- The `PyImport_ImportModuleNoBlock()`: Use `PyImport_ImportModule()` instead.

- `PyWeakref_GetObject()` and `PyWeakref_GET_OBJECT()`: Use `PyWeakref_GetRef()` instead. The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get

`PyWeakref_GetRef()` on Python 3.12 and older.

- `Py_UNICODE` type and the `Py_UNICODE_WIDE` macro: Use `wchar_t` instead.
- `PyUnicode_AsDecodedObject()`: Use `PyCodec_Decode()` instead.
- `PyUnicode_AsDecodedUnicode()`: Use `PyCodec_Decode()` instead; Note that some codecs (for example, «base64») may return a type other than `str`, such as `bytes`.
- `PyUnicode_AsEncodedObject()`: Use `PyCodec_Encode()` instead.
- `PyUnicode_AsEncodedUnicode()`: Use `PyCodec_Encode()` instead; Note that some codecs (for example, «base64») may return a type other than `bytes`, such as `str`.
- Python initialization functions, deprecated in Python 3.13:
    - `Py_GetPath()`: Use `PyConfig_Get("module_search_paths")` (`sys.path`) instead.
    - `Py_GetPrefix()`: Use `PyConfig_Get("base_prefix")` (`sys.base_prefix`) instead. Use `PyConfig_Get("prefix")` (`sys.prefix`) if virtual environments need to be handled.
    - `Py_GetExecPrefix()`: Use `PyConfig_Get("base_exec_prefix")` (`sys.base_exec_prefix`) instead. Use `PyConfig_Get("exec_prefix")` (`sys.exec_prefix`) if virtual environments need to be handled.
    - `Py_GetProgramFullPath()`: Use `PyConfig_Get("executable")` (`sys.executable`) instead.
    - `Py_GetProgramName()`: Use

    `PyConfig_Get("executable")` (`sys.executable`) instead.
  - `Py_GetPythonHome()`: Use `PyConfig_Get("home")` or the `PYTHONHOME` environment variable instead.

  The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get `PyConfig_Get()` on Python 3.13 and older.

- Functions to configure Python's initialization, deprecated in Python 3.11:

  - `PySys_SetArgvEx()`: Set `PyConfig.argv` instead.
  - `PySys_SetArgv()`: Set `PyConfig.argv` instead.
  - `Py_SetProgramName()`: Set `PyConfig.program_name` instead.
  - `Py_SetPythonHome()`: Set `PyConfig.home` instead.
  - `PySys_ResetWarnOptions()`: Clear `sys.warnoptions` and `warnings.filters` instead.

  The `Py_InitializeFromConfig()` API should be used with `PyConfig` instead.

- Global configuration variables:

  - `Py_DebugFlag`: Use `PyConfig.parser_debug` or `PyConfig_Get("parser_debug")` instead.
  - `Py_VerboseFlag`: Use `PyConfig.verbose` or `PyConfig_Get("verbose")` instead.
  - `Py_QuietFlag`: Use `PyConfig.quiet` or `PyConfig_Get("quiet")` instead.
  - `Py_InteractiveFlag`: Use `PyConfig.interactive` or `PyConfig_Get("interactive")` instead.
  - `Py_InspectFlag`: Use `PyConfig.inspect` or `PyConfig_Get("inspect")` instead.
  - `Py_OptimizeFlag`: Use `PyConfig.optimization_level` or `PyConfig_Get("optimization_level")` instead.
  - `Py_NoSiteFlag`: Use `PyConfig.site_import` or

    `PyConfig_Get("site_import")` instead.

  - `Py_BytesWarningFlag`: Use `PyConfig.bytes_warning` or `PyConfig_Get("bytes_warning")` instead.
  - `Py_FrozenFlag`: Use `PyConfig.pathconfig_warnings` or `PyConfig_Get("pathconfig_warnings")` instead.
  - `Py_IgnoreEnvironmentFlag`: Use `PyConfig.use_environment` or `PyConfig_Get("use_environment")` instead.
  - `Py_DontWriteBytecodeFlag`: Use `PyConfig.write_bytecode` or `PyConfig_Get("write_bytecode")` instead.
  - `Py_NoUserSiteDirectory`: Use `PyConfig.user_site_directory` or `PyConfig_Get("user_site_directory")` instead.
  - `Py_UnbufferedStdioFlag`: Use `PyConfig.buffered_stdio` or `PyConfig_Get("buffered_stdio")` instead.
  - `Py_HashRandomizationFlag`: Use `PyConfig.use_hash_seed` and `PyConfig.hash_seed` or `PyConfig_Get("hash_seed")` instead.
  - `Py_IsolatedFlag`: Use `PyConfig.isolated` or `PyConfig_Get("isolated")` instead.
  - `Py_LegacyWindowsFSEncodingFlag`: Use `PyPreConfig.legacy_windows_fs_encoding` or `PyConfig_Get("legacy_windows_fs_encoding")` instead.
  - `Py_LegacyWindowsStdioFlag`: Use `PyConfig.legacy_windows_stdio` or `PyConfig_Get("legacy_windows_stdio")` instead.
  - `Py_FileSystemDefaultEncoding`, `Py_HasFileSystemDefaultEncoding`: Use `PyConfig.filesystem_encoding` or `PyConfig_Get("filesystem_encoding")` instead.
  - `Py_FileSystemDefaultEncodeErrors`: Use `PyConfig.filesystem_errors` or `PyConfig_Get("filesystem_errors")` instead.

  - **`Py_UTF8Mode`**: Use `PyPreConfig.utf8_mode` or `PyConfig_Get("utf8_mode")` instead. (see `Py_PreInitialize()`)

The `Py_InitializeFromConfig()` API should be used with `PyConfig` to set these options. Or `PyConfig_Get()` can be used to get these options at runtime.

Pending removal in Python 3.16

- The bundled copy of `libmpdec`.

Pending removal in Python 3.18

- The following private functions are deprecated and planned for removal in Python 3.18:
  - **`_PyBytes_Join()`**: use `PyBytes_Join()`.
  - **`_PyDict_GetItemStringWithError()`**: use `PyDict_GetItemStringRef()`.
  - **`_PyDict_Pop()`**: use `PyDict_Pop()`.
  - **`_PyLong_Sign()`**: use `PyLong_GetSign()`.
  - **`_PyLong_FromDigits()`** and **`_PyLong_New()`**: use `PyLongWriter_Create()`.
  - **`_PyThreadState_UncheckedGet()`**: use `PyThreadState_GetUnchecked()`.
  - **`_PyUnicode_AsString()`**: use `PyUnicode_AsUTF8()`.
  - **`_PyUnicodeWriter_Init()`**: replace `_PyUnicodeWriter_Init(&writer)` with `writer = PyUnicodeWriter_Create(0)`.
  - **`_PyUnicodeWriter_Finish()`**: replace `_PyUnicodeWriter_Finish(&writer)` with `PyUnicodeWriter_Finish(writer)`.
  - **`_PyUnicodeWriter_Dealloc()`**: replace `_PyUnicodeWriter_Dealloc(&writer)` with `PyUnicodeWriter_Discard(writer)`.
  - **`_PyUnicodeWriter_WriteChar()`**: replace `_PyUnicodeWriter_WriteChar(&writer, ch)` with `PyUnicodeWriter_WriteChar(writer,

    `ch)`.
  - **`_PyUnicodeWriter_WriteStr()`**: replace `_PyUnicodeWriter_WriteStr(&writer, str)` with `PyUnicodeWriter_WriteStr(writer, str)`.
  - **`_PyUnicodeWriter_WriteSubstring()`**: replace `_PyUnicodeWriter_WriteSubstring(&writer, str, start, end)` with `PyUnicodeWriter_WriteSubstring(writer, str, start, end)`.
  - **`_PyUnicodeWriter_WriteASCIIString()`**: replace `_PyUnicodeWriter_WriteASCIIString(&writer, str)` with `PyUnicodeWriter_WriteASCII(writer, str)`.
  - **`_PyUnicodeWriter_WriteLatin1String()`**: replace `_PyUnicodeWriter_WriteLatin1String(&writer, str)` with `PyUnicodeWriter_WriteUTF8(writer, str)`.
  - **`_PyUnicodeWriter_Prepare()`**: (no replacement).
  - **`_PyUnicodeWriter_PrepareKind()`**: (no replacement).
  - **`_Py_HashPointer()`**: use `Py_HashPointer()`.
  - **`_Py_fopen_obj()`**: use `Py_fopen()`.

  The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get these new public functions on Python 3.13 and older. (Contributed by Victor Stinner in gh-128863 [https://github.com/python/cpython/issues/128863].)

Pending removal in future versions

The following APIs are deprecated and will be removed, although there is currently no date scheduled for their removal.

- `Py_TPFLAGS_HAVE_FINALIZE`: Unneeded since Python 3.8.
- `PyErr_Fetch()`: Use `PyErr_GetRaisedException()` instead.
- `PyErr_NormalizeException()`: Use

`PyErr_GetRaisedException()` instead.
- `PyErr_Restore()`: Use `PyErr_SetRaisedException()` instead.
- `PyModule_GetFilename()`: Use `PyModule_GetFilenameObject()` instead.
- `PyOS_AfterFork()`: Use `PyOS_AfterFork_Child()` instead.
- `PySlice_GetIndicesEx()`: Use `PySlice_Unpack()` and `PySlice_AdjustIndices()` instead.
- `PyUnicode_READY()`: Unneeded since Python 3.12
- **`PyErr_Display()`**: Use `PyErr_DisplayException()` instead.
- **`_PyErr_ChainExceptions()`**: Use **`_PyErr_ChainExceptions1()`** instead.
- **`PyBytesObject.ob_shash`** member: call `PyObject_Hash()` instead.
- Thread Local Storage (TLS) API:
  - `PyThread_create_key()`: Use `PyThread_tss_alloc()` instead.
  - `PyThread_delete_key()`: Use `PyThread_tss_free()` instead.
  - `PyThread_set_key_value()`: Use `PyThread_tss_set()` instead.
  - `PyThread_get_key_value()`: Use `PyThread_tss_get()` instead.
  - `PyThread_delete_key_value()`: Use `PyThread_tss_delete()` instead.
  - `PyThread_ReInitTLS()`: Unneeded since Python 3.7.

Build changes

- PEP 776 [https://peps.python.org/pep-0776/]: Emscripten is now an officially supported platform at tier 3 [https://peps.python.org/pep-0011/#tier-3]. As a part of this effort, more than 25 bugs in Emscripten libc [https://emscripten.org/docs/porting/emscripten-runtime-environment.html] were fixed. Emscripten now includes support for `ctypes`, `termios`, and `fcntl`, as well as experimental support for the new default interactive shell.

(Contributed by R. Hood Chatham in gh-127146 [https://github.com/python/cpython/issues/127146], gh-127683 [https://github.com/python/cpython/issues/127683], and gh-136931 [https://github.com/python/cpython/issues/136931].)

- Official Android binary releases are now provided on python.org [https://www.python.org/downloads/android/].
- GNU Autoconf 2.72 is now required to generate `configure`. (Contributed by Erlend Aasland in gh-115765 [https://github.com/python/cpython/issues/115765].)
- `wasm32-unknown-emscripten` is now a **PEP 11** [https://peps.python.org/pep-0011/] tier 3 platform. (Contributed by R. Hood Chatham in gh-127146 [https://github.com/python/cpython/issues/127146], gh-127683 [https://github.com/python/cpython/issues/127683], and gh-136931 [https://github.com/python/cpython/issues/136931].)
- `#pragma`-based linking with `python3*.lib` can now be switched off with Py_NO_LINK_LIB. (Contributed by Jean-Christophe Fillion-Robin in gh-82909 [https://github.com/python/cpython/issues/82909].)
- CPython now enables a set of recommended compiler options by default for improved security. Use the **`--disable-safety`** `configure` option to disable them, or the **`--enable-slower-safety`** option for a larger set of compiler options, albeit with a performance cost.
- The `WITH_FREELISTS` macro and `--without-freelists` `configure` option have been removed.
- The new `configure` option **`--with-tail-call-interp`** may be used to enable the experimental tail call interpreter. See A new type of interpreter for further details.
- To disable the new remote debugging support, use the **`--without-remote-debug`** `configure` option. This may be useful for security reasons.
- iOS and macOS apps can now be configured to redirect `stdout` and `stderr` content to the system log. (Contributed by Russell Keith-Magee in gh-127592 [https://github.com/python/cpython/issues/127592].)
- The iOS testbed is now able to stream test output while the test is running. The testbed can also be used to run the test suite of projects other than CPython itself. (Contributed by Russell Keith-Magee in gh-127592 [https://github.com/python/

cpython/issues/127592].)

`build-details.json`

Installations of Python now contain a new file, `build-details.json`. This is a static JSON document containing build details for CPython, to allow for introspection without needing to run code. This is helpful for use-cases such as Python launchers, cross-compilation, and so on.

`build-details.json` must be installed in the platform-independent standard library directory. This corresponds to the "stdlib" **`sysconfig`** installation path, which can be found by running `sysconfig.get_path('stdlib')`.

Ver también

PEP 739 [https://peps.python.org/pep-0739/] – `build-details.json` 1.0 – a static description file for Python build details

Discontinuation of PGP signatures

PGP (Pretty Good Privacy) signatures will not be provided for releases of Python 3.14 or future versions. To verify CPython artifacts, users must use Sigstore verification materials [https://www.python.org/downloads/metadata/sigstore/]. Releases have been signed using Sigstore [https://www.sigstore.dev/] since Python 3.11.

This change in release process was specified in **PEP 761** [https://peps.python.org/pep-0761/].

Free-threaded Python is officially supported

The free-threaded build of Python is now supported and no longer experimental. This is the start of phase II [https://discuss.python.org/t/37075] where free-threaded Python is officially supported but still optional.

The free-threading team are confident that the project is on the right path, and appreciate the continued dedication from everyone working to make free-threading ready for broader adoption across the Python community.

With these recommendations and the acceptance of this PEP, the Python developer community should broadly advertise that free-threading is a supported Python build option now and into the future, and that it will not be removed without a proper deprecation schedule.

Any decision to transition to phase III [https://discuss.python.org/t/37075], with free-threading as the default or sole build of Python is still undecided, and dependent on many factors both within CPython itself and the community. This decision is for the future.

Ver también

PEP 779 [https://peps.python.org/pep-0779/]

PEP 779's acceptance [https://discuss.python.org/t/84319/123]

Binary releases for the experimental just-in-time compiler

The official macOS and Windows release binaries now include an *experimental* just-in-time (JIT) compiler. Although it is **not** recommended for production use, it can be tested by setting `PYTHON_JIT=1` as an environment variable. Downstream source builds and redistributors can use the `--enable-experimental-jit=yes-off` configuration option for similar behavior.

The JIT is at an early stage and still in active development. As such, the typical performance impact of enabling it can range from 10% slower to 20% faster, depending on workload. To aid in testing and evaluation, a set of introspection functions has been provided in the `sys._jit` namespace. `sys._jit.is_available()` can be used to determine if the current executable supports JIT compilation, while `sys._jit.is_enabled()` can be used to tell if JIT compilation has been enabled for the current process.

Currently, the most significant missing functionality is that native debuggers and profilers like `gdb` and `perf` are unable to unwind through JIT frames (Python debuggers and profilers, like **`pdb`** or **`profile`**, continue to work without modification). Free-threaded builds do not support JIT compilation.

Please report any bugs or major performance regressions that you encounter!

Ver también

PEP 744 [https://peps.python.org/pep-0744/]

Porting to Python 3.14

This section lists previously described changes and other bugfixes that may require changes to your code.

Changes in the Python API

- On Unix platforms other than macOS, *forkserver* is now the default start method for **`multiprocessing`** and **`ProcessPoolExecutor`**, instead of *fork*.

  See (1) and (2) for details.

  If you encounter **`NameError`**s or pickling errors coming out of **`multiprocessing`** or **`concurrent.futures`**, see the forkserver restrictions.

  This change does not affect Windows or macOS, where "spawn" remains the default start method.

- **`functools.partial`** is now a method descriptor. Wrap it in **`staticmethod()`** if you want to preserve the old behavior. (Contributed by Serhiy Storchaka and Dominykas Grigonis in gh-121027 [https://github.com/python/cpython/issues/121027].)

- The garbage collector is now incremental, which means that the behavior of **`gc.collect()`** changes slightly:

  - ○ `gc.collect(1)`: Performs an increment of garbage collection, rather than collecting generation 1.
  - ○ Other calls to **`gc.collect()`** are unchanged.

- The **`locale.nl_langinfo()`** function now temporarily sets the `LC_CTYPE` locale in some cases. This temporary change affects other threads. (Contributed by Serhiy Storchaka in gh-69998 [https://github.com/python/cpython/issues/69998].)

- **`types.UnionType`** is now an alias for **`typing.Union`**, causing changes in some behaviors. See above for more details. (Contributed by Jelle Zijlstra in gh-105499 [https://github.com/python/cpython/issues/105499].)

- The runtime behavior of annotations has changed in various ways; see above for details. While most code that interacts with annotations should continue to work, some undocumented details may behave differently.

- As part of making the **`mimetypes`** CLI public, it now exits with `1` on failure instead of `0` and `2` on incorrect command-line parameters instead of `1`. Error messages are now printed to stderr.

- The `\B` pattern in regular expression now matches the empty string when given as the entire pattern, which may cause behavioural changes.

- On FreeBSD, **`sys.platform`** no longer contains the major version number.

Changes in annotations (PEP 649 [https://peps.python.org/pep-0649/] and PEP 749 [https://peps.python.org/pep-0749/])

This section contains guidance on changes that may be needed to annotations or Python code that interacts with or introspects annotations, due to the changes related to deferred evaluation of annotations.

In the majority of cases, working code from older versions of Python will not require any changes.

Implications for annotated code

If you define annotations in your code (for example, for use with a static type checker), then this change probably does not affect you: you can keep writing annotations the same way you did with previous versions of Python.

You will likely be able to remove quoted strings in annotations, which are frequently used for forward references. Similarly, if you use `from __future__ import annotations` to avoid having to write strings in annotations, you may well be able to remove that import once you support only Python 3.14 and newer. However, if you rely on third-party libraries that read annotations, those libraries may need changes to support unquoted annotations before they work as expected.

Implications for readers of `__annotations__`

If your code reads the `__annotations__` attribute on objects, you may want to make changes in order to support code that relies on deferred evaluation of annotations. For example, you may want to use `annotationlib.get_annotations()` with the `FORWARDREF` format, as the `dataclasses` module now does.

The external typing_extensions [https://pypi.org/project/typing_extensions/] package provides partial backports of some of the functionality of the `annotationlib` module, such as the `Format` enum and the `get_annotations()` function. These can be used to write cross-version code that takes advantage of the new behavior in Python 3.14.

Related changes

The changes in Python 3.14 are designed to rework how `__annotations__` works at runtime while minimizing breakage to code that contains annotations in source code and to code that reads `__annotations__`. However, if you rely on undocumented details of the annotation behavior or on private functions in the standard library, there are many ways in which your code may not work in Python 3.14. To safeguard your code against future changes, only use the documented functionality of the `annotationlib` module.

In particular, do not read annotations directly from the namespace dictionary attribute of type objects. Use `annotationlib.get_annotate_from_class_namespace()` during class construction and `annotationlib.get_annotations()` afterwards.

In previous releases, it was sometimes possible to access class annotations from an instance of an annotated class. This behavior was undocumented and accidental, and will no longer work in Python 3.14.

`from __future__ import annotations`

In Python 3.7, PEP 563 [https://peps.python.org/pep-0563/] introduced the `from __future__ import annotations` future statement, which turns all annotations into strings.

However, this statement is now deprecated and it is expected to be removed in a future version of Python. This removal will not happen until after Python 3.13 reaches its end of life in 2029, being the last version of Python without support for deferred evaluation of annotations.

In Python 3.14, the behavior of code using `from __future__ import annotations` is unchanged.

Changes in the C API

- `Py_Finalize()` now deletes all interned strings. This is backwards incompatible to any C extension that holds onto an interned string after a call to `Py_Finalize()` and is then reused after a call to `Py_Initialize()`. Any issues arising from this behavior will normally result in crashes during the execution of the subsequent call to **`Py_Initialize()`** from accessing uninitialized memory. To fix, use an address sanitizer to identify any use-after-free coming from an interned string and deallocate it during module shutdown. (Contributed by Eddie Elizondo in gh-113601 [https://github.com/python/cpython/issues/113601].)

- The Unicode Exception Objects C API now raises a **`TypeError`** if its exception argument is not a **`UnicodeError`** object. (Contributed by Bénédikt Tran in gh-127691 [https://github.com/python/cpython/issues/127691].)

- The interpreter internally avoids some reference count modifications when loading objects onto the operands stack by borrowing references when possible. This can lead to smaller reference count values compared to previous Python versions. C API extensions that checked **`Py_REFCNT()`** of `1` to determine if an function argument is not referenced by any other code should instead use **`PyUnstable_Object_IsUniqueReferencedTemporary()`** as a safer replacement.

- Private functions promoted to public C APIs:
  - ○ `_PyBytes_Join()`: **`PyBytes_Join()`**
  - ○ `_PyLong_IsNegative()`: **`PyLong_IsNegative()`**
  - ○ `_PyLong_IsPositive()`: **`PyLong_IsPositive()`**
  - ○ `_PyLong_IsZero()`: **`PyLong_IsZero()`**
  - ○ `_PyLong_Sign()`: **`PyLong_GetSign()`**
  - ○ `_PyUnicodeWriter_Dealloc()`: **`PyUnicodeWriter_Discard()`**
  - ○ `_PyUnicodeWriter_Finish()`: **`PyUnicodeWriter_Finish()`**
  - ○ `_PyUnicodeWriter_Init()`: use **`PyUnicodeWriter_Create()`**
  - ○ `_PyUnicodeWriter_Prepare()`: (no replacement)
  - ○ `_PyUnicodeWriter_PrepareKind()`: (no replacement)
  - ○ `_PyUnicodeWriter_WriteChar()`: **`PyUnicodeWriter_WriteChar()`**

    - `_PyUnicodeWriter_WriteStr()`: **`PyUnicodeWriter_WriteStr()`**
    - `_PyUnicodeWriter_WriteSubstring()`: **`PyUnicodeWriter_WriteSubstring()`**
    - `_PyUnicode_EQ()`: **`PyUnicode_Equal()`**
    - `_PyUnicode_Equal()`: **`PyUnicode_Equal()`**
    - `_Py_GetConfig()`: **`PyConfig_Get()`** and **`PyConfig_GetInt()`**
    - `_Py_HashBytes()`: **`Py_HashBuffer()`**
    - `_Py_fopen_obj()`: **`Py_fopen()`**
    - `PyMutex_IsLocked()` : **`PyMutex_IsLocked()`**

The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get most of these new functions on Python 3.13 and older.

Notable changes in 3.14.1

- Add **`PyUnstable_ThreadState_SetStackProtection()`** and **`PyUnstable_ThreadState_ResetStackProtection()`** functions to set the stack protection base address and stack protection size of a Python thread state. (Contributed by Victor Stinner in gh-139653 [https://github.com/python/cpython/issues/139653].)

What's New In Python 3.13

Editors:

Adam Turner and Thomas Wouters

This article explains the new features in Python 3.13, compared to 3.12. Python 3.13 was released on October 7, 2024. For full details, see the changelog.

Ver también

PEP 719 [https://peps.python.org/pep-0719/] – Python 3.13 Release Schedule

Summary – Release Highlights

Python 3.13 is a stable release of the Python programming language, with a mix of changes to the language, the implementation and the standard library. The biggest changes include a new interactive interpreter, experimental support for running in a free-threaded mode (**PEP 703** [https://peps.python.org/pep-0703/]), and a Just-In-Time compiler (**PEP 744** [https://peps.python.org/pep-0744/]).

Error messages continue to improve, with tracebacks now highlighted in color by default. The `locals()` builtin now has defined semantics for changing the returned mapping, and type parameters now support default values.

The library changes contain removal of deprecated APIs and modules, as well as the usual improvements in user-friendliness and correctness. Several legacy standard library modules have now been removed following their deprecation in Python 3.11 (**PEP 594** [https://peps.python.org/pep-0594/]).

This article doesn't attempt to provide a complete specification of all new features, but instead gives a convenient overview. For full details refer to the documentation, such as the Library Reference and Language Reference. To understand the complete implementation and design rationale for a change, refer to the PEP for a particular new feature; but note that PEPs usually are not kept up-to-date once a feature has been fully implemented. See Porting to Python 3.13 for guidance on upgrading from earlier versions of Python.

Interpreter improvements:

- A greatly improved interactive interpreter and improved error messages.
- **PEP 667** [https://peps.python.org/pep-0667/]: The `locals()` builtin now has defined semantics when mutating the returned mapping. Python debuggers and similar tools may now more reliably update local variables in optimized scopes even during concurrent code execution.
- **PEP 703** [https://peps.python.org/pep-0703/]: CPython 3.13 has experimental support for running with the global interpreter lock disabled. See Free-threaded CPython for more details.
- **PEP 744** [https://peps.python.org/pep-0744/]: A basic JIT compiler was added. It is currently disabled by default (though we may turn it on later). Performance improvements are modest – we expect to improve this over the next few releases.
- Color support in the new interactive interpreter, as well as in tracebacks and doctest output. This can be disabled through the `PYTHON_COLORS` and `NO_COLOR` [https://no-color.org/] environment variables.

Python data model improvements:

- `__static_attributes__` stores the names of attributes accessed through `self.X` in any function in a class body.
- `__firstlineno__` records the first line number of a class definition.

Significant improvements in the standard library:

- Add a new `PythonFinalizationError` exception, raised when an operation is blocked during finalization.
- The `argparse` module now supports deprecating command-line options, positional arguments, and subcommands.
- The new functions `base64.z85encode()` and `base64.z85decode()` support encoding and decoding Z85 data [https://rfc.zeromq.org/spec/32/].
- The `copy` module now has a `copy.replace()` function, with support for many builtin types and any class defining the `__replace__()` method.
- The new `dbm.sqlite3` module is now the default `dbm` backend.
- The `os` module has a suite of new functions for working with Linux's timer notification file descriptors.
- The `random` module now has a command-line interface.

Security improvements:

- `ssl.create_default_context()` sets `ssl.VERIFY_X509_PARTIAL_CHAIN` and `ssl.VERIFY_X509_STRICT` as default flags.

C API improvements:

- The `Py_mod_gil` slot is now used to indicate that an extension module supports running with the GIL disabled.
- The PyTime C API has been added, providing access to system clocks.
- `PyMutex` is a new lightweight mutex that occupies a single byte.
- There is a new suite of functions for generating **PEP 669** [https://peps.python.org/pep-0669/] monitoring events in the C API.

New typing features:

- **PEP 696** [https://peps.python.org/pep-0696/]: Type parameters (`typing.TypeVar`, `typing.ParamSpec`, and `typing.TypeVarTuple`) now support defaults.
- **PEP 702** [https://peps.python.org/pep-0702/]: The new `warnings.deprecated()` decorator adds support for marking deprecations in the type system and at runtime.

- **PEP 705** [https://peps.python.org/pep-0705/]: `typing.ReadOnly` can be used to mark an item of a `typing.TypedDict` as read-only for type checkers.
- **PEP 742** [https://peps.python.org/pep-0742/]: `typing.TypeIs` provides more intuitive type narrowing behavior, as an alternative to `typing.TypeGuard`.

Platform support:

- **PEP 730** [https://peps.python.org/pep-0730/]: Apple’s iOS is now an officially supported platform, at **tier 3** [https://peps.python.org/pep-0011/#tier-3].
- **PEP 738** [https://peps.python.org/pep-0738/]: Android is now an officially supported platform, at **tier 3** [https://peps.python.org/pep-0011/#tier-3].
- `wasm32-wasi` is now supported as a **tier 2** [https://peps.python.org/pep-0011/#tier-2] platform.
- `wasm32-emscripten` is no longer an officially supported platform.

Important removals:

- PEP 594: The remaining 19 «dead batteries» (legacy stdlib modules) have been removed from the standard library: `aifc`, `audioop`, `cgi`, `cgitb`, `chunk`, `crypt`, `imghdr`, `mailcap`, `msilib`, `nis`, `nntplib`, `ossaudiodev`, `pipes`, `sndhdr`, `spwd`, `sunau`, `telnetlib`, `uu` and `xdrlib`.
- Remove the **2to3** tool and `lib2to3` module (deprecated in Python 3.11).
- Remove the `tkinter.tix` module (deprecated in Python 3.6).
- Remove the `locale.resetlocale()` function.
- Remove the `typing.io` and `typing.re` namespaces.
- Remove chained `classmethod` descriptors.

Release schedule changes:

PEP 602 [https://peps.python.org/pep-0602/] («Annual Release Cycle for Python») has been updated to extend the full support (“bugfix”) period for new releases to two years. This updated policy means that:

- Python 3.9–3.12 have one and a half years of full support, followed by three and a half years of security fixes.
- Python 3.13 and later have two years of full support, followed by three years of security fixes.

New Features

A better interactive interpreter

Python now uses a new interactive shell by default, based on code from the PyPy project [https://pypy.org/]. When the user starts the REPL from an interactive terminal, the following new features are now supported:

- Multiline editing with history preservation.
- Direct support for REPL-specific commands like `help`, `exit`, and `quit`, without the need to call them as functions.
- Prompts and tracebacks with color enabled by default.
- Interactive help browsing using `F1` with a separate command history.
- History browsing using `F2` that skips output as well as the > > > and … prompts.
- «Paste mode» with `F3` that makes pasting larger blocks of code easier (press `F3` again to return to the regular prompt).

To disable the new interactive shell, set the **`PYTHON_BASIC_REPL`** environment variable. For more on interactive mode, see Modo interactivo.

(Contributed by Pablo Galindo Salgado, Łukasz Langa, and Lysandros Nikolaou in gh-111201 [https://github.com/python/cpython/issues/111201] based on code from the PyPy project. Windows support contributed by Dino Viehland and Anthony Shaw.)

Improved error messages

- The interpreter now uses color by default when displaying tracebacks in the terminal. This feature can be controlled via the new `PYTHON_COLORS` environment variable as well as the canonical `NO_COLOR` [https://no-color.org/] and `FORCE_COLOR` [https://force-color.org/] environment variables. (Contributed by Pablo Galindo Salgado in gh-112730 [https://github.com/python/cpython/issues/112730].)

- A common mistake is to write a script with the same name as a standard library module. When this results in errors, we now display a more helpful error message:

  ```
  $ python random.py
  Traceback (most recent call last):
    File "/home/me/random.py", line 1, in <module>
      import random
    File "/home/me/random.py", line 3, in <module>
      print(random.randint(5))
            ^^^^^^^^^^^^^^
  AttributeError: module 'random' has no attribute 'r
  ```

  Similarly, if a script has the same name as a third-party module that it attempts to import and this results in errors, we also display a more helpful error message:

  ```
  $ python numpy.py
  Traceback (most recent call last):
    File "/home/me/numpy.py", line 1, in <module>
      import numpy as np
    File "/home/me/numpy.py", line 3, in <module>
      np.array([1, 2, 3])
      ^^^^^^^^
  AttributeError: module 'numpy' has no attribute 'ar
  ```

  (Contributed by Shantanu Jain in gh-95754 [https://github.com/python/cpython/issues/95754].)

- The error message now tries to suggest the correct keyword argument when an incorrect keyword argument is passed to a function.

  ```
  >>> "Better error messages!".split(max_split=1)
  ```

```
Traceback (most recent call last):
  File "<python-input-0>", line 1, in <module>
    "Better error messages!".split(max_split=1)
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^
TypeError: split() got an unexpected keyword argume
```

(Contributed by Pablo Galindo Salgado and Shantanu Jain in gh-107944 [https://github.com/python/cpython/issues/107944].)

Free-threaded CPython

CPython now has experimental support for running in a free-threaded mode, with the global interpreter lock (GIL) disabled. This is an experimental feature and therefore is not enabled by default. The free-threaded mode requires a different executable, usually called `python3.13t` or `python3.13t.exe`. Pre-built binaries marked as *free-threaded* can be installed as part of the official Windows and macOS installers, or CPython can be built from source with the `--disable-gil` option.

Free-threaded execution allows for full utilization of the available processing power by running threads in parallel on available CPU cores. While not all software will benefit from this automatically, programs designed with threading in mind will run faster on multi-core hardware. **The free-threaded mode is experimental** and work is ongoing to improve it: expect some bugs and a substantial single-threaded performance hit. Free-threaded builds of CPython support optionally running with the GIL enabled at runtime using the environment variable `PYTHON_GIL` or the command-line option `-X gil=1`.

To check if the current interpreter supports free-threading, `python -VV` and `sys.version` contain «experimental free-threading build». The new `sys._is_gil_enabled()` function can be used to check whether the GIL is actually disabled in the running process.

C-API extension modules need to be built specifically for the free-threaded build. Extensions that support running with the GIL disabled should use the `Py_mod_gil` slot. Extensions using single-phase init should use `PyUnstable_Module_SetGIL()` to indicate whether they support running with the GIL disabled. Importing C extensions that don't use these mechanisms will cause the GIL to be enabled, unless the GIL was explicitly disabled with the `PYTHON_GIL` environment variable or the `-X gil=0` option. pip 24.1 or newer is required to install packages with C extensions in the free-threaded build.

This work was made possible thanks to many individuals and organizations, including the large community of contributors to Python and third-party projects to test and enable free-threading support. Notable contributors include: Sam Gross, Ken Jin, Donghee Na, Itamar Oren, Matt Page, Brett Simmers, Dino Viehland, Carl Meyer, Nathan Goldbaum, Ralf Gommers, Lysandros Nikolaou, and many others. Many of these contributors are employed by Meta, which has provided significant engineering resources to support this project.

Ver también

PEP 703 [https://peps.python.org/pep-0703/] «Making the Global Interpreter Lock Optional in CPython» contains rationale and information surrounding this work.

Porting Extension Modules to Support Free-Threading [https://py-free-threading.github.io/porting/]: A community-maintained porting guide for extension authors.

An experimental just-in-time (JIT) compiler

When CPython is configured and built using the **`--enable-experimental-jit`** option, a just-in-time (JIT) compiler is added which may speed up some Python programs. On Windows, use `PCbuild/build.bat --experimental-jit` to enable the JIT or `--experimental-jit-interpreter` to enable the Tier 2 interpreter. Build requirements and further supporting information are contained at [https://github.com/python/cpython/blob/main/Tools/jit/README.md] `Tools/jit/README.md`.

The **`--enable-experimental-jit`** option takes these (optional) values, defaulting to `yes` if **`--enable-experimental-jit`** is present without the optional value.

- `no`: Disable the entire Tier 2 and JIT pipeline.
- `yes`: Enable the JIT. To disable the JIT at runtime, pass the environment variable `PYTHON_JIT=0`.
- `yes-off`: Build the JIT but disable it by default. To enable the JIT at runtime, pass the environment variable `PYTHON_JIT=1`.
- `interpreter`: Enable the Tier 2 interpreter but disable the JIT. The interpreter can be disabled by running with `PYTHON_JIT=0`.

The internal architecture is roughly as follows:

- We start with specialized *Tier 1 bytecode*. See What's new in 3.11 for details.
- When the Tier 1 bytecode gets hot enough, it gets translated to a new purely internal intermediate representation (IR), called the *Tier 2 IR*, and sometimes referred to as micro-ops («uops»).
- The Tier 2 IR uses the same stack-based virtual machine as Tier 1, but the instruction format is better suited to translation to machine code.
- We have several optimization passes for Tier 2 IR, which are applied before it is interpreted or translated to machine code.
- There is a Tier 2 interpreter, but it is mostly intended for debugging the earlier stages of the optimization pipeline. The Tier 2 interpreter can be enabled by configuring Python with `--enable-experimental-jit=interpreter`.
- When the JIT is enabled, the optimized Tier 2 IR is translated to machine code, which is then executed.
- The machine code translation process uses a technique called *copy-and-patch*. It has no runtime dependencies, but there is a new build-time dependency on LLVM.

Ver también

PEP 744 [https://peps.python.org/pep-0744/]

(JIT by Brandt Bucher, inspired by a paper by Haoran Xu and Fredrik Kjolstad. Tier 2 IR by Mark Shannon and Guido van Rossum. Tier 2 optimizer by Ken Jin.)

Defined mutation semantics for `locals()`

Historically, the expected result of mutating the return value of `locals()` has been left to individual Python implementations to define. Starting from Python 3.13, PEP 667 [https://peps.python.org/pep-0667/] standardises the historical behavior of CPython for most code execution scopes, but changes optimized scopes (functions, generators, coroutines, comprehensions, and generator expressions) to explicitly return independent snapshots of the currently assigned local variables, including locally referenced nonlocal variables captured in closures.

This change to the semantics of `locals()` in optimized scopes also affects the default behavior of code execution functions that implicitly target `locals()` if no explicit namespace is provided (such as `exec()` and `eval()`). In previous versions, whether or not changes could be accessed by calling `locals()` after calling the code execution function was implementation-dependent. In CPython specifically, such code would typically appear to work as desired, but could sometimes fail in optimized scopes based on other code (including debuggers and code execution tracing tools) potentially resetting the shared snapshot in that scope. Now, the code will always run against an independent snapshot of the local variables in optimized scopes, and hence the changes will never be visible in subsequent calls to `locals()`. To access the changes made in these cases, an explicit namespace reference must now be passed to the relevant function. Alternatively, it may make sense to update affected code to use a higher level code execution API that returns the resulting code execution namespace (e.g. `runpy.run_path()` when executing Python files from disk).

To ensure debuggers and similar tools can reliably update local variables in scopes affected by this change, `FrameType.f_locals` now returns a write-through proxy to the frame’s local and locally referenced nonlocal variables in these scopes, rather than returning an inconsistently updated shared `dict` instance with undefined runtime semantics.

See **PEP 667** [https://peps.python.org/pep-0667/] for more details, including related C API changes and deprecations. Porting notes are also provided below for the affected Python APIs and C APIs.

(PEP and implementation contributed by Mark Shannon and Tian Gao in gh-74929 [https://github.com/python/cpython/issues/74929]. Documentation updates provided by Guido van Rossum and Alyssa Coghlan.)

Support for mobile platforms

PEP 730 [https://peps.python.org/pep-0730/]: iOS is now a **PEP 11** [https://peps.python.org/pep-0011/] supported platform, with the `arm64-apple-ios` and `arm64-apple-ios-simulator` targets at tier 3 (iPhone and iPad devices released after 2013 and the Xcode iOS simulator running on Apple silicon hardware, respectively). `x86_64-apple-ios-simulator` (the Xcode iOS simulator running on older `x86_64` hardware) is not a tier 3 supported platform, but will have best-effort support. (PEP written and implementation contributed by Russell Keith-Magee in gh-114099 [https://github.com/python/cpython/issues/114099].)

PEP 738 [https://peps.python.org/pep-0738/]: Android is now a **PEP 11** [https://peps.python.org/pep-0011/] supported platform, with the `aarch64-linux-android` and `x86_64-linux-android` targets at tier 3. The 32-bit targets `arm-linux-androideabi` and `i686-linux-android` are not tier 3 supported platforms, but will have best-effort support. (PEP written and implementation contributed by Malcolm Smith in gh-116622 [https://github.com/python/cpython/issues/116622].)

Ver también

PEP 730 [https://peps.python.org/pep-0730/], **PEP 738** [https://peps.python.org/pep-0738/]

Other Language Changes

- The compiler now strips common leading whitespace from every line in a docstring. This reduces the size of the bytecode cache (such as `.pyc` files), with reductions in file size of around 5%, for example in **`sqlalchemy.orm.session`** from SQLAlchemy 2.0. This change affects tools that use docstrings, such as **`doctest`**.

  ```
  >>> def spam():
  ...     """
  ...         This is a docstring with
  ...           leading whitespace.
  ...
  ...         It even has multiple paragraphs!
  ...     """
  ...
  >>> spam.__doc__
  '\nThis is a docstring with\n  leading whitespace.\
  ```

  (Contributed by Inada Naoki in gh-81283 [https://github.com/python/cpython/issues/81283].)

- Annotation scopes within class scopes can now contain lambdas and comprehensions. Comprehensions that are located within class scopes are not inlined into their parent scope.

  ```
  class C[T]:
      type Alias = lambda: T
  ```

  (Contributed by Jelle Zijlstra in gh-109118 [https://github.com/python/cpython/issues/109118] and gh-118160 [https://github.com/python/cpython/issues/118160].)

- Future statements are no longer triggered by relative imports of the **`__future__`** module, meaning that statements of the form `from .__future__ import ...` are now simply standard relative imports, with no special features activated. (Contributed by Jeremiah Gabriel Pascual in gh-118216 [https://github.com/python/cpython/issues/118216].)

- **`global`** declarations are now permitted in **`except`** blocks when that global is used in the `else` block. Previously this raised an erroneous `SyntaxError`. (Contributed by Irit Katriel in gh-111123 [https://github.com/python/cpython/issues/111123].)

- Add `PYTHON_FROZEN_MODULES`, a new environment variable that determines whether frozen modules are ignored by the import machinery, equivalent to the `-X frozen_modules` command-line option. (Contributed by Yilei Yang in gh-111374 [https://github.com/python/cpython/issues/111374].)
- Add support for the perf profiler working without frame pointers [https://en.wikipedia.org/wiki/Call_stack] through the new environment variable `PYTHON_PERF_JIT_SUPPORT` and command-line option `-X perf_jit`. (Contributed by Pablo Galindo in gh-118518 [https://github.com/python/cpython/issues/118518].)
- The location of a `.python_history` file can be changed via the new `PYTHON_HISTORY` environment variable. (Contributed by Levi Sabah, Zackery Spytz and Hugo van Kemenade in gh-73965 [https://github.com/python/cpython/issues/73965].)
- Classes have a new `__static_attributes__` attribute. This is populated by the compiler with a tuple of the class's attribute names which are assigned through `self.<name>` from any function in its body. (Contributed by Irit Katriel in gh-115775 [https://github.com/python/cpython/issues/115775].)
- The compiler now creates a `__firstlineno__` attribute on classes with the line number of the first line of the class definition. (Contributed by Serhiy Storchaka in gh-118465 [https://github.com/python/cpython/issues/118465].)
- The `exec()` and `eval()` builtins now accept the *globals* and *locals* arguments as keywords. (Contributed by Raphael Gaschignard in gh-105879 [https://github.com/python/cpython/issues/105879])

- The `compile()` builtin now accepts a new flag, `ast.PyCF_OPTIMIZED_AST`, which is similar to `ast.PyCF_ONLY_AST` except that the returned AST is optimized according to the value of the *optimize* argument. (Contributed by Irit Katriel in gh-108113 [https://github.com/python/cpython/issues/108113]).

- Add a `__name__` attribute on `property` objects. (Contributed by Eugene Toder in gh-101860 [https://github.com/python/cpython/issues/101860].)

- Add `PythonFinalizationError`, a new exception derived from `RuntimeError` and used to signal when operations are blocked during finalization. The following callables now raise `PythonFinalizationError`, instead of `RuntimeError`:

  - `_thread.start_new_thread()`
  - `os.fork()`
  - `os.forkpty()`
  - `subprocess.Popen`

  (Contributed by Victor Stinner in gh-114570 [https://github.com/python/cpython/issues/114570].)

- Allow the *count* argument of `str.replace()` to be a keyword. (Contributed by Hugo van Kemenade in gh-106487 [https://github.com/python/cpython/issues/106487].)

- Many functions now emit a warning if a boolean value is passed as a file descriptor argument. This can help catch some errors earlier. (Contributed by Serhiy Storchaka in gh-82626 [https://github.com/python/cpython/issues/82626].)

- Added `name` and `mode` attributes for compressed and archived file-like objects in the `bz2`, `lzma`, `tarfile`, and `zipfile` modules. (Contributed by Serhiy Storchaka in gh-115961 [https://github.com/python/cpython/issues/115961].)

New Modules

- `dbm.sqlite3`: An SQLite backend for `dbm`. (Contributed by

Raymond Hettinger and Erlend E. Aasland in gh-100414 [https://github.com/python/cpython/issues/100414].)

Improved Modules

argparse

- Add the *deprecated* parameter to the `add_argument()` and `add_parser()` methods, to enable deprecating command-line options, positional arguments, and subcommands. (Contributed by Serhiy Storchaka in gh-83648 [https://github.com/python/cpython/issues/83648].)

array

- Add the `'w'` type code (`Py_UCS4`) for Unicode characters. It should be used instead of the deprecated `'u'` type code. (Contributed by Inada Naoki in gh-80480 [https://github.com/python/cpython/issues/80480].)
- Register `array.array` as a `MutableSequence` by implementing the `clear()` method. (Contributed by Mike Zimin in gh-114894 [https://github.com/python/cpython/issues/114894].)

ast

- The constructors of node types in the `ast` module are now stricter in the arguments they accept, with more intuitive behavior when arguments are omitted.

  If an optional field on an AST node is not included as an argument when constructing an instance, the field will now be set to `None`. Similarly, if a list field is omitted, that field will now be set to an empty list, and if an `expr_context` field is omitted, it defaults to `Load()`. (Previously, in all cases, the attribute would be missing on the newly constructed AST node instance.)

  In all other cases, where a required argument is omitted, the node constructor will emit a **`DeprecationWarning`**. This will raise an exception in Python 3.15. Similarly, passing a keyword argument to the constructor that does not map to a field on the AST node is now deprecated, and will raise an exception in Python 3.15.

These changes do not apply to user-defined subclasses of **`ast.AST`** unless the class opts in to the new behavior by defining the **`AST._field_types`** mapping.

(Contributed by Jelle Zijlstra in gh-105858 [https://github.com/python/cpython/issues/105858], gh-117486 [https://github.com/python/cpython/issues/117486], and gh-118851 [https://github.com/python/cpython/issues/118851].)

- **`ast.parse()`** now accepts an optional argument *optimize* which is passed on to **`compile()`**. This makes it possible to obtain an optimized AST. (Contributed by Irit Katriel in gh-108113 [https://github.com/python/cpython/issues/108113].)

asyncio

- **`asyncio.as_completed()`** now returns an object that is both an asynchronous iterator and a plain iterator of awaitables. The awaitables yielded by asynchronous iteration include original task or future objects that were passed in, making it easier to associate results with the tasks being completed. (Contributed by Justin Arthur in gh-77714 [https://github.com/python/cpython/issues/77714].)

- **`asyncio.loop.create_unix_server()`** will now automatically remove the Unix socket when the server is closed. (Contributed by Pierre Ossman in gh-111246 [https://github.com/python/cpython/issues/111246].)

- **`DatagramTransport.sendto()`** will now send zero-length datagrams if called with an empty bytes object. The transport flow control also now accounts for the datagram header when calculating the buffer size. (Contributed by Jamie Phan in gh-115199 [https://github.com/python/cpython/issues/115199].)

- Add `Queue.shutdown` and `QueueShutDown` to manage queue termination. (Contributed by Laurie Opperman and Yves Duprat in gh-104228 [https://github.com/python/cpython/issues/104228].)

- Add the `Server.close_clients()` and `Server.abort_clients()` methods, which more forcefully close an asyncio server. (Contributed by Pierre Ossman in gh-113538 [https://github.com/python/cpython/issues/113538].)

- Accept a tuple of separators in `StreamReader.readuntil()`, stopping when any one of them is encountered. (Contributed by Bruce Merry in gh-81322 [https://github.com/python/cpython/issues/81322].)

- Improve the behavior of `TaskGroup` when an external cancellation collides with an internal cancellation. For example, when two task groups are nested and both experience an exception in a child task simultaneously, it was possible that the outer task group would hang, because its internal cancellation was swallowed by the inner task group.

  In the case where a task group is cancelled externally and also must raise an `ExceptionGroup`, it will now call the parent task's `cancel()` method. This ensures that a `CancelledError` will be raised at the next `await`, so the cancellation is not lost.

  An added benefit of these changes is that task groups now preserve the cancellation count (`cancelling()`).

  In order to handle some corner cases, `uncancel()` may now reset the undocumented `_must_cancel` flag when the cancellation count reaches zero.

  (Inspired by an issue reported by Arthur Tacca in gh-116720 [https://github.com/python/cpython/issues/116720].)

- When `TaskGroup.create_task()` is called on an inactive `TaskGroup`, the given coroutine will be closed (which prevents a `RuntimeWarning` about the given coroutine being never awaited). (Contributed by Arthur Tacca and Jason Zhang in gh-115957 [https://github.com/python/cpython/issues/115957].)

- The function and methods named `create_task` have received a new `**kwargs` argument that is passed through to the task constructor. This change was accidentally added in 3.13.3, and broke the API contract for custom task factories. Several third-party task factories implemented workarounds for this. In 3.13.4 and later releases the old factory contract is honored once again (until 3.14). To keep the workarounds working, the extra `**kwargs` argument still allows passing additional keyword arguments to **`Task`** and to custom task factories.

  This affects the following function and methods: **`asyncio.create_task()`**, **`asyncio.loop.create_task()`**, **`asyncio.TaskGroup.create_task()`**. (Contributed by Thomas Grainger in gh-128307 [https://github.com/python/cpython/issues/128307].)

base64

- Add **`z85encode()`** and **`z85decode()`** functions for encoding **`bytes`** as Z85 data [https://rfc.zeromq.org/spec/32/] and decoding Z85-encoded data to **`bytes`**. (Contributed by Matan Perelman in gh-75299 [https://github.com/python/cpython/issues/75299].)

compileall

- The default number of worker threads and processes is now selected using **`os.process_cpu_count()`** instead of **`os.cpu_count()`**. (Contributed by Victor Stinner in gh-109649 [https://github.com/python/cpython/issues/109649].)

concurrent.futures

- The default number of worker threads and processes is now selected using `os.process_cpu_count()` instead of `os.cpu_count()`. (Contributed by Victor Stinner in gh-109649 [https://github.com/python/cpython/issues/109649].)

configparser

- `ConfigParser` now has support for unnamed sections, which allows for top-level key-value pairs. This can be enabled with the new *allow_unnamed_section* parameter. (Contributed by Pedro Sousa Lacerda in gh-66449 [https://github.com/python/cpython/issues/66449].)

copy

- The new `replace()` function and the `replace protocol` make creating modified copies of objects much simpler. This is especially useful when working with immutable objects. The following types support the **`replace()`** function and implement the replace protocol:

  ○ `collections.namedtuple()`
  ○ `dataclasses.dataclass`
  ○ `datetime.datetime`, `datetime.date`, `datetime.time`
  ○ `inspect.Signature`, `inspect.Parameter`
  ○ `types.SimpleNamespace`
  ○ code objects

  Any user-defined class can also support `copy.replace()` by defining the `__replace__()` method. (Contributed by Serhiy Storchaka in gh-108751 [https://github.com/python/cpython/issues/108751].)

ctypes

- As a consequence of necessary internal refactoring, initialization of internal metaclasses now happens in `__init__` rather than in `__new__`. This affects projects that subclass these internal metaclasses to provide custom initialization. Generally:

- ○ Custom logic that was done in `__new__` after calling `super().__new__` should be moved to `__init__`.
- ○ To create a class, call the metaclass, not only the metaclass's `__new__` method.

See gh-124520 [https://github.com/python/cpython/issues/124520] for discussion and links to changes in some affected projects.

- **`ctypes.Structure`** objects have a new **`_align_`** attribute which allows the alignment of the structure being packed to/from memory to be specified explicitly. (Contributed by Matt Sanderson in gh-112433 [https://github.com/python/cpython/issues/112433])

dbm

- Add **`dbm.sqlite3`**, a new module which implements an SQLite backend, and make it the default **`dbm`** backend. (Contributed by Raymond Hettinger and Erlend E. Aasland in gh-100414 [https://github.com/python/cpython/issues/100414].)
- Allow removing all items from the database through the new **`clear()`** methods of the GDBM and NDBM database objects. (Contributed by Donghee Na in gh-107122 [https://github.com/python/cpython/issues/107122].)

dis

- Change the output of **`dis`** module functions to show logical labels for jump targets and exception handlers, rather than offsets. The offsets can be added with the new **`-O`** command-line option or the *show_offsets* argument. (Contributed by Irit Katriel in gh-112137 [https://github.com/python/cpython/issues/112137].)
- **`get_instructions()`** no longer represents cache entries as separate instructions. Instead, it returns them as part of the **`Instruction`**, in the new *cache_info* field. The *show_caches* argument to **`get_instructions()`** is deprecated and no longer has any effect. (Contributed by Irit Katriel in gh-112962 [https://github.com/python/cpython/issues/112962].)

doctest

- **`doctest`** output is now colored by default. This can be controlled via the new **`PYTHON_COLORS`** environment variable as well as the canonical **`NO_COLOR`** [https://no-color.org/] and **`FORCE_COLOR`** [https://force-color.org/] environment variables. See also Controlling color. (Contributed by Hugo van Kemenade in gh-117225 [https://github.com/python/cpython/issues/117225].)
- The **`DocTestRunner.run()`** method now counts the number of skipped tests. Add the **`DocTestRunner.skips`** and **`TestResults.skipped`** attributes. (Contributed by Victor Stinner in gh-108794 [https://github.com/python/cpython/issues/108794].)

email

- Headers with embedded newlines are now quoted on output. The **`generator`** will now refuse to serialize (write) headers that are improperly folded or delimited, such that they would be parsed as multiple headers or joined with adjacent data. If you need to turn this safety feature off, set **`verify_generated_headers`**. (Contributed by Bas Bloemsaat and Petr Viktorin in gh-121650 [https://github.com/python/cpython/issues/121650].)
- **`getaddresses()`** and **`parseaddr()`** now return `('', '')` pairs in more situations where invalid email addresses are encountered instead of potentially inaccurate values. The two functions have a new optional *strict* parameter (default `True`). To get the old behavior (accepting malformed input), use `strict=False`. `getattr(email.utils, 'supports_strict_parsing', False)` can be used to check if the *strict* parameter is available. (Contributed by Thomas Dwyer and Victor Stinner for gh-102988 [https://github.com/python/cpython/issues/102988] to improve the **CVE 2023-27043** [https://www.cve.org/CVERecord?id=CVE-2023-27043] fix.)

enum

- `EnumDict` has been made public to better support subclassing `EnumType`.

fractions

- `Fraction` objects now support the standard format specification mini-language rules for fill, alignment, sign handling, minimum width, and grouping. (Contributed by Mark Dickinson in gh-111320 [https://github.com/python/cpython/issues/111320].)

glob

- Add `translate()`, a function to convert a path specification with shell-style wildcards to a regular expression. (Contributed by Barney Gale in gh-72904 [https://github.com/python/cpython/issues/72904].)

importlib

- The following functions in `importlib.resources` now allow accessing a directory (or tree) of resources, using multiple positional arguments (the *encoding* and *errors* arguments in the text-reading functions are now keyword-only):

    ○ `is_resource()`
    ○ `open_binary()`
    ○ `open_text()`
    ○ `path()`
    ○ `read_binary()`
    ○ `read_text()`

  These functions are no longer deprecated and are not scheduled for removal. (Contributed by Petr Viktorin in gh-116608 [https://github.com/python/cpython/issues/116608].)

- `contents()` remains deprecated in favor of the fully-featured `Traversable` API. However, there is now no plan to remove it. (Contributed by Petr Viktorin in gh-116608

[https://github.com/python/cpython/issues/116608].)

io

- The **`IOBase`** finalizer now logs any errors raised by the **`close()`** method with **`sys.unraisablehook`**. Previously, errors were ignored silently by default, and only logged in Python Development Mode or when using a Python debug build. (Contributed by Victor Stinner in gh-62948 [https://github.com/python/cpython/issues/62948].)

ipaddress

- Add the **`IPv4Address.ipv6_mapped`** property, which returns the IPv4-mapped IPv6 address. (Contributed by Charles Machalow in gh-109466 [https://github.com/python/cpython/issues/109466].)
- Fix `is_global` and `is_private` behavior in **`IPv4Address`**, **`IPv6Address`**, **`IPv4Network`**, and **`IPv6Network`**. (Contributed by Jakub Stasiak in gh-113171 [https://github.com/python/cpython/issues/113171].)

itertools

- **`batched()`** has a new *strict* parameter, which raises a **`ValueError`** if the final batch is shorter than the specified batch size. (Contributed by Raymond Hettinger in gh-113202 [https://github.com/python/cpython/issues/113202].)

marshal

- Add the *allow_code* parameter in module functions. Passing `allow_code=False` prevents serialization and de-serialization of code objects which are incompatible between Python versions. (Contributed by Serhiy Storchaka in gh-113626 [https://github.com/python/cpython/issues/113626].)

math

- The new function **`fma()`** performs fused multiply-add operations. This computes `x * y + z` with only a single round, and so avoids any intermediate loss of precision. It wraps the `fma()` function provided by C99, and follows the specification of the IEEE 754 «fusedMultiplyAdd» operation for special cases. (Contributed by Mark Dickinson and Victor Stinner in gh-73468 [https://github.com/python/cpython/issues/73468].)

mimetypes

- Add the **`guess_file_type()`** function to guess a MIME type from a filesystem path. Using paths with **`guess_type()`** is now soft deprecated. (Contributed by Serhiy Storchaka in gh-66543 [https://github.com/python/cpython/issues/66543].)

mmap

- **`mmap`** is now protected from crashing on Windows when the mapped memory is inaccessible due to file system errors or access violations. (Contributed by Jannis Weigend in gh-118209 [https://github.com/python/cpython/issues/118209].)
- **`mmap`** has a new **`seekable()`** method that can be used when a seekable file-like object is required. The **`seek()`** method now returns the new absolute position. (Contributed by Donghee Na and Sylvie Liberman in gh-111835 [https://github.com/python/cpython/issues/111835].)
- The new UNIX-only *trackfd* parameter for **`mmap`** controls file descriptor duplication; if false, the file descriptor specified by *fileno* will not be duplicated. (Contributed by Zackery Spytz and Petr Viktorin in gh-78502 [https://github.com/python/cpython/issues/78502].)

multiprocessing

- The default number of worker threads and processes is now selected using **`os.process_cpu_count()`** instead of **`os.cpu_count()`**. (Contributed by Victor Stinner in gh-109649 [https://github.com/python/cpython/issues/109649].)

os

- Add **`process_cpu_count()`** function to get the number of logical CPU cores usable by the calling thread of the current process. (Contributed by Victor Stinner in gh-109649 [https://github.com/python/cpython/issues/109649].)
- **`cpu_count()`** and **`process_cpu_count()`** can be overridden through the new environment variable **`PYTHON_CPU_COUNT`** or the new command-line option **`-X cpu_count`**. This option is useful for users who need to limit CPU resources of a container system without having to modify application code or the container itself. (Contributed by Donghee Na in gh-109595 [https://github.com/python/cpython/issues/109595].)
- Add a low level interface to Linux's *timer file descriptors [https://manpages.debian.org/timerfd_create(2)]* via **`timerfd_create()`**, **`timerfd_settime()`**, **`timerfd_settime_ns()`**, **`timerfd_gettime()`**, **`timerfd_gettime_ns()`**, **`TFD_NONBLOCK`**, **`TFD_CLOEXEC`**, **`TFD_TIMER_ABSTIME`**, and **`TFD_TIMER_CANCEL_ON_SET`** (Contributed by Masaru Tsuchiyama in gh-108277 [https://github.com/python/cpython/issues/108277].)
- **`lchmod()`** and the *follow_symlinks* argument of **`chmod()`** are both now available on Windows. Note that the default value of *follow_symlinks* in **`lchmod()`** is `False` on Windows. (Contributed by Serhiy Storchaka in gh-59616 [https://github.com/python/cpython/issues/59616].)
- **`fchmod()`** and support for file descriptors in **`chmod()`** are both now available on Windows. (Contributed by Serhiy Storchaka in gh-113191 [https://github.com/python/cpython/issues/113191].)
- On Windows, **`mkdir()`** and **`makedirs()`** now support passing a *mode* value of `0o700` to apply access control to the new directory. This implicitly affects **`tempfile.mkdtemp()`** and is a mitigation for **CVE 2024-4030** [https://www.cve.org/CVERecord?id=CVE-2024-4030]. Other values for *mode* continue to be ignored. (Contributed by Steve Dower in gh-118486 [https://github.com/python/cpython/issues/118486].)
- **`posix_spawn()`** now accepts `None` for the *env* argument,

which makes the newly spawned process use the current process environment. (Contributed by Jakub Kulik in gh-113119 [https://github.com/python/cpython/issues/113119].)

- `posix_spawn()` can now use the `POSIX_SPAWN_CLOSEFROM` attribute in the *file_actions* parameter on platforms that support `posix_spawn_file_actions_addclosefrom_np()`. (Contributed by Jakub Kulik in gh-113117 [https://github.com/python/cpython/issues/113117].)

os.path

- Add `isreserved()` to check if a path is reserved on the current system. This function is only available on Windows. (Contributed by Barney Gale in gh-88569 [https://github.com/python/cpython/issues/88569].)
- On Windows, `isabs()` no longer considers paths starting with exactly one slash (`\` or `/`) to be absolute. (Contributed by Barney Gale and Jon Foster in gh-44626 [https://github.com/python/cpython/issues/44626].)
- `realpath()` now resolves MS-DOS style file names even if the file is not accessible. (Contributed by Moonsik Park in gh-82367 [https://github.com/python/cpython/issues/82367].)

pathlib

- Add `UnsupportedOperation`, which is raised instead of `NotImplementedError` when a path operation isn't supported. (Contributed by Barney Gale in gh-89812 [https://github.com/python/cpython/issues/89812].)
- Add a new constructor for creating `Path` objects from "file" URIs (`file:///`), `Path.from_uri()`. (Contributed by Barney Gale in gh-107465 [https://github.com/python/cpython/issues/107465].)
- Add `PurePath.full_match()` for matching paths with shell-style wildcards, including the recursive wildcard «`**`». (Contributed by Barney Gale in gh-73435 [https://github.com/python/cpython/issues/73435].)
- Add the `PurePath.parser` class attribute to store the implementation of `os.path` used for low-level path parsing

and joining. This will be either `posixpath` or `ntpath`.

- Add *recurse_symlinks* keyword-only argument to `Path.glob()` and `rglob()`. (Contributed by Barney Gale in gh-77609 [https://github.com/python/cpython/issues/77609].)
- `Path.glob()` and `rglob()` now return files and directories when given a pattern that ends with «`**`». Previously, only directories were returned. (Contributed by Barney Gale in gh-70303 [https://github.com/python/cpython/issues/70303].)
- Add the *follow_symlinks* keyword-only argument to `Path.is_file`, `Path.is_dir`, `Path.owner()`, and `Path.group()`. (Contributed by Barney Gale in gh-105793 [https://github.com/python/cpython/issues/105793] and Kamil Turek in gh-107962 [https://github.com/python/cpython/issues/107962].)

pdb

- `breakpoint()` and `set_trace()` now enter the debugger immediately rather than on the next line of code to be executed. This change prevents the debugger from breaking outside of the context when `breakpoint()` is positioned at the end of the context. (Contributed by Tian Gao in gh-118579 [https://github.com/python/cpython/issues/118579].)
- `sys.path[0]` is no longer replaced by the directory of the script being debugged when `sys.flags.safe_path` is set. (Contributed by Tian Gao and Christian Walther in gh-111762 [https://github.com/python/cpython/issues/111762].)
- `zipapp` is now supported as a debugging target. (Contributed by Tian Gao in gh-118501 [https://github.com/python/cpython/issues/118501].)
- Add ability to move between chained exceptions during post-mortem debugging in `pm()` using the new `exceptions [exc_number]` command for Pdb. (Contributed by Matthias Bussonnier in gh-106676 [https://github.com/python/cpython/issues/106676].)
- Expressions and statements whose prefix is a pdb command are now correctly identified and executed. (Contributed by Tian Gao in gh-108464 [https://github.com/python/cpython/issues/108464].)

queue

- Add **`Queue.shutdown`** and **`ShutDown`** to manage queue termination. (Contributed by Laurie Opperman and Yves Duprat in gh-104750 [https://github.com/python/cpython/issues/104750].)

random

- Add a command-line interface. (Contributed by Hugo van Kemenade in gh-118131 [https://github.com/python/cpython/issues/118131].)

re

- Rename **`re.error`** to **`PatternError`** for improved clarity. **`re.error`** is kept for backward compatibility.

shutil

- Support the *dir_fd* and *follow_symlinks* keyword arguments in **`chown()`**. (Contributed by Berker Peksag and Tahia K in gh-62308 [https://github.com/python/cpython/issues/62308])

site

- `.pth` files are now decoded using UTF-8 first, and then with the locale encoding if UTF-8 decoding fails. (Contributed by Inada Naoki in gh-117802 [https://github.com/python/cpython/issues/117802].)

sqlite3

- A **`ResourceWarning`** is now emitted if a **`Connection`** object is not **`closed`** explicitly. (Contributed by Erlend E. Aasland in gh-105539 [https://github.com/python/cpython/issues/105539].)
- Add the *filter* keyword-only parameter to **`Connection.iterdump()`** for filtering database objects to dump. (Contributed by Mariusz Felisiak in gh-91602 [https://github.com/python/cpython/issues/91602].)

ssl

- The `create_default_context()` API now includes `VERIFY_X509_PARTIAL_CHAIN` and `VERIFY_X509_STRICT` in its default flags.

  Nota

  `VERIFY_X509_STRICT` may reject pre-RFC 5280 [https://datatracker.ietf.org/doc/html/rfc5280.html] or malformed certificates that the underlying OpenSSL implementation might otherwise accept. Whilst disabling this is not recommended, you can do so using:

  ```
  import ssl

  ctx = ssl.create_default_context()
  ctx.verify_flags &= ~ssl.VERIFY_X509_STRICT
  ```

  (Contributed by William Woodruff in gh-112389 [https://github.com/python/cpython/issues/112389].)

statistics

- Add `kde()` for kernel density estimation. This makes it possible to estimate a continuous probability density function from a fixed number of discrete samples. (Contributed by Raymond Hettinger in gh-115863 [https://github.com/python/cpython/issues/115863].)
- Add `kde_random()` for sampling from an estimated probability density function created by `kde()`. (Contributed by Raymond Hettinger in gh-115863 [https://github.com/python/cpython/issues/115863].)

subprocess

- The `subprocess` module now uses the `posix_spawn()` function in more situations.

Notably, when *close_fds* is `True` (the default), **`posix_spawn()`** will be used when the C library provides **`posix_spawn_file_actions_addclosefrom_np()`**, which includes recent versions of Linux, FreeBSD, and Solaris. On Linux, this should perform similarly to the existing Linux **`vfork()`** based code.

A private control knob **`subprocess._USE_POSIX_SPAWN`** can be set to `False` if you need to force **`subprocess`** to never use **`posix_spawn()`**. Please report your reason and platform details in the issue tracker if you set this so that we can improve our API selection logic for everyone. (Contributed by Jakub Kulik in gh-113117 [https://github.com/python/cpython/issues/113117].)

sys

- Add the **`_is_interned()`** function to test if a string was interned. This function is not guaranteed to exist in all implementations of Python. (Contributed by Serhiy Storchaka in gh-78573 [https://github.com/python/cpython/issues/78573].)

tempfile

- On Windows, the default mode `0o700` used by **`tempfile.mkdtemp()`** now limits access to the new directory due to changes to **`os.mkdir()`**. This is a mitigation for **CVE 2024-4030** [https://www.cve.org/CVERecord?id=CVE-2024-4030]. (Contributed by Steve Dower in gh-118486 [https://github.com/python/cpython/issues/118486].)

time

- On Windows, **`monotonic()`** now uses the `QueryPerformanceCounter()` clock for a resolution of 1 microsecond, instead of the `GetTickCount64()` clock which has a resolution of 15.6 milliseconds. (Contributed by Victor Stinner in gh-88494 [https://github.com/python/cpython/issues/88494].)
- On Windows, **`time()`** now uses the

`GetSystemTimePreciseAsFileTime()` clock for a resolution of 1 microsecond, instead of the `GetSystemTimeAsFileTime()` clock which has a resolution of 15.6 milliseconds. (Contributed by Victor Stinner in gh-63207 [https://github.com/python/cpython/issues/63207].)

tkinter

- Add **`tkinter`** widget methods: **`tk_busy_hold()`**, **`tk_busy_configure()`**, **`tk_busy_cget()`**, **`tk_busy_forget()`**, **`tk_busy_current()`**, and **`tk_busy_status()`**. (Contributed by Miguel, klappnase and Serhiy Storchaka in gh-72684 [https://github.com/python/cpython/issues/72684].)
- The **`tkinter`** widget method **`wm_attributes()`** now accepts the attribute name without the minus prefix to get window attributes, for example `w.wm_attributes('alpha')` and allows specifying attributes and values to set as keyword arguments, for example `w.wm_attributes(alpha=0.5)`. (Contributed by Serhiy Storchaka in gh-43457 [https://github.com/python/cpython/issues/43457].)
- **`wm_attributes()`** can now return attributes as a **`dict`**, by using the new optional keyword-only parameter *return_python_dict*. (Contributed by Serhiy Storchaka in gh-43457 [https://github.com/python/cpython/issues/43457].)
- **`Text.count()`** can now return a simple **`int`** when the new optional keyword-only parameter *return_ints* is used. Otherwise, the single count is returned as a 1-tuple or `None`. (Contributed by Serhiy Storchaka in gh-97928 [https://github.com/python/cpython/issues/97928].)
- Support the «vsapi» element type in the **`element_create()`** method of **`tkinter.ttk.Style`**. (Contributed by Serhiy Storchaka in gh-68166 [https://github.com/python/cpython/issues/68166].)
- Add the **`after_info()`** method for Tkinter widgets. (Contributed by Cheryl Sabella in gh-77020 [https://github.com/python/cpython/issues/77020].)
- Add a new **`copy_replace()`** method to **`PhotoImage`** to

copy a region from one image to another, possibly with pixel zooming, subsampling, or both. (Contributed by Serhiy Storchaka in gh-118225 [https://github.com/python/cpython/issues/118225].)

- Add *from_coords* parameter to the `PhotoImage` methods `copy()`, `zoom()` and `subsample()`. Add *zoom* and *subsample* parameters to the `PhotoImage` method `copy()`. (Contributed by Serhiy Storchaka in gh-118225 [https://github.com/python/cpython/issues/118225].)
- Add the `PhotoImage` methods `read()` to read an image from a file and `data()` to get the image data. Add *background* and *grayscale* parameters to the `write()` method. (Contributed by Serhiy Storchaka in gh-118271 [https://github.com/python/cpython/issues/118271].)

traceback

- Add the `exc_type_str` attribute to `TracebackException`, which holds a string display of the *exc_type*. Deprecate the `exc_type` attribute, which holds the type object itself. Add parameter *save_exc_type* (default `True`) to indicate whether `exc_type` should be saved. (Contributed by Irit Katriel in gh-112332 [https://github.com/python/cpython/issues/112332].)
- Add a new *show_group* keyword-only parameter to `TracebackException.format_exception_only()` to (recursively) format the nested exceptions of a `BaseExceptionGroup` instance. (Contributed by Irit Katriel in gh-105292 [https://github.com/python/cpython/issues/105292].)

types

- `SimpleNamespace` can now take a single positional argument to initialise the namespace's arguments. This argument must either be a mapping or an iterable of key-value pairs. (Contributed by Serhiy Storchaka in gh-108191 [https://github.com/python/cpython/issues/108191].)

typing

- PEP 705 [https://peps.python.org/pep-0705/]: Add `ReadOnly`, a special typing construct to mark a `TypedDict` item as read-only for type checkers.
- PEP 742 [https://peps.python.org/pep-0742/]: Add `TypeIs`, a typing construct that can be used to instruct a type checker how to narrow a type.
- Add `NoDefault`, a sentinel object used to represent the defaults of some parameters in the `typing` module. (Contributed by Jelle Zijlstra in gh-116126 [https://github.com/python/cpython/issues/116126].)
- Add `get_protocol_members()` to return the set of members defining a `typing.Protocol`. (Contributed by Jelle Zijlstra in gh-104873 [https://github.com/python/cpython/issues/104873].)
- Add `is_protocol()` to check whether a class is a `Protocol`. (Contributed by Jelle Zijlstra in gh-104873 [https://github.com/python/cpython/issues/104873].)
- `ClassVar` can now be nested in `Final`, and vice versa. (Contributed by Mehdi Drissi in gh-89547 [https://github.com/python/cpython/issues/89547].)

unicodedata

- Update the Unicode database to version 15.1.0 [https://www.unicode.org/versions/Unicode15.1.0/]. (Contributed by James Gerity in gh-109559 [https://github.com/python/cpython/issues/109559].)

venv

- Add support for creating source control management (SCM) ignore files in a virtual environment's directory. By default, Git is supported. This is implemented as opt-in via the API, which can be extended to support other SCMs (`EnvBuilder` and `create()`), and opt-out via the CLI, using **`--without-scm-ignore-files`**. (Contributed by Brett Cannon in gh-108125 [https://github.com/python/cpython/issues/108125].)

warnings

- PEP 702 [https://peps.python.org/pep-0702/]: The new `warnings.deprecated()` decorator provides a way to communicate deprecations to a static type checker and to warn on usage of deprecated classes and functions. A `DeprecationWarning` may also be emitted when a decorated function or class is used at runtime. (Contributed by Jelle Zijlstra in gh-104003 [https://github.com/python/cpython/issues/104003].)

xml

- Allow controlling Expat > =2.6.0 reparse deferral (**CVE 2023-52425** [https://www.cve.org/CVERecord?id=CVE-2023-52425]) by adding five new methods:
  - `xml.etree.ElementTree.XMLParser.flush()`
  - `xml.etree.ElementTree.XMLPullParser.flush()`
  - `xml.parsers.expat.xmlparser.GetReparseDeferralE`
  - `xml.parsers.expat.xmlparser.SetReparseDeferralE`
  - `xml.sax.expatreader.ExpatParser.flush()`

  (Contributed by Sebastian Pipping in gh-115623 [https://github.com/python/cpython/issues/115623].)

- Add the `close()` method for the iterator returned by `iterparse()` for explicit cleanup. (Contributed by Serhiy Storchaka in gh-69893 [https://github.com/python/cpython/issues/69893].)

zipimport

- Add support for ZIP64 [https://en.wikipedia.org/wiki/Zip_(file_format)#ZIP64] format files. Everybody loves huge data, right? (Contributed by Tim Hatch in gh-94146 [https://github.com/python/cpython/issues/94146].)

Optimizations

- Several standard library modules have had their import times significantly improved. For example, the import time of the `typing` module has been reduced by around a third by removing dependencies on `re` and `contextlib`. Other modules to enjoy import-time speedups include `email.utils`, `enum`, `functools`, `importlib.metadata`, and `threading`. (Contributed by Alex Waygood, Shantanu Jain, Adam Turner, Daniel Hollas, and others in gh-109653 [https://github.com/python/cpython/issues/109653].)

- `textwrap.indent()` is now around 30% faster than before for large input. (Contributed by Inada Naoki in gh-107369 [https://github.com/python/cpython/issues/107369].)
- The `subprocess` module now uses the `posix_spawn()` function in more situations, including when *close_fds* is `True` (the default) on many modern platforms. This should provide a notable performance increase when launching processes on FreeBSD and Solaris. See the subprocess section above for details. (Contributed by Jakub Kulik in gh-113117 [https://github.com/python/cpython/issues/113117].)

Removed Modules And APIs

PEP 594: Remove «dead batteries» from the standard library

PEP 594 [https://peps.python.org/pep-0594/] proposed removing 19 modules from the standard library, colloquially referred to as “dead batteries” due to their historic, obsolete, or insecure status. All of the following modules were deprecated in Python 3.11, and are now removed:

- **`aifc`**
    - ○ standard-aifc [https://pypi.org/project/standard-aifc/]: Use the redistribution of `aifc` library from PyPI.
- **`audioop`**
    - ○ audioop-lts [https://pypi.org/project/audioop-lts/]: Use `audioop-lts` library from PyPI.

- **`chunk`**
  - standard-chunk [https://pypi.org/project/standard-chunk/]: Use the redistribution of `chunk` library from PyPI.
- **`cgi`** and **`cgitb`**
  - **`cgi.FieldStorage`** can typically be replaced with **`urllib.parse.parse_qsl()`** for `GET` and `HEAD` requests, and the **`email.message`** module or the multipart [https://pypi.org/project/multipart/] library for `POST` and `PUT` requests.
  - **`cgi.parse()`** can be replaced by calling **`urllib.parse.parse_qs()`** directly on the desired query string, unless the input is `multipart/form-data`, which should be replaced as described below for **`cgi.parse_multipart()`**.
  - **`cgi.parse_header()`** can be replaced with the functionality in the **`email`** package, which implements the same MIME RFCs. For example, with **`email.message.EmailMessage`**:

    ```
    from email.message import EmailMessage

    msg = EmailMessage()
    msg['content-type'] = 'application/json; charse
    main, params = msg.get_content_type(), msg['cor
    ```

  - **`cgi.parse_multipart()`** can be replaced with the functionality in the **`email`** package, which implements the same MIME RFCs, or with the multipart [https://pypi.org/project/multipart/] library. For example, the **`email.message.EmailMessage`** and **`email.message.Message`** classes.
  - standard-cgi [https://pypi.org/project/standard-cgi/]: and standard-cgitb [https://pypi.org/project/standard-cgitb/]: Use the redistribution of `cgi` and `cgitb` library from PyPI.

- **`crypt`** and the private **`_crypt`** extension. The **`hashlib`** module may be an appropriate replacement when simply hashing a value is required. Otherwise, various third-party libraries on PyPI are available:
    - bcrypt [https://pypi.org/project/bcrypt/]: Modern password hashing for your software and your servers.
    - argon2-cffi [https://pypi.org/project/argon2-cffi/]: The secure Argon2 password hashing algorithm.
    - legacycrypt [https://pypi.org/project/legacycrypt/]: **`ctypes`** wrapper to the POSIX crypt library call and associated functionality.
    - crypt_r [https://pypi.org/project/crypt_r/]: Fork of the **`crypt`** module, wrapper to the *crypt_r(3) [https://manpages.debian.org/crypt_r(3)]* library call and associated functionality.
    - standard-crypt [https://pypi.org/project/standard-crypt/] and deprecated-crypt-alternative [https://pypi.org/project/deprecated-crypt-alternative/]: Use the redistribution of `crypt` and reimplementation of `_crypt` libraries from PyPI.

- **`imghdr`**: The filetype [https://pypi.org/project/filetype/], puremagic [https://pypi.org/project/puremagic/], or python-magic [https://pypi.org/project/python-magic/] libraries should be used as replacements. For example, the **`puremagic.what()`** function can be used to replace the **`imghdr.what()`** function for all file formats that were supported by **`imghdr`**.
    - standard-imghdr [https://pypi.org/project/standard-imghdr/]: Use the redistribution of `imghdr` library from PyPI.

- **`mailcap`**: Use the **`mimetypes`** module instead.
    - standard-mailcap [https://pypi.org/project/standard-mailcap/]: Use the redistribution of `mailcap` library from PyPI.

- **`msilib`**

- **`nis`**

- **`nntplib`**: Use the pynntp [https://pypi.org/project/pynntp/] library from PyPI instead.

    - standard-nntplib [https://pypi.org/project/standard-nntplib/]: Use the redistribution of `nntplib` library from PyPI.

- **`ossaudiodev`**: For audio playback, use the pygame [https://pypi.org/project/pygame/] library from PyPI instead.

- **`pipes`**: Use the **`subprocess`** module instead. Use **`shlex.quote()`** to replace the undocumented `pipes.quote` function.

    - standard-pipes [https://pypi.org/project/standard-pipes/]: Use the redistribution of `pipes` library from PyPI.

- **`sndhdr`**: The filetype [https://pypi.org/project/filetype/], puremagic [https://pypi.org/project/puremagic/], or python-magic [https://pypi.org/project/python-magic/] libraries should be used as replacements.

    - standard-sndhdr [https://pypi.org/project/standard-sndhdr/]: Use the redistribution of `sndhdr` library from PyPI.

- **`spwd`**: Use the python-pam [https://pypi.org/project/python-pam/] library from PyPI instead.

- **`sunau`**

    - standard-sunau [https://pypi.org/project/standard-sunau/]: Use the redistribution of `sunau` library from PyPI.

- **`telnetlib`**, Use the telnetlib3 [https://pypi.org/project/telnetlib3/] or Exscript [https://pypi.org/project/Exscript/] libraries from PyPI instead.

    - standard-telnetlib [https://pypi.org/project/standard-telnetlib/]: Use the redistribution of `telnetlib` library from PyPI.

- **`uu`**: Use the **`base64`** module instead, as a modern alternative.

    - standard-uu [https://pypi.org/project/standard-uu/]: Use the redistribution of `uu` library from PyPI.

- **`xdrlib`**
    - standard-xdrlib [https://pypi.org/project/standard-xdrlib/]: Use the redistribution of `xdrlib` library from PyPI.

(Contributed by Victor Stinner and Zachary Ware in gh-104773 [https://github.com/python/cpython/issues/104773] and gh-104780 [https://github.com/python/cpython/issues/104780].)

2to3

- Remove the **2to3** program and the **`lib2to3`** module, previously deprecated in Python 3.11. (Contributed by Victor Stinner in gh-104780 [https://github.com/python/cpython/issues/104780].)

builtins

- Remove support for chained **`classmethod`** descriptors (introduced in gh-63272 [https://github.com/python/cpython/issues/63272]). These can no longer be used to wrap other descriptors, such as **`property`**. The core design of this feature was flawed and led to several problems. To «pass-through» a **`classmethod`**, consider using the **`__wrapped__`** attribute that was added in Python 3.10. (Contributed by Raymond Hettinger in gh-89519 [https://github.com/python/cpython/issues/89519].)
- Raise a **`RuntimeError`** when calling **`frame.clear()`** on a suspended frame (as has always been the case for an executing frame). (Contributed by Irit Katriel in gh-79932 [https://github.com/python/cpython/issues/79932].)

configparser

- Remove the undocumented **`LegacyInterpolation`** class, deprecated in the docstring since Python 3.2, and at runtime since Python 3.11. (Contributed by Hugo van Kemenade in

gh-104886 [https://github.com/python/cpython/issues/104886].)

importlib.metadata

- Remove deprecated subscript (`__getitem__()`) access for EntryPoint objects. (Contributed by Jason R. Coombs in gh-113175 [https://github.com/python/cpython/issues/113175].)

locale

- Remove the **`locale.resetlocale()`** function, deprecated in Python 3.11. Use `locale.setlocale(locale.LC_ALL, "")` instead. (Contributed by Victor Stinner in gh-104783 [https://github.com/python/cpython/issues/104783].)

opcode

- Move **`opcode.ENABLE_SPECIALIZATION`** to **`_opcode.ENABLE_SPECIALIZATION`**. This field was added in 3.12, it was never documented, and is not intended for external use. (Contributed by Irit Katriel in gh-105481 [https://github.com/python/cpython/issues/105481].)
- Remove **`opcode.is_pseudo()`**, **`opcode.MIN_PSEUDO_OPCODE`**, and **`opcode.MAX_PSEUDO_OPCODE`**, which were added in Python 3.12, but were neither documented nor exposed through `dis`, and were not intended to be used externally. (Contributed by Irit Katriel in gh-105481 [https://github.com/python/cpython/issues/105481].)

optparse

- This module is no longer considered soft deprecated. While `argparse` remains preferred for new projects that aren't using a third party command line argument processing library, there are aspects of the way `argparse` works that mean the lower level `optparse` module may provide a better foundation for *writing* argument processing libraries, and for implementing command line applications which adhere more strictly than `argparse` does to various Unix command line processing conventions that originate in the behaviour of the C **`getopt()`** function . (Contributed by Alyssa Coghlan and Serhiy Storchaka in gh-126180 [https://github.com/python/cpython/issues/126180].)

pathlib

- Remove the ability to use **`Path`** objects as context managers. This functionality was deprecated and has had no effect since Python 3.9. (Contributed by Barney Gale in gh-83863 [https://github.com/python/cpython/issues/83863].)

re

- Remove the undocumented, deprecated, and broken **`re.template()`** function and **`re.TEMPLATE`** / **`re.T`** flag. (Contributed by Serhiy Storchaka and Nikita Sobolev in gh-105687 [https://github.com/python/cpython/issues/105687].)

tkinter.tix

- Remove the **`tkinter.tix`** module, deprecated in Python 3.6. The third-party Tix library which the module wrapped is unmaintained. (Contributed by Zachary Ware in gh-75552 [https://github.com/python/cpython/issues/75552].)

turtle

- Remove the **`RawTurtle.settiltangle()`** method, deprecated in the documentation since Python 3.1 and at runtime since Python 3.11. (Contributed by Hugo van Kemenade in gh-104876 [https://github.com/python/cpython/issues/104876].)

typing

- Remove the **`typing.io`** and **`typing.re`** namespaces, deprecated since Python 3.8. The items in those namespaces can be imported directly from the `typing` module. (Contributed by Sebastian Rittau in gh-92871 [https://github.com/python/cpython/issues/92871].)

- Remove the keyword-argument method of creating `TypedDict` types, deprecated in Python 3.11. (Contributed by Tomas Roun in gh-104786 [https://github.com/python/cpython/issues/104786].)

unittest

- Remove the following `unittest` functions, deprecated in Python 3.11:

    - `unittest.findTestCases()`
    - `unittest.makeSuite()`
    - `unittest.getTestCaseNames()`

  Use `TestLoader` methods instead:

    - `loadTestsFromModule()`
    - `loadTestsFromTestCase()`
    - `getTestCaseNames()`

  (Contributed by Hugo van Kemenade in gh-104835 [https://github.com/python/cpython/issues/104835].)

- Remove the untested and undocumented `TestProgram.usageExit()` method, deprecated in Python 3.11. (Contributed by Hugo van Kemenade in gh-104992 [https://github.com/python/cpython/issues/104992].)

urllib

- Remove the *cafile*, *capath*, and *cadefault* parameters of the `urllib.request.urlopen()` function, deprecated in Python 3.6. Use the *context* parameter instead with an `SSLContext` instance. The `ssl.SSLContext.load_cert_chain()` function can be used to load specific certificates, or let `ssl.create_default_context()` select the operating system's trusted certificate authority (CA) certificates. (Contributed by Victor Stinner in gh-105382 [https://github.com/python/cpython/issues/105382].)

webbrowser

- Remove the untested and undocumented **`MacOSX`** class, deprecated in Python 3.11. Use the **`MacOSXOSAScript`** class (introduced in Python 3.2) instead. (Contributed by Hugo van Kemenade in gh-104804 [https://github.com/python/cpython/issues/104804].)
- Remove the deprecated **`MacOSXOSAScript._name`** attribute. Use the **`MacOSXOSAScript.name`** attribute instead. (Contributed by Nikita Sobolev in gh-105546 [https://github.com/python/cpython/issues/105546].)

New Deprecations

- User-defined functions:
  - ○ Deprecate assignment to a function's **`__code__`** attribute, where the new code object's type does not match the function's type. The different types are: plain function, generator, async generator, and coroutine. (Contributed by Irit Katriel in gh-81137 [https://github.com/python/cpython/issues/81137].)
- **`array`**:
  - ○ Deprecate the `'u'` format code (**`wchar_t`**) at runtime. This format code has been deprecated in documentation since Python 3.3, and will be removed in Python 3.16. Use the `'w'` format code (**`Py_UCS4`**) for Unicode characters instead. (Contributed by Hugo van Kemenade in gh-80480 [https://github.com/python/cpython/issues/80480].)
- **`ctypes`**:
  - ○ Deprecate the undocumented **`SetPointerType()`** function, to be removed in Python 3.15. (Contributed by Victor Stinner in gh-105733 [https://github.com/python/cpython/issues/105733].)
  - ○ Soft-deprecate the **`ARRAY()`** function in favour of

`type * length` multiplication. (Contributed by Victor Stinner in gh-105733 [https://github.com/python/cpython/issues/105733].)

- **`decimal`**:
    - ○ Deprecate the non-standard and undocumented **`Decimal`** format specifier `'N'`, which is only supported in the **`decimal`** module's C implementation. Scheduled to be removed in Python 3.18. (Contributed by Serhiy Storchaka in gh-89902 [https://github.com/python/cpython/issues/89902].)
- **`dis`**:
    - ○ Deprecate the **`HAVE_ARGUMENT`** separator. Check membership in **`hasarg`** instead. (Contributed by Irit Katriel in gh-109319 [https://github.com/python/cpython/issues/109319].)
- **`gettext`**:
    - ○ Deprecate non-integer numbers as arguments to functions and methods that consider plural forms in the **`gettext`** module, even if no translation was found. (Contributed by Serhiy Storchaka in gh-88434 [https://github.com/python/cpython/issues/88434].)
- **`glob`**:
    - ○ Deprecate the undocumented **`glob0()`** and **`glob1()`** functions. Use **`glob()`** and pass a path-like object specifying the root directory to the *root_dir* parameter instead. (Contributed by Barney Gale in gh-117337 [https://github.com/python/cpython/issues/117337].)
- **`http.server`**:
    - ○ Deprecate **`CGIHTTPRequestHandler`**, to be removed in Python 3.15. Process-based CGI HTTP servers have been out of favor for a very long time. This code was outdated, unmaintained, and rarely used. It has a high potential for both security and functionality bugs. (Contributed by Gregory P. Smith in gh-109096 [https://github.com/python/cpython/issues/109096].)
    - ○ Deprecate the **`--cgi`** flag to the **python -m http.server** command-line interface, to be removed in Python 3.15. (Contributed by Gregory P. Smith in gh-109096 [https://github.com/python/cpython/issues/109096].)
- **`mimetypes`**:

- Soft-deprecate file path arguments to `guess_type()`, use `guess_file_type()` instead. (Contributed by Serhiy Storchaka in gh-66543 [https://github.com/python/cpython/issues/66543].)

- `re`:
  - Deprecate passing the optional *maxsplit*, *count*, or *flags* arguments as positional arguments to the module-level `split()`, `sub()`, and `subn()` functions. These parameters will become keyword-only in a future version of Python. (Contributed by Serhiy Storchaka in gh-56166 [https://github.com/python/cpython/issues/56166].)
- `pathlib`:
  - Deprecate `PurePath.is_reserved()`, to be removed in Python 3.15. Use `os.path.isreserved()` to detect reserved paths on Windows. (Contributed by Barney Gale in gh-88569 [https://github.com/python/cpython/issues/88569].)
- `platform`:
  - Deprecate `java_ver()`, to be removed in Python 3.15. This function is only useful for Jython support, has a confusing API, and is largely untested. (Contributed by Nikita Sobolev in gh-116349 [https://github.com/python/cpython/issues/116349].)
- `pydoc`:
  - Deprecate the undocumented `ispackage()` function. (Contributed by Zackery Spytz in gh-64020 [https://github.com/python/cpython/issues/64020].)
- `sqlite3`:
  - Deprecate passing more than one positional argument to the `connect()` function and the `Connection` constructor. The remaining parameters will become keyword-only in Python 3.15. (Contributed by Erlend E. Aasland in gh-107948 [https://github.com/python/cpython/issues/107948].)
  - Deprecate passing name, number of arguments, and the callable as keyword arguments for `Connection.create_function()` and `Connection.create_aggregate()` These parameters will become positional-only in Python 3.15. (Contributed by Erlend E. Aasland in gh-108278 [https://

github.com/python/cpython/issues/108278].)

    - Deprecate passing the callback callable by keyword for the `set_authorizer()`, `set_progress_handler()`, and `set_trace_callback()` `Connection` methods. The callback callables will become positional-only in Python 3.15. (Contributed by Erlend E. Aasland in gh-108278 [https://github.com/python/cpython/issues/108278].)
- `sys`:
    - Deprecate the `_enablelegacywindowsfsencoding()` function, to be removed in Python 3.16. Use the `PYTHONLEGACYWINDOWSFSENCODING` environment variable instead. (Contributed by Inada Naoki in gh-73427 [https://github.com/python/cpython/issues/73427].)
- `tarfile`:
    - Deprecate the undocumented and unused `TarFile.tarfile` attribute, to be removed in Python 3.16. (Contributed in gh-115256 [https://github.com/python/cpython/issues/115256].)
- `traceback`:
    - Deprecate the `TracebackException.exc_type` attribute. Use `TracebackException.exc_type_str` instead. (Contributed by Irit Katriel in gh-112332 [https://github.com/python/cpython/issues/112332].)
- `typing`:
    - Deprecate the undocumented keyword argument syntax for creating `NamedTuple` classes (e.g. `Point = NamedTuple("Point", x=int, y=int)`), to be removed in Python 3.15. Use the class-based syntax or the functional syntax instead. (Contributed by Alex Waygood in gh-105566 [https://github.com/python/cpython/issues/105566].)
    - Deprecate omitting the *fields* parameter when creating a `NamedTuple` or `typing.TypedDict` class, and deprecate passing `None` to the *fields* parameter of both types. Python 3.15 will require a valid sequence for the *fields* parameter. To create a NamedTuple class with zero fields, use `class NT(NamedTuple): pass` or

    `NT = NamedTuple("NT", ())`. To create a TypedDict class with zero fields, use `class TD(TypedDict): pass` or `TD = TypedDict("TD", {})`. (Contributed by Alex Waygood in gh-105566 [https://github.com/python/cpython/issues/105566] and gh-105570 [https://github.com/python/cpython/issues/105570].)
  - Deprecate the **`typing.no_type_check_decorator()`** decorator function, to be removed in Python 3.15. After eight years in the **`typing`** module, it has yet to be supported by any major type checker. (Contributed by Alex Waygood in gh-106309 [https://github.com/python/cpython/issues/106309].)
  - Deprecate **`typing.AnyStr`**. In Python 3.16, it will be removed from `typing.__all__`, and a **`DeprecationWarning`** will be emitted at runtime when it is imported or accessed. It will be removed entirely in Python 3.18. Use the new type parameter syntax instead. (Contributed by Michael The in gh-107116 [https://github.com/python/cpython/issues/107116].)
- **`wave`**:
  - Deprecate the **`getmark()`**, **`setmark()`**, and **`getmarkers()`** methods of the **`Wave_read`** and **`Wave_write`** classes, to be removed in Python 3.15. (Contributed by Victor Stinner in gh-105096 [https://github.com/python/cpython/issues/105096].)

Pending removal in Python 3.14

- **`argparse`**: The *type*, *choices*, and *metavar* parameters of **`argparse.BooleanOptionalAction`** are deprecated and will be removed in 3.14. (Contributed by Nikita Sobolev in gh-92248 [https://github.com/python/cpython/issues/92248].)

- **`ast`**: The following features have been deprecated in documentation since Python 3.8, now cause a **`DeprecationWarning`** to be emitted at runtime when they are accessed or used, and will be removed in Python 3.14:

  - `ast.Num`
  - `ast.Str`
  - `ast.Bytes`
  - `ast.NameConstant`
  - `ast.Ellipsis`

  Use `ast.Constant` instead. (Contributed by Serhiy Storchaka in gh-90953 [https://github.com/python/cpython/issues/90953].)

- `asyncio`:

  - The child watcher classes `asyncio.MultiLoopChildWatcher`, `asyncio.FastChildWatcher`, `asyncio.AbstractChildWatcher` and `asyncio.SafeChildWatcher` are deprecated and will be removed in Python 3.14. (Contributed by Kumar Aditya in gh-94597 [https://github.com/python/cpython/issues/94597].)
  - `asyncio.set_child_watcher()`, `asyncio.get_child_watcher()`, `asyncio.AbstractEventLoopPolicy.set_child_watch` and `asyncio.AbstractEventLoopPolicy.get_child_watch` are deprecated and will be removed in Python 3.14. (Contributed by Kumar Aditya in gh-94597 [https://github.com/python/cpython/issues/94597].)
  - The `get_event_loop()` method of the default event loop policy now emits a `DeprecationWarning` if there is no current event loop set and it decides to create one. (Contributed by Serhiy Storchaka and Guido van Rossum in gh-100160 [https://github.com/python/cpython/issues/100160].)

- `email`: Deprecated the *isdst* parameter in `email.utils.localtime()`. (Contributed by Alan Williams in gh-72346 [https://github.com/python/cpython/issues/72346].)

- `importlib.abc` deprecated classes:

  - **`importlib.abc.ResourceReader`**
  - **`importlib.abc.Traversable`**
  - **`importlib.abc.TraversableResources`**

  Use **`importlib.resources.abc`** classes instead:

  - **`importlib.resources.abc.Traversable`**
  - **`importlib.resources.abc.TraversableResources`**

  (Contributed by Jason R. Coombs and Hugo van Kemenade in gh-93963 [https://github.com/python/cpython/issues/93963].)

- **`itertools`** had undocumented, inefficient, historically buggy, and inconsistent support for copy, deepcopy, and pickle operations. This will be removed in 3.14 for a significant reduction in code volume and maintenance burden. (Contributed by Raymond Hettinger in gh-101588 [https://github.com/python/cpython/issues/101588].)

- **`multiprocessing`**: The default start method will change to a safer one on Linux, BSDs, and other non-macOS POSIX platforms where `'fork'` is currently the default (gh-84559 [https://github.com/python/cpython/issues/84559]). Adding a runtime warning about this was deemed too disruptive as the majority of code is not expected to care. Use the **`get_context()`** or **`set_start_method()`** APIs to explicitly specify when your code *requires* `'fork'`. See Contextos y métodos de inicio.

- **`pathlib`**: **`is_relative_to()`** and **`relative_to()`**: passing additional arguments is deprecated.

- **`pkgutil`**: **`pkgutil.find_loader()`** and **`pkgutil.get_loader()`** now raise **`DeprecationWarning`**; use **`importlib.util.find_spec()`** instead. (Contributed by Nikita Sobolev in gh-97850 [https://github.com/python/cpython/issues/97850].)

- **`pty`**:

  - `master_open()`: use **`pty.openpty()`**.

    - `slave_open()`: use `pty.openpty()`.

- `sqlite3`:

    - `version` and `version_info`.
    - `execute()` and `executemany()` if named placeholders are used and *parameters* is a sequence instead of a `dict`.

- `urllib`: `urllib.parse.Quoter` is deprecated: it was not intended to be a public API. (Contributed by Gregory P. Smith in gh-88168 [https://github.com/python/cpython/issues/88168].)

Pending removal in Python 3.15

- The import system:
    - Setting `__cached__` on a module while failing to set `__spec__.cached` is deprecated. In Python 3.15, `__cached__` will cease to be set or take into consideration by the import system or standard library. (gh-97879 [https://github.com/python/cpython/issues/97879])
    - Setting `__package__` on a module while failing to set `__spec__.parent` is deprecated. In Python 3.15, `__package__` will cease to be set or take into consideration by the import system or standard library. (gh-97879 [https://github.com/python/cpython/issues/97879])
- `ctypes`:
    - The undocumented `ctypes.SetPointerType()` function has been deprecated since Python 3.13.
- `http.server`:
    - The obsolete and rarely used `CGIHTTPRequestHandler` has been deprecated since Python 3.13. No direct replacement exists. *Anything* is better than CGI to interface a web server with a request handler.
    - The `--cgi` flag to the **python -m http.server** command-line interface has been deprecated since Python 3.13.
- `importlib`:
    - `load_module()` method: use `exec_module()`

instead.

- `locale`:
    - ○ The `getdefaultlocale()` function has been deprecated since Python 3.11. Its removal was originally planned for Python 3.13 (gh-90817 [https://github.com/python/cpython/issues/90817]), but has been postponed to Python 3.15. Use `getlocale()`, `setlocale()`, and `getencoding()` instead. (Contributed by Hugo van Kemenade in gh-111187 [https://github.com/python/cpython/issues/111187].)
- `pathlib`:
    - ○ `PurePath.is_reserved()` has been deprecated since Python 3.13. Use `os.path.isreserved()` to detect reserved paths on Windows.
- `platform`:
    - ○ `java_ver()` has been deprecated since Python 3.13. This function is only useful for Jython support, has a confusing API, and is largely untested.
- `sysconfig`:
    - ○ The *check_home* argument of `sysconfig.is_python_build()` has been deprecated since Python 3.12.
- `threading`:
    - ○ `RLock()` will take no arguments in Python 3.15. Passing any arguments has been deprecated since Python 3.14, as the Python version does not permit any arguments, but the C version allows any number of positional or keyword arguments, ignoring every argument.
- `types`:
    - ○ `types.CodeType`: Accessing `co_lnotab` was deprecated in **PEP 626** [https://peps.python.org/pep-0626/] since 3.10 and was planned to be removed in 3.12, but it only got a proper `DeprecationWarning` in 3.12. May be removed in 3.15. (Contributed by Nikita Sobolev in gh-101866 [https://github.com/python/cpython/issues/101866].)
- `typing`:
    - ○ The undocumented keyword argument syntax for creating `NamedTuple` classes (for example, `Point =`

    `NamedTuple("Point", x=int, y=int)`) has been deprecated since Python 3.13. Use the class-based syntax or the functional syntax instead.
  - When using the functional syntax of **`TypedDict`**s, failing to pass a value to the *fields* parameter (`TD = TypedDict("TD")`) or passing `None` (`TD = TypedDict("TD", None)`) has been deprecated since Python 3.13. Use `class TD(TypedDict): pass` or `TD = TypedDict("TD", {})` to create a TypedDict with zero field.
  - The **`typing.no_type_check_decorator()`** decorator function has been deprecated since Python 3.13. After eight years in the **`typing`** module, it has yet to be supported by any major type checker.
- **`wave`**:
  - The **`getmark()`**, **`setmark()`**, and **`getmarkers()`** methods of the **`Wave_read`** and **`Wave_write`** classes have been deprecated since Python 3.13.
- **`zipimport`**:
  - **`load_module()`** has been deprecated since Python 3.10. Use **`exec_module()`** instead. (Contributed by Jiahao Li in gh-125746 [https://github.com/python/cpython/issues/125746].)

Pending removal in Python 3.16

- The import system:
  - Setting **`__loader__`** on a module while failing to set **`__spec__.loader`** is deprecated. In Python 3.16, **`__loader__`** will cease to be set or taken into consideration by the import system or the standard library.
- **`array`**:
  - The `'u'` format code (**`wchar_t`**) has been deprecated in documentation since Python 3.3 and at runtime since Python 3.13. Use the `'w'` format code (**`Py_UCS4`**) for Unicode characters instead.

- `asyncio`:
  - `asyncio.iscoroutinefunction()` is deprecated and will be removed in Python 3.16; use `inspect.iscoroutinefunction()` instead. (Contributed by Jiahao Li and Kumar Aditya in gh-122875 [https://github.com/python/cpython/issues/122875].)
  - `asyncio` policy system is deprecated and will be removed in Python 3.16. In particular, the following classes and functions are deprecated:
    - `asyncio.AbstractEventLoopPolicy`
    - `asyncio.DefaultEventLoopPolicy`
    - `asyncio.WindowsSelectorEventLoopPolicy`
    - `asyncio.WindowsProactorEventLoopPolicy`
    - `asyncio.get_event_loop_policy()`
    - `asyncio.set_event_loop_policy()`

    Users should use `asyncio.run()` or `asyncio.Runner` with *loop_factory* to use the desired event loop implementation.

    For example, to use `asyncio.SelectorEventLoop` on Windows:

    ```
    import asyncio

    async def main():
        ...

    asyncio.run(main(), loop_factory=asyncio.Select
    ```

    (Contributed by Kumar Aditya in gh-127949 [https://github.com/python/cpython/issues/127949].)
- `builtins`:
  - Bitwise inversion on boolean types, `~True` or `~False` has been deprecated since Python 3.12, as it produces surprising and unintuitive results (`-2` and `-1`). Use

  `not x` instead for the logical negation of a Boolean. In the rare case that you need the bitwise inversion of the underlying integer, convert to `int` explicitly (`~int(x)`).

- **`functools`**:
  - Calling the Python implementation of **`functools.reduce()`** with *function* or *sequence* as keyword arguments has been deprecated since Python 3.14.
- **`logging`**:

  Support for custom logging handlers with the *strm* argument is deprecated and scheduled for removal in Python 3.16. Define handlers with the *stream* argument instead. (Contributed by Mariusz Felisiak in gh-115032 [https://github.com/python/cpython/issues/115032].)
- **`mimetypes`**:
  - Valid extensions start with a "." or are empty for **`mimetypes.MimeTypes.add_type()`**. Undotted extensions are deprecated and will raise a **`ValueError`** in Python 3.16. (Contributed by Hugo van Kemenade in gh-75223 [https://github.com/python/cpython/issues/75223].)
- **`shutil`**:
  - The **`ExecError`** exception has been deprecated since Python 3.14. It has not been used by any function in **`shutil`** since Python 3.4, and is now an alias of **`RuntimeError`**.
- **`symtable`**:
  - The **`Class.get_methods`** method has been deprecated since Python 3.14.
- **`sys`**:

    - The `_enablelegacywindowsfsencoding()` function has been deprecated since Python 3.13. Use the `PYTHONLEGACYWINDOWSFSENCODING` environment variable instead.

- `sysconfig`:
    - The `sysconfig.expand_makefile_vars()` function has been deprecated since Python 3.14. Use the `vars` argument of `sysconfig.get_paths()` instead.

- `tarfile`:
    - The undocumented and unused `TarFile.tarfile` attribute has been deprecated since Python 3.13.

Pending removal in Python 3.17

- `collections.abc`:
    - `collections.abc.ByteString` is scheduled for removal in Python 3.17.

      Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use `Buffer` or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).

      `ByteString` was originally intended to be an abstract class that would serve as a supertype of both `bytes` and `bytearray`. However, since the ABC never had any methods, knowing that an object was an instance of `ByteString` never actually told you anything useful about the object. Other common buffer types such as `memoryview` were also never understood as subtypes of `ByteString` (either at runtime or by static type checkers).

  See **PEP 688** [https://peps.python.org/pep-0688/#current-options] for more details. (Contributed by Shantanu Jain in gh-91896 [https://github.com/python/cpython/issues/91896].)

- **`typing`**:
  - Before Python 3.14, old-style unions were implemented using the private class `typing._UnionGenericAlias`. This class is no longer needed for the implementation, but it has been retained for backward compatibility, with removal scheduled for Python 3.17. Users should use documented introspection helpers like **`typing.get_origin()`** and **`typing.get_args()`** instead of relying on private implementation details.
  - **`typing.ByteString`**, deprecated since Python 3.9, is scheduled for removal in Python 3.17.

    Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use **`Buffer`** or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).

    `ByteString` was originally intended to be an abstract class that would serve as a supertype of both **`bytes`** and **`bytearray`**. However, since the ABC never had any methods, knowing that an object was an instance of **`ByteString`** never actually told you anything useful about the object. Other common buffer types such as **`memoryview`** were also never understood as subtypes of **`ByteString`** (either at runtime or by static type checkers).

    See **PEP 688** [https://peps.python.org/pep-0688/#current-options] for more details. (Contributed by Shantanu Jain in gh-91896 [https://github.com/python/cpython/issues/91896].)

Pending removal in Python 3.18

- **`decimal`**:
    - The non-standard and undocumented **`Decimal`** format specifier `'N'`, which is only supported in the **`decimal`** module's C implementation, has been deprecated since Python 3.13. (Contributed by Serhiy Storchaka in gh-89902 [https://github.com/python/cpython/issues/89902].)

Pending removal in Python 3.19

- **`ctypes`**:
    - Implicitly switching to the MSVC-compatible struct layout by setting **`_pack_`** but not **`_layout_`** on non-Windows platforms.

Pending removal in future versions

The following APIs will be removed in the future, although there is currently no date scheduled for their removal.

- **`argparse`**:
    - Nesting argument groups and nesting mutually exclusive groups are deprecated.
    - Passing the undocumented keyword argument *prefix_chars* to **`add_argument_group()`** is now deprecated.
    - The **`argparse.FileType`** type converter is deprecated.
- **`builtins`**:
    - Generators: `throw(type, exc, tb)` and `athrow(type, exc, tb)` signature is deprecated: use `throw(exc)` and `athrow(exc)` instead, the single argument signature.
    - Currently Python accepts numeric literals immediately followed by keywords, for example `0in x`, `1or x`, `0if 1else 2`. It allows confusing and ambiguous expressions like `[0x1for x in y]` (which can be interpreted as `[0x1 for x in y]` or `[0x1f or x`

  `in y]`). A syntax warning is raised if the numeric literal is immediately followed by one of keywords **`and`**, **`else`**, **`for`**, **`if`**, **`in`**, **`is`** and **`or`**. In a future release it will be changed to a syntax error. (gh-87999 [https://github.com/python/cpython/issues/87999])
  - Support for `__index__()` and `__int__()` method returning non-int type: these methods will be required to return an instance of a strict subclass of **`int`**.
  - Support for `__float__()` method returning a strict subclass of **`float`**: these methods will be required to return an instance of **`float`**.
  - Support for `__complex__()` method returning a strict subclass of **`complex`**: these methods will be required to return an instance of **`complex`**.
  - Passing a complex number as the *real* or *imag* argument in the **`complex()`** constructor is now deprecated; it should only be passed as a single positional argument. (Contributed by Serhiy Storchaka in gh-109218 [https://github.com/python/cpython/issues/109218].)
- **`calendar`**: `calendar.January` and `calendar.February` constants are deprecated and replaced by **`calendar.JANUARY`** and **`calendar.FEBRUARY`**. (Contributed by Prince Roshan in gh-103636 [https://github.com/python/cpython/issues/103636].)
- **`codecs`**: use **`open()`** instead of **`codecs.open()`**. (gh-133038 [https://github.com/python/cpython/issues/133038])
- **`codeobject.co_lnotab`**: use the **`codeobject.co_lines()`** method instead.
- **`datetime`**:
  - **`utcnow()`**: use `datetime.datetime.now(tz=datetime.UTC)`.
  - **`utcfromtimestamp()`**: use `datetime.datetime.fromtimestamp(timestamp, tz=datetime.UTC)`.
- **`gettext`**: Plural value must be an integer.
- **`importlib`**:
  - **`cache_from_source()`** *debug_override* parameter is deprecated: use the *optimization* parameter instead.
- **`importlib.metadata`**:
  - `EntryPoints` tuple interface.

  - Implicit `None` on return values.
- **`logging`**: the `warn()` method has been deprecated since Python 3.3, use **`warning()`** instead.
- **`mailbox`**: Use of StringIO input and text mode is deprecated, use BytesIO and binary mode instead.
- **`os`**: Calling **`os.register_at_fork()`** in multi-threaded process.
- **`pydoc.ErrorDuringImport`**: A tuple value for *exc_info* parameter is deprecated, use an exception instance.
- **`re`**: More strict rules are now applied for numerical group references and group names in regular expressions. Only sequence of ASCII digits is now accepted as a numerical reference. The group name in bytes patterns and replacement strings can now only contain ASCII letters and digits and underscore. (Contributed by Serhiy Storchaka in gh-91760 [https://github.com/python/cpython/issues/91760].)
- **`sre_compile`**, **`sre_constants`** and **`sre_parse`** modules.
- **`shutil`**: **`rmtree()`**"s *onerror* parameter is deprecated in Python 3.12; use the *onexc* parameter instead.
- **`ssl`** options and protocols:
  - **`ssl.SSLContext`** without protocol argument is deprecated.
  - **`ssl.SSLContext`**: **`set_npn_protocols()`** and **`selected_npn_protocol()`** are deprecated: use ALPN instead.
  - `ssl.OP_NO_SSL*` options
  - `ssl.OP_NO_TLS*` options
  - `ssl.PROTOCOL_SSLv3`
  - `ssl.PROTOCOL_TLS`
  - `ssl.PROTOCOL_TLSv1`
  - `ssl.PROTOCOL_TLSv1_1`
  - `ssl.PROTOCOL_TLSv1_2`
  - `ssl.TLSVersion.SSLv3`
  - `ssl.TLSVersion.TLSv1`
  - `ssl.TLSVersion.TLSv1_1`
- **`threading`** methods:
  - **`threading.Condition.notifyAll()`**: use **`notify_all()`**.
  - **`threading.Event.isSet()`**: use **`is_set()`**.
  - **`threading.Thread.isDaemon()`**,

    `threading.Thread.setDaemon()`: use `threading.Thread.daemon` attribute.
  - `threading.Thread.getName()`, `threading.Thread.setName()`: use `threading.Thread.name` attribute.
  - `threading.currentThread()`: use `threading.current_thread()`.
  - `threading.activeCount()`: use `threading.active_count()`.
- `typing.Text` (gh-92332 [https://github.com/python/cpython/issues/92332]).
- The internal class `typing._UnionGenericAlias` is no longer used to implement `typing.Union`. To preserve compatibility with users using this private class, a compatibility shim will be provided until at least Python 3.17. (Contributed by Jelle Zijlstra in gh-105499 [https://github.com/python/cpython/issues/105499].)
- `unittest.IsolatedAsyncioTestCase`: it is deprecated to return a value that is not `None` from a test case.
- `urllib.parse` deprecated functions: `urlparse()` instead
  - `splitattr()`
  - `splithost()`
  - `splitnport()`
  - `splitpasswd()`
  - `splitport()`
  - `splitquery()`
  - `splittag()`
  - `splittype()`
  - `splituser()`
  - `splitvalue()`
  - `to_bytes()`
- `wsgiref`: `SimpleHandler.stdout.write()` should not do partial writes.
- `xml.etree.ElementTree`: Testing the truth value of an `Element` is deprecated. In a future release it will always return `True`. Prefer explicit `len(elem)` or `elem is not None` tests instead.
- `sys._clear_type_cache()` is deprecated: use `sys._clear_internal_caches()` instead.

CPython Bytecode Changes

- The oparg of `YIELD_VALUE` is now `1` if the yield is part of a yield-from or await, and `0` otherwise. The oparg of `RESUME` was changed to add a bit indicating if the except-depth is 1, which is needed to optimize closing of generators. (Contributed by Irit Katriel in gh-111354 [https://github.com/python/cpython/issues/111354].)

C API Changes

New Features

- Add the PyMonitoring C API for generating **PEP 669** [https://peps.python.org/pep-0669/] monitoring events:

  - `PyMonitoringState`
  - `PyMonitoring_FirePyStartEvent()`
  - `PyMonitoring_FirePyResumeEvent()`
  - `PyMonitoring_FirePyReturnEvent()`
  - `PyMonitoring_FirePyYieldEvent()`
  - `PyMonitoring_FireCallEvent()`
  - `PyMonitoring_FireLineEvent()`
  - `PyMonitoring_FireJumpEvent()`
  - `PyMonitoring_FireBranchEvent`
  - `PyMonitoring_FireCReturnEvent()`
  - `PyMonitoring_FirePyThrowEvent()`
  - `PyMonitoring_FireRaiseEvent()`
  - `PyMonitoring_FireCRaiseEvent()`
  - `PyMonitoring_FireReraiseEvent()`
  - `PyMonitoring_FireExceptionHandledEvent()`
  - `PyMonitoring_FirePyUnwindEvent()`
  - `PyMonitoring_FireStopIterationEvent()`
  - `PyMonitoring_EnterScope()`
  - `PyMonitoring_ExitScope()`

  (Contributed by Irit Katriel in gh-111997 [https://github.com/python/cpython/issues/111997]).

- Add **`PyMutex`**, a lightweight mutex that occupies a single byte, and the new **`PyMutex_Lock()`** and **`PyMutex_Unlock()`** functions. **`PyMutex_Lock()`** will release the GIL (if currently held) if the operation needs to block. (Contributed by Sam Gross in gh-108724 [https://github.com/python/cpython/issues/108724].)

- Add the PyTime C API to provide access to system clocks:
  - ○ **`PyTime_t`**.
  - ○ **`PyTime_MIN`** and **`PyTime_MAX`**.
  - ○ **`PyTime_AsSecondsDouble()`**.
  - ○ **`PyTime_Monotonic()`**.
  - ○ **`PyTime_MonotonicRaw()`**.
  - ○ **`PyTime_PerfCounter()`**.
  - ○ **`PyTime_PerfCounterRaw()`**.
  - ○ **`PyTime_Time()`**.
  - ○ **`PyTime_TimeRaw()`**.

  (Contributed by Victor Stinner and Petr Viktorin in gh-110850 [https://github.com/python/cpython/issues/110850].)

- Add the **`PyDict_ContainsString()`** function with the same behavior as **`PyDict_Contains()`**, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*. (Contributed by Victor Stinner in gh-108314 [https://github.com/python/cpython/issues/108314].)

- Add the **`PyDict_GetItemRef()`** and **`PyDict_GetItemStringRef()`** functions, which behave similarly to **`PyDict_GetItemWithError()`**, but return a strong reference instead of a borrowed reference. Moreover, these functions return `-1` on error, removing the need to check **`PyErr_Occurred()`**. (Contributed by Victor Stinner in gh-106004 [https://github.com/python/cpython/issues/106004].)

- Add the **`PyDict_SetDefaultRef()`** function, which behaves similarly to **`PyDict_SetDefault()`**, but returns a strong reference instead of a borrowed reference. This function returns `-1` on error, `0` on insertion, and `1` if the key was already present in the dictionary. (Contributed by

Sam Gross in gh-112066 [https://github.com/python/cpython/issues/112066].)

- Add the `PyDict_Pop()` and `PyDict_PopString()` functions to remove a key from a dictionary and optionally return the removed value. This is similar to `dict.pop()`, though there is no default value, and `KeyError` is not raised for missing keys. (Contributed by Stefan Behnel and Victor Stinner in gh-111262 [https://github.com/python/cpython/issues/111262].)

- Add the `PyMapping_GetOptionalItem()` and `PyMapping_GetOptionalItemString()` functions as alternatives to `PyObject_GetItem()` and `PyMapping_GetItemString()` respectively. The new functions do not raise `KeyError` if the requested key is missing from the mapping. These variants are more convenient and faster if a missing key should not be treated as a failure. (Contributed by Serhiy Storchaka in gh-106307 [https://github.com/python/cpython/issues/106307].)

- Add the `PyObject_GetOptionalAttr()` and `PyObject_GetOptionalAttrString()` functions as alternatives to `PyObject_GetAttr()` and `PyObject_GetAttrString()` respectively. The new functions do not raise `AttributeError` if the requested attribute is not found on the object. These variants are more convenient and faster if the missing attribute should not be treated as a failure. (Contributed by Serhiy Storchaka in gh-106521 [https://github.com/python/cpython/issues/106521].)

- Add the `PyErr_FormatUnraisable()` function as an extension to `PyErr_WriteUnraisable()` that allows customizing the warning message. (Contributed by Serhiy Storchaka in gh-108082 [https://github.com/python/cpython/issues/108082].)

- Add new functions that return a strong reference instead of a borrowed reference for frame locals, globals, and builtins, as part of PEP 667:

  - `PyEval_GetFrameBuiltins()` replaces `PyEval_GetBuiltins()`
  - `PyEval_GetFrameGlobals()` replaces `PyEval_GetGlobals()`
  - `PyEval_GetFrameLocals()` replaces `PyEval_GetLocals()`

  (Contributed by Mark Shannon and Tian Gao in gh-74929 [https://github.com/python/cpython/issues/74929].)

- Add the `Py_GetConstant()` and `Py_GetConstantBorrowed()` functions to get strong or borrowed references to constants. For example, `Py_GetConstant(Py_CONSTANT_ZERO)` returns a strong reference to the constant zero. (Contributed by Victor Stinner in gh-115754 [https://github.com/python/cpython/issues/115754].)

- Add the `PyImport_AddModuleRef()` function as a replacement for `PyImport_AddModule()` that returns a strong reference instead of a borrowed reference. (Contributed by Victor Stinner in gh-105922 [https://github.com/python/cpython/issues/105922].)

- Add the `Py_IsFinalizing()` function to check whether the main Python interpreter is shutting down. (Contributed by Victor Stinner in gh-108014 [https://github.com/python/cpython/issues/108014].)

- Add the `PyList_GetItemRef()` function as a replacement for `PyList_GetItem()` that returns a strong reference instead of a borrowed reference. (Contributed by Sam Gross in gh-114329 [https://github.com/python/cpython/issues/114329].)

- Add the `PyList_Extend()` and `PyList_Clear()` functions, mirroring the Python `list.extend()` and `list.clear()` methods. (Contributed by Victor Stinner in gh-111138 [https://github.com/python/cpython/issues/111138].)

- Add the `PyLong_AsInt()` function. It behaves similarly to `PyLong_AsLong()`, but stores the result in a C int instead of a C long. (Contributed by Victor Stinner in gh-108014 [https://

github.com/python/cpython/issues/108014].)

- Add the `PyLong_AsNativeBytes()`, `PyLong_FromNativeBytes()`, and `PyLong_FromUnsignedNativeBytes()` functions to simplify converting between native integer types and Python `int` objects. (Contributed by Steve Dower in gh-111140 [https://github.com/python/cpython/issues/111140].)

- Add `PyModule_Add()` function, which is similar to `PyModule_AddObjectRef()` and `PyModule_AddObject()`, but always steals a reference to the value. (Contributed by Serhiy Storchaka in gh-86493 [https://github.com/python/cpython/issues/86493].)

- Add the `PyObject_GenericHash()` function that implements the default hashing function of a Python object. (Contributed by Serhiy Storchaka in gh-113024 [https://github.com/python/cpython/issues/113024].)

- Add the `Py_HashPointer()` function to hash a raw pointer. (Contributed by Victor Stinner in gh-111545 [https://github.com/python/cpython/issues/111545].)

- Add the `PyObject_VisitManagedDict()` and `PyObject_ClearManagedDict()` functions. which must be called by the traverse and clear functions of a type using the `Py_TPFLAGS_MANAGED_DICT` flag. The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to use these functions with Python 3.11 and 3.12. (Contributed by Victor Stinner in gh-107073 [https://github.com/python/cpython/issues/107073].)

- Add the `PyRefTracer_SetTracer()` and `PyRefTracer_GetTracer()` functions, which enable tracking object creation and destruction in the same way that the `tracemalloc` module does. (Contributed by Pablo Galindo in gh-93502 [https://github.com/python/cpython/issues/93502].)

- Add the `PySys_AuditTuple()` function as an alternative to `PySys_Audit()` that takes event arguments as a Python `tuple` object. (Contributed by Victor Stinner in gh-85283 [https://github.com/python/cpython/issues/85283].)

- Add the `PyThreadState_GetUnchecked()` function as an alternative to `PyThreadState_Get()` that doesn't kill the process with a fatal error if it is `NULL`. The caller is responsible for checking if the result is `NULL`. (Contributed by Victor Stinner in gh-108867 [https://github.com/python/cpython/issues/108867].)
- Add the `PyType_GetFullyQualifiedName()` function to get the type's fully qualified name. The module name is prepended if `type.__module__` is a string and is not equal to either `'builtins'` or `'__main__'`. (Contributed by Victor Stinner in gh-111696 [https://github.com/python/cpython/issues/111696].)
- Add the `PyType_GetModuleName()` function to get the type's module name. This is equivalent to getting the `type.__module__` attribute. (Contributed by Eric Snow and Victor Stinner in gh-111696 [https://github.com/python/cpython/issues/111696].)
- Add the `PyUnicode_EqualToUTF8AndSize()` and `PyUnicode_EqualToUTF8()` functions to compare a Unicode object with a const char* UTF-8 encoded string and `1` if they are equal or `0` otherwise. These functions do not raise exceptions. (Contributed by Serhiy Storchaka in gh-110289 [https://github.com/python/cpython/issues/110289].)
- Add the `PyWeakref_GetRef()` function as an alternative to `PyWeakref_GetObject()` that returns a strong reference or `NULL` if the referent is no longer live. (Contributed by Victor Stinner in gh-105927 [https://github.com/python/cpython/issues/105927].)
- Add fixed variants of functions which silently ignore errors:
    - `PyObject_HasAttrWithError()` replaces `PyObject_HasAttr()`.

- `PyObject_HasAttrStringWithError()` replaces `PyObject_HasAttrString()`.
- `PyMapping_HasKeyWithError()` replaces `PyMapping_HasKey()`.
- `PyMapping_HasKeyStringWithError()` replaces `PyMapping_HasKeyString()`.

The new functions return `-1` for errors and the standard `1` for true and `0` for false.

(Contributed by Serhiy Storchaka in gh-108511 [https://github.com/python/cpython/issues/108511].)

Changed C APIs

- The *keywords* parameter of `PyArg_ParseTupleAndKeywords()` and `PyArg_VaParseTupleAndKeywords()` now has type char *const* in C and const char *const* in C++, instead of char**. In C++, this makes these functions compatible with arguments of type const char *const*, const char**, or char *const* without an explicit type cast. In C, the functions only support arguments of type char *const*. This can be overridden with the `PY_CXX_CONST` macro. (Contributed by Serhiy Storchaka in gh-65210 [https://github.com/python/cpython/issues/65210].)

- `PyArg_ParseTupleAndKeywords()` now supports non-ASCII keyword parameter names. (Contributed by Serhiy Storchaka in gh-110815 [https://github.com/python/cpython/issues/110815].)

- The `PyCode_GetFirstFree()` function is now unstable API and is now named `PyUnstable_Code_GetFirstFree()`. (Contributed by Bogdan Romanyuk in gh-115781 [https://github.com/python/cpython/issues/115781].)

- The `PyDict_GetItem()`, `PyDict_GetItemString()`, `PyMapping_HasKey()`, `PyMapping_HasKeyString()`, `PyObject_HasAttr()`, `PyObject_HasAttrString()`, and `PySys_GetObject()` functions, each of which clears all errors which occurred when calling them now reports these errors using `sys.unraisablehook()`. You may replace them with other functions as recommended in the documentation. (Contributed by Serhiy Storchaka in gh-106672 [https://github.com/python/cpython/issues/106672].)

- Add support for the `%T`, `%#T`, `%N` and `%#N` formats to `PyUnicode_FromFormat()`:

  - `%T`: Get the fully qualified name of an object type
  - `%#T`: As above, but use a colon as the separator
  - `%N`: Get the fully qualified name of a type
  - `%#N`: As above, but use a colon as the separator

  See **PEP 737** [https://peps.python.org/pep-0737/] for more information. (Contributed by Victor Stinner in gh-111696 [https://github.com/python/cpython/issues/111696].)

- You no longer have to define the `PY_SSIZE_T_CLEAN` macro before including `Python.h` when using `#` formats in format codes. APIs accepting the format codes always use `Py_ssize_t` for `#` formats. (Contributed by Inada Naoki in gh-104922 [https://github.com/python/cpython/issues/104922].)

- If Python is built in debug mode or `with assertions`, `PyTuple_SET_ITEM()` and `PyList_SET_ITEM()` now check the index argument with an assertion. (Contributed by Victor Stinner in gh-106168 [https://github.com/python/cpython/issues/106168].)

Limited C API Changes

- The following functions are now included in the Limited C API:

  - `PyMem_RawMalloc()`
  - `PyMem_RawCalloc()`
  - `PyMem_RawRealloc()`
  - `PyMem_RawFree()`
  - `PySys_Audit()`

  - ○ **`PySys_AuditTuple()`**
  - ○ **`PyType_GetModuleByDef()`**

  (Contributed by Victor Stinner in gh-85283 [https://github.com/python/cpython/issues/85283] and gh-116936 [https://github.com/python/cpython/issues/116936].)

- Python built with **`--with-trace-refs`** (tracing references) now supports the Limited API. (Contributed by Victor Stinner in gh-108634 [https://github.com/python/cpython/issues/108634].)

Removed C APIs

- Remove several functions, macros, variables, etc with names prefixed by `_Py` or `_PY` (which are considered private). If your project is affected by one of these removals and you believe that the removed API should remain available, please open a new issue to request a public C API and add `cc: @vstinner` to the issue to notify Victor Stinner. (Contributed by Victor Stinner in gh-106320 [https://github.com/python/cpython/issues/106320].)

- Remove old buffer protocols deprecated in Python 3.0. Use Protocolo búfer instead.

  - ○ **`PyObject_CheckReadBuffer()`**: Use **`PyObject_CheckBuffer()`** to test whether the object supports the buffer protocol. Note that **`PyObject_CheckBuffer()`** doesn't guarantee that **`PyObject_GetBuffer()`** will succeed. To test if the object is actually readable, see the next example of **`PyObject_GetBuffer()`**.

  - ○ **`PyObject_AsCharBuffer()`**, **`PyObject_AsReadBuffer()`**: Use **`PyObject_GetBuffer()`** and **`PyBuffer_Release()`** instead:

    ```
    Py_buffer view;
    if (PyObject_GetBuffer(obj, &view, PyBUF_SIMPLE
        return NULL;
    ```

```
}
// Use `view.buf` and `view.len` to read from t
// You may need to cast buf as `(const char*)vi
PyBuffer_Release(&view);
```

  - ○ **`PyObject_AsWriteBuffer()`**: Use `PyObject_GetBuffer()` and `PyBuffer_Release()` instead:

    ```
    Py_buffer view;
    if (PyObject_GetBuffer(obj, &view, PyBUF_WRITAB
        return NULL;
    }
    // Use `view.buf` and `view.len` to write to th
    PyBuffer_Release(&view);
    ```

  (Contributed by Inada Naoki in gh-85275 [https://github.com/python/cpython/issues/85275].)

- Remove various functions deprecated in Python 3.9:

  - ○ **`PyEval_CallObject()`**, **`PyEval_CallObjectWithKeywords()`**: Use `PyObject_CallNoArgs()` or `PyObject_Call()` instead.

    Advertencia

    In `PyObject_Call()`, positional arguments must be a `tuple` and must not be `NULL`, and keyword arguments must be a `dict` or `NULL`, whereas the removed functions checked argument types and accepted `NULL` positional and keyword arguments. To replace `PyEval_CallObjectWithKeywords(func, NULL, kwargs)` with **`PyObject_Call()`**, pass an empty tuple as positional arguments using `PyTuple_New(0)`.

    - **`PyEval_CallFunction()`**: Use `PyObject_CallFunction()` instead.

    - **`PyEval_CallMethod()`**: Use `PyObject_CallMethod()` instead.

    - **`PyCFunction_Call()`**: Use `PyObject_Call()` instead.

  (Contributed by Victor Stinner in gh-105107 [https://github.com/python/cpython/issues/105107].)

- Remove the following old functions to configure the Python initialization, deprecated in Python 3.11:

    - **`PySys_AddWarnOptionUnicode()`**: Use `PyConfig.warnoptions` instead.
    - **`PySys_AddWarnOption()`**: Use `PyConfig.warnoptions` instead.
    - **`PySys_AddXOption()`**: Use `PyConfig.xoptions` instead.
    - **`PySys_HasWarnOptions()`**: Use `PyConfig.xoptions` instead.
    - **`PySys_SetPath()`**: Set `PyConfig.module_search_paths` instead.
    - **`Py_SetPath()`**: Set `PyConfig.module_search_paths` instead.
    - **`Py_SetStandardStreamEncoding()`**: Set `PyConfig.stdio_encoding` instead, and set also maybe `PyConfig.legacy_windows_stdio` (on Windows).
    - **`_Py_SetProgramFullPath()`**: Set `PyConfig.executable` instead.

  Use the new `PyConfig` API of the Python Initialization Configuration instead (**PEP 587** [https://peps.python.org/pep-0587/]), added to Python 3.8. (Contributed by Victor Stinner in gh-105145 [https://github.com/python/cpython/issues/105145].)

- Remove **`PyEval_AcquireLock()`** and

`PyEval_ReleaseLock()` functions, deprecated in Python 3.2. They didn't update the current thread state. They can be replaced with:

  - `PyEval_SaveThread()` and `PyEval_RestoreThread()`;
  - low-level `PyEval_AcquireThread()` and `PyEval_RestoreThread()`;
  - or `PyGILState_Ensure()` and `PyGILState_Release()`.

  (Contributed by Victor Stinner in gh-105182 [https://github.com/python/cpython/issues/105182].)

- Remove the **`PyEval_ThreadsInitialized()`** function, deprecated in Python 3.9. Since Python 3.7, **`Py_Initialize()`** always creates the GIL: calling **`PyEval_InitThreads()`** does nothing and **`PyEval_ThreadsInitialized()`** always returns non-zero. (Contributed by Victor Stinner in gh-105182 [https://github.com/python/cpython/issues/105182].)

- Remove the **`_PyInterpreterState_Get()`** alias to `PyInterpreterState_Get()` which was kept for backward compatibility with Python 3.8. The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get **`PyInterpreterState_Get()`** on Python 3.8 and older. (Contributed by Victor Stinner in gh-106320 [https://github.com/python/cpython/issues/106320].)

- Remove the private **`_PyObject_FastCall()`** function: use **`PyObject_Vectorcall()`** which is available since Python 3.8 (**PEP 590** [https://peps.python.org/pep-0590/]). (Contributed by Victor Stinner in gh-106023 [https://github.com/python/cpython/issues/106023].)

- Remove the `cpython/pytime.h` header file, which only contained private functions. (Contributed by Victor Stinner in gh-106316 [https://github.com/python/cpython/issues/106316].)

- Remove the undocumented `PY_TIMEOUT_MAX` constant from the limited C API. (Contributed by Victor Stinner in gh-110014 [https://github.com/python/cpython/issues/110014].)

- Remove the old trashcan macros `Py_TRASHCAN_SAFE_BEGIN` and `Py_TRASHCAN_SAFE_END`. Replace both with the new macros `Py_TRASHCAN_BEGIN` and `Py_TRASHCAN_END`. (Contributed by Irit Katriel in gh-105111 [https://github.com/python/cpython/issues/105111].)

Deprecated C APIs

- Deprecate old Python initialization functions:
  - ○ **`PySys_ResetWarnOptions()`**: Clear **`sys.warnoptions`** and **`warnings.filters`** instead.
  - ○ **`Py_GetExecPrefix()`**: Get **`sys.exec_prefix`** instead.
  - ○ **`Py_GetPath()`**: Get **`sys.path`** instead.
  - ○ **`Py_GetPrefix()`**: Get **`sys.prefix`** instead.
  - ○ **`Py_GetProgramFullPath()`**: Get **`sys.executable`** instead.
  - ○ **`Py_GetProgramName()`**: Get **`sys.executable`** instead.
  - ○ **`Py_GetPythonHome()`**: Get **`PyConfig.home`** or the **`PYTHONHOME`** environment variable instead.

  (Contributed by Victor Stinner in gh-105145 [https://github.com/python/cpython/issues/105145].)

- Soft deprecate the **`PyEval_GetBuiltins()`**, **`PyEval_GetGlobals()`**, and **`PyEval_GetLocals()`** functions, which return a borrowed reference. (Soft deprecated as part of **PEP 667** [https://peps.python.org/pep-0667/].)

- Deprecate the **`PyImport_ImportModuleNoBlock()`** function, which is just an alias to **`PyImport_ImportModule()`** since Python 3.3. (Contributed by Victor Stinner in gh-105396 [https://github.com/python/cpython/issues/105396].)

- Soft deprecate the `PyModule_AddObject()` function. It should be replaced with `PyModule_Add()` or `PyModule_AddObjectRef()`. (Contributed by Serhiy Storchaka in gh-86493 [https://github.com/python/cpython/issues/86493].)

- Deprecate the old `Py_UNICODE` and `PY_UNICODE_TYPE` types and the `Py_UNICODE_WIDE` define. Use the `wchar_t` type directly instead. Since Python 3.3, `Py_UNICODE` and `PY_UNICODE_TYPE` are just aliases to `wchar_t`. (Contributed by Victor Stinner in gh-105156 [https://github.com/python/cpython/issues/105156].)

- Deprecate the `PyWeakref_GetObject()` and `PyWeakref_GET_OBJECT()` functions, which return a borrowed reference. Replace them with the new `PyWeakref_GetRef()` function, which returns a strong reference. The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get `PyWeakref_GetRef()` on Python 3.12 and older. (Contributed by Victor Stinner in gh-105927 [https://github.com/python/cpython/issues/105927].)

Pending removal in Python 3.14

- The `ma_version_tag` field in `PyDictObject` for extension modules (**PEP 699** [https://peps.python.org/pep-0699/]; gh-101193 [https://github.com/python/cpython/issues/101193]).
- Creating `immutable types` with mutable bases (gh-95388 [https://github.com/python/cpython/issues/95388]).

Pending removal in Python 3.15

- The `PyImport_ImportModuleNoBlock()`: Use `PyImport_ImportModule()` instead.

- `PyWeakref_GetObject()` and `PyWeakref_GET_OBJECT()`: Use `PyWeakref_GetRef()` instead. The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get

`PyWeakref_GetRef()` on Python 3.12 and older.

- `Py_UNICODE` type and the `Py_UNICODE_WIDE` macro: Use `wchar_t` instead.
- `PyUnicode_AsDecodedObject()`: Use `PyCodec_Decode()` instead.
- `PyUnicode_AsDecodedUnicode()`: Use `PyCodec_Decode()` instead; Note that some codecs (for example, «base64») may return a type other than `str`, such as `bytes`.
- `PyUnicode_AsEncodedObject()`: Use `PyCodec_Encode()` instead.
- `PyUnicode_AsEncodedUnicode()`: Use `PyCodec_Encode()` instead; Note that some codecs (for example, «base64») may return a type other than `bytes`, such as `str`.
- Python initialization functions, deprecated in Python 3.13:
    - `Py_GetPath()`: Use `PyConfig_Get("module_search_paths")` (`sys.path`) instead.
    - `Py_GetPrefix()`: Use `PyConfig_Get("base_prefix")` (`sys.base_prefix`) instead. Use `PyConfig_Get("prefix")` (`sys.prefix`) if virtual environments need to be handled.
    - `Py_GetExecPrefix()`: Use `PyConfig_Get("base_exec_prefix")` (`sys.base_exec_prefix`) instead. Use `PyConfig_Get("exec_prefix")` (`sys.exec_prefix`) if virtual environments need to be handled.
    - `Py_GetProgramFullPath()`: Use `PyConfig_Get("executable")` (`sys.executable`) instead.
    - `Py_GetProgramName()`: Use

    `PyConfig_Get("executable")` (`sys.executable`) instead.
  - `Py_GetPythonHome()`: Use `PyConfig_Get("home")` or the `PYTHONHOME` environment variable instead.

  The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get `PyConfig_Get()` on Python 3.13 and older.

- Functions to configure Python's initialization, deprecated in Python 3.11:

  - **`PySys_SetArgvEx()`**: Set `PyConfig.argv` instead.
  - **`PySys_SetArgv()`**: Set `PyConfig.argv` instead.
  - **`Py_SetProgramName()`**: Set `PyConfig.program_name` instead.
  - **`Py_SetPythonHome()`**: Set `PyConfig.home` instead.
  - `PySys_ResetWarnOptions()`: Clear `sys.warnoptions` and **`warnings.filters`** instead.

  The `Py_InitializeFromConfig()` API should be used with `PyConfig` instead.

- Global configuration variables:

  - `Py_DebugFlag`: Use `PyConfig.parser_debug` or `PyConfig_Get("parser_debug")` instead.
  - `Py_VerboseFlag`: Use `PyConfig.verbose` or `PyConfig_Get("verbose")` instead.
  - `Py_QuietFlag`: Use `PyConfig.quiet` or `PyConfig_Get("quiet")` instead.
  - `Py_InteractiveFlag`: Use `PyConfig.interactive` or `PyConfig_Get("interactive")` instead.
  - `Py_InspectFlag`: Use `PyConfig.inspect` or `PyConfig_Get("inspect")` instead.
  - `Py_OptimizeFlag`: Use `PyConfig.optimization_level` or `PyConfig_Get("optimization_level")` instead.
  - `Py_NoSiteFlag`: Use `PyConfig.site_import` or

`PyConfig_Get("site_import")` instead.

- `Py_BytesWarningFlag`: Use `PyConfig.bytes_warning` or `PyConfig_Get("bytes_warning")` instead.
- `Py_FrozenFlag`: Use `PyConfig.pathconfig_warnings` or `PyConfig_Get("pathconfig_warnings")` instead.
- `Py_IgnoreEnvironmentFlag`: Use `PyConfig.use_environment` or `PyConfig_Get("use_environment")` instead.
- `Py_DontWriteBytecodeFlag`: Use `PyConfig.write_bytecode` or `PyConfig_Get("write_bytecode")` instead.
- `Py_NoUserSiteDirectory`: Use `PyConfig.user_site_directory` or `PyConfig_Get("user_site_directory")` instead.
- `Py_UnbufferedStdioFlag`: Use `PyConfig.buffered_stdio` or `PyConfig_Get("buffered_stdio")` instead.
- `Py_HashRandomizationFlag`: Use `PyConfig.use_hash_seed` and `PyConfig.hash_seed` or `PyConfig_Get("hash_seed")` instead.
- `Py_IsolatedFlag`: Use `PyConfig.isolated` or `PyConfig_Get("isolated")` instead.
- `Py_LegacyWindowsFSEncodingFlag`: Use `PyPreConfig.legacy_windows_fs_encoding` or `PyConfig_Get("legacy_windows_fs_encoding")` instead.
- `Py_LegacyWindowsStdioFlag`: Use `PyConfig.legacy_windows_stdio` or `PyConfig_Get("legacy_windows_stdio")` instead.
- `Py_FileSystemDefaultEncoding`, `Py_HasFileSystemDefaultEncoding`: Use `PyConfig.filesystem_encoding` or `PyConfig_Get("filesystem_encoding")` instead.
- `Py_FileSystemDefaultEncodeErrors`: Use `PyConfig.filesystem_errors` or `PyConfig_Get("filesystem_errors")` instead.

  - **`Py_UTF8Mode`**: Use `PyPreConfig.utf8_mode` or `PyConfig_Get("utf8_mode")` instead. (see `Py_PreInitialize()`)

  The `Py_InitializeFromConfig()` API should be used with `PyConfig` to set these options. Or `PyConfig_Get()` can be used to get these options at runtime.

Pending removal in Python 3.16

- The bundled copy of `libmpdec`.

Pending removal in Python 3.18

- The following private functions are deprecated and planned for removal in Python 3.18:
  - **`_PyBytes_Join()`**: use `PyBytes_Join()`.
  - **`_PyDict_GetItemStringWithError()`**: use `PyDict_GetItemStringRef()`.
  - **`_PyDict_Pop()`**: use `PyDict_Pop()`.
  - **`_PyLong_Sign()`**: use `PyLong_GetSign()`.
  - **`_PyLong_FromDigits()`** and **`_PyLong_New()`**: use `PyLongWriter_Create()`.
  - **`_PyThreadState_UncheckedGet()`**: use `PyThreadState_GetUnchecked()`.
  - **`_PyUnicode_AsString()`**: use `PyUnicode_AsUTF8()`.
  - **`_PyUnicodeWriter_Init()`**: replace `_PyUnicodeWriter_Init(&writer)` with `writer = PyUnicodeWriter_Create(0)`.
  - **`_PyUnicodeWriter_Finish()`**: replace `_PyUnicodeWriter_Finish(&writer)` with `PyUnicodeWriter_Finish(writer)`.
  - **`_PyUnicodeWriter_Dealloc()`**: replace `_PyUnicodeWriter_Dealloc(&writer)` with `PyUnicodeWriter_Discard(writer)`.
  - **`_PyUnicodeWriter_WriteChar()`**: replace `_PyUnicodeWriter_WriteChar(&writer, ch)` with `PyUnicodeWriter_WriteChar(writer,

  `ch)`.
  - **`_PyUnicodeWriter_WriteStr()`**: replace `_PyUnicodeWriter_WriteStr(&writer, str)` with `PyUnicodeWriter_WriteStr(writer, str)`.
  - **`_PyUnicodeWriter_WriteSubstring()`**: replace `_PyUnicodeWriter_WriteSubstring(&writer, str, start, end)` with `PyUnicodeWriter_WriteSubstring(writer, str, start, end)`.
  - **`_PyUnicodeWriter_WriteASCIIString()`**: replace `_PyUnicodeWriter_WriteASCIIString(&writer, str)` with `PyUnicodeWriter_WriteASCII(writer, str)`.
  - **`_PyUnicodeWriter_WriteLatin1String()`**: replace `_PyUnicodeWriter_WriteLatin1String(&writer, str)` with `PyUnicodeWriter_WriteUTF8(writer, str)`.
  - **`_PyUnicodeWriter_Prepare()`**: (no replacement).
  - **`_PyUnicodeWriter_PrepareKind()`**: (no replacement).
  - **`_Py_HashPointer()`**: use `Py_HashPointer()`.
  - **`_Py_fopen_obj()`**: use `Py_fopen()`.

  The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get these new public functions on Python 3.13 and older. (Contributed by Victor Stinner in gh-128863 [https://github.com/python/cpython/issues/128863].)

Pending removal in future versions

The following APIs are deprecated and will be removed, although there is currently no date scheduled for their removal.

- `Py_TPFLAGS_HAVE_FINALIZE`: Unneeded since Python 3.8.
- `PyErr_Fetch()`: Use `PyErr_GetRaisedException()` instead.
- `PyErr_NormalizeException()`: Use

`PyErr_GetRaisedException()` instead.
- `PyErr_Restore()`: Use `PyErr_SetRaisedException()` instead.
- `PyModule_GetFilename()`: Use `PyModule_GetFilenameObject()` instead.
- `PyOS_AfterFork()`: Use `PyOS_AfterFork_Child()` instead.
- `PySlice_GetIndicesEx()`: Use `PySlice_Unpack()` and `PySlice_AdjustIndices()` instead.
- `PyUnicode_READY()`: Unneeded since Python 3.12
- `PyErr_Display()`: Use `PyErr_DisplayException()` instead.
- `_PyErr_ChainExceptions()`: Use `_PyErr_ChainExceptions1()` instead.
- `PyBytesObject.ob_shash` member: call `PyObject_Hash()` instead.
- Thread Local Storage (TLS) API:
  - `PyThread_create_key()`: Use `PyThread_tss_alloc()` instead.
  - `PyThread_delete_key()`: Use `PyThread_tss_free()` instead.
  - `PyThread_set_key_value()`: Use `PyThread_tss_set()` instead.
  - `PyThread_get_key_value()`: Use `PyThread_tss_get()` instead.
  - `PyThread_delete_key_value()`: Use `PyThread_tss_delete()` instead.
  - `PyThread_ReInitTLS()`: Unneeded since Python 3.7.

Build Changes

- `arm64-apple-ios` and `arm64-apple-ios-simulator` are both now **PEP 11** [https://peps.python.org/pep-0011/] tier 3 platforms. (PEP 730 written and implementation contributed by Russell Keith-Magee in gh-114099 [https://github.com/python/cpython/issues/114099].)
- `aarch64-linux-android` and `x86_64-linux-android` are both now **PEP 11** [https://peps.python.org/pep-0011/] tier 3

platforms. (PEP 738 written and implementation contributed by Malcolm Smith in gh-116622 [https://github.com/python/cpython/issues/116622].)

- `wasm32-wasi` is now a **PEP 11** [https://peps.python.org/pep-0011/] tier 2 platform. (Contributed by Brett Cannon in gh-115192 [https://github.com/python/cpython/issues/115192].)
- `wasm32-emscripten` is no longer a **PEP 11** [https://peps.python.org/pep-0011/] supported platform. (Contributed by Brett Cannon in gh-115192 [https://github.com/python/cpython/issues/115192].)
- Building CPython now requires a compiler with support for the C11 atomic library, GCC built-in atomic functions, or MSVC interlocked intrinsics.
- Autoconf 2.71 and aclocal 1.16.5 are now required to regenerate the `configure` script. (Contributed by Christian Heimes in gh-89886 [https://github.com/python/cpython/issues/89886] and by Victor Stinner in gh-112090 [https://github.com/python/cpython/issues/112090].)
- SQLite 3.15.2 or newer is required to build the **`sqlite3`** extension module. (Contributed by Erlend Aasland in gh-105875 [https://github.com/python/cpython/issues/105875].)
- CPython now bundles the mimalloc library [https://github.com/microsoft/mimalloc/] by default. It is licensed under the MIT license; see mimalloc license. The bundled mimalloc has custom changes, see gh-113141 [https://github.com/python/cpython/issues/113141] for details. (Contributed by Dino Viehland in gh-109914 [https://github.com/python/cpython/issues/109914].)
- The `configure` option **`--with-system-libmpdec`** now defaults to `yes`. The bundled copy of `libmpdec` will be removed in Python 3.16.
- Python built with `configure` **`--with-trace-refs`** (tracing references) is now ABI compatible with the Python release build and debug build. (Contributed by Victor Stinner in gh-108634 [https://github.com/python/cpython/issues/108634].)
- On POSIX systems, the pkg-config (`.pc`) filenames now include the ABI flags. For example, the free-threaded build generates `python-3.13t.pc` and the debug build generates `python-3.13d.pc`.
- The `errno`, `fcntl`, `grp`, `md5`, `pwd`, `resource`,

`termios`, `winsound`, `_ctypes_test`, `_multiprocessing.posixshmem`, `_scproxy`, `_stat`, `_statistics`, `_testconsole`, `_testimportmultiple` and `_uuid` C extensions are now built with the limited C API. (Contributed by Victor Stinner in gh-85283 [https://github.com/python/cpython/issues/85283].)

Porting to Python 3.13

This section lists previously described changes and other bugfixes that may require changes to your code.

Changes in the Python API

- PEP 667 introduces several changes to the semantics of **`locals()`** and **`f_locals`**:
  - Calling **`locals()`** in an optimized scope now produces an independent snapshot on each call, and hence no longer implicitly updates previously returned references. Obtaining the legacy CPython behavior now requires explicit calls to update the initially returned dictionary with the results of subsequent calls to **`locals()`**. Code execution functions that implicitly target **`locals()`** (such as `exec` and `eval`) must be passed an explicit namespace to access their results in an optimized scope. (Changed as part of **PEP 667** [https://peps.python.org/pep-0667/].)
  - Calling **`locals()`** from a comprehension at module or class scope (including via `exec` or `eval`) once more behaves as if the comprehension were running as an independent nested function (i.e. the local variables from the containing scope are not included). In Python 3.12, this had changed to include the local variables from the containing scope when implementing **PEP 709** [https://peps.python.org/pep-0709/]. (Changed as part of **PEP 667** [https://peps.python.org/pep-0667/].)
  - Accessing **`FrameType.f_locals`** in an optimized scope now returns a write-through proxy rather than a

snapshot that gets updated at ill-specified times. If a snapshot is desired, it must be created explicitly with `dict` or the proxy's `.copy()` method. (Changed as part of **PEP 667** [https://peps.python.org/pep-0667/].)

- **`functools.partial`** now emits a **`FutureWarning`** when used as a method. The behavior will change in future Python versions. Wrap it in **`staticmethod()`** if you want to preserve the old behavior. (Contributed by Serhiy Storchaka in gh-121027 [https://github.com/python/cpython/issues/121027].)
- An **`OSError`** is now raised by **`getpass.getuser()`** for any failure to retrieve a username, instead of **`ImportError`** on non-Unix platforms or **`KeyError`** on Unix platforms where the password database is empty.
- The value of the **`mode`** attribute of **`gzip.GzipFile`** is now a string (`'rb'` or `'wb'`) instead of an integer (`1` or `2`). The value of the **`mode`** attribute of the readable file-like object returned by **`zipfile.ZipFile.open()`** is now `'rb'` instead of `'r'`. (Contributed by Serhiy Storchaka in gh-115961 [https://github.com/python/cpython/issues/115961].)
- **`mailbox.Maildir`** now ignores files with a leading dot (`.`). (Contributed by Zackery Spytz in gh-65559 [https://github.com/python/cpython/issues/65559].)
- **`pathlib.Path.glob()`** and **`rglob()`** now return both files and directories if a pattern that ends with «`**`» is given, rather than directories only. Add a trailing slash to keep the previous behavior and only match directories.
- The **`threading`** module now expects the **`_thread`** module to have an **`_is_main_interpreter()`** function. This function takes no arguments and returns `True` if the current interpreter is the main interpreter.

  Any library or application that provides a custom **`_thread`** module must provide **`_is_main_interpreter()`**, just like the module's other «private» attributes. (gh-112826 [https://github.com/python/cpython/issues/112826].)

Changes in the C API

- `Python.h` no longer includes the `<ieeefp.h>` standard header. It was included for the **`finite()`** function which is now provided by the `<math.h>` header. It should now be included explicitly if needed. Remove also the `HAVE_IEEEFP_H` macro. (Contributed by Victor Stinner in gh-108765 [https://github.com/python/cpython/issues/108765].)

- `Python.h` no longer includes these standard header files: `<time.h>`, `<sys/select.h>` and `<sys/time.h>`. If needed, they should now be included explicitly. For example, `<time.h>` provides the **`clock()`** and **`gmtime()`** functions, `<sys/select.h>` provides the **`select()`** function, and `<sys/time.h>` provides the **`futimes()`**, **`gettimeofday()`** and **`setitimer()`** functions. (Contributed by Victor Stinner in gh-108765 [https://github.com/python/cpython/issues/108765].)

- On Windows, `Python.h` no longer includes the `<stddef.h>` standard header file. If needed, it should now be included explicitly. For example, it provides **`offsetof()`** function, and `size_t` and `ptrdiff_t` types. Including `<stddef.h>` explicitly was already needed by all other platforms, the `HAVE_STDDEF_H` macro is only defined on Windows. (Contributed by Victor Stinner in gh-108765 [https://github.com/python/cpython/issues/108765].)

- If the **`Py_LIMITED_API`** macro is defined, **`Py_BUILD_CORE`**, **`Py_BUILD_CORE_BUILTIN`** and **`Py_BUILD_CORE_MODULE`** macros are now undefined by `<Python.h>`. (Contributed by Victor Stinner in gh-85283 [https://github.com/python/cpython/issues/85283].)

- The old trashcan macros `Py_TRASHCAN_SAFE_BEGIN` and `Py_TRASHCAN_SAFE_END` were removed. They should be replaced by the new macros `Py_TRASHCAN_BEGIN` and `Py_TRASHCAN_END`.

  A `tp_dealloc` function that has the old macros, such as:

```
static void
mytype_dealloc(mytype *p)
{
    PyObject_GC_UnTrack(p);
    Py_TRASHCAN_SAFE_BEGIN(p);
    ...
    Py_TRASHCAN_SAFE_END
}
```

should migrate to the new macros as follows:

```
static void
mytype_dealloc(mytype *p)
{
    PyObject_GC_UnTrack(p);
    Py_TRASHCAN_BEGIN(p, mytype_dealloc)
    ...
    Py_TRASHCAN_END
}
```

Note that `Py_TRASHCAN_BEGIN` has a second argument which should be the deallocation function it is in. The new macros were added in Python 3.8 and the old macros were deprecated in Python 3.11. (Contributed by Irit Katriel in gh-105111 [https://github.com/python/cpython/issues/105111].)

- PEP 667 introduces several changes to frame-related functions:
    - The effects of mutating the dictionary returned from **`PyEval_GetLocals()`** in an optimized scope have changed. New dict entries added this way will now *only* be visible to subsequent **`PyEval_GetLocals()`** calls in that frame, as **`PyFrame_GetLocals()`**, **`locals()`**, and **`FrameType.f_locals`** no longer access the same underlying cached dictionary. Changes made to entries for actual variable names and names added via the write-through proxy interfaces will be overwritten on subsequent calls to **`PyEval_GetLocals()`** in that frame. The

  recommended code update depends on how the function was being used, so refer to the deprecation notice on the function for details.
  - Calling **`PyFrame_GetLocals()`** in an optimized scope now returns a write-through proxy rather than a snapshot that gets updated at ill-specified times. If a snapshot is desired, it must be created explicitly (e.g. with **`PyDict_Copy()`**), or by calling the new **`PyEval_GetFrameLocals()`** API.
  - **`PyFrame_FastToLocals()`** and **`PyFrame_FastToLocalsWithError()`** no longer have any effect. Calling these functions has been redundant since Python 3.11, when **`PyFrame_GetLocals()`** was first introduced.
  - **`PyFrame_LocalsToFast()`** no longer has any effect. Calling this function is redundant now that **`PyFrame_GetLocals()`** returns a write-through proxy for optimized scopes.

- Python 3.13 removed many private functions. Some of them can be replaced using these alternatives:

  - `_PyDict_Pop()`: **`PyDict_Pop()`** or **`PyDict_PopString()`**;
  - `_PyDict_GetItemWithError()`: **`PyDict_GetItemRef()`**;
  - `_PyErr_WriteUnraisableMsg()`: **`PyErr_FormatUnraisable()`**;
  - `_PyEval_SetTrace()`: **`PyEval_SetTrace()`** or **`PyEval_SetTraceAllThreads()`**;
  - `_PyList_Extend()`: **`PyList_Extend()`**;
  - `_PyLong_AsInt()`: **`PyLong_AsInt()`**;
  - `_PyMem_RawStrdup()`: `strdup()`;
  - `_PyMem_Strdup()`: `strdup()`;
  - `_PyObject_ClearManagedDict()`: **`PyObject_ClearManagedDict()`**;
  - `_PyObject_VisitManagedDict()`: **`PyObject_VisitManagedDict()`**;
  - `_PyThreadState_UncheckedGet()`: **`PyThreadState_GetUnchecked()`**;

  - `_PyTime_AsSecondsDouble()`: **`PyTime_AsSecondsDouble()`**;
  - `_PyTime_GetMonotonicClock()`: **`PyTime_Monotonic()`** or **`PyTime_MonotonicRaw()`**;
  - `_PyTime_GetPerfCounter()`: **`PyTime_PerfCounter()`** or **`PyTime_PerfCounterRaw()`**;
  - `_PyTime_GetSystemClock()`: **`PyTime_Time()`** or **`PyTime_TimeRaw()`**;
  - `_PyTime_MAX`: **`PyTime_MAX`**;
  - `_PyTime_MIN`: **`PyTime_MIN`**;
  - `_PyTime_t`: **`PyTime_t`**;
  - `_Py_HashPointer()`: **`Py_HashPointer()`**;
  - `_Py_IsFinalizing()`: **`Py_IsFinalizing()`**.

The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get most of these new functions on Python 3.12 and older.

Regression Test Changes

- Python built with `configure` **`--with-pydebug`** now supports a **`-X presite=package.module`** command-line option. If used, it specifies a module that should be imported early in the lifecycle of the interpreter, before `site.py` is executed. (Contributed by Łukasz Langa in gh-110769 [https://github.com/python/cpython/issues/110769].)

Novedades de Python 3.12

Editor:

Adam Turner

Este artículo explica las nuevas características de Python 3.12, en comparación con 3.11. Python 3.12 se lanzó el 2 de octubre de 2023. Para obtener detalles completos, consulte changelog.

Ver también

PEP 693 [https://peps.python.org/pep-0693/] - Calendario de lanzamiento de Python 3.12

Resumen: aspectos destacados de la versión

Python 3.12 is a stable release of the Python programming language, with a mix of changes to the language and the standard library. The library changes focus on cleaning up deprecated APIs, usability, and correctness. Of note, the `distutils` package has been removed from the standard library. Filesystem support in `os` and `pathlib` has seen a number of improvements, and several modules have better performance.

Los cambios de idioma se centran en la usabilidad, ya que a f-strings se le han eliminado muchas limitaciones y las sugerencias «¿Quiso decir…» continúan mejorando. La nueva declaración type parameter syntax y `type` mejora la ergonomía para usar generic types y type aliases con controladores de tipo estático.

Este artículo no intenta proporcionar una especificación completa de todas las características nuevas, sino que brinda una descripción general conveniente. Para obtener detalles completos, debe consultar la documentación, como Library Reference y Language Reference. Si desea comprender la implementación completa y la justificación del diseño de un cambio, consulte el PEP para conocer una característica nueva en particular; pero tenga en cuenta que los PEP generalmente no se mantienen actualizados una vez que una característica se ha implementado por completo.

Nuevas características de sintaxis:

- PEP 695, sintaxis de parámetro de tipo y sentencia `type`

Nuevas características gramaticales:

- PEP 701, f-strings en la gramática

Mejoras del intérprete:

- PEP 684, un GIL único por intérprete
- PEP 669, monitorización de bajo impacto
- Improved "Did you mean …" suggestions para excepciones `NameError`, `ImportError` y `SyntaxError`

Mejoras en el modelo de datos de Python:

- PEP 688, usando buffer protocol de Python

Mejoras significativas en la biblioteca estándar:

- La clase `pathlib.Path` ahora admite subclases
- El módulo `os` recibió varias mejoras para el soporte de Windows
- Se ha agregado un command-line interface al módulo `sqlite3`
- Los controles `isinstance()` frente a `runtime-checkable protocols` benefician de una aceleración de entre dos y 20 veces
- El paquete `asyncio` ha tenido una serie de mejoras de rendimiento, y algunos puntos de referencia muestran una aceleración del 75 %.
- Se ha agregado un command-line interface al módulo `uuid`
- Due to the changes in PEP 701, producing tokens via the

`tokenize` module is up to 64% faster.

Mejoras de seguridad:

- Reemplaza las implementaciones **`hashlib`** integradas de SHA1, SHA3, SHA2-384, SHA2-512 y MD5 con código verificado formalmente del proyecto HACL* [https://github.com/hacl-star/hacl-star/]. Estas implementaciones integradas permanecen como alternativas que solo se utilizan cuando OpenSSL no las proporciona.

Mejoras de la API C:

- PEP 697, nivel de API C inestable
- PEP 683, objetos inmortales

Mejoras en la implementación de CPython:

- PEP 709, comprensión en línea
- CPython support para el perfilador Linux `perf`
- Implementar protección contra desbordamiento de pila en plataformas compatibles

Nuevas funciones de tipado:

- PEP 692, usando **`TypedDict`** para anotar **kwargs
- PEP 698, decorador **`typing.override()`**

Depreciaciones, eliminaciones o restricciones importantes:

- **PEP 623** [https://peps.python.org/pep-0623/]: elimina `wstr` de los objetos Unicode en la API C de Python, reduciendo el tamaño de cada objeto **`str`** en al menos 8 bytes.
- **PEP 632** [https://peps.python.org/pep-0632/]: Remove the **`distutils`** package. See **the migration guide** [https://peps.python.org/pep-0632/#migration-advice] for advice replacing the APIs it provided. The third-party Setuptools [https://setuptools.pypa.io/en/latest/deprecated/distutils-legacy.html] package continues to provide **`distutils`**, if you still require it in Python 3.12 and beyond.
- gh-95299 [https://github.com/python/cpython/issues/95299]: No preinstala `setuptools` en entornos virtuales creados con

`venv`. Esto significa que `distutils`, `setuptools`, `pkg_resources` y `easy_install` ya no estarán disponibles de forma predeterminada; para acceder a ellos, ejecute `pip install setuptools` en el entorno virtual activated.

- Se han eliminado los módulos **`asynchat`**, **`asyncore`** y **`imp`**, junto con varios **`unittest.TestCase`** method aliases.

Nuevas características

PEP 695: Sintaxis de parámetro de tipo

Las clases y funciones genéricas en **PEP 484** [https://peps.python.org/pep-0484/] se declararon utilizando una sintaxis detallada que dejaba el alcance de los parámetros de tipo poco claro y requería declaraciones explícitas de variación.

PEP 695 [https://peps.python.org/pep-0695/] presenta una forma nueva, más compacta y explícita de crear generic classes y functions:

```
def max[T](args: Iterable[T]) -> T:
    ...

class list[T]:
    def __getitem__(self, index: int, /) -> T:
        ...

    def append(self, element: T) -> None:
        ...
```

Además, el PEP introduce una nueva forma de declarar type aliases utilizando la declaración **`type`**, que crea una instancia de **`TypeAliasType`**:

```
type Point = tuple[float, float]
```

Los alias de tipo también pueden ser generic:

```
type Point[T] = tuple[T, T]
```

La nueva sintaxis permite declarar los parámetros **`TypeVarTuple`** y **`ParamSpec`**, así como los parámetros **`TypeVar`** con límites o restricciones:

```
type IntFunc[**P] = Callable[P, int]  # ParamSpec
type LabeledTuple[*Ts] = tuple[str, *Ts]  # TypeVarTuple
type HashableSequence[T: Hashable] = Sequence[T]  # Type
type IntOrStrSequence[T: (int, str)] = Sequence[T]  # Ty
```

El valor de los alias de tipo y los límites y restricciones de las variables de tipo creadas mediante esta sintaxis se evalúan solo según demanda (consulte lazy evaluation). Esto significa que los alias de tipo pueden hacer referencia a otros tipos definidos más adelante en el archivo.

Los parámetros de tipo declarados a través de una lista de parámetros de tipo son visibles dentro del alcance de la declaración y de cualquier alcance anidado, pero no en el alcance externo. Por ejemplo, se pueden utilizar en las anotaciones de tipo de los métodos de una clase genérica o en el cuerpo de la clase. Sin embargo, no se pueden utilizar en el alcance del módulo una vez definida la clase. Consulte Listas de tipo parámetro para obtener una descripción detallada de la semántica de tiempo de ejecución de los parámetros de tipo.

Para respaldar esta semántica de alcance, se introduce un nuevo tipo de alcance, el annotation scope. Los ámbitos de anotación se comportan en su mayor parte como ámbitos de función, pero interactúan de manera diferente con los ámbitos de clase adjuntos. En Python 3.13, annotations también se evaluará en ámbitos de anotación.

Consulte **PEP 695** [https://peps.python.org/pep-0695/] para obtener más detalles.

(PEP escrito por Eric Traut. Implementación por Jelle Zijlstra, Eric Traut y otros en gh-103764 [https://github.com/python/cpython/issues/103764].)

PEP 701: Formalización sintáctica de cadenas f

PEP 701 [https://peps.python.org/pep-0701/] elimina algunas restricciones sobre el uso de f-strings. Los componentes de expresión dentro de cadenas f ahora pueden ser cualquier expresión Python válida, incluidas cadenas que reutilizan la misma comilla que la cadena f que las contiene, expresiones de varias líneas, comentarios, barras invertidas y secuencias de escape Unicode. Cubramos estos en detalle:

- Reutilización de comillas: en Python 3.11, reutilizar las mismas comillas que la cadena f adjunta genera un **`SyntaxError`**, lo que obliga al usuario a usar otras comillas disponibles (como usar comillas dobles o triples si la cadena f usa comillas simples). En Python 3.12, ahora puedes hacer cosas como esta:

  ```
  >>> songs = ['Take me back to Eden', 'Alkaline', 'A
  >>> f"This is the playlist: {", ".join(songs)}"
  'This is the playlist: Take me back to Eden, Alkali
  ```

  Tenga en cuenta que antes de este cambio no había un límite explícito en cómo se pueden anidar las cadenas f, pero el hecho de que las comillas de cadena no se pueden reutilizar dentro del componente de expresión de las cadenas f hacía imposible anidar cadenas f arbitrariamente. De hecho, esta es la cadena f más anidada que podría escribirse:

  ```
  >>> f"""{f'''{f'{f"{1+1}"}'}'''}"""
  '2'
  ```

  Como ahora las cadenas f pueden contener cualquier expresión Python válida dentro de los componentes de expresión, ahora es posible anidar cadenas f de forma arbitraria:

  ```
  >>> f"{f"{f"{f"{f"{f"{1+1}"}"}"}"}"}"
  '2'
  ```

- Expresiones y comentarios de varias líneas: en Python 3.11, las expresiones de cadena f deben definirse en una sola línea, incluso si la expresión dentro de la cadena f normalmente podría abarcar varias líneas (como listas literales que se definen en varias líneas), lo que las hace más difícil de leer. En Python 3.12 ahora puedes definir cadenas f que abarquen varias líneas y agregar comentarios en línea:

```
>>> f"This is the playlist: {", ".join([
...     'Take me back to Eden',  # My, my, those ey
...     'Alkaline',              # Not acid nor alk
...     'Ascensionism'           # Take to the brok
... ])}"
'This is the playlist: Take me back to Eden, Alkali
```

- Barras invertidas y caracteres Unicode: antes de Python 3.12, las expresiones de cadena f no podían contener ningún carácter `\`. Esto también afectó a Unicode escape sequences (como `\N{snowman}`), ya que contienen la parte `\N` que anteriormente no podía ser parte de los componentes de expresión de f-strings. Ahora puedes definir expresiones como esta:

```
>>> print(f"This is the playlist: {"\n".join(songs)
This is the playlist: Take me back to Eden
Alkaline
Ascensionism
>>> print(f"This is the playlist: {"\N{BLACK HEART
This is the playlist: Take me back to Eden♥Alkaline
```

Consulte **PEP 701** [https://peps.python.org/pep-0701/] para obtener más detalles.

As a positive side-effect of how this feature has been implemented (by parsing f-strings with **the PEG parser** [https://peps.python.org/pep-0617/]), now error messages for f-strings are more precise and include the exact location of the error. For example, in Python 3.11, the following f-string raises a **`SyntaxError`**:

```
>>> my_string = f"{x z y}" + f"{1 + 1}"
  File "<stdin>", line 1
    (x z y)
     ^^^
SyntaxError: f-string: invalid syntax. Perhaps you forgo
```

pero el mensaje de error no incluye la ubicación exacta del error dentro de la línea y también tiene la expresión artificialmente entre paréntesis. En Python 3.12, a medida que las cadenas f se analizan con el analizador PEG, los mensajes de error pueden ser más precisos y mostrar la línea completa:

```
>>> my_string = f"{x z y}" + f"{1 + 1}"
  File "<stdin>", line 1
    my_string = f"{x z y}" + f"{1 + 1}"
                   ^^^
SyntaxError: invalid syntax. Perhaps you forgot a comma
```

(Aportado por Pablo Galindo, Batuhan Taskaya, Lysandros Nikolaou, Cristián Maureira-Fredes y Marta Gómez en gh-102856 [https://github.com/python/cpython/issues/102856]. PEP escrito por Pablo Galindo, Batuhan Taskaya, Lysandros Nikolaou y Marta Gómez).

PEP 684: un GIL por intérprete

PEP 684 [https://peps.python.org/pep-0684/] introduce un GIL por intérprete, de modo que ahora se pueden crear subintérpretes con un GIL único por intérprete. Esto permite que los programas Python aprovechen al máximo múltiples núcleos de CPU. Actualmente, esto solo está disponible a través de C-API, aunque una API de Python es **anticipada para 3.13** [https://peps.python.org/pep-0554/].

Use the new `Py_NewInterpreterFromConfig()` function to create an interpreter with its own GIL:

```
PyInterpreterConfig config = {
    .check_multi_interp_extensions = 1,
    .gil = PyInterpreterConfig_OWN_GIL,
};
PyThreadState *tstate = NULL;
PyStatus status = Py_NewInterpreterFromConfig(&tstate, 
if (PyStatus_Exception(status)) {
    return -1;
}
/* The new interpreter is now active in the current thre
```

For further examples how to use the C-API for sub-interpreters with a per-interpreter GIL, see `Modules/_xxsubinterpretersmodule.c`.

(Aportado por Eric Snow en gh-104210 [https://github.com/python/cpython/issues/104210], etc.)

PEP 669: Monitoreo de bajo impacto para CPython

PEP 669 [https://peps.python.org/pep-0669/] define un nuevo `API` para perfiladores, depuradores y otras herramientas para monitorear eventos en CPython. Cubre una amplia gama de eventos, incluidas llamadas, devoluciones, líneas, excepciones, saltos y más. Esto significa que solo paga por lo que usa, brindando soporte para depuradores generales y herramientas de cobertura casi nulos. Consulte `sys.monitoring` para obtener más detalles.

(Aportado por Mark Shannon en gh-103082 [https://github.com/python/cpython/issues/103082].)

PEP 688: Hacer accesible el protocolo de búfer en Python

PEP 688 [https://peps.python.org/pep-0688/] presenta una forma de utilizar buffer protocol desde el código Python. Las clases que implementan el método `__buffer__()` ahora se pueden utilizar como tipos de búfer.

El nuevo `collections.abc.Buffer` ABC proporciona una forma estándar de representar objetos de búfer, por ejemplo en anotaciones de tipo. La nueva enumeración `inspect.BufferFlags` representa los indicadores que se pueden usar para personalizar la creación del búfer. (Aportado por Jelle Zijlstra en gh-102500 [https://github.com/python/cpython/issues/102500].)

PEP 709: Integración de la comprensión

Las comprensiones de diccionario, lista y conjunto ahora están integradas, en lugar de crear un nuevo objeto de función de un solo uso para cada ejecución de la comprensión. Esto acelera la ejecución de una comprensión hasta dos veces. Consulte **PEP 709** [https://peps.python.org/pep-0709/] para obtener más detalles.

Las variables de iteración de comprensión permanecen aisladas y no sobrescriben una variable del mismo nombre en el alcance externo, ni son visibles después de la comprensión. La incorporación da como resultado algunos cambios de comportamiento visibles:

- Ya no hay un marco separado para la comprensión en los rastreos, y el rastreo/elaboración de perfiles ya no muestra la comprensión como una llamada a función.
- El módulo **`symtable`** ya no producirá tablas de símbolos secundarios para cada comprensión; en cambio, los locales de comprensión se incluirán en la tabla de símbolos de la función principal.
- Llamar a **`locals()`** dentro de una comprensión ahora incluye variables externas a la comprensión y ya no incluye la variable sintética `.0` para el «argumento» de comprensión.
- Una comprensión que itera directamente sobre `locals()` (por ejemplo, `[k for k in locals()]`) puede ver «RuntimeError: el diccionario cambió de tamaño durante la iteración» cuando se ejecuta bajo seguimiento (por ejemplo, medición de cobertura de código). Este es el mismo comportamiento que ya se ha visto, por ejemplo, en `for k in locals():`. Para evitar el error, primero cree una lista de claves para iterar: `keys = list(locals()); [k for k in keys]`.

(Aportado por Carl Meyer y Vladimir Matveev en **PEP 709** [https://peps.python.org/pep-0709/].)

Mensajes de error mejorados

- Ahora se sugieren potencialmente módulos de la biblioteca estándar como parte de los mensajes de error que muestra el intérprete cuando un **`NameError`** se eleva al nivel superior. (Aportado por Pablo Galindo en gh-98254 [https://github.com/python/cpython/issues/98254].)

  ```
  >>> sys.version_info
  ```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sys' is not defined. Did you forge
```

- Mejora la sugerencia de error para excepciones **`NameError`** para instancias. Ahora, si se genera un **`NameError`** en un método y la instancia tiene un atributo que es exactamente igual al nombre en la excepción, la sugerencia incluirá `self.<NAME>` en lugar de la coincidencia más cercana en el alcance del método. (Aportado por Pablo Galindo en gh-99139 [https://github.com/python/cpython/issues/99139].)

```
>>> class A:
...    def __init__(self):
...        self.blech = 1
...
...    def foo(self):
...        somethin = blech
...
>>> A().foo()
Traceback (most recent call last):
  File "<stdin>", line 1
    somethin = blech
               ^^^^^
NameError: name 'blech' is not defined. Did you mea
```

- Mejora el mensaje de error **`SyntaxError`** cuando el usuario escribe `import x from y` en lugar de `from y import x`. (Aportado por Pablo Galindo en gh-98931 [https://github.com/python/cpython/issues/98931].)

```
>>> import a.y.z from b.y.z
Traceback (most recent call last):
  File "<stdin>", line 1
    import a.y.z from b.y.z
    ^^^^^^^^^^^^^^^^^^^^^^^
SyntaxError: Did you mean to use 'from ... import .
```

- Las excepciones **`ImportError`** generadas por sentencias `from <module> import <name>` fallidas ahora incluyen sugerencias para el valor de `<name>` según los nombres disponibles en `<module>`. (Aportado por Pablo Galindo en gh-91058 [https://github.com/python/cpython/issues/91058].)

```
>>> from collections import chainmap
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: cannot import name 'chainmap' from 'co
```

Nuevas funciones relacionadas con las sugerencias de escritura

Esta sección cubre los cambios principales que afectan a **type hints** [https://peps.python.org/pep-0484/] y al módulo **`typing`**.

PEP 692: uso de `TypedDict` para una escritura `**kwargs` más precisa

Escribir `**kwargs` en una firma de función introducida por **PEP 484** [https://peps.python.org/pep-0484/] permitía anotaciones válidas solo en los casos en que todos los `**kwargs` eran del mismo tipo.

PEP 692 [https://peps.python.org/pep-0692/] especifica una forma más precisa de escribir `**kwargs` basándose en diccionarios escritos:

```
from typing import TypedDict, Unpack

class Movie(TypedDict):
  name: str
  year: int

def foo(**kwargs: Unpack[Movie]): ...
```

Consulte **PEP 692** [https://peps.python.org/pep-0692/] para obtener más detalles.

(Aportado por Franek Magiera en gh-103629 [https://github.com/python/cpython/issues/103629].)

PEP 698: Decorador override para escritura estática

Se ha agregado un nuevo decorador **`typing.override()`** al módulo **`typing`**. Indica a los inspectores de tipo que el método está destinado a anular un método en una superclase. Esto permite a los verificadores de tipos detectar errores cuando un método destinado a anular algo en una clase base en realidad no lo hace.

Ejemplo:

```
from typing import override

class Base:
  def get_color(self) -> str:
    return "blue"

class GoodChild(Base):
  @override  # ok: overrides Base.get_color
  def get_color(self) -> str:
    return "yellow"

class BadChild(Base):
  @override  # type checker error: does not override Bas
  def get_colour(self) -> str:
    return "red"
```

Consulte **PEP 698** [https://peps.python.org/pep-0698/] para obtener más detalles.

(Aportado por Steven Troxler en gh-101561 [https://github.com/python/cpython/issues/101561].)

Otros cambios del lenguaje

- El analizador ahora genera **`SyntaxError`** al analizar el código fuente que contiene bytes nulos. (Aportado por Pablo Galindo en gh-96670 [https://github.com/python/cpython/issues/96670].)
- Un par de caracteres de barra invertida que no es una

secuencia de escape válida ahora genera un **`SyntaxWarning`**, en lugar de **`DeprecationWarning`**. Por ejemplo, `re.compile("\d+\.\d+")` ahora emite un **`SyntaxWarning`** (`"\d"` es una secuencia de escape no válida; utilice cadenas sin formato para la expresión regular: `re.compile(r"\d+\.\d+")`). En una versión futura de Python, eventualmente se generará **`SyntaxError`**, en lugar de **`SyntaxWarning`**. (Aportado por Victor Stinner en gh-98401 [https://github.com/python/cpython/issues/98401].)

- Los escapes octales con un valor mayor que `0o377` (por ejemplo, `"\477"`), obsoletos en Python 3.11, ahora producen un **`SyntaxWarning`**, en lugar de **`DeprecationWarning`**. En una futura versión de Python, eventualmente serán un **`SyntaxError`**. (Aportado por Victor Stinner en gh-98401 [https://github.com/python/cpython/issues/98401].)
- Las variables utilizadas en la parte de destino de las comprensiones que no están almacenadas ahora se pueden usar en expresiones de asignación (`:=`). Por ejemplo, en `[(b := 1) for a, b.prop in some_iter]`, ahora se permite la asignación a `b`. Tenga en cuenta que la asignación de variables almacenadas en la parte de destino de las comprensiones (como `a`) todavía no está permitida, según **PEP 572** [https://peps.python.org/pep-0572/]. (Aportado por Nikita Sobolev en gh-100581 [https://github.com/python/cpython/issues/100581].)
- Las excepciones generadas en el método `__set_name__` de una clase o tipo ya no están incluidas en un **`RuntimeError`**. La información de contexto se agrega a la excepción como una nota **PEP 678** [https://peps.python.org/pep-0678/]. (Aportado por Irit Katriel en gh-77757 [https://github.com/python/cpython/issues/77757].)
- Cuando una construcción `try-except*` maneja todo el **`ExceptionGroup`** y genera otra excepción, esa excepción ya no está incluida en un **`ExceptionGroup`**. También cambió en la versión 3.11.4. (Aportado por Irit Katriel en gh-103590 [https://github.com/python/cpython/issues/103590].)
- El recolector de basura ahora se ejecuta solo en el mecanismo de interrupción de evaluación del ciclo de evaluación del código de bytes de Python en lugar de en las asignaciones de

objetos. El GC también se puede ejecutar cuando se llama a `PyErr_CheckSignals()`, por lo que las extensiones C que necesitan ejecutarse durante un período prolongado sin ejecutar ningún código Python también tienen la posibilidad de ejecutar el GC periódicamente. (Aportado por Pablo Galindo en gh-97922 [https://github.com/python/cpython/issues/97922].)

- Todos los invocables integrados y de extensión que esperan parámetros booleanos ahora aceptan argumentos de cualquier tipo en lugar de solo `bool` y `int`. (Contribución de Serhiy Storchaka en gh-60203 [https://github.com/python/cpython/issues/60203].)
- `memoryview` ahora admite el tipo medio flotante (el código de formato «e»). (Contribución de Donghee Na y Antoine Pitrou en gh-90751 [https://github.com/python/cpython/issues/90751].)
- Los objetos `slice` ahora son hasheables, lo que permite usarlos como claves de dictado y elementos de configuración. (Contribución de Will Bradshaw, Furkan Onder y Raymond Hettinger en gh-101264 [https://github.com/python/cpython/issues/101264].)
- `sum()` ahora utiliza la suma de Neumaier para mejorar la precisión y la conmutatividad al sumar flotantes o enteros y flotantes mixtos. (Aportado por Raymond Hettinger en gh-100425 [https://github.com/python/cpython/issues/100425].)
- `ast.parse()` ahora genera `SyntaxError` en lugar de `ValueError` al analizar el código fuente que contiene bytes nulos. (Aportado por Pablo Galindo en gh-96670 [https://github.com/python/cpython/issues/96670].)
- Los métodos de extracción en `tarfile` y `shutil.unpack_archive()` tienen un nuevo argumento, *filter*, que permite limitar funciones tar que puedan resultar sorprendentes o peligrosas, como la creación de archivos fuera del directorio de destino. Consulte tarfile extraction filters para obtener más detalles. En Python 3.14, el valor predeterminado cambiará a `'data'`. (Aportado por Petr Viktorin en **PEP 706** [https://peps.python.org/pep-0706/].)
- Las instancias `types.MappingProxyType` ahora son hasheables si el mapeo subyacente es hasheable. (Contribución de Serhiy Storchaka en gh-87995 [https://

github.com/python/cpython/issues/87995].)

- Agregado support for the perf profiler a través de la nueva variable de entorno `PYTHONPERFSUPPORT` y la opción de línea de comandos `-X perf`, así como las nuevas funciones `sys.activate_stack_trampoline()`, `sys.deactivate_stack_trampoline()` y `sys.is_stack_trampoline_active()`. (Diseño de Pablo Galindo. Contribución de Pablo Galindo y Christian Heimes con contribuciones de Gregory P. Smith [Google] y Mark Shannon en gh-96123 [https://github.com/python/cpython/issues/96123].)

Nuevos módulos

- Ninguno.

Módulos mejorados

array

- La clase `array.array` ahora admite subíndices, lo que la convierte en generic type. (Aportado por Jelle Zijlstra en gh-98658 [https://github.com/python/cpython/issues/98658].)

asincio

- Se ha mejorado significativamente el rendimiento de escritura en sockets en `asyncio`. `asyncio` ahora evita copias innecesarias al escribir en sockets y usa `sendmsg()` si la plataforma lo admite. (Aportado por Kumar Aditya en gh-91166 [https://github.com/python/cpython/issues/91166].)
- Agregado las funciones `asyncio.eager_task_factory()` y `asyncio.create_eager_task_factory()` para permitir la opción de un bucle de eventos para la ejecución de tareas entusiastas, lo que hace que algunos casos de uso sean de 2 a 5 veces más rápidos. (Contribución de Jacob Bower e Itamar Oren en gh-102853 [https://github.com/python/cpython/issues/102853], gh-104140 [https://github.com/python/cpython/

issues/104140] y gh-104138 [https://github.com/python/cpython/issues/104138])

- On Linux, `asyncio` uses **`asyncio.PidfdChildWatcher`** by default if `os.pidfd_open()` is available and functional instead of **`asyncio.ThreadedChildWatcher`**. (Contributed by Kumar Aditya in gh-98024 [https://github.com/python/cpython/issues/98024].)
- The event loop now uses the best available child watcher for each platform (**`asyncio.PidfdChildWatcher`** if supported and **`asyncio.ThreadedChildWatcher`** otherwise), so manually configuring a child watcher is not recommended. (Contributed by Kumar Aditya in gh-94597 [https://github.com/python/cpython/issues/94597].)
- Agregado el parámetro *loop_factory* a `asyncio.run()` para permitir especificar una fábrica de bucle de eventos personalizada. (Aportado por Kumar Aditya en gh-99388 [https://github.com/python/cpython/issues/99388].)
- Agregado la implementación C de `asyncio.current_task()` para una aceleración de 4x-6x. (Aportado por Itamar Oren y Pranav Thulasiram Bhat en gh-100344 [https://github.com/python/cpython/issues/100344].)
- `asyncio.iscoroutine()` ahora devuelve `False` para generadores, ya que `asyncio` no admite rutinas heredadas basadas en generadores. (Aportado por Kumar Aditya en gh-102748 [https://github.com/python/cpython/issues/102748].)
- `asyncio.wait()` y `asyncio.as_completed()` ahora aceptan generadores que generan tareas. (Aportado por Kumar Aditya en gh-78530 [https://github.com/python/cpython/issues/78530].)

calendar

- Agregado las enumeraciones `calendar.Month` y `calendar.Day` que definen los meses del año y los días de la semana. (Aportado por el Príncipe Roshan en gh-103636 [https://github.com/python/cpython/issues/103636].)

csv

- Add `csv.QUOTE_NOTNULL` and `csv.QUOTE_STRINGS` flags to provide finer grained control of `None` and empty strings by `reader` and `writer` objects.

dis

- Los códigos de operación de pseudoinstrucción (que son utilizados por el compilador pero que no aparecen en el código de bytes ejecutable) ahora están expuestos en el módulo `dis`. `HAVE_ARGUMENT` sigue siendo relevante para códigos de operación reales, pero no es útil para pseudoinstrucciones. Utilice la nueva colección `dis.hasarg` en su lugar. (Aportado por Irit Katriel en gh-94216 [https://github.com/python/cpython/issues/94216].)
- Agregado la colección `dis.hasexc` para indicar instrucciones que establecen un controlador de excepciones. (Aportado por Irit Katriel en gh-94216 [https://github.com/python/cpython/issues/94216].)

fractions

- Los objetos de tipo `fractions.Fraction` ahora admiten formato de estilo flotante. (Aportado por Mark Dickinson en gh-100161 [https://github.com/python/cpython/issues/100161].)

importlib.resources

- `importlib.resources.as_file()` ahora admite directorios de recursos. (Aportado por Jason R. Coombs en gh-97930 [https://github.com/python/cpython/issues/97930].)
- Rename first parameter of `importlib.resources.files()` to *anchor*. (Contributed by Jason R. Coombs in gh-100598 [https://github.com/python/cpython/issues/100598].)

inspect

- Add `inspect.markcoroutinefunction()` to mark sync functions that return a coroutine for use with `inspect.iscoroutinefunction()`. (Contributed by

Carlton Gibson in gh-99247 [https://github.com/python/cpython/issues/99247].)

- Agregado `inspect.getasyncgenstate()` y `inspect.getasyncgenlocals()` para determinar el estado actual de los generadores asíncronos. (Aportado por Thomas Krennwallner en gh-79940 [https://github.com/python/cpython/issues/79940].)
- The performance of `inspect.getattr_static()` has been considerably improved. Most calls to the function should be at least 2x faster than they were in Python 3.11. (Contributed by Alex Waygood in gh-103193 [https://github.com/python/cpython/issues/103193].)

itertools

- Add `itertools.batched()` for collecting into even-sized tuples where the last batch may be shorter than the rest. (Contributed by Raymond Hettinger in gh-98363 [https://github.com/python/cpython/issues/98363].)

math

- Agregado `math.sumprod()` para calcular una suma de productos. (Aportado por Raymond Hettinger en gh-100485 [https://github.com/python/cpython/issues/100485].)
- Extend `math.nextafter()` to include a *steps* argument for moving up or down multiple steps at a time. (Contributed by Matthias Goergens, Mark Dickinson, and Raymond Hettinger in gh-94906 [https://github.com/python/cpython/issues/94906].)

os

- Agregado `os.PIDFD_NONBLOCK` para abrir un descriptor de archivo para un proceso con `os.pidfd_open()` en modo sin bloqueo. (Aportado por Kumar Aditya en gh-93312 [https://github.com/python/cpython/issues/93312].)
- `os.DirEntry` ahora incluye un método `os.DirEntry.is_junction()` para verificar si la entrada es un cruce. (Aportado por Charles Machalow en gh-99547 [https://github.com/python/cpython/issues/99547].)

- Agregado funciones `os.listdrives()`, `os.listvolumes()` y `os.listmounts()` en Windows para enumerar unidades, volúmenes y puntos de montaje. (Aportado por Steve Dower en gh-102519 [https://github.com/python/cpython/issues/102519].)
- `os.stat()` y `os.lstat()` ahora son más precisos en Windows. El campo `st_birthtime` ahora se completará con la hora de creación del archivo, y `st_ctime` está obsoleto pero aún contiene la hora de creación (pero en el futuro devolverá el último cambio de metadatos, para mantener la coherencia con otras plataformas). `st_dev` puede tener hasta 64 bits y `st_ino` hasta 128 bits dependiendo de su sistema de archivos, y `st_rdev` siempre está configurado en cero en lugar de valores incorrectos. Ambas funciones pueden ser significativamente más rápidas en las versiones más recientes de Windows. (Aportado por Steve Dower en gh-99726 [https://github.com/python/cpython/issues/99726].)

os.path

- Agregado `os.path.isjunction()` para verificar si una ruta determinada es un cruce. (Aportado por Charles Machalow en gh-99547 [https://github.com/python/cpython/issues/99547].)
- Agregado `os.path.splitroot()` para dividir una ruta en una tríada `(drive, root, tail)`. (Aportado por Barney Gale en gh-101000 [https://github.com/python/cpython/issues/101000].)

pathlib

- Agregado soporte para subclasificar `pathlib.PurePath` y `pathlib.Path`, además de sus variantes específicas de Posix y Windows. Las subclases pueden anular el método `pathlib.PurePath.with_segments()` para pasar información entre instancias de ruta.
- Agregado `pathlib.Path.walk()` para recorrer los árboles de directorios y generar todos los nombres de archivos o directorios dentro de ellos, similar a `os.walk()`. (Aportado

por Stanislav Zmiev en gh-90385 [https://github.com/python/cpython/issues/90385].)

- Agregado el parámetro opcional *walk_up* a **`pathlib.PurePath.relative_to()`** para permitir la inserción de entradas `..` en el resultado; este comportamiento es más consistente con **`os.path.relpath()`**. (Aportado por Domenico Ragusa en gh-84538 [https://github.com/python/cpython/issues/84538].)
- Agregado **`pathlib.Path.is_junction()`** como proxy a **`os.path.isjunction()`**. (Aportado por Charles Machalow en gh-99547 [https://github.com/python/cpython/issues/99547].)
- Agregado el parámetro opcional *case_sensitive* a **`pathlib.Path.glob()`**, **`pathlib.Path.rglob()`** y **`pathlib.PurePath.match()`** para hacer coincidir la distinción entre mayúsculas y minúsculas de la ruta, lo que permite un control más preciso sobre el proceso de coincidencia.

platform

- Add support for detecting Windows 11 and Windows Server releases past 2012. Previously, lookups on Windows Server platforms newer than Windows Server 2012 and on Windows 11 would return `Windows-10`. (Contributed by Steve Dower in gh-89545 [https://github.com/python/cpython/issues/89545].)

pbd

- Agregado variables convenientes para mantener valores temporalmente para la sesión de depuración y proporcionar acceso rápido a valores como el marco actual o el valor de retorno. (Aportado por Tian Gao en gh-103693 [https://github.com/python/cpython/issues/103693].)

random

- Agregado **`random.binomialvariate()`**. (Aportado por Raymond Hettinger en gh-81620 [https://github.com/python/cpython/issues/81620].)
- Agregado un valor predeterminado de `lambd=1.0` a

`random.expovariate()`. (Aportado por Raymond Hettinger en gh-100234 [https://github.com/python/cpython/issues/100234].)

shutil

- **`shutil.make_archive()`** ahora pasa el argumento *root_dir* a archivadores personalizados que lo admiten. En este caso, ya no cambia temporalmente el directorio de trabajo actual del proceso a *root_dir* para realizar el archivado. (Aportado por Serhiy Storchaka en gh-74696 [https://github.com/python/cpython/issues/74696].)
- **`shutil.rmtree()`** now accepts a new argument *onexc* which is an error handler like *onerror* but which expects an exception instance rather than a *(typ, val, tb)* triplet. *onerror* is deprecated. (Contributed by Irit Katriel in gh-102828 [https://github.com/python/cpython/issues/102828].)
- **`shutil.which()`** ahora consulta la variable de entorno *PATHEXT* para encontrar coincidencias dentro de *PATH* en Windows incluso cuando el *cmd* dado incluye un componente de directorio. (Aportado por Charles Machalow en gh-103179 [https://github.com/python/cpython/issues/103179].)

  `shutil.which()` llamará a `NeedCurrentDirectoryForExePathW` cuando solicite ejecutables en Windows para determinar si el directorio de trabajo actual debe anteponerse a la ruta de búsqueda. (Aportado por Charles Machalow en gh-103179 [https://github.com/python/cpython/issues/103179].)

  `shutil.which()` devolverá una ruta que coincida con *cmd* con un componente de `PATHEXT` antes de una coincidencia directa en otra parte de la ruta de búsqueda en Windows. (Aportado por Charles Machalow en gh-103179 [https://github.com/python/cpython/issues/103179].)

sqlite3

- Agregado un command-line interface. (Aportado por Erlend E. Aasland en gh-77617 [https://github.com/python/cpython/issues/77617].)
- Agregado el atributo `sqlite3.Connection.autocommit` a `sqlite3.Connection` y el parámetro *autocommit* a `sqlite3.connect()` para controlar transaction handling compatible con **PEP 249** [https://peps.python.org/pep-0249/]. (Aportado por Erlend E. Aasland en gh-83638 [https://github.com/python/cpython/issues/83638].)
- Agregado el parámetro de solo palabra clave *entrypoint* a `sqlite3.Connection.load_extension()`, para anular el punto de entrada de la extensión SQLite. (Aportado por Erlend E. Aasland en gh-103015 [https://github.com/python/cpython/issues/103015].)
- Agregado `sqlite3.Connection.getconfig()` y `sqlite3.Connection.setconfig()` a `sqlite3.Connection` para realizar cambios de configuración en una conexión de base de datos. (Aportado por Erlend E. Aasland en gh-103489 [https://github.com/python/cpython/issues/103489].)

statistics

- Se extendió `statistics.correlation()` para incluirlo como método `ranked` para calcular la correlación de Spearman de datos clasificados. (Aportado por Raymond Hettinger en gh-95861 [https://github.com/python/cpython/issues/95861].)

sys

- Agregado el espacio de nombres `sys.monitoring` para exponer la nueva API de supervisión PEP 669. (Aportado por Mark Shannon en gh-103082 [https://github.com/python/cpython/issues/103082].)
- Agregado `sys.activate_stack_trampoline()` y `sys.deactivate_stack_trampoline()` para activar y desactivar los trampolines del perfilador de pila, y `sys.is_stack_trampoline_active()` para consultar si los trampolines del perfilador de pila están activos.

(Contribuido por Pablo Galindo y Christian Heimes con contribuciones de Gregory P. Smith [Google] y Mark Shannon en gh-96123 [https://github.com/python/cpython/issues/96123].)

- Agregado `sys.last_exc` que contiene la última excepción no controlada que se generó (para casos de uso de depuración post-mortem). Deje obsoletos los tres campos que tienen la misma información en su formato heredado: `sys.last_type`, `sys.last_value` y `sys.last_traceback`. (Aportado por Irit Katriel en gh-102778 [https://github.com/python/cpython/issues/102778].)
- `sys._current_exceptions()` ahora devuelve una asignación de thread-id a una instancia de excepción, en lugar de a una tupla `(typ, exc, tb)`. (Aportado por Irit Katriel en gh-103176 [https://github.com/python/cpython/issues/103176].)
- `sys.setrecursionlimit()` y `sys.getrecursionlimit()`. El límite de recursividad ahora se aplica sólo al código Python. Las funciones integradas no utilizan el límite de recursividad, pero están protegidas por un mecanismo diferente que evita que la recursividad provoque un fallo de la máquina virtual.

tempfile

- La función `tempfile.NamedTemporaryFile` tiene un nuevo parámetro opcional *delete_on_close* (Aportado por Evgeny Zorin en gh-58451 [https://github.com/python/cpython/issues/58451].)
- `tempfile.mkdtemp()` ahora siempre devuelve una ruta absoluta, incluso si el argumento proporcionado al parámetro *dir* es una ruta relativa.

threading

- Agregado `threading.settrace_all_threads()` y `threading.setprofile_all_threads()` que permiten configurar funciones de seguimiento y creación de perfiles en todos los hilos en ejecución además del que realiza la llamada. (Aportado por Pablo Galindo en gh-93503 [https://github.com/python/cpython/issues/93503].)

tkinter

- `tkinter.Canvas.coords()` ahora aplana sus argumentos. Ahora acepta no sólo coordenadas como argumentos separados (`x1, y1, x2, y2, ...`) y una secuencia de coordenadas (`[x1, y1, x2, y2, ...]`), sino también coordenadas agrupadas en pares (`(x1, y1), (x2, y2), ...` y `[(x1, y1), (x2, y2), ...]`), como los métodos `create_*()`. (Aportado por Serhiy Storchaka en gh-94473 [https://github.com/python/cpython/issues/94473].)

tokenize

- El módulo **`tokenize`** incluye los cambios introducidos en **PEP 701** [https://peps.python.org/pep-0701/]. (Aportado por Marta Gómez Macías y Pablo Galindo en gh-102856 [https://github.com/python/cpython/issues/102856].) Consulte Portar a Python 3.12 para obtener más información sobre los cambios en el módulo **`tokenize`**.

types

- Agregado **`types.get_original_bases()`** para permitir una mayor introspección de Tipos genéricos definidos por el usuario cuando esté subclasificado. (Contribución de James Hilton-Balfe y Alex Waygood en gh-101827 [https://github.com/python/cpython/issues/101827].)

typing

- Las comprobaciones de **`isinstance()`** con **`runtime-checkable protocols`** ahora usan **`inspect.getattr_static()`** en lugar de **`hasattr()`** para buscar si existen atributos. Esto significa que los descriptores y los métodos **`__getattr__()`** ya no se evalúan inesperadamente durante las comprobaciones de `isinstance()` con protocolos verificables en tiempo de ejecución. Sin embargo, también puede significar que algunos objetos que solían considerarse instancias de un protocolo verificable en tiempo de ejecución ya no se consideran instancias de ese protocolo en Python 3.12+, y viceversa. Es poco probable que la mayoría de los usuarios se vean afectados por este cambio. (Contribuido por Alex Waygood en gh-102433 [https://github.com/python/cpython/issues/102433].)

- Los miembros de un protocolo verificable en tiempo de ejecución ahora se consideran «congelados» en tiempo de ejecución tan pronto como se crea la clase. La aplicación de parches de atributos en un protocolo verificable en tiempo de ejecución seguirá funcionando, pero no tendrá ningún impacto en las comprobaciones **`isinstance()`** que comparan objetos con el protocolo. Por ejemplo:

  ```
  >>> from typing import Protocol, runtime_checkable
  >>> @runtime_checkable
  ... class HasX(Protocol):
  ...     x = 1
  ...
  >>> class Foo: ...
  ...
  >>> f = Foo()
  >>> isinstance(f, HasX)
  False
  >>> f.x = 1
  >>> isinstance(f, HasX)
  True
  >>> HasX.y = 2
  >>> isinstance(f, HasX)  # unchanged, even though H
  True
  ```

  Este cambio se realizó para acelerar las comprobaciones de `isinstance()` en protocolos verificables en tiempo de ejecución.

- The performance profile of **`isinstance()`** checks against **`runtime-checkable protocols`** has changed significantly. Most `isinstance()` checks against protocols with only a few members should be at least 2x faster than in 3.11, and some may be 20x faster or more. However, `isinstance()` checks against protocols with many members may be slower than in Python 3.11. (Contributed by Alex Waygood in gh-74690 [https://github.com/python/cpython/issues/74690] and gh-103193 [https://github.com/python/cpython/issues/103193].)

- Todas las clases **`typing.TypedDict`** y **`typing.NamedTuple`** ahora tienen el atributo `__orig_bases__`. (Aportado por Adrián García Badaracco en gh-103699 [https://github.com/python/cpython/issues/103699].)
- Agregado el parámetro `frozen_default` a **`typing.dataclass_transform()`**. (Aportado por Erik De Bonte en gh-99957 [https://github.com/python/cpython/issues/99957].)

unicodedata

- La base de datos Unicode se ha actualizado a la versión 15.0.0. (Aportado por Benjamin Peterson en gh-96734 [https://github.com/python/cpython/issues/96734]).

unittest

Agregado una opción de línea de comando `--durations`, que muestra los N casos de prueba más lentos:

```
python3 -m unittest --durations=3 lib.tests.test_threadi
.....
Slowest test durations
--------------------------------------------------------
1.210s     test_timeout (Lib.test.test_threading.Barrier
1.003s     test_default_timeout (Lib.test.test_threadin
0.518s     test_timeout (Lib.test.test_threading.EventTe

(0.000 durations hidden.  Use -v to show these durations
--------------------------------------------------------
Ran 158 tests in 9.869s

OK (skipped=3)
```

(Aportado por Giampaolo Rodola en gh-48330 [https://github.com/python/cpython/issues/48330])

uuid

- Agregado un command-line interface. (Aportado por Adam Chhina en gh-88597 [https://github.com/python/cpython/issues/88597].)

Optimizaciones

- Elimine los miembros `wstr` y `wstr_length` de los objetos Unicode. Reduce el tamaño del objeto en 8 o 16 bytes en una plataforma de 64 bits. (**PEP 623** [https://peps.python.org/pep-0623/]) (Aportado por Inada Naoki en gh-92536 [https://github.com/python/cpython/issues/92536].)
- Agregado soporte experimental para usar el optimizador binario BOLT en el proceso de compilación, lo que mejora el rendimiento entre un 1% y un 5%. (Contribuido por Kevin Modzelewski en gh-90536 [https://github.com/python/cpython/issues/90536] y sintonizado por Donghee Na en gh-101525 [https://github.com/python/cpython/issues/101525])
- Acelere la sustitución de expresiones regulares (funciones `re.sub()` y `re.subn()` y los métodos `re.Pattern` correspondientes) para cadenas de reemplazo que contienen referencias de grupo entre 2 y 3 veces. (Aportado por Serhiy Storchaka en gh-91524 [https://github.com/python/cpython/issues/91524].)
- Acelere la creación de `asyncio.Task` aplazando el costoso formato de cadenas. (Aportado por Itamar Oren en gh-103793 [https://github.com/python/cpython/issues/103793].)
- Las funciones `tokenize.tokenize()` y `tokenize.generate_tokens()` son hasta un 64 % más rápidas como efecto secundario de los cambios necesarios para cubrir **PEP 701** [https://peps.python.org/pep-0701/] en el módulo `tokenize`. (Aportado por Marta Gómez Macías y Pablo Galindo en gh-102856 [https://github.com/python/cpython/issues/102856].)
- Acelere las llamadas al método `super()` y las cargas de

atributos mediante la nueva instrucción `LOAD_SUPER_ATTR`. (Contribución de Carl Meyer y Vladimir Matveev en gh-103497 [https://github.com/python/cpython/issues/103497].)

Cambios en el código de bytes de CPython

- Elimine la instrucción `LOAD_METHOD`. Se ha fusionado con `LOAD_ATTR`. `LOAD_ATTR` ahora se comportará como la antigua instrucción `LOAD_METHOD` si el bit bajo de su oparg está configurado. (Contribuido por Ken Jin en gh-93429 [https://github.com/python/cpython/issues/93429].)
- Elimine las instrucciones `JUMP_IF_FALSE_OR_POP` y `JUMP_IF_TRUE_OR_POP`. (Aportado por Irit Katriel en gh-102859 [https://github.com/python/cpython/issues/102859].)
- Elimine la instrucción `PRECALL`. (Aportado por Mark Shannon en gh-92925 [https://github.com/python/cpython/issues/92925].)
- Agregado las instrucciones `BINARY_SLICE` y `STORE_SLICE`. (Aportado por Mark Shannon en gh-94163 [https://github.com/python/cpython/issues/94163].)
- Agregado las instrucciones `CALL_INTRINSIC_1`. (Aportado por Mark Shannon en gh-99005 [https://github.com/python/cpython/issues/99005].)
- Agregado la instrucción `CALL_INTRINSIC_2`. (Aportado por Irit Katriel en gh-101799 [https://github.com/python/cpython/issues/101799].)
- Agregado la instrucción `CLEANUP_THROW`. (Aportado por Brandt Bucher en gh-90997 [https://github.com/python/cpython/issues/90997].)
- Agregado la instrucción `END_SEND`. (Aportado por Mark Shannon en gh-103082 [https://github.com/python/cpython/issues/103082].)
- Agregada la instrucción `LOAD_FAST_AND_CLEAR` como parte de la implementación de **PEP 709** [https://peps.python.org/pep-0709/]. (Aportado por Carl Meyer en gh-101441 [https://github.com/python/cpython/issues/101441].)
- Agregado la instrucción `LOAD_FAST_CHECK`. (Aportado por Dennis Sweeney en gh-93143 [https://github.com/python/cpython/issues/93143].)

- Agregado los códigos de operación **`LOAD_FROM_DICT_OR_DEREF`**, **`LOAD_FROM_DICT_OR_GLOBALS`** y **`LOAD_LOCALS`** como parte de la implementación de **PEP 695** [https://peps.python.org/pep-0695/]. Elimine el código de operación **`LOAD_CLASSDEREF`**, que se puede reemplazar con **`LOAD_LOCALS`** más **`LOAD_FROM_DICT_OR_DEREF`**. (Aportado por Jelle Zijlstra en gh-103764 [https://github.com/python/cpython/issues/103764].)
- Agregado la instrucción **`LOAD_SUPER_ATTR`**. (Contribución de Carl Meyer y Vladimir Matveev en gh-103497 [https://github.com/python/cpython/issues/103497].)
- Add the `RETURN_CONST` instruction. (Contributed by Wenyang Wang in gh-101632 [https://github.com/python/cpython/issues/101632].)

Demostraciones y herramientas

- Elimine el directorio `Tools/demo/` que contenía scripts de demostración antiguos. Se puede encontrar una copia en el old-demos project [https://github.com/gvanrossum/old-demos]. (Aportado por Victor Stinner en gh-97681 [https://github.com/python/cpython/issues/97681].)
- Elimine los scripts de ejemplo obsoletos del directorio `Tools/scripts/`. Se puede encontrar una copia en el old-demos project [https://github.com/gvanrossum/old-demos]. (Aportado por Victor Stinner en gh-97669 [https://github.com/python/cpython/issues/97669].)

Obsoleto

- **`argparse`**: Los parámetros *type*, *choices* y *metavar* de **`argparse.BooleanOptionalAction`** están obsoletos y se eliminarán en 3.14. (Contribución de Nikita Sobolev en gh-92248 [https://github.com/python/cpython/issues/92248].)

- **`ast`**: las siguientes características de **`ast`** han quedado obsoletas en la documentación desde Python 3.8, ahora provocan que se emita un **`DeprecationWarning`** en tiempo de ejecución cuando se accede a ellas o se usan, y se eliminarán en Python 3.14:

  - `ast.Num`
  - `ast.Str`
  - `ast.Bytes`
  - `ast.NameConstant`
  - `ast.Ellipsis`

  Utilice `ast.Constant` en su lugar. (Aportado por Serhiy Storchaka en gh-90953 [https://github.com/python/cpython/issues/90953].)

- `asyncio`:

  - The child watcher classes `asyncio.MultiLoopChildWatcher`, `asyncio.FastChildWatcher`, `asyncio.AbstractChildWatcher` and `asyncio.SafeChildWatcher` are deprecated and will be removed in Python 3.14. (Contributed by Kumar Aditya in gh-94597 [https://github.com/python/cpython/issues/94597].)
  - `asyncio.set_child_watcher()`, `asyncio.get_child_watcher()`, `asyncio.AbstractEventLoopPolicy.set_child_watch` and `asyncio.AbstractEventLoopPolicy.get_child_watch` are deprecated and will be removed in Python 3.14. (Contributed by Kumar Aditya in gh-94597 [https://github.com/python/cpython/issues/94597].)
  - El método `get_event_loop()` de la política de bucle de eventos predeterminada ahora emite un `DeprecationWarning` si no hay ningún bucle de eventos establecido y decide crear uno. (Contribución de Serhiy Storchaka y Guido van Rossum en gh-100160 [https://github.com/python/cpython/issues/100160].)

- `calendar`: las constantes `calendar.January` y `calendar.February` están obsoletas y reemplazadas por `calendar.JANUARY` y `calendar.FEBRUARY`. (Aportado por el Príncipe Roshan en gh-103636 [https://github.com/python/cpython/issues/103636].)

- `collections.abc`: Deprecated `collections.abc.ByteString`.

  Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use `Buffer` or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).

  `ByteString` was originally intended to be an abstract class that would serve as a supertype of both `bytes` and `bytearray`. However, since the ABC never had any methods, knowing that an object was an instance of **`ByteString`** never actually told you anything useful about the object. Other common buffer types such as `memoryview` were also never understood as subtypes of **`ByteString`** (either at runtime or by static type checkers).

  See PEP 688 [https://peps.python.org/pep-0688/#current-options] for more details. (Contributed by Shantanu Jain in gh-91896 [https://github.com/python/cpython/issues/91896].)

- `datetime`: `utcnow()` y `utcfromtimestamp()` de `datetime.datetime` están en desuso y se eliminarán en una versión futura. En su lugar, utilice objetos que tengan en cuenta la zona horaria para representar las fechas y horas en UTC: respectivamente, llame a `now()` y `fromtimestamp()` con el parámetro *tz* establecido en `datetime.UTC`. (Aportado por Paul Ganssle en gh-103857 [https://github.com/python/cpython/issues/103857].)

- `email`: obsoleto el parámetro *isdst* en `email.utils.localtime()`. (Aportado por Alan Williams en gh-72346 [https://github.com/python/cpython/issues/72346].)

- `importlib.abc`: Obsoletas las siguientes clases, cuya eliminación está programada en Python 3.14:

- ○ **`importlib.abc.ResourceReader`**
- ○ **`importlib.abc.Traversable`**
- ○ **`importlib.abc.TraversableResources`**

Utilice clases **`importlib.resources.abc`** en su lugar:

- ○ **`importlib.resources.abc.Traversable`**
- ○ **`importlib.resources.abc.TraversableResources`**

(Contribución de Jason R. Coombs y Hugo van Kemenade en gh-93963 [https://github.com/python/cpython/issues/93963].)

- **`itertools`**: obsoleta la compatibilidad con operaciones de copia, copia profunda y pickle, que no está documentada, es ineficiente, históricamente tiene errores e inconsistente. Esto se eliminará en 3.14 para lograr una reducción significativa en el volumen de código y la carga de mantenimiento. (Aportado por Raymond Hettinger en gh-101588 [https://github.com/python/cpython/issues/101588].)

- **`multiprocessing`**: en Python 3.14, el método de inicio predeterminado de **`multiprocessing`** cambiará a uno más seguro en Linux, BSD y otras plataformas POSIX que no sean macOS donde `'fork'` es actualmente el predeterminado (gh-84559 [https://github.com/python/cpython/issues/84559]). Agregar una advertencia de tiempo de ejecución sobre esto se consideró demasiado perjudicial ya que se espera que a la mayoría del código no le importe. Utilice las API **`get_context()`** o **`set_start_method()`** para especificar explícitamente cuándo su código *requires* `'fork'`. Ver contexts and start methods.

- **`pkgutil`**: **`pkgutil.find_loader()`** and **`pkgutil.get_loader()`** are deprecated and will be removed in Python 3.14; use **`importlib.util.find_spec()`** instead. (Contributed by Nikita Sobolev in gh-97850 [https://github.com/python/cpython/issues/97850].)

- **`pty`**: el módulo tiene dos funciones `master_open()` y `slave_open()` no documentadas que han quedado obsoletas desde Python 2 pero que solo obtuvieron un **`DeprecationWarning`** adecuado en 3.12. Elimínelos en 3.14. (Aportado por Soumendra Ganguly y Gregory P. Smith en gh-85984 [https://github.com/python/cpython/issues/85984].)

- **`os`**:
  - Los campos `st_ctime` devueltos por **`os.stat()`** y **`os.lstat()`** en Windows están en desuso. En una versión futura, contendrán la hora del último cambio de metadatos, de forma coherente con otras plataformas. Por ahora, todavía contienen la hora de creación, que también está disponible en el nuevo campo `st_birthtime`. (Aportado por Steve Dower en gh-99726 [https://github.com/python/cpython/issues/99726].)
  - On POSIX platforms, **`os.fork()`** can now raise a **`DeprecationWarning`** when it can detect being called from a multithreaded process. There has always been a fundamental incompatibility with the POSIX platform when doing so. Even if such code *appeared* to work. We added the warning to raise awareness as issues encountered by code doing this are becoming more frequent. See the **`os.fork()`** documentation for more details along with this discussion on fork being incompatible with threads [https://discuss.python.org/t/concerns-regarding-deprecation-of-fork-with-alive-threads/33555] for *why* we're now surfacing this longstanding platform compatibility problem to developers.

  Cuando aparece esta advertencia debido al uso de **`multiprocessing`** o **`concurrent.futures`**, la solución es utilizar un método de inicio de **`multiprocessing`** diferente, como `"spawn"` o `"forkserver"`.

- **`shutil`**: The *onerror* argument of **`shutil.rmtree()`** is deprecated; use *onexc* instead. (Contributed by Irit Katriel in gh-102828 [https://github.com/python/cpython/issues/102828].)
- **`sqlite3`**:
  - default adapters and converters ahora están en desuso.

    En su lugar, utilice el Ejemplos para adaptadores y convertidores y adáptelo a sus necesidades. (Aportado por Erlend E. Aasland en gh-90016 [https://github.com/python/cpython/issues/90016].)
  - En `execute()`, `DeprecationWarning` ahora se emite cuando named placeholders se utiliza junto con los parámetros proporcionados como sequence en lugar de como `dict`. A partir de Python 3.14, el uso de marcadores de posición con nombre con parámetros proporcionados como una secuencia generará un `ProgrammingError`. (Aportado por Erlend E. Aasland en gh-101698 [https://github.com/python/cpython/issues/101698].)

- `sys`: los campos `sys.last_type`, `sys.last_value` y `sys.last_traceback` están en desuso. Utilice `sys.last_exc` en su lugar. (Aportado por Irit Katriel en gh-102778 [https://github.com/python/cpython/issues/102778].)

- `tarfile`: la extracción de archivos tar sin especificar *filter* está en desuso hasta Python 3.14, cuando el filtro `'data'` se convertirá en el predeterminado. Consulte Extraction filters para obtener más detalles.

- `typing`:

  - `typing.Hashable` and `typing.Sized`, aliases for `collections.abc.Hashable` and `collections.abc.Sized` respectively, are deprecated. (gh-94309 [https://github.com/python/cpython/issues/94309].)
  - `typing.ByteString`, en desuso desde Python 3.9, ahora provoca que se emita un `DeprecationWarning` cuando se utiliza. (Aportado por Alex Waygood en gh-91896 [https://github.com/python/cpython/issues/91896].)

- `xml.etree.ElementTree`: el módulo ahora emite `DeprecationWarning` al probar el valor de verdad de un `xml.etree.ElementTree.Element`. Antes, la implementación de Python emitía `FutureWarning` y la implementación de C no emitía nada. (Aportado por Jacob Walls en gh-83122 [https://github.com/python/cpython/issues/83122].)

- Las firmas de 3 argumentos (tipo, valor, rastreo) de **`coroutine throw()`**, **`generator throw()`** y **`async generator throw()`** están en desuso y es posible que se eliminen en una versión futura de Python. Utilice en su lugar las versiones de un solo argumento de estas funciones. (Aportado por Ofey Chan en gh-89874 [https://github.com/python/cpython/issues/89874].)

- **`DeprecationWarning`** is now raised when **`__package__`** on a module differs from **`__spec__.parent`** (previously it was **`ImportWarning`**). (Contributed by Brett Cannon in gh-65961 [https://github.com/python/cpython/issues/65961].)

- Setting **`__package__`** or **`__cached__`** on a module is deprecated, and will cease to be set or taken into consideration by the import system in Python 3.14. (Contributed by Brett Cannon in gh-65961 [https://github.com/python/cpython/issues/65961].)

- The bitwise inversion operator (~) on bool is deprecated. It will throw an error in Python 3.16. Use `not` for logical negation of bools instead. In the rare case that you really need the bitwise inversion of the underlying `int`, convert to int explicitly: `~int(x)`. (Contributed by Tim Hoffmann in gh-103487 [https://github.com/python/cpython/issues/103487].)

- Accessing **`co_lnotab`** on code objects was deprecated in Python 3.10 via **PEP 626** [https://peps.python.org/pep-0626/], but it only got a proper **`DeprecationWarning`** in 3.12. May be removed in 3.15. (Contributed by Nikita Sobolev in gh-101866 [https://github.com/python/cpython/issues/101866].)

Pending removal in Python 3.13

Módulos (ver **PEP 594** [https://peps.python.org/pep-0594/]):

- **`aifc`**

- **audioop**
- **cgi**
- **cgitb**
- **chunk**
- **crypt**
- **imghdr**
- **mailcap**
- **msilib**
- **nis**
- **nntplib**
- **ossaudiodev**
- **pipes**
- **sndhdr**
- **spwd**
- **sunau**
- **telnetlib**
- **uu**
- **xdrlib**

Otros módulos:

- **lib2to3** y el programa **2to3** (gh-84540 [https://github.com/python/cpython/issues/84540])

APIs:

- **configparser.LegacyInterpolation** (gh-90765 [https://github.com/python/cpython/issues/90765])
- `locale.resetlocale()` (gh-90817 [https://github.com/python/cpython/issues/90817])
- **turtle.RawTurtle.settiltangle()** (gh-50096 [https://github.com/python/cpython/issues/50096])
- **unittest.findTestCases()** (gh-50096 [https://github.com/python/cpython/issues/50096])
- **unittest.getTestCaseNames()** (gh-50096 [https://github.com/python/cpython/issues/50096])
- **unittest.makeSuite()** (gh-50096 [https://github.com/python/cpython/issues/50096])
- **unittest.TestProgram.usageExit()** (gh-67048 [https://github.com/python/cpython/issues/67048])
- **webbrowser.MacOSX** (gh-86421 [https://github.com/python/

cpython/issues/86421])

- Encadenamiento de descriptores `classmethod` (gh-89519 [https://github.com/python/cpython/issues/89519])

Pending removal in Python 3.14

- `argparse`: Los parámetros *type*, *choices* y *metavar* de `argparse.BooleanOptionalAction` están obsoletos y se eliminarán en 3.14. (Contribución de Nikita Sobolev en gh-92248 [https://github.com/python/cpython/issues/92248].)

- `ast`: The following features have been deprecated in documentation since Python 3.8, now cause a `DeprecationWarning` to be emitted at runtime when they are accessed or used, and will be removed in Python 3.14:

  - `ast.Num`
  - `ast.Str`
  - `ast.Bytes`
  - `ast.NameConstant`
  - `ast.Ellipsis`

  Utilice `ast.Constant` en su lugar. (Aportado por Serhiy Storchaka en gh-90953 [https://github.com/python/cpython/issues/90953].)

- `asyncio`:

  - The child watcher classes `asyncio.MultiLoopChildWatcher`, `asyncio.FastChildWatcher`, `asyncio.AbstractChildWatcher` and `asyncio.SafeChildWatcher` are deprecated and will be removed in Python 3.14. (Contributed by Kumar Aditya in gh-94597 [https://github.com/python/cpython/issues/94597].)
  - `asyncio.set_child_watcher()`, `asyncio.get_child_watcher()`, `asyncio.AbstractEventLoopPolicy.set_child_watc` and `asyncio.AbstractEventLoopPolicy.get_child_watc`

  are deprecated and will be removed in Python 3.14. (Contributed by Kumar Aditya in gh-94597 [https://github.com/python/cpython/issues/94597].)
  - El método `get_event_loop()` de la política de bucle de eventos predeterminada ahora emite un `DeprecationWarning` si no hay ningún bucle de eventos establecido y decide crear uno. (Contribución de Serhiy Storchaka y Guido van Rossum en gh-100160 [https://github.com/python/cpython/issues/100160].)

- `email`: Deprecated the *isdst* parameter in `email.utils.localtime()`. (Contributed by Alan Williams in gh-72346 [https://github.com/python/cpython/issues/72346].)

- `importlib.abc` deprecated classes:

  - **`importlib.abc.ResourceReader`**
  - **`importlib.abc.Traversable`**
  - **`importlib.abc.TraversableResources`**

  Utilice clases `importlib.resources.abc` en su lugar:

  - `importlib.resources.abc.Traversable`
  - `importlib.resources.abc.TraversableResources`

  (Contribución de Jason R. Coombs y Hugo van Kemenade en gh-93963 [https://github.com/python/cpython/issues/93963].)

- `itertools` had undocumented, inefficient, historically buggy, and inconsistent support for copy, deepcopy, and pickle operations. This will be removed in 3.14 for a significant reduction in code volume and maintenance burden. (Contributed by Raymond Hettinger in gh-101588 [https://github.com/python/cpython/issues/101588].)

- `multiprocessing`: The default start method will change to a safer one on Linux, BSDs, and other non-macOS POSIX platforms where `'fork'` is currently the default (gh-84559 [https://github.com/python/cpython/issues/84559]). Adding a runtime warning about this was deemed too disruptive as the majority of code is not expected to care. Use the `get_context()` or `set_start_method()` APIs to explicitly specify when your code *requires* `'fork'`. See Contextos y métodos de inicio.

- `pathlib`: `is_relative_to()` and `relative_to()`: passing additional arguments is deprecated.
- `pkgutil`: **`pkgutil.find_loader()`** and **`pkgutil.get_loader()`** now raise `DeprecationWarning`; use `importlib.util.find_spec()` instead. (Contributed by Nikita Sobolev in gh-97850 [https://github.com/python/cpython/issues/97850].)
- `pty`:
  - `master_open()`: use `pty.openpty()`.
  - `slave_open()`: use `pty.openpty()`.
- `sqlite3`:
  - **`version`** and **`version_info`**.
  - `execute()` and `executemany()` if named placeholders are used and *parameters* is a sequence instead of a `dict`.
- `urllib`: **`urllib.parse.Quoter`** is deprecated: it was not intended to be a public API. (Contributed by Gregory P. Smith in gh-88168 [https://github.com/python/cpython/issues/88168].)

Pending removal in Python 3.15

- The import system:
  - Setting `__cached__` on a module while failing to set `__spec__.cached` is deprecated. In Python 3.15, **`__cached__`** will cease to be set or take into consideration by the import system or standard library. (gh-97879 [https://github.com/python/cpython/issues/97879])
  - Setting `__package__` on a module while failing to set `__spec__.parent` is deprecated. In Python 3.15, **`__package__`** will cease to be set or take into

consideration by the import system or standard library. (gh-97879 [https://github.com/python/cpython/issues/97879])

- `ctypes`:
  - The undocumented `ctypes.SetPointerType()` function has been deprecated since Python 3.13.
- `http.server`:
  - The obsolete and rarely used `CGIHTTPRequestHandler` has been deprecated since Python 3.13. No direct replacement exists. *Anything* is better than CGI to interface a web server with a request handler.
  - The `--cgi` flag to the **python -m http.server** command-line interface has been deprecated since Python 3.13.
- `importlib`:
  - `load_module()` method: use `exec_module()` instead.
- `locale`:
  - The `getdefaultlocale()` function has been deprecated since Python 3.11. Its removal was originally planned for Python 3.13 (gh-90817 [https://github.com/python/cpython/issues/90817]), but has been postponed to Python 3.15. Use `getlocale()`, `setlocale()`, and `getencoding()` instead. (Contributed by Hugo van Kemenade in gh-111187 [https://github.com/python/cpython/issues/111187].)
- `pathlib`:
  - `PurePath.is_reserved()` has been deprecated since Python 3.13. Use `os.path.isreserved()` to detect reserved paths on Windows.
- `platform`:
  - `java_ver()` has been deprecated since Python 3.13. This function is only useful for Jython support, has a confusing API, and is largely untested.
- `sysconfig`:
  - The *check_home* argument of `sysconfig.is_python_build()` has been deprecated since Python 3.12.
- `threading`:
  - `RLock()` will take no arguments in Python 3.15.

    Passing any arguments has been deprecated since Python 3.14, as the Python version does not permit any arguments, but the C version allows any number of positional or keyword arguments, ignoring every argument.

- **`types`**:
  - **`types.CodeType`**: Accessing **`co_lnotab`** was deprecated in **PEP 626** [https://peps.python.org/pep-0626/] since 3.10 and was planned to be removed in 3.12, but it only got a proper **`DeprecationWarning`** in 3.12. May be removed in 3.15. (Contributed by Nikita Sobolev in gh-101866 [https://github.com/python/cpython/issues/101866].)
- **`typing`**:
  - The undocumented keyword argument syntax for creating **`NamedTuple`** classes (for example, `Point = NamedTuple("Point", x=int, y=int)`) has been deprecated since Python 3.13. Use the class-based syntax or the functional syntax instead.
  - When using the functional syntax of **`TypedDict`**s, failing to pass a value to the *fields* parameter (`TD = TypedDict("TD")`) or passing `None` (`TD = TypedDict("TD", None)`) has been deprecated since Python 3.13. Use `class TD(TypedDict): pass` or `TD = TypedDict("TD", {})` to create a TypedDict with zero field.
  - The **`typing.no_type_check_decorator()`** decorator function has been deprecated since Python 3.13. After eight years in the **`typing`** module, it has yet to be supported by any major type checker.
- **`wave`**:
  - The **`getmark()`**, **`setmark()`**, and **`getmarkers()`** methods of the **`Wave_read`** and **`Wave_write`** classes have been deprecated since Python 3.13.
- **`zipimport`**:
  - **`load_module()`** has been deprecated since Python 3.10. Use **`exec_module()`** instead. (Contributed by Jiahao Li in gh-125746 [https://github.com/python/cpython/issues/125746].)

Pending removal in Python 3.16

- The import system:
  - Setting `__loader__` on a module while failing to set `__spec__.loader` is deprecated. In Python 3.16, `__loader__` will cease to be set or taken into consideration by the import system or the standard library.
- `array`:
  - The `'u'` format code (`wchar_t`) has been deprecated in documentation since Python 3.3 and at runtime since Python 3.13. Use the `'w'` format code (`Py_UCS4`) for Unicode characters instead.
- `asyncio`:
  - `asyncio.iscoroutinefunction()` is deprecated and will be removed in Python 3.16; use `inspect.iscoroutinefunction()` instead. (Contributed by Jiahao Li and Kumar Aditya in gh-122875 [https://github.com/python/cpython/issues/122875].)
  - `asyncio` policy system is deprecated and will be removed in Python 3.16. In particular, the following classes and functions are deprecated:
    - `asyncio.AbstractEventLoopPolicy`
    - `asyncio.DefaultEventLoopPolicy`
    - `asyncio.WindowsSelectorEventLoopPolicy`
    - `asyncio.WindowsProactorEventLoopPolicy`
    - `asyncio.get_event_loop_policy()`
    - `asyncio.set_event_loop_policy()`

    Users should use `asyncio.run()` or `asyncio.Runner` with *loop_factory* to use the desired event loop implementation.

    For example, to use `asyncio.SelectorEventLoop` on Windows:

    ```
    import asyncio

    async def main():
        ...

    asyncio.run(main(), loop_factory=asyncio.Select
    ```

    (Contributed by Kumar Aditya in gh-127949 [https://github.com/python/cpython/issues/127949].)

- **`builtins`**:
    - Bitwise inversion on boolean types, `~True` or `~False` has been deprecated since Python 3.12, as it produces surprising and unintuitive results (`-2` and `-1`). Use `not x` instead for the logical negation of a Boolean. In the rare case that you need the bitwise inversion of the underlying integer, convert to `int` explicitly (`~int(x)`).
- **`functools`**:
    - Calling the Python implementation of **`functools.reduce()`** with *function* or *sequence* as keyword arguments has been deprecated since Python 3.14.
- **`logging`**:

  Support for custom logging handlers with the *strm* argument is deprecated and scheduled for removal in Python 3.16. Define handlers with the *stream* argument instead. (Contributed by Mariusz Felisiak in gh-115032 [https://github.com/python/cpython/issues/115032].)
- **`mimetypes`**:
    - Valid extensions start with a “.” or are empty for **`mimetypes.MimeTypes.add_type()`**. Undotted

    extensions are deprecated and will raise a `ValueError` in Python 3.16. (Contributed by Hugo van Kemenade in gh-75223 [https://github.com/python/cpython/issues/75223].)

- `shutil`:
    - The `ExecError` exception has been deprecated since Python 3.14. It has not been used by any function in `shutil` since Python 3.4, and is now an alias of `RuntimeError`.
- `symtable`:
    - The `Class.get_methods` method has been deprecated since Python 3.14.
- `sys`:
    - The `_enablelegacywindowsfsencoding()` function has been deprecated since Python 3.13. Use the `PYTHONLEGACYWINDOWSFSENCODING` environment variable instead.
- `sysconfig`:
    - The `sysconfig.expand_makefile_vars()` function has been deprecated since Python 3.14. Use the `vars` argument of `sysconfig.get_paths()` instead.
- `tarfile`:
    - The undocumented and unused `TarFile.tarfile` attribute has been deprecated since Python 3.13.

Pending removal in Python 3.17

- `collections.abc`:
    - `collections.abc.ByteString` is scheduled for removal in Python 3.17.

    Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use **`Buffer`** or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).

    `ByteString` was originally intended to be an abstract class that would serve as a supertype of both **`bytes`** and **`bytearray`**. However, since the ABC never had any methods, knowing that an object was an instance of **`ByteString`** never actually told you anything useful about the object. Other common buffer types such as **`memoryview`** were also never understood as subtypes of **`ByteString`** (either at runtime or by static type checkers).

    See **PEP 688** [https://peps.python.org/pep-0688/#current-options] for more details. (Contributed by Shantanu Jain in gh-91896 [https://github.com/python/cpython/issues/91896].)

- **`typing`**:
    - Before Python 3.14, old-style unions were implemented using the private class `typing._UnionGenericAlias`. This class is no longer needed for the implementation, but it has been retained for backward compatibility, with removal scheduled for Python 3.17. Users should use documented introspection helpers like **`typing.get_origin()`** and **`typing.get_args()`** instead of relying on private implementation details.
    - **`typing.ByteString`**, deprecated since Python 3.9, is scheduled for removal in Python 3.17.

      Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use **`Buffer`** or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).

`ByteString` was originally intended to be an abstract class that would serve as a supertype of both **`bytes`** and **`bytearray`**. However, since the ABC never had any methods, knowing that an object was an instance of **`ByteString`** never actually told you anything useful about the object. Other common buffer types such as **`memoryview`** were also never understood as subtypes of **`ByteString`** (either at runtime or by static type checkers).

See **PEP 688** [https://peps.python.org/pep-0688/#current-options] for more details. (Contributed by Shantanu Jain in gh-91896 [https://github.com/python/cpython/issues/91896].)

Pending removal in future versions

The following APIs will be removed in the future, although there is currently no date scheduled for their removal.

- **`argparse`**:
    - Nesting argument groups and nesting mutually exclusive groups are deprecated.
    - Passing the undocumented keyword argument *prefix_chars* to **`add_argument_group()`** is now deprecated.
    - The **`argparse.FileType`** type converter is deprecated.
- **`builtins`**:
    - Generators: `throw(type, exc, tb)` and `athrow(type, exc, tb)` signature is deprecated: use `throw(exc)` and `athrow(exc)` instead, the single argument signature.
    - Currently Python accepts numeric literals immediately followed by keywords, for example `0in x`, `1or x`, `0if 1else 2`. It allows confusing and ambiguous expressions like `[0x1for x in y]` (which can be

    interpreted as `[0x1 for x in y]` or `[0x1f or x in y]`). A syntax warning is raised if the numeric literal is immediately followed by one of keywords **`and`**, **`else`**, **`for`**, **`if`**, **`in`**, **`is`** and **`or`**. In a future release it will be changed to a syntax error. (gh-87999 [https://github.com/python/cpython/issues/87999])
  - Support for `__index__()` and `__int__()` method returning non-int type: these methods will be required to return an instance of a strict subclass of **`int`**.
  - Support for `__float__()` method returning a strict subclass of **`float`**: these methods will be required to return an instance of **`float`**.
  - Support for `__complex__()` method returning a strict subclass of **`complex`**: these methods will be required to return an instance of **`complex`**.
  - Passing a complex number as the *real* or *imag* argument in the **`complex()`** constructor is now deprecated; it should only be passed as a single positional argument. (Contributed by Serhiy Storchaka in gh-109218 [https://github.com/python/cpython/issues/109218].)
- **`calendar`**: las constantes `calendar.January` y `calendar.February` están obsoletas y reemplazadas por **`calendar.JANUARY`** y **`calendar.FEBRUARY`**. (Aportado por el Príncipe Roshan en gh-103636 [https://github.com/python/cpython/issues/103636].)
- **`codecs`**: use **`open()`** instead of **`codecs.open()`**. (gh-133038 [https://github.com/python/cpython/issues/133038])
- **`codeobject.co_lnotab`**: use the **`codeobject.co_lines()`** method instead.
- **`datetime`**:
  - **`utcnow()`**: use `datetime.datetime.now(tz=datetime.UTC)`.
  - **`utcfromtimestamp()`**: use `datetime.datetime.fromtimestamp(timestamp, tz=datetime.UTC)`.
- **`gettext`**: Plural value must be an integer.
- **`importlib`**:
  - **`cache_from_source()`** *debug_override* parameter is deprecated: use the *optimization* parameter instead.
- **`importlib.metadata`**:

  - `EntryPoints` tuple interface.
  - Implicit `None` on return values.
- **`logging`**: the `warn()` method has been deprecated since Python 3.3, use **`warning()`** instead.
- **`mailbox`**: Use of StringIO input and text mode is deprecated, use BytesIO and binary mode instead.
- **`os`**: Calling **`os.register_at_fork()`** in multi-threaded process.
- **`pydoc.ErrorDuringImport`**: A tuple value for *exc_info* parameter is deprecated, use an exception instance.
- **`re`**: More strict rules are now applied for numerical group references and group names in regular expressions. Only sequence of ASCII digits is now accepted as a numerical reference. The group name in bytes patterns and replacement strings can now only contain ASCII letters and digits and underscore. (Contributed by Serhiy Storchaka in gh-91760 [https://github.com/python/cpython/issues/91760].)
- **`sre_compile`**, **`sre_constants`** and **`sre_parse`** modules.
- **`shutil`**: **`rmtree()`**"s *onerror* parameter is deprecated in Python 3.12; use the *onexc* parameter instead.
- **`ssl`** options and protocols:
  - **`ssl.SSLContext`** without protocol argument is deprecated.
  - **`ssl.SSLContext`**: **`set_npn_protocols()`** and **`selected_npn_protocol()`** are deprecated: use ALPN instead.
  - `ssl.OP_NO_SSL*` options
  - `ssl.OP_NO_TLS*` options
  - `ssl.PROTOCOL_SSLv3`
  - `ssl.PROTOCOL_TLS`
  - `ssl.PROTOCOL_TLSv1`
  - `ssl.PROTOCOL_TLSv1_1`
  - `ssl.PROTOCOL_TLSv1_2`
  - `ssl.TLSVersion.SSLv3`
  - `ssl.TLSVersion.TLSv1`
  - `ssl.TLSVersion.TLSv1_1`
- **`threading`** methods:
  - **`threading.Condition.notifyAll()`**: use **`notify_all()`**.
  - **`threading.Event.isSet()`**: use **`is_set()`**.

  - **`threading.Thread.isDaemon()`**, **`threading.Thread.setDaemon()`**: use **`threading.Thread.daemon`** attribute.
  - **`threading.Thread.getName()`**, **`threading.Thread.setName()`**: use **`threading.Thread.name`** attribute.
  - **`threading.currentThread()`**: use **`threading.current_thread()`**.
  - **`threading.activeCount()`**: use **`threading.active_count()`**.
- **`typing.Text`** (gh-92332 [https://github.com/python/cpython/issues/92332]).
- The internal class `typing._UnionGenericAlias` is no longer used to implement **`typing.Union`**. To preserve compatibility with users using this private class, a compatibility shim will be provided until at least Python 3.17. (Contributed by Jelle Zijlstra in gh-105499 [https://github.com/python/cpython/issues/105499].)
- **`unittest.IsolatedAsyncioTestCase`**: it is deprecated to return a value that is not `None` from a test case.
- **`urllib.parse`** deprecated functions: **`urlparse()`** instead
  - `splitattr()`
  - `splithost()`
  - `splitnport()`
  - `splitpasswd()`
  - `splitport()`
  - `splitquery()`
  - `splittag()`
  - `splittype()`
  - `splituser()`
  - `splitvalue()`
  - `to_bytes()`
- **`wsgiref`**: `SimpleHandler.stdout.write()` should not do partial writes.
- **`xml.etree.ElementTree`**: Testing the truth value of an **`Element`** is deprecated. In a future release it will always return `True`. Prefer explicit `len(elem)` or `elem is not None` tests instead.
- **`sys._clear_type_cache()`** is deprecated: use **`sys._clear_internal_caches()`** instead.

Eliminado

asynchat y asyncore

- Estos dos módulos se eliminaron según lo programado en **PEP 594** [https://peps.python.org/pep-0594/] y quedaron obsoletos en Python 3.6. Utilice **`asyncio`** en su lugar. (Aportado por Nikita Sobolev en gh-96580 [https://github.com/python/cpython/issues/96580].)

configparser

- Varios nombres obsoletos en **`configparser`** en 3.2 se han eliminado según gh-89336 [https://github.com/python/cpython/issues/89336]:
  - **`configparser.ParsingError`** ya no tiene un atributo o argumento `filename`. Utilice el atributo y el argumento `source` en su lugar.
  - **`configparser`** ya no tiene una clase `SafeConfigParser`. Utilice en su lugar el nombre **`ConfigParser`** más corto.
  - **`configparser.ConfigParser`** ya no tiene un método `readfp`. Utilice **`read_file()`** en su lugar.

distutils

- Elimine el paquete **`distutils`**. Quedó obsoleto en Python 3.10 por **PEP 632** [https://peps.python.org/pep-0632/] «Módulo distutils obsoleto». Para proyectos que todavía usan `distutils` y no se pueden actualizar a otra cosa, se puede instalar el proyecto `setuptools`: todavía proporciona `distutils`. (Aportado por Victor Stinner en gh-92584 [https://github.com/python/cpython/issues/92584].)

ensurepip

- Retire la rueda de herramientas de configuración incluida en **`ensurepip`** y deje de instalar herramientas de configuración en entornos creados por **`venv`**.

`pip (>= 22.1)` no requiere la instalación de herramientas de configuración en el entorno. Los paquetes basados en `setuptools` (y `distutils`) aún se pueden usar con `pip install`, ya que pip proporcionará `setuptools` en el entorno de compilación que utiliza para crear un paquete.

`easy_install`, `pkg_resources`, `setuptools` y `distutils` ya no se proporcionan de forma predeterminada en entornos creados con `venv` o arrancados con `ensurepip`, ya que forman parte del paquete `setuptools`. Para proyectos que dependen de estos en tiempo de ejecución, el proyecto `setuptools` debe declararse como una dependencia e instalarse por separado (generalmente, usando pip).

(Aportado por Pradyun Gedam en gh-95299 [https://github.com/python/cpython/issues/95299].)

enum

- Elimine `EnumMeta.__getattr__` de **`enum`**, que ya no es necesario para acceder al atributo de enumeración. (Aportado por Ethan Furman en gh-95083 [https://github.com/python/cpython/issues/95083].)

ftplib

- Elimine el atributo de clase `FTP_TLS.ssl_version` de **`ftplib`**: utilice el parámetro *context* en su lugar. (Aportado por Victor Stinner en gh-94172 [https://github.com/python/cpython/issues/94172].)

gzip

- Elimine el atributo `filename` del **`gzip.GzipFile`** de **`gzip`**, en desuso desde Python 2.6; utilice el atributo **`name`** en su lugar. En modo de escritura, el atributo `filename` agregaba la extensión de archivo `'.gz'` si no estaba presente. (Aportado por Victor Stinner en gh-94196 [https://github.com/python/cpython/issues/94196].)

hashlib

- Remove the pure Python implementation of `hashlib`’s `hashlib.pbkdf2_hmac()`, deprecated in Python 3.10. Python 3.10 and newer requires OpenSSL 1.1.1 (**PEP 644** [https://peps.python.org/pep-0644/]): this OpenSSL version provides a C implementation of `pbkdf2_hmac()` which is faster. (Contributed by Victor Stinner in gh-94199 [https://github.com/python/cpython/issues/94199].)

importlib

- Muchas limpiezas anteriormente obsoletas en `importlib` ahora se han completado:
    - References to, and support for `module_repr()` has been removed. (Contributed by Barry Warsaw in gh-97850 [https://github.com/python/cpython/issues/97850].)
    - Se han eliminado `importlib.util.set_package`, `importlib.util.set_loader` y `importlib.util.module_for_loader`. (Contribución de Brett Cannon y Nikita Sobolev en gh-65961 [https://github.com/python/cpython/issues/65961] y gh-97850 [https://github.com/python/cpython/issues/97850].)
    - Se eliminó la compatibilidad con las API `find_loader()` y `find_module()`. (Aportado por Barry Varsovia en gh-98040 [https://github.com/python/cpython/issues/98040].)
    - Se han eliminado `importlib.abc.Finder`, `pkgutil.ImpImporter` y `pkgutil.ImpLoader`. (Aportado por Barry Varsovia en gh-98040 [https://github.com/python/cpython/issues/98040].)

imp

- Se ha eliminado el módulo `imp`. (Aportado por Barry Varsovia en gh-98040 [https://github.com/python/cpython/issues/98040].)

  Para migrar consulte la siguiente tabla de correspondencias:

	importlib
`imp.NullImporter`	Insert None into `sys.path_importer_cache`
`imp.cache_from_source()`	**`importlib.util.cache_from_source()`**
`imp.find_module()`	**`importlib.util.find_spec()`**
`imp.get_magic()`	**`importlib.util.MAGIC_NUMBER`**
`imp.get_suffixes()`	**`importlib.machinery.SOURCE_SUFFIXES`**, **`importlib.machinery.EXTENSION_SUFFIXES`**, and **`importlib.machinery.BYTECODE_SUFFIXES`**
`imp.get_tag()`	**`sys.implementation.cache_tag`**
`imp.load_module()`	**`importlib.import_module()`**
`imp.new_module(name)`	`types.ModuleType(name)`
`imp.reload()`	**`importlib.reload()`**
`imp.source_from_cache()`	**`importlib.util.source_from_cache()`**
`imp.load_source()`	*Ver abajo*

Reemplace `imp.load_source()` con:

```
import importlib.util
import importlib.machinery

def load_source(modname, filename):
    loader = importlib.machinery.SourceFileLoader(m
    spec = importlib.util.spec_from_file_location(m
    module = importlib.util.module_from_spec(spec)
    # The module is always executed and not cached
    # Uncomment the following line to cache the mod
    # sys.modules[module.__name__] = module
    loader.exec_module(module)
    return module
```

- Elimine las funciones y atributos de **`imp`** sin reemplazos:
    - Funciones no documentadas:
        - `imp.init_builtin()`
        - `imp.load_compiled()`
        - `imp.load_dynamic()`
        - `imp.load_package()`
    - `imp.lock_held()`, `imp.acquire_lock()`,

    `imp.release_lock()`: el esquema de bloqueo ha cambiado en Python 3.3 a bloqueos por módulo.
  - Constantes `imp.find_module()`: `SEARCH_ERROR`, `PY_SOURCE`, `PY_COMPILED`, `C_EXTENSION`, `PY_RESOURCE`, `PKG_DIRECTORY`, `C_BUILTIN`, `PY_FROZEN`, `PY_CODERESOURCE`, `IMP_HOOK`.

io

- Elimine `io.OpenWrapper` y `_pyio.OpenWrapper` de **`io`**, obsoletos en Python 3.10: simplemente use **`open()`** en su lugar. La función **`open()`** (**`io.open()`**) es una función incorporada. Desde Python 3.10, **`_pyio.open()`** también es un método estático. (Aportado por Victor Stinner en gh-94169 [https://github.com/python/cpython/issues/94169].)

locale

- Elimine la función **`locale.format()`** de **`locale`**, obsoleta en Python 3.7: use **`locale.format_string()`** en su lugar. (Aportado por Victor Stinner en gh-94226 [https://github.com/python/cpython/issues/94226].)

smtpd

- The `smtpd` module has been removed according to the schedule in **PEP 594** [https://peps.python.org/pep-0594/], having been deprecated in Python 3.4.7 and 3.5.4. Use the aiosmtpd [https://pypi.org/project/aiosmtpd/] PyPI module or any other **`asyncio`**-based server instead. (Contributed by Oleg Iarygin in gh-93243 [https://github.com/python/cpython/issues/93243].)

sqlite3

- Las siguientes características **`sqlite3`** no documentadas, obsoletas en Python 3.10, ahora se eliminan:
  - `sqlite3.enable_shared_cache()`
  - `sqlite3.OptimizedUnicode`

Si se debe utilizar una caché compartida, abra la base de datos en modo URI utilizando el parámetro de consulta `cache=shared`.

La fábrica de textos `sqlite3.OptimizedUnicode` ha sido un alias para **`str`** desde Python 3.3. El código que previamente configuró la fábrica de texto en `OptimizedUnicode` puede usar `str` explícitamente o confiar en el valor predeterminado que también es `str`.

(Aportado por Erlend E. Aasland en gh-92548 [https://github.com/python/cpython/issues/92548].)

ssl

- Elimine la función **`ssl.RAND_pseudo_bytes()`** de **`ssl`**, obsoleta en Python 3.6: use **`os.urandom()`** o **`ssl.RAND_bytes()`** en su lugar. (Aportado por Victor Stinner en gh-94199 [https://github.com/python/cpython/issues/94199].)
- Elimine la función **`ssl.match_hostname()`**. Quedó obsoleto en Python 3.7. OpenSSL realiza coincidencias de nombres de host desde Python 3.7, Python ya no usa la función **`ssl.match_hostname()`**. (Aportado por Victor Stinner en gh-94199 [https://github.com/python/cpython/issues/94199].)
- Remove the **`ssl.wrap_socket()`** function, deprecated in Python 3.7: instead, create a **`ssl.SSLContext`** object and call its **`ssl.SSLContext.wrap_socket`** method. Any package that still uses **`ssl.wrap_socket()`** is broken and insecure. The function neither sends a SNI TLS extension nor validates the server hostname. Code is subject to **CWE 295** [https://cwe.mitre.org/data/definitions/295.html] (Improper Certificate Validation). (Contributed by Victor Stinner in gh-94199 [https://github.com/python/cpython/issues/94199].)

unittest

- Elimine muchas características de **`unittest`** que están en desuso desde hace mucho tiempo:

- ○ Varios alias del método `TestCase`:

Alias obsoleto	Nombre del método	Obsoleto en
`failUnless`	`assertTrue()`	3.1
`failIf`	`assertFalse()`	3.1
`failUnlessEqual`	`assertEqual()`	3.1
`failIfEqual`	`assertNotEqual()`	3.1
`failUnlessAlmostEqual`	`assertAlmostEqual()`	3.1
`failIfAlmostEqual`	`assertNotAlmostEqual()`	3.1
`failUnlessRaises`	`assertRaises()`	3.1
`assert_`	`assertTrue()`	3.2
`assertEquals`	`assertEqual()`	3.2
`assertNotEquals`	`assertNotEqual()`	3.2
`assertAlmostEquals`	`assertAlmostEqual()`	3.2
`assertNotAlmostEquals`	`assertNotAlmostEqual()`	3.2
`assertRegexpMatches`	`assertRegex()`	3.2
`assertRaisesRegexp`	`assertRaisesRegex()`	3.2
`assertNotRegexpMatches`	`assertNotRegex()`	3.5

  Puede utilizar https://github.com/isidentical/teyit para modernizar automáticamente sus pruebas unitarias.

- ○ Método `assertDictContainsSubset` de `TestCase` indocumentado y roto (obsoleto en Python 3.2).

- ○ Undocumented `TestLoader.loadTestsFromModule` parameter *use_load_tests* (deprecated and ignored since Python 3.5).

- ○ Un alias de la clase `TextTestResult`: `_TextTestResult` (obsoleto en Python 3.2).

(Contribución de Serhiy Storchaka en gh-89325 [https://github.com/python/cpython/issues/89325].)

webbrowser

- Elimina la compatibilidad con navegadores obsoletos de `webbrowser`. Los navegadores eliminados incluyen: Grail, Mosaic, Netscape, Galeon, Skipstone, Iceape, Firebird y Firefox versiones 35 e inferiores (gh-102871 [https://github.com/python/cpython/issues/102871]).

xml.etree.ElementTree

- Elimine el método `ElementTree.Element.copy()` de la implementación pura de Python, obsoleto en Python 3.10, use la función **`copy.copy()`** en su lugar. La implementación C de **`xml.etree.ElementTree`** no tiene ningún método `copy()`, solo un método `__copy__()`. (Aportado por Victor Stinner en gh-94383 [https://github.com/python/cpython/issues/94383].)

zipimport

- Elimine los métodos `find_loader()` y `find_module()` de **`zipimport`**, obsoletos en Python 3.10: use el método `find_spec()` en su lugar. Consulte **PEP 451** [https://peps.python.org/pep-0451/] para conocer el fundamento. (Aportado por Victor Stinner en gh-94379 [https://github.com/python/cpython/issues/94379].)

Otros

- Elimine la regla `suspicious` de la documentación `Makefile` y `Doc/tools/rstlint.py`, ambas a favor de sphinx-lint [https://github.com/sphinx-contrib/sphinx-lint]. (Contribución de Julien Palard en gh-98179 [https://github.com/python/cpython/issues/98179].)
- Elimine los parámetros *keyfile* y *certfile* de los módulos **`ftplib`**, **`imaplib`**, **`poplib`** y **`smtplib`**, y los parámetros *key_file*, *cert_file* y *check_hostname* del módulo **`http.client`**, todos obsoletos desde Python 3.6. Utilice el parámetro *context* (*ssl_context* en **`imaplib`**) en su lugar. (Aportado por Victor Stinner en gh-94172 [https://github.com/python/cpython/issues/94172].)
- Elimine los trucos de compatibilidad `Jython` de varios módulos y pruebas stdlib. (Aportado por Nikita Sobolev en gh-99482 [https://github.com/python/cpython/issues/99482].)

- Elimine el indicador `_use_broken_old_ctypes_structure_semantics_` del módulo **`ctypes`**. (Aportado por Nikita Sobolev en gh-99285 [https://github.com/python/cpython/issues/99285].)

Portar a Python 3.12

Esta sección enumera los cambios descritos anteriormente y otras correcciones de errores que pueden requerir cambios en su código.

Cambios en la API de Python

- Ahora se aplican reglas más estrictas para las referencias numéricas de grupos y los nombres de grupos en expresiones regulares. Ahora sólo se acepta como referencia numérica la secuencia de dígitos ASCII. El nombre del grupo en patrones de bytes y cadenas de reemplazo ahora solo puede contener letras y dígitos ASCII y guiones bajos. (Contribución de Serhiy Storchaka en gh-91760 [https://github.com/python/cpython/issues/91760].)

- Elimine la funcionalidad `randrange()` en desuso desde Python 3.10. Anteriormente, `randrange(10.0)` se convertía sin pérdidas a `randrange(10)`. Ahora, genera un **`TypeError`**. Además, la excepción planteada para valores no enteros como `randrange(10.5)` o `randrange('10')` se cambió de **`ValueError`** a **`TypeError`**. Esto también evita errores en los que `randrange(1e25)` seleccionaría silenciosamente de un rango mayor que `randrange(10**25)`. (Originalmente sugerido por Serhiy Storchaka gh-86388 [https://github.com/python/cpython/issues/86388].)

- **`argparse.ArgumentParser`** cambió la codificación y el controlador de errores para leer argumentos de un archivo (por ejemplo, la opción `fromfile_prefix_chars`) de la codificación de texto predeterminada (por ejemplo, **`locale.getpreferredencoding(False)`**) a filesystem encoding and error handler. Los archivos de argumentos deben codificarse en UTF-8 en lugar de en la página de códigos ANSI en Windows.

- Remove the `asyncore`-based `smtpd` module deprecated in Python 3.4.7 and 3.5.4. A recommended replacement is the **`asyncio`**-based aiosmtpd [https://pypi.org/project/aiosmtpd/] PyPI module.

- **`shlex.split()`**: Pasar `None` por el argumento *s* ahora genera una excepción, en lugar de leer **`sys.stdin`**. La característica quedó obsoleta en Python 3.9. (Aportado por Victor Stinner en gh-94352 [https://github.com/python/cpython/issues/94352].)

- El módulo **`os`** ya no acepta rutas tipo bytes, como los tipos **`bytearray`** y **`memoryview`**: solo se acepta el tipo exacto **`bytes`** para cadenas de bytes. (Aportado por Victor Stinner en gh-98393 [https://github.com/python/cpython/issues/98393].)

- **`syslog.openlog()`** y **`syslog.closelog()`** ahora fallan si se usan en subintérpretes. **`syslog.syslog()`** aún se puede usar en subintérpretes, pero ahora solo si **`syslog.openlog()`** ya ha sido llamado en el intérprete principal. Estas nuevas restricciones no se aplican al intérprete principal, por lo que sólo un conjunto muy pequeño de usuarios podría verse afectado. Este cambio ayuda con el aislamiento del intérprete. Además, **`syslog`** es un contenedor de recursos globales de procesos, que se administran mejor desde el intérprete principal. (Aportado por Donghee Na en gh-99127 [https://github.com/python/cpython/issues/99127].)

- Se elimina el comportamiento de bloqueo no documentado de **`cached_property()`**, porque bloqueaba todas las instancias de la clase, lo que generaba una alta contención de bloqueo. Esto significa que una función de obtención de propiedades almacenadas en caché ahora podría ejecutarse más de una vez para una sola instancia, si dos subprocesos se ejecutan. Para la mayoría de las propiedades almacenadas en caché simples (por ejemplo, aquellas que son idempotentes y simplemente calculan un valor en función de otros atributos de la instancia), esto estará bien. Si se necesita sincronización, implemente el bloqueo dentro de la función de obtención de propiedades en caché o alrededor de puntos de acceso de subprocesos múltiples.

- `sys._current_exceptions()` ahora devuelve una asignación de thread-id a una instancia de excepción, en lugar de a una tupla `(typ, exc, tb)`. (Aportado por Irit Katriel en gh-103176 [https://github.com/python/cpython/issues/103176].)

- Al extraer archivos tar usando `tarfile` o `shutil.unpack_archive()`, pase el argumento *filter* para limitar las funciones que pueden resultar sorprendentes o peligrosas. Consulte Extraction filters para obtener más detalles.

- La salida de las funciones `tokenize.tokenize()` y `tokenize.generate_tokens()` ahora cambia debido a los cambios introducidos en **PEP 701** [https://peps.python.org/pep-0701/]. Esto significa que los tokens `STRING` ya no se emiten para cadenas f y los tokens descritos en **PEP 701** [https://peps.python.org/pep-0701/] ahora se producen en su lugar: `FSTRING_START`, `FSTRING_MIDDLE` y `FSTRING_END` ahora se emiten para partes de «cadena» de cadena f además de los tokens apropiados para la tokenización. en los componentes de la expresión. Por ejemplo, para la cadena f `f"start {1+1} end"`, la versión anterior del tokenizador emitió:

  ```
  1,0-1,18:           STRING         'f"start {1+1} e
  ```

  mientras que la nueva versión emite:

  ```
  1,0-1,2:            FSTRING_START  'f"'
  1,2-1,8:            FSTRING_MIDDLE 'start '
  1,8-1,9:            OP             '{'
  1,9-1,10:           NUMBER         '1'
  1,10-1,11:          OP             '+'
  1,11-1,12:          NUMBER         '1'
  1,12-1,13:          OP             '}'
  1,13-1,17:          FSTRING_MIDDLE ' end'
  1,17-1,18:          FSTRING_END    '"'
  ```

Además, puede haber algunos cambios de comportamiento menores como consecuencia de los cambios necesarios para admitir **PEP 701** [https://peps.python.org/pep-0701/]. Algunos de estos cambios incluyen:

  - El atributo `type` de los tokens emitidos al tokenizar algunos caracteres de Python no válidos, como `!`, ha cambiado de `ERRORTOKEN` a `OP`.
  - Las cadenas incompletas de una sola línea ahora también generan **`tokenize.TokenError`** como lo hacen las cadenas incompletas de varias líneas.
  - Algún código Python incompleto o no válido ahora genera **`tokenize.TokenError`** en lugar de devolver tokens `ERRORTOKEN` arbitrarios al tokenizarlo.
  - Ya no se admite la combinación de tabulaciones y espacios como sangría en el mismo archivo y generará un **`TabError`**.

- The **`threading`** module now expects the **`_thread`** module to have an `_is_main_interpreter` attribute. It is a function with no arguments that returns `True` if the current interpreter is the main interpreter.

  Any library or application that provides a custom `_thread` module should provide `_is_main_interpreter()`. (See gh-112826 [https://github.com/python/cpython/issues/112826].)

Cambios de compilación

- Python ya no usa `setup.py` para crear módulos de extensión C compartidos. Los parámetros de compilación, como encabezados y bibliotecas, se detectan en el script `configure`. Las extensiones están construidas por `Makefile`. La mayoría de las extensiones usan `pkg-config` y recurren a la detección manual. (Aportado por Christian Heimes en gh-93939 [https://github.com/python/cpython/issues/93939].)

- Ahora se requiere `va_start()` con dos parámetros, como `va_start(args, format),`, para compilar Python.

`va_start()` ya no se llama con un solo parámetro. (Aportado por Kumar Aditya en gh-93207 [https://github.com/python/cpython/issues/93207].)

- CPython ahora usa la opción ThinLTO como política predeterminada de optimización del tiempo de enlace si el compilador Clang acepta la marca. (Aportado por Donghee Na en gh-89536 [https://github.com/python/cpython/issues/89536].)

- Agregado la variable `COMPILEALL_OPTS` en `Makefile` para anular las opciones de **`compileall`** (predeterminada: `-j0`) en `make install`. También fusionó los 3 comandos `compileall` en un solo comando para crear archivos .pyc para todos los niveles de optimización (0, 1, 2) a la vez. (Aportado por Victor Stinner en gh-99289 [https://github.com/python/cpython/issues/99289].)

- Agregado tripletes de plataforma para LoongArch de 64 bits:

  ○ loongarch64-linux-gnusf
  ○ loongarch64-linux-gnuf32
  ○ loongarch64-linux-gnu

  (Aportado por Zhang Na en gh-90656 [https://github.com/python/cpython/issues/90656].)

- `PYTHON_FOR_REGEN` ahora requiere Python 3.10 o posterior.

- Autoconf 2.71 and aclocal 1.16.4 is now required to regenerate `configure`. (Contributed by Christian Heimes in gh-89886 [https://github.com/python/cpython/issues/89886].)

- Las compilaciones de Windows y los instaladores de macOS de python.org ahora usan OpenSSL 3.0.

Cambios en la API de C

Nuevas características

- **PEP 697** [https://peps.python.org/pep-0697/]: presente Unstable C API tier, destinado a herramientas de bajo nivel como depuradores y compiladores JIT. Esta API puede cambiar en cada versión menor de CPython sin advertencias de obsolescencia. Su contenido está marcado con el prefijo `PyUnstable_` en los nombres.

Constructores de objetos de código:

  - `PyUnstable_Code_New()` (renombrado de `PyCode_New`)
  - `PyUnstable_Code_NewWithPosOnlyArgs()` (renombrado de `PyCode_NewWithPosOnlyArgs`)

Almacenamiento adicional para objetos de código (**PEP 523** [https://peps.python.org/pep-0523/]):

  - `PyUnstable_Eval_RequestCodeExtraIndex()` (renombrado de `_PyEval_RequestCodeExtraIndex`)
  - `PyUnstable_Code_GetExtra()` (renombrado de `_PyCode_GetExtra`)
  - `PyUnstable_Code_SetExtra()` (renombrado de `_PyCode_SetExtra`)

Los nombres originales seguirán estando disponibles hasta que cambie la API respectiva.

(Aportado por Petr Viktorin en gh-101101 [https://github.com/python/cpython/issues/101101].)

- **PEP 697** [https://peps.python.org/pep-0697/]: agregue una API para extender tipos cuyo diseño de memoria de instancia es opaco:

  - **`PyType_Spec.basicsize`** puede ser cero o negativo para especificar heredar o ampliar el tamaño de la clase base.
  - Se agregaron **`PyObject_GetTypeData()`** y **`PyType_GetTypeDataSize()`** para permitir el acceso a datos de instancia específicos de subclase.
  - Se agregaron **`Py_TPFLAGS_ITEMS_AT_END`** y **`PyObject_GetItemData()`** para permitir extender de forma segura ciertos tipos de tamaño variable, incluido

    `PyType_Type`.
  - Se agregó **`Py_RELATIVE_OFFSET`** para permitir definir **`members`** en términos de una estructura específica de subclase.

  (Aportado por Petr Viktorin en gh-103509 [https://github.com/python/cpython/issues/103509].)

- Agregado la nueva función limited C API **`PyType_FromMetaclass()`**, que generaliza el **`PyType_FromModuleAndSpec()`** existente utilizando un argumento de metaclase adicional. (Aportado por Wenzel Jakob en gh-93012 [https://github.com/python/cpython/issues/93012].)

- Se agregó API para crear objetos que se pueden llamar usando the vectorcall protocol a Limited API:

  - **`Py_TPFLAGS_HAVE_VECTORCALL`**
  - **`PyVectorcall_NARGS()`**
  - **`PyVectorcall_Call()`**
  - **`vectorcallfunc`**

  El indicador **`Py_TPFLAGS_HAVE_VECTORCALL`** ahora se elimina de una clase cuando se reasigna el método **`__call__()`** de la clase. Esto hace que vectorcall sea seguro de usar con tipos mutables (es decir, tipos de montón sin el indicador inmutable, **`Py_TPFLAGS_IMMUTABLETYPE`**). Los tipos mutables que no anulan **`tp_call`** ahora heredan el indicador `Py_TPFLAGS_HAVE_VECTORCALL`. (Aportado por Petr Viktorin en gh-93274 [https://github.com/python/cpython/issues/93274].)

  The **`Py_TPFLAGS_MANAGED_DICT`** and **`Py_TPFLAGS_MANAGED_WEAKREF`** flags have been added. This allows extensions classes to support object **`__dict__`** and weakrefs with less bookkeeping, using less memory and with faster access.

- Se agregó API para realizar llamadas usando the vectorcall protocol a Limited API:

    - `PyObject_Vectorcall()`
    - `PyObject_VectorcallMethod()`
    - `PY_VECTORCALL_ARGUMENTS_OFFSET`

  Esto significa que tanto el extremo entrante como el saliente del protocolo de llamada vectorial ahora están disponibles en el Limited API. (Aportado por Wenzel Jakob en gh-98586 [https://github.com/python/cpython/issues/98586].)

- Agregado dos nuevas funciones públicas, `PyEval_SetProfileAllThreads()` y `PyEval_SetTraceAllThreads()`, que permiten configurar funciones de seguimiento y creación de perfiles en todos los subprocesos en ejecución además del que realiza la llamada. (Aportado por Pablo Galindo en gh-93503 [https://github.com/python/cpython/issues/93503].)

- Agregado la nueva función `PyFunction_SetVectorcall()` a la API de C que establece el campo de llamada vectorial de un `PyFunctionObject` determinado. (Aportado por Andrew Frost en gh-92257 [https://github.com/python/cpython/issues/92257].)

- La API de C ahora permite registrar devoluciones de llamada a través de `PyDict_AddWatcher()`, `PyDict_Watch()` y API relacionadas para ser llamadas cada vez que se modifica un diccionario. Está pensado para optimizar intérpretes, compiladores JIT o depuradores. (Aportado por Carl Meyer en gh-91052 [https://github.com/python/cpython/issues/91052].)

- Agregado las API `PyType_AddWatcher()` y `PyType_Watch()` para registrar devoluciones de llamadas y recibir notificaciones sobre cambios en un tipo. (Aportado por Carl Meyer en gh-91051 [https://github.com/python/cpython/issues/91051].)

- Agregado las API `PyCode_AddWatcher()` y `PyCode_ClearWatcher()` para registrar devoluciones de llamadas y recibir notificaciones sobre la creación y destrucción de objetos de código. (Aportado por Itamar Oren en gh-91054 [https://github.com/python/cpython/issues/91054].)

- Agregado las funciones **`PyFrame_GetVar()`** y **`PyFrame_GetVarString()`** para obtener una variable de marco por su nombre. (Aportado por Victor Stinner en gh-91248 [https://github.com/python/cpython/issues/91248].)

- Agregado **`PyErr_GetRaisedException()`** y **`PyErr_SetRaisedException()`** para guardar y restaurar la excepción actual. Estas funciones devuelven y aceptan un único objeto de excepción, en lugar de los argumentos triples de los ahora obsoletos **`PyErr_Fetch()`** y **`PyErr_Restore()`**. Esto es menos propenso a errores y un poco más eficiente. (Aportado por Mark Shannon en gh-101578 [https://github.com/python/cpython/issues/101578].)

- Agregado `_PyErr_ChainExceptions1`, que toma una instancia de excepción, para reemplazar la API heredada `_PyErr_ChainExceptions`, que ahora está en desuso. (Aportado por Mark Shannon en gh-101578 [https://github.com/python/cpython/issues/101578].)

- Agregado **`PyException_GetArgs()`** y **`PyException_SetArgs()`** como funciones convenientes para recuperar y modificar el **`args`** pasado al constructor de la excepción. (Aportado por Mark Shannon en gh-101578 [https://github.com/python/cpython/issues/101578].)

- Agregado **`PyErr_DisplayException()`**, que toma una instancia de excepción, para reemplazar la API heredada **`PyErr_Display()`**. (Aportado por Irit Katriel en gh-102755 [https://github.com/python/cpython/issues/102755]).

- PEP 683 [https://peps.python.org/pep-0683/]: presente *Immortal Objects*, que permite que los objetos omitan los recuentos de referencias y los cambios relacionados en C-API:

    - `_Py_IMMORTAL_REFCNT`: el recuento de referencias que define un objeto
      como inmortal.

  - `_Py_IsImmortal` Comprueba si un objeto tiene el recuento de referencia inmortal.
  - `PyObject_HEAD_INIT` Esto ahora inicializará el recuento de referencias `_Py_IMMORTAL_REFCNT` cuando se utiliza con `Py_BUILD_CORE`.
  - `SSTATE_INTERNED_IMMORTAL` Un identificador para objetos Unicode internos que son inmortales.
  - `SSTATE_INTERNED_IMMORTAL_STATIC` Un identificador para Unicode interno Objetos inmortales y estáticos.
  - `sys.getunicodeinternedsize` Esto devuelve el número total de Unicode objetos que han sido internados. Esto ahora es necesario para que `refleak.py` rastree correctamente los recuentos de referencia y los bloques asignados.

  (Contribuido por Eddie Elizondo en gh-84436 [https://github.com/python/cpython/issues/84436].)

- PEP 684 [https://peps.python.org/pep-0684/]: agregue la nueva función **`Py_NewInterpreterFromConfig()`** y **`PyInterpreterConfig`**, que pueden usarse para crear subintérpretes con sus propios GIL. (Consulte PEP 684: un GIL por intérprete para obtener más información). (Contribución de Eric Snow en gh-104110 [https://github.com/python/cpython/issues/104110].)

- En la versión limitada de C API 3.12, las funciones **`Py_INCREF()`** y **`Py_DECREF()`** ahora se implementan como llamadas de función opacas para ocultar los detalles de implementación. (Aportado por Victor Stinner en gh-105387 [https://github.com/python/cpython/issues/105387].)

Portar a Python 3.12

- Se han eliminado las API Unicode heredadas basadas en la representación `Py_UNICODE*`. Migre a API basadas en UTF-8 o `wchar_t*`.

- Las funciones de análisis de argumentos como **`PyArg_ParseTuple()`** ya no admiten el formato basado en `Py_UNICODE*` (por ejemplo, `u`, `Z`). Migre a otros formatos para Unicode como `s`, `z`, `es` y `U`.

- `tp_weaklist` para todos los tipos integrados estáticos siempre es `NULL`. Este es un campo solo interno en `PyTypeObject`, pero señalamos el cambio en caso de que alguien acceda al campo directamente de todos modos. Para evitar roturas, considere utilizar la C-API pública existente o, si es necesario, la macro `_PyObject_GET_WEAKREFS_LISTPTR()` (solo interna).

- Es posible que este **`PyTypeObject.tp_subclasses`** solo interno ya no sea un puntero de objeto válido. Su tipo se cambió a void* para reflejar esto. Mencionamos esto en caso de que alguien acceda directamente al campo interno.

  To get a list of subclasses, call the Python method **`__subclasses__()`** (using **`PyObject_CallMethod()`**, for example).

- Agregado soporte para más opciones de formato (alineación a la izquierda, octales, hexadecimales en mayúsculas, cadenas **`intmax_t`**, **`ptrdiff_t`**, **`wchar_t`** C, ancho variable y precisión) en **`PyUnicode_FromFormat()`** y **`PyUnicode_FromFormatV()`**. (Aportado por Serhiy Storchaka en gh-98836 [https://github.com/python/cpython/issues/98836].)

- Un carácter de formato no reconocido en **`PyUnicode_FromFormat()`** y **`PyUnicode_FromFormatV()`** ahora establece un **`SystemError`**. En versiones anteriores, provocaba que el resto de la cadena de formato se copiara tal cual en la cadena de resultado y se descartaran los argumentos adicionales. (Aportado por Serhiy Storchaka en gh-95781 [https://github.com/python/cpython/issues/95781].)

- Se corrigió la ubicación incorrecta de los letreros en `PyUnicode_FromFormat()` y `PyUnicode_FromFormatV()`. (Aportado por Philip Georgi en gh-95504 [https://github.com/python/cpython/issues/95504].)

- Extension classes wanting to add a `__dict__` or weak reference slot should use `Py_TPFLAGS_MANAGED_DICT` and `Py_TPFLAGS_MANAGED_WEAKREF` instead of `tp_dictoffset` and `tp_weaklistoffset`, respectively. The use of `tp_dictoffset` and `tp_weaklistoffset` is still supported, but does not fully support multiple inheritance (gh-95589 [https://github.com/python/cpython/issues/95589]), and performance may be worse. Classes declaring `Py_TPFLAGS_MANAGED_DICT` must call `_PyObject_VisitManagedDict()` and `_PyObject_ClearManagedDict()` to traverse and clear their instance's dictionaries. To clear weakrefs, call `PyObject_ClearWeakRefs()`, as before.

- La función `PyUnicode_FSDecoder()` ya no acepta rutas de tipo bytes, como los tipos `bytearray` y `memoryview`: solo se acepta el tipo `bytes` exacto para cadenas de bytes. (Aportado por Victor Stinner en gh-98393 [https://github.com/python/cpython/issues/98393].)

- Las macros `Py_CLEAR`, `Py_SETREF` y `Py_XSETREF` ahora solo evalúan sus argumentos una vez. Si un argumento tiene efectos secundarios, estos efectos secundarios ya no se duplican. (Aportado por Victor Stinner en gh-98724 [https://github.com/python/cpython/issues/98724].)

- El indicador de error del intérprete ahora siempre está normalizado. Esto significa que `PyErr_SetObject()`, `PyErr_SetString()` y las otras funciones que configuran el indicador de error ahora normalizan la excepción antes de almacenarla. (Aportado por Mark Shannon en gh-101578 [https://github.com/python/cpython/issues/101578].)

- `_Py_RefTotal` ya no tiene autoridad y solo se conserva por compatibilidad con ABI. Tenga en cuenta que es un global interno y solo está disponible en compilaciones de depuración. Si lo está utilizando, deberá comenzar a utilizar `_Py_GetGlobalRefTotal()`.

- Las siguientes funciones ahora seleccionan una metaclase apropiada para el tipo recién creado:

  - **`PyType_FromSpec()`**
  - **`PyType_FromSpecWithBases()`**
  - **`PyType_FromModuleAndSpec()`**

  La creación de clases cuya metaclase anule **`tp_new`** está en desuso y en Python 3.14+ no estará permitida. Tenga en cuenta que estas funciones ignoran `tp_new` de la metaclase, lo que posiblemente permita una inicialización incompleta.

  Tenga en cuenta que **`PyType_FromMetaclass()`** (agregado en Python 3.12) ya no permite la creación de clases cuya metaclase anule `tp_new` (**`__new__()`** en Python).

  Dado que `tp_new` anula casi todo lo que hacen las funciones `PyType_From*`, las dos son incompatibles entre sí. El comportamiento existente (ignorar la metaclase durante varios pasos de la creación de tipos) no es seguro en general, ya que las (meta)clases suponen que se llamó a `tp_new`. No existe una solución general sencilla. Uno de los siguientes puede funcionar para usted:

  - Si controlas la metaclase, evita usar `tp_new` en ella:
    - Si se puede omitir la inicialización, se puede realizar en **`tp_init`**.
    - Si no es necesario crear una instancia de la metaclase desde Python, configure su `tp_new` en `NULL` usando el indicador **`Py_TPFLAGS_DISALLOW_INSTANTIATION`**. Esto lo hace aceptable para funciones `PyType_From*`.
  - Evite las funciones `PyType_From*`: si no necesita funciones específicas de C (ranuras o configuración del

tamaño de la instancia), cree tipos mediante la metaclase calling.
  - Si *know* puede omitir `tp_new` de forma segura, filtre la advertencia de obsolescencia usando **`warnings.catch_warnings()`** de Python.

- **`PyOS_InputHook`** y **`PyOS_ReadlineFunctionPointer`** ya no se llaman en subinterpreters. Esto se debe a que los clientes generalmente dependen del estado global de todo el proceso (ya que estas devoluciones de llamada no tienen forma de recuperar el estado del módulo de extensión).

  Esto también evita situaciones en las que las extensiones pueden encontrarse ejecutándose en un subintérprete que no admiten (o que aún no se han cargado). Consulte gh-104668 [https://github.com/python/cpython/issues/104668] para obtener más información.

- **`PyLongObject`** ha tenido cambios internos para un mejor rendimiento. Aunque las partes internas de **`PyLongObject`** son privadas, algunos módulos de extensión las utilizan. Ya no se debe acceder directamente a los campos internos, sino que se deben utilizar las funciones API que comienzan con `PyLong_...`. Se proporcionan dos nuevas funciones API *unstable* para un acceso eficiente al valor de **`PyLongObject`**s que cabe en una sola palabra de máquina:

  - **`PyUnstable_Long_IsCompact()`**
  - **`PyUnstable_Long_CompactValue()`**

- Ahora se requiere que los asignadores personalizados, configurados a través de **`PyMem_SetAllocator()`**, sean seguros para subprocesos, independientemente del dominio de memoria. Los asignadores que no tienen su propio estado, incluidos los «ganchos», no se ven afectados. Si su asignador personalizado aún no es seguro para subprocesos y necesita orientación, cree una nueva incidencia en GitHub y CC `@ericsnowcurrently`.

Obsoleto

- De acuerdo con **PEP 699** [https://peps.python.org/pep-0699/], el campo `ma_version_tag` en `PyDictObject` está obsoleto para los módulos de extensión. Acceder a este campo generará una advertencia del compilador en el momento de la compilación. Este campo se eliminará en Python 3.14. (Contribución de Ramvikrams y Kumar Aditya en gh-101193 [https://github.com/python/cpython/issues/101193]. PEP de Ken Jin.)

- Variable de configuración global obsoleta:
  - `Py_DebugFlag`: use `PyConfig.parser_debug`
  - `Py_VerboseFlag`: use `PyConfig.verbose`
  - `Py_QuietFlag`: use `PyConfig.quiet`
  - `Py_InteractiveFlag`: use `PyConfig.interactive`
  - `Py_InspectFlag`: use `PyConfig.inspect`
  - `Py_OptimizeFlag`: use `PyConfig.optimization_level`
  - `Py_NoSiteFlag`: use `PyConfig.site_import`
  - `Py_BytesWarningFlag`: use `PyConfig.bytes_warning`
  - `Py_FrozenFlag`: use `PyConfig.pathconfig_warnings`
  - `Py_IgnoreEnvironmentFlag`: use `PyConfig.use_environment`
  - `Py_DontWriteBytecodeFlag`: use `PyConfig.write_bytecode`
  - `Py_NoUserSiteDirectory`: use `PyConfig.user_site_directory`
  - `Py_UnbufferedStdioFlag`: use `PyConfig.buffered_stdio`
  - `Py_HashRandomizationFlag`: use `PyConfig.use_hash_seed` y `PyConfig.hash_seed`
  - `Py_IsolatedFlag`: use `PyConfig.isolated`
  - `Py_LegacyWindowsFSEncodingFlag`: use `PyPreConfig.legacy_windows_fs_encoding`
  - `Py_LegacyWindowsStdioFlag`: use `PyConfig.legacy_windows_stdio`
  - `Py_FileSystemDefaultEncoding`: use

    - `PyConfig.filesystem_encoding`
    - **`Py_HasFileSystemDefaultEncoding`**: use `PyConfig.filesystem_encoding`
    - **`Py_FileSystemDefaultEncodeErrors`**: use `PyConfig.filesystem_errors`
    - **`Py_UTF8Mode`**: use `PyPreConfig.utf8_mode` (ver `Py_PreInitialize()`)

  La API `Py_InitializeFromConfig()` debe usarse con `PyConfig` en su lugar. (Aportado por Victor Stinner en gh-77782 [https://github.com/python/cpython/issues/77782].)

- La creación de `immutable types` con bases mutables está obsoleta y se deshabilitará en Python 3.14. (gh-95388 [https://github.com/python/cpython/issues/95388])

- El encabezado `structmember.h` está en desuso, aunque sigue estando disponible y no hay planes para eliminarlo.

  Su contenido ahora está disponible simplemente incluyendo `Python.h`, con un prefijo `Py` agregado si faltara:

    - `PyMemberDef`, `PyMember_GetOne()` y `PyMember_SetOne()`
    - Escriba macros como `Py_T_INT`, `Py_T_DOUBLE`, etc. (anteriormente `T_INT`, `T_DOUBLE`, etc.)
    - Las banderas `Py_READONLY` (anteriormente `READONLY`) y `Py_AUDIT_READ` (anteriormente todas en mayúsculas)

  Varios elementos no están expuestos desde `Python.h`:

    - `T_OBJECT` (use `Py_T_OBJECT_EX`)
    - `T_NONE` (anteriormente indocumentado y bastante peculiar)
    - La macro `WRITE_RESTRICTED` que no hace nada.
    - Las macros `RESTRICTED` y `READ_RESTRICTED`, equivalentes a `Py_AUDIT_READ`.
    - En algunas configuraciones, `<stddef.h>` no se incluye en `Python.h`. Debe incluirse manualmente cuando se utiliza `offsetof()`.

El encabezado obsoleto continúa proporcionando su contenido original con los nombres originales. Su código antiguo puede permanecer sin cambios, a menos que las macros de inclusión y sin espacio de nombres adicionales le molesten mucho.

(Contribuido en gh-47146 [https://github.com/python/cpython/issues/47146] por Petr Viktorin, basado en trabajos anteriores de Alexander Belopolsky y Matthias Braun.)

- `PyErr_Fetch()` y `PyErr_Restore()` están en desuso. Utilice `PyErr_GetRaisedException()` y `PyErr_SetRaisedException()` en su lugar. (Aportado por Mark Shannon en gh-101578 [https://github.com/python/cpython/issues/101578].)

- `PyErr_Display()` está en desuso. Utilice `PyErr_DisplayException()` en su lugar. (Aportado por Irit Katriel en gh-102755 [https://github.com/python/cpython/issues/102755]).

- `_PyErr_ChainExceptions` está en desuso. Utilice `_PyErr_ChainExceptions1` en su lugar. (Aportado por Irit Katriel en gh-102192 [https://github.com/python/cpython/issues/102192].)

- El uso de `PyType_FromSpec()`, `PyType_FromSpecWithBases()` o `PyType_FromModuleAndSpec()` para crear una clase cuya metaclase anula `tp_new` está en desuso. En su lugar, llame a la metaclase.

Pending removal in Python 3.14

- El campo `ma_version_tag` en `PyDictObject` para módulos de extensión (**PEP 699** [https://peps.python.org/pep-0699/]; gh-101193 [https://github.com/python/cpython/issues/101193]).
- Creando `immutable types` con bases mutables (gh-95388 [https://github.com/python/cpython/issues/95388]).

Pending removal in Python 3.15

- The `PyImport_ImportModuleNoBlock()`: Use `PyImport_ImportModule()` instead.
- `PyWeakref_GetObject()` and `PyWeakref_GET_OBJECT()`: Use `PyWeakref_GetRef()` instead. The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get `PyWeakref_GetRef()` on Python 3.12 and older.
- `Py_UNICODE` type and the `Py_UNICODE_WIDE` macro: Use `wchar_t` instead.
- `PyUnicode_AsDecodedObject()`: Use `PyCodec_Decode()` instead.
- `PyUnicode_AsDecodedUnicode()`: Use `PyCodec_Decode()` instead; Note that some codecs (for example, «base64») may return a type other than `str`, such as `bytes`.
- `PyUnicode_AsEncodedObject()`: Use `PyCodec_Encode()` instead.
- `PyUnicode_AsEncodedUnicode()`: Use `PyCodec_Encode()` instead; Note that some codecs (for example, «base64») may return a type other than `bytes`, such as `str`.
- Python initialization functions, deprecated in Python 3.13:
  - `Py_GetPath()`: Use `PyConfig_Get("module_search_paths")` (`sys.path`) instead.
  - `Py_GetPrefix()`: Use `PyConfig_Get("base_prefix")` (`sys.base_prefix`) instead. Use `PyConfig_Get("prefix")` (`sys.prefix`) if virtual environments need to be handled.
  - `Py_GetExecPrefix()`: Use

    `PyConfig_Get("base_exec_prefix")` (`sys.base_exec_prefix`) instead. Use `PyConfig_Get("exec_prefix")` (`sys.exec_prefix`) if virtual environments need to be handled.
  - `Py_GetProgramFullPath()`: Use `PyConfig_Get("executable")` (`sys.executable`) instead.
  - `Py_GetProgramName()`: Use `PyConfig_Get("executable")` (`sys.executable`) instead.
  - `Py_GetPythonHome()`: Use `PyConfig_Get("home")` or the `PYTHONHOME` environment variable instead.

  The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get `PyConfig_Get()` on Python 3.13 and older.

- Functions to configure Python's initialization, deprecated in Python 3.11:

  - `PySys_SetArgvEx()`: Set `PyConfig.argv` instead.
  - `PySys_SetArgv()`: Set `PyConfig.argv` instead.
  - `Py_SetProgramName()`: Set `PyConfig.program_name` instead.
  - `Py_SetPythonHome()`: Set `PyConfig.home` instead.
  - `PySys_ResetWarnOptions()`: Clear `sys.warnoptions` and `warnings.filters` instead.

  La API `Py_InitializeFromConfig()` debe usarse con `PyConfig` en su lugar.

- Variables de configuración globales:

  - `Py_DebugFlag`: Use `PyConfig.parser_debug` or `PyConfig_Get("parser_debug")` instead.
  - `Py_VerboseFlag`: Use `PyConfig.verbose` or `PyConfig_Get("verbose")` instead.
  - `Py_QuietFlag`: Use `PyConfig.quiet` or `PyConfig_Get("quiet")` instead.

  - `Py_InteractiveFlag`: Use `PyConfig.interactive` or `PyConfig_Get("interactive")` instead.
  - `Py_InspectFlag`: Use `PyConfig.inspect` or `PyConfig_Get("inspect")` instead.
  - `Py_OptimizeFlag`: Use `PyConfig.optimization_level` or `PyConfig_Get("optimization_level")` instead.
  - `Py_NoSiteFlag`: Use `PyConfig.site_import` or `PyConfig_Get("site_import")` instead.
  - `Py_BytesWarningFlag`: Use `PyConfig.bytes_warning` or `PyConfig_Get("bytes_warning")` instead.
  - `Py_FrozenFlag`: Use `PyConfig.pathconfig_warnings` or `PyConfig_Get("pathconfig_warnings")` instead.
  - `Py_IgnoreEnvironmentFlag`: Use `PyConfig.use_environment` or `PyConfig_Get("use_environment")` instead.
  - `Py_DontWriteBytecodeFlag`: Use `PyConfig.write_bytecode` or `PyConfig_Get("write_bytecode")` instead.
  - `Py_NoUserSiteDirectory`: Use `PyConfig.user_site_directory` or `PyConfig_Get("user_site_directory")` instead.
  - `Py_UnbufferedStdioFlag`: Use `PyConfig.buffered_stdio` or `PyConfig_Get("buffered_stdio")` instead.
  - `Py_HashRandomizationFlag`: Use `PyConfig.use_hash_seed` and `PyConfig.hash_seed` or `PyConfig_Get("hash_seed")` instead.
  - `Py_IsolatedFlag`: Use `PyConfig.isolated` or `PyConfig_Get("isolated")` instead.
  - `Py_LegacyWindowsFSEncodingFlag`: Use `PyPreConfig.legacy_windows_fs_encoding` or `PyConfig_Get("legacy_windows_fs_encoding")` instead.
  - `Py_LegacyWindowsStdioFlag`: Use `PyConfig.legacy_windows_stdio` or

    `PyConfig_Get("legacy_windows_stdio")` instead.
  - **`Py_FileSystemDefaultEncoding`**, **`Py_HasFileSystemDefaultEncoding`**: Use `PyConfig.filesystem_encoding` or `PyConfig_Get("filesystem_encoding")` instead.
  - **`Py_FileSystemDefaultEncodeErrors`**: Use `PyConfig.filesystem_errors` or `PyConfig_Get("filesystem_errors")` instead.
  - **`Py_UTF8Mode`**: Use `PyPreConfig.utf8_mode` or `PyConfig_Get("utf8_mode")` instead. (see `Py_PreInitialize()`)

  The `Py_InitializeFromConfig()` API should be used with `PyConfig` to set these options. Or `PyConfig_Get()` can be used to get these options at runtime.

Pending removal in Python 3.16

- The bundled copy of `libmpdec`.

Pending removal in future versions

Las siguientes API están obsoletas y se eliminarán, aunque actualmente no hay una fecha programada para su eliminación.

- `Py_TPFLAGS_HAVE_FINALIZE`: Unneeded since Python 3.8.
- `PyErr_Fetch()`: Use `PyErr_GetRaisedException()` instead.
- `PyErr_NormalizeException()`: Use `PyErr_GetRaisedException()` instead.
- `PyErr_Restore()`: Use `PyErr_SetRaisedException()` instead.
- `PyModule_GetFilename()`: Use `PyModule_GetFilenameObject()` instead.
- `PyOS_AfterFork()`: Use `PyOS_AfterFork_Child()` instead.
- `PySlice_GetIndicesEx()`: Use `PySlice_Unpack()` and `PySlice_AdjustIndices()` instead.
- `PyUnicode_READY()`: Unneeded since Python 3.12

- **`PyErr_Display()`**: Use `PyErr_DisplayException()` instead.
- **`_PyErr_ChainExceptions()`**: Use **`_PyErr_ChainExceptions1()`** instead.
- **`PyBytesObject.ob_shash`** member: call `PyObject_Hash()` instead.
- API de almacenamiento local de subprocesos (TLS):
  - `PyThread_create_key()`: Use `PyThread_tss_alloc()` instead.
  - `PyThread_delete_key()`: Use `PyThread_tss_free()` instead.
  - `PyThread_set_key_value()`: Use `PyThread_tss_set()` instead.
  - `PyThread_get_key_value()`: Use `PyThread_tss_get()` instead.
  - `PyThread_delete_key_value()`: Use `PyThread_tss_delete()` instead.
  - `PyThread_ReInitTLS()`: Unneeded since Python 3.7.

Eliminado

- Elimine el archivo de encabezado `token.h`. Nunca hubo ninguna API C de tokenizador público. El archivo de encabezado `token.h` solo fue diseñado para ser utilizado por componentes internos de Python. (Aportado por Victor Stinner en gh-92651 [https://github.com/python/cpython/issues/92651].)
- Se han eliminado las API Unicode heredadas. Consulte **PEP 623** [https://peps.python.org/pep-0623/] para obtener más detalles.
  - **`PyUnicode_WCHAR_KIND`**
  - **`PyUnicode_AS_UNICODE()`**
  - **`PyUnicode_AsUnicode()`**
  - **`PyUnicode_AsUnicodeAndSize()`**
  - **`PyUnicode_AS_DATA()`**
  - **`PyUnicode_FromUnicode()`**
  - **`PyUnicode_GET_SIZE()`**
  - **`PyUnicode_GetSize()`**
  - **`PyUnicode_GET_DATA_SIZE()`**
- Elimine la macro de función

`PyUnicode_InternImmortal()`. (Aportado por Victor Stinner en gh-85858 [https://github.com/python/cpython/issues/85858].)

[1] [2] [3]

Qué hay de nuevo en Python 3.11

Editor:

Pablo Galindo Salgado

Este artículo explica las nuevas características de Python 3.11, en comparación con 3.10. Python 3.11 se lanzó el 24 de octubre de 2022. Para obtener detalles completos, consulte changelog.

Resumen – Aspectos destacados de la versión

- Python 3.11 es entre un 10 y un 60 % más rápido que Python 3.10. En promedio, medimos un aumento de velocidad de 1.25x en el conjunto de pruebas de referencia estándar. Ver CPython más rápido para más detalles.

Nuevas funciones de sintaxis:

- PEP 654: Grupos de excepción y except*

Nuevas funciones integradas:

- PEP 678: Las excepciones se pueden enriquecer con notas

Nuevos módulos de biblioteca estándar:

- **PEP 680** [https://peps.python.org/pep-0680/]: `tomllib`: soporte para analizar TOML [https://toml.io/] en la biblioteca estándar

Mejoras en el intérprete:

- PEP 657: Ubicaciones de errores detallados en rastreos
- Nueva opción de línea de comando `-P` y variable de entorno

`PYTHONSAFEPATH` a disable automatically prepending potentially unsafe paths a **`sys.path`**

Nuevas funciones de escritura:

- PEP 646: Genéricos Variádicos
- PEP 655: Marcado de elementos TypedDict individuales como requeridos o no requeridos
- PEP 673: tipo Self
- PEP 675: tipo de cadena literal arbitraria
- PEP 681: Transformaciones de clases de datos

Importantes depreciaciones, eliminaciones y restricciones:

- **PEP 594** [https://peps.python.org/pep-0594/]: Many legacy standard library modules have been deprecated y se eliminará en Python 3.13
- **PEP 624** [https://peps.python.org/pep-0624/]: Py_UNICODE encoder APIs have been removed
- **PEP 670** [https://peps.python.org/pep-0670/]: Macros converted to static inline functions

Nuevas características

PEP 657: Ubicaciones de errores detallados en rastreos

Al imprimir rastreos, el intérprete ahora señalará la expresión exacta que causó el error, en lugar de solo la línea. Por ejemplo:

```
Traceback (most recent call last):
  File "distance.py", line 11, in <module>
    print(manhattan_distance(p1, p2))
          ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "distance.py", line 6, in manhattan_distance
    return abs(point_1.x - point_2.x) + abs(point_1.y -
                           ^^^^^^^^^
AttributeError: 'NoneType' object has no attribute 'x'
```

Las versiones anteriores del intérprete apuntaban solo a la línea, por lo que resultaba ambiguo qué objeto era `None`. Estos errores mejorados también pueden ser útiles cuando se trata de objetos `dict` profundamente anidados y múltiples llamadas a funciones:

```
Traceback (most recent call last):
  File "query.py", line 37, in <module>
    magic_arithmetic('foo')
  File "query.py", line 18, in magic_arithmetic
    return add_counts(x) / 25
           ^^^^^^^^^^^^^
  File "query.py", line 24, in add_counts
    return 25 + query_user(user1) + query_user(user2)
                ^^^^^^^^^^^^^^^^^
  File "query.py", line 32, in query_user
    return 1 + query_count(db, response['a']['b']['c']['
                               ~~~~~~~~~~~~~~~~~~^^^^^
TypeError: 'NoneType' object is not subscriptable
```

Además de expresiones aritméticas complejas:

```
Traceback (most recent call last):
  File "calculation.py", line 54, in <module>
    result = (x / y / z) * (a / b / c)
              ~~~~~~^~~
ZeroDivisionError: division by zero
```

Además, la información utilizada por la función de rastreo mejorada está disponible a través de una API general, que se puede usar para correlacionar bytecode instructions con la ubicación del código fuente. Esta información se puede recuperar usando:

- El método `codeobject.co_positions()` en Python.
- La función `PyCode_Addr2Location()` en la API de C.

Ver **PEP 657** [https://peps.python.org/pep-0657/] para más detalles. (Aportado por Pablo Galindo, Batuhan Taskaya y Ammar Askar en bpo-43950 [https://bugs.python.org/issue?@action=redirect&bpo=43950].)

Nota

Esta característica requiere almacenar las posiciones de las columnas en Objetos código, lo que puede resultar en un pequeño aumento en el uso de la memoria del intérprete y el uso del disco para los archivos de Python compilados. Para evitar almacenar la información adicional y desactivar la impresión de la información de seguimiento adicional, utilice la opción de línea de comando `-X no_debug_ranges` o la variable de entorno `PYTHONNODEBUGRANGES`.

PEP 654: Grupos de excepción y `except*`

PEP 654 [https://peps.python.org/pep-0654/] presenta funciones de lenguaje que permiten que un programa genere y maneje múltiples excepciones no relacionadas simultáneamente. Los tipos integrados `ExceptionGroup` y `BaseExceptionGroup` permiten agrupar excepciones y generarlas juntas, y la nueva sintaxis `except*` generaliza `except` para hacer coincidir subgrupos de grupos de excepciones.

Ver **PEP 654** [https://peps.python.org/pep-0654/] para más detalles.

(Aportado por Irit Katriel en bpo-45292 [https://bugs.python.org/issue?@action=redirect&bpo=45292]. PEP escrito por Irit Katriel, Yury Selivanov y Guido van Rossum.)

PEP 678: Las excepciones se pueden enriquecer con notas

El método `add_note()` se agrega a `BaseException`. Se puede utilizar para enriquecer las excepciones con información de contexto que no está disponible en el momento en que se genera la excepción. Las notas añadidas aparecen en el rastreo predeterminado.

Ver **PEP 678** [https://peps.python.org/pep-0678/] para más detalles.

(Aportado por Irit Katriel en bpo-45607 [https://bugs.python.org/issue?@action=redirect&bpo=45607]. PEP escrito por Zac Hatfield-Dodds).

Mejoras en el iniciador de Windows `py.exe`

La copia del Python install manager incluida con Python 3.11 se ha actualizado significativamente. Ahora admite la sintaxis de empresa/etiqueta tal como se define en **PEP 514** [https://peps.python.org/pep-0514/] usando el argumento `-V:<company>/<tag>` en lugar del limitado `-<major>.< menor>`. Esto permite iniciar distribuciones distintas a `PythonCore`, la alojada en python.org [https://www.python.org].

Al usar los selectores `-V:`, se puede omitir la empresa o la etiqueta, pero se buscarán todas las instalaciones. Por ejemplo, `-V:OtherPython/` seleccionará la «mejor» etiqueta registrada para `OtherPython`, mientras que `-V:3.11` o `-V:/3.11` seleccionarán la «mejor» distribución con la etiqueta `3.11`.

Cuando se utilizan los argumentos obsoletos `-<principal>`, `-<principal>.<menor>`, `-<principal>-<bitness> ` o` `:samp:`-{<principal>}.{<menor>}-{<bitness>}`, se debe conservar todo el comportamiento existente de las versiones anteriores y solo se seleccionarán las versiones de `PythonCore`. Sin embargo, el sufijo `-64` ahora implica «no de 32 bits» (no necesariamente x86-64), ya que existen múltiples plataformas de 64 bits compatibles. Los tiempos de ejecución de 32 bits se detectan comprobando la etiqueta del tiempo de ejecución en busca de un sufijo `-32`. Todas las versiones de Python desde la 3.5 han incluido esto en sus versiones de 32 bits.

Nuevas funciones relacionadas con las sugerencias de tipo

Esta sección cubre los cambios principales que afectan las sugerencias de tipo **PEP 484** [https://peps.python.org/pep-0484/] y el módulo **`typing`**.

PEP 646: Genéricos Variádicos

PEP 484 [https://peps.python.org/pep-0484/] introdujo anteriormente **`TypeVar`**, lo que permite la creación de genéricos parametrizados con un solo tipo. **PEP 646** [https://peps.python.org/pep-0646/] añade **`TypeVarTuple`**, permitiendo la parametrización con un número de tipos *arbitrary*. En otras palabras, un **`TypeVarTuple`** es una variable de tipo *variadic* que permite los genéricos *variadic*.

Esto permite una amplia variedad de casos de uso. En particular, permite parametrizar con el arreglo *shape* el tipo de estructuras similares a arreglos en bibliotecas de computación numérica como NumPy y TensorFlow. Los verificadores de tipo estático ahora podrán detectar errores relacionados con la forma en el código que usa estas bibliotecas.

Ver **PEP 646** [https://peps.python.org/pep-0646/] para más detalles.

(Contribuido por Matthew Rahtz en bpo-43224 [https://bugs.python.org/issue?@action=redirect&bpo=43224], con contribuciones de Serhiy Storchaka y Jelle Zijlstra. PEP escrito por Mark Mendoza, Matthew Rahtz, Pradeep Kumar Srinivasan y Vincent Siles).

PEP 655: Marcado de elementos `TypedDict` individuales como requeridos o no requeridos

`Required` y **`NotRequired`** proporcionan una forma sencilla de marcar si deben estar presentes elementos individuales en un **`TypedDict`**. Anteriormente, esto solo era posible mediante la herencia.

Todos los campos siguen siendo obligatorios de forma predeterminada, a menos que el parámetro *total* se establezca en `False`, en cuyo caso todos los campos siguen sin ser obligatorios de forma predeterminada. Por ejemplo, lo siguiente especifica un **`TypedDict`** con una clave requerida y una no requerida:

```
class Movie(TypedDict):
   title: str
   year: NotRequired[int]

m1: Movie = {"title": "Black Panther", "year": 2018}  #
m2: Movie = {"title": "Star Wars"}  # OK (year is not re
m3: Movie = {"year": 2022}  # ERROR (missing required fi
```

La siguiente definición es equivalente:

```
class Movie(TypedDict, total=False):
   title: Required[str]
   year: int
```

Ver **PEP 655** [https://peps.python.org/pep-0655/] para más detalles.

(Aportado por David Foster y Jelle Zijlstra en bpo-47087 [https://bugs.python.org/issue?@action=redirect&bpo=47087]. PEP escrito por David Foster).

PEP 673: tipo `Self`

La nueva anotación **`Self`** proporciona una forma sencilla e intuitiva de anotar métodos que devuelven una instancia de su clase. Se comporta igual que el enfoque **specified in PEP 484** [https://peps.python.org/pep-0484/#annotating-instance-and-class-methods] basado en **`TypeVar`**, pero es más conciso y más fácil de seguir.

Los casos de uso comunes incluyen constructores alternativos proporcionados como **`classmethod`**s y métodos **`__enter__()`** que devuelven `self`:

```
class MyLock:
    def __enter__(self) -> Self:
        self.lock()
        return self


    ...

class MyInt:
    @classmethod
    def fromhex(cls, s: str) -> Self:
        return cls(int(s, 16))


    ...
```

`Self` también se puede usar para anotar parámetros de método o atributos del mismo tipo que su clase envolvente.

Ver **PEP 673** [https://peps.python.org/pep-0673/] para más detalles.

(Aportado por James Hilton-Balfe en bpo-46534 [https://bugs.python.org/issue?@action=redirect&bpo=46534]. PEP escrito por Pradeep Kumar Srinivasan y James Hilton-Balfe).

PEP 675: tipo de cadena literal arbitraria

La nueva anotación `LiteralString` se puede usar para indicar que un parámetro de función puede ser de cualquier tipo de cadena literal. Esto permite que una función acepte tipos de cadenas literales arbitrarias, así como cadenas creadas a partir de otras cadenas literales. Los verificadores de tipos pueden hacer cumplir que las funciones confidenciales, como las que ejecutan declaraciones SQL o comandos de shell, se llamen solo con argumentos estáticos, lo que brinda protección contra ataques de inyección.

Por ejemplo, una función de consulta SQL podría anotarse de la siguiente manera:

```
def run_query(sql: LiteralString) -> ...
    ...

def caller(
    arbitrary_string: str,
    query_string: LiteralString,
    table_name: LiteralString,
) -> None:
    run_query("SELECT * FROM students")      # ok
    run_query(query_string)                  # ok
    run_query("SELECT * FROM " + table_name)  # ok
    run_query(arbitrary_string)              # type che
    run_query(                               # type che
        f"SELECT * FROM students WHERE name = {arbitrary
    )
```

Ver **PEP 675** [https://peps.python.org/pep-0675/] para más detalles.

(Aportado por Jelle Zijlstra en bpo-47088 [https://bugs.python.org/issue?@action=redirect&bpo=47088]. PEP escrito por Pradeep Kumar Srinivasan y Graham Bleaney).

PEP 681: Transformaciones de clases de datos

`dataclass_transform` se puede usar para decorar una clase, una metaclase o una función que en sí misma es un decorador. La presencia de `@dataclass_transform()` le dice a un verificador de tipo estático que el objeto decorado realiza una «magia» en tiempo de ejecución que transforma una clase, dándole comportamientos similares a `dataclass`.

Por ejemplo:

```
# The create_model decorator is defined by a library.
@typing.dataclass_transform()
def create_model(cls: Type[T]) -> Type[T]:
    cls.__init__ = ...
    cls.__eq__ = ...
    cls.__ne__ = ...
    return cls

# The create_model decorator can now be used to create r
@create_model
class CustomerModel:
    id: int
    name: str

c = CustomerModel(id=327, name="Eric Idle")
```

Ver **PEP 681** [https://peps.python.org/pep-0681/] para más detalles.

(Aportado por Jelle Zijlstra en gh-91860 [https://github.com/python/cpython/issues/91860]. PEP escrito por Erik De Bonte y Eric Traut.)

PEP 563 puede no ser el futuro

PEP 563 [https://peps.python.org/pep-0563/] Evaluación pospuesta de anotaciones (`from __future__ import annotations` future statement) que originalmente se planeó para su lanzamiento en Python 3.10 se suspendió indefinidamente. Consulte this message from the Steering Council [https://mail.python.org/archives/list/python-dev@python.org/message/VIZEBX5EYMSYIJNDBF6DMUMZOCWHARSO/] para obtener más información.

Otros cambios de idioma

- Las expresiones de desempaquetado destacadas ahora se pueden usar en declaraciones `for`. (Consulte bpo-46725 [https://bugs.python.org/issue?@action=redirect&bpo=46725] para obtener más detalles).
- Ahora se permiten comprehensions asincrónicos dentro de las comprensiones en asynchronous functions. Las comprensiones externas implícitamente se vuelven asincrónicas en este caso. (Aportado por Serhiy Storchaka en bpo-33346 [https://bugs.python.org/issue?@action=redirect&bpo=33346].)
- Ahora se genera un `TypeError` en lugar de un `AttributeError` en declaraciones `with` y `contextlib.ExitStack.enter_context()` para objetos que no admiten el protocolo context manager, y en declaraciones `async with` y `contextlib.AsyncExitStack.enter_async_context()` para objetos que no admiten el protocolo asynchronous context manager. (Aportado por Serhiy Storchaka en bpo-12022 [https://bugs.python.org/issue?@action=redirect&bpo=12022] y bpo-44471 [https://bugs.python.org/issue?@action=redirect&bpo=44471]).
- Se agregó `object.__getstate__()`, que proporciona la implementación predeterminada del método `__getstate__()`. `copy`ando y `pickle`ando instancias de subclases de tipos integrados `bytearray`, `set`, `frozenset`, `collections.OrderedDict`, `collections.deque`, `weakref.WeakSet` y `datetime.tzinfo` ahora copian y seleccionan atributos de instancia implementados como slots. Este cambio tiene un efecto secundario no deseado: hace fallar a una pequeña minoría de proyectos Python existentes que no esperaban que existiera `object.__getstate__()`. Consulte los comentarios posteriores sobre gh-70766 [https://github.com/python/cpython/issues/70766] para obtener información sobre las soluciones alternativas que dicho código puede necesitar. (Aportado por Serhiy Storchaka en bpo-26579 [https://

bugs.python.org/issue?@action=redirect&bpo=26579].)

- Se agregó una opción de línea de comando `-P` y una variable de entorno `PYTHONSAFEPATH`, que deshabilitan la anteposición automática a `sys.path` del directorio del script cuando se ejecuta un script, o el directorio actual cuando se usa `-c` y `-m`. Esto garantiza que `import` solo recopile la biblioteca estándar y los módulos instalados, y evita el remedo involuntario o malicioso de los módulos con los de un directorio local (y normalmente el usuario puede escribir). (Aportado por Victor Stinner en gh-57684 [https://github.com/python/cpython/issues/57684].)
- Se agregó una opción `"z"` a Format specification mini-language que cambia negativo a cero positivo después de redondear a la precisión del formato. Ver **PEP 682** [https://peps.python.org/pep-0682/] para más detalles. (Aportado por John Belmonte en gh-90153 [https://github.com/python/cpython/issues/90153].)
- Ya no se aceptan bytes en `sys.path`. El soporte se interrumpió en algún momento entre Python 3.2 y 3.6, y nadie se dio cuenta hasta después del lanzamiento de Python 3.10.0. Además, recuperar la compatibilidad sería problemático debido a las interacciones entre `-b` y `sys.path_importer_cache` cuando hay una combinación de claves `str` y `bytes`. (Aportado por Thomas Grainger en gh-91181 [https://github.com/python/cpython/issues/91181].)

Otros cambios en la implementación de CPython

- Los métodos especiales `__complex__()` para `complex` y `__bytes__()` para `bytes` se implementan para admitir los protocolos `typing.SupportsComplex` y `typing.SupportsBytes`. (Contribución de Mark Dickinson y Donghee Na en bpo-24234 [https://bugs.python.org/issue?@action=redirect&bpo=24234]).
- `siphash13` se agrega como un nuevo algoritmo hash interno. Tiene propiedades de seguridad similares a `siphash24`, pero es un poco más rápido para entradas

largas. **`str`**, **`bytes`** y algunos otros tipos ahora lo usan como algoritmo predeterminado para **`hash()`**. **PEP 552** [https://peps.python.org/pep-0552/] hash-based .pyc files ahora también usa `siphash13`. (Aportado por Inada Naoki en bpo-29410 [https://bugs.python.org/issue?@action=redirect&bpo=29410].)

- Cuando una declaración **`raise`** sin parámetros vuelve a generar una excepción activa, el rastreo adjunto a esta excepción ahora siempre es `sys.exc_info()[1].__traceback__`. Esto significa que los cambios realizados en el rastreo en la cláusula **`except`** actual se reflejan en la excepción que se ha vuelto a generar. (Aportado por Irit Katriel en bpo-45711 [https://bugs.python.org/issue?@action=redirect&bpo=45711].)
- La representación del estado del intérprete de las excepciones manejadas (también conocido como `exc_info` o `_PyErr_StackItem`) ahora solo tiene el campo `exc_value`; Se han eliminado `exc_type` y `exc_traceback`, ya que se pueden derivar de `exc_value`. (Aportado por Irit Katriel en bpo-45711 [https://bugs.python.org/issue?@action=redirect&bpo=45711].)
- Se ha agregado un nuevo command line option, `AppendPath`, para el instalador de Windows. Se comporta de manera similar a `PrependPath`, pero agrega los directorios de instalación y scripts en lugar de anteponerlos. (Aportado por Bastian Neuburger en bpo-44934 [https://bugs.python.org/issue?@action=redirect&bpo=44934].)
- El campo **`PyConfig.module_search_paths_set`** ahora debe establecerse en `1` para que la inicialización use **`PyConfig.module_search_paths`** para inicializar **`sys.path`**. De lo contrario, la inicialización volverá a calcular la ruta y reemplazará los valores agregados a `module_search_paths`.
- La salida de la opción **`--help`** ahora cabe en 50 líneas/80 columnas. La información sobre las opciones Python environment variables y **`-X`** ahora está disponible utilizando los indicadores respectivos **`--help-env`** y **`--help-xoptions`**, y con el nuevo **`--help-all`**. (Contribución de Éric Araujo en bpo-46142 [https://bugs.python.org/issue?@action=redirect&bpo=46142].)

- Converting between `int` and `str` in bases other than 2 (binary), 4, 8 (octal), 16 (hexadecimal), or 32 such as base 10 (decimal) now raises a `ValueError` if the number of digits in string form is above a limit to avoid potential denial of service attacks due to the algorithmic complexity. This is a mitigation for **CVE 2020-10735** [https://www.cve.org/CVERecord?id=CVE-2020-10735]. This limit can be configured or disabled by environment variable, command line flag, or `sys` APIs. See the integer string conversion length limitation documentation. The default limit is 4300 digits in string form.

Nuevos Módulos

- `tomllib`: para analizar TOML [https://toml.io/]. Ver **PEP 680** [https://peps.python.org/pep-0680/] para más detalles. (Aportado por Taneli Hukkinen en bpo-40059 [https://bugs.python.org/issue?@action=redirect&bpo=40059].)
- `wsgiref.types`: tipos específicos de **WSGI** [https://peps.python.org/pep-3333/] para la comprobación de tipos estáticos. (Aportado por Sebastian Rittau en bpo-42012 [https://bugs.python.org/issue?@action=redirect&bpo=42012].)

Módulos mejorados

asíncio

- Se agregó la clase `TaskGroup`, un asynchronous context manager que contiene un grupo de tareas que las esperará a todas al salir. Para código nuevo, se recomienda usar `create_task()` y `gather()` directamente. (Aportado por Yury Selivanov y otros en gh-90908 [https://github.com/python/cpython/issues/90908].)
- Se agregó `timeout()`, un administrador de contexto asíncrono para establecer un tiempo de espera en operaciones asíncronas. Para código nuevo, se recomienda usar `wait_for()` directamente. (Aportado por Andrew Svetlov en gh-90927 [https://github.com/python/cpython/issues/90927].)
- Se agregó la clase `Runner`, que expone la maquinaria utilizada por `run()`. (Aportado por Andrew Svetlov en

gh-91218 [https://github.com/python/cpython/issues/91218].)

- Se agregó la clase `Barrier` a las primitivas de sincronización en la biblioteca asyncio y la excepción `BrokenBarrierError` relacionada. (Aportado por Yves Duprat y Andrew Svetlov en gh-87518 [https://github.com/python/cpython/issues/87518].)
- Se agregó el argumento de palabra clave *all_errors* a `asyncio.loop.create_connection()` para que se puedan generar varios errores de conexión como `ExceptionGroup`.
- Se agregó el método `asyncio.StreamWriter.start_tls()` para actualizar las conexiones basadas en secuencias existentes a TLS. (Aportado por Ian Good en bpo-34975 [https://bugs.python.org/issue?@action=redirect&bpo=34975].)
- Se agregaron funciones de socket de datagrama sin formato al bucle de eventos: `sock_sendto()`, `sock_recvfrom()` y `sock_recvfrom_into()`. Estos tienen implementaciones en `SelectorEventLoop` y `ProactorEventLoop`. (Aportado por Alex Grönholm en bpo-46805 [https://bugs.python.org/issue?@action=redirect&bpo=46805].)
- Se agregaron los métodos `cancelling()` y `uncancel()` a `Task`. Estos están destinados principalmente para uso interno, en particular por `TaskGroup`.

contextlib

- Se agregó un administrador de contexto `chdir()` no seguro en paralelo para cambiar el directorio de trabajo actual y luego restaurarlo al salir. Envoltorio simple alrededor de `chdir()`. (Aportación de Filipe Laíns en bpo-25625 [https://bugs.python.org/issue?@action=redirect&bpo=25625])

clases de datos

- Cambie la verificación de mutabilidad predeterminada del campo, permitiendo solo los valores predeterminados que son hashable en lugar de cualquier objeto que no sea una instancia de `dict`, `list` o `set`. (Aportado por Eric V. Smith en bpo-44674 [https://bugs.python.org/issue?

@action=redirect&bpo=44674].)

fecha y hora

- Agregue `datetime.UTC`, un alias conveniente para `datetime.timezone.utc`. (Aportado por Kabir Kwatra en gh-91973 [https://github.com/python/cpython/issues/91973].)
- `datetime.date.fromisoformat()`, `datetime.time.fromisoformat()` y `datetime.datetime.fromisoformat()` ahora se pueden usar para analizar la mayoría de los formatos ISO 8601 (excepto aquellos que admiten fracciones de horas y minutos). (Aportado por Paul Ganssle en gh-80010 [https://github.com/python/cpython/issues/80010].)

enumeración

- Cambió el nombre de **`EnumMeta`** a `EnumType` (**`EnumMeta`** se mantuvo como un alias).
- Se agregó `StrEnum`, con miembros que se pueden usar como (y deben ser) cadenas.
- Se agregó `ReprEnum`, que solo modifica el `__repr__()` de los miembros mientras devuelve sus valores literales (en lugar de nombres) para `__str__()` y `__format__()` (utilizados por `str()`, `format()` y f-strings).
- Changed `Enum.__format__()` (the default for `format()`, `str.format()` and f-strings) to always produce the same result as `Enum.__str__()`: for enums inheriting from `ReprEnum` it will be the member's value; for all other enums it will be the enum and member name (e.g. `Color.RED`).
- Se agregó un nuevo parámetro de clase *boundary* a las enumeraciones `Flag` y la enumeración `FlagBoundary` con sus opciones, para controlar cómo manejar los valores de marca fuera de rango.
- Se agregó el decorador de enumeración `verify()` y la enumeración `EnumCheck` con sus opciones, para verificar las clases de enumeración contra varias restricciones específicas.
- Se agregaron los decoradores `member()` y `nonmember()` para garantizar que el objeto decorado no se convierta en un miembro de enumeración.

- Se agregó el decorador `property()`, que funciona como `property()` excepto para las enumeraciones. Use esto en lugar de `types.DynamicClassAttribute()`.
- Se agregó el decorador de enumeración `global_enum()`, que ajusta `__repr__()` y `__str__()` para mostrar valores como miembros de su módulo en lugar de la clase de enumeración. Por ejemplo, `'re.ASCII'` para el miembro `ASCII` de `re.RegexFlag` en lugar de `'RegexFlag.ASCII'`.
- `Flag` mejorado para admitir `len()`, iteración y `in`/`not in` en sus miembros. Por ejemplo, ahora funciona lo siguiente: `len(AFlag(3)) == 2 and list(AFlag(3)) == (AFlag.ONE, AFlag.TWO)`
- Se cambiaron `Enum` y `Flag` para que los miembros ahora se definan antes de llamar a `__init_subclass__()`; `dir()` ahora incluye métodos, etc., de tipos de datos combinados.
- Se modificó `Flag` para considerar solo los valores primarios (potencia de dos) canónicos, mientras que los valores compuestos (`3`, `6`, `10`, etc.) se consideran alias; las banderas invertidas son forzadas a su equivalente positivo.

fcntl

- En FreeBSD, se admiten los indicadores **`F_DUP2FD`** y **`F_DUP2FD_CLOEXEC`** respectivamente; el primero equivale al uso de `dup2`, mientras que el segundo establece además el indicador `FD_CLOEXEC`.

fracciones

- Compatibilidad con la inicialización al estilo **PEP 515** [https://peps.python.org/pep-0515/] de `Fraction` desde una cadena. (Aportado por Sergey B Kirpichev en bpo-44258 [https://bugs.python.org/issue?@action=redirect&bpo=44258].)
- `Fraction` ahora implementa un método `__int__`, de modo que pasa una verificación `isinstance(some_fraction, typing.SupportsInt)`. (Contribuido por Mark Dickinson en bpo-44547 [https://bugs.python.org/issue?@action=redirect&bpo=44547].)

herramientas funcionales

- `functools.singledispatch()` now supports `types.UnionType` and `typing.Union` as annotations to the dispatch argument.:

```
>>> from functools import singledispatch
>>> @singledispatch
... def fun(arg, verbose=False):
...     if verbose:
...         print("Let me just say,", end=" ")
...     print(arg)
...
>>> @fun.register
... def _(arg: int | float, verbose=False):
...     if verbose:
...         print("Strength in numbers, eh?", end="
...     print(arg)
...
>>> from typing import Union
>>> @fun.register
... def _(arg: Union[list, set], verbose=False):
...     if verbose:
...         print("Enumerate this:")
...     for i, elem in enumerate(arg):
...         print(i, elem)
...
```

(Aportado por Yurii Karabas en bpo-46014 [https://bugs.python.org/issue?@action=redirect&bpo=46014].)

gzip

- The `gzip.compress()` function is now faster when used with the **mtime=0** argument as it delegates the compression entirely to a single `zlib.compress()` operation. There is one side effect of this change: The gzip file header contains an «OS» byte in its header. That was traditionally always set to a value of 255 representing «unknown» by the `gzip` module.

Now, when using `compress()` with **mtime**=**0**, it may be set to a different value by the underlying zlib C library Python was linked against. (See gh-112346 [https://github.com/python/cpython/issues/112346] for details on the side effect.)

hashlib

- `hashlib.blake2b()` y `hashlib.blake2s()` ahora prefieren libb2 [https://www.blake2.net/] a la copia proporcionada por Python. (Aportado por Christian Heimes en bpo-47095 [https://bugs.python.org/issue?@action=redirect&bpo=47095].)
- El módulo interno `_sha3` con algoritmos SHA3 y SHAKE ahora usa *tiny_sha3* en lugar de *Keccak Code Package* para reducir el código y el tamaño binario. El módulo `hashlib` prefiere implementaciones SHA3 y SHAKE optimizadas de OpenSSL. El cambio afecta solo a las instalaciones sin compatibilidad con OpenSSL. (Aportado por Christian Heimes en bpo-47098 [https://bugs.python.org/issue?@action=redirect&bpo=47098].)
- Agregue `hashlib.file_digest()`, una función de ayuda para el hash eficiente de archivos u objetos similares a archivos. (Aportado por Christian Heimes en gh-89313 [https://github.com/python/cpython/issues/89313].)

IDLE y libre de inactividad

- Aplicar resaltado de sintaxis a archivos `.pyi`. (Aportado por Alex Waygood y Terry Jan Reedy en bpo-45447 [https://bugs.python.org/issue?@action=redirect&bpo=45447]).
- Incluya avisos al guardar Shell con entradas y salidas. (Aportado por Terry Jan Reedy en gh-95191 [https://github.com/python/cpython/issues/95191].)

inspeccionar

- Agregue `getmembers_static()` para devolver todos los miembros sin activar la búsqueda dinámica a través del protocolo descriptor. (Contribuido por Weipeng Hong en bpo-30533 [https://bugs.python.org/issue?

@action=redirect&bpo=30533].)

- Agregue **`ismethodwrapper()`** para verificar si el tipo de un objeto es **`MethodWrapperType`**. (Aportado por Hakan Çelik en bpo-29418 [https://bugs.python.org/issue?@action=redirect&bpo=29418].)

- Cambie las funciones relacionadas con el marco en el módulo **`inspect`** para devolver nuevas instancias de clase **`FrameInfo`** y **`Traceback`** (compatibles con las interfaces similares a named tuple anteriores) que incluyen la información de posición **PEP 657** [https://peps.python.org/pep-0657/] extendida (número de línea final, columna y columna final). Las funciones afectadas son:

  - ○ **`inspect.getframeinfo()`**
  - ○ **`inspect.getouterframes()`**
  - ○ **`inspect.getinnerframes()`**,
  - ○ **`inspect.stack()`**
  - ○ **`inspect.trace()`**

  (Aportado por Pablo Galindo en gh-88116 [https://github.com/python/cpython/issues/88116].)

lugar

- Agregue **`locale.getencoding()`** para obtener la codificación de configuración regional actual. Es similar a `locale.getpreferredencoding(False)` pero ignora Python UTF-8 Mode.

Inicio sesión

- Se agregó **`getLevelNamesMapping()`** para devolver una asignación de nombres de nivel de registro (por ejemplo, `'CRITICAL'`) a los valores de su Niveles de logging correspondiente (por ejemplo, `50`, de forma predeterminada). (Aportado por Andrei Kulakovin en gh-88024 [https://github.com/python/cpython/issues/88024].)
- Se agregó un método **`createSocket()`** a **`SysLogHandler`**

para que coincida con **`SocketHandler.createSocket()`**. Se llama automáticamente durante la inicialización del controlador y cuando se emite un evento, si no hay un socket activo. (Aportado por Kirill Pinchuk en gh-88457 [https://github.com/python/cpython/issues/88457].)

Matemáticas

- Suma **`math.exp2()`**: devuelve 2 elevado a la potencia de x. (Aportado por Gideon Mitchell en bpo-45917 [https://bugs.python.org/issue?@action=redirect&bpo=45917].)
- Agregue **`math.cbrt()`**: devuelva la raíz cúbica de x. (Aportado por Ajith Ramachandran en bpo-44357 [https://bugs.python.org/issue?@action=redirect&bpo=44357].)
- Se cambió el comportamiento de dos casos de esquina **`math.pow()`**, para mantener la coherencia con la especificación IEEE 754. Las operaciones `math.pow(0.0, -math.inf)` y `math.pow(-0.0, -math.inf)` ahora devuelven `inf`. Anteriormente plantearon **`ValueError`**. (Contribuido por Mark Dickinson en bpo-44339 [https://bugs.python.org/issue?@action=redirect&bpo=44339].)
- El valor **`math.nan`** ahora está siempre disponible. (Aportado por Victor Stinner en bpo-46917 [https://bugs.python.org/issue?@action=redirect&bpo=46917].)

operador

- Se ha añadido una nueva función `operator.call`, de forma que `operator.call(obj, *args, **kwargs) == obj(*args, **kwargs)`. (Aportado por Antony Lee en bpo-44019 [https://bugs.python.org/issue?@action=redirect&bpo=44019].)

sistema operativo

- En Windows, **`os.urandom()`** ahora usa `BCryptGenRandom()`, en lugar de `CryptGenRandom()`, que está en desuso. (Aportado por Donghee Na en bpo-44611 [https://bugs.python.org/issue?@action=redirect&bpo=44611].)

rutalib

- **`glob()`** y **`rglob()`** solo devuelven directorios si *pattern* termina con un separador de componentes de nombre de ruta: **`sep`** o **`altsep`**. (Aportado por Eisuke Kawasima en bpo-22276 [https://bugs.python.org/issue?@action=redirect&bpo=22276] y bpo-33392 [https://bugs.python.org/issue?@action=redirect&bpo=33392]).

re

- La agrupación atómica (`(?>...)`) y los cuantificadores posesivos (`*+`, `++`, `?+`, `{m,n}+`) ahora son compatibles con las expresiones regulares. (Aportado por Jeffrey C. Jacobs y Serhiy Storchaka en bpo-433030 [https://bugs.python.org/issue?@action=redirect&bpo=433030]).

cerrar

- Agregue el parámetro opcional *dir_fd* en **`shutil.rmtree()`**. (Aportado por Serhiy Storchaka en bpo-46245 [https://bugs.python.org/issue?@action=redirect&bpo=46245].)

enchufe

- Agregue compatibilidad con CAN Socket para NetBSD. (Aportado por Thomas Klausner en bpo-30512 [https://bugs.python.org/issue?@action=redirect&bpo=30512].)
- **`create_connection()`** tiene una opción para generar, en caso de falla de conexión, un **`ExceptionGroup`** que contiene todos los errores en lugar de generar solo el último error. (Aportado por Irit Katriel en bpo-29980 [https://bugs.python.org/issue?@action=redirect&bpo=29980].)

sqlite3

- Ahora puede deshabilitar el autorizador pasando **`None`** a **`set_authorizer()`**. (Aportado por Erlend E. Aasland en bpo-44491 [https://bugs.python.org/issue?

@action=redirect&bpo=44491].)

- El nombre de intercalación `create_collation()` ahora puede contener cualquier carácter Unicode. Los nombres de intercalación con caracteres no válidos ahora generan `UnicodeEncodeError` en lugar de `sqlite3.ProgrammingError`. (Aportado por Erlend E. Aasland en bpo-44688 [https://bugs.python.org/issue?@action=redirect&bpo=44688].)
- Las excepciones `sqlite3` ahora incluyen el código de error extendido de SQLite como `sqlite_errorcode` y el nombre de error de SQLite como `sqlite_errorname`. (Aportado por Aviv Palivoda, Daniel Shahaf y Erlend E. Aasland en bpo-16379 [https://bugs.python.org/issue?@action=redirect&bpo=16379] y bpo-24139 [https://bugs.python.org/issue?@action=redirect&bpo=24139]).
- Agregue `setlimit()` y `getlimit()` a `sqlite3.Connection` para configurar y obtener límites de SQLite por conexión. (Aportado por Erlend E. Aasland en bpo-45243 [https://bugs.python.org/issue?@action=redirect&bpo=45243].)
- `sqlite3` ahora establece `sqlite3.threadsafety` en función del modo de subprocesamiento predeterminado con el que se ha compilado la biblioteca SQLite subyacente. (Aportado por Erlend E. Aasland en bpo-45613 [https://bugs.python.org/issue?@action=redirect&bpo=45613].)
- Las devoluciones de llamada de `sqlite3` C ahora usan excepciones que no se pueden generar si las devoluciones de llamada están habilitadas. Los usuarios ahora pueden registrar un `unraisable hook handler` para mejorar su experiencia de depuración. (Aportado por Erlend E. Aasland en bpo-45828 [https://bugs.python.org/issue?@action=redirect&bpo=45828].)
- Recuperar a través de reversión ya no genera `InterfaceError`. En cambio, dejamos que la biblioteca SQLite maneje estos casos. (Aportado por Erlend E. Aasland en bpo-44092 [https://bugs.python.org/issue?@action=redirect&bpo=44092].)
- Agregue `serialize()` y `deserialize()` a `sqlite3.Connection` para serializar y deserializar bases de datos. (Aportado por Erlend E. Aasland en bpo-41930 [https://

bugs.python.org/issue?@action=redirect&bpo=41930].)

- Agregue `create_window_function()` a `sqlite3.Connection` para crear funciones de ventana agregadas. (Aportado por Erlend E. Aasland en bpo-34916 [https://bugs.python.org/issue?@action=redirect&bpo=34916].)
- Agregue `blobopen()` a `sqlite3.Connection`. `sqlite3.Blob` permite operaciones de E/S incrementales en blobs. (Contribuido por Aviv Palivoda y Erlend E. Aasland en bpo-24905 [https://bugs.python.org/issue?@action=redirect&bpo=24905]).

cuerda

- Agregue `get_identifiers()` y `is_valid()` a `string.Template`, que devuelven respectivamente todos los marcadores de posición válidos y si hay algún marcador de posición no válido presente. (Aportado por Ben Kehoe en gh-90465 [https://github.com/python/cpython/issues/90465].)

sistema

- `sys.exc_info()` ahora deriva los campos `type` y `traceback` de `value` (la instancia de excepción), de modo que cuando se modifica una excepción mientras se está manipulando, los cambios se reflejan en los resultados de llamadas posteriores a `exc_info()`. (Aportado por Irit Katriel en bpo-45711 [https://bugs.python.org/issue?@action=redirect&bpo=45711].)
- Agregue `sys.exception()` que devuelve la instancia de excepción activa (equivalente a `sys.exc_info()[1]`). (Aportado por Irit Katriel en bpo-46328 [https://bugs.python.org/issue?@action=redirect&bpo=46328].)
- Agregue el indicador `sys.flags.safe_path`. (Aportado por Victor Stinner en gh-57684 [https://github.com/python/cpython/issues/57684].)

configuración del sistema

- Se agregaron tres nuevos installation schemes (*posix_venv*, *nt_venv* y *venv*) y se usan cuando Python crea nuevos entornos virtuales o cuando se ejecuta desde un entorno virtual. Los primeros dos esquemas (*posix_venv* y *nt_venv*) son específicos del sistema operativo para Windows y no Windows, el *venv* es esencialmente un alias para uno de ellos según el sistema operativo en el que se ejecuta Python. Esto es útil para los distribuidores posteriores que modifican **`sysconfig.get_preferred_scheme()`**. El código de terceros que crea nuevos entornos virtuales debe usar el nuevo esquema de instalación *venv* para determinar las rutas, al igual que **`venv`**. (Aportado por Miro Hrončok en bpo-45413 [https://bugs.python.org/issue?@action=redirect&bpo=45413].)

archivo temporal

- Los objetos **`SpooledTemporaryFile`** ahora implementan completamente los métodos de **`io.BufferedIOBase`** o **`io.TextIOBase`** (según el modo de archivo). Esto les permite trabajar correctamente con API que esperan objetos similares a archivos, como módulos de compresión. (Aportado por Carey Metcalfe en gh-70363 [https://github.com/python/cpython/issues/70363].)

enhebrar

- En Unix, si la función `sem_clockwait()` está disponible en la biblioteca C (glibc 2.30 y posteriores), el método **`threading.Lock.acquire()`** ahora usa el reloj monótono (**`time.CLOCK_MONOTONIC`**) para el tiempo de espera, en lugar de usar el reloj del sistema (**`time.CLOCK_REALTIME`**), para no verse afectado por cambios de reloj del sistema. (Contribución de Victor Stinner en bpo-41710 [https://bugs.python.org/issue?@action=redirect&bpo=41710].)

tiempo

- En Unix, **`time.sleep()`** ahora usa la función `clock_nanosleep()` o `nanosleep()`, si está disponible, que tiene una resolución de 1 nanosegundo (10^{-9} segundos), en lugar de usar `select()` que tiene una resolución de 1 microsegundo (10^{-6} segundos). (Aportado por Benjamin Szőke y Victor Stinner en bpo-21302 [https://bugs.python.org/issue?@action=redirect&bpo=21302].)

- En Windows 8.1 y versiones posteriores, **`time.sleep()`** ahora utiliza un temporizador de espera basado en high-resolution timers [https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/high-resolution-timers] que tiene una resolución de 100 nanosegundos (10^{-7} segundos). Anteriormente, tenía una resolución de 1 milisegundo (10^{-3} segundos). (Contribución de Benjamin Szőke, Donghee Na, Eryk Sun y Victor Stinner en bpo-21302 [https://bugs.python.org/issue?@action=redirect&bpo=21302] y bpo-45429 [https://bugs.python.org/issue?@action=redirect&bpo=45429]).

tkinter

- Se agregó el método `info_patchlevel()` que devuelve la versión exacta de la biblioteca Tcl como una tupla con nombre similar a **`sys.version_info`**. (Aportado por Serhiy Storchaka en gh-91827 [https://github.com/python/cpython/issues/91827].)

rastrear

- Agregue **`traceback.StackSummary.format_frame_summary()`** para permitir que los usuarios anulen qué marcos aparecen en el rastreo y cómo están formateados. (Aportado por Ammar Askar en bpo-44569 [https://bugs.python.org/issue?@action=redirect&bpo=44569].)
- Agregue **`traceback.TracebackException.print()`**, que imprime la instancia **`TracebackException`** formateada en un archivo. (Aportado por Irit Katriel en bpo-33809 [https://bugs.python.org/issue?@action=redirect&bpo=33809].)

mecanografía

Para conocer los cambios importantes, consulte Nuevas funciones relacionadas con las sugerencias de tipo.

- Agregue `typing.assert_never()` y `typing.Never`. **`typing.assert_never()`** es útil para pedirle a un verificador de tipos que confirme que no se puede acceder a una línea de código. En tiempo de ejecución, genera un `AssertionError`. (Aportado por Jelle Zijlstra en gh-90633 [https://github.com/python/cpython/issues/90633].)
- Agregue `typing.reveal_type()`. Esto es útil para preguntarle a un verificador de tipos qué tipo ha inferido para una expresión dada. En tiempo de ejecución imprime el tipo del valor recibido. (Aportado por Jelle Zijlstra en gh-90572 [https://github.com/python/cpython/issues/90572].)
- Agregue `typing.assert_type()`. Esto es útil para pedirle a un verificador de tipos que confirme que el tipo que ha inferido para una expresión dada coincide con el tipo dado. En tiempo de ejecución, simplemente devuelve el valor recibido. (Aportado por Jelle Zijlstra en gh-90638 [https://github.com/python/cpython/issues/90638].)
- Los tipos `typing.TypedDict` ahora pueden ser genéricos. (Aportado por Samodya Abeysiriwardane en gh-89026 [https://github.com/python/cpython/issues/89026].)
- Los tipos `NamedTuple` ahora pueden ser genéricos. (Aportado por Serhiy Storchaka en bpo-43923 [https://bugs.python.org/issue?@action=redirect&bpo=43923].)
- Permitir subclases de `typing.Any`. Esto es útil para evitar errores de verificación de tipos relacionados con clases altamente dinámicas, como simulacros. (Aportado por Shantanu Jain en gh-91154 [https://github.com/python/cpython/issues/91154].)
- El decorador `typing.final()` ahora establece el atributo `__final__` en el objeto decorado. (Aportado por Jelle Zijlstra en gh-90500 [https://github.com/python/cpython/issues/90500].)
- La función `typing.get_overloads()` se puede utilizar para la introspección de las sobrecargas de una función. `typing.clear_overloads()` se puede utilizar para borrar todas las sobrecargas registradas de una función. (Aportado por Jelle Zijlstra en gh-89263 [https://github.com/python/cpython/issues/89263].)
- Ahora se conserva el método `__init__()` de las subclases `Protocol`. (Aportado por Adrián García Badarasco en

gh-88970 [https://github.com/python/cpython/issues/88970].)

- La representación de tipos de tuplas vacías (`Tuple[()]`) se simplifica. Esto afecta la introspección, p. `get_args(Tuple[()])` ahora se evalúa como `()` en lugar de `((),)`. (Aportado por Serhiy Storchaka en gh-91137 [https://github.com/python/cpython/issues/91137].)
- Afloje los requisitos de tiempo de ejecución para las anotaciones de tipo eliminando la verificación invocable en la función privada `typing._type_check`. (Aportado por Gregory Beauregard en gh-90802 [https://github.com/python/cpython/issues/90802].)
- **`typing.get_type_hints()`** ahora admite la evaluación de cadenas como referencias directas en PEP 585 generic aliases. (Aportado por Niklas Rosenstein en gh-85542 [https://github.com/python/cpython/issues/85542].)
- **`typing.get_type_hints()`** ya no agrega **`Optional`** a los parámetros con `None` como predeterminado. (Aportado por Nikita Sobolev en gh-90353 [https://github.com/python/cpython/issues/90353].)
- **`typing.get_type_hints()`** ahora admite la evaluación de anotaciones **`ClassVar`** con cadenas desnudas. (Aportado por Gregory Beauregard en gh-90711 [https://github.com/python/cpython/issues/90711].)
- **`typing.no_type_check()`** ya no modifica clases y funciones externas. Ahora también marca correctamente los métodos de clase para que no se verifique el tipo. (Aportado por Nikita Sobolev en gh-90729 [https://github.com/python/cpython/issues/90729].)

unicodedata

- La base de datos Unicode se ha actualizado a la versión 14.0.0. (Aportado por Benjamin Peterson en bpo-45190 [https://bugs.python.org/issue?@action=redirect&bpo=45190]).

prueba de unidad

- Se agregaron los métodos **`enterContext()`** y **`enterClassContext()`** de la clase **`TestCase`**, el método **`enterAsyncContext()`** de la clase

`IsolatedAsyncioTestCase` y la función **`unittest.enterModuleContext()`**. (Aportado por Serhiy Storchaka en bpo-45046 [https://bugs.python.org/issue?@action=redirect&bpo=45046].)

venv

- Cuando se crean nuevos entornos virtuales de Python, el *venv* sysconfig installation scheme se utiliza para determinar las rutas dentro del entorno. Cuando Python se ejecuta en un entorno virtual, el mismo esquema de instalación es el predeterminado. Eso significa que los distribuidores intermedios pueden cambiar el esquema de instalación de sysconfig predeterminado sin cambiar el comportamiento de los entornos virtuales. El código de terceros que también crea nuevos entornos virtuales debería hacer lo mismo. (Aportado por Miro Hrončok en bpo-45413 [https://bugs.python.org/issue?@action=redirect&bpo=45413].)

advertencias

- **`warnings.catch_warnings()`** ahora acepta argumentos para **`warnings.simplefilter()`**, lo que proporciona una forma más concisa de ignorar localmente las advertencias o convertirlas en errores. (Aportado por Zac Hatfield-Dodds en bpo-47074 [https://bugs.python.org/issue?@action=redirect&bpo=47074]).

archivo zip

- Se agregó compatibilidad para especificar la codificación de nombres de miembros para leer metadatos en los encabezados de archivos y directorios de **`ZipFile`**. (Aportado por Stephen J. Turnbull y Serhiy Storchaka en bpo-28080 [https://bugs.python.org/issue?@action=redirect&bpo=28080]).
- Se agregó **`ZipFile.mkdir()`** para crear nuevos directorios dentro de archivos ZIP. (Aportado por Sam Ezeh en gh-49083 [https://github.com/python/cpython/issues/49083].)
- Se agregaron **`stem`**, **`suffix`** y **`suffixes`** a **`zipfile.Path`**. (Aportado por Miguel Brito en gh-88261

[https://github.com/python/cpython/issues/88261].)

Optimizaciones

Esta sección cubre optimizaciones específicas independientes del proyecto CPython más rápido, que se trata en su propia sección.

- El compilador ahora optimiza printf-style % formatting simple en cadenas literales que contienen solo los códigos de formato `%s`, `%r` y `%a` y lo hace tan rápido como una expresión f-string correspondiente. (Aportado por Serhiy Storchaka en bpo-28307 [https://bugs.python.org/issue?@action=redirect&bpo=28307].)
- La división de enteros a la baja (`//`) está mejor ajustada para la optimización por parte de los compiladores. Ahora es un 20 % más rápido en x86-64 cuando se divide un **`int`** por un valor menor que `2**30`. (Aportado por Gregory P. Smith y Tim Peters en gh-90564 [https://github.com/python/cpython/issues/90564].)
- **`sum()`** ahora es casi un 30 % más rápido para números enteros más pequeños que `2**30`. (Aportado por Stefan Behnel en gh-68264 [https://github.com/python/cpython/issues/68264].)
- El cambio de tamaño de las listas está simplificado para el caso común, acelerando **`list.append()`** en ≈15 % y list comprehensions simples hasta en un 20-30 % (Contribuido por Dennis Sweeney en gh-91165 [https://github.com/python/cpython/issues/91165]).
- Los diccionarios no almacenan valores hash cuando todas las claves son objetos Unicode, lo que reduce el tamaño de **`dict`**. Por ejemplo, `sys.getsizeof(dict.fromkeys("abcdefg"))` se reduce de 352 bytes a 272 bytes (un 23 % más pequeño) en plataformas de 64 bits. (Aportado por Inada Naoki en bpo-46845 [https://bugs.python.org/issue?@action=redirect&bpo=46845].)
- El uso de **`asyncio.DatagramProtocol`** ahora es mucho más rápido cuando se transfieren archivos grandes a través de UDP, con velocidades 100 veces más altas para un archivo de ≈60 MiB. (Aportado por msoxzw en gh-91487 [https://

github.com/python/cpython/issues/91487].)

- Las funciones `math` `comb()` y `perm()` ahora son ≈10 veces más rápidas para argumentos grandes (con una aceleración mayor para *k* más grandes). (Aportado por Serhiy Storchaka en bpo-37295 [https://bugs.python.org/issue?@action=redirect&bpo=37295].)
- Las funciones `statistics` `mean()`, `variance()` y `stdev()` ahora consumen iteradores en una sola pasada en lugar de convertirlos primero en `list`. Esto es el doble de rápido y puede ahorrar una cantidad considerable de memoria. (Aportado por Raymond Hettinger en gh-90415 [https://github.com/python/cpython/issues/90415].)
- `unicodedata.normalize()` ahora normaliza cadenas ASCII puras en tiempo constante. (Contribución de Donghee Na en bpo-44987 [https://bugs.python.org/issue?@action=redirect&bpo=44987].)

CPython más rápido

CPython 3.11 es en promedio 25% faster [https://github.com/faster-cpython/ideas#published-results] que CPython 3.10 cuando se mide con el conjunto de pruebas comparativas pyperformance [https://github.com/python/pyperformance] y se compila con GCC en Ubuntu Linux. Dependiendo de su carga de trabajo, la aceleración podría ser hasta un 10-60% más rápida.

Este proyecto se centra en dos áreas principales de Python: Inicio más rápido y Tiempo de ejecución más rápido. Las optimizaciones no cubiertas por este proyecto se enumeran por separado en Optimizaciones.

Inicio más rápido

Importaciones congeladas / Objetos de código estático

Python almacena en caché el bytecode en el directorio __pycache__ para acelerar la carga del módulo.

Previamente en 3.10, la ejecución del módulo de Python se veía así:

```
Read __pycache__ -> Unmarshal -> Heap allocated code obj
```

En Python 3.11, los módulos principales esenciales para el inicio de Python están «congelados». Esto significa que el intérprete asigna estáticamente sus Objetos código (y sus códigos de bytes). Esto reduce los pasos en el proceso de ejecución del módulo a:

```
Statically allocated code object -> Evaluate
```

El inicio del intérprete ahora es un 10-15 % más rápido en Python 3.11. Esto tiene un gran impacto para los programas de ejecución corta que usan Python.

(Aportado por Eric Snow, Guido van Rossum y Kumar Aditya en numerosos asuntos).

Tiempo de ejecución más rápido

Marcos de Python más baratos y perezosos

Los marcos de Python, que contienen información de ejecución, se crean cada vez que Python llama a una función de Python. Las siguientes son nuevas optimizaciones de marcos:

- Simplificó el proceso de creación de marcos.
- Se evitó la asignación de memoria al reutilizar generosamente el espacio de marcos en la pila C.
- Simplificó la estructura del marco interno para que contenga solo información esencial. Los marcos contenían previamente información adicional de gestión de memoria y depuración.

Los frame objects de estilo antiguo ahora se crean solo cuando lo solicitan los depuradores o las funciones de introspección de Python como `sys._getframe()` y `inspect.currentframe()`. Para la mayoría del código de usuario, no se crea ningún objeto de marco. Como resultado, casi todas las llamadas a funciones de Python se han acelerado significativamente. Medimos una aceleración del 3 al 7% en pyperformance.

(Aportado por Mark Shannon en bpo-44590 [https://bugs.python.org/issue?@action=redirect&bpo=44590].)

Llamadas a funciones de Python en línea

Durante una llamada de función de Python, Python llamará a una función C de evaluación para interpretar el código de esa función. Esto limita efectivamente la recursión pura de Python a lo que es seguro para la pila de C.

En 3.11, cuando CPython detecta código de Python que llama a otra función de Python, configura un nuevo marco y «salta» al nuevo código dentro del nuevo marco. Esto evita llamar a la función de interpretación de C por completo.

La mayoría de las llamadas a funciones de Python ahora no consumen espacio en la pila de C, lo que las acelera. En funciones recursivas simples como Fibonacci o factorial, observamos una aceleración de 1,7 veces. Esto también significa que las funciones recursivas pueden recurrir significativamente más profundamente (si el usuario aumenta el límite de recursividad con `sys.setrecursionlimit()`). Medimos una mejora del 1 al 3% en pyperformance.

(Aportado por Pablo Galindo y Mark Shannon en bpo-45256 [https://bugs.python.org/issue?@action=redirect&bpo=45256].)

PEP 659: Intérprete Adaptativo Especializado

PEP 659 [https://peps.python.org/pep-0659/] es una de las partes clave del proyecto Faster CPython. La idea general es que, si bien Python es un lenguaje dinámico, la mayor parte del código tiene regiones donde los objetos y tipos rara vez cambian. Este concepto se conoce como *type stability*.

En tiempo de ejecución, Python intentará buscar patrones comunes y estabilidad de tipos en el código en ejecución. Luego, Python reemplazará la operación actual por una más especializada. Esta operación especializada utiliza rutas rápidas disponibles solo para aquellos casos/tipos de uso, que generalmente superan a sus contrapartes genéricas. Esto también trae consigo otro concepto llamado *inline caching*, donde Python almacena en caché los resultados de operaciones costosas directamente en el bytecode.

El especialista también combinará ciertos pares de instrucciones comunes en una superinstrucción, reduciendo la sobrecarga durante la ejecución.

Python solo se especializará cuando vea código «caliente» (ejecutado varias veces). Esto evita que Python pierda tiempo en código de ejecución única. Python también puede desespecializarse cuando el código es demasiado dinámico o cuando cambia el uso. La especialización se intenta periódicamente y los intentos de especialización no son demasiado costosos, lo que permite que la especialización se adapte a nuevas circunstancias.

(PEP escrito por Mark Shannon, con ideas inspiradas por Stefan Brunthaler. Consulte **PEP 659** [https://peps.python.org/pep-0659/] para obtener más información. Implementación por Mark Shannon y Brandt Bucher, con ayuda adicional de Irit Katriel y Dennis Sweeney).

Operación	Forma	Especialización	Aceleración de la operación (hasta)	Colaborador(es)

Operaciones binarias	`x + x`
	`x - x`
	`x * x`
		Las sumas, multiplicaciones y restas binarias para tipos comunes como `int`, `float` y `str` toman rutas rápidas personalizadas para sus tipos subyacentes.	10%	Mark Shannon, Donghee Na, Brandt Bucher, Dennis Sweeney

Subíndice	`a[i]`	Los tipos de contenedores de subíndices, como `list`, `tuple` y `dict`, indexan directamente las estructuras de datos subyacentes. El subíndice `__getitem__()` personalizado también está incluido de manera similar a Llamadas a funciones de Python en línea.	10-25%	Irit Katriel, Mark Shannon
Almacenar subíndice	`a[i] = z`	Similar a la especialización de subíndices anterior.	10-25%	Dennis Sweeney
Llamadas	`f(arg)` `C(arg)`	Llamadas a funciones y tipos integrados (C) comunes, como `len()` y `str`, llaman directamente a su versión C subyacente. Esto evita pasar por la convención de llamadas interna.	20%	Mark Shannon, Ken Jin
Cargar variable global	`print` `len`	El índice del objeto en el espacio de nombres globales/integrados se almacena en caché. La carga de globales e integrados requiere cero búsquedas de espacios de nombres.	[1]	Mark Shannon
Cargar atributo	`o.attr`	Similar a cargar variables globales. El índice del atributo dentro del espacio de nombres de la clase/objeto se almacena en caché. En la mayoría de los casos, la carga de atributos requerirá cero búsquedas de espacios de nombres.	[2]	Mark Shannon
Cargar métodos para llamadas	`o.meth()`	La dirección real del método se almacena en caché. La carga de métodos ahora no tiene búsquedas de espacio de nombres,	10-20%	Ken Jin, Mark Shannon

incluso para clases con largas cadenas de herencia.
[illegible] Similar a la optimización de atributos de carga.
[illegible] Especializado para contenedores comunes como `list` y `tuple`.
Evita la convención de llamadas internas.

Ya existía una optimización similar desde Python 3.8. 3.11 se especializa en más formas y reduce algunos gastos generales.

Ya existía una optimización similar desde Python 3.10. 3.11 se especializa en más formularios. Además, bpo-45947 [https://bugs.python.org/issue?@action=redirect&bpo=45947] debería acelerar todas las cargas de atributos.

Varios

- Los objetos ahora requieren menos memoria debido a los espacios de nombres de objetos creados con pereza. Sus diccionarios de espacio de nombres ahora también comparten claves más libremente. (Contribuido por Mark Shannon en bpo-45340 [https://bugs.python.org/issue?@action=redirect&bpo=45340] y bpo-40116 [https://bugs.python.org/issue?@action=redirect&bpo=40116]).
- Se implementan excepciones de «costo cero», lo que elimina el costo de las declaraciones `try` cuando no se plantea ninguna excepción. (Contribución de Mark Shannon en bpo-40222 [https://bugs.python.org/issue?@action=redirect&bpo=40222].)
- Una representación más concisa de las excepciones en el intérprete redujo el tiempo necesario para detectar una excepción en aproximadamente un 10 %. (Aportado por Irit Katriel en bpo-45711 [https://bugs.python.org/issue?@action=redirect&bpo=45711].)
- El motor de coincidencia de expresiones regulares de `re` ha sido parcialmente refactorizado y ahora utiliza gotos calculados (o «código de subprocesos») en las plataformas compatibles. Como resultado, Python 3.11 ejecuta pyperformance regular expression benchmarks [https://pyperformance.readthedocs.io/benchmarks.html#regex-dna] hasta un 10% más rápido que Python 3.10. (Contribución de Brandt Bucher en gh-91404 [https://github.com/python/cpython/

issues/91404].)

Preguntas más frecuentes

¿Cómo debo escribir mi código para utilizar estas aceleraciones?

No tienes que cambiar tu código, escribe código Pythonic que siga las mejores prácticas comunes. El proyecto Faster CPython optimiza los patrones de código comunes que observamos.

¿CPython 3.11 usará más memoria?

Tal vez no. No esperamos que el uso de memoria supere el 20% más que 3.10. Esto se compensa con las optimizaciones de memoria para objetos de marco y diccionarios de objetos, como se mencionó anteriormente.

No veo ninguna aceleración en mi carga de trabajo. ¿Por qué?

Cierto código no tendrá beneficios notables. Si su código pasa la mayor parte de su tiempo en operaciones de E/S, o ya realiza la mayor parte de su cálculo en una biblioteca de extensión C como numpy, no habrá una aceleración significativa. Actualmente, este proyecto es el que más beneficia a las cargas de trabajo de Python puro.

Además, las cifras de pyperformance son una media geométrica. Incluso dentro de los puntos de referencia de pyrendimiento, ciertos puntos de referencia se han ralentizado ligeramente, mientras que otros se han acelerado casi 2 veces.

¿Existe un compilador JIT?

No. Todavía estamos explorando otras optimizaciones.

Sobre

Faster CPython explora optimizaciones para CPython. El equipo principal está financiado por Microsoft para trabajar en esto a tiempo completo. Pablo Galindo Salgado también está financiado por Bloomberg LP para trabajar en el proyecto a tiempo parcial. Finalmente, muchos contribuyentes son voluntarios de la comunidad.

Cambios en el código de bytes de CPython

El código de bytes ahora contiene entradas de caché en línea, que toman la forma de las instrucciones `CACHE` recién agregadas. Muchos códigos de operación esperan ser seguidos por una cantidad exacta de cachés e indican al intérprete que los omita en tiempo de ejecución. Los cachés poblados pueden parecer instrucciones arbitrarias, por lo que se debe tener mucho cuidado al leer o modificar el código de bytes adaptativo sin procesar que contiene datos acelerados.

Nuevos códigos de operación

- **`ASYNC_GEN_WRAP`**, `RETURN_GENERATOR` and `SEND`, used in generators and co-routines.
- `COPY_FREE_VARS`, que evita la necesidad de un código especial del lado de la persona que llama para los cierres.
- `JUMP_BACKWARD_NO_INTERRUPT`, para usar en ciertos bucles donde no es deseable manejar interrupciones.
- `MAKE_CELL`, para crear Objetos celda.
- `CHECK_EG_MATCH` and **`PREP_RERAISE_STAR`**, to handle the new exception groups and except* added in **PEP 654** [https://peps.python.org/pep-0654/].
- `PUSH_EXC_INFO`, para uso en controladores de excepciones.
- `RESUME`, no operativo, para el seguimiento interno, la depuración y las comprobaciones de optimización.

Códigos de operación reemplazados

Códigos de operación reemplazados	Nuevos códigos de operación	Notas
`INPLACE_*`	`BINARY_OP`	Reemplazó todos los códigos de operación numéricos binarios/en el lugar por un solo código de operación
`CALL_FUNCTION`	`CALL`	Separa[illegible] de argumentos para métodos del manejo de

CALL_FUNCTION_KW **CALL_METHOD**	**KW_NAMES** **PRECALL** **PUSH_NULL**	de palabras clave; permite una mejor especialización de las llamadas
DUP_TOP **DUP_TOP_TWO** **ROT_TWO** **ROT_THREE** **ROT_FOUR** **ROT_N**	**COPY** **SWAP**	Instrucciones de manipulación de pilas
JUMP_IF_NOT_EXC_MATCH	**CHECK_EXC_MATCH**	Ahora realiza la comprobación pero no salta.
JUMP_ABSOLUTE **POP_JUMP_IF_FALSE** **POP_JUMP_IF_TRUE**	**JUMP_BACKWARD** **POP_JUMP_BACKWARD_IF_*** **POP_JUMP_FORWARD_IF_***	Ver [3]; Variantes `TRUE`, `FALSE`, `NONE` y `NOT_NONE` para cada dirección
SETUP_WITH **SETUP_ASYNC_WITH**	**BEFORE_WITH**	Configuración del bloque **with**

All jump opcodes are now relative, including the existing **JUMP_IF_TRUE_OR_POP** and **JUMP_IF_FALSE_OR_POP**. The argument is now an offset from the current instruction rather than an absolute location.

Códigos de operación cambiados/eliminados

- Se cambiaron **MATCH_CLASS** y **MATCH_KEYS** para que ya no envíen un valor booleano adicional para indicar éxito/fracaso. En su lugar, se inserta `None` en caso de error en lugar de la tupla de valores extraídos.
- Se cambiaron los códigos de operación que funcionan con excepciones para reflejarlas y ahora se representan como un elemento en la pila en lugar de tres (ver gh-89874 [https://github.com/python/cpython/issues/89874]).
- Se eliminaron **COPY_DICT_WITHOUT_KEYS**, **GEN_START**, **POP_BLOCK**, **SETUP_FINALLY** y **YIELD_FROM**.

Obsoleto

Esta sección enumera las API de Python que han quedado obsoletas en Python 3.11.

Las API de C en desuso son listed separately.

Idioma/Construidos

- El encadenamiento de descriptores `classmethod` (introducido en bpo-19072 [https://bugs.python.org/issue?@action=redirect&bpo=19072]) ahora está en desuso. Ya no se puede usar para envolver otros descriptores como `property`. El diseño central de esta función tenía fallas y causó una serie de problemas posteriores. Para «transmitir» un `classmethod`, considere usar el atributo `__wrapped__` que se agregó en Python 3.10. (Aportado por Raymond Hettinger en gh-89519 [https://github.com/python/cpython/issues/89519].)
- Los escapes octales en cadenas y bytes literales con valores mayores que `0o377` (255 en decimal) ahora producen un `DeprecationWarning`. En una futura versión de Python, generarán un `SyntaxWarning` y eventualmente un `SyntaxError`. (Aportado por Serhiy Storchaka en gh-81548 [https://github.com/python/cpython/issues/81548].)
- La delegación de `int()` a `__trunc__()` ahora está obsoleta. Llamar a `int(a)` cuando `type(a)` implementa `__trunc__()` pero no `__int__()` o `__index__()` ahora genera un `DeprecationWarning`. (Aportado por Zackery Spytz en bpo-44977 [https://bugs.python.org/issue?@action=redirect&bpo=44977]).

Módulos

- PEP 594 [https://peps.python.org/pep-0594/] condujo a la desaprobación de los siguientes módulos programados para su eliminación en Python 3.13:

`aifc`	`chunk`	`msilib`	`pipes`	`telnetlib`
`audioop`	`crypt`	`nis`	`sndhdr`	`uu`
`cgi`	`imghdr`	`nntplib`	`spwd`	`xdrlib`
`cgitb`	`mailcap`	`ossaudiodev`	`sunau`	

(Aportado por Brett Cannon en bpo-47061 [https://bugs.python.org/issue?@action=redirect&bpo=47061] y Victor Stinner en gh-68966 [https://github.com/python/cpython/issues/68966]).

- The **asynchat**, **asyncore** and **smtpd** modules have been deprecated since at least Python 3.6. Their documentation and deprecation warnings have now been updated to note they will be removed in Python 3.12. (Contributed by Hugo van Kemenade in bpo-47022 [https://bugs.python.org/issue?@action=redirect&bpo=47022].)
- The **lib2to3** package and `2to3` tool are now deprecated and may not be able to parse Python 3.10 or newer. See **PEP 617** [https://peps.python.org/pep-0617/], introducing the new PEG parser, for details. (Contributed by Victor Stinner in bpo-40360 [https://bugs.python.org/issue?@action=redirect&bpo=40360].)
- Los módulos no documentados **sre_compile**, **sre_constants** y **sre_parse** ahora están obsoletos. (Aportado por Serhiy Storchaka en bpo-47152 [https://bugs.python.org/issue?@action=redirect&bpo=47152].)

Biblioteca estándar

- Lo siguiente ha quedado obsoleto en **configparser** desde Python 3.2. Sus advertencias de obsolescencia ahora se han actualizado para tener en cuenta que se eliminarán en Python 3.12:
    - ○ la clase **configparser.SafeConfigParser**
    - ○ la propiedad **configparser.ParsingError.filename**
    - ○ the **configparser.RawConfigParser.readfp()** method

  (Aportado por Hugo van Kemenade en bpo-45173 [https://bugs.python.org/issue?@action=redirect&bpo=45173].)
- **configparser.LegacyInterpolation** ha quedado obsoleto en la cadena de documentación desde Python 3.2 y no aparece en la documentación de **configparser**. Ahora emite un **DeprecationWarning** y se eliminará en Python 3.13. Utilice **configparser.BasicInterpolation** o **configparser.ExtendedInterpolation** en su lugar.

(Aportado por Hugo van Kemenade en bpo-46607 [https://bugs.python.org/issue?@action=redirect&bpo=46607].)

- El conjunto anterior de funciones `importlib.resources` quedó en desuso en favor de los reemplazos agregados en Python 3.9 y se eliminará en una versión futura de Python, debido a que no admite recursos ubicados dentro de los subdirectorios del paquete:
    - `importlib.resources.contents()`
    - `importlib.resources.is_resource()`
    - `importlib.resources.open_binary()`
    - `importlib.resources.open_text()`
    - `importlib.resources.read_binary()`
    - `importlib.resources.read_text()`
    - `importlib.resources.path()`
- The `locale.getdefaultlocale()` function is deprecated and will be removed in Python 3.15. Use `locale.setlocale()`, `locale.getpreferredencoding(False)` and `locale.getlocale()` functions instead. (Contributed by Victor Stinner in gh-90817 [https://github.com/python/cpython/issues/90817].)
- The `locale.resetlocale()` function is deprecated and will be removed in Python 3.13. Use `locale.setlocale(locale.LC_ALL, "")` instead. (Contributed by Victor Stinner in gh-90817 [https://github.com/python/cpython/issues/90817].)
- Ahora se aplicarán reglas más estrictas para las referencias de grupos numéricos y los nombres de grupos en regular expressions. Ahora solo se aceptarán secuencias de dígitos ASCII como referencia numérica, y el nombre del grupo en los patrones `bytes` y las cadenas de reemplazo solo pueden contener letras ASCII, dígitos y guiones bajos. Por ahora, se genera una advertencia de desaprobación para la sintaxis que viola estas reglas. (Aportado por Serhiy Storchaka en gh-91760 [https://github.com/python/cpython/issues/91760].)

- En el módulo `re`, la función `re.template()` y los indicadores `re.TEMPLATE` y `re.T` correspondientes están en desuso, ya que no estaban documentados y carecían de un propósito obvio. Se eliminarán en Python 3.13. (Contribución de Serhiy Storchaka y Miro Hrončok en gh-92728 [https://github.com/python/cpython/issues/92728].)

- `turtle.settiltangle()` has been deprecated since Python 3.1; it now emits a deprecation warning and will be removed in Python 3.13. Use `turtle.tiltangle()` instead (it was earlier incorrectly marked as deprecated, and its docstring is now corrected). (Contributed by Hugo van Kemenade in bpo-45837 [https://bugs.python.org/issue?@action=redirect&bpo=45837].)

- `typing.Text`, que existe únicamente para proporcionar soporte de compatibilidad entre el código de Python 2 y Python 3, ahora está obsoleto. Su eliminación no está planificada actualmente, pero se recomienda a los usuarios que usen `str` en su lugar siempre que sea posible. (Aportado por Alex Waygood en gh-92332 [https://github.com/python/cpython/issues/92332].)

- La sintaxis de argumento de palabra clave para construir tipos `typing.TypedDict` ahora está obsoleta. Se eliminará el soporte en Python 3.13. (Aportado por Jingchen Ye en gh-90224 [https://github.com/python/cpython/issues/90224].)

- `webbrowser.MacOSX` está en desuso y se eliminará en Python 3.13. No está probado, no está documentado y no lo utiliza el propio `webbrowser`. (Contribución de Donghee Na en bpo-42255 [https://bugs.python.org/issue?@action=redirect&bpo=42255].)

- El comportamiento de devolver un valor de los métodos de prueba `TestCase` y `IsolatedAsyncioTestCase` (aparte del valor `None` predeterminado) ahora está obsoleto.

- Se descartaron las siguientes funciones `unittest` no documentadas formalmente, programadas para su eliminación en Python 3.13:

  - ○ `unittest.findTestCases()`
  - ○ `unittest.makeSuite()`
  - ○ `unittest.getTestCaseNames()`

  Utilice métodos `TestLoader` en su lugar:

  - ○ `unittest.TestLoader.loadTestsFromModule()`
  - ○ `unittest.TestLoader.loadTestsFromTestCase()`
  - ○ `unittest.TestLoader.getTestCaseNames()`

  (Aportado por Erlend E. Aasland en bpo-5846 [https://bugs.python.org/issue?@action=redirect&bpo=5846].)

- `unittest.TestProgram.usageExit()` is marked deprecated, to be removed in 3.13. (Contributed by Carlos Damázio in gh-67048 [https://github.com/python/cpython/issues/67048].)

Eliminación pendiente en Python 3.12

Las siguientes API de Python han quedado obsoletas en versiones anteriores de Python y se eliminarán en Python 3.12.

Las API de C pendientes de eliminación son listed separately.

- The **asynchat** module
- The **asyncore** module
- The entire distutils package
- El módulo **imp**
- El espacio de nombres `typing.io`
- El espacio de nombres `typing.re`
- `cgi.log()`
- `importlib.find_loader()`
- `importlib.abc.Loader.module_repr()`

- `importlib.abc.MetaPathFinder.find_module()`
- `importlib.abc.PathEntryFinder.find_loader()`
- `importlib.abc.PathEntryFinder.find_module()`
- `importlib.machinery.BuiltinImporter.find_module()`
- `importlib.machinery.BuiltinLoader.module_repr()`
- `importlib.machinery.FileFinder.find_loader()`
- `importlib.machinery.FileFinder.find_module()`
- `importlib.machinery.FrozenImporter.find_module()`
- `importlib.machinery.FrozenLoader.module_repr()`
- `importlib.machinery.PathFinder.find_module()`
- `importlib.machinery.WindowsRegistryFinder.find_modu`
- `importlib.util.module_for_loader()`
- `importlib.util.set_loader_wrapper()`
- `importlib.util.set_package_wrapper()`
- `pkgutil.ImpImporter`
- `pkgutil.ImpLoader`
- `pathlib.Path.link_to()`
- `sqlite3.enable_shared_cache()`
- `sqlite3.OptimizedUnicode()`
- `PYTHONTHREADDEBUG` environment variable
- Los siguientes alias obsoletos en `unittest`:

Nombre del método	Alias obsoleto	Obsoleto en

`failUnless`	`assertTrue()`	3.1
`failIf`	`assertFalse()`	3.1
`failUnlessEqual`	`assertEqual()`	3.1
`failIfEqual`	`assertNotEqual()`	3.1
`failUnlessAlmostEqual`	`assertAlmostEqual()`	3.1
`failIfAlmostEqual`	`assertNotAlmostEqual()`	3.1
`failUnlessRaises`	`assertRaises()`	3.1
`assert_`	`assertTrue()`	3.2
`assertEquals`	`assertEqual()`	3.2
`assertNotEquals`	`assertNotEqual()`	3.2
`assertAlmostEquals`	`assertAlmostEqual()`	3.2
`assertNotAlmostEquals`	`assertNotAlmostEqual()`	3.2
`assertRegexpMatches`	`assertRegex()`	3.2
`assertRaisesRegexp`	`assertRaisesRegex()`	3.2
`assertNotRegexpMatches`	`assertNotRegex()`	3.5

Remoto

Esta sección enumera las API de Python que se eliminaron en Python 3.11.

Las API C eliminadas son listed separately.

- Se eliminó **`@asyncio.coroutine()`** decorator, lo que permite que las corrutinas basadas en generadores heredados sean compatibles con el código `async`/`await`. La función ha quedado obsoleta desde Python 3.8 y la eliminación se programó inicialmente para Python 3.10. Utilice `async def` en su lugar. (Aportado por Illia Volochii en bpo-43216 [https://bugs.python.org/issue?@action=redirect&bpo=43216].)

- Se eliminó **`asyncio.coroutines.CoroWrapper`** utilizado para envolver objetos de corrutina basados en generadores heredados en el modo de depuración. (Aportado por Illia Volochii en bpo-43216 [https://bugs.python.org/issue?@action=redirect&bpo=43216].)

- Debido a importantes problemas de seguridad, el parámetro *reuse_address* de `asyncio.loop.create_datagram_endpoint()`, deshabilitado en Python 3.9, ahora se eliminó por completo. Esto se debe al comportamiento de la opción de socket `SO_REUSEADDR` en UDP. (Aportado por Hugo van Kemenade en bpo-45129 [https://bugs.python.org/issue?@action=redirect&bpo=45129].)

- Se eliminó el módulo **`binhex`**, obsoleto en Python 3.9. También se eliminaron las funciones **`binascii`** relacionadas y obsoletas de manera similar:

  ○ **`binascii.a2b_hqx()`**
  ○ **`binascii.b2a_hqx()`**
  ○ **`binascii.rlecode_hqx()`**
  ○ **`binascii.rldecode_hqx()`**

  La función **`binascii.crc_hqx()`** permanece disponible.

  (Aportado por Victor Stinner en bpo-45085 [https://bugs.python.org/issue?@action=redirect&bpo=45085].)

- Removed the **`distutils`** `bdist_msi` command deprecated in Python 3.9. Use `bdist_wheel` (wheel packages) instead. (Contributed by Hugo van Kemenade in bpo-45124 [https://bugs.python.org/issue?@action=redirect&bpo=45124].)

- Se eliminaron los métodos **`__getitem__()`** de **`xml.dom.pulldom.DOMEventStream`**, **`wsgiref.util.FileWrapper`** y **`fileinput.FileInput`**, obsoletos desde Python 3.9. (Aportado por Hugo van Kemenade en bpo-45132 [https://bugs.python.org/issue?@action=redirect&bpo=45132].)

- Se eliminaron las funciones **`gettext`** obsoletas **`lgettext()`**, **`ldgettext()`**, **`lngettext()`** y **`ldngettext()`**. También se eliminó la función **`bind_textdomain_codeset()`**, los métodos **`NullTranslations.output_charset()`** y **`NullTranslations.set_output_charset()`**, y el parámetro *codeset* de **`translation()`** y **`install()`**, ya que solo se usan para las funciones **`l*gettext()`**. (Contribución de Donghee Na y Serhiy Storchaka en bpo-44235 [https://bugs.python.org/issue?@action=redirect&bpo=44235]).

- Eliminado del módulo `inspect`:
    - La función `getargspec()`, en desuso desde Python 3.0; utilice `inspect.signature()` o `inspect.getfullargspec()` en su lugar.
    - La función `formatargspec()`, obsoleta desde Python 3.5; use la función `inspect.signature()` o el objeto `inspect.Signature` directamente.
    - Los métodos `Signature.from_builtin()` y `Signature.from_function()` no documentados, obsoletos desde Python 3.5; utilice el método `Signature.from_callable()` en su lugar.

  (Aportado por Hugo van Kemenade en bpo-45320 [https://bugs.python.org/issue?@action=redirect&bpo=45320].)

- Se eliminó el método `__class_getitem__()` de `pathlib.PurePath`, porque no se usó y se agregó por error en versiones anteriores. (Aportado por Nikita Sobolev en bpo-46483 [https://bugs.python.org/issue?@action=redirect&bpo=46483].)

- Removed the `MailmanProxy` class in the `smtpd` module, as it is unusable without the external `mailman` package. (Contributed by Donghee Na in bpo-35800 [https://bugs.python.org/issue?@action=redirect&bpo=35800].)

- Se eliminó el método obsoleto `split()` de `_tkinter.TkappType`. (Aportado por Erlend E. Aasland en bpo-38371 [https://bugs.python.org/issue?@action=redirect&bpo=38371].)

- Se eliminó la compatibilidad con el paquete de espacio de nombres del descubrimiento `unittest`. Se introdujo en Python 3.4 pero se rompió desde Python 3.7. (Aportado por Inada Naoki en bpo-23882 [https://bugs.python.org/issue?@action=redirect&bpo=23882].)

- Removed the undocumented private **`float.__set_format__()`** method, previously known as **`float.__setformat__()`** in Python 3.7. Its docstring said: «You probably don't want to use this function. It exists mainly to be used in Python's test suite.» (Contributed by Victor Stinner in bpo-46852 [https://bugs.python.org/issue?@action=redirect&bpo=46852].)

- El indicador de configuración **`--experimental-isolated-subinterpreters`** (y la macro **`EXPERIMENTAL_ISOLATED_SUBINTERPRETERS`** correspondiente) se han eliminado.

- Pynche [https://pypi.org/project/Pynche/] — The Pythonically Natural Color and Hue Editor — has been moved out of `Tools/scripts` and is being developed independently [https://gitlab.com/warsaw/pynche/-/tree/main] from the Python source tree.

Migración a Python 3.11

Esta sección enumera los cambios descritos anteriormente y otras correcciones de errores en la API de Python que pueden requerir cambios en su código de Python.

Las notas de portabilidad para la API de C son listed separately.

- **`open()`**, **`io.open()`**, **`codecs.open()`** y **`fileinput.FileInput`** ya no aceptan `'U'` («nueva línea universal») en el modo de archivo. En Python 3, el modo «nueva línea universal» se usa de forma predeterminada cada vez que se abre un archivo en modo de texto, y el indicador `'U'` ha quedado obsoleto desde Python 3.3. El newline parameter para estas funciones controla cómo funcionan las nuevas líneas universales. (Aportado por Victor Stinner en bpo-37330 [https://bugs.python.org/issue?@action=redirect&bpo=37330].)
- Las posiciones de los nodos **`ast.AST`** ahora se validan cuando se proporcionan a **`compile()`** y otras funciones relacionadas. Si se detectan posiciones no válidas, se generará

un `ValueError`. (Aportado por Pablo Galindo en gh-93351 [https://github.com/python/cpython/issues/93351])

- Prohibido pasar ejecutores que no sean `concurrent.futures.ThreadPoolExecutor` a `asyncio.loop.set_default_executor()` luego de una obsolescencia en Python 3.8. (Aportado por Illia Volochii en bpo-43234 [https://bugs.python.org/issue?@action=redirect&bpo=43234].)
- `calendar`: las clases `calendar.LocaleTextCalendar` y `calendar.LocaleHTMLCalendar` ahora usan `locale.getlocale()`, en lugar de usar `locale.getdefaultlocale()`, si no se especifica una configuración regional. (Aportado por Victor Stinner en bpo-46659 [https://bugs.python.org/issue?@action=redirect&bpo=46659].)
- El módulo `pdb` ahora lee el archivo de configuración `.pdbrc` con la codificación `'UTF-8'`. (Contribuido por Srinivas Reddy Thatiparthy (శ్రీనివాస్ రెడ్డి తాటిపర్తి) en bpo-41137 [https://bugs.python.org/issue?@action=redirect&bpo=41137].)
- El parámetro *population* de `random.sample()` debe ser una secuencia y ya no se admite la conversión automática de `set`s a `list`s. Además, si el tamaño de la muestra es mayor que el tamaño de la población, se genera un `ValueError`. (Aportado por Raymond Hettinger en bpo-40465 [https://bugs.python.org/issue?@action=redirect&bpo=40465].)
- Se eliminó el parámetro opcional *random* de `random.shuffle()`. Anteriormente, era una función aleatoria arbitraria para usar en la reproducción aleatoria; ahora, siempre se utilizará `random.random()` (su valor predeterminado anterior).
- En `re` Sintaxis de expresiones regulares, las banderas en línea globales (por ejemplo, `(?i)`) ahora solo se pueden usar al comienzo de las expresiones regulares. Su uso en otros lugares ha quedado obsoleto desde Python 3.6. (Aportado por Serhiy Storchaka en bpo-47066 [https://bugs.python.org/issue?@action=redirect&bpo=47066].)
- En el módulo `re`, se corrigieron varios errores antiguos que, en casos excepcionales, podían hacer que los grupos de captura obtuvieran un resultado incorrecto. Por lo tanto, esto podría cambiar la salida capturada en estos casos.

(Contribuido por Ma Lin en bpo-35859 [https://bugs.python.org/issue?@action=redirect&bpo=35859].)

Construir cambios

- CPython ahora tiene **PEP 11** [https://peps.python.org/pep-0011/] **Tier 3 support** [https://peps.python.org/pep-0011/#tier-3] para la compilación cruzada en las plataformas WebAssembly [https://webassembly.org/] Emscripten [https://emscripten.org/] (`wasm32-unknown-emscripten`, es decir, Python en el navegador) y WebAssembly System Interface (WASI) [https://wasi.dev/] (`wasm32-unknown-wasi`). El esfuerzo está inspirado en trabajos anteriores como Pyodide [https://pyodide.org/]. Estas plataformas proporcionan un subconjunto limitado de API POSIX; Las funciones y módulos de las bibliotecas estándar de Python relacionadas con redes, procesos, subprocesos, señales, mmap y usuarios/grupos no están disponibles o no funcionan. (Escritos aportados por Christian Heimes y Ethan Smith en gh-84461 [https://github.com/python/cpython/issues/84461] y WASI aportados por Christian Heimes en gh-90473 [https://github.com/python/cpython/issues/90473]; plataformas promocionadas en gh-95085 [https://github.com/python/cpython/issues/95085])

- Construir CPython ahora requiere:

  - ○ Un compilador C11 [https://en.cppreference.com/w/c/11] y una biblioteca estándar. Optional C11 features [https://en.wikipedia.org/wiki/C11_(C_standard_revision)#Optional_features] no son necesarios. (Aportado por Victor Stinner en bpo-46656 [https://bugs.python.org/issue?@action=redirect&bpo=46656], bpo-45440 [https://bugs.python.org/issue?@action=redirect&bpo=45440] y bpo-46640 [https://bugs.python.org/issue?@action=redirect&bpo=46640].)
  - ○ Support for IEEE 754 [https://en.wikipedia.org/wiki/IEEE_754] floating-point numbers. (Contributed by Victor Stinner in bpo-46917 [https://bugs.python.org/issue?@action=redirect&bpo=46917].)

- La macro `Py_NO_NAN` ha sido eliminada. Dado que CPython ahora requiere flotantes IEEE 754, los valores NaN siempre están disponibles. (Contribución de Victor Stinner en bpo-46656 [https://bugs.python.org/issue?@action=redirect&bpo=46656].)

- El paquete `tkinter` ahora requiere Tcl/Tk [https://www.tcl.tk] versión 8.5.12 o posterior. (Aportado por Serhiy Storchaka en bpo-46996 [https://bugs.python.org/issue?@action=redirect&bpo=46996].)

- Las dependencias de compilación, los indicadores del compilador y los indicadores del vinculador para la mayoría de los módulos de extensión stdlib ahora son detectados por **configure**. pkg-config [https://www.freedesktop.org/wiki/Software/pkg-config/] detecta los indicadores libffi, libnsl, libsqlite3, zlib, bzip2, liblzma, libcrypt, Tcl/Tk y uuid (si están disponibles). `tkinter` ahora requiere un comando pkg-config para detectar configuraciones de desarrollo para encabezados y bibliotecas Tcl/Tk [https://www.tcl.tk]. (Aportado por Christian Heimes y Erlend Egeberg Aasland en bpo-45847 [https://bugs.python.org/issue?@action=redirect&bpo=45847], bpo-45747 [https://bugs.python.org/issue?@action=redirect&bpo=45747] y bpo-45763 [https://bugs.python.org/issue?@action=redirect&bpo=45763]).

- libpython ya no está vinculado con libcrypt. (Aportado por Mike Gilbert en bpo-45433 [https://bugs.python.org/issue?@action=redirect&bpo=45433].)

- CPython ahora se puede construir con la opción ThinLTO [https://clang.llvm.org/docs/ThinLTO.html] pasando `thin` a `--with-lto`, es decir, `--with-lto=thin`. (Contribución de Donghee Na y Brett Holman en bpo-44340 [https://bugs.python.org/issue?@action=redirect&bpo=44340]).

- Freelists for object structs can now be disabled. A new **configure** option `--without-freelists` can be used to disable all freelists except empty tuple singleton. (Contributed by Christian Heimes in bpo-45522 [https://bugs.python.org/issue?@action=redirect&bpo=45522].)

- `Modules/Setup` y `Modules/makesetup` se han mejorado y atado. Los módulos de extensión ahora se pueden construir a través de `makesetup`. Todos, excepto algunos módulos de prueba, se pueden vincular estáticamente a una biblioteca o binario principal. (Aportado por Brett Cannon y Christian Heimes en bpo-45548 [https://bugs.python.org/issue?@action=redirect&bpo=45548], bpo-45570 [https://bugs.python.org/issue?@action=redirect&bpo=45570], bpo-45571 [https://bugs.python.org/issue?@action=redirect&bpo=45571] y bpo-43974 [https://bugs.python.org/issue?@action=redirect&bpo=43974]).

  Nota

  Utilice las variables de entorno **`TCLTK_CFLAGS`** y **`TCLTK_LIBS`** para especificar manualmente la ubicación de los encabezados y bibliotecas Tcl/Tk. Se han eliminado las opciones **configure** **`--with-tcltk-includes`** y **`--with-tcltk-libs`**.

  En RHEL 7 y CentOS 7, los paquetes de desarrollo no proporcionan `tcl.pc` y `tk.pc`; usa `TCLTK_LIBS="-ltk8.5 -ltkstub8.5 -ltcl8.5"`. El directorio `Misc/rhel7` contiene archivos `.pc` e instrucciones sobre cómo compilar Python con Tcl/Tk y OpenSSL de RHEL 7 y CentOS 7.

- CPython ahora usará dígitos de 30 bits de manera predeterminada para la implementación de Python **`int`**. Anteriormente, el valor predeterminado era usar dígitos de 30 bits en plataformas con `SIZEOF_VOID_P >= 8` y dígitos de 15 bits en caso contrario. Todavía es posible solicitar explícitamente el uso de dígitos de 15 bits a través de la opción **`--enable-big-digits`** en el script de configuración o (para Windows) la variable `PYLONG_BITS_IN_DIGIT` en `PC/pyconfig.h`, pero es posible que esta opción se elimine en algún momento en el futuro. (Contribuido por Mark Dickinson en bpo-45569 [https://bugs.python.org/issue?@action=redirect&bpo=45569].)

Cambios en la API de C

Nuevas características

- Agregue una nueva función `PyType_GetName()` para obtener el nombre corto del tipo. (Aportado por Hai Shi en bpo-42035 [https://bugs.python.org/issue?@action=redirect&bpo=42035].)
- Agregue una nueva función `PyType_GetQualName()` para obtener el nombre completo del tipo. (Aportado por Hai Shi en bpo-42035 [https://bugs.python.org/issue?@action=redirect&bpo=42035].)
- Agregue nuevas funciones `PyThreadState_EnterTracing()` y `PyThreadState_LeaveTracing()` a la API de C limitada para suspender y reanudar el seguimiento y la creación de perfiles. (Aportado por Victor Stinner en bpo-43760 [https://bugs.python.org/issue?@action=redirect&bpo=43760].)
- Se agregó la constante `Py_Version` que tiene el mismo valor que `PY_VERSION_HEX`. (Aportado por Gabriele N. Tornetta en bpo-43931 [https://bugs.python.org/issue?@action=redirect&bpo=43931].)
- `Py_buffer` y las API ahora forman parte de la API limitada y la ABI estable:
  - `PyObject_CheckBuffer()`
  - `PyObject_GetBuffer()`
  - `PyBuffer_GetPointer()`
  - `PyBuffer_SizeFromFormat()`
  - `PyBuffer_ToContiguous()`
  - `PyBuffer_FromContiguous()`
  - `PyObject_CopyData()`
  - `PyBuffer_IsContiguous()`
  - `PyBuffer_FillContiguousStrides()`
  - `PyBuffer_FillInfo()`
  - `PyBuffer_Release()`

  - `PyMemoryView_FromBuffer()`
  - Ranuras de tipo `bf_getbuffer` y `bf_releasebuffer`

  (Aportado por Christian Heimes en bpo-45459 [https://bugs.python.org/issue?@action=redirect&bpo=45459].)

- Se agregó la función `PyType_GetModuleByDef()`, utilizada para obtener el módulo en el que se definió un método, en los casos en que esta información no esté disponible directamente (a través de `PyCMethod`). (Aportado por Petr Viktorin en bpo-46613 [https://bugs.python.org/issue?@action=redirect&bpo=46613].)

- Agrega nuevas funciones para empaquetar y desempaquetar C doble (serializar y deserializar): `PyFloat_Pack2()`, `PyFloat_Pack4()`, `PyFloat_Pack8()`, `PyFloat_Unpack2()`, `PyFloat_Unpack4()` y `PyFloat_Unpack8()`. (Aportado por Victor Stinner en bpo-46906 [https://bugs.python.org/issue?@action=redirect&bpo=46906].)

- Agregue nuevas funciones para obtener atributos de objetos de marco: `PyFrame_GetBuiltins()`, `PyFrame_GetGenerator()`, `PyFrame_GetGlobals()`, `PyFrame_GetLasti()`.

- Se agregaron dos funciones nuevas para obtener y configurar la instancia de excepción activa: `PyErr_GetHandledException()` y `PyErr_SetHandledException()`. Estas son alternativas a `PyErr_SetExcInfo()` y `PyErr_GetExcInfo()` que funcionan con la representación de excepciones heredada de 3 tuplas. (Aportado por Irit Katriel en bpo-46343 [https://bugs.python.org/issue?@action=redirect&bpo=46343].)

- Se agregó el miembro `PyConfig.safe_path`. (Aportado por Victor Stinner en gh-57684 [https://github.com/python/cpython/issues/57684].)

Migración a Python 3.11

- Algunas macros se han convertido en funciones estáticas en línea para evitar macro pitfalls [https://gcc.gnu.org/onlinedocs/cpp/Macro-Pitfalls.html]. El cambio debería ser en su mayoría transparente para los usuarios, ya que las funciones de reemplazo emitirán sus argumentos a los tipos esperados para evitar advertencias del compilador debido a comprobaciones de tipos estáticos. Sin embargo, cuando la API de C limitada se establece en > =3.11, estas conversiones no se realizan y las personas que llaman deberán convertir argumentos a sus tipos esperados. Ver **PEP 670** [https://peps.python.org/pep-0670/] para más detalles. (Aportado por Victor Stinner y Erlend E. Aasland en gh-89653 [https://github.com/python/cpython/issues/89653].)

- **`PyErr_SetExcInfo()`** ya no usa los argumentos `type` y `traceback`, el intérprete ahora deriva esos valores de la instancia de excepción (el argumento `value`). La función aún roba referencias de los tres argumentos. (Aportado por Irit Katriel en bpo-45711 [https://bugs.python.org/issue?@action=redirect&bpo=45711].)

- **`PyErr_GetExcInfo()`** ahora deriva los campos `type` y `traceback` del resultado de la instancia de excepción (el campo `value`). (Aportado por Irit Katriel en bpo-45711 [https://bugs.python.org/issue?@action=redirect&bpo=45711].)

- **`_frozen`** tiene un nuevo campo `is_package` para indicar si el módulo congelado es o no un paquete. Anteriormente, un valor negativo en el campo `size` era el indicador. Ahora solo se pueden usar valores no negativos para `size`. (Aportado por Kumar Aditya en bpo-46608 [https://bugs.python.org/issue?@action=redirect&bpo=46608].)

- **`_PyFrameEvalFunction()`** ahora toma `_PyInterpreterFrame*` como su segundo parámetro, en lugar de `PyFrameObject*`. Consulte **PEP 523** [https://peps.python.org/pep-0523/] para obtener más detalles sobre cómo usar este tipo de puntero de función.

- **`PyCode_New()`** and **`PyCode_NewWithPosOnlyArgs()`** now take an additional `exception_table` argument. Using these functions should be avoided, if at all possible. To get a custom code object: create a code object using the compiler, then get a modified version with the `replace` method.

- **`PyCodeObject`** ya no tiene los campos `co_code`, `co_varnames`, `co_cellvars` y `co_freevars`. En su lugar, utilice **`PyCode_GetCode()`**, **`PyCode_GetVarnames()`**, **`PyCode_GetCellvars()`** y **`PyCode_GetFreevars()`** respectivamente para acceder a ellos a través de la API de C. (Aportado por Brandt Bucher en bpo-46841 [https://bugs.python.org/issue?@action=redirect&bpo=46841] y Ken Jin en gh-92154 [https://github.com/python/cpython/issues/92154] y gh-94936 [https://github.com/python/cpython/issues/94936]).

- Las antiguas macros de papelera (`Py_TRASHCAN_SAFE_BEGIN`/`Py_TRASHCAN_SAFE_END`) ahora están obsoletas. Deberían ser reemplazadas por las nuevas macros `Py_TRASHCAN_BEGIN` y `Py_TRASHCAN_END`.

  Una función tp_dealloc que tiene las macros antiguas, como:

  ```
  static void
  mytype_dealloc(mytype *p)
  {
      PyObject_GC_UnTrack(p);
      Py_TRASHCAN_SAFE_BEGIN(p);
      ...
      Py_TRASHCAN_SAFE_END
  }
  ```

  debe migrar a las nuevas macros de la siguiente manera:

  ```
  static void
  mytype_dealloc(mytype *p)
  {
      PyObject_GC_UnTrack(p);
      Py_TRASHCAN_BEGIN(p, mytype_dealloc)
      ...
  ```

```
    Py_TRASHCAN_END
}
```

Tenga en cuenta que `Py_TRASHCAN_BEGIN` tiene un segundo argumento que debería ser la función de desasignación en la que se encuentra.

Para admitir versiones anteriores de Python en la misma base de código, puede definir las siguientes macros y usarlas en todo el código (crédito: se copiaron de la base de código `mypy`):

```
#if PY_VERSION_HEX >= 0x03080000
#  define CPy_TRASHCAN_BEGIN(op, dealloc) Py_TRASHC
#  define CPy_TRASHCAN_END(op) Py_TRASHCAN_END
#else
#  define CPy_TRASHCAN_BEGIN(op, dealloc) Py_TRASHC
#  define CPy_TRASHCAN_END(op) Py_TRASHCAN_SAFE_END
#endif
```

- La función **`PyType_Ready()`** ahora genera un error si un tipo se define con el indicador **`Py_TPFLAGS_HAVE_GC`** establecido pero no tiene función transversal (**`PyTypeObject.tp_traverse`**). (Aportado por Victor Stinner en bpo-44263 [https://bugs.python.org/issue?@action=redirect&bpo=44263].)

- Los tipos de montón con el indicador **`Py_TPFLAGS_IMMUTABLETYPE`** ahora pueden heredar el protocolo de llamada vectorial **PEP 590** [https://peps.python.org/pep-0590/]. Anteriormente, esto sólo era posible para static types. (Aportado por Erlend E. Aasland en bpo-43908 [https://bugs.python.org/issue?@action=redirect&bpo=43908])

- Dado que **`Py_TYPE()`** se cambia a una función estática en línea, `Py_TYPE(obj) = new_type` debe reemplazarse por `Py_SET_TYPE(obj, new_type)`: consulte la función **`Py_SET_TYPE()`** (disponible desde Python 3.9). Para compatibilidad con versiones anteriores, se puede usar esta macro:

```
#if PY_VERSION_HEX < 0x030900A4 && !defined(Py_SET_
static inline void _Py_SET_TYPE(PyObject *ob, PyTyp
{ ob->ob_type = type; }
#define Py_SET_TYPE(ob, type) _Py_SET_TYPE((PyObjec
#endif
```

(Aportado por Victor Stinner en bpo-39573 [https://bugs.python.org/issue?@action=redirect&bpo=39573].)

- Dado que **`Py_SIZE()`** se cambia a una función estática en línea, `Py_SIZE(obj) = new_size` debe reemplazarse por `Py_SET_SIZE(obj, new_size)`: consulte la función **`Py_SET_SIZE()`** (disponible desde Python 3.9). Para compatibilidad con versiones anteriores, se puede usar esta macro:

  ```
  #if PY_VERSION_HEX < 0x030900A4 && !defined(Py_SET_
  static inline void _Py_SET_SIZE(PyVarObject *ob, Py
  { ob->ob_size = size; }
  #define Py_SET_SIZE(ob, size) _Py_SET_SIZE((PyVarOb
  #endif
  ```

  (Aportado por Victor Stinner en bpo-39573 [https://bugs.python.org/issue?@action=redirect&bpo=39573].)

- `<Python.h>` ya no incluye los archivos de encabezado `<stdlib.h>`, `<stdio.h>`, `<errno.h>` y `<string.h>` cuando la macro `Py_LIMITED_API` se establece en `0x030b0000` (Python 3.11) o superior. Las extensiones de C deben incluir explícitamente los archivos de encabezado después de `#include <Python.h>`. (Aportado por Victor Stinner en bpo-45434 [https://bugs.python.org/issue?@action=redirect&bpo=45434].)

- Los archivos de API no limitados `cellobject.h`, `classobject.h`, `code.h`, `context.h`, `funcobject.h`, `genobject.h` y `longintrepr.h` se han movido al directorio `Include/cpython`. Además, se eliminó el archivo de encabezado `eval.h`. Estos archivos no deben incluirse directamente, ya que ya están incluidos en `Python.h`: Include Files. Si se han incluido directamente, considere incluir `Python.h` en su lugar. (Aportado por Victor Stinner en bpo-35134 [https://bugs.python.org/issue?@action=redirect&bpo=35134].)

- The **`PyUnicode_CHECK_INTERNED()`** macro has been excluded from the limited C API. It was never usable there, because it used internal structures which are not available in the limited C API. (Contributed by Victor Stinner in bpo-46007 [https://bugs.python.org/issue?@action=redirect&bpo=46007].)

- Las siguientes funciones y tipos de cuadros ahora están disponibles directamente con `#include <Python.h>`, ya no es necesario agregar `#include <frameobject.h>`:

  - ○ **`PyFrame_Check()`**
  - ○ **`PyFrame_GetBack()`**
  - ○ **`PyFrame_GetBuiltins()`**
  - ○ **`PyFrame_GetGenerator()`**
  - ○ **`PyFrame_GetGlobals()`**
  - ○ **`PyFrame_GetLasti()`**
  - ○ **`PyFrame_GetLocals()`**
  - ○ **`PyFrame_Type`**

  (Aportado por Victor Stinner en gh-93937 [https://github.com/python/cpython/issues/93937].)

- Los miembros de la estructura **`PyFrameObject`** se han eliminado de la API de C pública.

  Si bien la documentación señala que los campos **`PyFrameObject`** están sujetos a cambios en cualquier momento, se han mantenido estables durante mucho tiempo y se usaron en varias extensiones populares.

  En Python 3.11, la estructura del marco se reorganizó para permitir optimizaciones de rendimiento. Algunos campos se eliminaron por completo, ya que eran detalles de la implementación anterior.

  Campos **`PyFrameObject`**:

- `f_back`: usa **`PyFrame_GetBack()`**.
- `f_blockstack`: eliminado.
- `f_builtins`: usa **`PyFrame_GetBuiltins()`**.
- `f_code`: usa **`PyFrame_GetCode()`**.
- `f_gen`: usa **`PyFrame_GetGenerator()`**.
- `f_globals`: usa **`PyFrame_GetGlobals()`**.
- `f_iblock`: eliminado.
- `f_lasti`: usa **`PyFrame_GetLasti()`**. El código que usa `f_lasti` con `PyCode_Addr2Line()` debería usar **`PyFrame_GetLineNumber()`** en su lugar; puede ser más rápido.
- `f_lineno`: usar **`PyFrame_GetLineNumber()`**
- `f_locals`: usa **`PyFrame_GetLocals()`**.
- `f_stackdepth`: eliminado.
- `f_state`: sin API pública (renombrado como `f_frame.f_state`).
- `f_trace`: sin API pública.
- `f_trace_lines`: utiliza `PyObject_GetAttrString((PyObject*)frame, "f_trace_lines")`.
- `f_trace_opcodes`: utiliza `PyObject_GetAttrString((PyObject*)frame, "f_trace_opcodes")`.
- `f_localsplus`: sin API pública (renombrado como `f_frame.localsplus`).
- `f_valuestack`: eliminado.

The Python frame object is now created lazily. A side effect is that the **`f_back`** member must not be accessed directly, since its value is now also computed lazily. The **`PyFrame_GetBack()`** function must be called instead.

Debuggers that accessed the **`f_locals`** directly *must* call **`PyFrame_GetLocals()`** instead. They no longer need to call **`PyFrame_FastToLocalsWithError()`** or **`PyFrame_LocalsToFast()`**, in fact they should not call those functions. The necessary updating of the frame is now managed by the virtual machine.

Código que define `PyFrame_GetCode()` en Python 3.8 y anteriores:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyCodeObject* PyFrame_GetCode(PyFrame
{
    Py_INCREF(frame->f_code);
    return frame->f_code;
}
#endif
```

Código que define `PyFrame_GetBack()` en Python 3.8 y anteriores:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyFrameObject* PyFrame_GetBack(PyFram
{
    Py_XINCREF(frame->f_back);
    return frame->f_back;
}
#endif
```

O use pythoncapi_compat project [https://github.com/python/pythoncapi-compat] para obtener estas dos funciones en versiones anteriores de Python.

- Cambios de los miembros de la estructura **`PyThreadState`**:
    - `frame`: eliminado, use **`PyThreadState_GetFrame()`** (función agregada a Python 3.9 por bpo-40429 [https://bugs.python.org/issue?@action=redirect&bpo=40429]). Advertencia: la función devuelve un strong reference, necesita llamar a **`Py_XDECREF()`**.
    - `tracing`: cambiado, use **`PyThreadState_EnterTracing()`** y **`PyThreadState_LeaveTracing()`** (funciones agregadas a Python 3.11 por bpo-43760 [https://bugs.python.org/issue?@action=redirect&bpo=43760]).
    - `recursion_depth`: eliminado, use `(tstate->recursion_limit - tstate-`

`>recursion_remaining)` en su lugar.

  - `stackcheck_counter`: eliminado.

Código que define `PyThreadState_GetFrame()` en Python 3.8 y anteriores:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyFrameObject* PyThreadState_GetFrame
{
    Py_XINCREF(tstate->frame);
    return tstate->frame;
}
#endif
```

Código que define `PyThreadState_EnterTracing()` y `PyThreadState_LeaveTracing()` en Python 3.10 y versiones anteriores:

```
#if PY_VERSION_HEX < 0x030B00A2
static inline void PyThreadState_EnterTracing(PyThr
{
    tstate->tracing++;
#if PY_VERSION_HEX >= 0x030A00A1
    tstate->cframe->use_tracing = 0;
#else
    tstate->use_tracing = 0;
#endif
}

static inline void PyThreadState_LeaveTracing(PyThr
{
    int use_tracing = (tstate->c_tracefunc != NULL
    tstate->tracing--;
#if PY_VERSION_HEX >= 0x030A00A1
    tstate->cframe->use_tracing = use_tracing;
#else
    tstate->use_tracing = use_tracing;
#endif
}
```

```
#endif
```

O use the pythoncapi-compat project [https://github.com/python/pythoncapi-compat] para obtener estas funciones en funciones antiguas de Python.

- Se alienta a los distribuidores a compilar Python con la biblioteca Blake2 optimizada libb2 [https://www.blake2.net/].

- El campo **`PyConfig.module_search_paths_set`** ahora debe establecerse en 1 para que la inicialización use **`PyConfig.module_search_paths`** para inicializar **`sys.path`**. De lo contrario, la inicialización volverá a calcular la ruta y reemplazará los valores agregados a `module_search_paths`.

- **`PyConfig_Read()`** ya no calcula la ruta de búsqueda inicial y no completará ningún valor en **`PyConfig.module_search_paths`**. Para calcular rutas predeterminadas y luego modificarlas, finalice la inicialización y use **`PySys_GetObject()`** para recuperar **`sys.path`** como un objeto de lista de Python y modificarlo directamente.

Obsoleto

- Deseche las siguientes funciones para configurar la inicialización de Python:
    - ○ **`PySys_AddWarnOptionUnicode()`**
    - ○ **`PySys_AddWarnOption()`**
    - ○ **`PySys_AddXOption()`**
    - ○ **`PySys_HasWarnOptions()`**
    - ○ **`PySys_SetArgvEx()`**
    - ○ **`PySys_SetArgv()`**
    - ○ **`PySys_SetPath()`**
    - ○ **`Py_SetPath()`**
    - ○ **`Py_SetProgramName()`**
    - ○ **`Py_SetPythonHome()`**
    - ○ **`Py_SetStandardStreamEncoding()`**
    - ○ **`_Py_SetProgramFullPath()`**

Utilice la nueva API **`PyConfig`** de Python Initialization Configuration en su lugar (**PEP 587** [https://peps.python.org/pep-0587/]). (Aportado por Victor Stinner en gh-88279 [https://github.com/python/cpython/issues/88279].)

- Deje obsoleto el miembro `ob_shash` de **`PyBytesObject`**. Utilice **`PyObject_Hash()`** en su lugar. (Aportado por Inada Naoki en bpo-46864 [https://bugs.python.org/issue?@action=redirect&bpo=46864].)

Eliminación pendiente en Python 3.12

Las siguientes API de C quedaron obsoletas en versiones anteriores de Python y se eliminarán en Python 3.12.

- **`PyUnicode_AS_DATA()`**
- **`PyUnicode_AS_UNICODE()`**
- **`PyUnicode_AsUnicodeAndSize()`**
- **`PyUnicode_AsUnicode()`**
- **`PyUnicode_FromUnicode()`**
- **`PyUnicode_GET_DATA_SIZE()`**
- **`PyUnicode_GET_SIZE()`**
- **`PyUnicode_GetSize()`**
- **`PyUnicode_IS_COMPACT()`**
- **`PyUnicode_IS_READY()`**
- **`PyUnicode_READY()`**
- **`Py_UNICODE_WSTR_LENGTH()`**
- **`_PyUnicode_AsUnicode()`**
- **`PyUnicode_WCHAR_KIND`**
- **`PyUnicodeObject`**
- **`PyUnicode_InternImmortal()`**

Remoto

- Se han eliminado **`PyFrame_BlockSetup()`** y **`PyFrame_BlockPop()`**. (Aportado por Mark Shannon en bpo-40222 [https://bugs.python.org/issue?@action=redirect&bpo=40222].)

- Quite las siguientes macros matemáticas usando la variable `errno`:

  - `Py_ADJUST_ERANGE1()`
  - `Py_ADJUST_ERANGE2()`
  - `Py_OVERFLOWED()`
  - `Py_SET_ERANGE_IF_OVERFLOW()`
  - `Py_SET_ERRNO_ON_MATH_ERROR()`

  (Aportado por Victor Stinner en bpo-45412 [https://bugs.python.org/issue?@action=redirect&bpo=45412].)

- Elimine las macros `Py_UNICODE_COPY()` y `Py_UNICODE_FILL()`, obsoletas desde Python 3.3. Utilice `PyUnicode_CopyCharacters()` o `memcpy()` (cadena `wchar_t*`) y las funciones `PyUnicode_Fill()` en su lugar. (Aportado por Victor Stinner en bpo-41123 [https://bugs.python.org/issue?@action=redirect&bpo=41123].)

- Elimine el archivo de encabezado `pystrhex.h`. Solo contiene funciones privadas. Las extensiones C solo deben incluir el archivo de encabezado principal `<Python.h>`. (Aportado por Victor Stinner en bpo-45434 [https://bugs.python.org/issue?@action=redirect&bpo=45434].)

- Quite la macro `Py_FORCE_DOUBLE()`. Fue utilizado por la macro `Py_IS_INFINITY()`. (Aportado por Victor Stinner en bpo-45440 [https://bugs.python.org/issue?@action=redirect&bpo=45440].)

- Los siguientes elementos ya no están disponibles cuando se define **`Py_LIMITED_API`**:

  - **`PyMarshal_WriteLongToFile()`**
  - **`PyMarshal_WriteObjectToFile()`**
  - **`PyMarshal_ReadObjectFromString()`**
  - **`PyMarshal_WriteObjectToString()`**
  - la macro `Py_MARSHAL_VERSION`

  Estos no forman parte del limited API.

  (Aportado por Victor Stinner en bpo-45474 [https://

bugs.python.org/issue?@action=redirect&bpo=45474].)

- Exclude `PyWeakref_GET_OBJECT()` from the limited C API. It never worked since the **`PyWeakReference`** structure is opaque in the limited C API. (Contributed by Victor Stinner in bpo-35134 [https://bugs.python.org/issue?@action=redirect&bpo=35134].)
- Quite la macro `PyHeapType_GET_MEMBERS()`. Fue expuesto en la API pública de C por error, solo debe ser utilizado por Python internamente. Utilice el miembro `PyTypeObject.tp_members` en su lugar. (Aportado por Victor Stinner en bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170].)
- Elimine la macro `HAVE_PY_SET_53BIT_PRECISION` (movida a la API de C interna). (Aportado por Victor Stinner en bpo-45412 [https://bugs.python.org/issue?@action=redirect&bpo=45412].)
- Elimine las API del codificador `Py_UNICODE`, ya que han quedado obsoletas desde Python 3.3, se usan poco y son ineficientes en relación con las alternativas recomendadas.

  Las funciones eliminadas son:

  ○ **`PyUnicode_Encode()`**
  ○ **`PyUnicode_EncodeASCII()`**
  ○ **`PyUnicode_EncodeLatin1()`**
  ○ **`PyUnicode_EncodeUTF7()`**
  ○ **`PyUnicode_EncodeUTF8()`**
  ○ **`PyUnicode_EncodeUTF16()`**
  ○ **`PyUnicode_EncodeUTF32()`**
  ○ **`PyUnicode_EncodeUnicodeEscape()`**
  ○ **`PyUnicode_EncodeRawUnicodeEscape()`**
  ○ **`PyUnicode_EncodeCharmap()`**
  ○ **`PyUnicode_TranslateCharmap()`**
  ○ **`PyUnicode_EncodeDecimal()`**
  ○ **`PyUnicode_TransformDecimalToASCII()`**

  Ver **PEP 624** [https://peps.python.org/pep-0624/] para más detalles

y **migration guidance** [https://peps.python.org/pep-0624/#alternative-apis]. (Aportado por Inada Naoki en bpo-44029 [https://bugs.python.org/issue?@action=redirect&bpo=44029].)

Notable changes in 3.11.4

tarfile

- The extraction methods in **`tarfile`**, and **`shutil.unpack_archive()`**, have a new a *filter* argument that allows limiting tar features than may be surprising or dangerous, such as creating files outside the destination directory. See Extraction filters for details. In Python 3.12, use without the *filter* argument will show a **`DeprecationWarning`**. In Python 3.14, the default will switch to `'data'`. (Contributed by Petr Viktorin in **PEP 706** [https://peps.python.org/pep-0706/].)

Notable changes in 3.11.5

OpenSSL

- Windows builds and macOS installers from python.org now use OpenSSL 3.0.

Qué hay de nuevo en Python 3.10

Editor:

Pablo Galindo Salgado

Este artículo explica las nuevas características de Python 3.10, en comparación con la 3.9. Python 3.10 se publicó el 4 de octubre de 2021. Para obtener todos los detalles, consulte el changelog.

Resumen: aspectos destacados de la versión

Nuevas funciones de sintaxis:

- **PEP 634** [https://peps.python.org/pep-0634/], Coincidencia de patrones estructurales: Especificación
- **PEP 635** [https://peps.python.org/pep-0635/], Coincidencia de patrones estructurales: motivación y fundamento
- **PEP 636** [https://peps.python.org/pep-0636/], Coincidencia de patrones estructurales: Tutorial
- bpo-12782 [https://bugs.python.org/issue?@action=redirect&bpo=12782], los administradores de contexto entre paréntesis ahora están oficialmente permitidos.

Nuevas funciones en la biblioteca estándar:

- **PEP 618** [https://peps.python.org/pep-0618/], agregue verificación de longitud opcional al cierre.

Mejoras en el intérprete:

- **PEP 626** [https://peps.python.org/pep-0626/], números de línea precisos para depuración y otras herramientas.

Nuevas funciones de escritura:

- **PEP 604** [https://peps.python.org/pep-0604/], Permitir escribir tipos de unión como X | Y
- **PEP 612** [https://peps.python.org/pep-0612/], variables de especificación de parámetros
- **PEP 613** [https://peps.python.org/pep-0613/], alias de tipo explícito
- **PEP 647** [https://peps.python.org/pep-0647/], protectores de tipo definidos por el usuario

Desactivaciones, eliminaciones o restricciones importantes:

- **PEP 644** [https://peps.python.org/pep-0644/], requiere OpenSSL 1.1.1 o más reciente
- **PEP 632** [https://peps.python.org/pep-0632/], módulo distutils obsoleto.
- **PEP 623** [https://peps.python.org/pep-0623/], desaprobar y prepararse para la eliminación del miembro wstr en PyUnicodeObject.
- **PEP 624** [https://peps.python.org/pep-0624/], eliminar las API del codificador Py_UNICODE
- **PEP 597** [https://peps.python.org/pep-0597/], agregar codificación opcional

Nuevas características

Administradores de contexto entre paréntesis

Ahora se admite el uso de entre paréntesis para continuar en varias líneas en los administradores de contexto. Esto permite formatear una colección larga de administradores de contexto en múltiples líneas de una manera similar a como era posible anteriormente con declaraciones de importación. Por ejemplo, todos estos ejemplos ahora son válidos:

```
with (CtxManager() as ejemplo):
    ...

with (
    CtxManager1(),
    CtxManager2()
):
    ...

with (CtxManager1() as ejemplo,
      CtxManager2()):
    ...

with (CtxManager1(),
      CtxManager2() as ejemplo):
    ...

with (
    CtxManager1() as ejemplo1,
    CtxManager2() as ejemplo2
):
    ...
```

también es posible usar una coma al final del grupo adjunto:

```
with (
    CtxManager1() as ejemplo1,
    CtxManager2() as ejemplo2,
    CtxManager3() as ejemplo3,
):
    ...
```

Esta nueva sintaxis utiliza las capacidades no LL (1) del nuevo analizador. Consulte **PEP 617** [https://peps.python.org/pep-0617/] para obtener más detalles.

(Contribuido por Guido van Rossum, Pablo Galindo y Lysandros Nikolaou en bpo-12782 [https://bugs.python.org/issue?@action=redirect&bpo=12782] y bpo-40334 [https://bugs.python.org/issue?@action=redirect&bpo=40334].)

Mejores mensajes de error

SyntaxErrors

Al analizar el código que contiene paréntesis o corchetes sin cerrar, el intérprete ahora incluye la ubicación del corchete de paréntesis sin cerrar en lugar de mostrar *SyntaxError: unexpected EOF while parsing* o señalar una ubicación incorrecta. Por ejemplo, considere el siguiente código (observe el “{” sin cerrar):

```
expected = {9: 1, 18: 2, 19: 2, 27: 3, 28: 3, 29: 3, 36:
            38: 4, 39: 4, 45: 5, 46: 5, 47: 5, 48: 5, 49
some_other_code = foo()
```

Las versiones anteriores del intérprete informaron lugares confusos como la ubicación del error de sintaxis:

```
File "example.py", line 3
    some_other_code = foo()
                    ^
SyntaxError: invalid syntax
```

pero en Python 3.10 se emite un error más informativo:

```
File "example.py", line 1
    expected = {9: 1, 18: 2, 19: 2, 27: 3, 28: 3, 29: 3,
               ^
SyntaxError: '{' was never closed
```

De manera similar, los errores que involucran cadenas literales no cerradas (entre comillas simples y triples) ahora apuntan al inicio de la cadena en lugar de informar EOF / EOL.

Estas mejoras están inspiradas en trabajos anteriores en el intérprete de PyPy.

(Contribuido por Pablo Galindo en bpo-42864 [https://bugs.python.org/issue?@action=redirect&bpo=42864] y Batuhan Taskaya en bpo-40176 [https://bugs.python.org/issue?@action=redirect&bpo=40176].)

Las excepciones de **`SyntaxError`** planteadas por el intérprete ahora resaltarán el rango de error completo de la expresión que constituye el error de sintaxis en sí, en lugar de solo dónde se detecta el problema. De esta manera, en lugar de mostrar (antes de Python 3.10):

```
>>> foo(x, z for z in range(10), t, w)
  File "<stdin>", line 1
    foo(x, z for z in range(10), t, w)
           ^
SyntaxError: Generator expression must be parenthesized
```

ahora Python 3.10 mostrará la excepción como:

```
>>> foo(x, z for z in range(10), t, w)
  File "<stdin>", line 1
    foo(x, z for z in range(10), t, w)
           ^^^^^^^^^^^^^^^^^^^^
SyntaxError: Generator expression must be parenthesized
```

Esta mejora fue aportada por Pablo Galindo en bpo-43914 [https://bugs.python.org/issue?@action=redirect&bpo=43914].

Se ha incorporado una cantidad considerable de nuevos mensajes especializados para excepciones **`SyntaxError`**. Algunos de los más notables son los siguientes:

- Falta `:` antes de los bloques:

  ```
  >>> if rocket.position > event_horizon
    File "<stdin>", line 1
      if rocket.position > event_horizon
                                        ^
  SyntaxError: expected ':'
  ```

  (Aportado por Pablo Galindo en bpo-42997 [https://bugs.python.org/issue?@action=redirect&bpo=42997].)

- Tuplas sin paréntesis en objetivos de comprensión:

  ```
  >>> {x,y for x,y in zip('abcd', '1234')}
    File "<stdin>", line 1
      {x,y for x,y in zip('abcd', '1234')}
       ^
  SyntaxError: did you forget parentheses around the
  ```

  (Aportado por Pablo Galindo en bpo-43017 [https://

bugs.python.org/issue?@action=redirect&bpo=43017].)

- Faltan comas en literales de colección y entre expresiones:

  ```
  >>> items = {
  ... x: 1,
  ... y: 2
  ... z: 3,
    File "<stdin>", line 3
      y: 2
         ^
  SyntaxError: invalid syntax. Perhaps you forgot a c
  ```

  (Aportado por Pablo Galindo en bpo-43822 [https://bugs.python.org/issue?@action=redirect&bpo=43822].)

- Varios tipos de excepciones sin paréntesis:

  ```
  >>> try:
  ...     build_dyson_sphere()
  ... except NotEnoughScienceError, NotEnoughResource
    File "<stdin>", line 3
      except NotEnoughScienceError, NotEnoughResource
             ^
  SyntaxError: multiple exception types must be paren
  ```

  (Aportado por Pablo Galindo en bpo-43149 [https://bugs.python.org/issue?@action=redirect&bpo=43149].)

- Falta `:` y valores en literales de diccionario:

  ```
  >>> values = {
  ... x: 1,
  ... y: 2,
  ... z:
  ... }
    File "<stdin>", line 4
      z:
       ^
  SyntaxError: expression expected after dictionary k
  ```

```
>>> values = {x:1, y:2, z w:3}
  File "<stdin>", line 1
    values = {x:1, y:2, z w:3}
                        ^
SyntaxError: ':' expected after dictionary key
```

(Aportado por Pablo Galindo en bpo-43823 [https://bugs.python.org/issue?@action=redirect&bpo=43823].)

- Bloques `try` sin bloques `except` o `finally`:

```
>>> try:
...     x = 2
... something = 3
  File "<stdin>", line 3
    something  = 3
    ^^^^^^^^^
SyntaxError: expected 'except' or 'finally' block
```

(Aportado por Pablo Galindo en bpo-44305 [https://bugs.python.org/issue?@action=redirect&bpo=44305].)

- Uso de `=` en lugar de `==` en comparaciones:

```
>>> if rocket.position = event_horizon:
  File "<stdin>", line 1
    if rocket.position = event_horizon:
                       ^
SyntaxError: cannot assign to attribute here. Maybe
```

(Aportado por Pablo Galindo en bpo-43797 [https://bugs.python.org/issue?@action=redirect&bpo=43797].)

- Uso de `*` en f-strings:

```
>>> f"Black holes {*all_black_holes} and revelation
  File "<stdin>", line 1
    (*all_black_holes)
     ^
```

```
SyntaxError: f-string: cannot use starred expressio
```

(Aportado por Pablo Galindo en bpo-41064 [https://bugs.python.org/issue?@action=redirect&bpo=41064].)

Errores de sangría

Muchas excepciones de `IndentationError` ahora tienen más contexto con respecto a qué tipo de bloque esperaba una sangría, incluida la ubicación de la declaración:

```
>>> def foo():
...    if lel:
...    x = 2
  File "<stdin>", line 3
    x = 2
    ^
IndentationError: expected an indented block after 'if'
```

AttributeErrors

Al imprimir `AttributeError`, **`PyErr_Display()`** ofrecerá sugerencias de nombres de atributos similares en el objeto desde el que se generó la excepción:

```
>>> collections.namedtoplo
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'collections' has no attribute 'r
```

(Contribuido por Pablo Galindo en bpo-38530 [https://bugs.python.org/issue?@action=redirect&bpo=38530].)

Advertencia

Tenga en cuenta que esto no funcionará si no se llama a **`PyErr_Display()`** para mostrar el error, lo que puede ocurrir si se utiliza alguna otra función de visualización de errores personalizada. Este es un escenario común en algunos REPL como IPython.

NameErrors

Al imprimir `NameError` generado por el intérprete, `PyErr_Display()` ofrecerá sugerencias de nombres de variables similares en la función desde la que se generó la excepción:

```
>>> schwarzschild_black_hole = None
>>> schwarschild_black_hole
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'schwarschild_black_hole' is not defined
```

(Contribuido por Pablo Galindo en bpo-38530 [https://bugs.python.org/issue?@action=redirect&bpo=38530].)

Advertencia

Tenga en cuenta que esto no funcionará si no se llama a `PyErr_Display()` para mostrar el error, lo que puede ocurrir si se utiliza alguna otra función de visualización de errores personalizada. Este es un escenario común en algunos REPL como IPython.

PEP 626: números de línea precisos para depuración y otras herramientas

PEP 626 ofrece números de línea más precisos y confiables para herramientas de depuración, creación de perfiles y cobertura. Los eventos de rastreo, con el número de línea correcto, se generan para todas las líneas de código ejecutadas y solo para las líneas de código que se ejecutan.

El atributo `f_lineno` de los objetos de marco siempre contendrá el número de línea esperado.

El atributo `co_lnotab` de code objects está obsoleto y se eliminará en 3.12. El código que necesita convertirse de desplazamiento a número de línea debe utilizar el nuevo método `co_lines()`.

PEP 634: Coincidencia de patrones estructurales

Se ha agregado la coincidencia de patrones estructurales en forma de un *match statement* and *case statements* de patrones con acciones asociadas. Los patrones constan de secuencias, asignaciones, tipos de datos primitivos e instancias de clases. La coincidencia de patrones permite a los programas extraer información de tipos de datos complejos, ramificar la estructura de los datos y aplicar acciones específicas basadas en diferentes formas de datos.

Sintaxis y operaciones

La sintaxis genérica de la coincidencia de patrones es:

```
match subject:
    case <pattern_1>:
        <action_1>
    case <pattern_2>:
        <action_2>
    case <pattern_3>:
        <action_3>
    case _:
        <action_wildcard>
```

Una declaración de coincidencia toma una expresión y compara su valor con los patrones sucesivos dados como uno o más bloques de casos. Específicamente, la coincidencia de patrones funciona mediante:

1. usando datos con tipo y forma (el `subject`)
2. evaluar el `subject` en la declaración `match`
3. comparar el sujeto con cada patrón en una declaración `case` de arriba a abajo hasta que se confirme una coincidencia.
4. ejecutar la acción asociada con el patrón de la coincidencia confirmada
5. Si no se confirma una coincidencia exacta, el último caso, un comodín `_`, si se proporciona, se utilizará como caso coincidente. Si no se confirma una coincidencia exacta y no

existe un comodín, todo el bloque de coincidencias es inactivo.

Enfoque declarativo

Los lectores pueden ser conscientes de la coincidencia de patrones a través del simple ejemplo de hacer coincidir un sujeto (objeto de datos) con un literal (patrón) con la declaración de cambio que se encuentra en C, Java o JavaScript (y muchos otros lenguajes). A menudo, la declaración de cambio se utiliza para comparar un objeto / expresión con declaraciones de casos que contienen literales.

Se pueden encontrar ejemplos más poderosos de coincidencia de patrones en lenguajes como Scala y Elixir. Con la coincidencia de patrones estructurales, el enfoque es «declarativo» y establece explícitamente las condiciones (los patrones) para que los datos coincidan.

Si bien una serie «imperativa» de instrucciones que utilizan declaraciones «if» anidadas podría usarse para lograr algo similar a la coincidencia de patrones estructurales, es menos claro que el enfoque «declarativo». En cambio, el enfoque «declarativo» establece las condiciones que se deben cumplir para una coincidencia y es más legible a través de sus patrones explícitos. Si bien la coincidencia de patrones estructurales se puede usar en su forma más simple comparando una variable con un literal en una declaración de caso, su verdadero valor para Python radica en su manejo del tipo y la forma del sujeto.

Patrón simple: coincidir con un literal

Veamos este ejemplo como coincidencia de patrones en su forma más simple: un valor, el sujeto, se empareja con varios literales, los patrones. En el siguiente ejemplo, `status` es el tema de la declaración de coincidencia. Los patrones son cada una de las declaraciones de casos, donde los literales representan códigos de estado de solicitud. La acción asociada al caso se ejecuta después de una coincidencia:

```
def http_error(estado):
    estado del partido:
        caso 400:
            devolver "Solicitud incorrecta"
        caso 404:
            devolver "No encontrado"
        caso 418:
            volver "soy una tetera"
        caso _:
            devolver "Algo anda mal con Internet"
```

Si a la función anterior se le pasa un `status` de 418, se devuelve «Soy una tetera». Si a la función anterior se le pasa un `status` de 500, la declaración de caso con _ coincidirá como un comodín y se devuelve «Algo anda mal con Internet». Tenga en cuenta el último bloque: el nombre de la variable, _, actúa como *wildcard* y asegura que el sujeto siempre coincidirá. El uso de _ es opcional.

Puede combinar varios literales en un solo patrón usando | («o»)

```
caso 401 | 403 | 404:
    devolver "No permitido"
```

Comportamiento sin el comodín

Si modificamos el ejemplo anterior eliminando el último bloque de caso, el ejemplo se convierte en:

```
def http_error(estado):
    estado del partido:
        caso 400:
            devolver "Solicitud incorrecta"
        caso 404:
            devolver "No encontrado"
        caso 418:
            volver "soy una tetera"
```

Sin el uso de _ en una declaración de caso, es posible que no exista una coincidencia. Si no existe ninguna coincidencia, el comportamiento es inactivo. Por ejemplo, si se pasa `status` de

500, se produce una no operación.

Patrones con un literal y una variable

Los patrones pueden verse como asignaciones de desempaquetado y se puede usar un patrón para vincular variables. En este ejemplo, un punto de datos se puede descomprimir en su coordenada xy coordenada y:

```
# punto es una tupla (x, y)
punto de partido:
    caso (0, 0):
        imprimir("Origen")
    caso (0, y):
        imprimir(f"Y={y}")
    caso (x, 0):
        imprimir(f"X={x}")
    caso (x, y):
        imprimir(f"X={x}, Y={y}")
    caso _:
        elevar ValueError ("No es un punto")
```

El primer patrón tiene dos literales, `(0, 0)`, y se puede considerar como una extensión del patrón literal que se muestra arriba. Los siguientes dos patrones combinan un literal y una variable, y la variable *binds* un valor del sujeto (`point`). El cuarto patrón captura dos valores, lo que lo hace conceptualmente similar a la asignación de desembalaje `(x, y) = point`.

Patrones y clases

Si está utilizando clases para estructurar sus datos, puede usar como patrón el nombre de la clase seguido de una lista de argumentos que se asemeja a un constructor. Este patrón tiene la capacidad de capturar atributos de instancia en variables:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

def location(point):
    match point:
        case Point(x=0, y=0):
            print("Origin is the point's location.")
        case Point(x=0, y=y):
            print(f"Y={y} and the point is on the y-axis
        case Point(x=x, y=0):
            print(f"X={x} and the point is on the x-axis
        case Point():
            print("The point is located somewhere else
        case _:
            print("Not a point")
```

Patrones con parámetros posicionales

Puede usar parámetros posicionales con algunas clases integradas que proporcionan un orden para sus atributos (por ejemplo, clases de datos). También puede definir una posición específica para atributos en patrones configurando el atributo especial `__match_args__` en sus clases. Si se establece en («x», «y»), los siguientes patrones son todos equivalentes (y todos vinculan el atributo `y` a la variable `var`):

```
Punto(1, var)
Punto(1, y=var)
Punto(x=1, y=var)
Punto(y=var, x=1)
```

Patrones anidados

Los patrones se pueden anidar arbitrariamente. Por ejemplo, si nuestros datos son una lista corta de puntos, podrían coincidir así:

```
puntos de partido:
    caso []:
        print("No hay puntos en la lista.")
    caso [Punto (0, 0)]:
        print("El origen es el único punto de la lista."
    caso [Punto (x, y)]:
        print(f"Un solo punto {x}, {y} está en la lista.
    caso [Punto (0, y1), Punto (0, y2)]:
        print(f"Dos puntos en el eje Y en {y1}, {y2} est
    caso _:
        print("Se encuentra algo más en la lista.")
```

Patrones complejos y el comodín

Hasta este punto, los ejemplos han utilizado `_` solo en la última declaración de caso. Se puede utilizar un comodín en patrones más complejos, como `('error', code, _)`. Por ejemplo:

```
match test_variable:
    case ('warning', code, 40):
        print("A warning has been received.")
    case ('error', code, _):
        print(f"An error {code} occurred.")
```

En el caso anterior, `test_variable` coincidirá con (“error”, código, 100) y (“error”, código, 800).

Guardia

Podemos agregar una cláusula `if` a un patrón, conocido como «guardia». Si la guardia es falsa, `match` pasa a probar el siguiente bloque de caso. Tenga en cuenta que la captura de valor ocurre antes de que se evalúe la guardia:

```
punto de partido:
    caso Punto(x, y) si x == y:
        print(f"El punto está ubicado en la diagonal Y=X
    caso Punto (x, y):
        print(f"El punto no está en la diagonal.")
```

Otras características clave

Varias otras características clave:

- Al igual que las asignaciones de desempaquetado, los patrones de tupla y lista tienen exactamente el mismo significado y en realidad coinciden con secuencias arbitrarias. Técnicamente, el tema debe ser una secuencia. Por lo tanto, una excepción importante es que los patrones no coinciden con los iteradores. Además, para evitar un error común, los patrones de secuencia no coinciden con las cadenas.

- Los patrones de secuencia admiten comodines: `[x, y, *rest]` y `(x, y, *rest)` funcionan de manera similar a los comodines en las asignaciones de desempaquetado. El nombre después de `*` también puede ser _, por lo que `(x, y, *_)` coincide con una secuencia de al menos dos elementos sin vincular los elementos restantes.

- Patrones de mapeo: `{"bandwidth": b, "latency": l}` captura los valores `"bandwidth"` y `"latency"` de un dict. A diferencia de los patrones de secuencia, las claves adicionales se ignoran. También se admite un comodín `**rest`. (Pero `**_` sería redundante, por lo que no está permitido).

- Los subpatrones se pueden capturar utilizando la palabra clave `as`:

  ```
  caso (Punto (x1, y1), Punto (x2, y2) como p2): ...
  ```

  Esto vincula x1, y1, x2, y2 como cabría esperar sin la cláusula `as` y p2 a todo el segundo elemento del tema.

- La mayoría de los literales se comparan por igualdad. Sin embargo, los singleton `True`, `False` y `None` se comparan por identidad.

- Las constantes con nombre se pueden usar en patrones. Estas constantes nombradas deben ser nombres con puntos para evitar que la constante se interprete como una variable de captura:

  ```
  desde enumeración importar enumeración
  Color de clase (Enumeración):
  ```

```
    ROJO = 0
    VERDE = 1
    AZUL = 2

color = Color.VERDE
color del partido:
    Color de la caja.ROJO:
        print("¡Veo rojo!")
    Color de la caja.VERDE:
        imprimir("La hierba es verde")
    Color de la caja.AZUL:
        print("Estoy sintiendo tristeza :()"
```

Para obtener la especificación completa, consulte **PEP 634** [https://peps.python.org/pep-0634/]. La motivación y el fundamento están en **PEP 635** [https://peps.python.org/pep-0635/], y un tutorial más largo está en **PEP 636** [https://peps.python.org/pep-0636/].

Opción opcional `EncodingWarning` y `encoding="locale"`

La codificación predeterminada de **`TextIOWrapper`** y **`open()`** depende de la plataforma y la configuración regional. Dado que UTF-8 se utiliza en la mayoría de las plataformas Unix, omitir la opción `encoding` al abrir archivos UTF-8 (por ejemplo, JSON, YAML, TOML, Markdown) es un error muy común. Por ejemplo:

```
# ERROR: se debe utilizar el modo "rb" o codificación="u
con open("data.json") como f:
    datos = json.cargar(f)
```

Para encontrar este tipo de error, se agrega un `EncodingWarning` opcional. Se emite cuando **`sys.flags.warn_default_encoding`** es verdadero y se utiliza la codificación predeterminada específica de la configuración regional.

Se agregan la opción `-X warn_default_encoding` y **`PYTHONWARNDEFAULTENCODING`** para habilitar la advertencia.

Consulte Codificación de texto para obtener más información.

Nuevas funciones relacionadas con las sugerencias de tipos

Esta sección cubre los cambios importantes que afectan a las sugerencias de tipo **PEP 484** [https://peps.python.org/pep-0484/] y al módulo `typing`.

PEP 604: Nuevo tipo de operador unión

Se introdujo un nuevo tipo de operador de unión que habilita la sintaxis `X | Y`. Esto proporciona una forma más clara de expresar «tipo X o tipo Y» en lugar de usar `typing.Union`, especialmente en sugerencias de tipo.

En versiones anteriores de Python, para aplicar una sugerencia de tipo para funciones que aceptan argumentos de múltiples tipos, se usaba `typing.Union`:

```
def cuadrado(número: Unión[int, float]) -> Unión[int, fl
    número de devolución ** 2
```

Las sugerencias de tipo ahora se pueden escribir de una manera más sucinta:

```
def cuadrado(número: int | float) -> int | flotar:
    número de devolución ** 2
```

Esta nueva sintaxis también se acepta como segundo argumento para `isinstance()` y `issubclass()`:

```
>>> isinstance(1, int | str)
True
```

Consulte Tipo de conversión y **PEP 604** [https://peps.python.org/pep-0604/] para obtener más detalles.

(Contribuido por Maggie Moss y Philippe Prados en bpo-41428 [https://bugs.python.org/issue?@action=redirect&bpo=41428], con adiciones de Yurii Karabas y Serhiy Storchaka en bpo-44490 [https://bugs.python.org/issue?@action=redirect&bpo=44490].)

PEP 612: Variables de especificación de parámetros

Se han agregado al módulo `typing` dos nuevas opciones para mejorar la información proporcionada a los verificadores de tipo estático para PEP 484 [https://peps.python.org/pep-0484/] "s `Callable`.

La primera es la variable de especificación de parámetros. Se utilizan para reenviar los tipos de parámetros de un invocable a otro invocable, un patrón que se encuentra comúnmente en funciones y decoradores de orden superior. Se pueden encontrar ejemplos de uso en `typing.ParamSpec`. Anteriormente, no había una manera fácil de escribir anotar la dependencia de los tipos de parámetros de una manera tan precisa.

La segunda opción es el nuevo operador `Concatenate`. Se usa junto con las variables de especificación de parámetros para escribir anotar un invocable de orden superior que agrega o elimina parámetros de otro invocable. Se pueden encontrar ejemplos de uso en `typing.Concatenate`.

Consulte `typing.Callable`, `typing.ParamSpec`, `typing.Concatenate`, `typing.ParamSpecArgs`, `typing.ParamSpecKwargs` y PEP 612 [https://peps.python.org/pep-0612/] para obtener más detalles.

(Contribuido por Ken Jin en bpo-41559 [https://bugs.python.org/issue?@action=redirect&bpo=41559], con pequeñas mejoras de Jelle Zijlstra en bpo-43783 [https://bugs.python.org/issue?@action=redirect&bpo=43783]. PEP escrito por Mark Mendoza.)

PEP 613: TypeAlias

PEP 484 [https://peps.python.org/pep-0484/] introdujo el concepto de alias de tipo, solo requiriendo que sean asignaciones no anotadas de nivel superior. Esta simplicidad a veces dificultaba que los verificadores de tipos distinguieran entre alias de tipos y asignaciones ordinarias, especialmente cuando se trataba de referencias directas o tipos no válidos. Comparar:

```
StrCache = 'Cache[str]' # un alias de tipo
LOG_PREFIX = 'LOG[DEBUG]' # una constante del módulo
```

Ahora el módulo `typing` tiene un valor especial `TypeAlias` que le permite declarar alias de tipo de forma más explícita:

```
StrCache: TypeAlias = 'Cache[str]' # un alias de tipo
LOG_PREFIX = 'LOG[DEBUG]' # una constante del módulo
```

Consulte **PEP 613** [https://peps.python.org/pep-0613/] para obtener más detalles.

(Contribuido por Mikhail Golubev en bpo-41923 [https://bugs.python.org/issue?@action=redirect&bpo=41923].)

PEP 647: protectores de tipo definidos por el usuario

Se agregó `TypeGuard` al módulo `typing` para anotar funciones de protección de tipos y mejorar la información proporcionada a los verificadores de tipos estáticos durante la reducción de tipos. Para obtener más información, consulte la documentación de `TypeGuard` y **PEP 647** [https://peps.python.org/pep-0647/].

(Contribuido por Ken Jin y Guido van Rossum en bpo-43766 [https://bugs.python.org/issue?@action=redirect&bpo=43766]. PEP escrito por Eric Traut.)

Otros cambios de idioma

- El tipo `int` tiene un nuevo método `int.bit_count()`, que devuelve el número de unos en la expansión binaria de un entero dado, también conocido como recuento de población. (Contribuido por Niklas Fiekas en bpo-29882 [https://bugs.python.org/issue?@action=redirect&bpo=29882].)
- Las vistas devueltas por `dict.keys()`, `dict.values()` y `dict.items()` ahora tienen todas un atributo `mapping` que proporciona un objeto `types.MappingProxyType` que envuelve el diccionario original. (Contribuido por Dennis

Sweeney en bpo-40890 [https://bugs.python.org/issue?@action=redirect&bpo=40890].)

- **PEP 618** [https://peps.python.org/pep-0618/]: La función `zip()` ahora tiene un indicador `strict` opcional, que se utiliza para requerir que todos los iterables tengan la misma longitud.
- Las funciones integradas y de extensión que toman argumentos enteros ya no aceptan `Decimal` s, `Fraction` sy otros objetos que se pueden convertir a números enteros solo con una pérdida (por ejemplo, que tienen el método `__int__()` pero no tienen el método `__index__()`). (Contribuido por Serhiy Storchaka en bpo-37999 [https://bugs.python.org/issue?@action=redirect&bpo=37999].)
- Si `object.__ipow__()` devuelve `NotImplemented`, el operador recurrirá correctamente a `object.__pow__()` y `object.__rpow__()` como se esperaba. (Aportado por Alex Shkop en bpo-38302 [https://bugs.python.org/issue?@action=redirect&bpo=38302].)
- Las expresiones de asignación ahora se pueden usar sin paréntesis dentro de literales de conjuntos y comprensiones de conjuntos, así como en índices de secuencia (pero no por sectores).
- Las funciones tienen un nuevo atributo `__builtins__` que se usa para buscar símbolos incorporados cuando se ejecuta una función, en lugar de buscar en `__globals__['__builtins__']`. El atributo se inicializa desde `__globals__["__builtins__"]` si existe, de lo contrario desde las incorporaciones actuales. (Contribuido por Mark Shannon en bpo-42990 [https://bugs.python.org/issue?@action=redirect&bpo=42990].)
- Se han agregado dos nuevas funciones integradas: `aiter()` y `anext()` para proporcionar contrapartes asíncronas a `iter()` y `next()`, respectivamente. (Contribuido por Joshua Bronson, Daniel Pope y Justin Wang en bpo-31861 [https://bugs.python.org/issue?@action=redirect&bpo=31861].)
- Static methods (`@staticmethod`) and class methods (`@classmethod`) now inherit the method attributes (`__module__`, `__name__`, `__qualname__`, `__doc__`, `__annotations__`) and have a new `__wrapped__` attribute. Moreover, static methods are now callable as

regular functions. (Contributed by Victor Stinner in bpo-43682 [https://bugs.python.org/issue?@action=redirect&bpo=43682].)

- Las anotaciones para objetivos complejos (todo junto a los objetivos `simple name` definidos por **PEP 526** [https://peps.python.org/pep-0526/]) ya no causan ningún efecto de tiempo de ejecución con `from __future__ import annotations`. (Contribuido por Batuhan Taskaya en bpo-42737 [https://bugs.python.org/issue?@action=redirect&bpo=42737].)
- Los objetos de clase y módulo ahora crean de forma perezosa anotaciones vacías dictados a pedido. Los dictados de anotaciones se almacenan en el `__dict__` del objeto para compatibilidad con versiones anteriores. Esto mejora las mejores prácticas para trabajar con `__annotations__`; para obtener más información, consulte Prácticas recomendadas para las anotaciones. (Contribuido por Larry Hastings en bpo-43901 [https://bugs.python.org/issue?@action=redirect&bpo=43901].)
- Las anotaciones consisten en `yield`, `yield from`, `await` o expresiones con nombre ahora están prohibidas bajo `from __future__ import annotations` debido a sus efectos secundarios. (Contribuido por Batuhan Taskaya en bpo-42725 [https://bugs.python.org/issue?@action=redirect&bpo=42725].)
- El uso de variables independientes, `super()` y otras expresiones que podrían alterar el procesamiento de la tabla de símbolos como anotaciones ahora no tienen efecto bajo `from __future__ import annotations`. (Contribuido por Batuhan Taskaya en bpo-42725 [https://bugs.python.org/issue?@action=redirect&bpo=42725].)
- Los valores hash de NaN tanto del tipo **`float`** como del tipo **`decimal.Decimal`** ahora dependen de la identidad del objeto. Anteriormente, siempre tenían hash en `0` aunque los valores de NaN no son iguales entre sí. Esto provocó un comportamiento de tiempo de ejecución potencialmente cuadrático debido a colisiones de hash excesivas al crear diccionarios y conjuntos que contienen varios NaN. (Contribuido por Raymond Hettinger en bpo-43475 [https://bugs.python.org/issue?@action=redirect&bpo=43475].)
- Se generará un **`SyntaxError`** (en lugar de un **`NameError`**)

al eliminar la constante `__debug__`. (Aportado por Donghee Na en bpo-45000 [https://bugs.python.org/issue?@action=redirect&bpo=45000].)

- Las excepciones `SyntaxError` ahora tienen atributos `end_lineno` y `end_offset`. Serán `None` si no se determinan. (Contribuido por Pablo Galindo en bpo-43914 [https://bugs.python.org/issue?@action=redirect&bpo=43914].)

Nuevos módulos

- Ninguno todavía.

Módulos mejorados

asyncio

Agregue el método `connect_accepted_socket()` que falta. (Aportado por Alex Grönholm en bpo-41332 [https://bugs.python.org/issue?@action=redirect&bpo=41332].)

argumentar

La frase engañosa «argumentos opcionales» fue reemplazada por «opciones» en la ayuda de argparse. Algunas pruebas pueden requerir una adaptación si se basan en la coincidencia exacta de la salida. (Contribuido por Raymond Hettinger en bpo-9694 [https://bugs.python.org/issue?@action=redirect&bpo=9694].)

formación

El método `index()` de `array.array` ahora tiene parámetros *start* and *stop* opcionales. (Contribuido por Anders Lorentsen y Zackery Spytz en bpo-31956 [https://bugs.python.org/issue?@action=redirect&bpo=31956].)

asynchat, asyncore, smtpd

Estos módulos se han marcado como obsoletos en la documentación del módulo desde Python 3.6. Ahora se ha agregado un `DeprecationWarning` en tiempo de importación a estos tres módulos.

base64

Agregue `base64.b32hexencode()` y `base64.b32hexdecode()` para admitir la codificación Base32 con alfabeto hexadecimal extendido.

bdb

Agregue **`clearBreakpoints()`** para restablecer todos los puntos de interrupción establecidos. (Aportado por Irit Katriel en bpo-24160 [https://bugs.python.org/issue?@action=redirect&bpo=24160].)

bisecar

Se agregó la posibilidad de proporcionar una función *key* a las API en el módulo `bisect`. (Contribuido por Raymond Hettinger en bpo-4356 [https://bugs.python.org/issue?@action=redirect&bpo=4356].)

códecs

Agregue una función `codecs.unregister()` para anular el registro de una función de búsqueda de códec. (Contribuido por Hai Shi en bpo-41842 [https://bugs.python.org/issue?@action=redirect&bpo=41842].)

colecciones.abc

El `__args__` del parameterized generic para `collections.abc.Callable` ahora es consistente con `typing.Callable`. **`collections.abc.Callable`** genérico ahora aplana los parámetros de tipo, similar a lo que hace actualmente **`typing.Callable`**. Esto significa que `collections.abc.Callable[[int, str], str]` tendrá `__args__` de `(int, str, str)`; anteriormente esto era `([int, str], str)`. Para permitir este cambio, `types.GenericAlias` ahora puede ser subclasificado, y se devolverá una subclase al subíndice el tipo **`collections.abc.Callable`**. Tenga en cuenta que se puede generar un `TypeError` para formas no válidas de parametrizar **`collections.abc.Callable`** que pueden haber pasado silenciosamente en Python 3.9. (Contribuido por Ken Jin en bpo-42195 [https://bugs.python.org/issue?@action=redirect&bpo=42195].)

contextlib

Agregue un administrador de contexto `contextlib.aclosing()` para cerrar de forma segura los generadores asíncronos y los objetos que representan recursos liberados de manera asíncrona. (Contribuido por Joongi Kim y John Belmonte en bpo-41229 [https://bugs.python.org/issue?@action=redirect&bpo=41229].)

Agregue soporte de administrador de contexto asincrónico a `contextlib.nullcontext()`. (Contribuido por Tom Gringauz en bpo-41543 [https://bugs.python.org/issue?@action=redirect&bpo=41543].)

Agregue `AsyncContextDecorator` para admitir el uso de administradores de contexto asíncronos como decoradores.

maldiciones

Las funciones de color extendidas agregadas en ncurses 6.1 serán utilizadas de forma transparente por `curses.color_content()`, `curses.init_color()`, `curses.init_pair()` y `curses.pair_content()`. Una nueva función, `curses.has_extended_color_support()`, indica si la biblioteca ncurses subyacente proporciona compatibilidad de color ampliada. (Contribuido por Jeffrey Kintscher y Hans Petter Jansson en bpo-36982 [https://bugs.python.org/issue?@action=redirect&bpo=36982].)

Las constantes `BUTTON5_*` ahora se exponen en el módulo `curses` si las proporciona la biblioteca de curses subyacente. (Contribuido por Zackery Spytz en bpo-39273 [https://bugs.python.org/issue?@action=redirect&bpo=39273].)

clases de datos

slots

Se agregó el parámetro `slots` en el decorador **`dataclasses.dataclass()`**. (Contribuido por Yurii Karabas en bpo-42269 [https://bugs.python.org/issue?@action=redirect&bpo=42269])

Campos solo de palabras clave

dataclasses ahora admite campos que son solo palabras clave en el método _init_ generado. Hay varias formas de especificar campos de solo palabras clave.

Puede decir que todos los campos son solo palabras clave:

```
desde clases de datos importar clase de datos

@dataclass(kw_only=Verdadero)
Cumpleaños de clase:
    nombre: cadena
    cumpleaños: fechahora.fecha
```

Tanto `name` como `birthday` son parámetros de solo palabras clave para el método _init_ generado.

Puede especificar solo palabras clave por campo:

```
desde clases de datos importar clase de datos, campo

@clasededatos
Cumpleaños de clase:
    nombre: cadena
    cumpleaños: fechahora.fecha = campo(kw_only=True)
```

Aquí solo `birthday` es solo palabra clave. Si configura `kw_only` en campos individuales, tenga en cuenta que existen reglas sobre el reordenamiento de los campos debido a que los campos de solo palabras clave deben seguir campos que no son solo de palabras clave. Consulte la documentación completa de clases de datos para obtener más detalles.

También puede especificar que todos los campos que siguen a un marcador KW_ONLY sean solo de palabras clave. Este será probablemente el uso más común:

```
desde clases de datos importar clase de datos, KW_ONLY

@clasededatos
Punto de clase:
    x: flotar
    y: flotar
    _: KW_ONLY
    z: flotante = 0,0
    t: flotante = 0,0
```

Aquí, `z` y `t` son parámetros de solo palabras clave, mientras que `x` y `y` no lo son. (Aportado por Eric V. Smith en bpo-43532 [https://bugs.python.org/issue?@action=redirect&bpo=43532].)

distutils

Todo el paquete `distutils` está obsoleto y se eliminará en Python 3.12. Su funcionalidad para especificar compilaciones de paquetes ya ha sido completamente reemplazada por paquetes de terceros `setuptools` y `packaging`, y la mayoría de las otras API de uso común están disponibles en otras partes de la biblioteca estándar (como **`platform`**, **`shutil`**, **`subprocess`** o **`sysconfig`**). No hay planes para migrar ninguna otra funcionalidad de `distutils`, y las aplicaciones que utilizan otras funciones deben planificar la realización de copias privadas del código. Consulte **PEP 632** [https://peps.python.org/pep-0632/] para obtener más información.

Se eliminó el comando `bdist_wininst` en desuso en Python 3.8. Ahora se recomienda el comando `bdist_wheel` para distribuir paquetes binarios en Windows. (Contribuido por Victor Stinner en bpo-42802 [https://bugs.python.org/issue?@action=redirect&bpo=42802].)

doctest

Cuando un módulo no define `__loader__`, recurre a `__spec__.loader`. (Contribuido por Brett Cannon en bpo-42133 [https://bugs.python.org/issue?@action=redirect&bpo=42133].)

codificaciones

`encodings.normalize_encoding()` ahora ignora los caracteres que no son ASCII. (Contribuido por Hai Shi en bpo-39337 [https://bugs.python.org/issue?@action=redirect&bpo=39337].)

enum

`Enum` **`__repr__()`** ahora devuelve `enum_name.member_name` y **`__str__()`** ahora devuelve `member_name`. Las enumeraciones Stdlib disponibles como constantes de módulo tienen un **`repr()`** de `module_name.member_name`. (Aportado por Ethan Furman en bpo-40066 [https://bugs.python.org/issue?@action=redirect&bpo=40066].)

Añadir **`enum.StrEnum`** para los enums en los que todos los miembros son cadenas. (Contribuido por Ethan Furman en bpo-41816 [https://bugs.python.org/issue?@action=redirect&bpo=41816].)

entrada de archivo

Agregue los parámetros *encoding* and *errors* en **`fileinput.input()`** y **`fileinput.FileInput`**. (Contribuido por Inada Naoki en bpo-43712 [https://bugs.python.org/issue?@action=redirect&bpo=43712].)

`fileinput.hook_compressed()` ahora devuelve el objeto **`TextIOWrapper`** cuando *mode* es «r» y el archivo está comprimido, como archivos sin comprimir. (Aportado por Inada Naoki en bpo-5758 [https://bugs.python.org/issue?@action=redirect&bpo=5758].)

manipulador de faltas

El módulo **`faulthandler`** ahora detecta si ocurre un error fatal durante la recolección de un recolector de basura. (Contribuido por Victor Stinner en bpo-44466 [https://bugs.python.org/issue?@action=redirect&bpo=44466].)

gc

Agregue ganchos de auditoría para `gc.get_objects()`, `gc.get_referrers()` y `gc.get_referents()`. (Contribuido por Pablo Galindo en bpo-43439 [https://bugs.python.org/issue?@action=redirect&bpo=43439].)

glob

Agregue los parámetros *root_dir* and *dir_fd* en `glob()` y `iglob()` que permiten especificar el directorio raíz para la búsqueda. (Contribuido por Serhiy Storchaka en bpo-38144 [https://bugs.python.org/issue?@action=redirect&bpo=38144].)

hashlib

El módulo hashlib requiere OpenSSL 1.1.1 o más reciente. (Contribuido por Christian Heimes en **PEP 644** [https://peps.python.org/pep-0644/] y bpo-43669 [https://bugs.python.org/issue?@action=redirect&bpo=43669].)

El módulo hashlib tiene soporte preliminar para OpenSSL 3.0.0. (Contribuido por Christian Heimes en bpo-38820 [https://bugs.python.org/issue?@action=redirect&bpo=38820] y otros números).

El respaldo de Python puro de `pbkdf2_hmac()` está en desuso. En el futuro, PBKDF2-HMAC solo estará disponible cuando Python se haya construido con soporte OpenSSL. (Contribuido por Christian Heimes en bpo-43880 [https://bugs.python.org/issue?@action=redirect&bpo=43880].)

hmac

El módulo hmac ahora usa la implementación HMAC de OpenSSL internamente. (Contribuido por Christian Heimes en bpo-40645 [https://bugs.python.org/issue?@action=redirect&bpo=40645].)

IDLE e idlelib

Hacer que IDLE invoque `sys.excepthook()` (cuando se inicia sin

"-n"). Los ganchos de usuario se ignoraban anteriormente. (Aportado por Ken Hilton en bpo-43008 [https://bugs.python.org/issue?@action=redirect&bpo=43008].)

Reorganizar el diálogo de configuración. Dividir la pestaña General en las pestañas Windows y Shell/Ed. Mover las fuentes de ayuda, que amplían el menú Ayuda, a la pestaña Extensiones. Hacer espacio para nuevas opciones y acortar el diálogo. Esto último hace que el diálogo se adapte mejor a las pantallas pequeñas. (Contribuido por Terry Jan Reedy en bpo-40468 [https://bugs.python.org/issue?@action=redirect&bpo=40468].) Mover la configuración del espacio de indentación de la pestaña de Fuente a la nueva pestaña Windows. (Contribuido por Mark Roseman y Terry Jan Reedy en bpo-33962 [https://bugs.python.org/issue?@action=redirect&bpo=33962].)

Los cambios anteriores se trasladaron a una versión de mantenimiento 3.9.

Agrega una barra lateral de Shell. Mueva el indicador principal ("> > >") a la barra lateral. Agregue mensajes secundarios ("…") a la barra lateral. El clic izquierdo y el arrastre opcional seleccionan una o más líneas de texto, como con la barra lateral del número de línea del editor. Al hacer clic derecho después de seleccionar líneas de texto, se muestra un menú contextual con "copiar con indicaciones". Esto comprime los mensajes de la barra lateral con líneas del texto seleccionado. Esta opción también aparece en el menú contextual del texto. (Contribuido por Tal Einat en bpo-37903 [https://bugs.python.org/issue?@action=redirect&bpo=37903].)

Use espacios en lugar de tabulaciones para sangrar el código interactivo. Esto hace que las entradas de código interactivo "se vean bien". Hacer esto factible fue una de las principales motivaciones para agregar la barra lateral de shell. (Aportado por Terry Jan Reedy en bpo-37892 [https://bugs.python.org/issue?@action=redirect&bpo=37892].)

Resalte los nuevos soft keywords **`match`**, **`case`** y **`_`** en declaraciones de coincidencia de patrones. Sin embargo, este resaltado no es perfecto y será incorrecto en algunos casos excepcionales, incluidos algunos `_`-s en patrones `case`. (Aportado por Tal Einat en bpo-44010 [https://bugs.python.org/issue?@action=redirect&bpo=44010].)

Nuevo en la versión de mantenimiento 3.10.

Aplicar el resaltado de sintaxis a los archivos `.pyi`. (Contribución de Alex Waygood y Terry Jan Reedy en bpo-45447 [https://bugs.python.org/issue?@action=redirect&bpo=45447].)

Incluir avisos al guardar Shell con entradas y salidas. (Contribuido por Terry Jan Reedy en gh-95191 [https://github.com/python/cpython/issues/95191].)

importlib.metadata

Paridad de características con `importlib_metadata` 4.6 (history [https://importlib-metadata.readthedocs.io/en/latest/history.html]).

importlib.metadata entry points ahora brinda una mejor experiencia para seleccionar puntos de entrada por grupo y nombre a través de una nueva clase importlib.metadata.EntryPoints. Consulte la Nota de compatibilidad en los documentos para obtener más información sobre la desactivación y el uso.

Se agregó importlib.metadata.packages_distributions() para resolver módulos y paquetes de Python de nivel superior en su importlib.metadata.Distribution.

inspeccionar

Cuando un módulo no define `__loader__`, recurre a `__spec__.loader`. (Contribuido por Brett Cannon en bpo-42133 [https://bugs.python.org/issue?@action=redirect&bpo=42133].)

Agregue **`inspect.get_annotations()`**, que calcula de forma segura las anotaciones definidas en un objeto. Evita las peculiaridades de acceder a las anotaciones de varios tipos de objetos y hace muy pocas suposiciones sobre el objeto que examina. **`inspect.get_annotations()`** también puede eliminar correctamente las anotaciones en cadena. **`inspect.get_annotations()`** ahora se considera la mejor práctica para acceder al dictado de anotaciones definido en cualquier objeto de Python; Para obtener más información sobre las mejores prácticas para trabajar con anotaciones, consulte Prácticas recomendadas para las anotaciones. De manera relacionada, `inspect.signature()`, `inspect.Signature.from_callable()` y `inspect.Signature.from_function()` ahora llaman a `inspect.get_annotations()` para recuperar anotaciones. Esto significa que `inspect.signature()` y `inspect.Signature.from_callable()` ahora también pueden eliminar las anotaciones en cadena. (Aportado por Larry Hastings en bpo-43817 [https://bugs.python.org/issue?@action=redirect&bpo=43817].)

itertools

Agregue `itertools.pairwise()`. (Aportado por Raymond Hettinger en bpo-38200 [https://bugs.python.org/issue?@action=redirect&bpo=38200].)

caché de línea

Cuando un módulo no define `__loader__`, recurre a `__spec__.loader`. (Contribuido por Brett Cannon en bpo-42133 [https://bugs.python.org/issue?@action=redirect&bpo=42133].)

os

Agregue soporte `os.cpu_count()` para VxWorks RTOS. (Aportado por Peixing Xin en bpo-41440 [https://bugs.python.org/issue?@action=redirect&bpo=41440].)

Agregue una nueva función `os.eventfd()` y ayudantes relacionados para envolver el syscall `eventfd2` en Linux. (Contribuido por Christian Heimes en bpo-41001 [https://bugs.python.org/issue?@action=redirect&bpo=41001].)

Agregue `os.splice()` que permite mover datos entre dos descriptores de archivos sin copiar entre el espacio de direcciones del kernel y el espacio de direcciones del usuario, donde uno de los descriptores de archivos debe hacer referencia a una tubería.

(Aportado por Pablo Galindo en bpo-41625 [https://bugs.python.org/issue?@action=redirect&bpo=41625].)

Agregue `O_EVTONLY`, `O_FSYNC`, `O_SYMLINK` y `O_NOFOLLOW_ANY` para macOS. (Aportado por Donghee Na en bpo-43106 [https://bugs.python.org/issue?@action=redirect&bpo=43106].)

os.path

`os.path.realpath()` ahora acepta un argumento de solo palabra clave *strict*. Cuando se establece en `True`, `OSError` se genera si no existe una ruta o se encuentra un bucle de enlace simbólico. (Contribuido por Barney Gale en bpo-43757 [https://bugs.python.org/issue?@action=redirect&bpo=43757].)

pathlib

Agregue soporte de división a `PurePath.parents`. (Aportado por Joshua Cannon en bpo-35498 [https://bugs.python.org/issue?@action=redirect&bpo=35498].)

Agregue soporte de indexación negativa a `PurePath.parents`. (Aportado por Yaroslav Pankovych en bpo-21041 [https://bugs.python.org/issue?@action=redirect&bpo=21041].)

Agregue el método `Path.hardlink_to` que reemplaza a `link_to()`. El nuevo método tiene el mismo orden de argumentos que `symlink_to()`. (Aportado por Barney Gale en bpo-39950 [https://bugs.python.org/issue?@action=redirect&bpo=39950].)

`pathlib.Path.stat()` y `chmod()` ahora aceptan un argumento de solo palabra clave *follow_symlinks* para mantener la coherencia con las funciones correspondientes en el módulo `os`. (Contribuido por Barney Gale en bpo-39906 [https://bugs.python.org/issue?@action=redirect&bpo=39906].)

plataforma

Agregue `platform.freedesktop_os_release()` para recuperar la identificación del sistema operativo del archivo estándar freedesktop.org os-release [https://www.freedesktop.org/software/systemd/man/os-release.html]. (Aportado por Christian Heimes en bpo-28468 [https://bugs.python.org/issue?@action=redirect&bpo=28468].)

pprint

`pprint.pprint()` ahora acepta un nuevo argumento de palabra clave `underscore_numbers`. (Contribuido por sblondon en bpo-42914 [https://bugs.python.org/issue?@action=redirect&bpo=42914].)

`pprint` ahora puede imprimir de forma bonita instancias de **`dataclasses.dataclass`**. (Contribuido por Lewis Gaul en bpo-43080 [https://bugs.python.org/issue?@action=redirect&bpo=43080].)

py_compile

Agregue la opción `--quiet` a la interfaz de línea de comandos de **`py_compile`**. (Contribuido por Gregory Schevchenko en bpo-38731 [https://bugs.python.org/issue?@action=redirect&bpo=38731].)

pyclbr

Agregue un atributo `end_lineno` a los objetos `Function` y `Class` en el árbol devuelto por **`pyclbr.readmodule()`** y **`pyclbr.readmodule_ex()`**. Coincide con el `lineno` existente (inicio). (Aportado por Aviral Srivastava en bpo-38307 [https://bugs.python.org/issue?@action=redirect&bpo=38307].)

dejar de lado

El módulo **`shelve`** ahora usa **`pickle.DEFAULT_PROTOCOL`** de forma predeterminada en lugar del protocolo **`pickle`** `3` al crear estantes. (Aportado por Zackery Spytz en bpo-34204 [https://bugs.python.org/issue?@action=redirect&bpo=34204].)

estadísticas

Agregue las funciones **`covariance()`**, **`correlation()`** de Pearson y **`linear_regression()`** simple. (Contribuido por Tymoteusz Wołodźko en bpo-38490 [https://bugs.python.org/issue?

@action=redirect&bpo=38490].)

sitio

Cuando un módulo no define `__loader__`, recurre a `__spec__.loader`. (Contribuido por Brett Cannon en bpo-42133 [https://bugs.python.org/issue?@action=redirect&bpo=42133].)

enchufe

La excepción **`socket.timeout`** ahora es un alias de **`TimeoutError`**. (Contribuido por Christian Heimes en bpo-42413 [https://bugs.python.org/issue?@action=redirect&bpo=42413].)

Agregue la opción para crear sockets MPTCP con `IPPROTO_MPTCP` (Contribuido por Rui Cunha en bpo-43571 [https://bugs.python.org/issue?@action=redirect&bpo=43571].)

Agregue la opción `IP_RECVTOS` para recibir el tipo de servicio (ToS) o los campos DSCP / ECN (Contribuido por Georg Sauthoff en bpo-44077 [https://bugs.python.org/issue?@action=redirect&bpo=44077]).

ssl

El módulo ssl requiere OpenSSL 1.1.1 o más reciente. (Contribuido por Christian Heimes en **PEP 644** [https://peps.python.org/pep-0644/] y bpo-43669 [https://bugs.python.org/issue?@action=redirect&bpo=43669].)

El módulo SSL tiene soporte preliminar para OpenSSL 3.0.0 y la nueva opción **`OP_IGNORE_UNEXPECTED_EOF`**. (Aportado por Christian Heimes en bpo-38820 [https://bugs.python.org/issue?@action=redirect&bpo=38820], bpo-43794 [https://bugs.python.org/issue?@action=redirect&bpo=43794], bpo-43788 [https://bugs.python.org/issue?@action=redirect&bpo=43788], bpo-43791 [https://bugs.python.org/issue?@action=redirect&bpo=43791], bpo-43799 [https://bugs.python.org/issue?@action=redirect&bpo=43799], bpo-43920 [https://bugs.python.org/issue?@action=redirect&bpo=43920], bpo-43789 [https://bugs.python.org/issue?@action=redirect&bpo=43789] y bpo-43811 [https://bugs.python.org/issue?@action=redirect&bpo=43811]).

La función obsoleta y el uso de constantes obsoletas ahora dan como resultado un `DeprecationWarning`. `ssl.SSLContext.options` tiene `OP_NO_SSLv2` y `OP_NO_SSLv3` configurados de forma predeterminada y, por lo tanto, no puede advertir sobre la configuración de la bandera nuevamente. El deprecation section tiene una lista de funciones obsoletas. (Contribuido por Christian Heimes en bpo-43880 [https://bugs.python.org/issue?@action=redirect&bpo=43880].)

El módulo ssl ahora tiene una configuración predeterminada más segura. Los cifrados sin confidencialidad directa o SHA-1 MAC están deshabilitados de forma predeterminada. El nivel de seguridad 2 prohíbe claves RSA, DH y ECC débiles con menos de 112 bits de seguridad. `SSLContext` tiene por defecto la versión mínima del protocolo TLS 1.2. Los ajustes se basan en la investigación de Hynek Schlawack. (Contribuido por Christian Heimes en bpo-43998 [https://bugs.python.org/issue?@action=redirect&bpo=43998].)

Los protocolos obsoletos SSL 3.0, TLS 1.0 y TLS 1.1 ya no son compatibles oficialmente. Python no los bloquea activamente. Sin embargo, las opciones de compilación de OpenSSL, las configuraciones de distribución, los parches de proveedores y los conjuntos de cifrado pueden impedir un protocolo de enlace exitoso.

Agregue un parámetro *timeout* a la función `ssl.get_server_certificate()`. (Contribuido por Zackery Spytz en bpo-31870 [https://bugs.python.org/issue?@action=redirect&bpo=31870].)

El módulo ssl utiliza tipos de pila e inicialización multifase. (Contribuido por Christian Heimes en bpo-42333 [https://bugs.python.org/issue?@action=redirect&bpo=42333].)

Se agregó una nueva bandera de verificación `VERIFY_X509_PARTIAL_CHAIN`. (Aportado por l0x en bpo-40849 [https://bugs.python.org/issue?@action=redirect&bpo=40849].)

sqlite3

Agregue eventos de auditoría para `connect()`, `enable_load_extension()` y `load_extension()`. (Aportado por Erlend E. Aasland en bpo-43762 [https://bugs.python.org/issue?@action=redirect&bpo=43762].)

sys

Agregar atributo `sys.orig_argv`: la lista de los argumentos originales de la línea de comando pasados al ejecutable de Python. (Contribuido por Victor Stinner en bpo-23427 [https://bugs.python.org/issue?@action=redirect&bpo=23427].)

Agregue `sys.stdlib_module_names`, que contiene la lista de nombres de módulos de biblioteca estándar. (Contribuido por Victor Stinner en bpo-42955 [https://bugs.python.org/issue?@action=redirect&bpo=42955].)

_hilo

`_thread.interrupt_main()` ahora toma un número de señal opcional para simular (el valor predeterminado sigue siendo `signal.SIGINT`). (Aportado por Antoine Pitrou en bpo-43356 [https://bugs.python.org/issue?@action=redirect&bpo=43356].)

enhebrar

Agregue `threading.gettrace()` y `threading.getprofile()` para recuperar las funciones establecidas por `threading.settrace()` y `threading.setprofile()` respectivamente. (Contribuido por Mario Corchero en bpo-42251 [https://bugs.python.org/issue?@action=redirect&bpo=42251].)

Agregue `threading.__excepthook__` para permitir recuperar el valor original de `threading.excepthook()` en caso de que esté configurado en un valor roto o diferente. (Contribuido por Mario Corchero en bpo-42308 [https://bugs.python.org/issue?@action=redirect&bpo=42308].)

rastrear

Las funciones **`format_exception()`**, **`format_exception_only()`** y **`print_exception()`** ahora pueden tomar un objeto de excepción como un argumento solo posicional. (Contribuido por Zackery Spytz y Matthias Bussonnier en bpo-26389 [https://bugs.python.org/issue?@action=redirect&bpo=26389].)

tipos

Reintroduzca las clases **`types.EllipsisType`**, **`types.NoneType`** y **`types.NotImplementedType`**, proporcionando un nuevo conjunto de tipos fácilmente interpretables por los verificadores de tipos. (Contribuido por Bas van Beek en bpo-41810 [https://bugs.python.org/issue?@action=redirect&bpo=41810].)

mecanografía

Para conocer los cambios importantes, consulte Nuevas funciones relacionadas con las sugerencias de tipos.

El comportamiento de **`typing.Literal`** se modificó para cumplir con **PEP 586** [https://peps.python.org/pep-0586/] y para coincidir con el comportamiento de los verificadores de tipo estático especificados en el PEP.

1. `Literal` ahora elimina los parámetros duplicados.

2. Las comparaciones de igualdad entre objetos `Literal` ahora son independientes del orden.

3. Las comparaciones `Literal` ahora respetan los tipos. Por ejemplo, `Literal[0] == Literal[False]` se evaluó previamente como `True`. Ahora es `False`. Para admitir este cambio, la memoria caché de tipos utilizada internamente ahora admite tipos diferenciados.

4. Los objetos `Literal` ahora generarán una excepción **`TypeError`** durante las comparaciones de igualdad si alguno de sus parámetros no es hashable. Tenga en cuenta que declarar `Literal` con parámetros que no se pueden aplicar hash no arrojará un error:

```
>>> from typing import Literal
>>> Literal[{0}]
>>> Literal[{0}] == Literal[{False}]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'set'
```

(Contribuido por Yurii Karabas en bpo-42345 [https://bugs.python.org/issue?@action=redirect&bpo=42345].)

Agregue la nueva función **`typing.is_typeddict()`** a la introspección si una anotación es **`typing.TypedDict`**. (Contribuido por Patrick Reader en bpo-41792 [https://bugs.python.org/issue?@action=redirect&bpo=41792].)

Las subclases de `typing.Protocol` que solo tienen variables de datos declaradas ahora generarán un `TypeError` cuando se verifiquen con `isinstance` a menos que estén decoradas con **`runtime_checkable()`**. Anteriormente, estos controles se realizaban en silencio. Los usuarios deben decorar sus subclases con el decorador **`runtime_checkable()`** si quieren protocolos de tiempo de ejecución. (Aportado por Yurii Karabas en bpo-38908 [https://bugs.python.org/issue?@action=redirect&bpo=38908].)

La importación desde los submódulos `typing.io` y `typing.re` ahora emitirá **`DeprecationWarning`**. Estos submódulos quedaron obsoletos desde Python 3.8 y se eliminarán en una versión futura de Python. Todo lo que pertenezca a esos submódulos debe importarse directamente desde **`typing`**. (Aportado por Sebastian Rittau en bpo-38291 [https://bugs.python.org/issue?@action=redirect&bpo=38291].)

prueba de unidad

Agregue un nuevo método **`assertNoLogs()`** para complementar el **`assertLogs()`** existente. (Contribuido por Kit Yan Choi en bpo-39385 [https://bugs.python.org/issue?@action=redirect&bpo=39385].)

urllib.parse

Las versiones de Python anteriores a Python 3.10 permitían usar `;`

y `&` como separadores de parámetros de consulta en **`urllib.parse.parse_qs()`** y **`urllib.parse.parse_qsl()`**. Debido a preocupaciones de seguridad y para cumplir con las recomendaciones más recientes del W3C, esto se ha cambiado para permitir solo una única clave separadora, con `&` como valor predeterminado. Este cambio también afecta a **`cgi.parse()`** y **`cgi.parse_multipart()`** ya que utilizan las funciones afectadas internamente. Para obtener más detalles, consulte su documentación respectiva. (Contribución de Adam Goldschmidt, Senthil Kumaran y Ken Jin en bpo-42967 [https://bugs.python.org/issue?@action=redirect&bpo=42967]).

La presencia de caracteres de nueva línea o tabulación en partes de una URL permite algunas formas de ataques. Siguiendo la especificación WHATWG que actualiza **RFC 3986** [https://datatracker.ietf.org/doc/html/rfc3986.html], el analizador en **`urllib.parse`** elimina de la URL los caracteres de nueva línea ASCII `\n`, `\r` y tab `\t` para evitar tales ataques. Los caracteres de eliminación están controlados por una nueva variable de nivel de módulo `urllib.parse._UNSAFE_URL_BYTES_TO_REMOVE`. (Ver gh-88048 [https://github.com/python/cpython/issues/88048])

xml

Agregue una clase **`LexicalHandler`** al módulo **`xml.sax.handler`**. (Contribuido por Jonathan Gossage y Zackery Spytz en bpo-35018 [https://bugs.python.org/issue?@action=redirect&bpo=35018].)

zipimport

Agregue métodos relacionados con **PEP 451** [https://peps.python.org/pep-0451/]: **`find_spec()`**, **`zipimport.zipimporter.create_module()`** y **`zipimport.zipimporter.exec_module()`**. (Contribuido por Brett Cannon en bpo-42131 [https://bugs.python.org/issue?@action=redirect&bpo=42131].)

Agregue el método **`invalidate_caches()`**. (Contribuido por Desmond Cheong en bpo-14678 [https://bugs.python.org/issue?

@action=redirect&bpo=14678].)

Optimizaciones

- Los constructores **`str()`**, **`bytes()`** y **`bytearray()`** ahora son más rápidos (alrededor del 30-40% para objetos pequeños). (Contribuido por Serhiy Storchaka en bpo-41334 [https://bugs.python.org/issue?@action=redirect&bpo=41334].)
- El módulo **`runpy`** ahora importa menos módulos. El tiempo de inicio del comando `python3 -m module-name` es 1,4 veces más rápido en promedio. En Linux, `python3 -I -m module-name` importa 69 módulos en Python 3.9, mientras que solo importa 51 módulos (-18) en Python 3.10. (Contribuido por Victor Stinner en bpo-41006 [https://bugs.python.org/issue?@action=redirect&bpo=41006] y bpo-41718 [https://bugs.python.org/issue?@action=redirect&bpo=41718].)
- La instrucción `LOAD_ATTR` ahora usa un nuevo mecanismo «por caché de código de operación». Es aproximadamente un 36% más rápido ahora para los atributos regulares y un 44% más rápido para las tragamonedas. (Contribuido por Pablo Galindo y Yury Selivanov en bpo-42093 [https://bugs.python.org/issue?@action=redirect&bpo=42093] y Guido van Rossum en bpo-42927 [https://bugs.python.org/issue?@action=redirect&bpo=42927], basado en ideas implementadas originalmente en PyPy y MicroPython).
- Al compilar Python con **`--enable-optimizations`**, ahora `-fno-semantic-interposition` se agrega tanto a la línea de compilación como a la de enlace. Esto acelera las compilaciones del intérprete de Python creado con **`--enable-shared`** con `gcc` hasta en un 30%. Consulte this article [https://developers.redhat.com/blog/2020/06/25/red-hat-enterprise-linux-8-2-brings-faster-python-3-8-run-speeds/] para obtener más detalles. (Contribuido por Victor Stinner y Pablo Galindo en bpo-38980 [https://bugs.python.org/issue?@action=redirect&bpo=38980].)
- Utilice un nuevo código de gestión del búfer de salida para los módulos **`bz2`** / **`lzma`** / **`zlib`** y agregue la función `.readall()` a la clase `_compression.DecompressReader`. La descompresión de

bz2 ahora es 1.09x ~ 1.17x más rápida, la descompresión de lzma 1.20x ~ 1.32x más rápida, `GzipFile.read(-1)` 1.11x ~ 1.18x más rápida. (Contribuido por Ma Lin, revisado por Gregory P. Smith, en bpo-41486 [https://bugs.python.org/issue?@action=redirect&bpo=41486])

- Cuando se utilizan anotaciones en cadena, los dictados de anotaciones para funciones ya no se crean cuando se crea la función. En su lugar, se almacenan como una tupla de cadenas, y el objeto de función convierte esto de forma perezosa en el dictado de anotaciones a pedido. Esta optimización reduce a la mitad el tiempo de CPU necesario para definir una función anotada. (Aportado por Yurii Karabas e Inada Naoki en bpo-42202 [https://bugs.python.org/issue?@action=redirect&bpo=42202].)
- Las funciones de búsqueda de subcadenas como `str1 in str2` y `str2.find(str1)` ahora utilizan a veces el algoritmo de búsqueda de cadenas «bidireccional» de Crochemore & Perrin para evitar el comportamiento cuadrático en cadenas largas. (Contribuido por Dennis Sweeney en bpo-41972 [https://bugs.python.org/issue?@action=redirect&bpo=41972])
- Agregue microoptimizaciones a `_PyType_Lookup()` para mejorar el rendimiento de búsqueda de caché de atributos de tipo en el caso común de coincidencias de caché. Esto hace que el intérprete sea 1,04 veces más rápido de media. (Aportado por Dino Viehland en bpo-43452 [https://bugs.python.org/issue?@action=redirect&bpo=43452].)
- Las siguientes funciones integradas ahora admiten la convención de llamada de vectorcall **PEP 590** [https://peps.python.org/pep-0590/] más rápida: **`map()`**, **`filter()`**, **`reversed()`**, **`bool()`** y **`float()`**. (Aportado por Donghee Na y Jeroen Demeyer en bpo-43575 [https://bugs.python.org/issue?@action=redirect&bpo=43575], bpo-43287 [https://bugs.python.org/issue?@action=redirect&bpo=43287], bpo-41922 [https://bugs.python.org/issue?@action=redirect&bpo=41922], bpo-41873 [https://bugs.python.org/issue?@action=redirect&bpo=41873] y bpo-41870 [https://bugs.python.org/issue?@action=redirect&bpo=41870].)
- El rendimiento de **`BZ2File`** se mejora eliminando el `RLock` interno. Esto hace que el subproceso **`BZ2File`** sea inseguro

frente a múltiples lectores o escritores simultáneos, tal como siempre lo han sido sus clases equivalentes en **`gzip`** y **`lzma`**. (Aportado por Inada Naoki en bpo-43785 [https://bugs.python.org/issue?@action=redirect&bpo=43785].)

Obsoleto

- Actualmente, Python acepta literales numéricos seguidos inmediatamente de palabras clave, por ejemplo, `0in x`, `1or x`, `0if 1else 2`. Permite expresiones confusas y ambiguas como `[0x1for x in y]` (que puede interpretarse como `[0x1 for x in y]` o `[0x1f or x in y]`). A partir de esta versión, se genera una advertencia de desuso si el literal numérico va seguido inmediatamente por una de las palabras clave **`and`**, **`else`**, **`for`**, **`if`**, **`in`**, **`is`** y **`or`**. En versiones futuras, se cambiará a advertencia de sintaxis y, finalmente, a error de sintaxis. (Aportado por Serhiy Storchaka en bpo-43833 [https://bugs.python.org/issue?@action=redirect&bpo=43833].)

- A partir de esta versión, habrá un esfuerzo concertado para comenzar a limpiar la semántica de importación antigua que se mantuvo para la compatibilidad con Python 2.7. Específicamente, **`find_loader()`**/**`find_module()`** (reemplazado por **`find_spec()`**), **`load_module()`** (reemplazado por **`exec_module()`**), **`module_repr()`** (que el sistema de importación se encarga de usted), el atributo `__package__` (reemplazado por `__spec__.parent`), el atributo `__loader__` (reemplazado por `__spec__.loader`) y el atributo `__cached__` (reemplazado por `__spec__.cached`) eliminarse lentamente (así como otras clases y métodos en **`importlib`**). **`ImportWarning`** y/o **`DeprecationWarning`** se generarán según corresponda para ayudar a identificar el código que necesita actualización durante esta transición.

- Todo el espacio de nombres `distutils` está obsoleto y se eliminará en Python 3.12. Consulte la sección module changes para obtener más información.

- Los argumentos que no son enteros para `random.randrange()` están en desuso. El `ValueError` está en desuso en favor de un `TypeError`. (Contribuido por Serhiy Storchaka y Raymond Hettinger en bpo-37319 [https://bugs.python.org/issue?@action=redirect&bpo=37319].)

- Los diversos métodos `load_module()` de `importlib` se han documentado como obsoletos desde Python 3.6, pero ahora también activarán un `DeprecationWarning`. Utilice `exec_module()` en su lugar. (Contribuido por Brett Cannon en bpo-26131 [https://bugs.python.org/issue?@action=redirect&bpo=26131].)

- **`zimport.zipimporter.load_module()`** ha quedado obsoleto y se prefiere `exec_module()`. (Aportado por Brett Cannon en bpo-26131 [https://bugs.python.org/issue?@action=redirect&bpo=26131].)

- El uso de `load_module()` por parte del sistema de importación ahora activa un `ImportWarning` ya que se prefiere `exec_module()`. (Contribuido por Brett Cannon en bpo-26131 [https://bugs.python.org/issue?@action=redirect&bpo=26131].)

- El uso de **`importlib.abc.MetaPathFinder.find_module()`** y **`importlib.abc.PathEntryFinder.find_module()`** por parte del sistema de importación ahora activa un `ImportWarning`, ya que se prefieren `importlib.abc.MetaPathFinder.find_spec()` y `importlib.abc.PathEntryFinder.find_spec()`, respectivamente. Puede utilizar `importlib.util.spec_from_loader()` para ayudar en la migración. (Aportado por Brett Cannon en bpo-42134 [https://bugs.python.org/issue?@action=redirect&bpo=42134].)

- El uso de **`importlib.abc.PathEntryFinder.find_loader()`** por parte del sistema de importación ahora activa un `ImportWarning`, ya que se prefiere `importlib.abc.PathEntryFinder.find_spec()`.

Puede utilizar `importlib.util.spec_from_loader()` para ayudar en la migración. (Aportado por Brett Cannon en bpo-43672 [https://bugs.python.org/issue?@action=redirect&bpo=43672].)

- Las diversas implementaciones de `importlib.abc.MetaPathFinder.find_module()` (`importlib.machinery.BuiltinImporter.find_module()`, `importlib.machinery.FrozenImporter.find_module()`, `importlib.machinery.WindowsRegistryFinder.find_modu` `importlib.machinery.PathFinder.find_module()`, `importlib.abc.MetaPathFinder.find_module()`), `importlib.abc.PathEntryFinder.find_module()` (`importlib.machinery.FileFinder.find_module()`) y `importlib.abc.PathEntryFinder.find_loader()` (`importlib.machinery.FileFinder.find_loader()`) ahora generan `DeprecationWarning` y están programadas para su eliminación en Python 3.12 (anteriormente estaban documentadas como obsoletas en Python 3.4). (Aportado por Brett Cannon en bpo-42135 [https://bugs.python.org/issue?@action=redirect&bpo=42135].)

- `importlib.abc.Finder` está en desuso (incluido su único método, `find_module()`). Tanto `importlib.abc.MetaPathFinder` como `importlib.abc.PathEntryFinder` ya no heredan de la clase. En su lugar, los usuarios deben heredar de una de estas dos clases según corresponda. (Aportado por Brett Cannon en bpo-42135 [https://bugs.python.org/issue?@action=redirect&bpo=42135].)

- Las obsolescencias de `imp`, `importlib.find_loader()`, `importlib.util.set_package_wrapper()`, `importlib.util.set_loader_wrapper()`, `importlib.util.module_for_loader()`, `pkgutil.ImpImporter` y `pkgutil.ImpLoader` se han actualizado para incluir Python 3.12 como la versión programada de eliminación (comenzaron a generar `DeprecationWarning` en versiones anteriores de Python). (Aportado por Brett Cannon en bpo-43720 [https://

bugs.python.org/issue?@action=redirect&bpo=43720].)

- El sistema de importación ahora utiliza el atributo `__spec__` en los módulos antes de recurrir a **`module_repr()`** para el método `__repr__()` de un módulo. La eliminación del uso de `module_repr()` está prevista para Python 3.12. (Aportado por Brett Cannon en bpo-42137 [https://bugs.python.org/issue?@action=redirect&bpo=42137].)

- **`importlib.abc.Loader.module_repr()`**, **`importlib.machinery.FrozenLoader.module_repr()`** y **`importlib.machinery.BuiltinLoader.module_repr()`** están obsoletos y su eliminación está prevista en Python 3.12. (Aportado por Brett Cannon en bpo-42136 [https://bugs.python.org/issue?@action=redirect&bpo=42136].)

- `sqlite3.OptimizedUnicode` ha sido indocumentado y obsoleto desde Python 3.3, cuando se convirtió en un alias para **`str`**. Ahora está en desuso, programado para su eliminación en Python 3.12. (Contribuido por Erlend E. Aasland en bpo-42264 [https://bugs.python.org/issue?@action=redirect&bpo=42264].)

- La función incorporada no documentada `sqlite3.enable_shared_cache` ahora está en desuso, programada para su eliminación en Python 3.12. Su uso está fuertemente desaconsejado por la documentación de SQLite3. Consulte the SQLite3 docs [https://sqlite.org/c3ref/enable_shared_cache.html] para obtener más detalles. Si se debe usar una caché compartida, abra la base de datos en modo URI usando el parámetro de consulta `cache=shared`. (Contribuido por Erlend E. Aasland en bpo-24464 [https://bugs.python.org/issue?@action=redirect&bpo=24464].)

- Los siguientes métodos `threading` ahora están en desuso:

    - `threading.currentThread` => **`threading.current_thread()`**
    - `threading.activeCount` =>

    `threading.active_count()`
  - `threading.Condition.notifyAll` => `threading.Condition.notify_all()`
  - `threading.Event.isSet` => `threading.Event.is_set()`
  - `threading.Thread.setName` => `threading.Thread.name`
  - `threading.thread.getName` => `threading.Thread.name`
  - `threading.Thread.isDaemon` => `threading.Thread.daemon`
  - `threading.Thread.setDaemon` => `threading.Thread.daemon`

  (Aportado por Jelle Zijlstra en gh-87889 [https://github.com/python/cpython/issues/87889].)

- **`pathlib.Path.link_to()`** está en desuso y está previsto su eliminación en Python 3.12. Utilice `pathlib.Path.hardlink_to()` en su lugar. (Aportado por Barney Gale en bpo-39950 [https://bugs.python.org/issue?@action=redirect&bpo=39950].)

- `cgi.log()` está obsoleto y está programado para su eliminación en Python 3.12. (Contribuido por Inada Naoki en bpo-41139 [https://bugs.python.org/issue?@action=redirect&bpo=41139].)

- Las siguientes funciones de `ssl` han quedado obsoletas desde Python 3.6, Python 3.7 u OpenSSL 1.1.0 y se eliminarán en 3.11:

  - **`OP_NO_SSLv2`**, **`OP_NO_SSLv3`**, **`OP_NO_TLSv1`**, **`OP_NO_TLSv1_1`**, **`OP_NO_TLSv1_2`** y **`OP_NO_TLSv1_3`** se reemplazan por `minimum_version` y `maximum_version`.
  - **`PROTOCOL_SSLv2`**, **`PROTOCOL_SSLv3`**, **`PROTOCOL_SSLv23`**, **`PROTOCOL_TLSv1`**, **`PROTOCOL_TLSv1_1`**, **`PROTOCOL_TLSv1_2`** y **`PROTOCOL_TLS`** están en desuso en favor de `PROTOCOL_TLS_CLIENT` y `PROTOCOL_TLS_SERVER`

  - `wrap_socket()` es reemplazado por `ssl.SSLContext.wrap_socket()`
  - `match_hostname()`
  - `RAND_pseudo_bytes()`, `RAND_egd()`
  - Las funciones NPN como `ssl.SSLSocket.selected_npn_protocol()` y `ssl.SSLContext.set_npn_protocols()` son reemplazadas por ALPN.

- La depuración de subprocesos (variable de entorno **PYTHONTHREADDEBUG**) está obsoleta en Python 3.10 y se eliminará en Python 3.12. Esta característica requiere un debug build of Python. (Aportado por Victor Stinner en bpo-44584 [https://bugs.python.org/issue?@action=redirect&bpo=44584].)

- La importación desde los submódulos `typing.io` y `typing.re` ahora emitirá `DeprecationWarning`. Estos submódulos se eliminarán en una versión futura de Python. Todo lo que pertenezca a estos submódulos debe importarse directamente desde `typing`. (Aportado por Sebastian Rittau en bpo-38291 [https://bugs.python.org/issue?@action=redirect&bpo=38291].)

Eliminado

- Se eliminaron los métodos especiales `__int__`, `__float__`, `__floordiv__`, `__mod__`, `__divmod__`, `__rfloordiv__`, `__rmod__` y `__rdivmod__` de la clase `complex`. Siempre levantaron un `TypeError`. (Contribuido por Serhiy Storchaka en bpo-41974 [https://bugs.python.org/issue?@action=redirect&bpo=41974].)

- Se ha eliminado el método `ParserBase.error()` del módulo `_markupbase` privado e indocumentado. `html.parser.HTMLParser` es la única subclase de `ParserBase` y su implementación `error()` ya se eliminó en Python 3.5. (Contribuido por Berker Peksag en bpo-31844 [https://bugs.python.org/issue?@action=redirect&bpo=31844].)

- Se eliminó el atributo `unicodedata.ucnhash_CAPI` que era un objeto interno de PyCapsule. La estructura `_PyUnicode_Name_CAPI` privada relacionada se movió a la API C interna. (Contribuido por Victor Stinner en bpo-42157 [https://bugs.python.org/issue?@action=redirect&bpo=42157].)

- Se eliminó el módulo `parser`, que quedó obsoleto en 3.9 debido al cambio al nuevo analizador PEG, así como todos los archivos fuente y de encabezado C que solo usaba el analizador anterior, incluidos `node.h`, `parser.h`, `graminit.h` y `grammar.h`.

- Se eliminaron las funciones de la API pública de C `PyParser_SimpleParseStringFlags`, `PyParser_SimpleParseStringFlagsFilename`, `PyParser_SimpleParseFileFlags` y `PyNode_Compile` que estaban en desuso en 3.9 debido al cambio al nuevo analizador PEG.

- Se eliminó el módulo `formatter`, que quedó obsoleto en Python 3.4. Está algo obsoleto, poco utilizado y no probado. Originalmente estaba programado para eliminarse en Python 3.6, pero dichas eliminaciones se retrasaron hasta después del EOL de Python 2.7. Los usuarios existentes deben copiar las clases que utilicen en su código. (Contribución de Donghee Na y Terry J. Reedy en bpo-42299 [https://bugs.python.org/issue?@action=redirect&bpo=42299]).

- Se eliminó la función **`PyModule_GetWarningsModule()`** que ahora era inútil debido a que el módulo **`_warnings`** se convirtió en un módulo integrado en 2.6. (Aportado por Hai Shi en bpo-42599 [https://bugs.python.org/issue?@action=redirect&bpo=42599].)

- Elimine los alias obsoletos a Colecciones clases base abstractas del módulo **`collections`**. (Contribuido por Victor Stinner en bpo-37324 [https://bugs.python.org/issue?@action=redirect&bpo=37324].)

- El parámetro `loop` se ha eliminado de la mayoría de **`asyncio`**“s API de alto nivel después de la desaprobación en

Python 3.8. La motivación detrás de este cambio es múltiple:

1. Esto simplifica la API de alto nivel.
2. Las funciones en la API de alto nivel han obtenido implícitamente el bucle de eventos en ejecución del hilo actual desde Python 3.7. No es necesario pasar el bucle de eventos a la API en la mayoría de los casos de uso normales.
3. El paso de bucles de eventos es propenso a errores, especialmente cuando se trata de bucles que se ejecutan en diferentes subprocesos.

Tenga en cuenta que la API de bajo nivel seguirá aceptando `loop`. Consulte Cambios en la API de Python para ver ejemplos de cómo reemplazar el código existente.

(Contribuido por Yurii Karabas, Andrew Svetlov, Yury Selivanov y Kyle Stanley en bpo-42392 [https://bugs.python.org/issue?@action=redirect&bpo=42392].)

Portar a Python 3.10

Esta sección enumera los cambios descritos anteriormente y otras correcciones de errores que pueden requerir cambios en su código.

Cambios en la sintaxis de Python

- Ahora se emite una advertencia de desaprobación cuando se compila una sintaxis previamente válida si el literal numérico va seguido inmediatamente de una palabra clave (como en `0in x`). En versiones futuras, se cambiará a advertencia de sintaxis y, finalmente, a error de sintaxis. Para deshacerse de la advertencia y hacer que el código sea compatible con versiones futuras, simplemente agregue un espacio entre el literal numérico y la siguiente palabra clave. (Aportado por Serhiy Storchaka en bpo-43833 [https://bugs.python.org/issue?@action=redirect&bpo=43833].)

Cambios en la API de Python

- Los parámetros *etype* de las funciones **`format_exception()`**, **`format_exception_only()`**, y **`print_exception()`** en el módulo **`traceback`** han sido renombradas a *exc*. (Contribuido por Zackery Spytz y Matthias Bussonnier en bpo-26389 [https://bugs.python.org/issue?@action=redirect&bpo=26389].)

- **`atexit`**: en la salida de Python, si falla una devolución de llamada registrada con **`atexit.register()`**, ahora se registra su excepción. Anteriormente, solo se registraban algunas excepciones y la última excepción siempre se ignoraba en silencio. (Contribuido por Victor Stinner en bpo-42639 [https://bugs.python.org/issue?@action=redirect&bpo=42639].)

- **`collections.abc.Callable`** genérico ahora aplana los parámetros de tipo, similar a lo que hace actualmente **`typing.Callable`**. Esto significa que `collections.abc.Callable[[int, str], str]` tendrá `__args__` de `(int, str, str)`; anteriormente esto era `([int, str], str)`. El código que accede a los argumentos a través de **`typing.get_args()`** o `__args__` debe tener en cuenta este cambio. Además, **`TypeError`** se puede generar para formas no válidas de parametrizar **`collections.abc.Callable`** que pueden haber pasado silenciosamente en Python 3.9. (Contribuido por Ken Jin en bpo-42195 [https://bugs.python.org/issue?@action=redirect&bpo=42195].)

- **`socket.htons()`** y **`socket.ntohs()`** ahora generan **`OverflowError`** en lugar de **`DeprecationWarning`** si el parámetro dado no cabe en un entero sin signo de 16 bits. (Contribuido por Erlend E. Aasland en bpo-42393 [https://bugs.python.org/issue?@action=redirect&bpo=42393].)

- El parámetro `loop` se ha eliminado de la mayoría de **`asyncio`**“s API de alto nivel después de la desaprobación en Python 3.8.

  Una corrutina que actualmente se ve así:

```
asíncrono def foo (bucle):
    espere asyncio.sleep (1, bucle = bucle)
```

Debería ser reemplazado por esto:

```
asíncrono def foo():
    espera asyncio.sleep(1)
```

Si `foo()` fue diseñado específicamente *not* para ejecutarse en el bucle de eventos en ejecución del hilo actual (por ejemplo, ejecutándose en el bucle de eventos de otro hilo), considere usar **`asyncio.run_coroutine_threadsafe()`** en su lugar.

(Contribuido por Yurii Karabas, Andrew Svetlov, Yury Selivanov y Kyle Stanley en bpo-42392 [https://bugs.python.org/issue?@action=redirect&bpo=42392].)

- El constructor **`types.FunctionType`** ahora hereda las incorporaciones actuales si el diccionario *globals* no tiene clave `"__builtins__"`, en lugar de usar `{"None": None}` como incorporaciones: el mismo comportamiento que las funciones **`eval()`** y **`exec()`**. La definición de una función con `def function(...): ...` en Python no se ve afectada, los globales no se pueden anular con esta sintaxis: también hereda las incorporaciones actuales. (Contribuido por Victor Stinner en bpo-42990 [https://bugs.python.org/issue?@action=redirect&bpo=42990].)

Cambios en la API de C

- Las funciones de API C `PyParser_SimpleParseStringFlags`, `PyParser_SimpleParseStringFlagsFilename`, `PyParser_SimpleParseFileFlags`, `PyNode_Compile` y el tipo utilizado por estas funciones, `struct _node`, se eliminaron debido al cambio al nuevo analizador PEG.

  La fuente debe compilarse ahora directamente en un objeto de código utilizando, por ejemplo, **`Py_CompileString()`**. A continuación, el objeto de código resultante se puede evaluar utilizando, por ejemplo, `PyEval_EvalCode()`.

Específicamente:

- Una llamada a `PyParser_SimpleParseStringFlags` seguida de `PyNode_Compile` se puede reemplazar llamando a `Py_CompileString()`.

- No hay reemplazo directo para `PyParser_SimpleParseFileFlags`. Para compilar código a partir de un argumento `FILE *`, deberá leer el archivo en C y pasar el búfer resultante a `Py_CompileString()`.

- Para compilar un archivo con un nombre de archivo `char *`, abra explícitamente el archivo, léalo y compile el resultado. Una forma de hacerlo es utilizando el módulo `io` con `PyImport_ImportModule()`, `PyObject_CallMethod()`, `PyBytes_AsString()` y `Py_CompileString()`, como se muestra a continuación. (Se omiten las declaraciones y el manejo de errores).

  ```
  io_module = Import_ImportModule("io");
  objetoarchivo = PyObject_CallMethod(io_module,
  source_bytes_object = PyObject_CallMethod(objet
  resultado = PyObject_CallMethod (objeto de arch
  fuente_buf = PyBytes_AsString(fuente_bytes_obje
  código = Py_CompileString(source_buf, nombre de
  ```

- Para los objetos `FrameObject`, el miembro `f_lasti` ahora representa un desplazamiento de código de palabra en lugar de un desplazamiento simple en la cadena de código de bytes. Esto significa que este número debe multiplicarse por 2 para usarse con API que esperan un desplazamiento de bytes (como `PyCode_Addr2Line()`, por ejemplo). Tenga en cuenta también que el miembro **`f_lasti`** de los objetos `FrameObject` no se considera estable: utilice

`PyFrame_GetLineNumber()` en su lugar.

Cambios en el código de bytes de CPython

- La instrucción `MAKE_FUNCTION` ahora acepta un dict o una tupla de cadenas como anotaciones de la función. (Aportado por Yurii Karabas e Inada Naoki en bpo-42202 [https://bugs.python.org/issue?@action=redirect&bpo=42202].)

Construir cambios

- **PEP 644** [https://peps.python.org/pep-0644/]: Python ahora requiere OpenSSL 1.1.1 o más reciente. OpenSSL 1.0.2 ya no es compatible. (Contribuido por Christian Heimes en bpo-43669 [https://bugs.python.org/issue?@action=redirect&bpo=43669].)
- Las funciones C99 `snprintf()` y `vsnprintf()` ahora son necesarias para construir Python. (Contribuido por Victor Stinner en bpo-36020 [https://bugs.python.org/issue?@action=redirect&bpo=36020].)
- `sqlite3` requiere SQLite 3.7.15 o superior. (Aportado por Sergey Fedoseev y Erlend E. Aasland en bpo-40744 [https://bugs.python.org/issue?@action=redirect&bpo=40744] y bpo-40810 [https://bugs.python.org/issue?@action=redirect&bpo=40810]).
- El módulo `atexit` ahora debe construirse siempre como un módulo integrado. (Contribuido por Victor Stinner en bpo-42639 [https://bugs.python.org/issue?@action=redirect&bpo=42639].)
- Agregue la opción `--disable-test-modules` al script `configure`: no cree ni instale módulos de prueba. (Contribuido por Xavier de Gaye, Thomas Petazzoni y Peixing Xin en bpo-27640 [https://bugs.python.org/issue?@action=redirect&bpo=27640].)
- Agregue `--with-wheel-pkg-dir=PATH option` al script `./configure`. Si se especifica, el módulo `ensurepip` busca

paquetes de ruedas `setuptools` y `pip` en este directorio: si ambos están presentes, estos paquetes de ruedas se utilizan en lugar de los paquetes de ruedas asegurados.

Algunas políticas de empaquetado de distribución de Linux recomiendan no empaquetar dependencias. Por ejemplo, Fedora instala paquetes de rueda en el directorio `/usr/share/python-wheels/` y no instala el paquete `ensurepip._bundled`.

(Contribuido por Victor Stinner en bpo-42856 [https://bugs.python.org/issue?@action=redirect&bpo=42856].)

- Agregue un nuevo **`configure --without-static-libpython option`** para no construir la biblioteca estática `libpythonMAJOR.MINOR.a` y no instalar el archivo de objeto `python.o`.

  (Contribuido por Victor Stinner en bpo-43103 [https://bugs.python.org/issue?@action=redirect&bpo=43103].)

- El script `configure` ahora usa la utilidad `pkg-config`, si está disponible, para detectar la ubicación de los encabezados y bibliotecas Tcl/Tk. Como antes, esas ubicaciones se pueden especificar explícitamente con las opciones de configuración `--with-tcltk-includes` y `--with-tcltk-libs`. (Aportado por Manolis Stamatogiannakis en bpo-42603 [https://bugs.python.org/issue?@action=redirect&bpo=42603].)

- Agregue la opción **`--with-openssl-rpath`** al script `configure`. La opción simplifica la construcción de Python con una instalación personalizada de OpenSSL, p. Ej. `./configure --with-openssl=/path/to/openssl --with-openssl-rpath=auto`. (Contribuido por Christian Heimes en bpo-43466 [https://bugs.python.org/issue?@action=redirect&bpo=43466].)

Cambios en la API de C

PEP 652: Mantenimiento del ABI estable

La ABI estable (interfaz binaria de aplicación) para módulos de extensión o incrustación de Python ahora está definida explícitamente. Estabilidad de la API en C describe las garantías de estabilidad C API y ABI junto con las mejores prácticas para usar la ABI estable.

(Contribuido por Petr Viktorin en **PEP 652** [https://peps.python.org/pep-0652/] y bpo-43795 [https://bugs.python.org/issue?@action=redirect&bpo=43795].)

Nuevas características

- El resultado de `PyNumber_Index()` ahora siempre tiene el tipo exacto `int`. Anteriormente, el resultado podría haber sido una instancia de una subclase de `int`. (Contribuido por Serhiy Storchaka en bpo-40792 [https://bugs.python.org/issue?@action=redirect&bpo=40792].)
- Agregue un nuevo miembro `orig_argv` a la estructura `PyConfig`: la lista de los argumentos originales de la línea de comandos pasados al ejecutable de Python. (Contribuido por Victor Stinner en bpo-23427 [https://bugs.python.org/issue?@action=redirect&bpo=23427].)
- Se han agregado las macros `PyDateTime_DATE_GET_TZINFO()` y `PyDateTime_TIME_GET_TZINFO()` para acceder a los atributos `tzinfo` de los objetos `datetime.datetime` y `datetime.time`. (Contribuido por Zackery Spytz en bpo-30155 [https://bugs.python.org/issue?@action=redirect&bpo=30155].)
- Agregue una función `PyCodec_Unregister()` para anular el registro de una función de búsqueda de códec. (Contribuido por Hai Shi en bpo-41842 [https://bugs.python.org/issue?@action=redirect&bpo=41842].)
- Se agregó la función `PyIter_Send()` para permitir el envío de valor al iterador sin generar la excepción `StopIteration`. (Contribuido por Vladimir Matveev en bpo-41756 [https://bugs.python.org/issue?

@action=redirect&bpo=41756].)

- Agregue **`PyUnicode_AsUTF8AndSize()`** a la API C limitada. (Contribuido por Alex Gaynor en bpo-41784 [https://bugs.python.org/issue?@action=redirect&bpo=41784].)

- Agregue la función **`PyModule_AddObjectRef()`**: similar a **`PyModule_AddObject()`** pero no robe una referencia al valor en caso de éxito. (Contribuido por Victor Stinner en bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741].)

- Agregue las funciones **`Py_NewRef()`** y **`Py_XNewRef()`** para incrementar el recuento de referencia de un objeto y devolver el objeto. (Contribuido por Victor Stinner en bpo-42262 [https://bugs.python.org/issue?@action=redirect&bpo=42262].)

- Las funciones **`PyType_FromSpecWithBases()`** y **`PyType_FromModuleAndSpec()`** ahora aceptan una sola clase como argumento *bases*. (Contribuido por Serhiy Storchaka en bpo-42423 [https://bugs.python.org/issue?@action=redirect&bpo=42423].)

- La función **`PyType_FromModuleAndSpec()`** ahora acepta la ranura NULL `tp_doc`. (Contribuido por Hai Shi en bpo-41832 [https://bugs.python.org/issue?@action=redirect&bpo=41832].)

- La función **`PyType_GetSlot()`** puede aceptar static types. (Contribuido por Hai Shi y Petr Viktorin en bpo-41073 [https://bugs.python.org/issue?@action=redirect&bpo=41073].)

- Agregue una nueva función **`PySet_CheckExact()`** a la C-API para verificar si un objeto es una instancia de **`set`** pero no una instancia de un subtipo. (Contribuido por Pablo Galindo en bpo-43277 [https://bugs.python.org/issue?@action=redirect&bpo=43277].)

- Agregue **`PyErr_SetInterruptEx()`** que permite pasar un número de señal para simular. (Contribuido por Antoine

Pitrou en bpo-43356 [https://bugs.python.org/issue?@action=redirect&bpo=43356].)

- La API C limitada ahora es compatible si Python is built in debug mode (si se define la macro `Py_DEBUG`). En la API C limitada, las funciones **`Py_INCREF()`** y **`Py_DECREF()`** ahora se implementan como llamadas de función opacas, en lugar de acceder directamente al miembro **`PyObject.ob_refcnt`**, si Python está construido en modo de depuración y la macro `Py_LIMITED_API` apunta a Python 3.10 o más reciente. Se hizo posible admitir la API C limitada en modo de depuración porque la estructura **`PyObject`** es la misma en el modo de liberación y depuración desde Python 3.8 (ver bpo-36465 [https://bugs.python.org/issue?@action=redirect&bpo=36465]).

  La API C limitada todavía no es compatible con la compilación especial **`--with-trace-refs`** (macro `Py_TRACE_REFS`). (Contribuido por Victor Stinner en bpo-43688 [https://bugs.python.org/issue?@action=redirect&bpo=43688].)

- Agregue la función **`Py_Is(x, y)`** para probar si el objeto *x* object is the *y*, lo mismo que `x is y` en Python. Agregue también las funciones **`Py_IsNone()`**, **`Py_IsTrue()`**, **`Py_IsFalse()`** para probar si un objeto es, respectivamente, el singleton `None`, el singleton `True` o el singleton `False`. (Contribuido por Victor Stinner en bpo-43753 [https://bugs.python.org/issue?@action=redirect&bpo=43753].)

- Agregue nuevas funciones para controlar el recolector de basura desde el código C: **`PyGC_Enable()`**, **`PyGC_Disable()`**, **`PyGC_IsEnabled()`**. Estas funciones permiten activar, desactivar y consultar el estado del recolector de basura desde código C sin tener que importar el módulo **`gc`**.

- Agregue un nuevo indicador de tipo **`Py_TPFLAGS_DISALLOW_INSTANTIATION`** para no permitir la creación de instancias de tipo. (Aportado por Victor Stinner en bpo-43916 [https://bugs.python.org/issue?

@action=redirect&bpo=43916].)

- Agregue un nuevo indicador de tipo **`Py_TPFLAGS_IMMUTABLETYPE`** para crear objetos de tipo inmutable: los atributos de tipo no se pueden establecer ni eliminar. (Aportado por Victor Stinner y Erlend E. Aasland en bpo-43908 [https://bugs.python.org/issue?@action=redirect&bpo=43908].)

Portar a Python 3.10

- La macro `PY_SSIZE_T_CLEAN` ahora debe definirse para usar los formatos **`PyArg_ParseTuple()`** y **`Py_BuildValue()`** que usan `#`: `es#`, `et#`, `s#`, `u#`, `y#`, `z#`, `U#` y `Z#`. Consulte Analizando argumentos y construyendo valores y **PEP 353** [https://peps.python.org/pep-0353/]. (Aportado por Victor Stinner en bpo-40943 [https://bugs.python.org/issue?@action=redirect&bpo=40943].)

- Dado que **`Py_REFCNT()`** se cambia a la función estática en línea, `Py_REFCNT(obj) = new_refcnt` debe reemplazarse con `Py_SET_REFCNT(obj, new_refcnt)`: consulte **`Py_SET_REFCNT()`** (disponible desde Python 3.9). Para compatibilidad con versiones anteriores, esta macro se puede utilizar:

  ```
  #si PY_VERSION_HEX < 0x030900A4
  # define Py_SET_REFCNT(obj, refcnt) ((Py_REFCNT(obj
  #endif
  ```

  (Contribuido por Victor Stinner en bpo-39573 [https://bugs.python.org/issue?@action=redirect&bpo=39573].)

- Se había permitido llamar a **`PyDict_GetItem()`** sin GIL retenido por motivos históricos. Ya no está permitido. (Contribuido por Victor Stinner en bpo-40839 [https://bugs.python.org/issue?@action=redirect&bpo=40839].)

- `PyUnicode_FromUnicode(NULL, size)` y `PyUnicode_FromStringAndSize(NULL, size)` generan `DeprecationWarning` ahora. Utilice **`PyUnicode_New()`**

para asignar un objeto Unicode sin datos iniciales. (Contribuido por Inada Naoki en bpo-36346 [https://bugs.python.org/issue?@action=redirect&bpo=36346].)

- La estructura `_PyUnicode_Name_CAPI` privada de la API `unicodedata.ucnhash_CAPI` de PyCapsule se ha movido a la API C interna. (Contribuido por Victor Stinner en bpo-42157 [https://bugs.python.org/issue?@action=redirect&bpo=42157].)

- Las funciones **`Py_GetPath()`**, **`Py_GetPrefix()`**, **`Py_GetExecPrefix()`**, **`Py_GetProgramFullPath()`**, **`Py_GetPythonHome()`** y **`Py_GetProgramName()`** ahora devuelven `NULL` si se llama antes de **`Py_Initialize()`** (antes de que se inicialice Python). Utilice la nueva API Configuración de inicialización de Python para obtener el Configuración de la ruta de Python. (Aportado por Victor Stinner en bpo-42260 [https://bugs.python.org/issue?@action=redirect&bpo=42260].)

- Las macros **`PyList_SET_ITEM()`**, **`PyTuple_SET_ITEM()`** y **`PyCell_SET()`** ya no se pueden utilizar como valor l o valor r. Por ejemplo, `x = PyList_SET_ITEM(a, b, c)` y `PyList_SET_ITEM(a, b, c) = x` ahora fallan con un error del compilador. Previene errores como la prueba `if (PyList_SET_ITEM (a, b, c) < 0) ...`. (Contribuido por Zackery Spytz y Victor Stinner en bpo-30459 [https://bugs.python.org/issue?@action=redirect&bpo=30459].)

- Los archivos de API no limitados `odictobject.h`, `parser_interface.h`, `picklebufobject.h`, `pyarena.h`, `pyctype.h`, `pydebug.h`, `pyfpe.h` y `pytime.h` se han movido al directorio `Include/cpython`. Estos archivos no deben incluirse directamente, ya que ya están incluidos en `Python.h`; ver Archivos de cabecera (Include). Si se han incluido directamente, considere incluir `Python.h` en su lugar. (Aportado por Nicholas Sim en bpo-35134 [https://bugs.python.org/issue?@action=redirect&bpo=35134].)

- Utilice el indicador de tipo **`Py_TPFLAGS_IMMUTABLETYPE`** para crear objetos de tipo inmutable. No confíe en **`Py_TPFLAGS_HEAPTYPE`** para decidir si un objeto de tipo es mutable o no; compruebe si **`Py_TPFLAGS_IMMUTABLETYPE`** está configurado en su lugar. (Aportado por Victor Stinner y Erlend E. Aasland en bpo-43908 [https://bugs.python.org/issue?@action=redirect&bpo=43908].)

- La función no documentada `Py_FrozenMain` se eliminó de la API limitada. La función es principalmente útil para compilaciones personalizadas de Python. (Aportado por Petr Viktorin en bpo-26241 [https://bugs.python.org/issue?@action=redirect&bpo=26241].)

Obsoleto

- La función `PyUnicode_InternImmortal()` ahora está en desuso y se eliminará en Python 3.12: use **`PyUnicode_InternInPlace()`** en su lugar. (Contribuido por Victor Stinner en bpo-41692 [https://bugs.python.org/issue?@action=redirect&bpo=41692].)

Eliminado

- Se eliminaron las funciones `Py_UNICODE_str*` que manipulaban cadenas `Py_UNICODE*`. (Contribuido por Inada Naoki en bpo-41123 [https://bugs.python.org/issue?@action=redirect&bpo=41123].)
    - `Py_UNICODE_strlen`: utilice **`PyUnicode_GetLength()`** o **`PyUnicode_GET_LENGTH`**
    - `Py_UNICODE_strcat`: utilice **`PyUnicode_CopyCharacters()`** o **`PyUnicode_FromFormat()`**
    - `Py_UNICODE_strcpy`, `Py_UNICODE_strncpy`: utilice **`PyUnicode_CopyCharacters()`** o **`PyUnicode_Substring()`**
    - `Py_UNICODE_strcmp`: utilice **`PyUnicode_Compare()`**
    - `Py_UNICODE_strncmp`: utilice

  `PyUnicode_Tailmatch()`
  - `Py_UNICODE_strchr`, `Py_UNICODE_strrchr`: utilice **`PyUnicode_FindChar()`**

- Eliminado `PyUnicode_GetMax()`. Migra a las API nuevas (PEP 393 [https://peps.python.org/pep-0393/]). (Contribuido por Inada Naoki en bpo-41103 [https://bugs.python.org/issue?@action=redirect&bpo=41103].)

- Eliminado `PyLong_FromUnicode()`. Migra a **`PyLong_FromUnicodeObject()`**. (Contribuido por Inada Naoki en bpo-41103 [https://bugs.python.org/issue?@action=redirect&bpo=41103].)

- Eliminado `PyUnicode_AsUnicodeCopy()`. Utilice **`PyUnicode_AsUCS4Copy()`** o **`PyUnicode_AsWideCharString()`** (contribución de Inada Naoki en bpo-41103 [https://bugs.python.org/issue?@action=redirect&bpo=41103]).

- Variable `_Py_CheckRecursionLimit` eliminada: ha sido reemplazada por `ceval.recursion_limit` de la estructura **`PyInterpreterState`**. (Contribuido por Victor Stinner en bpo-41834 [https://bugs.python.org/issue?@action=redirect&bpo=41834].)

- Se eliminaron las macros `Py_ALLOW_RECURSION` y `Py_END_ALLOW_RECURSION` sin documentar y el campo `recursion_critical` de la estructura **`PyInterpreterState`**. (Contribuido por Serhiy Storchaka en bpo-41936 [https://bugs.python.org/issue?@action=redirect&bpo=41936].)

- Se eliminó la función `PyOS_InitInterrupts()` indocumentada. La inicialización de Python ya instala implícitamente controladores de señales: consulte **`PyConfig.install_signal_handlers`**. (Contribuido por Victor Stinner en bpo-41713 [https://bugs.python.org/issue?@action=redirect&bpo=41713].)

- Elimina la función `PyAST_Validate()`. Ya no es posible construir un objeto AST (tipo `mod_ty`) con la API C pública. La función ya estaba excluida de la API C limitada (**PEP 384** [https://peps.python.org/pep-0384/]). (Contribuido por Victor Stinner en bpo-43244 [https://bugs.python.org/issue?@action=redirect&bpo=43244].)

- Elimine el archivo de encabezado `symtable.h` y las funciones no documentadas:

  ○ `PyST_GetScope()`
  ○ `PySymtable_Build()`
  ○ `PySymtable_BuildObject()`
  ○ `PySymtable_Free()`
  ○ `Py_SymtableString()`
  ○ `Py_SymtableStringObject()`

  La función `Py_SymtableString()` fue parte de la ABI estable por error, pero no se pudo usar porque el archivo de encabezado `symtable.h` se excluyó de la API C limitada.

  En su lugar, utilice el módulo Python **`symtable`**. (Contribuido por Victor Stinner en bpo-43244 [https://bugs.python.org/issue?@action=redirect&bpo=43244].)

- Elimine **`PyOS_ReadlineFunctionPointer()`** de los encabezados limitados de la API C y de `python3.dll`, la biblioteca que proporciona la ABI estable en Windows. Dado que la función toma un argumento `FILE*`, no se puede garantizar su estabilidad ABI. (Contribuido por Petr Viktorin en bpo-43868 [https://bugs.python.org/issue?@action=redirect&bpo=43868].)

- Elimine los archivos de encabezado `ast.h`, `asdl.h` y `Python-ast.h`. Estas funciones no estaban documentadas y se excluyeron de la API C limitada. La mayoría de los nombres definidos por estos archivos de encabezado no tenían el prefijo `Py` y, por lo tanto, podrían crear conflictos de nombres. Por ejemplo, `Python-ast.h` definió una macro `Yield` que estaba en conflicto con el nombre `Yield` utilizado por el encabezado `<winbase.h>` de Windows. En su lugar, utilice el módulo Python **`ast`**. (Contribuido por

Victor Stinner en bpo-43244 [https://bugs.python.org/issue?@action=redirect&bpo=43244].)

- Elimine las funciones de compilador y analizador utilizando el tipo `struct _mod`, porque se eliminó la API de AST C pública:

    - `PyAST_Compile()`
    - `PyAST_CompileEx()`
    - `PyAST_CompileObject()`
    - `PyFuture_FromAST()`
    - `PyFuture_FromASTObject()`
    - `PyParser_ASTFromFile()`
    - `PyParser_ASTFromFileObject()`
    - `PyParser_ASTFromFilename()`
    - `PyParser_ASTFromString()`
    - `PyParser_ASTFromStringObject()`

  Estas funciones no estaban documentadas y se excluyeron de la API C limitada. (Contribuido por Victor Stinner en bpo-43244 [https://bugs.python.org/issue?@action=redirect&bpo=43244].)

- Elimine el archivo de encabezado `pyarena.h` con funciones:

    - `PyArena_New()`
    - `PyArena_Free()`
    - `PyArena_Malloc()`
    - `PyArena_AddPyObject()`

  Estas funciones no estaban documentadas, estaban excluidas de la API C limitada y el compilador solo las usaba internamente. (Contribuido por Victor Stinner en bpo-43244 [https://bugs.python.org/issue?@action=redirect&bpo=43244].)

- El miembro `PyThreadState.use_tracing` se ha eliminado para optimizar Python. (Contribuido por Mark Shannon en bpo-43760 [https://bugs.python.org/issue?@action=redirect&bpo=43760].)

Característica de seguridad notable en

3.10.7

La conversión entre `int` y `str` en bases distintas de 2 (binario), 4, 8 (octal), 16 (hexadecimal) o 32, como la base 10 (decimal), ahora genera un `ValueError` si el número de dígitos en forma de cadena está por encima de un límite para evitar posibles ataques de denegación de servicio debido a la complejidad algorítmica. Esta es una mitigación para **CVE 2020-10735** [https://www.cve.org/CVERecord?id=CVE-2020-10735]. Este límite se puede configurar o deshabilitar mediante una variable de entorno, un indicador de línea de comando o las API `sys`. Consulte la documentación de integer string conversion length limitation. El límite predeterminado es 4300 dígitos en forma de cadena.

Característica de seguridad notable en 3.10.8

El módulo `mailcap` en desuso ahora se niega a inyectar texto no seguro (nombres de archivos, tipos MIME, parámetros) en los comandos del shell. En lugar de utilizar dicho texto, advertirá y actuará como si no se encontrara una coincidencia (o para comandos de prueba, como si la prueba fallara). (Aportado por Petr Viktorin en gh-98966 [https://github.com/python/cpython/issues/98966].)

Cambios notables en 3.10.12

archivo tar

- Los métodos de extracción en `tarfile` y `shutil.unpack_archive()` tienen un nuevo argumento, *filter*, que permite limitar funciones tar que puedan resultar sorprendentes o peligrosas, como la creación de archivos fuera del directorio de destino. Consulte Extraction filters para obtener más detalles. En Python 3.12, el uso sin el argumento *filter* mostrará un `DeprecationWarning`. En Python 3.14, el valor predeterminado cambiará a `'data'`. (Aportado por Petr Viktorin en **PEP 706** [https://peps.python.org/pep-0706/].)

Qué hay de nuevo en Python 3.9

Editor:

Łukasz Langa

This article explains the new features in Python 3.9, compared to 3.8. Python 3.9 was released on October 5, 2020. For full details, see the changelog.

Ver también

PEP 596 [https://peps.python.org/pep-0596/] - Programa de lanzamiento de Python 3.9

Resumen: aspectos destacados de la versión

Nuevas funciones de sintaxis:

- **PEP 584** [https://peps.python.org/pep-0584/], operadores unión agregados a `dict`;
- **PEP 585** [https://peps.python.org/pep-0585/], genéricos de sugerencia de tipo en colecciones estándar;
- **PEP 614** [https://peps.python.org/pep-0614/], restricciones gramaticales relajadas para los decoradores.

Nuevas funciones integradas:

- **PEP 616** [https://peps.python.org/pep-0616/], métodos de cadena para eliminar prefijos y sufijos.

Nuevas funciones en la biblioteca estándar:

- **PEP 593** [https://peps.python.org/pep-0593/], función flexible y anotaciones variables;
- Se agregó `os.pidfd_open()` que permite la gestión de procesos sin razas ni señales.

Mejoras en el intérprete:

- **PEP 573** [https://peps.python.org/pep-0573/], acceso rápido al estado del módulo desde métodos de tipos de extensión C;
- **PEP 617** [https://peps.python.org/pep-0617/], CPython ahora usa un nuevo analizador basado en PEG;
- varias incorporaciones de Python (rango, tupla, conjunto, conjunto frozenset, lista, dict) ahora se aceleran usando **PEP 590** [https://peps.python.org/pep-0590/] vectorcall;
- la recolección de basura no bloquea los objetos resucitados;
- a number of Python modules (`_abc`, `audioop`, `_bz2`, `_codecs`, `_contextvars`, `_crypt`, `_functools`, `_json`, `_locale`, `math`, `operator`, `resource`, `time`, `_weakref`) now use multiphase initialization as defined by PEP 489;
- a number of standard library modules (`audioop`, `ast`, `grp`, `_hashlib`, `pwd`, `_posixsubprocess`, `random`, `select`, `struct`, `termios`, `zlib`) are now using the stable ABI defined by PEP 384.

Nuevos módulos de biblioteca:

- **PEP 615** [https://peps.python.org/pep-0615/], la base de datos de zona horaria de IANA ahora está presente en la biblioteca estándar en el módulo `zoneinfo`;
- ahora se proporciona una implementación de una especie de gráfico topológico en el nuevo módulo `graphlib`.

Cambios en el proceso de lanzamiento:

- **PEP 602** [https://peps.python.org/pep-0602/], CPython adopta un ciclo de lanzamiento anual.

Debe verificar DeprecationWarning en su código

Cuando todavía se admitía Python 2.7, se conservaba una gran cantidad de funciones en Python 3 por compatibilidad con versiones anteriores de Python 2.7. Con el fin de la compatibilidad con Python 2, estas capas de compatibilidad con versiones anteriores se eliminaron o se eliminarán pronto. La mayoría de ellos emitió una advertencia **`DeprecationWarning`** durante varios años. Por ejemplo, usar `collections.Mapping` en lugar de `collections.abc.Mapping` emite un **`DeprecationWarning`** desde Python 3.3, lanzado en 2012.

Pruebe su aplicación con la opción de línea de comandos **`-W`** `default` para ver **`DeprecationWarning`** y **`PendingDeprecationWarning`**, o incluso con **`-W`** `error` para tratarlos como errores. Warnings Filter se puede utilizar para ignorar las advertencias del código de terceros.

Python 3.9 es la última versión que proporciona esas capas de compatibilidad con versiones anteriores de Python 2, para dar más tiempo a los mantenedores de proyectos de Python para organizar la eliminación del soporte de Python 2 y agregar soporte para Python 3.9.

Los alias de Abstract Base Classes en el módulo **`collections`**, como el alias de `collections.Mapping` de **`collections.abc.Mapping`**, se mantienen para una última versión por compatibilidad con versiones anteriores. Se eliminarán de Python 3.10.

De manera más general, intente ejecutar sus pruebas en el Python Development Mode, lo que ayuda a preparar su código para que sea compatible con la próxima versión de Python.

Nota: también se eliminaron varias obsoletas en esta versión de Python. Consulte la sección Remoto.

Nuevas características

Operadores de combinación y actualización de diccionarios

Los operadores de fusión (`|`) y actualización (`|=`) se han agregado a la clase **`dict`** incorporada. Estos complementan los métodos `dict.update` y `{**d1, **d2}` existentes para fusionar diccionarios.

Ejemplo:

```
>>> x = {"key1": "value1 from x", "key2": "value2 from x
>>> y = {"key2": "value2 from y", "key3": "value3 from y
>>> x | y
{'key1': 'value1 from x', 'key2': 'value2 from y', 'key3
>>> y | x
{'key2': 'value2 from x', 'key3': 'value3 from y', 'key1
```

Consulte **PEP 584** [https://peps.python.org/pep-0584/] para obtener una descripción completa. (Contribuido por Brandt Bucher en bpo-36144 [https://bugs.python.org/issue?@action=redirect&bpo=36144].)

Nuevos métodos de cadena para eliminar prefijos y sufijos

Se han agregado **`str.removeprefix(prefix)`** y **`str.removesuffix(suffix)`** para eliminar fácilmente un prefijo o sufijo innecesario de una cadena. También se han agregado los métodos correspondientes `bytes`, `bytearray` y `collections.UserString`. Consulte **PEP 616** [https://peps.python.org/pep-0616/] para obtener una descripción completa. (Contribuido por Dennis Sweeney en bpo-39939 [https://bugs.python.org/issue?@action=redirect&bpo=39939].)

Tipos genéricos de sugerencia en colecciones estándar

En las anotaciones de tipo, ahora puede usar tipos de colección integrados como `list` y `dict` como tipos genéricos en lugar de importar los tipos en mayúsculas correspondientes (por ejemplo, `List` o `Dict`) desde `typing`. Algunos otros tipos de la biblioteca estándar ahora también son genéricos, por ejemplo, `queue.Queue`.

Ejemplo:

```
def greet_all(names: list[str]) -> None:
    for name in names:
        print("Hello", name)
```

Consulte **PEP 585** [https://peps.python.org/pep-0585/] para obtener más detalles. (Contribuido por Guido van Rossum, Ethan Smith y Batuhan Taşkaya en bpo-39481 [https://bugs.python.org/issue?@action=redirect&bpo=39481].)

Nuevo analizador

Python 3.9 usa un nuevo analizador, basado en PEG [https://en.wikipedia.org/wiki/Parsing_expression_grammar] en lugar de LL(1) [https://en.wikipedia.org/wiki/LL_parser]. El rendimiento del analizador nuevo es aproximadamente comparable al del analizador anterior, pero el formalismo PEG es más flexible que LL (1) cuando se trata de diseñar nuevas funciones de lenguaje. Comenzaremos a usar esta flexibilidad en Python 3.10 y versiones posteriores.

El módulo `ast` usa el nuevo analizador y produce el mismo AST que el analizador anterior.

In Python 3.10, the old parser will be deleted and so will all functionality that depends on it (primarily the `parser` module, which has long been deprecated). In Python 3.9 *only*, you can switch back to the LL(1) parser using a command line switch (`-X oldparser`) or an environment variable (`PYTHONOLDPARSER=1`).

Consulte **PEP 617** [https://peps.python.org/pep-0617/] para obtener más detalles. (Contribuido por Guido van Rossum, Pablo Galindo y Lysandros Nikolaou en bpo-40334 [https://bugs.python.org/issue?@action=redirect&bpo=40334].)

Otros cambios de idioma

- `__import__()` ahora lanza `ImportError` en lugar de `ValueError`, que solía ocurrir cuando una importación relativa pasaba de su paquete de nivel superior. (Contribuido por Ngalim Siregar en bpo-37444 [https://bugs.python.org/issue?@action=redirect&bpo=37444].)

- Python ahora obtiene la ruta absoluta del nombre de archivo del script especificado en la línea de comando (por ejemplo: `python3 script.py`): el módulo `__file__` attribute of the **`__main__`** se convirtió en una ruta absoluta, en lugar de una ruta relativa. Estas rutas ahora siguen siendo válidas después de que las tramas del módulo **`os.chdir()`**. Como efecto alterno, la traza también muestra la ruta absoluta para los marcos del módulo **`__main__`** en este caso. (Contribuido por Victor Stinner en bpo-20443 [https://bugs.python.org/issue?@action=redirect&bpo=20443].)

- En Python Development Mode y en debug build, los argumentos de *encoding* and *errors* ahora se comprueban para las operaciones de codificación y decodificación de cadenas. Ejemplos: **`open()`**, **`str.encode()`** y **`bytes.decode()`**.

  De forma predeterminada, para un mejor rendimiento, el argumento *errors* argument is only checked at the first encoding/decoding error and the *encoding* a veces se ignora para cadenas vacías. (Contribuido por Victor Stinner en bpo-37388 [https://bugs.python.org/issue?@action=redirect&bpo=37388].)

- `"".replace("", s, n)` ahora devuelve `s` en lugar de una cadena vacía para todos los `n` distintos de cero. Ahora es compatible con `"".replace("", s)`. Hay cambios similares para los objetos **`bytes`** y **`bytearray`**. (Contribuido por Serhiy Storchaka en bpo-28029 [https://bugs.python.org/issue?@action=redirect&bpo=28029].)

- Cualquier expresión válida ahora se puede utilizar como decorator. Anteriormente, la gramática era mucho más restrictiva. Consulte **PEP 614** [https://peps.python.org/pep-0614/] para obtener más detalles. (Contribuido por Brandt Bucher en bpo-39702 [https://bugs.python.org/issue?@action=redirect&bpo=39702].)

- Ayuda mejorada para el módulo **`typing`**. Las cadenas de documentos ahora se muestran para todas las formas especiales y alias genéricos especiales (como `Union` y `List`). El uso de **`help()`** con un alias genérico como

`List[int]` mostrará la ayuda para el tipo concreto correspondiente (`list` en este caso). (Contribuido por Serhiy Storchaka en bpo-40257 [https://bugs.python.org/issue?@action=redirect&bpo=40257].)

- La ejecución en paralelo de **`aclose()`** / **`asend()`** / **`athrow()`** ahora está prohibida, y `ag_running` ahora refleja el estado de ejecución real del generador asíncrono. (Contribuido por Yury Selivanov en bpo-30773 [https://bugs.python.org/issue?@action=redirect&bpo=30773].)
- Los errores inesperados al llamar al método `__iter__` ya no están enmascarados por `TypeError` en el operador **`in`** y funciones **`contains()`**, **`indexOf()`** y **`countOf()`** del módulo **`operator`**. (Contribuido por Serhiy Storchaka en bpo-40824 [https://bugs.python.org/issue?@action=redirect&bpo=40824].)
- Las expresiones lambda sin paréntesis ya no pueden ser la parte de expresión en una cláusula `if` en comprensiones y expresiones generadoras. Consulte bpo-41848 [https://bugs.python.org/issue?@action=redirect&bpo=41848] y bpo-43755 [https://bugs.python.org/issue?@action=redirect&bpo=43755] para obtener más detalles.

Nuevos módulos

zoneinfo

El módulo **`zoneinfo`** brinda soporte para la base de datos de zona horaria de IANA a la biblioteca estándar. Agrega **`zoneinfo.ZoneInfo`**, una implementación de **`datetime.tzinfo`** concreta respaldada por los datos de la zona horaria del sistema.

Ejemplo:

```
>>> from zoneinfo import ZoneInfo
>>> import datetime as dt
>>> # Daylight saving time
>>> when = dt.datetime(2020, 10, 31, 12, tzinfo=ZoneInfo
>>> print(when)
2020-10-31 12:00:00-07:00
>>> when.tzname()
'PDT'

>>> # Standard time
>>> when += dt.timedelta(days=7)
>>> print(when)
2020-11-07 12:00:00-08:00
>>> print(when.tzname())
PST
```

As a fall-back source of data for platforms that don’t ship the IANA database, the tzdata [https://pypi.org/project/tzdata/] module was released as a first-party package – distributed via PyPI and maintained by the CPython core team.

Ver también

PEP 615 [https://peps.python.org/pep-0615/]: compatibilidad con la base de datos de zona horaria de IANA en la biblioteca estándar

PEP escrito e implementado por Paul Ganssle

Graphlib

Se agregó un nuevo módulo, `graphlib`, que contiene la clase `graphlib.TopologicalSorter` para ofrecer funcionalidad para realizar la clasificación topológica de gráficos. (Contribuido por Pablo Galindo, Tim Peters y Larry Hastings en bpo-17005 [https://bugs.python.org/issue?@action=redirect&bpo=17005].)

Módulos mejorados

ast

Se agregó la opción *indent* a `dump()` que le permite producir una salida con sangría de varias líneas. (Contribuido por Serhiy Storchaka en bpo-37995 [https://bugs.python.org/issue?@action=redirect&bpo=37995].)

Se agregó `ast.unparse()` como una función en el módulo `ast` que se puede usar para descomprimir un objeto `ast.AST` y producir una cadena con código que produciría un objeto **`ast.AST`** equivalente cuando se analiza. (Contribuido por Pablo Galindo y Batuhan Taskaya en bpo-38870 [https://bugs.python.org/issue?@action=redirect&bpo=38870].)

Se agregaron cadenas de documentación a los nodos AST que contienen la firma ASDL utilizada para construir ese nodo. (Contribuido por Batuhan Taskaya en bpo-39638 [https://bugs.python.org/issue?@action=redirect&bpo=39638].)

asyncio

Debido a importantes problemas de seguridad, el parámetro *reuse_address* de `asyncio.loop.create_datagram_endpoint()` ya no es compatible. Esto se debe al comportamiento de la opción de socket `SO_REUSEADDR` en UDP. Para obtener más detalles, consulte la documentación de `loop.create_datagram_endpoint()`. (Contribuido por Kyle Stanley, Antoine Pitrou y Yury Selivanov en bpo-37228 [https://bugs.python.org/issue?@action=redirect&bpo=37228].)

Se agregó un nuevo coroutine `shutdown_default_executor()` que programa un apagado para el ejecutor predeterminado que espera en el `ThreadPoolExecutor` para finalizar el cierre. Además, `asyncio.run()` se ha actualizado para utilizar el nuevo coroutine. (Contribuido por Kyle Stanley en bpo-34037 [https://bugs.python.org/issue?@action=redirect&bpo=34037].)

Added **`asyncio.PidfdChildWatcher`**, a Linux-specific child watcher implementation that polls process file descriptors. (bpo-38692 [https://bugs.python.org/issue?@action=redirect&bpo=38692])

Se agregó un nuevo coroutine `asyncio.to_thread()`. Se utiliza principalmente para ejecutar funciones vinculadas a IO en un hilo separado para evitar bloquear el bucle de eventos, y esencialmente funciona como una versión de alto nivel de `run_in_executor()` que puede tomar directamente argumentos de palabras clave. (Contribuido por Kyle Stanley y Yury Selivanov en bpo-32309 [https://bugs.python.org/issue?@action=redirect&bpo=32309].)

Al cancelar la tarea debido a un tiempo de espera, `asyncio.wait_for()` ahora esperará hasta que se complete la cancelación también en el caso de que *timeout* sea <= 0, como ocurre con los tiempos de espera positivos. (Contribuido por Elvis Pranskevichus en bpo-32751 [https://bugs.python.org/issue?@action=redirect&bpo=32751].)

`asyncio` now raises `TypeError` when calling incompatible methods with an `ssl.SSLSocket` socket. (Contributed by Ido Michael in bpo-37404 [https://bugs.python.org/issue?@action=redirect&bpo=37404].)

compilar todo

Se agregó una nueva posibilidad de usar enlaces duros para archivos `.pyc` duplicados: parámetro *hardlink_dupes* y opción de línea de comando –hardlink-dupes. (Contribuido por Lumír "Frenzy" Balhar en bpo-40495 [https://bugs.python.org/issue?@action=redirect&bpo=40495].)

Se agregaron nuevas opciones para la manipulación de rutas en los archivos `.pyc` resultantes: parámetros *stripdir*, *prependdir*, *limit_sl_dest* y opciones de línea de comando -s, -p, -e. Se agregó la posibilidad de especificar la opción para un nivel de optimización varias veces. (Contribuido por Lumír "Frenzy" Balhar en bpo-38112 [https://bugs.python.org/issue?@action=redirect&bpo=38112].)

Futuros concurrentes

Se agregó un nuevo parámetro *cancel_futures* a `concurrent.futures.Executor.shutdown()` que cancela todos los futuros pendientes que no han comenzado a ejecutarse, en lugar de esperar a que se completen antes de apagar el ejecutor. (Contribuido por Kyle Stanley en bpo-39349 [https://bugs.python.org/

issue?@action=redirect&bpo=39349].)

Se eliminaron los subprocesos del demonio de `ThreadPoolExecutor` y `ProcessPoolExecutor`. Esto mejora la compatibilidad con los subinterpretadores y la previsibilidad en sus procesos de cierre. (Contribuido por Kyle Stanley en bpo-39812 [https://bugs.python.org/issue?@action=redirect&bpo=39812].)

Los trabajadores en `ProcessPoolExecutor` ahora se generan a pedido, solo cuando no hay trabajadores inactivos disponibles para reutilizar. Esto optimiza la sobrecarga de inicio y reduce la cantidad de tiempo de CPU perdido para los trabajadores inactivos. (Contribuido por Kyle Stanley en bpo-39207 [https://bugs.python.org/issue?@action=redirect&bpo=39207].)

maldiciones

Se agregaron funciones `curses.get_escdelay()`, `curses.set_escdelay()`, `curses.get_tabsize()` y `curses.set_tabsize()`. (Contribuido por Anthony Sottile en bpo-38312 [https://bugs.python.org/issue?@action=redirect&bpo=38312].)

fecha y hora

The `isocalendar()` of `datetime.date` and `isocalendar()` of `datetime.datetime` methods now returns a `namedtuple()` instead of a `tuple`. (Contributed by Donghee Na in bpo-24416 [https://bugs.python.org/issue?@action=redirect&bpo=24416].)

distutils

El comando: command: *upload* ahora crea resúmenes de hash SHA2-256 y Blake2b-256. Omite MD5 en plataformas que bloquean el resumen de MD5. (Contribuido por Christian Heimes en bpo-40698 [https://bugs.python.org/issue?@action=redirect&bpo=40698].)

fcntl

Added constants **`fcntl.F_OFD_GETLK`**, **`fcntl.F_OFD_SETLK`** and **`fcntl.F_OFD_SETLKW`**. (Contributed by Donghee Na in

bpo-38602 [https://bugs.python.org/issue?@action=redirect&bpo=38602].)

ftplib

`FTP` and `FTP_TLS` now raise a `ValueError` if the given timeout for their constructor is zero to prevent the creation of a non-blocking socket. (Contributed by Donghee Na in bpo-39259 [https://bugs.python.org/issue?@action=redirect&bpo=39259].)

GC

Cuando el recolector de basura realiza una recolección en la que resucitan algunos objetos (se puede acceder a ellos desde fuera de los ciclos aislados después de que se hayan ejecutado los finalizadores), no bloquee la recolección de todos los objetos que aún no se pueden alcanzar. (Contribuido por Pablo Galindo y Tim Peters en bpo-38379 [https://bugs.python.org/issue?@action=redirect&bpo=38379].)

Se agregó una nueva función `gc.is_finalized()` para verificar si el recolector de basura ha finalizado un objeto. (Contribuido por Pablo Galindo en bpo-39322 [https://bugs.python.org/issue?@action=redirect&bpo=39322].)

hashlib

El módulo `hashlib` ahora puede usar hashes SHA3 y SHAKE XOF de OpenSSL cuando esté disponible. (Contribuido por Christian Heimes en bpo-37630 [https://bugs.python.org/issue?@action=redirect&bpo=37630].)

Los módulos hash incorporados ahora pueden desactivarse con `./configure --without-builtin-hashlib-hashes` o activarse selectivamente con p. Ej. `./configure --with-builtin-hashlib-hashes=sha3,blake2` para forzar el uso de la implementación basada en OpenSSL. (Contribuido por Christian Heimes en bpo-40479 [https://bugs.python.org/issue?@action=redirect&bpo=40479])

http

HTTP status codes `103 EARLY_HINTS`, `418 IM_A_TEAPOT` and `425 TOO_EARLY` are added to **`http.HTTPStatus`**. (Contributed by Donghee Na in bpo-39509 [https://bugs.python.org/issue?@action=redirect&bpo=39509] and Ross Rhodes in bpo-39507 [https://bugs.python.org/issue?@action=redirect&bpo=39507].)

IDLE e idlelib

Opción agregada para apagar el cursor parpadeando. (Contribuido por Zackery Spytz en bpo-4603 [https://bugs.python.org/issue?@action=redirect&bpo=4603].)

La tecla Escape ahora cierra las ventanas de finalización IDLE. (Contribuido por Johnny Najera en bpo-38944 [https://bugs.python.org/issue?@action=redirect&bpo=38944].)

Se agregaron palabras clave a la lista de finalización del nombre del módulo. (Contribuido por Terry J. Reedy en bpo-37765 [https://bugs.python.org/issue?@action=redirect&bpo=37765].)

Novedades de la versión de mantenimiento 3.9

Hacer que IDLE invoque **`sys.excepthook()`** (cuando se inicia sin "-n"). Anteriormente se ignoraban los hooks de usuario. (Contribuido por Ken Hilton en bpo-43008 [https://bugs.python.org/issue?@action=redirect&bpo=43008].)

Los cambios anteriores se han actualizado a las versiones de mantenimiento 3.8.

Reorganizar el diálogo de configuración. Dividir la pestaña General en Windows y Shell/Ed. Mover las fuentes de ayuda, que amplían el menú Ayuda, a la pestaña Extensiones. Hacer espacio para nuevas opciones y acortar el diálogo. Esto último hace que el diálogo se adapte mejor a las pantallas pequeñas. (Contribuido por Terry Jan Reedy en bpo-40468 [https://bugs.python.org/issue?@action=redirect&bpo=40468].) Mover la configuración del espacio de sangría de la pestaña Fuente a la nueva pestaña Windows. (Contribuido por Mark Roseman y Terry Jan Reedy en bpo-33962 [https://bugs.python.org/issue?@action=redirect&bpo=33962].)

Aplicar el resaltado de sintaxis a los archivos `.pyi`. (Contribución de Alex Waygood y Terry Jan Reedy en bpo-45447 [https://bugs.python.org/issue?@action=redirect&bpo=45447])

imaplib

`IMAP4` and **`IMAP4_SSL`** now have an optional *timeout* parameter for their constructors. Also, the **`open()`** method now has an optional *timeout* parameter with this change. The overridden methods of **`IMAP4_SSL`** and **`IMAP4_stream`** were applied to this change. (Contributed by Donghee Na in bpo-38615 [https://bugs.python.org/issue?@action=redirect&bpo=38615].)

`imaplib.IMAP4.unselect()` is added. **`imaplib.IMAP4.unselect()`** frees server's resources associated with the selected mailbox and returns the server to the authenticated state. This command performs the same actions as **`imaplib.IMAP4.close()`**, except that no messages are permanently removed from the currently selected mailbox. (Contributed by Donghee Na in bpo-40375 [https://bugs.python.org/issue?@action=redirect&bpo=40375].)

importlib

Para mejorar la coherencia con las declaraciones de importación, **`importlib.util.resolve_name()`** ahora lanza **`ImportError`** en lugar de **`ValueError`** para intentos de importación relativa no válidos. (Contribuido por Ngalim Siregar en bpo-37444 [https://bugs.python.org/issue?@action=redirect&bpo=37444].)

Los cargadores de importación que publican objetos de módulo inmutables ahora pueden publicar paquetes inmutables además de módulos individuales. (Contribuido por Dino Viehland en bpo-39336 [https://bugs.python.org/issue?@action=redirect&bpo=39336].)

Se agregó la función **`importlib.resources.files()`** con soporte para subdirectorios en los datos del paquete, compatible con el backport en `importlib_resources` versión 1.5. (Contribuido por Jason R. Coombs en bpo-39791 [https://bugs.python.org/issue?@action=redirect&bpo=39791].)

`importlib.metadata` actualizado de `importlib_metadata` versión 1.6.1.

inspeccionar

`inspect.BoundArguments.arguments` se cambia de `OrderedDict` a dictado regular. (Contribuido por Inada Naoki en bpo-36350 [https://bugs.python.org/issue?@action=redirect&bpo=36350] y bpo-39775 [https://bugs.python.org/issue?@action=redirect&bpo=39775].)

dirección IP

`ipaddress` ahora admite direcciones de ámbito IPv6 (dirección IPv6 con sufijo `%<scope_id>`).

Las direcciones IPv6 con alcance se pueden analizar mediante **`ipaddress.IPv6Address`**. Si está presente, la identificación de la zona del alcance está disponible a través del atributo **`scope_id`**. (Contribuido por Oleksandr Pavliuk en bpo-34788 [https://bugs.python.org/issue?@action=redirect&bpo=34788].)

A partir de Python 3.9.5, el módulo **`ipaddress`** ya no acepta ceros a la izquierda en cadenas de direcciones IPv4. (Contribuido por Christian Heimes en bpo-36384 [https://bugs.python.org/issue?@action=redirect&bpo=36384]).

Matemáticas

Se amplió la función **`math.gcd()`** para manejar múltiples argumentos. Anteriormente, solo respaldaba dos argumentos. (Contribuido por Serhiy Storchaka en bpo-39648 [https://bugs.python.org/issue?@action=redirect&bpo=39648].)

`math.lcm()` agregado: devuelve el mínimo común múltiplo de argumentos especificados. (Contribuido por Mark Dickinson, Ananthakrishnan y Serhiy Storchaka en bpo-39479 [https://bugs.python.org/issue?@action=redirect&bpo=39479] y bpo-39648 [https://bugs.python.org/issue?@action=redirect&bpo=39648].)

`math.nextafter()` agregado: devuelve el siguiente valor de punto flotante después de *x* towards *y*. (Contribuido por Victor Stinner en bpo-39288 [https://bugs.python.org/issue?@action=redirect&bpo=39288].)

`math.ulp()` agregado: devuelve el valor del bit menos significativo de un flotador. (Contribuido por Victor Stinner en bpo-39310 [https://bugs.python.org/issue?@action=redirect&bpo=39310].)

multiprocesamiento

La clase `multiprocessing.SimpleQueue` tiene un nuevo método `close()` para cerrar explícitamente la cola. (Contribuido por Victor Stinner en bpo-30966 [https://bugs.python.org/issue?@action=redirect&bpo=30966].)

nntplib

`NNTP` and `NNTP_SSL` now raise a `ValueError` if the given timeout for their constructor is zero to prevent the creation of a non-blocking socket. (Contributed by Donghee Na in bpo-39259 [https://bugs.python.org/issue?@action=redirect&bpo=39259].)

os

Added `CLD_KILLED` and `CLD_STOPPED` for `si_code`. (Contributed by Donghee Na in bpo-38493 [https://bugs.python.org/issue?@action=redirect&bpo=38493].)

Exposed the Linux-specific `os.pidfd_open()` (bpo-38692 [https://bugs.python.org/issue?@action=redirect&bpo=38692]) and `os.P_PIDFD` (bpo-38713 [https://bugs.python.org/issue?@action=redirect&bpo=38713]) for process management with file descriptors.

La función `os.unsetenv()` ahora también está disponible en Windows. (Contribuido por Victor Stinner en bpo-39413 [https://bugs.python.org/issue?@action=redirect&bpo=39413].)

Las funciones `os.putenv()` y `os.unsetenv()` ahora están siempre disponibles. (Contribuido por Victor Stinner en bpo-39395 [https://bugs.python.org/issue?@action=redirect&bpo=39395].)

Función **`os.waitstatus_to_exitcode()`** agregada: convierte un estado de espera en un código de salida. (Contribuido por Victor Stinner en bpo-40094 [https://bugs.python.org/issue?@action=redirect&bpo=40094].)

Pathlib

Added **`pathlib.Path.readlink()`** which acts similarly to **`os.readlink()`**. (Contributed by Girts Folkmanis in bpo-30618 [https://bugs.python.org/issue?@action=redirect&bpo=30618])

pdb

En Windows, ahora **`Pdb`** es compatible con `~/.pdbrc`. (Contribuido por Tim Hopper y Dan Lidral-Porter en bpo-20523 [https://bugs.python.org/issue?@action=redirect&bpo=20523].)

poplib

`POP3` and **`POP3_SSL`** now raise a **`ValueError`** if the given timeout for their constructor is zero to prevent the creation of a non-blocking socket. (Contributed by Donghee Na in bpo-39259 [https://bugs.python.org/issue?@action=redirect&bpo=39259].)

pprint

`pprint` ahora puede imprimir **`types.SimpleNamespace`** de forma bonita. (Contribuido por Carl Bordum Hansen en bpo-37376 [https://bugs.python.org/issue?@action=redirect&bpo=37376].)

Pydoc

The documentation string is now shown not only for class, function, method etc, but for any object that has its own **`__doc__`** attribute. (Contributed by Serhiy Storchaka in bpo-40257 [https://bugs.python.org/issue?@action=redirect&bpo=40257].)

aleatorio

Added a new `random.Random.randbytes()` method: generate random bytes. (Contributed by Victor Stinner in bpo-40286 [https://bugs.python.org/issue?@action=redirect&bpo=40286].)

señal

Expuso el `signal.pidfd_send_signal()` específico de Linux para enviar señales a un proceso utilizando un descriptor de archivo en lugar de un pid. (bpo-38712 [https://bugs.python.org/issue?@action=redirect&bpo=38712])

smtplib

`SMTP` and `SMTP_SSL` now raise a `ValueError` if the given timeout for their constructor is zero to prevent the creation of a non-blocking socket. (Contributed by Donghee Na in bpo-39259 [https://bugs.python.org/issue?@action=redirect&bpo=39259].)

`LMTP` constructor now has an optional *timeout* parameter. (Contributed by Donghee Na in bpo-39329 [https://bugs.python.org/issue?@action=redirect&bpo=39329].)

enchufe

The `socket` module now exports the `CAN_RAW_JOIN_FILTERS` constant on Linux 4.1 and greater. (Contributed by Stefan Tatschner and Zackery Spytz in bpo-25780 [https://bugs.python.org/issue?@action=redirect&bpo=25780].)

The socket module now supports the `CAN_J1939` protocol on platforms that support it. (Contributed by Karl Ding in bpo-40291 [https://bugs.python.org/issue?@action=redirect&bpo=40291].)

El módulo de enchufe ahora tiene las funciones `socket.send_fds()` y `socket.recv_fds()`. (Contribuido por Joannah Nanjekye, Shinya Okano y Victor Stinner en bpo-28724 [https://bugs.python.org/issue?@action=redirect&bpo=28724].)

tiempo

En AIX, `thread_time()` ahora se implementa con `thread_cputime()` que tiene una resolución de nanosegundos, en lugar de `clock_gettime(CLOCK_THREAD_CPUTIME_ID)` que tiene una resolución de 10 ms. (Contribuido por Batuhan Taskaya en bpo-40192 [https://bugs.python.org/issue?@action=redirect&bpo=40192])

sys

Added a new `sys.platlibdir` attribute: name of the platform-specific library directory. It is used to build the path of standard library and the paths of installed extension modules. It is equal to `"lib"` on most platforms. On Fedora and SuSE, it is equal to `"lib64"` on 64-bit platforms. (Contributed by Jan Matějek, Matěj Cepl, Charalampos Stratakis and Victor Stinner in bpo-1294959 [https://bugs.python.org/issue?@action=redirect&bpo=1294959].)

Previously, `sys.stderr` was block-buffered when non-interactive. Now `stderr` defaults to always being line-buffered. (Contributed by Jendrik Seipp in bpo-13601 [https://bugs.python.org/issue?@action=redirect&bpo=13601].)

tracemalloc

Se agregó `tracemalloc.reset_peak()` para establecer el tamaño máximo de los bloques de memoria rastreados al tamaño actual, para medir el pico de piezas de código específicas. (Contribuido por Huon Wilson en bpo-40630 [https://bugs.python.org/issue?@action=redirect&bpo=40630].)

mecanografía

PEP 593 [https://peps.python.org/pep-0593/] introdujo un tipo `typing.Annotated` para decorar tipos existentes con metadatos específicos del contexto y el nuevo parámetro `include_extras` en `typing.get_type_hints()` para acceder a los metadatos en tiempo de ejecución. (Contribuido por Till Varoquaux y Konstantin Kashin.)

unicodedata

La base de datos Unicode se ha actualizado a la versión 13.0.0. (bpo-39926 [https://bugs.python.org/issue?@action=redirect&bpo=39926]).

venv

Los scripts de activación proporcionados por `venv` ahora especifican su personalización de solicitud de manera consistente utilizando siempre el valor especificado por `__VENV_PROMPT__`. Anteriormente, algunos scripts usaban `__VENV_PROMPT__` incondicionalmente, otros solo si estaba configurado (que era el caso predeterminado) y uno usaba `__VENV_NAME__` en su lugar. (Contribuido por Brett Cannon en bpo-37663 [https://bugs.python.org/issue?@action=redirect&bpo=37663].)

xml

Los caracteres de espacio en blanco dentro de los atributos ahora se conservan al serializar `xml.etree.ElementTree` en un archivo XML. Los EOLN ya no se normalizan a «n». Este es el resultado de la discusión sobre cómo interpretar la sección 2.11 de la especificación XML. (Contribuido por Mefistotelis en bpo-39011 [https://bugs.python.org/issue?@action=redirect&bpo=39011].)

Optimizaciones

- Optimización del idioma para la asignación de una variable temporal en las comprensiones. Ahora `for y in [expr]` en comprensiones es tan rápido como una simple asignación `y = expr`. Por ejemplo:

  sumas = [s para s en [0] para x en datos para s en [s + x]]

  A diferencia del operador `:=`, este idioma no filtra una variable al alcance externo.

  (Contribuido por Serhiy Storchaka en bpo-32856 [https://bugs.python.org/issue?@action=redirect&bpo=32856].)

- Manejo de señal optimizado en aplicaciones multiproceso. Si un subproceso diferente al subproceso principal recibe una señal, el bucle de evaluación del código de bytes ya no se interrumpe en cada instrucción de código de bytes para verificar si hay señales pendientes que no se pueden manejar. Solo el hilo principal del intérprete principal puede manejar señales.

Anteriormente, el ciclo de evaluación del código de bytes se interrumpía en cada instrucción hasta que el hilo principal manejaba las señales. (Contribuido por Victor Stinner en bpo-40010 [https://bugs.python.org/issue?@action=redirect&bpo=40010].)

- Optimicé el módulo **`subprocess`** en FreeBSD usando `closefrom()`. (Contribuido por Ed Maste, Conrad Meyer, Kyle Evans, Kubilay Kocak y Victor Stinner en bpo-38061 [https://bugs.python.org/issue?@action=redirect&bpo=38061].)

- **`PyLong_FromDouble()`** ahora es hasta 1,87 veces más rápido para valores que se ajustan a long. (Contribuido por Sergey Fedoseev en bpo-37986 [https://bugs.python.org/issue?@action=redirect&bpo=37986].)

- A number of Python builtins (**`range`**, **`tuple`**, **`set`**, **`frozenset`**, **`list`**, **`dict`**) are now sped up by using **PEP 590** [https://peps.python.org/pep-0590/] vectorcall protocol. (Contributed by Donghee Na, Mark Shannon, Jeroen Demeyer and Petr Viktorin in bpo-37207 [https://bugs.python.org/issue?@action=redirect&bpo=37207].)

- Optimized **`set.difference_update()`** for the case when the other set is much larger than the base set. (Suggested by Evgeny Kapun with code contributed by Michele Orrù in bpo-8425 [https://bugs.python.org/issue?@action=redirect&bpo=8425].)

- El asignador de objetos pequeños de Python (`obmalloc.c`) ahora permite (no más de) que una arena vacía permanezca disponible para su reutilización inmediata, sin devolverla al sistema operativo. Esto evita la paliza en bucles simples donde una arena podría crearse y destruirse de nuevo en cada iteración. (Contribuido por Tim Peters en bpo-37257 [https://bugs.python.org/issue?@action=redirect&bpo=37257].)

- floor division de operación flotante ahora tiene un mejor rendimiento. También se actualiza el mensaje de `ZeroDivisionError` para esta operación. (Contribuido por Donghee Na en bpo-39434 [https://bugs.python.org/issue?@action=redirect&bpo=39434].)
- La decodificación de cadenas cortas ASCII con códecs UTF-8 y ascii ahora es aproximadamente un 15% más rápida. (Contribuido por Inada Naoki en bpo-37348 [https://bugs.python.org/issue?@action=redirect&bpo=37348].)

A continuación, se muestra un resumen de las mejoras de rendimiento desde Python 3.4 hasta Python 3.9:

```
Python version                       3.4     3.5     3.6
--------------                       ---     ---     ---

Variable and attribute read access:
    read_local                       7.1     7.1     5.4
    read_nonlocal                    7.1     8.1     5.8
    read_global                     15.5    19.0    14.3
    read_builtin                    21.1    21.6    18.5
    read_classvar_from_class        25.6    26.5    20.7
    read_classvar_from_instance     22.8    23.5    18.8
    read_instancevar                32.4    33.1    28.0
    read_instancevar_slots          27.8    31.3    20.8
    read_namedtuple                 73.8    57.5    45.0
    read_boundmethod                37.6    37.9    29.6

Variable and attribute write access:
    write_local                      8.7     9.3     5.5
    write_nonlocal                  10.5    11.1     5.6
    write_global                    19.7    21.2    18.0
    write_classvar                  92.9    96.0   104.6
    write_instancevar               44.6    45.8    40.0
    write_instancevar_slots         35.6    36.1    27.3
```

```
Data structure read access:
    read_list                       24.2    24.5    20.8
    read_deque                      24.7    25.5    20.2
    read_dict                       24.3    25.7    22.3
    read_strdict                    22.6    24.3    19.5

Data structure write access:
    write_list                      27.1    28.5    22.5
    write_deque                     28.7    30.1    22.7
    write_dict                      31.4    33.3    29.3
    write_strdict                   28.4    29.9    27.5

Stack (or queue) operations:
    list_append_pop                 93.4   112.7    75.4
    deque_append_pop                43.5    57.0    49.4
    deque_append_popleft            43.7    57.3    49.7

Timing loop:
    loop_overhead                    0.5     0.6     0.4
```

These results were generated from the variable access benchmark script at: `Tools/scripts/var_access_benchmark.py`. The benchmark script displays timings in nanoseconds. The benchmarks were measured on an Intel® Core™ i7-4960HQ processor [https://ark.intel.com/content/www/us/en/ark/products/76088/intel-core-i7-4960hq-processor-6m-cache-up-to-3-80-ghz.html] running the macOS 64-bit builds found at python.org [https://www.python.org/downloads/macos/].

Obsoleto

- El comando distutils `bdist_msi` ahora está en desuso, use `bdist_wheel` (paquetes de ruedas) en su lugar. (Contribuido por Hugo van Kemenade en bpo-39586 [https://bugs.python.org/issue?@action=redirect&bpo=39586].)
- Actualmente, **`math.factorial()`** acepta instancias de **`float`** con valores enteros no negativos (como `5.0`). Sube un **`ValueError`** para flotadores no integrales y negativos. Ahora está en desuso. En futuras versiones de Python, lanzará

un `TypeError` para todos los flotadores. (Contribuido por Serhiy Storchaka en bpo-37315 [https://bugs.python.org/issue?@action=redirect&bpo=37315].)

- The **`parser`** and **`symbol`** modules are deprecated and will be removed in future versions of Python. For the majority of use cases, users can leverage the Abstract Syntax Tree (AST) generation and compilation stage, using the `ast` module.

- The Public C API functions **`PyParser_SimpleParseStringFlags()`**, **`PyParser_SimpleParseStringFlagsFilename()`**, **`PyParser_SimpleParseFileFlags()`** and **`PyNode_Compile()`** are deprecated and will be removed in Python 3.10 together with the old parser.

- El uso de `NotImplemented` en un contexto booleano ha quedado obsoleto, ya que es casi exclusivamente el resultado de implementaciones incorrectas de comparadores ricos. Se convertirá en `TypeError` en una versión futura de Python. (Contribuido por Josh Rosenberg en bpo-35712 [https://bugs.python.org/issue?@action=redirect&bpo=35712].)

- El módulo `random` actualmente acepta cualquier tipo hash como posible valor semilla. Desafortunadamente, no se garantiza que algunos de esos tipos tengan un valor hash determinista. Después de Python 3.9, el módulo restringirá sus semillas a `None`, `int`, `float`, `str`, `bytes` y `bytearray`.

- Abrir el archivo `GzipFile` para escritura sin especificar el argumento *mode* argument is deprecated. In future Python versions it will always be opened for reading by default. Specify the *mode* para abrirlo para escribir y silenciar una advertencia. (Contribuido por Serhiy Storchaka en bpo-28286 [https://bugs.python.org/issue?@action=redirect&bpo=28286].)

- Deprecated the `split()` method of **`_tkinter.TkappType`** in favour of the `splitlist()` method which has more consistent and predictable behavior. (Contributed by Serhiy Storchaka in bpo-38371 [https://

bugs.python.org/issue?@action=redirect&bpo=38371].)

- El paso explícito de objetos de rutina a **`asyncio.wait()`** ha quedado obsoleto y se eliminará en la versión 3.11. (Contribuido por Yury Selivanov y Kyle Stanley en bpo-34790 [https://bugs.python.org/issue?@action=redirect&bpo=34790].)
- binhex4 and hexbin4 standards are now deprecated. The **`binhex`** module and the following **`binascii`** functions are now deprecated:

  ○ **`b2a_hqx()`**, **`a2b_hqx()`**
  ○ **`rlecode_hqx()`**, **`rledecode_hqx()`**

  (Contribuido por Victor Stinner en bpo-39353 [https://bugs.python.org/issue?@action=redirect&bpo=39353].)
- Las clases **`ast`** `slice`, `Index` y `ExtSlice` se consideran obsoletas y se eliminarán en futuras versiones de Python. El propio `value` debe utilizarse en lugar de `Index(value)`. Se debe utilizar `Tuple(slices, Load())` en lugar de `ExtSlice(slices)`. (Contribuido por Serhiy Storchaka en bpo-34822 [https://bugs.python.org/issue?@action=redirect&bpo=34822].)
- Las clases **`ast`** `Suite`, `Param`, `AugLoad` y `AugStore` se consideran obsoletas y se eliminarán en futuras versiones de Python. No fueron generados por el analizador y no aceptados por el generador de código en Python 3. (Contribuido por Batuhan Taskaya en bpo-39639 [https://bugs.python.org/issue?@action=redirect&bpo=39639] y bpo-39969 [https://bugs.python.org/issue?@action=redirect&bpo=39969] y Serhiy Storchaka en bpo-39988 [https://bugs.python.org/issue?@action=redirect&bpo=39988]).
- The **`PyEval_InitThreads()`** and **`PyEval_ThreadsInitialized()`** functions are now deprecated and will be removed in Python 3.11. Calling **`PyEval_InitThreads()`** now does nothing. The GIL is initialized by **`Py_Initialize()`** since Python 3.7. (Contributed by Victor Stinner in bpo-39877 [https://

bugs.python.org/issue?@action=redirect&bpo=39877].)

- Pasar `None` como primer argumento a la función **`shlex.split()`** ha quedado obsoleto. (Contribuido por Zackery Spytz en bpo-33262 [https://bugs.python.org/issue?@action=redirect&bpo=33262].)
- **`smtpd.MailmanProxy()`** is now deprecated as it is unusable without an external module, `mailman`. (Contributed by Samuel Colvin in bpo-35800 [https://bugs.python.org/issue?@action=redirect&bpo=35800].)
- The **`lib2to3`** module now emits a **`PendingDeprecationWarning`**. Python 3.9 switched to a PEG parser (see **PEP 617** [https://peps.python.org/pep-0617/]), and Python 3.10 may include new language syntax that is not parsable by lib2to3's LL(1) parser. The **`lib2to3`** module may be removed from the standard library in a future Python version. Consider third-party alternatives such as LibCST [https://libcst.readthedocs.io/] or parso [https://parso.readthedocs.io/]. (Contributed by Carl Meyer in bpo-40360 [https://bugs.python.org/issue?@action=redirect&bpo=40360].)
- El parámetro *random* de **`random.shuffle()`** ha quedado obsoleto. (Contribuido por Raymond Hettinger en bpo-40465 [https://bugs.python.org/issue?@action=redirect&bpo=40465])

Remoto

- The erroneous version at **`unittest.mock.__version__`** has been removed.
- **`nntplib.NNTP`**: `xpath()` and `xgtitle()` methods have been removed. These methods are deprecated since Python 3.3. Generally, these extensions are not supported or not enabled by NNTP server administrators. For `xgtitle()`, please use **`nntplib.NNTP.descriptions()`** or **`nntplib.NNTP.description()`** instead. (Contributed by Donghee Na in bpo-39366 [https://bugs.python.org/issue?@action=redirect&bpo=39366].)
- **`array.array`**: Se han eliminado los métodos `tostring()`

y `fromstring()`. Eran alias de `tobytes()` y `frombytes()`, obsoletos desde Python 3.2. (Contribuido por Victor Stinner en bpo-38916 [https://bugs.python.org/issue?@action=redirect&bpo=38916].)

- Se ha eliminado la función `sys.callstats()` sin documentar. Desde Python 3.7, quedó obsoleto y siempre devolvió **`None`**. Requería una opción de compilación especial `CALL_PROFILE` que ya se eliminó en Python 3.7. (Contribuido por Victor Stinner en bpo-37414 [https://bugs.python.org/issue?@action=redirect&bpo=37414].)
- Se han eliminado las funciones `sys.getcheckinterval()` y `sys.setcheckinterval()`. Fueron obsoletos desde Python 3.2. Utilice **`sys.getswitchinterval()`** y **`sys.setswitchinterval()`** en su lugar. (Contribuido por Victor Stinner en bpo-37392 [https://bugs.python.org/issue?@action=redirect&bpo=37392].)
- Se ha eliminado la función C `PyImport_Cleanup()`. Se documentó como: «Vacíe la tabla del módulo. Solo para uso interno». (Contribuido por Victor Stinner en bpo-36710 [https://bugs.python.org/issue?@action=redirect&bpo=36710].)
- Se han eliminado los módulos `_dummy_thread` y `dummy_threading`. Estos módulos quedaron obsoletos desde Python 3.7, que requiere compatibilidad con subprocesos. (Contribuido por Victor Stinner en bpo-37312 [https://bugs.python.org/issue?@action=redirect&bpo=37312].)
- `aifc.openfp()` alias to `aifc.open()`, `sunau.openfp()` alias to `sunau.open()`, and `wave.openfp()` alias to **`wave.open()`** have been removed. They were deprecated since Python 3.7. (Contributed by Victor Stinner in bpo-37320 [https://bugs.python.org/issue?@action=redirect&bpo=37320].)
- The **`isAlive()`** method of **`threading.Thread`** has been removed. It was deprecated since Python 3.8. Use **`is_alive()`** instead. (Contributed by Donghee Na in bpo-37804 [https://bugs.python.org/issue?@action=redirect&bpo=37804].)
- Se han eliminado los métodos `getchildren()` y `getiterator()` de las clases **`ElementTree`** y **`Element`** en el módulo **`ElementTree`**. Fueron obsoletos en Python 3.2. Utilice `iter(x)` o `list(x)` en lugar de

`x.getchildren()` y `x.iter()` o `list(x.iter())` en lugar de `x.getiterator()`. (Contribuido por Serhiy Storchaka en bpo-36543 [https://bugs.python.org/issue?@action=redirect&bpo=36543].)

- Se eliminó la antigua API **`plistlib`**, quedó en desuso desde Python 3.4. Utilice las funciones **`load()`**, **`loads()`**, **`dump()`** y **`dumps()`**. Además, se eliminó el parámetro *use_builtin_types*; en su lugar, siempre se utilizan objetos **`bytes`** estándar. (Contribuido por Jon Janzen en bpo-36409 [https://bugs.python.org/issue?@action=redirect&bpo=36409].)
- Se ha eliminado la función C `PyGen_NeedsFinalizing`. No se documentó, probó ni usó en ningún lugar dentro de CPython después de la implementación de **PEP 442** [https://peps.python.org/pep-0442/]. Parche de Joannah Nanjekye. (Contribuido por Joannah Nanjekye en bpo-15088 [https://bugs.python.org/issue?@action=redirect&bpo=15088])
- `base64.encodestring()` y `base64.decodestring()`, los alias obsoletos desde Python 3.1, se han eliminado: utilice **`base64.encodebytes()`** y **`base64.decodebytes()`** en su lugar. (Contribuido por Victor Stinner en bpo-39351 [https://bugs.python.org/issue?@action=redirect&bpo=39351].)
- La función `fractions.gcd()` se ha eliminado, está obsoleta desde Python 3.5 (bpo-22486 [https://bugs.python.org/issue?@action=redirect&bpo=22486]): utilice **`math.gcd()`** en su lugar. (Contribuido por Victor Stinner en bpo-39350 [https://bugs.python.org/issue?@action=redirect&bpo=39350].)
- Se ha eliminado el parámetro *buffering* de **`bz2.BZ2File`**. Desde Python 3.0, se ignoró y su uso emitió un **`DeprecationWarning`**. Pase un objeto de archivo abierto para controlar cómo se abre el archivo. (Contribuido por Victor Stinner en bpo-39357 [https://bugs.python.org/issue?@action=redirect&bpo=39357].)
- Se ha eliminado el parámetro *encoding* de **`json.loads()`**. A partir de Python 3.1, fue obsoleto e ignorado; su uso ha emitido un **`DeprecationWarning`** desde Python 3.8. (Contribuido por Inada Naoki en bpo-39377 [https://bugs.python.org/issue?@action=redirect&bpo=39377])
- Las declaraciones `with (await asyncio.lock):` y `with (yield from asyncio.lock):` ya no son compatibles, utilice `async with lock` en su lugar. Lo

mismo es correcto para `asyncio.Condition` y `asyncio.Semaphore`. (Contribuido por Andrew Svetlov en bpo-34793 [https://bugs.python.org/issue?@action=redirect&bpo=34793].)

- The **`sys.getcounts()`** function, the `-X showalloccount` command line option and the `show_alloc_count` field of the C structure **`PyConfig`** have been removed. They required a special Python build by defining `COUNT_ALLOCS` macro. (Contributed by Victor Stinner in bpo-39489 [https://bugs.python.org/issue?@action=redirect&bpo=39489].)
- Se ha eliminado el atributo `_field_types` de la clase **`typing.NamedTuple`**. Quedó en desuso desde Python 3.8. En su lugar, utilice el atributo `__annotations__`. (Contribuido por Serhiy Storchaka en bpo-40182 [https://bugs.python.org/issue?@action=redirect&bpo=40182].)
- The **`symtable.SymbolTable.has_exec()`** method has been removed. It was deprecated since 2006, and only returning `False` when it's called. (Contributed by Batuhan Taskaya in bpo-40208 [https://bugs.python.org/issue?@action=redirect&bpo=40208])
- The **`asyncio.Task.current_task()`** and **`asyncio.Task.all_tasks()`** have been removed. They were deprecated since Python 3.7 and you can use **`asyncio.current_task()`** and **`asyncio.all_tasks()`** instead. (Contributed by Rémi Lapeyre in bpo-40967 [https://bugs.python.org/issue?@action=redirect&bpo=40967])
- El método `unescape()` en la clase **`html.parser.HTMLParser`** se ha eliminado (quedó obsoleto desde Python 3.4). **`html.unescape()`** debe usarse para convertir referencias de caracteres a los caracteres Unicode correspondientes.

Portar a Python 3.9

Esta sección enumera los cambios descritos anteriormente y otras correcciones de errores que pueden requerir cambios en su código.

Cambios en la API de Python

- **`__import__()`** y **`importlib.util.resolve_name()`** ahora lanzan **`ImportError`** donde anteriormente lanzaban **`ValueError`**. Las personas que llaman que detectan el tipo de excepción específico y son compatibles con Python 3.9 y versiones anteriores deberán detectar ambas usando `except (ImportError, ValueError):`.
- Los scripts de activación de **`venv`** ya no son casos especiales cuando `__VENV_PROMPT__` se establece en `""`.
- The **`select.epoll.unregister()`** method no longer ignores the **`EBADF`** error. (Contributed by Victor Stinner in bpo-39239 [https://bugs.python.org/issue?@action=redirect&bpo=39239].)
- Se ha eliminado el parámetro *compresslevel* **`bz2.BZ2File`** se convirtió solo en un parámetro de llave, ya que *buffering* se quitó. (Contribuido por Victor Stinner en bpo-39357 [https://bugs.python.org/issue?@action=redirect&bpo=39357].)
- AST simplificado para suscripción. Los índices simples se representarán por su valor, los segmentos extendidos se representarán como tuplas. `Index(value)` devolverá un `value` en sí mismo, `ExtSlice(slices)` devolverá `Tuple(slices, Load())`. (Contribuido por Serhiy Storchaka en bpo-34822 [https://bugs.python.org/issue?@action=redirect&bpo=34822].)
- El módulo **`importlib`** ahora ignora la variable de entorno **`PYTHONCASEOK`** cuando se utilizan las opciones de línea de comando **`-E`** o **`-I`**.
- El parámetro *encoding* se ha agregado a las clases **`ftplib.FTP`** y **`ftplib.FTP_TLS`** como un parámetro de solo palabra clave, y la codificación predeterminada se cambia de Latin-1 a UTF-8 para seguir: rfc: *2640*.
- **`asyncio.loop.shutdown_default_executor()`** se ha agregado a **`AbstractEventLoop`**, lo que significa que los bucles de eventos alternativos que heredan de él deben tener definido este método. (Contribuido por Kyle Stanley en bpo-34037 [https://bugs.python.org/issue?@action=redirect&bpo=34037].)
- Los valores constantes de los indicadores futuros en el módulo **`__future__`** se actualizan para evitar la colisión con los indicadores del compilador. Anteriormente, `PyCF_ALLOW_TOP_LEVEL_AWAIT` chocaba con

`CO_FUTURE_DIVISION`. (Contribuido por Batuhan Taskaya en bpo-39562 [https://bugs.python.org/issue?@action=redirect&bpo=39562])

- `array('u')` now uses **`wchar_t`** as C type instead of `Py_UNICODE`. This change doesn't affect to its behavior because `Py_UNICODE` is alias of **`wchar_t`** since Python 3.3. (Contributed by Inada Naoki in bpo-34538 [https://bugs.python.org/issue?@action=redirect&bpo=34538].)
- La API **`logging.getLogger()`** ahora devuelve el registrador raíz cuando se le pasó el nombre `'root'`, mientras que anteriormente devolvía un registrador no raíz llamado `'root'`. Esto podría afectar los casos en los que el código de usuario desea explícitamente un registrador no root llamado `'root'`, o crea una instancia de un registrador usando `logging.getLogger(__name__)` en algún módulo de nivel superior llamado `'root.py'`. (Contribuido por Vinay Sajip en bpo-37742 [https://bugs.python.org/issue?@action=redirect&bpo=37742].)
- Division handling of **`PurePath`** now returns **`NotImplemented`** instead of raising a **`TypeError`** when passed something other than an instance of `str` or **`PurePath`**. This allows creating compatible classes that don't inherit from those mentioned types. (Contributed by Roger Aiudi in bpo-34775 [https://bugs.python.org/issue?@action=redirect&bpo=34775]).
- A partir de Python 3.9.5, el módulo **`ipaddress`** ya no acepta ceros a la izquierda en las cadenas de direcciones IPv4. Los ceros iniciales son ambiguos y algunas bibliotecas los interpretan como notación octal. Por ejemplo, la función heredada **`socket.inet_aton()`** trata los ceros iniciales como una notación octal. La implementación glibc del **`inet_pton()`** moderno no acepta ceros a la izquierda. (Contribuido por Christian Heimes en bpo-36384 [https://bugs.python.org/issue?@action=redirect&bpo=36384]).
- **`codecs.lookup()`** ahora normaliza el nombre de codificación de la misma manera que **`encodings.normalize_encoding()`**, excepto que **`codecs.lookup()`** también convierte el nombre a minúsculas. Por ejemplo, el nombre de codificación `"latex+latin1"` ahora está normalizado a `"latex_latin1"`.

(Contribuido por Jordon Xu en bpo-37751 [https://bugs.python.org/issue?@action=redirect&bpo=37751].)

Cambios en la API de C

- Las instancias de heap-allocated types (como las creadas con **`PyType_FromSpec()`** y API similares) contienen una referencia a su objeto de tipo desde Python 3.8. Como se indica en los «Cambios en la API C» de Python 3.8, para la gran mayoría de los casos, no debería haber efectos secundarios, pero para los tipos que tienen una función personalizada **`tp_traverse`**, asegúrese de que todas las funciones `tp_traverse` personalizadas de los tipos asignados al montón visitan el tipo del objeto.

  Ejemplo:

  ```
  int
  foo_traverse(PyObject *self, visitproc visit,
  {
  // Rest of the traverse function
  #if PY_VERSION_HEX >= 0x03090000
      // This was not needed before Python 3.9
      Py_VISIT(Py_TYPE(self));
  #endif
  }
  ```

  Si su función transversal delega en `tp_traverse` de su clase base (u otro tipo), asegúrese de que `Py_TYPE(self)` se visite solo una vez. Tenga en cuenta que solo se espera que heap type visite el tipo en `tp_traverse`.

  Por ejemplo, si su función `tp_traverse` incluye:

  ```
  base->tp_traverse(self, visit, arg)
  ```

  Luego añade:

  ```
  #if PY_VERSION_HEX >= 0x03090000
      // This was not needed before Python 3.9
  ```

```
    if (base->tp_flags & Py_TPFLAGS_HEAPTYPE)
        // a heap type's tp_traverse already
    } else {
        Py_VISIT(Py_TYPE(self));
    }
#else
```

(Consulte bpo-35810 [https://bugs.python.org/issue?@action=redirect&bpo=35810] y bpo-40217 [https://bugs.python.org/issue?@action=redirect&bpo=40217] para obtener más información).

- Las funciones `PyEval_CallObject`, `PyEval_CallFunction`, `PyEval_CallMethod` y `PyEval_CallObjectWithKeywords` están en desuso. Utilice **`PyObject_Call()`** y sus variantes en su lugar. (Ver más detalles en bpo-29548 [https://bugs.python.org/issue?@action=redirect&bpo=29548].)

Cambios en el código de bytes de CPython

- The **`LOAD_ASSERTION_ERROR`** opcode was added for handling the **`assert`** statement. Previously, the assert statement would not work correctly if the **`AssertionError`** exception was being shadowed. (Contributed by Zackery Spytz in bpo-34880 [https://bugs.python.org/issue?@action=redirect&bpo=34880].)
- El código de operación **`COMPARE_OP`** se dividió en cuatro instrucciones distintas:
    - `COMPARE_OP` para comparaciones enriquecidas
    - `IS_OP` para pruebas "es" y "no es"
    - `CONTAINS_OP` para pruebas "in" y "not in"
    - `JUMP_IF_NOT_EXC_MATCH` para verificar las excepciones en las declaraciones "try-except".

  (Contribuido por Mark Shannon en bpo-39156 [https://bugs.python.org/issue?@action=redirect&bpo=39156].)

Construir cambios

- Added `--with-platlibdir` option to the `configure` script: name of the platform-specific library directory, stored in the new **`sys.platlibdir`** attribute. See **`sys.platlibdir`** attribute for more information. (Contributed by Jan Matějek, Matěj Cepl, Charalampos Stratakis and Victor Stinner in bpo-1294959 [https://bugs.python.org/issue?@action=redirect&bpo=1294959].)
- Se ha eliminado la macro de compilación especial `COUNT_ALLOCS`. (Contribuido por Victor Stinner en bpo-39489 [https://bugs.python.org/issue?@action=redirect&bpo=39489].)
- On non-Windows platforms, the **`setenv()`** and **`unsetenv()`** functions are now required to build Python. (Contributed by Victor Stinner in bpo-39395 [https://bugs.python.org/issue?@action=redirect&bpo=39395].)
- En plataformas que no son de Windows, la creación de instaladores `bdist_wininst` ahora no es compatible oficialmente. (Consulte bpo-10945 [https://bugs.python.org/issue?@action=redirect&bpo=10945] para obtener más detalles).
- Al compilar Python en macOS desde la fuente, `_tkinter` ahora se vincula con marcos Tcl y Tk que no son del sistema si están instalados en `/Library/Frameworks`, como había sido el caso en versiones anteriores de macOS. Si un SDK de macOS está configurado explícitamente, mediante **`--enable-universalsdk`** o `-isysroot`, solo se busca en el SDK. El comportamiento predeterminado aún se puede anular con *–with-tcltk-includes* y *–with-tcltk-libs*. (Contribuido por Ned Deily en bpo-34956 [https://bugs.python.org/issue?@action=redirect&bpo=34956].)
- Python ahora se puede construir para Windows 10 ARM64. (Contribuido por Steve Dower en bpo-33125 [https://bugs.python.org/issue?@action=redirect&bpo=33125].)
- Algunas pruebas individuales ahora se omiten cuando se usa `--pgo`. Las pruebas en cuestión aumentaron significativamente el tiempo de la tarea PGO y probablemente no ayudaron a mejorar la optimización del ejecutable final. Esto acelera la tarea en un factor de aproximadamente 15

veces. La ejecución del conjunto de pruebas unitarias completo es lenta. Este cambio puede resultar en una compilación ligeramente menos optimizada ya que no se ejecutarán tantas ramas de código. Si está dispuesto a esperar la compilación mucho más lenta, el comportamiento anterior se puede restaurar usando `./configure [..] PROFILE_TASK="-m test --pgo-extended"`. No garantizamos qué conjunto de tareas de PGO produce una compilación más rápida. Los usuarios interesados deben ejecutar sus propios puntos de referencia relevantes, ya que los resultados pueden depender del entorno, la carga de trabajo y la cadena de herramientas del compilador. (Consulte bpo-36044 [https://bugs.python.org/issue?@action=redirect&bpo=36044] y bpo-37707 [https://bugs.python.org/issue?@action=redirect&bpo=37707] para obtener más detalles).

Cambios en la API de C

Nuevas características

- **PEP 573** [https://peps.python.org/pep-0573/]: Added **`PyType_FromModuleAndSpec()`** to associate a module with a class; **`PyType_GetModule()`** and **`PyType_GetModuleState()`** to retrieve the module and its state; and **`PyCMethod`** and **`METH_METHOD`** to allow a method to access the class it was defined in. (Contributed by Marcel Plch and Petr Viktorin in bpo-38787 [https://bugs.python.org/issue?@action=redirect&bpo=38787].)

- Se agregó la función **`PyFrame_GetCode()`**: obtenga un código de marco. Se agregó la función **`PyFrame_GetBack()`**: obtenga el siguiente marco exterior del marco. (Contribuido por Victor Stinner en bpo-40421 [https://bugs.python.org/issue?@action=redirect&bpo=40421].)

- Se agregó **`PyFrame_GetLineNumber()`** a la API C limitada. (Contribuido por Victor Stinner en bpo-40421 [https://bugs.python.org/issue?@action=redirect&bpo=40421].)

- Se agregaron funciones

`PyThreadState_GetInterpreter()` y **`PyInterpreterState_Get()`** para obtener el intérprete. Se agregó la función **`PyThreadState_GetFrame()`** para obtener el marco actual de un estado de subproceso de Python. Se agregó la función **`PyThreadState_GetID()`**: obtenga el identificador único de un estado de subproceso de Python. (Contribuido por Victor Stinner en bpo-39947 [https://bugs.python.org/issue?@action=redirect&bpo=39947].)

- Se agregó una nueva función pública **`PyObject_CallNoArgs()`** a la API de C, que llama a un objeto de Python invocable sin ningún argumento. Es la forma más eficiente de llamar a un objeto de Python invocable sin ningún argumento. (Contribuido por Victor Stinner en bpo-37194 [https://bugs.python.org/issue?@action=redirect&bpo=37194].)

- Cambios en la API C limitada (si se define la macro `Py_LIMITED_API`):

  - Proporcione **`Py_EnterRecursiveCall()`** y **`Py_LeaveRecursiveCall()`** como funciones regulares para la API limitada. Anteriormente, se definían como macros, pero estas macros no se compilaban con la API C limitada que no puede acceder al campo `PyThreadState.recursion_depth` (la estructura es opaca en la API C limitada).
  - `PyObject_INIT()` y `PyObject_INIT_VAR()` se convierten en funciones «opacas» habituales para ocultar los detalles de la implementación.

  (Contribuido por Victor Stinner en bpo-38644 [https://bugs.python.org/issue?@action=redirect&bpo=38644] y bpo-39542 [https://bugs.python.org/issue?@action=redirect&bpo=39542].)

- The **`PyModule_AddType()`** function is added to help adding a type to a module. (Contributed by Donghee Na in bpo-40024 [https://bugs.python.org/issue?@action=redirect&bpo=40024].)

- Se agregaron las funciones **`PyObject_GC_IsTracked()`** y

`PyObject_GC_IsFinalized()` a la API pública para permitir consultar si los objetos de Python se están rastreando actualmente o si ya han sido finalizados por el recolector de basura, respectivamente. (Contribuido por Pablo Galindo Salgado en bpo-40241 [https://bugs.python.org/issue?@action=redirect&bpo=40241].)

- Added **`_PyObject_FunctionStr()`** to get a user-friendly string representation of a function-like object. (Patch by Jeroen Demeyer in bpo-37645 [https://bugs.python.org/issue?@action=redirect&bpo=37645].)
- Se agregó **`PyObject_CallOneArg()`** para llamar a un objeto con un argumento posicional (parche de Jeroen Demeyer en bpo-37483 [https://bugs.python.org/issue?@action=redirect&bpo=37483]).

Portar a Python 3.9

- `PyInterpreterState.eval_frame` (**PEP 523** [https://peps.python.org/pep-0523/]) ahora requiere un nuevo parámetro *tstate* obligatorio (`PyThreadState*`). (Contribuido por Victor Stinner en bpo-38500 [https://bugs.python.org/issue?@action=redirect&bpo=38500].)
- Módulos de extensión: **`m_traverse`**, **`m_clear`** y **`m_free`** Las funciones de **`PyModuleDef`** ya no se llaman si el estado del módulo era solicitado pero aún no asignado. Este es el caso inmediatamente después de que se crea el módulo y antes de que se ejecute (función **`Py_mod_exec`**). Más precisamente, estas funciones no se llaman si **`m_size`** es mayor que 0 y el estado del módulo (como lo devuelve **`PyModule_GetState()`**) es `NULL`.

  Los módulos de extensión sin estado de módulo (`m_size <= 0`) no se ven afectados.
- Si se llama a **`Py_AddPendingCall()`** en un subinterpretador, la función ahora está programada para ser llamada desde el subinterpretador, en lugar de ser llamada desde el intérprete principal. Cada subinterpretador ahora

tiene su propia lista de llamadas programadas. (Contribuido por Victor Stinner en bpo-39984 [https://bugs.python.org/issue?@action=redirect&bpo=39984].)

- El registro de Windows ya no se usa para inicializar **`sys.path`** cuando se usa la opción `-E` (si **`PyConfig.use_environment`** está configurado en `0`). Esto es importante al incrustar Python en Windows. (Contribuido por Zackery Spytz en bpo-8901 [https://bugs.python.org/issue?@action=redirect&bpo=8901].)

- La variable global **`PyStructSequence_UnnamedField`** ahora es una constante y se refiere a una cadena constante. (Contribuido por Serhiy Storchaka en bpo-38650 [https://bugs.python.org/issue?@action=redirect&bpo=38650].)

- The **`PyGC_Head`** structure is now opaque. It is only defined in the internal C API (`pycore_gc.h`). (Contributed by Victor Stinner in bpo-40241 [https://bugs.python.org/issue?@action=redirect&bpo=40241].)

- The `Py_UNICODE_COPY`, `Py_UNICODE_FILL`, `PyUnicode_WSTR_LENGTH`, **`PyUnicode_FromUnicode()`**, **`PyUnicode_AsUnicode()`**, `_PyUnicode_AsUnicode`, and **`PyUnicode_AsUnicodeAndSize()`** are marked as deprecated in C. They have been deprecated by **PEP 393** [https://peps.python.org/pep-0393/] since Python 3.3. (Contributed by Inada Naoki in bpo-36346 [https://bugs.python.org/issue?@action=redirect&bpo=36346].)

- La función **`Py_FatalError()`** se reemplaza con una macro que registra automáticamente el nombre de la función actual, a menos que se defina la macro `Py_LIMITED_API`. (Contribuido por Victor Stinner en bpo-39882 [https://bugs.python.org/issue?@action=redirect&bpo=39882].)

- El protocolo vectorcall ahora requiere que la persona que llama solo pase cadenas como nombres de palabras clave. (Consulte bpo-37540 [https://bugs.python.org/issue?@action=redirect&bpo=37540] para obtener más información).

- Los detalles de implementación de una serie de macros y funciones ahora están ocultos:

  - La macro `PyObject_IS_GC()` se convirtió en una función.
  - The `PyObject_NEW()` macro becomes an alias to the `PyObject_New` macro, and the `PyObject_NEW_VAR()` macro becomes an alias to the `PyObject_NewVar` macro. They no longer access directly the `PyTypeObject.tp_basicsize` member.
  - `PyObject_GET_WEAKREFS_LISTPTR()` macro was converted to a function: the macro accessed directly the `PyTypeObject.tp_weaklistoffset` member.
  - La macro `PyObject_CheckBuffer()` se convirtió en una función: la macro accedió directamente al miembro `PyTypeObject.tp_as_buffer`.
  - `PyIndex_Check()` ahora siempre se declara como una función opaca para ocultar los detalles de implementación: se eliminó la macro `PyIndex_Check()`. La macro accedió directamente al miembro `PyTypeObject.tp_as_number`.

  (Consulte bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170] para obtener más detalles).

Remoto

- Se excluyeron las macros `PyFPE_START_PROTECT()` y `PyFPE_END_PROTECT()` de `pyfpe.h` de la API C limitada. (Contribuido por Victor Stinner en bpo-38835 [https://bugs.python.org/issue?@action=redirect&bpo=38835].)

- Se ha eliminado la ranura `tp_print` de PyTypeObject. Se usó para imprimir objetos en archivos en Python 2.7 y antes. Desde Python 3.0, se ha ignorado y no se ha utilizado. (Contribuido por Jeroen Demeyer en bpo-36974 [https://bugs.python.org/issue?@action=redirect&bpo=36974].)

- Cambios en la API C limitada (si se define la macro `Py_LIMITED_API`):

  - Se excluyeron las siguientes funciones de la API C limitada:
    - `PyThreadState_DeleteCurrent()` (Contribuido por Joannah Nanjekye en bpo-37878 [https://bugs.python.org/issue?@action=redirect&bpo=37878].)
    - `_Py_CheckRecursionLimit`
    - `_Py_NewReference()`
    - `_Py_ForgetReference()`
    - `_PyTraceMalloc_NewReference()`
    - `_Py_GetRefTotal()`
    - El mecanismo de papelera que nunca funcionó en la API C limitada.
    - `PyTrash_UNWIND_LEVEL`
    - `Py_TRASHCAN_BEGIN_CONDITION`
    - `Py_TRASHCAN_BEGIN`
    - `Py_TRASHCAN_END`
    - `Py_TRASHCAN_SAFE_BEGIN`
    - `Py_TRASHCAN_SAFE_END`
  - Se movieron las siguientes funciones y definiciones a la API de C interna:
    - `_PyDebug_PrintTotalRefs()`
    - `_Py_PrintReferences()`
    - `_Py_PrintReferenceAddresses()`
    - `_Py_tracemalloc_config`
    - `_Py_AddToAllObjects()` (específico para `Py_TRACE_REFS` acumulación)

  (Contribuido por Victor Stinner en bpo-38644 [https://bugs.python.org/issue?@action=redirect&bpo=38644] y bpo-39542 [https://bugs.python.org/issue?@action=redirect&bpo=39542].)

- Se eliminó el gancho `_PyRuntime.getframe` y se eliminó la macro `_PyThreadState_GetFrame`, que era un alias de `_PyRuntime.getframe`. Solo fueron expuestos por la API C interna. También se eliminó el tipo `PyThreadFrameGetter`. (Contribuido por Victor Stinner en bpo-39946 [https://bugs.python.org/issue?@action=redirect&bpo=39946].)

- Se eliminaron las siguientes funciones de la API de C. Llame a **`PyGC_Collect()`** explícitamente para borrar todas las listas libres. (Contribuido por Inada Naoki y Victor Stinner en bpo-37340 [https://bugs.python.org/issue?@action=redirect&bpo=37340], bpo-38896 [https://bugs.python.org/issue?@action=redirect&bpo=38896] y bpo-40428 [https://bugs.python.org/issue?@action=redirect&bpo=40428].)
    - ○ `PyAsyncGen_ClearFreeLists()`
    - ○ `PyContext_ClearFreeList()`
    - ○ `PyDict_ClearFreeList()`
    - ○ `PyFloat_ClearFreeList()`
    - ○ `PyFrame_ClearFreeList()`
    - ○ `PyList_ClearFreeList()`
    - ○ `PyMethod_ClearFreeList()` y `PyCFunction_ClearFreeList()`: se han eliminado las listas libres de objetos de método enlazados.
    - ○ `PySet_ClearFreeList()`: la lista de conjuntos libres se ha eliminado en Python 3.4.
    - ○ `PyTuple_ClearFreeList()`
    - ○ `PyUnicode_ClearFreeList()`: la lista libre de Unicode se ha eliminado en Python 3.3.

- Función `_PyUnicode_ClearStaticStrings()` eliminada. (Contribuido por Victor Stinner en bpo-39465 [https://bugs.python.org/issue?@action=redirect&bpo=39465].)

- Eliminado `Py_UNICODE_MATCH`. Ha sido obsoleto por **PEP 393** [https://peps.python.org/pep-0393/] y roto desde Python 3.3. En su lugar, se puede utilizar la función **`PyUnicode_Tailmatch()`**. (Contribuido por Inada Naoki en bpo-36346 [https://bugs.python.org/issue?@action=redirect&bpo=36346].)

- Archivos de encabezado limpios de interfaces definidas pero sin implementación. Los símbolos API públicos que se eliminan son: `_PyBytes_InsertThousandsGroupingLocale`, `_PyBytes_InsertThousandsGrouping`, `_Py_InitializeFromArgs`, `_Py_InitializeFromWideArgs`, `_PyFloat_Repr`, `_PyFloat_Digits`, `_PyFloat_DigitsInit`, `PyFrame_ExtendStack`, `_PyAIterWrapper_Type`, `PyNullImporter_Type`, `PyCmpWrapper_Type`, `PySortWrapper_Type`, `PyNoArgsFunction`. (Contribuido por Pablo Galindo Salgado en bpo-39372 [https://bugs.python.org/issue?@action=redirect&bpo=39372].)

Cambios notables en Python 3.9.1

mecanografía

El comportamiento de **`typing.Literal`** se modificó para cumplir con **PEP 586** [https://peps.python.org/pep-0586/] y para coincidir con el comportamiento de los verificadores de tipo estático especificados en el PEP.

1. `Literal` ahora elimina los parámetros duplicados.
2. Las comparaciones de igualdad entre objetos `Literal` ahora son independientes del orden.
3. Las comparaciones de `Literal` ahora respetan los tipos. Por ejemplo, `Literal[0] == Literal[False]` evaluado previamente como `True`. Ahora es `False`. Para respaldar este cambio, la caché de tipos utilizada internamente ahora admite tipos diferenciados.
4. Los objetos `Literal` ahora lanzarán una excepción **`TypeError`** durante las comparaciones de igualdad si alguno de sus parámetros no es hashable. Tenga en cuenta que declarar `Literal` con parámetros mutables no arrojará un error:

```
>>> from typing import Literal
>>> Literal[{0}]
>>> Literal[{0}] == Literal[{False}]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'set'
```

(Contribuido por Yurii Karabas en bpo-42345 [https://bugs.python.org/issue?@action=redirect&bpo=42345].)

Compatibilidad con macOS 11.0 (Big Sur) y Apple Silicon Mac

A partir de 3.9.1, Python ahora es totalmente compatible con la construcción y ejecución en macOS 11.0 (Big Sur) y en Apple Silicon Macs (basado en la arquitectura `ARM64`). Una nueva variante de compilación universal, `universal2`, ahora está disponible para admitir de forma nativa tanto `ARM64` como `Intel 64` en un conjunto de ejecutables. Los binarios ahora también se pueden construir en versiones actuales de macOS para ser implementados en una variedad de versiones anteriores de macOS (probadas a 10.9) mientras hacen que algunas funciones y opciones más nuevas del sistema operativo estén disponibles condicionalmente según la versión del sistema operativo en uso en tiempo de ejecución («vinculación débil»).

(Contribuido por Ronald Oussoren y Lawrence D'Anna en bpo-41100 [https://bugs.python.org/issue?@action=redirect&bpo=41100].)

Cambios notables en Python 3.9.2

colecciones.abc

`collections.abc.Callable` genérico ahora aplana los parámetros de tipo, similar a lo que hace actualmente **`typing.Callable`**. Esto significa que `collections.abc.Callable[[int, str], str]` tendrá `__args__` de `(int, str, str)`; anteriormente esto era `([int, str], str)`. Para permitir este cambio, **`types.GenericAlias`** ahora puede ser subclasificado, y se devolverá una subclase al subíndice el tipo **`collections.abc.Callable`**. El código que accede a los argumentos a través de **`typing.get_args()`** o `__args__` debe tener en cuenta este cambio. Se puede emitir un **`DeprecationWarning`** para formas no válidas de parametrizar **`collections.abc.Callable`** que pueden haber pasado

silenciosamente en Python 3.9.1. Este **`DeprecationWarning`** se convertirá en un **`TypeError`** en Python 3.10. (Contribuido por Ken Jin en bpo-42195 [https://bugs.python.org/issue?@action=redirect&bpo=42195].)

urllib.parse

Earlier Python versions allowed using both `;` and `&` as query parameter separators in **`urllib.parse.parse_qs()`** and **`urllib.parse.parse_qsl()`**. Due to security concerns, and to conform with newer W3C recommendations, this has been changed to allow only a single separator key, with `&` as the default. This change also affects **`cgi.parse()`** and **`cgi.parse_multipart()`** as they use the affected functions internally. For more details, please see their respective documentation. (Contributed by Adam Goldschmidt, Senthil Kumaran and Ken Jin in bpo-42967 [https://bugs.python.org/issue?@action=redirect&bpo=42967].)

Notable changes in Python 3.9.3

A security fix alters the **`ftplib.FTP`** behavior to not trust the IPv4 address sent from the remote server when setting up a passive data channel. We reuse the ftp server IP address instead. For unusual code requiring the old behavior, set a `trust_server_pasv_ipv4_address` attribute on your FTP instance to `True`. (See gh-87451 [https://github.com/python/cpython/issues/87451])

Notable changes in Python 3.9.5

urllib.parse

The presence of newline or tab characters in parts of a URL allows for some forms of attacks. Following the WHATWG specification that updates **RFC 3986** [https://datatracker.ietf.org/doc/html/rfc3986.html], ASCII newline `\n`, `\r` and tab `\t` characters are stripped from the URL by the parser in **`urllib.parse`** preventing such attacks. The removal characters are controlled by a new module level variable `urllib.parse._UNSAFE_URL_BYTES_TO_REMOVE`. (See gh-88048 [https://github.com/python/cpython/issues/88048])

Notable security feature in 3.9.14

Converting between **`int`** and **`str`** in bases other than 2 (binary), 4, 8 (octal), 16 (hexadecimal), or 32 such as base 10 (decimal) now raises a **`ValueError`** if the number of digits in string form is above a limit to avoid potential denial of service attacks due to the algorithmic complexity. This is a mitigation for **CVE 2020-10735** [https://www.cve.org/CVERecord?id=CVE-2020-10735]. This limit can be configured or disabled by environment variable, command line flag, or **`sys`** APIs. See the integer string conversion length limitation documentation. The default limit is 4300 digits in string form.

Notable changes in 3.9.17

tarfile

- The extraction methods in **`tarfile`**, and **`shutil.unpack_archive()`**, have a new a *filter* argument that allows limiting tar features than may be surprising or dangerous, such as creating files outside the destination directory. See Extraction filters for details. In Python 3.12, use without the *filter* argument will show a **`DeprecationWarning`**. In Python 3.14, the default will switch to `'data'`. (Contributed by Petr Viktorin in **PEP 706** [https://peps.python.org/pep-0706/].)

Qué hay de nuevo en Python 3.8

Editor:

Raymond Hettinger

Este artículo explica las nuevas características de Python 3.8, en comparación con 3.7. Python 3.8 se lanzó el 14 de octubre de 2019. Para obtener detalles completos, consulte changelog.

Resumen – Aspectos destacados de la versión

Nuevas características

Expresiones de asignación

La nueva sintaxis `:=` asigna valores a variables como parte de una expresión más grande. Se le conoce cariñosamente como «el operador morsa» debido a su parecido con los ojos y colmillos de una morsa [https://en.wikipedia.org/wiki/Walrus#/media/File:Pacific_Walrus_-_Bull_(8247646168).jpg].

En el siguiente ejemplo, la expresión de asignación ayuda evitando que se tenga que llamar a `len()` dos veces:

```
if (n := len(a)) > 10:
    print(f"List is too long ({n} elements, expected <=
```

Un beneficio similar surge durante la búsqueda de coincidencias mediante expresiones regulares donde los objetos de coincidencias se necesitan dos veces, una para comprobar si se produjo una coincidencia y otra para extraer un subgrupo:

```
discount = 0.0
if (mo := re.search(r'(\d+)% discount', advertisement)):
    discount = float(mo.group(1)) / 100.0
```

El operador también es útil en bucles while que calculan un valor para comprobar la terminación del bucle y posteriormente necesitan nuevamente ese mismo valor en el cuerpo del bucle:

```
# Loop over fixed length blocks
while (block := f.read(256)) != '':
    process(block)
```

Otra motivación de uso surge en las listas por comprensión en las que un valor calculado en una condición de filtrado también se necesita en el cuerpo de la expresión:

```
[clean_name.title() for name in names
 if (clean_name := normalize('NFC', name)) in allowed_na
```

Intenta limitar el uso del «operador morsa» a aquellos casos en los que reduce la complejidad y mejora la legibilidad del código.

Consultar **PEP 572** [https://peps.python.org/pep-0572/] para obtener una descripción completa.

(Contribución de Emily Morehouse en bpo-35224 [https://bugs.python.org/issue?@action=redirect&bpo=35224].)

Parámetros solo posicionales

There is a new function parameter syntax `/` to indicate that some function parameters must be specified positionally and cannot be used as keyword arguments. This is the same notation shown by `help()` for C functions annotated with Larry Hastings” Argument Clinic [https://devguide.python.org/development-tools/clinic/] tool.

En el siguiente ejemplo, los parámetros *a* y *b* son solo posicionales, mientras que *c* o *d* pueden ser posicionales o por palabra clave y *e* o *f* deben proporcionarse por palabra clave exclusivamente:

```
def f(a, b, /, c, d, *, e, f):
    print(a, b, c, d, e, f)
```

La siguiente es una invocación válida:

```
f(10, 20, 30, d=40, e=50, f=60)
```

Sin embargo, estas son invocaciones inválidas:

```
f(10, b=20, c=30, d=40, e=50, f=60)   # b cannot be a ke
f(10, 20, 30, 40, 50, f=60)           # e must be a keyw
```

Un caso de uso de esta notación es permitir que las funciones puras de Python emulen completamente los comportamientos de las funciones codificadas en C existentes. Por ejemplo, la función incorporada **`divmod()`** no acepta argumentos por palabra clave:

```
def divmod(a, b, /):
    "Emulate the built in divmod() function"
    return (a // b, a % b)
```

Otro caso de uso es excluir los argumentos por palabra clave cuando el nombre del parámetro no es útil. Por ejemplo, la función incorporada **`len()`** tiene la firma `len(obj, /)`. Esto excluye llamadas inoportunas como:

```
len(obj='hello')  # The "obj" keyword argument impairs r
```

Un beneficio adicional de marcar un parámetro como solo posicional es que permite cambiar el nombre del parámetro en el futuro sin riesgo de romper el código del cliente. Por ejemplo, en el módulo **`statistics`**, el nombre del parámetro *dist* puede cambiarse en el futuro. Esto es posible gracias a la siguiente especificación de función:

```
def quantiles(dist, /, *, n=4, method='exclusive')
    ...
```

Dado que los parámetros a la izquierda de `/` no se exponen como posibles palabras clave, los nombres de los parámetros permanecen disponibles para su uso en `**kwargs`:

```
>>> def f(a, b, /, **kwargs):
...     print(a, b, kwargs)
...
>>> f(10, 20, a=1, b=2, c=3)         # a and b are used
10 20 {'a': 1, 'b': 2, 'c': 3}
```

Esto simplifica enormemente la implementación de funciones y métodos que necesitan aceptar argumentos por palabra clave arbitrarios. Por ejemplo, aquí hay un extracto del código del módulo `collections`:

```
class Counter(dict):

    def __init__(self, iterable=None, /, **kwds):
        # Note "iterable" is a possible keyword argument
```

Consultar **PEP 570** [https://peps.python.org/pep-0570/] para obtener una descripción completa.

(Contribución de Pablo Galindo en bpo-36540 [https://bugs.python.org/issue?@action=redirect&bpo=36540].)

Caché del sistema de archivos paralelo para archivos de bytecode compilados

La nueva configuración `PYTHONPYCACHEPREFIX` (también disponible mediante la opción `-X` `pycache_prefix`) configura la caché implícita de bytecode para que use un árbol del sistema de archivos paralelo separado, en lugar de los subdirectorios `__pycache__` predeterminados dentro cada directorio de origen.

La ubicación de la caché se define en `sys.pycache_prefix` (`None` indica la ubicación predeterminada en los subdirectorios `__pycache__`).

(Contribución de Carl Meyer en bpo-33499 [https://bugs.python.org/issue?@action=redirect&bpo=33499].)

La compilación de depuración usa la misma ABI que la compilación de lanzamiento

The ABI of Python debug builds is now compatible with Python release builds. On Unix, when Python is built in debug mode, it is now possible to load C extensions built in release mode and C extensions built using the stable ABI. The inverse is not true, as debug builds expose additional symbols not available in release builds.

Defining the `Py_DEBUG` macro no longer implies the `Py_TRACE_REFS` macro, which introduces the only ABI incompatibility. The `Py_TRACE_REFS` macro, which adds the **`sys.getobjects()`** function and the **`PYTHONDUMPREFS`** environment variable, can be set using the new **`./configure --with-trace-refs`** build option. (Contributed by Victor Stinner in bpo-36465 [https://bugs.python.org/issue?@action=redirect&bpo=36465].)

En Unix, las extensiones en C ya no están enlazadas a libpython, excepto en Android y Cygwin. Ahora es posible que un intérprete Python enlazado estáticamente cargue una extensión de C creada con una biblioteca dinámica de Python. (Contribución de Victor Stinner en bpo-21536 [https://bugs.python.org/issue?@action=redirect&bpo=21536].)

En Unix, cuando Python se compila en modo de depuración, la importación ahora también busca extensiones C compiladas en modo de lanzamiento y extensiones C compiladas con la ABI estable. (Contribución de Victor Stinner en bpo-36722 [https://bugs.python.org/issue?@action=redirect&bpo=36722].)

Para integrar Python en una aplicación, se debe pasar una nueva opción `--embed` a `python3-config --libs --embed` para obtener `-lpython3.8` (enlaza la aplicación a libpython). Para ser compatible con 3.8 y versiones anteriores, prueba primero con `python3-config --libs --embed` y vuelve a usar `python3-config --libs` (sin `--embed`) si falla el comando anterior.

Agrega un módulo pkg-config `python-3.8-embed` para integrar Python en una aplicación: `pkg-config python-3.8-embed --libs` incluye `-lpython3.8`. Para que sea compatible con 3.8 y versiones anteriores, primero intenta usar `pkg-config python-X.Y-embed --libs` y vuelve a `pkg-config python-X.Y --libs` (sin `--embed`) si el comando anterior falla (reemplaza `XY` con la versión de Python).

Por otro lado, `pkg-config python3.8 --libs` ya no contiene `-lpython3.8`. Las extensiones en C no deben estar enlazadas a libpython (excepto en Android y Cygwin, cuyos casos son manejados por el script); este cambio no es retrocompatible a propósito. (Contribución de Victor Stinner en bpo-36721 [https://bugs.python.org/issue?@action=redirect&bpo=36721].)

los f-strings soportan `=` para expresiones autodocumentadas y depuración

Se ha agregado un especificador `=` a los f-strings. Un f-string como `f'{expr=}'` se expandirá al texto de la expresión, seguido de un signo igual y luego la representación de la expresión evaluada. Por ejemplo:

```
>>> import datetime as dt
>>> user = 'eric_idle'
>>> member_since = dt.date(1975, 7, 31)
>>> f'{user=} {member_since=}'
"user='eric_idle' member_since=datetime.date(1975, 7, 31
```

Los especificadores de formato de los f-string usuales permiten un mayor control sobre como se muestra el resultado de la expresión:

```
>>> delta = dt.date.today() - member_since
>>> f'{user=!s}  {delta.days=:,d}'
'user=eric_idle  delta.days=16,075'
```

El especificador `=` mostrará la expresión completa para que se puedan mostrar los cálculos:

```
>>> print(f'{theta=}  {cos(radians(theta))=:.3f}')
theta=30  cos(radians(theta))=0.866
```

(Contribución de Eric V. Smith y Larry Hastings en bpo-36817 [https://bugs.python.org/issue?@action=redirect&bpo=36817].)

PEP 578: Ganchos de auditoría en tiempo de

ejecución de Python

Este PEP agrega un gancho de auditoría y un gancho abierto verificado. Ambos están disponibles desde Python y desde el código nativo, lo que permite que las aplicaciones y los frameworks escritos en código Python puro aprovechen las notificaciones adicionales, al tiempo que permiten a los integradores o administradores de sistemas implementar compilaciones de Python donde la auditoría siempre está habilitada.

Consultar **PEP 578** [https://peps.python.org/pep-0578/] para obtener más detalles.

PEP 587: Configuración de inicialización de Python

El **PEP 587** [https://peps.python.org/pep-0587/] agrega una nueva API de C para configurar la inicialización de Python, proporcionando un control más preciso de toda la configuración y mejores informes de errores.

Nuevas estructuras:

- `PyConfig`
- `PyPreConfig`
- `PyStatus`
- `PyWideStringList`

Nuevas funciones:

- `PyConfig_Clear()`
- `PyConfig_InitIsolatedConfig()`
- `PyConfig_InitPythonConfig()`
- `PyConfig_Read()`
- `PyConfig_SetArgv()`
- `PyConfig_SetBytesArgv()`
- `PyConfig_SetBytesString()`
- `PyConfig_SetString()`
- `PyPreConfig_InitIsolatedConfig()`
- `PyPreConfig_InitPythonConfig()`
- `PyStatus_Error()`
- `PyStatus_Exception()`

- **`PyStatus_Exit()`**
- **`PyStatus_IsError()`**
- **`PyStatus_IsExit()`**
- **`PyStatus_NoMemory()`**
- **`PyStatus_Ok()`**
- **`PyWideStringList_Append()`**
- **`PyWideStringList_Insert()`**
- **`Py_BytesMain()`**
- **`Py_ExitStatusException()`**
- **`Py_InitializeFromConfig()`**
- **`Py_PreInitialize()`**
- **`Py_PreInitializeFromArgs()`**
- **`Py_PreInitializeFromBytesArgs()`**
- **`Py_RunMain()`**

Este PEP también agrega los campos `_PyRuntimeState.preconfig` (tipo **`PyPreConfig`**) y `PyInterpreterState.config` (tipo **`PyConfig`**) a estas estructuras internas. `PyInterpreterState.config` se convierte en la nueva configuración de referencia, reemplazando las variables de configuración globales y otras variables privadas.

Consultar Configuración de inicialización de Python para la documentación.

Consultar **PEP 587** [https://peps.python.org/pep-0587/] para obtener una descripción completa.

(Contribución de Victor Stinner en bpo-36763 [https://bugs.python.org/issue?@action=redirect&bpo=36763].)

PEP 590: Vectorcall: un protocolo de llamada rápida para CPython

El protocolo vectorcall se agrega a la API de Python / C. Está destinado a formalizar las optimizaciones existentes que ya se realizaron para varias clases. Cualquier static type que implemente un invocable puede utilizar este protocolo.

Actualmente es provisional. El objetivo es hacerlo completamente público en Python 3.9.

Consultar **PEP 590** [https://peps.python.org/pep-0590/] para obtener una descripción completa.

(Contribución de Jeroen Demeyer, Mark Shannon y Petr Viktorin en bpo-36974 [https://bugs.python.org/issue?@action=redirect&bpo=36974].)

Protocolo 5 de Pickle con búferes de datos fuera de banda

Cuando `pickle` se usa para transferir grandes cantidades de datos entre procesos de Python, con la finalidad de aprovechar el procesamiento de múltiples núcleos o máquinas, es importante optimizar la transferencia reduciendo las copias en memoria y posiblemente aplicando técnicas personalizadas, como la compresión dependiente de datos.

El protocolo 5 de `pickle` introduce soporte para búferes fuera de banda, donde los datos compatibles con **PEP 3118** [https://peps.python.org/pep-3118/] se pueden transmitir separados del flujo principal de pickle, a discreción de la capa de comunicación.

Consultar **PEP 574** [https://peps.python.org/pep-0574/] para obtener una descripción completa.

(Contribución de Antoine Pitrou en bpo-36785 [https://bugs.python.org/issue?@action=redirect&bpo=36785].)

Otros cambios en el lenguaje

- La declaración `continue` era ilegal en la cláusula `finally` debido a un problema con la implementación. En Python 3.8 se ha eliminado esta restricción. (Contribución de Serhiy Storchaka en bpo-32489 [https://bugs.python.org/issue?@action=redirect&bpo=32489].)
- Los tipos `bool`, `int` y `fractions.Fraction` ahora tienen un método `as_integer_ratio()` como el que se encuentra en `float` y `decimal.Decimal`. Esta extensión

menor de la API hace posible escribir `numerator, denominator = x.as_integer_ratio()` y hacer que funcione con múltiples tipos numéricos. (Contribución de Lisa Roach en bpo-33073 [https://bugs.python.org/issue?@action=redirect&bpo=33073] y de Raymond Hettinger en bpo-37819 [https://bugs.python.org/issue?@action=redirect&bpo=37819].)

- Los constructores de `int`, `float` y `complex` ahora usarán el método especial `__index__()`, si está disponible y el método correspondiente `__int__()`, `__float__()` o `__complex__()` no está disponible. (Contribución de Serhiy Storchaka en bpo-20092 [https://bugs.python.org/issue?@action=redirect&bpo=20092].)

- Agregado soporte para escapes `\N{name}` en `expresiones regulares`:

  ```
  >>> notice = 'Copyright © 2019'
  >>> copyright_year_pattern = re.compile(r'\N{copyri
  >>> int(copyright_year_pattern.search(notice).group
  2019
  ```

  (Contribución de Jonathan Eunice y Serhiy Storchaka in bpo-30688 [https://bugs.python.org/issue?@action=redirect&bpo=30688].)

- Los diccionarios y sus vistas ahora se pueden iterar en orden inverso de inserción usando `reversed()`. (Contribución de Rémi Lapeyre en bpo-33462 [https://bugs.python.org/issue?@action=redirect&bpo=33462].)

- La sintaxis permitida para los nombres por palabra clave en las llamadas a funciones se ha restringido aún más. En particular, `f((keyword)=arg)` ya no está permitido. Nunca se tuvo intención de permitir algo más que un simple nombre en el lado izquierdo de un término de asignación de argumento por palabra clave. (Contribución de Benjamin Peterson en bpo-34641 [https://bugs.python.org/issue?@action=redirect&bpo=34641].)

- El desempaquetado de iterables generalizado en declaraciones `yield` y `return` ya no requiere ser encerrado entre paréntesis. Esto hace que la sintaxis de *yield* y *return* se asemeje más a la sintaxis normal de asignación:

  ```
  >>> def parse(family):
  ...     lastname, *members = family.split()
  ...     return lastname.upper(), *members
  ...
  >>> parse('simpsons homer marge bart lisa maggie')
  ('SIMPSONS', 'homer', 'marge', 'bart', 'lisa', 'mag
  ```

  (Contribución de David Cuthbert y Jordan Chapman en bpo-32117 [https://bugs.python.org/issue?@action=redirect&bpo=32117].)

- Cuando falta una coma en el código, como en `[(10, 20)(30, 40)]`, el compilador muestra un `SyntaxWarning` con una sugerencia útil. Esto representa una mejora con respecto a la implementación previa en la que solo se mostraba un `TypeError` indicando que la primera tupla no era invocable. (Contribución de Serhiy Storchaka en bpo-15248 [https://bugs.python.org/issue?@action=redirect&bpo=15248].)

- Las operaciones aritméticas entre subclases de `datetime.date` o `datetime.datetime` y `datetime.timedelta` ahora retornan una instancia de la subclase, en lugar de la clase base. Esto también afecta al tipo de retorno de las operaciones cuya implementación (directa o indirectamente) usa la aritmética de **`datetime.timedelta`**, como `astimezone()`. (Contribución de Paul Ganssle en bpo-32417 [https://bugs.python.org/issue?@action=redirect&bpo=32417].)

- Cuando el intérprete de Python es interrumpido por Ctrl-C (SIGINT) y la excepción `KeyboardInterrupt` resultante no se detecta, el proceso de Python ahora termina su ejecución a través de una señal SIGINT o con el código de salida correcto, de modo que el proceso que lo invocó puede detectar que murió debido a Ctrl-C. Los shells en POSIX y Windows usan esto para terminar la ejecución de los scripts en sesiones interactivas de forma correcta. (Contribución de Google a través de Gregory P. Smith en bpo-1054041 [https://bugs.python.org/issue?@action=redirect&bpo=1054041].)

- Algunos estilos de programación avanzados requieren actualizar el objeto `types.CodeType` para una función ya existente. Dado que los objetos de código son inmutables, es necesario crear un nuevo objeto de código, que sea modelado a partir del objeto de código existente. Con 19 parámetros, esto era algo tedioso. Ahora, el nuevo método `replace()` hace posible crear un clon con algunos parámetros alterados.

  Aquí hay un ejemplo que modifica la función `statistics.mean()` para evitar que el parámetro *data* se use como un argumento por palabra clave:

  ```
  >>> from statistics import mean
  >>> mean(data=[10, 20, 90])
  40
  >>> mean.__code__ = mean.__code__.replace(co_posonl
  >>> mean(data=[10, 20, 90])
  Traceback (most recent call last):
    ...
  TypeError: mean() got some positional-only argument
  ```

  (Contribución de Victor Stinner en bpo-37032 [https://bugs.python.org/issue?@action=redirect&bpo=37032].)

- Para enteros, la forma de tres argumentos de la función `pow()` ahora permite que el exponente sea negativo en el caso de que la base y el módulo sean primos relativos (coprimos). Luego calcula un inverso modular a la base cuando el exponente es `-1` y una potencia adecuada de ese inverso en el caso de otros exponentes negativos. Por ejemplo, para calcular el inverso multiplicativo [https://es.wikipedia.org/wiki/Inverso_multiplicativo_(aritm%C3%A9tica_modular)] de 38 módulo 137, escribe:

  ```
  >>> pow(38, -1, 137)
  119
  ```

```
>>> 119 * 38 % 137
1
```

Los inversos modulares surgen de la solución de ecuaciones diofánticas lineales [https://es.wikipedia.org/wiki/Ecuaci %C3%B3n_diof%C3%A1ntica]. Por ejemplo, para encontrar soluciones enteras para `4258`x `+ 147`y `= 369`, primero debes reescribirla como `4258`x `≡ 369 (mod 147)` y luego resolver:

```
>>> x = 369 * pow(4258, -1, 147) % 147
>>> y = (4258 * x - 369) // -147
>>> 4258 * x + 147 * y
369
```

(Contribución de Mark Dickinson en bpo-36027 [https:// bugs.python.org/issue?@action=redirect&bpo=36027].)

- Las compresiones de diccionarios se han sincronizado con los literales de diccionario para que primero se calcule la clave y posteriormente el valor:

  ```
  >>> # Dict comprehension
  >>> cast = {input('role? '): input('actor? ') for i
  role? King Arthur
  actor? Chapman
  role? Black Knight
  actor? Cleese

  >>> # Dict literal
  >>> cast = {input('role? '): input('actor? ')}
  role? Sir Robin
  actor? Eric Idle
  ```

  Este orden de ejecución garantizado es especialmente útil en las expresiones de asignación porque las variables asignadas en la expresión de la clave estarán disponibles en la expresión del valor:

  ```
  >>> names = ['Martin von Löwis', 'Łukasz Langa', 'W
  ```

```
>>> {(n := normalize('NFC', name)).casefold() : n f
{'martin von löwis': 'Martin von Löwis',
 'łukasz langa': 'Łukasz Langa',
 'walter dörwald': 'Walter Dörwald'}
```

(Contribución de Jörn Heissler en bpo-35224 [https://bugs.python.org/issue?@action=redirect&bpo=35224].)

- The **`object.__reduce__()`** method can now return a tuple from two to six elements long. Formerly, five was the limit. The new, optional sixth element is a callable with a `(obj, state)` signature. This allows the direct control over the state-updating behavior of a specific object. If not *None*, this callable will have priority over the object's **`__setstate__()`** method. (Contributed by Pierre Glaser and Olivier Grisel in bpo-35900 [https://bugs.python.org/issue?@action=redirect&bpo=35900].)

Nuevos módulos

- El nuevo módulo **`importlib.metadata`** proporciona soporte (provisional) para leer metadatos de paquetes de terceros. Por ejemplo, puede extraer el número de versión de un paquete instalado, la lista de puntos de entrada y más:

```
>>> # Note following example requires that the popu
>>> # package has been installed.
>>>
>>> from importlib.metadata import version, require
>>> version('requests')
'2.22.0'
>>> list(requires('requests'))
['chardet (<3.1.0,>=3.0.2)']
>>> list(files('requests'))[:5]
[PackagePath('requests-2.22.0.dist-info/INSTALLER')
 PackagePath('requests-2.22.0.dist-info/LICENSE'),
 PackagePath('requests-2.22.0.dist-info/METADATA'),
 PackagePath('requests-2.22.0.dist-info/RECORD'),
 PackagePath('requests-2.22.0.dist-info/WHEEL')]
```

(Contribución de Barry Warsaw y Jason R. Coombs en bpo-34632 [https://bugs.python.org/issue?@action=redirect&bpo=34632].)

Módulos mejorados

ast

Los nodos de AST ahora disponen de los atributos `end_lineno` y `end_col_offset`, que proporcionan la localización precisa del final del nodo. (Esto solo se aplica a los nodos que tienen los atributos `lineno` y `col_offset`.)

La nueva función **`ast.get_source_segment()`** retorna el código fuente de un nodo AST específico.

(Contribución de Ivan Levkivskyi en bpo-33416 [https://bugs.python.org/issue?@action=redirect&bpo=33416].)

La función **`ast.parse()`** tiene algunos flags nuevos:

- `type_comments=True` causa que la función retorne el texto de los comentarios de tipo especificados en **PEP 484** [https://peps.python.org/pep-0484/] y **PEP 526** [https://peps.python.org/pep-0526/] asociados con ciertos nodos AST;
- `mode='func_type'` puede usarse para realizar un análisis sintáctico de los «comentarios de tipo de firma» de **PEP 484** [https://peps.python.org/pep-0484/] (retornados por los nodos AST de definición de funciones);
- `feature_version=(3, N)` permite especificar una versión de Python 3 previa. Por ejemplo, `feature_version=(3, 4)` hará que se trate a **`async`** y **`await`** como palabras no reservadas.

(Contribución de Guido van Rossum en bpo-35766 [https://bugs.python.org/issue?@action=redirect&bpo=35766].)

asyncio

`asyncio.run()` ha pasado de la API provisional a la estable. Esta función se puede utilizar para ejecutar una coroutine y retornar el resultado mientras se gestiona automáticamente el bucle de eventos. Por ejemplo:

```
import asyncio

async def main():
    await asyncio.sleep(0)
    return 42

asyncio.run(main())
```

Esto es *aproximadamente* equivalente a:

```
import asyncio

async def main():
    await asyncio.sleep(0)
    return 42

loop = asyncio.new_event_loop()
asyncio.set_event_loop(loop)
try:
    loop.run_until_complete(main())
finally:
    asyncio.set_event_loop(None)
    loop.close()
```

La implementación real es significativamente más compleja. Por lo tanto, **`asyncio.run()`** debería ser la forma preferida de ejecutar programas asyncio.

(Contribución de Yury Selivanov en bpo-32314 [https://bugs.python.org/issue?@action=redirect&bpo=32314].)

La ejecución de `python -m asyncio` lanza un REPL asincrónico de forma nativa. Esto permite una rápida experimentación con código que tiene un nivel **`await`** superior. Ya no es necesario llamar directamente a `asyncio.run()`, lo que generaría un nuevo ciclo de eventos en cada invocación:

```
$ python -m asyncio
asyncio REPL 3.8.0
Use "await" directly instead of "asyncio.run()".
Type "help", "copyright", "credits" or "license" for mor
>>> import asyncio
>>> await asyncio.sleep(10, result='hello')
hello
```

(Contribución de Yury Selivanov en bpo-37028 [https://bugs.python.org/issue?@action=redirect&bpo=37028].)

La excepción `asyncio.CancelledError` ahora hereda de `BaseException` en lugar de `Exception` y ya no hereda de `concurrent.futures.CancelledError`. (Contribuido por Yury Selivanov en bpo-32528 [https://bugs.python.org/issue?@action=redirect&bpo=32528].)

En Windows, el ciclo de eventos predeterminado ahora es `ProactorEventLoop`. (Contribución de Victor Stinner en bpo-34687 [https://bugs.python.org/issue?@action=redirect&bpo=34687].)

`ProactorEventLoop` ahora también es compatible con UDP. (Contribución de Adam Meily y Andrew Svetlov en bpo-29883 [https://bugs.python.org/issue?@action=redirect&bpo=29883].)

`ProactorEventLoop` ahora puede ser interrumpido por `KeyboardInterrupt` («CTRL+C»). (Contribución de Vladimir Matveev en bpo-23057 [https://bugs.python.org/issue?@action=redirect&bpo=23057].)

Se ha agregado `asyncio.Task.get_coro()` para obtener la corrutina envuelta dentro de `asyncio.Task`. (Contribución de Alex Grönholm en bpo-36999 [https://bugs.python.org/issue?@action=redirect&bpo=36999].)

Las tareas de Asyncio ahora se pueden nombrar, ya sea pasando el argumento por palabra clave `name` a `asyncio.create_task()` o al método `create_task()` del bucle de eventos, o invocando al método `set_name()` en el objeto de tarea. El nombre de la tarea es visible en la salida de `repr()` de `asyncio.Task` y también se puede recuperar usando el método `get_name()`. (Contribución de

Alex Grönholm en bpo-34270 [https://bugs.python.org/issue?@action=redirect&bpo=34270].)

Se agregó soporte para Happy Eyeballs [https://en.wikipedia.org/wiki/Happy_Eyeballs] a **`asyncio.loop.create_connection()`**. Para especificar el comportamiento, se han agregado dos nuevos parámetros: *happy_eyeballs_delay* e *interleave*. El algoritmo Happy Eyeballs mejora la capacidad de respuesta en aplicaciones que admiten IPv4 e IPv6 al intentar conectarse simultáneamente utilizando ambos. (Contribución de twisteroid ambassador en bpo-33530 [https://bugs.python.org/issue?@action=redirect&bpo=33530].)

builtins

La función incorporada **`compile()`** se ha mejorado para que acepte el flag `ast.PyCF_ALLOW_TOP_LEVEL_AWAIT`. Si se pasa este nuevo flag, **`compile()`** permitirá construcciones de nivel superior `await`, `async for` y `async with`, que normalmente se consideran sintaxis inválida. El objeto de código asíncrono marcado con el flag `CO_COROUTINE` puede ser retornado. (Contribución de Matthias Bussonnier en bpo-34616 [https://bugs.python.org/issue?@action=redirect&bpo=34616].)

collections

El método **`_asdict()`** para **`collections.namedtuple()`** ahora retorna una instancia de **`dict`** en lugar de una de **`collections.OrderedDict`**. Esto funciona porque se garantiza que los diccionarios regulares mantienen el orden de inserción desde Python 3.7. Si se requieren las características adicionales de **`OrderedDict`**, la solución sugerida es realizar una conversión del resultado al tipo deseado: `OrderedDict(nt._asdict())`. (Contribución de Raymond Hettinger en bpo-35864 [https://bugs.python.org/issue?@action=redirect&bpo=35864].)

cProfile

La clase **`cProfile.Profile`** ahora puede usarse como gestor de contexto. Ahora se puede perfilar un bloque de código ejecutando:

```
import cProfile

with cProfile.Profile() as profiler:
      # code to be profiled
      ...
```

(Contribución de Scott Sanderson en bpo-29235 [https://bugs.python.org/issue?@action=redirect&bpo=29235].)

csv

`csv.DictReader` ahora retorna instancias de `dict` en lugar de `collections.OrderedDict`. La herramienta ahora es más rápida y usa menos memoria, mientras conserva el orden de los campos. (Contribución de Michael Selik en bpo-34003 [https://bugs.python.org/issue?@action=redirect&bpo=34003].)

curses

Se agregó una nueva variable que contiene información de versión estructurada para la biblioteca ncurses subyacente: `ncurses_version`. (Contribución de Serhiy Storchaka en bpo-31680 [https://bugs.python.org/issue?@action=redirect&bpo=31680].)

ctypes

En Windows, `CDLL` y sus subclases ahora aceptan un parámetro *winmode* para especificar flags para la invocación subyacente de `LoadLibraryEx`. Los flags predeterminados están establecidos para cargar solo las dependencias de DLL desde ubicaciones confiables, incluida la ruta donde se almacena la DLL (si se usa una ruta completa o parcial para cargar la DLL inicial) y las rutas agregadas por `add_dll_directory()` . (Contribución de Steve Dower en bpo-36085 [https://bugs.python.org/issue?@action=redirect&bpo=36085].)

datetime

Added new alternate constructors

`datetime.date.fromisocalendar()` and **`datetime.datetime.fromisocalendar()`**, which construct **`date`** and **`datetime`** objects respectively from ISO year, week number, and weekday; these are the inverse of each class's `isocalendar` method. (Contributed by Paul Ganssle in bpo-36004 [https://bugs.python.org/issue?@action=redirect&bpo=36004].)

functools

`functools.lru_cache()` ahora se puede usar como un decorador directo en lugar de como una función que retorna un decorador. De forma que ambos son compatibles ahora:

```
@lru_cache
def f(x):
    ...

@lru_cache(maxsize=256)
def f(x):
    ...
```

(Contribución de Raymond Hettinger en bpo-36772 [https://bugs.python.org/issue?@action=redirect&bpo=36772].)

Se ha agregado un nuevo decorador **`functools.cached_property()`**, para propiedades calculadas almacenadas en caché durante toda la vida útil de la instancia.

```
import functools
import statistics

class Dataset:
   def __init__(self, sequence_of_numbers):
      self.data = sequence_of_numbers

   @functools.cached_property
   def variance(self):
      return statistics.variance(self.data)
```

(Contribución de Carl Meyer en bpo-21145 [https://bugs.python.org/

issue?@action=redirect&bpo=21145].)

Se ha agregado un nuevo decorador `functools.singledispatchmethod()` que convierte métodos en funciones genéricas usando single dispatch:

```
from functools import singledispatchmethod
from contextlib import suppress

class TaskManager:

    def __init__(self, tasks):
        self.tasks = list(tasks)

    @singledispatchmethod
    def discard(self, value):
        with suppress(ValueError):
            self.tasks.remove(value)

    @discard.register(list)
    def _(self, tasks):
        targets = set(tasks)
        self.tasks = [x for x in self.tasks if x not in
```

(Contribución de Ethan Smith en bpo-32380 [https://bugs.python.org/issue?@action=redirect&bpo=32380])

gc

`get_objects()` ahora puede recibir un parámetro opcional *generation* que indica la generación de la que recolectar objetos. (Contribución de Pablo Galindo en bpo-36016 [https://bugs.python.org/issue?@action=redirect&bpo=36016].)

gettext

Agregado `pgettext()` y sus variantes. (Contribución de Franz Glasner, Éric Araujo y Cheryl Sabella en bpo-2504 [https://bugs.python.org/issue?@action=redirect&bpo=2504].)

gzip

Se ha agregó el parámetro *mtime* a `gzip.compress()` para una salida reproducible. (Contribución de Guo Ci Teo en bpo-34898 [https://bugs.python.org/issue?@action=redirect&bpo=34898].)

Una excepción `BadGzipFile` es lanzada ahora, en lugar de `OSError`, para ciertos tipos de archivos gzip no válidos o corruptos. (Contribución de Filip Gruszczyński, Michele Orrù y Zackery Spytz en bpo-6584 [https://bugs.python.org/issue?@action=redirect&bpo=6584].)

IDLE e idlelib

Las salidas superiores a N líneas (50 por defecto) se pliegan en un botón. N se puede cambiar en la sección PyShell de la página General del cuadro de diálogo Settings. Se pueden plegar menos líneas, pero posiblemente más largas, haciendo clic derecho en la salida. La salida plegada se puede expandir en su lugar haciendo doble clic en el botón o en el portapapeles o en una ventana separada haciendo clic derecho en el botón. (Contribución de Tal Einat en bpo-1529353 [https://bugs.python.org/issue?@action=redirect&bpo=1529353].)

Se ha agregado «Run Customized» al menú Run para ejecutar un módulo con configuraciones personalizadas. Cualquier argumento de la línea de comandos ingresado se agrega a sys.argv. Además, vuelven a aparecer en el cuadro para la próxima ejecución personalizada. También se puede suprimir el reinicio normal del módulo principal de la Shell. (Contribución de Cheryl Sabella, Terry Jan Reedy y otros en bpo-5680 [https://bugs.python.org/issue?@action=redirect&bpo=5680] y bpo-37627 [https://bugs.python.org/issue?@action=redirect&bpo=37627].)

Se agregaron números de línea opcionales para las ventanas del editor IDLE. Las ventanas se abren sin números de línea, a menos que se establezca lo contrario en la pestaña General del cuadro de diálogo de configuración. Los números de línea de una ventana existente se muestran y ocultan en el menú Options. (Contribución de Tal Einat y Saimadhav Heblikar en bpo-17535 [https://bugs.python.org/issue?@action=redirect&bpo=17535].)

La codificación nativa del sistema operativo ahora se usa para convertir entre cadenas de Python y objetos Tcl. Esto permite que el IDLE funcione con emoji y otros caracteres que no son BMP. Estos caracteres se pueden mostrar o copiar y pegar en ,o desde, el portapapeles. Convertir cadenas de Tcl a Python y viceversa ahora nunca falla. (Mucha gente trabajó en esto durante ocho años, pero el problema finalmente lo resolvió Serhiy Storchaka en bpo-13153 [https://bugs.python.org/issue?@action=redirect&bpo=13153].)

Nuevo en 3.8.1:

Agregue la opción para desactivar el parpadeo del cursor. (Contribuido por Zackery Spytz en bpo-4603 [https://bugs.python.org/issue?@action=redirect&bpo=4603].)

La tecla Escape ahora cierra las ventanas de finalización IDLE. (Contribuido por Johnny Najera en bpo-38944 [https://bugs.python.org/issue?@action=redirect&bpo=38944].)

Los cambios anteriores se han portado a las versiones de mantenimiento de Python 3.7.

Agrega palabras clave a la lista de finalización del nombre del módulo. (Contribución de Terry J. Reedy en bpo-37765 [https://bugs.python.org/issue?@action=redirect&bpo=37765].)

inspect

La función `inspect.getdoc()` puede ahora encontrar cadenas de documentación para `__slots__` si el este atributo es un `dict` cuyos valores son las cadenas de documentación. Esto proporciona opciones de documentación similares a las que ya tenemos para `property()`, `classmethod()` y `staticmethod()`:

```
class AudioClip:
    __slots__ = {'bit_rate': 'expressed in kilohertz to
                 'duration': 'in seconds, rounded up to
    def __init__(self, bit_rate, duration):
        self.bit_rate = round(bit_rate / 1000.0, 1)
        self.duration = ceil(duration)
```

(Contribución de Raymond Hettinger en bpo-36326 [https://bugs.python.org/issue?@action=redirect&bpo=36326].)

io

En el modo de desarrollo (`-X` `env`) y en debug build, el finalizador `io.IOBase` ahora registra la excepción si falla el método `close()`. La excepción se ignora de forma silenciosa de forma predeterminada en la versión de la versión. (Contribuido por Victor Stinner en bpo-18748 [https://bugs.python.org/issue?@action=redirect&bpo=18748].)

itertools

Se ha agregado un argumento por palabra clave opcional *initial* a la función `itertools.accumulate()` para permitir especificar un valor inicial:

```
>>> from itertools import accumulate
>>> list(accumulate([10, 5, 30, 15], initial=1000))
[1000, 1010, 1015, 1045, 1060]
```

(Contribución de Lisa Roach en bpo-34659 [https://bugs.python.org/issue?@action=redirect&bpo=34659].)

json.tool

Agregadas las opciones `--json-lines` para analizar sintácticamente cada línea de entrada como un objeto JSON separado. (Contribución de Weipeng Hong en bpo-31553 [https://bugs.python.org/issue?@action=redirect&bpo=31553].)

logging

Added a *force* keyword argument to `logging.basicConfig()`. When set to true, any existing handlers attached to the root logger are removed and closed before carrying out the configuration specified by the other arguments.

Esto resuelve un problema de larga data. Una vez que se había invocado a un registrador o a *basicConfig()*, las invocaciones posteriores a *basicConfig()* se ignoraban en silencio. Esto dificultaba la actualización, la experimentación o la instrucción de las diversas opciones de configuración de registro mediante el interprete interactivo o el bloc de notas de Jupyter.

(Sugerencia de Raymond Hettinger, implementación de Dong-hee Na y revisión de Vinay Sajip en bpo-33897 [https://bugs.python.org/issue?@action=redirect&bpo=33897].)

math

Se ha agregado la nueva función `math.dist()` para calcular la distancia euclidiana entre dos puntos. (Contribución de Raymond Hettinger en bpo-33089 [https://bugs.python.org/issue?@action=redirect&bpo=33089].)

Se ha expandido la función `math.hypot()` para manejar múltiples dimensiones. Anteriormente, solo admitía dos dimensiones. (Contribución de Raymond Hettinger en bpo-33089 [https://bugs.python.org/issue?@action=redirect&bpo=33089].)

Agregada una nueva función, `math.prod()`, como función análoga a `sum()`, que retorna el producto de todos los elementos de un iterable de números partiendo de un valor de inicio (*start*) (por defecto: 1):

```
>>> prior = 0.8
>>> likelihoods = [0.625, 0.84, 0.30]
>>> math.prod(likelihoods, start=prior)
0.126
```

(Contribución de Pablo Galindo en bpo-35606 [https://bugs.python.org/issue?@action=redirect&bpo=35606].)

Agregadas dos nuevas funciones combinatorias, `math.perm()` y `math.comb()`:

```
>>> math.perm(10, 3)    # Permutations of 10 things take
720
>>> math.comb(10, 3)    # Combinations of 10 things take
```

(Contribución de Yash Aggarwal, Keller Fuchs, Serhiy Storchaka y Raymond Hettinger en bpo-37128 [https://bugs.python.org/issue?@action=redirect&bpo=37128], bpo-37178 [https://bugs.python.org/issue?@action=redirect&bpo=37178] y bpo-35431 [https://bugs.python.org/issue?@action=redirect&bpo=35431].)

Se ha agregada una nueva función **`math.isqrt()`** para calcular raíces cuadradas enteras precisas sin conversión a coma flotante. La nueva función admite números enteros arbitrariamente grandes. Es más rápida que `floor(sqrt(n))` pero más lenta que **`math.sqrt()`**:

```
>>> r = 650320427
>>> s = r ** 2
>>> isqrt(s - 1)         # correct
650320426
>>> floor(sqrt(s - 1))   # incorrect
650320427
```

(Contribución de Mark Dickinson en bpo-36887 [https://bugs.python.org/issue?@action=redirect&bpo=36887].)

La función **`math.factorial()`** ya no acepta argumentos que no sean similares a enteros. (Contribución de Pablo Galindo en bpo-33083 [https://bugs.python.org/issue?@action=redirect&bpo=33083].)

mmap

La clase **`mmap.mmap`** ahora tiene un método **`madvise()`** para acceder a la llamada al sistema `madvise()`. (Contribución de Zackery Spytz en bpo-32941 [https://bugs.python.org/issue?@action=redirect&bpo=32941].)

multiprocessing

Agregado el nuevo módulo **`multiprocessing.shared_memory`**. (Contribución de Davin Potts en bpo-35813 [https://bugs.python.org/issue?@action=redirect&bpo=35813].)

En macOS, el método de inicio *spawn* se usa ahora por defecto. (Contribución de Victor Stinner en bpo-33725 [https://bugs.python.org/issue?@action=redirect&bpo=33725].)

os

Se agregó una nueva función `add_dll_directory()` en Windows para proporcionar rutas de búsqueda adicionales para las dependencias nativas al importar módulos de extensión o al cargar archivos DLL utilizando `ctypes`. (Contribución de Steve Dower en bpo-36085 [https://bugs.python.org/issue?@action=redirect&bpo=36085].)

Se agregó una nueva función `os.memfd_create()` para envolver la llamada al sistema `memfd_create()`. (Contribución de Zackery Spytz y Christian Heimes en bpo-26836 [https://bugs.python.org/issue?@action=redirect&bpo=26836].)

On Windows, much of the manual logic for handling reparse points (including symlinks and directory junctions) has been delegated to the operating system. Specifically, `os.stat()` will now traverse anything supported by the operating system, while `os.lstat()` will only open reparse points that identify as «name surrogates» while others are opened as for `os.stat()`. In all cases, `os.stat_result.st_mode` will only have `S_IFLNK` set for symbolic links and not other kinds of reparse points. To identify other kinds of reparse point, check the new `os.stat_result.st_reparse_tag` attribute.

En Windows, `os.readlink()` ahora puede leer uniones de directorio. Ten en cuenta que `islink()` retornará `False` para las uniones de directorios, por lo que el código que comprueba en primer lugar `islink` continuará tratando las uniones como directorios, mientras que el código que maneja los errores de `os.readlink()` ahora puede tratar las uniones como enlaces.

(Contribución de Steve Dower en bpo-37834 [https://bugs.python.org/issue?@action=redirect&bpo=37834].)

os.path

Las funciones de `os.path` que retornan un resultado booleano como `exists()`, `lexists()`, `isdir()`, `isfile()`, `islink()` e `ismount()` ahora retornan `False` en lugar de lanzar una excepción `ValueError`, o sus subclases `UnicodeEncodeError` y `UnicodeDecodeError`, para rutas que contienen caracteres o bytes irrepresentables a nivel del sistema operativo. (Contribución de Serhiy Storchaka en bpo-33721 [https://bugs.python.org/issue?@action=redirect&bpo=33721].)

`expanduser()` en Windows ahora usa preferentemente la variable de entorno **`USERPROFILE`** y no usa **`HOME`**, que normalmente no está establecido para cuentas de usuario normales. (Contribución de Anthony Sottile en bpo-36264 [https://bugs.python.org/issue?@action=redirect&bpo=36264].)

`isdir()` en Windows ya no retorna `True` para un enlace a un directorio no existente.

`realpath()` en Windows ahora resuelve puntos de reinterpretación (reparse points), incluidos enlaces simbólicos y uniones de directorio.

(Contribución de Steve Dower en bpo-37834 [https://bugs.python.org/issue?@action=redirect&bpo=37834].)

pathlib

`pathlib.Path` methods that return a boolean result like `exists()`, `is_dir()`, `is_file()`, `is_mount()`, `is_symlink()`, `is_block_device()`, `is_char_device()`, `is_fifo()`, `is_socket()` now return `False` instead of raising `ValueError` or its subclass `UnicodeEncodeError` for paths that contain characters unrepresentable at the OS level. (Contributed by Serhiy Storchaka in bpo-33721 [https://bugs.python.org/issue?@action=redirect&bpo=33721].)

Added **`pathlib.Path.link_to()`** which creates a hard link pointing to a path. (Contributed by Joannah Nanjekye in bpo-26978 [https://bugs.python.org/issue?@action=redirect&bpo=26978]) Note that `link_to` was deprecated in 3.10 and removed in 3.12 in favor of a `hardlink_to` method added in 3.10 which matches the semantics of the existing `symlink_to` method.

pickle

Las extensiones de **`pickle`** que subclasifican la clase optimizada de C **`Pickler`** ahora pueden anular la lógica de pickling de funciones y clases definiendo el método especial **`reducer_override()`**. (Contribución de Pierre Glaser y Olivier Grisel en bpo-35900 [https://bugs.python.org/issue?@action=redirect&bpo=35900].)

plistlib

Se ha agregado la nueva clase **`plistlib.UID`** y se ha habilitado el soporte para leer y escribir plists binarios codificados por NSKeyedArchiver. (Contribución de Jon Janzen en bpo-26707 [https://bugs.python.org/issue?@action=redirect&bpo=26707].)

pprint

Se ha agregado el parámetro *sort_dicts* a varias funciones del módulo **`pprint`**. De forma predeterminada, esas funciones continúan ordenando los diccionarios antes de procesarlos o imprimirlos. Sin embargo, si *sort_dicts* se establece en falso, los diccionarios conservan el orden en que se insertaron las claves. Esto puede resultar útil para la comparación con entradas JSON durante la depuración.

Además, hay una nueva función de conveniencia, **`pprint.pp()`**, que es igual que **`pprint.pprint()`** pero con *sort_dicts* establecido en `False` por defecto:

```
>>> from pprint import pprint, pp
>>> d = dict(source='input.txt', operation='filter', des
>>> pp(d, width=40)                    # Original order
{'source': 'input.txt',
 'operation': 'filter',
 'destination': 'output.txt'}
>>> pprint(d, width=40)                # Keys sorted alpha
{'destination': 'output.txt',
 'operation': 'filter',
 'source': 'input.txt'}
```

(Contribución de Rémi Lapeyre en bpo-30670 [https://bugs.python.org/issue?@action=redirect&bpo=30670].)

py_compile

`py_compile.compile()` ahora admite el modo silencioso. (Contribución de Joannah Nanjekye en bpo-22640 [https://bugs.python.org/issue?@action=redirect&bpo=22640].)

shlex

La nueva función `shlex.join()` actúa a la inversa de `shlex.split()`. (Contribución de Bo Bayles en bpo-32102 [https://bugs.python.org/issue?@action=redirect&bpo=32102].)

shutil

`shutil.copytree()` ahora acepta el nuevo argumento por palabra clave `dirs_exist_ok`. (Contribución de Josh Bronson en bpo-20849 [https://bugs.python.org/issue?@action=redirect&bpo=20849].)

`shutil.make_archive()` ahora usa por defecto el formato pax moderno (POSIX.1-2001) para nuevos archivos para mejorar la portabilidad y la conformidad con los estándares, heredado el cambio correspondiente del módulo `tarfile`. (Contribución de C.A.M. Gerlach en bpo-30661 [https://bugs.python.org/issue?@action=redirect&bpo=30661].)

`shutil.rmtree()` en Windows ahora elimina las uniones de directorio sin eliminar recursivamente su contenido primero. (Contribución de Steve Dower en bpo-37834 [https://bugs.python.org/issue?@action=redirect&bpo=37834].)

socket

Added `create_server()` and `has_dualstack_ipv6()` convenience functions to automate the necessary tasks usually involved when creating a server socket, including accepting both IPv4 and IPv6 connections on the same socket. (Contributed by Giampaolo Rodolà in bpo-17561 [https://bugs.python.org/issue?@action=redirect&bpo=17561].)

The `socket.if_nameindex()`, `socket.if_nametoindex()`, and `socket.if_indextoname()` functions have been implemented on Windows. (Contributed by Zackery Spytz in bpo-37007 [https://bugs.python.org/issue?@action=redirect&bpo=37007].)

ssl

Se ha agregado `post_handshake_auth` para habilitar y `verify_client_post_handshake()` para iniciar la autenticación tras el establecimiento de la comunicación en TLS 1.3. (Contribución de Christian Heimes en bpo-34670 [https://bugs.python.org/issue?@action=redirect&bpo=34670].)

statistics

Added `statistics.fmean()` as a faster, floating-point variant of `statistics.mean()`. (Contributed by Raymond Hettinger and Steven D'Aprano in bpo-35904 [https://bugs.python.org/issue?@action=redirect&bpo=35904].)

Added `statistics.geometric_mean()` (Contributed by Raymond Hettinger in bpo-27181 [https://bugs.python.org/issue?@action=redirect&bpo=27181].)

Se ha agregado `statistics.multimode()` que retorna una lista con los valores más comunes. (Contribución de Raymond Hettinger en bpo-35892 [https://bugs.python.org/issue?@action=redirect&bpo=35892].)

Se ha agregado `statistics.quantiles()` que divide datos o una distribución en intervalos equiprobables (por ejemplo, cuartiles, deciles o percentiles). (Contribución de Raymond Hettinger en bpo-36546 [https://bugs.python.org/issue?@action=redirect&bpo=36546].)

Se ha agregado `statistics.NormalDist`, una herramienta para crear y manipular distribuciones normales de una variable aleatoria. (Contribución de Raymond Hettinger en bpo-36018

[https://bugs.python.org/issue?@action=redirect&bpo=36018].)

```
>>> temperature_feb = NormalDist.from_samples([4, 12, -3
>>> temperature_feb.mean
6.0
>>> temperature_feb.stdev
6.356099432828281

>>> temperature_feb.cdf(3)            # Chance of being
0.3184678262814532
>>> # Relative chance of being 7 degrees versus 10 degre
>>> temperature_feb.pdf(7) / temperature_feb.pdf(10)
1.2039930378537762

>>> el_niño = NormalDist(4, 2.5)
>>> temperature_feb += el_niño        # Add in a climate
>>> temperature_feb
NormalDist(mu=10.0, sigma=6.830080526611674)

>>> temperature_feb * (9/5) + 32      # Convert to Fahre
NormalDist(mu=50.0, sigma=12.294144947901014)
>>> temperature_feb.samples(3)        # Generate random
[7.672102882379219, 12.000027119750287, 4.647488369766
```

sys

Se ha agregado la nueva función `sys.unraisablehook()` que se puede anular para controlar cómo se manejan las «excepciones no lanzables». Se llama cuando se ha producido una excepción, pero Python no tiene forma de manejarla. Por ejemplo, cuando un destructor lanza una excepción o durante la recolección de basura (`gc.collect()`). (Contribución de Victor Stinner en bpo-36829 [https://bugs.python.org/issue?@action=redirect&bpo=36829].)

tarfile

El módulo `tarfile` ahora tiene por defecto el formato pax moderno (POSIX.1-2001) para nuevos archivos, en lugar del anterior específico de GNU. Esto mejora la portabilidad multiplataforma con una codificación consistente (UTF-8) en un formato estandarizado y extensible, y ofrece otros varios beneficios. (Contribución de C.A.M. Gerlach en bpo-36268 [https://bugs.python.org/issue?@action=redirect&bpo=36268].)

threading

Se ha agregado una nueva función `threading.excepthook()` que maneja las excepciones `threading.Thread.run()` no capturadas. Se puede anular para controlar cómo se manejan las excepciones `threading.Thread.run()` no capturadas. (Contribución de Victor Stinner en bpo-1230540 [https://bugs.python.org/issue?@action=redirect&bpo=1230540].)

Se han agregado una nueva función `threading.get_native_id()` y un atributo `native_id` a la clase `threading.Thread`. Estos retornan el Thread ID nativo integral del hilo actual asignado por el kernel. Esta función solo está disponible en determinadas plataformas, consulta `get_native_id` para obtener más información. (Contribución de Jake Tesler en bpo-36084 [https://bugs.python.org/issue?@action=redirect&bpo=36084].)

tokenize

El módulo `tokenize` ahora emite implícitamente un token `NEWLINE` cuando se le proporciona una entrada sin una nueva línea al final. Este comportamiento ahora coincide con lo que hace internamente el tokenizador de C. (Contribución de Ammar Askar en bpo-33899 [https://bugs.python.org/issue?@action=redirect&bpo=33899].)

tkinter

Added methods `selection_from()`, `selection_present()`, `selection_range()` and `selection_to()` in the `tkinter.Spinbox` class. (Contributed by Juliette Monsel in bpo-34829 [https://bugs.python.org/issue?@action=redirect&bpo=34829].)

Added method `moveto()` in the `tkinter.Canvas` class. (Contributed by Juliette Monsel in bpo-23831 [https://bugs.python.org/

issue?@action=redirect&bpo=23831].)

The **`tkinter.PhotoImage`** class now has **`transparency_get()`** and **`transparency_set()`** methods. (Contributed by Zackery Spytz in bpo-25451 [https://bugs.python.org/issue?@action=redirect&bpo=25451].)

time

Se ha agregado el nuevo reloj **`CLOCK_UPTIME_RAW`** para macOS 10.12. (Contribución de Joannah Nanjekye en bpo-35702 [https://bugs.python.org/issue?@action=redirect&bpo=35702].)

typing

Se han incorporado varias características al módulo **`typing`**:

- Un tipo de diccionario con tipos para cada clave. Consultar **PEP 589** [https://peps.python.org/pep-0589/] y **`typing.TypedDict`**. TypedDict usa solo claves de cadenas de caracteres. De forma predeterminada, se requiere que todas las claves estén presentes. Especifica «total=False» para permitir que las claves sean opcionales:

  ```
  class Location(TypedDict, total=False):
      lat_long: tuple
      grid_square: str
      xy_coordinate: tuple
  ```

- Tipos literales. Consultar **PEP 586** [https://peps.python.org/pep-0586/] y **`typing.Literal`**. Los tipos literales indican que un parámetro o valor de retorno está restringido a uno o más valores literales específicos:

  ```
  def get_status(port: int) -> Literal['connected', '
      ...
  ```

- Variables, funciones, métodos y clases «finales». Consultar **PEP 591** [https://peps.python.org/pep-0591/], **`typing.Final`** y **`typing.final()`**. El clasificador final instruye a un validador estático de tipos para restringir la subclasificación, anulación o reasignación:

```
pi: Final[float] = 3.1415926536
```

- Definiciones de protocolo. Consultar **PEP 544** [https://peps.python.org/pep-0544/], `typing.Protocol` y `typing.runtime_checkable()`. ABCs simples como `typing.SupportsInt` ahora son subclases de `Protocol`.
- Nueva clase protocolo `typing.SupportsIndex`.
- Nuevas funciones `typing.get_origin()` y `typing.get_args()`.

unicodedata

El módulo `unicodedata` se ha actualizado para usar la versión Unicode 12.1.0 [https://blog.unicode.org/2019/05/unicode-12-1-en.html].

La nueva función `is_normalized()` puede usarse para verificar que una cadena está en una forma normal específica, lo que es a menudo mucho más rápido que normalizar la cadena. (Contribución de Max Belanger, David Euresti y Greg Price en bpo-32285 [https://bugs.python.org/issue?@action=redirect&bpo=32285] y bpo-37966 [https://bugs.python.org/issue?@action=redirect&bpo=37966]).

unittest

Se ha agregado `AsyncMock` para admitir una versión asincrónica de `Mock`. También se han agregado nuevas funciones de aserción apropiadas para las pruebas. (Contribución de Lisa Roach en bpo-26467 [https://bugs.python.org/issue?@action=redirect&bpo=26467]).

Added `addModuleCleanup()` and `addClassCleanup()` to unittest to support cleanups for `setUpModule()` and `setUpClass()`. (Contributed by Lisa Roach in bpo-24412 [https://bugs.python.org/issue?@action=redirect&bpo=24412].)

Varias funciones de aserción simulada ahora también imprimen una lista de llamadas reales en caso de fallo. (Contribución de Petter

Strandmark en bpo-35047 [https://bugs.python.org/issue?@action=redirect&bpo=35047].)

El módulo `unittest` ha obtenido soporte para corrutinas que se utilizarán como casos de prueba con `unittest.IsolatedAsyncioTestCase`. (Contribución de Andrew Svetlov en bpo-32972 [https://bugs.python.org/issue?@action=redirect&bpo=32972].)

Ejemplo:

```
import unittest


class TestRequest(unittest.IsolatedAsyncioTestCase):

    async def asyncSetUp(self):
        self.connection = await AsyncConnection()

    async def test_get(self):
        response = await self.connection.get("https://ex
        self.assertEqual(response.status_code, 200)

    async def asyncTearDown(self):
        await self.connection.close()


if __name__ == "__main__":
    unittest.main()
```

venv

`venv` ahora incluye un script `Activate.ps1` en todas las plataformas para activar entornos virtuales en PowerShell Core 6.1. (Contribución de Brett Cannon en bpo-32718 [https://bugs.python.org/issue?@action=redirect&bpo=32718].)

weakref

Los objetos proxy retornados por **`debilref.proxy()`** ahora admiten los operadores de multiplicación de matrices `@` y `@=`, además de los otros operadores numéricos. (Contribución de Mark Dickinson en bpo-36669 [https://bugs.python.org/issue?@action=redirect&bpo=36669].)

xml

Como mitigación contra DTD y recuperación de entidades externas, los módulos **`xml.dom.minidom`** y **`xml.sax`** ya no procesan entidades externas de forma predeterminada. (Contribución de Christian Heimes en bpo-17239 [https://bugs.python.org/issue?@action=redirect&bpo=17239].)

Los métodos `.find*()` del módulo **`xml.etree.ElementTree`** admiten búsquedas con comodines, como `{*}tag`, que ignora el espacio de nombres, y `{namespace}*`, que retorna todas las etiquetas en el espacio de nombres dado. (Contribución de Stefan Behnel en bpo-28238 [https://bugs.python.org/issue?@action=redirect&bpo=28238].)

The **`xml.etree.ElementTree`** module provides a new function **`canonicalize()`** that implements C14N 2.0. (Contributed by Stefan Behnel in bpo-13611 [https://bugs.python.org/issue?@action=redirect&bpo=13611].)

El objeto de destino de **`xml.etree.ElementTree.XMLParser`** puede recibir eventos de declaración de espacio de nombres a través de los nuevos métodos de retrollamada `start_ns()` y `end_ns()`. Además, el destino **`xml.etree.ElementTree.TreeBuilder`** se puede configurar para procesar eventos sobre comentarios e instrucciones de procesamiento para incluirlos en el árbol generado. (Contribución de Stefan Behnel en bpo-36676 [https://bugs.python.org/issue?@action=redirect&bpo=36676] y bpo-36673 [https://bugs.python.org/issue?@action=redirect&bpo=36673].)

xmlrpc

`xmlrpc.client.ServerProxy` ahora admite un argumento por palabra clave *headers* opcional para una secuencia de encabezados HTTP que se enviarán con cada solicitud. Entre otras cosas, esto permite actualizar desde la autenticación básica predeterminada a una autenticación de sesión más rápida. (Contribución de Cédric Krier en bpo-35153 [https://bugs.python.org/issue?@action=redirect&bpo=35153].)

Optimizaciones

- El módulo `subprocess` ahora puede usar en ciertos casos la función `os.posix_spawn()` para mejorar el rendimiento. Actualmente, solo se usa en macOS y Linux (usando glibc 2.24 o una versión más reciente) y siempre que se cumplan todas estas condiciones:
  - ○ *close_fds* es falso;
  - ○ los parámetros *preexec_fn*, *pass_fds*, *cwd* y *start_new_session* no están establecidos;
  - ○ la ruta *executable* contiene un directorio.

  (Contribución de Joannah Nanjekye y Victor Stinner en bpo-35537 [https://bugs.python.org/issue?@action=redirect&bpo=35537].)

- `shutil.copyfile()`, `shutil.copy()`, `shutil.copy2()`, `shutil.copytree()` y `shutil.move()` usan llamadas al sistema de «copia-rápida» específicas de la plataforma en Linux y macOS, con la finalidad de copiar ficheros más eficientemente. «copia-rápida» significa que la operación de copiado tiene lugar en el propio kernel, evitando que Python haga uso de búferes en el espacio de usuario como «`outfd.write(infd.read())`». En Windows, **`shutil.copyfile()`** usa un tamaño de búfer predeterminado más grande (1 MiB en lugar de 16 KiB) y se usa una variante de `shutil.copyfileobj()` basada en `memoryview()`. La aceleración al copiar un archivo de 512 MiB dentro de la misma partición es de aproximadamente +26% en Linux, +50% en macOS y +40% en Windows. Además, se consumen muchos menos ciclos de CPU. Consultar la sección Operaciones de copia eficientes dependientes de la plataforma. (Contribución de Giampaolo Rodolà en bpo-33671 [https://bugs.python.org/issue?@action=redirect&bpo=33671].)

- Ahora `shutil.copytree()` usa la función `os.scandir()` y todas las funciones de copia que dependen de ella usan el valor en caché de `os.stat()`. Al copiar un directorio con 8000 archivos, la mejora de velocidad es de +9% en Linux, +20% en Windows y +30% en recursos compartidos de Windows SMB. Además, el número de llamadas al sistema de `os.stat()` se reduce en un 38%, lo que hace que `shutil.copytree()` sea más rápida especialmente en sistemas de archivos de red. (Contribución de Giampaolo Rodolà en bpo-33695 [https://bugs.python.org/issue?@action=redirect&bpo=33695].)

- El protocolo por defecto del módulo `pickle` es ahora el Protocolo 4, introducido por primera vez en Python 3.4. Ofrece un mejor desempeño y un menor tamaño, en comparación con el Protocolo 3 disponible desde Python 3.0.

- Se eliminó un miembro `Py_ssize_t` de `PyGC_Head`. El tamaño de todos los objetos rastreados de GC (por ejemplo, tupla, lista, dict) se reduce en 4 u 8 bytes. (Aportado por Inada Naoki en bpo-33597 [https://bugs.python.org/issue?@action=redirect&bpo=33597].)

- La clase `uuid.UUID` ahora usa `__slots__` para reducir su impacto en la memoria. (Contribución de Wouter Bolsterlee y Tal Einat en bpo-30977 [https://bugs.python.org/issue?@action=redirect&bpo=30977].)

- Mejorado el rendimiento de `operator.itemgetter()` en un 33%. Se ha optimizado el manejo de argumentos y se ha agregado una ruta rápida al caso común de índices enteros no negativos únicos en tuplas (que constituye un caso de uso común en la biblioteca estándar). (Contribución de Raymond Hettinger en bpo-35664 [https://bugs.python.org/issue?@action=redirect&bpo=35664].)

- Se han acelerado las búsquedas de campos en `collections.namedtuple()`. Ahora son más del doble de rápidas, lo que las convierte en la forma más rápida de búsqueda de variables de instancia en Python. (Contribución de Raymond Hettinger, Pablo Galindo, Joe Jevnik y Serhiy Storchaka en bpo-32492 [https://bugs.python.org/issue?@action=redirect&bpo=32492].)

- El constructor de `list` no sobre-asignará el búfer del elemento interno si se conoce la longitud de la entrada iterable (es decir, si la entrada implementa `__len__`). Esto hace que la lista generada sea, en promedio, un 12% más pequeña. (Contribución de Raymond Hettinger y Pablo Galindo en bpo-33234 [https://bugs.python.org/issue?@action=redirect&bpo=33234].)
- De ha duplicado la velocidad de escritura de variables de clase. Antes, cuando se actualizaba un atributo non-dunder, había una llamada innecesaria para actualizar slots. (Contribución de Stefan Behnel, Pablo Galindo Salgado, Raymond Hettinger, Neil Schemenauer y Serhiy Storchaka en bpo-36012 [https://bugs.python.org/issue?@action=redirect&bpo=36012].)
- Se ha reducido la sobrecarga de conversión de argumentos pasados a muchas funciones y métodos integrados. Esto acelera la llamada a algunas funciones y métodos incorporados simples hasta un 20-50%. (Contribución de Serhiy Storchaka en bpo-23867 [https://bugs.python.org/issue?@action=redirect&bpo=23867], bpo-35582 [https://bugs.python.org/issue?@action=redirect&bpo=35582] y bpo-36127 [https://bugs.python.org/issue?@action=redirect&bpo=36127].)
- La instrucción `LOAD_GLOBAL` ahora utiliza el nuevo mecanismo «per opcode cache». Ahora es aproximadamente un 40% más rápida. (Contribución de Yury Selivanov e Inada Naoki en bpo-26219 [https://bugs.python.org/issue?@action=redirect&bpo=26219].)

Cambios en la compilación y la API de C

- El valor predeterminado para `sys.abiflags` es ahora una cadena vacía: el flag `m` para pymalloc se ha vuelto innecesario (las compilaciones con y sin pymalloc son ABI compatibles) y por lo tanto se ha eliminado. (Contribución de Victor Stinner en bpo-36707 [https://bugs.python.org/issue?@action=redirect&bpo=36707].)

  Ejemplos del cambio:

  - Solo el programa `python3.8` es instalado, el programa `python3.8m` se ha eliminado.
  - Solo el script `python3.8-config` es instalado, el script `python3.8m-config` se ha eliminado.
  - El flag `m` se ha eliminado del sufijo de los nombres de archivo de las bibliotecas dinámicas: los módulos de extensión de la biblioteca estándar, así como los producidos e instalados por paquetes de terceros, como los descargados desde PyPI. En Linux, por ejemplo, el sufijo `.cpython-37m-x86_64-linux-gnu.so` en Python 3.7 se ha convertido en `.cpython-38-x86_64-linux-gnu.so` en Python 3.8.

- Los archivos de cabeceras se han reorganizado para separar mejor los diferentes tipos de APIs:

  - `Include/*.h` debe ser la API de C portable, pública y estable.
  - `Include/cpython/*.h` debe ser la API de C inestable específica de CPython. Una API pública, con alguna API privada marcada con los prefijos `_Py` o `_PY`.
  - `Include/internal/*.h` es la API de C interna privada, muy específica de CPython. Esta API no tiene retro-compatibilidad garantizada con versiones anteriores y no debe ser usada fuera de CPython. Solo está expuesta para cubrir necesidades muy específicas, como es el caso de depuradores y perfiladores, que necesitan acceder a los componentes internos de CPython sin llamar directamente a las funciones. Esta API es ahora instalada por `make install`.

  (Contribución de Victor Stinner en bpo-35134 [https://

bugs.python.org/issue?@action=redirect&bpo=35134] y bpo-35081 [https://bugs.python.org/issue?@action=redirect&bpo=35081], trabajo iniciado por Eric Snow en Python 3.7.)

- Algunas macros se han convertido a funciones inline estáticas: los tipos de los parámetros y el tipo de retorno están bien definidos, no entrañan cuestiones que precisen el uso específico de macros y las variables tienen ámbito local. Algunos ejemplos:
  - **`Py_INCREF()`**, **`Py_DECREF()`**
  - **`Py_XINCREF()`**, **`Py_XDECREF()`**
  - **`PyObject_INIT`**, **`PyObject_INIT_VAR`**
  - Funciones privadas: **`_PyObject_GC_TRACK()`**, **`_PyObject_GC_UNTRACK()`**, **`_Py_Dealloc()`**

  (Contribución de Victor Stinner en bpo-35059 [https://bugs.python.org/issue?@action=redirect&bpo=35059].)

- Las funciones **`PyByteArray_Init()`** y **`PyByteArray_Fini()`** se han eliminado. No eran de utilidad desde Python 2.7.4 y Python 3.2.0, cuando fueron excluidas de la API limitada (ABI estable) y dejaron de estar documentadas. (Contribución de Victor Stinner en bpo-35713 [https://bugs.python.org/issue?@action=redirect&bpo=35713].)

- El resultado de **`PyExceptionClass_Name()`** es ahora de tipo `const char *` en vez de `char *`. (Contribución de Serhiy Storchaka en bpo-33818 [https://bugs.python.org/issue?@action=redirect&bpo=33818].)

- La dualidad conformada por `Modules/Setup.dist` y `Modules/Setup` ha sido eliminada. Anteriormente, al actualizar el árbol de fuentes de CPython, se tenía que copiar manualmente `Modules/Setup.dist` (dentro del árbol de fuentes) a `Modules/Setup` (dentro del árbol de compilación) para reflejar cualquier cambio en sentido ascendente. Esto suponía un pequeño beneficio para los empaquetadores, a expensas de una frecuente molestia para los desarrolladores de CPython, ya que olvidarse de copiar el archivo podía ocasionar fallos de compilación.

Ahora el sistema de compilación siempre lee desde `Modules/Setup` dentro del árbol de fuentes. Se recomienda a las personas que deseen personalizar este archivo que mantengan sus cambios en un fork de git de CPython o como archivos de parche, como harían con cualquier otro cambio en el árbol de fuentes.

(Contribución de Antoine Pitrou en bpo-32430 [https://bugs.python.org/issue?@action=redirect&bpo=32430].)

- Las funciones que convierten un número de Python a un entero de C, como **`PyLong_AsLong()`**, y las funciones de análisis de argumentos como **`PyArg_ParseTuple()`** con unidades de formato de conversión de enteros como `'i'`, ahora usarán el método especial **`__index__()`** si está disponible, en lugar de **`__int__()`**. Una advertencia de deprecación se emitirá para aquellos objetos que tengan el método `__int__()` pero no el método `__index__()` (como **`Decimal`** y **`Fraction`**). **`PyNumber_Check()`** ahora retornará `1` para los objetos que implementen `__index__()`. **`PyNumber_Long()`**, **`PyNumber_Float()`** y **`PyFloat_AsDouble()`** ahora también usan el método `__index__()` si está disponible. (Contribución de Serhiy Storchaka en bpo-36048 [https://bugs.python.org/issue?@action=redirect&bpo=36048] y bpo-20092 [https://bugs.python.org/issue?@action=redirect&bpo=20092].)

- Los objetos de tipo asignados al montículo ahora aumentarán su recuento de referencias en **`PyObject_Init()`** (y en su macro paralela `PyObject_INIT`) en lugar de en **`PyType_GenericAlloc()`**. Es posible que deban ajustarse los tipos que modifican la asignación o desasignación de instancias. (Contribución de Eddie Elizondo en bpo-35810 [https://bugs.python.org/issue?@action=redirect&bpo=35810].)

- The new function **`PyCode_NewWithPosOnlyArgs()`** allows to create code objects like **`PyCode_New()`**, but with an extra *posonlyargcount* parameter for indicating the number of positional-only arguments. (Contributed by Pablo Galindo in bpo-37221 [https://bugs.python.org/issue?@action=redirect&bpo=37221].)

- **`Py_SetPath()`** ahora establece **`sys.executable`** en la ruta completa del programa (**`Py_GetProgramFullPath()`**), en vez de en el nombre del programa (**`Py_GetProgramName()`**). (Contribución de Victor Stinner en bpo-38234 [https://bugs.python.org/issue?@action=redirect&bpo=38234].)

Obsoleto

- El comando `bdist_wininst` de distutils está ahora obsoleto, usar `bdist_wheel` (paquetes wheel) en su lugar. (Contribución de Victor Stinner en bpo-37481 [https://bugs.python.org/issue?@action=redirect&bpo=37481].)
- Los métodos `getchildren()` y `getiterator()` del módulo **`ElementTree`** ahora emiten una advertencia **`DeprecationWarning`**, en lugar de **`PendingDeprecationWarning`**. Serán eliminados en Python 3.9. (Contribución de Serhiy Storchaka en bpo-29209 [https://bugs.python.org/issue?@action=redirect&bpo=29209].)
- Pasar un objeto a **`loop.set_default_executor()`** que no sea una instancia de **`concurrent.futures.ThreadPoolExecutor`** está obsoleto y será prohibido en Python 3.9. (Contribución de Elvis Pranskevichus en bpo-34075 [https://bugs.python.org/issue?@action=redirect&bpo=34075].)
- The **`__getitem__()`** methods of **`xml.dom.pulldom.DOMEventStream`**, **`wsgiref.util.FileWrapper`** and **`fileinput.FileInput`** have been deprecated.

  Las implementaciones de estos métodos han estado ignorando su parámetro *index* y retornando el siguiente item en su lugar. (Contribución de Berker Peksag en bpo-9372 [https://bugs.python.org/issue?@action=redirect&bpo=9372].)
- El atributo `_field_types` de la clase **`typing.NamedTuple`** está ahora obsoleto en favor del

atributo `__annotations__`, que contiene la misma información. (Contribución de Raymond Hettinger en bpo-36320 [https://bugs.python.org/issue?@action=redirect&bpo=36320].)

- Las clases `Num`, `Str`, `Bytes`, `NameConstant` y `Ellipsis` del módulo **`ast`** están consideradas obsoletas y serán eliminadas en versiones futuras de Python. La clase **`Constant`** debe ser usada en su lugar. (Contribución de Serhiy Storchaka en bpo-32892 [https://bugs.python.org/issue?@action=redirect&bpo=32892].)

- Los métodos `visit_Num()`, `visit_Str()`, `visit_Bytes()`, `visit_NameConstant()` y `visit_Ellipsis()` de la clase **`ast.NodeVisitor`** están obsoletos ahora y serán invocados en versiones futuras de Python. Agregar el método **`visit_Constant()`** para manejar todos los nodos constantes. (Contribución de Serhiy Storchaka en bpo-36917 [https://bugs.python.org/issue?@action=redirect&bpo=36917].)

- The **`@asyncio.coroutine`** decorator is deprecated and will be removed in version 3.10. Instead of `@asyncio.coroutine`, use **`async def`** instead. (Contributed by Andrew Svetlov in bpo-36921 [https://bugs.python.org/issue?@action=redirect&bpo=36921].)

- En **`asyncio`**, pasar un argumento *loop* de forma explícita está ahora obsoleto y será eliminado en Python 3.10 para las siguientes funciones y constructores: **`asyncio.sleep()`**, **`asyncio.gather()`**, **`asyncio.shield()`**, **`asyncio.wait_for()`**, **`asyncio.wait()`**, **`asyncio.as_completed()`**, **`asyncio.Task`**, **`asyncio.Lock`**, **`asyncio.Event`**, **`asyncio.Condition`**, **`asyncio.Semaphore`**, **`asyncio.BoundedSemaphore`**, **`asyncio.Queue`**, **`asyncio.create_subprocess_exec()`** y **`asyncio.create_subprocess_shell()`**.

- El paso explícito de objetos corrutina a **`asyncio.wait()`** está ahora obsoleto y será eliminado en Python 3.11.

(Contribución de Yury Selivanov en bpo-34790 [https://bugs.python.org/issue?@action=redirect&bpo=34790].)

- The following functions and methods are deprecated in the `gettext` module: **`lgettext()`**, **`ldgettext()`**, **`lngettext()`** and **`ldngettext()`**. They return encoded bytes, and it's possible that you will get unexpected Unicode-related exceptions if there are encoding problems with the translated strings. It's much better to use alternatives which return Unicode strings in Python 3. These functions have been broken for a long time.

  Function **`bind_textdomain_codeset()`**, methods **`NullTranslations.output_charset()`** and **`NullTranslations.set_output_charset()`**, and the *codeset* parameter of functions `translation()` and `install()` are also deprecated, since they are only used for the `l*gettext()` functions. (Contributed by Serhiy Storchaka in bpo-33710 [https://bugs.python.org/issue?@action=redirect&bpo=33710].)

- The **`isAlive()`** method of `threading.Thread` has been deprecated. (Contributed by Donghee Na in bpo-35283 [https://bugs.python.org/issue?@action=redirect&bpo=35283].)

- Muchas funciones incorporadas y de extensión que toman argumentos enteros ahora emitirán una advertencia de deprecación para `Decimal`s, `Fraction`s y cualquier otro objeto que se pueda convertir a enteros solamente con pérdida (por ejemplo, aquellos que tienen el método **`__int__()`** pero no el método **`__index__()`**). En una versión futura, esto constituirá un error. (Contribución de Serhiy Storchaka en bpo-36048 [https://bugs.python.org/issue?@action=redirect&bpo=36048].)

- El paso de los siguientes argumentos como argumentos por palabra clave está ahora obsoleto:

  - *func* in `functools.partialmethod()`, `weakref.finalize()`, `profile.Profile.runcall()`,

    `cProfile.Profile.runcall()`, `bdb.Bdb.runcall()`, `trace.Trace.runfunc()` and `curses.wrapper()`.
  - *function* en `unittest.TestCase.addCleanup()`.
  - *fn* en el método `submit()` de las clases `concurrent.futures.ThreadPoolExecutor` y `concurrent.futures.ProcessPoolExecutor`.
  - *callback* in `contextlib.ExitStack.callback()`, `contextlib.AsyncExitStack.callback()` and `contextlib.AsyncExitStack.push_async_callback()`
  - *c* and *typeid* in the `create()` method of `multiprocessing.managers.Server` and `multiprocessing.managers.SharedMemoryServer`.
  - *obj* en `weakref.finalize()`.

  En futuros lanzamientos de Python, todos ellos serán argumentos solo posicionales. (Contribución de Serhiy Storchaka en bpo-36492 [https://bugs.python.org/issue?@action=redirect&bpo=36492].)

APIs y características eliminadas

Las siguientes características y APIs se han eliminado de Python 3.8:

- Starting with Python 3.3, importing ABCs from `collections` was deprecated, and importing should be done from `collections.abc`. Being able to import from collections was marked for removal in 3.8, but has been delayed to 3.9. (See gh-81134 [https://github.com/python/cpython/issues/81134].)
- The `macpath` module, deprecated in Python 3.7, has been removed. (Contributed by Victor Stinner in bpo-35471 [https://bugs.python.org/issue?@action=redirect&bpo=35471].)
- The function `platform.popen()` has been removed, after having been deprecated since Python 3.3: use `os.popen()` instead. (Contributed by Victor Stinner in bpo-35345 [https://bugs.python.org/issue?@action=redirect&bpo=35345].)
- The function `time.clock()` has been removed, after having been deprecated since Python 3.3: use

`time.perf_counter()` or **`time.process_time()`** instead, depending on your requirements, to have well-defined behavior. (Contributed by Matthias Bussonnier in bpo-36895 [https://bugs.python.org/issue?@action=redirect&bpo=36895].)

- El script `pyvenv` se ha eliminado, en favor de `python3.8 -m venv`, para ayudar a eliminar la confusión sobre a qué intérprete de Python está vinculado el script `pyvenv`. (Contribución de Brett Cannon en bpo-25427 [https://bugs.python.org/issue?@action=redirect&bpo=25427].)
- `parse_qs`, `parse_qsl`, and `escape` are removed from the **`cgi`** module. They are deprecated in Python 3.2 or older. They should be imported from the `urllib.parse` and `html` modules instead.
- La función `filemode` se ha eliminado del módulo **`tarfile`**. Estaba indocumentada y obsoleta desde Python 3.3.
- El constructor de **`XMLParser`** ya no acepta el argumento *html*. Nunca tuvo efecto y quedó obsoleto en Python 3.4. Todos los demás parámetros son ahora parámetros solo nombrados. (Contribución de Serhiy Storchaka en bpo-29209 [https://bugs.python.org/issue?@action=redirect&bpo=29209].)
- Se ha eliminado el método `doctype()` de **`XMLParser`**. (Contribución de Serhiy Storchaka en bpo-29209 [https://bugs.python.org/issue?@action=redirect&bpo=29209].)
- Se ha elimina el códec «unicode_internal». (Contribución de Inada Naoki en bpo-36297 [https://bugs.python.org/issue?@action=redirect&bpo=36297].)
- Los objetos `Cache` y `Statement` del módulo **`sqlite3`** no estarán expuestos al usuario a partir de ahora. (Contribución de Aviv Palivoda en bpo-30262 [https://bugs.python.org/issue?@action=redirect&bpo=30262].)
- El argumento por palabra clave `bufsize` de **`fileinput.input()`** y **`fileinput.FileInput()`**, marcado como obsoleto e ignorado desde Python 3.6, ha sido eliminado. bpo-36952 [https://bugs.python.org/issue?@action=redirect&bpo=36952] (Contribución de Matthias Bussonnier.)
- The functions **`sys.set_coroutine_wrapper()`** and **`sys.get_coroutine_wrapper()`** deprecated in Python

3.7 have been removed; bpo-36933 [https://bugs.python.org/issue?@action=redirect&bpo=36933] (Contributed by Matthias Bussonnier.)

Portando a Python 3.8

Esta sección enumera los cambios descritos anteriormente y otras correcciones de errores que pueden requerir cambios en tu código.

Cambios en el comportamiento de Python

- Las expresiones yield (tanto la cláusula `yield` como la cláusula `yield from`) ahora no están permitidas en comprensiones y expresiones generadoras (excepto en la expresión iterable en la cláusula **`for`** situada más a la izquierda). (Contribución de Serhiy Storchaka en bpo-10544 [https://bugs.python.org/issue?@action=redirect&bpo=10544].)
- El compilador ahora produce una advertencia **`SyntaxWarning`** cuando se utilizan comprobaciones de identidad (`is` e `is not`) con ciertos tipos de literales (por ejemplo, cadenas, números). A menudo, estas pueden funcionar accidentalmente en CPython, pero no está garantizado por las especificaciones del lenguaje. La advertencia advierte a los usuarios que utilicen pruebas de igualdad (== y `!=`) en su lugar. (Contribución de Serhiy Storchaka en bpo-34850 [https://bugs.python.org/issue?@action=redirect&bpo=34850].)
- El intérprete de CPython puede tolerar excepciones en algunas circunstancias. En Python 3.8 esto sucederá con menos frecuencia. En particular, las excepciones que se generan al obtener atributos del diccionario de tipos ya no son ignoradas. (Contribución de Serhiy Storchaka en bpo-35459 [https://bugs.python.org/issue?@action=redirect&bpo=35459].)
- Se ha eliminado las implementaciones de `__str__` para los tipos incorporados **`bool`**, **`int`**, **`float`**, **`complex`** y algunas clases de la biblioteca estándar. Ahora heredan el método `__str__()` de **`object`**. Como resultado, definir el método `__repr__()` en una subclase de estas clases afectará a su

representación como cadena de caracteres. (Contribución de Serhiy Storchaka en bpo-36793 [https://bugs.python.org/issue?@action=redirect&bpo=36793].)

- En AIX, el atributo `sys.platform` ya no contiene la versión principal. Es decir, siempre es `'aix'`, en lugar de `'aix3'` .. `'aix7'`. Dado que las versiones anteriores de Python incluyen el número de versión, se recomienda usar siempre `sys.platform.startswith('aix')`. (Contribución de M. Felt en bpo-36588 [https://bugs.python.org/issue?@action=redirect&bpo=36588].)
- `PyEval_AcquireLock()` y `PyEval_AcquireThread()` ahora terminan el hilo actual si se llaman mientras el intérprete está finalizando, haciéndolos consistentes con `PyEval_RestoreThread()`, `Py_END_ALLOW_THREADS()` y `PyGILState_Ensure()`. Si no se desea este comportamiento, se tiene que proteger la invocación comprobando `_Py_IsFinalizing()` o `sys.is_finalizing()`. (Contribución de Joannah Nanjekye en bpo-36475 [https://bugs.python.org/issue?@action=redirect&bpo=36475].)

Cambios en la API de Python

- La función `os.getcwdb()` ahora usa la codificación UTF-8 en Windows, en lugar de la página de códigos ANSI: consultar **PEP 529** [https://peps.python.org/pep-0529/] para el fundamento. La función ya no está obsoleta en Windows. (Contribución de Victor Stinner en bpo-37412 [https://bugs.python.org/issue?@action=redirect&bpo=37412].)
- `subprocess.Popen` can now use `os.posix_spawn()` in some cases for better performance. On Windows Subsystem for Linux and QEMU User Emulation, the `Popen` constructor using `os.posix_spawn()` no longer raises an exception on errors like «missing program». Instead the child process fails with a non-zero `returncode`. (Contributed by Joannah Nanjekye and Victor Stinner in bpo-35537 [https://bugs.python.org/issue?@action=redirect&bpo=35537].)
- El argumento *preexec_fn* de `subprocess.Popen` ya no es compatible con subintérpretes. El uso del parámetro en un subintérprete ahora lanza una excepción `RuntimeError`.

(Contribución de Eric Snow en bpo-34651 [https://bugs.python.org/issue?@action=redirect&bpo=34651], modificado por Christian Heimes en bpo-37951 [https://bugs.python.org/issue?@action=redirect&bpo=37951].)

- The `imaplib.IMAP4.logout()` method no longer silently ignores arbitrary exceptions. (Contributed by Victor Stinner in bpo-36348 [https://bugs.python.org/issue?@action=redirect&bpo=36348].)
- The function `platform.popen()` has been removed, after having been deprecated since Python 3.3: use `os.popen()` instead. (Contributed by Victor Stinner in bpo-35345 [https://bugs.python.org/issue?@action=redirect&bpo=35345].)
- La función `statistics.mode()` ya no lanza una excepción cuando se proporcionan datos multimodales. Ahora, en cambio, retorna la primera moda encontrada en los datos de entrada. (Contribución de Raymond Hettinger en bpo-35892 [https://bugs.python.org/issue?@action=redirect&bpo=35892].)
- El método `selection()` de la clase `tkinter.ttk.Treeview` ya no acepta argumentos. Usarlo con argumentos para cambiar la selección quedó obsoleto en Python 3.6. Utiliza métodos especializados, como `selection_set()`, para cambiar la selección. (Contribución de Serhiy Storchaka en bpo-31508 [https://bugs.python.org/issue?@action=redirect&bpo=31508].)
- The `writexml()`, `toxml()` and `toprettyxml()` methods of `xml.dom.minidom` and the `write()` method of `xml.etree.ElementTree` now preserve the attribute order specified by the user. (Contributed by Diego Rojas and Raymond Hettinger in bpo-34160 [https://bugs.python.org/issue?@action=redirect&bpo=34160].)
- Una base de datos `dbm.dumb` abierta con el flag `'r'` ahora es de solo lectura. `dbm.dumb.open()` con los flags `'r'` y `'w'` ya no crea una base de datos si no existe. (Contribución de Serhiy Storchaka en bpo-32749 [https://bugs.python.org/issue?@action=redirect&bpo=32749].)
- El método `doctype()` definido en una subclase de `XMLParser` ya no será invocado y emitirá una advertencia `RuntimeWarning` en lugar de `DeprecationWarning`. Define el método `doctype()` en un objetivo para manejar una declaración doctype de XML. (Contribución de Serhiy

Storchaka en bpo-29209 [https://bugs.python.org/issue?@action=redirect&bpo=29209].)

- Ahora se lanza una excepción **`RuntimeError`** cuando la metaclase personalizada no proporciona la entrada `__classcell__` en el espacio de nombres pasado a `type.__new__`. En Python 3.6–3.7. se emitía una advertencia **`DeprecationWarning`** (Contribución de Serhiy Storchaka en bpo-23722 [https://bugs.python.org/issue?@action=redirect&bpo=23722].)
- The **`cProfile.Profile`** class can now be used as a context manager. (Contributed by Scott Sanderson in bpo-29235 [https://bugs.python.org/issue?@action=redirect&bpo=29235].)
- **`shutil.copyfile()`**, **`shutil.copy()`**, **`shutil.copy2()`**, **`shutil.copytree()`** y **`shutil.move()`** usan llamadas al sistema de «copia-rápida» específicas de la plataforma. (Consultar la sección Operaciones de copia eficientes dependientes de la plataforma).
- El tamaño predeterminado del búfer de **`shutil.copyfile()`** en Windows se ha cambiado de 16 KiB a 1 MiB.
- La estructura `PyGC_Head` ha cambiado por completo. Todo código que haga uso de algún miembro de la estructura debe reescribirse. (Consultar bpo-33597 [https://bugs.python.org/issue?@action=redirect&bpo=33597].)
- La estructura **`PyInterpreterState`** se ha movido a los archivos de cabeceras «internos» (específicamente a Include/internal/pycore_pystate.h). Un `PyInterpreterState` opaco todavía está disponible como parte de la API pública (y ABI estable). La documentación indica que ninguno de los campos de la estructura es público, por lo que esperamos que nadie los haya estado usando. Sin embargo, si utilizas uno o más de esos campos privados y no tienes otra alternativa, abre una issue BPO. Trabajaremos para ayudarte a adaptarlo (posiblemente incluyendo funciones de acceso a la API pública). (Consultar bpo-35886 [https://bugs.python.org/issue?@action=redirect&bpo=35886].)
- El método **`mmap.flush()`** ahora retorna `None` en caso de éxito y lanza una excepción en caso de error, en todas las plataformas. Anteriormente, su comportamiento dependía de

la plataforma: en Windows retornaba un valor distinto de cero en caso de éxito y cero ante un error. En Unix se retornaba un valor de cero en caso de éxito y se lanzaba una excepción ante un error. (Contribución de Berker Peksag en bpo-2122 [https://bugs.python.org/issue?@action=redirect&bpo=2122].)

- Los módulos `xml.dom.minidom` y `xml.sax` ya no procesan entidades externas de forma predeterminada. (Contribución de Christian Heimes en bpo-17239 [https://bugs.python.org/issue?@action=redirect&bpo=17239].)
- Deleting a key from a read-only `dbm` database (`dbm.dumb`, `dbm.gnu` or `dbm.ndbm`) raises `error` (`dbm.dumb.error`, `dbm.gnu.error` or `dbm.ndbm.error`) instead of `KeyError`. (Contributed by Xiang Zhang in bpo-33106 [https://bugs.python.org/issue?@action=redirect&bpo=33106].)
- AST simplificado para literales. Todas las constantes se representarán como instancias de `ast.Constant`. La instanciación de las antiguas clases `Num`, `Str`, `Bytes`, `NameConstant` y `Ellipsis` retornará ahora una instancia de `Constant`. (Contribución de Serhiy Storchaka en bpo-32892 [https://bugs.python.org/issue?@action=redirect&bpo=32892].)
- `expanduser()` en Windows ahora usa preferentemente la variable de entorno `USERPROFILE` y no usa `HOME`, que normalmente no está establecido para cuentas de usuario normales. (Contribución de Anthony Sottile en bpo-36264 [https://bugs.python.org/issue?@action=redirect&bpo=36264].)
- La excepción `asyncio.CancelledError` ahora hereda de `BaseException` en lugar de `Exception` y ya no hereda de `concurrent.futures.CancelledError`. (Contribuido por Yury Selivanov en bpo-32528 [https://bugs.python.org/issue?@action=redirect&bpo=32528].)
- La función `asyncio.wait_for()` ahora espera correctamente la cancelación cuando se usa una instancia de `asyncio.Task`. Anteriormente, al alcanzar *timeout*, se cancelaba y retornaba de inmediato. (Contribución de Elvis Pranskevichus en bpo-32751 [https://bugs.python.org/issue?@action=redirect&bpo=32751].)
- La función `asyncio.BaseTransport.get_extra_info()` ahora

retorna un objeto socket seguro cuando se pasa "socket" al parámetro *name*. (Contribución de Yury Selivanov en bpo-37027 [https://bugs.python.org/issue?@action=redirect&bpo=37027].)

- **`asyncio.BufferedProtocol`** ha pasado a formar parte de la API estable.

- Las dependencias de DLLs para módulos de extensión y DLLs cargadas con **`ctypes`** en Windows ahora se resuelven de forma más segura. Solo las rutas del sistema, el directorio que contiene el archivo DLL o PYD y los directorios agregados mediante **`add_dll_directory()`** se buscan para las dependencias en tiempo de carga. Específicamente, **`PATH`** y el directorio de trabajo actual ya no se utilizan, y las modificaciones de estos ya no tendrán ningún efecto en la resolución normal de la DLL. Si tu aplicación se basa en estos mecanismos, debes buscar **`add_dll_directory()`** y, si existe, utilizarlo para agregar tu directorio DLL mientras carga tu biblioteca. Ten en cuenta que los usuarios de Windows 7 deberán asegurarse de que se haya instalado Windows Update KB2533623 (esto también lo verifica el instalador). (Contribución de Steve Dower en bpo-36085 [https://bugs.python.org/issue?@action=redirect&bpo=36085].)
- Los archivos de cabeceras y las funciones relacionadas con pgen se han eliminado después de su reemplazo por una implementación pura de Python. (Contribución de Pablo Galindo en bpo-36623 [https://bugs.python.org/issue?@action=redirect&bpo=36623].)
- **`types.CodeType`** tiene un nuevo parámetro en la segunda posición del constructor (*posonlyargcount*) para admitir argumentos solo posicionales, definidos en **PEP 570** [https://peps.python.org/pep-0570/]. El primer argumento (*argcount*) ahora representa el número total de argumentos posicionales (incluidos los argumentos solo posicionales). El nuevo método `replace()` de **`types.CodeType`** se puede utilizar para hacer que el código esté preparado para el futuro.
- El parámetro `digestmod` de **`hmac.new()`** ya no usa el resumen MD5 por defecto.

Cambios en la API de C

- La estructura `PyCompilerFlags` obtuvo un nuevo campo *cf_feature_version*. Debe inicializarse en `PY_MINOR_VERSION`. El campo se ignora de forma predeterminada y se usa solo si el indicador `PyCF_ONLY_AST` está establecido en *cf_flags*. (Aportado por Guido van Rossum en bpo-35766 [https://bugs.python.org/issue?@action=redirect&bpo=35766].)

- La función **`PyEval_ReInitThreads()`** se ha eliminado de la API de C. No debe llamarse explícitamente: usa `PyOS_AfterFork_Child()` en su lugar. (Contribución de Victor Stinner en bpo-36728 [https://bugs.python.org/issue?@action=redirect&bpo=36728].)

- En Unix, las extensiones de C ya no están vinculadas a libpython, excepto en Android y Cygwin. Cuando Python está integrado, `libpython` no debe cargarse con `RTLD_LOCAL`, sino con `RTLD_GLOBAL` en su lugar. Anteriormente, no era posible usar `RTLD_LOCAL` para cargar extensiones de C que no estuvieran vinculadas a `libpython`, como las extensiones de C de la biblioteca estándar construida por la sección `*shared*` de `Modules/Setup`. (Contribución de Victor Stinner en bpo-21536 [https://bugs.python.org/issue?@action=redirect&bpo=21536].)

- El uso de variantes de formato `#` en el análisis o la construcción de valores (por ejemplo: `PyArg_ParseTuple()`, `Py_BuildValue()`, `PyObject_CallFunction()`, etc.) sin `PY_SSIZE_T_CLEAN` definido ahora lanza una advertencia `DeprecationWarning`. Se eliminará en Python 3.10 ó 4.0. Consultar Analizando argumentos y construyendo valores para más detalles. (Contribución de Inada Naoki en bpo-36381 [https://bugs.python.org/issue?@action=redirect&bpo=36381].)

- Instances of heap-allocated types (such as those created with `PyType_FromSpec()`) hold a reference to their type object. Increasing the reference count of these type objects has been moved from `PyType_GenericAlloc()` to the more low-level functions, `PyObject_Init()` and **`PyObject_INIT`**. This makes types created through **`PyType_FromSpec()`**

behave like other classes in managed code.

Statically allocated types no se ve afectado.

Para la gran mayoría de casos, no debería haber efectos secundarios. Sin embargo, los tipos que aumentan manualmente el recuento de referencias después de asignar una instancia (quizás para evitar el error) ahora pueden volverse inmortales. Para evitar esto, estas clases deben llamar a Py_DECREF en el objeto de tipo durante la desasignación de la instancia.

Para portar correctamente estos tipos a Python 3.8, aplica los siguientes cambios:

- Remove `Py_INCREF` on the type object after allocating an instance - if any. This may happen after calling `PyObject_New`, `PyObject_NewVar`, `PyObject_GC_New()`, `PyObject_GC_NewVar()`, or any other custom allocator that uses `PyObject_Init()` or **`PyObject_INIT`**.

  Ejemplo:

  ```
  static foo_struct *
  foo_new(PyObject *type) {
      foo_struct *foo = PyObject_GC_New(foo_struc
      if (foo == NULL)
          return NULL;
  #if PY_VERSION_HEX < 0x03080000
      // Workaround for Python issue 35810; no lo
      PY_INCREF(type)
  #endif
      return foo;
  }
  ```

- Asegúrate de que todas las funciones personalizadas `tp_dealloc` de los tipos asignados al montón disminuyan el recuento de referencias del tipo.

  Ejemplo:

```
static void
foo_dealloc(foo_struct *instance) {
    PyObject *type = Py_TYPE(instance);
    PyObject_GC_Del(instance);
#if PY_VERSION_HEX >= 0x03080000
    // This was not needed before Python 3.8 (E
    Py_DECREF(type);
#endif
}
```

(Contribución de Eddie Elizondo en bpo-35810 [https://bugs.python.org/issue?@action=redirect&bpo=35810].)

- La macro **`Py_DEPRECATED()`** ha sido implementada para MSVC. La macro ahora debe ser colocada antes del nombre del símbolo.

  Ejemplo:

  ```
  Py_DEPRECATED(3.8) PyAPI_FUNC(int) Py_OldFunction(v
  ```

  (Contribución de Zackery Spytz en bpo-33407 [https://bugs.python.org/issue?@action=redirect&bpo=33407].)

- El intérprete ya no pretende dar suporte nunca más a la compatibilidad binaria de tipos de extensión entre versiones de características. Un **`PyTypeObject`** exportado por un módulo de extensión de terceros se supone que tiene todas las ranuras esperadas por la versión actual de Python, incluyendo **`tp_finalize`** (**`Py_TPFLAGS_HAVE_FINALIZE`** ya no se verifica antes de leer **`tp_finalize`**).

  (Contribución de Antoine Pitrou en bpo-32388 [https://bugs.python.org/issue?@action=redirect&bpo=32388].)

- Las funciones **`PyNode_AddChild()`** y **`PyParser_AddToken()`** ahora aceptan dos argumentos `int` adicionales, *end_lineno* y *end_col_offset*.

- El archivo `libpython38.a` ya no se incluye en la distribución normal de Windows. Si se necesita este archivo, se puede generar con las herramientas `gendef` y `dlltool`, que son parte del paquete binutils de MinGW:

```
gendef - python38.dll > tmp.def
dlltool --dllname python38.dll --def tmp.def --outp
```

La ubicación de un archivo `pythonXY.dll` instalado dependerá de las opciones de instalación y de la versión y el idioma de Windows. Consultar Uso de Python en Windows para obtener más información. La biblioteca resultante debe colocarse en el mismo directorio que `pythonXY.lib`, que generalmente es el directorio `libs` en tu instalación de Python.

(Contribución de Steve Dower en bpo-37351 [https://bugs.python.org/issue?@action=redirect&bpo=37351].)

Cambios en el bytecode de CPython

- El bucle del intérprete se ha simplificado moviendo al compilador la lógica para el desenredo de la pila de bloques. El compilador ahora emite instrucciones explícitas para ajustar la pila de valores y llamar al código de limpieza para **`break`**, **`continue`** y **`return`**.

  Removed opcodes **`BREAK_LOOP`**, **`CONTINUE_LOOP`**, **`SETUP_LOOP`** and **`SETUP_EXCEPT`**. Added new opcodes **`ROT_FOUR`**, **`BEGIN_FINALLY`**, **`CALL_FINALLY`** and **`POP_FINALLY`**. Changed the behavior of **`END_FINALLY`** and **`WITH_CLEANUP_START`**.

  (Contribución de Mark Shannon, Antoine Pitrou y Serhiy Storchaka en bpo-17611 [https://bugs.python.org/issue?@action=redirect&bpo=17611].)

- Agregado el código de operación **`END_ASYNC_FOR`** para manejar excepciones lanzadas mientras se espera al siguiente elemento en un ciclo **`async for`**. (Contribución de Serhiy Storchaka en bpo-33041 [https://bugs.python.org/issue?@action=redirect&bpo=33041].)

- El código de operación **`MAP_ADD`** ahora espera el valor como primer elemento de la pila y la clave como segundo elemento. Este cambio se realizó para que la clave siempre se evalúe antes que el valor en los diccionarios por compresión, como se propone en **PEP 572** [https://peps.python.org/pep-0572/]. (Contribución de Jörn Heissler en bpo-35224 [https://bugs.python.org/issue?@action=redirect&bpo=35224].)

Demos y herramientas

Se ha añadido un script de evaluación de rendimiento para cronometrar varias formas de acceder a variables: `Tools/scripts/var_access_benchmark.py`. (Contribución de Raymond Hettinger en bpo-35884 [https://bugs.python.org/issue?@action=redirect&bpo=35884].)

A continuación, se muestra un resumen de las mejoras de rendimiento desde Python 3.3:

```
Python version                       3.3     3.4     3.5
--------------                       ---     ---     ---

Variable and attribute read access:
    read_local                       4.0     7.1     7.1
    read_nonlocal                    5.3     7.1     8.1
    read_global                     13.3    15.5    19.0
    read_builtin                    20.0    21.1    21.6
    read_classvar_from_class        20.5    25.6    26.5
    read_classvar_from_instance     18.5    22.8    23.5
    read_instancevar                26.8    32.4    33.1
    read_instancevar_slots          23.7    27.8    31.3
    read_namedtuple                 68.5    73.8    57.5
    read_boundmethod                29.8    37.6    37.9

Variable and attribute write access:
    write_local                      4.6     8.7     9.3
    write_nonlocal                   7.3    10.5    11.1
    write_global                    15.9    19.7    21.2
    write_classvar                  81.9    92.9    96.0
    write_instancevar               36.4    44.6    45.8
    write_instancevar_slots         28.7    35.6    36.1

Data structure read access:
    read_list                       19.2    24.2    24.5
    read_deque                      19.9    24.7    25.5
    read_dict                       19.7    24.3    25.7
    read_strdict                    17.9    22.6    24.3

Data structure write access:
    write_list                      21.2    27.1    28.5
    write_deque                     23.8    28.7    30.1
    write_dict                      25.9    31.4    33.3
    write_strdict                   22.9    28.4    29.9

Stack (or queue) operations:
    list_append_pop                144.2    93.4   112.7
    deque_append_pop                30.4    43.5    57.0
    deque_append_popleft            30.8    43.7    57.3

Timing loop:
    loop_overhead                    0.3     0.5     0.6
```

Las evaluaciones de rendimiento se realizaron en un procesador Intel® Core™ i7-4960HQ [https://ark.intel.com/content/www/us/en/ark/products/76088/intel-core-i7-4960hq-processor-6m-cache-up-to-3-80-ghz.html] ejecutando las compilaciones de 64-bits para macOS disponibles en python.org [https://www.python.org/downloads/macos/]. El script de evaluación de rendimiento muestra los tiempos en nanosegundos.

Cambios notables en Python 3.8.1

Debido a importantes preocupaciones de seguridad, el parámetro *reuse_address* de **`asyncio.loop.create_datagram_endpoint()`** ya no está soportado. Esto se debe al comportamiento de la opción `SO_REUSEADDR` del socket en UDP. Para obtener más detalles, consulta la documentación de `loop.create_datagram_endpoint()`. (Contribución de Kyle

Stanley, Antoine Pitrou y Yury Selivanov en bpo-37228 [https://bugs.python.org/issue?@action=redirect&bpo=37228].)

Notable changes in Python 3.8.2

Fixed a regression with the `ignore` callback of **`shutil.copytree()`**. The argument types are now str and List[str] again. (Contributed by Manuel Barkhau and Giampaolo Rodola in gh-83571 [https://github.com/python/cpython/issues/83571].)

Notable changes in Python 3.8.3

The constant values of future flags in the **`__future__`** module are updated in order to prevent collision with compiler flags. Previously `PyCF_ALLOW_TOP_LEVEL_AWAIT` was clashing with `CO_FUTURE_DIVISION`. (Contributed by Batuhan Taskaya in gh-83743 [https://github.com/python/cpython/issues/83743])

Cambios notables en Python 3.8.8

Earlier Python versions allowed using both `;` and `&` as query parameter separators in **`urllib.parse.parse_qs()`** and **`urllib.parse.parse_qsl()`**. Due to security concerns, and to conform with newer W3C recommendations, this has been changed to allow only a single separator key, with `&` as the default. This change also affects **`cgi.parse()`** and **`cgi.parse_multipart()`** as they use the affected functions internally. For more details, please see their respective documentation. (Contributed by Adam Goldschmidt, Senthil Kumaran and Ken Jin in bpo-42967 [https://bugs.python.org/issue?@action=redirect&bpo=42967].)

Notable changes in Python 3.8.9

A security fix alters the **`ftplib.FTP`** behavior to not trust the IPv4 address sent from the remote server when setting up a passive data channel. We reuse the ftp server IP address instead. For unusual code requiring the old behavior, set a `trust_server_pasv_ipv4_address` attribute on your FTP instance to `True`. (See gh-87451 [https://github.com/python/cpython/issues/87451])

Notable changes in Python 3.8.10

macOS 11.0 (Big Sur) and Apple Silicon Mac support

As of 3.8.10, Python now supports building and running on macOS 11 (Big Sur) and on Apple Silicon Macs (based on the `ARM64` architecture). A new universal build variant, `universal2`, is now available to natively support both `ARM64` and `Intel 64` in one set of executables. Note that support for «weaklinking», building binaries targeted for newer versions of macOS that will also run correctly on older versions by testing at runtime for missing features, is not included in this backport from Python 3.9; to support a range of macOS versions, continue to target for and build on the oldest version in the range.

(Originally contributed by Ronald Oussoren and Lawrence D'Anna in gh-85272 [https://github.com/python/cpython/issues/85272], with fixes by FX Coudert and Eli Rykoff, and backported to 3.8 by Maxime Bélanger and Ned Deily)

Notable changes in Python 3.8.10

urllib.parse

The presence of newline or tab characters in parts of a URL allows for some forms of attacks. Following the WHATWG specification that updates **RFC 3986** [https://datatracker.ietf.org/doc/html/rfc3986.html], ASCII newline `\n`, `\r` and tab `\t` characters are stripped from the URL by the parser in **`urllib.parse`** preventing such attacks. The removal characters are controlled by a new module level variable `urllib.parse._UNSAFE_URL_BYTES_TO_REMOVE`. (See bpo-43882 [https://bugs.python.org/issue?@action=redirect&bpo=43882])

Cambios notables en Python 3.8.1

Cambios en la API de Python

A partir de Python 3.8.12, el módulo `ipaddress` ya no acepta ceros a la izquierda en las cadenas de direcciones IPv4. Los ceros iniciales son ambiguos y algunas bibliotecas los interpretan como notación octal. Por ejemplo, la función heredada `socket.inet_aton()` trata los ceros iniciales como notación octal. La implementación glibc del `inet_pton()` moderno no acepta ceros a la izquierda.

(Contribuido originalmente por Christian Heimes en bpo-36384 [https://bugs.python.org/issue?@action=redirect&bpo=36384], y actualizado a 3.8 por Achraf Merzouki).

Notable security feature in 3.8.14

Converting between `int` and `str` in bases other than 2 (binary), 4, 8 (octal), 16 (hexadecimal), or 32 such as base 10 (decimal) now raises a `ValueError` if the number of digits in string form is above a limit to avoid potential denial of service attacks due to the algorithmic complexity. This is a mitigation for **CVE 2020-10735** [https://www.cve.org/CVERecord?id=CVE-2020-10735]. This limit can be configured or disabled by environment variable, command line flag, or `sys` APIs. See the integer string conversion length limitation documentation. The default limit is 4300 digits in string form.

Notable changes in 3.8.17

tarfile

- The extraction methods in `tarfile`, and `shutil.unpack_archive()`, have a new a *filter* argument that allows limiting tar features than may be surprising or dangerous, such as creating files outside the destination directory. See Extraction filters for details. In Python 3.12, use without the *filter* argument will show a `DeprecationWarning`. In Python 3.14, the default will switch to `'data'`. (Contributed by Petr Viktorin in **PEP 706** [https://peps.python.org/pep-0706/].)

Qué hay de nuevo en Python 3.7

Editor:

Elvis Pranskevichus <elvis@magic.io>

El artículo explica las nuevas características en Python 3.7, comparado con 3.6. Python 3.7 fue lanzado el 27, Junio, 2018. Para más detalles, consultar la changelog.

Resumen – Lanzamientos Destacados

Sintaxis de nuevas características:

- PEP 563, evaluación pospuesta de anotaciones de tipo.

Cambios de sintaxis incompatibles hacia atrás:

- `async` y `await` ahora son palabras clave reservadas.

Nuevos módulos de biblioteca:

- `contextvars`: PEP 567 – Variables de Contexto
- `dataclasses`: PEP 557 – Clases de Datos
- importlib.resources

Nuevas funciones integradas:

- PEP 553, la nueva función `breakpoint()`.

Mejoras en el modelo de datos de Python:

- PEP 562, personalización del acceso a los atributos del módulo.
- PEP 560, soporte básico para módulo de mecanografía y tipos genéricos.

- la naturaleza de conservación del orden de inserción de objetos dict han sido declarado [https://mail.python.org/pipermail/python-dev/2017-December/151283.html] para ser una parte oficial de la especificación del lenguaje Python.

Mejoras significativas in la librería estándar:

- El modulo **`asyncio`** ha recibido nuevas funciones, significativas mejoras de usabilidad y rendimiento.
- El modulo **`time`** adquirió soporte de funciones con resolución de nanosegundos.

Mejoras en la implementación de CPython:

- Evitando el uso de ASCII como codificación de texto predeterminada:
  - PEP 538, coerción de configuración regional de C heredada
  - PEP 540, modo de tiempo de ejecución UTF-8 forzado
- PEP 552, determinista *.pycs*
- El nuevo modo de ejecución de desarrollo
- PEP 565, manejo mejorado **`DeprecationWarning`**

Mejoras en la API de C:

- PEP 539, nueva API de C para almacenamiento local de subprocesos

Mejoras de Documentación:

- PEP 545, Traducción de documentación de Python
- Nuevas traducciones de documentación: Japonés [https://docs.python.org/ja/], Francés [https://docs.python.org/fr/], y Coreano [https://docs.python.org/ko/].

Esta versión presenta notables mejoras de rendimiento en muchas áreas. La sección Optimizaciones los enumera en detalle.

Para obtener una lista de cambios que pueden afectar la compatibilidad con versiones anteriores de Python por favor consultar la sección Portando a Python 3.7.

Nuevas Características

PEP 563: Evaluación pospuesta de anotaciones

La llegada de las sugerencias de tipo en Python descubrió dos problemas de usabilidad evidentes con la funcionalidad de las anotaciones agregadas en **PEP 3107** [https://peps.python.org/pep-3107/] y mas refinado en **PEP 526** [https://peps.python.org/pep-0526/]:

- las anotaciones solo podían usar nombres que ya estaban disponibles en el alcance actual, en otras palabras, no admitían referencias futuras de ningún tipo; y
- anotar el código fuente tuvo efectos adversos en el tiempo de inicio de los programas Python.

Ambos problemas se solucionan posponiendo la evaluación de anotaciones. En lugar de compilar código que ejecuta expresiones en anotaciones en su momento de definición, el compilador almacena la anotación en forma de cadena equivalente al AST de la expresión en cuestión. Si es necesario, las anotaciones se pueden resolver en tiempo de ejecución usando `typing.get_type_hints()`. En el común de los casos esto no es necesario, las anotaciones son más fáciles de almacenar (dado que las cadenas cortas son internadas por el intérprete.) y hacen que el tiempo de inicio sea mas rápido.

En términos de usabilidad, las anotaciones ahora admiten referencias hacia adelante, lo que hace que la siguiente sintaxis sea válida:

```
class C:
    @classmethod
    def from_string(cls, source: str) -> C:
        ...

    def validate_b(self, obj: B) -> bool:
        ...

class B:
    ...
```

Dado que este cambio rompe la compatibilidad, el nuevo comportamiento debe habilitarse por modulo en Python 3.7 usando una importación `__future__`:

```
from __future__ import annotations
```

Se convertirá en el predeterminado en Python 3.10.

Ver también

PEP 563 [https://peps.python.org/pep-0563/] – Evaluación aplazada de anotaciones

PEP escrito y implementado por *Łukasz Langa*.

PEP 538: Coerción de configuración regional del Legado de C

Un desafío continuo dentro de la serie Python 3 ha sido determinar una estrategia predeterminada sensata para manejar el supuesto de codificación de texto "ASCII de 7 bits" implícito actualmente en el uso de la configuración regional C o POSIX predeterminada en plataformas que no son Windows.

PEP 538 [https://peps.python.org/pep-0538/] actualiza la interfaz de línea de comandos del intérprete predeterminada para convertir automáticamente esa configuración regional en una configuración regional basada en UTF-8 disponible, como se describe en la documentación de la nueva variable de entorno `PYTHONCOERCECLOCALE`. Configurar automáticamente `LC_CTYPE` de esta manera significa que tanto el intérprete central como las extensiones C que reconocen la configuración regional (por ejemplo `readline`) asumirá el uso de UTF-8 como codificación de texto predeterminada, en lugar de ASCII.

La definición de soporte de plataforma en PEP 11 [https://peps.python.org/pep-0011/] también se ha actualizado para limitar la compatibilidad con el manejo de texto completo a configuraciones regionales no basadas en ASCII configuradas adecuadamente.

Como parte de este cambio, el controlador de errores predeterminado para **`stdin`** y **`stdout`** es ahora `surrogateescape` (antes que `strict`) al usar cualquiera de las configuraciones regionales de destino de coerción definidas (actualmente `C.UTF-8`, `C.utf8`, y `UTF-8`). El controlador de errores predeterminado para **`stderr`** continua siendo `backslashreplace`, independientemente de local.

La coerción local es silenciosa de forma predeterminada, pero para ayudar a depurar problemas de integración potencialmente relacionados con la configuración regional, advertencias explícitas (emitido directamente en **`stderr`**) se puede solicitar configurando `PYTHONCOERCECLOCALE=warn`. Esta configuración también hará que el tiempo de ejecución de Python emita una advertencia si la configuración regional C heredada permanece activa cuando se inicializa el intérprete principal.

Mientras la coerción local **PEP 538** [https://peps.python.org/pep-0538/] tiene la ventaja de afectar también a los módulos de extensión (como GNU `readline`), así como procesos secundarios (incluidos los que ejecutan aplicaciones que no son de Python y versiones anteriores de Python), tiene la desventaja de requerir que esté presente una ubicación de destino adecuada en el sistema en ejecución. Para manejar mejor el caso en el que no hay una configuración regional de destino adecuada disponible (como ocurre en RHEL / CentOS 7, por ejemplo), Python 3.7 también implementa PEP 540: Modo de tiempo de ejecución UTF-8 forzado.

Ver también

PEP 538 [https://peps.python.org/pep-0538/] – Coaccionar la configuración regional C heredada a una configuración regional basada en UTF-8

PEP escrito y implementado por *Nick Coghlan*.

PEP 540: Modo de tiempo de ejecución UTF-8 forzado

La nueva opción de lineas de comando **`-X`** `utf8` y la variable de entorno **`PYTHONUTF8`** se puede utilizar para habilitar el modo Python UTF-8.

Cuando está en modo UTF-8, CPython ignora la configuración regional y usa la codificación UTF-8 de forma predeterminada. Los controladores de errores para los flujos `sys.stdin` y `sys.stdout` se establecen en `surrogateescape`.

El modo UTF-8 forzado se puede usar para cambiar el comportamiento de manejo de texto en un intérprete de Python integrado sin cambiar la configuración regional de una aplicación de inserción.

Mientras el modo UTF-8 **PEP 540** [https://peps.python.org/pep-0540/] tiene la ventaja de funcionar independientemente de las configuraciones regionales disponibles en el sistema en ejecución, tiene la desventaja de no tener ningún efecto en los módulos de extensión (como GNU `readline`), procesos secundarios que ejecutan aplicaciones que no son de Python y procesos secundarios que ejecutan versiones anteriores de Python. Para reducir el riesgo de dañar los datos de texto al comunicarse con dichos componentes, Python 3.7 también implementa PEP 540: Modo de tiempo de ejecución UTF-8 forzado.

El modo UTF-8 está habilitado de forma predeterminada cuando la configuración regional es `C` o `POSIX`, y la función de coerción local **PEP 538** [https://peps.python.org/pep-0538/] no lo cambia a una alternativa basada en UTF-8 (si ese fallo se debe a que se ha establecido `PYTHONCOERCECLOCALE=0`, `LC_ALL` se esta configurando, o la falta de un lugar de destino adecuado).

Ver también

PEP 540 [https://peps.python.org/pep-0540/] – Agrega un nuevo modo UTF-8

PEP escrito y implementado por *Victor Stinner*

PEP 553: Incorporada en `breakpoint()`

Python 3.7 incluye la nueva función incorporada `breakpoint()` como una forma fácil y consistente de ingresar al depurador de Python.

Llamadas `breakpoint()` incorporadas en **`sys.breakpointhook()`**. Por defecto, este último importa **`pdb`** y luego llama `pdb.set_trace()`, pero vinculando `sys.breakpointhook()` a la función de su elección, `breakpoint()` puede entrar en cualquier depurador. Adicionalmente, la variable de entorno **`PYTHONBREAKPOINT`** se puede configurar como el depurador que elija. Configure `PYTHONBREAKPOINT=0` para deshabilitar completamente el `breakpoint()`.

Ver también

PEP 553 [https://peps.python.org/pep-0553/] – Incorporada en *breakpoint()*

PEP escrito y implementado por *Barry Warsaw*

PEP 539: Nueva API C para almacenamiento local de subprocesos

Mientras que Python proporciona una API C para soporte de almacenamiento local de subprocesos; el Thread Local Storage (TLS) API existente ha usado int para representar claves TLS en todas las plataformas. En general, esto no ha sido un problema para las plataformas de soporte oficial, pero no es compatible con POSIX ni portátil en ningún sentido práctico.

PEP 539 [https://peps.python.org/pep-0539/] cambia esto al proporcionar un nuevo Thread Specific Storage (TSS) API a CPython que reemplaza el uso de la API TLS existente dentro del intérprete de CPython, al tiempo que deja obsoleta la API existente. La API de TSS utiliza un nuevo tipo **`Py_tss_t`** en lugar de int para representar claves TSS, un tipo opaco cuya definición puede depender de la implementación de TLS subyacente. Por lo tanto, esto permitirá compilar CPython en plataformas donde la clave TLS nativa se define de una manera que no se puede convertir de manera segura a int.

Tenga en cuenta que en las plataformas en las que la clave TLS nativa se define de una manera que no se puede convertir de forma segura a int, todas las funciones de la API de TLS existente no funcionarán e inmediatamente devolverán un error. Esto indica claramente que la antigua API no es compatible con plataformas en las que no se puede usar de manera confiable y que no se hará ningún esfuerzo para agregar dicho soporte.

Ver también

PEP 539 [https://peps.python.org/pep-0539/] – Un nueva *API-C* para almacenamiento local de subprocesos en *CPython*
PEP escrito por *Erik M. Bray*; implementación por *Masayuki Yamamoto*.

PEP 562: Personalización del acceso a los atributos del módulo

Python 3.7 allows defining `__getattr__()` on modules and will call it whenever a module attribute is otherwise not found. Defining `__dir__()` on modules is now also allowed.

Un ejemplo típico de dónde esto puede ser útil es la desaprobación de los atributos del módulo y la carga diferida.

Ver también

PEP 562 [https://peps.python.org/pep-0562/] – Módulo `__getattr__` y `__dir__`
PEP escrito y implementado por *Ivan Levkivskyi*

PEP 564: Nuevas funciones de tiempo con resolución de nanosegundos

La resolución de los relojes en los sistemas modernos puede superar la precisión limitada de un número de punto flotante devuelto por la función `time.time()` y sus variantes. Para evitar la pérdida de precisión, PEP 564 [https://peps.python.org/pep-0564/] agrega seis nuevas variantes de «nanosegundos» de las funciones de temporizador existentes al módulo `time`:

- `time.clock_gettime_ns()`
- `time.clock_settime_ns()`
- `time.monotonic_ns()`
- `time.perf_counter_ns()`
- `time.process_time_ns()`
- `time.time_ns()`

Las nuevas funciones devuelven el número de nanosegundos como un valor entero.

Measurements [https://peps.python.org/pep-0564/#annex-clocks-resolution-in-python] muestra que en Linux y Windows la resolución de `time.time_ns()` es aproximadamente 3 veces mejor que la de `time.time()`.

Ver también

PEP 564 [https://peps.python.org/pep-0564/] – Agrega nuevas funciones de tiempo y resolución de nanosegundos
PEP escrito y implementado por *Victor Stinner*

PEP 565: Mostrar *DeprecationWarning* en `__main__`

El manejo predeterminado de `DeprecationWarning` Se ha cambiado de modo que estas advertencias se muestren una vez más de forma predeterminada, pero solo cuando el código que las activa se ejecuta directamente en el módulo `__main__`. Como resultado, los desarrolladores de scripts de un solo archivo y aquellos que usan Python de forma interactiva deberían comenzar a ver una vez más advertencias de desaprobación para las API que usan, pero las advertencias de desaprobación activadas por la aplicación importada, la biblioteca y los módulos de marco seguirán ocultos de forma predeterminada.

Como resultado de este cambio, la biblioteca estándar ahora permite a los desarrolladores elegir entre tres comportamientos de advertencia de obsolescencia diferentes:

- `FutureWarning`: siempre se muestra por defecto, recomendado para las advertencias destinadas a los usuarios finales de la aplicación (por ejemplo, para los ajustes de configuración de la aplicación obsoletos).

- `DeprecationWarning`: se muestra por defecto solo en `__main__` y cuando se ejecutan pruebas, se recomienda para advertencias destinadas a ser vistas por otros desarrolladores de Python donde una actualización de la versión puede resultar en un cambio de comportamiento o un error.
- `PendingDeprecationWarning`: se muestra de forma predeterminada solo cuando se ejecutan pruebas, diseñado para casos en los que una actualización de versión futura cambiará la categoría de advertencia a `DeprecationWarning` o `FutureWarning`.

Anteriormente ambos `DeprecationWarning` y `PendingDeprecationWarning` solo eran visibles cuando se ejecutaban pruebas, lo que significaba que los desarrolladores que escribían principalmente scripts de un solo archivo o usaban Python de forma interactiva podrían sorprenderse al ver cambios importantes en las API que usaban.

Ver también

PEP 565 [https://peps.python.org/pep-0565/] – Muestra *DeprecationWarning* en `__main__`

PEP escrito y implementado por *Nick Coghlan*

PEP 560: Soporte básico para el módulo de `typing` y tipos genéricos

Initially PEP 484 [https://peps.python.org/pep-0484/] was designed in such way that it would not introduce *any* changes to the core CPython interpreter. Now type hints and the `typing` module are extensively used by the community, so this restriction is removed. The PEP introduces two special methods `__class_getitem__()` and `__mro_entries__()`, these methods are now used by most classes and special constructs in `typing`. As a result, the speed of various operations with types increased up to 7 times, the generic types can be used without metaclass conflicts, and several long standing bugs in `typing` module are fixed.

Ver también

PEP 560 [https://peps.python.org/pep-0560/] – Soporte básico para el modulo de escritura y tipos genéricos
PEP escrito y implementado por *Ivan Levkivskyi*

PEP 552: Archivos .pyc basados en hash

Python ha verificado tradicionalmente la actualización de los archivos de caché de código de bytes (es decir, archivos `.pyc`) comparando los metadatos de origen (marca de tiempo y tamaño de última modificación) con la fuente de metadatos guardados en el encabezado del archivo de caché cuando se generó. Si bien es efectivo, este método de invalidación tiene sus inconvenientes. Cuando las marcas de tiempo del sistema de archivos son demasiado burdas, Python puede perder las actualizaciones de origen, lo que genera confusión en el usuario. Además, tener una marca de tiempo en el archivo de caché es problemático para construir reproducibilidad [https://reproducible-builds.org/] y sistemas de construcción basados en contenido.

PEP 552 [https://peps.python.org/pep-0552/] extendiendo el formato *ppc* para permitir que el hash del archivo de origen se utilice para invalidación en lugar de la marca de tiempo de origen. Estos archivos `.pyc` se denominan "hash-based". Por defecto, Python todavía usa la invalidación basada en la marca de tiempo y no genera archivos `.pyc` basados en has de tiempos de ejecución. Los archivos `.pyc` basados en hash se pueden generar con **`py_compile`** o **`compileall`**.

Archivos `.pyc` basados en hash Vienen en dos variantes: marcada y no marcada. Python valida los archivos `.pyc` basados en hash comprobados con los archivos de origen correspondientes en tiempo de ejecución, pero no lo hace para *pycs* basados en hash no verificados. Archivos `.pyc` no comprobados son una optimización de rendimiento útil para entornos donde un sistema externo a Python (por ejemplo, el sistema de compilación) es responsable de mantener actualizados los archivos `.pyc`.

Consultar Invalidación del código de bytes en caché para mas información.

Ver también

PEP 552 [https://peps.python.org/pep-0552/] – *Pycs* deterministas
PEP escrito y implementado por *Benjamin Peterson*

PEP 545: Traducciones de Documentaciones de Python

PEP 545 [https://peps.python.org/pep-0545/] describe el proceso de creación y mantenimiento de traducciones de documentación de Python.

Tres nuevas traducciones han sido agregadas:

- Japonés: https://docs.python.org/ja/
- Francés: https://docs.python.org/fr/
- Coreano: https://docs.python.org/ko/

Ver también

PEP 545 [https://peps.python.org/pep-0545/] – Traducciones de Documentación de *Python*
PEP escrito y implementado por *Julien Palard*, *Inada Naoki* y *Victor Stinner*.

Modo de desarrollo de Python (-X dev)

La nueva opción **`-X`** `dev` opción de línea de comando o la nueva variable de entorno **`PYTHONDEVMODE`** se puede usar para habilitar Python Development Mode. Cuando está en modo de desarrollo, Python realiza comprobaciones de tiempo de ejecución adicionales que son demasiado caras para habilitarse de forma predeterminada. Ver Python Development Mode documentación para la descripción completa.

Otros cambios en el lenguaje

- Una expresión **`await`** y compresiones que contienen una cláusula **`async for`** eran ilegales en las expresiones en formatted string literals debido a un problema con la implementación. En Python 3.7 se eliminó esta restricción.
- Ahora se pueden pasar más de 255 argumentos a una función y una función ahora puede tener más de 255 parámetros. (Contribuido por *Serhiy Storchaka* en bpo-12844 [https://bugs.python.org/issue?@action=redirect&bpo=12844] y bpo-18896 [https://bugs.python.org/issue?@action=redirect&bpo=18896].)
- **`bytes.fromhex()`** y **`bytearray.fromhex()`** ahora ignore todos los espacios en blanco ASCII, no solo los espacios. (Contribuido por *Robert Xiao* en bpo-28927 [https://bugs.python.org/issue?@action=redirect&bpo=28927].)
- **`str`**, **`bytes`**, y **`bytearray`** adquirió soporte para el nuevo método **`isascii()`**, que se puede usar para probar si una cadena o bytes contienen solo los caracteres ASCII. (Contribuido por *INADA Naoki* en bpo-32677 [https://bugs.python.org/issue?@action=redirect&bpo=32677].)
- **`ImportError`** ahora muestra el nombre del módulo y la ruta del módulo `__file__` cuando falla `from ... import ...`. (Contribuido por *Matthias Bussonnier* en bpo-29546 [https://bugs.python.org/issue?@action=redirect&bpo=29546].)
- Ahora se admiten las importaciones circulares que involucran importaciones absolutas con un submódulo vinculado a un nombre. (Contribuido por *Serhiy Storchaka* en bpo-30024 [https://bugs.python.org/issue?@action=redirect&bpo=30024].)
- `object.__format__(x, '')` ahora es equivalente a `str(x)` antes que `format(str(self), '')`. (Contribuido por *Serhiy Storchaka* en bpo-28974 [https://bugs.python.org/issue?@action=redirect&bpo=28974].)
- Para brindar un mejor soporte a la creación dinámica de seguimientos de pila, ahora se puede crear una instancia de **`types.TracebackType`** a partir del código Python, y el atributo **`tb_next`** en tracebacks ahora se puede escribir. (Contribución de Nathaniel J. Smith en bpo-30579 [https://bugs.python.org/issue?@action=redirect&bpo=30579]).
- Al usar la **`-m`** switch, `sys.path[0]` ahora se expande con

entusiasmo a la ruta completa del directorio de inicio, en lugar de dejarse como el directorio vacío (que permite importaciones desde el directorio de trabajo * actual * en el momento en que ocurre una importación) (Contribuido por *Nick Coghlan* en bpo-33053 [https://bugs.python.org/issue?@action=redirect&bpo=33053].)

- La nueva opción `-X importtime` o la variable de entorno `PYTHONPROFILEIMPORTTIME` pueden usarse para mostrar el tiempo de cada importación de módulos. (Contribuido por Inada Naoki en bpo-31415 [https://bugs.python.org/issue?@action=redirect&bpo=31415].)

Nuevos módulos

contextvars

El nuevo módulo `contextvars` y un conjunto de nuevas APIs C introducir soporte para *variables de contexto*. Las variables de contexto son conceptualmente similares a las variables locales del proceso. A diferencia de TLS, las variables de contexto admiten l código asincrónico correctamente.

Los módulos `asyncio` y `decimal` se han actualizado para utilizar y admitir variables de contexto listas para usar. En particular, el contexto decimal activo ahora se almacena en una variable de contexto, lo que permite que las operaciones decimales funcionen con el contexto correcto en código asincrónico.

Ver también

PEP 567 [https://peps.python.org/pep-0567/] – Variables de Contexto

PEP escrito y implementado por *Yury Selivanov*

dataclasses

El nuevo decorador `dataclass()` proporciona una forma de declarar *data classes*. Una clase de datos describe sus atributos usando anotaciones de variables de clase. Su constructor y otros métodos mágicos, como `__repr__()`, `__eq__()`, y `__hash__()` se generan automáticamente.

Ejemplo:

```
@dataclass
class Point:
    x: float
    y: float
    z: float = 0.0

p = Point(1.5, 2.5)
print(p)   # produces "Point(x=1.5, y=2.5, z=0.0)"
```

Ver también

PEP 557 [https://peps.python.org/pep-0557/] – Clases de Datos
PEP escrito y implementado por *Eric V. Smith*

importlib.resources

The new `importlib.resources` module provides several new APIs and one new ABC for access to, opening, and reading *resources* inside packages. Resources are roughly similar to files inside packages, but they needn't be actual files on the physical file system. Module loaders can provide a `get_resource_reader()` function which returns a `importlib.abc.ResourceReader` instance to support this new API. Built-in file path loaders and zip file loaders both support this.

Contribuido por *Barry Warsaw* y *Brett Cannon* en bpo-32248 [https://bugs.python.org/issue?@action=redirect&bpo=32248].

Ver también

importlib_resources [https://importlib-resources.readthedocs.io/en/latest/]: un backport de PyPI para versiones anteriores de Python.

Módulos mejorados

argparse

El nuevo método `ArgumentParser.parse_intermixed_args()` permite permite entremezclar opciones y argumentos posicionales. (Contribuido por *paul.j3* en bpo-14191 [https://bugs.python.org/issue?@action=redirect&bpo=14191].)

asyncio

El modulo `asyncio` ha recibido muchas funciones nuevas, usabilidad y mejoras de rendimiento. Los cambios notables incluyen:

- La nueva función provisional `asyncio.run()` se puede utilizar para ejecutar una corrutina desde código síncrono creando y destruyendo automáticamente el bucle de eventos. (Contribuido por *Yury Selivanov* en bpo-32314 [https://bugs.python.org/issue?@action=redirect&bpo=32314].)
- asyncio adquirió soporte para `contextvars`. `loop.call_soon()`, `loop.call_soon_threadsafe()`, `loop.call_later()`, `loop.call_at()`, y `Future.add_done_callback()` tener un nuevo parámetro opcional * de contexto * solo de palabras clave. `Tasks` ahora rastrea su contexto automáticamente. Ver **PEP 567** [https://peps.python.org/pep-0567/] para mas detalles. (Contribuido por *Yury Selivanov* en bpo-32436 [https://bugs.python.org/issue?@action=redirect&bpo=32436].)
- La nueva función `asyncio.create_task`se ha agregado como un atajo a ``asyncio.get_event_loop().create_task()`().` (Contribuido por *Andrew Svetlov* en bpo-32311 [https://bugs.python.org/issue?@action=redirect&bpo=32311].)
- El nuevo método `loop.start_tls()` puede ser usado para actualizar una conexión existente para TLS. (Contribuido por

Yury Selivanov en bpo-23749 [https://bugs.python.org/issue?@action=redirect&bpo=23749].)

- El nuevo método `loop.sock_recv_into()` permite leer datos de un socket directamente en un búfer proporcionado, lo que permite reducir las copias de datos. (Contribuido por *Antoine Pitrou* en bpo-31819 [https://bugs.python.org/issue?@action=redirect&bpo=31819].)
- La nueva función `asyncio.current_task()` devuelve la instancia `Task` que se está ejecutando actualmente y la nueva función `asyncio.all_tasks()` devuelve un conjunto de todas las instancias `Task` existentes en un bucle determinado. Los métodos **`Task.current_task()`** y **`Task.all_tasks()`** han quedado obsoletos. (Contribución de Andrew Svetlov en bpo-32250 [https://bugs.python.org/issue?@action=redirect&bpo=32250]).
- La nueva clase *provisional* `BufferedProtocol` permite implementar protocolos de transmisión con control manual sobre el búfer de recepción. (Contribuido por *Yury Selivanov* en bpo-32251 [https://bugs.python.org/issue?@action=redirect&bpo=32251].)
- La nueva función `asyncio.get_running_loop()` devuelve el bucle que se está ejecutando actualmente, y genera un `RuntimeError` si el bucle esta corriendo. Esto contrasta con `asyncio.get_event_loop()`, que * creará * un nuevo bucle de eventos si no se está ejecutando ninguno. (Contribuido por *Yury Selivanov* en bpo-32269 [https://bugs.python.org/issue?@action=redirect&bpo=32269].)
- El nuevo método de rutina `StreamWriter.wait_closed()` permite esperar hasta que se cierre el escritor de secuencias. El nuevo método `StreamWriter.is_closing()` puede ser usado para determinar si el escritor esta cerrando. (Contribuido por *Andrew Svetlov* en bpo-32391 [https://bugs.python.org/issue?@action=redirect&bpo=32391].)
- El nuevo método de rutina `loop.sock_sendfile()`

permite enviar archivos usando **`os.sendfile`** cuando sea posible. (Contribuido por *Andrew Svetlov* en bpo-32410 [https://bugs.python.org/issue?@action=redirect&bpo=32410].)

- El nuevo método **`Future.get_loop()`** y `Task.get_loop()` devuelve la instancia del bucle en el que se creó una tarea o un futuro. **`Server.get_loop()`** permite hacer lo mismo para objetos de **`asyncio.Server`**. (Contribuido por *Yury Selivanov* en bpo-32415 [https://bugs.python.org/issue?@action=redirect&bpo=32415] y *Srinivas Reddy Thatiparthy* en bpo-32418 [https://bugs.python.org/issue?@action=redirect&bpo=32418].)

- Ahora es posible controlar las instancias de **`asyncio.Server`** cuando comienzan. Anteriormente, el servidor comenzaba a funcionar inmediatamente cuando se creaba. El nuevo argumento de palabra clave *start_serving* para **`loop.create_server()`** y **`loop.create_unix_server()`**, así como **`Server.start_serving()`**, y **`Server.serve_forever()`** se puede usar para desacoplar la instanciación y el servicio de servidor. El nuevo método **`Server.is_serving()`** devuelve `True` si el servidor esta iniciando. **`Server`** los objetos ahora son administradores de contexto asíncronos:

  ```
  srv = await loop.create_server(...)

  async with srv:
      # some code

  # En éste momento, srv está cerrado y ya no acepta
  ```

  (Contribuido por *Yury Selivanov* en bpo-32662 [https://bugs.python.org/issue?@action=redirect&bpo=32662].)

- Objetos de devolución de llamada devueltos por **`loop.call_later()`** conseguir el nuevo método **`when()`** que devuelve una marca de tiempo de devolución de llamada programada absoluta. (Contribuido por *Andrew Svetlov* en bpo-32741 [https://bugs.python.org/issue?

@action=redirect&bpo=32741].)

- El método `loop.create_datagram_endpoint()` adquirió soporte para *sockets UNIX*.(Contribuido por *Quentin Dawans* en bpo-31245 [https://bugs.python.org/issue?@action=redirect&bpo=31245].)

- Los métodos `asyncio.open_connection()`, `asyncio.start_server()` functions, `loop.create_connection()`, `loop.create_server()`, ``loop.create_accepted_socket() <asyncio.loop.connect_accepted_socket>`y sus correspondientes variantes de socket UNIX ahora aceptan el argumento de palabra clave *ssl_handshake_timeout*. (Contribuido por *Neil Aspinall* en :issue:`29970()``.)

- El nuevo método `Handle.cancelled()` devuelve `True` si se canceló la devolución de llamada. (Contribuido por *Marat Sharafutdinov* en bpo-31943 [https://bugs.python.org/issue?@action=redirect&bpo=31943].)

- La fuente asyncio se ha convertido para utilizar la sintaxis `async/await`. (Contribuido por *Andrew Svetlov* en bpo-32193 [https://bugs.python.org/issue?@action=redirect&bpo=32193].)

- El nuevo método `ReadTransport.is_reading()` puede ser usado para determinar el estado de lectura del transporte. Adicionalmente, las llamadas a `ReadTransport.resume_reading()` y `ReadTransport.pause_reading()` ahora son idempotentes. (Contribuido por *Yury Selivanov* en bpo-32356 [https://bugs.python.org/issue?@action=redirect&bpo=32356].)

- Los métodos de bucle que aceptan rutas de socket ahora admiten el paso objetos con forma de ruta. (Contribuido por *Yury Selivanov* en bpo-32066 [https://bugs.python.org/issue?@action=redirect&bpo=32066].)

- En `asyncio`, los sockets TCP en Linux ahora son creados con el indicador `TCP_NODELAY` configurado por defecto. (Contribuido por *Yury Selivanov* y *Victor Stinner* en bpo-27456 [https://bugs.python.org/issue?@action=redirect&bpo=27456].)

- Las excepciones que ocurren en tareas canceladas ya no se registran. (Contribuido por *Yury Selivanov* en bpo-30508 [https://bugs.python.org/issue?@action=redirect&bpo=30508].)
- Nuevas clases `WindowsSelectorEventLoopPolicy` y `WindowsProactorEventLoopPolicy`. (Contribuido por *Yury Selivanov* en bpo-33792 [https://bugs.python.org/issue?@action=redirect&bpo=33792].)

Varias APIs `asyncio` se han obsoleta.

binascii

La función **`b2a_uu()`** ahora acepta un argumento de palabra clave opcional *backtick*. Cuando es cierto, los ceros están representados por `'`'` en lugar de espacios. (Contribuido por *Xiang Zhang* en bpo-30103 [https://bugs.python.org/issue?@action=redirect&bpo=30103].)

calendar

La clase **`(Contribuido por *Oz Tiram* en :issue:`30095`**.)

collections

`collections.namedtuple()` ahora soporta valores por defecto. (Contribuido por *Raymond Hettinger* en bpo-32320 [https://bugs.python.org/issue?@action=redirect&bpo=32320].)

compileall

`compileall.compile_dir()` aprendió el nuevo parámetro *invalidation_mode*, que se puede utilizar para habilitar invalidación de .pyc basada en hash. El modo de invalidación también se puede especificar en la línea de comando usando el nuevo argumento `--invalidation-mode`. (Contribuido por *Benjamin Peterson* en

bpo-31650 [https://bugs.python.org/issue?@action=redirect&bpo=31650].)

concurrent.futures

`ProcessPoolExecutor` y **`ThreadPoolExecutor`** ahora soporta los nuevos argumentos de constructor *initializer* y *initargs*. (Contribuido por *Antoine Pitrou* en bpo-21423 [https://bugs.python.org/issue?@action=redirect&bpo=21423].)

La **`ProcessPoolExecutor`** puede ahora tomar el contexto de multiprocesamiento a través del nuevo argumento *mp_context*. (Contribuido por *Thomas Moreau* en bpo-31540 [https://bugs.python.org/issue?@action=redirect&bpo=31540].)

contextlib

El nuevo **`nullcontext()`** es un administrador de contexto sin operaciones simple y rápido que **`ExitStack`**. (Contribuido por *Jesse-Bakker* en bpo-10049 [https://bugs.python.org/issue?@action=redirect&bpo=10049].)

Se han añadido los nuevos **`asynccontextmanager()`**, **`AbstractAsyncContextManager`**, y **`AsyncExitStack`** para complementar sus partes sincronicas. (Contribuido por *Jelle Zijlstra* en bpo-29679 [https://bugs.python.org/issue?@action=redirect&bpo=29679] y bpo-30241 [https://bugs.python.org/issue?@action=redirect&bpo=30241], y por *Alexander Mohr* y *Ilya Kulakov* en bpo-29302 [https://bugs.python.org/issue?@action=redirect&bpo=29302].)

cProfile

La linea de comandos **`cProfile`** ahora acepta `-m module_name` como alternativa a la ruta del *script*. (Contribuido por *Sanyam Khurana* en bpo-21862 [https://bugs.python.org/issue?@action=redirect&bpo=21862].)

crypt

El módulo **`crypt`** ahora admite el método de hash Blowfish. (Contribuido por Serhiy Storchaka en bpo-31664 [https://

bugs.python.org/issue?@action=redirect&bpo=31664]).

La función **`mksalt()`** ahora permite especificar el número de rondas para el hash. (Contribución de Serhiy Storchaka en bpo-31702 [https://bugs.python.org/issue?@action=redirect&bpo=31702]).

datetime

Los nuevos objetos `datetime.fromisoformat()` un método constructor `datetime` de una cadena en uno de los formatos de salida de `datetime.isoformat()`. (Contribuido por *Paul Ganssle* en bpo-15873 [https://bugs.python.org/issue?@action=redirect&bpo=15873].)

La clase `tzinfo` ahora admite compensaciones de subminutos. (Contribuido por *Alexander Belopolsky* en bpo-5288 [https://bugs.python.org/issue?@action=redirect&bpo=5288].)

dbm

`dbm.dumb` ahora admite la lectura de archivos de solo lectura y ya no escribe el archivo de índice cuándo no se cambia.

decimal

El módulo `decimal` ahora usa variables de contexto para almacenar el contexto decimal. (Contribuido por *Yury Selivanov* en bpo-32630 [https://bugs.python.org/issue?@action=redirect&bpo=32630].)

dis

La función `dis()` ahora puede desensamblar objetos de código anidados (el código de comprensiones, expresiones generadoras y funciones anidadas, y el código utilizado para construir clases anidadas). La profundidad máxima de la recursividad de desmontaje se controla mediante el nuevo parámetro * profundidad . *(Contribuido por *Serhiy Storchaka* en bpo-11822 [https://bugs.python.org/issue?@action=redirect&bpo=11822].)

distutils

`README.rst` ahora se incluye en la lista de READMEs estándar de distutils y, por lo tanto, se incluye en las distribuciones de código fuente. (Contribuido por *Ryan Gonzalez* en bpo-11913 [https://bugs.python.org/issue?@action=redirect&bpo=11913].)

enum

La **`Enum`** aprendió la nueva propiedad de clase `_ignore_`, que permite enumerar los nombres de propiedades que no deben convertirse en miembros de enumeración. (Contribuido por *Ethan Furman* en bpo-31801 [https://bugs.python.org/issue?@action=redirect&bpo=31801].)

In Python 3.8, attempting to check for non-Enum objects in **`Enum`** classes will raise a **`TypeError`** (e.g. `1 in Color`); similarly, attempting to check for non-Flag objects in a **`Flag`** member will raise **`TypeError`** (e.g. `1 in Perm.RW`); currently, both operations return **`False`** instead and are deprecated. (Contributed by Ethan Furman in bpo-33217 [https://bugs.python.org/issue?@action=redirect&bpo=33217].)

functools

`functools.singledispatch()` ahora admite el registro de implementaciones mediante anotaciones de tipo. (Contribuido por *Łukasz Langa* en bpo-32227 [https://bugs.python.org/issue?@action=redirect&bpo=32227].)

gc

La nueva función **`gc.freeze()`** permite congelar todos los objetos rastreados por el recolector de basura y excluirlos de colecciones futuras. Esto se puede utilizar antes de llamar POSIX `fork()` para hacer que la copia en escritura de GC sea amigable o para acelerar la recopilación. La nuevas funciones **`gc.unfreeze()`** revierten esta operación. Además, **`gc.get_freeze_count()`** se puede utilizar para obtener el número de objetos congelados. (Contribuido por *Li Zekun* en bpo-31558 [https://bugs.python.org/issue?@action=redirect&bpo=31558].)

hmac

El módulo `hmac` ahora tiene una función one-shot optimizada `digest()`, que es hasta tres veces más rápida que `HMAC()`.

http.client

`HTTPConnection` y `HTTPSConnection` ahora admite el nuevo argumento * tamaño de bloque * para mejorar el rendimiento de carga. (Contribuido por *Nir Soffer* en bpo-31945 [https://bugs.python.org/issue?@action=redirect&bpo=31945].)

http.server

`SimpleHTTPRequestHandler` ahora admite el encabezado HTTP `If-Modified-Since`. El servidor devuelve el estado de respuesta 304 si el archivo de destino no se modificó después del tiempo especificado en el encabezado. (Contribuido por *Pierre Quentel* en bpo-29654 [https://bugs.python.org/issue?@action=redirect&bpo=29654].)

`SimpleHTTPRequestHandler` acepta el nuevo argumento *directorio*, además del nuevo argumento de línea de comando – `directory`. Con este parámetro, el servidor sirve el directorio especificado, por defecto usa el directorio de trabajo actual. (Contribuido por *Stéphane Wirtel* y *Julien Palard* en bpo-28707 [https://bugs.python.org/issue?@action=redirect&bpo=28707].)

The new `ThreadingHTTPServer` class uses threads to handle requests using `ThreadingMixIn`. It is used when `http.server` is run with `-m`. (Contributed by Julien Palard in bpo-31639 [https://bugs.python.org/issue?@action=redirect&bpo=31639].)

idlelib and IDLE

Varias correcciones para autocompletar. (Contribuido por *Louie Lu* en bpo-15786 [https://bugs.python.org/issue?@action=redirect&bpo=15786].)

El Explorador de módulos (en el menú Archivo, antes llamado Explorador de clases), ahora muestra funciones y clases anidadas además de funciones y clases de nivel superior. (Contribuido por

Guilherme Polo, *Cheryl Sabella*, y *Terry Jan Reedy* en bpo-1612262 [https://bugs.python.org/issue?@action=redirect&bpo=1612262].)

El cuadro de diálogo Configuración (Opciones, Configurar IDLE) se ha reescrito parcialmente para mejorar tanto la apariencia como la función. (Contribuido por *Cheryl Sabella* y *Terry Jan Reedy* en múltiple issues.)

La muestra de fuente ahora incluye una selección de caracteres no latinos para que los usuarios puedan ver mejor el efecto de seleccionar una fuente en particular. (Contribuido por *Terry Jan Reedy* en bpo-13802 [https://bugs.python.org/issue?@action=redirect&bpo=13802].). La muestra se puede editar para incluir otros caracteres. (Contribuido por *Serhiy Storchaka* en bpo-31860 [https://bugs.python.org/issue?@action=redirect&bpo=31860].)

Las funciones IDLE implementadas anteriormente como extensiones se han vuelto a implementar como funciones normales. Su configuración se ha movido de la pestaña Extensiones a otras pestañas de diálogo. (Contribuido por *Charles Wohlganger* y *Terry Jan Reedy* en bpo-27099 [https://bugs.python.org/issue?@action=redirect&bpo=27099].)

Se revisó la opción de contexto del código del editor. El cuadro muestra todas las líneas de contexto hasta las líneas máximas. Al hacer clic en una línea de contexto, el editor pasa a esa línea. Los colores de contexto para temas personalizados se agregan a la pestaña Destacados del cuadro de diálogo Configuración. (Contribuido por *Cheryl Sabella* y *Terry Jan Reedy* en bpo-33642 [https://bugs.python.org/issue?@action=redirect&bpo=33642], bpo-33768 [https://bugs.python.org/issue?@action=redirect&bpo=33768], y bpo-33679 [https://bugs.python.org/issue?@action=redirect&bpo=33679].)

En Windows, una nueva llamada a la API le dice a Windows que tk escala para DPI. En Windows 8.1+ o 10, con las propiedades de compatibilidad de DPI del binario de Python sin cambios y una resolución de monitor superior a 96 DPI, esto debería hacer que el texto y las líneas sean más nítidos. De lo contrario, no debería tener ningún efecto. (Contribuido por *Terry Jan Reedy* en bpo-33656 [https://bugs.python.org/issue?@action=redirect&bpo=33656].)

Nuevo en 3.7.1:

La salida sobre N líneas (50 por defecto) se reduce a un botón. N se puede cambiar en la sección PyShell de la página General del cuadro de diálogo Configuración. Se pueden comprimir menos líneas, pero posiblemente más largas, haciendo clic derecho en la salida. La salida comprimida se puede expandir en su lugar haciendo doble clic en el botón o en el portapapeles o en una ventana separada haciendo clic derecho en el botón. (Contribuido por *Tal Einat* en bpo-1529353 [https://bugs.python.org/issue?@action=redirect&bpo=1529353].)

Los cambios anteriores se han actualizado a las versiones de mantenimiento 3.6.

Nuevo en 3.7.4:

Agregado "Run Customized" al menú Correr para ejecutar un módulo con configuraciones personalizadas. Cualquier argumento de línea de comando ingresado se agrega a sys.argv. Vuelven a aparecer en el cuadro para la siguiente ejecución personalizada. También se puede suprimir el reinicio normal del módulo principal de Shell. (Contribuidor por *Cheryl Sabella*, *Terry Jan Reedy*, y otros en bpo-5680 [https://bugs.python.org/issue?@action=redirect&bpo=5680] y bpo-37627 [https://bugs.python.org/issue?@action=redirect&bpo=37627].)

Nuevo en 3.7.5:

Agregue números de línea opcionales para las ventanas del editor IDLE. Las ventanas se abren sin números de línea a menos que se establezca lo contrario en la pestaña General del cuadro de diálogo de configuración. Los números de línea de una ventana existente se muestran y ocultan en el menú Opciones. (Contribuido por *Tal Einat* y *Saimadhav Heblikar* en bpo-17535 [https://bugs.python.org/issue?@action=redirect&bpo=17535].)

importlib

The `importlib.abc.ResourceReader` ABC was introduced to support the loading of resources from packages. See also importlib.resources. (Contributed by Barry Warsaw, Brett Cannon in

bpo-32248 [https://bugs.python.org/issue?@action=redirect&bpo=32248].)

`importlib.reload()` ahora mejora **`ModuleNotFoundError`** si el módulo carece de especificación. (Contribuido por *Garvit Khatri* en bpo-29851 [https://bugs.python.org/issue?@action=redirect&bpo=29851].)

`importlib.util.find_spec()` now raises **`ModuleNotFoundError`** instead of **`AttributeError`** if the specified parent module is not a package (i.e. lacks a `__path__` attribute). (Contributed by Milan Oberkirch in bpo-30436 [https://bugs.python.org/issue?@action=redirect&bpo=30436].)

The new **`importlib.util.source_hash()`** can be used to compute the hash of the passed source. A hash-based .pyc file embeds the value returned by this function.

io

El nuevo método **`TextIOWrapper.reconfigure()`** se puede utilizar para reconfigurar el flujo de texto con la nueva configuración. (Contribuido por *Antoine Pitrou* en bpo-30526 [https://bugs.python.org/issue?@action=redirect&bpo=30526] y *INADA Naoki* en bpo-15216 [https://bugs.python.org/issue?@action=redirect&bpo=15216].)

ipaddress

Los nuevos métodos `subnet_of()` y `supernet_of()` de **`ipaddress.IPv6Network`** y **`ipaddress.IPv4Network`** pueden ser usados para pruebas de contención de redes. (Contribuido por *Michel Albert* y *Cheryl Sabella* en bpo-20825 [https://bugs.python.org/issue?@action=redirect&bpo=20825].)

itertools

`itertools.islice()` ahora acepta **`integer-like objects`** como argumentos de inicio, parada y corte. (Contribuido por *Will Roberts* en bpo-30537 [https://bugs.python.org/issue?@action=redirect&bpo=30537].)

locale

El nuevo argumento * monetario * para **`locale.format_string()`** se puede utilizar para hacer que la conversión utilice separadores de miles monetarios y cadenas de agrupación. (Contribuido por *Garvit* en bpo-10379 [https://bugs.python.org/issue?@action=redirect&bpo=10379].)

La función **`locale.getpreferredencoding()`** ahora siempre devuelve `'UTF-8'` en Android o cuando está en el modo UTF-8 forzado.

logging

`Logger` instancias ahora pueden ser decapadas. (Contribuido por *Vinay Sajip* en bpo-30520 [https://bugs.python.org/issue?@action=redirect&bpo=30520].)

El nuevo método **`StreamHandler.setStream()`** se puede utilizar para reemplazar la secuencia del registrador después de la creación del controlador. (Contribuido por *Vinay Sajip* en bpo-30522 [https://bugs.python.org/issue?@action=redirect&bpo=30522].)

Ahora es posible especificar argumentos de palabras clave para los constructores de manejadores en la configuración pasada a **`logging.config.fileConfig()`**. (Contribuido por *Preston Landers* en bpo-31080 [https://bugs.python.org/issue?@action=redirect&bpo=31080].)

math

La nueva función **`math.remainder()`** implementa la operación de resto de estilo IEEE 754. (Contribuido por *Mark Dickinson* en bpo-29962 [https://bugs.python.org/issue?@action=redirect&bpo=29962].)

mimetypes

El tipo MIME de .bmp se ha cambiado de `'image/x-ms-bmp'` a `'image/bmp'`. (Contribuido por *Nitish Chandra* en bpo-22589 [https://bugs.python.org/issue?@action=redirect&bpo=22589].)

msilib

El nuevo método `Database.Close()` se puede utilizar para cerrar la base de datos MSI. (Contribuido por Berker Peksag en bpo-20486 [https://bugs.python.org/issue?@action=redirect&bpo=20486]).

multiprocessing

El nuevo método `Process.close()` cierra explícitamente el objeto de proceso y libera todos los recursos asociados con él. `ValueError` se genera si el proceso subyacente aún se está ejecutando. (Contribuido por *Antoine Pitrou* en bpo-30596 [https://bugs.python.org/issue?@action=redirect&bpo=30596].)

The new `Process.kill()` method can be used to terminate the process using the `SIGKILL` signal on Unix. (Contributed by Vitor Pereira in bpo-30794 [https://bugs.python.org/issue?@action=redirect&bpo=30794].)

Subprocesos no demoníacos creados por `Process` ahora se unen al salir del proceso. (Contribuido por *Antoine Pitrou* en bpo-18966 [https://bugs.python.org/issue?@action=redirect&bpo=18966].)

os

`os.fwalk()` ahora acepta el argumento *path* como `bytes`. (Contribuido por *Serhiy Storchaka* en bpo-28682 [https://bugs.python.org/issue?@action=redirect&bpo=28682].)

`os.scandir()` obtuvo soporte de descriptores de archivo. (Contribuido por *Serhiy Storchaka* en bpo-25996 [https://bugs.python.org/issue?@action=redirect&bpo=25996].)

La nueva función `register_at_fork()` permite registrar devoluciones de llamada de Python para que se ejecuten en la bifurcación del proceso. (Contribuido por *Antoine Pitrou* en bpo-16500 [https://bugs.python.org/issue?@action=redirect&bpo=16500].)

Agrega funciones `os.preadv()` (combinando la funcionalidad de `os.readv()` y `os.pread()`) y `os.pwritev()` (combinando la funcionalidad de `os.writev()` y `os.pwrite()`). (Contribuido por *Pablo Galindo* en bpo-31368 [https://bugs.python.org/issue?@action=redirect&bpo=31368].)

El argumento de modo de **`os.makedirs()`** ya no afecta los bits de permiso de archivo de los directorios de nivel intermedio recién creados. (Aportado por Serhiy Storchaka en bpo-19930 [https://bugs.python.org/issue?@action=redirect&bpo=19930].)

`os.dup2()` ahora retorna el nuevo descriptor de archivo. Previamente, `None` siempre fue retornado. (Contribuido por *Benjamin Peterson* en bpo-32441 [https://bugs.python.org/issue?@action=redirect&bpo=32441].)

La estructura retornada por **`os.stat()`** ahora contiene el atributo **`st_fstype`** en Solaris y sus derivados. (Contribuido por *Jesús Cea Avión* en bpo-32659 [https://bugs.python.org/issue?@action=redirect&bpo=32659].)

pathlib

El nuevo método **`Path.is_mount()`** ahora está disponible en los sistemas POSIX y se puede utilizar para determinar si una ruta es un punto de montaje. (Contribuido por *Cooper Ry Lees* en bpo-30897 [https://bugs.python.org/issue?@action=redirect&bpo=30897].)

pdb

`pdb.set_trace()` ahora toma un argumento opcional *header* solo de palabra clave. Si se proporciona, se imprime en la consola justo antes de que comience la depuración. (Contribuido por *Barry Warsaw* en bpo-31389 [https://bugs.python.org/issue?@action=redirect&bpo=31389].)

`pdb` la línea de comando ahora acepta `-m module_name` como una alternativa al archivo de script. (Contribuido por *Mario Corchero* en bpo-32206 [https://bugs.python.org/issue?@action=redirect&bpo=32206].)

py_compile

`py_compile.compile()` — y por extensión, **`compileall`** — ahora respeta la variable de entorno **`SOURCE_DATE_EPOCH`** creando incondicionalmente archivos `.pyc` para la validación basada en hash. Esto permite garantizar compilaciones reproducibles [https://reproducible-builds.org/] de archivos `.pyc` cuando se crean con entusiasmo. (Contribuido por *Bernhard M. Wiedemann* en bpo-29708 [https://bugs.python.org/issue?@action=redirect&bpo=29708].)

pydoc

El servidor *pydoc* ahora puede vincularse a un nombre de host arbitrario especificado por el nuevo argumento de línea de comandos `-n`. (Contribuido por *Feanil Patel* en bpo-31128 [https://bugs.python.org/issue?@action=redirect&bpo=31128].)

queue

La nueva clase **`SimpleQueue`** es una cola ilimitada FIFO. (Contribuido por *Antoine Pitrou* en bpo-14976 [https://bugs.python.org/issue?@action=redirect&bpo=14976].)

re

Las banderas **`re.ASCII`**, **`re.LOCALE`** y **`re.UNICODE`** se pueden establecer dentro del alcance de un grupo. (Contribuido por *Serhiy Storchaka* en bpo-31690 [https://bugs.python.org/issue?@action=redirect&bpo=31690].)

`re.split()` ahora admite la división en un patrón como `r'\b'`, `'^$'` or `(?=-)` que coincide con una cadena vacía. (Contribuido por *Serhiy Storchaka* en bpo-25054 [https://bugs.python.org/issue?@action=redirect&bpo=25054].)

Las expresiones regulares compiladas con el indicador **`re.LOCALE`** ya no dependen de la configuración regional en el momento de la compilación. La configuración regional se aplica solo cuando se utiliza la expresión regular compilada. (Contribuido por *Serhiy Storchaka* en bpo-30215 [https://bugs.python.org/issue?@action=redirect&bpo=30215].)

`FutureWarning` ahora se emite si una expresión regular contiene construcciones de conjuntos de caracteres que cambiarán semánticamente en el futuro, como conjuntos anidados y operaciones de conjuntos. (Contribuido por *Serhiy Storchaka* en bpo-30349 [https://bugs.python.org/issue?@action=redirect&bpo=30349].)

Las expresiones regulares compiladas y los objetos coincidentes ahora se pueden copiar usando `copy.copy()` y `copy.deepcopy()`. (Contribuido por *Serhiy Storchaka* en bpo-10076 [https://bugs.python.org/issue?@action=redirect&bpo=10076].)

signal

El nuevo argumento * warn_on_full_buffer * para la función `signal.set_wakeup_fd()` permite especificar si Python imprime una advertencia en *stderr* cuando el búfer de activación se desborda. (Contribuido por *Nathaniel J. Smith* en bpo-30050 [https://bugs.python.org/issue?@action=redirect&bpo=30050].)

socket

El nuevo método `socket.getblocking()` retorna `True` si el socket esta en modo bloqueo y `False` en caso contrario. (Contribuido por *Yury Selivanov* en bpo-32373 [https://bugs.python.org/issue?@action=redirect&bpo=32373].)

La nueva función `socket.close()` cierra el descriptor de archivo de socket pasado. Esta función debe usarse en lugar de `os.close()` para una mejor compatibilidad entre plataformas. (Contribuido por *Christian Heimes* en bpo-32454 [https://bugs.python.org/issue?@action=redirect&bpo=32454].)

The `socket` module now exposes the socket.TCP_CONGESTION (Linux 2.6.13), socket.TCP_USER_TIMEOUT (Linux 2.6.37), and socket.TCP_NOTSENT_LOWAT (Linux 3.12) constants. (Contributed by Omar Sandoval in bpo-26273 [https://bugs.python.org/issue?@action=redirect&bpo=26273] and Nathaniel J. Smith in bpo-29728 [https://bugs.python.org/issue?@action=redirect&bpo=29728].)

Se agregó compatibilidad con sockets `socket.AF_VSOCK` para permitir la comunicación entre máquinas virtuales y sus hosts. (Contribución de Cathy Avery en bpo-27584 [https://bugs.python.org/issue?@action=redirect&bpo=27584]).

Los sockets ahora detectan automáticamente la familia, el tipo y el protocolo del descriptor de archivo de forma predeterminada. (Contribuido por *Christian Heimes* en bpo-28134 [https://bugs.python.org/issue?@action=redirect&bpo=28134].)

socketserver

`socketserver.ThreadingMixIn.server_close` now waits until all non-daemon threads complete. **`socketserver.ForkingMixIn.server_close`** now waits until all child processes complete.

Add a new `socketserver.ForkingMixIn.block_on_close` class attribute to `socketserver.ForkingMixIn` and `socketserver.ThreadingMixIn` classes. Set the class attribute to `False` to get the pre-3.7 behaviour.

sqlite3

`sqlite3.Connection` ahora expone el método `backup()` cuando la biblioteca SQLite subyacente está en la versión 3.6.11 o superior. (Contribuido por *Lele Gaifax* en bpo-27645 [https://bugs.python.org/issue?@action=redirect&bpo=27645].)

El argumento * base de datos * de `sqlite3.connect()` ahora acepta cualquier path-like object, en lugar de solo una cadena. (Contribuido por *Anders Lorentsen* en bpo-31843 [https://bugs.python.org/issue?@action=redirect&bpo=31843].)

ssl

The `ssl` module now uses OpenSSL's builtin API instead of **`match_hostname()`** to check a host name or an IP address. Values are validated during TLS handshake. Any certificate validation error including failing the host name check now raises `SSLCertVerificationError` and aborts the handshake with a proper TLS Alert message. The new exception contains additional information. Host name validation can be customized with `SSLContext.hostname_checks_common_name`. (Contributed by Christian Heimes in bpo-31399 [https://bugs.python.org/issue?

@action=redirect&bpo=31399].)

Nota

La verificación mejorada del nombre de host requiere una implementación *libssl* compatible con OpenSSL 1.0.2 o 1.1. En consecuencia, OpenSSL 0.9.8 y 1.0.1 ya no son compatibles (consulte Eliminación de soporte de plataforma para más detalles). El módulo ssl es principalmente compatible con LibreSSL 2.7.2 y versiones posteriores.

El módulo `ssl` ya no envía direcciones IP en la extensión SNI TLS. (Contribuido por *Christian Heimes* en bpo-32185 [https://bugs.python.org/issue?@action=redirect&bpo=32185].)

`match_hostname()` no longer supports partial wildcards like `www*.example.org`. (Contributed by Mandeep Singh in bpo-23033 [https://bugs.python.org/issue?@action=redirect&bpo=23033] and Christian Heimes in bpo-31399 [https://bugs.python.org/issue?@action=redirect&bpo=31399].)

La selección del conjunto de cifrado predeterminado del módulo `ssl` ahora usa un enfoque de lista negra en lugar de una lista blanca codificada. Python ya no vuelve a habilitar los cifrados que han sido bloqueados por las actualizaciones de seguridad de OpenSSL. La selección del conjunto de cifrado predeterminado se puede configurar en tiempo de compilación. (Contribuido por *Christian Heimes* en bpo-31429 [https://bugs.python.org/issue?@action=redirect&bpo=31429].)

Ahora se admite la validación de certificados de servidor que contienen nombres de dominio internacionalizados (IDN). Como parte de este cambio, el atributo **`SSLSocket.server_hostname`** ahora almacena el nombre de host esperado en forma de etiqueta A ("`xn--pythn-mua.org`"), en lugar de la forma de etiqueta U ("`pythön.org`"). (Contribuido por *Nathaniel J. Smith* y *Christian Heimes* en bpo-28414 [https://bugs.python.org/issue?@action=redirect&bpo=28414].)

El módulo `ssl` tiene soporte preliminar y experimental para TLS

1.3 y OpenSSL 1.1.1. En el momento del lanzamiento de Python 3.7.0, OpenSSL 1.1.1 todavía está en desarrollo y TLS 1.3 aún no se ha finalizado. El protocolo y el protocolo de enlace TLS 1.3 se comportan de forma ligeramente diferente a TLS 1.2 y versiones anteriores, consulte TLS 1.3. (Contribuido por *Christian Heimes* en bpo-32947 [https://bugs.python.org/issue?@action=redirect&bpo=32947], bpo-20995 [https://bugs.python.org/issue?@action=redirect&bpo=20995], bpo-29136 [https://bugs.python.org/issue?@action=redirect&bpo=29136], bpo-30622 [https://bugs.python.org/issue?@action=redirect&bpo=30622] y bpo-33618 [https://bugs.python.org/issue?@action=redirect&bpo=33618])

`SSLSocket` y `SSLObject` ya no tiene un constructor público. La creación de instancias directa nunca fue una característica documentada y compatible. Las instancias deben crearse con métodos `SSLContext` `wrap_socket()` y `wrap_bio()`. (Contribuido por *Christian Heimes* en bpo-32951 [https://bugs.python.org/issue?@action=redirect&bpo=32951])

Las API de OpenSSL 1.1 para configurar la versión mínima y máxima del protocolo TLS están disponibles como `SSLContext.minimum_version` y `SSLContext.maximum_version`. Los protocolos admitidos se indican mediante varios indicadores nuevos, como `HAS_TLSv1_1`. (Contribuido por *Christian Heimes* en bpo-32609 [https://bugs.python.org/issue?@action=redirect&bpo=32609].)

Se agregó `ssl.SSLContext.post_handshake_auth` para habilitar y `ssl.SSLSocket.verify_client_post_handshake()` para iniciar la autenticación posterior al protocolo de enlace TLS 1.3. (Contribuido por Christian Heimes en gh-78851 [https://github.com/python/cpython/issues/78851].)

string

`string.Template` ahora le permite modificar opcionalmente el patrón de expresión regular para marcadores de posición reforzados y marcadores de posición no reforzados por separado. (Contribuido por *Barry Warsaw* en bpo-1198569 [https://bugs.python.org/issue?@action=redirect&bpo=1198569].)

subprocess

La función **`subprocess.run()`** acepta el nuevo argumento de palabra clave *capture_output*. Cuando es verdadero, se capturarán stdout y stderr. Esto es equivalente a pasar **`subprocess.PIPE`** como argumentos *stdout* y *stderr*. (Contribuido por Bo Bayles en bpo-32102 [https://bugs.python.org/issue?@action=redirect&bpo=32102]).

La función `subprocess.run` y el constructor **`subprocess.Popen`** ahora aceptan la palabra clave del argumento *text* como un alias para *universal_newlines*. (Contribuido por *Andrew Clegg* en bpo-31756 [https://bugs.python.org/issue?@action=redirect&bpo=31756].)

En Windows, el valor predeterminado de *close_fds* se cambió de `False` a `True` al redirigir los identificadores estándar. Ahora es posible establecer *close_fds* en verdadero al redirigir los identificadores estándar. Ver **`subprocess.Popen`**. Esto significa que *close_fds* ahora tiene el valor predeterminado de `True` en todas las plataformas compatibles. (Contribuido por *Segev Finer* en bpo-19764 [https://bugs.python.org/issue?@action=redirect&bpo=19764].)

El módulo de subproceso ahora es más elegante al manejar **`KeyboardInterrupt`** durante: **`subprocess.call()`**, **`subprocess.run()`**, o en un administrador de contexto **`Popen`**. Ahora espera un corto período de tiempo para que el niño salga, antes de continuar con el manejo de la excepción `KeyboardInterrupt`. (Contribuido por *Gregory P. Smith* en bpo-25942 [https://bugs.python.org/issue?@action=redirect&bpo=25942].)

sys

La nueva función de enganche **`sys.breakpointhook()`** es llamada por la incorporada **`breakpoint()`**. (Contribuido por *Barry Warsaw* en bpo-31353 [https://bugs.python.org/issue?@action=redirect&bpo=31353].)

En Android, la nueva **`sys.getandroidapilevel()`** devuelve la versión de la API de Android en tiempo de compilación. (Contribuido por *Victor Stinner* en bpo-28740 [https://bugs.python.org/

issue?@action=redirect&bpo=28740].)

The new `sys.get_coroutine_origin_tracking_depth()` function returns the current coroutine origin tracking depth, as set by the new `sys.set_coroutine_origin_tracking_depth()`. `asyncio` has been converted to use this new API instead of the deprecated **`sys.set_coroutine_wrapper()`**. (Contributed by Nathaniel J. Smith in bpo-32591 [https://bugs.python.org/issue?@action=redirect&bpo=32591].)

time

PEP 564 [https://peps.python.org/pep-0564/] añade seis nuevas funciones con resolución de nanosegundos al modulo `time`:

- `time.clock_gettime_ns()`
- `time.clock_settime_ns()`
- `time.monotonic_ns()`
- `time.perf_counter_ns()`
- `time.process_time_ns()`
- `time.time_ns()`

Se han agregado nuevos identificadores de reloj:

- `time.CLOCK_BOOTTIME` (Linux): idéntico a `time.CLOCK_MONOTONIC`, excepto que también incluye cualquier momento en que el sistema esté suspendido.
- `time.CLOCK_PROF` (FreeBSD, NetBSD y OpenBSD): Temporizador de CPU por proceso de alta resolución.
- `time.CLOCK_UPTIME` (FreeBSD, OpenBSD): Tiempo cuyo valor absoluto es el tiempo que el sistema ha estado funcionando y no suspendido, proporcionando una medición precisa del tiempo de actividad.

La nuevas funciones `time.thread_time()` y `time.thread_time_ns()` se puede utilizar para obtener medidas de tiempo de CPU por subproceso. (Contribuido por *Antoine Pitrou* en bpo-32025 [https://bugs.python.org/issue?@action=redirect&bpo=32025].)

La nueva función `time.pthread_getcpuclockid()` devuelve el

ID de reloj del reloj de tiempo de CPU específico del subproceso.

tkinter

La nueva clase **`tkinter.ttk.Spinbox`** esta disponible ahora. (Contribuido por *Alan Moore* en bpo-32585 [https://bugs.python.org/issue?@action=redirect&bpo=32585].)

tracemalloc

`tracemalloc.Traceback` se comporta más como rastreos regulares, ordenando los fotogramas del más antiguo al más reciente. **`Traceback.format()`** ahora acepta *límite* negativo, truncando el resultado a los marcos más antiguos de `abs(limit)`. Para obtener el comportamiento anterior, use el nuevo argumento *most_recent_first* para `Traceback.format()`. (Contribuido por *Jesse Bakker* en bpo-32121 [https://bugs.python.org/issue?@action=redirect&bpo=32121].)

types

Las nuevas clases **`WrapperDescriptorType`**, **`MethodWrapperType`**, **`MethodDescriptorType`**, y **`ClassMethodDescriptorType`** ahora están disponibles. (Contribuido por *Manuel Krebber* y *Guido van Rossum* en bpo-29377 [https://bugs.python.org/issue?@action=redirect&bpo=29377], y *Serhiy Storchaka* en bpo-32265 [https://bugs.python.org/issue?@action=redirect&bpo=32265].)

La nueva función **`types.resolve_bases()`** resuelve las entradas MRO dinámicamente según lo especificado por **PEP 560** [https://peps.python.org/pep-0560/]. (Contribuido por *Ivan Levkivskyi* en bpo-32717 [https://bugs.python.org/issue?@action=redirect&bpo=32717].)

unicodedata

La base de datos interna **`unicodedata`** se actualizó para usar Unicode 11 [https://www.unicode.org/versions/Unicode11.0.0/]. (Aportado por Benjamín Peterson.)

unittest

La nueva opción de la linea de comandos `-k` permite filtrar pruebas por una subcadena de nombre o un patrón similar a un shell de Unix. Por ejemplo, `python -m unittest -k foo` ejecuta `foo_tests.SomeTest.test_something`, `bar_tests.SomeTest.test_foo`, pero no `bar_tests.FooTest.test_something`. (Contribuido por *Jonas Haag* en bpo-32071 [https://bugs.python.org/issue?@action=redirect&bpo=32071].)

unittest.mock

Los atributos **`sentinel`** ahora preservan su identidad cuando están **`copied`** o **`pickled`**. (Contribuido por *Serhiy Storchaka* en bpo-20804 [https://bugs.python.org/issue?@action=redirect&bpo=20804].)

La nueva función **`seal()`** permite sellar instancias **`Mock`**, que no permitirá la creación de simulacros de atributos. El sello se aplica de forma recursiva a todos los atributos que en sí mismos son burlas. (Contribuido por *Mario Corchero* en bpo-30541 [https://bugs.python.org/issue?@action=redirect&bpo=30541].)

urllib.parse

`urllib.parse.quote()` ha sido actualizada desde **RFC 2396** [https://datatracker.ietf.org/doc/html/rfc2396.html] a **RFC 3986** [https://datatracker.ietf.org/doc/html/rfc3986.html], agregado `~` para el conjunto de caracteres que nunca se cotizan por defecto. (Contribuido por *Christian Theune* y *Ratnadeep Debnath* en bpo-16285 [https://bugs.python.org/issue?@action=redirect&bpo=16285].)

uu

La función **`uu.encode()`** ahora acepta un argumento de palabra clave *backtick* opcional. Cuando es verdadero, los ceros se representan con **`'`'`** en lugar de espacios. (Contribuido por Xiang Zhang en bpo-30103 [https://bugs.python.org/issue?@action=redirect&bpo=30103]).

uuid

El nuevo atributo `UUID.is_safe` transmite información de la plataforma sobre si los UUID generados se generan con un método seguro para multiprocesamiento. (Contribuido por *Barry Warsaw* en bpo-22807 [https://bugs.python.org/issue?@action=redirect&bpo=22807].)

`uuid.getnode()` ahora prefiere las direcciones MAC administradas universalmente a las direcciones MAC administradas localmente. Esto ofrece una mejor garantía de la unicidad global de los UUID devueltos desde `uuid.uuid1()`. Si solo hay disponibles direcciones MAC administradas localmente, se devuelve la primera que se encuentre. (Contribuido por *Barry Warsaw* en bpo-32107 [https://bugs.python.org/issue?@action=redirect&bpo=32107].)

warnings

La inicialización de los filtros de advertencias predeterminados ha cambiado de la siguiente manera:

- advertencias habilitadas a través de opciones de línea de comando (incluidos los de `-b` y la nueva opción específica de CPython `-X dev`) siempre se pasan a la maquinaria de advertencias a través del atributo `sys.warnoptions`.
- los filtros de advertencias habilitados a través de la línea de comandos o el entorno ahora tienen el siguiente orden de prioridad:
    - ○ el filtro `BytesWarning` para `-b` (o `-bb`)
    - ○ cualquier filtro especificado con la opción `-W`
    - ○ cualquier filtro especificado con la variable de entorno `PYTHONWARNINGS`
    - ○ cualquier otro filtro específico de CPython (por ejemplo, el filtro `default` agregado para el nuevo modo `-X dev`)
    - ○ cualquier filtro implícito definido directamente por la maquinaria de advertencias
- en las compilaciones de depuración de CPython, ahora todas las advertencias se muestran de forma predeterminada (la lista de filtros implícitos está vacía)

(Contribuido por *Nick Coghlan* y *Victor Stinner* en bpo-20361 [https://bugs.python.org/issue?@action=redirect&bpo=20361], bpo-32043 [https://bugs.python.org/issue?@action=redirect&bpo=32043], and bpo-32230 [https://bugs.python.org/issue?@action=redirect&bpo=32230].)

Las advertencias de obsolescencia se muestran una vez más de forma predeterminada en scripts de un solo archivo y en el indicador interactivo. Consultar PEP 565: Mostrar DeprecationWarning en __main__ para mas detalles. (Contribuido por *Nick Coghlan* en bpo-31975 [https://bugs.python.org/issue?@action=redirect&bpo=31975].)

xml

Como medida de mitigación frente a la recuperación de DTD y entidades externas, los módulos `xml.dom.minidom` y `xml.sax` ya no procesan entidades externas de forma predeterminada. (Contribuido por Christian Heimes en gh-61441 [https://github.com/python/cpython/issues/61441].)

xml.etree

ElementPath predicates in the `find()` methods can now compare text of the current node with `[. = "text"]`, not only text in children. Predicates also allow adding spaces for better readability. (Contributed by Stefan Behnel in bpo-31648 [https://bugs.python.org/issue?@action=redirect&bpo=31648].)

xmlrpc.server

`SimpleXMLRPCDispatcher.register_function()` can now be used as a decorator. (Contributed by Xiang Zhang in bpo-7769 [https://bugs.python.org/issue?@action=redirect&bpo=7769].)

zipapp

Función `create_archive()` ahora acepta un argumento opcional *filter* para permitir al usuario seleccionar que archivos deben incluirse en el

Función `create_archive()` ahora acepta un argumento opcional * comprimido * para generar un archivo comprimido. También se ha agregado una opción de línea de comando `—compress` para admitir la compresión. (Contribuido por *Zhiming Wang* en bpo-31638 [https://bugs.python.org/issue?@action=redirect&bpo=31638].)

zipfile

`ZipFile` ahora acepta el nuevo parámetro *compresslevel* para controlar el nivel de compresión. (Contribuido por *Bo Bayles* en bpo-21417 [https://bugs.python.org/issue?@action=redirect&bpo=21417].)

Subdirectorios en archivos creados por `ZipFile` ahora son guardados en orden alfabético. (Contribuido por *Bernhard M. Wiedemann* en bpo-30693 [https://bugs.python.org/issue?@action=redirect&bpo=30693].)

Cambios en la API C

Se implementó una nueva API para el almacenamiento local de subprocesos. Ver PEP 539: Nueva API C para almacenamiento local de subprocesos para una descripción general y Thread-specific storage API para una referencia completa. (Contribuido por *Masayuki Yamamoto* en bpo-25658 [https://bugs.python.org/issue?@action=redirect&bpo=25658].)

La nueva funcionalidad variables de contexto expone un numero de nuevas APIs C.

La nueva función **`PyImport_GetModule()`** devuelve el módulo previamente importado con el nombre dado. (Contribuido por *Eric Snow* en bpo-28411 [https://bugs.python.org/issue?@action=redirect&bpo=28411].)

El nuevo macro **`Py_RETURN_RICHCOMPARE`** facilita la escritura de funciones de comparación enriquecidas. (Contribuido por *Petr Victorin* en bpo-23699 [https://bugs.python.org/issue?@action=redirect&bpo=23699].)

El nuevo macro **`Py_UNREACHABLE`** se puede utilizar para marcar rutas de código inalcanzables. (Contribuido por *Barry Warsaw* en bpo-31338 [https://bugs.python.org/issue?@action=redirect&bpo=31338].)

El **`tracemalloc`** ahora expone una API C a través de las nuevas funciones **`PyTraceMalloc_Track()`** y **`PyTraceMalloc_Untrack()`**. (Contribuido por *Victor Stinner* en bpo-30054 [https://bugs.python.org/issue?@action=redirect&bpo=30054].)

The new import_find_load_start and import_find_load_done static markers can be used to trace module imports. (Contributed by Christian Heimes in bpo-31574 [https://bugs.python.org/issue?@action=redirect&bpo=31574].)

The fields **`name`** and **`doc`** of structures **`PyMemberDef`**, **`PyGetSetDef`**, **`PyStructSequence_Field`**, **`PyStructSequence_Desc`**, and **`wrapperbase`** are now of type `const char *` rather of `char *`. (Contributed by Serhiy Storchaka in bpo-28761 [https://bugs.python.org/issue?@action=redirect&bpo=28761].)

El resultado de **`PyUnicode_AsUTF8AndSize()`** y **`PyUnicode_AsUTF8()`** es ahora de tipo `const char *` en lugar de `char *`. (Contribuido por *Serhiy Storchaka* en bpo-28769 [https://bugs.python.org/issue?@action=redirect&bpo=28769].)

El resultado de **`PyMapping_Keys()`**, **`PyMapping_Values()`** y **`PyMapping_Items()`** ahora es siempre una lista, en lugar de una lista o una tupla. (Contribuido por *Oren Milman* en bpo-28280 [https://bugs.python.org/issue?@action=redirect&bpo=28280].)

Funciones agregadas **`PySlice_Unpack()`** y **`PySlice_AdjustIndices()`**. (Contribuido por *Serhiy Storchaka* en bpo-27867 [https://bugs.python.org/issue?@action=redirect&bpo=27867].)

`PyOS_AfterFork()` está en desuso en favor de las nuevas funciones **`PyOS_BeforeFork()`**, **`PyOS_AfterFork_Parent()`** y **`PyOS_AfterFork_Child()`**. (Contribuido por *Antoine Pitrou* en bpo-16500 [https://bugs.python.org/issue?@action=redirect&bpo=16500].)

El único `PyExc_RecursionErrorInst` que formaba parte de la API pública se ha eliminado ya que sus miembros nunca borrados pueden causar un error de segmentación durante la finalización del intérprete. Contribuido por *Xavier de Gaye* en bpo-22898 [https://bugs.python.org/issue?@action=redirect&bpo=22898] y bpo-30697 [https://bugs.python.org/issue?@action=redirect&bpo=30697].

Se agregó compatibilidad con C API para zonas horarias con constructores de zonas horarias `PyTimeZone_FromOffset()` and `PyTimeZone_FromOffsetAndName()`, y acceso único al UTC con `PyDateTime_TimeZone_UTC`. Contribuido por *Paul Ganssle* en bpo-10381 [https://bugs.python.org/issue?@action=redirect&bpo=10381].

The type of results of `PyThread_start_new_thread()` and `PyThread_get_thread_ident()`, and the *id* parameter of `PyThreadState_SetAsyncExc()` changed from long to unsigned long. (Contributed by Serhiy Storchaka in bpo-6532 [https://bugs.python.org/issue?@action=redirect&bpo=6532].)

`PyUnicode_AsWideCharString()` ahora genera un `ValueError` si el segundo argumento es `NULL` y la cadena wchar_t* contiene caracteres nulos. (Aportado por Serhiy Storchaka en bpo-30708 [https://bugs.python.org/issue?@action=redirect&bpo=30708].)

Los cambios en la secuencia de inicio y la gestión de los asignadores de memoria dinámica significan que el requisito documentado durante mucho tiempo de llamar `Py_Initialize()` antes de llamar a la mayoría de las funciones de la API de C, ahora se confía más en él, y no cumplirlo puede provocar errores de segmentación en las aplicaciones integradas. Ver la sección Portando a Python 3.7 en este documento y en la sección Before Python initialization en la documentación de la API de C para obtener más detalles.

El nuevo: c: func `PyInterpreterState_GetID()` devuelve él ID único para un intérprete dado. (Contribuido por *Eric Snow* en bpo-29102 [https://bugs.python.org/issue?@action=redirect&bpo=29102].)

`Py_DecodeLocale()`, `Py_EncodeLocale()` ahora usa la codificación UTF-8 cuando el modo UTF-8 esta habilitado. (Contribuido por *Victor Stinner* en bpo-29240 [https://bugs.python.org/issue?@action=redirect&bpo=29240].)

`PyUnicode_DecodeLocaleAndSize()` y

`PyUnicode_EncodeLocale()` ahora usan la codificación local actual para el controlador de errores `surrogateescape`. (Contribuido por *Victor Stinner* en bpo-29240 [https://bugs.python.org/issue?@action=redirect&bpo=29240].)

Los parámetros *start* y *end* de **`PyUnicode_FindChar()`** ahora se ajustan para comportarse como cortes de cuerda. (Contribuido por *Xiang Zhang* en bpo-28822 [https://bugs.python.org/issue?@action=redirect&bpo=28822].)

Construir cambios

Se ha eliminado el soporte para construir `—without-threads`. El módulo **`threading`** ahora está siempre disponible. (Contribuido por *Antoine Pitrou* en bpo-31370 [https://bugs.python.org/issue?@action=redirect&bpo=31370].).

Ya no se incluye una copia completa de *libffi* para su uso al compilar el módulo **`_ctypes`** en plataformas que no son OSX UNIX. Ahora se requiere una copia instalada de *libffi* al construir `_ctypes` en tales plataformas. (Contribuido por *Zachary Ware* en bpo-27979 [https://bugs.python.org/issue?@action=redirect&bpo=27979].)

El proceso de compilación de Windows ya no depende de una sub versión para extraer fuentes externas; en su lugar, se usa un script de Python para descargar archivos zip desde GitHub. Si no se encuentra Python 3.6 en el sistema (a través de `py -3.6`), *NuGet* se usa para descargar una copia de Python de 32 bits para este propósito. (Contribuido por *Zachary Ware* en bpo-30450 [https://bugs.python.org/issue?@action=redirect&bpo=30450].)

El módulo **`ssl`** requiere *libssl* compatible con OpenSSL 1.0.2 o 1.1. OpenSSL 1.0.1 llegó al final de su vida útil el 31 de diciembre de 2016 y ya no es compatible. LibreSSL tampoco es compatible temporalmente. Las versiones de LibreSSL hasta la versión 2.6.4 carecen de las API de OpenSSL 1.0.2 necesarias.

Optimizaciones

La sobrecarga de llamar a muchos métodos de varias clases de biblioteca estándar implementadas en C se ha reducido significativamente al portar más código para usar la convención `METH_FASTCALL`. (Contribuido por *Victor Stinner* en bpo-29300 [https://bugs.python.org/issue?@action=redirect&bpo=29300], bpo-29507 [https://bugs.python.org/issue?@action=redirect&bpo=29507], bpo-29452 [https://bugs.python.org/issue?@action=redirect&bpo=29452], y bpo-29286 [https://bugs.python.org/issue?@action=redirect&bpo=29286].)

Varias optimizaciones han reducido el tiempo de inicio de Python en un 10% en Linux y hasta en un 30% en macOS. (Contribuido por *Victor Stinner*, *INADA Naoki* en bpo-29585 [https://bugs.python.org/issue?@action=redirect&bpo=29585], y *Ivan Levkivskyi* en bpo-31333 [https://bugs.python.org/issue?@action=redirect&bpo=31333].)

Las llamadas a métodos ahora son hasta un 20% más rápidas debido a los cambios de código de bytes que evitan la creación de instancias de métodos enlazados. (Contribuidos por *Yury Selivanov* y *INADA Naoki* en bpo-26110 [https://bugs.python.org/issue?@action=redirect&bpo=26110].)

El módulo **`asyncio`** recibió una serie de optimizaciones notables para funciones de uso común:

- La función **`asyncio.get_event_loop()`** se ha vuelto a implementar en C para hacerlo hasta 15 veces más rápido. (Contribuido por *Yury Selivanov* en bpo-32296 [https://bugs.python.org/issue?@action=redirect&bpo=32296].)
- Se ha optimizado la gestión de devolución de llamada **`asyncio.Future`**. (Contribuido por *Yury Selivanov* en bpo-32348 [https://bugs.python.org/issue?@action=redirect&bpo=32348].)
- **`asyncio.gather()`** ahora es hasta un 15% más rápido. (Contribuido por *Yury Selivanov* en bpo-32355 [https://bugs.python.org/issue?@action=redirect&bpo=32355].)
- **`asyncio.sleep()`** ahora es hasta 2 veces más rápido cuando el argumento *delay* es cero o negativo. (Contribuido por *Andrew Svetlov* en bpo-32351 [https://bugs.python.org/issue?@action=redirect&bpo=32351].)
- Se ha reducido la sobrecarga de rendimiento del modo de depuración asyncio. (Contribuido por *Antoine Pitrou* en

bpo-31970 [https://bugs.python.org/issue?@action=redirect&bpo=31970].)

Como resultado de PEP 560 trabajo, el tiempo de importación de `typing` se ha reducido en un factor de 7, y muchas operaciones de mecanografía ahora son más rápidas. (Contribuido por *Ivan Levkivskyi* en bpo-32226 [https://bugs.python.org/issue?@action=redirect&bpo=32226].)

`sorted()` y `list.sort()` se han optimizado para que los casos comunes sean hasta un 40-75% más rápidos. (Contribuido por *Elliot Gorokhovsky* en bpo-28685 [https://bugs.python.org/issue?@action=redirect&bpo=28685].)

`dict.copy()` es ahora 5.5 veces más rápido. (Contribuido por *Yury Selivanov* en bpo-31179 [https://bugs.python.org/issue?@action=redirect&bpo=31179].)

`hasattr()` y `getattr()` ahora son aproximadamente 4 veces más rápidos cuando *name* no se encuentra y *obj* no se anula `object.__getattr__()` o `object.__getattribute__()`. (Contribuido por *INADA Naoki* en bpo-32544 [https://bugs.python.org/issue?@action=redirect&bpo=32544].)

La búsqueda de ciertos caracteres Unicode (como la mayúscula ucraniana "Є") en una cadena fue hasta 25 veces más lenta que la búsqueda de otros caracteres. Ahora es solo 3 veces más lento en el peor de los casos. (Contribuido por *Serhiy Storchaka* en bpo-24821 [https://bugs.python.org/issue?@action=redirect&bpo=24821].)

La fábrica `collections.namedtuple()` ha sido re-implementada para hacer que la creación de tuplas con nombre sea de 4 a 6 veces más rápida. (Contribuido por *Jelle Zijlstra* con nuevas mejoras de *INADA Naoki*, *Serhiy Storchaka*, y *Raymond Hettinger* en bpo-28638 [https://bugs.python.org/issue?@action=redirect&bpo=28638].)

`datetime.date.fromordinal()` and `datetime.date.fromtimestamp()` are now up to 30% faster in the common case. (Contributed by Paul Ganssle in bpo-32403 [https://bugs.python.org/issue?@action=redirect&bpo=32403].)

La función `os.fwalk()` es ahora 2 veces mas rápida gracias al uso de `os.scandir()`. (Contribuido por *Serhiy Storchaka* en bpo-25996 [https://bugs.python.org/issue?@action=redirect&bpo=25996].)

La rapidez de la función `shutil.rmtree()` se ha mejorado en un 20-40% gracias al uso de la función `os.scandir()`. (Contribuido por *Serhiy Storchaka* en bpo-28564 [https://bugs.python.org/issue?@action=redirect&bpo=28564].)

Búsqueda y coincidencia optimizadas que no distinguen entre mayúsculas y minúsculas de `expresiones regulares`. La búsqueda de algunos patrones ahora puede ser hasta 20 veces más rápida. (Contribuido por *Serhiy Storchaka* en bpo-30285 [https://bugs.python.org/issue?@action=redirect&bpo=30285].)

`re.compile()` ahora convierte el parámetro `flags` en un objeto int si es `RegexFlag`. Ahora es tan rápido como Python 3.5 y más rápido que Python 3.6 en aproximadamente un 10% dependiendo del patrón. (Contribuido por *INADA Naoki* en bpo-31671 [https://bugs.python.org/issue?@action=redirect&bpo=31671].)

El método `modify()` de la clase `selectors.EpollSelector`, `selectors.PollSelector` y `selectors.DevpollSelector` puede ser alrededor de un 10% más rápido bajo cargas pesadas. (Contribuido por *Giampaolo Rodola* en bpo-30014 [https://bugs.python.org/issue?@action=redirect&bpo=30014])

El plegado constante se ha movido del optimizador de mirilla al nuevo optimizador AST, que puede realizar optimizaciones de manera más consistente. (Contribuid por *Eugene Toder* y *INADA Naoki* en bpo-29469 [https://bugs.python.org/issue?@action=redirect&bpo=29469] y bpo-11549 [https://bugs.python.org/issue?@action=redirect&bpo=11549].)

La mayoría de funciones y métodos en `abc` se han reescrito en C. Esto hace que la creación de clases base abstractas y la llamada a `isinstance()` y `issubclass()` en ellas sean 1.5 veces más rápidas. Esto también reduce el tiempo de inicio de Python hasta en un 10%. (Contribuido por *Ivan Levkivskyi* y *INADA Naoki* en bpo-31333 [https://bugs.python.org/issue?@action=redirect&bpo=31333])

Mejoras significativas de velocidad para constructores alternativos para `datetime.date` y `datetime.datetime` mediante el uso de constructores de ruta rápida cuando no se construyen subclases. (Contribuido por *Paul Ganssle* en bpo-32403 [https://bugs.python.org/issue?@action=redirect&bpo=32403])

La velocidad de comparación de instancias de `array.array` se ha mejorado considerablemente en determinados casos. Ahora es de 10 a 70 veces más rápido cuando se comparan matrices que contienen valores del mismo tipo de entero. (Contribuido por *Adrian Wielgosik* en bpo-24700 [https://bugs.python.org/issue?@action=redirect&bpo=24700].)

Las funciones `math.erf()` y `math.erfc()` ahora use la implementación de la biblioteca C (más rápida) en la mayoría de las plataformas. (Contribuido por *Serhiy Storchaka* en bpo-26121 [https://bugs.python.org/issue?@action=redirect&bpo=26121].)

Otros cambios de implementación de CPython

- Los ganchos de seguimiento ahora pueden optar por no recibir `line` y optar por recibir los eventos `opcode` del intérprete configurando los nuevos atributos `f_trace_lines` y `f_trace_opcodes` correspondientes en el marco que se está rastreando. (Contribuido por Nick Coghlan en bpo-31344 [https://bugs.python.org/issue?@action=redirect&bpo=31344]).
- Se corrigieron algunos problemas de coherencia con los atributos del módulo del paquete de espacio de nombres. Los objetos del módulo de espacio de nombres ahora tienen un `__file__` que está configurado como `None` (previamente no configurado), y su `__spec__.origin` también está configurado como `None` (anteriormente la cadena `"namespace"`). Ver bpo-32305 [https://bugs.python.org/issue?@action=redirect&bpo=32305]. Además, el `__spec__.loader` del objeto del módulo de espacio de nombres se establece en el mismo valor que `__loader__` (anteriormente, el primero se estableció en `None`). Ver bpo-32303 [https://bugs.python.org/issue?@action=redirect&bpo=32303].

- El diccionario `locals()` ahora se muestra en el orden léxico en que se definieron las variables. Anteriormente, el orden no estaba definido. (Contribuido por *Raymond Hettinger* en bpo-32690 [https://bugs.python.org/issue?@action=redirect&bpo=32690].)
- El comando `distutils upload` ya no intenta cambiar los caracteres de fin de línea CR a CRLF. Esto soluciona un problema de corrupción con sdists que terminaban con un byte equivalente a CR. (Contribuido por Bo Bayles en bpo-32304 [https://bugs.python.org/issue?@action=redirect&bpo=32304]).

Comportamiento obsoleto de Python

Las expresiones de rendimiento (tanto las cláusulas `yield` como las cláusulas `yield from`) ahora están en desuso en las comprensiones y las expresiones generadoras (aparte de la expresión iterable en la cláusula más a la izquierda **`for`**). Esto asegura que las comprensiones siempre devuelvan inmediatamente un contenedor del tipo apropiado (en lugar de potencialmente devolver un objeto generator iterator), mientras que las expresiones generadoras no intentarán intercalar su salida implícita con la salida de ninguna expresión de rendimiento explícita. En Python 3.7, tales expresiones emiten `DeprecationWarning` cuando se compilan, en Python 3.8 será un `SyntaxError`. (Contribuido por *Serhiy Storchaka* en bpo-10544 [https://bugs.python.org/issue?@action=redirect&bpo=10544].)

Devolviendo una subclase de `complex` desde `object.__complex__()` está obsoleto y será un error en futuras versiones de Python. Esto hace que `__complex__()` se compatible con `object.__int__()` y `object.__float__()`. (Contribuido por *Serhiy Storchaka* en bpo-28894 [https://bugs.python.org/issue?@action=redirect&bpo=28894].)

Módulos, funciones y métodos de Python obsoletos

aifc

`aifc.openfp()` ha quedado obsoleto y se eliminará en Python 3.9. Utilice **`aifc.open()`** en su lugar. (Contribuido por Brian Curtin en bpo-31985 [https://bugs.python.org/issue?@action=redirect&bpo=31985]).

asyncio

Se ha desaprobado el soporte para instancias en `await` de **`asyncio.Lock`** y otras primitivas de sincronización asyncio directamente. Se debe utilizar un administrador de contexto asincrónico para adquirir y liberar el recurso de sincronización. (Contribuido por *Andrew Svetlov* en bpo-32253 [https://bugs.python.org/issue?@action=redirect&bpo=32253].)

Los métodos **`asyncio.Task.current_task()`** y **`asyncio.Task.all_tasks()`** han quedado obsoletos. (Contribución de Andrew Svetlov en bpo-32250 [https://bugs.python.org/issue?@action=redirect&bpo=32250]).

collections

En Python 3.8, las clases base abstractas en **`collections.abc`** ya no estará expuesto en el módulo regular **`collections`**. Esto ayudará a crear una distinción más clara entre las clases concretas y las clases base abstractas. (Contribuido por *Serhiy Storchaka* en bpo-25988 [https://bugs.python.org/issue?@action=redirect&bpo=25988].)

dbm

`dbm.dumb` ahora admite la lectura de archivos de solo lectura y ya no escribe el archivo de índice cuando no se cambia. Ahora se emite una advertencia de obsolescencia si falta el archivo de índice y se recrea en los modos `'r'` y `'w'` (esto será un error en futuras versiones de Python). (Contribuido por *Serhiy Storchaka* en bpo-28847 [https://bugs.python.org/issue?@action=redirect&bpo=28847].)

enum

In Python 3.8, attempting to check for non-Enum objects in `Enum` classes will raise a `TypeError` (e.g. `1 in Color`); similarly, attempting to check for non-Flag objects in a `Flag` member will raise **`TypeError`** (e.g. `1 in Perm.RW`); currently, both operations return `False` instead. (Contributed by Ethan Furman in bpo-33217 [https://bugs.python.org/issue?@action=redirect&bpo=33217].)

gettext

Usando un valor no entero para seleccionar una forma plural en `gettext` ahora está en desuso. Nunca funcionó correctamente. (Contribuido por *Serhiy Storchaka* en bpo-28692 [https://bugs.python.org/issue?@action=redirect&bpo=28692].)

importlib

Los métodos **`MetaPathFinder.find_module()`** (reemplazado por `MetaPathFinder.find_spec()`) y **`PathEntryFinder.find_loader()`** (reemplazado por `PathEntryFinder.find_spec()`), ambos obsoletos en Python 3.4, ahora emiten `DeprecationWarning`. (Contribuido por Matthias Bussonnier en bpo-29576 [https://bugs.python.org/issue?@action=redirect&bpo=29576]).

The `importlib.abc.ResourceLoader` ABC has been deprecated in favour of **`importlib.abc.ResourceReader`**.

locale

`locale.format()` has been deprecated, use `locale.format_string()` instead. (Contributed by Garvit in bpo-10379 [https://bugs.python.org/issue?@action=redirect&bpo=10379].)

macpath

The **`macpath`** is now deprecated and will be removed in Python 3.8. (Contributed by Chi Hsuan Yen in bpo-9850 [https://bugs.python.org/issue?@action=redirect&bpo=9850].)

threading

`dummy_threading` y **`_dummy_thread`** han quedado obsoletos. Ya no es posible compilar Python con los subprocesos deshabilitados. Utilice **`threading`** en su lugar. (Contribuido por Antoine Pitrou en bpo-31370 [https://bugs.python.org/issue?@action=redirect&bpo=31370]).

socket

El truncamiento del valor del argumento silencioso en **`socket.htons()`** y **`socket.ntohs()`** ha quedado en desuso. En futuras versiones de Python, si el argumento pasado es mayor de 16 bits, se lanzará una excepción. (Contribuido por *Oren Milman* en bpo-28332 [https://bugs.python.org/issue?@action=redirect&bpo=28332].)

ssl

`ssl.wrap_socket()` is deprecated. Use **`ssl.SSLContext.wrap_socket()`** instead. (Contributed by Christian Heimes in bpo-28124 [https://bugs.python.org/issue?@action=redirect&bpo=28124].)

sunau

`sunau.openfp()` ha quedado obsoleto y se eliminará en Python 3.9. Utilice **`sunau.open()`** en su lugar. (Contribuido por Brian Curtin en bpo-31985 [https://bugs.python.org/issue?@action=redirect&bpo=31985]).

sys

Deprecated **`sys.set_coroutine_wrapper()`** and **`sys.get_coroutine_wrapper()`**.

La función indocumentada `sys.callstats()` ha quedado obsoleto y se eliminará en una futura versión de Python. (Contribuido por *Victor Stinner* en bpo-28799 [https://bugs.python.org/issue?@action=redirect&bpo=28799].)

wave

`wave.openfp()` has been deprecated and will be removed in Python 3.9. Use **`wave.open()`** instead. (Contributed by Brian Curtin in bpo-31985 [https://bugs.python.org/issue?@action=redirect&bpo=31985].)

Funciones y tipos obsoletos de la API C

Función **`PySlice_GetIndicesEx()`** está en desuso y se reemplaza con una macro si `Py_LIMITED_API` no está configurado o configurado en un valor en el rango entre `0x03050400` y `0x03060000` (no incluido), o es `0x03060100` o superior. (Contribuido por *Serhiy Storchaka* en bpo-27867 [https://bugs.python.org/issue?@action=redirect&bpo=27867].)

`PyOS_AfterFork()` ha quedado obsoleto. En su lugar usar **`PyOS_BeforeFork()`**, **`PyOS_AfterFork_Parent()`** o **`PyOS_AfterFork_Child()`**. (Contribuido por *Antoine Pitrou* en bpo-16500 [https://bugs.python.org/issue?@action=redirect&bpo=16500].)

Eliminación de soporte de plataforma

- *FreeBSD* 9 y las versiones anteriores ya no son compatibles oficialmente.
- Para una compatibilidad total con Unicode, incluso dentro de los módulos de extensión, ahora se espera que las plataformas *nix proporcionen al menos uno de `C.UTF-8` (configuración regional completa), `C.utf8` (configuración regional completa) o `UTF-8` (configuración regional exclusiva de `LC_CTYPE`) como una alternativa a la configuración regional `C` heredada basada en `ASCII`.
- OpenSSL 0.9.8 y 1.0.1 ya no son compatibles, lo que significa que la compilación de CPython 3.7 con compatibilidad con SSL / TLS en plataformas más antiguas que aún usan estas versiones requiere opciones de compilación personalizadas que se vinculan a una versión más reciente de OpenSSL.

En particular, este problema afecta a las distribuciones LTS Linux Debian 8 (también conocido como “jessie”) y *Ubuntu* 14.04 (también conocido como “Trusty”), ya que todavía usan OpenSSL 1.0.1 por defecto.

Debian 9 (“stretch”) y *Ubuntu* 16.04 (“xenial”), así como las versiones recientes de otras versiones de LTS Linux (por ejemplo, RHEL / CentOS 7.5, SLES 12-SP3), usan OpenSSL 1.0.2 o posterior y siguen siendo compatibles en la configuración de compilación predeterminada.

El propio CI configuration file [https://github.com/python/cpython/blob/v3.7.13/.travis.yml] de CPython proporciona un ejemplo del uso de SSL compatibility testing infrastructure [https://github.com/python/cpython/tree/3.14/Tools/ssl/multissltests.py] en el conjunto de pruebas de CPython para crear y vincular OpenSSL 1.1.0 en lugar de un sistema obsoleto proporcionado por OpenSSL.

Eliminaciones de API y funciones

Las siguientes funciones y API se han eliminado de Python 3.7:

- La función `os.stat_float_times()` ha sido removida. Se introdujo en Python 2.3 por compatibilidad con versiones anteriores de Python 2.2 y quedó obsoleto desde Python 3.1.
- Los escapes desconocidos que consisten en `'\'` y una letra ASCII en las plantillas de reemplazo para **`re.sub()`** quedaron obsoletos en Python 3.5 y ahora causarán un error.
- Se eliminó el soporte del argumento * exclude * en **`tarfile.TarFile.add()`**. Fue obsoleto en Python 2.7 y 3.2. Utilice el argumento * filtro * en su lugar.
- La función **`ntpath.splitunc()`** quedó obsoleta en Python 3.1 y ahora se ha eliminado. Utilice **`splitdrive()`** en su lugar.
- **`collections.namedtuple()`** ya no admite el parámetro *verbose* o el atributo `_source` que mostraba el código fuente generado para la clase de tupla nombrada. Esto fue parte de una optimización diseñada para acelerar la creación de clases. (Contribuido por *Jelle Zijlstra* con nuevas mejoras de *INADA*

Naoki, *Serhiy Storchaka*, y *Raymond Hettinger* en bpo-28638 [https://bugs.python.org/issue?@action=redirect&bpo=28638].)

- Funciones **`bool()`**, **`float()`**, **`list()`** y **`tuple()`** ya no aceptan argumentos de palabras clave. El primer argumento de **`int()`** ahora solo se puede pasar como argumento posicional.
- Removed previously deprecated in Python 2.4 classes `Plist`, `Dict` and `_InternalDict` in the **`plistlib`** module. Dict values in the result of functions **`readPlist()`** and **`readPlistFromBytes()`** are now normal dicts. You no longer can use attribute access to access items of these dictionaries.
- La función `asyncio.windows_utils.socketpair()` ha sido removida. En su lugar use la función **`socket.socketpair()`**, está disponible en todas las plataformas desde Python 3.5. `asyncio.windows_utils.socketpair` era un alias para `socket.socketpair` en Python 3.5 y mas nuevo.
- **`asyncio`** ya no exporta los módulos **`selectors`** y **`_overlapped`** como `asyncio.selectors` y `asyncio._overlapped`. Reemplace `from asyncio import selectors` por `import selectors`.
- Direct instantiation of **`ssl.SSLSocket`** and **`ssl.SSLObject`** objects is now prohibited. The constructors were never documented, tested, or designed as public constructors. Users were supposed to use **`ssl.wrap_socket()`** or **`ssl.SSLContext`**. (Contributed by Christian Heimes in bpo-32951 [https://bugs.python.org/issue?@action=redirect&bpo=32951].)
- Se ha eliminado el comando `distutils install_misc` no utilizado. (Contribuido por Eric N. Vander Weele en bpo-29218 [https://bugs.python.org/issue?@action=redirect&bpo=29218]).

Eliminaciones de módulos

El modulo `fpectl` ha sido removido. Nunca se habilitó de forma predeterminada, nunca funcionó correctamente en x86-64 y cambió la ABI de Python de manera que causó la rotura inesperada de las extensiones C. (Contribuido por *Nathaniel J. Smith* en bpo-29137 [https://bugs.python.org/issue?@action=redirect&bpo=29137].)

Cambios solo en Windows

El lanzador de python, (py.exe), puede aceptar especificadores de 32 y 64 bits **sin** tener que especificar también una versión secundaria. Entonces, `py -3-32` y `py -3-64` se vuelven válidos, así como `py -3.7-32`, también los -*m*-64 and -*m.n*-64. Los formularios ahora se aceptan para forzar Python de 64 bits incluso si de otro modo se hubieran utilizado 32 bits. Si la versión especificada no está disponible, py.exe saldrá del error. (Contribuido por *Steve Barnes* en bpo-30291 [https://bugs.python.org/issue?@action=redirect&bpo=30291].)

El lanzador se puede ejecutar como `py -0` para producir una lista de las pitones instaladas, *con las marcas predeterminadas con un asterisco*. La ejecución de `py -0p` incluirá las rutas. Si py se ejecuta con un especificador de versión que no puede coincidir, también imprimirá la *forma corta* lista de especificadores disponibles. (Contribuido por *Steve Barnes* en bpo-30362 [https://bugs.python.org/issue?@action=redirect&bpo=30362].)

Portando a Python 3.7

Esta sección enumera los cambios descritos anteriormente y otras correcciones de errores que pueden requerir cambios en su código.

Cambios en el comportamiento de Python

- **`async`** y **`await`** los nombres ahora son palabras clave reservadas. El código que usa estos nombres como identificadores lanzará ahora un **`SyntaxError`**. (Contribuido por *Jelle Zijlstra* en bpo-30406 [https://bugs.python.org/issue?@action=redirect&bpo=30406].)
- **PEP 479** [https://peps.python.org/pep-0479/] está habilitado para todo el código en Python 3.7, lo que significa que las excepciones de **`StopIteration`** generadas directa o

indirectamente en corutinas y generadores se transforman en excepciones de **`RuntimeError`**. (Contribuido por *Yury Selivanov* en bpo-32670 [https://bugs.python.org/issue?@action=redirect&bpo=32670].)

- Los métodos **`object.__aiter__()`** ya no se pueden declarar cómo asincrónicos. (Contribuido por *Yury Selivanov* en bpo-31709 [https://bugs.python.org/issue?@action=redirect&bpo=31709].)

- Debido a un descuido, las versiones anteriores de Python aceptaron erróneamente la siguiente sintaxis:

  ```
  f(1 for x in [1],)

  class C(1 for x in [1]):
      pass
  ```

  Python 3.7 ahora lanza correctamente una **`SyntaxError`**, como una expresión generadora siempre debe estar directamente dentro de un conjunto de paréntesis y no puede tener una coma a ambos lados, y la duplicación de los paréntesis solo se puede omitir en las llamadas. (Contribuido por *Serhiy Storchaka* en bpo-32012 [https://bugs.python.org/issue?@action=redirect&bpo=32012] y bpo-32023 [https://bugs.python.org/issue?@action=redirect&bpo=32023].)

- Al usar el interruptor **`-m`**, el directorio de trabajo inicial ahora se agrega a: data **`sys.path`**, en lugar de una cadena vacía (que denota dinámicamente el directorio de trabajo actual en el momento de cada importación). Cualquier programa que esté comprobando la cadena vacía, o que dependa de otro modo del comportamiento anterior, deberá actualizarse en consecuencia (por ejemplo, comprobando también si hay `os.getcwd()` o `os.path.dirname(__main__.__file__)`, dependiendo de por qué el código buscaba la cadena vacía en primer lugar).

Cambios en la API Python

- **`socketserver.ThreadingMixIn.server_close`** now waits until all non-daemon threads complete. Set the new **`socketserver.ThreadingMixIn.block_on_close`** class attribute to `False` to get the pre-3.7 behaviour. (Contributed by Victor Stinner in bpo-31233 [https://bugs.python.org/issue?@action=redirect&bpo=31233] and bpo-33540 [https://bugs.python.org/issue?@action=redirect&bpo=33540].)

- **`socketserver.ForkingMixIn.server_close`** now waits until all child processes complete. Set the new **`socketserver.ForkingMixIn.block_on_close`** class attribute to `False` to get the pre-3.7 behaviour. (Contributed by Victor Stinner in bpo-31151 [https://bugs.python.org/issue?@action=redirect&bpo=31151] and bpo-33540 [https://bugs.python.org/issue?@action=redirect&bpo=33540].)

- La función **`locale.localeconv()`** ahora establece temporalmente la configuración regional `LC_CTYPE` en el valor de `` `LC_NUMERIC `` en algunos casos. (Contribuido por *Victor Stinner* en bpo-31900 [https://bugs.python.org/issue?@action=redirect&bpo=31900].)

- **`pkgutil.walk_packages()`** ahora genera un **`ValueError`** si la* ruta * es una cadena. Anteriormente se devolvió una lista vacía. (Contribuido por *Sanyam Khurana* en bpo-24744 [https://bugs.python.org/issue?@action=redirect&bpo=24744].)

- Un argumento de cadena de formato para **`string.Formatter.format()`** es ahora solo posicional. Pasarlo como un argumento de palabra clave quedó obsoleto en Python 3.5. (Contribuido por *Serhiy Storchaka* en bpo-29193 [https://bugs.python.org/issue?@action=redirect&bpo=29193].)

- Atributos **`key`**, **`value`** y **`coded_value`** de la clase **`http.cookies.Morsel`** ahora son de solo lectura. La asignación a ellos quedó obsoleta en Python 3.5. Utilizar el método **`set()`** para configurarlos. (Contribuido por *Serhiy Storchaka* en bpo-29192 [https://bugs.python.org/issue?@action=redirect&bpo=29192].)

- El argumento *mode* de `os.makedirs()` ya no afecta los bits de permiso de archivo de los directorios de nivel intermedio recién creados. Para configurar sus bits de permiso de archivo, puede configurar umask antes de invocar `makedirs()`. (Aportado por Serhiy Storchaka en bpo-19930 [https://bugs.python.org/issue?@action=redirect&bpo=19930].)

- El tipo `struct.Struct.format` ahora es `str` en lugar de: class `bytes`. (Contribuido por *Victor Stinner* en bpo-21071 [https://bugs.python.org/issue?@action=redirect&bpo=21071].)

- **`cgi.parse_multipart()`** ahora acepta los argumentos *encoding* y *errors* y devuelve los mismos resultados que **`FieldStorage`**: para los campos que no son archivos, el valor asociado a una clave es una lista de cadenas, no bytes. (Contribuido por Pierre Quentel en bpo-29979 [https://bugs.python.org/issue?@action=redirect&bpo=29979]).

- Debido a cambios internos en `socket`, llamar `socket.fromshare()` en un socket creado por `socket.share` en versiones anteriores de Python no es compatible.

- `repr` para `BaseException` ha cambiado para no incluir la coma final. La mayoría de las excepciones se ven afectadas por este cambio. (Contribuido por *Serhiy Storchaka* en bpo-30399 [https://bugs.python.org/issue?@action=redirect&bpo=30399].)

- `repr` para `datetime.timedelta` ha cambiado para incluir los argumentos de palabras clave en la salida. (Contribuido por *Utkarsh Upadhyay* en bpo-30302 [https://bugs.python.org/issue?@action=redirect&bpo=30302].)

- Debido a que `shutil.rmtree()` ahora se implementa usando la función `os.scandir()`, el controlador especificado por el usuario *onerror* ahora se llama con el primer argumento `os.scandir` en lugar de `os.listdir` cuando falla el listado del directorio.

- Es posible que en el futuro se agregue soporte para conjuntos anidados y operaciones de conjuntos en expresiones regulares como en Unicode Technical Standard #18 [https://unicode.org/reports/tr18/]. Esto cambiaría la sintaxis. Para facilitar este cambio futuro, por el momento, se planteará un **`FutureWarning`** en casos ambiguos. Que incluyen conjuntos que comienzan con un literal ′[` o que contienen secuencias de caracteres literales ′–`, ′&&′, ′~~′, and ′||′. Para evitar una advertencia, escapa de ellos con una barra invertida. (Contribuido por *Serhiy Storchaka* en bpo-30349 [https://bugs.python.org/issue?@action=redirect&bpo=30349].)

- El resultado de dividir una cuerda en un **`expresión regular`** que podría coincidir con una cadena vacía. Por ejemplo, dividir en `r′\s*′` ahora dividirá no solo en espacios en blanco como lo hizo anteriormente, sino también en cadenas vacías antes de todos los caracteres que no sean espacios en blanco y justo antes del final de la cadena. El comportamiento anterior se puede restaurar cambiando el patrón a `r′\s+′`. Una **`FutureWarning`** se emitió para tales patrones desde Python 3.5.

  Para patrones que coinciden con cadenas vacías y no vacías, el resultado de la búsqueda de todas las coincidencias también puede cambiarse en otros casos. Por ejemplo en la cadena `'a\n\n'`, el patrón `′\n'` no solo coincidirá con cadenas vacías en las posiciones 2 y 3, sino también la cadena `'\n'` en las posiciones 2–3. Para hacer coincidir solo líneas en blanco, el patrón debe reescribirse como `r'(?m)^[^\S\n]*$'`.

  `re.sub()` ahora reemplaza las coincidencias vacías adyacentes a una coincidencia no vacía anterior. Por ejemplo, `re.sub('x*', '-', 'abxd')` ahora devuelve `'-a-b--d-'` en lugar de `'-a-b-d-'` (el primer signo menos entre “b” y “d” reemplaza a “x”, y el segundo signo menos reemplaza a una cadena vacía entre “x” y “d”).

  (Contribuido por *Serhiy Storchaka* en bpo-25054 [https://bugs.python.org/issue?@action=redirect&bpo=25054] y bpo-32308 [https://bugs.python.org/issue?@action=redirect&bpo=32308].)

- Cambie `re.escape()` para escapar solo de los caracteres especiales regex en lugar de escapar de todos los caracteres que no sean letras ASCII, números y `'_'`. (Contribuido por *Serhiy Storchaka* en bpo-29995 [https://bugs.python.org/issue?@action=redirect&bpo=29995].)

- Los marcos `tracemalloc.Traceback` ahora se ordenan del más antiguo al más reciente para ser más coherentes con `traceback`. (Contribuido por *Jesse Bakker* en bpo-32121 [https://bugs.python.org/issue?@action=redirect&bpo=32121].)

- En sistemas operativos que admiten banderas de bits `socket.SOCK_NONBLOCK` o `socket.SOCK_CLOEXEC`, el `socket.type` ya no los tiene aplicados. Por lo tanto, comprobaciones como `if sock.type == socket.SOCK_STREAM` funcionan como se espera en todas las plataformas. (Contribuido por *Yury Selivanov* en bpo-32331 [https://bugs.python.org/issue?@action=redirect&bpo=32331].)

- En Windows, el valor predeterminado para el argumento *close_fds* de `subprocess.Popen` se cambió de `False` a `True` al redirigir los identificadores estándar. Si anteriormente dependía de que los identificadores fueran heredados al usar **`subprocess.Popen`** con la redirección estándar de io, tendrá que pasar `close_fds=False` para preservar el comportamiento anterior, o usar `STARTUPINFO.lpAttributeList`.

- `importlib.machinery.PathFinder.invalidate_caches()` – que afecta implícitamente a `importlib.invalidate_caches()` – ahora borra entradas en `sys.path_importer_cache` que están configuradas en `None`. (Contribuido por *Brett Cannon* en bpo-33169 [https://bugs.python.org/issue?@action=redirect&bpo=33169].)

- En `asyncio`, `loop.sock_recv()`, `loop.sock_sendall()`, `loop.sock_accept()`, `loop.getaddrinfo()`, `loop.getnameinfo()` se han cambiado para que sean métodos de rutina adecuados para que coincidan con su documentación. Anteriormente, estos métodos devolvían instancias **`asyncio.Future`**. (Contribuido por *Yury Selivanov* en bpo-32327 [https://bugs.python.org/issue?@action=redirect&bpo=32327].)

- **`asyncio.Server.sockets`** ahora devuelve una copia de la lista interna de sockets del servidor, en lugar de devolverla directamente. (Contribuido por *Yury Selivanov* en bpo-32662 [https://bugs.python.org/issue?@action=redirect&bpo=32662].)
- **`Struct.format`** es ahora una instancia **`str`** en lugar de una instancia **`bytes`**. (Contribuido por *Victor Stinner* en bpo-21071 [https://bugs.python.org/issue?@action=redirect&bpo=21071].)
- **`argparse`** *subparsers* ahora pueden hacerse obligatorios pasando `required=True` a **`ArgumentParser.add_subparsers()`**. (Contribuido por *Anthony Sottile* en bpo-26510 [https://bugs.python.org/issue?@action=redirect&bpo=26510].)
- El sistema **`ast.literal_eval()`** es ahora más estricto. Ya no se permite la suma ni la resta de números arbitrarios. (Contribuido por Serhiy Storchaka en bpo-31778 [https://bugs.python.org/issue?@action=redirect&bpo=31778]).
- **`Calendar.itermonthdates`** ahora lanzará una excepción cuando una fecha cae fuera del rango de `0001-01-01` a `9999-12-31`. Para admitir aplicaciones que no pueden tolerar tales excepciones, se puede utilizar el nuevo **`Calendar.itermonthdays3`** y **`Calendar.itermonthdays4`**. Los nuevos métodos devuelven tuplas y no están restringidos por el rango admitido por **`datetime.date`**. (Contribuido por *Alexander Belopolsky* en bpo-28292 [https://bugs.python.org/issue?@action=redirect&bpo=28292].)
- **`collections.ChainMap`** ahora conserva el orden de las asignaciones subyacentes. (Contribuido por *Raymond Hettinger* en bpo-32792 [https://bugs.python.org/issue?@action=redirect&bpo=32792].)

- El método `submit()` de **`concurrent.futures.ThreadPoolExecutor`** y **`concurrent.futures.ProcessPoolExecutor`** ahora genera un **`RuntimeError`** si se llama durante el cierre del intérprete. (Contribuido por *Mark Nemec* en bpo-33097 [https://bugs.python.org/issue?@action=redirect&bpo=33097].)

- El constructor **`configparser.ConfigParser`** ahora usa `read_dict()` para procesar los valores predeterminados, haciendo que su comportamiento sea coherente con el resto del analizador. Las claves y los valores que no son cadenas en el diccionario de valores predeterminados ahora se están convirtiendo implícitamente en cadenas. (Contribuido por *James Tocknell* en bpo-23835 [https://bugs.python.org/issue?@action=redirect&bpo=23835].)

- Se eliminaron varias importaciones internas indocumentadas. Un ejemplo es que `os.errno` ya no está disponible; use `import errno` directamente en su lugar. Tenga en cuenta que estas importaciones internas no documentadas pueden eliminarse en cualquier momento sin previo aviso, incluso en las versiones micro.

Cambios en la API de C

La función **`PySlice_GetIndicesEx()`** se considera insegura para secuencias de tamaño variable. Si los índices de corte no son instancias de **`int`**, sino objetos que implementan el método **`__index__()`**, la secuencia se puede cambiar de tamaño después de pasar su longitud a **`PySlice_GetIndicesEx()`**. Esto puede llevar a que se devuelvan índices fuera de la longitud de la secuencia. Esto puede llevar a devolver índices fuera de la longitud de la secuencia. Para evitar posibles problemas, use las nuevas funciones **`PySlice_Unpack()`** y **`PySlice_AdjustIndices()`**. (Contribuido por *Serhiy Storchaka* en bpo-27867 [https://bugs.python.org/issue?@action=redirect&bpo=27867].)

Cambios en el código de bytes de CPython

There are two new opcodes: **`LOAD_METHOD`** and **`CALL_METHOD`**.

(Contributed by Yury Selivanov and INADA Naoki in bpo-26110 [https://bugs.python.org/issue?@action=redirect&bpo=26110].)

The **`STORE_ANNOTATION`** opcode has been removed. (Contributed by Mark Shannon in bpo-32550 [https://bugs.python.org/issue?@action=redirect&bpo=32550].)

Cambios solo en Windows

El archivo utilizado para anular **`sys.path`** ahora se llama `<python-executable>._pth` en lugar de `'sys.path'`. Consulte Encontrar módulos para obtener más información. (Aportado por Steve Dower en bpo-28137 [https://bugs.python.org/issue?@action=redirect&bpo=28137].)

Otra implementación de cambios en CPython

En preparación para posibles cambios futuros en la API pública de inicialización del tiempo de ejecución de CPython (ver **PEP 432** [https://peps.python.org/pep-0432/] para un borrador inicial, pero algo desactualizado), La lógica de gestión de configuración y puesta en marcha interna de CPython se ha re-factorizado significativamente. Si bien estas actualizaciones están destinadas a ser completamente transparentes tanto para las aplicaciones integradas como para los usuarios de la CLI de CPython normal, se mencionan aquí ya que la refactorización cambia el orden interno de varias operaciones durante el inicio del intérprete y, por lo tanto, pueden descubrir defectos previamente latentes, ya sea en aplicaciones incrustadas, o en el propio CPython. (Inicialmente contribuido por *Nick Coghlan* y *Eric Snow* como parte de bpo-22257 [https://bugs.python.org/issue?@action=redirect&bpo=22257], y actualizado por *Nick*, *Eric*, y *Victor Stinner* en varios otros números). Algunos detalles conocidos afectados:

- Actualmente, las aplicaciones integradas no pueden utilizar **`PySys_AddWarnOptionUnicode()`** debido al requisito de crear un objeto Unicode antes de llamar a `Py_Initialize`. Utilice **`PySys_AddWarnOption()`** en su lugar.
- Los filtros de advertencias agregados por una aplicación de incrustación con **`PySys_AddWarnOption()`** ahora deberían

tener prioridad de manera más consistente sobre los filtros predeterminados establecidos por el intérprete

Debido a cambios en la forma en que se configuran los filtros de advertencias predeterminados, establecer **`Py_BytesWarningFlag`** en un valor mayor que uno ya no es suficiente para que ambos emitan mensajes **`BytesWarning`** y los conviertan en excepciones. En su lugar, se debe establecer la bandera (para hacer que las advertencias se emitan en primer lugar) y se debe agregar un filtro de advertencias explícito `error::BytesWarning` para convertirlas en excepciones.

Debido a un cambio en la forma en que el compilador maneja las cadenas de documentos, el implícito `return None` en el cuerpo de una función que consiste únicamente en una cadena de documentos ahora se marca como ocurriendo en la misma línea que la cadena de documentos, no en la línea de encabezado de la función.

El estado de excepción actual se ha movido del objeto de trama a la co-rutina. Esto simplificó el intérprete y corrigió un par de errores oscuros causados por tener un estado de excepción de intercambio al ingresar o salir de un generador. (Contribuido por *Mark Shannon* en bpo-25612 [https://bugs.python.org/issue?@action=redirect&bpo=25612].)

Cambios notables en Python 3.7.1

A partir de 3.7.1, **`Py_Initialize()`** ahora lee y respeta constantemente todos los mismos ajustes de entorno que **`Py_Main()`** (en versiones anteriores de Python, respetaba un subconjunto mal definido de esas variables de entorno, mientras que en Python 3.7.0 no leyó ninguna de ellas debido a bpo-34247 [https://bugs.python.org/issue?@action=redirect&bpo=34247]). Si este comportamiento no es deseado, establezca **`Py_IgnoreEnvironmentFlag`** en 1 antes de llamar a **`Py_Initialize()`**.

En 3.7.1 la API C para las variables de Contexto ha sido actualizada para usar punteros **`PyObject`**. Ver también bpo-34762 [https://bugs.python.org/issue?@action=redirect&bpo=34762].

En 3.7.1 el módulo `tokenize` ahora emite un token `NEWLINE` cuando se le proporciona una entrada que no tiene una nueva línea al final. Este comportamiento ahora coincide con lo que hace el tokenizador C internamente. (Contribuido por *Ammar Askar* en bpo-33899 [https://bugs.python.org/issue?@action=redirect&bpo=33899].)

Cambios notables en Python 3.7.2

En 3.7.2, `venv` en Windows ya no copia los binarios originales, sino que crea scripts de redireccionamiento llamados `python.exe` y `pythonw.exe` en su lugar. Esto resuelve un problema de larga data en el que todos los entornos virtuales tendrían que actualizarse o recrearse con cada actualización de Python. Sin embargo, tenga en cuenta que esta versión aún requerirá la recreación de entornos virtuales para obtener los nuevos scripts.

Cambios notables en Python 3.7.6

Debido a importantes problemas de seguridad, el parámetro *reuse_address* de `asyncio.loop.create_datagram_endpoint()` ya no es compatible. Esto se debe al comportamiento de la opción socket `SO_REUSEADDR` en UDP. Para obtener más detalles, consulte la documentación de `loop.create_datagram_endpoint()`. (Contribuido por *Kyle Stanley*, *Antoine Pitrou*, y *Yury Selivanov* en bpo-37228 [https://bugs.python.org/issue?@action=redirect&bpo=37228].)

Cambios notables en Python 3.7.10

Las versiones anteriores de Python permitían utilizar tanto `;` como `&` como separadores de parámetros de consulta en `urllib.parse.parse_qs()` y `urllib.parse.parse_qsl()`. Por cuestiones de seguridad y para cumplir con las recomendaciones más recientes del W3C, esto se ha modificado para permitir solo una clave separadora, con `&` como valor predeterminado. Este cambio también afecta a **`cgi.parse()`** y **`cgi.parse_multipart()`**, ya que utilizan las funciones afectadas internamente. Para obtener más detalles, consulte su documentación respectiva. (Contribuido por Adam Goldschmidt, Senthil Kumaran y Ken Jin en bpo-42967 [https://bugs.python.org/issue?@action=redirect&bpo=42967]).

Cambios notables en Python 3.7.11

Una corrección de seguridad modifica el comportamiento de **`ftplib.FTP`** para que no confíe en la dirección IPv4 enviada desde el servidor remoto al configurar un canal de datos pasivo. En su lugar, reutilizamos la dirección IP del servidor FTP. Para el código inusual que requiere el comportamiento anterior, configure un atributo `trust_server_pasv_ipv4_address` en su instancia FTP en `True`. (Consulte gh-87451 [https://github.com/python/cpython/issues/87451])

La presencia de caracteres de nueva línea o de tabulación en partes de una URL permite algunas formas de ataques. Siguiendo la especificación WHATWG que actualiza RFC 3986, el analizador **`urllib.parse()`** elimina de la URL los caracteres de nueva línea ASCII `\n`, `\r` y de tabulación `\t` para evitar este tipo de ataques. Los caracteres eliminados se controlan mediante una nueva variable de nivel de módulo `urllib.parse._UNSAFE_URL_BYTES_TO_REMOVE`. (Véase gh-88048 [https://github.com/python/cpython/issues/88048])

Cambios notables en Python 3.7.14

La conversión entre **`int`** y **`str`** en bases distintas de 2 (binario), 4, 8 (octal), 16 (hexadecimal) o 32, como la base 10 (decimal), ahora genera un error **`ValueError`** si la cantidad de dígitos en formato de cadena supera un límite para evitar posibles ataques de denegación de servicio debido a la complejidad algorítmica. Esta es una mitigación para **CVE 2020-10735** [https://www.cve.org/CVERecord?id=CVE-2020-10735]. Este límite se puede configurar o deshabilitar mediante una variable de entorno, un indicador de línea de comandos o las API de **`sys`**. Consulte la documentación de integer string conversion length limitation. El límite predeterminado es de 4300 dígitos en formato de cadena.

Qué hay de nuevo en Python 3.6

Editors:

Elvis Pranskevichus <elvis@magic.io>, Yury Selivanov <yury@magic.io>

This article explains the new features in Python 3.6, compared to 3.5. Python 3.6 was released on December 23, 2016. See the changelog [https://docs.python.org/3.6/whatsnew/changelog.html] for a full list of changes.

Ver también

PEP 494 [https://peps.python.org/pep-0494/] - Programa de lanzamiento de Python 3.6

Resumen: aspectos destacados de la versión

Nuevas funciones de sintaxis:

- PEP 498, literales de cadena formateados.
- PEP 515, guiones bajos en literales numéricos.
- PEP 526, sintaxis para anotaciones variables.
- PEP 525, generadores asincrónicos.
- PEP 530: comprensiones asincrónicas.

Nuevos módulos de biblioteca:

- **`secrets`**: PEP 506 – Adding A Secrets Module To The Standard Library.

Mejoras en la implementación de CPython:

- El tipo dict se ha vuelto a implementar para utilizar un more compact representation basado en a proposal by Raymond Hettinger [https://mail.python.org/pipermail/python-dev/2012-December/123028.html] y similar al PyPy dict implementation [https://morepypy.blogspot.com/2015/01/faster-more-memory-efficient-and-more.html]. Esto dio como resultado que los diccionarios usaran entre un 20% y un 25% menos de memoria en comparación con Python 3.5.
- La personalización de la creación de clases se ha simplificado con el new protocol.
- El orden de definición de atributo de clase es now preserved.
- El orden de los elementos en `**kwargs` ahora corresponds to the order en el que se pasaron argumentos de palabras clave a la función.
- Se han agregado DTrace y SystemTap probing support.
- La nueva variable de entorno PYTHONMALLOC ahora se puede utilizar para depurar la asignación de memoria del intérprete y los errores de acceso.

Mejoras significativas en la biblioteca estándar:

- El módulo **`asyncio`** ha recibido nuevas funciones, mejoras significativas de rendimiento y usabilidad, y una buena cantidad de correcciones de errores. A partir de Python 3.6, el módulo `asyncio` ya no es provisional y su API se considera estable.
- Se ha implementado un nuevo file system path protocol para admitir path-like objects. Todas las funciones de biblioteca estándar que operan en rutas se han actualizado para trabajar con el nuevo protocolo.
- El módulo **`datetime`** ha ganado soporte para Local Time Disambiguation.
- El módulo **`typing`** recibió varios improvements.
- El módulo **`tracemalloc`** se ha rediseñado significativamente y ahora se utiliza para proporcionar una mejor salida para **`ResourceWarning`**, así como para proporcionar un mejor diagnóstico de errores de asignación de memoria. Consulte PYTHONMALLOC section para obtener más información.

Mejoras de seguridad:

- Se ha agregado el nuevo módulo **`secrets`** para simplificar la generación de números pseudoaleatorios criptográficamente fuertes adecuados para administrar secretos como autenticación de cuentas, tokens y similares.
- En Linux, **`os.urandom()`** ahora se bloquea hasta que se inicializa el grupo de entropía aleatoria del sistema para aumentar la seguridad. Consulte el **PEP 524** [https://peps.python.org/pep-0524/] para conocer la justificación.
- Los módulos **`hashlib`** y **`ssl`** ahora son compatibles con OpenSSL 1.1.0.
- Se han mejorado la configuración predeterminada y el conjunto de funciones del módulo **`ssl`**.
- El módulo **`hashlib`** recibió soporte para los algoritmos hash BLAKE2, SHA-3 y SHAKE y la función de derivación de claves **`scrypt()`**.

Mejoras de Windows:

- PEP 528 y PEP 529, el sistema de archivos de Windows y la codificación de la consola cambiaron a UTF-8.
- El lanzador `py.exe`, cuando se usa de forma interactiva, ya no prefiere Python 2 sobre Python 3 cuando el usuario no especifica una versión (a través de argumentos de línea de comando o un archivo de configuración). El manejo de las líneas shebang permanece sin cambios - «python» se refiere a Python 2 en ese caso.
- `python.exe` y `pythonw.exe` se han marcado como con reconocimiento de ruta larga, lo que significa que es posible que ya no se aplique el límite de ruta de 260 caracteres. Consulte removing the MAX_PATH limitation para obtener más detalles.
- Se puede agregar un archivo `._pth` para forzar el modo aislado y especificar completamente todas las rutas de búsqueda para evitar búsquedas en el registro y el entorno. Consulte la documentación para obtener más información.
- Un archivo `python36.zip` ahora funciona como un punto de referencia para inferir **`PYTHONHOME`**. Consulte la documentación para obtener más información.

Nuevas características

PEP 498: Literales de cadena formateados

PEP 498 [https://peps.python.org/pep-0498/] introduce un nuevo tipo de cadenas literales: *f-strings* o formatted string literals.

Los literales de cadena formateados tienen el prefijo `'f'` y son similares a las cadenas de formato aceptadas por **`str.format()`**. Contienen campos de reemplazo rodeados por llaves. Los campos de reemplazo son expresiones, que se evalúan en tiempo de ejecución y luego se formatean usando el protocolo **`format()`**:

```
>>> name = "Fred"
>>> f"He said his name is {name}."
'He said his name is Fred.'
>>> width = 10
>>> precision = 4
>>> value = decimal.Decimal("12.34567")
>>> f"result: {value:{width}.{precision}}"  # nested fie
'result:      12.35'
```

Ver también

PEP 498 [https://peps.python.org/pep-0498/] - Interpolación de cadenas literal.
PEP escrito e implementado por Eric V. Smith.

Feature documentation.

PEP 526: Sintaxis para anotaciones de variables

PEP 484 [https://peps.python.org/pep-0484/] introdujo el estándar para anotaciones de tipo de parámetros de función, también conocido como sugerencias de tipo. Este PEP agrega sintaxis a Python para anotar los tipos de variables, incluidas las variables de clase y las variables de instancia:

```
primes: List[int] = []

captain: str  # Note: no initial value!
```

```
class Starship:
    stats: Dict[str, int] = {}
```

Al igual que para las anotaciones de funciones, el intérprete de Python no asigna ningún significado particular a las anotaciones de variables y solo las almacena en el atributo `__annotations__` de una clase o módulo.

A diferencia de las declaraciones de variables en lenguajes tipados estáticamente, el objetivo de la sintaxis de anotación es proporcionar una manera fácil de especificar metadatos de tipo estructurado para herramientas y bibliotecas de terceros a través del árbol de sintaxis abstracta y el atributo `__annotations__`.

Ver también

PEP 526 [https://peps.python.org/pep-0526/]: sintaxis para anotaciones de variables.

PEP escrito por Ryan Gonzalez, Philip House, Ivan Levkivskyi, Lisa Roach y Guido van Rossum. Implementado por Ivan Levkivskyi.

Herramientas que utilizan o utilizarán la nueva sintaxis: mypy [https://www.mypy-lang.org/], pytype [https://github.com/google/pytype], PyCharm, etc.

PEP 515: subrayados en literales numéricos

PEP 515 [https://peps.python.org/pep-0515/] agrega la capacidad de usar guiones bajos en literales numéricos para mejorar la legibilidad. Por ejemplo:

```
>>> 1_000_000_000_000_000
1000000000000000
>>> 0x_FF_FF_FF_FF
4294967295
```

Se permiten guiones bajos simples entre dígitos y después de cualquier especificador de base. No se permiten guiones bajos al principio, al final o múltiples en una fila.

El lenguaje string formatting ahora también admite la opción `'_'` para indicar el uso de un guión bajo como separador de miles para los tipos de presentación de punto flotante y para el tipo de presentación de enteros `'d'`. Para los tipos de presentación de enteros `'b'`, `'o'`, `'x'` y `'X'`, se insertarán guiones bajos cada 4 dígitos:

```
>>> '{:_}'.format(1000000)
'1_000_000'
>>> '{:_x}'.format(0xFFFFFFFF)
'ffff_ffff'
```

Ver también

PEP 515 [https://peps.python.org/pep-0515/] - Guiones bajos en literales numéricos

PEP escrito por Georg Brandl y Serhiy Storchaka.

PEP 525: Generadores asíncronos

PEP 492 [https://peps.python.org/pep-0492/] introdujo soporte para corrutinas nativas y sintaxis `async` / `await` para Python 3.5. Una limitación notable de la implementación de Python 3.5 es que no fue posible usar `await` y `yield` en el mismo cuerpo de función. En Python 3.6 se eliminó esta restricción, lo que permite definir *asynchronous generators*

```
async def ticker(delay, to):
    """Yield numbers from 0 to *to* every *delay* secon
    for i in range(to):
        yield i
        await asyncio.sleep(delay)
```

La nueva sintaxis permite un código más rápido y conciso.

Ver también

PEP 525 [https://peps.python.org/pep-0525/] - Generadores asincrónicos

PEP escrito e implementado por Yury Selivanov.

PEP 530: Comprensiones asincrónicas

PEP 530 [https://peps.python.org/pep-0530/] agrega soporte para usar `async for` en listas, conjuntos, dict comprensiones y expresiones generadoras:

```
result = [i async for i in aiter() if i % 2]
```

Además, las expresiones `await` son compatibles con todo tipo de comprensiones:

```
result = [await fun() for fun in funcs if await conditio
```

Ver también

PEP 530 [https://peps.python.org/pep-0530/] - Comprensiones asincrónicas

PEP escrito e implementado por Yury Selivanov.

PEP 487: personalización más sencilla de la creación de clases

Ahora es posible personalizar la creación de subclases sin utilizar una metaclase. El nuevo método de clase `__init_subclass__` se llamará en la clase base siempre que se cree una nueva subclase:

```
class PluginBase:
    subclasses = []

    def __init_subclass__(cls, **kwargs):
        super().__init_subclass__(**kwargs)
        cls.subclasses.append(cls)

class Plugin1(PluginBase):
    pass

class Plugin2(PluginBase):
    pass
```

Para permitir que las llamadas `super()` sin argumentos funcionen correctamente desde las implementaciones `__init_subclass__()`, las metaclases personalizadas deben garantizar que la nueva entrada del espacio de nombres `__classcell__` se propague a `type.__new__` (como se describe en Creando el objeto de clase).

Ver también

PEP 487 [https://peps.python.org/pep-0487/]: personalización más sencilla de la creación de clases

PEP escrito e implementado por Martin Teichmann.

Feature documentation

PEP 487: Mejoras en el protocolo descriptor

PEP 487 [https://peps.python.org/pep-0487/] amplía el protocolo descriptor para incluir el nuevo método opcional `__set_name__()`. Siempre que se defina una nueva clase, se llamará al nuevo método en todos los descriptores incluidos en la definición, proporcionándoles una referencia a la clase que se está definiendo y el nombre dado al descriptor dentro del espacio de nombres de la clase. En otras palabras, las instancias de descriptores ahora pueden conocer el nombre de atributo del descriptor en la clase propietaria:

```
class IntField:
    def __get__(self, instance, owner):
        return instance.__dict__[self.name]

    def __set__(self, instance, value):
        if not isinstance(value, int):
            raise ValueError(f'expecting integer in {sel
        instance.__dict__[self.name] = value

    # this is the new initializer:
    def __set_name__(self, owner, name):
        self.name = name

class Model:
    int_field = IntField()
```

Ver también

PEP 487 [https://peps.python.org/pep-0487/]: personalización más sencilla de la creación de clases
PEP escrito e implementado por Martin Teichmann.

Feature documentation

PEP 519: Agregar un protocolo de ruta del sistema de archivos

Históricamente, las rutas del sistema de archivos se han representado como objetos `str` o `bytes`. Esto ha llevado a las personas que escriben códigos que operan en las rutas del sistema de archivos a asumir que dichos objetos son solo uno de esos dos tipos (un `int` que representa un descriptor de archivo no cuenta, ya que no es una ruta de archivo). Desafortunadamente, esa suposición evita que las representaciones de objetos alternativos de las rutas del sistema de archivos como `pathlib` funcionen con código preexistente, incluida la biblioteca estándar de Python.

Para solucionar esta situación, se ha definido una nueva interfaz representada por `os.PathLike`. Al implementar el método `__fspath__()`, un objeto indica que representa una ruta. Entonces, un objeto puede proporcionar una representación de bajo nivel de una ruta del sistema de archivos como un objeto `str` o `bytes`. Esto significa que un objeto se considera path-like si implementa `os.PathLike` o es un objeto `str` o `bytes` que representa una ruta del sistema de archivos. El código puede usar `os.fspath()`, `os.fsdecode()` o `os.fsencode()` para obtener explícitamente una representación **`str`** y / o **`bytes`** de un objeto similar a una ruta.

La función **`open()`** incorporada se ha actualizado para aceptar objetos **`os.PathLike`**, al igual que todas las funciones relevantes en los módulos **`os`** y **`os.path`**, y la mayoría de las otras funciones y clases en la biblioteca estándar. La clase **`os.DirEntry`** y las clases relevantes en **`pathlib`** también se han actualizado para implementar **`os.PathLike`**.

La esperanza es que la actualización de las funciones fundamentales para operar en las rutas del sistema de archivos conducirá a un código de terceros para admitir implícitamente todos los path-like objects sin ningún cambio de código, o al menos muy mínimos (por ejemplo, llamar a **`os.fspath()`** al comienzo del código antes de operar en un objeto similar a una ruta).

A continuación, se muestran algunos ejemplos de cómo la nueva interfaz permite que **`pathlib.Path`** se utilice de manera más fácil y transparente con código preexistente:

```
>>> import pathlib
>>> with open(pathlib.Path("README")) as f:
...     contents = f.read()
...
>>> import os.path
>>> os.path.splitext(pathlib.Path("some_file.txt"))
('some_file', '.txt')
>>> os.path.join("/a/b", pathlib.Path("c"))
'/a/b/c'
>>> import os
>>> os.fspath(pathlib.Path("some_file.txt"))
'some_file.txt'
```

(Implementado por Brett Cannon, Ethan Furman, Dusty Phillips y Jelle Zijlstra).

Ver también

PEP 519 [https://peps.python.org/pep-0519/]: adición de un

protocolo de ruta del sistema de archivos

PEP escrito por Brett Cannon y Koos Zevenhoven.

PEP 495: desambiguación de la hora local

En la mayoría de las ubicaciones del mundo, ha habido y habrá ocasiones en las que los relojes locales se retrasaron. En esos horarios, se introducen intervalos en los que los relojes locales marcan la misma hora dos veces en el mismo día. En estas situaciones, la información mostrada en un reloj local (o almacenada en una instancia de fecha y hora de Python) es insuficiente para identificar un momento particular en el tiempo.

PEP 495 [https://peps.python.org/pep-0495/] agrega el nuevo atributo *fold* a las instancias de las clases `datetime.datetime` y `datetime.time` para diferenciar entre dos momentos en el tiempo en los que las horas locales son las mismas:

```
>>> u0 = datetime(2016, 11, 6, 4, tzinfo=timezone.utc)
>>> for i in range(4):
...     u = u0 + i*HOUR
...     t = u.astimezone(Eastern)
...     print(u.time(), 'UTC =', t.time(), t.tzname(), t
...
04:00:00 UTC = 00:00:00 EDT 0
05:00:00 UTC = 01:00:00 EDT 0
06:00:00 UTC = 01:00:00 EST 1
07:00:00 UTC = 02:00:00 EST 0
```

Los valores del atributo `fold` tienen el valor `0` para todas las instancias excepto aquellas que representan el segundo momento (cronológicamente) en el tiempo en un caso ambiguo.

Ver también

PEP 495 [https://peps.python.org/pep-0495/] - Desambiguación de hora local

PEP escrito por Alexander Belopolsky y Tim Peters, implementación por Alexander Belopolsky.

PEP 529: cambie la codificación del sistema de archivos de Windows a UTF-8

La representación de las rutas del sistema de archivos se realiza mejor con str (Unicode) en lugar de bytes. Sin embargo, hay algunas situaciones en las que el uso de bytes es suficiente y correcto.

Antes de Python 3.6, se podían producir pérdidas de datos al utilizar rutas de bytes en Windows. Con este cambio, ahora se admite el uso de bytes para representar rutas en Windows, siempre que esos bytes estén codificados con la codificación devuelta por `sys.getfilesystemencoding()`, que ahora tiene como valor predeterminado `'utf-8'`.

Las aplicaciones que no utilizan str para representar rutas deben utilizar `os.fsencode()` y `os.fsdecode()` para garantizar que sus bytes estén codificados correctamente. Para volver al comportamiento anterior, configure `PYTHONLEGACYWINDOWSFSENCODING` o llame a `sys._enablelegacywindowsfsencoding()`.

Consulte PEP 529 [https://peps.python.org/pep-0529/] para obtener más información y una discusión sobre las modificaciones de código que pueden ser necesarias.

PEP 528: cambie la codificación de la consola de Windows a UTF-8

La consola predeterminada en Windows ahora aceptará todos los caracteres Unicode y proporcionará objetos str leídos correctamente al código Python. `sys.stdin`, `sys.stdout` y `sys.stderr` ahora tienen la codificación utf-8 por defecto.

Este cambio solo se aplica cuando se usa una consola interactiva y no cuando se redirigen archivos o canalizaciones. Para volver al comportamiento anterior para el uso de la consola interactiva, configure `PYTHONLEGACYWINDOWSSTDIO`.

Ver también

PEP 528 [https://peps.python.org/pep-0528/] : cambie la codificación de la consola de Windows a UTF-8
PEP escrito e implementado por Steve Dower.

PEP 520: Conservación del orden de definición de atributos de clase

Los atributos del cuerpo de una definición de clase tienen un orden natural: el mismo orden en el que aparecen los nombres en el código fuente. Este orden ahora se conserva en el atributo `__dict__` de la nueva clase.

Además, el espacio de nombres de clase *execution* predeterminado efectivo (devuelto desde type.__prepare__()) ahora es una asignación que conserva el orden de inserción.

Ver también

PEP 520 [https://peps.python.org/pep-0520/] - Conservación del orden de definición de atributos de clase
PEP escrito e implementado por Eric Snow.

PEP 468: Conservación del orden de los argumentos de las palabras clave

Ahora se garantiza que `**kwargs` en una firma de función es un mapeo que conserva el orden de inserción.

Ver también

PEP 468 [https://peps.python.org/pep-0468/] - Conservación del orden de los argumentos de las palabras clave
PEP escrito e implementado por Eric Snow.

Nueva implementación de dict

El tipo dict ahora usa una representación «compacta» basada en a proposal by Raymond Hettinger [https://mail.python.org/pipermail/python-dev/2012-December/123028.html] que era first implemented by PyPy [https://morepypy.blogspot.com/2015/01/faster-more-memory-efficient-and-more.html]. El uso de memoria del nuevo `dict()` es entre un 20% y un 25% menor en comparación con Python 3.5.

El aspecto de conservación del orden de esta nueva implementación se considera un detalle de implementación y no se debe confiar en él (esto puede cambiar en el futuro, pero se desea tener esta nueva implementación de dict en el idioma para algunas versiones antes de cambiar la especificación del idioma para exigir la semántica de preservación del orden para todas las implementaciones de Python actuales y futuras; esto también ayuda a preservar la compatibilidad con versiones anteriores del lenguaje donde el orden de iteración aleatorio todavía está vigente, por ejemplo, Python 3.5).

(Contribuido por INADA Naoki en bpo-27350 [https://bugs.python.org/issue?@action=redirect&bpo=27350]. Idea originally suggested by Raymond Hettinger [https://mail.python.org/pipermail/python-dev/2012-December/123028.html].)

PEP 523: Agregar una API de evaluación de marcos a CPython

Si bien Python proporciona un amplio soporte para personalizar cómo se ejecuta el código, un lugar en el que no lo ha hecho es en la evaluación de objetos de marco. Si quisiera alguna forma de interceptar la evaluación de marcos en Python, realmente no había ninguna forma sin manipular directamente los punteros de función para las funciones definidas.

PEP 523 [https://peps.python.org/pep-0523/] cambia esto al proporcionar una API para hacer que la evaluación de tramas se pueda conectar en el nivel C. Esto permitirá que herramientas como depuradores y JIT intercepten la evaluación del marco antes de que comience la ejecución del código Python. Esto permite el uso de implementaciones de evaluación alternativas para el código Python, seguimiento de la evaluación del marco, etc.

Esta API no forma parte de la API C limitada y está marcada como privada para indicar que se espera que el uso de esta API sea limitado y solo se aplique a casos de uso muy selectos y de bajo nivel. La semántica de la API cambiará con Python según sea necesario.

Ver también

PEP 523 [https://peps.python.org/pep-0523/] : agregar una API de evaluación de marcos a CPython

PEP escrito por Brett Cannon y Dino Viehland.

Variable de entorno PYTHONMALLOC

La nueva variable de entorno **`PYTHONMALLOC`** permite configurar los asignadores de memoria e instalar Python ganchos de depuración.

Ahora es posible instalar enlaces de depuración en asignadores de memoria de Python en Python compilado en modo de lanzamiento usando `PYTHONMALLOC=debug`. Efectos de los ganchos de depuración:

- La memoria recién asignada se llena con el byte `0xCB`
- La memoria liberada se llena con el byte `0xDB`
- Detecta violaciones de la API de asignación de memoria de Python. Por ejemplo, **`PyObject_Free()`** llamó a un bloque de memoria asignado por **`PyMem_Malloc()`**.
- Detectar escrituras antes del inicio de un búfer (subdesbordamientos del búfer)
- Detectar escrituras después del final de un búfer (desbordes de búfer)
- Verifique que GIL se mantenga cuando se llamen las funciones de asignador de los dominios **`PYMEM_DOMAIN_OBJ`** (por ejemplo: **`PyObject_Malloc()`**) y **`PYMEM_DOMAIN_MEM`** (por ejemplo: **`PyMem_Malloc()`**).

Verificar si el GIL se mantiene también es una nueva característica de Python 3.6.

Consulte la función **`PyMem_SetupDebugHooks()`** para ver los enlaces de depuración en los asignadores de memoria de Python.

Ahora también es posible forzar el uso del asignador **`malloc()`** de la biblioteca C para todas las asignaciones de memoria de Python usando `PYTHONMALLOC=malloc`. Esto es útil cuando se utilizan depuradores de memoria externa como Valgrind en un Python compilado en modo de lanzamiento.

En caso de error, los ganchos de depuración en los asignadores de memoria de Python ahora usan el módulo **`tracemalloc`** para obtener el rastreo donde se asignó un bloque de memoria.

Ejemplo de error fatal en el desbordamiento del búfer usando `python3.6 -X tracemalloc=5` (almacenar 5 cuadros en trazas):

```
Debug memory block at address p=0x7fbcd41666f8: API 'o'
    4 bytes originally requested
    The 7 pad bytes at p-7 are FORBIDDENBYTE, as expecte
    The 8 pad bytes at tail=0x7fbcd41666fc are not all F
        at tail+0: 0x02 *** OUCH
        at tail+1: 0xfb
        at tail+2: 0xfb
        at tail+3: 0xfb
        at tail+4: 0xfb
        at tail+5: 0xfb
        at tail+6: 0xfb
        at tail+7: 0xfb
    The block was made by call #1233329 to debug malloc/
    Data at p: 1a 2b 30 00

Memory block allocated at (most recent call first):
  File "test/test_bytes.py", line 323
  File "unittest/case.py", line 600
  File "unittest/case.py", line 648
  File "unittest/suite.py", line 122
  File "unittest/suite.py", line 84

Fatal Python error: bad trailing pad byte

Current thread 0x00007fbcdbd32700 (most recent call firs
  File "test/test_bytes.py", line 323 in test_hex
  File "unittest/case.py", line 600 in run
  File "unittest/case.py", line 648 in __call__
  File "unittest/suite.py", line 122 in run
  File "unittest/suite.py", line 84 in __call__
  File "unittest/suite.py", line 122 in run
  File "unittest/suite.py", line 84 in __call__
  ...
```

(Contribución de Victor Stinner en bpo-26516 [https://bugs.python.org/issue?@action=redirect&bpo=26516] y :issue:` 26564`.)

Soporte de sondeo DTrace y SystemTap

Python ahora se puede construir `--with-dtrace` que habilita marcadores estáticos para los siguientes eventos en el intérprete:

- función llamada / retorno
- recogida de basura iniciada / finalizada
- línea de código ejecutada.

Esto se puede utilizar para instrumentar intérpretes en ejecución en producción, sin la necesidad de volver a compilar debug builds específico o proporcionar código de depuración / perfilado específico de la aplicación.

Más detalles en Instrumentación de CPython con DTrace y SystemTap.

La implementación actual se prueba en Linux y macOS. Es posible que se agreguen marcadores adicionales en el futuro.

(Contribuido por Łukasz Langa en bpo-21590 [https://bugs.python.org/issue?@action=redirect&bpo=21590], basado en parches de Jesús Cea Avión, David Malcolm y Nikhil Benesch.)

Otros cambios de idioma

Algunos cambios más pequeños realizados en el lenguaje central de Python son:

- Una declaración `global` o `nonlocal` ahora debe aparecer textualmente antes del primer uso del nombre afectado en el mismo ámbito. Anteriormente, este era un **`SyntaxWarning`**.
- It is now possible to set a special method to `None` to indicate that the corresponding operation is not available. For example, if a class sets **`__iter__()`** to `None`, the class is not iterable. (Contributed by Andrew Barnert and Ivan Levkivskyi in bpo-25958 [https://bugs.python.org/issue?@action=redirect&bpo=25958].)
- Las secuencias largas de líneas de rastreo repetidas ahora se abrevian como `"[Previous line repeated {count} more times]"` (consulte rastrear para ver un ejemplo). (Contribución de Emanuel Barry en bpo-26823 [https://bugs.python.org/issue?@action=redirect&bpo=26823].)
- Importar ahora lanza la nueva excepción **`ModuleNotFoundError`** (subclase de **`ImportError`**) cuando no puede encontrar un módulo. El código que actualmente busca ImportError (en try-except) seguirá funcionando. (Contribución de Eric Snow en bpo-15767 [https://bugs.python.org/issue?@action=redirect&bpo=15767].)
- Los métodos de clase que se basan en `super()` de argumento cero ahora funcionarán correctamente cuando se llamen desde métodos de metaclase durante la creación de la clase. (Contribución de Martin Teichmann en bpo-23722 [https://bugs.python.org/issue?@action=redirect&bpo=23722].)

Nuevos módulos

misterios

El objetivo principal del nuevo módulo **`secrets`** es proporcionar una forma obvia de generar de manera confiable valores pseudoaleatorios criptográficamente fuertes adecuados para administrar secretos, como autenticación de cuentas, tokens y similares.

Advertencia

Tenga en cuenta que los generadores pseudoaleatorios del módulo `random` deben usarse con fines de seguridad. Use `secrets` en Python 3.6+ y `os.urandom()` en Python 3.5 y versiones anteriores.

Ver también

PEP 506 [https://peps.python.org/pep-0506/]: adición de un módulo de secretos a la biblioteca estándar

PEP escrito e implementado por Steven D'Aprano.

Módulos mejorados

formación

Los iteradores agotados de `array.array` ahora permanecerán agotados incluso si se extiende la matriz iterada. Esto es consistente con el comportamiento de otras secuencias mutables.

Contribuido por Serhiy Storchaka en bpo-26492 [https://bugs.python.org/issue?@action=redirect&bpo=26492].

ast

Se ha agregado el nuevo nodo `ast.Constant` AST. Puede ser utilizado por optimizadores AST externos con el propósito de plegado constante.

Contribuido por Victor Stinner en bpo-26146 [https://bugs.python.org/issue?@action=redirect&bpo=26146].

asyncio

A partir de Python 3.6, el módulo `asyncio` ya no es provisional y su API se considera estable.

Cambios notables en el módulo `asyncio` desde Python 3.5.0 (todos retroportados a 3.5.x debido al estado provisional):

- La función `get_event_loop()` se ha cambiado para devolver siempre el bucle que se está ejecutando actualmente cuando se llama desde corrutinas y devoluciones de llamada. (Contribuido por Yury Selivanov en bpo-28613 [https://bugs.python.org/issue?@action=redirect&bpo=28613].)
- La función `ensure_future()` y todas las funciones que la utilizan, como `loop.run_until_complete()`, ahora aceptan todo tipo de awaitable objects. (Contribuido por Yury Selivanov.)
- Nueva función `run_coroutine_threadsafe()` para enviar corrutinas a bucles de eventos de otros hilos. (Contribución de Vincent Michel.)
- Nuevo método `Transport.is_closing()` para comprobar si el transporte está cerrando o está cerrado. (Contribuido por Yury Selivanov.)
- El método `loop.create_server()` ahora puede aceptar una lista de hosts. (Contribuido por Yann Sionneau.)
- Nuevo método `loop.create_future()` para crear objetos Future. Esto permite implementaciones alternativas de bucle de eventos, como uvloop [https://github.com/MagicStack/uvloop], para proporcionar una implementación `asyncio.Future` más rápida. (Contribuido por Yury Selivanov en bpo-27041 [https://bugs.python.org/issue?@action=redirect&bpo=27041].)
- Nuevo método `loop.get_exception_handler()` para obtener el controlador de excepciones actual. (Contribuido por Yury Selivanov en bpo-27040 [https://bugs.python.org/issue?@action=redirect&bpo=27040].)
- Nuevo método `StreamReader.readuntil()` para leer datos de la secuencia hasta que aparezca una secuencia de bytes de separación. (Contribuido por Mark Korenberg.)
- Se ha mejorado el rendimiento de `StreamReader.readexactly()`. (Contribuido por Mark Korenberg en bpo-28370 [https://bugs.python.org/issue?@action=redirect&bpo=28370].)
- El método `loop.getaddrinfo()` está optimizado para evitar llamar a la función `getaddrinfo` del sistema si la dirección ya está resuelta. (Contribuido por A. Jesse Jiryu Davis.)
- El método `loop.stop()` se ha cambiado para detener el bucle inmediatamente después de la iteración actual. Se

descartarán todas las devoluciones de llamada nuevas programadas como resultado de la última iteración. (Contribuido por Guido van Rossum en bpo-25593 [https://bugs.python.org/issue?@action=redirect&bpo=25593].)

- `Future.set_exception` will now raise `TypeError` when passed an instance of the `StopIteration` exception. (Contributed by Chris Angelico in bpo-26221 [https://bugs.python.org/issue?@action=redirect&bpo=26221].)
- Nuevo método `loop.connect_accepted_socket()` para ser usado por servidores que aceptan conexiones fuera de asyncio, pero que usan asyncio para manejarlas. (Contribuido por Jim Fulton en bpo-27392 [https://bugs.python.org/issue?@action=redirect&bpo=27392].)
- El indicador `TCP_NODELAY` ahora está configurado para todos los transportes TCP de forma predeterminada. (Contribuido por Yury Selivanov en bpo-27456 [https://bugs.python.org/issue?@action=redirect&bpo=27456].)
- Nuevo `loop.shutdown_asyncgens()` para cerrar correctamente los generadores asincrónicos pendientes antes de cerrar el ciclo. (Contribuido por Yury Selivanov en bpo-28003 [https://bugs.python.org/issue?@action=redirect&bpo=28003].)
- Las clases `Future` y `Task` ahora tienen una implementación C optimizada que hace que el código asyncio sea hasta un 30% más rápido. (Contribuido por Yury Selivanov e INADA Naoki en bpo-26081 [https://bugs.python.org/issue?@action=redirect&bpo=26081] y bpo-28544 [https://bugs.python.org/issue?@action=redirect&bpo=28544].)

binascii

La función `b2a_base64()` ahora acepta un argumento de palabra clave *newline* opcional para controlar si el carácter de nueva línea se agrega al valor de retorno. (Contribuido por Victor Stinner en bpo-25357 [https://bugs.python.org/issue?@action=redirect&bpo=25357].)

cmath

Se ha agregado la nueva constante `cmath.tau` (τ). (Contribuido por Lisa Roach en bpo-12345 [https://bugs.python.org/issue?

@action=redirect&bpo=12345], consulte **PEP 628** [https://peps.python.org/pep-0628/] para obtener más detalles).

Nuevas constantes: `cmath.inf` y `cmath.nan` para que coincidan con `math.inf` y `math.nan`, y también `cmath.infj` y `cmath.nanj` para que coincidan con el formato utilizado por la repetición compleja. (Contribuido por Mark Dickinson en bpo-23229 [https://bugs.python.org/issue?@action=redirect&bpo=23229].)

colecciones

Se ha agregado la nueva clase base abstracta `Collection` para representar clases de contenedores iterables de tamaño. (Contribuido por Ivan Levkivskyi, documentos de Neil Girdhar en bpo-27598 [https://bugs.python.org/issue?@action=redirect&bpo=27598].)

The new `Reversible` abstract base class represents iterable classes that also provide the `__reversed__()` method. (Contributed by Ivan Levkivskyi in bpo-25987 [https://bugs.python.org/issue?@action=redirect&bpo=25987].)

La nueva clase base abstracta `AsyncGenerator` representa generadores asincrónicos. (Contribuido por Yury Selivanov en bpo-28720 [https://bugs.python.org/issue?@action=redirect&bpo=28720].)

La función `namedtuple()` ahora acepta un argumento de palabra clave opcional *module*, que, cuando se especifica, se utiliza para el atributo `__module__` de la clase de tupla nombrada que se devuelve. (Contribuido por Raymond Hettinger en bpo-17941 [https://bugs.python.org/issue?@action=redirect&bpo=17941]).

Los argumentos *verbose* and *rename* para `namedtuple()` ahora son solo palabras clave. (Contribuido por Raymond Hettinger en bpo-25628 [https://bugs.python.org/issue?@action=redirect&bpo=25628].)

Las instancias de `collections.deque` recursivas ahora se pueden eliminar. (Contribuido por Serhiy Storchaka en bpo-26482 [https://bugs.python.org/issue?@action=redirect&bpo=26482].)

concurrent.futures

El constructor de la clase `ThreadPoolExecutor` ahora acepta un argumento *thread_name_prefix* opcional para que sea posible personalizar los nombres de los subprocesos creados por el grupo. (Contribuido por Gregory P. Smith en bpo-27664 [https://bugs.python.org/issue?@action=redirect&bpo=27664].)

contextlib

La clase `contextlib.AbstractContextManager` fue agregada para proveer una clase abstracta para gestores de contexto. Provee una implementación por defecto sensible para *__enter__()* que retorna `self` y deja un método abstracto *__exit__()*. Se ha agregado una clase coincidente al módulo `typing` como `typing.ContextManager`. (Contribuido por Brett Cannon en bpo-25609 [https://bugs.python.org/issue?@action=redirect&bpo=25609].)

fecha y hora

The `datetime` and `time` classes have the new `fold` attribute used to disambiguate local time when necessary. Many functions in the `datetime` have been updated to support local time disambiguation. See Local Time Disambiguation section for more information. (Contributed by Alexander Belopolsky in bpo-24773 [https://bugs.python.org/issue?@action=redirect&bpo=24773].)

Los métodos `datetime.strftime()` y `date.strftime()` ahora admiten las directivas de fecha ISO 8601 `%G`, `%u` y `%V`. (Contribuido por Ashley Anderson en bpo-12006 [https://bugs.python.org/issue?@action=redirect&bpo=12006].)

La función `datetime.isoformat()` ahora acepta un argumento *timespec* opcional que especifica el número de componentes adicionales del valor de tiempo a incluir. (Contribuido por Alessandro Cucci y Alexander Belopolsky en bpo-19475 [https://bugs.python.org/issue?@action=redirect&bpo=19475].)

El `datetime.combine()` ahora acepta un argumento *tzinfo* opcional. (Contribuido por Alexander Belopolsky en bpo-27661 [https://bugs.python.org/issue?@action=redirect&bpo=27661].)

decimal

Nuevo método `Decimal.as_integer_ratio()` que devuelve un par `(n, d)` de enteros que representan la instancia `Decimal` dada como una fracción, en términos más bajos y con un denominador positivo:

```
>>> Decimal('-3.14').as_integer_ratio()
(-157, 50)
```

(Contribuido por Stefan Krah y Mark Dickinson en bpo-25928 [https://bugs.python.org/issue?@action=redirect&bpo=25928].)

distutils

El atributo `default_format` se ha eliminado de `distutils.command.sdist.sdist` y el atributo `formats` tiene como valor predeterminado `['gztar']`. Aunque no se prevé, es posible que sea necesario adaptar cualquier código que dependa de la presencia de `default_format`. Consulte bpo-27819 [https://bugs.python.org/issue?@action=redirect&bpo=27819] para obtener más detalles.

email

La nueva API de correo electrónico, habilitada a través de la palabra clave *policy* para varios constructores, ya no es provisional. La documentación de `email` se ha reorganizado y reescrito para centrarse en la nueva API, conservando la documentación anterior para la API heredada. (Contribuido por R. David Murray en bpo-24277 [https://bugs.python.org/issue?@action=redirect&bpo=24277].)

Las clases `email.mime` ahora aceptan todas una palabra clave *policy* opcional. (Contribuido por Berker Peksag en bpo-27331 [https://bugs.python.org/issue?@action=redirect&bpo=27331].)

El `DecodedGenerator` ahora admite la palabra clave *policy*.

Hay un nuevo atributo `policy`, `message_factory`, que controla qué clase se usa por defecto cuando el analizador crea nuevos objetos de mensaje. Para la política **`email.policy.compat32`**, es **`Message`**, para las nuevas políticas es **`EmailMessage`**. (Contribuido por R. David Murray en bpo-20476 [https://bugs.python.org/issue?@action=redirect&bpo=20476].)

codificaciones

En Windows, se agregó la codificación `'oem'` para usar `CP_OEMCP` y el alias `'ansi'` para la codificación `'mbcs'` existente, que usa la página de códigos `CP_ACP`. (Contribuido por Steve Dower en bpo-27959 [https://bugs.python.org/issue?@action=redirect&bpo=27959].)

enumeración

Two new enumeration base classes have been added to the **`enum`** module: **`Flag`** and **`IntFlag`**. Both are used to define constants that can be combined using the bitwise operators. (Contributed by Ethan Furman in bpo-23591 [https://bugs.python.org/issue?@action=redirect&bpo=23591].)

Many standard library modules have been updated to use the **`IntFlag`** class for their constants.

El nuevo valor de **`enum.auto`** se puede utilizar para asignar valores a los miembros de enumeración automáticamente:

```
>>> from enum import Enum, auto
>>> class Color(Enum):
...     red = auto()
...     blue = auto()
...     green = auto()
...
>>> list(Color)
[<Color.red: 1>, <Color.blue: 2>, <Color.green: 3>]
```

manipulador de faltas

En Windows, el módulo **`faulthandler`** ahora instala un controlador para las excepciones de Windows: consulte `faulthandler.enable()`. (Contribuido por Victor Stinner en bpo-23848 [https://bugs.python.org/issue?@action=redirect&bpo=23848].)

entrada de archivo

`hook_encoded()` ahora admite el argumento *errors*. (Contribuido por Joseph Hackman en bpo-25788 [https://bugs.python.org/issue?@action=redirect&bpo=25788].)

hashlib

`hashlib` es compatible con OpenSSL 1.1.0. La versión mínima recomendada es 1.0.2. (Contribuido por Christian Heimes en bpo-26470 [https://bugs.python.org/issue?@action=redirect&bpo=26470].)

Se agregaron funciones hash BLAKE2 al módulo. `blake2b()` y `blake2s()` siempre están disponibles y admiten el conjunto completo de funciones de BLAKE2. (Contribuido por Christian Heimes en bpo-26798 [https://bugs.python.org/issue?@action=redirect&bpo=26798] basado en código de Dmitry Chestnykh y Samuel Neves. Documentación escrita por Dmitry Chestnykh.)

Se agregaron las funciones hash SHA-3 `sha3_224()`, `sha3_256()`, `sha3_384()`, `sha3_512()` y las funciones hash SHAKE `shake_128()` y `shake_256()`. (Contribuido por Christian Heimes en bpo-16113 [https://bugs.python.org/issue?@action=redirect&bpo=16113]. Paquete de código Keccak por Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche y Ronny Van Keer.)

La función de derivación de claves basada en contraseña `scrypt()` ahora está disponible con OpenSSL 1.1.0 y versiones posteriores. (Contribuido por Christian Heimes en bpo-27928 [https://bugs.python.org/issue?@action=redirect&bpo=27928].)

http.client

`HTTPConnection.request()` y `endheaders()` ahora admiten cuerpos de solicitud de codificación fragmentados. (Contribuido por Demian Brecht y Rolf Krahl en bpo-12319 [https://bugs.python.org/issue?

@action=redirect&bpo=12319].)

idlelib y IDLE

El paquete idlelib se está modernizando y refactorizando para que IDLE se vea y funcione mejor y para que el código sea más fácil de entender, probar y mejorar. Parte de hacer que IDLE se vea mejor, especialmente en Linux y Mac, es usar widgets ttk, principalmente en los cuadros de diálogo. Como resultado, IDLE ya no se ejecuta con tcl / tk 8.4. Ahora requiere tcl / tk 8.5 o 8.6. Recomendamos ejecutar la última versión de cualquiera.

La "modernización" incluye el cambio de nombre y la consolidación de módulos idlelib. El cambio de nombre de archivos con nombres en mayúsculas parciales es similar al cambio de nombre de, por ejemplo, Tkinter y TkFont a tkinter y tkinter.font en 3.0. Como resultado, las importaciones de archivos idlelib que funcionaban en 3.5 normalmente no funcionarán en 3.6. Se necesitará al menos un cambio de nombre de módulo (ver idlelib / README.txt), a veces más. (Los cambios de nombre aportados por Al Swiegart y Terry Reedy en bpo-24225 [https://bugs.python.org/issue?@action=redirect&bpo=24225]. La mayoría de los parches idlelib desde entonces han sido y serán parte del proceso).

En compensación, el resultado final será que algunas clases de idlelib serán más fáciles de usar, con mejores API y cadenas de documentación que las expliquen. Se agregará información útil adicional a idlelib cuando esté disponible.

Nuevo en 3.6.2:

Varias correcciones para autocompletar. (Contribuido por Louie Lu en bpo-15786 [https://bugs.python.org/issue?@action=redirect&bpo=15786].)

Nuevo en 3.6.3:

El Explorador de módulos (en el menú Archivo, antes llamado Explorador de clases), ahora muestra funciones y clases anidadas además de funciones y clases de nivel superior. (Contribuido por Guilherme Polo, Cheryl Sabella y Terry Jan Reedy en bpo-1612262 [https://bugs.python.org/issue?@action=redirect&bpo=1612262].)

Las funciones IDLE implementadas anteriormente como extensiones se han vuelto a implementar como funciones normales. Su configuración se ha movido de la pestaña Extensiones a otras pestañas de diálogo. (Contribuido por Charles Wohlganger y Terry Jan Reedy en bpo-27099 [https://bugs.python.org/issue?@action=redirect&bpo=27099].)

El cuadro de diálogo Configuración (Opciones, Configurar IDLE) se ha reescrito parcialmente para mejorar tanto la apariencia como la función. (Contribuido por Cheryl Sabella y Terry Jan Reedy en varios números).

Nuevo en 3.6.4:

La muestra de fuente ahora incluye una selección de caracteres no latinos para que los usuarios puedan ver mejor el efecto de seleccionar una fuente en particular. (Contribuido por Terry Jan Reedy en bpo-13802 [https://bugs.python.org/issue?@action=redirect&bpo=13802].) La muestra se puede editar para incluir otros caracteres. (Contribuido por Serhiy Storchaka en bpo-31860 [https://bugs.python.org/issue?@action=redirect&bpo=31860].)

Nuevo en 3.6.6:

Se revisó la opción de contexto del código del editor. El cuadro muestra todas las líneas de contexto hasta las líneas máximas. Al hacer clic en una línea de contexto, el editor salta a esa línea. Los colores de contexto para temas personalizados se agregan a la pestaña Destacados del cuadro de diálogo Configuración. (Contribuido por Cheryl Sabella y Terry Jan Reedy en bpo-33642 [https://bugs.python.org/issue?@action=redirect&bpo=33642], bpo-33768 [https://bugs.python.org/issue?@action=redirect&bpo=33768] y bpo-33679 [https://bugs.python.org/issue?@action=redirect&bpo=33679].)

En Windows, una nueva llamada a la API le dice a Windows que tk escala para DPI. En Windows 8.1+ o 10, con las propiedades de compatibilidad de DPI del binario de Python sin cambios y una resolución de monitor superior a 96 DPI, esto debería hacer que el texto y las líneas sean más nítidos. De lo contrario, no debería tener ningún efecto. (Contribuido por Terry Jan Reedy en bpo-33656 [https://bugs.python.org/issue?@action=redirect&bpo=33656].)

Nuevo en 3.6.7:

La salida sobre N líneas (50 por defecto) se reduce a un botón. N se puede cambiar en la sección PyShell de la página General del cuadro de diálogo Configuración. Se pueden comprimir menos líneas, pero posiblemente más largas, haciendo clic derecho en la salida. La salida comprimida se puede expandir en su lugar haciendo doble clic en el botón o en el portapapeles o en una ventana separada haciendo clic derecho en el botón. (Contribuido por Tal Einat en bpo-1529353 [https://bugs.python.org/issue?@action=redirect&bpo=1529353].)

importlib

Importar ahora lanza la nueva excepción `ModuleNotFoundError` (subclase de `ImportError`) cuando no puede encontrar un módulo. Codifique que las comprobaciones actuales para `ImportError` (en try-except) seguirán funcionando. (Contribuido por Eric Snow en bpo-15767 [https://bugs.python.org/issue?@action=redirect&bpo=15767].)

`importlib.util.LazyLoader` ahora llama a `create_module()` en el cargador envuelto, eliminando la restricción de que `importlib.machinery.BuiltinImporter` y `importlib.machinery.ExtensionFileLoader` no se pueden usar con **`importlib.util.LazyLoader`**.

`importlib.util.cache_from_source()`, `importlib.util.source_from_cache()` y `importlib.util.spec_from_file_location()` ahora aceptan un path-like object.

inspeccionar

La función `inspect.signature()` ahora reporta los parámetros `.0` implícitos generados por el compilador para los alcances de comprensión y expresión del generador como si fueran parámetros solo posicionales llamados `implicit0`. (Contribuido por Jelle Zijlstra en bpo-19611 [https://bugs.python.org/issue?@action=redirect&bpo=19611].)

To reduce code churn when upgrading from Python 2.7 and the legacy `inspect.getargspec()` API, the previously documented deprecation of `inspect.getfullargspec()` has been reversed. While this function is convenient for single/source Python 2/3 code bases, the richer `inspect.signature()` interface remains the recommended approach for new code. (Contributed by Nick Coghlan in bpo-27172 [https://bugs.python.org/issue?@action=redirect&bpo=27172])

json

`json.load()` y `json.loads()` ahora admiten entrada binaria. El JSON codificado debe representarse mediante UTF-8, UTF-16 o UTF-32. (Contribuido por Serhiy Storchaka en bpo-17909 [https://bugs.python.org/issue?@action=redirect&bpo=17909].)

logging

Se agregó el nuevo método `WatchedFileHandler.reopenIfNeeded()` para agregar la capacidad de verificar si es necesario volver a abrir el archivo de registro. (Contribuido por Marian Horban en bpo-24884 [https://bugs.python.org/issue?@action=redirect&bpo=24884].)

math

La constante tau (τ) se ha agregado a los módulos `math` y `cmath`. (Contribuido por Lisa Roach en bpo-12345 [https://bugs.python.org/issue?@action=redirect&bpo=12345], consulte **PEP 628** [https://peps.python.org/pep-0628/] para obtener más detalles).

multiprocesamiento

Proxy Objects devuelto por `multiprocessing.Manager()` ahora se puede anidar. (Contribuido por Davin Potts en bpo-6766 [https://bugs.python.org/issue?@action=redirect&bpo=6766].)

os

Consulte el resumen de PEP 519 para obtener detalles sobre cómo los módulos `os` y `os.path` ahora admiten path-like objects.

`scandir()` ahora admite rutas `bytes` en Windows.

A new `close()` method allows explicitly closing a `scandir()` iterator. The `scandir()` iterator now supports the context manager protocol. If a `scandir()` iterator is neither exhausted nor explicitly closed a `ResourceWarning` will be emitted in its destructor. (Contributed by Serhiy Storchaka in bpo-25994 [https://bugs.python.org/issue?@action=redirect&bpo=25994].)

En Linux, `os.urandom()` ahora se bloquea hasta que se inicializa el grupo de entropía aleatoria del sistema para aumentar la seguridad. Consulte el **PEP 524** [https://peps.python.org/pep-0524/] para conocer la justificación.

El syscall de Linux `getrandom()` (obtener bytes aleatorios) ahora se expone como la nueva función `os.getrandom()`. (Contribuido por Victor Stinner, parte del **PEP 524** [https://peps.python.org/pep-0524/])

pathlib

`pathlib` ahora es compatible con path-like objects. (Contribuido por Brett Cannon en bpo-27186 [https://bugs.python.org/issue?@action=redirect&bpo=27186].)

Consulte el resumen de PEP 519 para obtener más detalles.

pdb

El constructor de la clase `Pdb` tiene un nuevo argumento *readrc* opcional para controlar si se deben leer los archivos `.pdbrc`.

pepinillo

Los objetos que necesitan que se llame a `__new__` con argumentos de palabras clave ahora se pueden seleccionar utilizando pickle protocols anterior a la versión del protocolo 4. La versión 4 del protocolo ya es compatible con este caso. (Contribuido por Serhiy Storchaka en bpo-24164 [https://bugs.python.org/issue?@action=redirect&bpo=24164].)

pepinillos

`pickletools.dis()` ahora genera el índice de memo implícito para el código de operación `MEMOIZE`. (Contribuido por Serhiy Storchaka en bpo-25382 [https://bugs.python.org/issue?@action=redirect&bpo=25382]).

pydoc

El módulo **`pydoc`** ha aprendido a respetar la variable de entorno `MANPAGER`. (Contribuido por Matthias Klose en bpo-8637 [https://bugs.python.org/issue?@action=redirect&bpo=8637].)

`help()` y **`pydoc`** ahora pueden enumerar campos de tupla con nombre en el orden en que se definieron en lugar de alfabéticamente. (Contribuido por Raymond Hettinger en bpo-24879 [https://bugs.python.org/issue?@action=redirect&bpo=24879].)

aleatorio

La nueva función **`choices()`** devuelve una lista de elementos de tamaño especificado de la población dada con pesos opcionales. (Contribuido por Raymond Hettinger en bpo-18844 [https://bugs.python.org/issue?@action=redirect&bpo=18844].)

re

Se agregó soporte para lapsos de modificadores en expresiones regulares. Ejemplos: `'(?i:p)ython'` coincide con `'python'` y `'Python'`, pero no con `'PYTHON'`; `'(?i)g(?-i:v)r'` coincide con `'GvR'` y `'gvr'`, pero no con `'GVR'`. (Contribuido por Serhiy Storchaka en bpo-433028 [https://bugs.python.org/issue?@action=redirect&bpo=433028].)

Se puede acceder a los grupos de objetos coincidentes mediante `__getitem__`, que es equivalente a `group()`. Entonces `mo['name']` ahora es equivalente a `mo.group('name')`. (Contribuido por Eric Smith en bpo-24454 [https://bugs.python.org/issue?@action=redirect&bpo=24454].)

Los objetos **`Match`** ahora admiten **`index-like objects`** como índices de grupo. (Contribuido por Jeroen Demeyer y Xiang Zhang en bpo-27177 [https://bugs.python.org/issue?@action=redirect&bpo=27177].)

readline

Se agregó **`set_auto_history()`** para habilitar o deshabilitar la adición automática de entradas a la lista del historial. (Contribuido por Tyler Crompton en bpo-26870 [https://bugs.python.org/issue?@action=redirect&bpo=26870].)

rlcompleter

Los nombres de atributos privados y especiales ahora se omiten a menos que el prefijo comience con guiones bajos. Se agrega un espacio o dos puntos después de algunas palabras clave completadas. (Contribuido por Serhiy Storchaka en bpo-25011 [https://bugs.python.org/issue?@action=redirect&bpo=25011] y bpo-25209 [https://bugs.python.org/issue?@action=redirect&bpo=25209].)

shlex

El **`shlex`** tiene mucho improved shell compatibility a través del nuevo argumento *punctuation_chars* para controlar qué caracteres se tratan como puntuación. (Contribuido por Vinay Sajip en bpo-1521950 [https://bugs.python.org/issue?@action=redirect&bpo=1521950].)

sitio

Al especificar rutas para agregar a **`sys.path`** en un archivo `.pth`, ahora puede especificar rutas de archivos sobre directorios (por ejemplo, archivos zip). (Contribuido por Wolfgang Langner en bpo-26587 [https://bugs.python.org/issue?@action=redirect&bpo=26587]).

sqlite3

`sqlite3.Cursor.lastrowid` ahora admite la declaración `REPLACE`. (Contribuido por Alex LordThorsen en bpo-16864 [https://bugs.python.org/issue?@action=redirect&bpo=16864].)

enchufe

La función **`ioctl()`** ahora admite el código de control **`SIO_LOOPBACK_FAST_PATH`**. (Contribución de Daniel Stokes en bpo-26536 [https://bugs.python.org/issue?@action=redirect&bpo=26536]).

Las constantes **`getsockopt()`** `SO_DOMAIN`, `SO_PROTOCOL`, `SO_PEERSEC` y `SO_PASSSEC` ahora son compatibles. (Contribuido por Christian Heimes en bpo-26907 [https://bugs.python.org/issue?@action=redirect&bpo=26907].)

El **`setsockopt()`** ahora admite el formulario `setsockopt(level, optname, None, optlen: int)`. (Contribuido por Christian Heimes en bpo-27744 [https://bugs.python.org/issue?@action=redirect&bpo=27744].)

El módulo de socket ahora admite la familia de direcciones **`AF_ALG`** para interactuar con la API de cifrado del kernel de Linux. Se agregaron `ALG_*`, `SOL_ALG` y **`sendmsg_afalg()`**. (Contribución de Christian Heimes en bpo-27744 [https://bugs.python.org/issue?@action=redirect&bpo=27744] con el apoyo de Victor Stinner).

Se agregaron las nuevas constantes de Linux `TCP_USER_TIMEOUT` y `TCP_CONGESTION`. (Contribuido por Omar Sandoval, bpo-26273 [https://bugs.python.org/issue?@action=redirect&bpo=26273]).

servidor de sockets

Los servidores basados en el módulo **`socketserver`**, incluidos los definidos en **`http.server`**, **`xmlrpc.server`** y **`wsgiref.simple_server`**, ahora admiten el protocolo context manager. (Contribuido por Aviv Palivoda en bpo-26404 [https://bugs.python.org/issue?@action=redirect&bpo=26404].)

The `wfile` attribute of `StreamRequestHandler` classes now implements the `io.BufferedIOBase` writable interface. In particular, calling `write()` is now guaranteed to send the data in full. (Contributed by Martin Panter in bpo-26721 [https://bugs.python.org/issue?@action=redirect&bpo=26721].)

ssl

`ssl` es compatible con OpenSSL 1.1.0. La versión mínima recomendada es 1.0.2. (Contribuido por Christian Heimes en bpo-26470 [https://bugs.python.org/issue?@action=redirect&bpo=26470].)

Se ha eliminado 3DES de los conjuntos de cifrado predeterminados y se han agregado conjuntos de cifrado ChaCha20 Poly1305. (Contribuido por Christian Heimes en bpo-27850 [https://bugs.python.org/issue?@action=redirect&bpo=27850] y bpo-27766 [https://bugs.python.org/issue?@action=redirect&bpo=27766].)

`SSLContext` tiene una mejor configuración predeterminada para opciones y cifrados. (Contribuido por Christian Heimes en bpo-28043 [https://bugs.python.org/issue?@action=redirect&bpo=28043].)

La sesión SSL se puede copiar de una conexión del lado del cliente a otra con la nueva clase `SSLSession`. La reanudación de la sesión de TLS puede acelerar el apretón de manos inicial, reducir la latencia y mejorar el rendimiento (Contribuido por Christian Heimes en bpo-19500 [https://bugs.python.org/issue?@action=redirect&bpo=19500] basado en un borrador de Alex Warhawk).

El nuevo método `get_ciphers()` se puede utilizar para obtener una lista de cifrados habilitados en orden de prioridad de cifrado.

All constants and flags have been converted to `IntEnum` and `IntFlag`. (Contributed by Christian Heimes in bpo-28025 [https://bugs.python.org/issue?@action=redirect&bpo=28025].)

Se agregaron protocolos TLS específicos del lado del servidor y del cliente para `SSLContext`. (Contribuido por Christian Heimes en bpo-28085 [https://bugs.python.org/issue?@action=redirect&bpo=28085].)

Se agregó `ssl.SSLContext.post_handshake_auth` para habilitar y `ssl.SSLSocket.verify_client_post_handshake()` para iniciar la autenticación posterior al protocolo de enlace TLS 1.3. (Contribuido por Christian Heimes en gh-78851 [https://github.com/python/cpython/issues/78851].)

Estadísticas

Se ha agregado una nueva función `harmonic_mean()`. (Contribuido por Steven D'Aprano en bpo-27181 [https://bugs.python.org/issue?@action=redirect&bpo=27181].)

estructura

`struct` ahora admite flotadores de media precisión IEEE 754 a través del especificador de formato `'e'`. (Contribuido por Eli Stevens, Mark Dickinson en bpo-11734 [https://bugs.python.org/issue?@action=redirect&bpo=11734].)

subproceso

El destructor `subprocess.Popen` ahora emite una advertencia `ResourceWarning` si el proceso hijo todavía se está ejecutando. Utilice el protocolo del administrador de contexto (`with proc: ...`) o llame explícitamente al método `wait()` para leer el estado de salida del proceso hijo. (Contribuido por Victor Stinner en bpo-26741 [https://bugs.python.org/issue?@action=redirect&bpo=26741].)

El constructor `subprocess.Popen` y todas las funciones que le pasan argumentos ahora aceptan argumentos *encoding* y *errors*. Especificando alguno de ellos activará el modo texto para *stdin*, *stdout* y *stderr*. (Contribuido por Steve Dower en bpo-6135 [https://bugs.python.org/issue?@action=redirect&bpo=6135].)

sys

La nueva función `getfilesystemencodeerrors()` devuelve el nombre del modo de error utilizado para convertir entre nombres de archivo Unicode y nombres de archivo en bytes. (Contribuido por Steve Dower en bpo-27781 [https://bugs.python.org/issue?@action=redirect&bpo=27781].)

En Windows, el valor de retorno de la función `getwindowsversion()` ahora incluye el campo *platform_version* que contiene la versión principal precisa, la versión secundaria y el número de compilación del sistema operativo actual, en lugar de la versión que se está emulando para el proceso (contribución de Steve Dower en bpo-27932 [https://bugs.python.org/issue?@action=redirect&bpo=27932] .)

telnetlib

`telnetlib.Telnet` es ahora un administrador de contexto (contribución de Stéphane Wirtel en bpo-25485 [https://bugs.python.org/issue?@action=redirect&bpo=25485]).

tiempo

The `struct_time` attributes **`tm_gmtoff`** and **`tm_zone`** are now available on all platforms.

cronométralo

Se ha agregado el nuevo método de conveniencia `Timer.autorange()` para llamar a `Timer.timeit()` repetidamente para que el tiempo de ejecución total sea mayor o igual a 200 milisegundos. (Contribuido por Steven D'Aprano en bpo-6422 [https://bugs.python.org/issue?@action=redirect&bpo=6422].)

`timeit` ahora advierte cuando hay una variación sustancial (4x) entre el mejor y el peor momento. (Contribuido por Serhiy Storchaka en bpo-23552 [https://bugs.python.org/issue?@action=redirect&bpo=23552].)

tkinter

Added methods **`Variable.trace_add()`**, **`Variable.trace_remove()`** and **`trace_info()`** in the **`tkinter.Variable`** class. They replace old methods

`trace_variable()`, **`trace()`**, **`trace_vdelete()`** and **`trace_vinfo()`** that use obsolete Tcl commands and might not work in future versions of Tcl. (Contributed by Serhiy Storchaka in bpo-22115 [https://bugs.python.org/issue?@action=redirect&bpo=22115]).

rastrear

Tanto el módulo de rastreo como la pantalla de excepción incorporada del intérprete ahora abrevian largas secuencias de líneas repetidas en los rastreos como se muestra en el siguiente ejemplo:

```
>>> def f(): f()
...
>>> f()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 1, in f
  File "<stdin>", line 1, in f
  File "<stdin>", line 1, in f
  [Previous line repeated 995 more times]
RecursionError: maximum recursion depth exceeded
```

(Contribuido por Emanuel Barry en bpo-26823 [https://bugs.python.org/issue?@action=redirect&bpo=26823].)

tracemalloc

El módulo **`tracemalloc`** ahora admite el seguimiento de asignaciones de memoria en varios espacios de direcciones diferentes.

Se ha agregado la nueva clase de filtro **`DomainFilter`** para filtrar los rastros de bloques por su espacio de direcciones (dominio).

(Contribuido por Victor Stinner en bpo-26588 [https://bugs.python.org/issue?@action=redirect&bpo=26588].)

mecanografía

Dado que el módulo **`typing`** es provisional, todos los cambios introducidos en Python 3.6 también se han actualizado a Python 3.5.x.

El módulo **`typing`** tiene un soporte mucho mejor para los alias de tipo genérico. Por ejemplo, `Dict[str, Tuple[S, T]]` ahora es una anotación de tipo válida. (Contribuido por Guido van Rossum en Github #195 [https://github.com/python/typing/pull/195].)

Se ha agregado la clase **`typing.ContextManager`** para representar **`contextlib.AbstractContextManager`**. (Contribuido por Brett Cannon en bpo-25609 [https://bugs.python.org/issue?@action=redirect&bpo=25609].)

Se ha agregado la clase **`typing.Collection`** para representar **`collections.abc.Collection`**. (Contribuido por Ivan Levkivskyi en bpo-27598 [https://bugs.python.org/issue?@action=redirect&bpo=27598].)

Se ha agregado la construcción de tipo **`typing.ClassVar`** para marcar las variables de clase. Como se introdujo en **PEP 526** [https://peps.python.org/pep-0526/], una anotación de variable envuelta en ClassVar indica que un atributo dado está destinado a ser utilizado como una variable de clase y no debe establecerse en instancias de esa clase. (Contribuido por Ivan Levkivskyi en Github #280 [https://github.com/python/typing/pull/280].)

Una nueva constante **`TYPE_CHECKING`** que se supone que es `True` por los verificadores de tipo estático, pero es `False` en tiempo de ejecución. (Contribuido por Guido van Rossum en Github #230 [https://github.com/python/typing/issues/230].)

Se ha agregado una nueva función auxiliar **`NewType()`** para crear tipos distintos ligeros para anotaciones:

```
from typing import NewType

UserId = NewType('UserId', int)
some_id = UserId(524313)
```

El verificador de tipo estático tratará el nuevo tipo como si fuera una subclase del tipo original. (Contribuido por Ivan Levkivskyi en Github #189 [https://github.com/python/typing/issues/189].)

unicodedata

El módulo **`unicodedata`** ahora usa datos de Unicode 9.0.0 [http://unicode.org/versions/Unicode9.0.0/]. (Contribuido por Benjamin Peterson).

unittest.mock

La clase **`Mock`** tiene las siguientes mejoras:

- Dos nuevos métodos, **`Mock.assert_called()`** y **`Mock.assert_called_once()`** para verificar si se llamó al objeto simulado. (Contribuido por Amit Saha en bpo-26323 [https://bugs.python.org/issue?@action=redirect&bpo=26323].)
- El método **`Mock.reset_mock()`** ahora tiene dos argumentos opcionales de solo palabras clave: *return_value* and *side_effect*. (Contribuido por Kushal Das en bpo-21271 [https://bugs.python.org/issue?@action=redirect&bpo=21271].)

urllib.request

If a HTTP request has a file or iterable body (other than a bytes object) but no `Content-Length` header, rather than throwing an error, **`AbstractHTTPHandler`** now falls back to use chunked transfer encoding. (Contributed by Demian Brecht and Rolf Krahl in bpo-12319 [https://bugs.python.org/issue?@action=redirect&bpo=12319].)

urllib.robotparser

`RobotFileParser` ahora admite las extensiones `Crawl-delay` y `Request-rate`. (Contribuido por Nikolay Bogoychev en bpo-16099 [https://bugs.python.org/issue?@action=redirect&bpo=16099].)

venv

`venv` acepta un nuevo parámetro `--prompt`. Este parámetro proporciona un prefijo alternativo para el entorno virtual.

(Propuesto por Łukasz Balcerzak y portado a 3.6 por Stéphane Wirtel en bpo-22829 [https://bugs.python.org/issue?@action=redirect&bpo=22829].)

advertencias

A new optional *source* parameter has been added to the `warnings.warn_explicit()` function: the destroyed object which emitted a `ResourceWarning`. A *source* attribute has also been added to `warnings.WarningMessage` (contributed by Victor Stinner in bpo-26568 [https://bugs.python.org/issue?@action=redirect&bpo=26568] and bpo-26567 [https://bugs.python.org/issue?@action=redirect&bpo=26567]).

Cuando se registra una advertencia `ResourceWarning`, el módulo `tracemalloc` ahora se usa para intentar recuperar el rastreo donde se asignó el objeto destruido.

Ejemplo con el script `example.py`:

```
import warnings

def func():
    return open(__file__)

f = func()
f = None
```

Salida del comando `python3.6 -Wd -X tracemalloc=5 example.py`:

```
example.py:7: ResourceWarning: unclosed file <_io.TextIO
  f = None
Object allocated at (most recent call first):
  File "example.py", lineno 4
    return open(__file__)
  File "example.py", lineno 6
    f = func()
```

El rastreo «Objeto asignado en» es nuevo y solo se muestra si `tracemalloc` está rastreando asignaciones de memoria de Python y si el módulo `warnings` ya fue importado.

winreg

Se agregó el tipo entero de 64 bits `REG_QWORD`. (Contribuido por Clement Rouault en bpo-23026 [https://bugs.python.org/issue?@action=redirect&bpo=23026].)

winsonido

Se permite que los argumentos de palabras clave se pasen a `Beep`, `MessageBeep` y `PlaySound` (bpo-27982 [https://bugs.python.org/issue?@action=redirect&bpo=27982]).

xmlrpc.client

El módulo `xmlrpc.client` ahora admite la descomposición de tipos de datos adicionales utilizados por la implementación Apache XML-RPC para números y `None`. (Contribuido por Serhiy Storchaka en bpo-26885 [https://bugs.python.org/issue?@action=redirect&bpo=26885].)

archivo zip

Un nuevo método de clase `ZipInfo.from_file()` permite crear una instancia `ZipInfo` a partir de un archivo del sistema de archivos. Se puede usar un nuevo método `ZipInfo.is_dir()` para verificar si la instancia `ZipInfo` representa un directorio. (Contribuido por Thomas Kluyver en bpo-26039 [https://bugs.python.org/issue?@action=redirect&bpo=26039].)

El método `ZipFile.open()` ahora se puede utilizar para escribir datos en un archivo ZIP, así como para extraer datos. (Contribuido por Thomas Kluyver en bpo-26039 [https://bugs.python.org/issue?@action=redirect&bpo=26039].)

zlib

Las funciones `compress()` y `decompress()` ahora aceptan argumentos de palabras clave. (Contribuido por Aviv Palivoda en bpo-26243 [https://bugs.python.org/issue?@action=redirect&bpo=26243] y Xiang Zhang en bpo-16764 [https://bugs.python.org/issue?@action=redirect&bpo=16764] respectivamente).

Optimizaciones

- El intérprete de Python ahora usa un código de palabra de 16 bits en lugar de un código de bytes, lo que hizo posible una serie de optimizaciones de código de operación. (Contribuido por Demur Rumed con aportes y reseñas de Serhiy Storchaka y Victor Stinner en bpo-26647 [https://bugs.python.org/issue?@action=redirect&bpo=26647] y bpo-28050 [https://bugs.python.org/issue?@action=redirect&bpo=28050].)
- La clase **`asyncio.Future`** ahora tiene una implementación C optimizada. (Contribuido por Yury Selivanov e INADA Naoki en bpo-26081 [https://bugs.python.org/issue?@action=redirect&bpo=26081].)
- La clase **`asyncio.Task`** ahora tiene una implementación C optimizada. (Contribuido por Yury Selivanov en bpo-28544 [https://bugs.python.org/issue?@action=redirect&bpo=28544].)
- Varias mejoras de implementación en el módulo **`typing`** (como el almacenamiento en caché de tipos genéricos) permiten mejoras de rendimiento hasta 30 veces mayores y una huella de memoria reducida.
- El decodificador ASCII ahora es hasta 60 veces más rápido para los controladores de errores `surrogateescape`, `ignore` y `replace` (Contribuido por Victor Stinner en bpo-24870 [https://bugs.python.org/issue?@action=redirect&bpo=24870]).
- Los codificadores ASCII y Latin1 ahora son hasta 3 veces más rápidos para el controlador de errores `surrogateescape` (Contribuido por Victor Stinner en bpo-25227 [https://bugs.python.org/issue?@action=redirect&bpo=25227]).
- El codificador UTF-8 ahora es hasta 75 veces más rápido para los controladores de errores `ignore`, `replace`, `surrogateescape`, `surrogatepass` (Contribuido por Victor Stinner en bpo-25267 [https://bugs.python.org/issue?@action=redirect&bpo=25267]).
- El decodificador UTF-8 ahora es hasta 15 veces más rápido

para los controladores de errores `ignore`, `replace` y `surrogateescape` (Contribuido por Victor Stinner en bpo-25301 [https://bugs.python.org/issue?@action=redirect&bpo=25301]).

- `bytes % args` ahora es hasta 2 veces más rápido. (Contribuido por Victor Stinner en bpo-25349 [https://bugs.python.org/issue?@action=redirect&bpo=25349]).
- `bytearray % args` es ahora entre 2,5 y 5 veces más rápido. (Contribuido por Victor Stinner en bpo-25399 [https://bugs.python.org/issue?@action=redirect&bpo=25399]).
- Optimice **`bytes.fromhex()`** y **`bytearray.fromhex()`**: ahora son entre 2 y 3,5 veces más rápidos. (Contribuido por Victor Stinner en bpo-25401 [https://bugs.python.org/issue?@action=redirect&bpo=25401]).
- Optimice `bytes.replace(b'', b'.')` y `bytearray.replace(b'', b'.')`: hasta un 80% más rápido. (Contribuido por Josh Snider en bpo-26574 [https://bugs.python.org/issue?@action=redirect&bpo=26574]).
- Las funciones de asignación del dominio **`PyMem_Malloc()`** (**`PYMEM_DOMAIN_MEM`**) ahora utilizan la función pymalloc memory allocator en lugar de la función **`malloc()`** de la biblioteca C. El asignador pymalloc está optimizado para objetos más pequeños o iguales a 512 bytes con una vida útil corta, y utiliza **`malloc()`** para bloques de memoria más grandes. (Contribuido por Victor Stinner en bpo-26249 [https://bugs.python.org/issue?@action=redirect&bpo=26249]).
- **`pickle.load()`** y **`pickle.loads()`** ahora son hasta un 10% más rápidos al deserializar muchos objetos pequeños (contribución de Victor Stinner en bpo-27056 [https://bugs.python.org/issue?@action=redirect&bpo=27056]).
- Pasar keyword arguments a una función tiene una sobrecarga en comparación con pasar positional arguments. Ahora, en las funciones de extensión implementadas con el uso de Argument Clinic, esta sobrecarga se reduce significativamente. (Contribuido por Serhiy Storchaka en bpo-27574 [https://bugs.python.org/issue?@action=redirect&bpo=27574]).
- Funciones **`glob()`** y **`iglob()`** optimizadas en el módulo **`glob`**; ahora son unas 3 a 6 veces más rápidas. (Contribuido por Serhiy Storchaka en bpo-25596 [https://bugs.python.org/issue?

@action=redirect&bpo=25596]).

- Globbing optimizado en **`pathlib`** mediante **`os.scandir()`**; ahora es alrededor de 1,5 a 4 veces más rápido. (Contribuido por Serhiy Storchaka en bpo-26032 [https://bugs.python.org/issue?@action=redirect&bpo=26032]).
- Se ha mejorado significativamente el rendimiento de análisis, iteración y copia profunda de **`xml.etree.ElementTree`**. (Contribuido por Serhiy Storchaka en bpo-25638 [https://bugs.python.org/issue?@action=redirect&bpo=25638], bpo-25873 [https://bugs.python.org/issue?@action=redirect&bpo=25873] y bpo-25869 [https://bugs.python.org/issue?@action=redirect&bpo=25869].)
- La creación de instancias de **`fractions.Fraction`** a partir de flotantes y decimales es ahora de 2 a 3 veces más rápida. (Contribuido por Serhiy Storchaka en bpo-25971 [https://bugs.python.org/issue?@action=redirect&bpo=25971].)

Cambios en la API de Build y C

- Python ahora requiere algo de compatibilidad con C99 en la cadena de herramientas para compilarse. En particular, Python ahora usa macros y tipos de enteros estándar en lugar de macros personalizadas como `PY_LONG_LONG`. Para obtener más información, consulte **PEP 7** [https://peps.python.org/pep-0007/] y bpo-17884 [https://bugs.python.org/issue?@action=redirect&bpo=17884].
- La compilación cruzada de CPython con el NDK de Android y el nivel de API de Android establecido en 21 (Android 5.0 Lollipop) o superior se ejecuta correctamente. Si bien Android aún no es una plataforma compatible, el conjunto de pruebas de Python se ejecuta en el emulador de Android con solo alrededor de 16 fallas en las pruebas. Consulte el meta-problema de Android bpo-26865 [https://bugs.python.org/issue?@action=redirect&bpo=26865].
- Se ha agregado el indicador de configuración `--enable-optimizations`. Activarlo activará optimizaciones costosas como PGO. (Parche original de Alecsandru Patrascu de Intel en bpo-26359 [https://bugs.python.org/issue?@action=redirect&bpo=26359].)

- Ahora se debe mantener GIL cuando se llaman las funciones de asignador de los dominios `PYMEM_DOMAIN_OBJ` (por ejemplo: `PyObject_Malloc()`) y `PYMEM_DOMAIN_MEM` (por ejemplo: `PyMem_Malloc()`).
- Nueva API `Py_FinalizeEx()` que indica si fallaron los datos almacenados en búfer. (Contribuido por Martin Panter en bpo-5319 [https://bugs.python.org/issue?@action=redirect&bpo=5319].)
- `PyArg_ParseTupleAndKeywords()` ahora es compatible con positional-only parameters. Los parámetros solo posicionales se definen mediante nombres vacíos. (Contribuido por Serhiy Storchaka en bpo-26282 [https://bugs.python.org/issue?@action=redirect&bpo=26282]).
- El método `PyTraceback_Print` ahora abrevia secuencias largas de líneas repetidas como `"[Previous line repeated {count} more times]"`. (Contribuido por Emanuel Barry en bpo-26823 [https://bugs.python.org/issue?@action=redirect&bpo=26823].)
- La nueva función `PyErr_SetImportErrorSubclass()` permite especificar una subclase de `ImportError` para aumentar. (Contribuido por Eric Snow en bpo-15767 [https://bugs.python.org/issue?@action=redirect&bpo=15767].)
- La nueva función `PyErr_ResourceWarning()` se puede utilizar para generar un `ResourceWarning` que proporcione la fuente de la asignación de recursos. (Contribuido por Victor Stinner en bpo-26567 [https://bugs.python.org/issue?@action=redirect&bpo=26567].)
- La nueva función `PyOS_FSPath()` devuelve la representación del sistema de archivos de un path-like object. (Contribuido por Brett Cannon en bpo-27186 [https://bugs.python.org/issue?@action=redirect&bpo=27186].)
- Las funciones `PyUnicode_FSConverter()` y `PyUnicode_FSDecoder()` ahora aceptarán path-like objects.

Otras mejoras

- Cuando `--version` (forma corta: `-V`) se suministra dos veces, Python imprime `sys.version` para obtener información detallada.

```
$ ./python -VV
Python 3.6.0b4+ (3.6:223967b49e49+, Nov 21 2016, 20
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0
```

Obsoleto

Nuevas palabras clave

No se recomienda el uso de `async` y `await` como nombres de variable, clase, función o módulo. Introducidos por **PEP 492** [https://peps.python.org/pep-0492/] en Python 3.5, se convertirán en palabras clave adecuadas en Python 3.7. A partir de Python 3.6, el uso de `async` o `await` como nombres lanzará un **`DeprecationWarning`**.

Comportamiento de Python obsoleto

Al generar la excepción **`StopIteration`** dentro de un generador, ahora se lanzará un **`DeprecationWarning`** y se activará un **`RuntimeError`** en Python 3.7. Consulte PEP 479: Cambiar el gestor de StopIteration dentro de generadores para obtener más detalles.

The **`__aiter__()`** method is now expected to return an asynchronous iterator directly instead of returning an awaitable as previously. Doing the former will trigger a **`DeprecationWarning`**. Backward compatibility will be removed in Python 3.7. (Contributed by Yury Selivanov in bpo-27243 [https://bugs.python.org/issue?@action=redirect&bpo=27243].)

Un par de caracteres de barra invertida que no es una secuencia de escape válida ahora genera un **`DeprecationWarning`**. Aunque eventualmente se convertirá en un **`SyntaxError`**, no será así para varias versiones de Python. (Contribuido por Emanuel Barry en bpo-27364 [https://bugs.python.org/issue?@action=redirect&bpo=27364].)

Al realizar una importación relativa, recurrir a `__name__` y `__path__` desde el módulo de llamada cuando `__spec__` o `__package__` no están definidos ahora genera un **`ImportWarning`**. (Contribuido por Rose Ames en bpo-25791 [https://bugs.python.org/issue?@action=redirect&bpo=25791].)

Módulos, funciones y métodos de Python obsoletos

asynchat

El **`asynchat`** ha quedado obsoleto en favor del **`asyncio`**. (Contribuido por Mariatta en bpo-25002 [https://bugs.python.org/issue?@action=redirect&bpo=25002].)

asyncore

El **`asyncore`** ha quedado obsoleto en favor del **`asyncio`**. (Contribuido por Mariatta en bpo-25002 [https://bugs.python.org/issue?@action=redirect&bpo=25002].)

dbm

A diferencia de otras implementaciones de **`dbm`**, el módulo **`dbm.dumb`** crea bases de datos con el modo `'rw'` y permite modificar la base de datos abierta con el modo `'r'`. Este comportamiento ahora está en desuso y se eliminará en 3.8. (Contribuido por Serhiy Storchaka en bpo-21708 [https://bugs.python.org/issue?@action=redirect&bpo=21708].)

distutils

El argumento `extra_path` no documentado para el constructor `distutils.Distribution` ahora se considera obsoleto y generará una advertencia si se configura. La compatibilidad con este parámetro se eliminará en una futura versión de Python. Consulte bpo-27919 [https://bugs.python.org/issue?@action=redirect&bpo=27919] para obtener más detalles.

grp

La compatibilidad con argumentos no enteros en **`getgrgid()`** ha quedado obsoleta. (Contribuido por Serhiy Storchaka en bpo-26129 [https://bugs.python.org/issue?@action=redirect&bpo=26129].)

importlib

Los métodos `importlib.machinery.SourceFileLoader.load_module()` y `importlib.machinery.SourcelessFileLoader.load_module()` ahora están en desuso. Eran las únicas implementaciones restantes de `importlib.abc.Loader.load_module()` en `importlib` que no habían quedado obsoletas en versiones anteriores de Python a favor de `importlib.abc.Loader.exec_module()`.

La clase `importlib.machinery.WindowsRegistryFinder` ya no se utiliza. A partir de la versión 3.6.0, todavía se agrega a `sys.meta_path` de manera predeterminada (en Windows), pero esto puede cambiar en futuras versiones.

os

El soporte no documentado de bytes-like objects general como rutas en funciones `os`, `compile()` y funciones similares ahora está en desuso. (Contribuido por Serhiy Storchaka en bpo-25791 [https://bugs.python.org/issue?@action=redirect&bpo=25791] y bpo-26754 [https://bugs.python.org/issue?@action=redirect&bpo=26754].)

re

La compatibilidad con indicadores en línea `(?letters)` en el medio de la expresión regular ha quedado obsoleta y se eliminará en una versión futura de Python. Aún se permiten las banderas al comienzo de una expresión regular. (Contribuido por Serhiy Storchaka en bpo-22493 [https://bugs.python.org/issue?@action=redirect&bpo=22493].)

ssl

OpenSSL 0.9.8, 1.0.0 y 1.0.1 están obsoletos y ya no son compatibles. En el futuro, el módulo **`ssl`** requerirá al menos OpenSSL 1.0.2 o 1.1.0.

Los argumentos relacionados con SSL como `certfile`, `keyfile` y `check_hostname` en **`ftplib`**, **`http.client`**, **`imaplib`**, **`poplib`** y **`smtplib`** se han desaprobado en favor de `context`. (Contribuido por Christian Heimes en bpo-28022 [https://bugs.python.org/issue?@action=redirect&bpo=28022].)

Un par de protocolos y funciones del módulo **`ssl`** ahora están en desuso. Algunas funciones ya no estarán disponibles en futuras versiones de OpenSSL. Otras funciones están obsoletas en favor de una API diferente. (Contribuido por Christian Heimes en bpo-28022 [https://bugs.python.org/issue?@action=redirect&bpo=28022] y bpo-26470 [https://bugs.python.org/issue?@action=redirect&bpo=26470].)

tkinter

El módulo **`tkinter.tix`** ahora está en desuso. Los usuarios de **`tkinter`** deberían utilizar **`tkinter.ttk`** en su lugar.

venv

La secuencia de comandos `pyvenv` se ha desaprobado en favor de `python3 -m venv`. Esto evita la confusión en cuanto a qué intérprete de Python `pyvenv` está conectado y, por lo tanto, qué intérprete de Python será utilizado por el entorno virtual. (Contribuido por Brett Cannon en bpo-25154 [https://bugs.python.org/issue?@action=redirect&bpo=25154].)

xml

- Como medida de mitigación frente a la recuperación de DTD y entidades externas, los módulos **`xml.dom.minidom`** y **`xml.sax`** ya no procesan entidades externas de forma predeterminada. (Contribuido por Christian Heimes en gh-61441 [https://github.com/python/cpython/issues/61441].)

Funciones y tipos obsoletos de la API de C

Las funciones no documentadas **`PyUnicode_AsEncodedObject()`**, **`PyUnicode_AsDecodedObject()`**, **`PyUnicode_AsEncodedUnicode()`** y **`PyUnicode_AsDecodedUnicode()`** ya no se utilizan. Utilice generic codec based API en su lugar.

Opciones de compilación obsoletas

El indicador de configuración `--with-system-ffi` ahora está activado de forma predeterminada en plataformas UNIX que no son macOS. Puede deshabilitarse mediante `--without-system-ffi`, pero el uso de la marca está obsoleto y no se aceptará en Python 3.7. macOS no se ve afectado por este cambio. Tenga en cuenta que muchos distribuidores de sistemas operativos ya utilizan el indicador `--with-system-ffi` al crear su sistema Python.

Remoto

Eliminaciones de API y funciones

- Los escapes desconocidos que constan de `'\'` y una letra ASCII en expresiones regulares ahora provocarán un error. En las plantillas de reemplazo para **`re.sub()`**, todavía están permitidas, pero en desuso. La bandera **`re.LOCALE`** ahora solo se puede usar con patrones binarios.
- Se eliminó `inspect.getmoduleinfo()` (quedó en desuso desde CPython 3.3). Se debe utilizar **`inspect.getmodulename()`** para obtener el nombre del módulo para una ruta determinada. (Contribuido por Yury Selivanov en bpo-13248 [https://bugs.python.org/issue?@action=redirect&bpo=13248].)
- La clase `traceback.Ignore` y los métodos `traceback.usage`, `traceback.modname`, `traceback.fullmodname`, `traceback.find_lines_from_code`, `traceback.find_lines`, `traceback.find_strings`, `traceback.find_executable_lines` se eliminaron del módulo **`traceback`**. Eran métodos no documentados en

desuso desde Python 3.2 y la funcionalidad equivalente está disponible en métodos privados.

- Se eliminaron los métodos ficticios `tk_menuBar()` y `tk_bindForTraversal()` en las clases de widgets **`tkinter`** (los comandos Tk correspondientes estaban obsoletos desde Tk 4.0).
- El método **`open()`** de la clase **`zipfile.ZipFile`** ya no admite el modo `'U'` (quedó obsoleto desde Python 3.4). Utilice **`io.TextIOWrapper`** para leer archivos de texto comprimidos en modo universal newlines.
- Los módulos indocumentados `IN`, `CDROM`, `DLFCN`, `TYPES`, `CDIO` y `STROPTS` se han eliminado. Habían estado disponibles en los directorios `Lib/plat-*/` específicos de la plataforma, pero estaban desactualizados de forma crónica, estaban disponibles de manera incoherente en todas las plataformas y no se mantenían. El script que creó estos módulos todavía está disponible en la distribución fuente en: source: Tools / scripts / h2py.py [https://github.com/python/cpython/blob/v3.6.15/Tools/scripts/h2py.py].
- Se ha eliminado la clase `asynchat.fifo` obsoleta.

Portar a Python 3.6

Esta sección enumera los cambios descritos anteriormente y otras correcciones de errores que pueden requerir cambios en su código.

Cambios en el comportamiento del comando "python"

- La salida de una compilación especial de Python con macros `COUNT_ALLOCS`, `SHOW_ALLOC_COUNT` o `SHOW_TRACK_COUNT` definidas ahora está desactivada de forma predeterminada. Se puede volver a habilitar usando la opción `-X showalloccount`. Ahora emite a `stderr` en lugar de `stdout`. (Contribuido por Serhiy Storchaka en bpo-23034 [https://bugs.python.org/issue?@action=redirect&bpo=23034].)

Cambios en la API de Python

- `open()` ya no permitirá combinar el indicador de modo `'U'` con `'+'`. (Contribuido por Jeff Balogh y John O’Connor en bpo-2091 [https://bugs.python.org/issue?@action=redirect&bpo=2091].)

- `sqlite3` ya no confirma implícitamente una transacción abierta antes de las declaraciones DDL.

- En Linux, `os.urandom()` ahora se bloquea hasta que se inicializa el grupo de entropía aleatoria del sistema para aumentar la seguridad.

- Cuando se define `importlib.abc.Loader.exec_module()`, también se debe definir `importlib.abc.Loader.create_module()`.

- `PyErr_SetImportError()` ahora establece `TypeError` cuando su argumento **msg** no está establecido. Anteriormente, solo se devolvía `NULL`.

- El formato del atributo `co_lnotab` de los objetos de código cambió para admitir un delta de número de línea negativo. De manera predeterminada, Python no emite código de bytes con un delta de número de línea negativo. Las funciones que usan `frame.f_lineno`, `PyFrame_GetLineNumber()` o `PyCode_Addr2Line()` no se ven afectadas. Las funciones que decodifican directamente `co_lnotab` deben actualizarse para usar un tipo entero de 8 bits con signo para el delta de número de línea, pero esto solo es necesario para admitir aplicaciones que usen un delta de número de línea negativo. Consulte `Objects/lnotab_notes.txt` para conocer el formato `co_lnotab` y cómo decodificarlo, y consulte PEP 511 [https://peps.python.org/pep-0511/] para conocer la justificación.

- Las funciones en el módulo `compileall` ahora devuelven booleanos en lugar de `1` o `0` para representar éxito o fracaso, respectivamente. Gracias a que los booleanos son una subclase de números enteros, esto solo debería ser un problema si estaba haciendo verificaciones de identidad para

`1` o `0`. Ver bpo-25768 [https://bugs.python.org/issue?@action=redirect&bpo=25768].

- Reading the **`port`** attribute of **`urllib.parse.urlsplit()`** and **`urlparse()`** results now raises **`ValueError`** for out-of-range values, rather than returning **`None`**. See bpo-20059 [https://bugs.python.org/issue?@action=redirect&bpo=20059].

- El módulo **`imp`** ahora lanza un **`DeprecationWarning`** en lugar de **`PendingDeprecationWarning`**.

- The following modules have had missing APIs added to their **`__all__`** attributes to match the documented APIs: **`calendar`**, **`cgi`**, **`csv`**, **`ElementTree`**, **`enum`**, **`fileinput`**, **`ftplib`**, **`logging`**, **`mailbox`**, **`mimetypes`**, **`optparse`**, **`plistlib`**, **`smtpd`**, **`subprocess`**, **`tarfile`**, **`threading`** and **`wave`**. This means they will export new symbols when `import *` is used. (Contributed by Joel Taddei and Jacek Kołodziej in bpo-23883 [https://bugs.python.org/issue?@action=redirect&bpo=23883].)

- Al realizar una importación relativa, si `__package__` no se compara igual que `__spec__.parent`, entonces **`ImportWarning`** se lanza. (Contribuido por Brett Cannon en bpo-25791 [https://bugs.python.org/issue?@action=redirect&bpo=25791].)

- Cuando se realiza una importación relativa y no se conoce ningún paquete principal, se lanzará **`ImportError`**. Anteriormente, se podía subir **`SystemError`**. (Contribuido por Brett Cannon en bpo-18018 [https://bugs.python.org/issue?@action=redirect&bpo=18018].)

- Los servidores basados en el módulo **`socketserver`**, incluidos los definidos en **`http.server`**, **`xmlrpc.server`** y **`wsgiref.simple_server`**, ahora solo detectan excepciones derivadas de **`Exception`**. Por lo tanto, si un controlador de solicitudes lanza una excepción como **`SystemExit`** o **`KeyboardInterrupt`**, ya no se llama a **`handle_error()`** y la excepción detendrá un servidor de un solo subproceso. (Contribuido por Martin Panter en bpo-23430 [https://bugs.python.org/issue?@action=redirect&bpo=23430].)

- `spwd.getspnam()` ahora lanza un `PermissionError` en lugar de `KeyError` si el usuario no tiene privilegios.

- El método `socket.socket.close()` ahora lanza una excepción si la llamada al sistema subyacente informó un error (por ejemplo, `EBADF`). (Contribuido por Martin Panter en bpo-26685 [https://bugs.python.org/issue?@action=redirect&bpo=26685].)

- El argumento *decode_data* para los constructores `smtpd.SMTPChannel` y `smtpd.SMTPServer` ahora es `False` de manera predeterminada. Esto significa que el argumento que se pasa a `process_message()` ahora es un objeto bytes de manera predeterminada y a `process_message()` se le pasarán argumentos de palabras clave. El código que ya se haya actualizado de acuerdo con la advertencia de obsolescencia generada por la versión 3.5 no se verá afectado.

- Todos los argumentos opcionales de las funciones `dump()`, `dumps()`, `load()` y `loads()` y los constructores de clase `JSONEncoder` y `JSONDecoder` en el módulo `json` ahora son keyword-only. (Contribuido por Serhiy Storchaka en bpo-18726 [https://bugs.python.org/issue?@action=redirect&bpo=18726].)

- Es posible que las subclases de `type` que no anulan `type.__new__` ya no utilicen la forma de un argumento para obtener el tipo de un objeto.

- As part of **PEP 487** [https://peps.python.org/pep-0487/], the handling of keyword arguments passed to `type` (other than the metaclass hint, `metaclass`) is now consistently delegated to `object.__init_subclass__()`. This means that `type.__new__` and `type.__init__` both now accept arbitrary keyword arguments, but `object.__init_subclass__()` (which is called from `type.__new__`) will reject them by default. Custom metaclasses accepting additional keyword arguments will need to adjust their calls to `type.__new__` (whether direct or via `super`) accordingly.

- En `distutils.command.sdist.sdist`, se ha eliminado el atributo `default_format` y ya no se respeta. En su lugar, el formato de archivo tar comprimido con gzip es el predeterminado en todas las plataformas y no se realiza ninguna selección específica de la plataforma. En entornos donde las distribuciones se crean en Windows y se requieren distribuciones zip, configure el proyecto con un archivo `setup.cfg` que contenga lo siguiente:

  ```
  [sdist]
  formats=zip
  ```

  Este comportamiento también ha sido actualizado a versiones anteriores de Python por Setuptools 26.0.0.

- En el módulo `urllib.request` y el método `http.client.HTTPConnection.request()`, si no se ha especificado ningún campo de encabezado Content-Length y el cuerpo de la solicitud es un objeto de archivo, ahora se envía con codificación fragmentada HTTP 1.1. Si un objeto de archivo debe enviarse a un servidor HTTP 1.0, la persona que llama ahora debe especificar el valor de Longitud de contenido. (Contribuido por Demian Brecht y Rolf Krahl con ajustes de Martin Panter en bpo-12319 [https://bugs.python.org/issue?@action=redirect&bpo=12319].)

- El `DictReader` ahora devuelve filas de tipo `OrderedDict`. (Contribuido por Steve Holden en bpo-27842 [https://bugs.python.org/issue?@action=redirect&bpo=27842].)

- El `crypt.METHOD_CRYPT` ya no se agregará a `crypt.methods` si la plataforma no lo admite. (Contribuido por Victor Stinner en bpo-25287 [https://bugs.python.org/issue?@action=redirect&bpo=25287].)

- Los argumentos *verbose* and *rename* para `namedtuple()`

ahora son solo palabras clave. (Contribuido por Raymond Hettinger en bpo-25628 [https://bugs.python.org/issue?@action=redirect&bpo=25628].)

- En Linux, `ctypes.util.find_library()` ahora busca bibliotecas compartidas en `LD_LIBRARY_PATH`. (Contribuido por Vinay Sajip en bpo-9998 [https://bugs.python.org/issue?@action=redirect&bpo=9998].)

- La clase `imaplib.IMAP4` ahora maneja banderas que contienen el carácter `']'` en mensajes enviados desde el servidor para mejorar la compatibilidad en el mundo real. (Contribuido por Lita Cho en bpo-21815 [https://bugs.python.org/issue?@action=redirect&bpo=21815].)

- The `mmap.mmap.write()` function now returns the number of bytes written like other write methods. (Contributed by Jakub Stasiak in bpo-26335 [https://bugs.python.org/issue?@action=redirect&bpo=26335].)

- Las funciones `pkgutil.iter_modules()` y `pkgutil.walk_packages()` ahora devuelven tuplas con nombre `ModuleInfo`. (Contribuido por Ramchandra Apte en bpo-17211 [https://bugs.python.org/issue?@action=redirect&bpo=17211].)

- `re.sub()` ahora lanza un error para referencias de grupos numéricos no válidos en plantillas de reemplazo, incluso si el patrón no se encuentra en la cadena. El mensaje de error para referencias de grupo no válidas ahora incluye el índice de grupo y la posición de la referencia. (Contribuido por SilentGhost, Serhiy Storchaka en bpo-25953 [https://bugs.python.org/issue?@action=redirect&bpo=25953].)

- `zipfile.ZipFile` ahora lanzará `NotImplementedError` para valores de compresión no reconocidos. Anteriormente se planteó un `RuntimeError` simple. Además, llamar a los métodos `ZipFile` en un ZipFile cerrado o llamar al método `write()` en un ZipFile creado con el modo `'r'` lanzará un `ValueError`. Anteriormente, se planteó un `RuntimeError` en esos escenarios.

- cuando las metaclases personalizadas se combinan con `super()` de argumento cero o referencias directas de métodos a la variable de cierre `__class__` implícita, la entrada de espacio de nombres `__classcell__` implícita ahora debe pasarse a `type.__new__` para la inicialización. No hacerlo resultará en un `DeprecationWarning` en Python 3.6 y un `RuntimeError` en Python 3.8.

- Con la introducción de `ModuleNotFoundError`, los consumidores del sistema de importación pueden comenzar a esperar que los reemplazos del sistema de importación generen esa excepción más específica cuando sea apropiado, en lugar de la excepción menos específica `ImportError`. Para proporcionar compatibilidad futura con dichos consumidores, los implementadores de sistemas de importación alternativos que reemplacen completamente a `__import__()` deberán actualizar sus implementaciones para generar la nueva subclase cuando no se pueda encontrar un módulo. Los implementadores de complementos compatibles con el sistema de importación predeterminado no deberían necesitar realizar ningún cambio, ya que el sistema de importación predeterminado generará la nueva subclase cuando sea apropiado.

Cambios en la API de C

- La familia de asignadores `PyMem_Malloc()` ahora usa el pymalloc allocator en lugar del sistema `malloc()`. Las aplicaciones que llaman a `PyMem_Malloc()` sin sostener el GIL ahora pueden fallar. Establezca la variable de entorno `PYTHONMALLOC` en `debug` para validar el uso de asignadores de memoria en su aplicación. Ver bpo-26249 [https://bugs.python.org/issue?@action=redirect&bpo=26249].
- `Py_Exit()` (y el intérprete principal) ahora anula el estado de salida con 120 si fallaron los datos almacenados en el búfer. Ver bpo-5319 [https://bugs.python.org/issue?@action=redirect&bpo=5319].

Cambios en el código de bytes de CPython

Ha habido varios cambios importantes en el bytecode en Python 3.6.

- El intérprete de Python ahora usa un código de palabra de 16 bits en lugar de un código de bytes. (Contribuido por Demur Rumed con aportes y reseñas de Serhiy Storchaka y Victor Stinner en bpo-26647 [https://bugs.python.org/issue?@action=redirect&bpo=26647] y bpo-28050 [https://bugs.python.org/issue?@action=redirect&bpo=28050].)
- The new **`FORMAT_VALUE`** and **`BUILD_STRING`** opcodes as part of the formatted string literal implementation. (Contributed by Eric Smith in bpo-25483 [https://bugs.python.org/issue?@action=redirect&bpo=25483] and Serhiy Storchaka in bpo-27078 [https://bugs.python.org/issue?@action=redirect&bpo=27078].)
- The new **`BUILD_CONST_KEY_MAP`** opcode to optimize the creation of dictionaries with constant keys. (Contributed by Serhiy Storchaka in bpo-27140 [https://bugs.python.org/issue?@action=redirect&bpo=27140].)
- The function call opcodes have been heavily reworked for better performance and simpler implementation. The **`MAKE_FUNCTION`**, **`CALL_FUNCTION`**, **`CALL_FUNCTION_KW`** and **`BUILD_MAP_UNPACK_WITH_CALL`** opcodes have been modified, the new **`CALL_FUNCTION_EX`** and **`BUILD_TUPLE_UNPACK_WITH_CALL`** have been added, and `CALL_FUNCTION_VAR`, `CALL_FUNCTION_VAR_KW` and `MAKE_CLOSURE` opcodes have been removed. (Contributed by Demur Rumed in bpo-27095 [https://bugs.python.org/issue?@action=redirect&bpo=27095], and Serhiy Storchaka in bpo-27213 [https://bugs.python.org/issue?@action=redirect&bpo=27213], bpo-28257 [https://bugs.python.org/issue?@action=redirect&bpo=28257].)
- The new **`SETUP_ANNOTATIONS`** and **`STORE_ANNOTATION`** opcodes have been added to support the new variable annotation syntax. (Contributed by Ivan Levkivskyi in bpo-27985 [https://bugs.python.org/issue?@action=redirect&bpo=27985].)

Cambios notables en Python 3.6.2

Nuevo objetivo de compilación `make regen-all`

Para simplificar la compilación cruzada y garantizar que CPython se pueda compilar de manera confiable sin requerir que una versión existente de Python ya esté disponible, el sistema de compilación basado en autotools ya no intenta volver a compilar implícitamente los archivos generados en función de los tiempos de modificación del archivo.

En su lugar, se ha agregado un nuevo comando `make regen-all` para forzar la regeneración de estos archivos cuando se desee (por ejemplo, después de que ya se haya creado una versión inicial de Python basada en las versiones pregeneradas).

También se definen objetivos de regeneración más selectivos; consulte: fuente: *Makefile.pre.in* para obtener más detalles.

(Contribuido por Victor Stinner en bpo-23404 [https://bugs.python.org/issue?@action=redirect&bpo=23404].)

Added in version 3.6.2.

Eliminación del objetivo de compilación `make touch`

Se ha eliminado el objetivo de compilación `make touch` que se utilizaba anteriormente para solicitar la regeneración implícita de archivos generados mediante la actualización de sus tiempos de modificación.

Ha sido reemplazado por el nuevo objetivo `make regen-all`.

(Contribuido por Victor Stinner en bpo-23404 [https://bugs.python.org/issue?@action=redirect&bpo=23404].)

Distinto en la versión 3.6.2.

Cambios notables en Python 3.6.4

El singleton `PyExc_RecursionErrorInst` que formaba parte de la API pública se ha eliminado ya que sus miembros nunca borrados pueden causar una falla de segmentación durante la finalización del intérprete. (Contribuido por Xavier de Gaye en bpo-22898 [https://bugs.python.org/issue?@action=redirect&bpo=22898] y bpo-30697 [https://bugs.python.org/issue?@action=redirect&bpo=30697].)

Cambios notables en Python 3.6.5

La función **`locale.localeconv()`** ahora establece temporalmente la configuración regional `LC_CTYPE` en la configuración regional `LC_NUMERIC` en algunos casos. (Contribuido por Victor Stinner en bpo-31900 [https://bugs.python.org/issue?@action=redirect&bpo=31900].)

Cambios notables en Python 3.6.7

Los módulos **`xml.dom.minidom`** y **`xml.sax`** ya no procesan entidades externas de forma predeterminada. Consulte también gh-61441 [https://github.com/python/cpython/issues/61441].

En 3.6.7, el módulo **`tokenize`** ahora emite implícitamente un token `NEWLINE` cuando se le proporciona una entrada que no tiene una nueva línea al final. Este comportamiento ahora coincide con lo que hace el tokenizador C internamente. (Contribuido por Ammar Askar en bpo-33899 [https://bugs.python.org/issue?@action=redirect&bpo=33899].)

Cambios notables en Python 3.6.10

Debido a importantes problemas de seguridad, el parámetro *reuse_address* de **`asyncio.loop.create_datagram_endpoint()`** ya no es compatible. Esto se debe al comportamiento de la opción de socket `SO_REUSEADDR` en UDP. Para obtener más detalles, consulte la documentación de `loop.create_datagram_endpoint()`. (Contribuido por Kyle Stanley, Antoine Pitrou y Yury Selivanov en bpo-37228 [https://bugs.python.org/issue?@action=redirect&bpo=37228].)

Cambios notables en Python 3.6.13

Las versiones anteriores de Python permitían utilizar tanto `;` como `&` como separadores de parámetros de consulta en **`urllib.parse.parse_qs()`** y **`urllib.parse.parse_qsl()`**. Por cuestiones de seguridad y para cumplir con las recomendaciones más recientes del W3C, esto se ha modificado para permitir solo una clave separadora, con `&` como valor predeterminado. Este cambio también afecta a **`cgi.parse()`** y **`cgi.parse_multipart()`**, ya que utilizan las funciones afectadas internamente. Para obtener más detalles, consulte su documentación respectiva. (Contribuido por Adam Goldschmidt, Senthil Kumaran y Ken Jin en bpo-42967 [https://bugs.python.org/issue?@action=redirect&bpo=42967]).

Cambios notables en Python 3.6.14

Una corrección de seguridad modifica el comportamiento de **`ftplib.FTP`** para que no confíe en la dirección IPv4 enviada desde el servidor remoto al configurar un canal de datos pasivo. En su lugar, reutilizamos la dirección IP del servidor FTP. Para el código inusual que requiere el comportamiento anterior, configure un atributo `trust_server_pasv_ipv4_address` en su instancia FTP en `True`. (Consulte gh-87451 [https://github.com/python/cpython/issues/87451])

La presencia de caracteres de nueva línea o de tabulación en partes de una URL permite algunas formas de ataques. Siguiendo la especificación WHATWG que actualiza RFC 3986, el analizador **`urllib.parse()`** elimina de la URL los caracteres de nueva línea ASCII `\n`, `\r` y de tabulación `\t` para evitar este tipo de ataques. Los caracteres eliminados se controlan mediante una nueva variable de nivel de módulo `urllib.parse._UNSAFE_URL_BYTES_TO_REMOVE`. (Véase gh-88048 [https://github.com/python/cpython/issues/88048])

Qué hay de nuevo en Python 3.5

Editores:

Elvis Pranskevichus <elvis@magic.io>, Yury Selivanov <yury@magic.io>

This article explains the new features in Python 3.5, compared to 3.4. Python 3.5 was released on September 13, 2015. See the changelog [https://docs.python.org/3.5/whatsnew/changelog.html] for a full list of changes.

Ver también

PEP 478 [https://peps.python.org/pep-0478/] - Python 3.5 Calendario de lanzamiento de Python 3.5

Resumen – Aspectos destacados de la versión

Nuevas características de sintaxis:

- PEP 492, corrutinas con sintaxis async y await.
- PEP 465, un nuevo operador de multiplicación de matrices: `a @ b`.
- PEP 448, generalizaciones de desembalaje adicionales.

Nuevos módulos:

- **`typing`**: PEP 484 – Indicador de tipos.
- **`zipapp`**: PEP 441 Mejora de soporte de la aplicación Python ZIP.

Nuevas características integradas:

- `bytes % args`, `bytearray % args`: PEP 461 – Agrega formato `%` a bytes y bytearray.
- Nuevos métodos **`bytes.hex()`**, **`bytearray.hex()`** y **`memoryview.hex()`**. (Contribución de Arnon Yaari en bpo-9951 [https://bugs.python.org/issue?@action=redirect&bpo=9951].)
- **`memoryview`** ahora admite la indexación de tuplas (incluida la multidimensional). (Contribución de Antoine Pitrou en bpo-23632 [https://bugs.python.org/issue?@action=redirect&bpo=23632].)
- Los generadores tienen un nuevo atributo `gi_yieldfrom`, el cual retorna el objeto que está siendo iterado por expresiones `yield from`. (Contribución de Benno Leslie y Yury Selivanov en bpo-24450 [https://bugs.python.org/issue?@action=redirect&bpo=24450].)
- Ahora se genera una nueva excepción **`RecursionError`** cuando se alcanza la profundidad máxima de recursividad. (Contribución de Georg Brandl en bpo-19235 [https://bugs.python.org/issue?@action=redirect&bpo=19235].)

Mejoras en la implementación de CPython:

- Cuando la configuración regional `LC_TYPE` es la configuración regional POSIX (configuración regional `C`), ahora **`sys.stdin`** y **`sys.stdout`** usan el controlador de errores `surrogateescape`, en lugar del controlador de errores `strict`. (Contribución de Victor Stinner en bpo-19977 [https://bugs.python.org/issue?@action=redirect&bpo=19977].)
- Los archivos `.pyo` ya no se utilizan y han sido reemplazados por un esquema más flexible que incluye el nivel de optimización explícitamente en el nombre `.pyc`. (Consultar la PEP 488 descripción general.)
- Ahora los módulos integrados y de extensión son inicializados en un proceso de múltiples fases, que es similar a cómo los módulos de Python son cargados. (Consultar PEP 489 descripción general.)

Mejoras significativas en la biblioteca estándar:

- **`collections.OrderedDict`** ahora está implementado en

C, que lo hace de 4 a 100 veces más rápido.
- El módulo **`ssl`** adquirió soporte para la memoria BIO, que desacopla el manejo del protocolo SSL de la red de E/S.
- La nueva función **`os.scandir()`** proporciona una forma mejor y significativamente más rápida de recorrido de directorio.
- **`functools.lru_cache()`** ha sido principalmente reimplementado en C, produciendo un rendimiento mucho mejor.
- La nueva función **`subprocess.run()`** proporciona una forma simplificada de ejecutar subprocesos.
- El módulo **`traceback`** se ha mejorado significativamente para mejorar el rendimiento y la conveniencia del desarrollador.

Mejoras de seguridad:

- Ahora SSLv3 está deshabilitado en toda la biblioteca estándar. Aún se puede habilitar creando una instancia de **`ssl.SSLContext`** manualmente. (Consultar bpo-22638 [https://bugs.python.org/issue?@action=redirect&bpo=22638] para más detalles; este cambio fue respaldado a CPython 3.4 y 2.7.)
- Ahora el análisis de cookies HTTP es más estricto, con el fin de proteger contra posibles ataques de entrada. (Contribución de Antoine Pitrou en bpo-22796 [https://bugs.python.org/issue?@action=redirect&bpo=22796].)

Mejoras de Windows:

- Un nuevo instalador para Windows ha reemplazado al antiguo MSI. Consultar Uso de Python en Windows para más información.
- Ahora las compilaciones de Windows usan Microsoft Visual C ++ 14.0, y los módulos de extensión deberían utilizar lo mismo.

Por favor sigue leyendo para obtener una lista completa de los cambios que se enfrentan los usuarios, incluidas muchas otras mejoras menores, optimizaciones de CPython, deprecaciones y posibles problemas de portabilidad.

Nuevas características

PEP 492 - Corrutinas con sintaxis async y await

PEP 492 [https://peps.python.org/pep-0492/] mejora enormemente el soporte para la programación asíncrona en Python al agregar objetos aguardables, funciones corrutina, iteración asincrónica, y gestores asincrónicos de contexto.

Las funciones corrutina se declaran usando la nueva sintaxis `async def`:

```
>>> async def coro():
...     return 'spam'
```

Inside a coroutine function, the new `await` expression can be used to suspend coroutine execution until the result is available. Any object can be be *awaited*, as long as it implements the awaitable protocol by defining the `__await__()` method.

PEP 492 también agrega la declaración `async for` para una iteración conveniente sobre iterables asincrónicas.

Un ejemplo de un cliente rudimentario HTTP escrito con la nueva sintaxis:

```
import asyncio

async def http_get(domain):
    reader, writer = await asyncio.open_connection(domai

    writer.write(b'\r\n'.join([
        b'GET / HTTP/1.1',
        b'Host: %b' % domain.encode('latin-1'),
        b'Connection: close',
        b'', b''
    ]))

    async for line in reader:
        print('>>>', line)
    writer.close()

loop = asyncio.get_event_loop()
try:
    loop.run_until_complete(http_get('example.com'))
finally:
    loop.close()
```

De manera similar a la iteración asincrónica, hay una nueva sintaxis para los gestores asincrónicos de contexto. El siguiente script:

```
import asyncio

async def coro(name, lock):
    print('coro {}: waiting for lock'.format(name))
    async with lock:
        print('coro {}: holding the lock'.format(name))
        await asyncio.sleep(1)
        print('coro {}: releasing the lock'.format(name)

loop = asyncio.get_event_loop()
lock = asyncio.Lock()
coros = asyncio.gather(coro(1, lock), coro(2, lock))
try:
    loop.run_until_complete(coros)
finally:
    loop.close()
```

saldrá:

```
coro 2: waiting for lock
coro 2: holding the lock
coro 1: waiting for lock
coro 2: releasing the lock
coro 1: holding the lock
coro 1: releasing the lock
```

Ten en cuenta que tanto `async for` como `async with` solamente pueden ser utilizadas dentro de una función corrutina declarada con `async def`.

Las funciones corrutina están destinadas para ser ejecutadas dentro de un bucle de eventos compatible, como asyncio loop.

Nota

Distinto en la versión 3.5.2: A partir de CPython 3.5.2, `__aiter__` puede retornar directamente iteradores asincrónicos. Retornar un objeto awaitable resultará en `PendingDeprecationWarning`.

Consultar más detalles en la sección de la documentación Iteradores asíncronos.

Ver también

PEP 492 [https://peps.python.org/pep-0492/] – Corrutinas con sintaxis async y await

PEP escrito e implementado por Yury Selivanov.

PEP 465 - Un operador infijo dedicado para la multiplicación de matrices

PEP 465 [https://peps.python.org/pep-0465/] adds the `@` infix operator for matrix multiplication. Currently, no builtin Python types implement the new operator, however, it can be implemented by defining `__matmul__()`, `__rmatmul__()`, and `__imatmul__()` for regular, reflected, and in-place matrix multiplication. The semantics of these methods is similar to that of methods defining other infix arithmetic operators.

La multiplicación de matrices es una operación notablemente común en muchos campos de las matemáticas, la ciencia, la ingeniería, y la adición de `@` permite escribir código más limpio:

```
S = (H @ beta - r).T @ inv(H @ V @ H.T) @ (H @ beta - r)
```

en lugar de:

```
S = dot((dot(H, beta) - r).T,
        dot(inv(dot(dot(H, V), H.T)), dot(H, beta) - r))
```

NumPy 1.10 tiene soporte para el nuevo operador:

```
>>> import numpy

>>> x = numpy.ones(3)
>>> x
array([ 1., 1., 1.])

>>> m = numpy.eye(3)
>>> m
array([[ 1., 0., 0.],
       [ 0., 1., 0.],
       [ 0., 0., 1.]])

>>> x @ m
array([ 1., 1., 1.])
```

Ver también

PEP 465 [https://peps.python.org/pep-0465/] – Un operador infijo dedicado para la multiplicación de matrices

PEP escrito por Nathaniel J. Smith; implementado por Benjamin Peterson.

PEP 448 - Generalizaciones de desembalaje adicionales

PEP 448 [https://peps.python.org/pep-0448/] extiende los usos permitidos del operador de desembalaje iterable `*` y del operador de desembalaje del diccionario `**`. Ahora es posible utilizar un número arbitrario de desembalajes en llamadas a funciones:

```
>>> print(*[1], *[2], 3, *[4, 5])
1 2 3 4 5

>>> def fn(a, b, c, d):
```

```
...     print(a, b, c, d)
...

>>> fn(**{'a': 1, 'c': 3}, **{'b': 2, 'd': 4})
1 2 3 4
```

De manera similar, las pantallas de tupla, lista, conjunto y diccionario permiten desembalajes múltiples (consultar Listas de expresiones and Despliegues de diccionario):

```
>>> *range(4), 4
(0, 1, 2, 3, 4)

>>> [*range(4), 4]
[0, 1, 2, 3, 4]

>>> {*range(4), 4, *(5, 6, 7)}
{0, 1, 2, 3, 4, 5, 6, 7}

>>> {'x': 1, **{'y': 2}}
{'x': 1, 'y': 2}
```

Ver también

PEP 448 [https://peps.python.org/pep-0448/] – Generalizaciones de desembalaje adicionales

PEP escrito por Joshua Landau; implementado por Neil Girdhar, Thomas Wouters y Joshua Landau.

PEP 461 - soporte de formateo porcentual para bytes y bytearray

PEP 461 [https://peps.python.org/pep-0461/] agrega soporte para el operador de interpolación `%` a **`bytes`** y **`bytearray`**.

Si bien la interpolación generalmente se considera una operación de cadena de caracteres, hay casos donde la interpolación en `bytes` o `bytearrays` tiene sentido y el trabajo necesario para compensar esta funcionalidad faltante resta valor a la legibilidad general del código. Este problema es particularmente importante cuando se trata con protocolos de formato de cable, que a menudo son una mezcla de texto compatible binario y ASCII.

Ejemplos:

```
>>> b'Hello %b!' % b'World'
b'Hello World!'

>>> b'x=%i y=%f' % (1, 2.5)
b'x=1 y=2.500000'
```

Unicode no está permitido para `%b`, pero es aceptado por `%a` (equivalente a `repr(obj).encode('ascii', 'backslashreplace')`):

```
>>> b'Hello %b!' % 'World'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: %b requires bytes, or an object that implemen

>>> b'price: %a' % '10€'
b"price: '10\\u20ac'"
```

Ten en cuenta que los tipos de conversión `%s` y `%r`, aunque son compatibles, solo deben usarse en bases de código que necesiten compatibilidad con Python 2.

Ver también

PEP 461 [https://peps.python.org/pep-0461/] – Agrega formato `%` a bytes y bytearray

PEP escrito por Ethan Furman; implementado por Neil Schemenauer y Ethan Furman.

PEP 484 - Indicador de tipos

La sintaxis de anotaciones de funciones ha sido una característica de Python desde la versión 3.0 (PEP 3107 [https://peps.python.org/

pep-3107/]), sin embargo, la semántica de anotaciones se ha dejado indefinida.

La experiencia ha mostrado que la mayoría de los usos de las anotaciones de funciones eran para proporcionar el indicador de tipos para los parámetros de función y los valores de retorno. Se hizo evidente que sería beneficioso para los usuarios de Python, si la biblioteca estándar incluyera las definiciones y las herramientas base para las anotaciones de tipo.

PEP 484 [https://peps.python.org/pep-0484/] introduce un módulo provisional para proporcionar estas definiciones y herramientas estándar, junto con algunas convenciones para situaciones en la que las anotaciones no están disponibles.

Por ejemplo, aquí hay una función simple cuyo argumento y tipo de retorno se declaran en las anotaciones:

```
def greeting(name: str) -> str:
    return 'Hello ' + name
```

While these annotations are available at runtime through the usual `__annotations__` attribute, *no automatic type checking happens at runtime*. Instead, it is assumed that a separate off-line type checker (e.g. mypy [https://mypy-lang.org]) will be used for on-demand source code analysis.

El sistema de tipos admite uniones, tipos genéricos y un tipo especial llamado `Any` el cual es consistente con (esto es asignable desde y hacia) todos los tipos.

Ver también

- `typing` documentación del módulo
- **PEP 484** [https://peps.python.org/pep-0484/] **– Indicador de tipos**

  PEP escrito por Guido van Rossum, Jukka Lehtosalo y Łukasz Langa; implementado por Guido van Rossum.

- **PEP 483 [https://peps.python.org/pep-0483/] – La teoría del indicador de tipos**
  PEP escrito por Guido van Rossum

PEP 471 - Función os.scandir() – un iterador de directorio mejor y más rápido

PEP 471 [https://peps.python.org/pep-0471/] agrega una nueva función de iteración de directorio, **`os.scandir()`**, a la biblioteca estándar. Adicionalmente, ahora **`os.walk()`** se implementa utilizando `scandir`, lo que lo hace de 3 a 5 veces más rápido en sistemas POSIX y de 7 a 20 veces más rápido en sistemas Windows. Esto se logra en gran medida al reducir enormemente el número de llamadas necesarias de **`os.stat()`** para recorrer un árbol de directorios.

También `scandir` retorna un iterador, en lugar de retornar una lista de nombres de archivos, lo que mejora la eficiencia de la memoria cuando se itera sobre directorios muy grandes.

El siguiente ejemplo presenta un uso simple de **`os.scandir()`** para mostrar todos los archivos (excluyendo directorios) en el *path* dado que no comienzan con `'.'`. La llamada a **`entry.is_file()`** generalmente no hará una llamada adicional al sistema:

```
for entry in os.scandir(path):
    if not entry.name.startswith('.') and entry.is_file
        print(entry.name)
```

Ver también

PEP 471 [https://peps.python.org/pep-0471/] – Función os.scandir() – un iterador de directorio mejor y más rápido
PEP escrito e implementado por Ben Hoyt con la ayuda de Victor Stinner.

PEP 475: Reintentar las llamadas al sistema que fallan

con EINTR

An `errno.EINTR` error code is returned whenever a system call, that is waiting for I/O, is interrupted by a signal. Previously, Python would raise `InterruptedError` in such cases. This meant that, when writing a Python application, the developer had two choices:

1. Ignorar el `InterruptedError`.
2. Manejar el `InterruptedError` y probar reiniciar la llamada interrumpida al sistema en cada sitio de llamada.

La primera opción hace que una aplicación falle intermitentemente. La segunda opción agrega una gran cantidad de código repetitivo que hace que el código sea casi ilegible. Compara:

```
print("Hello World")
```

y:

```
while True:
    try:
        print("Hello World")
        break
    except InterruptedError:
        continue
```

PEP 475 [https://peps.python.org/pep-0475/] implementa el reintento automático de las llamadas al sistema en `EINTR`. Esto elimina la carga de lidiar con `EINTR` o la excepción `InterruptedError` en el código de usuario en la mayoría de las situaciones y hace que los programas de Python, incluida la biblioteca estándar, sean más robustos. Ten en cuenta que la llamada al sistema sólo se reintenta si el gestor de señales no lanza una excepción.

A continuación se muestra una lista de funciones que ahora se reintentan cuando son interrumpidas por una señal:

- `open()` y `io.open()`;
- funciones del módulo `faulthandler`;
- funciones del módulo `os`: `fchdir()`, `fchmod()`, `fchown()`, `fdatasync()`, `fstat()`, `fstatvfs()`,

  `fsync()`, `ftruncate()`, `mkfifo()`, `mknod()`, `open()`, `posix_fadvise()`, `posix_fallocate()`, `pread()`, `pwrite()`, `read()`, `readv()`, `sendfile()`, `wait3()`, `wait4()`, `wait()`, `waitid()`, `waitpid()`, `write()`, `writev()`;
- special cases: `os.close()` and `os.dup2()` now ignore `EINTR` errors; the syscall is not retried (see the PEP for the rationale);
- funciones del módulo `select`: `devpoll.poll()`, `epoll.poll()`, `kqueue.control()`, `poll.poll()`, `select()`;
- métodos de la clase `socket`: `accept()`, `connect()` (excepto para sockets sin bloqueo), `recv()`, `recvfrom()`, `recvmsg()`, `send()`, `sendall()`, `sendmsg()`, `sendto()`;
- `signal.sigtimedwait()` y `signal.sigwaitinfo()`;
- `time.sleep()`.

Ver también

PEP 475 [https://peps.python.org/pep-0475/] – Reintentar las llamadas al sistema que fallan con EINTR

PEP e implementación escrita por Charles-François Natali y Victor Stinner, con la ayuda de Antoine Pitrou (la conexión francesa).

PEP 479: Cambiar el gestor de StopIteration dentro de generadores

La interacción de los generadores y `StopIteration` en las versiones de Python 3.4 y anteriores a veces fue sorprendente y podría ocultar bugs oscuros. Anteriormente, el `StopIteration` lanzado accidentalmente dentro de una función de generadores se interpretó como el final de la iteración por la construcción de bucle que impulsa el generador.

PEP 479 [https://peps.python.org/pep-0479/] cambia el comportamiento de los generadores: cuando se lanza una excepción `StopIteration` dentro de un generador, se reemplaza con una excepción **`RuntimeError`** antes de que salga del marco del generador. El objetivo principal de este cambio es facilitar la depuración en la situación en la que una llamada desprotegida a la función **`next()`** lanza una excepción `StopIteration` y hace que la iteración controlada por el generador termine silenciosamente. Esto es particularmente pernicioso en combinación con la construcción `yield from`.

Este es un cambio incompatible hacia atrás, así que para habilitar el nuevo comportamiento, es necesario importar __future__:

```
>>> from __future__ import generator_stop

>>> def gen():
...     next(iter([]))
...     yield
...
>>> next(gen())
Traceback (most recent call last):
  File "<stdin>", line 2, in gen
StopIteration

The above exception was the direct cause of the followin

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: generator raised StopIteration
```

Si no se importa `__future__`, se lanzará la excepción **`PendingDeprecationWarning`** siempre que se lance una excepción **`StopIteration`** dentro de un generador.

Ver también

PEP 479 [https://peps.python.org/pep-0479/] – Cambiar el gestor de StopIteration dentro de generadores

PEP escrito por Chris Angelico y Guido van Rossum. Implementado por Chris Angelico, Yury Selivanov y Nick Coghlan.

PEP 485: Una función para probar la igualdad aproximada

PEP 485 [https://peps.python.org/pep-0485/] agrega las funciones **`math.isclose()`** y **`cmath.isclose()`** que indican si dos valores son aproximadamente iguales o «cercanos» entre sí. Si dos valores se consideran cercanos o no, se determina conforme con las tolerancias absolutas y relativas dadas. La tolerancia relativa es la diferencia máxima permitida entre los argumentos `isclose`, en relación con el valor absoluto mayor:

```
>>> import math
>>> a = 5.0
>>> b = 4.99998
>>> math.isclose(a, b, rel_tol=1e-5)
True
>>> math.isclose(a, b, rel_tol=1e-6)
False
```

También es posible comparar dos valores usando una tolerancia absoluta, que debe ser un valor no negativo:

```
>>> import math
>>> a = 5.0
>>> b = 4.99998
>>> math.isclose(a, b, abs_tol=0.00003)
True
>>> math.isclose(a, b, abs_tol=0.00001)
False
```

Ver también

PEP 485 [https://peps.python.org/pep-0485/] – Una función para probar la igualdad aproximada

PEP escrito por Christopher Barker; implementado por Chris Barker y Tal Einat.

PEP 486: Hacer que el launcher de Python sea consciente de los entornos virtuales

PEP 486 [https://peps.python.org/pep-0486/] hace que el launcher de Windows (consultar **PEP 397** [https://peps.python.org/pep-0397/]) sea consciente de un entorno virtual activo. Cuando se usa el intérprete predeterminado y se establece la variable de entorno `VIRTUAL_ENV` está configurado, se utilizará el intérprete en el entorno virtual.

Ver también

PEP 486 [https://peps.python.org/pep-0486/] – Hacer que el launcher de Python sea consciente de los entornos virtuales

PEP escrito e implementado por Paul Moore.

PEP 488: Eliminación de archivos PYO

PEP 488 [https://peps.python.org/pep-0488/] elimina el concepto de archivos `.pyo`. Esto significa que los archivos `.pyc` representan tanto bytecode optimizado y sin optimizar. Para evitar la necesidad de regenerar constantemente archivos bytecode, ahora los archivos `.pyc` tienen una etiqueta opcional `opt-` en su nombre cuando se optimiza el bytecode. Esto tiene el efecto secundario de que no habrá más conflictos de nombres de archivos de bytecode cuando se ejecuta bajo `-O` o `-OO`. Por consiguiente, ahora los archivos bytecode generados a partir de `-O` y `-OO` pueden existir simultáneamente. `importlib.util.cache_from_source()` tiene una API actualizada para ayudar con este cambio.

Ver también

PEP 488 [https://peps.python.org/pep-0488/] – Eliminación de archivos PYO

PEP escrito e implementado por Brett Cannon.

PEP 489: Inicialización del módulo de extensión multifase

PEP 489 [https://peps.python.org/pep-0489/] actualiza la inicialización del módulo de extensión para aprovechar el mecanismo de carga del módulo de dos pasos introducido por **PEP 451** [https://peps.python.org/pep-0451/] en Python 3.4.

Este cambio acerca la semántica de importación de los módulos de extensión que optan por usar el mecanismo nuevo a la de los módulos fuente y de bytecode de Python, incluida la capacidad de utilizar cualquier identificador válido como un nombre de módulo, en lugar de estar restringido a ASCII.

Ver también

PEP 489 [https://peps.python.org/pep-0489/] – Inicialización del módulo de extensión multifase

PEP escrito por Petr Viktorin, Stefan Behnel y Nick Coghlan; implementado por Petr Viktorin.

Otros cambios en el lenguaje

Algunos cambios más pequeños que se hicieron en el lenguaje central de Python son:

- Se agregaron los gestores de error `"namereplace"`. Ahora los gestores de error `"backslashreplace"` funcionan con decodificación y traducción. (Contribución de Serhiy Storchaka en bpo-19676 [https://bugs.python.org/issue?@action=redirect&bpo=19676] y bpo-22286 [https://bugs.python.org/issue?@action=redirect&bpo=22286].)
- Ahora la opción **`-b`** afecta comparaciones de **`bytes`** con **`int`**. (Contribución de Serhiy Storchaka en bpo-23681 [https://bugs.python.org/issue?@action=redirect&bpo=23681].)
- Nuevos códecs Kazakh `kz1048` y Tajik `koi8_t`. (Contribución de Serhiy Storchaka en bpo-22682 [https://bugs.python.org/issue?@action=redirect&bpo=22682] y bpo-22681 [https://bugs.python.org/issue?@action=redirect&bpo=22681].)
- Ahora las docstrings de propiedad se pueden escribir. Esto es especialmente útil para las docstrings **`collections.namedtuple()`**. (Contribución de Berker Peksag en bpo-24064 [https://bugs.python.org/issue?@action=redirect&bpo=24064].)

- Ahora se admiten las importaciones circulares que involucran importaciones relativas. (Contribución de Brett Cannon y Antoine Pitrou en bpo-17636 [https://bugs.python.org/issue?@action=redirect&bpo=17636].)

Nuevos módulos

typing

El módulo nuevo provisional `typing` proporciona definiciones y herramientas estándar para anotaciones de tipos de funciones. Consultar Indicador de tipos para más información.

zipapp

El módulo nuevo `zipapp` (especificado en **PEP 441** [https://peps.python.org/pep-0441/]) proporciona una API y una herramienta de línea de comandos para crear aplicaciones Python Zip ejecutables, las cuales se introdujeron en la versión de Python 2.6 en bpo-1739468 [https://bugs.python.org/issue?@action=redirect&bpo=1739468], pero no se publicitaron bien, ni en ese momento ni desde entonces.

Con el módulo nuevo, empaquetar tu aplicación es tan simple como colocar todos los archivos, incluido un archivo `__main__.py` en un directorio `myapp` y ejecutar:

```
$ python -m zipapp myapp
$ python myapp.pyz
```

La implementación del módulo ha sido contribución de Paul Moore en bpo-23491 [https://bugs.python.org/issue?@action=redirect&bpo=23491].

Ver también

PEP 441 [https://peps.python.org/pep-0441/] – Mejora de soporte de la aplicación Python ZIP

Módulos mejorados

argparse

Ahora la clase **`ArgumentParser`** permite deshabilitar el uso abreviado de opciones largas al establecer allow_abbrev en `False`. (Contribución de Jonathan Paugh, Steven Bethard, paul j3 y Daniel Eriksson en bpo-14910 [https://bugs.python.org/issue?@action=redirect&bpo=14910].)

asyncio

Ya que el módulo **`asyncio`** es provisional, todos los cambios que se introdujeron en Python 3.5 también se han actualizado en Python 3.4.x.

Cambios notables en el módulo **`asyncio`** desde Python 3.4.0:

- Nuevas API de depuración: métodos **`loop.set_debug()`** y **`loop.get_debug()`**. (Contribución de Victor Stinner.)
- Ahora el ciclo de eventos proactor es compatible con SSL. (Contribución de Antoine Pitrou y Victor Stinner en bpo-22560 [https://bugs.python.org/issue?@action=redirect&bpo=22560].)
- Un nuevo método **`loop.is_closed()`** para comprobar si el ciclo de eventos está cerrado. (Contribución de Victor Stinner en bpo-21326 [https://bugs.python.org/issue?@action=redirect&bpo=21326].)
- Un nuevo método **`loop.create_task()`** para crear y programar convenientemente una nueva clase **`Task`** para una corrutina. El método `create_task` también se usa para todas las funciones asyncio que contengan corrutinas en tareas, tales como **`asyncio.wait()`**, **`asyncio.gather()`**, etc. (Contribución de Victor Stinner.)
- Un nuevo método **`transport.get_write_buffer_limits()`** para consultar los límites *high*- (superior) y *low*- (inferior) del control de flujo. (Contribución de Victor Stinner.)
- The **`async()`** function is deprecated in favor of

`ensure_future()`. (Contributed by Yury Selivanov.)
- Nuevos métodos `loop.set_task_factory()` y `loop.get_task_factory()` para personalizar la fábrica de tareas que el método `loop.create_task()` usa. (Contribución de Yury Selivanov.)
- Nuevos métodos de cola `Queue.join()` y `Queue.task_done()`. (Contribución de Victor Stinner.)
- Se eliminó la clase `JoinableQueue`, en favor de la clase `asyncio.Queue`. (Contribución de Victor Stinner.)

Actualizaciones en 3.5.1:

- La función `ensure_future()` y todas las funciones que la utilizan, como el método `loop.run_until_complete()`, ahora acepta todo tipo de objetos aguardables. (Contribución de Yury Selivanov.)
- Nueva función `run_coroutine_threadsafe()` para enviar corrutinas a bucles de eventos de otros hilos. (Contribución de Vincent Michel.)
- Nuevo método `Transport.is_closing()` para comprobar si el transporte se está cerrando o ya está cerrado. (Contribución de Yury Selivanov.)
- Ahora el método `loop.create_server()` puede aceptar una lista de hosts. (Contribución de Yann Sionneau.)

Actualizaciones en 3.5.2:

- Nuevo método `loop.create_future()` para crear objetos Future. Esto permite implementaciones alternativas de bucles de eventos, como uvloop [https://github.com/MagicStack/uvloop], para proporcionar una implementación más rápida de `asyncio.Future`. (Contribución de Yury Selivanov.)
- Nuevo método `loop.get_exception_handler()` para obtener el gestor de excepciones actual. (Contribución de Yury Selivanov.)
- Nuevo método `StreamReader.readuntil()` para leer datos de la secuencia hasta que aparezca una secuencia de separador de bytes. (Contribución de Mark Korenberg.)
- Se optimizaron los métodos `loop.create_connection()` y `loop.create_server()` para evitar llamar al sistema la función `getaddrinfo` si la dirección ya está resuelta.

(Contribución de A. Jesse Jiryu Davis.)

- El método `loop.sock_connect(sock, address)` ya no requiere *address* para que se resuelva antes de la llamada. (Contribución de A. Jesse Jiryu Davis.)

bz2

Ahora el método `BZ2Decompressor.decompress` acepta un argumento opcional *max_length* para limitar el tamaño máximo de datos descomprimidos. (Contribución de Nikolaus Rath en bpo-15955 [https://bugs.python.org/issue?@action=redirect&bpo=15955].)

cgi

The **`FieldStorage`** class now supports the context manager protocol. (Contributed by Berker Peksag in bpo-20289 [https://bugs.python.org/issue?@action=redirect&bpo=20289].)

cmath

Una nueva función `isclose()` proporciona una forma de probar la igualdad aproximada. (Contribución de Chris Barker y Tal Einat en bpo-24270 [https://bugs.python.org/issue?@action=redirect&bpo=24270].)

code

Ahora el método `InteractiveInterpreter.showtraceback()` imprime el rastreo encadenado completo, al igual que el interprete interactivo. (Contribución de Claudiu Popa en bpo-17442 [https://bugs.python.org/issue?@action=redirect&bpo=17442].)

collections

Ahora la clase `OrderedDict` está implementada en C, lo que la hace de 4 a 100 veces más rápida. (Contribución de Eric Snow en bpo-16991 [https://bugs.python.org/issue?@action=redirect&bpo=16991].)

`OrderedDict.items()`, `OrderedDict.keys()`, and `OrderedDict.values()` views now support `reversed()`

iteration. (Contributed by Serhiy Storchaka in bpo-19505 [https://bugs.python.org/issue?@action=redirect&bpo=19505].)

Ahora la clase `deque` define `index()`, `insert()` y `copy()`, y soporta los operadores `+` y `*`. Esto permite que las deques sean reconocidas como `MutableSequence` y mejora su sustituibilidad por listas. (Contribución de Raymond Hettinger en bpo-23704 [https://bugs.python.org/issue?@action=redirect&bpo=23704].)

Ahora las docstrings producidas por `namedtuple()` se pueden actualizar:

```
Point = namedtuple('Point', ['x', 'y'])
Point.__doc__ += ': Cartesian coordinate'
Point.x.__doc__ = 'abscissa'
Point.y.__doc__ = 'ordinate'
```

(Contribución de Berker Peksag en bpo-24064 [https://bugs.python.org/issue?@action=redirect&bpo=24064].)

The `UserString` class now implements the `__getnewargs__()`, `__rmod__()`, `casefold()`, `format_map()`, `isprintable()`, and `maketrans()` methods to match the corresponding methods of `str`. (Contributed by Joe Jevnik in bpo-22189 [https://bugs.python.org/issue?@action=redirect&bpo=22189].)

collections.abc

The `Sequence.index()` method now accepts *start* and *stop* arguments to match the corresponding methods of `tuple`, `list`, etc. (Contributed by Devin Jeanpierre in bpo-23086 [https://bugs.python.org/issue?@action=redirect&bpo=23086].)

Una nueva clase de base abstracta `Generator`. (Contribución de Stefan Behnel en bpo-24018 [https://bugs.python.org/issue?@action=redirect&bpo=24018].)

Nuevas clases de base abstracta `Awaitable`, `Coroutine`, `AsyncIterator` y `AsyncIterable`. (Contribución de Yury Selivanov en bpo-24184 [https://bugs.python.org/issue?

@action=redirect&bpo=24184].)

For earlier Python versions, a backport of the new ABCs is available in an external PyPI package [https://pypi.org/project/backports_abc/].

compileall

Una nueva opción **`compileall`**, `-j` *N*, permite ejecutar *N* workers simultáneamente para realizar la compilación de bytecode en paralelo. La función **`compile_dir()`** tiene un parámetro `workers` correspondiente. (Contribución de Claudiu Popa en bpo-16104 [https://bugs.python.org/issue?@action=redirect&bpo=16104].)

Otra nueva opción, `-r`, permite controlar el nivel máximo de recursividad de los subdirectorios. (Contribución de Claudiu Popa en bpo-19628 [https://bugs.python.org/issue?@action=redirect&bpo=19628].)

Ahora la opción de línea de comando `-q` se puede especificar más de una vez, en cuyo caso todas las salidas, incluyendo errores, se suprimirán. Ahora el parámetro `quiet` correspondiente en **`compile_dir()`**, **`compile_file()`** y **`compile_path()`** puede aceptar un valor entero que indica el nivel de supresión de salida. (Contribución de Thomas Kluyver en bpo-21338 [https://bugs.python.org/issue?@action=redirect&bpo=21338].)

concurrent.futures

Ahora el método **`Executor.map()`** acepta un argumento *chunksize* para permitir el procesamiento por lotes de tareas para mejorar el desempeño cuando se usa **`ProcessPoolExecutor()`**. (Contribución de Dan O'Reilly en bpo-11271 [https://bugs.python.org/issue?@action=redirect&bpo=11271].)

Ahora el número de workers en el constructor **`ThreadPoolExecutor`** es opcional. El valor predeterminado es 5 veces el número de CPUs. (Contribución de Claudiu Popa en bpo-21527 [https://bugs.python.org/issue?@action=redirect&bpo=21527].)

configparser

Ahora **`configparser`** proporciona una forma de personalizar la conversión de valores al especificar un diccionario de convertidores en el constructor **`ConfigParser`** o al definirlos como métodos en las subclases `ConfigParser`. Los convertidores definidos en una instancia de analizador se heredan por sus proxies de sección.

Ejemplo:

```
>>> import configparser
>>> conv = {}
>>> conv['list'] = lambda v: [e.strip() for e in v.split
>>> cfg = configparser.ConfigParser(converters=conv)
>>> cfg.read_string("""
... [s]
... list = a b c d e f g
... """)
>>> cfg.get('s', 'list')
'a b c d e f g'
>>> cfg.getlist('s', 'list')
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> section = cfg['s']
>>> section.getlist('list')
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

(Contribución de Łukasz Langa en bpo-18159 [https://bugs.python.org/issue?@action=redirect&bpo=18159].)

contextlib

El nuevo context manager **`redirect_stderr()`** (similar a **`redirect_stdout()`**) facilita que los scripts de utilidad manejen APIs inflexibles que escriben su salida a **`sys.stderr`** y no proporcionen ninguna opción para redirigirlo:

```
>>> import contextlib, io, logging
>>> f = io.StringIO()
>>> with contextlib.redirect_stderr(f):
...     logging.warning('warning')
...
>>> f.getvalue()
```

```
'WARNING:root:warning\n'
```

(Contribución de Berker Peksag en bpo-22389 [https://bugs.python.org/issue?@action=redirect&bpo=22389].)

csv

Ahora el método `writerow()` admite iterables arbitrarios, no sólo secuencias. (Contribución de Serhiy Storchaka en bpo-23171 [https://bugs.python.org/issue?@action=redirect&bpo=23171].)

curses

The new `update_lines_cols()` function updates the `LINES` and `COLS` module variables. This is useful for detecting manual screen resizing. (Contributed by Arnon Yaari in bpo-4254 [https://bugs.python.org/issue?@action=redirect&bpo=4254].)

dbm

`dumb.open` siempre crea una nueva base de datos cuando la bandera tiene el valor `"n"`. (Contribución de Claudiu Popa en bpo-18039 [https://bugs.python.org/issue?@action=redirect&bpo=18039].)

difflib

Ahora el juego de caracteres de los documentos HTML generado por `HtmlDiff.make_file()` se puede personalizar usando un nuevo argumento de solo palabras clave *charset*. El juego de caracteres predeterminado del documento HTML cambió de `"ISO-8859-1"` a `"utf-8"`. (Contribución de Berker Peksag en bpo-2052 [https://bugs.python.org/issue?@action=redirect&bpo=2052].)

Ahora la función `diff_bytes()` puede comparar listas de bytes de cadenas de caracteres. Esto arregla una regresión de Python 2. (Contribución de Terry J. Reedy y Greg Ward en bpo-17445 [https://bugs.python.org/issue?@action=redirect&bpo=17445].)

distutils

Ahora los comandos `build` y `build_ext` aceptan una opción `-j` para habilitar la construcción paralela de módulos de extensión. (Contribución de Antoine Pitrou en bpo-5309 [https://bugs.python.org/issue?@action=redirect&bpo=5309].)

The `distutils` module now supports `xz` compression, and can be enabled by passing `xztar` as an argument to `bdist --format`. (Contributed by Serhiy Storchaka in bpo-16314 [https://bugs.python.org/issue?@action=redirect&bpo=16314].)

doctest

La función **`DocTestSuite()`** retorna una clase **`unittest.TestSuite`** vacía si *module* no contiene docstrings, en lugar de lanzar **`ValueError`**. (Contribución de Glenn Jones en bpo-15916 [https://bugs.python.org/issue?@action=redirect&bpo=15916].)

email

Una nueva opción de directiva **`Policy.mangle_from_`** controla si las líneas que empiecen o no con `"From "` en los cuerpos del correo electrónico tienen el prefijo de un caracter `">"` por generadores. El valor predeterminado es `True` para **`compat32`** y `False` para todas las demás directivas. (Contribución de Milan Oberkirch en bpo-20098 [https://bugs.python.org/issue?@action=redirect&bpo=20098].)

Un nuevo método **`Message.get_content_disposition()`** proporciona fácil acceso a un valor canónico para el encabezado *Content-Disposition*. (Contribución de Abhilash Raj en bpo-21083 [https://bugs.python.org/issue?@action=redirect&bpo=21083].)

Una nueva opción de directiva **`EmailPolicy.utf8`** se puede configurar en `True` para codificar los encabezados de correo electrónico usando el juego de caracteres UTF-8 en lugar de utilizar palabras codificadas. Esto permite formatear `Messages` de acuerdo a **RFC 6532** [https://datatracker.ietf.org/doc/html/rfc6532.html] y usarlos con un servidor SMTP que admita la extensión **RFC 6531** [https://datatracker.ietf.org/doc/html/rfc6531.html] `SMTPUTF8`. (Contribución de R. David Murray en bpo-24211 [https://bugs.python.org/issue?

@action=redirect&bpo=24211].)

Ahora el constructor `mime.text.MIMEText` admite una instancia de `charset.Charset`. (Contribución de Claude Paroz y Berker Peksag en bpo-16324 [https://bugs.python.org/issue?@action=redirect&bpo=16324].)

enum

La clase invocable `Enum` tiene un nuevo parámetro *start* para especificar el número inicial de los valores de enumeración si solo se proporcionan *names*:

```
>>> Animal = enum.Enum('Animal', 'cat dog', start=10)
>>> Animal.cat
<Animal.cat: 10>
>>> Animal.dog
<Animal.dog: 11>
```

(Contribución de Ethan Furman en bpo-21706 [https://bugs.python.org/issue?@action=redirect&bpo=21706].)

faulthandler

Ahora las funciones `enable()`, `register()`, `dump_traceback()` y `dump_traceback_later()` aceptan descriptores de archivo además de objetos similares a archivos. (Contribución de Wei Wu en bpo-23566 [https://bugs.python.org/issue?@action=redirect&bpo=23566].)

functools

Ahora la mayoría de la maquinaria de `lru_cache()` se implementa en C, lo que la hace significativamente más rápida. (Contribución de Matt Joiner, Alexey Kachayev y Serhiy Storchaka en bpo-14373 [https://bugs.python.org/issue?@action=redirect&bpo=14373].)

glob

Ahora las funciones `iglob()` y `glob()` admiten búsquedas recursivas en subdirectorios usando el patrón `"**"`. (Contribución de Serhiy Storchaka en bpo-13968 [https://bugs.python.org/issue?@action=redirect&bpo=13968].)

gzip

Ahora el argumento *mode* del constructor de **`GzipFile`** acepta `"x"` para solicitar la creación exclusiva. (Contribución de Tim Heaney en bpo-19222 [https://bugs.python.org/issue?@action=redirect&bpo=19222].)

heapq

Ahora la comparación de elementos en **`merge()`** se puede personalizar pasando una key function en un nuevo argumento de palabra clave opcional *key*, y un nuevo argumento de palabra clave opcional *reverse* se puede usar para revertir la comparación de elementos:

```
>>> import heapq
>>> a = ['9', '777', '55555']
>>> b = ['88', '6666']
>>> list(heapq.merge(a, b, key=len))
['9', '88', '777', '6666', '55555']
>>> list(heapq.merge(reversed(a), reversed(b), key=len,
['55555', '6666', '777', '88', '9']
```

(Contribución de Raymond Hettinger en bpo-13742 [https://bugs.python.org/issue?@action=redirect&bpo=13742].)

http

Una nueva enumeración **`HTTPStatus`** que define un conjunto de códigos de estado HTTP, frases de motivo y descripciones largas escritas en inglés. (Contribución de Demian Brecht en bpo-21793 [https://bugs.python.org/issue?@action=redirect&bpo=21793].)

http.client

Ahora **`HTTPConnection.getresponse()`** lanza una excepción

`RemoteDisconnected` cuando una conexión de servidor remoto se cierra inesperadamente. Además, si se lanza una excepción **`ConnectionError`** (del cual `RemoteDisconnected` es una subclase), ahora el socket del cliente se cierra automáticamente y se reconectará en la siguiente solicitud:

```
import http.client
conn = http.client.HTTPConnection('www.python.org')
for retries in range(3):
    try:
        conn.request('GET', '/')
        resp = conn.getresponse()
    except http.client.RemoteDisconnected:
        pass
```

(Contribución de Martin Panter en bpo-3566 [https://bugs.python.org/issue?@action=redirect&bpo=3566].)

idlelib e IDLE

Desde que idlelib implementa el shell y el editor IDLE y no se destina para ser importado por otros programas, obtiene mejoras con cada versión. Consultar `Lib/idlelib/NEWS.txt` para una lista acumulativa de cambios desde la 3.4.0, así como los cambios realizados en futuras versiones 3.5.x. Este archivo también está disponible en IDLE *Help* ‣ *About IDLE*.

imaplib

Ahora la clase **`IMAP4`** admite el protocolo context manager. Cuando se utiliza en una declaración **`with`**, el comando IMAP4 `LOGOUT` se llamará automáticamente al final del bloque. (Contribución de Tarek Ziadé y Serhiy Storchaka en bpo-4972 [https://bugs.python.org/issue?@action=redirect&bpo=4972].)

Ahora el módulo **`imaplib`** admite **RFC 5161** [https://datatracker.ietf.org/doc/html/rfc5161.html] (extensión ENABLE) y **RFC 6855** [https://datatracker.ietf.org/doc/html/rfc6855.html] (soporte UTF-8) a través del método **`IMAP4.enable()`**. Un nuevo atributo **`IMAP4.utf8_enabled`** rastrea si el soporte con **RFC 6855** [https://

datatracker.ietf.org/doc/html/rfc6855.html] está habilitado o no. (Contribución de Milan Oberkirch, R. David Murray y Maciej Szulik en bpo-21800 [https://bugs.python.org/issue?@action=redirect&bpo=21800].)

Ahora el módulo **`imaplib`** codifica automáticamente nombres de usuario y contraseñas de cadenas de caracteres no ASCII usando UTF-8, como recomiendan las RFCs. (Contribución de Milan Oberkirch en bpo-21800 [https://bugs.python.org/issue?@action=redirect&bpo=21800].)

imghdr

The **`what()`** function now recognizes the OpenEXR [https://www.openexr.com] format (contributed by Martin Vignali and Claudiu Popa in bpo-20295 [https://bugs.python.org/issue?@action=redirect&bpo=20295]), and the WebP [https://en.wikipedia.org/wiki/WebP] format (contributed by Fabrice Aneche and Claudiu Popa in bpo-20197 [https://bugs.python.org/issue?@action=redirect&bpo=20197].)

importlib

La clase **`util.LazyLoader`** permite la carga lenta de módulos en aplicaciones donde el tiempo de inicio es importante. (Contribución de Brett Cannon en bpo-17621 [https://bugs.python.org/issue?@action=redirect&bpo=17621].)

Ahora el método es **`abc.InspectLoader.source_to_code()`** es un método estático. Esto hace más fácil la inicialización de un objeto de módulo con código compilado a partir de una cadena de caracteres ejecutando `exec(code, module.__dict__)`. (Contribución de Brett Cannon en bpo-21156 [https://bugs.python.org/issue?@action=redirect&bpo=21156].)

Ahora la nueva función **`util.module_from_spec()`** es la forma preferida para crear un nuevo módulo. A diferencia de crear directamente una instancia **`types.ModuleType`**, esta nueva función configurará varios atributos controlados por importación basados en el objeto de especificación pasado. (Contribución de Brett Cannon en bpo-20383 [https://bugs.python.org/issue?@action=redirect&bpo=20383].)

inspect

Ahora las clases `Signature` y `Parameter` se pueden seleccionar y manipular. (Contribución de Yury Selivanov en bpo-20726 [https://bugs.python.org/issue?@action=redirect&bpo=20726] y bpo-20334 [https://bugs.python.org/issue?@action=redirect&bpo=20334].)

Un nuevo método `BoundArguments.apply_defaults()` proporciona una forma de establecer valores predeterminados para los argumentos faltantes:

```
>>> def foo(a, b='ham', *args): pass
>>> ba = inspect.signature(foo).bind('spam')
>>> ba.apply_defaults()
>>> ba.arguments
OrderedDict([('a', 'spam'), ('b', 'ham'), ('args', ())])
```

(Contribución de Yury Selivanov en bpo-24190 [https://bugs.python.org/issue?@action=redirect&bpo=24190].)

Un nuevo método de clase `Signature.from_callable()` facilita la subclasificación de `Signature`. (Contribución de Yury Selivanov y Eric Snow en bpo-17373 [https://bugs.python.org/issue?@action=redirect&bpo=17373].)

Ahora la función `signature()` acepta un argumento de palabra clave opcional *follow_wrapped*, el cual, cuando se establece en `False`, deshabilita el seguimiento automático de enlaces `__wrapped__`. (Contribución de Yury Selivanov en bpo-20691 [https://bugs.python.org/issue?@action=redirect&bpo=20691].)

Se ha agregado un conjunto de nuevas funciones para inspeccionar funciones corrutina y objetos corrutina: `iscoroutine()`, `iscoroutinefunction()`, `isawaitable()`, `getcoroutinelocals()` y `getcoroutinestate()`. (Contribución de Yury Selivanov en bpo-24017 [https://bugs.python.org/issue?@action=redirect&bpo=24017] y bpo-24400 [https://bugs.python.org/issue?@action=redirect&bpo=24400].)

Ahora las funciones `stack()`, `trace()`, `getouterframes()` y `getinnerframes()` retornan una lista de tuplas con nombre.

(Contribución de Daniel Shahaf en bpo-16808 [https://bugs.python.org/issue?@action=redirect&bpo=16808].)

io

Un nuevo método **`BufferedIOBase.readinto1()`**, que usa como mucho una llamada al flujo sin procesar subyacente de los métodos **`RawIOBase.read()`** o **`RawIOBase.readinto()`**. (Contribución de Nikolaus Rath en bpo-20578 [https://bugs.python.org/issue?@action=redirect&bpo=20578].)

ipaddress

Ahora las clases **`IPv4Network`** y **`IPv6Network`** aceptan un argumento de tupla `(address, netmask)`, para construir fácilmente objetos de red a partir de direcciones existentes:

```
>>> import ipaddress
>>> ipaddress.IPv4Network(('127.0.0.0', 8))
IPv4Network('127.0.0.0/8')
>>> ipaddress.IPv4Network(('127.0.0.0', '255.0.0.0'))
IPv4Network('127.0.0.0/8')
```

(Contribución de Peter Moody y Antoine Pitrou en bpo-16531 [https://bugs.python.org/issue?@action=redirect&bpo=16531].)

A new **`reverse_pointer`** attribute for the **`IPv4Address`** and **`IPv6Address`** classes returns the name of the reverse DNS PTR record:

```
>>> import ipaddress
>>> addr = ipaddress.IPv4Address('127.0.0.1')
>>> addr.reverse_pointer
'1.0.0.127.in-addr.arpa'
>>> addr6 = ipaddress.IPv6Address('::1')
>>> addr6.reverse_pointer
'1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
```

(Contribución de Leon Weber en bpo-20480 [https://bugs.python.org/issue?@action=redirect&bpo=20480].)

json

Ahora la interfaz de la línea de comandos **`json.tool`** conserva el orden de los nombres en los objetos JSON pasados en la entrada. La nueva opción `--sort-keys` se puede usar para ordenar los nombres alfabéticamente. (Contribución de Berker Peksag en bpo-21650 [https://bugs.python.org/issue?@action=redirect&bpo=21650].)

Ahora el decodificador JSON lanza **`JSONDecodeError`** en lugar de **`ValueError`** para proporcionar una mejor información de contexto sobre el error. (Contribución de Serhiy Storchaka en bpo-19361 [https://bugs.python.org/issue?@action=redirect&bpo=19361].)

linecache

Una nueva función **`lazycache()`** se puede usar para capturar información sobre un módulo no basado en archivos para permitir obtener sus líneas más tarde a través de **`getline()`**. Esto evita hacer E/S hasta que una línea sea realmente necesaria, sin tener que llevar los módulos globales indefinidamente. (Contribución de Robert Collins en bpo-17911 [https://bugs.python.org/issue?@action=redirect&bpo=17911].)

locale

Una nueva función **`delocalize()`** se puede usar para convertir una cadena de caracteres en una cadena numérica normalizada, teniendo la configuración `LC_NUMERIC` en cuenta:

```
>>> import locale
>>> locale.setlocale(locale.LC_NUMERIC, 'de_DE.UTF-8')
'de_DE.UTF-8'
>>> locale.delocalize('1.234,56')
'1234.56'
>>> locale.setlocale(locale.LC_NUMERIC, 'en_US.UTF-8')
'en_US.UTF-8'
>>> locale.delocalize('1,234.56')
'1234.56'
```

(Contribución de Cédric Krier en bpo-13918 [https://bugs.python.org/

issue?@action=redirect&bpo=13918].)

logging

Ahora todos los métodos del módulo logging (**`Logger log()`**, **`exception()`**, **`critical()`**, **`debug()`**, etc.), aceptan instancias de excepción como un argumento *exc_info*, además de valores booleanos y tuplas de excepción:

```
>>> import logging
>>> try:
...     1/0
... except ZeroDivisionError as ex:
...     logging.error('exception', exc_info=ex)
ERROR:root:exception
```

(Contribución de Yury Selivanov en bpo-20537 [https://bugs.python.org/issue?@action=redirect&bpo=20537].)

Ahora la clase **`handlers.HTTPHandler`** acepta una instancia opcional **`ssl.SSLContext`** para establecer la configuración SSL usada en una conexión HTTP. (Contribución de Alex Gaynor en bpo-22788 [https://bugs.python.org/issue?@action=redirect&bpo=22788].)

Ahora la clase **`handlers.QueueListener`** toma un argumento de palabra clave *respect_handler_level*, el cual, si se establece en `True`, pasará mensajes a gestores teniendo en cuenta los gestores de niveles. (Contribución de Vinay Sajip.)

lzma

Ahora el método **`LZMADecompressor.decompress()`** acepta un argumento opcional *max_length* para limitar el tamaño máximo de datos descomprimidos. (Contribución de Martin Panter en bpo-15955 [https://bugs.python.org/issue?@action=redirect&bpo=15955].)

math

Se agregaron dos nuevas contantes en el módulo **`math`**: **`inf`** y **`nan`**. (Contribución de Mark Dickinson en bpo-23185 [https://

bugs.python.org/issue?@action=redirect&bpo=23185].)

Una nueva función `isclose()` proporciona una forma de comprobar la igualdad aproximada. (Contribución de Chris Barker y Tal Einat en bpo-24270 [https://bugs.python.org/issue?@action=redirect&bpo=24270].)

A new `gcd()` function has been added. The `fractions.gcd()` function is now deprecated. (Contributed by Mark Dickinson and Serhiy Storchaka in bpo-22486 [https://bugs.python.org/issue?@action=redirect&bpo=22486].)

multiprocessing

Ahora los objetos `sharedctypes.synchronized()` admiten el protocolo context manager. (Contribución de Charles-François Natali en bpo-21565 [https://bugs.python.org/issue?@action=redirect&bpo=21565].)

operator

Ahora los objetos `attrgetter()`, `itemgetter()` y `methodcaller()` admiten pickling. (Contribución de Josh Rosenberg y Serhiy Storchaka en bpo-22955 [https://bugs.python.org/issue?@action=redirect&bpo=22955].)

Las nuevas funciones `matmul()` y `imatmul()` para realizar multiplicación de matrices. (Contribución de Benjamin Peterson en bpo-21176 [https://bugs.python.org/issue?@action=redirect&bpo=21176].)

os

Se agregó la nueva función `scandir()` que retorna un iterador de `DirEntry`. Si es posible, `scandir()` extrae los atributos del archivo mientras escanea un directorio, eliminando la necesidad de realizar llamadas posteriores al sistema para determinar el tipo de archivo o los atributos, lo que puede mejorar significativamente el rendimiento. (Contribución de Ben Hoyt con la ayuda de Victor Stinner en bpo-22524 [https://bugs.python.org/issue?@action=redirect&bpo=22524].)

En Windows, ahora un nuevo atributo `stat_result.st_file_attributes` está disponible. Corresponde al miembro `dwFileAttributes` de la estructura `BY_HANDLE_FILE_INFORMATION` retornado por `GetFileInformationByHandle()`. (Contribución de Ben Hoyt en bpo-21719 [https://bugs.python.org/issue?@action=redirect&bpo=21719].)

Ahora la función `urandom()` usa la llamada al sistema `getrandom()` en Linux 3.17 o versiones más recientes, y `getentropy()` en OpenBSD 5.6 y versiones más recientes, eliminando la necesidad de utilizar `/dev/urandom` y evitando fallas debido al posible agotamiento del descriptor de archivo. (Contribución de Victor Stinner en bpo-22181 [https://bugs.python.org/issue?@action=redirect&bpo=22181].)

New `get_blocking()` and `set_blocking()` functions allow getting and setting a file descriptor's blocking mode (`O_NONBLOCK`.) (Contributed by Victor Stinner in bpo-22054 [https://bugs.python.org/issue?@action=redirect&bpo=22054].)

Ahora las funciones `truncate()` y `ftruncate()` se admiten en Windows. (Contribución de Steve Dower en bpo-23668 [https://bugs.python.org/issue?@action=redirect&bpo=23668].)

Hay una nueva función `os.path.commonpath()` que retorna la sub-ruta común más larga de cada nombre de ruta pasado. A diferencia de la función `os.path.commonprefix()`, siempre retorna una ruta válida:

```
>>> os.path.commonprefix(['/usr/lib', '/usr/local/lib'])
'/usr/l'

>>> os.path.commonpath(['/usr/lib', '/usr/local/lib'])
'/usr'
```

(Contribución de Rafik Draoui y Serhiy Storchaka en bpo-10395 [https://bugs.python.org/issue?@action=redirect&bpo=10395].)

pathlib

El nuevo método `Path.samefile()` se puede usar para verificar si la ruta apunta al mismo archivo que otra ruta, el cual puede ser otro objeto **`Path`** o una cadena de caracteres:

```
>>> import pathlib
>>> p1 = pathlib.Path('/etc/hosts')
>>> p2 = pathlib.Path('/etc/../etc/hosts')
>>> p1.samefile(p2)
True
```

(Contribución de Vajrasky Kok y Antoine Pitrou en bpo-19775 [https://bugs.python.org/issue?@action=redirect&bpo=19775].)

Ahora el método **`Path.mkdir()`** acepta un nuevo argumento opcional *exist_ok* para que coincida con la funcionalidad `mkdir -p` y **`os.makedirs()`**. (Contribución de Berker Peksag en bpo-21539 [https://bugs.python.org/issue?@action=redirect&bpo=21539].)

Hay un nuevo método **`Path.expanduser()`** para expandir los prefijos `~` y `~user`. (Contribución de Serhiy Storchaka y Claudiu Popa en bpo-19776 [https://bugs.python.org/issue?@action=redirect&bpo=19776].)

Una nueva clase de método **`Path.home()`** se puede usar para obtener una instancia **`Path`** que representa el directorio de inicio del usuario. (Contribución de Victor Salgado y Mayank Tripathi en bpo-19777 [https://bugs.python.org/issue?@action=redirect&bpo=19777].)

Los nuevos métodos **`Path.write_text()`**, **`Path.read_text()`**, **`Path.write_bytes()`**, **`Path.read_bytes()`** para simplificar la lectura/escritura de operaciones en los archivos.

El siguiente fragmento de código creará o reescribirá el archivo existente `~/spam42`:

```
>>> import pathlib
>>> p = pathlib.Path('~/spam42')
>>> p.expanduser().write_text('ham')
3
```

(Contribución de Christopher Welborn en bpo-20218 [https://

bugs.python.org/issue?@action=redirect&bpo=20218].)

pickle

Ahora los objetos anidados, como los métodos independientes o las clases anidadas, se pueden serializar con pickle usando protocolos pickle anteriores al protocolo de la versión 4. El protocolo de la versión 4 ya admite estos casos. (Contribución de Serhiy Storchaka en bpo-23611 [https://bugs.python.org/issue?@action=redirect&bpo=23611].)

poplib

Un nuevo comando **`POP3.utf8()`** habilita el soporte (correo electrónico internacionalizado) **RFC 6856** [https://datatracker.ietf.org/doc/html/rfc6856.html], si un servidor POP lo admite. (Contribución de Milan Oberkirch en bpo-21804 [https://bugs.python.org/issue?@action=redirect&bpo=21804].)

re

Ahora las referencias y referencias condicionales a grupos con longitud fija se permiten en afirmaciones retrospectivas:

```
>>> import re
>>> pat = re.compile(r'(a|b).(?<=\1)c')
>>> pat.match('aac')
<_sre.SRE_Match object; span=(0, 3), match='aac'>
>>> pat.match('bbc')
<_sre.SRE_Match object; span=(0, 3), match='bbc'>
```

(Contribución de Serhiy Storchaka en bpo-9179 [https://bugs.python.org/issue?@action=redirect&bpo=9179].)

El número de captura de grupos en expresiones regulares ya no se limita a 100. (Contribución de Serhiy Storchaka en bpo-22437 [https://bugs.python.org/issue?@action=redirect&bpo=22437].)

Ahora las funciones **`sub()`** y **`subn()`** reemplazan grupos que no coinciden con cadenas de caracteres vacías en lugar de lanzar una excepción. (Contribución de Serhiy Storchaka en bpo-1519638

[https://bugs.python.org/issue?@action=redirect&bpo=1519638].)

The `re.error` exceptions have new attributes, `msg`, `pattern`, `pos`, `lineno`, and `colno`, that provide better context information about the error:

```
>>> re.compile("""
...     (?x)
...     .++
... """)
Traceback (most recent call last):
   ...
sre_constants.error: multiple repeat at position 16 (lir
```

(Contribución de Serhiy Storchaka en bpo-22578 [https://bugs.python.org/issue?@action=redirect&bpo=22578].)

readline

Una nueva función `append_history_file()` se puede usar para adjuntar el número especificado de los elementos finales en el historial al archivo dado. (Contribución de Bruno Cauet en bpo-22940 [https://bugs.python.org/issue?@action=redirect&bpo=22940].)

selectors

La nueva clase `DevpollSelector` admite sondeos eficientes `/dev/poll` en Solaris. (Contribución de Giampaolo Rodola” en bpo-18931 [https://bugs.python.org/issue?@action=redirect&bpo=18931].)

shutil

Ahora la función `move()` acepta un argumento *copy_function*, permitiendo, por ejemplo, que se use la función `copy()` en lugar de la predeterminada `copy2()` si es necesario ignorar metadatos de archivo al moverlo. (Contribución de Claudiu Popa en bpo-19840 [https://bugs.python.org/issue?@action=redirect&bpo=19840].)

Ahora la función `make_archive()` admite el formato *xztar*. (Contribución de Serhiy Storchaka en bpo-5411 [https://

bugs.python.org/issue?@action=redirect&bpo=5411].)

signal

En Windows, ahora la función **`set_wakeup_fd()`** también admite identificadores de socket. (Contribución de Victor Stinner en bpo-22018 [https://bugs.python.org/issue?@action=redirect&bpo=22018].)

Varias constantes `SIG*` en el módulo **`signal`** se convirtieron en **`Enums`**. Esto permite que los nombres significativos se impriman durante la depuración en lugar de «números mágicos» enteros. (Contribución de Giampaolo Rodola" en bpo-21076 [https://bugs.python.org/issue?@action=redirect&bpo=21076].)

smtpd

Both the **`SMTPServer`** and **`SMTPChannel`** classes now accept a *decode_data* keyword argument to determine if the `DATA` portion of the SMTP transaction is decoded using the `"utf-8"` codec or is instead provided to the **`SMTPServer.process_message()`** method as a byte string. The default is `True` for backward compatibility reasons, but will change to `False` in Python 3.6. If *decode_data* is set to `False`, the `process_message` method must be prepared to accept keyword arguments. (Contributed by Maciej Szulik in bpo-19662 [https://bugs.python.org/issue?@action=redirect&bpo=19662].)

The **`SMTPServer`** class now advertises the `8BITMIME` extension (**RFC 6152** [https://datatracker.ietf.org/doc/html/rfc6152.html]) if *decode_data* has been set `True`. If the client specifies `BODY=8BITMIME` on the `MAIL` command, it is passed to **`SMTPServer.process_message()`** via the *mail_options* keyword. (Contributed by Milan Oberkirch and R. David Murray in bpo-21795 [https://bugs.python.org/issue?@action=redirect&bpo=21795].)

The **`SMTPServer`** class now also supports the `SMTPUTF8` extension (**RFC 6531** [https://datatracker.ietf.org/doc/html/rfc6531.html]: Internationalized Email). If the client specified `SMTPUTF8 BODY=8BITMIME` on the `MAIL` command, they are passed to **`SMTPServer.process_message()`** via the *mail_options* keyword.

It is the responsibility of the `process_message` method to correctly handle the `SMTPUTF8` data. (Contributed by Milan Oberkirch in bpo-21725 [https://bugs.python.org/issue?@action=redirect&bpo=21725].)

It is now possible to provide, directly or via name resolution, IPv6 addresses in the **`SMTPServer`** constructor, and have it successfully connect. (Contributed by Milan Oberkirch in bpo-14758 [https://bugs.python.org/issue?@action=redirect&bpo=14758].)

smtplib

Un nuevo método **`SMTP.auth()`** proporciona una forma conveniente de implementar mecanismos de autenticación personalizados. (Contribución de Milan Oberkirch en bpo-15014 [https://bugs.python.org/issue?@action=redirect&bpo=15014].)

Ahora el método **`SMTP.set_debuglevel()`** acepta un nivel de depuración adicional (2), el cual habilita las marcas de tiempo en los mensajes de depuración. (Contribución de Gavin Chappell y Maciej Szulik en bpo-16914 [https://bugs.python.org/issue?@action=redirect&bpo=16914].)

Ahora los métodos **`SMTP.sendmail()`** y **`SMTP.send_message()`** admiten **RFC 6531** [https://datatracker.ietf.org/doc/html/rfc6531.html] (SMTPUTF8). (Contribución de Milan Oberkirch y R. David Murray en bpo-22027 [https://bugs.python.org/issue?@action=redirect&bpo=22027].)

sndhdr

The **`what()`** and **`whathdr()`** functions now return a **`namedtuple()`**. (Contributed by Claudiu Popa in bpo-18615 [https://bugs.python.org/issue?@action=redirect&bpo=18615].)

socket

Ahora las funciones con tiempo de espera usan un reloj monótono en lugar de un reloj del sistema. (Contribución de Victor Stinner en bpo-22043 [https://bugs.python.org/issue?@action=redirect&bpo=22043].)

Un nuevo método `socket.sendfile()` permite enviar un archivo a través de un socket usando la función de alto rendimiento `os.sendfile()` en UNIX, lo que hace que las cargas sean de 2 a 3 veces más rápido que cuando se usa el método simple `socket.send()`. (Contribución de Giampaolo Rodola" en bpo-17552 [https://bugs.python.org/issue?@action=redirect&bpo=17552].)

El método `socket.sendall()` ya no reinicia el tiempo de espera del socket cada vez que se reciben o se envían bytes. Ahora el tiempo de espera del socket es la duración total máxima para enviar todos los datos. (Contribución de Victor Stinner en bpo-23853 [https://bugs.python.org/issue?@action=redirect&bpo=23853].)

Ahora el argumento *backlog* del método `socket.listen()` es opcional. De forma predeterminada, se establece en `SOMAXCONN` o en `128`, lo que sea menor. (Contribución de Charles-François Natali en bpo-21455 [https://bugs.python.org/issue?@action=redirect&bpo=21455].)

ssl

Soporte de memoria BIO

(Contribución de Geert Jansen en bpo-21965 [https://bugs.python.org/issue?@action=redirect&bpo=21965].)

Se ha agregado la nueva clase `SSLObject` para proporcionar soporte del protocolo SSL para los casos en que las capacidades de E/S de red de `SSLSocket` no son necesarias o no son óptimas. `SSLObject` representa una instancia del protocolo SSL pero no implementa ningún método de E/S de red y, en su lugar, proporciona una interfaz de búfer de memoria. La nueva clase `MemoryBIO` se puede usar para pasar datos entre Python y una instancia del protocolo SSL.

El soporte de la memoria BIO SSL está destinado principalmente para ser usado en frameworks que implementan E/S asíncronas para las cuales el modelo de preparación («select/poll») de `SSLSocket` es ineficiente.

Un nuevo método `SSLContext.wrap_bio()` se puede usar para crear una nueva instancia `SSLObject`.

Soporte de negociación de protocolo de capa de aplicación

(Contribución de Benjamin Peterson en bpo-20188 [https://bugs.python.org/issue?@action=redirect&bpo=20188].)

Donde está presente el soporte de OpenSSL, ahora el módulo `ssl` implementa la extensión TLS *Application-Layer Protocol Negotiation* como se describe en **RFC 7301** [https://datatracker.ietf.org/doc/html/rfc7301.html].

El nuevo método `SSLContext.set_alpn_protocols()` se puede usar para especificar qué protocolos deben anunciar un socket durante el protocolo de enlace TLS.

The new `SSLSocket.selected_alpn_protocol()` returns the protocol that was selected during the TLS handshake. The `HAS_ALPN` flag indicates whether ALPN support is present.

Otros cambios

Hay un nuevo método `SSLSocket.version()` para consultar la versión actual del protocolo en uso. (Contribución de Antoine Pitrou en bpo-20421 [https://bugs.python.org/issue?@action=redirect&bpo=20421].)

The `SSLSocket` class now implements a `SSLSocket.sendfile()` method. (Contributed by Giampaolo Rodola" in bpo-17552 [https://bugs.python.org/issue?@action=redirect&bpo=17552].)

The `SSLSocket.send()` method now raises either the `ssl.SSLWantReadError` or `ssl.SSLWantWriteError` exception on a non-blocking socket if the operation would block. Previously, it would return `0`. (Contributed by Nikolaus Rath in bpo-20951 [https://bugs.python.org/issue?@action=redirect&bpo=20951].)

Ahora la función `cert_time_to_seconds()` interpreta la hora de entrada como UTC y no como hora local, por **RFC 5280** [https://datatracker.ietf.org/doc/html/rfc5280.html]. Además, el valor de retorno siempre es un `int`. (Contribución de Akira Li en bpo-19940 [https://bugs.python.org/issue?@action=redirect&bpo=19940].)

New **`SSLObject.shared_ciphers()`** and `SSLSocket.shared_ciphers()` methods return the list of ciphers sent by the client during the handshake. (Contributed by Benjamin Peterson in bpo-23186 [https://bugs.python.org/issue?@action=redirect&bpo=23186].)

The `SSLSocket.do_handshake()`, `SSLSocket.read()`, **`SSLSocket.shutdown()`**, and `SSLSocket.write()` methods of the `SSLSocket` class no longer reset the socket timeout every time bytes are received or sent. The socket timeout is now the maximum total duration of the method. (Contributed by Victor Stinner in bpo-23853 [https://bugs.python.org/issue?@action=redirect&bpo=23853].)

The **`match_hostname()`** function now supports matching of IP addresses. (Contributed by Antoine Pitrou in bpo-23239 [https://bugs.python.org/issue?@action=redirect&bpo=23239].)

sqlite3

Ahora la clase `Row` admite totalmente el protocolo de secuencia, en particular la iteración `reversed()` y la indexación segmentada. (Contribución de Claudiu Popa en bpo-10203 [https://bugs.python.org/issue?@action=redirect&bpo=10203]; de Lucas Sinclair, Jessica McKellar, y Serhiy Storchaka en bpo-13583 [https://bugs.python.org/issue?@action=redirect&bpo=13583].)

subprocess

Se agregó la nueva función `run()`. Ejecuta el comando especificado y retorna un objeto `CompletedProcess`, el cual describe un proceso terminado. La nueva API es más consistente y es el enfoque recomendado para invocar subprocesos en código Python que no necesita mantener la compatibilidad con versiones anteriores de Python. (Contribución de Thomas Kluyver en bpo-23342 [https://bugs.python.org/issue?@action=redirect&bpo=23342].)

Ejemplos:

```
>>> subprocess.run(["ls", "-l"])  # doesn't capture outp
CompletedProcess(args=['ls', '-l'], returncode=0)

>>> subprocess.run("exit 1", shell=True, check=True)
Traceback (most recent call last):
  ...
subprocess.CalledProcessError: Command 'exit 1' returnec

>>> subprocess.run(["ls", "-l", "/dev/null"], stdout=sub
CompletedProcess(args=['ls', '-l', '/dev/null'], returnc
stdout=b'crw-rw-rw- 1 root root 1, 3 Jan 23 16:23 /dev/r
```

sys

A new **`set_coroutine_wrapper()`** function allows setting a global hook that will be called whenever a coroutine object is created by an **`async def`** function. A corresponding **`get_coroutine_wrapper()`** can be used to obtain a currently set wrapper. Both functions are provisional, and are intended for debugging purposes only. (Contributed by Yury Selivanov in bpo-24017 [https://bugs.python.org/issue?@action=redirect&bpo=24017].)

Una nueva función **`is_finalizing()`** se puede usar para verificar si el intérprete de Python se está apagando. (Contribución de Antoine Pitrou en bpo-22696 [https://bugs.python.org/issue?@action=redirect&bpo=22696].)

sysconfig

Ahora el nombre del directorio de los scripts del usuario en Windows incluye los dos primeros componentes de la versión de Python. (Contribución de Paul Moore en bpo-23437 [https://bugs.python.org/issue?@action=redirect&bpo=23437].)

tarfile

Ahora el argumento *mode* de la función **`open()`** acepta `"x"` para solicitar una creación exclusiva. (Contribución de Berker Peksag en bpo-21717 [https://bugs.python.org/issue?@action=redirect&bpo=21717].)

Ahora los métodos **`TarFile.extractall()`** y **`TarFile.extract()`** toman un argumento de palabra clave *numeric_owner*. Si se establece como `True`, los archivos y directorios extraídos serán propiedad de los números `uid` y `gid` del archivo tar. Si se establece como `False` (el valor predeterminado y el comportamiento en versiones anteriores a la 3.5), serán propiedad del usuario y el grupo nombrados en el archivo tar. (Contribución de Michael Vogt y Eric Smith en bpo-23193 [https://bugs.python.org/issue?@action=redirect&bpo=23193].)

Ahora **`TarFile.list()`** acepta un argumento de palabra clave opcional *members* que se puede establecer en un subconjunto de la lista retornada por **`TarFile.getmembers()`**. (Contribución de Serhiy Storchaka en bpo-21549 [https://bugs.python.org/issue?@action=redirect&bpo=21549].)

threading

Ahora los métodos **`Lock.acquire()`** y **`RLock.acquire()`** usan un reloj monótono para la administración del tiempo de espera. (Contribución de Victor Stinner en bpo-22043 [https://bugs.python.org/issue?@action=redirect&bpo=22043].)

time

Ahora la función **`monotonic()`** siempre está disponible. (Contribución de Victor Stinner en bpo-22043 [https://bugs.python.org/issue?@action=redirect&bpo=22043].)

timeit

Una nueva opción de línea de comando `-u` o `--unit=U` se puede usar para especificar la unidad de tiempo para la salida del temporizador. Las opciones que se admiten son `usec`, `msec` o `sec`. (Contribución de Julian Gindi en bpo-18983 [https://bugs.python.org/issue?@action=redirect&bpo=18983].)

La función **`timeit()`** tiene un nuevo parámetro *globals* para especificar el espacio de nombres donde el código se ejecutará. (Contribución de Ben Roberts en bpo-2527 [https://bugs.python.org/

issue?@action=redirect&bpo=2527].)

tkinter

The `tkinter._fix` module used for setting up the Tcl/Tk environment on Windows has been replaced by a private function in the `_tkinter` module which makes no permanent changes to environment variables. (Contributed by Zachary Ware in bpo-20035 [https://bugs.python.org/issue?@action=redirect&bpo=20035].)

traceback

New `walk_stack()` and `walk_tb()` functions to conveniently traverse frame and traceback objects. (Contributed by Robert Collins in bpo-17911 [https://bugs.python.org/issue?@action=redirect&bpo=17911].)

Nuevas clases ligeras: `TracebackException`, `StackSummary` y `FrameSummary`. (Contribución de Robert Collins en bpo-17911 [https://bugs.python.org/issue?@action=redirect&bpo=17911].)

Ahora las funciones `print_tb()` y `print_stack()` admiten valores negativos para el argumento *limit*. (Contribución de Dmitry Kazakov en bpo-22619 [https://bugs.python.org/issue?@action=redirect&bpo=22619].)

types

Una nueva función `coroutine()` para transformar un iterador generador y objetos `generator-like` en aguardables. (Contribución de Yury Selivanov en bpo-24017 [https://bugs.python.org/issue?@action=redirect&bpo=24017].)

Un nuevo tipo llamado `CoroutineType`, el cual se usa para objetos coroutine creados por funciones `async def`. (Contribución de Yury Selivanov en bpo-24400 [https://bugs.python.org/issue?@action=redirect&bpo=24400].)

unicodedata

El módulo **`unicodedata`** ahora usa datos de Unicode 8.0.0 [https://unicode.org/versions/Unicode8.0.0/].

unittest

Ahora el método **`TestLoader.loadTestsFromModule()`** acepta un solo argumento de palabra clave *pattern* que se pasa a `load_tests` como el tercer argumento. Ahora los paquetes encontrados se verifican en busca de `load_tests` independientemente de si su ruta coincide con *pattern*, porque es imposible que el nombre de un paquete coincida con el modelo predeterminado. (Contribución de Robert Collins y Barry A. Warsaw en bpo-16662 [https://bugs.python.org/issue?@action=redirect&bpo=16662].)

Ahora los errores de descubrimiento de unittest se exponen en el atributo **`TestLoader.errors`** de la instancia **`TestLoader`**. (Contribución de Robert Collins en bpo-19746 [https://bugs.python.org/issue?@action=redirect&bpo=19746].)

Una nueva opción de línea de comando `--locals` muestra variables locales en rastreo. (Contribución de Robert Collins en bpo-22936 [https://bugs.python.org/issue?@action=redirect&bpo=22936].)

unittest.mock

La clase **`Mock`** tiene las siguientes mejoras:

- El constructor de la clase tiene un nuevo parámetro *unsafe*, que hace que los objetos simulados lancen **`AttributeError`** en los nombres de atributo que comienzan con `"assert"`. (Contribución de Kushal Das en bpo-21238 [https://bugs.python.org/issue?@action=redirect&bpo=21238].)
- Un nuevo método **`Mock.assert_not_called()`** para verificar si se llamó el objeto simulado. (Contribución de Kushal Das en bpo-21262 [https://bugs.python.org/issue?@action=redirect&bpo=21262].)

The **`MagicMock`** class now supports **`__truediv__()`**, **`__divmod__()`** and **`__matmul__()`** operators. (Contributed by Johannes Baiter in bpo-20968 [https://bugs.python.org/issue?

@action=redirect&bpo=20968], and Håkan Lövdahl in bpo-23581 [https://bugs.python.org/issue?@action=redirect&bpo=23581] and bpo-23568 [https://bugs.python.org/issue?@action=redirect&bpo=23568].)

Ya no es necesario pasar explícitamente `create=True` a la función **`patch()`** cuando se parchean los nombres incorporados. (Contribución de Kushal Das en bpo-17660 [https://bugs.python.org/issue?@action=redirect&bpo=17660].)

urllib

Una nueva clase **`request.HTTPPasswordMgrWithPriorAuth`** permite que se administren las credenciales de autenticación básica HTTP para eliminar el manejo innecesario de respuestas `401` o enviar credenciales incondicionalmente en la primera solicitud para comunicarse con los servidores que retornen una respuesta `404` en lugar de un `401` si el encabezado `Authorization` no se envía. (Contribución de Matej Cepl en bpo-19494 [https://bugs.python.org/issue?@action=redirect&bpo=19494] y de Akshit Khurana en bpo-7159 [https://bugs.python.org/issue?@action=redirect&bpo=7159].)

Un nuevo argumento *quote_via* para la función **`parse.urlencode()`** proporciona una forma de controlar la codificación de las partes de la consulta si es necesario. (Contribución de Samwyse y Arnon Yaari en bpo-13866 [https://bugs.python.org/issue?@action=redirect&bpo=13866].)

La función **`request.urlopen()`** acepta un objeto **`ssl.SSLContext`** como un argumento *context*, el cual se usará para la conexión HTTPS. (Contribución de Alex Gaynor en bpo-22366 [https://bugs.python.org/issue?@action=redirect&bpo=22366].)

Se actualizó **`parse.urljoin()`** para usar la semántica **RFC 3986** [https://datatracker.ietf.org/doc/html/rfc3986.html] para la resolución de URL relativas, en lugar de **RFC 1808** [https://datatracker.ietf.org/doc/html/rfc1808.html] y **RFC 2396** [https://datatracker.ietf.org/doc/html/rfc2396.html]. (Contribución de Demian Brecht y Senthil Kumaran en bpo-22118 [https://bugs.python.org/issue?@action=redirect&bpo=22118].)

wsgiref

Ahora el argumento *headers* del constructor de la clase `headers.Headers` es opcional. (Contribución de Pablo Torres Navarrete y SilentGhost en bpo-5800 [https://bugs.python.org/issue?@action=redirect&bpo=5800].)

xmlrpc

Ahora la clase `client.ServerProxy` admite el protocolo context manager. (Contribución de Claudiu Popa en bpo-20627 [https://bugs.python.org/issue?@action=redirect&bpo=20627].)

Ahora el constructor de `client.ServerProxy` acepta una instancia opcional `ssl.SSLContext`. (Contribución de Alex Gaynor en bpo-22960 [https://bugs.python.org/issue?@action=redirect&bpo=22960].)

xml.sax

Ahora los analizadores SAX admiten un flujo de caracteres del objeto `xmlreader.InputSource`. (Contribución de Serhiy Storchaka en bpo-2175 [https://bugs.python.org/issue?@action=redirect&bpo=2175].)

Ahora `parseString()` acepta una instancia `str`. (Contribución de Serhiy Storchaka en bpo-10590 [https://bugs.python.org/issue?@action=redirect&bpo=10590].)

zipfile

Ahora la salida ZIP se puede escribir en flujos inescrutables. (Contribución de Serhiy Storchaka en bpo-23252 [https://bugs.python.org/issue?@action=redirect&bpo=23252].)

Ahora el argumento *mode* del método `ZipFile.open()` acepta `"x"` para solicitar una creación exclusiva. (Contribución de Serhiy Storchaka en bpo-21717 [https://bugs.python.org/issue?@action=redirect&bpo=21717].)

Otros cambios a nivel de módulo

Many functions in the **`mmap`**, **`ossaudiodev`**, **`socket`**, **`ssl`**, and **`codecs`** modules now accept writable bytes-like objects. (Contributed by Serhiy Storchaka in bpo-23001 [https://bugs.python.org/issue?@action=redirect&bpo=23001].)

Optimizaciones

La función **`os.walk()`** se aceleró de 3 a 5 veces en los sistemas POSIX y de 7 a 20 veces en Windows. Esto se hizo usando la nueva función **`os.scandir()`**, el cual expone la información del archivo de las llamadas al sistema `readdir` o `FindFirstFile/FindNextFile`. (Contribución de Ben Hoyt con ayuda de Victor Stinner en bpo-23605 [https://bugs.python.org/issue?@action=redirect&bpo=23605].)

La construcción de `bytes(int)` (llenado con cero bytes) es más rápida y usa menos memoria para objetos grandes. Se usa `calloc()` en lugar de `malloc()` para asignar memoria para estos objetos. (Contribución de Victor Stinner en bpo-21233 [https://bugs.python.org/issue?@action=redirect&bpo=21233].)

Algunas operaciones en **`ipaddress`** **`IPv4Network`** y **`IPv6Network`** se aceleraron masivamente, tal como **`subnets()`**, **`supernet()`**, **`summarize_address_range()`**, **`collapse_addresses()`**. La velocidad puede variar de 3 a 15 veces. (Contribución de Antoine Pitrou, Michel Albert y Markus en bpo-21486 [https://bugs.python.org/issue?@action=redirect&bpo=21486], bpo-21487 [https://bugs.python.org/issue?@action=redirect&bpo=21487], bpo-20826 [https://bugs.python.org/issue?@action=redirect&bpo=20826], bpo-23266 [https://bugs.python.org/issue?@action=redirect&bpo=23266].)

Se optimizó pickling de objetos **`ipaddress`** para producir una salida significativamente menor. (Contribución de Serhiy Storchaka en bpo-23133 [https://bugs.python.org/issue?@action=redirect&bpo=23133].)

Ahora muchas operaciones en **`io.BytesIO`** son de 50% a 100% más rápidas. (Contribución de Serhiy Storchaka en bpo-15381 [https://bugs.python.org/issue?@action=redirect&bpo=15381] y de David Wilson en bpo-22003 [https://bugs.python.org/issue?@action=redirect&bpo=22003].)

Ahora la función **`marshal.dumps()`** es más rápida: 65–85% en las versiones 3 y 4, 20–25% en las versiones 0 a 2 con datos típicos, y hasta 5 veces en el mejor de los casos. (Contribución de Serhiy Storchaka en bpo-20416 [https://bugs.python.org/issue?@action=redirect&bpo=20416] y bpo-23344 [https://bugs.python.org/issue?@action=redirect&bpo=23344].)

Ahora el codificador UTF-32 es de 3 a 7 veces más rápido. (Contribución de Serhiy Storchaka en bpo-15027 [https://bugs.python.org/issue?@action=redirect&bpo=15027].)

Ahora las expresiones regulares se analizan hasta un 10% más rápido. (Contribución de Serhiy Storchaka en bpo-19380 [https://bugs.python.org/issue?@action=redirect&bpo=19380].)

Se optimizó la función **`json.dumps()`** para ejecutarse con `ensure_ascii=False` tan rápido como con `ensure_ascii=True`. (Contribución de Naoki Inada en bpo-23206 [https://bugs.python.org/issue?@action=redirect&bpo=23206].)

Las funciones **`PyObject_IsInstance()`** y **`PyObject_IsSubclass()`** se aceleraron en el caso común de que el segundo argumento tenga **`type`** como su metaclase. (Contribución de Georg Brandl en bpo-22540 [https://bugs.python.org/issue?@action=redirect&bpo=22540].)

Se mejoró ligeramente el almacenamiento en caché del método, lo que arrojó una mejora del rendimiento de hasta un 5% en algunos puntos de referencia. (Contribución de Antoine Pitrou en bpo-22847 [https://bugs.python.org/issue?@action=redirect&bpo=22847].)

Ahora los objetos del módulo **`random`** usan un 50% menos de memoria en compilaciones de 64 bits. (Contribución de Serhiy Storchaka en bpo-23488 [https://bugs.python.org/issue?@action=redirect&bpo=23488].)

Las llamadas getter de **`property()`** son hasta un 25% más rápidas. (Contribución de Joe Jevnik en bpo-23910 [https://bugs.python.org/issue?@action=redirect&bpo=23910].)

Ahora la instanciación de **`fractions.Fraction`** es hasta un 30%

más rápida. (Contribución de Stefan Behnel en bpo-22464 [https://bugs.python.org/issue?@action=redirect&bpo=22464].)

Ahora los métodos de cadena de caracteres **`find()`**, **`rfind()`**, **`split()`**, **`partition()`** y el operador **`in`** son significativamente más rápidos para buscar subcadenas de 1 carácter. (Contribución de Serhiy Storchaka en bpo-23573 [https://bugs.python.org/issue?@action=redirect&bpo=23573].)

Cambios en la compilación y la API de C

Se agregaron las nuevas funciones `calloc`:

- **`PyMem_RawCalloc()`**,
- **`PyMem_Calloc()`**,
- **`PyObject_Calloc()`**.

(Contribución de Victor Stinner en bpo-21233 [https://bugs.python.org/issue?@action=redirect&bpo=21233].)

Nuevas funciones auxiliares de codificación/decodificación:

- **`Py_DecodeLocale()`** (reemplazó `_Py_char2wchar()`),
- **`Py_EncodeLocale()`** (reemplazó `_Py_wchar2char()`).

(Contribución de Victor Stinner en bpo-18395 [https://bugs.python.org/issue?@action=redirect&bpo=18395].)

Una nueva función **`PyCodec_NameReplaceErrors()`** para reemplazar el error de codificación unicode con `\N{...}` escapes. (Contribución de Serhiy Storchaka en bpo-19676 [https://bugs.python.org/issue?@action=redirect&bpo=19676].)

A new **`PyErr_FormatV()`** function similar to **`PyErr_Format()`**, but accepts a **`va_list`** argument. (Contributed by Antoine Pitrou in bpo-18711 [https://bugs.python.org/issue?@action=redirect&bpo=18711].)

Una nueva excepción **`PyExc_RecursionError`**. (Contribución de Georg Brandl en bpo-19235 [https://bugs.python.org/issue?@action=redirect&bpo=19235].)

Se introdujeron las nuevas funciones `PyModule_FromDefAndSpec()`, `PyModule_FromDefAndSpec2()` y `PyModule_ExecDef()` por **PEP 489** [https://peps.python.org/pep-0489/] – inicialización del módulo de extensión multifase. (Contribución de Petr Viktorin en bpo-24268 [https://bugs.python.org/issue?@action=redirect&bpo=24268].)

Nuevas funciones `PyNumber_MatrixMultiply()` y `PyNumber_InPlaceMatrixMultiply()` para realizar la multiplicación de matrices. (Contribución de Benjamin Peterson en bpo-21176 [https://bugs.python.org/issue?@action=redirect&bpo=21176]. Consultar también **PEP 465** [https://peps.python.org/pep-0465/] para más detalles.)

Ahora el espacio `PyTypeObject.tp_finalize` es parte de la ABI estable.

Windows builds now require Microsoft Visual C++ 14.0, which is available as part of Visual Studio 2015 [https://visualstudio.microsoft.com/en/vs/older-downloads/#visual-studio-2015-and-other-products].

Ahora los módulos de extensión incluyen una etiqueta de información de la plataforma en su nombre de archivo en algunas plataformas (la etiqueta es opcional y CPython importará las extensiones sin ella, aunque si la etiqueta está presente y no coincide, no se cargará la extensión):

- En Linux, los nombres de archivo del módulo de extensión termina con `.cpython-<major><minor>m-<architecture>-<os>.pyd`:
    - `<major>` es el número principal de la versión de Python; para Python 3.5 este es `3`.
    - `<minor>` es el número menor de la versión de Python; para Python 3.5 este es `5`.
    - `<architecture>` es la arquitectura de hardware para la que se construyó el módulo de extensión. Es más común `i386` para plataformas Intel de 32 bits o `x86_64` para plataformas Intel (y AMD) de 64 bits.
    - `<os>` siempre es `linux-gnu`, excepto para las extensiones que se crearon para comunicarse con la ABI

de 32 bits en plataformas de 64 bits, en cuyo caso es `linux-gnu32` (y `<architecture>` será `x86_64`).

- En Windows, los nombres de archivo del módulo de extensión termina con `<debug>.cp<major><minor>-<platform>.pyd`:
  - ○ `<major>` es el número principal de la versión de Python; para Python 3.5 este es `3`.
  - ○ `<minor>` es el número menor de la versión de Python; para Python 3.5 este es `5`.
  - ○ `<platform>` es la plataforma para la que se construyó el módulo de extensión, ya sea `win32` para Win32, `win_amd64` para Win64, `win_ia64` para Windows Itanium 64 y `win_arm` para Windows con ARM.
  - ○ Si está integrado en el modo de depuración, `<debug>` será `_d`, de lo contrario estará en blanco.
- En plataformas OS X, ahora los nombres de archivo del módulo de extensión terminan con `-darwin.so`.
- En todas las demás plataformas, los nombres de archivo del módulo de extensión son los mismos que en Python 3.4.

Obsoleto

Nuevas palabras clave

No se recomienda usarse `async` y `await` como nombres de variable, clase, función o módulo. Se introdujo por **PEP 492** [https://peps.python.org/pep-0492/] en Python 3.5, se convertirán en palabras clave adecuadas en Python 3.7.

Comportamiento obsoleto de Python

Al lanzar la excepción **`StopIteration`** dentro de un generador ahora se lanzará un **`PendingDeprecationWarning`** silencioso, el cual se convertirá en una advertencia de deprecación no silenciosa en Python 3.6 y activará un **`RuntimeError`** en Python 3.7. Consultar PEP 479: Cambiar el gestor de StopIteration dentro de generadores para más detalles.

Sistemas operativos no compatibles

Windows XP ya no es compatible con Microsoft, por lo tanto, según **PEP 11** [https://peps.python.org/pep-0011/], CPython 3.5 oficialmente ya no es compatible con este sistema operativo.

Módulos, funciones y métodos obsoletos de Python

The **`formatter`** module has now graduated to full deprecation and is still slated for removal in Python 3.6.

The **`asyncio.async()`** function is deprecated in favor of **`ensure_future()`**.

The **`smtpd`** module has in the past always decoded the DATA portion of email messages using the `utf-8` codec. This can now be controlled by the new *decode_data* keyword to **`SMTPServer`**. The default value is `True`, but this default is deprecated. Specify the *decode_data* keyword with an appropriate value to avoid the deprecation warning.

Está obsoleto asignar valores directamente en **`key`**, **`value`** y **`coded_value`** en los objetos de **`http.cookies.Morsel`**. Usar en su lugar el método **`set()`**. Además, el parámetro no documentado *LegalChars* de **`set()`** está obsoleto y ahora se ignora.

Pasar una cadena de caracteres de formato como argumento de palabra clave *format_string* al método **`format()`** de la clase **`string.Formatter`** se ha quedado obsoleto. (Contribución de Serhiy Storchaka en bpo-23671 [https://bugs.python.org/issue?@action=redirect&bpo=23671].)

The **`platform.dist()`** and **`platform.linux_distribution()`** functions are now deprecated. Linux distributions use too many different ways of describing themselves, so the functionality is left to a package. (Contributed by Vajrasky Kok and Berker Peksag in bpo-1322 [https://bugs.python.org/issue?@action=redirect&bpo=1322].)

Los métodos `from_function` y `from_builtin` previamente no documentados de **`inspect.Signature`** están obsoletos. Utilizar en su lugar el nuevo método **`Signature.from_callable()`**. (Contribución de Yury Selivanov en bpo-24248 [https://

bugs.python.org/issue?@action=redirect&bpo=24248].)

The `inspect.getargspec()` function is deprecated and scheduled to be removed in Python 3.6. (See bpo-20438 [https://bugs.python.org/issue?@action=redirect&bpo=20438] for details.)

The `inspect getfullargspec()`, `getcallargs()`, and `formatargspec()` functions are deprecated in favor of the `inspect.signature()` API. (Contributed by Yury Selivanov in bpo-20438 [https://bugs.python.org/issue?@action=redirect&bpo=20438].)

Las funciones `getargvalues()` y `formatargvalues()` se marcaron inadvertidamente como obsoletas con el lanzamiento de Python 3.5.0.

Ahora el uso de la bandera `re.LOCALE` con patrones str o `re.ASCII` está obsoleto. (Contribución de Serhiy Storchaka en bpo-22407 [https://bugs.python.org/issue?@action=redirect&bpo=22407].)

El uso de secuencias especiales no reconocidas que consisten en `'\'` y una letra ASCII en patrones de expresión regular y patrones de reemplazo ahora lanza una advertencia de deprecación y estará prohibido en Python 3.6. (Contribución de Serhiy Storchaka en bpo-23622 [https://bugs.python.org/issue?@action=redirect&bpo=23622].)

Ahora el argumento predeterminado no documentado y no oficial *use_load_tests* del método `unittest.TestLoader.loadTestsFromModule()` está obsoleto e ignorado. (Contribución de Robert Collins y Barry A. Warsaw en bpo-16662 [https://bugs.python.org/issue?@action=redirect&bpo=16662].)

Eliminado

APIs y características eliminadas

Se eliminaron las siguientes APIs y características obsoletas y anteriormente obsoletas:

- Se eliminó el atributo `__version__` del paquete de correo electrónico. No se envió el código de correo electrónico por separado de stdlib durante mucho tiempo, y la cadena de caracteres `__version__` no se actualizó en las últimas versiones.

- La clase interna `Netrc` en el módulo **`ftplib`** quedó obsoleta en la versión 3.4 y ahora se eliminó. (Contribución de Matt Chaput en bpo-6623 [https://bugs.python.org/issue?@action=redirect&bpo=6623].)
- Se eliminó el concepto de los archivos `.pyo`.
- La clase JoinableQueue en el módulo provisional **`asyncio`** quedó obsoleto en la versión 3.4.4 y ahora se eliminó. (Contribución de A. Jesse Jiryu Davis en bpo-23464 [https://bugs.python.org/issue?@action=redirect&bpo=23464].)

Portando a Python 3.5

Esta sección enumera cambios descritos anteriormente y otras correcciones de errores que pueden requerir cambios en tu código.

Cambios en el comportamiento de Python

- Debido a un descuido, erróneamente en anteriores versiones de Python aceptaron la siguiente sintaxis:

  ```
  f(1 for x in [1], *args)
  f(1 for x in [1], **kwargs)
  ```

  Ahora Python 3.5 lanza correctamente un **`SyntaxError`**, ya que las expresiones generadoras deben ponerse entre paréntesis si no son el único argumento de una función.

Cambios en la API de Python

- **PEP 475** [https://peps.python.org/pep-0475/]: Ahora se reintentan las llamadas al sistema cuando se interrumpen por una señal en lugar de lanzar **`InterruptedError`** si el gestor de señales de Python no lanza una excepción.
- Antes de Python 3.5, se consideraba falso un objeto **`datetime.time`** si representaba la medianoche en UTC. Este

comportamiento se consideraba oscuro y propenso a errores y se eliminó en Python 3.5. Consultar bpo-13936 [https://bugs.python.org/issue?@action=redirect&bpo=13936] para más detalles.

- The **`ssl.SSLSocket.send()`** method now raises either **`ssl.SSLWantReadError`** or **`ssl.SSLWantWriteError`** on a non-blocking socket if the operation would block. Previously, it would return `0`. (Contributed by Nikolaus Rath in bpo-20951 [https://bugs.python.org/issue?@action=redirect&bpo=20951].)
- Ahora el atributo `__name__` de los generadores se establece desde el nombre de la función en lugar de establecerse desde el nombre del código. Usar `gen.gi_code.co_name` para recuperar el nombre del código. También los generadores tienen un nuevo atributo `__qualname__`, el nombre calificado, el cual se utiliza ahora para la representación de un generador (`repr(gen)`). (Contribución de Victor Stinner en bpo-21205 [https://bugs.python.org/issue?@action=redirect&bpo=21205].)
- The deprecated «strict» mode and argument of **`HTMLParser`**, **`HTMLParser.error()`**, and the **`HTMLParserError`** exception have been removed. (Contributed by Ezio Melotti in bpo-15114 [https://bugs.python.org/issue?@action=redirect&bpo=15114].) The *convert_charrefs* argument of **`HTMLParser`** is now `True` by default. (Contributed by Berker Peksag in bpo-21047 [https://bugs.python.org/issue?@action=redirect&bpo=21047].)
- Aunque no es parte formalmente de la API, vale la pena señalar con fines de portabilidad (es decir: pruebas de reparación) que ahora los mensajes de error que anteriormente tenían el formato «"sometype" no es compatible con el protocolo búfer» son del formato «se requiere bytes-like object, no "sometype"». (Contribución de Ezio Melotti en bpo-16518 [https://bugs.python.org/issue?@action=redirect&bpo=16518].)
- Si el directorio actual se establece en un directorio que ya no existe entonces ya no se lanzará **`FileNotFoundError`** y en su lugar **`find_spec()`** retornará `None` **sin** almacenamiento en caché `None` en **`sys.path_importer_cache`**, el cual es diferente al caso típico (bpo-22834 [https://bugs.python.org/issue?

@action=redirect&bpo=22834]).

- El código de estado HTTP y los mensajes de `http.client` y `http.server` se refactorizaron en una enumeración `HTTPStatus`. Los valores en **`http.client`** y **`http.server`** permanecen disponibles para compatibilidad con versiones anteriores. (Contribución de Demian Brecht en bpo-21793 [https://bugs.python.org/issue?@action=redirect&bpo=21793].)
- When an import loader defines `exec_module()` it is now expected to also define `create_module()` (raises a `DeprecationWarning` now, will be an error in Python 3.6). If the loader inherits from `importlib.abc.Loader` then there is nothing to do, else simply define **`create_module()`** to return `None`. (Contributed by Brett Cannon in bpo-23014 [https://bugs.python.org/issue?@action=redirect&bpo=23014].)
- La función `re.split()` siempre ignoraba las coincidencias de patrones vacías, así que el patrón `"x*"` funcionaba igual que `"x+"` y el patrón `"\b"` nunca funcionó. Ahora **`re.split()`** lanza una advertencia si el patrón puede coincidir con una cadena de caracteres vacía. Para compatibilidad, usar patrones que nunca coincidan con una cadena vacía (p. ej. `"x+"` en lugar de `"x*"`). Ahora los patrones que solo pueden coincidir con una cadena vacía (como `"\b"`) lanzan un error. (Contribución de Serhiy Storchaka en bpo-22818 [https://bugs.python.org/issue?@action=redirect&bpo=22818].)
- La interfaz similar a diccionarios `http.cookies.Morsel` se ha hecho autoconsistente: ahora la comparación de morsel toma en cuenta `key` y `value`, ahora `copy()` da como resultado en una instancia **`Morsel`** en lugar de un `dict`, y ahora `update()` lanzará una excepción si alguna de las claves del diccionario de actualización no es válida. Además, el parámetro no documentado *LegalChars* de `set()` está obsoleto y ahora se ignora. (Contribución de Demian Brecht en bpo-2211 [https://bugs.python.org/issue?@action=redirect&bpo=2211].)
- **PEP 488** [https://peps.python.org/pep-0488/] eliminó los archivos `.pyo` de Python e introdujo la etiqueta opcional `opt-` en los nombres de archivo `.pyc`. El

`importlib.util.cache_from_source()` ha obtenido un parámetro *optimization* para ayudar a controlar la etiqueta `opt-`. Debido a esto, el parámetro *debug_override* de la función ahora está en desuso. Los archivos `.pyo` ya no se admiten como argumento de archivo para el intérprete de Python y, por lo tanto, no sirven para nada cuando se distribuyen por sí solos (es decir, distribución de código sin código fuente). Debido al hecho de que el número mágico para el código de bytes ha cambiado en Python 3.5, todos los archivos `.pyo` antiguos de versiones anteriores de Python no son válidos independientemente de este PEP.

- The `socket` module now exports the `CAN_RAW_FD_FRAMES` constant on linux 3.6 and greater.
- Ahora la función `ssl.cert_time_to_seconds()` interpreta la hora de entrada como UTC y no como hora local, por RFC 5280 [https://datatracker.ietf.org/doc/html/rfc5280.html]. Además, el valor de retorno siempre es un `int`. (Contribución de Akira Li en bpo-19940 [https://bugs.python.org/issue?@action=redirect&bpo=19940].)
- Ahora la herramienta `pygettext.py` usa el formato estándar +NNNN para zonas horarias en el encabezado POT-Creation-Date.
- The `smtplib` module now uses `sys.stderr` instead of the previous module-level `stderr` variable for debug output. If your (test) program depends on patching the module-level variable to capture the debug output, you will need to update it to capture sys.stderr instead.
- Los métodos `str.startswith()` y `str.endswith()` ya no retornan `True` cuando encuentran la cadena de caracteres vacía y los índices están completamente fuera de rango. (Contribución de Serhiy Storchaka en bpo-24284 [https://bugs.python.org/issue?@action=redirect&bpo=24284].)
- Ahora la función `inspect.getdoc()` retorna cadenas de documentación que se heredaron de clases base. Ya no es necesario duplicar las cadenas de documentación si la documentación que se heredó es apropiada. Para suprimir una cadena que se heredó, se debe especificar una cadena vacía (o se puede completar la documentación). Este cambio afecta la salida del módulo `pydoc` y la función `help()`. (Contribución de Serhiy Storchaka en bpo-15582 [https://

bugs.python.org/issue?@action=redirect&bpo=15582].)

- Ahora las llamadas anidadas **`functools.partial()`** se aplanan. Si se confiaba en el comportamiento anterior, ahora se puede agregar un atributo a un objeto **`functools.partial()`** o se puede crear una subclase de **`functools.partial()`**. (Contribución de Alexander Belopolsky en bpo-7830 [https://bugs.python.org/issue?@action=redirect&bpo=7830].)

Cambios en la API de C

- The undocumented **`format`** member of the (non-public) **`PyMemoryViewObject`** structure has been removed. All extensions relying on the relevant parts in `memoryobject.h` must be rebuilt.
- The **`PyMemAllocator`** structure was renamed to **`PyMemAllocatorEx`** and a new `calloc` field was added.
- Removed non-documented macro **`PyObject_REPR()`** which leaked references. Use format character `%R` in **`PyUnicode_FromFormat()`**-like functions to format the **`repr()`** of the object. (Contributed by Serhiy Storchaka in bpo-22453 [https://bugs.python.org/issue?@action=redirect&bpo=22453].)
- Because the lack of the **`__module__`** attribute breaks pickling and introspection, a deprecation warning is now raised for builtin types without the **`__module__`** attribute. This will be an **`AttributeError`** in the future. (Contributed by Serhiy Storchaka in bpo-20204 [https://bugs.python.org/issue?@action=redirect&bpo=20204].)
- As part of the **PEP 492** [https://peps.python.org/pep-0492/] implementation, the `tp_reserved` slot of **`PyTypeObject`** was replaced with a **`tp_as_async`** slot. Refer to Objetos corrutina for new types, structures and functions.

Cambios notables en Python 3.5.4

Nuevo objetivo de compilación `make regen-all`

Para simplificar la compilación cruzada y asegurar que CPython se pueda compilar de manera confiable sin requerir que una versión existente de Python ya esté disponible, el sistema de compilación basado en herramientas automáticas ya no intenta volver a compilar implícitamente los archivos generados en función de los tiempos de modificación de los archivos.

En su lugar, se agregó un nuevo comando `make regen-all` para forzar la regeneración de estos archivos cuando se desee (p. ej. después de que ya se haya creado una versión inicial de Python basada en las versiones pregeneradas).

También se definen objetivos de regeneración más selectivos - consultar Makefile.pre.in [https://github.com/python/cpython/tree/3.14/Makefile.pre.in] para más detalles.

(Contribución de Victor Stinner en bpo-23404 [https://bugs.python.org/issue?@action=redirect&bpo=23404].)

Added in version 3.5.4.

Eliminación del objetivo de compilación `make touch`

Se eliminó el objetivo de compilación `make touch` que anteriormente se usaba para solicitar la regeneración implícita de los archivos generados mediante la actualización de sus tiempos de modificación.

Se reemplazó por el nuevo objetivo `make regen-all`.

(Contribución de Victor Stinner en bpo-23404 [https://bugs.python.org/issue?@action=redirect&bpo=23404].)

Distinto en la versión 3.5.4.

Qué hay de nuevo en Python 3.4

Autor:

R. David Murray <rdmurray@bitdance.com> (Editor)

Este artículo explica las nuevas características de Python 3.4, en comparación con la 3.3. Python 3.4 se publicó el 16 de marzo de 2014. Para obtener todos los detalles, consulte el changelog [https://docs.python.org/3.4/whatsnew/changelog.html].

Ver también

PEP 429 [https://peps.python.org/pep-0429/] – Calendario de lanzamiento de Python 3.4

Resumen – Puntos destacados del lanzamiento

Nuevas características de sintaxis:

- En Python 3.4 no se han añadido nuevas características sintácticas.

Otras novedades:

- pip debe estar siempre disponible (**PEP 453** [https://peps.python.org/pep-0453/]).
- Los descriptores de archivo recién creados no son heredables (**PEP 446** [https://peps.python.org/pep-0446/]).
- opción de línea de comandos para modo aislado (bpo-16499 [https://bugs.python.org/issue?@action=redirect&bpo=16499]).

- mejoras en el manejo de códecs que no son codificaciones de texto (múltiples problemas).improvements>.
- Un tipo de ModuleSpec para el sistema de importación (**PEP 451** [https://peps.python.org/pep-0451/]). (Afecta a los autores de los importadores)
- El formato **`marshal`** se ha hecho más compacto y eficiente (bpo-16475 [https://bugs.python.org/issue?@action=redirect&bpo=16475]).

Nuevos módulos de la biblioteca:

- **`asyncio`**: Nueva API provisional para IO asíncrona (**PEP 3156** [https://peps.python.org/pep-3156/]).
- **`ensurepip`**: Bootstrapping the pip installer (**PEP 453** [https://peps.python.org/pep-0453/]).
- **`enum`**: Soporte para tipos de enumeración (**PEP 435** [https://peps.python.org/pep-0435/]).
- **`pathlib`**: Rutas del sistema de archivos orientadas a objetos (**PEP 428** [https://peps.python.org/pep-0428/]).
- **`selectors`**: Multiplexación de E/S eficiente y de alto nivel, construida sobre las primitivas del módulo **`select`** (parte de **PEP 3156** [https://peps.python.org/pep-3156/]).
- **`statistics`**: Una básica numerically stable statistics library (**PEP 450** [https://peps.python.org/pep-0450/]).
- **`tracemalloc`**: Trace Python memory allocations (**PEP 454** [https://peps.python.org/pep-0454/]).

Mejora significativa de los módulos de la biblioteca:

- Funciones genéricas de despacho único en **`functools`** (**PEP 443** [https://peps.python.org/pep-0443/]).
- Nuevo **`pickle`** protocolo 4 (**PEP 3154** [https://peps.python.org/pep-3154/]).
- **`multiproceso`** tiene ahora una opción para evitar el uso de os.fork en Unix (bpo-8713 [https://bugs.python.org/issue?@action=redirect&bpo=8713]).
- **`email`** has a new submodule, **`contentmanager`**, and a new **`Message`** subclass (**`EmailMessage`**) that simplify MIME handling (bpo-18891 [https://bugs.python.org/issue?@action=redirect&bpo=18891]).
- Los módulos **`inspect`** y **`pydoc`** son ahora capaces de

introspeccionar correctamente una variedad mucho más amplia de objetos invocables, lo que mejora la salida del sistema Python `help()`.
- La API del módulo `ipaddress` ha sido declarada estable

Mejoras en la seguridad:

- Algoritmo hash seguro e intercambiable (**PEP 456** [https://peps.python.org/pep-0456/]).
- Hacer no heredables los descriptores de archivo recién creados (**PEP 446** [https://peps.python.org/pep-0446/]) para evitar la filtración de descriptores de archivo a los procesos hijos.
- Nueva opción de línea de comandos para modo aislado, (bpo-16499 [https://bugs.python.org/issue?@action=redirect&bpo=16499]).
- `multiprocessing` tiene ahora una opción para evitar el uso de os.fork en Unix. *spawn* y *forkserver* son más seguros porque evitan compartir datos con los procesos hijos.
- Los procesos hijos de `multiprocessing` en Windows ya no heredan todos los manejadores heredables del padre, sólo los necesarios.
- Una nueva función `hashlib.pbkdf2_hmac()` proporciona la función de derivación de clave basada en contraseña PKCS#5 2 [https://en.wikipedia.org/wiki/PBKDF2].
- Soporte de TLSv1.1 y TLSv1.2 para `ssl`.
- Recuperación de certificados del soporte del almacén de certificados del sistema Windows para `ssl`.
- Soporte SNI (Server Name Indication) del lado del servidor para `ssl`.
- La clase `ssl.SSLContext` tiene un mucho que mejorar.
- All modules in the standard library that support SSL now support server certificate verification, including hostname matching (`ssl.match_hostname()`) and CRLs (Certificate Revocation lists, see `ssl.SSLContext.load_verify_locations()`).

Mejoras en la implementación de CPython:

- Finalización segura de objetos (**PEP 442** [https://peps.python.org/pep-0442/]).
- Aprovechando **PEP 442** [https://peps.python.org/pep-0442/], en la

mayoría de los casos los globales de los módulos ya no se establecen como Ninguno durante la finalización (bpo-18214 [https://bugs.python.org/issue?@action=redirect&bpo=18214]).
- Asignadores de memoria configurable (**PEP 445** [https://peps.python.org/pep-0445/]).
- Consultorio de argumentos (**PEP 436** [https://peps.python.org/pep-0436/]).

Por favor, siga leyendo para ver una lista completa de los cambios orientados al usuario, incluyendo muchas otras mejoras menores, optimizaciones de CPython, elementos obsoletos y posibles problemas de portabilidad.

Nuevas Funciones

PEP 453: Arranque explícito de PIP en instalaciones de Python

Puesta en marcha de pip por defecto

El nuevo módulo **`ensurepip`** (definido en **PEP 453** [https://peps.python.org/pep-0453/]) proporciona un mecanismo estándar multiplataforma para arrancar el instalador pip en instalaciones de Python y entornos virtuales. La versión de `pip` incluida con Python 3.4.0 es `pip` 1.5.4, y las futuras versiones de mantenimiento de 3.4.x actualizarán la versión incluida a la última versión de `pip` que esté disponible en el momento de crear la versión candidata.

Por defecto, los comandos `pipX` y `pipX.Y` se instalarán en todas las plataformas (donde X.Y representa la versión de la instalación de Python), junto con el paquete de Python `pip` y sus dependencias. En Windows y en los entornos virtuales de todas las plataformas, también se instalará el comando `pip` sin versionar. En otras plataformas, el comando `pip` no versionado en todo el sistema suele referirse a la versión de Python 2 instalada por separado.

La utilidad de línea de comandos `pyvenv` y el módulo **`venv`** hacen uso del módulo **`ensurepip`** para que `pip` esté disponible en entornos virtuales. Cuando se utiliza la utilidad de línea de comandos, `pip` se instala por defecto, mientras que cuando se utiliza el módulo **`venv`** API se debe solicitar explícitamente la instalación de `pip`.

Para las construcciones de CPython source en sistemas POSIX, los comandos `make install` y `make altinstall` arrancan `pip` por defecto. Este comportamiento puede ser controlado a través de las opciones de configuración, y anulado a través de las opciones del Makefile.

En Windows y Mac OS X, los instaladores de CPython ahora instalan por defecto `pip` junto con el propio CPython (los usuarios pueden optar por no instalarlo durante el proceso de instalación). Los usuarios de Windows tendrán que optar por las modificaciones automáticas del `PATH` para que `pip` esté disponible por defecto desde la línea de comandos, de lo contrario se puede acceder a través del lanzador de Python para Windows como `py -m pip`.

Como los empaquetadores de la plataforma **discussed in the PEP** [https://peps.python.org/pep-0453/#recommendations-for-downstream-distributors] pueden elegir no instalar estos comandos de forma predeterminada, siempre que, cuando se invoquen, proporcionen instrucciones claras y simples sobre cómo instalarlos en esa plataforma (generalmente usando el administrador de paquetes del sistema).

Nota

Para evitar conflictos entre instalaciones paralelas de Python 2 y Python 3, sólo los comandos versionados `pip3` y `pip3.4` son arrancados por defecto cuando se invoca `ensurepip` directamente - la opción `--default-pip` es necesaria para solicitar también el comando no versionado `pip`. `pyvenv` y el instalador de Windows aseguran que el comando `pip` sin calificar esté disponible en esos entornos, y `pip` siempre puede ser invocado a través del interruptor `-m` en lugar de directamente para evitar la ambigüedad en sistemas con múltiples instalaciones de Python.

Cambios en la documentación

Como parte de este cambio, las secciones Installing Python modules y Distribuir módulos de Python de la documentación han sido completamente rediseñadas como breves documentos de inicio y preguntas frecuentes. La mayor parte de la documentación de empaquetado se ha trasladado a la «Guía del usuario de empaquetado de Python» mantenida por la Autoridad de Empaquetado de Python <https://packaging.python.org>`_ y a la documentación de los proyectos individuales.

Sin embargo, como esta migración está aún incompleta, las versiones anteriores de esas guías siguen disponibles como Building C and C++ Extensions with setuptools y Building C and C++ Extensions with setuptools.

Ver también

PEP 453 [https://peps.python.org/pep-0453/] – Arranque explícito de pip en instalaciones de Python

PEP escrito por Donald Stufft y Nick Coghlan, implementado por Donald Stufft, Nick Coghlan, Martin von Löwis y Ned Deily.

PEP 446: Los descriptores de archivos recién creados no son heredables

PEP 446 [https://peps.python.org/pep-0446/] hace que los descriptores de archivo recién creados sean no heredables. En general, este es el comportamiento que una aplicación querrá: al lanzar un nuevo proceso, tener archivos actualmente abiertos también en el nuevo proceso puede llevar a todo tipo de errores difíciles de encontrar, y potencialmente a problemas de seguridad.

Sin embargo, hay ocasiones en las que se desea la herencia. Para dar soporte a estos casos, están disponibles las siguientes funciones y métodos nuevos:

- `os.get_inheritable()`, `os.set_inheritable()`

- `os.get_handle_inheritable()`, `os.set_handle_inheritable()`
- `socket.socket.get_inheritable()`, `socket.socket.set_inheritable()`

Ver también

PEP 446 [https://peps.python.org/pep-0446/] – Hacer que los descriptores de archivo recién creados no sean heredables
PEP escrito y aplicado por Victor Stinner.

Mejoras en el manejo de códecs

Desde que se introdujo por primera vez, el módulo `codecs` siempre ha pretendido funcionar como un sistema de codificación y decodificación dinámica de tipo neutro. Sin embargo, su estrecho acoplamiento con el modelo de texto de Python, especialmente los métodos de conveniencia restringidos a los tipos `str`, `bytes` y `bytearray`, ha ocultado históricamente este hecho.

Como paso clave para aclarar la situación, las funciones de conveniencia `codecs.encode()` y `codecs.decode()` están ahora debidamente documentadas en Python 2.7, 3.3 y 3.4. Estas funciones han existido en el módulo `codecs` (y han sido cubiertas por el conjunto de pruebas de regresión) desde Python 2.4, pero anteriormente sólo se podían descubrir a través de la introspección en tiempo de ejecución.

A diferencia de los métodos de conveniencia de `str`, `bytes` y `bytearray`, las funciones de conveniencia de `codecs` soportan codificaciones arbitrarias tanto en Python 2 como en Python 3, en lugar de limitarse a codificaciones de texto Unicode (en Python 3) o a <->conversiones de `basestring` (en Python 2).

En Python 3.4, el intérprete es capaz de identificar las codificaciones no textuales conocidas proporcionadas en la biblioteca estándar y dirigir a los usuarios hacia estas funciones de conveniencia de propósito general cuando sea apropiado:

```
>>> b"abcdef".decode("hex")
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
LookupError: 'hex' is not a text encoding; use codecs.de

>>> "hello".encode("rot13")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
LookupError: 'rot13' is not a text encoding; use codecs.

>>> open("foo.txt", encoding="hex")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
LookupError: 'hex' is not a text encoding; use codecs.op
```

En un cambio relacionado, siempre que sea factible sin romper la compatibilidad con versiones anteriores, las excepciones planteadas durante las operaciones de codificación y decodificación se envuelven en una excepción encadenada del mismo tipo que menciona el nombre del códec responsable de producir el error:

```
>>> import codecs

>>> codecs.decode(b"abcdefgh", "hex")
Traceback (most recent call last):
  File "/usr/lib/python3.4/encodings/hex_codec.py", line
    return (binascii.a2b_hex(input), len(input))
binascii.Error: Non-hexadecimal digit found

The above exception was the direct cause of the followin

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
binascii.Error: decoding with 'hex' codec failed (Error:

>>> codecs.encode("hello", "bz2")
Traceback (most recent call last):
  File "/usr/lib/python3.4/encodings/bz2_codec.py", line
    return (bz2.compress(input), len(input))
  File "/usr/lib/python3.4/bz2.py", line 498, in compres
```

```
    return comp.compress(data) + comp.flush()
TypeError: 'str' does not support the buffer interface

The above exception was the direct cause of the followin

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: encoding with 'bz2' codec failed (TypeError:
```

Por último, como muestran los ejemplos anteriores, estas mejoras han permitido restaurar los alias de conveniencia para los códecs no Unicode que fueron restaurados a su vez en Python 3.2. Esto significa que la codificación de datos binarios a y desde su representación hexadecimal (por ejemplo) puede escribirse ahora como:

```
>>> from codecs import encode, decode
>>> encode(b"hello", "hex")
b'68656c6c6f'
>>> decode(b"68656c6c6f", "hex")
b'hello'
```

Las transformaciones binarias y de texto proporcionadas en la biblioteca estándar se detallan en Transformaciones Binarias y Transformaciones de texto.

(Contribución de Nick Coghlan en bpo-7475 [https://bugs.python.org/issue?@action=redirect&bpo=7475], bpo-17827 [https://bugs.python.org/issue?@action=redirect&bpo=17827], bpo-17828 [https://bugs.python.org/issue?@action=redirect&bpo=17828] y bpo-19619 [https://bugs.python.org/issue?@action=redirect&bpo=19619])

PEP 451: Un tipo ModuleSpec para el sistema de importación

PEP 451 [https://peps.python.org/pep-0451/] proporciona una encapsulación de la información sobre un módulo que la maquinaria de importación utilizará para cargarlo (es decir, una especificación del módulo). Esto ayuda a simplificar tanto la implementación de la importación como varias APIs relacionadas con la importación. El cambio es también un peldaño para varias mejoras futuras relacionadas con la importación [https://mail.python.org/pipermail/python-dev/2013-November/130111.html].

Los cambios públicos del PEP son totalmente compatibles con las versiones anteriores. Además, deberían ser transparentes para todos, excepto para los autores de los importadores. Los métodos del buscador y del cargador de claves han quedado obsoletos, pero seguirán funcionando. Los nuevos importadores deben utilizar los nuevos métodos descritos en el PEP. Los importadores existentes deben ser actualizados para implementar los nuevos métodos. Consulte la sección Obsoleto para obtener una lista de los métodos que deben ser reemplazados y sus sustituciones.

Otros cambios de lenguaje

Algunos de los cambios mas pequeños hechos al núcleo del lenguaje de Python son:

- Base de datos Unicode actualizada a la versión 6.3 de UCD.
- **`min()`** y **`max()`** aceptan ahora un argumento *por defecto* que puede utilizarse para especificar el valor que devuelven si el iterable que están evaluando no tiene elementos. (Contribuido por Julian Berman en bpo-18111 [https://bugs.python.org/issue?@action=redirect&bpo=18111].)
- Los objetos del módulo ahora son weakly referenceable.
- Los atributos del módulo `file__` (y los valores relacionados) deberían ahora contener siempre rutas absolutas por defecto, con la única excepción de `main__.__file__` cuando un script ha sido ejecutado directamente usando una ruta relativa. (Contribuido por Brett Cannon en bpo-18416 [https://bugs.python.org/issue?@action=redirect&bpo=18416].)
- Todos los códecs UTF-* (excepto UTF-7) rechazan ahora los sustitutos tanto durante la codificación como la decodificación, a menos que se utilice el controlador de errores `surrogatepass`, con la excepción del decodificador UTF-16 (que acepta pares de sustitutos válidos) y el codificador UTF-16 (que los produce al codificar caracteres no BMP). (Contribución de Victor Stinner, Kang-Hao (Kenny)

Lu y Serhiy Storchaka en bpo-12892 [https://bugs.python.org/issue?@action=redirect&bpo=12892])

- Nuevo EBCDIC alemán codec `cp273`. (Contribución de Michael Bierenfeld y Andrew Kuchling en bpo-1097797 [https://bugs.python.org/issue?@action=redirect&bpo=1097797])
- Nuevo Ucraniano codec `cp1125` (Contribuido por Serhiy Storchaka en bpo-19668 [https://bugs.python.org/issue?@action=redirect&bpo=19668].)
- **`bytes`**.join() y **`bytearray`**.join() aceptan ahora objetos buffer arbitrarios como argumentos. (Contribuido por Antoine Pitrou en bpo-15958 [https://bugs.python.org/issue?@action=redirect&bpo=15958].)
- El constructor de **`int`** ahora acepta cualquier objeto que tenga un método `index__` para su argumento *base*. (Contribuido por Mark Dickinson en bpo-16772 [https://bugs.python.org/issue?@action=redirect&bpo=16772].)
- Los objetos Frame tienen ahora un método **`clear()`** que borra todas las referencias a las variables locales del frame. (Contribuido por Antoine Pitrou en bpo-17934 [https://bugs.python.org/issue?@action=redirect&bpo=17934].)
- **`memoryview`** está ahora registrada como **`Sequence`**, y soporta el builtin **`reversed()`**. (Contribución de Nick Coghlan y Claudiu Popa en bpo-18690 [https://bugs.python.org/issue?@action=redirect&bpo=18690] y bpo-19078 [https://bugs.python.org/issue?@action=redirect&bpo=19078])
- Las firmas reportadas por **`help()`** han sido modificadas y mejoradas en varios casos como resultado de la introducción de Argument Clinic y otros cambios en los módulos **`inspect`** y **`pydoc`**.
- **`__length_hint__()`** es ahora parte de la especificación formal del lenguaje (ver **PEP 424** [https://peps.python.org/pep-0424/]). (Contribuido por Armin Ronacher en bpo-16148 [https://bugs.python.org/issue?@action=redirect&bpo=16148].)

Nuevos Módulos

asyncio

El nuevo módulo **`asyncio`** (definido en **PEP 3156** [https://

peps.python.org/pep-3156/]) proporciona un modelo de bucle de eventos enchufable estándar para Python, proporcionando un sólido soporte de IO asíncrono en la biblioteca estándar, y facilitando que otras implementaciones de bucle de eventos interoperen con la biblioteca estándar y entre sí.

Para Python 3.4, este módulo se considera un provisional API.

Ver también

PEP 3156 [https://peps.python.org/pep-3156/] – Soporte de IO asíncrono reiniciado: el módulo «asyncio

PEP redactado y ejecutado por Guido van Rossum.

ensurepip

El nuevo módulo **`ensurepip`** es la infraestructura principal para la implementación de **PEP 453** [https://peps.python.org/pep-0453/]. En el curso normal de los acontecimientos, los usuarios finales no necesitarán interactuar con este módulo, pero puede utilizarse para arrancar manualmente `pip` si se rechaza el arranque automático en una instalación o entorno virtual.

`ensurepip` incluye una copia de `pip`, actualizada a partir de la primera versión candidata de la versión de CPython con la que se envía (esto se aplica tanto a las versiones de mantenimiento como a las versiones de características). `ensurepip` no tiene acceso a Internet. Si la instalación tiene acceso a Internet, después de ejecutar `ensurepip` se puede utilizar el paquete `pip` para actualizar `pip` a una versión más reciente que la incluida. (Ten en cuenta que esa versión actualizada de `pip` se considera un paquete instalado por separado y no se eliminará si se desinstala Python)

El módulo se llama *ensure* pip porque si se llama cuando `pip` ya está instalado, no hace nada. También tiene una opción `--upgrade` que hará que se instale la copia incluida de `pip` si la versión instalada de `pip` es más antigua que la copia incluida.

enum

El nuevo módulo `enum` (definido en PEP 435 [https://peps.python.org/pep-0435/]) proporciona una implementación estándar de los tipos de enumeración, lo que permite a otros módulos (como `socket`) proporcionar mensajes de error más informativos y un mejor soporte de depuración al sustituir las constantes enteras opacas por valores de enumeración compatibles con versiones anteriores.

Ver también

PEP 435 [https://peps.python.org/pep-0435/] – Añadir un tipo Enum a la biblioteca estándar de Python

PEP escrito por Barry Warsaw, Eli Bendersky y Ethan Furman, implementado por Ethan Furman.

pathlib

El nuevo módulo `pathlib` ofrece clases que representan las rutas del sistema de ficheros con una semántica apropiada para diferentes sistemas operativos. Las clases de rutas se dividen entre *rutas puras*, que proporcionan operaciones puramente computacionales sin E/S, y *rutas concretas*, que heredan de las rutas puras pero también proporcionan operaciones de E/S.

Para Python 3.4, este módulo se considera un provisional API.

Ver también

PEP 428 [https://peps.python.org/pep-0428/] – El módulo pathlib – rutas del sistema de archivos orientadas a objetos

PEP escrito y aplicado por Antoine Pitrou.

selectores

El nuevo módulo `selectors` (creado como parte de la implementación de PEP 3156 [https://peps.python.org/pep-3156/]) permite una multiplexación de E/S eficiente y de alto nivel, construida sobre las primitivas del módulo `select`.

estadísticas

El nuevo módulo `statistics` (definido en PEP 450 [https://peps.python.org/pep-0450/]) ofrece algunas funciones estadísticas básicas directamente en la biblioteca estándar. Este módulo permite calcular la media, la mediana, la moda, la varianza y la desviación estándar de una serie de datos.

Ver también

PEP 450 [https://peps.python.org/pep-0450/] – Añadir un módulo de estadísticas a la biblioteca estándar
PEP escrito y aplicado por Steven D’Aprano

tracemalloc

El nuevo módulo `tracemalloc` (definido en PEP 454 [https://peps.python.org/pep-0454/]) es una herramienta de depuración para rastrear los bloques de memoria asignados por Python. Proporciona la siguiente información:

- Rastrear dónde se asignó un objeto
- Estadísticas sobre los bloques de memoria asignados por nombre de archivo y por número de línea: tamaño total, número y tamaño medio de los bloques de memoria asignados
- Calcular las diferencias entre dos instantáneas para detectar fugas de memoria

Ver también

PEP 454 [https://peps.python.org/pep-0454/] – Añade un nuevo módulo tracemalloc para rastrear las asignaciones de memoria de Python
PEP escrito y aplicado por Victor Stinner

Módulos mejorados

abc

La nueva función `abc.get_cache_token()` puede utilizarse para saber cuándo invalidar las cachés que se ven afectadas por cambios en el gráfico de objetos. (Contribuido por Łukasz Langa en bpo-16832 [https://bugs.python.org/issue?@action=redirect&bpo=16832].)

La nueva clase `ABC` tiene `ABCMeta` como metaclase. Usar `ABC` como clase base tiene esencialmente el mismo efecto que especificar `metaclass=abc.ABCMeta`, pero es más sencillo de escribir y más fácil de leer. (Contribuido por Bruno Dupuis en bpo-16049 [https://bugs.python.org/issue?@action=redirect&bpo=16049].)

aifc

El método `getparams()` ahora retorna una tupla con nombre en lugar de una tupla simple. (Contribuido por Claudiu Popa en bpo-17487 [https://bugs.python.org/issue?@action=redirect&bpo=17487].)

`aifc.open()` ahora soporta el protocolo de gestión de contexto: cuando se utiliza en un bloque `with`, el método `close()` del objeto devuelto será llamado automáticamente al final del bloque. (Contribuido por Serhiy Storchacha en bpo-16486 [https://bugs.python.org/issue?@action=redirect&bpo=16486].)

Los métodos `writeframesraw()` y `writeframes()` aceptan ahora cualquier bytes-like object. (Contribuido por Serhiy Storchaka en bpo-8311 [https://bugs.python.org/issue?@action=redirect&bpo=8311].)

argparse

La clase `FileType` acepta ahora los argumentos *encoding* y *errors*, que se pasan a `open()`. (Contribuido por Lucas Maystre en bpo-11175 [https://bugs.python.org/issue?@action=redirect&bpo=11175].)

audioop

`audioop` ahora soporta muestras de 24 bits. (Contribuido por Serhiy Storchaka en bpo-12866 [https://bugs.python.org/issue?@action=redirect&bpo=12866].)

La nueva función **`byteswap()`** convierte muestras big-endian en little-endian y viceversa. (Contribuido por Serhiy Storchaka en bpo-19641 [https://bugs.python.org/issue?@action=redirect&bpo=19641].)

Todas las funciones **`audioop`** aceptan ahora cualquier bytes-like object. No se aceptan cadenas: antes no funcionaban, ahora dan un error de inmediato. (Contribuido por Serhiy Storchaka en bpo-16685 [https://bugs.python.org/issue?@action=redirect&bpo=16685].)

base64

Las funciones de codificación y decodificación de **`base64`** aceptan ahora cualquier bytes-like object en los casos en que antes se requería una instancia de **`bytes`** o **`bytearray`**. (Contribución de Nick Coghlan en bpo-17839 [https://bugs.python.org/issue?@action=redirect&bpo=17839].)

Las nuevas funciones **`a85encode()`**, **`a85decode()`**, **`b85encode()`**, y **`b85decode()`** proporcionan la capacidad de codificar y decodificar datos binarios desde y hacia los formatos `Ascii85` y git/mercurial `Base85`, respectivamente. Las funciones `a85` tienen opciones que se pueden utilizar para hacerlas compatibles con las variantes de la codificación `Ascii85`, incluyendo la variante de Adobe. (Contribuido por Martin Morrison, el proyecto Mercurial, Serhiy Storchaka y Antoine Pitrou en bpo-17618 [https://bugs.python.org/issue?@action=redirect&bpo=17618])

colecciones

El método **`ChainMap.new_child()`** acepta ahora un argumento *m* que especifica el mapa hijo a añadir a la cadena. Esto permite utilizar un mapa existente y/o un tipo de mapa personalizado para el hijo. (Contribuido por Vinay Sajip en bpo-16613 [https://bugs.python.org/issue?@action=redirect&bpo=16613].)

colorsys

El número de dígitos en los coeficientes para las conversiones RGB — YIQ se ha ampliado para que coincidan con las versiones FCC NTSC. El cambio en los resultados debería ser inferior al 1% a y puede coincidir mejor con los resultados encontrados en otros lugares. (Contribución de Brian Landers y Serhiy Storchaka en bpo-14323 [https://bugs.python.org/issue?@action=redirect&bpo=14323].)

contextlib

El nuevo gestor de contexto `contextlib.suppress` ayuda a clarificar la intención del código que suprime deliberadamente las excepciones de una única sentencia. (Contribuido por Raymond Hettinger en bpo-15806 [https://bugs.python.org/issue?@action=redirect&bpo=15806] y Zero Piraeus en bpo-19266 [https://bugs.python.org/issue?@action=redirect&bpo=19266])

El nuevo gestor de contexto `contextlib.redirect_stdout()` facilita a los scripts de utilidades el manejo de APIs inflexibles que escriben su salida en `sys.stdout` y no proporcionan ninguna opción para redirigirla. Usando el gestor de contexto, la salida de **`sys.stdout`** puede ser redirigida a cualquier otro flujo o, junto con `io.StringIO`, a una cadena. Esto último puede ser especialmente útil, por ejemplo, para capturar la salida de una función escrita para implementar una interfaz de línea de comandos. Se recomienda sólo para scripts de utilidad porque afecta al estado global de **`sys.stdout`**. (Contribuido por Raymond Hettinger en bpo-15805 [https://bugs.python.org/issue?@action=redirect&bpo=15805].)

También se ha actualizado la documentación de `contextlib` para incluir una discusión de las diferencias entre gestores de contexto de un solo uso, reutilizables y reentrantes.

dbm

Los objetos `dbm.open()` soportan ahora el protocolo de gestión de contexto. Cuando se utiliza en una sentencia `with`, el método `close` del objeto de base de datos será llamado automáticamente al final del bloque. (Contribución de Claudiu Popa y Nick Coghlan en bpo-19282 [https://bugs.python.org/issue?@action=redirect&bpo=19282])

dis

Las funciones `show_code()`, `dis()`, `distb()`, y `disassemble()` aceptan ahora un argumento de *archivo* que controla dónde escriben su salida.

El módulo `dis` se construye ahora en torno a una clase `Instruction` que proporciona acceso orientado a objetos a los detalles de cada operación individual del código de bytes.

Un nuevo método, `get_instructions()`, proporciona un iterador que emite el flujo de instrucciones para un trozo de código Python dado. Por lo tanto, ahora es posible escribir un programa que inspeccione y manipule un objeto de código de bytes de forma diferente a la proporcionada por el propio módulo `dis`. Por ejemplo:

```
>>> import dis
>>> for instr in dis.get_instructions(lambda x: x + 1):
...     print(instr.opname)
LOAD_FAST
LOAD_CONST
BINARY_ADD
RETURN_VALUE
```

Las distintas herramientas de visualización del módulo `dis` se han reescrito para utilizar estos nuevos componentes.

In addition, a new application-friendly class `Bytecode` provides an object-oriented API for inspecting bytecode in both in human-readable form and for iterating over instructions. The **`Bytecode`** constructor takes the same arguments that `get_instructions()` does (plus an optional *current_offset*), and the resulting object can be iterated to produce `Instruction` objects. But it also has a `dis` method, equivalent to calling `dis` on the constructor argument, but returned as a multi-line string:

```
>>> bytecode = dis.Bytecode(lambda x: x + 1, current_off
>>> for instr in bytecode:
...     print('{} ({})'.format(instr.opname, instr.opco
LOAD_FAST (124)
LOAD_CONST (100)
```

```
BINARY_ADD (23)
RETURN_VALUE (83)
>>> bytecode.dis().splitlines()
['  1           0 LOAD_FAST                0 (x)',
 '      -->     3 LOAD_CONST               1 (1)',
 '              6 BINARY_ADD',
 '              7 RETURN_VALUE']
```

`Bytecode` también tiene un método de clase, `from_traceback()`, que proporciona la capacidad de manipular un traceback (es decir, `print(Bytecode.from_traceback(tb).dis())` es equivalente a `distb(tb)`).

(Contribución de Nick Coghlan, Ryan Kelly y Thomas Kluyver en bpo-11816 [https://bugs.python.org/issue?@action=redirect&bpo=11816] y Claudiu Popa en bpo-17916 [https://bugs.python.org/issue?@action=redirect&bpo=17916])

La nueva función `stack_effect()` calcula el efecto en la pila de Python de un opcode y un argumento dados, información que no está disponible de otro modo. (Contribuido por Larry Hastings en bpo-19722 [https://bugs.python.org/issue?@action=redirect&bpo=19722].)

doctest

Un nuevo option flag, `FAIL_FAST`, detiene la ejecución de la prueba tan pronto como se detecta la primera falla. (Contribuido por R. David Murray y Daniel Urban en bpo-16522 [https://bugs.python.org/issue?@action=redirect&bpo=16522]).

La interfaz de línea de comandos de `doctest` utiliza ahora `argparse`, y tiene dos nuevas opciones, `-` y `-f`. `-o` permite <doctest-options> especificar las opciones de opciones doctest en la línea de órdenes, y `-f` es una abreviatura de `- FAIL_FAST` (en paralelo a la opción similar soportada por el CLI de `unittest`). (Contribuido por R. David Murray en bpo-11390 [https://bugs.python.org/issue?@action=redirect&bpo=11390].)

`doctest` ahora encontrará doctests en las cadenas del módulo de extensión `doc___`. (Contribuido por Zachary Ware en bpo-3158 [https://bugs.python.org/issue?@action=redirect&bpo=3158].)

email

`as_string()` acepta ahora un argumento *policy* para anular la política por defecto del mensaje al generar una representación de cadena del mismo. Esto significa que `as_string` puede utilizarse en más circunstancias, en lugar de tener que crear y utilizar un **`generator`** para pasar parámetros de formato a su método `flatten`. (Contribuido por R. David Murray en bpo-18600 [https://bugs.python.org/issue?@action=redirect&bpo=18600].)

Nuevo método **`as_bytes()`** añadido para producir una representación en bytes del mensaje de forma similar a como `as_string` produce una representación en cadena. No acepta el argumento *maxheaderlen*, pero sí los argumentos *unixfrom* y *policy*. El método **`Message`** **`__bytes__()`** lo llama, lo que significa que `bytes(mymsg)` producirá ahora el resultado intuitivo: un objeto bytes que contiene el mensaje completamente formateado. (Contribuido por R. David Murray en bpo-18600 [https://bugs.python.org/issue?@action=redirect&bpo=18600].)

El mensaje **`Message.set_param()`** acepta ahora un argumento de palabra clave *replace*. Cuando se especifica, la cabecera asociada se actualizará sin cambiar su ubicación en la lista de cabeceras. Por compatibilidad, el valor por defecto es `False`. (Contribuido por R. David Murray en bpo-18891 [https://bugs.python.org/issue?@action=redirect&bpo=18891].)

Se han añadido un par de nuevas subclases de **`Message`** (**`EmailMessage`** y **`MIMEPart`**), junto con un nuevo submódulo, **`contentmanager`** y un nuevo atributo **`policy`** **`content_manager`**. Toda la documentación se encuentra actualmente en el nuevo módulo, que se añade como parte del nuevo provisional API de email. Estas clases proporcionan una serie de nuevos métodos que facilitan la extracción e inserción de contenidos en los mensajes de correo electrónico. Para más detalles, consulte la documentación de **`contentmanager`** y la email: Ejemplos. Estas adiciones a la API completan la mayor parte del trabajo que estaba previsto como parte del proyecto email6. Está previsto que la API, actualmente provisional, se convierta en la definitiva en Python 3.5 (posiblemente con algunos añadidos menores en el área de gestión de errores). (Contribuido por R. David Murray en bpo-18891 [https://bugs.python.org/issue?@action=redirect&bpo=18891].)

archivoecmp

Una nueva función **`clear_cache()`** ofrece la posibilidad de borrar la caché de comparación **`filecmp`**, que utiliza la información de **`os.stat()`** para determinar si el fichero ha cambiado desde la última comparación. Esto puede usarse, por ejemplo, si el archivo puede haber sido modificado y vuelto a comprobar en menos tiempo que la resolución del campo de tiempo de modificación de un archivo del sistema de archivos en particular. (Contribuido por Mark Levitt en bpo-18149 [https://bugs.python.org/issue?@action=redirect&bpo=18149].)

El nuevo atributo de módulo **`DEFAULT_IGNORES`** proporciona la lista de directorios que se utilizan como valor predeterminado para el parámetro *ignore* de la función **`dircmp()`**. (Contribuido por Eli Bendersky en bpo-15442 [https://bugs.python.org/issue?@action=redirect&bpo=15442]).

functools

El nuevo descriptor **`partialmethod()`** aporta la aplicación de argumentos parciales a los descriptores, al igual que **`partial()`** proporciona para los callables normales. El nuevo descriptor también facilita la obtención de llamadas arbitrarias (incluyendo instancias de **`partial()`**) para que se comporten como métodos de instancia normales cuando se incluyen en la definición de una clase. (Contribución de Alon Horev y Nick Coghlan en bpo-4331 [https://bugs.python.org/issue?@action=redirect&bpo=4331])

El nuevo decorador **`singledispatch()`** aporta soporte para funciones genéricas single-dispatch a la biblioteca estándar de Python. Mientras que la programación orientada a objetos se centra en agrupar múltiples operaciones sobre un conjunto común de datos en una clase, una función genérica se centra en agrupar múltiples implementaciones de una operación que le permite trabajar con *diferentes* tipos de datos.

Ver también

PEP 443 [https://peps.python.org/pep-0443/] – Funciones genéricas de despacho único
PEP escrito e implementado por Łukasz Langa.

`total_ordering()` ahora admite un valor de retorno de `NotImplemented` de la función de comparación subyacente. (Contribución de Katie Miller en bpo-10042 [https://bugs.python.org/issue?@action=redirect&bpo=10042]).

Una versión en python puro de la función `partial()` está ahora en la stdlib; en CPython está anulada por la versión acelerada en C, pero está disponible para que otras implementaciones la usen. (Contribuido por Brian Thorne en bpo-12428 [https://bugs.python.org/issue?@action=redirect&bpo=12428].)

gc

La nueva función `get_stats()` devuelve una lista de tres diccionarios por generación que contienen las estadísticas de las colecciones desde el inicio del intérprete. (Contribuido por Antoine Pitrou en bpo-16351 [https://bugs.python.org/issue?@action=redirect&bpo=16351].)

glob

Una nueva función `escape()` proporciona una forma de escapar los caracteres especiales de un nombre de archivo para que no formen parte de la expansión globbing sino que se comparen literalmente. (Contribuido por Serhiy Storchaka en bpo-8402 [https://bugs.python.org/issue?@action=redirect&bpo=8402].)

hashlib

Una nueva función `hashlib.pbkdf2_hmac()` proporciona la función de derivación de clave basada en contraseña PKCS#5 2 [https://en.wikipedia.org/wiki/PBKDF2]. (Contribuido por Christian Heimes en bpo-18582 [https://bugs.python.org/issue?@action=redirect&bpo=18582].)

El atributo `name` de los objetos hash `hashlib` es ahora una interfaz formalmente soportada. Siempre ha existido en el `hashlib` de CPython (aunque no devolvía nombres en minúsculas para todos los hashes soportados), pero no era una interfaz pública y por eso algunas otras implementaciones de Python no la han soportado previamente. (Contribuido por Jason R. Coombs en bpo-18532 [https://bugs.python.org/issue?@action=redirect&bpo=18532].)

hmac

`hmac` acepta ahora `bytearray` así como `bytes` para el argumento *clave* de la función `new()`, y el parámetro *msg* tanto de la función `new()` como del método `update()` acepta ahora cualquier tipo soportado por el módulo `hashlib`. (Contribuido por Jonas Borgström en bpo-18240 [https://bugs.python.org/issue?@action=redirect&bpo=18240].)

El argumento *digestmod* de la función `hmac.new()` puede ser ahora cualquier nombre de digesto hash reconocido por `hashlib`. Además, el comportamiento actual en el que el valor de *digestmod* por defecto es `MD5` queda obsoleto: en una futura versión de Python no habrá valor por defecto. (Contribuido por Christian Heimes en bpo-17276 [https://bugs.python.org/issue?@action=redirect&bpo=17276].)

Con la adición de los atributos `block_size` y `name` (y la documentación formal del atributo `digest_size`), el módulo `hmac` se ajusta ahora plenamente a la API **PEP 247** [https://peps.python.org/pep-0247/]. (Contribuido por Christian Heimes en bpo-18775 [https://bugs.python.org/issue?@action=redirect&bpo=18775].)

html

La nueva función `unescape()` convierte las referencias de caracteres de HTML5 en los correspondientes caracteres Unicode. (Contribuido por Ezio Melotti en bpo-2927 [https://bugs.python.org/issue?@action=redirect&bpo=2927].)

`HTMLParser` acepta un nuevo argumento de palabra clave *convert_charrefs* que, cuando es `True`, convierte automáticamente todas las referencias de caracteres. Por compatibilidad, su valor por defecto es `False`, pero cambiará a `True` en una futura versión de Python, así que te invitamos a establecerlo explícitamente y actualizar tu código para usar esta nueva característica. (Contribuido por Ezio Melotti en bpo-13633 [https://bugs.python.org/issue?@action=redirect&bpo=13633].)

El argumento *strict* de **`HTMLParser`** está ahora obsoleto. (Contribuido por Ezio Melotti en bpo-15114 [https://bugs.python.org/issue?@action=redirect&bpo=15114].)

http

`send_error()` now accepts an optional additional *explain* parameter which can be used to provide an extended error description, overriding the hardcoded default if there is one. This extended error description will be formatted using the **`error_message_format`** attribute and sent as the body of the error response. (Contributed by Karl Cow in bpo-12921 [https://bugs.python.org/issue?@action=redirect&bpo=12921].)

El **`http.server`** interfaz de línea de comandos tiene ahora una opción `-b/--bind` que hace que el servidor escuche en una dirección específica. (Contribuido por Malte Swart en bpo-17764 [https://bugs.python.org/issue?@action=redirect&bpo=17764].)

idlelib y IDLE

Como idlelib implementa el shell y el editor de IDLE y no está pensado para ser importado por otros programas, recibe mejoras con cada versión. Vea `Lib/idlelib/NEWS.txt` para una lista acumulativa de cambios desde la versión 3.3.0, así como los cambios realizados en futuras versiones 3.4.x. Este fichero también está disponible en el diálogo de IDLE *Ayuda* ‣ *Sobre IDLE*.

importlib

El ABC de `InspectLoader` define un nuevo método, `source_to_code()` que acepta datos de origen y una ruta y devuelve un objeto de código. La implementación por defecto es equivalente a `compile(data, path, 'exec', dont_inherit=True)`. (Contribuido por Eric Snow y Brett Cannon en bpo-15627 [https://bugs.python.org/issue?@action=redirect&bpo=15627].)

`InspectLoader` también tiene ahora una implementación por defecto para el método `get_code()`. Sin embargo, normalmente será deseable anular la implementación por defecto por razones de rendimiento. (Contribuido por Brett Cannon en bpo-18072 [https://bugs.python.org/issue?@action=redirect&bpo=18072].)

La función `reload()` se ha trasladado de `imp` a `importlib` como parte de la descontinuación del módulo `imp`. (Contribución de Berker Peksag en bpo-18193 [https://bugs.python.org/issue?@action=redirect&bpo=18193]).

`importlib.util` now has a `MAGIC_NUMBER` attribute providing access to the bytecode version number. This replaces the `get_magic()` function in the deprecated `imp` module. (Contributed by Brett Cannon in bpo-18192 [https://bugs.python.org/issue?@action=redirect&bpo=18192].)

Las nuevas funciones `importlib.util` `cache_from_source()` y `source_from_cache()` reemplazan las funciones con el mismo nombre en el módulo `imp` obsoleto. (Contribución de Brett Cannon en bpo-18194 [https://bugs.python.org/issue?@action=redirect&bpo=18194]).

El bootstrap `importlib` `NamespaceLoader` se ajusta ahora al ABC de `InspectLoader`, lo que significa que `runpy` y `python -m` pueden utilizarse ahora con paquetes de espacios de nombres. (Contribuido por Brett Cannon en bpo-18058 [https://bugs.python.org/issue?@action=redirect&bpo=18058].)

`importlib.util` tiene una nueva función `decode_source()` que decodifica la fuente a partir de bytes utilizando el procesamiento universal de nuevas líneas. Esto es útil para implementar los métodos `InspectLoader.get_source()`.

`importlib.machinery.ExtensionFileLoader` tiene ahora un método `get_filename()`. Esto fue omitido inadvertidamente en la implementación original. (Contribuido por Eric Snow en bpo-19152 [https://bugs.python.org/issue?@action=redirect&bpo=19152].)

inspeccionar

El módulo `inspect` ofrece ahora una interfaz de línea de comandos básica para mostrar rápidamente el código fuente y otra información de módulos, clases y funciones. (Contribución de Claudiu Popa y Nick Coghlan en bpo-18626 [https://bugs.python.org/issue?@action=redirect&bpo=18626])

`unwrap()` facilita el desenvolvimiento de las cadenas de funciones envolventes creadas por `functools.wraps()` (y cualquier otra API que establezca el atributo `__wrapped__` en una función envolvente). (Contribución de Daniel Urban, Aaron Iles y Nick Coghlan en bpo-13266 [https://bugs.python.org/issue?@action=redirect&bpo=13266])

Como parte de la implementación del nuevo módulo `enum`, el módulo `inspect` tiene ahora un soporte sustancialmente mejor para los métodos personalizados `__dir__` y los atributos de clase dinámicos proporcionados a través de metaclases. (Contribuido por Ethan Furman en bpo-18929 [https://bugs.python.org/issue?@action=redirect&bpo=18929] y bpo-19030 [https://bugs.python.org/issue?@action=redirect&bpo=19030])

`getfullargspec()` and `getargspec()` now use the `signature()` API. This allows them to support a much broader range of callables, including those with `__signature__` attributes, those with metadata provided by argument clinic, `functools.partial()` objects and more. Note that, unlike `signature()`, these functions still ignore `__wrapped__` attributes, and report the already bound first argument for bound methods, so it is still necessary to update your code to use `signature()` directly if those features are desired. (Contributed by Yury Selivanov in bpo-17481 [https://bugs.python.org/issue?@action=redirect&bpo=17481].)

`signature()` ahora soporta los tipos de pato de las funciones CPython, lo que añade soporte para las funciones compiladas con Cython. (Contribuido por Stefan Behnel y Yury Selivanov en bpo-17159 [https://bugs.python.org/issue?@action=redirect&bpo=17159])

dirección IP

`ipaddress` se añadió a la biblioteca estándar en Python 3.3 como una provisional API. Con el lanzamiento de Python 3.4, se ha eliminado esta calificación: `ipaddress` se considera ahora una API estable, cubierta por los requisitos normales de la biblioteca estándar para mantener la compatibilidad hacia atrás.

Una nueva propiedad `is_global` es `True` si una dirección es enrutable globalmente. (Contribuido por Peter Moody en bpo-17400 [https://bugs.python.org/issue?@action=redirect&bpo=17400].)

logging

La clase `TimedRotatingFileHandler` tiene un nuevo parámetro *atTime* que puede utilizarse para especificar la hora del día en que debe producirse la renovación. (Contribuido por Ronald Oussoren en bpo-9556 [https://bugs.python.org/issue?@action=redirect&bpo=9556].)

`SocketHandler` y `DatagramHandler` ahora soportan sockets de dominio Unix (estableciendo *puerto* a `None`). (Contribuido por Vinay Sajip en el commit ce46195b56a9.)

`fileConfig()` ahora acepta una instancia de la subclase `configparser.RawConfigParser` para el parámetro *fname*. Esto facilita el uso de un archivo de configuración cuando la configuración de registro es sólo una parte de la configuración general de la aplicación, o cuando la aplicación modifica la configuración antes de pasarla a `fileConfig()`. (Contribuido por Vinay Sajip en bpo-16110 [https://bugs.python.org/issue?@action=redirect&bpo=16110].)

Los datos de configuración de registro recibidos desde un socket a través de la función `logging.config.listen()` pueden ahora ser validados antes de ser procesados suministrando una función de verificación como argumento de la nueva palabra clave *verify*. (Contribuido por Vinay Sajip en bpo-15452 [https://bugs.python.org/issue?@action=redirect&bpo=15452].)

mariscal

La versión por defecto de **`marshal`** ha sido aumentada a 3. El código que implementa la nueva versión restablece el comportamiento de Python2 de registrar sólo una copia de las cadenas internadas y preservar la internación en la deserialización, y extiende esta capacidad de «una copia» a cualquier tipo de objeto (incluyendo el manejo de referencias recursivas). Esto reduce tanto el tamaño de los archivos `.pyc` como la cantidad de memoria que ocupa un módulo en la memoria cuando se carga desde un archivo `.pyc` (o `.pyo`). (Contribuido por Kristján Valur Jónsson en bpo-16475 [https://bugs.python.org/issue?@action=redirect&bpo=16475], con mejoras adicionales de velocidad por Antoine Pitrou en bpo-19219 [https://bugs.python.org/issue?@action=redirect&bpo=19219])

mmap

los objetos mmap ahora son weakly referenceable. (Aportado por Valerie Lambert en bpo-4885 [https://bugs.python.org/issue?@action=redirect&bpo=4885].)

multiprocesamiento

En Unix se han añadido métodos de comienzo `spawn` y `forkserver`, para iniciar procesos usando **`multiprocessing`**. Estos hacen que la mezcla de procesos con hilos sea más robusta, y el método `spawn` coincide con la semántica que el multiprocesamiento siempre ha utilizado en Windows. La nueva función **`get_all_start_methods()`** informa de todos los métodos de inicio disponibles en la plataforma, **`get_start_method()`** informa del método de inicio actual, y **`set_start_method()`** establece el método de inicio. (Contribuido por Richard Oudkerk en bpo-8713 [https://bugs.python.org/issue?@action=redirect&bpo=8713].)

`multiprocessing` también tiene ahora el concepto de `contexto`, que determina cómo se crean los procesos hijos. La nueva función `get_context()` devuelve un contexto que utiliza un método de inicio especificado. Tiene la misma API que el propio módulo `multiprocessing`, por lo que se puede utilizar para crear `pool` y otros objetos que operarán dentro de ese contexto. Esto permite que un framework y una aplicación o diferentes partes de la misma aplicación utilicen el multiprocesamiento sin interferir entre sí. (Contribuido por Richard Oudkerk en bpo-18999 [https://bugs.python.org/issue?@action=redirect&bpo=18999].)

Excepto cuando se utiliza el antiguo método de inicio *fork*, los procesos hijos ya no heredan los handles/descriptores de archivo innecesarios de sus padres (parte de bpo-8713 [https://bugs.python.org/issue?@action=redirect&bpo=8713]).

`multiprocessing` ahora se basa en `runpy` (que implementa el conmutador `-m`) para inicializar `__main__` apropiadamente en los procesos hijos cuando se utilizan los métodos de inicio `spawn` o `forkserver`. Esto resuelve algunos casos extremos en los que la combinación de multiprocesamiento, el conmutador de línea de comandos `-m` y las importaciones relativas explícitas podían causar fallos oscuros en los procesos hijos. (Contribuido por Nick Coghlan en bpo-19946 [https://bugs.python.org/issue?@action=redirect&bpo=19946].)

operator

La nueva función `length_hint()` proporciona una implementación de la especificación de cómo debe utilizarse el método especial `__length_hint__()`, como parte de la especificación formal **PEP 424** [https://peps.python.org/pep-0424/] de esta característica del lenguaje. (Contribuido por Armin Ronacher en bpo-16148 [https://bugs.python.org/issue?@action=redirect&bpo=16148].)

Ahora hay una versión puramente python del módulo `operator` disponible como referencia y para ser utilizada por implementaciones alternativas de Python. (Contribuido por Zachary Ware en bpo-16694 [https://bugs.python.org/issue?@action=redirect&bpo=16694].)

os

Hay nuevas funciones para obtener y establecer la bandera heritable de un descriptor de archivo (**`os.get_inheritable()`**, **`os.set_inheritable()`**) o de un handle de Windows (**`os.get_handle_inheritable()`**, **`os.set_handle_inheritable()`**).

La nueva función **`cpu_count()`** informa del número de CPUs disponibles en la plataforma en la que se está ejecutando Python (o `None` si no se puede determinar el recuento). La función **`multiprocessing.cpu_count()`** está ahora implementada en términos de esta función). (Contribución de Trent Nelson, Yogesh Chaudhari, Victor Stinner y Charles-François Natali en bpo-17914 [https://bugs.python.org/issue?@action=redirect&bpo=17914])

`os.path.samestat()` está ahora disponible en la plataforma Windows (y la implementación de **`os.path.samefile()`** es ahora compartida entre Unix y Windows). (Contribuido por Brian Curtin en bpo-11939 [https://bugs.python.org/issue?@action=redirect&bpo=11939].)

`os.path.ismount()` ahora reconoce los volúmenes montados por debajo de la raíz de una unidad en Windows. (Contribuido por Tim Golden en bpo-9035 [https://bugs.python.org/issue?@action=redirect&bpo=9035].)

`os.open()` admite dos nuevos indicadores en las plataformas que los proporcionan: **`O_PATH`** (descriptor de archivo no abierto) y **`O_TMPFILE`** (archivo temporal sin nombre; a partir de la versión 3.4.0, disponible solo en sistemas Linux con una versión de kernel 3.11 o más reciente que tengan encabezados uapi). (Contribuido por Christian Heimes en bpo-18673 [https://bugs.python.org/issue?@action=redirect&bpo=18673] y Benjamin Peterson, respectivamente).

pdb

`pdb` ha sido mejorado para manejar generadores, **`yield`**, y `yield from` de una manera más útil. Esto es especialmente útil cuando se depuran programas basados en **`asyncio`**. (Contribución de Andrew Svetlov y Xavier de Gaye en bpo-16596 [https://bugs.python.org/issue?@action=redirect&bpo=16596])

El comando `print` ha sido eliminado de **`pdb`**, restaurando el acceso a la función de Python **`print()`** desde la línea de comandos de la pdb. El comando `pdb` de Python2 no tenía un comando `print`; en su lugar, al introducir `print` se ejecutaba la sentencia `print`. En Python3 `print` se convirtió por error en un alias del comando pdb **`p`**. Sin embargo, `p` imprime la `repr` de su argumento, no la `str` como hacía el comando `print` de Python2. Peor aún, el comando `pdb print` de Python3 ensombrece la función `print` de Python3, haciéndola inaccesible en el prompt de `pdb`. (Contribuido por Connor Osborn en bpo-18764 [https://bugs.python.org/issue?@action=redirect&bpo=18764].)

pepinillo

`pickle` now supports (but does not use by default) a new pickle protocol, protocol 4. This new protocol addresses a number of issues that were present in previous protocols, such as the serialization of nested classes, very large strings and containers, and classes whose **`__new__()`** method takes keyword-only arguments. It also provides some efficiency improvements.

Ver también

PEP 3154 [https://peps.python.org/pep-3154/] – Protocolo Pickle 4
PEP escrito por Antoine Pitrou e implementado por Alexandre Vassalotti.

plistlib

`plistlib` ahora tiene una API similar al patrón estándar para los protocolos de serialización de stdlib, con las nuevas funciones **`load()`**, **`dump()`**, **`loads()`** y **`dumps()`**. (La API anterior ya está obsoleta). Además del formato plist XML ya compatible (**`FMT_XML`**), ahora también admite el formato plist binario (**`FMT_BINARY`**). (Contribuido por Ronald Oussoren y otros en bpo-14455 [https://bugs.python.org/issue?@action=redirect&bpo=14455]).

poplib

Se han añadido dos nuevos métodos a **`poplib`**: **`capa()`**, que devuelve la lista de capacidades anunciadas por el servidor POP, y **`stls()`**, que cambia una sesión POP3 de texto claro a una sesión POP3 cifrada si el servidor POP lo soporta. (Contribuido por Lorenzo Catucci en bpo-4473 [https://bugs.python.org/issue?@action=redirect&bpo=4473].)

pprint

La clase **`PrettyPrinter`** del módulo **`pprint`** y sus funciones **`pformat()`**, y **`pprint()`** tienen una nueva opción, *compact*, que controla cómo se formatea la salida. Actualmente, establecer *compact* a `True` significa que las secuencias se imprimirán con tantos elementos de secuencia como quepan en el *ancho* de cada línea (indentada). (Contribuido por Serhiy Storchaka en bpo-19132 [https://bugs.python.org/issue?@action=redirect&bpo=19132].)

Las cadenas largas ahora se envuelven usando la sintaxis normal de continuación de línea de Python. (Contribuido por Antoine Pitrou en bpo-17150 [https://bugs.python.org/issue?@action=redirect&bpo=17150].)

pty

`pty.spawn()` ahora devuelve el valor de estado de **`os.waitpid()`** en el proceso hijo, en lugar de `None`. (Contribución de Gregory P. Smith)

pydoc

El módulo **`pydoc`** se basa ahora directamente en la API de introspección **`inspect.signature()`**, lo que le permite proporcionar información de firma para una mayor variedad de objetos invocables. Este cambio también significa que los atributos `__wrapped__` se tienen ahora en cuenta al mostrar la información de ayuda. (Contribución de Larry Hastings en bpo-19674 [https://bugs.python.org/issue?@action=redirect&bpo=19674].)

El módulo **`pydoc`** ya no muestra el parámetro `self` para los métodos ya vinculados. En su lugar, pretende mostrar siempre la firma actual exacta de la llamada suministrada. (Contribuido por

Larry Hastings en bpo-20710 [https://bugs.python.org/issue?@action=redirect&bpo=20710].)

Además de los cambios que se han realizado en **`pydoc`** directamente, su manejo de los métodos personalizados `__dir__` y varios comportamientos del descriptor también se han mejorado sustancialmente por los cambios subyacentes en el módulo **`inspect`**.

Como el builtin **`help()`** está basado en **`pydoc`**, los cambios anteriores también afectan al comportamiento de **`help()`**.

re

New **`fullmatch()`** function and **`Pattern.fullmatch()`** method anchor the pattern at both ends of the string to match. This provides a way to be explicit about the goal of the match, which avoids a class of subtle bugs where `$` characters get lost during code changes or the addition of alternatives to an existing regular expression. (Contributed by Matthew Barnett in bpo-16203 [https://bugs.python.org/issue?@action=redirect&bpo=16203].)

La repr de los regex objects ahora incluye el patrón y las banderas; la repr de los objetos regex ahora incluye el inicio, el final y la parte de la cadena que coincide. (Contribución de Hugo Lopes Tavares y Serhiy Storchaka en bpo-13592 [https://bugs.python.org/issue?@action=redirect&bpo=13592] y bpo-17087 [https://bugs.python.org/issue?@action=redirect&bpo=17087])

recurso

La nueva función **`prlimit()`**, disponible en plataformas Linux con un kernel de versión 2.6.36 o posterior y glibc de 2.13 o posterior, proporciona la capacidad de consultar o establecer los límites de recursos para procesos distintos del que realiza la llamada. (Contribuido por Christian Heimes en bpo-16595 [https://bugs.python.org/issue?@action=redirect&bpo=16595].)

En la versión 2.6.36 o posterior del kernel de Linux, también hay algunas nuevas constantes específicas de Linux:

`RLIMIT_MSGQUEUE`, `RLIMIT_NICE`, `RLIMIT_RTPRIO`, `RLIMIT_RTTIME` y `RLIMIT_SIGPENDING`. (Contribuido por Christian Heimes en bpo-19324 [https://bugs.python.org/issue?@action=redirect&bpo=19324]).

En la versión 9 y posteriores de FreeBSD, hay algunas nuevas constantes específicas de FreeBSD: `RLIMIT_SBSIZE`, `RLIMIT_SWAP` y `RLIMIT_NPTS`. (Contribución de Claudiu Popa en bpo-19343 [https://bugs.python.org/issue?@action=redirect&bpo=19343]).

seleccionar

Los objetos `epoll` ahora soportan el protocolo de gestión de contexto. Cuando se utiliza en una declaración `with`, el método `close()` será llamado automáticamente al final del bloque. (Contribuido por Serhiy Storchaka en bpo-16488 [https://bugs.python.org/issue?@action=redirect&bpo=16488].)

Los objetos `devpoll` tienen ahora métodos `fileno()` y `close()`, así como un nuevo atributo `closed`. (Contribuido por Victor Stinner en bpo-18794 [https://bugs.python.org/issue?@action=redirect&bpo=18794].)

estante

Las instancias de `Shelf` ahora pueden utilizarse en declaraciones `with`, y se cerrarán automáticamente al final del bloque `with`. (Contribuido por Filip Gruszczyński en bpo-13896 [https://bugs.python.org/issue?@action=redirect&bpo=13896].)

shutil

`copyfile()` ahora lanza una subclase específica de `Error`, `SameFileError`, cuando el origen y el destino son el mismo fichero, lo que permite a una aplicación tomar la acción apropiada en este error específico. (Contribución de Atsuo Ishimoto y Hynek Schlawack en bpo-1492704 [https://bugs.python.org/issue?@action=redirect&bpo=1492704])

smtpd

Las clases **`SMTPServer`** y **`SMTPChannel`** ahora aceptan un argumento de palabra clave *map* que, si se especifica, se pasa a **`asynchat.async_chat`** como su argumento *map*. Esto permite que una aplicación evite afectar el mapa de sockets global. (Contribuido por Vinay Sajip en bpo-11959 [https://bugs.python.org/issue?@action=redirect&bpo=11959]).

smtplib

`SMTPException` es ahora una subclase de **`OSError`**, que permite que tanto los errores de nivel de socket como los de nivel de protocolo SMTP sean capturados en una sentencia try/except por un código que sólo se preocupa de si se ha producido o no un error. (Contribuido por Ned Jackson Lovely en bpo-2118 [https://bugs.python.org/issue?@action=redirect&bpo=2118].)

enchufe

El módulo de socket ahora admite el protocolo **`CAN_BCM`** en las plataformas que lo admiten. (Contribución de Brian Thorne en bpo-15359 [https://bugs.python.org/issue?@action=redirect&bpo=15359]).

Los objetos Socket tienen nuevos métodos para obtener o establecer su bandera heredable, **`get_inheritable()`** y **`set_inheritable()`**.

Las constantes `socket.AF_*` y `socket.SOCK_*` son ahora valores de enumeración utilizando el nuevo módulo **`enum`**. Esto permite imprimir nombres significativos durante la depuración, en lugar de «números mágicos» enteros.

La constante **`AF_LINK`** está ahora disponible en BSD y OSX.

`inet_pton()` y **`inet_ntop()`** son ahora compatibles con Windows. (Contribuido por Atsuo Ishimoto en bpo-7171 [https://bugs.python.org/issue?@action=redirect&bpo=7171].)

sqlite3

Un nuevo parámetro booleano para la función **`connect()`**, *uri*, puede usarse para indicar que el parámetro *base de datos* es una `uri` (véase la documentación SQLite URI [https://www.sqlite.org/uri.html]). (Contribuido por poq en bpo-13773 [https://bugs.python.org/issue?@action=redirect&bpo=13773].)

ssl

Se han añadido **`PROTOCOL_TLSv1_1`** y **`PROTOCOL_TLSv1_2`** (soporte de TLSv1.1 y TLSv1.2); el soporte de estos protocolos sólo está disponible si Python está enlazado con OpenSSL 1.0.1 o posterior. (Contribuido por Michele Orrù y Antoine Pitrou en bpo-16692 [https://bugs.python.org/issue?@action=redirect&bpo=16692])

La nueva función **`create_default_context()`** proporciona una forma estándar de obtener un **`SSLContext`** cuya configuración pretende ser un equilibrio razonable entre compatibilidad y seguridad. Estos ajustes son más estrictos que los valores por defecto proporcionados por el constructor **`SSLContext`**, y pueden ser ajustados en el futuro, sin necesidad de una depreciación previa, si los requisitos de seguridad de las mejores prácticas cambian. La nueva práctica recomendada para usar las bibliotecas stdlib que soportan SSL es usar **`create_default_context()`** para obtener un objeto **`SSLContext`**, modificarlo si es necesario, y luego pasarlo como el argumento *context* de la API stdlib apropiada. (Contribuido por Christian Heimes en bpo-19689 [https://bugs.python.org/issue?@action=redirect&bpo=19689].)

El método **`SSLContext load_verify_locations()`** acepta un nuevo argumento opcional *cadata*, que puede utilizarse para proporcionar certificados codificados en PEM o DER directamente mediante cadenas o bytes, respectivamente. (Contribuido por Christian Heimes en bpo-18138 [https://bugs.python.org/issue?@action=redirect&bpo=18138].)

La nueva función **`get_default_verify_paths()`** devuelve una tupla con nombre de las rutas y variables de entorno que el método **`set_default_verify_paths()`** utiliza para establecer el `cafile` y el `capath` por defecto de OpenSSL. Esto puede ser una ayuda para depurar problemas de verificación por defecto. (Contribuido por Christian Heimes en bpo-18143 [https://

bugs.python.org/issue?@action=redirect&bpo=18143].)

`SSLContext` tiene un nuevo método, **`cert_store_stats()`**, que informa del número de certificados `X.509` cargados, certificados `X.509 CA` y listas de revocación de certificados (`crl`), así como un método **`get_ca_certs()`** que devuelve una lista de los certificados `CA` cargados. (Contribuido por Christian Heimes en bpo-18147 [https://bugs.python.org/issue?@action=redirect&bpo=18147].)

Si está disponible OpenSSL 0.9.8 o posterior, **`SSLContext`** tiene un nuevo atributo **`verify_flags`** que se puede utilizar para controlar el proceso de verificación de certificados estableciéndolo en alguna combinación de las nuevas constantes **`VERIFY_DEFAULT`**, **`VERIFY_CRL_CHECK_LEAF`**, **`VERIFY_CRL_CHECK_CHAIN`** o **`VERIFY_X509_STRICT`**. OpenSSL no realiza ninguna verificación de CRL de forma predeterminada. (Contribuido por Christien Heimes en bpo-8813 [https://bugs.python.org/issue?@action=redirect&bpo=8813]).

Nuevo método **`SSLContext load_default_certs()`** carga un conjunto de certificados de «autoridad de certificación» (CA) por defecto desde ubicaciones predeterminadas, que varían según la plataforma. Puede utilizarse para cargar tanto certificados de autenticación de servidores web TLS (`purpose=`**`SERVER_AUTH`**) para que un cliente los utilice para verificar un servidor, como certificados para que un servidor los utilice para verificar certificados de clientes (`purpose=`**`CLIENT_AUTH`**). (Contribuido por Christian Heimes en bpo-19292 [https://bugs.python.org/issue?@action=redirect&bpo=19292].)

Dos nuevas funciones sólo para Windows, **`enum_certificates()`** y **`enum_crls()`** ofrecen la posibilidad de recuperar certificados, información de certificados y CRLs del almacén de certificados de Windows. (Contribuido por Christian Heimes en bpo-17134 [https://bugs.python.org/issue?@action=redirect&bpo=17134].)

Soporte para SNI (Server Name Indication) del lado del servidor utilizando el nuevo método **`ssl.SSLContext.set_servername_callback()`**. (Contribuido por Daniel Black en bpo-8109 [https://bugs.python.org/issue?

@action=redirect&bpo=8109].)

El diccionario devuelto por **`SSLSocket.getpeercert()`** contiene elementos de extensión `X509v3` adicionales: `crlDistributionPoints`, `calIssuers`, y `OCSP` URIs. (Contribución de Christian Heimes en bpo-18379 [https://bugs.python.org/issue?@action=redirect&bpo=18379].)

stat

El módulo **`stat`** ahora cuenta con el respaldo de una implementación en C en **`_stat`**. Se requiere una implementación en C ya que la mayoría de los valores no están estandarizados y dependen de la plataforma. (Contribución de Christian Heimes en bpo-11016 [https://bugs.python.org/issue?@action=redirect&bpo=11016]).

El módulo admite los nuevos indicadores **`ST_MODE`**, **`S_IFDOOR`**, **`S_IFPORT`** y **`S_IFWHT`**. (Contribuido por Christian Hiemes en bpo-11016 [https://bugs.python.org/issue?@action=redirect&bpo=11016]).

struct

La nueva función **`iter_unpack`** y un nuevo método **`struct.Struct.iter_unpack()`** en formatos compilados proporcionan un desempaquetado en flujo de un buffer que contiene instancias repetidas de un formato de datos dado. (Contribuido por Antoine Pitrou en bpo-17804 [https://bugs.python.org/issue?@action=redirect&bpo=17804].)

subproceso

`check_output()` acepta ahora un argumento *input* que puede utilizarse para proporcionar el contenido de `stdin` para el comando que se ejecuta. (Contribuido por Zack Weinberg en bpo-16624 [https://bugs.python.org/issue?@action=redirect&bpo=16624].)

`getoutput()` and **`getstatusoutput()`** now work on Windows. This change was actually inadvertently made in 3.3.4. (Contributed by Tim Golden in bpo-10197 [https://bugs.python.org/issue?@action=redirect&bpo=10197].)

sunau

El método **`getparams()`** ahora devuelve una tupla con nombre en lugar de una tupla simple. (Contribuido por Claudiu Popa en bpo-18901 [https://bugs.python.org/issue?@action=redirect&bpo=18901].)

`sunau.open()` ahora admite el protocolo de gestión de contexto: cuando se utiliza en un bloque **`with`**, el método `close` del objeto devuelto se llamará automáticamente al final del bloque. (Contribuido por Serhiy Storchaka en bpo-18878 [https://bugs.python.org/issue?@action=redirect&bpo=18878]).

`AU_write.setsampwidth()` now supports 24 bit samples, thus adding support for writing 24 sample using the module. (Contributed by Serhiy Storchaka in bpo-19261 [https://bugs.python.org/issue?@action=redirect&bpo=19261].)

Los métodos **`writeframesraw()`** y **`writeframes()`** aceptan ahora cualquier bytes-like object. (Contribuido por Serhiy Storchaka en bpo-8311 [https://bugs.python.org/issue?@action=redirect&bpo=8311].)

sys

La nueva función **`sys.getallocatedblocks()`** devuelve el número actual de bloques asignados por el intérprete. (En CPython con la configuración por defecto `--with-pymalloc`, se trata de asignaciones realizadas a través de la API **`PyObject_Malloc()`**) Esto puede ser útil para rastrear las fugas de memoria, especialmente si se automatiza a través de un conjunto de pruebas. (Contribuido por Antoine Pitrou en bpo-13390 [https://bugs.python.org/issue?@action=redirect&bpo=13390].)

Cuando el intérprete de Python se inicia en modo interactivo, busca un atributo **`__interactivehook__`** en el módulo **`sys`**. Si el atributo existe, se llama a su valor sin argumentos justo antes de iniciar el modo interactivo. La comprobación se realiza después de leer el fichero **`PYTHONSTARTUP`**, por lo que puede establecerse allí. El módulo **`site`** lo establece a una función que permite completar el tabulador y guardar el historial (en `~/.python-history`) si la plataforma soporta **`readline`**. Si no quiere este (nuevo) comportamiento, puede anularlo en **`PYTHONSTARTUP`**, `sitecustomize`, o `usercustomize` borrando este atributo de **`sys`** (o poniéndolo en alguna otra llamada). (Contribuido por Éric Araujo y Antoine Pitrou en bpo-5845 [https://bugs.python.org/issue?@action=redirect&bpo=5845].)

tarfile

El módulo `tarfile` soporta ahora una simple Interfaz de línea de comandos cuando se llama como un script directamente o a través de `-m`. Esto puede utilizarse para crear y extraer archivos tarfile. (Contribuido por Berker Peksag en bpo-13477 [https://bugs.python.org/issue?@action=redirect&bpo=13477].)

textwrap

La clase `TextWrapper` tiene dos nuevos atributos/argumentos de construcción: `max_lines`, que limita el número de líneas en la salida, y `placeholder`, que es una cadena que aparecerá al final de la salida si se ha truncado debido a *max_lines*. Basándose en estas capacidades, una nueva función de conveniencia `shorten()` colapsa todos los espacios en blanco de la entrada a espacios simples y produce una sola línea de un *ancho* dado que termina con el *marcador de posición* (por defecto, `[...]`). (Contribuido por Antoine Pitrou y Serhiy Storchaka en bpo-18585 [https://bugs.python.org/issue?@action=redirect&bpo=18585] y bpo-18725 [https://bugs.python.org/issue?@action=redirect&bpo=18725])

roscado

El objeto `Thread` que representa el hilo principal puede obtenerse de la nueva función `main_thread()`. En condiciones normales será el hilo desde el que se inició el intérprete de Python. (Contribuido por Andrew Svetlov en bpo-18882 [https://bugs.python.org/issue?@action=redirect&bpo=18882].)

traceback

Una nueva función `traceback.clear_frames()` toma un objeto traceback y borra las variables locales en todos los marcos a los que hace referencia, reduciendo la cantidad de memoria consumida. (Contribuido por Andrew Kuchling en bpo-1565525 [https://bugs.python.org/issue?@action=redirect&bpo=1565525].)

tipos

A new **`DynamicClassAttribute()`** descriptor provides a way to define an attribute that acts normally when looked up through an instance object, but which is routed to the *class* `__getattr__` when looked up through the class. This allows one to have properties active on a class, and have virtual attributes on the class with the same name (see **`enum`** for an example). (Contributed by Ethan Furman in bpo-19030 [https://bugs.python.org/issue?@action=redirect&bpo=19030].)

urllib

`urllib.request` ahora soporta URLs `data:` a través de la clase **`DataHandler`**. (Contribuido por Mathias Panzenböck en bpo-16423 [https://bugs.python.org/issue?@action=redirect&bpo=16423].)

El método http que será utilizado por una clase **`Request`** puede ahora especificarse estableciendo un atributo de clase **`method`** en la subclase. (Contribuido por Jason R Coombs en bpo-18978 [https://bugs.python.org/issue?@action=redirect&bpo=18978].)

Los objetos **`Request`** son ahora reutilizables: si los atributos **`full_url`** o **`data`** se modifican, todas las propiedades internas relevantes se actualizan. Esto significa, por ejemplo, que ahora es posible utilizar el mismo objeto **`Request`** en más de una llamada **`OpenerDirector.open()`** con diferentes argumentos *data*, o modificar la `url` de un **`Request`**` en lugar de volver a calcularla desde cero. También hay un nuevo método **`remove_header()`** que puede utilizarse para eliminar las cabeceras de una **`Request`**. (Contribuido por Alexey Kachayev en bpo-16464 [https://bugs.python.org/issue?@action=redirect&bpo=16464], Daniel Wozniak en bpo-17485 [https://bugs.python.org/issue?@action=redirect&bpo=17485], y Damien Brecht y Senthil Kumaran en bpo-17272 [https://bugs.python.org/issue?@action=redirect&bpo=17272])

Los objetos **`HTTPError`** tienen ahora un atributo **`headers`** que proporciona acceso a las cabeceras de respuesta HTTP asociadas al error. (Contribuido por Berker Peksag en bpo-15701 [https://bugs.python.org/issue?@action=redirect&bpo=15701].)

unittest

La clase **`TestCase`** tiene un nuevo método, **`subTest()`**, que produce un gestor de contexto cuyo bloque **`with`** se convierte en una «subprueba». Este gestor de contexto permite que un método de prueba genere subpruebas dinámicamente, por ejemplo, llamando al gestor de contexto `subTest` dentro de un bucle. Por lo tanto, un solo método de prueba puede producir un número indefinido de pruebas identificadas y contadas por separado, todas las cuales se ejecutarán incluso si una o más de ellas fallan. Por ejemplo:

```
class NumbersTest(unittest.TestCase):
    def test_even(self):
        for i in range(6):
            with self.subTest(i=i):
                self.assertEqual(i % 2, 0)
```

dará como resultado seis subpruebas, cada una identificada en la salida verbose de unittest con una etiqueta que consiste en el nombre de la variable `i` y un valor particular para esa variable (`i=0`, `i=1`, etc). Consulte la versión completa de este ejemplo en Distinguiendo iteraciones de tests empleando subtests. (Contribuido por Antoine Pitrou en bpo-16997 [https://bugs.python.org/issue?@action=redirect&bpo=16997].)

`unittest.main()` ahora acepta un iterable de nombres de pruebas para *defaultTest*, donde antes sólo aceptaba un único nombre de prueba como cadena. (Contribuido por Jyrki Pulliainen en bpo-15132 [https://bugs.python.org/issue?@action=redirect&bpo=15132].)

Si **`SkipTest`** se lanza durante el descubrimiento de la prueba (es decir, en el nivel de módulo en el archivo de prueba), ahora se reporta como un salto en lugar de un error. (Contribuido por Zach Ware en bpo-16935 [https://bugs.python.org/issue?@action=redirect&bpo=16935].)

`discover()` ahora ordena los archivos descubiertos para proporcionar un orden de pruebas consistente. (Contribuido por Martin Melin y Jeff Ramnani en bpo-16709 [https://bugs.python.org/issue?@action=redirect&bpo=16709].)

`TestSuite` ahora elimina las referencias a las pruebas tan pronto como la prueba se ha ejecutado, si la prueba tiene éxito. En los intérpretes de Python que hacen recolección de basura, esto permite que las pruebas sean recolectadas si no hay nada más que mantenga una referencia a la prueba. Es posible anular este comportamiento creando una subclase `TestSuite` que defina un método personalizado `_removeTestAtIndex`. (Contribución de Tom Wardill, Matt McClure y Andrew Svetlov en bpo-11798 [https://bugs.python.org/issue?@action=redirect&bpo=11798])

Un nuevo gestor de contexto de aserción de pruebas, `assertLogs()`, asegurará que un bloque de código dado emita un mensaje de registro utilizando el módulo `logging`. Por defecto, el mensaje puede provenir de cualquier registrador y tener una prioridad de `INFO` o superior, pero se puede especificar tanto el nombre del registrador como un nivel de registro mínimo alternativo. El objeto devuelto por el gestor de contexto puede ser consultado para los mensajes `LogRecord` s y/o formateados que fueron registrados. (Contribuido por Antoine Pitrou en bpo-18937 [https://bugs.python.org/issue?@action=redirect&bpo=18937].)

El descubrimiento de pruebas ahora funciona con paquetes de espacios de nombres (Contribuido por Claudiu Popa en bpo-17457 [https://bugs.python.org/issue?@action=redirect&bpo=17457].)

Los objetos `unittest.mock` ahora inspeccionan sus firmas de especificación cuando coinciden con las llamadas, lo que significa que un argumento ahora puede coincidir por posición o por nombre, en lugar de sólo por posición. (Contribuido por Antoine Pitrou en bpo-17015 [https://bugs.python.org/issue?@action=redirect&bpo=17015].)

`mock_open()` objects now have `readline` and `readlines` methods. (Contributed by Toshio Kuratomi in bpo-17467 [https://bugs.python.org/issue?@action=redirect&bpo=17467].)

venv

`venv` incluye ahora scripts de activación para los shells `csh` y `fish`. (Contribuido por Andrew Svetlov en bpo-15417 [https://bugs.python.org/issue?@action=redirect&bpo=15417].)

`EnvBuilder` y la función de conveniencia **`create()`** toman un nuevo argumento de palabra clave *with_pip*, que por defecto es `False`, que controla si **`EnvBuilder`** asegura que `pip` está instalado en el entorno virtual. (Contribuido por Nick Coghlan en bpo-19552 [https://bugs.python.org/issue?@action=redirect&bpo=19552] como parte de la implementación de **PEP 453** [https://peps.python.org/pep-0453/])

onda

The **`getparams()`** method now returns a namedtuple rather than a plain tuple. (Contributed by Claudiu Popa in bpo-17487 [https://bugs.python.org/issue?@action=redirect&bpo=17487].)

`wave.open()` ahora soporta el protocolo de gestión de contexto. (Contribuido por Claudiu Popa en bpo-17616 [https://bugs.python.org/issue?@action=redirect&bpo=17616].)

`wave` puede ahora escribir la salida en archivos no buscables. (Contribución de David Jones, Guilherme Polo y Serhiy Storchaka en bpo-5202 [https://bugs.python.org/issue?@action=redirect&bpo=5202])

Los métodos **`writeframesraw()`** y **`writeframes()`** aceptan ahora cualquier bytes-like object. (Contribuido por Serhiy Storchaka en bpo-8311 [https://bugs.python.org/issue?@action=redirect&bpo=8311].)

weakref

La nueva clase **`WeakMethod`** simula las referencias débiles a los métodos vinculados. (Contribuido por Antoine Pitrou en bpo-14631 [https://bugs.python.org/issue?@action=redirect&bpo=14631].)

La nueva clase **`finalize`** permite registrar una llamada de retorno para ser invocada cuando un objeto es recogido de la basura, sin necesidad de gestionar cuidadosamente el ciclo de vida de la propia referencia débil. (Contribuido por Richard Oudkerk en bpo-15528 [https://bugs.python.org/issue?@action=redirect&bpo=15528].)

La llamada de retorno, si la hay, asociada a un **`ref`** se expone ahora a través del atributo **`__callback__`**. (Contribuido por Mark Dickinson en bpo-17643 [https://bugs.python.org/issue?@action=redirect&bpo=17643].)

xml.etree

Un nuevo analizador sintáctico, **`XMLPullParser`**, permite que una aplicación no bloqueante analice documentos XML. Un ejemplo puede verse en API de consulta para un procesamiento no bloqueante. (Contribuido por Antoine Pitrou en bpo-17741 [https://bugs.python.org/issue?@action=redirect&bpo=17741].)

Las funciones **`xml.etree.ElementTree tostring()`** y **`tostringlist()`**, y el método **`ElementTree`** **`write()`**, tienen ahora un parámetro *short_empty_elements* parámetro de solo keyword que permite controlar si los elementos sin contenido se escriben de forma abreviada (`<tag />`) o expandida (`<tag></tag>`). (Contribución de Ariel Poliak y Serhiy Storchaka en bpo-14377 [https://bugs.python.org/issue?@action=redirect&bpo=14377])

archivo zip

El método **`writepy()`** de la clase **`PyZipFile`** tiene una nueva opción *filterfunc* que puede utilizarse para controlar qué directorios y ficheros se añaden al archivo. Por ejemplo, esto puede ser usado para excluir archivos de prueba del archivo. (Contribuido por Christian Tismer en bpo-19274 [https://bugs.python.org/issue?@action=redirect&bpo=19274].)

The *allowZip64* parameter to **`ZipFile`** and **`PyZipFile`** is now `True` by default. (Contributed by William Mallard in bpo-17201 [https://bugs.python.org/issue?@action=redirect&bpo=17201].)

Cambios en la implementación de

CPython

PEP 445: Personalización de los Asignadores de Memoria de CPython

PEP 445 [https://peps.python.org/pep-0445/] añade nuevas interfaces de nivel C para personalizar la asignación de memoria en el intérprete de CPython.

Ver también

PEP 445 [https://peps.python.org/pep-0445/] – Añadir nuevas APIs para personalizar los asignadores de memoria de Python

PEP escrito y aplicado por Victor Stinner.

PEP 442: Finalización segura de objetos

PEP 442 [https://peps.python.org/pep-0442/] removes the current limitations and quirks of object finalization in CPython. With it, objects with `__del__()` methods, as well as generators with `finally` clauses, can be finalized when they are part of a reference cycle.

Como parte de este cambio, los globales de los módulos ya no se establecen forzosamente a `None` durante el cierre del intérprete en la mayoría de los casos, sino que se confía en el funcionamiento normal del recolector de basura cíclico. Esto evita toda una clase de errores durante el cierre del intérprete, normalmente relacionados con los métodos `del__`, que han afectado a Python desde que se introdujo el GC cíclico.

Ver también

PEP 442 [https://peps.python.org/pep-0442/] – Finalización segura del objeto

PEP escrito y aplicado por Antoine Pitrou.

PEP 456: Algoritmo Hash seguro e intercambiable

PEP 456 [https://peps.python.org/pep-0456/] es la continuación de un trabajo anterior de corrección de seguridad realizado en el algoritmo hash de Python para hacer frente a ciertos ataques DOS a los que pueden estar sujetas las APIs de cara al público respaldadas por búsquedas de diccionario. (El PEP unifica el código hash de CPython para facilitar que un empaquetador sustituya un algoritmo hash diferente, y cambia la implementación por defecto de Python a una implementación SipHash en plataformas que tienen un tipo de datos de 64 bits. Cualquier diferencia de rendimiento en comparación con el antiguo algoritmo FNV es trivial.

El PEP agrega campos adicionales a la tupla con nombre `sys.hash_info` para describir el algoritmo hash que utiliza el binario que se está ejecutando actualmente. De lo contrario, el PEP no modifica ninguna API CPython existente.

PEP 436: Clínica de Argumentación

«Argument Clinic» (**PEP 436** [https://peps.python.org/pep-0436/]) es ahora parte del proceso de construcción de CPython y puede ser utilizado para simplificar el proceso de definición y mantenimiento de firmas precisas para builtins y módulos de extensión de la biblioteca estándar implementados en C.

Algunos módulos de extensión de la biblioteca estándar han sido convertidos para utilizar Argument Clinic en Python 3.4, y `pydoc` y `inspect` han sido actualizados en consecuencia.

Se espera que los metadatos de firma para la introspección programática se añadan a los callables adicionales implementados en C como parte de las versiones de mantenimiento de Python 3.4.

Nota

El PEP de Argument Clinic no está completamente actualizado con el estado de la implementación. Esto ha sido considerado aceptable por el director de la versión y el equipo de desarrollo del núcleo en este caso, ya que Argument Clinic no estará disponible como una API pública para el uso de terceros en Python 3.4.

Ver también

PEP 436 [https://peps.python.org/pep-0436/] – La clínica de argumentos DSL

PEP escrito y aplicado por Larry Hastings.

Otros cambios en la API de construcción y C

- La nueva función `PyType_GetSlot()` ha sido añadida a la ABI estable, permitiendo la recuperación de punteros a funciones desde ranuras de tipos con nombre cuando se utiliza la API limitada. (Contribuido por Martin von Löwis en bpo-17162 [https://bugs.python.org/issue?@action=redirect&bpo=17162].)
- La nueva API de preinicialización `Py_SetStandardStreamEncoding()` permite que las aplicaciones que incorporan el intérprete CPython fuercen de manera confiable una codificación y un controlador de errores particulares para los flujos estándar. (Contribución de Bastien Montagne y Nick Coghlan en bpo-16129 [https://bugs.python.org/issue?@action=redirect&bpo=16129]).
- La mayoría de las APIs de C de Python que no mutan los argumentos de cadena están ahora correctamente marcadas como aceptando `const char *` en lugar de `char *`. (Contribuido por Serhiy Storchaka en bpo-1772673 [https://bugs.python.org/issue?@action=redirect&bpo=1772673].)
- Una nueva versión de shell de `python-config` puede ser utilizada incluso cuando un intérprete de python no está disponible (por ejemplo, en escenarios de compilación cruzada).
- `PyUnicode_FromFormat()` ahora soporta especificaciones de anchura y precisión para `%s`, `%A`, `%U`, `%V`, `%S` y `%R`. (Contribución de Ysj Ray y Victor Stinner en bpo-7330 [https://bugs.python.org/issue?@action=redirect&bpo=7330].)
- La nueva función `PyStructSequence_InitType2()` complementa la función existente `PyStructSequence_InitType()`. La diferencia es que devuelve `0` en caso de éxito y `-1` en caso de fallo.
- El código fuente de CPython puede ahora ser compilado

usando las características de comprobación de sanidad de direcciones de las versiones recientes de GCC y clang: las falsas alarmas en el asignador de objetos pequeños han sido silenciadas. (Contribuido por Dhiru Kholia en bpo-18596 [https://bugs.python.org/issue?@action=redirect&bpo=18596].)

- La versión de Windows utiliza ahora Address Space Layout Randomization [https://en.wikipedia.org/wiki/Address_space_layout_randomization] y Data Execution Prevention [https://en.wikipedia.org/wiki/Data_Execution_Prevention]. (Contribuido por Christian Heimes en bpo-16632 [https://bugs.python.org/issue?@action=redirect&bpo=16632].)
- La nueva función `PyObject_LengthHint()` es el equivalente en la API de C de `operator.length_hint()`. (Contribuido por Armin Ronacher en bpo-16148 [https://bugs.python.org/issue?@action=redirect&bpo=16148].)

Otras mejoras

- El comando python tiene un nuevo option, `-I`, que hace que se ejecute en «modo aislado», lo que significa que `sys.path` no contiene ni el directorio del script ni el directorio `site-packages` del usuario, y se ignoran todas las variables de entorno **PYTHON*** (implica tanto `-s` como `-E`). También se pueden aplicar otras restricciones en el futuro, con el objetivo de aislar la ejecución de un script del entorno del usuario. Esto es apropiado, por ejemplo, cuando se utiliza Python para ejecutar un script del sistema. En la mayoría de los sistemas POSIX puede y debe utilizarse en la línea `#!` de scripts del sistema. (Contribuido por Christian Heimes en bpo-16499 [https://bugs.python.org/issue?@action=redirect&bpo=16499].)
- El completamiento de tabulaciones está ahora habilitado por defecto en el intérprete interactivo en los sistemas que soportan `readline`. El historial también está activado por defecto, y se escribe (y se lee) en el archivo `~/.python-history`. (Contribuido por Antoine Pitrou y Éric Araujo en bpo-5845 [https://bugs.python.org/issue?@action=redirect&bpo=5845].)
- La invocación del intérprete de Python con `--versión` ahora muestra la versión en la salida estándar en lugar de en el error estándar (bpo-18338 [https://bugs.python.org/issue?

@action=redirect&bpo=18338]). Se han realizado cambios similares en **`argparse`** (bpo-18920 [https://bugs.python.org/issue?@action=redirect&bpo=18920]) y en otros módulos que tienen capacidades de invocación tipo script (bpo-18922 [https://bugs.python.org/issue?@action=redirect&bpo=18922]).

- El instalador de CPython para Windows añade ahora `.py` a la variable **`PATHEXT`** cuando se registran las extensiones, permitiendo a los usuarios ejecutar un script de python en el símbolo del sistema de Windows con sólo escribir su nombre sin la extensión `.py`. (Contribuido por Paul Moore en bpo-18569 [https://bugs.python.org/issue?@action=redirect&bpo=18569].)
- Un nuevo objetivo `make`, coverage-report [https://devguide.python.org/coverage/#measuring-coverage-of-c-code-with-gcov-and-lcov], compilará Python, ejecutará el conjunto de pruebas y generará un informe de cobertura HTML para la base de código C utilizando `gcov` y lcov [https://github.com/linux-test-project/lcov].
- La opción `-R` de python regression test suite ahora también verifica si hay fugas de asignación de memoria, utilizando **`sys.getallocatedblocks()`**. (Contribuido por Antoine Pitrou en bpo-13390 [https://bugs.python.org/issue?@action=redirect&bpo=13390]).
- `python -m` ahora funciona con paquetes de espacios de nombres.
- El módulo **`stat`** está ahora implementado en C, lo que significa que obtiene los valores de sus constantes de los archivos de cabecera de C, en lugar de tener los valores codificados en el módulo de python como ocurría anteriormente.
- La carga de múltiples módulos de python desde un único módulo del sistema operativo (`.so`, `.dll`) ahora funciona correctamente (antes devolvía silenciosamente el primer módulo de python del archivo). (Contribuido por Václav Šmilauer en bpo-16421 [https://bugs.python.org/issue?@action=redirect&bpo=16421].)
- A new opcode, **`LOAD_CLASSDEREF`**, has been added to fix a bug in the loading of free variables in class bodies that could be triggered by certain uses of __prepare__. (Contributed by Benjamin Peterson in bpo-17853 [https://bugs.python.org/issue?

@action=redirect&bpo=17853].)

- Victor Stinner identificó y corrigió una serie de fallos relacionados con MemoryError utilizando su herramienta `pyfailmalloc` basada en **PEP 445** [https://peps.python.org/pep-0445/] (bpo-18408 [https://bugs.python.org/issue?@action=redirect&bpo=18408], bpo-18520 [https://bugs.python.org/issue?@action=redirect&bpo=18520]).
- El comando `pyvenv` acepta ahora la opción `--copies` para utilizar copias en lugar de enlaces simbólicos, incluso en sistemas en los que los enlaces simbólicos son los predeterminados. (Contribuido por Vinay Sajip en bpo-18807 [https://bugs.python.org/issue?@action=redirect&bpo=18807].)
- El comando `pyvenv` también acepta la opción `--without-pip` para suprimir el arranque automático de pip en el entorno virtual. (Contribuido por Nick Coghlan en bpo-19552 [https://bugs.python.org/issue?@action=redirect&bpo=19552] como parte de la implementación de **PEP 453** [https://peps.python.org/pep-0453/])
- El nombre de la codificación es ahora opcional en el valor establecido para la variable de entorno **`PYTHONIOENCODING`**. Esto hace posible establecer sólo el manejador de errores, sin cambiar la codificación por defecto. (Contribuido por Serhiy Storchaka en bpo-18818 [https://bugs.python.org/issue?@action=redirect&bpo=18818].)
- Las funciones del módulo **`bz2`**, **`lzma`** y **`gzip`** ahora soportan el modo `x` (creación exclusiva). (Contribución de Tim Heaney y Vajrasky Kok en bpo-19201 [https://bugs.python.org/issue?@action=redirect&bpo=19201], bpo-19222 [https://bugs.python.org/issue?@action=redirect&bpo=19222] y bpo-19223 [https://bugs.python.org/issue?@action=redirect&bpo=19223])

Optimizaciones significativas

- El decodificador de UTF-32 es ahora entre 3 y 4 veces más rápido. (Contribuido por Serhiy Storchaka en bpo-14625 [https://bugs.python.org/issue?@action=redirect&bpo=14625].)
- Se ha reducido el coste de las colisiones de hash para los conjuntos. Cada sondeo de la tabla hash comprueba ahora una serie de pares clave/hash consecutivos y adyacentes antes

de continuar haciendo sondeos aleatorios a través de la tabla hash. Esto aprovecha la localidad de la caché para que la resolución de colisiones sea menos costosa. El esquema de resolución de colisiones puede describirse como un híbrido de sondeo lineal y direccionamiento abierto. El número de sondeos lineales adicionales es por defecto de nueve. Esto puede cambiarse en tiempo de compilación definiendo LINEAR_PROBES con cualquier valor. Establezca LINEAR_PROBES=0 para desactivar completamente el sondeo lineal. (Contribuido por Raymond Hettinger en bpo-18771 [https://bugs.python.org/issue?@action=redirect&bpo=18771].)

- El intérprete comienza alrededor de 30% faster. Un par de medidas conducen a la aceleración. El intérprete carga menos módulos al iniciar, por ejemplo, los módulos **`re`**, **`collections`** y **`locale`** y sus dependencias ya no se importan por defecto. Se ha mejorado el módulo marshal para cargar más rápidamente el código Python compilado. (Contribución de Antoine Pitrou, Christian Heimes y Victor Stinner en bpo-19219 [https://bugs.python.org/issue?@action=redirect&bpo=19219], bpo-19218 [https://bugs.python.org/issue?@action=redirect&bpo=19218], bpo-19209 [https://bugs.python.org/issue?@action=redirect&bpo=19209], bpo-19205 [https://bugs.python.org/issue?@action=redirect&bpo=19205] y bpo-9548 [https://bugs.python.org/issue?@action=redirect&bpo=9548])
- **`bz2.BZ2File`** es ahora tan rápido o más que la versión de Python2 en la mayoría de los casos. **`lzma.LZMAFile`** también ha sido optimizado. (Contribuido por Serhiy Storchaka y Nadeem Vawda en bpo-16034 [https://bugs.python.org/issue?@action=redirect&bpo=16034])
- **`random.getrandbits()`** es un 20%-40% más rápido para enteros pequeños (el caso de uso más común). (Contribuido por Serhiy Storchaka en bpo-16674 [https://bugs.python.org/issue?@action=redirect&bpo=16674].)
- Al aprovechar el nuevo formato de almacenamiento de las cadenas, el decapado de las cadenas es ahora significativamente más rápido. (Contribución de Victor Stinner y Antoine Pitrou en bpo-15596 [https://bugs.python.org/issue?@action=redirect&bpo=15596].)
- A performance issue in **`io.FileIO.readall()`** has been

solved. This particularly affects Windows, and significantly speeds up the case of piping significant amounts of data through `subprocess`. (Contributed by Richard Oudkerk in bpo-15758 [https://bugs.python.org/issue?@action=redirect&bpo=15758].)

- `html.escape()` es ahora 10 veces más rápido. (Contribuido por Matt Bryant en bpo-18020 [https://bugs.python.org/issue?@action=redirect&bpo=18020].)
- En Windows, ahora se utiliza el `VirtualAlloc` nativo en lugar del `malloc` de CRT en `obmalloc`. Los benchmarks artificiales muestran un ahorro de memoria de alrededor del 3%.
- `os.urandom()` ahora usa un descriptor de archivo persistente abierto de forma diferida para evitar el uso de muchos descriptores de archivo cuando se ejecuta en paralelo desde varios subprocesos. (Aportado por Antoine Pitrou en bpo-18756 [https://bugs.python.org/issue?@action=redirect&bpo=18756].)

Obsoleto

Esta sección cubre varias APIs y otras características que han sido desaprobadas en Python 3.4, y que serán eliminadas en Python 3.5 o posterior. En la mayoría de los casos (pero no en todos), el uso de las APIs obsoletas producirá un `DeprecationWarning` cuando el intérprete se ejecute con las advertencias de obturación activadas (por ejemplo, usando `-Wd`).

Desapariciones en la API de Python

- Como se menciona en PEP 451: Un tipo ModuleSpec para el sistema de importación, varios métodos y funciones de `importlib` están obsoletos: **`importlib.find_loader()`** se reemplaza por `importlib.util.find_spec()`; **`importlib.machinery.PathFinder.find_module()`** se reemplaza por `importlib.machinery.PathFinder.find_spec()`; **`importlib.abc.MetaPathFinder.find_module()`** se reemplaza por

`importlib.abc.MetaPathFinder.find_spec()`; **`importlib.abc.PathEntryFinder.find_loader()`** y **`find_module()`** se reemplazan por **`importlib.abc.PathEntryFinder.find_spec()`**; todos los métodos ABC `load_module` de *xxx*`Loader` (**`importlib.abc.Loader.load_module()`**, **`importlib.abc.InspectLoader.load_module()`**, **`importlib.abc.FileLoader.load_module()`**, **`importlib.abc.SourceLoader.load_module()`**) ya no se deben implementar, en su lugar, los cargadores deben implementar un método `exec_module` (**`importlib.abc.Loader.exec_module()`**, **`importlib.abc.InspectLoader.exec_module()`** **`importlib.abc.SourceLoader.exec_module()`**) y dejar que el sistema de importación se encargue del resto; y **`importlib.abc.Loader.module_repr()`**, **`importlib.util.module_for_loader()`**, **`importlib.util.set_loader()`** y **`importlib.util.set_package()`** ya no son necesarios porque sus funciones ahora son manejadas automáticamente por el sistema de importación.

- El módulo **`imp`** está pendiente de ser eliminado. Para mantener la compatibilidad con las bases de código de Python 2/3, la eliminación del módulo no está programada actualmente.
- The **`formatter`** module is pending deprecation and is slated for removal in Python 3.6.
- `MD5` como *digestmod* por defecto para la función **`hmac.new()`** está en desuso. Python 3.6 requerirá un nombre de compendio o constructor explícito como argumento *digestmod*.
- La clase interna `Netrc` en el módulo **`ftplib`** ha sido documentada como obsoleta en su docstring durante bastante tiempo. Ahora emite un **`DeprecationWarning`** y se eliminará por completo en Python 3.5.
- El argumento indocumentado *endtime* de **`subprocess.Popen.wait()`** no debería haber sido expuesto y es de esperar que no esté en uso; es obsoleto y muy probablemente será eliminado en Python 3.5.
- El argumento *strict* de **`HTMLParser`** está obsoleto.

- The `plistlib` `readPlist()`, `writePlist()`, `readPlistFromBytes()`, and `writePlistToBytes()` functions are deprecated in favor of the corresponding new functions `load()`, `dump()`, `loads()`, and `dumps()`. `Data()` is deprecated in favor of just using the `bytes` constructor.
- La clave `sysconfig` `SO` está obsoleta, ha sido sustituida por `EXT_SUFFIX`.
- El modo `U` aceptado por varias funciones `open` está obsoleto. En Python3 no hace nada útil, y debe ser reemplazado por usos apropiados de `io.TextIOWrapper` (si es necesario) y su argumento *newline*.
- El argumento *parser* de `xml.etree.ElementTree.iterparse()` ha quedado obsoleto, al igual que el argumento *html* de `XMLParser()`. Para prepararse para la eliminación de este último, todos los argumentos de `XMLParser` deben pasarse por palabra clave.

Funciones obsoletas

- La ejecución de IDLE — Python editor and shell con la bandera `n` (sin subproceso) está obsoleta. Sin embargo, la función no se eliminará hasta que se resuelva el problema bpo-18823 [https://bugs.python.org/issue?@action=redirect&bpo=18823].
- El módulo site que añade un directorio «site-python» a sys.path, si existe, está obsoleto (bpo-19375 [https://bugs.python.org/issue?@action=redirect&bpo=19375]).

Removido

Sistemas operativos que ya no son compatibles

Se ha eliminado la compatibilidad con los siguientes sistemas operativos en las herramientas de origen y de compilación:

- OS/2 (bpo-16135 [https://bugs.python.org/issue?@action=redirect&bpo=16135]).
- Windows 2000 (changeset e52df05b496a).

- Sistemas Windows donde `COMSPEC` apunta a `command.com` (bpo-14470 [https://bugs.python.org/issue?@action=redirect&bpo=14470]).
- VMS (bpo-16136 [https://bugs.python.org/issue?@action=redirect&bpo=16136]).

Eliminación de la API y de las funciones

Se han eliminado las siguientes APIs y características obsoletas y previamente obsoletas:

- Los directorios `Misc/TextMate` y `Misc/vim` no mantenidos han sido eliminados (ver la devguide [https://devguide.python.org] para sugerencias sobre qué usar en su lugar).
- Se ha eliminado la macro makefile `SO` (se ha sustituido por las macros `SHLIB_SUFFIX` y `EXT_SUFFIX`) (bpo-16754 [https://bugs.python.org/issue?@action=redirect&bpo=16754]).
- Se ha eliminado el campo `PyThreadState.tick_counter`; su valor no tiene sentido desde Python 3.2, cuando se introdujo el «nuevo GIL» (bpo-19199 [https://bugs.python.org/issue?@action=redirect&bpo=19199]).
- `PyLoader` y `PyPycLoader` han sido eliminados de **`importlib`**. (Contribuido por Taras Lyapun en bpo-15641 [https://bugs.python.org/issue?@action=redirect&bpo=15641].)
- Se ha eliminado el argumento *estricto* de **`HTTPConnection`** y **`HTTPSConnection`**. Ya no se admiten las «respuestas simples» del estilo de HTTP 0.9.
- Los métodos getter y setter **`urllib.request`** obsoletos `add_data`, `has_data`, `get_data`, `get_type`, `get_host`, `get_selector`, `set_proxy`, `get_origin_req_host`, y `is_unverifiable` han sido eliminados (utilice en su lugar el acceso directo a atributos).
- Se ha eliminado el soporte para cargar el obsoleto `TYPE_INT64` de **`marshal`**. (Contribuido por Dan Riti en bpo-15480 [https://bugs.python.org/issue?@action=redirect&bpo=15480].)
- **`inspect.Signature`**: ahora se requiere que los parámetros sólo posicionales tengan un nombre válido.

- **`object.__format__()`** ya no acepta cadenas de formato no vacías, ahora lanza un **`TypeError`** en su lugar. El uso de una cadena no vacía ha sido desaprobado desde Python 3.2. Este cambio se ha hecho para evitar una situación en la que el código que antes funcionaba (pero era incorrecto) empezaba a fallar si un objeto ganaba un método _format_, lo que significa que su código puede ahora lanzar un **`TypeError`** si está usando un código de formato `s` con objetos que no tienen un método _format_ que lo maneje. Véase bpo-7994 [https://bugs.python.org/issue?@action=redirect&bpo=7994] para más información.
- **`difflib.SequenceMatcher.isbjunk()`** and **`difflib.SequenceMatcher.isbpopular()`** were deprecated in 3.2, and have now been removed: use `x in sm.bjunk` and `x in sm.bpopular`, where *sm* is a **`SequenceMatcher`** object (bpo-13248 [https://bugs.python.org/issue?@action=redirect&bpo=13248]).

Limpieza del código

- Se ha eliminado la clase interna `Scanner` no utilizada y no documentada del módulo **`pydoc`**.
- Se ha eliminado el módulo privado y efectivamente inutilizado `_gestalt`, junto con las funciones privadas de **`platform`** `_mac_ver_lookup`, `_mac_ver_gstalt`, y `_bcd2str`, que sólo habrían sido llamadas en sistemas OSX muy estropeados (ver bpo-18393 [https://bugs.python.org/issue?@action=redirect&bpo=18393]).
- Se han eliminado las copias codificadas de ciertas constantes **`stat`** que se incluían en el espacio de nombres del módulo **`tarfile`**.

Adaptación a Python 3.4

Esta sección enumera los cambios descritos anteriormente y otras correcciones de errores que pueden requerir cambios en su código.

Cambios en el comportamiento del comando “python”

- En un shell posix, establecer la variable de entorno **`PATH`** a un valor vacío es equivalente a no establecerla en absoluto. Sin embargo, establecer **`PYTHONPATH`** a un valor vacío era *no* equivalente a no establecerla en absoluto: establecer **`PYTHONPATH`** a un valor vacío era equivalente a establecerla a `.`, lo que lleva a confusión cuando se razona por analogía con el funcionamiento de **`PATH`**. El comportamiento ahora se ajusta a la convención posix para **`PATH`**.
- La salida [X refs, Y blocks] de una compilación de depuración (`--con-pydebug`) del intérprete de CPython está ahora desactivada por defecto. Se puede volver a activar usando la opción `-X showrefcount`. (Contribuido por Ezio Melotti en bpo-17323 [https://bugs.python.org/issue?@action=redirect&bpo=17323].)
- El comando python y la mayoría de los scripts de stdlib (así como **`argparse`**) ahora envían la información de `--version` a `stdout` en lugar de a `stderr` (para la lista de problemas, véase Otras mejoras más arriba).

Cambios en la API de Python

- Los ABCs definidos en **`importlib.abc`** ahora lanzan la excepción apropiada o devuelven un valor por defecto en lugar de lanzar **`NotImplementedError`** a ciegas. Esto sólo afectará al código que llame a **`super()`** y que llegue hasta el ABC. Por compatibilidad, capture tanto **`NotImplementedError`** como la excepción apropiada según sea necesario.
- The module type now initializes the **`__package__`** and **`__loader__`** attributes to `None` by default. To determine if these attributes were set in a backwards-compatible fashion, use e.g. `getattr(module, '__loader__', None) is not None`. (bpo-17115 [https://bugs.python.org/issue?@action=redirect&bpo=17115].)
- **`importlib.util.module_for_loader()`** now sets `__loader__` and `__package__` unconditionally to properly support reloading. If this is not desired then you will need to set these attributes manually. You can use **`importlib.util.module_to_load()`** for module management.

- Importar ahora restablece los atributos relevantes (por ejemplo, `__name__`, `__loader__`, `__package__`, `__file__`, `__cached__`) incondicionalmente cuando se recarga. Tenga en cuenta que esto restablece un comportamiento anterior a la versión 3.3 en el sentido de que un módulo se vuelve a encontrar cuando se recarga (bpo-19413 [https://bugs.python.org/issue?@action=redirect&bpo=19413]).
- Los paquetes congelados ya no establecen `ruta__` en una lista que contenga el nombre del paquete, ahora lo hacen en una lista vacía. El comportamiento anterior podía hacer que el sistema de importación hiciera algo incorrecto en las importaciones de submódulos si también había un directorio con el mismo nombre que el paquete congelado. La forma correcta de determinar si un módulo es un paquete o no es utilizar `hasattr(module, '__path__')` (bpo-18065 [https://bugs.python.org/issue?@action=redirect&bpo=18065]).
- Los módulos congelados ya no definen un atributo `__file__`. Es semánticamente incorrecto que los módulos congelados establezcan el atributo, ya que no se cargan desde ninguna ubicación explícita. Si debe saber que un módulo proviene de un código congelado, puede ver si el `__spec__.location` del módulo está configurado en `'frozen'`, verificar si el cargador es una subclase de **`importlib.machinery.FrozenImporter`** o, si es necesaria la compatibilidad con Python 2, puede usar **`imp.is_frozen()`**.
- **`py_compile.compile()`** ahora lanza **`FileExistsError`** si la ruta de archivo en la que escribiría es un enlace simbólico o un archivo no regular. Esto es para actuar como una advertencia de que la importación sobrescribirá esos archivos con un archivo regular sin importar el tipo de ruta de archivo que eran originalmente.
- **`importlib.abc.SourceLoader.get_source()`** ya no lanza **`ImportError`** cuando el código fuente que se carga provoca un **`SyntaxError`** o **`UnicodeDecodeError`**. Como **`ImportError`** está pensado para ser levantado sólo cuando el código fuente no puede ser encontrado pero debería, se sintió que se sobrepasaba/sobrecargaba ese significado cuando el código fuente es encontrado pero inadecuadamente

estructurado. Si usted estaba atrapando ImportError antes y desea continuar ignorando los problemas de sintaxis o decodificación, atrape las tres excepciones ahora.

- `functools.update_wrapper()` y `functools.wraps()` ahora establecen correctamente el atributo `wrapped__` a la función que está siendo envuelta, incluso si esa función también tiene su atributo `wrapped__` establecido. Esto significa que los atributos `__wrapped__` ahora enlazan correctamente una pila de funciones decoradas en lugar de que cada atributo `__wrapped__` en la cadena haga referencia a la función más interna. Las bibliotecas de introspección que asumían que el comportamiento anterior era intencionado pueden utilizar `inspect.unwrap()` para acceder a la primera función de la cadena que no tenga el atributo `__wrapped__`.
- `inspect.getfullargspec()` ha sido reimplementado sobre `inspect.signature()` y por lo tanto maneja una variedad mucho más amplia de objetos llamables que en el pasado. Se espera que otras llamadas de módulos incorporados y de extensión obtengan metadatos de firma en el curso de la serie Python 3.4. El código que asume que **`inspect.getfullargspec()`** fallará en las llamadas que no son de Python puede necesitar ser ajustado en consecuencia.
- `importlib.machinery.PathFinder` ahora pasa el directorio de trabajo actual a los objetos en `sys.path_hooks` para la cadena vacía. Esto hace que `sys.path_importer_cache` nunca contenga `''`, por lo que iterar a través de **`sys.path_importer_cache`** basándose en `sys.path` no encontrará todas las claves. El `__file__` de un módulo cuando se importa en el directorio de trabajo actual también tendrá ahora una ruta absoluta, incluso cuando se utiliza `-m` con el intérprete (excepto para `__main__.__file__` cuando un script se ha ejecutado directamente utilizando una ruta relativa) (Contribuido por Brett Cannon en bpo-18416 [https://bugs.python.org/issue?@action=redirect&bpo=18416]). se especifica en la línea de comandos) (bpo-18416 [https://bugs.python.org/issue?@action=redirect&bpo=18416]).
- La eliminación del argumento *estricto* de `HTTPConnection`

y `HTTPSConnection` cambia el significado de los argumentos restantes si los especifica posicionalmente en lugar de por palabra clave. Si ha prestado atención a las advertencias de desaprobación, su código ya debería especificar los argumentos adicionales mediante palabras clave.

- Las cadenas entre las sentencias `from __future__ import ...` ahora *siempre* lanzan un `SyntaxError`. Anteriormente, si no había un docstring principal, a veces se ignoraba una cadena intersticial. Esto hace que CPython cumpla con la especificación del lenguaje; Jython y PyPy ya lo hacían. (bpo-17434 [https://bugs.python.org/issue?@action=redirect&bpo=17434]).
- `ssl.SSLSocket.getpeercert()` y `ssl.SSLSocket.do_handshake()` ahora lanzan un `OSError` con `ENOTCONN` cuando el `SSLSocket` no está conectado, en lugar del comportamiento anterior de lanzar un `AttributeError`. Además, **`getpeercert()`** lanzará un `ValueError` si el handshake aún no se ha realizado.
- `base64.b32decode()` ahora genera un `binascii.Error` cuando la cadena de entrada contiene caracteres que no son de tipo b32, en lugar de un `TypeError`. Este **`TypeError`** en particular se perdió cuando se convirtieron los otros **`TypeError`**. (Contribuido por Serhiy Storchaka en bpo-18011 [https://bugs.python.org/issue?@action=redirect&bpo=18011].) Nota: este cambio también se aplicó inadvertidamente en Python 3.3.3.
- El atributo **`file`** ahora se cierra automáticamente cuando se realiza la recolección de basura de la instancia **`cgi.FieldStorage`** que la crea. Si extrajo el objeto de archivo por separado de la instancia **`cgi.FieldStorage`** y no mantuvo activa la instancia, entonces debe almacenar toda la instancia **`cgi.FieldStorage`** o leer el contenido del archivo antes de que se realice la recolección de basura de la instancia **`cgi.FieldStorage`**.
- Llamar a `lectura` o `escritura` en un socket SSL cerrado ahora genera un informativo `ValueError` en lugar del anterior y más misterioso `AttributeError` (bpo-9177 [https://bugs.python.org/issue?@action=redirect&bpo=9177]).
- `slice.indices()` ya no produce un `OverflowError`

para valores enormes. Como consecuencia de esta corrección, **`slice.indices()`** ahora produce un **`ValueError`** si se le da una longitud negativa; antes devolvía valores sin sentido (bpo-14794 [https://bugs.python.org/issue?@action=redirect&bpo=14794]).

- El constructor **`complex`**, a diferencia de las funciones **`cmath`**, aceptaba incorrectamente valores **`float`** si el método especial `__complex__` de un objeto devolvía uno. Ahora se produce un error **`TypeError`**. (bpo-16290 [https://bugs.python.org/issue?@action=redirect&bpo=16290].)
- El constructor **`int`** de las versiones 3.2 y 3.3 aceptaba erróneamente valores **`float`** para el parámetro *base*. Es poco probable que alguien esté haciendo esto, pero si es así, ahora lanzará un **`TypeError`** (bpo-16772 [https://bugs.python.org/issue?@action=redirect&bpo=16772]).
- Los valores por defecto de los argumentos de palabras clave se evalúan ahora *después* de los valores por defecto de los argumentos de palabras clave normales, en lugar de antes. Esperemos que nadie haya escrito ningún código que dependa del comportamiento erróneo anterior (bpo-16967 [https://bugs.python.org/issue?@action=redirect&bpo=16967]).
- Los estados de los hilos antiguos se borran ahora después de **`fork()`**. Esto puede hacer que se liberen algunos recursos del sistema que antes se mantenían vivos de forma incorrecta (por ejemplo, las conexiones a la base de datos mantenidas en el almacenamiento local del hilo). (bpo-17094 [https://bugs.python.org/issue?@action=redirect&bpo=17094].)
- Los nombres de parámetros en los diccionarios `__annotations__` ahora se modifican correctamente, de manera similar a **`__kwdefaults__`**. (Contribuido por Yury Selivanov en bpo-20625 [https://bugs.python.org/issue?@action=redirect&bpo=20625]).
- **`hashlib.hash.name`** ahora siempre devuelve el identificador en minúsculas. Anteriormente, algunos hashes incorporados tenían nombres en mayúsculas, pero ahora que se trata de una interfaz pública formal, la denominación se ha hecho consistente (bpo-18532 [https://bugs.python.org/issue?@action=redirect&bpo=18532]).
- Debido a que **`unittest.TestSuite`** ahora elimina las referencias a las pruebas después de que se ejecutan, los

conjuntos de pruebas que reutilizan un **`TestSuite`** para volver a ejecutar un conjunto de pruebas pueden fallar. Los conjuntos de pruebas no se deben reutilizar de esta manera, ya que significa que el estado se conserva entre ejecuciones de pruebas, lo que rompe el aislamiento de pruebas que **`unittest`** está diseñado para proporcionar. Sin embargo, si la falta de aislamiento se considera aceptable, el comportamiento anterior se puede restaurar creando una subclase **`TestSuite`** que defina un método `_removeTestAtIndex` que no haga nada (consulte **`TestSuite.__iter__()`**) (bpo-11798 [https://bugs.python.org/issue?@action=redirect&bpo=11798]).

- **`unittest`** ahora utiliza **`argparse`** para el análisis de la línea de comandos. Hay ciertas formas de comando inválidas que solían funcionar y que ya no están permitidas; en teoría, esto no debería causar problemas de compatibilidad con versiones anteriores, ya que las formas de comando no permitidas no tenían ningún sentido y es poco probable que estén en uso.
- The **`re.split()`**, **`re.findall()`**, and **`re.sub()`** functions, and the **`group()`** and **`groups()`** methods of `match` objects now always return a *bytes* object when the string to be matched is a bytes-like object. Previously the return type matched the input type, so if your code was depending on the return value being, say, a `bytearray`, you will need to change your code.
- Las funciones de **`audioop`** ahora lanzan un error inmediatamente si se les pasa una cadena de entrada, en lugar de fallar aleatoriamente más tarde (bpo-16685 [https://bugs.python.org/issue?@action=redirect&bpo=16685]).
- El nuevo argumento *convert_charrefs* de **`HTMLParser`** tiene por defecto el valor `False` por compatibilidad con versiones anteriores, pero con el tiempo se cambiará a `True`. Se recomienda añadir esta palabra clave, con el valor apropiado, a cualquier llamada de **`HTMLParser`** en su código (bpo-13633 [https://bugs.python.org/issue?@action=redirect&bpo=13633]).
- Dado que el argumento *digestmod* de la función **`hmac.new()`** no tendrá en el futuro ningún valor por defecto, todas las llamadas a **`hmac.new()`** deben cambiarse para especificar

explícitamente un *digestmod* (bpo-17276 [https://bugs.python.org/issue?@action=redirect&bpo=17276]).

- Llamar a **`sysconfig.get_config_var()`** con la clave `SO`, o buscar `SO` en los resultados de una llamada a **`sysconfig.get_config_vars()`** está obsoleto. Esta clave debería ser reemplazada por `EXT_SUFFIX` o `SHLIB_SUFFIX`, dependiendo del contexto (bpo-19555 [https://bugs.python.org/issue?@action=redirect&bpo=19555]).
- Cualquier llamada a las funciones `open` que especifique `U` debe ser modificada. `U` es ineficaz en Python3 y acabará dando un error si se utiliza. Dependiendo de la función, el equivalente a su antiguo comportamiento en Python2 puede conseguirse usando un argumento *newline*, o si es necesario envolviendo el flujo en **`TextIOWrapper`** para usar su argumento *newline* (bpo-15204 [https://bugs.python.org/issue?@action=redirect&bpo=15204]).
- Si utilizas `pyvenv` en un script y deseas que pip *no* se instale, debes añadir `--without-pip` a tu invocación del comando.
- El comportamiento por defecto de **`json.dump()`** y **`json.dumps()`** cuando se especifica una sangría ha cambiado: ya no produce espacios finales después de las comas de separación al final de las líneas. Esto sólo importará si tiene pruebas que hacen comparaciones sensibles a los espacios en blanco de dicha salida (bpo-16333 [https://bugs.python.org/issue?@action=redirect&bpo=16333]).
- **`doctest`** ahora busca doctests en las cadenas del módulo de extensión `__doc__`, por lo que si su descubrimiento de pruebas de doctest incluye módulos de extensión que tienen cosas que parecen doctests en ellos, puede ver fallos de prueba que nunca ha visto antes al ejecutar sus pruebas (bpo-3158 [https://bugs.python.org/issue?@action=redirect&bpo=3158]).
- El módulo **`collections.abc`** ha sido ligeramente refactorizado como parte de las mejoras de inicio de Python. Como consecuencia de esto, ya no se da el caso de que al importar **`collections`** se importe automáticamente **`collections.abc`**. Si su programa dependía de la importación implícita (no documentada), tendrá que añadir un `importar colecciones.abc` explícitamente (bpo-20784 [https://bugs.python.org/issue?

@action=redirect&bpo=20784]).

Cambios en la API de C

- `PyEval_EvalFrameEx()`, `PyObject_Repr()`, y `PyObject_Str()`, junto con algunas otras APIs internas de C, incluyen ahora una aserción de depuración que asegura que no se utilicen en situaciones en las que puedan descartar silenciosamente una excepción actualmente activa. En los casos en los que se espera y se desea descartar la excepción activa (por ejemplo, porque ya se ha guardado localmente con `PyErr_Fetch()` o se está sustituyendo deliberadamente por una excepción diferente), se necesitará una llamada explícita a `PyErr_Clear()` para evitar que se active la aserción cuando se invoquen estas operaciones (directa o indirectamente) y se ejecuten contra una versión de Python que esté compilada con las aserciones activadas.
- `PyErr_SetImportError()` ahora establece `TypeError` cuando su argumento **msg** no está establecido. Anteriormente sólo se devolvía `NULL` sin establecer ninguna excepción.
- El resultado de la llamada de retorno de `PyOS_ReadlineFunctionPointer` debe ser ahora una cadena asignada por `PyMem_RawMalloc()` o `PyMem_RawRealloc()`, o `NULL` si se ha producido un error, en lugar de una cadena asignada por `PyMem_Malloc()` o `PyMem_Realloc()` (bpo-16742 [https://bugs.python.org/issue?@action=redirect&bpo=16742])
- `PyThread_set_key_value()` ahora siempre establece el valor. En Python 3.3, la función no hacía nada si la clave ya existía (si el valor actual es un puntero no `NULL`).
- Se ha eliminado el campo `f_tstate` (estado del hilo) de la estructura `PyFrameObject` para corregir un error: véase bpo-14432 [https://bugs.python.org/issue?@action=redirect&bpo=14432] para conocer los motivos.

Cambiado en 3.4.3

PEP 476: Habilitar la verificación de certificados por defecto para los clientes http stdlib

`http.client` y los módulos que lo utilizan, como **`urllib.request`** y **`xmlrpc.client`**, ahora verificarán que el servidor presenta un certificado que está firmado por una CA en el almacén de confianza de la plataforma y cuyo nombre de host coincide con el nombre de host que se solicita por defecto, mejorando significativamente la seguridad de muchas aplicaciones.

Para las aplicaciones que requieren el antiguo comportamiento anterior, pueden pasar un contexto alternativo:

```
import urllib.request
import ssl

# This disables all verification
context = ssl._create_unverified_context()

# This allows using a specific certificate for the host,
# to be in the trust store
context = ssl.create_default_context(cafile="/path/to/fi

urllib.request.urlopen("https://invalid-cert", context=c
```

Qué hay de nuevo en Python 3.3

Éste artículo explica las nuevas características en python 3.3, en comparación con la versión 3.2. Python 3.3 se lanzó el 29 de septiembre de 2012. Para más detalles, ver el log de cambios en la dirección <https://docs.python.org/3.3/whatsnew/changelog.html>.

Ver también

PEP 398 [https://peps.python.org/pep-0398/] - Calendario de Publicación Python 3.3

Resumen – Aspectos destacados de la versión

Nuevas características de sintaxis:

- Nueva expresión `yield from` para delegación de generadores.
- La sintaxis `u'unicode'` se acepta de nuevo para los objetos **`str`**.

Nuevos módulos de biblioteca:

- **`faulthandler`** (ayuda a depurar bloqueos de bajo nivel)
- **`ipaddress`** (Objetos de alto nivel que representan direcciones IP y máscaras)
- **`lzma`** (comprime datos usando el algoritmo XZ / LZMA)
- **`unittest.mock`** (Utiliza objetos que simulen partes de tu sistema en ambiente de pruebas)
- **`venv`** (Python entornos virtuales, como en el paquete original Virtualenv)

Nuevas características incorporadas:

- Se reformula la Jerarquía de excepción I/O .

Mejoras de implementación:

- Se reescribe Maquinaria de importación con base en **`importlib`**.
- cadenas de caracteres unicode más compactas.
- diccionarios de atributos más compactos.

Módulos de biblioteca con mejoras significativas:

- Acelerador de C para el módulo decimal .
- Mejor gestión de *unicode* en el módulo email (provisional).

Mejoras de seguridad:

- La aleatoriedad en los hash está habilitada de forma predeterminada.

Por favor continúa leyendo para obtener una lista completa de los cambios de cara al usuario.

PEP 405: Entornos virtuales

Los entornos virtuales ayudan a crear configuraciones separadas de Python mientras comparten una instalación base en todo el sistema, para facilitar el mantenimiento. Los entornos virtuales tienen su propio conjunto de paquetes de sitios privados (es decir, bibliotecas instaladas localmente) y están opcionalmente segregados de los paquetes de sitios de todo el sistema. Su concepto e implementación están inspirados en el popular paquete de terceros `virtualenv`, pero se benefician de una integración más estrecha con el núcleo del intérprete.

Este PEP agrega el módulo **`venv`** para acceso programático, y el *script* `pyvenv` para administración y acceso a través de línea de comandos. El intérprete de python busca un archivo llamado `pyvenv.cfg`, cuya existencia señala la base de un árbol de directorios en el entorno virtual.

Ver también

PEP 405 [https://peps.python.org/pep-0405/] - Entornos virtuales en python

PEP escrito por Carl Meyer; implementación por Carl Meyer y Vinay Sajip

PEP 420: Paquetes para espacios de nombres implícitos

Soporte nativo para directorios de paquetes que no requieran el archivo `__init.py__` y puedan abarcar automáticamente múltiples segmentos de rutas de acceso (Inspirado en varios enfoques de terceros para espacios de nombres de paquetes, tal como está especificado en PEP 420 [https://peps.python.org/pep-0420/])

Ver también

PEP 420 [https://peps.python.org/pep-0420/] - Paquetes de espacio de nombres implícitos

PEP escrito por Eric V. Smith; implementación por Eric V. Smith y Barry Warsaw

PEP 3118: Nueva implementación de vista de memoria y en la documentación del protocolo del buffer

La implementación del PEP 3118 [https://peps.python.org/pep-3118/] ha tenido mejoras significativas.

La nueva implementación de la vista de memoria corrige exhaustivamente todos los problemas de propiedad y duración de los campos asignados dinámicamente en la estructura de Py_buffer que ha dado lugar a varios informes de bloqueo. Además, se han corregido varias funciones que se bloquearon o retornaron resultados incorrectos para entradas no contigua o multidimensional.

El objeto de vista de memoria dispone ahora de PEP-3118 que obedece a getbufferproc(), el cual verifica el tipo de petición del cliente. Se han agregado nuevas características, la mayoría de ellas funcionan generalmente para arrays no contiguos y arrays con subindices.

La documentación se ha actualizado y se han establecido claramente las responsabilidades tanto para los exportadores como para los consumidores. Los indicadores de solicitud de buffer se agrupan en indicadores básicos y compuestos. Se explica el diseño de memoria de matrices de estilo NumPy no contiguas y multidimensionales.

Características

- Ahora se admiten todos los especificadores nativos de formato de un solo carácter en la sintaxis del módulo de estructura (opcionalmente con el prefijo "@").
- Con algunas reestricciones, el método cast() permite cambiar el formato y forma de los arrays contiguos.
- La representación de listas multidimensionales es aceptada por cualquier tipo de array.
- Se admiten comparaciones multidimensionales para cualquier tipo de array.
- Los objetos de vista de memoria unidimensionales de tipo hash con formatos B, b, o c ahora son hashables (Contribución por Antoine Pitrou en el issue *13411*.)
- Se admite la segmentación arbitraria de cualquier tipo de matriz 1-D. Por ejemplo, ahora es posible invertir una vista de memoria en $O(1)$ mediante un paso negativo.

Cambios en la API

- Oficialmente, se limita el número máximo de dimensiones a 64.
- Las formas vacías, pasos, y subíndices ahora son representados por una tupla vacía, en vez de `None`.
- El acceso al elemento de una vista de memoria con formato

“B” (bytes sin signo) ahora retorna un entero (de acuerdo con la sintaxis del módulo de estructura). Para retornar un objeto *bytes*, primero la vista debe ser convertida a «C».
- las comparaciones de la vista de memoria ahora utilizan la estructura lógica de los operandos y comparan todos los elementos de array por valor. Se admiten todos los formatos strings en la sintaxis del módulo de estructura. Aún se admiten las vistas con strings que tengan un formato no reconocido, pero siempre se compararan como desiguales, independientemente del contenido de la vista.
- Para más cambios, consulte Compilación y cambios en la API de C y Portabilidad del código C.

(Contribución por Stefan Krah en el issue *10181* .)

Ver también

PEP 3118 [https://peps.python.org/pep-3118/] - Revisando el protocolo de buffer

PEP 393: Representación flexible de cadenas de caracteres

El tipo de cadena *Unicode* es modificado para admitir múltiples representaciones internas, dependiendo del caracter con el ordinal Unicode más grande (1, 2 o 4 *bytes*) en la cadena representada. Esto permite una representación que ahorra espacio en casos comunes, pero da acceso a UCS-4 completo en todos los sistemas. Para compatibilidad con APIS existentes, pueden existir varias representaciones en paralelo; con el tiempo, esta compatibilidad debería eliminarse gradualmente.

Por parte de Python, no debería haber inconvenientes por éste cambio.

Por parte de la API de C, **PEP 393** [https://peps.python.org/pep-0393/] es totalmente compatible con versiones anteriores. La API antigua debería quedar disponible al menos por cinco años. Las aplicaciones que usan la API antigua no saldrán beneficiadas por la reducción de memoria, o -peor- podrían usar un bit más de memoria, debido a que python necesitaría mantener dos versiones de cada cadena de caracteres (en el formato antiguo y en el nuevo almacenamiento)

Funcionalidad

Los cambios introducidos en el **PEP 393** [https://peps.python.org/pep-0393/] son los siguientes:

- Ahora python admite siempre el rango completo de códigos Unicode, incluyendo los que no son BMP (es decir, `U+0000` to `U+10FFFF`). La distinción entre ancho y angosto ya no existe, y python se comporta ahora como una compilación amplia, incluso en sistemas windows.
- Con la eliminación de las construcciones estrechas, han sido solucionados también los problemas específicos referentes a este tipo de construcciones, por ejemplo:
    - `len()` Ahora siempre retorna 1 para los caracteres que no son BMP, de modo que `len('\U0010FFFF') == 1`;
    - los pares sustitutos no se recombinan en cadenas literales, de modo que `'\uDBFF\uDFFF' != '\U0010FFFF'`;
    - la indexación o el corte de caracteres que no son BMP, retorna el valor esperado, de modo que `'\U0010FFFF'[0]` ahora retorna `'\U0010FFFF'` y no `'\uDBFF'`;
    - todas las demás funciones en la librería estándar ahora gestionan de forma correcta los puntos de código que no son BMP.
- El valor de `sys.maxunicode` ahora es siempre `1114111` (`0x10FFFF` en hexadecimal). La función `PyUnicode_GetMax()` aún devuelve `0xFFFF` o `0x10FFFF` por compatibilidad con versiones anteriores y no debe usarse con la nueva API Unicode (consulte bpo-13054 [https://bugs.python.org/issue?@action=redirect&bpo=13054]).
- Se ha eliminado la bandera `./configure --with-wide-unicode`.

Rendimiento y uso de recursos

El almacenamiento de las cadenas *Unicode* ahora depende del punto de código más alto en la cadena:

- las cadenas de caracteres en ASCII puro y Latin1 (`U+0000-U+00FF`) usan 1 byte por punto de código;
- Las cadenas de caracteres BMP (`U+0000-U+FFFF`) usan 2 *bytes* por punto de código;
- las cadenas de caracteres que no son BMP (`U+10000-U+10FFFF`) usan 4 *bytes* por punto de código.

El efecto neto es que para la mayoría de las aplicaciones, el uso de memoria del almacenamiento de cadenas de caracteres debería disminuir significativamente, especialmente en comparación con las construcciones *unicode* anchas, ya que, en muchos casos, las cadenas serán ASCII puro incluso en contextos internacionales (porque muchas cadenas almacenan datos en lenguaje no humano, tal como fragmentos XML, encabezados HTTP, datos codificados en JSON, etc.). También esperamos que, por las mismas razones, aumente la eficiencia de la caché de la CPU en aplicaciones no triviales. El uso de memoria de Python 3.3 es dos o tres veces más pequeño que Python 3.2, y un poco mejor que Python 2.7, en un punto de referencia de Django (consulte el PEP para obtener más detalles).

Ver también

PEP 393 [https://peps.python.org/pep-0393/] - Representación flexible de cadenas de caracteres

PEP escrito por Martin von Lowis; implementado por Torsten Becker y Martin von Lowis.

PEP 397: Lanzador de python para windows

El instalador de Python 3.3 para windows ahora incluye una aplicación lanzadora `py` que puede ser utilizada para ejecutar aplicaciones python de forma independiente de la versión.

Este lanzador se invoca de forma implícita cuando haces doble click en archivos `*.py`. Si el sistema tiene instalada una única versión de python, ésta versión será usada para ejecutar el archivo. Si hay múltiples versiones instaladas en el sistema, se utilizara por defecto la versión más reciente pero ésto puede ser anulado incluyendo una «línea shebang» estilo Unix en el *script* de Python.

El lanzador también se puede usar explícitamente desde la línea de comando como la aplicación `py`. La ejecución de `py` sigue las mismas reglas de selección de versión que la ejecución implícita de scripts, pero se puede seleccionar una versión más específica pasando los argumentos apropiados (como `-3` para solicitar Python 3 cuando Python 2 también está instalado, o `-2.6` para solicitar específicamente una versión anterior de Python cuando se instala una versión más reciente).

Además del lanzador, el instalador de Windows ahora incluye una opción para agregar el Python recién instalado al PATH del sistema. (Contribución de Brian Curtin en bpo-3561 [https://bugs.python.org/issue?@action=redirect&bpo=3561].)

Ver también

PEP 397 [https://peps.python.org/pep-0397/] - Python Launcher for Windows

PEP escrito por Mark Hammond y Martin v. Löwis; implementado por Vinay Sajip.

Documentación del lanzador: Python install manager

Modificación de la ruta del instalador: Python install manager

PEP 3151: Reelaborando de la jerarquía de excepciones de IO y OS

La jerarquía de excepciones generada por errores del sistema operativo ahora está simplificada y es más detallada.

You don’t have to worry anymore about choosing the appropriate exception type between **`OSError`**, **`IOError`**, **`EnvironmentError`**, **`WindowsError`**, **`mmap.error`**,

`socket.error` or **`select.error`**. All these exception types are now only one: **`OSError`**. The other names are kept as aliases for compatibility reasons.

Además, ahora es más fácil detectar una condición de error específica. En lugar de inspeccionar el atributo `errno` (o `args[0]`) para una constante particular del módulo **`errno`**, puede seleccionar la subclase **`OSError`** adecuada. Las subclases disponibles son las siguientes:

- **`BlockingIOError`**
- **`ChildProcessError`**
- **`ConnectionError`**
- **`FileExistsError`**
- **`FileNotFoundError`**
- **`InterruptedError`**
- **`IsADirectoryError`**
- **`NotADirectoryError`**
- **`PermissionError`**
- **`ProcessLookupError`**
- **`TimeoutError`**

Y el mismo **`ConnectionError`** tiene subclases más detalladas:

- **`BrokenPipeError`**
- **`ConnectionAbortedError`**
- **`ConnectionRefusedError`**
- **`ConnectionResetError`**

Gracias a las nuevas excepciones, los usos comunes de **`errno`** ahora se pueden evitar. Por ejemplo, el siguiente código escrito para Python 3.2:

```
from errno import ENOENT, EACCES, EPERM

try:
with open("document.txt") as f:
content = f.read()
except IOError as err:
if err.errno == ENOENT:
print("Falta el archivo document.txt")
```

```
elif err.errno in (EACCES, EPERM):
print("No tiene permiso para leer document.txt")
else:
raise
```

ahora se pueden escribir sin la importación de **`errno`** y sin la inspección manual de atributos de excepción:

```
Intente:
con open("document.txt") como f:
content = f.read()
except FileNotFoundError:
print("Falta el archivo document.txt")
except PermissionError:
print("No tiene permiso para leer document.txt")
```

Ver también

PEP 3151 [https://peps.python.org/pep-3151/] - Re elaboración de la jerarquía de excepciones de IO y OS
PEP escrito e implementado por Antoine Pitrou

PEP 380: Sintaxis para delegar en un subgenerador

PEP 380 agrega la expresión `yield from`, permitiendo que un generator delegue parte de sus operaciones a otro generador. Esto permite tomar en consideración una sección de código que contenga la palabra **`yield`** y ubicarla en otro generador. Además, el subgenerador puede regresar con un valor y el valor se pone a disposición del generador que delega.

Aunque está diseñada principalmente para su uso en la delegación a un subgenerador, la expresión `yield from` en realidad permite la delegación a subiteradores arbitrarios.

Para iteradores simples, `yield from iterable` es esencialmente una forma abreviada de `for item in iterable: yield` `item`:

```
>>> def g(x):
... rendimiento de rango(x, 0, -1)
... rendimiento de rango(x)
...
>>> lista(g(5))
[5, 4, 3, 2, 1, 0, 1, 2, 3, 4]
```

Sin embargo, a diferencia de un ciclo ordinario, `yield from` permite que los subgeneradores reciban valores enviados y lanzados directamente desde el ámbito de la llamada y retornen un valor final al generador externo:

```
>>> def accumulate():
...     tally = 0
...     while 1:
...         next = yield
...         if next is None:
...             return tally
...         tally += next
...
>>> def gather_tallies(tallies):
...     while 1:
...         tally = yield from accumulate()
...         tallies.append(tally)
...
>>> tallies = []
>>> acc = gather_tallies(tallies)
>>> next(acc)  # Asegura que acumulador esté listo y ace
>>> for i in range(4):
...     acc.send(i)
...
>>> acc.send(None)  # Finaliza el primer recuento
>>> for i in range(5):
...     acc.send(i)
...
>>> acc.send(None)  # Finaliza el segundo recuento
>>> tallies
```

```
[6, 10]
```

Lo más importante de éste cambio es permitir que incluso los generadores que están diseñados para ser usados con los métodos `send` y `throw` sean separados en múltiples subgeneradores tan fácilmente como las funciones grandes se pueden dividir en varias subfunciones.

Ver también

PEP 380 [https://peps.python.org/pep-0380/] - Sintaxis para delegar a un subgenerador

PEP escrito por Greg Ewing; implementación por Greg Ewing, integrado en 3.3 por Renaud Blanch, Ryan Kelly y Nick Coghlan; documentación de Zbigniew Jędrzejewski-Szmek y Nick Coghlan

PEP 409: Suprimir el contexto de excepción

PEP 409 introduce una nueva sintaxis que permite deshabilitar la visualización del contexto de excepción encadenado. Esto permite mensajes de error más limpios en aplicaciones que convierten entre tipos de excepción:

```
>>> class D:
...     def __init__(self, extra):
...         self._extra_attributes = extra
...     def __getattr__(self, attr):
...         try:
...             return self._extra_attributes[attr]
...         except KeyError:
...             raise AttributeError(attr) from None
...
>>> D({}).x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 8, in __getattr__
```

```
AttributeError: x
```

Sin el sufijo `from None` para suprimir la causa, la excepción original se mostraría de forma predeterminada:

```
>>> class C:
...     def __init__(self, extra):
...         self._extra_attributes = extra
...     def __getattr__(self, attr):
...         try:
...             return self._extra_attributes[attr]
...         except KeyError:
...             raise AttributeError(attr)
...
>>> C({}).x
Traceback (most recent call last):
  File "<stdin>", line 6, in __getattr__
KeyError: 'x'

During handling of the above exception, another exceptio

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 8, in __getattr__
AttributeError: x
```

No se pierde ninguna capacidad de depuración, ya que el contexto de la excepción original permanece disponible en caso de ser necesario (por ejemplo, si una biblioteca suprime detalles importantes y valiosos de forma incorrecta)

```
>>> try:
...     D({}).x
... except AttributeError as exc:
...     print(repr(exc.__context__))
...
KeyError('x',)
```

Ver también

PEP 409 [https://peps.python.org/pep-0409/] - Supresión del contexto de excepción

PEP escrito por Ethan Furman; implementado por Ethan Furman y Nick Coghlan.

PEP 414: Literales *Unicode* explícitos

Para facilitar la transición de Python 2 para las aplicaciones de Python que reconocen *Unicode* y que hacen un uso intensivo de literales Unicode, Python 3.3 una vez más admite el prefijo «`u`» para cadenas literales. Este prefijo no tiene importancia semántica en Python 3, se proporciona únicamente para reducir el número de cambios puramente mecánicos en la migración a Python 3, lo que facilita a los desarrolladores centrarse en los cambios semánticos más significativos (como la separación predeterminada más estricta de binary y datos de texto).

Ver también

PEP 414 [https://peps.python.org/pep-0414/] - Literales *Unicode* Explícitos

PEP escrito por Armin Ronacher.

PEP 3155: Nombres calificados para clases y funciones

Las funciones y los objetos de clase tienen un nuevo atributo `__qualname__` que representa la «ruta» desde el nivel superior del módulo hasta su definición. Para las funciones y clases globales, esto es lo mismo que `__name__`. Para otras funciones y clases, proporciona mejor información sobre dónde se definieron realmente y cómo se puede acceder a ellas desde el ámbito global.

Ejemplo con métodos (no vinculados):

```
>>> class C:
...     def meth(self):
```

```
...         pass
...
>>> C.meth.__name__
'meth'
>>> C.meth.__qualname__
'C.meth'
```

Ejemplo con clases anidadas:

```
>>> class C:
...     class D:
...         def meth(self):
...             pass
...
>>> C.D.__name__
'D'
>>> C.D.__qualname__
'C.D'
>>> C.D.meth.__name__
'meth'
>>> C.D.meth.__qualname__
'C.D.meth'
```

Ejemplo con funciones anidadas:

```
>>> def outer():
...     def inner():
...         pass
...     return inner
...
>>> outer().__name__
'inner'
>>> outer().__qualname__
'outer.<locals>.inner'
```

La representación en cadena de estos objetos, también se ha modificado para incluir la información nueva y más precisa:

```
>>> str(C.D)
"<class '__main__.C.D'>"
```

```
>>> str(C.D.meth)
'<function C.D.meth at 0x7f46b9fe31e0>'
```

Ver también

PEP 3155 [https://peps.python.org/pep-3155/] - Nombres calificados para clases y funciones
PEP Escrito e implementado por Antoine Pitrou.

PEP 412: Diccionario de intercambio de claves

Los diccionarios usados para almacenar atributos de objetos, ahora pueden compartir parte de su almacenamiento interno entre sí (a saber, la parte que almacena las claves y sus respectivos hashes). Esto reduce el consumo de memoria de programas creando varias instancias de tipos no integrados.

Ver también

PEP 412 [https://peps.python.org/pep-0412/] -Diccionario de intercambio de claves
PEP escrito e implementado por Mark Shannon.

PEP 362: Objeto de firma de función

Una nueva función `inspect.signature()` hace que la introspección de las llamadas de Python sea fácil y directa. Se admite una amplia gama de invocables: funciones python, decoradas o no, clases y objetos `functools.partial()`. Las Nuevas clases `inspect.Signature`, `inspect.Parameter` y `inspect.BoundArguments` contienen información sobre las firmas de llamadas, tales como anotaciones, valores por defecto, tipos de parámetros, y los argumentos vinculados, lo cual simplifica considerablemente la escritura de decoradores y de cualquier código que valide o modifique las firmas de llamadas o argumentos.

Ver también

PEP 362 [https://peps.python.org/pep-0362/]: - Objeto de firmas de función

PEP escrito por Brett Cannon, Yury Selivanov, Larry Hastings, Jiwon Seo; implementado por Yury Selivanov.

PEP 421: Agregar sys.implementation

Un nuevo atributo en el módulo **`sys`** expone detalles específicos de la implementación del intérprete que se ésta ejecutando en el momento actual. El conjunto inicial de atributos en **`sys.implementation`** son `name`, `version`, `hexversion`, y `cache_tag`.

La intención de `sys.implementation` es consolidar dentro de un espacio de nombres, la implementación específica de datos usado por la biblioteca estándar. Esto permite que diferentes implementaciones de python puedan compartir código de la biblioteca estándar de manera más fácil. En su estado inicial, `sys.implementation` contiene solo una pequeña porción de los datos específicos de la implementación. Con el tiempo esta proporción cambiará para hacer que la biblioteca estándar sea más portable..

Un ejemplo de mejor portabilidad de la biblioteca estándar es `cache_tag`. A partir de Python 3.3, `sys.implementation.cache_tag` es utilizado por **`importlib`** para admitir el cumplimiento PEP 3147 [https://peps.python.org/pep-3147/]. Cualquier implementación de Python que utilice `importlib` para su sistema de importación integrado puede usar `cache_tag` para controlar el comportamiento de almacenamiento en caché de los módulos.

SimpleNamespace

La implementación de `sys.implementation` también trae un nuevo tipo a Python: La clase **`types.SimpleNamespace`**. En comparación con un nombre de espacio basado en mapas, como `dict`, `SimpleNamespace` es basado en atributos, como `object`. Sin embargo a diferencia de `object`, las instancias de `SimpleNamespace` son editables. Esto significa que puedes agregar, eliminar, y modificar el nombre de espacio a través del acceso normal a los atributos.

Ver también

PEP 421 [https://peps.python.org/pep-0421/] - Agregar sys.implementation

PEP escrito e implementado por Mark Shannon.

Usar importlib como implementación de Import

bpo-2377 [https://bugs.python.org/issue?@action=redirect&bpo=2377] - Replace_import_w/ importlib._import_ bpo-13959 [https://bugs.python.org/issue?@action=redirect&bpo=13959] - Re-implementar partes de **`imp`** en Python puro bpo-14605 [https://bugs.python.org/issue?@action=redirect&bpo=14605] - Hacer que la maquinaria de importación sea explícita bpo-14646 [https://bugs.python.org/issue?@action=redirect&bpo=14646] - Requerir que los cargadores establezcan _loader_ y _package_

La función `__import__()` ahora es potenciada por la función `importlib.__import__()`. Este trabajo conduce a completar la «fase 2» de PEP 302 [https://peps.python.org/pep-0302/]. Hay múltiples beneficios en éste cambio. En primer lugar, ha permitido que la mayoría de la maquinaria que potencia las importaciones sea expuesta, en vez de estar implícitas y ocultas en el código de C. También proporciona una única implementación para todas las máquinas virtuales de python que admitan el uso de Python 3.3 lo que ayuda a terminar con cualquier desviación específica de VM en la semántica de importación. Y finalmente facilita el mantenimiento de la importación, permitiendo que ocurra un crecimiento futuro.

Para el usuario común, no debería haber cambios visibles en la semántica. Para aquellos cuyo código realmente manipula la importación o las llamadas de importación de forma programática, los cambios de código que posiblemente sean necesarios, se tratan en la sección Porting Python code de éste documento.

Nuevas APIs

Uno de los grandes beneficios de éste trabajo es la exposición de lo que implica hacer que funcione la sentencia de importación. Esto significa que los varios importadores que una vez fueron implícitos, ahora son expuestos completamente como parte del paquete `importlib`.

Las clases base abstractas definidas en `importlib.abc` se han expandido para delimitar adecuadamente entre buscadores de rutas meta y buscadores de entrada de rutas al introducir la clase `importlib.abc.MetaPathFinder` y `importlib.abc.PathEntryFinder` respectivamente. El antiguo ABC de la clase `importlib.abc.Finder` ahora solo se proporciona para compatibilidad con versiones anteriores y no exige ningún requisito de método.

En términos de buscadores, `importlib.machinery.FileFinder` expone el mecanismo utilizado para buscar archivos fuente y de código de *bytes* de un módulo. Anteriormente, ésta clase era un miembro implícito de `sys.path_hooks`.

Para los cargadores, la nueva clase base abstracta `importlib.abc.FileLoader` ayuda a escribir un cargador que usa el sistema de archivos como mecanismo de almacenamiento para el código de un módulo. El cargador de archivos fuente (`importlib.machinery.SourceFileLoader`), archivos de código byte sin fuente (`importlib.machinery.SourcelessFileLoader`) y módulos de extensión (`importlib.machinery.ExtensionFileLoader`) ahora están disponibles para su uso directo.

`ImportError` ahora tiene los atributos `name` y `path` que se establecen cuando hay datos relevantes para proporcionar. También, el mensaje de importaciones fallidas ahora proporcionará el nombre completo del módulo en lugar de solo el final del nombre del mismo.

La función `importlib.invalidate_caches()` ahora llamará al método con el mismo nombre en todos los buscadores almacenados en `sys.path_importer_cache` para ayudar a limpiar cualquier estado almacenado según sea necesario.

Cambios visibles

Para conocer los posibles cambios necesarios en el código, consulte la sección Porting Python code .

Más allá de la extensión de lo que ahora expone `importlib`, hay otros cambios visibles para importar. El más grande es que `sys.meta_path` y `sys.path_hooks` ahora almacenan todos los buscadores de rutas meta y los ganchos de entrada de rutas usados por la importación. Anteriormente, los buscadores estaban implícitos y ocultos dentro del código C de importación en lugar de estar expuestos directamente. Esto significa que ahora uno puede eliminar o cambiar fácilmente el orden de los diversos buscadores para satisfacer sus necesidades.

Otro cambio es que todos los módulos tienen un atributo `__loader__` que almacena el cargador usado para crear el módulo. La **PEP 302** [https://peps.python.org/pep-0302/] se ha actualizado para hacer que este atributo sea obligatorio para implementar los cargadores, de manera que en el futuro, una vez que los cargadores de terceros hayan sido actualizados, las personas puedan confiar en la existencia de éste atributo. Sin embargo, hasta entonces la importación está configurando la post carga del módulo.

También se espera que ahora los cargadores establezcan el atributo `__package__` de **PEP 366** [https://peps.python.org/pep-0366/]. Una vez más, la importación por sí misma ya está configurando ésto en todos los cargadores del módulo `importlib` y la importación misma está configurando la post carga del atributo.

Ahora se incluye el atributo `None` dentro de `sys.path_importer_cache` cuando no se encuentre ningún buscador en `sys.path_hooks`. Dado que `imp.NullImporter` no está directamente expuesto en `sys.path_hooks`, podría no haber una certeza de que esté disponible para su uso como un valor que represente que no se ha encontrado ningún buscador.

Todos los demás cambios se relacionan con los cambios de semántica que deben tomarse en cuenta al actualizar el código a python 3.3, por lo tanto, deben leerse en la sección Portar código Python de este documento.

(Implementación por Brett Cannon)

Otros cambios de idioma

Algunos cambios más pequeños realizados en el lenguaje Python principal son:

- Se agregó compatibilidad con alias de nombres Unicode y secuencias con nombre. Tanto **`unicodedata.lookup()`** como `'\N{...}'` ahora resuelven alias de nombres, y **`unicodedata.lookup()`** también resuelve secuencias con nombre.

  (Contribución de Ezio Melotti en bpo-12753 [https://bugs.python.org/issue?@action=redirect&bpo=12753].)

- Base de datos *Unicode* actualizada a la versión 6.1.0 de UCD

- Las comparaciones de igualdad en los objetos **`range()`** ahora retornan un resultado que refleja la igualdad de las secuencias subyacentes generadas por esos objetos de rango. (bpo-13201 [https://bugs.python.org/issue?@action=redirect&bpo=13201])

- Los métodos `count()`, `find()`, `rfind()`, `index()` y `rindex()` de los objetos **`bytes`** y **`bytearray`** ahora aceptan un entero entre 0 y 255 como su primer argumento.

  (Contribución por Petri Lehtinen en bpo-12170 [https://bugs.python.org/issue?@action=redirect&bpo=12170].)

- Los métodos `rjust()`, `ljust()`, y `center()` de **`bytes`** y **`bytearray`** ahora aceptan un **`bytearray`** para el argumento `fill`. (Contribución de Petri Lehtinen en bpo-12380 [https://bugs.python.org/issue?@action=redirect&bpo=12380].)

- New methods have been added to **`list`** and **`bytearray`**: `copy()` and `clear()` (bpo-10516 [https://bugs.python.org/issue?@action=redirect&bpo=10516]). Consequently, **`MutableSequence`** now also defines a **`clear()`** method (bpo-11388 [https://bugs.python.org/issue?@action=redirect&bpo=11388]).
- Los literales de *bytes* sin formato ahora se pueden escribir `rb"..."` así como `br"..."`.

  (Contribución de Antoine Pitrou en bpo-13748 [https://bugs.python.org/issue?@action=redirect&bpo=13748].)
- El método **`dict.setdefault()`** ahora solo hace una sola búsqueda de la clave dada, haciéndola atómica cuando se usa con tipos incorporados.

  (Contribución de Filip Gruszczyński en bpo-13521 [https://bugs.python.org/issue?@action=redirect&bpo=13521].)
- Los mensajes de error producidos cuando la llamada de una función no coincide con la firma han sido mejorados significativamente.

  (Contribución de Benjamin Peterson.)

Un bloqueo de importación más detallado

Las versiones anteriores de CPython siempre se han basado en un bloqueo de importación global. Ésto conduce a problemas inesperados tales como bloqueos cuando la importación un módulo debería disparar la ejecución de código en un hilo diferente como un efecto secundario. A veces se empleaban soluciones torpes, como la función de API C **`PyImport_ImportModuleNoBlock()`** .

En Python 3.3, la importación de un módulo toma un bloqueo por módulo. Esto serializa correctamente la importación de un módulo determinado a partir de múltiples subprocesos (impidiendo la exposición de módulos inicializados incompletamente), mientras elimina las molestias antes mencionadas.

(Contribución de Antoine Pitrou en bpo-9260 [https://bugs.python.org/issue?@action=redirect&bpo=9260].)

Funciones y tipos incorporados

- `open()` obtiene un nuevo parámetro *opener*: el descriptor de archivo subyacente para el objeto de archivo se obtiene llamando a *opener* con (*file*, *flags*). Se puede utilizar para utilizar banderas personalizadas como `os.O_CLOEXEC`, por ejemplo. Se ha añadido el modo `'x'` : abierto para la creación exclusiva, fallando si el archivo ya existe.
- `print()`: agregó el argumento de palabra clave *flush*. Si el argumento de palabra clave *flush* es verdadero, la secuencia se vacía por la fuerza.
- `hash()`: la aleatorización de hash está habilitada de forma predeterminada, consulte `object.__hash__()` y `PYTHONHASHSEED`.
- El tipo `str` obtiene un nuevo método `casefold()`: retorna una copia de la cadena con mayúsculas y minúsculas, las cadenas con mayúsculas y minúsculas se pueden utilizar para la coincidencia sin mayúsculas. Por ejemplo, `'ß'.casefold()` retorna `'ss'`.
- La documentación de secuencia se ha reescrito sustancialmente para explicar mejor la distinción de secuencia binaria/texto y para proporcionar secciones de documentación específicas para los tipos de secuencia individuales integrados(bpo-4966 [https://bugs.python.org/issue?@action=redirect&bpo=4966]).

Nuevos módulos

faulthandler

This new debug module `faulthandler` contains functions to dump Python tracebacks explicitly, on a fault (a crash like a segmentation fault), after a timeout, or on a user signal. Call **`faulthandler.enable()`** to install fault handlers for the **`SIGSEGV`**, **`SIGFPE`**, **`SIGABRT`**, **`SIGBUS`**, and **`SIGILL`** signals. You can also enable them at startup by setting the **`PYTHONFAULTHANDLER`** environment variable or by using **`-X`** `faulthandler` command line option.

Ejemplo de una falla de segmentación en Linux:

```
$ python -q -X faulthandler
>>> import ctypes
>>> ctypes.string_at(0)
Fatal Python error: Segmentation fault

Current thread 0x00007fb899f39700:
  File "/home/python/cpython/Lib/ctypes/__init__.py", li
  File "<stdin>", line 1 in <module>
Segmentation fault
```

ipaddress

El nuevo módulo **`ipaddress`** proporciona herramientas para crear y manipular objetos que representen direcciones, redes e interfaces IPv4 e IPv6 (es decir, alguna dirección IP asociada con una subred IP específica)

(Contribución de Google y Peter Moody en **PEP 3144** [https://peps.python.org/pep-3144/])

lzma

El módulo **`lzma`** recién agregado proporciona compresión y descompresión de datos utilizando el algoritmo LZMA, incluida la compatibilidad con los formatos de archivo `.xz` y `.lzma`.

(Contribución de Nadeem Vawda y Per Øyvind Karlsen en bpo-6715 [https://bugs.python.org/issue?@action=redirect&bpo=6715].)

Módulos mejorados

abc

Improved support for abstract base classes containing descriptors composed with abstract methods. The recommended approach to declaring abstract descriptors is now to provide **`__isabstractmethod__`** as a dynamically updated property. The built-in descriptors have been updated accordingly.

- La clase `abc.abstractproperty` ha quedado obsoleta, en su lugar use la clase `property` con `abc.abstractmethod()`.
- La clase `abc.abstractclassmethod` ha quedado obsoleta. En su lugar use el método `classmethod` with `abc.abstractmethod()`.
- `abc.abstractstaticmethod` ha quedado obsoleta, use `staticmethod` with `abc.abstractmethod()`.

(Contribución de Darren Dale en bpo-11610 [https://bugs.python.org/issue?@action=redirect&bpo=11610].)

El método `abc.ABCMeta.register()` ahora retorna las subclases registradas, lo cual significa que ahora éste puede ser usado como un decorador de clases (bpo-10868 [https://bugs.python.org/issue?@action=redirect&bpo=10868]).

array

El módulo `array` admite el tipo long long utilizando los códigos de tipo `q` y `Q`.

(Contribución de Oren Tirosh y Hirokazu Yamamoto en bpo-1172711 [https://bugs.python.org/issue?@action=redirect&bpo=1172711].)

base64

Las cadenas *Unicode* de solo ASCII ahora son aceptadas por las funciones de decodificación de la interfaz moderna `base64`. Por ejemplo, `base64.b64decode ('YWJj')` retorna `b'abc'`. (Contribución por Catalin Iacob en: número: *13641*.)

binascii

Además de los objetos binarios que normalmente aceptan, las funciones `a2b_` ahora también aceptan cadenas de sólo ASCII como dato de entrada. (Contribución por Antoine Petrou en bpo-13637 [https://bugs.python.org/issue?@action=redirect&bpo=13637].)

bz2

El módulo `bz2` ha sido reescrito desde cero. Durante el proceso, han sido agregadas varias características:

- Nueva función `bz2.open()`: abre un archivo comprimido por bzip2, en modo binario o en modo de texto.
- `bz2.BZ2File` ahora puede leer y escribir en objetos de tipo archivo arbitrarios, mediante el argumento *fileobj* de su constructor.

  (Contribución de Nadeem Vawda en bpo-5863 [https://bugs.python.org/issue?@action=redirect&bpo=5863].)
- La clase `bz2.BZ2File` y la función `bz2.decompress()` ahora descomprimen entradas de flujo múltiple (como las producidas por la herramienta **pbzip2**). La clase `bz2.BZ2File` ahora puede ser usada también para crear este tipo de archivos, usando el modo `'a'` (agregar).

  (Contribución de Nir Aides en bpo-1625 [https://bugs.python.org/issue?@action=redirect&bpo=1625].)
- `bz2.BZ2File` now implements all of the `io.BufferedIOBase` API, except for the `detach()` and `truncate()` methods.

códecs

El códec `mbcs` ha sido reescrito para manejar correctamente los controladores de error `reemplazar` e `ignorar` en todas las versiones Windows. El códec `mbcs` ahora admite todos los controladores de error, en lugar de solo `replace` para codificar e `ignore` para decodificar.

Ha sido agregado un nuevo códec únicamente para windows: `cp65001` (bpo-13216 [https://bugs.python.org/issue?@action=redirect&bpo=13216]). Es la página de códigos de Windows con el código 65001 (Windows UTF-8, `CP_UTF8`). Por ejemplo, es usado por `sys.stdout` si la página de códigos de salida de la consola se establece en cp65001 (por ejemplo, usando el comando `chcp 65001`).

Los decodificadores CJK multibyte ahora se re-sincronizan más rápido. Solo ignoran el primer byte de una secuencia de *bytes* no válida. Por ejemplo, `b'\xff\n'.decode('gb2312', 'replace')` ahora retorna un `\n` después del carácter de reemplazo.

(bpo-12016 [https://bugs.python.org/issue?@action=redirect&bpo=12016])

Los codificadores de códec incrementales CJK ya no se restablecen en cada llamada a sus métodos encode(). Por ejemplo:

```
>>> import codecs
>>> encoder = codecs.getincrementalencoder('hz')('strict
>>> b''.join(encoder.encode(x) for x in '\u52ff\u65bd\u6
b'~{NpJ)l6HK!#~} Bye.'
```

Este ejemplo resulta `b'~{Np~}~{J)~}~{l6~}~{HK~}~{!#~} Bye.'` con versiones anteriores de Python.

(bpo-12100 [https://bugs.python.org/issue?@action=redirect&bpo=12100])

El códec `unicode_internal` ha quedado obsoleto.

colecciones

Adición de una nueva **`ChainMap`** para permitir el tratamiento de un número de asignaciones como una sola unidad. (Escrito por Raymond Hettinger para bpo-11089 [https://bugs.python.org/issue?@action=redirect&bpo=11089], publicado en bpo-11297 [https://bugs.python.org/issue?@action=redirect&bpo=11297].)

Las clases base abstractas se han movido en un nuevo módulo `collections.abc`, para diferenciar mejor entre las clases de colecciones abstractas y concretas. Los alias para ABC siguen presentes en el módulo `collections` para preservar las importaciones existentes. (bpo-11085 [https://bugs.python.org/issue?@action=redirect&bpo=11085])

La clase `Counter` ahora admite los operadores "" + "" y ""-"" unarios, así como los operadores in situ `+=`, `-=`, `|=`, and `&=`. (Contribución de Raymond Hettinger en bpo-13121 [https://bugs.python.org/issue?@action=redirect&bpo=13121].)

contextlib

`ExitStack` ahora proporciona una base sólida para la manipulación programática de los administradores de contexto y una funcionalidad de limpieza similar. A diferencia de la API anterior `contextlib.nested` (que estaba en desuso y se eliminó), la nueva API está diseñada para funcionar correctamente independientemente de si los administradores de contexto adquieren sus recursos en su método `__init__` (por ejemplo, objetos de archivo) o en su método `__enter__` (por ejemplo, objetos de sincronización del módulo `threading`).

(bpo-13585 [https://bugs.python.org/issue?@action=redirect&bpo=13585])

crypt

Adición del formato de sal y criptografía modular (método hash) y la función **`mksalt()`** al módulo **`crypt`**.

(bpo-10924 [https://bugs.python.org/issue?@action=redirect&bpo=10924])

curses

- If the `curses` module is linked to the ncursesw library, use Unicode functions when Unicode strings or characters are passed (e.g. **`waddwstr()`**), and bytes functions otherwise (e.g. **`waddstr()`**).
- Usa la codificación local en vez de `utf-8` para codificar

cadenas *Unicode*.

- La clase `curses.window` tiene un nuevo atributo `curses.window.encoding`.
- La clase `curses.window` tiene un nuevo método `get_wch()` para obtener un caracter ancho
- El módulo `curses` tiene una nueva función `unget_wch()` para enviar un caracter ancho de manera que el próximo `get_wch()` lo retorne

(Contribución por Iñigo Serna en bpo-6755 [https://bugs.python.org/issue?@action=redirect&bpo=6755].)

datetime

- Las comparaciones de igualdad entre instancias conscientes e ingenuas `datetime` ahora retornan `False` en lugar de generar `TypeError` (bpo-15006 [https://bugs.python.org/issue?@action=redirect&bpo=15006]).
- Nuevo método `datetime.datetime.timestamp()`: Retorna la marca de tiempo POSIX correspondiente a la instancia `datetime`.
- El método `datetime.datetime.strftime()` admite el formato de años anteriores a 1000.
- Ahora se puede llamar al método `datetime.datetime.astimezone()` sin argumentos para convertir la instancia datetime en la zona horaria del sistema.

decimal

bpo-7652 [https://bugs.python.org/issue?@action=redirect&bpo=7652] - integrar la aritmética decimal nativa rápida.

Módulo C y libmpdec escritos por Stefan Krah.

La nueva versión C del módulo decimal integra la biblioteca libmpdec de alta velocidad para aritmética de punto flotante decimal redondeada correctamente con precisión arbitraria. libmpdec cumple con la Especificación Aritmética Decimal General de IBM.

Las mejoras de rendimiento varían de 10x para aplicaciones de bases de datos a 100x para aplicaciones con uso intensivo de números. Estas cifras son ganancias esperadas para las precisiones estándar utilizadas en aritmética de punto flotante decimal. Dado que la precisión es configurable por el usuario, las cifras exactas pueden variar. Por ejemplo, en la aritmética de números enteros grandes, las diferencias pueden ser significativamente mayores.

La siguiente tabla está pensada como una ilustración. Los puntos de referencia están disponibles en https://www.bytereef.org/mpdecimal/quickstart.html.

	decimal	_decimal	speedup
pi	42.02s	0.345s	120x
telco	172.19s	5.68s	30x
psycopg	3.57s	0.29s	12x

Características

- La señal `FloatOperation` habilita opcionalmente las semánticas estrictas para mezclar decimales y flotantes.
- Si python se compila sin subprocesos, la versión C deshabilita automáticamente la costosa maquinaria de contexto local de subprocesos. En éste caso, la variable `HAVE_THREADS` se establece en `False`.

Cambios en la API

- El módulo C tiene los siguientes límites de contexto, dependiendo de la arquitectura de la máquina:

	32-bit	64-bit
`MAX_PREC`	`425000000`	`999999999999999999`
`MAX_EMAX`	`425000000`	`999999999999999999`
`MIN_EMIN`	`-425000000`	`-999999999999999999`

- In the context templates (`DefaultContext`, `BasicContext` and `ExtendedContext`) the magnitude of `Emax` and `Emin` has changed to `999999`.

- El constructor `Decimal` en decimal.py no observa los límites de contexto y convierte valores con exponentes arbitrarios o precisión exacta. Dado que la versión C tiene límites internos, se usa el siguiente esquema: Si es posible, los valores son convertidos a valores exactos, de lo contrario se genera `InvalidOperation` y el resultado es NaN. En éste último caso siempre es posible usar `create_decimal()` para obtener un valor redondeado o inexacto.

- La función de potencia en decimal.py siempre se redondea correctamente. En la versión C, se define en términos de las funciones `exp()` y `ln()` correctamente redondeadas, pero el resultado final es solo «casi siempre correctamente redondeado».
- En la versión C, el diccionario de contexto que contiene las señales, es una `MutableMapping`. Por razones de velocidad, `flags` y `traps` siempre se refiere a la misma `MutableMapping` que el contexto con el que se inicializó. Si se asigna un nuevo diccionario de señales, `flags` y `traps` son actualizados con los nuevos valores, pero no hacen referencia al diccionario RHS.
- Decapar a `Context` produce una salida diferente para tener un formato de intercambio común para las versiones de Python y C.
- El orden de los argumentos en el constructor de la clase `Context`, ha sido cambiado para que coincida con el orden mostrado por la función `repr()`.
- El parámetro `watchexp` en el método `quantize()` ha quedado obsoleto.

email

Marco de políticas

El paquete de correo electrónico ahora tiene un marco `policy`. Un `Policy` es un objeto con varios métodos y propiedades que controlan el comportamiento del paquete de correo electrónico. La política principal de Python 3.3 es la política **`Compat32`**, que proporciona compatibilidad con versiones anteriores del paquete de correo electrónico en Python 3.2. Se puede especificar una `policy` cuando un mensaje de correo electrónico es analizado por un **`parser`**, o cuando se crea un objeto **`Message`**, o cuando se serializa un correo electrónico mediante un **`generator`**. A menos que se invalide, una política pasada a un `parser` es heredada por todos los objetos `Message` y los subobjetos creados por el `parser`. De forma predeterminada, un `generator` utilizará la política del objeto `Message` que está serializando. La directiva predeterminada es **`compat32`**.

El conjunto mínimo de controles implementado por todos los objetos `policy` son:

max_line_length
La longitud máxima, excluyendo el(los) caracter(es) *linesep*, que las líneas individuales pueden tener cuando un `Message` se serializa es por defecto de 78.

linesep
El caracter usado para separar las líneas individuales cuando se serializa un `Message` es por defecto `\n`.

cte_type
7bit o `8bit`. `8bit` sólo se aplica a un `Bytes generator`, y significa que pueden utilizarse «no ASCII» cuando lo permita el protocolo (o cuando exista en la entrada original).

raise_on_defect
Hace que el “”analizador”” genere el error cuando se encuentran defectos en lugar de agregarlos a la lista “”Mensaje”” del objeto “”defectos””.

Una nueva instancia de directiva, con nueva configuración, se crea mediante el método **`clone()`** de objetos de política. `clone` toma cualquiera de los controles anteriores como argumentos de palabra clave. Cualquier control no especificado en la llamada conserva su valor predeterminado. Por lo tanto, puede crear una directiva que utilice caracteres *linesep* `\r\n` como este:

```
mypolicy = compat32.clone(linesep='\r\n')
```

Las directivas pueden ser usadas para facilitar la generación de mensajes en el formato requerido por la aplicación. En vez de tener que recordar especificar `linesep='\r\n'` en todos los lugares en donde llamas a `generator`, puedes especificarlo una sola vez cuando estableces la política usada por el `analizador` o el

`Mensaje`, el que tu programa utilice para crear los objetos `Mensaje`. Por otra parte, si necesitas generar mensajes en múltiples formas, aún puedes especificar los parámetros en la llamada al generador apropiado. O puedes tener instancias de política personalizadas para tus distintos casos, y pasarlos al crear el `generator`.

Política provisional con nueva API de encabezado

Aunque el marco de políticas vale la pena por sí solo, la principal motivación para presentarlo es permitir la creación de nuevas políticas que implementen nuevas características para el paquete de correo electrónico, de modo que mantenga la compatibilidad con versiones anteriores para aquellos quienes no usen las nuevas directivas. Dado que las nuevas políticas introducen una nueva API, estamos lanzándolas en Python 3.3 como una provisional policy. Pueden producirse cambios incompatibles con versiones anteriores (hasta la eliminación del código) si los desarrolladores principales lo consideran necesario.

Las nuevas directivas son instancias de **`EmailPolicy`**, y agregan los siguientes controles adicionales:

refold_source
Controla si los encabezados analizados por un **`parser`** son replegados por **`generator`**. Éste puede ser `none`, `long`, o `all`. El valor por defecto es `long`, lo que significa que los encabezados principales con una linea más larga que `max_line_length` se repliegan. `none` significa que no se repliega ninguna linea, y `all` significa que todas las lineas son replegadas.

header_factory
Un invocable que toma un `nombre` y un `valor` y produce un objeto de encabezado personalizado.

El `header_factory` es la clave a las nuevas características proporcionadas por las nuevas directivas. Cuando se usa una de las nuevas directivas, cualquier encabezado retornado por un objeto `Message` es un objeto producido por el `header_factory`, y en cualquier momento que se establece un encabezado en un `Message`, éste se convierte en un objeto producido por `header_factory`. Todos éstos objetos de encabezado tienen un atributo `name` igual al nombre del encabezado. Los encabezados de direcciones y fechas tienen atributos adicionales que te dan acceso a los datos analizados del encabezado. Ésto significa que ahora puedes hacer cosas como ésta:

```
>>> m = Message(policy=SMTP)
>>> m['To'] = 'Éric <foo@example.com>'
>>> m['to']
'Éric <foo@example.com>'
>>> m['to'].addresses
(Address(display_name='Éric', username='foo', domain='ex
>>> m['to'].addresses[0].username
'foo'
>>> m['to'].addresses[0].display_name
'Éric'
>>> m['Date'] = email.utils.localtime()
>>> m['Date'].datetime
datetime.datetime(2012, 5, 25, 21, 39, 24, 465484, tzinf
>>> m['Date']
'Fri, 25 May 2012 21:44:27 -0400'
>>> print(m)
To: =?utf-8?q?=C3=89ric?= <foo@example.com>
Date: Fri, 25 May 2012 21:44:27 -0400
```

Notarás que el nombre de visualización *Unicode* se codifica automáticamente como `utf-8` cuando se serializa el mensaje, pero que cuando se accede al encabezado directamente, obtienes la versión unicode. Ésto elimina cualquier necesidad de lidiar con las funciones **`email.header decode_header()`** or **`make_header()`**.

También puedes crear direcciones a partir de partes:

```
>>> m['cc'] = [Group('pals', [Address('Bob', 'bob', 'exa
...                           Address('Sally', 'sally',
...               Address('Bonzo', addr_spec='bonz@laugh.co
>>> print(m)
To: =?utf-8?q?=C3=89ric?= <foo@example.com>
Date: Fri, 25 May 2012 21:44:27 -0400
cc: pals: Bob <bob@example.com>, Sally <sally@example.co
```

La decodificación a *Unicode* se realiza automáticamente:

```
>>> m2 = message_from_string(str(m))
>>> m2['to']
'Éric <foo@example.com>'
```

Cuando analizas un mensaje, puedes usar los atributos `addresses` y `groups` de los objetos de encabezado, para acceder a los grupos y direcciones individuales:

```
>>> m2['cc'].addresses
(Address(display_name='Bob', username='bob', domain='exa
>>> m2['cc'].groups
(Group(display_name='pals', addresses=(Address(display_n
```

En resumen, si usas una de las nuevas directivas, la manipulación de encabezados funciona del modo que debería: Su aplicación funciona con cadenas *unicode*, y el paquete de correo electrónico codifica y decodifica de forma transparente el unicode desde y hacia las codificaciones de transferencia de contenido estándar RFC.

Otros cambios en la API

Nuevo **`BytesHeaderParser`**, añadido al módulo **`parser`** para complementar **`HeaderParser`** y completar la API Bytes.

Nuevas funciones de utilidad:

- **`format_datetime()`**: dado un **`datetime`**, produce una cadena formateada para su uso en un encabezado de correo electrónico.
- **`email.utils.parsedate_to_datetime()`**: dada una cadena de fecha de un encabezado de correo electrónico, conviértela en un **`datetime.datetime`**, o un ingenuo **`datetime.datetime`** si el desplazamiento es `-0000`.
- **`localtime()`**: Sin argumento, retorna la hora local actual como un reconocimiento **`datetime`** utilizando el local **`timezone`**. Teniendo en cuenta **`datetime`**, lo convierte en un **`datetime`** con el valor local **`timezone`**.

ftplib

- **`ftplib.FTP`** ahora acepta un argumento de palabra clave `source_address` para especificar el `(host, port)` que se usará como dirección de origen en la llamada de enlace al crear el socket saliente. (Contribución de Giampaolo Rodolà en bpo-8594 [https://bugs.python.org/issue?@action=redirect&bpo=8594].)
- La clase **`FTP_TLS`** ahora proporciona una nueva función **`ccc()`** para revertir el canal de control a texto sin formato. Esto puede ser útil para aprovechar los firewalls que saben cómo manejar NAT con FTP no seguro sin abrir puertos fijos. (Contribución de Giampaolo Rodolà en bpo-12139 [https://bugs.python.org/issue?@action=redirect&bpo=12139].)
- Añadido el método **`ftplib.FTP.mlsd()`** que proporciona un formato de listado de directorios analizable y pone en desuso **`ftplib.FTP.nlst()`** y **`ftplib.FTP.dir()`**. (Contribución de Giampaolo Rodolà en bpo-11072 [https://bugs.python.org/issue?@action=redirect&bpo=11072].)

functools

El decorador **`functools.lru_cache()`** ahora acepta un argumento de palabra clave `typed` (que por defecto tiene el valor `False` para asegurar que almacene en caché valores de diferentes tipos que se comparen igual en ranuras separadas de caché. (Contribución de Raymond Hettinger en bpo-13227 [https://bugs.python.org/issue?@action=redirect&bpo=13227].)

gc

Ahora es posible registrar devoluciones de llamadas invocadas por el recolector de basura antes y después de la recolección, utilizando la nueva lista **`callbacks`**.

hmac

Se ha agregado una nueva función **`compare_digest()`** para prevenir los ataques de canales adyacentes en resúmenes durante el análisis de tiempos. (Contribución de Nick Coghlan y Christian Heimes en bpo-15061 [https://bugs.python.org/issue?@action=redirect&bpo=15061].)

http

`http.server.BaseHTTPRequestHandler` ahora almacena en buffer los encabezados y los escribe todos a la vez cuando se llama a **`end_headers()`**. Se puede usar un nuevo método **`flush_headers()`** para administrar directamente cuándo se envían los encabezados acumulados. (Contribución por Andrew Schaaf en bpo-3709 [https://bugs.python.org/issue?@action=redirect&bpo=3709].)

`http.server` ahora produce una salida `HTML 4.01 strict` válida. (Contribución de Ezio Melotti in bpo-13295 [https://bugs.python.org/issue?@action=redirect&bpo=13295])

La clase **`http.client.HTTPResponse`** ahora tiene un método **`readinto()`**, lo que significa que puede ser usado como una clase **`io.RawIOBase`**. (Contribución de John Kuhn in bpo-13464 [https://bugs.python.org/issue?@action=redirect&bpo=13464])

html

`html.parser.HTMLParser` is now able to parse broken markup without raising errors, therefore the *strict* argument of the constructor and the **`HTMLParseError`** exception are now deprecated. The ability to parse broken markup is the result of a number of bug fixes that are also available on the latest bug fix releases of Python 2.7/3.2. (Contributed by Ezio Melotti in bpo-15114 [https://bugs.python.org/issue?@action=redirect&bpo=15114], and bpo-14538 [https://bugs.python.org/issue?@action=redirect&bpo=14538], bpo-13993 [https://bugs.python.org/issue?@action=redirect&bpo=13993], bpo-13960 [https://bugs.python.org/issue?@action=redirect&bpo=13960], bpo-13358 [https://bugs.python.org/issue?@action=redirect&bpo=13358], bpo-1745761 [https://bugs.python.org/issue?@action=redirect&bpo=1745761], bpo-755670 [https://bugs.python.org/issue?@action=redirect&bpo=755670], bpo-13357 [https://bugs.python.org/issue?@action=redirect&bpo=13357], bpo-12629 [https://bugs.python.org/issue?

@action=redirect&bpo=12629], bpo-1200313 [https://bugs.python.org/issue?@action=redirect&bpo=1200313], bpo-670664 [https://bugs.python.org/issue?@action=redirect&bpo=670664], bpo-13273 [https://bugs.python.org/issue?@action=redirect&bpo=13273], bpo-12888 [https://bugs.python.org/issue?@action=redirect&bpo=12888], bpo-7311 [https://bugs.python.org/issue?@action=redirect&bpo=7311].)

Se ha agregado un nuevo diccionario **`html5`** que asigna referencias de caracteres con nombre HTML5 a los caracteres *Unicode* equivalentes (por ejemplo, `html5['gt;'] == '>'`) ha sido agregado al módulo **`html.entities`**. El diccionario ahora también es utilizado por **`HTMLParser`**. (Contribución por Ezio Melotti en bpo-11113 [https://bugs.python.org/issue?@action=redirect&bpo=11113] y bpo-15156 [https://bugs.python.org/issue?@action=redirect&bpo=15156].)

imaplib

El constructor de **`IMAP4_SSL`** ahora acepta un parámetro SSLContext para controlar los parámetros del canal seguro.

(Contribución de Sijin Joseph en bpo-8808 [https://bugs.python.org/issue?@action=redirect&bpo=8808].)

inspect

Se ha añadido una nueva función **`getclosurevars()`**. Esta función notifica el enlace actual de todos los nombres a los que se hace referencia desde el cuerpo de la función y dónde se resolvieron esos nombres, lo que facilita la comprobación del estado interno correcto al probar el código que se basa en cierres con estado.

(Contribución de Meador Inge y Nick Coghlan en bpo-13062 [https://bugs.python.org/issue?@action=redirect&bpo=13062].)

Se ha añadido una nueva función **`getgeneratorlocals()`**. Esta función informa del enlace actual de variables locales en el marco de pila del generador, lo que facilita la comprobación del estado interno correcto al probar generadores.

(Contribución de Meador Inge en bpo-15153 [https://bugs.python.org/

issue?@action=redirect&bpo=15153].)

io

La función `open()` tiene un nuevo modo `'x'` que puede ser usado exclusivamente para crear un nuevo archivo, y lanzar un `FileExistsError` si el archivo ya existe. Este está basado en el modo C11 "x" para fopen().

(Contribución de David Townsend en bpo-12760 [https://bugs.python.org/issue?@action=redirect&bpo=12760].)

The constructor of the `TextIOWrapper` class has a new *write_through* optional argument. If *write_through* is `True`, calls to `write()` are guaranteed not to be buffered: any data written on the `TextIOWrapper` object is immediately handled to its underlying binary buffer.

itertools

`accumulate()` ahora toma un argumento opcional `func` para proporcionar una función binaria proporcionada por el usuario.

logging

La función `basicConfig()` ahora admite un argumento opcional `handlers` tomando un iterable de manejadores para agregarlo en el registrador raíz.

A class level attribute `append_nul` has been added to `SysLogHandler` to allow control of the appending of the `NUL` (`\000`) byte to syslog records, since for some daemons it is required while for others it is passed through to the log.

math

El módulo `math` tiene una nueva función, `log2()`, la cual retorna el logaritmo de *x* en base 2.

(Escrito por Mark Dickinson en bpo-11888 [https://bugs.python.org/

issue?@action=redirect&bpo=11888].)

mmap

El método `read()` ahora es más compatible con otros objetos similares a archivos: Si se omite el argumento o se especifica como `None`, retorna los bytes desde la posición actual hasta el final del mapeo. (Contribución de Petri Lehtinen en bpo-12021 [https://bugs.python.org/issue?@action=redirect&bpo=12021].)

multiprocesamiento

La nueva función `multiprocessing.connection.wait()` permite sondear varios objetos (como conexiones, sockets y tuberías) con un tiempo de espera. (Contribución de Richard Oudkerk en bpo-12328 [https://bugs.python.org/issue?@action=redirect&bpo=12328].)

`multiprocessing.connection.Connection` objects can now be transferred over multiprocessing connections. (Contributed by Richard Oudkerk in bpo-4892 [https://bugs.python.org/issue?@action=redirect&bpo=4892].)

`multiprocessing.Process` ahora acepta un argumento de palabra clave `daemon` para invalidar el comportamiento predeterminado de heredar el indicador `daemon` del proceso primario (bpo-6064 [https://bugs.python.org/issue?@action=redirect&bpo=6064]).

Nuevo atributo `multiprocessing.Process.sentinel` permite que un programa espere múltiples objetos `Process` a la vez utilizando las primitivas adecuadas del sistema operativo (por ejemplo, `select` en sistemas posix).

Los nuevos métodos `multiprocessing.pool.Pool.starmap()` y `starmap_async()` proporcionan `itertools.starmap()` equivalentes a las funciones existentes `multiprocessing.pool.Pool.map()` y `map_async()`. (Contribución de Hynek Schlawack en bpo-12708 [https://

bugs.python.org/issue?@action=redirect&bpo=12708].)

nntplib

La clase `nntplib.NNTP` ahora admite el protocolo de administración de contexto para consumir incondicionalmente las excepciones `socket.error` y cerrar la conexión NNTP cuando finalice:

```
>>> from nntplib import NNTP
>>> with NNTP('news.gmane.org') as n:
...     n.group('gmane.comp.python.committers')
...
('211 1755 1 1755 gmane.comp.python.committers', 1755, 1
```

(Contribución de Giampaolo Rodolà en bpo-9795 [https://bugs.python.org/issue?@action=redirect&bpo=9795].)

os

- El módulo `os` tiene una nueva función `pipe2()` que hace posible crear una tubería con banderas `O_CLOEXEC` o `O_NONBLOCK` establecida automáticamente. Esto es especialmente útil para evitar las condiciones de carrera en programas multi-hilos.

- El módulo `os` tiene una nueva función `sendfile()` que proporciona una forma eficiente de «copia cero» para copiar datos desde un descriptor de archivo (o socket) a otro. La frase «copia cero» se refiere al hecho de que la copia de los datos entre dos descriptores es realizada en su totalidad por el kernel, sin copiar los datos en los búferes del espacio de usuario. La función `sendfile()` puede ser usada para copiar datos de forma eficiente desde un archivo en el disco a un conector de red, por ejemplo para descargar un archivo.

  (Revisión enviada por Ross Lagerwall y Giampaolo Rodolà en bpo-10882 [https://bugs.python.org/issue?@action=redirect&bpo=10882].)

- Para evitar las condiciones de carrera, como los ataques symlink, y errores con archivos temporales y directorios, es preferible (y más rápido) manipular los descriptores de archivos en vez de los nombres de archivos. Python 3.3 mejora las funciones existentes e introduce nuevas funciones para trabajar con los descriptores de archivos (bpo-4761 [https://bugs.python.org/issue?@action=redirect&bpo=4761], bpo-10755 [https://bugs.python.org/issue?@action=redirect&bpo=10755] and bpo-14626 [https://bugs.python.org/issue?@action=redirect&bpo=14626]).

    - El módulo `os` tiene una nueva función `fwalk()` similar a `walk()` excepto que también incluye descriptores de archivos que se refieren a los directorios visitados. Esto es especialmente útil para prevenir las carreras de enlaces simbólicos.
    - The following functions get new optional *dir_fd* (paths relative to directory descriptors) and/or *follow_symlinks* (not following symlinks): `access()`, `chflags()`, `chmod()`, `chown()`, `link()`, `lstat()`, `mkdir()`, `mkfifo()`, `mknod()`, `open()`, `readlink()`, `remove()`, `rename()`, `replace()`, `rmdir()`, `stat()`, `symlink()`, `unlink()`, `utime()`. Platform support for using these parameters can be checked via the sets `os.supports_dir_fd` and `os.supports_follow_symlinks`.
    - Las siguientes funciones ahora admiten un descriptor de archivos para su argumento de ruta: `chdir()`, `chmod()`, `chown()`, `execve()`, `listdir()`, `pathconf()`, `exists()`, `stat()`, `statvfs()`, `utime()`. El soporte de la plataforma para ésto puede ser verificado a través del conjunto `os.supports_fd`.

- `access()` acepta un argumento de palabra clave `effective_ids` acepta el uso del uid/gid efectivo en lugar del uid/gid real en la comprobación de acceso. El soporte de la plataforma para ésto, se puede comprobar a través del conjunto `supports_effective_ids`.

- El módulo `os` tiene dos nuevas funciones `getpriority()`

y `setpriority()`. Estas pueden ser usadas para establecer o recuperar la prioridad en una forma similar a `os.nice()` pero extendidos a todos los procesos en vez de solo al proceso actual.

(Revisión enviada por Giampaolo Rodolà en bpo-10784 [https://bugs.python.org/issue?@action=redirect&bpo=10784].)

- La nueva función `os.replace()` permite cambiar el nombre de un archivo entre plataformas sobrescribiendo el destino. Con `os.rename()`, un archivo de destino existente se sobrescribe bajo POSIX, pero lanza un error en Windows. (Contribución de Antoine Pitrou en bpo-8828 [https://bugs.python.org/issue?@action=redirect&bpo=8828].)
- La familia de estadísticas de funciones (`stat()`, `fstat()`, y `lstat()`) ahora admite la lectura de marcas de tiempo de un archivo con precisión de nanosegundos. Simétricamente, `utime()` ahora puede escribir marcas de tiempo de archivos con precisión de nanosegundos. (Contribución de Larry Hastings en bpo-14127 [https://bugs.python.org/issue?@action=redirect&bpo=14127].)
- La nueva función `os.get_terminal_size()` consulta el tamaño del terminal adjunto al descriptor del archivo. Vea también `shutil.get_terminal_size()`. (Contribución de Zbigniew Jędrzejewski-Szmek en bpo-13609 [https://bugs.python.org/issue?@action=redirect&bpo=13609].)
- Nuevas funciones para admitir atributos extendidos de Linux. (bpo-12720 [https://bugs.python.org/issue?@action=redirect&bpo=12720]): `getxattr()`, `listxattr()`, `removexattr()`, `setxattr()`.
- Nueva interfaz para el planificador. Estas funciones controlan como el sistema operativo le asigna tiempo de CPU a un proceso. Nuevas funciones: `sched_get_priority_max()`, `sched_get_priority_min()`, `sched_getaffinity()`, `sched_getparam()`, `sched_getscheduler()`, `sched_rr_get_interval()`, `sched_setaffinity()`, `sched_setparam()`, `sched_setscheduler()`, `sched_yield()`,

- Nuevas funciones para controlar el sistema de archivos:
    - `posix_fadvise()`: Anuncia una intención de acceder a los datos en un patrón específico, permitiendo al kernel hacer optimizaciones.
    - `posix_fallocate()`: Garantiza que se asigne el espacio suficiente a un archivo.
    - `sync()`: Fuerza la escritura de cualquier cosa en el disco.
- Nuevas funciones posix adicionales:
    - `lockf()`: Aplicar, probar o eliminar un bloqueo POSIX en un descriptor de archivo abierto.
    - `pread()`: Leer desde un descriptor de archivo en un desplazamiento, el desplazamiento del archivo permanece sin cambios.
    - `pwrite()`: Escribe en un descriptor de archivo desde un desplazamiento, dejando el desplazamiento del archivo sin cambios.
    - `readv()`: Leer desde el descriptor de un archivo en varios búferes editables.
    - `truncate()`: Truncar el archivo correspondiente a *path*, para que tenga como máximo *length bytes* de tamaño.
    - `waitid()`: Espera que se completen uno o más procesos secundarios.
    - `writev()`: Escribir los contenidos de *buffers* a un archivo descriptor, donde *buffers* es una secuencia arbitraria de *buffers*.
    - `getgrouplist()` (bpo-9344 [https://bugs.python.org/issue?@action=redirect&bpo=9344]): Retorna una lista de ids de grupo al que pertenece el usuario especificado.
- `times()` and `uname()`: El tipo de retorno cambió de tupla a un objeto tipo tupla con atributos con nombre.
- Varias plataformas admiten constantes adicionales para la función `lseek()`, tal como `os.SEEK_HOLE` y `os.SEEK_DATA`.
- New constants `RTLD_LAZY`, `RTLD_NOW`, `RTLD_GLOBAL`, `RTLD_LOCAL`, `RTLD_NODELETE`, `RTLD_NOLOAD`, and `RTLD_DEEPBIND` are available on platforms that support them. These are for use with the `sys.setdlopenflags()` function, and supersede the similar constants defined in

`ctypes` and **`DLFCN`**. (Contributed by Victor Stinner in bpo-13226 [https://bugs.python.org/issue?@action=redirect&bpo=13226].)

- `os.symlink()` ahora acepta (e ignora) el argumento de palabra clave `target_is_directory` en plataformas diferentes a windows, para facilitar el soporte multiplataforma.

pdb

La finalización de tabulaciones ahora está disponible no solo para los nombres de los comandos, sino también para sus argumentos. Por ejemplo, para el comando `break`, se completan las funciones y los nombres de archivo.

(Contribución de Georg Brandl en bpo-14210 [https://bugs.python.org/issue?@action=redirect&bpo=14210])

pickle

Los objetos `pickle.Pickler` ahora tienen un atributo opcional `dispatch_table` que permite establecer funciones de reducción por recolector.

(Contribución de Richard Oudkerk in bpo-14166 [https://bugs.python.org/issue?@action=redirect&bpo=14166].)

pydoc

The Tk GUI and the **`serve()`** function have been removed from the `pydoc` module: `pydoc -g` and **`serve()`** have been deprecated in Python 3.2.

re

Las expresiones regulares de la clase `str` ahora admiten escapes de `\u` and `\U`.

(Contribución de Serhiy Storchaka en bpo-3665 [https://bugs.python.org/issue?@action=redirect&bpo=3665].)

sched

- `run()` ahora acepta un parámetro *blocking* que al configurarse en false, hace que el método ejecute los eventos planificados debido a la pronta expiración (si corresponde) y luego responde inmediatamente. Esto es útil en caso de que quieras usar la `scheduler` en aplicaciones sin bloqueo. (Contribución de Giampaolo Rodolà in bpo-13449 [https://bugs.python.org/issue?@action=redirect&bpo=13449].)
- La clase `scheduler` ahora se puede utilizar de forma segura en entornos multiproceso. (Contribución por Josiah Carlson y Giampaolo Rodolà en bpo-8684 [https://bugs.python.org/issue?@action=redirect&bpo=8684].)
- Los parámetros *timefunc* y *delayfunct* del constructor de la clase `scheduler` ahora son opcionales y tienen el valor predeterminado `time.time()` y `time.sleep()` respectivamente. (Contribución de Chris Clark en bpo-13245 [https://bugs.python.org/issue?@action=redirect&bpo=13245].)
- El parámetro *argument* de `enter()` and `enterabs()` ahora es opcional. (Contribución de Chris Clark en bpo-13245 [https://bugs.python.org/issue?@action=redirect&bpo=13245].)
- `enter()` y `enterabs()` ahora aceptan un parámetro *kwargs*. (Contribución de Chris Clark en bpo-13245 [https://bugs.python.org/issue?@action=redirect&bpo=13245].)

select

Solaris y plataformas derivadas tienen una nueva clase `select.devpoll` para sockets asincrónicos de alto rendimiento a través de `/dev/poll`. (Contribución de Jesús Cea Avión en bpo-6397 [https://bugs.python.org/issue?@action=redirect&bpo=6397].)

shlex

La función auxiliar `quote`, que anteriormente no estaba documentada, de los módulos **`pipes`** se ha trasladado al módulo `shlex` y se ha documentado. `quote()` escapa correctamente todos los caracteres de una cadena a la que, de otro modo, el shell podría asignarle un significado especial.

shutil

- Nuevas funciones:
    - `disk_usage()`: Proporciona estadísticas sobre el espacio total, usado, y libre en el disco. (Contribución de Giampaolo Rodolà in bpo-12442 [https://bugs.python.org/issue?@action=redirect&bpo=12442].)
    - `chown()`: Permite cambiar el usuario y/o grupo de las rutas dadas, además de especificar los nombres de usuario/grupo, y no solos sus id's numèricos. (Contribución de Sandro Tosi in bpo-12191 [https://bugs.python.org/issue?@action=redirect&bpo=12191].)
    - `shutil.get_terminal_size()`: Retorna el tamaño de la ventana del terminar a la cual se adjunta el intérprete. (Contribución de Zbigniew Jędrzejewski-Szmek in bpo-13609 [https://bugs.python.org/issue?@action=redirect&bpo=13609].)
- `copy2()` and `copystat()` ahora conservan las marcas de tiempo de archivos con precisión de nanosegundos en plataformas que lo soporten. También conservan los «atributos extendidos» de archivo en Linux (Contribución de Larry Hastings en bpo-14127 [https://bugs.python.org/issue?@action=redirect&bpo=14127] y bpo-15238 [https://bugs.python.org/issue?@action=redirect&bpo=15238].)
- Algunas funciones ahora toman un argumento opcional de `symlinks`: Cuando ese parámetro es verdadero, los enlaces simbólicos no se desreferencian y la operación actúa en el enlace simbólico en sí. (o crea uno, si es relevante). (Contribución de Hynek Schlawack in bpo-12715 [https://bugs.python.org/issue?@action=redirect&bpo=12715].)
- Al copiar archivos a un sistema de archivos diferente, `move()` ahora gestiona los enlaces simbólicos de la forma que lo hace el comando `mv` de posix, recreando el enlace simbólico en vez de copiar el contenido del archivo de destino. (Contribución de Jonathan Niehof en bpo-9993 [https://bugs.python.org/issue?@action=redirect&bpo=9993].) **`move()`** ahora también retorna el argumento `dst` como resultado.
- `rmtree()` ahora es resistente a los ataques de enlaces simbólicos en plataformas que admiten el nuevo parámetro `dir_fd` en `os.open()` y `os.unlink()`. (Contribución de

Martin von Löwis and Hynek Schlawack in bpo-4489 [https://bugs.python.org/issue?@action=redirect&bpo=4489].)

signal

- El módulo `signal` tiene nuevas funciones:
    - `pthread_sigmask()`: busca y/o cambia la máscara de señal del hilo de llamada (Contribución de Jean-Paul Calderone in bpo-8407 [https://bugs.python.org/issue?@action=redirect&bpo=8407]);
    - `pthread_kill()`: Envía una señal a un hilo;
    - `sigpending()`: Examina funciones pendientes;
    - `sigwait()`: Espera una señal;
    - `sigwaitinfo()`: Espera una señal, retornando información detallada sobre ella;
    - `sigtimedwait()`: Es similar a `sigwaitinfo()` pero con un tiempo de espera.
- El manejador de señales escribe el número de señal, como un único byte en vez de un byte nulo en el descriptor del archivo de activación. Por lo tanto, es posible esperar más de una señal y saber cuales señales han sido lanzadas.
- `signal.signal()` y `signal.siginterrupt()` lanzan un OSError, en vez de un RuntimeError: OSError no tiene atributo errno.

smtpd

El módulo **`smtpd`** ahora es compatible con **RFC 5321** [https://datatracker.ietf.org/doc/html/rfc5321.html] (SMTP extendido) y **RFC 1870** [https://datatracker.ietf.org/doc/html/rfc1870.html] (extensión de tamaño). Según el estándar, estas extensiones se habilitan solo si el cliente inicia la sesión con un comando `EHLO`.

Soporte inicial de `ELHO` de Alberto Treviño. Ampliación de tamaño por Juhana Jauhiainen. Trabajo adicional sustancial en la revisión aportada por Michele Orrù y Dan Boswell. bpo-8739 [https://bugs.python.org/issue?@action=redirect&bpo=8739])

smtplib

Las clases **`SMTP`**, **`SMTP_SSL`**, y **`LMTP`** ahora aceptan un argumento de palabra clave `source_address`, para especificar el `(host, port)` a usar como dirección de origen en la llamada de enlace al crear el socket saliente. (Contribución de Paulo Scardine en bpo-11281 [https://bugs.python.org/issue?@action=redirect&bpo=11281].)

`SMTP` ahora admite el protocolo de administración de contexto, permitiendo usar a una instancia `SMTP` en una declaración `with`. (Contribución de Giampaolo Rodolà en bpo-11289 [https://bugs.python.org/issue?@action=redirect&bpo=11289].)

El constructor **`SMTP_SSL`** y el método **`starttls()`** ahora aceptan un parámetro SSLContext, para controlar los parámetros del canal seguro. (Contribución por Kasun Herath en bpo-8809 [https://bugs.python.org/issue?@action=redirect&bpo=8809].)

socket

- La clase **`socket`** ahora expone métodos adicionales para procesar datos auxiliares cuando sean compatibles con la plataforma subyacente:
  - **`sendmsg()`**
  - **`recvmsg()`**
  - **`recvmsg_into()`**

  (Contribución de David Watson en bpo-6560 [https://bugs.python.org/issue?@action=redirect&bpo=6560], basado en una revisión anterior de Heiko Wundram)

- La clase **`socket`** ahora admite la familia del protocolo PF_CAN (https://en.wikipedia.org/wiki/Socketcan), en Linux (https://lwn.net/Articles/253425).

  (Contribución de Matthias Fuchs, actualizado por Tiago Gonçalves en bpo-10141 [https://bugs.python.org/issue?@action=redirect&bpo=10141].)

- La clase **`socket`** ahora admite la familia de protocolos PF_RDS (https://en.wikipedia.org/wiki/Reliable_Datagram_Sockets y https://oss.oracle.com/projects/

rds [https://web.archive.org/web/20130115155505/https://oss.oracle.com/projects/rds/]).

- La clase `socket` ahora admite la familia del protocolo `PF_SYSTEM` en OS X. (Contribución de Michael Goderbauer en bpo-13777 [https://bugs.python.org/issue?@action=redirect&bpo=13777].)
- La nueva función `sethostname()` permite establecer el nombre de host en sistemas Unix si el proceso de llamada tiene suficientes privilegios. (Aportado por Ross Lagerwall en bpo-10866 [https://bugs.python.org/issue?@action=redirect&bpo=10866].)

socketserver

`BaseServer` ahora tiene un método reemplazable `service_actions()` que es llamado por el método `serve_forever()` en el ciclo del servicio. `ForkingMixIn` ahora usa ésto para limpiar procesos secundarios zombies. (Contribución de Justin Warkentin en bpo-11109 [https://bugs.python.org/issue?@action=redirect&bpo=11109].)

sqlite3

Nuevo método `sqlite3.Connection` `set_trace_callback()` puede ser usado para capturar un rastro de todos los comandos sql procesados por sqlite. (Contribución de Torsten Landschoff en bpo-11688 [https://bugs.python.org/issue?@action=redirect&bpo=11688].)

ssl

- El módulo `ssl` tiene dos funciones de generación aleatoria:
    - ○ `RAND_bytes()`: genera bytes pseudo-aleatorios criptográficamente fuertes.
    - ○ `RAND_pseudo_bytes()`: generate pseudo-random bytes.

(Contribución de Victor Stinner en bpo-12049 [https://bugs.python.org/issue?@action=redirect&bpo=12049].)

- El módulo `ssl` ahora expone una jerarquía de excepciones más detallada para facilitar la inspección de varias clases de errores. (Contribución de Antoine Pitrou en bpo-11183 [https://bugs.python.org/issue?@action=redirect&bpo=11183].)

- `load_cert_chain()` ahora acepta un argumento *password* que será usado si la clave privada es encriptada. (Contribución de Adam Simpkins en bpo-12803 [https://bugs.python.org/issue?@action=redirect&bpo=12803].)

- El intercambio de claves Diffie-Hellman, tanto regular como basado en curvas elípticas, ahora se admite a través de los métodos `load_dh_params()` y `set_ecdh_curve()`. (Contribución de Antoine Pitrou en bpo-13626 [https://bugs.python.org/issue?@action=redirect&bpo=13626] y bpo-13627 [https://bugs.python.org/issue?@action=redirect&bpo=13627].)

- Los sockets SSL tienen un nuevo método `get_channel_binding()` que permite la implementación de ciertos mecanismos de autenticación tales como SCRAM-SHA-1-PLUS. (Contribución por Jacek Konieczny en bpo-12551 [https://bugs.python.org/issue?@action=redirect&bpo=12551].)

- Puedes consultar el algoritmo de compresión SSL usado por un socket SSL, gracias a su nuevo método `compression()`. El nuevo atributo `OP_NO_COMPRESSION` puede ser usado para deshabilitar la compresión. (Contribución de Antoine Pitrou en bpo-13634 [https://bugs.python.org/issue?@action=redirect&bpo=13634].)

- Se ha agregado soporte para la extensión *Next Protocol Negotiation* usando el método `ssl.SSLContext.set_npn_protocols()`. (Contribución por Colin Marc en bpo-14204 [https://bugs.python.org/issue?@action=redirect&bpo=14204].)

- Los errores de SSL ahora pueden introspectarse más fácilmente gracias a los atributos `library` y `reason`. (Contribución de Antoine Pitrou en bpo-14837 [https://bugs.python.org/issue?@action=redirect&bpo=14837].)

- La función `get_server_certificate()` ahora admite IPv6. (Contribución por Charles-François Natali en bpo-11811 [https://bugs.python.org/issue?@action=redirect&bpo=11811].)
- El nuevo atributo `OP_CIPHER_SERVER_PREFERENCE` permite configurar los sockets del servidor SSLv3 para usar la preferencia de ordenación de cifrado del servidor en lugar de la del cliente. (bpo-13635 [https://bugs.python.org/issue?@action=redirect&bpo=13635]).

stat

La función indocumentada tarfile.filemode ha sido movida a `stat.filemode()`. Este puede ser usado para convertir el modo de un archivo a una cadena de la forma "-rwxrwxrwx".

(Contribución de Giampaolo Rodolà in bpo-14807 [https://bugs.python.org/issue?@action=redirect&bpo=14807].)

struct

El módulo `struct` ahora admite `ssize_t` y `size_t` a través de los nuevos códigos `n` and `N`, respectivamente. (Contribución de Antoine Pitrou en bpo-3163 [https://bugs.python.org/issue?@action=redirect&bpo=3163].)

subprocess

Las cadenas de comandos ahora pueden ser objetos de *bytes* en las plataformas posix. (Contribución de Victor Stinner in bpo-8513 [https://bugs.python.org/issue?@action=redirect&bpo=8513].)

Una nueva constante `DEVNULL` permite suprimir la salida de forma independiente de la plataforma. (Contribución de Ross Lagerwall en bpo-5870 [https://bugs.python.org/issue?@action=redirect&bpo=5870].)

sys

El módulo `sys` tiene un nuevo `thread_info` named tuple que almacena información sobre la implementación del hilo (bpo-11223 [https://bugs.python.org/issue?@action=redirect&bpo=11223]).

tarfile

`tarfile` ahora admite la codificación `lzma` a través del módulo `lzma`. (Contribución de Lars Gustäbel en bpo-5689 [https://bugs.python.org/issue?@action=redirect&bpo=5689].)

tempfile

`tempfile.SpooledTemporaryFile`'s `truncate()` method now accepts a `size` parameter. (Contributed by Ryan Kelly in bpo-9957 [https://bugs.python.org/issue?@action=redirect&bpo=9957].)

textwrap

El módulo `textwrap` tiene una nueva función `indent()` que facilita la adición de un prefijo común para las lineas seleccionadas en un bloque de texto (bpo-13857 [https://bugs.python.org/issue?@action=redirect&bpo=13857]).

threading

`threading.Condition`, `threading.Semaphore`, `threading.BoundedSemaphore`, `threading.Event`, y `threading.Timer`, todas las cuales son usadas para ser funciones de fábrica que retornen una instancia de clase, ahora son clases y pueden ser subclasificadas (Contribución de Éric Araujo in bpo-10968 [https://bugs.python.org/issue?@action=redirect&bpo=10968].)

El constructor de la clase `threading.Thread` ahora acepta un argumento de palabra clave `daemon` para anular el comportamiento predeterminado de heredar el valor del indicador `daemon` desde el hilo principal (bpo-6064 [https://bugs.python.org/issue?@action=redirect&bpo=6064]).

La función anteriormente privada `_thread.get_ident` ahora está disponible como función pública **`threading.get_ident()`**. Ésto elimina varios casos de acceso directo al módulo `_thread` en el stdlib. El código de terceros que usaba `_thread.get_ident` debería de igual forma ser cambiado para usar la nueva interfaz pública.

time

El **PEP 418** [https://peps.python.org/pep-0418/] agrega nuevas funciones al módulo **`time`**:

- **`get_clock_info()`**: Obtiene información en un reloj.
- **`monotonic()`**: Reloj monotónico (no se puede retroceder), no se ve afectado por la actualización del reloj del sistema.
- **`perf_counter()`**: Contador de rendimiento con la más alta resolución disponible para medir una corta duración.
- **`process_time()`**: Suma del tiempo de CPU del sistema y del usuario del proceso actual.

Otras nuevas funciones:

- Las funciones **`clock_getres()`**, **`clock_gettime()`** y **`clock_settime()`** con constantes `CLOCK_xxx` (Contribución de Victor Stinner in bpo-10278 [https://bugs.python.org/issue?@action=redirect&bpo=10278].)

Para mejorar la consistencia entre plataformas, la función **`sleep()`** ahora lanza un **`ValueError`** cuando se le pasa un valor negativo. Esto era anteriormente un error en posix, pero producía una suspensión definitiva en windows.

types

Adición de una nueva clase **`types.MappingProxyType`**: Proxy de solo lectura de una asignación. (bpo-14386 [https://bugs.python.org/issue?@action=redirect&bpo=14386])

Las nuevas funciones **`types.new_class()`** y **`types.prepare_class()`** proporcionan soporte para la creación de tipos dinámicos compatibles. (bpo-14588 [https://bugs.python.org/

issue?@action=redirect&bpo=14588])

unittest

Los métodos `assertRaises()`, `assertRaisesRegex()`, `assertWarns()`, y `assertWarnsRegex()` ahora aceptan un argumento de palabra clave *msg* cuando son usados como administradores de contexto. (Contribución por Ezio Melotti y Winston Ewert en bpo-10775 [https://bugs.python.org/issue?@action=redirect&bpo=10775].)

`unittest.TestCase.run()` ahora retorna el objeto `TestResult`.

urllib

La clase `Request` ahora acepta un argumento *method* usado por `get_method()` para determinar que método HTTP debería usarse. Por ejemplo, esto enviará una solicitud `'HEAD'`:

```
>>> urlopen(Solicitud('https://www.python.org', método='
```

(bpo-1673007 [https://bugs.python.org/issue?@action=redirect&bpo=1673007])

webbrowser

El módulo `webbrowser` admite más «navegadores»: Google Chrome (llamado **chrome**, **chromium**, **chrome-browser** or **chromium-browser** dependiendo de la versión y del sistema operativo), y los lanzadores genéricos **xdg-open**, del proyecto FreeDesktop.org, y **gvfs-open**, el cual es el controlador URI predeterminado para GNOME3. (El primero agregado por Arnaud Calmettes en bpo-13620 [https://bugs.python.org/issue?@action=redirect&bpo=13620], el segundo por Matthias Klose en bpo-14493 [https://bugs.python.org/issue?@action=redirect&bpo=14493].)

xml.etree.ElementTree

The `xml.etree.ElementTree` module now imports its C accelerator by default; there is no longer a need to explicitly import **`xml.etree.cElementTree`** (this module stays for backwards compatibility, but is now deprecated). In addition, the `iter` family of methods of **`Element`** has been optimized (rewritten in C). The module's documentation has also been greatly improved with added examples and a more detailed reference.

zlib

El nuevo atributo **`zlib.Decompress.eof`** permite distinguir entre un flujo comprimido correctamente formado y uno incompleto o truncado. (Aportado por Nadeem Vawda en bpo-12646 [https://bugs.python.org/issue?@action=redirect&bpo=12646].)

Nuevo atributo **`zlib.ZLIB_RUNTIME_VERSION`** informa la cadena de versión de la biblioteca `zlib` subyacente que se carga en tiempo de ejecución. (Contribución de Torsten Landschoff en bpo-12306 [https://bugs.python.org/issue?@action=redirect&bpo=12306].)

Optimizaciones

Se han agregado importantes mejoras de rendimiento:

- Gracias a **PEP 393** [https://peps.python.org/pep-0393/], algunas operaciones en las cadenas *Unicode* han sido optimizadas:
    - la huella de memoria se divide de 2 a 4 según el texto
    - codificar una cadena ASCII a UTF-8 ya no necesita codificar caracteres, la representación UTF-8 se comparte con la representación ASCII
    - el codificador UTF-8 ha sido optimizado
    - repetir una única letra ASCII y obtener una subcadena de una cadena ASCII es 4 veces más rápido
- UTF-8 ahora es 2x a 4x más rápido. La codificación UTF-16 ahora es 10x más rápida.

  (Contribución de Serhiy Storchaka, bpo-14624 [https://bugs.python.org/issue?@action=redirect&bpo=14624], bpo-14738 [https://bugs.python.org/issue?@action=redirect&bpo=14738] y

bpo-15026 [https://bugs.python.org/issue?@action=redirect&bpo=15026].)

Construcción y cambios en la API de C

Los cambios en el proceso de compilación de Python y en la API de C incluyen:

- Nueva función relacionada con **PEP 3118** [https://peps.python.org/pep-3118/]:
  - `PyMemoryView_FromMemory()`
- **PEP 393** [https://peps.python.org/pep-0393/] Agregó nuevos tipos *unicode*, macros y funciones:
  - API de alto nivel:
    - `PyUnicode_CopyCharacters()`
    - `PyUnicode_FindChar()`
    - `PyUnicode_GetLength()`, `PyUnicode_GET_LENGTH`
    - `PyUnicode_New()`
    - `PyUnicode_Substring()`
    - `PyUnicode_ReadChar()`, `PyUnicode_WriteChar()`
  - API de bajo nivel:
    - Tipos `Py_UCS1`, `Py_UCS2`, `Py_UCS4`
    - Estructuras `PyASCIIObject` y `PyCompactUnicodeObject`
    - `PyUnicode_READY`
    - `PyUnicode_FromKindAndData()`
    - `PyUnicode_AsUCS4()`, `PyUnicode_AsUCS4Copy()`
    - `PyUnicode_DATA`, `PyUnicode_1BYTE_DATA`, `PyUnicode_2BYTE_DATA`, `PyUnicode_4BYTE_DATA`
    - `PyUnicode_KIND` with `PyUnicode_Kind` enum: `PyUnicode_WCHAR_KIND`, `PyUnicode_1BYTE_KIND`, `PyUnicode_2BYTE_KIND`, `PyUnicode_4BYTE_KIND`
    - `PyUnicode_READ`, `PyUnicode_READ_CHAR`, `PyUnicode_WRITE`

    - **`PyUnicode_MAX_CHAR_VALUE`**
- **`PyArg_ParseTuple`** ahora acepta un **`bytearray`** para el formato `c` (bpo-12380 [https://bugs.python.org/issue?@action=redirect&bpo=12380]).

Obsoleto

Sistemas operativos no compatibles

OS/2 y VMS no son compatibles debido a la falta de un responsable de mantenimiento.

Windows 2000 y las plataformas windows que establezcan `COMSPEC` a `command.com` ya no son compatibles debido a la carga de mantenimiento.

La compatibilidad con OSF, que quedó obsoleta en 3.2, ha sido completamente eliminada.

Módulos , funciones y métodos obsoletos en Python

- Pasar una cadena no vacía a `object.__format__()` está descontinuado, y producirá un **`TypeError`** en Python 3.4 (bpo-9856 [https://bugs.python.org/issue?@action=redirect&bpo=9856]).
- El códec `unicode_internal` quedó obsoleto a causa del **PEP 393** [https://peps.python.org/pep-0393/], use UTF-8, UTF-16 (`utf-16-le` o `utf-16-be`), o UTF-32 (`utf-32-le` or `utf-32-be`)
- **`ftplib.FTP.nlst()`** y **`ftplib.FTP.dir()`**: use **`ftplib.FTP.mlsd()`**
- **`platform.popen()`**: use the **`subprocess`** module. Check especially the Replacing Older Functions with the subprocess Module section (bpo-11377 [https://bugs.python.org/issue?@action=redirect&bpo=11377]).
- bpo-13374 [https://bugs.python.org/issue?@action=redirect&bpo=13374]: La API de *bytes* de Windows ha sido descontinuada en el módulo **`os`**. Use nombres de archivo *Unicode*, en vez de nombres de archivo de bytes, para no

depender más de la página de códigos ANSI y admitir cualquier nombre de archivo.

- bpo-13988 [https://bugs.python.org/issue?@action=redirect&bpo=13988]: The `xml.etree.cElementTree` module is deprecated. The accelerator is used automatically whenever available.
- The behaviour of `time.clock()` depends on the platform: use the new `time.perf_counter()` or `time.process_time()` function instead, depending on your requirements, to have a well defined behaviour.
- The `os.stat_float_times()` function is deprecated.
- `abc` module:
  - La clase `abc.abstractproperty` ha quedado obsoleta, en su lugar use la clase `property` con `abc.abstractmethod()`.
  - La clase `abc.abstractclassmethod` ha quedado obsoleta. En su lugar use el método `classmethod` with `abc.abstractmethod()`.
  - `abc.abstractstaticmethod` ha quedado obsoleta, use `staticmethod` with `abc.abstractmethod()`.
- paquete `importlib`:
  - `importlib.abc.SourceLoader.path_mtime()` ha quedado obsoleto, a favor de `importlib.abc.SourceLoader.path_stats()` ya que los archivos de código de *bytes* ahora almacenan tanto la hora de modificación como el tamaño del archivo fuente desde el cual se compiló el archivo de código de bytes.

Funciones y tipos obsoletos en la API de C

El `Py_UNICODE` ha quedado obsoleto por PEP 393 [https://peps.python.org/pep-0393/] y será eliminado en Python 4. Todas las funciones que utilizan éste tipo quedarán en desuso:

Funciones y métodos Unicode que utilizan los tipos `Py_UNICODE` y Py_UNICODE*:

- `PyUnicode_FromUnicode`: use `PyUnicode_FromWideChar()` o

`PyUnicode_FromKindAndData()`

- **`PyUnicode_AS_UNICODE`**, **`PyUnicode_AsUnicode()`**, **`PyUnicode_AsUnicodeAndSize()`**: use `PyUnicode_AsWideCharString()`
- **`PyUnicode_AS_DATA`**: use `PyUnicode_DATA` con `PyUnicode_READ` y `PyUnicode_WRITE`
- **`PyUnicode_GET_SIZE`**, **`PyUnicode_GetSize()`**: use `PyUnicode_GET_LENGTH` o `PyUnicode_GetLength()`
- **`PyUnicode_GET_DATA_SIZE`**: use `PyUnicode_GET_LENGTH(str) * PyUnicode_KIND(str)` (sólo funciona en cadenas listas)
- **`PyUnicode_AsUnicodeCopy()`**: use `PyUnicode_AsUCS4Copy()` o `PyUnicode_AsWideCharString()`
- **`PyUnicode_GetMax()`**

Funciones y macros que manipulen cadenas de caracteres Py_UNICODE*:

- **`Py_UNICODE_strlen`**: use `PyUnicode_GetLength()` o `PyUnicode_GET_LENGTH`
- **`Py_UNICODE_strcat`**: use `PyUnicode_CopyCharacters()` o `PyUnicode_FromFormat()`
- **`Py_UNICODE_strcpy`**, **`Py_UNICODE_strncpy`**, **`Py_UNICODE_COPY`**: use `PyUnicode_CopyCharacters()` o `PyUnicode_Substring()`
- **`Py_UNICODE_strcmp()`**: use `PyUnicode_Compare()`
- **`Py_UNICODE_strncmp()`**: use `PyUnicode_Tailmatch()`
- **`Py_UNICODE_strchr()`**, **`Py_UNICODE_strrchr()`**: use `PyUnicode_FindChar()`
- **`Py_UNICODE_FILL()`**: use `PyUnicode_Fill()`
- **`Py_UNICODE_MATCH`**

Codificadores:

- **`PyUnicode_Encode()`**: use **`PyUnicode_AsEncodedObject()`**
- **`PyUnicode_EncodeUTF7()`**
- **`PyUnicode_EncodeUTF8()`**: use `PyUnicode_AsUTF8()` o `PyUnicode_AsUTF8String()`

- **`PyUnicode_EncodeUTF32()`**
- **`PyUnicode_EncodeUTF16()`**
- **`PyUnicode_EncodeUnicodeEscape()`** use `PyUnicode_AsUnicodeEscapeString()`
- **`PyUnicode_EncodeRawUnicodeEscape()`** use `PyUnicode_AsRawUnicodeEscapeString()`
- **`PyUnicode_EncodeLatin1()`**: use `PyUnicode_AsLatin1String()`
- **`PyUnicode_EncodeASCII()`**: use `PyUnicode_AsASCIIString()`
- **`PyUnicode_EncodeCharmap()`**
- **`PyUnicode_TranslateCharmap()`**
- **`PyUnicode_EncodeMBCS()`**: use `PyUnicode_AsMBCSString()` o `PyUnicode_EncodeCodePage()` (con `CP_ACP` code_page)
- **`PyUnicode_EncodeDecimal()`**, **`PyUnicode_TransformDecimalToASCII()`**

Características obsoletas

El código del formato `'u'` del módulo `array` ha quedado obsoleto y será eliminado en Python 4, junto con el resto de la API (`Py_UNICODE`).

Migración a Python 3.3

Esta sección enumera los cambios descritos previamente y otras correcciones que pueden requerir cambios en su código.

Portando código Python

- La aleatoriedad de Hash ha sido habilitada de forma predeterminada. Establezca la variable de entorno `PYTHONHASHSEED` a 0, para deshabilitar la aleatoriedad de Hash. Vea también el método `object.__hash__()` method.
- bpo-12326 [https://bugs.python.org/issue?@action=redirect&bpo=12326]: En Linux, sys.platform ya no contiene la versión principal. Ahora siempre es “linux”, en

vez de “linux2” o “linux3” dependiendo de la versión de Linux usada para compilar Python. Reemplace sys.platform = = “linux2” con sys.platform.startswith(“linux”), o directamente sys.platform = = “linux” si no necesita admitir versiones anteriores de Python.

- bpo-13847 [https://bugs.python.org/issue?@action=redirect&bpo=13847], bpo-14180 [https://bugs.python.org/issue?@action=redirect&bpo=14180]: **`time`** y **`datetime`**: **`OverflowError`** ahora se lanza en lugar de **`ValueError`** si una marca de tiempo está fuera de rango. **`OSError`** ahora se genera si fallaron las funciones C **`gmtime()`** o **`localtime()`**.
- Los buscadores predeterminados usados para las importaciones, ahora utilizan un caché de lo que está contenido en un directorio específico. Si crea un archivo fuente de Python o archivo de código de *bytes* sin fuente, asegúrese de llamar a **`importlib.invalidate_caches()`** para limpiar la caché para que los buscadores encuentren el nuevo archivo.
- **`ImportError`** ahora usa el nombre completo del módulo que se intentó importar. Las pruebas de documentos que verifican el mensaje de ImportErrors necesitaran ser actualizados para usar el nombre completo del módulo en vez de sólo la terminación del nombre.
- El argumento *index* para la función **`__import__()`** ahora tiene un valor predeterminado de 0, en vez de -1, y ya no admite valores negativos. Ha sido un descuido en la implementación de **PEP 328** [https://peps.python.org/pep-0328/] que el valor predeterminado permaneciera en -1. Si necesita continuar realizando una importación relativa seguida por una importación absoluta, entonces realice la importación relativa usando un índice de 1, seguida por otra importación que use un índice de 0. Sin embargo, es preferible usar **`importlib.import_module()`** en vez de llamar a la función **`__import__()`** directamente.
- **`__import__()`** ya no admite usar un valor de índice distinto de 0 para los módulos de alto nivel. Por ejemplo, `__import__('sys', level=1)` ahora es un error.
- Dado que **`sys.meta_path`** y **`sys.path_hooks`** ahora tienen buscadores por defecto, lo más probable es que desee

usar `list.insert()` en vez de `list.append()` para agregar a esas listas.

- Dado que `None` ahora ha sido insertado en `sys.path_importer_cache`, si usted está borrando entradas en el diccionario de rutas que no tienen un buscador, usted necesitará eliminar los pares con valores de `None` y `imp.NullImporter` para ser compatible con versiones anteriores. Esto conducirá a una sobrecarga adicional con las versiones más antiguas de Python que re-inserten `None` dentro de `sys.path_importer_cache` donde éste represente el uso de buscadores implícitos, pero semánticamente no debería cambiar nada.
- `importlib.abc.Finder` ya no especifica un método abstracto `find_module()` que debe implementarse. Si confiaba en subclases para implementar ese método, asegúrese de verificar primero la existencia del método. Sin embargo, probablemente querrá verificar primero `find_loader()`, en el caso de trabajar con path entry finders.
- `pkgutil` ha sido convertido para usar `importlib` internamente. Esto elimina varios casos extremos donde el antiguo comportamiento de la emulación de importación **PEP 302** [https://peps.python.org/pep-0302/] fallaba al comparar el comportamiento del sistema real de importación. La emulación de importación en sí aún está presente, pero está en desuso. Las funciones `pkgutil.iter_importers()` y `pkgutil.walk_packages()` en un caso especial son los ganchos de importación estándar, por lo que aún son compatibles aunque no proporcionen el método no estándar `iter_modules ()`.
- Se ha corregido un error de larga data de cumplimiento de RFC (bpo-1079 [https://bugs.python.org/issue?@action=redirect&bpo=1079]) en el análisis realizado por `email.header.decode_header()`. El código que usa el lenguaje estándar para convertir encabezados codificados en *unicode* (`str (make_header (decode_header (h))`) no verá cambios, pero el código que mira las tuplas individuales retornadas por decode_header verá ese espacio en blanco que precede o sigue las secciones `ASCII` ahora está incluido en la sección `ASCII`. El código que crea encabezados usando

`make_header` también debería continuar funcionando sin cambios, ya que `make_header` continúa agregando espacios en blanco entre `Secciones ASCII` y no `ASCII` si aún no está presente en las cadenas de caracteres entrantes.

- **`email.utils.formataddr()`** ahora realiza la codificación de transferencia de contenido correcta, al pasar nombres para mostrar que no son ASCII. Cualquier código que dependa del comportamiento defectuoso anterior que conservó el *unicode* que no es `ASCII` en la cadena de salida formateada deberá cambiarse (bpo-1690608 [https://bugs.python.org/issue?@action=redirect&bpo=1690608]).
- **`poplib.POP3.quit()`** ahora puede generar errores de protocolos como todos los demás métodos `poplib`. El código que asume que `quit` no genera errores **`poplib.error_proto`** puede necesitar ser cambiado si una aplicación en particular encuentran errores en `quit` (bpo-11291 [https://bugs.python.org/issue?@action=redirect&bpo=11291]).
- El argumento `strict` para la clase **`email.parser.Parser`**, que quedó obsoleto desde Python 2.4, finalmente ha sido eliminado.
- El método `unittest.TestCase.assertSameElements` que estaba obsoleto, ha sido eliminado.
- La variable `time.accept2dyear` que estaba obsoleta, ha sido eliminada.
- El atributo `Context._clamp` que estaba obsoleto, ha sido eliminado del módulo **`decimal`**. Este se reemplazó anteriormente por el atributo público **`clamp`**. (Ver bpo-8540 [https://bugs.python.org/issue?@action=redirect&bpo=8540].)
- La clase interna auxiliar indocumentada `SSLFakeFile`, ha sido eliminada de **`smtplib`**, dado que su funcionalidad ha sido proporcionada directamente por **`socket.socket.makefile()`**.
- El paso de valores negativos a la función **`time.sleep()`** en Windows, ahora lanza un error en vez de detenerse definitivamente. Este siempre ha lanzado un error en posix.
- La constante `ast.__version__` ha sido eliminada. Si usted necesita tomar decisiones afectadas por la versión AST, use **`sys.version_info`** para tomar la decisión.
- El código que solía solucionar el hecho de que el módulo

`threading` utilizaba funciones de fábrica mediante la subclase de las clases privadas tendrá que cambiar a la subclase de las clases ahora públicas.
- La indocumentada maquinaria de depuración en el módulo threading ha sido eliminada, simplificando el código. Esto no debería tener efecto en el código de producción, pero se menciona aquí en caso de que algún framework de depuración de aplicaciones interactúe con el (bpo-13550 [https://bugs.python.org/issue?@action=redirect&bpo=13550]).

Portando código C

- In the course of changes to the buffer API the undocumented **`smalltable`** member of the **`Py_buffer`** structure has been removed and the layout of the **`PyMemoryViewObject`** has changed.

  Todas las extensiones que quedan en las partes relevantes de `memoryobject.h` o `object.h` deben ser reconstruídas.

- Debido a PEP 393, el tipo **`Py_UNICODE`** y todas las funciones que lo utilicen han quedado obsoletas (pero seguirán estando disponibles por al menos 5 años). Si usted estuvo usando APIS Unicode de bajo nivel para construir y acceder a objetos unicode, y desea beneficiarse de la reducción de huella de memoria proporcionada por **PEP 393** [https://peps.python.org/pep-0393/], usted debe convertir su código al nuevo Unicode API.

  Sin embargo, si sólo ha estado usando funciones de alto nivel como **`PyUnicode_Concat()`**, **`PyUnicode_Join()`** o **`PyUnicode_FromFormat()`**, automáticamente su código se beneficiará de las nuevas representaciones *unicode*.

- **`PyImport_GetMagicNumber()`** ahora retorna `-1` en caso de falla.

- Como valor negativo para el argumento *level* de la función **`__import__()`** ya no es válido, y lo mismo ahora es válido para la función **`PyImport_ImportModuleLevel()`**. Esto también significa que el valor de *level* usado por la función

`PyImport_ImportModuleEx()` ahora es `0` en vez de `-1`.

Construyendo extensiones C

- Se ha reducido el rango de nombres de archivo posibles para las extensiones C. Muy raramente se han suprimido las ortografías utilizadas: en POSIX, los archivos denominados `xxxmodule.so`, `xxxmodule.abi3.so` y `xxxmodule.cpython-*.so` ya no se reconocen como la implementación del módulo `xxx`. Si ha estado generando estos archivos, tiene que cambiar a las otras ortografías (es decir, eliminar la cadena de caracteres `módulo` de los nombres de archivo).

  (implementado en bpo-14040 [https://bugs.python.org/issue?@action=redirect&bpo=14040].)

Cambios en el conmutador de línea de comandos

- Se han quitado el indicador de línea de comandos -Q y los artefactos relacionados. La comprobación de código sys.flags.division_warning tendrá que actualizar.

  (bpo-10998 [https://bugs.python.org/issue?@action=redirect&bpo=10998], contribución de Éric Araujo.)

- Cuando se inicia **python** con **`-S`**, `import site` ya no agregará las rutas específicas de sitio al módulo de búsqueda de rutas. En las versiones anteriores si lo hacía.

  (bpo-11591 [https://bugs.python.org/issue?@action=redirect&bpo=11591], contribución de Carl Meyer con ediciones de Éric Araujo.)

Qué hay de nuevo en Python 3.2

Autor:

Raymond Hettinger

Este artículo explica las nuevas características de Python 3.2 en comparación con 3.1. Python 3.2 se lanzó el 20 de febrero de 2011. Se centra en algunos aspectos destacados y ofrece algunos ejemplos. Para obtener detalles completos, consulte el archivo Misc/NEWS [https://github.com/python/cpython/blob/076ca6c3c8df3030307e548d9be792ce3c1c6eea/Misc/NEWS].

Ver también

PEP 392 [https://peps.python.org/pep-0392/] - Calendario de Publicación Python 3.2

PEP 384: Definición de un ABI estable

En el pasado, los módulos de extensión creados para una versión de Python a menudo no se podían usar con otras versiones de Python. Particularmente en Windows, cada lanzamiento de funciones de Python requería reconstruir todos los módulos de extensión que uno quería usar. Este requisito fue el resultado del acceso gratuito a los componentes internos del intérprete de Python que los módulos de extensión podían usar.

Con Python 3.2, está disponible un enfoque alternativo: los módulos de extensión que se restringen a una API limitada (al definir Py_LIMITED_API) no pueden usar muchas de las funciones internas, pero están restringidos a un conjunto de funciones API que se promete serán estables para varias versiones. Como consecuencia, los módulos de extensión creados para 3.2 en ese modo también funcionarán con 3.3, 3.4, etc. Los módulos de extensión que hacen uso de los detalles de las estructuras de memoria aún se pueden construir, pero deberán recompilarse para cada lanzamiento de funciones.

Ver también

PEP 384 [https://peps.python.org/pep-0384/] - Definición de un ABI estable
PEP escrito por Martin von Löwis.

PEP 389: Módulo de análisis sintáctico (*Parser*) de línea de comandos Argparse

Se introdujo un nuevo módulo para el análisis de la línea de comandos, `argparse`, para superar las limitaciones de `optparse` que no brindaba soporte para argumentos posicionales (no solo opciones), subcomandos, opciones requeridas y otros patrones comunes para especificar y validar opciones.

Este módulo ya ha tenido un éxito generalizado en la comunidad como módulo de terceros. Con más funciones que su predecesor, el módulo `argparse` es ahora el módulo preferido para el procesamiento de la línea de comandos. El módulo más antiguo todavía se mantiene disponible debido a la gran cantidad de código heredado que depende de él.

Aquí hay un analizador de ejemplo comentado que muestra características de cómo limitar los resultados a un conjunto de opciones, especificar un *metavar* en la pantalla de ayuda, validar que uno o más argumentos posicionales están presentes y hacer una opción requerida::

```
import argparse
parser = argparse.ArgumentParser(
description = 'Administrar servidores', # descripción pr
epilog = 'Probado en Solaris y Linux') # mostrado despué
parser.add_argument('action', # nombre del argumento
choices = ['deploy', 'start', 'stop'], # tres valores pe
help = 'action on each target') # mensaje de ayuda
parser.add_argument('targets',
metavar = 'HOSTNAME', # nombre de la variable usada en e
nargs = '+', # requiere uno o más objetivos
help = 'url para las máquinas de destino') # explicación
parser.add_argument('-u', '--user', # opción -u o --user
required = True, # haz que sea un argumento obligatorio
help = 'iniciar sesión como usuario')
```

Ejemplo de llamada al analizador en una cadena de caracteres de comandos:

```
>>> cmd = 'implementar estornudos.ejemplo.com sueño.ejem
>>> resultado = parser.parse_args(cmd.split())
>>> resultado.acción
'implementar'
>>> resultado.objetivos
['estornudos.ejemplo.com', 'sueño.ejemplo.com']
>>> resultado.usuario
'skycaptain'
```

Ejemplo de ayuda generada automáticamente por el analizador:

```
>>> parser.parse_args('-h'.split())

uso: manage_cloud.py [-h] -u USER
{deploy,start,stop} HOSTNAME [HOSTNAME ...]

Administrar servidores

argumentos posicionales:
{deploy,start,stop} acción en cada destino
HOSTNAME url para las máquinas de destino

argumentos opcionales:
-h, --help mostrar este mensaje de ayuda y salir
-u USER, --user USER iniciar sesión como usuario
```

```
Probado en Solaris y Linux
```

Una característica especialmente agradable de **`argparse`** es la capacidad de definir sub-analizadores (*subparsers*), cada uno con sus propios patrones de argumentos y pantallas de ayuda:

```
import argparse
parser = argparse.ArgumentParser(prog='HELM')
subparsers = parser.add_subparsers()

parser_l = subparsers.add_parser('launch', help='Launch
parser_l.add_argument('-m', '--missiles', action='store_
parser_l.add_argument('-t', '--torpedos', action='store_

parser_m = subparsers.add_parser('move', help='Move Vess
aliases=('steer', 'turn')) # nombres equivalentes
parser_m.add_argument('-c', '--course', type=int, requer
parser_m.add_argument('-s', '--speed', tipo=int, valor p

$ ./helm.py --help # ayuda de nivel superior (lanzamient
$ ./helm.py launch --help # ayuda para las opciones de l
$ ./helm.py launch --missiles # establece misiles=True y
$ ./helm.py steer --course 180 --speed 5 # establece par
```

Ver también

PEP 389 [https://peps.python.org/pep-0389/] - Nuevo módulo de análisis de línea de comandos
PEP escrito por Steven Bethard.

Migrating optparse code to argparse para obtener detalles sobre las diferencias con **`optparse`**.

PEP 391: Configuración basada en diccionario para Logging

El módulo **`logging`** proporcionaba dos tipos de configuración: un estilo con llamadas de función para cada opción u otro estilo controlado por un archivo externo guardado en formato **`configparser`**. Esas opciones no proporcionaban la flexibilidad necesaria para crear configuraciones a partir de archivos JSON o YAML, ni tampoco admitían la configuración incremental, que es necesaria para especificar las opciones del registrador desde una línea de comandos.

Para permitir un estilo más flexible, el módulo ahora ofrece **`logging.config.dictConfig()`** para especificar la configuración de logging con diccionarios simples de Python. Las opciones de configuración incluyen formateadores, gestores, filtros y loggers. Aquí hay un ejemplo funcional de un diccionario de configuración:

```
{"versión": 1,
"formateadores": {"breve": {"formato": "%(nombrenivel)-8
"completo": {"formato": "%(tiempoasc)s %(nombre)-15s %(r
},
"controladores": {"consola": {
"clase": "registro.StreamHandler",
"formateador": "breve",
"nivel": "INFO",
"flujo": "ext://sys.stdout"},
"prioridad_consola": {
"clase": "registro.StreamHandler",
"formateador": "completo",
"nivel": "ERROR",
"flujo": "ext://sys.stderr"}
},
"raíz": {"nivel": "DEBUG", "controladores": ["consola",
```

Si ese diccionario está almacenado en un archivo llamado `conf.json`, se puede cargar y llamar con un código como este:

```
>>> import json, logging.config
>>> with open('conf.json') as f:
... conf = json.load(f)
...
>>> logging.config.dictConfig(conf)
>>> logging.info("Transacción completada normalmente")
```

```
INFO : root : Transacción completada normalmente
>>> logging.critical("Terminación anormal")
2011-02-17 11:14:36,694 root CRITICAL Terminación anorma
```

Ver también

PEP 391 [https://peps.python.org/pep-0391/] - Configuración basada en diccionario para Logging
PEP escrito por Vinay Sajip.

PEP 3148: El módulo `concurrent.futures`

El código para crear y administrar la concurrencia se está recopilando en un nuevo espacio de nombres de nivel superior, *concurrent*. Su primer miembro es un paquete *futures* que proporciona una interfaz uniforme de alto nivel para administrar hilos y procesos.

El diseño de `concurrent.futures` se inspiró en el paquete *java.util.concurrent*. En ese modelo, una llamada en ejecución y su resultado están representados por un objeto `Future` que abstrae características comunes a hilos, procesos y llamadas a procedimientos remotos. Ese objeto admite verificaciones de estado (en ejecución o terminadas), tiempos de espera, cancelaciones, agregar devoluciones de llamada y acceso a resultados o excepciones.

El agregado principal del nuevo módulo es un par de clases ejecutoras para lanzar y administrar llamadas. El objetivo de los ejecutores es facilitar el uso de las herramientas existentes para realizar llamadas en paralelo. Ahorran el esfuerzo necesario para configurar un grupo de recursos, lanzar las llamadas, crear una cola de resultados, agregar gestión de tiempo de espera y limitar la cantidad total de hilos, procesos o llamadas a procedimientos remotos.

Idealmente, cada aplicación debería compartir un solo ejecutor en varios componentes para que los límites de procesos e hilos se puedan administrar de forma centralizada. Esto resuelve el desafío de diseño que surge cuando cada componente tiene su propia estrategia competitiva para la gestión de recursos.

Ambas clases comparten una interfaz común con tres métodos: `submit()` para programar un invocable y retornar un objeto `Future` `map()` para programar muchas llamadas asincrónicas a la vez, y `shutdown()` para liberar recursos. La clase es un context manager y se puede usar en una declaración `with` para asegurar que los recursos se liberen automáticamente cuando los futuros actualmente pendientes terminan de ejecutarse.

Un ejemplo simple de `ThreadPoolExecutor` es un lanzamiento de cuatro hilos paralelos para copiar archivos:

```
importar concurrente.futures, shutil
con concurrente.futures.ThreadPoolExecutor(max_workers=4
e.submit(shutil.copy, 'src1.txt', 'dest1.txt')
e.submit(shutil.copy, 'src2.txt', 'dest2.txt')
e.submit(shutil.copy, 'src3.txt', 'dest3.txt')
e.submit(shutil.copy, 'src3.txt', 'dest4.txt')
```

Ver también

PEP 3148 [https://peps.python.org/pep-3148/] - Futuros – Ejecutar Cómputos Asincrónicos

PEP escrito por Brian Quinlan.

Código para lecturas de URL en paralelo con grupos de hilos, un ejemplo que usa hilos para buscar varias páginas web en paralelo.

Código para calcular números primos en paralelo, un ejemplo demostrando `ProcessPoolExecutor`.

PEP 3147: Directorios del repositorio de PYC

El esquema de Python para almacenar en caché bytecode en archivos *.pyc* no funcionó bien en entornos con múltiples intérpretes de Python. Si un intérprete encontrara un archivo en caché creado por otro intérprete, volvería a compilar la fuente y sobrescribiría el archivo en caché, perdiendo así los beneficios del almacenamiento en caché.

El problema de las «peleas de pyc» se ha vuelto más pronunciado a medida que se ha convertido en un lugar común para las distribuciones de Linux enviadas con múltiples versiones de Python. Estos conflictos también surgen con alternativas de CPython como *Unladen Swallow*.

Para resolver este problema, el mecanismo de importación de Python se ha ampliado para utilizar nombres de archivo distintos para cada intérprete. En lugar de que Python 3.2 y Python 3.3 y *Unladen Swallow* compitan por un archivo llamado «mymodule.pyc», ahora buscarán «mymodule.cpython-32.pyc», «mymodule.cpython-33.pyc» y «mymodule .unladen10.pyc». Y para evitar que todos estos nuevos archivos abarroten los directorios de origen, los archivos *pyc* ahora se recopilan en un directorio «_pycache_» almacenado en el directorio del paquete.

Aparte de los nombres de archivo y directorios de destino, el nuevo esquema tiene algunos aspectos que son visibles para el programador:

- Imported modules now have a **`__cached__`** attribute which stores the name of the actual file that was imported:

  ```
  >>> import collections
  >>> collections.__cached__
  'c:/py32/lib/__pycache__/collections.cpython-32.pyc
  ```

- La etiqueta que es única para cada intérprete es accesible desde el módulo **`imp`**:

  ```
  >>> import imp
  >>> imp.get_tag()
  'cpython-32'
  ```

- Los scripts que intentan deducir el nombre del archivo de origen a partir del archivo importado ahora deben ser más inteligentes. Ya no basta con quitar la «c» de un nombre de archivo «.pyc». En su lugar, utilice las nuevas funciones del módulo **`imp`**:

```
>>> imp.source_from_cache('c:/py32/lib/__pycache__/
'c:/py32/lib/collections.py'
>>> imp.cache_from_source('c:/py32/lib/collections.
'c:/py32/lib/__pycache__/collections.cpython-32.pyc
```

- Los módulos **`py_compile`** y **`compileall`** se han actualizado para reflejar la nueva convención de nomenclatura y el directorio de destino. La invocación por línea de comandos de *compileall* tiene nuevas opciones: `-i` para especificar una lista de archivos y directorios para compilar y `-b` que hace que los archivos bytecode se escriban en su ubicación heredada en lugar de *_pycache_*.

- El módulo **`importlib.abc`** se ha actualizado con el nuevo módulo abstract base classes para cargar archivos de bytecode. Los ABC obsoletos, **`PyLoader`** y **`PyPycLoader`**, han quedado obsoletos (las instrucciones sobre cómo mantener la compatibilidad con Python 3.1 se incluyen en la documentación).

Ver también

PEP 3147 [https://peps.python.org/pep-3147/] - Directorio del repositorio PYC

PEP escrito por Barry Warsaw.

PEP 3149: Archivos .so con etiquetado de versión para ABI

El directorio del repositorio PYC permite la ubicación conjunta de varios archivos de cache bytecode. Este PEP implementa un mecanismo similar para los archivos de objetos compartidos dándoles un directorio común y nombres distintos para cada versión.

El directorio común es «pyshared» y los nombres de los archivos se diferencian identificando la implementación de Python (como CPython, PyPy, Jython, etc.), los números de versión principal y secundaria, y las banderas de compilación opcionales (como «d» para debug, «m» para pymalloc, «u» para ancho-unicode). Para un paquete arbitrario «foo», se verán estos archivos cuando se instala el paquete de distribución:

```
/usr/share/pyshared/foo.cpython-32m.so
/usr/share/pyshared/foo.cpython-33md.so
```

En Python mismo, las etiquetas son accesibles desde funciones en el módulo `sysconfig`:

```
>>> import sysconfig
>>> sysconfig.get_config_var('SOABI') # encuentra la eti
'cpython-32mu'
>>> sysconfig.get_config_var('EXT_SUFFIX') # encuentra l
'.cpython-32mu.so'
```

Ver también

PEP 3149 [https://peps.python.org/pep-3149/] - PEP 3149: Archivos .so con etiquetado de versión para ABI
PEP escrito por Barry Warsaw.

PEP 3333: Interfaz de puerta de enlace del servidor web Python v1.0.1

Este PEP informativo aclara cómo el protocolo WSGI debe gestionar los problemas de bytes/texto. El desafío es que el manejo de cadenas de caracteres en Python 3 se maneja de manera más conveniente con el tipo `str` aunque el protocolo HTTP está orientado a bytes.

El PEP diferencia las llamadas *cadenas nativas* que se utilizan para los encabezados y metadatos de solicitud/respuesta (*request/*

response) frente a las *cadenas de bytes* que se utilizan para los cuerpos de las solicitudes y respuestas.

Los *native strings* son siempre del tipo `str`, pero están restringidos a puntos de código entre *U+0000* y *U+00FF* que se pueden traducir a bytes mediante la codificación *Latin-1*. Estas cadenas se utilizan para las claves y los valores en el diccionario de entorno y para los encabezados de respuesta y los estados en la función `start_response()`. Deben seguir a **RFC 2616** [https://datatracker.ietf.org/doc/html/rfc2616.html] con respecto a la codificación. Es decir, deben ser caracteres *ISO-8859-1* o utilizar la codificación MIME **RFC 2047** [https://datatracker.ietf.org/doc/html/rfc2047.html].

Para los desarrolladores que portan aplicaciones WSGI desde Python 2, estos son los puntos destacados:

- Si la aplicación ya usó cadenas para encabezados en Python 2, no es necesario ningún cambio.
- Si, en cambio, la aplicación codificó encabezados de salida o decodificó encabezados de entrada, entonces los encabezados deberán volver a codificarse en Latin-1. Por ejemplo, un encabezado de salida codificado en utf-8 usaba `h.encode('utf-8')`, ahora se necesita convertir de bytes a cadenas nativas usando `h.encode('utf-8').decode('latin-1')`.
- Los valores generados por una aplicación o enviados mediante el método `write()` deben ser cadenas de bytes. La función `start_response()` y el entorno deben utilizar cadenas nativas. No se pueden mezclar.

Para los implementadores de servidores que escriben rutas de CGI a WSGI u otros protocolos de estilo CGI, los usuarios deben poder acceder al entorno utilizando cadenas nativas, aunque la plataforma subyacente pueda tener una convención diferente. Para salvar esta brecha, el módulo `wsgiref` tiene una nueva función, `wsgiref.handlers.read_environ()`, para transcodificar variables CGI de `os.environ` en cadenas nativas y devolver un nuevo diccionario.

Ver también

PEP 3333 [https://peps.python.org/pep-3333/] - Interfaz de puerta de enlace del servidor web Python v1.0.1

PEP escrito por Phillip Eby.

Otros cambios de lenguaje

Algunos cambios más pequeños realizados en el lenguaje central de Python son:

- El formato de cadena de caracteres para `format()` y `str.format()` obtuvo nuevas capacidades para el carácter de formato #. Anteriormente, para enteros en binario, octal o hexadecimal, hacía que la salida tuviera el prefijo “0b”, “0o” o “0x” respectivamente. Ahora también puede manejar flotantes, complejos y decimales, lo que hace que la salida siempre tenga un punto decimal, incluso cuando no le siguen dígitos.

  ```
  >>> format(20, '#o')
  '0o24'
  >>> format(12.34, '#5.0f')
  '   12.'
  ```

  (Sugerido por Mark Dickinson e implementado por Eric Smith en bpo-7094 [https://bugs.python.org/issue?@action=redirect&bpo=7094].)

- There is also a new `str.format_map()` method that extends the capabilities of the existing `str.format()` method by accepting arbitrary mapping objects. This new method makes it possible to use string formatting with any of Python’s many dictionary-like objects such as `defaultdict`, `Shelf`, `ConfigParser`, or `dbm`. It is also useful with custom `dict` subclasses that normalize keys before look-up or that supply a `__missing__()` method for unknown keys:

  ```
  >>> import shelve
  >>> d = shelve.open('tmp.shl')
  ```

```
>>> 'El estado de {project_name} es {status} a part
'El estado del proyecto de prueba es verde a partir

>>> class LowerCasedDict(dict):
... def __getitem__(self, key):
... return dict.__getitem__(self, key.lower())
...
>>> lcd = LowerCasedDict(part='widgets', quantity=1
>>> 'Hay {QUANTITY} {Part} en stock'.format_map(lcd
'Hay 10 widgets en stock'

>>> class PlaceholderDict(dict):
... def __missing__(self, key):
... return '<{}>'.format(key)
...
>>> 'Hola {nombre}, bienvenido a {ubicación}'.forma
'Hola <nombre>, bienvenido a <ubicación>'
```

(Sugerido por Raymond Hettinger e implementado por Eric Smith en bpo-6081 [https://bugs.python.org/issue?@action=redirect&bpo=6081].)

- Ahora es posible iniciar el intérprete con una opción silenciosa, `-q`, para evitar que la información de copyright y de versión se muestre en el modo interactivo. La opción se puede analizar mediante el atributo `sys.flags`:

```
$ python -q
>>> sys.flags
sys.flags(depuración=0, advertencia_de_división=0,
optimización=0, no_escribir_código_de_bytes=0, sin_
ignorar_entorno=0, verbose=0, advertencia_de_bytes=
```

(Contribución de Marcin Wojdyr en bpo-1772833 [https://bugs.python.org/issue?@action=redirect&bpo=1772833]).

- La función `hasattr()` funciona llamando a `getattr()` y detectando si se genera una excepción. Esta técnica le permite detectar métodos creados dinámicamente por `__getattr__()` o `__getattribute__()` que, de otro modo, no estarían en el diccionario de clases. Anteriormente, *hasattr* capturaba cualquier excepción, posiblemente enmascarando errores genuinos. Ahora, *hasattr* se ha ajustado para que solo capture `AttributeError` y permita que pasen otras excepciones:

```
>>> class A:
... @property
... def f(self):
... return 1 // 0
...
>>> a = A()
>>> hasattr(a, 'f')
Traceback (última llamada más reciente):
...
ZeroDivisionError: división entera o módulo por cer
```

(Descubierto por Yury Selivanov y arreglado por Benjamin Peterson; bpo-9666 [https://bugs.python.org/issue?@action=redirect&bpo=9666].)

- La `str()` de un número flotante o complejo ahora es el mismo que su `repr()`. Anteriormente, el formulario `str()` era más corto, pero eso solo causaba confusión y ya no es necesario ahora que el más corto posible `repr()` se muestra por defecto:

```
>>> import math
>>> repr(math.pi)
'3.141592653589793'
>>> str(math.pi)
'3.141592653589793'
```

  (Propuesto e implementado por Mark Dickinson; bpo-9337 [https://bugs.python.org/issue?@action=redirect&bpo=9337].)

- Los objetos `memoryview` ahora tienen un método `release()` y también admiten el protocolo de administración de contexto. Esto permite la liberación oportuna de cualquier recurso que se haya adquirido al solicitar un búfer del objeto original.

```
>>> with memoryview(b'abcdefgh') as v:
...     print(v.tolist())
[97, 98, 99, 100, 101, 102, 103, 104]
```

(Añadido por Antoine Pitrou; bpo-9757 [https://bugs.python.org/issue?@action=redirect&bpo=9757].)

- Anteriormente, era ilegal eliminar un nombre del espacio de nombres local si aparece como una variable libre en un bloque anidado:

  ```
  def exterior(x):
  def interior():
  return x
  interior()
  del x
  ```

  Esto ahora está permitido. Recuerde que el objetivo de una cláusula **`except`** se borra, por lo que este código que solía funcionar con Python 2.6, lanza un **`SyntaxError`** con Python 3.1 y ahora funciona nuevamente:

  ```
  def f():
  def print_error():
  print(e)
  try:
  something
  except Exception as e:
  print_error()
  # "del e" implícito aquí
  ```

  (Ver bpo-4617 [https://bugs.python.org/issue?@action=redirect&bpo=4617].)

- Struct sequence types ahora son subclases de tuplas. Esto significa que las estructuras de C como las devueltas por **`os.stat()`**, **`time.gmtime()`** y **`sys.version_info`** ahora funcionan como named tuple y funcionan con funciones y métodos que esperan una tupla como argumento.

Este es un gran paso adelante para hacer que las estructuras de C sean tan flexibles como sus contrapartes de Python puro:

```
>>> import sys
>>> isinstance(sys.version_info, tuple)
True
>>> 'Version %d.%d.%d %s(%d)' % sys.version_info
'Version 3.2.0 final(0)'
```

(Sugerido por Arfrever Frehtes Taifersar Arahesis e implementado por Benjamin Peterson en bpo-8413 [https://bugs.python.org/issue?@action=redirect&bpo=8413].)

- Las *Warnings* ahora son más fáciles de controlar usando la variable de entorno **`PYTHONWARNINGS`** como alternativa al uso de `-W` en la línea de comando:

  ```
  $ export PYTHONWARNINGS='ignore::Advertencia en tie
  ```

  (Sugerido por Barry Warsaw e implementado por Philip Jenvey en bpo-7301 [https://bugs.python.org/issue?@action=redirect&bpo=7301].)

- Se ha agregado una nueva categoría de advertencia, **`ResourceWarning`**. Se emite cuando se detectan problemas potenciales con el consumo de recursos o la limpieza. Está silenciado de forma predeterminada en las versiones de lanzamiento normales, pero se puede habilitar a través de los medios proporcionados por el módulo **`warnings`** o en la línea de comandos.

  Se emite un **`ResourceWarning`** al apagar el intérprete si la lista **`gc.garbage`** no está vacía y, si está configurado **`gc.DEBUG_UNCOLLECTABLE`**, se imprimen todos los objetos que no se pueden recolectar. Esto tiene como objetivo que el programador sepa que su código contiene problemas de finalización de objetos.

  Un **`ResourceWarning`** también se emite cuando un file object se destruye sin haber sido cerrado explícitamente. Si bien el desasignador de dicho objeto asegura que cierra el recurso del sistema operativo subyacente (generalmente, un descriptor de archivo), la demora en desasignar el objeto podría producir varios problemas, especialmente en Windows. A continuación, se muestra un ejemplo de cómo habilitar la advertencia desde la línea de comando:

```
$ python -q -Wdefault
>>> f = open("foo", "wb")
>>> del f
__main__:1: ResourceWarning: archivo sin cerrar <_i
```

(Agregado por Antoine Pitrou y Georg Brandl en bpo-10093 [https://bugs.python.org/issue?@action=redirect&bpo=10093] y bpo-477863 [https://bugs.python.org/issue?@action=redirect&bpo=477863].)

- Los objetos `range` ahora admiten los métodos *index* y *count*. Esto es parte de un esfuerzo por lograr que más objetos implementen completamente los métodos `collections.Sequence` y abstract base class. Como resultado, el lenguaje tendrá una API más uniforme. Además, los objetos `range` ahora admiten segmentación e índices negativos, incluso con valores mayores que `sys.maxsize`. Esto hace que *range* sea más interoperable con las listas:

  ```
  >>> rango(0, 100, 2).count(10)
  1
  >>> rango(0, 100, 2).index(10)
  5
  >>> rango(0, 100, 2)[5]
  10
  >>> rango(0, 100, 2)[0:5]
  rango(0, 10, 2)
  ```

  (Contribuido por Daniel Stutzbach en bpo-9213 [https://bugs.python.org/issue?@action=redirect&bpo=9213], por Alexander Belopolsky en bpo-2690 [https://bugs.python.org/issue?@action=redirect&bpo=2690], y por Nick Coghlan en bpo-10889 [https://bugs.python.org/issue?@action=redirect&bpo=10889].)

- Se resucitó la función incorporada **`callable()`** de Py2.x. Proporciona una alternativa concisa y legible al uso de una abstract base class en una expresión como `isinstance(x, collections.Callable)`:

  ```
  >>> callable(max)
  True
  >>> callable(20)
  False
  ```

  (Ver bpo-10518 [https://bugs.python.org/issue?@action=redirect&bpo=10518].)

- El mecanismo de importación de Python ahora puede cargar módulos instalados en directorios con caracteres no ASCII en el nombre de la ruta. Esto resolvió un problema agravante con los directorios de inicio para los usuarios con caracteres no ASCII en sus nombres de usuario.

  > (Trabajo extenso requerido por Victor Stinner en bpo-9425 [https://bugs.python.org/issue?@action=redirect&bpo=9425].)

Módulos nuevos, mejorados y obsoletos

La biblioteca estándar de Python ha sido objeto de importantes esfuerzos de mantenimiento y mejoras de calidad.

La novedad más importante de Python 3.2 es que el paquete **`email`**, el módulo **`mailbox`** y los módulos **`nntplib`** ahora funcionan correctamente con el modelo bytes/texto en Python 3. Por primera vez, hay un manejo correcto de mensajes con codificaciones mixtas.

En toda la biblioteca estándar, se ha prestado más atención a las codificaciones y los problemas de texto frente a bytes. En particular, las interacciones con el sistema operativo ahora son más capaces de intercambiar datos que no son ASCII mediante la codificación MBCS de Windows, codificaciones con reconocimiento de configuración regional o UTF-8.

Otro logro importante es la adición de un soporte sustancialmente mejor para conexiones *SSL* y certificados de seguridad.

Además, más clases ahora implementan context manager para soportar una conveniente y confiable limpieza de recursos usando una declaración `with`.

email

La usabilidad del paquete `email` en Python 3 se ha solucionado principalmente gracias a los amplios esfuerzos de R. David Murray. El problema era que los correos electrónicos generalmente se leen y almacenan en forma de `bytes` en lugar de texto `str`, y pueden contener múltiples codificaciones dentro de un solo correo electrónico. Por lo tanto, el paquete email tuvo que ampliarse para analizar y generar mensajes de correo electrónico en formato de bytes.

- Nuevas funciones `message_from_bytes()` y `message_from_binary_file()`, y nuevas clases `BytesFeedParser` y `BytesParser` permiten mensajes de datos en binario para ser analizados (*parsed*) en objetos modelados.

- Dada la entrada de bytes al modelo, `get_payload()` decodificará por defecto un cuerpo de mensaje que tiene un *Content-Transfer-Encoding* de *8bit* usando el juego de caracteres especificado en el encabezados MIME y retorna la cadena de caracteres resultante.

- Dada la entrada de bytes al modelo, `Generator` convertirá los cuerpos de los mensajes que tengan un *Content-Transfer-Encoding* de *8bit* para tener en su lugar un *7bitContent-Transfer-Encoding*.

  Los encabezados con bytes no codificados que no sean ASCII se consideran **RFC 2047** [https://datatracker.ietf.org/doc/html/rfc2047.html]-codificados utilizando el juego de caracteres *unknown-8bit*.

- Una nueva clase `BytesGenerator` produce bytes como salida, conservando cualquier dato no ASCII sin cambios que estuviera presente en la entrada utilizada para construir el modelo, incluidos los cuerpos de los mensajes con un *Content-Transfer-Encoding* de *8bit*.

- La clase `smtplibSMTP` ahora acepta una cadena de bytes para el argumento *msg* para el método `sendmail()`, y un nuevo método, `send_message()` acepta un objeto `Message` y opcionalmente puede obtener las direcciones *from_addr* y *to_addrs* directamente del objeto.

(Propuesto e implementado por R. David Murray, bpo-4661 [https://bugs.python.org/issue?@action=redirect&bpo=4661] y bpo-10321 [https://bugs.python.org/issue?@action=redirect&bpo=10321].)

elementtree

El paquete `xml.etree.ElementTree` y su contraparte `xml.etree.cElementTree` se han actualizado a la versión 1.3.

Se han agregado varias nuevas y útiles funciones y métodos:

- `xml.etree.ElementTree.fromstringlist()` que crea un documento XML a partir de una secuencia de fragmentos
- `xml.etree.ElementTree.register_namespace()` para registrar un prefijo de espacio de nombres global
- `xml.etree.ElementTree.tostringlist()` para la representación de cadenas de caracteres, incluidas todas las sublistas
- `xml.etree.ElementTree.Element.extend()` para agregar una secuencia de cero o más elementos
- `xml.etree.ElementTree.Element.iterfind()` busca un elemento y subelementos
- `xml.etree.ElementTree.Element.itertext()` crea un iterador de texto sobre un elemento y sus subelementos
- `xml.etree.ElementTree.TreeBuilder.end()` cierra el elemento actual
- `xml.etree.ElementTree.TreeBuilder.doctype()` gestiona la declaración *doctype*

Dos métodos que han sido deprecados:

- **`xml.etree.ElementTree.getchildren()`** usa `list(elem)` en su lugar.
- **`xml.etree.ElementTree.getiterator()`** usa `Element.iter` en su lugar.

Para obtener más información sobre la actualización, consulte Introducing ElementTree [https://web.archive.org/web/20200703234532/http://effbot.org/zone/elementtree-13-intro.htm] en el sitio web de Fredrik Lundh.

(Contribución de Florent Xicluna y Fredrik Lundh, bpo-6472 [https://bugs.python.org/issue?@action=redirect&bpo=6472].)

functools

- El módulo **`functools`** incluye un nuevo decorador para almacenar en caché las llamadas a funciones. **`functools.lru_cache()`** puede guardar consultas repetidas en un recurso externo siempre que se espere que los resultados sean los mismos.

  Por ejemplo, agregar un decorador de almacenamiento en caché a una función de consulta de base de datos puede ahorrar accesos a la base de datos para búsquedas populares:

  ```
  >>> import functools
  >>> @functools.lru_cache(maxsize=300)
  ... def get_phone_number(name):
  ...     c = conn.cursor()
  ...     c.execute('SELECT phonenumber FROM phonelis
  ...     return c.fetchone()[0]

  >>> for name in user_requests:
  ...     get_phone_number(name)        # cached look
  ```

  Para ayudar a elegir un tamaño de caché efectivo, la función encapsulada está instrumentada para rastrear estadísticas de caché:

  ```
  >>> get_phone_number.cache_info()
  CacheInfo(hits=4805, misses=980, maxsize=300, currs
  ```

Si la tabla *phonelist* se actualiza, el contenido desactualizado de la caché se puede borrar con:

```
>>> get_phone_number.cache_clear()
```

(Contribuido por Raymond Hettinger e incorporando ideas de diseño de Jim Baker, Miki Tebeka y Nick Coghlan; consulte recipe 498245 [https://code.activestate.com/recipes/498245-lru-and-lfu-cache-decorators/], recipe 577479 [https://code.activestate.com/recipes/577479-simple-caching-decorator/], bpo-10586 [https://bugs.python.org/issue?@action=redirect&bpo=10586] y bpo-10593 [https://bugs.python.org/issue?@action=redirect&bpo=10593]).

- El decorador `functools.wraps()` ahora agrega un atributo **`__wrapped__`** que apunta a la función invocable original. Esto permite que se puedan introspeccionar las funciones encapsuladas. También copia `__annotations__` si está definido. Y ahora también omite con elegancia los atributos faltantes, como `__doc__`, que podrían no estar definidos para la función encapsulada.

  En el ejemplo anterior, la caché se puede eliminar recuperando la función original:

  ```
  >>> get_phone_number = get_phone_number.__wrapped__
  ```

  (Por Nick Coghlan y Terrence Cole; bpo-9567 [https://bugs.python.org/issue?@action=redirect&bpo=9567], bpo-3445 [https://bugs.python.org/issue?@action=redirect&bpo=3445], y bpo-8814 [https://bugs.python.org/issue?@action=redirect&bpo=8814].)

- Para ayudar a escribir clases con métodos de comparación enriquecidos, un nuevo decorador `functools.total_ordering()` usará métodos de igualdad y desigualdad existentes para completar los métodos restantes.

  Por ejemplo, suministrar *_eq_* y *_lt_* permitirá que `total_ordering()` complete *_le_*, *_gt_* and *_ge_*:

```
@total_ordering
clase Estudiante:
def __eq__(self, other):
return ((self.apellido.inferior(), self.nombre.infe
(other.apellido.inferior(), other.nombre.inferior()

def __lt__(self, other):
return ((self.apellido.inferior(), self.nombre.infe
(other.apellido.inferior(), other.nombre.inferior()
```

Con el decorador *total_ordering*, los métodos de comparación restantes se completan automáticamente.

(Contribución por Raymond Hettinger.)

- Para ayudar en la migración de programas desde Python 2, la función **`functools.cmp_to_key()`** convierte una función de comparación de estilo antiguo en la nueva key function:

  ```
  >>> # locale-aware sort order
  >>> sorted(iterable, key=cmp_to_key(locale.strcoll)
  ```

  Para obtener ejemplos de ordenación y un breve tutorial, consulte el tutorial Sorting HowTo [https://wiki.python.org/moin/HowTo/Sorting/].

  (Contribución por Raymond Hettinger.)

itertools

- El módulo **`itertools`** tiene una nueva función **`accumulate()`** modelada en el operador *scan* de APL y la función *accumulate* de Numpy:

  ```
  >>> from itertools import accumulate
  >>> list(accumulate([8, 2, 50]))
  [8, 10, 60]

  >>> prob_dist = [0.1, 0.4, 0.2, 0.3]
  >>> list(accumulate(prob_dist))      # cumulative p
  ```

```
[0.1, 0.5, 0.7, 1.0]
```

Para un ejemplo usando `accumulate()`, vea los ejemplos para el módulo aleatorio.

(Contribución de Raymond Hettinger e incorporando sugerencias de diseño de Mark Dickinson.)

collections

- La clase `collections.Counter` ahora tiene dos formas de resta in situ, el operador existente `-=` para saturación de resta [https://es.wikipedia.org/wiki/Aritm%C3%A9tica_de_saturaci%C3%B3n] y el nuevo método `subtract()` para la resta regular. El primero es adecuado para muticonjunto [https://es.wikipedia.org/wiki/Multiconjunto] que solo tienen recuentos positivos, y el último es más adecuado para casos de uso que permiten recuentos negativos:

  ```
  >>> from collections import Counter
  >>> tally = Counter(dogs=5, cats=3)
  >>> tally -= Counter(dogs=2, cats=8)    # saturatin
  >>> tally
  Counter({'dogs': 3})

  >>> tally = Counter(dogs=5, cats=3)
  >>> tally.subtract(dogs=2, cats=8)      # regular s
  >>> tally
  Counter({'dogs': 3, 'cats': -5})
  ```

  (Contribución por Raymond Hettinger.)

- La clase `collections.OrderedDict` tiene un nuevo método `move_to_end()` que toma una clave existente y mueve a la primera o última posición de la secuencia ordenada.

  El valor predeterminado es mover un elemento a la última posición. Esto es equivalente a renovar una entrada con `od[k] = od.pop(k)`.

Una operación de mover al final rápida es útil para volver a secuenciar entradas. Por ejemplo, se puede usar un diccionario ordenado para rastrear el orden de acceso a las entradas por antigüedad desde la más antigua hasta la más reciente.

```
>>> from collections import OrderedDict
>>> d = OrderedDict.fromkeys(['a', 'b', 'X', 'd', '
>>> list(d)
['a', 'b', 'X', 'd', 'e']
>>> d.move_to_end('X')
>>> list(d)
['a', 'b', 'd', 'e', 'X']
```

(Contribución por Raymond Hettinger.)

- La clase **`collections.deque`** desarrolló dos métodos nuevos **`count()`** y **`reverse()`** que los hacen más intercambiables con los objetos **`list`**:

```
>>> from collections import deque
>>> d = deque('simsalabim')
>>> d.count('s')
2
>>> d.reverse()
>>> d
deque(['m', 'i', 'b', 'a', 'l', 'a', 's', 'm', 'i',
```

(Contribución por Raymond Hettinger.)

threading

El módulo **`threading`** tiene una nueva clase **`Barrier`** de sincronización para hacer que varios hilos esperen hasta que todos hayan alcanzado un punto de barrera común. Las barreras son útiles para asegurarse de que una tarea con múltiples condiciones previas no se ejecute hasta que se completen todas las tareas predecesoras.

Las barreras pueden trabajar con un número arbitrario de hilos. Esto es una generalización de barrera informática [https://

es.wikipedia.org/wiki/Barrera_(inform%C3%A1tica)] que se define solo para dos hilos.

Implementado como una barrera cíclica de dos fases, los objetos `Barrier` son adecuados para su uso en bucles. Las fases separadas de *llenado* y *drenaje* aseguran que todos los hilos se liberen (drenan) antes de que cualquiera de ellos pueda retornar y volver a entrar en la barrera. La barrera se restablece completamente después de cada ciclo.

Ejemplo de uso de barreras:

```
de threading import Barrier, Thread

def get_votes(site):
ballots = conduct_election(site)
all_polls_closed.wait() # no contar hasta que se cierre
totals = summary(ballots)
publish(site, totals)

all_polls_closed = Barrier(len(sites))
for site in sites:
Thread(target=get_votes, args=(site,)).start()
```

En este ejemplo, la barrera impone una regla de que los votos no se pueden contar en ningún lugar de votación hasta que se cierren todas las urnas. Observe cómo una solución con una barrera es similar a una con `threading.Thread.join()`, pero los hilos se mantienen vivos y continúan trabajando (resumiendo las papeletas) después de que se cruza el punto de la barrera.

Si alguna de las tareas predecesoras puede bloquearse o retrasarse, se puede crear una barrera con un parámetro opcional *timeout*. Luego, si el período de tiempo de espera transcurre antes de que todas las tareas predecesoras alcancen el punto de barrera, se liberan todos los hilos en espera y se lanza una excepción `BrokenBarrierError`:

```
def get_votes(sitio):
votos = realizar_elecciones(sitio)
```

```
try:
all_polls_closed.wait(timeout=midnight - time.now())
except BrokenBarrierError:
lockbox = sellar_votos(votos)
queue.put(lockbox)
else:
totales = resumir(votos)
publish(sitio, totales)
```

En este ejemplo, la barrera impone una regla más sólida. Si algunos sitios electorales no terminan antes de la medianoche, la barrera expira y las boletas se sellan y se depositan en una cola para su posterior manejo.

Consulte Barrier Synchronization Patterns [https://osl.cs.illinois.edu/media/papers/karmani-2009-barrier_synchronization_pattern.pdf] para obtener más ejemplos de cómo se pueden usar las barreras en la computación paralela. Además, hay una explicación simple pero completa de las barreras en The Little Book of Semaphores [https://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf], *section 3.6*.

(Contribución de Kristján Valur Jónsson con una revisión de API de Jeffrey Yasskin en bpo-8777 [https://bugs.python.org/issue?@action=redirect&bpo=8777].)

datetime and time

- El módulo `datetime` tiene un nuevo tipo `timezone` que implementa la interfaz `tzinfo` retornando un desplazamiento fijo desde UTC y un nombre de zona horaria. Esto hace que sea más fácil crear objetos de fecha y hora que tengan en cuenta la zona horaria:

  ```
  >>> import datetime as dt

  >>> dt.datetime.now(dt.timezone.utc)
  datetime.datetime(2010, 12, 8, 21, 4, 2, 923754, tz

  >>> dt.datetime.strptime("01/01/2000 12:00 +0000",
  datetime.datetime(2000, 1, 1, 12, 0, tzinfo=datetim
  ```

- Además, los objetos **`timedelta`** ahora pueden ser multiplicados por objetos **`float`** y divididos por objetos **`float`** e **`int`**. Y los objetos **`timedelta`** ahora se pueden dividir entre sí.

- El método **`datetime.date.strftime()`** ya no está restringido a años posteriores a 1900. El nuevo rango de años admitido es de 1000 a 9999 inclusive.

- Siempre que se utiliza un año de dos dígitos en una tupla de tiempo, la interpretación se rige por **`time.accept2dyear`**. El valor predeterminado es `True`, lo que significa que, para un año de dos dígitos, el siglo se calcula de acuerdo con las reglas POSIX que rigen el formato strptime `%y`.

  A partir de Py3.2, el uso de la heurística de estimación del siglo emitirá un **`DeprecationWarning`**. En cambio, se recomienda que **`time.accept2dyear`** se configure en `False` para que se puedan usar rangos de fechas grandes sin necesidad de realizar conjeturas:

```
>>> import time, Warnings
>>> Warnings.resetwarnings() # eliminar los filtros

>>> time.accept2dyear = True # adivinar si 11 signi
>>> time.asctime((11, 1, 1, 12, 34, 56, 4, 1, 0))
Advertencia (del módulo de advertencias):
...
Advertencia de desuso: información del siglo supues
'Fri Jan 1 12:34:56 2011'

>>> time.accept2dyear = False # usar el rango compl
>>> time.asctime((11, 1, 1, 12, 34, 56, 4, 1, 0))
'Fri Jan 1 12:34:56 11'
```

  Varias funciones ahora tienen rangos de fechas significativamente ampliados. Cuando **`time.accept2dyear`** es falso, la función **`time.asctime()`** aceptará cualquier año que quepa en un int de C, mientras que las funciones **`time.mktime()`** y

`time.strftime()` aceptarán el rango completo admitido por las funciones del sistema operativo correspondientes.

(Contribución de Alexander Belopolsky y Victor Stinner en bpo-1289118 [https://bugs.python.org/issue?@action=redirect&bpo=1289118], bpo-5094 [https://bugs.python.org/issue?@action=redirect&bpo=5094], bpo-6641 [https://bugs.python.org/issue?@action=redirect&bpo=6641], bpo-2706 [https://bugs.python.org/issue?@action=redirect&bpo=2706], bpo-1777412 [https://bugs.python.org/issue?@action=redirect&bpo=1777412], bpo-8013 [https://bugs.python.org/issue?@action=redirect&bpo=8013], y bpo-10827 [https://bugs.python.org/issue?@action=redirect&bpo=10827].)

math

El módulo **`math`** se ha actualizado con seis nuevas funciones inspiradas en el estándar C99.

La función **`isfinite()`** proporciona una forma rápida y fiable de detectar valores especiales. Retorna `True` para números regulares y `False` para *Nan* o *Infinity*:

```
>>> from math import isfinite
>>> [isfinite(x) for x in (123, 4.56, float('Nan'), floa
[True, True, False, False]
```

La función **`expm1()`** calcula `e**x-1` para valores pequeños de *x* sin incurrir en la pérdida de precisión que generalmente acompaña a la resta de cantidades casi iguales:

```
>>> from math import expm1
>>> expm1(0.013671875)   # more accurate way to compute
0.013765762467652909
```

La función **`erf()`** calcula una integral de probabilidad o función de error gaussiano [https://es.wikipedia.org/wiki/Funci%C3%B3n_error]. La función de error complementaria, **`erfc()`**, es `1 - erf(x)`:

```
>>> from math import erf, erfc, sqrt
>>> erf(1.0/sqrt(2.0)) # porción de distribución normal
0.682689492137086
```

```
>>> erfc(1.0/sqrt(2.0)) # porción de distribución normal
0.31731050786291404
>>> erf(1.0/sqrt(2.0)) + erfc(1.0/sqrt(2.0))
1.0
```

La función `gamma()` es una extensión continua de la función factorial. Consulte https://es.wikipedia.org/wiki/Funci%C3%B3n_gamma para obtener más detalles. Debido a que la función está relacionada con factoriales, crece incluso para valores pequeños de *x*, por lo que también hay una función `lgamma()` para calcular el logaritmo natural de la función gamma:

```
>>> from math import gamma, lgamma
>>> gamma(7.0)           # six factorial
720.0
>>> lgamma(801.0)        # log(800 factorial)
4551.950730698041
```

(Contribución de Mark Dickinson.)

abc

El módulo `abc` ahora admite `abstractclassmethod()` y `abstractstaticmethod()`.

Estas herramientas permiten definir una abstract base class que requiere un `classmethod()` o `staticmethod()` particular para ser implementado:

```
clase Temperatura(metaclase=abc.ABCMeta):
@abc.abstractclassmethod
def from_fahrenheit(cls, t):
...
@abc.abstractclassmethod
def from_celsius(cls, t):
...
```

(Parche enviado por Daniel Urban; bpo-5867 [https://bugs.python.org/issue?@action=redirect&bpo=5867].)

io

La clase `io.BytesIO` tiene un nuevo método, `getbuffer()`, que proporciona una funcionalidad similar a `memoryview()`. Crea una vista editable de los datos sin hacer una copia. El acceso aleatorio del búfer y la compatibilidad con la notación de sectores son adecuados para la edición in situ:

```
>>> REC_LEN, LOC_START, LOC_LEN = 34, 7, 11

>>> def change_location(buffer, record_number, location)
... start = record_number * REC_LEN + LOC_START
... buffer[start: start+LOC_LEN] = location

>>> import io

>>> byte_stream = io.BytesIO(
... b'G3805 almacén Chasis principal '
... b'X7899 envío Piñón de reserva '
... b'L6988 recepción Piñón primario'
... )
>>> buffer = byte_stream.getbuffer()
>>> change_location(buffer, 1, b'almacén ')
>>> change_location(buffer, 0, b'sala de exposición ')
>>> print(byte_stream.getvalue())
b'G3805 sala de exposición Chasis principal '
b'X7899 piñón de reserva de almacén '
b'L6988 piñón de recepción primario'
```

(Contribución de Antoine Pitrou en bpo-5506 [https://bugs.python.org/issue?@action=redirect&bpo=5506].)

reprlib

Al escribir un método `__repr__()` para un contenedor personalizado, es fácil olvidarse de manejar el caso en el que un miembro hace referencia al propio contenedor. Los objetos integrados de Python, como `list` y `set`, manejan la autorreferencia mostrando «…» en la parte recursiva de la cadena de representación.

Para ayudar a escribir dichos métodos `__repr__()`, el módulo `reprlib` tiene un nuevo decorador, `recursive_repr()`, para detectar llamadas recursivas a `__repr__()` y sustituir una cadena de marcador de posición en su lugar:

```
>>> clase MiLista(lista):
... @recursive_repr()
... def __repr__(self):
... return '<' + '|'.join(map(repr, self)) + ">'
...
>>> m = MiLista('abc')
>>> m.append(m)
>>> m.append('x')
>>> print(m)
<'a'|'b'|'c'|...|'x'>
```

(Contribución de Raymond Hettinger en bpo-9826 [https://bugs.python.org/issue?@action=redirect&bpo=9826] y bpo-9840 [https://bugs.python.org/issue?@action=redirect&bpo=9840].)

logging

Además de la configuración basada en diccionario descrita anteriormente, el paquete `logging` tiene muchas otras mejoras.

La documentación de logging se ha aumentado con un tutorial básico, un tutorial avanzado, y un libro de recetas. Estos documentos son la forma más rápida de aprender sobre logging.

La función de configuración `logging.basicConfig()` adquirió un argumento *style* para admitir tres tipos diferentes de formato de cadena de caracteres. Su valor predeterminado es «%» para el formato % tradicional, se puede establecer en «{» para el nuevo estilo `str.format()`, o se puede establecer en «$» para el formato de estilo shell proporcionado por `string.Template`. Las siguientes tres configuraciones son equivalentes:

```
>>> desde el registro, importe basicConfig
>>> basicConfig(style='%', format="%(nombre)s -> %(nombr
>>> basicConfig(style='{', format="{nombre} -> {nombreni
>>> basicConfig(style='$', format="$nombre -> $nombreniv
```

Si no se configura ninguna configuración antes de que se produzca un evento de registro, ahora hay una configuración predeterminada que utiliza un `StreamHandler` dirigido a `sys.stderr` para eventos de nivel `WARNING` o superior. Anteriormente, un evento que se producía antes de que se estableciera una configuración generaba una excepción o descartaba el evento de forma silenciosa según el valor de `logging.raiseExceptions`. El nuevo controlador predeterminado se almacena en `logging.lastResort`.

El uso de filtros ha sido simplificado. En lugar de crear un objeto `Filter`, la declaración puede ser cualquier invocable Python que retorna `True` o `False`.

Hubo una serie de otras mejoras que agregan flexibilidad y simplifican la configuración. Consulte la documentación del módulo para obtener una lista completa de los cambios en Python 3.2.

csv

El módulo `csv` ahora admite un nuevo dialecto, `unix_dialect`, que aplica comillas para todos los campos y un estilo Unix tradicional con `'\n'` como terminador de línea. El nombre del dialecto registrado es `unix`.

La clase `csv.DictWriter` tiene un nuevo método, `writeheader()` para escribir una fila inicial para documentar los nombres de los campos:

```
>>> import csv, sys
>>> w = csv.DictWriter(sys.stdout, ['nombre', 'dept'], d
>>> w.writeheader()
"nombre", "dept"
>>> w.writerows([
... {'nombre': 'tom', 'dept': 'contabilidad'},
... {'nombre': 'susan', 'dept': 'Ventas'}])
```

```
"tom", "contabilidad"
"susan", "ventas"
```

(Nuevo dialecto sugerido por Jay Talbot en bpo-5975 [https://bugs.python.org/issue?@action=redirect&bpo=5975], y el nuevo método sugerido por Ed Abraham en bpo-1537721 [https://bugs.python.org/issue?@action=redirect&bpo=1537721].)

contextlib

Hay una nueva y ligeramente alucinante herramienta **`ContextDecorator`** que es útil para crear un context manager que cumple una doble función como decorador de funciones.

Para su comodidad, esta nueva funcionalidad es utilizada por **`contextmanager()`** de modo que no se necesita ningún esfuerzo adicional para admitir ambos roles.

La idea básica es que tanto los gestores de contexto como los decoradores de funciones se pueden usar para envoltorios (*wrappers*) previos y posteriores a la acción. Los gestores de contexto envuelven un grupo de declaraciones usando la declaración **`with`** y los decoradores de funciones envuelven un grupo de declaraciones encerradas en una función. Por lo tanto, ocasionalmente es necesario escribir un envoltorio de acción previa o posterior que se pueda usar en cualquiera de los roles.

Por ejemplo, a veces es útil envolver funciones o grupos de declaraciones con un logger que pueda rastrear el tiempo de entrada y el tiempo de salida. En lugar de escribir tanto un decorador de funciones como un gestor de contexto para la tarea, **`contextmanager()`** proporciona ambas capacidades en una sola definición:

```
de contextlib import contextmanager
import logging

logging.basicConfig(level=logging.INFO)

@contextmanager
```

```
def track_entry_and_exit(name):
logging.info('Entrando: %s', nombre)
yield
logging.info('Saliendo: %s', nombre)
```

Anteriormente, esto solo se podía usar como gestor de contexto:

```
con track_entry_and_exit('cargador de widgets'):
print('Aquí se realiza una actividad que consume mucho t
load_widget()
```

Ahora, también se puede usar como decorador:

```
@track_entry_and_exit('cargador de widgets')
def activity():
print('Aquí se realiza una actividad que consume mucho t
load_widget()
```

Tratar de cumplir dos roles a la vez impone algunas limitaciones a la técnica. Los gestores de contexto normalmente tienen la flexibilidad de retornar un argumento utilizable por una instrucción **`with`**, pero no hay paralelo para los decoradores de funciones.

En el ejemplo anterior, no hay una forma limpia para que el gestor de contexto *track_entry_and_exit* retorne una instancia de logging para usar en el cuerpo de las declaraciones adjuntas.

(Contribución de Michael Foord en bpo-9110 [https://bugs.python.org/issue?@action=redirect&bpo=9110].)

decimal and fractions

Mark Dickinson diseñó un esquema elegante y eficiente para asegurar que diferentes tipos de datos numéricos tendrán el mismo valor hash siempre que sus valores reales sean iguales (bpo-8188 [https://bugs.python.org/issue?@action=redirect&bpo=8188]):

```
afirmar hash(Fracción(3, 2)) == hash(1.5) == \
hash(Decimal("1.5")) == hash(complejo(1.5, 0))
```

Algunos de los detalles del hash se exponen a través de un nuevo atributo, **`sys.hash_info`**, que describe el ancho de bits del valor hash, el módulo primo, los valores hash para *infinity* y *nan*, y el multiplicador utilizado para la parte imaginaria de un número:

```
>>> sys.hash_info
sys.hash_info(width=64, modulus=2305843009213693951, inf
```

Se ha relajado una decisión temprana de limitar la interoperabilidad de varios tipos numéricos. Todavía no se admite (y no es aconsejable) la mezcla implícita en expresiones aritméticas como `Decimal('1.1') + float('1.1')` porque esta última pierde información en el proceso de construcción del flotante binario. Sin embargo, dado que los valores de punto flotante existentes se pueden convertir sin pérdida a una representación decimal o racional, tiene sentido agregarlos al constructor y admitir comparaciones de tipos mixtos.

- El constructor **`decimal.Decimal`** ahora acepta objetos **`float`** directamente, por lo que ya no es necesario utilizar el método **`from_float()`** (bpo-8257 [https://bugs.python.org/issue?@action=redirect&bpo=8257]).
- Las comparaciones de tipos mixto ahora son totalmente compatibles, de modo que los objetos **`Decimal`** se pueden comparar directamente con **`float`** y **`fractions.Fraction`** (bpo-2531 [https://bugs.python.org/issue?@action=redirect&bpo=2531] y bpo-8188 [https://bugs.python.org/issue?@action=redirect&bpo=8188]).

Se realizaron cambios similares en **`fractions.Fraction`**, por lo que los métodos **`from_float()`** y **`from_decimal()`** ya no son necesarios (bpo-8294 [https://bugs.python.org/issue?@action=redirect&bpo=8294]):

```
>>> from decimal import Decimal
>>> from fractions import Fraction
>>> Decimal(1.1)
Decimal('1.100000000000000088817841970012523233890533447
>>> Fraction(1.1)
Fraction(2476979795053773, 2251799813685248)
```

Otro cambio útil para el módulo **`decimal`** es que el atributo

`Context.clamp` ahora es público. Esto resulta útil para crear contextos que corresponden a los formatos de intercambio decimal especificados en IEEE 754 (consulte bpo-8540 [https://bugs.python.org/issue?@action=redirect&bpo=8540]).

(Contribución de Mark Dickinson y Raymond Hettinger.)

ftp

La clase `ftplib.FTP` ahora admite el protocolo de gestor de contexto para consumir incondicionalmente las excepciones `socket.error` y cerrar la conexión FTP cuando haya terminado:

```
>>> de ftplib import FTP
>>> con FTP("ftp1.at.proftpd.org") como ftp:
ftp.login()
ftp.dir()

'230 Inicio de sesión anónimo correcto, se aplican restr
dr-xr-xr-x 9 ftp ftp 154 6 de mayo 10:43 .
dr-xr-xr-x 9 ftp ftp 154 6 de mayo 10:43 ..
dr-xr-xr-x 5 ftp ftp 4096 6 de mayo 10:43 CentOS
dr-xr-xr-x 3 ftp ftp 18 Jul 10 2008 Fedora
```

Otros objetos similares a archivos como `mmap.mmap` y `fileinput.input()` también adquirieron gestores de contexto de cierre automático:

```
con fileinput.input(files=('log1.txt', 'log2.txt')) com
para línea en f:
proceso(línea)
```

(Contribución de Tarek Ziadé y Giampaolo Rodolà en bpo-4972 [https://bugs.python.org/issue?@action=redirect&bpo=4972], y por Georg Brandl en bpo-8046 [https://bugs.python.org/issue?@action=redirect&bpo=8046] y bpo-1286 [https://bugs.python.org/issue?@action=redirect&bpo=1286].)

La clase `FTP_TLS` ahora acepta un parámetro *context*, que es un objeto `ssl.SSLContext` que permite agrupar opciones de configuración SSL, certificados y claves privadas en una sola (y potencialmente de larga duración) estructura.

(Contribución de Giampaolo Rodolà; bpo-8806 [https://bugs.python.org/issue?@action=redirect&bpo=8806].)

popen

Las funciones **`os.popen()`** y **`subprocess.Popen()`** ahora admiten declaraciones **`with`** para el cierre automático de los descriptores de archivo.

(Contribución de Antoine Pitrou y Brian Curtin en bpo-7461 [https://bugs.python.org/issue?@action=redirect&bpo=7461] y bpo-10554 [https://bugs.python.org/issue?@action=redirect&bpo=10554].)

select

El módulo **`select`** ahora expone un nuevo atributo constante, **`PIPE_BUF`**, que proporciona la cantidad mínima de bytes que se garantiza que no se bloquearán cuando **`select.select()`** indica que una tubería está lista para escribir.

```
>>> import select
>>> select.PIPE_BUF
512
```

(Disponible en sistemas Unix. Parche de Sébastien Sablé en bpo-9862 [https://bugs.python.org/issue?@action=redirect&bpo=9862])

gzip y zipfile

`gzip.GzipFile` ahora implementa **`io.BufferedIOBase`** clase base abstracta (excepto para `truncate()`). También tiene un método **`peek()`** y admite objetos de archivo con rellenos de ceros y que no se pueden buscar.

El módulo **`gzip`** también obtiene las funciones **`compress()`** y **`decompress()`** para facilitar la compresión y descompresión en memoria. Tenga en cuenta que el texto debe codificarse como `bytes` antes de comprimir y descomprimir:

```
>>> import gzip
>>> s = 'Three shall be the number thou shalt count, '
>>> s += 'and the number of the counting shall be three'
>>> b = s.encode()                        # convert to u
>>> len(b)
89
>>> c = gzip.compress(b)
>>> len(c)
77
>>> gzip.decompress(c).decode()[:42]      # decompress a
'Three shall be the number thou shalt count'
```

(Contribución de Anand B. Pillai en bpo-3488 [https://bugs.python.org/issue?@action=redirect&bpo=3488]; y de Antoine Pitrou, Nir Aides y Brian Curtin en bpo-9962 [https://bugs.python.org/issue?@action=redirect&bpo=9962], bpo-1675951 [https://bugs.python.org/issue?@action=redirect&bpo=1675951], bpo-7471 [https://bugs.python.org/issue?@action=redirect&bpo=7471] y bpo-2846 [https://bugs.python.org/issue?@action=redirect&bpo=2846].)

Además, la clase `zipfile.ZipExtFile` se modificó internamente para representar archivos almacenados dentro de un archivo. La nueva implementación es significativamente más rápida y se puede incluir en un objeto `io.BufferedReader` para lograr más velocidad. También resuelve un problema en el que las llamadas intercaladas a *read* y *readline* arrojaban resultados incorrectos.

(Parche enviado por Nir Aides en bpo-7610 [https://bugs.python.org/issue?@action=redirect&bpo=7610].)

tarfile

La clase `TarFile` ahora se puede usar como gestor de contexto. Además, su método `add()` tiene una nueva opción, *filter*, que controla qué archivos se agregan y permite editar los metadatos del archivo.

La nueva opción *filter* reemplaza el parámetro anterior y menos flexible *exclude* que ahora está deprecado. Si se especifica, el parámetro opcional *filter* debe ser un keyword argument. La función *filter* proporcionada por el usuario acepta un objeto **`TarInfo`** y retorna un objeto **`TarInfo`** actualizado, o si desea que el archivo sea excluido, la función puede retornar `None`:

```
>>> import tarfile, glob

>>> def myfilter(tarinfo):
... if tarinfo.isfile(): # solo guardar archivos reales
... tarinfo.uname = 'monty' # redactar el nombre de usua
... return tarinfo

>>> with tarfile.open(name='myarchive.tar.gz', mode='w:g
... for filename in glob.glob('*.txt'):
... tf.add(filename, filter=myfilter)
... tf.list()
-rw-r--r-- monty/501 902 2011-01-26 17:59:11 annotations
-rw-r--r-- monty/501 123 2011-01-26 17:59:11 preguntas_g
-rw-r--r-- monty/501 3514 2011-01-26 17:59:11 prion.txt
-rw-r--r-- monty/501 124 2011-01-26 17:59:11 py_todo.txt
-rw-r--r-- monty/501 1399 2011-01-26 17:59:11 notas_semá
```

(Propuesto por Tarek Ziadé e implementado por Lars Gustäbel en bpo-6856 [https://bugs.python.org/issue?@action=redirect&bpo=6856].)

hashlib

El módulo **`hashlib`** tiene dos nuevos atributos constantes que enumeran los algoritmos hash garantizados para estar presentes en todas las implementaciones y los disponibles en la implementación actual:

```
>>> importar hashlib

>>> hashlib.algoritmos_garantizados
{'sha1', 'sha224', 'sha384', 'sha256', 'sha512', 'md5'}

>>> hashlib.algoritmos_disponibles
```

```
{'md2', 'SHA256', 'SHA512', 'dsaWithSHA', 'mdc2', 'SHA22
'sha512', 'ripemd160', 'SHA1', 'MDC2', 'SHA', 'SHA384',
'ecdsa-with-SHA1','md4', 'md5', 'sha1', 'DSA-SHA', 'sha2
'dsaEncryption', 'DSA', 'RIPEMD160', 'sha', 'MD5', 'sha3
```

(Sugerido por Carl Chenet en bpo-7418 [https://bugs.python.org/issue?@action=redirect&bpo=7418].)

ast

El módulo `ast` tiene una maravillosa herramienta de propósito general para evaluar cadenas de expresión de forma segura utilizando la sintaxis literal de Python. La función `ast.literal_eval()` sirve como una alternativa segura a la función incorporada `eval()` la cual se abusa fácilmente. Python 3.2 agrega `bytes` y `set` literales a la lista de tipos admitidos: cadenas de caracteres, bytes, números, tuplas, listas, diccionarios, sets, valores booleanos y `None`.

```
>>> from ast import literal_eval

>>> request = "{'req': 3, 'func': 'pow', 'args': (2, 0.5
>>> literal_eval(request)
{'args': (2, 0.5), 'req': 3, 'func': 'pow'}

>>> request = "os.system('do something dangerous')"
>>> literal_eval(request)
Traceback (última llamada más reciente):
...
ValueError: nodo o cadena con formato incorrecto: <_ast.
```

(Implementado por Benjamin Peterson y Georg Brandl.)

os

Los diferentes sistemas operativos utilizan varias codificaciones para los nombres de archivo y las variables de entorno. El módulo `os` proporciona dos funciones nuevas, `fsencode()` y `fsdecode()`, para codificar y decodificar nombres de archivo:

```
>>> import os
>>> filename = 'Sehenswürdigkeiten'
>>> os.fsencode(filename)
b'Sehensw\xc3\xbcrdigkeiten'
```

Algunos sistemas operativos permiten el acceso directo a bytes codificados en el entorno. En ese caso, la constante `os.supports_bytes_environ` será verdadera.

Para acceder directamente a las variables de entorno codificadas (si está disponible), utilice la nueva función `os.getenvb()` o utilice `os.environb`, que es una versión en bytes de `os.environ`.

(Contribución de Victor Stinner.)

shutil

La función `shutil.copytree()` tiene dos nuevas opciones:

- *ignore_dangling_symlinks*: cuando `symlinks=False` para que la función copie un archivo apuntado por un enlace simbólico, no el enlace simbólico en sí. Esta opción silenciará el error lanzado si el archivo no existe.
- *copy_function*: es un invocable que se utilizará para copiar archivos. `shutil.copy2()` se usa por defecto.

(Contribución de Tarek Ziadé.)

Además, el módulo `shutil` ahora admite operaciones de archivado para archivos zip, archivos tar sin comprimir, archivos tar comprimidos con gzip y archivos tar bzip. Y hay funciones para registrar formatos de archivado de archivo adicionales (como archivos tar comprimidos xz o formatos personalizados).

Las funciones principales son `make_archive()` y `unpack_archive()`. Por defecto, ambas operan en el directorio actual (que puede ser configurado por `os.chdir()`) y en cualquier subdirectorio. El nombre del archivo de almacenamiento debe especificarse con una ruta completa. El paso de archivado no es destructivo (los archivos originales no se modifican).

```
>>> import shutil, pprint

>>> os.chdir('mydata') # cambiar al directorio de origen
>>> f = shutil.make_archive('/var/backup/mydata',
... 'zip') # archivar el directorio actual
>>> f # mostrar el nombre del archivo
'/var/backup/mydata.zip'
>>> os.chdir('tmp') # cambiar a un desempaquetado
>>> shutil.unpack_archive('/var/backup/mydata.zip') # re

>>> pprint.pprint(shutil.get_archive_formats()) # mostra
[('bztar', "archivo tar comprimido con bzip2"),
('gztar', "archivo tar comprimido con gzip"),
('tar', 'archivo tar sin comprimir'),
('zip', 'ZIP file')]

>>> shutil.register_archive_format( # registrar un nuevo
... name='xz',
... function=xz.compress, # función de archivado invocab
... extra_args=[('level', 8)], # argumentos para la func
... description='xz compression'
... )
```

(Contribución de Tarek Ziadé.)

sqlite3

El módulo `sqlite3` se actualizó a la versión 2.6.0 de pysqlite. Tiene dos nuevas capacidades.

- El atributo **`sqlite3.Connection.in_transit`** es verdadero si hay una transacción activa para cambios no confirmados.
- Los métodos `sqlite3.Connection.enable_load_extension()` y `sqlite3.Connection.load_extension()` le permiten cargar extensiones SQLite desde archivos «.so». Una extensión conocida es la extensión de búsqueda de texto completo distribuida con SQLite.

(Contribución de R. David Murray y Shashwat Anand; bpo-8845 [https://bugs.python.org/issue?@action=redirect&bpo=8845].)

html

Se introdujo un nuevo módulo `html` con una sola función, `escape()`, que se usa para escapar caracteres reservados del marcado HTML:

```
>>> import html
>>> html.escape('x > 2 && x < 7')
'x &gt; 2 &amp;&amp; x &lt; 7'
```

socket

El módulo `socket` tiene dos nuevas mejoras.

- Los objetos de socket ahora tienen un método `detach()` que pone el socket en estado cerrado sin cerrar realmente el descriptor de archivo subyacente. Este último puede reutilizarse para otros fines. (Agregado por Antoine Pitrou; bpo-8524 [https://bugs.python.org/issue?@action=redirect&bpo=8524].)
- `socket.create_connection()` ahora admite el protocolo de administración de contexto para consumir incondicionalmente las excepciones `socket.error` y cerrar el socket cuando haya terminado. (Contribución de Giampaolo Rodolà; bpo-9794 [https://bugs.python.org/issue?@action=redirect&bpo=9794].)

ssl

El módulo `ssl` agregó una serie de características para satisfacer los requisitos comunes para conexiones a Internet seguras (encriptadas, autenticadas):

- Una nueva clase, `SSLContext`, sirve como contenedor para datos SSL persistentes, como configuraciones de protocolo, certificados, claves privadas y varias opciones más. Incluye un `wrap_socket()` para crear un socket SSL a partir de un contexto SSL.

- Una nueva función, `ssl.match_hostname()`, admite la verificación de identidad del servidor para protocolos de nivel superior mediante la implementación de las reglas de HTTPS (de RFC 2818 [https://datatracker.ietf.org/doc/html/rfc2818.html]) que también son adecuadas para otros protocolos.
- La función constructora `ssl.wrap_socket()` ahora acepta un argumento *ciphers*. La cadena *ciphers* enumera los algoritmos de cifrado permitidos utilizando el formato descrito en OpenSSL documentation [https://docs.openssl.org/1.0.2/man1/ciphers/#cipher-list-format].
- Cuando se vincula con versiones recientes de OpenSSL, el módulo `ssl` ahora admite la extensión de Server Name Indication (SNI, en español: Indicador del nombre del servidor) para el protocolo TLS, lo que permite que varios» hosts virtuales «utilicen diferentes certificados en un solo puerto IP. Esta extensión solo es compatible en modo cliente y se activa pasando el argumento *server_hostname* a `ssl.SSLContext.wrap_socket()`.
- Se han agregado varias opciones al módulo `ssl`, como `OP_NO_SSLv2` que deshabilita el protocolo SSLv2 inseguro y obsoleto.
- La extensión ahora carga todos los algoritmos de resumen y cifrados OpenSSL. Si algunos certificados SSL no se pueden verificar, se notifican como un error de «algoritmo desconocido».
- La versión de OpenSSL que se está utilizando ahora es accesible mediante los atributos del módulo `ssl.OPENSSL_VERSION` (una cadena), `ssl.OPENSSL_VERSION_INFO` (una tupla de 5) y `ssl.OPENSSL_VERSION_NUMBER` (un entero).

(Contribución de Antoine Pitrou en bpo-8850 [https://bugs.python.org/issue?@action=redirect&bpo=8850], bpo-1589 [https://bugs.python.org/issue?@action=redirect&bpo=1589], bpo-8322 [https://bugs.python.org/issue?@action=redirect&bpo=8322], bpo-5639 [https://bugs.python.org/issue?@action=redirect&bpo=5639], bpo-4870 [https://bugs.python.org/issue?@action=redirect&bpo=4870], bpo-8484 [https://bugs.python.org/issue?@action=redirect&bpo=8484], y bpo-8321 [https://bugs.python.org/issue?@action=redirect&bpo=8321].)

nntp

El módulo `nntplib` tiene una implementación renovada con una mejor semántica de bytes y texto, así como API más prácticas. Estas mejoras rompen la compatibilidad con la versión nntplib en Python 3.1, que en parte era disfuncional en sí misma.

También se ha agregado soporte para conexiones seguras a través de TLS implícito (usando `nntplib.NNTP_SSL`) y explícito (usando `nntplib.NNTP.starttls()`).

(Contribución de Antoine Pitrou en bpo-9360 [https://bugs.python.org/issue?@action=redirect&bpo=9360] y Andrew Vant en bpo-1926 [https://bugs.python.org/issue?@action=redirect&bpo=1926].)

certificados

`http.client.HTTPSConnection`, `urllib.request.HTTPSHandler` y `urllib.request.urlopen()` ahora toman argumentos opcionales para permitir la comprobación del certificado del servidor con un conjunto de Autoridades de Certificación recomendado en usos públicos de HTTPS.

(Agregado por Antoine Pitrou, bpo-9003 [https://bugs.python.org/issue?@action=redirect&bpo=9003].)

imaplib

Se ha agregado soporte para TLS explícito en conexiones IMAP4 estándar a través del nuevo método `imaplib.IMAP4.starttls`.

(Contribución de Lorenzo M. Catucci y Antoine Pitrou, bpo-4471 [https://bugs.python.org/issue?@action=redirect&bpo=4471].)

http.client

Hubo una serie de pequeñas mejoras de API en el módulo `http.client`. Las respuestas simples HTTP 0.9 de estilo antiguo ya no son compatibles y el parámetro *strict* está obsoleto en todas las clases.

Las clases `HTTPConnection` y `HTTPSConnection` ahora tienen un parámetro *source_address* para una tupla (host, port) que indica desde dónde se realiza la conexión HTTP.

Se agregó soporte para verificación de certificados y hosts virtuales HTTPS a `HTTPSConnection`.

El método `request()` en los objetos de conexión permitía un argumento *body* opcional para que un file object pudiera usarse para proporcionar el contenido de la solicitud. Convenientemente, el argumento *body* ahora también acepta un objeto iterable siempre que incluya un encabezado explícito `Content-Length`. Esta interfaz ampliada es mucho más flexible que antes.

Para establecer una conexión HTTPS a través de un servidor proxy, existe un nuevo método `set_tunnel()` que establece el host y el puerto para el túnel HTTP Connect.

Para que coincida con el comportamiento de `http.server`, la biblioteca de cliente HTTP ahora también codifica encabezados con codificación ISO-8859-1 (Latin-1). Ya lo estaba haciendo para los encabezados entrantes, por lo que ahora el comportamiento es coherente para el tráfico entrante y saliente. (Véase el trabajo de Armin Ronacher en bpo-10980 [https://bugs.python.org/issue?@action=redirect&bpo=10980])

unittest

El módulo unittest tiene una serie de mejoras con soporte para descubrimiento de pruebas para paquetes, una experimentación más sencilla en el indicador interactivo, nuevos métodos de casos de prueba, mensajes de diagnóstico mejorados para fallas en las pruebas y mejores nombres de métodos.

- La llamada de línea de comandos `python -m unittest` ahora puede aceptar rutas de archivo en lugar de nombres de módulos para ejecutar pruebas específicas (bpo-10620 [https://bugs.python.org/issue?@action=redirect&bpo=10620]). El nuevo descubrimiento de pruebas puede encontrar pruebas dentro de los paquetes, localizando cualquier prueba que se pueda importar desde el directorio de nivel superior. El directorio de nivel superior se puede especificar con la opción `-t`, un patrón para hacer coincidir archivos con `-p` y un directorio para iniciar el descubrimiento con `-s`:

```
$ python -m unittest discover -s mi_directorio_del_
```

(Contribución de Michael Foord.)

- La experimentación en el indicador interactivo ahora es más sencilla porque la clase `unittest.TestCase` ahora se puede instanciar sin argumentos:

  ```
  >>> from unittest import TestCase
  >>> TestCase().assertEqual(pow(2, 3), 8)
  ```

  (Contribución de Michael Foord.)

- El módulo `unittest` tiene dos métodos nuevos, `assertWarns()` y `assertWarnsRegex()` para verificar que un tipo de advertencia dado es activado por el código bajo prueba:

  ```
  con self.assertWarns(Advertencia de desuso):
  legacy_function('XYZ')
  ```

  (Contribución de Antoine Pitrou, bpo-9754 [https://bugs.python.org/issue?@action=redirect&bpo=9754].)

  Otro método nuevo, `assertCountEqual()` se usa para comparar dos iterables para determinar si sus conteos de elementos son iguales (si los mismos elementos están presentes con el mismo número de ocurrencias independientemente del orden):

  ```
  def test_anagram(self):
  self.assertCountEqual('algoritmo', 'logaritmo')
  ```

  (Contribución por Raymond Hettinger.)

- Una característica principal del módulo unittest es un esfuerzo por producir diagnósticos significativos cuando falla una prueba. Cuando es posible, la falla se registra junto con una diferencia de la salida. Esto es especialmente útil para analizar archivos de registro de ejecuciones de prueba fallidas. Sin embargo, dado que las diferencias en ocasiones pueden ser voluminosas, hay un nuevo atributo `maxDiff` que establece la longitud máxima de las diferencias mostradas.

- Además, los nombres de los métodos en el módulo se han sometido a una serie de limpiezas.

  Por ejemplo, `assertRegex()` es el nuevo nombre de **`assertRegexpMatches()`**, que se nombró incorrectamente porque la prueba usa `re.search()`, no `re.match()`. Otros métodos que usan expresiones regulares ahora se nombran usando la forma corta «Regex» en lugar de «Regexp»; esto coincide con los nombres usados en otras implementaciones de pruebas unitarias, coincide con el nombre antiguo de Python para el módulo `re` y tiene un uso inequívoco de mayúsculas y minúsculas.

  (Contribución de Raymond Hettinger e implementado por Ezio Melotti.)

- Para mejorar la coherencia, algunos alias de métodos antiguos se están deprecando en favor de los nombres preferidos:

Old Name	Preferred Name
`assert_()`	`assertTrue()`
`assertEquals()`	`assertEqual()`
`assertNotEquals()`	`assertNotEqual()`
`assertAlmostEquals()`	`assertAlmostEqual()`
`assertNotAlmostEquals()`	`assertNotAlmostEqual()`

  Asimismo, se espera que los métodos `TestCase.fail*` obsoletos en Python 3.1 se eliminen en Python 3.3.

  (Contribución de Ezio Melotti; bpo-9424 [https://bugs.python.org/issue?@action=redirect&bpo=9424].)

- El método **`assertDictContainsSubset()`** quedó obsoleto porque se implementó incorrectamente con los argumentos en el orden incorrecto. Esto generó ilusiones ópticas difíciles de depurar donde fallaban pruebas como `TestCase().assertDictContainsSubset({'a':1, 'b':2}, {'a':1})`.

  (Contribución por Raymond Hettinger.)

random

Los métodos enteros en el módulo **`random`** ahora hacen un mejor trabajo al producir distribuciones uniformes. Anteriormente, calculaban las selecciones con `int(n*random())` que tenía un ligero sesgo siempre que *n* no era una potencia de dos. Ahora, se realizan selecciones múltiples desde un rango hasta la siguiente potencia de dos y una selección se mantiene solo cuando cae dentro del rango `0 <= x < n`. Las funciones y métodos afectados son **`randrange()`**, **`randint()`**, **`choice()`**, **`shuffle()`** y **`sample()`**.

(Contribución de Raymond Hettinger; bpo-9025 [https://bugs.python.org/issue?@action=redirect&bpo=9025].)

poplib

`POP3_SSL` la clase ahora acepta un parámetro *context*, que es un objeto **`ssl.SSLContext`** que permite agrupar opciones de configuración SSL, certificados y claves privadas en una única estructura (potencialmente de larga duración) .

(Contribución de Giampaolo Rodolà; bpo-8807 [https://bugs.python.org/issue?@action=redirect&bpo=8807].)

asyncore

`asyncore.dispatcher` ahora proporciona un método **`handle_accepted()`** que devuelve un par `(sock, addr)` que se llama cuando se ha establecido una conexión con un nuevo punto final remoto. Se supone que esto se usa como reemplazo del antiguo

`handle_accept()` y evita que el usuario llame directamente a **`accept()`**.

(Contribución de Giampaolo Rodolà; bpo-6706 [https://bugs.python.org/issue?@action=redirect&bpo=6706].)

tempfile

El módulo **`tempfile`** tiene un nuevo gestor de contexto, **`TemporaryDirectory`** que proporciona una limpieza determinista fácil de directorios temporales

```
con tempfile.TemporaryDirectory() como tmpdirname:
print('directorio temporal creado:', tmpdirname)
```

(Contribución de Neil Schemenauer y Nick Coghlan; bpo-5178 [https://bugs.python.org/issue?@action=redirect&bpo=5178].)

inspect

- El módulo **`inspect`** tiene una nueva función **`getgeneratorstate()`** para identificar fácilmente el estado actual de un generador iterador:

  ```
  >>> de inspeccionar importar obtenergeneratorstate
  >>> def gen():
  ... rendimiento 'demo'
  ...
  >>> g = gen()
  >>> obtenergeneratorstate(g)
  'GEN_CREATED'
  >>> siguiente(g)
  'demo'
  >>> obtenergeneratorstate(g)
  'GEN_SUSPENDED'
  >>> siguiente(g, None)
  >>> obtenergeneratorstate(g)
  'GEN_CLOSED'
  ```

  (Contribución de Rodolpho Eckhardt y Nick Coghlan,

bpo-10220 [https://bugs.python.org/issue?@action=redirect&bpo=10220].)

- Para admitir búsquedas sin la posibilidad de activar un atributo dinámico, el módulo `inspect` tiene una nueva función, `getattr_static()`. A diferencia de `hasattr()`, esta es una verdadera búsqueda de solo lectura, garantizada que no cambiará de estado mientras está buscando:

```
>>> clase A:
... @property
... def f(self):
... print('En ejecución')
... return 10
...
>>> a = A()
>>> getattr(a, 'f')
En ejecución
10
>>> inspect.getattr_static(a, 'f')
<objeto de propiedad en 0x1022bd788>
```

  (Contribución de Michael Foord.)

pydoc

El módulo `pydoc` ahora proporciona una interfaz de servidor web muy mejorada, así como una nueva opción de línea de comandos `-b` para abrir automáticamente una ventana del navegador para mostrar ese servidor:

```
$pydoc3.2-b
```

(Contribución de Ron Adam; bpo-2001 [https://bugs.python.org/issue?@action=redirect&bpo=2001].)

dis

El módulo `dis` obtuvo dos nuevas funciones para inspeccionar código, `code_info()` y `show_code()`. Ambas proporcionan información detallada del objeto de código para la función, método, cadena de código fuente o objeto de código suministrados. El primero retorna una cadena de caracteres y el segundo la imprime:

```
>>> import dis, random
>>> dis.show_code(random.choice)
Nombre: choice
Nombre de archivo: /Library/Frameworks/Python.framework/
Número de argumentos: 2
Argumentos solo de Kw: 0
Número de variables locales: 3
Tamaño de pila: 11
Indicadores: OPTIMIZED, NEWLOCALS, NOFREE
Constantes:
0: 'Elija un elemento aleatorio de una secuencia no vací
1: 'No se puede elegir de una secuencia vacía'
Nombres:
0: _randbelow
1: len
2: ValueError
3: IndexError
Nombres de variables:
0: self
1: seq
2: i
```

Además, la función **`dis()`** ahora acepta argumentos de cadena para que el modismo común `dis(compile(s, '', 'eval'))` se pueda abreviar a `dis(s)`:

```
>>> dis('3*x+1 si x%2==1 sino x//2')
1 0 CARGAR_NOMBRE 0 (x)
3 CARGAR_CONST 0 (2)
6 MÓDULO_BINARIO
7 CARGAR_CONST 1 (1)
10 COMPARAR_OP 2 (==)
13 POP_SALTAR_SI_FALSO 28
16 CARGAR_CONST 2 (3)
19 CARGAR_NOMBRE 0 (x)
```

```
22 MULTIPLICACIÓN_BINARIA
23 CARGAR_CONST 1 (1)
26 SUMA_BINARIA
27 VALOR_DEVUELTO
>> 28 CARGAR_NOMBRE 0 (x)
31 CARGAR_CONST 0 (2)
34 DIVISIÓN_PISO_BINARIO
35 VALOR_DEVUELTO
```

En conjunto, estas mejoras facilitan explorar cómo se implementa CPython y ver por sí mismo qué hace la sintaxis del lenguaje bajo el capó.

(Contribución de Nick Coghlan en bpo-9147 [https://bugs.python.org/issue?@action=redirect&bpo=9147].)

dbm

Todos los módulos de base de datos ahora admiten los métodos **`get()`** y **`setdefault()`**.

(Sugerido por Ray Allen en bpo-9523 [https://bugs.python.org/issue?@action=redirect&bpo=9523].)

ctypes

Un nuevo tipo, **`ctypes.c_ssize_t`** representa el tipo de datos C **`ssize_t`**.

site

El módulo **`site`** tiene tres nuevas funciones útiles para informar sobre los detalles de una determinada instalación de Python.

- **`getsitepackages()`** enumera todos los directorios de paquetes de sitios globales.
- **`getuserbase()`** informa sobre el directorio base del usuario donde se pueden almacenar los datos.
- **`getusersitepackages()`** revela la ruta del directorio de paquetes de sitio específico del usuario.

```
>>> importar sitio
>>> site.getsitepackages()
['/Library/Frameworks/Python.framework/Versions/3.2/lib/
'/Library/Frameworks/Python.framework/Versions/3.2/lib/s
'/Library/Python/3.2/site-packages']
>>> site.getuserbase()
'/Usuarios/raymondhettinger/Library/Python/3.2'
>>> site.getusersitepackages()
'/Usuarios/raymondhettinger/Library/Python/3.2/lib/pytho
```

Convenientemente, se puede acceder a algunas de las funciones del sitio directamente desde la línea de comandos:

```
$ python -m site --user-base
/Usuarios/raymondhettinger/.local
$ python -m site --user-site
/Usuarios/raymondhettinger/.local/lib/python3.2/site-pac
```

(Contribución de Tarek Ziadé en bpo-6693 [https://bugs.python.org/issue?@action=redirect&bpo=6693].)

sysconfig

El nuevo módulo `sysconfig` hace que sea sencillo descubrir rutas de instalación y variables de configuración que varían entre plataformas e instalaciones.

El módulo ofrece acceso a funciones de acceso simple para información de plataforma y versión:

- `get_platform()` retornando valores como *linux-i586* o *macosx-10.6-ppc*.
- `get_python_version()` retorna una cadena de versión de Python como» 3.2 «.

También proporciona acceso a las rutas y variables correspondientes a uno de los siete esquemas con nombre utilizados por `distutils`, entre los que se incluyen *posix_prefix*, *posix_home*, *posix_user*, *nt*, *nt_user*, *os2* y *os2_home*:

- **`get_paths()`** crea un diccionario que contiene las rutas de instalación para el esquema de instalación actual.
- **`get_config_vars()`** retorna un diccionario de variables específicas de la plataforma.

También hay una conveniente interfaz de línea de comandos:

```
C:\Python32>python -m sysconfig
Plataforma: "win32"
Versión de Python: "3.2"
Esquema de instalación actual: "nt"

Rutas:
data = "C:\Python32"
include = "C:\Python32\Include"
platinclude = "C:\Python32\Include"
platlib = "C:\Python32\Lib\site-packages"
platstdlib = "C:\Python32\Lib"
purelib = "C:\Python32\Lib\site-packages"
scripts = "C:\Python32\Scripts"
stdlib = "C:\Python32\Lib"

Variables:
BINDIR = "C:\Python32"
BINLIBDEST = "C:\Python32\Lib"
EXE = ".exe"
INCLUDEPY = "C:\Python32\Include"
LIBDEST = "C:\Python32\Lib"
SO = ".pyd"
VERSION = "32"
abiflags = ""
base = "C:\Python32"
exec_prefix = "C:\Python32"
platbase = "C:\Python32"
prefix = "C:\Python32"
projectbase = "C:\Python32"
py_version = "3.2"
py_version_nodot = "32"
py_version_short = "3.2"
```

```
srcdir = "C:\Python32"
userbase = "C:\Documentos y configuraciones\Raymond\Dato
```

(Sacado de Distutils por Tarek Ziadé.)

pdb

El módulo depurador **`pdb`** obtuvo una serie de mejoras de usabilidad:

- `pdb.py` ahora tiene una opción `-c` que ejecuta comandos como se indica en un archivo de script `.pdbrc`.
- Un archivo de secuencia de comandos `.pdbrc` puede contener comandos `continue` y `next` que continúan depurando.
- El constructor de la clase **`Pdb`** ahora acepta un argumento *nosigint*.
- Nuevos comandos: `l(list)`, `ll(long list)` y `source` para enumerar el código fuente.
- Nuevos comandos: `display` y `undisplay` para mostrar u ocultar el valor de una expresión si ha cambiado.
- Nuevo comando: `interact` para iniciar un intérprete interactivo que contiene los nombres globales y locales que se encuentran en el alcance actual.
- Los puntos de interrupción se pueden borrar por número de punto de interrupción.

(Contribución de Georg Brandl, Antonio Cuni e Ilya Sandler.)

configparser

El módulo **`configparser`** se modificó para mejorar la usabilidad y la previsibilidad del analizador predeterminado y su sintaxis INI compatible. La antigua clase **`ConfigParser`** se eliminó en favor de **`SafeConfigParser`**, que a su vez se renombró como **`ConfigParser`**. La compatibilidad con comentarios en línea ahora está desactivada de manera predeterminada y no se permiten secciones u opciones duplicadas en una única fuente de configuración.

Los analizadores de configuración obtuvieron una nueva API basada en el protocolo de mapeo:

```
>>> parser = ConfigParser()
>>> parser.read_string("""
... [PREDETERMINADO]
... ubicación = esquina superior izquierda
... visible = sí
... editable = no
... color = azul
...
... [principal]
... título = Menú principal
... color = verde
...
... [opciones]
... título = Opciones
... """)
>>> parser['main']['color']
'verde'
>>> parser['main']['editable']
'no'
>>> sección = parser['opciones']
>>> sección['título']
'Opciones'
>>> sección['título'] = 'Opciones (editables: %(editable
>>> sección['título']
'Opciones (editables: no)'
```

La nueva API se implementa sobre la API clásica, por lo que las subclases de analizadores personalizados deberían poder usarla sin modificaciones.

La estructura de archivos INI aceptada por los analizadores de configuración ahora se puede personalizar. Los usuarios pueden especificar opciones alternativas/delimitadores de valores y prefijos de comentarios, cambiar el nombre de la sección *DEFAULT* o cambiar la sintaxis de interpolación.

Hay soporte para la interpolación conectable que incluye un controlador de interpolación adicional **`ExtendedInterpolation`**:

```
>>> parser = ConfigParser(interpolation=ExtendedInterpol
>>> parser.read_dict({'buildout': {'directorio': '/home/
... 'personalizado': {'prefijo': '/usr/local'}})
>>> parser.read_string("""
... [buildout]
... partes =
... zope9
... instancia
... enlaces-de-busqueda =
... ${buildout:directorio}/descargas/dist
...
... [zope9]
... receta = plone.recipe.zope9install
... ubicación = /opt/zope
...
... [instancia]
... receta = plone.recipe.zope9instance
... ubicación-zope9 = ${zope9:ubicación}
... configuración-zope = ${personalizado:prefijo}/etc/zo
... """)
>>> parser['buildout']['find-links']
'\n/home/ambv/zope9/downloads/dist'
>>> parser['instancia']['zope-conf']
'/usr/local/etc/zope.conf'
>>> instancia = parser['instancia']
>>> instancia['zope-conf']
'/usr/local/etc/zope.conf'
>>> instancia['zope9-location']
'/opt/zope'
```

También se introdujeron una serie de características más pequeñas, como soporte para especificar codificación en operaciones de lectura, especificar valores de respaldo para funciones de obtención o leer directamente desde diccionarios y cadenas.

(Todos los cambios aportados por Łukasz Langa.)

urllib.parse

Se realizaron varias mejoras de usabilidad para el módulo **`urllib.parse`**.

La función **`urlparse()`** ahora admite direcciones IPv6 [https://en.wikipedia.org/wiki/IPv6] como se describe en **RFC 2732** [https://datatracker.ietf.org/doc/html/rfc2732.html]:

```
>>> import urllib.parse
>>> urllib.parse.urlparse('http://[dead:beef:cafe:5417:a
ParseResult(scheme='http',
            netloc='[dead:beef:cafe:5417:affe:8FA3:deaf:
            path='/foo/',
            params='',
            query='',
            fragment='')
```

La función **`urldefrag()`** ahora devuelve un named tuple:

```
>>> r = urllib.parse.urldefrag('http://python.org/about/
>>> r
DefragResult(url='http://python.org/about/', fragment='t
>>> r[0]
'http://python.org/about/'
>>> r.fragment
'target'
```

Y, la función **`urlencode()`** ahora es mucho más flexible, aceptando un tipo de cadena o bytes para el argumento *query*. Si es una cadena, entonces los parámetros *safe*, *encoding* y *error* se envían a **`quote_plus()`** para codificar:

```
>>> urllib.parse.urlencode([
...  ('tipo', 'telenovela'),
...  ('nombre', '¿Dónde Está Elisa?')],
... codificación = 'latin-1')
'tipo=telenovela&nombre=%BFD%F3nde+Est%E1+Elisa%3F'
```

Como se detalla en Análisis de bytes codificados ASCII, todas las funciones `urllib.parse` ahora aceptan cadenas de bytes codificadas en ASCII como entrada, siempre que no estén mezcladas con cadenas normales. Si se proporcionan cadenas de bytes codificadas en ASCII como parámetros, los tipos de retorno también serán cadenas de bytes codificadas en ASCII:

```
>>> urllib.parse.urlparse(b'http://www.python.org:80/abo
ParseResultBytes(scheme=b'http', netloc=b'www.python.org
                 path=b'/about/', params=b'', query=b'',
```

(Trabajo de Nick Coghlan, Dan Mahn y Senthil Kumaran en bpo-2987 [https://bugs.python.org/issue?@action=redirect&bpo=2987], bpo-5468 [https://bugs.python.org/issue?@action=redirect&bpo=5468], y bpo-9873 [https://bugs.python.org/issue?@action=redirect&bpo=9873].)

mailbox

Gracias a un esfuerzo concertado de R. David Murray, el módulo `mailbox` se ha corregido para Python 3.2. El desafío era que el buzón había sido diseñado originalmente con una interfaz de texto, pero los mensajes de correo electrónico se representan mejor con `bytes` porque varias partes de un mensaje pueden tener diferentes codificaciones.

La solución aprovechó el soporte binario de los paquete `email` para analizar mensajes de correo electrónico arbitrarios. Además, la solución requirió una serie de cambios en la API.

Como era de esperar, el método `add()` para los objetos `mailbox.Mailbox` ahora acepta entrada binaria.

`StringIO` y la entrada del archivo de texto están en desuso. Además, la entrada de cadenas fallará antes si se utilizan caracteres que no sean ASCII. Anteriormente, fallaba cuando el correo electrónico se procesaba en un paso posterior.

También hay soporte para salida binaria. El método `get_file()` ahora devuelve un archivo en modo binario (donde solía configurar incorrectamente el archivo en modo texto). También hay un nuevo método `get_bytes()` que devuelve una representación `bytes` de un mensaje correspondiente a una *key* dada.

Todavía es posible obtener una salida no binaria usando el método de la API anterior `get_string()`, pero ese enfoque no es muy útil. En su lugar, es mejor extraer mensajes de un objeto `Message` o cargarlos desde una entrada binaria.

(Contribuido por R. David Murray, con esfuerzos de Steffen Daode Nurpmeso y un parche inicial de Victor Stinner en bpo-9124 [https://bugs.python.org/issue?@action=redirect&bpo=9124].)

turtledemo

El código de demostración del módulo `turtle` se trasladó del directorio *Demo* a la biblioteca principal. Incluye más de una docena de scripts de muestra con visualizaciones dinámicas. Al estar en `sys.path`, ahora se puede ejecutar directamente desde la línea de comandos:

```
$ python -m turtledemo
```

(Movido del directorio Demo por Alexander Belopolsky en bpo-10199 [https://bugs.python.org/issue?@action=redirect&bpo=10199].)

Multi-threading

- Se ha reescrito el mecanismo para serializar la ejecución de subprocesos de Python que se ejecutan simultáneamente (generalmente conocido como GIL o Global Interpreter Lock). Entre los objetivos se encontraban intervalos de conmutación más predecibles y una reducción de la sobrecarga debido a la contención de bloqueos y la cantidad de llamadas al sistema resultantes. Se ha abandonado la noción de un «intervalo de verificación» para permitir cambios de subproceso y se ha reemplazado por una duración absoluta expresada en segundos. Este parámetro se puede ajustar a través de `sys.setswitchinterval()`. Actualmente, el valor predeterminado es 5 milisegundos.

  Se pueden leer detalles adicionales sobre la implementación en un mensaje de lista de correo python-dev mailing-list message [https://mail.python.org/pipermail/python-dev/2009-

October/093321.html] (sin embargo, «prioridad las solicitudes» como se expone en este mensaje no se han mantenido para su inclusión).

(Contribución de Antoine Pitrou.)

- Los bloqueos regulares y recursivos ahora aceptan un argumento opcional *timeout* para su método **`acquire()`** . (Contribución de Antoine Pitrou; bpo-7316 [https://bugs.python.org/issue?@action=redirect&bpo=7316].)
- De manera similar, **`threading.Semaphore.acquire()`** también ganó un argumento *timeout*. (Contribución de Torsten Landschoff; bpo-850728 [https://bugs.python.org/issue?@action=redirect&bpo=850728].)
- Las adquisiciones de bloqueos regulares y recursivas ahora pueden ser interrumpidas por señales en plataformas que usan Pthreads. Esto significa que los programas de Python que se bloquean mientras adquieren bloqueos pueden eliminarse con éxito enviando repetidamente SIGINT al proceso (presionando `Ctrl+C` en la mayoría de los shells). (Contribuido por Reid Kleckner; bpo-8844 [https://bugs.python.org/issue?@action=redirect&bpo=8844].)

Optimizations

Se han agregado una serie de pequeñas mejoras de rendimiento:

- El optimizador de mirilla de Python ahora reconoce patrones como `x in {1, 2, 3}` como una prueba de pertenencia a un conjunto de constantes. El optimizador reformula **`set`** como **`frozenset`** y almacena la constante preconstruida.

  Ahora que la penalización de velocidad se ha ido, es práctico comenzar a escribir pruebas de membresía usando notación de conjuntos. Este estilo es semánticamente claro y operativamente rápido:

  ```
  extensión = nombre.rpartition('.')[2]
  ```

```
si extensión en {'xml', 'html', 'xhtml', 'css'}:
handle(nombre)
```

(Parche y pruebas adicionales aportadas por Dave Malcolm; bpo-6690 [https://bugs.python.org/issue?@action=redirect&bpo=6690]).

- Serializar y deserializar datos usando el módulo **`pickle`** ahora es varias veces más rápido.

  (Contribución de Alexandre Vassalotti, Antoine Pitrou y el equipo de Unladen Swallow en bpo-9410 [https://bugs.python.org/issue?@action=redirect&bpo=9410] y bpo-3873 [https://bugs.python.org/issue?@action=redirect&bpo=3873].)

- El algoritmo Timsort algorithm [https://en.wikipedia.org/wiki/Timsort] usado en **`list.sort()`** y **`sorted()`** ahora se ejecuta más rápido y usa menos memoria cuando se llama con key function. Anteriormente, cada elemento de una lista se envolvía con un objeto temporal que recordaba el valor clave asociado con cada elemento. Ahora, dos matrices de claves y valores se ordenan en paralelo. Esto ahorra la memoria consumida por los contenedores de clasificación y ahorra el tiempo perdido al delegar comparaciones.

  (Parche de Daniel Stutzbach en bpo-9915 [https://bugs.python.org/issue?@action=redirect&bpo=9915].)

- El rendimiento de la decodificación JSON mejora y el consumo de memoria se reduce siempre que se repite la misma cadena para varias claves. Además, la codificación JSON ahora usa las aceleraciones de C cuando el argumento `sort_keys` es verdadero.

  (Contribución de Antoine Pitrou en bpo-7451 [https://bugs.python.org/issue?@action=redirect&bpo=7451] y Raymond Hettinger y Antoine Pitrou en bpo-10314 [https://bugs.python.org/issue?@action=redirect&bpo=10314].)

- Los bloqueos recursivos (creados con la API **`threading.RLock()`**) ahora se benefician de una implementación de C que los hace tan rápidos como los bloqueos normales, y entre 10 y 15 veces más rápidos que su anterior implementación pura de Python.

(Contribución de Antoine Pitrou; bpo-3001 [https://bugs.python.org/issue?@action=redirect&bpo=3001].)

- El algoritmo de búsqueda rápida de stringlib ahora lo utilizan los métodos `split()`, `rsplit()`, `splitlines()` y `replace()` en los objetos `bytes`, `bytearray` y `str`. Asimismo, el algoritmo también lo utilizan `rfind()`, `rindex()`, `rsplit()` y `rpartition()`.

  (Parche de Florent Xicluna en bpo-7622 [https://bugs.python.org/issue?@action=redirect&bpo=7622] y bpo-7462 [https://bugs.python.org/issue?@action=redirect&bpo=7462].)

- Las conversiones de enteros a cadenas de caracteres ahora funcionan con dos «dígitos» a la vez, lo que reduce el número de operaciones de división y módulo.

  (bpo-6713 [https://bugs.python.org/issue?@action=redirect&bpo=6713] de Gawain Bolton, Mark Dickinson y Victor Stinner.)

Se realizaron otras optimizaciones menores. La diferenciación de conjuntos ahora se ejecuta más rápido cuando un operando es mucho más grande que el otro (parche de Andress Bennetts en bpo-8685 [https://bugs.python.org/issue?@action=redirect&bpo=8685]). El método `array.repeat()` tiene una implementación más rápida (bpo-1569291 [https://bugs.python.org/issue?@action=redirect&bpo=1569291] de Alexander Belopolsky). El `BaseHTTPRequestHandler` tiene un almacenamiento en búfer más eficiente (bpo-3709 [https://bugs.python.org/issue?@action=redirect&bpo=3709] de Andrew Schaaf). La función `operator.attrgetter()` se ha acelerado (bpo-10160 [https://bugs.python.org/issue?@action=redirect&bpo=10160] de Christos Georgiou). Y `ConfigParser` carga argumentos de varias líneas un poco más rápido (bpo-7113 [https://bugs.python.org/issue?@action=redirect&bpo=7113] de Łukasz Langa).

Unicode

Python se ha actualizado a Unicode 6.0.0 [https://unicode.org/versions/Unicode6.0.0/]. La actualización del estándar agrega más de 2000 caracteres nuevos, incluidos los símbolos emoji [https://en.wikipedia.org/wiki/Emoji] que son importantes para los teléfonos móviles.

Además, el estándar actualizado modificó las propiedades de dos caracteres de Kannada (U+0CF1, U+0CF2) y un carácter numérico New Tai Lue (U+19DA), lo que hace que el primero sea elegible para su uso en identificadores y descalifica al segundo. Para obtener más información, consulte Unicode Character Database Changes [https://www.unicode.org/versions/Unicode6.0.0/#Database_Changes].

Codecs

Se agregó soporte para *cp720* codificación DOS árabe (bpo-1616979 [https://bugs.python.org/issue?@action=redirect&bpo=1616979]).

La codificación MBCS ya no ignora el argumento del controlador de errores. En el modo estricto predeterminado, genera un **`UnicodeDecodeError`** cuando encuentra una secuencia de bytes no codificable y un **`UnicodeEncodeError`** para un carácter no codificable.

El códec MBCS admite los controladores de errores `'strict'` e `'ignore'` para la decodificación y `'strict'` y `'replace'` para la codificación.

Para emular la codificación Python3.1 MBCS, seleccione el controlador `'ignore'` para decodificar y el controlador `'replace'` para codificar.

En Mac OS X, Python decodifica los argumentos de la línea de comandos con `'utf-8'` en lugar de la codificación local.

De forma predeterminada, **`tarfile`** usa la codificación `'utf-8'` en Windows (en lugar de `'mbcs'`) y el controlador de errores `'surrogateescape'` en todos los sistemas operativos.

Documentación

La documentación sigue mejorando.

- Se ha agregado una tabla de enlaces rápidos en la parte superior de secciones largas como Funciones incorporadas. En el caso de `itertools`, los enlaces están acompañados de tablas de resúmenes al estilo de una hoja de trucos para proporcionar una descripción general y un cambio de memoria sin tener que leer todos los documentos.

- En algunos casos, el código fuente puro de Python puede ser un complemento útil de la documentación, por lo que ahora muchos módulos incluyen enlaces rápidos a la última versión del código fuente. Por ejemplo, la documentación del módulo `functools` tiene un enlace rápido en la parte superior etiquetado:

  > **Código fuente** Lib/functools.py [https://github.com/python/cpython/tree/3.14/Lib/functools.py].

  (Contribuido por Raymond Hettinger; ver rationale [https://rhettinger.wordpress.com/2011/01/28/open-your-source-more/].)

- Los documentos ahora contienen más ejemplos y recetas. En particular, el módulo `re` tiene una sección extensa, Ejemplos de expresiones regulares. Asimismo, el módulo `itertools` continúa siendo actualizado con nuevo Fórmulas con itertools.

- El módulo `datetime` ahora tiene una implementación auxiliar en Python puro. No se modificó ninguna funcionalidad. Esto solo proporciona una implementación alternativa más fácil de leer.

  (Contribuido por Alexander Belopolsky en bpo-9528 [https://bugs.python.org/issue?@action=redirect&bpo=9528].)

- Se ha eliminado el directorio no mantenido `Demo`. Algunas demostraciones se integraron en la documentación, otras se movieron al directorio `Tools/demo` y otras se eliminaron por completo.

(Contribución de Georg Brandl en bpo-7962 [https://bugs.python.org/issue?@action=redirect&bpo=7962].)

IDLE

- El menú de formato ahora tiene una opción para limpiar archivos de origen eliminando los espacios en blanco finales.

  (Contribución de Raymond Hettinger; bpo-5150 [https://bugs.python.org/issue?@action=redirect&bpo=5150].)

- IDLE en Mac OS X ahora funciona con Carbon AquaTk y Cocoa AquaTk.

  (Contribución de Kevin Walzer, Ned Deily y Ronald Oussoren; bpo-6075 [https://bugs.python.org/issue?@action=redirect&bpo=6075].)

Repositorio de código

Además del repositorio de código Subversion existente en https://svn.python.org, ahora hay un repositorio Mercurial [https://www.mercurial-scm.org/] en https://hg.python.org/.

Después de la versión 3.2, hay planes para cambiar a Mercurial como repositorio principal. Este sistema de control de versiones distribuido debería facilitar a los miembros de la comunidad la creación y el intercambio de conjuntos de cambios externos. Consulte **PEP 385** [https://peps.python.org/pep-0385/] para obtener más detalles.

Para aprender a utilizar el nuevo sistema de control de versiones, consulte el Quick Start [https://www.mercurial-scm.org/wiki/QuickStart] o la Guide to Mercurial Workflows [https://www.mercurial-scm.org/guide].

Cambios en la API de construcción y C

Los cambios en el proceso de compilación de Python y en la API de C incluyen:

- Los scripts *idle*, *pydoc* y *2to3* ahora están instalados con un sufijo específico de la versión en `make altinstall` (bpo-10679 [https://bugs.python.org/issue?@action=redirect&bpo=10679]).
- Las funciones C que acceden a la base de datos Unicode ahora aceptan y retornan caracteres del rango Unicode completo, incluso en compilaciones Unicode estrechas (Py_UNICODE_TOLOWER, Py_UNICODE_ISDECIMAL, y otras). Una diferencia visible en Python es que **`unicodedata.numeric()`** ahora devuelve el valor correcto para puntos de código grandes y **`repr()`** puede considerar más caracteres como imprimibles.

  (Informado por Bupjoe Lee y corregido por Amaury Forgeot D'Arc; bpo-5127 [https://bugs.python.org/issue?@action=redirect&bpo=5127].)
- Los gotos calculados ahora están habilitados de forma predeterminada en los compiladores compatibles (que son detectados por el script de configuración). Todavía se pueden desactivar de forma selectiva especificando `--without-computed-gotos`.

  (Contribución de Antoine Pitrou; bpo-9203 [https://bugs.python.org/issue?@action=redirect&bpo=9203].)
- Se eliminó la opción `--with-wctype-functions`. La base de datos unicode incorporada ahora se usa para todas las funciones.

  (Contribución de Amaury Forgeot D'Arc; bpo-9210 [https://bugs.python.org/issue?@action=redirect&bpo=9210].)
- Los valores hash ahora son valores de un nuevo tipo, **`Py_hash_t`**, que se define como del mismo tamaño que un puntero. Anteriormente eran del tipo long, que en algunos sistemas operativos de 64 bits todavía tiene una longitud de

solo 32 bits. Como resultado de esta corrección, `set` y `dict` ahora pueden contener más de `2**32` entradas en compilaciones con punteros de 64 bits (anteriormente, podían crecer hasta ese tamaño pero su desempeño se degradaba catastróficamente).

(Sugerido por Raymond Hettinger e implementado por Benjamin Peterson; bpo-9778 [https://bugs.python.org/issue?@action=redirect&bpo=9778].)

- Una nueva macro **`Py_VA_COPY`** copia el estado de la lista de argumentos de la variable. Es equivalente a C99 *va_copy* pero está disponible en todas las plataformas Python (bpo-2443 [https://bugs.python.org/issue?@action=redirect&bpo=2443]).
- Una nueva función API C **`PySys_SetArgvEx()`** permite que un intérprete integrado configure `sys.argv` sin modificar también `sys.path` (bpo-5753 [https://bugs.python.org/issue?@action=redirect&bpo=5753]).
- **`PyEval_CallObject()`** ahora solo está disponible en formato macro. La declaración de función, que se mantuvo por razones de compatibilidad con versiones anteriores, se ha eliminado: la macro se introdujo en 1997 (bpo-8276 [https://bugs.python.org/issue?@action=redirect&bpo=8276]).
- Hay una nueva función `PyLong_AsLongLongAndOverflow()` que es análoga a `PyLong_AsLongAndOverflow()`. Ambos sirven para convertir Python `int` en un tipo nativo de ancho fijo al tiempo que brindan detección de casos en los que la conversión no encaja (bpo-7767 [https://bugs.python.org/issue?@action=redirect&bpo=7767]).
- La función `PyUnicode_CompareWithASCIIString()` ahora devuelve *not equal* si la cadena de Python es *NUL* terminada.
- Hay una nueva función `PyErr_NewExceptionWithDoc()` que es como `PyErr_NewException()` pero permite especificar una cadena de documentos. Esto permite que las

excepciones de C tengan las mismas capacidades de autodocumentación que sus contrapartes de Python puro (bpo-7033 [https://bugs.python.org/issue?@action=redirect&bpo=7033]).

- Cuando se compila con la opción `--with-valgrind`, el asignador de pymalloc se desactivará automáticamente cuando se ejecute en Valgrind. Esto brinda una detección mejorada de fugas de memoria cuando se ejecuta en Valgrind, mientras se aprovecha pymalloc en otros momentos (bpo-2422 [https://bugs.python.org/issue?@action=redirect&bpo=2422]).

- Se eliminó el formato `O?` De las funciones *PyArg_Parse*. El formato ya no se usa y nunca se había documentado (bpo-8837 [https://bugs.python.org/issue?@action=redirect&bpo=8837]).

Hubo una serie de otros pequeños cambios en la C-API. Consulte el archivo Misc/NEWS [https://github.com/python/cpython/blob/v3.2.6/Misc/NEWS] para obtener una lista completa.

Además, se realizaron varias actualizaciones en la compilación de Mac OS X; consulte Mac/BuildScript/README.txt [https://github.com/python/cpython/blob/v3.2.6/Mac/BuildScript/README.txt] para obtener más detalles. Para los usuarios que ejecutan una compilación de 32/64 bits, existe un problema conocido con el Tcl/Tk predeterminado en Mac OS X 10.6. En consecuencia, recomendamos instalar una alternativa actualizada como ActiveState Tcl/Tk 8.5.9 [https://web.archive.org/web/20101208191259/https://www.activestate.com/activetcl/downloads]. Consulte https://www.python.org/download/mac/tcltk/ para obtener más detalles.

Portar a Python 3.2

Esta sección enumera los cambios descritos anteriormente y otras correcciones de errores que pueden requerir cambios en su código:

- El módulo **`configparser`** tiene una serie de mejoras. El cambio principal es reemplazar la antigua clase

`ConfigParser` por la alternativa preferida de larga data **`SafeConfigParser`**. Además, hay una serie de incompatibilidades menores:

  - La sintaxis de interpolación ahora está validada en las operaciones `get()` y `set()`. En el esquema de interpolación predeterminado, solo dos tokens con signos de porcentaje son válidos: `%(name)s` y `%%`, siendo este último un signo de porcentaje de escape.
  - Los métodos `set()` y `add_section()` ahora verifican que los valores sean cadenas reales. Anteriormente, los tipos no admitidos podían introducirse involuntariamente.
  - Las secciones u opciones duplicadas de una sola fuente ahora generan `DuplicateSectionError` o `DuplicateOptionError`. Anteriormente, los duplicados sobrescribían silenciosamente una entrada anterior.
  - Los comentarios en línea ahora están deshabilitados de forma predeterminada, por lo que ahora el carácter ; se puede usar de forma segura en los valores.
  - Los comentarios ahora se pueden sangrar. En consecuencia, para que ; o # aparezcan al principio de una línea en valores multilínea, debe interpolarse. Esto evita que los caracteres de prefijo de comentario en valores se confundan con comentarios.
  - `""` ahora es un valor válido y ya no se convierte automáticamente en una cadena vacía. Para cadenas vacías, use `"option ="` en una línea.

- El módulo **`nntplib`** fue rediseñado ampliamente, lo que significa que sus API a menudo son incompatibles con las API 3.1.

- `bytearray` ya no se pueden usar como nombres de archivo; en su lugar, deben convertirse a `bytes`.

- Los nombres **`array.tostring()`** y **`array.fromstring()`** se han cambiado a `array.tobytes()` y `array.frombytes()` para mayor claridad. Los nombres antiguos han quedado obsoletos.

(Consulte bpo-8990 [https://bugs.python.org/issue?@action=redirect&bpo=8990]).

- Funciones `PyArg_Parse*()`:
    - Se ha eliminado el formato «t#»: utilice «s#» o «s*» en su lugar
    - Se han eliminado los formatos «w» y «w #»: utilice «w*» en su lugar

- Se ha eliminado el tipo **`PyCObject`**, que quedó obsoleto en la versión 3.1. Para encapsular punteros C opacos en objetos Python, se debe utilizar la API **`PyCapsule`**; el nuevo tipo tiene una interfaz bien definida para pasar información de seguridad de tipado y una firma menos complicada para llamar a un destructor.

- La función **`sys.setfilesystemencoding()`** se eliminó porque tenía un diseño defectuoso.

- La función y el método **`random.seed()`** ahora saltan semillas de cadenas con una función hash sha512. Para acceder a la versión anterior de *seed* para reproducir secuencias de Python 3.1, establezca el argumento *version* en *1*, `random.seed(s, version=1)`.

- La función **`string.maketrans()`**, que anteriormente estaba en desuso, se ha eliminado en favor de los métodos estáticos **`bytes.maketrans()`** y **`bytearray.maketrans()`**. Este cambio resuelve la confusión sobre qué tipos eran compatibles con el módulo **`string`**. Ahora, **`str`**, **`bytes`** y **`bytearray`** tienen sus propios métodos **maketrans** y **translate** con tablas de traducción intermedias del tipo adecuado.

  (Contribución de Georg Brandl; bpo-5675 [https://bugs.python.org/issue?@action=redirect&bpo=5675].)

- La función **`contextlib.nested()`**, que anteriormente estaba en desuso, se ha eliminado en favor de una declaración **`with`** simple que puede aceptar múltiples administradores de contexto. La última técnica es más rápida (porque está incorporada) y hace un mejor trabajo al finalizar múltiples administradores de contexto cuando uno de ellos genera una excepción:

```
con open('mylog.txt') como archivo de entrada, open
para línea en archivo de entrada:
si '<critical>' está en línea:
archivo de salida.write(línea)
```

(Contribución de Georg Brandl y Mattias Brändström; appspot issue 53094 [https://codereview.appspot.com/53094].)

- `struct.pack()` ahora permite bytes para el código del paquete de cadenas `s`. Anteriormente, aceptaba argumentos de texto y los codificaba implícitamente en bytes usando UTF-8. Esto era problemático porque hacía suposiciones sobre la codificación correcta y porque una codificación de longitud variable puede fallar al escribir en un segmento de longitud fija de una estructura.

  El código como `struct.pack('<6sHHBBB', 'GIF87a', x, y)` debe reescribirse para usar bytes en lugar de texto, `struct.pack('<6sHHBBB', b'GIF87a', x, y)`.

  (Descubierto por David Beazley y corregido por Victor Stinner; bpo-10783 [https://bugs.python.org/issue?@action=redirect&bpo=10783].)

- La clase `xml.etree.ElementTree` ahora genera un `xml.etree.ElementTree.ParseError` cuando falla un análisis. Anteriormente generaba un `xml.parsers.expat.ExpatError`.

- El nuevo valor más largo de `str()` en flotantes puede romper las pruebas de documentación que se basan en el formato de salida anterior.

- En `subprocess.Popen`, el valor predeterminado para *close_fds* es ahora `True` en Unix; en Windows, es `True` si los tres flujos estándar están configurados como `None`, o `False` en caso contrario. Anteriormente, *close_fds* siempre era `False` de forma predeterminada, lo que producía errores difíciles de resolver o condiciones de carrera cuando los descriptores de archivos abiertos se filtraban en el proceso hijo.

- Se ha eliminado la compatibilidad con HTTP 0.9 heredado de **`urllib.request`** y **`http.client`**. Dicho soporte todavía está presente en el lado del servidor (en **`http.server`**).

  (Contribución de Antoine Pitrou, bpo-10711 [https://bugs.python.org/issue?@action=redirect&bpo=10711].)

- Los sockets SSL en modo de tiempo de espera ahora generan **`socket.timeout`** cuando ocurre un tiempo de espera, en lugar de un genérico **`SSLError`**.

  (Contribución de Antoine Pitrou, bpo-10272 [https://bugs.python.org/issue?@action=redirect&bpo=10272].)

- Las funciones engañosas **`PyEval_AcquireLock()`** y **`PyEval_ReleaseLock()`** han quedado oficialmente obsoletas. En su lugar, se deben utilizar las API que reconocen el estado del subproceso (como **`PyEval_SaveThread()`** y **`PyEval_RestoreThread()`**).

- Debido a riesgos de seguridad, **`asyncore.handle_accept()`** ha quedado obsoleto y se agregó una nueva función, **`asyncore.handle_accepted()`**, para reemplazarlo.

  (Contribución de Giampaolo Rodola en bpo-6706 [https://bugs.python.org/issue?@action=redirect&bpo=6706].)

- Debido a la nueva implementación de GIL, ya no se puede llamar a **`PyEval_InitThreads()`** antes que a **`Py_Initialize()`**.

Qué hay de nuevo en Python 3.1

Autor:

Raymond Hettinger

This article explains the new features in Python 3.1, compared to 3.0. Python 3.1 was released on June 27, 2009.

PEP 372: Diccionarios ordenados

Los diccionarios de Python normales iteran sobre los pares clave/valor en orden arbitrario. A través de los años, varios autores han escrito implementaciones alternativas que recuerdan el orden en el que se insertaron originalmente las claves. Basándose en las experiencias de esas implementaciones, una nueva clase `collections.OrderedDict` ha sido introducida.

La API de OrderedDict es sustancialmente la misma que la de los diccionarios normales, pero iterará sobre las claves y los valores en un orden garantizado, dependiendo de cuándo se insertó una clave por primera vez. Si una nueva entrada sobrescribe una entrada existente, la posición de inserción original se deja sin cambios. Eliminar una entrada y volver a insertarla la moverá hasta el final.

La biblioteca estándar ahora admite el uso de diccionarios ordenados en varios módulos. El módulo `configparser` los usa por defecto. Esto permite leer, modificar y volver a escribir los archivos de configuración en su orden original. El método *_asdict()* para `collections.namedtuple()` ahora devuelve un diccionario ordenado con los valores que aparecen en el mismo orden que los índices de tupla subyacentes. El módulo `json` se está construyendo con un *object_pairs_hook* para permitir que el decodificador construya OrderedDicts. También se agregó soporte para herramientas de terceros como PyYAML [https://pyyaml.org/].

Ver también

PEP 372 [https://peps.python.org/pep-0372/] - Diccionarios ordenados

PEP escrito por Armin Ronacher y Raymond Hettinger. Implementación escrita por Raymond Hettinger.

Dado que un diccionario ordenado recuerda su orden de inserción, se puede utilizar junto con la ordenación para crear un diccionario ordenado:

```
>>> # diccionario regular sin clasificar
>>> d = {'banana': 3, 'manzana':4, 'pera': 1, 'naranja':

>>> # diccionario ordenado por clave
>>> OrderedDict(sorted(d.items(), key=lambda t: t[0]))
OrderedDict([('manzana', 4), ('banana', 3), ('naranja',

>>> # diccionario ordenado por valor
>>> OrderedDict(sorted(d.items(), key=lambda t: t[1]))
OrderedDict([('pera', 1), ('naranja', 2), ('banana', 3),

>>> # diccionario ordenado por longitud de la clave cade
>>> OrderedDict(sorted(d.items(), key=lambda t: len(t[0]
OrderedDict([('pera', 1), ('manzana', 4), ('naranja', 2)
```

Los nuevos diccionarios ordenados mantienen su orden de clasificación cuando se eliminan entradas, pero cuando se agregan nuevas claves, estas se agregan al final y no se mantiene el orden.

PEP 378: Especificador de formato para el separador de miles

La función incorporada `format()` y el método `str.format()` usan un mini-lenguaje que ahora incluye una forma simple, que no tiene en cuenta la configuración regional, de formatear un número con un separador de miles. Eso proporciona una manera de humanizar la salida de un programa, mejorando su apariencia profesional y legibilidad:

```
>>> formato(1234567, ',d')
'1,234,567'
>>> formato(1234567.89, ',.2f')
'1,234,567.89'
>>> formato(12345.6 + 8901234.12j, ',f')
'12,345.600000+8,901,234.120000j'
>>> formato(Decimal('1234567.89'), ',f')
'1,234,567.89'
```

Los tipos soportados son `int`, `float`, `complex` y `decimal.Decimal`.

Se está discutiendo cómo especificar separadores alternativos como puntos, espacios, apóstrofos o guiones bajos. Las aplicaciones que reconocen la configuración regional deben usar el especificador de formato *n* existente, que ya es compatible con el separadores de miles.

Ver también

PEP 378 [https://peps.python.org/pep-0378/] - Especificador de formato para el separador de miles

PEP escrito por Raymond Hettinger e implementado por Eric Smith y Mark Dickinson.

Otros cambios del lenguaje

Algunos cambios pequeños en el núcleo del lenguaje Python son:

- Los directorios y archivos zip que contienen un archivo `__main__.py` pueden ahora ser ejecutados directamente pasando su nombre al intérprete. El directorio/archivo zip es automáticamente insertado como la primera entrada en sys.path. (Sugerencia y parche inicial por Andy Chu; parche revisado por Phillip J. Eby y Nick Coghlan; bpo-1739468 [https://bugs.python.org/issue?@action=redirect&bpo=1739468].)

- El tipo `int()` ganó un método `bit_length` que retorna el número de bits necesarios para representar sus argumentos en binario:

  ```
  >>> n = 37
  >>> bin(37)
  '0b100101'
  >>> n.bit_length()
  6
  >>> n = 2**123-1
  >>> n.bit_length()
  123
  >>> (n+1).bit_length()
  124
  ```

  (Contribución de Fredrik Johansson, Victor Stinner, Raymond Hettinger y Mark Dickinson; bpo-3439 [https://bugs.python.org/issue?@action=redirect&bpo=3439].)

- Los campos en las cadenas de formato de `format()` ahora pueden ser automáticamente numerados:

  ```
  >>> 'Señor {} de {}'.format('Gallahad', 'Camelot')
  'Sir Gallahad de Camelot'
  ```

  Anteriormente, la cadena requería campos numerados como: `'Sir {0} of {1}'`.

  (Contribución de Eric Smith; bpo-5237 [https://bugs.python.org/issue?@action=redirect&bpo=5237].)

- La función **`string.maketrans()`** ha quedado obsoleta y se ha reemplazado por nuevos métodos estáticos, `bytes.maketrans()` y `bytearray.maketrans()`. Este cambio resuelve la confusión sobre qué tipos eran compatibles con el módulo `string`. Ahora, `str`, `bytes` y `bytearray` tienen sus propios métodos **maketrans** y **translate** con tablas de traducción intermedias del tipo adecuado.

  (Contribución de Georg Brandl; bpo-5675 [https://

bugs.python.org/issue?@action=redirect&bpo=5675].)

- La sintaxis de la sentencia **`with`** ahora permite múltiples gestores de contexto en una sola declaración:

  ```
  >>> con open('mylog.txt') como archivo de entrada,
  ... para línea en archivo de entrada:
  ... si '<critical>' está en línea:
  ... archivo de salida.write(línea)
  ```

  Con la nueva sintaxis, la función **`contextlib.nested()`** ya no es necesaria y ahora está obsoleta.

  (Contribución de Georg Brandl y Mattias Brändström; appspot issue 53094 [https://codereview.appspot.com/53094].)

- `round(x, n)` ahora retorna un número entero si *x* es un entero. Anteriormente retornaba un número flotante:

  ```
  >>> redondear(1123, -2)
  1100
  ```

  (Contribución de Mark Dickinson; bpo-4707 [https://bugs.python.org/issue?@action=redirect&bpo=4707].)

- Python ahora utiliza el algoritmo de David Gay para encontrar la representación de punto flotante más corta que no cambia su valor. Esto debería ayudar a mitigar parte de la confusión que rodea a los números binarios de punto flotante.

  La importancia se aprecia fácilmente con un número como `1.1`, que no tiene un equivalente exacto en coma flotante binaria. Como no hay un equivalente exacto, una expresión como `float('1.1')` se evalúa como el valor representable más cercano, que es `0x1.199999999999ap+0` en hexadecimal o `1.1000000000000000888178419700125232338905334472656` en decimal. Ese valor más cercano se utilizó y todavía se utiliza en cálculos de coma flotante posteriores.

  Lo nuevo es cómo se muestra el número. Anteriormente, Python utilizaba un enfoque simple. El valor de `repr(1.1)` se calculaba como `format(1.1, '.17g')`, que se evaluaba como `'1.1000000000000001'`. La ventaja de utilizar 17 dígitos era que dependía de las garantías IEEE-754 para asegurar que `eval(repr(1.1))` regresaría exactamente a su valor original. La desventaja es que muchas personas encontraron que el resultado era confuso (confundiendo las limitaciones intrínsecas de la representación binaria de punto flotante con un problema del propio Python).

El nuevo algoritmo para `repr(1.1)` es más inteligente y retorna `'1.1'`. Efectivamente, busca todas las representaciones de cadenas equivalentes (las que se almacenan con el mismo valor flotante subyacente) y retorna la representación más corta.

El nuevo algoritmo tiende a emitir representaciones más limpias cuando es posible, pero esto no cambia los valores subyacentes. Por lo tanto, todavía se da el caso `1.1 + 2.2 != 3.3`, aún cuando las representaciones puedan sugerir lo contrario.

El nuevo algoritmo depende de ciertas características de la implementación de punto flotante subyacente. Si no se encuentran las características requeridas, se seguirá utilizando el algoritmo antiguo. Además, los protocolos de pickle de texto garantizan la portabilidad entre plataformas mediante el uso del algoritmo antiguo.

(Contribución de Eric Smith y Mark Dickinson; bpo-1580 [https://bugs.python.org/issue?@action=redirect&bpo=1580])

Módulos nuevos, mejorados y obsoletos

- Se ha añadido una clase **`collections.Counter`** para admitir el recuento conveniente de elementos únicos en una secuencia o una iteración:

```
>>> Contador(['rojo', 'azul', 'rojo', 'verde', 'azu
Contador({'azul': 3, 'rojo': 2, 'verde': 1})
```

(Contribución de Raymond Hettinger; bpo-1696199 [https://bugs.python.org/issue?@action=redirect&bpo=1696199].)

- Se ha añadido un nuevo módulo `tkinter.ttk` para acceder al conjunto de widgets temáticos de Tk. La idea básica de ttk es separar, dentro de lo posible, el código que implementa el comportamiento de un widget del código que implementa su apariencia.

  (Contribución de Guilherme Polo; bpo-2983 [https://bugs.python.org/issue?@action=redirect&bpo=2983].)

- Las clases `gzip.GzipFile` y `bz2.BZ2File` ahora admiten el protocolo de gestión de contexto:

  ```
  >>> # Cerrar automáticamente el archivo después de
  >>> con gzip.GzipFile(filename, "wb") como f:
  ...     f.write(b"xxx")
  ```

  (Contribución de Antoine Pitrou.)

- El módulo `decimal` ahora admite métodos para crear un objeto decimal de un `float` binario. La conversión es exacta pero puede ser a veces una sorpresa:

  ```
  >>> Decimal.from_float(1.1)
  Decimal('1.1000000000000000888178419700125232338905
  ```

  El resultado decimal largo muestra la fracción binaria real que se almacena para *1.1*. La fracción tiene muchos dígitos porque *1.1* no se puede representar exactamente en binario.

  (Contribución de Raymond Hettinger y Mark Dickinson.)

- El módulo `itertools` desarrolló dos nuevas funciones. La función `itertools.combinations_with_replacement()` es una de las cuatro para generar combinatorias que incluyen permutaciones y productos cartesianos. La función `itertools.compress()` imita su homónimo de APL.

Además, la función existente `itertools.count()` tiene ahora un argumento *step* opcional y puede aceptar cualquier tipo de secuencia de conteo, incluyendo `fractions.Fraction` y `decimal.Decimal`:

```
>>> [p+q para p,q en combinaciones_con_reemplazo('L
['LL', 'LO', 'LV', 'LE', 'OO', 'OV', 'OE', 'VV', 'V

>>> lista(comprimir(datos=rango(10), selectores=[0,
[2, 3, 5, 7]

>>> c = conteo(inicio=Fracción(1,2), paso=Fracción(
>>> [siguiente(c), siguiente(c), siguiente(c), sigu
[Fracción(1, 2), Fracción(2, 3), Fracción(5, 6), Fr
```

(Contribución de Raymond Hettinger.)

- `collections.namedtuple()` ahora admite un argumento de palabra clave *rename* que permite que los campos de nombre inválidos se conviertan automáticamente en nombres posicionados de la forma _0, _1, etc. Esto es de utilidad cuando los nombres del campo están siendo creados por una fuente externa como un encabezado CSV, una lista de campos SQL, o la entrada del usuario:

  ```
  >>> consulta = entrada()
  SELECT región, departamento, count(*) FROM main GRO

  >>> cursor.execute(query)
  >>> query_fields = [desc[0] for desc in cursor.desc
  >>> UserQuery = namedtuple('UserQuery', query_field
  >>> pprint.pprint([UserQuery(*row) para la fila en
  [UserQuery(región='Sur', departamento='Envíos', _2=
  UserQuery(región='Norte', departamento='Contabilida
  UserQuery(región='Oeste', departamento='Ventas', _2
  ```

  (Contribución de Raymond Hettinger; bpo-1818 [https://bugs.python.org/issue?@action=redirect&bpo=1818].)

- Las funciones `re.sub()`, `re.subn()` y `re.split()`

ahora admiten un parámetro *flags*.

(Contribución de Gregory Smith.)

- El módulo **`logging`** ahora implementa una clase simple **`logging.NullHandler`** para aplicaciones que no utilizan el registro pero llaman al código de una biblioteca que si lo hace. La configuración de un controlador nulo suprimirá las advertencias falsas como «No handlers could be found for logger foo»:

  ```
  >>> h = registro.NullHandler()
  >>> registro.getLogger("foo").addHandler(h)
  ```

  (Contribución de Vinay Sajip; bpo-4384 [https://bugs.python.org/issue?@action=redirect&bpo=4384]).

- El módulo **`runpy`** que admite el modificador de línea de comando `-m` ahora admite la ejecución de paquetes al buscar y ejecutar un submódulo `__main__` cuando se proporciona un nombre de paquete.

  (Contribución de Andi Vajda; bpo-4195 [https://bugs.python.org/issue?@action=redirect&bpo=4195].)

- El módulo **`pdb`** puede ahora acceder y mostrar el código fuente cargado a través de **`zipimport`** (o cualquier otro cargador conforme **PEP 302** [https://peps.python.org/pep-0302/]).

  (Contribución de Alexander Belopolsky; bpo-4201 [https://bugs.python.org/issue?@action=redirect&bpo=4201].)

- Los objetos **`functools.partial`** pueden ser ahora serializados (*pickled*).

  > (Sugerido por Antoine Pitrou y Jesse Noller. Implementado por Jack Diederich; bpo-5228 [https://bugs.python.org/issue?@action=redirect&bpo=5228].)

- Se agrega temas de ayuda **`pydoc`** para símbolos de modo que `help('@')` funcione como se espera en un entorno interactivo.

(Contribución de David Laban; bpo-4739 [https://bugs.python.org/issue?@action=redirect&bpo=4739].)

- El módulo `unittest` ahora admite saltear pruebas individuales o clases de pruebas. Y admite marcar una prueba como una falla esperada, una prueba que se sabe que está rota, pero que no debe contarse como una falla en un TestResult:

  ```
  clase TestGizmo(unittest.TestCase):

  @unittest.skipUnless(sys.platform.startswith("win")
  def test_gizmo_on_windows(self):
  ...

  @unittest.expectedFailure
  def test_gimzo_without_required_library(self):
  ...
  ```

  Además, se han creado pruebas de excepciones para trabajar con gestores de contexto usando la declaración `with`:

  ```
  def test_division_by_zero(self):
  con self.assertRaises(ZeroDivisionError):
  x / 0
  ```

  Además, se agregaron varios métodos de afirmación nuevos, incluidos `assertSetEqual()`, `assertDictEqual()`, **`assertDictContainsSubset()`**, `assertListEqual()`, `assertTupleEqual()`, `assertSequenceEqual()`, `assertRaisesRegexp()`, `assertIsNone()` y `assertIsNotNone()`.

  (Contribución de Benjamin Peterson y Antoine Pitrou.)

- El módulo `io` tiene tres nuevas constantes para el método `seek()`: `SEEK_SET`, `SEEK_CUR` y `SEEK_END`.

- La tupla `sys.version_info` ahora es una tupla con nombre:

```
>>> sys.version_info
sys.version_info(mayor=3, menor=1, micro=0, nivel d
```

(Contribución de Ross Light; bpo-4285 [https://bugs.python.org/issue?@action=redirect&bpo=4285].)

- Los módulos **`nntplib`** y **`imaplib`** ahora admiten IPv6.

  (Contribución de Derek Morr; bpo-1655 [https://bugs.python.org/issue?@action=redirect&bpo=1655] y bpo-1664 [https://bugs.python.org/issue?@action=redirect&bpo=1664].)

- El módulo **`pickle`** ha sido adaptado para una mejor interoperabilidad con 2.x cuando es usado con un protocolo 2 o menor. La reorganización de la biblioteca estándar cambió la referencia formal para varios objetos. Por ejemplo, `__builtin__.set` en Python 2 es llamado `builtins.set` en Python 3. Este cambio confundió los esfuerzos de compartir datos entre diferentes versiones de Python. Pero ahora cuando el protocolo 2 o menor es seleccionado, el pickler va a usar automáticamente los nombres antiguos de Python 2 tanto para carga como para volcado. Esta reasignación es activada de manera predeterminada pero puede ser desactivada con la opción *fix_imports*:

  ```
  >>> s = {1, 2, 3}
  >>> pickle.dumps(s, protocolo=0)
  b'c__builtin__\nestablecer\np0\n((lp1\nL1L\naL2L\na
  >>> pickle.dumps(s, protocolo=0, fix_imports=False)
  b'cbuiltins\nestablecer\np0\n((lp1\nL1L\naL2L\naL3L
  ```

  Un efecto secundario, desafortunado pero inevitable, de este cambio es que los pickles del protocolo 2 producidos con Python 3.1 no serán legibles con Python 3.0. El protocolo de pickle más reciente, protocolo 3, debe utilizarse al migrar datos entre implementaciones de Python 3.x, ya que no intenta seguir siendo compatible con Python 2.x.

  (Contribución de Alexandre Vassalotti y Antoine Pitrou,

bpo-6137 [https://bugs.python.org/issue?@action=redirect&bpo=6137].)

- Se agregó un nuevo módulo, `importlib`. Proporciona una implementación de referencia de Python completa, portátil y pura de la instrucción `import` y su contraparte, la función `__import__()`. Representa un avance sustancial en la documentación y definición de las acciones que tienen lugar durante las importaciones.

  (Contribución de Brett Cannon.)

Optimizaciones

Se han agregado importantes mejoras de rendimiento:

- La nueva biblioteca I/O (definida en **PEP 3116** [https://peps.python.org/pep-3116/]) estaba escrita mayormente en Python y rápidamente demostró ser un cuello de botella problemático en Python 3.0. En Python 3.1, la biblioteca I/O ha sido reescrita enteramente en C y es de 2 a 20 veces más rápida dependiendo en la tarea a manejar. La versión puramente en Python está aún disponible para fines experimentales a través del módulo `_pyio`.

  (Contribución de Amaury Forgeot d'Arc y Antoine Pitrou.)

- Se ha añadido una heurística para que el recolector de basura no realice el seguimiento de tuplas y diccionarios que contengan solo objetos no rastreables. Esto puede reducir el tamaño de las colecciones y, por lo tanto, la sobrecarga de recolección de elementos no utilizados en programas de larga ejecución, en función de su uso particular de los tipos de datos.

  (Contribución de Antoine Pitrou, bpo-4688 [https://bugs.python.org/issue?@action=redirect&bpo=4688].)

- Habilitando una opción de configuración llamada `--with-computed-gotos` en compiladores que la admiten (en particular: gcc, SunPro, icc), el ciclo de evaluación del bytecode se compila con un nuevo mecanismo de despacho que proporciona aceleraciones de hasta un 20% , dependiendo del sistema, del compilador y del punto de referencia.

(Contribución de Antoine Pitrou junto con varios otros participantes, bpo-4753 [https://bugs.python.org/issue?@action=redirect&bpo=4753]).

- La decodificación de UTF-8, UTF-16 y LATIN-1 es ahora de dos a cuatro veces más rápida.

  (Contribución de Antoine Pitrou y Amaury Forgeot d'Arc, bpo-4868 [https://bugs.python.org/issue?@action=redirect&bpo=4868].)

- El módulo **`json`** ahora tiene una extensión C para mejorar sustancialmente su rendimiento. Además, se modificó la API para que json funcione solo con **`str`**, no con **`bytes`**. Ese cambio hace que el módulo coincida estrechamente con el JSON specification [https://json.org/] que se define en términos de Unicode.

  (Contribución de Bob Ippolito y convertido a Py3.1 por Antoine Pitrou y Benjamin Peterson; bpo-4136 [https://bugs.python.org/issue?@action=redirect&bpo=4136].)

- La deserialización (*unpickling*) ahora interna los nombres de los atributos de los objetos serializados (*pickled*). Esto ahorra memoria y permite que los pickles sean más pequeños.

  (Contribución de Jake McGuire y Antoine Pitrou; bpo-5084 [https://bugs.python.org/issue?@action=redirect&bpo=5084].)

IDLE

- El menú de formato del IDLE ahora proporciona una opción para eliminar los espacios en blanco finales de un archivo de código fuente.

  (Contribución de Roger D. Serwy; bpo-5150 [https://

bugs.python.org/issue?@action=redirect&bpo=5150].)

Cambios en la compilación y la API de C

Los cambios en el proceso de compilación de Python y en la API C incluyen:

- Los enteros ahora se almacenan internamente en base 2**15 o en base 2**30, la base se determina en el momento de la construcción. Anteriormente, siempre se almacenaban en base 2**15. El uso de base 2**30 ofrece mejoras significativas en el rendimiento en máquinas de 64 bits, pero los resultados de las pruebas comparativas en máquinas de 32 bits han sido diversos. Por lo tanto, el valor predeterminado es usar base 2**30 en máquinas de 64 bits y base 2**15 en máquinas de 32 bits; en Unix, hay una nueva opción de configuración `--enable-big-digits` que puede ser usada para sobre escribir este valor predeterminado.

  Aparte de las mejoras de rendimiento, este cambio debería ser invisible para los usuarios finales, con una excepción: para fines de prueba y depuración hay un nuevo **`sys.int_info`** que proporciona información sobre el formato interno, dando el número de bits por dígito y el tamaño en bytes del tipo C utilizado para almacenar cada dígito:

  ```
  >>> import sys
  >>> sys.int_info
  sys.int_info(bits_per_digit=30, sizeof_digit=4)
  ```

  (Contribución de Mark Dickinson; bpo-4258 [https://bugs.python.org/issue?@action=redirect&bpo=4258].)

- La función **`PyLong_AsUnsignedLongLong()`** es ahora capaz de manejar un *pylong* negativo lanzando una excepción **`OverflowError`** en lugar de **`TypeError`**.

  (Contribución de Mark Dickinson y Lisandro Dalcrin; bpo-5175 [https://bugs.python.org/issue?@action=redirect&bpo=5175].)

- **`PyNumber_Int()`** está ahora obsoleto. Utilice `PyNumber_Long()` en su lugar.

  (Contribución de Mark Dickinson; bpo-4910 [https://bugs.python.org/issue?@action=redirect&bpo=4910].)

- Se agregó una nueva función `PyOS_string_to_double()` para reemplazar las funciones obsoletas **`PyOS_ascii_strtod()`** y **`PyOS_ascii_atof()`**.

  (Contribución de Mark Dickinson; bpo-5914 [https://bugs.python.org/issue?@action=redirect&bpo=5914].)

- Se agregó `PyCapsule` como reemplazo de la API **`PyCObject`**. La principal diferencia es que el nuevo tipo tiene una interfaz bien definida para pasar información de seguridad de tipado y una firma menos complicada para llamar a un destructor. El tipo anterior tenía una API problemática y ahora está obsoleto.

  (Contribución de Larry Hastings; bpo-5630 [https://bugs.python.org/issue?@action=redirect&bpo=5630].)

Portando a Python 3.1

Esta sección enumera los cambios descritos anteriormente y otras correcciones de errores que pueden requerir cambios en su código:

- Las nuevas representaciones de cadenas de punto flotante pueden romper las pruebas de documentación existentes. Por ejemplo:

  ```
  def e():
      '''Compute the base of natural logarithms.

      >>> e()
      2.7182818284590451

      '''
      return sum(1/math.factorial(x) for x in reverse
  ```

```
doctest.testmod()

****************************************************
Failed example:
    e()
Expected:
    2.7182818284590451
Got:
    2.718281828459045
****************************************************
```

- La reasignación automática de nombres en el módulo pickle para el protocolo 2 o inferior puede hacer que los pickles de Python 3.1 sean ilegibles en Python 3.0. Una solución es usar el protocolo 3. Otra solución es establecer la opción *fix_imports* en `False`. Consulte la discusión anterior para obtener más detalles.

Qué hay de nuevo en Python 3.0

Autor:

Guido van Rossum

Este artículo explica las nuevas características de Python 3.0, en comparación con 2.6. Python 3.0, también conocido como «Python 3000» o «Py3K», es la primera versión *intentionally backwards incompatible* de Python. Python 3.0 se lanzó el 3 de diciembre de 2008. Hay más cambios que en una versión típica y más que son importantes para todos los usuarios de Python. Sin embargo, después de digerir los cambios, descubrirá que Python realmente no ha cambiado tanto; en general, estamos solucionando principalmente molestias y defectos conocidos, y eliminando una gran cantidad de viejas cosas.

Este artículo no intenta proporcionar una especificación completa de todas las características nuevas, sino que intenta proporcionar una visión general conveniente. Para obtener más información, se debe consultar la documentación de Python 3.0 y/o los muchos PEP a los que se hace referencia en el texto. Si se desea comprender la lógica completa de implementación y diseño para una característica determinada, los PEP suelen tener más detalles que la documentación regular, pero téngase en cuenta que los PEP generalmente no se mantienen actualizados una vez que una característica se ha implementado completamente.

Debido a restricciones de tiempo, este documento no es tan completo como debería haber sido. Como siempre para una nueva versión, el archivo `Misc/NEWS` en la distribución de origen contiene una gran cantidad de información detallada sobre cada pequeña cosa que se cambió.

Escollos comunes

Esta sección contiene esos pequeños cambios que probablemente generarán tropiezos si se está acostumbrado a Python 2.5.

Print es una función

La declaración `print` se ha sustituido por una función **`print()`**, con argumentos de palabra clave para reemplazar la mayor parte de la sintaxis especial de la antigua declaración `print` (**PEP 3105** [https://peps.python.org/pep-3105/]). Ejemplos:

```
Old: print "The answer is", 2*2
New: print("The answer is", 2*2)

Old: print x,           # Trailing comma suppresses newl
New: print(x, end=" ")  # Appends a space instead of a r

Old: print              # Prints a newline
New: print()            # You must call the function!

Old: print >>sys.stderr, "fatal error"
New: print("fatal error", file=sys.stderr)

Old: print (x, y)       # prints repr((x, y))
New: print((x, y))      # Not the same as print(x, y)!
```

También se puede personalizar el separador entre ítems, por ejemplo:

```
print("There are <", 2**32, "> possibilities!", sep="")
```

que genera:

```
¡Hay <4294967296> posibilidades!
```

Nota:

- La función **`print()`** no soporta la característica «softspace» de la declaración `print` vieja. Por ejemplo, en Python 2.x `print "A\n", "B"` escribiría `"A\nB\n"`; pero en Python 3.0, `print("A\n", "B")` escribe `"A\n B\n"`.

- Inicialmente, te encontrarás escribiendo mucho la antigua `print x` en modo interactivo. ¡Es hora de volver a entrenar los dedos para escribir `print(x)` en su lugar!
- En el uso de la herramienta de conversión fuente-a-fuente `2to3`, todos las declaraciones `print` son automáticamente convertidas a una llamada de la función **`print()`**, por lo que en general esto no es un problema para proyectos más grandes.

Vistas e iteradores en lugar de listas

Algunas APIs bien conocidas no retornan más listas:

- Los métodos **`dict`** **`dict.keys()`**, **`dict.items()`** y **`dict.values()`** retornan «vistas» en lugar de listas. Por ejemplo, esto no funciona más: `k = d.keys(); k.sort()`. Se usa `k = sorted(d)` en su lugar (esto funciona también en Python 2.5 y es igual de eficiente).

- Also, the **`dict.iterkeys()`**, **`dict.iteritems()`** and **`dict.itervalues()`** methods are no longer supported.

- **`map()`** y **`filter()`** retornan iteradores. Si realmente se necesita una lista y las secuencias de entrada son todas de igual longitud, una solución rápida es envolver **`map()`** en **`list()`**, por ejemplo `list(map(...))`, pero una mejor solución es a menudo utilizar una lista por comprensión (especialmente cuando el código original utiliza **`lambda`**), o reescribir el código para que no necesite una lista en absoluto. Particularmente complicado es **`map()`** invocado para los efectos secundarios de la función; la transformación correcta es usar un bucle **`for`** normal (ya que crear una lista sería simplemente un desperdicio).

  Si las secuencias de entrada no tienen la misma longitud, **`map()`** se detendrá en la terminación de la más corta de las secuencias. Para una compatibilidad completa con **`map()`** de Python 2.x, también se envuelve las secuencias en **`itertools.zip_longest()`**, por ejemplo `map(func, *sequences)` se convierte en `list(map(func, itertools.zip_longest(*sequences)))`.

- `range()` now behaves like `xrange()` used to behave, except it works with values of arbitrary size. The latter no longer exists.
- `zip()` ahora retorna un iterador.

Comparaciones de ordenamiento

Python 3.0 ha simplificado las reglas para las comparaciones de ordenamiento:

- Los operadores de comparaciones de ordenamiento (<, <,``”>``, >) lanzan una excepción *TypeError* cuando los operandos no tienen un orden natural significativo. Por lo tanto, expresiones como `1 < ''`, `0 > None` o `len <= len` ya no son válidas, y por ejemplo `Ninguno < Ninguno` lanza `TypeError` en lugar de retornar `False`. Un corolario es que ordenar una lista heterogénea ya no tiene sentido – todos los elementos deben ser comparables entre sí. Tener en cuenta que esto no se aplica a los operadores `==` y `!=`: los objetos de diferentes tipos incomparables siempre se comparan desiguales entre sí.
- `sorted()` and `list.sort()` no longer accept the *cmp* argument providing a comparison function. Use the *key* argument instead. N.B. the *key* and *reverse* arguments are now «keyword-only».
- The `cmp()` function should be treated as gone, and the `__cmp__()` special method is no longer supported. Use `__lt__()` for sorting, `__eq__()` with `__hash__()`, and other rich comparisons as needed. (If you really need the `cmp()` functionality, you could use the expression `(a > b) - (a < b)` as the equivalent for `cmp(a, b)`.)

Enteros

- PEP 237 [https://peps.python.org/pep-0237/]: Essentially, `long` renamed to `int`. That is, there is only one built-in integral type, named `int`; but it behaves mostly like the old `long` type.
- PEP 238 [https://peps.python.org/pep-0238/]: Una expresión como

`1/2` retorna un float. Utilizar `1//2` para obtener el comportamiento de truncamiento. (Esta última sintaxis ha existido durante años, al menos desde Python 2.2.)
- The **`sys.maxint`** constant was removed, since there is no longer a limit to the value of integers. However, **`sys.maxsize`** can be used as an integer larger than any practical list or string index. It conforms to the implementation's «natural» integer size and is typically the same as **`sys.maxint`** in previous releases on the same platform (assuming the same build options).
- El **`repr()`** de un entero largo ya no incluye la `L` final, por lo que el código que elimina incondicionalmente ese carácter cortará el último dígito en su lugar. (Utilizar **`str()`** en su lugar.)
- Los literales octales ya no usan la forma `0720`; úsese `0o720` en su lugar.

Texto vs. datos en lugar de unicode vs. 8 bits

Todo lo que pensabas saber sobre datos binarios y Unicode ha cambiado.

- Python 3.0 utiliza los conceptos de *text* y *datos* (binarios) en lugar de cadenas de caracteres Unicode y cadenas de caracteres de 8 bits. Todo el texto es Unicode; sin embargo el Unicode *codificado* se representa como datos binarios. El tipo utilizado para contener texto es **`str`**, el tipo utilizado para contener datos es **`bytes`**. La mayor diferencia con la situación 2.x es que cualquier intento de mezclar texto y datos en Python 3.0 lanza un **`TypeError`**, mientras que si se mezclan cadenas Unicode y de 8 bits en Python 2.x, funcionaría si la cadena de 8 bits contuviera sólo bytes de 7 bits (ASCII), pero se obtendría un **`UnicodeDecodeError`** si contenía valores no ASCII. Este comportamiento específico del valor ha causado numerosas caras tristes a lo largo de los años.
- As a consequence of this change in philosophy, pretty much all code that uses Unicode, encodings or binary data most likely has to change. The change is for the better, as in the 2.x world there were numerous bugs having to do with mixing

encoded and unencoded text. To be prepared in Python 2.x, start using **`unicode`** for all unencoded text, and **`str`** for binary or encoded data only. Then the `2to3` tool will do most of the work for you.

- Ya no se puede utilizar literales `u"..."` para texto Unicode. Sin embargo, se debe usar literales `b"..."` para los datos binarios.
- Como los tipos **`str`** y **`bytes`** no se pueden mezclar, siempre se deben convertir explícitamente entre ellos. Utilizar **`str.encode()`** para pasar de **`str`** a **`bytes`**, y **`bytes.decode()`** para pasar de **`bytes`** a **`str`**. También se puede utilizar `bytes(s, encoding=...)` y `str(b, encoding=...)`, respectivamente.
- Like **`str`**, the **`bytes`** type is immutable. There is a separate *mutable* type to hold buffered binary data, **`bytearray`**. Nearly all APIs that accept **`bytes`** also accept **`bytearray`**. The mutable API is based on **`collections.MutableSequence`**.
- Todas las barras invertidas en literales de cadena de caracteres sin formato se interpretan literalmente. Esto significa que los escapes `'\U'` y `'\u'` en cadenas de caracteres sin formato no se tratan especialmente. Por ejemplo, `r'\u20ac'` es una cadena de 6 caracteres en Python 3.0, mientras que en 2.6, `ur'\u20ac'` era el único carácter «euro». (Por supuesto, este cambio sólo afecta a los literales de cadena de caracteres sin formato; el carácter del euro es `'\u20ac'` en Python 3.0.)
- The built-in **`basestring`** abstract type was removed. Use **`str`** instead. The **`str`** and **`bytes`** types don't have functionality enough in common to warrant a shared base class. The `2to3` tool (see below) replaces every occurrence of **`basestring`** with **`str`**.
- Los archivos abiertos como archivos de texto (todavía el modo predeterminado para **`open()`**) siempre utilizan una codificación para asignar entre cadenas de caracteres(en memoria) y bytes (en disco). Los archivos binarios (abiertos con una `b` en el argumento modo) siempre utilizan bytes en la memoria. Esto significa que si un archivo se abre utilizando un modo o codificación incorrectos, es probable que la E/S falle ruidosamente, en lugar de producir datos incorrectos de

forma silenciosa. También significa que incluso los usuarios de Unix tendrán que especificar el modo correcto (texto o binario) al abrir un archivo. Hay una codificación predeterminada dependiente de la plataforma, que en las plataformas Unix se puede establecer con la variable de entorno `LANG` (y a veces también con algunas otras variables de entorno relacionadas con la configuración regional específicas de la plataforma). En muchos casos, pero no en todos, el valor predeterminado del sistema es UTF-8; nunca se debe contar con este valor predeterminado. Cualquier aplicación que lea o escriba más que texto ASCII puro probablemente debería tener una manera de invalidar la codificación. Ya no es necesario utilizar las secuencias compatibles con la codificación en el módulo `codecs`.

- Los valores iniciales de `sys.stdin`, `sys.stdout` y `sys.stderr` ahora son archivos de texto solo unicode (es decir, son instancias de `io.TextIOBase`). Para leer y escribir datos de bytes con estas secuencias, se debe usar su atributo `io.TextIOBase.buffer`.
- Los nombres de archivos son pasados y retornados de cadenas desde APIs como cadenas de caracteres (Unicode). Esto puede presentar problemas específicos de plataformas porque en algunas plataformas los nombres de archivos son cadenas de caracteres de bytes arbitrarios. (Por otro lado, en Windows los nombres de archivos son almacenados de forma nativa como Unicode.) Como solución alternativa, la mayoría de las APIs (por ejemplo `open()` y muchas otras funciones en el módulo `os`) que toman nombres de archivos aceptan tanto objetos `bytes` como cadenas de caracteres, y algunas APIs tienen una forma de demandar por un valor de retorno `bytes`. Por lo tanto, `os.listdir()` retorna una lista de instancias `bytes` si el argumento es una instancia `bytes`, y `os.getcwdb()` retorna el directorio de trabajo actual como una instancia `bytes`. Tener en cuenta que cuando `os.listdir()` retorna una lista de cadenas de caracteres, los nombres de archivo que no se pueden decodificar correctamente se omiten en lugar de lanzar `UnicodeError`.
- Algunas APIs del sistema como `os.environ` y `sys.argv` también pueden presentar problemas cuando los bytes puestos a disposición por el sistema no son interpretables

usando la codificación predeterminada. Probablemente el mejor abordaje sea asignando la variable `LANG` y volviendo a ejecutar el programa.

- **PEP 3138** [https://peps.python.org/pep-3138/]: El `repr()` de una cadena de caracteres ya no escapa caracteres no ASCII. Sin embargo, todavía escapa caracteres de control caracteres y puntos de código con estado no imprimible en el estándar the Unicode.
- **PEP 3120** [https://peps.python.org/pep-3120/]: La codificación de fuente predeterminada ahora es UTF-8.
- **PEP 3131** [https://peps.python.org/pep-3131/]: Letras no ASCII ahora están permitidas en identificadores. (De todas maneras, la librería estándar permanece como sólo ASCII con la excepción de nombres de colaboradores en comentarios.)
- Los módulos `StringIO` and `cStringIO` ya no están. En su lugar, se importa el módulo the `io` y se usa `io.StringIO` o `io.BytesIO` para texto y datos respectivamente.
- Ver también CÓMO (HOWTO) Unicode, que fue actualizado para Python 3.0.

Descripción general de los cambios de sintaxis

Esta sección brinda una descripción general breve de cada cambio *sintáctico* en Python 3.0.

Nueva sintaxis

- **PEP 3107** [https://peps.python.org/pep-3107/]: Function argument and return value annotations. This provides a standardized way of annotating a function’s parameters and return value. There are no semantics attached to such annotations except that they can be introspected at runtime using the `__annotations__` attribute. The intent is to encourage experimentation through metaclasses, decorators or frameworks.

- **PEP 3102** [https://peps.python.org/pep-3102/]: Argumentos de sólo palabra clave. Los parámetros con nombre que se producen después de `*args` en la lista de parámetros *deben* especificarse mediante la sintaxis de palabra clave en la llamada. También puede usarse apenas un `*` sólo en la lista de parámetros para indicar que no se acepta una lista de argumentos de longitud variable, pero se tiene argumentos de sólo palabra clave.

- Los argumentos de palabra clave se permiten después de la lista de clases base en una definición de clase. Esto lo usa la nueva convención para especificar una metaclase (consultar la sección siguiente), pero también se puede usar para otros fines, siempre y cuando la metaclase la admita.

- **PEP 3104** [https://peps.python.org/pep-3104/]: Declaración **`nonlocal`**. Utilizando `nonlocal x` ahora se puede asignar directamente a una variable en un entorno externo (pero no global). **`nonlocal`** es una nueva palabra reservada.

- **PEP 3132** [https://peps.python.org/pep-3132/]: Desempaque iterable extendido. Ahora se puede escribir cosas como `a, b, *rest = some_sequence`. E incluso `*rest, a = cosas`. El objeto `rest` es siempre una lista (posiblemente vacía); el lado derecho puede ser cualquier iterable. Ejemplo:

  ```
  (a, *rest, b) = range(5)
  ```

  Esto asigna *a* a `0`, *b* a `4`, y *rest* a `[1, 2, 3]`.

- Comprensiones por diccionario: `{k: v for k, v in cosas}` significa lo mismo que `dict(cosas)`, pero es más flexible. (Esto es **PEP 274** [https://peps.python.org/pep-0274/] reivindicado. :-)

- Asignar los literales, por ejemplo, `{1, 2}`. Tener en cuenta que `{}` es un diccionario vacío; utilizar `set()` para un conjunto vacío. También se admiten las comprensiones de conjunto; por ejemplo, `x for x in cosas` significa lo mismo que `set(cosas)`, pero es más flexible.

- Nuevos octales literales, por ejemplo, `0o720` (ya en 2.6). Los octales literales viejos (`0720`) se eliminaron.

- Nuevos literales binarios, por ejemplo, `0b1010` (ya en 2.6), y hay una nueva función incorporada correspondiente, `bin()`.
- Los literales de bytes se introducen con un `b` o `B` delantero, y hay una nueva función incorporada correspondiente, `bytes()`.

Sintaxis modificada

- **PEP 3109** [https://peps.python.org/pep-3109/] y **PEP 3134** [https://peps.python.org/pep-3134/]: Nueva sintaxis de la instrucción `raise`: `raise [expr [from expr]]`. Ver a continuación.
- `as` y `with` ahora son palabras reservadas. (Desde 2.6, de hecho.)
- `True`, `False` y `None` son palabras reservadas. (2.6 parcialmente estableció restricciones ya en `None`.)
- Cambio de `except` *exc*, *var* a `except` *exc* `as` *var*.Ver **PEP 3110** [https://peps.python.org/pep-3110/].
- **PEP 3115** [https://peps.python.org/pep-3115/]: Nueva sintaxis de metaclase. En lugar de:

  ```
  clase C:
  __metaclass__ = M
  ...
  ```

  ahora se debe usar:

  ```
  clase C(metaclase=M):
  ...
  ```

  The module-global `__metaclass__` variable is no longer supported. (It was a crutch to make it easier to default to new-style classes without deriving every class from `object`.)
- Las listas por comprensión ya no admiten la forma sintáctica `[... for var in item1, item2, ...]`. Utilizar `[... forvar in (item1, item2, ...)]` en su lugar.

También tener en cuenta que las listas por comprensión tienen semántica diferente: están más cerca del azúcar sintáctico para una expresión de generador dentro de un constructor `list()` y, en particular, las variables de control de bucle ya no se filtran en el ámbito circundante.

- La *elipsis* (`...`) puede ser utilizada como una expresión atómica en cualquier lugar. (Previamente sólo estaba permitida en segmentos.) Asimismo, esto ahora *debe* ser escrito como `...`. (Anteriormente, también podía ser escrito como `. . .`, por un mero accidente de la gramática.)

Sintaxis eliminada

- **PEP 3113** [https://peps.python.org/pep-3113/]: Desempaque de parámetros de tupla eliminados. Ya no se puede escribir `def foo(a, (b, c)): ...`. Utilizar `def foo(a, b_c): b, c = b_c` en su lugar.
- Comillas invertidas eliminadas (usar `repr()` en su lugar).
- Eliminados `<>` (usar `!=` en su lugar).
- Palabra clave eliminada: `exec()` ya no es una palabra clave; continúa siendo una función. (Afortunadamente la sintaxis de la función también era aceptada en 2.x.) Asimismo, nótese que la `exec()` ya no toma un argumento de flujo; en lugar de `exec(f)` se puede utilizar `exec(f.read())`.
- Literales enteros no admiten más `l` o `L` finales.
- Literales de cadena de caracteres no admiten más `u` or `U` iniciales.
- La sintaxis `from` *module* `import *` solo se permite en el nivel de módulo, ya no dentro de las funciones.
- La única sintaxis aceptable para *imports* relativos es `from .[module] import name`. Todos los formularios `import` que no comienzan con `.` son interpretados como *imports* absolutos
- Se eliminaron las clases clásicas.

Cambios ya presentes en Python 2.6

Dado que es de suponer que muchos usuarios saltan directamente de Python 2.5 a Python 3.0, esta sección recuerda al lector nuevas características que se diseñaron originalmente para Python 3.0 pero que fueron llevadas hacia atrás a Python 2.6. Las secciones correspondientes en Qué hay de nuevo en Python 2.6 deberían ser consultadas para descripciones más largas.

- PEP 343: La sentencia "with".La declaración `with` ahora es una característica estándar y ya no necesita ser importada del `__future__`. También puede revisarse Escribiendo gestores de contexto y El módulo contextlib.
- PEP 366: Importaciones relativas explícitas desde un módulo principal. Esto mejora la utilidad de opción `-m` cuando el módulo al que se hace referencia vive en un paquete.
- PEP 370: Directorio de site-packages por usuario.
- PEP 371: El paquete multiprocessing.
- PEP 3101: Formateo avanzado de cadena de caracteres. Nota: la descripción 2.6 menciona el método `format()` tanto para cadenas de caracteres 8 bits como de Unicode. En 3.0, solo el tipo `str` (cadenas de texto con compatibilidad con Unicode) admite este método; el tipo `bytes` no. El plan es a la larga convertir esta en la única API para el formato de cadena de caracteres y comenzar a dejar de utilizar el operador `%` en Python 3.1.
- PEP 3105: print como función. Esta ahora es una característica estándar y ya no necesita ser importada de `__future__`. Más detalles fueron dados anteriormente.
- PEP 3110: Cambios en el manejo de excepciones. La sintaxis `except` *exc* `as` *var* ahora es estándar y `except` *exc*, *var* ya no es compatible. (Por supuesto, la parte `as` *var* sigue siendo opcional.)
- PEP 3112: Literales de bytes. La notación literal de cadena de caracteres `b"..."` (y sus variantes como `b'...'`, `b"""..."""`, y `br"..."`) ahora genera un literal de tipo `bytes`.
- PEP 3116: Nueva biblioteca de E/S. The `io` module is now the standard way of doing file I/O. The built-in `open()` function is now an alias for `io.open()` and has additional keyword arguments *encoding*, *errors*, *newline* and *closefd*. Also note that an invalid *mode* argument now raises `ValueError`, not `IOError`. The binary file object

underlying a text file object can be accessed as `f.buffer` (but beware that the text object maintains a buffer of itself in order to speed up the encoding and decoding operations).
- PEP 3118: Protocolo revisado de la memoria intermedia. The old builtin `buffer()` is now really gone; the new builtin `memoryview()` provides (mostly) similar functionality.
- PEP 3119: Clases base abstractas. The `abc` module and the ABCs defined in the `collections` module plays a somewhat more prominent role in the language now, and built-in collection types like `dict` and `list` conform to the `collections.MutableMapping` and `collections.MutableSequence` ABCs, respectively.
- PEP 3127: Soporte y sintaxis de literales enteros. Como se mencionó anteriormente, la nueva notación literal octal es la única admitida y se han agregado literales binarios.
- PEP 3129: Decoradores de clase.
- PEP 3141: Una jerarquía de tipos para los números. El módulo `numbers` es otro uso nuevo de ABCs, definiendo la «torre numérica» de Python. También tener en cuenta el nuevo módulo `fractions` que implementa `numbers.Rational`.

Cambios de biblioteca

Debido a las limitaciones de tiempo, este documento no cubre exhaustivamente los muy extensos cambios en la biblioteca estándar. **PEP 3108** [https://peps.python.org/pep-3108/] es la referencia para los principales cambios en la biblioteca. Aquí hay una revisión de cápsula:

- Se eliminaron muchos módulos antiguos. Algunos, como `gopherlib` (que ya no se utiliza) y `md5` (reemplazado por `hashlib`), ya estaban obsoletos en **PEP 4** [https://peps.python.org/pep-0004/]. Otros se eliminaron como resultado de la eliminación del soporte para varias plataformas, como Irix, BeOS y Mac OS 9 (consulte **PEP 11** [https://peps.python.org/pep-0011/]). También se seleccionaron algunos módulos para su eliminación en Python 3.0 debido a la falta de uso o porque existe un mejor reemplazo. Consulte **PEP 3108** [https://peps.python.org/pep-3108/] para obtener una lista exhaustiva.

- El paquete **`bsddb3`** fue eliminado porque su presencia en la biblioteca estándar principal ha demostrado con el tiempo ser una carga particular para los desarrolladores principales debido a la inestabilidad de las pruebas y al cronograma de lanzamiento de Berkeley DB. Sin embargo, el paquete está activo y bien, mantenido externamente en https://www.jcea.es/programacion/pybsddb.htm.

- Algunos módulos fueron renombrados porque su antiguo nombre desobedeció **PEP 8** [https://peps.python.org/pep-0008/], o por varias otras razones. Aquí está la lista:

Nombre anterior	Nombre nuevo
_winreg	winreg
ConfigParser	configparser
copy_reg	copyreg
Queue	queue
SocketServer	socketserver
markupbase	_markupbase
repr	reprlib
test.test_support	test.support

- Un patrón común en Python 2.x es tener una versión de un módulo implementado en Python puro, con una versión acelerada opcional implementada como una extensión de C; por ejemplo, **`pickle`** y **`cPickle`**. Esto hace que la carga de importar la versión acelerada y recurrir a la versión Python pura recaiga sobre cada usuario de estos módulos. En Python 3.0, las versiones aceleradas se consideran detalles de implementación de las versiones Python puras. Los usuarios siempre deben importar la versión estándar, que intenta importar la versión acelerada y recurre a la versión Python pura. El par **`pickle`** / **`cPickle`** recibió este tratamiento. El módulo **`profile`** está en la lista para 3.1. El módulo **`StringIO`** se ha convertido en una clase en el módulo **`io`**.

- Algunos módulos relacionados se han agrupado en paquetes y, por lo general, los nombres de submódulo se han simplificado. Los nuevos paquetes resultantes son:

  - **`dbm`** (**`anydbm`**, **`dbhash`**, **`dbm`**, **`dumbdbm`**, **`gdbm`**,

`whichdb`).

  - ○ **`html`** (**`HTMLParser`**, **`htmlentitydefs`**).
  - ○ **`http`** (**`httplib`**, **`BaseHTTPServer`**, **`CGIHTTPServer`**, **`SimpleHTTPServer`**, **`Cookie`**, **`cookielib`**).
  - ○ **`tkinter`** (all `Tkinter`-related modules except **`turtle`**). The target audience of **`turtle`** doesn't really care about **`tkinter`**. Also note that as of Python 2.6, the functionality of **`turtle`** has been greatly enhanced.
  - ○ **`urllib`** (**`urllib`**, **`urllib2`**, **`urlparse`**, **`robotparse`**).
  - ○ **`xmlrpc`** (**`xmlrpclib`**, **`DocXMLRPCServer`**, **`SimpleXMLRPCServer`**).

Algunos otros cambios a los módulos de la biblioteca estándar, no cubiertos por **PEP 3108** [https://peps.python.org/pep-3108/]:

- Quitado **`sets`**. Usar la clase incorporada **`set()`**.
- Cleanup of the **`sys`** module: removed **`sys.exitfunc()`**, **`sys.exc_clear()`**, **`sys.exc_type`**, **`sys.exc_value`**, **`sys.exc_traceback`**. (Note that **`sys.last_type`** etc. remain.)
- Cleanup of the **`array.array`** type: the **`read()`** and **`write()`** methods are gone; use **`fromfile()`** and **`tofile()`** instead. Also, the `'c'` typecode for array is gone – use either `'b'` for bytes or `'u'` for Unicode characters.
- Cleanup of the **`operator`** module: removed **`sequenceIncludes()`** and **`isCallable()`**.
- Limpieza del módulo **`thread`**: **`acquire_lock()`** y **`release_lock()`** han desaparecido; utilice **`acquire()`** y **`release()`** en su lugar.
- Cleanup of the **`random`** module: removed the **`jumpahead()`** API.
- El módulo **`new`** ya no está.
- The functions **`os.tmpnam()`**, **`os.tempnam()`** and **`os.tmpfile()`** have been removed in favor of the **`tempfile`** module.
- El módulo **`tokenize`** se ha cambiado para trabajar con bytes. El punto de entrada principal es ahora

`tokenize.tokenize()`, en lugar de generate_tokens.

- `string.letters` and its friends (`string.lowercase` and `string.uppercase`) are gone. Use `string.ascii_letters` etc. instead. (The reason for the removal is that `string.letters` and friends had locale-specific behavior, which is a bad idea for such attractively named global «constants».)
- Renamed module `__builtin__` to `builtins` (removing the underscores, adding an "s"). The `__builtins__` variable found in most global namespaces is unchanged. To modify a builtin, you should use `builtins`, not `__builtins__`!

PEP 3101 [https://peps.python.org/pep-3101/]: Un nuevo enfoque al formateo de cadena de caracteres

- Un nuevo sistema para operaciones de formateo de cadenas de caracteres incorporadas reemplaza el operador de formato de cadena de caracteres `%`. (De todas maneras, el operador `%` sigue siento soportado; será obsoleto en Python 3.1 y quitado del lenguaje en algún momento más adelante.) Leer PEP 3101 [https://peps.python.org/pep-3101/] para la primicia completa.

Cambios a excepciones

Las APIs para lanzar y capturar excepciones se limpiaron y se agregaron nuevas características poderosas:

- PEP 352 [https://peps.python.org/pep-0352/]: Todas las excepciones deben derivarse (directa o indirectamente) de `BaseException`. Esta es la raíz de la jerarquía de excepciones. Esto no es nuevo como recomendación, pero el *requisito* de heredar de `BaseException` es nuevo. (Python 2.6 aún permitía que las clases clásicas se lanzaran, y no ponía ninguna restricción en lo que se podía capturar.) Como consecuencia, las excepciones de cadena de caracteres finalmente están verdaderamente y completamente muertas.

- Casi todas las excepciones deberían de hecho derivarse de `Exception`; `BaseException` sólo debe utilizarse como clase base para las excepciones que solo deben controlarse en el nivel superior, como `SystemExit` o `KeyboardInterrupt`. El modismo recomendado para controlar todas las excepciones excepto esta última categoría es usar `except Exception`.

- **`StandardError`** was removed.

- Exceptions no longer behave as sequences. Use the `args` attribute instead.

- **PEP 3109** [https://peps.python.org/pep-3109/]: Generación de excepciones. Ahora debe utilizar `raise Exception(args)` en lugar de `raise Exception, args`. Además, ya no puede especificar explícitamente un seguimiento; en su lugar, si desea hacerlo, puede asignarlo directamente al atributo `__traceback__` (consulte a continuación).

- **PEP 3110** [https://peps.python.org/pep-3110/]: Atrapando excepciones. Ahora se debe usar `except AlgunaExcepcion como variable` en lugar de `except AlgunaExcepcion, variable`. Además, la *variable* es específicamente eliminada cuando el bloque `except` se deja.

- **PEP 3134** [https://peps.python.org/pep-3134/]: encadenamiento de excepciones. Existen dos casos: encadenamiento implícito y encadenamiento explícito. El encadenamiento implícito se produce cuando se genera una excepción en un bloque de controlador `except` o `finally`. Esto suele ocurrir debido a un error en el bloque de controlador; lo llamamos excepción *secondary*. En este caso, la excepción original (que se estaba manejando) se guarda como el atributo `__context__` de la excepción secundaria. El encadenamiento explícito se invoca con esta sintaxis:

  ```
  generar SecondaryException() desde primary_exceptio
  ```

(donde *primary_exception* es cualquier expresión que produce un objeto de excepción, probablemente una excepción que se detectó previamente). En este caso, la excepción principal se almacena en el atributo `__cause__` de la excepción secundaria. El seguimiento que se imprime cuando se produce una excepción no controlada recorre la cadena de atributos `__cause__` y `__context__` e imprime un seguimiento independiente para cada componente de la cadena, con la excepción principal en la parte superior. (Los usuarios de Java pueden reconocer este comportamiento).

- **PEP 3134** [https://peps.python.org/pep-3134/]: Los objetos de excepción ahora almacenan su seguimiento como atributo `__traceback__`. Esto significa que un objeto de excepción ahora contiene toda la información relacionada con una excepción y hay menos razones para usar `sys.exc_info()` (aunque este último no se ha eliminado).

- Algunos mensajes de excepción se mejoraron cuando Windows falla al cargar un módulo de extensión. Por ejemplo, `error code 193` ahora es `%1 is not a valid Win32 application`. Las cadenas de caracteres ahora tratan con configuraciones regionales no inglesas.

Otros cambios diversos

Operadores y métodos especiales

- `!=` ahora retorna lo opuesto de `==`, salvo que `==` retorne `NotImplemented`.
- El concepto de «métodos independientes» fue quitado del lenguaje. Cuando se refiere a un método como atributo de clase, ahora se obtiene un objeto de función simple.
- `__getslice__()`, `__setslice__()` and `__delslice__()` were killed. The syntax `a[i:j]` now translates to `a.__getitem__(slice(i, j))` (or `__setitem__()` or `__delitem__()`, when used as an assignment or deletion target, respectively).
- **PEP 3114** [https://peps.python.org/pep-3114/]: El método estándar `next()` se ha renombrado como `__next__()`.

- The `__oct__()` and `__hex__()` special methods are removed – `oct()` and `hex()` use `__index__()` now to convert the argument to an integer.
- Removed support for `__members__` and `__methods__`.
- Los atributos de función denominados `func_X` se han renombrado para utilizar el formato `__X__`, lo que libera estos nombres en el espacio de nombres de atributos de función para los atributos definidos por el usuario. Es decir, `func_closure`, `func_code`, `func_defaults`, `func_dict`, `func_doc`, `func_globals`, `func_name` se renombraron a `__closure__`, `__code__`, `__defaults__`, `__dict__`, `__doc__`, `__globals__`, `__name__`, respectivamente.
- `__nonzero__()` ahora es `__bool__()`.

Incorporados

- PEP 3135 [https://peps.python.org/pep-3135/]: Nuevo `super()`. Ahora se puede invocar `super()` sin argumentos y (suponiendo que se encuentra en un método de instancia normal definido dentro de una declaración `class`) se elegirá automáticamente la clase y la instancia correctas. Con argumentos, el comportamiento de `super()` no cambia.
- PEP 3111 [https://peps.python.org/pep-3111/]: `raw_input()` was renamed to `input()`. That is, the new `input()` function reads a line from `sys.stdin` and returns it with the trailing newline stripped. It raises `EOFError` if the input is terminated prematurely. To get the old behavior of `input()`, use `eval(input())`.
- Se añadió una nueva función incorporada `next()` para llamar al método `__next__()` sobre un objeto.
- La estrategia de redondeo de la función `round()` y el tipo de retorno han cambiado. Los casos exactamente a mitad de camino ahora se redondean al resultado par más cercano en lugar de alejarse de cero. (Por ejemplo, `round(2.5)` ahora retorna “”2”” en lugar de `3`.) `round(x[, n])` ahora delega en `x.__round__([n])` en lugar de retornar siempre un float. Por lo general, retorna un entero cuando se llama con un único argumento y un valor del mismo tipo que `x` cuando se llama con dos argumentos.

- Moved **`intern()`** to **`sys.intern()`**.
- Removed: **`apply()`**. Instead of `apply(f, args)` use `f(*args)`.
- Removed **`callable()`**. Instead of `callable(f)` you can use `isinstance(f, collections.Callable)`. The **`operator.isCallable()`** function is also gone.
- Removed **`coerce()`**. This function no longer serves a purpose now that classic classes are gone.
- Removed **`execfile()`**. Instead of `execfile(fn)` use `exec(open(fn).read())`.
- Removed the **`file`** type. Use **`open()`**. There are now several different kinds of streams that open can return in the **`io`** module.
- Removed **`reduce()`**. Use **`functools.reduce()`** if you really need it; however, 99 percent of the time an explicit **`for`** loop is more readable.
- Removed **`reload()`**. Use **`imp.reload()`**.
- Removed. **`dict.has_key()`** – use the **`in`** operator instead.

Construcción y cambios a la API de C

Debido a restricciones de tiempo, aquí hay una *muy* incompleta lista de cambios a la API de C.

- Se ha eliminado el soporte para varias plataformas, incluyendo pero no limitado a Mac OS 9, BeOS, RISCOS, Irix y Tru64.
- **PEP 3118** [https://peps.python.org/pep-3118/]: Nueva API de búfer.
- **PEP 3121** [https://peps.python.org/pep-3121/]: Inicialización y finalización de módulos de extensión.
- **PEP 3123** [https://peps.python.org/pep-3123/]: Haciendo **`PyObject_HEAD`** conforme al estándar C.
- No más compatibilidad con API C para ejecución restringida.
- Las API C **`PyNumber_Coerce()`**, **`PyNumber_CoerceEx()`**, **`PyMember_Get()`**, y **`PyMember_Set()`** se eliminan.
- La nueva API C **`PyImport_ImportModuleNoBlock()`**, trabaja como **`PyImport_ImportModule()`** pero no bloqueará en el bloqueo de importación (en su lugar, retornará un error).
- Se renombró la conversión booleana de ranura de nivel C y el

método: `nb_nonzero` ahora es `nb_bool`.
- Se eliminó **`METH_OLDARGS`** y **`WITH_CYCLE_GC`** de la API C.

Rendimiento

El resultado neto de las generalizaciones 3.0 es que Python 3.0 ejecuta el punto de referencia *pystone* alrededor de 10% más lenta que Python 2.5. Lo más probable es que la mayor causa sea la eliminación de mayúsculas y minúsculas especiales para enteros pequeños. Hay margen de mejora, ¡pero sucederá después de que se publique 3.0!

Migración a Python 3.0

Para migrar código Python 2.5 o 2.6 existente a Python 3.0, la mejor estrategia es la siguiente:

1. (Prerrequisito:) Comenzar con excelente cobertura de test.
2. Migrar a Python 2.6. Esto no debería representar más trabajo que la migración promedio de Python 2.x a Python 2.(x+1). Asegurarse que todos los test pasen.
3. (Todavía utilizando 2.6:) Activar el modificador de línea de comandos **`-3`**. Esto habilita las advertencias sobre las características que se eliminarán (o cambiarán) en 3.0. Ejecutar el conjunto de pruebas de nuevo y corregir el código sobre el que se recibe advertencias hasta que no queden advertencias y todas las pruebas sigan pasando.
4. Ejecute el traductor de código fuente a código fuente `2to3` sobre su árbol de código fuente. Ejecute el resultado de la traducción en Python 3.0. Corrija manualmente los problemas restantes hasta que se aprueben todas las pruebas nuevamente.

No se recomienda intentar escribir código fuente que se ejecute sin cambios en Python 2.6 y 3.0; se tendría que usar un estilo de codificación muy contorsionado, por ejemplo, evitando las sentencias `print`, metaclases y mucho más. Si se mantiene una biblioteca que necesita soportar Python 2.6 y Python 3.0, el mejor enfoque es modificar el paso 3 anterior editando la versión 2.6 del código fuente y ejecutando el traductor `2to3` de nuevo, en lugar de editar la versión 3.0 del código fuente.

Para migrar las extensiones C a Python 3.0, por favor ver Portar módulos de extensión a Python 3.

Qué hay de nuevo en Python 2.7

Autor:

A.M. Kuchling (amk en amk.ca)

Este articulo explica las nuevas características en Python 2.7. Python 2.7 fue publicado el 3 de Julio de 2010.

El manejo numérico ha sido mejorado en muchas formas, tanto para los números de punto flotante como para la clase `Decimal`. Hay algunas adiciones útiles a la biblioteca estándar, como una gran mejora al modulo `unittest`, el modulo `argparse` para analizar las opciones de la línea de comandos, adecuando las clases `OrderedDict` y `Counter` en el modulo `collections`, y muchas otras mejoras.

Python 2.7 esta previsto que sea el ultimo lanzamiento de la serie 2.x, así que hemos trabajado para que sea una versión a largo plazo. Para ayudar con la migración a Python 3, se han incluido varias características nuevas de la serie Python 3.x en 2.7.

Este articulo no intenta proporcionar una especificación completa de las nuevas características, sino que proporciona una visión general conveniente. Para obtener mas información, debería consultar la documentación de Python 2.7 en https://docs.python.org. Si desea comprender la justificación para el diseño y la implementación, consultar el PEP para una nueva característica particular o el error en el que se discutió un cambio en https://bugs.python.org. Siempre que sea posible, «Que hay de nuevo en Python» enlaza el elemento error/parche para cada cambio.

El futuro de Python 2.x

Python 2.7 es el último lanzamiento importante de la serie 2.x, ya que los mantenedores de Python han cambiado el enfoque de sus esfuerzos en el desarrollo de nuevas características para la serie de Python 3.x. Esto significa que mientras Python 2 continúe recibiendo corrección de errores, y sea actualizado para construir correctamente sobre nuevo hardware y versiones de sistemas operativos compatibles, no habrá nuevas versiones completas de funciones para el idioma o la biblioteca estándar.

Sin embargo, si bien existe un gran subconjunto común entre Python 2.7 y Python 3, y muchos de los cambios involucrados en la migración a ese subconjunto común, o directamente a Python 3, se pueden automatizar de manera segura, algunos otros cambios (en particular los asociados con el manejo de Unicode) puede requerir una consideración cuidadosa, y preferiblemente conjuntos de pruebas de regresión automatizada robustos, para migrar de manera efectiva.

Esto significa que Python 2.7 permanecerá en su lugar durante un largo tiempo, proporcionando una plataforma base estable y compatible para sistemas de producción que aún no se han adaptado a Python 3. La expectativa completa del ciclo de vida de la versión de Python 2.7 esta detallada en **PEP 373** [https://peps.python.org/pep-0373/].

Algunas consecuencias clave de la importancia a largo plazo de 2.7 son:

- Como se señalo anteriormente, la versión 2.7 tiene un período de mantenimiento mucho más largo en comparación con las versiones anteriores 2.x. Actualmente se espera que Python 2.7 siga siendo compatible con el equipo de desarrollo central (recibiendo actualizaciones de seguridad y otras correcciones de errores) al menos hasta 2020 (10 años después de su lanzamiento inicial, comparado con el típico periodo de mantenimiento de 18–24 meses).
- A medida que la biblioteca estándar de Python envejece, hacer un uso eficaz del índice de paquetes de Python (ya sea directamente o a través de un distribuidor) se vuelve mas importante para los usuarios de Python 2. Además de una amplia variedad de paquetes de terceros para diversas tareas, los paquetes disponibles incluyen *backports* de nuevos

módulos y características de la biblioteca estándar de Python 3 que son compatibles con Python 2, así como varias herramientas y librerías que pueden hacer mas fácil la migración a Python 3. La Guía de usuario de paquetes de Python [https://packaging.python.org] proporciona un información sobre como descargar e instalar software desde el índice de paquetes de Python.

- Aunque el enfoque preferido para mejorar Python 2 es ahora la publicación de nuevos paquetes en el índice de paquetes de Python, este enfoque no necesariamente funciona en todos los casos, especialmente aquellos relacionados a la seguridad de la red. En casos excepcionales que no puedan ser manejados adecuadamente mediante la publicación de nuevos paquetes o actualizaciones en PyPI, el proceso de propuestas de mejoras de Python podría ser usado para crear el caso para añadir nuevas funciones a la librería estándar de Python 2. Dichas adiciones, y las versiones de mantenimiento donde fueron agregadas, se observaran en la sección Nuevas funciones añadidas a las versiones de mantenimiento de Python 2.7 mas abajo.

Para los proyectos que deseen migrar de Python 2 a Python 3, o para los desarrolladores de *frameworks* y libreras que deseen dar soporte a usuarios en Python 2 y Python 3, hay una variedad de herramientas y guías disponibles para ayudarles a decidir sobre un enfoque adecuado y manejar algunos de los detalles técnicos involucrados. El punto de partida recomendado es la guía HOWTO How to port Python 2 Code to Python 3.

Cambios en el manejo de las advertencias de desuso

Para Python 2.7, se tomo la decisión de silenciar las advertencias solo de interés para los desarrolladores de manera predeterminada. **``DeprecationWarning`y sus descendientes son ahora ignoradas a menos que se solicite lo contrario, impidiendo a los usuarios estar viendo las advertencias disparadas por una aplicación. Este cambio se realizo en la rama que se convirtió en``**

`Python 3.2. (Discutido en stdlib-sig y llevado a cabo en :issue:`7319`).

En versiones previas, los mensajes **`DeprecationWarning`** estaban habilitados de manera predeterminada, proporcionando a los desarrolladores de Python una indicación clara de donde podría romperse su código en una futura versión principal de Python.

Sin embargo, cada vez hay más usuarios de aplicaciones basadas en Python que no están directamente involucrados en el desarrollo de esas aplicaciones. Los mensajes **`DeprecationWarning`** son irrelevantes para estos usuarios, hacen que se preocupen por una aplicación que realmente está funcionando correctamente y sobrecargando a los desarrolladores de aplicaciones con responder a estas preocupaciones.

Puede volver a habilitar la visualización de los mensajes **`DeprecationWarning`** ejecutando Python con el modificador **`-Wdefault`** (forma corta: **`-Wd`**), o estableciendo la variable de entorno **`PYTHONWARNINGS`** en `"default"` (o `"d"`) antes de ejecutar Python. El código Python también puede volver a habilitarlos llamando a `warnings.simplefilter('default')`.

El módulo `unittest` también vuelve a instalar automáticamente las advertencias de desuso al ejecutar pruebas.

Características de Python 3.1

Al igual que Python 2.6 incorporó características de Python 3.0, la versión 2.7 incorpora algunas de las nuevas características de Python 3.1. La serie 2.x continúa proporcionando herramientas para migrar a la serie 3.x.

Una lista parcial de características 3.1 que se retro importaron a 2.7:

- La sintaxis para establecer literales (`{1,2,3}` es un conjunto mutable).
- Diccionario y establecimiento de compresiones (`{i: i*2 for i in range(3)}`).

- Múltiples administradores de contexto en una sola declaración `with`.
- Una nueva versión para la librería `io`, rescrito en C para un mejor desempeño.
- El tipo de ordenador de diccionario descrito en PEP 372: Adición de un diccionario ordenado a las colecciones.
- El nuevo especificador de formato `","` descrito en PEP 378: Especificador de formato para separador de miles.
- El objeto `memoryview`.
- Un pequeño subconjunto del modulo `importlib`, descrito abajo.
- La `repr()` de un flotante `x` es más corto en muchos casos: ahora se basa en la cadena decimal más corta que se garantiza que redondea a `x`. Como en versiones anteriores de Python, se garantiza que `float(repr(x))` retorne `x`.
- Las conversiones flotante a cadena y cadena a flotante se redondean correctamente. La función `round()` también ahora se redondea correctamente.
- El tipo `PyCapsule`, utilizado para proporcionar una API de C para módulos de extensión.
- La función de API de C `PyLong_AsLongAndOverflow()`.

Otras nuevas advertencias en modo Python3 incluyen:

- `operator.isCallable()` y `operator.sequenceIncludes()`, que no se admiten en 3.x, ahora activan advertencias.
- El modificador `-3` ahora habilita automáticamente el modificador `-Qwarn` que provoca advertencias sobre el uso de la división clásica con enteros y enteros largos.

PEP 372: Adición de un diccionario ordenado a las colecciones

Los diccionarios de python normales recorren en iteración los pares clave/valor en un orden arbitrario. A lo largo de los años, varios autores han escrito implementaciones alternativas que recuerdan el orden en que se insertaron originalmente las claves. Basado en las experiencias de esas implementaciones, 2.7 presenta una nueva clase **`OrderedDict`** en el módulo **`collections`**.

La API **`OrderedDict`** proporciona la misma interfaz que los diccionarios normales, pero recorre en iteración las claves y los valores en un orden garantizado en función de cuándo se insertó una clave por primera vez:

```
>>> from collections import OrderedDict
>>> d = OrderedDict([('first', 1),
...                  ('second', 2),
...                  ('third', 3)])
>>> d.items()
[('first', 1), ('second', 2), ('third', 3)]
```

Si una nueva entrada sobrescribe una entrada existente, la posición de inserción original no se modifica:

```
>>> d['second'] = 4
>>> d.items()
[('first', 1), ('second', 4), ('third', 3)]
```

Borrando una entrada y luego reinsertándola, moverá la misma al final:

```
>>> del d['second']
>>> d['second'] = 5
>>> d.items()
[('first', 1), ('third', 3), ('second', 5)]
```

El método **`popitem()`** tiene un argumento *last* opcional que por defecto esta en `True` (verdadero). Si *last* es verdadero, la ultima llave ingresada es regresada y eliminada; si es falso, se selecciona la llave más antigua:

```
>>> od = OrderedDict([(x,0) for x in range(20)])
>>> od.popitem()
(19, 0)
>>> od.popitem()
(18, 0)
>>> od.popitem(last=False)
```

```
(0, 0)
>>> od.popitem(last=False)
(1, 0)
```

La comparación de dos diccionarios ordenados comprueba tanto las claves como los valores, y requiere que el orden de inserción sea el mismo:

```
>>> od1 = OrderedDict([('first', 1),
...                        ('second', 2),
...                        ('third', 3)])
>>> od2 = OrderedDict([('third', 3),
...                        ('first', 1),
...                        ('second', 2)])
>>> od1 == od2
False
>>> # Move 'third' key to the end
>>> del od2['third']; od2['third'] = 3
>>> od1 == od2
True
```

Comparando a un `OrderedDict` con un diccionario normal ignora el orden de inserción y simplemente compara las llaves y los valores.

¿Cómo funciona `OrderedDict`? Mantiene una lista doblemente enlazada de claves y añade nuevas claves a la lista a medida que se insertan. Un diccionario secundario asigna las claves a su nodo de lista correspondiente, por lo que la eliminación no tiene que recorrer toda la lista enlazada y, por lo tanto, sigue siendo $O(1)$.

La biblioteca estándar ahora admite el uso de diccionarios ordenados en varios módulos.

- El módulo `ConfigParser` los utiliza de forma predeterminada, lo que significa que ahora los archivos de configuración se pueden leer, modificar y luego volver a escribir en su orden original.
- El método `_asdict()` para `collections.namedtuple()` ahora devuelve un

diccionario ordenado con los valores que aparecen en el mismo orden que los índices de tupla subyacentes.

- El constructor de clase `JSONDecoder` del módulo `json` se amplió con un parámetro *object_pairs_hook* para permitir que el decodificador construya instancias `OrderedDict`. También se agregó soporte para herramientas de terceros como PyYAML [https://pyyaml.org/].

Ver también

PEP 372 [https://peps.python.org/pep-0372/] - Adición de un diccionario ordenado a las *collections*

PEP escrito por Armin Ronacher y Raymond Hettinger; implementado por Raymond Hettinger.

PEP 378: Especificador de formato para separador de miles

Para que la salida del programa sea más legible, puede ser útil agregar separadores a números grandes, representándolos como 18,446,744,073,709,551,616 en lugar de 18446744073709551616.

La solución totalmente general para hacer esto es el módulo `locale`, que puede utilizar diferentes separadores («,» en América del Norte, «.» en Europa) y diferentes tamaños de agrupación, pero `locale` es complicado de usar y no es adecuado para aplicaciones multiproceso donde diferentes hilos están produciendo resultados para diferentes configuraciones regionales.

Por lo tanto, se ha añadido un mecanismo simple de agrupación de comas al mini-lenguaje utilizado por el método `str.format()`. Al dar formato a un número de punto flotante, simplemente incluya una coma entre el ancho y la precisión:

```
>>> '{:20,.2f}'.format(18446744073709551616.0)
'18,446,744,073,709,551,616.00'
```

Al dar formato a un entero, incluya la coma después del ancho:

```
>>> '{:20,d}'.format(18446744073709551616)
'18,446,744,073,709,551,616'
```

Este mecanismo no es adaptable en absoluto; Las comas siempre se utilizan como separador y la agrupación siempre está en grupos de tres dígitos. El mecanismo de formato de coma no es tan general como el módulo `locale`, pero es más fácil de usar.

Ver también

PEP 378 [https://peps.python.org/pep-0378/] - Especificador de formato para separador de miles
PEP escrito por Raymond Hettinger; implementado por Eric Smith.

PEP 389: El módulo argparse para el análisis de líneas de comando

El módulo `argparse` para analizar argumentos de línea de comandos se agregó como un reemplazo más potente para el módulo `optparse`.

Esto significa que Python ahora admite tres módulos diferentes para analizar argumentos de línea de comandos: `getopt`, `optparse` y `argparse`. El módulo `getopt` se parece mucho a la función `getopt()` de la biblioteca C, por lo que sigue siendo útil si está escribiendo un prototipo de Python que eventualmente se reescribirá en C. `optparse` se vuelve redundante, pero no hay planes de eliminarlo porque hay muchos scripts que aún lo usan y no hay una forma automatizada de actualizar estos scripts. (Se discutió hacer que la API `argparse` sea consistente con la interfaz de `optparse`, pero se rechazó por ser demasiado desordenada y difícil).

En resumen, si estás escribiendo un nuevo script y no necesitas preocuparte por la compatibilidad con versiones anteriores de Python, usa `argparse` en lugar de `optparse`.

Este es un ejemplo:

```
import argparse

parser = argparse.ArgumentParser(description='Command-li

# Add optional switches
parser.add_argument('-v', action='store_true', dest='is_
                    help='produce verbose output')
parser.add_argument('-o', action='store', dest='output',
                    metavar='FILE',
                    help='direct output to FILE instead
parser.add_argument('-C', action='store', type=int, dest
                    metavar='NUM', default=0,
                    help='display NUM lines of added con

# Allow any number of additional arguments.
parser.add_argument(nargs='*', action='store', dest='inp
                    help='input filenames (default is st

args = parser.parse_args()
print args.__dict__
```

A menos que lo reemplace, los modificadores **`-h`** y **`--help`** se agregan automáticamente, y producen una salida con formato ordenado:

```
-> ./python.exe argparse-example.py --help
usage: argparse-example.py [-h] [-v] [-o FILE] [-C NUM]

Command-line example.

positional arguments:
  inputs      input filenames (default is stdin)

optional arguments:
  -h, --help  show this help message and exit
  -v          produce verbose output
  -o FILE     direct output to FILE instead of stdout
  -C NUM      display NUM lines of added context
```

Al igual que con `optparse`, los modificadores y argumentos de línea de comandos se retornan como un objeto con atributos denominados por los parámetros *dest*:

```
-> ./python.exe argparse-example.py -v
{'output': None,
 'is_verbose': True,
 'context': 0,
 'inputs': []}

-> ./python.exe argparse-example.py -v -o /tmp/output -C
{'output': '/tmp/output',
 'is_verbose': True,
 'context': 4,
 'inputs': ['file1', 'file2']}
```

`argparse` tiene una validación mucho más fantasiosa que `optparse`; puede especificar un número exacto de argumentos como un entero, 0 o más argumentos pasando `'*'`, 1 o más pasando `'+'`, o un argumento opcional con `'?'`. Un analizador de nivel superior puede contener sub analizadores para definir subcomandos que tienen diferentes conjuntos de modificadores, como en `svn commit`, `svn checkout`, etc. Puede especificar el tipo de un argumento como `FileType`, que abrirá automáticamente los archivos y entiende que `'-'` significa entrada o salida estándar.

Ver también

`argparse` documentación

La página de documentación del módulo argparse.

Migrating optparse code to argparse

Parte de la documentación de Python, que describe cómo convertir código que usa `optparse`.

PEP 389 [https://peps.python.org/pep-0389/] - argparse - Nuevo módulo de análisis de línea de comandos

PEP escrito e implementado por Steven Bethard.

PEP 391: Configuración basada en diccionarios para el registro

El módulo `logging` es muy flexible; las aplicaciones pueden definir un árbol de subsistemas de registro, y cada registrador de este árbol puede filtrar ciertos mensajes, formatearlos de forma diferente y dirigir mensajes a un número variable de controladores.

Toda esta flexibilidad puede requerir mucha configuración. Puedes escribir instrucciones Python para crear objetos y establecer sus propiedades, pero una configuración compleja requiere un código extenso pero aburrido. `logging` también admite una función `fileConfig()` que analiza un archivo, pero el formato de archivo no admite la configuración de filtros y es más complicado generarlo mediante programación.

Python 2.7 agrega una función `dictConfig()` que utiliza un diccionario para configurar el registro. Hay muchas maneras de producir un diccionario a partir de diferentes fuentes: construir uno con código; analizar un archivo que contenga JSON; o utilizar una biblioteca de análisis de YAML si hay una instalada. Para obtener más información, consulte Funciones de configuración.

En el ejemplo siguiente se configuran dos registradores, el registrador raíz y un registrador denominado «network». Los mensajes enviados al registrador raíz se enviarán al registro del sistema mediante el protocolo syslog, y los mensajes al registrador de la «network» se escribirán en un archivo `network.log` que se rotará una vez que el registro alcance 1MB.

```
import logging
import logging.config

configdict = {
 'version': 1,    # Configuration schema in use; must be
 'formatters': {
     'standard': {
         'format': ('%(asctime)s %(name)-15s '
                    '%(levelname)-8s %(message)s')}},
 'handlers': {'netlog': {'backupCount': 10,
                     'class': 'logging.handlers.Rotating
                     'filename': '/logs/network.log',
                     'formatter': 'standard',
                     'level': 'INFO',
                     'maxBytes': 1000000},
              'syslog': {'class': 'logging.handlers.SysL
                         'formatter': 'standard',
                         'level': 'ERROR'}},

 # Specify all the subordinate loggers
 'loggers': {
             'network': {
                         'handlers': ['netlog']
             }
 },
 # Specify properties of the root logger
 'root': {
          'handlers': ['syslog']
 },
}

# Set up configuration
logging.config.dictConfig(configdict)

# As an example, log two error messages
logger = logging.getLogger('/')
logger.error('Database not found')

netlogger = logging.getLogger('network')
netlogger.error('Connection failed')
```

Tres mejoras más pequeñas en el módulo `logging`, todas implementadas por Vinay Sajip, son:

- La clase `SysLogHandler` ahora admite syslogging a través de TCP. El constructor tiene un parámetro *socktype* que proporciona el tipo de socket que se va a usar, ya sea `socket.SOCK_DGRAM` para UDP o `socket.SOCK_STREAM` para TCP. El protocolo predeterminado sigue siendo UDP.

- Las instancias `Logger` ganaron un método `getChild()` que recupera un registrador descendiente mediante una ruta de acceso relativa. Por ejemplo, una vez que se recupera un registrador haciendo `log = getLogger('app')`, llamando `log.getChild('network.listen')` es equivalente a `getLogger('app.network.listen')`.
- La clase `LoggerAdapter` obtuvo un método `isEnabledFor()` que toma un *level* y devuelve si el registrador subyacente procesaría un mensaje de ese nivel de importancia.

Ver también

PEP 391 [https://peps.python.org/pep-0391/] - Configuración basada en diccionarios para el registro

PEP escrito e implementado por Vinay Sajip.

PEP 3106: Vistas de diccionario

Los métodos de diccionarios `keys()`, `values()` y `items()` son diferentes en Python 3.x. Ellos regresan un objeto llamado *view* en lugar de una lista completamente materializada.

No es posible cambiar los valores de retorno de `keys()`, `values()` y `items()` en Python 2.7 porque demasiado código podría fallar. En su lugar, se agregaron las versiones 3.x con los nuevos nombres **`viewkeys()`**, **`viewvalues()`** y **`viewitems()`**.

```
>>> d = dict((i*10, chr(65+i)) para i en rango(26))
>>> d
{0: 'A', 130: 'N', 10: 'B', 140: 'O', 20: ..., 250: 'Z'
>>> d.viewkeys()
dict_keys([0, 130, 10, 140, 20, 150, 30, ..., 250])
```

Las vistas se pueden iterar, pero las vistas de elementos y claves también se comportan como conjuntos. El operador `&` realiza la intersección y `|` realiza una unión:

```
>>> d1 = dict((i*10, chr(65+i)) for i in range(26))
>>> d2 = dict((i**.5, i) for i in range(1000))
>>> d1.viewkeys() & d2.viewkeys()
set([0.0, 10.0, 20.0, 30.0])
>>> d1.viewkeys() | range(0, 30)
set([0, 1, 130, 3, 4, 5, 6, ..., 120, 250])
```

La vista realiza un seguimiento del diccionario y su contenido cambia a medida que se modifica el diccionario:

```
>>> vk = d.viewkeys()
>>> vk
dict_keys([0, 130, 10, ..., 250])
>>> d[260] = '&'
>>> vk
dict_keys([0, 130, 260, 10, ..., 250])
```

Sin embargo, tenga en cuenta que no puede agregar o eliminar claves mientras está iterando sobre la vista:

```
>>> for k in vk:
...     d[k*2] = k
...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: dictionary changed size during iteration
```

Puede usar los métodos de vista en el código en Python 2.x, y el convertidor 2to3 los cambiará a los métodos estándar `keys()`, `values()` y `items()`.

Ver también

PEP 3106 [https://peps.python.org/pep-3106/] - Renovación dict.keys(), .values() y .items()

PEP escrito por Guido van Rossum. Portado a 2.7 por Alexandre Vassalotti; bpo-1967 [https://bugs.python.org/issue?@action=redirect&bpo=1967].

PEP 3137: El objeto memoryview

El objeto `memoryview` proporciona una vista del contenido de la memoria de otro objeto que coincide con la interfaz de tipo `bytes`.

```
>>> import string
>>> m = memoryview(string.letters)
>>> m
<memory at 0x37f850>
>>> len(m)           # Returns length of underlying obje
52
>>> m[0], m[25], m[26]   # Indexing returns one byte
('a', 'z', 'A')
>>> m2 = m[0:26]         # Slicing returns another memor
>>> m2
<memory at 0x37f080>
```

El contenido de la vista se puede convertir en una cadena de bytes o una lista de números enteros:

```
>>> m2.tobytes()
'abcdefghijklmnopqrstuvwxyz'
>>> m2.tolist()
[97, 98, 99, 100, 101, 102, 103, ... 121, 122]
>>>
```

Los objetos `memoryview` permiten modificar el objeto subyacente si es un objeto mutable.

```
>>> m2[0] = 75
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot modify read-only memory
>>> b = bytearray(string.letters)  # Creating a mutable
>>> b
bytearray(b'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQ
>>> mb = memoryview(b)
>>> mb[0] = '*'         # Assign to view, changing the
>>> b[0:5]              # The bytearray has been change
```

```
bytearray(b'*bcde')
>>>
```

Ver también

PEP 3137 [https://peps.python.org/pep-3137/] - Bytes inmutables y búfer mutable

PEP escrito por Guido van Rossum. Implementado por Travis Oliphant, Antoine Pitrou y otros. Portado a 2.7 por Antoine Pitrou; bpo-2396 [https://bugs.python.org/issue?@action=redirect&bpo=2396].

Otros cambios de lenguaje

Algunos de los cambios mas pequeños hechos al núcleo del lenguaje de Python son:

- La sintaxis para literales de conjuntos se ha actualizado desde Python 3.x. Los corchetes se utilizan para rodear el contenido del conjunto mutable resultante; los literales de conjunto se distinguen de los diccionarios por no contener dos puntos ni valores. `{}` sigue representando un diccionario vacío; use `set()` para un conjunto vacío.

  ```
  >>> {1, 2, 3, 4, 5}
  set([1, 2, 3, 4, 5])
  >>> set() # empty set
  set([])
  >>> {}    # empty dict
  {}
  ```

  Portado por Alexandre Vassalotti; bpo-2335 [https://bugs.python.org/issue?@action=redirect&bpo=2335].

- El diccionario y las comprensiones de conjuntos son otra característica portada desde 3.x, que generaliza las comprensiones de listas / generadores para usar la sintaxis literal para conjuntos y diccionarios.

```
>>> {x: x*x for x in range(6)}
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
>>> {('a'*x) for x in range(6)}
set(['', 'a', 'aa', 'aaa', 'aaaa', 'aaaaa'])
```

Portado por Alexandre Vassalotti; bpo-2333 [https://bugs.python.org/issue?@action=redirect&bpo=2333].

- La declaración **with** ahora puede usar múltiples administradores de contexto en una declaración. Los administradores de contexto se procesan de izquierda a derecha y cada uno se trata como si comenzara una nueva declaración **with**. Esto significa que:

  ```
  with A() as a, B() as b:
      ... suite of statements ...
  ```

  es equivalente a:

  ```
  with A() as a:
      with B() as b:
          ... suite of statements ...
  ```

  La función **contextlib.nested()** proporciona una función muy similar, por lo que ya no es necesaria y ha quedado obsoleta.

  (Propuesta en https://codereview.appspot.com/53094; implementado por Georg Brandl.)

- Las conversiones entre números de punto flotante y cadenas ahora se redondean correctamente en la mayoría de las plataformas. Estas conversiones ocurren en muchos lugares diferentes: **str()** en flotantes y números complejos; los constructores **float** y **complex**; formato numérico; serializar y deserializar flotantes y números complejos usando los módulos **marshal**, **pickle** y **json**; análisis de literales flotantes e imaginarios en código Python; y conversión **Decimal** a flotante.

  Relacionado con esto, el **repr()** de un número de punto flotante *x* ahora retorna un resultado basado en la cadena decimal más corta que se garantiza que se redondeará a *x* con el redondeo correcto (con el modo de redondeo *round-half-to-even*). Anteriormente, daba una cadena basada en redondear x a 17 dígitos decimales.

La biblioteca de redondeo responsable de esta mejora funciona en Windows y en plataformas Unix utilizando los compiladores gcc, icc o suncc. Puede haber un pequeño número de plataformas donde no se puede garantizar el correcto funcionamiento de este código, por lo que el código no se utiliza en dichos sistemas. Puede averiguar qué código se está utilizando marcando **`sys.float_repr_style`**, que será `short` si el nuevo código está en uso y `legacy` si no lo está.

Implementado por Eric Smith y Mark Dickinson, utilizando la biblioteca de David Gay's `dtoa.c`; bpo-7117 [https://bugs.python.org/issue?@action=redirect&bpo=7117].

- Las conversiones de enteros largos y enteros regulares a punto flotante ahora redondean de forma diferente, retornando el número de punto flotante más cercano al número. Esto no importa para los enteros pequeños que se pueden convertir exactamente, pero para los números grandes que inevitablemente perderán precisión, Python 2.7 ahora se aproxima más. Por ejemplo, Python 2.6 calculaba lo siguiente:

  ```
  >>> n = 295147905179352891391
  >>> float(n)
  2.9514790517935283e+20
  >>> n - long(float(n))
  65535L
  ```

  El resultado de punto flotante de Python 2.7 es mayor, pero mucho más cerca del valor verdadero:

  ```
  >>> n = 295147905179352891391
  >>> float(n)
  2.9514790517935289e+20
  ```

```
>>> n - long(float(n))
-1L
```

(Implementado por Mark Dickinson; bpo-3166 [https://bugs.python.org/issue?@action=redirect&bpo=3166].)

La división de enteros también es más precisa en su comportamiento de redondeo. (También implementado por Mark Dickinson; bpo-1811 [https://bugs.python.org/issue?@action=redirect&bpo=1811].)

- Se ha eliminado la coerción implícita para números complejos; el intérprete ya no intentará llamar a un método `__coerce__()` en objetos complejos. (Eliminado por Meador Inge y Mark Dickinson; bpo-5211 [https://bugs.python.org/issue?@action=redirect&bpo=5211]).
- El método `str.format()` soporta ahora la numeración automática de los campos de sustitución. Esto hace que el uso de `str.format()` se asemeje más al uso del formateo `%s`:

  ```
  >>> '{}:{}:{}'.format(2009, 04, 'Sunday')
  '2009:4:Sunday'
  >>> '{}:{}:{day}'.format(2009, 4, day='Sunday')
  '2009:4:Sunday'
  ```

  La numeración automática toma los campos de izquierda a derecha, por lo que el primer especificador `{...}` utilizará el primer argumento de `str.format()`, el siguiente especificador utilizará el siguiente argumento, y así sucesivamente. No se puede mezclar la numeración automática con la explícita - o se numeran todos los campos del especificador o ninguno - pero se puede mezclar la numeración automática con los campos con nombre, como en el segundo ejemplo anterior. (Contribución de Eric Smith; bpo-5237 [https://bugs.python.org/issue?@action=redirect&bpo=5237].)

  Los números complejos ahora soportan correctamente el uso con `format()`, y por defecto están alineados a la derecha. La especificación de una precisión o separación por comas se aplica tanto a la parte real como a la imaginaria del número, pero la anchura de campo y la alineación especificadas se aplican a la totalidad de la salida resultante `1,5+3j`. (Contribución de Eric Smith; bpo-1588 [https://bugs.python.org/issue?@action=redirect&bpo=1588] y bpo-7988 [https://bugs.python.org/issue?@action=redirect&bpo=7988])

El código de formato "F" ahora siempre formatea su salida utilizando caracteres en mayúsculas, por lo que ahora producirá "INF" y "NAN". (Contribución de Eric Smith; bpo-3382 [https://bugs.python.org/issue?@action=redirect&bpo=3382].)

Un cambio de bajo nivel: el método **`object.__format__()`** ahora activa un **`PendingDeprecationWarning`** si se le pasa una cadena de formato, porque el método **`__format__()`** para **`object`** convierte el objeto en una representación de cadena y la formatea. Anteriormente, el método aplicaba silenciosamente la cadena de formato a la representación de cadena, pero eso podía ocultar errores en el código Python. Si estás proporcionando información de formato, como una alineación o precisión, presumiblemente estás esperando que el formato se aplique de alguna manera específica del objeto. (Corregido por Eric Smith; bpo-7994 [https://bugs.python.org/issue?@action=redirect&bpo=7994]).

- Los tipos **`int()`** y **`long()`** obtuvieron un método `bit_length` que devuelve la cantidad de bits necesarios para representar su argumento en binario:

```
>>> n = 37
>>> bin(n)
'0b100101'
>>> n.bit_length()
6
>>> n = 2**123-1
>>> n.bit_length()
123
>>> (n+1).bit_length()
```

(Contribución de Fredrik Johansson y Victor Stinner; bpo-3439 [https://bugs.python.org/issue?@action=redirect&bpo=3439].)

- La instrucción `import` ya no intentará una importación absoluta si se produce un error en una importación relativa (por ejemplo, `from .os import sep`). Esto corrige un error, pero posiblemente podría romper ciertas instrucciones `import` que solo funcionaban por accidente. (Arreglado por Meador Inge; bpo-7902 [https://bugs.python.org/issue?@action=redirect&bpo=7902].)

- Ahora es posible que una subclase del tipo `unicode` incorporado anule el método `__unicode__()`. (Implementado por Victor Stinner; bpo-1583863 [https://bugs.python.org/issue?@action=redirect&bpo=1583863]).

- El método `translate()` del tipo `bytearray` ahora acepta `None` como primer argumento. (Corregido por Georg Brandl; bpo-4759 [https://bugs.python.org/issue?@action=redirect&bpo=4759].)

- When using `@classmethod` and `@staticmethod` to wrap methods as class or static methods, the wrapper object now exposes the wrapped function as their `__func__` attribute. (Contributed by Amaury Forgeot d'Arc, after a suggestion by George Sakkis; bpo-5982 [https://bugs.python.org/issue?@action=redirect&bpo=5982].)

- Cuando se establecía un conjunto restringido de atributos utilizando `__slots__`, la eliminación de un atributo no establecido no lanzaba `AttributeError` como cabría esperar. Corregido por Benjamin Peterson; bpo-7604 [https://bugs.python.org/issue?@action=redirect&bpo=7604])

- Ahora se admiten dos nuevas codificaciones: «cp720», utilizada principalmente para el texto árabe; y «cp858», una variante de CP 850 que añade el símbolo del euro. (CP720 contribuido por Alexander Belchenko y Amaury Forgeot d'Arc en bpo-1616979 [https://bugs.python.org/issue?@action=redirect&bpo=1616979]; CP858 contribuido por Tim Hatch en bpo-8016 [https://bugs.python.org/issue?@action=redirect&bpo=8016])

- El objeto `file` ahora establecerá el atributo `filename` en la excepción `IOError` al intentar abrir un directorio en plataformas POSIX (observado por Jan Kaliszewski; bpo-4764 [https://bugs.python.org/issue?@action=redirect&bpo=4764]), y ahora verifica explícitamente y prohíbe escribir en objetos de archivos de solo lectura en lugar de confiar en la biblioteca C para detectar e informar el error (corregido por Stefan Krah; bpo-5677 [https://bugs.python.org/issue?@action=redirect&bpo=5677]).

- El tokenizador de Python ahora traduce los finales de línea por sí mismo, por lo que la función incorporada `compile()` ahora acepta código que utilice cualquier convención de final de línea. Además, ya no requiere que el código termine en una nueva línea.

- Los paréntesis adicionales en las definiciones de función son ilegales en Python 3.x, lo que significa que se obtiene un error de sintaxis de `def f((x)): pass`. En el modo de advertencia Python3, Python 2.7 ahora advertirá sobre este uso no común. (Anotado por James Lingard; bpo-7362 [https://bugs.python.org/issue?@action=redirect&bpo=7362].)

- Ahora es posible crear referencias débiles a objetos de clase de estilo antiguo. Las clases de estilo nuevo siempre eran de referencia débil. (Corregido por Antoine Pitrou; bpo-8268 [https://bugs.python.org/issue?@action=redirect&bpo=8268].)

- Cuando se recicla un objeto de módulo, el diccionario del módulo ahora solo se borra si nadie más tiene una referencia al diccionario (bpo-7140 [https://bugs.python.org/issue?@action=redirect&bpo=7140]).

Cambios en el intérprete

Una nueva variable de entorno, `PYTHONWARNINGS`, permite controlar las advertencias. Debe establecerse en una cadena que contenga la configuración de advertencia, equivalente a las utilizadas con el modificador `-W`, separado por comas. (Contribuido por Brian Curtin; bpo-7301 [https://bugs.python.org/issue?@action=redirect&bpo=7301].)

Por ejemplo, la siguiente configuración imprimirá advertencias cada vez que se produzcan, pero convertirá las advertencias del módulo `Cookie` en un error. (La sintaxis exacta para configurar una variable de entorno varía según los sistemas operativos y los shells).

```
exportar PYTHONWARNINGS=all,error::Cookie:0
```

Optimizaciones

Se han añadido varias mejoras de rendimiento:

- Se agregó un nuevo código de operación para realizar la configuración inicial de las instrucciones `with` y buscar los métodos `__enter__()` y `__exit__()`. (Contribuido por Benjamin Peterson).
- El recolector de basura ahora funciona mejor para un patrón de uso común: cuando se asignan muchos objetos sin desasignar ninguno de ellos. Antes, la recogida de basura tardaba un tiempo cuadrático, pero ahora el número de recogidas de basura completas se reduce a medida que crece el número de objetos en el montón. La nueva lógica sólo realiza una pasada completa de recogida de basura cuando la generación intermedia ha sido recogida 10 veces y cuando el número de objetos supervivientes de la generación intermedia supera el 10% del número de objetos de la generación más antigua. (Sugerido por Martin von Löwis e implementado por Antoine Pitrou; bpo-4074 [https://bugs.python.org/issue?@action=redirect&bpo=4074].)
- El recolector de basura intenta evitar el seguimiento de contenedores simples que no pueden formar parte de un ciclo. En Python 2.7, esto es ahora cierto para tuplas y diccionarios que contienen tipos atómicos (como enteros, cadenas, etc.). Transitoriamente, un dict que contenga tuplas de tipos

atómicos tampoco será rastreado. Esto ayuda a reducir el coste de cada recogida de basura al disminuir el número de objetos que debe considerar y recorrer el recolector. (Contribución de Antoine Pitrou; bpo-4688 [https://bugs.python.org/issue?@action=redirect&bpo=4688].)

- Los enteros largos se almacenan ahora internamente en base 2**15 o en base 2**30, determinándose la base en el momento de la construcción. Anteriormente, siempre se almacenaban en base 2**15. El uso de la base 2**30 proporciona mejoras significativas en el rendimiento de las máquinas de 64 bits, pero los resultados de las pruebas comparativas en las máquinas de 32 bits han sido contradictorios. Por lo tanto, el valor por defecto es utilizar la base 2**30 en máquinas de 64 bits y la base 2**15 en máquinas de 32 bits; en Unix, hay una nueva opción de configuración **`--enable-big-digits`** que puede utilizarse para anular este valor por defecto.

  Aparte de las mejoras de rendimiento, este cambio debería ser invisible para los usuarios finales, con una excepción: para fines de prueba y depuración, hay un nuevo structseq **`sys.long_info`** que proporciona información sobre el formato interno, dando el número de bits por dígito y el tamaño en bytes del tipo C utilizado para almacenar cada dígito:

  ```
  >>> importar sys
  >>> sys.long_info
  sys.long_info(bits_por_dígito=30, tamaño_del_dígito
  ```

  (Contribución de Mark Dickinson; bpo-4258 [https://bugs.python.org/issue?@action=redirect&bpo=4258].)

  Otro conjunto de cambios hizo que los objetos largos fueran unos pocos bytes más pequeños: 2 bytes menos en sistemas de 32 bits y 6 bytes en los de 64 bits. (Contribución de Mark Dickinson; bpo-5260 [https://bugs.python.org/issue?@action=redirect&bpo=5260].)

- El algoritmo de división de enteros largos se ha hecho más rápido ajustando el bucle interno, haciendo desplazamientos en lugar de multiplicaciones, y arreglando una iteración extra innecesaria. Varias pruebas de referencia muestran un aumento de velocidad de entre el 50% a y el 150% en las divisiones de enteros largos y en las operaciones de módulo. (Contribución de Mark Dickinson; bpo-5512 [https://bugs.python.org/issue?@action=redirect&bpo=5512]). Las operaciones a nivel de bit también son significativamente más rápidas (parche inicial de Gregory Smith; bpo-1087418 [https://bugs.python.org/issue?@action=redirect&bpo=1087418]).

- La implementación de `%` comprueba si el operando de la izquierda es una cadena de Python y lo convierte en un caso especial; esto resulta en un aumento de rendimiento del 1–3% p para aplicaciones que utilizan frecuentemente `%` con cadenas, como las bibliotecas de plantillas. (Implementado por Collin Winter; bpo-5176 [https://bugs.python.org/issue?@action=redirect&bpo=5176].)

- Las comprensiones de listas con una condición `if` se compilan en código de bytes más rápido. (Parche de Antoine Pitrou, portado a la 2.7 por Jeffrey Yasskin; bpo-4715 [https://bugs.python.org/issue?@action=redirect&bpo=4715])

- La conversión de un entero o un entero largo a una cadena decimal se ha hecho más rápida mediante la conversión en base 10 en lugar de utilizar una función de conversión generalizada que soporta bases arbitrarias. (Parche de Gawain Bolton; bpo-6713 [https://bugs.python.org/issue?@action=redirect&bpo=6713].)

- Los métodos **`split()`**, **`replace()`**, **`rindex()`**, **`rpartition()`** y **`rsplit()`** de tipos similares a cadenas (cadenas, cadenas Unicode y objetos **`bytearray`**) ahora utilizan un algoritmo de búsqueda inversa rápida en lugar de un escaneo carácter por carácter. Esto a veces es diez veces más rápido. (Agregado por Florent Xicluna; bpo-7462 [https://bugs.python.org/issue?@action=redirect&bpo=7462] y bpo-7622 [https://bugs.python.org/issue?@action=redirect&bpo=7622]).

- Los módulos **`pickle`** y **`cPickle`** ahora almacenan automáticamente en caché las cadenas utilizadas para los nombres de atributos, lo que reduce el uso de memoria de los objetos resultantes de la descompresión. (Contribuido por Jake McGuire; bpo-5084 [https://bugs.python.org/issue?@action=redirect&bpo=5084]).

- El módulo **`cPickle`** ahora crea casos especiales para los diccionarios, reduciendo casi a la mitad el tiempo necesario para procesarlos. (Contribución de Collin Winter; bpo-5670 [https://bugs.python.org/issue?@action=redirect&bpo=5670].)

Módulos nuevos y mejorados

Como en cada versión, la biblioteca estándar de Python ha recibido una serie de mejoras y correcciones de errores. Aquí hay una lista parcial de los cambios más notables, ordenados alfabéticamente por nombre de módulo. Consulta el archivo `Misc/NEWS` en el árbol de código fuente para una lista más completa de cambios, o mira los registros de Subversion para todos los detalles.

- La clase base de depuración del módulo **`bdb`** **`Bdb`** obtuvo una función para omitir módulos. El constructor ahora toma un iterable que contiene patrones de estilo glob como `django.*`; el depurador no entrará en los marcos de pila de un módulo que coincida con uno de estos patrones. (Aportado por Maru Newby tras una sugerencia de Senthil Kumaran; bpo-5142 [https://bugs.python.org/issue?@action=redirect&bpo=5142].)

- El módulo **`binascii`** soporta ahora la API del buffer, por lo que puede utilizarse con instancias de **`memoryview`** y otros objetos de buffer similares. (Tomado de la 3.x por Florent Xicluna; bpo-7703 [https://bugs.python.org/issue?@action=redirect&bpo=7703].)

- Módulo actualizado: el módulo **`bsddb`** ha sido actualizado desde 4.7.2devel9 a la versión 4.8.4 de the pybsddb package [https://www.jcea.es/programacion/pybsddb.htm]. La nueva versión incluye una mejor compatibilidad con Python 3.x, varias correcciones de errores y añade varios nuevos indicadores y métodos de BerkeleyDB. (Actualizado por Jesús Cea Avión; bpo-8156 [https://bugs.python.org/issue?@action=redirect&bpo=8156]. El registro de cambios de pybsddb se puede leer en https://hg.jcea.es/pybsddb/file/tip/ChangeLog.)

- El módulo **`bz2`** de **`BZ2File`** ahora soporta el protocolo de gestión de contexto, por lo que se puede escribir `con bz2.BZ2File(...) como f:`. (Contribución de Hagen Fürstenau; bpo-3860 [https://bugs.python.org/issue?@action=redirect&bpo=3860].)
- Nueva clase: la clase **`Counter`** del módulo **`collections`** es útil para el recuento de datos. Las instancias de **`Counter`** se comportan mayoritariamente como los diccionarios, pero retornan cero si faltan claves en lugar de lanzar un **`KeyError`**:

```
>>> from collections import Counter
>>> c = Counter()
>>> for letter in 'here is a sample of english text
...     c[letter] += 1
...
>>> c
Counter({' ': 6, 'e': 5, 's': 3, 'a': 2, 'i': 2, 'h
'l': 2, 't': 2, 'g': 1, 'f': 1, 'm': 1, 'o': 1, 'n'
'p': 1, 'r': 1, 'x': 1})
>>> c['e']
5
>>> c['z']
0
```

Hay tres métodos adicionales de **`Counter`**. **`most_common()`** retorna los N elementos más comunes y sus recuentos. **`elements()`** retorna un iterador sobre los elementos contenidos, repitiendo cada elemento tantas veces como su recuento. **`subtract()`** toma un iterable y resta uno por cada elemento en lugar de sumarlo; si el argumento es un diccionario u otra **`Counter`**, los recuentos se restan.

```
>>> c.most_common(5)
```

```
[(' ', 6), ('e', 5), ('s', 3), ('a', 2), ('i', 2)]
>>> c.elements() ->
   'a', 'a', ' ', ' ', ' ', ' ', ' ', ' ',
   'e', 'e', 'e', 'e', 'e', 'g', 'f', 'i', 'i',
   'h', 'h', 'm', 'l', 'l', 'o', 'n', 'p', 's',
   's', 's', 'r', 't', 't', 'x'
>>> c['e']
5
>>> c.subtract('very heavy on the letter e')
>>> c['e']    # Count is now lower
-1
```

Contribución de Raymond Hettinger; bpo-1696199 [https://bugs.python.org/issue?@action=redirect&bpo=1696199].

Nueva clase: `OrderedDict` se describe en la sección anterior PEP 372: Adición de un diccionario ordenado a las colecciones.

Nuevo método: El tipo de datos `deque` tiene ahora un método `count()` que retorna el número de elementos contenidos igual al argumento suministrado *x*, y un método `reverse()` que invierte los elementos del deque en su lugar. **`deque`** también expone su longitud máxima como el atributo de sólo lectura `maxlen`. (Ambas características han sido añadidas por Raymond Hettinger)

La clase `namedtuple` tiene ahora un parámetro opcional *rename*. Si *rename* es verdadero, los nombres de campo que no son válidos porque se han repetido o no son identificadores legales de Python serán renombrados a nombres legales que se derivan de la posición del campo dentro de la lista de campos:

```
>>> from collections import namedtuple
>>> T = namedtuple('T', ['field1', '$illegal', 'for
>>> T._fields
('field1', '_1', '_2', 'field2')
```

(Añadido por Raymond Hettinger; bpo-1818 [https://

bugs.python.org/issue?@action=redirect&bpo=1818].)

Por último, la clase base abstracta `Mapping` ahora devuelve `NotImplemented` si se compara una asignación con otro tipo que no es `Mapping`. (Corregido por Daniel Stutzbach; bpo-8729 [https://bugs.python.org/issue?@action=redirect&bpo=8729]).

- Los constructores de las clases de análisis del módulo `ConfigParser` ahora aceptan un parámetro *allow_no_value*, cuyo valor predeterminado es falso; si es verdadero, se permitirán opciones sin valores. Por ejemplo:

  ```
  >>> import ConfigParser, StringIO
  >>> sample_config = """
  ... [mysqld]
  ... user = mysql
  ... pid-file = /var/run/mysqld/mysqld.pid
  ... skip-bdb
  ... """
  >>> config = ConfigParser.RawConfigParser(allow_no_
  >>> config.readfp(StringIO.StringIO(sample_config))
  >>> config.get('mysqld', 'user')
  'mysql'
  >>> print config.get('mysqld', 'skip-bdb')
  None
  >>> print config.get('mysqld', 'unknown')
  Traceback (most recent call last):
    ...
  NoOptionError: No option 'unknown' in section: 'mys
  ```

  (Contribución de Mats Kindahl; bpo-7005 [https://bugs.python.org/issue?@action=redirect&bpo=7005].)

- Función obsoleta: **`contextlib.nested()`**, que permite manejar más de un administrador de contexto con una sola declaración `with`, ha quedado obsoleta, porque la declaración **`with`** ahora admite múltiples administradores de contexto.

- El módulo **`cookielib`** ahora ignora las cookies que tienen un campo de versión no válido, es decir, que no contiene un valor entero. (Corregido por John J. Lee; bpo-3924 [https://bugs.python.org/issue?@action=redirect&bpo=3924]).

- La función **`deepcopy()`** del módulo **`copy`** ahora copiará correctamente los métodos de instancia vinculados. (Implementado por Robert Collins; bpo-1515 [https://bugs.python.org/issue?@action=redirect&bpo=1515].)

- El módulo **`ctypes`** ahora convierte siempre `None` a un puntero C `NULL` para los argumentos declarados como punteros. (Cambiado por Thomas Heller; bpo-4606 [https://bugs.python.org/issue?@action=redirect&bpo=4606].) La biblioteca subyacente libffi [https://sourceware.org/libffi/] ha sido actualizada a la versión 3.0.9, que contiene varias correcciones para diferentes plataformas. (Actualizado por Matthias Klose; bpo-8142 [https://bugs.python.org/issue?@action=redirect&bpo=8142].)

- Nuevo método: la clase **`timedelta`** del módulo **`datetime`** ha ganado un método **`total_seconds()`** que retorna el número de segundos de la duración. (Contribución de Brian Quinlan; bpo-5788 [https://bugs.python.org/issue?@action=redirect&bpo=5788].)

- Nuevo método: la clase **`Decimal`** obtuvo un método de clase **`from_float()`** que realiza una conversión exacta de un número de punto flotante a un **`Decimal`**. Esta conversión exacta busca la aproximación decimal más cercana al valor de la representación de punto flotante; por lo tanto, el valor decimal resultante seguirá incluyendo la inexactitud, si la hubiera. Por ejemplo, `Decimal.from_float(0.1)` devuelve `Decimal('0.1000000000000000055511151231257827021181` (Implementado por Raymond Hettinger; bpo-4796 [https://bugs.python.org/issue?@action=redirect&bpo=4796]).

  La comparación de instancias de **`Decimal`** con números de punto flotante ahora produce resultados razonables basados en los valores numéricos de los operandos. Anteriormente, dichas comparaciones recurrían a las reglas predeterminadas de Python para comparar objetos, que producían resultados arbitrarios basados en su tipo. Tenga en cuenta que aún no puede combinar `Decimal` y punto flotante en otras operaciones como la suma, ya que debe elegir explícitamente cómo convertir entre float y `Decimal`. (Corregido por Mark Dickinson; bpo-2531 [https://bugs.python.org/issue?@action=redirect&bpo=2531]).

El constructor de `Decimal` acepta ahora números en coma flotante (añadido por Raymond Hettinger; bpo-8257 [https://bugs.python.org/issue?@action=redirect&bpo=8257]) y caracteres Unicode no europeos, como los dígitos arábigos-índicos (contribución de Mark Dickinson; bpo-6595 [https://bugs.python.org/issue?@action=redirect&bpo=6595]).

La mayoría de los métodos de la clase `Context` ahora aceptan enteros así como instancias de `Decimal`; las únicas excepciones son los métodos `canonical()` y `is_canonical()`. (Parche de Juan José Conti; bpo-7633 [https://bugs.python.org/issue?@action=redirect&bpo=7633].)

Cuando se utilizan instancias `Decimal` con el método `format()` de una cadena, la alineación por defecto era antes la izquierda. Esto se ha cambiado a la alineación derecha, que es más sensible para los tipos numéricos. (Cambiado por Mark Dickinson; bpo-6857 [https://bugs.python.org/issue?@action=redirect&bpo=6857].)

Las comparaciones que implican un valor NaN de señalización (o `sNAN`) señalan ahora `InvalidOperation` en lugar de retornar silenciosamente un valor verdadero o falso dependiendo del operador de comparación. Los valores NaN silenciosos (o `NaN`) son ahora hashable. (Corregido por Mark Dickinson; bpo-7279 [https://bugs.python.org/issue?@action=redirect&bpo=7279].)

- El módulo `difflib` produce ahora una salida más compatible con las herramientas modernas **diff/patch** mediante un pequeño cambio, utilizando un carácter de tabulación en lugar de espacios como separador en la cabecera que da el nombre del fichero. (Corregido por Anatoly Techtonik; bpo-7585 [https://bugs.python.org/issue?@action=redirect&bpo=7585].)

- El comando Distutils `sdist` ahora siempre regenera el archivo `MANIFEST`, ya que aunque los archivos `MANIFEST.in` o `setup.py` no hayan sido modificados, el usuario podría haber creado algunos archivos nuevos que deberían ser incluidos. (Corregido por Tarek Ziadé; bpo-8688 [https://bugs.python.org/issue?@action=redirect&bpo=8688].)

- El indicador **`IGNORE_EXCEPTION_DETAIL`** del módulo **`doctest`** ahora ignorará el nombre del módulo que contiene la excepción que se está probando. (Parche de Lennart Regebro; bpo-7490 [https://bugs.python.org/issue?@action=redirect&bpo=7490]).

- La clase **`Message`** del módulo **`email`** ahora aceptará una carga útil con valor Unicode y convertirá automáticamente la carga útil a la codificación especificada por **`output_charset`**. (Agregado por R. David Murray; bpo-1368247 [https://bugs.python.org/issue?@action=redirect&bpo=1368247]).

- La clase **`Fraction`** acepta ahora una instancia de float o **`Decimal`**, o dos números racionales, como argumentos para su constructor. (Implementado por Mark Dickinson; los racionales se añadieron en bpo-5812 [https://bugs.python.org/issue?@action=redirect&bpo=5812], y float/decimal en bpo-8294 [https://bugs.python.org/issue?@action=redirect&bpo=8294])

  Las comparaciones de ordenación (<, <=, >, >=) entre fracciones y números complejos ahora lanzan un **`TypeError`**. Esto corrige un descuido, haciendo que la **`Fraction`** coincida con los otros tipos numéricos.

- Nueva clase: **`FTP_TLS`** en el módulo **`ftplib`** proporciona conexiones FTP seguras utilizando la encapsulación TLS de la autenticación, así como las posteriores transferencias de control y datos. (Contribuido por Giampaolo Rodola; bpo-2054 [https://bugs.python.org/issue?

@action=redirect&bpo=2054].)

El método `storbinary()` para subidas binarias puede ahora reiniciar las subidas gracias a un parámetro *rest* añadido (parche de Pablo Mouzo; bpo-6845 [https://bugs.python.org/issue?@action=redirect&bpo=6845])

- Nuevo decorador de clase: `total_ordering()` en el módulo `functools` toma una clase que define un método `__eq__()` y uno de `__lt__()`, `__le__()`, `__gt__()` o `__ge__()`, y genera los métodos de comparación faltantes. Dado que el método `__cmp__()` está quedando obsoleto en Python 3.x, este decorador facilita la definición de clases ordenadas. (Agregado por Raymond Hettinger; bpo-5479 [https://bugs.python.org/issue?@action=redirect&bpo=5479]).

  Nueva función: `cmp_to_key()` tomará una función de comparación de estilo antiguo que espera dos argumentos y retornará una nueva llamada que puede usarse como parámetro *clave* para funciones como `sorted()`, `min()` y `max()`, etc. El principal uso previsto es ayudar a hacer el código compatible con Python 3.x. (Añadido por Raymond Hettinger.)

- Nueva función: la `gc()` del módulo `is_tracked` retorna true si una instancia dada es rastreada por el recolector de basura, false en caso contrario. (Contribución de Antoine Pitrou; bpo-4688 [https://bugs.python.org/issue?@action=redirect&bpo=4688].)

- El módulo `gzip` `GzipFile` soporta ahora el protocolo de gestión de contexto, por lo que se puede escribir `con gzip.GzipFile(...) como f:` (contribución de Hagen Fürstenau; bpo-3860 [https://bugs.python.org/issue?@action=redirect&bpo=3860]), y ahora implementa el ABC de `io.BufferedIOBase`, por lo que se puede envolver con `io.BufferedReader` para un procesamiento más rápido (contribución de Nir Aides; bpo-7471 [https://bugs.python.org/issue?@action=redirect&bpo=7471]). Ahora también es posible anular la hora de modificación registrada en un archivo gzipped proporcionando una marca de tiempo opcional al constructor. (Contribución de Jacques Frechet; bpo-4272 [https://bugs.python.org/issue?@action=redirect&bpo=4272].)

Los archivos en formato gzip pueden ser rellenados con bytes cero al final; el módulo `gzip` ahora consumirá estos bytes al final. (Corregido por Tadek Pietraszek y Brian Curtin; bpo-2846 [https://bugs.python.org/issue?@action=redirect&bpo=2846])

- Nuevo atributo: el módulo `hashlib` ahora tiene un atributo **`algorithms`** que contiene una tupla que nombra los algoritmos admitidos. En Python 2.7, `hashlib.algorithms` contiene `('md5', 'sha1', 'sha224', 'sha256', 'sha384', 'sha512')`. (Contribuido por Carl Chenet; bpo-7418 [https://bugs.python.org/issue?@action=redirect&bpo=7418]).

- La clase `HTTPResponse` predeterminada que utiliza el módulo `httplib` ahora admite el almacenamiento en búfer, lo que permite una lectura mucho más rápida de las respuestas HTTP. (Contribuido por Kristján Valur Jónsson; bpo-4879 [https://bugs.python.org/issue?@action=redirect&bpo=4879]).

  Las clases `HTTPConnection` y `HTTPSConnection` ahora admiten un parámetro *source_address*, una tupla de 2 `(host, port)` que proporciona la dirección de origen que se utilizará para la conexión. (Contribuido por Eldon Ziegler; bpo-3972 [https://bugs.python.org/issue?@action=redirect&bpo=3972]).

- El módulo **`ihooks`** ahora admite importaciones relativas. Tenga en cuenta que **`ihooks`** es un módulo más antiguo para personalizar importaciones, reemplazado por el módulo **`imputil`** agregado en Python 2.0. (Neil Schemenauer agregó compatibilidad con importaciones relativas).

- El módulo `imaplib` ahora soporta direcciones IPv6. (Contribución de Derek Morr; bpo-1655 [https://bugs.python.org/issue?@action=redirect&bpo=1655].)

- Nueva función: la `getcallargs()` del módulo `inspect` toma una llamada y sus argumentos posicionales y de palabra clave, y calcula qué parámetros de la llamada recibirán cada argumento, retornando un diccionario que asigna los nombres de los argumentos a sus valores. Por ejemplo:

```
>>> from inspect import getcallargs
>>> def f(a, b=1, *pos, **named):
...     pass
...
>>> getcallargs(f, 1, 2, 3)
{'a': 1, 'b': 2, 'pos': (3,), 'named': {}}
>>> getcallargs(f, a=2, x=4)
{'a': 2, 'b': 1, 'pos': (), 'named': {'x': 4}}
>>> getcallargs(f)
Traceback (most recent call last):
...
TypeError: f() takes at least 1 argument (0 given)
```

Contribución de George Sakkis; bpo-3135 [https://bugs.python.org/issue?@action=redirect&bpo=3135].

- Módulo actualizado: La biblioteca **`io`** se ha actualizado a la versión que viene con Python 3.1. Para la versión 3.1, la biblioteca de E/S se reescribió completamente en C y es de 2 a 20 veces más rápida según la tarea que se esté realizando. La versión original de Python se renombró como módulo **`_pyio`**.

  Un cambio resultante menor: la clase **`io.TextIOBase`** ahora tiene un atributo **`errors`** que proporciona la configuración de error utilizada para codificar y decodificar errores (uno de `'strict'`, `'replace'`, `'ignore'`).

  La clase **`io.FileIO`** ahora lanza un **`OSError`** cuando se le pasa un descriptor de fichero no válido. (Implementado por Benjamin Peterson; bpo-4991 [https://bugs.python.org/issue?@action=redirect&bpo=4991].) El método **`truncate()`** ahora preserva la posición del archivo; antes cambiaba la posición del archivo al final del nuevo archivo. (Corregido por Pascal Chambon; bpo-6939 [https://bugs.python.org/issue?@action=redirect&bpo=6939].)

- Nueva función: `itertools.compress(data, selectors)` toma dos iteradores. Los elementos de *datos* se retornan si el valor correspondiente en *selectores* es verdadero:

  ```
  itertools.compress('ABCDEF', [1,0,1,0,1,1]) =>
  A, C, E, F
  ```

  Nueva función: `itertools.combinations_with_replacement(iter, r)` retorna todas las posibles combinaciones de elementos de longitud *r* del iterable *iter*. A diferencia de **`combinations()`**, los elementos individuales pueden repetirse en las combinaciones generadas:

  ```
  itertools.combinations_with_replacement('abc', 2) =
    ('a', 'a'), ('a', 'b'), ('a', 'c'),
    ('b', 'b'), ('b', 'c'), ('c', 'c')
  ```

  Tenga en cuenta que los elementos se tratan como únicos en función de su posición en la entrada, no de sus valores reales.

  La función **`itertools.count()`** tiene ahora un argumento *step* que permite incrementar por valores distintos de 1. **`count()`** también permite ahora argumentos de palabra clave, y utilizar valores no enteros como números de punto flotante o instancias de **`Decimal`**. (Implementado por Raymond Hettinger; bpo-5032 [https://bugs.python.org/issue?@action=redirect&bpo=5032].)

  `itertools.combinaciones()` y **`itertools.producto()`** anteriormente lanzaban **`ValueError`** para valores de *r* mayores que el iterable de entrada. Esto se consideraba un error de especificación, por lo que ahora retornan un iterador vacío. (Corregido por Raymond Hettinger; bpo-4816 [https://bugs.python.org/issue?@action=redirect&bpo=4816].)

- Módulo actualizado: El módulo **`json`** ha sido actualizado a la versión 2.0.9 del paquete simplejson, que incluye una extensión en C que hace más rápida la codificación y decodificación. (Contribuido por Bob Ippolito; bpo-4136

[https://bugs.python.org/issue?@action=redirect&bpo=4136].)

Para soportar el nuevo tipo `collections.OrderedDict`, `json.load()` tiene ahora un parámetro opcional *object_pairs_hook* que será llamado con cualquier objeto literal que decodifique a una lista de pares. (Contribución de Raymond Hettinger; bpo-5381 [https://bugs.python.org/issue?@action=redirect&bpo=5381].)

- La clase `mailbox` del módulo `Maildir` ahora registra la marca de tiempo de los directorios que lee, y sólo los relee si la hora de modificación ha cambiado posteriormente. Esto mejora el rendimiento al evitar escaneos innecesarios de directorios. (Corregido por A.M. Kuchling y Antoine Pitrou; bpo-1607951 [https://bugs.python.org/issue?@action=redirect&bpo=1607951], bpo-6896 [https://bugs.python.org/issue?@action=redirect&bpo=6896])
- Nuevas funciones: el módulo `math` ganó `erf()` y `erfc()` para la función de error y la función de error complementaria, `expm1()` que calcula `e**x - 1` con más precisión que usando `exp()` y restando 1, `gamma()` para la función Gamma, y `lgamma()` para el logaritmo natural de la función Gamma. (Contribución de Mark Dickinson y nirinA raseliarison; bpo-3366 [https://bugs.python.org/issue?@action=redirect&bpo=3366].)
- A las clases `Manager*` del módulo `multiprocessing` ahora se les puede pasar un elemento invocable que se llamará cada vez que se inicie un subproceso, junto con un conjunto de argumentos que se le pasarán al elemento invocable. (Contribuido por lekma; bpo-5585 [https://bugs.python.org/issue?@action=redirect&bpo=5585]).

  La clase `Pool`, que controla un grupo de procesos de trabajo, ahora tiene un parámetro *maxtasksperchild* opcional. Los procesos de trabajo realizarán la cantidad especificada de tareas y luego saldrán, lo que hará que `Pool` inicie un nuevo proceso de trabajo. Esto es útil si las tareas pueden perder memoria u otros recursos, o si algunas tareas harán que el proceso de trabajo se vuelva muy grande. (Contribuido por

Charles Cazabon; bpo-6963 [https://bugs.python.org/issue?@action=redirect&bpo=6963]).

- El módulo **`nntplib`** ahora admite direcciones IPv6. (Contribución de Derek Morr; bpo-1664 [https://bugs.python.org/issue?@action=redirect&bpo=1664]).

- Nuevas funciones: el módulo **`os`** envuelve las siguientes llamadas al sistema POSIX: **`getresgid()`** y **`getresuid()`**, que retornan los GIDs y UIDs reales, efectivos y guardados; **`setresgid()`** y **`setresuid()`**, que establecen los GIDs y UIDs reales, efectivos y guardados a nuevos valores; **`initgroups()`**, que inicializan la lista de acceso a grupos para el proceso actual. (Funciones GID/UID contribuidas por Travis H.; bpo-6508 [https://bugs.python.org/issue?@action=redirect&bpo=6508]. Soporte para initgroups añadido por Jean-Paul Calderone; bpo-7333 [https://bugs.python.org/issue?@action=redirect&bpo=7333])

  La función **`os.fork()`** ahora reinicializa el bloqueo de importación en el proceso hijo; esto soluciona los problemas en Solaris cuando **`fork()`** se llama desde un hilo. (Corregido por Zsolt Cserna; bpo-7242 [https://bugs.python.org/issue?@action=redirect&bpo=7242].)

- En el módulo **`os.path`**, las funciones **`normpath()`** y **`abspath()`** ahora preservan Unicode; si su ruta de entrada es una cadena Unicode, el valor de retorno es también una cadena Unicode. (**`normpath()`** arreglado por Matt Giuca en bpo-5827 [https://bugs.python.org/issue?@action=redirect&bpo=5827]; **`abspath()`** arreglado por Ezio Melotti en bpo-3426 [https://bugs.python.org/issue?@action=redirect&bpo=3426])

- El módulo **`pydoc`** tiene ahora ayuda para los distintos símbolos que utiliza Python. Ahora puedes hacer `help('<<')` o `help('@')`, por ejemplo. (Contribución de David Laban; bpo-4739 [https://bugs.python.org/issue?@action=redirect&bpo=4739].)

- Las funciones **`re`** del módulo **`split()`**, **`sub()`** y **`subn()`** aceptan ahora un argumento opcional *flags*, por coherencia con las demás funciones del módulo. (Añadido por Gregory P. Smith)

- Nueva función: `run_path()` en el módulo `runpy` ejecutará el código en un argumento *ruta* proporcionado. *path* puede ser la ruta de un fichero fuente de Python (`ejemplo.py`), un fichero bytecode compilado (`ejemplo.pyc`), un directorio (`./package/`), o un archivo zip (`ejemplo.zip`). Si se proporciona un directorio o una ruta zip, se añadirá al frente de `sys.path` y se importará el módulo `__main__`. Se espera que el directorio o zip contenga un `__main__.py`; si no es así, algún otro `__main__.py` podría ser importado desde una ubicación posterior en `sys.path`. Esto hace que más de la maquinaria de **`runpy`** esté disponible para los scripts que quieran imitar la forma en que la línea de comandos de Python procesa un nombre de ruta explícito. (Añadido por Nick Coghlan; bpo-6816 [https://bugs.python.org/issue?@action=redirect&bpo=6816].)

- Nueva función: en el módulo `shutil`, `make_archive()` toma un nombre de fichero, un tipo de archivo (formato zip o tar) y una ruta de directorio, y crea un archivo con el contenido del directorio. (Añadido por Tarek Ziadé.)

  Las funciones `shutil` `copyfile()` y `copytree()` ahora lanzan una excepción `SpecialFileError` cuando se les pide que copien una tubería con nombre. Anteriormente, el código trataba las tuberías con nombre como un archivo normal, abriéndolas para su lectura, lo que bloqueaba indefinidamente. (Corregido por Antoine Pitrou; bpo-3002 [https://bugs.python.org/issue?@action=redirect&bpo=3002].)

- El módulo `signal` ya no reinstala el manejador de señales a menos que sea realmente necesario, lo que corrige un error que podía hacer imposible atrapar la señal EINTR de forma robusta. (Corregido por Charles-Francois Natali; bpo-8354 [https://bugs.python.org/issue?@action=redirect&bpo=8354].)

- Nuevas funciones: en el módulo `site`, tres nuevas funciones devuelven varias rutas específicas del sitio y del usuario. `getsitepackages()` devuelve una lista que contiene todos los directorios de paquetes del sitio globales, `getusersitepackages()` devuelve la ruta del directorio de paquetes del sitio del usuario y `getuserbase()` devuelve el valor de la variable de entorno `USER_BASE`, que proporciona la ruta a un directorio que se puede utilizar para almacenar datos. (Contribuido por Tarek Ziadé; bpo-6693 [https://bugs.python.org/issue?@action=redirect&bpo=6693].)

El módulo `site` ahora informa de las excepciones que se producen cuando se importa el módulo `sitecustomize`, y ya no atrapará y tragará la excepción `KeyboardInterrupt`. (Corregido por Victor Stinner; bpo-3137 [https://bugs.python.org/issue?@action=redirect&bpo=3137].)

- La función `create_connection()` ganó un parámetro *dirección_de_origen*, una 2-tupla `(host, port)` que da la dirección de origen que se utilizará para la conexión. (Contribuido por Eldon Ziegler; bpo-3972 [https://bugs.python.org/issue?@action=redirect&bpo=3972].)

  Los métodos `recv_into()` y `recvfrom_into()` ahora escribirán en objetos que soporten la API de búfer, más útilmente los objetos `bytearray` y `memoryview`. (Implementado por Antoine Pitrou; bpo-8104 [https://bugs.python.org/issue?@action=redirect&bpo=8104].)

- La clase `TCPServer` del módulo `SocketServer` ahora admite tiempos de espera de socket y la desactivación del algoritmo Nagle. El atributo de clase **`disable_nagle_algorithm`** tiene como valor predeterminado `False`; si se sobrescribe como verdadero, las nuevas conexiones de solicitud tendrán la opción TCP_NODELAY configurada para evitar el almacenamiento en búfer de muchos envíos pequeños en un solo paquete TCP. El atributo de clase `timeout` puede contener un tiempo de espera en segundos que se aplicará al socket de solicitud; si no se recibe ninguna solicitud dentro de ese tiempo, se llamará a `handle_timeout()` y `handle_request()` regresará. (Contribuido por Kristján Valur Jónsson; bpo-6192 [https://bugs.python.org/issue?@action=redirect&bpo=6192] y bpo-6267 [https://bugs.python.org/issue?

@action=redirect&bpo=6267]).

- Módulo actualizado: el módulo `sqlite3` ha sido actualizado a la versión 2.6.0 del paquete pysqlite [https://github.com/ghaering/pysqlite]. La versión 2.6.0 incluye una serie de correcciones de errores y añade la posibilidad de cargar extensiones de SQLite desde bibliotecas compartidas. Llame al método `enable_load_extension(True)` para habilitar las extensiones, y luego llame a `load_extension()` para cargar una biblioteca compartida en particular. (Actualizado por Gerhard Häring.)

- The `ssl` module's `SSLSocket` objects now support the buffer API, which fixed a test suite failure (fix by Antoine Pitrou; bpo-7133 [https://bugs.python.org/issue?@action=redirect&bpo=7133]) and automatically set OpenSSL's **`SSL_MODE_AUTO_RETRY`**, which will prevent an error code being returned from **`recv()`** operations that trigger an SSL renegotiation (fix by Antoine Pitrou; bpo-8222 [https://bugs.python.org/issue?@action=redirect&bpo=8222]).

  La función constructora `wrap_socket()` ahora acepta un argumento *ciphers* que es una cadena que enumera los algoritmos de cifrado que se permitirán; el formato de la cadena se describe en in the OpenSSL documentation [https://docs.openssl.org/1.0.2/man1/ciphers/]. (Agregado por Antoine Pitrou; bpo-8322 [https://bugs.python.org/issue?@action=redirect&bpo=8322]).

  Otro cambio hace que la extensión cargue todos los cifrados y algoritmos de compendio de OpenSSL para que estén todos disponibles. Algunos certificados SSL no podían ser verificados, reportando un error de «algoritmo desconocido». (Informado por Beda Kosata, y corregido por Antoine Pitrou; bpo-8484 [https://bugs.python.org/issue?@action=redirect&bpo=8484])

  La versión de OpenSSL que se está utilizando ahora está disponible como atributos de módulo `ssl.OPENSSL_VERSION` (una cadena), `ssl.OPENSSL_VERSION_INFO` (una tupla de 5) y `ssl.OPENSSL_VERSION_NUMBER` (un entero). (Agregado por Antoine Pitrou; bpo-8321 [https://bugs.python.org/issue?@action=redirect&bpo=8321]).

- El módulo **`struct`** ya no ignorará silenciosamente los errores de desbordamiento cuando un valor sea demasiado grande para un código de formato de entero en particular (uno de `bBhHiIlLqQ`); ahora siempre genera una excepción **`struct.error`**. (Modificado por Mark Dickinson; bpo-1523 [https://bugs.python.org/issue?@action=redirect&bpo=1523]). La función **`pack()`** también intentará usar **`__index__()`** para convertir y empaquetar números que no sean enteros antes de intentar el método **`__int__()`** o informar un error. (Modificado por Mark Dickinson; bpo-8300 [https://bugs.python.org/issue?@action=redirect&bpo=8300]).

- Nueva función: la función **`check_output()`** del módulo **`subprocess`** ejecuta un comando con un conjunto de argumentos especificado y retorna la salida del comando como una cadena de texto cuando el comando se ejecuta sin errores, o lanza una excepción **`CalledProcessError`** en caso contrario.

  ```
  >>> subprocess.check_output(['df', '-h', '.'])
  'Filesystem     Size   Used  Avail Capacity  Mounte
  /dev/disk0s2    52G    49G   3.0G    94%    /\n'

  >>> subprocess.check_output(['df', '-h', '/bogus'])
    ...
  subprocess.CalledProcessError: Command '['df', '-h'
  ```

  (Contribución de Gregory P. Smith.)

  El módulo **`subprocess`** ahora volverá a intentar realizar llamadas internas al sistema al recibir una señal **`EINTR`**. (Varias personas lo informaron; parche final de Gregory P. Smith en bpo-1068268 [https://bugs.python.org/issue?@action=redirect&bpo=1068268]).

- Nueva función: **`is_declared_global()`** en el módulo **`symtable`** retorna true para las variables declaradas explícitamente como globales, false para las que son implícitamente globales. (Contribución de Jeremy Hylton)

- El módulo `syslog` utilizará ahora el valor de `sys.argv[0]` como identificador en lugar del anterior valor por defecto de `'python`. (Cambiado por Sean Reifschneider; bpo-8451 [https://bugs.python.org/issue?@action=redirect&bpo=8451].)

- The `sys.version_info` value is now a named tuple, with attributes named **`major`**, **`minor`**, **`micro`**, **`releaselevel`**, and **`serial`**. (Contributed by Ross Light; bpo-4285 [https://bugs.python.org/issue?@action=redirect&bpo=4285].)

  `sys.getwindowsversion()` también devuelve una tupla con nombre, con atributos denominados **`major`**, **`minor`**, **`build`**, **`platform`**, **`service_pack`**, **`service_pack_major`**, **`service_pack_minor`**, **`suite_mask`** y **`product_type`**. (Contribuido por Brian Curtin; bpo-7766 [https://bugs.python.org/issue?@action=redirect&bpo=7766]).

- El manejo de errores por defecto del módulo `tarfile` ha cambiado, para no suprimir los errores fatales. El nivel de error por defecto era antes 0, lo que significaba que los errores sólo daban lugar a que se escribiera un mensaje en el registro de depuración, pero como el registro de depuración no está activado por defecto, estos errores pasan desapercibidos. El nivel de error por defecto es ahora 1, que lanza una excepción si hay un error. (Cambiado por Lars Gustäbel; bpo-7357 [https://bugs.python.org/issue?@action=redirect&bpo=7357].)

  `tarfile` ahora soporta el filtrado de los objetos `TarInfo` que se añaden a un fichero tar. Cuando llame a `add()`, puede suministrar un argumento opcional *filtro* que es una llamada. Al *filtro* se le pasará la clase **`TarInfo`** de cada fichero que se añada, y podrá modificarlo y retornarlo. Si la llamada retorna `Nada`, el fichero será excluido del archivo resultante. Esto es más potente que el argumento *exclude* existente, que por lo tanto ha sido obviado. (Añadido por Lars Gustäbel; bpo-6856 [https://bugs.python.org/issue?

@action=redirect&bpo=6856].) La clase `TarFile` también soporta ahora el protocolo de gestión de contexto. (Añadido por Lars Gustäbel; bpo-7232 [https://bugs.python.org/issue?@action=redirect&bpo=7232].)

- El método `wait()` de la clase `threading.Event` ahora retorna la bandera interna al salir. Esto significa que el método normalmente retornará true porque `wait()` se supone que bloquea hasta que la bandera interna se convierte en true. El valor de retorno sólo será falso si se proporcionó un tiempo de espera y la operación expiró. (Contribuido por Tim Lesher; bpo-1674032 [https://bugs.python.org/issue?@action=redirect&bpo=1674032].)

- La base de datos Unicode proporcionada por el módulo `unicodedata` se utiliza ahora internamente para determinar qué caracteres son numéricos, espacios en blanco o representan saltos de línea. La base de datos también incluye información del archivo de datos `Unihan.txt` (parche de Anders Chrigström y Amaury Forgeot d'Arc; bpo-1571184 [https://bugs.python.org/issue?@action=redirect&bpo=1571184]) y se ha actualizado a la versión 5.2.0 (actualizado por Florent Xicluna; bpo-8024 [https://bugs.python.org/issue?@action=redirect&bpo=8024]).

- El módulo `urlsplit()` del `urlparse` ahora maneja esquemas de URL desconocidos de una manera compatible con el RFC 3986 [https://datatracker.ietf.org/doc/html/rfc3986.html]: si la URL tiene el formato `"<something>://..."`, el texto antes del `://` se trata como el esquema, incluso si es un esquema inventado que el módulo no conoce. Este cambio puede romper el código que funcionaba con el comportamiento anterior. Por ejemplo, Python 2.6.4 o 2.5 devolverá lo siguiente:

  ```
  >>> import urlparse
  >>> urlparse.urlsplit('invented://host/nombre_archi
  ('invented', '', '//host/nombre_archivo?consulta',
  ```

  Python 2.7 (y Python 2.6.5) retornará:

```
>>> import urlparse
>>> urlparse.urlsplit('invented://host/filename?que
('invented', 'host', '/filename?query', '', '')
```

(Python 2.7 en realidad produce una salida ligeramente diferente, ya que retorna una tupla con nombre en lugar de una tupla estándar)

El módulo `urlparse` también admite direcciones literales IPv6 según lo definido por RFC 2732 [https://datatracker.ietf.org/doc/html/rfc2732.html] (contribuido por Senthil Kumaran; bpo-2987 [https://bugs.python.org/issue?@action=redirect&bpo=2987]).

```
>>> urlparse.urlparse('http://[1080::8:800:200C:417
ParseResult(scheme='http', netloc='[1080::8:800:200
path='/foo', params='', query='', fragment='')
```

- Nueva clase: la clase `WeakSet` del módulo `weakref` es un conjunto que sólo contiene referencias débiles a sus elementos; los elementos serán eliminados una vez que no haya referencias que apunten a ellos. (Implementado originalmente en Python 3.x por Raymond Hettinger, y Portado a 2.7 por Michael Foord)

- La biblioteca `xml.etree.ElementTree` ya no omite los símbolos & ni los corchetes angulares al generar una instrucción de procesamiento XML (que se parece a `<?xml-stylesheet href="#style1"?>`) o un comentario (que se parece a `<!-- comment -->`). (Parche de Neil Muller; bpo-2746 [https://bugs.python.org/issue?@action=redirect&bpo=2746]).

- El cliente y el servidor XML-RPC, proporcionados por los módulos `xmlrpclib` y `SimpleXMLRPCServer`, han mejorado su rendimiento al admitir la función keep-alive de HTTP/1.1 y al utilizar opcionalmente la codificación gzip para comprimir el XML que se intercambia. La compresión gzip está controlada por el atributo **`encode_threshold`** de `SimpleXMLRPCRequestHandler`, que contiene un tamaño en bytes; las respuestas más grandes que este tamaño se comprimirán. (Contribuido por Kristján Valur Jónsson; bpo-6267 [https://bugs.python.org/issue?@action=redirect&bpo=6267].)

- El módulo `zipfile` de `ZipFile` soporta ahora el protocolo de gestión de contexto, por lo que se puede escribir `con zipfile.ZipFile(...) como f:`. (Contribución de Brian Curtin; bpo-5511 [https://bugs.python.org/issue?@action=redirect&bpo=5511].)

  `zipfile` ahora también permite archivar directorios vacíos y extraerlos correctamente. (Corregido por Kuba Wieczorek; bpo-4710 [https://bugs.python.org/issue?@action=redirect&bpo=4710].) La lectura de archivos de un archivo es más rápida y la intercalación de `read()` y `readline()` ahora funciona correctamente. (Contribuido por Nir Aides; bpo-7610 [https://bugs.python.org/issue?@action=redirect&bpo=7610].)

  La función `is_zipfile()` acepta ahora un objeto archivo, además de los nombres de ruta aceptados en versiones anteriores. (Contribución de Gabriel Genellina; bpo-4756 [https://bugs.python.org/issue?@action=redirect&bpo=4756].)

  El método `writestr()` tiene ahora un parámetro opcional *compress_type* que permite anular el método de compresión por defecto especificado en el constructor `ZipFile`. (Contribución de Ronald Oussoren; bpo-6003 [https://bugs.python.org/issue?@action=redirect&bpo=6003].)

Nuevo módulo: importlib

Python 3.1 incluye el paquete `importlib`, una reimplementación de la lógica subyacente a la declaración `import` de Python. `importlib` es útil para los implementadores de intérpretes de Python y para los usuarios que desean escribir nuevos importadores que puedan participar en el proceso de importación. Python 2.7 no contiene el paquete `importlib` completo, sino que tiene un pequeño subconjunto que contiene una única función, `import_module()`.

`import_module(name, package=None)` importa un módulo. *name* es una cadena que contiene el nombre del módulo o paquete. Es posible realizar importaciones relativas proporcionando una cadena que comience con un carácter `.`, como `..utils.errors`. Para las importaciones relativas, el argumento *paquete* debe ser proporcionado y es el nombre del paquete que se utilizará como ancla para la importación relativa. **`import_module()`** inserta el módulo importado en `sys.modules` y retorna el objeto módulo.

He aquí algunos ejemplos:

```
>>> from importlib import import_module
>>> anydbm = import_module('anydbm')  # Standard absolut
>>> anydbm
<module 'anydbm' from '/p/python/Lib/anydbm.py'>
>>> # Relative import
>>> file_util = import_module('..file_util', 'distutils.
>>> file_util
<module 'distutils.file_util' from '/python/Lib/distutil
```

`importlib` fue implementado por Brett Cannon e introducido en Python 3.1.

Nuevo módulo: sysconfig

El módulo **`sysconfig`** ha sido retirado del paquete Distutils, convirtiéndose en un nuevo módulo de alto nivel por derecho propio. **`sysconfig`** proporciona funciones para obtener información sobre el proceso de construcción de Python: interruptores del compilador, rutas de instalación, el nombre de la plataforma y si Python se está ejecutando desde su directorio fuente.

Algunas de las funciones del módulo son:

- **`get_config_var()`** retorna variables del Makefile de Python y del fichero `pyconfig.h`.
- **`get_config_vars()`** retorna un diccionario que contiene todas las variables de configuración.
- **`get_path()`** retorna la ruta configurada para un tipo de

  módulo concreto: la biblioteca estándar, módulos específicos del sitio, módulos específicos de la plataforma, etc.
- `is_python_build()` retorna true si está ejecutando un binario desde un árbol de fuentes de Python, y false en caso contrario.

Consulte la documentación de `sysconfig` para más detalles y para una lista completa de funciones.

El paquete Distutils y `sysconfig` son ahora mantenidos por Tarek Ziadé, que también ha iniciado un paquete Distutils2 (repositorio de fuentes en https://hg.python.org/distutils2/) para desarrollar una versión de próxima generación de Distutils.

ttk: Widgets temáticos para Tk

Tcl/Tk 8.5 incluye un conjunto de widgets temáticos que reimplementan los widgets básicos de Tk pero tienen una apariencia más personalizable y, por tanto, pueden parecerse más a los widgets de la plataforma nativa. Este conjunto de widgets se llamaba originalmente Tile, pero fue renombrado a Ttk (por «Tk temático») al ser añadido a la versión 8.5 de Tcl/Tck.

Para obtener más información, lea la documentación del módulo `ttk`. También puede leer la página del manual de Tcl/Tk que describe el motor de temas Ttk, disponible en https://www.tcl.tk/man/tcl8.5/TkCmd/ttk_intro.html. Algunas capturas de pantalla del código Python/Ttk en uso se encuentran en https://code.google.com/archive/p/python-ttk/wikis/Screenshots.wiki.

El módulo `tkinter.ttk` fue escrito por Guilherme Polo y se agregó en bpo-2983 [https://bugs.python.org/issue?@action=redirect&bpo=2983]. Se propuso una versión alternativa llamada `Tile.py`, escrita por Martin Franklin y mantenida por Kevin Walzer, para su inclusión en bpo-2618 [https://bugs.python.org/issue?@action=redirect&bpo=2618], pero los autores argumentaron que el trabajo de Guilherme Polo era más completo.

Módulo actualizado: unittest

El módulo `unittest` se ha mejorado considerablemente y se han añadido muchas funciones nuevas. La mayoría de estas funciones fueron implementadas por Michael Foord, a menos que se indique lo contrario. La versión mejorada del módulo se puede descargar por separado para su uso con las versiones 2.4 a 2.6 de Python, empaquetada como el paquete **`unittest2`**, desde unittest2 [https://pypi.org/project/unittest2/].

Cuando se usa desde la línea de comandos, el módulo puede descubrir pruebas automáticamente. No es tan elegante como py.test [https://pytest.org] o nose [https://nose.readthedocs.io/], pero proporciona una forma sencilla de ejecutar pruebas guardadas dentro de un conjunto de directorios de paquetes. Por ejemplo, el siguiente comando buscará en el subdirectorio `test/` cualquier archivo de prueba importable llamado `test*.py`:

```
python -m unittest descubre -s prueba
```

Consulte la documentación del módulo `unittest` para más detalles. (Desarrollado en bpo-6001 [https://bugs.python.org/issue?@action=redirect&bpo=6001].)

La función `main()` soporta algunas otras opciones nuevas:

- `-b` o **`--buffer`** almacenará en búfer la salida estándar y los flujos de error estándar durante cada prueba. Si la prueba pasa, se descartará cualquier salida resultante; en caso de falla, se mostrará la salida almacenada en búfer.
- `-c` o **`--catch`** hará que la interrupción de control-C se maneje con más gracia. En lugar de interrumpir el proceso de prueba inmediatamente, se completará la prueba que se esté ejecutando en ese momento y luego se informará de los resultados parciales hasta la interrupción. Si está impaciente, una segunda pulsación de control-C provocará una interrupción inmediata.</unittest>

  Este manejador de control-C intenta evitar que se produzcan problemas cuando el código que se está probando o las pruebas que se están ejecutando han definido un manejador de señales propio, advirtiendo que ya se ha establecido un manejador de señales y llamándolo. Si esto no te funciona, hay un decorador `removeHandler()` que puede utilizarse para marcar las pruebas que deben tener el manejador de control-C desactivado.

- `-f` o `--failfast` hace que la ejecución de la prueba se detenga inmediatamente cuando una prueba falla en lugar de continuar ejecutando más pruebas. (Sugerido por Cliff Dyer e implementado por Michael Foord; bpo-8074 [https://bugs.python.org/issue?@action=redirect&bpo=8074]) </unittest>

Los mensajes de progreso ahora muestran "x" para los fallos esperados y "u" para los éxitos inesperados cuando se ejecuta en modo verboso. (Contribución de Benjamin Peterson)

Los casos de prueba pueden lanzar la excepción `SkipTest` para saltarse una prueba (bpo-1034053 [https://bugs.python.org/issue?@action=redirect&bpo=1034053]).

Los mensajes de error para los fallos `assertEqual()`, `assertTrue()`, y `assertFalse()` ahora proporcionan más información. Si estableces el atributo `longMessage` de tus clases `TestCase` a true, tanto el mensaje de error estándar como cualquier mensaje adicional que proporciones se imprimirán para los fallos. (Añadido por Michael Foord; bpo-5663 [https://bugs.python.org/issue?@action=redirect&bpo=5663].)

El método `assertRaises()` ahora retorna un manejador de contexto cuando se llama sin proporcionar un objeto invocable para ejecutar. Por ejemplo, puede escribir esto:

```
with self.assertRaises(KeyError):
    {}['foo']
```

(Implementado por Antoine Pitrou; bpo-4444 [https://bugs.python.org/issue?@action=redirect&bpo=4444].)

Ahora se soportan las funciones de configuración y desmontaje a nivel de módulo y de clase. Los módulos pueden contener funciones `setUpModule()` y `tearDownModule()`. Las clases pueden tener métodos `setUpClass()` y `tearDownClass()` que deben ser definidos como métodos de clase (usando `@classmethod` o equivalente). Estas funciones y métodos se invocan cuando el ejecutor de pruebas cambia a un caso de prueba en un módulo o clase diferente.

Se han añadido los métodos `addCleanup()` y `doCleanups()`. **`addCleanup()`** permite añadir funciones de limpieza que serán llamadas incondicionalmente (después de `setUp()` si **`setUp()`** falla, en caso contrario después de `tearDown()`). Esto permite una asignación y desasignación de recursos mucho más sencilla durante las pruebas (bpo-5679 [https://bugs.python.org/issue?@action=redirect&bpo=5679]).

Se añadieron varios métodos nuevos que proporcionan pruebas más especializadas. Muchos de estos métodos fueron escritos por ingenieros de Google para utilizarlos en sus conjuntos de pruebas; Gregory P. Smith, Michael Foord y GvR trabajaron en la fusión de los mismos en la versión de Python de `unittest`.

- `assertIsNone()` y `assertIsNotNone()` toman una expresión y verifican que el resultado es o no es `None`.
- `assertIs()` y `assertIsNot()` toman dos valores y comprueban si los dos valores se evalúan al mismo objeto o no. (Añadido por Michael Foord; bpo-2578 [https://bugs.python.org/issue?@action=redirect&bpo=2578].)
- `assertIsInstance()` y `assertNotIsInstance()` comprueban si el objeto resultante es una instancia de una clase concreta, o de una tupla de clases. (Añadido por Georg Brandl; bpo-7031 [https://bugs.python.org/issue?@action=redirect&bpo=7031].)
- `assertGreater()`, `assertGreaterEqual()`, `assertLess()`, y `assertLessEqual()` comparan dos cantidades.
- `assertMultiLineEqual()` compara dos cadenas, y si no son iguales, muestra una comparación útil que resalta las diferencias en las dos cadenas. Esta comparación se utiliza ahora por defecto cuando las cadenas Unicode se comparan con `assertEqual()`.
- `assertRegexpMatches()` y `assertNotRegexpMatches()` verifican si el primer

argumento es una cadena que coincide o no con la expresión regular proporcionada como segundo argumento (bpo-8038 [https://bugs.python.org/issue?@action=redirect&bpo=8038]).

- `assertRaisesRegexp()` verifica si se genera una excepción particular y luego también verifica que la representación de cadena de la excepción coincida con la expresión regular proporcionada.
- `assertIn()` y `assertNotIn()` comprueban si *primero* está o no está en *segundo*.
- `assertItemsEqual()` prueba si dos secuencias proporcionadas contienen los mismos elementos.
- `assertSetEqual()` compara si dos conjuntos son iguales, y sólo informa de las diferencias entre los conjuntos en caso de error.
- De forma similar, `assertListEqual()` y `assertTupleEqual()` comparan los tipos especificados y explican cualquier diferencia sin imprimir necesariamente sus valores completos; estos métodos se utilizan ahora por defecto cuando se comparan listas y tuplas utilizando `assertEqual()`. De forma más general, `assertSequenceEqual()` compara dos secuencias y puede comprobar opcionalmente si ambas secuencias son de un tipo determinado.
- `assertDictEqual()` compara dos diccionarios e informa las diferencias; ahora se utiliza de forma predeterminada cuando se comparan dos diccionarios mediante `assertEqual()`. `assertDictContainsSubset()` comprueba si todos los pares clave/valor de *first* se encuentran en *second*.
- `assertAlmostEqual()` y `assertNotAlmostEqual()` prueban si *first* y *second* son aproximadamente iguales. Este método puede redondear su diferencia a un número especificado opcionalmente de *lugares* (el valor predeterminado es 7) y compararlo con cero, o requerir que la diferencia sea menor que un valor *delta* proporcionado.
- `loadTestsFromName()` respeta correctamente el atributo `suiteClass` de la `TestLoader`. (Corregido por Mark Roddy; bpo-6866 [https://bugs.python.org/issue?@action=redirect&bpo=6866].)
- Un nuevo hook permite extender el método

`assertEqual()` para manejar nuevos tipos de datos. El método `addTypeEqualityFunc()` toma un objeto de tipo y una función. La función se utilizará cuando los dos objetos que se comparan sean del tipo especificado. Esta función debe comparar los dos objetos y lanzar una excepción si no coinciden; es una buena idea que la función proporcione información adicional acerca de por qué los dos objetos no coinciden, al igual que los nuevos métodos de comparación de secuencias.

`unittest.main()` ahora toma un argumento opcional `exit`. Si es falso, `main()` no llama a `sys.exit()`, permitiendo que `main()` sea utilizado desde el intérprete interactivo. (Contribuido por J. Pablo Fernández; bpo-3379 [https://bugs.python.org/issue?@action=redirect&bpo=3379].)

`TestResult` tiene nuevos métodos `startTestRun()` y `stopTestRun()` que se llaman inmediatamente antes y después de la ejecución de una prueba. (Contribución de Robert Collins; bpo-5728 [https://bugs.python.org/issue?@action=redirect&bpo=5728].)

Con todos estos cambios, el archivo `unittest.py` se estaba volviendo incómodamente grande, por lo que el módulo se convirtió en un paquete y el código se dividió en varios archivos (por Benjamin Peterson). Esto no afecta a la forma de importar o utilizar el módulo.

Ver también

https://web.archive.org/web/20210619163128/http://www.voidspace.org.uk/python/articles/unittest2.shtml
Describe las nuevas funciones, cómo utilizarlas y la justificación de varias decisiones de diseño. (Por Michael Foord.)

Módulo actualizado: ElementTree 1.3

La versión de la biblioteca ElementTree incluida en Python se ha actualizado a la versión 1.3. Algunas de las nuevas características son:

- Las diversas funciones de análisis ahora toman un argumento de palabra clave *parser* que da una instancia de `XMLParser` que se utilizará. Esto hace posible anular la codificación interna del archivo:

  ```
  p = ET.XMLParser(codificación='utf-8')
  t = ET.XML("""<root/>""", analizador=p)
  ```

  Los errores al analizar XML ahora generan una excepción `ParseError`, cuyas instancias tienen un atributo **`position`** que contiene una tupla (*line*, *column*) que indica la ubicación del problema.

- ElementTree's code for converting trees to a string has been significantly reworked, making it roughly twice as fast in many cases. The `ElementTree.write()` and **`Element.write()`** methods now have a *method* parameter that can be «xml» (the default), «html», or «text». HTML mode will output empty elements as `<empty></empty>` instead of `<empty/>`, and text mode will skip over elements and only output the text chunks. If you set the `tag` attribute of an element to `None` but leave its children in place, the element will be omitted when the tree is written out, so you don't need to do more extensive rearrangement to remove a single element.

  Namespace handling has also been improved. All `xmlns:<whatever>` declarations are now output on the root element, not scattered throughout the resulting XML. You can set the default namespace for a tree by setting the **`default_namespace`** attribute and can register new prefixes with `register_namespace()`. In XML mode, you can use the true/false *xml_declaration* parameter to suppress the XML declaration.

- Nuevo método `Element`: `extend()` añade los elementos de una secuencia a los hijos del elemento. Los propios elementos se comportan como secuencias, por lo que es fácil mover los hijos de un elemento a otro:

  ```
  from xml.etree import ElementTree as ET
  ```

```
t = ET.XML("""<list>
  <item>1</item> <item>2</item>  <item>3</item>
</list>""")
new = ET.XML('<root/>')
new.extend(t)

# Outputs <root><item>1</item>...</root>
print ET.tostring(new)
```

- Nuevo método **`Element`**: **`iter()`** genera los elementos secundarios del elemento como generador. También es posible escribir `for child in elem:` para recorrer los elementos secundarios de un elemento. El método existente **`getiterator()`** ya no se utiliza, al igual que **`getchildren()`**, que construye y devuelve una lista de elementos secundarios.

- Nuevo método **`Element`**: **`itertext()`** genera todos los fragmentos de texto que son descendientes del elemento. Por ejemplo:

  ```
  t = ET.XML("""<list>
    <item>1</item> <item>2</item>  <item>3</item>
  </list>""")

  # Outputs ['\n  ', '1', ' ', '2', '  ', '3', '\n']
  print list(t.itertext())
  ```

- Deprecado: el uso de un elemento como booleano (es decir, `if elem:`) retornaba true si el elemento tenía algún hijo, o false si no había ningún hijo. Este comportamiento es confuso – `None` es falso, pero también lo es un elemento sin hijos? – por lo que ahora provocará un **`FutureWarning`**. En tu código, deberías ser explícito: escribe `len(elem) != 0` si te interesa el número de hijos, o `elem no es None`.

Fredrik Lundh desarrolla ElementTree y produjo la versión 1.3; puede leer su artículo que describe 1.3 en https://web.archive.org/web/20200703234532/http://effbot.org/zone/elementtree-13-

intro.htm. Florent Xicluna actualizó la versión incluida con Python, luego de discusiones en python-dev y en bpo-6472 [https://bugs.python.org/issue?@action=redirect&bpo=6472].)

Cambios en la API de construcción y C

Los cambios en el proceso de construcción de Python y en la API de C incluyen:

- La última versión del depurador GNU, GDB 7, puede ser scripted using Python [https://web.archive.org/web/20110715084810/http://sourceware.org/gdb/current/onlinedocs/gdb/Python.html]. Cuando comience a depurar un programa ejecutable P, GDB buscará un archivo llamado `P-gdb.py` y lo leerá automáticamente. Dave Malcolm contribuyó con un `python-gdb.py` que agrega una serie de comandos útiles para depurar Python en sí. Por ejemplo, `py-up` y `py-down` suben o bajan un marco de pila de Python, que generalmente corresponde a varios marcos de pila de C. `py-print` imprime el valor de una variable de Python y `py-bt` imprime el seguimiento de la pila de Python. (Agregado como resultado de bpo-8032 [https://bugs.python.org/issue?@action=redirect&bpo=8032]).

- Si utiliza el archivo `.gdbinit` proporcionado con Python, la macro «pyo» en la versión 2.7 ahora funciona correctamente cuando el hilo que se está depurando no tiene el GIL; la macro ahora lo adquiere antes de imprimir. (Contribución de Victor Stinner; bpo-3632 [https://bugs.python.org/issue?@action=redirect&bpo=3632].)

- **`Py_AddPendingCall()`** ahora es seguro para los hilos, permitiendo que cualquier hilo trabajador envíe notificaciones al hilo principal de Python. Esto es particularmente útil para las operaciones asíncronas de IO. (Contribuido por Kristján Valur Jónsson; bpo-4293 [https://bugs.python.org/issue?@action=redirect&bpo=4293].)

- Nueva función: **`PyCode_NewEmpty()`** crea un objeto de código vacío; solo se requieren el nombre del archivo, el nombre de la función y el número de la primera línea. Esto resulta útil para los módulos de extensión que intentan construir una pila de seguimiento más útil. Anteriormente, dichas extensiones necesitaban llamar a `PyCode_New()`, que tenía muchos más argumentos. (Agregado por Jeffrey Yasskin).

- Nueva función: `PyErr_NewExceptionWithDoc()` crea una nueva clase de excepción, igual que la existente `PyErr_NewException()`, pero toma un argumento extra `char *` que contiene el docstring de la nueva clase de excepción. (Añadido por "lekma" en el bug tracker de Python; bpo-7033 [https://bugs.python.org/issue?@action=redirect&bpo=7033])

- Nueva función: `PyFrame_GetLineNumber()` toma un objeto frame y retorna el número de línea que el frame está ejecutando actualmente. Anteriormente, el código tenía que obtener el índice de la instrucción de código de bytes que se estaba ejecutando, y luego buscar el número de línea correspondiente a esa dirección. (Añadido por Jeffrey Yasskin)

- Nuevas funciones: `PyLong_AsLongAndOverflow()` y `PyLong_AsLongLongAndOverflow()` aproximan un entero largo de Python como C long o long long. Si el número es demasiado grande para caber en el tipo de salida, se establece un indicador *overflow* y se devuelve a la persona que llama. (Aportado por Case Van Horsen; bpo-7528 [https://bugs.python.org/issue?@action=redirect&bpo=7528] y bpo-7767 [https://bugs.python.org/issue?@action=redirect&bpo=7767].)

- Nueva función: a partir de la reescritura de la conversión de cadena a flotante, se agregó una nueva función `PyOS_string_to_double()`. Las antiguas funciones `PyOS_ascii_strtod()` y `PyOS_ascii_atof()` ahora están obsoletas.

- Nueva función: `PySys_SetArgvEx()` establece el valor de `sys.argv` y opcionalmente puede actualizar `sys.path` para incluir el directorio que contiene el script nombrado por `sys.argv[0]` dependiendo del valor de un parámetro *updatepath*.

Esta función se agregó para cerrar un agujero de seguridad en las aplicaciones que incorporan Python. La función anterior, **`PySys_SetArgv()`**, siempre actualizaba `sys.path` y, a veces, añadía el directorio actual. Esto significaba que, si ejecutaba una aplicación que incorporaba Python en un directorio controlado por otra persona, los atacantes podían colocar un módulo troyano en el directorio (por ejemplo, un archivo llamado `os.py`) que su aplicación luego importaría y ejecutaría.

Si mantiene una aplicación C/C++ que incorpora Python, verifique si está llamando a **`PySys_SetArgv()`** y considere cuidadosamente si la aplicación debería usar **`PySys_SetArgvEx()`** con *updatepath* establecido en falso.

Problema de seguridad informado como **CVE 2008-5983** [https://www.cve.org/CVERecord?id=CVE-2008-5983]; discutido en bpo-5753 [https://bugs.python.org/issue?@action=redirect&bpo=5753] y solucionado por Antoine Pitrou.

- New macros: the Python header files now define the following macros: **`Py_ISALNUM`**, **`Py_ISALPHA`**, **`Py_ISDIGIT`**, **`Py_ISLOWER`**, **`Py_ISSPACE`**, **`Py_ISUPPER`**, **`Py_ISXDIGIT`**, **`Py_TOLOWER`**, and **`Py_TOUPPER`**. All of these functions are analogous to the C standard macros for classifying characters, but ignore the current locale setting, because in several places Python needs to analyze characters in a locale-independent way. (Added by Eric Smith; bpo-5793 [https://bugs.python.org/issue?@action=redirect&bpo=5793].)

- Función eliminada: **`PyEval_CallObject()`** ahora solo está disponible como macro. Se mantuvo una versión de función para preservar la compatibilidad con enlaces ABI, pero eso fue en 1997; por lo tanto, ya se puede eliminar. (Eliminada por Antoine Pitrou; bpo-8276 [https://bugs.python.org/issue?@action=redirect&bpo=8276]).

- Nuevos códigos de formato: las funciones **`PyString_FromFormat()`**, **`PyString_FromFormatV()`** y **`PyErr_Format()`** ahora aceptan los códigos de formato `%lld` y `%llu` para mostrar los tipos long long de C.

(Contribuido por Mark Dickinson; bpo-7228 [https://bugs.python.org/issue?@action=redirect&bpo=7228].)

- Se ha cambiado la complicada interacción entre los hilos y la bifurcación de procesos. Anteriormente, el proceso hijo creado por `os.fork()` podía fallar porque el hijo se creaba con un solo hilo en ejecución, el hilo que realizaba el `os.fork()`. Si otros hilos estuvieran manteniendo un bloqueo, como el bloqueo de importación de Python, cuando se realizara el fork, el bloqueo seguiría marcado como «mantenido» en el nuevo proceso. Pero en el proceso hijo nada liberaría el bloqueo, ya que los otros hilos no estaban replicados, y el proceso hijo ya no podría realizar importaciones.

  Python 2.7 adquiere el bloqueo de importación antes de realizar un `os.fork()`, y también limpiará cualquier bloqueo creado usando el módulo `threading`. Los módulos de extensión C que tienen bloqueos internos, o que llaman a `fork()` ellos mismos, no se beneficiarán de esta limpieza.

  (Corregido por Thomas Wouters; bpo-1590864 [https://bugs.python.org/issue?@action=redirect&bpo=1590864].)

- La función `Py_Finalize()` ahora llama a la función interna `threading._shutdown()`; esto evita que se generen algunas excepciones cuando se apaga un intérprete. (Parche de Adam Olsen; bpo-1722344 [https://bugs.python.org/issue?@action=redirect&bpo=1722344]).

- When using the `PyMemberDef` structure to define attributes of a type, Python will no longer let you try to delete or set a `T_STRING_INPLACE` attribute.

- Los símbolos globales definidos por el módulo `ctypes` llevan ahora el prefijo `Py`, o bien `_ctypes`. (Implementado por Thomas Heller; bpo-3102 [https://bugs.python.org/issue?@action=redirect&bpo=3102].)

- Nueva opción de configuración: el conmutador `--with-system-expat` permite crear el módulo `pyexpat` para utilizar la biblioteca Expat del sistema. (Contribución de Arfrever Frehtes Taifersar Arahesis; bpo-7609 [https://bugs.python.org/issue?@action=redirect&bpo=7609]).

- Nueva opción de configuración: la opción **`--with-valgrind`** ahora desactivará el asignador pymalloc, que es difícil de analizar correctamente por el detector de errores de memoria de Valgrind. Por lo tanto, Valgrind detectará mejor las fugas de memoria y los desbordamientos. (Contribución de James Henstridge; bpo-2422 [https://bugs.python.org/issue?@action=redirect&bpo=2422].)

- Nueva opción de configuración: ahora puede suministrar una cadena vacía a **`--with-dbmliborder=`** para desactivar todos los módulos DBM. (Añadido por Arfrever Frehtes Taifersar Arahesis; bpo-6491 [https://bugs.python.org/issue?@action=redirect&bpo=6491])

- The **configure** script now checks for floating-point rounding bugs on certain 32-bit Intel chips and defines a **`X87_DOUBLE_ROUNDING`** preprocessor definition. No code currently uses this definition, but it's available if anyone wishes to use it. (Added by Mark Dickinson; bpo-2937 [https://bugs.python.org/issue?@action=redirect&bpo=2937].)

  configure also now sets a **`LDCXXSHARED`** Makefile variable for supporting C++ linking. (Contributed by Arfrever Frehtes Taifersar Arahesis; bpo-1222585 [https://bugs.python.org/issue?@action=redirect&bpo=1222585].)

- El proceso de construcción ahora crea los archivos necesarios para el soporte de pkg-config. (Contribución de Clinton Roy; bpo-3585 [https://bugs.python.org/issue?@action=redirect&bpo=3585].)

- El proceso de construcción ahora soporta Subversion 1.7. (Contribuido por Arfrever Frehtes Taifersar Arahesis; bpo-6094 [https://bugs.python.org/issue?@action=redirect&bpo=6094].)

Cápsulas

Python 3.1 añade un nuevo tipo de datos C, **`PyCapsule`**, para proporcionar una API C a un módulo de extensión. Una cápsula es esencialmente el soporte de un puntero C `void *`, y está disponible como un atributo del módulo; por ejemplo, la API del módulo **`socket`** se expone como `socket.CAPI`, y **`unicodedata`** expone `ucnhash_CAPI`. Otras extensiones pueden importar el módulo, acceder a su diccionario para obtener el objeto cápsula, y luego obtener el puntero `void *`, que normalmente apuntará a una matriz de punteros a las diversas funciones de la API del módulo.

Ya existe un tipo de datos que se utiliza para esto, **`PyCObject`**, pero no proporciona seguridad de tipos. Un código malicioso escrito en Python puro podría provocar un error de segmentación al tomar un **`PyCObject`** del módulo A y sustituirlo de alguna manera por el **`PyCObject`** del módulo B. Las cápsulas conocen su propio nombre y, para obtener el puntero, es necesario proporcionar el nombre:

```
void *vtable;

if (!PyCapsule_IsValid(capsule, "mymodule.CAPI") {
        PyErr_SetString(PyExc_ValueError, "argument type
        return NULL;
}

vtable = PyCapsule_GetPointer(capsule, "mymodule.CAPI");
```

Se asegura que `vtable` apunta a lo que se espera. Si se pasara una cápsula diferente, **`PyCapsule_IsValid()`** detectaría el nombre erróneo y retornaría false. Consulta Proporcionar una API C para un módulo de extensión para más información sobre el uso de estos objetos.

Python 2.7 ahora utiliza cápsulas internamente para proporcionar varias API de módulos de extensión, pero se modificó **`PyCObject_AsVoidPtr()`** para manejar cápsulas, lo que preserva la compatibilidad en tiempo de compilación con la interfaz **`PyCObject`**. El uso de **`PyCObject_AsVoidPtr()`** indicará un **`PendingDeprecationWarning`**, que es silencioso de manera predeterminada.

Implementado en Python 3.1 y portado a 2.7 por Larry Hastings; discutido en bpo-5630 [https://bugs.python.org/issue?@action=redirect&bpo=5630].

Cambios específicos en los puertos: Windows

- El módulo `msvcrt` ahora contiene algunas constantes del archivo de encabezado `crtassem.h`: `CRT_ASSEMBLY_VERSION`, `VC_ASSEMBLY_PUBLICKEYTOKEN` y `LIBRARIES_ASSEMBLY_NAME_PREFIX`. (Contribuido por David Cournapeau; bpo-4365 [https://bugs.python.org/issue?@action=redirect&bpo=4365]).
- El módulo `_winreg` para acceder al registro ahora implementa las funciones `CreateKeyEx()` y `DeleteKeyEx()`, versiones extendidas de funciones admitidas anteriormente que toman varios argumentos adicionales. También se probaron y documentaron las funciones `DisableReflectionKey()`, `EnableReflectionKey()` y `QueryReflectionKey()`. (Implementado por Brian Curtin: bpo-7347 [https://bugs.python.org/issue?@action=redirect&bpo=7347]).
- La nueva API `_beginthreadex()` se utiliza para iniciar subprocesos y ahora se utilizan las funciones nativas de almacenamiento local de subprocesos. (Contribución de Kristján Valur Jónsson; bpo-3582 [https://bugs.python.org/issue?@action=redirect&bpo=3582]).
- La función `os.kill()` ahora funciona en Windows. El valor de la señal puede ser las constantes `CTRL_C_EVENT`, `CTRL_BREAK_EVENT` o cualquier número entero. Las dos primeras constantes enviarán eventos de pulsación de teclas `Control-C` y `Control-Break` a los subprocesos; cualquier otro valor utilizará la API `TerminateProcess()`. (Contribuido por Miki Tebeka; bpo-1220212 [https://bugs.python.org/issue?@action=redirect&bpo=1220212].)
- La función `os.listdir()` ahora falla correctamente si la ruta está vacía. (Corregido por Hirokazu Yamamoto; bpo-5913 [https://bugs.python.org/issue?@action=redirect&bpo=5913].)
- El módulo `mimetypes` ahora leerá la base de datos MIME

del registro de Windows al inicializarse. (Parche de Gabriel Genellina; bpo-4969 [https://bugs.python.org/issue?@action=redirect&bpo=4969]).

Cambios específicos en los puertos: Mac OS X

- La ruta `/Library/Python/2.7/site-packages` se añade ahora a `sys.path`, para compartir los paquetes añadidos entre la instalación del sistema y una copia de la misma versión instalada por el usuario. (Cambiado por Ronald Oussoren; bpo-4865 [https://bugs.python.org/issue?@action=redirect&bpo=4865].)

  Distinto en la versión 2.7.13: A partir de la versión 2.7.13, se ha eliminado este cambio. `/Library/Python/2.7/site-packages`, el directorio site-packages utilizado por el sistema Python 2.7 suministrado por Apple ya no se añade a `sys.path` para los Python instalados por el usuario, como los instaladores de python.org. A partir de macOS 10.12, Apple ha cambiado la configuración del directorio site-packages del sistema, lo que puede hacer que falle la instalación de componentes pip, como setuptools. Los paquetes instalados para el sistema Python ya no se compartirán con los Python instalados por el usuario. (bpo-28440 [https://bugs.python.org/issue?@action=redirect&bpo=28440])

Cambios específicos en los puertos: FreeBSD

- La constante **`SO_SETFIB`** de FreeBSD 7.1, utilizada con los métodos **`getsockopt()`**/**`setsockopt()`** de **`socket()`** para seleccionar una tabla de enrutamiento alternativa, ahora está disponible en el módulo **`socket`**. (Agregado por Kyle VanderBeek; bpo-8235 [https://bugs.python.org/issue?@action=redirect&bpo=8235].)

Otros cambios y correcciones

- Dos scripts de evaluación, `iobench` y `ccbench`, han sido añadidos al directorio `Tools`. `iobench` mide la velocidad de los objetos de E/S de los archivos retornados por **`open()`** al realizar varias operaciones, y `ccbench` es una evaluación de concurrencia que trata de medir el rendimiento de la computación, la latencia del cambio de hilos y el ancho de banda del procesamiento de E/S al realizar varias tareas utilizando un número variable de hilos.

- El script `Tools/i18n/msgfmt.py` ahora entiende las formas plurales en los archivos `.po`. (Corregido por Martin von Löwis; bpo-5464 [https://bugs.python.org/issue?@action=redirect&bpo=5464].)
- Al importar un módulo desde un archivo `.pyc` o `.pyo` con un equivalente `.py` existente, los atributos **`co_filename`** de los objetos de código resultantes se sobrescriben cuando el nombre de archivo original queda obsoleto. Esto puede suceder si el archivo ha cambiado de nombre, se ha movido o se accede a él a través de rutas diferentes. (Parche de Ziga Seilnacht y Jean-Paul Calderone; bpo-1180193 [https://bugs.python.org/issue?@action=redirect&bpo=1180193]).
- El script `regrtest.py` ahora toma una opción **`--randseed=`** que toma un entero que será usado como semilla aleatoria para la opción **`-r`** que ejecuta las pruebas en orden aleatorio. La opción **`-r`** también informa de la semilla que se ha utilizado (Añadido por Collin Winter.)
- Otra opción `regrtest.py` es **`-j`**, que toma un número entero que especifica cuántas pruebas se ejecutan en paralelo. Esto permite reducir el tiempo total de ejecución en máquinas multinúcleo. Esta opción es compatible con varias otras opciones, incluyendo la opción **`-R`** que es conocida por producir tiempos de ejecución largos. (Añadido por Antoine Pitrou, bpo-6152 [https://bugs.python.org/issue?@action=redirect&bpo=6152].) También puede utilizarse con una nueva opción **`-F`** que ejecuta las pruebas seleccionadas en un bucle hasta que fallen. (Añadido por Antoine Pitrou; bpo-7312 [https://bugs.python.org/issue?@action=redirect&bpo=7312].)
- Cuando se ejecuta como un script, el módulo `py_compile.py` ahora acepta `'-'` como argumento, que leerá la entrada estándar para la lista de nombres de archivos a compilar. (Contribuido por Piotr Ożarowski; bpo-8233

[https://bugs.python.org/issue?@action=redirect&bpo=8233].)

Adaptación a Python 2.7

Esta sección enumera los cambios descritos anteriormente y otras correcciones de errores que pueden requerir cambios en su código:

- La función `range()` procesa sus argumentos de manera más consistente; ahora llamará a `__int__()` en los argumentos que no sean números flotantes ni enteros que se le proporcionen. (Corregido por Alexander Belopolsky; bpo-1533 [https://bugs.python.org/issue?@action=redirect&bpo=1533]).
- El método string `format()` ha cambiado la precisión por defecto utilizada para los números de punto flotante y complejos de 6 decimales a 12, que coincide con la precisión utilizada por `str()`. (Cambiado por Eric Smith; bpo-5920 [https://bugs.python.org/issue?@action=redirect&bpo=5920].)
- Debido a una optimización de la declaración `with`, los métodos especiales `__enter__()` y `__exit__()` deben pertenecer al tipo del objeto y no pueden adjuntarse directamente a la instancia del objeto. Esto afecta a las clases de nuevo estilo (derivadas de `object`) y a los tipos de extensión de C. (bpo-6101 [https://bugs.python.org/issue?@action=redirect&bpo=6101]).
- Debido a un error en Python 2.6, el parámetro *exc_value* de los métodos `__exit__()` era a menudo la representación de cadena de la excepción, no una instancia. Esto se solucionó en la versión 2.7, por lo que *exc_value* será una instancia como se esperaba. (Solucionado por Florent Xicluna; bpo-7853 [https://bugs.python.org/issue?@action=redirect&bpo=7853]).
- Cuando se establecía un conjunto restringido de atributos utilizando `__slots__`, la eliminación de un atributo no establecido no lanzaba `AttributeError` como cabría esperar. Corregido por Benjamin Peterson; bpo-7604 [https://bugs.python.org/issue?@action=redirect&bpo=7604])

En la biblioteca estándar:

- Las operaciones con instancias `datetime` que daban como resultado un año que caía fuera del rango soportado no siempre lanzaban `OverflowError`. Estos errores se comprueban ahora con más cuidado y se lanzará la excepción. (Informado por Mark Leander, parche de Anand B. Pillai y Alexander Belopolsky; bpo-7150 [https://bugs.python.org/issue?@action=redirect&bpo=7150])

- Cuando se utilizan instancias `Decimal` con el método `format()` de una cadena, la alineación por defecto era antes la alineación a la izquierda. Se ha cambiado a alineación derecha, lo que podría cambiar la salida de sus programas. (Cambiado por Mark Dickinson; bpo-6857 [https://bugs.python.org/issue?@action=redirect&bpo=6857].)

  Las comparaciones que implican un valor NaN de señalización (o `sNAN`) señalan ahora `InvalidOperation` en lugar de retornar silenciosamente un valor verdadero o falso dependiendo del operador de comparación. Los valores NaN silenciosos (o `NaN`) son ahora hashable. (Corregido por Mark Dickinson; bpo-7279 [https://bugs.python.org/issue?@action=redirect&bpo=7279].)

- La biblioteca `xml.etree.ElementTree` ya no omite los símbolos & y los corchetes angulares al generar una instrucción de procesamiento XML (que se parece a `<?xml-stylesheet href="#style1"?>`) o un comentario (que se parece a `<!-- comment -->`). (Parche de Neil Muller; bpo-2746 [https://bugs.python.org/issue?@action=redirect&bpo=2746]).

- El método **`readline()`** de los objetos `StringIO` ahora no hace nada cuando se solicita una longitud negativa, como lo hacen otros objetos similares a archivos. (bpo-7348 [https://bugs.python.org/issue?@action=redirect&bpo=7348]).

- El módulo `syslog` utilizará ahora el valor de `sys.argv[0]` como identificador en lugar del anterior valor por defecto de `'python`. (Cambiado por Sean Reifschneider; bpo-8451 [https://bugs.python.org/issue?@action=redirect&bpo=8451].)

- El manejo de errores por defecto del módulo **`tarfile`** ha cambiado, para no suprimir los errores fatales. El nivel de error por defecto era antes 0, lo que significaba que los errores sólo daban lugar a que se escribiera un mensaje en el registro de depuración, pero como el registro de depuración no está activado por defecto, estos errores pasan desapercibidos. El nivel de error por defecto es ahora 1, que lanza una excepción si hay un error. (Cambiado por Lars Gustäbel; bpo-7357 [https://bugs.python.org/issue?@action=redirect&bpo=7357].)

- El módulo **`urlsplit()`** del **`urlparse`** ahora maneja esquemas de URL desconocidos de una manera compatible con el **RFC 3986** [https://datatracker.ietf.org/doc/html/rfc3986.html]: si la URL tiene el formato `"<something>://..."`, el texto antes del `://` se trata como el esquema, incluso si es un esquema inventado que el módulo no conoce. Este cambio puede romper el código que funcionaba con el comportamiento anterior. Por ejemplo, Python 2.6.4 o 2.5 devolverá lo siguiente:

  ```
  >>> import urlparse
  >>> urlparse.urlsplit('invented://host/nombre_archi
  ('invented', '', '//host/nombre_archivo?consulta',
  ```

  Python 2.7 (y Python 2.6.5) retornará:

  ```
  >>> import urlparse
  >>> urlparse.urlsplit('invented://host/filename?que
  ('invented', 'host', '/filename?query', '', '')
  ```

  (Python 2.7 en realidad produce una salida ligeramente diferente, ya que retorna una tupla con nombre en lugar de una tupla estándar)

Para las extensiones en C:

- Las extensiones de C que utilizan códigos de formato entero con la familia de funciones `PyArg_Parse*` ahora lanzarán una excepción **`TypeError`** en lugar de provocar un **`DeprecationWarning`** (bpo-5080 [https://bugs.python.org/issue?

@action=redirect&bpo=5080]).

- Utilice la nueva función `PyOS_string_to_double()` en lugar de las antiguas funciones `PyOS_ascii_strtod()` y `PyOS_ascii_atof()`, que ahora están obsoletas.

Para aplicaciones que incrustan Python:

- Se agregó la función `PySys_SetArgvEx()`, que permite a las aplicaciones cerrar un agujero de seguridad cuando se utiliza la función `PySys_SetArgv()` existente. Verifique si está llamando a `PySys_SetArgv()` y considere cuidadosamente si la aplicación debería usar `PySys_SetArgvEx()` con *updatepath* configurado como falso.

Nuevas funciones añadidas a las versiones de mantenimiento de Python 2.7

Se pueden añadir nuevas características a las versiones de mantenimiento de Python 2.7 cuando la situación lo requiera realmente. Cualquier adición debe pasar por el proceso de Propuesta de Mejora de Python, y hacer un caso convincente de por qué no se puede abordar adecuadamente mediante la adición de la nueva característica únicamente a Python 3, o bien mediante su publicación en el Índice de Paquetes de Python.

Además de las propuestas específicas enumeradas a continuación, existe una exención general que permite añadir nuevas advertencias `-3` en cualquier versión de mantenimiento de Python 2.7.

Dos nuevas variables de entorno para el modo de depuración

En el modo de depuración, la estadística `[xxx refs]` no se escribe de manera predeterminada; ahora también se debe configurar la variable de entorno **PYTHONSHOWREFCOUNT**. (Contribución de Victor Stinner; bpo-31733 [https://bugs.python.org/issue?@action=redirect&bpo=31733]).

Cuando Python se compila con `COUNT_ALLOC` definido, los recuentos de asignación ya no se vuelcan de forma predeterminada: ahora también se debe configurar la variable de entorno **`PYTHONSHOWALLOCCOUNT`**. Además, los recuentos de asignación ahora se vuelcan en stderr, en lugar de stdout. (Contribuido por Victor Stinner; bpo-31692 [https://bugs.python.org/issue?@action=redirect&bpo=31692]).

Added in version 2.7.15.

PEP 434: Excepción de mejora de IDLE para todas las ramas

PEP 434 [https://peps.python.org/pep-0434/] describe una exención general para los cambios realizados en el entorno de desarrollo de IDLE que se envía junto con Python. Esta exención hace posible que los desarrolladores de IDLE proporcionen una experiencia de usuario más consistente en todas las versiones soportadas de Python 2 y 3.

Para conocer los detalles de los cambios de IDLE, consulte el archivo NEWS de la versión en cuestión.

PEP 466: Mejoras en la seguridad de la red para Python 2.7

PEP 466 [https://peps.python.org/pep-0466/] describe una serie de propuestas de mejora de la seguridad de la red que han sido aprobadas para su inclusión en las versiones de mantenimiento de Python 2.7, y el primero de esos cambios aparecerá en la versión 2.7.7 de Python.

Funciones relacionadas con *466* añadidas en Python 2.7.7:

- **`hmac.compare_digest()`** fue portado desde Python 3 para hacer una operación de comparación resistente a los ataques de tiempo disponible para las aplicaciones de Python 2. (Contribuido por Alex Gaynor; bpo-21306 [https://bugs.python.org/issue?@action=redirect&bpo=21306].)
- OpenSSL 1.0.1g fue actualizado en los instaladores oficiales

de Windows publicados en python.org. (Contribuido por Zachary Ware; bpo-21462 [https://bugs.python.org/issue?@action=redirect&bpo=21462].)

Funciones relacionadas con *466* añadidas en Python 2.7.8:

- **`hashlib.pbkdf2_hmac()`** fue portado desde Python 3 para hacer que un algoritmo de hash adecuado para el almacenamiento seguro de contraseñas esté ampliamente disponible para las aplicaciones de Python 2. (Contribuido por Alex Gaynor; bpo-21304 [https://bugs.python.org/issue?@action=redirect&bpo=21304].)
- OpenSSL 1.0.1h se actualizó para los instaladores oficiales de Windows publicados en python.org. (Contribución de Zachary Ware en bpo-21671 [https://bugs.python.org/issue?@action=redirect&bpo=21671] para **CVE 2014-0224** [https://www.cve.org/CVERecord?id=CVE-2014-0224].)

Funciones relacionadas con *466* añadidas en Python 2.7.9:

- La mayor parte del módulo **`ssl`** de Python 3.4 ha sido portado a esta versión. Esto significa que **`ssl`** ahora soporta la Indicación de Nombre de Servidor, la configuración de TLS1.x, el acceso al almacén de certificados de la plataforma, la clase **`SSLContext`**, y otras características. (Contribución de Alex Gaynor y David Reid; bpo-21308 [https://bugs.python.org/issue?@action=redirect&bpo=21308].)

  Consulte las notas «Versión añadida: 2.7.9» en la documentación del módulo para obtener detalles específicos.

- **`os.urandom()`** ha sido modificado para almacenar en caché un descriptor de fichero en `/dev/urandom` en lugar de reabrir `/dev/urandom` en cada llamada. (Contribuido por Alex Gaynor; bpo-21305 [https://bugs.python.org/issue?@action=redirect&bpo=21305].)

- **`hashlib.algorithms_guaranteed`** y **`hashlib.algorithms_available`** han sido portados desde Python 3 para facilitar a las aplicaciones de Python 2 la selección del algoritmo hash más potente disponible.

(Contribuido por Alex Gaynor en bpo-21307 [https://bugs.python.org/issue?@action=redirect&bpo=21307])

PEP 477: Backport ensurepip (PEP 453) a Python 2.7

PEP 477 [https://peps.python.org/pep-0477/] aprueba la inclusión del módulo PEP 453 [https://peps.python.org/pep-0453/] ensurepip y la documentación mejorada que ha permitido en las versiones de mantenimiento de Python 2.7, apareciendo primero en la versión 2.7.9 de Python.

Puesta en marcha de pip por defecto

El nuevo módulo **`ensurepip`** (definido en PEP 453 [https://peps.python.org/pep-0453/]) proporciona un mecanismo estándar multiplataforma para arrancar el instalador pip en las instalaciones de Python. La versión de `pip` incluida con Python 2.7.9 es `pip` 1.5.6, y las futuras versiones de mantenimiento de 2.7.x actualizarán la versión incluida a la última versión de `pip` que esté disponible en el momento de crear la versión candidata.

Por defecto, los comandos `pip`, `pipX` y `pipX.Y` se instalarán en todas las plataformas (donde X.Y representa la versión de la instalación de Python), junto con el paquete de Python `pip` y sus dependencias.

Para las construcciones de CPython source en sistemas POSIX, los comandos `make install` y `make altinstall` no arrancan `pip` por defecto. Este comportamiento puede ser controlado a través de las opciones de configuración, y anulado a través de las opciones del Makefile.</building-python-on-unix>

En Windows y Mac OS X, los instaladores de CPython ahora instalan por defecto `pip` junto con el propio CPython (los usuarios pueden optar por no instalarlo durante el proceso de instalación). Los usuarios de Windows tendrán que optar por las modificaciones automáticas del `PATH` para que `pip` esté disponible por defecto desde la línea de comandos, de lo contrario se puede acceder a través del lanzador de Python para Windows como `py -m pip`.

Como en **discussed in the PEP** [https://peps.python.org/pep-0477/#disabling-ensurepip-by-downstream-distributors], los empaquetadores de plataformas pueden elegir no instalar estos comandos de forma predeterminada, siempre que, cuando se invoquen, proporcionen instrucciones claras y simples sobre cómo instalarlos en esa plataforma (generalmente usando el administrador de paquetes del sistema).

Cambios en la documentación

Como parte de este cambio, las secciones Installing Python modules y Distribuir módulos de Python de la documentación se han rediseñado por completo como documentos breves de introducción y preguntas frecuentes. La mayor parte de la documentación de empaquetado ahora se ha trasladado a Python Packaging Authority mantenido Python Packaging User Guide [https://packaging.python.org] y la documentación de los proyectos individuales.

Sin embargo, como esta migración aún no está completa, las versiones anteriores de esas guías siguen estando disponibles como Building C and C++ Extensions with setuptools y Building C and C++ Extensions with setuptools.

Ver también

PEP 453 [https://peps.python.org/pep-0453/] – Arranque explícito de pip en instalaciones de Python

PEP escrito por Donald Stufft y Nick Coghlan, implementado por Donald Stufft, Nick Coghlan, Martin von Löwis y Ned Deily.

PEP 476: Habilitar la verificación de certificados por defecto para los clientes http stdlib

PEP 476 [https://peps.python.org/pep-0476/] actualizó `httplib` y los módulos que lo utilizan, como `urllib2` y `xmlrpclib`, para verificar que el servidor presente un certificado firmado por una autoridad de certificación en el almacén de confianza de la plataforma y cuyo nombre de host coincida con el nombre de host que se solicita de manera predeterminada, lo que mejora significativamente la seguridad para muchas aplicaciones. Este cambio se realizó en la versión Python 2.7.9.

Para las aplicaciones que requieren el antiguo comportamiento anterior, pueden pasar un contexto alternativo:

```
import urllib2
import ssl

# This disables all verification
context = ssl._create_unverified_context()

# This allows using a specific certificate for the host,
# to be in the trust store
context = ssl.create_default_context(cafile="/path/to/fi

urllib2.urlopen("https://invalid-cert", context=context)
```

PEP 493: Herramientas de migración de verificación HTTPS para Python 2.7

PEP 493 [https://peps.python.org/pep-0493/] proporciona herramientas de migración adicionales para soportar un proceso de actualización de la infraestructura más incremental para los entornos que contienen aplicaciones y servicios que dependen del procesamiento históricamente permisivo de los certificados de servidor cuando se establecen conexiones HTTPS de cliente. Estas adiciones se realizaron en la versión 2.7.12 de Python.

Estas herramientas están pensadas para su uso en casos en los que las aplicaciones y servicios afectados no pueden modificarse para pasar explícitamente un contexto SSL más permisivo al establecer la conexión.

Para las aplicaciones y servicios que no pueden ser modificados en absoluto, la nueva variable de entorno `PYTHONHTTPSVERIFY` puede establecerse a `0` para revertir todo un proceso de Python al comportamiento permisivo por defecto de Python 2.7.8 y anteriores.

Para los casos en los que no se puede modificar el código de establecimiento de conexión, pero sí la aplicación general, se puede utilizar la nueva función **`ssl._https_verify_certificates()`** para ajustar el comportamiento predeterminado en tiempo de ejecución.

Nuevo objetivo de construcción `make regen-all`

Para simplificar la compilación cruzada, y para asegurar que CPython pueda ser compilado de forma fiable sin requerir que una versión existente de Python esté ya disponible, el sistema de construcción basado en autotools ya no intenta recompilar implícitamente los archivos generados basándose en los tiempos de modificación de los archivos.

En su lugar, se ha añadido un nuevo comando `make regen-all` para forzar la regeneración de estos archivos cuando se desee (por ejemplo, después de que se haya construido una versión inicial de Python basada en las versiones pregeneradas).

También se definen objetivos de regeneración más selectivos - ver Makefile.pre.in [https://github.com/python/cpython/tree/3.14/Makefile.pre.in] para más detalles.

(Contribución de Victor Stinner en bpo-23404 [https://bugs.python.org/issue?@action=redirect&bpo=23404].)

Added in version 2.7.14.

Eliminación del objetivo de construcción `make touch`

Se ha eliminado el objetivo de compilación `make touch` que anteriormente se utilizaba para solicitar la regeneración implícita de los archivos generados mediante la actualización de sus tiempos de modificación.

Se ha sustituido por el nuevo objetivo `make regen-all`.

(Contribución de Victor Stinner en bpo-23404 [https://bugs.python.org/issue?@action=redirect&bpo=23404].)

Distinto en la versión 2.7.14.

Agradecimientos

El autor desea agradecer a las siguientes personas sus sugerencias, correcciones y ayuda en varios borradores de este artículo: Nick Coghlan, Philip Jenvey, Ryan Lovett, R. David Murray y Hugh Secker-Walker.

Qué hay de nuevo en Python 2.6

Autor:

A.M. Kuchling (amk arroba amk.ca)

Este artículo explica las nuevas funciones de Python 2.6, publicadas el 1 de octubre de 2008. El programa de publicación se describe en **PEP 361** [https://peps.python.org/pep-0361/].

The major theme of Python 2.6 is preparing the migration path to Python 3.0, a major redesign of the language. Whenever possible, Python 2.6 incorporates new features and syntax from 3.0 while remaining compatible with existing code by not removing older features or syntax. When it's not possible to do that, Python 2.6 tries to do what it can, adding compatibility functions in a `future_builtins` module and a `-3` switch to warn about usages that will become unsupported in 3.0.

Se han agregado algunos paquetes nuevos importantes a la biblioteca estándar, como los módulos `multiprocessing` y `json`, pero no hay muchas características nuevas que no estén relacionadas con Python 3.0 de alguna manera.

Python 2.6 también incluye una serie de mejoras y correcciones de errores en el código fuente. Una búsqueda en los registros de cambios encuentra que se aplicaron 259 parches y se corrigieron 612 errores entre Python 2.5 y 2.6. Es probable que ambas cifras estén subestimadas.

Este artículo no intenta proporcionar una especificación completa de las nuevas características, sino que proporciona una conveniente descripción general. Para obtener detalles completos, debe consultar la documentación de Python 2.6. Si desea comprender la justificación del diseño y la implementación, consulte el PEP de una característica nueva en particular. Siempre que sea posible, «Qué hay de nuevo en Python» enlaza con el elemento de error / parche para cada cambio.

Python 3.0

El ciclo de desarrollo de las versiones 2.6 y 3.0 de Python se sincronizó, y las versiones alfa y beta de ambos lanzamientos se realizaron los mismos días. El desarrollo de 3.0 ha influido en muchas características de 2.6.

Python 3.0 es un rediseño de Python de gran alcance que rompe la compatibilidad con la serie 2.x. Esto significa que el código Python existente necesitará alguna conversión para poder ejecutarse en Python 3.0. Sin embargo, no todos los cambios en 3.0 rompen necesariamente la compatibilidad. En los casos en que las nuevas funciones no provoquen la rotura del código existente, se han actualizado a 2.6 y se describen en este documento en el lugar correspondiente. Algunas de las características derivadas de 3.0 son:

- El método `__complex__()` para convertir objetos en un número complejo.
- Sintaxis alternativa para detectar excepciones: `except TypeError as exc`.
- La adición de `functools.reduce()` como sinónimo de la función incorporada `reduce()`.

Python 3.0 adds several new built-in functions and changes the semantics of some existing builtins. Functions that are new in 3.0 such as `bin()` have simply been added to Python 2.6, but existing builtins haven’t been changed; instead, the `future_builtins` module has versions with the new 3.0 semantics. Code written to be compatible with 3.0 can do `from future_builtins import hex, map` as necessary.

A new command-line switch, `-3`, enables warnings about features that will be removed in Python 3.0. You can run code with this switch to see how much work will be necessary to port code to 3.0. The value of this switch is available to Python code as the boolean variable `sys.py3kwarning`, and to C extension code as

`Py_Py3kWarningFlag`.

Ver también

La serie 3*xxx* de PEP, que contiene propuestas para Python 3.0. **PEP 3000** [https://peps.python.org/pep-3000/] describe el proceso de desarrollo para Python 3.0. Comience con **PEP 3100** [https://peps.python.org/pep-3100/], que describe los objetivos generales para Python 3.0, y luego explore los PEP de mayor número que proponen características específicas.

Cambios en el proceso de desarrollo

Mientras se desarrollaba 2.6, el proceso de desarrollo de Python experimentó dos cambios significativos: cambiamos del seguidor de incidentes (*issue tracker*) de SourceForge a una instalación personalizada de Roundup, y la documentación se convirtió de LaTeX a reStructuredText.

Nuevo seguidor de incidentes: Roundup

Durante mucho tiempo, los desarrolladores de Python estaban cada vez más molestos por el seguidor de errores de SourceForge. La solución alojada en SourceForge no permite mucha personalización; por ejemplo, no fue posible personalizar el ciclo de vida de los problemas.

Por lo tanto, el comité de infraestructura de la Python Software Foundation publicó una convocatoria de rastreadores de problemas, solicitando voluntarios para configurar diferentes productos e importar algunos de los errores y parches de SourceForge. Se examinaron cuatro rastreadores diferentes: Jira [https://www.atlassian.com/software/jira/], Launchpad [https://launchpad.net/], Roundup [https://roundup.sourceforge.io/] y Trac [https://trac.edgewall.org/]. El comité finalmente se decidió por Jira y Roundup como los dos candidatos. Jira es un producto comercial que ofrece instancias alojadas sin costo para proyectos de software libre; Roundup es un proyecto de código abierto que requiere voluntarios para administrarlo y un servidor para alojarlo.

Después de publicar una llamada para voluntarios, se configuró una nueva instalación de Roundup en https://bugs.python.org. Una instalación de Roundup puede alojar varios seguidores, y este servidor ahora también aloja seguidores de problemas para Jython y para el sitio web de Python. Seguramente encontrará otros usos en el futuro. Siempre que sea posible, esta edición de «Qué hay de nuevo en Python» se vincula al elemento de error/parche para cada cambio.

Upfront Systems [https://upfrontsoftware.co.za] de Stellenbosch, Sudáfrica, proporciona amablemente el alojamiento del rastreador de errores de Python. Martin von Löwis se esforzó mucho en importar errores y parches existentes desde SourceForge; Sus scripts para esta operación de importación están en `https://svn.python.org/view/tracker/importer/` y pueden ser útiles para otros proyectos que deseen pasar de SourceForge a Roundup.

Ver también

https://bugs.python.org
El seguidor de errores de Python.

https://bugs.jython.org:
El seguidor de errores de Jython.

https://roundup.sourceforge.io/
Descargas y documentación de Roundup.

https://svn.python.org/view/tracker/importer/
Scripts de conversión de Martin von Löwis.

Nuevo formato de documentación: texto reestructurado con Sphinx

La documentación de Python se escribió usando LaTeX desde que el proyecto comenzó alrededor de 1989. En la década de 1980 y principios de la de 1990, la mayor parte de la documentación se imprimió para su estudio posterior, no se vio en línea. LaTeX fue ampliamente utilizado porque proporcionaba una salida impresa atractiva sin dejar de ser sencillo de escribir una vez que se aprendían las reglas básicas de marcado.

Hoy en día, LaTeX todavía se usa para escribir publicaciones destinadas a la impresión, pero el panorama de las herramientas de programación ha cambiado. Ya no imprimimos montones de documentación; en su lugar, lo navegamos en línea y HTML se ha convertido en el formato más importante para dar soporte. Desafortunadamente, convertir LaTeX a HTML es bastante complicado y Fred L. Drake Jr., el editor de documentación de Python desde hace mucho tiempo, pasó mucho tiempo manteniendo el proceso de conversión. De vez en cuando, la gente sugeriría convertir la documentación a SGML y luego a XML, pero realizar una buena conversión es una tarea importante y nadie comprometió el tiempo necesario para terminar el trabajo.

Durante el ciclo de desarrollo 2.6, Georg Brandl se esforzó mucho en construir una nueva cadena de herramientas para procesar la documentación. El paquete resultante se llama Sphinx y está disponible en https://www.sphinx-doc.org/.

Sphinx se concentra en la salida HTML, produciendo HTML moderno y con un estilo atractivo; la salida impresa todavía se admite mediante la conversión a LaTeX. El formato de entrada es reStructuredText, una sintaxis de marcado que admite extensiones y directivas personalizadas que se usa comúnmente en la comunidad de Python.

Sphinx es un paquete independiente que se puede usar para escribir, y casi dos docenas de otros proyectos (enumerados en el sitio web de Sphinx [https://www.sphinx-doc.org/en/master/examples.html]) han adoptado Sphinx como su herramienta de documentación.

Ver también

Documentando Python [https://devguide.python.org/documenting/]
Describe cómo escribir para la documentación de Python.

Sphinx [https://www.sphinx-doc.org/]
Documentación y código para la cadena de herramientas Sphinx.

Docutils **[https://docutils.sourceforge.io]**
El analizador sintáctico y el conjunto de herramientas *reStructuredText* subyacentes.

PEP 343: La sentencia *"with"*

La versión anterior, Python 2.5, agregó la instrucción "`with`" como una característica opcional, para ser habilitada por una directiva `from __future__ import with_statement`. En 2.6, la instrucción ya no necesita estar habilitada especialmente; esto significa que ahora `with` es siempre una palabra clave. El resto de esta sección es una copia de la sección correspondiente del documento «Qué hay de nuevo en Python 2.5»; si está familiarizado con la declaración "`with`" de Python 2.5, puede omitir esta sección.

La sentencia "`with`" resuelve el código que anteriormente usaría bloques `try...finally` para garantizar que el código de limpieza se ejecute. En esta sección, discutiré como se usará comúnmente la declaración. En la siguiente sección, examinaré los detalles de la implementación y mostraré cómo escribir objetos para usar con esta declaración.

La sentencia "`with`" es una estructura de control de flujo cuya estructura básica es:

```
with expression [as variable]:
    with-block
```

Se evalúa la expresión y debería dar como resultado un objeto que admita el protocolo de gestión de contexto (es decir, que tenga los métodos `__enter__()` y `__exit__()`).

El objeto `__enter__()` se llama antes de que se ejecute *with-block* y, por lo tanto, puede ejecutar el código de configuración. También puede devolver un valor vinculado al nombre *variable*, si se proporciona. (Tenga en cuenta que *variable* es *not* al que se le asigna el resultado de *expression*).

Una vez finalizada la ejecución de *with-block*, se llama al método `__exit__()` del objeto, incluso si el bloque generó una excepción y, por lo tanto, puede ejecutar código de limpieza.

Algunos objetos estándar de Python ahora admiten el protocolo de administración de contexto y se pueden usar con la sentencia "`with`". Los objetos de archivo son un ejemplo:

```
with open('/etc/passwd', 'r') as f:
    for line in f:
        print line
        ... more processing code ...
```

Después de que se haya ejecutado esta sentencia, el objeto de archivo en *f* se habrá cerrado automáticamente, incluso si el bucle `for` generó una excepción en la mitad del bloque.

Nota

En este caso, *f* es el mismo objeto creado por `open()`, porque `__enter__()` retorna *self*.

Los *locks* y las condiciones variables del módulo `threading` también admiten la sentencia "`with`":

```
lock = threading.Lock()
with lock:
    # Critical section of code
    ...
```

El *lock* se adquiere antes de que se ejecute el bloque y siempre se libera una vez que este se completa.

The `localcontext()` function in the `decimal` module makes it easy to save and restore the current decimal context, which encapsulates the desired precision and rounding characteristics for computations:

```
from decimal import Decimal, Context, localcontext

# Displays with default precision of 28 digits
v = Decimal('578')
```

```
print v.sqrt()

with localcontext(Context(prec=16)):
    # All code in this block uses a precision of 16 digi
    # The original context is restored on exiting the bl
    print v.sqrt()
```

Escribiendo gestores de contexto

Por detrás, la sentencia “**`with`**” es bastante complicada. La mayoría de las personas solo usarán “**`with`**” en compañía de objetos existentes y no necesitan conocer estos detalles, por lo que puede omitir el resto de esta sección si lo desea. Los autores de nuevos objetos deberán comprender los detalles de la implementación subyacente y deben seguir leyendo.

Una explicación de alto nivel del protocolo de gestor de contexto es:

- La expresión se evalúa y debería dar como resultado un objeto llamado «gestor de contexto». El gestor de contexto debe tener los métodos **`__enter__()`** y **`__exit__()`**.
- Se llama al método **`__enter__()`** del gestor de contexto. El valor retornado se asigna a *VAR*. Si no hay una cláusula `as VAR`, el valor simplemente se descarta.
- Se ejecuta el código en *BLOCK*.
- Si *BLOCK* genera una excepción, se llama al método **`__exit__()`** del administrador de contexto con tres argumentos: los detalles de la excepción (`type, value, traceback`, los mismos valores devueltos por **`sys.exc_info()`**, que también pueden ser `None` si no se produjo ninguna excepción). El valor de retorno del método controla si se vuelve a generar una excepción: cualquier valor falso vuelve a generar la excepción y `True` provocará su supresión. Solo en raras ocasiones querrá suprimir la excepción, porque si lo hace, el autor del código que contiene la declaración “**`with`**” nunca se dará cuenta de que algo salió mal.
- Si *BLOCK* no lanzó una excepción, el método **`__exit__()`** continúa llamándose, pero *type*, *value* y *traceback* son todos `None`.

Pensemos en un ejemplo. No presentaré un código detallado, solo bosquejaré los métodos necesarios para una base de datos que admita transacciones.

(Para las personas que no están familiarizadas con la terminología de la base de datos: un conjunto de cambios en la base de datos se agrupa en una transacción. Las transacciones pueden confirmarse, lo que significa que todos los cambios se escriben en la base de datos, o deshacerse, lo que significa que todos los cambios se descartan y la base de datos no ha cambiado. Consulte cualquier libro de texto de base de datos para obtener más información.)

Supongamos que hay un objeto que representa una conexión de base de datos. Nuestro objetivo será permitir que el usuario escriba un código como este:

```
db_connection = DatabaseConnection()
with db_connection as cursor:
    cursor.execute('insert into ...')
    cursor.execute('delete from ...')
    # ... more operations ...
```

The transaction should be committed if the code in the block runs flawlessly or rolled back if there’s an exception. Here’s the basic interface for **`DatabaseConnection`** that I’ll assume:

```
class DatabaseConnection:
    # Database interface
    def cursor(self):
        "Returns a cursor object and starts a new transa
    def commit(self):
        "Commits current transaction"
    def rollback(self):
        "Rolls back current transaction"
```

El método **`__enter__()`** es bastante sencillo, ya que solo hay que iniciar una nueva transacción. Para esta aplicación, el objeto cursor resultante sería un resultado útil, por lo que el método lo devolverá. El usuario puede entonces añadir `as cursor` a su declaración “**`with`**” para vincular el cursor a un nombre de variable.

```
class DatabaseConnection:
    ...
    def __enter__(self):
        # Code to start a new transaction
        cursor = self.cursor()
        return cursor
```

El método `__exit__()` es el más complicado porque es donde se debe realizar la mayor parte del trabajo. El método debe verificar si se produjo una excepción. Si no hubo excepción, se confirma la transacción. Si hubo una excepción, se revierte la transacción.

En el siguiente código, la ejecución simplemente caerá al final de la función, retornando el valor predeterminado `None`. `None` es falso, por lo que la excepción se volverá a lanzar automáticamente. Si lo desea, puede ser más explícito y agregar una sentencia `return` en la ubicación marcada.

```
class DatabaseConnection:
    ...
    def __exit__(self, type, value, tb):
        if tb is None:
            # No exception, so commit
            self.commit()
        else:
            # Exception occurred, so rollback.
            self.rollback()
            # return False
```

El módulo contextlib

El módulo `contextlib` proporciona algunas funciones y un decorador que son útiles al escribir objetos para usar con la sentencia “`with`”.

The decorator is called `contextmanager()`, and lets you write a single generator function instead of defining a new class. The generator should yield exactly one value. The code up to the `yield` will be executed as the `__enter__()` method, and the value yielded will be the method’s return value that will get bound to the variable in the “`with`” statement’s `as` clause, if any. The code after the `yield` will be executed in the `__exit__()` method. Any exception raised in the block will be raised by the `yield` statement.

Usando este decorador, nuestro ejemplo de base de datos de la sección anterior podría escribirse como:

```
from contextlib import contextmanager

@contextmanager
def db_transaction(connection):
    cursor = connection.cursor()
    try:
        yield cursor
    except:
        connection.rollback()
        raise
    else:
        connection.commit()

db = DatabaseConnection()
with db_transaction(db) as cursor:
    ...
```

El módulo `contextlib` también tiene una función `nested(mgr1, mgr2, ...)` que combina varios gestores de contexto para que no necesite escribir sentencias “`with`” anidadas. En este ejemplo, se utiliza una única sentencia “`with`” que inicia una transacción de base de datos y adquiere un bloqueo del hilo:

```
lock = threading.Lock()
with nested (db_transaction(db), lock) as (cursor, locke
    ...
```

Finally, the `closing()` function returns its argument so that it can be bound to a variable, and calls the argument’s `.close()` method at the end of the block.

```
importar urllib, sys
```

```
desde contextlib importar closing

con closing(urllib.urlopen('http://www.yahoo.com')) como
para línea en f:
sys.stdout.write(line)
```

Ver también

PEP 343 [https://peps.python.org/pep-0343/] - La sentencia «with»
PEP escrito por Guido van Rossum y Nick Coghlan; implementado por Mike Bland, Guido van Rossum y Neal Norwitz. El PEP muestra el código generado para una sentencia “`with`”, que puede ser útil para aprender cómo la sentencia funciona.

La documentación para el módulo `contextlib`.

PEP 366: Importaciones relativas explícitas desde un módulo principal

El modificador de Python `-m` permite ejecutar un módulo como un script. Cuando ejecutabas un módulo que estaba ubicado dentro de un paquete, las importaciones relativas no funcionaban correctamente.

The fix for Python 2.6 adds a `module.__package__` attribute. When this attribute is present, relative imports will be relative to the value of this attribute instead of the `__name__` attribute.

PEP 302-style importers can then set `__package__` as necessary. The `runpy` module that implements the `-m` switch now does this, so relative imports will now work correctly in scripts running from inside a package.

PEP 370: Directorio de `site-packages` por usuario

Cuando ejecuta Python, la ruta de búsqueda del módulo `sys.path` generalmente incluye un directorio cuya ruta termina en `"site-packages"`. Este directorio está destinado a contener paquetes instalados localmente disponibles para todos los usuarios que utilizan una máquina o un sitio de instalación en particular.

Python 2.6 introduce una convención para directorios de sitios específicos del usuario. El directorio varía según la plataforma:

- Unix y Mac OS X: `~/.local/`
- Windows: `%APPDATA%/Python`

Dentro de este directorio, habrá subdirectorios específicos de versión, como `lib/python2.6/site-packages` en Unix/Mac OS y `Python26/site-packages` en Windows.

If you don’t like the default directory, it can be overridden by an environment variable. **`PYTHONUSERBASE`** sets the root directory used for all Python versions supporting this feature. On Windows, the directory for application-specific data can be changed by setting the **`APPDATA`** environment variable. You can also modify the `site.py` file for your Python installation.

La característica se puede desactivar por completo ejecutando Python con la opción **`-s`** o *seteando* la variable de entorno **`PYTHONNOUSERSITE`**.

Ver también

PEP 370: Directorio de `site-packages` por usuario
PEP escrito e implementado por Christian Heimes.

PEP 371: El paquete `multiprocessing`

El nuevo paquete **`multiprocessing`** permite a los programas de Python crear nuevos procesos que realizarán un cálculo y retornaran un resultado al padre. Los procesos padre e hijo pueden comunicarse mediante colas (*queues*) y tuberías (*pipes*), sincronizar sus operaciones mediante bloqueos y semáforos, y pueden compartir matrices simples de datos.

The `multiprocessing` module started out as an exact emulation of the `threading` module using processes instead of threads. That goal was discarded along the path to Python 2.6, but the general approach of the module is still similar. The fundamental class is the `Process`, which is passed a callable object and a collection of arguments. The `start()` method sets the callable running in a subprocess, after which you can call the `is_alive()` method to check whether the subprocess is still running and the `join()` method to wait for the process to exit.

Aquí hay un ejemplo simple donde el subproceso calculará un factorial. La función que realiza el cálculo está escrita de forma extraña, por lo que lleva mucho más tiempo cuando el argumento de entrada es un múltiplo de 4.

```
import time
from multiprocessing import Process, Queue


def factorial(queue, N):
    "Compute a factorial."
    # If N is a multiple of 4, this function will take m
    if (N % 4) == 0:
        time.sleep(.05 * N/4)

    # Calculate the result
    fact = 1L
    for i in range(1, N+1):
        fact = fact * i

    # Put the result on the queue
    queue.put(fact)

if __name__ == '__main__':
    queue = Queue()

    N = 5
    p = Process(target=factorial, args=(queue, N))
    p.start()
    p.join()

    result = queue.get()
    print 'Factorial', N, '=', result
```

Un **`Queue`** se usa para comunicar el resultado del factorial. El objeto **`Queue`** se almacena en una variable global. El proceso hijo usará el valor de la variable cuando se creó el hijo; porque es una **`Queue`**, padre e hijo pueden usar el objeto para comunicarse. (Si el padre cambiara el valor de la variable global, el valor del hijo no se vería afectado y viceversa).

Two other classes, **`Pool`** and **`Manager`**, provide higher-level interfaces. **`Pool`** will create a fixed number of worker processes, and requests can then be distributed to the workers by calling **`apply()`** or **`apply_async()`** to add a single request, and **`map()`** or **`map_async()`** to add a number of requests. The following code uses a **`Pool`** to spread requests across 5 worker processes and retrieve a list of results:

```
from multiprocessing import Pool

def factorial(N, dictionary):
    "Compute a factorial."
    ...
p = Pool(5)
result = p.map(factorial, range(1, 1000, 10))
for v in result:
    print v
```

Esto produce la siguiente salida:

```
1
39916800
51090942171709440000
8222838654177922817725562880000000
33452526613163807108170062053440751665152000000000
```

```
...
```

The other high-level interface, the `Manager` class, creates a separate server process that can hold master copies of Python data structures. Other processes can then access and modify these data structures using proxy objects. The following example creates a shared dictionary by calling the `dict()` method; the worker processes then insert values into the dictionary. (Locking is not done for you automatically, which doesn't matter in this example. **`Manager`**'s methods also include `Lock()`, `RLock()`, and `Semaphore()` to create shared locks.)

```
import time
from multiprocessing import Pool, Manager

def factorial(N, dictionary):
    "Compute a factorial."
    # Calculate the result
    fact = 1L
    for i in range(1, N+1):
        fact = fact * i

    # Store result in dictionary
    dictionary[N] = fact

if __name__ == '__main__':
    p = Pool(5)
    mgr = Manager()
    d = mgr.dict()          # Create shared dictionary

    # Run tasks using the pool
    for N in range(1, 1000, 10):
        p.apply_async(factorial, (N, d))

    # Mark pool as closed -- no more tasks can be added.
    p.close()

    # Wait for tasks to exit
    p.join()
    # Output results
    for k, v in sorted(d.items()):
        print k, v
```

Esto producirá la salida:

```
1 1
11 39916800
21 51090942171709440000
31 8222838654177922817725562880000000
41 33452526613163807108170062053440751665152000000000
51 155111875328738228022424301646930321106325972001698 61
```

Ver también

La documentación del módulo `multiprocessing`.

PEP 371 [https://peps.python.org/pep-0371/] - Adición del paquete de multiprocesamiento
PEP escrito por Jesse Noller y Richard Oudkerk; implementado por Richard Oudkerk y Jesse Noller.

PEP 3101: Formateo avanzado de cadena de caracteres

En Python 3.0, el operador `%` se complementa con un método de formato de cadena más potente, `format()`. La compatibilidad con el método `str.format()` se ha retroalimentado a Python 2.6.

En 2.6, tanto las cadenas de 8 bits como las Unicode tienen un método `.format()` que trata la cadena como una plantilla y toma los argumentos para formatear. La plantilla de formato utiliza llaves (`{`, `}`) como caracteres especiales:

```
>>> # Substitute positional argument 0 into the string.
>>> "User ID: {0}".format("root")
'User ID: root'
```

```
>>> # Use the named keyword arguments
>>> "User ID: {uid}   Last seen: {last_login}".format(
...    uid="root",
...    last_login = "5 Mar 2008 07:20")
'User ID: root   Last seen: 5 Mar 2008 07:20'
```

Las llaves se pueden escapar duplicándose:

```
>>> "Empty dict: {{}}".format()
"Empty dict: {}"
```

Los nombres de campo pueden ser números enteros que indican argumentos posicionales, como `{0}`, `{1}`, etc. o nombres de argumentos de palabras clave. También puede proporcionar nombres de campos compuestos que lean atributos o accedan a claves de diccionario:

```
>>> import sys
>>> print 'Platform: {0.platform}\nPython version: {0.ve
Platform: darwin
Python version: 2.6a1+ (trunk:61261M, Mar  5 2008, 20:29
[GCC 4.0.1 (Apple Computer, Inc. build 5367)]'

>>> import mimetypes
>>> 'Content-type: {0[.mp4]}'.format(mimetypes.types_map
'Content-type: video/mp4'
```

Tenga en cuenta que cuando utilice una notación de estilo diccionario como `[.mp4]`, no es necesario poner comillas alrededor de la cadena; se buscará el valor usando `.mp4` como clave. Las cadenas de caracteres que comienzan con un número se convertirán en entero. No puede escribir expresiones más complicadas dentro de una cadena de formato.

Hasta ahora hemos mostrado cómo especificar qué campo sustituir en la cadena resultante. El formato preciso utilizado también se puede controlar agregando dos puntos seguidos de un especificador de formato. Por ejemplo:

```
>>> # Field 0: left justify, pad to 15 characters
>>> # Field 1: right justify, pad to 6 characters
>>> fmt = '{0:15} ${1:>6}'
>>> fmt.format('Registration', 35)
'Registration    $    35'
>>> fmt.format('Tutorial', 50)
'Tutorial        $    50'
>>> fmt.format('Banquet', 125)
'Banquet         $   125'
```

Los especificadores de formato pueden hacer referencia a otros campos a través del anidamiento:

```
>>> fmt = '{0:{1}}'
>>> width = 15
>>> fmt.format('Invoice #1234', width)
'Invoice #1234  '
>>> width = 35
>>> fmt.format('Invoice #1234', width)
'Invoice #1234                      '
```

Se puede especificar la alineación de un campo dentro del ancho deseado:

Carácter	**Efecto**
< **(por defecto)**	Alinear a la izquierda
>	Alinear a la derecha
^	Centrado
=	(Solo para tipos numéricos) Relleno después del signo.

Los especificadores de formato también pueden incluir un tipo de presentación, que controla cómo se formatea el valor. Por ejemplo, los números de punto flotante pueden formatearse como un número general o en notación exponencial:

```
>>> '{0:g}'.format(3.75)
'3.75'
>>> '{0:e}'.format(3.75)
'3.750000e+00'
```

Hay una variedad de tipos de presentación disponibles. Consulte la documentación 2.6 para obtener una lista completa; aquí hay un ejemplo:

b	Binario. Emite el número en base 2.
c	Carácter. Convierte el número entero en el carácter Unicode correspondiente antes de imprimirlo.
d	Entero Decimal. Muestra el número en base 10.
o	Formato octal. Da salida al número en base 8.
x	Formato hexadecimal. Muestra el número en base 16, utilizando letras minúsculas para los dígitos superiores a 9.
e	Notación de exponente. Imprime el número en notación científica utilizando la letra “e” para indicar el exponente.
g	Formato general. Esto imprime el número como un número de punto fijo, a menos que el número sea demasiado grande, en cuyo caso cambia a la notación de exponente “e”.
n	Número. Es lo mismo que “g” (para flotantes) o “d” (para enteros), salvo que utiliza la configuración regional actual para insertar los caracteres separadores de números adecuados.
%	Porcentaje. Multiplica el número por 100 y lo muestra en formato fijo (“f”), seguido de un signo de porcentaje.

Classes and types can define a `__format__()` method to control how they’re formatted. It receives a single argument, the format specifier:

```
def __format__(self, format_spec):
    if isinstance(format_spec, unicode):
        return unicode(str(self))
    else:
        return str(self)
```

There’s also a `format()` builtin that will format a single value. It calls the type’s `__format__()` method with the provided specifier:

```
>>> format(75.6564, '.2f')
'75.66'
```

Ver también

Format string syntax

La documentación de referencia para los campos de formato.

PEP 3101 [https://peps.python.org/pep-3101/] - Formato avanzado de cadenas

PEP escrito por Talin. Implementado por Eric Smith.

PEP 3105: `print` como función

La sentencia `print` se convierte en la función `print()` en Python 3.0. Hacer de `print()` una función hace posible reemplazar la función haciendo `def print(...)` o importando una nueva función desde otro lugar.

Python 2.6 tiene una importación `__future__` que elimina `print` como sintaxis del lenguaje, permitiéndote usar la forma funcional en su lugar. Por ejemplo:

```
>>> from __future__ import print_function
>>> print('# de entradas', len(diccionario), file=sys.st
```

La firma de la nueva función es:

```
def print(*args, sep=' ', end='\n', file=None)
```

Los parámetros son:

- *args*: argumentos posicionales cuyos valores se imprimirán.
- *sep*: el separador que se imprimirá entre los argumentos.
- *end*: el texto final, que se imprimirá después de que se hayan emitido todos los argumentos.
- *archivo*: el objeto archivo al que se enviará la salida.

Ver también

PEP 3105 [https://peps.python.org/pep-3105/] - Hacer de la impresión una función

PEP escrito por Georg Brandl.

PEP 3110: Cambios en el manejo de excepciones

Un error que ocasionalmente cometen los programadores de Python es escribir el siguiente código:

```
La firma de la nueva función es::try:
    ...
except TypeError, ValueError:  # ¡Incorrecto!
    ...
```

El autor probablemente está tratando de atrapar ambas excepciones **`TypeError`** y **`ValueError`**, pero este código en realidad hace algo diferente: atrapará **`TypeError`** y vinculará el objeto de excepción resultante al nombre local `"ValueError"`. La excepción **`ValueError`** no será capturada en absoluto. El código correcto especifica una tupla de excepciones:

```
try:
    ...
except (TypeError, ValueError):
    ...
```

Este error se produce porque el uso de la coma aquí es ambiguo: ¿indica dos nodos diferentes en el árbol de análisis sintáctico, o un único nodo que es una tupla?

Python 3.0 hace que esto sea inequívoco al sustituir la coma por la palabra «as». Para atrapar una excepción y almacenar el objeto de excepción en la variable `exc`, debes escribir:

```
try:
    ...
except TypeError as exc:
    ...
```

Python 3.0 sólo soporta el uso de «as», y por lo tanto interpreta el primer ejemplo como la captura de dos excepciones diferentes. Python 2.6 soporta tanto la coma como «as», por lo que el código existente seguirá funcionando. Por lo tanto, sugerimos utilizar «as» cuando se escriba nuevo código Python que sólo se ejecutará con la versión 2.6.

Ver también

PEP 3110 [https://peps.python.org/pep-3110/] - Captura de excepciones en Python 3000
PEP escrito y ejecutado por Collin Winter.

PEP 3112: Literales de bytes

Python 3.0 adopta Unicode como el tipo de cadena fundamental del lenguaje y denota los literales de 8 bits de forma diferente, ya sea como `b'string` o utilizando un constructor **`bytes`**. Por compatibilidad futura, Python 2.6 añade **`bytes`** como sinónimo del tipo **`str`**, y también soporta la notación `b''`.

El tipo **`str`** de la versión 2.6 difiere del tipo **`bytes`** de la versión 3.0 en varios aspectos; el más notable es que el constructor es completamente diferente. En la 3.0, `bytes([65, 66, 67])` tiene 3 elementos, que contienen los bytes que representan `ABC`; en la 2.6, `bytes([65, 66, 67])` devuelve la cadena de 12 bytes que representa el **`str()`** de la lista.

El uso principal de **`bytes`** en 2.6 será escribir pruebas de tipo de objeto como `isinstance(x, bytes)`. Esto ayudará al convertidor de 2 a 3, que no puede decir si el código de 2.x pretende que las cadenas contengan caracteres o bytes de 8 bits; ahora puede utilizar **`bytes`** o **`str`** para representar su intención exactamente, y el código resultante también será correcto en Python 3.0.

También hay una importación `__future__` que hace que todos los literales de cadena se conviertan en cadenas Unicode. Esto significa que las secuencias de escape `u` pueden ser utilizadas para incluir caracteres Unicode:

```
from __future__ import unicode_literals
```

```
s = ('\u751f\u3080\u304e\u3000\u751f\u3054'
     '\u3081\u3000\u751f\u305f\u307e\u3054')

print len(s)               # 12 Unicode characters
```

A nivel de C, Python 3.0 renombrará el tipo de cadena de 8 bits existente, llamado **`PyStringObject`** en Python 2.x, a **`PyBytesObject`**. Python 2.6 utiliza `#define` para soportar el uso de los nombres **`PyBytesObject()`**, **`PyBytes_Check()`**, **`PyBytes_FromStringAndSize()`**, y todas las demás funciones y macros utilizadas con cadenas.

Las instancias del tipo **`bytes`** son inmutables al igual que las cadenas. Un nuevo tipo **`bytearray`** almacena una secuencia mutable de bytes:

```
>>> bytearray([65, 66, 67])
bytearray(b'ABC')
>>> b = bytearray(u'\u21ef\u3244', 'utf-8')
>>> b
bytearray(b'\xe2\x87\xaf\xe3\x89\x84')
>>> b[0] = '\xe3'
>>> b
bytearray(b'\xe3\x87\xaf\xe3\x89\x84')
>>> unicode(str(b), 'utf-8')
u'\u31ef \u3244'
```

Byte arrays support most of the methods of string types, such as **`startswith()`**/**`endswith()`**, **`find()`**/**`rfind()`**, and some of the methods of lists, such as **`append()`**, **`pop()`**, and **`reverse()`**.

```
>>> b = bytearray('ABC')
>>> b.append('d')
>>> b.append(ord('e'))
>>> b
bytearray(b'ABCde')
```

También existe la correspondiente API en C, con **`PyByteArray_FromObject()`**, **`PyByteArray_FromStringAndSize()`**, y varias otras funciones.

Ver también

PEP 3112 [https://peps.python.org/pep-3112/] - Literales de bytes en Python 3000

PEP escrito por Jason Orendorff; retroalimentado a 2.6 por Christian Heimes.

PEP 3116: Nueva biblioteca de E/S

Python's built-in file objects support a number of methods, but file-like objects don't necessarily support all of them. Objects that imitate files usually support `read()` and `write()`, but they may not support `readline()`, for example. Python 3.0 introduces a layered I/O library in the `io` module that separates buffering and text-handling features from the fundamental read and write operations.

Existen tres niveles de clases base abstractas proporcionadas por el módulo `io`:

- `RawIOBase` defines raw I/O operations: `read()`, `readinto()`, `write()`, `seek()`, `tell()`, `truncate()`, and `close()`. Most of the methods of this class will often map to a single system call. There are also `readable()`, `writable()`, and `seekable()` methods for determining what operations a given object will allow.

  Python 3.0 tiene implementaciones concretas de esta clase para archivos y sockets, pero Python 2.6 no ha reestructurado sus objetos archivo y socket de esta manera.

- `BufferedIOBase` is an abstract base class that buffers data in memory to reduce the number of system calls used, making I/O processing more efficient. It supports all of the methods of `RawIOBase`, and adds a `raw` attribute holding the underlying raw object.

  There are five concrete classes implementing this ABC. `BufferedWriter` and `BufferedReader` are for objects that support write-only or read-only usage that have a `seek()` method for random access. `BufferedRandom` objects support read and write access upon the same underlying stream, and `BufferedRWPair` is for objects such as TTYs that have both read and write operations acting upon unconnected streams of data. The `BytesIO` class supports reading, writing, and seeking over an in-memory buffer.

- `TextIOBase`: Provides functions for reading and writing strings (remember, strings will be Unicode in Python 3.0), and supporting universal newlines. `TextIOBase` defines the `readline()` method and supports iteration upon objects.

  There are two concrete implementations. `TextIOWrapper` wraps a buffered I/O object, supporting all of the methods for text I/O and adding a `buffer` attribute for access to the underlying object. `StringIO` simply buffers everything in memory without ever writing anything to disk.

  (En Python 2.6, `io.StringIO` se implementa en Python puro, por lo que es bastante lento. Por lo tanto, debe quedarse con el módulo `StringIO` o `cStringIO` existente por ahora. En algún momento, el módulo `io` de Python 3.0 se reescribirá en C para mayor velocidad, y tal vez la implementación de C se incorpore a las versiones 2.x).

En Python 2.6, las implementaciones subyacentes no han sido reestructuradas para construir sobre las clases del módulo `io`. El módulo se proporciona para facilitar la escritura de código compatible con la versión 3.0, y para ahorrar a los desarrolladores el esfuerzo de escribir sus propias implementaciones de búfer y E/S de texto.

Ver también

PEP 3116 [https://peps.python.org/pep-3116/] - Nueva E/S
PEP escrito por Daniel Stutzbach, Mike Verdone y Guido van Rossum. Código de Guido van Rossum, Georg Brandl, Walter Doerwald, Jeremy Hylton, Martin von Löwis, Tony Lownds y otros.

PEP 3118: Protocolo revisado de la memoria intermedia

El protocolo de búferes es una API de nivel C que permite a los tipos de Python intercambiar punteros a sus representaciones internas. Un archivo mapeado en memoria puede ser visto como un buffer de caracteres, por ejemplo, y esto permite que otro módulo como **`re`** trate los archivos mapeados en memoria como una cadena de caracteres a buscar.

Los principales usuarios del protocolo de búferes son los paquetes de procesamiento numérico como NumPy, que exponen la representación interna de los arrays para que los invocadores puedan escribir datos directamente en un array en lugar de pasar por una API más lenta. Este PEP actualiza el protocolo de búfer a la luz de la experiencia del desarrollo de NumPy, añadiendo una serie de nuevas características como la indicación de la forma de un array o el bloqueo de una región de memoria.

La nueva función más importante de la API en C es `PyObject_GetBuffer(PyObject *obj, Py_buffer *view, int flags)`, que toma un objeto y un conjunto de flags, y rellena la estructura `Py_buffer` con información sobre la representación en memoria del objeto. Los objetos pueden utilizar esta operación para bloquear la memoria en su lugar mientras un llamador externo podría estar modificando el contenido, por lo que hay un correspondiente `PyBuffer_Release(Py_buffer *view)` para indicar que el llamador externo ha terminado.

El argumento *flags* de **`PyObject_GetBuffer()`** especifica las restricciones de la memoria devuelta. Algunos ejemplos son:

- **`PyBUF_WRITABLE`** indica que la memoria debe ser grabable.
- **`PyBUF_LOCK`** solicita un bloqueo de sólo lectura o exclusivo en la memoria.
- **`PyBUF_C_CONTIGUOUS`** y **`PyBUF_F_CONTIGUOUS`** solicitan una disposición de matriz contigua en C (la última dimensión varía más rápidamente) o contigua en Fortran (la primera dimensión varía más rápidamente).

Dos nuevos códigos de argumento para `PyArg_ParseTuple()`, `s*` y `z*`, devuelven objetos buffer bloqueados para un parámetro.

Ver también

PEP 3118 [https://peps.python.org/pep-3118/] - Revisión del protocolo del buffer
PEP escrito por Travis Oliphant y Carl Banks; implementado por Travis Oliphant.

PEP 3119: Clases base abstractas

Some object-oriented languages such as Java support interfaces, declaring that a class has a given set of methods or supports a given access protocol. Abstract Base Classes (or ABCs) are an equivalent feature for Python. The ABC support consists of an `abc` module containing a metaclass called `ABCMeta`, special handling of this metaclass by the `isinstance()` and `issubclass()` builtins, and a collection of basic ABCs that the Python developers think will be widely useful. Future versions of Python will probably add more ABCs.

Let's say you have a particular class and wish to know whether it supports dictionary-style access. The phrase «dictionary-style» is vague, however. It probably means that accessing items with `obj[1]` works. Does it imply that setting items with `obj[2] = value` works? Or that the object will have `keys()`, `values()`, and `items()` methods? What about the iterative variants such as `iterkeys()`? `copy`and :meth:()`!update`? Iterating over the object with `iter()`?

The Python 2.6 `collections` module includes a number of different ABCs that represent these distinctions. `Iterable` indicates that a class defines `__iter__()`, and `Container` means the class defines a `__contains__()` method and therefore supports `x in y` expressions. The basic dictionary interface of getting items, setting items, and `keys()`, `values()`, and `items()`, is defined by the `MutableMapping` ABC.

Puedes derivar tus propias clases de un ABC particular para indicar que soportan la interfaz de ese ABC:

```
import collections

class Storage(collections.MutableMapping):
    ...
```

Alternatively, you could write the class without deriving from the desired ABC and instead register the class by calling the ABC’s `register()` method:

```
import collections

class Storage:
    ...

collections.MutableMapping.register(Storage)
```

For classes that you write, deriving from the ABC is probably clearer. The `register()` method is useful when you’ve written a new ABC that can describe an existing type or class, or if you want to declare that some third-party class implements an ABC. For example, if you defined a **`PrintableType`** ABC, it’s legal to do:

```
# Register Python's types
PrintableType.register(int)
PrintableType.register(float)
PrintableType.register(str)
```

Las clases deben obedecer la semántica especificada por un ABC, pero Python no puede comprobarlo; depende del autor de la clase entender los requisitos del ABC e implementar el código en consecuencia.

Para comprobar si un objeto es compatible con una determinada interfaz, ahora se puede escribir:

```
def func(d):
    if not isinstance(d, collections.MutableMapping):
        raise ValueError("Mapping object expected, not %
```

No sientas que ahora debes empezar a escribir muchas comprobaciones como en el ejemplo anterior. Python tiene una fuerte tradición de «duck-typing», donde la comprobación explícita de tipos nunca se hace y el código simplemente llama a los métodos de un objeto, confiando en que esos métodos estarán ahí y lanzando una excepción si no lo están. Sea juicioso al comprobar el ABC y hágalo sólo cuando sea absolutamente necesario.

Puedes escribir tu propio ABC utilizando `abc.ABCMeta` como metaclase en una definición de clase:

```
from abc import ABCMeta, abstractmethod

class Drawable():
    __metaclass__ = ABCMeta

    @abstractmethod
    def draw(self, x, y, scale=1.0):
        pass

    def draw_doubled(self, x, y):
        self.draw(x, y, scale=2.0)


class Square(Drawable):
    def draw(self, x, y, scale):
        ...
```

In the **`Drawable`** ABC above, the **`draw_doubled()`** method renders the object at twice its size and can be implemented in terms of other methods described in **`Drawable`**. Classes implementing this ABC therefore don't need to provide their own implementation of **`draw_doubled()`**, though they can do so. An implementation of **`draw()`** is necessary, though; the ABC can't provide a useful generic implementation.

You can apply the **`@~abc.abstractmethod`** decorator to methods such as **`draw()`** that must be implemented; Python will then raise an exception for classes that don't define the method. Note that the exception is only raised when you actually try to create an instance of a subclass lacking the method:

```
>>> class Circle(Drawable):
...     pass
...
>>> c = Circle()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class Circle with
>>>
```

Los atributos de datos abstractos pueden declararse utilizando el decorador `@abstractproperty`:

```
from abc import abstractproperty
...

@abstractproperty
def readonly(self):
   return self._x
```

Subclasses must then define a `readonly` property.

Ver también

PEP 3119 [https://peps.python.org/pep-3119/] - Presentación de las clases base abstractas

PEP escrito por Guido van Rossum y Talin. Implementado por Guido van Rossum. Adaptado a la versión 2.6 por Benjamin Aranguren, con Alex Martelli.

PEP 3127: Soporte y sintaxis de literales enteros

Python 3.0 cambia la sintaxis de los literales enteros octales (base-8), prefijándolos con «0o» o «0O» en lugar de un cero inicial, y añade soporte para los literales enteros binarios (base-2), señalados con un prefijo «0b» o «0B».

Python 2.6 no elimina el soporte para el 0 inicial que señala un número octal, pero sí añade soporte para «0o» y «0b»:

```
>>> 0o21, 2*8 + 1
(17, 17)
>>> 0b101111
47
```

La función `oct()` sigue devolviendo números prefijados con un cero a la izquierda, y la nueva función `bin()` devuelve la representación binaria de un número:

```
>>> oct(42)
'052'
>>> future_builtins.oct(42)
'0o52'
>>> bin(173)
'0b10101101'
```

Las funciones built-in `int()` y `long()` aceptarán ahora los prefijos «0o» y «0b» cuando se solicite base-8 o base-2, o cuando el argumento *base* sea cero (indicando que la base utilizada debe determinarse a partir de la cadena):

```
>>> int ('0o52', 0)
42
>>> int('1101', 2)
13
>>> int('0b1101', 2)
13
>>> int('0b1101', 0)
13
```

Ver también

PEP 3127 [https://peps.python.org/pep-3127/] - Soporte y sintaxis de literales enteros

PEP escrito por Patrick Maupin; retroalimentado a 2.6 por Eric Smith.

PEP 3129: Decoradores de clase

Los decoradores se han ampliado de funciones a clases. Ahora es legal escribir:

```
@foo
@bar
class A:
  pass
```

Esto equivale a:

```
class A:
  pass

A = foo(bar(A))
```

Ver también

PEP 3129 [https://peps.python.org/pep-3129/] - Decoradores de clase

PEP escrito por Collin Winter.

PEP 3141: Una jerarquía de tipos para los números

Python 3.0 añade varias clases base abstractas para tipos numéricos inspiradas en la torre numérica de Scheme. Estas clases fueron retrocedidas a la versión 2.6 como el módulo **`numbers`**.

El ABC más general es **`Number`**. No define ninguna operación, y sólo existe para permitir comprobar si un objeto es un número haciendo `isinstance(obj, Number)`.

`Complex` es una subclase de **`Number`**. Los números complejos pueden someterse a las operaciones básicas de suma, resta, multiplicación, división y exponenciación, y se pueden recuperar las partes real e imaginaria y obtener el conjugado de un número. El tipo complejo incorporado en Python es una implementación de `Complex`.

`Real` deriva a su vez de `Complex`, y añade operaciones que sólo funcionan con números reales: `floor()`, `trunc()`, redondeo, toma del resto mod N, división entera a la baja y comparaciones.

Los números `Rational` derivan de `Real`, tienen las propiedades `numerator` y `denominator`, y se pueden convertir en flotantes. Python 2.6 añade una clase simple de números racionales, `Fraction`, en el módulo `fractions`. (Se llama `Fraction` en lugar de `Rational` para evitar un choque de nombres con `numbers.Rational`)

Los números de `Integral` derivan de `Rational`, y pueden desplazarse a la izquierda y a la derecha con < y >`, combinarse utilizando operaciones de bits como `&` y |, y pueden utilizarse como índices de matrices y límites de cortes.

En Python 3.0, el PEP redefine ligeramente las funciones integradas existentes `round()`, `math.floor()`, `math.ceil()`, y añade uno nuevo, `math.trunc()`, que ha sido retrocedido a Python 2.6. `math.trunc()` redondea hacia cero, devolviendo el `Integral` más cercano que esté entre el argumento de la función y cero.

Ver también

PEP 3141 [https://peps.python.org/pep-3141/] - Una jerarquía de tipos para los números

PEP escrito por Jeffrey Yasskin.

Torre numérica del esquema [https://www.gnu.org/software/guile/manual/html_node/Numerical-Tower.html#Numerical-Tower], del manual de Guile.

Scheme’s number datatypes [https://conservatory.scheme.org/schemers/Documents/Standards/R5RS/HTML/r5rs-Z-H-9.html#%_sec_6.2] de la especificación del esquema R5RS.

El módulo `fractions`

Para completar la jerarquía de tipos numéricos, el módulo **`fractions`** proporciona una clase de números racionales. Los números racionales almacenan sus valores como un numerador y un denominador que forman una fracción, y pueden representar exactamente números como `2/3` que los números de punto flotante sólo pueden aproximar.

El constructor **`Fraction`** toma dos valores **`Integral`** que serán el numerador y el denominador de la fracción resultante.

```
>>> from fractions import Fraction
>>> a = Fraction(2, 3)
>>> b = Fraction(2, 5)
>>> float(a), float(b)
(0.66666666666666663, 0.40000000000000002)
>>> a+b
Fraction(16, 15)
>>> a/b
Fraction(5, 3)
```

Para convertir números de punto flotante en racionales, el tipo float ahora tiene un método **`as_integer_ratio()`** que devuelve el numerador y el denominador de una fracción que evalúa el mismo valor de punto flotante:

```
>>> (2.5) .as_integer_ratio()
(5, 2)
>>> (3.1415) .as_integer_ratio()
(7074029114692207L, 2251799813685248L)
>>> (1./3) .as_integer_ratio()
(6004799503160661L, 18014398509481984L)
```

Tenga en cuenta que los valores que sólo pueden ser aproximados por números de punto flotante, como 1,/3, no se simplifican al número que se está aproximando; la fracción intenta coincidir con el valor de punto flotante **exactamente**.

El módulo **`fractions`** se basa en una implementación de Sjoerd Mullender que estuvo en el directorio `Demo/classes/` de Python durante mucho tiempo. Esta implementación fue significativamente actualizada por Jeffrey Yasskin.

Otros cambios lingüísticos

Algunos de los cambios más pequeños realizados en el núcleo del lenguaje Python son:

- Los directorios y archivos zip que contengan un archivo `__main__.py` pueden ahora ejecutarse directamente pasando su nombre al intérprete. El directorio o archivo zip se inserta automáticamente como la primera entrada en sys.path. (Sugerencia y parche inicial de Andy Chu, revisado posteriormente por Phillip J. Eby y Nick Coghlan; bpo-1739468 [https://bugs.python.org/issue?@action=redirect&bpo=1739468])

- La función **`hasattr()`** estaba capturando e ignorando todos los errores, bajo la suposición de que significaban que un método **`__getattr__()`** estaba fallando de alguna manera y que el valor de retorno de **`hasattr()`** sería por tanto `False`. Esta lógica no debería aplicarse a **`KeyboardInterrupt`** y **`SystemExit`**, sin embargo; Python 2.6 ya no descartará tales excepciones cuando **`hasattr()`** las encuentre. (Corregido por Benjamin Peterson; bpo-2196 [https://bugs.python.org/issue?@action=redirect&bpo=2196].)

- Cuando se llama a una función utilizando la sintaxis `**` para proporcionar argumentos de palabras clave, ya no es necesario utilizar un diccionario de Python; ahora funcionará cualquier asignación:

  ```
  >>> def f(**kw):
  ...    print sorted(kw)
  ...
  >>> ud=UserDict.UserDict()
  >>> ud['a'] = 1
  >>> ud['b'] = 'string'
  >>> f(**ud)
  ['a', 'b']
  ```

(Contribución de Alexander Belopolsky; bpo-1686487 [https://bugs.python.org/issue?@action=redirect&bpo=1686487].)

También se ha convertido en legal proporcionar argumentos de palabras clave después de un argumento `*args` a una llamada de función.

```
>>> def f(*args, **kw):
...     print args, kw
...
>>> f(1,2,3, *(4,5,6), keyword=13)
(1, 2, 3, 4, 5, 6) {'keyword': 13}
```

Anteriormente, esto habría sido un error de sintaxis. (Contribución de Amaury Forgeot d'Arc; bpo-3473 [https://bugs.python.org/issue?@action=redirect&bpo=3473].)

- Un nuevo builtin, `next(iterator, [default])` devuelve el siguiente elemento del iterador especificado. Si se suministra el argumento *default*, se devolverá si *iterador* se ha agotado; en caso contrario, se lanzará la excepción **`StopIteration`**. (Se ha modificado en bpo-2719 [https://bugs.python.org/issue?@action=redirect&bpo=2719].)

- Tuples now have **`index()`** and **`count()`** methods matching the list type's **`index()`** and **`count()`** methods:

  ```
  >>> t = (0,1,2,3,4,0,1,2)
  >>> t.index(3)
  3
  >>> t.count(0)
  2
  ```

  (Contribución de Raymond Hettinger)

- Los tipos incorporados tienen ahora un soporte mejorado para la sintaxis de corte extendida, aceptando varias combinaciones de `(inicio, parada, paso)`. Anteriormente, el soporte era parcial y algunos casos de esquina no funcionaban. (Implementado por Thomas Wouters)

- Las propiedades tienen ahora tres atributos, **`getter`**, **`setter`** y **`deleter`**, que son decoradores que proporcionan atajos útiles para añadir una función getter, setter o deleter a una propiedad existente. Los usarás así:

  ```
  class C(object):
      @property
      def x(self):
          return self._x

      @x.setter
      def x(self, value):
          self._x = value

      @x.deleter
      def x(self):
          del self._x

  class D(C):
      @C.x.getter
      def x(self):
          return self._x * 2

      @x.setter
      def x(self, value):
          self._x = value / 2
  ```

- Varios métodos de los tipos de conjuntos incorporados aceptan ahora múltiples iterables: **`intersection()`**, **`intersection_update()`**, **`union()`**, **`update()`**, **`difference()`** y **`difference_update()`**.

  ```
  >>> s=set('1234567890')
  >>> s.intersection('abc123', 'cdf246')  # Intersect
  set(['2'])
  >>> s.difference('246', '789')
  set(['1', '0', '3', '5'])
  ```

  (Contribución de Raymond Hettinger.)

- Se han añadido muchas funciones de punto flotante. La función **`float()`** ahora convertirá la cadena `nan` en un valor IEEE 754 Not A Number, y `+inf` y `-inf` en infinito positivo o negativo. Esto funciona en cualquier plataforma con semántica IEEE 754. (Contribución de Christian Heimes; bpo-1635 [https://bugs.python.org/issue?@action=redirect&bpo=1635].)

  Otras funciones del módulo **`math`**, **`isinf()`** y **`isnan()`**, devuelven true si su argumento en coma flotante es infinito o No es un número. (bpo-1640 [https://bugs.python.org/issue?@action=redirect&bpo=1640])

  Se han añadido funciones de conversión para convertir números de punto flotante en cadenas hexadecimales (bpo-3008 [https://bugs.python.org/issue?@action=redirect&bpo=3008]). Estas funciones convierten los números flotantes a y desde una representación de cadena sin introducir errores de redondeo por la conversión entre decimal y binario. Los flotadores tienen un método **`hex()`** que devuelve una representación de cadena, y el método `float.fromhex()` convierte una cadena de nuevo en un número:

  ```
  >>> a = 3.75
  >>> a.hex()
  '0x1.e000000000000p+1'
  >>> float.fromhex('0x1.e000000000000p+1')
  3.75
  >>> b=1./3
  >>> b.hex()
  '0x1.5555555555555p-2'
  ```

- Un detalle numérico: cuando se crea un número complejo a partir de dos flotantes en sistemas que admiten ceros con signo (-0 y +0), el constructor **`complex()`** conserva ahora el signo del cero. (Corregido por Mark T. Dickinson; bpo-1507 [https://bugs.python.org/issue?@action=redirect&bpo=1507].)

- Las clases que heredan un método **`__hash__()`** de una clase padre pueden establecer `__hash__ = None` para indicar que la clase no es hashable. Esto hará que `hash(obj)` lance un **`TypeError`** y la clase no será indicada como implementando el ABC de **`Hashable`**.

Debes hacer esto cuando hayas definido un método **`__cmp__()`** o **`__eq__()`** que compare objetos por su valor en lugar de por su identidad. Todos los objetos tienen un método hash por defecto que utiliza `id(obj)` como valor hash. No hay una forma ordenada de eliminar el método **`__hash__()`** heredado de una clase padre, por lo que la asignación de `None` fue implementada como una sobreescritura. A nivel de C, las extensiones pueden establecer `tp_hash` a **`PyObject_HashNotImplemented()`**. (Corregido por Nick Coghlan y Amaury Forgeot d'Arc; bpo-2235 [https://bugs.python.org/issue?@action=redirect&bpo=2235].)

- La excepción **`GeneratorExit`** ahora subclasa **`BaseException`** en lugar de **`Exception`**. Esto significa que un manejador de excepciones que haga `except Exception:` no atrapará inadvertidamente **`GeneratorExit`**. (Contribuido por Chad Austin; bpo-1537 [https://bugs.python.org/issue?@action=redirect&bpo=1537].)

- Los objetos generadores tienen ahora un atributo **`gi_code`** que hace referencia al objeto de código original que respalda al generador. (Contribución de Collin Winter; bpo-1473257 [https://bugs.python.org/issue?@action=redirect&bpo=1473257].)

- La función incorporada **`compile()`** ahora acepta argumentos de palabras clave así como parámetros posicionales. (Contribución de Thomas Wouters; bpo-1444529 [https://bugs.python.org/issue?@action=redirect&bpo=1444529].)

- El constructor **`complex()`** ahora acepta cadenas que contengan números complejos entre paréntesis, lo que significa que `complex(repr(cplx))` ahora redondeará los valores. Por ejemplo, `complex('(3+4j)')` ahora devuelve el valor (3+4j). (bpo-1491866 [https://bugs.python.org/issue?

@action=redirect&bpo=1491866])

- El método string **`translate()`** acepta ahora `None` como parámetro de la tabla de traducción, que se trata como la transformación de identidad. Esto facilita la realización de operaciones que sólo eliminan caracteres. (Contribuido por Bengt Richter e implementado por Raymond Hettinger; bpo-1193128 [https://bugs.python.org/issue?@action=redirect&bpo=1193128])

- La función incorporada **`dir()`** ahora comprueba si existe un método **`__dir__()`** en los objetos que recibe. Este método debe devolver una lista de cadenas que contengan los nombres de los atributos válidos para el objeto, y permite al objeto controlar el valor que **`dir()`** produce. Los objetos que tienen métodos **`__getattr__()`** o **`__getattribute__()`** pueden usar esto para anunciar los pseudo-atributos que respetarán. (bpo-1591665 [https://bugs.python.org/issue?@action=redirect&bpo=1591665])

- Los objetos de método de instancia tienen nuevos atributos para el objeto y la función que componen el método; el nuevo sinónimo de **`im_self`** es **`__self__`**, y **`im_func`** también está disponible como **`__func__`**. Los nombres antiguos aún se admiten en Python 2.6, pero ya no están disponibles en la versión 3.0.

- Un cambio oscuro: cuando se utiliza la función **`locals()`** dentro de una sentencia **`class`**, el diccionario resultante ya no devuelve variables libres. (Las variables libres, en este caso, son variables referenciadas en la sentencia **`class`** que no son atributos de la clase)

Optimizaciones

- El módulo **`warnings`** ha sido reescrito en C. Esto hace posible invocar advertencias desde el analizador sintáctico, y también puede hacer que el arranque del intérprete sea más rápido. (Contribuido por Neal Norwitz y Brett Cannon; bpo-1631171 [https://bugs.python.org/issue?@action=redirect&bpo=1631171].)

- Los objetos de tipo tienen ahora una caché de métodos que puede reducir el trabajo necesario para encontrar la implementación del método correcto para una clase en particular; una vez almacenada en la caché, el intérprete no necesita recorrer las clases base para averiguar el método correcto a llamar. La caché se borra si una clase base o la propia clase se modifica, por lo que la caché debería seguir siendo correcta incluso ante la naturaleza dinámica de Python. (Optimización original implementada por Armin Rigo, actualizada para Python 2.6 por Kevin Jacobs; bpo-1700288 [https://bugs.python.org/issue?@action=redirect&bpo=1700288].)

  Por defecto, este cambio sólo se aplica a los tipos que se incluyen con el núcleo de Python. Los módulos de extensión no son necesariamente compatibles con esta caché, por lo que deben añadir explícitamente **`Py_TPFLAGS_HAVE_VERSION_TAG`** al campo `tp_flags` del módulo para habilitar la caché de métodos. (Para ser compatible con la caché de métodos, el código del módulo de extensión no debe acceder directamente ni modificar el miembro `tp_dict` de ninguno de los tipos que implementa. La mayoría de los módulos no lo hacen, pero es imposible que el intérprete de Python lo determine. Ver bpo-1878 [https://bugs.python.org/issue?@action=redirect&bpo=1878] para una discusión)

- Las llamadas a funciones que utilizan argumentos de palabras clave son significativamente más rápidas al hacer una comparación rápida de punteros, lo que suele ahorrar el tiempo de una comparación completa de cadenas. (Contribución de Raymond Hettinger, tras una implementación inicial de Antoine Pitrou; bpo-1819 [https://bugs.python.org/issue?@action=redirect&bpo=1819])

- Todas las funciones del módulo **`struct`** han sido reescritas en C, gracias al trabajo en el sprint de Need For Speed. (Contribución de Raymond Hettinger)

- Algunos de los tipos estándar incorporados ahora establecen un bit en sus objetos de tipo. Esto acelera la comprobación de si un objeto es una subclase de uno de estos tipos. (Contribución de Neal Norwitz)

- Las cadenas Unicode utilizan ahora un código más rápido para detectar los espacios en blanco y los saltos de línea; esto acelera el método **`split()`** en un 25% a y **`splitlines()`** en un 35%. (Contribuido por Antoine Pitrou.) El uso de la memoria se reduce utilizando pymalloc para los datos de la cadena Unicode.

- La declaración `with` ahora almacena el método **`__exit__()`** en la pila, lo que produce una pequeña mejora en la velocidad. (Implementado por Jeffrey Yasskin).

- Para reducir el uso de memoria, el recolector de basura ahora borrará las listas libres internas cuando recolecte la generación más alta de objetos. Esto puede devolver la memoria al sistema operativo antes.

Cambios de intérprete

Two command-line options have been reserved for use by other Python implementations. The **`-J`** switch has been reserved for use by Jython for Jython-specific options, such as switches that are passed to the underlying JVM. **`-X`** has been reserved for options specific to a particular implementation of Python such as CPython, Jython, or IronPython. If either option is used with Python 2.6, the interpreter will report that the option isn't currently used.

Ahora se puede evitar que Python escriba archivos `.pyc` o `.pyo` proporcionando el modificador **`-B`** al intérprete de Python, o estableciendo la variable de entorno **`PYTHONDONTWRITEBYTECODE`** antes de ejecutar el intérprete. Esta configuración está disponible para los programas de Python como la variable `sys.dont_write_bytecode`, y el código de Python puede cambiar el valor para modificar el comportamiento del intérprete. (Contribución de Neal Norwitz y Georg Brandl)

La codificación utilizada para la entrada, la salida y el error estándar puede especificarse estableciendo la variable de entorno **`PYTHONIOENCODING`** antes de ejecutar el intérprete. El valor debe ser una cadena de la forma `<encoding>` o `<encoding>:<errorhandler>`. La parte *encoding* especifica el nombre de la codificación, por ejemplo `utf-8` o `latin-1`; la parte opcional *errorhandler* especifica qué hacer con los caracteres que no pueden ser manejados por la codificación, y debe ser una de las opciones «error», «ignorar» o «reemplazar». (Contribución de Martin von Löwis)</errorhandler></encoding></encoding>

Módulos nuevos y mejorados

Como en cada versión, la biblioteca estándar de Python ha recibido una serie de mejoras y correcciones de errores. Aquí hay una lista parcial de los cambios más notables, ordenados alfabéticamente por nombre de módulo. Consulta el archivo `Misc/NEWS` en el árbol de fuentes para una lista más completa de cambios, o mira los registros de Subversion para todos los detalles.

- Los módulos **`asyncore`** y **`asynchat`** están recibiendo mantenimiento activo nuevamente y se aplicaron varios parches y correcciones de errores. (Mantenimiento a cargo de Josiah Carlson; consulte bpo-1736190 [https://bugs.python.org/issue?@action=redirect&bpo=1736190] para obtener un parche).

- El módulo **`bsddb`** también tiene un nuevo mantenedor, Jesús Cea Avión, y el paquete está ahora disponible como paquete independiente. La página web del paquete es www.jcea.es/programacion/pybsddb.htm [https://www.jcea.es/programacion/pybsddb.htm]. El plan es eliminar el paquete de la biblioteca estándar en Python 3.0, porque su ritmo de lanzamientos es mucho más frecuente que el de Python.

  El módulo **`bsddb.dbshelve`** utiliza ahora el protocolo de decapado más alto disponible, en lugar de limitarse al protocolo 1. (Contribución de W. Barnes.)

- El módulo **`cgi`** ahora leerá variables de la cadena de consulta de una solicitud HTTP POST. Esto permite utilizar acciones de formulario con URL que incluyen cadenas de consulta como «/cgi-bin/add.py?category=1». (Contribuido por Alexandre Fiori y Nubis; bpo-1817 [https://bugs.python.org/

issue?@action=redirect&bpo=1817]).

Las funciones `parse_qs()` y `parse_qsl()` se han reubicado del módulo `cgi` al módulo `urlparse`. Las versiones que aún están disponibles en el módulo `cgi` activarán los mensajes `PendingDeprecationWarning` en la versión 2.6 (bpo-600362 [https://bugs.python.org/issue?@action=redirect&bpo=600362]).

- El módulo `cmath` ha sido objeto de una amplia revisión, a la que han contribuido Mark Dickinson y Christian Heimes. Se han añadido cinco nuevas funciones:
  - `polar()` convierte un número complejo en forma polar, devolviendo el módulo y el argumento del número complejo.
  - `rect()` hace lo contrario, convirtiendo un par de módulos y argumentos en el correspondiente número complejo.
  - `phase()` devuelve el argumento (también llamado ángulo) de un número complejo.
  - `isnan()` devuelve True si la parte real o imaginaria de su argumento es un NaN.
  - `isinf()` devuelve True si la parte real o imaginaria de su argumento es infinita.

  Las revisiones también han mejorado la solidez numérica del módulo `cmath`. Para todas las funciones, las partes real e imaginaria de los resultados son exactas con unas pocas unidades de precisión mínima (ulps) siempre que sea posible. Véase bpo-1381 [https://bugs.python.org/issue?@action=redirect&bpo=1381] para los detalles. También se han corregido los cortes de rama para `asinh()`, `atanh()`: y `atan()`.

  Las pruebas del módulo se han ampliado considerablemente; casi 2000 nuevos casos de prueba ejercitan las funciones algebraicas.

  En las plataformas IEEE 754, el módulo `cmath` maneja ahora los valores especiales IEEE 754 y las excepciones de punto flotante de forma consistente con el Anexo “G” del estándar C99.

- Un nuevo tipo de datos en el módulo `collections`: `namedtuple(typename, fieldnames)` es una función de fábrica que crea subclases de la tupla estándar cuyos campos son accesibles tanto por nombre como por índice. Por ejemplo:

  ```
  >>> var_type = collections.namedtuple('variable',
  ...              'id name type size')
  >>> # Names are separated by spaces or commas.
  >>> # 'id, name, type, size' would also work.
  >>> var_type._fields
  ('id', 'name', 'type', 'size')

  >>> var = var_type(1, 'frequency', 'int', 4)
  >>> print var[0], var.id    # Equivalent
  1 1
  >>> print var[2], var.type  # Equivalent
  int int
  >>> var._asdict()
  {'size': 4, 'type': 'int', 'id': 1, 'name': 'freque
  >>> v2 = var._replace(name='amplitude')
  >>> v2
  variable(id=1, name='amplitude', type='int', size=4
  ```

  Varios lugares de la biblioteca estándar que devolvían tuplas han sido modificados para devolver instancias de **`namedtuple()`**. Por ejemplo, el método **`Decimal.as_tuple()`** ahora devuelve una tupla con nombre con los campos **`sign`**, **`digits`** y **`exponent`**.

  (Contribución de Raymond Hettinger.)

- Otro cambio en el módulo `collections` es que el tipo **`deque`** admite ahora un parámetro opcional *maxlen*; si se suministra, el tamaño del deque estará restringido a no más de *maxlen* elementos. Si se añaden más elementos a un deque lleno, se descartan los elementos antiguos.

```
>>> from collections import deque
>>> dq=deque(maxlen=3)
>>> dq
deque([], maxlen=3)
>>> dq.append(1); dq.append(2); dq.append(3)
>>> dq
deque([1, 2, 3], maxlen=3)
>>> dq.append(4)
>>> dq
deque([2, 3, 4], maxlen=3)
```

(Contribución de Raymond Hettinger.)

- Los objetos `Morsel` del módulo `Cookie` ahora admiten un atributo `httponly`. En algunos navegadores, el código JavaScript no puede acceder ni manipular las cookies con este atributo establecido. (Contribuido por Arvin Schnell; bpo-1638033 [https://bugs.python.org/issue?@action=redirect&bpo=1638033]).

- Un nuevo método de ventana en el módulo `curses`, **`chgat()`**, cambia los atributos de visualización para un determinado número de caracteres en una sola línea. (Contribución de Fabian Kreutz)

  ```
  # Boldface text starting at y=0,x=21
  # and affecting the rest of the line.
  stdscr.chgat(0, 21, curses.A_BOLD)
  ```

  La clase **`Textbox`** del módulo `curses.textpad` soporta ahora la edición en modo de inserción así como en modo de sobrescritura. El modo de inserción se activa proporcionando un valor verdadero para el parámetro *insert_mode* al crear la instancia de **`Textbox`**.

- Los métodos **`strftime()`** del módulo `datetime` soportan ahora un código de formato `%f` que se expande al número de microsegundos en el objeto, rellenado de cero a la izquierda hasta seis lugares. (Contribución de Skip Montanaro; bpo-1158 [https://bugs.python.org/issue?

@action=redirect&bpo=1158].)

- El módulo **decimal** se actualizó a la versión 1.66 de the General Decimal Specification [https://speleotrove.com/decimal/decarith.html]. Las nuevas características incluyen algunos métodos para algunas funciones matemáticas básicas como **exp()** y **log10()**:

  ```
  >>> Decimal(1).exp()
  Decimal("2.718281828459045235360287471")
  >>> Decimal("2.7182818").ln()
  Decimal("0.9999999895305022877376682436")
  >>> Decimal(1000).log10()
  Decimal("3")
  ```

  El método **as_tuple()** de los objetos **Decimal** devuelve ahora una tupla con nombre con los campos **signo**, **dígitos** y **exponente**.

  (Implementado por Facundo Batista y Mark Dickinson. Soporte de tuplas con nombre añadido por Raymond Hettinger)

- La clase **difflib** del módulo **SequenceMatcher** devuelve ahora tuplas con nombre que representan coincidencias, con atributos **a**, **b** y **size**. (Contribución de Raymond Hettinger)

- Se ha añadido un parámetro opcional `timeout`, que especifica un tiempo de espera medido en segundos, al constructor de la clase **ftplib.FTP** así como al método **connect()**. (Añadido por Facundo Batista.) Además, los métodos **storbinary()** y **storlines()** de la clase **FTP** ahora toman un parámetro opcional *callback* que será llamado con cada bloque de datos después de que los datos hayan sido enviados. (Contribuido por Phil Schwartz; bpo-1221598 [https://bugs.python.org/issue?@action=redirect&bpo=1221598].)

- La función incorporada **reduce()** también está disponible en el módulo **functools**. En Python 3.0, la función incorporada se ha eliminado y **reduce()** sólo está disponible en **`functools`**; actualmente no hay planes para eliminar la función incorporada en la serie 2.x. (Parcheado por Christian Heimes; bpo-1739906 [https://bugs.python.org/issue?@action=redirect&bpo=1739906].)

- Cuando sea posible, el módulo **`getpass`** utilizará ahora `/dev/tty` para imprimir un mensaje de aviso y leer la contraseña, retrocediendo al error estándar y a la entrada estándar. Si la contraseña puede ser enviada al terminal, se imprimirá una advertencia antes de que se muestre el aviso. (Contribución de Gregory P. Smith)

- La función **`glob.glob()`** ahora puede devolver nombres de archivo Unicode si se ha utilizado una ruta Unicode y se han encontrado nombres de archivo Unicode dentro del directorio. (bpo-1001604 [https://bugs.python.org/issue?@action=redirect&bpo=1001604])

- Una nueva función en el módulo **`heapq`**, `merge(iter1, iter2, ...)`, toma cualquier número de iterables que devuelven datos en orden ordenado, y devuelve un nuevo generador que devuelve el contenido de todos los iteradores, también en ordenado. Por ejemplo:

  ```
  >>> list(heapq.merge([1, 3, 5, 9], [2, 8, 16]))
  [1, 2, 3, 5, 8, 9, 16]
  ```

  Otra nueva función, `heappushpop(heap, item)`, empuja *item* a *heap*, luego lo saca y devuelve el elemento más pequeño. Esto es más eficiente que hacer una llamada a **`heappush()`** y luego a **`heappop()`**.

  `heapq` se ha implementado para utilizar únicamente la comparación menor que, en lugar de la comparación menor o igual que utilizaba anteriormente. Esto hace que el uso de **`heapq`** de un tipo coincida con el método **`list.sort()`**. (Contribución de Raymond Hettinger)

- Se agregó un parámetro opcional `timeout` a los constructores de clases **`httplib.HTTPConnection`** y **`HTTPSConnection`**, que especifica un tiempo de espera medido en segundos. (Agregado por Facundo Batista).

- La mayoría de las funciones del módulo **`inspect`**, como **`getmoduleinfo()`** y **`getargs()`**, ahora devuelven tuplas con nombre. Además de comportarse como tuplas, también se puede acceder a los elementos del valor devuelto como atributos. (Contribución de Raymond Hettinger)

  Algunas de las nuevas funciones del módulo son **`isgenerator()`**, **`isgeneratorfunction()`** y **`isabstract()`**.

- El módulo **`itertools`** ha ganado varias funciones nuevas.

  `izip_longest(iter1, iter2, ...[, fillvalue])` hace tuplas de cada uno de los elementos; si algunos de los iterables son más cortos que otros, los valores que faltan se ponen a *fillvalue*. Por ejemplo:

  ```
  >>> tuple(itertools.izip_longest([1,2,3], [1,2,3,4,
  ((1, 1), (2, 2), (3, 3), (None, 4), (None, 5))
  ```

  `product(iter1, iter2, ..., [repeat=N])` devuelve el producto cartesiano de los iterables suministrados, un conjunto de tuplas que contiene todas las combinaciones posibles de los elementos devueltos de cada iterable.

  ```
  >>> list(itertools.product([1,2,3], [4,5,6]))
  [(1, 4), (1, 5), (1, 6),
   (2, 4), (2, 5), (2, 6),
   (3, 4), (3, 5), (3, 6)]
  ```

  El argumento opcional de la palabra clave *repeat* se utiliza para tomar el producto de un iterable o un conjunto de iterables con ellos mismos, repetido *N* veces. Con un único argumento iterable, se devuelven *N*-tuplas:

  ```
  >>> list(itertools.product([1,2], repeat=3))
  [(1, 1, 1), (1, 1, 2), (1, 2, 1), (1, 2, 2),
   (2, 1, 1), (2, 1, 2), (2, 2, 1), (2, 2, 2)]
  ```

Con dos iterables, se devuelven *2N*-tuplas.

```
>>> list(itertools.product([1,2], [3,4], repeat=2))
[(1, 3, 1, 3), (1, 3, 1, 4), (1, 3, 2, 3), (1, 3, 2
 (1, 4, 1, 3), (1, 4, 1, 4), (1, 4, 2, 3), (1, 4, 2
 (2, 3, 1, 3), (2, 3, 1, 4), (2, 3, 2, 3), (2, 3, 2
 (2, 4, 1, 3), (2, 4, 1, 4), (2, 4, 2, 3), (2, 4, 2
```

`combinaciones(iterable, r)` devuelve subsecuencias de longitud *r* de los elementos de *iterable*.

```
>>> list(itertools.combinations('123', 2))
[('1', '2'), ('1', '3'), ('2', '3')]
>>> list(itertools.combinations('123', 3))
[('1', '2', '3')]
>>> list(itertools.combinations('1234', 3))
[('1', '2', '3'), ('1', '2', '4'),
 ('1', '3', '4'), ('2', '3', '4')]
```

`permutaciones(iter[, r])` devuelve todas las permutaciones de longitud *r* de los elementos del iterable. Si no se especifica *r*, por defecto será el número de elementos producidos por el iterable.

```
>>> list(itertools.permutations([1,2,3,4], 2))
[(1, 2), (1, 3), (1, 4),
 (2, 1), (2, 3), (2, 4),
 (3, 1), (3, 2), (3, 4),
 (4, 1), (4, 2), (4, 3)]
```

`itertools.chain(*iterables)` es una función existente en **`itertools`** que obtuvo un nuevo constructor en Python 2.6. `itertools.chain.from_iterable(iterable)` toma un único iterable que debe devolver otros iterables. **`chain()`** devolverá entonces todos los elementos del primer iterable, luego todos los elementos del segundo, y así sucesivamente.

```
>>> list(itertools.chain.from_iterable([[1,2,3], [4
[1, 2, 3, 4, 5, 6]
```

(Todo ello aportado por Raymond Hettinger)

- La clase `logging` del módulo `FileHandler` y sus subclases `WatchedFileHandler`, `RotatingFileHandler` y `TimedRotatingFileHandler` tienen ahora un parámetro opcional *delay* en sus constructores. Si *retraso* es verdadero, la apertura del fichero de registro se retrasa hasta que se realice la primera llamada `emit()`. (Contribuido por Vinay Sajip.)

  `TimedRotatingFileHandler` también tiene un parámetro constructor *utc*. Si el argumento es verdadero, se utilizará la hora UTC para determinar cuándo ocurre la medianoche y para generar los nombres de los archivos; en caso contrario, se utilizará la hora local.

- Se han añadido varias funciones nuevas al módulo `math`:

  - `isinf()` y `isnan()` determinan si un flotador dado es un infinito (positivo o negativo) o un NaN (Not a Number), respectivamente.
  - `copysign()` copia el bit de signo de un número IEEE 754, devolviendo el valor absoluto de *x* combinado con el bit de signo de *y*. Por ejemplo, `math.copysign(1, -0.0)` devuelve -1.0. (Contribución de Christian Heimes)
  - `factorial()` calcula el factorial de un número. (Contribuido por Raymond Hettinger; bpo-2138 [https://bugs.python.org/issue?@action=redirect&bpo=2138].)
  - `fsum()` suma el flujo de números de un iterable, y tiene cuidado de evitar la pérdida de precisión mediante el uso de sumas parciales. (Contribución de Jean Brouwers, Raymond Hettinger y Mark Dickinson; bpo-2819 [https://bugs.python.org/issue?@action=redirect&bpo=2819])
  - `acosh()`, `asinh()` y `atanh()` calculan las funciones hiperbólicas inversas.
  - `log1p()` devuelve el logaritmo natural de $1+x$ (base *e*).
  - `trunc()` redondea un número hacia cero, devolviendo el `Integral` más cercano que esté entre el argumento

de la función y el cero. Añadido como parte del backport de la jerarquía de tipos de números de PEP 3141.

- El módulo `math` se ha mejorado para ofrecer un comportamiento más coherente en todas las plataformas, especialmente en lo que respecta al manejo de las excepciones de punto flotante y los valores especiales IEEE 754.

  Siempre que es posible, el módulo sigue las recomendaciones del estándar C99 sobre los valores especiales de 754. Por ejemplo, `sqrt(-1.)` debería dar ahora un `ValueError` en casi todas las plataformas, mientras que `sqrt(float('NaN'))` debería devolver un NaN en todas las plataformas IEEE 754. Cuando el Anexo "F" del estándar C99 recomiende señalar "dividir por cero" o "inválido", Python lanzará `ValueError`. Cuando el Anexo "F" del estándar C99 recomiende señalar "desbordamiento", Python lanzará `OverflowError`. (Ver bpo-711019 [https://bugs.python.org/issue?@action=redirect&bpo=711019] y bpo-1640 [https://bugs.python.org/issue?@action=redirect&bpo=1640])

  (Contribución de Christian Heimes y Mark Dickinson)

- Los objetos `mmap` tienen ahora un método `rfind()` que busca una subcadena empezando por el final de la cadena y buscando hacia atrás. El método `find()` también ha ganado un parámetro *end* que da un índice en el que parar la búsqueda. (Contribución de John Lenton)

- El módulo `operator` ganó una función `methodcaller()` que toma un nombre y un conjunto opcional de argumentos, devolviendo un callable que llamará a la función nombrada en cualquier argumento que se le pase. Por ejemplo:

  ```
  >>> # Equivalent to lambda s: s.replace('old', 'new
  >>> replacer = operator.methodcaller('replace', 'ol
  >>> replacer('old wine in old bottles')
  'new wine in new bottles'
  ```

  (Contribución de Georg Brandl, tras una sugerencia de

Gregory Petrosyan)

La función **`attrgetter()`** acepta ahora nombres con puntos y realiza las correspondientes búsquedas de atributos:

```
>>> inst_name = operator.attrgetter(
...         '__class__.__name__')
>>> inst_name('')
'str'
>>> inst_name(help)
'_Helper'
```

(Contribución de Georg Brandl, tras una sugerencia de Barry Warsaw)

- El módulo **`os`** incluye ahora varias llamadas nuevas al sistema. `fchmod(fd, mode)` y `fchown(fd, uid, gid)` cambian el modo y la propiedad de un fichero abierto, y `lchmod(path, mode)` cambia el modo de un enlace simbólico. (Contribución de Georg Brandl y Christian Heimes)

  `chflags()` y **`lchflags()`** son envoltorios para las correspondientes llamadas al sistema (cuando están disponibles), que cambian las banderas establecidas en un archivo. Las constantes para los valores de las banderas se definen en el módulo **`stat`**; algunos valores posibles son **`UF_IMMUTABLE`** para indicar que el fichero no puede ser modificado y **`UF_APPEND`** para indicar que sólo se pueden añadir datos al fichero. (Contribución de M. Levinson)

  `os.closerange(low, high)` cierra eficazmente todos los descriptores de fichero desde *low* hasta *high*, ignorando cualquier error y sin incluir a *high* mismo. Esta función es ahora utilizada por el módulo **`subprocess`** para hacer más rápido el arranque de procesos. (Contribuido por Georg Brandl; bpo-1663329 [https://bugs.python.org/issue?@action=redirect&bpo=1663329].)

- El método **`clear()`** del objeto `os.environ` ahora desajustará las variables de entorno utilizando **`os.unsetenv()`** además de borrar las claves del objeto.

(Contribuido por Martin Horcicka; bpo-1181 [https://bugs.python.org/issue?@action=redirect&bpo=1181].)

- La función `os.walk()` tiene ahora un parámetro `followlinks`. Si se establece como True, seguirá los enlaces simbólicos que apunten a directorios y visitará el contenido del directorio. Por compatibilidad con el pasado, el valor por defecto del parámetro es false. Tenga en cuenta que la función puede caer en una recursión infinita si hay un enlace simbólico que apunte a un directorio padre. (bpo-1273829 [https://bugs.python.org/issue?@action=redirect&bpo=1273829])
- En el módulo `os.path`, la función **`splitext()`** ha sido modificada para no dividir los caracteres de punto inicial. Esto produce mejores resultados cuando se opera con archivos de puntos de Unix. Por ejemplo, `os.path.splitext('.ipython')` ahora devuelve `('.ipython', '')` en lugar de `('', '.ipython')`. (bpo-1115886 [https://bugs.python.org/issue?@action=redirect&bpo=1115886])

  Una nueva función, `os.path.relpath(path, start='.')`, devuelve una ruta relativa desde la ruta `start`, si se suministra, o desde el directorio de trabajo actual hasta el destino `path`. (Contribución de Richard Barran; bpo-1339796 [https://bugs.python.org/issue?@action=redirect&bpo=1339796].)

  En Windows, `os.path.expandvars()` ahora expandirá las variables de entorno dadas en la forma «%var%», y «~user» se expandirá en la ruta del directorio personal del usuario. (Contribución de Josiah Carlson; bpo-957650 [https://bugs.python.org/issue?@action=redirect&bpo=957650].)
- El depurador de Python proporcionado por el módulo `pdb` ganó un nuevo comando: «run» reinicia el programa Python que se está depurando y puede tomar opcionalmente nuevos argumentos de línea de comandos para el programa. (Contribución de Rocky Bernstein; bpo-1393667 [https://bugs.python.org/issue?@action=redirect&bpo=1393667].)

- La función `pdb.post_mortem()`, utilizada para iniciar la depuración de un traceback, utilizará ahora el traceback devuelto por `sys.exc_info()` si no se suministra ningún traceback. (Contribuido por Facundo Batista; bpo-1106316 [https://bugs.python.org/issue?@action=redirect&bpo=1106316].)

- El módulo `pickletools` tiene ahora una función `optimize()` que toma una cadena que contiene un pickle y elimina algunos opcodes no utilizados, devolviendo un pickle más corto que contiene la misma estructura de datos. (Contribución de Raymond Hettinger)

- Se ha añadido una función `get_data()` al módulo `pkgutil` que devuelve el contenido de los archivos de recursos incluidos con un paquete Python instalado. Por ejemplo:

  ```
  >>> import pkgutil
  >>> print pkgutil.get_data('test', 'exception_hiera
  BaseException
   +-- SystemExit
   +-- KeyboardInterrupt
   +-- GeneratorExit
   +-- Exception
        +-- StopIteration
        +-- StandardError
   ...
  ```

  (Contribución de Paul Moore; bpo-2439 [https://bugs.python.org/issue?@action=redirect&bpo=2439].)

- Los objetos `pyexpat` del módulo `Parser` permiten ahora establecer su atributo `buffer_size` para cambiar el tamaño del buffer utilizado para mantener los datos de caracteres. (Contribución de Achim Gaedke; bpo-1137 [https://bugs.python.org/issue?@action=redirect&bpo=1137].)

- El módulo `Queue` proporciona ahora variantes de colas que recuperan las entradas en diferentes órdenes. La clase `PriorityQueue` almacena los elementos en cola en un montón y los recupera en orden de prioridad, y `LifoQueue` recupera primero las entradas añadidas más recientemente, lo que significa que se comporta como una pila. (Contribución de Raymond Hettinger)

- Los objetos **`Random`** del módulo **`random`** ahora pueden ser decapados en un sistema de 32 bits y desempaquetados en un sistema de 64 bits, y viceversa. Desafortunadamente, este cambio también significa que los objetos **`Random`** de Python 2.6 no pueden ser desempaquetados correctamente en versiones anteriores de Python. (Contribuido por Shawn Ligocki; bpo-1727780 [https://bugs.python.org/issue?@action=redirect&bpo=1727780].)

  La nueva función `triangular(low, high, mode)` devuelve números aleatorios siguiendo una distribución triangular. Los valores devueltos están entre *bajos* y *altos*, sin incluir los propios *altos*, y con *modo* como valor más frecuente en la distribución. (Contribución de Wladmir van der Laan y Raymond Hettinger; bpo-1681432 [https://bugs.python.org/issue?@action=redirect&bpo=1681432].)

- Las búsquedas de expresiones regulares largas llevadas a cabo por el módulo **`re`** comprobarán la entrega de señales, por lo que ahora se pueden interrumpir las búsquedas que consumen mucho tiempo. (Contribución de Josh Hoyt y Ralf Schmitt; bpo-846388 [https://bugs.python.org/issue?@action=redirect&bpo=846388].)

  El módulo de expresiones regulares se implementa compilando bytecodes para una pequeña máquina virtual específica de regex. El código no fiable podría crear cadenas maliciosas de código de bytes directamente y causar caídas, por lo que Python 2.6 incluye un verificador para el código de bytes regex. (Contribuido por Guido van Rossum del trabajo para Google App Engine; bpo-3487 [https://bugs.python.org/issue?@action=redirect&bpo=3487].)

- El método **`Completer.complete()`** del módulo **`rlcompleter`** ahora ignorará las excepciones activadas al evaluar un nombre. (Corregido por Lorenz Quack; bpo-2250 [https://bugs.python.org/issue?@action=redirect&bpo=2250]).

- Las instancias `sched` del módulo `scheduler` tienen ahora un atributo `queue` de sólo lectura que devuelve el contenido de la cola del planificador, representado como una lista de tuplas con nombre con los campos `(tiempo, prioridad, acción, argumento)`. (Contribución de Raymond Hettinger; bpo-1861 [https://bugs.python.org/issue?@action=redirect&bpo=1861].)

- El módulo `select` tiene ahora funciones de envoltura para las llamadas al sistema Linux `epoll()` y BSD `kqueue()`. Se ha añadido el método `modify()` a los objetos `poll` existentes; `pollobj.modify(fd, eventmask)` toma un descriptor de fichero o un objeto de fichero y una máscara de evento, modificando la máscara de evento registrada para ese fichero. (Contribución de Christian Heimes; bpo-1657 [https://bugs.python.org/issue?@action=redirect&bpo=1657].)

- La función `shutil.copytree()` tiene ahora un argumento opcional *ignore* que toma un objeto callable. Esta llamada recibirá cada ruta de directorio y una lista del contenido del directorio, y devuelve una lista de nombres que serán ignorados, no copiados.

  El módulo `shutil` también proporciona una función `ignore_patterns()` para su uso con este nuevo parámetro. `ignore_patterns()` toma un número arbitrario de patrones de estilo glob y devuelve una llamada que ignorará cualquier archivo o directorio que coincida con cualquiera de estos patrones. El siguiente ejemplo copia un árbol de directorios, pero omite los directorios `.svn` y los ficheros de copia de seguridad de Emacs, que tienen nombres que terminan en “~”:

  ```
  shutil.copytree('Doc/library', '/tmp/library',
                  ignore=shutil.ignore_patterns('*~',
  ```

  (Contribución de Tarek Ziadé; bpo-2663 [https://bugs.python.org/issue?@action=redirect&bpo=2663].)

- Integrar el manejo de señales con los bucles de eventos de manejo de la interfaz gráfica de usuario (GUI), como los utilizados por Tkinter o GTk+, ha sido durante mucho tiempo un problema; la mayoría del software acaba sondeando, despertando cada fracción de segundo para comprobar si se ha producido algún evento de la GUI. El módulo **`signal`** puede ahora hacer esto más eficiente. Llamando a `signal.set_wakeup_fd(fd)` se establece un descriptor de fichero a utilizar; cuando se recibe una señal, se escribe un byte en ese descriptor de fichero. También hay una función en C, **`PySignal_SetWakeupFd()`**, para establecer el descriptor.

Los bucles de eventos utilizarán esto abriendo una tubería para crear dos descriptores, uno de lectura y otro de escritura. El descriptor de escritura se pasará a **`set_wakeup_fd()`**, y el descriptor de lectura se añadirá a la lista de descriptores monitorizados por el bucle de eventos mediante **`select()`** o **`poll()`**. Al recibir una señal, se escribirá un byte y se despertará el bucle de eventos principal, evitando la necesidad de hacer un sondeo.

(Contribución de Adam Olsen; bpo-1583 [https://bugs.python.org/issue?@action=redirect&bpo=1583].)

La función **`siginterrupt()`** está ahora disponible desde el código Python, y permite cambiar si las señales pueden interrumpir las llamadas al sistema o no. (Contribución de Ralf Schmitt)

También se han añadido las funciones **`setitimer()`** y **`getitimer()`** (cuando están disponibles). **`setitimer()`** permite establecer temporizadores de intervalo que harán que se entregue una señal al proceso después de un tiempo especificado, medido en tiempo de reloj de pared, tiempo de proceso consumido, o tiempo combinado de proceso +sistema. (Contribuido por Guilherme Polo; bpo-2240 [https://bugs.python.org/issue?@action=redirect&bpo=2240].)

- El módulo **`smtplib`** ahora soporta SMTP sobre SSL gracias a la adición de la clase **`SMTP_SSL`**. Esta clase soporta una interfaz idéntica a la clase **`SMTP`** existente. (Contribuido por Monty Taylor.) Ambos constructores de la clase también tienen un parámetro opcional `timeout` que especifica un tiempo de espera para el intento de conexión inicial, medido en segundos. (Contribuido por Facundo Batista.)

También se ha añadido al módulo una implementación del protocolo LMTP (**RFC 2033** [https://datatracker.ietf.org/doc/html/rfc2033.html]). LMTP se utiliza en lugar de SMTP cuando se transfiere correo electrónico entre agentes que no gestionan una cola de correo. (LMTP implementado por Leif Hedstrom; bpo-957003 [https://bugs.python.org/issue?@action=redirect&bpo=957003].)

`SMTP.starttls()` ahora cumple con **RFC 3207** [https://datatracker.ietf.org/doc/html/rfc3207.html] y olvida cualquier conocimiento obtenido del servidor que no se haya obtenido de la propia negociación TLS. (Parche aportado por Bill Fenner; bpo-829951 [https://bugs.python.org/issue?@action=redirect&bpo=829951].)

- El módulo **`socket`** soporta ahora TIPC (http://tipc.sourceforge.net/), un protocolo de alto rendimiento no basado en IP y diseñado para su uso en entornos agrupados. Las direcciones TIPC son de 4 o 5 tuplas. (Contribución de Alberto Bertogli; bpo-1646 [https://bugs.python.org/issue?@action=redirect&bpo=1646].)

  Una nueva función, **`create_connection()`**, toma una dirección y se conecta a ella utilizando un valor opcional de tiempo de espera, devolviendo el objeto socket conectado. Esta función también busca el tipo de dirección y se conecta a ella usando IPv4 o IPv6 según sea el caso. Cambiar su código para usar **`create_connection()`** en lugar de `socket(socket.AF_INET, ...)` puede ser todo lo que se necesita para que su código funcione con IPv6.

- Las clases base del módulo **`SocketServer`** ahora admiten la llamada a un método **`handle_timeout()`** después de un período de inactividad especificado por el atributo **`timeout`** del servidor. (Contribuido por Michael Pomraning.) El método **`serve_forever()`** ahora utiliza un intervalo de sondeo opcional medido en segundos, que controla la frecuencia con la que el servidor comprobará si hay una solicitud de apagado. (Contribuido por Pedro Werneck y Jeffrey Yasskin; bpo-742598 [https://bugs.python.org/issue?@action=redirect&bpo=742598], bpo-1193577 [https://bugs.python.org/issue?@action=redirect&bpo=1193577].)

- El módulo `sqlite3`, mantenido por Gerhard Häring, ha sido actualizado de la versión 2.3.2 en Python 2.5 a la versión 2.4.1.

- El módulo `struct` ahora admite el tipo C99 _Bool, utilizando el carácter de formato `'?'`. (Aportado por David Remahl.)

- Los objetos `Popen` proporcionados por el módulo `subprocess` ahora tienen los métodos `terminate()`, `kill()` y `send_signal()`. En Windows, `send_signal()` solo admite la señal `SIGTERM` y todos estos métodos son alias de la función API Win32 `TerminateProcess()`. (Contribuido por Christian Heimes).

- Una nueva variable en el módulo `sys`, `float_info`, es un objeto que contiene información derivada del archivo `float.h` sobre el soporte de punto flotante de la plataforma. Los atributos de este objeto incluyen `mant_dig` (número de dígitos en la mantisa), `epsilon` (diferencia más pequeña entre 1.0 y el siguiente valor más grande representable), y varios otros. (Contribuido por Christian Heimes; bpo-1534 [https://bugs.python.org/issue?@action=redirect&bpo=1534].)

  Otra nueva variable, `dont_write_bytecode`, controla si Python escribe algún archivo `.pyc` o `.pyo` al importar un módulo. Si esta variable es verdadera, los archivos compilados no se escriben. La variable se establece inicialmente al iniciar el intérprete de Python con la opción `-B`, o estableciendo la variable de entorno `PYTHONDONTWRITEBYTECODE` antes de ejecutar el intérprete. El código de Python puede cambiar posteriormente el valor de esta variable para controlar si los archivos de código de bytes se escriben o no. (Contribución de Neal Norwitz y Georg Brandl)

La información sobre los argumentos de la línea de comandos suministrados al intérprete de Python está disponible leyendo los atributos de una tupla con nombre disponible como `sys.flags`. Por ejemplo, el atributo **`verbose`** es verdadero si Python fue ejecutado en modo verbose, **`debug`** es verdadero en modo debugging, etc. Todos estos atributos son de sólo lectura. (Contribución de Christian Heimes)

Una nueva función, **`getsizeof()`**, toma un objeto Python y devuelve la cantidad de memoria utilizada por el objeto, medida en bytes. Los objetos incorporados devuelven resultados correctos; las extensiones de terceros pueden no hacerlo, pero pueden definir un método **`__sizeof__()`** para devolver el tamaño del objeto. (Contribución de Robert Schuppenies; bpo-2898 [https://bugs.python.org/issue?@action=redirect&bpo=2898].)

Ahora es posible determinar las funciones actuales del perfilador y del trazador llamando a **`sys.getprofile()`** y **`sys.gettrace()`**. (Contribuido por Georg Brandl; bpo-1648 [https://bugs.python.org/issue?@action=redirect&bpo=1648].)

- El módulo **`tarfile`** soporta ahora archivos tar POSIX.1-2001 (pax) además de los formatos POSIX.1-1988 (ustar) y GNU tar que ya estaban soportados. El formato por defecto es GNU tar; especifique el parámetro `format` para abrir un archivo con un formato diferente:

  ```
  tar = tarfile.open("output.tar", "w",
                     format=tarfile.PAX_FORMAT)
  ```

  Los nuevos parámetros `encoding` y `errors` especifican una codificación y un esquema de manejo de errores para las conversiones de caracteres. `'strict'`, `'ignore'`, y `'replace'` son las tres formas estándar en las que Python puede manejar los errores; `'utf-8'` es un valor especial que reemplaza los caracteres erróneos con su representación UTF-8. (Las conversiones de caracteres ocurren porque el formato PAX soporta nombres de archivo Unicode, por defecto con codificación UTF-8)

El método **`TarFile.add()`** ahora acepta un argumento `exclude` que es una función que se puede usar para excluir ciertos nombres de archivo de un archivo. La función debe tomar un nombre de archivo y retornar verdadero si el archivo debe excluirse o falso si debe archivarse. La función se aplica tanto al nombre pasado inicialmente a **`add()`** como a los nombres de los archivos en los directorios agregados recursivamente.

(Todos los cambios han sido aportados por Lars Gustäbel).

- Se agregó un parámetro opcional `timeout` al constructor de la clase **`telnetlib.Telnet`**, que especifica un tiempo de espera medido en segundos. (Agregado por Facundo Batista).
- La clase **`tempfile.NamedTemporaryFile`** suele borrar el archivo temporal que ha creado cuando se cierra el archivo. Este comportamiento se puede cambiar pasando `delete=False` al constructor. (Contribución de Damien Miller; bpo-1537850 [https://bugs.python.org/issue?@action=redirect&bpo=1537850].)

  Una nueva clase, **`SpooledTemporaryFile`**, se comporta como un archivo temporal pero almacena sus datos en memoria hasta que se supera un tamaño máximo. Al llegar a ese límite, el contenido se escribirá en un archivo temporal en el disco. (Contribución de Dustin J. Mitchell)

  Las clases **`NamedTemporaryFile`** y **`SpooledTemporaryFile`** funcionan como gestores de contexto, por lo que puedes escribir `con tempfile.NamedTemporaryFile() como tmp: ...`. (Contribuido por Alexander Belopolsky; bpo-2021 [https://bugs.python.org/issue?@action=redirect&bpo=2021].)
- El módulo **`test.test_support`** ha incorporado una serie de administradores de contexto útiles para escribir pruebas. **`EnvironmentVarGuard()`** es un administrador de contexto que cambia temporalmente las variables de entorno y las restaura automáticamente a sus valores anteriores.

Otro gestor de contexto, **`TransientResource`**, puede rodear las llamadas a recursos que pueden o no estar disponibles; atrapará e ignorará una lista especificada de excepciones. Por ejemplo, una prueba de red puede ignorar ciertos fallos al conectarse a un sitio web externo:

```
with test_support.TransientResource(IOError,
                                errno=errno.ETIMEDO
    f = urllib.urlopen('https://sf.net')
    ...
```

Por último, **`check_warnings()`** restablece los filtros de advertencia del módulo **`warning`** y devuelve un objeto que registrará todos los mensajes de advertencia activados (bpo-3781 [https://bugs.python.org/issue?@action=redirect&bpo=3781]):

```
with test_support.check_warnings() as wrec:
    warnings.simplefilter("always")
    # ... code that triggers a warning ...
    assert str(wrec.message) == "function is outdat
    assert len(wrec.warnings) == 1, "Multiple warni
```

(Contribución de Brett Cannon.)

- El módulo **`textwrap`** ahora puede conservar los espacios en blanco existentes al principio y al final de las líneas recién creadas especificando `drop_whitespace=False` como argumento:

```
>>> S = """This  sentence  has a bunch   of
...   extra   whitespace."""
>>> print textwrap.fill(S, width=15)
This  sentence
has a bunch
of    extra
whitespace.
>>> print textwrap.fill(S, drop_whitespace=False, w
This  sentence
  has a bunch
```

```
    of    extra
    whitespace.
>>>
```

(Contribución de Dwayne Bailey; bpo-1581073 [https://bugs.python.org/issue?@action=redirect&bpo=1581073].)

- La API del módulo `threading` se ha cambiado para utilizar propiedades como **`daemon`** en lugar de los métodos **`setDaemon()`** y **`isDaemon()`**, y algunos métodos se han renombrado para utilizar guiones bajos en lugar de mayúsculas; por ejemplo, el método **`activeCount()`** se ha renombrado a **`active_count()`**. Tanto la versión 2.6 como la 3.0 del módulo soportan las mismas propiedades y métodos renombrados, pero no eliminan los métodos antiguos. No se ha fijado una fecha para la eliminación de las antiguas APIs en Python 3.x; las antiguas APIs no se eliminarán en ninguna versión 2.x. (Llevado a cabo por varias personas, sobre todo por Benjamin Peterson)

  Los objetos `threading` del módulo **`Thread`** han ganado una propiedad **`ident`** que devuelve el identificador del hilo, un entero no nulo. (Contribución de Gregory P. Smith; bpo-2871 [https://bugs.python.org/issue?@action=redirect&bpo=2871].)

- El módulo `timeit` acepta ahora callables así como cadenas para la sentencia que se está cronometrando y para el código de configuración. Se han añadido dos funciones para crear instancias de **`Timer`**: `repeat(stmt, setup, time, repeat, number)` y `timeit(stmt, setup, time, number)` crean una instancia y llaman al método correspondiente. (Contribución de Erik Demaine; bpo-1533909 [https://bugs.python.org/issue?@action=redirect&bpo=1533909].)

- El módulo **`Tkinter`** ahora acepta listas y tuplas para las opciones, separando los elementos por espacios antes de pasar el valor resultante a Tcl/Tk. (Contribuido por Guilherme Polo; bpo-2906 [https://bugs.python.org/issue?@action=redirect&bpo=2906].)

- El módulo `turtle` para los gráficos de tortugas ha sido muy mejorado por Gregor Lingl. Las nuevas características del módulo incluyen:

  - Mejor animación del movimiento y la rotación de la tortuga.
  - Control del movimiento de las tortugas mediante los nuevos métodos **`retraso()`**, **`tracer()`** y **`velocidad()`**.
  - La posibilidad de establecer nuevas formas para la tortuga y de definir un nuevo sistema de coordenadas.
  - Las tortugas ahora tienen un método **`undo()`** que puede revertir acciones.
  - Soporte sencillo para reaccionar a eventos de entrada como la actividad del ratón y del teclado, lo que permite escribir juegos sencillos.
  - Se puede utilizar un archivo `turtle.cfg` para personalizar la apariencia inicial de la pantalla de la tortuga.
  - Los docstrings del módulo pueden ser sustituidos por nuevos docstrings traducidos a otro idioma.

  (bpo-1513695 [https://bugs.python.org/issue?@action=redirect&bpo=1513695])

- Se agregó un parámetro opcional `timeout` a la función `urllib.urlopen` y al constructor de la clase **`urllib.ftpwrapper`**, así como a la función **`urllib2.urlopen`**. El parámetro especifica un tiempo de espera medido en segundos. Por ejemplo:

```
>>> u = urllib2.urlopen("http://slow.example.com",
                        timeout=3)
Traceback (most recent call last):
  ...
urllib2.URLError: <urlopen error timed out>
>>>
```

  (Añadido por Facundo Batista.)

- La base de datos Unicode proporcionada por el módulo `unicodedata` ha sido actualizada a la versión 5.1.0. (Actualizado por Martin von Löwis; bpo-3811 [https://bugs.python.org/issue?@action=redirect&bpo=3811].)

- Las funciones `warnings` del módulo **`formatwarning()`** y **`showwarning()`** han ganado un argumento opcional *line* que puede utilizarse para proporcionar la línea de código fuente. (Añadido como parte de bpo-1631171 [https://bugs.python.org/issue?@action=redirect&bpo=1631171], que reimplementaba parte del módulo **`warnings`** en código C)

  Una nueva función, **`catch_warnings()`**, es un gestor de contexto destinado a la realización de pruebas que permite modificar temporalmente los filtros de advertencia y luego restaurar sus valores originales (bpo-3781 [https://bugs.python.org/issue?@action=redirect&bpo=3781]).

- Ahora es posible evitar que las clases XML-RPC `SimpleXMLRPCServer` y **`DocXMLRPCServer`** se abran y enlacen inmediatamente con su socket al pasar `False` como parámetro del constructor *bind_and_activate*. Esto se puede utilizar para modificar el atributo **`allow_reuse_address`** de la instancia antes de llamar a los métodos **`server_bind()`** y **`server_activate()`** para abrir el socket y comenzar a escuchar conexiones. (Contribuido por Peter Parente; bpo-1599845 [https://bugs.python.org/issue?@action=redirect&bpo=1599845]).

  `SimpleXMLRPCServer` también tiene un atributo **`_send_traceback_header`**; si es true, la excepción y el traceback formateado se devuelven como cabeceras HTTP «X-Exception» y «X-Traceback». Esta característica es sólo para propósitos de depuración y no debe ser usada en servidores de producción porque los trazos pueden revelar contraseñas u otra información sensible. (Contribuido por Alan McIntyre como parte de su proyecto para el Summer of Code 2007 de Google)

- El módulo `xmlrpclib` ya no convierte automáticamente `datetime.date` y `datetime.time` al tipo `xmlrpclib.DateTime`; la semántica de conversión no era necesariamente correcta para todas las aplicaciones. El código que utiliza **`xmlrpclib`** debería convertir las instancias **`date`** y **`time`**. (bpo-1330538 [https://bugs.python.org/issue?@action=redirect&bpo=1330538]) El código también puede manejar fechas anteriores a 1900 (contribuido por Ralf Schmitt; bpo-2014 [https://bugs.python.org/issue?@action=redirect&bpo=2014]) y números enteros de 64 bits representados mediante el uso de `<i8>` en respuestas XML-RPC (contribuido por Riku Lindblad; bpo-2985 [https://bugs.python.org/issue?@action=redirect&bpo=2985]).

- La clase **`zipfile`** del módulo **`ZipFile`** tiene ahora los métodos **`extract()`** y **`extractall()`** que desempaquetarán un solo fichero o todos los ficheros del archivo en el directorio actual, o en un directorio especificado:

```
z = zipfile.ZipFile('python-251.zip')

# Unpack a single file, writing it relative
# to the /tmp directory.
z.extract('Python/sysmodule.c', '/tmp')

# Unpack all the files in the archive.
z.extractall()
```

(Contribución de Alan McIntyre; bpo-467924 [https://bugs.python.org/issue?@action=redirect&bpo=467924].)

Los métodos **`open()`**, **`read()`** y **`extract()`** ahora pueden tomar un nombre de archivo o un objeto **`ZipInfo`**. Esto es útil cuando un archivo contiene accidentalmente un nombre de archivo duplicado. (Contribución de Graham Horler; bpo-1775025 [https://bugs.python.org/issue?@action=redirect&bpo=1775025].)

Por último, **`zipfile`** soporta ahora el uso de nombres de archivo Unicode para los ficheros archivados. (Contribución de Alexey Borzenkov; bpo-1734346 [https://bugs.python.org/issue?@action=redirect&bpo=1734346].)

El módulo `ast`

El módulo `ast` proporciona una representación de árbol de sintaxis abstracta del código Python, y Armin Ronacher ha contribuido con un conjunto de funciones de ayuda que realizan una variedad de tareas comunes. Éstas serán útiles para paquetes de plantillas HTML, analizadores de código y herramientas similares que procesan código Python.

La función **`parse()`** toma una expresión y devuelve un AST. La función **`dump()`** devuelve una representación de un árbol, adecuada para la depuración:

```
import ast

t = ast.parse("""
d = {}
for i in 'abcdefghijklm':
    d[i + i] = ord(i) - ord('a') + 1
print d
""")
print ast.dump(t)
```

Esto produce un árbol profundamente anidado:

```
Module(body=[
  Assign(targets=[
    Name(id='d', ctx=Store())
   ], value=Dict(keys=[], values=[]))
  For(target=Name(id='i', ctx=Store()),
      iter=Str(s='abcdefghijklm'), body=[
    Assign(targets=[
      Subscript(value=
        Name(id='d', ctx=Load()),
          slice=
          Index(value=
            BinOp(left=Name(id='i', ctx=Load()), op=Add
             right=Name(id='i', ctx=Load()))), ctx=Store
     ], value=
      BinOp(left=
       BinOp(left=
        Call(func=
         Name(id='ord', ctx=Load()), args=[
           Name(id='i', ctx=Load())
          ], keywords=[], starargs=None, kwargs=None),
        op=Sub(), right=Call(func=
         Name(id='ord', ctx=Load()), args=[
           Str(s='a')
          ], keywords=[], starargs=None, kwargs=None)),
        op=Add(), right=Num(n=1)))
     ], orelse=[])
    Print(dest=None, values=[
      Name(id='d', ctx=Load())
    ], nl=True)
  ])
```

El método **`literal_eval()`** toma una cadena o un AST que representa una expresión literal, la analiza y evalúa, y devuelve el valor resultante. Una expresión literal es una expresión de Python que sólo contiene cadenas, números, diccionarios, etc. pero no declaraciones o llamadas a funciones. Si necesita evaluar una expresión pero no puede aceptar el riesgo de seguridad que supone utilizar una llamada a **`eval()`**, **`literal_eval()`** lo hará de forma segura:

```
>>> literal = '("a", "b", {2:4, 3:8, 1:2})'
>>> print ast.literal_eval(literal)
('a', 'b', {1: 2, 2: 4, 3: 8})
>>> print ast.literal_eval('"a" + "b"')
Traceback (most recent call last):
  ...
ValueError: malformed string
```

El módulo también incluye las clases **`NodeVisitor`** y **`NodeTransformer`** para recorrer y modificar un AST, y funciones para transformaciones comunes como el cambio de números de línea.

The `future_builtins` module

Python 3.0 makes many changes to the repertoire of built-in functions, and most of the changes can't be introduced in the Python 2.x series because they would break compatibility. The **`future_builtins`** module provides versions of these built-in functions that can be imported when writing 3.0-compatible code.

Las funciones de este módulo incluyen actualmente:

- `ascii(obj)`: equivalente a **`repr()`**. En Python 3.0, **`repr()`** devolverá una cadena Unicode, mientras que **`ascii()`** devolverá una cadena de bytes ASCII pura.
- `filtro(predicado, iterable)`, `map(func, iterable1, ...)`: las versiones 3.0 devuelven iteradores, a diferencia de las funciones integradas 2.x que devuelven listas.
- `hex(valor)`, `oct(valor)`: en lugar de llamar a los métodos **`__hex__()`** o **`__oct__()`**, estas versiones llamarán al método **`__index__()`** y convertirán el resultado a hexadecimal u octal. **`oct()`** utilizará la nueva notación `0o` para su resultado.

El módulo `json`: Notación de objetos de JavaScript

El nuevo módulo **`json`** soporta la codificación y decodificación de tipos Python en JSON (*Javascript Object Notation*). JSON es un formato de intercambio ligero que se utiliza a menudo en las aplicaciones web. Para más información sobre JSON, consulte http://www.json.org.

`json` viene con soporte para decodificar y codificar la mayoría de los tipos incorporados de Python. El siguiente ejemplo codifica y decodifica un diccionario:

```
>>> import json
>>> data = {"spam": "foo", "parrot": 42}
>>> in_json = json.dumps(data) # Encode the data
>>> in_json
'{"parrot": 42, "spam": "foo"}'
```

```
>>> json.loads(in_json) # Decode into a Python object
{"spam": "foo", "parrot": 42}
```

También es posible escribir tus propios decodificadores y codificadores para soportar más tipos. También se admite la impresión bonita de las cadenas JSON.

`json` (originalmente llamado simplejson) fue escrito por Bob Ippolito.

El módulo `plistlib`: Un analizador de listas de propiedades

El formato `.plist` se utiliza habitualmente en Mac OS X para almacenar tipos de datos básicos (números, cadenas, listas y diccionarios) serializándolos en un formato basado en XML. Se asemeja a la serialización XML-RPC de los tipos de datos.

A pesar de ser utilizado principalmente en Mac OS X, el formato no tiene nada de específico para Mac y la implementación de Python funciona en cualquier plataforma que soporte Python, por lo que el módulo `plistlib` ha sido promovido a la biblioteca estándar.

El uso del módulo es sencillo:

```
import sys
import plistlib
import datetime as dt

# Create data structure
data_struct = dict(lastAccessed=dt.datetime.now(),
                   version=1,
                   categories=('Personal','Shared','Priv

# Create string containing XML.
plist_str = plistlib.writePlistToString(data_struct)
new_struct = plistlib.readPlistFromString(plist_str)
print data_struct
print new_struct
```

```
# Write data structure to a file and read it back.
plistlib.writePlist(data_struct, '/tmp/customizations.pl
new_struct = plistlib.readPlist('/tmp/customizations.pli

# read/writePlist accepts file-like objects as well as p
plistlib.writePlist(data_struct, sys.stdout)
```

mejoras en ctypes

Thomas Heller siguió manteniendo y mejorando el módulo `ctypes`.

`ctypes` soporta ahora un tipo de datos **`c_bool`** que representa el tipo C99 `bool`. (Contribución de David Remahl; bpo-1649190 [https://bugs.python.org/issue?@action=redirect&bpo=1649190].)

Los tipos de cadena, buffer y array de `ctypes` han mejorado el soporte para la sintaxis de corte extendida, donde se suministran varias combinaciones de `(start, stop, step)`. (Implementado por Thomas Wouters)

Todos los tipos de datos `ctypes` soportan ahora los métodos **`from_buffer()`** y **`from_buffer_copy()`** que crean una instancia de ctypes basada en un objeto buffer proporcionado. **`from_buffer_copy()`** copia el contenido del objeto, mientras que **`from_buffer()`** compartirá la misma área de memoria.

Una nueva convención de llamada indica a `ctypes` que borre las variables `errno` o Win32 LastError al inicio de cada llamada envuelta. (Implementado por Thomas Heller; bpo-1798 [https://bugs.python.org/issue?@action=redirect&bpo=1798].)

Ahora puede recuperar la variable Unix `errno` después de una llamada a una función. Al crear una función envuelta, puede suministrar `use_errno=True` como parámetro de palabra clave a la función **`DLL()`** y luego llamar a los métodos de nivel de módulo **`set_errno()`** y **`get_errno()`** para establecer y recuperar el valor del error.

La variable Win32 LastError es soportada de forma similar por las funciones **`DLL()`**, **`OleDLL()`**, y **`WinDLL()`**. Se suministra `use_last_error=True` como parámetro de palabra clave y luego se llaman los métodos a nivel de módulo **`set_last_error()`** y **`get_last_error()`**.

La función **`byref()`**, utilizada para recuperar un puntero a una instancia de ctypes, tiene ahora un parámetro opcional *offset* que es un recuento de bytes que se añadirá al puntero devuelto.

Mejora de la compatibilidad con SSL

Bill Janssen ha realizado amplias mejoras en el soporte de Python 2.6 para la Capa de Conexión Segura añadiendo un nuevo módulo, **`ssl`**, que está construido sobre la biblioteca OpenSSL [https://www.openssl.org/]. Este nuevo módulo proporciona más control sobre el protocolo negociado, los certificados X.509 utilizados, y tiene un mejor soporte para escribir servidores SSL (a diferencia de los clientes) en Python. El soporte SSL existente en el módulo **`socket`** no ha sido eliminado y sigue funcionando, aunque será eliminado en Python 3.0.

Para utilizar el nuevo módulo, primero hay que crear una conexión TCP de la forma habitual y luego pasarla a la función **`ssl.wrap_socket()`**. Es posible especificar si se requiere un certificado, y obtener información sobre el mismo llamando al método **`getpeercert()`**.

Ver también

La documentación del módulo **`ssl`**.

Cancelaciones y eliminaciones

- Se han eliminado las excepciones de cadena. Al intentar utilizarlas se produce un **`TypeError`**.
- Se siguen realizando cambios en la interfaz **`Exception`** según lo dictado por **PEP 352** [https://peps.python.org/pep-0352/]. En la versión 2.6, el atributo **`message`** se está reemplazando por el atributo **`args`**.

- (modo 3.0-warning) Python 3.0 presentará una biblioteca estándar reorganizada que eliminará muchos módulos obsoletos y renombrará otros. Python 2.6 funcionando en modo 3.0-warning avisará de estos módulos cuando se importen.

  La lista de módulos obsoletos es: **`audiodev`**, **`bgenlocations`**, **`buildtools`**, **`bundlebuilder`**, **`Canvas`**, **`compiler`**, **`dircache`**, **`dl`**, **`fpformat`**, **`gensuitemodule`**, **`ihooks`**, **`imageop`**, **`imgfile`**, **`linuxaudiodev`**, **`mhlib`**, **`mimetools`**, **`multifile`**, **`new`**, **`pure`**, **`statvfs`**, **`sunaudiodev`**, **`test.testall`** y **`toaiff`**.

- El módulo **`gopherlib`** ha sido eliminado.

- El módulo **`MimeWriter`** y el módulo **`mimify`** han quedado obsoletos; utilice en su lugar el paquete **`email`**.

- El módulo **`md5`** ha quedado obsoleto; utilice en su lugar el módulo **`hashlib`**.

- El módulo **`posixfile`** ha quedado obsoleto; **`fcntl.lockf()`** proporciona un mejor bloqueo.

- El módulo **`popen2`** ha quedado obsoleto; utilice el módulo **`subprocess`**.

- Se ha eliminado el módulo **`rgbimg`**.

- El módulo **`sets`** ha quedado obsoleto; es mejor utilizar los tipos incorporados **`set`** y **`frozenset`**.

- El módulo **`sha`** ha quedado obsoleto; utilice en su lugar el módulo **`hashlib`**.

Cambios en la API de construcción y C

Los cambios en el proceso de construcción de Python y en la API de C incluyen:

- Python ahora debe ser compilado con compiladores C89 (¡después de 19 años!). Esto significa que el árbol de fuentes de Python ha abandonado sus propias implementaciones de **`memmove()`** y **`strerror()`**, que están en la biblioteca estándar de C89.

- Python 2.6 puede ser construido con Microsoft Visual Studio 2008 (versión 9.0), y este es el nuevo compilador por defecto. Ver el directorio `PCbuild` para los archivos de construcción. (Implementado por Christian Heimes)

- En Mac OS X, Python 2.6 puede ser compilado como una construcción universal de 4 vías. El script **configure** puede tomar una opción *:!–with-universal-archs=[32-bit|64-bit|all]*, controlando si los binarios se construyen para arquitecturas de 32 bits (x86, PowerPC), 64 bits (x86-64 y PPC-64), o ambas. (Contribución de Ronald Oussoren)

- Una nueva función agregada en Python 2.6.6, **`PySys_SetArgvEx()`**, establece el valor de `sys.argv` y opcionalmente puede actualizar `sys.path` para incluir el directorio que contiene el script nombrado por `sys.argv[0]` dependiendo del valor de un parámetro *updatepath*.

  Esta función se agregó para cerrar un agujero de seguridad en las aplicaciones que incorporan Python. La función anterior, **`PySys_SetArgv()`**, siempre actualizaba `sys.path` y, a veces, añadía el directorio actual. Esto significaba que, si ejecutaba una aplicación que incorporaba Python en un directorio controlado por otra persona, los atacantes podían colocar un módulo troyano en el directorio (por ejemplo, un archivo llamado `os.py`) que su aplicación luego importaría y ejecutaría.

  Si mantiene una aplicación C/C++ que incorpora Python, verifique si está llamando a **`PySys_SetArgv()`** y considere cuidadosamente si la aplicación debería usar **`PySys_SetArgvEx()`** con *updatepath* configurado como falso. Tenga en cuenta que el uso de esta función romperá la compatibilidad con las versiones 2.6.5 y anteriores de Python;

si tiene que continuar trabajando con versiones anteriores, puede dejar la llamada a **`PySys_SetArgv()`** y llamar a `PyRun_SimpleString("sys.path.pop(0)\n")` después para descartar el primer componente `sys.path`.

Problema de seguridad informado como **CVE 2008-5983** [https://www.cve.org/CVERecord?id=CVE-2008-5983]; discutido en gh-50003 [https://github.com/python/cpython/issues/50003] y solucionado por Antoine Pitrou.

- El módulo BerkeleyDB tiene ahora un objeto C API, disponible como `bsddb.db.api`. Este objeto puede ser utilizado por otras extensiones de C que deseen utilizar el módulo **`bsddb`** para sus propios fines. (Contribución de Duncan Grisby)

- La nueva interfaz del buffer, descrita previamente en la sección PEP 3118, añade **`PyObject_GetBuffer()`** y **`PyBuffer_Release()`**, así como algunas otras funciones.

- El uso que hace Python de la biblioteca stdio de C es ahora seguro para los hilos, o al menos tan seguro como la biblioteca subyacente. Un error potencial de larga data ocurría si un hilo cerraba un objeto de archivo mientras otro hilo estaba leyendo o escribiendo en el objeto. En la versión 2.6 los objetos archivo tienen un contador de referencias, manipulado por las funciones **`PyFile_IncUseCount()`** y **`PyFile_DecUseCount()`**. Los objetos de archivo no pueden cerrarse a menos que el recuento de referencias sea cero. **`PyFile_IncUseCount()`** debe llamarse mientras se mantiene el GIL, antes de realizar una operación de E/S utilizando el puntero `FILE *`, y **`PyFile_DecUseCount()`** debe llamarse inmediatamente después de recuperar el GIL. (Contribución de Antoine Pitrou y Gregory P. Smith)

- Importar módulos simultáneamente en dos hilos diferentes ya no se bloquea; ahora lanzará un **`ImportError`**. Una nueva función de la API, **`PyImport_ImportModuleNoBlock()`**, buscará un módulo en `sys.modules` primero, y luego intentará importarlo después de adquirir un bloqueo de importación. Si el bloqueo de importación está en manos de otro hilo, se genera un **`ImportError`**. (Contribuido por Christian Heimes.)

- Varias funciones devuelven información sobre la compatibilidad de punto flotante de la plataforma. **`PyFloat_GetMax()`** devuelve el valor máximo representable de punto flotante y **`PyFloat_GetMin()`** devuelve el valor positivo mínimo. **`PyFloat_GetInfo()`** devuelve un objeto que contiene más información del archivo `float.h`, como `"mant_dig"` (número de dígitos en la mantisa), `"epsilon"` (diferencia más pequeña entre 1.0 y el siguiente valor más grande representable) y varios otros. (Contribuido por Christian Heimes; bpo-1534 [https://bugs.python.org/issue?@action=redirect&bpo=1534].)

- Las funciones y métodos de C que utilizan **`PyComplex_AsCComplex()`** ahora aceptarán argumentos que tengan un método **`__complex__()`**. En particular, las funciones del módulo **`cmath`** ahora aceptarán objetos con este método. Este es un retroceso de un cambio en Python 3.0. (Contribuido por Mark Dickinson; bpo-1675423 [https://bugs.python.org/issue?@action=redirect&bpo=1675423].)

- La API de C de Python incluye ahora dos funciones para comparar cadenas sin distinción de mayúsculas y minúsculas, `PyOS_stricmp(char*, char*)` y `PyOS_strnicmp(char*, char*, Py_ssize_t)`. (Contribución de Christian Heimes; bpo-1635 [https://bugs.python.org/issue?@action=redirect&bpo=1635].)

- Muchas extensiones de C definen su propia macro para añadir enteros y cadenas al diccionario del módulo en la función `init*`. Python 2.6 finalmente define macros estándar para añadir valores a un módulo, **`PyModule_AddStringMacro`** y **`PyModule_AddIntMacro()`**. (Contribución de Christian Heimes.)

- Algunas macros han sido renombradas tanto en la 3.0 como en la 2.6 para dejar más claro que son macros y no funciones. **`Py_Size()`** se convierte en **`Py_SIZE()`**, **`Py_Type()`** se convierte en **`Py_TYPE()`**, y **`Py_Refcnt()`** se convierte en

`Py_REFCNT()`. Las macros de mayúsculas y minúsculas siguen estando disponibles en Python 2.6 por compatibilidad con versiones anteriores. (bpo-1629 [https://bugs.python.org/issue?@action=redirect&bpo=1629])

- Distutils ahora coloca las extensiones C que construye en un directorio diferente cuando se ejecuta en una versión de depuración de Python. (Contribución de Collin Winter; bpo-1530959 [https://bugs.python.org/issue?@action=redirect&bpo=1530959].)

- Varios tipos de datos básicos, como los enteros y las cadenas, mantienen listas internas de objetos libres que pueden reutilizarse. Las estructuras de datos para estas listas libres siguen ahora una convención de nomenclatura: la variable se llama siempre `free_list`, el contador se llama siempre `numfree`, y siempre se define una macro `Py<typename>_MAXFREELIST`

- Un nuevo objetivo de Makefile, «make patchcheck», prepara el árbol de fuentes de Python para hacer un parche: corrige los espacios en blanco al final de todos los archivos `.py` modificados, comprueba si la documentación ha sido cambiada, e informa si los archivos `Misc/ACKS` y `Misc/NEWS` han sido actualizados. (Contribuido por Brett Cannon.)

  Otro nuevo objetivo, «make profile-opt», compila un binario de Python utilizando la optimización guiada por perfiles de GCC. Compila Python con el perfil habilitado, ejecuta el conjunto de pruebas para obtener un conjunto de resultados de perfil, y luego compila usando estos resultados para la optimización. (Contribución de Gregory P. Smith)

Cambios específicos en los puertos: Windows

- Se ha eliminado el soporte para Windows 95, 98, ME y NT4. Python 2.6 requiere al menos Windows 2000 SP4.

- El nuevo compilador por defecto en Windows es Visual Studio 2008 (versión 9.0). Los directorios de compilación para Visual Studio 2003 (versión 7.1) y 2005 (versión 8.0) se han trasladado al directorio PC/. El nuevo directorio `PCbuild` admite la compilación cruzada para X64, las compilaciones de depuración y la optimización guiada por perfil (PGO). Las compilaciones PGO son aproximadamente un 10% f más rápidas que las normales. (Contribuido por Christian Heimes con la ayuda de Amaury Forgeot d'Arc y Martin von Löwis)

- El módulo `msvcrt` ahora admite las variantes de caracteres normales y anchos de la API de E/S de la consola. La función `getwch()` lee una pulsación de tecla y devuelve un valor Unicode, al igual que la función `getwche()`. La función `putwch()` toma un carácter Unicode y lo escribe en la consola. (Contribuido por Christian Heimes).

- `os.path.expandvars()` ahora expandirá las variables de entorno de la forma «%var%», y «~user» se expandirá en la ruta del directorio personal del usuario. (Contribución de Josiah Carlson; bpo-957650 [https://bugs.python.org/issue?@action=redirect&bpo=957650].)

- Los objetos de socket del módulo `socket` ahora tienen un método `ioctl()` que proporciona una interfaz limitada a la interfaz del sistema **`WSAIoctl()`**.

- El módulo `_winreg` ahora tiene una función, `ExpandEnvironmentStrings()`, que expande las referencias de variables de entorno como `%NAME%` en una cadena de entrada. Los objetos de control proporcionados por este módulo ahora admiten el protocolo de contexto, por lo que se pueden utilizar en instrucciones `with`. (Contribuido por Christian Heimes).

  `_winreg` también tiene un mejor soporte para sistemas x64, exponiendo las funciones `DisableReflectionKey()`, `EnableReflectionKey()` y `QueryReflectionKey()`, que habilitan y deshabilitan la reflexión del registro para procesos de 32 bits que se ejecutan en sistemas de 64 bits. (bpo-1753245 [https://bugs.python.org/issue?@action=redirect&bpo=1753245])

- El objeto **`Record`** del módulo **`msilib`** obtuvo los métodos

`GetInteger()` y `GetString()` que devuelven valores de campo como un entero o una cadena. (Contribuido por Floris Bruynooghe; bpo-2125 [https://bugs.python.org/issue?@action=redirect&bpo=2125]).

Cambios específicos en los puertos: Mac OS X

- Cuando se compila una estructura de Python, ahora se puede especificar el nombre de la estructura que se utilizará proporcionando la opción `--with-framework-name=` al script **configure**.
- Se ha eliminado el módulo `macfs`. Esto a su vez requirió que se eliminara la función `macostools.touched()` porque dependía del módulo `macfs`. (bpo-1490190 [https://bugs.python.org/issue?@action=redirect&bpo=1490190])
- Muchos otros módulos de Mac OS han quedado obsoletos y se eliminarán en Python 3.0: `_builtinSuites`, `aepack`, `aetools`, `aetypes`, `applesingle`, `appletrawmain`, `appletrunner`, `argvemulator`, `Audio_mac`, `autoGIL`, `Carbon`, `cfmfile`, `CodeWarrior`, `ColorPicker`, `EasyDialogs`, `Explorer`, `Finder`, `FrameWork`, `findertools`, `ic`, `icglue`, `icopen`, `macerrors`, `MacOS`, `macfs`, `macostools`, `macresource`, `MiniAEFrame`, `Nav`, `Netscape`, `OSATerminology`, `pimp`, `PixMapWrapper`, `StdSuites`, `SystemEvents`, `Terminal` y `terminalcommand`.

Cambios específicos en los puertos: IRIX

Varios módulos antiguos específicos de IRIX quedaron obsoletos y se eliminarán en Python 3.0: `al` y `AL`, `cd`, `cddb`, `cdplayer`, `CL` y `cl`, `DEVICE`, `ERRNO`, `FILE`, `FL` y `fl`, `flp`, `fm`, `GET`, `GLWS`, `GL` y `gl`, `IN`, `IOCTL`, `jpeg`, `panelparser`, `readcd`, `SV` y `sv`, `torgb`, `videoreader` y `WAIT`.

Adaptación a Python 2.6

Esta sección enumera los cambios descritos anteriormente y otras correcciones de errores que pueden requerir cambios en su código:

- Las clases que no se supone que son hashable deben establecer `__hash__ = None` en sus definiciones para indicar el hecho.

- Se han eliminado las excepciones de cadena. Al intentar utilizarlas se produce un `TypeError`.

- El método `__init__()` de `collections.deque` ahora borra cualquier contenido existente del deque antes de añadir elementos del iterable. Este cambio hace que el comportamiento coincida con `list.__init__()`.

- `object.__init__()` anteriormente aceptaba argumentos arbitrarios y argumentos de palabras clave, ignorándolos. En Python 2.6, esto ya no está permitido y resultará en un `TypeError`. Esto afectará a los métodos `__init__()` que acaben llamando al método correspondiente en `object` (quizás mediante el uso de `super()`). Ver bpo-1683368 [https://bugs.python.org/issue?@action=redirect&bpo=1683368] para su discusión.

- El constructor `Decimal` ahora acepta los espacios en blanco iniciales y finales cuando se le pasa una cadena. Antes lanzaba una excepción `InvalidOperation`. Por otro lado, el método `create_decimal()` de los objetos `Context` ahora desestima explícitamente los espacios en blanco adicionales, lanzando una excepción `ConversionSyntax`.

- Debido a un accidente de implementación, si se pasaba una ruta de archivo a la función incorporada `__import__()`, ésta importaría realmente el archivo especificado. Sin embargo, nunca se pretendió que esto funcionara, y la implementación ahora comprueba explícitamente este caso y lanza un `ImportError`.

- API C: las funciones `PyImport_Import()` y `PyImport_ImportModule()` ahora se ajustan por defecto a importaciones absolutas, no a importaciones relativas. Esto afectará a las extensiones de C que importen otros módulos.

- API C: los tipos de datos de extensión que no deben ser hashable deben definir su ranura `tp_hash` a `PyObject_HashNotImplemented()`.

- La excepción del módulo `socket` `socket.error` ahora hereda de `IOError`. Anteriormente no era una subclase de `StandardError` pero ahora lo es, a través de `IOError`. (Implementado por Gregory P. Smith; bpo-1706815 [https://bugs.python.org/issue?@action=redirect&bpo=1706815].)

- El módulo `xmlrpclib` ya no convierte automáticamente `datetime.date` y `datetime.time` al tipo `xmlrpclib.DateTime`; la semántica de conversión no era necesariamente correcta para todas las aplicaciones. El código que utiliza `xmlrpclib` debería convertir las instancias `date` y `time`. (bpo-1330538 [https://bugs.python.org/issue?@action=redirect&bpo=1330538])

- (Modo de advertencia 3.0) La clase `Exception` ahora advierte cuando se accede utilizando el acceso a la rebanada o al índice; tener `Exception` se comporta como una tupla está siendo eliminado.

- (modo de advertencia 3.0) las comparaciones de desigualdad entre dos diccionarios o dos objetos que no implementan métodos de comparación se reportan como advertencias. `dict1 == dict2` sigue funcionando, pero `dict1 < dict2` está siendo eliminado.

  Las comparaciones entre celdas, que son un detalle de implementación de las reglas de alcance de Python, también causan advertencias porque tales comparaciones están prohibidas por completo en 3.0.

Para aplicaciones que incorporan Python:

- La función `PySys_SetArgvEx()` se agregó en Python 2.6.6, lo que permite que las aplicaciones cierren un agujero de seguridad cuando se usa la función `PySys_SetArgv()` existente. Verifique si está llamando a `PySys_SetArgv()` y considere cuidadosamente si la aplicación debería usar `PySys_SetArgvEx()` con *updatepath* configurado como falso.

Agradecimientos

El autor desea agradecer a las siguientes personas sus sugerencias, correcciones y ayuda en varios borradores de este artículo: Georg Brandl, Steve Brown, Nick Coghlan, Ralph Corderoy, Jim Jewett, Kent Johnson, Chris Lambacher, Martin Michlmayr, Antoine Pitrou, Brian Warner.

Qué hay de nuevo en Python 2.5

Autor:

A.M. Kuchling

Este artículo explica las nuevas características de Python 2.5. La versión final de Python 2.5 está prevista para agosto de 2006; **PEP 356** [https://peps.python.org/pep-0356/] describe el cronograma de lanzamiento planificado.

Los cambios en Python 2.5 son una interesante mezcla de mejoras en el lenguaje y en las bibliotecas. Las mejoras en las bibliotecas serán más importantes para la comunidad de usuarios de Python, creo, debido a que se han añadido varios paquetes bastante útiles. Los nuevos módulos incluyen ElementTree para el procesamiento de XML (`xml.etree`), el módulo de base de datos SQLite (`sqlite`), y el módulo `ctypes` para llamar a funciones escritas en lenguaje C.

Los cambios en el lenguaje son de mediana importancia. Se han añadido algunas características nuevas y agradables, pero la mayoría de ellas no son características que vaya a utilizar todos los días. Las expresiones condicionales fueron finalmente añadidas al lenguaje usando una nueva sintaxis; ver sección PEP 308: Expresiones condicionales. La nueva sentencia "`with`" facilitará la escritura de código de limpieza (sección PEP 343: La declaración «con). Ahora se pueden pasar valores a los generadores (sección PEP 342: Nuevas funciones del generador). Las importaciones son ahora visibles como absolutas o relativas (sección PEP 328: Importaciones absolutas y relativas). Se han mejorado algunos casos de manejo de excepciones (sección PEP 341: Try/except/finally unificados). Todas estas mejoras merecen la pena, pero son mejoras de una u otra característica específica del lenguaje; ninguna de ellas es una modificación amplia de la semántica de Python.

Además de las adiciones al lenguaje y a la biblioteca, se han realizado otras mejoras y correcciones de errores en todo el árbol de código fuente. Una búsqueda en los registros de cambios del SVN revela que se aplicaron 353 parches y se corrigieron 458 errores entre Python 2.4 y 2.5. (Es probable que ambas cifras estén subestimadas)

Este artículo no pretende ser una especificación completa de las nuevas características; en su lugar, los cambios se introducen brevemente utilizando ejemplos útiles. Para obtener todos los detalles, siempre debes consultar la documentación de Python 2.5 en https://docs.python.org. Si quieres entender la implementación completa y los fundamentos del diseño, consulta el PEP de una nueva característica en particular.

Son bienvenidos los comentarios, las sugerencias y los informes de errores para este documento; por favor, envíelos por correo electrónico al autor o abra un error en el rastreador de errores de Python.

PEP 308: Expresiones condicionales

Durante mucho tiempo, la gente ha solicitado una forma de escribir expresiones condicionales, que son expresiones que devuelven el valor A o el valor B dependiendo de si un valor booleano es verdadero o falso. Una expresión condicional le permite escribir una única sentencia de asignación que tiene el mismo efecto que la siguiente:

```
Si la condición es:
x = valor_verdadero
De lo contrario:
x = valor_falso
```

Ha habido interminables y tediosas discusiones sobre la sintaxis tanto en python-dev como en comp.lang.python. Incluso se llevó a cabo una votación en la que se descubrió que la mayoría de los votantes querían expresiones condicionales de alguna forma, pero no había ninguna sintaxis que fuera preferida por una clara mayoría. Los candidatos incluían `cond ? true_v : false_v`, `if cond then true_v else false_v`, y otras 16 variaciones.

Guido van Rossum eligió finalmente una sintaxis sorprendente:

```
x = valor_verdadero si condición de lo contrario valor_f
```

La evaluación sigue siendo perezosa como en las expresiones booleanas existentes, por lo que el orden de evaluación salta un poco. La expresión *condición* del medio se evalúa primero, y la expresión *valor_verdadero* se evalúa sólo si la condición es verdadera. Del mismo modo, la expresión *valor_falso* sólo se evalúa cuando la condición es falsa.

Esta sintaxis puede parecer extraña y retrógrada; ¿por qué la condición va en el *medio* de la expresión, y no en la parte delantera como en `c ? x : y` de C? La decisión se comprobó aplicando la nueva sintaxis a los módulos de la biblioteca estándar y viendo cómo se leía el código resultante. En muchos casos en los que se utiliza una expresión condicional, un valor parece ser el «caso común» y otro valor es un «caso excepcional», utilizado sólo en las raras ocasiones en las que no se cumple la condición. La sintaxis condicional hace que este patrón sea un poco más obvio:

```
contenido = ((doc + '\n') si doc si no '')
```

Leo la afirmación anterior en el sentido de que «aquí se asigna a *contents* un valor de `doc+'\n'`; a veces *doc* está vacío, en cuyo caso especial se devuelve una cadena vacía» Dudo que use expresiones condicionales muy a menudo donde no hay un caso común y no común claro.

Hubo alguna discusión sobre si el lenguaje debería requerir rodear las expresiones condicionales con paréntesis. Se tomó la decisión de *no* requerir paréntesis en la gramática del lenguaje Python, pero como una cuestión de estilo creo que siempre deberías usarlos. Considere estas dos declaraciones:

```
# Primera versión: sin paréntesis
nivel = 1 si se registra, de lo contrario, 0

# Segunda versión: con paréntesis
```

```
nivel = (1 si se registra, de lo contrario, 0)
```

En la primera versión, creo que el ojo de un lector podría agrupar la sentencia en "nivel = 1", "si registro", "si no 0", y pensar que la condición decide si se realiza la asignación a *nivel*. La segunda versión se lee mejor, en mi opinión, porque deja claro que la asignación se realiza siempre y que se está eligiendo entre dos valores.

Otra razón para incluir los paréntesis: algunas combinaciones extrañas de comprensiones de listas y lambdas podrían parecer expresiones condicionales incorrectas. Véase PEP 308 [https://peps.python.org/pep-0308/] para algunos ejemplos. Si pone paréntesis alrededor de sus expresiones condicionales, no se encontrará con este caso.

Ver también

PEP 308 [https://peps.python.org/pep-0308/] - Expresiones condicionales

PEP escrito por Guido van Rossum y Raymond D. Hettinger; implementado por Thomas Wouters.

PEP 309: Aplicación parcial de funciones

El módulo `functools` está destinado a contener herramientas para la programación de estilo funcional.

Una herramienta útil de este módulo es la función **`partial()`**. Para los programas escritos en un estilo funcional, a veces querrá construir variantes de funciones existentes que tengan algunos de los parámetros rellenados. Considere una función Python `f(a, b, c)`; podría crear una nueva función `g(b, c)` que fuera equivalente a `f(1, b, c)`. Esto se llama «aplicación parcial de funciones».

`parcial()` toma los argumentos `(function, arg1, arg2, ... kwarg1=valor1, kwarg2=valor2)`. El objeto resultante es invocable, por lo que puedes llamarlo para invocar la *función* con los argumentos rellenados.

He aquí un pequeño pero realista ejemplo:

```
import functools

def log (message, subsystem):
"Escribe el contenido de 'message' en el subsistema espe
print '%s: %s' % (subsystem, message)
...

server_log = functools.partial(log, subsystem='server')
server_log('No se puede abrir el socket')
```

Aquí hay otro ejemplo, de un programa que utiliza PyGTK. Aquí se está construyendo dinámicamente un menú emergente sensible al contexto. El callback proporcionado para la opción de menú es una versión parcialmente aplicada del método **`open_item()`**, donde se ha proporcionado el primer argumento

```
...
class Application:
    def open_item(self, path):
       ...
    def init (self):
        open_func = functools.partial(self.open_item, it
        popup_menu.append( ("Open", open_func, 1) )
```

Otra función del módulo **`functools`** es la función `update_wrapper(wrapper, wrapped)` que le ayuda a escribir decoradores con un buen comportamiento. **`update_wrapper()`** copia el nombre, el módulo y el atributo docstring a una función wrapper para que las trazas dentro de la función envuelta sean más fáciles de entender. Por ejemplo, puedes escribir:

```
def my_decorator(f):
    def wrapper(*args, **kwds):
        print 'Calling decorated function'
        return f(*args, **kwds)
    functools.update_wrapper(wrapper, f)
    return wrapper
```

`wraps()` es un decorador que se puede utilizar dentro de sus propios decoradores para copiar la información de la función envuelta. Una versión alternativa del ejemplo anterior sería:

```
def my_decorator(f):
    @functools.wraps(f)
    def wrapper(*args, **kwds):
        print 'Calling decorated function'
        return f(*args, **kwds)
    return wrapper
```

Ver también

PEP 309 [https://peps.python.org/pep-0309/] - Aplicación parcial de funciones

PEP propuesto y escrito por Peter Harris; implementado por Hye-Shik Chang y Nick Coghlan, con adaptaciones de Raymond Hettinger.

PEP 314: Metadatos para paquetes de software Python v1.1

Se ha añadido a Distutils un sencillo soporte de dependencias. La función **`setup()`** ahora tiene parámetros de palabras clave `requires`, `provides` y `obsoletes`. Cuando se construye una distribución de origen utilizando el comando `sdist`, la información de las dependencias se registrará en el archivo `PKG-INFO`.

Otro nuevo parámetro de palabra clave es `download_url`, que debe establecerse como una URL para el código fuente del paquete. Esto significa que ahora es posible buscar una entrada en el índice de paquetes, determinar las dependencias de un paquete y descargar los paquetes necesarios.

```
VERSION = '1.0'
```

```
setup(name='PyPackage',
      version=VERSION,
      requires=['numarray', 'zlib (>=1.1.4)'],
      obsoletes=['OldPackage']
      download_url=('http://www.example.com/pypackage/di
                    % VERSION),
     )
```

Otra nueva mejora en el índice de paquetes de Python en https://pypi.org es el almacenamiento de archivos fuente y binarios de un paquete. El nuevo comando **upload** de Distutils subirá un paquete al repositorio.

Antes de poder subir un paquete, debes ser capaz de construir una distribución usando el comando **sdist** de Distutils. Una vez que funcione, puedes ejecutar `python setup.py upload` para añadir tu paquete al archivo PyPI. Opcionalmente puedes firmar el paquete con GPG suministrando las opciones **`--sign`** y **`--identity`**.

La carga de paquetes fue implementada por Martin von Löwis y Richard Jones.

Ver también

PEP 314 [https://peps.python.org/pep-0314/] - Metadatos para paquetes de software Python v1.1

PEP propuesto y redactado por A.M. Kuchling, Richard Jones y Fred Drake; aplicado por Richard Jones y Fred Drake.

PEP 328: Importaciones absolutas y relativas

La parte más sencilla de **PEP 328** [https://peps.python.org/pep-0328/] se implementó en Python 2.4: los paréntesis podían utilizarse ahora para encerrar los nombres importados de un módulo utilizando la sentencia `from ... import ...`, facilitando la importación de muchos nombres diferentes.

La parte más complicada se ha implementado en Python 2.5: la importación de un módulo puede especificarse para utilizar importaciones absolutas o relativas al paquete. El plan es hacer que las importaciones absolutas sean el valor por defecto en futuras versiones de Python.

Digamos que tienes un directorio de paquetes como este:

```
pkg/
pkg/__init__.py
pkg/main.py
pkg/string.py
```

Esto define un paquete llamado **`pkg`** que contiene los submódulos **`pkg.main`** y **`pkg.string`**.

Considera el código del módulo `main.py`. ¿Qué ocurre si ejecuta la sentencia `importar cadena`? En Python 2.4 y anteriores, primero buscará en el directorio del paquete para realizar una importación relativa, encuentra `pkg/string.py`, importa el contenido de ese archivo como el módulo **`pkg.string`**, y ese módulo se vincula al nombre `string` en el espacio de nombres del módulo **`pkg.main`**.

Eso está bien si **`pkg.string`** era lo que querías. ¿Pero qué pasa si quieres el módulo estándar de Python **`string`**? No hay una forma limpia de ignorar **`pkg.string`** y buscar el módulo estándar; generalmente tienes que mirar el contenido de `sys.modules`, lo cual es ligeramente sucio. El paquete **`py.std`** de Holger Krekel proporciona una forma más ordenada de realizar importaciones desde la biblioteca estándar, `import py; py.std.string.join()`, pero ese paquete no está disponible en todas las instalaciones de Python.

La lectura de código que depende de importaciones relativas también es menos clara, porque un lector puede confundirse sobre qué módulo, **`cadena`** o **`cadena.pkg`**, se pretende utilizar. Los usuarios de Python aprendieron pronto a no duplicar los nombres de los módulos de la biblioteca estándar en los nombres de los submódulos de sus paquetes, pero no puedes protegerte de que el nombre de tu submódulo se utilice para un nuevo módulo añadido en una futura versión de Python.

En Python 2.5, puedes cambiar el comportamiento de **`import`** a importaciones absolutas usando una directiva `from __future__ import absolute_import`. Este comportamiento de importación absoluta será el predeterminado en una versión futura (probablemente Python 2.7). Una vez que las importaciones absolutas sean el valor por defecto, `import string` siempre encontrará la versión de la biblioteca estándar. Se sugiere que los usuarios comiencen a usar importaciones absolutas tanto como sea posible, así que es preferible comenzar a escribir `de pkg import string` en su código.

Las importaciones relativas siguen siendo posibles añadiendo un punto inicial al nombre del módulo cuando se utiliza la forma `from ... import`:

```
# Import names from pkg.string
from .string import name1, name2
# Import pkg.string
from . import string
```

Esto importa el módulo **`string`** relativo al paquete actual, así que en **`pkg.main`** esto importará *nombre1* y *nombre2* de **`pkg.string`**. Los puntos iniciales adicionales realizan la importación relativa empezando por el padre del paquete actual. Por ejemplo, el código en el módulo **`A.B.C`** puede hacer:

```
from . import D                 # Imports A.B.D
from .. import E                # Imports A.E
from ..F import G               # Imports A.F.G
```

Los puntos suspensivos no pueden usarse con la forma `importar nombre de modelo` de la sentencia import, sólo con la forma `de ... import`.

Ver también

PEP 328 [https://peps.python.org/pep-0328/] - Importaciones: Multilínea y Absoluto/Relativo

PEP escrito por Aahz; implementado por Thomas Wouters.

https://pylib.readthedocs.io/

La biblioteca py de Holger Krekel, que contiene el paquete `py.std`.

PEP 338: Ejecutando Módulos como Scripts

El conmutador `-m` añadido en Python 2.4 para ejecutar un módulo como un script ganó algunas habilidades más. En lugar de estar implementado en código C dentro del intérprete de Python, el interruptor ahora utiliza una implementación en un nuevo módulo, `runpy`.

El módulo `runpy` implementa un mecanismo de importación más sofisticado de forma que ahora es posible ejecutar módulos en un paquete como `pychecker.checker`. El módulo también soporta mecanismos de importación alternativos como el módulo `zipimport`. Esto significa que puede añadir la ruta de un archivo .zip a `sys.path` y luego utilizar el modificador `-m` para ejecutar el código del archivo.

Ver también

PEP 338 [https://peps.python.org/pep-0338/] - Ejecución de módulos como scripts

PEP escrito e implementado por Nick Coghlan.

PEP 341: Try/except/finally unificados

Hasta la versión 2.5 de Python, la sentencia `try` tenía dos variantes. Podías usar un bloque `finally` para asegurarte de que el código se ejecutaba siempre, o uno o más bloques `except` para capturar excepciones específicas. No podías combinar ambos bloques `except` y un bloque `finally`, porque generar el bytecode correcto para la versión combinada era complicado y no estaba claro cuál debía ser la semántica de la sentencia combinada.

Guido van Rossum pasó algún tiempo trabajando con Java, que sí soporta el equivalente de combinar bloques `except` y un bloque `finally`, y esto aclaró lo que debería significar la declaración. En Python 2.5, ahora se puede escribir:

```
try:
    block-1 ...
except Exception1:
    handler-1 ...
except Exception2:
    handler-2 ...
else:
    else-block
finally:
    final-block
```

Se ejecuta el código del *bloque-1*. Si el código lanza una excepción, se comprueban los distintos bloques `except`: si la excepción es de clase `Exception1`, se ejecuta *handler-1*; en caso contrario, si es de clase `Exception2`, se ejecuta *handler-2*, y así sucesivamente. Si no se produce ninguna excepción, se ejecuta el bloque *else*.

No importa lo que haya sucedido previamente, el *bloque final* se ejecuta una vez que el bloque de código se ha completado y se han manejado las excepciones planteadas. Incluso si hay un error en un manejador de excepciones o en el *bloque else* y se lanza una nueva excepción, el código del *bloque final* se sigue ejecutando.

Ver también

PEP 341 [https://peps.python.org/pep-0341/] - Unificar try-except y try-finally

PEP escrito por Georg Brandl; implementación por Thomas Lee.

PEP 342: Nuevas funciones del generador

Python 2.5 añade una forma sencilla de pasar valores *a* un generador. Tal y como se introdujo en Python 2.3, los generadores sólo producen salida; una vez que se invoca el código de un generador para crear un iterador, no hay forma de pasar ninguna información nueva a la función cuando se reanuda su ejecución. A veces, la capacidad de pasar alguna información sería útil. Las soluciones más ingeniosas para esto incluyen hacer que el código del generador mire a una variable global y luego cambie el valor de la variable global, o pasar algún objeto mutable que los llamadores luego modifiquen.

Para refrescar la memoria de los generadores básicos, he aquí un ejemplo sencillo:

```
def counter (maximum):
    i = 0
    while i < maximum:
        yield i
        i += 1
```

Cuando se llama a `contador(10)`, el resultado es un iterador que devuelve los valores de 0 a 9. Al encontrar la sentencia **`yield`**, el iterador devuelve el valor proporcionado y suspende la ejecución de la función, preservando las variables locales. La ejecución se reanuda en la siguiente llamada al método **`next()`** del iterador, retomando después de la sentencia **`yield`**.

En Python 2.3, **`yield`** era una declaración; no devolvía ningún valor. En 2.5, **`yield`** es ahora una expresión, que devuelve un valor que se puede asignar a una variable o que se puede operar de otra manera:

```
val = (rendimiento i)
```

Te recomiendo que siempre pongas paréntesis alrededor de una expresión **`yield`** cuando estés haciendo algo con el valor devuelto, como en el ejemplo anterior. Los paréntesis no siempre son necesarios, pero es más fácil añadirlos siempre en lugar de tener que recordar cuándo son necesarios.

(**PEP 342** [https://peps.python.org/pep-0342/] explica las reglas exactas, que consisten en que una expresión **`yield`**debe ir siempre entre paréntesis, excepto cuando ocurre en la expresión de nivel superior en el lado derecho de una asignación. Esto significa que puedes escribir `val = yield i` pero tienes que usar paréntesis cuando hay una operación, como en `val = (yield i) + 12`)

Los valores se envían a un generador llamando a su método `send(value)`. El código del generador se reanuda y la expresión **`yield`** devuelve el *valor* especificado. Si se llama al método regular **`next()`**, la expresión **`yield`** devuelve **`None`**.

Aquí está el ejemplo anterior, modificado para permitir cambiar el valor del contador interno.

```
def counter (maximum):
    i = 0
    while i < maximum:
        val = (yield i)
        # If value provided, change counter
        if val is not None:
            i = val
        else:
            i += 1
```

Y aquí hay un ejemplo de cambio de contador:

```
>>> it = counter(10)
>>> print it.next()
0
>>> print it.next()
1
>>> print it.send(8)
8
>>> print it.next()
9
>>> print it.next()
Traceback (most recent call last):
  File "t.py", line 15, in ?
    print it.next()
StopIteration
```

`yield` normalmente devolverá **`None`**, por lo que siempre debes comprobar este caso. No utilices su valor en las expresiones sin más, a menos que estés seguro de que el método **`send()`** será el único utilizado para reanudar tu función generadora.

Además de **`send()`**, hay otros dos nuevos métodos en los generadores:

- `throw(type, value=None, traceback=None)` se utiliza para lanzar una excepción dentro del generador; la excepción es lanzada por la expresión **`yield`** donde la ejecución del generador se pausa.

- **`close()`** lanza una nueva excepción **`GeneratorExit`** dentro del generador para terminar la iteración. Al recibir esta excepción, el código del generador debe lanzar **`GeneratorExit`** o **`StopIteration`**. Capturar la excepción **`GeneratorExit`** y devolver un valor es ilegal y provocará un **`RuntimeError`**; si la función lanza alguna otra excepción, esa excepción se propaga a quien la llama. **`close()`** también será llamado por el recolector de basura de Python cuando el generador sea recolectado.

  Si necesitas ejecutar código de limpieza cuando se produce un **`GeneratorExit`**, te sugiero que utilices un conjunto `try: ... finally:` en lugar de atrapar **`GeneratorExit`**.

El efecto acumulativo de estos cambios es que los generadores pasan de ser productores unidireccionales de información a ser tanto productores como consumidores.

Los generadores también se convierten en *corutinas*, una forma más generalizada de subrutinas. Las subrutinas se introducen en un punto y se salen en otro (la parte superior de la función, y una declaración **`return`**), pero las coroutines pueden introducirse, salirse y reanudarse en muchos puntos diferentes (las declaraciones **`yield`**). Tendremos que descubrir patrones para usar coroutines de forma efectiva en Python.

La adición del método **`close()`** tiene un efecto secundario que no es obvio. **`close()`** es llamado cuando un generador es recogido por la basura, lo que significa que el código del generador tiene una última oportunidad de ejecutarse antes de que el generador sea destruido. Esta última oportunidad significa que ahora se puede garantizar que las sentencias `intentar...finalmente` en los generadores funcionen; la cláusula **`finally`** ahora siempre tendrá una oportunidad de ejecutarse. Por lo tanto, se ha eliminado la restricción sintáctica que impedía mezclar sentencias **`yield`** con un conjunto `try...finally`. Esto parece una trivialidad menor del lenguaje, pero el uso de generadores y `try...finally` es realmente necesario para implementar la sentencia **`with`** descrita por PEP 343 [https://peps.python.org/pep-0343/]. Veré esta nueva sentencia en la siguiente sección.

Otro efecto aún más esotérico de este cambio: antes, el atributo **`gi_frame`** de un generador era siempre un objeto frame. Ahora es posible que **`gi_frame`** sea `None` una vez que el generador se ha agotado.

Ver también

PEP 342 [https://peps.python.org/pep-0342/] - Coroutines mediante generadores mejorados

PEP escrito por Guido van Rossum y Phillip J. Eby; implementado por Phillip J. Eby. Incluye ejemplos de algunos usos más sofisticados de los generadores como coroutines.

Versiones anteriores de estas características fueron propuestas en PEP 288 [https://peps.python.org/pep-0288/] por Raymond Hettinger y PEP 325 [https://peps.python.org/pep-0325/] por Samuele Pedroni.

https://en.wikipedia.org/wiki/Coroutine

La entrada de Wikipedia para las coroutines.

https://web.archive.org/web/20160321211320/http://

www.sidhe.org/~dan/blog/archives/000178.html

Una explicación de las coroutines desde el punto de vista de Perl, escrita por Dan Sugalski.

PEP 343: La declaración «con

La sentencia “**`with`**” aclara el código que antes utilizaba bloques `try...finally`” para asegurar que se ejecuta el código de limpieza. En esta sección, hablaré de la sentencia tal y como se utiliza habitualmente. En la siguiente sección, examinaré los detalles de la implementación y mostraré cómo escribir objetos para usar con esta sentencia.

La declaración “**`with`**” es una nueva estructura de flujo de control cuya estructura básica es:

```
with expression [as variable]:
    with-block
```

Se evalúa la expresión y debería dar como resultado un objeto que admita el protocolo de gestión de contexto (es decir, que tenga los métodos **`__enter__()`** y **`__exit__()`**).

El objeto **`__enter__()`** se llama antes de que se ejecute *with-block* y, por lo tanto, puede ejecutar el código de configuración. También puede devolver un valor vinculado al nombre *variable*, si se proporciona. (Tenga en cuenta que *variable* es *not* al que se le asigna el resultado de *expression*).

Una vez finalizada la ejecución de *with-block*, se llama al método **`__exit__()`** del objeto, incluso si el bloque generó una excepción y, por lo tanto, puede ejecutar código de limpieza.

Para habilitar la declaración en Python 2.5, debe añadir la siguiente directiva a su módulo:

```
desde __future__ importar con_declaración
```

La declaración siempre estará habilitada en Python 2.6.

Algunos objetos estándar de Python soportan ahora el protocolo de gestión de contextos y pueden utilizarse con la sentencia “`with`””. Los objetos de archivo son un ejemplo:

```
with open('/etc/passwd', 'r') as f:
    for line in f:
        print line
        ... more processing code ...
```

Después de que esta sentencia se haya ejecutado, el objeto archivo en *f* se habrá cerrado automáticamente, incluso si el bucle `for` lanzó una excepción a mitad del bloque.

Nota

En este caso, *f* es el mismo objeto creado por `open()`, porque `__enter__()` devuelve *self*.

Los bloqueos y las variables de condición del módulo `threading` también soportan la sentencia “`with`”:

```
lock = threading.Lock()
with lock:
    # Critical section of code
    ...
```

El bloqueo se adquiere antes de que se ejecute el bloque y siempre se libera una vez que el bloque se ha completado.

La nueva función `localcontext()` del módulo `decimal` facilita el guardado y la restauración del contexto decimal actual, que encapsula las características de precisión y redondeo deseadas para los cálculos:

```
from decimal import Decimal, Context, localcontext

# Displays with default precision of 28 digits
v = Decimal('578')
print v.sqrt()
```

```
with localcontext(Context(prec=16)):
    # All code in this block uses a precision of 16 digi
    # The original context is restored on exiting the bl
    print v.sqrt()
```

Redacción de Gestores de Contexto

Bajo el capó, la sentencia “`with`” es bastante complicada. La mayoría de la gente sólo utilizará “`with`” en compañía de objetos existentes y no necesita conocer estos detalles, así que puedes saltarte el resto de esta sección si quieres. Los autores de nuevos objetos necesitarán entender los detalles de la implementación subyacente y deberían seguir leyendo.

Una explicación de alto nivel del protocolo de gestión del contexto es:

- La expresión se evalúa y debería generar un objeto denominado «administrador de contexto». El administrador de contexto debe tener los métodos `__enter__()` y `__exit__()`.
- Se llama al método `__enter__()` del administrador de contexto. El valor devuelto se asigna a *VAR*. Si no hay ninguna cláusula `'as VAR'`, el valor simplemente se descarta.
- Se ejecuta el código en *BLOQUE*.
- Si *BLOCK* lanza una excepción, se llama a `__exit__(type, value, traceback)` con los detalles de la excepción, los mismos valores devueltos por `sys.exc_info()`. El valor de retorno del método controla si la excepción se vuelve a lanzar: cualquier valor falso vuelve a lanzar la excepción, y `True` resultará en suprimirla. Sólo en raras ocasiones querrá suprimir la excepción, porque si lo hace el autor del código que contiene la declaración “`with`” nunca se dará cuenta de que algo ha ido mal.
- Si *BLOCK* no generó una excepción, aún se llama al método `__exit__()`, pero *type*, *value* y *traceback* son todos `None`.

Pensemos en un ejemplo. No presentaré un código detallado, sino que sólo esbozaré los métodos necesarios para una base de datos que soporte transacciones.

(Para quienes no estén familiarizados con la terminología de las bases de datos: un conjunto de cambios en la base de datos se agrupa en una transacción. Las transacciones pueden ser confirmadas, lo que significa que todos los cambios se escriben en la base de datos, o revertidas, lo que significa que todos los cambios se descartan y la base de datos no se modifica. Consulte cualquier libro de texto sobre bases de datos para obtener más información)

Supongamos que hay un objeto que representa una conexión a la base de datos. Nuestro objetivo será permitir que el usuario escriba código como este:

```
db_connection = DatabaseConnection()
with db_connection as cursor:
    cursor.execute('insert into ...')
    cursor.execute('delete from ...')
    # ... more operations ...
```

La transacción debe ser confirmada si el código en el bloque se ejecuta sin problemas o revertida si hay una excepción. Aquí está la interfaz básica para `DatabaseConnection` que voy a asumir:

```
class DatabaseConnection:
    # Database interface
    def cursor (self):
        "Returns a cursor object and starts a new transa
    def commit (self):
        "Commits current transaction"
    def rollback (self):
        "Rolls back current transaction"
```

El método `__enter__()` es bastante sencillo, ya que solo hay que iniciar una nueva transacción. Para esta aplicación, el objeto cursor resultante sería un resultado útil, por lo que el método lo devolverá. El usuario puede entonces añadir `as cursor` a su declaración “`with`” para vincular el cursor a un nombre de variable.

```
class DatabaseConnection:
    ...
    def __enter__ (self):
        # Code to start a new transaction
        cursor = self.cursor()
        return cursor
```

El método `__exit__()` es el más complicado porque es donde se debe realizar la mayor parte del trabajo. El método debe verificar si se produjo una excepción. Si no hubo excepción, se confirma la transacción. Si hubo una excepción, se revierte la transacción.

En el código de abajo, la ejecución simplemente caerá al final de la función, devolviendo el valor por defecto de `None`. `None` es falso, por lo que la excepción se volverá a lanzar automáticamente. Si lo desea, puede ser más explícito y añadir una declaración `return` en el lugar marcado.

```
class DatabaseConnection:
    ...
    def __exit__ (self, type, value, tb):
        if tb is None:
            # No exception, so commit
            self.commit()
        else:
            # Exception occurred, so rollback.
            self.rollback()
            # return False
```

El módulo contextlib

El nuevo módulo `contextlib` proporciona algunas funciones y un decorador que son útiles para escribir objetos para usar con la sentencia “`with`”.

El decorador se llama **`contextmanager()`** y le permite escribir una única función generadora en lugar de definir una nueva clase. El generador debería generar exactamente un valor. El código hasta `yield` se ejecutará como el método `__enter__()` y el valor generado será el valor de retorno del método que se vinculará a la variable en la cláusula **`as`** de la declaración “`with`”, si la hay. El código después de **`yield`** se ejecutará en el método **`__exit__()`**. Cualquier excepción generada en el bloque será generada por la declaración **`yield`**.

Nuestro ejemplo de base de datos de la sección anterior podría escribirse utilizando este decorador como:

```
from contextlib import contextmanager

@contextmanager
def db_transaction (connection):
    cursor = connection.cursor()
    try:
        yield cursor
    except:
        connection.rollback()
        raise
    else:
        connection.commit()

db = DatabaseConnection()
with db_transaction(db) as cursor:
    ...
```

El módulo **`contextlib`** también tiene una función `anidada(mgr1, mgr2, ...)` que combina varios gestores de contexto para que no sea necesario escribir sentencias “**`with`**” anidadas. En este ejemplo, la única sentencia “**`with`**` inicia una transacción de base de datos y adquiere un bloqueo de hilo:

```
lock = threading.Lock()
con anidado (db_transaction(db), lock) como (cursor, bl
...
```

Por último, la función `closing(object)` devuelve el *objeto* para que pueda ser vinculado a una variable, y llama a `object.close` al final del bloque.

```
import urllib, sys
from contextlib import closing
```

```
with closing(urllib.urlopen('http://www.yahoo.com')) as
    for line in f:
        sys.stdout.write(line)
```

Ver también

PEP 343 [https://peps.python.org/pep-0343/] - La declaración «con»

PEP escrito por Guido van Rossum y Nick Coghlan; implementado por Mike Bland, Guido van Rossum y Neal Norwitz. El PEP muestra el código generado para una sentencia “`with`”, que puede ser útil para aprender cómo funciona la sentencia.

La documentación del módulo `contextlib`.

PEP 352: Las excepciones como clases de nuevo estilo

Las clases de excepción ahora pueden ser clases de nuevo estilo, no sólo clases clásicas, y la clase incorporada `Exception` y todas las excepciones incorporadas estándar (`NameError`, `ValueError`, etc.) son ahora clases de nuevo estilo.

La jerarquía de herencia de las excepciones se ha reordenado un poco. En 2.5, las relaciones de herencia son:

```
BaseException       # New in Python 2.5
|- KeyboardInterrupt
|- SystemExit
|- Exception
   |- (all other current built-in exceptions)
```

Esta reorganización se hizo porque la gente a menudo quiere atrapar todas las excepciones que indican errores del programa. `KeyboardInterrupt` y `SystemExit` no son errores, sin embargo, y por lo general representan una acción explícita como el usuario pulsando `Control-C` o el código llamando a **`sys.exit()`**. Una simple `except:` atrapará todas las excepciones, por lo que comúnmente se necesita listar **`KeyboardInterrupt`** y **`SystemExit`** para volver a lanzarlas. El patrón habitual es:

```
try:
    ...
except (KeyboardInterrupt, SystemExit):
    raise
except:
    # Log error...
    # Continue running program...
```

En Python 2.5, ahora puedes escribir `except Exception` para conseguir el mismo resultado, capturando todas las excepciones que suelen indicar errores pero dejando **`KeyboardInterrupt`** y **`SystemExit`** en paz. Como en versiones anteriores, un `except:` desnudo sigue capturando todas las excepciones.

El objetivo de Python 3.0 es requerir que cualquier clase lanzada como excepción derive de **`BaseException`** o de algún descendiente de **`BaseException`**, y las futuras versiones de la serie Python 2.x pueden empezar a imponer esta restricción. Por lo tanto, sugiero que empieces a hacer que todas tus clases de excepción deriven de **`Exception`** ahora. Se ha sugerido que la forma desnuda `except:` sea eliminada en Python 3.0, pero Guido van Rossum no ha decidido si hacerlo o no.

El lanzamiento de cadenas como excepciones, como en la declaración `raise "Error occurred"`, está obsoleto en Python 2.5 y provocará una advertencia. El objetivo es poder eliminar la función de excepción de cadena en algunas versiones.

Ver también

PEP 352 [https://peps.python.org/pep-0352/] - Superclase necesaria para las excepciones

PEP escrito por Brett Cannon y Guido van Rossum; implementado por Brett Cannon.

PEP 353: Uso de ssize_t como tipo de índice

Un cambio de gran alcance en la API C de Python, que usa una nueva definición de tipo **`Py_ssize_t`** en lugar de int, permitirá que el intérprete maneje más datos en plataformas de 64 bits. Este cambio no afecta la capacidad de Python en plataformas de 32 bits.

Varias piezas del intérprete de Python usaban el tipo int de C para almacenar tamaños o conteos; por ejemplo, la cantidad de elementos en una lista o tupla se almacenaron en un int. Los compiladores de C para la mayoría de las plataformas de 64 bits aún definen int como un tipo de 32 bits, lo que significa que las listas solo pueden contener hasta `2**31 - 1` = 2147483647 elementos. (En realidad, hay algunos modelos de programación diferentes que los compiladores de C de 64 bits pueden usar; consulte https://unix.org/version2/whatsnew/lp64_wp.html para ver una discusión, pero el modelo más comúnmente disponible deja int como 32 bits).

Un límite de 2147483647 elementos no importa realmente en una plataforma de 32 bits porque te quedarás sin memoria antes de alcanzar el límite de longitud. Cada elemento de la lista requiere espacio para un puntero, que es de 4 bytes, más espacio para un **`PyObject`** que representa el elemento. 2147483647*4 ya son más bytes de los que puede contener un espacio de direcciones de 32 bits.

Sin embargo, es posible abordar esa cantidad de memoria en una plataforma de 64 bits. Los punteros para una lista de ese tamaño solo requerirían 16 GiB de espacio, por lo que no es irrazonable que los programadores de Python puedan construir listas tan grandes. Por lo tanto, el intérprete de Python tuvo que cambiarse para usar algún tipo diferente a int, y este será un tipo de 64 bits en plataformas de 64 bits. El cambio provocará incompatibilidades en las máquinas de 64 bits, por lo que se consideró que valía la pena hacer la transición ahora, mientras que la cantidad de usuarios de 64 bits aún es relativamente pequeña. (En 5 o 10 años, es posible que *todo* esté en máquinas de 64 bits y la transición sería más dolorosa entonces).

Este cambio afecta en mayor medida a los autores de módulos de extensión de C. Las cadenas de Python y los tipos contenedores como las listas y las tuplas utilizan ahora `Py_ssize_t` para almacenar su tamaño. Funciones como `PyList_Size()` ahora devuelven `Py_ssize_t`. Por lo tanto, el código de los módulos de extensión puede necesitar cambiar algunas variables a `Py_ssize_t`.

Las funciones `PyArg_ParseTuple()` y `Py_BuildValue()` tienen un nuevo código de conversión, `n`, para `Py_ssize_t`. `s#` y `t#` de `PyArg_ParseTuple()` aún generan int de forma predeterminada, pero puede definir la macro `PY_SSIZE_T_CLEAN` antes de incluir `Python.h` para que devuelvan `Py_ssize_t`.

PEP 353 [https://peps.python.org/pep-0353/] tiene una sección sobre directrices de conversión que los autores de extensiones deberían leer para aprender a soportar plataformas de 64 bits.

Ver también

PEP 353 [https://peps.python.org/pep-0353/] - Uso de ssize_t como tipo de índice

PEP escrito y aplicado por Martin von Löwis.

PEP 357: El método "_index_"

Los desarrolladores de NumPy tenían un problema que sólo podía resolverse añadiendo un nuevo método especial, `__index__()`. Cuando se utiliza la notación de trozos, como en `[start:stop:step]`, los valores de los índices *start*, *stop* y *step* deben ser todos enteros o enteros largos. NumPy define una variedad de tipos de enteros especializados que corresponden a enteros sin signo y con signo de 8, 16, 32 y 64 bits, pero no había forma de señalar que estos tipos pudieran usarse como índices de trozos.

El rebanado no puede utilizar el método `__int__()` existente porque ese método también se utiliza para implementar la coerción a enteros. Si el rebanado utilizara `__int__()`, los números de punto flotante también se convertirían en índices de rebanada legales y eso es claramente un comportamiento indeseable.

En su lugar, se ha añadido un nuevo método especial llamado `__index__()`. No toma argumentos y devuelve un entero que da el índice de corte a utilizar. Por ejemplo:

```
class C:
    def __index__ (self):
        return self.value
```

El valor devuelto debe ser un entero de Python o un entero largo. El intérprete comprobará que el tipo devuelto es correcto, y lanza un `TypeError` si no se cumple este requisito.

Se agregó una ranura `nb_index` correspondiente a la estructura `PyNumberMethods` de nivel C para permitir que las extensiones C implementen este protocolo. `PyNumber_Index(obj)` se puede utilizar en código de extensión para llamar a la función `__index__()` y recuperar su resultado.

Ver también

PEP 357 [https://peps.python.org/pep-0357/] - Permitir el uso de cualquier objeto para rebanar

PEP escrito e implementado por Travis Oliphant.

Otros cambios lingüísticos

Estos son todos los cambios que Python 2.5 introduce en el núcleo del lenguaje Python.

- El tipo `dict` tiene un nuevo gancho para permitir que las subclases proporcionen un valor por defecto cuando una clave no está contenida en el diccionario. Cuando no se encuentre una clave, se llamará al método `missing__(key)` del diccionario. Este hook se utiliza para implementar la nueva clase **`defaultdict`** en el módulo **`collections`**. El siguiente ejemplo define un diccionario que devuelve cero para cualquier clave que falte:

```
class zerodict (dict):
    def __missing__ (self, key):
        return 0

d = zerodict({1:1, 2:2})
print d[1], d[2]   # Prints 1, 2
print d[3], d[4]   # Prints 0, 0
```

- Tanto las cadenas de 8 bits como las de Unicode tienen nuevos métodos `partition(sep)` y `rpartition(sep)` que simplifican un caso de uso común.

  El método `find(S)` se utiliza a menudo para obtener un índice que luego se utiliza para cortar la cadena y obtener las piezas que están antes y después del separador. El método `partition(sep)` condensa este patrón en una sola llamada al método que devuelve una tripleta que contiene la subcadena antes del separador, el propio separador y la subcadena después del separador. Si no se encuentra el separador, el primer elemento de la tupla es la cadena completa y los otros dos elementos están vacíos. `rpartition(sep)` también devuelve una tupla de 3 elementos, pero empieza a buscar desde el final de la cadena; la `r` significa «al revés».

  Algunos ejemplos:

```
>>> ('http://www.python.org').partition('://')
('http', '://', 'www.python.org')
>>> ('file:/usr/share/doc/index.html').partition(':
('file:/usr/share/doc/index.html', '', '')
>>> (u'Subject: a quick question').partition(':')
(u'Subject', u':', u' a quick question')
>>> 'www.python.org'.rpartition('.')
('www.python', '.', 'org')
>>> 'www.python.org'.rpartition(':')
```

```
('', '', 'www.python.org')
```

(Implementado por Fredrik Lundh tras una sugerencia de Raymond Hettinger)

- Los métodos **`startswith()`** y **`endswith()`** de los tipos de cadena aceptan ahora tuplas de cadenas para su comprobación.

  ```
  def is_image_file (nombre_archivo):
  return nombre_archivo.endswith(('.gif', '.jpg', '.t
  ```

  (Implementado por Georg Brandl tras una sugerencia de Tom Lynn)

- Las funciones incorporadas **`min()`** y **`max()`** han ganado un parámetro de palabra clave `key` análogo al argumento `key` de **`sort()`**. Este parámetro proporciona una función que toma un único argumento y es llamada para cada valor de la lista; **`min()`**/**`max()`** devolverá el elemento con el valor de retorno más pequeño/más grande de esta función. Por ejemplo, para encontrar la cadena más larga de una lista, puede hacer:

  ```
  L = ['medium', 'longest', 'short']
  # Prints 'longest'
  print max(L, key=len)
  # Prints 'short', because lexicographically 'short'
  print max(L)
  ```

  (Contribución de Steven Bethard y Raymond Hettinger)

- Dos nuevas funciones incorporadas, **`any()`** y **`all()`**, evalúan si un iterador contiene algún valor verdadero o falso. **`any()`** devuelve **`True`** si cualquier valor devuelto por el iterador es verdadero; en caso contrario devolverá **`False`**. **`all()`** devuelve **`True`** sólo si todos los valores devueltos por el iterador se evalúan como verdaderos. (Sugerido por Guido van Rossum, e implementado por Raymond Hettinger)

- El resultado del método **`__hash__()`** de una clase puede ser ahora un entero largo o un entero normal. Si se devuelve un entero largo, se toma el hash de ese valor. En versiones anteriores se requería que el valor del hash fuera un entero regular, pero en 2.5 el built-in **`id()`** se cambió para devolver siempre números no negativos, y los usuarios parecen utilizar a menudo `id(self)` en los métodos **`__hash__()`** (aunque esto se desaconseja).

- ASCII es ahora la codificación por defecto para los módulos. Ahora es un error de sintaxis si un módulo contiene literales de cadena con caracteres de 8 bits pero no tiene una declaración de codificación. En Python 2.4 esto provocaba una advertencia, no un error de sintaxis. Vea en **PEP 263** [https://peps.python.org/pep-0263/] cómo declarar la codificación de un módulo; por ejemplo, puede añadir una línea como ésta cerca de la parte superior del fichero fuente:

  ```
  #-*- coding: latin1 -*-
  ```

- Una nueva advertencia, **`UnicodeWarning`**, se activa cuando se intenta comparar una cadena Unicode y una cadena de 8 bits que no se puede convertir a Unicode utilizando la codificación ASCII por defecto. El resultado de la comparación es falso:

  ```
  >>> chr(128) == unichr(128)   # Can't convert chr(1
  __main__:1: UnicodeWarning: Unicode equal compariso
    to convert both arguments to Unicode - interpreti
    as being unequal
  False
  >>> chr(127) == unichr(127)   # chr(127) can be con
  True
  ```

  Anteriormente, esto lanzaba una excepción **`UnicodeDecodeError`**, pero en 2.5 esto podía dar lugar a problemas desconcertantes al acceder a un diccionario. Si se buscaba `unichr(128)` y se utilizaba `chr(128)` como clave, se producía una excepción **`UnicodeDecodeError`**. Otros cambios en la versión 2.5 hicieron que esta excepción se lanzara en lugar de ser suprimida por el código de `dictobject.c` que implementa los diccionarios.

Lanzar una excepción para tal comparación es estrictamente correcto, pero el cambio podría haber roto el código, así que en su lugar se introdujo **`UnicodeWarning`**.

(Implementado por Marc-André Lemburg.)

- Un error que a veces cometen los programadores de Python es olvidarse de incluir un módulo `__init__.py` en el directorio de un paquete. Depurar este error puede ser confuso, y normalmente requiere ejecutar Python con el modificador **`-v`** para registrar todas las rutas buscadas. En Python 2.5, una nueva advertencia **`ImportWarning`** se activa cuando una importación habría recogido un directorio como paquete pero no se encontró ningún `__init__.py`. Esta advertencia se ignora silenciosamente por defecto; proporcione la opción **`-Wd`** cuando ejecute el ejecutable de Python para mostrar el mensaje de advertencia. (Implementado por Thomas Wouters)</-W>

- La lista de clases base en una definición de clase ahora puede estar vacía. Como ejemplo, esto es ahora legal:

  ```
  clase C():
      pass
  ```

  (Implementado por Brett Cannon.)

Cambios en el intérprete interactivo

En el intérprete interactivo, `quit` y `exit` han sido durante mucho tiempo cadenas para que los nuevos usuarios obtengan un mensaje algo útil cuando intenten salir:

```
>>> salir
'Utilice Ctrl-D (es decir, EOF) para salir.'
```

En Python 2.5, `quit` y `exit` son ahora objetos que siguen produciendo representaciones de cadena de sí mismos, pero también son invocables. Los novatos que prueben `quit()` o `exit()` ahora saldrán del intérprete como se espera. (Implementado por Georg Brandl.)

El ejecutable de Python ahora acepta las opciones largas estándar **`--help`** y **`--version`**; en Windows, también acepta la opción **`/?`** para mostrar un mensaje de ayuda. (Implementado por Georg Brandl)

Optimizaciones

Varias de las optimizaciones se desarrollaron en el sprint NeedForSpeed, un evento celebrado en Reikiavik, Islandia, del 21 al 28 de mayo de 2006. El sprint se centró en las mejoras de velocidad de la implementación de CPython y fue financiado por EWT LLC con el apoyo local de CCP Games. Las optimizaciones añadidas en este sprint están especialmente marcadas en la siguiente lista.

- Cuando se introdujeron en Python 2.4, los tipos incorporados **`set`** y **`frozenset`** se construyeron sobre el tipo diccionario de Python. En 2.5 la estructura de datos interna se ha personalizado para implementar conjuntos, y como resultado los conjuntos utilizarán un tercio menos de memoria y son algo más rápidos. (Implementado por Raymond Hettinger)
- Se ha mejorado la velocidad de algunas operaciones Unicode, como la búsqueda de subcadenas, la división de cadenas y la codificación y decodificación de mapas de caracteres. (Las mejoras en la búsqueda de subcadenas y la división fueron añadidas por Fredrik Lundh y Andrew Dalke en el sprint NeedForSpeed. Los mapas de caracteres fueron mejorados por Walter Dörwald y Martin von Löwis)
- La función `long(str, base)` es ahora más rápida en cadenas de dígitos largos porque se calculan menos resultados intermedios. El máximo es para cadenas de alrededor de 800-1000 dígitos, donde la función es 6 veces más rápida. (Aportado por Alan McIntyre y comprometido en el sprint NeedForSpeed)
- Ahora es ilegal mezclar la iteración sobre un archivo con `for line in file` y llamar a los métodos **`read()`**/**`readline()`**/**`readlines()`** del objeto de archivo. La iteración utiliza un búfer interno y los métodos **`read*()`** no

utilizan ese búfer. En su lugar, devolverían los datos después del búfer, lo que haría que los datos aparecieran desordenados. Mezclar la iteración y estos métodos ahora activará un `ValueError` desde el método `read*()`. (Implementado por Thomas Wouters).

- El módulo `struct` ahora compila cadenas de formato de estructura en una representación interna y almacena en caché esta representación, lo que supone una mejora del 20% s. (Contribuido por Bob Ippolito en el sprint NeedForSpeed)
- El módulo `re` obtuvo un 1 o 2% s de velocidad al cambiar a las funciones asignadoras de Python en lugar de las `malloc()` y `free()` del sistema. (Contribuido por Jack Diederich en el sprint NeedForSpeed)
- El optimizador de la mirilla del generador de código realiza ahora un simple plegado de constantes en las expresiones. Si escribes algo como `a = 2+3`, el generador de código hará la aritmética y producirá el código correspondiente a `a = 5`. (Propuesto e implementado por Raymond Hettinger)
- Las llamadas a funciones son ahora más rápidas porque los objetos de código guardan ahora el último fotograma terminado (un «fotograma zombi») en un campo interno del objeto de código, reutilizándolo la próxima vez que se invoque el objeto de código. (Parche original de Michael Hudson, modificado por Armin Rigo y Richard Jones; confirmado en el sprint NeedForSpeed) Los objetos marco son también ligeramente más pequeños, lo que puede mejorar la localidad de la caché y reducir un poco el uso de la memoria. (Contribución de Neal Norwitz)
- Las excepciones incorporadas en Python son ahora clases de nuevo estilo, un cambio que acelera considerablemente la instanciación. El manejo de excepciones en Python 2.5 es, por tanto, un 30% f más rápido que en 2.4. (Contribución de Richard Jones, Georg Brandl y Sean Reifschneider en el sprint NeedForSpeed)
- La importación ahora almacena en caché las rutas intentadas, registrando si existen o no para que el intérprete haga menos llamadas a `open()` y `stat()` al iniciar. (Contribución de Martin von Löwis y Georg Brandl)

Módulos nuevos, mejorados y eliminados

La biblioteca estándar ha recibido muchas mejoras y correcciones de errores en Python 2.5. Aquí hay una lista parcial de los cambios más notables, ordenados alfabéticamente por el nombre del módulo. Consulte el archivo `Misc/NEWS` en el árbol de fuentes para obtener una lista más completa de los cambios, o busque en los registros de SVN para obtener todos los detalles.

- El módulo **`audioop`** ahora admite la codificación a-LAW y se ha mejorado el código para la codificación u-LAW. (Contribución de Lars Immisch).

- El módulo **`codecs`** ha ganado soporte para códecs incrementales. La función **`codec.lookup()`** devuelve ahora una instancia de **`CodecInfo`** en lugar de una tupla. Las instancias de **`CodecInfo`** se comportan como una cuádruple tupla para preservar la compatibilidad con versiones anteriores, pero también tienen los atributos **`encode`**, **`decode`**, **`incrementalencoder`**, **`incrementaldecoder`**, **`streamwriter`** y **`streamreader`**. Los códecs incrementales pueden recibir la entrada y producir la salida en varios trozos; la salida es la misma que si la entrada completa fuera alimentada por el códec no incremental. Consulte la documentación del módulo **`codecs`** para más detalles. (Diseñado e implementado por Walter Dörwald)

- El módulo **`collections`** ha ganado un nuevo tipo, **`defaultdict`**, que subclasifica el tipo estándar **`dict`**. El nuevo tipo se comporta principalmente como un diccionario pero construye un valor por defecto cuando una clave no está presente, añadiéndolo automáticamente al diccionario para el valor de la clave solicitada.

  El primer argumento del constructor de **`defaultdict`** es una función de fábrica que se llama cada vez que se solicita una clave pero no se encuentra. Esta función de fábrica no recibe argumentos, por lo que puede utilizar constructores de tipo incorporado como **`list()`** o **`int()`**. Por ejemplo, puedes hacer un índice de palabras basado en su letra inicial así:

```
palabras = """Nel mezzo del cammin di nuestra vita
mi ritrovai per una selva oscura
che la diritta via era smarrita""".lower().split()

índice = dictamen predeterminado (lista)

para w en palabras:
    letra_inicial = w[0]
    índice[letra_inicio].append(w)
```

Al imprimir `index` se obtiene la siguiente salida:

```
defaultdict(<type 'list'>, {'c': ['cammin', 'che'],
        'd': ['del', 'di', 'diritta'], 'm': ['mezzo
        'l': ['la'], 'o': ['oscura'], 'n': ['nel',
        'p': ['per'], 's': ['selva', 'smarrita'],
        'r': ['ritrovai'], 'u': ['una'], 'v': ['vit
```

(Contribución de Guido van Rossum.)

- El tipo de cola doble **`deque`** suministrado por el módulo **`collections`** tiene ahora un método `remove(value)` que elimina la primera aparición de *value* en la cola, lanzando **`ValueError`** si no se encuentra el valor. (Contribución de Raymond Hettinger)

- Nuevo módulo: El módulo **`contextlib`** contiene funciones de ayuda para usar con la nueva sentencia “**`with`**”. Consulte la sección El módulo contextlib para obtener más información sobre este módulo.

- Nuevo módulo: El módulo **`cProfile`** es una implementación en C del módulo existente **`profile`** que tiene una sobrecarga mucho menor. La interfaz del módulo es la misma que la de **`profile`**: se ejecuta `cProfile.run('main()')` para perfilar una función, se pueden guardar los datos del perfil en un archivo, etc. Todavía no se sabe si el perfilador Hotshot, que también está escrito en C pero no coincide con la interfaz del módulo **`profile`**, seguirá manteniéndose en futuras versiones de Python. (Contribución de Armin Rigo.)

Además, el módulo **`pstats`** para analizar los datos medidos por el perfilador ahora soporta dirigir la salida a cualquier objeto archivo suministrando un argumento *stream* al constructor **`Stats`**. (Contribución de Skip Montanaro)

- El módulo **`csv`**, que analiza archivos en formato de valores separados por comas, recibió varias mejoras y varias correcciones de errores. Ahora puede establecer el tamaño máximo en bytes de un campo llamando a la función `csv.field_size_limit(new_limit)`; omitir el argumento *new_limit* devolverá el límite establecido actualmente. La clase **`reader`** ahora tiene un atributo **`line_num`** que cuenta la cantidad de líneas físicas leídas de la fuente; los registros pueden abarcar varias líneas físicas, por lo que **`line_num`** no es lo mismo que el número de registros leídos.

  El analizador CSV es ahora más estricto con los campos entrecomillados de varias líneas. Anteriormente, si una línea terminaba dentro de un campo entre comillas sin un carácter de nueva línea final, se insertaba una nueva línea en el campo devuelto. Este comportamiento causaba problemas cuando se leían archivos que contenían caracteres de retorno de carro dentro de los campos, por lo que se cambió el código para devolver el campo sin insertar nuevas líneas. Como consecuencia, si las nuevas líneas incrustadas dentro de los campos son importantes, la entrada debe dividirse en líneas de manera que se conserven los caracteres de nueva línea.

  (Contribución de Skip Montanaro y Andrew McNamara)

- La clase **`datetime`** del módulo **`datetime`** tiene ahora un método `strptime(string, format)` para analizar cadenas de fechas, aportado por Josh Spoerri. Utiliza los mismos caracteres de formato que **`time.strptime()`** y **`time.strftime()`**:

```
import datetime as dt
```

```
ts = dt.datetime.strptime('10:13:15 2006-03-07',
                                  '%H:%M:%S %Y-%m-%d')
```

- El método **`SequenceMatcher.get_matching_blocks()`** del módulo **`difflib`** garantiza ahora la devolución de una lista mínima de bloques que describen subsecuencias coincidentes. Anteriormente, el algoritmo rompía ocasionalmente un bloque de elementos coincidentes en dos entradas de la lista. (Mejora realizada por Tim Peters)

- El módulo **`doctest`** ha ganado una opción `SKIP` que impide que un ejemplo se ejecute en absoluto. Esto está pensado para los fragmentos de código que son ejemplos de uso destinados al lector y que no son realmente casos de prueba.

  Se ha añadido un parámetro *encoding* a la función **`testfile()`** y a la clase **`DocFileSuite`** para especificar la codificación del archivo. Esto facilita el uso de caracteres no ASCII en las pruebas contenidas en un docstring. (Contribución de Bjorn Tillenius)

- El paquete **`email`** ha sido actualizado a la versión 4.0. (Contribución de Barry Warsaw.)

- El módulo **`fileinput`** se hizo más flexible. Ahora se admiten los nombres de archivo Unicode y se agregó un parámetro *mode* que tiene como valor predeterminado `"r"` a la función **`input()`** para permitir la apertura de archivos en modo binario o universal newlines. Otro parámetro nuevo, *openhook*, le permite usar una función distinta de **`open()`** para abrir los archivos de entrada. Una vez que esté iterando sobre el conjunto de archivos, el nuevo **`fileno()`** del objeto **`FileInput`** devuelve el descriptor de archivo para el archivo abierto actualmente. (Contribuido por Georg Brandl.)

- En el módulo **`gc`**, la nueva función **`get_count()`** devuelve una 3-tupla que contiene los recuentos de recolección actuales para las tres generaciones de GC. Se trata de información contable para el recolector de basura; cuando estos recuentos alcanzan un umbral especificado, se realiza un barrido de recolección de basura. La función `gc.collect()` existente ahora toma un argumento opcional *generación* de 0, 1 o 2 para especificar qué generación recoger. (Contribución de Barry Warsaw)

- Las funciones **`nsmallest()`** y **`nlargest()`** del módulo **`heapq`** soportan ahora un parámetro de palabra clave `key` similar al proporcionado por las funciones `min()`/`max()` y los métodos **`sort()`**. Por ejemplo:

  ```
  >>> import heapq
  >>> L = ["short", 'medium', 'longest', 'longer stil
  >>> heapq.nsmallest(2, L)  # Return two lowest elem
  ['longer still', 'longest']
  >>> heapq.nsmallest(2, L, key=len)   # Return two s
  ['short', 'medium']
  ```

  (Contribución de Raymond Hettinger.)

- La función `itertools.islice()` ahora acepta `None` para los argumentos de inicio y paso. Esto la hace más compatible con los atributos de los objetos slice, por lo que ahora se puede escribir lo siguiente:

  ```
  s = slice(5) # Crear objeto slice
  itertools.islice(iterable, s.start, s.stop, s.step)
  ```

  (Contribución de Raymond Hettinger.)

- Se ha modificado la función `format()` del módulo `locale` y se han añadido dos nuevas funciones, **`format_string()`** y **`currency()`**.

  El parámetro *val* de la función `format()` podía ser antes una cadena siempre que no apareciera más de un especificador %char; ahora el parámetro debe ser exactamente un especificador %char sin texto alrededor. También se ha añadido un parámetro opcional *monetario* que, si es `Verdadero`, utilizará las reglas de la configuración regional para formatear la moneda al colocar un separador entre grupos de tres dígitos.

Para formatear cadenas con múltiples especificadores %char, utilice la nueva función **`format_string()`** que funciona como **`format()`** pero que también permite mezclar especificadores %char con texto arbitrario.

También se ha añadido una nueva función **`currency()`** que formatea un número de acuerdo con la configuración local actual.

(Contribución de Georg Brandl.)

- El módulo **`mailbox`** ha sido reestructurado de forma masiva para añadir la capacidad de modificar los buzones además de leerlos. Un nuevo conjunto de clases que incluyen **`mbox`**, **`MH`**, y **`Maildir`** se utilizan para leer buzones, y tienen un método `add(message)` para añadir mensajes, `remove(key)` para eliminar mensajes, y **`lock()`**/**`unlock()`** para bloquear/desbloquear el buzón. El siguiente ejemplo convierte un buzón con formato maildir en uno con formato mbox:

  ```
  importar buzón

  # 'factory=None' utiliza email.Message.Message como
  # los mensajes individuales.
  src = mailbox.Maildir('maildir', factory=None)
  dest = mailbox.mbox('/tmp/mbox')

  para msg en src:
  dest.add(msg)
  ```

  (Contribución de Gregory K. Johnson. La financiación fue proporcionada por el Summer of Code 2005 de Google)

- Nuevo módulo: el módulo **`msilib`** permite crear archivos `.msi` y archivos CAB de Microsoft Installer. También se incluye cierta compatibilidad para leer la base de datos `.msi`. (Contribución de Martin von Löwis).

- El módulo **`nis`** ahora permite acceder a dominios distintos del dominio predeterminado del sistema mediante el suministro de un argumento *domain* a las funciones

`nis.match()` y **`nis.maps()`**. (Contribuido por Ben Bell).

- Las funciones **`operator`** del módulo **`itemgetter()`** y **`attrgetter()`** ahora soportan múltiples campos. Una llamada como `operator.attrgetter('a', 'b')` devolverá una función que recupera los atributos **`a`** y **`b`**. La combinación de esta nueva función con el parámetro `key` del método **`sort()`** permite ordenar fácilmente las listas utilizando varios campos. (Contribución de Raymond Hettinger)

- El módulo **`optparse`** se ha actualizado a la versión 1.5.1 de la biblioteca Optik. La clase **`OptionParser`** obtuvo un atributo **`epilog`**, una cadena que se imprimirá después del mensaje de ayuda, y un método **`destroy()`** para romper los ciclos de referencia creados por el objeto. (Contribuido por Greg Ward.)

- El módulo **`os`** ha sufrido varios cambios. La variable **`stat_float_times`** ahora está predeterminada a true, lo que significa que **`os.stat()`** ahora devolverá los valores de tiempo como flotantes. (Esto no significa necesariamente que **`os.stat()`** devolverá tiempos con una precisión de fracciones de segundo; no todos los sistemas soportan dicha precisión)

  Se han añadido las constantes **`os.SEEK_SET`**, **`os.SEEK_CUR`** y **`os.SEEK_END`**, que son los parámetros de la función **`os.lseek()`**. Dos nuevas constantes para el bloqueo son **`os.O_SHLOCK`** y **`os.O_EXLOCK`**.

  Se han añadido dos nuevas funciones, **`wait3()`** y **`wait4()`**. Son similares a la función **`waitpid()`** que espera la salida de un proceso hijo y devuelve una tupla con el ID del proceso y su estado de salida, pero **`wait3()`** y **`wait4()`** devuelven información adicional. **`wait3()`** no toma un ID de proceso como entrada, por lo que espera a que cualquier proceso hijo salga y devuelve una tupla de *id de proceso*, *estado de salida*, *uso de recursos* tal y como se devuelve desde la función **`resource.getrusage()`**. `wait4(pid)` toma un ID de proceso. (Contribuido por Chad J. Schroeder.)

En FreeBSD, la función **`os.stat()`** devuelve ahora tiempos con resolución de nanosegundos, y el objeto devuelto tiene ahora **`st_gen`** y **`st_birthtime`**. El atributo **`st_flags`** también está disponible, si la plataforma lo soporta. (Contribución de Antti Louko y Diego Pettenò)

- El depurador de Python proporcionado por el módulo **`pdb`** puede ahora almacenar listas de comandos a ejecutar cuando se alcanza un punto de ruptura y se detiene la ejecución. Una vez creado el punto de interrupción nº 1, introduzca `comandos 1` e introduzca una serie de comandos a ejecutar, terminando la lista con `end`. La lista de comandos puede incluir comandos que reanuden la ejecución, como `continue` o `next`. (Contribución de Grégoire Dooms.)

- Los módulos **`pickle`** y **`cPickle`** ya no aceptan un valor de retorno de `None` del método **`__reduce__()`**; en su lugar, el método debe devolver una tupla de argumentos. La capacidad de devolver `None` quedó obsoleta en Python 2.4, por lo que esto completa la eliminación de la función.

- El módulo **`pkgutil`**, que contiene varias funciones de utilidad para encontrar paquetes, fue mejorado para soportar los ganchos de importación de **PEP 302** [https://peps.python.org/pep-0302/] y ahora también funciona para paquetes almacenados en archivos con formato ZIP. (Contribución de Phillip J. Eby)

- El conjunto de pruebas pybench de Marc-André Lemburg se incluye ahora en el directorio `Tools/pybench`. El conjunto de pruebas pybench es una mejora del programa `pystone.py` de uso común, ya que pybench proporciona una medición más detallada de la velocidad del intérprete. Calcula el tiempo de determinadas operaciones como las llamadas a funciones, el corte de tuplas, las búsquedas de métodos y las operaciones numéricas, en lugar de realizar muchas operaciones diferentes y reducir el resultado a un único número como hace `pystone.py`.

- El módulo **`pyexpat`** utiliza ahora la versión 2.0 del analizador sintáctico Expat. (Contribución de Trent Mick)

- La clase `Queue` proporcionada por el módulo `Queue` ha ganado dos nuevos métodos. `join()` bloquea hasta que todos los elementos de la cola han sido recuperados y todo el trabajo de procesamiento de los elementos ha sido completado. Los hilos de trabajo llaman al otro método nuevo, `task_done()`, para indicar que el procesamiento de un elemento ha finalizado. (Contribución de Raymond Hettinger)

- Los antiguos módulos `regex` y `regsub`, obsoletos desde Python 2.0, han sido finalmente eliminados. Otros módulos eliminados: `statcache`, `tzparse`, `whrandom`.

- También se ha eliminado el directorio `lib-old`, que incluye módulos antiguos como `dircmp` y `ni`. `lib-old` no estaba en el directorio por defecto `sys.path`, así que a menos que tus programas añadan explícitamente el directorio a `sys.path`, esta eliminación no debería afectar a tu código.

- El módulo `rlcompleter` ya no depende de la importación del módulo `readline` y, por lo tanto, ahora funciona en plataformas no Unix. (Parche de Robert Kiendl.)

- Las clases `SimpleXMLRPCServer` y `DocXMLRPCServer` ahora tienen un atributo `rpc_paths` que restringe las operaciones XML-RPC a un conjunto limitado de rutas URL; el valor predeterminado es permitir solo `'/'` y `'/RPC2'`. Si se configura `rpc_paths` en `None` o en una tupla vacía, se deshabilita esta verificación de ruta.

- El módulo `socket` ahora soporta sockets `AF_NETLINK` en Linux, gracias a un parche de Philippe Biondi. Los sockets Netlink son un mecanismo específico de Linux para las comunicaciones entre un proceso en el espacio del usuario y el código del kernel; hay un artículo introductorio sobre ellos en https://www.linuxjournal.com/article/7356. En el código Python, las direcciones de netlink se representan como una tupla de 2 enteros, `(pid, group_mask)`.

  Dos nuevos métodos en objetos socket, `recv_into(buffer)` y `recvrom_into(buffer)`, almacenan los datos recibidos en un objeto que soporta el protocolo de buffer en lugar de devolver los datos como una cadena. Esto significa que puedes poner los datos directamente en un array o en un archivo mapeado en memoria.

Los objetos Socket también han ganado métodos de acceso **`getfamily()`**, **`gettype()`** y **`getproto()`** para recuperar los valores de familia, tipo y protocolo del socket.

- Nuevo módulo: el módulo **`spwd`** proporciona funciones para acceder a la base de datos de contraseñas ocultas en sistemas que admiten contraseñas ocultas.

- El módulo **`struct`** es ahora más rápido porque compila cadenas de formato en objetos **`Struct`** con métodos **`pack()`** y **`unpack()`**. Esto es similar a cómo el módulo **`re`** permite crear objetos de expresión regular compilados. Puede seguir utilizando las funciones **`pack()`** y **`unpack()`** a nivel de módulo; éstas crearán objetos **`Struct`** y los almacenarán en caché. O puede utilizar directamente las instancias de **`Struct`**:

  ```
  s = struct.Struct('ih3s')

  data = s.pack(1972, 187, 'abc')
  year, number, name = s.unpack(data)
  ```

  También puedes empaquetar y desempaquetar datos hacia y desde objetos buffer directamente usando los métodos `pack_into(buffer, offset, v1, v2, ...)` y `unpack_from(buffer, offset)`. Esto te permite almacenar datos directamente en un array o en un archivo mapeado en memoria.

  (**`Struct`** objects fueron implementados por Bob Ippolito en el sprint NeedForSpeed. El soporte para los objetos buffer fue añadido por Martin Blais, también en el sprint NeedForSpeed)

- Los desarrolladores de Python cambiaron de CVS a Subversion durante el proceso de desarrollo de la versión 2.5.

La información sobre la versión exacta de construcción está disponible como la variable `sys.subversion`, una 3-tupla de `(nombre del intérprete, nombre de la rama, rango de revisión)`. Por ejemplo, en el momento de escribir esto, mi copia de 2.5 informaba de `('CPython', 'trunk', '45313:45315')`.

Esta información también está disponible para las extensiones de C a través de la función **`Py_GetBuildInfo()`** que devuelve una cadena de información de compilación como esta `"trunk:45355:45356M, Apr 13 2006, 07:42:19"`. (Contribuido por Barry Warsaw.)

- Otra nueva función, **`sys._current_frames()`**, devuelve los marcos de pila actuales para todos los hilos en ejecución como un diccionario que asigna los identificadores de los hilos al marco de pila superior actualmente activo en ese hilo en el momento en que se llama a la función. (Contribuido por Tim Peters.)

- La clase **`TarFile`** del módulo **`tarfile`** tiene ahora un método **`extractall()`** que extrae todos los miembros del archivo en el directorio de trabajo actual. También es posible establecer un directorio diferente como destino de la extracción, y desempaquetar sólo un subconjunto de los miembros del archivo.

  La compresión utilizada para un archivo tar abierto en modo stream puede ahora ser autodetectada utilizando el modo `'r|*'`. (Contribución de Lars Gustäbel)

- El módulo **`threading`** permite ahora establecer el tamaño de la pila utilizado cuando se crean nuevos hilos. La función `stack_size([*size*])` devuelve el tamaño de pila actualmente configurado, y suministrando el parámetro opcional *size* establece un nuevo valor. No todas las plataformas soportan el cambio del tamaño de la pila, pero Windows, POSIX threading y OS/2 lo hacen. (Contribución de Andrew MacIntyre)

- El módulo **`unicodedata`** ha sido actualizado para utilizar la versión 4.1.0 de la base de datos de caracteres Unicode. La versión 3.2.0 es requerida por algunas especificaciones, por lo que sigue estando disponible como `unicodedata.ucd_3_2_0`.

- Nuevo módulo: el módulo `uuid` genera identificadores únicos universales (UUID) de acuerdo con RFC 4122 [https://datatracker.ietf.org/doc/html/rfc4122.html]. El RFC define varias versiones diferentes de UUID que se generan a partir de una cadena inicial, de propiedades del sistema o de forma puramente aleatoria. Este módulo contiene una clase **`UUID`** y funciones llamadas **`uuid1()`**, **`uuid3()`**, **`uuid4()`**, y **`uuid5()`** para generar diferentes versiones de UUID. (Los UUID de la versión 2 no se especifican en RFC 4122 [https://datatracker.ietf.org/doc/html/rfc4122.html] y no están soportados por este módulo)

```
>>> import uuid
>>> # make a UUID based on the host ID and current
>>> uuid.uuid1()
UUID('a8098c1a-f86e-11da-bd1a-00112444be1e')

>>> # make a UUID using an MD5 hash of a namespace
>>> uuid.uuid3(uuid.NAMESPACE_DNS, 'python.org')
UUID('6fa459ea-ee8a-3ca4-894e-db77e160355e')

>>> # make a random UUID
>>> uuid.uuid4()
UUID('16fd2706-8baf-433b-82eb-8c7fada847da')

>>> # make a UUID using a SHA-1 hash of a namespace
>>> uuid.uuid5(uuid.NAMESPACE_DNS, 'python.org')
UUID('886313e1-3b8a-5372-9b90-0c9aee199e5d')
```

  (Contribución de Ka-Ping Yee.)

- Los tipos `weakref` del módulo **`WeakKeyDictionary`** y **`WeakValueDictionary`** han obtenido nuevos métodos para iterar sobre las referencias débiles contenidas en el diccionario. Se han añadido los métodos **`iterkeyrefs()`** y

`keyrefs()` a **`WeakKeyDictionary`**, y **`itervaluerefs()`** y **`valuerefs()`** a **`WeakValueDictionary`**. (Contribución de Fred L. Drake, Jr.)

- El módulo **`webbrowser`** ha recibido una serie de mejoras. Ahora se puede utilizar como un script con `python -m webbrowser`, tomando una URL como argumento; hay una serie de interruptores para controlar el comportamiento (**`-n`** para una nueva ventana del navegador, **`-t`** para una nueva pestaña). Se han añadido nuevas funciones a nivel de módulo, **`open_new()`** y **`open_new_tab()`**, para soportar esto. La función **`open()`** del módulo soporta una característica adicional, un parámetro *autoraise* que señala si se debe levantar la ventana abierta cuando sea posible. Se añadieron varios navegadores adicionales a la lista de soportados, como Firefox, Opera, Konqueror y elinks. (Contribución de Oleg Broytmann y Georg Brandl)

- El módulo **`xmlrpclib`** ahora admite la devolución de objetos **`datetime`** para el tipo de fecha XML-RPC. Proporcione `use_datetime=True` a la función **`loads()`** o a la clase **`Unmarshaller`** para habilitar esta función. (Contribuido por Skip Montanaro).

- El módulo **`zipfile`** ahora soporta la versión ZIP64 del formato, lo que significa que un archivo .zip ahora puede ser mayor de 4 GiB y puede contener archivos individuales mayores de 4 GiB. (Contribución de Ronald Oussoren)

- Los objetos **`zlib`** y **`Decompress`** del módulo **`zlib`** soportan ahora un método **`copy()`** que realiza una copia del estado interno del objeto y devuelve un nuevo objeto **`Compress`** o **`Decompress`**. (Contribución de Chris AtLee)

El paquete ctypes

El paquete **`ctypes`**, escrito por Thomas Heller, se ha añadido a la biblioteca estándar. **`ctypes`** permite llamar a funciones arbitrarias en bibliotecas compartidas o DLL. Los usuarios habituales pueden recordar el módulo **`dl`**, que proporciona funciones para cargar bibliotecas compartidas y llamar a funciones en ellas. El paquete **ctypes** es mucho más sofisticado.

Para cargar una biblioteca compartida o DLL, debes crear una instancia de la clase **CDLL** y proporcionar el nombre o la ruta de la biblioteca compartida o DLL. Una vez hecho esto, puedes llamar a funciones arbitrarias accediendo a ellas como atributos del objeto **CDLL**.

```
import ctypes

libc = ctypes.CDLL('libc.so.6')
result = libc.printf("Line of output\n")
```

Se proporcionan constructores de tipo para los distintos tipos de C: **c_int()**, **c_float()**, **c_double()**, **c_char_p()** (equivalente a char*), etcétera. A diferencia de los tipos de Python, las versiones C son todas mutables; puede asignar a su atributo **value** para cambiar el valor envuelto. Los enteros y las cadenas de Python se convertirán automáticamente a los tipos C correspondientes, pero para otros tipos debe llamar al constructor de tipo correcto. (Y me refiero a *debe*; hacerlo mal a menudo resultará en que el intérprete falle con una falla de segmentación).

No debería usar **c_char_p()** con una cadena de Python cuando la función C vaya a modificar el área de memoria, porque se supone que las cadenas de Python son inmutables; romper esta regla causará errores desconcertantes. Cuando necesite un área de memoria modificable, utilice **create_string_buffer()**:

```
s = "this is a string"
buf = ctypes.create_string_buffer(s)
libc.strfry(buf)
```

Se supone que las funciones en C devuelven números enteros, pero se puede establecer el atributo **restype** del objeto de la función para cambiar esto:

```
>>> libc.atof('2.71828')
-1783957616
>>> libc.atof.restype = ctypes.c_double
>>> libc.atof('2.71828')
2.71828
```

`ctypes` también proporciona un contenedor para la API C de Python como el objeto `ctypes.pythonapi`. Este objeto libera el bloqueo global del intérprete antes de llamar a una función, porque el bloqueo debe mantenerse al llamar al código del intérprete. Hay un constructor de tipo `py_object` que creará un puntero PyObject*. Un uso simple:

```
import ctypes

d = {}
ctypes.pythonapi.PyObject_SetItem(ctypes.py_object(d),
          ctypes.py_object("abc"),  ctypes.py_object(1))
# d is now {'abc', 1}.
```

No olvide utilizar `py_object()`; si lo omite, terminará con un error de segmentación.

`ctypes` existe desde hace tiempo, pero la gente sigue escribiendo y distribuyendo módulos de extensión codificados a mano porque no se puede confiar en que `ctypes` esté presente. Quizás los desarrolladores empiecen a escribir envoltorios de Python sobre una biblioteca a la que se accede a través de `ctypes` en lugar de módulos de extensión, ahora que `ctypes` está incluido en el núcleo de Python.

Ver también

https://web.archive.org/web/20180410025338/http://starship.python.net/crew/theller/ctypes/

La página web pre-stdlib ctypes, con un tutorial, referencia y preguntas frecuentes.

La documentación del módulo `ctypes`.

El paquete ElementTree

Un subconjunto de la biblioteca ElementTree de Fredrik Lundh para el procesamiento de XML se ha añadido a la biblioteca estándar como **`xml.etree`**. Los módulos disponibles son **`ElementTree`**, **`ElementPath`** y **`ElementInclude`** de ElementTree 1.2.6. También se incluye el módulo acelerador **`cElementTree`**.

El resto de esta sección proporcionará una breve descripción general del uso de ElementTree. La documentación completa de ElementTree está disponible en https://web.archive.org/web/20201124024954/http://effbot.org/zone/element-index.htm.

ElementTree representa un documento XML como un árbol de nodos de elementos. El contenido de texto del documento se almacena como los atributos **`text`** y **`tail`** de (Esta es una de las principales diferencias entre ElementTree y el Modelo de Objetos del Documento; en el DOM hay muchos tipos diferentes de nodo, incluyendo **`TextNode`**)

La función de análisis más utilizada es **`parse()`**, que toma una cadena (se supone que contiene un nombre de archivo) o un objeto similar a un archivo y devuelve una instancia de **`ElementTree`**:

```
from xml.etree import ElementTree as ET

tree = ET.parse('ex-1.xml')

feed = urllib.urlopen(
          'http://planet.python.org/rss10.xml')
tree = ET.parse(feed)
```

Una vez que tengas una instancia de **`ElementTree`**, puedes llamar a su método **`getroot()`** para obtener el nodo raíz de **`Element`**.

También hay una función **`XML()`** que toma un literal de cadena y devuelve un nodo **`Element`** (no un **`ElementTree`**). Esta función proporciona una forma ordenada de incorporar fragmentos XML, acercándose a la comodidad de un literal XML:

```
svg = ET.XML("""<svg width="10px" version="1.0">
             </svg>""")
svg.set('height', '320px')
svg.append(elem1)
```

Cada elemento XML admite algunos métodos de acceso tipo diccionario y otros tipo lista. Las operaciones tipo diccionario se utilizan para acceder a los valores de los atributos, y las operaciones tipo lista se utilizan para acceder a los nodos hijos.

Operación	**Resultado**
`elem[n]`	Devuelve el elemento hijo enésimo.
`elem[m:n]`	Devuelve la lista de m’s a n’s elementos hijos.
`len(elem)`	Devuelve el número de elementos hijos.
`list(elem)`	Devuelve la lista de elementos hijos.
`elem.append(elem2)`	Añade *elemento2* como hijo.
`elem.insert(index, elem2)`	Inserta *elemento2* en el lugar especificado.
`del elem[n]`	Elimina el elemento hijo enésimo.
`elem.keys()`	Devuelve la lista de nombres de atributos.
`elem.get(name)`	Devuelve el valor del atributo *nombre*.
`elem.set(nombre, valor)`	Establece el nuevo valor del atributo *nombre*.
`elem.attrib`	Recupera el diccionario que contiene los atributos.
`del elem.attrib[name]`	Borra el atributo *nombre*.

Los comentarios y las instrucciones de procesamiento también se representan como nodos **`Element`**. Para comprobar si un nodo es un comentario o unas instrucciones de procesamiento:

```
if elem.tag is ET.Comment:
    ...
elif elem.tag is ET.ProcessingInstruction:
    ...
```

Para generar una salida XML, debes llamar al método **`ElementTree.write()`**. Al igual que **`parse()`**, puede tomar una cadena o un objeto tipo archivo:

```
# Encoding is US-ASCII
tree.write('output.xml')

# Encoding is UTF-8
f = open('output.xml', 'w')
tree.write(f, encoding='utf-8')
```

(Atención: la codificación por defecto utilizada para la salida es ASCII. Para el trabajo general de XML, donde el nombre de un elemento puede contener caracteres Unicode arbitrarios, ASCII no es una codificación muy útil porque lanzará una excepción si el nombre de un elemento contiene cualquier carácter con valores superiores a 127. Por lo tanto, es mejor especificar una codificación diferente, como UTF-8, que puede manejar cualquier carácter Unicode)

Esta sección es sólo una descripción parcial de las interfaces de ElementTree. Por favor, lee la documentación oficial del paquete para más detalles.

Ver también

https://web.archive.org/web/20201124024954/http://effbot.org/zone/element-index.htm
Documentación oficial de ElementTree.

El paquete hashlib

Se ha añadido un nuevo módulo **`hashlib`**, escrito por Gregory P. Smith, para sustituir a los módulos **`md5`** y **`sha`**. **`hashlib`** añade compatibilidad con hashes seguros adicionales (SHA-224, SHA-256, SHA-384 y SHA-512). Cuando está disponible, el módulo utiliza OpenSSL para implementaciones rápidas y optimizadas de algoritmos para plataformas.

Los módulos antiguos **`md5`** y **`sha`** todavía existen como contenedores de hashlib para preservar la compatibilidad con versiones anteriores. La interfaz del nuevo módulo es muy similar a la de los módulos antiguos, pero no idéntica. La diferencia más significativa es que las funciones constructoras para crear nuevos objetos hash tienen nombres diferentes.

```
# Versiones anteriores
h = md5.md5()
h = md5.new()

# Nueva versión
h = hashlib.md5()
```

```
# Versiones anteriores
h = sha.sha()
h = sha.new()

# Nueva versión
h = hashlib.sha1()

# Hash que no estaban disponibles anteriormente
h = hashlib.sha224()
h = hashlib.sha256()
h = hashlib.sha384()
h = hashlib.sha512()

# Forma alternativa
h = hashlib.new('md5') # Proporcionar algoritmo como cac
```

Una vez que se ha creado un objeto hash, sus métodos son los mismos que antes: `actualizar(cadena)` convierte la cadena especificada en el estado de resumen actual, **`digest()`** y **`hexdigest()`** devuelven el valor de resumen como una cadena binaria o una cadena de dígitos hexadecimales, y **`copiar()`** devuelve un nuevo objeto hash con el mismo estado de resumen.

Ver también

La documentación del módulo **`hashlib`**.

El paquete sqlite3

El módulo pysqlite (https://www.pysqlite.org), un envoltorio para la base de datos incrustada de SQLite, se agregó a la biblioteca estándar con el nombre de paquete **`sqlite3`**.

SQLite es una biblioteca en C que proporciona una base de datos ligera basada en disco que no requiere un proceso de servidor independiente y permite acceder a la base de datos utilizando una variante no estándar del lenguaje de consulta SQL. Algunas aplicaciones pueden utilizar SQLite para el almacenamiento interno de datos. También es posible crear un prototipo de una aplicación utilizando SQLite y luego portar el código a una base de datos más grande como PostgreSQL u Oracle.

pysqlite fue escrito por Gerhard Häring y proporciona una interfaz SQL que cumple con la especificación DB-API 2.0 descrita por **PEP 249** [https://peps.python.org/pep-0249/].

Si estás compilando el código fuente de Python tú mismo, ten en cuenta que el árbol de código fuente no incluye el código de SQLite, sólo el módulo envolvente. Necesitarás tener las librerías y cabeceras de SQLite instaladas antes de compilar Python, y el proceso de construcción compilará el módulo cuando las cabeceras necesarias estén disponibles.

Para utilizar el módulo, primero hay que crear un objeto **`Connection`** que represente la base de datos. Aquí los datos se almacenarán en el archivo `/tmp/example`:

```
conn = sqlite3.connect('/tmp/ejemplo')
```

También puede suministrar el nombre especial `:memory:` para crear una base de datos en la RAM.

Una vez que tengas una **`Connection`**, puedes crear un objeto **`Cursor`** y llamar a su método **`execute()`** para realizar comandos SQL:

```
c = conn.cursor()

# Create table
c.execute('''create table stocks
(date text, trans text, symbol text,
 qty real, price real)''')

# Insert a row of data
c.execute("""insert into stocks
          values ('2006-01-05','BUY','RHAT',100,35.14)""
```

Normalmente tus operaciones SQL necesitarán usar valores de variables de Python. No deberías montar tu consulta usando las operaciones de cadena de Python porque hacerlo es inseguro; hace que tu programa sea vulnerable a un ataque de inyección SQL.

En su lugar, utilice la sustitución de parámetros de la DB-API. Ponga `?` como marcador de posición donde quiera usar un valor, y luego proporcione una tupla de valores como segundo argumento al método **`execute()`** del cursor. (Otros módulos de base de datos pueden utilizar un marcador de posición diferente, como `%s` o `:1`) Por ejemplo:

```
# Never do this -- insecure!
symbol = 'IBM'
c.execute("... where symbol = '%s'" % symbol)

# Do this instead
t = (symbol,)
c.execute('select * from stocks where symbol=?', t)

# Larger example
for t in (('2006-03-28', 'BUY', 'IBM', 1000, 45.00),
          ('2006-04-05', 'BUY', 'MSOFT', 1000, 72.00),
          ('2006-04-06', 'SELL', 'IBM', 500, 53.00),
         ):
    c.execute('insert into stocks values (?,?,?,?,?)', t
```

Para recuperar datos después de ejecutar una sentencia SELECT, puede tratar el cursor como un iterador, llamar al método **`fetchone()`** del cursor para recuperar una sola fila que coincida, o llamar a **`fetchall()`** para obtener una lista de las filas que coincidan.

Este ejemplo utiliza la forma del iterador:

```
>>> c = conn.cursor()
>>> c.execute('select * from stocks order by price')
>>> for row in c:
...    print row
...
(u'2006-01-05', u'BUY', u'RHAT', 100, 35.140000000000001
(u'2006-03-28', u'BUY', u'IBM', 1000, 45.0)
```

```
(u'2006-04-06', u'SELL', u'IBM', 500, 53.0)
(u'2006-04-05', u'BUY', u'MSOFT', 1000, 72.0)
>>>
```

Para más información sobre el dialecto SQL soportado por SQLite, consulte https://www.sqlite.org.

Ver también

https://www.pysqlite.org
La página web de pysqlite.

https://www.sqlite.org
La página web de SQLite; la documentación describe la sintaxis y los tipos de datos disponibles para el dialecto SQL soportado.

La documentación del módulo **`sqlite3`**.

PEP 249 [https://peps.python.org/pep-0249/] - Especificación de la API de la base de datos 2.0
PEP escrito por Marc-André Lemburg.

El paquete wsgiref

La Interfaz de Pasarela del Servidor Web (WSGI) v1.0 define una interfaz estándar entre los servidores web y las aplicaciones web de Python y se describe en **PEP 333** [https://peps.python.org/pep-0333/]. El paquete **`wsgiref`** es una implementación de referencia de la especificación WSGI.

El paquete incluye un servidor HTTP básico que ejecutará una aplicación WSGI; este servidor es útil para la depuración pero no está pensado para su uso en producción. La configuración de un servidor sólo requiere unas pocas líneas de código:

```
from wsgiref import simple_server

wsgi_app = ...
```

```
host = ''
port = 8000
httpd = simple_server.make_server(host, port, wsgi_app)
httpd.serve_forever()
```

Ver también

https://web.archive.org/web/20160331090247/http://wsgi.readthedocs.org/en/latest/
Un sitio web central para los recursos relacionados con WSGI.

PEP 333 [https://peps.python.org/pep-0333/] - Interfaz del servidor web Python v1.0
PEP escrito por Phillip J. Eby.

Cambios en la API de construcción y C

Los cambios en el proceso de construcción de Python y en la API de C incluyen:

- El árbol de fuentes de Python fue convertido de CVS a Subversion, en un complejo procedimiento de migración que fue supervisado y llevado a cabo de forma impecable por Martin von Löwis. El procedimiento se desarrolló como **PEP 347** [https://peps.python.org/pep-0347/].

- Coverity, una empresa que comercializa una herramienta de análisis de código fuente llamada Prevent, proporcionó los resultados de su examen del código fuente de Python. El análisis encontró alrededor de 60 errores que fueron rápidamente corregidos. Muchos de los errores eran problemas de recuento, que a menudo se producen en el código de gestión de errores. Consulte las estadísticas en https://scan.coverity.com.

- El mayor cambio en la API de C provino de **PEP 353** [https://peps.python.org/pep-0353/], que modifica el intérprete para usar una definición de tipo `Py_ssize_t` en lugar de int. Consulte la sección anterior PEP 353: Uso de ssize_t como tipo de índice para obtener una explicación de este cambio.

- El diseño del compilador de código de bytes ha cambiado mucho, ya no genera código de bytes recorriendo el árbol de análisis sintáctico. En su lugar, el árbol de análisis se convierte en un árbol de sintaxis abstracta (o AST), y es el árbol de sintaxis abstracta el que se recorre para producir el código de bytes.

  Es posible que el código Python obtenga objetos AST utilizando el built-in **`compile()`** y especificando `_ast.PyCF_ONLY_AST` como valor del parámetro *flags*:

  ```
  from _ast import PyCF_ONLY_AST
  ast = compile("""a=0
  for i in range(10):
      a += i
  """, "<string>", 'exec', PyCF_ONLY_AST)

  assignment = ast.body[0]
  for_loop = ast.body[1]
  ```

  Todavía no se ha escrito ninguna documentación oficial para el código AST, pero **PEP 339** [https://peps.python.org/pep-0339/] discute el diseño. Para empezar a conocer el código, lea la definición de los distintos nodos AST en `Parser/Python.asdl`. Un script de Python lee este archivo y genera un conjunto de definiciones de estructuras C en `Include/Python-ast.h`. Las funciones **`PyParser_ASTFromString()`** y **`PyParser_ASTFromFile()`**, definidas en `Include/pythonrun.h`, toman el código fuente de Python como entrada y devuelven la raíz de un AST que representa el contenido. Este AST puede convertirse en un objeto de código mediante **`PyAST_Compile()`**. Para más información, lea el código fuente, y luego haga preguntas en python-dev.

  El código de la AST fue desarrollado bajo la dirección de Jeremy Hylton, e implementado por (en orden alfabético)

Brett Cannon, Nick Coghlan, Grant Edwards, John Ehresman, Kurt Kaiser, Neal Norwitz, Tim Peters, Armin Rigo y Neil Schemenauer, además de los participantes en varios sprints de la AST en conferencias como la PyCon.

- Se aplicó el parche de Evan Jones a obmalloc, descrito por primera vez en una charla en la PyCon DC 2005. Python 2.4 asignaba objetos pequeños en arenas de 256K, pero nunca liberaba arenas. Con este parche, Python liberará arenas cuando estén vacías. El efecto neto es que en algunas plataformas, cuando se asignan muchos objetos, el uso de la memoria de Python puede realmente caer cuando se borran y la memoria puede ser devuelta al sistema operativo. (Implementado por Evan Jones, y reelaborado por Tim Peters)

  Tenga en cuenta que este cambio significa que los módulos de extensión deben ser más cuidadosos al asignar memoria. La API de Python tiene muchas funciones diferentes para asignar memoria que se agrupan en familias. Por ejemplo, `PyMem_Malloc()`, `PyMem_Realloc()`, y `PyMem_Free()` son una familia que asigna memoria en bruto, mientras que `PyObject_Malloc()`, `PyObject_Realloc()`, y `PyObject_Free()` son otra familia que se supone que se utiliza para crear objetos de Python.

  Anteriormente estas diferentes familias se reducían a las funciones **`malloc()`** y **`free()`** de la plataforma. Esto significaba que no importaba si te equivocabas y asignabas memoria con la función `PyMem` pero la liberabas con la función `PyObject`. Con los cambios de la versión 2.5 en obmalloc, estas familias hacen ahora cosas diferentes y los desajustes probablemente darán lugar a un fallo de seguridad. Deberías probar cuidadosamente tus módulos de extensión C con Python 2.5.

- Los tipos de conjuntos incorporados tienen ahora una API oficial en C. Llame a `PySet_New()` y `PyFrozenSet_New()` para crear un nuevo conjunto, `PySet_Add()` y `PySet_Discard()` para añadir y eliminar elementos, y `PySet_Contains()` y `PySet_Size()` para examinar el estado del conjunto. (Contribución de Raymond Hettinger)

- El código C puede ahora obtener información sobre la revisión exacta del intérprete de Python llamando a la función **`Py_GetBuildInfo()`** que devuelve una cadena de información de compilación como esta `"trunk:45355:45356M, Apr 13 2006, 07:42:19"`. (Contribuido por Barry Warsaw.)

- Two new macros can be used to indicate C functions that are local to the current file so that a faster calling convention can be used. **`Py_LOCAL`** declares the function as returning a value of the specified *type* and uses a fast-calling qualifier. **`Py_LOCAL_INLINE`** does the same thing and also requests the function be inlined. If macro **`PY_LOCAL_AGGRESSIVE`** is defined before `python.h` is included, a set of more aggressive optimizations are enabled for the module; you should benchmark the results to find out if these optimizations actually make the code faster. (Contributed by Fredrik Lundh at the NeedForSpeed sprint.)

- `PyErr_NewException(name, base, dict)` ahora puede aceptar una tupla de clases base como su argumento *base*. (Contribuido por Georg Brandl.)

- La función **`PyErr_Warn()`** para emitir avisos está ahora obsoleta en favor de `PyErr_WarnEx(category, message, stacklevel)` que permite especificar el número de marcos de pila que separan esta función y la que la llama. Un *stacklevel* de 1 es la función que llama a **`PyErr_WarnEx()`**, 2 es la función que está por encima, y así sucesivamente. (Añadido por Neal Norwitz.)

- El intérprete de CPython sigue estando escrito en C, pero el código ahora puede ser compilado con un compilador de C++ sin errores. (Implementado por Anthony Baxter, Martin von Löwis, Skip Montanaro)

- Se ha eliminado la función **`PyRange_New()`**. Nunca se documentó, nunca se utilizó en el código del núcleo, y tenía una comprobación de errores peligrosamente laxa. En el improbable caso de que sus extensiones la utilizaran, puede sustituirla por algo como lo siguiente:

```
range = PyObject_CallFunction((PyObject*) &PyRange_
                              start, stop, step);
```

Cambios específicos en los puertos

- MacOS X (10.3 y superior): la carga dinámica de módulos utiliza ahora la función **`dlopen()`** en lugar de funciones específicas de MacOS.
- MacOS X: se ha añadido una opción **`--enable-universalsdk`** al script **configure** que compila el intérprete como un binario universal capaz de funcionar tanto en procesadores PowerPC como Intel. (Contribución de Ronald Oussoren; bpo-2573 [https://bugs.python.org/issue?@action=redirect&bpo=2573].)
- Windows: `.dll` ya no se admite como extensión de nombre de archivo para los módulos de extensión. `.pyd` es ahora la única extensión de nombre de archivo que se buscará.

Adaptación a Python 2.5

Esta sección enumera los cambios descritos anteriormente que pueden requerir cambios en su código:

- ASCII es ahora la codificación por defecto para los módulos. Ahora es un error de sintaxis si un módulo contiene literales de cadena con caracteres de 8 bits pero no tiene una declaración de codificación. En Python 2.4 esto provocaba una advertencia, no un error de sintaxis.
- Anteriormente, el atributo **`gi_frame`** de un generador era siempre un objeto frame. Debido a los cambios de **PEP 342** [https://peps.python.org/pep-0342/] descritos en la sección PEP 342: Nuevas funciones del generador, ahora es posible que **`gi_frame`** sea `None`.
- Una nueva advertencia, **`UnicodeWarning`**, se lanza cuando se intenta comparar una cadena Unicode y una cadena de 8 bits que no puede ser convertida a Unicode utilizando la

codificación ASCII por defecto. Anteriormente estas comparaciones lanzaban una excepción **`UnicodeDecodeError`**.

- Biblioteca: el módulo **`csv`** es ahora más estricto con los campos citados en varias líneas. Si sus archivos contienen nuevas líneas incrustadas dentro de los campos, la entrada debe dividirse en líneas de manera que se conserven los caracteres de nueva línea.
- Biblioteca: la función **`format()`** del módulo **`locale`** aceptaba antes cualquier cadena siempre que no apareciera más de un especificador %char. En Python 2.5, el argumento debe ser exactamente un especificador %char sin texto alrededor.
- Biblioteca: Los módulos **`pickle`** y **`cPickle`** ya no aceptan un valor de retorno de `None` del método **`__reduce__()`**; en su lugar, el método debe devolver una tupla de argumentos. Los módulos tampoco aceptan el parámetro de palabra clave *bin*, que ya no se utiliza.
- Biblioteca: Las clases **`SimpleXMLRPCServer`** y **`DocXMLRPCServer`** ahora tienen un atributo **`rpc_paths`** que restringe las operaciones XML-RPC a un conjunto limitado de rutas URL; el valor predeterminado es permitir solo `'/'` y `'/RPC2'`. Si se configura **`rpc_paths`** en `None` o en una tupla vacía, se deshabilita esta verificación de ruta.
- API de C: muchas funciones ahora usan **`Py_ssize_t`** en lugar de int para permitir el procesamiento de más datos en máquinas de 64 bits. Es posible que el código de extensión deba realizar el mismo cambio para evitar advertencias y admitir máquinas de 64 bits. Consulte la sección anterior PEP 353: Uso de ssize_t como tipo de índice para obtener una explicación de este cambio.
- API C: Los cambios en obmalloc significan que debe tener cuidado de no mezclar el uso de las familias de funciones `PyMem_*` y `PyObject_*`. La memoria asignada con la función `*_Malloc` de una familia, debe ser liberada con la función `*_Free` de la familia correspondiente.

Agradecimientos

El autor desea agradecer a las siguientes personas sus sugerencias, correcciones y ayuda en varios borradores de este artículo: Georg Brandl, Nick Coghlan, Phillip J. Eby, Lars Gustäbel, Raymond Hettinger, Ralf W. Grosse-Kunstleve, Kent Johnson, Iain Lowe, Martin von Löwis, Fredrik Lundh, Andrew McNamara, Skip Montanaro, Gustavo Niemeyer, Paul Prescod, James Pryor, Mike Rovner, Scott Weikart, Barry Warsaw, Thomas Wouters.

Qué hay de nuevo en Python 2.4

Autor:

A.M. Kuchling

Este artículo explica las nuevas funcionalidades en Python 2.4.1 lanzadas el 30 de marzo de 2005.

Python 2.4 es un lanzamiento de término medio. Esto significa que no se han introducido grandes cambios como en la versión 2.2 de Python pero si son más significativas que la versión 2.3. Lo más relevante de estas nuevas funcionalidades son los decoradores y generadores de expresiones, la mayor parte del resto de los cambios son a la librería estándar.

De acuerdo al registro de cambios CVS , han sido aplicados 481 parches y 502 bugs han sido corregidos entre las versiones 2.3 y 2.4 de Python. Es posible que ambas cifras no concuerden con la realidad, podrían estar subestimadas.

Este artículo no pretende proveer una especificación completa y detallada de cada funcionalidad nueva, en vez de esto, se provee una breve descripción de las nuevas características. Para obtener más información se debe consultar la documentación oficial de Python 2.4, tal como la librería de referencia de Python y el manual de referencia de Python. Otra fuente de información es PEP (guía de estilo-mejoras de python) para buscar información sobre la implementación y fundamentación del diseño.

PEP 218: Objetos conjunto integrados

Python 2.3 introdujo el módulo **`sets`** , implementaciones del lenguaje C de tipos de datos han sido agregados al core de Python como dos nuevos tipos integrados. `set (iterable)` y `frozenset (iterable)`. Ellos proveen operaciones de alto rendimiento para pruebas de pertenencia, eliminar datos duplicados de una secuencia y operaciones matemáticas como: uniones, intersecciones, diferencias y diferencias simétricas.

```
>>> a = set('abracadabra') # formar un conjunto a partir
>>> 'z' en a # prueba rápida de pertenencia
Falso
>>> a # letras únicas en a
set(['a', 'r', 'b', 'c', 'd'])
>>> ''.join(a) # convertir de nuevo a una cadena
'arbcd'

>>> b = set('alacazam') # formar un segundo conjunto
>>> a - b # letras en a pero no en b
set(['r', 'd', 'b'])
>>> a | b # letras en a o b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b # letras en a y b
set(['a', 'c'])
>>> a ^ b # letras en a o b pero no en ambos
set(['r', 'd', 'b', 'm', 'z', 'l'])

>>> a.add('z') # agrega un nuevo elemento
>>> a.update('wxy') # agrega varios elementos nuevos
>>> a
set(['a', 'c', 'b', 'd', 'r', 'w', 'y', 'x', 'z'])
>>> a.remove('x') # saca un elemento
>>> a
set(['a', 'c', 'b', 'd', 'r', 'w', 'y', 'z'])
```

La función de conjuntos integrados **`frozenset()`** es una versión inmutable de **`set()`**. Como es inmutable y hashable (su valor no cambia), se puede usar como una llave de diccionario o como parte de otro set.

El módulo set **`sets`** se encuentra en la librería estándar y puede ser útil si se desea usar una subclase de **`Set`** o **`ImmutableSet`** No existen planes de deprecación del módulo.

Ver también

PEP 218 [https://peps.python.org/pep-0218/] - Agregando tipo de objeto integrado set

Originalmente propuesto por Greg Wilson e implementado finalmente por Raymond Hettinger.

PEP 237: Unificando enteros largos y enteros

El prolongado proceso de transición para este PEP que comenzó en Python 2.2 ha dado un paso más adelante en Python 2.4. En la versión 2.3 ciertas operaciones de enteros podían comportarse diferente después de lanzar la unificación del tipo de dato int-long entregando alertas `FutureWarning` y retornando valores limitados a 32 o 64 bits dependiendo de su plataforma. En la versión 2.4 estas expresiones ya no producen alertas y en vez de eso entrega un resultado diferente que suele ser un entero largo.

Las expresiones problemáticas son principalmente desplazamientos a la izquierda y constantes largas de hexadecimales y octales. Por ejemplo, `2 << 32` entrega una alerta en la versión 2.3 y evalúa 0 en plataformas 32-bit. Ahora en Python 2.4 la expresión retorna la respuesta correcta, 8589934592.

Ver también

PEP 237 [https://peps.python.org/pep-0237/] - Unificando enteros largos y enteros

PEP original escrito por Moshe Zadka y GvR. Los cambios a la versión 2.4 fueron implementados por Kalle Svensson.

PEP 289: Expresiones generadoras

La función de iterador introducida en Python 2.2 y el módulo `itertools` facilitan la escritura de programas que recorren grandes conjuntos de datos sin tener todo el conjunto de datos en la memoria al mismo tiempo. Las listas de comprensión no encajan muy bien en esta imagen porque producen un objeto de lista de Python que contiene todos los elementos. Esto inevitablemente lleva todos los objetos a la memoria, lo que puede ser un problema si su conjunto de datos es muy grande. Al tratar de escribir un programa de estilo funcional, sería natural escribir algo como:

```
links = [enlace para enlace en get_all_links() si no es
para enlace en enlaces:
...
```

en vez de

```
para el enlace en get_all_links():
si link.followed:
continúa
...
```

La primera forma es mas concisa y quizás mas legible pero en caso de trabajar con un gran número largo de objetos podría ser necesario el escribir una segunda línea para evitar tener los objetos enlazados en memoria al mismo tiempo.

Las expresiones generadoras trabajan de forma similar a las listas por comprensión pero no materializan la lista completa, en vez de eso se crea un generador que retornará los elementos uno a uno. El siguiente ejemplo podría ser escrito como:

```
links = (enlace para enlace en get_all_links() si no se
para enlace en enlaces:
...
```

Los generadores de expresiones siempre tienen que ser escritos dentro de paréntesis como en el ejemplo descrito arriba. Los paréntesis señalan una llamada a la función que también se puede contar, entonces si se necesita crear un iterador que inmediatamente pasara a una función se podría escribir:

```
imprimir suma(obj.count para obj en list_all_objects())
```

Las expresiones generadoras difieren de las listas de comprensión por pequeñas diferencias. La mas notable es que la variable del loop (*obj* en el ejemplo de arriba) no es accesible fuera de la expresión generadora. Las listas de comprensión dejan la variable asignada a su último valor, en versiones futuras de Python esto será cambiado en lo que respecta a hacer que las listas de comprensión coincidan con las expresiones de los generadores.

Ver también

PEP 289 [https://peps.python.org/pep-0289/] - Expresiones generadoras

Propuesto por Raymond Hettinger e implementada por Jiwon Seo con los primeros esfuerzos dirigidos por Hye-Shik Chang.

PEP 292: Sustituciones simples de cadenas de caracteres

Algunas nuevas clases en la librería estándar proveen un mecanismo alternativo para sustituir variables dentro de cadenas de caracteres, este estilo de sustituciones puede ser mejor por aplicaciones donde usuarios sin experiencia necesitan editar plantillas.

La manera usual de sustituir variables por el nombre es con el operador `%`:

```
>>> '%(page)i: %(title)s' % {'page':2, 'title': 'Lo mejo
'2: Lo mejor de los tiempos'
```

Cuando se escriben plantillas con cadenas de caracteres puede ser muy fácil olvidar el uso de `i` or `s` después de cerrar paréntesis. Esto no es un problema grande si se está dentro de un módulo de Python pues al correr el código se lanza el siguiente mensaje de error «Unsupported format character» **`ValueError`** y el problema se resuelve. Sin embargo, tengamos en cuenta una aplicación como Mailman donde una plantilla de caracteres o traducciones son editadas por usuarios que no están familiarizados con Python. La sintaxis de como formatear las cadenas de caracteres es complicada de explicar para ciertos usuarios y si ellos cometen un error es una dificultad el entregar retroalimentación y ayuda.

PEP 292 agrega una clase **`Template`** al módulo `string` al módulo que usa `$` para indicar una sustitución:

```
>>> import string
>>> t = string.Template('$page: $title')
>>> t.substitute({'page':2, 'title': 'Lo mejor de los ti
'2: Lo mejor de los tiempos'
```

Si una llave falta en el diccionario, el método **`substitute()`** lanzará una `KeyError`. También el método **`safe_substitute()`** que ignora las llaves faltantes:

```
>>> t = string.Template('$página: $título')
>>> t.safe_substitute({'página':3})
'3: $título'
```

Ver también

PEP 292 [https://peps.python.org/pep-0292/] - Sustituciones simples de cadenas de caracteres

Escrito e implementado por Barry Warsaw.

PEP 318: Decoradores para funciones y métodos

Python 2.2 extendió el modelo de objeto de Python añadiendo métodos estáticos y métodos de clase pero no se extendió la sintaxis de Python para proveer alguna nueva forma de definir métodos estáticos o de clase. En vez de eso se tenía que escribir la palabra clave `def` de manera usual y pasar el método resultante a la `staticmethod()` o a la `classmethod()` que podía agrupar la función como método de un nuevo tipo. El código podría verse como esto:

```
clase C:
def meth (cls):
...
```

```
meth = classmethod(meth) # Volver a vincular el nombre a
```

Si el método era muy largo, era muy probable el perder o olvidar la invocación `classmethod()` dentro del cuerpo de la función.

La intención era siempre agregar alguna sintaxis que hicieran las definiciones mas entendibles pero al tiempo del lanzamiento de la versión 2.2 una buena sintaxis no era clara. Al día de hoy una buena sintaxis aun no es clara pero los usuarios están preguntando por formas más fáciles de acceder a las características, entonces una nueva funcionalidad sintáctica ha sido añadida para satisfacer esta necesidad.

La nueva característica es llamada decoradores de funciones. El nombre proviene de la idea de que `classmethod()`, `staticmethod()` son amigos y guardan información adicional en un objeto de una función. ellos están decorando funciones con mas detalles.

La notación proviene de Java y usa el `'@'` símbolo arroba como un indicador. Usando la nueva sintaxis el ejemplo descrito arriba podría ser escrito de la siguiente forma:

```
clase C:

@classmethod
def meth (cls):
...
```

El `@classmethod` es un atajo para `meth=classmethod(meth)`. De forma general si se tiene lo siguiente:

```
@A
@B
@C
def f():
...
```

Es el equivalente para el siguiente código de pre decorador:

```
definición f(): ...
f = A(B(C(f)))
```

Los decoradores deben agregarse en una línea antes de la definición de la función, un decorador por línea y no puede estar en la misma línea donde se usa la palabra def para comenzar una función, por ejemplo `@A def f(): ...` es incorrecto. Solo se puede decorar una definición de función tanto al nivel de módulo o dentro de una clase pero no puedes decorar definiciones de clase.

Un decorador es simplemente una función que toma la función a ser decorada como un argumento y retorna tanto la misma función o un nuevo objeto. El valor de retorno del decorador no necesita ser llamado (como típicamente se cree) si no que los decoradores podrán ser aplicados al resultado. Es fácil escribir sus propios decoradores. El siguiente ejemplo agrega un atributo a la función objetiva:

```
>>> def deco(func):
... func.attr = 'decorado'
... return func
...
>>> @deco
... def f(): pass
...
>>> f
<función f en 0x402ef0d4>
>>> f.attr
'decorado'
>>>
```

Como un ejemplo un poco mas realista, el siguiente decorador revisa si el argumento entregado es un entero:

```
def require_int (función):
def contenedor (arg):
assert isinstance(arg, int)
return func(arg)

return contenedor
```

```
@require_int
def p1 (arg):
print arg

@require_int
def p2(arg):
print arg*2
```

Un ejemplo en PEP 318 [https://peps.python.org/pep-0318/] contiene una versión mas elegante de esta idea donde deja que usted elija específicamente el tipo y revisa el tipo de retorno.

Los decoradores de funciones pueden tomar argumentos. Si los argumentos son provistos, el decorador de la función es llamado con solo esos argumentos y retorna un nuevo decorador de función, esta función debe tomar una función sola y retorna una función como previamente se ha descrito. En otras palabras `@A @B @C(args)` comienza:

```
def f(): ...
_deco = C(argumentos)
f = A(B(_deco(f)))
```

Conseguir que esto resulte correcto puede ser un acertijo pero tampoco es muy dificultoso.

Un pequeño cambio relacionado hace que el atributo **`func_name`** de las funciones sea escribible. Este atributo se utiliza para mostrar los nombres de las funciones en los seguimientos, por lo que los decoradores deben cambiar el nombre de cualquier función nueva que se construya y devuelva.

Ver también

PEP 318 [https://peps.python.org/pep-0318/] Decoradores para funciones, métodos y clases

Escrito por Kevin D.Smith, Jim Jewett y Skip Montanaro. Varias personas escribieron parches sobre la implementación de los decoradores de funciones, pero el único que fue chequeado fue #979728 por Mark Russell.

https://wiki.python.org/moin/PythonDecoratorLibrary
Esta pagina de wiki contiene múltiples ejemplos de decoradores.

PEP 322: Iteración inversa

Una nueva función de conjuntos integrados, `reversed(seq)` toma una secuencia y retorna un iterador que recorre los elementos de la secuencia en orden inverso.:

```
>>> para i en reversed(xrange(1,4)):
... imprimir i
...
3
2
1
```

Comparado con el uso tradicional de la segmentación tal como `range(1,4)[::-1]` la función **`reversed()`** es fácil de leer, se ejecuta mas rápido y usa sustancialmente menos memoria.

Señalar que la función **`reversed()`** solo acepta secuencias, no iteradores arbitrarios. Si se quiere revertir un iterador, primero se debe convertir a una lista usando la función **`list()`**.:

```
>>> input = open('/etc/passwd', 'r')
>>> for line in reversed(list(input)):
... print line
...
root:*:0:0:Administrador del sistema:/var/root:/bin/tcsh
...
```

Ver también

PEP 322 [https://peps.python.org/pep-0322/] - Iteración inversa
Escrita e implementada por Raymond Hettinger.

PEP 324: Nuevo módulo de subproceso

La biblioteca estándar proporciona varias formas de ejecutar un subproceso, ofreciendo distintas funciones y distintos niveles de complejidad. `os.system(command)` es fácil de usar, pero lento (ejecuta un proceso de shell que ejecuta el comando) y peligroso (hay que tener cuidado con el escape de los metacaracteres del shell). El módulo **`popen2`** ofrece clases que pueden capturar la salida estándar y el error estándar del subproceso, pero la nomenclatura es confusa. El módulo **`subprocess`** soluciona este problema, proporcionando una interfaz unificada que ofrece todas las funciones que se pueden necesitar.

En vez de **`popen2`** que es una colección de clases el **`subprocess`** contiene una sola clase llamada **`subprocess.Popen`** cuyo constructor soporta un número de diferentes argumentos de palabra clave:

```
clase Popen(args, bufsize=0, ejecutable=Ninguno,
stdin=Ninguno, stdout=Ninguno, stderr=Ninguno,
preexec_fn=Ninguno, close_fds=False, shell=False,
cwd=Ninguno, env=Ninguno, universal_newlines=False,
startupinfo=Ninguno, creationflags=0):
```

args es comúnmente una secuencia de cadenas de caracteres que podrán ser los argumentos del programa ejecutado como un subproceso. (Si el argumento *shell* es verdadero, *args* puede ser una cadena de caracteres que serán pasados directamente a la shell para su interpretación, tal como **`os.system()`** lo hace.)

stdin, *stdout*, y *stderr* especifican que flujos de entrada, salida y error serán utilizados. Se puede proceder un objeto de archivo o un archivo de descripción o se puede usar la constante `subprocess.PIPE` para crear un enlace entre el subproceso y el proceso principal.

El constructor tiene un número de opciones útiles:

- *close_fds* requiere que todos los descriptores de archivos sean cerrados antes de correr el subproceso.

- *cwd* especifica el directorio operativo donde el subproceso será ejecutado (por defecto, cualquiera que sea el directorio principal)
- *env* es un diccionario donde se especifican las variables de entorno.
- *preexec_fn* es una función que es llamada antes de que el proceso hijo sea llamado.
- *universal_newlines* abre el flujo de entrada y salida del hijo usando la característica de Python universal newlines .

Una vez que se ha creado la instancia de la **`Popen`** se puede llamar al método **`wait()`** para poner en pausa hasta que el subproceso ha terminado, el método **`poll()`** comprueba si se ha salido sin pausar o el uso de `communicate(data)` envía la cadena de caracteres *data* a la entrada estándar del subproceso. Y el uso de `communicate(data)` que lee cualquier dato que el subproceso ha enviado a la salida estándar, retornando una tupla `(stdout_data, stderr_data)`.

`call()` es un atajo que pasa estos argumentos a lo largo del constructor de la clase **`Popen`**, espera por el comando para completar la secuencia y retorna el código de estatus del subproceso. Esto puede servir como un análogo mas seguro de **`os.system()`**:

```
sts = subprocess.call(['dpkg', '-i', '/tmp/new-package.
if sts == 0:
# Éxito
...
else:
# dpkg devolvió un error
...
```

El comando es invocado sin el uso del interprete de comandos. Si se desea usar el intérprete se puede agregar `shell=True` como un argumento de palabra clave y proveer una cadena de caracteres en vez de una secuencia:

```
sts = subproceso.call('dpkg -i /tmp/nuevo-paquete.deb',
```

La guía de estilo de Python toma varios ejemplos de línea de comandos y código python y muestra como estos son traducidos en el código python `subprocess`. Leer esta sección de la guía de estilo es altamente recomendado.

Ver también

PEP 324 [https://peps.python.org/pep-0324/] - subprocess - Nuevo módulo de procesos
Escrito e implementado por Peter Åstrand, con asistencia de Frederik Lundh y otros.

PEP 327: Tipo de dato decimal

Python siempre ha soportado números de punto flotante (FP), basados en el tipo subyacente C double, como un tipo de datos. Sin embargo, aunque la mayoría de los lenguajes de programación proporcionan un tipo de punto flotante, muchas personas (incluso los programadores) no saben que los números de punto flotante no representan ciertas fracciones decimales con precisión. El nuevo tipo `Decimal` puede representar estas fracciones con precisión, hasta un límite de precisión especificado por el usuario.

¿Por qué se necesita Decimal?

Las limitaciones surgen de la representación usada para los números de punto flotante. Los números de punto flotante están hechos de tres componentes:

- El signo, el cual es positivo o negativo.
- El significando o mantisa el cual es el un dígito binario seguido de su parte fraccional. Por ejemplo, `1.01` en notación base 2 es `1 + 0/2 + 1/4` o 1.25 en notación decimal.
- El exponente, que indica dónde se encuentra el punto decimal en el número representado.

Por ejemplo, el número 1.25 tiene un signo positivo, una mantisa de valor 1.01 (en binario) y un exponente de 0 (el punto decimal no necesita ser desplazado. El números 5 tiene el mismo signo y significando pero, el exponente es 2 porque el significando es multiplicado por 4 (2 elevado a la potencia del exponente 2), 1.25 * 4 es igual a 5.

Los sistemas modernos generalmente brindan soporte de coma flotante que se ajusta a un estándar llamado IEEE 754. El tipo double de C generalmente se implementa como un número IEEE 754 de 64 bits, que usa 52 bits de espacio para la mantisa. Esto significa que los números solo se pueden especificar con 52 bits de precisión. Si está tratando de representar números cuya expansión se repite sin cesar, la expansión se corta después de 52 bits. Desafortunadamente, la mayoría del software necesita producir resultados en base 10, y las fracciones comunes en base 10 a menudo son decimales repetidos en binario. Por ejemplo, 1.1 decimal es binario `1.0001100110011 ...`; .1 = 1/16 + 1/32 + 1/256 más un número infinito de términos adicionales. IEEE 754 tiene que cortar ese decimal repetido infinitamente después de 52 dígitos, por lo que la representación es un poco inexacta.

Algunas veces se puede ver esta inexactitud cuando el número es impreso por pantalla:

```
>>> 1.1
1.1000000000000001
```

La inexactitud no siempre es visible cuando se imprime el número porque el punto flotante a decimal y conversión a cadena de texto es entregada por la librería C y la mayoría de las librerías C tratan de producir una salida sensible. Incluso si no es mostrado, la inexactitud sigue ahí y las operaciones subsecuentes pueden agrandar el error.

Para muchas aplicaciones esto no importa. Si estoy trazando puntos y mostrándolos en el monitor la diferencia entre 1.1 y 1.1000000000000001 es muy pequeña para ser visible. Los reportes a menudo limitan la salida de un cierto número de decimales y si se redondea el número a dos o tres e incluso a 8 decimales, el error nunca aparece. Sin embargo, para las aplicaciones donde esto importa, es una cantidad de trabajo importante el personalizar los cálculos aritméticos.

Es por eso que el tipo de clase **`Decimal`** fue creada.

El tipo de clase **`Decimal`**

Un nuevo módulo, **`decimal`**, fue añadido a la librería estándar de Python. Esta contiene dos clases, **`Decimal`** y **`Context`**. Las instancias de la clase decimal **`Decimal`** representan números y la instancia de la **`Context`** son usadas para condensar varios ajustes como la precisión y el modo de redondeo por defecto.

Las instancias de la clase **`Decimal`** son inmutables como los números enteros y números de punto flotante en Python, una vez han sido creadas no se puede cambiar el valor de lo que la instancia representa. Las instancias de la **`Decimal`** pueden ser creados desde números enteros o cadenas de texto:

```
>>> importar decimal
>>> decimal.Decimal(1972)
Decimal("1972")
>>> decimal.Decimal("1.1")
Decimal("1.1")
```

También se pueden proveer tuplas que contentan el signo, la mantisa representada como una tupla de números decimales y el exponente:

```
>>> decimal.Decimal((1, (1, 4, 7, 5), -2))
Decimal("-14,75")
```

Precaución: El signo bit es un valor booleano entonces 0 es positivo y 1 es negativo.

Convertir desde números de punto flotante posee un pequeño problema, el punto flotante representa 1.1 y se convierte en número decimal como 1.1 o por 1.1 mas cualquier inexactitud que se introduzca? La decisión fue evitar el problema y dejar tal conversión fuera de la API. En su lugar se debe convertir el número de punto flotante en una cadena con la precisión deseada y pasar la cadena al constructor **`Decimal`** :

```
>>> f = 1,1
>>> decimal.Decimal(str(f))
Decimal("1,1")
>>> decimal.Decimal('%.12f' % f)
Decimal("1,100000000000")
```

Una vez que se tiene la instancia de la clase **`Decimal`** se pueden realizar las operaciones matemáticas de rigor. Una limitación, la exponenciación requiere un exponente expresado en tipo de dato entero:

```
>>> a = decimal.Decimal('35,72')
>>> b = decimal.Decimal('1,73')
>>> a+b
Decimal("37,45")
>>> a-b
Decimal("33,99")
>>> a*b
Decimal("61.7956")
>>> a/b
Decimal("20.64739884393063583815028902")
>>> a XASDF00b
Traceback (most recent call last):
  ...
decimal.InvalidOperation: x ** (no entero)
```

Se puede combinar la instancia de la clase **`Decimal`** con enteros pero no con números de punto flotante:

```
>>> a + 4
Decimal("39.72")
>>> a + 4.5
Traceback (última llamada más reciente):
...
TypeError: Solo puede interactuar con Decimal con tipos
>>>
```

Los números en la clase **`Decimal`** pueden ser usados con los módulos **`math`** y **`cmath`** pero es preciso aclarar que ellos serán inmediatamente convertidos a números de punto flotante antes de realizar la operación, resultando en una posible perdida de precisión y exactitud. También obtendrá un número de punto flotante y no una clase **`Decimal`**:

```
>>> importar matemáticas, cmath
>>> d = decimal.Decimal('123456789012.345')
>>> matemáticas.sqrt(d)
351364.18288201344
>>> cmath.sqrt(-d)
351364.18288201344j
```

Las instancias de la clase **`Decimal`** poseen un método **`sqrt()`** que retorna una clase **`Decimal`** pero si se necesita otro tipo de cosas como funciones trigonométricas se tendrían que implementar:

```
>>> d.sqrt()
Decimal("351364.1828820134592177245001")
```

El tipo de clase `Context`

Las instancias de la **`Context`** encapsulan varios ajustes para operaciones con decimales:

- atributo **`prec`** es la precisión del número de decimales.
- El atributo **`rounding`** especifica el modo de redondeo. El módulo **`decimal`** tiene constantes para varias posibilidades, **`ROUND_DOWN`**, **`ROUND_CEILING`**, **`ROUND_HALF_EVEN`** y varias otras.
- El atributo **`traps`** es un diccionario que especifica lo que sucede al encontrar ciertas condiciones de error: Ya sea una excepción lanzada o un valor es retornado. algunos ejemplos de errores son las divisiones por cero, perdida de precisión o desbordamiento.

Hay un contexto predeterminado local de subprocesos disponible llamando a la función **`getcontext()`**, se puede cambiar las propiedades de este contexto para alterar la precisión por defecto, redondear o manejar trampas. el siguiente ejemplo muestra el efecto de cambiar la precisión del contexto por defecto:

```
>>> decimal.getcontext().prec
```

```
28
>>> decimal.Decimal(1) / decimal.Decimal(7)
Decimal("0,142857142857142857142857142857142857")
>>> decimal.getcontext().prec = 9
>>> decimal.Decimal(1) / decimal.Decimal(7)
Decimal("0,142857143")
```

Las acciones por defecto para condiciones de error se puede seleccionar, ya sea que el módulo pueda retornar un valor especial como un infinito o un valor no numérico NaN o se pueden lanzar excepciones:

```
>>> decimal.Decimal(1) / decimal.Decimal(0)
Traceback (última llamada más reciente):
...
decimal.DivisionByZero: x / 0
>>> decimal.getcontext().traps[decimal.DivisionByZero] =
>>> decimal.Decimal(1) / decimal.Decimal(0)
Decimal("Infinito")
>>>
```

La instancia de la clase **`Context`** también tiene varios métodos para formatear números tales como **`to_eng_string()`** y **`to_sci_string()`**.

Para mas información, revisar la documentación para el módulo **`decimal`** el cual incluye un tutoría de inicio rápido y una referencia.

Ver también

PEP 327 [https://peps.python.org/pep-0327/] - Tipo de dato decimal

Escrito por Facundo Batista e implementado por Facundo Batista, Eric Price, Raymond Hettinger, Aahz y Tim Peters.

http://www.lahey.com/float.htm [https://web.archive.org/web/20230604072523/http://www.lahey.com/float.htm]

El articulo usa código Fortran para ilustrar varios de los problemas que la inexactitud de los puntos flotantes pueden causar.

https://speleotrove.com/decimal/
Una descripción de una representación basada en decimales. Esta representación ha sido propuesta como un estándar y subyace al nuevo tipo de dato decimal en Python. La mayor parte de este material fue escrita por Mike Cowlishaw, diseñador del lenguaje Rexx.

PEP 328: Importaciones multilínea

Un pequeño cambio en el lenguaje ha sido una modificación a una sintaxis confusa que ha sido cambiada para hacer mas fácil el agregar varios nombres en un módulo. En una declaración `from module import names`, *names* es una secuencia de nombres separados por comas. Si la sentencia es muy larga, se puede ya sea escribir múltiples sentencias import desde el mismo módulo o se pueden usar barras invertidas para escapar de los términos de línea como esto:

```
de SimpleXMLRPCServer importar SimpleXMLRPCServer,\
SimpleXMLRPCRequestHandler,\
CGIXMLRPCRequestHandler,\
resolve_dotted_attribute
```

La sintaxis cambió en Python 2.4 y es simplemente el permitir poner los nombres dentro de paréntesis. Python ignora nuevas líneas dentro de una expresión entre paréntesis por lo que las barras invertidas ya no se necesitan:

```
from SimpleXMLRPCServer import (SimpleXMLRPCServer,
                                SimpleXMLRPCRequestHandl
                                CGIXMLRPCRequestHandler,
                                resolve_dotted_attribute
```

La convención de estilo también propone que todas las declaraciones usando `import` son importaciones absolutas, con el siguiente carácter `.` indica una importación relativa. Esta parte de la convención de estilo no fue implementada para Python 2.4 pero fue completada para la versión 2.5.

Ver también

PEP 328 [https://peps.python.org/pep-0328/] - Importaciones multi línea y absolutas/relativas
Escrita por Aahz. Importaciones multi línea fueron implementadas por Dima Dorfman.

PEP 331: Conversiones locales-independientes números flotantes/ cadenas de texto

Los módulos `locale` deja que el software de Python seleccione varias conversiones y despliegue convenciones que están localizadas en un país particular o lenguaje. Sin embargo, el módulo es cuidadoso en no cambiar el local numérico porque la implementación de Python requería que la configuración regional numérica permaneciera establecida en la configuración regional `'C'`. A menudo, esto se debía a que el código utilizaba la función **`atof()`** de la biblioteca C.

No configurar la configuración regional numérica causaba problemas para las extensiones que usaban librerías externas de C, sin embargo, porque no configuraban la correcta configuración. El ejemplo mas patente fue GTK+, cuya interfaz de usuario no mostraba números en la ubicación actual.

La solución descrita en la convención de estilo es añadir tres nuevas funciones a la API de python que ejecuta solo conversiones a ASCII, ignorando las configuraciones locales:

- `PyOS_ascii_strtod(str, ptr)` y `PyOS_ascii_atof(str, ptr)` convierten una cadena en C double.
- `PyOS_ascii_formatd(buffer, buf_len, format, d)` convierte un double en una cadena ASCII.

El código para estas funciones proviene de la biblioteca GLib (https://developer-old.gnome.org/glib/2.26/ [http://web.archive.org/web/20210306104320/https://developer.gnome.org/glib/2.26/]), cuyos desarrolladores amablemente renovaron la licencia de las funciones relevantes y las donaron a la Python Software Foundation. El módulo `locale` ahora puede cambiar la configuración regional numérica, lo que permite que extensiones como GTK+ produzcan los resultados correctos.

Ver también

PEP 331 [https://peps.python.org/pep-0331/] **Conversiones Local-Independiente flotante/cadena de texto**

Escrito por Christian R. Reis, e implementado por Gustavo Carneiro.

Otros cambios en el lenguaje

Aquí están todos los cambios que Python 2.4 ha hecho al core del lenguaje Python.

- Fueron añadidos decoraciones para funciones (**PEP 318** [https://peps.python.org/pep-0318/]).

- Funciones de conjuntos integrados `set()` y `frozenset()` fueron añadidas (**PEP 218** [https://peps.python.org/pep-0218/]). Otras nuevas funciones de conjuntos integrados incluyen la función `reversed(seq)` (**PEP 322** [https://peps.python.org/pep-0322/]).

- Generador de expresiones fueron añadidas (**PEP 289** [https://peps.python.org/pep-0289/]).

- Algunas expresiones numéricas ya no retornan valores restringidos a 32 o 64 bits (**PEP 237** [https://peps.python.org/pep-0237/]).

- Se puede poner entre paréntesis una lista de nombres en la declaración `from module import names` (**PEP 328** [https://peps.python.org/pep-0328/]).

- El método `dict.update()` ahora acepta el mismo argumento que el constructor de la `dict`. Esto incluye cualquier tipo de mapeo, iterable pares de clave/valor y argumentos de palabra clave. (Contribución de Raymond Hettinger)

- Los métodos de cadena de caracteres **`ljust()`**, **`rjust()`** y **`center()`** ahora toman un argumento de tipo opcional para especificar un carácter de relleno que no sea un espacio. (Contribución de Raymond Hettinger).

- Cadenas de caracteres también han ganado un método **`rsplit()`** que funciona como el método **`split()`** pero divide desde el final de la cadena de caracteres. (Contribución de Sean Reifschneider)

  ```
  >>> 'www.python.org'.split('.', 1)
  ['www', 'python.org']
  'www.python.org'.rsplit('.', 1)
  ['www.python', 'org']
  ```

- Tres parámetros de argumentos de palabra clave *cmp*, *key* y *reverse* fueron añadidos al método de listas **`sort()`**. Estos parámetros hacen algunos usos comunes del método **`sort()`** mas simple. Todos estos parámetros son opcionales.

  Para el parámetro *cmp* el valor debe ser una función comparativa que toma dos parámetros y retorna -1, 0 o + 1 dependiendo de como compare los parámetros. Esta función será usada para ordenar la lista. Previamente esto era el único parámetro que podía ser entregado al **`sort()`**.

  key debe ser una función con un solo parámetro que toma un elemento de la lista y retorna una llave de comparación para el elemento. Entonces la lista es ordenado usando las claves de comparación. El siguiente ejemplo ordena una lista sin distinción de mayúsculas ni minúsculas:

  ```
  >>> L = ['A', 'b', 'c', 'D']
  ```

```
>>> L.sort() # Ordenación que distingue entre mayús
>>> L
['A', 'D', 'b', 'c']
>>> # Uso del parámetro 'key' para ordenar la lista
>>> L.sort(key=lambda x: x.lower())
>>> L
['A', 'b', 'c', 'D']
>>> # Método tradicional
>>> L.sort(cmp=lambda x,y: cmp(x.lower(), y.lower()
>>> L
['A', 'b', 'c', 'D']
```

El ultimo ejemplo el cual usa el parámetro *cmp* es la antigua forma de ejecutar el orden sin distinción de mayúsculas y minúsculas. Funciona pero es mas lento en vez de usar el parámetro *key*. El uso de *key* llama al método `lower()` una vez para cada elemento de la lista mientras el uso de *cmp* lo llamara dos veces para cada comparación, entonces al usar * key * se guardan las llamadas del método `lower()`.

Para funciones de clave sencillas y comparativas es a menudo posible el obviar la palabra clave `lambda` usando un método sin ligar en reemplazo. El siguiente ejemplo de orden sin importar mayúsculas ni minúsculas es mejor escrito como se muestra:

```
>>> L.sort(key=str.lower)
>>> L
['A', 'b', 'c', 'D']
```

Finalmente el parámetro *reverse* toma un valor de tipo booleano. Si el valor es verdadero la lista será ordenada de forma inversa. En vez de `L.sort(); L.reverse()` ahora se puede escribir `L.sort(reverse=True)`.

Los resultados del ordenamiento ahora están garantizados que son estables. Esto significa que dos entradas con iguales claves serán retiradas en el mismo orden en que fueron ingresadas. Por ejemplo se puede ordenar una lista de personas por el nombre y entonces ordenarla por edad, resultando en una lista ordenada donde las personas con la misma edad están en el mismo orden por nombre.

(Todos los cambios al método **`sort()`** fueron realizados por Raymond Hettinger.)

- Se tiene una nueva función incorporada `sorted(iterable)` que trabaja como el método in situ **`list.sort()`** que se puede usar en expresiones. Las diferencias son:
- el dato de entrada puede ser algún iterable;
- se ordena una copia recién formada, manteniendo el original intacto y
- la expresión retorna una nueva copia ordenada

```
>>> L = [9,7,8,3,2,4,1,6,5]
>>> [10+i for i in sorted(L)] # utilizable en una c
[11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> L # el original no se modifica
[9,7,8,3,2,4,1,6,5]
>>> sorted('Monty Python') # cualquier iterable pue
[' ', 'M', 'P', 'h', 'n', 'n', 'o', 'o', 't', 't',

>>> # Lista el contenido de un diccionario ordenado
>>> colormap = dict(red=1, blue=2, green=3, black=4
>>> for k, v en sorted(colormap.iteritems()):
... imprimir k, v
...
negro 4
azul 2
verde 3
rojo 1
amarillo 5
```

(Contribución de Raymond Hettinger.)

- Operaciones de números enteros no lanzaran una excepción **`OverflowWarning`**. La advertencia sobre

`OverflowWarning` desaparecerá en Python 2.5.

- Se añadió al interprete una nueva opción de cambio **`-m`** la cual toma un nombre, busca el módulo correspondiente en `sys.path` y corre el módulo como script. Ahora por ejemplo se puede correr el perfilador de Python con `python -m profile`. (Contribución de Nick Coghlan)

- La expresión `eval(expr, globals, locals)` la función `execfile(filename, globals, locals)` y la sentencia `exec` ahora aceptan cualquier tipo de mapeo para parámetros locales. Previamente esto era parte de un diccionario de Python. (Contribución de Raymond Hettinger.)

- La función incorporada **`zip()`** y la función **`itertools.izip()`** ahora retornan una lista hacia si se llama sin argumentos. Previamente estos lanzaban una excepción de tipo **`TypeError`**. Esto lo hace mas apropiado de usar con listas de argumentos de longitud variable:

  ```
  >>> def transpose(matriz):
  ... return zip(*matriz)
  ...
  >>> transpose([(1,2,3), (4,5,6)])
  [(1, 4), (2, 5), (3, 6)]
  >>> transpose([])
  []
  ```

  (Contribución de Raymond Hettinger.)

- Encontrar un error al importar un módulo ya no deja un objeto de módulo parcialmente inicializado en `sys.modules`. El objeto de módulo incompleto que se deja atrás engañaría a las importaciones posteriores del mismo módulo para que tuvieran éxito, lo que generaría errores confusos. (Reparado por Tim Peters.)

- **`None`es ahora una constante, el código que une un nuevo valor con el nombre :const:`None`** es ahora un error de sintaxis (Contribución de Raymond Hettinger)

Optimizaciones

- Los bucles internos para separar listas y tuplas fueron optimizados y ahora corren un tercio mas rápido. Los bucles internos para diccionarios también fueron optimizados resultando en una mejora en el rendimiento para `keys()`, `values()`, `items()`, `iterkeys()`, `itervalues()`, y `iteritems()`. (Contribución de Raymond Hettinger)
- El mecanismo para mejorar y contraer listas fue optimizada para efectos de velocidad y eficiencia en la utilización de recursos. Anexar y quitar elementos de una lista es ahora mas rápido debido al código mas eficiente y al menos frecuente uso de la función subyacente `realloc()`. Las listas por comprensión también son beneficiadas, el método `list.extend()` fue también optimizado y ya no convierte mas el argumento en una lista temporal antes de extender la lista base. (Contribución de Raymond Hettinger)
- Las funciones `list()`, `tuple()`, `map()`, `filter()` y `zip()` ahora corren mucho mas rápido con argumentos no secuenciales que ahora suministra el método `__len__()`. (Contribución Raymond Hettinger.)
- Los métodos `list.__getitem__()`, `dict.__getitem__()` y `dict.__contains__()` ahora son implementados como objetos de la clase `method_descriptor` en vez de los objetos de la clase `wrapper_descriptor`. Este acceso dobla el rendimiento y hace que sea mas apropiado para ser usado como argumentos de:`map(mydict.__getitem__, keylist)`. (Contribución de Raymond Hettinger)
- Adicionalmente se ha agregado a un nuevo código de operación `LIST_APPEND` esto simplifica la generación de código a nivel de byte para las listas por comprensión y las agiliza en aproximadamente un tercio.
- El optimizador de código a nivel de byte ha sido mejorado para producir código a nivel de byte mas corto y rápido. Esto resulta en código mas legible. (Contribución de Raymond Hettinger)
- La concatenación de cadenas de texto en la declaración de `s = s + "abc"` and `s += "abc"` ahora son ejecutadas de forma mas eficiente en ciertas circunstancias. Esta

optimizaciones no serán mostradas en otras implementaciones de Python tales como Python, entonces no se debe utilizar, es mejor seguir utilizando el método **`join()`** de cadenas de caracteres si se quiere mantener la eficiencia de concatenar un largo número de cadenas de caracteres. (Contribución de Armin Rigo.)

El resultado de las optimizaciones de la versión 2.4 es que tomando Python 2.4 como punto de referencia corre alrededor de un 5% mas rápido que Python 2.3 y un 35% mas rápido que Python 2.2 (pystone no es particularmente un buen punto de referencia pero es el mas usado para medir rendimiento en Python. Sus propias aplicaciones pueden mostrar beneficios mas grandes o mínimos desde Python 2.4)

Módulos nuevos, mejorados y obsoletos

Como es usual, la librería estándar de Python recibió un número de mejoras y corrección de errores. Aquí hay una lista parcial de los cambios mas notables, ordenados alfabéticamente por nombre de módulo. Consulte el archivo `Misc/NEWS` en la estructura del directorio para una completa lista de los cambios o se puede buscar a través del registro CVS para obtener todos los detalles.

- La función **`loop()`** del módulo **`asyncore`** ahora tiene un parámetro *count* que le permite realizar una cantidad limitada de pasadas a través del bucle de sondeo. El valor predeterminado sigue siendo el bucle indefinidamente.

- El módulo **`base64`** ahora tiene un soporte mas completo **RFC 3548** [https://datatracker.ietf.org/doc/html/rfc3548.html] para Base64, Base32 y Base16 para codificación y descodificación, incluyendo procesos de convertir caracteres a minúsculas y alfabetos alternativos. (Contribución de Barry Warsaw)

- El módulo **`bisect`** ahora tiene una implementación C subyacente para mejorar el rendimiento. (Contribuido por Dmitry Vasiliev.)

- Las colecciones CJKCodecs de códecs de Asia oriental, mantenidas por Hye-Shik Chang, se integraron en 2.4. Las nuevas codificaciones son:

- Chino (PRC): gb2312, gbk, gb18030, big5hkscs, hz
- Chino (República de China): big5, cp950
- Japonés: cp932, euc-jis-2004, euc-jp, euc-jisx0213, iso-2022-jp,
  iso-2022-jp-1, iso-2022-jp-2, iso-2022-jp-3, iso-2022-jp-ext, iso-2022-jp-2004, shift-jis, shift-jisx0213, shift-jis-2004
- Coreano: cp949, euc-kr, johab, iso-2022-kr
- Se agregaron algunas otras codificaciones nuevas: HP Roman8, ISO_8859-11, ISO_8859-16, PCTP-154 y TIS-620.
- Los códecs UTF-8 y UTF-16 ahora se adaptan mejor a la recepción de entradas parciales. Anteriormente, la clase **`StreamReader`** intentaba leer más datos, lo que hacía imposible reanudar la decodificación desde el flujo. El método **`read()`** ahora devolverá tantos datos como pueda y las llamadas futuras reanudarán la decodificación donde lo dejaron las anteriores. (Implementado por Walter Dörwald.)
- Hay un nuevo módulo **`collections`** para varios tipos de datos de recopilación especializados. Actualmente contiene solo un tipo, **`deque`**, una cola de dos extremos que admite la adición y eliminación de elementos de manera eficiente desde cualquier extremo:

```
>>> from collections import deque
>>> d = deque('ghi') # crea un nuevo deque con tres
>>> d.append('j') # agrega una nueva entrada al lad
>>> d.appendleft('f') # agrega una nueva entrada al
>>> d # muestra la representación del deque
deque(['f', 'g', 'h', 'i', 'j'])
>>> d.pop() # devuelve y elimina el elemento más a
'j'
>>> d.popleft() # devuelve y elimina el elemento má
```

```
'f'
>>> list(d) # lista el contenido del deque
['g', 'h', 'i']
>>> 'h' in d # busca el deque
True
```

Varios módulos, como los módulos `Queue` y `threading`, ahora aprovechan `collections.deque` para mejorar el rendimiento. (Contribuido por Raymond Hettinger.)

- Las clases `ConfigParser` se han mejorado ligeramente. El método `read()` ahora devuelve una lista de los archivos que se analizaron correctamente y el método `set()` genera `TypeError` si se le pasa un argumento *value* que no sea una cadena. (Contribuido por John Belmonte y David Goodger).

- El módulo `curses` ahora admite la extensión ncurses `use_default_colors()`. En plataformas donde el terminal admite transparencia, esto permite utilizar un fondo transparente. (Contribución de Jörg Lehmann.)

- El módulo `difflib` ahora incluye una clase `HtmlDiff` que crea una tabla HTML que muestra una comparación lado a lado de dos versiones de un texto. (Contribuido por Dan Gass.)

- El paquete `email` se actualizó a la versión 3.0, que eliminó varias API obsoletas y elimina la compatibilidad con versiones de Python anteriores a la 2.3. La versión 3.0 del paquete usa un nuevo analizador incremental para mensajes MIME, disponible en el módulo `email.FeedParser`. El nuevo analizador no requiere leer todo el mensaje en la memoria y no genera excepciones si un mensaje tiene un formato incorrecto; en su lugar, registra cualquier problema en el atributo `defect` del mensaje. (Desarrollado por Anthony Baxter, Barry Warsaw, Thomas Wouters y otros).

- El módulo `heapq` se ha convertido a C. La mejora de diez veces resultante en la velocidad hace que el módulo sea adecuado para manejar grandes volúmenes de datos. Además, el módulo tiene dos funciones nuevas `nlargest()` y `nsmallest()` que usan montones para encontrar los N valores más grandes o más pequeños en un conjunto de datos sin el gasto de una clasificación completa. (Contribuido por Raymond Hettinger.)

- El módulo `httplib` ahora contiene constantes para los códigos de estado HTTP definidos en varios documentos RFC relacionados con HTTP. Las constantes tienen nombres como `OK`, `CREATED`, `CONTINUE` y `MOVED_PERMANENTLY`; utilice pydoc para obtener una lista completa. (Contribuido por Andrew Eland).

- El módulo `imaplib` ahora es compatible con el comando THREAD de IMAP (aportado por Yves Dionne) y los nuevos métodos `deleteacl()` y `myrights()` (aportado por Arnaud Mazin).

- El módulo `itertools` ganó una función `groupby(iterable[, *func*])`. El parámetro *iterable* is something that can be iterated over to return a stream of elements, and the optional *func* es una función que toma un elemento y devuelve un valor clave; si se omite, la clave es simplemente el elemento en sí. `groupby()` luego agrupa los elementos en subsecuencias que tienen valores coincidentes de la clave y devuelve una serie de 2 tuplas que contienen el valor de la clave y un iterador sobre la subsecuencia.

  Aquí hay un ejemplo para aclarar esto. La función *key* simplemente devuelve si un número es par o impar, por lo que el resultado de `groupby()` es devolver series consecutivas de números pares o impares.

  ```
  >>> importar itertools
  >>> L = [2, 4, 6, 7, 8, 9, 11, 12, 14]
  >>> para key_val, it en itertools.groupby(L, lambda
  ... imprimir key_val, list(it)
  ...
  0 [2, 4, 6]
  1 [7]
  0 [8]
  ```

```
1 [9, 11]
0 [12, 14]
>>>
```

`groupby()` se utiliza normalmente con entrada ordenada. La lógica para **`groupby()`** es similar al filtro Unix `uniq`, lo que lo hace útil para eliminar, contar o identificar elementos duplicados:

```
>>> palabra = 'abracadabra'
>>> letras = sorted(palabra) # Convierte una cadena
>>> letras
['a', 'a', 'a', 'a', 'a', 'b', 'b', 'c', 'd', 'r',
>>> for k, g in itertools.groupby(letters):
... print k, list(g)
...
a ['a', 'a', 'a', 'a', 'a']
b ['b', 'b']
c ['c']
d ['d']
r ['r', 'r']
>>> # Lista de letras únicas
>>> [k for k, g in groupby(letters)]
['a', 'b', 'c', 'd', 'r']
>>> # Cuenta las ocurrencias de las letras
>>> [(k, len(lista(g))) para k, g en groupby(letras
[('a', 5), ('b', 2), ('c', 1), ('d', 1), ('r', 2)]
```

(Contribuido por Hye-Shik Chang.)

- **`itertools`** también obtuvo una función denominada `tee(iterator, N)` que devuelve *N* independent iterators that replicate *iterator*. If *N* se omite, el valor predeterminado es 2.

```
>>> L = [1,2,3]
>>> i1, i2 = itertools.tee(L)
>>> i1,i2
(<itertools.tee object at 0x402c2080>, <itertools.t
>>> list(i1) # Ejecuta el primer iterador hasta el
```

```
[1, 2, 3]
>>> list(i2) # Ejecuta el segundo iterador hasta el
[1, 2, 3]
```

Tenga en cuenta que **`tee()`** debe mantener copias de los valores devueltos por el iterador; en el peor de los casos, es posible que deba conservarlos todos. Por lo tanto, esto debe usarse con cuidado si el iterador principal puede ejecutarse muy por delante del iterador final en un flujo largo de entradas. Si la separación es grande, entonces también podría usar **`list()`** en su lugar. Cuando los iteradores se siguen de cerca entre sí, **`tee()`** es ideal. Las posibles aplicaciones incluyen marcadores, ventanas o iteradores de anticipación. (Contribuido por Raymond Hettinger.)

- Se agregaron varias funciones al módulo **`locale`**, como **`bind_textdomain_codeset()`** para especificar una codificación particular y una familia de funciones **`l*gettext()`** que devuelven mensajes en la codificación elegida. (Contribuido por Gustavo Niemeyer.)

- Se agregaron algunos argumentos de palabras clave a la función **`basicConfig()`** del paquete **`logging`** para simplificar la configuración del registro. El comportamiento predeterminado es registrar mensajes con error estándar, pero se pueden especificar varios argumentos de palabras clave para registrar en un archivo en particular, cambiar el formato de registro o establecer el nivel de registro. Por ejemplo:

```
importar registro
logging.basicConfig(filename='/var/log/application.
level=0, # Registrar todos los mensajes
format='%(levelname):%(process):%(thread):%(message
```

Otras adiciones al paquete **`logging`** incluyen un método de conveniencia `log(level, msg)`, así como una clase **`TimedRotatingFileHandler`** que rota sus archivos de registro en un intervalo cronometrado. El módulo ya tenía **`RotatingFileHandler`**, que rotaba los registros una vez que el archivo excedía un cierto tamaño. Ambas clases derivan de una nueva clase **`BaseRotatingHandler`** que se puede utilizar para implementar otros controladores rotativos.

(Cambios implementados por Vinay Sajip.)

- El módulo **`marshal`** ahora comparte cadenas internas al desempaquetar una estructura de datos. Esto puede reducir el tamaño de ciertas cadenas de pickle, pero el efecto principal es hacer que los archivos `.pyc` sean significativamente más pequeños. (Contribución de Martin von Löwis.)

- La clase **`NNTP`** del módulo **`nntplib`** obtuvo los métodos **`description()`** y **`descriptions()`** para recuperar descripciones de grupos de noticias para un solo grupo o para un rango de grupos. (Contribución de Jürgen A. Erhard).

- Se agregaron dos nuevas funciones al módulo **`operator`**, `attrgetter(attr)` y `itemgetter(index)`. Ambas funciones devuelven invocables que toman un solo argumento y devuelven el atributo o elemento correspondiente; estos llamables son excelentes extractores de datos cuando se utilizan con **`map()`** o **`sorted()`**. Por ejemplo:

  ```
  >>> L = [('c', 2), ('d', 1), ('a', 4), ('b', 3)]
  >>> map(operator.itemgetter(0), L)
  ['c', 'd', 'a', 'b']
  >>> map(operator.itemgetter(1), L)
  [2, 1, 4, 3]
  >>> sorted(L, key=operator.itemgetter(1)) # Ordenar
  [('d', 1), ('c', 2), ('b', 3), ('a', 4)]
  ```

  (Contribución de Raymond Hettinger.)

- El módulo **`optparse`** se actualizó de varias formas. El módulo ahora pasa sus mensajes a través de **`gettext.gettext()`**, lo que permite internacionalizar los mensajes de error y ayuda de Optik. Los mensajes de ayuda para las opciones ahora pueden incluir la cadena `'%default'`, que será reemplazada por el valor predeterminado de la opción. (Contribuido por Greg Ward.)

- El plan a largo plazo es dejar obsoleto el módulo **`rfc822`** en alguna versión futura de Python a favor del paquete **`email`**. Con este fin, se ha modificado la función **`email.Utils.formatdate`** para que pueda utilizarse como reemplazo de **`rfc822.formatdate()`**. Es posible que desee escribir un nuevo código de procesamiento de correo electrónico teniendo esto en cuenta. (Cambio implementado por Anthony Baxter).

- Se agregó una nueva función `urandom(n)` al módulo **`os`**, que devuelve una cadena que contiene *n* bytes de datos aleatorios. Esta función proporciona acceso a fuentes de aleatoriedad específicas de la plataforma, como `/dev/urandom` en Linux o Windows CryptoAPI. (Contribuido por Trevor Perrin.)

- Otra función nueva: `os.path.lexists(path)` devuelve verdadero si el archivo especificado por *path* exists, whether or not it's a symbolic link. This differs from the existing ASDF01 function, which returns false if *path* es un enlace simbólico que apunta a un destino que no existe. (Contribuido por Beni Cherniavsky.)

- Se agregó una nueva función **`getsid()`** al módulo **`posix`** que subyace al módulo **`os`**. (Contribuido por J. Raynor.)

- El módulo **`poplib`** ahora admite POP sobre SSL. (Contribuido por Héctor Urtubia.)

- El módulo **`profile`** ahora puede perfilar funciones de extensión C. (Contribuido por Nick Bastin.)

- El módulo **`random`** tiene un nuevo método llamado `getrandbits(N)` que devuelve un entero largo *N* bits de longitud. El método **`randrange()`** existente ahora usa **`getrandbits()`** cuando corresponde, lo que hace que la generación de números aleatorios arbitrariamente grandes sea más eficiente. (Contribuido por Raymond Hettinger.)

- El lenguaje de expresiones regulares aceptado por el módulo **`re`** se amplió con expresiones condicionales simples, escritas como `(?(group)A|B)`. En su lugar, se utilizará *group* is either a numeric group ID or a group name defined with ASDF02 earlier in the expression. If the specified group matched, the regular expression pattern *A* will be tested against the string; if the group didn't match, the pattern *B*. (Contribución de Gustavo Niemeyer.)

- El módulo **`re`** ya no es recursivo, gracias a la enorme cantidad de trabajo de Gustavo Niemeyer. En un motor de expresión regular recursivo, ciertos patrones dan como resultado que se consuma una gran cantidad de espacio de pila de C y es posible desbordar la pila. Por ejemplo, si comparó una cadena de 30000 bytes de caracteres `a` con la expresión `(a|b)+`, se consumió un marco de pila por carácter. Python 2.3 intentó verificar el desbordamiento de la pila y generar una excepción **`RuntimeError`**, pero ciertos patrones podrían eludir la verificación y, si no tuvo suerte, Python podría segregar. El motor de expresiones regulares de Python 2.4 puede coincidir con este patrón sin problemas.

- El módulo **`signal`** ahora realiza una verificación de errores más estricta en los parámetros de la función **`signal.signal()`**. Por ejemplo, no puede establecer un controlador en la señal **`SIGKILL`**; las versiones anteriores de Python aceptarían silenciosamente esto, pero 2.4 lanzará una excepción **`RuntimeError`**.

- Se agregaron dos nuevas funciones al módulo **`socket`**. **`socketpair()`** devuelve un par de sockets conectados y `getservbyport(port)` busca el nombre del servicio para un número de puerto determinado. (Contribuido por Dave Cole y Barry Warsaw.)

- La función **`sys.exitfunc()`** ha quedado obsoleta. El código debe usar el módulo **`atexit`** existente, que maneja correctamente la llamada a múltiples funciones de salida. Finalmente, **`sys.exitfunc()`** se convertirá en una interfaz puramente interna, a la que solo accederá **`atexit`**.

- El módulo **`tarfile`** ahora genera archivos tar en formato GNU de forma predeterminada. (Contribución de Lars

Gustäbel.)

- El módulo `threading` ahora tiene una forma elegante y sencilla de admitir datos locales de subprocesos. El módulo contiene una clase `local` cuyos valores de atributo son locales para diferentes subprocesos.

  ```
  importar subprocesos

  datos = threading.local()
  datos.number = 42
  datos.url = ('www.python.org', 80)
  ```

  Otros subprocesos pueden asignar y recuperar sus propios valores para los atributos `number` y `url`. Puede crear una subclase de `local` para inicializar atributos o agregar métodos. (Contribuido por Jim Fulton.)

- El módulo `timeit` ahora deshabilita automáticamente la recolección de basura periódica durante el ciclo de temporización. Este cambio hace que los tiempos consecutivos sean más comparables. (Contribuido por Raymond Hettinger.)

- El módulo `weakref` ahora admite una variedad más amplia de objetos, incluidas funciones de Python, instancias de clases, conjuntos, frozensets, deques, matrices, archivos, sockets y objetos de patrones de expresión regular. (Contribuido por Raymond Hettinger.)

- El módulo `xmlrpclib` ahora admite una extensión de múltiples llamadas para transmitir múltiples llamadas XML-RPC en una sola operación HTTP. (Contribuido por Brian Quinlan).

- Se han eliminado los módulos `mpz`, `rotor` y `xreadlines`.

cookielib

La biblioteca `cookielib` admite el manejo de cookies HTTP por parte del cliente, lo que refleja la compatibilidad con cookies del lado del servidor del módulo `Cookie`. Las cookies se almacenan en contenedores de cookies; la biblioteca almacena de forma transparente las cookies ofrecidas por el servidor web en el contenedor de cookies y recupera la cookie del contenedor cuando se conecta al servidor. Al igual que en los navegadores web, los objetos de política controlan si se aceptan o no las cookies.

Para almacenar cookies entre sesiones, se proporcionan dos implementaciones de tarros de cookies: una que almacena cookies en formato Netscape para que las aplicaciones puedan usar los archivos de cookies de Mozilla o Lynx, y otra que almacena cookies en el mismo formato que la biblioteca libwww de Perl.

Se ha modificado `urllib2` para interactuar con `cookielib`: **`HTTPCookieProcessor`** administra un contenedor de cookies que se utiliza al acceder a las URL.

Este módulo fue una contribución de John J. Lee.

doctest

El módulo `doctest` se sometió a una refactorización considerable gracias a Edward Loper y Tim Peters. Las pruebas pueden ser tan simples como ejecutar `doctest.testmod()`, pero las refactorizaciones permiten personalizar el funcionamiento del módulo de varias formas.

La nueva clase **`DocTestFinder`** extrae las pruebas de las cadenas de documentos de un objeto dado:

```
def f (x, y):
""">>> f(2,2)
4
>>> f(3,2)
6
"""
return x*y

finder = doctest.DocTestFinder()
```

```
# Obtener la lista de instancias de DocTest
tests = finder.find(f)
```

La nueva clase **`DocTestRunner`** luego ejecuta pruebas individuales y puede producir un resumen de los resultados:

```
runner = doctest.DocTestRunner()
para t en pruebas:
intentado, fallido = runner.run(t)

runner.summarize(verbose=1)
```

El ejemplo anterior produce la siguiente salida:

```
1 ítems pasaron todas las pruebas:
2 pruebas en f
2 pruebas en 1 ítem.
2 pasaron y 0 reprobaron.
Prueba aprobada.
```

`DocTestRunner` usa una instancia de la clase **`OutputChecker`** para comparar la salida esperada con la salida real. Esta clase toma varios indicadores diferentes que personalizan su comportamiento; Los usuarios ambiciosos también pueden escribir una subclase completamente nueva de **`OutputChecker`**.

El comprobador de salida predeterminado proporciona una serie de funciones útiles. Por ejemplo, con el indicador de opción **`doctest.ELLIPSIS`**, una elipsis (`...`) en la salida esperada coincide con cualquier subcadena, lo que facilita la adaptación de salidas que varían en formas menores:

```
def o (n):
    """>>> o(1)
<__main__.C instance at 0x...>
>>>
"""
```

Otra cadena especial, `<BLANKLINE>`, coincide con una línea en blanco:

```
def p (n):
    """>>> p(1)
<BLANKLINE>
>>>
"""
```

Otra nueva capacidad es producir una visualización de estilo diff de la salida especificando `doctest.REPORT_UDIFF` (diferencias unificadas), `doctest.REPORT_CDIFF` (diferencias de contexto), o `doctest.REPORT_NDIFF` Indicadores de opción (estilo delta). Por ejemplo:

```
def g (n):
    """>>> g(4)
here
is
a
lengthy
>>>"""
    L = 'here is a rather lengthy list of words'.split()
    for word in L[:n]:
        print word
```

Al ejecutar las pruebas de la función anterior con `doctest.REPORT_UDIFF` especificado, obtiene el siguiente resultado:

```
********************************************************
File "t.py", line 15, in g
Failed example:
    g(4)
Differences (unified diff with -expected +actual):
    @@ -2,3 +2,3 @@
     is
     a
    -lengthy
    +rather
********************************************************
```

Cambios en la API de Build y C

Algunos de los cambios en el proceso de compilación de Python y en la API de C son:

- Se agregaron tres nuevas macros de conveniencia para valores de retorno comunes de funciones de extensión: **`Py_RETURN_NONE`**, **`Py_RETURN_TRUE`**, y **`Py_RETURN_FALSE`**. (Contribuido por Brett Cannon.)
- Otra macro nueva, **`Py_CLEAR`**, reduce el recuento de referencias de *obj* y establece *obj* en el puntero nulo. (Aportado por Jim Fulton.)
- Una nueva función, `PyTuple_Pack(N, obj1, obj2, ..., objN)`, construye tuplas a partir de una lista de argumentos de longitud variable de objetos Python. (Contribuido por Raymond Hettinger.)
- Una nueva función, `PyDict_Contains(d, k)`, implementa búsquedas rápidas de diccionarios sin enmascarar las excepciones que surgen durante el proceso de búsqueda. (Contribuido por Raymond Hettinger.)
- La macro Py_IS_NAN(X) retorna 1 si su argumento flotante o doble *X* es un NaN. (Aportado por Tim Peters.)
- El código C puede evitar el bloqueo innecesario mediante el uso de la nueva función **`PyEval_ThreadsInitialized()`** para saber si se ha realizado alguna operación de subproceso. Si esta función devuelve falso, no se necesitan operaciones de bloqueo. (Contribuido por Nick Coghlan.)
- Una nueva función, **`PyArg_VaParseTupleAndKeywords()`**, es la misma que **`PyArg_ParseTupleAndKeywords()`** pero toma un **`va_list`** en lugar de varios argumentos. (Contribuido por Greg Chapman.)
- Un nuevo indicador de método, **`METH_COEXISTS`**, permite que una función definida en ranuras coexista con un **`PyCFunction`** que tiene el mismo nombre. Esto puede reducir a la mitad el tiempo de acceso para un método como **`set.__contains__()`**. (Contribuido por Raymond Hettinger.)
- Python ahora se puede construir con perfiles adicionales para el intérprete en sí, con la intención de ayudar a las personas

que desarrollan el núcleo de Python. Proporcionar `–enable-profiling` al **configure** script le permitirá perfilar el intérprete con **gprof**, y proporcionar el `– with-tsc` switch permite la creación de perfiles utilizando el registro de contador de marcas de tiempo del Pentium. Tenga en cuenta que el conmutador `– with-tsc` tiene un nombre ligeramente incorrecto, porque la función de creación de perfiles también funciona en la plataforma PowerPC, aunque la arquitectura del procesador no llama a ese registro» el registro TSC «. (Contribuido por Jeremy Hylton.)
- El tipo `tracebackobject` ha sido renombrado a `PyTracebackObject`.

Cambios específicos del puerto

- El puerto de Windows ahora se construye bajo MSVC + + 7.1 y también con la versión 6. (Contribuido por Martin von Löwis).

Portar a Python 2.4

Esta sección enumera los cambios descritos anteriormente que pueden requerir cambios en su código:

- Los desplazamientos a la izquierda y las constantes hexadecimales / octales que son demasiado grandes ya no activan un `FutureWarning` y devuelven un valor limitado a 32 o 64 bits; en su lugar, devuelven un entero largo.
- Operaciones de números enteros no lanzaran una excepción `OverflowWarning`. La advertencia sobre `OverflowWarning` desaparecerá en Python 2.5.
- La función de conjunto integrado `zip()` y la `itertools.izip()` ahora retornan una lista hacia en vez de lanzar una excepción de tipo `TypeError` si son llamados sin argumentos.
- Ya no puede comparar las instancias `date` y `datetime` proporcionadas por el módulo `datetime`. Dos instancias de clases diferentes ahora siempre serán desiguales, y las comparaciones relativas (<, >) lanzarán un `TypeError`.

- **`dircache.listdir()`** ahora pasa excepciones a la persona que llama en lugar de devolver listas vacías.
- **`LexicalHandler.startDTD()`** solía recibir los identificadores públicos y del sistema en el orden incorrecto. Esto ha sido corregido; las aplicaciones que se basan en el orden incorrecto deben corregirse.
- **`fcntl.ioctl()`** ahora advierte si el argumento *mutate* se omite y es relevante.
- El módulo **`tarfile`** ahora genera archivos de GNU tar-formato por defecto.
- Encontrar un error al importar un módulo ya no deja un objeto de módulo parcialmente inicializado en `sys.modules`.
- **`None`** ahora es una constante; El código que une un nuevo valor al nombre `None` ahora es un error de sintaxis.
- La función **`signals.signal()`** ahora genera una excepción **`RuntimeError`** para ciertos valores ilegales; anteriormente estos errores pasaban silenciosamente. Por ejemplo, ya no puede configurar un controlador en la señal **`SIGKILL`**.

Agradecimientos

El autor desea agradecer a las siguientes personas por ofrecer sugerencias, correcciones y ayuda con varios borradores de este artículo: Koray Can, Hye-Shik Chang, Michael Dyck, Raymond Hettinger, Brian Hurt, Hamish Lawson, Fredrik Lundh, Sean Reifschneider, Sadruddin Rejeb.

Qué hay de nuevo en Python 2.3

Autor:

A.M. Kuchling

Este artículo explica las nuevas características de Python 2.3. Python 2.3 se publicó el 29 de julio de 2003.

Los temas principales de Python 2.3 son el pulido de algunas de las características añadidas en la 2.2, la adición de varias mejoras pequeñas pero útiles al núcleo del lenguaje y la ampliación de la biblioteca estándar. El nuevo modelo de objetos introducido en la versión anterior se ha beneficiado de 18 meses de correcciones de errores y de esfuerzos de optimización que han mejorado el rendimiento de las clases de nuevo estilo. Se han añadido algunas funciones incorporadas, como `sum()` y `enumerate()`. El operador `in` puede utilizarse ahora para búsquedas de subcadenas (por ejemplo, `"ab" en "abc"` retorna `True`).

Algunas de las nuevas características de la biblioteca son los tipos de datos booleanos, de conjunto, de montón y de fecha/hora, la posibilidad de importar módulos desde archivos con formato ZIP, el soporte de metadatos para el tan esperado catálogo de Python, una versión actualizada de IDLE y módulos para registrar mensajes, envolver texto, analizar archivos CSV, procesar opciones de línea de comandos, utilizar bases de datos BerkeleyDB… la lista de módulos nuevos y mejorados es larga.

Este artículo no pretende proporcionar una especificación completa de las nuevas características, sino que proporciona una visión general conveniente. Para obtener todos los detalles, debes consultar la documentación de Python 2.3, como la Referencia de la Biblioteca de Python y el Manual de Referencia de Python. Si quieres entender la implementación completa y los fundamentos del diseño, consulta el PEP de una nueva característica en particular.

PEP 218: Un tipo de datos de conjunto estándar

El nuevo módulo **`sets`** contiene una implementación de un tipo de datos de conjunto. La clase **`Set`** es para conjuntos mutables, conjuntos a los que se les pueden agregar y quitar miembros. La clase **`ImmutableSet`** es para conjuntos que no se pueden modificar y, por lo tanto, las instancias de **`ImmutableSet`** se pueden usar como claves de diccionario. Los conjuntos se construyen sobre diccionarios, por lo que los elementos dentro de un conjunto deben ser codificables.

Aquí hay un ejemplo simple:

```
>>> importar conjuntos
>>> S = conjuntos.Set([1,2,3])
>>> S
Set([1, 2, 3])
>>> 1 en S
Verdadero
>>> 0 en S
Falso
>>> S.add(5)
>>> S.remove(3)
>>> S
Set([1, 2, 5])
>>>
```

The union and intersection of sets can be computed with the **`union()`** and **`intersection()`** methods; an alternative notation uses the bitwise operators `&` and `|`. Mutable sets also have in-place versions of these methods, **`union_update()`** and **`intersection_update()`**.

```
>>> S1 = conjuntos.Set([1,2,3])
>>> S2 = conjuntos.Set([4,5,6])
>>> S1.union(S2)
```

```
Set([1, 2, 3, 4, 5, 6])
>>> S1 | S2 # Notación alternativa
Set([1, 2, 3, 4, 5, 6])
>>> S1.intersection(S2)
Set([])
>>> S1 y S2 # Notación alternativa
Set([])
>>> S1.union_update(S2)
>>> S1
Set([1, 2, 3, 4, 5, 6])
>>>
```

It's also possible to take the symmetric difference of two sets. This is the set of all elements in the union that aren't in the intersection. Another way of putting it is that the symmetric difference contains all elements that are in exactly one set. Again, there's an alternative notation (^), and an in-place version with the ungainly name **`symmetric_difference_update()`**.

```
>>> S1 = conjuntos.Conjunto([1,2,3,4])
>>> S2 = conjuntos.Conjunto([3,4,5,6])
>>> S1.diferencia_simétrica(S2)
Conjunto([1, 2, 5, 6])
>>> S1 ^ S2
Conjunto([1, 2, 5, 6])
>>>
```

También existen los métodos **`issubset()`** y **`issuperset()`** para comprobar si un conjunto es un subconjunto o superconjunto de otro:

```
>>> S1 = conjuntos.Set([1,2,3])
>>> S2 = conjuntos.Set([2,3])
>>> S2.issubset(S1)
Verdadero
>>> S1.issubset(S2)
Falso
>>> S1.issuperset(S2)
Verdadero
```

```
>>>
```

Ver también

PEP 218 [https://peps.python.org/pep-0218/] - Añadiendo un tipo de objeto de conjunto incorporado

PEP escrito por Greg V. Wilson. Implementado por Greg V. Wilson, Alex Martelli y GvR.

PEP 255: Generadores simples

En Python 2.2, los generadores se añadieron como una característica opcional, que se activaba mediante una directiva `from __future__ import generators`. En 2.3 los generadores ya no necesitan ser habilitados especialmente, y ahora están siempre presentes; esto significa que `yield` es ahora siempre una palabra clave. El resto de esta sección es una copia de la descripción de los generadores del documento «What's New in Python 2.2»; si lo leíste cuando salió Python 2.2, puedes saltarte el resto de esta sección.

Sin duda estás familiarizado con cómo funcionan las llamadas a funciones en Python o C. Cuando llamas a una función, ésta obtiene un espacio de nombres privado donde se crean sus variables locales. Cuando la función llega a una declaración `return`, las variables locales se destruyen y el valor resultante se retorna a quien la llamó. Una llamada posterior a la misma función obtendrá un nuevo conjunto de variables locales. Pero, ¿qué pasaría si las variables locales no se tiraran al salir de una función? ¿Qué pasaría si pudieras reanudar la función donde la dejaste? Esto es lo que proporcionan los generadores; se puede pensar en ellos como funciones reanudables.

Este es el ejemplo más sencillo de una función generadora:

```
def generate_ints(N):
para i en rango(N):
rendimiento i
```

Se ha introducido una nueva palabra clave, `yield`, para los generadores. Cualquier función que contenga una declaración `yield` es una función generadora; esto es detectado por el compilador de código de bits de Python que compila la función especialmente como resultado.

Cuando se llama a una función generadora, ésta no retorna un único valor, sino que retorna un objeto generador que soporta el protocolo de los iteradores. Al ejecutar la sentencia `yield`, el generador retorna el valor de `i`, de forma similar a una sentencia `return`. La gran diferencia entre `yield` y una sentencia `return` es que al llegar a una sentencia `yield` se suspende el estado de ejecución del generador y se conservan las variables locales. En la siguiente llamada al método `.next()` del generador, la función se reanudará la ejecución inmediatamente después de la sentencia `yield`. (Por razones complicadas, la sentencia `yield` no está permitida dentro del bloque `try` de una sentencia `try`…`; lea PEP 255 [https://peps.python.org/pep-0255/] para una explicación completa de la interacción entre `yield` y las excepciones)

Este es un ejemplo de uso del generador `generate_ints()`:

```
>>> gen = generate_ints(3)
>>> gen
<objeto generador en 0x8117f90>
>>> gen.next()
0
>>> gen.next()
1
>>> gen.next()
2
>>> gen.next()
Traceback (última llamada más reciente):
Archivo "stdin", línea 1, en ?
Archivo "stdin", línea 2, en generate_ints
StopIteration
```

También podrías escribir `for i in generate_ints(5)`, o `a,b,c = generate_ints(3)`.

Dentro de una función generadora, la expresión **`return`** sólo puede usarse sin un valor, y señala el final de la procesión de valores; después el generador no puede retornar más valores. **`return`** con un valor, como `return 5`, es un error de sintaxis dentro de una función generadora. El final de los resultados del generador también puede indicarse levantando manualmente **`StopIteration`**, o simplemente dejando que el flujo de ejecución caiga en el fondo de la función.

Puedes conseguir el efecto de los generadores manualmente escribiendo tu propia clase y almacenando todas las variables locales del generador como variables de instancia. Por ejemplo, la devolución de una lista de enteros podría hacerse estableciendo `self.count` a 0, y haciendo que el método **`next()`** incremente `self.count` y lo retorne. Sin embargo, para un generador medianamente complicado, escribir la clase correspondiente sería mucho más complicado. `Lib/test/test_generators.py` contiene varios ejemplos más interesantes. El más sencillo implementa un recorrido en orden de un árbol utilizando generadores de forma recursiva

```
# Un generador recursivo que genera hojas de árboles en
def inorder(t):
if t:
for x in inorder(t.left):
yield x
yield t.label
for x in inorder(t.right):
yield x
```

Otros dos ejemplos en `Lib/test/test_generators.py` producen soluciones para el problema de las N reinas (colocar N reinas en un tablero de ajedrez NxN de forma que ninguna reina amenace a otra) y el recorrido del caballero (una ruta que lleva a un caballo a cada casilla de un tablero de ajedrez NxN sin visitar ninguna casilla dos veces).

La idea de los generadores proviene de otros lenguajes de programación, especialmente de Icon (https://www2.cs.arizona.edu/icon/), donde la idea de los generadores es fundamental. En Icon, cada expresión y llamada a una función se comporta como un generador. Un ejemplo de «*An Overview of the Icon Programming Language*» en https://www2.cs.arizona.edu/icon/docs/ipd266.htm da una idea de cómo es esto:

```
oración := "Guardarlo en el puerto vecino"
si (i := find("o", oración)) > 5 entonces write(i)
```

En Icon, la función `find()` devuelve los índices en los que se encuentra la subcadena «o»: 3, 23, 33. En la declaración `if`, a `i` primero se le asigna un valor de 3, pero 3 es menor que 5, por lo que la comparación falla y Icon la vuelve a intentar con el segundo valor de 23. 23 es mayor que 5, por lo que la comparación ahora tiene éxito y el código imprime el valor 23 en la pantalla.

Python no va tan lejos como Icon en la adopción de generadores como concepto central. Los generadores se consideran parte del núcleo del lenguaje Python, pero aprenderlos o utilizarlos no es obligatorio; si no resuelven ningún problema que tengas, siéntete libre de ignorarlos. Una característica novedosa de la interfaz de Python en comparación con la de Icon es que el estado de un generador se representa como un objeto concreto (el iterador) que puede pasarse a otras funciones o almacenarse en una estructura de datos.

Ver también

PEP 255 [https://peps.python.org/pep-0255/] - Generadores simples

Escrito por Neil Schemenauer, Tim Peters, Magnus Lie Hetland. Implementado principalmente por Neil Schemenauer y Tim Peters, con otras correcciones del equipo de Python Labs.

PEP 263: Codificación del código fuente

Los archivos fuente de Python ahora pueden declararse con diferentes codificaciones de conjuntos de caracteres. Las codificaciones se declaran incluyendo un comentario con formato especial en la primera o segunda línea del archivo fuente. Por ejemplo, un archivo UTF-8 puede declararse con:

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
```

Sin esta declaración de codificación, la codificación por defecto utilizada es ASCII de 7 bits. Ejecutar o importar módulos que contengan literales de cadena con caracteres de 8 bits y que no tengan una declaración de codificación dará lugar a un `DeprecationWarning` señalado por Python 2.3; en 2.4 será un error de sintaxis.

La declaración de codificación sólo afecta a los literales de cadena Unicode, que se convertirán a Unicode utilizando la codificación especificada. Ten en cuenta que los identificadores de Python siguen restringidos a caracteres ASCII, por lo que no puedes tener nombres de variables que utilicen caracteres fuera de los alfanuméricos habituales.

Ver también

PEP 263 [https://peps.python.org/pep-0263/] - Definición de las codificaciones del código fuente de Python

Escrito por Marc-André Lemburg y Martin von Löwis; realizado por Suzuki Hisao y Martin von Löwis.

PEP 273: Importar módulos desde archivos ZIP

El nuevo módulo `zipimport` añade soporte para importar módulos desde un archivo en formato ZIP. No es necesario importar el módulo explícitamente; se importará automáticamente si se añade el nombre de un archivo ZIP a `sys.path`. Por ejemplo:

```
amk@nyman:~/src/python$ unzip -l /tmp/example.zip
Archivo: /tmp/example.zip
Longitud Fecha Hora Nombre
-------- ---- ---- ----
```

```
8467 11-26-02 22:30 jwzthreading.py
-------- -------
8467 1 archivo
amk@nyman:~/src/python$ ./python
Python 2.3 (#1, 1 de agosto de 2003, 19:54:32)
>>> import sys
>>> sys.path.insert(0, '/tmp/example.zip') # Agregar arc
>>> import jwzthreading
>>> jwzthreading.__file__
'/tmp/ejemplo.zip/jwzthreading.py'
>>>
```

Una entrada en `sys.path` puede ser ahora el nombre de un archivo ZIP. El archivo ZIP puede contener cualquier tipo de ficheros, pero sólo se pueden importar los ficheros llamados `*.py`, `*.pyc`, o `*.pyo`. Si un archivo sólo contiene ficheros `*.py`, Python no intentará modificar el archivo añadiendo el correspondiente fichero `*.pyc`, lo que significa que si un archivo ZIP no contiene ficheros `*.pyc`, la importación puede ser bastante lenta.

También se puede especificar una ruta dentro del archivo para importar sólo de un subdirectorio; por ejemplo, la ruta `/tmp/example.zip/lib/` sólo importaría del subdirectorio `lib/` dentro del archivo.

Ver también

PEP 273 [https://peps.python.org/pep-0273/] - Importación de módulos desde archivos Zip

Escrito por James C. Ahlstrom, que también proporcionó una implementación. Python 2.3 sigue la especificación en **PEP 273** [https://peps.python.org/pep-0273/], pero utiliza una implementación escrita por Just van Rossum que utiliza los ganchos de importación descritos en **PEP 302** [https://peps.python.org/pep-0302/]. Vea la sección PEP 302: Nuevos ganchos de importación para una descripción de los nuevos ganchos de importación.

PEP 277: Soporte de nombres de archivo Unicode para Windows NT

En Windows NT, 2000 y XP, el sistema almacena los nombres de archivo como cadenas Unicode. Tradicionalmente, Python ha representado los nombres de archivo como cadenas de bytes, lo cual es inadecuado porque hace que algunos nombres de archivo sean inaccesibles.

Python ahora permite el uso de cadenas Unicode arbitrarias (dentro de las limitaciones del sistema de archivos) para todas las funciones que esperan nombres de archivos, en particular la función incorporada `open()`. Si se pasa una cadena Unicode a `os.listdir()`, Python ahora devuelve una lista de cadenas Unicode. Una nueva función, `os.getcwdu()`, devuelve el directorio actual como una cadena Unicode.

Las cadenas de bytes siguen funcionando como nombres de archivo, y en Windows Python las convertirá de forma transparente a Unicode utilizando la codificación `mbcs`.

Other systems also allow Unicode strings as file names but convert them to byte strings before passing them to the system, which can cause a `UnicodeError` to be raised. Applications can test whether arbitrary Unicode strings are supported as file names by checking `os.path.supports_unicode_filenames`, a Boolean value.

En MacOS, `os.listdir()` ahora puede retornar nombres de archivo Unicode.

Ver también

PEP 277 [https://peps.python.org/pep-0277/] - Soporte de nombres de archivo Unicode para Windows NT

Escrito por Neil Hodgson; realizado por Neil Hodgson, Martin von Löwis y Mark Hammond.

PEP 278: Soporte universal de nuevas

líneas

Los tres principales sistemas operativos que se utilizan hoy en día son Microsoft Windows, el sistema operativo Macintosh de Apple y los diversos derivados de Unix. Una pequeña molestia del trabajo entre plataformas es que estas tres plataformas utilizan diferentes caracteres para marcar el final de las líneas en los archivos de texto. Unix utiliza el salto de línea (carácter ASCII 10), MacOS utiliza el retorno de carro (carácter ASCII 13), y Windows utiliza una secuencia de dos caracteres de un retorno de carro más una nueva línea.

Los objetos de archivo de Python ahora pueden admitir convenciones de fin de línea distintas a la que sigue la plataforma en la que se ejecuta Python. Al abrir un archivo con el modo `'U'` o `'rU'`, se abrirá un archivo para leer en modo universal newlines. Las tres convenciones de fin de línea se traducirán a `'\n'` en las cadenas devueltas por los distintos métodos de archivo, como **`read()`** y **`readline()`**.

También se utiliza la compatibilidad universal con saltos de línea al importar módulos y al ejecutar un archivo con la función **`execfile()`**. Esto significa que los módulos de Python se pueden compartir entre los tres sistemas operativos sin necesidad de convertir los finales de línea.

Esta función puede desactivarse al compilar Python especificando la opción **`--without-universal-newlines`** al ejecutar el script **configure** de Python.

Ver también

PEP 278 [https://peps.python.org/pep-0278/] - Soporte universal de nuevas líneas

Escrito y ejecutado por Jack Jansen.

PEP 279: enumerate()

Una nueva función incorporada, **`enumerate()`**, hará que ciertos bucles sean un poco más claros. `enumerate(cosa)`, donde *cosa* es un iterador o una secuencia, retorna un iterador que retornará `(0, cosa[0])`, `(1, cosa[1])`, `(2, cosa[2])`, y así sucesivamente.

Un modismo común para cambiar cada elemento de una lista tiene el siguiente aspecto:

```
para i en rango(len(L)):
item = L[i]
# ... calcular algún resultado basado en el elemento ...
L[i] = resultado
```

Esto se puede reescribir usando `enumerate()` como:

```
para i, elemento en enumerate(L):
# ... calcular algún resultado basado en el elemento ...
L[i] = resultado
```

Ver también

PEP 279 [https://peps.python.org/pep-0279/] - La función incorporada enumerate()
Escrito y ejecutado por Raymond D. Hettinger.

PEP 282: El paquete de registro

Se ha añadido a Python 2.3 un paquete estándar para escribir registros, `logging`. Proporciona un mecanismo potente y flexible para generar salidas de registro que pueden ser filtradas y procesadas de varias maneras. Se puede utilizar un archivo de configuración escrito en un formato estándar para controlar el comportamiento de registro de un programa. Python incluye manejadores que escribirán los registros en el error estándar o en un archivo o socket, los enviarán al registro del sistema, o incluso los enviarán por correo electrónico a una dirección particular; por supuesto, también es posible escribir tus propias clases de manejadores.

La clase **`Logger`** es la clase principal. La mayoría del código de la aplicación se ocupará de uno o más objetos **`Logger`**, cada uno de ellos utilizado por un subsistema particular de la aplicación. Cada **`Logger`** se identifica con un nombre, y los nombres se organizan en una jerarquía utilizando `.` como separador de componentes. Por ejemplo, puede tener instancias **`Logger`** denominadas `server`, `server.auth` y `server.network`. Las dos últimas instancias están por debajo de `server` en la jerarquía. Esto significa que si aumenta el nivel de detalle de `server` o dirige los mensajes `server` a un controlador diferente, los cambios también se aplicarán a los registros registrados en `server.auth` y `server.network`. También hay un **`Logger`** raíz que es el padre de todos los demás registradores.

Para usos sencillos, el paquete **`logging`** contiene algunas funciones de conveniencia que siempre utilizan la raíz log:

```
import logging

logging.debug('Información de depuración')
logging.info('Mensaje informativo')
logging.warning('Advertencia: no se encontró el archivo
logging.error('Se produjo un error')
logging.critical('Error crítico: apagando el sistema')
```

Esto produce la siguiente salida:

```
ADVERTENCIA:root:Advertencia:no se encontró el archivo
ERROR:root:Se produjo un error
CRÍTICO:root:Error crítico: se está apagando
```

En la configuración predeterminada, se suprimen los mensajes informativos y de depuración y la salida se envía al error estándar. Puede habilitar la visualización de mensajes informativos y de depuración llamando al método **`setLevel()`** en el registrador raíz.

Observe el uso de operadores de formato de cadena en la llamada **`warning()`**; todas las funciones para registrar mensajes toman los argumentos `(msg, arg1, arg2, ...)` y registran la cadena resultante de `msg % (arg1, arg2, ...)`.

También hay una función **`exception()`** que registra el seguimiento más reciente. Cualquiera de las otras funciones también registrará el seguimiento si especifica un valor verdadero para el argumento de palabra clave *exc_info*.

```
def f():
try: 1/0
except: logging.exception('Problema registrado')

f()
```

Esto produce la siguiente salida:

```
ERROR:root:Problema registrado
Traceback (última llamada más reciente):
Archivo "t.py", línea 6, en f
1/0
ZeroDivisionError: división entera o módulo por cero
```

Los programas un poco más avanzados utilizarán un logger distinto del logger raíz. La función `getLogger(nombre)` se utiliza para obtener un registro en particular, creándolo si aún no existe. `getLogger(None)` retorna el logger raíz.

```
log = logging.getLogger('servidor')
...
log.info('Escuchando en el puerto %i', puerto)
...
log.critical('Disco lleno')
...
```

Los registros normalmente se propagan hacia arriba en la jerarquía, por lo que un mensaje registrado en `server.auth` también es visto por `server` y `root`, pero un **`Logger`** puede evitar esto configurando su atributo **`propagate`** en **`False`**.

El paquete **`logging`** proporciona más clases que se pueden personalizar. Cuando se le indica a una instancia **`Logger`** que registre un mensaje, crea una instancia **`LogRecord`** que se envía a cualquier cantidad de instancias **`Handler`** diferentes. Los registradores y controladores también pueden tener una lista adjunta de filtros, y cada filtro puede hacer que se ignore el `LogRecord` o puede modificar el registro antes de pasarlo. Cuando finalmente se generan, las instancias `LogRecord` se convierten en texto mediante una clase `Formatter`. Todas estas clases se pueden reemplazar por sus propias clases especialmente escritas.

Con todas estas características, el paquete `logging` debería proporcionar suficiente flexibilidad incluso para las aplicaciones más complicadas. Esto es sólo un resumen incompleto de sus características, así que por favor consulte la documentación de referencia del paquete para todos los detalles. La lectura de PEP 282 [https://peps.python.org/pep-0282/] también será útil.

Ver también

PEP 282 [https://peps.python.org/pep-0282/] - Un sistema de registro

Escrito por Vinay Sajip y Trent Mick; implementado por Vinay Sajip.

PEP 285: Un tipo booleano

Se agregó un tipo booleano a Python 2.3. Se agregaron dos nuevas constantes al módulo `__builtin__`, `True` y `False`. (Las constantes `True` y `False` se agregaron a las funciones integradas en Python 2.2.1, pero las versiones 2.2.1 simplemente se configuran en valores enteros de 1 y 0 y no son de un tipo diferente).

El objeto de tipo para este nuevo tipo se denomina `bool`; su constructor toma cualquier valor de Python y lo convierte en `True` o `False`.

```
>>> bool(1)
Verdadero
>>> bool(0)
Falso
>>> bool([])
Falso
```

```
>>> bool( (1,) )
Verdadero
```

La mayoría de los módulos de la biblioteca estándar y las funciones incorporadas se han modificado para retornar booleanos.

```
>>> obj = []
>>> hasattr(obj, 'append')
Verdadero
>>> isinstance(obj, lista)
Verdadero
>>> isinstance(obj, tupla)
Falso
```

Los booleanos de Python se añadieron con el objetivo principal de hacer el código más claro. Por ejemplo, si estás leyendo una función y te encuentras con la sentencia `return 1`, podrías preguntarte si el `1` representa un valor de verdad booleano, un índice o un coeficiente que multiplica alguna otra cantidad. Sin embargo, si la sentencia es `return True`, el significado del valor de retorno es bastante claro.

Los booleanos de Python *no* se añadieron en aras de una comprobación de tipos estricta. Un lenguaje muy estricto como Pascal también le impediría realizar aritmética con booleanos, y requeriría que la expresión en una declaración **`if`** siempre se evaluara a un resultado booleano. Python no es tan estricto y nunca lo será, como dice explícitamente **PEP 285** [https://peps.python.org/pep-0285/]. Esto significa que puede utilizar cualquier expresión en una sentencia **`if`**, incluso las que se evalúan a una lista o tupla o algún objeto aleatorio. El tipo Booleano es una subclase de la clase **`int`** por lo que la aritmética que utiliza un Booleano sigue funcionando.

```
>>> Verdadero + 1
2
>>> Falso + 1
1
>>> Falso * 75
0
```

```
>>> True * 75
75
```

Para resumir `True` and `False` en una frase: son formas alternativas de deletrear los valores enteros 1 y 0, con la única diferencia de que `str()` y `repr()` retornan las cadenas `Verdadero` y `Falso` en lugar de `1` y `0`.

Ver también

PEP 285 [https://peps.python.org/pep-0285/] - Añadir un tipo booleano
Escrito y ejecutado por GvR.

PEP 293: Llamadas de retorno para el manejo de errores del códec

Al codificar una cadena Unicode en una cadena de bytes, pueden encontrarse caracteres no codificables. Hasta ahora, Python ha permitido especificar el procesamiento del error como «estricto» (lanzando `UnicodeError`), «ignorar» (saltando el carácter), o «reemplazar» (usando un signo de interrogación en la cadena de salida), siendo «estricto» el comportamiento por defecto. Puede ser deseable especificar un procesamiento alternativo de tales errores, como insertar una referencia de carácter XML o una referencia de entidad HTML en la cadena convertida.

Python tiene ahora un marco flexible para añadir diferentes estrategias de procesamiento. Se pueden añadir nuevos manejadores de errores con `codecs.register_error()`, y los códecs pueden acceder al manejador de errores con `codecs.lookup_error()`. Se ha añadido una API en C equivalente para los códecs escritos en C. El gestor de errores obtiene la información de estado necesaria, como la cadena que se está convirtiendo, la posición en la cadena donde se ha detectado el error y la codificación de destino. El controlador puede entonces lanzar una excepción o retornar una cadena de reemplazo.

Se han implementado dos manejadores de error adicionales utilizando este marco: «backslashreplace» utiliza las comillas de barra invertida de Python para representar los caracteres no codificables y «xmlcharrefreplace» emite referencias de caracteres XML.

Ver también

PEP 293 [https://peps.python.org/pep-0293/] - Retrollamadas de manejo de errores del códec
Escrito y ejecutado por Walter Dörwald.

PEP 301: Índice de paquetes y metadatos para Distutils

La compatibilidad con el catálogo de Python, largamente solicitada, hace su primera aparición en 2.3.

El corazón del catálogo es el nuevo comando **register** de Distutils. Ejecutando `python setup.py register` se recogen los metadatos que describen un paquete, como su nombre, versión, mantenedor, descripción, etc., y se envían a un servidor de catálogo central. El catálogo resultante está disponible en https://pypi.org.

Para que el catálogo sea un poco más útil, se ha añadido un nuevo argumento de palabra clave opcional *classifiers* a la función **`setup()`** de Distutils. Se puede proporcionar una lista de cadenas de estilo Trove [http://catb.org/~esr/trove/] para ayudar a clasificar el software.

Aquí hay un ejemplo `setup.py` con clasificadores, escrito para que sea compatible con las versiones más antiguas de Distutils:

```
from distutils import core
kw = {'name': "Quixote",
      'version': "0.5.1",
      'description': "A highly Pythonic Web application
      # ...
      }
```

```
if (hasattr(core, 'setup_keywords') and
    'classifiers' in core.setup_keywords):
    kw['classifiers'] = \
        ['Topic :: Internet :: WWW/HTTP :: Dynamic Conte
         'Environment :: No Input/Output (Daemon)',
         'Intended Audience :: Developers'],

core.setup(**kw)
```

La lista completa de clasificadores se puede obtener ejecutando `python setup.py register --list-classifiers`.

Ver también

PEP 301 [https://peps.python.org/pep-0301/] - Índice de paquetes y metadatos para Distutils
Escrito y ejecutado por Richard Jones.

PEP 302: Nuevos ganchos de importación

Si bien ha sido posible escribir ganchos de importación personalizados desde que se introdujo el módulo **`ihooks`** en Python 1.3, nadie ha estado realmente satisfecho con ello porque escribir nuevos ganchos de importación es difícil y complicado. Se han propuesto varias alternativas, como los módulos **`imputil`** y **`iu`**, pero ninguna de ellas ha tenido mucha aceptación y ninguna de ellas se podía usar fácilmente desde el código C.

PEP 302 [https://peps.python.org/pep-0302/] toma prestadas ideas de sus predecesores, especialmente del módulo **`iu`** de Gordon McMillan. Se han añadido tres elementos nuevos al módulo **`sys`**:

- `sys.path_hooks` es una lista de objetos invocables; la mayoría de las veces serán clases. Cada llamada toma una cadena que contiene una ruta y retorna un objeto importador que manejará las importaciones desde esta ruta o lanza una excepción **`ImportError`** si no puede manejar esta ruta.
- `sys.path_importer_cache` almacena en caché los objetos del importador para cada ruta, por lo que `sys.path_hooks`

sólo tendrá que ser recorrido una vez para cada ruta.
- `sys.meta_path` es una lista de objetos importadores que se recorrerán antes de comprobar `sys.path`. Esta lista está inicialmente vacía, pero el código de usuario puede añadir objetos a ella. Los módulos adicionales incorporados y congelados pueden ser importados por un objeto añadido a esta lista.

Los objetos importadores deben tener un único método, `find_module(fullname, path=None)`. *fullname* será un nombre de módulo o paquete, por ejemplo `string` o `distutils.core`. **`find_module()`** debe retornar un objeto cargador que tenga un único método, `load_module(fullname)`, que cree y retorne el objeto módulo correspondiente.

Por lo tanto, el pseudocódigo de la nueva lógica de importación de Python es algo así (simplificado un poco; véase **PEP 302** [https://peps.python.org/pep-0302/] para los detalles completos):

```
para mp en sys.meta_path:
loader = mp(fullname)
si loader no es None:
<module> = loader.load_module(fullname)

para path en sys.path:
para hook en sys.path_hooks:
try:
importer = hook(path)
except ImportError:
# ImportError, así que prueba los otros ganchos de path
pass
de lo contrario:
loader = importer.find_module(fullname)
<module> = loader.load_module(fullname)

# ¡No encontrado!
raise ImportError
```

Ver también

PEP 302 [https://peps.python.org/pep-0302/] - Nuevos ganchos de importación

Escrito por Just van Rossum y Paul Moore. Implementado por Just van Rossum.

PEP 305: Archivos separados por comas

Los archivos separados por comas son un formato frecuentemente utilizado para exportar datos de bases de datos y hojas de cálculo. Python 2.3 añade un analizador de archivos separados por comas.

El formato separado por comas es engañosamente sencillo a primera vista:

```
Costos,150,200,3.95
```

Leer una línea y llamar a `line.split(',')`: ¿qué puede ser más sencillo? Pero si se añaden datos de cadena que pueden contener comas, las cosas se complican:

```
"Costos",150,200,3.95,"Incluye impuestos, envío y artícu
```

Una expresión regular grande y fea puede analizar esto, pero usar el nuevo paquete **`csv`** es mucho más sencillo:

```
importar csv

entrada = open('archivo_datos', 'rb')
lector = csv.reader(entrada)
para línea en lector:
imprimir línea
```

La función **`reader()`** acepta varias opciones diferentes. El separador de campo no se limita a la coma y se puede cambiar por cualquier carácter, al igual que los caracteres de comillas y de fin de línea.

Se pueden definir y registrar diferentes dialectos de archivos separados por comas; actualmente hay dos dialectos, ambos utilizados por Microsoft Excel. Una clase **`csv.writer`** independiente generará archivos separados por comas a partir de una sucesión de tuplas o listas, citando cadenas que contengan el delimitador.

Ver también

PEP 305 [https://peps.python.org/pep-0305/] - API de archivos CSV
Escrito y realizado por Kevin Altis, Dave Cole, Andrew McNamara, Skip Montanaro, Cliff Wells.

PEP 307: Mejoras en Pickle

Los módulos `pickle` y `cPickle` recibieron cierta atención durante el ciclo de desarrollo 2.3. En 2.2, las clases de nuevo estilo podían ser encurtidas sin dificultad, pero no de forma muy compacta; PEP 307 [https://peps.python.org/pep-0307/] cita un ejemplo trivial en el que una clase de nuevo estilo da como resultado una cadena encurtida tres veces más larga que la de una clase clásica.

La solución fue inventar un nuevo protocolo pickle. La función `pickle.dumps()` soporta desde hace tiempo una bandera de texto o binario. En la versión 2.3, esta bandera se ha redefinido, pasando de ser un booleano a un entero: 0 es el antiguo formato pickle en modo texto, 1 es el antiguo formato binario, y ahora 2 es un nuevo formato específico de 2.3. Una nueva constante, `pickle.HIGHEST_PROTOCOL`, puede utilizarse para seleccionar el protocolo más elegante disponible.

El desempaquetado ya no se considera una operación segura. `pickle` de la versión 2.2 proporcionaba ganchos para intentar evitar que se desempaquetaran clases no seguras (específicamente, un atributo `__safe_for_unpickling__`), pero ninguno de este código fue auditado y, por lo tanto, todo se eliminó en la versión 2.3. No debe desempaquetar datos no confiables en ninguna versión de Python.

Para reducir la sobrecarga de decapado para las clases de estilo nuevo, se agregó una nueva interfaz para personalizar el decapado mediante tres métodos especiales: `__getstate__()`, `__setstate__()` y `__getnewargs__()`. Consulte PEP 307 [https://peps.python.org/pep-0307/] para conocer la semántica completa de estos métodos.

Como forma de comprimir aún más los pickles, ahora es posible utilizar códigos enteros en lugar de cadenas largas para identificar las clases serializadas. La Python Software Foundation mantendrá una lista de códigos estandarizados; también hay una gama de códigos para uso privado. Actualmente no se ha especificado ningún código.

Ver también

PEP 307 [https://peps.python.org/pep-0307/] - Extensiones del protocolo pickle
Escrito y ejecutado por Guido van Rossum y Tim Peters.

Rebanadas ampliadas

Desde la versión 1.4 de Python, la sintaxis de corte admite un tercer argumento opcional «paso» o «zancada». Por ejemplo, estas son todas las sintaxis legales de Python: `L[1:10:2]`, `L[:-1:1]`, `L[::-1]`. Esto se añadió a Python a petición de los desarrolladores de Numerical Python, que utiliza ampliamente el tercer argumento. Sin embargo, los tipos de secuencias de listas, tuplas y cadenas incorporados en Python nunca han soportado esta característica, y lanzan un `TypeError` si lo intentas. Michael Hudson ha contribuido con un parche para solucionar este problema.

Por ejemplo, ahora puede extraer fácilmente los elementos de una lista que tengan índices pares:

```
>>> L = rango(10)
>>> L[::2]
[0, 2, 4, 6, 8]
```

Los valores negativos también sirven para hacer una copia de la misma lista en orden inverso:

```
>>> L[::-1]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Esto también funciona para tuplas, arrays y cadenas:

```
>>> s='abcd'
>>> s[::2]
'ac'
>>> s[::-1]
'dcba'
```

Si tienes una secuencia mutable, como una lista o un array, puedes asignar o eliminar una rebanada extendida, pero hay algunas diferencias entre la asignación a rebanadas extendidas y regulares. La asignación a una rebanada regular se puede utilizar para cambiar la longitud de la secuencia:

```
>>> a = rango(3)
>>> a
[0, 1, 2]
>>> a[1:3] = [4, 5, 6]
>>> a
[0, 4, 5, 6]
```

Las rebanadas extendidas no son tan flexibles. Cuando se asigna a una rebanada extendida, la lista a la derecha de la declaración debe contener el mismo número de elementos que la rebanada que está reemplazando:

```
>>> a = range(4)
>>> a
[0, 1, 2, 3]
>>> a[::2]
[0, 2]
>>> a[::2] = [0, -1]
>>> a
[0, 1, -1, 3]
>>> a[::2] = [0,1,2]
Traceback (última llamada más reciente):
Archivo "<stdin>", línea 1, en ?
```

```
ValueError: intento de asignar una secuencia de tamaño 3
```

La eliminación es más sencilla:

```
>>> a = rango(4)
>>> a
[0, 1, 2, 3]
>>> a[::2]
[0, 2]
>>> del a[::2]
>>> a
[1, 3]
```

Ahora también se pueden pasar objetos de corte a los métodos **`__getitem__()`** de las secuencias integradas:

```
>>> rango(10).__getitem__(sección(0, 5, 2))
[0, 2, 4]
```

O utilizar los objetos de corte directamente en los subíndices:

```
>>> rango(10)[segmento(0, 5, 2)]
[0, 2, 4]
```

Para simplificar la implementación de secuencias que admiten cortes extendidos, los objetos de corte ahora tienen un método `indices(length)` que, dada la longitud de una secuencia, devuelve una tupla `(start, stop, step)` que se puede pasar directamente a **`range()`**. **`indices()`** maneja los índices omitidos y fuera de límites de una manera consistente con los cortes regulares (¡y esta frase inocua esconde una maraña de detalles confusos!). El método está pensado para usarse de esta manera:

```
class FakeSeq:
    ...
    def calc_item(self, i):
        ...
    def __getitem__(self, item):
        if isinstance(item, slice):
            indices = item.indices(len(self))
            return FakeSeq([self.calc_item(i) for i in r
        else:
            return self.calc_item(i)
```

En este ejemplo también se puede ver que el objeto incorporado **`slice`** es ahora el objeto tipo para el tipo slice, y ya no es una función. Esto es consistente con Python 2.2, donde **`int`**, **`str`**, etc., sufrieron el mismo cambio.

Otros cambios en el lenguaje

Estos son todos los cambios que Python 2.3 introduce en el núcleo del lenguaje Python.

- La expresión **`yield`** es ahora siempre una palabra clave, como se describe en la sección PEP 255: Generadores simples de este documento.

- Se ha añadido una nueva función incorporada **`enumerate()`**, como se describe en la sección PEP 279: enumerate() de este documento.

- Se han añadido dos nuevas constantes, **`True`** y **`False`** junto con el tipo incorporado **`bool`**, como se describe en la sección PEP 285: Un tipo booleano de este documento.

- El constructor de tipo **`int()`** ahora retornará un entero largo en lugar de lanzar un **`OverflowError`** cuando una cadena o un número de punto flotante es demasiado grande para caber en un entero. Esto puede llevar al resultado paradójico de que `isinstance(int(expresión), int)` sea falso, pero parece poco probable que cause problemas en la práctica.

- Los tipos incorporados ahora soportan la sintaxis de rebanado extendida, como se describe en la sección Rebanadas ampliadas de este documento.

- Una nueva función incorporada, `suma(iterable, start=0)`, suma los elementos numéricos en el objeto iterable y retorna su suma. **`suma()`** sólo acepta números, lo que significa que no se puede utilizar para concatenar un montón de cadenas. (Contribución de Alex Martelli)

- `list.insert(pos, valor)` solía insertar *valor* al principio de la lista cuando *pos* era negativo. El comportamiento ha sido cambiado para ser consistente con la indexación de las rebanadas, así que cuando *pos* es -1 el valor será insertado antes del último elemento, y así sucesivamente.

- `list.index(value)`, que busca *valor* dentro de la lista y retorna su índice, ahora toma los argumentos opcionales *start* y *stop* para limitar la búsqueda sólo a una parte de la lista.

- Los diccionarios tienen un nuevo método, `pop(key[, *default*])`, que retorna el valor correspondiente a *key* y elimina ese par clave/valor del diccionario. Si la clave solicitada no está presente en el diccionario, se retorna *default* si está especificada y se lanza **`KeyError`** si no lo está:

```
>>> d = {1:2}
>>> d
{1: 2}
>>> d.pop(4)
Traceback (última llamada más reciente):
Archivo "stdin", línea 1, en ?
KeyError: 4
>>> d.pop(1)
2
>>> d.pop(1)
Traceback (última llamada más reciente):
Archivo "stdin", línea 1, en ?
KeyError: 'pop(): el diccionario está vacío'
>>> d
{}
>>>
```

  También hay un nuevo método de clase, `dict.fromkeys(iterable, value)`, que crea un diccionario con claves tomadas del iterador *iterable* suministrado y todos los valores establecidos a *value*, por defecto a `None`.

(Parches aportados por Raymond Hettinger)

Además, el constructor `dict()` ahora acepta argumentos de palabras clave para simplificar la creación de pequeños diccionarios:

```
>>> dict(rojo=1, azul=2, verde=3, negro=4)
{'azul': 2, 'negro': 4, 'verde': 3, 'rojo': 1}
```

(Contribución de Just van Rossum.)

- La expresión `assert` ya no comprueba la bandera `debug__`, por lo que ya no se pueden desactivar las aserciones asignando a `__debug__`. Ejecutar Python con la opción `-O` seguirá generando código que no ejecute ninguna aserción.

- La mayoría de los objetos de tipo ahora se pueden llamar, por lo que puede usarlos para crear nuevos objetos, como funciones, clases y módulos. (Esto significa que el módulo **new** puede quedar obsoleto en una versión futura de Python, porque ahora puede usar los objetos de tipo disponibles en el módulo `types`). Por ejemplo, puede crear un nuevo objeto de módulo con el siguiente código:

```
>>> tipos de importación
>>> m = tipos.ModuleType('abc','docstring')
>>> m
<módulo 'abc' (integrado)>
>>> m.__doc__
'docstring'
```

- Se ha añadido una nueva advertencia, `PendingDeprecationWarning` para indicar las características que están en proceso de ser obsoletas. La advertencia no se imprimirá por defecto. Para comprobar el uso de funciones que quedarán obsoletas en el futuro, proporcione `-Walways::PendingDeprecationWarning::` en la línea de comandos o utilice `warnings.filterwarnings()`.

- Ha comenzado el proceso de desaprobación de las excepciones basadas en cadenas, como en `lanzamiento de "Error ocurred"`. Al lanzar una cadena, ahora se activará `PendingDeprecationWarning`.

- El uso de `None` como nombre de una variable ahora resultará en una advertencia `SyntaxWarning`. En una futura versión de Python, `None` podría convertirse en una palabra clave.

- El método **`xreadlines()`** de los objetos de archivo, introducido en Python 2.1, ya no es necesario porque los archivos ahora se comportan como su propio iterador. **`xreadlines()`** se introdujo originalmente como una forma más rápida de recorrer todas las líneas de un archivo, pero ahora puedes escribir simplemente `for line in file_obj`. Los objetos de archivo también tienen un nuevo atributo **`encoding`** de solo lectura que proporciona la codificación utilizada por el archivo; las cadenas Unicode escritas en el archivo se convertirán automáticamente a bytes utilizando la codificación dada.

- El orden de resolución de métodos utilizado por las clases de nuevo estilo ha cambiado, aunque solo notarás la diferencia si tienes una jerarquía de herencia realmente complicada. Las clases clásicas no se ven afectadas por este cambio. Python 2.2 originalmente usaba una clasificación topológica de los ancestros de una clase, pero 2.3 ahora usa el algoritmo C3 como se describe en el artículo «A Monotonic Superclass Linearization for Dylan» [https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.3910]. Para entender la motivación de este cambio, lee el artículo The Python 2.3 Method Resolution Order de Michele Simionato, o lee el hilo en python-dev que comienza con el mensaje en https://mail.python.org/pipermail/python-dev/2002-October/029035.html. Samuele Pedroni fue el primero en señalar el problema y también implementó la solución codificando el algoritmo C3.

- Python ejecuta programas multiproceso alternando entre subprocesos después de ejecutar N bytecodes. El valor predeterminado de N se ha incrementado de 10 a 100 bytecodes, lo que acelera las aplicaciones de un solo subproceso al reducir la sobrecarga de conmutación. Algunas aplicaciones multiproceso pueden sufrir un tiempo de respuesta más lento, pero eso se soluciona fácilmente estableciendo el límite en un número menor utilizando `sys.setcheckinterval(N)`. El límite se puede recuperar con la nueva función **`sys.getcheckinterval()`**.

- Un cambio menor pero de gran alcance es que los nombres de los tipos de extensión definidos por los módulos incluidos con Python ahora contienen el módulo y un `'.'` delante del nombre del tipo. Por ejemplo, en Python 2.2, si creaba un socket e imprimía su **`__class__`**, obtenía este resultado:

  ```
  >>> s = socket.socket()
  >>> s.__class__
  <tipo 'socket'>
  ```

  En 2.3, se obtiene esto:

  ```
  >>> s.__class__
  <tipo '_socket.socket'>
  ```

- Se ha eliminado una de las incompatibilidades observadas entre las clases de estilo antiguo y nuevo: ahora se pueden asignar atributos **`__name__`** y **`__bases__`** a las clases de estilo nuevo. Existen algunas restricciones sobre lo que se puede asignar a **`__bases__`** similares a las relacionadas con la asignación al atributo **`__class__`** de una instancia.

Cambios en las cadenas de texto

- El operador **`in`** ahora funciona de forma diferente para las cadenas. Antes, cuando se evaluaba `X en Y` donde *X* y *Y* eran cadenas, *X* sólo podía ser un único carácter. Esto ha cambiado; *X* puede ser una cadena de cualquier longitud, y `X en Y` retornará **`True`** si *X* es una subcadena de *Y*. Si *X* es una cadena vacía, el resultado es siempre **`True`**.

```
>>> 'ab' en 'abcd'
Verdadero
>>> 'ad' en 'abcd'
Falso
>>> '' en 'abcd'
Verdadero
```

Tenga en cuenta que esto no le dice dónde empieza la subcadena; si necesita esa información, utilice el método **`find()`** string.

- Los métodos de cadena **`strip()`**, **`lstrip()`** y **`rstrip()`** ahora tienen un argumento opcional para especificar los caracteres que se eliminarán. El valor predeterminado sigue siendo eliminar todos los caracteres de espacio en blanco:

```
>>> ' abc '.strip()
'abc'
>>> '><><abc<><><>'.strip('<>')
'abc'
>>> '><><abc<><><>\n'.strip('<>')
'abc<><><>\n'
>>> u'\u4000\u4001abc\u4000'.strip(u'\u4000')
u'\u4001abc'
>>>
```

(Sugerido por Simon Brunning y aplicado por Walter Dörwald)

- Los métodos de cadena **`startswith()`** y **`endswith()`** ahora aceptan números negativos para los parámetros *start* y *end*.

- Otro nuevo método de cadena es **`zfill()`**, originalmente una función del módulo **`string`**. **`zfill()`** rellena una cadena numérica con ceros a la izquierda hasta que alcanza el ancho especificado. Tenga en cuenta que el operador `%` sigue siendo más flexible y potente que **`zfill()`**.

```
>>> '45'.zfill(4)
```

```
'0045'
>>> '12345'.zfill(4)
'12345'
>>> 'tonto'.zfill(6)
'0tonto'
```

(Contribución de Walter Dörwald.)

- Se ha añadido un nuevo tipo de objeto, **`basestring`**. Tanto las cadenas de 8 bits como las cadenas Unicode heredan de este tipo, por lo que `isinstance(obj, basestring)` devolverá **`True`** para cualquier tipo de cadena. Es un tipo completamente abstracto, por lo que no se pueden crear instancias de **`basestring`**.

- Las cadenas internas ya no son inmortales y ahora serán recolectadas de la forma habitual cuando la única referencia a ellas sea desde el diccionario interno de cadenas internas. (Implementado por Oren Tirosh)

Optimizaciones

- La creación de instancias de clases de estilo nuevo se ha hecho mucho más rápida; ¡ahora son más rápidas que las clases clásicas!
- El método **`sort()`** de objetos de lista ha sido reescrito ampliamente por Tim Peters, y la implementación es significativamente más rápida.
- La multiplicación de números enteros grandes es ahora mucho más rápida gracias a una implementación de la multiplicación Karatsuba, un algoritmo que escala mejor que el $O(n^2)$ requerido para el algoritmo de multiplicación de nivel primario. (Parche original de Christopher A. Craig y reelaborado significativamente por Tim Peters).
- El opcode `SET_LINENO` ha desaparecido. Esto puede proporcionar un pequeño aumento de velocidad, dependiendo de la idiosincrasia de su compilador. Vea la sección Otros cambios y correcciones para una explicación más larga. (Eliminado por Michael Hudson)
- Los objetos **`xrange()`** ahora tienen su propio iterador, lo

que hace que `for i in xrange(n)` sea ligeramente más rápido que `for i in range(n)`. (Parche de Raymond Hettinger).
- Se han realizado una serie de pequeños reajustes en varios puntos conflictivos para mejorar el rendimiento, como por ejemplo alinear una función o eliminar algo de código. (Implementado principalmente por GvR, pero mucha gente ha contribuido con cambios individuales)

El resultado neto de las optimizaciones de la versión 2.3 es que Python 2.3 ejecuta el benchmark pystone alrededor de un 25% f más rápido que Python 2.2.

Módulos nuevos, mejorados y obsoletos

Como es habitual, la biblioteca estándar de Python ha recibido una serie de mejoras y correcciones de errores. Aquí hay una lista parcial de los cambios más notables, ordenados alfabéticamente por nombre de módulo. Consulte el archivo `Misc/NEWS` en el árbol de fuentes para obtener una lista más completa de los cambios, o busque en los registros de CVS para obtener todos los detalles.

- El módulo **`array`** soporta ahora matrices de caracteres Unicode que utilizan el carácter de formato `'u`. Las matrices también soportan ahora el uso del operador de asignación `+=` para añadir el contenido de otra matriz, y el operador de asignación `*=` para repetir una matriz. (Contribución de Jason Orendorff)

- El módulo **`bsddb`** ha sido reemplazado por la versión 4.1.6 del paquete PyBSDDB [https://pybsddb.sourceforge.net], proporcionando una interfaz más completa para las características transaccionales de la biblioteca BerkeleyDB.

  La versión anterior del módulo ha cambiado de nombre a **`bsddb185`** y ya no se compila automáticamente; deberá editar `Modules/Setup` para habilitarlo. Tenga en cuenta que el nuevo paquete **`bsddb`** está pensado para ser compatible con el módulo anterior, así que asegúrese de informar de los errores si descubre alguna incompatibilidad.

Al actualizar a Python 2.3, si el nuevo intérprete se compila con una nueva versión de la biblioteca BerkeleyDB subyacente, es casi seguro que tendrá que convertir los archivos de su base de datos a la nueva versión. Puede hacerlo con bastante facilidad con los nuevos scripts `db2pickle.py` y `pickle2db.py` que encontrará en el directorio `Tools/scripts` de la distribución. Si ya ha estado usando el paquete PyBSDDB y lo ha importado como **`bsddb3`**, tendrá que cambiar sus declaraciones `import` para importarlo como **`bsddb`**.

- El nuevo módulo **`bz2`** es una interfaz para la biblioteca de compresión de datos bz2. Los datos comprimidos con bz2 suelen ser más pequeños que los correspondientes datos comprimidos con **`zlib`**. (Contribución de Gustavo Niemeyer)

- Se ha añadido un conjunto de tipos de fecha/hora estándar en el nuevo módulo **`datetime`**. Consulte la siguiente sección para obtener más detalles.

- La clase **`Extension`** de Distutils ahora admite un argumento de constructor adicional llamado *depends* para enumerar archivos de origen adicionales de los que depende una extensión. Esto permite que Distutils vuelva a compilar el módulo si se modifica alguno de los archivos de dependencia. Por ejemplo, si `sampmodule.c` incluye el archivo de encabezado `sample.h`, crearía el objeto **`Extension`** de la siguiente manera:

  ```
  ext = Extensión("samp",
  fuentes=["sampmodule.c"],
  depende=["sample.h"])
  ```

  La modificación de `sample.h` haría que el módulo se recompilara. (Contribución de Jeremy Hylton)

- Otros cambios menores en Distutils: ahora verifica las variables de entorno **`CC`**, **`CFLAGS`**, **`CPP`**, **`LDFLAGS`** y **`CPPFLAGS`**, usándolas para anular las configuraciones en la configuración de Python (contribuido por Robert Weber).

- Anteriormente, el módulo `doctest` solo buscaba casos de prueba en las cadenas de documentación de métodos y funciones públicos, pero ahora también examina los casos de prueba privados. La función `DocTestSuite()` crea un objeto `unittest.TestSuite` a partir de un conjunto de pruebas `doctest`.

- La nueva función `gc.get_referents(object)` retorna una lista de todos los objetos referenciados por *object*.

- El módulo `getopt` obtuvo una nueva función, `gnu_getopt()`, que admite los mismos argumentos que la función `getopt()` existente, pero utiliza el modo de escaneo de estilo GNU. El módulo `getopt()` existente deja de procesar opciones tan pronto como se encuentra un argumento que no es una opción, pero en el modo de estilo GNU el procesamiento continúa, lo que significa que se pueden mezclar opciones y argumentos. Por ejemplo:

  ```
  >>> getopt.getopt(['-f', 'nombrearchivo', 'salida',
  ([('-f', 'nombrearchivo')], ['salida', '-v'])
  >>> getopt.gnu_getopt(['-f', 'nombrearchivo', 'sali
  ([('-f', 'nombrearchivo'), ('-v', '')], ['salida'])
  ```

  (Contribución de Peter Åstrand.)

- Los módulos `grp`, `pwd` y `resource` retornan ahora tuplas mejoradas:

  ```
  >>> importar grp
  >>> g = grp.getgrnam('amk')
  >>> g.gr_name, g.gr_gid
  ('amk', 500)
  ```

- El módulo `gzip` ahora puede manejar archivos de más de 2 GiB.

- El nuevo módulo `heapq` contiene una implementación de un algoritmo de cola de montón. Un montón es una estructura de datos similar a una matriz que mantiene los elementos en un orden parcialmente ordenado de modo que, para cada índice *k*, `heap[k] <= heap[2*k+1]` y `heap[k] <= heap[2*k+2]`, se elimina rápidamente el elemento más pequeño y se inserta un elemento nuevo mientras se mantiene la propiedad de montón *O*(log *n*). (Consulte https://xlinux.nist.gov/dads//HTML/priorityque.html para obtener más información sobre la estructura de datos de la cola de prioridad).

El módulo **`heapq`** proporciona las funciones **`heappush()`** y **`heappop()`** para agregar y eliminar elementos mientras se mantiene la propiedad de montón sobre algún otro tipo de secuencia mutable de Python. Aquí hay un ejemplo que utiliza una lista de Python:

```
>>> import heapq
>>> montón = []
>>> para el elemento en [3, 7, 5, 11, 1]:
... heapq.heappush(montón, elemento)
...
>>> montón
[1, 3, 5, 11, 7]
>>> heapq.heappop(montón)
1
>>> heapq.heappop(montón)
3
>>> montón
[5, 7, 11]
```

(Contribución de Kevin O'Connor.)

- El entorno de desarrollo integrado IDLE ha sido actualizado utilizando el código del proyecto IDLEfork (http://idlefork.sourceforge.net). La característica más notable es que el código que se está desarrollando se ejecuta ahora en un subproceso, lo que significa que ya no es necesario realizar operaciones manuales de `reload()`. El código central de IDLE ha sido incorporado a la biblioteca estándar como el paquete **`idlelib`**.
- El módulo **`imaplib`** ahora admite IMAP sobre SSL. (Contribuido por Piers Lauder y Tino Lange.)

- `itertools` contiene varias funciones útiles para usar con iteradores, inspiradas en varias funciones proporcionadas por los lenguajes ML y Haskell. Por ejemplo, `itertools.ifilter(predicate, iterator)` devuelve todos los elementos del iterador para los que la función `predicate()` devuelve `True`, y `itertools.repeat(obj, N)` devuelve `obj` *N* veces. Hay varias otras funciones en el módulo; consulte la documentación de referencia del paquete para obtener más detalles. (Contribuido por Raymond Hettinger).

- Dos nuevas funciones en el módulo `math`, `degrees(rads)` y `radians(degs)`, convierten entre radianes y grados. Otras funciones del módulo `math`, como `math.sin()` y `math.cos()`, siempre han requerido valores de entrada medidos en radianes. Además, se agregó un argumento opcional *base* a `math.log()` para facilitar el cálculo de logaritmos para bases distintas de `e` y `10`. (Contribuido por Raymond Hettinger.)

- Se agregaron varias funciones POSIX nuevas (`getpgid()`, `killpg()`, `lchown()`, `loadavg()`, `major()`, `makedev()`, `minor()` y `mknod()`) al módulo `posix` que sirve de base al módulo `os`. (Contribución de Gustavo Niemeyer, Geert Jansen y Denis S. Otkidach).

- En el módulo `os`, la familia de funciones `*stat()` ahora puede informar fracciones de segundo en una marca de tiempo. Dichas marcas de tiempo se representan como números flotantes, similares al valor devuelto por `time.time()`.

  Durante las pruebas, se descubrió que algunas aplicaciones fallan si las marcas de tiempo son números de punto flotante. Por cuestiones de compatibilidad, al usar la interfaz de tupla de `stat_result`, las marcas de tiempo se representarán como números enteros. Al usar campos con nombre (una característica introducida por primera vez en Python 2.2), las marcas de tiempo aún se representan como números enteros, a menos que se invoque `os.stat_float_times()` para habilitar los valores de retorno de punto flotante:

```
>>> os.stat("/tmp").st_mtime
1034791200
>>> os.stat_float_times(Verdadero)
>>> os.stat("/tmp").st_mtime
1034791200.6335014
```

En Python 2.4, el valor predeterminado cambiará para devolver siempre flotantes.

Los desarrolladores de aplicaciones deberían habilitar esta función solo si todas sus bibliotecas funcionan correctamente cuando se enfrentan a marcas de tiempo de punto flotante o si utilizan la API de tuplas. Si se utiliza, la función debería activarse a nivel de aplicación en lugar de intentar habilitarla según el uso.

- El módulo `optparse` contiene un nuevo analizador para argumentos de línea de comandos que puede convertir valores de opción a un tipo de Python en particular y generará automáticamente un mensaje de uso. Consulte la siguiente sección para obtener más detalles.

- El antiguo módulo **`linuxaudiodev`**, que nunca se documentó, ha quedado obsoleto y se ha añadido una nueva versión denominada **`ossaudiodev`**. El módulo ha cambiado de nombre porque los controladores de sonido OSS se pueden utilizar en plataformas distintas a Linux y la interfaz también se ha ordenado y actualizado de varias maneras. (Contribución de Greg Ward y Nicholas FitzRoy-Dale).

- El nuevo módulo `platform` contiene una serie de funciones que intentan determinar varias propiedades de la plataforma en la que se está ejecutando. Hay funciones para obtener la arquitectura, el tipo de CPU, la versión del sistema operativo Windows e incluso la versión de distribución de Linux. (Contribución de Marc-André Lemburg.)

- Los objetos del analizador proporcionados por el módulo `pyexpat` ahora pueden almacenar en búfer datos de caracteres de manera opcional, lo que genera menos llamadas al controlador de datos de caracteres y, por lo tanto, un rendimiento más rápido. Si se configura el atributo `buffer_text` del objeto del analizador en `True`, se habilitará el almacenamiento en búfer.

- Se agregó la función `sample(population, k)` al módulo `random`. *population* es una secuencia o un objeto `xrange` que contiene los elementos de una población y `sample()` elige elementos *k* de la población sin reemplazar los elementos elegidos. *k* puede ser cualquier valor hasta `len(population)`. Por ejemplo:

```
>>> days = ['Mo', 'Tu', 'We', 'Th', 'Fr', 'St', 'Sn
>>> random.sample(days, 3) # Elige 3 elementos
['St', 'Sn', 'Th']
>>> random.sample(days, 7) # Elige 7 elementos
['Tu', 'Th', 'Mo', 'We', 'St', 'Fr', 'Sn']
>>> random.sample(days, 7) # Elige 7 nuevamente
['We', 'Mo', 'Sn', 'Fr', 'Tu', 'St', 'Th']
>>> random.sample(days, 8) # No se pueden elegir oc
Traceback (última llamada más reciente):
Archivo "<stdin>", línea 1, en ?
Archivo "random.py", línea 414, en ejemplo
genera ValueError, "muestra mayor que la población"
ValueError: muestra mayor que la población
>>> random.sample(xrange(1,10000,2), 10) # Elige di
[3407, 3805, 1505, 7023, 2401, 2267, 9733, 3151, 80
```

  El módulo `random` ahora usa un nuevo algoritmo, el Mersenne Twister, implementado en C. Es más rápido y más estudiado que el algoritmo anterior.

  (Parches aportados por Raymond Hettinger)

- El módulo `readline` también obtuvo una serie de nuevas funciones: `get_history_item()`, `get_current_history_length()` y `redisplay()`.

- Los módulos `rexec` y `Bastion` se han declarado inactivos y los intentos de importarlos fallarán con un `RuntimeError`. Las clases de nuevo estilo proporcionan nuevas formas de salir del entorno de ejecución restringido que ofrece **`rexec`** y nadie tiene interés en arreglarlas ni tiempo para hacerlo. Si tiene aplicaciones que utilizan **`rexec`**, reescríbalas para utilizar algo diferente.

(Seguir usando Python 2.2 o 2.1 no hará que sus aplicaciones sean más seguras porque hay errores conocidos en el módulo **`rexec`** en esas versiones. Para repetir: si está usando **`rexec`**, deje de usarlo inmediatamente).

- El módulo **`rotor`** ha quedado obsoleto porque se cree que el algoritmo que utiliza para el cifrado no es seguro. Si necesita cifrado, utilice uno de los diversos módulos AES de Python que están disponibles por separado.

- El módulo **`shutil`** obtuvo una función `move(src, dest)` que mueve recursivamente un archivo o directorio a una nueva ubicación.

- Se agregó soporte para un manejo de señal POSIX más avanzado al **`signal`**, pero luego se eliminó nuevamente, ya que resultó imposible hacerlo funcionar de manera confiable en todas las plataformas.

- El módulo **`socket`** ahora admite tiempos de espera. Puede llamar al método `settimeout(t)` en un objeto de socket para establecer un tiempo de espera de *t* seconds. Subsequent socket operations that take longer than *t* segundos para completar, abortará y lanzará una excepción **`socket.timeout`**.

  La implementación del tiempo de espera original fue realizada por Tim O'Malley. Michael Gilfix lo integró en el módulo Python **`socket`** y lo guió a través de una extensa revisión. Después de que se registró el código, Guido van Rossum reescribió partes del mismo. (Este es un buen ejemplo de un proceso de desarrollo colaborativo en acción).

- En Windows, el módulo **`socket`** ahora se envía con compatibilidad con Secure Sockets Layer (SSL).

- El valor de la macro C **`PYTHON_API_VERSION`** ahora se expone en el nivel de Python como `sys.api_version`. La excepción actual se puede eliminar llamando a la nueva función **`sys.exc_clear()`**.

- El nuevo módulo **`tarfile`** permite leer y escribir en: programa: *tar* -format archivos de almacenamiento. (Contribución de Lars Gustäbel.)

- El nuevo módulo **`textwrap`** contiene funciones para encapsular cadenas que contienen párrafos de texto. La función `wrap(text, width)` toma una cadena y devuelve una lista que contiene el texto dividido en líneas de un ancho no mayor que el elegido. La función `fill(text, width)` devuelve una sola cadena, reformateada para que quepa en líneas de un ancho no mayor que el elegido. (Como puede adivinar, **`fill()`** está construido sobre **`wrap()`**. Por ejemplo:

  ```
  >>> import textwrap
  >>> párrafo = "Ni un ápice, desafiamos el augurio:
  >>> textwrap.wrap(párrafo, 60)
  ["Ni un ápice, desafiamos el augurio: hay una provi
  "la caída de un gorrión. Si es ahora, no está por v
  ...]
  >>> print textwrap.fill(párrafo, 35)
  Ni un ápice, desafiamos el augurio: hay
  una providencia especial en la caída de
  un gorrión. Si es ahora, no está por venir; si no e
  será ahora; si no es ahora, aún
  vendrá: la preparación lo es todo.
  >>>
  ```

  El módulo también contiene una clase **`TextWrapper`** que implementa la estrategia de ajuste de texto. Tanto la clase **`TextWrapper`** como las funciones **`wrap()`** y **`fill()`** admiten una serie de argumentos de palabras clave adicionales para ajustar el formato; consulte la documentación del módulo para obtener más detalles. (Contribuido por Greg Ward).

- Los módulos **`thread`** y **`threading`** ahora tienen módulos complementarios, **`dummy_thread`** y **`dummy_threading`**, que proporcionan una implementación sin hacer nada de la interfaz del módulo **`thread`** para plataformas donde no se admiten subprocesos. La intención es simplificar los módulos que admiten subprocesos (aquellos que dependen de subprocesos para ejecutarse) colocando el siguiente código en la parte superior:

  ```
  Intente:
  importar subprocesos como _threading
  excepto ImportError:
  importar dummy_threading como _threading
  ```

  En este ejemplo, se utiliza **`_threading`** como nombre del módulo para dejar claro que el módulo que se utiliza no es necesariamente el módulo **`threading`** real. El código puede llamar a funciones y usar clases en **`_threading`** independientemente de si se admiten subprocesos o no, lo que evita una declaración **`if`** y hace que el código sea un poco más claro. Este módulo no hará que el código multiproceso se ejecute mágicamente sin subprocesos; el código que espera a que otro subproceso regrese o haga algo simplemente se bloqueará para siempre.

- La función **`strptime()`** del módulo **`time`** ha sido una molestia durante mucho tiempo porque utiliza la implementación **`strptime()`** de la biblioteca de la plataforma C, y a veces las diferentes plataformas tienen errores extraños. Brett Cannon contribuyó con una implementación portátil que está escrita en Python puro y debería comportarse de manera idéntica en todas las plataformas.

- El nuevo módulo **`timeit`** ayuda a medir cuánto tardan en ejecutarse fragmentos de código Python. El archivo `timeit.py` se puede ejecutar directamente desde la línea de comandos, o se puede importar y utilizar directamente la clase **`Timer`** del módulo. A continuación, se incluye un breve ejemplo que determina si es más rápido convertir una cadena de 8 bits a Unicode añadiéndole una cadena Unicode vacía o utilizando la función **`unicode()`**:

```
import timeit

timer1 = timeit.Timer('unicode("abc")')
timer2 = timeit.Timer('"abc" + u""')

# Ejecutar tres pruebas
print timer1.repeat(repeat=3, number=100000)
print timer2.repeat(repeat=3, number=100000)

# En mi computadora portátil, esto genera:
# [0.36831796169281006, 0.37441694736480713, 0.3530
# [0.17574405670166016, 0.18193507194519043, 0.1756
```

- El módulo **`Tix`** ha recibido varias correcciones de errores y actualizaciones para la versión actual del paquete Tix.

- El módulo **`Tkinter`** ahora funciona con una versión de Tcl habilitada para subprocesos. El modelo de subprocesos de Tcl requiere que solo se acceda a los widgets desde el subproceso en el que se crearon; los accesos desde otro subproceso pueden hacer que Tcl entre en pánico. Para ciertas interfaces de Tcl, **`Tkinter`** ahora evitará esto automáticamente cuando se acceda a un widget desde un subproceso diferente mediante la ordenación de un comando, pasándolo al subproceso correcto y esperando los resultados. Otras interfaces no se pueden manejar automáticamente, pero **`Tkinter`** ahora generará una excepción en dicho acceso para que al menos pueda averiguar sobre el problema. Consulte https://mail.python.org/pipermail/python-dev/2002-December/031107.html para obtener una explicación más detallada de este cambio. (Implementado por Martin von Löwis).

- La llamada a métodos Tcl a través de **`_tkinter`** ya no devuelve solo cadenas. En cambio, si Tcl devuelve otros objetos, esos objetos se convierten en su equivalente de Python, si existe uno, o se encapsulan con un objeto

`_tkinter.Tcl_Obj` si no existe un equivalente de Python. Este comportamiento se puede controlar a través del método **`wantobjects()`** de los objetos **`tkapp`**.

Al utilizar **`_tkinter`** a través del módulo **`Tkinter`** (como lo hacen la mayoría de las aplicaciones de Tkinter), esta función siempre está activada. No debería causar problemas de compatibilidad, ya que Tkinter siempre convertiría los resultados de cadenas a tipos de Python cuando fuera posible.

Si se encuentran incompatibilidades, se puede restaurar el comportamiento anterior configurando la variable **`wantobjects`** en el módulo **`Tkinter`** como falsa antes de crear el primer objeto **`tkapp`**.

```
importar Tkinter
Tkinter.wantobjects = 0
```

Cualquier rotura causada por este cambio debe notificarse como un error.

- El módulo **`UserDict`** tiene una nueva clase **`DictMixin`** que define todos los métodos de diccionario para las clases que ya tienen una interfaz de mapeo mínima. Esto simplifica enormemente la escritura de clases que deben ser sustituibles por diccionarios, como las clases del módulo **`shelve`**.

  Al agregar la combinación como superclase, se obtiene la interfaz de diccionario completa siempre que la clase defina **`__getitem__()`**, **`__setitem__()`**, **`__delitem__()`** y **`keys()`**. Por ejemplo:

```
>>> import UserDict
>>> class SeqDict(UserDict.DictMixin):
... """Similar a un diccionario implementado con li
... def __init__(self):
... self.keylist = []
... self.valuelist = []
... def __getitem__(self, key):
... try:
... i = self.keylist.index(key)
... except ValueError:
... raise KeyError
... return self.valuelist[i]
... def __setitem__(self, key, value):
... try:
... i = self.keylist.index(key)
... self.valuelist[i] = value
... except ValueError:
... self.keylist.append(key)
... self.valuelist.append(value)
... def __delitem__(self, key):
... try:
... i = self.keylist.index(key)
... except ValueError:
... raise KeyError
... self.keylist.pop(i)
... self.valuelist.pop(i)
... def keys(self):
... return list(self.keylist)
...
>>> s = SeqDict()
>>> dir(s) # Verificar que se implementen otros mét
['__cmp__', '__contains__', '__delitem__', '__doc__
'__init__', '__iter__', '__len__', '__module__', '_
'__setitem__', 'clear', 'get', 'has_key', 'items',
'iterkeys', 'itervalues', 'keylist', 'keys', 'pop',
'setdefault', 'update', 'valuelist', 'valores']
```

(Parches aportados por Raymond Hettinger)

- La implementación de DOM en `xml.dom.minidom` ahora puede generar salida XML en una codificación particular al proporcionar un argumento de codificación opcional a los métodos `toxml()` y `toprettyxml()` de los nodos DOM.

- El módulo **`xmlrpclib`** ahora admite una extensión XML-RPC para manejar valores de datos nulos, como `None` de Python. Los valores nulos siempre se admiten al desagrupar una respuesta XML-RPC. Para generar solicitudes que contengan `None`, debe proporcionar un valor verdadero para el parámetro *allow_none* al crear una instancia **`Marshaller`**.

- El nuevo módulo **`DocXMLRPCServer`** permite escribir servidores XML-RPC autodocumentados. Ejecútelo en modo de demostración (como programa) para verlo en acción. Al apuntar el navegador web al servidor RPC se genera documentación de estilo pydoc; al apuntar xmlrpclib al servidor se pueden invocar los métodos reales. (Contribuido por Brian Quinlan.)

- Se ha agregado soporte para nombres de dominio internacionalizados (RFCs 3454, 3490, 3491 y 3492). La codificación «idna» se puede utilizar para convertir entre un nombre de dominio Unicode y la codificación compatible con ASCII (ACE) de ese nombre.

  ```
  >{}>{}> u"www.Alliancefrançaise.nu".encode("idna")
  'www.xn--alliancefranaise-npb.nu'
  ```

  El módulo **`socket`** también se ha ampliado para convertir de forma transparente los nombres de host Unicode a la versión ACE antes de pasarlos a la biblioteca C. Los módulos que se ocupan de los nombres de host, como **`httplib`** y **`ftplib`**, también admiten nombres de host Unicode; **`httplib`** también envía encabezados HTTP `Host` utilizando la versión ACE del nombre de dominio. **`urllib`** admite URL Unicode con nombres de host que no sean ASCII siempre que la parte `path` de la URL sea solo ASCII.

  Para implementar este cambio, se han agregado el módulo **`stringprep`**, la herramienta `mkstringprep` y la codificación `punycode`.

Tipo de fecha / hora

Se agregaron tipos de fecha y hora adecuados para expresar marcas de tiempo como módulo **`datetime`**. Los tipos no admiten diferentes calendarios ni muchas funciones sofisticadas, y solo se ciñen a los conceptos básicos de la representación del tiempo.

Los tres tipos principales son: **`date`**, que representa un día, un mes y un año; **`time`**, que consta de horas, minutos y segundos; y **`datetime`**, que contiene todos los atributos de **`date`** y **`time`**. También existe una clase **`timedelta`** que representa las diferencias entre dos puntos en el tiempo, y la lógica de la zona horaria se implementa mediante clases que heredan de la clase abstracta **`tzinfo`**.

Puede crear instancias de **`date`** y **`time`** proporcionando argumentos de palabras clave al constructor adecuado, por ejemplo, `datetime.date(year=1972, month=10, day=15)`, o utilizando uno de los diversos métodos de clase. Por ejemplo, el método de clase **`today()`** devuelve la fecha local actual.

Una vez creadas, las instancias de las clases de fecha / hora son inmutables. Hay varios métodos para producir cadenas formateadas a partir de objetos:

```
>>> import datetime as dt
>>> now = dt.datetime.now()
>>> now.isoformat()
'2002-12-30T21:27:03.994956'
>>> now.ctime()  # Only available on date, datetime
'Mon Dec 30 21:27:03 2002'
>>> now.strftime('%Y %d %b')
'2002 30 Dec'
```

El método **`replace()`** permite modificar uno o más campos de una instancia **`date`** o **`datetime`**, devolviendo una nueva instancia:

```
>>> d = dt.datetime.now()
>>> d
datetime.datetime(2002, 12, 30, 22, 15, 38, 827738)
>>> d.replace(year=2001, hour=12)
datetime.datetime(2001, 12, 30, 12, 15, 38, 827738)
>>>
```

Las instancias se pueden comparar, codificar y convertir en cadenas (el resultado es el mismo que el de **`isoformat()`**). Las instancias `date` y `datetime` se pueden restar entre sí y agregar a las instancias `timedelta`. La característica que falta más es que no hay compatibilidad con la biblioteca estándar para analizar cadenas y obtener un **`date`** o **`datetime`**.

Para obtener más información, consulte la documentación de referencia del módulo. (Contribuido por Tim Peters.)

El módulo optparse

El módulo `getopt` proporciona un análisis sencillo de los argumentos de la línea de comandos. El nuevo módulo `optparse` (originalmente llamado Optik) proporciona un análisis de línea de comandos más elaborado que sigue las convenciones de Unix, crea automáticamente la salida para **`--help`** y puede realizar diferentes acciones para diferentes opciones.

Comienza creando una instancia de `OptionParser` y diciéndole cuáles son las opciones de tu programa.

```
import sys
from optparse import OptionParser

op = OptionParser()
op.add_option('-i', '--input',
action='store', type='string', dest='input',
help='establecer nombre de archivo de entrada')
op.add_option('-l', '--length',
action='store', type='int', dest='length',
help='establecer longitud máxima de salida')
```

Luego, el análisis de una línea de comando se realiza llamando al método `parse_args()`.

```
opciones, args = op.parse_args(sys.argv[1:])
imprimir opciones
imprimir argumentos
```

Esto devuelve un objeto que contiene todos los valores de las opciones y una lista de cadenas que contienen los argumentos restantes.

Invocar el script con los distintos argumentos ahora funciona como era de esperar. Tenga en cuenta que el argumento de longitud se convierte automáticamente en un número entero.

```
$ ./python opt.py -i data arg1
<Valores en 0x400cad4c: {'entrada': 'datos', 'longitud':
['arg1']
$ ./python opt.py --input=data --length=4
<Valores en 0x400cad2c: {'entrada': 'datos', 'longitud':
[]
$
```

El mensaje de ayuda se genera automáticamente para usted:

```
$ ./python opt.py --help
uso: opt.py [opciones]

opciones:
-h, --help muestra este mensaje de ayuda y sale
-iINPUT, --input=INPUT
establece el nombre del archivo de entrada
-lLENGTH, --length=LENGTH
establece la longitud máxima de salida
$
```

Consulte la documentación del módulo para obtener más detalles.

Optik fue escrito por Greg Ward, con sugerencias de los lectores de Getopt SIG.

Pymalloc: un asignador de objetos especializado

Pymalloc, un asignador de objetos especializado escrito por Vladimir Marangozov, fue una característica agregada a Python 2.1. Pymalloc está diseñado para ser más rápido que el sistema **`malloc()`** y tener menos sobrecarga de memoria para los patrones de asignación típicos de los programas Python. El asignador utiliza la función `malloc()` de C para obtener grandes grupos de memoria y luego satisface las solicitudes de memoria más pequeñas de estos grupos.

En 2.1 y 2.2, pymalloc era una función experimental y no estaba habilitada de forma predeterminada; tenía que habilitarlo explícitamente al compilar Python proporcionando la opción `--with-pymalloc` al script: program: *configure*. En 2.3, pymalloc ha tenido más mejoras y ahora está habilitado de forma predeterminada; tendrá que suministrar `--without-pymalloc` para deshabilitarlo.

Este cambio es transparente para el código escrito en Python; sin embargo, pymalloc puede exponer errores en las extensiones C. Los autores de los módulos de extensión C deben probar su código con pymalloc habilitado, ya que algunos códigos incorrectos pueden causar volcados de núcleo en tiempo de ejecución.

Hay un error particularmente común que causa problemas. Hay una serie de funciones de asignación de memoria en la API C de Python que anteriormente solo eran alias para `malloc()` y `free()` de la biblioteca C, lo que significa que si llama accidentalmente a funciones que no coinciden, el error no se notará. Cuando el asignador de objetos está habilitado, estas funciones ya no son alias de `malloc()` y `free()`, y llamar a la función incorrecta para liberar memoria puede generar un volcado de memoria. Por ejemplo, si la memoria se asignó utilizando `PyObject_Malloc()`, debe liberarse utilizando `PyObject_Free()`, no `free()`. Algunos módulos incluidos con Python entraron en conflicto con esto y tuvieron que ser reparados; sin duda hay más módulos de terceros que tendrán el mismo problema.

Como parte de este cambio, las confusas interfaces múltiples para asignar memoria se han consolidado en dos familias de API. La memoria asignada a una familia no debe manipularse con funciones de la otra familia. Hay una familia para asignar fragmentos de memoria y otra familia de funciones específicamente para asignar objetos Python.

- Para asignar y liberar una porción de memoria no

distinguida, use la familia de «memoria sin procesar»: `PyMem_Malloc()`, `PyMem_Realloc()` y `PyMem_Free()`.
- La familia de «memoria de objetos» es la interfaz para la instalación de pymalloc descrita anteriormente y está sesgada hacia un gran número de asignaciones «pequeñas»: `PyObject_Malloc()`, `PyObject_Realloc()` y `PyObject_Free()`.
- Para asignar y liberar objetos de Python, utilice la familia de «objetos» `PyObject_New`, `PyObject_NewVar` y `PyObject_Del()`.

Gracias al gran trabajo de Tim Peters, pymalloc en 2.3 también proporciona funciones de depuración para detectar sobrescrituras de memoria y liberaciones duplicadas en ambos módulos de extensión y en el propio intérprete. Para habilitar este soporte, compile una versión de depuración del intérprete de Python ejecutando: programa: *configure* con **`--with-pydebug`**.

Para ayudar a los escritores de extensiones, se distribuye un archivo de encabezado `Misc/pymemcompat.h` con la fuente a Python 2.3 que permite que las extensiones de Python usen las interfaces 2.3 para la asignación de memoria mientras se compila con cualquier versión de Python desde la 1.5.2. Debería copiar el archivo de la distribución fuente de Python y empaquetarlo con la fuente de su extensión.

Ver también

https://hg.python.org/cpython/file/default/Objects/obmalloc.c

Para obtener todos los detalles de la implementación de pymalloc, consulte los comentarios en la parte superior del archivo `Objects/obmalloc.c` en el código fuente de Python. El enlace anterior apunta al archivo dentro del navegador SVN de python.org.

Cambios en la API de Build y C

Los cambios en el proceso de compilación de Python y en la API de

C incluyen:

- La implementación de detección de ciclos utilizada por la recolección de basura ha demostrado ser estable, por lo que ahora se ha hecho obligatoria. Ya no puede compilar Python sin él, y el cambio **`--with-cycle-gc`** a: program: *configure* ha sido eliminado.
- Python ahora se puede construir opcionalmente como una biblioteca compartida (`libpython2.3.so`) proporcionando **`--enable-shared`** cuando se ejecuta el script Python: program: *configure*. (Contribuido por Ondrej Palkovsky.)
- Las macros **`DL_EXPORT`** y **`DL_IMPORT`** ahora están en desuso. Las funciones de inicialización para los módulos de extensión de Python ahora deben declararse usando la nueva macro **`PyMODINIT_FUNC`**, mientras que el núcleo de Python generalmente usará las macros **`PyAPI_FUNC`** y **`PyAPI_DATA`**.
- El intérprete se puede compilar sin ninguna cadena de documentación para las funciones y módulos incorporados proporcionando **`--without-doc-strings`** al script: program: *configure*. Esto hace que el ejecutable de Python sea un 10% más pequeño, pero también significa que no puede obtener ayuda para las funciones integradas de Python. (Contribución de Gustavo Niemeyer.)
- La macro **`PyArg_NoArgs()`** ahora está en desuso y el código que la usa debe cambiarse. Para Python 2.2 y versiones posteriores, la tabla de definición de métodos puede especificar la marca **`METH_NOARGS`**, lo que indica que no hay argumentos, y luego se puede eliminar la verificación de argumentos. Si la compatibilidad con las versiones anteriores a la 2.2 de Python es importante, el código podría usar `PyArg_ParseTuple(args, "")` en su lugar, pero esto será más lento que usar **`METH_NOARGS`**.
- **`PyArg_ParseTuple()`** acepta caracteres de nuevo formato para varios tamaños de enteros sin signo: `B` para unsigned char, `H` para unsigned short int, `I` para unsigned int, y `K` para unsigned long long.
- Se agregó una nueva función, `PyObject_DelItemString(mapping, char *key)`, como abreviatura de `PyObject_DelItem(mapping,`

`PyString_New(key))`.

- Los objetos de archivo ahora administran su búfer de cadena interno de manera diferente, incrementándolo exponencialmente cuando es necesario. Esto da como resultado que las pruebas de referencia en `Lib/test/test_bufio.py` se aceleren considerablemente (de 57 segundos a 1,7 segundos, según una medición).
- Ahora es posible definir métodos estáticos y de clase para un tipo de extensión C configurando los indicadores **`METH_CLASS`** o **`METH_STATIC`** en la estructura **`PyMethodDef`** de un método.
- Python ahora incluye una copia del código fuente del analizador XML de Expat, eliminando cualquier dependencia de una versión del sistema o instalación local de Expat.
- Si asigna objetos de tipo de forma dinámica en su extensión, debe tener en cuenta un cambio en las reglas relacionadas con los atributos **`__module__`** y **`__name__`**. En resumen, deberá asegurarse de que el diccionario del tipo contenga una clave `'__module__'`; hacer que el nombre del módulo sea la parte del nombre del tipo que precede al punto final ya no tendrá el efecto deseado. Para obtener más detalles, lea la documentación de referencia de la API o el código fuente.

Cambios específicos del puerto

El soporte para un puerto para el sistema operativo IBM OS/2 que utiliza el entorno de ejecución EMX se fusionó con el árbol de código fuente principal de Python. EMX es una capa de emulación POSIX sobre las API del sistema OS/2. El puerto de Python para EMX intenta soportar toda la capacidad similar a POSIX expuesta por el entorno de ejecución EMX y, en su mayoría, lo logra; **`fork()`** y **`fcntl()`** están restringidos por las limitaciones de la capa de emulación subyacente. El puerto estándar para OS/2, que utiliza el compilador Visual Age de IBM, también obtuvo soporte para semánticas de importación que distinguen entre mayúsculas y minúsculas como parte de la integración del puerto EMX en CVS. (Contribuido por Andrew MacIntyre.)

En MacOS, la mayoría de los módulos de la caja de herramientas tienen vínculos débiles para mejorar la compatibilidad con versiones anteriores. Esto significa que los módulos ya no dejarán de cargarse si falta una rutina en la versión actual del sistema operativo. En su lugar, llamar a la rutina que falta lanzará una excepción. (Contribuido por Jack Jansen.)

Los archivos de especificaciones de RPM, que se encuentran en el directorio `Misc/RPM/` en la distribución fuente de Python, se actualizaron para la versión 2.3. (Contribución de Sean Reifschneider.)

Otras plataformas nuevas ahora compatibles con Python incluyen AtheOS (http://www.atheos.cx/), GNU / Hurd y OpenVMS.

Otros cambios y correcciones

Como de costumbre, hubo un montón de otras mejoras y correcciones de errores esparcidas por todo el árbol de fuentes. Una búsqueda en los registros de cambios de CVS encuentra que se aplicaron 523 parches y se corrigieron 514 errores entre Python 2.2 y 2.3. Es probable que ambas cifras estén subestimadas.

Algunos de los cambios más notables son:

- Si se establece la variable de entorno **`PYTHONINSPECT`**, el intérprete de Python ingresará al indicador interactivo después de ejecutar un programa de Python, como si Python hubiera sido invocado con la opción **`-i`**. La variable de entorno se puede configurar antes de ejecutar el intérprete de Python, o el programa Python puede configurarla como parte de su ejecución.

- El script `regrtest.py` ahora proporciona una forma de permitir «todos los recursos excepto *foo*». Un nombre de recurso pasado a la opción **`-u`** ahora se puede prefijar con un guión (`'-'`) para significar «eliminar este recurso». Por ejemplo, la opción "`-uall,-bsddb`" podría usarse para habilitar el uso de todos los recursos excepto `bsddb`.

- Las herramientas utilizadas para crear la documentación ahora funcionan tanto en Cygwin como en Unix.

- Se ha eliminado el código de operación `SET_LINENO`. En la noche de los tiempos, este código de operación era necesario para producir números de línea en rastreos y admitir funciones de rastreo (para, por ejemplo, **`pdb`**). Desde Python 1.5, los números de línea en los rastreos se han calculado utilizando un mecanismo diferente que funciona con «python -O». Para Python 2.3, Michael Hudson implementó un esquema similar para determinar cuándo llamar a la función de seguimiento, eliminando por completo la necesidad de `SET_LINENO`.

  Sería difícil detectar cualquier diferencia resultante del código Python, aparte de una ligera aceleración cuando Python se ejecuta sin **`-O`**.

  Las extensiones de C que acceden al campo **`f_lineno`** de los objetos de marco deberían llamar a `PyCode_Addr2Line(f->f_code, f->f_lasti)`. Esto tendrá el efecto adicional de hacer que el código funcione como se desea con «python -O» en versiones anteriores de Python.

  Una característica nueva y elegante es que las funciones de seguimiento ahora pueden asignarse al atributo **`f_lineno`** de los objetos de marco, lo que cambia la línea que se ejecutará a continuación. Se ha agregado un comando `jump` al depurador **`pdb`** que aprovecha esta nueva característica. (Implementado por Richie Hindle).

Portar a Python 2.3

Esta sección enumera los cambios descritos anteriormente que pueden requerir cambios en su código:

- **`yield`** ahora es siempre una palabra clave; si se usa como nombre de variable en su código, se debe elegir un nombre diferente.

- Para cadenas, *X* and *Y*, ASDF00 now works if *X* tiene más de un carácter.

- El constructor de tipo **`int()`** ahora retornará un entero largo en lugar de lanzar un **`OverflowError`** cuando una cadena o un número de punto flotante es demasiado grande para caber en un entero. Esto puede llevar al resultado paradójico de que `isinstance(int(expresión), int)` sea falso, pero parece poco probable que cause problemas en la práctica.

- Si tiene cadenas Unicode que contienen caracteres de 8 bits, debe declarar la codificación del archivo (UTF-8, Latin-1 o lo que sea) agregando un comentario en la parte superior del archivo. Consulte la sección PEP 263: Codificación del código fuente para obtener más información.

- La llamada a métodos Tcl a través de **`_tkinter`** ya no devuelve solo cadenas. En cambio, si Tcl devuelve otros objetos, esos objetos se convierten en su equivalente de Python, si existe uno, o se encapsulan con un objeto **`_tkinter.Tcl_Obj`** si no existe un equivalente de Python.

- Grandes literales octales y hexadecimales como `0xffffffff` ahora activan un **`FutureWarning`**. Actualmente se almacenan como números de 32 bits y dan como resultado un valor negativo, pero en Python 2.4 se convertirán en enteros largos positivos.

  Hay varias formas de corregir esta advertencia. Si realmente necesita un número positivo, simplemente agregue un `L` al final del literal. Si está tratando de obtener un entero de 32 bits con bits bajos establecidos y ha usado previamente una expresión como `~(1 << 31)`, probablemente sea más claro comenzar con todos los bits establecidos y borrar los bits superiores deseados. Por ejemplo, para borrar solo el bit superior (bit 31), puede escribir `0xffffffffL &~(1L<<31)`.

- Ya no puede deshabilitar las aserciones asignando a `__debug__`.

- La función **`setup()`** de Distutils ha obtenido varios argumentos de palabras clave nuevos, como *depends*. Las versiones anteriores de Distutils se interrumpirán si se les pasan palabras clave desconocidas. Una solución es comprobar la presencia de la nueva función **`get_distutil_options()`** en su `setup.py` y utilizar las nuevas palabras clave solo con una versión de Distutils que las admita:

```
from distutils import core

kw = {'sources': 'foo.c', ...}
if hasattr(core, 'get_distutil_options'):
    kw['depends'] = ['foo.h']
ext = Extension(**kw)
```

- El uso de `None` como nombre de una variable ahora resultará en una advertencia **`SyntaxWarning`**. En una futura versión de Python, `None` podría convertirse en una palabra clave.

- Los nombres de los tipos de extensión definidos por los módulos incluidos con Python ahora contienen el módulo y un `'.'` delante del nombre del tipo.

Agradecimientos

El autor desea agradecer a las siguientes personas por ofrecer sugerencias, correcciones y ayuda con varios borradores de este artículo: Jeff Bauer, Simon Brunning, Brett Cannon, Michael Chermside, Andrew Dalke, Scott David Daniels, Fred L. Drake, Jr., David Fraser, Kelly Gerber, Raymond Hettinger, Michael Hudson, Chris Lambert, Detlef Lannert, Martin von Löwis, Andrew MacIntyre, Lalo Martins, Chad Netzer, Gustavo Niemeyer, Neal Norwitz, Hans Nowak, Chris Reedy, Francesco Ricciardi, Vinay Sajip, Neil Schemenauer, Roman Suzi, Jason Tishler, Just van Rossum.

Qué hay de nuevo en Python 2.2

Autor:

A.M. Kuchling

Introducción

Este artículo explica las nuevas características en Python 2.2.2, publicado el 14 de octubre de 2002. Python 2.2.2 es una versión de corrección de errores de Python 2.2, lanzada originalmente el 21 de diciembre de 2001.

Python 2.2 se puede considerar como la «versión de limpieza». Hay algunas características como los generadores e iteradores que son completamente nuevas, pero la mayoría de los cambios, aunque sean significativos y de gran alcance, tienen como objetivo limpiar las irregularidades y los rincones oscuros del diseño del lenguaje.

Este artículo no procura proporcionar una especificación completa de las nuevas características, pero en su lugar proporciona una descripción general conveniente. Para más detalles, deberías consultar la documentación de Python 2.2, como Python Library Reference [https://docs.python.org/2.2/lib/lib.html] y Python Reference Manual [https://docs.python.org/2.2/ref/ref.html]. Si quieres comprender la justificación completa de la implementación y el diseño de un cambio, consultar la PEP para conocer una característica nueva en particular.

PEPs 252 y 253: Cambios de tipo y clase

Los cambios más grandes y de mayor alcance en Python 2.2 son el modelo de objetos y clases de Python. Los cambios deben ser compatibles con versiones anteriores, por lo que es probable que tu código continuará ejecutándose sin cambios, pero los cambios proporcionan algunas capacidades nuevas increíbles. Antes de comenzar esta, la sección más larga y complicada de este artículo, brindaré una descripción general de los cambios y ofreceré algunos comentarios.

Hace mucho tiempo escribí una página web en la que enumeraba los defectos de diseño de Python. Uno de los defectos más importantes era que resulta imposible crear subclases de los tipos de Python implementados en C. En particular, no es posible crear subclases de los tipos integrados, por lo que no se pueden crear subclases de, por ejemplo, listas para añadirles un único método útil. El módulo `UserList` proporciona una clase que admite todos los métodos de las listas y que se puede subclasificar aún más, pero hay mucho código C que espera una lista de Python normal y no acepta una instancia de `UserList`.

Python 2.2 corrige esto y, en el proceso, agrega algunas capacidades nuevas interesantes. Un breve resumen:

- Puedes subclasificar tipos incorporados como listas e incluso enteros, y tus subclases deberían funcionar en todos los lugares que requieran el tipo original.
- Ahora es posible definir métodos estáticos y de clase, además de los métodos de instancia disponibles en versiones anteriores de Python.
- También es posible llamar automáticamente a métodos al acceder o configurar un atributo de instancia mediante un nuevo mecanismo llamado *properties*. Muchos usos de `__getattr__()` se pueden reescribir para utilizar propiedades en su lugar, lo que hace que el código resultante sea más simple y rápido. Como pequeño beneficio adicional, los atributos ahora también pueden tener cadenas de documentación.
- La lista de atributos legales para una instancia se puede limitar a un conjunto particular usando *slots*, lo que hace posible protegerse contra errores tipográficos y quizás hacer posibles más optimizaciones en versiones futuras de Python.

Algunos usuarios han expresado preocupación por todos estos cambios. Claro, dicen, las nuevas características son ordenadas y se prestan a todo tipo de trucos que no eran posibles en versiones anteriores de Python, pero también hacen que el lenguaje sea más complicado. Algunas personas han dicho que siempre han recomendado Python por su simplicidad, y sienten que su simplicidad se está perdiendo.

Personalmente. pienso que no hay que preocuparse. Muchas de las nuevas características son bastante esotéricas, y puedes escribir mucho código de Python sin tener que estar al tanto de ellas. Escribir una clase simple no es más difícil de lo que nunca fue, así que no necesitas molestarte en aprender o enseñarlos a menos que realmente sean necesarios. Algunas tareas muy complicadas que antes solo eran posibles desde C ahora serán posibles en Python puro, y en mi opinión, esto es todo para mejor.

No voy a intentar cubrir todas los casos de las esquinas y los pequeños cambios que fueron necesarios para hacer que las nuevas características funcionen. En su lugar, esta sección pintará solo a grandes rasgos. Consultar la sección Enlaces relacionados, «Enlaces relacionados», para más fuentes de información sobre el nuevo modelo de objetos de Python 2.2.

Clases antiguas y nuevas

Primero, debes saber que realmente Python 2.2 tiene dos tipos de clases: clases clásicas o de estilo antiguo y clases de estilo nuevo. El modelo de clase de estilo antiguo exactamente es el mismo que el modelo de clase en versiones anteriores de Python. Todas las nuevas características descritas en esta sección se aplican solo a las clases de estilo nuevo. Esta divergencia no está destinada a durar para siempre; eventualmente las clases de estilo antiguo se eliminarán, posiblemente en Python 3.0.

Entonces, ¿cómo defines una clase de estilo nuevo? Lo haces subclasificando una clases de estilo nuevo existente. La mayoría de los tipos integrados de Python, como enteros, listas, diccionarios e incluso archivos, ahora son clases de estilo nuevo. También se agregó una clase de estilo nuevo llamada `object`, la clase base para todos los tipos integrados, por lo que si ningún tipo integrado es apropiado, puedes solo subclasificar `object`:

```
clase C(objeto):
def __init__(self):
...
...
```

Esto significa que las instrucciones `class` que no tienen ninguna clase base siempre son clases clásicas en Python 2.2. (En realidad, también puede cambiar esto configurando una variable a nivel de módulo llamada `__metaclass__` — consulte PEP 253 [https://peps.python.org/pep-0253/] para obtener más detalles — pero es más fácil simplemente subclasificar `object`).

Los objetos de tipo para los tipos integrados están disponibles como incorporados, nombrados mediante un truco inteligente. Python siempre ha tenido funciones incorporadas llamadas `int()`, `float()` y `str()`. En la versión 2.2, ya no son funciones, sino objetos de tipo que se comportan como fábricas cuando se les llaman.

```
>>> int
<tipo 'int'>
>>> int('123')
123
```

Para completar el conjunto de tipos, se han añadido nuevos objetos de tipo, como `dict()` y `file()`. A continuación, se muestra un ejemplo más interesante, en el que se añade un método `lock()` a los objetos de archivo:

```
clase LockableFile(archivo):
def lock (self, operación, longitud=0, inicio=0, origen=
import fcntl
return fcntl.lockf(self.fileno(), operación,
longitud, inicio, origen)
```

El módulo `posixfile`, ahora obsoleto, contenía una clase que emulaba todos los métodos de un objeto de archivo y también agregaba un método `lock()`, pero esta clase no podía pasarse a funciones internas que esperaban un archivo integrado, algo que es posible con nuestro nuevo `LockableFile`.

Descriptores

En versiones anteriores de Python, no había una forma consistente de descubrir qué atributos y métodos eran compatibles con un objeto. Existían algunas convenciones informales, como definir los atributos **`__members__`** y **`__methods__`** que eran listas de nombres, pero a menudo el autor de un tipo de extensión o una clase no se molestaba en definirlos. Se podía recurrir a inspeccionar el **`__dict__`** de un objeto, pero cuando se utilizaba la herencia de clase o un gancho **`__getattr__()`** arbitrario, esto podía seguir siendo inexacto.

La única gran idea que subyace al nuevo modelo de clases es que se ha formalizado una API para describir los atributos de un objeto usando *descriptors*. Los descriptores especifican el valor de un atributo, indicando si es un método o un campo. Con la API de un descriptor, los métodos estáticos y de clase se vuelven posibles, así como constructos más exóticos.

Los descriptores de atributos son objetos que viven dentro de los objetos de clase y tienen algunos atributos propios:

- **`__name__`** es el nombre del atributo.
- **`__doc__`** es el docstring del atributo.
- `__get__(object)` es un método que recupera el valor del atributo de *object*.
- `__set__(object, value)` establece el atributo de *object* en *value*.
- `__delete__(object, value)` elimina el atributo *value* de *object*.

Por ejemplo, cuando escribes `obj.x`, los pasos que realmente Python realiza son:

```
descriptor = obj.__class__.x
descriptor.__get__(obj)
```

Para los métodos, **`descriptor.__get__`** devuelve un objeto temporal que se puede llamar y envuelve la instancia y el método que se llamará en él. Esta es también la razón por la que ahora son posibles los métodos estáticos y los métodos de clase; tienen descriptores que envuelven solo el método, o el método y la clase. Como breve explicación de estos nuevos tipos de métodos, los métodos estáticos no se pasan a la instancia y, por lo tanto, se parecen a las funciones normales. A los métodos de clase se les pasa la clase del objeto, pero no el objeto en sí. Los métodos estáticos y de clase se definen de la siguiente manera:

```
clase C(objeto):
def f(arg1, arg2):
...
f = método estático(f)

def g(cls, arg1, arg2):
...
g = método de clase(g)
```

La función **`staticmethod()`** toma la función **`f()`** y la devuelve envuelta en un descriptor para que pueda almacenarse en el objeto de clase. Se podría esperar que existiera una sintaxis especial para crear dichos métodos (`def static f`, `defstatic f()` o algo similar), pero aún no se ha definido dicha sintaxis; eso se ha dejado para futuras versiones de Python.

También se implementan más características nuevas, como ranuras y propiedades, como nuevos tipos de descriptores, y no es difícil escribir una clase de descriptor que haga algo nuevo. Por ejemplo, sería posible escribir una clase de descriptor que hiciera posible escribir condiciones previas al estilo Eiffel y posteriores para un método. Una clase que usó esta característica podría definirse así:

```
from eiffel import eiffelmethod

class C(object):
def f(self, arg1, arg2):
# La función actual
...
def pre_f(self):
# Verificar condiciones previas
...
def post_f(self):
```

```
# Verificar condiciones posteriores
...

f = eiffelmethod(f, pre_f, post_f)
```

Tenga en cuenta que una persona que utilice el nuevo **`eiffelmethod()`** no tiene por qué entender nada sobre descriptores. Por eso creo que las nuevas características no aumentan la complejidad básica del lenguaje. Habrá algunos expertos que necesitarán saber sobre él para escribir **`eiffelmethod()`** o ZODB o lo que sea, pero la mayoría de los usuarios simplemente escribirán código sobre las bibliotecas resultantes e ignorarán los detalles de implementación.

Herencia múltiple: la regla del diamante

La herencia múltiple también se ha hecho más útil al cambiar las reglas bajo las cuales se resuelven los nombres. Considera este conjunto de clases (diagrama tomado de **PEP 253** [https://peps.python.org/pep-0253/] de Guido van Rossum):

```
clase A:
^ ^ def save(self): ...
/ \
/ \
/ \
/ \
clase B clase C:
^ ^ def save(self): ...
\ /
\ /
\ /
\ /
clase D
```

La regla de búsqueda para las clases clásicas es simple pero no muy inteligente; las clases base se buscan primero en profundidad, de izquierda a derecha. Una referencia a **`D.save()`** buscará las clases **`D`**, **`B`** y luego **`A`**, donde se encontraría y devolvería **`save()`**. **`C.save()`** nunca se encontraría. Esto es malo, porque si el método

`save()` de **`C`** está guardando algún estado interno específico de **`C`**, no llamarlo hará que ese estado nunca se guarde.

Las clases de estilo nuevo siguen un algoritmo diferente que es más complicado de explicar, pero hace lo correcto en esta situación. (Toma en cuenta que Python 2.3 cambia este algoritmo a uno que produce los mismos resultados en la mayoría de los casos, pero produce resultados más útiles para gráficos de herencia realmente complicados.)

1. Enumere todas las clases base siguiendo la regla de búsqueda clásica e incluya una clase varias veces si se la visita repetidamente. En el ejemplo anterior, la lista de clases visitadas es [**`D`**, **`B`**, **`A`**, **`C`**, **`A`**].
2. Examine la lista en busca de clases duplicadas. Si encuentra alguna, elimine todas las instancias excepto una y deje la *last* en la lista. En el ejemplo anterior, la lista se convierte en [**`D`**, **`B`**, **`C`**, **`A`**] después de eliminar los duplicados.

Siguiendo esta regla, al hacer referencia a **`D.save()`** se devolverá **`C.save()`**, que es el comportamiento que buscamos. Esta regla de búsqueda es la misma que sigue Common Lisp. Una nueva función incorporada, **`super()`**, proporciona una forma de acceder a las superclases de una clase sin tener que volver a implementar el algoritmo de Python. La forma más utilizada será `super(class, obj)`, que devuelve un objeto de superclase enlazado (no el objeto de clase real). Esta forma se utilizará en métodos para llamar a un método en la superclase; por ejemplo, el método **`save()`** de **`D`** se vería así:

```
clase D (B,C):
def save (self):
# Llamar a la superclase .save()
super(D, self).save()
# Guardar la información privada de D aquí
...
```

También **`super()`** puede retornar objetos de superclase no vinculados cuando se llama como `super(class)` o `super(class1, class2)`, pero probablemente esto no sea útil a menudo.

Acceso a atributos

Una buena cantidad de clases sofisticadas de Python definen ganchos para el acceso a atributos mediante **`__getattr__()`**; lo más común es que esto se haga por conveniencia, para que el código sea más legible al asignar automáticamente un acceso a atributos como `obj.parent` a una llamada de método como `obj.get_parent`. Python 2.2 agrega algunas formas nuevas de controlar el acceso a atributos.

Primero, `__getattr__(attr_name)` aún es compatible con las clases de nuevo estilo, y nada al respecto ha cambiado. Como antes, se llamará cuando se intente acceder a `obj.foo` y no se encuentre ningún atributo llamado `foo` en el diccionario de la instancia.

Las clases de nuevo estilo también admiten un nuevo método, `__getattribute__(attr_name)`. La diferencia entre los dos métodos es que **`__getattribute__()`** se llama *always* siempre que se accede a un atributo, mientras que el antiguo **`__getattr__()`** solo se llama si `foo` no se encuentra en el diccionario de la instancia.

Sin embargo, el soporte de Python 2.2 para *properties* será a menudo una forma más sencilla de atrapar referencias de atributos. Escribir un método **`__getattr__()`** es complicado porque para evitar la recursión no se pueden utilizar accesos a atributos regulares dentro de ellos, y en su lugar hay que jugar con el contenido de **`__dict__`**. Los métodos **`__getattr__()`** también terminan siendo llamados por Python cuando comprueba otros métodos como **`__repr__()`** o **`__coerce__()`**, y por lo tanto tienen que ser escritos con esto en mente. Finalmente, llamar a una función en cada acceso a un atributo da como resultado una pérdida de rendimiento considerable.

`property` es un nuevo tipo integrado que incluye tres funciones que obtienen, establecen o eliminan un atributo y una cadena de documentación. Por ejemplo, si desea definir un atributo **`size`** que se calcule, pero que también se pueda configurar, puede escribir:

```
clase C(objeto):
def get_size (self):
```

```
result = ... computation ...
return result
def set_size (self, size):
... calcula algo en función del tamaño
y establece el estado interno de forma adecuada ...

# Define una propiedad. El método 'eliminar este atribut
# se define como None, por lo que el atributo
# no se puede eliminar.
size = property(get_size, set_size,
None,
"Tamaño de almacenamiento de esta instancia")
```

Sin duda, esto es más claro y más fácil de escribir que un par de métodos **`__getattr__()`**/**`__setattr__()`** que comprueban el atributo **`size`** y lo gestionan de forma especial mientras recuperan todos los demás atributos del **`__dict__`** de la instancia. Los accesos a **`size`** también son los únicos que tienen que realizar el trabajo de llamar a una función, por lo que las referencias a otros atributos se ejecutan a su velocidad habitual.

Finalmente, es posible restringir la lista de atributos que se pueden referenciar en un objeto usando el nuevo atributo de clase **`__slots__`**. Los objetos de Python por lo general son muy dinámicos; en cualquier momento es posible definir un nuevo atributo en una instancia haciendo simplemente `obj.new_attr=1`. Una clase de estilo nuevo puede definir un atributo de clase llamado **`__slots__`** para limitar los atributos legales a un conjunto particular de nombres. Un ejemplo hará esto claro:

```
>>> class C(object):
... __slots__ = ('template', 'name')
...
>>> obj = C()
>>> print obj.template
None
>>> obj.template = 'Test'
>>> print obj.template
Test
```

```
>>> obj.newattr = None
Traceback (última llamada más reciente):
Archivo "<stdin>", línea 1, en ?
AttributeError: el objeto 'C' no tiene el atributo 'newa
```

Toma en cuenta cómo obtienes un `AttributeError` en el intento de asignar a un atributo que no aparece en `__slots__`.

Enlaces relacionados

Esta sección solo ha sido una descripción rápida de las nuevas características, brindando una explicación suficiente para comenzar a programar, pero muchos detalles se han simplificado o ignorado. ¿Dónde deberías ir para obtener una imagen más completa?

Descriptor Guide es una introducción extensa y tutorial a las características del descriptor, escrita por Guido van Rossum. Si mi descripción le ha abierto el apetito, lea este tutorial a continuación, porque brinda mucho más detalle sobre las nuevas características y sigue siendo bastante fácil de leer.

A continuación, hay dos PEPs relevantes, **PEP 252** [https://peps.python.org/pep-0252/] y **PEP 253** [https://peps.python.org/pep-0253/]. **PEP 252** [https://peps.python.org/pep-0252/] se titula «Hacer que los tipos se parezcan más a las clases», y cubre la API del descriptor. **PEP 253** [https://peps.python.org/pep-0253/] se titula «Subtipado de tipos incorporados», y describe los cambios de los objetos de tipo que hacen posible el subtipo de objetos incorporados. **PEP 253** [https://peps.python.org/pep-0253/] es la PEP más complicada de las dos, y en algunos puntos las explicaciones necesarias de tipos y meta-tipos pueden causar que tu cabeza explote. Ambas PEPs se escribieron e implementaron por Guido van Rossum con la asistencia sustancial del resto del equipo de Zope Corp.

Finalmente, está la máxima autoridad: el código fuente. La mayoría de la maquinaria para el manejo de tipos está en `Objects/typeobject.c`, pero solo debes recurrir a él después de que se hayan agotado todas las demás vías, incluida la publicación de una pregunta en python-list o python-dev.

PEP 234: Iteradores

Otra adición significativa a la versión 2.2 es una interfaz de iteración tanto a nivel de C como de Python. Los objetos pueden definir cómo pueden ser iterados por quienes los llaman.

En las versiones de Python hasta la 2.1, la forma habitual de hacer que funcione `for item in obj` es definir un método **`__getitem__()`** que se parezca a esto:

```
def __getitem__(self, index):
return <siguiente elemento>
```

`__getitem__()` se utiliza de forma más adecuada para definir una operación de indexación en un objeto, de modo que pueda escribir `obj[5]` para recuperar el sexto elemento. Es un poco engañoso cuando se utiliza esto solo para admitir bucles **`for`**. Considere un objeto similar a un archivo que desea ser recorrido en bucle; el parámetro *index* esencialmente no tiene sentido, ya que la clase probablemente asume que se realizará una serie de llamadas **`__getitem__()`** con *index* incrementándose en uno cada vez. En otras palabras, la presencia del método **`__getitem__()`** no significa que el uso de `file[5]` para acceder aleatoriamente al sexto elemento funcionará, aunque realmente debería funcionar.

En Python 2.2, la iteración se puede implementar por separado y los métodos **`__getitem__()`** se pueden limitar a las clases que realmente admiten el acceso aleatorio. La idea básica de los iteradores es simple. Se utiliza una nueva función incorporada, `iter(obj)` o `iter(C, sentinel)`, para obtener un iterador. `iter(obj)` devuelve un iterador para el objeto *obj*, mientras que `iter(C, sentinel)` devuelve un iterador que invocará el objeto invocable *C* hasta que devuelva *sentinel* para indicar que el iterador ha finalizado.

Las clases de Python pueden definir un método **`__iter__()`**, que debe crear y devolver un nuevo iterador para el objeto; si el objeto es su propio iterador, este método puede devolver simplemente `self`. En particular, los iteradores normalmente serán sus propios iteradores. Los tipos de extensión implementados en C pueden implementar una función **`tp_iter`** para devolver un iterador, y los tipos de extensión que quieran comportarse como iteradores pueden definir una función **`tp_iternext`**.

Entonces, después de todo esto, ¿qué hacen realmente los iteradores? Tienen un método obligatorio, **`next()`**, que no toma argumentos y retorna el siguiente valor. Cuando no hay más valores que retornar, la llamada a **`next()`** debería lanzar la excepción **`StopIteration`**.

```
>>> L = [1,2,3]
>>> i = iter(L)
>>> print i
<iterator object at 0x8116870>
>>> i.next()
1
>>> i.next()
2
>>> i.next()
3
>>> i.next()
Traceback (última llamada más reciente):
Archivo "<stdin>", línea 1, en ?
StopIteration
>>>
```

En la versión 2.2, la declaración **`for`** de Python ya no espera una secuencia, sino algo para lo que **`iter()`** devolverá un iterador. Para mayor comodidad y compatibilidad con versiones anteriores, se construye automáticamente un iterador para las secuencias que no implementan **`__iter__()`** o una ranura **`tp_iter`**, por lo que `for i in [1,2,3]` seguirá funcionando. Siempre que el intérprete de Python recorra una secuencia, se ha modificado para utilizar el protocolo de iterador. Esto significa que puede hacer cosas como esta:

```
>>> L = [1,2,3]
>>> i = iter(L)
>>> a,b,c = i
>>> a,b,c
```

```
(1, 2, 3)
```

Se ha añadido soporte de iteradores a algunos de los tipos básicos de Python. Llamar a **`iter()`** sobre un diccionario retornará un iterador que hace un bucle sobre sus claves:

```
>>> m = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May':
...      'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov':
>>> for key in m: print key, m[key]
...
Mar 3
Feb 2
Aug 8
Sep 9
May 5
Jun 6
Jul 7
Jan 1
Apr 4
Nov 11
Dec 12
Oct 10
```

Ese es simplemente el comportamiento predeterminado. Si desea iterar sobre claves, valores o pares clave/valor, puede llamar explícitamente a los métodos **`iterkeys()`**, **`itervalues()`** o **`iteritems()`** para obtener un iterador adecuado. En un cambio menor relacionado, el operador **`in`** ahora funciona en diccionarios, por lo que `key in dict` ahora es equivalente a `dict.has_key(key)`.

Los archivos también proporcionan un iterador, que llama al método **`readline()`** hasta que no hay más líneas en el archivo. Esto significa que ahora puede leer cada línea de un archivo utilizando un código como este:

```
for line in file:
    # do something for each line
    ...
```

Tenga en cuenta que sólo puede avanzar en un iterador; no hay forma de obtener el elemento anterior, reiniciar el iterador o hacer una copia del mismo. Un objeto iterador podría proporcionar estas capacidades adicionales, pero el protocolo iterador sólo requiere un método `next()`.

Ver también

PEP 234 [https://peps.python.org/pep-0234/] - Iteradores
Escrito por Ka-Ping Yee y GvR; implementado por el equipo de Python Labs, principalmente por GvR y Tim Peters.

PEP 255: Generadores simples

Los generadores son otra novedad que interactúa con la introducción de los iteradores.

Sin duda estás familiarizado con cómo funcionan las llamadas a funciones en Python o C. Cuando llamas a una función, ésta obtiene un espacio de nombres privado donde se crean sus variables locales. Cuando la función llega a una declaración `return`, las variables locales se destruyen y el valor resultante se retorna a quien la llamó. Una llamada posterior a la misma función obtendrá un nuevo conjunto de variables locales. Pero, ¿qué pasaría si las variables locales no se tiraran al salir de una función? ¿Qué pasaría si pudieras reanudar la función donde la dejaste? Esto es lo que proporcionan los generadores; se puede pensar en ellos como funciones reanudables.

Este es el ejemplo más sencillo de una función generadora:

```
def generate_ints(N):
    for i in range(N):
        yield i
```

Se ha introducido una nueva palabra clave, `yield`, para los generadores. Cualquier función que contenga una declaración `yield` es una función generadora; esto es detectado por el compilador de código de bits de Python que compila la función especialmente como resultado. Debido a la introducción de una nueva palabra clave, los generadores deben ser explícitamente habilitados en un módulo incluyendo una declaración `from __future__ import generators` cerca de la parte superior del código fuente del módulo. En Python 2.3 esta declaración será innecesaria.

Cuando se llama a una función generadora, ésta no retorna un único valor, sino que retorna un objeto generador que soporta el protocolo de los iteradores. Al ejecutar la sentencia **`yield`**, el generador retorna el valor de `i`, de forma similar a una sentencia **`return`**. La gran diferencia entre **`yield`** y una sentencia **`return`** es que al llegar a una sentencia **`yield`** se suspende el estado de ejecución del generador y se conservan las variables locales. En la siguiente llamada al método `next()` del generador, la función se reanudará la ejecución inmediatamente después de la sentencia **`yield`**. (Por razones complicadas, la sentencia **`yield`** no está permitida dentro del bloque **`try`** de una sentencia **`try`**…`**`finally`**; lea PEP 255 [https://peps.python.org/pep-0255/] para una explicación completa de la interacción entre **`yield`** y las excepciones)

Este es un ejemplo de uso del generador **`generate_ints()`**:

```
>>> gen = generate_ints(3)
>>> gen
<objeto generador en 0x8117f90>
>>> gen.next()
0
>>> gen.next()
1
>>> gen.next()
2
>>> gen.next()
Traceback (última llamada más reciente):
Archivo "<stdin>", línea 1, en ?
Archivo "<stdin>", línea 2, en generate_ints
StopIteration
```

También podrías escribir `for i in generate_ints(5)`, o `a,b,c = generate_ints(3)`.

Dentro de una función generadora, la sentencia **`return`** sólo puede usarse sin un valor, y señala el final de la procesión de valores; después el generador no puede retornar más valores. **`return`** con un valor, como `return 5`, es un error de sintaxis dentro de una función generadora. El final de los resultados del generador también puede indicarse levantando manualmente **`StopIteration`**, o simplemente dejando que el flujo de ejecución caiga en el fondo de la función.

Puedes conseguir el efecto de los generadores manualmente escribiendo tu propia clase y almacenando todas las variables locales del generador como variables de instancia. Por ejemplo, la lución de una lista de enteros podría hacerse estableciendo `self.count` a 0, y haciendo que el método **`next()`** incremente `self.count` y lo retorne. Sin embargo, para un generador medianamente complicado, escribir la clase correspondiente sería mucho más complicado. `Lib/test/test_generators.py` contiene varios ejemplos más interesantes. El más sencillo implementa un recorrido en orden de un árbol utilizando generadores de forma recursiva

```
# Un generador recursivo que genera hojas de árboles en
def inorder(t):
if t:
for x in inorder(t.left):
yield x
yield t.label
for x in inorder(t.right):
yield x
```

Otros dos ejemplos en `Lib/test/test_generators.py` producen soluciones para el problema de las N reinas (colocar N reinas en un tablero de ajedrez NxN de forma que ninguna reina amenace a otra) y el recorrido del caballero (una ruta que lleva a un caballo a cada casilla de un tablero de ajedrez NxN sin visitar ninguna casilla dos veces).

La idea de los generadores proviene de otros lenguajes de programación, especialmente Icon (https://www2.cs.arizona.edu/icon/), donde la idea de los generadores es central. En Icon, cada expresión y llamada a función se comporta como un generador. Un ejemplo de «An Overview of the Icon Programming Language» en https://www2.cs.arizona.edu/icon/docs/ipd266.htm da una idea de cómo se ve esto:

```
sentence := "Store it in the neighboring harbor"
if (i := find("or", sentence)) > 5 then write(i)
```

En Icon, la función `find()` devuelve los índices en los que se encuentra la subcadena «o»: 3, 23, 33. En la declaración `if`, a `i` primero se le asigna un valor de 3, pero 3 es menor que 5, por lo que la comparación falla y Icon la vuelve a intentar con el segundo valor de 23. 23 es mayor que 5, por lo que la comparación ahora tiene éxito y el código imprime el valor 23 en la pantalla.

Python no va tan lejos como Icon en la adopción de generadores como concepto central. Los generadores se consideran una nueva parte del núcleo del lenguaje Python, pero aprenderlos o utilizarlos no es obligatorio; si no resuelven ningún problema que tengas, siéntete libre de ignorarlos. Una característica novedosa de la interfaz de Python en comparación con la de Icon es que el estado de un generador se representa como un objeto concreto (el iterador) que puede pasarse a otras funciones o almacenarse en una estructura de datos.

Ver también

PEP 255 [https://peps.python.org/pep-0255/] - Generadores simples

Escrito por Neil Schemenauer, Tim Peters, Magnus Lie Hetland. Implementado principalmente por Neil Schemenauer y Tim Peters, con otras correcciones del equipo de Python Labs.

PEP 237: Unificación de enteros largos y enteros

En versiones recientes, la distinción entre números enteros regulares, que son valores de 32 bits en la mayoría de las máquinas, y números enteros largos, que pueden ser de tamaño arbitrario, se estaba volviendo una molestia. Por ejemplo, en plataformas que admiten archivos de más de bytes que `2**32`, el método **`tell()`** de objetos de archivo tiene que devolver un número entero largo. Sin embargo, había varias partes de Python que esperaban números enteros simples y generarían un error si se proporcionaba un número entero largo en su lugar. Por ejemplo, en Python 1.5, solo se podían usar números enteros regulares como índice de segmento, y `'abc'[1L:]` generaría una excepción **`TypeError`** con el mensaje "el índice de segmento debe ser int".

Python 2.2 cambiará los valores de enteros cortos a enteros largos según sea necesario. El sufijo "L" ya no es necesario para indicar un literal entero largo, ya que ahora el compilador elegirá el tipo apropiado. (El uso del sufijo "L" se desaconsejará en futuras versiones 2.x de Python, provocando una advertencia en Python 2.4, y probablemente se eliminará en Python 3.0) Muchas operaciones que solían lanzar un **`OverflowError`** ahora retornarán un entero largo como resultado. Por ejemplo:

```
>>> 1234567890123
1234567890123L
>>> 2 ** 64
18446744073709551616L
```

En la mayoría de los casos, los enteros y los enteros largos se tratarán ahora de forma idéntica. Todavía se pueden distinguir con la función incorporada **`type()`**, pero rara vez se necesita.

Ver también

PEP 237 [https://peps.python.org/pep-0237/] - Unificación de enteros largos y enteros

Escrito por Moshe Zadka y Guido van Rossum. Implementado principalmente por Guido van Rossum.

PEP 238: Cambio del operador de división

El cambio más controvertido de Python 2.2 anuncia el inicio de un esfuerzo por arreglar un viejo defecto de diseño que ha estado en Python desde el principio. Actualmente, el operador de división de Python, `/`, se comporta como el operador de división de C cuando se le presentan dos argumentos enteros: retorna un resultado entero que se trunca cuando hay una parte fraccionaria. Por ejemplo, `3/2` es 1, no 1,5, y `(-1)/2` es -1, no -0,5. Esto significa que los resultados de la división pueden variar inesperadamente dependiendo del tipo de los dos operandos y, como Python está tipado dinámicamente, puede ser difícil determinar los posibles tipos de los operandos.

(La controversia se centra en si esto es *realmente* un defecto de diseño, y si vale la pena romper el código existente para arreglarlo. Ha provocado interminables discusiones en python-dev, y en julio de 2001 estalló una tormenta de publicaciones ácidamente sarcásticas en *comp.lang.python*. No argumentaré aquí a favor de ninguno de los dos bandos y me limitaré a describir lo que se ha implementado en la 2.2. Lea **PEP 238** [https://peps.python.org/pep-0238/] para un resumen de los argumentos y contra-argumentos)

Debido a que este cambio podría romper el código, se está introduciendo de forma muy gradual. Python 2.2 comienza la transición, pero el cambio no será completo hasta Python 3.0.

En primer lugar, tomaré prestada alguna terminología de **PEP 238** [https://peps.python.org/pep-0238/]. La «división verdadera» es la división con la que la mayoría de los no programadores están familiarizados: 3/2 es 1,5, 1/4 es 0,25, y así sucesivamente. La «división entera a la baja» es lo que hace actualmente el operador / de Python cuando se le dan operandos enteros; el resultado es el redondeo a la baja (*floor*) del valor retornado por la división verdadera. La «división clásica» es el comportamiento mixto actual de `/`; retorna el resultado de la división entera a la baja cuando los operandos son enteros, y retorna el resultado de la división verdadera cuando uno de los operandos es un número de punto flotante.

Estos son los cambios que introduce la versión 2.2:

- Un nuevo operador, `//`, es el operador de división entera a la baja. (Sí, ya sabemos que se parece al símbolo de comentario de C++.) `//` *siempre* realiza la división entera a la baja sin importar los tipos de sus operandos, así que `1 // 2` es 0 y `1.0 // 2.0` también es 0.0.

`//` está siempre disponible en Python 2.2; no es necesario habilitarlo mediante una sentencia `__future__`.

- Al incluir una declaración `from __future__ import division` en un módulo, el operador `/` se cambiará para retornar el resultado de la división verdadera, por lo que `1/2` es 0,5. Sin la declaración `__future__`, `/` sigue significando la división clásica. El significado por defecto de `/` no cambiará hasta Python 3.0.

- Las clases pueden definir métodos denominados **`__truediv__()`** y **`__floordiv__()`** para sobrecargar los dos operadores de división. En el nivel C, también hay espacios en la estructura **`PyNumberMethods`** para que los tipos de extensión puedan definir los dos operadores.

- Python 2.2 admite algunos argumentos de línea de comandos para comprobar si el código funcionará con la semántica de división modificada. Ejecutar python con **`-Q warn`** hará que se emita una advertencia cada vez que se aplique la división a dos enteros. Puedes usar esto para encontrar el código que está afectado por el cambio y arreglarlo. Por defecto, Python 2.2 simplemente realizará la división clásica sin una advertencia; la advertencia se activará por defecto en Python 2.3.

Ver también

PEP 238 [https://peps.python.org/pep-0238/] - Cambio del operador de división

Escrito por Moshe Zadka y Guido van Rossum. Implementado por Guido van Rossum..

Cambios en Unicode

El soporte de Unicode de Python se ha mejorado un poco en la versión 2.2. Las cadenas Unicode se almacenan normalmente como UCS-2, como enteros sin signo de 16 bits. Python 2.2 también puede ser compilado para usar UCS-4, enteros sin signo de 32 bits, como su codificación interna suministrando `--enable-unicode=ucs4` al script de configuración. (También es posible especificar `--disable-unicode` para desactivar completamente el soporte de Unicode)

Cuando se crea para utilizar UCS-4 (un «Python ancho»), el intérprete puede manejar de forma nativa caracteres Unicode desde U+000000 hasta U+110000, por lo que el rango de valores legales para la función `unichr()` se amplía en consecuencia. Si se utiliza un intérprete compilado para utilizar UCS-2 (un «Python estrecho»), los valores mayores que 65535 seguirán haciendo que `unichr()` genere una excepción `ValueError`. Todo esto se describe en **PEP 261** [https://peps.python.org/pep-0261/], «Compatibilidad con caracteres Unicode “anchos”»; consúltelo para obtener más detalles.

Otro cambio es más sencillo de explicar. Desde su introducción, las cadenas Unicode admiten un método `encode()` para convertir la cadena a una codificación seleccionada, como UTF-8 o Latin-1. En la versión 2.2 se ha añadido un método `decode([*encoding*])` simétrico a las cadenas de 8 bits (aunque no a las cadenas Unicode). `decode()` supone que la cadena está en la codificación especificada y la decodifica, devolviendo lo que devuelva el códec.

Gracias a esta nueva función, se han añadido códecs para tareas no relacionadas directamente con Unicode. Por ejemplo, se han añadido códecs para la codificación uu, la codificación base64 de MIME y la compresión con el módulo `zlib`:

```
>>> s = """Aquí hay un largo fragmento de texto redundar
... y repetitivo.
... """
>>> data = s.encode('zlib')
>>> data
'x\x9c\r\xc9\xc1\r\x80 \x10\x04\xc0?Ul...'
>>> data.decode('zlib')
'Aquí hay un largo fragmento de texto redundante, excesi
```

```
>>> print s.encode('uu')
begin 666 <datos>
M2&5R92!I<R!A(&QE;F=T:'D@<&EE8V4@;V8@<F5D=6YD86YT+"!O=F5
>=F5R8F]S92P*86YD(')E<&5T:71I=F4@=&5X="X*

fin
>>> "caramba".encode('rot-13')
'furrfu'
```

Para convertir una instancia de clase a Unicode, se puede definir un método **`__unicode__()`** mediante una clase, análogo a **`__str__()`**.

Marc-André Lemburg implementó **`encode()`**, **`decode()`** y **`__unicode__()`**. Fredrik Lundh y Martin von Löwis implementaron los cambios para admitir el uso interno de UCS-4.

Ver también

PEP 261 [https://peps.python.org/pep-0261/] - Soporte para caracteres Unicode "anchos"

Escrito por Paul Prescod.

PEP 227: Ámbitos anidados

En Python 2.1, los ámbitos anidados estáticamente se añadieron como una característica opcional, que se activaba mediante una directiva `from __future__ import nested_scopes`. En 2.2 los ámbitos anidados ya no necesitan ser habilitados especialmente, y ahora están siempre presentes. El resto de esta sección es una copia de la descripción de los ámbitos anidados de mi documento «What's New in Python 2.1»; si lo leíste cuando salió la 2.1, puedes saltarte el resto de esta sección.

El mayor cambio introducido en Python 2.1, y completado en 2.2, es el de las reglas de alcance de Python. En Python 2.0, en cualquier momento hay como máximo tres espacios de nombres utilizados para buscar nombres de variables: local, a nivel de módulo y el espacio de nombres incorporado. Esto a menudo sorprendía a la gente porque no coincidía con sus expectativas intuitivas. Por ejemplo, una definición de función recursiva anidada no funciona:

```
def f():
...
def g(valor):
...
return g(valor-1) + 1
...
```

La función `g()` siempre generará una excepción `NameError`, porque la vinculación del nombre `g` no está en su espacio de nombres local ni en el espacio de nombres a nivel de módulo. Esto no es un gran problema en la práctica (¿con qué frecuencia se definen recursivamente funciones internas como esta?), pero esto también hizo que el uso de la expresión `lambda` fuera más complicado, y esto fue un problema en la práctica. En el código que usa `lambda`, a menudo se pueden encontrar variables locales que se copian al pasarlas como valores predeterminados de los argumentos.

```
def find(self, name):
"Devuelve una lista de todas las entradas iguales a 'nam
L = filter(lambda x, name=name: x == name,
self.list_attribute)
return L
```

La legibilidad del código Python escrito en un estilo fuertemente funcional sufre mucho como resultado.

El cambio más significativo de Python 2.2 es que se ha añadido al lenguaje el ámbito estático para solucionar este problema. Como primer efecto, el argumento por defecto `name=name` es ahora innecesario en el ejemplo anterior. En pocas palabras, cuando a un nombre de variable dado no se le asigna un valor dentro de una función (mediante una asignación, o las sentencias `def`, `class`, o `import`), las referencias a la variable se buscarán en el espacio de nombres local del ámbito que la rodea. Puede encontrar una explicación más detallada de las reglas y una disección de la implementación en el PEP.

Este cambio puede causar algunos problemas de compatibilidad para el código en el que el mismo nombre de variable se utiliza tanto a nivel de módulo como de variable local dentro de una función que contiene otras definiciones de función. Sin embargo, esto parece bastante improbable, ya que dicho código habría sido bastante confuso de leer en primer lugar.

Un efecto secundario del cambio es que las sentencias `from module import *` y `exec` se han hecho ilegales dentro del ámbito de una función bajo ciertas condiciones. El manual de referencia de Python ha dicho todo el tiempo que `from module import *` sólo es legal en el nivel superior de un módulo, pero el intérprete de CPython nunca ha aplicado esto antes. Como parte de la implementación de los ámbitos anidados, el compilador que convierte el código fuente de Python en bytecodes tiene que generar un código diferente para acceder a las variables de un ámbito contenedor. Los códigos `from module import *` y `exec` hacen que el compilador no pueda averiguar esto, porque añaden nombres al espacio de nombres local que son desconocidos en tiempo de compilación. Por lo tanto, si una función contiene definiciones de funciones o expresiones **`lambda`** con variables libres, el compilador lo señalará lanzando una excepción **`SyntaxError`**.

Para que la explicación anterior quede un poco más clara, he aquí un ejemplo:

```
x = 1
def f():
# La siguiente línea es un error de sintaxis
exec 'x=2'
def g():
return x
```

La línea 4 que contiene la declaración `exec` es un error de sintaxis, ya que `exec` definiría una nueva variable local denominada `x` a cuyo valor debería acceder **`g()`**.

Esto no debería ser una gran limitación, ya que `exec` rara vez se utiliza en la mayoría del código de Python (y cuando se utiliza, a menudo es un signo de un mal diseño de todos modos).

Ver también

PEP 227 [https://peps.python.org/pep-0227/] - Ámbitos anidados estáticamente
Escrito e implementado por Jeremy Hylton.

Módulos nuevos y mejorados

- El módulo **`xmlrpclib`** fue aportado a la biblioteca estándar por Fredrik Lundh, que proporciona soporte para escribir clientes XML-RPC. XML-RPC es un protocolo simple de llamada a procedimiento remoto creado sobre HTTP y XML. Por ejemplo, el siguiente fragmento recupera una lista de canales RSS de la red O’Reilly y, a continuación, enumera los titulares recientes de un canal:

  ```
  import xmlrpclib
  s = xmlrpclib.Server(
  'http://www.oreillynet.com/meerkat/xml-rpc/server.p
  channels = s.meerkat.getChannels()
  # channels es una lista de diccionarios, como esta:
  # [{'id': 4, 'title': 'Freshmeat Daily News'}
  # {'id': 190, 'title': '32Bits Online'},
  # {'id': 4549, 'title': '3DGamers'}, ... ]

  # Obtener los elementos de un canal
  items = s.meerkat.getItems( {'channel': 4} )

  # 'items' es otra lista de diccionarios, como esta:
  # [{'link': 'http://freshmeat.net/releases/52719/',
  # 'description': 'Una utilidad que convierte HTML a
  # 'title': 'html2fo 0.3 (Predeterminado)'}, ... ]
  ```

  El módulo **`SimpleXMLRPCServer`** facilita la creación de servidores XML-RPC sencillos. Consulte http://xmlrpc.scripting.com/ para obtener más información sobre XML-RPC.

- El nuevo módulo **`hmac`** implementa el algoritmo HMAC descrito por RFC 2104 [https://datatracker.ietf.org/doc/html/rfc2104.html]. (Contribución de Gerhard Häring)

- Varias funciones que originalmente devolvían tuplas extensas ahora devuelven pseudosecuencias que aún se comportan como tuplas pero que también tienen atributos mnemotécnicos como **`memberst_mtime`** o `tm_year`. Las funciones mejoradas incluyen `stat()`, `fstat()`, `statvfs()` y `fstatvfs()` en el módulo `os`, y `localtime()`, `gmtime()` y `strptime()` en el módulo `time`.

  Por ejemplo, para obtener el tamaño de un archivo utilizando las antiguas tuplas, se terminaba escribiendo algo como `tamaño_de_archivo = os.stat(nombre_de_archivo)[stat.ST_SIZE]`, pero ahora se puede escribir más claramente como `tamaño_de_archivo = os.stat(nombre_de_archivo).st_size`.

  El parche original para esta función fue aportado por Nick Mathewson.

- El perfilador de Python ha sido ampliamente revisado y se han corregido varios errores en su salida. (Contribución de Fred L. Drake, Jr. y Tim Peters)

- El módulo `socket` puede ser compilado para soportar IPv6; especifica la opción **`--enable-ipv6`** al script configure de Python. (Contribución de Jun-ichiro «itojun» Hagino)

- Se agregaron dos nuevos caracteres de formato al módulo `struct` para enteros de 64 bits en plataformas que admiten el tipo C long long. `q` es para un entero de 64 bits con signo y `Q` es para uno sin firmar. El valor se retorna en el tipo de entero largo de Python. (Aportado por Tim Peters.)

- En el modo interactivo del intérprete, hay una nueva función incorporada `help()` que utiliza el módulo `pydoc` introducido en Python 2.1 para proporcionar ayuda interactiva. **`help()`** sin ningún argumento te sitúa en una utilidad de ayuda online, donde puedes introducir los nombres de las funciones, clases o módulos para leer su texto de ayuda. (Contribuido por Guido van Rossum, usando el módulo `pydoc` de Ka-Ping Yee)

- Se han realizado varias correcciones de errores y mejoras de rendimiento en el motor SRE subyacente al módulo `re`. Por ejemplo, las funciones `re.sub()` y `re.split()` se han reescrito en C. Otro parche aportado acelera ciertos rangos de caracteres Unicode por un factor de dos, y un nuevo método `finditer()` que devuelve un iterador sobre todas las coincidencias no superpuestas en una cadena dada. (SRE es mantenido por Fredrik Lundh. El parche BIGCHARSET fue aportado por Martin von Löwis).

- El módulo `smtplib` soporta ahora RFC 2487 [https://datatracker.ietf.org/doc/html/rfc2487.html], «Secure SMTP over TLS», por lo que ahora es posible cifrar el tráfico SMTP entre un programa Python y el agente de transporte de correo que recibe un mensaje. `smtplib` también soporta la autenticación SMTP. (Contribución de Gerhard Häring)

- El módulo `imaplib`, mantenido por Piers Lauder, tiene soporte para varias extensiones nuevas: la extensión NAMESPACE definida en RFC 2342 [https://datatracker.ietf.org/doc/html/rfc2342.html], SORT, GETACL y SETACL. (Contribución de Anthony Baxter y Michel Pelletier)

- El análisis de direcciones de correo electrónico del módulo `rfc822` ahora es compatible con RFC 2822 [https://datatracker.ietf.org/doc/html/rfc2822.html], una actualización de RFC 822 [https://datatracker.ietf.org/doc/html/rfc822.html]. (El nombre del módulo, *not*, se cambiará a `rfc2822`). También se ha añadido un nuevo paquete, `email`, para analizar y generar mensajes de correo electrónico. (Contribuido por Barry Warsaw y derivado de su trabajo en Mailman).

- El módulo `difflib` ahora contiene una nueva clase `Differ` para producir listas legibles por humanos de cambios (un «delta») entre dos secuencias de líneas de texto. También hay dos funciones generadoras, `ndiff()` y `restore()`, que devuelven respectivamente un delta de dos secuencias, o una de las secuencias originales de un delta. (Trabajo básico aportado por David Goodger, a partir del código ndiff.py de Tim Peters, quien luego realizó la generación).

- Se agregaron las nuevas constantes `ascii_letters`, `ascii_lowercase` y `ascii_uppercase` al módulo `string`. Había varios módulos en la biblioteca estándar que usaban `string.letters` para referirse a los rangos A-Za-z, pero esa suposición es incorrecta cuando se usan las configuraciones regionales, porque `string.letters` varía según el conjunto de caracteres legales definidos por la configuración regional actual. Todos los módulos con errores se han corregido para que usen `ascii_letters` en su lugar. (Informado por una persona desconocida; corregido por Fred L. Drake, Jr.)
- El módulo `mimetypes` ahora facilita el uso de bases de datos de tipo MIME alternativas mediante la incorporación de una clase `MimeTypes`, que toma una lista de nombres de archivos para analizar. (Contribuido por Fred L. Drake, Jr.)
- Se agregó una clase `Timer` al módulo `threading` que permite programar una actividad para que se realice en un momento futuro. (Contribución de Itamar Shtull-Trauring).

Cambios y correcciones en el intérprete

Algunos de los cambios sólo afectan a la gente que trata con el intérprete de Python a nivel de C porque están escribiendo módulos de extensión de Python, incrustando el intérprete, o simplemente hackeando el propio intérprete. Si sólo escribes código Python, ninguno de los cambios descritos aquí te afectará mucho.

- Las funciones de perfilado y rastreo pueden implementarse ahora en C, que puede operar a velocidades mucho mayores que las funciones basadas en Python y debería reducir la sobrecarga de perfilado y rastreo. Esto será de interés para los autores de entornos de desarrollo para Python. Se han añadido dos nuevas funciones en C a la API de Python, `PyEval_SetProfile()` y `PyEval_SetTrace()`. Las funciones `sys.setprofile()` y `sys.settrace()` existentes siguen existiendo, y simplemente se han cambiado para utilizar la nueva interfaz de nivel C. (Contribución de Fred L. Drake, Jr.)

- Se agregó otra API de bajo nivel, principalmente de interés para los implementadores de depuradores y herramientas de desarrollo de Python. `PyInterpreterState_Head()` y `PyInterpreterState_Next()` permiten que un llamador recorra todos los objetos de intérprete existentes; `PyInterpreterState_ThreadHead()` y `PyThreadState_Next()` permiten recorrer todos los estados de subprocesos para un intérprete determinado. (Contribución de David Beazley).

- La interfaz a nivel de C para el recolector de basura ha sido cambiada para facilitar la escritura de tipos de extensión que soporten la recolección de basura y para depurar los malos usos de las funciones. Varias funciones tienen una semántica ligeramente diferente, por lo que hubo que cambiar el nombre de un montón de funciones. Las extensiones que utilizan la antigua API seguirán compilando pero *no* participarán en la recolección de basura, por lo que actualizarlas para la 2.2 debería considerarse de alta prioridad.

  Para actualizar un módulo de extensión a la nueva API, realice los siguientes pasos:

- Cambia el nombre de `Py_TPFLAGS_GC()` a `PyTPFLAGS_HAVE_GC()`.

- Utilice `PyObject_GC_New()` o `PyObject_GC_NewVar()` para asignar
  objetos, y `PyObject_GC_Del()` para desocuparlos.

- Cambie el nombre de `PyObject_GC_Init()` a `PyObject_GC_Track()` y de `PyObject_GC_Fini()` a `PyObject_GC_UnTrack()`.

- Eliminar `PyGC_HEAD_SIZE` de los cálculos de tamaño de objeto.

- Eliminar llamadas a **`PyObject_AS_GC()`** y **`PyObject_FROM_GC()`**.
- Se ha añadido una nueva secuencia de formato `et` a **`PyArg_ParseTuple()`**; `et` toma tanto un parámetro como un nombre de codificación, y convierte el parámetro a la codificación dada si el parámetro resulta ser una cadena Unicode, o lo deja solo si es una cadena de 8 bits, asumiendo que ya está en la codificación deseada. Esto difiere del carácter de formato `es`, que asume que las cadenas de 8 bits están en la codificación ASCII por defecto de Python y las convierte a la nueva codificación especificada. (Contribuido por M.-A. Lemburg, y utilizado para el soporte de MBCS en Windows descrito en la siguiente sección)
- Se ha agregado una función de análisis de argumentos diferente, **`PyArg_UnpackTuple()`**, que es más simple y presumiblemente más rápida. En lugar de especificar una cadena de formato, la persona que llama simplemente proporciona el número mínimo y máximo de argumentos esperados y un conjunto de punteros a las variables PyObject* que se completarán con los valores de los argumentos.
- Hay dos nuevos indicadores **`METH_NOARGS`** y **`METH_O`** disponibles en las tablas de definición de métodos para simplificar la implementación de métodos sin argumentos o con un único argumento sin tipo. Llamar a estos métodos es más eficiente que llamar a un método correspondiente que utilice **`METH_VARARGS`**. Además, el antiguo estilo **`METH_OLDARGS`** de escribir métodos en C ahora está oficialmente obsoleto.
- Se agregaron dos nuevas funciones de contenedor, **`PyOS_snprintf()`** y **`PyOS_vsnprintf()`**, para proporcionar implementaciones multiplataforma para las relativamente nuevas API de bibliotecas C **`snprintf()`** y **`vsnprintf()`**. A diferencia de las funciones estándar **`sprintf()`** y **`vsprintf()`**, las versiones de Python verifican los límites del búfer utilizado para protegerse contra

desbordamientos del búfer. (Contribuido por M.-A. Lemburg).

- La función `_PyTuple_Resize()` ha perdido un parámetro que no se utilizaba, por lo que ahora toma 2 parámetros en lugar de 3. El tercer argumento nunca se utilizaba, y puede descartarse simplemente al portar el código de versiones anteriores a Python 2.2.

Otros cambios y correcciones

Como es habitual, hubo un montón de otras mejoras y correcciones de errores repartidas por todo el árbol de fuentes. Una búsqueda en los registros de cambios de CVS revela que se aplicaron 527 parches y se corrigieron 683 errores entre Python 2.1 y 2.2; en 2.2.1 se aplicaron 139 parches y se corrigieron 143 errores; en 2.2.2 se aplicaron 106 parches y se corrigieron 82 errores. Es probable que estas cifras estén subestimadas.

Algunos de los cambios más notables son:

- El código del puerto MacOS para Python, mantenido por Jack Jansen, se mantiene ahora en el árbol CVS principal de Python, y se han realizado muchos cambios para soportar MacOS X.

  El cambio más significativo es la capacidad de construir Python como un marco de trabajo, que se activa proporcionando la opción **`--enable-framework`** al script de configuración cuando se compila Python. Según Jack Jansen, «Esto instala una instalación autónoma de Python más el «pegamento» del framework de OS X en `/Library/Frameworks/Python.framework` (o en otra ubicación de su elección). Por ahora hay poco beneficio inmediato añadido a esto (en realidad, existe la desventaja de que tienes que cambiar tu PATH para poder encontrar Python), pero es la base para crear una aplicación Python completa, portar el IDE de MacPython, posiblemente usar Python como un lenguaje de scripting estándar de OSA y mucho más.»

  La mayoría de los módulos de la caja de herramientas de

MacPython, que interactúan con las APIs de MacOS como ventanas, QuickTime, scripts, etc. han sido portados a OS X, pero se han dejado comentados en `setup.py`. Las personas que quieran experimentar con estos módulos pueden descomentarlos manualmente.

- Los argumentos de palabras clave pasados a funciones incorporadas que no los aceptan ahora provocan una excepción **`TypeError`**, con el mensaje «*function* no acepta argumentos de palabras clave».

- Las referencias débiles, añadidas en Python 2.1 como un módulo de extensión, son ahora parte del núcleo porque se utilizan en la implementación de clases de nuevo estilo. Por lo tanto, la excepción **`ReferenceError`** se ha movido del módulo **`weakref`** para convertirse en una excepción incorporada.

- Un nuevo script, `Tools/scripts/cleanfuture.py` de Tim Peters, elimina automáticamente las sentencias `__future__` obsoletas del código fuente de Python.

- Se ha añadido un argumento adicional *flags* a la función incorporada **`compile()`**, por lo que el comportamiento de las sentencias `__future__` puede ahora observarse correctamente en shells simulados, como los presentados por IDLE y otros entornos de desarrollo. Esto se describe en **PEP 264** [https://peps.python.org/pep-0264/]. (Contribución de Michael Hudson)

- La nueva licencia introducida con Python 1.6 no era compatible con la GPL. Esto se ha solucionado con algunos cambios textuales menores en la licencia 2.2, de modo que ahora es legal volver a incrustar Python dentro de un programa con licencia GPL. Tenga en cuenta que Python en sí mismo no es GPL, sino que está bajo una licencia que es esencialmente equivalente a la licencia BSD, igual que siempre. Los cambios en la licencia también se aplicaron a las versiones 2.0.1 y 2.1.1 de Python.

- Cuando se presenta un nombre de archivo Unicode en

Windows, Python ahora lo convertirá en una cadena codificada en MBCS, como la que utilizan las APIs de archivos de Microsoft. Como las APIs de archivos utilizan explícitamente MBCS, la elección de Python de ASCII como codificación por defecto resulta ser una molestia. En Unix, se utiliza el juego de caracteres de la localización si `locale.nl_langinfo(CODESET)` está disponible. (El soporte de Windows fue contribuido por Mark Hammond con la ayuda de Marc-André Lemburg. El soporte para Unix fue añadido por Martin von Löwis)

- La compatibilidad con archivos de gran tamaño ya está activada en Windows. (Contribución de Tim Peters.)

- El script `Tools/scripts/ftpmirror.py` ahora analiza un archivo `.netrc`, si tiene uno. (Contribución de Mike Romberg)

- Algunas características del objeto devuelto por la función **`xrange()`** ahora están obsoletas y activan advertencias cuando se accede a ellas; desaparecerán en Python 2.3. Los objetos **`xrange`** intentaron simular que eran tipos de secuencia completa al admitir la segmentación, la multiplicación de secuencias y el operador **`in`**, pero estas características se usaban rara vez y, por lo tanto, presentaban errores. El método **`tolist()`** y los atributos **`start`**, **`stop`** y **`step`** también están en desuso. En el nivel C, el cuarto argumento de la función **`PyRange_New()`**, `repeat`, también ha quedado en desuso.

- Hubo un montón de parches para la implementación del diccionario, sobre todo para arreglar posibles vertidos del núcleo si un diccionario contiene objetos que cambian furtivamente su valor hash, o mutan el diccionario que contienen. Durante un tiempo python-dev cayó en un suave ritmo de Michael Hudson encontrando un caso que volcaba el núcleo, Tim Peters corrigiendo el error, Michael encontrando otro caso, y así sucesivamente.

- En Windows, Python puede ahora compilarse con Borland C gracias a una serie de parches aportados por Stephen Hansen, aunque el resultado aún no es totalmente funcional. (Pero esto *es* un progreso…)

- Otra mejora de Windows: Wise Solutions ofreció generosamente a PythonLabs el uso de su sistema InstallerMaster 8.1. Los anteriores instaladores de PythonLabs para Windows utilizaban Wise 5.0a, que estaba empezando a mostrar su edad. (Empaquetado por Tim Peters)
- Los archivos que terminan en `.pyw` pueden importarse ahora en Windows. `.pyw` es algo exclusivo de Windows, que se utiliza para indicar que un script debe ejecutarse utilizando PYTHONW.EXE en lugar de PYTHON.EXE para evitar que aparezca una consola DOS para mostrar la salida. Este parche hace posible la importación de tales scripts, en caso de que también se puedan utilizar como módulos. (Implementado por David Bolen)
- En las plataformas en las que Python utiliza la función C **`dlopen()`** para cargar módulos de extensión, ahora es posible establecer las banderas utilizadas por **`dlopen()`** utilizando las funciones **`sys.getdlopenflags()`** y **`sys.setdlopenflags()`**. (Contribución de Bram Stolk.)
- La función incorporada **`pow()`** ya no admite 3 argumentos cuando se suministran números de punto flotante. `pow(x, y, z)` devuelve `(x**y) % z`, pero esto nunca es útil para números de punto flotante y el resultado final varía de manera impredecible según la plataforma. Una llamada como `pow(2.0, 8.0, 7.0)` ahora generará una excepción **`TypeError`**.

Agradecimientos

El autor desea agradecer a las siguientes personas sus sugerencias, correcciones y ayuda en varios borradores de este artículo: Fred Bremmer, Keith Briggs, Andrew Dalke, Fred L. Drake, Jr, Carel Fellinger, David Goodger, Mark Hammond, Stephen Hansen, Michael Hudson, Jack Jansen, Marc-André Lemburg, Martin von Löwis, Fredrik Lundh, Michael McLay, Nick Mathewson, Paul

Moore, Gustavo Niemeyer, Don O’Donnell, Joonas Paalasma, Tim Peters, Jens Quade, Tom Reinhardt, Neil Schemenauer, Guido van Rossum, Greg Ward, Edward Welbourne.

Qué hay de nuevo en Python 2.1

Autor:

A.M. Kuchling

Introducción

Este artículo explica las nuevas características de Python 2.1. Aunque no hay tantos cambios en 2.1 como en Python 2.0, todavía hay algunas sorpresas agradables. La versión 2.1 es la primera que se dirige mediante el uso de Propuestas de Mejora de Python, o PEPs, por lo que la mayoría de los cambios importantes tienen PEPs adjuntos que proporcionan una documentación más completa y una justificación de diseño para el cambio. Este artículo no intenta documentar las nuevas características por completo, sino que simplemente proporciona una visión general de las nuevas características para los programadores de Python. Consulta la documentación de Python 2.1, o el PEP específico, para obtener más detalles sobre cualquier nueva característica que te interese particularmente.

Un objetivo reciente del equipo de desarrollo de Python ha sido acelerar el ritmo de las nuevas versiones, con una nueva versión cada 6 a 9 meses. La versión 2.1 es la primera que sale a este ritmo más rápido, con la primera alfa que apareció en enero, 3 meses después de que se publicara la versión final de la 2.0.

La versión final de Python 2.1 se realizó el 17 de abril de 2001.

PEP 227: Ámbitos anidados

El mayor cambio en Python 2.1 es el de las reglas de alcance de Python. En Python 2.0, en cualquier momento hay como máximo tres espacios de nombres utilizados para buscar nombres de variables: local, a nivel de módulo y el espacio de nombres incorporado. Esto a menudo sorprendía a la gente porque no coincidía con sus expectativas intuitivas. Por ejemplo, una definición de función recursiva anidada no funciona:

```
def f():
...
def g(valor):
...
return g(valor-1) + 1
...
```

La función **`g()`** siempre generará una excepción **`NameError`**, porque la vinculación del nombre `g` no está en su espacio de nombres local ni en el espacio de nombres a nivel de módulo. Esto no es un gran problema en la práctica (¿con qué frecuencia se definen recursivamente funciones internas como esta?), pero esto también hizo que el uso de la expresión **`lambda`** fuera más complicado, y esto fue un problema en la práctica. En el código que usa **`lambda`**, a menudo se pueden encontrar variables locales que se copian al pasarlas como valores predeterminados de los argumentos.

```
def find(self, name):
"Devuelve una lista de todas las entradas iguales a 'nam
L = filter(lambda x, name=name: x == name,
self.list_attribute)
return L
```

La legibilidad del código Python escrito en un estilo fuertemente funcional sufre mucho como resultado.

El cambio más significativo de Python 2.1 es que se ha añadido al lenguaje el ámbito estático para solucionar este problema. Como primer efecto, el argumento por defecto `name=name` es ahora innecesario en el ejemplo anterior. En pocas palabras, cuando a un nombre de variable dado no se le asigna un valor dentro de una función (mediante una asignación, o las sentencias **`def`**, **`class`**, o **`import`**), las referencias a la variable se buscarán en el espacio de nombres local del ámbito que la rodea. Puede encontrar una explicación más detallada de las reglas y una disección de la implementación en el PEP.

Este cambio puede causar algunos problemas de compatibilidad para el código en el que el mismo nombre de variable se utiliza tanto a nivel de módulo como de variable local dentro de una función que contiene otras definiciones de función. Sin embargo, esto parece bastante improbable, ya que dicho código habría sido bastante confuso de leer en primer lugar.

Un efecto secundario del cambio es que las sentencias `from module import *` y `exec` se han hecho ilegales dentro del ámbito de una función bajo ciertas condiciones. El manual de referencia de Python ha dicho todo el tiempo que `from module import *` sólo es legal en el nivel superior de un módulo, pero el intérprete de CPython nunca ha aplicado esto antes. Como parte de la implementación de los ámbitos anidados, el compilador que convierte el código fuente de Python en bytecodes tiene que generar un código diferente para acceder a las variables de un ámbito contenedor. Los códigos `from module import *` y `exec` hacen que el compilador no pueda averiguar esto, porque añaden nombres al espacio de nombres local que son desconocidos en tiempo de compilación. Por lo tanto, si una función contiene definiciones de funciones o expresiones **`lambda`** con variables libres, el compilador lo señalará lanzando una excepción **`SyntaxError`**.

Para que la explicación anterior quede un poco más clara, he aquí un ejemplo:

```
x = 1
def f():
# La siguiente línea es un error de sintaxis
exec 'x=2'
def g():
return x
```

La línea 4 que contiene la sentencia `exec` es un error de sintaxis, ya que `exec` definiría una nueva variable local llamada `x` cuyo valor debería ser accedido por **`g()`**.

Esto no debería ser una gran limitación, ya que `exec` rara vez se utiliza en la mayoría del código de Python (y cuando se utiliza, a menudo es un signo de un mal diseño de todos modos).

Los problemas de compatibilidad han llevado a que los ámbitos anidados se introduzcan gradualmente; en Python 2.1, no están habilitados por defecto, pero pueden activarse dentro de un módulo utilizando una sentencia future como se describe en **PEP 236** [https://peps.python.org/pep-0236/]. (En Python 2.2, los ámbitos anidados se convertirán en el valor por defecto y no habrá forma de desactivarlos, pero los usuarios habrán tenido toda la vida de la versión 2.1 para arreglar cualquier rotura resultante de su introducción.

Ver también

PEP 227 [https://peps.python.org/pep-0227/] - Ámbitos anidados estáticamente

Escrito e implementado por Jeremy Hylton.

PEP 236: Directivas _future_

La reacción a los ámbitos anidados fue una preocupación generalizada sobre los peligros de romper el código con la versión 2.1, y fue lo suficientemente fuerte como para que los Pythonistas adoptaran un enfoque más conservador. Este enfoque consiste en introducir una convención para habilitar una funcionalidad opcional en la versión N que se convertirá en obligatoria en la versión N+1.

La sintaxis utiliza una sentencia `from...import` utilizando el nombre de módulo reservado **`__future__`**. Los ámbitos anidados pueden habilitarse mediante la siguiente sentencia:

```
desde __future__ importar ámbitos anidados
```

Aunque parece una sentencia **`import`** normal, no lo es; hay reglas estrictas sobre dónde se puede poner una sentencia future. Sólo pueden estar en la parte superior de un módulo, y deben preceder a cualquier código Python o a las sentencias `import` normales. Esto se debe a que tales declaraciones pueden afectar a la forma en que el compilador de código de bytes de Python analiza el código y genera el código de bytes, por lo que deben preceder a cualquier declaración que dé lugar a la producción de códigos de bytes.

Ver también

PEP 236 [https://peps.python.org/pep-0236/] - De vuelta al `__future__`

Escrito por Tim Peters, y ejecutado principalmente por Jeremy Hylton.

PEP 207: Comparaciones Enriquecidas

En versiones anteriores, el soporte de Python para implementar comparaciones en clases definidas por el usuario y tipos de extensión era bastante simple. Las clases podían implementar un método `__cmp__()` al que se le daban dos instancias de una clase, y solo podía devolver 0 si eran iguales o +1 o -1 si no lo eran; el método no podía generar una excepción o devolver nada más que un valor booleano. Los usuarios de Python numérico a menudo encontraban que este modelo era demasiado débil y restrictivo, porque en los programas de procesamiento numérico para los que se usa Python numérico, sería más útil poder realizar comparaciones elemento por elemento de dos matrices, devolviendo una matriz que contenga los resultados de una comparación dada para cada elemento. Si las dos matrices son de diferentes tamaños, entonces la comparación tiene que poder generar una excepción para señalar el error.

En Python 2.1, se añadieron comparaciones enriquecidas para dar soporte a esta necesidad. Las clases de Python pueden ahora sobrecargar individualmente cada una de las operaciones <, <=, >, >=, == y !=. Los nuevos nombres de métodos mágicos son:

Operación	Nombre del método
<	`__lt__()`
<=	`__le__()`

>`__gt__()`
>=`__ge__()`
==`__eq__()`
!=`__ne__()`

(Los métodos mágicos se denominan como los correspondientes operadores de Fortran `.LT.`. `.LE.`, &c. Los programadores numéricos están casi seguramente bastante familiarizados con estos nombres y los encontrarán fáciles de recordar.)

Cada uno de estos métodos mágicos tiene la forma `method(self, other)`, donde `self` será el objeto que se encuentre en el lado izquierdo del operador, mientras que `other` será el objeto que se encuentre en el lado derecho. Por ejemplo, la expresión `A < B` hará que se llame a `A.__lt__(B)`.

Cada uno de estos métodos mágicos puede retornar cualquier cosa: un booleano, una matriz, una lista o cualquier otro objeto de Python. También pueden lanzar una excepción si la comparación es imposible, inconsistente o no tiene sentido.

La función `cmp(A,B)` incorporada puede utilizar la rica maquinaria de comparación y ahora acepta un argumento opcional que especifica qué operación de comparación utilizar; esto se proporciona como una de las cadenas `"<"`, `"<="`, `">"`, `">="`, `"=="` o `"!="`. Si se llama sin el tercer argumento opcional, **`cmp()`** solo devolverá -1, 0 o +1 como en versiones anteriores de Python; de lo contrario, llamará al método apropiado y puede devolver cualquier objeto de Python.

También hay cambios correspondientes de interés para los programadores de C; hay una nueva ranura `tp_richcmp` en los objetos de tipo y una API para realizar una comparación rica determinada. No cubrirá la API de C aquí, sino que le remitiré a PEP 207 [https://peps.python.org/pep-0207/], o a la documentación de la API de C de 2.1, para la lista completa de funciones relacionadas.

Ver también

PEP 207 [https://peps.python.org/pep-0207/] - Comparaciones enriquecidas

Escrito por Guido van Rossum, basado en gran medida en un trabajo anterior de David Ascher, e implementado por Guido van Rossum.

PEP 230: Marco de advertencia

A lo largo de sus 10 años de existencia, Python ha acumulado un cierto número de módulos y características obsoletas en el camino. Es difícil saber cuándo es seguro eliminar una característica, ya que no hay manera de saber cuánto código la utiliza — tal vez ningún programa depende de la característica, o tal vez muchos lo hacen. Para permitir la eliminación de características antiguas de una manera más estructurada, se añadió un marco de advertencia. Cuando los desarrolladores de Python quieran deshacerse de una característica, primero se activará una advertencia en la siguiente versión de Python. La siguiente versión de Python puede entonces eliminar la característica, y los usuarios habrán tenido un ciclo de lanzamiento completo para eliminar los usos de la antigua característica.

Python 2.1 añade el marco de trabajo de las advertencias para ser utilizado en este esquema. Añade un módulo `warnings` que proporciona funciones para emitir advertencias, y para filtrar las advertencias que no se quieren mostrar. Los módulos de terceros también pueden utilizar este marco de trabajo para dejar de lado funciones antiguas que ya no desean soportar.

Por ejemplo, en Python 2.1 el módulo `regex` está obsoleto, por lo que al importarlo se imprime una advertencia:

```
>>> import regex
__main__:1: DeprecationWarning: el módulo regex
está obsoleto; utilice el módulo re
>>>
```

Las advertencias se pueden emitir llamando a la función `warnings.warn()`:

```
advertencias.warn("la función X ya no es compatible")
```

El primer parámetro es el mensaje de advertencia; se pueden utilizar otros parámetros opcionales para especificar una categoría de advertencia concreta.

Se pueden agregar filtros para desactivar ciertas advertencias; se puede aplicar un patrón de expresión regular al mensaje o al nombre del módulo para suprimir una advertencia. Por ejemplo, puede tener un programa que utiliza el módulo **`regex`** y no desea perder tiempo en convertirlo para que utilice el módulo **`re`** en este momento. La advertencia se puede suprimir llamando a

```
advertencias de importación
warnings.filterwarnings(action = 'ignore',
message='.*regex module is obsolete',
category=DeprecationWarning,
module = '__main__')
```

Esto agrega un filtro que se aplicará solo a las advertencias de la clase **`DeprecationWarning`** activadas en el módulo **`__main__`** y aplica una expresión regular para que coincida solo con el mensaje sobre que el módulo **`regex`** está obsoleto y hará que se ignoren dichas advertencias. Las advertencias también se pueden imprimir solo una vez, imprimirse cada vez que se ejecuta el código infractor o convertirse en excepciones que harán que el programa se detenga (a menos que las excepciones se detecten de la manera habitual, por supuesto).

También se agregaron funciones a la API de C de Python para emitir advertencias; consulte el PEP 230 o la documentación de la API de Python para conocer los detalles.

Ver también

PEP 5 [https://peps.python.org/pep-0005/] - Directrices para la evolución del lenguaje

Escrito por Paul Prescod, para especificar los procedimientos a seguir cuando se eliminan características antiguas de PythonLa política descrita en este PEP no ha sido adoptada oficialmente, pero la política final probablemente no será muy diferente de la propuesta de

Prescod.

PEP 230 [https://peps.python.org/pep-0230/] - Marco de advertencia
Escrito y ejecutado por Guido van Rossum.

PEP 229: Sistema de construcción nuevo

Al compilar Python, el usuario tenía que entrar y editar el archivo `Modules/Setup` para habilitar varios módulos adicionales; el conjunto por defecto es relativamente pequeño y se limita a los módulos que se compilan en la mayoría de las plataformas Unix. Esto significa que en plataformas Unix con muchas más características, sobre todo Linux, las instalaciones de Python no suelen contener todos los módulos útiles que podrían.

Python 2.0 añadió los Distutils, un conjunto de módulos para distribuir e instalar extensiones. En Python 2.1, los Distutils se utilizan para compilar gran parte de la biblioteca estándar de módulos de extensión, autodetectando cuáles son compatibles con la máquina actual Se espera que esto haga que las instalaciones de Python sean más fáciles y tengan más funciones.

En lugar de tener que editar el archivo `Modules/Setup` para habilitar los módulos, un script `setup.py` en el directorio superior de la distribución de fuentes de Python se ejecuta en el momento de la compilación, e intenta descubrir qué módulos pueden ser habilitados examinando los módulos y archivos de cabecera en el sistema. Si un módulo está configurado en `Modules/Setup`, el script `setup.py` no intentará compilar ese módulo y se remitirá al contenido del archivo `Modules/Setup`. Esto proporciona una manera de especificar cualquier flag de línea de comandos extraña o bibliotecas que se requieren para una plataforma específica.

En otro cambio de gran alcance en el mecanismo de construcción, Neil Schemenauer reestructuró las cosas para que Python ahora utilice un único makefile que no es recursivo, en lugar de makefiles en el directorio superior y en cada uno de los subdirectorios `Python/`, `Parser/`, `Objects/`, y `Modules/`. Esto hace que la construcción de Python sea más rápida y también hace que el hackeo de los Makefiles sea más claro y sencillo.

Ver también

PEP 229 [https://peps.python.org/pep-0229/] - Uso de Distutils para construir Python
Escrito y ejecutado por A.M. Kuchling.

PEP 205: Referencias débiles

Las referencias débiles, disponibles a través del módulo **`weakref`**, son un nuevo tipo de datos menor pero útil en la caja de herramientas del programador de Python.

Almacenar una referencia a un objeto (por ejemplo, en un diccionario o una lista) tiene el efecto secundario de mantener ese objeto vivo para siempre. Hay algunos casos específicos en los que este comportamiento es indeseable, siendo las cachés de objetos el más común, y otro son las referencias circulares en estructuras de datos como los árboles.

Por ejemplo, considere una función de memoización que almacena en caché los resultados de otra función `f(x)` almacenando el argumento de la función y su resultado en un diccionario:

```
_cache = {}
def memoize(x):
if _cache.has_key(x):
return _cache[x]

retval = f(x)

# Almacenar en caché el objeto devuelto
_cache[x] = retval

return retval
```

Esta versión funciona para cosas simples como números enteros, pero tiene un efecto secundario: el diccionario `_cache` contiene una referencia a los valores de retorno, por lo que nunca se desasignarán hasta que el proceso de Python salga y limpie. Esto no es muy notorio para los números enteros, pero si **`f()`** devuelve un objeto o una estructura de datos que ocupa mucha memoria, esto puede ser un problema.

Las referencias débiles proporcionan una forma de implementar una caché que no mantendrá los objetos vivos más allá de su tiempo. Si un objeto sólo es accesible a través de referencias débiles, el objeto será desasignado y las referencias débiles indicarán ahora que el objeto al que se refería ya no existe. Una referencia débil a un objeto *obj* se crea llamando `wr = weakref.ref(obj)`. El objeto al que se hace referencia se retorna llamando a la referencia débil como si fuera una función: `wr()` retornará el objeto referenciado, o `None` si el objeto ya no existe.

Esto permite escribir una función **`memoize()`** cuyo caché no mantiene vivos los objetos, almacenando referencias débiles en el caché.

```
_cache = {}
def memoize(x):
if _cache.has_key(x):
obj = _cache[x]()
# Si el objeto de referencia débil aún existe,
# lo devuelve
if obj no es None: return obj

retval = f(x)

# Almacena en caché una referencia débil
_cache[x] = weakref.ref(retval)

return retval
```

El módulo **`weakref`** también permite crear objetos proxy que se comportan como referencias débiles (un objeto al que sólo hacen referencia objetos proxy se desasigna), pero en lugar de requerir una llamada explícita para recuperar el objeto, el proxy reenvía de forma transparente todas las operaciones al objeto mientras éste siga existiendo. Si se desasigna el objeto, intentar usar un proxy provocará que se genere una excepción **`weakref.ReferenceError`**.

```
proxy = weakref.proxy(obj)
proxy.attr # Equivalente a obj.attr
proxy.meth() # Equivalente a obj.meth()
del obj
proxy.attr # lanza weakref.ReferenceError
```

Ver también

PEP 205 [https://peps.python.org/pep-0205/] - Referencias débiles
Escrito e implementado por Fred L. Drake, Jr.

PEP 232: Atributos de la función

En Python 2.1, las funciones pueden tener información arbitraria adjunta. La gente solía usar cadenas de documentación para guardar información sobre funciones y métodos, porque el atributo `__doc__` era la única forma de adjuntar información a una función. Por ejemplo, en el servidor de aplicaciones web Zope, las funciones se marcan como seguras para el acceso público al tener una cadena de documentación, y en el marco de análisis SPARK de John Aycock, las cadenas de documentación contienen partes de la gramática BNF que se analizarán. Esta sobrecarga es desafortunada, ya que las cadenas de documentación están realmente pensadas para guardar la documentación de una función; por ejemplo, significa que no se pueden documentar adecuadamente las funciones destinadas a un uso privado en Zope.

Ahora se pueden establecer y recuperar atributos arbitrarios en las funciones utilizando la sintaxis normal de Python:

```
def f(): pass

f.publish = 1
f.secure = 1
f.grammar = "A ::= B (C D)*"
```

Se puede acceder al diccionario que contiene los atributos como `__dict__` de la función. A diferencia del atributo `__dict__` de las instancias de clase, en las funciones se puede asignar un nuevo diccionario a `__dict__`, aunque el nuevo valor está restringido a un diccionario Python normal; *can't* puede ser astuto y establecerlo en una instancia `UserDict` o cualquier otro objeto aleatorio que se comporte como una asignación.

Ver también

PEP 232 [https://peps.python.org/pep-0232/] - Atributos de la función
Escrito y ejecutado por Barry Warsaw.

PEP 235: Importación de módulos en plataformas que no distinguen entre mayúsculas y minúsculas

Algunos sistemas operativos tienen sistemas de archivos que no distinguen entre mayúsculas y minúsculas, siendo MacOS y Windows los principales ejemplos; en estos sistemas, es imposible distinguir los nombres de archivo `FILE.PY` y `file.py`, aunque almacenan el nombre del archivo en su caso original (también preservan las mayúsculas).

En Python 2.1, la sentencia `import` funcionará para simular la distinción entre mayúsculas y minúsculas en plataformas que no las distinguen. Python buscará ahora la primera coincidencia entre mayúsculas y minúsculas por defecto, lanzando un `ImportError` si no se encuentra dicho fichero, por lo que `import file` no importará un módulo llamado `FILE.PY`. La coincidencia insensible a mayúsculas y minúsculas puede solicitarse estableciendo la variable de entorno `PYTHONCASEOK` antes de iniciar el intérprete de Python.

PEP 217: Gancho de pantalla interactivo

Cuando se utiliza el intérprete de Python de forma interactiva, la salida de los comandos se muestra utilizando la función incorporada `repr()`. En Python 2.1, la variable `sys.displayhook()` puede establecerse a un objeto invocable que será llamado en lugar de `repr()`. Por ejemplo, puede establecerla a una función especial de impresión bonita:

```
>>> # Crea una estructura de datos recursiva
... L = [1,2,3]
>>> L.append(L)
>>> L # Muestra la salida predeterminada de Python
[1, 2, 3, [...]]
>>> # Usa pprint.pprint() como función de visualización
... import sys, pprint
>>> sys.displayhook = pprint.pprint
>>> L
[1, 2, 3, <Recursion on list with id=135143996>]
>>>
```

Ver también

PEP 217 [https://peps.python.org/pep-0217/] - Gancho de visualización para uso interactivo

Escrito y ejecutado por Moshe Zadka.

PEP 208: Nuevo modelo de coerción

Se ha modificado significativamente la forma en que se realiza la coerción numérica a nivel de C. Esto sólo afectará a los autores de las extensiones de C a Python, permitiéndoles más flexibilidad a la hora de escribir tipos de extensión que soporten operaciones numéricas.

Los tipos de extensión pueden ahora establecer el indicador de tipo `Py_TPFLAGS_CHECKTYPES` en su estructura `PyTypeObject` para indicar que soportan el nuevo modelo de coerción. En tales tipos de extensión, las funciones numéricas de ranura ya no pueden asumir que se les pasarán dos argumentos del mismo tipo; en su lugar, se les pueden pasar dos argumentos de tipos diferentes, y entonces pueden realizar su propia coerción interna. Si a la función de ranura se le pasa un tipo que no puede manejar, puede indicar el fallo retornando una referencia al valor singleton `Py_NotImplemented`. Las funciones numéricas del otro tipo serán entonces probadas, y quizás puedan manejar la operación; si el otro tipo también retorna `Py_NotImplemented`, entonces se levantará un **TypeError`Los métodos numéricos escritos en Python también pueden retornar ``Py_NotImplemented`**, haciendo que el intérprete actúe como si el método no existiera (tal vez lanzando un **`TypeError`**, tal vez probando los métodos numéricos de otro objeto).

Ver también

PEP 208 [https://peps.python.org/pep-0208/] - Reformulación del modelo de coerción

Escrito e implementado por Neil Schemenauer, basado en gran medida en el trabajo anterior de Marc-André Lemburg. Léalo para entender los puntos finos de cómo las operaciones numéricas serán ahora procesadas en el nivel C.

PEP 241: Metadatos en paquetes de Python

Una queja común de los usuarios de Python es que no existe un solo catálogo de todos los módulos de Python existentes. T. Middleton’s Vault of Parnassus en `www.vex.net/parnassus/` (borrado en febrero de 2009, disponible en Internet Archive Wayback Machine [https://web.archive.org/web/20090130140102/http://www.vex.net/parnassus/]) fue uno de los mayores catálogos de módulos de Python, pero registrar software en los archivos es opcional, y a mucha gente no le importó.

Como primer pequeño paso para solucionar el problema, el software de Python empaquetado con el comando **sdist** de Distutils incluirá un archivo llamado `PKG-INFO` que contiene información sobre el paquete, como su nombre, versión y autor (metadatos, en terminología de catalogación). **PEP 241** [https://peps.python.org/pep-0241/] contiene la lista completa de campos que pueden estar presentes en el archivo `PKG-INFO`A medida que la gente empiece a empaquetar su software usando Python 2.1, más y más paquetes incluirán metadatos, haciendo posible construir sistemas de catalogación automatizados y experimentar con ellosCon la experiencia resultante, tal vez sea posible diseñar un catálogo realmente bueno y luego construir soporte para él en Python 2.2. Por ejemplo, los comandos Distutils :command:`sdist` y **bdist_*** podrían soportar una opción `upload` que subiera automáticamente tu paquete a un servidor de catálogos.

Puedes empezar a crear paquetes que contengan `PKG-INFO` incluso si no estás usando Python 2.1, ya que se hará una nueva versión de las Distutils para los usuarios de versiones anteriores de PythonLa versión 1.0.2 de las Distutils incluye los cambios descritos en **PEP 241** [https://peps.python.org/pep-0241/], así como varias correcciones de errores y mejoras. Estará disponible en el SIG de Distutils en https://www.python.org/community/sigs/current/distutils-sig/.

Ver también

PEP 241 [https://peps.python.org/pep-0241/] - Metadatos para paquetes de software de Python

Escrito y ejecutado por A.M. Kuchling.

PEP 243 [https://peps.python.org/pep-0243/] - Mecanismo de carga del repositorio de módulos

Escrito por Sean Reifschneider, este borrador de PEP describe un mecanismo propuesto para subir paquetes de Python a un servidor central.

Módulos nuevos y mejorados

- Ka-Ping Yee contribuyó con dos nuevos módulos: **`inspect.py`**, un módulo para obtener información sobre código Python en vivo, y **`pydoc.py`**, un módulo para convertir de forma interactiva cadenas de documentación a HTML o texto. Como beneficio adicional, `Tools/scripts/pydoc`, que ahora se instala automáticamente, utiliza **`pydoc.py`** para mostrar la documentación según el nombre de un módulo, paquete o clase de Python. Por ejemplo, `pydoc xml.dom` muestra lo siguiente:

```
Documentación de la biblioteca de Python: paquete x

NOMBRE
xml.dom: implementación del modelo de objetos de do

ARCHIVO
/usr/local/lib/python2.1/xml/dom/__init__.pyc

DESCRIPCIÓN
La asignación de Python del modelo de objetos de do
Referencia de la biblioteca de Python en la sección

Este paquete contiene los siguientes módulos:
...
```

`pydoc` también incluye un navegador de ayuda interactiva basado en Tk. `pydoc` se vuelve rápidamente adictivo; ¡pruébalo!

- Se agregaron dos módulos diferentes para pruebas unitarias a la biblioteca estándar. El módulo **`doctest`**, aportado por Tim Peters, proporciona un marco de pruebas basado en la ejecución de ejemplos integrados en cadenas de documentación y la comparación de los resultados con la salida esperada. PyUnit, aportado por Steve Purcell, es un marco de pruebas unitarias inspirado en JUnit, que a su vez fue una adaptación del marco de pruebas Smalltalk de Kent Beck. Consulte https://pyunit.sourceforge.net/ para obtener más información sobre PyUnit.

- El módulo **`difflib`** contiene una clase, **`SequenceMatcher`**, que compara dos secuencias y calcula los cambios necesarios para transformar una secuencia en la otra. Por ejemplo, este módulo se puede utilizar para escribir una herramienta similar al programa **diff** de Unix y, de hecho, el programa de ejemplo `Tools/scripts/ndiff.py` demuestra cómo escribir un script de este tipo.

- `curses.panel`, una envoltura para la biblioteca de paneles, parte de ncurses y de curses SYSV, fue contribuida por Thomas Gellekum. La biblioteca de paneles proporciona ventanas con la característica adicional de la profundidad. Las ventanas pueden ser movidas más arriba o más abajo en el ordenamiento de la profundidad, y la librería de paneles calcula dónde se superponen los paneles y qué secciones son visibles.

- El paquete PyXML ha pasado por varias versiones desde Python 2.0, y Python 2.1 incluye una versión actualizada del paquete `xml`. Algunos de los cambios notables incluyen compatibilidad con Expat 1.2 y versiones posteriores, la capacidad de los analizadores de Expat para manejar archivos en cualquier codificación compatible con Python y varias correcciones de errores para SAX, DOM y el módulo `minidom`.

- Ping también contribuyó con otro gancho para manejar excepciones no detectadas. `sys.excepthook()` se puede configurar como un objeto invocable. Cuando una excepción no es detectada por ningún bloque `try`…`except`, la excepción se pasará a `sys.excepthook()`, que puede hacer lo que quiera. En la Novena Conferencia de Python, Ping demostró una aplicación para este gancho: imprimir un rastreo extendido que no solo enumera los marcos de la pila, sino que también enumera los argumentos de la función y las variables locales para cada marco.

- Varias funciones del módulo `time`, como `asctime()` y `localtime()`, requieren un argumento de punto flotante que contenga el tiempo en segundos desde la época. El uso más común de estas funciones es trabajar con la hora actual, por lo que el argumento de punto flotante se ha vuelto opcional; cuando no se proporciona un valor, se utilizará la hora actual. Por ejemplo, las entradas de archivos de registro generalmente necesitan una cadena que contenga la hora actual; en Python 2.1, se puede utilizar `time.asctime()`, en lugar del `time.asctime(time.localtime(time.time()))` más extenso que se requería anteriormente.

Este cambio fue propuesto y aplicado por Thomas Wouters.

- El módulo **`ftplib`** ahora recupera por defecto los ficheros en modo pasivo, porque es más probable que el modo pasivo funcione desde detrás de un cortafuegos. Esta petición vino del sistema de seguimiento de errores de Debian, ya que otros paquetes de Debian utilizan **`ftplib`** para recuperar archivos y entonces no funcionan desde detrás de un cortafuegos. Se considera poco probable que esto cause problemas a nadie, porque Netscape utiliza por defecto el modo pasivo y poca gente se queja, pero si el modo pasivo no es adecuado para su aplicación o configuración de red, llame a `set_pasv(0)` en los objetos FTP para desactivar el modo pasivo.

- Se ha añadido soporte para el acceso a sockets sin procesar en el módulo **`socket`**, aportado por Grant Edwards.

- El módulo **`pstats`** contiene ahora un sencillo navegador de estadísticas interactivo para mostrar los perfiles de tiempo de los programas de Python, invocado cuando el módulo se ejecuta como un script. Contribuido por Eric S. Raymond.

- Se ha agregado una nueva función dependiente de la implementación, `sys._getframe([depth])`, para devolver un objeto de marco determinado de la pila de llamadas actual. **`sys._getframe()`** devuelve el marco en la parte superior de la pila de llamadas; si el argumento de entero opcional *depth* is supplied, the function returns the frame that is *depth* llama debajo de la parte superior de la pila. Por ejemplo, `sys._getframe(1)` devuelve el objeto de trama de la persona que llama.

  Esta función sólo está presente en CPython, no en Jython ni en la implementación de .NET. Utilízala para depurar, y resiste la tentación de ponerla en el código de producción.

Otros cambios y correcciones

En Python 2.1 se hicieron relativamente pocos cambios pequeños debido al ciclo de publicación más corto. Una búsqueda en los registros de cambios de CVS muestra 117 parches aplicados y 136 errores corregidos; es probable que ambas cifras estén subestimadas. Algunos de los cambios más notables son:

- Ahora está disponible opcionalmente un asignador de objetos especializado, que debería ser más rápido que el `malloc()` del sistema y tener menos sobrecarga de memoria. El asignador utiliza la función `malloc()` de C para obtener grandes grupos de memoria y luego cumple con las solicitudes de memoria más pequeñas de estos grupos. Se puede habilitar proporcionando la opción `--with-pymalloc` al script **configure**; consulte `Objects/obmalloc.c` para obtener los detalles de implementación.

  Los autores de módulos de extensión de C deberían probar su código con el asignador de objetos habilitado, porque algún código incorrecto puede fallar, causando volcados de memoria en tiempo de ejecución. Hay un montón de funciones de asignación de memoria en la API de C de Python que anteriormente eran solo alias para `malloc()` y `free()` de la biblioteca de C, lo que significa que si accidentalmente llamabas a funciones que no coinciden, el error no sería perceptible. Cuando el asignador de objetos está habilitado, estas funciones ya no son alias de `malloc()` y `free()`, y llamar a la función incorrecta para liberar memoria te dará un volcado de memoria. Por ejemplo, si la memoria se asignó usando `PyMem_New`, debe liberarse usando `PyMem_Del()`, no `free()`. Algunos módulos incluidos con Python tuvieron problemas con esto y tuvieron que ser reparados; sin duda hay más módulos de terceros que tendrán el mismo problema.

  El asignador de objetos fue aportado por Vladimir Marangozov.

- La velocidad de la E/S de archivos orientada a líneas se ha mejorado porque la gente suele quejarse de su falta de velocidad y porque a menudo se ha utilizado como un punto de referencia ingenuo. Por lo tanto, el método **`readline()`** de objetos de archivo se ha reescrito para que sea mucho más rápido. La cantidad exacta de la aceleración variará de una plataforma a otra dependiendo de lo lento que fuera el **`getc()`** de la biblioteca C, pero es de alrededor del 66% y potencialmente mucho más rápido en algunos sistemas operativos particulares. Tim Peters realizó gran parte de la evaluación comparativa y la codificación para este cambio, motivado por una discusión en comp.lang.python.

También se agregó un nuevo módulo y método para objetos de archivo, aportado por Jeff Epler. El nuevo método, **`xreadlines()`**, es similar al **`xrange()`** existente. **`xreadlines()`** devuelve un objeto de secuencia opaca que solo admite iteraciones, leyendo una línea en cada iteración, pero no leyendo el archivo completo en la memoria como lo hace el método **`readlines()`** existente. Lo usaría de esta manera:

```
para la línea en sys.stdin.xreadlines():
# ... hacer algo para cada línea ...
...
```

Para una discusión más completa de los cambios en la línea de E/S, véase el resumen de python-dev del 1 al 15 de enero de 2001 en https://mail.python.org/pipermail/python-dev/2001-January/.

- Se agregó un nuevo método, **`popitem()`**, a los diccionarios para permitir la iteración destructiva a través del contenido de un diccionario; esto puede ser más rápido para diccionarios grandes porque no hay necesidad de construir una lista que contenga todas las claves o valores. `D.popitem()` elimina un par aleatorio `(key, value)` del diccionario `D` y lo devuelve como una tupla de 2. Esto fue implementado principalmente por Tim Peters y Guido van Rossum, después de una sugerencia y un parche preliminar de Moshe Zadka.
- Ahora los módulos pueden controlar qué nombres se

importan cuando se utiliza `from module import *`, definiendo un atributo `__all__` que contiene una lista de nombres que se importarán. Una queja común es que si el módulo importa otros módulos como **`sys`** o **`string`**, `from module import *` los añadirá al espacio de nombres del módulo importador. Para arreglar esto, simplemente liste los nombres públicos en `__all__`:

```
# Lista de nombres públicos
__all__ = ['Base de datos', 'abierta']
```

Una versión más estricta de este parche fue primero sugerida e implementada por Ben Wolfson, pero después de algunas discusiones de python-dev, una versión final más débil fue revisada.

- Al aplicar **`repr()`** a las cadenas, antes se utilizaban escapes octales para los caracteres no imprimibles; por ejemplo, una nueva línea era `'\012'`. Esto era un vestigio de la ascendencia de Python en C, pero hoy en día el octal tiene muy poco uso práctico. Ka-Ping Yee sugirió usar escapes hexadecimales en lugar de octales, y usar los escapes `\n`, `\t`, `\r` para los caracteres apropiados, e implementó este nuevo formato.

- Los errores de sintaxis detectados en tiempo de compilación pueden ahora lanzar excepciones que contienen el nombre del archivo y el número de línea del error, un agradable efecto secundario de la reorganización del compilador realizada por Jeremy Hylton.

- Las extensiones C que importan otros módulos se han modificado para utilizar **`PyImport_ImportModule()`**, lo que significa que utilizarán cualquier gancho de importación que se haya instalado. Esto también se recomienda para extensiones de terceros que necesiten importar algún otro módulo desde el código C.

- El tamaño de la base de datos de caracteres Unicode se redujo en otros 340K gracias a Fredrik Lundh.

- Se han aportado algunos puertos nuevos: MacOS X (por Steven Majewski), Cygwin (por Jason Tishler); RISCOS (por Dietmar Schwertberger); Unixware 7 (por Billy G. Allie).

Y hay la lista habitual de correcciones de errores menores, fugas de memoria menores, ediciones de docstrings, y otros ajustes, demasiado largos para que valga la pena detallarlos; vea los registros de CVS para los detalles completos si los quiere.

Agradecimientos

El autor desea agradecer a las siguientes personas sus sugerencias sobre varios borradores de este artículo: Graeme Cross, David Goodger, Jay Graves, Michael Hudson, Marc-André Lemburg, Fredrik Lundh, Neil Schemenauer, Thomas Wouters.

Qué hay de nuevo en Python 2.0

Autor:

A.M. Kuchling y Moshe Zadka

Introducción

El 16 de octubre de 2000 se publicó una nueva versión de Python, la 2.0. Este artículo cubre las emocionantes nuevas características de la 2.0, destaca algunos otros cambios útiles y señala algunos cambios incompatibles que pueden requerir reescribir el código.

El desarrollo de Python nunca se detiene por completo entre versiones, y siempre se envía un flujo constante de correcciones de errores y mejoras. Una gran cantidad de correcciones menores, algunas optimizaciones, cadenas de documentación adicionales y mejores mensajes de error entraron en 2.0; enumerarlos a todos sería imposible, pero ciertamente son significativos. Consulte los registros de CVS disponibles públicamente si desea ver la lista completa. Este progreso se debe a que a los cinco desarrolladores que trabajan para PythonLabs ahora se les paga por dedicar sus días a corregir errores, y también a la mejora de la comunicación resultante de la migración a SourceForge.

¿Qué pasa con Python 1.6?

Python 1.6 puede considerarse como la versión de Python de las Obligaciones Contractuales. Después de que el equipo principal de desarrollo dejara el CNRI en mayo de 2000, el CNRI pidió que se creara una versión 1.6 que contuviera todo el trabajo sobre Python que se había realizado en el CNRI. Por lo tanto, Python 1.6 representa el estado del árbol CVS en mayo de 2000, siendo la novedad más importante el soporte de Unicode. El desarrollo continuó después de mayo, por supuesto, así que el árbol 1.6 recibió algunas correcciones para asegurar que es compatible con Python 2.0. 1.Por lo tanto, la 6 es parte de la evolución de Python, y no una rama lateral.

Entonces, ¿deberías interesarte mucho por Python 1.6? Probablemente no. Las versiones 1.6final y 2.0beta1 se publicaron el mismo día (5 de septiembre de 2000), y el plan es finalizar Python 2.0 en un mes más o menos. Si tienes aplicaciones que mantener, no parece que tenga mucho sentido romper cosas al pasar a la 1.6, arreglarlas, y luego tener otra ronda de roturas dentro de un mes al pasar a la 2.0; es mejor pasar directamente a la 2.0. La mayoría de las características realmente interesantes descritas en este documento sólo están en la 2.0, porque se hizo mucho trabajo entre mayo y septiembre.

Nuevo proceso de desarrollo

El cambio más importante en Python 2.0 puede que no sea en el código en absoluto, sino en la forma de desarrollar Python: en mayo de 2000 los desarrolladores de Python comenzaron a utilizar las herramientas puestas a disposición por SourceForge para almacenar el código fuente, rastrear los informes de errores y gestionar la cola de envíos de parches. Para informar de errores o enviar parches para Python 2.0, utilice las herramientas de seguimiento de errores y gestión de parches disponibles en la página del proyecto Python, situada en https://sourceforge.net/projects/python/.

El más importante de los servicios alojados ahora en SourceForge es el árbol CVS de Python, el repositorio de versiones controladas que contiene el código fuente de Python. Anteriormente, había unas 7 personas que tenían acceso de escritura al árbol CVS, y todos los parches tenían que ser inspeccionados y comprobados por una de las personas de esta corta lista. Obviamente, esto no era muy escalable. Al trasladar el árbol CVS a SourceForge, fue posible conceder acceso de escritura a más personas; en septiembre de 2000 había 27 personas que podían revisar los cambios, un aumento de cuatro veces. Esto hace posible cambios a gran escala que no se intentarían si tuvieran que pasar por el pequeño grupo de desarrolladores del núcleo. Por ejemplo, un día a Peter Schneider-

Kamp se le ocurrió dejar de lado la compatibilidad con K&R C y convertir el código fuente de Python a ANSI C. Después de obtener la aprobación en la lista de correo de python-dev, se lanzó a una ráfaga de revisiones que duró aproximadamente una semana, otros desarrolladores se unieron para ayudar, y el trabajo estaba hecho. Si sólo hubiera habido 5 personas con acceso de escritura, probablemente esa tarea habría sido considerada como «agradable, pero no vale la pena el tiempo y el esfuerzo necesarios» y nunca se habría realizado.

El cambio al uso de los servicios de SourceForge ha dado lugar a un notable aumento de la velocidad de desarrollo. Ahora los parches se envían, se comentan, son revisados por otras personas además del remitente original, y van de un lado a otro hasta que se considera que el parche merece ser revisado. Los errores se rastrean en una ubicación central y se pueden asignar a una persona específica para que los corrija, y podemos contar el número de errores abiertos para medir el progreso. Esto no ha tenido un coste: los desarrolladores tienen ahora más correo electrónico con el que lidiar, más listas de correo que seguir, y se han tenido que escribir herramientas especiales para el nuevo entorno. Por ejemplo, SourceForge envía por defecto mensajes de correo electrónico de notificación de parches y errores que son completamente inútiles, por lo que Ka-Ping Yee escribió un raspador de pantalla HTML que envía mensajes más útiles.

La facilidad para añadir código provocó algunos problemas iniciales de crecimiento, como el hecho de que el código se registrara antes de estar listo o sin obtener un acuerdo claro del grupo de desarrolladores. El proceso de aprobación que ha surgido es algo similar al utilizado por el grupo Apache. Los desarrolladores pueden votar +1, +0, -0 o -1 sobre un parche; +1 y -1 denotan aceptación o rechazo, mientras que +0 y -0 significan que el desarrollador es mayormente indiferente al cambio, aunque con un ligero sesgo positivo o negativo. El cambio más significativo con respecto al modelo de Apache es que la votación es esencialmente consultiva, lo que permite a Guido van Rossum, que tiene el estatus de Dictador Benevolente Vitalicio, saber cuál es la opinión general. Puede seguir ignorando el resultado de una votación y aprobar o rechazar un cambio aunque la comunidad no esté de acuerdo con él.

Producir un parche real es el último paso en la adición de una nueva característica, y suele ser fácil en comparación con la tarea anterior de llegar a un buen diseño. Las discusiones sobre nuevas funcionalidades a menudo pueden explotar en largos hilos de la lista de correo, haciendo que la discusión sea difícil de seguir, y nadie puede leer todos los mensajes en python-dev. Por lo tanto, se ha establecido un proceso relativamente formal para escribir Propuestas de Mejora de Python (PEPs), siguiendo el modelo del proceso RFC de Internet. Las PEP son borradores de documentos que describen una nueva característica propuesta, y se revisan continuamente hasta que la comunidad llega a un consenso, aceptando o rechazando la propuesta. Cita de la introducción de **PEP 1** [https://peps.python.org/pep-0001/], «PEP Purpose and Guidelines»:

> PEP son las siglas de Python Enhancement Proposal. Un PEP es un documento de diseño que proporciona información a la comunidad de Python, o que describe una nueva característica para Python. El PEP debe proporcionar una especificación técnica concisa de la característica y una justificación de la misma.
>
> Pretendemos que los PEPs sean los mecanismos principales para proponer nuevas características, para recoger las opiniones de la comunidad sobre un tema y para documentar las decisiones de diseño que se han tomado en Python. El autor del PEP es responsable de crear consenso dentro de la comunidad y de documentar las opiniones discrepantes.

Lea el resto de **PEP 1** [https://peps.python.org/pep-0001/] para conocer los detalles del proceso editorial, el estilo y el formato de PEP. Los PEP se guardan en el árbol CVS de Python en SourceForge, aunque no son parte de la distribución de Python 2.0, y también están disponibles en formato HTML en https://peps.python.org/. A partir de septiembre de 2000, existen 25 PEP, que van desde **PEP 201** [https://peps.python.org/pep-0201/], «Iteración en sincronía», hasta PEP 225, «Operadores por elementos y por objetos».

Unicode

La mayor novedad de Python 2.0 es un nuevo tipo de datos fundamental: Las cadenas Unicode. Unicode utiliza números de 16 bits para representar los caracteres en lugar de los 8 bits utilizados por ASCII, lo que significa que se pueden admitir 65.536 caracteres distintos.

La interfaz final para el soporte de Unicode se alcanzó a través de innumerables discusiones, a menudo tormentosas, en la lista de correo de python-dev, y fue implementada en su mayor parte por Marc-André Lemburg, basándose en una implementación del tipo de cadena Unicode de Fredrik Lundh. Una explicación detallada de la interfaz fue escrita como **PEP 100** [https://peps.python.org/pep-0100/], «Python Unicode Integration». Este artículo se limitará a cubrir los puntos más significativos de las interfaces Unicode.

En el código fuente de Python, las cadenas Unicode se escriben como `u"string"`. Se pueden escribir caracteres Unicode arbitrarios utilizando una nueva secuencia de escape, `\u`*`HHHH`*, donde *HHHH* es un número hexadecimal de 4 dígitos de 0000 a FFFF. También se puede utilizar la secuencia de escape existente `\x`*`HH`*, y se pueden utilizar escapes octales para caracteres hasta U +01FF, que se representa mediante `\777`.

Las cadenas Unicode, al igual que las cadenas normales, son un tipo de secuencia inmutable. Pueden ser indexadas y cortadas, pero no modificadas en su lugar. Las cadenas Unicode tienen un método `encode( [encoding] )` que retorna una cadena de 8 bits en la codificación deseada. Las codificaciones son nombradas por cadenas, como `'ascii'`, `'utf-8'`, `'iso-8859-1'`, o lo que sea. Se define una API de códecs para implementar y registrar nuevas codificaciones que luego están disponibles en todo el programa Python. Si no se especifica una codificación, la codificación por defecto suele ser ASCII de 7 bits, aunque puede cambiarse para tu instalación de Python llamando a la función `sys.setdefaultencoding(encoding)` en una versión personalizada de `site.py`.

La combinación de cadenas de 8 bits y Unicode siempre fuerza conversión a Unicode, utilizando la codificación ASCII por defecto; el resultado de `'a' + u'bc'` es `u'abc'`.

Se han añadido nuevas funciones incorporadas y se han modificado las existentes para que sean compatibles con Unicode:

- `unichr(ch)` retorna una cadena Unicode de 1 carácter, que contiene el carácter *ch*.
- `ord(u)`, donde *u* es una cadena regular o Unicode de 1 carácter, retorna el número del carácter como un entero.
- `unicode(string [, encoding] [, errors] )` crea una cadena Unicode a partir de una cadena de 8 bits. `encoding` es una cadena que nombra la codificación a utilizar. El parámetro `errors` especifica el tratamiento de los caracteres que no son válidos para la codificación actual; pasar `'strict` como valor hace que se lance una excepción en cualquier error de codificación, mientras que `'ignore` hace que los errores se ignoren silenciosamente y `'replace` utiliza U+FFFD, el carácter oficial de reemplazo, en caso de cualquier problema.
- La sentencia `exec`, y varias funciones integradas como `eval()`, `getattr()`, y `setattr()` también aceptarán cadenas Unicode así como cadenas regulares. (Es posible que en el proceso de corrección de esto se hayan pasado por alto algunas funciones incorporadas; si encuentra una función incorporada que acepte cadenas pero que no acepte cadenas Unicode en absoluto, por favor, infórmelo como un error)

Un nuevo módulo, **`unicodedata`**, proporciona una interfaz para las propiedades de los caracteres Unicode. Por ejemplo, `unicodedata.category(u'A')` retorna la cadena de 2 caracteres “Lu”, la “L” denota que es una letra, y la “u” significa que es mayúscula. `unicodedata.bidirectional(u'\u0660')` retorna “AN”, lo que significa que U+0660 es un número árabe.

El módulo **`codecs`** contiene funciones para buscar codificaciones existentes y registrar otras nuevas. A menos que quiera implementar una nueva codificación, lo más habitual es que utilice la función `codecs.lookup(encoding)`, que retorna una tupla de 4 elementos: `(encode_func, decode_func, stream_reader, stream_writer)`.

- *encode_func* es una función que toma una cadena Unicode, y retorna una 2-tupla `(string, length)`. *string* es una cadena de 8 bits que contiene una porción (tal vez toda) de la cadena Unicode convertida a la codificación dada, y *longitud* indica la cantidad de cadena Unicode convertida.
- *decode_func* es lo opuesto a *encode_func*, tomando una cadena de 8 bits y retornando una 2-tupla `(ustring, length)`, que consiste en la cadena Unicode resultante *ustring* y el entero *length* que dice cuánto de la cadena de 8 bits se consumió.
- *stream_reader* es una clase que admite la decodificación de la entrada de un flujo. *stream_reader(file_obj)* devuelve un objeto que admite los métodos **`read()`**, **`readline()`** y **`readlines()`**. Todos estos métodos traducirán a partir de la codificación dada y devolverán cadenas Unicode.
- De manera similar, *stream_writer* es una clase que admite la codificación de la salida en una secuencia. *stream_writer(file_obj)* devuelve un objeto que admite los métodos **`write()`** y **`writelines()`**. Estos métodos esperan cadenas Unicode y las traducen a la codificación dada en la salida.

Por ejemplo, el siguiente código escribe una cadena Unicode en un archivo, codificándola como UTF-8:

```
import codecs

unistr = u'\u0660\u2000ab ...'

(UTF8_encode, UTF8_decode,
 UTF8_streamreader, UTF8_streamwriter) = codecs.lookup('

output = UTF8_streamwriter( open( '/tmp/output', 'wb') )
output.write( unistr )
output.close()
```

El siguiente código leería la entrada UTF-8 del archivo:

```
input = UTF8_streamreader( open( '/tmp/output', 'rb') )
print repr(input.read())
```

```
input.close()
```

Las expresiones regulares compatibles con Unicode están disponibles a través del módulo **`re`**, que tiene una nueva implementación subyacente llamada SRE escrita por Fredrik Lundh de Secret Labs AB.

Se ha añadido una opción de línea de comandos `-U` que hace que el compilador de Python interprete todos los literales de cadena como literales de cadena Unicode. Esta opción está pensada para ser utilizada en las pruebas y para asegurar el futuro de su código Python, ya que alguna versión futura de Python puede dejar de soportar cadenas de 8 bits y proporcionar sólo cadenas Unicode.

Comprensión de listas

Las listas son un tipo de datos muy útil en Python, y muchos programas manipulan una lista en algún momento. Dos operaciones comunes en las listas son hacer un bucle sobre ellas, y escoger los elementos que cumplen un cierto criterio, o aplicar alguna función a cada elemento. Por ejemplo, dada una lista de cadenas, podrías querer sacar todas las cadenas que contengan una determinada subcadena, o quitar los espacios en blanco de cada línea.

Las funciones **`map()`** y **`filter()`** existentes pueden usarse para este propósito, pero requieren una función como uno de sus argumentos. Esto está bien si hay una función incorporada que se puede pasar directamente, pero si no la hay, hay que crear una pequeña función para hacer el trabajo requerido, y las reglas de ámbito de Python hacen que el resultado sea feo si la pequeña función necesita información adicional. Tomemos el primer ejemplo del párrafo anterior, encontrar todas las cadenas de la lista que contienen una subcadena dada. Podrías escribir lo siguiente para hacerlo:

```
# Given the list L, make a list of all strings
# containing the substring S.
sublist = filter( lambda s, substring=S:
                     string.find(s, substring) != -1,
                  L)
```

Debido a las reglas de ámbito de Python, se utiliza un argumento por defecto para que la función anónima creada por la expresión **`lambda`** sepa qué subcadena se está buscando. Las comprensiones de lista hacen esto más limpio:

```
sublista = [ s para s en L si cadena.find(s, S) != -1 ]
```

Las comprensiones de listas tienen la forma:

```
[ expression for expr in sequence1
             for expr2 in sequence2 ...
             for exprN in sequenceN
             if condition ]
```

Las cláusulas **`for`**...**`in`** contienen las secuencias a iterar. Las secuencias no tienen por qué tener la misma longitud, ya que no se itera sobre ellas en paralelo, sino de izquierda a derecha; esto se explica más claramente en los párrafos siguientes. Los elementos de la lista generada serán los valores sucesivos de la *expresión*. La cláusula final **`if`** es opcional; si está presente, la *expresión* sólo se evalúa y se añade al resultado si la *condición* es verdadera.

Para dejar muy clara la semántica, una comprensión de lista equivale al siguiente código de Python:

```
for expr1 in sequence1:
    for expr2 in sequence2:
    ...
        for exprN in sequenceN:
             if (condition):
                  # Append the value of
                  # the expression to the
                  # resulting list.
```

Esto significa que cuando hay múltiples cláusulas **`for`**...**`in`**, la lista resultante será igual al producto de las longitudes de todas las secuencias. Si tiene dos listas de longitud 3, la lista de salida tendrá 9 elementos:

```
seq1 = 'abc'
seq2 = (1,2,3)
```

```
>>> [ (x,y) for x in seq1 for y in seq2]
[('a', 1), ('a', 2), ('a', 3), ('b', 1), ('b', 2), ('b',
('c', 2), ('c', 3)]
```

Para evitar introducir una ambigüedad en la gramática de Python, si *expresión* está creando una tupla, debe estar rodeada de paréntesis. La primera comprensión de la lista a continuación es un error de sintaxis, mientras que la segunda es correcta:

```
# Syntax error
[ x,y for x in seq1 for y in seq2]
# Correct
[ (x,y) for x in seq1 for y in seq2]
```

La idea de las comprensiones de listas procede originalmente del lenguaje de programación funcional Haskell (https://www.haskell.org). Greg Ewing fue el que más abogó por añadirlas a Python y escribió el parche inicial de comprensión de listas, que luego se discutió durante un tiempo aparentemente interminable en la lista de correo de python-dev y se mantuvo actualizada por Skip Montanaro.

Asignación aumentada

Los operadores de asignación aumentados, otra característica largamente solicitada, han sido añadidos a Python 2.0. Los operadores de asignación aumentados incluyen `+=`, `-=`, `*=`, etc. Por ejemplo, la sentencia `a += 2` incrementa el valor de la variable `a` en 2, lo que equivale a la sentencia algo más larga `a = a + 2`.

La lista completa de operadores de asignación admitidos es `+=`, `-=`, `*=`, `/=`, `%=`, `**=`, `&=`, `|=`, `^=`, `>>=` y `<<=`. Las clases de Python pueden anular los operadores de asignación aumentados definiendo métodos denominados **`__iadd__()`**, **`__isub__()`**, etc. Por ejemplo, la siguiente clase **`Number`** almacena un número y admite el uso de + = para crear una nueva instancia con un valor incrementado.

```
class Number:
    def __init__(self, value):
        self.value = value
    def __iadd__(self, increment):
        return Number( self.value + increment)

n = Number(5)
n += 3
print n.value
```

El método especial **`__iadd__()`** se llama con el valor del incremento y debe devolver una nueva instancia con un valor modificado apropiadamente; este valor de retorno está vinculado como el nuevo valor de la variable en el lado izquierdo.

Los operadores de asignación aumentada se introdujeron por primera vez en el lenguaje de programación C, y la mayoría de los lenguajes derivados de C, como **awk**, C++, Java, Perl y PHP también los soportan. El parche de asignación aumentada fue implementado por Thomas Wouters.

Métodos de cadena de caracteres

Hasta ahora, la funcionalidad de manipulación de cadenas estaba en el módulo **`string`**, que normalmente era una interfaz para el módulo **`strop`** escrito en C. La incorporación de Unicode planteó una dificultad para el módulo **`strop`**, porque todas las funciones debían reescribirse para aceptar cadenas de 8 bits o Unicode. Para funciones como **`string.replace()`**, que acepta 3 cadenas como argumentos, eso significa ocho permutaciones posibles y, en consecuencia, un código complicado.

En cambio, Python 2.0 traslada el problema al tipo de cadena de caracteres, haciendo que la funcionalidad de manipulación de cadenas esté disponible a través de métodos tanto en cadenas de 8 bits como en cadenas Unicode.

```
>>> 'andrew'.capitalize()
'Andrew'
>>> 'hostname'.replace('os', 'linux')
'hlinuxtname'
```

```
>>> 'moshe'.find('sh')
2
```

Una cosa que no ha cambiado, a pesar de una notable broma de April Fools, es que las cadenas de Python son inmutables. Así, los métodos de cadenas retornan cadenas nuevas, y no modifican la cadena sobre la que operan.

El antiguo módulo **`string`** sigue existiendo por compatibilidad con el pasado, pero actúa principalmente como un front-end para los nuevos métodos de cadena de caracteres.

Dos métodos que no tienen paralelo en versiones anteriores a la 2.0, aunque existieron en JPython durante bastante tiempo, son **`startswith()`** y **`endswith()`**. `s.startswith(t)` es equivalente a `s[:len(t)] == t`, mientras que `s.endswith(t)` es equivalente a `s[-len(t):] == t`.

Otro método que merece una mención especial es el **`join()`**. El método **`join()`** de una cadena recibe un parámetro, una secuencia de cadenas, y es equivalente a la función **`string.join()`** del antiguo módulo **`string`**, con los argumentos invertidos. En otras palabras, el `s.join(seq)` es equivalente al antiguo `string.join(seq, s)`.

Recogida de basura de los ciclos

La implementación en C de Python utiliza el conteo de referencias para implementar la recolección de basura. Cada objeto de Python mantiene un recuento del número de referencias que apuntan a sí mismo, y ajusta el recuento a medida que se crean o destruyen referencias. Una vez que el recuento de referencias llega a cero, el objeto deja de ser accesible, ya que es necesario tener una referencia a un objeto para acceder a él, y si el recuento es cero, ya no existen referencias.

El conteo de referencias tiene algunas propiedades agradables: es fácil de entender e implementar, y la implementación resultante es portable, bastante rápida, y reacciona bien con otras bibliotecas que implementan sus propios esquemas de manejo de memoria. El mayor problema del conteo de referencias es que a veces no se da cuenta de que los objetos ya no son accesibles, lo que provoca una fuga de memoria. Esto ocurre cuando hay ciclos de referencias.

Consideremos el ciclo más simple posible, una instancia de clase que tiene una referencia a sí misma:

```
instancia = SomeClass()
instancia.myself = instancia
```

Después de ejecutar las dos líneas de código anteriores, la cuenta de referencias de `instance` es 2; una referencia es de la variable llamada `'instance`, y la otra es del atributo `myself` de la instancia.

Si la siguiente línea de código es `del instance`, ¿qué ocurre? La cuenta de referencias de `instance` se reduce en 1, por lo que tiene una cuenta de referencias de 1; la referencia en el atributo `myself` sigue existiendo. Sin embargo, la instancia ya no es accesible a través del código de Python, y podría ser eliminada. Varios objetos pueden participar en un ciclo si tienen referencias entre sí, haciendo que todos los objetos se filtren.

Python 2.0 soluciona este problema ejecutando periódicamente un algoritmo de detección de ciclos que busca los ciclos inaccesibles y borra los objetos implicados. Un nuevo módulo **`gc`** proporciona funciones para realizar una recolección de basura, obtener estadísticas de depuración y afinar los parámetros del recolector.

Ejecutar el algoritmo de detección de ciclos lleva algo de tiempo, y por lo tanto resultará en una sobrecarga adicional. Se espera que después de que hayamos adquirido experiencia con la recogida de ciclos al utilizar la versión 2.0, Python 2.1 sea capaz de minimizar la sobrecarga con un cuidadoso ajuste. Todavía no es obvio cuánto rendimiento se pierde, porque la evaluación comparativa de esto es difícil y depende crucialmente de la frecuencia con la que el programa crea y destruye objetos. La detección de ciclos puede ser desactivada cuando Python es compilado, si no puede permitirse ni siquiera una pequeña penalización de velocidad o sospecha que la recolección de ciclos es un error, especificando la opción **`--without-cycle-gc`** cuando se ejecuta el script **configure**.

Varias personas abordaron este problema y contribuyeron a una solución. Una primera implementación del enfoque de detección de ciclos fue escrita por Toby Kelsey. El algoritmo actual fue sugerido por Eric Tiedemann durante una visita al CNRI, y Guido van Rossum y Neil Schemenauer escribieron dos implementaciones diferentes, que posteriormente fueron integradas por Neil. Muchas otras personas ofrecieron sugerencias a lo largo del camino; los archivos de marzo de 2000 de la lista de correo python-dev contienen la mayor parte de la discusión relevante, especialmente en los hilos titulados «Colección de ciclos de referencia para Python» y «Finalización de nuevo».

Otros cambios en el núcleo

Se han realizado varios cambios menores en la sintaxis y las funciones incorporadas de Python. Ninguno de los cambios es de gran alcance, pero son conveniencias prácticas.

Cambios menores del lenguaje

Una nueva sintaxis hace que sea más conveniente llamar a una función dada con una tupla de argumentos y/o un diccionario de argumentos de palabras clave. En Python 1.5 y anteriores, se usaba la función incorporada **`apply()`**: `apply(f, args, kw)` llama a la función **`f()`** con la tupla de argumentos *args* y los argumentos de palabras clave en el diccionario *kw*. **`apply()`** es lo mismo en 2.0, pero gracias a un parche de Greg Ewing, `f(*args, **kw)` es una forma más corta y clara de lograr el mismo efecto. Esta sintaxis es simétrica con la sintaxis para definir funciones:

```
def f(*args, **kw):
    # args is a tuple of positional args,
    # kw is a dictionary of keyword args
    ...
```

Ahora, la salida de la sentencia `print` puede dirigirse a un objeto similar a un archivo si se sigue `print` con `>> file`, de forma similar al operador de redirección de los shells de Unix. Antes, se tenía que utilizar el método **`write()`** del objeto similar a un archivo, que carece de la comodidad y la simplicidad de `print`, o se podía asignar un nuevo valor a `sys.stdout` y luego restaurar el valor anterior. Para enviar la salida al error estándar, es mucho más fácil escribir esto:

```
print >> sys.stderr, "Warning: action field not supplie
```

Ahora se puede cambiar el nombre de los módulos al importarlos, utilizando la sintaxis `import module as name` o `from module import name as othername`. El parche fue enviado por Thomas Wouters.

Un nuevo estilo de formato está disponible cuando se utiliza el operador `%`; “%r” insertará el **`repr()`** de su argumento. Esto también se añadió por consideraciones de simetría, esta vez por simetría con el estilo de formato existente “%s”, que inserta el **`str()`** de su argumento. Por ejemplo, `'%r %s' % ('abc', 'abc')` retorna una cadena que contiene `'abc' abc`.

Anteriormente no había forma de implementar una clase que anulara el operador **`in`** incorporado de Python e implementara una versión personalizada. `obj in seq` devuelve verdadero si *obj* está presente en la secuencia *seq*; Python calcula esto simplemente probando cada índice de la secuencia hasta que se encuentre *obj* o **`IndexError`**. Moshe Zadka contribuyó con un parche que agrega un método mágico **`__contains__()`** para proporcionar una implementación personalizada para **`in`**. Además, los nuevos objetos incorporados escritos en C pueden definir lo que **`in`** significa para ellos a través de una nueva ranura en el protocolo de secuencia.

Las versiones anteriores de Python utilizaban un algoritmo recursivo para borrar objetos. Las estructuras de datos muy anidadas podían hacer que el intérprete llenara la pila de C y se bloqueara; Christian Tismer reescribió la lógica de borrado para solucionar este problema. En una nota relacionada, la comparación de objetos recursivos se repite infinitamente y se bloquea; Jeremy Hylton reescribió el código para que no se bloquee, produciendo un resultado útil. Por ejemplo, después de este código:

```
a = []
b = []
```

```
a.append(a)
b.append(b)
```

La comparación `a==b` devuelve verdadero, porque las dos estructuras de datos recursivas son isomorfas. Consulte el hilo «trashcan y PR#7» en los archivos de abril de 2000 de la lista de correo python-dev para ver la discusión que condujo a esta implementación y algunos enlaces relevantes útiles. Tenga en cuenta que las comparaciones ahora también pueden generar excepciones. En versiones anteriores de Python, una operación de comparación como `cmp(a,b)` siempre produciría una respuesta, incluso si un método **`__cmp__()`** definido por el usuario encontrara un error, ya que la excepción resultante simplemente se tragaría silenciosamente.

Se ha trabajado en la migración de Python a Windows de 64 bits en el procesador Itanium, principalmente por Trent Mick de ActiveState. (Confusamente, `sys.platform` sigue siendo `'win32'` en Win64 porque parece que, para facilitar la migración, MS Visual C++ trata el código como de 32 bits en Itanium). PythonWin también es compatible con Windows CE; consulte la página de Python CE en https://pythonce.sourceforge.net/ para obtener más información.

Otra plataforma nueva es Darwin/MacOS X; el soporte inicial para ella está en Python 2.0. La carga dinámica funciona, si se especifica «configure –with-dyld –with-suffix=.x». Consulte el README de la distribución de fuentes de Python para obtener más instrucciones.

Se ha intentado aliviar uno de los defectos de Python, la a menudo confusa excepción **`NameError`** cuando el código hace referencia a una variable local antes de que se le haya asignado un valor. Por ejemplo, el siguiente código lanza una excepción en la sentencia `print` tanto en 1.5.2 como en 2.0; en 1.5.2 se lanza una excepción **`NameError`**, mientras que en 2.0 se lanza una nueva excepción **`UnboundLocalError`**. **`UnboundLocalError`** es una subclase de **`NameError`**, así que cualquier código existente que espere que se lance **`NameError`** debería seguir funcionando.

```
def f():
    print "i=",i
    i = i + 1
f()
```

Se han introducido dos nuevas excepciones, **`TabError`** y **`IndentationError`**. Ambas son subclases de **`SyntaxError`**, y se lanzan cuando el código Python se encuentra con una sangría incorrecta.

Cambios en las funciones incorporadas

Se ha añadido un nuevo built-in, `zip(seq1, seq2, ...)`. **`zip()`** retorna una lista de tuplas donde cada tupla contiene el i-ésimo elemento de cada una de las secuencias del argumento. La diferencia entre **`zip()`** y `map(None, seq1, seq2)` es que **`map()`** rellena las secuencias con `None` si las secuencias no tienen la misma longitud, mientras que **`zip()`** trunca la lista retornada a la longitud de la secuencia argumental más corta.

Las funciones **`int()`** y **`long()`** ahora aceptan un parámetro «base» opcional cuando el primer argumento es una cadena. `int('123', 10)` devuelve 123, mientras que `int('123', 16)` devuelve 291. `int(123, 16)` genera una excepción **`TypeError`** con el mensaje «no se puede convertir una cadena con una base explícita».

Se ha añadido al módulo **`sys`** una nueva variable que contiene información más detallada sobre la versión. `sys.version_info` es una tupla `(major, minor, micro, level, serial)` Por ejemplo, en una hipotética 2.0.1beta1, `sys.version_info` sería `(2, 0, 1, 'beta', 1)`. *level* es una cadena como `"alpha"`, `"beta"`, o `"final"` para una versión final.

Los diccionarios tienen un nuevo método extraño, `setdefault(key, default)`, que se comporta de manera similar al método **`get()`** existente. Sin embargo, si falta la clave, **`setdefault()`** devuelve el valor de *default* como lo haría **`get()`** y también lo inserta en el diccionario como el valor de *key*. Por lo tanto, las siguientes líneas de código:

```
if dict.has_key( key ): return dict[key]
else:
    dict[key] = []
    return dict[key]
```

puede reducirse a una única sentencia `return dict.setdefault(key, [])`.

El intérprete establece una profundidad de recursión máxima para atrapar la recursión desbocada antes de llenar la pila de C y causar un volcado del núcleo o GPF. Anteriormente este límite se fijaba cuando se compilaba Python, pero en la versión 2.0 la profundidad máxima de recursión puede leerse y modificarse usando **`sys.getrecursionlimit()`** y **`sys.setrecursionlimit()`**. El valor por defecto es 1000, y se puede encontrar un valor máximo aproximado para una plataforma determinada ejecutando un nuevo script, `Misc/find_recursionlimit.py`.

Adaptación a la versión 2.0

Las nuevas versiones de Python se esfuerzan por ser compatibles con las anteriores, y el historial ha sido bastante bueno. Sin embargo, algunos cambios se consideran lo suficientemente útiles, normalmente porque corrigen decisiones de diseño iniciales que resultaron ser activamente erróneas, que no siempre se puede evitar romper la compatibilidad hacia atrás. Esta sección enumera los cambios en Python 2.0 que pueden hacer que el código Python antiguo se rompa.

The change which will probably break the most code is tightening up the arguments accepted by some methods. Some methods would take multiple arguments and treat them as a tuple, particularly various list methods such as **`append()`** and **`insert()`**. In earlier versions of Python, if `L` is a list, `L.append( 1,2 )` appends the tuple `(1,2)` to the list. In Python 2.0 this causes a **`TypeError`** exception to be raised, with the message: “append requires exactly 1 argument; 2 given”. The fix is to simply add an extra set of parentheses to pass both values as a tuple: `L.append( (1,2) )`.

Las versiones anteriores de estos métodos eran más indulgentes porque utilizaban una función antigua en la interfaz C de Python para analizar sus argumentos; la versión 2.0 los moderniza para utilizar **`PyArg_ParseTuple()`**, la función de análisis de argumentos actual, que proporciona mensajes de error más útiles y trata las llamadas de múltiples argumentos como errores. Si es absolutamente necesario utilizar la versión 2.0 pero no puede corregir su código, puede editar `Objects/listobject.c` y definir el símbolo de preprocesador `NO_STRICT_LIST_APPEND` para preservar el comportamiento anterior; esto no se recomienda.

Algunas de las funciones del módulo **`socket`** siguen siendo indulgentes en este sentido. Por ejemplo, `socket.connect( ('hostname', 25) )` es la forma correcta, ya que pasa una tupla que representa una dirección IP, pero `socket.connect('hostname', 25)` también funciona. **`socket.connect_ex`** y **`socket.bind`** son igualmente fáciles de usar. 2.0alpha1 restringió estas funciones, pero debido a que la documentación en realidad usaba la forma de argumentos múltiples errónea, muchas personas escribieron código que rompería con la verificación más estricta. GvR retiró los cambios ante la reacción del público, por lo que para el módulo **`socket`**, la documentación se corrigió y la forma de argumentos múltiples simplemente se marcó como obsoleta; *will* se ajustará nuevamente en una futura versión de Python.

El escape `\x` en los literales de cadena ahora toma exactamente 2 dígitos hexadecimales. Antes consumía todos los dígitos hexadecimales que seguían a la “x” y tomaba los 8 bits más bajos del resultado, por lo que `\x123456` era equivalente a `\x56`.

Las excepciones **`AttributeError`** y **`NameError`** tienen un mensaje de error más amigable, cuyo texto será algo así como `'Spam' instance has no attribute 'eggs'` o `name 'eggs' is not defined`. Anteriormente, el mensaje de error era simplemente la falta del nombre del atributo `eggs`, y el código escrito para aprovechar este hecho se romperá en la versión 2.0.

Se ha trabajado un poco para que los números enteros y los números enteros largos sean un poco más intercambiables. En 1.5.2, se agregó soporte para archivos grandes para Solaris, para permitir la lectura de archivos mayores a 2 GiB; esto hizo que el método **`tell()`** de objetos de archivo devolviera un número entero largo en lugar de un número entero regular. Algunos códigos restaban dos desplazamientos de archivo e intentaban usar el resultado para multiplicar una secuencia o cortar una cadena, pero esto generaba un **`TypeError`**. En 2.0, los números enteros largos se pueden usar para multiplicar o cortar una secuencia, y se comportará como esperaría intuitivamente; `3L * 'abc'` produce “abcabcabc” y `(0,1,2,3)[2L:4L]` produce (2,3). Los números enteros largos también se pueden usar en varios contextos donde anteriormente solo se aceptaban números enteros, como en el método **`seek()`** de objetos de archivo y en los formatos admitidos por el operador `%` (`%d`, `%i`, `%x`, etc.). Por ejemplo, `"%d" % 2L**64` producirá la cadena `18446744073709551616`.

El cambio más sutil de los enteros largos es que el **`str()`** de un entero largo ya no tiene un carácter “L” al final, aunque **`repr()`** todavía lo incluye. La “L” molestaba a muchas personas que querían imprimir enteros largos con el mismo aspecto que los enteros normales, ya que tenían que esforzarse por cortar el carácter. Esto ya no es un problema en 2.0, pero el código que hace `str(longval)[:-1]` y asume que la “L” está ahí, ahora perderá el dígito final.

Ahora, al tomar el **`repr()`** de un float, se utiliza una precisión de formato diferente a la de **`str()`**. **`repr()`** utiliza la cadena de formato `%.17g` para **`sprintf()`** de C, mientras que **`str()`** utiliza `%.12g` como antes. El efecto es que **`repr()`** puede mostrar ocasionalmente más decimales que **`str()`** para ciertos números. Por ejemplo, el número 8.1 no se puede representar exactamente en binario, por lo que `repr(8.1)` es `'8.0999999999999996'`, mientras que str(8.1) es `'8.1'`.

Se ha eliminado la opción de línea de comandos `-X`, que convertía todas las excepciones estándar en cadenas en lugar de clases; ahora las excepciones estándar siempre serán clases. El módulo **`exceptions`** que contiene las excepciones estándar se tradujo de Python a un módulo C integrado, escrito por Barry Warsaw y Fredrik Lundh.

Extensión/Incorporación de cambios

Algunos de los cambios están bajo la cubierta, y sólo serán evidentes para la gente que escribe módulos de extensión de C o que incrusta un intérprete de Python en una aplicación más grande. Si no estás tratando con la API de C de Python, puedes saltarte esta sección.

El número de versión de la API C de Python se incrementó, por lo que las extensiones C compiladas para 1.5.2 deben ser recompiladas para que funcionen con 2.0. En Windows, no es posible que Python 2.0 importe una extensión de terceros construida para Python 1.5.x debido a cómo funcionan las DLL de Windows, por lo que Python lanzará una excepción y la importación fallará.

Los usuarios del módulo ExtensionClass de Jim Fulton estarán encantados de saber que se han añadido ganchos para que las ExtensionClasses sean ahora compatibles con **`isinstance()`** y **`issubclass()`**. Esto significa que ya no tiene que recordar escribir código como `if type(obj) == myExtensionClass`, sino que puede utilizar el más natural `if isinstance(obj, myExtensionClass)`.

El archivo `Python/importdl.c`, que era una masa de #ifdefs para soportar la carga dinámica en muchas plataformas diferentes, fue limpiado y reorganizado por Greg Stein. `importdl.c` es ahora bastante pequeño, y el código específico de la plataforma se ha movido a un montón de archivos `Python/dynload_*.c`. Otra limpieza: también había una serie de archivos `my*.h` en el directorio Include/ que contenían varios hacks de portabilidad; se han fusionado en un único archivo, `Include/pyport.h`.

La esperada reestructuración de malloc de Vladimir Marangozov se completó para facilitar que el intérprete de Python use un asignador personalizado en lugar del **`malloc()`** estándar de C. Para consultar la documentación, lea los comentarios en `Include/pymem.h` y `Include/objimpl.h`. Para consultar las extensas discusiones durante las cuales se elaboró la interfaz, consulte los archivos web de las listas “patches” y “python-dev” en python.org.

Las versiones recientes del entorno de desarrollo GUSI para MacOS soportan hilos POSIX. Por lo tanto, el soporte de hilos POSIX de Python ahora funciona en Macintosh. También se ha contribuido al soporte de hilos utilizando la biblioteca GNU `pth` del espacio de usuario.

También se ha mejorado el soporte de hilos en Windows. Windows soporta bloqueos de hilos que utilizan objetos del núcleo sólo en caso de contención; en el caso común cuando no hay contención, utilizan funciones más simples que son un orden de magnitud más rápido. Una versión con hilos de Python 1.5.2 en NT es dos veces más lenta que una versión sin hilos; con los cambios de la 2.0, la diferencia es sólo del 10%. Estas mejoras fueron aportadas por Yakov Markovitch.

El código fuente de Python 2.0 ahora sólo utiliza prototipos ANSI C, por lo que la compilación de Python ahora requiere un compilador ANSI C, y ya no puede hacerse utilizando un compilador que sólo soporte K&R C.

Anteriormente, la máquina virtual de Python utilizaba números de 16 bits en su bytecode, lo que limitaba el tamaño de los archivos fuente. En particular, esto afectaba al tamaño máximo de las listas literales y los diccionarios en el código fuente de Python; ocasionalmente, las personas que generan código Python se encontraban con este límite. Un parche de Charles G. Waldman eleva el límite de `2^16` a `2^{32}`.

Se agregaron tres nuevas funciones de conveniencia destinadas a agregar constantes al diccionario de un módulo en el momento de inicialización del módulo: `PyModule_AddObject()`, `PyModule_AddIntConstant()` y `PyModule_AddStringConstant()`. Cada una de estas funciones toma un objeto de módulo, una cadena C terminada en cero que contiene el nombre que se agregará y un tercer argumento para el valor que se asignará al nombre. Este tercer argumento es, respectivamente, un objeto Python, un long C o una cadena C.

Se agregó una API contenedora para los controladores de señales de estilo Unix. `PyOS_getsig()` obtiene un controlador de señales y `PyOS_setsig()` establecerá un nuevo controlador.

Distutils: Facilitando la instalación de módulos

Antes de Python 2.0, la instalación de módulos era un asunto tedioso – no había forma de averiguar automáticamente dónde se instalaba Python, o qué opciones del compilador se debían usar para los módulos de extensión. Los autores de software tenían que pasar por un arduo ritual de edición de Makefiles y archivos de configuración, que sólo funcionaban realmente en Unix y dejaban sin soporte a Windows y MacOS. Los usuarios de Python se enfrentaban a instrucciones de instalación muy diferentes que variaban entre los distintos paquetes de extensión, lo que hacía que la administración de una instalación de Python fuera una tarea ardua.

El SIG para utilidades de distribución, dirigido por Greg Ward, ha creado Distutils, un sistema para facilitar enormemente la instalación de paquetes. Forman el paquete `distutils`, una nueva parte de la biblioteca estándar de Python. En el mejor de los casos, instalar un módulo Python desde el código fuente requerirá los mismos pasos: primero, simplemente hay que descomprimir el archivo tarball o zip y ejecutar «`python setup.py install`». La plataforma se detectará automáticamente, se reconocerá el compilador, se compilarán los módulos de extensión de C y se instalará la distribución en el directorio adecuado. Los argumentos opcionales de la línea de comandos proporcionan más control sobre el proceso de instalación; el paquete distutils ofrece muchos lugares para anular los valores predeterminados, separando la compilación de la instalación, compilando o instalando en directorios que no sean los predeterminados, y más.

Para usar las Distutils, necesitas escribir un script `setup.py`. Para el caso simple, cuando el software contiene sólo archivos .py, un `setup.py` mínimo puede tener sólo unas pocas líneas:

```
from distutils.core import setup
setup (name = "foo", version = "1.0",
       py_modules = ["module1", "module2"])
```

El archivo `setup.py` no es mucho más complicado si el software consta de unos pocos paquetes:

```
from distutils.core import setup
setup (name = "foo", version = "1.0",
        packages = ["package", "package.subpackage"])
```

Una extensión en C puede ser el caso más complicado; he aquí un ejemplo tomado del paquete PyXML:

```
from distutils.core import setup, Extension

expat_extension = Extension('xml.parsers.pyexpat',
     define_macros = [('XML_NS', None)],
     include_dirs = [ 'extensions/expat/xmltok',
                      'extensions/expat/xmlparse' ],
     sources = [ 'extensions/pyexpat.c',
                 'extensions/expat/xmltok/xmltok.c',
                 'extensions/expat/xmltok/xmlrole.c', ]
       )
setup (name = "PyXML", version = "0.5.4",
       ext_modules =[ expat_extension ] )
```

Las Distutils también pueden encargarse de crear distribuciones fuente y binarias. El comando «sdist», ejecutado por «`python setup.py sdist`, construye una distribución fuente como `foo-1.0.tar.gz`. Añadir nuevos comandos no es difícil, ya se han aportado los comandos «bdist_rpm» y «bdist_wininst» para crear una distribución RPM y un instalador de Windows para el software, respectivamente. Los comandos para crear otros formatos de distribución, como los paquetes de Debian y los archivos `.pkg` de Solaris, se encuentran en diversas etapas de desarrollo.

Todo esto está documentado en un nuevo manual, *Distribución de módulos de Python*, que se une al conjunto básico de documentación de Python.

Módulos XML

Python 1.5.2 incluía un analizador XML simple en forma del módulo **`xmllib`**, aportado por Sjoerd Mullender. Desde el lanzamiento de la versión 1.5.2, se han vuelto comunes dos interfaces diferentes para procesar XML: SAX2 (versión 2 de la API simple para XML) proporciona una interfaz basada en eventos con algunas similitudes con **`xmllib`**, y DOM (Document Object Model) proporciona una interfaz basada en árboles, transformando un documento XML en un árbol de nodos que se puede recorrer y modificar. Python 2.0 incluye una interfaz SAX2 y una interfaz DOM simplificada como parte del paquete **`xml`**. Aquí daremos una breve descripción general de estas nuevas interfaces; consulte la documentación de Python o el código fuente para obtener detalles completos. El SIG de XML de Python también está trabajando en una documentación mejorada.

Soporte de SAX2

SAX define una interfaz controlada por eventos para analizar XML. Para utilizar SAX, debe escribir una clase controladora de SAX. Las clases controladoras heredan de varias clases proporcionadas por SAX y reemplazan varios métodos que luego llamará el analizador XML. Por ejemplo, los métodos **`startElement()`** y **`endElement()`** se llaman para cada etiqueta de inicio y fin que encuentre el analizador, el método **`characters()`** se llama para cada fragmento de datos de caracteres, y así sucesivamente.

La ventaja del enfoque basado en eventos es que todo el documento no tiene que residir en la memoria en un momento dado, lo cual es importante si estás procesando documentos realmente enormes. Sin embargo, escribir la clase manejadora de SAX puede ser muy complicado si se intenta modificar la estructura del documento de alguna manera elaborada.

Por ejemplo, este pequeño programa de ejemplo define un manejador que imprime un mensaje para cada etiqueta inicial y final, y luego analiza el archivo `hamlet.xml` usándolo:

```
from xml import sax

class SimpleHandler(sax.ContentHandler):
    def startElement(self, name, attrs):
        print 'Start of element:', name, attrs.keys()

    def endElement(self, name):
        print 'End of element:', name
```

```
# Create a parser object
parser = sax.make_parser()

# Tell it what handler to use
handler = SimpleHandler()
parser.setContentHandler( handler )

# Parse a file!
parser.parse( 'hamlet.xml' )
```

Para más información, consulte la documentación de Python o el XML HOWTO en https://pyxml.sourceforge.net/topics/howto/xml-howto.html.

Soporte DOM

El modelo de objetos de documento es una representación basada en árboles para un documento XML. Una instancia **`Document`** de nivel superior es la raíz del árbol y tiene un único elemento secundario que es la instancia **`Element`** de nivel superior. Este **`Element`** tiene nodos secundarios que representan datos de caracteres y cualquier subelemento, que puede tener otros elementos secundarios propios, y así sucesivamente. Con el DOM, puede recorrer el árbol resultante de la forma que desee, acceder a los valores de los elementos y atributos, insertar y eliminar nodos y convertir el árbol nuevamente en XML.

El DOM es útil para modificar documentos XML, porque se puede crear un árbol DOM, modificarlo añadiendo nuevos nodos o reordenando subárboles, y luego producir un nuevo documento XML como salida. También se puede construir un árbol DOM manualmente y convertirlo en XML, lo que puede ser una forma más flexible de producir una salida XML que simplemente escribir `<tag1>...</tag1>` un archivo.

La implementación del DOM incluida con Python se encuentra en el módulo **`xml.dom.minidom`**. Es una implementación liviana del DOM de nivel 1 con soporte para espacios de nombres XML. Las funciones de conveniencia **`parse()`** y **`parseString()`** se proporcionan para generar un árbol DOM:

```
from xml.dom import minidom
doc = minidom.parse('hamlet.xml')
```

`doc` es una instancia de **`Document`**. **`Document`**, al igual que todas las demás clases DOM como **`Element`** y **`Text`**, es una subclase de la clase base **`Node`**. Por lo tanto, todos los nodos de un árbol DOM admiten ciertos métodos comunes, como **`toxml()`**, que devuelve una cadena que contiene la representación XML del nodo y sus hijos. Cada clase también tiene sus propios métodos especiales; por ejemplo, las instancias **`Element`** y **`Document`** tienen un método para encontrar todos los elementos secundarios con un nombre de etiqueta determinado. Continuando con el ejemplo anterior de 2 líneas:

```
perslist = doc.getElementsByTagName( 'PERSONA' )
print perslist[0].toxml()
print perslist[1].toxml()
```

Para el archivo XML *Hamlet*, las líneas anteriores dan como resultado:

```
<PERSONA>CLAUDIO, rey de Dinamarca. </PERSONA>
<PERSONA>HAMLET, hijo del difunto rey y sobrino del actu
```

El elemento raíz del documento está disponible como `doc.documentElement`, y sus hijos pueden modificarse fácilmente borrando, añadiendo o eliminando nodos:

```
root = doc.documentElement

# Eliminar el primer hijo
root.removeChild( root.childNodes[0] )

# Mover el nuevo primer hijo al final
root.appendChild( root.childNodes[0] )

# Insertar el nuevo primer hijo (originalmente,
# el tercer hijo) antes del hijo número 20.
root.insertBefore( root.childNodes[0], root.childNodes[2
```

Una vez más, te remito a la documentación de Python para obtener una lista completa de las diferentes clases `Node` y sus diversos métodos.

Relación con PyXML

El Grupo de Interés Especial XML lleva un tiempo trabajando en código Python relacionado con XML. Su distribución de código, llamada PyXML, está disponible en las páginas web del SIG en https://www.python.org/community/sigs/current/xml-sig. La distribución de PyXML también utiliza el nombre de paquete `xml`. Si has escrito programas que utilizan PyXML, probablemente te preguntes sobre su compatibilidad con el paquete 2.0 `xml`.

La respuesta es que el paquete `xml` de Python 2.0 no es compatible con PyXML, pero puede hacerse compatible instalando una versión reciente de PyXML. Muchas aplicaciones pueden arreglárselas con el soporte XML que se incluye en Python 2.0, pero las aplicaciones más complicadas requerirán que se instale el paquete PyXML completo. Cuando se instala, las versiones 0.6.0 o superiores de PyXML sustituyen al paquete `xml` que se entrega con Python, y son un estricto superconjunto del paquete estándar, añadiendo un montón de características adicionales. Algunas de las características adicionales de PyXML incluyen:

- 4DOM, una implementación completa de DOM de FourThought, Inc.
- El parser de validación xmlproc, escrito por Lars Marius Garshol.
- El módulo acelerador del parser `sgmlop`, escrito por Fredrik Lundh.

Cambios en los módulos

Se realizaron muchas mejoras y correcciones de errores en la extensa biblioteca estándar de Python; algunos de los módulos afectados incluyen `readline`, `ConfigParser`, `cgi`, `calendar`, `posix`, `readline`, `xmllib`, `aifc`, `chunk`, `wave`, `random`, `shelve` y `nntplib`. Consulte los registros CVS para obtener detalles exactos de cada parche.

Brian Gallew contribuyó con la compatibilidad con OpenSSL para el módulo `socket`. OpenSSL es una implementación de la capa de sockets seguros, que cifra los datos que se envían a través de un socket. Al compilar Python, puede editar `Modules/Setup` para incluir compatibilidad con SSL, lo que agrega una función adicional al módulo **`socket`**: `socket.ssl(socket, keyfile, certfile)`, que toma un objeto de socket y devuelve un socket SSL. Los módulos `httplib` y `urllib` también se modificaron para admitir las URL de `https://`, aunque nadie ha implementado FTP o SMTP sobre SSL.

Greg Stein ha reescrito el módulo `httplib` para admitir HTTP/1.1.

Se proporciona compatibilidad con versiones anteriores de la versión 1.5 de **`httplib`**, aunque el uso de funciones HTTP/1.1 como la canalización requerirá reescribir el código para usar un conjunto diferente de interfaces.

El módulo **`Tkinter`** ahora es compatible con las versiones 8.1, 8.2 u 8.3 de Tcl/Tk y se ha eliminado la compatibilidad con las versiones 7.x anteriores. El módulo Tkinter ahora es compatible con la visualización de cadenas Unicode en widgets Tk. Además, Fredrik Lundh contribuyó con una optimización que hace que operaciones como `create_line` y `create_polygon` sean mucho más rápidas, especialmente cuando se utilizan muchas coordenadas.

El módulo `curses` ha sido ampliado en gran medida, a partir de la versión mejorada de Oliver Andrich, para proporcionar muchas funciones adicionales de los curses ncurses y SYSV, como el color, el soporte de conjuntos de caracteres alternativos, los pads y el soporte de ratón. Esto significa que el módulo ya no es compatible con los sistemas operativos que sólo tienen curses BSD, pero no parece haber ningún sistema operativo actualmente mantenido que caiga en esta categoría.

Como se mencionó en la discusión anterior sobre el soporte Unicode de la 2.0, la implementación subyacente de las expresiones regulares proporcionadas por el módulo `re` ha sido cambiada. SRE, un nuevo motor de expresiones regulares escrito por Fredrik Lundh y parcialmente financiado por Hewlett Packard, soporta la comparación con cadenas de 8 bits y cadenas Unicode.

Nuevos módulos

Se han añadido varios módulos nuevos. Nos limitaremos a enumerarlos con breves descripciones; consulte la documentación de la versión 2.0 para conocer los detalles de un módulo concreto.

- **`atexit`**: Para registrar las funciones que serán llamadas antes de que el intérprete de Python salga. El código que actualmente establece `sys.exitfunc` directamente debe cambiarse para usar el módulo **`atexit`** en su lugar, importando **`atexit`** y llamando a **`atexit.register()`** con la función a llamar al salir. (Contribución de Skip Montanaro)
- **`codecs`**, **`encodings`**, **`unicodedata`**: Añadidos como parte del nuevo soporte de Unicode.
- **`filecmp`**: Sustituye a los antiguos módulos **`cmp`**, **`cmpcache`** y **`dircmp`**, que han quedado obsoletos. (Contribución de Gordon MacMillan y Moshe Zadka)
- **`gettext`**: Este módulo proporciona soporte de internacionalización (I18N) y localización (L10N) para los programas de Python, proporcionando una interfaz a la biblioteca de catálogo de mensajes GNU gettext. (Integrado por Barry Warsaw, a partir de contribuciones separadas de Martin von Löwis, Peter Funk y James Henstridge)
- **`linuxaudiodev`**: Soporte para el dispositivo `/dev/audio` en Linux, un gemelo del módulo existente **`sunaudiodev`**. (Contribuido por Peter Bosch, con correcciones de Jeremy Hylton)
- **`mmap`**: Una interfaz para archivos mapeados en memoria tanto en Windows como en Unix. El contenido de un fichero puede ser mapeado directamente en memoria, en cuyo momento se comporta como una cadena mutable, por lo que su contenido puede ser leído y modificado. Incluso pueden pasarse a funciones que esperan cadenas ordinarias, como el módulo **`re`**. (Contribución de Sam Rushing, con algunas extensiones de A.M. Kuchling)
- **`pyexpat`**: Una interfaz para el analizador XML de Expat. (Contribuido por Paul Prescod.)

- **`robotparser`**: Analiza un archivo `robots.txt`, que se utiliza para escribir arañas web que evitan amablemente ciertas áreas de un sitio web. El analizador acepta el contenido de un archivo `robots.txt`, construye un conjunto de reglas a partir de él y puede responder a preguntas sobre la capacidad de búsqueda de una URL determinada. (Contribución de Skip Montanaro)
- **`tabnanny`**: Un módulo/script para comprobar el código fuente de Python en busca de sangrías ambiguas. (Contribuido por Tim Peters.)
- **`UserString`**: Una clase base útil para derivar objetos que se comportan como cadenas.
- **`webbrowser`**: Un módulo que proporciona una forma independiente de la plataforma para lanzar un navegador web en una URL específica. Para cada plataforma, se prueban varios navegadores en un orden específico. El usuario puede modificar el navegador que se lanza estableciendo la variable de entorno *BROWSER*. (Originalmente inspirado por el parche de Eric S. Raymond a **`urllib`** que añadía una funcionalidad similar, pero el módulo final proviene de un código originalmente implementado por Fred Drake como `Tools/idle/BrowserControl.py`, y adaptado para la biblioteca estándar por Fred)
- **`_winreg`**: Una interfaz para el registro de Windows. **`_winreg`** es una adaptación de las funciones que han formado parte de PythonWin desde 1995, pero ahora se ha añadido a la distribución principal, y se ha mejorado para soportar Unicode. **`_winreg`** fue escrito por Bill Tutt y Mark Hammond.
- **`zipfile`**: Un módulo para leer y escribir archivos con formato ZIP. Se trata de archivos producidos por **PKZIP** en DOS/Windows o **zip** en Unix, que no deben confundirse con los archivos con formato **gzip** (que son compatibles con el módulo **`gzip`**) (Contribución de James C. Ahlstrom.)
- **`imputil`**: Un módulo que proporciona una forma más sencilla de escribir ganchos de importación personalizados, en comparación con el módulo **`ihooks`** existente. (Implementado por Greg Stein, con mucha discusión en python-dev a lo largo del camino)

Mejoras en IDLE

IDLE es el IDE oficial de Python multiplataforma, escrito con Tkinter. Python 2.0 incluye IDLE 0.6, que añade una serie de nuevas características y mejoras. Una lista parcial:

- Mejoras y optimizaciones de la interfaz de usuario, especialmente en el área de resaltado de sintaxis y auto-indentación.
- El navegador de clases muestra ahora más información, como las funciones de nivel superior de un módulo.
- El ancho del tabulador es ahora una opción configurable por el usuario. Al abrir un archivo Python existente, IDLE detecta automáticamente las convenciones de sangría y se adapta.
- Ahora hay soporte para llamar a los navegadores en varias plataformas, utilizado para abrir la documentación de Python en un navegador.
- IDLE ahora tiene una línea de comandos, que es en gran medida similar al intérprete de Python vainilla.
- Se añadieron consejos de llamada en muchos lugares.
- Ahora IDLE puede instalarse como un paquete.
- En la ventana del editor, ahora hay una barra de líneas/columnas en la parte inferior.
- Tres nuevos comandos de teclado: Comprobar módulo (`Alt-F5`), Importar módulo (`F5`) y Ejecutar script (`Ctrl-F5`).

Módulos eliminados y obsoletos

Se han eliminado algunos módulos porque son obsoletos, o porque ahora hay mejores formas de hacer lo mismo. El módulo **`stdwin`** ha desaparecido; era para un conjunto de herramientas de ventanas independientes de la plataforma que ya no se desarrolla.

Se han movido varios módulos al subdirectorio `lib-old`: **`cmp`**, **`cmpcache`**, **`dircmp`**, **`dump`**, **`find`**, **`grep`**, **`packmail`**, **`poly`**, **`util`**, **`whatsound`**, **`zmod`**. Si tiene código que depende de un módulo que se ha movido a `lib-old`, puede simplemente agregar ese directorio a `sys.path` para recuperarlo, pero se le recomienda que actualice cualquier código que use estos módulos.

Agradecimientos

Los autores desean agradecer a las siguientes personas sus sugerencias sobre varios borradores de este artículo: David Bolen, Mark Hammond, Gregg Hauser, Jeremy Hylton, Fredrik Lundh, Detlef Lannert, Aahz Maruch, Skip Montanaro, Vladimir Marangozov, Tobias Polzin, Guido van Rossum, Neil Schemenauer y Russ Schmidt.

Registro de cambios

Python next

Release date: XXXX-XX-XX

Windows

- gh-146458 [https://github.com/python/cpython/issues/146458]: Fix incorrect REPL height and width tracking on console window resize on Windows.

Security

- gh-149017 [https://github.com/python/cpython/issues/149017]: Update bundled libexpat [https://libexpat.github.io/] to version 2.8.0.
- gh-90309 [https://github.com/python/cpython/issues/90309]: Base64-encode values when embedding cookies to JavaScript using the `http.cookies.BaseCookie.js_output()` method to avoid injection and escaping.
- gh-148808 [https://github.com/python/cpython/issues/148808]: Added buffer boundary check when using `nbytes` parameter with `asyncio.AbstractEventLoop.sock_recvfrom_into()`. Only relevant for Windows and the `asyncio.ProactorEventLoop`.
- gh-148395 [https://github.com/python/cpython/issues/148395]: Fix a dangling input pointer in `lzma.LZMADecompressor`, `bz2.BZ2Decompressor`, and internal `zlib._ZlibDecompressor` when memory allocation fails with `MemoryError`, which could let a subsequent `decompress()` call read or write through a stale pointer to the already-released caller buffer.
- gh-148169 [https://github.com/python/cpython/issues/148169]: A bypass in `webbrowser` allowed URLs prefixed with

`%action` to pass the dash-prefix safety check.

- gh-146581 [https://github.com/python/cpython/issues/146581]: Fix vulnerability in **`shutil.unpack_archive()`** for ZIP files on Windows which allowed to write files outside of the destination tree if the patch in the archive contains a Windows drive prefix. Now such invalid paths will be skipped. Files containing «..» in the name (like «foo..bar») are no longer skipped.
- gh-146333 [https://github.com/python/cpython/issues/146333]: Fix quadratic backtracking in **`configparser.RawConfigParser`** option parsing regexes (`OPTCRE` and `OPTCRE_NV`). A crafted configuration line with many whitespace characters could cause excessive CPU usage.
- gh-146211 [https://github.com/python/cpython/issues/146211]: Reject CR/LF characters in tunnel request headers for the HTTPConnection.set_tunnel() method.

Library

- gh-149148 [https://github.com/python/cpython/issues/149148]: **`ensurepip`**: Upgrade bundled pip to 26.1. This version fixes the **CVE 2026-3219** [https://www.cve.org/CVERecord?id=CVE-2026-3219] vulnerability. Patch by Victor Stinner.
- gh-148947 [https://github.com/python/cpython/issues/148947]: Fix crash in **`@dataclasses.dataclass`** with `slots=True` that occurred when a function found within the class had an empty `__class__` cell.
- gh-148680 [https://github.com/python/cpython/issues/148680]: `ForwardRef` objects that contain internal names to represent known objects now show the `type_repr` of the known object rather than the internal `__annotationlib_name_x__` name when evaluated as strings.
- gh-148801 [https://github.com/python/cpython/issues/148801]: **`xml.etree.ElementTree`**: Fix a crash in **`Element.__deepcopy__`** on deeply nested trees.
- gh-148735 [https://github.com/python/cpython/issues/148735]: **`xml.etree.ElementTree`**: Fix a use-after-free in **`Element.findtext`** when the element tree is mutated concurrently during the search.

- gh-148740 [https://github.com/python/cpython/issues/148740]: Fix usage for `uuid` command-line interface to support a custom namespace be provided for uuid3 and uuid5.
- gh-148651 [https://github.com/python/cpython/issues/148651]: Fix reference leak in `compression.zstd.ZstdDecompressor` when an invalid option key is passed.
- gh-146553 [https://github.com/python/cpython/issues/146553]: Fix infinite loop in `typing.get_type_hints()` when `__wrapped__` forms a cycle. Patch by Shamil Abdulaev.
- gh-148508 [https://github.com/python/cpython/issues/148508]: An intermittent timing error when running SSL tests on iOS has been resolved.
- gh-148518 [https://github.com/python/cpython/issues/148518]: If an email containing an address header that ended in an open double quote was parsed with a non-`compat32` policy, accessing the `username` attribute of the mailbox accessed through that header object would result in an `IndexError`. It now correctly returns an empty string as the result.
- gh-148464 [https://github.com/python/cpython/issues/148464]: Add missing `__ctype_le/be__` attributes for `c_float_complex` and `c_double_complex`. Patch by Sergey B Kirpichev.
- gh-148370 [https://github.com/python/cpython/issues/148370]: `configparser`: prevent quadratic behavior when a `ParsingError` is raised after a parser fails to parse multiple lines. Patch by Bénédikt Tran.
- gh-148254 [https://github.com/python/cpython/issues/148254]: Use singular «sec» instead of «secs» in `timeit` verbose output for consistency with other time units.
- gh-148192 [https://github.com/python/cpython/issues/148192]: `email.generator.Generator._make_boundary` could fail to detect a duplicate boundary string if linesep was not n. It now correctly detects boundary strings when linesep is rn as well.
- gh-146313 [https://github.com/python/cpython/issues/146313]: Fix a deadlock in `multiprocessing`'s resource tracker where the parent process could hang indefinitely in `os.waitpid()` during interpreter shutdown if a child created via `os.fork()` still held the resource tracker's pipe open.

- gh-145831 [https://github.com/python/cpython/issues/145831]: Fix **`email.quoprimime.decode()`** leaving a stray `\r` when `eol='\r\n'` by stripping the full *eol* string instead of one character.
- gh-145105 [https://github.com/python/cpython/issues/145105]: Fix crash in **`csv`** reader when iterating with a re-entrant iterator that calls **`next()`** on the same reader from within `__next__`.
- gh-105936 [https://github.com/python/cpython/issues/105936]: Attempting to mutate non-field attributes of **`dataclasses`** with both *frozen* and *slots* being `True` now raises **`FrozenInstanceError`** instead of **`TypeError`**. Their non-dataclass subclasses can now freely mutate non-field attributes, and the original non-slotted class can be garbage collected.
- gh-140287 [https://github.com/python/cpython/issues/140287]: The **`asyncio`** REPL now handles exceptions when executing **`PYTHONSTARTUP`** scripts. Patch by Bartosz Sławecki.
- gh-132631 [https://github.com/python/cpython/issues/132631]: Fix «I/O operation on closed file» when parsing JSON Lines file with **`JSON CLI`**.
- gh-70039 [https://github.com/python/cpython/issues/70039]: Fixed bug where **`smtplib.SMTP.starttls()`** could fail if **`smtplib.SMTP.connect()`** is called explicitly rather than implicitly.
- gh-83281 [https://github.com/python/cpython/issues/83281]: **`email`**: improve handling trailing garbage in address lists to avoid throwing AttributeError in certain edge cases

Documentation

- gh-148663 [https://github.com/python/cpython/issues/148663]: Document that **`calendar.IllegalMonthError`** is a subclass of both **`ValueError`** and **`IndexError`** since Python 3.12.
- gh-146646 [https://github.com/python/cpython/issues/146646]: Document that **`glob.glob()`**, **`glob.iglob()`**, **`pathlib.Path.glob()`**, and **`pathlib.Path.rglob()`** silently suppress **`OSError`** exceptions raised from scanning the filesystem.

Core and Builtins

- gh-149122 [https://github.com/python/cpython/issues/149122]: Fix a crash in optimized calls to **`all()`**, **`any()`**, **`tuple()`**, **`list()`**, and **`set()`** with an async generator expression argument (for example, `tuple(await x for x in y)`). These calls now correctly raise `TypeError` instead of crashing.
- gh-113956 [https://github.com/python/cpython/issues/113956]: Fix a data race in **`sys.intern()`** in the free-threaded build when interning a string owned by another thread. An interned copy owned by the current thread is used instead when it is not safe to immortalize the original.
- gh-148820 [https://github.com/python/cpython/issues/148820]: Fix a race in **`_PyRawMutex`** on the free-threaded build where a `Py_PARK_INTR` return from `_PySemaphore_Wait` could let the waiter destroy its semaphore before the unlocking thread's `_PySemaphore_Wakeup` completed, causing a fatal `ReleaseSemaphore` error.
- gh-148653 [https://github.com/python/cpython/issues/148653]: Forbid **`marshalling`** recursive code objects and **`slice`** objects which cannot be correctly unmarshalled.
- gh-142516 [https://github.com/python/cpython/issues/142516]: Forward-port the generational cycle garbage collector to the default 3.14 build, replacing the incremental collector while leaving the free-threaded collector unchanged.
- gh-148393 [https://github.com/python/cpython/issues/148393]: Fix data races between **`PyDict_Watch()`** / **`PyDict_Unwatch()`** and concurrent dict mutation in the free-threaded build.
- gh-148284 [https://github.com/python/cpython/issues/148284]: Fix high stack consumption in Python's interpreter loop on Clang 22 by setting function limits for inlining when building with computed gotos.
- gh-148037 [https://github.com/python/cpython/issues/148037]: Remove critical section from **`PyCode_Addr2Line()`** in free-threading.
- gh-148222 [https://github.com/python/cpython/issues/148222]: Fix vectorcall support in **`types.GenericAlias`** when the underlying type does not support the vectorcall protocol. Fix

possible leaks in **`types.GenericAlias`** and **`types.UnionType`** in case of memory error.
- gh-148208 [https://github.com/python/cpython/issues/148208]: Fix recursion depth leak in **`PyObject_Print()`**
- gh-137814 [https://github.com/python/cpython/issues/137814]: Fix the `__qualname__` attribute of `__annotate__` functions on functions.
- gh-146455 [https://github.com/python/cpython/issues/146455]: Fix $O(N^2)$ compile-time regression in constant folding after it was moved from AST to CFG optimizer.

Build

- gh-148535 [https://github.com/python/cpython/issues/148535]: No longer use the `gcc -fprofile-update=atomic` flag on i686. The flag has been added to fix a random GCC internal error on PGO build (gh-145801 [https://github.com/python/cpython/issues/145801]) caused by corruption of profile data (.gcda files). The problem is that it makes the PGO build way slower (up to 47x slower) on i686. Since the GCC internal error was not seen on i686 so far, don't use `-fprofile-update=atomic` on i686 anymore. Patch by Victor Stinner.
- gh-146264 [https://github.com/python/cpython/issues/146264]: Fix static module builds on non-WASI targets by linking HACL dependencies as static libraries when `MODULE_BUILDTYPE=static`, preventing duplicate `_Py_LibHacl_*` symbol errors at link time.

Python 3.14.4 final

Release date: 2026-04-07

Security

- gh-145986 [https://github.com/python/cpython/issues/145986]: **`xml.parsers.expat`**: Fixed a crash caused by unbounded C recursion when converting deeply nested XML content models with **`ElementDeclHandler()`**. This addresses **CVE 2026-4224** [https://www.cve.org/CVERecord?id=CVE-2026-4224].

- gh-145599 [https://github.com/python/cpython/issues/145599]: Reject control characters in **`http.cookies.Morsel`** **`update()`** and **`js_output()`**. This addresses **CVE 2026-3644** [https://www.cve.org/CVERecord?id=CVE-2026-3644].
- gh-145506 [https://github.com/python/cpython/issues/145506]: Fixes **CVE 2026-2297** [https://www.cve.org/CVERecord?id=CVE-2026-2297] by ensuring that `SourcelessFileLoader` uses **`io.open_code()`** when opening `.pyc` files.
- gh-144370 [https://github.com/python/cpython/issues/144370]: Disallow usage of control characters in status in **`wsgiref.handlers`** to prevent HTTP header injections. Patch by Benedikt Johannes.
- gh-143930 [https://github.com/python/cpython/issues/143930]: Reject leading dashes in URLs passed to **`webbrowser.open()`**.

Core and Builtins

- gh-148157 [https://github.com/python/cpython/issues/148157]: Fix an unlikely crash when parsing an invalid type comments for function parameters. Found by OSS Fuzz in #492782951 [https://issues.oss-fuzz.com/issues/492782951].

- gh-148144 [https://github.com/python/cpython/issues/148144]: Initialize `_PyInterpreterFrame.visited` when copying interpreter frames so incremental GC does not read an uninitialized byte from generator and frame-object copies.

- gh-146615 [https://github.com/python/cpython/issues/146615]: Fix a crash in **`__get__()`** for METH_METHOD descriptors when an invalid (non-type) object is passed as the second argument. Patch by Steven Sun.

- gh-146308 [https://github.com/python/cpython/issues/146308]: Fixed several error handling issues in the **`_remote_debugging`** module, including safer validation of remote `int` objects, clearer asyncio task chain failures, and cache cleanup fixes that avoid leaking or double-freeing metadata on allocation failure. Patch by Pablo Galindo.

- gh-146128 [https://github.com/python/cpython/issues/146128]: Fix a bug which could cause constant values to be partially corrupted in AArch64 JIT code. This issue is theoretical, and hasn't actually been observed in unmodified Python interpreters.

- gh-146250 [https://github.com/python/cpython/issues/146250]: Fixed a memory leak in **`SyntaxError`** when re-initializing it.
- gh-146245 [https://github.com/python/cpython/issues/146245]: Fixed reference leaks in **`socket`** when audit hooks raise exceptions in **`socket.getaddrinfo()`** and **`socket.sendto()`**.
- gh-146196 [https://github.com/python/cpython/issues/146196]: Fix potential Undefined Behavior in **`PyUnicodeWriter_WriteASCII()`** by adding a zero-length check. Patch by Shamil Abdulaev.
- gh-146227 [https://github.com/python/cpython/issues/146227]: Fix wrong type in `_Py_atomic_load_uint16` in the C11 atomics backend (`pyatomic_std.h`), which used a 32-bit atomic load instead of 16-bit. Found by Mohammed Zuhaib.
- gh-146056 [https://github.com/python/cpython/issues/146056]: Fix **`repr()`** for lists and tuples containing `NULL`s.
- gh-146092 [https://github.com/python/cpython/issues/146092]: Handle properly memory allocation failures on str and float opcodes. Patch by Victor Stinner.
- gh-146041 [https://github.com/python/cpython/issues/146041]: Fix free-threading scaling bottleneck in **`sys.intern()`** and **`PyObject_SetAttr()`** by avoiding the interpreter-wide lock when the string is already interned and immortalized.
- gh-145990 [https://github.com/python/cpython/issues/145990]: `python --help-env` sections are now sorted by environment variable name.
- gh-145990 [https://github.com/python/cpython/issues/145990]: `python --help-xoptions` is now sorted by `-X` option name.
- gh-145376 [https://github.com/python/cpython/issues/145376]: Fix

GC tracking in `structseq.__replace__()`.

- gh-145792 [https://github.com/python/cpython/issues/145792]: Fix out-of-bounds access when invoking faulthandler on a CPython build compiled without support for VLAs.

- gh-142183 [https://github.com/python/cpython/issues/142183]: Avoid a pathological case where repeated calls at a specific stack depth could be significantly slower.

- gh-145779 [https://github.com/python/cpython/issues/145779]: Improve scaling of **`classmethod()`** and **`staticmethod()`** calls in the free-threaded build by avoiding the descriptor `__get__` call.

- gh-145783 [https://github.com/python/cpython/issues/145783]: Fix an unlikely crash in the parser when certain errors were erroneously not propagated. Found by OSS Fuzz in #491369109 [https://issues.oss-fuzz.com/issues/491369109].

- gh-145685 [https://github.com/python/cpython/issues/145685]: Improve scaling of type attribute lookups in the free-threaded build by avoiding contention on the internal type lock.

- gh-145701 [https://github.com/python/cpython/issues/145701]: Fix **`SystemError`** when `__classdict__` or `__conditional_annotations__` is in a class-scope inlined comprehension. Found by OSS Fuzz in #491105000 [https://issues.oss-fuzz.com/issues/491105000].

- gh-145713 [https://github.com/python/cpython/issues/145713]: Make **`bytearray.resize()`** thread-safe in the free-threaded build by using a critical section and calling the lock-held variant of the resize function.

- gh-145615 [https://github.com/python/cpython/issues/145615]: Fixed a memory leak in the free-threaded build where mimalloc pages could become permanently unreclaimable until the owning thread exited.

- gh-145566 [https://github.com/python/cpython/issues/145566]: In the free threading build, skip the stop-the-world pause when reassigning `__class__` on a newly created object.

- gh-145335 [https://github.com/python/cpython/issues/145335]: Fix a crash in `os.pathconf()` when called with `-1` as the path argument.
- gh-145036 [https://github.com/python/cpython/issues/145036]: In free-threaded build, fix race condition when calling `__sizeof__()` on a `list`
- gh-145376 [https://github.com/python/cpython/issues/145376]: Fix reference leaks in various unusual error scenarios.
- gh-145234 [https://github.com/python/cpython/issues/145234]: Fixed a `SystemError` in the parser when an encoding cookie (for example, UTF-7) decodes to carriage returns (`\r`). Newlines are now normalized after decoding in the string tokenizer.

  Patch by Pablo Galindo.
- gh-130555 [https://github.com/python/cpython/issues/130555]: Fix use-after-free in `dict.clear()` when the dictionary values are embedded in an object and a destructor causes re-entrant mutation of the dictionary.
- gh-145187 [https://github.com/python/cpython/issues/145187]: Fix compiler assertion fail when a type parameter bound contains an invalid expression in a conditional block.
- gh-145142 [https://github.com/python/cpython/issues/145142]: Fix a crash in the free-threaded build when the dictionary argument to `str.maketrans()` is concurrently modified.
- gh-144872 [https://github.com/python/cpython/issues/144872]: Fix heap buffer overflow in the parser found by OSS-Fuzz.
- gh-144766 [https://github.com/python/cpython/issues/144766]: Fix a crash in fork child process when perf support is enabled.
- gh-144759 [https://github.com/python/cpython/issues/144759]: Fix undefined behavior in the lexer when `start` and

`multi_line_start` pointers are `NULL` in `_PyLexer_remember_fstring_buffers()` and `_PyLexer_restore_fstring_buffers()`. The `NULL` pointer arithmetic (`NULL - valid_pointer`) is now guarded with explicit `NULL` checks.

- gh-144563 [https://github.com/python/cpython/issues/144563]: Fix interaction of the Tachyon profiler and **`ctypes`** and other modules that load the Python shared library (if present) in an independent map as this was causing the mechanism that loads the binary information to be confused. Patch by Pablo Galindo

- gh-144601 [https://github.com/python/cpython/issues/144601]: Fix crash when importing a module whose `PyInit` function raises an exception from a subinterpreter.

- gh-144438 [https://github.com/python/cpython/issues/144438]: Align the QSBR thread state array to a 64-byte cache line boundary to avoid false sharing in the free-threaded build.

- gh-144513 [https://github.com/python/cpython/issues/144513]: Fix potential deadlock when using critical sections during stop-the-world pauses in the free-threaded build.

- gh-144446 [https://github.com/python/cpython/issues/144446]: Fix data races in the free-threaded build when reading frame object attributes while another thread is executing the frame.

- gh-143636 [https://github.com/python/cpython/issues/143636]: Fix a crash when calling **`SimpleNamespace.__replace__()`** on non-namespace instances. Patch by Bénédikt Tran.

- gh-143650 [https://github.com/python/cpython/issues/143650]: Fix race condition in **`importlib`** where a thread could receive a stale module reference when another thread's import fails.

- gh-141732 [https://github.com/python/cpython/issues/141732]: Ensure the **`__repr__()`** for **`ExceptionGroup`** and **`BaseExceptionGroup`** does not change when the exception sequence that was original passed in to its constructor is subsequently mutated.

- gh-140594 [https://github.com/python/cpython/issues/140594]: Fix an out of bounds read when a single NUL character is read from the standard input. Patch by Shamil Abdulaev.

- gh-91636 [https://github.com/python/cpython/issues/91636]: While performing garbage collection, clear weakrefs to unreachable objects that are created during running of finalizers. If those weakrefs were are not cleared, they could reveal unreachable objects.

- gh-130327 [https://github.com/python/cpython/issues/130327]: Fix erroneous clearing of an object's **`__dict__`** if overwritten at runtime.

- gh-80667 [https://github.com/python/cpython/issues/80667]: Literals using the `\N{name}` escape syntax can now construct CJK ideographs and Hangul syllables using case-insensitive names.

Library

- gh-144503 [https://github.com/python/cpython/issues/144503]: Fix a regression introduced in 3.14.3 and 3.13.12 where the **`multiprocessing`** `forkserver` start method would fail with **`BrokenPipeError`** when the parent process had a very large **`sys.argv`**. The argv is now passed to the forkserver as separate command-line arguments rather than being embedded in the `-c` command string, avoiding the operating system's per-argument length limit.
- gh-146613 [https://github.com/python/cpython/issues/146613]: **`itertools`**: Fix a crash in **`itertools.groupby()`** when the grouper iterator is concurrently mutated.
- gh-146080 [https://github.com/python/cpython/issues/146080]: **`ssl`**: fix a crash when an SNI callback tries to use an SSL object that has already been garbage-collected. Patch by Bénédikt Tran.
- gh-146556 [https://github.com/python/cpython/issues/146556]: Fix **`annotationlib.get_annotations()`** hanging indefinitely when called with `eval_str=True` on a callable that has a circular `__wrapped__` chain (e.g.

`f.__wrapped__ = f`). Cycle detection using an id-based visited set now stops the traversal and falls back to the globals found so far, mirroring the approach of **`inspect.unwrap()`**.

- gh-146090 [https://github.com/python/cpython/issues/146090]: **`sqlite3`**: fix a crash when **`sqlite3.Connection.create_collation()`** fails with SQLITE_BUSY [https://sqlite.org/rescode.html#busy]. Patch by Bénédikt Tran.
- gh-146090 [https://github.com/python/cpython/issues/146090]: **`sqlite3`**: properly raise **`MemoryError`** instead of **`SystemError`** when a context callback fails to be allocated. Patch by Bénédikt Tran.
- gh-145633 [https://github.com/python/cpython/issues/145633]: Fix `struct.pack('f', float)`: use **`PyFloat_Pack4()`** to raise **`OverflowError`**. Patch by Sergey B Kirpichev and Victor Stinner.
- gh-146310 [https://github.com/python/cpython/issues/146310]: The **`ensurepip`** module no longer looks for `pip-*.whl` wheel packages in the current directory.
- gh-146083 [https://github.com/python/cpython/issues/146083]: Update bundled libexpat [https://libexpat.github.io/] to version 2.7.5.
- gh-146076 [https://github.com/python/cpython/issues/146076]: **`zoneinfo`**: fix crashes when deleting `_weak_cache` from a **`zoneinfo.ZoneInfo`** subclass.
- gh-146054 [https://github.com/python/cpython/issues/146054]: Limit the size of **`encodings.search_function()`** cache. Found by OSS Fuzz in #493449985 [https://issues.oss-fuzz.com/issues/493449985].
- gh-146004 [https://github.com/python/cpython/issues/146004]: All **`-X`** options from the Python command line are now propagated to child processes spawned by **`multiprocessing`**, not just a hard-coded subset. This makes the behavior consistent between default «spawn» and «forkserver» start methods and the old «fork» start method. The options that were previously not propagated are: `context_aware_warnings`, `cpu_count`, `disable-remote-debug`, `int_max_str_digits`, `lazy_imports`, `no_debug_ranges`, `pathconfig_warnings`, `perf`,

`perf_jit`, `presite`, `pycache_prefix`, `thread_inherit_context`, and `warn_default_encoding`.

- gh-145883 [https://github.com/python/cpython/issues/145883]: **`zoneinfo`**: Fix heap buffer overflow reads from malformed TZif data. Found by OSS Fuzz, issues #492245058 [https://issues.oss-fuzz.com/issues/492245058] and #492230068 [https://issues.oss-fuzz.com/issues/492230068].
- gh-145754 [https://github.com/python/cpython/issues/145754]: Request signature during mock autospec with `FORWARDREF` annotation format. This prevents runtime errors when an annotation uses a name that is not defined at runtime.
- gh-145750 [https://github.com/python/cpython/issues/145750]: Avoid undefined behaviour from signed integer overflow when parsing format strings in the **`struct`** module. Found by OSS Fuzz in #488466741 [https://issues.oss-fuzz.com/issues/488466741].
- gh-145492 [https://github.com/python/cpython/issues/145492]: Fix infinite recursion in **`collections.defaultdict`** `__repr__` when a `defaultdict` contains itself. Based on analysis by KowalskiThomas in gh-145492 [https://github.com/python/cpython/issues/145492].
- gh-145623 [https://github.com/python/cpython/issues/145623]: Fix crash in **`struct`** when calling **`repr()`** or `__sizeof__()` on an uninitialized **`struct.Struct`** object created via `Struct.__new__()` without calling `__init__()`.
- gh-145616 [https://github.com/python/cpython/issues/145616]: Detect Android sysconfig ABI correctly on 32-bit ARM Android on 64-bit ARM kernel
- gh-145551 [https://github.com/python/cpython/issues/145551]: Fix InvalidStateError when cancelling process created by **`asyncio.create_subprocess_exec()`** or **`asyncio.create_subprocess_shell()`**. Patch by Daan De Meyer.
- gh-145446 [https://github.com/python/cpython/issues/145446]: Now **`functools`** is safer in free-threaded build when using keywords in **`functools.partial()`**
- gh-145417 [https://github.com/python/cpython/issues/145417]: **`venv`**: Prevent incorrect preservation of SELinux context when copying the `Activate.ps1` script. The script inherited the SELinux security context of the system template

directory, rather than the destination project directory.

- gh-145376 [https://github.com/python/cpython/issues/145376]: Fix double free and null pointer dereference in unusual error scenarios in `hashlib` and `hmac` modules.
- gh-145301 [https://github.com/python/cpython/issues/145301]: `hmac`: fix a crash when the initialization of the underlying C extension module fails.
- gh-145301 [https://github.com/python/cpython/issues/145301]: `hashlib`: fix a crash when the initialization of the underlying C extension module fails.
- gh-145264 [https://github.com/python/cpython/issues/145264]: Base64 decoder (see `binascii.a2b_base64()`, `base64.b64decode()`, etc) no longer ignores excess data after the first padded quad in non-strict (default) mode. Instead, in conformance with RFC 4648 [https://datatracker.ietf.org/doc/html/rfc4648.html], section 3.3, it now ignores the pad character, «=», if it is present before the end of the encoded data.
- gh-145158 [https://github.com/python/cpython/issues/145158]: Avoid undefined behaviour from signed integer overflow when parsing format strings in the `struct` module.
- gh-144984 [https://github.com/python/cpython/issues/144984]: Fix crash in `xml.parsers.expat.xmlparser.ExternalEntityParserCre` when an allocation fails. The error paths could dereference NULL `handlers` and double-decrement the parent parser's reference count.
- gh-88091 [https://github.com/python/cpython/issues/88091]: Fix `unicodedata.decomposition()` for Hangul characters.
- gh-144986 [https://github.com/python/cpython/issues/144986]: Fix a memory leak in `atexit.register()`. Patch by Shamil Abdulaev.
- gh-144777 [https://github.com/python/cpython/issues/144777]: Fix data races in `io.IncrementalNewlineDecoder` in the free-threaded build.
- gh-144809 [https://github.com/python/cpython/issues/144809]: Make `collections.deque` copy atomic in the free-threaded build.
- gh-144835 [https://github.com/python/cpython/issues/144835]: Added missing explanations for some parameters in `glob.glob()`

and `glob.iglob()`.

- gh-144833 [https://github.com/python/cpython/issues/144833]: Fixed a use-after-free in `ssl` when `SSL_new()` returns NULL in `newPySSLSocket()`. The error was reported via a dangling pointer after the object had already been freed.
- gh-144782 [https://github.com/python/cpython/issues/144782]: Fix `argparse.ArgumentParser` to be `pickleable`.
- gh-144259 [https://github.com/python/cpython/issues/144259]: Fix inconsistent display of long multiline pasted content in the REPL.
- gh-144156 [https://github.com/python/cpython/issues/144156]: Fix the folding of headers by the `email` library when **RFC 2047** [https://datatracker.ietf.org/doc/html/rfc2047.html] encoded words are used. Now whitespace is correctly preserved and also correctly added between adjacent encoded words. The latter property was broken by the fix for gh-92081 [https://github.com/python/cpython/issues/92081], which mostly fixed previous failures to preserve whitespace.
- gh-66305 [https://github.com/python/cpython/issues/66305]: Fixed a hang on Windows in the `tempfile` module when trying to create a temporary file or subdirectory in a non-writable directory.
- gh-140814 [https://github.com/python/cpython/issues/140814]: `multiprocessing.freeze_support()` no longer sets the default start method as a side effect, which previously caused a subsequent `multiprocessing.set_start_method()` call to raise `RuntimeError`.
- gh-144475 [https://github.com/python/cpython/issues/144475]: Calling `repr()` on `functools.partial()` is now safer when the partial object's internal attributes are replaced while the string representation is being generated.
- gh-144538 [https://github.com/python/cpython/issues/144538]: Bump the version of pip bundled in ensurepip to version 26.0.1
- gh-144494 [https://github.com/python/cpython/issues/144494]: Fix performance regression in `asyncio.all_tasks()` on free-threaded builds. Patch by Kumar Aditya.
- gh-144316 [https://github.com/python/cpython/issues/144316]: Fix crash in `_remote_debugging` that caused `test_external_inspection` to intermittently fail. Patch by Taegyun Kim.

- gh-144363 [https://github.com/python/cpython/issues/144363]: Update bundled libexpat [https://libexpat.github.io/] to 2.7.4
- gh-143637 [https://github.com/python/cpython/issues/143637]: Fixed a crash in socket.sendmsg() that could occur if ancillary data is mutated re-entrantly during argument parsing.
- gh-143543 [https://github.com/python/cpython/issues/143543]: Fix a crash in itertools.groupby that could occur when a user-defined `__eq__()` method re-enters the iterator during key comparison.
- gh-140652 [https://github.com/python/cpython/issues/140652]: Fix a crash in `_interpchannels.list_all()` after closing a channel.
- gh-143698 [https://github.com/python/cpython/issues/143698]: Allow *scheduler* and *setpgroup* arguments to be explicitly `None` when calling `os.posix_spawn()` or `os.posix_spawnp()`. Patch by Bénédikt Tran.
- gh-143698 [https://github.com/python/cpython/issues/143698]: Raise `TypeError` instead of `SystemError` when the *scheduler* in `os.posix_spawn()` or `os.posix_spawnp()` is not a tuple. Patch by Bénédikt Tran.
- gh-142516 [https://github.com/python/cpython/issues/142516]: `ssl`: fix reference leaks in `ssl.SSLContext` objects. Patch by Bénédikt Tran.
- gh-143304 [https://github.com/python/cpython/issues/143304]: Fix `ctypes.CDLL` to honor the `handle` parameter on POSIX systems.
- gh-142781 [https://github.com/python/cpython/issues/142781]: `zoneinfo`: fix a crash when instantiating `ZoneInfo` objects for which the internal class-level cache is inconsistent.
- gh-142763 [https://github.com/python/cpython/issues/142763]: Fix a race condition between `zoneinfo.ZoneInfo` creation and `zoneinfo.ZoneInfo.clear_cache()` that could raise `KeyError`.
- gh-142787 [https://github.com/python/cpython/issues/142787]: Fix assertion failure in `sqlite3` blob subscript when slicing with indices that result in an empty slice.
- gh-142352 [https://github.com/python/cpython/issues/142352]: Fix `asyncio.StreamWriter.start_tls()` to transfer buffered data from `StreamReader` to the SSL layer, preventing data loss when upgrading a connection to TLS

mid-stream (e.g., when implementing PROXY protocol support).

- gh-141707 [https://github.com/python/cpython/issues/141707]: Don't change **`tarfile.TarInfo`** type from `AREGTYPE` to `DIRTYPE` when parsing GNU long name or link headers.
- gh-139933 [https://github.com/python/cpython/issues/139933]: Improve **`AttributeError`** suggestions for classes with a custom **`__dir__()`** method returning a list of unsortable values. Patch by Bénédikt Tran.
- gh-137335 [https://github.com/python/cpython/issues/137335]: Get rid of any possibility of a name conflict for named pipes in **`multiprocessing`** and **`asyncio`** on Windows, no matter how small.
- gh-80667 [https://github.com/python/cpython/issues/80667]: Support lookup for Tangut Ideographs in **`unicodedata`**.
- bpo-40243 [https://bugs.python.org/issue?@action=redirect&bpo=40243]: Fix **`unicodedata.ucd_3_2_0.numeric()`** for non-decimal values.

Documentation

- gh-126676 [https://github.com/python/cpython/issues/126676]: Expand **`argparse`** documentation for `type=bool` with a demonstration of the surprising behavior and pointers to common alternatives.
- gh-145649 [https://github.com/python/cpython/issues/145649]: Fix text wrapping and formatting of `-X` option descriptions in the *python(1) [https://manpages.debian.org/python(1)]* man page by using proper roff markup.
- gh-145450 [https://github.com/python/cpython/issues/145450]: Document missing public **`wave.Wave_write`** getter methods.
- gh-136246 [https://github.com/python/cpython/issues/136246]: A new «Improve this page» link is available in the left-hand sidebar of the docs, offering links to create GitHub issues, discussion forum posts, or pull requests.

Tests

- gh-144418 [https://github.com/python/cpython/issues/144418]: The Android testbed's emulator RAM has been increased from 2 GB to 4 GB.
- gh-146202 [https://github.com/python/cpython/issues/146202]: Fix a race condition in regrtest: make sure that the temporary directory is created in the worker process. Previously, temp_cwd() could fail on Windows if the «build» directory was not created. Patch by Victor Stinner.
- gh-144739 [https://github.com/python/cpython/issues/144739]: When Python was compiled with system expat older then 2.7.2 but tests run with newer expat, still skip **`test.test_pyexpat.MemoryProtectionTest`**.

Build

- gh-146541 [https://github.com/python/cpython/issues/146541]: The Android testbed can now be built for 32-bit ARM and x86 targets.
- gh-146498 [https://github.com/python/cpython/issues/146498]: The iOS XCframework build script now ensures libpython isn't included in installed app content, and is more robust in identifying standard library binary content that requires processing.
- gh-146450 [https://github.com/python/cpython/issues/146450]: The Android build script was modified to improve parity with other platform build scripts.
- gh-146446 [https://github.com/python/cpython/issues/146446]: The clean target for the Apple/iOS XCframework build script is now more selective when targeting a single architecture.
- gh-145801 [https://github.com/python/cpython/issues/145801]: When Python build is optimized with GCC using PGO, use `-fprofile-update=atomic` option to use atomic operations when updating profile information. This option reduces the risk of gcov Data Files (.gcda) corruption which can cause random GCC crashes. Patch by Victor Stinner.

Windows

- gh-145307 [https://github.com/python/cpython/issues/145307]: Defers loading of the `psapi.dll` module until it is used by

`ctypes.util.dllist()`.

- gh-144551 [https://github.com/python/cpython/issues/144551]: Updated bundled version of OpenSSL to 3.0.19.
- gh-140131 [https://github.com/python/cpython/issues/140131]: Fix REPL cursor position on Windows when module completion suggestion line hits console width.

macOS

- gh-144551 [https://github.com/python/cpython/issues/144551]: Update macOS installer to use OpenSSL 3.0.19.
- gh-137586 [https://github.com/python/cpython/issues/137586]: Invoke **osascript** with absolute path in **`webbrowser`** and **`turtledemo`**.

C API

- gh-146056 [https://github.com/python/cpython/issues/146056]: **`PyUnicodeWriter_WriteRepr()`** now supports `NULL` argument.
- gh-145010 [https://github.com/python/cpython/issues/145010]: Use GCC dialect alternatives for inline assembly in `object.h` so that the Python headers compile correctly with `-masm=intel`.
- gh-144981 [https://github.com/python/cpython/issues/144981]: Made **`PyUnstable_Code_SetExtra()`**, **`PyUnstable_Code_GetExtra()`**, and **`PyUnstable_Eval_RequestCodeExtraIndex()`** thread-safe on the free threaded build.

Python 3.14.3 final

Release date: 2026-02-03

Windows

- gh-128067 [https://github.com/python/cpython/issues/128067]: Fix a bug in PyREPL on Windows where output without a trailing newline was overwritten by the next prompt.

Tools/Demos

- gh-142095 [https://github.com/python/cpython/issues/142095]: Make gdb "py-bt" command use frame from thread local state when available. Patch by Sam Gross and Victor Stinner.

Tests

- gh-144415 [https://github.com/python/cpython/issues/144415]: The Android testbed now distinguishes between stdout/stderr messages which were triggered by a newline, and those triggered by a manual call to `flush`. This fixes logging of progress indicators and similar content.
- gh-143460 [https://github.com/python/cpython/issues/143460]: Skip tests relying on infinite recusion if stack size is unlimited.
- gh-65784 [https://github.com/python/cpython/issues/65784]: Add support for parametrized resource `wantobjects` in regrtests, which allows to run Tkinter tests with the specified value of **`tkinter.wantobjects`**, for example `-u wantobjects=0`.
- gh-143553 [https://github.com/python/cpython/issues/143553]: Add support for parametrized resources, such as `-u xpickle=2.7`.
- gh-142836 [https://github.com/python/cpython/issues/142836]: Accommodated Solaris in `test_pdb.test_script_target_anonymous_pipe`.
- bpo-31391 [https://bugs.python.org/issue?@action=redirect&bpo=31391]: Forward-port test_xpickle from Python 2 to Python 3 and add the resource back to test's command line.

Security

- gh-144125 [https://github.com/python/cpython/issues/144125]: **`BytesGenerator`** will now refuse to serialize (write) headers that are unsafely folded or delimited; see **`verify_generated_headers`**. (Contributed by Bas Bloemsaat and Petr Viktorin in gh-121650 [https://github.com/python/cpython/issues/121650]).
- gh-143935 [https://github.com/python/cpython/issues/143935]: Fixed

a bug in the folding of comments when flattening an email message using a modern email policy. Comments consisting of a very long sequence of non-foldable characters could trigger a forced line wrap that omitted the required leading space on the continuation line, causing the remainder of the comment to be interpreted as a new header field. This enabled header injection with carefully crafted inputs.

- gh-143925 [https://github.com/python/cpython/issues/143925]: Reject control characters in `data:` URL media types.
- gh-143919 [https://github.com/python/cpython/issues/143919]: Reject control characters in **`http.cookies.Morsel`** fields and values.
- gh-143916 [https://github.com/python/cpython/issues/143916]: Reject C0 control characters within wsgiref.headers.Headers fields, values, and parameters.

Library

- gh-144380 [https://github.com/python/cpython/issues/144380]: Improve performance of **`io.BufferedReader`** line iteration by ~49%.
- gh-144169 [https://github.com/python/cpython/issues/144169]: Fix three crashes when non-string keyword arguments are supplied to objects in the **`ast`** module.
- gh-144100 [https://github.com/python/cpython/issues/144100]: Fixed a crash in ctypes when using a deprecated `POINTER(str)` type in `argtypes`. Instead of aborting, ctypes now raises a proper Python exception when the pointer target type is unresolved.
- gh-144050 [https://github.com/python/cpython/issues/144050]: Fix **`stat.filemode()`** in the pure-Python implementation to avoid misclassifying invalid mode values as block devices.
- gh-144023 [https://github.com/python/cpython/issues/144023]: Fixed validation of file descriptor 0 in posix functions when used with follow_symlinks parameter.
- gh-143999 [https://github.com/python/cpython/issues/143999]: Fix an issue where **`inspect.getgeneratorstate()`** and **`inspect.getcoroutinestate()`** could fail for generators wrapped by **`types.coroutine()`** in the suspended state.
- gh-143831 [https://github.com/python/cpython/issues/143831]:

`annotationlib.ForwardRef` objects are now hashable when created from annotation scopes with closures. Previously, hashing such objects would throw an exception. Patch by Bartosz Sławecki.

- gh-143874 [https://github.com/python/cpython/issues/143874]: Fixed a bug in `pdb` where expression results were not sent back to remote client.
- gh-143880 [https://github.com/python/cpython/issues/143880]: Fix data race in `functools.partial()` in the free threading build.
- gh-143706 [https://github.com/python/cpython/issues/143706]: Fix `multiprocessing` forkserver so that `sys.argv` is correctly set before `__main__` is preloaded. Previously, `sys.argv` was empty during main module import in forkserver child processes. This fixes a regression introduced in 3.13.8 and 3.14.1. Root caused by Aaron Wieczorek, test provided by Thomas Watson, thanks!
- gh-143638 [https://github.com/python/cpython/issues/143638]: Forbid reentrant calls of the `pickle.Pickler` and `pickle.Unpickler` methods for the C implementation. Previously, this could cause crash or data corruption, now concurrent calls of methods of the same object raise `RuntimeError`.
- gh-78724 [https://github.com/python/cpython/issues/78724]: Raise `RuntimeError`'s when user attempts to call methods on half-initialized `Struct` objects, For example, created by `Struct.__new__(Struct)`. Patch by Sergey B Kirpichev.
- gh-143196 [https://github.com/python/cpython/issues/143196]: Fix crash when the internal encoder object returned by undocumented function `json.encoder.c_make_encoder()` was called with non-zero second (*_current_indent_level*) argument.
- gh-143191 [https://github.com/python/cpython/issues/143191]: `_thread.stack_size()` now raises `ValueError` if the stack size is too small. Patch by Victor Stinner.
- gh-143602 [https://github.com/python/cpython/issues/143602]: Fix a inconsistency issue in `write()` that leads to unexpected buffer overwrite by deduplicating the buffer exports.
- gh-143547 [https://github.com/python/cpython/issues/143547]: Fix `sys.unraisablehook()` when the hook raises an exception

and changes `sys.unraisablehook()`: hold a strong reference to the old hook. Patch by Victor Stinner.

- gh-143517 [https://github.com/python/cpython/issues/143517]: `annotationlib.get_annotations()` no longer raises a `SyntaxError` when evaluating a stringified starred annotation that starts with one or more whitespace characters followed by a `*`. Patch by Bartosz Sławecki.
- gh-143378 [https://github.com/python/cpython/issues/143378]: Fix use-after-free crashes when a `BytesIO` object is concurrently mutated during `write()` or `writelines()`.
- gh-143346 [https://github.com/python/cpython/issues/143346]: Fix incorrect wrapping of the Base64 data in `plistlib._PlistWriter` when the indent contains a mix of tabs and spaces.
- gh-143310 [https://github.com/python/cpython/issues/143310]: `tkinter`: fix a crash when a Python `list` is mutated during the conversion to a Tcl object (e.g., when setting a Tcl variable). Patch by Bénédikt Tran.
- gh-143309 [https://github.com/python/cpython/issues/143309]: Fix a crash in `os.execve()` on non-Windows platforms when given a custom environment mapping which is then mutated during parsing. Patch by Bénédikt Tran.
- gh-143308 [https://github.com/python/cpython/issues/143308]: `pickle`: fix use-after-free crashes when a `PickleBuffer` is concurrently mutated by a custom buffer callback during pickling. Patch by Bénédikt Tran and Aaron Wieczorek.
- gh-143237 [https://github.com/python/cpython/issues/143237]: Fix support of named pipes in the rotating `logging` handlers.
- gh-143249 [https://github.com/python/cpython/issues/143249]: Fix possible buffer leaks in Windows overlapped I/O on error handling.
- gh-143241 [https://github.com/python/cpython/issues/143241]: `zoneinfo`: fix infinite loop in `ZoneInfo.from_file` when parsing a malformed TZif file. Patch by Fatih Celik.
- gh-142830 [https://github.com/python/cpython/issues/142830]: `sqlite3`: fix use-after-free crashes when the connection's callbacks are mutated during a callback execution. Patch by Bénédikt Tran.
- gh-143200 [https://github.com/python/cpython/issues/143200]: `xml.etree.ElementTree`: fix use-after-free crashes in

`__getitem__()` and `__setitem__()` methods of `Element` when the element is concurrently mutated. Patch by Bénédikt Tran.

- gh-142195 [https://github.com/python/cpython/issues/142195]: Updated timeout evaluation logic in `subprocess` to be compatible with deterministic environments like Shadow where time moves exactly as requested.
- gh-142164 [https://github.com/python/cpython/issues/142164]: Fix the ctypes bitfield overflow error message to report the correct offset and size calculation.
- gh-143145 [https://github.com/python/cpython/issues/143145]: Fixed a possible reference leak in ctypes when constructing results with multiple output parameters on error.
- gh-122431 [https://github.com/python/cpython/issues/122431]: Corrected the error message in `readline.append_history_file()` to state that `nelements` must be non-negative instead of positive.
- gh-143004 [https://github.com/python/cpython/issues/143004]: Fix a potential use-after-free in `collections.Counter.update()` when user code mutates the Counter during an update.
- gh-143046 [https://github.com/python/cpython/issues/143046]: The `asyncio` REPL no longer prints copyright and version messages in the quiet mode (`-q`). Patch by Bartosz Sławecki.
- gh-140648 [https://github.com/python/cpython/issues/140648]: The `asyncio` REPL now respects the `-I` flag (isolated mode). Previously, it would load and execute `PYTHONSTARTUP` even if the flag was set. Contributed by Bartosz Sławecki.
- gh-142991 [https://github.com/python/cpython/issues/142991]: Fixed socket operations such as recvfrom() and sendto() for FreeBSD divert(4) socket.
- gh-143010 [https://github.com/python/cpython/issues/143010]: Fixed a bug in `mailbox` where the precise timing of an external event could result in the library opening an existing file instead of a file it expected to create.
- gh-142881 [https://github.com/python/cpython/issues/142881]: Fix concurrent and reentrant call of `atexit.unregister()`.
- gh-112127 [https://github.com/python/cpython/issues/112127]: Fix possible use-after-free in `atexit.unregister()` when the callback is unregistered during comparison.

- gh-142783 [https://github.com/python/cpython/issues/142783]: Fix zoneinfo use-after-free with descriptor _weak_cache. a descriptor as _weak_cache could cause crashes during object creation. The fix ensures proper reference counting for descriptor-provided objects.
- gh-142754 [https://github.com/python/cpython/issues/142754]: Add the *ownerDocument* attribute to **`xml.dom.minidom`** elements and attributes created by directly instantiating the `Element` or `Attr` class. Note that this way of creating nodes is not supported; creator functions like **`xml.dom.Document.documentElement()`** should be used instead.
- gh-142784 [https://github.com/python/cpython/issues/142784]: The **`asyncio`** REPL now properly closes the loop upon the end of interactive session. Previously, it could cause surprising warnings. Contributed by Bartosz Sławecki.
- gh-142555 [https://github.com/python/cpython/issues/142555]: **`array`**: fix a crash in `a[i] = v` when converting *i* to an index via **`i.__index__`** or **`i.__float__`** mutates the array.
- gh-142594 [https://github.com/python/cpython/issues/142594]: Fix crash in `TextIOWrapper.close()` when the underlying buffer's `closed` property calls **`detach()`**.
- gh-142451 [https://github.com/python/cpython/issues/142451]: **`hmac`**: Ensure that the **`HMAC.block_size`** attribute is correctly copied by **`HMAC.copy`**. Patch by Bénédikt Tran.
- gh-142495 [https://github.com/python/cpython/issues/142495]: **`collections.defaultdict`** now prioritizes **`__setitem__()`** when inserting default values from `default_factory`. This prevents race conditions where a default value would overwrite a value set before `default_factory` returns.
- gh-142651 [https://github.com/python/cpython/issues/142651]: **`unittest.mock`**: fix a thread safety issue where **`Mock.call_count`** may return inaccurate values when the mock is called concurrently from multiple threads.
- gh-142595 [https://github.com/python/cpython/issues/142595]: Added type check during initialization of the **`decimal`** module to prevent a crash in case of broken stdlib. Patch by Sergey B Kirpichev.

- gh-142556 [https://github.com/python/cpython/issues/142556]: Fix crash when a task gets re-registered during finalization in **`asyncio`**. Patch by Kumar Aditya.
- gh-123241 [https://github.com/python/cpython/issues/123241]: Avoid reference count operations in garbage collection of **`ctypes`** objects.
- gh-142517 [https://github.com/python/cpython/issues/142517]: The non-`compat32` **`email`** policies now correctly handle refolding encoded words that contain bytes that can not be decoded in their specified character set. Previously this resulted in an encoding exception during folding.
- gh-112527 [https://github.com/python/cpython/issues/112527]: The help text for required options in **`argparse`** no longer extended with « (default: None)».
- gh-142346 [https://github.com/python/cpython/issues/142346]: Fix usage formatting for mutually exclusive groups in **`argparse`** when they are preceded by positional arguments or followed or intermixed with other optional arguments.
- gh-142315 [https://github.com/python/cpython/issues/142315]: Pdb can now run scripts from anonymous pipes used in process substitution. Patch by Bartosz Sławecki.
- gh-142332 [https://github.com/python/cpython/issues/142332]: Fix usage formatting for positional arguments in mutually exclusive groups in **`argparse`**. in **`argparse`**.
- gh-142282 [https://github.com/python/cpython/issues/142282]: Fix **`winreg.QueryValueEx()`** to not accidentally read garbage buffer under race condition.
- gh-75949 [https://github.com/python/cpython/issues/75949]: Fix **`argparse`** to preserve `|` separators in mutually exclusive groups when the usage line wraps due to length.
- gh-142267 [https://github.com/python/cpython/issues/142267]: Improve **`argparse`** performance by caching the formatter used for argument validation.
- gh-68552 [https://github.com/python/cpython/issues/68552]: `MisplacedEnvelopeHeaderDefect` and `Missing header name` defects are now correctly passed to the `handle_defect` method of `policy` in **`FeedParser`**.
- gh-142006 [https://github.com/python/cpython/issues/142006]: Fix a bug in the **`email.policy.default`** folding algorithm which incorrectly resulted in a doubled newline when a line

ending at exactly max_line_length was followed by an unfoldable token.

- gh-105836 [https://github.com/python/cpython/issues/105836]: Fix **`asyncio.run_coroutine_threadsafe()`** leaving underlying cancelled asyncio task running.
- gh-139971 [https://github.com/python/cpython/issues/139971]: **`pydoc`**: Ensure that the link to the online documentation of a stdlib module is correct.
- gh-139262 [https://github.com/python/cpython/issues/139262]: Some keystrokes can be swallowed in the new `PyREPL` on Windows, especially when used together with the ALT key. Fix by Chris Eibl.
- gh-138897 [https://github.com/python/cpython/issues/138897]: Improved **`license`**/**`copyright`**/**`credits`** display in the REPL: now uses a pager.
- gh-79986 [https://github.com/python/cpython/issues/79986]: Add parsing for `References` and `In-Reply-To` headers to the **`email`** library that parses the header content as lists of message id tokens. This prevents them from being folded incorrectly.
- gh-136282 [https://github.com/python/cpython/issues/136282]: Add support for **`UNNAMED_SECTION`** when creating a section via the mapping protocol access
- gh-109263 [https://github.com/python/cpython/issues/109263]: Starting a process from spawn context in **`multiprocessing`** no longer sets the start method globally.
- gh-133253 [https://github.com/python/cpython/issues/133253]: Fix thread-safety issues in **`linecache`**.
- gh-132715 [https://github.com/python/cpython/issues/132715]: Skip writing objects during marshalling once a failure has occurred.

IDLE

- gh-143774 [https://github.com/python/cpython/issues/143774]: Better explain the operation of Format / Format Paragraph.

Documentation

- gh-140806 [https://github.com/python/cpython/issues/140806]: Add

documentation for `enum.bin()`.

Core and Builtins

- gh-144307 [https://github.com/python/cpython/issues/144307]: Prevent a reference leak in module teardown at interpreter finalization.
- gh-144194 [https://github.com/python/cpython/issues/144194]: Fix error handling in perf jitdump initialization on memory allocation failure.
- gh-144012 [https://github.com/python/cpython/issues/144012]: Check if the result is `NULL` in `BINARY_OP_EXTENT` opcode.
- gh-141805 [https://github.com/python/cpython/issues/141805]: Fix crash in `set` when objects with the same hash are concurrently added to the set after removing an element with the same hash while the set still contains elements with the same hash.
- gh-143670 [https://github.com/python/cpython/issues/143670]: Fixes a crash in `ga_repr_items_list` function.
- gh-143377 [https://github.com/python/cpython/issues/143377]: Fix a crash in `_interpreters.capture_exception()` when the exception is incorrectly formatted. Patch by Bénédikt Tran.
- gh-136924 [https://github.com/python/cpython/issues/136924]: The interactive help mode in the REPL no longer incorrectly syntax highlights text input as Python code. Contributed by Olga Matoula.
- gh-143189 [https://github.com/python/cpython/issues/143189]: Fix crash when inserting a non-`str` key into a split table dictionary when the key matches an existing key in the split table but has no corresponding value in the dict.
- gh-143228 [https://github.com/python/cpython/issues/143228]: Fix use-after-free in perf trampoline when toggling profiling while threads are running or during interpreter finalization with daemon threads active. The fix uses reference counting to ensure trampolines are not freed while any code object could still reference them. Pach by Pablo Galindo
- gh-142664 [https://github.com/python/cpython/issues/142664]: Fix a use-after-free crash in `memoryview.__hash__` when the `__hash__` method of the referenced object mutates that

object or the view. Patch by Bénédikt Tran.

- gh-142557 [https://github.com/python/cpython/issues/142557]: Fix a use-after-free crash in bytearray.__mod__ when the **`bytearray`** is mutated while formatting the `%`-style arguments. Patch by Bénédikt Tran.
- gh-143195 [https://github.com/python/cpython/issues/143195]: Fix use-after-free crashes in **`bytearray.hex()`** and **`memoryview.hex()`** when the separator's **`__len__()`** mutates the original object. Patch by Bénédikt Tran.
- gh-142975 [https://github.com/python/cpython/issues/142975]: Fix crash after unfreezing all objects tracked by the garbage collector on the free threaded build.
- gh-143135 [https://github.com/python/cpython/issues/143135]: Set **`sys.flags.inspect`** to `1` when **`PYTHONINSPECT`** is `0`. Previously, it was set to `0` in this case.
- gh-143003 [https://github.com/python/cpython/issues/143003]: Fix an overflow of the shared empty buffer in **`bytearray.extend()`** when `__length_hint__()` returns 0 for non-empty iterator.
- gh-143006 [https://github.com/python/cpython/issues/143006]: Fix a possible assertion error when comparing negative non-integer `float` and `int` with the same number of bits in the integer part.
- gh-143057 [https://github.com/python/cpython/issues/143057]: Avoid locking in **`PyTraceMalloc_Track()`** and **`PyTraceMalloc_Untrack()`** when **`tracemalloc`** is not enabled.
- gh-142776 [https://github.com/python/cpython/issues/142776]: Fix a file descriptor leak in import.c
- gh-142829 [https://github.com/python/cpython/issues/142829]: Fix a use-after-free crash in **`contextvars.Context`** comparison when a custom `__eq__` method modifies the context via **`set()`**.
- gh-142766 [https://github.com/python/cpython/issues/142766]: Clear the frame of a generator when **`generator.close()`** is called.
- gh-142737 [https://github.com/python/cpython/issues/142737]: Tracebacks will be displayed in fallback mode even if **`io.open()`** is lost. Previously, this would crash the interpreter. Patch by Bartosz Sławecki.

- gh-142554 [https://github.com/python/cpython/issues/142554]: Fix a crash in `divmod()` when `_pylong.int_divmod()` does not return a tuple of length two exactly. Patch by Bénédikt Tran.
- gh-142560 [https://github.com/python/cpython/issues/142560]: Fix use-after-free in `bytearray` search-like methods (`find()`, `count()`, `index()`, `rindex()`, and `rfind()`) by marking the storage as exported which causes reallocation attempts to raise `BufferError`. For `contains()`, `split()`, and `rsplit()` the buffer protocol is used for this.
- gh-142531 [https://github.com/python/cpython/issues/142531]: Fix a free-threaded GC performance regression. If there are many untracked tuples, the GC will run too often, resulting in poor performance. The fix is to include untracked tuples in the «long lived» object count. The number of frozen objects is also now included since the free-threaded GC must scan those too.
- gh-142402 [https://github.com/python/cpython/issues/142402]: Fix reference counting when adjacent literal parts are merged while constructing `string.templatelib.Template`, preventing the displaced string object from leaking.
- gh-133932 [https://github.com/python/cpython/issues/133932]: Fix crash in the free threading build when clearing frames that hold tagged integers.
- gh-142343 [https://github.com/python/cpython/issues/142343]: Fix SIGILL crash on m68k due to incorrect assembly constraint.
- gh-100964 [https://github.com/python/cpython/issues/100964]: Fix reference cycle in exhausted generator frames. Patch by Savannah Ostrowski.
- gh-69605 [https://github.com/python/cpython/issues/69605]: Fix edge-cases around already imported modules in the REPL auto-completion of imports.
- gh-138568 [https://github.com/python/cpython/issues/138568]: Adjusted the built-in `help()` function so that empty inputs are ignored in interactive mode.
- gh-137007 [https://github.com/python/cpython/issues/137007]: Fix a bug during JIT compilation failure which caused garbage collection debug assertions to fail.

C API

- gh-142589 [https://github.com/python/cpython/issues/142589]: Fix **`PyUnstable_Object_IsUniqueReferencedTemporary()`** handling of tagged ints on the interpreter stack.
- gh-142571 [https://github.com/python/cpython/issues/142571]: **`PyUnstable_CopyPerfMapFile()`** now checks that opening the file succeeded before flushing.

Build

- gh-142454 [https://github.com/python/cpython/issues/142454]: When calculating the digest of the JIT stencils input, sort the hashed files by filenames before adding their content to the hasher. This ensures deterministic hash input and hence deterministic hash, independent on filesystem order.
- gh-141808 [https://github.com/python/cpython/issues/141808]: When running `make clean-retain-profile`, keep the generated JIT stencils. That way, the stencils are not generated twice when Profile-guided optimization (PGO) is used. It also allows distributors to supply their own pre-built JIT stencils.
- gh-138061 [https://github.com/python/cpython/issues/138061]: Ensure reproducible builds by making JIT stencil header generation deterministic.

Python 3.14.2 final

Release date: 2025-12-05

Security

- gh-142145 [https://github.com/python/cpython/issues/142145]: Remove quadratic behavior in `xml.minidom` node ID cache clearing.
- gh-119452 [https://github.com/python/cpython/issues/119452]: Fix a potential memory denial of service in the **`http.server`** module. When a malicious user is connected to the CGI server on Windows, it could cause an arbitrary amount of memory to be allocated. This could have led to symptoms including a **`MemoryError`**, swapping, out of memory (OOM) killed

processes or containers, or even system crashes.

Library

- gh-140797 [https://github.com/python/cpython/issues/140797]: Revert changes to the undocumented **`re.Scanner`** class. Capturing groups are still allowed for backward compatibility, although using them can lead to incorrect result. They will be forbidden in future Python versions.

- gh-142206 [https://github.com/python/cpython/issues/142206]: The resource tracker in the **`multiprocessing`** module now uses the original communication protocol, as in Python 3.14.0 and below, by default. This avoids issues with upgrading Python while it is running. (Note that such “in-place” upgrades are not tested.) The tracker remains compatible with subprocesses that use new protocol (that is, subprocesses using Python 3.13.10, 3.14.1 and 3.15).

- gh-142214 [https://github.com/python/cpython/issues/142214]: Fix two regressions in **`dataclasses`** in Python 3.14.1 related to annotations.

    - An exception is no longer raised if `slots=True` is used and the `__init__` method does not have an `__annotate__` attribute (likely because `init=False` was used).
    - An exception is no longer raised if annotations are requested on the `__init__` method and one of the fields is not present in the class annotations. This can occur in certain dynamic scenarios.

  Patch by Jelle Zijlstra.

Core and Builtins

- gh-142218 [https://github.com/python/cpython/issues/142218]: Fix crash when inserting into a split table dictionary with a non **`str`** key that matches an existing key.

Library

- gh-116738 [https://github.com/python/cpython/issues/116738]: Fix **`cmath`** data race when initializing trigonometric tables with subinterpreters.

Python 3.14.1 final

Release date: 2025-12-02

Windows

- gh-139810 [https://github.com/python/cpython/issues/139810]: Installing with `py install 3[.x]-dev` will now select final versions as well as prereleases.

Tools/Demos

- gh-141692 [https://github.com/python/cpython/issues/141692]: Each slice of an iOS XCframework now contains a `lib` folder that contains a symlink to the libpython dylib. This allows binary modules to be compiled for iOS using dynamic libreary linking, rather than Framework linking.
- gh-141442 [https://github.com/python/cpython/issues/141442]: The iOS testbed now correctly handles test arguments that contain spaces.
- gh-140702 [https://github.com/python/cpython/issues/140702]: The iOS testbed app will now expose the `GITHUB_ACTIONS` environment variable to iOS apps being tested.
- gh-137484 [https://github.com/python/cpython/issues/137484]: Have `Tools/wasm/wasi` put the build Python into a directory named after the build triple instead of «build».
- gh-137248 [https://github.com/python/cpython/issues/137248]: Add a `--logdir` option to `Tools/wasm/wasi` for specifying where to write log files.
- gh-137243 [https://github.com/python/cpython/issues/137243]: Have Tools/wasm/wasi detect a WASI SDK install in /opt when it was directly extracted from a release tarball.

Tests

- gh-140482 [https://github.com/python/cpython/issues/140482]: Preserve and restore the state of `stty echo` as part of the test environment.
- gh-140082 [https://github.com/python/cpython/issues/140082]: Update `python -m test` to set `FORCE_COLOR=1` when being run with color enabled so that **`unittest`** which is run by it with redirected output will output in color.
- gh-139208 [https://github.com/python/cpython/issues/139208]: Fix regrtest `--fast-ci --verbose`: don't ignore the `--verbose` option anymore. Patch by Victor Stinner.
- gh-136442 [https://github.com/python/cpython/issues/136442]: Use exitcode `1` instead of `5` if **`unittest.TestCase.setUpClass()`** raises an exception

Security

- gh-139700 [https://github.com/python/cpython/issues/139700]: Check consistency of the zip64 end of central directory record. Support records with «zip64 extensible data» if there are no bytes prepended to the ZIP file.
- gh-139283 [https://github.com/python/cpython/issues/139283]: **`sqlite3`**: correctly handle maximum number of rows to fetch in **`Cursor.fetchmany`** and reject negative values for **`Cursor.arraysize`**. Patch by Bénédikt Tran.
- gh-137836 [https://github.com/python/cpython/issues/137836]: Add support of the «plaintext» element, RAWTEXT elements «xmp», «iframe», «noembed» and «noframes», and optionally RAWTEXT element «noscript» in **`html.parser.HTMLParser`**.
- gh-136063 [https://github.com/python/cpython/issues/136063]: **`email.message`**: ensure linear complexity for legacy HTTP parameters parsing. Patch by Bénédikt Tran.
- gh-136065 [https://github.com/python/cpython/issues/136065]: Fix quadratic complexity in **`os.path.expandvars()`**.
- gh-119451 [https://github.com/python/cpython/issues/119451]: Fix a potential memory denial of service in the **`http.client`** module. When connecting to a malicious server, it could cause an arbitrary amount of memory to be allocated. This

could have led to symptoms including a `MemoryError`, swapping, out of memory (OOM) killed processes or containers, or even system crashes.

- gh-119342 [https://github.com/python/cpython/issues/119342]: Fix a potential memory denial of service in the `plistlib` module. When reading a Plist file received from untrusted source, it could cause an arbitrary amount of memory to be allocated. This could have led to symptoms including a `MemoryError`, swapping, out of memory (OOM) killed processes or containers, or even system crashes.

Library

- gh-74389 [https://github.com/python/cpython/issues/74389]: When the stdin being used by a `subprocess.Popen` instance is closed, this is now ignored in `subprocess.Popen.communicate()` instead of leaving the class in an inconsistent state.

- gh-87512 [https://github.com/python/cpython/issues/87512]: Fix `subprocess.Popen.communicate()` timeout handling on Windows when writing large input. Previously, the timeout was ignored during stdin writing, causing the method to block indefinitely if the child process did not consume input quickly. The stdin write is now performed in a background thread, allowing the timeout to be properly enforced.

- gh-141473 [https://github.com/python/cpython/issues/141473]: When `subprocess.Popen.communicate()` was called with *input* and a *timeout* and is called for a second time after a `TimeoutExpired` exception before the process has died, it should no longer hang.

- gh-59000 [https://github.com/python/cpython/issues/59000]: Fix `pdb` breakpoint resolution for class methods when the module defining the class is not imported.

- gh-141570 [https://github.com/python/cpython/issues/141570]: Support file-like object raising `OSError` from `fileno()` in color detection (`_colorize.can_colorize()`). This can occur when `sys.stdout` is redirected.

- gh-141659 [https://github.com/python/cpython/issues/141659]: Fix bad file descriptor errors from `_posixsubprocess` on AIX.
- gh-141600 [https://github.com/python/cpython/issues/141600]: Fix musl version detection on Void Linux.
- gh-141497 [https://github.com/python/cpython/issues/141497]: **`ipaddress`**: ensure that the methods **`IPv4Network.hosts()`** and **`IPv6Network.hosts()`** always return an iterator.
- gh-140938 [https://github.com/python/cpython/issues/140938]: The **`statistics.stdev()`** and **`statistics.pstdev()`** functions now raise a **`ValueError`** when the input contains an infinity or a NaN.
- gh-124111 [https://github.com/python/cpython/issues/124111]: Updated Tcl threading configuration in **`_tkinter`** to assume that threads are always available in Tcl 9 and later.
- gh-137109 [https://github.com/python/cpython/issues/137109]: The **`os.fork`** and related forking APIs will no longer warn in the common case where Linux or macOS platform APIs return the number of threads in a process and find the answer to be 1 even when a **`os.register_at_fork()`** `after_in_parent=` callback (re)starts a thread.
- gh-141314 [https://github.com/python/cpython/issues/141314]: Fix assertion failure in **`io.TextIOWrapper.tell()`** when reading files with standalone carriage return (`\r`) line endings.
- gh-141311 [https://github.com/python/cpython/issues/141311]: Fix assertion failure in **`io.BytesIO.readinto()`** and undefined behavior arising when read position is above capcity in **`io.BytesIO`**.
- gh-141141 [https://github.com/python/cpython/issues/141141]: Fix a thread safety issue with **`base64.b85decode()`**. Contributed by Benel Tayar.

- gh-137969 [https://github.com/python/cpython/issues/137969]: Fix `annotationlib.ForwardRef.evaluate()` returning `ForwardRef` objects which don't update with new globals.

- gh-140911 [https://github.com/python/cpython/issues/140911]: `collections`: Ensure that the methods `UserString.rindex()` and `UserString.index()` accept `collections.UserString` instances as the sub argument.

- gh-140797 [https://github.com/python/cpython/issues/140797]: The undocumented `re.Scanner` class now forbids regular expressions containing capturing groups in its lexicon patterns. Patterns using capturing groups could previously lead to crashes with segmentation fault. Use non-capturing groups (?:…) instead.

- gh-125115 [https://github.com/python/cpython/issues/125115]: Refactor the `pdb` parsing issue so positional arguments can pass through intuitively.

- gh-140815 [https://github.com/python/cpython/issues/140815]: `faulthandler` now detects if a frame or a code object is invalid or freed. Patch by Victor Stinner.

- gh-100218 [https://github.com/python/cpython/issues/100218]: Correctly set `errno` when `socket.if_nametoindex()` or `socket.if_indextoname()` raise an `OSError`. Patch by Bénédikt Tran.

- gh-140875 [https://github.com/python/cpython/issues/140875]: Fix handling of unclosed character references (named and numerical) followed by the end of file in `html.parser.HTMLParser` with `convert_charrefs=False`.

- gh-140734 [https://github.com/python/cpython/issues/140734]: `multiprocessing`: fix off-by-one error when checking the length of a temporary socket file path. Patch by Bénédikt Tran.

- gh-140874 [https://github.com/python/cpython/issues/140874]: Bump the version of pip bundled in ensurepip to version 25.3

- gh-140691 [https://github.com/python/cpython/issues/140691]: In `urllib.request`, when opening a FTP URL fails because a data connection cannot be made, the control connection's socket is now closed to avoid a `ResourceWarning`.

- gh-103847 [https://github.com/python/cpython/issues/103847]: Fix hang when cancelling process created by `asyncio.create_subprocess_exec()` or `asyncio.create_subprocess_shell()`. Patch by Kumar Aditya.

- gh-120057 [https://github.com/python/cpython/issues/120057]: Add `os.reload_environ()` to `os.__all__`.

- gh-140228 [https://github.com/python/cpython/issues/140228]: Avoid making unnecessary filesystem calls for frozen modules in `linecache` when the global module cache is not present.

- gh-140590 [https://github.com/python/cpython/issues/140590]: Fix arguments checking for the `functools.partial.__setstate__()` that may lead to internal state corruption and crash. Patch by Sergey Miryanov.

- gh-125434 [https://github.com/python/cpython/issues/125434]: Display thread name in `faulthandler` on Windows. Patch by Victor Stinner.

- gh-140634 [https://github.com/python/cpython/issues/140634]: Fix a reference counting bug in `os.sched_param.__reduce__()`.

- gh-140633 [https://github.com/python/cpython/issues/140633]: Ignore `AttributeError` when setting a module's `__file__` attribute when loading an extension module packaged as Apple Framework.

- gh-140593 [https://github.com/python/cpython/issues/140593]:

`xml.parsers.expat`: Fix a memory leak that could affect users with `ElementDeclHandler()` set to a custom element declaration handler. Patch by Sebastian Pipping.

- gh-140607 [https://github.com/python/cpython/issues/140607]: Inside `io.RawIOBase.read()`, validate that the count of bytes returned by `io.RawIOBase.readinto()` is valid (inside the provided buffer).
- gh-138162 [https://github.com/python/cpython/issues/138162]: Fix `logging.LoggerAdapter` with `merge_extra=True` and without the *extra* argument.
- gh-138774 [https://github.com/python/cpython/issues/138774]: `ast.unparse()` now generates full source code when handling `ast.Interpolation` nodes that do not have a specified source.
- gh-140474 [https://github.com/python/cpython/issues/140474]: Fix memory leak in `array.array` when creating arrays from an empty `str` and the `u` type code.
- gh-137530 [https://github.com/python/cpython/issues/137530]: `dataclasses` Fix annotations for generated `__init__` methods by replacing the annotations that were in-line in the generated source code with `__annotate__` functions attached to the methods.
- gh-140348 [https://github.com/python/cpython/issues/140348]: Fix regression in Python 3.14.0 where using the `|` operator on a `typing.Union` object combined with an object that is not a type would raise an error.
- gh-140272 [https://github.com/python/cpython/issues/140272]: Fix memory leak in the `clear()` method of the `dbm.gnu` database.
- gh-140041 [https://github.com/python/cpython/issues/140041]: Fix import of `ctypes` on Android and Cygwin when ABI flags are present.

- gh-140120 [https://github.com/python/cpython/issues/140120]: Fixed a memory leak in `hmac` when it was using the hacl-star backend. Discovered by `@ashm-dev` using AddressSanitizer.

- gh-139905 [https://github.com/python/cpython/issues/139905]: Add suggestion to error message for `typing.Generic` subclasses when `cls.__parameters__` is missing due to a parent class failing to call `super().__init_subclass__()` in its `__init_subclass__`.

- gh-139894 [https://github.com/python/cpython/issues/139894]: Fix incorrect sharing of current task with the child process while forking in `asyncio`. Patch by Kumar Aditya.

- gh-139845 [https://github.com/python/cpython/issues/139845]: Fix to not print KeyboardInterrupt twice in default asyncio REPL.

- gh-139783 [https://github.com/python/cpython/issues/139783]: Fix `inspect.getsourcelines()` for the case when a decorator is followed by a comment or an empty line.

- gh-139809 [https://github.com/python/cpython/issues/139809]: Prevent premature colorization of subparser `prog` in `argparse.ArgumentParser.add_subparsers()` to respect color environment variable changes after parser creation.

- gh-139736 [https://github.com/python/cpython/issues/139736]: Fix excessive indentation in the default `argparse` **`HelpFormatter`**. Patch by Alexander Edland.

- gh-70765 [https://github.com/python/cpython/issues/70765]: `http.server`: fix default handling of HTTP/0.9 requests in `BaseHTTPRequestHandler`. Previously, **`BaseHTTPRequestHandler.parse_request()`** incorrectly waited for headers in the request although those are not supported in HTTP/0.9. Patch by Bénédikt Tran.

- gh-63161 [https://github.com/python/cpython/issues/63161]: Fix `tokenize.detect_encoding()`. Support non-UTF-8 shebang and comments if non-UTF-8 encoding is specified.

Detect decoding error for non-UTF-8 encoding. Detect null bytes in source code.

- gh-139391 [https://github.com/python/cpython/issues/139391]: Fix an issue when, on non-Windows platforms, it was not possible to gracefully exit a `python -m asyncio` process suspended by Ctrl+Z and later resumed by *fg* *[https://manpages.debian.org/fg]* other than with *kill* *[https://manpages.debian.org/kill]*.

- gh-101828 [https://github.com/python/cpython/issues/101828]: Fix `'shift_jisx0213'`, `'shift_jis_2004'`, `'euc_jisx0213'` and `'euc_jis_2004'` codecs truncating null chars as they were treated as part of multi-character sequences.

- gh-139289 [https://github.com/python/cpython/issues/139289]: Do a real lazy-import on **`rlcompleter`** in **`pdb`** and restore the existing completer after importing **`rlcompleter`**.

- gh-139246 [https://github.com/python/cpython/issues/139246]: fix: paste zero-width in default repl width is wrong.

- gh-90949 [https://github.com/python/cpython/issues/90949]: Add **`SetAllocTrackerActivationThreshold()`** and **`SetAllocTrackerMaximumAmplification()`** to xmlparser objects to prevent use of disproportional amounts of dynamic memory from within an Expat parser. Patch by Bénédikt Tran.

- gh-139210 [https://github.com/python/cpython/issues/139210]: Fix use-after-free when reporting unknown event in **`xml.etree.ElementTree.iterparse()`**. Patch by Ken Jin.

- gh-138860 [https://github.com/python/cpython/issues/138860]: Lazy import **`rlcompleter`** in **`pdb`** to avoid deadlock in subprocess.

- gh-112729 [https://github.com/python/cpython/issues/112729]: Fix crash when calling **`concurrent.interpreters.create()`** when the process is out of memory.

- gh-135729 [https://github.com/python/cpython/issues/135729]: Fix unraisable exception during finalization when using **`concurrent.interpreters`** in the REPL.

- gh-139076 [https://github.com/python/cpython/issues/139076]: Fix a bug in the **`pydoc`** module that was hiding functions in a Python module if they were implemented in an extension module and the module did not have `__all__`.

- gh-139065 [https://github.com/python/cpython/issues/139065]: Fix trailing space before a wrapped long word if the line length is exactly *width* in **`textwrap`**.

- gh-139001 [https://github.com/python/cpython/issues/139001]: Fix race condition in **`pathlib.Path`** on the internal `_raw_paths` field.

- gh-138813 [https://github.com/python/cpython/issues/138813]: **`multiprocessing.BaseProcess`** defaults `kwargs` to `None` instead of a shared dictionary.

- gh-138993 [https://github.com/python/cpython/issues/138993]: Dedent **`credits`** text.

- gh-138891 [https://github.com/python/cpython/issues/138891]: Fix `SyntaxError` when `inspect.get_annotations(f, eval_str=True)` is called on a function annotated with a **PEP 646** [https://peps.python.org/pep-0646/] `star_expression`

- gh-130567 [https://github.com/python/cpython/issues/130567]: Fix possible crash in **`locale.strxfrm()`** due to a platform bug on macOS.

- gh-138859 [https://github.com/python/cpython/issues/138859]: Fix generic type parameterization raising a **`TypeError`** when omitting a **`ParamSpec`** that has a default which is not a list of types.

- gh-138764 [https://github.com/python/cpython/issues/138764]: Prevent **`annotationlib.call_annotate_function()`**

from calling `__annotate__` functions that don't support `VALUE_WITH_FAKE_GLOBALS` in a fake globals namespace with empty globals.

Make `FORWARDREF` and `STRING` annotations fall back to using `VALUE` annotations in the case that neither their own format, nor `VALUE_WITH_FAKE_GLOBALS` are supported.

- gh-138775 [https://github.com/python/cpython/issues/138775]: Use of `python -m` with **`base64`** has been fixed to detect input from a terminal so that it properly notices EOF.
- gh-138779 [https://github.com/python/cpython/issues/138779]: Support device numbers larger than `2**63-1` for the **`st_rdev`** field of the **`os.stat_result`** structure.
- gh-137706 [https://github.com/python/cpython/issues/137706]: Fix the partial evaluation of annotations that use `typing.Annotated[T, x]` where `T` is a forward reference.
- gh-88375 [https://github.com/python/cpython/issues/88375]: Fix normalization of the `robots.txt` rules and URLs in the **`urllib.robotparser`** module. No longer ignore trailing `?`. Distinguish raw special characters `?`, `=` and `&` from the percent-encoded ones.
- gh-111788 [https://github.com/python/cpython/issues/111788]: Fix parsing errors in the **`urllib.robotparser`** module. Don't fail trying to parse weird paths. Don't fail trying to decode non-UTF-8 `robots.txt` files.
- gh-98896 [https://github.com/python/cpython/issues/98896]: Fix a failure in multiprocessing resource_tracker when SharedMemory names contain colons. Patch by Rani Pinchuk.
- gh-138425 [https://github.com/python/cpython/issues/138425]: Fix partial evaluation of **`annotationlib.ForwardRef`** objects which rely on names defined as globals.
- gh-138432 [https://github.com/python/cpython/issues/138432]:

`zoneinfo.reset_tzpath()` will now convert any `os.PathLike` objects it receives into strings before adding them to `TZPATH`. It will raise `TypeError` if anything other than a string is found after this conversion. If given an `os.PathLike` object that represents a relative path, it will now raise `ValueError` instead of `TypeError`, and present a more informative error message.

- gh-138008 [https://github.com/python/cpython/issues/138008]: Fix segmentation faults in the `ctypes` module due to invalid `argtypes`. Patch by Dung Nguyen.

- gh-60462 [https://github.com/python/cpython/issues/60462]: Fix `locale.strxfrm()` on Solaris (and possibly other platforms).

- gh-138239 [https://github.com/python/cpython/issues/138239]: The REPL now highlights `type` as a soft keyword in type statements.

- gh-138204 [https://github.com/python/cpython/issues/138204]: Forbid expansion of shared anonymous `memory maps` on Linux, which caused a bus error.

- gh-138010 [https://github.com/python/cpython/issues/138010]: Fix an issue where defining a class with an `@warnings.deprecated`-decorated base class may not invoke the correct `__init_subclass__()` method in cases involving multiple inheritance. Patch by Brian Schubert.

- gh-138151 [https://github.com/python/cpython/issues/138151]: In `annotationlib`, improve evaluation of forward references to nonlocal variables that are not yet defined when the annotations are initially evaluated.

- gh-137317 [https://github.com/python/cpython/issues/137317]: `inspect.signature()` now correctly handles classes that use a descriptor on a wrapped `__init__()` or `__new__()` method. Contributed by Yongyu Yan.

- gh-137754 [https://github.com/python/cpython/issues/137754]: Fix

import of the `zoneinfo` module if the C implementation of the `datetime` module is not available.

- gh-137490 [https://github.com/python/cpython/issues/137490]: Handle `ECANCELED` in the same way as `EINTR` in `signal.sigwaitinfo()` on NetBSD.

- gh-137477 [https://github.com/python/cpython/issues/137477]: Fix `inspect.getblock()`, `inspect.getsourcelines()` and `inspect.getsource()` for generator expressions.

- gh-137044 [https://github.com/python/cpython/issues/137044]: Return large limit values as positive integers instead of negative integers in `resource.getrlimit()`. Accept large values and reject negative values (except `RLIM_INFINITY`) for limits in `resource.setrlimit()`.

- gh-75989 [https://github.com/python/cpython/issues/75989]: `tarfile.TarFile.extractall()` and `tarfile.TarFile.extract()` now overwrite symlinks when extracting hardlinks. (Contributed by Alexander Enrique Urieles Nieto in gh-75989 [https://github.com/python/cpython/issues/75989].)

- gh-137017 [https://github.com/python/cpython/issues/137017]: Fix `threading.Thread.is_alive` to remain `True` until the underlying OS thread is fully cleaned up. This avoids false negatives in edge cases involving thread monitoring or premature `threading.Thread.is_alive` calls.

- gh-137273 [https://github.com/python/cpython/issues/137273]: Fix debug assertion failure in `locale.setlocale()` on Windows.

- gh-137239 [https://github.com/python/cpython/issues/137239]: `heapq`: Update `heapq.__all__` with `*_max` functions.

- gh-81325 [https://github.com/python/cpython/issues/81325]: `tarfile.TarFile` now accepts a path-like when working on a tar archive. (Contributed by Alexander Enrique Urieles Nieto in gh-81325 [https://github.com/python/cpython/

issues/81325].)

- gh-137185 [https://github.com/python/cpython/issues/137185]: Fix a potential async-signal-safety issue in `faulthandler` when printing C stack traces.
- gh-136914 [https://github.com/python/cpython/issues/136914]: Fix retrieval of `doctest.DocTest.lineno` for objects decorated with `functools.cache()` or `functools.cached_property`.
- gh-136912 [https://github.com/python/cpython/issues/136912]: `hmac.digest()` now properly handles large keys and messages by falling back to the pure Python implementation when necessary. Patch by Bénédikt Tran.
- gh-83424 [https://github.com/python/cpython/issues/83424]: Allows creating a `ctypes.CDLL` without name when passing a handle as an argument.
- gh-136234 [https://github.com/python/cpython/issues/136234]: Fix `asyncio.WriteTransport.writelines()` to be robust to connection failure, by using the same behavior as `write()`.
- gh-136507 [https://github.com/python/cpython/issues/136507]: Fix mimetypes CLI to handle multiple file parameters.
- gh-136057 [https://github.com/python/cpython/issues/136057]: Fixed the bug in `pdb` and `bdb` where `next` and `step` can't go over the line if a loop exists in the line.
- gh-135386 [https://github.com/python/cpython/issues/135386]: Fix opening a `dbm.sqlite3` database for reading from read-only file or directory.
- gh-135444 [https://github.com/python/cpython/issues/135444]: Fix `asyncio.DatagramTransport.sendto()` to account for datagram header size when data cannot be sent.
- gh-126631 [https://github.com/python/cpython/issues/126631]: Fix `multiprocessing` `forkserver` bug which prevented

`__main__` from being preloaded.

- gh-135307 [https://github.com/python/cpython/issues/135307]: **`email`**: Fix exception in `set_content()` when encoding text and max_line_length is set to `0` or `None` (unlimited).

- gh-134453 [https://github.com/python/cpython/issues/134453]: Fixed **`subprocess.Popen.communicate()`** `input=` handling of **`memoryview`** instances that were non-byte shaped on POSIX platforms. Those are now properly cast to a byte shaped view instead of truncating the input. Windows platforms did not have this bug.

- gh-134698 [https://github.com/python/cpython/issues/134698]: Fix a crash when calling methods of **`ssl.SSLContext`** or **`ssl.SSLSocket`** across multiple threads.

- gh-125996 [https://github.com/python/cpython/issues/125996]: Fix thread safety of **`collections.OrderedDict`**. Patch by Kumar Aditya.

- gh-133789 [https://github.com/python/cpython/issues/133789]: Fix unpickling of **`pathlib`** objects that were pickled in Python 3.13.

- gh-127081 [https://github.com/python/cpython/issues/127081]: Fix libc thread safety issues with **`dbm`** by performing stateful operations in critical sections.

- gh-132551 [https://github.com/python/cpython/issues/132551]: Make **`io.BytesIO`** safe in free-threaded build.

- gh-131788 [https://github.com/python/cpython/issues/131788]: Make `ResourceTracker.send` from **`multiprocessing`** re-entrant safe

- gh-118981 [https://github.com/python/cpython/issues/118981]: Fix potential hang in `multiprocessing.popen_spawn_posix` that can happen when the child proc dies early by closing the child fds right away.

- gh-102431 [https://github.com/python/cpython/issues/102431]: Clarify constraints for «logical» arguments in methods of `decimal.Context`.

- gh-78319 [https://github.com/python/cpython/issues/78319]: UTF8 support for the IMAP APPEND command has been made RFC compliant.

- bpo-38735 [https://bugs.python.org/issue?@action=redirect&bpo=38735]: Fix failure when importing a module from the root directory on unix-like platforms with sys.pycache_prefix set.

- bpo-41839 [https://bugs.python.org/issue?@action=redirect&bpo=41839]: Allow negative priority values from `os.sched_get_priority_min()` and `os.sched_get_priority_max()` functions.

IDLE

- gh-96491 [https://github.com/python/cpython/issues/96491]: Deduplicate version number in IDLE shell title bar after saving to a file.
- gh-139742 [https://github.com/python/cpython/issues/139742]: Colorize t-string prefixes for template strings in IDLE, as done for f-string prefixes.

Documentation

- gh-141994 [https://github.com/python/cpython/issues/141994]: `xml.sax.handler`: Make Documentation of `xml.sax.handler.feature_external_ges` warn of opening up to external entity attacks [https://en.wikipedia.org/wiki/XML_external_entity_attack]. Patch by Sebastian Pipping.
- gh-140578 [https://github.com/python/cpython/issues/140578]: Remove outdated sencence in the documentation for `multiprocessing`, that implied that `concurrent.futures.ThreadPoolExecutor` did not exist.

Core and Builtins

- gh-142048 [https://github.com/python/cpython/issues/142048]: Fix quadratically increasing garbage collection delays in free-threaded build.

Library

- gh-116738 [https://github.com/python/cpython/issues/116738]: Fix thread safety issue with `re` scanner objects in free-threaded builds.

Core and Builtins

- gh-141930 [https://github.com/python/cpython/issues/141930]: When importing a module, use Python's regular file object to ensure that writes to `.pyc` files are complete or an appropriate error is raised.
- gh-120158 [https://github.com/python/cpython/issues/120158]: Fix inconsistent state when enabling or disabling monitoring events too many times.
- gh-139653 [https://github.com/python/cpython/issues/139653]: Only raise a `RecursionError` or trigger a fatal error if the stack pointer is both below the limit pointer *and* above the stack base. If outside of these bounds assume that it is OK. This prevents false positives when user-space threads swap stacks.
- gh-139103 [https://github.com/python/cpython/issues/139103]: Improve multithreaded scaling of dataclasses on the free-threaded build.
- gh-141579 [https://github.com/python/cpython/issues/141579]: Fix `sys.activate_stack_trampoline()` to properly support the `perf_jit` backend. Patch by Pablo Galindo.
- gh-114203 [https://github.com/python/cpython/issues/114203]: Skip locking if object is already locked by two-mutex critical section.
- gh-141528 [https://github.com/python/cpython/issues/141528]: Suggest using `concurrent.interpreters.Interpreter.close()` instead of the private `_interpreters.destroy` function when warning about remaining subinterpreters. Patch by

Sergey Miryanov.

- gh-141312 [https://github.com/python/cpython/issues/141312]: Fix the assertion failure in the `__setstate__` method of the range iterator when a non-integer argument is passed. Patch by Sergey Miryanov.

Library

- gh-116738 [https://github.com/python/cpython/issues/116738]: Make csv module thread-safe on the free threaded build.

Core and Builtins

- gh-140939 [https://github.com/python/cpython/issues/140939]: Fix memory leak when `bytearray` or `bytes` is formated with the `%*b` format with a large width that results in a `MemoryError`.

Library

- gh-140260 [https://github.com/python/cpython/issues/140260]: Fix `struct` data race in endian table initialization with subinterpreters. Patch by Shamil Abdulaev.

Core and Builtins

- gh-140530 [https://github.com/python/cpython/issues/140530]: Fix a reference leak when `raise exc from cause` fails. Patch by Bénédikt Tran.
- gh-140373 [https://github.com/python/cpython/issues/140373]: Correctly emit `PY_UNWIND` event when generator object is closed. Patch by Mikhail Efimov.
- gh-140576 [https://github.com/python/cpython/issues/140576]: Fixed crash in `tokenize.generate_tokens()` in case of specific incorrect input. Patch by Mikhail Efimov.
- gh-140551 [https://github.com/python/cpython/issues/140551]: Fixed crash in `dict` if `dict.clear()` is called at the lookup stage. Patch by Mikhail Efimov and Inada Naoki.
- gh-140517 [https://github.com/python/cpython/issues/140517]: Fixed

a reference leak when iterating over the result of `map()` with `strict=True` when the input iterables have different lengths. Patch by Mikhail Efimov.

- gh-140471 [https://github.com/python/cpython/issues/140471]: Fix potential buffer overflow in `ast.AST` node initialization when encountering malformed `_fields` containing non-`str`.
- gh-140431 [https://github.com/python/cpython/issues/140431]: Fix a crash in Python's garbage collector due to partially initialized coroutine objects when coroutine origin tracking depth is enabled (`sys.set_coroutine_origin_tracking_depth()`).

Library

- gh-140398 [https://github.com/python/cpython/issues/140398]: Fix memory leaks in `readline` functions `read_init_file()`, `read_history_file()`, `write_history_file()`, and `append_history_file()` when `PySys_Audit()` fails.

Core and Builtins

- gh-140406 [https://github.com/python/cpython/issues/140406]: Fix memory leak when an object's `__hash__()` method returns an object that isn't an `int`.
- gh-140358 [https://github.com/python/cpython/issues/140358]: Restore elapsed time and unreachable object count in GC debug output. These were inadvertently removed during a refactor of `gc.c`. The debug log now again reports elapsed collection time and the number of unreachable objects. Contributed by Pål Grønås Drange.
- gh-140306 [https://github.com/python/cpython/issues/140306]: Fix memory leaks in cross-interpreter channel operations and shared namespace handling.
- gh-140301 [https://github.com/python/cpython/issues/140301]: Fix memory leak of `PyConfig` in subinterpreters.
- gh-140257 [https://github.com/python/cpython/issues/140257]: Fix data race between interpreter_clear() and take_gil() on eval_breaker during finalization with daemon threads.
- gh-139951 [https://github.com/python/cpython/issues/139951]: Fixes

a regression in GC performance for a growing heap composed mostly of small tuples.
  - Counts number of actually tracked objects, instead of trackable objects. This ensures that untracking tuples has the desired effect of reducing GC overhead.
  - Does not track most untrackable tuples during creation. This prevents large numbers of small tuples causing excessive GCs.
- gh-140104 [https://github.com/python/cpython/issues/140104]: Fix a bug with exception handling in the JIT. Patch by Ken Jin. Bug reported by Daniel Diniz.
- gh-140061 [https://github.com/python/cpython/issues/140061]: Fixing the checking of whether an object is uniquely referenced to ensure free-threaded compatibility. Patch by Sergey Miryanov.
- gh-140067 [https://github.com/python/cpython/issues/140067]: Fix memory leak in sub-interpreter creation.
- gh-140000 [https://github.com/python/cpython/issues/140000]: Fix potential memory leak when a reference cycle exists between an instance of `typing.TypeAliasType`, `typing.TypeVar`, `typing.ParamSpec`, or `typing.TypeVarTuple` and its `__name__` attribute. Patch by Mikhail Efimov.
- gh-139914 [https://github.com/python/cpython/issues/139914]: Restore support for HP PA-RISC, which has an upwards-growing stack.
- gh-139988 [https://github.com/python/cpython/issues/139988]: Fix a memory leak when failing to create a `Union` type. Patch by Bénédikt Tran.
- gh-139748 [https://github.com/python/cpython/issues/139748]: Fix reference leaks in error branches of functions accepting path strings or bytes such as `compile()` and `os.system()`. Patch by Bénédikt Tran.
- gh-139516 [https://github.com/python/cpython/issues/139516]: Fix lambda colon erroneously start format spec in f-string in tokenizer.
- gh-139640 [https://github.com/python/cpython/issues/139640]: `ast.parse()` no longer emits syntax warnings for `return`/`break`/`continue` in `finally` (see **PEP 765** [https://peps.python.org/pep-0765/]) – they are only emitted during

compilation.

- gh-139640 [https://github.com/python/cpython/issues/139640]: Fix swallowing some syntax warnings in different modules if they accidentally have the same message and are emitted from the same line. Fix duplicated warnings in the `finally` block.
- gh-63161 [https://github.com/python/cpython/issues/63161]: Support non-UTF-8 shebang and comments in Python source files if non-UTF-8 encoding is specified. Detect decoding error in comments for default (UTF-8) encoding. Show the line and position of decoding error for default encoding in a traceback. Show the line containing the coding cookie when it conflicts with the BOM in a traceback.

Library

- gh-116738 [https://github.com/python/cpython/issues/116738]: Make **`mmap`** thread-safe on the free threaded build.

Core and Builtins

- gh-138558 [https://github.com/python/cpython/issues/138558]: Fix handling of unusual t-string annotations in annotationlib. Patch by Dave Peck.
- gh-134466 [https://github.com/python/cpython/issues/134466]: Don’t run PyREPL in a degraded environment where setting termios attributes is not allowed.
- gh-138944 [https://github.com/python/cpython/issues/138944]: Fix **`SyntaxError`** message when invalid syntax appears on the same line as a valid `import ... as ...` or `from ... import ... as ...` statement. Patch by Brian Schubert.
- gh-105487 [https://github.com/python/cpython/issues/105487]: Remove non-existent **`__copy__()`**, **`__deepcopy__()`**, and **`__bases__`** from the **`__dir__()`** entries of **`types.GenericAlias`**.
- gh-69605 [https://github.com/python/cpython/issues/69605]: Fix some standard library submodules missing from the REPL auto-completion of imports.

Library

- gh-116738 [https://github.com/python/cpython/issues/116738]: Make `cProfile` thread-safe on the free threaded build.
- gh-138004 [https://github.com/python/cpython/issues/138004]: On Solaris/Illumos platforms, thread names are now encoded as ASCII to avoid errors on systems (e.g. OpenIndiana) that don't support non-ASCII names.

Core and Builtins

- gh-137433 [https://github.com/python/cpython/issues/137433]: Fix a potential deadlock in the free threading build when daemon threads enable or disable profiling or tracing while the main thread is shutting down the interpreter.
- gh-137400 [https://github.com/python/cpython/issues/137400]: Fix a crash in the free threading build when disabling profiling or tracing across all threads with `PyEval_SetProfileAllThreads()` or `PyEval_SetTraceAllThreads()` or their Python equivalents `threading.settrace_all_threads()` and `threading.setprofile_all_threads()`.
- gh-58124 [https://github.com/python/cpython/issues/58124]: Fix name of the Python encoding in Unicode errors of the code page codec: use «cp65000» and «cp65001» instead of «CP_UTF7» and «CP_UTF8» which are not valid Python code names. Patch by Victor Stinner.
- gh-132657 [https://github.com/python/cpython/issues/132657]: Improve performance of `frozenset` by removing locks in the free-threading build.
- gh-133400 [https://github.com/python/cpython/issues/133400]: Fixed Ctrl+D (^D) behavior in _pyrepl module to match old pre-3.13 REPL behavior.
- gh-128640 [https://github.com/python/cpython/issues/128640]: Fix a crash when using threads inside of a subinterpreter.

C API

- gh-137422 [https://github.com/python/cpython/issues/137422]: Fix free threading race condition in `PyImport_AddModuleRef()`. It was previously possible for two calls to the function return two different objects, only one of which was stored in `sys.modules`.

- gh-140042 [https://github.com/python/cpython/issues/140042]: Removed the sqlite3_shutdown call that could cause closing connections for sqlite when used with multiple sub interpreters.
- gh-141042 [https://github.com/python/cpython/issues/141042]: Make qNaN in `PyFloat_Pack2()` and `PyFloat_Pack4()`, if while conversion to a narrower precision floating-point format — the remaining after truncation payload will be zero. Patch by Sergey B Kirpichev.
- gh-140487 [https://github.com/python/cpython/issues/140487]: Fix `Py_RETURN_NOTIMPLEMENTED` in limited C API 3.11 and older: don't treat `Py_NotImplemented` as immortal. Patch by Victor Stinner.
- gh-140153 [https://github.com/python/cpython/issues/140153]: Fix `Py_REFCNT()` definition on limited C API 3.11-3.13. Patch by Victor Stinner.
- gh-139653 [https://github.com/python/cpython/issues/139653]: Add `PyUnstable_ThreadState_SetStackProtection()` and `PyUnstable_ThreadState_ResetStackProtection()` functions to set the stack protection base address and stack protection size of a Python thread state. Patch by Victor Stinner.

Build

- gh-141808 [https://github.com/python/cpython/issues/141808]: Do not generate the jit stencils twice in case of PGO builds on Windows.

- gh-141784 [https://github.com/python/cpython/issues/141784]: Fix `_remote_debugging_module.c` compilation on 32-bit Linux. Include Python.h before system headers to make sure that `_remote_debugging_module.c` uses the same types (ABI) than Python. Patch by Victor Stinner.

- gh-140768 [https://github.com/python/cpython/issues/140768]: Warn when the WASI SDK version doesn't match what's supported.

- gh-140513 [https://github.com/python/cpython/issues/140513]: Generate a clear compilation error when `_Py_TAIL_CALL_INTERP` is enabled but either `preserve_none` or `musttail` is not supported.

- gh-140189 [https://github.com/python/cpython/issues/140189]: iOS builds were added to CI.

- gh-138489 [https://github.com/python/cpython/issues/138489]: When cross-compiling for WASI by `build_wasm` or `build_emscripten`, the `build-details.json` step is now included in the build process, just like with native builds.

  This fixes the `libinstall` task which requires the `build-details.json` file during the process.

- gh-137618 [https://github.com/python/cpython/issues/137618]: `PYTHON_FOR_REGEN` now requires Python 3.10 to Python 3.15. Patch by Adam Turner.

- gh-123681 [https://github.com/python/cpython/issues/123681]: Check the `strftime()` behavior at runtime instead of at the compile time to support cross-compiling. Remove the internal macro `_Py_NORMALIZE_CENTURY`.

Python 3.14.0 final

Release date: 2025-10-07

macOS

- gh-124111 [https://github.com/python/cpython/issues/124111]: Update macOS installer to use Tcl/Tk 8.6.17.
- gh-139573 [https://github.com/python/cpython/issues/139573]: Updated bundled version of OpenSSL to 3.0.18.

Windows

- gh-139573 [https://github.com/python/cpython/issues/139573]:

Updated bundled version of OpenSSL to 3.0.18.

Tools/Demos

- gh-139330 [https://github.com/python/cpython/issues/139330]: SBOM generation tool didn't cross-check the version and checksum values against the `Modules/expat/refresh.sh` script, leading to the values becoming out-of-date during routine updates.
- gh-132006 [https://github.com/python/cpython/issues/132006]: XCframeworks now include privacy manifests to satisfy Apple App Store submission requirements.
- gh-138171 [https://github.com/python/cpython/issues/138171]: A script for building an iOS XCframework was added. As part of this change, the top level `iOS` folder has been moved to be a subdirectory of the `Apple` folder.

Security

- gh-139400 [https://github.com/python/cpython/issues/139400]: **`xml.parsers.expat`**: Make sure that parent Expat parsers are only garbage-collected once they are no longer referenced by subparsers created by **`ExternalEntityParserCreate()`**. Patch by Sebastian Pipping.

Library

- gh-139312 [https://github.com/python/cpython/issues/139312]: Upgrade bundled libexpat to 2.7.3

Python 3.14.0 release candidate 3

Release date: 2025-09-18

Windows

- gh-138896 [https://github.com/python/cpython/issues/138896]: Fix error installing C runtime on non-updated Windows machines

Tools/Demos

- gh-137873 [https://github.com/python/cpython/issues/137873]: The iOS test runner has been simplified, resolving some issues that have been observed using the runner in GitHub Actions and Azure Pipelines test environments.

Security

- gh-135661 [https://github.com/python/cpython/issues/135661]: Fix CDATA section parsing in `html.parser.HTMLParser` according to the HTML5 standard: `] ]>` and `]] >` no longer end the CDATA section. Add private method `_set_support_cdata()` which can be used to specify how to parse `<[CDATA[` — as a CDATA section in foreign content (SVG or MathML) or as a bogus comment in the HTML namespace.

Library

- gh-138998 [https://github.com/python/cpython/issues/138998]: Update bundled libexpat to 2.7.2
- gh-118803 [https://github.com/python/cpython/issues/118803]: Add back `collections.abc.ByteString` and `typing.ByteString`. Both had been removed in prior alpha, beta and release candidates for Python 3.14, but their removal has now been postponed to Python 3.17.
- gh-137226 [https://github.com/python/cpython/issues/137226]: Fix `typing.get_type_hints()` calls on generic `typing.TypedDict` classes defined with string annotations.
- gh-138804 [https://github.com/python/cpython/issues/138804]: Raise `TypeError` instead of `AttributeError` when an argument of incorrect type is passed to `shlex.quote()`. This restores the behavior of the function prior to 3.14.
- gh-128636 [https://github.com/python/cpython/issues/128636]: Fix crash in PyREPL when os.environ is overwritten with an invalid value for mac
- gh-138514 [https://github.com/python/cpython/issues/138514]: Raise `ValueError` when a multi-character string is passed to the *echo_char* parameter of `getpass.getpass()`. Patch by

Benjamin Johnson.

- gh-138515 [https://github.com/python/cpython/issues/138515]: `email` is added to Emscripten build.
- gh-99948 [https://github.com/python/cpython/issues/99948]: `ctypes.util.find_library()` now works in Emscripten build.
- gh-138253 [https://github.com/python/cpython/issues/138253]: Add the *block* parameter in the `put()` and `get()` methods of the `concurrent.interpreters` queues for compatibility with the `queue.Queue` interface.
- gh-138133 [https://github.com/python/cpython/issues/138133]: Prevent infinite traceback loop when sending CTRLˆC to Python through `strace`.
- gh-134869 [https://github.com/python/cpython/issues/134869]: Fix an issue where pressing Ctrl+C during tab completion in the REPL would leave the autocompletion menu in a corrupted state.
- gh-90548 [https://github.com/python/cpython/issues/90548]: Fix `musl` detection for `platform.libc_ver()` on Alpine Linux if compiled with –strip-all.
- gh-136134 [https://github.com/python/cpython/issues/136134]: `SMTP.auth_cram_md5()` now raises an `SMTPException` instead of a `ValueError` if Python has been built without MD5 support. In particular, `SMTP` clients will not attempt to use this method even if the remote server is assumed to support it. Patch by Bénédikt Tran.
- gh-136134 [https://github.com/python/cpython/issues/136134]: `IMAP4.login_cram_md5` now raises an `IMAP4.error` if CRAM-MD5 authentication is not supported. Patch by Bénédikt Tran.
- gh-134953 [https://github.com/python/cpython/issues/134953]: Expand `_colorize` theme with `keyword_constant` and implement in repl.

Core and Builtins

- gh-71810 [https://github.com/python/cpython/issues/71810]: Raise `OverflowError` for `(-1).to_bytes()` for signed conversions when bytes count is zero. Patch by Sergey B Kirpichev.

- gh-138192 [https://github.com/python/cpython/issues/138192]: Fix `contextvars` initialization so that all subinterpreters are assigned the `MISSING` value.
- gh-138479 [https://github.com/python/cpython/issues/138479]: Fix a crash when a generic object's `__typing_subst__` returns an object that isn't a `tuple`.
- gh-138372 [https://github.com/python/cpython/issues/138372]: Fix `SyntaxWarning` emitted for erroneous subscript expressions involving template string literals. Patch by Brian Schubert.
- gh-138318 [https://github.com/python/cpython/issues/138318]: The default REPL now avoids highlighting built-in names (for instance `set` or `format()`) when they are used as attribute names (for instance in `value.set` or `text.format`).
- gh-138349 [https://github.com/python/cpython/issues/138349]: Fix crash in certain cases where a module contains both a module-level annotation and a comprehension.
- gh-137384 [https://github.com/python/cpython/issues/137384]: Fix a crash when using the `warnings` module in a finalizer at shutdown. Patch by Kumar Aditya.
- gh-137883 [https://github.com/python/cpython/issues/137883]: Fix runaway recursion when calling a function with keyword arguments.
- gh-137079 [https://github.com/python/cpython/issues/137079]: Fix keyword typo recognition when parsing files. Patch by Pablo Galindo.
- gh-137728 [https://github.com/python/cpython/issues/137728]: Fix the JIT's handling of many local variables. This previously caused a segfault.
- gh-137576 [https://github.com/python/cpython/issues/137576]: Fix for incorrect source code being shown in tracebacks from the Basic REPL when `PYTHONSTARTUP` is given. Patch by Adam Hartz.

Python 3.14.0 release candidate 2

Release date: 2025-08-14

macOS

- gh-137450 [https://github.com/python/cpython/issues/137450]: macOS installer shell path management improvements: separate the installer `Shell profile updater` postinstall script from the `Update Shell Profile.command` to enable more robust error handling.
- gh-137134 [https://github.com/python/cpython/issues/137134]: Update macOS installer to ship with SQLite version 3.50.4.

Windows

- gh-137134 [https://github.com/python/cpython/issues/137134]: Update Windows installer to ship with SQLite 3.50.4.

Library

- gh-137426 [https://github.com/python/cpython/issues/137426]: Remove the code deprecation of `importlib.abc.ResourceLoader`. It is documented as deprecated, but left for backwards compatibility with other classes in `importlib.abc`.
- gh-137282 [https://github.com/python/cpython/issues/137282]: Fix tab completion and **`dir()`** on **`concurrent.futures`**.
- gh-137257 [https://github.com/python/cpython/issues/137257]: Bump the version of pip bundled in ensurepip to version 25.2
- gh-137226 [https://github.com/python/cpython/issues/137226]: Fix behavior of **`annotationlib.ForwardRef.evaluate()`** when the *type_params* parameter is passed and the name of a type param is also present in an enclosing scope.
- gh-130522 [https://github.com/python/cpython/issues/130522]: Fix unraisable **`TypeError`** raised during interpreter shutdown in the **`threading`** module.
- gh-137059 [https://github.com/python/cpython/issues/137059]: Fix handling of file URLs with a Windows drive letter in the URL authority by **`urllib.request.url2pathname()`**. This fixes a regression in earlier pre-releases of Python 3.14.
- gh-130577 [https://github.com/python/cpython/issues/130577]: **`tarfile`** now validates archives to ensure member offsets are non-negative. (Contributed by Alexander Enrique Urieles Nieto in gh-130577 [https://github.com/python/cpython/issues/130577].)

- gh-135228 [https://github.com/python/cpython/issues/135228]: When **`dataclasses`** replaces a class with a slotted dataclass, the original class can now be garbage collected again. Earlier changes in Python 3.14 caused this class to always remain in existence together with the replacement class synthesized by **`dataclasses`**.

Documentation

- gh-136155 [https://github.com/python/cpython/issues/136155]: We are now checking for fatal errors in EPUB builds in CI.

Core and Builtins

- gh-137400 [https://github.com/python/cpython/issues/137400]: Fix a crash in the free threading build when disabling profiling or tracing across all threads with **`PyEval_SetProfileAllThreads()`** or **`PyEval_SetTraceAllThreads()`** or their Python equivalents **`threading.settrace_all_threads()`** and **`threading.setprofile_all_threads()`**.
- gh-137314 [https://github.com/python/cpython/issues/137314]: Fixed a regression where raw f-strings incorrectly interpreted escape sequences in format specifications. Raw f-strings now properly preserve literal backslashes in format specs, matching the behavior from Python 3.11. For example, `rf"{obj:\xFF}"` now correctly produces `'\\xFF'` instead of `'ÿ'`. Patch by Pablo Galindo.
- gh-137308 [https://github.com/python/cpython/issues/137308]: A standalone docstring in a node body is optimized as a **`pass`** statement to ensure that the node's body is never empty. There was a **`ValueError`** in **`compile()`** otherwise.
- gh-137288 [https://github.com/python/cpython/issues/137288]: Fix bug where some bytecode instructions of a boolean expression are not associated with the correct exception handler.
- gh-134291 [https://github.com/python/cpython/issues/134291]: Remove some newer macOS API usage from the JIT compiler in order to restore compatibility with older OSX 10.15 deployment targets.

- gh-131338 [https://github.com/python/cpython/issues/131338]: Disable computed stack limit checks on non-glibc linux platforms to fix crashes on deep recursion.
- gh-136870 [https://github.com/python/cpython/issues/136870]: Fix data races while de-instrumenting bytecode of code objects running concurrently in threads.

C API

- gh-137573 [https://github.com/python/cpython/issues/137573]: Mark `_PyOptimizer_Optimize` as **`Py_NO_INLINE`** to prevent stack overflow crashes on macOS.

Build

- gh-132339 [https://github.com/python/cpython/issues/132339]: Add support for OpenSSL 3.5.

Python 3.14.0 release candidate 1

Release date: 2025-07-22

Tools/Demos

- gh-136251 [https://github.com/python/cpython/issues/136251]: Fixes and usability improvements for `Tools/wasm/emscripten/web_example`

Security

- gh-135661 [https://github.com/python/cpython/issues/135661]: Fix parsing attributes with whitespaces around the `=` separator in **`html.parser.HTMLParser`** according to the HTML5 standard.
- gh-118350 [https://github.com/python/cpython/issues/118350]: Fix support of escapable raw text mode (elements «textarea» and «title») in **`html.parser.HTMLParser`**.

Library

- gh-136170 [https://github.com/python/cpython/issues/136170]: Removed the unreleased `zipfile.ZipFile.data_offset` property added in 3.14.0a7 as it wasn't fully clear which behavior it should have in some situations so the result was not always what a user might expect.
- gh-124621 [https://github.com/python/cpython/issues/124621]: pyrepl now works in Emscripten.
- gh-136874 [https://github.com/python/cpython/issues/136874]: Discard URL query and fragment in **`urllib.request.url2pathname()`**.
- gh-130645 [https://github.com/python/cpython/issues/130645]: Enable color help by default in **`argparse`**.
- gh-136549 [https://github.com/python/cpython/issues/136549]: Fix signature of **`threading.excepthook()`**.
- gh-136523 [https://github.com/python/cpython/issues/136523]: Fix **`wave.Wave_write`** emitting an unraisable when open raises.
- gh-52876 [https://github.com/python/cpython/issues/52876]: Add missing `keepends` (default `True`) parameter to **`codecs.StreamReaderWriter.readline()`** and **`codecs.StreamReaderWriter.readlines()`**.
- gh-136470 [https://github.com/python/cpython/issues/136470]: Correct **`concurrent.futures.InterpreterPoolExecutor`**'s default thread name.
- gh-136476 [https://github.com/python/cpython/issues/136476]: Fix a bug that was causing the `get_async_stack_trace` function to miss some frames in the stack trace.
- gh-136434 [https://github.com/python/cpython/issues/136434]: Fix docs generation of `UnboundItem` in **`concurrent.interpreters`** when running with **`-OO`**.
- gh-136380 [https://github.com/python/cpython/issues/136380]: Raises **`AttributeError`** when accessing **`concurrent.futures.InterpreterPoolExecutor`** and subinterpreters are not available.
- gh-134759 [https://github.com/python/cpython/issues/134759]: Fix **`UnboundLocalError`** in **`email.message.Message.get_payload()`** when the payload to decode is a **`bytes`** object. Patch by Kliment Lamonov.

- gh-134657 [https://github.com/python/cpython/issues/134657]: **`asyncio`**: Remove some private names from `asyncio.__all__`.

Core and Builtins

- gh-136801 [https://github.com/python/cpython/issues/136801]: Fix PyREPL syntax highlighting on match cases after multi-line case. Contributed by Olga Matoula.

Library

- gh-136421 [https://github.com/python/cpython/issues/136421]: Fix crash when initializing **`datetime`** concurrently.

Core and Builtins

- gh-136541 [https://github.com/python/cpython/issues/136541]: Fix some issues with the perf trampolines on x86-64 and aarch64. The trampolines were not being generated correctly for some cases, which could lead to the perf integration not working correctly. Patch by Pablo Galindo.

- gh-136517 [https://github.com/python/cpython/issues/136517]: Fixed a typo that prevented printing of uncollectable objects when the **`gc.DEBUG_UNCOLLECTABLE`** mode was set.

- gh-136525 [https://github.com/python/cpython/issues/136525]: Fix issue where per-thread bytecode was not instrumented for newly created threads.

- gh-132661 [https://github.com/python/cpython/issues/132661]: `Interpolation.expression` now has a default, the empty string.

- gh-132661 [https://github.com/python/cpython/issues/132661]: Reflect recent **PEP 750** [https://peps.python.org/pep-0750/] change.

  Disallow concatenation of `string.templatelib.Template` and **`str`**. Also, disallow implicit concatenation of t-string literals with string or f-string literals.

Library

- gh-116738 [https://github.com/python/cpython/issues/116738]: Make functions in `grp` thread-safe on the free threaded build.

Core and Builtins

- gh-135148 [https://github.com/python/cpython/issues/135148]: Fixed a bug where f-string debug expressions (using =) would incorrectly strip out parts of strings containing escaped quotes and # characters. Patch by Pablo Galindo.
- gh-133136 [https://github.com/python/cpython/issues/133136]: Limit excess memory usage in the free threading build when a large dictionary or list is resized and accessed by multiple threads.
- gh-91153 [https://github.com/python/cpython/issues/91153]: Fix a crash when a `bytearray` is concurrently mutated during item assignment.
- gh-127971 [https://github.com/python/cpython/issues/127971]: Fix off-by-one read beyond the end of a string in string search.

C API

- gh-112068 [https://github.com/python/cpython/issues/112068]: Revert support of nullable arguments in `PyArg_Parse()`.
- gh-133296 [https://github.com/python/cpython/issues/133296]: New variants for the critical section API that accept one or two `PyMutex` pointers rather than `PyObject` instances are now public in the non-limited C API.
- gh-134009 [https://github.com/python/cpython/issues/134009]: Expose `PyMutex_IsLocked()` as part of the public C API.

Build

- gh-135621 [https://github.com/python/cpython/issues/135621]: PyREPL no longer depends on the `curses` standard library. Contributed by Łukasz Langa.

Python 3.14.0 beta 4

Release date: 2025-07-08

Tools/Demos

- gh-135968 [https://github.com/python/cpython/issues/135968]: Stubs for `strip` are now provided as part of an iOS install.
- gh-133600 [https://github.com/python/cpython/issues/133600]: Backport file reorganization for Tools/wasm/wasi.

  This should make backporting future code changes easier. It also simplifies instructions around how to do WASI builds in the devguide.

Tests

- gh-135966 [https://github.com/python/cpython/issues/135966]: The iOS testbed now handles the `app_packages` folder as a site directory.
- gh-135494 [https://github.com/python/cpython/issues/135494]: Fix regrtest to support excluding tests from `--pgo` tests. Patch by Victor Stinner.

Security

- gh-136053 [https://github.com/python/cpython/issues/136053]: **`marshal`**: fix a possible crash when deserializing **`slice`** objects.
- gh-135661 [https://github.com/python/cpython/issues/135661]: Fix parsing start and end tags in **`html.parser.HTMLParser`** according to the HTML5 standard.
  - Whitespaces no longer accepted between `</` and the tag name. E.g. `</ script>` does not end the script section.
  - Vertical tabulation (`\v`) and non-ASCII whitespaces no longer recognized as whitespaces. The only whitespaces are `\t\n\r\f` and space.

  - Null character (U+0000) no longer ends the tag name.
  - Attributes and slashes after the tag name in end tags are now ignored, instead of terminating after the first `>` in quoted attribute value. E.g. `</script/ foo=">"/>`.
  - Multiple slashes and whitespaces between the last attribute and closing `>` are now ignored in both start and end tags. E.g. `<a foo=bar/ //>`.
  - Multiple `=` between attribute name and value are no longer collapsed. E.g. `<a foo==bar>` produces attribute «foo» with value «=bar».
  - [Reverted in gh-136927 [https://github.com/python/cpython/issues/136927]] Whitespaces between the `=` separator and attribute name or value are no longer ignored. E.g. `<a foo =bar>` produces two attributes «foo» and «=bar», both with value None; `<a foo= bar>` produces two attributes: «foo» with value «» and «bar» with value None.
- gh-102555 [https://github.com/python/cpython/issues/102555]: Fix comment parsing in `html.parser.HTMLParser` according to the HTML5 standard. `--!>` now ends the comment. `-- >` no longer ends the comment. Support abnormally ended empty comments `<-->` and `<--->`.

Library

- gh-136286 [https://github.com/python/cpython/issues/136286]: Fix pickling failures for protocols 0 and 1 for many objects realted to subinterpreters.
- gh-136316 [https://github.com/python/cpython/issues/136316]: Improve support for evaluating nested forward references in `typing.evaluate_forward_ref()`.
- gh-85702 [https://github.com/python/cpython/issues/85702]: If `zoneinfo._common.load_tzdata` is given a package without a resource a `zoneinfo.ZoneInfoNotFoundError` is raised rather than a `PermissionError`. Patch by Victor Stinner.
- gh-136028 [https://github.com/python/cpython/issues/136028]: Fix parsing month names containing «İ» (U+0130, LATIN CAPITAL LETTER I WITH DOT ABOVE) in

`time.strptime()`. This affects locales az_AZ, ber_DZ, ber_MA and crh_UA.

- gh-135995 [https://github.com/python/cpython/issues/135995]: In the palmos encoding, make byte `0x9b` decode to › (U+203A - SINGLE RIGHT-POINTING ANGLE QUOTATION MARK).
- gh-53203 [https://github.com/python/cpython/issues/53203]: Fix `time.strptime()` for `%c` and `%x` formats on locales byn_ER, wal_ET and lzh_TW, and for `%X` format on locales ar_SA, bg_BG and lzh_TW.
- gh-91555 [https://github.com/python/cpython/issues/91555]: An earlier change, which was introduced in 3.14.0b2, has been reverted. It disabled logging for a logger during handling of log messages for that logger. Since the reversion, the behaviour should be as it was before 3.14.0b2.
- gh-135878 [https://github.com/python/cpython/issues/135878]: Fixes a crash of `types.SimpleNamespace` on free threading builds, when several threads were calling its `__repr__()` method at the same time.
- gh-135836 [https://github.com/python/cpython/issues/135836]: Fix `IndexError` in `asyncio.loop.create_connection()` that could occur when non-`OSError` exception is raised during connection and socket's `close()` raises **`OSError`**.
- gh-135836 [https://github.com/python/cpython/issues/135836]: Fix `IndexError` in `asyncio.loop.create_connection()` that could occur when the Happy Eyeballs algorithm resulted in an empty exceptions list during connection attempts.
- gh-135855 [https://github.com/python/cpython/issues/135855]: Raise `TypeError` instead of `SystemError` when **`_interpreters.set___main___attrs()`** is passed a non-dict object. Patch by Brian Schubert.
- gh-135815 [https://github.com/python/cpython/issues/135815]: `netrc`: skip security checks if `os.getuid()` is missing. Patch by Bénédikt Tran.
- gh-135640 [https://github.com/python/cpython/issues/135640]: Address bug where it was possible to call `xml.etree.ElementTree.ElementTree.write()` on an ElementTree object with an invalid root element. This behavior blanked the file passed to `write` if it already existed.
- gh-135645 [https://github.com/python/cpython/issues/135645]: Added

`supports_isolated_interpreters` field to **`sys.implementation`**.

- gh-135646 [https://github.com/python/cpython/issues/135646]: Raise consistent **`NameError`** exceptions in **`annotationlib.ForwardRef.evaluate()`**
- gh-135557 [https://github.com/python/cpython/issues/135557]: Fix races on **`heapq`** updates and **`list`** reads on the free threaded build.
- gh-119180 [https://github.com/python/cpython/issues/119180]: Only fetch globals and locals if necessary in **`annotationlib.get_annotations()`**
- gh-135561 [https://github.com/python/cpython/issues/135561]: Fix a crash on DEBUG builds when an HACL* HMAC routine fails. Patch by Bénédikt Tran.
- gh-135487 [https://github.com/python/cpython/issues/135487]: Fix **`reprlib.Repr.repr_int()`** when given integers with more than **`sys.get_int_max_str_digits()`** digits. Patch by Bénédikt Tran.
- gh-135335 [https://github.com/python/cpython/issues/135335]: **`multiprocessing`**: Flush `stdout` and `stderr` after preloading modules in the `forkserver`.
- gh-135069 [https://github.com/python/cpython/issues/135069]: Fix the «Invalid error handling» exception in **`encodings.idna.IncrementalDecoder`** to correctly replace the “errors” parameter.
- gh-130662 [https://github.com/python/cpython/issues/130662]: +Accept leading zeros in precision and width fields for +:class:**`Decimal`** formatting, for example `format(Decimal(1.25), '.016f')`.
- gh-130662 [https://github.com/python/cpython/issues/130662]: Accept leading zeros in precision and width fields for **`Fraction`** formatting, for example `format(Fraction(1, 3), '.016f')`.
- gh-87790 [https://github.com/python/cpython/issues/87790]: Support underscore and comma as thousands separators in the fractional part for **`Fraction`**’s formatting. Patch by Sergey B Kirpichev.
- gh-87790 [https://github.com/python/cpython/issues/87790]: Support underscore and comma as thousands separators in the fractional part for **`Decimal`**’s formatting. Patch by Sergey B

Kirpichev.

- gh-130664 [https://github.com/python/cpython/issues/130664]: Handle corner-case for **`Fraction`**'s formatting: treat zero-padding (preceding the width field by a zero (`'0'`) character) as an equivalent to a fill character of `'0'` with an alignment type of `'='`, just as in case of **`float`**'s.

Documentation

- gh-136155 [https://github.com/python/cpython/issues/136155]: EPUB builds are fixed by excluding non-XHTML-compatible tags.

Core and Builtins

- gh-109700 [https://github.com/python/cpython/issues/109700]: Fix memory error handling in **`PyDict_SetDefault()`**.
- gh-78465 [https://github.com/python/cpython/issues/78465]: Fix error message for `cls.__new__(cls, ...)` where `cls` is not instantiable builtin or extension type (with `tp_new` set to `NULL`).
- gh-129958 [https://github.com/python/cpython/issues/129958]: Differentiate between t-strings and f-strings in syntax error for newlines in format specifiers of single-quoted interpolated strings.
- gh-135871 [https://github.com/python/cpython/issues/135871]: Non-blocking mutex lock attempts now return immediately when the lock is busy instead of briefly spinning in the free threading build.
- gh-135106 [https://github.com/python/cpython/issues/135106]: Restrict the trashcan mechanism to GC'ed objects and untrack them while in the trashcan to prevent the GC and trashcan mechanisms conflicting.
- gh-135607 [https://github.com/python/cpython/issues/135607]: Fix potential **`weakref`** races in an object's destructor on the free threaded build.
- gh-135608 [https://github.com/python/cpython/issues/135608]: Fix a crash in the JIT involving attributes of modules.
- gh-135543 [https://github.com/python/cpython/issues/135543]: Emit `sys.remote_exec` audit event when **`sys.remote_exec()`** is called and migrate

`remote_debugger_script` to `cpython.remote_debugger_script`.
- gh-135280 [https://github.com/python/cpython/issues/134280]: Disable constant folding for `~` with a boolean argument. This moves the deprecation warning from compile time to runtime.

C API

- gh-135906 [https://github.com/python/cpython/issues/135906]: Fix compilation errors when compiling the internal headers with a C++ compiler.

Build

- gh-134273 [https://github.com/python/cpython/issues/134273]: Add support for configuring compiler flags for the JIT with `CFLAGS_JIT`

Python 3.14.0 beta 3

Release date: 2025-06-17

Windows

- gh-135099 [https://github.com/python/cpython/issues/135099]: Fix a crash that could occur on Windows when a background thread waits on a **`PyMutex`** while the main thread is shutting down the interpreter.

Tests

- gh-132815 [https://github.com/python/cpython/issues/132815]: Fix test_opcode: add `JUMP_BACKWARD` to specialization stats.
- gh-135489 [https://github.com/python/cpython/issues/135489]: Show verbose output for failing tests during PGO profiling step with –enable-optimizations.
- gh-135120 [https://github.com/python/cpython/issues/135120]: Add **`test.support.subTests()`**.

Security

- gh-135462 [https://github.com/python/cpython/issues/135462]: Fix quadratic complexity in processing specially crafted input in **`html.parser.HTMLParser`**. End-of-file errors are now handled according to the HTML5 specs – comments and declarations are automatically closed, tags are ignored.

- gh-135034 [https://github.com/python/cpython/issues/135034]: Fixes multiple issues that allowed `tarfile` extraction filters (`filter="data"` and `filter="tar"`) to be bypassed using crafted symlinks and hard links.

  Addresses **CVE 2024-12718** [https://www.cve.org/CVERecord?id=CVE-2024-12718], **CVE 2025-4138** [https://www.cve.org/CVERecord?id=CVE-2025-4138], **CVE 2025-4330** [https://www.cve.org/CVERecord?id=CVE-2025-4330], and **CVE 2025-4517** [https://www.cve.org/CVERecord?id=CVE-2025-4517].

Library

- gh-65697 [https://github.com/python/cpython/issues/65697]: **`configparser`**'s error message when attempting to write an invalid key is now more helpful.
- gh-135497 [https://github.com/python/cpython/issues/135497]: Fix **`os.getlogin()`** failing for longer usernames on BSD-based platforms.
- gh-135429 [https://github.com/python/cpython/issues/135429]: Fix the argument mismatch in `_lsprof` for `PY_THROW` event.
- gh-135368 [https://github.com/python/cpython/issues/135368]: Fix **`unittest.mock.Mock`** generation on **`dataclasses.dataclass()`** objects. Now all special attributes are set as it was before gh-124429 [https://github.com/python/cpython/issues/124429].
- gh-133967 [https://github.com/python/cpython/issues/133967]: Do not normalize **`locale`** name "C.UTF-8" to "en_US.UTF-8".
- gh-135321 [https://github.com/python/cpython/issues/135321]: Raise a correct exception for values greater than 0x7fffffff for the `BINSTRING` opcode in the C implementation of **`pickle`**.
- gh-135276 [https://github.com/python/cpython/issues/135276]:

Backported bugfixes in zipfile.Path from zipp 3.23. Fixed `.name`, `.stem` and other basename-based properties on Windows when working with a zipfile on disk.

- gh-135244 [https://github.com/python/cpython/issues/135244]: **`uuid`**: when the MAC address cannot be determined, the 48-bit node ID is now generated with a cryptographically-secure pseudo-random number generator (CSPRNG) as per **RFC 9562, §6.10.3** [https://datatracker.ietf.org/doc/html/rfc9562.html#section-6.10-3]. This affects **`uuid1()`** and **`uuid6()`**.
- gh-134970 [https://github.com/python/cpython/issues/134970]: Fix the «unknown action» exception in **`argparse.ArgumentParser.add_argument_group()`** to correctly replace the action class.
- gh-134718 [https://github.com/python/cpython/issues/134718]: **`ast.dump()`** now only omits `None` and `[]` values if they are default values.
- gh-134939 [https://github.com/python/cpython/issues/134939]: Add the **`concurrent.interpreters`** module. See **PEP 734** [https://peps.python.org/pep-0734/].
- gh-134885 [https://github.com/python/cpython/issues/134885]: Fix possible crash in the **`compression.zstd`** module related to setting parameter types. Patch by Jelle Zijlstra.
- gh-134857 [https://github.com/python/cpython/issues/134857]: Improve error report for **`doctest`**s run with **`unittest`**. Remove **`doctest`** module frames from tracebacks and redundant newline character from a failure message.
- gh-128840 [https://github.com/python/cpython/issues/128840]: Fix parsing long IPv6 addresses with embedded IPv4 address.
- gh-134637 [https://github.com/python/cpython/issues/134637]: Fix performance regression in calling a **`ctypes`** function pointer in free threading.
- gh-134696 [https://github.com/python/cpython/issues/134696]: Built-in HACL* and OpenSSL implementations of hash function constructors now correctly accept the same *documented* named arguments. For instance, **`md5()`** could be previously invoked as `md5(data=data)` or `md5(string=string)` depending on the underlying implementation but these calls were not compatible. Patch by Bénédikt Tran.
- gh-134151 [https://github.com/python/cpython/issues/134151]:

`email`: Fix **`TypeError`** in **`email.utils.decode_params()`** when sorting **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html] continuations that contain an unnumbered section.

- gh-134210 [https://github.com/python/cpython/issues/134210]: **`curses.window.getch()`** now correctly handles signals. Patch by Bénédikt Tran.
- gh-134152 [https://github.com/python/cpython/issues/134152]: **`email`**: Fix parsing of email message ID with invalid domain.
- gh-133489 [https://github.com/python/cpython/issues/133489]: **`random.getrandbits()`** can now generate more that 2^{31} bits. **`random.randbytes()`** can now generate more that 256 MiB.
- gh-132813 [https://github.com/python/cpython/issues/132813]: Improve error messages for incorrect types and values of **`csv.Dialect`** attributes.
- gh-132969 [https://github.com/python/cpython/issues/132969]: Prevent the **`ProcessPoolExecutor`** executor thread, which remains running when **`shutdown(wait=False)`**, from attempting to adjust the pool's worker processes after the object state has already been reset during shutdown. A combination of conditions, including a worker process having terminated abormally, resulted in an exception and a potential hang when the still-running executor thread attempted to replace dead workers within the pool.
- gh-127081 [https://github.com/python/cpython/issues/127081]: Fix libc thread safety issues with **`os`** by replacing `getlogin` with `getlogin_r` re-entrant version.
- gh-131884 [https://github.com/python/cpython/issues/131884]: Fix formatting issues in **`json.dump()`** when both *indent* and *skipkeys* are used.
- gh-130999 [https://github.com/python/cpython/issues/130999]: Avoid exiting the new REPL and offer suggestions even if there are non-string candidates when errors occur.

Documentation

- gh-135171 [https://github.com/python/cpython/issues/135171]: Document that the iterator for the leftmost **`for`** clause in the generator expression is created immediately.

- bpo-45210 [https://bugs.python.org/issue?@action=redirect&bpo=45210]: Document that error indicator may be set in tp_dealloc, and how to avoid clobbering it.

Core and Builtins

- gh-135496 [https://github.com/python/cpython/issues/135496]: Fix typo in the f-string conversion type error («exclamanation» -> «exclamation»).
- gh-135371 [https://github.com/python/cpython/issues/135371]: Fixed **`asyncio`** debugging tools to properly display internal coroutine call stacks alongside external task dependencies. The `python -m asyncio ps` and `python -m asyncio pstree` commands now show complete execution context. Patch by Pablo Galindo.

Library

- gh-127319 [https://github.com/python/cpython/issues/127319]: Set the `allow_reuse_port` class variable to `False` on the XMLRPC, logging, and HTTP servers. This matches the behavior in prior Python releases, which is to not allow port reuse.

Core and Builtins

- gh-135171 [https://github.com/python/cpython/issues/135171]: Reverts the behavior of async generator expressions when created with object w/o _aiter_ method to the pre-3.13 behavior of raising a TypeError.
- gh-130077 [https://github.com/python/cpython/issues/130077]: Properly raise custom syntax errors when incorrect syntax containing names that are prefixes of soft keywords is encountered. Patch by Pablo Galindo.
- gh-135171 [https://github.com/python/cpython/issues/135171]: Reverts the behavior of generator expressions when created with a non-iterable to the pre-3.13 behavior of raising a TypeError. It is no longer possible to cause a crash in the debugger by altering the generator expression's local variables. This is achieved by moving the `GET_ITER`

instruction back to the creation of the generator expression and adding an additional check to `FOR_ITER`.

Library

- gh-116738 [https://github.com/python/cpython/issues/116738]: Make methods in **`heapq`** thread-safe on the free threaded build.

Core and Builtins

- gh-134876 [https://github.com/python/cpython/issues/134876]: Add support to **PEP 768** [https://peps.python.org/pep-0768/] remote debugging for Linux kernels which don't have CONFIG_CROSS_MEMORY_ATTACH configured.
- gh-134889 [https://github.com/python/cpython/issues/134889]: Fix handling of a few opcodes that leave operands on the stack when optimizing `LOAD_FAST`.

Library

- gh-134908 [https://github.com/python/cpython/issues/134908]: Fix crash when iterating over lines in a text file on the free threaded build.

Core and Builtins

- gh-132617 [https://github.com/python/cpython/issues/132617]: Fix **`dict.update()`** modification check that could incorrectly raise a «dict mutated during update» error when a different dictionary was modified that happens to share the same underlying keys object.
- gh-134679 [https://github.com/python/cpython/issues/134679]: Fix crash in the free threading build's QSBR code that could occur when changing an object's `__dict__` attribute.
- gh-127682 [https://github.com/python/cpython/issues/127682]: No longer call `__iter__` twice in list comprehensions. This brings the behavior of list comprehensions in line with other forms of iteration
- gh-133912 [https://github.com/python/cpython/issues/133912]: Fix

the C API function `PyObject_GenericSetDict` to handle extension classes with inline values.

C API

- gh-134989 [https://github.com/python/cpython/issues/134989]: Fix `Py_RETURN_NONE`, `Py_RETURN_TRUE` and `Py_RETURN_FALSE` macros in the limited C API 3.11 and older: don't treat `Py_None`, `Py_True` and `Py_False` as immortal. Patch by Victor Stinner.
- gh-134989 [https://github.com/python/cpython/issues/134989]: Implement **`PyObject_DelAttr()`** and **`PyObject_DelAttrString()`** as macros in the limited C API 3.12 and older. Patch by Victor Stinner.
- gh-133968 [https://github.com/python/cpython/issues/133968]: Add **`PyUnicodeWriter_WriteASCII()`** function to write an ASCII string into a **`PyUnicodeWriter`**. The function is faster than **`PyUnicodeWriter_WriteUTF8()`**, but has an undefined behavior if the input string contains non-ASCII characters. Patch by Victor Stinner.

Build

- gh-119132 [https://github.com/python/cpython/issues/119132]: Remove «experimental» tag from the CPython free-threading build.
- gh-135497 [https://github.com/python/cpython/issues/135497]: Fix the detection of `MAXLOGNAME` in the `configure.ac` script.
- gh-134923 [https://github.com/python/cpython/issues/134923]: Windows builds with profile-guided optimization enabled now use `/GENPROFILE` and `/USEPROFILE` instead of deprecated `/LTCG:` options.
- gh-134774 [https://github.com/python/cpython/issues/134774]: Fix **`Py_DEBUG`** macro redefinition warnings on Windows debug builds. Patch by Chris Eibl.
- gh-134632 [https://github.com/python/cpython/issues/134632]: Fixed `build-details.json` generation to use `INCLUDEPY`, in order to reference the `pythonX.Y` subdirectory of the include directory, as required in **PEP 739** [https://peps.python.org/pep-0739/], instead of the top-level include

directory.

Python 3.14.0 beta 2

Release date: 2025-05-26

Windows

- gh-130727 [https://github.com/python/cpython/issues/130727]: Fix a race in internal calls into WMI that can result in an «invalid handle» exception under high load. Patch by Chris Eibl.
- gh-76023 [https://github.com/python/cpython/issues/76023]: Make **`os.path.realpath()`** ignore Windows error 1005 when in non-strict mode.
- gh-133779 [https://github.com/python/cpython/issues/133779]: Reverts the change to generate different `pyconfig.h` files based on compiler settings, as it was frequently causing extension builds to break. In particular, the `Py_GIL_DISABLED` preprocessor variable must now always be defined explicitly when compiling for the experimental free-threaded runtime. The **`sysconfig.get_config_var()`** function can be used to determine whether the current runtime was compiled with that flag or not.
- gh-133626 [https://github.com/python/cpython/issues/133626]: Ensures packages are not accidentally bundled into the traditional installer.

Tools/Demos

- gh-134215 [https://github.com/python/cpython/issues/134215]: REPL import autocomplete only suggests private modules when explicitly specified.

Tests

- gh-133744 [https://github.com/python/cpython/issues/133744]: Fix multiprocessing interrupt test. Add an event to synchronize the parent process with the child process: wait until the child process starts sleeping. Patch by Victor Stinner.

- gh-133682 [https://github.com/python/cpython/issues/133682]: Fixed test case `test.test_annotationlib.TestStringFormat.test_displ` which ensures proper handling of complex data structures (lists, sets, dictionaries, and tuples) in string annotations.
- gh-133639 [https://github.com/python/cpython/issues/133639]: Fix `TestPyReplAutoindent.test_auto_indent_default()` doesn’t run `input_code`.

Security

- gh-133767 [https://github.com/python/cpython/issues/133767]: Fix use-after-free in the «unicode-escape» decoder with a non-«strict» error handler.
- gh-128840 [https://github.com/python/cpython/issues/128840]: Short-circuit the processing of long IPv6 addresses early in `ipaddress` to prevent excessive memory consumption and a minor denial-of-service.

Library

- gh-132710 [https://github.com/python/cpython/issues/132710]: If possible, ensure that `uuid.getnode()` returns the same result even across different processes. Previously, the result was constant only within the same process. Patch by Bénédikt Tran.
- gh-80334 [https://github.com/python/cpython/issues/80334]: `multiprocessing.freeze_support()` now checks for work on any «spawn» start method platform rather than only on Windows.
- gh-134582 [https://github.com/python/cpython/issues/134582]: Fix tokenize.untokenize() round-trip errors related to t-strings braces escaping
- gh-134546 [https://github.com/python/cpython/issues/134546]: Ensure `pdb` remote debugging script is readable by remote Python process.
- gh-134451 [https://github.com/python/cpython/issues/134451]: Converted `asyncio.tools.CycleFoundException` from dataclass to a regular exception type.

- gh-114177 [https://github.com/python/cpython/issues/114177]: Fix `asyncio` to not close subprocess pipes which would otherwise error out when the event loop is already closed.
- gh-90871 [https://github.com/python/cpython/issues/90871]: Fixed an off by one error concerning the backlog parameter in `create_unix_server()`. Contributed by Christian Harries.
- gh-134323 [https://github.com/python/cpython/issues/134323]: Fix the `threading.RLock.locked()` method.
- gh-86802 [https://github.com/python/cpython/issues/86802]: Fixed asyncio memory leak in cancelled shield tasks. For shielded tasks where the shield was cancelled, log potential exceptions through the exception handler. Contributed by Christian Harries.
- gh-134209 [https://github.com/python/cpython/issues/134209]: `curses`: The `curses.window.instr()` and `curses.window.getstr()` methods now allocate their internal buffer on the heap instead of the stack; in addition, the max buffer size is increased from 1023 to 2047.
- gh-134235 [https://github.com/python/cpython/issues/134235]: Updated tab completion on REPL to include builtin modules. Contributed by Tom Wang, Hunter Young
- gh-134152 [https://github.com/python/cpython/issues/134152]: Fixed `UnboundLocalError` that could occur during `email` header parsing if an expected trailing delimiter is missing in some contexts.
- gh-134168 [https://github.com/python/cpython/issues/134168]: `http.server`: Fix IPv6 address binding and `--directory` handling when using HTTPS.
- gh-62184 [https://github.com/python/cpython/issues/62184]: Remove import of C implementation of `io.FileIO` from Python implementation which has its own implementation
- gh-133982 [https://github.com/python/cpython/issues/133982]: Emit `RuntimeWarning` in the Python implementation of `io` when the file-like object is not closed explicitly in the presence of multiple I/O layers.
- gh-133890 [https://github.com/python/cpython/issues/133890]: The `tarfile` module now handles `UnicodeEncodeError` in the same way as `OSError` when cannot extract a member.
- gh-134097 [https://github.com/python/cpython/issues/134097]: Fix interaction of the new REPL and `-X showrefcount`

command line option.

- gh-133889 [https://github.com/python/cpython/issues/133889]: The generated directory listing page in `http.server.SimpleHTTPRequestHandler` now only shows the decoded path component of the requested URL, and not the query and fragment.
- gh-134098 [https://github.com/python/cpython/issues/134098]: Fix handling paths that end with a percent-encoded slash (`%2f` or `%2F`) in `http.server.SimpleHTTPRequestHandler`.
- gh-132124 [https://github.com/python/cpython/issues/132124]: On POSIX-compliant systems, `multiprocessing.util.get_temp_dir()` now ignores `TMPDIR` (and similar environment variables) if the path length of `AF_UNIX` socket files exceeds the platform-specific maximum length when using the forkserver start method. Patch by Bénédikt Tran.
- gh-134062 [https://github.com/python/cpython/issues/134062]: `ipaddress`: fix collisions in `__hash__()` for `IPv4Network` and `IPv6Network` objects.
- gh-133970 [https://github.com/python/cpython/issues/133970]: Make `string.templatelib.Template` and `string.templatelib.Interpolation` generic.
- gh-71253 [https://github.com/python/cpython/issues/71253]: Raise `ValueError` in `open()` if *opener* returns a negative file-descriptor in the Python implementation of `io` to match the C implementation.
- gh-133960 [https://github.com/python/cpython/issues/133960]: Simplify and improve `typing.evaluate_forward_ref()`. It now no longer raises errors on certain invalid types. In several situations, it is now able to evaluate forward references that were previously unsupported.
- gh-133925 [https://github.com/python/cpython/issues/133925]: Make the private class `typing._UnionGenericAlias` hashable.
- gh-133653 [https://github.com/python/cpython/issues/133653]: Fix `argparse.ArgumentParser` with the *formatter_class* argument. Fix TypeError when *formatter_class* is a custom subclass of `HelpFormatter`. Fix TypeError when *formatter_class* is not a subclass of `HelpFormatter` and non-standard *prefix_char* is used. Fix support of colorizing when

formatter_class is not a subclass of **`HelpFormatter`**.

- gh-132641 [https://github.com/python/cpython/issues/132641]: Fixed a race in **`functools.lru_cache()`** under free-threading.
- gh-133783 [https://github.com/python/cpython/issues/133783]: Fix bug with applying **`copy.replace()`** to **`ast`** objects. Attributes that default to `None` were incorrectly treated as required for manually created AST nodes.
- gh-133684 [https://github.com/python/cpython/issues/133684]: Fix bug where **`annotationlib.get_annotations()`** would return the wrong result for certain classes that are part of a class hierarchy where `from __future__ import annotations` is used.
- gh-77057 [https://github.com/python/cpython/issues/77057]: Fix handling of invalid markup declarations in **`html.parser.HTMLParser`**.
- gh-130328 [https://github.com/python/cpython/issues/130328]: Speedup pasting in `PyREPL` on Windows in a legacy console. Patch by Chris Eibl.
- gh-133701 [https://github.com/python/cpython/issues/133701]: Fix bug where **`typing.TypedDict`** classes defined under `from __future__ import annotations` and inheriting from another `TypedDict` had an incorrect `__annotations__` attribute.
- gh-133581 [https://github.com/python/cpython/issues/133581]: Improve unparsing of t-strings in **`ast.unparse()`** and `from __future__ import annotations`. Empty t-strings now round-trip correctly and formatting in interpolations is preserved. Patch by Jelle Zijlstra.
- gh-133551 [https://github.com/python/cpython/issues/133551]: Support t-strings (**PEP 750** [https://peps.python.org/pep-0750/]) in **`annotationlib`**. Patch by Jelle Zijlstra.
- gh-133439 [https://github.com/python/cpython/issues/133439]: Fix dot commands with trailing spaces are mistaken for multi-line SQL statements in the sqlite3 command-line interface.
- gh-132493 [https://github.com/python/cpython/issues/132493]: Avoid accessing `__annotations__` unnecessarily in **`inspect.signature()`**.
- gh-132876 [https://github.com/python/cpython/issues/132876]: `ldexp()` on Windows doesn't round subnormal results before Windows 11, but should. Python's **`math.ldexp()`** wrapper

now does round them, so results may change slightly, in rare cases of very small results, on Windows versions before 11.

- gh-133009 [https://github.com/python/cpython/issues/133009]: **`xml.etree.ElementTree`**: Fix a crash in **`Element.__deepcopy__`** when the element is concurrently mutated. Patch by Bénédikt Tran.
- gh-91555 [https://github.com/python/cpython/issues/91555]: Ignore log messages generated during handling of log messages, to avoid deadlock or infinite recursion. [NOTE: This change has since been reverted.]
- gh-125028 [https://github.com/python/cpython/issues/125028]: **`functools.Placeholder`** cannot be passed to **`functools.partial()`** as a keyword argument.
- gh-62824 [https://github.com/python/cpython/issues/62824]: Fix aliases for `iso8859_8` encoding. Patch by Dave Goncalves.
- gh-86155 [https://github.com/python/cpython/issues/86155]: **`html.parser.HTMLParser.close()`** no longer loses data when the `<script>` tag is not closed. Patch by Waylan Limberg.
- gh-69426 [https://github.com/python/cpython/issues/69426]: Fix **`html.parser.HTMLParser`** to not unescape character entities in attribute values if they are followed by an ASCII alphanumeric or an equals sign.
- bpo-28494 [https://bugs.python.org/issue?@action=redirect&bpo=28494]: Improve Zip file validation false positive rate in **`zipfile.is_zipfile()`**.
- gh-134381 [https://github.com/python/cpython/issues/134381]: Fix **`RuntimeError`** when using a not-started **`threading.Thread`** after calling **`os.fork()`**

Core and Builtins

- gh-127960 [https://github.com/python/cpython/issues/127960]: PyREPL interactive shell no longer starts with `__package__` and `__file__` global names set to `_pyrepl` package internals. Contributed by Yuichiro Tachibana.
- gh-130397 [https://github.com/python/cpython/issues/130397]: Remove special-casing for C stack depth limits for WASI. Due to WebAssembly's built-in stack protection this does not pose a security concern.

- gh-128066 [https://github.com/python/cpython/issues/128066]: Fixes an edge case where PyREPL improperly threw an error when Python is invoked on a read only filesystem while trying to write history file entries.
- gh-69605 [https://github.com/python/cpython/issues/69605]: When auto-completing an import in the REPL, finding no candidates now issues no suggestion, rather than suggestions from the current namespace.
- gh-134158 [https://github.com/python/cpython/issues/134158]: Fix coloring of double braces in f-strings and t-strings in the REPL.
- gh-134119 [https://github.com/python/cpython/issues/134119]: Fix crash when calling `next()` on an exhausted template string iterator. Patch by Jelle Zijlstra.
- gh-134100 [https://github.com/python/cpython/issues/134100]: Fix a use-after-free bug that occurs when an imported module isn't in `sys.modules` after its initial import. Patch by Nico-Posada.
- gh-133999 [https://github.com/python/cpython/issues/133999]: Fix `SyntaxError` regression in `except` parsing after gh-123440 [https://github.com/python/cpython/issues/123440].
- gh-133886 [https://github.com/python/cpython/issues/133886]: Fix `sys.remote_exec()` for non-ASCII paths in non-UTF-8 locales and non-UTF-8 paths in UTF-8 locales.
- gh-133703 [https://github.com/python/cpython/issues/133703]: Fix hashtable in dict can be bigger than intended in some situations.
- gh-133778 [https://github.com/python/cpython/issues/133778]: Fix bug where assigning to the `__annotations__` attributes of classes defined under `from __future__ import annotations` had no effect.
- gh-132762 [https://github.com/python/cpython/issues/132762]: `fromkeys()` no longer loops forever when adding a small set of keys to a large base dict. Patch by Angela Liss.
- gh-133541 [https://github.com/python/cpython/issues/133541]: Inconsistent indentation in user input crashed the new REPL when syntax highlighting was active. This is now fixed.
- gh-133516 [https://github.com/python/cpython/issues/133516]: Raise `ValueError` when constants `True`, `False` or `None` are used as an identifier after NFKC normalization.

- gh-132542 [https://github.com/python/cpython/issues/132542]: Update **`Thread.native_id`** after *fork(2) [https://manpages.debian.org/fork(2)]* to ensure accuracy. Patch by Noam Cohen.

C API

- gh-134144 [https://github.com/python/cpython/issues/134144]: Fix crash when calling **`Py_EndInterpreter()`** with a thread state that isn't the initial thread for the interpreter.

Build

- gh-134455 [https://github.com/python/cpython/issues/134455]: Fixed `build-details.json` generation to use the correct `c_api.headers` as defined in **PEP 739** [https://peps.python.org/pep-0739/], instead of `c_api.include`.
- gh-131769 [https://github.com/python/cpython/issues/131769]: Fix detecting when the build Python in a cross-build is a pydebug build.
- gh-117088 [https://github.com/python/cpython/issues/117088]: AIX linker don't support -h option, so avoid it through platform check

Python 3.14.0 beta 1

Release date: 2025-05-06

Windows

- gh-132930 [https://github.com/python/cpython/issues/132930]: Marks the installer for Windows as deprecated and updates documentation to cover the new Python install manager.
- gh-127405 [https://github.com/python/cpython/issues/127405]: Add `ABIFLAGS` to **`sysconfig.get_config_vars()`** on Windows. Patch by Xuehai Pan.

Tools/Demos

- gh-130453 [https://github.com/python/cpython/issues/130453]: Allow passing multiple keyword arguments with the same function name in **pygettext**.
- gh-130195 [https://github.com/python/cpython/issues/130195]: Add warning messages when **pygettext** unimplemented `-a/--extract-all` option is called.

Tests

- gh-133131 [https://github.com/python/cpython/issues/133131]: The iOS testbed will now select the most recently released «SE-class» device for testing if a device isn't explicitly specified.
- gh-91048 [https://github.com/python/cpython/issues/91048]: Add ability to externally inspect all pending asyncio tasks, even if no task is currently entered on the event loop.
- gh-109981 [https://github.com/python/cpython/issues/109981]: The test helper that counts the list of open file descriptors now uses the optimised `/dev/fd` approach on all Apple platforms, not just macOS. This avoids crashes caused by guarded file descriptors.
- gh-132678 [https://github.com/python/cpython/issues/132678]: Add `--prioritize` to `-m test`. This option allows the user to specify which selected tests should execute first, even if the order is otherwise randomized. This is particularly useful for tests that run the longest.
- gh-131290 [https://github.com/python/cpython/issues/131290]: Tests in `Lib/test` can now be correctly executed as standalone scripts.

Security

- gh-115322 [https://github.com/python/cpython/issues/115322]: The underlying extension modules behind **`readline`**:, **`subprocess`**, and **`ctypes`** now raise audit events on previously uncovered code paths that could lead to file system access related to C function calling and external binary execution. The `ctypes.call_function` audit hook has also been fixed to use an unsigned value for its `function pointer`.

Library

- gh-133490 [https://github.com/python/cpython/issues/133490]: Add color support to PDB in remote mode.
- gh-132493 [https://github.com/python/cpython/issues/132493]: Avoid eagerly evaluating annotations in functions decorated with **`reprlib.recursive_repr()`**.
- gh-130645 [https://github.com/python/cpython/issues/130645]: Add color to stdlib argparse CLIs. Patch by Hugo van Kemenade.
- gh-119180 [https://github.com/python/cpython/issues/119180]: Make **`annotationlib.get_annotations()`** succeed with the `FORWARDREF` format if evaluating the annotations throws an exception other than **`NameError`** or **`AttributeError`**.
- gh-133351 [https://github.com/python/cpython/issues/133351]: Fix remote PDB to correctly request tab completions for Python expressions from the server when completing a continuation line of a multi-line Python block.
- gh-133367 [https://github.com/python/cpython/issues/133367]: Add the `--feature-version`, `--optimize`, and `--show-empty` options to the **`ast`** command-line interface. Patch by Semyon Moroz.
- gh-133363 [https://github.com/python/cpython/issues/133363]: The **`cmd.Cmd`** class has been fixed to reliably call the `completedefault` method whenever the `do_shell` method is not defined and tab completion is requested for a line beginning with `!`.
- gh-113081 [https://github.com/python/cpython/issues/113081]: Highlight syntax on source code in **`pdb`**.
- gh-133349 [https://github.com/python/cpython/issues/133349]: Introduced auto-indent in **`pdb`** multi-line input.
- gh-133306 [https://github.com/python/cpython/issues/133306]: Use `\z` instead of `\Z` in **`fnmatch.translate()`** and

`glob.translate()`.

- gh-133306 [https://github.com/python/cpython/issues/133306]: Support `\z` as a synonym for `\Z` in **`regular expressions`**.
- gh-133300 [https://github.com/python/cpython/issues/133300]: Make **`argparse.ArgumentParser`**'s `suggest_on_error` a keyword-only parameter. Patch by Hugo van Kemenade.
- gh-133290 [https://github.com/python/cpython/issues/133290]: Fix attribute caching issue when setting **`ctypes._Pointer._type_`** in the undocumented and deprecated **`ctypes.SetPointerType()`** function and the undocumented **`set_type()`** method.
- gh-133223 [https://github.com/python/cpython/issues/133223]: When PDB is attached to a remote process, do a better job of intercepting Ctrl+C and forwarding it to the remote process.
- gh-133153 [https://github.com/python/cpython/issues/133153]: Do not complete **`pdb`** commands in `interact` mode of **`pdb`**.
- gh-133139 [https://github.com/python/cpython/issues/133139]: Add the **`curses.assume_default_colors()`** function, a refinement of the **`curses.use_default_colors()`** function which allows to change the color pair `0`.
- gh-133089 [https://github.com/python/cpython/issues/133089]: Use original timeout value for **`subprocess.TimeoutExpired`** when the func **`subprocess.run()`** is called with a timeout instead of sometimes a confusing partial remaining time out value used internally on the final `wait()`.
- gh-133036 [https://github.com/python/cpython/issues/133036]: **`codecs.open()`** is now deprecated. Use **`open()`** instead. Contributed by Inada Naoki.
- gh-132987 [https://github.com/python/cpython/issues/132987]: Many builtin and extension functions which accept an unsigned integer argument, now use **`__index__()`** if available.

- gh-124703 [https://github.com/python/cpython/issues/124703]: Set return code to `1` when aborting process from `pdb`.
- gh-133005 [https://github.com/python/cpython/issues/133005]: Support passing `preset` option to `tarfile.open()` when using `'w|xz'` mode.
- gh-115032 [https://github.com/python/cpython/issues/115032]: Support for custom logging handlers with the *strm* argument is deprecated and scheduled for removal in Python 3.16. Define handlers with the *stream* argument instead. Patch by Mariusz Felisiak.
- gh-132991 [https://github.com/python/cpython/issues/132991]: Add `socket.IP_FREEBIND` constant on Linux 2.4 and later.
- gh-132995 [https://github.com/python/cpython/issues/132995]: Bump the version of pip bundled in ensurepip to version 25.1.1
- gh-132933 [https://github.com/python/cpython/issues/132933]: The zipapp module now applies the filter when creating the list of files to add, rather than waiting until the file is being added to the archive.
- gh-121249 [https://github.com/python/cpython/issues/121249]: Always support the float complex and double complex C types in the `struct` module. Patch by Sergey B Kirpichev.
- gh-132915 [https://github.com/python/cpython/issues/132915]: `fcntl.fcntl()` and `fcntl.ioctl()` can now detect a buffer overflow and raise `SystemError`. The stack and memory can be corrupted in such case, so treat this error as fatal.
- gh-132017 [https://github.com/python/cpython/issues/132017]: Fix error when `pyrepl` is suspended, then resumed and terminated.
- gh-132893 [https://github.com/python/cpython/issues/132893]: Improved `statistics.NormalDist.cdf()` accuracy for inputs smaller than the mean.

- gh-130328 [https://github.com/python/cpython/issues/130328]: Speedup pasting in `PyREPL` on Windows. Fix by Chris Eibl.

- gh-132882 [https://github.com/python/cpython/issues/132882]: Fix copying of **`typing.Union`** objects containing objects that do not support the `|` operator.

- gh-93696 [https://github.com/python/cpython/issues/93696]: Fixed the breakpoint display error for frozen modules in **`pdb`**.

- gh-129965 [https://github.com/python/cpython/issues/129965]: Add MIME types for `.7z`, `.apk`, `.deb`, `.glb`, `.gltf`, `.gz`, `.m4v`, `.php`, `.rar`, `.rpm`, `.stl` and `.wmv`. Patch by Hugo van Kemenade.

- gh-132742 [https://github.com/python/cpython/issues/132742]: **`fcntl.fcntl()`** now supports arbitrary bytes-like objects, not only **`bytes`**. **`fcntl.ioctl()`** now automatically retries system calls failing with EINTR and releases the GIL during a system call even for large bytes-like object.

- gh-132451 [https://github.com/python/cpython/issues/132451]: The CLI for the PDB debugger now accepts a `-p PID` argument to allow attaching to a running process. The process must be running the same version of Python as the one running PDB.

- gh-125618 [https://github.com/python/cpython/issues/125618]: Add a *format* parameter to **`annotationlib.ForwardRef.evaluate()`**. Evaluating annotations in the `FORWARDREF` format now succeeds in more cases that would previously have raised an exception.

- gh-132805 [https://github.com/python/cpython/issues/132805]: Fix incorrect handling of nested non-constant values in the FORWARDREF format in **`annotationlib`**.

- gh-132734 [https://github.com/python/cpython/issues/132734]: Add new constants for Bluetooth **`sockets`**.

- gh-132673 [https://github.com/python/cpython/issues/132673]: Fix **`AssertionError`** raised on **`ctypes.Structure`** with `_align_ = 0` and `_fields_ = []`.

- gh-132578 [https://github.com/python/cpython/issues/132578]: Rename the `threading.Thread._handle` field to avoid shadowing methods defined on subclasses of `threading.Thread`.

- gh-132561 [https://github.com/python/cpython/issues/132561]: Fix the public `locked` method of `multiprocessing.SemLock` class. Also adding 2 tests for the derivated **`multiprocessing.Lock`** and **`multiprocessing.RLock`** classes.

- gh-121468 [https://github.com/python/cpython/issues/121468]: Add **`pdb.set_trace_async()`** function to support **`await`** statements in **`pdb`**.

- gh-132493 [https://github.com/python/cpython/issues/132493]: **`typing.Protocol`** now uses **`annotationlib.get_annotations()`** when checking whether or not an instance implements the protocol with **`isinstance()`**. This enables support for `isinstance` checks against classes with deferred annotations.

- gh-132536 [https://github.com/python/cpython/issues/132536]: Do not disable **`PY_THROW`** event in **`bdb`** because it can't be disabled.

- gh-132527 [https://github.com/python/cpython/issues/132527]: Include the valid typecode "w" in the error message when an invalid typecode is passed to **`array.array`**.

- gh-132099 [https://github.com/python/cpython/issues/132099]: The Bluetooth socket with the **`BTPROTO_HCI`** protocol on Linux now accepts an address in the format of an integer `device_id`, not only a tuple `(device_id,)`.

- gh-81793 [https://github.com/python/cpython/issues/81793]: Fix **`os.link()`** on platforms (like Linux) where the system **`link()`** function does not follow symlinks. On Linux, it now follows symlinks by default or if `follow_symlinks=True` is specified. On Windows, it now raises an error if `follow_symlinks=True` is passed. On macOS, it now raises an error if `follow_symlinks=False` is passed and the system **`linkat()`** function is not available at runtime.

- gh-132493 [https://github.com/python/cpython/issues/132493]: Support creation of **`typing.Protocol`** classes with annotations that cannot be resolved at class creation time.

- gh-132491 [https://github.com/python/cpython/issues/132491]: Rename `annotationlib.value_to_string` to **`annotationlib.type_repr()`** and provide better handling for function objects.

- gh-132426 [https://github.com/python/cpython/issues/132426]: Add **`annotationlib.get_annotate_from_class_namespace()`** as a helper for accessing annotations in metaclasses, and remove `annotationlib.get_annotate_function`.

- gh-70145 [https://github.com/python/cpython/issues/70145]: Add support for channels in Bluetooth HCI protocol (**`BTPROTO_HCI`**).

- gh-131913 [https://github.com/python/cpython/issues/131913]: Add a shortcut function **`multiprocessing.Process.interrupt()`** alongside the existing **`multiprocessing.Process.terminate()`** and **`multiprocessing.Process.kill()`** for an improved control over child process termination.

- gh-132439 [https://github.com/python/cpython/issues/132439]: Fix `PyREPL` on Windows: characters entered via AltGr are swallowed. Patch by Chris Eibl.

- gh-132429 [https://github.com/python/cpython/issues/132429]: Fix support of Bluetooth sockets on NetBSD and DragonFly BSD. Add support for *cid* and *bdaddr_type* in the BTPROTO_L2CAP address on FreeBSD. Return *cid* in `getsockname()` for BTPROTO_L2CAP if it is not zero.

- gh-132106 [https://github.com/python/cpython/issues/132106]:

`QueueListener.start` now raises a **`RuntimeError`** if the listener is already started.

- gh-132417 [https://github.com/python/cpython/issues/132417]: Fix a `NULL` pointer dereference when a C function called using **`ctypes`** with `restype` **`py_object`** returns `NULL`.

- gh-132385 [https://github.com/python/cpython/issues/132385]: Fix instance error suggestions trigger potential exceptions in **`object.__getattr__()`** in **`traceback`**.

- gh-125866 [https://github.com/python/cpython/issues/125866]: Add optional *add_scheme* argument to **`urllib.request.pathname2url()`**; when set to true, a complete URL is returned. Likewise add optional *require_scheme* argument to **`url2pathname()`**; when set to true, a complete URL is accepted.

- gh-132308 [https://github.com/python/cpython/issues/132308]: A **`traceback.TracebackException`** now correctly renders the `__context__` and `__cause__` attributes from falsey **`Exception`**, and the `exceptions` attribute from falsey **`ExceptionGroup`**.

- gh-130645 [https://github.com/python/cpython/issues/130645]: Add colour to **`argparse`** help output. Patch by Hugo van Kemenade.

- gh-127495 [https://github.com/python/cpython/issues/127495]: In PyREPL, append a new entry to the `PYTHON_HISTORY` file *after* every statement. This should preserve command-line history after interpreter is terminated. Patch by Sergey B Kirpichev.

- gh-129463 [https://github.com/python/cpython/issues/129463]: Comparison of **`annotationlib.ForwardRef`** objects no longer uses the internal `__code__` and `__ast_node__` attributes, which are used as caches.

- gh-132250 [https://github.com/python/cpython/issues/132250]: Fixed the **`SystemError`** in **`cProfile`** when locating the actual C function of a method raises an exception.

- gh-132064 [https://github.com/python/cpython/issues/132064]: **`annotationlib.get_annotations()`** now uses the `__annotate__` attribute if it is present, even if `__annotations__` is not present. Additionally, the function now raises a **`TypeError`** if it is passed an object that does not have any annotatins.

- gh-130664 [https://github.com/python/cpython/issues/130664]: Support the `'_'` digit separator in formatting of the integral part of **`Decimal`**'s. Patch by Sergey B Kirpichev.

- gh-131952 [https://github.com/python/cpython/issues/131952]: Add color output to the **json** CLI. Patch by Tomas Roun.

- gh-132063 [https://github.com/python/cpython/issues/132063]: Prevent exceptions that evaluate as falsey (namely, when their `__bool__` method returns `False` or their `__len__` method returns 0) from being ignored by **`concurrent.futures.ProcessPoolExecutor`** and **`concurrent.futures.ThreadPoolExecutor`**.

- gh-132106 [https://github.com/python/cpython/issues/132106]: **`logging.handlers.QueueListener`** now implements the context manager protocol, allowing it to be used in a **`with`** statement.

- gh-132054 [https://github.com/python/cpython/issues/132054]: The `application/yaml` mime type (**RFC 9512** [https://datatracker.ietf.org/doc/html/rfc9512.html]) is now supported by **`mimetypes`**. Patch by Sasha «Nelie» Chernykh and Hugo van Kemenade.

- gh-119605 [https://github.com/python/cpython/issues/119605]: Respect `follow_wrapped` for **`__init__()`** and **`__new__()`** methods when getting the class signature for a class with **`inspect.signature()`**. Preserve class signature after wrapping with **`warnings.deprecated()`**. Patch by Xuehai Pan.

- gh-118761 [https://github.com/python/cpython/issues/118761]: Improve import times by up to 33x for the `shlex` module, and improve the performance of `shlex.quote()` by up to 12x. Patch by Adam Turner.

- gh-85302 [https://github.com/python/cpython/issues/85302]: Add support for `BTPROTO_SCO` in sockets on FreeBSD.

- gh-131757 [https://github.com/python/cpython/issues/131757]: Make `functools.lru_cache()` call the cached function unlocked to allow concurrency.

- gh-131423 [https://github.com/python/cpython/issues/131423]: `ssl` can show descriptions for errors added in OpenSSL 3.4.1. Patch by Bénédikt Tran.

- gh-131434 [https://github.com/python/cpython/issues/131434]: Improve error reporting for incorrect format in `time.strptime()`.

- gh-131524 [https://github.com/python/cpython/issues/131524]: Add help message to `platform` command-line interface. Contributed by Harry Lees.

- gh-100926 [https://github.com/python/cpython/issues/100926]: Move `ctypes.POINTER()` types cache from a global internal cache (`_pointer_type_cache`) to the `ctypes._CData.__pointer_type__` attribute of the corresponding `ctypes` types. This will stop the cache from growing without limits in some situations.

- gh-85702 [https://github.com/python/cpython/issues/85702]: If `zoneinfo._common.load_tzdata` is given a package without a resource a `ZoneInfoNotFoundError` is raised rather than a `IsADirectoryError`.

- gh-123471 [https://github.com/python/cpython/issues/123471]: Make concurrent iterations over `itertools.repeat` safe under free-threading.

- gh-131127 [https://github.com/python/cpython/issues/131127]:

Systems using LibreSSL now successfully build.

- gh-89157 [https://github.com/python/cpython/issues/89157]: Make the pure Python implementation of `datetime.date.fromisoformat()`, only accept ASCII strings for consistency with the C implementation.

- gh-130941 [https://github.com/python/cpython/issues/130941]: Fix `configparser.ConfigParser` parsing empty interpolation with `allow_no_value` set to `True`.

- gh-110067 [https://github.com/python/cpython/issues/110067]: Make `heapq` max-heap functions `heapq.heapify_max()`, `heapq.heappush_max()`, `heapq.heappop_max()`, and `heapq.heapreplace_max()` public. Previous underscored naming is kept for backwards compatibility. Additionally, the missing function `heapq.heappushpop_max()` has been added to both the C and Python implementations.

- gh-129098 [https://github.com/python/cpython/issues/129098]: Fix REPL traceback reporting when using `compile()` with an inexisting file. Patch by Bénédikt Tran.

- gh-130631 [https://github.com/python/cpython/issues/130631]: `http.cookiejar.join_header_words()` is now more similar to the original Perl version. It now quotes the same set of characters and always quote values that end with `"\n"`.

- gh-130482 [https://github.com/python/cpython/issues/130482]: Add ability to specify name for `tkinter.OptionMenu` and `tkinter.ttk.OptionMenu`.

- gh-77065 [https://github.com/python/cpython/issues/77065]: Add keyword-only optional argument *echo_char* for `getpass.getpass()` for optional visual keyboard feedback support. Patch by Semyon Moroz.

- gh-130317 [https://github.com/python/cpython/issues/130317]: Fix `PyFloat_Pack2()` and `PyFloat_Unpack2()` for NaN’s with payload. This corrects round-trip for `struct.unpack()` and `struct.pack()` in case of the

IEEE 754 binary16 «half precision» type. Patch by Sergey B Kirpichev.

- gh-130402 [https://github.com/python/cpython/issues/130402]: Joining running daemon threads during interpreter shutdown now raises `PythonFinalizationError`.

- gh-130167 [https://github.com/python/cpython/issues/130167]: Improve speed of `difflib.IS_LINE_JUNK()`. Patch by Semyon Moroz.

- gh-101410 [https://github.com/python/cpython/issues/101410]: Added more detailed messages for domain errors in the `math` module.

- gh-128384 [https://github.com/python/cpython/issues/128384]: Make `warnings.catch_warnings` use a context variable for holding the warning filtering state if the `sys.flags.context_aware_warnings` flag is set to true. This makes using the context manager thread-safe in multi-threaded programs. The flag is true by default in free-threaded builds and is otherwise false. The value of the flag can be overridden by the the `-X context_aware_warnings` command-line option or by the `PYTHON_CONTEXT_AWARE_WARNINGS` environment variable.

- gh-129719 [https://github.com/python/cpython/issues/129719]: Fix missing `socket.CAN_RAW_ERR_FILTER` constant in the socket module on Linux systems. It was missing since Python 3.11.

- gh-129027 [https://github.com/python/cpython/issues/129027]: Raise `DeprecationWarning` for `sys._clear_type_cache()`. This function was deprecated in Python 3.13 but it didn’t raise a runtime warning.

- gh-128307 [https://github.com/python/cpython/issues/128307]: Add `eager_start` keyword argument to `asyncio.loop.create_task()`

- gh-127604 [https://github.com/python/cpython/issues/127604]: Add

support for printing the C stack trace on systems that support it via `faulthandler.dump_c_stack()` or via the *c_stack* argument in `faulthandler.enable()`.

- gh-127385 [https://github.com/python/cpython/issues/127385]: Add the `F_DUPFD_QUERY` constant to the `fcntl` module.

- gh-126838 [https://github.com/python/cpython/issues/126838]: Add *resolve_host* keyword-only parameter to `urllib.request.url2pathname()`, and fix handling of file URLs with authorities.

- gh-82129 [https://github.com/python/cpython/issues/82129]: Fix `NameError` when calling `typing.get_type_hints()` on a `dataclasses.dataclass()` created by `dataclasses.make_dataclass()` with un-annotated fields.

- gh-122559 [https://github.com/python/cpython/issues/122559]: Remove `__reduce__()` and `__reduce_ex__()` methods that always raise `TypeError` in the C implementation of `io.FileIO`, `io.BufferedReader`, `io.BufferedWriter` and `io.BufferedRandom` and replace them with default `__getstate__()` methods that raise `TypeError`. This restores fine details of behavior of Python 3.11 and older versions.

- gh-122179 [https://github.com/python/cpython/issues/122179]: `hashlib.file_digest()` now raises `BlockingIOError` when no data is available during non-blocking I/O. Before, it added spurious null bytes to the digest.

- gh-53032 [https://github.com/python/cpython/issues/53032]: Expose `decimal.IEEEContext()` to support creation of contexts corresponding to the IEEE 754 (2008) decimal interchange formats. Patch by Sergey B Kirpichev.

- gh-120220 [https://github.com/python/cpython/issues/120220]: Deprecate the `tkinter.Variable` methods `trace_variable()`, `trace_vdelete()` and `trace_vinfo()`. Methods `trace_add()`, `trace_remove()` and `trace_info()` can be used instead.

- gh-113539 [https://github.com/python/cpython/issues/113539]: `webbrowser`: Names in the `BROWSER` environment variable can now refer to already registered web browsers, instead of always generating a new browser command.

  This makes it possible to set `BROWSER` to the value of one of the supported browsers on macOS.

- bpo-44172 [https://bugs.python.org/issue?@action=redirect&bpo=44172]: Keep a reference to original `curses` windows in subwindows so that the original window does not get deleted before subwindows.

- gh-75223 [https://github.com/python/cpython/issues/75223]: Deprecate undotted extensions in `mimetypes.MimeTypes.add_type()`. Patch by Hugo van Kemenade.

IDLE

- gh-112936 [https://github.com/python/cpython/issues/112936]: fix IDLE: no Shell menu item in single-process mode.

Documentation

- gh-107006 [https://github.com/python/cpython/issues/107006]: Move documentation and example code for `threading.local` from its docstring to the official docs.
- gh-125142 [https://github.com/python/cpython/issues/125142]: As part of the builtin help intro text, show the keyboard shortcuts for the new, non-basic REPL (F1, F2, and F3).

Core and Builtins

- gh-133336 [https://github.com/python/cpython/issues/133336]: `-J` is no longer reserved for use by Jython. Patch by Adam Turner.
- gh-133261 [https://github.com/python/cpython/issues/133261]: Fix bug where the cycle GC could untrack objects in the trashcan

because they looked like they were immortal. When objects are added to the trashcan, we take care to ensure they keep a mortal reference count.

- gh-133346 [https://github.com/python/cpython/issues/133346]: Added experimental color theming support to the `_colorize` module.
- gh-132917 [https://github.com/python/cpython/issues/132917]: For the free-threaded build, check the process memory usage increase before triggering a full automatic garbage collection. If the memory used has not increased 10% since the last collection then defer it.
- gh-91048 [https://github.com/python/cpython/issues/91048]: Add a new `python -m asyncio ps PID` command-line interface to inspect asyncio tasks in a running Python process. Displays a flat table of await relationships. A variant showing a tree view is also available as `python -m asyncio pstree PID`. Both are useful for debugging async code. Patch by Pablo Galindo, Łukasz Langa, Yury Selivanov, and Marta Gomez Macias.
- gh-133304 [https://github.com/python/cpython/issues/133304]: Workaround NaN's «canonicalization» in **`PyFloat_Pack4()`** and **`PyFloat_Unpack4()`** on RISC-V.
- gh-133197 [https://github.com/python/cpython/issues/133197]: Improve **`SyntaxError`** error messages for incompatible string / bytes prefixes.
- gh-133231 [https://github.com/python/cpython/issues/133231]: Add new utilities of observing JIT compilation: **`sys._jit.is_available()`**, **`sys._jit.is_enabled()`**, and **`sys._jit.is_active()`**.

Library

- gh-133194 [https://github.com/python/cpython/issues/133194]: **`ast.parse()`** will no longer parse new **PEP 758** [https://peps.python.org/pep-0758/] syntax with older *feature_version* passed.

Core and Builtins

- gh-131798 [https://github.com/python/cpython/issues/131798]: Split

`CALL_LEN` into several uops allowing the JIT to remove them when optimizing. Patch by Diego Russo.

- gh-131798 [https://github.com/python/cpython/issues/131798]: Use `sym_new_type` instead of `sym_new_not_null` for _BUILD_STRING, _BUILD_SET
- gh-132942 [https://github.com/python/cpython/issues/132942]: Fix two races in the type lookup cache. This affected the free-threaded build and could cause crashes (apparently quite difficult to trigger).
- gh-131798 [https://github.com/python/cpython/issues/131798]: Propagate the return type of `_BINARY_OP_SUBSCR_TUPLE_INT` in JIT. Patch by Tomas Roun
- gh-132952 [https://github.com/python/cpython/issues/132952]: Speed up startup with the `-S` argument by importing the private `_io` module instead of **`io`**. This fixes a performance regression introduced earlier in Python 3.14 development and restores performance to the level of Python 3.13.
- gh-131798 [https://github.com/python/cpython/issues/131798]: Allow the JIT to remove int guards after `_CALL_LEN` by setting the return type to int. Patch by Diego Russo
- gh-131798 [https://github.com/python/cpython/issues/131798]: Split `CALL_TUPLE_1` into several uops allowing the JIT to remove some of them. Patch by Tomas Roun
- gh-131798 [https://github.com/python/cpython/issues/131798]: Split `CALL_STR_1` into several uops allowing the JIT to remove some of them. Patch by Tomas Roun
- gh-132825 [https://github.com/python/cpython/issues/132825]: Enhance unhashable key/element error messages for **`dict`** and **`set`**. Patch by Victor Stinner.
- gh-131591 [https://github.com/python/cpython/issues/131591]: Reset any **PEP 768** [https://peps.python.org/pep-0768/] remote debugging pending call in children after **`os.fork()`** calls.
- gh-132713 [https://github.com/python/cpython/issues/132713]: Fix `repr(list)` race condition: hold a strong reference to the item while calling `repr(item)`. Patch by Victor Stinner.
- gh-132661 [https://github.com/python/cpython/issues/132661]: Implement **PEP 750** [https://peps.python.org/pep-0750/] (Template Strings). Add new syntax for t-strings and implement new internal **`string.templatelib.Template`** and

`string.templatelib.Interpolation` types.

- gh-132479 [https://github.com/python/cpython/issues/132479]: Fix compiler crash in certain circumstances where multiple module-level annotations include comprehensions and other nested scopes.
- gh-132747 [https://github.com/python/cpython/issues/132747]: Fix a crash when calling **`__get__()`** of a method with a **`None`** second argument.
- gh-132744 [https://github.com/python/cpython/issues/132744]: Certain calls now check for runaway recursion and respect the system recursion limit.
- gh-132449 [https://github.com/python/cpython/issues/132449]: Syntax errors that look like misspellings of Python keywords now provide a helpful fix suggestion for the typo. Contributed by Pablo Galindo Salgado.

Library

- gh-132737 [https://github.com/python/cpython/issues/132737]: Support profiling code that requires `__main__`, such as **`pickle`**.

Core and Builtins

- gh-132639 [https://github.com/python/cpython/issues/132639]: Added **`PyLong_AsNativeBytes()`**, **`PyLong_FromNativeBytes()`** and **`PyLong_FromUnsignedNativeBytes()`** to the limited C API.
- gh-100239 [https://github.com/python/cpython/issues/100239]: Add specialisation for `BINARY_OP/SUBSCR` on list and slice.
- gh-132508 [https://github.com/python/cpython/issues/132508]: Uses tagged integers on the evaluation stack to represent the instruction offsets when reraising an exception. This avoids the need to box the integer which could fail in low memory conditions.
- gh-124476 [https://github.com/python/cpython/issues/124476]: Fix decoding from the locale encoding in the C.UTF-8 locale.
- gh-131927 [https://github.com/python/cpython/issues/131927]: Compiler warnings originating from the same module and

line number are now only emitted once, matching the behaviour of warnings emitted from user code. This can also be configured with **`warnings`** filters.

- gh-132457 [https://github.com/python/cpython/issues/132457]: Make **`staticmethod()`** and **`classmethod()`** generic.
- gh-131798 [https://github.com/python/cpython/issues/131798]: Use `sym_new_type` instead of `sym_new_not_null` for _BUILD_LIST, _BUILD_SET, _BUILD_MAP
- gh-131798 [https://github.com/python/cpython/issues/131798]: Split `CALL_TYPE_1` into several uops allowing the JIT to remove some of them.
- gh-132386 [https://github.com/python/cpython/issues/132386]: Fix crash when passing a dict subclass as the `globals` parameter to **`exec()`**.
- gh-127682 [https://github.com/python/cpython/issues/127682]: No longer call `__iter__` twice when creating and executing a generator expression. Creating a generator expression from a non-interable will raise only when the generator expression is executed. This brings the behavior of generator expressions in line with other generators.
- gh-132261 [https://github.com/python/cpython/issues/132261]: The internal storage for annotations and annotate functions on classes now uses different keys in the class dictionary. This eliminates various edge cases where access to the `__annotate__` and `__annotations__` attributes would behave unpredictably.
- gh-132284 [https://github.com/python/cpython/issues/132284]: Don’t wrap base `PyCFunction` slots on class creation if not overridden.
- gh-130415 [https://github.com/python/cpython/issues/130415]: Improve the JIT’s ability to remove unused constant and local variable loads, and fix an issue where deallocating unused values could cause JIT code to crash or behave incorrectly.
- gh-126703 [https://github.com/python/cpython/issues/126703]: Fix possible use after free in cases where a method’s definition has the same lifetime as its `self`.
- gh-132286 [https://github.com/python/cpython/issues/132286]: Fix that **`type.__annotate__`** was not deleted, when **`type.__annotations__`** was deleted.
- gh-131798 [https://github.com/python/cpython/issues/131798]: Allow

the JIT to remove an extra `_TO_BOOL_BOOL` instruction after `_CONTAINS_OP_DICT` by setting the return type to bool.

- gh-124715 [https://github.com/python/cpython/issues/124715]: Prevents against stack overflows when calling **`Py_DECREF()`**. Third-party extension objects no longer need to use the «trashcan» mechanism, as protection is now built into the **`Py_DECREF()`** macro.
- gh-131798 [https://github.com/python/cpython/issues/131798]: Allow the JIT compiler to remove some type checks for operations on lists, tuples, dictionaries, and sets.
- gh-128398 [https://github.com/python/cpython/issues/128398]: Improve error message when an object supporting the synchronous (resp. asynchronous) context manager protocol is entered using **`async with`** (resp. **`with`**) instead of **`with`** (resp. **`async with`**). Patch by Bénédikt Tran.
- gh-131798 [https://github.com/python/cpython/issues/131798]: Allow the JIT to remove unicode guards after `_BINARY_OP_SUBSCR_STR_INT` by setting the return type to string.
- gh-131878 [https://github.com/python/cpython/issues/131878]: Handle uncaught exceptions in the main input loop for the new REPL.
- gh-131878 [https://github.com/python/cpython/issues/131878]: Fix support of unicode characters with two or more codepoints on Windows in the new REPL.
- gh-126835 [https://github.com/python/cpython/issues/126835]: Move constant folding to the peephole optimizer. Rename AST optimization related files (Python/ast_opt.c -> Python/ast_preprocess.c), structs (_PyASTOptimizeState -> _PyASTPreprocessState) and functions (_PyAST_Optimize -> _PyAST_Preprocess, _PyCompile_AstOptimize -> _PyCompile_AstPreprocess).
- gh-114809 [https://github.com/python/cpython/issues/114809]: Add support for macOS multi-arch builds with the JIT enabled
- gh-131507 [https://github.com/python/cpython/issues/131507]: PyREPL now supports syntax highlighting. Contributed by Łukasz Langa.
- gh-130907 [https://github.com/python/cpython/issues/130907]: If the `__annotations__` of a module object are accessed while the module is executing, return the annotations that have

been defined so far, without caching them.

- gh-130104 [https://github.com/python/cpython/issues/130104]: Three-argument `pow()` now try calling `__rpow__()` if necessary. Previously it was only called in two-argument `pow()` and the binary power operator.
- gh-130070 [https://github.com/python/cpython/issues/130070]: Fixed an assertion error for `exec()` passed a string `source` and a non-`None` `closure`. Patch by Bartosz Sławecki.
- gh-129958 [https://github.com/python/cpython/issues/129958]: Fix a bug that was allowing newlines inconsistently in format specifiers for single-quoted f-strings. Patch by Pablo Galindo.
- gh-129858 [https://github.com/python/cpython/issues/129858]: `elif` statements that follow an `else` block now have a specific error message.
- gh-69605 [https://github.com/python/cpython/issues/69605]: Add module autocomplete to PyREPL.
- gh-128555 [https://github.com/python/cpython/issues/128555]: Add the `sys.flags.thread_inherit_context` flag.
  - This flag is set to true by default on the free-threaded build and false otherwise. If the flag is true, starting a new thread using `threading.Thread` will, by default, use a copy of the `contextvars.Context` from the caller of `threading.Thread.start()` rather than using an empty context.
  - Add the `-X thread_inherit_context` command-line option and `PYTHON_THREAD_INHERIT_CONTEXT` environment variable, which set the `thread_inherit_context` flag.
  - Add the `context` keyword parameter to `Thread`. It can be used to explicitly pass a context value to be used by a new thread.
  - Make the `_contextvars` module built-in.
- gh-123539 [https://github.com/python/cpython/issues/123539]: Improve `SyntaxError` message for using `import ... as` and `from ... import ... as` with not a name.
- gh-102567 [https://github.com/python/cpython/issues/102567]: `-X importtime` now accepts value `2`, which indicates that an `importtime` entry should also be printed if an imported module has already been loaded. Patch by Noah Kim and Adam Turner.

- gh-116436 [https://github.com/python/cpython/issues/116436]: Improve error message when `TypeError` occurs during `dict.update()`
- gh-103997 [https://github.com/python/cpython/issues/103997]: String arguments passed to «-c» are now automatically dedented as if by `textwrap.dedent()`. This allows «python -c» invocations to be indented in shell scripts without causing indentation errors. (Patch by Jon Crall and Steven Sun)

Library

- gh-89562 [https://github.com/python/cpython/issues/89562]: Remove `hostflags` member from `PySSLContext` struct.

C API

- gh-133166 [https://github.com/python/cpython/issues/133166]: Fix regression where `PyType_GetModuleByDef()` returns NULL without setting `TypeError` when a static type is passed.
- gh-133164 [https://github.com/python/cpython/issues/133164]: Add `PyUnstable_Object_IsUniqueReferencedTemporary()` function for determining if an object exists as a unique temporary variable on the interpreter's stack. This is a replacement for some cases where checking that `Py_REFCNT()` is one is no longer sufficient to determine if it's safe to modify a Python object in-place with no visible side effects.
- gh-133140 [https://github.com/python/cpython/issues/133140]: Add `PyUnstable_Object_IsUniquelyReferenced()` as a replacement for `Py_REFNCT(op) == 1` on free threaded builds of Python.
- gh-131747 [https://github.com/python/cpython/issues/131747]: On non-Windows platforms, deprecate using `ctypes.Structure._pack_` to use a Windows-compatible layout on non-Windows platforms. The layout should be specified explicitly by setting `ctypes.Structure._layout_` to `'ms'`.
- gh-128972 [https://github.com/python/cpython/issues/128972]: For non-free-threaded builds, the memory layout of

`PyASCIIObject` is reverted to match Python 3.13. (Note that the structure is not part of stable ABI and so its memory layout is *guaranteed* to remain stable.)

- gh-133079 [https://github.com/python/cpython/issues/133079]: The undocumented APIs `Py_C_RECURSION_LIMIT` and `PyThreadState.c_recursion_remaining`, added in 3.13, are removed without a deprecation period.
- gh-132987 [https://github.com/python/cpython/issues/132987]: The `k` and `K` formats in `PyArg_Parse()` now support the `__index__()` special method, like all other integer formats.
- gh-132909 [https://github.com/python/cpython/issues/132909]: Fix an overflow when handling the K format in `Py_BuildValue()`. Patch by Bénédikt Tran.
- gh-132798 [https://github.com/python/cpython/issues/132798]: Deprecated and undocumented functions `PyUnicode_AsEncodedObject()`, `PyUnicode_AsDecodedObject()`, `PyUnicode_AsEncodedUnicode()` and `PyUnicode_AsDecodedUnicode()` are scheduled for removal in 3.15.
- gh-132470 [https://github.com/python/cpython/issues/132470]: Creating a `ctypes.CField` with a *byte_size* that does not match the actual type size now raises a `ValueError` instead of crashing the interpreter.
- gh-112068 [https://github.com/python/cpython/issues/112068]: [Reverted in gh-136991 [https://github.com/python/cpython/issues/136991]] Add support of nullable arguments in `PyArg_Parse()` and similar functions. Adding `?` after any format unit makes `None` be accepted as a value.
- gh-50333 [https://github.com/python/cpython/issues/50333]: Non-tuple sequences are deprecated as argument for the `(items)` format unit in `PyArg_ParseTuple()` and other argument parsing functions if *items* contains format units which store a borrowed buffer or a borrowed reference.

Build

- gh-113464 [https://github.com/python/cpython/issues/113464]: Use the cpython-bin-deps «externals» repository for Windows LLVM dependency management. Installing LLVM manually is no longer necessary for Windows JIT builds.

- gh-133183 [https://github.com/python/cpython/issues/133183]: iOS compiler shims now include `IPHONEOS_DEPLOYMENT_TARGET` in target triples, ensuring that SDK version minimums are honored.
- gh-133167 [https://github.com/python/cpython/issues/133167]: Fix compilation process with `--enable-optimizations` and `--without-docstrings`.
- gh-133171 [https://github.com/python/cpython/issues/133171]: Since free-threaded builds do not support the experimental JIT compiler, prevent these configurations from being combined.
- gh-132758 [https://github.com/python/cpython/issues/132758]: Fix building with tail call interpreter and pystats.
- gh-132649 [https://github.com/python/cpython/issues/132649]: The `PClayout` script now allows passing `--include-tcltk` on Windows ARM64.
- gh-132257 [https://github.com/python/cpython/issues/132257]: Change the default LTO flags on GCC to not pass `-flto-partition=none`, and allow parallelization of LTO. For newer GNU makes and GCC, this has a multiple factor speedup for LTO build times, with no noticeable loss in performance.
- gh-132026 [https://github.com/python/cpython/issues/132026]: Fix use of undefined identifiers in platform triplet detection on MIPS Linux platforms.

Python 3.14.0 alpha 7

Release date: 2025-04-08

macOS

- gh-124111 [https://github.com/python/cpython/issues/124111]: Update macOS installer to use Tcl/Tk 8.6.16.
- gh-131423 [https://github.com/python/cpython/issues/131423]: Update macOS installer to use OpenSSL 3.0.16. Patch by Bénédikt Tran.
- gh-131025 [https://github.com/python/cpython/issues/131025]: Update macOS installer to ship with SQLite 3.49.1.

Windows

- gh-131423 [https://github.com/python/cpython/issues/131423]: Update bundled version of OpenSSL to 3.0.16. The new build also disables uplink support, which may be relevant to embedders but has no impact on normal use.
- gh-131453 [https://github.com/python/cpython/issues/131453]: Some **`SND_*`** and **`MB_*`** constants are added to **`winsound`**.
- gh-91349 [https://github.com/python/cpython/issues/91349]: Replaces our copy of `zlib` with `zlib-ng`, for performance improvements in **`zlib`**.
- gh-131025 [https://github.com/python/cpython/issues/131025]: Update Windows installer to ship with SQLite 3.49.1.

Tools/Demos

- gh-132121 [https://github.com/python/cpython/issues/132121]: Always escape non-printable Unicode characters in **pygettext**.
- gh-131852 [https://github.com/python/cpython/issues/131852]: **msgfmt** no longer adds the `POT-Creation-Date` to generated `.mo` files for consistency with GNU `msgfmt`.

Tests

- gh-131277 [https://github.com/python/cpython/issues/131277]: Allow to unset one or more environment variables at once via **`EnvironmentVarGuard.unset()`**. Patch by Bénédikt Tran.
- gh-131050 [https://github.com/python/cpython/issues/131050]: `test_ssl.test_dh_params` is skipped if the underlying TLS library does not support finite-field ephemeral Diffie-Hellman.

Security

- gh-131809 [https://github.com/python/cpython/issues/131809]: Update bundled libexpat to 2.7.1
- gh-131261 [https://github.com/python/cpython/issues/131261]: Upgrade to libexpat 2.7.0

- gh-121284 [https://github.com/python/cpython/issues/121284]: Fix bug in the folding of rfc2047 encoded-words when flattening an email message using a modern email policy. Previously when an encoded-word was too long for a line, it would be decoded, split across lines, and re-encoded. But commas and other special characters in the original text could be left unencoded and unquoted. This could theoretically be used to spoof header lines using a carefully constructed encoded-word if the resulting rendered email was transmitted or re-parsed.

Library

- gh-132174 [https://github.com/python/cpython/issues/132174]: Fix function name in error message of `_interpreters.run_string`.
- gh-132171 [https://github.com/python/cpython/issues/132171]: Fix crash of `_interpreters.run_string` on string subclasses.
- gh-129204 [https://github.com/python/cpython/issues/129204]: Introduce new `_PYTHON_SUBPROCESS_USE_POSIX_SPAWN` environment variable knob in **`subprocess`** to control the use of **`os.posix_spawn()`**.
- gh-132159 [https://github.com/python/cpython/issues/132159]: Do not shadow user arguments in generated **`__new__()`** by decorator **`warnings.deprecated`**. Patch by Xuehai Pan.
- gh-132168 [https://github.com/python/cpython/issues/132168]: The **`ctypes.py_object`** type now supports subscription, making it a generic type.
- gh-84481 [https://github.com/python/cpython/issues/84481]: Add the **`zipfile.ZipFile.data_offset`** attribute, which stores the offset to the beginning of ZIP data in a file when available. When the **`zipfile.ZipFile`** is opened in either mode `'w'` or `'x'` and the underlying file does not support `tell()`, the value will be `None` instead.
- gh-132075 [https://github.com/python/cpython/issues/132075]: Fix possible use of **`socket`** address structures with uninitialized members. Now all structure members are initialized with zeroes by default.
- gh-118761 [https://github.com/python/cpython/issues/118761]:

Improve import times by up to 27x for the **`string`** module. Patch by Adam Turner.

- gh-125434 [https://github.com/python/cpython/issues/125434]: Display thread name in **`faulthandler`**. Patch by Victor Stinner.
- gh-132002 [https://github.com/python/cpython/issues/132002]: Fix crash when deallocating **`contextvars.ContextVar`** with weird unahashable string names.
- gh-131938 [https://github.com/python/cpython/issues/131938]: **`xml.etree.ElementTree`**: update the error message when an element to remove via **`Element.remove`** is not found. Patch by Bénédikt Tran.
- gh-115942 [https://github.com/python/cpython/issues/115942]: Add **`threading.RLock.locked()`**, **`multiprocessing.Lock.locked()`**, **`multiprocessing.RLock.locked()`**, and allow **`multiprocessing.managers.SyncManager.Lock()`** and **`multiprocessing.managers.SyncManager.RLock()`** to proxy `locked()` call.
- gh-131974 [https://github.com/python/cpython/issues/131974]: Fix several thread-safety issues in **`ctypes`** on the free threaded build.
- gh-118761 [https://github.com/python/cpython/issues/118761]: Improve the import time of the **`ast`** module by extracting the **`unparse()`** function to a helper module.
- gh-107369 [https://github.com/python/cpython/issues/107369]: Improved performance of **`textwrap.indent()`** by an average of ~1.3x. Patch by Adam Turner.
- gh-131792 [https://github.com/python/cpython/issues/131792]: Improved performance of **`textwrap.dedent()`** by an average of ~2.4x, (with improvements of up to 4x for large inputs), and fixed a bug where blank lines with whitespace characters other than space or horizontal tab were not normalised to the newline. Patch by Adam Turner, Marius Juston, and Pieter Eendebak.
- gh-131668 [https://github.com/python/cpython/issues/131668]: **`socket`**: Fix code parsing AF_BLUETOOTH socket addresses.
- gh-60115 [https://github.com/python/cpython/issues/60115]: Support frozen modules for **`linecache.getline()`**.

- gh-131492 [https://github.com/python/cpython/issues/131492]: Fix a resource leak when constructing a **`gzip.GzipFile`** with a filename fails, for example when passing an invalid `compresslevel`.
- gh-131435 [https://github.com/python/cpython/issues/131435]: 10-20% performance improvement of **`random.randint()`**.
- gh-131461 [https://github.com/python/cpython/issues/131461]: Fix **`ResourceWarning`** when constructing a **`gzip.GzipFile`** in write mode with a broken file object.
- gh-125866 [https://github.com/python/cpython/issues/125866]: Deprecate the **`nturl2path`** module. Call **`urllib.request.url2pathname()`** and **`pathname2url()`** instead.
- gh-126367 [https://github.com/python/cpython/issues/126367]: Fix issue where **`urllib.request.url2pathname()`** raised **`OSError`** when given a Windows URI containing a colon character not following a drive letter, such as before an NTFS alternate data stream.
- gh-120144 [https://github.com/python/cpython/issues/120144]: Disable `CALL` event in **`bdb`** in `monitoring` backend when we don't need any new events on the code object to get a better performance.
- gh-131358 [https://github.com/python/cpython/issues/131358]: Register `cseuckr` as an encoding alias for `euc_kr`.
- gh-131325 [https://github.com/python/cpython/issues/131325]: Fix sendfile fallback implementation to drain data after writing to transport in **`asyncio`**.
- gh-90548 [https://github.com/python/cpython/issues/90548]: **`platform.libc_ver()`** can now detect and report the version of `musl` on Alpine Linux.
- gh-129843 [https://github.com/python/cpython/issues/129843]: Fix incorrect argument passing in **`warnings.warn_explicit()`**.
- gh-70647 [https://github.com/python/cpython/issues/70647]: When creating a **`datetime`** object with an out of range date a more informative error is raised.
- gh-130914 [https://github.com/python/cpython/issues/130914]: Allow **`graphlib.TopologicalSorter.prepare()`** to be called more than once as long as sorting has not started. Patch by Daniel Pope.

- gh-131236 [https://github.com/python/cpython/issues/131236]: Allow to generate multiple UUIDs at once via `python -m uuid --count`.
- gh-126895 [https://github.com/python/cpython/issues/126895]: Fix `readline` in free-threaded build.
- gh-121468 [https://github.com/python/cpython/issues/121468]: `$_asynctask` is added as a `pdb` convenience variable to access the current asyncio task if applicable.
- gh-118761 [https://github.com/python/cpython/issues/118761]: Improve import time of `locale` using lazy import `re`. Patch by Semyon Moroz.
- gh-129598 [https://github.com/python/cpython/issues/129598]: Fix `ast.unparse()` when `ast.Interactive` contains multiple statements.
- gh-85162 [https://github.com/python/cpython/issues/85162]: The `http.server` module now includes built-in support for HTTPS servers exposed by `http.server.HTTPSServer`. This functionality is exposed by the command-line interface (`python -m http.server`) through the `--tls-cert`, `--tls-key` and `--tls-password-file` options. Patch by Semyon Moroz.
- gh-129463 [https://github.com/python/cpython/issues/129463]: The implementations of equality and hashing for `annotationlib.ForwardRef` now use all attributes on the object. Two `ForwardRef` objects are equal only if all attributes are equal.
- gh-128593 [https://github.com/python/cpython/issues/128593]: `annotationlib.ForwardRef` objects no longer cache their value when they are successfully evaluated. Successive calls to `annotationlib.ForwardRef.evaluate()` may return different values.
- gh-117779 [https://github.com/python/cpython/issues/117779]: Fix reading duplicated entries in `zipfile` by name. Reading duplicated entries (except the last one) by `ZipInfo` now emits a warning instead of raising an exception.
- gh-128715 [https://github.com/python/cpython/issues/128715]: The class of `Structure`/`Union` field descriptors is now available as `CField`, and has new attributes to aid debugging and introspection.
- gh-128055 [https://github.com/python/cpython/issues/128055]: Fix

`test.test_sysconfig.test_sysconfigdata_json` when running outside the build directory (eg. after installing).

- gh-126037 [https://github.com/python/cpython/issues/126037]: `xml.etree.ElementTree`: Fix a crash in `Element.find`, `Element.findtext` and `Element.findall` when the tag to find implements an `__eq__()` method mutating the element being queried. Patch by Bénédikt Tran.
- gh-127794 [https://github.com/python/cpython/issues/127794]: When headers are added to `email.message.Message` objects, either through `email.message.Message.__setitem__()` or `email.message.Message.add_header()`, the field name is now validated according to RFC 5322, Section 2.2 [https://datatracker.ietf.org/doc/html/rfc5322.html#section-2.2] and a `ValueError` is raised if the field name contains any invalid characters.
- gh-123599 [https://github.com/python/cpython/issues/123599]: Deprecate **`pathlib.PurePath.as_uri()`**; use `pathlib.Path.as_uri()` instead.
- gh-126033 [https://github.com/python/cpython/issues/126033]: `xml.etree.ElementTree`: Fix a crash in `Element.remove` when the element is concurrently mutated. Patch by Bénédikt Tran.
- gh-120144 [https://github.com/python/cpython/issues/120144]: Add the optional backend of `sys.monitoring` to `bdb` and use it for `pdb`.
- gh-74598 [https://github.com/python/cpython/issues/74598]: Add `fnmatch.filterfalse()` for excluding names matching a pattern. Patch by Bénédikt Tran.
- gh-114917 [https://github.com/python/cpython/issues/114917]: Add support for AI_NUMERICSERV in getaddrinfo emulation
- bpo-17254 [https://bugs.python.org/issue?@action=redirect&bpo=17254]: Added aliases for Thai Language using Microsoft Code Pages.

Documentation

- gh-131417 [https://github.com/python/cpython/issues/131417]: Mention `asyncio.Future` and `asyncio.Task` in generic classes list.

Core and Builtins

- gh-131798 [https://github.com/python/cpython/issues/131798]: Allow the JIT to remove an extra `_TO_BOOL_BOOL` instruction after `_CONTAINS_OP_SET` by setting the return type to bool.
- gh-132011 [https://github.com/python/cpython/issues/132011]: Fix crash when calling **`list.append()`** as an unbound method.
- gh-131998 [https://github.com/python/cpython/issues/131998]: Fix a crash when using an unbound method descriptor object in a function where a bound method descriptor was used.
- gh-131591 [https://github.com/python/cpython/issues/131591]: Implement **PEP 768** [https://peps.python.org/pep-0768/] (Safe external debugger interface for CPython). Add a new **`sys.remote_exec()`** function to the **`sys`** module. This function schedules the execution of a Python file in a separate process. Patch by Pablo Galindo, Matt Wozniski and Ivona Stojanovic.
- gh-131798 [https://github.com/python/cpython/issues/131798]: Allow JIT to omit str guard in truthiness test when str type is known.
- gh-131833 [https://github.com/python/cpython/issues/131833]: Add support for optionally dropping grouping parentheses when using multiple exception types as per **PEP 758** [https://peps.python.org/pep-0758/]. Patch by Pablo Galindo
- gh-130924 [https://github.com/python/cpython/issues/130924]: Usage of a name in a function-scope annotation no longer triggers creation of a cell for that variable. This fixes a regression in earlier alphas of Python 3.14.
- gh-131800 [https://github.com/python/cpython/issues/131800]: Improve the experimental JIT’s ability to remove type checks for certain subscripting operations.
- gh-131738 [https://github.com/python/cpython/issues/131738]: Compiler emits optimized code for builtin any/all/tuple calls over a generator expression.
- gh-131719 [https://github.com/python/cpython/issues/131719]: Fix missing NULL check in `_PyMem_FreeDelayed` in free-threaded build.
- gh-131670 [https://github.com/python/cpython/issues/131670]: Fix

`anext()` failing on sync **`__anext__()`** raising an exception.

- gh-131666 [https://github.com/python/cpython/issues/131666]: Fix signature of `anext_awaitable.close` objects. Patch by Bénédikt Tran.
- gh-130415 [https://github.com/python/cpython/issues/130415]: Optimize comparison of two constants in JIT builds
- gh-129149 [https://github.com/python/cpython/issues/129149]: Add fast path for small and medium-size integers in **`PyLong_FromInt32()`**, **`PyLong_FromUInt32()`**, **`PyLong_FromInt64()`** and **`PyLong_FromUInt64()`**. Patch by Chris Eibl.
- gh-130887 [https://github.com/python/cpython/issues/130887]: Optimize the AArch64 code generation for the JIT. Patch by Diego Russo
- gh-130956 [https://github.com/python/cpython/issues/130956]: Optimize the AArch64 code generation for the JIT. Patch by Diego Russo
- gh-130928 [https://github.com/python/cpython/issues/130928]: Fix error message when formatting bytes using the `'i'` flag. Patch by Maxim Ageev.
- gh-130935 [https://github.com/python/cpython/issues/130935]: Annotations at the class and module level that are conditionally defined are now only reflected in `__annotations__` if the block they are in is executed. Patch by Jelle Zijlstra.
- gh-130775 [https://github.com/python/cpython/issues/130775]: Do not crash on negative `column` and `end_column` in **`ast`** locations.
- gh-130704 [https://github.com/python/cpython/issues/130704]: Optimize `LOAD_FAST` and its superinstruction form to reduce reference counting overhead. These instructions are replaced with faster variants that load borrowed references onto the operand stack when we can prove that the reference in the frame outlives the reference loaded onto the stack.
- gh-88887 [https://github.com/python/cpython/issues/88887]: Fixing multiprocessing Resource Tracker process leaking, usually observed when running Python as PID 1.
- gh-130115 [https://github.com/python/cpython/issues/130115]: Fix an issue with thread identifiers being sign-extended on some

platforms.

Library

- gh-99108 [https://github.com/python/cpython/issues/99108]: Add support for built-in implementation of HMAC (**RFC 2104** [https://datatracker.ietf.org/doc/html/rfc2104.html]) based on HACL*. Patch by Bénédikt Tran.

Core and Builtins

- gh-130080 [https://github.com/python/cpython/issues/130080]: Implement PEP 765: Disallow return/break/continue that exit a finally block.
- gh-129900 [https://github.com/python/cpython/issues/129900]: Fix return codes inside **`SystemExit`** not getting returned by the REPL.
- gh-128632 [https://github.com/python/cpython/issues/128632]: Disallow `__classdict__` as the name of a type parameter. Using this name would previously crash the interpreter in some circumstances.
- gh-126703 [https://github.com/python/cpython/issues/126703]: Improve performance of builtin methods by using a freelist.
- gh-126703 [https://github.com/python/cpython/issues/126703]: Improve performance of **`range`** by using a freelist.

C API

- gh-131740 [https://github.com/python/cpython/issues/131740]: Update PyUnstable_GC_VisitObjects to traverse perm gen.
- gh-131525 [https://github.com/python/cpython/issues/131525]: The PyTupleObject now caches the computed hash value in the new field ob_hash.

Build

- gh-131865 [https://github.com/python/cpython/issues/131865]: The DTrace build now properly passes the `CC` and `CFLAGS` variables to the `dtrace` command when utilizing SystemTap

on Linux.
- gh-131675 [https://github.com/python/cpython/issues/131675]: Fix mimalloc library builds for 32-bit ARM targets.
- gh-131691 [https://github.com/python/cpython/issues/131691]: clang-cl on Windows needs option `/EHa` to support SEH (structured exception handling) correctly. Fix by Chris Eibl.
- gh-131278 [https://github.com/python/cpython/issues/131278]: Add optimizing flag `WITH_COMPUTED_GOTOS` to Windows builds for when using a compiler that supports it (currently clang-cl). Patch by Chris Eibl.
- gh-130213 [https://github.com/python/cpython/issues/130213]: Update the vendored HACL* library to fix build issues with older clang compilers.
- gh-130673 [https://github.com/python/cpython/issues/130673]: Fix potential `KeyError` when handling object sections during JIT building process.

Python 3.14.0 alpha 6

Release date: 2025-03-14

macOS

- gh-128540 [https://github.com/python/cpython/issues/128540]: Ensure web browser is launched by **`webbrowser.open()`** on macOS, even for `file://` URLs.

Windows

- gh-131020 [https://github.com/python/cpython/issues/131020]: pylauncher [https://github.com/python/cpython/tree/3.14/PC/launcher2.c] correctly detects a BOM when searching for the shebang. Fix by Chris Eibl.

Tools/Demos

- gh-130453 [https://github.com/python/cpython/issues/130453]: Make it possible to override default keywords in **pygettext**.
- gh-85012 [https://github.com/python/cpython/issues/85012]:

Correctly reset `msgctxt` when compiling messages in **msgfmt**.
- gh-130453 [https://github.com/python/cpython/issues/130453]: Extend support for specifying custom keywords in **pygettext**.
- gh-130195 [https://github.com/python/cpython/issues/130195]: Add warning messages when `pygettext` unimplemented `-a/--extract-all` option is called.
- gh-130057 [https://github.com/python/cpython/issues/130057]: Add support for translator comments in **pygettext.py**.
- gh-130025 [https://github.com/python/cpython/issues/130025]: The iOS testbed now correctly handles symlinks used as Python framework references.
- gh-129911 [https://github.com/python/cpython/issues/129911]: Fix the keyword entry in the help output of **pygettext**.

Tests

- gh-129200 [https://github.com/python/cpython/issues/129200]: Multiple iOS testbed runners can now be started at the same time without introducing an ambiguity over simulator ownership.
- gh-130292 [https://github.com/python/cpython/issues/130292]: The iOS testbed will now run successfully on a machine that has not previously run Xcode tests (such as CI configurations).
- gh-130293 [https://github.com/python/cpython/issues/130293]: The tests of terminal colorization are no longer sensitive to the value of the `TERM` variable in the testing environment.
- gh-129401 [https://github.com/python/cpython/issues/129401]: Fix a flaky test in `test_repr_rlock` that checks the representation of **`multiprocessing.RLock`**.
- gh-126332 [https://github.com/python/cpython/issues/126332]: Add unit tests for pyrepl.

Security

- gh-127371 [https://github.com/python/cpython/issues/127371]: Avoid unbounded buffering for **`tempfile.SpooledTemporaryFile.writelines()`**. Previously, disk spillover was only checked after the lines iterator had been exhausted. This is now done after each line is written.

Library

- gh-131204 [https://github.com/python/cpython/issues/131204]: Use monospace font from System Font Stack for cross-platform support in **`difflib.HtmlDiff`**.
- gh-131196 [https://github.com/python/cpython/issues/131196]: Improve performance of **`uuid.UUID.hex`** and **`uuid.UUID.__str__`**.
- gh-130940 [https://github.com/python/cpython/issues/130940]: The behavior of `PyConfig.use_system_logger` was modified to be enabled by default on iOS. It remains disabled by default on macOS.
- gh-131123 [https://github.com/python/cpython/issues/131123]: Supported completions for attributes of convenience variables in **`pdb`**.
- gh-93096 [https://github.com/python/cpython/issues/93096]: Removed undocumented CLI `python -m difflib`. Use `python -m doctest Lib/difflib.py -v` instead. Patch by Semyon Moroz.
- gh-93096 [https://github.com/python/cpython/issues/93096]: Removed undocumented `-t` and `-v` arguments of `python -m pickle`. Use `python -m doctest Lib/pickle.py -v` instead. Patch by Semyon Moroz.
- gh-81267 [https://github.com/python/cpython/issues/81267]: Correct **`time.sleep()`** error message when an object that cannot be interpreted as an integer or float is provided.
- gh-93096 [https://github.com/python/cpython/issues/93096]: Removed undocumented `-t` and `-v` arguments of `python -m pickletools`. Use `python -m doctest Lib/pickletools.py -v` instead. Patch by Semyon Moroz.
- gh-131045 [https://github.com/python/cpython/issues/131045]: Fix issue with `__contains__`, values, and pseudo-members for **`enum.Flag`**.
- gh-130959 [https://github.com/python/cpython/issues/130959]: Fix pure-Python implementation of **`datetime.time.fromisoformat()`** to reject times with spaces in fractional part (for example, `12:34:56.400 +02:00`), matching the C implementation. Patch by Michał

Gorny.

- gh-130806 [https://github.com/python/cpython/issues/130806]: Deleting **`gzip.GzipFile`** before it is closed now emits a **`ResourceWarning`**.
- gh-130637 [https://github.com/python/cpython/issues/130637]: Add validation for numeric response data in poplib.POP3.stat() method
- gh-130665 [https://github.com/python/cpython/issues/130665]: Only apply locale to calendar CLI when set via `--locale` and not via `LANG` environment variable.
- gh-130660 [https://github.com/python/cpython/issues/130660]: `sys.ps1` and `sys.ps2` are now restored after **`code.interact()`** call.
- gh-130608 [https://github.com/python/cpython/issues/130608]: Remove *dirs_exist_ok* argument from **`pathlib.Path.copy()`** and **`copy_into()`**. These methods are new in Python 3.14.
- gh-130461 [https://github.com/python/cpython/issues/130461]: Remove `.. index::` directives from the **`uuid`** module documentation. These directives previously created entries in the general index for **`getnode()`** as well as the **`uuid1()`**, **`uuid3()`**, **`uuid4()`**, **`uuid5()`**, and **`uuid8()`** constructor functions.
- gh-130379 [https://github.com/python/cpython/issues/130379]: The zipapp module now calculates the list of files to be added to the archive before creating the archive. This avoids accidentally including the target when it is being created in the source directory.
- gh-82987 [https://github.com/python/cpython/issues/82987]: Inline breakpoints like **`breakpoint()`** or **`pdb.set_trace()`** will always stop the program at calling frame, ignoring the `skip` pattern (if any).
- gh-125377 [https://github.com/python/cpython/issues/125377]: `<tab>` at the beginning of the line in **`pdb`** multi-line input will fill in a 4-space indentation now, instead of inserting a `\t` character.
- gh-125413 [https://github.com/python/cpython/issues/125413]: Ensure the path returned from **`pathlib.Path.copy()`** or **`move()`** has fresh **`info`**.
- gh-65697 [https://github.com/python/cpython/issues/65697]: stdlib

configparser will now attempt to validate that keys it writes will not result in file corruption (creating a file unable to be accurately parsed by a future read() call from the same parser). Attempting a corrupting write() will raise an InvalidWriteError.

- gh-125413 [https://github.com/python/cpython/issues/125413]: Speed up **`Path.copy`** by making better use of **`info`** internally.
- gh-130285 [https://github.com/python/cpython/issues/130285]: Fix corner case for **`random.sample()`** allowing the *counts* parameter to specify an empty population. So now, `sample([], 0, counts=[])` and `sample('abc', k=0, counts=[0, 0, 0])` both give the same result as `sample([], 0)`.
- gh-124703 [https://github.com/python/cpython/issues/124703]: Executing `quit` command in **`pdb`** will raise **`bdb.BdbQuit`** when **`pdb`** is started from an interactive console using **`breakpoint()`** or **`pdb.set_trace()`**.
- gh-107773 [https://github.com/python/cpython/issues/107773]: Make **`datetime`** subclass **`__repr__()`** consistent both implementations. Patch by Semyon Moroz.
- gh-130250 [https://github.com/python/cpython/issues/130250]: Fix regression in `traceback.print_last()`.
- gh-123471 [https://github.com/python/cpython/issues/123471]: Make concurrent iterations over **`itertools.batched`** safe under free-threading.
- gh-57537 [https://github.com/python/cpython/issues/57537]: Support breakpoints for **`zipimport`** modules on **`pdb`**
- gh-130230 [https://github.com/python/cpython/issues/130230]: Fix crash in **`pow()`** with only **`Decimal`** third argument.
- gh-126944 [https://github.com/python/cpython/issues/126944]: Show explicit errors when required arguments of **`pdb`** commands are missing
- gh-127750 [https://github.com/python/cpython/issues/127750]: Improve repr of **`functools.singledispatchmethod`** methods and descriptors.
- gh-128520 [https://github.com/python/cpython/issues/128520]: Apply type conversion consistently in **`pathlib.PurePath`** and **`Path`** methods can accept a path object as an argument, such as **`match()`** and **`rename()`**. The argument is now converted to path object if it lacks a **`with_segments()`**

attribute, and not otherwise.

- gh-118761 [https://github.com/python/cpython/issues/118761]: Reverts a change in the previous release attempting to make some stdlib imports used within the **`subprocess`** module lazy as this was causing errors during `__del__` finalizers calling methods such as `terminate`, or `kill`, or `send_signal`.
- gh-130164 [https://github.com/python/cpython/issues/130164]: Fixed failure to raise **`TypeError`** in **`inspect.Signature.bind()`** for positional-only arguments provided by keyword when a variadic keyword argument (e.g. `**kwargs`) is present.
- gh-130151 [https://github.com/python/cpython/issues/130151]: Fix reference leaks in **`_hashlib.hmac_new()`** and **`_hashlib.hmac_digest()`**. Patch by Bénédikt Tran.
- gh-130145 [https://github.com/python/cpython/issues/130145]: Fix **`asyncio.AbstractEventloop.run_forever()`** when another loop is already running.
- gh-130139 [https://github.com/python/cpython/issues/130139]: Fix bug where **`ast.parse()`** did not error on AST input which is not of the correct type, when called with optimize=False.
- gh-127260 [https://github.com/python/cpython/issues/127260]: Forbid the use of colon («:») as a fractional component separator and other improvements to the consistency of error raising between the C and Python implementations of **`datetime.time.fromisoformat()`** and **`datetime.datetime.fromisoformat()`**. Patch by Semyon Moroz.
- gh-85795 [https://github.com/python/cpython/issues/85795]: Using **`super()`** and `__class__` closure variable in user-defined methods of **`typing.NamedTuple`** subclasses is now explicitly prohibited at runtime. Contributed by Bartosz Sławecki in gh-130082 [https://github.com/python/cpython/issues/130082].
- gh-118761 [https://github.com/python/cpython/issues/118761]: Improve import time of **`cmd`** by lazy importing **`inspect`** and removing **`string`**. Patch by Semyon Moroz.
- gh-129726 [https://github.com/python/cpython/issues/129726]: Fix **`gzip.GzipFile`** raising an unraisable exception during garbage collection when referring to a temporary object by

breaking the reference loop with `weakref`.

- gh-127750 [https://github.com/python/cpython/issues/127750]: Remove broken `functools.singledispatchmethod()` caching introduced in gh-85160 [https://github.com/python/cpython/issues/85160]. Achieve the same performance using different optimization.
- gh-129948 [https://github.com/python/cpython/issues/129948]: Add support for shared `set` to `multiprocessing.managers.SyncManager` via `SyncManager.set()`.
- gh-129965 [https://github.com/python/cpython/issues/129965]: Update MIME types for `.avi` and `.wav`. Add MIME types for `.docx`, `.pptx`, `.xlsx`, `.epub`, `.flac`, `.m4a`, `.odg`, `.odp`, `.ods`, `.odt`, `.oga`, `.ogg`, `.ogx` and `.weba`. Patch by Hugo van Kemenade.
- gh-129889 [https://github.com/python/cpython/issues/129889]: Support context manager protocol by `contextvars.Token`. Patch by Andrew Svetlov.
- gh-97850 [https://github.com/python/cpython/issues/97850]: Update the deprecation warning of `importlib.abc.Loader.load_module()`.
- gh-129678 [https://github.com/python/cpython/issues/129678]: `configparser.ConfigParser`: do not write an empty unnamed section
- gh-128641 [https://github.com/python/cpython/issues/128641]: Restore `configparser.ConfigParser.read()` performance.
- gh-129569 [https://github.com/python/cpython/issues/129569]: Fix `unicodedata.normalize()` to always return a built-in `str` object when given an input of a `str` subclass, regardless of whether the string is already normalized.
- gh-128231 [https://github.com/python/cpython/issues/128231]: Execution of multiple statements in the new REPL now stops immediately upon the first exception encountered. Patch by Bartosz Sławecki.
- gh-96092 [https://github.com/python/cpython/issues/96092]: Fix bug in `traceback.walk_stack()` called with None where it was skipping more frames than in prior versions. This bug fix also changes walk_stack to walk the stack in the frame where it was called rather than where it first gets used.

- gh-129288 [https://github.com/python/cpython/issues/129288]: Add optional `l2_cid` and `l2_bdaddr_type` fields to `socket` `BTPROTO_L2CAP` sockaddr tuple.
- gh-128703 [https://github.com/python/cpython/issues/128703]: Fix `mimetypes.guess_type()` to use default mapping for empty `Content-Type` in registry.
- gh-128647 [https://github.com/python/cpython/issues/128647]: Eagerly write to buffers passed to `gzip.GzipFile`'s `readinto()` and `readinto1()` implementations, avoiding unnecessary allocations. Patch by Chris Markiewicz.
- gh-128184 [https://github.com/python/cpython/issues/128184]: Improve display of `annotationlib.ForwardRef` object within `inspect.Signature` representations. This also fixes a `NameError` that was raised when using `dataclasses.dataclass()` on classes with unresolvable forward references.
- gh-128041 [https://github.com/python/cpython/issues/128041]: Add `concurrent.futures.ProcessPoolExecutor.terminate_wo` and `concurrent.futures.ProcessPoolExecutor.kill_workers` as ways to terminate or kill all living worker processes in the given pool. (Contributed by Charles Machalow in gh-130849 [https://github.com/python/cpython/issues/130849].)
- gh-127647 [https://github.com/python/cpython/issues/127647]: Add protocols `io.Reader` and `io.Writer` as alternatives to `typing.IO`, `typing.TextIO`, and `typing.BinaryIO`.
- gh-109798 [https://github.com/python/cpython/issues/109798]: Added additional information into error messages in `datetime`, and made the messages more consistent between the C and Python implementations. Patch by Semyon Moroz.
- gh-125746 [https://github.com/python/cpython/issues/125746]: Delay deprecated `zipimport.zipimporter.load_module()` removal time to 3.15. Use `zipimport.zipimporter.exec_module()` instead.
- gh-74028 [https://github.com/python/cpython/issues/74028]: Add the optional `buffersize` parameter to `concurrent.futures.Executor.map()` to limit the number of submitted tasks whose results have not yet been yielded. If the buffer is full, iteration over the *iterables* pauses until a result is yielded from the buffer.

- gh-124927 [https://github.com/python/cpython/issues/124927]: Non-printing characters are now properly handled in the new REPL.
- gh-124096 [https://github.com/python/cpython/issues/124096]: Turn on virtual terminal mode and enable bracketed paste in REPL on Windows console. (If the terminal does not support bracketed paste, enabling it does nothing.)
- gh-89083 [https://github.com/python/cpython/issues/89083]: Add **`uuid.uuid7()`** for generating UUIDv7 objects as specified in **RFC 9562** [https://datatracker.ietf.org/doc/html/rfc9562.html]. Patch by Bénédikt Tran.
- gh-89083 [https://github.com/python/cpython/issues/89083]: Add **`uuid.uuid6()`** for generating UUIDv6 objects as specified in **RFC 9562** [https://datatracker.ietf.org/doc/html/rfc9562.html]. Patch by Bénédikt Tran.
- gh-117151 [https://github.com/python/cpython/issues/117151]: Increase `io.DEFAULT_BUFFER_SIZE` from 8k to 128k and adjust **`open()`** on platforms where **`os.fstat()`** provides a `st_blksize` field (such as Linux) to use `max(min(blocksize, 8 MiB), io.DEFAULT_BUFFER_SIZE)` rather than always using the device block size. This should improve I/O performance. Patch by Romain Morotti.
- gh-105499 [https://github.com/python/cpython/issues/105499]: Make **`types.UnionType`** an alias for **`typing.Union`**. Both `int | str` and `Union[int, str]` now create instances of the same type. Patch by Jelle Zijlstra.
- gh-93096 [https://github.com/python/cpython/issues/93096]: Document the command-line for **`mimetypes`**. It now exits with `1` on failure instead of `0` and `2` on incorrect command-line parameters instead of `1`. Also, errors are printed to stderr instead of stdout and their text is made tighter. Patch by Oleg Iarygin and Hugo van Kemenade.

Documentation

- gh-125722 [https://github.com/python/cpython/issues/125722]: Require Sphinx 8.2.0 or later to build the Python documentation. Patch by Adam Turner.
- gh-129712 [https://github.com/python/cpython/issues/129712]: The

wheel tags supported by each macOS universal SDK option are now documented.
- gh-46236 [https://github.com/python/cpython/issues/46236]: C API: Document **`PyUnicode_RSplit()`**, **`PyUnicode_Partition()`** and **`PyUnicode_RPartition()`**.

Core and Builtins

- gh-131141 [https://github.com/python/cpython/issues/131141]: Fix data race in **`sys.monitoring`** instrumentation while registering callback.
- gh-130804 [https://github.com/python/cpython/issues/130804]: Fix support of unicode characters on Windows in the new REPL.
- gh-130932 [https://github.com/python/cpython/issues/130932]: Fix incorrect exception handling in `_PyModule_IsPossiblyShadowing`
- gh-122029 [https://github.com/python/cpython/issues/122029]: **`sys.setprofile()`** and **`sys.settrace()`** will not generate a `c_call` event for `INSTRUMENTED_CALL_FUNCTION_EX` if the callable is a method with a C function wrapped, because we do not generate `c_return` event in such case.
- gh-129964 [https://github.com/python/cpython/issues/129964]: Fix JIT crash on Windows on Arm. Patch by Diego Russo and Brandt Bucher.
- gh-130851 [https://github.com/python/cpython/issues/130851]: Fix a crash in the free threading build when constructing a **`code`** object with **`co_consts`** that contains instances of types that are not otherwise generated by the bytecode compiler.
- gh-128534 [https://github.com/python/cpython/issues/128534]: Ensure that both left and right branches have the same source for `async for` loops. Add these branches to the `co_branches()` iterator.
- gh-130794 [https://github.com/python/cpython/issues/130794]: Fix memory leak in the free threaded build when resizing a shared list or dictionary from multiple short-lived threads.
- gh-130415 [https://github.com/python/cpython/issues/130415]: Improve JIT understanding of integers in boolean context.
- gh-130382 [https://github.com/python/cpython/issues/130382]: Fix

`PyRefTracer_DESTROY` not being sent from `Python/ceval.c` `Py_DECREF()`.

- gh-130574 [https://github.com/python/cpython/issues/130574]: Renumber **`RESUME`** from 149 to 128.
- gh-124878 [https://github.com/python/cpython/issues/124878]: Fix race conditions during runtime finalization that could lead to accessing freed memory.
- gh-130415 [https://github.com/python/cpython/issues/130415]: Improve the experimental JIT's ability to narrow boolean values based on the results of truthiness tests.
- gh-130618 [https://github.com/python/cpython/issues/130618]: Fix a bug that was causing `UnicodeDecodeError` or `SystemError` to be raised when using f-strings with `lambda` expressions with non-ASCII characters. Patch by Pablo Galindo
- gh-123044 [https://github.com/python/cpython/issues/123044]: Make sure that the location of branch targets in `match` cases is in the body, not the pattern.
- gh-128534 [https://github.com/python/cpython/issues/128534]: Add branch monitoring (`BRANCH_LEFT` and `BRANCH_RIGHT` events) for `async for` loops.
- gh-130163 [https://github.com/python/cpython/issues/130163]: Fix possible crashes related to concurrent change and use of the **`sys`** module attributes.
- gh-122029 [https://github.com/python/cpython/issues/122029]: `INSTRUMENTED_CALL_KW` will expand the method before monitoring to reflect the actual behavior more accurately.
- gh-130415 [https://github.com/python/cpython/issues/130415]: Improve JIT's ability to optimize strings in boolean contexts.
- gh-130396 [https://github.com/python/cpython/issues/130396]: Use actual stack limits (from *pthread_getattr_np(3)* *[https://manpages.debian.org/pthread_getattr_np(3)]*) for linux, and other systems with `_GNU_SOURCE` defined, when determining limits for C stack protection.
- gh-128396 [https://github.com/python/cpython/issues/128396]: Fix a crash that occurs when calling **`locals()`** inside an inline comprehension that uses the same local variable as the outer frame scope where the variable is a free or cell var.
- gh-129107 [https://github.com/python/cpython/issues/129107]: Fix two more **`bytearray`** functions for free threading.

- gh-127705 [https://github.com/python/cpython/issues/127705]: Use tagged references (`_PyStackRef`) for the default build as well as for the free-threading build. This has a small negative performance impact short-term but will enable larger speedups in the future and significantly reduce maintenance costs by allowing a single implementation of tagged references in the future.
- gh-130094 [https://github.com/python/cpython/issues/130094]: Fix two race conditions involving concurrent imports that could lead to spurious failures with **`ModuleNotFoundError`**.
- gh-129107 [https://github.com/python/cpython/issues/129107]: Make **`bytearray`** iterator safe under free threading.
- gh-115802 [https://github.com/python/cpython/issues/115802]: Use the more efficient «medium» code model for JIT-compiled code on supported platforms.
- gh-107956 [https://github.com/python/cpython/issues/107956]: A `build-details.json` file is now install in the platform-independent standard library directory (**PEP 739** [https://peps.python.org/pep-0739/] implementation).
- gh-116042 [https://github.com/python/cpython/issues/116042]: Fix location for SyntaxErrors of invalid escapes in the tokenizer. Patch by Pablo Galindo
- gh-91079 [https://github.com/python/cpython/issues/91079]: Change C stack overflow protection to consider the amount of stack consumed, rather than a counter. This allows deeper recursion in many cases, but remains safe.
- gh-129715 [https://github.com/python/cpython/issues/129715]: Improve the experimental JIT's handling of returns to unknown callers.

Library

- gh-129983 [https://github.com/python/cpython/issues/129983]: Fix data race in compile_template in `sre.c`.

Core and Builtins

- gh-129967 [https://github.com/python/cpython/issues/129967]: Fix a race condition in the free threading build when `repr(set)` is called concurrently with `set.clear()`.

- gh-129953 [https://github.com/python/cpython/issues/129953]: The internal (evaluation) stack is now spilled to memory whenever execution escapes from the interpreter or JIT compiled code. This should have no observable effect in either Python or builtin extensions, but will allow various important optimizations in the future.
- gh-129515 [https://github.com/python/cpython/issues/129515]: Clarify syntax error messages for conditional expressions when a statement is specified before an `if` or after an `else` keyword.
- gh-129349 [https://github.com/python/cpython/issues/129349]: `bytes.fromhex()` and `bytearray.fromhex()` now accepts ASCII `bytes` and bytes-like objects.
- gh-129149 [https://github.com/python/cpython/issues/129149]: Add fast path for medium-size integers in `PyLong_FromSsize_t()`. Patch by Chris Eibl.
- gh-129107 [https://github.com/python/cpython/issues/129107]: Make the `bytearray` safe under free threading.
- gh-128974 [https://github.com/python/cpython/issues/128974]: Fix a crash in `UnicodeError.__str__` when custom attributes implement `__str__()` with side-effects. Patch by Bénédikt Tran.
- gh-126085 [https://github.com/python/cpython/issues/126085]: `typing.TypeAliasType` now supports star unpacking.
- gh-125331 [https://github.com/python/cpython/issues/125331]: `from __future__ import barry_as_FLUFL` now works in more contexts, including when it is used in files, with the `-c` flag, and in the REPL when there are multiple statements on the same line. Previously, it worked only on subsequent lines in the REPL, and when the appropriate flags were passed directly to `compile()`. Patch by Pablo Galindo.
- gh-121464 [https://github.com/python/cpython/issues/121464]: Make concurrent iterations over the same `enumerate()` iterator safe under free-threading. See Strategy for Iterators in Free Threading [https://github.com/python/cpython/issues/124397].
- gh-87790 [https://github.com/python/cpython/issues/87790]: Support underscore and comma as thousands separators in the fractional part for floating-point presentation types of the new-style string formatting (with `format()` or f-strings). Patch by Sergey B Kirpichev.

- gh-124445 [https://github.com/python/cpython/issues/124445]: Fix specialization of generic aliases that are generic over a `typing.ParamSpec` and have been specialized with a nested type variable.
- gh-120608 [https://github.com/python/cpython/issues/120608]: Adapt `reversed()` for use in the free-threading build. The `reversed()` is still not thread-safe in the sense that concurrent iterations may see the same object, but they will not corrupt the interpreter state.

Library

- gh-100388 [https://github.com/python/cpython/issues/100388]: Fix the `platform._sys_version()` method when `__DATE__` is undefined at buildtime by changing default buildtime datetime string to the UNIX epoch.

Core and Builtins

- bpo-44369 [https://bugs.python.org/issue?@action=redirect&bpo=44369]: Improve syntax errors for incorrectly closed strings. Patch by Pablo Galindo

C API

- gh-111178 [https://github.com/python/cpython/issues/111178]: Fix `PyCMethod` API: replace `size_t nargs` with `Py_ssize_t nargs` in `PyCMethod`. Patch by Victor Stinner.

- gh-130947 [https://github.com/python/cpython/issues/130947]: Add again `PySequence_Fast()` to the limited C API. Patch by Victor Stinner.

- gh-128863 [https://github.com/python/cpython/issues/128863]: The following private functions are deprecated and planned for removal in Python 3.18:

    - `_PyUnicodeWriter_Init()`: replace `_PyUnicodeWriter_Init(&writer)` with `writer`

    `= PyUnicodeWriter_Create(0)`.
    - **`_PyUnicodeWriter_Finish()`**: replace `_PyUnicodeWriter_Finish(&writer)` with **`PyUnicodeWriter_Finish(writer)`**.
    - **`_PyUnicodeWriter_Dealloc()`**: replace `_PyUnicodeWriter_Dealloc(&writer)` with **`PyUnicodeWriter_Discard(writer)`**.
    - **`_PyUnicodeWriter_WriteChar()`**: replace `_PyUnicodeWriter_WriteChar(&writer, ch)` with **`PyUnicodeWriter_WriteChar(writer, ch)`**.
    - **`_PyUnicodeWriter_WriteStr()`**: replace `_PyUnicodeWriter_WriteStr(&writer, str)` with **`PyUnicodeWriter_WriteStr(writer, str)`**.
    - **`_PyUnicodeWriter_WriteSubstring()`**: replace `_PyUnicodeWriter_WriteSubstring(&writer, str, start, end)` with **`PyUnicodeWriter_WriteSubstring(writer, str, start, end)`**.
    - **`_PyUnicodeWriter_WriteASCIIString()`**: replace `_PyUnicodeWriter_WriteASCIIString(&writer, str)` with **`PyUnicodeWriter_WriteUTF8(writer, str)`**.
    - **`_PyUnicodeWriter_WriteLatin1String()`**: replace `_PyUnicodeWriter_WriteLatin1String(&writer, str)` with **`PyUnicodeWriter_WriteUTF8(writer, str)`**.
    - **`_PyUnicodeWriter_Prepare()`**: (no replacement).
    - **`_PyUnicodeWriter_PrepareKind()`**: (no replacement).

  The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get these new public functions on Python 3.13 and older.

  Patch by Victor Stinner.

- gh-45325 [https://github.com/python/cpython/issues/45325]: Add a new `p` format parameter to **`Py_BuildValue()`** that allows to take a C integer and produce a Python **`bool`** object. Patch by Pablo Galindo.

Build

- gh-131035 [https://github.com/python/cpython/issues/131035]: Use `-flto=thin` for faster build times using clang-cl on Windows. Patch by Chris Eibl.
- gh-130740 [https://github.com/python/cpython/issues/130740]: Ensure that `Python.h` is included before `stdbool.h` unless `pyconfig.h` is included before or in some platform-specific contexts.
- gh-130090 [https://github.com/python/cpython/issues/130090]: Building with `PlatformToolset=ClangCL` on Windows now supports PGO (profile guided optimization). Patch by Chris Eibl with invaluable support from Steve Dover.
- gh-129819 [https://github.com/python/cpython/issues/129819]: Allow building the JIT with the tailcall interpreter.
- gh-129989 [https://github.com/python/cpython/issues/129989]: Fix a bug where the tailcall interpreter was enabled when `--without-tail-call-interp` was provided to the configure script.
- gh-129838 [https://github.com/python/cpython/issues/129838]: Don’t redefine `_Py_NO_SANITIZE_UNDEFINED` when compiling with a recent GCC version and undefined sanitizer enabled.
- gh-82909 [https://github.com/python/cpython/issues/82909]: `#pragma`-based linking with `python3*.lib` can now be switched off with Py_NO_LINK_LIB. Patch by Jean-Christophe Fillion-Robin.

Python 3.14.0 alpha 5

Release date: 2025-02-11

macOS

- gh-91132 [https://github.com/python/cpython/issues/91132]: Update macOS installer to use ncurses 6.5.

Tools/Demos

- gh-129248 [https://github.com/python/cpython/issues/129248]: The iOS test runner now strips the log prefix from each line output by the test suite.
- gh-104400 [https://github.com/python/cpython/issues/104400]: Fix several bugs in extraction by switching to an AST parser in **pygettext**.

Tests

- gh-129386 [https://github.com/python/cpython/issues/129386]: Add `test.support.reset_code`, which can be used to reset various bytecode-level optimizations and local instrumentation for a function.
- gh-128474 [https://github.com/python/cpython/issues/128474]: Disable `test_embed` test cases that segfault on BOLT instrument binaries. The tests are only disabled when BOLT is enabled.
- gh-128003 [https://github.com/python/cpython/issues/128003]: Add an option `--parallel-threads=N` to the regression test runner that runs individual tests in multiple threads in parallel in order to find concurrency bugs. Note that most of the test suite is not yet reviewed for thread-safety or annotated with `@thread_unsafe` when necessary.

Security

- gh-105704 [https://github.com/python/cpython/issues/105704]: When using **`urllib.parse.urlsplit()`** and **`urllib.parse.urlparse()`** host parsing would not reject domain names containing square brackets (`[` and `]`). Square brackets are only valid for IPv6 and IPvFuture hosts according to RFC 3986 Section 3.2.2 [https://www.rfc-editor.org/rfc/rfc3986#section-3.2.2].
- gh-126108 [https://github.com/python/cpython/issues/126108]: Fix a possible `NULL` pointer dereference in **`PySys_AddWarnOptionUnicode()`**.
- gh-80222 [https://github.com/python/cpython/issues/80222]: Fix bug in the folding of quoted strings when flattening an email

message using a modern email policy. Previously when a quoted string was folded so that it spanned more than one line, the surrounding quotes and internal escapes would be omitted. This could theoretically be used to spoof header lines using a carefully constructed quoted string if the resulting rendered email was transmitted or re-parsed.

- gh-119511 [https://github.com/python/cpython/issues/119511]: Fix a potential denial of service in the **`imaplib`** module. When connecting to a malicious server, it could cause an arbitrary amount of memory to be allocated. On many systems this is harmless as unused virtual memory is only a mapping, but if this hit a virtual address size limit it could lead to a **`MemoryError`** or other process crash. On unusual systems or builds where all allocated memory is touched and backed by actual ram or storage it could've consumed resources doing so until similarly crashing.

Library

- gh-129939 [https://github.com/python/cpython/issues/129939]: Comparison pages with highlighted changes generated by the **`difflib.HtmlDiff`** class now support dark mode.
- gh-129928 [https://github.com/python/cpython/issues/129928]: Raise **`sqlite3.ProgrammingError`** if a user-defined SQL function with invalid number of parameters is created. Patch by Erlend Aasland.
- gh-129583 [https://github.com/python/cpython/issues/129583]: Update bundled pip to 25.0.1
- gh-129766 [https://github.com/python/cpython/issues/129766]: Fix crash in **`warnings`**, when calling `_release_lock()` with no existing lock.
- gh-129005 [https://github.com/python/cpython/issues/129005]: `_pyio.FileIO.readall()` now allocates, resizes, and fills a data buffer using the same algorithm `_io.FileIO.readall()` uses.
- gh-129646 [https://github.com/python/cpython/issues/129646]: Update the locale alias mapping in the **`locale`** module to match the latest X Org locale alias mapping and support new locales in Glibc 2.41.
- gh-128317 [https://github.com/python/cpython/issues/128317]: Put

CLI calendar highlighting in private class, removing `highlight_day` from public **`calendar.TextCalendar`** API. Patch by Hugo van Kemenade.

- gh-129603 [https://github.com/python/cpython/issues/129603]: Fix bugs where **`sqlite3.Row`** objects could segfault if their inherited **`description`** was set to `None`. Patch by Erlend Aasland.
- gh-129559 [https://github.com/python/cpython/issues/129559]: Add **`bytearray.resize()`** method so **`bytearray`** can be efficiently resized in place.
- gh-129502 [https://github.com/python/cpython/issues/129502]: Unlikely errors in preparing arguments for **`ctypes`** callback are now handled in the same way as errors raised in the callback of in converting the result of the callback – using **`sys.unraisablehook()`** instead of **`sys.excepthook()`** and not setting **`sys.last_exc`** and other variables.
- gh-129403 [https://github.com/python/cpython/issues/129403]: Corrected **`ValueError`** message for **`asyncio.Barrier`** and **`threading.Barrier`**.
- gh-129409 [https://github.com/python/cpython/issues/129409]: Fix an integer overflow in the **`csv`** module when writing a data field larger than 2GB.
- gh-126400 [https://github.com/python/cpython/issues/126400]: Add a socket *timeout* keyword argument to **`logging.handlers.SysLogHandler`**.
- gh-118761 [https://github.com/python/cpython/issues/118761]: Always lazy import `warnings` in **`threading`**. Patch by Taneli Hukkinen.
- gh-118761 [https://github.com/python/cpython/issues/118761]: Improve import time of **`subprocess`** by lazy importing `locale` and `signal`. Patch by Taneli Hukkinen.
- gh-129346 [https://github.com/python/cpython/issues/129346]: In **`sqlite3`**, handle out-of-memory when creating user-defined SQL functions.
- gh-129005 [https://github.com/python/cpython/issues/129005]: Optimize `_pyio.FileIO.readinto` by avoiding unnecessary objects and copies using **`os.readinto()`**.
- gh-129195 [https://github.com/python/cpython/issues/129195]: Support reporting call graph information from **`asyncio.staggered.staggered_race()`**.

- gh-129205 [https://github.com/python/cpython/issues/129205]: Add **`os.readinto()`** to read into a buffer object from a file descriptor.
- gh-128772 [https://github.com/python/cpython/issues/128772]: Fix **`pydoc`** for methods with the `__module__` attribute equal to `None`.
- gh-129061 [https://github.com/python/cpython/issues/129061]: Fix FORCE_COLOR and NO_COLOR when empty strings. Patch by Hugo van Kemenade.
- gh-92897 [https://github.com/python/cpython/issues/92897]: Scheduled the deprecation of the `check_home` argument of **`sysconfig.is_python_build()`** to Python 3.15.
- gh-129064 [https://github.com/python/cpython/issues/129064]: Deprecate **`sysconfig.expand_makefile_vars()`**, in favor of using **`sysconfig.get_paths()`** with the `vars` argument.
- gh-128550 [https://github.com/python/cpython/issues/128550]: Removed an incorrect optimization relating to eager tasks in **`asyncio.TaskGroup`** that resulted in cancellations being missed.
- gh-128991 [https://github.com/python/cpython/issues/128991]: Release the enter frame reference within **`bdb`** callback
- gh-118761 [https://github.com/python/cpython/issues/118761]: Reduce import time of **`pstats`** and **`zipfile`** by up to 20%, by removing unnecessary imports to **`typing`**. Patch by Bénédikt Tran.
- gh-128978 [https://github.com/python/cpython/issues/128978]: Fix a **`NameError`** in **`sysconfig.expand_makefile_vars()`**. Patch by Bénédikt Tran.
- gh-128961 [https://github.com/python/cpython/issues/128961]: Fix a crash when setting state on an exhausted **`array.array`** iterator.
- gh-128894 [https://github.com/python/cpython/issues/128894]: Fix `traceback.TracebackException._format_syntax_error` not to fail on exceptions with custom metadata.
- gh-128916 [https://github.com/python/cpython/issues/128916]: Do not attempt to set `SO_REUSEPORT` on sockets of address families other than `AF_INET` and `AF_INET6`, as it is meaningless with these address families, and the call with fail with Linux kernel 6.12.9 and newer.

- gh-118761 [https://github.com/python/cpython/issues/118761]: Improve import time of **`tomllib`** by removing `typing`, `string`, and `tomllib._types` imports. Patch by Taneli Hukkinen.
- gh-128679 [https://github.com/python/cpython/issues/128679]: **`tracemalloc`**: Fix race conditions when **`tracemalloc.stop()`** is called by a thread, while other threads are tracing memory allocations. Patch by Victor Stinner.
- gh-128891 [https://github.com/python/cpython/issues/128891]: Add specialized opcodes to `opcode.opname`.
- gh-118761 [https://github.com/python/cpython/issues/118761]: Reduce import time of **`gettext`** by up to ten times, by importing **`re`** on demand. In particular, `re` is no longer implicitly exposed as `gettext.re`. Patch by Eli Schwartz.
- gh-118761 [https://github.com/python/cpython/issues/118761]: Reduce the import time of **`optparse`** when no help text is printed. Patch by Eli Schwartz.
- gh-128657 [https://github.com/python/cpython/issues/128657]: Fix possible extra reference when using objects returned by **`hashlib.sha256()`** under free threading.
- gh-118761 [https://github.com/python/cpython/issues/118761]: Reduce the import time of **`csv`** by up to five times, by importing **`re`** on demand. In particular, `re` is no more implicitly exposed as `csv.re`. Patch by Bénédikt Tran.
- gh-128308 [https://github.com/python/cpython/issues/128308]: Support the *name* keyword argument for eager tasks in **`asyncio.loop.create_task()`**, **`asyncio.create_task()`** and **`asyncio.TaskGroup.create_task()`**, by passing on all *kwargs* to the task factory set by **`asyncio.loop.set_task_factory()`**.
- gh-118761 [https://github.com/python/cpython/issues/118761]: Improve the performance of **`base64.b16decode()`** by up to ten times by more efficiently checking the byte-string for hexadecimal digits. Reduce the import time of **`base64`** by up to six times, by no longer importing **`re`**. Patch by Bénédikt Tran, Chris Markiewicz, and Adam Turner.
- gh-128156 [https://github.com/python/cpython/issues/128156]: When using macOS system `libffi`, support for complex types in

`ctypes` is now checked at runtime (macOS 10.15 or newer). The types must also be available at build time.

- gh-128636 [https://github.com/python/cpython/issues/128636]: Fix PyREPL failure when **`os.environ`** is overwritten with an invalid value.
- gh-128498 [https://github.com/python/cpython/issues/128498]: Default to stdout isatty for color detection instead of stderr. Patch by Hugo van Kemenade.
- gh-128384 [https://github.com/python/cpython/issues/128384]: Add locking to **`warnings`** to avoid some data races when free-threading is used. Change `_warnings_runtime_state.mutex` to be a recursive mutex and expose it to **`warnings`**, via the **`_acquire_lock()`** and **`_release_lock()`** functions. The lock is held when `filters` and `_filters_version` are updated.
- gh-128509 [https://github.com/python/cpython/issues/128509]: Add **`sys._is_immortal()`** for identifying immortal objects at runtime.
- gh-128479 [https://github.com/python/cpython/issues/128479]: Fix **`asyncio.staggered.staggered_race()`** leaking tasks and issuing an unhandled exception.
- gh-128427 [https://github.com/python/cpython/issues/128427]: **`uuid.NIL`** and **`uuid.MAX`** are now available to represent the Nil and Max UUID formats as defined by **RFC 9562** [https://datatracker.ietf.org/doc/html/rfc9562.html].
- gh-91279 [https://github.com/python/cpython/issues/91279]: **`zipfile.ZipFile.writestr()`** now respect `SOURCE_DATE_EPOCH` that distributions can set centrally and have build tools consume this in order to produce reproducible output.
- gh-112064 [https://github.com/python/cpython/issues/112064]: Fix incorrect handling of negative read sizes in **`HTTPResponse.read`**. Patch by Yury Manushkin.
- gh-58956 [https://github.com/python/cpython/issues/58956]: Fixed a frame reference leak in **`bdb`**.
- gh-128131 [https://github.com/python/cpython/issues/128131]: Completely support random access of uncompressed unencrypted read-only zip files obtained by **`ZipFile.open`**.
- gh-127975 [https://github.com/python/cpython/issues/127975]: Avoid

reusing quote types in `ast.unparse()` if not needed.

- gh-115514 [https://github.com/python/cpython/issues/115514]: Fix exceptions and incomplete writes after **`asyncio._SelectorTransport`** is closed before writes are completed.
- gh-121604 [https://github.com/python/cpython/issues/121604]: Add missing Deprecation warnings for `importlib.machinery.DEBUG_BYTECODE_SUFFIXES`, `importlib.machinery.OPTIMIZED_BYTECODE_SUFFIXES`, `importlib.machinery.WindowsRegistryFinder`, `importlib.abc.ResourceLoader`, `importlib.abc.SourceLoader.path_mtime()`.
- gh-127873 [https://github.com/python/cpython/issues/127873]: When `-E` is set, only ignore `PYTHON_COLORS` and not `FORCE_COLOR/NO_COLOR/TERM` when colourising output. Patch by Hugo van Kemenade.
- gh-125413 [https://github.com/python/cpython/issues/125413]: Add `pathlib.Path.info` attribute, which stores an object implementing the `pathlib.types.PathInfo` protocol (also new). The object supports querying the file type and internally caching `stat()` results. Path objects generated by `iterdir()` are initialized with file type information gleaned from scanning the parent directory.
- gh-127712 [https://github.com/python/cpython/issues/127712]: Fix handling of the `secure` argument of `logging.handlers.SMTPHandler`.
- gh-127096 [https://github.com/python/cpython/issues/127096]: Do not recreate unnamed section on every read in `configparser.ConfigParser`. Patch by Andrey Efremov.
- gh-124369 [https://github.com/python/cpython/issues/124369]: Deprecate `pdb.Pdb.curframe_locals`
- gh-126332 [https://github.com/python/cpython/issues/126332]: Fix _pyrepl crash when entering a double CTRL-Z on an overflowing line.
- gh-125553 [https://github.com/python/cpython/issues/125553]: Fix round-trip invariance for backslash continuations in `tokenize.untokenize()`.
- gh-91048 [https://github.com/python/cpython/issues/91048]: Add `asyncio.capture_call_graph()` and `asyncio.print_call_graph()` functions.

- gh-124703 [https://github.com/python/cpython/issues/124703]: Quitting `pdb` in `inline` mode will emit a confirmation prompt and exit gracefully now, instead of printing an exception traceback.
- gh-123987 [https://github.com/python/cpython/issues/123987]: Fixed issue in NamespaceReader where a non-path item in a namespace path, such as a sentinel added by an editable installer, would break resource loading.
- gh-119349 [https://github.com/python/cpython/issues/119349]: Add the `ctypes.util.dllist()` function to list the loaded shared libraries for the current process.
- gh-55454 [https://github.com/python/cpython/issues/55454]: Add IMAP4 `IDLE` support to the `imaplib` module. Patch by Forest.
- gh-119257 [https://github.com/python/cpython/issues/119257]: Show tab completions menu below the current line, which results in less janky behaviour, and fixes a cursor movement bug. Patch by Daniel Hollas
- gh-101410 [https://github.com/python/cpython/issues/101410]: Support custom messages for domain errors in the `math` module (`math.sqrt()`, `math.log()` and `math.atanh()` were modified as examples). Patch by Charlie Zhao and Sergey B Kirpichev.
- gh-81340 [https://github.com/python/cpython/issues/81340]: Use `os.copy_file_range()` in `shutil.copy()`, `shutil.copy2()`, and `shutil.copyfile()` functions by default. An underlying Linux system call gives filesystems an opportunity to implement the use of copy-on-write (in case of btrfs and XFS) or server-side copy (in the case of NFS.) Patch by Illia Volochii.
- bpo-27307 [https://bugs.python.org/issue?@action=redirect&bpo=27307]: Add attribute and item access support to `string.Formatter` in auto-numbering mode, which allows format strings like “{.name}” and “{[1]}”.

IDLE

- gh-129873 [https://github.com/python/cpython/issues/129873]: Simplify displaying the IDLE doc by only copying the text section of idle.html to idlelib/help.html. Patch by Stan

Ulbrych.

Documentation

- gh-125722 [https://github.com/python/cpython/issues/125722]: Require Sphinx 8.1.3 or later to build the Python documentation. Patch by Adam Turner.
- gh-67206 [https://github.com/python/cpython/issues/67206]: Document that `string.printable` is not printable in the POSIX sense. In particular, `string.printable.isprintable()` returns `False`. Patch by Bénédikt Tran.

Core and Builtins

- gh-100239 [https://github.com/python/cpython/issues/100239]: Replace the opcode BINARY_SUBSCR and its family by BINARY_OP with oparg NB_SUBSCR.
- gh-129732 [https://github.com/python/cpython/issues/129732]: Fixed a race in `_Py_qsbr_reserve` in the free threading build.
- gh-129763 [https://github.com/python/cpython/issues/129763]: Remove the internal `LLTRACE` macro (use `Py_DEBUG` instead).
- gh-129715 [https://github.com/python/cpython/issues/129715]: Improve JIT performance for generators.
- gh-129643 [https://github.com/python/cpython/issues/129643]: Fix thread safety of `PyList_Insert()` in free-threading builds.
- gh-129668 [https://github.com/python/cpython/issues/129668]: Fix race condition when raising `MemoryError` in the free threaded build.
- gh-129643 [https://github.com/python/cpython/issues/129643]: Fix thread safety of `PyList_SetItem()` in free-threading builds. Patch by Kumar Aditya.
- gh-128563 [https://github.com/python/cpython/issues/128563]: Fix an issue where the «lltrace» debug feature could have been incorrectly enabled for some frames.
- gh-129393 [https://github.com/python/cpython/issues/129393]: On FreeBSD, `sys.platform` doesn't contain the major version anymore. It is always `'freebsd'`, instead of `'freebsd13'` or `'freebsd14'`.

Library

- gh-129345 [https://github.com/python/cpython/issues/129345]: Fix null pointer dereference in `syslog.openlog()` when an audit hook raises an exception.

Core and Builtins

- gh-129231 [https://github.com/python/cpython/issues/129231]: Improve memory layout of JIT traces. Patch by Diego Russo
- gh-129149 [https://github.com/python/cpython/issues/129149]: Add fast path for medium-size integers in `PyLong_FromUnsignedLong()`, `PyLong_FromUnsignedLongLong()` and `PyLong_FromSize_t()`.
- gh-129201 [https://github.com/python/cpython/issues/129201]: The free-threaded version of the cyclic garbage collector has been optimized to conditionally use CPU prefetch instructions during the collection. This can reduce collection times by making it more likely that data is in the CPU cache when it is needed. The prefetch instructions are enabled if the number of long-lived objects (objects surviving a full collection) exceeds a threshold.
- gh-129093 [https://github.com/python/cpython/issues/129093]: Fix f-strings such as `f'{expr=}'` sometimes not displaying the full expression when the expression contains `!=`.
- gh-124363 [https://github.com/python/cpython/issues/124363]: Treat debug expressions in f-string as raw strings. Patch by Pablo Galindo
- gh-128714 [https://github.com/python/cpython/issues/128714]: Fix the potential races in get/set dunder methods `__annotations__`, `__annotate__` and `__type_params__` for function object, and add related tests.
- gh-128799 [https://github.com/python/cpython/issues/128799]: Add frame of `except*` to traceback when it wraps a naked exception.
- gh-128842 [https://github.com/python/cpython/issues/128842]: Collect JIT memory stats using pystats. Patch by Diego Russo.
- gh-100239 [https://github.com/python/cpython/issues/100239]: Specialize `BINARY_OP` for bitwise logical operations on

compact ints.

- gh-128910 [https://github.com/python/cpython/issues/128910]: Undocumented and unused private C-API functions `_PyTrash_begin` and `_PyTrash_end` are removed.
- gh-128807 [https://github.com/python/cpython/issues/128807]: Add a marking phase to the free-threaded GC. This is similar to what was done in gh-126491 [https://github.com/python/cpython/issues/126491]. Since the free-threaded GC does not have generations and is not incremental, the marking phase looks for all objects reachable from known roots. The roots are objects known to not be garbage, like the module dictionary for **`sys`**. For most programs, this marking phase should make the GC a bit faster since typically less work is done per object.
- gh-100239 [https://github.com/python/cpython/issues/100239]: Add opcode `BINARY_OP_EXTEND` which executes a pair of functions (guard and specialization functions) accessed from the inline cache.
- gh-128563 [https://github.com/python/cpython/issues/128563]: A new type of interpreter has been added to CPython. This interpreter uses tail calls for its instruction handlers. Preliminary benchmark results suggest 7-11% geometric mean faster on pyperformance (depending on platform), and up to 30% faster on Python-intensive workloads. This interpreter currently only works on newer compilers, such as `clang-19`. Other compilers will continue using the old interpreter. Patch by Ken Jin, with ideas on how to implement this in CPython by Mark Shannon, Garret Gu, Haoran Xu, and Josh Haberman.
- gh-126703 [https://github.com/python/cpython/issues/126703]: Improve performance of iterating over lists and tuples by using a freelist for the iterator objects.
- gh-127953 [https://github.com/python/cpython/issues/127953]: The time to handle a `LINE` event in sys.monitoring (and sys.settrace) is now independent of the number of lines in the code object.
- gh-128330 [https://github.com/python/cpython/issues/128330]: Restore terminal control characters on REPL exit.
- gh-128016 [https://github.com/python/cpython/issues/128016]: Improved the `SyntaxWarning` message for invalid escape sequences to clarify that such sequences will raise a

`SyntaxError` in future Python releases. The new message also suggests a potential fix, i.e., `Did you mean "\\e"?`.

- gh-126004 [https://github.com/python/cpython/issues/126004]: Fix handling of `UnicodeError.start` and `UnicodeError.end` values in the `codecs.replace_errors()` error handler. Patch by Bénédikt Tran.
- gh-126004 [https://github.com/python/cpython/issues/126004]: Fix handling of `UnicodeError.start` and `UnicodeError.end` values in the `codecs.backslashreplace_errors()` error handler. Patch by Bénédikt Tran.
- gh-126004 [https://github.com/python/cpython/issues/126004]: Fix handling of `UnicodeError.start` and `UnicodeError.end` values in the `codecs.xmlcharrefreplace_errors()` error handler. Patch by Bénédikt Tran.
- gh-127119 [https://github.com/python/cpython/issues/127119]: Slightly optimize the `int` deallocator.
- gh-127349 [https://github.com/python/cpython/issues/127349]: Fixed the error when resizing terminal in Python REPL. Patch by Semyon Moroz.
- gh-125723 [https://github.com/python/cpython/issues/125723]: Fix crash with `gi_frame.f_locals` when generator frames outlive their generator. Patch by Mikhail Efimov.

Library

- gh-126349 [https://github.com/python/cpython/issues/126349]: Add `turtle.fill()`, `turtle.poly()` and `turtle.no_animation()` context managers. Patch by Marie Roald and Yngve Mardal Moe.

Core and Builtins

- gh-115911 [https://github.com/python/cpython/issues/115911]: If the current working directory cannot be determined due to permissions, then import will no longer raise `PermissionError`. Patch by Alex Willmer.

Library

- gh-112713 [https://github.com/python/cpython/issues/112713]: Added support for the `Partitioned` cookie flag in **`http.cookies`**.

C API

- gh-129533 [https://github.com/python/cpython/issues/129533]: Update **`PyGC_Enable()`**, **`PyGC_Disable()`**, **`PyGC_IsEnabled()`** to use atomic operation for thread-safety at free-threading build. Patch by Donghee Na.
- gh-89188 [https://github.com/python/cpython/issues/89188]: Implement **`PyUnicode_KIND()`** and **`PyUnicode_DATA()`** as function, in addition to the macros with the same names. The macros rely on C bit fields which have compiler-specific layout. Patch by Victor Stinner.
- gh-91417 [https://github.com/python/cpython/issues/91417]: Remove **`PySequence_Fast()`** from the limited C API, since this function has to be used with **`PySequence_Fast_GET_ITEM`** which never worked in the limited C API. Patch by Victor Stinner.
- gh-128509 [https://github.com/python/cpython/issues/128509]: Add **`PyUnstable_IsImmortal()`** for determining whether an object is immortal.
- gh-129033 [https://github.com/python/cpython/issues/129033]: Remove `_PyInterpreterState_GetConfigCopy()` and `_PyInterpreterState_SetConfig()` private functions. Use instead **`PyConfig_Get()`** and **`PyConfig_Set()`**, public C API added by **PEP 741** [https://peps.python.org/pep-0741/] «Python Configuration C API». Patch by Victor Stinner.
- gh-129033 [https://github.com/python/cpython/issues/129033]: Remove the private `_Py_InitializeMain()` function. It was a provisional API added to Python 3.8 by **PEP 587** [https://peps.python.org/pep-0587/]. Patch by Victor Stinner.

- gh-128844 [https://github.com/python/cpython/issues/128844]: Add `PyUnstable_TryIncRef()` and `PyUnstable_EnableTryIncRef()` unstable APIs. These are helpers for dealing with unowned references in a thread-safe way, particularly in the free threading build.
- gh-128911 [https://github.com/python/cpython/issues/128911]: Add `PyImport_ImportModuleAttr()` and `PyImport_ImportModuleAttrString()` helper functions to import a module and get an attribute of the module. Patch by Victor Stinner.
- gh-128863 [https://github.com/python/cpython/issues/128863]: The following private functions are deprecated and planned for removal in Python 3.18:

    - `_PyBytes_Join()`: use `PyBytes_Join()`.
    - `_PyDict_GetItemStringWithError()`: use `PyDict_GetItemStringRef()`.
    - `_PyDict_Pop()`: use `PyDict_Pop()`.
    - `_PyLong_Sign()`: use `PyLong_GetSign()`.
    - `_PyLong_FromDigits()` and `_PyLong_New()`: use `PyLongWriter_Create()`.
    - `_PyThreadState_UncheckedGet()`: use `PyThreadState_GetUnchecked()`.
    - `_PyUnicode_AsString()`: use `PyUnicode_AsUTF8()`.
    - `_Py_HashPointer()`: use `Py_HashPointer()`.
    - `_Py_fopen_obj()`: use `Py_fopen()`.

    The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get these new public functions on Python 3.13 and older.

    Patch by Victor Stinner.
- gh-126599 [https://github.com/python/cpython/issues/126599]: Remove some internal test APIs for the experimental JIT compiler.
- gh-127925 [https://github.com/python/cpython/issues/127925]:

Convert the `decimal` module to use PEP 757 [https://peps.python.org/pep-0757/] C API (export-import integers), offering some speed-up if the integer part of the `Decimal` instance is small. Patch by Sergey B Kirpichev.

Build

- gh-129660 [https://github.com/python/cpython/issues/129660]: Drop `test_embed` from PGO training, whose contribution in recent versions is considered to be ignorable.
- gh-128902 [https://github.com/python/cpython/issues/128902]: Fix compile errors with Clang 9 and older due to lack of `__attribute__((fallthrough))` support.

Python 3.14.0 alpha 4

Release date: 2025-01-14

macOS

- gh-127592 [https://github.com/python/cpython/issues/127592]: Usage of the unified Apple System Log APIs was disabled when the minimum macOS version is earlier than 10.12.

Tools/Demos

- gh-128152 [https://github.com/python/cpython/issues/128152]: Fix a bug where Argument Clinic's C pre-processor parser tried to parse pre-processor directives inside C comments. Patch by Erlend Aasland.

Tests

- gh-128690 [https://github.com/python/cpython/issues/128690]: Temporarily do not use test_embed in PGO profile builds until the problem with test_init_pyvenv_cfg failing in some configurations is resolved.

Library

- gh-128731 [https://github.com/python/cpython/issues/128731]: Fix **`ResourceWarning`** in **`urllib.robotparser.RobotFileParser.read()`**.
- gh-71339 [https://github.com/python/cpython/issues/71339]: Add new assertion methods for **`unittest`**: **`assertHasAttr()`**, **`assertNotHasAttr()`**, **`assertIsSubclass()`**, **`assertNotIsSubclass()`** **`assertStartsWith()`**, **`assertNotStartsWith()`**, **`assertEndsWith()`** and **`assertNotEndsWith()`**.
- gh-118761 [https://github.com/python/cpython/issues/118761]: Improve import time of **`pickle`** by 25% by removing an unnecessary regular expression. As such, **`re`** is no more implicitly available as `pickle.re`. Patch by Bénédikt Tran.
- gh-128661 [https://github.com/python/cpython/issues/128661]: Fixes **`typing.evaluate_forward_ref()`** not showing deprecation when `type_params` arg is not passed.
- gh-128562 [https://github.com/python/cpython/issues/128562]: Fix possible conflicts in generated **`tkinter`** widget names if the widget class name ends with a digit.
- gh-128559 [https://github.com/python/cpython/issues/128559]: Improved import time of **`asyncio`**.
- gh-128552 [https://github.com/python/cpython/issues/128552]: Fix cyclic garbage introduced by **`asyncio.loop.create_task()`** and **`asyncio.TaskGroup.create_task()`** holding a reference to the created task if it is eager.
- gh-128340 [https://github.com/python/cpython/issues/128340]: Add internal thread safe handle to be used in **`asyncio.loop.call_soon_threadsafe()`** for thread safe cancellation.
- gh-128182 [https://github.com/python/cpython/issues/128182]: Fix crash when using **`ctypes`** pointers concurrently on the free threaded build.
- gh-128400 [https://github.com/python/cpython/issues/128400]: Only show the current thread in **`faulthandler`** on the free threaded build to prevent races.
- gh-128400 [https://github.com/python/cpython/issues/128400]: Fix crash when using **`faulthandler.dump_traceback()`** while other threads are active on the free threaded build.
- gh-128388 [https://github.com/python/cpython/issues/128388]: Fix

`PyREPL` on Windows to support more keybindings, like the `Control-←` and `Control-→` word-skipping keybindings and those with meta (i.e. `Alt`), e.g. `Alt-d` to `kill-word` or `Alt-Backspace` `backward-kill-word`.

- gh-88834 [https://github.com/python/cpython/issues/88834]: Unify the instance check for **`typing.Union`** and **`types.UnionType`**: **`Union`** now uses the instance checks against its parameters instead of the subclass checks.
- gh-128302 [https://github.com/python/cpython/issues/128302]: Fix **`xml.dom.xmlbuilder.DOMEntityResolver.resolveEntity(`** which was broken by the Python 3.0 transition.
- gh-128317 [https://github.com/python/cpython/issues/128317]: Highlight today in colour in **`calendar`**'s CLI output. Patch by Hugo van Kemenade.
- gh-128302 [https://github.com/python/cpython/issues/128302]: Allow **`xml.dom.xmlbuilder.DOMParser.parse()`** to correctly handle **`xml.dom.xmlbuilder.DOMInputSource`** instances that only have a **`systemId`** attribute set.
- gh-128151 [https://github.com/python/cpython/issues/128151]: Improve generation of **`UUID`** objects version 3, 4, 5, and 8 via their dedicated functions by 30%. Patch by Bénédikt Tran.
- gh-128118 [https://github.com/python/cpython/issues/128118]: Improve performance of **`copy.copy()`** by 30% via a fast path for atomic types and container types.
- gh-127946 [https://github.com/python/cpython/issues/127946]: Fix crash when modifying **`ctypes._CFuncPtr`** objects concurrently on the free threaded build.
- gh-128062 [https://github.com/python/cpython/issues/128062]: Revert the font of **`turtledemo`**'s menu bar to its default value and display the shortcut keys in the correct position.
- gh-128014 [https://github.com/python/cpython/issues/128014]: Fix resetting the default window icon by passing `default=''` to the **`tkinter`** method **`wm_iconbitmap()`**.
- gh-41872 [https://github.com/python/cpython/issues/41872]: Fix quick extraction of module docstrings from a file in **`pydoc`**. It now supports docstrings with single quotes, escape sequences, raw string literals, and other Python syntax.
- gh-127060 [https://github.com/python/cpython/issues/127060]: Set TERM environment variable to «dumb» to disable traceback colors in IDLE, since IDLE doesn't understand ANSI escape

sequences. Patch by Victor Stinner.

- gh-126742 [https://github.com/python/cpython/issues/126742]: Fix support of localized error messages reported by *dlerror(3)* *[https://manpages.debian.org/dlerror(3)]* and *gdbm_strerror* *[https://manpages.debian.org/gdbm(3)]* in `ctypes` and `dbm.gnu` functions respectively. Patch by Bénédikt Tran.
- gh-122548 [https://github.com/python/cpython/issues/122548]: Adds two new local events to sys.monitoring, `BRANCH_LEFT` and `BRANCH_RIGHT`. This allows the two arms of the branch to be disabled independently, which should hugely improve performance of branch-level coverage tools. The old branch event, `BRANCH` is now deprecated.
- gh-127847 [https://github.com/python/cpython/issues/127847]: Fix the position when doing interleaved seeks and reads in uncompressed, unencrypted zip files returned by `zipfile.ZipFile.open()`.
- gh-127688 [https://github.com/python/cpython/issues/127688]: Add the `SCHED_DEADLINE` and `SCHED_NORMAL` constants to the `os` module.
- gh-83662 [https://github.com/python/cpython/issues/83662]: Add missing `__class_getitem__` method to the Python implementation of `functools.partial()`, to make it compatible with the C version. This is mainly relevant for alternative Python implementations like PyPy and GraalPy, because CPython will usually use the C-implementation of that function.
- gh-127586 [https://github.com/python/cpython/issues/127586]: `multiprocessing.pool.Pool` now properly restores blocked signal handlers of the parent thread when creating processes via either *spawn* or *forkserver*.
- gh-98188 [https://github.com/python/cpython/issues/98188]: Fix an issue in `email.message.Message.get_payload()` where data cannot be decoded if the Content Transfer Encoding mechanism contains trailing whitespaces or additional junk text. Patch by Hui Liu.
- gh-127529 [https://github.com/python/cpython/issues/127529]: Correct behavior of `asyncio.selector_events.BaseSelectorEventLoop._acce` in handling `ConnectionAbortedError` in a loop. This improves performance on OpenBSD.

- gh-127360 [https://github.com/python/cpython/issues/127360]: When a descriptive error message cannot be provided for an **`ssl.SSLError`**, the «unknown error» message now shows the internal error code (as retrieved by `ERR_get_error` and similar OpenSSL functions).
- gh-127196 [https://github.com/python/cpython/issues/127196]: Fix crash when dict with keys in invalid encoding were passed to several functions in `_interpreters` module.
- gh-124130 [https://github.com/python/cpython/issues/124130]: Fix a bug in matching regular expression `\B` in empty input string. Now it is always the opposite of `\b`. To get an old behavior, use `(?!\A\Z)\B`. To get a new behavior in old Python versions, use `(?!\b)`.
- gh-126639 [https://github.com/python/cpython/issues/126639]: **`tempfile.NamedTemporaryFile`** will now issue a **`ResourceWarning`** when it is finalized by the garbage collector without being explicitly closed.
- gh-126624 [https://github.com/python/cpython/issues/126624]: Expose error code **`XML_ERROR_NOT_STARTED`** of Expat $>=$2.6.4 in **`xml.parsers.expat.errors`**.
- gh-126225 [https://github.com/python/cpython/issues/126225]: **`getopt`** and **`optparse`** are no longer marked as deprecated. There are legitimate reasons to use one of these modules in preference to **`argparse`**, and none of these modules are at risk of being removed from the standard library. Of the three, `argparse` remains the recommended default choice, *unless* one of the concerns noted at the top of the `optparse` module documentation applies.
- gh-124761 [https://github.com/python/cpython/issues/124761]: Add **`SO_REUSEPORT_LB`** constant to **`socket`** for FreeBSD.
- gh-121720 [https://github.com/python/cpython/issues/121720]: **`enum.EnumDict`** can now be used without resorting to private API.
- gh-123424 [https://github.com/python/cpython/issues/123424]: Add **`zipfile.ZipInfo._for_archive()`** setting default properties on **`ZipInfo`** objects. Patch by Bénédikt Tran and Jason R. Coombs.
- gh-121676 [https://github.com/python/cpython/issues/121676]: Deprecate calling the Python implementation of **`functools.reduce()`** with a `function` or `sequence` as

a keyword argument. This will be forbidden in Python 3.16 in order to match the C implementation.

- gh-112015 [https://github.com/python/cpython/issues/112015]: **`ctypes.memoryview_at()`** now exists to create a **`memoryview`** object that refers to the supplied pointer and length. This works like **`ctypes.string_at()`** except it avoids a buffer copy, and is typically useful when implementing pure Python callback functions that are passed dynamically-sized buffers.
- gh-95371 [https://github.com/python/cpython/issues/95371]: Added support for other image formats (PNG, PGM, and PPM) to the turtle module. Patch by Shin-myoung-serp.

Core and Builtins

- gh-128078 [https://github.com/python/cpython/issues/128078]: Fix a **`SystemError`** when using **`anext()`** with a default tuple value. Patch by Bénédikt Tran.
- gh-128717 [https://github.com/python/cpython/issues/128717]: Fix a crash when setting the recursion limit while other threads are active on the free threaded build.
- gh-124483 [https://github.com/python/cpython/issues/124483]: Treat `Py_DECREF` and variants as escaping when generating opcode and uop metadata. This prevents the possibility of a `__del__` method causing the JIT to behave incorrectly.
- gh-126703 [https://github.com/python/cpython/issues/126703]: Improve performance of class methods by using a freelist.
- gh-128137 [https://github.com/python/cpython/issues/128137]: Update **`PyASCIIObject`** layout to handle interned field with the atomic operation. Patch by Donghee Na.

Library

- gh-128192 [https://github.com/python/cpython/issues/128192]: Upgrade HTTP digest authentication algorithm for **`urllib.request`** by supporting SHA-256 digest authentication as specified in **RFC 7616** [https://datatracker.ietf.org/doc/html/rfc7616.html].

Core and Builtins

- gh-126868 [https://github.com/python/cpython/issues/126868]: Increase usage of freelist for **`int`** allocation.

- gh-114203 [https://github.com/python/cpython/issues/114203]: Optimize `Py_BEGIN_CRITICAL_SECTION` for simple recursive calls.

- gh-127705 [https://github.com/python/cpython/issues/127705]: Adds stackref debugging when `Py_STACKREF_DEBUG` is set. Finds all double-closes and leaks, logging the origin and last borrow.

  Inspired by HPy's debug mode. https://docs.hpyproject.org/en/latest/debug-mode.html

- gh-128079 [https://github.com/python/cpython/issues/128079]: Fix a bug where **`except*`** does not properly check the return value of an **`ExceptionGroup`**'s **`split()`** function, leading to a crash in some cases. Now when **`split()`** returns an invalid object, **`except*`** raises a **`TypeError`** with the original raised **`ExceptionGroup`** object chained to it.

- gh-128030 [https://github.com/python/cpython/issues/128030]: Avoid error from calling `PyModule_GetFilenameObject` on a non-module object when importing a non-existent symbol from a non-module object.

Library

- gh-128035 [https://github.com/python/cpython/issues/128035]: Indicate through **`ssl.HAS_PHA`** whether the **`ssl`** module supports TLSv1.3 post-handshake client authentication (PHA). Patch by Will Childs-Klein.

Core and Builtins

- gh-127274 [https://github.com/python/cpython/issues/127274]: Add a new flag, `CO_METHOD`, to **`co_flags`** that indicates whether the code object belongs to a function defined in class scope.
- gh-66409 [https://github.com/python/cpython/issues/66409]: During the path initialization, we now check if `base_exec_prefix`

is the same as `base_prefix` before falling back to searching the Python interpreter directory.

- gh-127970 [https://github.com/python/cpython/issues/127970]: We now use the location of the `libpython` runtime library used in the current process to determine **`sys.base_prefix`** on all platforms implementing the dladdr [https://pubs.opengroup.org/onlinepubs/9799919799/functions/dladdr.html] function defined by the UNIX standard — this includes Linux, Android, macOS, iOS, FreeBSD, etc. This was already the case on Windows and macOS Framework builds.
- gh-127773 [https://github.com/python/cpython/issues/127773]: Do not use the type attribute cache for types with incompatible MRO.
- gh-127903 [https://github.com/python/cpython/issues/127903]: `Objects/unicodeobject.c`: fix a crash on DEBUG builds in `_copy_characters` when there is nothing to copy.
- gh-127809 [https://github.com/python/cpython/issues/127809]: Fix an issue where the experimental JIT may infer an incorrect result type for exponentiation (`**` and `**=`), leading to bugs or crashes.
- gh-126862 [https://github.com/python/cpython/issues/126862]: Fix a possible overflow when a class inherits from an absurd number of super-classes. Reported by Valery Fedorenko. Patch by Bénédikt Tran.

C API

- gh-128400 [https://github.com/python/cpython/issues/128400]: **`Py_FatalError()`** no longer shows all threads on the free threaded build to prevent crashes.
- gh-128629 [https://github.com/python/cpython/issues/128629]: Add macros **`Py_PACK_VERSION()`** and **`Py_PACK_FULL_VERSION()`** for bit-packing Python version numbers.
- gh-128008 [https://github.com/python/cpython/issues/128008]: Add **`PyWeakref_IsDead()`** function, which tests if a weak reference is dead.
- gh-127350 [https://github.com/python/cpython/issues/127350]: Add **`Py_fopen()`** function to open a file. Similar to the **`fopen()`** function, but the *path* parameter is a Python object and an

exception is set on error. Add also **`Py_fclose()`** function to close a file, function needed for Windows support. Patch by Victor Stinner.

Build

- gh-128627 [https://github.com/python/cpython/issues/128627]: For Emscripten builds the function pointer cast call trampoline now uses the wasm-gc ref.test instruction if it's available instead of Wasm JS type reflection.
- gh-128472 [https://github.com/python/cpython/issues/128472]: Skip BOLT optimization of functions using computed gotos, fixing errors on build with LLVM 19.
- gh-115765 [https://github.com/python/cpython/issues/115765]: GNU Autoconf 2.72 is now required to generate `configure`. Patch by Erlend Aasland.
- gh-123925 [https://github.com/python/cpython/issues/123925]: Fix building the **`curses`** module on platforms with libncurses but without libncursesw.
- gh-90905 [https://github.com/python/cpython/issues/90905]: Add support for cross-compiling to x86_64 on aarch64/arm64 macOS.
- gh-128321 [https://github.com/python/cpython/issues/128321]: Set `LIBS` instead of `LDFLAGS` when checking if **`sqlite3`** library functions are available. This fixes the ordering of linked libraries during checks, which was incorrect when using a statically linked `libsqlite3`.
- gh-100384 [https://github.com/python/cpython/issues/100384]: Error on `unguarded-availability` in macOS builds, preventing invalid use of symbols that are not available in older versions of the OS.
- gh-128104 [https://github.com/python/cpython/issues/128104]: Remove `Py_STRFTIME_C99_SUPPORT` conditions in favor of requiring C99 *strftime(3)* *[https://manpages.debian.org/strftime(3)]* specifier support at build time. When cross-compiling, there is no build time check and support is assumed.
- gh-127951 [https://github.com/python/cpython/issues/127951]: Add option `--pystats` to the Windows build to enable performance statistics collection.

Python 3.14.0 alpha 3

Release date: 2024-12-17

Windows

- gh-127353 [https://github.com/python/cpython/issues/127353]: Allow to force color output on Windows using environment variables. Patch by Andrey Efremov.
- gh-125729 [https://github.com/python/cpython/issues/125729]: Makes the presence of the `turtle` module dependent on the Tcl/Tk installer option. Previously, the module was always installed but would be unusable without Tcl/Tk.

Tools/Demos

- gh-126700 [https://github.com/python/cpython/issues/126700]: Add support for multi-argument `gettext` functions in **pygettext.py**.

Tests

- gh-127906 [https://github.com/python/cpython/issues/127906]: Test the limited C API in test_cppext. Patch by Victor Stinner.
- gh-127637 [https://github.com/python/cpython/issues/127637]: Add tests for the `dis` command-line interface. Patch by Bénédikt Tran.
- gh-126925 [https://github.com/python/cpython/issues/126925]: iOS test results are now streamed during test execution, and the deprecated xcresulttool is no longer used.
- gh-127076 [https://github.com/python/cpython/issues/127076]: Disable strace based system call tests when LD_PRELOAD is set.
- gh-127076 [https://github.com/python/cpython/issues/127076]: Filter out memory-related `mmap`, `munmap`, and `mprotect` calls from file-related ones when testing `io` behavior using strace.

Security

- gh-127655 [https://github.com/python/cpython/issues/127655]: Fixed the `asyncio.selector_events._SelectorSocketTransport` transport not pausing writes for the protocol when the buffer reaches the high water mark when using `asyncio.WriteTransport.writelines()`.

Library

- gh-126907 [https://github.com/python/cpython/issues/126907]: Fix crash when using `atexit` concurrently on the free-threaded build.
- gh-127870 [https://github.com/python/cpython/issues/127870]: Detect recursive calls in ctypes `_as_parameter_` handling. Patch by Victor Stinner.
- gh-127732 [https://github.com/python/cpython/issues/127732]: The `platform` module now correctly detects Windows Server 2025.
- gh-126789 [https://github.com/python/cpython/issues/126789]: Fixed `sysconfig.get_config_vars()`, `sysconfig.get_paths()`, and siblings, returning outdated cached data if the value of `sys.prefix` or `sys.exec_prefix` changes. Overwriting `sys.prefix` or `sys.exec_prefix` still is discouraged, as that might break other parts of the code.
- gh-127718 [https://github.com/python/cpython/issues/127718]: Add colour to `test.regrtest` output. Patch by Hugo van Kemenade.
- gh-127610 [https://github.com/python/cpython/issues/127610]: Added validation for more than one var-positional or var-keyword parameters in `inspect.Signature`. Patch by Maxim Ageev.
- gh-127627 [https://github.com/python/cpython/issues/127627]: Added `posix._emscripten_debugger()` to help with debugging the test suite on the Emscripten target.

- gh-126821 [https://github.com/python/cpython/issues/126821]: macOS and iOS apps can now choose to redirect stdout and stderr to the system log during interpreter configuration.
- gh-93312 [https://github.com/python/cpython/issues/93312]: Include `<sys/pidfd.h>` to get `os.PIDFD_NONBLOCK` constant. Patch by Victor Stinner.
- gh-127481 [https://github.com/python/cpython/issues/127481]: Add the `EPOLLWAKEUP` constant to the **`select`** module.
- gh-127065 [https://github.com/python/cpython/issues/127065]: Make **`operator.methodcaller()`** thread-safe and re-entrant safe.
- gh-127321 [https://github.com/python/cpython/issues/127321]: **`pdb.set_trace()`** will not stop at an opcode that does not have an associated line number anymore.
- gh-127429 [https://github.com/python/cpython/issues/127429]: Fixed bug where, on cross-builds, the **`sysconfig`** POSIX data was being generated with the host Python's `Makefile`. The data is now generated from current build's `Makefile`.
- gh-127413 [https://github.com/python/cpython/issues/127413]: Add the **`dis --specialized`** command-line option to show specialized bytecode. Patch by Bénédikt Tran.
- gh-125413 [https://github.com/python/cpython/issues/125413]: Revert addition of **`pathlib.Path.scandir()`**. This method was added in 3.14.0a2. The optimizations remain for file system paths, but other subclasses should only have to implement **`pathlib.Path.iterdir()`**.
- gh-127257 [https://github.com/python/cpython/issues/127257]: In **`ssl`**, system call failures that OpenSSL reports using `ERR_LIB_SYS` are now raised as **`OSError`**.
- gh-59705 [https://github.com/python/cpython/issues/59705]: On Linux, **`threading.Thread`** now sets the thread name to the operating system. Patch by Victor Stinner.

- gh-127303 [https://github.com/python/cpython/issues/127303]: Publicly expose `EXACT_TOKEN_TYPES` in `token.__all__`.

- gh-127331 [https://github.com/python/cpython/issues/127331]: `ssl` can show descriptions for errors added in OpenSSL 3.4.

- gh-123967 [https://github.com/python/cpython/issues/123967]: Fix faulthandler for trampoline frames. If the top-most frame is a trampoline frame, skip it. Patch by Victor Stinner.

- gh-127178 [https://github.com/python/cpython/issues/127178]: A `_sysconfig_vars_(...).json` file is now shipped in the standard library directory. It contains the output of `sysconfig.get_config_vars()` on the default environment encoded as JSON data. This is an implementation detail, and may change at any time.

- gh-127072 [https://github.com/python/cpython/issues/127072]: Remove outdated `socket.NETLINK_*` constants not present in Linux kernels beyond 2.6.17.

- gh-127255 [https://github.com/python/cpython/issues/127255]: The `CopyComPointer()` function is now public. Previously, this was private and only available in `_ctypes`.

- gh-127182 [https://github.com/python/cpython/issues/127182]: Fix `io.StringIO.__setstate__()` crash, when `None` was passed as the first value.

- gh-127217 [https://github.com/python/cpython/issues/127217]: Fix `urllib.request.pathname2url()` for paths starting with multiple slashes on Posix.

- gh-125866 [https://github.com/python/cpython/issues/125866]: `urllib.request.pathname2url()` now adds an empty authority when generating a URL for a path that begins with exactly one slash. For example, the path `/etc/hosts` is converted to the scheme-less URL `///etc/hosts`. As a result of this change, URLs without authorities are only generated for relative paths.

- gh-127221 [https://github.com/python/cpython/issues/127221]: Add colour to `unittest` output. Patch by Hugo van Kemenade.

- gh-127035 [https://github.com/python/cpython/issues/127035]: Fix `shutil.which` on Windows. Now it looks at direct match if and only if the command ends with a PATHEXT extension or X_OK is not in mode. Support extensionless files if «.» is in PATHEXT. Support PATHEXT extensions that end with a dot.

- gh-122273 [https://github.com/python/cpython/issues/122273]: Support PyREPL history on Windows. Patch by devdanzin and Victor Stinner.

- gh-125866 [https://github.com/python/cpython/issues/125866]: `urllib.request.pathname2url()` and `url2pathname()` no longer convert Windows drive letters to uppercase.

- gh-127078 [https://github.com/python/cpython/issues/127078]: Fix issue where `urllib.request.url2pathname()` failed to discard an extra slash before a UNC drive in the URL path on Windows.

- gh-126766 [https://github.com/python/cpython/issues/126766]: Fix issue where `urllib.request.url2pathname()` failed to discard any “localhost” authority present in the URL.

- gh-127065 [https://github.com/python/cpython/issues/127065]: Fix crash when calling a `operator.methodcaller()` instance from multiple threads in the free threading build.

- gh-127090 [https://github.com/python/cpython/issues/127090]: Fix value of `urllib.response.addinfourl.url` for `file:` URLs that express relative paths and absolute Windows paths. The canonical URL generated by `urllib.request.pathname2url()` is now used.

- gh-126992 [https://github.com/python/cpython/issues/126992]: Fix LONG and INT opcodes to only use base 10 for string to integer conversion in `pickle`.

- gh-126997 [https://github.com/python/cpython/issues/126997]: Fix support of STRING and GLOBAL opcodes with non-ASCII arguments in `pickletools`. `pickletools.dis()` now outputs non-ASCII bytes in STRING, BINSTRING and SHORT_BINSTRING arguments as escaped (`\xXX`).

- gh-126316 [https://github.com/python/cpython/issues/126316]: `grp`: Make `grp.getgrall()` thread-safe by adding a mutex. Patch by Victor Stinner.

- gh-126618 [https://github.com/python/cpython/issues/126618]: Fix the representation of `itertools.count` objects when the count value is `sys.maxsize`.

- gh-126615 [https://github.com/python/cpython/issues/126615]: The `COMError` exception is now public. Previously, this was private and only available in `_ctypes`.

- gh-126985 [https://github.com/python/cpython/issues/126985]: When running under a virtual environment with the `site` disabled (see `-S`), `sys.prefix` and `sys.base_prefix` will now point to the virtual environment, instead of the base installation.

- gh-112192 [https://github.com/python/cpython/issues/112192]: In the `trace` module, increase the coverage precision (`cov%`) to one decimal.

- gh-118761 [https://github.com/python/cpython/issues/118761]: Improve import time of `mimetypes` by around 11-16 times. Patch by Hugo van Kemenade.

- gh-126947 [https://github.com/python/cpython/issues/126947]: Raise `TypeError` in `_pydatetime.timedelta.__new__()` if the passed arguments are not `int` or `float`, so that the Python implementation is in line with the C implementation.

- gh-126946 [https://github.com/python/cpython/issues/126946]: Improve the `GetoptError` error message when a long option prefix matches multiple accepted options in `getopt.getopt()` and `getopt.gnu_getopt()`.

- gh-126899 [https://github.com/python/cpython/issues/126899]: Make tkinter widget methods **`after()`** and **`after_idle()`** accept arguments passed by keyword.

- gh-85168 [https://github.com/python/cpython/issues/85168]: Fix issue where **`urllib.request.url2pathname()`** and **`pathname2url()`** always used UTF-8 when quoting and unquoting file URIs. They now use the filesystem encoding and error handler.

- gh-126780 [https://github.com/python/cpython/issues/126780]: Fix **`os.path.normpath()`** for drive-relative paths on Windows.

- gh-126775 [https://github.com/python/cpython/issues/126775]: Make **`linecache.checkcache()`** thread safe and GC re-entrancy safe.

- gh-126601 [https://github.com/python/cpython/issues/126601]: Fix issue where **`urllib.request.pathname2url()`** raised **`OSError`** when given a Windows path containing a colon character not following a drive letter, such as before an NTFS alternate data stream.

- gh-126727 [https://github.com/python/cpython/issues/126727]: `locale.nl_langinfo(locale.ERA)` now returns multiple era description segments separated by semicolons. Previously it only returned the first segment on platforms with Glibc.

- gh-85046 [https://github.com/python/cpython/issues/85046]: Add **`EHWPOISON`** error code to **`errno`**.

- gh-118201 [https://github.com/python/cpython/issues/118201]: Fixed intermittent failures of **`os.confstr`**, **`os.pathconf`** and **`os.sysconf`** on iOS and Android.

- gh-86463 [https://github.com/python/cpython/issues/86463]: The `usage` parameter of **`argparse.ArgumentParser`** no longer affects the default value of the `prog` parameter in subparsers.

- gh-124008 [https://github.com/python/cpython/issues/124008]: Fix

possible crash (in debug build), incorrect output or returning incorrect value from raw binary `write()` when writing to console on Windows.

- gh-123401 [https://github.com/python/cpython/issues/123401]: The **`http.cookies`** module now supports parsing obsolete **RFC 850** [https://datatracker.ietf.org/doc/html/rfc850.html] date formats, in accordance with **RFC 9110** [https://datatracker.ietf.org/doc/html/rfc9110.html] requirements. Patch by Nano Zheng.

- gh-122431 [https://github.com/python/cpython/issues/122431]: **`readline.append_history_file()`** now raises a **`ValueError`** when given a negative value.

- gh-122356 [https://github.com/python/cpython/issues/122356]: Guarantee that the position of a file-like object passed to **`zipfile.is_zipfile()`** is left untouched after the call. Patch by Bénédikt Tran.

- gh-122288 [https://github.com/python/cpython/issues/122288]: Improve the performances of **`fnmatch.translate()`** by a factor 1.7. Patch by Bénédikt Tran.

- gh-88110 [https://github.com/python/cpython/issues/88110]: Fixed **`multiprocessing.Process`** reporting a `.exitcode` of 1 even on success when using the `"fork"` start method while using a **`concurrent.futures.ThreadPoolExecutor`**.

- gh-97514 [https://github.com/python/cpython/issues/97514]: Authentication was added to the **`multiprocessing`** forkserver start method control socket so that only processes with the authentication key generated by the process that spawned the forkserver can control it. This is an enhancement over the other gh-97514 [https://github.com/python/cpython/issues/97514] fixes so that access is no longer limited only by filesystem permissions.

  The file descriptor exchange of control pipes with the forked worker process now requires an explicit acknowledgement byte to be sent over the socket after the exchange on all forkserver supporting platforms. That makes testing the above much easier.

Documentation

- gh-127347 [https://github.com/python/cpython/issues/127347]: Publicly expose `traceback.print_list()` in **`traceback.__all__`**.

Core and Builtins

- gh-127740 [https://github.com/python/cpython/issues/127740]: Fix error message in `bytes.fromhex()` when given an odd number of digits to properly indicate that an even number of hexadecimal digits is required.
- gh-127058 [https://github.com/python/cpython/issues/127058]: `PySequence_Tuple` now creates the resulting tuple atomically, preventing partially created tuples being visible to the garbage collector or through `gc.get_referrers()`
- gh-127599 [https://github.com/python/cpython/issues/127599]: Fix statistics for increments of object reference counts (in particular, when a reference count was increased by more than 1 in a single operation).
- gh-127651 [https://github.com/python/cpython/issues/127651]: When raising `ImportError` for missing symbols in `from` imports, use `__file__` in the error message if `__spec__.origin` is not a location
- gh-127582 [https://github.com/python/cpython/issues/127582]: Fix non-thread-safe object resurrection when calling finalizers and watcher callbacks in the free threading build.
- gh-127434 [https://github.com/python/cpython/issues/127434]: The iOS compiler shims can now accept arguments with spaces.
- gh-127536 [https://github.com/python/cpython/issues/127536]: Add missing locks around some list assignment operations in the free threading build.
- gh-127085 [https://github.com/python/cpython/issues/127085]: Fix race when exporting a buffer from a `memoryview` object on the free-threaded build.
- gh-127238 [https://github.com/python/cpython/issues/127238]: Correct error message for `sys.set_int_max_str_digits()`.

- gh-113841 [https://github.com/python/cpython/issues/113841]: Fix possible undefined behavior division by zero in `complex`’s `_Py_c_pow()`.

Library

- gh-127133 [https://github.com/python/cpython/issues/127133]: Calling `argparse.ArgumentParser.add_argument_group()` on an argument group, and calling `argparse.ArgumentParser.add_argument_group()` or `argparse.ArgumentParser.add_mutually_exclusive_grou` on a mutually exclusive group now raise exceptions. This nesting was never supported, often failed to work correctly, and was unintentionally exposed through inheritance. This functionality has been deprecated since Python 3.11.

Core and Builtins

- gh-126491 [https://github.com/python/cpython/issues/126491]: Add a marking phase to the GC. All objects that can be transitively reached from builtin modules or the stacks are marked as reachable before cycle detection. This reduces the amount of work done by the GC by approximately half.
- gh-127020 [https://github.com/python/cpython/issues/127020]: Fix a crash in the free threading build when `PyCode_GetCode()`, `PyCode_GetVarnames()`, `PyCode_GetCellvars()`, or `PyCode_GetFreevars()` were called from multiple threads at the same time.
- gh-127010 [https://github.com/python/cpython/issues/127010]: Simplify GC tracking of dictionaries. All dictionaries are tracked when created, rather than being lazily tracked when a trackable object was added to them. This simplifies the code considerably and results in a slight speedup.
- gh-126980 [https://github.com/python/cpython/issues/126980]: Fix `__buffer__()` of `bytearray` crashing when `READ` or `WRITE` are passed as flags.
- gh-126937 [https://github.com/python/cpython/issues/126937]: Fix `TypeError` when a `ctypes.Structure` has a field size that doesn’t fit into an unsigned 16-bit integer. Instead, the

maximum number of *bits* is `sys.maxsize`.

- gh-126868 [https://github.com/python/cpython/issues/126868]: Increase performance of `int` by adding a freelist for compact ints.
- gh-126881 [https://github.com/python/cpython/issues/126881]: Fix crash in finalization of dtoa state. Patch by Kumar Aditya.
- gh-126892 [https://github.com/python/cpython/issues/126892]: Require cold or invalidated code to «warm up» before being JIT compiled again.
- gh-126091 [https://github.com/python/cpython/issues/126091]: Ensure stack traces are complete when throwing into a generator chain that ends in a custom generator.
- gh-126024 [https://github.com/python/cpython/issues/126024]: Optimize decoding of short UTF-8 sequences containing non-ASCII characters by approximately 15%.
- gh-125420 [https://github.com/python/cpython/issues/125420]: Add `memoryview.index()` to `memoryview` objects. Patch by Bénédikt Tran.
- gh-125420 [https://github.com/python/cpython/issues/125420]: Add `memoryview.count()` to `memoryview` objects. Patch by Bénédikt Tran.
- gh-124470 [https://github.com/python/cpython/issues/124470]: Fix crash in free-threaded builds when replacing object dictionary while reading attribute on another thread
- gh-69639 [https://github.com/python/cpython/issues/69639]: Implement mixed-mode arithmetic rules combining real and complex numbers as specified by C standards since C99. Patch by Sergey B Kirpichev.
- gh-120010 [https://github.com/python/cpython/issues/120010]: Correct invalid corner cases which resulted in `(nan+nanj)` output in complex multiplication, e.g., `(1e300+1j)*(nan+infj)`. Patch by Sergey B Kirpichev.
- gh-109746 [https://github.com/python/cpython/issues/109746]: If `_thread.start_new_thread()` fails to start a new thread, it deletes its state from interpreter and thus avoids its repeated cleanup on finalization.

C API

- gh-127896 [https://github.com/python/cpython/issues/127896]: The

previously undocumented function `PySequence_In()` is soft deprecated. Use `PySequence_Contains()` instead.

- gh-127791 [https://github.com/python/cpython/issues/127791]: Fix loss of callbacks after more than one call to `PyUnstable_AtExit()`.

- gh-127691 [https://github.com/python/cpython/issues/127691]: The Unicode Exception Objects C API now raises a `TypeError` if its exception argument is not a `UnicodeError` object. Patch by Bénédikt Tran.

- gh-123378 [https://github.com/python/cpython/issues/123378]: Ensure that the value of `UnicodeEncodeError.end` retrieved by `PyUnicodeEncodeError_GetEnd()` lies in `[min(1, objlen), max(min(1, objlen), objlen)]` where *objlen* is the length of `UnicodeEncodeError.object`. Similar arguments apply to `UnicodeDecodeError` and `UnicodeTranslateError` and their corresponding C interface. Patch by Bénédikt Tran.

- gh-127314 [https://github.com/python/cpython/issues/127314]: Improve error message when calling the C API without an active thread state on the free-threaded build.

- gh-123378 [https://github.com/python/cpython/issues/123378]: Ensure that the value of `UnicodeEncodeError.start` retrieved by `PyUnicodeEncodeError_GetStart()` lies in `[0, max(0, objlen - 1)]` where *objlen* is the length of `UnicodeEncodeError.object`. Similar arguments apply to `UnicodeDecodeError` and `UnicodeTranslateError` and their corresponding C interface. Patch by Bénédikt Tran.

- gh-109523 [https://github.com/python/cpython/issues/109523]: Reading text from a non-blocking stream with `read` may now raise a `BlockingIOError` if the operation cannot immediately return bytes.

- gh-102471 [https://github.com/python/cpython/issues/102471]: Add a new import and export API for Python `int` objects (**PEP 757** [https://peps.python.org/pep-0757/]):

    - **`PyLong_GetNativeLayout()`**;
    - **`PyLong_Export()`**;
    - **`PyLong_FreeExport()`**;
    - **`PyLongWriter_Create()`**;
    - **`PyLongWriter_Finish()`**;
    - **`PyLongWriter_Discard()`**.

  Patch by Victor Stinner.

- gh-121058 [https://github.com/python/cpython/issues/121058]: `PyThreadState_Clear()` now warns (and calls `sys.excepthook`) if the thread state still has an active exception.

Build

- gh-127865 [https://github.com/python/cpython/issues/127865]: Fix build failure on systems without thread-locals support.
- gh-127629 [https://github.com/python/cpython/issues/127629]: Emscripten builds now include ctypes support.
- gh-127111 [https://github.com/python/cpython/issues/127111]: Updated the Emscripten web example to use ES6 modules and be built into a distinct `web_example` subfolder.
- gh-115869 [https://github.com/python/cpython/issues/115869]: Make `jit_stencils.h` (which is produced during JIT builds) reproducible.
- gh-126898 [https://github.com/python/cpython/issues/126898]: The Emscripten build of Python is now based on ES6 modules.

Python 3.14.0 alpha 2

Release date: 2024-11-19

Windows

- gh-126911 [https://github.com/python/cpython/issues/126911]: Update credits command output.
- gh-118973 [https://github.com/python/cpython/issues/118973]: Ensures the experimental free-threaded install includes the

`_tkinter` module. The optional Tcl/Tk component must also be installed in order for the module to work.

- gh-126497 [https://github.com/python/cpython/issues/126497]: Fixes venv failure due to missing redirector executables in experimental free-threaded installs.
- gh-126074 [https://github.com/python/cpython/issues/126074]: Removed unnecessary DLLs from Windows embeddable package
- gh-125315 [https://github.com/python/cpython/issues/125315]: Avoid crashing in `platform` due to slow WMI calls on some Windows machines.
- gh-126084 [https://github.com/python/cpython/issues/126084]: Fix venvwlauncher to launch pythonw instead of python so no extra console window is created.
- gh-125842 [https://github.com/python/cpython/issues/125842]: Fix a `SystemError` when `sys.exit()` is called with `0xffffffff` on Windows.
- gh-125550 [https://github.com/python/cpython/issues/125550]: Enable the Python install manager to detect Python 3.14 installs from the Windows Store.
- gh-123803 [https://github.com/python/cpython/issues/123803]: All Windows code pages are now supported as «cpXXX» codecs on Windows.

Tools/Demos

- gh-126807 [https://github.com/python/cpython/issues/126807]: Fix extraction warnings in **pygettext.py** caused by mistaking function definitions for function calls.
- gh-126167 [https://github.com/python/cpython/issues/126167]: The iOS testbed was modified so that it can be used by third-party projects for testing purposes.

Tests

- gh-126909 [https://github.com/python/cpython/issues/126909]: Fix test_os extended attribute tests to work on filesystems with 1 KiB xattr size limit.
- gh-125730 [https://github.com/python/cpython/issues/125730]: Change `make test` to not run GUI tests by default. Use

`make ci` to run tests with GUI tests instead.

- gh-124295 [https://github.com/python/cpython/issues/124295]: Add translation tests to the **`argparse`** module.

Security

- gh-126623 [https://github.com/python/cpython/issues/126623]: Upgrade libexpat to 2.6.4

Library

- gh-85957 [https://github.com/python/cpython/issues/85957]: Add missing MIME types for images with RFCs: emf, fits, g3fax, jp2, jpm, jpx, t38, tiff-fx and wmf. Patch by Hugo van Kemenade.

- gh-126920 [https://github.com/python/cpython/issues/126920]: Fix the `prefix` and `exec_prefix` keys from **`sysconfig.get_config_vars()`** incorrectly having the same value as **`sys.base_prefix`** and **`sys.base_exec_prefix`**, respectively, inside virtual environments. They now accurately reflect **`sys.prefix`** and **`sys.exec_prefix`**.

- gh-67877 [https://github.com/python/cpython/issues/67877]: Fix memory leaks when **`regular expression`** matching terminates abruptly, either because of a signal or because memory allocation fails.

- gh-125063 [https://github.com/python/cpython/issues/125063]: **`marshal`** now supports **`slice`** objects. The marshal format version was increased to 5.

- gh-126789 [https://github.com/python/cpython/issues/126789]: Fixed the values of **`sysconfig.get_config_vars()`**, **`sysconfig.get_paths()`**, and their siblings when the **`site`** initialization happens after **`sysconfig`** has built a cache for **`sysconfig.get_config_vars()`**.

- gh-126188 [https://github.com/python/cpython/issues/126188]: Update bundled pip to 24.3.1

- gh-126766 [https://github.com/python/cpython/issues/126766]: Fix issue where `urllib.request.url2pathname()` failed to discard two leading slashes introducing an empty authority section.

- gh-126705 [https://github.com/python/cpython/issues/126705]: Allow `os.PathLike` to be a base for Protocols.

- gh-126699 [https://github.com/python/cpython/issues/126699]: Allow `collections.abc.AsyncIterator` to be a base for Protocols.

- gh-126654 [https://github.com/python/cpython/issues/126654]: Fix crash when non-dict was passed to several functions in `_interpreters` module.

- gh-104745 [https://github.com/python/cpython/issues/104745]: Limit starting a patcher (from `unittest.mock.patch()` or `unittest.mock.patch.object()`) more than once without stopping it

- gh-126595 [https://github.com/python/cpython/issues/126595]: Fix a crash when instantiating `itertools.count` with an initial count of `sys.maxsize` on debug builds. Patch by Bénédikt Tran.

- gh-120423 [https://github.com/python/cpython/issues/120423]: Fix issue where `urllib.request.pathname2url()` mishandled Windows paths with embedded forward slashes.

- gh-126565 [https://github.com/python/cpython/issues/126565]: Improve performances of `zipfile.Path.open()` for non-reading modes.

- gh-126505 [https://github.com/python/cpython/issues/126505]: Fix bugs in compiling case-insensitive `regular expressions` with character classes containing non-BMP characters: upper-case non-BMP character did was ignored and the ASCII flag was ignored when matching a character range whose upper bound is beyond the BMP region.

- gh-117378 [https://github.com/python/cpython/issues/117378]: Fixed the `multiprocessing` `"forkserver"` start method forkserver process to correctly inherit the parent's `sys.path` during the importing of `multiprocessing.set_forkserver_preload()` modules in the same manner as **`sys.path`** is configured in workers before executing work items.

  This bug caused some forkserver module preloading to silently fail to preload. This manifested as a performance degradation in child processes when the `sys.path` was required due to additional repeated work in every worker.

  It could also have a side effect of `""` remaining in `sys.path` during forkserver preload imports instead of the absolute path from `os.getcwd()` at multiprocessing import time used in the worker `sys.path`.

  The `sys.path` differences between phases in the child process could potentially have caused preload to import incorrect things from the wrong location. We are unaware of that actually having happened in practice.

- gh-125679 [https://github.com/python/cpython/issues/125679]: The `multiprocessing.Lock` and `multiprocessing.RLock` `repr` values no longer say «unknown» on macOS.

- gh-126476 [https://github.com/python/cpython/issues/126476]: Raise `calendar.IllegalMonthError` (now a subclass of `IndexError`) for `calendar.month()` when the input month is not correct.

- gh-126489 [https://github.com/python/cpython/issues/126489]: The Python implementation of `pickle` no longer calls `pickle.Pickler.persistent_id()` for the result of **`persistent_id()`**.

- gh-126451 [https://github.com/python/cpython/issues/126451]: Register the `contextvars.Context` type to `collections.abc.Mapping`.

- gh-126175 [https://github.com/python/cpython/issues/126175]: Add `msg`, `doc`, `pos`, `lineno` and `colno` attributes to `tomllib.TOMLDecodeError`. Deprecate instantiating with free-form arguments.

- gh-89416 [https://github.com/python/cpython/issues/89416]: Add **RFC 9559** [https://datatracker.ietf.org/doc/html/rfc9559.html] MIME types for Matroska audiovisual container formats. Patch by Hugo van Kemenade.

- gh-126417 [https://github.com/python/cpython/issues/126417]: Register the `multiprocessing.managers.DictProxy` and `multiprocessing.managers.ListProxy` types in `multiprocessing.managers` to `collections.abc.MutableMapping` and `collections.abc.MutableSequence`, respectively.

- gh-126390 [https://github.com/python/cpython/issues/126390]: Add support for returning intermixed options and non-option arguments in order in `getopt.gnu_getopt()`.

- gh-126374 [https://github.com/python/cpython/issues/126374]: Add support for options with optional arguments in the `getopt` module.

- gh-126363 [https://github.com/python/cpython/issues/126363]: Speed up pattern parsing in `pathlib.Path.glob()` by skipping creation of a `pathlib.Path` object for the pattern.

- gh-126353 [https://github.com/python/cpython/issues/126353]: `asyncio.get_event_loop()` now does not implicitly creates an event loop. It now raises a `RuntimeError` if there is no set event loop. Patch by Kumar Aditya.

- gh-126313 [https://github.com/python/cpython/issues/126313]: Fix an issue in `curses.napms()` when `curses.initscr()` has not yet been called. Patch by Bénédikt Tran.

- gh-126303 [https://github.com/python/cpython/issues/126303]: Fix pickling and copying of `os.sched_param` objects.

- gh-126138 [https://github.com/python/cpython/issues/126138]: Fix a use-after-free crash on `asyncio.Task` objects whose underlying coroutine yields an object that implements an evil `__getattribute__()`. Patch by Nico Posada.

- gh-120057 [https://github.com/python/cpython/issues/120057]: Replace the `os.environ.refresh()` method with a new `os.reload_environ()` function. Patch by Victor Stinner.

- gh-126220 [https://github.com/python/cpython/issues/126220]: Fix crash in `cProfile.Profile` and `_lsprof.Profiler` when their callbacks were directly called with 0 arguments.

- gh-126212 [https://github.com/python/cpython/issues/126212]: Fix issue where `urllib.request.pathname2url()` and `url2pathname()` removed slashes from Windows DOS drive paths and URLs.

- gh-126223 [https://github.com/python/cpython/issues/126223]: Raise a `UnicodeEncodeError` instead of a `SystemError` upon calling `_interpreters.create()` with an invalid Unicode character.

- gh-126205 [https://github.com/python/cpython/issues/126205]: Fix issue where `urllib.request.pathname2url()` generated URLs beginning with four slashes (rather than two) when given a Windows UNC path.

- gh-126156 [https://github.com/python/cpython/issues/126156]: Improved performances of creating `Morsel` objects by a factor of 3.8x.

- gh-126105 [https://github.com/python/cpython/issues/126105]: Fix a crash in `ast` when the `ast.AST._fields` attribute is deleted.

- gh-126106 [https://github.com/python/cpython/issues/126106]: Fixes a possible `NULL` pointer dereference in `ssl`.

- gh-126080 [https://github.com/python/cpython/issues/126080]: Fix a use-after-free crash on `asyncio.Task` objects for which the underlying event loop implements an evil **`__getattribute__()`**. Reported by Nico-Posada. Patch by Bénédikt Tran.

- gh-125322 [https://github.com/python/cpython/issues/125322]: Correct detection of complex numbers support in libffi.
- gh-126083 [https://github.com/python/cpython/issues/126083]: Fixed a reference leak in **`asyncio.Task`** objects when reinitializing the same object with a non-`None` context. Patch by Nico Posada.
- gh-126068 [https://github.com/python/cpython/issues/126068]: Fix exceptions in the **`argparse`** module so that only error messages for ArgumentError and ArgumentTypeError are now translated. ArgumentError is now only used for command line errors, not for logical errors in the program. TypeError is now raised instead of ValueError for some logical errors.
- gh-125413 [https://github.com/python/cpython/issues/125413]: Add **`pathlib.Path.scandir()`** method to efficiently fetch directory children and their file attributes. This is a trivial wrapper of **`os.scandir()`**.
- gh-125984 [https://github.com/python/cpython/issues/125984]: Fix use-after-free crashes on **`asyncio.Future`** objects for which the underlying event loop implements an evil **`__getattribute__()`**. Reported by Nico-Posada. Patch by Bénédikt Tran.
- gh-125926 [https://github.com/python/cpython/issues/125926]: Fix **`urllib.parse.urljoin()`** for base URI with undefined authority. Although **RFC 3986** [https://datatracker.ietf.org/doc/html/rfc3986.html] only specify reference resolution for absolute base URI, **`urljoin()`** should continue to return sensible result for relative base URI.
- gh-125969 [https://github.com/python/cpython/issues/125969]: Fix an out-of-bounds crash when an evil **`asyncio.loop.call_soon()`** mutates the length of the internal callbacks list. Patch by Bénédikt Tran.

- gh-125966 [https://github.com/python/cpython/issues/125966]: Fix a use-after-free crash in `asyncio.Future.remove_done_callback()`. Patch by Bénédikt Tran.

- gh-125789 [https://github.com/python/cpython/issues/125789]: Fix possible crash when mutating list of callbacks returned by **`asyncio.Future._callbacks`**. It now always returns a new copy in C implementation **`_asyncio`**. Patch by Kumar Aditya.

- gh-126916 [https://github.com/python/cpython/issues/126916]: Allow the *initial* parameter of `functools.reduce()` to be passed as a keyword argument. Patch by Sayandip Dutta.

- gh-124452 [https://github.com/python/cpython/issues/124452]: Fix an issue in `email.policy.EmailPolicy.header_source_parse()` and `email.policy.Compat32.header_source_parse()` that introduced spurious leading whitespaces into header values when the header includes a newline character after the header name delimiter (`:`) and before the value.

- gh-117941 [https://github.com/python/cpython/issues/117941]: **`argparse.BooleanOptionalAction`** now rejects option names starting with `--no-`.

- gh-125884 [https://github.com/python/cpython/issues/125884]: Fixed the bug for `pdb` where it can’t set breakpoints on functions with certain annotations.

- gh-125355 [https://github.com/python/cpython/issues/125355]: Fix several bugs in `argparse.ArgumentParser.parse_intermixed_args()`.

    - The parser no longer changes temporarily during parsing.
    - Default values are not processed twice.
    - Required mutually exclusive groups containing positional arguments are now supported.

  - The missing arguments report now includes the names of all required optional and positional arguments.
  - Unknown options can be intermixed with positional arguments in parse_known_intermixed_args().

- gh-125767 [https://github.com/python/cpython/issues/125767]: `super` objects are now `pickleable` and `copyable`.

- gh-124969 [https://github.com/python/cpython/issues/124969]: `locale.nl_langinfo(locale.ALT_DIGITS)` now returns a string again. The returned value consists of up to 100 semicolon-separated symbols.

- gh-84850 [https://github.com/python/cpython/issues/84850]: Remove `URLopener` and `FancyURLopener` classes from `urllib.request`. They had previously raised `DeprecationWarning` since Python 3.3.

- gh-125666 [https://github.com/python/cpython/issues/125666]: Avoid the exiting the interpreter if a null byte is given as input in the new REPL.

- gh-125710 [https://github.com/python/cpython/issues/125710]: [Enum] fix hashable<->nonhashable comparisons for member values

- gh-125631 [https://github.com/python/cpython/issues/125631]: Restore ability to set `persistent_id` and `persistent_load` attributes of instances of the `Pickler` and `Unpickler` classes in the `pickle` module.

- gh-125378 [https://github.com/python/cpython/issues/125378]: Fixed the bug in `pdb` where after a multi-line command, an empty line repeats the first line of the multi-line command, instead of the full command.

- gh-125682 [https://github.com/python/cpython/issues/125682]: Reject non-ASCII digits in the Python implementation of `json.loads()` conforming to the JSON specification.

- gh-125660 [https://github.com/python/cpython/issues/125660]: Reject invalid unicode escapes for Python implementation of `json.loads()`.

- gh-52551 [https://github.com/python/cpython/issues/52551]: Use **`wcsftime()`** to implement `time.strftime()` on Windows.
- gh-125259 [https://github.com/python/cpython/issues/125259]: Fix the notes removal logic for errors thrown in enum initialization.
- gh-125633 [https://github.com/python/cpython/issues/125633]: Add function `inspect.ispackage()` to determine whether an object is a package or not.
- gh-125614 [https://github.com/python/cpython/issues/125614]: In the `FORWARDREF` format of `annotationlib`, fix bug where nested expressions were not returned as `annotationlib.ForwardRef` format.
- gh-125590 [https://github.com/python/cpython/issues/125590]: Allow `FrameLocalsProxy` to delete and pop if the key is not a fast variable.
- gh-125600 [https://github.com/python/cpython/issues/125600]: Only show stale code warning in `pdb` when we display source code.
- gh-125542 [https://github.com/python/cpython/issues/125542]: Deprecate passing keyword-only *prefix_chars* argument to `argparse.ArgumentParser.add_argument_group()`.
- gh-125541 [https://github.com/python/cpython/issues/125541]: Pressing `Ctrl-C` while blocked in `threading.Lock.acquire()`, `threading.RLock.acquire()`, and `threading.Thread.join()` now interrupts the function call and raises a `KeyboardInterrupt` exception on Windows, similar to how those functions behave on macOS and Linux.

- gh-125519 [https://github.com/python/cpython/issues/125519]: Improve traceback if `importlib.reload()` is called with an object that is not a module. Patch by Alex Waygood.

- gh-125451 [https://github.com/python/cpython/issues/125451]: Fix deadlock when `concurrent.futures.ProcessPoolExecutor` shuts down concurrently with an error when feeding a job to a worker process.

- gh-125115 [https://github.com/python/cpython/issues/125115]: Fixed a bug in `pdb` where arguments starting with `-` can't be passed to the debugged script.

- gh-125398 [https://github.com/python/cpython/issues/125398]: Fix the conversion of the `VIRTUAL_ENV` path in the activate script in `venv` when running in Git Bash for Windows.

- gh-125245 [https://github.com/python/cpython/issues/125245]: Fix race condition when importing `collections.abc`, which could incorrectly return an empty module.

- gh-52551 [https://github.com/python/cpython/issues/52551]: Fix encoding issues in `time.strftime()`, the `strftime()` method of the `datetime` classes `datetime`, `date` and `time` and formatting of these classes. Characters not encodable in the current locale are now acceptable in the format string. Surrogate pairs and sequence of surrogatescape-encoded bytes are no longer recombinated. Embedded null character no longer terminates the format string.

- gh-124984 [https://github.com/python/cpython/issues/124984]: Fixed thread safety in `ssl` in the free-threaded build. OpenSSL operations are now protected by a per-object lock.

- gh-124651 [https://github.com/python/cpython/issues/124651]: Properly quote template strings in `venv` activation scripts.

- gh-124694 [https://github.com/python/cpython/issues/124694]: We've added `concurrent.futures.InterpreterPoolExecutor`, which allows you to run code in multiple isolated interpreters. This allows you to circumvent the limitations of CPU-bound threads (due to the GIL). Patch by Eric Snow.

This addition is unrelated to **PEP 734** [https://peps.python.org/pep-0734/].

- gh-58032 [https://github.com/python/cpython/issues/58032]: Deprecate the `argparse.FileType` type converter.
- gh-99749 [https://github.com/python/cpython/issues/99749]: Adds a feature to optionally enable suggestions for argument choices and subparser names if mistyped by the user.
- gh-58956 [https://github.com/python/cpython/issues/58956]: Fixed a bug in `pdb` where sometimes the breakpoint won't trigger if it was set on a function which is already in the call stack.
- gh-124111 [https://github.com/python/cpython/issues/124111]: The tkinter module can now be built to use either the new version 9.0.0 of Tcl/Tk or the latest release 8.6.15 of Tcl/Tk 8. Tcl/Tk 9 includes many improvements, both to the Tcl language and to the appearance and utility of the graphical user interface provided by Tk.
- gh-80958 [https://github.com/python/cpython/issues/80958]: unittest discovery supports PEP 420 namespace packages as start directory again.
- gh-123370 [https://github.com/python/cpython/issues/123370]: Fix the canvas not clearing after running turtledemo clock.
- gh-89083 [https://github.com/python/cpython/issues/89083]: Add `uuid.uuid8()` for generating UUIDv8 objects as specified in **RFC 9562** [https://datatracker.ietf.org/doc/html/rfc9562.html]. Patch by Bénédikt Tran
- gh-122549 [https://github.com/python/cpython/issues/122549]: Add `platform.invalidate_caches()` to invalidate cached results.

- gh-120754 [https://github.com/python/cpython/issues/120754]: Update unbounded `read` calls in **`zipfile`** to specify an explicit `size` putting a limit on how much data they may read. This also updates handling around ZIP max comment size to match the standard instead of reading comments that are one byte too long.

- gh-121267 [https://github.com/python/cpython/issues/121267]: Improve the performance of **`tarfile`** when writing files, by caching user names and group names.

- gh-70764 [https://github.com/python/cpython/issues/70764]: Fixed an issue where **`inspect.getclosurevars()`** would incorrectly classify an attribute name as a global variable when the name exists both as an attribute name and a global variable.

- gh-118289 [https://github.com/python/cpython/issues/118289]: **`posixpath.realpath()`** now raises **`NotADirectoryError`** when *strict* mode is enabled and a non-directory path with a trailing slash is supplied.

- gh-119826 [https://github.com/python/cpython/issues/119826]: Always return an absolute path for **`os.path.abspath()`** on Windows.

- gh-97850 [https://github.com/python/cpython/issues/97850]: Remove deprecated **`pkgutil.get_loader()`** and **`pkgutil.find_loader()`**.

- gh-118986 [https://github.com/python/cpython/issues/118986]: Add **`socket.IPV6_RECVERR`** constant (available since Linux 2.2).

- gh-116897 [https://github.com/python/cpython/issues/116897]: Accepting objects with false values (like `0` and `[]`) except empty strings, byte-like objects and `None` in **`urllib.parse`** functions **`parse_qsl()`** and **`parse_qs()`** is now deprecated.

- gh-101955 [https://github.com/python/cpython/issues/101955]: Fix

SystemError when match regular expression pattern containing some combination of possessive quantifier, alternative and capture group.

- gh-71936 [https://github.com/python/cpython/issues/71936]: Fix a race condition in `multiprocessing.pool.Pool`.

- bpo-46128 [https://bugs.python.org/issue?@action=redirect&bpo=46128]: Strip `unittest.IsolatedAsyncioTestCase` stack frames from reported stacktraces.

- gh-84852 [https://github.com/python/cpython/issues/84852]: Add MIME types for MS Embedded OpenType, OpenType Layout, TrueType, WOFF 1.0 and 2.0 fonts. Patch by Sahil Prajapati and Hugo van Kemenade.

Documentation

- gh-126622 [https://github.com/python/cpython/issues/126622]: Added stub pages for removed modules explaining their removal, where to find replacements, and linking to the last Python version that supported them. Contributed by Ned Batchelder.
- gh-125277 [https://github.com/python/cpython/issues/125277]: Require Sphinx 7.2.6 or later to build the Python documentation. Patch by Adam Turner.
- gh-60712 [https://github.com/python/cpython/issues/60712]: Include the `object` type in the lists of documented types. Change by Furkan Onder and Martin Panter.

Core and Builtins

- gh-126795 [https://github.com/python/cpython/issues/126795]: Increase the threshold for JIT code warmup. Depending on platform and workload, this can result in performance gains of 1-9% and memory savings of 3-5%.
- gh-126341 [https://github.com/python/cpython/issues/126341]: Now `ValueError` is raised instead of `SystemError` when trying to iterate over a released `memoryview` object.
- gh-126688 [https://github.com/python/cpython/issues/126688]: Fix a

crash when calling `os.fork()` on some operating systems, including SerenityOS.

Library

- gh-126066 [https://github.com/python/cpython/issues/126066]: Fix `importlib` to not write an incomplete .pyc files when a ulimit or some other operating system mechanism is preventing the write to go through fully.

Core and Builtins

- gh-126222 [https://github.com/python/cpython/issues/126222]: Do not include count of «peek» items in `_PyUop_num_popped`. This ensures that the correct number of items are popped from the stack when a micro-op exits with an error.
- gh-126366 [https://github.com/python/cpython/issues/126366]: Fix crash when using `yield from` on an object that raises an exception in its `__iter__`.

Library

- gh-126209 [https://github.com/python/cpython/issues/126209]: Fix an issue with `skip_file_prefixes` parameter which resulted in an inconsistent behaviour between the C and Python implementations of `warnings.warn()`. Patch by Daehee Kim.

Core and Builtins

- gh-126312 [https://github.com/python/cpython/issues/126312]: Fix crash during garbage collection on an object frozen by `gc.freeze()` on the free-threaded build.
- gh-103951 [https://github.com/python/cpython/issues/103951]: Relax optimization requirements to allow fast attribute access to module subclasses.
- gh-126072 [https://github.com/python/cpython/issues/126072]: Following gh-126101 [https://github.com/python/cpython/issues/126101], for Objetos código like lambda, annotation and

type alias, we no longer add `None` to its **`co_consts`**.

- gh-126195 [https://github.com/python/cpython/issues/126195]: Improve JIT performance by 1.4% on macOS Apple Silicon by using platform-specific memory protection APIs. Patch by Diego Russo.
- gh-126139 [https://github.com/python/cpython/issues/126139]: Provide better error location when attempting to use a future statement with an unknown future feature.
- gh-126072 [https://github.com/python/cpython/issues/126072]: Add a new attribute in **`co_flags`** to indicate whether the first item in **`co_consts`** is the docstring. If a code object has no docstring, `None` will **NOT** be inserted.
- gh-126076 [https://github.com/python/cpython/issues/126076]: Relocated objects such as `tuple`, `bytes` and `str` objects are properly tracked by **`tracemalloc`** and its associated hooks. Patch by Pablo Galindo.
- gh-90370 [https://github.com/python/cpython/issues/90370]: Avoid temporary tuple creation for vararg in argument passing with Argument Clinic generated code (if arguments either vararg or positional-only).
- gh-126018 [https://github.com/python/cpython/issues/126018]: Fix a crash in **`sys.audit()`** when passing a non-string as first argument and Python was compiled in debug mode.
- gh-126012 [https://github.com/python/cpython/issues/126012]: The **`memoryview`** type now supports subscription, making it a generic type.
- gh-125837 [https://github.com/python/cpython/issues/125837]: Adds **`LOAD_SMALL_INT`** and **`LOAD_CONST_IMMORTAL`** instructions. `LOAD_SMALL_INT` pushes a small integer equal to the `oparg` to the stack. `LOAD_CONST_IMMORTAL` does the same as `LOAD_CONST` but is more efficient for immortal objects. Removes `RETURN_CONST` instruction.
- gh-125942 [https://github.com/python/cpython/issues/125942]: On Android, the `errors` setting of **`sys.stdout`** was changed from `surrogateescape` to `backslashreplace`.
- gh-125859 [https://github.com/python/cpython/issues/125859]: Fix a crash in the free threading build when **`gc.get_objects()`** or **`gc.get_referrers()`** is called during an in-progress garbage collection.
- gh-125868 [https://github.com/python/cpython/issues/125868]: It was

possible in 3.14.0a1 only for attribute lookup to give the wrong value. This was due to an incorrect specialization in very specific circumstances. This is fixed in 3.14.0a2.

- gh-125498 [https://github.com/python/cpython/issues/125498]: The JIT has been updated to leverage Clang 19's new `preserve_none` attribute, which supports more platforms and is more useful than LLVM's existing `ghccc` calling convention. This also removes the need to manually patch the calling convention in LLVM IR, simplifying the JIT compilation process.
- gh-125703 [https://github.com/python/cpython/issues/125703]: Correctly honour **`tracemalloc`** hooks in specialized `Py_DECREF` paths. Patch by Pablo Galindo
- gh-125593 [https://github.com/python/cpython/issues/125593]: Use color to highlight error locations in traceback from exception group
- gh-125017 [https://github.com/python/cpython/issues/125017]: Fix crash on certain accesses to the `__annotations__` of **`staticmethod`** and **`classmethod`** objects.
- gh-125588 [https://github.com/python/cpython/issues/125588]: The Python PEG generator can now use f-strings in the grammar actions. Patch by Pablo Galindo
- gh-125444 [https://github.com/python/cpython/issues/125444]: Fix illegal instruction for older Arm architectures. Patch by Diego Russo, testing by Ross Burton.
- gh-118423 [https://github.com/python/cpython/issues/118423]: Add a new `INSTRUCTION_SIZE` macro to the cases generator which returns the current instruction size.
- gh-125038 [https://github.com/python/cpython/issues/125038]: Fix crash when iterating over a generator expression after direct changes on `gi_frame.f_locals`. Patch by Mikhail Efimov.
- gh-124855 [https://github.com/python/cpython/issues/124855]: Don't allow the JIT and perf support to be active at the same time. Patch by Pablo Galindo
- gh-123714 [https://github.com/python/cpython/issues/123714]: Update JIT compilation to use LLVM 19
- gh-123930 [https://github.com/python/cpython/issues/123930]: Improve the error message when a script shadowing a module from the standard library causes **`ImportError`** to be raised during a «from» import. Similarly, improve the error message

when a script shadowing a third party module attempts to «from» import an attribute from that third party module while still initialising.
- gh-119793 [https://github.com/python/cpython/issues/119793]: The `map()` built-in now has an optional keyword-only *strict* flag like `zip()` to check that all the iterables are of equal length. Patch by Wannes Boeykens.

Library

- gh-118950 [https://github.com/python/cpython/issues/118950]: Fix bug where SSLProtocol.connection_lost wasn't getting called when OSError was thrown on writing to socket.
- gh-113570 [https://github.com/python/cpython/issues/113570]: Fixed a bug in `reprlib.repr` where it incorrectly called the repr method on shadowed Python built-in types.

C API

- gh-126554 [https://github.com/python/cpython/issues/126554]: Fix error handling in `ctypes.CDLL` objects which could result in a crash in rare situations.
- gh-126061 [https://github.com/python/cpython/issues/126061]: Add `PyLong_IsPositive()`, `PyLong_IsNegative()` and `PyLong_IsZero()` for checking if a `PyLongObject` is positive, negative, or zero, respectively.
- gh-125608 [https://github.com/python/cpython/issues/125608]: Fix a bug where dictionary watchers (e.g., `PyDict_Watch()`) on an object's attribute dictionary (`__dict__`) were not triggered when the object's attributes were modified.
- gh-123619 [https://github.com/python/cpython/issues/123619]: Added the `PyUnstable_Object_EnableDeferredRefcount()` function for enabling **PEP 703** [https://peps.python.org/pep-0703/] deferred reference counting.
- gh-121654 [https://github.com/python/cpython/issues/121654]: Add `PyType_Freeze()` function to make a type immutable. Patch by Victor Stinner.
- gh-120026 [https://github.com/python/cpython/issues/120026]: The `Py_HUGE_VAL` macro is soft deprecated.

Build

- gh-126691 [https://github.com/python/cpython/issues/126691]: Removed the `--with-emscripten-target` configure flag. We unified the `node` and `browser` options and the same build can now be used, independent of target runtime.
- gh-123877 [https://github.com/python/cpython/issues/123877]: Use `wasm32-wasip1` as the target triple for WASI instead of `wasm32-wasi`. The latter will eventually be reclaimed for WASI 1.0 while CPython currently only supports WASI preview1.
- gh-126458 [https://github.com/python/cpython/issues/126458]: Disable SIMD support for HACL under WASI.
- gh-89640 [https://github.com/python/cpython/issues/89640]: Hard-code float word ordering as little endian on WASM.
- gh-126206 [https://github.com/python/cpython/issues/126206]: `make clinic` now runs Argument Clinic using the `--force` option, thus forcefully regenerating generated code.
- gh-126187 [https://github.com/python/cpython/issues/126187]: Introduced `Tools/wasm/emscripten.py` to simplify doing Emscripten builds.
- gh-124932 [https://github.com/python/cpython/issues/124932]: For cross builds, there is now support for having a different install `prefix` than the `host_prefix` used by `getpath.py`. This is set to `/` by default for Emscripten, on other platforms the default behavior is the same as before.
- gh-125946 [https://github.com/python/cpython/issues/125946]: The minimum supported Android version is now 7.0 (API level 24).
- gh-125940 [https://github.com/python/cpython/issues/125940]: The Android build now supports 16 KB page sizes [https://developer.android.com/guide/practices/page-sizes].
- gh-89640 [https://github.com/python/cpython/issues/89640]: Improve detection of float word ordering on Linux when link-time optimizations are enabled.
- gh-124928 [https://github.com/python/cpython/issues/124928]: Emscripten builds now require node $>=$ 18.
- gh-115382 [https://github.com/python/cpython/issues/115382]: Fix cross compile failures when the host and target SOABIs match.

Python 3.14.0 alpha 1

Release date: 2024-10-15

macOS

- gh-124448 [https://github.com/python/cpython/issues/124448]: Update bundled Tcl/Tk in macOS installer to 8.6.15.
- gh-123797 [https://github.com/python/cpython/issues/123797]: Check for runtime availability of `ptsname_r` function on macos.
- gh-123418 [https://github.com/python/cpython/issues/123418]: Updated macOS installer build to use OpenSSL 3.0.15.

Windows

- gh-124487 [https://github.com/python/cpython/issues/124487]: Increases Windows required OS and API level to Windows 10.
- gh-124609 [https://github.com/python/cpython/issues/124609]: Fix `_Py_ThreadId` for Windows builds using MinGW. Patch by Tony Roberts.
- gh-124448 [https://github.com/python/cpython/issues/124448]: Updated bundled Tcl/Tk to 8.6.15.
- gh-124254 [https://github.com/python/cpython/issues/124254]: Ensures experimental free-threaded binaries remain installed when updating.
- gh-123915 [https://github.com/python/cpython/issues/123915]: Ensure that `Tools\msi\buildrelease.bat` uses different directories for AMD64 and ARM64 builds.
- gh-123418 [https://github.com/python/cpython/issues/123418]: Updated Windows build to use OpenSSL 3.0.15.
- gh-123476 [https://github.com/python/cpython/issues/123476]: Add support for `socket.TCP_QUICKACK` on Windows platforms.
- gh-122573 [https://github.com/python/cpython/issues/122573]: The Windows build of CPython now requires 3.10 or newer.
- gh-100256 [https://github.com/python/cpython/issues/100256]: **`mimetypes`** no longer fails when it encounters an inaccessible registry key.
- gh-119679 [https://github.com/python/cpython/issues/119679]: Ensures correct import libraries are included in Windows

installs.

- gh-119690 [https://github.com/python/cpython/issues/119690]: Adds Unicode support and fixes audit events for `_winapi.CreateNamedPipe`.
- gh-111201 [https://github.com/python/cpython/issues/111201]: Add support for new pyrepl on Windows
- gh-119070 [https://github.com/python/cpython/issues/119070]: Fixes `py.exe` handling of shebangs like `/usr/bin/env python3.12`, which were previously interpreted as `python3.exe` instead of `python3.12.exe`.
- gh-117505 [https://github.com/python/cpython/issues/117505]: Fixes an issue with the Windows installer not running ensurepip in a fully isolated environment. This could cause unexpected interactions with the user site-packages.
- gh-118209 [https://github.com/python/cpython/issues/118209]: Avoid crashing in **`mmap`** on Windows when the mapped memory is inaccessible due to file system errors or access violations.
- gh-79846 [https://github.com/python/cpython/issues/79846]: Makes `ssl.create_default_context()` ignore invalid certificates in the Windows certificate store

Tools/Demos

- gh-123418 [https://github.com/python/cpython/issues/123418]: Update GitHub CI workflows to use OpenSSL 3.0.15 and multissltests to use 3.0.15, 3.1.7, and 3.2.3.

Tests

- gh-125041 [https://github.com/python/cpython/issues/125041]: Re-enable skipped tests for **`zlib`** on the s390x architecture: only skip checks of the compressed bytes, which can be different between zlib’s software implementation and the hardware-accelerated implementation.
- gh-124378 [https://github.com/python/cpython/issues/124378]: Updated `test_ttk` to pass with Tcl/Tk 8.6.15.
- gh-124213 [https://github.com/python/cpython/issues/124213]: Detect whether the test suite is running inside a systemd-nspawn container with `--suppress-sync=true` option, and skip the `test_os` and `test_mmap` tests that are failing in this

scenario.

- gh-124190 [https://github.com/python/cpython/issues/124190]: Add capability to ignore entire files or directories in check warning CI tool
- gh-121921 [https://github.com/python/cpython/issues/121921]: Update `Lib/test/crashers/bogus_code_obj.py` so that it crashes properly again.
- gh-112301 [https://github.com/python/cpython/issues/112301]: Add tooling to check for changes in compiler warnings. Patch by Nate Ohlson.
- gh-59022 [https://github.com/python/cpython/issues/59022]: Add tests for **`pkgutil.extend_path()`**. Patch by Andreas Stocker.
- gh-99242 [https://github.com/python/cpython/issues/99242]: **`os.getloadavg()`** may throw **`OSError`** when running regression tests under certain conditions (e.g. chroot). This error is now caught and ignored, since reporting load average is optional.
- gh-121084 [https://github.com/python/cpython/issues/121084]: Fix test_typing random leaks. Clear typing ABC caches when running tests for refleaks (`-R` option): call `_abc_caches_clear()` on typing abstract classes and their subclasses. Patch by Victor Stinner.
- gh-121160 [https://github.com/python/cpython/issues/121160]: Add a test for **`readline.set_history_length()`**. Note that this test may fail on readline libraries.
- gh-121200 [https://github.com/python/cpython/issues/121200]: Fix `test_expanduser_pwd2()` of `test_posixpath`. Call `getpwnam()` to get `pw_dir`, since it can be different than `getpwall()` `pw_dir`. Patch by Victor Stinner.
- gh-121188 [https://github.com/python/cpython/issues/121188]: When creating the JUnit XML file, regrtest now escapes characters which are invalid in XML, such as the chr(27) control character used in ANSI escape sequences. Patch by Victor Stinner.
- gh-120801 [https://github.com/python/cpython/issues/120801]: Cleaned up fixtures for importlib.metadata tests and consolidated behavior with “test.support.os_helper”.
- gh-119727 [https://github.com/python/cpython/issues/119727]: Add `--single-process` command line option to Python test

runner (regrtest). Patch by Victor Stinner.

- gh-119273 [https://github.com/python/cpython/issues/119273]: Python test runner no longer runs tests using TTY (ex: test_ioctl) in a process group (using `setsid()`). Previously, tests using TTY were skipped. Patch by Victor Stinner.
- gh-119050 [https://github.com/python/cpython/issues/119050]: regrtest test runner: Add XML support to the refleak checker (-R option). Patch by Victor Stinner.
- gh-101525 [https://github.com/python/cpython/issues/101525]: Skip `test_gdb` if the binary is relocated by BOLT. Patch by Donghee Na.
- gh-107562 [https://github.com/python/cpython/issues/107562]: Test certificates have been updated to expire far in the future. This allows testing Y2038 with system time set to after that, so that actual Y2038 issues can be exposed, and not masked by expired certificate errors.

Security

- gh-125140 [https://github.com/python/cpython/issues/125140]: Remove the current directory from `sys.path` when using PyREPL.

- gh-123678 [https://github.com/python/cpython/issues/123678]: Upgrade libexpat to 2.6.3

- gh-112301 [https://github.com/python/cpython/issues/112301]: Enable compiler options that warn of potential security vulnerabilities.

- gh-122792 [https://github.com/python/cpython/issues/122792]: Changed IPv4-mapped `ipaddress.IPv6Address` to consistently use the mapped IPv4 address value for deciding properties. Properties which have their behavior fixed are `is_multicast`, `is_reserved`, `is_link_local`, `is_global`, and `is_unspecified`.

- gh-112301 [https://github.com/python/cpython/issues/112301]: Add ability to ignore warnings per file with warning count in warning checking tooling. Patch by Nate Ohlson.

- gh-112301 [https://github.com/python/cpython/issues/112301]: Add macOS warning tracking to warning check tooling. Patch by Nate Ohlson.

- gh-122133 [https://github.com/python/cpython/issues/122133]: Authenticate the socket connection for the `socket.socketpair()` fallback on platforms where `AF_UNIX` is not available like Windows.

  Patch by Gregory P. Smith <greg@krypto.org> and Seth Larson <seth@python.org>. Reported by Ellie <el@horse64.org>

- gh-121957 [https://github.com/python/cpython/issues/121957]: Fixed missing audit events around interactive use of Python, now also properly firing for `python -i`, as well as for `python -m asyncio`. The events in question are `cpython.run_stdin` and `cpython.run_startup`.

- gh-112301 [https://github.com/python/cpython/issues/112301]: Enable runtime protections for glibc to abort execution when unsafe behavior is encountered, for all platforms except Windows.

- gh-121285 [https://github.com/python/cpython/issues/121285]: Remove backtracking from tarfile header parsing for `hdrcharset`, PAX, and GNU sparse headers.

- gh-112301 [https://github.com/python/cpython/issues/112301]: Add default compiler options to improve security. Enable -Wimplicit-fallthrough, -fstack-protector-strong, -Wtrampolines.

- gh-118773 [https://github.com/python/cpython/issues/118773]: Fixes creation of ACLs in **`os.mkdir()`** on Windows to work correctly on non-English machines.

- gh-118486 [https://github.com/python/cpython/issues/118486]: **`os.mkdir()`** on Windows now accepts *mode* of `0o700` to restrict the new directory to the current user. This fixes **CVE 2024-4030** [https://www.cve.org/CVERecord?id=CVE-2024-4030] affecting `tempfile.mkdtemp()` in scenarios where the base temporary directory is more permissive than the default.

Library

- gh-125422 [https://github.com/python/cpython/issues/125422]: Fixed the bug where `pdb` and `bdb` can step into the bottom caller frame.

- gh-100141 [https://github.com/python/cpython/issues/100141]: Fixed the bug where `pdb` will be stuck in an infinite loop when debugging an empty file.

- gh-53203 [https://github.com/python/cpython/issues/53203]: Fix `time.strptime()` for `%c`, `%x` and `%X` formats in many locales that use non-ASCII digits, like Persian, Burmese, Odia and Shan.

- gh-125206 [https://github.com/python/cpython/issues/125206]: Workaround for old libffi versions is added. Module ctypes supports double complex only with libffi >= 3.3.0. Patch by Mikhail Efimov.

- gh-89967 [https://github.com/python/cpython/issues/89967]: Make `WeakKeyDictionary` and `WeakValueDictionary` safe against concurrent mutations from other threads. Patch by Kumar Aditya.

- gh-125260 [https://github.com/python/cpython/issues/125260]: The `gzip.compress()` *mtime* parameter now defaults to 0 for reproducible output. Patch by Bernhard M. Wiedemann and Adam Turner.

- gh-125243 [https://github.com/python/cpython/issues/125243]: Fix data race when creating `zoneinfo.ZoneInfo` objects in the free threading build.

- gh-125254 [https://github.com/python/cpython/issues/125254]: Fix a bug where ArgumentError includes the incorrect ambiguous option in `argparse`.

- gh-125235 [https://github.com/python/cpython/issues/125235]: Keep `tkinter` TCL paths in venv pointing to base installation on Windows.

- gh-61011 [https://github.com/python/cpython/issues/61011]: Fix inheritance of nested mutually exclusive groups from parent parser in `argparse.ArgumentParser`. Previously, all nested mutually exclusive groups lost their connection to the group containing them and were displayed as belonging directly to the parser.

- gh-125118 [https://github.com/python/cpython/issues/125118]: Don't copy arbitrary values to _Bool in the `struct` module.

- gh-125069 [https://github.com/python/cpython/issues/125069]: Fix an issue where providing a `pathlib.PurePath` object as an initializer argument to a second `PurePath` object with a different `parser` resulted in arguments to the former object's initializer being joined by the latter object's parser.

- gh-125096 [https://github.com/python/cpython/issues/125096]: If the `PYTHON_BASIC_REPL` environment variable is set, the `site` module no longer imports the `_pyrepl` module. Moreover, the `site` module now respects `-E` and `-I` command line options: ignore `PYTHON_BASIC_REPL` in this case. Patch by Victor Stinner.

- gh-124969 [https://github.com/python/cpython/issues/124969]: Fix `locale.nl_langinfo(locale.ALT_DIGITS)`. Now it returns a tuple of up to 100 strings (an empty tuple on most locales). Previously it returned the first item of that tuple or an empty string.

- gh-124960 [https://github.com/python/cpython/issues/124960]: Fix support for the `barry_as_FLUFL` future flag in the new REPL.

- gh-69998 [https://github.com/python/cpython/issues/69998]: Fix `locale.nl_langinfo()` in case when different categories have different locales. The function now sets temporarily the `LC_CTYPE` locale in some cases. This temporary change affects other threads.

- gh-124958 [https://github.com/python/cpython/issues/124958]: Fix refcycles in exceptions raised from `asyncio.TaskGroup` and the python implementation of `asyncio.Future`
- gh-53203 [https://github.com/python/cpython/issues/53203]: Fix `time.strptime()` for `%c` and `%x` formats in many locales: Arabic, Bislama, Breton, Bodo, Kashubian, Chuvash, Estonian, French, Irish, Ge'ez, Gurajati, Manx Gaelic, Hebrew, Hindi, Chhattisgarhi, Haitian Kreyol, Japanese, Kannada, Korean, Marathi, Malay, Norwegian, Nynorsk, Punjabi, Rajasthani, Tok Pisin, Yoruba, Yue Chinese, Yau/Nungon and Chinese.
- gh-123961 [https://github.com/python/cpython/issues/123961]: Convert `curses` to multi-phase initialization (**PEP 489** [https://peps.python.org/pep-0489/]), thereby fixing reference leaks at interpreter shutdown. Patch by Bénédikt Tran.
- gh-124944 [https://github.com/python/cpython/issues/124944]: Add `SO_ORIGINAL_DST` to the `socket` module.
- gh-117151 [https://github.com/python/cpython/issues/117151]: The default buffer size used by `shutil.copyfileobj()` has been increased from 64k to 256k on non-Windows platforms. It was already larger on Windows.
- gh-90102 [https://github.com/python/cpython/issues/90102]: Skip the `isatty` system call during open() when the file is known to not be a character device. This provides a slight performance improvement when reading whole files.
- gh-124917 [https://github.com/python/cpython/issues/124917]: Allow calling `os.path.exists()` and `os.path.lexists()` with keyword arguments on Windows. Fixes a regression in 3.13.0.
- gh-65865 [https://github.com/python/cpython/issues/65865]: `argparse` now raises early error for invalid `help` arguments to `add_argument()`, `add_subparsers()` and **`add_parser()`**.

- gh-124653 [https://github.com/python/cpython/issues/124653]: Fix detection of the minimal Queue API needed by the `logging` module. Patch by Bénédikt Tran.

- gh-91818 [https://github.com/python/cpython/issues/91818]: The CLI of many modules (`ast`, `ensurepip`, `json`, `pdb`, `sqlite3`, `tokenize`, `venv`) now uses the actual executable name instead of simply «python» to display in the usage message.

- gh-124858 [https://github.com/python/cpython/issues/124858]: Fix reference cycles left in tracebacks in `asyncio.open_connection()` when used with `happy_eyeballs_delay`

- gh-124390 [https://github.com/python/cpython/issues/124390]: Fixed `AssertionError` when using **`asyncio.staggered.staggered_race()`** with `asyncio.eager_task_factory`.

- gh-85935 [https://github.com/python/cpython/issues/85935]: `argparse.ArgumentParser.add_argument()` now raises an exception if an action that does not consume arguments (like “store_const” or “store_true”) or explicit `nargs=0` are specified for positional arguments.

- gh-124835 [https://github.com/python/cpython/issues/124835]: Make `tomllib.loads()` raise `TypeError` not `AttributeError` on bad input types that do not have the `replace` attribute. Improve error message when `bytes` is received.

- gh-124693 [https://github.com/python/cpython/issues/124693]: Fix a bug where `argparse` doesn’t recognize negative complex numbers or negative numbers using scientific notation.

- gh-124787 [https://github.com/python/cpython/issues/124787]: Fix `typing.TypeAliasType` with incorrect `type_params` argument. Now it raises a `TypeError` when a type parameter without a default follows one with a default, and when an entry in the `type_params` tuple is not a type parameter object.

- gh-66436 [https://github.com/python/cpython/issues/66436]: Improved prog default value for `argparse.ArgumentParser`. It will now include the name of the Python executable along with the module or package name, or the path to a directory, ZIP file, or directory within a ZIP file if the code was run that way.

- gh-116850 [https://github.com/python/cpython/issues/116850]: Fix `argparse` for namespaces with not directly writable dict (e.g. classes).

- gh-101552 [https://github.com/python/cpython/issues/101552]: Add an *annoation_format* parameter to `inspect.signature()`. Add an *quote_annotation_strings* parameter to `inspect.Signature.format()`. Use the new functionality to improve the display of annotations in signatures in `pydoc`. Patch by Jelle Zijlstra.

- gh-58573 [https://github.com/python/cpython/issues/58573]: Fix conflicts between abbreviated long options in the parent parser and subparsers in `argparse`.

- gh-124594 [https://github.com/python/cpython/issues/124594]: All `asyncio` REPL prompts run in the same `context`. Contributed by Bartosz Sławecki.

- gh-61181 [https://github.com/python/cpython/issues/61181]: Fix support of choices with string value in `argparse`. Substrings of the specified string no longer considered valid values.

- gh-116750 [https://github.com/python/cpython/issues/116750]: Provide `sys.monitoring.clear_tool_id()` to unregister all events and callbacks set by the tool.

- gh-124552 [https://github.com/python/cpython/issues/124552]: Improve the accuracy of `bdb`'s check for the possibility of breakpoint in a frame. This makes it possible to disable unnecessary events in functions.

- gh-124538 [https://github.com/python/cpython/issues/124538]: Fixed crash when using **`gc.get_referents()`** on a capsule object.

- gh-80259 [https://github.com/python/cpython/issues/80259]: Fix **`argparse`** support of positional arguments with `nargs='?'`, `default=argparse.SUPPRESS` and specified `type`.

- gh-120378 [https://github.com/python/cpython/issues/120378]: Fix a crash related to an integer overflow in **`curses.resizeterm()`** and **`curses.resize_term()`**.

- gh-124498 [https://github.com/python/cpython/issues/124498]: Fix **`typing.TypeAliasType`** not to be generic, when `type_params` is an empty tuple.

- gh-53834 [https://github.com/python/cpython/issues/53834]: Fix support of arguments with choices in **`argparse`**. Positional arguments with nargs equal to `'?'` or `'*'` no longer check default against `choices`. Optional arguments with `nargs` equal to `'?'` no longer check const against `choices`.

- gh-123884 [https://github.com/python/cpython/issues/123884]: Fixed bug in itertools.tee() handling of other tee inputs (a tee in a tee). The output now has the promised *n* independent new iterators. Formerly, the first iterator was identical (not independent) to the input iterator. This would sometimes give surprising results.

- gh-123017 [https://github.com/python/cpython/issues/123017]: Due to unreliable results on some devices, **`time.strftime()`** no longer accepts negative years on Android.

- gh-123014 [https://github.com/python/cpython/issues/123014]: **`os.pidfd_open()`** and **`signal.pidfd_send_signal()`** are now unavailable when building against Android API levels older than 31, since the underlying system calls may cause a crash.

- gh-124176 [https://github.com/python/cpython/issues/124176]: Add

support for `dataclasses.dataclass()` in `unittest.mock.create_autospec()`. Now `create_autospec` will check for potential dataclasses and use `dataclasses.fields()` function to retrieve the spec information.

- gh-124345 [https://github.com/python/cpython/issues/124345]: `argparse` supports abbreviated single-dash long options separated by `=` from its value.

- gh-124400 [https://github.com/python/cpython/issues/124400]: Fixed a `pdb` bug where `until` has no effect when it appears in a `commands` sequence. Also avoid printing the frame information at a breakpoint that has a command list containing a command that resumes execution.

- gh-90562 [https://github.com/python/cpython/issues/90562]: Modify dataclasses to support zero-argument super() when `slots=True` is specified. This works by modifying all references to `__class__` to point to the newly created class.

- gh-104860 [https://github.com/python/cpython/issues/104860]: Fix disallowing abbreviation of single-dash long options in `argparse` with `allow_abbrev=False`.

- gh-63143 [https://github.com/python/cpython/issues/63143]: Fix parsing mutually exclusive arguments in `argparse`. Arguments with the value identical to the default value (e.g. booleans, small integers, empty or 1-character strings) are no longer considered «not present».

- gh-72795 [https://github.com/python/cpython/issues/72795]: Positional arguments with nargs equal to `'*'` or **`argparse.REMAINDER`** are no longer required. This allows to use positional argument with `nargs='*'` and without `default` in mutually exclusive group and improves error message about required arguments.

- gh-59317 [https://github.com/python/cpython/issues/59317]: Fix parsing positional argument with nargs equal to `'?'` or `'*'` if it is preceded by an option and another positional argument.

- gh-100980 [https://github.com/python/cpython/issues/100980]: The `_fields_` attribute of `ctypes.Structure` and `Union` is no longer set if the setattr operation raises an error.

- gh-53780 [https://github.com/python/cpython/issues/53780]: `argparse` now ignores the first `"--"` (double dash) between an option and command.

- gh-124217 [https://github.com/python/cpython/issues/124217]: Add RFC 9637 reserved IPv6 block `3fff::/20` in `ipaddress` module.

- gh-111513 [https://github.com/python/cpython/issues/111513]: Improve the error message that may be raised by `datetime.date.fromtimestamp()`.

- gh-124248 [https://github.com/python/cpython/issues/124248]: Fixed potential crash when using `struct` to process zero-width "Pascal string" fields (`0p`).

- gh-81691 [https://github.com/python/cpython/issues/81691]: Fix handling of multiple `"--"` (double dashes) in `argparse`. Only the first one has now been removed, all subsequent ones are now taken literally.

- gh-87041 [https://github.com/python/cpython/issues/87041]: Fix a bug in `argparse` where lengthy subparser argument help is incorrectly indented.

- gh-84559 [https://github.com/python/cpython/issues/84559]: The default `multiprocessing` start method on Linux and other POSIX systems has been changed away from often unsafe `"fork"` to `"forkserver"` (when the platform supports sending file handles over pipes as most do) or `"spawn"`. Mac and Windows are unchanged as they already default to `"spawn"`.

- gh-124212 [https://github.com/python/cpython/issues/124212]: Fix invalid variable in `venv` handling of failed symlink on

Windows

- gh-124171 [https://github.com/python/cpython/issues/124171]: Add workaround for broken **`fmod()`** implementations on Windows, that loose zero sign (e.g. `fmod(-10, 1)` returns `0.0`). Patch by Sergey B Kirpichev.

- gh-123978 [https://github.com/python/cpython/issues/123978]: Remove broken **`time.thread_time()`** and **`time.thread_time_ns()`** on NetBSD.

- gh-123934 [https://github.com/python/cpython/issues/123934]: Fix **`unittest.mock.MagicMock`** resetting magic methods return values after `.reset_mock(return_value=True)` was called.

- gh-124016 [https://github.com/python/cpython/issues/124016]: Update **`unicodedata`** database to Unicode 16.0.0.

- gh-123968 [https://github.com/python/cpython/issues/123968]: Fix the command-line interface for the **`random`** module to select floats between 0 and N, not 1 and N.

- gh-123945 [https://github.com/python/cpython/issues/123945]: Fix a bug where **`argparse`** doesn't recognize negative numbers with underscores

- gh-123935 [https://github.com/python/cpython/issues/123935]: Fix parent slots detection for dataclasses that inherit from classes with `__dictoffset__`.

- gh-123892 [https://github.com/python/cpython/issues/123892]: Add `"_wmi"` to **`sys.stdlib_module_names`**. Patch by Victor Stinner.

- gh-84808 [https://github.com/python/cpython/issues/84808]: Fix error handling in **`socket`** method **`connect_ex()`** on platforms where **`errno`** can be negative.

- gh-123756 [https://github.com/python/cpython/issues/123756]: Added a new argument `mode` to **`pdb.Pdb`**. Only allow **`pdb`** from command line to use `restart` command.

- gh-122765 [https://github.com/python/cpython/issues/122765]: Fix unbalanced quote errors occurring when activate.csh in **`venv`** was sourced with a custom prompt containing unpaired quotes or newlines.

- gh-123657 [https://github.com/python/cpython/issues/123657]: Fix crash and memory leak in **`decimal.getcontext()`**. It crashed when using a thread-local context by `--with-decimal-contextvar=no`.

- gh-123339 [https://github.com/python/cpython/issues/123339]: Fix **`inspect.getsource()`** for classes in **`collections.abc`** and **`decimal`** (for pure Python implementation) modules. **`inspect.getcomments()`** now raises OSError instead of IndexError if the `__firstlineno__` value for a class is out of bound.

- gh-123374 [https://github.com/python/cpython/issues/123374]: Remove check for redefined memo entry in **`pickletools.dis()`**.

- gh-123504 [https://github.com/python/cpython/issues/123504]: Fixed reference leak in the finalization of **`tkinter`**.

- gh-123430 [https://github.com/python/cpython/issues/123430]: Pages generated by the **`http.server`** module allow the browser to apply its default dark mode.

- gh-123446 [https://github.com/python/cpython/issues/123446]: Fix empty function name in **`TypeError`** when **`csv.reader()`**, **`csv.writer()`**, or **`csv.register_dialect()`** are used without the required args.

- gh-123448 [https://github.com/python/cpython/issues/123448]: Fixed memory leak of **`typing.NoDefault`** by moving it to the static types array.

- gh-123409 [https://github.com/python/cpython/issues/123409]: Fix **`ipaddress.IPv6Address.reverse_pointer`** output according to RFC 3596, §2.5 [https://datatracker.ietf.org/doc/html/rfc3596.html#section-2.5]. Patch by Bénédikt Tran.

- gh-123089 [https://github.com/python/cpython/issues/123089]: Make `weakref.WeakSet` safe against concurrent mutations while it is being iterated. Patch by Kumar Aditya.
- gh-123363 [https://github.com/python/cpython/issues/123363]: Show string value of `CONTAINS_OP` oparg in `dis` output. Patch by Alexandr153.
- gh-123341 [https://github.com/python/cpython/issues/123341]: Add `__class_getitem__()` to `tkinter.Event` for type subscript support at runtime. Patch by Adonis Rakateli.
- gh-123340 [https://github.com/python/cpython/issues/123340]: Show string value of `IS_OP` oparg in `dis` output.
- gh-119518 [https://github.com/python/cpython/issues/119518]: Speed up normalization of `pathlib.PurePath` and `Path` objects by not interning string parts.
- gh-123270 [https://github.com/python/cpython/issues/123270]: Applied a more surgical fix for malformed payloads in `zipfile.Path` causing infinite loops (gh-122905 [https://github.com/python/cpython/issues/122905]) without breaking contents using legitimate characters.
- gh-73991 [https://github.com/python/cpython/issues/73991]: Add `pathlib.Path.copy_into()` and `move_into()`, which copy and move files and directories into *existing* directories.
- gh-123228 [https://github.com/python/cpython/issues/123228]: Fix return type for `_pyrepl.readline._ReadlineWrapper.get_line_buffer()` to be `str()`. Patch by Sergey B Kirpichev.
- gh-123240 [https://github.com/python/cpython/issues/123240]: Raise audit events for the `input()` in the new REPL.
- gh-76960 [https://github.com/python/cpython/issues/76960]: Fix `urllib.parse.urljoin()` and `urllib.parse.urldefrag()` for URIs containing empty components. For example, `urljoin()` with relative

reference «?» now sets empty query and removes fragment. Preserve empty components (authority, params, query, fragment) in `urljoin()`. Preserve empty components (authority, params, query) in `urldefrag()`.

- gh-116810 [https://github.com/python/cpython/issues/116810]: Resolve a memory leak introduced in CPython 3.10's `ssl` when the `ssl.SSLSocket.session` property was accessed. Speeds up read and write access to said property by no longer unnecessarily cloning session objects via serialization.

- gh-123243 [https://github.com/python/cpython/issues/123243]: Fix memory leak in `_decimal`.

- gh-122546 [https://github.com/python/cpython/issues/122546]: Consistently use same file name for different exceptions in the new repl. Patch by Sergey B Kirpichev.

- gh-123213 [https://github.com/python/cpython/issues/123213]: `xml.etree.ElementTree.Element.extend()` and `Element` assignment no longer hide the internal exception if an erroneous generator is passed. Patch by Bar Harel.

- gh-85110 [https://github.com/python/cpython/issues/85110]: Preserve relative path in URL without netloc in `urllib.parse.urlunsplit()` and `urllib.parse.urlunparse()`.

- gh-123165 [https://github.com/python/cpython/issues/123165]: Add support for rendering `Positions` in `dis`.

- gh-122909 [https://github.com/python/cpython/issues/122909]: In urllib.request when URLError is raised opening an ftp URL, the exception argument is now consistently a string. Earlier versions passed either a string or an ftplib exception instance as the argument to URLError.

- gh-123084 [https://github.com/python/cpython/issues/123084]: Deprecate `shutil.ExecError`, which hasn't been raised by any `shutil` function since Python 3.4. It's now an alias for

`RuntimeError`.

- gh-123085 [https://github.com/python/cpython/issues/123085]: In a bare call to **`importlib.resources.files()`**, ensure the caller's frame is properly detected when `importlib.resources` is itself available as a compiled module only (no source).
- gh-123067 [https://github.com/python/cpython/issues/123067]: Fix quadratic complexity in parsing `"`-quoted cookie values with backslashes by **`http.cookies`**.
- gh-123049 [https://github.com/python/cpython/issues/123049]: Add support for **`UNNAMED_SECTION`** in **`configparser.ConfigParser.add_section()`**.
- gh-121735 [https://github.com/python/cpython/issues/121735]: When working with zip archives, importlib.resources now properly honors module-adjacent references (e.g. `files(pkg.mod)` and not just `files(pkg)`).
- gh-122981 [https://github.com/python/cpython/issues/122981]: Fix **`inspect.getsource()`** for generated classes with Python base classes (e.g. enums).
- gh-122903 [https://github.com/python/cpython/issues/122903]: `zipfile.Path.glob` now correctly matches directories instead of silently omitting them.
- gh-122905 [https://github.com/python/cpython/issues/122905]: **`zipfile.Path`** objects now sanitize names from the zipfile.
- gh-122873 [https://github.com/python/cpython/issues/122873]: Enable **`json`** module to work as a script using the **`-m`** switch: `python -m json`. See the JSON command-line interface documentation. Patch by Trey Hunner.
- gh-122858 [https://github.com/python/cpython/issues/122858]: Deprecate **`asyncio.iscoroutinefunction()`** in favor of **`inspect.iscoroutinefunction()`**.
- gh-116263 [https://github.com/python/cpython/issues/116263]:

`logging.handlers.RotatingFileHandler` no longer rolls over empty log files.

- gh-105376 [https://github.com/python/cpython/issues/105376]: Restore the deprecated **`logging`** `warn()` method. It was removed in Python 3.13 alpha 1. Keep the deprecated `warn()` method in Python 3.13. Patch by Victor Stinner.

- gh-122311 [https://github.com/python/cpython/issues/122311]: Improve errors in the **`pickle`** module. **`PicklingError`** is now raised more often instead of **`UnicodeEncodeError`**, **`ValueError`** and **`AttributeError`**, and the original exception is chained to it. Improve and unify error messages in Python and C implementations.

- gh-122744 [https://github.com/python/cpython/issues/122744]: Bump the version of pip bundled in ensurepip to version 24.2.

- gh-118761 [https://github.com/python/cpython/issues/118761]: Improve import time of **`pprint`** by around seven times. Patch by Hugo van Kemenade.

- gh-118974 [https://github.com/python/cpython/issues/118974]: Add `decorator` parameter to **`dataclasses.make_dataclass()`** to customize the functional creation of dataclasses.

- gh-118814 [https://github.com/python/cpython/issues/118814]: Fix the **`typing.TypeVar`** constructor when name is passed by keyword.

- gh-122637 [https://github.com/python/cpython/issues/122637]: Adjust `cmath.tanh(nanj)` and `cmath.tanh(infj)` for recent C standards.

- gh-122478 [https://github.com/python/cpython/issues/122478]: Remove internal frames from tracebacks shown in **`code.InteractiveInterpreter`** with non-default **`sys.excepthook()`**. Save correct tracebacks in **`sys.last_traceback`** and update `__traceback__` attribute of **`sys.last_value`** and **`sys.last_exc`**.

- gh-116622 [https://github.com/python/cpython/issues/116622]: On Android, the `FICLONE` and `FICLONERANGE` constants are no longer exposed by `fcntl`, as these ioctls are blocked by SELinux.

- gh-82378 [https://github.com/python/cpython/issues/82378]: Make sure that the new REPL interprets `sys.tracebacklimit` in the same way that the classic REPL did.

- gh-122334 [https://github.com/python/cpython/issues/122334]: Fix crash when importing `ssl` after the main interpreter restarts.

- gh-122459 [https://github.com/python/cpython/issues/122459]: Optimize `pickling` by name objects without the `__module__` attribute.

- gh-87320 [https://github.com/python/cpython/issues/87320]: In `code.InteractiveInterpreter`, handle exceptions caused by calling a non-default `sys.excepthook()`. Before, the exception bubbled up to the caller, ending the REPL.

- gh-122272 [https://github.com/python/cpython/issues/122272]: On some platforms such as Linux, year with century was not 0-padded when formatted by `strftime()` with C99-specific specifiers `'%C'` or `'%F'`. The 0-padding behavior is now guaranteed when the format specifiers `'%C'` and `'%F'` are supported by the C library. Patch by Ben Hsing

- gh-122400 [https://github.com/python/cpython/issues/122400]: Handle `ValueError`s raised by `os.stat()` in `filecmp.dircmp` and `filecmp.cmpfiles()`. Patch by Bénédikt Tran.

- gh-122311 [https://github.com/python/cpython/issues/122311]: Fix some error messages in `pickle`.

- gh-121650 [https://github.com/python/cpython/issues/121650]: `email` headers with embedded newlines are now quoted on output. The `generator` will now refuse to serialize (write) headers that are unsafely folded or delimited; see

`verify_generated_headers`. (Contributed by Bas Bloemsaat and Petr Viktorin in gh-121650 [https://github.com/python/cpython/issues/121650].)

- gh-122332 [https://github.com/python/cpython/issues/122332]: Fixed segfault with **`asyncio.Task.get_coro()`** when using an eager task factory.

- gh-105733 [https://github.com/python/cpython/issues/105733]: **`ctypes.ARRAY()`** is now soft deprecated: it no longer emits deprecation warnings and is not scheduled for removal.

- gh-122213 [https://github.com/python/cpython/issues/122213]: Add notes for pickle serialization errors that allow to identify the source of the error.

- gh-119180 [https://github.com/python/cpython/issues/119180]: As part of **PEP 749** [https://peps.python.org/pep-0749/], add the following attributes for customizing evaluation of annotation scopes:
  - `evaluate_value` on **`typing.TypeAliasType`**
  - `evaluate_bound`, `evaluate_constraints`, and `evaluate_default` on **`typing.TypeVar`**
  - `evaluate_default` on **`typing.ParamSpec`**
  - `evaluate_default` on **`typing.TypeVarTuple`**

- gh-119180 [https://github.com/python/cpython/issues/119180]: Fix handling of classes with custom metaclasses in `annotationlib.get_annotations`.

- gh-122170 [https://github.com/python/cpython/issues/122170]: Handle **`ValueError`**s raised by **`os.stat()`** in **`linecache`**. Patch by Bénédikt Tran.

- gh-122163 [https://github.com/python/cpython/issues/122163]: Add notes for JSON serialization errors that allow to identify the source of the error.

- gh-122129 [https://github.com/python/cpython/issues/122129]: Improve support of method descriptors and wrappers in the help title.

- gh-122145 [https://github.com/python/cpython/issues/122145]: Fix an issue when reporting tracebacks corresponding to Python code emitting an empty AST body. Patch by Nikita Sobolev and Bénédikt Tran.

- gh-121723 [https://github.com/python/cpython/issues/121723]: Make `logging.config.dictConfig()` accept any object implementing the Queue public API. See the queue configuration section for details. Patch by Bénédikt Tran.

- gh-82951 [https://github.com/python/cpython/issues/82951]: Serializing objects with complex `__qualname__` (such as unbound methods and nested classes) by name no longer involves serializing parent objects by value in pickle protocols < 4.

- gh-120754 [https://github.com/python/cpython/issues/120754]: `Pathlib.read_bytes` no longer opens the file in Python's buffered I/O mode. This reduces overheads as the code reads a file in whole leading to a modest speedup.

- gh-113785 [https://github.com/python/cpython/issues/113785]: `csv` now correctly parses numeric fields (when used with `csv.QUOTE_NONNUMERIC` or `csv.QUOTE_STRINGS`) which start with an escape character.

- gh-122088 [https://github.com/python/cpython/issues/122088]: `@warnings.deprecated` now copies the coroutine status of functions and methods so that `inspect.iscoroutinefunction()` returns the correct result.

- gh-122081 [https://github.com/python/cpython/issues/122081]: Fix a crash in the `decimal.IEEEContext()` optional function available via the `EXTRA_FUNCTIONALITY` configuration flag.

- gh-73991 [https://github.com/python/cpython/issues/73991]: Add `pathlib.Path.move()`, which moves a file or directory tree.

- gh-121268 [https://github.com/python/cpython/issues/121268]: Remove workarounds for non-IEEE 754 systems in **`cmath`**.

- gh-119698 [https://github.com/python/cpython/issues/119698]: Due to the lack of interest for **`symtable.Class.get_methods()`**, the method is marked as deprecated and will be removed in Python 3.16. Patch by Bénédikt Tran.

- gh-121889 [https://github.com/python/cpython/issues/121889]: Adjusts `cmath.acosh(complex('0+nanj'))` for recent C standards.

- gh-121804 [https://github.com/python/cpython/issues/121804]: Correctly show error locations, when **`SyntaxError`** raised in new repl. Patch by Sergey B Kirpichev.

- gh-121797 [https://github.com/python/cpython/issues/121797]: Add alternative **`Fraction`** constructor **`Fraction.from_number()`**.

- gh-121798 [https://github.com/python/cpython/issues/121798]: Add alternative **`Decimal`** constructor **`Decimal.from_number()`**.

- gh-120930 [https://github.com/python/cpython/issues/120930]: Fixed a bug introduced by gh-92081 [https://github.com/python/cpython/issues/92081] that added an incorrect extra blank to encoded words occurring in wrapped headers.

- gh-57141 [https://github.com/python/cpython/issues/57141]: The *shallow* argument to **`filecmp.dircmp`** (new in Python 3.13) is now keyword-only.

- gh-121245 [https://github.com/python/cpython/issues/121245]: Simplify handling of the history file in `site.register_readline()` helper. The `CAN_USE_PYREPL` variable now will be initialized, when imported. Patch by Sergey B Kirpichev.

- gh-121249 [https://github.com/python/cpython/issues/121249]: Support the float complex and double complex C types in the `struct` module if the compiler has C11 complex arithmetic. Patch by Sergey B Kirpichev.

- gh-121486 [https://github.com/python/cpython/issues/121486]: `math` functions `isqrt()`, `log()`, `log2()` and `log10()` now support integers larger than `2**2**32` on 32-bit platforms.

- gh-121474 [https://github.com/python/cpython/issues/121474]: Fix missing sanity check for `parties` arg in `threading.Barrier` constructor. Patch by Clinton Christian (pygeek).

- gh-121450 [https://github.com/python/cpython/issues/121450]: Hard-coded breakpoints (`breakpoint()` and `pdb.set_trace()`) now reuse the most recent `Pdb` instance that calls `Pdb.set_trace()`, instead of creating a new one each time. As a result, all the instance specific data like `display` and `commands` are preserved across Hard-coded breakpoints.

- gh-119169 [https://github.com/python/cpython/issues/119169]: Slightly speed up `os.walk()` by simplifying exception handling.

- gh-121423 [https://github.com/python/cpython/issues/121423]: Improve import time of `socket` by lazy importing modules and writing `socket.errorTab` as a constant.

- gh-59110 [https://github.com/python/cpython/issues/59110]: `zipimport` supports now namespace packages when no directory entry exists.

- gh-119004 [https://github.com/python/cpython/issues/119004]: Fix a crash in OrderedDict.__eq__ when operands are mutated during the check. Patch by Bénédikt Tran.

- gh-121313 [https://github.com/python/cpython/issues/121313]: Limit the reading size in the `multiprocessing.connection.Connection` class to 64

KiB to prevent memory overallocation and unnecessary memory management system calls.

- gh-121332 [https://github.com/python/cpython/issues/121332]: Fix constructor of `ast` nodes with custom `_attributes`. Previously, passing custom attributes would raise a `DeprecationWarning`. Passing arguments to the constructor that are not in `_fields` or `_attributes` remains deprecated. Patch by Jelle Zijlstra.

- gh-121245 [https://github.com/python/cpython/issues/121245]: Fix a bug in the handling of the command history of the new REPL that caused the history file to be wiped at REPL exit.

- gh-121210 [https://github.com/python/cpython/issues/121210]: Handle AST nodes with missing runtime fields or attributes in `ast.compare()`. Patch by Bénédikt Tran.

- gh-121163 [https://github.com/python/cpython/issues/121163]: Add support for `all` as an valid `action` for `warnings.simplefilter()` and `warnings.filterwarnings()`.

- gh-121151 [https://github.com/python/cpython/issues/121151]: Fix wrapping of long usage text of arguments inside a mutually exclusive group in `argparse`.

- gh-121141 [https://github.com/python/cpython/issues/121141]: Add support for `copy.replace()` to AST nodes. Patch by Bénédikt Tran.

- gh-87744 [https://github.com/python/cpython/issues/87744]: Fix waitpid race while calling `send_signal()` in asyncio. Patch by Kumar Aditya.

- gh-121027 [https://github.com/python/cpython/issues/121027]: Add a future warning in `functools.partial.__get__()`. In future Python versions `functools.partial` will be a method descriptor.

- gh-121027 [https://github.com/python/cpython/issues/121027]: Make

the `functools.partial` object a method descriptor.

- gh-117784 [https://github.com/python/cpython/issues/117784]: CPython now detects whether its linked TLS library supports TLSv1.3 post-handshake authentication and disables that feature if support is lacking.
- gh-121025 [https://github.com/python/cpython/issues/121025]: Improve the `__repr__()` of `functools.partialmethod`. Patch by Bénédikt Tran.
- gh-121018 [https://github.com/python/cpython/issues/121018]: Fixed issues where **`argparse.ArgumentParser.parse_args()`** did not honor `exit_on_error=False`. Based on patch by Ben Hsing.
- gh-119614 [https://github.com/python/cpython/issues/119614]: Fix truncation of strings with embedded null characters in some internal operations in `tkinter`.
- gh-120910 [https://github.com/python/cpython/issues/120910]: When reading installed files from an egg, use `relative_to(walk_up=True)` to honor files installed outside of the installation root.
- gh-61103 [https://github.com/python/cpython/issues/61103]: Support float complex, double complex and long double complex C types in `ctypes` as `c_float_complex`, `c_double_complex` and `c_longdouble_complex` if the compiler has C11 complex arithmetic. Patch by Sergey B Kirpichev.
- gh-120888 [https://github.com/python/cpython/issues/120888]: Upgrade pip wheel bundled with ensurepip (pip 24.1.1)
- gh-101830 [https://github.com/python/cpython/issues/101830]: Accessing the `tkinter` object’s string representation no longer converts the underlying Tcl object to a string on Windows.

- gh-120678 [https://github.com/python/cpython/issues/120678]: Fix regression in the new REPL that meant that globals from files passed using the `-i` argument would not be included in the REPL's global namespace. Patch by Alex Waygood.

- gh-120811 [https://github.com/python/cpython/issues/120811]: Fix possible memory leak in `contextvars.Context.run()`.

- gh-120782 [https://github.com/python/cpython/issues/120782]: Fix wrong references of the `datetime` types after reloading the module.

- gh-120713 [https://github.com/python/cpython/issues/120713]: `datetime.datetime.strftime()` now 0-pads years with less than four digits for the format specifiers `%Y` and `%G` on Linux. Patch by Ben Hsing

- gh-120769 [https://github.com/python/cpython/issues/120769]: Make empty line in `pdb` repeats the last command even when the command is from `cmdqueue`.

- gh-120780 [https://github.com/python/cpython/issues/120780]: Show string value of LOAD_SPECIAL oparg in `dis` output.

- gh-41431 [https://github.com/python/cpython/issues/41431]: Add `datetime.time.strptime()` and `datetime.date.strptime()`. Contributed by Wannes Boeykens.

- gh-120743 [https://github.com/python/cpython/issues/120743]: Soft deprecate `os.popen()` and `os.spawn*` functions. They should no longer be used to write new code. The `subprocess` module is recommended instead. Patch by Victor Stinner.

- gh-120732 [https://github.com/python/cpython/issues/120732]: Fix `name` passing to `unittest.mock.Mock` object when using `unittest.mock.create_autospec()`.

- gh-111259 [https://github.com/python/cpython/issues/111259]: `re` now handles patterns like `"[\s\S]"` or `"\s|\S"` which match any character as effectively as a dot with the `DOTALL` modifier (`"(?s:.)"`).

- gh-120683 [https://github.com/python/cpython/issues/120683]: Fix an error in **`logging.LogRecord`**, when the integer part of the timestamp is rounded up, while the millisecond calculation truncates, causing the log timestamp to be wrong by up to 999 ms (affected roughly 1 in 8 million timestamps).

- gh-118710 [https://github.com/python/cpython/issues/118710]: **`ipaddress.IPv4Address`** and **`ipaddress.IPv6Address`** attributes `version` and `max_prefixlen` are now available on the class.

- gh-120633 [https://github.com/python/cpython/issues/120633]: Move scrollbar and remove tear-off menus in turtledemo.

- gh-120606 [https://github.com/python/cpython/issues/120606]: Allow users to use EOF to exit `commands` definition in **`pdb`**

- gh-120284 [https://github.com/python/cpython/issues/120284]: Allow **`asyncio.Runner.run()`** to accept awaitable objects instead of simply coroutines.

- gh-120541 [https://github.com/python/cpython/issues/120541]: Improve the prompt in the «less» pager when **`help()`** is called with non-string argument.

- gh-120495 [https://github.com/python/cpython/issues/120495]: Fix incorrect exception handling in Tab Nanny. Patch by Wulian233.

- gh-120388 [https://github.com/python/cpython/issues/120388]: Improve a warning message when a test method in **`unittest`** returns something other than `None`. Now we show the returned object type and optional asyncio-related tip.

- gh-120381 [https://github.com/python/cpython/issues/120381]: Correct **`inspect.ismethoddescriptor()`** to check also for the lack of **`__delete__()`**. Patch by Jan Kaliszewski.

- gh-90425 [https://github.com/python/cpython/issues/90425]: The OS byte in gzip headers is now always set to 255 when using `gzip.compress()`.

- gh-120343 [https://github.com/python/cpython/issues/120343]: Fix column offset reporting for tokens that come after multiline f-strings in the `tokenize` module.

- gh-119180 [https://github.com/python/cpython/issues/119180]: As part of implementing **PEP 649** [https://peps.python.org/pep-0649/] and **PEP 749** [https://peps.python.org/pep-0749/], add a new module `annotationlib`. Add support for unresolved forward references in annotations to `dataclasses`, `typing.TypedDict`, and `typing.NamedTuple`.

- gh-119600 [https://github.com/python/cpython/issues/119600]: Fix `unittest.mock.patch()` to not read attributes of the target when `new_callable` is set. Patch by Robert Collins.

- gh-120289 [https://github.com/python/cpython/issues/120289]: Fixed the use-after-free issue in `cProfile` by disallowing `disable()` and `clear()` in external timers.

- gh-82017 [https://github.com/python/cpython/issues/82017]: Added support for converting any objects that have the `as_integer_ratio()` method to a `Fraction`.

- gh-114053 [https://github.com/python/cpython/issues/114053]: Fix edge-case bug where `typing.get_type_hints()` would produce incorrect results if type parameters in a class scope were overridden by assignments in a class scope and `from __future__ import annotations` semantics were enabled. Patch by Alex Waygood.

- gh-114053 [https://github.com/python/cpython/issues/114053]: Fix erroneous `NameError` when calling `inspect.get_annotations()` with `eval_str=True`` on a class that made use of **PEP 695** [https://peps.python.org/pep-0695/] type parameters in a module that had `from __future__ import annotations` at the top of the file. Patch by Alex Waygood.

- gh-120268 [https://github.com/python/cpython/issues/120268]: Prohibit passing `None` to pure-Python `datetime.date.fromtimestamp()` to achieve consistency with C-extension implementation.

- gh-120244 [https://github.com/python/cpython/issues/120244]: Fix memory leak in `re.sub()` when the replacement string contains backreferences.

- gh-120254 [https://github.com/python/cpython/issues/120254]: Added `commands` argument to `pdb.set_trace()` which allows users to send debugger commands from the source file.

- gh-120211 [https://github.com/python/cpython/issues/120211]: Fix `tkinter.ttk` with Tcl/Tk 9.0.

- gh-71587 [https://github.com/python/cpython/issues/71587]: Fix crash in C version of `datetime.datetime.strptime()` when called again on the restarted interpreter.

- gh-117983 [https://github.com/python/cpython/issues/117983]: Defer the `threading` import in `importlib.util` until lazy loading is used.

- gh-120157 [https://github.com/python/cpython/issues/120157]: Remove unused constant `concurrent.futures._base._FUTURE_STATES` in `concurrent.futures`. Patch by Clinton Christian (pygeek).

- gh-120161 [https://github.com/python/cpython/issues/120161]: `datetime` no longer crashes in certain complex reference cycle situations.

- gh-119698 [https://github.com/python/cpython/issues/119698]: Fix `symtable.Class.get_methods()` and document its behaviour. Patch by Bénédikt Tran.

- gh-120121 [https://github.com/python/cpython/issues/120121]: Add `concurrent.futures.InvalidStateError` to module's `__all__`.

- gh-119933 [https://github.com/python/cpython/issues/119933]: Add

the `symtable.SymbolTableType` enumeration to represent the possible outputs of the `symtable.SymbolTable.get_type` method. Patch by Bénédikt Tran.

- gh-120029 [https://github.com/python/cpython/issues/120029]: Expose `symtable.Symbol` methods `is_free_class()`, `is_comp_iter()` and `is_comp_cell()`. Patch by Bénédikt Tran.

- gh-120108 [https://github.com/python/cpython/issues/120108]: Fix calling `copy.deepcopy()` on `ast` trees that have been modified to have references to parent nodes. Patch by Jelle Zijlstra.

- gh-120056 [https://github.com/python/cpython/issues/120056]: Add `socket.IP_RECVERR` and `socket.IP_RECVTTL` constants (both available since Linux 2.2). And `socket.IP_RECVORIGDSTADDR` constant (available since Linux 2.6.29).

- gh-112672 [https://github.com/python/cpython/issues/112672]: Support building `tkinter` with Tcl 9.0.

- gh-120057 [https://github.com/python/cpython/issues/120057]: Added the `os.environ.refresh()` method to update `os.environ` with changes to the environment made by `os.putenv()`, by `os.unsetenv()`, or made outside Python in the same process. Patch by Victor Stinner.

- gh-120029 [https://github.com/python/cpython/issues/120029]: Expose `symtable.Symbol.is_type_parameter()` in the `symtable` module. Patch by Bénédikt Tran.

- gh-119819 [https://github.com/python/cpython/issues/119819]: Fix regression to allow logging configuration with multiprocessing queue types.

- gh-65454 [https://github.com/python/cpython/issues/65454]: `unittest.mock.Mock.attach_mock()` no longer triggers a call to a `PropertyMock` being attached.

- gh-117142 [https://github.com/python/cpython/issues/117142]: The **`ctypes`** module may now be imported in all subinterpreters, including those that have their own GIL.

- gh-118835 [https://github.com/python/cpython/issues/118835]: Fix _pyrepl crash when using custom prompt with ANSI escape codes.

- gh-81936 [https://github.com/python/cpython/issues/81936]: **`help()`** and **`showtopic()`** methods now respect a configured *output* argument to **`pydoc.Helper`** and not use the pager in such cases. Patch by Enrico Tröger.

- gh-117398 [https://github.com/python/cpython/issues/117398]: The `_datetime` module (C implementation for **`datetime`**) now supports being imported in multiple interpreters.

- gh-119824 [https://github.com/python/cpython/issues/119824]: Print stack entry in **`pdb`** when and only when user input is needed.

- gh-119838 [https://github.com/python/cpython/issues/119838]: In mixed arithmetic operations with **`Fraction`** and complex, the fraction is now converted to **`float`** instead of **`complex`**.

- gh-119770 [https://github.com/python/cpython/issues/119770]: Make **`termios`** `ioctl()` constants positive. Patch by Victor Stinner.

- gh-89727 [https://github.com/python/cpython/issues/89727]: Fix issue with **`shutil.rmtree()`** where a **`RecursionError`** is raised on deep directory trees.

- gh-119577 [https://github.com/python/cpython/issues/119577]: The **`DeprecationWarning`** emitted when testing the truth value of an **`xml.etree.ElementTree.Element`** now describes unconditionally returning `True` in a future version rather than raising an exception in Python 3.14.

- gh-89727 [https://github.com/python/cpython/issues/89727]: Partially fix issue with **`shutil.rmtree()`** where a **`RecursionError`** is raised on deep directory trees. A recursion error is no longer raised when **`rmtree.avoids_symlink_attacks`** is false.

- gh-93963 [https://github.com/python/cpython/issues/93963]: Remove deprecated names from `importlib.abc` as found in `importlib.resources.abc`.

- gh-119118 [https://github.com/python/cpython/issues/119118]: Fix performance regression in the **`tokenize`** module by caching the `line` token attribute and calculating the column offset more efficiently.

- gh-89727 [https://github.com/python/cpython/issues/89727]: Fix issue with **`os.fwalk()`** where a **`RecursionError`** was raised on deep directory trees by adjusting the implementation to be iterative instead of recursive.

- gh-119594 [https://github.com/python/cpython/issues/119594]: If one calls pow(fractions.Fraction, x, module) with modulo not None, the error message now says that the types are incompatible rather than saying pow only takes 2 arguments. Patch by Wim Jeantine-Glenn and Mark Dickinson.

- gh-119588 [https://github.com/python/cpython/issues/119588]: `zipfile.Path.is_symlink` now assesses if the given path is a symlink.

- gh-119562 [https://github.com/python/cpython/issues/119562]: Remove **`ast.Num`**, **`ast.Str`**, **`ast.Bytes`**, **`ast.NameConstant`** and **`ast.Ellipsis`**. They had all emitted deprecation warnings since Python 3.12. Patch by Alex Waygood.

- gh-119555 [https://github.com/python/cpython/issues/119555]: Catch **`SyntaxError`** from **`compile()`** in the runsource() method of the InteractiveColoredConsole. Patch by Sergey B Kirpichev.

- gh-118908 [https://github.com/python/cpython/issues/118908]: Limit exposed globals from internal imports and definitions on new REPL startup. Patch by Eugene Triguba and Pablo Galindo.

- gh-117865 [https://github.com/python/cpython/issues/117865]: Improve the import time of the `ast` module by deferring the import of `re`. Patch by Jelle Zijlstra.

- gh-119127 [https://github.com/python/cpython/issues/119127]: Positional arguments of `functools.partial()` objects now support placeholders via `functools.Placeholder`.

- gh-113892 [https://github.com/python/cpython/issues/113892]: Now, the method `sock_connect` of `asyncio.ProactorEventLoop` raises a `ValueError` if given socket is not in non-blocking mode, as well as in other loop implementations.

- gh-119506 [https://github.com/python/cpython/issues/119506]: Fix **`io.TextIOWrapper.write()`** method breaks internal buffer when the method is called again during flushing internal buffer.

- gh-69214 [https://github.com/python/cpython/issues/69214]: Fix `fcntl.ioctl()` *request* parameter: use an `unsigned long` instead of an `unsigned int` for the *request* parameter of `fcntl.ioctl()` to support requests larger than `UINT_MAX`. Patch by Victor Stinner.

- gh-119105 [https://github.com/python/cpython/issues/119105]: `difflib`'s `DIffer.compare()` (and so also `ndiff`) can no longer be provoked into cubic-time behavior, or into unbounded recursion, and should generally be faster in ordinary cases too. Results may change in some cases, although that should be rare. Correctness of diffs is not affected. Some similar lines far apart may be reported as deleting one and adding the other, where before they were displayed on adjacent output lines with markup showing the intraline differences.

- gh-119443 [https://github.com/python/cpython/issues/119443]: The interactive REPL no longer runs with `from __future__ import annotations` enabled. Patch by Jelle Zijlstra.

- gh-119461 [https://github.com/python/cpython/issues/119461]: Add

`socket.VMADDR_CID_LOCAL` constant. Patch by Victor Stinner.

- gh-117398 [https://github.com/python/cpython/issues/117398]: Objects in the datetime C-API are now all statically allocated, which means better memory safety, especially when the module is reloaded. This should be transparent to users.
- gh-118894 [https://github.com/python/cpython/issues/118894]: **`asyncio`** REPL now has the same capabilities as PyREPL.
- gh-118830 [https://github.com/python/cpython/issues/118830]: Bump **`pickle`** default protocol to `5`.
- gh-118911 [https://github.com/python/cpython/issues/118911]: In PyREPL, updated `maybe-accept`'s logic so that if the user hits `Enter` twice, they are able to terminate the block even if there's trailing whitespace. Also, now when the user hits arrow up, the cursor is on the last functional line. This matches IPython's behavior. Patch by Aya Elsayed.
- gh-115225 [https://github.com/python/cpython/issues/115225]: Raise error on certain technically valid but pathological ISO 8601 strings passed to **`datetime.time.fromisoformat()`** that were previously parsed incorrectly.
- gh-111201 [https://github.com/python/cpython/issues/111201]: Remove dependency to **`readline`** from the new Python REPL.
- gh-119189 [https://github.com/python/cpython/issues/119189]: When using the `**` operator or **`pow()`** with **`Fraction`** as the base and an exponent that is not rational, a float, or a complex, the fraction is no longer converted to a float.
- gh-119174 [https://github.com/python/cpython/issues/119174]: Fix high DPI causes turtledemo(turtle-graphics examples) windows blurry Patch by Wulian233 and Terry Jan Reedy
- gh-119121 [https://github.com/python/cpython/issues/119121]: Fix a NameError happening in

`asyncio.staggered.staggered_race`. This function is now tested.

- gh-119105 [https://github.com/python/cpython/issues/119105]: `difflib.Differ` is much faster for some cases of diffs where many pairs of lines are equally similar.

- gh-119113 [https://github.com/python/cpython/issues/119113]: Fix issue where **`pathlib.PurePath.with_suffix()`** didn't raise **`TypeError`** when given `None` as a suffix.

- gh-118643 [https://github.com/python/cpython/issues/118643]: Fix an AttributeError in the **`email`** module when re-fold a long address list. Also fix more cases of incorrect encoding of the address separator in the address list.

- gh-73991 [https://github.com/python/cpython/issues/73991]: Add **`pathlib.Path.copy()`**, which copies a file or directory to another.

- gh-58933 [https://github.com/python/cpython/issues/58933]: Make **`pdb`** return to caller frame correctly when `f_trace` of the caller frame is not set

- gh-82805 [https://github.com/python/cpython/issues/82805]: Support single-dot file extensions in **`pathlib.PurePath.suffix`** and related attributes and methods. For example, the **`suffixes`** of `PurePath('foo.bar.')` are now `['.bar', '.']` rather than `[]`. This brings file extension splitting in line with **`os.path.splitext()`**.

- gh-118924 [https://github.com/python/cpython/issues/118924]: Remove **`version`** and **`version_info`** from **`sqlite3`**. Patch by Hugo van Kemenade.

- gh-118928 [https://github.com/python/cpython/issues/118928]: Disallow using a sequence of parameters with named placeholders in **`sqlite3`** queries. Patch by Erlend E. Aasland.

- gh-118895 [https://github.com/python/cpython/issues/118895]:

Setting attributes on `typing.NoDefault` now raises `AttributeError` instead of `TypeError`.

- gh-118868 [https://github.com/python/cpython/issues/118868]: Fixed issue where kwargs were no longer passed to the logging handler QueueHandler

- gh-118827 [https://github.com/python/cpython/issues/118827]: Remove deprecated `Quoter` class from `urllib.parse`. It had previously raised a `DeprecationWarning` since Python 3.11. Patch by Nikita Sobolev.

- gh-118824 [https://github.com/python/cpython/issues/118824]: Remove deprecated `pty.master_open()` and `pty.slave_open()`. Use `pty.openpty()` instead. Patch by Nikita Sobolev.

- gh-118851 [https://github.com/python/cpython/issues/118851]: `ctx` arguments to the constructors of `ast` node classes now default to `ast.Load()`. Patch by Jelle Zijlstra.

- gh-101588 [https://github.com/python/cpython/issues/101588]: Remove copy, deepcopy, and pickle from itertools. These had previously raised a DeprecationWarning since Python 3.12.

- gh-118805 [https://github.com/python/cpython/issues/118805]: Remove *type*, *choices*, and *metavar* parameters of `argparse.BooleanOptionalAction`. They were deprecated since Python 3.12.

- gh-118803 [https://github.com/python/cpython/issues/118803]: `typing.ByteString` and `collections.abc.ByteString` are removed. They had previously raised a `DeprecationWarning` since Python 3.12.

- gh-118798 [https://github.com/python/cpython/issues/118798]: The *isdst* parameter has been removed from `email.utils.localtime()`. Patch by Hugo van Kemenade.

- gh-118760 [https://github.com/python/cpython/issues/118760]: Restore the default value of `tkiter.wantobjects` to `1`.
- gh-118760 [https://github.com/python/cpython/issues/118760]: Fix errors in calling Tkinter bindings on Windows.
- gh-74033 [https://github.com/python/cpython/issues/74033]: Drop support for passing keyword arguments to **`pathlib.Path`**.
- gh-101357 [https://github.com/python/cpython/issues/101357]: Suppress all **`OSError`** exceptions from **`pathlib.Path.exists()`** and `is_*()` methods, rather than a selection of more common errors. The new behaviour is consistent with **`os.path.exists()`**, **`os.path.isdir()`**, etc. Use **`pathlib.Path.stat()`** to retrieve the file status without suppressing exceptions.
- gh-78707 [https://github.com/python/cpython/issues/78707]: Drop support for passing additional positional arguments to **`pathlib.PurePath.relative_to()`** and **`is_relative_to()`**.
- gh-118507 [https://github.com/python/cpython/issues/118507]: Fix **`os.path.isfile()`** on Windows for pipes. Speedup **`os.path.isjunction()`** and **`os.path.lexists()`** on Windows with a native implementation.
- gh-118772 [https://github.com/python/cpython/issues/118772]: Allow **`typing.TypeVar`** instances without a default to follow instances without a default in some cases. Patch by Jelle Zijlstra.
- gh-118714 [https://github.com/python/cpython/issues/118714]: Allow `restart` in post-mortem debugging of **`pdb`**. Removed restart message when the user quits pdb from post-mortem mode.
- gh-118673 [https://github.com/python/cpython/issues/118673]: Removed executable bits and shebang from stdlib modules.
- gh-110863 [https://github.com/python/cpython/issues/110863]: **`os.path.realpath()`** now suppresses any **`OSError`** from

`os.readlink()` when *strict* mode is disabled (the default).

- gh-118263 [https://github.com/python/cpython/issues/118263]: Speed up **`os.path.splitroot()`** & **`os.path.normpath()`** with a direct C call.

- gh-110190 [https://github.com/python/cpython/issues/110190]: Fix ctypes structs with array on SPARC by setting `MAX_STRUCT_SIZE` to 32 in stgdict. Patch by Jakub Kulik

- gh-118033 [https://github.com/python/cpython/issues/118033]: Fix **`dataclasses.dataclass()`** not creating a `__weakref__` slot when subclassing **`typing.Generic`**.

- gh-117766 [https://github.com/python/cpython/issues/117766]: Always use **`str()`** to print `choices` in **`argparse`**.

- gh-106531 [https://github.com/python/cpython/issues/106531]: In **`importlib.resources`**, sync with importlib_resources 6.3.2 [https://importlib-resources.readthedocs.io/en/latest/history.html#v6-3-2], including: `MultiplexedPath` now expects `Traversable` paths, deprecating string arguments to `MultiplexedPath`; Enabled support for resources in namespace packages in zip files; Fixed `NotADirectoryError` when calling files on a subdirectory of a namespace package.

- gh-115937 [https://github.com/python/cpython/issues/115937]: Removed extra preprocessing for the `__signature__` attribute: the code just check if it's a **`inspect.Signature`** instance. Patch by Sergey B Kirpichev.

- gh-115808 [https://github.com/python/cpython/issues/115808]: Add **`operator.is_none()`** and **`operator.is_not_none()`** functions.

- gh-114264 [https://github.com/python/cpython/issues/114264]: Improve performance of **`copy.deepcopy()`** by adding a fast path for atomic types.

- gh-113878 [https://github.com/python/cpython/issues/113878]: Add

doc parameter to **`dataclasses.field()`**, so it can be stored and shown as a documentation / metadata. If `@dataclass(slots=True)` is used, then the supplied string is available in the **`__slots__`** dict. Otherwise, the supplied string is only available in the corresponding **`dataclasses.Field`** object.

In order to support this feature we are changing the `__slots__` format in dataclasses from **`tuple`** to **`dict`** when documentation / metadata is present.

- gh-113978 [https://github.com/python/cpython/issues/113978]: Ignore warnings on text completion inside REPL.
- gh-113008 [https://github.com/python/cpython/issues/113008]: Correct argparse usage output for required, mutually exclusive groups containing a positional argument
- gh-108172 [https://github.com/python/cpython/issues/108172]: `webbrowser` honors OS preferred browser on Linux when its desktop entry name contains the text of a known browser name.
- gh-109109 [https://github.com/python/cpython/issues/109109]: You can now get the raw TLS certificate chains from TLS connections via **`ssl.SSLSocket.get_verified_chain()`** and **`ssl.SSLSocket.get_unverified_chain()`** methods.

  Contributed by Mateusz Nowak.
- gh-105623 [https://github.com/python/cpython/issues/105623]: Fix performance degradation in **`logging.handlers.RotatingFileHandler`**. Patch by Craig Robson.
- gh-102450 [https://github.com/python/cpython/issues/102450]: Add missing ISO-8601 24:00 alternative to midnight of next day to **`datetime.datetime.fromisoformat()`** and **`datetime.time.fromisoformat()`**. Patch by Izan «TizzySaurus» Robinson (tizzysaurus@gmail.com)

- gh-103956 [https://github.com/python/cpython/issues/103956]: Fix lack of newline characters in `trace` module output when line tracing is enabled but source code line for current frame is not available.

- gh-92081 [https://github.com/python/cpython/issues/92081]: Fix missing spaces in email headers when the spaces are mixed with encoded 8-bit characters.

- gh-103194 [https://github.com/python/cpython/issues/103194]: Prepare Tkinter for C API changes in Tcl 8.7/9.0 to avoid `_tkinter.Tcl_Obj` being unexpectedly returned instead of `bool`, `str`, `bytearray`, or `int`.

- gh-87106 [https://github.com/python/cpython/issues/87106]: Fixed handling in `inspect.Signature.bind()` of keyword arguments having the same name as positional-only arguments when a variadic keyword argument (e.g. `**kwargs`) is present.

- gh-103134 [https://github.com/python/cpython/issues/103134]: Add additional methods to proxy objects in the `multiprocessing` module:

    - `clear()` and `copy()` for proxies of `list`
    - `fromkeys()`, `reversed(d)`, `d | {}`, `{} | d`, `d |= {'b': 2}` for proxies of `dict`

- bpo-45767 [https://bugs.python.org/issue?@action=redirect&bpo=45767]: Fix integer conversion in `os.major()`, `os.minor()`, and `os.makedev()`. Support device numbers larger than `2**63-1`. Support non-existent device number (`NODEV`).

- bpo-44864 [https://bugs.python.org/issue?@action=redirect&bpo=44864]: Do not translate user-provided strings in `argparse.ArgumentParser`.

- bpo-41843 [https://bugs.python.org/issue?@action=redirect&bpo=41843]: Solaris now uses `os.sendfile()` fast-copy syscall for more efficient `shutil` file copy related functions.

- bpo-15987 [https://bugs.python.org/issue?@action=redirect&bpo=15987]: Implemented `ast.compare()` for comparing two ASTs. Patch by Batuhan Taskaya with some help from Jeremy Hylton.

- gh-67693 [https://github.com/python/cpython/issues/67693]: Fix `urllib.parse.urlunparse()` and `urllib.parse.urlunsplit()` for URIs with path starting with multiple slashes and no authority. Based on patch by Ashwin Ramaswami.

- bpo-14074 [https://bugs.python.org/issue?@action=redirect&bpo=14074]: Fix `argparse` metavar processing to allow positional arguments to have a tuple metavar.

IDLE

- gh-122392 [https://github.com/python/cpython/issues/122392]: Increase currently inadequate vertical spacing for the IDLE browsers (path, module, and stack) on high-resolution monitors.
- gh-112938 [https://github.com/python/cpython/issues/112938]: Fix uninteruptable hang when Shell gets rapid continuous output.
- gh-122482 [https://github.com/python/cpython/issues/122482]: Change About IDLE to direct users to discuss.python.org instead of the now unused idle-dev email and mailing list.
- gh-78889 [https://github.com/python/cpython/issues/78889]: Stop Shell freezes by blocking user access to non-method sys.stdout.shell attributes, which are all private.
- gh-120083 [https://github.com/python/cpython/issues/120083]: Add explicit black IDLE Hovertip foreground color needed for recent macOS. Fixes Sonoma showing unreadable white on pale yellow. Patch by John Riggles.
- gh-120104 [https://github.com/python/cpython/issues/120104]: Fix padding in config and search dialog windows in IDLE.

Documentation

- gh-124872 [https://github.com/python/cpython/issues/124872]: Added definitions for context, current context, and context management protocol, updated related definitions to be consistent, and expanded the documentation for **`contextvars.Context`**.

- gh-125018 [https://github.com/python/cpython/issues/125018]: The **`importlib.metadata`** documentation now includes semantic cross-reference targets for the significant documented APIs. This means intersphinx references like **`importlib.metadata.version()`** will now work as expected.

- gh-124720 [https://github.com/python/cpython/issues/124720]: Update «Using Python on a Mac» section of the «Python Setup and Usage» document and include information on installing free-threading support.

- gh-124457 [https://github.com/python/cpython/issues/124457]: Remove coverity scan from the CPython repo. It has not been used since 2020 and is currently unmaintained.

- gh-116622 [https://github.com/python/cpython/issues/116622]: Add an Android platform guide, and flag modules not available on Android.

- gh-123976 [https://github.com/python/cpython/issues/123976]: Refresh docs around custom providers.

- gh-70870 [https://github.com/python/cpython/issues/70870]: Clarified the dual usage of the term «free variable» (both the formal meaning of any reference to names defined outside the local scope, and the narrower pragmatic meaning of nonlocal variables named in `co_freevars`).

- gh-121277 [https://github.com/python/cpython/issues/121277]: Writers of CPython's documentation can now use `next` as the version for the `versionchanged`, `versionadded`, `deprecated` directives.

- gh-117765 [https://github.com/python/cpython/issues/117765]:

Improved documentation for `unittest.mock.patch.dict()`

- gh-121749 [https://github.com/python/cpython/issues/121749]: Fix documentation for `PyModule_AddObjectRef()`.
- gh-120012 [https://github.com/python/cpython/issues/120012]: Clarify the behaviours of `multiprocessing.Queue.empty()` and `multiprocessing.SimpleQueue.empty()` on closed queues. Patch by Bénédikt Tran.
- gh-119574 [https://github.com/python/cpython/issues/119574]: Added some missing environment variables to the output of `--help-env`.
- bpo-34008 [https://bugs.python.org/issue?@action=redirect&bpo=34008]: The `Py_Main()` documentation moved from the «Very High Level API» section to the «Initialization and Finalization» section.

  Also make it explicit that we expect `Py_Main` to typically be called instead of `Py_Initialize` rather than after it (since `Py_Main` makes its own call to `Py_Initialize`). Document that calling both is supported but is version dependent on which settings will be applied correctly.

Core and Builtins

- gh-124375 [https://github.com/python/cpython/issues/124375]: Fix a crash in the free threading build when the GC runs concurrently with a new thread starting.
- gh-125221 [https://github.com/python/cpython/issues/125221]: Fix possible race condition when calling `__reduce_ex__()` for the first time in the free threading build.
- gh-125174 [https://github.com/python/cpython/issues/125174]: Make the handling of reference counts of immortal objects more robust. Immortal objects with reference counts that deviate from their original reference count by up to a billion (half a billion on 32 bit builds) are still counted as immortal.
- gh-125039 [https://github.com/python/cpython/issues/125039]: Make

`this_instr` and `prev_instr` const in cases generator.

Library

- gh-125008 [https://github.com/python/cpython/issues/125008]: Fix `tokenize.untokenize()` producing invalid syntax for double braces preceded by certain escape characters.

Core and Builtins

- gh-124871 [https://github.com/python/cpython/issues/124871]: Fix compiler bug (in some versions of 3.13) where an assertion fails during reachability analysis.
- gh-123378 [https://github.com/python/cpython/issues/123378]: Fix a crash in the `__str__()` method of `UnicodeError` objects when the `UnicodeError.start` and `UnicodeError.end` values are invalid or out-of-range. Patch by Bénédikt Tran.
- gh-118093 [https://github.com/python/cpython/issues/118093]: Improve the experimental JIT compiler's ability to stay «on trace» when encountering highly-biased branches.
- gh-124642 [https://github.com/python/cpython/issues/124642]: Fixed scalability issue in free-threaded builds for lock-free reads from dictionaries in multi-threaded scenarios
- gh-116510 [https://github.com/python/cpython/issues/116510]: Fix a crash caused by immortal interned strings being shared between sub-interpreters that use basic single-phase init. In that case, the string can be used by an interpreter that outlives the interpreter that created and interned it. For interpreters that share obmalloc state, also share the interned dict with the main interpreter.
- gh-116510 [https://github.com/python/cpython/issues/116510]: Fix a bug that can cause a crash when sub-interpreters use «basic» single-phase extension modules. Shared objects could refer to PyGC_Head nodes that had been freed as part of interpreter cleanup.
- gh-119180 [https://github.com/python/cpython/issues/119180]: The `__main__` module no longer always contains an `__annotations__` dictionary in its global namespace.
- gh-124547 [https://github.com/python/cpython/issues/124547]: When

deallocating an object with inline values whose `__dict__` is still live: if memory allocation for the inline values fails, clear the dictionary. Prevents an interpreter crash.

- gh-124513 [https://github.com/python/cpython/issues/124513]: Fix a crash in FrameLocalsProxy constructor: check the number of arguments. Patch by Victor Stinner.
- gh-124442 [https://github.com/python/cpython/issues/124442]: Fix nondeterminism in compilation by sorting the value of **`__static_attributes__`**. Patch by kp2pml30.
- gh-124285 [https://github.com/python/cpython/issues/124285]: Fix bug where `bool(a)` can be invoked more than once during the evaluation of a compound boolean expression.
- gh-123856 [https://github.com/python/cpython/issues/123856]: Fix PyREPL failure when a keyboard interrupt is triggered after using a history search
- gh-65961 [https://github.com/python/cpython/issues/65961]: Deprecate the setting and using `__package__` and `__cached__`.
- gh-119726 [https://github.com/python/cpython/issues/119726]: The JIT now generates more efficient code for calls to C functions resulting in up to 0.8% memory savings and 1.5% speed improvement on AArch64. Patch by Diego Russo.
- gh-122878 [https://github.com/python/cpython/issues/122878]: Use the `pager` binary, if available (e.g. on Debian and derivatives), to display REPL `help()`.
- gh-124188 [https://github.com/python/cpython/issues/124188]: Fix reading and decoding a line from the source file with non-UTF-8 encoding for syntax errors raised in the compiler.
- gh-124027 [https://github.com/python/cpython/issues/124027]: Support `<page up>`, `<page down>`, and `<delete>` keys in the Python REPL when `$TERM` is set to `vt100`.
- gh-124022 [https://github.com/python/cpython/issues/124022]: Fix bug where docstring is removed from classes in interactive mode.
- gh-123958 [https://github.com/python/cpython/issues/123958]: docstrings are now removed from the optimized AST in optimization level 2.
- gh-123923 [https://github.com/python/cpython/issues/123923]: The `f_executable` field in the internal **`_PyInterpreterFrame`** struct now uses a tagged pointer.

Profilers and debuggers that uses this field should clear the least significant bit to recover the PyObject* pointer.

- gh-77894 [https://github.com/python/cpython/issues/77894]: Fix possible crash in the garbage collector when it tries to break a reference loop containing a **`memoryview`** object. Now a **`memoryview`** object can only be cleared if there are no buffers that refer it.
- gh-120221 [https://github.com/python/cpython/issues/120221]: asyncio REPL is now again properly recognizing KeyboardInterrupts. Display of exceptions raised in secondary threads is fixed.
- gh-119310 [https://github.com/python/cpython/issues/119310]: Allow the new interactive shell to read history files written with the editline library that use unicode-escaped entries. Patch by aorcajo and Łukasz Langa.
- gh-123572 [https://github.com/python/cpython/issues/123572]: Fix key mappings for various F-keys in Windows for the new REPL. Patch by devdanzin

Library

- gh-123614 [https://github.com/python/cpython/issues/123614]: Add **`turtle.save()`** to easily save Turtle drawings as PostScript files. Patch by Marie Roald and Yngve Mardal Moe.

Core and Builtins

- gh-123339 [https://github.com/python/cpython/issues/123339]: Setting the **`__module__`** attribute for a class now removes the `__firstlineno__` item from the type's dict, so they will no longer be inconsistent.

- gh-119034 [https://github.com/python/cpython/issues/119034]: Change `<page up>` and `<page down>` keys of the Python REPL to history search forward/backward. Patch by Victor Stinner.

- gh-123562 [https://github.com/python/cpython/issues/123562]: Improve **`SyntaxError`** message for using `case ... as ...` with not a name.

- gh-123545 [https://github.com/python/cpython/issues/123545]: Fix a double decref in rare cases on experimental JIT builds.

- gh-123484 [https://github.com/python/cpython/issues/123484]: Fix `_Py_DebugOffsets` for long objects to be relative to the start of the object rather than the start of a subobject.

- gh-123446 [https://github.com/python/cpython/issues/123446]: Fix empty function name in **`TypeError`** when builtin magic methods are used without the required args.

- gh-123440 [https://github.com/python/cpython/issues/123440]: Improve **`SyntaxError`** message for using `except as` with not a name.

- gh-116017 [https://github.com/python/cpython/issues/116017]: Improved JIT memory consumption by periodically freeing memory used by infrequently-executed code. This change is especially likely to improve the memory footprint of long-running programs.

- gh-123344 [https://github.com/python/cpython/issues/123344]: Add AST optimizations for type parameter defaults.

- gh-123321 [https://github.com/python/cpython/issues/123321]: Prevent Parser/myreadline race condition from segfaulting on multi-threaded use. Patch by Bar Harel and Amit Wienner.

- gh-123177 [https://github.com/python/cpython/issues/123177]: Fix a bug causing stray prompts to appear in the middle of wrapped lines in the new REPL.

- gh-122982 [https://github.com/python/cpython/issues/122982]: Extend the deprecation period for bool inversion (~) by two years.

- gh-123271 [https://github.com/python/cpython/issues/123271]: Make concurrent iterations over the same **`zip()`** iterator safe under free-threading.

- gh-123275 [https://github.com/python/cpython/issues/123275]: Support **`-X gil=1`** and **`PYTHON_GIL=1`** on non-free-threaded builds.

- gh-123177 [https://github.com/python/cpython/issues/123177]: Deactivate line wrap in the Apple Terminal via a ANSI escape code. Patch by Pablo Galindo
- gh-123229 [https://github.com/python/cpython/issues/123229]: Fix valgrind warning by initializing the f-string buffers to 0 in the tokenizer. Patch by Pablo Galindo
- gh-122298 [https://github.com/python/cpython/issues/122298]: Restore printout of GC stats when `gc.set_debug(gc.DEBUG_STATS)` is called. This feature was accidentally removed when implementing incremental GC.
- gh-121804 [https://github.com/python/cpython/issues/121804]: Correctly show error locations when a **`SyntaxError`** is raised in the basic REPL. Patch by Sergey B Kirpichev.
- gh-115776 [https://github.com/python/cpython/issues/115776]: Enables inline values (Python’s equivalent of hidden classes) on any class who’s instances are of a fixed size.
- gh-123142 [https://github.com/python/cpython/issues/123142]: Fix too-wide source location in exception tracebacks coming from broken iterables in comprehensions.
- gh-123048 [https://github.com/python/cpython/issues/123048]: Fix a bug where pattern matching code could emit a **`JUMP_FORWARD`** with no source location.
- gh-118093 [https://github.com/python/cpython/issues/118093]: Break up `CALL_ALLOC_AND_ENTER_INIT` into micro-ops and relax requirement for exact args, in order to increase the amount of code supported by tier 2.
- gh-123123 [https://github.com/python/cpython/issues/123123]: Fix displaying **`SyntaxError`** exceptions covering multiple lines. Patch by Pablo Galindo
- gh-123083 [https://github.com/python/cpython/issues/123083]: Fix a

potential use-after-free in `STORE_ATTR_WITH_HINT`.

- gh-123022 [https://github.com/python/cpython/issues/123022]: Fix crash in free-threaded build when calling `Py_Initialize()` from a non-main thread.
- gh-118093 [https://github.com/python/cpython/issues/118093]: Add three specializations for `CALL_KW`:
  - `CALL_KW_PY` for calls to Python functions
  - `CALL_KW_BOUND_METHOD` for calls to bound methods
  - `CALL_KW_NON_PY` for all other calls
- gh-122821 [https://github.com/python/cpython/issues/122821]: Make sure that branches in `while` statements have consistent offsets for `sys.monitoring`. `while` statements are now compiled with a simple jump at the end of the body, instead of duplicating the test.
- gh-122907 [https://github.com/python/cpython/issues/122907]: Building with `HAVE_DYNAMIC_LOADING` now works as well as it did in 3.12. Existing deficiences will be addressed separately. (See https://github.com/python/cpython/issues/122950.)
- gh-122888 [https://github.com/python/cpython/issues/122888]: Fix crash on certain calls to `str()` with positional arguments of the wrong type. Patch by Jelle Zijlstra.
- gh-118093 [https://github.com/python/cpython/issues/118093]: Improve the experimental JIT's handling of polymorphic code.
- gh-122697 [https://github.com/python/cpython/issues/122697]: Fixed memory leaks at interpreter shutdown in the free-threaded build, and also reporting of leaked memory blocks via `-X showrefcount`.
- gh-116622 [https://github.com/python/cpython/issues/116622]: Fix Android stdout and stderr messages being truncated or lost.
- gh-122527 [https://github.com/python/cpython/issues/122527]: Fix a

crash that occurred when a `PyStructSequence` was deallocated after its type's dictionary was cleared by the GC. The type's **`tp_basicsize`** now accounts for non-sequence fields that aren't included in the **`Py_SIZE`** of the sequence.

- gh-122445 [https://github.com/python/cpython/issues/122445]: Add only fields which are modified via self.* to **`__static_attributes__`**.

- gh-122417 [https://github.com/python/cpython/issues/122417]: In the free-threaded build, the reference counts for heap type objects are now partially stored in a distributed manner in per-thread arrays. This reduces contention on the heap type's reference count fields when creating or destroying instances of the same type from multiple threads concurrently.

- gh-116090 [https://github.com/python/cpython/issues/116090]: Fix an issue in JIT builds that prevented some **`for`** loops from correctly firing **`RAISE`** monitoring events.

- gh-122300 [https://github.com/python/cpython/issues/122300]: Preserve AST nodes for f-string with single-element format specifiers. Patch by Pablo Galindo

- gh-120906 [https://github.com/python/cpython/issues/120906]: **`frame.f_locals`** now supports arbitrary hashable objects as keys.

- gh-122239 [https://github.com/python/cpython/issues/122239]: When a **`list`**, **`tuple`** or **`dict`** with too many elements is unpacked, show the actual length in the error message.

- gh-122245 [https://github.com/python/cpython/issues/122245]: Detection of writes to `__debug__` is moved from the compiler's codegen stage to the symtable. This means that these errors are now detected even in code that is optimized away before codegen (such as assertions with the **`-O`** command line option).

- gh-122234 [https://github.com/python/cpython/issues/122234]: Specializations for sums with float and complex inputs in

`sum()` now always use compensated summation. Also, for integer items in above specializations: **`PyLong_AsDouble()`** is used, instead of **`PyLong_AsLongAndOverflow()`**. Patch by Sergey B Kirpichev.

- gh-122208 [https://github.com/python/cpython/issues/122208]: Dictionary watchers now only deliver the PyDict_EVENT_ADDED event when the insertion is in a known good state to succeed.

- gh-122160 [https://github.com/python/cpython/issues/122160]: Remove the `BUILD_CONST_KEY_MAP` opcode. Use **`BUILD_MAP`** instead.

- gh-122029 [https://github.com/python/cpython/issues/122029]: Emit `c_call` events in **`sys.setprofile()`** when a `PyMethodObject` pointing to a `PyCFunction` is called.

- gh-122026 [https://github.com/python/cpython/issues/122026]: Fix a bug that caused the tokenizer to not correctly identify mismatched parentheses inside f-strings in some situations. Patch by Pablo Galindo

- gh-99108 [https://github.com/python/cpython/issues/99108]: Python's hashlib now unconditionally uses the vendored HACL* library for Blake2. Python no longer accepts libb2 as an optional dependency for Blake2.

  We refreshed HACL* to the latest version, and now vendor HACL*"s 128-bit and 256-bit wide vector implementations for Blake2, which are used on x86/x64 toolchains when the required CPU features are available at runtime.

  HACL*"s 128-bit wide vector implementation of Blake2 can also run on ARM NEON and Power8, but lacking evidence of a performance gain, these are not enabled (yet).

Library

- gh-121999 [https://github.com/python/cpython/issues/121999]: The

default extraction filter for the **`tarfile`** module is now set to **`'data'`**.

Core and Builtins

- gh-121860 [https://github.com/python/cpython/issues/121860]: Fix crash when rematerializing a managed dictionary after it was deleted.
- gh-121795 [https://github.com/python/cpython/issues/121795]: Improve performance of set membership testing, `set.remove()` and `set.discard()` when the argument is a set.
- gh-121814 [https://github.com/python/cpython/issues/121814]: Fixed the SegFault when **`PyEval_SetTrace()`** is used with no Python frame on stack.
- gh-121295 [https://github.com/python/cpython/issues/121295]: Fix PyREPL console getting into a blocked state after interrupting a long paste
- gh-121794 [https://github.com/python/cpython/issues/121794]: Fix bug in free-threaded Python where a resurrected object could lead to a negative ref count assertion failure.
- gh-121657 [https://github.com/python/cpython/issues/121657]: Improve the **`SyntaxError`** message if the user tries to use **`yield from`** outside a function.
- gh-121609 [https://github.com/python/cpython/issues/121609]: Fix pasting of characters containing unicode character joiners in the new REPL. Patch by Marta Gomez Macias
- gh-121297 [https://github.com/python/cpython/issues/121297]: Previously, incorrect usage of **`await`** or asynchronous comprehensions in code removed by the **`-O`** option was not flagged by the Python compiler. Now, such code raises **`SyntaxError`**. Patch by Jelle Zijlstra.
- gh-117482 [https://github.com/python/cpython/issues/117482]: Unexpected slot wrappers are no longer created for builtin static types in subinterpreters.
- gh-121562 [https://github.com/python/cpython/issues/121562]: Optimized performance of hex_from_char by replacing switch-case with a lookup table
- gh-121499 [https://github.com/python/cpython/issues/121499]: Fix a bug affecting how multi-line history was being rendered in

the new REPL after interacting with the new screen cache. Patch by Pablo Galindo

- gh-121497 [https://github.com/python/cpython/issues/121497]: Fix a bug that was preventing the REPL to correctly respect the history when an input hook was set. Patch by Pablo Galindo
- gh-121012 [https://github.com/python/cpython/issues/121012]: Tier 2 execution now ensures that list iterators remain exhausted, once they become exhausted.
- gh-121439 [https://github.com/python/cpython/issues/121439]: Allow tuples of length 20 in the freelist to be reused.
- gh-121288 [https://github.com/python/cpython/issues/121288]: **`ValueError`** messages for **`list.index()`**, **`range.index()`**, **`deque.index()`**, **`deque.remove()`** and **`ShareableList.index()`** no longer contain the repr of the searched value (which can be arbitrary large) and are consistent with error messages for other **`index()`** and **`remove()`** methods.
- gh-121368 [https://github.com/python/cpython/issues/121368]: Fix race condition in `_PyType_Lookup` in the free-threaded build due to a missing memory fence. This could lead to `_PyType_Lookup` returning incorrect results on arm64.
- gh-121149 [https://github.com/python/cpython/issues/121149]: Added specialization for summation of complexes, this also improves accuracy of builtin **`sum()`** for such inputs. Patch by Sergey B Kirpichev.
- gh-121130 [https://github.com/python/cpython/issues/121130]: Fix f-strings with debug expressions in format specifiers. Patch by Pablo Galindo

Library

- gh-121381 [https://github.com/python/cpython/issues/121381]: Remove `subprocess._USE_VFORK` escape hatch code and documentation. It was added just in case, and doesn’t have any known cases that require it.

Core and Builtins

- gh-119726 [https://github.com/python/cpython/issues/119726]: Optimize code layout for calls to C functions from the JIT on

AArch64. Patch by Diego Russo.

- gh-121115 [https://github.com/python/cpython/issues/121115]: **`PyLong_AsNativeBytes()`** no longer uses **`__index__()`** methods by default. The `Py_ASNATIVEBYTES_ALLOW_INDEX` flag has been added to allow it.
- gh-120838 [https://github.com/python/cpython/issues/120838]: **`Py_Finalize()`** and **`Py_FinalizeEx()`** now always run with the main interpreter active.
- gh-113433 [https://github.com/python/cpython/issues/113433]: Subinterpreters now get cleaned up automatically during runtime finalization.
- gh-119726 [https://github.com/python/cpython/issues/119726]: Improve the speed and memory use of C function calls from JIT code on AArch64. Patch by Diego Russo
- gh-116017 [https://github.com/python/cpython/issues/116017]: Simplify the warmup mechanism used for «side exits» in JIT code, resulting in slightly better performance and slightly lower memory usage for most platforms.
- gh-98442 [https://github.com/python/cpython/issues/98442]: Fix too wide source locations of the cleanup instructions of a with statement.
- gh-120754 [https://github.com/python/cpython/issues/120754]: Reduce the number of system calls invoked when reading a whole file (ex. `open('a.txt').read()`). For a sample program that reads the contents of the 400+ `.rst` files in the cpython repository `Doc` folder, there is an over 10% reduction in system call count.
- gh-119462 [https://github.com/python/cpython/issues/119462]: Make sure that invariants of type versioning are maintained: * Superclasses always have their version number assigned before subclasses * The version tag is always zero if the tag is not valid. * The version tag is always non-if the tag is valid.
- gh-120437 [https://github.com/python/cpython/issues/120437]: Fix `_CHECK_STACK_SPACE` optimization problems introduced in gh-118322 [https://github.com/python/cpython/issues/118322].
- gh-120722 [https://github.com/python/cpython/issues/120722]: Correctly set the bytecode position on return instructions within lambdas. Patch by Jelle Zijlstra.
- gh-120367 [https://github.com/python/cpython/issues/120367]: Fix

bug where compiler creates a redundant jump during pseudo-op replacement. Can only happen with a synthetic AST that has a try on the same line as the instruction following the exception handler.

- gh-120507 [https://github.com/python/cpython/issues/120507]: Remove the `BEFORE_WITH` and `BEFORE_ASYNC_WITH` instructions. Add the new **`LOAD_SPECIAL`** instruction. Generate code for `with` and `async with` statements using the new instruction.
- gh-113993 [https://github.com/python/cpython/issues/113993]: Strings interned with **`sys.intern()`** are again garbage-collected when no longer used, as per the documentation. Strings interned with the C function **`PyUnicode_InternInPlace()`** are still immortal. Internals of the string interning mechanism have been changed. This may affect performance and identities of **`str`** objects.

Library

- gh-120485 [https://github.com/python/cpython/issues/120485]: Add an override of `allow_reuse_port` on classes subclassing `socketserver.TCPServer` where `allow_reuse_address` is also overridden.

Core and Builtins

- gh-120384 [https://github.com/python/cpython/issues/120384]: Fix an array out of bounds crash in `list_ass_subscript`, which could be invoked via some specifically tailored input: including concurrent modification of a list object, where one thread assigns a slice and another clears it.
- gh-120367 [https://github.com/python/cpython/issues/120367]: Fix crash in compiler on code with redundant NOPs and JUMPs which show up after exception handlers are moved to the end of the code.

Library

- gh-120380 [https://github.com/python/cpython/issues/120380]: Fix Python implementation of **`pickle.Pickler`** for **`bytes`** and `bytearray` objects when using protocol version 5. Patch by Bénédikt Tran.

Core and Builtins

- gh-120397 [https://github.com/python/cpython/issues/120397]: Improve the throughput by up to two times for the `str.count()`, `bytes.count()` and `bytearray.count()` methods for counting single characters.
- gh-120221 [https://github.com/python/cpython/issues/120221]: Deliver real signals on Ctrl-C and Ctrl-Z in the new REPL. Patch by Pablo Galindo
- gh-120346 [https://github.com/python/cpython/issues/120346]: Respect `PYTHON_BASIC_REPL` when running in interactive inspect mode (`python -i`). Patch by Pablo Galindo
- gh-93691 [https://github.com/python/cpython/issues/93691]: Fix source locations of instructions generated for the iterator of a for statement.
- gh-120198 [https://github.com/python/cpython/issues/120198]: Fix a crash when multiple threads read and write to the same `__class__` of an object concurrently.
- gh-120298 [https://github.com/python/cpython/issues/120298]: Fix use-after free in `list_richcompare_impl` which can be invoked via some specifically tailored evil input.
- gh-119666 [https://github.com/python/cpython/issues/119666]: Fix a compiler crash in the case where two comprehensions in class scope both reference `__class__`.
- gh-119726 [https://github.com/python/cpython/issues/119726]: JIT: Re-use trampolines on AArch64 when creating stencils. Patch by Diego Russo
- gh-120225 [https://github.com/python/cpython/issues/120225]: Fix crash in compiler on empty block at end of exception handler.

- gh-93691 [https://github.com/python/cpython/issues/93691]: Fix source locations of instructions generated for with statements.

- gh-120097 [https://github.com/python/cpython/issues/120097]: `FrameLocalsProxy` now subclasses `collections.abc.Mapping` and can be matched as a mapping in `match` statements

- gh-120080 [https://github.com/python/cpython/issues/120080]: Direct call to the **`int.__round__()`** now accepts `None` as a valid argument.

- gh-119933 [https://github.com/python/cpython/issues/119933]: Improve **`SyntaxError`** messages for invalid expressions in a type parameters bound, a type parameter constraint tuple or a default type parameter. Patch by Bénédikt Tran.

- gh-119724 [https://github.com/python/cpython/issues/119724]: Reverted improvements to error messages for `elif/else` statements not matching any valid statements, which made in hard to locate the syntax errors inside those `elif/else` blocks.

- gh-119879 [https://github.com/python/cpython/issues/119879]: String search is now slightly faster for certain cases. It now utilizes last character gap (good suffix rule) for two-way periodic needles.

- gh-119842 [https://github.com/python/cpython/issues/119842]: Honor **`PyOS_InputHook()`** in the new REPL. Patch by Pablo Galindo

- gh-119180 [https://github.com/python/cpython/issues/119180]: **`classmethod()`** and **`staticmethod()`** now wrap the **`__annotations__`** and **`__annotate__`** attributes of their underlying callable lazily. See **PEP 649** [https://peps.python.org/pep-0649/]. Patch by Jelle Zijlstra.

- gh-119821 [https://github.com/python/cpython/issues/119821]: Fix execution of annotation scopes within classes when `globals` is set to a non-dict. Patch by Jelle Zijlstra.

- gh-118934 [https://github.com/python/cpython/issues/118934]: Make `PyEval_GetLocals` return borrowed reference
- gh-119740 [https://github.com/python/cpython/issues/119740]: Remove the previously-deprecated delegation of **`int()`** to **`__trunc__()`**.
- gh-119689 [https://github.com/python/cpython/issues/119689]: Generate stack effect metadata for pseudo instructions from bytecodes.c.
- gh-109218 [https://github.com/python/cpython/issues/109218]: **`complex()`** accepts now a string only as a positional argument. Passing a complex number as the «real» or «imag» argument is deprecated; it should only be passed as a single positional argument.
- gh-119548 [https://github.com/python/cpython/issues/119548]: Add a `clear` command to the REPL. Patch by Pablo Galindo
- gh-111999 [https://github.com/python/cpython/issues/111999]: Fix the signature of **`str.format_map()`**.
- gh-119560 [https://github.com/python/cpython/issues/119560]: An invalid assert in beta 1 has been removed. The assert would fail if `PyState_FindModule()` was used in an extension module's init function before the module def had been initialized.
- gh-119369 [https://github.com/python/cpython/issues/119369]: Fix deadlock during thread deletion in free-threaded build, which could occur when the GIL was enabled at runtime.
- gh-119525 [https://github.com/python/cpython/issues/119525]: Fix deadlock involving `_PyType_Lookup()` cache in the free-threaded build when the GIL is dynamically enabled at runtime.
- gh-119258 [https://github.com/python/cpython/issues/119258]: Eliminate type version guards in the tier two interpreter.

  Note that setting the `tp_version_tag` manually (which has never been supported) may result in crashes.

- gh-119311 [https://github.com/python/cpython/issues/119311]: Fix bug where names are unexpectedly mangled in the bases of generic classes.

- gh-119395 [https://github.com/python/cpython/issues/119395]: Fix bug where names appearing after a generic class are mangled as if they are in the generic class.

- gh-119372 [https://github.com/python/cpython/issues/119372]: Correct invalid corner cases in complex division (resulted in `(nan+nanj)` output), e.g. `1/complex('(inf+infj)')`. Patch by Sergey B Kirpichev.

- gh-119180 [https://github.com/python/cpython/issues/119180]: Evaluation of annotations is now deferred. See **PEP 649** [https://peps.python.org/pep-0649/] for details.

- gh-119180 [https://github.com/python/cpython/issues/119180]: Replace **`LOAD_ASSERTION_ERROR`** opcode with **`LOAD_COMMON_CONSTANT`** and add support for **`NotImplementedError`**.

- gh-119213 [https://github.com/python/cpython/issues/119213]: Non-builtin modules built with argument clinic were crashing if used in a subinterpreter before the main interpreter. The objects that were causing the problem by leaking between interpreters carelessly have been fixed.

- gh-119011 [https://github.com/python/cpython/issues/119011]: Fixes `type.__type_params__` to return an empty tuple instead of a descriptor.

- gh-118692 [https://github.com/python/cpython/issues/118692]: Avoid creating unnecessary **`StopIteration`** instances for monitoring.

- gh-119180 [https://github.com/python/cpython/issues/119180]: Add an `__annotate__` attribute to functions, classes, and modules as part of **PEP 649** [https://peps.python.org/pep-0649/].

Patch by Jelle Zijlstra.

- gh-119049 [https://github.com/python/cpython/issues/119049]: Fix displaying the source line for warnings created by the C API if the **`warnings`** module had not yet been imported.

- gh-119057 [https://github.com/python/cpython/issues/119057]: Improve **`ZeroDivisionError`** error message. Now, all error messages are harmonized: all `/`, `//`, and `%` operations just use «division by zero» message. And `0 ** -1` operation uses «zero to a negative power».

- gh-118844 [https://github.com/python/cpython/issues/118844]: Fix build failures when configuring with both `--disable-gil` and `--enable-experimental-jit`.

- gh-118921 [https://github.com/python/cpython/issues/118921]: Add `copy()` method for `FrameLocalsProxy` which returns a snapshot `dict` for local variables.

- gh-117657 [https://github.com/python/cpython/issues/117657]: Fix data races on the field that stores a pointer to the interpreter's main thread that occur in free-threaded builds.

- gh-118750 [https://github.com/python/cpython/issues/118750]: If the C version of the `decimal` module is available, `int(str)` now uses it to supply an asymptotically much faster conversion. However, this only applies if the string contains over about 2 million digits.

- gh-118767 [https://github.com/python/cpython/issues/118767]: Using **`NotImplemented`** in a boolean context now raises **`TypeError`**. Contributed by Jelle Zijlstra.

- gh-118561 [https://github.com/python/cpython/issues/118561]: Fix race condition in free-threaded build where **`list.extend()`** could expose uninitialised memory to concurrent readers.

- gh-117139 [https://github.com/python/cpython/issues/117139]: Convert the Python evaluation stack to use internal stack references. The purpose is to support tagged pointers. In **PEP 703** [https://peps.python.org/pep-0703/], this will allow for its form of deferred reference counting. For both the default and free-threaded builds, this sets up the infrastructure for unboxed integers in the future.

Library

- gh-115801 [https://github.com/python/cpython/issues/115801]: Raise `TypeError` when passing a string to **`difflib.unified_diff()`** and **`difflib.context_diff()`**.

Core and Builtins

- gh-117558 [https://github.com/python/cpython/issues/117558]: Improve error messages when a string, bytes or bytearray object of length 1 is expected.
- gh-117195 [https://github.com/python/cpython/issues/117195]: Avoid assertion failure for debug builds when calling `object.__sizeof__(1)`
- gh-116022 [https://github.com/python/cpython/issues/116022]: Improve the **`__repr__()`** output of **`AST`** nodes.
- gh-114091 [https://github.com/python/cpython/issues/114091]: Changed the error message for awaiting something that can't be awaited from «object <type> can't be used in an await expression» to «"<type>" object can't be awaited».
- gh-113190 [https://github.com/python/cpython/issues/113190]: **`Py_Finalize()`** now deletes all interned strings.
- gh-84978 [https://github.com/python/cpython/issues/84978]: Add class methods **`float.from_number()`** and **`complex.from_number()`**.
- gh-95144 [https://github.com/python/cpython/issues/95144]: Improve the error message from `a in b` when `b` is not a container to mention the term «container».
- bpo-24766 [https://bugs.python.org/issue?@action=redirect&bpo=24766]: Fix handling of `doc` argument to subclasses of `property`.

C API

- gh-123961 [https://github.com/python/cpython/issues/123961]: Convert the curses.window static type exposed by the **PyCursesWindow_Type** macro in `Include/py_curses.h` to a heap type. Patch by Bénédikt Tran.

- gh-124502 [https://github.com/python/cpython/issues/124502]: Add **`PyUnicode_Equal()`** function to the limited C API: test if two strings are equal. Patch by Victor Stinner.

- gh-124296 [https://github.com/python/cpython/issues/124296]: **`PyDictObject`** no longer maintains a private version tag field `ma_version_tag` per **PEP 699** [https://peps.python.org/pep-0699/]. This field was originally added in Python 3.6 (**PEP 509** [https://peps.python.org/pep-0509/]) and deprecated in Python 3.12.

- gh-124160 [https://github.com/python/cpython/issues/124160]: Fix crash when importing modules containing state and single-phase initialization in a subinterpreter.

- gh-124153 [https://github.com/python/cpython/issues/124153]: Add **`PyType_GetBaseByToken()`** and **`Py_tp_token`** slot for easier type checking, related to **PEP 489** [https://peps.python.org/pep-0489/] and **PEP 630** [https://peps.python.org/pep-0630/].

- gh-124127 [https://github.com/python/cpython/issues/124127]: In the limited C API 3.14 and newer, **`Py_REFCNT()`** is now implemented as an opaque function call to hide implementation details. Patch by Victor Stinner.

- gh-123880 [https://github.com/python/cpython/issues/123880]: Fixed a bug that prevented circular imports of extension modules that use single-phase initialization.

- gh-123909 [https://github.com/python/cpython/issues/123909]: **`PyType_FromSpec()`**, **`PyType_FromSpecWithBases()`** and **`PyType_FromModuleAndSpec()`** will now fail if the metaclass of the new type has custom **`tp_new`**.

- gh-107954 [https://github.com/python/cpython/issues/107954]: Add functions to configure the Python initialization (**PEP 741** [https://peps.python.org/pep-0741/]):

  - `PyInitConfig_Create()`
  - `PyInitConfig_Free()`
  - `PyInitConfig_GetError()`
  - `PyInitConfig_GetExitCode()`
  - `PyInitConfig_HasOption()`
  - `PyInitConfig_GetInt()`
  - `PyInitConfig_GetStr()`
  - `PyInitConfig_GetStrList()`
  - `PyInitConfig_FreeStrList()`
  - `PyInitConfig_SetInt()`
  - `PyInitConfig_SetStr()`
  - `PyInitConfig_SetStrList()`
  - `PyInitConfig_AddModule()`
  - `Py_InitializeFromInitConfig()`

  Patch by Victor Stinner.

- gh-107954 [https://github.com/python/cpython/issues/107954]: Add functions to get and set the current runtime Python configuration:

  - `PyConfig_Get()`
  - `PyConfig_GetInt()`
  - `PyConfig_Set()`
  - `PyConfig_Names()`

  Patch by Victor Stinner.

- gh-123465 [https://github.com/python/cpython/issues/123465]: `Py_RELATIVE_OFFSET` is now allowed in `PyMemberDef` for the special offset member `"__vectorcalloffset__"`, as well as the discouraged special offset members `"__dictoffset__"` and `"__weaklistoffset__"`

- gh-100554 [https://github.com/python/cpython/issues/100554]: Added a slot `Py_tp_vectorcall` to set `tp_vectorcall` via the `PyType_FromSpec()` function family. Limited API extensions can use this feature to provide more efficient vector call-based implementation of `__new__` and `__init__`.

- gh-122854 [https://github.com/python/cpython/issues/122854]: Add `Py_HashBuffer()` to compute and return the hash value of a buffer. Patch by Antoine Pitrou and Victor Stinner.
- gh-122728 [https://github.com/python/cpython/issues/122728]: Fix `PyEval_GetLocals()` to avoid `SystemError` («bad argument to internal function»). Patch by Victor Stinner.
- gh-116622 [https://github.com/python/cpython/issues/116622]: Make `PyObject_Print` work around a bug in Android and OpenBSD which prevented it from throwing an exception when trying to write to a read-only stream.
- gh-105201 [https://github.com/python/cpython/issues/105201]: Add `PyIter_NextItem()` to replace `PyIter_Next()`, which has an ambiguous return value. Patch by Irit Katriel and Erlend Aasland.
- gh-121489 [https://github.com/python/cpython/issues/121489]: Export private `_PyBytes_Join()` again.
- gh-121645 [https://github.com/python/cpython/issues/121645]: Add `PyBytes_Join(sep, iterable)` function, similar to `sep.join(iterable)` in Python. Patch by Victor Stinner.
- gh-89364 [https://github.com/python/cpython/issues/89364]: Export the `PySignal_SetWakeupFd()` function. Previously, the function was documented but it couldn't be used in 3rd party code. Patch by Victor Stinner.
- gh-113993 [https://github.com/python/cpython/issues/113993]: `PyUnicode_InternInPlace()` no longer prevents its argument from being garbage collected.

  Several functions that take `char *` are now documented as possibly preventing string objects from being garbage collected; refer to their documentation for details: `PyUnicode_InternFromString()`, `PyDict_SetItemString()`, `PyObject_SetAttrString()`, `PyObject_DelAttrString()`, **`PyUnicode_InternFromString()`**, and `PyModule_Add*` convenience functions.

- gh-113601 [https://github.com/python/cpython/issues/113601]: Removed debug build assertions related to interning strings, which were falsely triggered by stable ABI extensions.

- gh-112136 [https://github.com/python/cpython/issues/112136]: Restore the private `_PyArg_Parser` structure and the private `_PyArg_ParseTupleAndKeywordsFast()` function, previously removed in Python 3.13 alpha 1. Patch by Victor Stinner.

- gh-120642 [https://github.com/python/cpython/issues/120642]: Remove the private `_Py_CODEUNIT` type from the public C API. The internal `pycore_code.h` header should now be used to get this internal type. Patch by Victor Stinner.

- gh-120858 [https://github.com/python/cpython/issues/120858]: `PyDict_Next()` no longer locks the dictionary in the free-threaded build. The locking needs to be done by the caller around the entire iteration loop.

- gh-120642 [https://github.com/python/cpython/issues/120642]: Remove the following unstable functions:

    - `PyUnstable_Replace_Executor()`
    - `PyUnstable_SetOptimizer()`
    - `PyUnstable_GetOptimizer()`
    - `PyUnstable_GetExecutor()`
    - `PyUnstable_Optimizer_NewCounter()`
    - `PyUnstable_Optimizer_NewUOpOptimizer()`

  Patch by Victor Stinner.

- gh-120389 [https://github.com/python/cpython/issues/120389]: Add new functions to convert C `<stdint.h>` numbers from/to Python `int`:

  - `PyLong_FromInt32()`
  - `PyLong_FromUInt32()`
  - `PyLong_FromInt64()`
  - `PyLong_FromUInt64()`
  - `PyLong_AsInt32()`
  - `PyLong_AsUInt32()`
  - `PyLong_AsInt64()`
  - `PyLong_AsUInt64()`

  Patch by Victor Stinner.

- gh-120600 [https://github.com/python/cpython/issues/120600]: In the limited C API 3.14 and newer, `Py_TYPE()` is now implemented as an opaque function call to hide implementation details. Patch by Victor Stinner.

- gh-70278 [https://github.com/python/cpython/issues/70278]: `PyUnicode_FromFormat()` no longer produces the ending `\ufffd` character for truncated C string when use precision with `%s` and `%V`. It now truncates the string before the start of truncated multibyte sequences.

- gh-119771 [https://github.com/python/cpython/issues/119771]: Set `errno` in `_Py_c_pow()` on overflows. Patch by Sergey B Kirpichev.

- gh-119182 [https://github.com/python/cpython/issues/119182]: Add a new `PyUnicodeWriter` API to create a Python `str` object:

  - `PyUnicodeWriter_Create()`.
  - `PyUnicodeWriter_Discard()`.
  - `PyUnicodeWriter_Finish()`.
  - `PyUnicodeWriter_WriteChar()`.
  - `PyUnicodeWriter_WriteUTF8()`.
  - `PyUnicodeWriter_WriteUCS4()`.
  - `PyUnicodeWriter_WriteWideChar()`.
  - `PyUnicodeWriter_WriteStr()`.
  - `PyUnicodeWriter_WriteRepr()`.
  - `PyUnicodeWriter_WriteSubstring()`.
  - `PyUnicodeWriter_Format()`.
  - `PyUnicodeWriter_DecodeUTF8Stateful()`.

Patch by Victor Stinner.

- gh-119613 [https://github.com/python/cpython/issues/119613]: Soft deprecate the `Py_MEMCPY` macro: use directly `memcpy()` instead. Patch by Victor Stinner.

- gh-119775 [https://github.com/python/cpython/issues/119775]: Creating `immutable types` with mutable bases was deprecated since 3.12 and now raises a `TypeError`.

- gh-119585 [https://github.com/python/cpython/issues/119585]: Fix crash when a thread state that was created by `PyGILState_Ensure()` calls a destructor that during `PyThreadState_Clear()` that calls back into **`PyGILState_Ensure()`** and `PyGILState_Release()`. This might occur when in the free-threaded build or when using thread-local variables whose destructors call **`PyGILState_Ensure()`**.

- gh-119613 [https://github.com/python/cpython/issues/119613]: Macros `Py_IS_NAN`, `Py_IS_INFINITY` and `Py_IS_FINITE` are soft deprecated.

- gh-119336 [https://github.com/python/cpython/issues/119336]: Restore the removed `_PyLong_NumBits()` function. It is used by the pywin32 project. Patch by Ethan Smith

- gh-119344 [https://github.com/python/cpython/issues/119344]: The critical section API is now public as part of the non-limited C API.

- gh-119333 [https://github.com/python/cpython/issues/119333]: Add `PyContext_AddWatcher()` and `PyContext_ClearWatcher()` APIs to register callbacks to receive notification on enter and exit of context objects.

- gh-119247 [https://github.com/python/cpython/issues/119247]: Added `Py_BEGIN_CRITICAL_SECTION_SEQUENCE_FAST` and `Py_END_CRITICAL_SECTION_SEQUENCE_FAST` macros to make it possible to use PySequence_Fast APIs safely when free-threaded, and update str.join to work without the GIL using them.

- gh-111389 [https://github.com/python/cpython/issues/111389]: Add **`PyHASH_MULTIPLIER`** constant: prime multiplier used in string and various other hashes. Patch by Victor Stinner.

- gh-118771 [https://github.com/python/cpython/issues/118771]: Several C declarations with names that didn't start with the `Py` or `_Py` prefixes, which were added by mistake in 3.13 alpha and beta releases, were moved to internal headers.

- gh-116984 [https://github.com/python/cpython/issues/116984]: Make mimalloc includes relative to the current file to avoid embedders or extensions needing to include `Internal/mimalloc` if they are already including internal CPython headers.

- gh-118789 [https://github.com/python/cpython/issues/118789]: Add **`PyUnstable_Object_ClearWeakRefsNoCallbacks()`**, which clears weakrefs without calling their callbacks.

- gh-118789 [https://github.com/python/cpython/issues/118789]: Restore `_PyWeakref_ClearRef` that was previously removed in Python 3.13 alpha 1.

- gh-117511 [https://github.com/python/cpython/issues/117511]: Make the **`PyMutex`** public in the non-limited C API.

- gh-116560 [https://github.com/python/cpython/issues/116560]: Add **`PyLong_GetSign()`** function. Patch by Sergey B Kirpichev.

- gh-97588 [https://github.com/python/cpython/issues/97588]: Fix creating bitfields in **`ctypes`** structures and unions. Fields no longer overlap.

- gh-87135 [https://github.com/python/cpython/issues/87135]: Attempting to acquire the GIL after runtime finalization has begun in a different thread now causes the thread to hang rather than terminate, which avoids potential crashes or memory corruption caused by attempting to terminate a thread that is running code not specifically designed to support termination. In most cases this hanging is harmless since the process will soon exit anyway.

The `PyThread_exit_thread` function is now deprecated. Its behavior is inconsistent across platforms, and it can only be used safely in the unlikely case that every function in the entire call stack has been designed to support the platform-dependent termination mechanism. It is recommended that users of this function change their design to not require thread termination. In the unlikely case that thread termination is needed and can be done safely, users may migrate to calling platform-specific APIs such as `pthread_exit` (POSIX) or `_endthreadex` (Windows) directly.

- bpo-34008 [https://bugs.python.org/issue?@action=redirect&bpo=34008]: Added `Py_IsInitialized` to the list of APIs that are safe to call before the interpreter is initialized, and updated the embedding tests to cover it.

Build

- gh-125269 [https://github.com/python/cpython/issues/125269]: Fix detection of whether `-latomic` is needed when cross-compiling CPython using the configure script.
- gh-123990 [https://github.com/python/cpython/issues/123990]: Remove `WITH_FREELISTS` macro and `--without-freelists` build configuration
- gh-124102 [https://github.com/python/cpython/issues/124102]: Update internal documentation under PCbuild, so it now correctly states that Windows requires VS2017 or later and Python 3.10 or later
- gh-124043 [https://github.com/python/cpython/issues/124043]: Building using **`--with-trace-refs`** is (temporarily) disallowed when the GIL is disabled.
- gh-123418 [https://github.com/python/cpython/issues/123418]: Updated Android build to use OpenSSL 3.0.15.
- gh-123297 [https://github.com/python/cpython/issues/123297]: Propagate the value of `LDFLAGS` to `LDCXXSHARED` in **`sysconfig`**. Patch by Pablo Galindo

- gh-121634 [https://github.com/python/cpython/issues/121634]: Allow for specifying the target compile triple for WASI.
- gh-122578 [https://github.com/python/cpython/issues/122578]: Use WASI SDK 24 for testing.
- gh-116622 [https://github.com/python/cpython/issues/116622]: Rename build variable `MODULE_LDFLAGS` back to `LIBPYTHON`, as it’s used by package build systems (e.g. Meson).
- gh-118943 [https://github.com/python/cpython/issues/118943]: Fix an issue where the experimental JIT could be built several times by the `make regen-all` target, leading to possible race conditions on heavily parallelized builds.
- gh-121996 [https://github.com/python/cpython/issues/121996]: Introduce ./configure –disable-safety and –enable-slower-safety options. Patch by Donghee Na.
- gh-120522 [https://github.com/python/cpython/issues/120522]: Added a **`--with-app-store-compliance`** option to patch out known issues with macOS/iOS App Store review processes.
- gh-120371 [https://github.com/python/cpython/issues/120371]: Support WASI SDK 22 by explicitly skipping functions that are just stubs in wasi-libc.
- gh-121731 [https://github.com/python/cpython/issues/121731]: Fix mimalloc compile error on GNU/Hurd
- gh-121487 [https://github.com/python/cpython/issues/121487]: Fix deprecation warning for ATOMIC_VAR_INIT in mimalloc.
- gh-121467 [https://github.com/python/cpython/issues/121467]: Fix a Makefile bug that prevented mimalloc header files from being installed.
- gh-121103 [https://github.com/python/cpython/issues/121103]: On POSIX systems, excluding macOS framework installs, the lib directory for the free-threaded build now includes a «t» suffix to avoid conflicts with a co-located default build installation.
- gh-120831 [https://github.com/python/cpython/issues/120831]: The default minimum iOS version was increased to 13.0.
- gh-121082 [https://github.com/python/cpython/issues/121082]: Fix build failure when the developer use `--enable-pystats` arguments in configuration command after #118450.
- gh-120671 [https://github.com/python/cpython/issues/120671]: Fix failing configure tests due to a missing space when appending to CFLAGS.

- gh-120602 [https://github.com/python/cpython/issues/120602]: Correctly handle LLVM installs with `LLVM_VERSION_SUFFIX` when building with `--enable-experimental-jit`.
- gh-120688 [https://github.com/python/cpython/issues/120688]: On WASI in debug mode, Python is now built with compiler flag `-O3` instead of `-Og`, to support more recursive calls. Patch by Victor Stinner.
- gh-118943 [https://github.com/python/cpython/issues/118943]: Fix a possible race condition affecting parallel builds configured with `--enable-experimental-jit`, in which **`FileNotFoundError`** could be caused by another process already moving `jit_stencils.h.new` to `jit_stencils.h`.
- gh-120326 [https://github.com/python/cpython/issues/120326]: On Windows, fix build error when `--disable-gil` and `--experimental-jit` options are combined.
- gh-120291 [https://github.com/python/cpython/issues/120291]: Make the `python-config` shell script compatible with non-bash shells.
- gh-113565 [https://github.com/python/cpython/issues/113565]: Improve **`curses`** and **`curses.panel`** dependency checks in **configure**.
- gh-119729 [https://github.com/python/cpython/issues/119729]: On POSIX systems, the pkg-config (`.pc`) filenames now include the ABI flags, which may include debug («d») and free-threaded («t»). For example: * `python-3.14.pc` (default, non-debug build) * `python-3.14d.pc` (default, debug build) * `python-3.14t.pc` (free-threaded build)
- gh-119400 [https://github.com/python/cpython/issues/119400]: `make_ssl_certs`, the script that prepares certificate data for the test suite, now allows specifying expiration dates.
- gh-115119 [https://github.com/python/cpython/issues/115119]: Fall back to the bundled libmpdec if a system version cannot be found.
- gh-119132 [https://github.com/python/cpython/issues/119132]: Update **`sys.version`** to identify whether the build is default build or free-threading build. Patch By Donghee Na.
- gh-118836 [https://github.com/python/cpython/issues/118836]: Fix an `AssertionError` when building with `--enable-`

`experimental-jit` and the compiler emits a `SHT_NOTE` section.
- gh-118943 [https://github.com/python/cpython/issues/118943]: Fix a possible race condition affecting parallel builds configured with `--enable-experimental-jit`, in which compilation errors could be caused by an incompletely-generated header file.

Python 3.13.0 beta 1

Release date: 2024-05-08

Security

- gh-116741 [https://github.com/python/cpython/issues/116741]: Update bundled libexpat to 2.6.2
- gh-117233 [https://github.com/python/cpython/issues/117233]: Detect BLAKE2, SHA3, Shake, & truncated SHA512 support in the OpenSSL-ish libcrypto library at build time. This allows **`hashlib`** to be used with libraries that do not to support every algorithm that upstream OpenSSL does.

Core and Builtins

- gh-118414 [https://github.com/python/cpython/issues/118414]: Add instrumented opcodes to YIELD_VALUE assertion for tracing cases.
- gh-117953 [https://github.com/python/cpython/issues/117953]: When a builtin or extension module is imported for the first time, while a subinterpreter is active, the module's init function is now run by the main interpreter first before import continues in the subinterpreter. Consequently, single-phase init modules now fail in an isolated subinterpreter without the init function running under that interpreter, whereas before it would run under the subinterpreter *before* failing, potentially leaving behind global state and callbacks and otherwise leaving the module in an inconsistent state.
- gh-117549 [https://github.com/python/cpython/issues/117549]: Don't use designated initializer syntax in inline functions in internal

headers. They cause problems for C++ or MSVC users who aren't yet using the latest C++ standard (C++20). While internal, pycore_backoff.h, is included (indirectly, via pycore_code.h) by some key 3rd party software that does so for speed.

Library

- gh-95382 [https://github.com/python/cpython/issues/95382]: Improve performance of `json.dumps()` and `json.dump()` when using the argument *indent*. Depending on the data the encoding using `json.dumps()` with *indent* can be up to 2 to 3 times faster.

Core and Builtins

- gh-116322 [https://github.com/python/cpython/issues/116322]: In `--disable-gil` builds, the GIL will be enabled while loading C extension modules. If the module indicates that it supports running without the GIL, the GIL will be disabled once loading is complete. Otherwise, the GIL will remain enabled for the remainder of the interpreter's lifetime. This behavior does not apply if the GIL has been explicitly enabled or disabled with `PYTHON_GIL` or `-Xgil`.
- gh-118513 [https://github.com/python/cpython/issues/118513]: Fix incorrect `UnboundLocalError` when two comprehensions in the same function both reference the same name, and in one comprehension the name is bound while in the other it's an implicit global.
- gh-118518 [https://github.com/python/cpython/issues/118518]: Allow the Linux perf support to work without frame pointers using perf's advanced JIT support. The feature is activated when using the `PYTHON_PERF_JIT_SUPPORT` environment variable or when running Python with `-Xperf_jit`. Patch by Pablo Galindo.
- gh-117514 [https://github.com/python/cpython/issues/117514]: Add `sys._is_gil_enabled()` function that returns whether

the GIL is currently enabled. In the default build it always returns `True` because the GIL is always enabled. In the free-threaded build, it may return `True` or `False`.

- gh-118164 [https://github.com/python/cpython/issues/118164]: Break a loop between the Python implementation of the **`decimal`** module and the Python code for integer to string conversion. Also optimize integer to string conversion for values in the range from 9_000 to 135_000 decimal digits.

- gh-118473 [https://github.com/python/cpython/issues/118473]: Fix **`sys.set_asyncgen_hooks()`** not to be partially set when raising **`TypeError`**.

- gh-118465 [https://github.com/python/cpython/issues/118465]: Compiler populates the new `__firstlineno__` field on a class with the line number of the first line of the class definition.

- gh-118492 [https://github.com/python/cpython/issues/118492]: Fix an issue where the type cache can expose a previously accessed attribute when a finalizer is run.

- gh-117714 [https://github.com/python/cpython/issues/117714]: update `async_generator.athrow().close()` and `async_generator.asend().close()` to close their section of the underlying async generator

- gh-111201 [https://github.com/python/cpython/issues/111201]: The interactive interpreter is now implemented in Python, which allows for a number of new features like colors, multiline input, history viewing, and paste mode. Contributed by Pablo Galindo, Łukasz Langa and Lysandros Nikolaou based on code from the PyPy project.

- gh-74929 [https://github.com/python/cpython/issues/74929]: Implement PEP 667: converted **`FrameType.f_locals`** and **`PyFrame_GetLocals()`** to return a write-through proxy object when the frame refers to a function or comprehension.

- gh-116767 [https://github.com/python/cpython/issues/116767]: Fix

crash in compiler on “async with” that has many context managers.

- gh-118335 [https://github.com/python/cpython/issues/118335]: Change how to use the tier 2 interpreter. Instead of running Python with `-X uops` or setting the environment variable `PYTHON_UOPS=1`, this choice is now made at build time by configuring with `--enable-experimental-jit=interpreter`.

  Beware! This changes the environment variable to enable or disable micro-ops to `PYTHON_JIT`. The old `PYTHON_UOPS` is no longer used.

- gh-118306 [https://github.com/python/cpython/issues/118306]: Update JIT compilation to use LLVM 18

- gh-118160 [https://github.com/python/cpython/issues/118160]: Annotation scopes within classes can now contain comprehensions. However, such comprehensions are not inlined into their parent scope at runtime. Patch by Jelle Zijlstra.

- gh-118272 [https://github.com/python/cpython/issues/118272]: Fix bug where `generator.close` does not free the generator frame’s locals.

- gh-118216 [https://github.com/python/cpython/issues/118216]: Don’t consider **`__future__`** imports with dots before the module name.

- gh-118074 [https://github.com/python/cpython/issues/118074]: Make sure that the Executor objects in the COLD_EXITS array aren’t assumed to be GC-able (which would access bytes outside the object).

- gh-107674 [https://github.com/python/cpython/issues/107674]: Lazy load frame line number to improve performance of tracing

- gh-118082 [https://github.com/python/cpython/issues/118082]: Improve **`SyntaxError`** message for imports without names, like in `from x import` and `import` cases. It now points out to users that **`import`** expects at least one name after it.

- gh-118090 [https://github.com/python/cpython/issues/118090]: Improve **`SyntaxError`** message for empty type param brackets.

- gh-117958 [https://github.com/python/cpython/issues/117958]: Added a `get_jit_code()` method to access JIT compiled machine code from the UOp Executor when the experimental JIT is enabled. Patch by Anthony Shaw.

- gh-117901 [https://github.com/python/cpython/issues/117901]: Add option for compiler's codegen to save nested instruction sequences for introspection.

- gh-116622 [https://github.com/python/cpython/issues/116622]: Redirect stdout and stderr to system log when embedded in an Android app.

- gh-109118 [https://github.com/python/cpython/issues/109118]: annotation scope within class scopes can now contain lambdas.

- gh-117894 [https://github.com/python/cpython/issues/117894]: Prevent `agen.aclose()` objects being re-used after `.throw()`.

- gh-117881 [https://github.com/python/cpython/issues/117881]: prevent concurrent access to an async generator via athrow().throw() or asend().throw()

- gh-117536 [https://github.com/python/cpython/issues/117536]: Fix a **`RuntimeWarning`** when calling `agen.aclose().throw(Exception)`.

- gh-117755 [https://github.com/python/cpython/issues/117755]: Fix mimalloc allocator for huge memory allocation (around 8,589,934,592 GiB) on s390x. Patch by Victor Stinner.

- gh-117750 [https://github.com/python/cpython/issues/117750]: Fix issue where an object's dict would get out of sync with the object's internal values when being cleared. `obj.__dict__.clear()` now clears the internal values, but leaves the dict attached to the object.

- gh-117431 [https://github.com/python/cpython/issues/117431]: Improve the performance of the following **`bytes`** and **`bytearray`** methods by adapting them to the **`METH_FASTCALL`** calling convention:

    - **`count()`**
    - **`find()`**
    - **`index()`**
    - **`rfind()`**
    - **`rindex()`**

- gh-117709 [https://github.com/python/cpython/issues/117709]: Speed up calls to **`str()`** with positional-only argument, by using the **PEP 590** [https://peps.python.org/pep-0590/] `vectorcall` calling convention. Patch by Erlend Aasland.

- gh-117680 [https://github.com/python/cpython/issues/117680]: Give `_PyInstructionSequence` a Python interface and use it in tests.

- gh-115776 [https://github.com/python/cpython/issues/115776]: Statically allocated objects are, by definition, immortal so must be marked as such regardless of whether they are in extension modules or not.

- gh-117385 [https://github.com/python/cpython/issues/117385]: Remove unhandled `PY_MONITORING_EVENT_BRANCH` and `PY_MONITORING_EVENT_EXCEPTION_HANDLED` events from **`sys.settrace()`**.

- gh-116322 [https://github.com/python/cpython/issues/116322]: Extension modules may indicate to the runtime that they can run without the GIL. Multi-phase init modules do so by calling providing `Py_MOD_GIL_NOT_USED` for the `Py_mod_gil` slot, while single-phase init modules call `PyUnstable_Module_SetGIL(mod, Py_MOD_GIL_NOT_USED)` from their init function.

- gh-116129 [https://github.com/python/cpython/issues/116129]: Implement **PEP 696** [https://peps.python.org/pep-0696/], adding support for defaults on type parameters. Patch by Jelle Zijlstra.

- gh-93502 [https://github.com/python/cpython/issues/93502]: Add two new functions to the C-API, `PyRefTracer_SetTracer()` and `PyRefTracer_GetTracer()`, that allows to track object creation and destruction the same way the `tracemalloc` module does. Patch by Pablo Galindo

- gh-107674 [https://github.com/python/cpython/issues/107674]: Improved the performance of `sys.settrace()` significantly

- gh-95754 [https://github.com/python/cpython/issues/95754]: Improve the error message when a script shadowing a module from the standard library causes `AttributeError` to be raised. Similarly, improve the error message when a script shadowing a third party module attempts to access an attribute from that third party module while still initialising.

- gh-99180 [https://github.com/python/cpython/issues/99180]: Elide uninformative traceback indicators in `return` and simple `assignment` statements. Patch by Pablo Galindo.

- gh-105879 [https://github.com/python/cpython/issues/105879]: Allow the *globals* and *locals* arguments to `exec()` and `eval()` to be passed as keywords.

Library

- gh-118418 [https://github.com/python/cpython/issues/118418]: A `DeprecationWarning` is now emitted if you fail to pass a value to the new *type_params* parameter of `typing._eval_type()` or `typing.ForwardRef._evaluate()`. (Using either of these private and undocumented functions is discouraged to begin with, but failing to pass a value to the `type_params` parameter may lead to incorrect behaviour on Python 3.12 or newer.)

- gh-118660 [https://github.com/python/cpython/issues/118660]: Add an optional second type parameter to **`typing.ContextManager`** and **`typing.AsyncContextManager`**, representing the return types of **`__exit__()`** and **`__aexit__()`** respectively. This parameter defaults to `bool | None`.

- gh-118650 [https://github.com/python/cpython/issues/118650]: The `enum` module allows method named `_repr_*` to be defined on `Enum` types.

- gh-118648 [https://github.com/python/cpython/issues/118648]: Add type parameter defaults to **`typing.Generator`** and **`typing.AsyncGenerator`**.

- gh-101137 [https://github.com/python/cpython/issues/101137]: Mime type `text/x-rst` is now supported by **`mimetypes`**.

- gh-118164 [https://github.com/python/cpython/issues/118164]: The Python implementation of the `decimal` module could appear to hang in relatively small power cases (like `2**117`) if context precision was set to a very high value. A different method to check for exactly representable results is used now that doesn't rely on computing `10**precision` (which could be effectively too large to compute).

- gh-111744 [https://github.com/python/cpython/issues/111744]: `breakpoint()` and `pdb.set_trace()` now enter the debugger immediately after the call rather than before the next line is executed.

- gh-118500 [https://github.com/python/cpython/issues/118500]: Add **`pdb`** support for zipapps

- gh-118406 [https://github.com/python/cpython/issues/118406]: Add signature for **`sqlite3.Connection`** objects.

- gh-101732 [https://github.com/python/cpython/issues/101732]: Use a Y2038 compatible openssl time function when available.

- gh-118404 [https://github.com/python/cpython/issues/118404]: Fix

`inspect.signature()` for non-comparable callables.

- gh-118402 [https://github.com/python/cpython/issues/118402]: Fix `inspect.signature()` for the result of the `functools.cmp_to_key()` call.
- gh-116622 [https://github.com/python/cpython/issues/116622]: On Android, `sysconfig.get_platform` now returns the format specified by PEP 738 [https://peps.python.org/pep-0738/].
- gh-118285 [https://github.com/python/cpython/issues/118285]: Allow to specify the signature of custom callable instances of extension type by the `__text_signature__` attribute. Specify signatures of `operator.attrgetter`, `operator.itemgetter`, and `operator.methodcaller` instances.
- gh-118314 [https://github.com/python/cpython/issues/118314]: Fix an edge case in `binascii.a2b_base64()` strict mode, where excessive padding is not detected when no padding is necessary.
- gh-118271 [https://github.com/python/cpython/issues/118271]: Add the `PhotoImage` methods `read()` to read an image from a file and `data()` to get the image data. Add *background* and *grayscale* parameters to `PhotoImage` method `write()`.
- gh-118225 [https://github.com/python/cpython/issues/118225]: Add the `PhotoImage` method `copy_replace()` to copy a region from one image to other image, possibly with pixel zooming and/or subsampling. Add *from_coords* parameter to `PhotoImage` methods `copy()`, `zoom()` and `subsample()`. Add *zoom* and *subsample* parameters to `PhotoImage` method `copy()`.
- gh-118221 [https://github.com/python/cpython/issues/118221]: Fix a bug where `sqlite3.Connection.iterdump()` could fail if a custom `row factory` was used. Patch by Erlend Aasland.
- gh-118013 [https://github.com/python/cpython/issues/118013]: Fix

regression introduced in gh-103193 [https://github.com/python/cpython/issues/103193] that meant that calling `inspect.getattr_static()` on an instance would cause a strong reference to that instance's class to persist in an internal cache in the `inspect` module. This caused unexpected memory consumption if the class was dynamically created, the class held strong references to other objects which took up a significant amount of memory, and the cache contained the sole strong reference to the class. The fix for the regression leads to a slowdown in `getattr_static()`, but the function should still be significantly faster than it was in Python 3.11. Patch by Alex Waygood.

- gh-118218 [https://github.com/python/cpython/issues/118218]: Speed up `itertools.pairwise()` in the common case by up to 1.8x.

- gh-117486 [https://github.com/python/cpython/issues/117486]: Improve the behavior of user-defined subclasses of `ast.AST`. Such classes will now require no changes in the usual case to conform with the behavior changes of the `ast` module in Python 3.13. Patch by Jelle Zijlstra.

- gh-90848 [https://github.com/python/cpython/issues/90848]: Fixed `unittest.mock.create_autospec()` to configure parent mock with keyword arguments.

- gh-118168 [https://github.com/python/cpython/issues/118168]: Fix incorrect argument substitution when `typing.Unpack` is used with the builtin `tuple`. `typing.Unpack` now raises `TypeError` when used with certain invalid types. Patch by Jelle Zijlstra.

- gh-118131 [https://github.com/python/cpython/issues/118131]: Add command-line interface for the `random` module. Patch by Hugo van Kemenade.

- gh-118107 [https://github.com/python/cpython/issues/118107]: Fix `zipimport` reading of ZIP64 files with file entries that are too big or offset too far.

- gh-102511 [https://github.com/python/cpython/issues/102511]: Fix **`os.path.normpath()`** for UNC paths on Windows. Speed up **`os.path.splitroot()`** with a native implementation.

- gh-117535 [https://github.com/python/cpython/issues/117535]: Change the unknown filename of **`warnings`** from `sys` to `<sys>` to clarify that it's not a real filename.

- gh-114053 [https://github.com/python/cpython/issues/114053]: Fix erroneous **`NameError`** when calling **`typing.get_type_hints()`** on a class that made use of **PEP 695** [https://peps.python.org/pep-0695/] type parameters in a module that had `from __future__ import annotations` at the top of the file. Patch by Alex Waygood.

- gh-116931 [https://github.com/python/cpython/issues/116931]: Add parameter *fileobj* check for **`tarfile.TarFile.addfile()`**

- gh-117995 [https://github.com/python/cpython/issues/117995]: Don't raise **`DeprecationWarning`** when a sequence of parameters is used to bind indexed, nameless placeholders. See also gh-100668 [https://github.com/python/cpython/issues/100668].

- gh-80361 [https://github.com/python/cpython/issues/80361]: Fix TypeError in **`email.message.Message.get_payload()`** when the charset is **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html] encoded.

- gh-86650 [https://github.com/python/cpython/issues/86650]: Fix IndexError when parse some emails with invalid Message-ID (including one-off addresses generated by Microsoft Outlook).

- gh-117691 [https://github.com/python/cpython/issues/117691]: Improve the error messages emitted by **`tarfile`** deprecation warnings relating to PEP 706. If a `filter` argument is not provided to `extract()` or `extractall`, the deprecation warning now points to the line in the user's code where the relevant function was called. Patch by Alex Waygood.

- gh-115874 [https://github.com/python/cpython/issues/115874]: Fixed

a possible segfault during garbage collection of `_asyncio.FutureIter` objects. Patch by Savannah Ostrowski.

- gh-115060 [https://github.com/python/cpython/issues/115060]: Speed up **`pathlib.Path.glob()`** by omitting an initial **`is_dir()`** call. As a result of this change, **`glob()`** can no longer raise **`OSError`**.

- gh-77102 [https://github.com/python/cpython/issues/77102]: **`site`** module now parses `.pth` file with UTF-8 first, and locale encoding if `UnicodeDecodeError` happened. It supported only locale encoding before.

- gh-76785 [https://github.com/python/cpython/issues/76785]: We've exposed the low-level **`_interpreters`** module for the sake of the PyPI implementation of **PEP 734** [https://peps.python.org/pep-0734/]. It was sometimes available as the **`_xxsubinterpreters`** module and was formerly used only for testing. For the most part, it should be considered an internal module, like **`_thread`** and **`_imp`**. See https://discuss.python.org/t/pep-734-multiple-interpreters-in-the-stdlib/41147/26.

- gh-115060 [https://github.com/python/cpython/issues/115060]: Speed up **`pathlib.Path.glob()`** by not scanning directories for non-wildcard pattern segments.

- gh-117727 [https://github.com/python/cpython/issues/117727]: Speed up **`pathlib.Path.iterdir()`** by using **`os.scandir()`** internally.

- gh-117586 [https://github.com/python/cpython/issues/117586]: Speed up **`pathlib.Path.walk()`** by working with strings internally.

- gh-117722 [https://github.com/python/cpython/issues/117722]: Change the new multi-separator support in **`asyncio.StreamReader.readuntil()`** to only accept tuples of separators rather than arbitrary iterables.

- gh-117692 [https://github.com/python/cpython/issues/117692]: Fixes a bug when `doctest.DocTestFinder` was failing on wrapped `builtin_function_or_method`.
- gh-117348 [https://github.com/python/cpython/issues/117348]: Largely restored import time performance of configparser by avoiding dataclasses.
- gh-117641 [https://github.com/python/cpython/issues/117641]: Speedup `os.path.commonpath()` on Unix.
- gh-117663 [https://github.com/python/cpython/issues/117663]: Fix `_simple_enum` to detect aliases when multiple arguments are present but only one is the member value.
- gh-117636 [https://github.com/python/cpython/issues/117636]: Speedup `os.path.join()`.
- gh-117618 [https://github.com/python/cpython/issues/117618]: Support `package.module` as `filename` for `break` command of `pdb`
- gh-102247 [https://github.com/python/cpython/issues/102247]: the status codes enum with constants in http.HTTPStatus are updated to include the names from RFC9110. This RFC includes some HTTP statuses previously only used for WEBDAV and assigns more generic names to them.

  The old constants are preserved for backwards compatibility.
- gh-117607 [https://github.com/python/cpython/issues/117607]: Speedup `os.path.relpath()`.
- gh-117586 [https://github.com/python/cpython/issues/117586]: Speed up `pathlib.Path.glob()` by working with strings internally.
- gh-117225 [https://github.com/python/cpython/issues/117225]: Add colour to doctest output. Patch by Hugo van Kemenade.
- gh-117566 [https://github.com/python/cpython/issues/117566]: `ipaddress.IPv6Address.is_loopback()` will now

return `True` for IPv4-mapped loopback addresses, i.e. addresses in the `::ffff:127.0.0.0/104` address space.

- gh-117546 [https://github.com/python/cpython/issues/117546]: Fix issue where **`os.path.realpath()`** stopped resolving symlinks after encountering a symlink loop on POSIX.
- gh-116720 [https://github.com/python/cpython/issues/116720]: Improved behavior of **`asyncio.TaskGroup`** when an external cancellation collides with an internal cancellation. For example, when two task groups are nested and both experience an exception in a child task simultaneously, it was possible that the outer task group would misbehave, because its internal cancellation was swallowed by the inner task group.

  In the case where a task group is cancelled externally and also must raise an **`ExceptionGroup`**, it will now call the parent task's **`cancel()`** method. This ensures that a **`asyncio.CancelledError`** will be raised at the next **`await`**, so the cancellation is not lost.

  An added benefit of these changes is that task groups now preserve the cancellation count (**`asyncio.Task.cancelling()`**).

  In order to handle some corner cases, **`asyncio.Task.uncancel()`** may now reset the undocumented `_must_cancel` flag when the cancellation count reaches zero.
- gh-117516 [https://github.com/python/cpython/issues/117516]: Add **`typing.TypeIs`**, implementing **PEP 742** [https://peps.python.org/pep-0742/]. Patch by Jelle Zijlstra.
- gh-117503 [https://github.com/python/cpython/issues/117503]: Fix support of non-ASCII user names in bytes paths in **`os.path.expanduser()`** on Posix.
- gh-117394 [https://github.com/python/cpython/issues/117394]: **`os.path.ismount()`** is now 2-3 times faster if the user has

permissions.

- gh-117313 [https://github.com/python/cpython/issues/117313]: Only treat `'\n'`, `'\r'` and `'\r\n'` as line separators in re-folding the **`email`** messages. Preserve control characters `'\v'`, `'\f'`, `'\x1c'`, `'\x1d'` and `'\x1e'` and Unicode line separators `'\x85'`, `'\u2028'` and `'\u2029'` as is.

- gh-117142 [https://github.com/python/cpython/issues/117142]: Convert **`_ctypes`** to multi-phase initialisation (**PEP 489** [https://peps.python.org/pep-0489/]).

- gh-66543 [https://github.com/python/cpython/issues/66543]: Add the **`mimetypes.guess_file_type()`** function which works with file path. Passing file path instead of URL in **`guess_type()`** is soft deprecated.

- gh-68583 [https://github.com/python/cpython/issues/68583]: webbrowser CLI: replace getopt with argparse, add long options. Patch by Hugo van Kemenade.

- gh-116871 [https://github.com/python/cpython/issues/116871]: Name suggestions for **`AttributeError`** and **`ImportError`** now only include underscored names if the original name was underscored.

- gh-116023 [https://github.com/python/cpython/issues/116023]: Don’t show empty fields (value `None` or `[]`) in **`ast.dump()`** by default. Add `show_empty=False` parameter to optionally show them.

- gh-115961 [https://github.com/python/cpython/issues/115961]: Added **`name`** and **`mode`** attributes for compressed and archived file-like objects in modules **`bz2`**, **`lzma`**, **`tarfile`** and **`zipfile`**. The value of the **`mode`** attribute of **`gzip.GzipFile`** was changed from integer (`1` or `2`) to string (`'rb'` or `'wb'`). The value of the **`mode`** attribute of the readable file-like object returned by **`zipfile.ZipFile.open()`** was changed from `'r'` to `'rb'`.

- gh-82062 [https://github.com/python/cpython/issues/82062]: Fix **`inspect.signature()`** to correctly handle parameter defaults on methods in extension modules that use names defined in the module namespace.

- gh-83856 [https://github.com/python/cpython/issues/83856]: Honor **`atexit`** for all **`multiprocessing`** start methods

- gh-113081 [https://github.com/python/cpython/issues/113081]: Print colorized exception just like built-in traceback in **`pdb`**

- gh-112855 [https://github.com/python/cpython/issues/112855]: Speed up pickling of **`pathlib.PurePath`** objects. Patch by Barney Gale.

- gh-111744 [https://github.com/python/cpython/issues/111744]: Support opcode events in **`bdb`**

- gh-109617 [https://github.com/python/cpython/issues/109617]: **`ncurses`**: fixed a crash that could occur on macOS 13 or earlier when Python was built with Apple Xcode 15’s SDK.

- gh-83151 [https://github.com/python/cpython/issues/83151]: Enabled arbitrary statements and evaluations in **`pdb`** shell to access the local variables of the current frame, which made it possible for multi-scope code like generators or nested function to work.

- gh-110209 [https://github.com/python/cpython/issues/110209]: Add **`__class_getitem__()`** to **`types.GeneratorType`** and **`types.CoroutineType`** for type hinting purposes. Patch by James Hilton-Balfe.

- gh-108191 [https://github.com/python/cpython/issues/108191]: The **`types.SimpleNamespace`** now accepts an optional positional argument which specifies initial values of attributes as a dict or an iterable of key-value pairs.

- gh-62090 [https://github.com/python/cpython/issues/62090]: Fix assertion errors caused by whitespace in metavars or `SUPPRESS`-ed groups in **`argparse`** by simplifying usage formatting. Patch by Ali Hamdan.

- gh-102402 [https://github.com/python/cpython/issues/102402]: Adjust `logging.LogRecord` to use `time.time_ns()` and fix minor bug related to floating-point math.

- gh-100242 [https://github.com/python/cpython/issues/100242]: Bring pure Python implementation `functools.partial.__new__` more in line with the C-implementation by not just always checking for the presence of the attribute `'func'` on the first argument of `partial`. Instead, both the Python version and the C version perform an `isinstance(func, partial)` check on the first argument of `partial`.

- gh-99730 [https://github.com/python/cpython/issues/99730]: HEAD requests are no longer upgraded to GET request during redirects in urllib.

- gh-66410 [https://github.com/python/cpython/issues/66410]: Setting the **`tkinter`** module global **`wantobjects`** to `2` before creating the **`Tk`** object or call the **`wantobjects()`** method of the **`Tk`** object with argument `2` makes now arguments to callbacks registered in the **`tkinter`** module to be passed as various Python objects (`int`, `float`, `bytes`, `tuple`), depending on their internal representation in Tcl, instead of always `str`. **`tkinter.wantobjects`** is now set to `2` by default.

- bpo-40943 [https://bugs.python.org/issue?@action=redirect&bpo=40943]: Fix several IndexError when parse emails with truncated Message-ID, address, routes, etc, e.g. `example@`.

- bpo-39324 [https://bugs.python.org/issue?@action=redirect&bpo=39324]: Add mime type mapping for .md <-> text/markdown

- bpo-18108 [https://bugs.python.org/issue?@action=redirect&bpo=18108]: **`shutil.chown()`** now supports *dir_fd* and *follow_symlinks* keyword arguments.

- bpo-30988 [https://bugs.python.org/issue?@action=redirect&bpo=30988]: Fix parsing of emails with invalid address headers having a leading or trailing dot. Patch by tsufeki.

- bpo-32839 [https://bugs.python.org/issue?@action=redirect&bpo=32839]: Add the **`after_info()`** method for Tkinter widgets.

Documentation

- gh-117928 [https://github.com/python/cpython/issues/117928]: The minimum Sphinx version required for the documentation is now 6.2.1.

Build

- gh-118734 [https://github.com/python/cpython/issues/118734]: Fixes Windows build when invoked directly (not through the `build.bat` script) without specifying a value for `UseTIER2`.
- gh-115119 [https://github.com/python/cpython/issues/115119]: The `configure` option **`--with-system-libmpdec`** now defaults to `yes`. The bundled copy of `libmpdecimal` will be removed in Python 3.15.
- gh-117845 [https://github.com/python/cpython/issues/117845]: Fix building against recent libedit versions by detecting readline hook signatures in **configure**.
- gh-116622 [https://github.com/python/cpython/issues/116622]: A testbed project was added to run the test suite on Android.
- gh-117645 [https://github.com/python/cpython/issues/117645]: Increase WASI stack size from 512 KiB to 8 MiB and the initial memory from 10 MiB to 20 MiB. Patch by Victor Stinner.
- gh-115119 [https://github.com/python/cpython/issues/115119]: **configure** now uses **pkg-config** to detect **`decimal`** dependencies if the **`--with-system-libmpdec`** option is given.

Windows

- gh-115119 [https://github.com/python/cpython/issues/115119]: Update Windows installer to use libmpdecimal 4.0.0.
- gh-118486 [https://github.com/python/cpython/issues/118486]: **`os.mkdir()`** now accepts *mode* of `0o700` to restrict the new directory to the current user.
- gh-118347 [https://github.com/python/cpython/issues/118347]: Fixes launcher updates not being installed.
- gh-118293 [https://github.com/python/cpython/issues/118293]: The `multiprocessing` module now passes the `STARTF_FORCEOFFFEEDBACK` flag when spawning processes to tell Windows not to change the mouse cursor.
- gh-115009 [https://github.com/python/cpython/issues/115009]: Update Windows installer to use SQLite 3.45.3.
- gh-90329 [https://github.com/python/cpython/issues/90329]: Suppress the warning displayed on virtual environment creation when the requested and created paths differ only by a short (8.3 style) name. Warnings will continue to be shown if a junction or symlink in the path caused the venv to be created in a different location than originally requested.
- gh-117786 [https://github.com/python/cpython/issues/117786]: Fixes virtual environments not correctly launching when created from a Store install.

macOS

- gh-115119 [https://github.com/python/cpython/issues/115119]: Update macOS installer to use libmpdecimal 4.0.0.
- gh-114099 [https://github.com/python/cpython/issues/114099]: iOS preprocessor symbol usage was made compatible with older macOS SDKs.
- gh-115009 [https://github.com/python/cpython/issues/115009]: Update macOS installer to use SQLite 3.45.3.
- gh-91629 [https://github.com/python/cpython/issues/91629]: Use `~/.config/fish/conf.d` configs and **fish_add_path** to set **`PATH`** when installing for the Fish shell.

IDLE

- bpo-34774 [https://bugs.python.org/issue?@action=redirect&bpo=34774]: Use user-selected color theme for

Help = > IDLE Doc.

C API

- gh-118124 [https://github.com/python/cpython/issues/118124]: Fix **`Py_BUILD_ASSERT`** and **`Py_BUILD_ASSERT_EXPR`** for non-constant expressions: use `static_assert()` on C11 and newer. Patch by Victor Stinner.
- gh-110850 [https://github.com/python/cpython/issues/110850]: Add «Raw» variant of PyTime functions

    - ○ **`PyTime_MonotonicRaw()`**
    - ○ **`PyTime_PerfCounterRaw()`**
    - ○ **`PyTime_TimeRaw()`**

  Patch by Victor Stinner.
- gh-117987 [https://github.com/python/cpython/issues/117987]: Restore functions removed in Python 3.13 alpha 1:

    - ○ **`Py_SetPythonHome()`**
    - ○ **`Py_SetProgramName()`**
    - ○ **`PySys_SetArgvEx()`**
    - ○ **`PySys_SetArgv()`**

  Patch by Victor Stinner.
- gh-117929 [https://github.com/python/cpython/issues/117929]: Restore removed **`PyEval_InitThreads()`** function. Patch by Victor Stinner.
- gh-117534 [https://github.com/python/cpython/issues/117534]: Improve validation logic in the C implementation of **`datetime.datetime.fromisoformat()`** to better handle invalid years. Patch by Vlad Efanov.
- gh-68114 [https://github.com/python/cpython/issues/68114]: Fixed skipitem()”s handling of the old “w” and “w#” formatters. These are no longer supported and now raise an exception if used.

- gh-111997 [https://github.com/python/cpython/issues/111997]: Add a C-API for firing monitoring events.

Python 3.13.0 alpha 6

Release date: 2024-04-09

Core and Builtins

- gh-117494 [https://github.com/python/cpython/issues/117494]: Refactored the instruction sequence data structure out of compile.c into instruction_sequence.c.

- gh-116968 [https://github.com/python/cpython/issues/116968]: Introduce a unified 16-bit backoff counter type (`_Py_BackoffCounter`), shared between the Tier 1 adaptive specializer and the Tier 2 optimizer. The API used for adaptive specialization counters is changed but the behavior is (supposed to be) identical.

  The behavior of the Tier 2 counters is changed:

  - There are no longer dynamic thresholds (we never varied these).
  - All counters now use the same exponential backoff.
  - The counter for `JUMP_BACKWARD` starts counting down from 16.
  - The `temperature` in side exits starts counting down from 64.

- gh-117431 [https://github.com/python/cpython/issues/117431]: Improve the performance of the following `bytes` and `bytearray` methods by adapting them to the `METH_FASTCALL` calling convention:

  - **`endswith()`**
  - **`startswith()`**

- gh-117431 [https://github.com/python/cpython/issues/117431]: Improve the performance of the following `str` methods by adapting them to the **`METH_FASTCALL`** calling convention:

  - **`count()`**
  - **`endswith()`**
  - **`find()`**
  - **`index()`**
  - **`rfind()`**
  - **`rindex()`**
  - **`startswith()`**

- gh-117411 [https://github.com/python/cpython/issues/117411]: Move `PyFutureFeatures` to an internal header and make it private.

- gh-109120 [https://github.com/python/cpython/issues/109120]: Added handle of incorrect star expressions, e.g `f(3, *)`. Patch by Grigoryev Semyon

- gh-117266 [https://github.com/python/cpython/issues/117266]: Fix crashes for certain user-created subclasses of **`ast.AST`**. Such classes are now expected to set the `_field_types` attribute.

- gh-99108 [https://github.com/python/cpython/issues/99108]: Updated the **`hashlib`** built-in HACL* project [https://github.com/hacl-star/hacl-star] C code from upstream that we use for many implementations when they are not present via OpenSSL in a given build. This also avoids the rare potential for a C symbol name one definition rule linking issue.

- gh-117108 [https://github.com/python/cpython/issues/117108]: Change the old space bit of objects in the young generation from 0 to gcstate->visited, so that any objects created during GC will have the old bit set correctly if they get moved into the old generation.

- gh-117108 [https://github.com/python/cpython/issues/117108]: The cycle GC now chooses the size of increments based on the total heap size, instead of the rate of object creation. This ensures that it can keep up with growing heaps.

- gh-116735 [https://github.com/python/cpython/issues/116735]: For

`INSTRUMENTED_CALL_FUNCTION_EX`, set `arg0` to `sys.monitoring.MISSING` instead of `None` for **`CALL`** event.

- gh-113964 [https://github.com/python/cpython/issues/113964]: Starting new threads and process creation through **`os.fork()`** are now only prevented once all non-daemon threads exit.

- gh-116626 [https://github.com/python/cpython/issues/116626]: Ensure `INSTRUMENTED_CALL_FUNCTION_EX` always emits **`CALL`**

- gh-116554 [https://github.com/python/cpython/issues/116554]: `list.sort()` now exploits more cases of partial ordering, particularly those with long descending runs with sub-runs of equal values. Those are recognized as single runs now (previously, each block of repeated values caused a new run to be created).

- gh-114099 [https://github.com/python/cpython/issues/114099]: Added a Loader that can discover extension modules in an iOS-style Frameworks folder.

- gh-115775 [https://github.com/python/cpython/issues/115775]: Compiler populates the new `__static_attributes__` field on a class with the names of attributes of this class which are accessed through self.X from any function in its body.

- gh-115776 [https://github.com/python/cpython/issues/115776]: The array of values, the `PyDictValues` struct is now embedded in the object during allocation. This provides better performance in the common case, and does not degrade as much when the object's `__dict__` is materialized.

- gh-108362 [https://github.com/python/cpython/issues/108362]: Implement an incremental cyclic garbage collector. By collecting the old generation in increments, there is no need for a full heap scan. This can hugely reduce maximum pause time for programs with large heaps.

Reduce the number of generations from three to two. The old generation is split into two spaces, «visited» and «pending».

Collection happens in two steps:: * An increment is formed from the young generation and a small part of the pending space. * This increment is scanned and the survivors moved to the end of the visited space.

When the collecting space becomes empty, the two spaces are swapped.

Library

- gh-109870 [https://github.com/python/cpython/issues/109870]: Dataclasses now calls **`exec()`** once per dataclass, instead of once per method being added. This can speed up dataclass creation by up to 20%.
- gh-97901 [https://github.com/python/cpython/issues/97901]: Mime type `text/rtf` is now supported by **`mimetypes`**.

Core and Builtins

- bpo-24612 [https://bugs.python.org/issue?@action=redirect&bpo=24612]: Improve the **`SyntaxError`** that happens when “not” appears after an operator. Patch by Pablo Galindo

Library

- gh-117648 [https://github.com/python/cpython/issues/117648]: Improve performance of **`os.path.join()`** and **`os.path.expanduser()`**.

- gh-117584 [https://github.com/python/cpython/issues/117584]: Raise **`TypeError`** for non-paths in **`posixpath.relpath`**.

- gh-117467 [https://github.com/python/cpython/issues/117467]: Preserve mailbox ownership when rewriting in **`mailbox.mbox.flush`**. Patch by Tony Mountifield.

- gh-114848 [https://github.com/python/cpython/issues/114848]: Raise `FileNotFoundError` when `getcwd()` returns "(unreachable)", which can happen on Linux > = 2.6.36 with glibc < 2.27.

- gh-117459 [https://github.com/python/cpython/issues/117459]: `asyncio.asyncio.run_coroutine_threadsafe()` now keeps the traceback of `CancelledError`, `TimeoutError` and `InvalidStateError` which are raised in the coroutine.

- gh-117381 [https://github.com/python/cpython/issues/117381]: Fix error message for `ntpath.commonpath()`.

- gh-117337 [https://github.com/python/cpython/issues/117337]: Deprecate undocumented `glob.glob0()` and `glob.glob1()` functions. Use `glob.glob()` and pass a directory to its *root_dir* argument instead.

- gh-117349 [https://github.com/python/cpython/issues/117349]: Optimise several functions in `os.path`.

- gh-117348 [https://github.com/python/cpython/issues/117348]: Refactored `configparser.RawConfigParser._read()` to reduce cyclometric complexity and improve comprehensibility.

- gh-117335 [https://github.com/python/cpython/issues/117335]: Raise TypeError for non-sequences for `ntpath.commonpath()`.

- gh-66449 [https://github.com/python/cpython/issues/66449]: `configparser.ConfigParser` now accepts unnamed sections before named ones, if configured to do so.

- gh-88014 [https://github.com/python/cpython/issues/88014]: In documentation of `gzip.GzipFile` in module gzip, explain data type of optional constructor argument *mtime*, and recommend `mtime = 0` for generating deterministic streams.

- gh-117310 [https://github.com/python/cpython/issues/117310]: Fixed

an unlikely early & extra `Py_DECREF` triggered crash in **`ssl`** when creating a new `_ssl._SSLContext` if CPython was built implausibly such that the default cipher list is empty **or** the SSL library it was linked against reports a failure from its C `SSL_CTX_set_cipher_list()` API.

- gh-117294 [https://github.com/python/cpython/issues/117294]: A `DocTestCase` now reports as skipped if all examples in the doctest are skipped.
- gh-98966 [https://github.com/python/cpython/issues/98966]: In **`subprocess`**, raise a more informative message when `stdout=STDOUT`.
- gh-117225 [https://github.com/python/cpython/issues/117225]: doctest: only print «and X failed» when non-zero, don't pluralise «1 items». Patch by Hugo van Kemenade.
- gh-117205 [https://github.com/python/cpython/issues/117205]: Speed up **`compileall.compile_dir()`** by 20% when using multiprocessing by increasing `chunksize`.
- gh-117178 [https://github.com/python/cpython/issues/117178]: Fix regression in lazy loading of self-referential modules, introduced in gh-114781 [https://github.com/python/cpython/issues/114781].
- gh-112383 [https://github.com/python/cpython/issues/112383]: Fix **`dis`** module's handling of `ENTER_EXECUTOR` instructions.
- gh-117182 [https://github.com/python/cpython/issues/117182]: Lazy-loading of modules that modify their own `__class__` no longer reverts the `__class__` to **`types.ModuleType`**.
- gh-117084 [https://github.com/python/cpython/issues/117084]: Fix **`zipfile`** extraction for directory entries with the name containing backslashes on Windows.
- gh-117114 [https://github.com/python/cpython/issues/117114]: Make **`os.path.isdevdrive()`** available on all platforms. For those that do not offer Dev Drives, it will always return

`False`.

- gh-117110 [https://github.com/python/cpython/issues/117110]: Fix a bug that prevents subclasses of `typing.Any` to be instantiated with arguments. Patch by Chris Fu.

- gh-109653 [https://github.com/python/cpython/issues/109653]: Deferred select imports in importlib.metadata and importlib.resources for a 14% speedup.

- gh-70647 [https://github.com/python/cpython/issues/70647]: Start the deprecation period for the current behavior of `datetime.datetime.strptime()` and `time.strptime()` which always fails to parse a date string with a `ValueError` involving a day of month such as `strptime("02-29", "%m-%d")` when a year is **not** specified and the date happen to be February 29th. This should help avoid users finding new bugs every four years due to a natural mistaken assumption about the API when parsing partial date values.

- gh-116987 [https://github.com/python/cpython/issues/116987]: Fixed `inspect.findsource()` for class code objects.

- gh-114099 [https://github.com/python/cpython/issues/114099]: Modify standard library to allow for iOS platform differences.

- gh-90872 [https://github.com/python/cpython/issues/90872]: On Windows, `subprocess.Popen.wait()` no longer calls `WaitForSingleObject()` with a negative timeout: pass `0` ms if the timeout is negative. Patch by Victor Stinner.

- gh-116957 [https://github.com/python/cpython/issues/116957]: configparser: Don’t leave ConfigParser values in an invalid state (stored as a list instead of a str) after an earlier read raised DuplicateSectionError or DuplicateOptionError.

- gh-115538 [https://github.com/python/cpython/issues/115538]: `_io.WindowsConsoleIO` now emit a warning if a boolean value is passed as a filedescriptor argument.

- gh-90095 [https://github.com/python/cpython/issues/90095]: Ignore empty lines and comments in `.pdbrc`

- gh-106531 [https://github.com/python/cpython/issues/106531]: Refreshed zipfile._path from zipp 3.18 [https://zipp.readthedocs.io/en/latest/history.html#v3-18-0], providing better compatibility for PyPy, better glob performance for deeply nested zipfiles, and providing internal access to `CompleteDirs.inject` for use in other tests (like importlib.resources).

- gh-63207 [https://github.com/python/cpython/issues/63207]: On Windows, **`time.time()`** now uses the `GetSystemTimePreciseAsFileTime()` clock to have a resolution better than 1 us, instead of the `GetSystemTimeAsFileTime()` clock which has a resolution of 15.6 ms. Patch by Victor Stinner.

- gh-116764 [https://github.com/python/cpython/issues/116764]: Restore support of `None` and other false values in **`urllib.parse`** functions **`parse_qs()`** and **`parse_qsl()`**. Also, they now raise a TypeError for non-zero integers and non-empty sequences.

- gh-116811 [https://github.com/python/cpython/issues/116811]: In `PathFinder.invalidate_caches`, delegate to `MetadataPathFinder.invalidate_caches`.

- gh-116647 [https://github.com/python/cpython/issues/116647]: Fix recursive child in dataclasses

- gh-113171 [https://github.com/python/cpython/issues/113171]: Fixed various false positives and false negatives in

    - **`ipaddress.IPv4Address.is_private`** (see these docs for details)
    - **`ipaddress.IPv4Address.is_global`**
    - **`ipaddress.IPv6Address.is_private`**
    - **`ipaddress.IPv6Address.is_global`**

  Also in the corresponding **`ipaddress.IPv4Network`** and **`ipaddress.IPv6Network`** attributes.

- gh-63283 [https://github.com/python/cpython/issues/63283]: In `encodings.idna`, any capitalization of the ACE prefix (`xn--`) is now acceptable. Patch by Pepijn de Vos and Zackery Spytz.

- gh-71042 [https://github.com/python/cpython/issues/71042]: Add `platform.android_ver()`, which provides device and OS information on Android.

- gh-73468 [https://github.com/python/cpython/issues/73468]: Added new `math.fma()` function, wrapping C99's `fma()` operation: fused multiply-add function. Patch by Mark Dickinson and Victor Stinner.

- gh-116608 [https://github.com/python/cpython/issues/116608]: The `importlib.resources` functions `is_resource()`, `open_binary()`, `open_text()`, `path()`, `read_binary()`, and `read_text()` are un-deprecated, and support subdirectories via multiple positional arguments. The `contents()` function also allows subdirectories, but remains deprecated.

- gh-116484 [https://github.com/python/cpython/issues/116484]: Change automatically generated `tkinter.Checkbutton` widget names to avoid collisions with automatically generated `tkinter.ttk.Checkbutton` widget names within the same parent widget.

- gh-114314 [https://github.com/python/cpython/issues/114314]: In `ctypes`, ctype data is now stored in type objects directly rather than in a dict subclass. This is an internal change that should not affect usage.

- gh-116401 [https://github.com/python/cpython/issues/116401]: Fix blocking `os.fwalk()` and `shutil.rmtree()` on opening named pipe.

- gh-71052 [https://github.com/python/cpython/issues/71052]: Implement `ctypes.util.find_library()` on Android.

- gh-90535 [https://github.com/python/cpython/issues/90535]: Fix

support of *interval* values > 1 in **`logging.TimedRotatingFileHandler`** for `when='MIDNIGHT'` and `when='Wx'`.

- gh-113308 [https://github.com/python/cpython/issues/113308]: Remove some internal protected parts from **`uuid`**: `_has_uuid_generate_time_safe`, `_netbios_getnode`, `_ipconfig_getnode`, and `_load_system_functions`. They were unused.

- gh-115627 [https://github.com/python/cpython/issues/115627]: Fix the **`ssl`** module error handling of connection terminate by peer. It now throws an OSError with the appropriate error code instead of an EOFError.

- gh-114847 [https://github.com/python/cpython/issues/114847]: Speed up **`os.path.realpath()`** on non-Windows platforms.

- gh-114271 [https://github.com/python/cpython/issues/114271]: Fix a race in `threading.Thread.join()`.

  `threading._MainThread` now always represents the main thread of the main interpreter.

  `PyThreadState.on_delete` and `PyThreadState.on_delete_data` have been removed.

- gh-113538 [https://github.com/python/cpython/issues/113538]: Add **`asyncio.Server.close_clients()`** and **`asyncio.Server.abort_clients()`** methods which allow to more forcefully close an asyncio server.

- gh-85287 [https://github.com/python/cpython/issues/85287]: Changes Unicode codecs to return UnicodeEncodeError or UnicodeDecodeError, rather than just UnicodeError.

- gh-113548 [https://github.com/python/cpython/issues/113548]: **`pdb`** now allows CLI arguments to `pdb -m`.

- gh-112948 [https://github.com/python/cpython/issues/112948]: Make completion of **`pdb`** similar to Python REPL

- gh-105866 [https://github.com/python/cpython/issues/105866]: Fixed `_get_slots` bug which caused error when defining dataclasses with slots and a weakref_slot.

- gh-96471 [https://github.com/python/cpython/issues/96471]: Add **`asyncio.Queue`** termination with **`shutdown()`** method.

- gh-89739 [https://github.com/python/cpython/issues/89739]: The **`zipimport`** module can now read ZIP64 files.

- bpo-33533 [https://bugs.python.org/issue?@action=redirect&bpo=33533]: **`asyncio.as_completed()`** now returns an object that is both an asynchronous iterator and plain iterator. The new asynchronous iteration pattern allows for easier correlation between prior tasks and their completed results. This is a closer match to **`concurrent.futures.as_completed()`**"s iteration pattern. Patch by Justin Arthur.

- bpo-27578 [https://bugs.python.org/issue?@action=redirect&bpo=27578]: **`inspect.getsource()`** (and related functions) work with empty module files, returning `'\n'` (or reasonable equivalent) instead of raising `OSError`. Patch by Kernc.

- bpo-37141 [https://bugs.python.org/issue?@action=redirect&bpo=37141]: Accept an iterable of separators in **`asyncio.StreamReader.readuntil()`**, stopping when one of them is encountered.

- gh-66543 [https://github.com/python/cpython/issues/66543]: Make **`mimetypes.guess_type()`** properly parsing of URLs with only a host name, URLs containing fragment or query, and filenames with only a UNC sharepoint on Windows. Based on patch by Dong-hee Na.

- bpo-15010 [https://bugs.python.org/issue?@action=redirect&bpo=15010]: **`unittest.TestLoader.discover()`** now saves the original value of `unittest.TestLoader._top_level_dir` and restores it at the end of the call.

Documentation

- gh-115977 [https://github.com/python/cpython/issues/115977]: Remove compatibility references to Emscripten.
- gh-114099 [https://github.com/python/cpython/issues/114099]: Add an iOS platform guide, and flag modules not available on iOS.
- gh-91565 [https://github.com/python/cpython/issues/91565]: Changes to documentation files and config outputs to reflect the new location for reporting bugs - i.e. GitHub rather than bugs.python.org.

Tests

- gh-83434 [https://github.com/python/cpython/issues/83434]: Disable JUnit XML output (`--junit-xml=FILE` command line option) in regrtest when hunting for reference leaks (`-R` option). Patch by Victor Stinner.
- gh-117187 [https://github.com/python/cpython/issues/117187]: Fix XML tests for vanilla Expat <2.6.0.
- gh-116333 [https://github.com/python/cpython/issues/116333]: Tests of TLS related things (error codes, etc) were updated to be more lenient about specific error message strings and behaviors as seen in the BoringSSL and AWS-LC forks of OpenSSL.
- gh-117089 [https://github.com/python/cpython/issues/117089]: Consolidated tests for importlib.metadata in their own `metadata` package.
- gh-115979 [https://github.com/python/cpython/issues/115979]: Update test_importlib so that it passes under WASI SDK 21.
- gh-112536 [https://github.com/python/cpython/issues/112536]: Add –tsan to test.regrtest for running TSAN tests in reasonable execution times. Patch by Donghee Na.
- gh-116307 [https://github.com/python/cpython/issues/116307]: Added import helper `isolated_modules` as `CleanImport` does not remove modules imported during the context. Use it in importlib.resources tests to avoid leaving `mod` around to impede importlib.metadata tests.

Build

- gh-114736 [https://github.com/python/cpython/issues/114736]: Have WASI builds use WASI SDK 21.
- gh-115983 [https://github.com/python/cpython/issues/115983]: Skip building test modules that must be built as shared under WASI.
- gh-71052 [https://github.com/python/cpython/issues/71052]: Add Android build script and instructions.

Windows

- gh-117267 [https://github.com/python/cpython/issues/117267]: Ensure `DirEntry.stat().st_ctime` behaves consistently with **`os.stat()`** during the deprecation period of `st_ctime` by containing the same value as `st_birthtime`. After the deprecation period, `st_ctime` will be the metadata change time (or unavailable through `DirEntry`), and only `st_birthtime` will contain the creation time.
- gh-116195 [https://github.com/python/cpython/issues/116195]: Improves performance of **`os.getppid()`** by using an alternate system API when available. Contributed by vxiiduu.
- gh-88494 [https://github.com/python/cpython/issues/88494]: On Windows, **`time.monotonic()`** now uses the `QueryPerformanceCounter()` clock to have a resolution better than 1 us, instead of the `GetTickCount64()` clock which has a resolution of 15.6 ms. Patch by Victor Stinner.
- gh-116773 [https://github.com/python/cpython/issues/116773]: Fix instances of `<_overlapped.Overlapped object at 0xXXX> still has pending operation at deallocation, the process may crash`.
- gh-91227 [https://github.com/python/cpython/issues/91227]: Fix the asyncio ProactorEventLoop implementation so that sending a datagram to an address that is not listening does not prevent receiving any more datagrams.
- gh-115119 [https://github.com/python/cpython/issues/115119]: Switched from vendored `libmpdecimal` code to a separately-hosted external package in the `cpython-source-deps` repository when building the `_decimal` module.

C API

- gh-117642 [https://github.com/python/cpython/issues/117642]: Fix **PEP 737** [https://peps.python.org/pep-0737/] implementation for `%#T` and `%#N`.
- gh-87193 [https://github.com/python/cpython/issues/87193]: `_PyBytes_Resize()` can now be called for bytes objects with reference count > 1, including 1-byte bytes objects. It creates a new bytes object and destroys the old one if it has reference count > 1.
- gh-117021 [https://github.com/python/cpython/issues/117021]: Fix integer overflow in `PyLong_AsPid()` on non-Windows 64-bit platforms.
- gh-115756 [https://github.com/python/cpython/issues/115756]: `PyCode_GetFirstFree()` is an ustable API now and has been renamed to `PyUnstable_Code_GetFirstFree()`. (Contributed by Bogdan Romanyuk in gh-115781 [https://github.com/python/cpython/issues/115781])
- gh-116869 [https://github.com/python/cpython/issues/116869]: Add `test_cext` test: build a C extension to check if the Python C API emits C compiler warnings. Patch by Victor Stinner.
- gh-116869 [https://github.com/python/cpython/issues/116869]: Make the C API compatible with `-Werror=declaration-after-statement` compiler flag again. Patch by Victor Stinner.
- gh-116936 [https://github.com/python/cpython/issues/116936]: Add `PyType_GetModuleByDef()` to the limited C API. Patch by Victor Stinner.
- gh-116809 [https://github.com/python/cpython/issues/116809]: Restore removed private `_PyErr_ChainExceptions1()` function. Patch by Victor Stinner.
- gh-115754 [https://github.com/python/cpython/issues/115754]: In the limited C API version 3.13, getting `Py_None`, `Py_False`, `Py_True`, `Py_Ellipsis` and `Py_NotImplemented` singletons is now implemented as function calls at the stable ABI level to hide implementation details. Getting these constants still return borrowed references. Patch by Victor Stinner.
- gh-115754 [https://github.com/python/cpython/issues/115754]: Add `Py_GetConstant()` and `Py_GetConstantBorrowed()`

functions to get constants. For example, `Py_GetConstant(Py_CONSTANT_ZERO)` returns a strong reference to the constant zero. Patch by Victor Stinner.

- gh-111696 [https://github.com/python/cpython/issues/111696]: Add support for `%T`, `%T#`, `%N` and `%N#` formats to **`PyUnicode_FromFormat()`**: format the fully qualified name of an object type and of a type: call **`PyType_GetModuleName()`**. See **PEP 737** [https://peps.python.org/pep-0737/] for more information. Patch by Victor Stinner.
- gh-111696 [https://github.com/python/cpython/issues/111696]: Add **`PyType_GetModuleName()`** function to get the type's module name. Equivalent to getting the `type.__module__` attribute. Patch by Eric Snow and Victor Stinner.
- gh-111696 [https://github.com/python/cpython/issues/111696]: Add **`PyType_GetFullyQualifiedName()`** function to get the type's fully qualified name. Equivalent to `f"{type.__module__}.{type.__qualname__}"`, or `type.__qualname__` if `type.__module__` is not a string or is equal to `"builtins"`. Patch by Victor Stinner.
- gh-85283 [https://github.com/python/cpython/issues/85283]: The `fcntl`, `grp`, `pwd`, `termios`, `_statistics` and `_testconsole` C extensions are now built with the limited C API. Patch by Victor Stinner.
- gh-111140 [https://github.com/python/cpython/issues/111140]: Add additional flags to **`PyLong_AsNativeBytes()`** and **`PyLong_FromNativeBytes()`** to allow the caller to determine how to handle edge cases around values that fill the entire buffer.
- gh-113024 [https://github.com/python/cpython/issues/113024]: Add **`PyObject_GenericHash()`** function.

Python 3.13.0 alpha 5

Release date: 2024-03-12

Security

- gh-115398 [https://github.com/python/cpython/issues/115398]: Allow

controlling Expat >=2.6.0 reparse deferral (**CVE 2023-52425** [https://www.cve.org/CVERecord?id=CVE-2023-52425]) by adding five new methods:
  - `xml.etree.ElementTree.XMLParser.flush()`
  - `xml.etree.ElementTree.XMLPullParser.flush()`
  - `xml.parsers.expat.xmlparser.GetReparseDeferralE`
  - `xml.parsers.expat.xmlparser.SetReparseDeferralE`
  - `xml.sax.expatreader.ExpatParser.flush()`
- gh-114572 [https://github.com/python/cpython/issues/114572]: `ssl.SSLContext.cert_store_stats()` and `ssl.SSLContext.get_ca_certs()` now correctly lock access to the certificate store, when the `ssl.SSLContext` is shared across multiple threads.

Core and Builtins

- gh-116604 [https://github.com/python/cpython/issues/116604]: Respect the status of the garbage collector when indirect calls are made via `PyErr_CheckSignals()` and the evaluation breaker. Patch by Pablo Galindo
- gh-112087 [https://github.com/python/cpython/issues/112087]: `list` is now compatible with the implementation of **PEP 703** [https://peps.python.org/pep-0703/].
- gh-116381 [https://github.com/python/cpython/issues/116381]: Add specialization for `CONTAINS_OP`.
- gh-116296 [https://github.com/python/cpython/issues/116296]: Fix possible refleak in `object.__reduce__()` internal error handling.
- gh-115823 [https://github.com/python/cpython/issues/115823]: Properly calculate error ranges in the parser when raising `SyntaxError` exceptions caused by invalid byte sequences. Patch by Pablo Galindo
- gh-115778 [https://github.com/python/cpython/issues/115778]: Add `tierN` annotation for instruction definition in interpreter DSL.
- gh-115733 [https://github.com/python/cpython/issues/115733]: Fix crash when calling `next()` on exhausted list iterators.
- gh-115700 [https://github.com/python/cpython/issues/115700]: The regen-cases build stage now works on Windows.
- gh-115347 [https://github.com/python/cpython/issues/115347]: Fix

bug where docstring was replaced by a redundant NOP when Python is run with `-OO`.
- gh-115323 [https://github.com/python/cpython/issues/115323]: Make error message more meaningful for when **`bytearray.extend()`** is called with a **`str`** object.
- gh-112175 [https://github.com/python/cpython/issues/112175]: Every `PyThreadState` now has its own `eval_breaker`, allowing specific threads to be interrupted.

Library

- gh-115154 [https://github.com/python/cpython/issues/115154]: Fix a bug that was causing the **`tokenize.untokenize()`** function to handle unicode named literals incorrectly. Patch by Pablo Galindo

Core and Builtins

- gh-112433 [https://github.com/python/cpython/issues/112433]: Add ability to force alignment of **`ctypes.Structure`** by way of the new `_align_` attribute on the class.

Library

- gh-104090 [https://github.com/python/cpython/issues/104090]: The multiprocessing resource tracker now exits with non-zero status code if a resource leak was detected. It still exits with status code 0 otherwise.

Core and Builtins

- gh-105858 [https://github.com/python/cpython/issues/105858]: Improve the constructors for **`ast`** nodes. Arguments of list types now default to an empty list if omitted, and optional fields default to `None`. AST nodes now have an `__annotations__` attribute with the expected types of their attributes. Passing unrecognized extra arguments to AST nodes is deprecated and will become an error in Python 3.15. Omitting a required argument to an AST node is deprecated and will become an error in Python 3.15. Patch by Jelle Zijlstra.

- gh-101860 [https://github.com/python/cpython/issues/101860]: Expose `__name__` attribute on property.
- gh-96497 [https://github.com/python/cpython/issues/96497]: Fix incorrect resolution of mangled class variables used in assignment expressions in comprehensions.

Library

- gh-116600 [https://github.com/python/cpython/issues/116600]: Fix **`repr()`** for global **`Flag`** members.

- gh-116349 [https://github.com/python/cpython/issues/116349]: **`platform.java_ver()`** is deprecated and will be removed in 3.15. It was largely untested, had a confusing API, and was only useful for Jython support.

- gh-116143 [https://github.com/python/cpython/issues/116143]: Fix a race in pydoc `_start_server`, eliminating a window in which `_start_server` can return a thread that is «serving» but without a `docserver` set.

- gh-116127 [https://github.com/python/cpython/issues/116127]: **`typing`**: implement **PEP 705** [https://peps.python.org/pep-0705/] which adds **`typing.ReadOnly`** support to **`typing.TypedDict`**.

- gh-116325 [https://github.com/python/cpython/issues/116325]: **`typing`**: raise **`SyntaxError`** instead of **`AttributeError`** on forward references as empty strings.

- gh-115957 [https://github.com/python/cpython/issues/115957]: When `asyncio.TaskGroup.create_task` is called on an inactive `asyncio.TaskGroup`, the given coroutine will be closed (which prevents a `RuntimeWarning`).

- gh-115978 [https://github.com/python/cpython/issues/115978]: Disable preadv(), readv(), pwritev(), and writev() on WASI.

  Under wasmtime for WASI 0.2, these functions don’t pass test_posix (https://github.com/bytecodealliance/wasmtime/issues/7830).

- gh-88352 [https://github.com/python/cpython/issues/88352]: Fix the computation of the next rollover time in the **`logging.TimedRotatingFileHandler`** handler. **`computeRollover()`** now always returns a timestamp larger than the specified time and works correctly during the DST change. **`doRollover()`** no longer overwrite the already rolled over file, saving from data loss when run at midnight or during repeated time at the DST change.

- gh-87115 [https://github.com/python/cpython/issues/87115]: Set `__main__.__spec__` to `None` when running a script with **`pdb`**

- gh-76511 [https://github.com/python/cpython/issues/76511]: Fix UnicodeEncodeError in **`email.Message.as_string()`** that results when a message that claims to be in the ascii character set actually has non-ascii characters. Non-ascii characters are now replaced with the U+FFFD replacement character, like in the `replace` error handler.

- gh-89547 [https://github.com/python/cpython/issues/89547]: Add support for nested typing special forms like Final[ClassVar[int]].

- gh-65824 [https://github.com/python/cpython/issues/65824]: Improve the `less` prompt in **`pydoc`**.

- gh-116040 [https://github.com/python/cpython/issues/116040]: [Enum] fix by-value calls when second value is falsey; e.g. Cardinal(1, 0)

- gh-115821 [https://github.com/python/cpython/issues/115821]: [Enum] Improve error message when calling super().__new__() in custom __new__.

- gh-85644 [https://github.com/python/cpython/issues/85644]: Use the `XDG_CURRENT_DESKTOP` environment variable in **`webbrowser`** to check desktop. Prefer it to the deprecated

`GNOME_DESKTOP_SESSION_ID` for GNOME detection.

- gh-75988 [https://github.com/python/cpython/issues/75988]: Fixed **`unittest.mock.create_autospec()`** to pass the call through to the wrapped object to return the real result.

- gh-115881 [https://github.com/python/cpython/issues/115881]: Fix issue where **`ast.parse()`** would incorrectly flag conditional context managers (such as `with (x() if y else z()): ...`) as invalid syntax if `feature_version=(3, 8)` was passed. This reverts changes to the grammar made as part of gh-94949 [https://github.com/python/cpython/issues/94949].

- gh-115886 [https://github.com/python/cpython/issues/115886]: Fix silent truncation of the name with an embedded null character in **`multiprocessing.shared_memory.SharedMemory`**.

- gh-115532 [https://github.com/python/cpython/issues/115532]: Add kernel density estimation to the statistics module.

- gh-115714 [https://github.com/python/cpython/issues/115714]: On WASI, the **`time`** module no longer get process time using `times()` or `CLOCK_PROCESS_CPUTIME_ID`, system API is that is unreliable and is likely to be removed from WASI. The affected clock functions fall back to calling `clock()`.

- gh-115809 [https://github.com/python/cpython/issues/115809]: Improve algorithm for computing which rolled-over log files to delete in **`logging.TimedRotatingFileHandler`**. It is now reliable for handlers without `namer` and with arbitrary deterministic `namer` that leaves the datetime part in the file name unmodified.

- gh-74668 [https://github.com/python/cpython/issues/74668]: **`urllib.parse`** functions **`parse_qs()`** and **`parse_qsl()`** now support bytes arguments containing raw and percent-encoded non-ASCII data.

- gh-67044 [https://github.com/python/cpython/issues/67044]:

`csv.writer()` now always quotes or escapes `'\r'` and `'\n'`, regardless of *lineterminator* value.

- gh-115712 [https://github.com/python/cpython/issues/115712]: Restore support of space delimiter with `skipinitialspace=True` in `csv`. `csv.writer()` now quotes empty fields if delimiter is a space and skipinitialspace is true and raises exception if quoting is not possible.

- gh-112364 [https://github.com/python/cpython/issues/112364]: Fixed `ast.unparse()` to handle format_spec with `"`, `'` or `\\`. Patched by Frank Hoffmann.

- gh-112997 [https://github.com/python/cpython/issues/112997]: Stop logging potentially sensitive callback arguments in `asyncio` unless debug mode is active.

- gh-114914 [https://github.com/python/cpython/issues/114914]: Fix an issue where an abandoned `StreamWriter` would not be garbage collected.

- gh-111358 [https://github.com/python/cpython/issues/111358]: Fix a bug in `asyncio.BaseEventLoop.shutdown_default_executor()` to ensure the timeout passed to the coroutine behaves as expected.

- gh-115618 [https://github.com/python/cpython/issues/115618]: Fix improper decreasing the reference count for `None` argument in `property` methods `getter()`, `setter()` and `deleter()`.

- gh-112720 [https://github.com/python/cpython/issues/112720]: Refactor `dis.ArgResolver` to make it possible to subclass and change the way jump args are interpreted.

- gh-112006 [https://github.com/python/cpython/issues/112006]: Fix `inspect.unwrap()` for types with the `__wrapper__` data descriptor. Fix `inspect.Signature.from_callable()` for builtins `classmethod()` and `staticmethod()`.

- gh-101293 [https://github.com/python/cpython/issues/101293]: Support callables with the `__call__()` method and types with `__new__()` and `__init__()` methods set to class methods, static methods, bound methods, partial functions, and other types of methods and descriptors in **`inspect.Signature.from_callable()`**.

- gh-103092 [https://github.com/python/cpython/issues/103092]: Isolate **`_lsprof`** (apply **PEP 687** [https://peps.python.org/pep-0687/]).

- gh-113942 [https://github.com/python/cpython/issues/113942]: **`pydoc`** no longer skips global functions implemented as builtin methods, such as **`MethodDescriptorType`** and **`WrapperDescriptorType`**.

- gh-115256 [https://github.com/python/cpython/issues/115256]: Added DeprecationWarning when accessing the tarfile attribute of TarInfo objects. The attribute is never used internally and is only attached to TarInfos when the tarfile is opened in write-mode, not read-mode. The attribute creates an unnecessary reference cycle which may cause corruption when not closing the handle after writing a tarfile.

- gh-115197 [https://github.com/python/cpython/issues/115197]: `urllib.request` no longer resolves the hostname before checking it against the system's proxy bypass list on macOS and Windows.

- gh-113812 [https://github.com/python/cpython/issues/113812]: **`DatagramTransport.sendto()`** will now send zero-length datagrams if called with an empty bytes object. The transport flow control also now accounts for the datagram header when calculating the buffer size.

- gh-114763 [https://github.com/python/cpython/issues/114763]: Protect modules loaded with **`importlib.util.LazyLoader`** from race conditions when multiple threads try to access attributes before the loading is complete.

- gh-114709 [https://github.com/python/cpython/issues/114709]: **`posixpath.commonpath()`** now raises a **`ValueError`** exception when passed an empty iterable. Previously, **`IndexError`** was raised.

  `posixpath.commonpath()` now raises a **`TypeError`** exception when passed `None`. Previously, **`ValueError`** was raised.

- gh-114610 [https://github.com/python/cpython/issues/114610]: Fix bug where **`pathlib.PurePath.with_stem()`** converted a non-empty path suffix to a stem when given an empty *stem* argument. It now raises **`ValueError`**, just like **`pathlib.PurePath.with_suffix()`** does when called on a path with an empty stem, given a non-empty *suffix* argument.

- gh-107361 [https://github.com/python/cpython/issues/107361]: Add **`ssl.VERIFY_X509_PARTIAL_CHAIN`** and **`VERIFY_X509_STRICT`** to the default SSL context created with **`ssl.create_default_context()`**.

- gh-112281 [https://github.com/python/cpython/issues/112281]: Allow creating union of types for **`typing.Annotated`** with unhashable metadata.

- gh-111775 [https://github.com/python/cpython/issues/111775]: Fix **`importlib.resources.simple.ResourceHandle.open()`** for text mode, added missed `stream` argument.

- gh-90095 [https://github.com/python/cpython/issues/90095]: Make .pdbrc and -c work with any valid pdb commands.

- gh-107625 [https://github.com/python/cpython/issues/107625]: Raise **`configparser.ParsingError`** from **`read()`** and **`read_file()`** methods of **`configparser.ConfigParser`** if a key without a corresponding value is continued (that is, followed by an indented line).

- gh-107155 [https://github.com/python/cpython/issues/107155]: Fix incorrect output of `help(x)` where `x` is a **`lambda`** function, which has an `__annotations__` dictionary attribute with a `"return"` key.

- gh-57141 [https://github.com/python/cpython/issues/57141]: Add option for *non-shallow* comparisons to **`filecmp.dircmp`** like **`filecmp.cmp()`**. Original patch by Steven Ward. Enhanced by Tobias Rautenkranz
- gh-69990 [https://github.com/python/cpython/issues/69990]: **`Profile.print_stats()`** has been improved to accept multiple sort arguments. Patched by Chiu-Hsiang Hsu and Furkan Onder.
- gh-104061 [https://github.com/python/cpython/issues/104061]: Add **`socket.SO_BINDTOIFINDEX`** constant.
- gh-60346 [https://github.com/python/cpython/issues/60346]: Fix ArgumentParser inconsistent with parse_known_args.
- gh-102389 [https://github.com/python/cpython/issues/102389]: Add `windows_31j` to aliases for `cp932` codec
- gh-72249 [https://github.com/python/cpython/issues/72249]: Always include the module name in the **`repr()`** of **`functools.partial()`** objects. Patch by Furkan Onder and Anilyka Barry.
- gh-100985 [https://github.com/python/cpython/issues/100985]: Update HTTPSConnection to consistently wrap IPv6 Addresses when using a proxy.
- gh-100884 [https://github.com/python/cpython/issues/100884]: email: fix misfolding of comma in address-lists over multiple lines in combination with unicode encoding.
- gh-95782 [https://github.com/python/cpython/issues/95782]: Fix **`io.BufferedReader.tell()`**, **`io.BufferedReader.seek()`**, **`_pyio.BufferedReader.tell()`**, **`io.BufferedRandom.tell()`**, **`io.BufferedRandom.seek()`** and

`_pyio.BufferedRandom.tell()` being able to return negative offsets.

- gh-96310 [https://github.com/python/cpython/issues/96310]: Fix a traceback in **`argparse`** when all options in a mutually exclusive group are suppressed.

- gh-93205 [https://github.com/python/cpython/issues/93205]: Fixed a bug in **`logging.handlers.TimedRotatingFileHandler`** where multiple rotating handler instances pointing to files with the same name but different extensions would conflict and not delete the correct files.

- bpo-31116 [https://bugs.python.org/issue?@action=redirect&bpo=31116]: Add Z85 encoding to `base64`.

- bpo-44865 [https://bugs.python.org/issue?@action=redirect&bpo=44865]: Add missing call to localization function in **`argparse`**.

- bpo-43952 [https://bugs.python.org/issue?@action=redirect&bpo=43952]: Fix **`multiprocessing.connection.Listener.accept()`** to accept empty bytes as authkey. Not accepting empty bytes as key causes it to hang indefinitely.

- bpo-42125 [https://bugs.python.org/issue?@action=redirect&bpo=42125]: linecache: get module name from `__spec__` if available. This allows getting source code for the `__main__` module when a custom loader is used.

- bpo-41122 [https://bugs.python.org/issue?@action=redirect&bpo=41122]: Failing to pass arguments properly to **`functools.singledispatchmethod()`** now throws a TypeError instead of hitting an index out of bounds internally.

- bpo-40818 [https://bugs.python.org/issue?@action=redirect&bpo=40818]: The asyncio REPL now runs **`sys.__interactivehook__`** on startup. The default implementation of **`sys.__interactivehook__`** provides auto-completion to the asyncio REPL. Patch contributed by Rémi Lapeyre.

- bpo-33775 [https://bugs.python.org/issue?@action=redirect&bpo=33775]: Add “default” and “version” help text for localization in argparse.

Documentation

- gh-115399 [https://github.com/python/cpython/issues/115399]: Document **CVE 2023-52425** [https://www.cve.org/CVERecord?id=CVE-2023-52425] of Expat <2.6.0 under «XML vulnerabilities».
- gh-109653 [https://github.com/python/cpython/issues/109653]: Improve import time of **`uuid`** on Linux.

Tests

- gh-71052 [https://github.com/python/cpython/issues/71052]: Add test exclusions to support running the test suite on Android.
- gh-71052 [https://github.com/python/cpython/issues/71052]: Enable `test_concurrent_futures` on platforms that support threading but not multiprocessing.
- gh-115796 [https://github.com/python/cpython/issues/115796]: Make “_testinternalcapi.assemble_code_object” construct the exception table for the code object.
- gh-115720 [https://github.com/python/cpython/issues/115720]: Leak tests (`-R`, `--huntrleaks`) now show a summary of the number of leaks found in each iteration.
- gh-115122 [https://github.com/python/cpython/issues/115122]: Add `--bisect` option to regrtest test runner: run failed tests with `test.bisect_cmd` to identify failing tests. Patch by Victor Stinner.
- gh-115596 [https://github.com/python/cpython/issues/115596]: Fix `ProgramPriorityTests` in `test_os` permanently changing the process priority.
- gh-115556 [https://github.com/python/cpython/issues/115556]: On Windows, commas passed in arguments to `Tools\buildbot\test.bat` and `PCbuild\\rt.bat` are now

properly handled.

- gh-115420 [https://github.com/python/cpython/issues/115420]: Fix translation of exception handler targets by `_testinternalcapi.optimize_cfg`.
- gh-115376 [https://github.com/python/cpython/issues/115376]: Fix segfault in `_testinternalcapi.compiler_codegen` on bad input.

Build

- gh-116313 [https://github.com/python/cpython/issues/116313]: Get WASI builds to work under wasmtime 18 w/ WASI 0.2/preview2 primitives.
- gh-71052 [https://github.com/python/cpython/issues/71052]: Change Android's **`sys.platform`** from `"linux"` to `"android"`.
- gh-116117 [https://github.com/python/cpython/issues/116117]: Backport `libb2`'s PR #42 to fix compiling CPython on 32-bit Windows with `clang-cl`.
- gh-71052 [https://github.com/python/cpython/issues/71052]: Fix several Android build issues
- gh-114099 [https://github.com/python/cpython/issues/114099]: A testbed project was added to run the test suite on iOS.
- gh-115350 [https://github.com/python/cpython/issues/115350]: Fix building ctypes module with -DWIN32_LEAN_AND_MEAN defined
- gh-111225 [https://github.com/python/cpython/issues/111225]: Link extension modules against libpython on Android.
- gh-115737 [https://github.com/python/cpython/issues/115737]: The install name for libPython is now correctly set for non-framework macOS builds.
- gh-114099 [https://github.com/python/cpython/issues/114099]: Makefile targets were added to support compiling an iOS-compatible framework build.

Windows

- gh-116012 [https://github.com/python/cpython/issues/116012]: Ensure the value of `GetLastError()` is preserved across GIL operations.
- gh-115582 [https://github.com/python/cpython/issues/115582]:

Building extensions intended for free-threaded builds of CPython now require compiling with `/DPy_GIL_DISABLED` manually when using a regular install. This is expected to change in future releases.

- gh-115554 [https://github.com/python/cpython/issues/115554]: The installer now has more strict rules about updating the Python install manager. In general, most users only have a single launcher installed and will see no difference. When multiple launchers have been installed, the option to install the launcher is disabled until all but one have been removed. Downgrading the launcher (which was never allowed) is now more obviously blocked.
- gh-115543 [https://github.com/python/cpython/issues/115543]: Python install manager can now detect Python 3.13 when installed from the Microsoft Store, and will install Python 3.12 by default when **`PYLAUNCHER_ALLOW_INSTALL`** is set.

macOS

- gh-116145 [https://github.com/python/cpython/issues/116145]: Update macOS installer to Tcl/Tk 8.6.14.

IDLE

- gh-88516 [https://github.com/python/cpython/issues/88516]: On macOS show a proxy icon in the title bar of editor windows to match platform behaviour.

Tools/Demos

- gh-100176 [https://github.com/python/cpython/issues/100176]: Remove outdated Tools/{io,cc,string}bench
- bpo-45101 [https://bugs.python.org/issue?@action=redirect&bpo=45101]: Add consistency in usage message IO between 2 versions of python-config.

C API

- gh-114626 [https://github.com/python/cpython/issues/114626]: Add

again `_PyCFunctionFastWithKeywords` name, removed in Python 3.13 alpha 4 by mistake. Keep the old private `_PyCFunctionFastWithKeywords` name (Python 3.7) as an alias to the new public name `PyCFunctionFastWithKeywords` (Python 3.13a4). Patch by Victor Stinner.
- gh-111418 [https://github.com/python/cpython/issues/111418]: Add **`PyHASH_MODULUS`**, **`PyHASH_BITS`**, **`PyHASH_INF`** and **`PyHASH_IMAG`** C macros. Patch by Sergey B Kirpichev.

Python 3.13.0 alpha 4

Release date: 2024-02-15

Security

- gh-115399 [https://github.com/python/cpython/issues/115399]: Update bundled libexpat to 2.6.0
- gh-115243 [https://github.com/python/cpython/issues/115243]: Fix possible crashes in **`collections.deque.index()`** when the deque is concurrently modified.

Core and Builtins

- gh-112087 [https://github.com/python/cpython/issues/112087]: For an empty reverse iterator for list will be reduced to **`reversed()`**. Patch by Donghee Na
- gh-114570 [https://github.com/python/cpython/issues/114570]: Add **`PythonFinalizationError`** exception. This exception derived from **`RuntimeError`** is raised when an operation is blocked during the Python finalization. Patch by Victor Stinner.
- gh-114695 [https://github.com/python/cpython/issues/114695]: Add **`sys._clear_internal_caches()`**, which clears all internal performance-related caches (and deprecate the less-general **`sys._clear_type_cache()`** function).
- gh-114828 [https://github.com/python/cpython/issues/114828]: Fix compilation crashes in uncommon code examples using **`super()`** inside a comprehension in a class body.

- gh-112069 [https://github.com/python/cpython/issues/112069]: Adapt `set` and `frozenset` methods to Argument Clinic.
- gh-115011 [https://github.com/python/cpython/issues/115011]: Setters for members with an unsigned integer type now support the same range of valid values for objects that has a `__index__()` method as for `int`.
- gh-114887 [https://github.com/python/cpython/issues/114887]: Changed socket type validation in `create_datagram_endpoint()` to accept all non-stream sockets. This fixes a regression in compatibility with raw sockets.
- gh-114944 [https://github.com/python/cpython/issues/114944]: Fixes a race between `PyParkingLot_Park` and `_PyParkingLot_UnparkAll`.
- gh-113462 [https://github.com/python/cpython/issues/113462]: Limit the number of versions that a single class can use. Prevents a few wayward classes using up all the version numbers.
- gh-76763 [https://github.com/python/cpython/issues/76763]: The `chr()` builtin function now always raises `ValueError` for values outside the valid range. Previously it raised `OverflowError` for very large or small values.
- gh-114806 [https://github.com/python/cpython/issues/114806]: No longer specialize calls to classes, if those classes have metaclasses. Fixes bug where the `__call__` method of the metaclass was not being called.
- gh-107944 [https://github.com/python/cpython/issues/107944]: Improve error message for function calls with bad keyword arguments via getargs
- gh-112529 [https://github.com/python/cpython/issues/112529]: The free-threaded build no longer allocates space for the `PyGC_Head` structure in objects that support cyclic garbage collection. A number of other fields and data structures are used as replacements, including `ob_gc_bits`, `ob_tid`, and mimalloc internal data structures.
- gh-114456 [https://github.com/python/cpython/issues/114456]: Lower the recursion limit under a debug build of WASI.
- gh-114083 [https://github.com/python/cpython/issues/114083]: Compiler applies folding of LOAD_CONST with following instruction in a separate pass before other optimisations. This enables jump threading in certain circumstances.

- gh-114388 [https://github.com/python/cpython/issues/114388]: Fix a **`RuntimeWarning`** emitted when assign an integer-like value that is not an instance of **`int`** to an attribute that corresponds to a C struct member of type T_UINT and T_ULONG. Fix a double **`RuntimeWarning`** emitted when assign a negative integer value to an attribute that corresponds to a C struct member of type T_UINT.
- gh-114265 [https://github.com/python/cpython/issues/114265]: Compiler propagates line numbers before optimization, leading to more optimization opportunities and removing the need for the `guarantee_lineno_for_exits` hack.
- gh-112529 [https://github.com/python/cpython/issues/112529]: The free-threaded build now has its own thread-safe GC implementation that uses mimalloc to find GC tracked objects. It is non-generational, unlike the existing GC implementation.
- gh-114050 [https://github.com/python/cpython/issues/114050]: Fix segmentation fault caused by an incorrect format string in `TypeError` exception when more than two arguments are passed to `int`.
- gh-112354 [https://github.com/python/cpython/issues/112354]: The `END_FOR` instruction now pops only one value. This is to better support side exits in loops.
- gh-113884 [https://github.com/python/cpython/issues/113884]: Make **`queue.SimpleQueue`** thread safe when the GIL is disabled.
- gh-114058 [https://github.com/python/cpython/issues/114058]: Implement the foundations of the Tier 2 redundancy eliminator.
- gh-113939 [https://github.com/python/cpython/issues/113939]: frame.clear(): Clear frame.f_locals as well, and not only the fast locals. This is relevant once frame.f_locals was accessed, which would contain also references to all the locals.
- gh-112050 [https://github.com/python/cpython/issues/112050]: Convert **`collections.deque`** to use Argument Clinic.
- gh-112050 [https://github.com/python/cpython/issues/112050]: Make methods on **`collections.deque`** thread-safe when the GIL is disabled.
- gh-113464 [https://github.com/python/cpython/issues/113464]: Add an option (`--enable-experimental-jit` for `configure`-based builds or `--experimental-jit` for

`PCbuild`-based ones) to build an *experimental* just-in-time compiler, based on copy-and-patch [https://fredrikbk.com/publications/copy-and-patch.pdf]

- gh-113055 [https://github.com/python/cpython/issues/113055]: Make interp->obmalloc a pointer. For interpreters that share state with the main interpreter, this points to the same static memory structure. For interpreters with their own obmalloc state, it is heap allocated. Add free_obmalloc_arenas() which will free the obmalloc arenas and radix tree structures for interpreters with their own obmalloc state.
- gh-55664 [https://github.com/python/cpython/issues/55664]: Add warning when creating **`type`** using a namespace dictionary with non-string keys. Patched by Daniel Urban and Furkan Onder.
- gh-104530 [https://github.com/python/cpython/issues/104530]: Use native Win32 condition variables.

Library

- gh-115392 [https://github.com/python/cpython/issues/115392]: Fix a bug in **`doctest`** where incorrect line numbers would be reported for decorated functions.

- gh-114563 [https://github.com/python/cpython/issues/114563]: Fix several **`format()`** bugs when using the C implementation of **`Decimal`**: * memory leak in some rare cases when using the `z` format option (coerce negative 0) * incorrect output when applying the `z` format option to type `F` (fixed-point with capital `NAN` / `INF`) * incorrect output when applying the `#` format option (alternate form)

- gh-102840 [https://github.com/python/cpython/issues/102840]: Fix confused traceback when floordiv, mod, or divmod operations happens between instances of **`fractions.Fraction`** and **`complex`**.

- gh-115165 [https://github.com/python/cpython/issues/115165]: Most exceptions are now ignored when attempting to set the `__orig_class__` attribute on objects returned when calling **`typing`** generic aliases (including generic aliases created using **`typing.Annotated`**). Previously only

`AttributeError` was ignored. Patch by Dave Shawley.

- gh-112903 [https://github.com/python/cpython/issues/112903]: Fix «issubclass() arg 1 must be a class» errors in certain cases of multiple inheritance with generic aliases (regression in early 3.13 alpha releases).

- gh-115133 [https://github.com/python/cpython/issues/115133]: Fix tests for `XMLPullParser` with Expat 2.6.0.

- gh-115059 [https://github.com/python/cpython/issues/115059]: **`io.BufferedRandom.read1()`** now flushes the underlying write buffer.

- gh-79382 [https://github.com/python/cpython/issues/79382]: Trailing `**` no longer allows to match files and non-existing paths in recursive `glob()`.

- gh-67837 [https://github.com/python/cpython/issues/67837]: Avoid race conditions in the creation of directories during concurrent extraction in `tarfile` and `zipfile`.

- gh-115060 [https://github.com/python/cpython/issues/115060]: Speed up `pathlib.Path.glob()` by removing redundant regex matching.

- gh-97928 [https://github.com/python/cpython/issues/97928]: Partially revert the behavior of **`tkinter.Text.count()`**. By default it preserves the behavior of older Python versions, except that setting `wantobjects` to 0 no longer has effect. Add a new parameter *return_ints*: specifying `return_ints=True` makes `Text.count()` always returning the single count as an integer instead of a 1-tuple or `None`.

- gh-114628 [https://github.com/python/cpython/issues/114628]: When csv.Error is raised when handling TypeError, do not print the TypeError traceback.

- gh-85984 [https://github.com/python/cpython/issues/85984]: Added `_POSIX_VDISABLE` from C's `<unistd.h>` to `termios`.

- gh-114965 [https://github.com/python/cpython/issues/114965]:

Update bundled pip to 24.0

- gh-114959 [https://github.com/python/cpython/issues/114959]: `tarfile` no longer ignores errors when trying to extract a directory on top of a file.

- gh-114894 [https://github.com/python/cpython/issues/114894]: Add `array.array.clear()`.

- gh-114071 [https://github.com/python/cpython/issues/114071]: Support tuple subclasses using auto() for enum member value.

- gh-109475 [https://github.com/python/cpython/issues/109475]: Fix support of explicit option value «–» in `argparse` (e.g. `--option=--`).

- gh-49766 [https://github.com/python/cpython/issues/49766]: Fix `date`-`datetime` comparison. Now the special comparison methods like `__eq__` and `__lt__` return `NotImplemented` if one of comparands is `date` and other is `datetime` instead of ignoring the time part and the time zone or forcefully return «not equal» or raise `TypeError`. It makes comparison of `date` and `datetime` subclasses more symmetric and allows to change the default behavior by overriding the special comparison methods in subclasses.

- gh-110190 [https://github.com/python/cpython/issues/110190]: Fix ctypes structs with array on Windows ARM64 platform by setting `MAX_STRUCT_SIZE` to 32 in stgdict. Patch by Diego Russo

- gh-114678 [https://github.com/python/cpython/issues/114678]: Ensure that deprecation warning for "N" specifier in `Decimal` format is not raised for cases where "N" appears in other places in the format specifier. Based on patch by Stefan Krah.

- gh-70303 [https://github.com/python/cpython/issues/70303]: Return both files and directories from `pathlib.Path.glob()` if a pattern ends with «`**`». Previously only directories were returned.

- gh-109653 [https://github.com/python/cpython/issues/109653]: Improve import time of `importlib.metadata` and `email.utils`.

- gh-113280 [https://github.com/python/cpython/issues/113280]: Fix a leak of open socket in rare cases when error occurred in `ssl.SSLSocket` creation.

- gh-77749 [https://github.com/python/cpython/issues/77749]: `email.policy.EmailPolicy.fold()` now always encodes non-ASCII characters in headers if `utf8` is false.

- gh-83383 [https://github.com/python/cpython/issues/83383]: Synchronization of the `dbm.dumb` database is now no-op if there was no modification since opening or last synchronization. The directory file for a newly created empty `dbm.dumb` database is now created immediately after opening instead of deferring this until synchronizing or closing.

- gh-91602 [https://github.com/python/cpython/issues/91602]: Add *filter* keyword-only parameter to `sqlite3.Connection.iterdump()` for filtering database objects to dump. Patch by Mariusz Felisiak.

- gh-112451 [https://github.com/python/cpython/issues/112451]: Prohibit subclassing pure-Python `datetime.timezone`. This is consistent with C-extension implementation. Patch by Mariusz Felisiak.

- gh-69893 [https://github.com/python/cpython/issues/69893]: Add the `close()` method for the iterator returned by `xml.etree.ElementTree.iterparse()`.

- gh-109653 [https://github.com/python/cpython/issues/109653]: Reduce the import time of `threading` module by ~50%. Patch by Daniel Hollas.

- gh-114492 [https://github.com/python/cpython/issues/114492]: Make

the result of `termios.tcgetattr()` reproducible on Alpine Linux. Previously it could leave a random garbage in some fields.

- gh-114315 [https://github.com/python/cpython/issues/114315]: Make `threading.Lock` a real class, not a factory function. Add `__new__` to `_thread.lock` type.

- gh-100414 [https://github.com/python/cpython/issues/100414]: Add `dbm.sqlite3` as a backend to `dbm`, and make it the new default `dbm` backend. Patch by Raymond Hettinger and Erlend E. Aasland.

- gh-113267 [https://github.com/python/cpython/issues/113267]: Revert changes in gh-106584 [https://github.com/python/cpython/issues/106584] which made calls of `TestResult` methods `startTest()` and `stopTest()` unbalanced.

- gh-75128 [https://github.com/python/cpython/issues/75128]: Ignore an `OSError` in `asyncio.BaseEventLoop.create_server()` when IPv6 is available but the interface cannot actually support it.

- gh-114423 [https://github.com/python/cpython/issues/114423]: `_DummyThread` entries in `threading._active` are now automatically removed when the related thread dies.

- gh-114257 [https://github.com/python/cpython/issues/114257]: Dismiss the `FileNotFound` error in `ctypes.util.find_library()` and just return `None` on Linux.

- gh-114321 [https://github.com/python/cpython/issues/114321]: Expose more platform specific constants in the `fcntl` module on Linux, macOS, FreeBSD and NetBSD.

- gh-114328 [https://github.com/python/cpython/issues/114328]: The `tty.setcbreak()` and new `tty.cfmakecbreak()` no longer clears the terminal input ICRLF flag. This fixes a regression introduced in 3.12 that no longer matched how OSes define cbreak mode in their `stty(1)` manual pages.

- gh-114281 [https://github.com/python/cpython/issues/114281]: Remove type hints from `Lib/asyncio/staggered.py`. The annotations in the typeshed [https://github.com/python/typeshed] project should be used instead.

- gh-101438 [https://github.com/python/cpython/issues/101438]: Avoid reference cycle in ElementTree.iterparse. The iterator returned by `ElementTree.iterparse` may hold on to a file descriptor. The reference cycle prevented prompt clean-up of the file descriptor if the returned iterator was not exhausted.

- gh-114198 [https://github.com/python/cpython/issues/114198]: The signature for the `__replace__` method on **`dataclasses`** now has the first argument named `self`, rather than `obj`.

- gh-104522 [https://github.com/python/cpython/issues/104522]: **`OSError`** raised when run a subprocess now only has *filename* attribute set to *cwd* if the error was caused by a failed attempt to change the current directory.

- gh-114149 [https://github.com/python/cpython/issues/114149]: Enum: correctly handle tuple subclasses in custom `__new__`.

- gh-83648 [https://github.com/python/cpython/issues/83648]: Support deprecation of options, positional arguments and subcommands in **`argparse`**.

- gh-114087 [https://github.com/python/cpython/issues/114087]: Speed up `dataclasses.asdict` up to 1.35x.

- gh-109534 [https://github.com/python/cpython/issues/109534]: Fix a reference leak in **`asyncio.selector_events.BaseSelectorEventLoop`** when SSL handshakes fail. Patch contributed by Jamie Phan.

- gh-79634 [https://github.com/python/cpython/issues/79634]: Accept path-like objects as patterns in **`pathlib.Path.glob()`** and **`rglob()`**.

- gh-112202 [https://github.com/python/cpython/issues/112202]:

Ensure that a **`asyncio.Condition.notify()`** call does not get lost if the awakened `Task` is simultaneously cancelled or encounters any other error.

- gh-113951 [https://github.com/python/cpython/issues/113951]: Fix the behavior of `tag_unbind()` methods of **`tkinter.Text`** and **`tkinter.Canvas`** classes with three arguments. Previously, `widget.tag_unbind(tag, sequence, funcid)` destroyed the current binding for *sequence*, leaving *sequence* unbound, and deleted the *funcid* command. Now it removes only *funcid* from the binding for *sequence*, keeping other commands, and deletes the *funcid* command. It leaves *sequence* unbound only if *funcid* was the last bound command.

- gh-97959 [https://github.com/python/cpython/issues/97959]: Fix rendering class methods, bound methods, method and function aliases in **`pydoc`**. Class methods no longer have «method of builtins.type instance» note. Corresponding notes are now added for class and unbound methods. Method and function aliases now have references to the module or the class where the origin was defined if it differs from the current. Bound methods are now listed in the static methods section. Methods of builtin classes are now supported as well as methods of Python classes.

- gh-113796 [https://github.com/python/cpython/issues/113796]: Add more validation checks in the **`csv.Dialect`** constructor. **`ValueError`** is now raised if the same character is used in different roles.

- gh-113732 [https://github.com/python/cpython/issues/113732]: Fix support of **`QUOTE_NOTNULL`** and **`QUOTE_STRINGS`** in **`csv.reader()`**.

- gh-113225 [https://github.com/python/cpython/issues/113225]: Speed up **`pathlib.Path.walk()`** by using **`os.DirEntry.path`** where possible.

- gh-89039 [https://github.com/python/cpython/issues/89039]: When replace() method is called on a subclass of datetime, date or time, properly call derived constructor. Previously, only the base class's constructor was called.

Also, make sure to pass non-zero fold values when creating subclasses in various methods. Previously, fold was silently ignored.

- gh-112919 [https://github.com/python/cpython/issues/112919]: Speed-up **`datetime.datetime.replace()`**, **`datetime.date.replace()`** and **`datetime.time.replace()`**.

- gh-59013 [https://github.com/python/cpython/issues/59013]: Set breakpoint on the first executable line of the function, instead of the line of function definition when the user do `break func` using **`pdb`**

- gh-112343 [https://github.com/python/cpython/issues/112343]: Improve handling of pdb convenience variables to avoid replacing string contents.

- gh-112240 [https://github.com/python/cpython/issues/112240]: Add option to calendar module CLI to specify the weekday to start each week. Patch by Steven Ward.

- gh-111741 [https://github.com/python/cpython/issues/111741]: Recognise `image/webp` as a standard format in the **`mimetypes`** module.

- gh-43457 [https://github.com/python/cpython/issues/43457]: Fix the **`tkinter`** widget method **`wm_attributes()`**. It now accepts the attribute name without the minus prefix to get window attributes and allows to specify attributes and values to set as keyword arguments. Add new optional keyword argument *return_python_dict*: calling `w.wm_attributes(return_python_dict=True)` returns the attributes as a dict instead of a tuple. Calling `w.wm_attributes()` now returns a tuple instead of string if *wantobjects* was set to 0.

- gh-82626 [https://github.com/python/cpython/issues/82626]: Many functions now emit a warning if a boolean value is passed as a file descriptor argument.

- gh-111051 [https://github.com/python/cpython/issues/111051]: Added check for file modification during debugging with `pdb`
- gh-110345 [https://github.com/python/cpython/issues/110345]: Show the Tcl/Tk patchlevel (rather than version) in `tkinter._test()`.
- gh-38807 [https://github.com/python/cpython/issues/38807]: Fix race condition in `trace`. Instead of checking if a directory exists and creating it, directly call `os.makedirs()` with the kwarg `exist_ok=True`.
- gh-75705 [https://github.com/python/cpython/issues/75705]: Set unixfrom envelope in `mailbox.mbox` and `mailbox.MMDF`.
- gh-106233 [https://github.com/python/cpython/issues/106233]: Fix stacklevel in `InvalidTZPathWarning` during `zoneinfo` module import.
- gh-105102 [https://github.com/python/cpython/issues/105102]: Allow `ctypes.Union` to be nested in `ctypes.Structure` when the system endianness is the opposite of the classes.
- gh-104282 [https://github.com/python/cpython/issues/104282]: Fix null pointer dereference in `lzma._decode_filter_properties()` due to improper handling of BCJ filters with properties of zero length. Patch by Radislav Chugunov.
- gh-96471 [https://github.com/python/cpython/issues/96471]: Add `queue.Queue` termination with `shutdown()`.
- gh-101599 [https://github.com/python/cpython/issues/101599]: Changed argparse flag options formatting to remove redundancy.
- gh-85984 [https://github.com/python/cpython/issues/85984]: Add POSIX pseudo-terminal functions `os.posix_openpt()`, `os.grantpt()`, `os.unlockpt()`, and `os.ptsname()`.

- gh-102512 [https://github.com/python/cpython/issues/102512]: When `os.fork()` is called from a foreign thread (aka `_DummyThread`), the type of the thread in a child process is changed to `_MainThread`. Also changed its name and daemonic status, it can be now joined.
- gh-88569 [https://github.com/python/cpython/issues/88569]: Add `os.path.isreserved()`, which identifies reserved pathnames such as «NUL», «AUX» and «CON». This function is only available on Windows.

  Deprecate `pathlib.PurePath.is_reserved()`.
- bpo-38364 [https://bugs.python.org/issue?@action=redirect&bpo=38364]: The `inspect` functions `isgeneratorfunction`, `iscoroutinefunction`, `isasyncgenfunction` now support `functools.partialmethod` wrapped functions the same way they support `functools.partial`.

Documentation

- gh-115233 [https://github.com/python/cpython/issues/115233]: Fix an example for `LoggerAdapter` in the Logging Cookbook.
- gh-114123 [https://github.com/python/cpython/issues/114123]: Move the `csv` module docstring to the `csv` module instead of reexporting it from the internal `_csv` module, and remove `__doc__` from `csv.__all__`.

  Move `csv.__version__` to the `csv` module instead of reexporting it from the internal `_csv` module, and remove `__version__` from `csv.__all__`.

Tests

- gh-114099 [https://github.com/python/cpython/issues/114099]: Added test exclusions required to run the test suite on iOS.
- gh-105089 [https://github.com/python/cpython/issues/105089]: Fix `test.test_zipfile.test_core.TestWithDirectory.test_` test in AIX by doing a bitwise AND of 0xFFFF on mode , so

that it will be in sync with `zinfo.external_attr`

Build

- gh-115167 [https://github.com/python/cpython/issues/115167]: Avoid vendoring `vcruntime140_threads.dll` when building with Visual Studio 2022 version 17.8.
- gh-113632 [https://github.com/python/cpython/issues/113632]: Promote WASI to a tier 2 platform and drop Emscripten from tier 3 in configure.ac.
- gh-114099 [https://github.com/python/cpython/issues/114099]: configure and Makefile were refactored to accommodate framework builds on Apple platforms other than macOS.
- gh-114875 [https://github.com/python/cpython/issues/114875]: Add **`getgrent()`** as a prerequisite for building the **`grp`** module.

Windows

- gh-115049 [https://github.com/python/cpython/issues/115049]: Fixes `py.exe` launcher failing when run as users without user profiles.
- gh-115009 [https://github.com/python/cpython/issues/115009]: Update Windows installer to use SQLite 3.45.1.
- gh-109991 [https://github.com/python/cpython/issues/109991]: Update Windows build to use OpenSSL 3.0.13.
- gh-111239 [https://github.com/python/cpython/issues/111239]: Update Windows builds to use zlib v1.3.1.
- gh-100107 [https://github.com/python/cpython/issues/100107]: The `py.exe` launcher will no longer attempt to run the Microsoft Store redirector when launching a script containing a `/usr/bin/env` shebang
- gh-112984 [https://github.com/python/cpython/issues/112984]: Adds free-threaded binaries to Windows installer as an optional component.
- gh-89240 [https://github.com/python/cpython/issues/89240]: Allows **`multiprocessing`** to create pools of greater than 62 processes.

macOS

- gh-115009 [https://github.com/python/cpython/issues/115009]: Update macOS installer to use SQLite 3.45.1.
- gh-109991 [https://github.com/python/cpython/issues/109991]: Update macOS installer to use OpenSSL 3.0.13.
- gh-114490 [https://github.com/python/cpython/issues/114490]: Add Mach-O linkage support for **`platform.architecture()`**.
- gh-87804 [https://github.com/python/cpython/issues/87804]: On macOS the result of `os.statvfs` and `os.fstatvfs` now correctly report the size of very large disks, in previous versions the reported number of blocks was wrong for disks with at least 2**32 blocks.

IDLE

- gh-96905 [https://github.com/python/cpython/issues/96905]: In idlelib code, stop redefining built-ins “dict” and “object”.
- gh-103820 [https://github.com/python/cpython/issues/103820]: Revise IDLE bindings so that events from mouse button 4/5 on non-X11 windowing systems (i.e. Win32 and Aqua) are not mistaken for scrolling.

Tools/Demos

- gh-113516 [https://github.com/python/cpython/issues/113516]: Don’t set `LDSHARED` when building for WASI.
- gh-109991 [https://github.com/python/cpython/issues/109991]: Update GitHub CI workflows to use OpenSSL 3.0.13 and multissltests to use 1.1.1w, 3.0.13, 3.1.5, and 3.2.1.
- gh-115015 [https://github.com/python/cpython/issues/115015]: Fix a bug in Argument Clinic that generated incorrect code for methods with no parameters that use the METH_METHOD | METH_FASTCALL | METH_KEYWORDS calling convention. Only the positional parameter count was checked; any keyword argument passed would be silently accepted.

C API

- gh-111140 [https://github.com/python/cpython/issues/111140]: Adds **`PyLong_AsNativeBytes()`**, **`PyLong_FromNativeBytes()`** and

`PyLong_FromUnsignedNativeBytes()` functions.

- gh-114685 [https://github.com/python/cpython/issues/114685]: `PyBuffer_FillInfo()` now raises a `SystemError` if called with `PyBUF_READ` or `PyBUF_WRITE` as flags. These flags should only be used with the `PyMemoryView_*` C API.

- gh-114685 [https://github.com/python/cpython/issues/114685]: `PyObject_GetBuffer()` now raises a `SystemError` if called with `PyBUF_READ` or `PyBUF_WRITE` as flags. These flags should only be used with the `PyMemoryView_*` C API.

- gh-114626 [https://github.com/python/cpython/issues/114626]: Add `PyCFunctionFast` and `PyCFunctionFastWithKeywords` typedefs (identical to the existing `_PyCFunctionFast` and `_PyCFunctionFastWithKeywords` typedefs, just without a leading `_` prefix).

- gh-114329 [https://github.com/python/cpython/issues/114329]: Add `PyList_GetItemRef()`, which is similar to `PyList_GetItem()` but returns a strong reference instead of a borrowed reference.

- gh-110850 [https://github.com/python/cpython/issues/110850]: Add PyTime C API:

  ○ `PyTime_t` type.
  ○ `PyTime_MIN` and `PyTime_MAX` constants.
  ○ `PyTime_AsSecondsDouble()`, `PyTime_Monotonic()`, `PyTime_PerfCounter()`, and `PyTime_Time()` functions.

  Patch by Victor Stinner.

- gh-112066 [https://github.com/python/cpython/issues/112066]: Add `PyDict_SetDefaultRef()`: insert a key and value into a dictionary if the key is not already present. This is similar to `dict.setdefault()`, but returns an integer value indicating if the key was already present. It is also similar to `PyDict_SetDefault()`, but returns a strong reference instead of a borrowed reference.

Python 3.13.0 alpha 3

Release date: 2024-01-17

Security

- gh-113659 [https://github.com/python/cpython/issues/113659]: Skip `.pth` files with names starting with a dot or hidden file attribute.
- gh-112302 [https://github.com/python/cpython/issues/112302]: Created a Software Bill-of-Materials document and tooling for tracking dependencies.

Core and Builtins

- gh-107901 [https://github.com/python/cpython/issues/107901]: Compiler duplicates basic blocks that have an eval breaker check, no line number, and multiple predecessors.

- gh-107901 [https://github.com/python/cpython/issues/107901]: A jump leaving an exception handler back to normal code no longer checks the eval breaker.

- gh-113655 [https://github.com/python/cpython/issues/113655]: Set the C recursion limit to 4000 on Windows, and 10000 on Linux/OSX. This seems to be near the sweet spot to maintain safety, but not compromise backwards compatibility.

- gh-113710 [https://github.com/python/cpython/issues/113710]: Add typed stack effects to the interpreter DSL, along with various instruction annotations.

- gh-77046 [https://github.com/python/cpython/issues/77046]: On Windows, file descriptors wrapping Windows handles are now created non inheritable by default (**PEP 446** [https://peps.python.org/pep-0446/]). Patch by Zackery Spytz and Victor Stinner.

- gh-113853 [https://github.com/python/cpython/issues/113853]: Guarantee that all executors make progress. This then guarantees that tier 2 execution always makes progress.

- gh-113753 [https://github.com/python/cpython/issues/113753]: Fix an issue where the finalizer of `PyAsyncGenASend` objects might not be called if they were allocated from a free list.

- gh-107901 [https://github.com/python/cpython/issues/107901]: Compiler changed so that synthetic jumps which are not at loop end no longer check the eval breaker.

- gh-113703 [https://github.com/python/cpython/issues/113703]: Fix a regression in the **`codeop`** module that was causing it to incorrectly identify incomplete f-strings. Patch by Pablo Galindo

- gh-89811 [https://github.com/python/cpython/issues/89811]: Check for a valid `tp_version_tag` before performing bytecode specializations that rely on this value being usable.

- gh-111488 [https://github.com/python/cpython/issues/111488]: Changed error message in case of no “in” keyword after “for” in list comprehensions

- gh-113657 [https://github.com/python/cpython/issues/113657]: Fix an issue that caused important instruction pointer updates to be optimized out of tier two traces.

- gh-113603 [https://github.com/python/cpython/issues/113603]: Fixed bug where a redundant NOP is not removed, causing an assertion to fail in the compiler in debug mode.

- gh-113602 [https://github.com/python/cpython/issues/113602]: Fix an error that was causing the parser to try to overwrite existing errors and crashing in the process. Patch by Pablo Galindo

- gh-113486 [https://github.com/python/cpython/issues/113486]: No longer issue spurious `PY_UNWIND` events for optimized calls to classes.

- gh-113297 [https://github.com/python/cpython/issues/113297]: Fix

segfault in the compiler on with statement with 19 context managers.

- gh-113212 [https://github.com/python/cpython/issues/113212]: Improve `super` error messages.
- gh-111375 [https://github.com/python/cpython/issues/111375]: Only use `NULL` in the exception stack to indicate an exception was handled. Patch by Carey Metcalfe.
- gh-112215 [https://github.com/python/cpython/issues/112215]: Increase the C recursion limit by a factor of 3 for non-debug builds, except for webassembly and s390 platforms which are unchanged. This mitigates some regressions in 3.12 with deep recursion mixing builtin (C) and Python code.
- gh-113054 [https://github.com/python/cpython/issues/113054]: Fixed bug where a redundant NOP is not removed, causing an assertion to fail in the compiler in debug mode.
- gh-106905 [https://github.com/python/cpython/issues/106905]: Use per AST-parser state rather than global state to track recursion depth within the AST parser to prevent potential race condition due to simultaneous parsing.

  The issue primarily showed up in 3.11 by multithreaded users of `ast.parse()`. In 3.12 a change to when garbage collection can be triggered prevented the race condition from occurring.
- gh-108866 [https://github.com/python/cpython/issues/108866]: Change the API and contract of `_PyExecutorObject` to return the next_instr pointer, instead of the frame, and to always execute at least one instruction.
- gh-90350 [https://github.com/python/cpython/issues/90350]: Optimize builtin functions `min()` and `max()`.
- gh-112943 [https://github.com/python/cpython/issues/112943]: Correctly compute end column offsets for multiline tokens in the `tokenize` module. Patch by Pablo Galindo

- gh-112125 [https://github.com/python/cpython/issues/112125]: Fix `None.__ne__(None)` returning `NotImplemented` instead of `False`.

- gh-74616 [https://github.com/python/cpython/issues/74616]: **`input()`** now raises a ValueError when output on the terminal if the prompt contains embedded null characters instead of silently truncating it.

- gh-112716 [https://github.com/python/cpython/issues/112716]: Fix SystemError in the `import` statement and in `__reduce__()` methods of builtin types when `__builtins__` is not a dict.

- gh-112730 [https://github.com/python/cpython/issues/112730]: Use color to highlight error locations in tracebacks. Patch by Pablo Galindo

- gh-112625 [https://github.com/python/cpython/issues/112625]: Fixes a bug where a bytearray object could be cleared while iterating over an argument in the `bytearray.join()` method that could result in reading memory after it was freed.

- gh-112660 [https://github.com/python/cpython/issues/112660]: Do not clear unexpected errors during formatting error messages for ImportError and AttributeError for modules.

- gh-105967 [https://github.com/python/cpython/issues/105967]: Workaround a bug in Apple's macOS platform zlib library where **`zlib.crc32()`** and **`binascii.crc32()`** could produce incorrect results on multi-gigabyte inputs. Including when using **`zipfile`** on zips containing large data.

- gh-95754 [https://github.com/python/cpython/issues/95754]: Provide a better error message when accessing invalid attributes on partially initialized modules. The origin of the module being accessed is now included in the message to help with the common issue of shadowing other modules.

- gh-112217 [https://github.com/python/cpython/issues/112217]: Add

check for the type of `__cause__` returned from calling the type `T` in `raise from T`.

- gh-111058 [https://github.com/python/cpython/issues/111058]: Change coro.cr_frame/gen.gi_frame to return `None` after the coroutine/generator has been closed. This fixes a bug where **`getcoroutinestate()`** and **`getgeneratorstate()`** return the wrong state for a closed coroutine/generator.

- gh-112388 [https://github.com/python/cpython/issues/112388]: Fix an error that was causing the parser to try to overwrite tokenizer errors. Patch by pablo Galindo

- gh-112387 [https://github.com/python/cpython/issues/112387]: Fix error positions for decoded strings with backwards tokenize errors. Patch by Pablo Galindo

- gh-99606 [https://github.com/python/cpython/issues/99606]: Make code generated for an empty f-string identical to the code of an empty normal string.

- gh-112367 [https://github.com/python/cpython/issues/112367]: Avoid undefined behaviour when using the perf trampolines by not freeing the code arenas until shutdown. Patch by Pablo Galindo

- gh-112320 [https://github.com/python/cpython/issues/112320]: The Tier 2 translator now tracks the confidence level for staying «on trace» (i.e. not exiting back to the Tier 1 interpreter) for branch instructions based on the number of bits set in the branch «counter». Trace translation ends when the confidence drops below 1/3rd.

- gh-109598 [https://github.com/python/cpython/issues/109598]: **`PyComplex_RealAsDouble()`**/ **`PyComplex_ImagAsDouble()`** now tries to convert an object to a **`complex`** instance using its `__complex__()` method before falling back to the `__float__()` method. Patch by Sergey B Kirpichev.

- gh-94606 [https://github.com/python/cpython/issues/94606]: Fix

UnicodeEncodeError when `email.message.get_payload()` reads a message with a Unicode surrogate character and the message content is not well-formed for surrogateescape encoding. Patch by Sidney Markowitz.

- bpo-21861 [https://bugs.python.org/issue?@action=redirect&bpo=21861]: Use the object's actual class name in `_io.FileIO.__repr__()`, `_io._WindowsConsoleIO()` and `_io.TextIOWrapper.__repr__()`, to make these methods subclass friendly.

- bpo-45369 [https://bugs.python.org/issue?@action=redirect&bpo=45369]: Remove LibreSSL workarounds as per **PEP 644** [https://peps.python.org/pep-0644/].

- bpo-34392 [https://bugs.python.org/issue?@action=redirect&bpo=34392]: Added `sys._is_interned()`.

Library

- gh-114077 [https://github.com/python/cpython/issues/114077]: Fix possible `OverflowError` in `socket.socket.sendfile()` when pass *count* larger than 2 GiB on 32-bit platform.

- gh-111803 [https://github.com/python/cpython/issues/111803]: `plistlib` now supports loading more deeply nested lists in binary format.

- gh-114014 [https://github.com/python/cpython/issues/114014]: Fixed a bug in `fractions.Fraction` where an invalid string using `d` in the decimals part creates a different error compared to other invalid letters/characters. Patch by Jeremiah Gabriel Pascual.

- gh-108364 [https://github.com/python/cpython/issues/108364]: `sqlite3.Connection.iterdump()` now ensures that foreign key support is disabled before dumping the database schema, if there is any foreign key violation. Patch by Erlend

E. Aasland and Mariusz Felisiak.

- gh-113971 [https://github.com/python/cpython/issues/113971]: The **`zipfile.ZipInfo`** previously protected `._compresslevel` attribute has been made public as `.compress_level` with the old `_compresslevel` name remaining available as a property to retain compatibility.

- gh-113877 [https://github.com/python/cpython/issues/113877]: Fix **`tkinter`** method `winfo_pathname()` on 64-bit Windows.

- gh-113868 [https://github.com/python/cpython/issues/113868]: Added **`mmap.MAP_NORESERVE`**, **`mmap.MAP_NOEXTEND`**, **`mmap.MAP_HASSEMAPHORE`**, **`mmap.MAP_NOCACHE`**, **`mmap.MAP_JIT`**, **`mmap.MAP_RESILIENT_CODESIGN`**, **`mmap.MAP_RESILIENT_MEDIA`**, **`mmap.MAP_32BIT`**, **`mmap.MAP_TRANSLATED_ALLOW_EXECUTE`**, **`mmap.MAP_UNIX03`** and **`mmap.MAP_TPRO`**. All of them are `mmap(2)` flags on macOS.

- gh-113848 [https://github.com/python/cpython/issues/113848]: **`asyncio.TaskGroup()`** and **`asyncio.timeout()`** context managers now handle **`CancelledError`** subclasses as well as exact **`CancelledError`**.

- gh-113661 [https://github.com/python/cpython/issues/113661]: unittest runner: Don’t exit 5 if tests were skipped. The intention of exiting 5 was to detect issues where the test suite wasn’t discovered at all. If we skipped tests, it was correctly discovered.

- gh-96037 [https://github.com/python/cpython/issues/96037]: Insert **`TimeoutError`** in the context of the exception that was raised during exiting an expired **`asyncio.timeout()`** block.

- gh-113781 [https://github.com/python/cpython/issues/113781]: Silence unraisable AttributeError when warnings are emitted during Python finalization.

- gh-113238 [https://github.com/python/cpython/issues/113238]: Add

`Anchor` to `importlib.resources` (in order for the code to comply with the documentation)

- gh-111693 [https://github.com/python/cpython/issues/111693]: **`asyncio.Condition.wait()`** now re-raises the same **`CancelledError`** instance that may have caused it to be interrupted. Fixed race condition in **`asyncio.Semaphore.acquire()`** when interrupted with a **`CancelledError`**.

- gh-113791 [https://github.com/python/cpython/issues/113791]: Add `CLOCK_MONOTONIC_RAW_APPROX` and `CLOCK_UPTIME_RAW_APPROX` to **`time`** on macOS. These are clocks available on macOS 10.12 or later.

- gh-112932 [https://github.com/python/cpython/issues/112932]: Restore the ability for **`zipfile`** to `extractall` from zip files with a «/» directory entry in them as is commonly added to zips by some wiki or bug tracker data exporters.

- gh-113568 [https://github.com/python/cpython/issues/113568]: Raise deprecation warnings from **`pathlib.PurePath`** and not its private base class `PurePathBase`.

- gh-113594 [https://github.com/python/cpython/issues/113594]: Fix **`UnicodeEncodeError`** in **`email`** when re-fold lines that contain unknown-8bit encoded part followed by non-unknown-8bit encoded part.

- gh-113538 [https://github.com/python/cpython/issues/113538]: In **`asyncio.StreamReaderProtocol.connection_made()`**, there is callback that logs an error if the task wrapping the «connected callback» fails. This callback would itself fail if the task was cancelled. Prevent this by checking whether the task was cancelled first. If so, close the transport but don't log an error.

- gh-113626 [https://github.com/python/cpython/issues/113626]: Add support for the *allow_code* argument in the **`marshal`** module. Passing `allow_code=False` prevents serialization and de-serialization of code objects which is incompatible between

Python versions.

- gh-85567 [https://github.com/python/cpython/issues/85567]: Fix resource warnings for unclosed files in `pickle` and `pickletools` command line interfaces.

- gh-113537 [https://github.com/python/cpython/issues/113537]: Support loads `str` in `plistlib.loads()`.

- gh-89850 [https://github.com/python/cpython/issues/89850]: Add default implementations of `pickle.Pickler.persistent_id()` and `pickle.Unpickler.persistent_load()` methods in the C implementation. Calling `super().persistent_id()` and `super().persistent_load()` in subclasses of the C implementation of `pickle.Pickler` and `pickle.Unpickler` classes no longer causes infinite recursion.

- gh-113569 [https://github.com/python/cpython/issues/113569]: Indicate if there were no actual calls in unittest `assert_has_calls()` failure.

- gh-101225 [https://github.com/python/cpython/issues/101225]: Increase the backlog for `multiprocessing.connection.Listener` objects created by **`multiprocessing.manager`** and **`multiprocessing.resource_sharer`** to significantly reduce the risk of getting a connection refused error when creating a `multiprocessing.connection.Connection` to them.

- gh-113568 [https://github.com/python/cpython/issues/113568]: Raise audit events from `pathlib.Path` and not its private base class `PathBase`.

- gh-113543 [https://github.com/python/cpython/issues/113543]: Make sure that `webbrowser.MacOSXOSAScript` sends `webbrowser.open` audit event.

- gh-113028 [https://github.com/python/cpython/issues/113028]: When

a second reference to a string appears in the input to **`pickle`**, and the Python implementation is in use, we are guaranteed that a single copy gets pickled and a single object is shared when reloaded. Previously, in protocol 0, when a string contained certain characters (e.g. newline) it resulted in duplicate objects.

- gh-113421 [https://github.com/python/cpython/issues/113421]: Fix multiprocessing logger for `%(filename)s`.

- gh-111784 [https://github.com/python/cpython/issues/111784]: Fix segfaults in the `_elementtree` module. Fix first segfault during deallocation of `_elementtree.XMLParser` instances by keeping strong reference to `pyexpat` module in module state for capsule lifetime. Fix second segfault which happens in the same deallocation process by keeping strong reference to `_elementtree` module in `XMLParser` structure for `_elementtree` module lifetime.

- gh-113407 [https://github.com/python/cpython/issues/113407]: Fix import of **`unittest.mock`** when CPython is built without docstrings.

- gh-113320 [https://github.com/python/cpython/issues/113320]: Fix regression in Python 3.12 where **`Protocol`** classes that were not marked as **`runtime-checkable`** would be unnecessarily introspected, potentially causing exceptions to be raised if the protocol had problematic members. Patch by Alex Waygood.

- gh-53502 [https://github.com/python/cpython/issues/53502]: Add a new option `aware_datetime` in **`plistlib`** to loads or dumps aware datetime.

- gh-113358 [https://github.com/python/cpython/issues/113358]: Fix rendering tracebacks with exceptions with a broken _getattr_

- gh-113214 [https://github.com/python/cpython/issues/113214]: Fix an `AttributeError` during asyncio SSL protocol aborts in SSL-over-SSL scenarios.

- gh-113246 [https://github.com/python/cpython/issues/113246]: Update bundled pip to 23.3.2.

- gh-87264 [https://github.com/python/cpython/issues/87264]: Fixed tarfile list() method to show file type.

- gh-112182 [https://github.com/python/cpython/issues/112182]: **`asyncio.futures.Future.set_exception()`** now transforms **`StopIteration`** into **`RuntimeError`** instead of hanging or other misbehavior. Patch contributed by Jamie Phan.

- gh-113225 [https://github.com/python/cpython/issues/113225]: Speed up **`pathlib.Path.glob()`** by using **`os.DirEntry.path`** where possible.

- gh-113149 [https://github.com/python/cpython/issues/113149]: Improve error message when a JSON array or object contains a trailing comma. Patch by Carson Radtke.

- gh-113117 [https://github.com/python/cpython/issues/113117]: The **`subprocess`** module can now use the **`os.posix_spawn()`** function with `close_fds=True` on platforms where `posix_spawn_file_actions_addclosefrom_np` is available. Patch by Jakub Kulik.

- gh-113199 [https://github.com/python/cpython/issues/113199]: Make `http.client.HTTPResponse.read1` and `http.client.HTTPResponse.readline` close IO after reading all data when content length is known. Patch by Illia Volochii.

- gh-113191 [https://github.com/python/cpython/issues/113191]: Add support of **`os.fchmod()`** and a file descriptor in **`os.chmod()`** on Windows.

- gh-113188 [https://github.com/python/cpython/issues/113188]: Fix **`shutil.copymode()`** and **`shutil.copystat()`** on Windows. Previously they worked differently if *dst* is a symbolic link: they modified the permission bits of *dst* itself rather than the file it points to if *follow_symlinks* is true or *src* is not a symbolic link, and did not modify the permission bits if *follow_symlinks* is false and *src* is a symbolic link.

- gh-113119 [https://github.com/python/cpython/issues/113119]: **`os.posix_spawn()`** now accepts `env=None`, which makes the newly spawned process use the current process environment. Patch by Jakub Kulik.
- gh-113202 [https://github.com/python/cpython/issues/113202]: Add a `strict` option to `batched()` in the `itertools` module.
- gh-61648 [https://github.com/python/cpython/issues/61648]: Detect line numbers of properties in doctests.
- gh-113175 [https://github.com/python/cpython/issues/113175]: Sync with importlib_metadata 7.0, including improved type annotations, fixed issue with symlinked packages in `package_distributions`, added `EntryPoints.__repr__`, introduced the `diagnose` script, added `Distribution.origin` property, and removed deprecated `EntryPoint` access by numeric index (tuple behavior).
- gh-59616 [https://github.com/python/cpython/issues/59616]: Add support of **`os.lchmod()`** and the *follow_symlinks* argument in **`os.chmod()`** on Windows. Note that the default value of *follow_symlinks* in **`os.lchmod()`** is `False` on Windows.
- gh-112559 [https://github.com/python/cpython/issues/112559]: **`signal.signal()`** and **`signal.getsignal()`** no longer call `repr` on callable handlers. **`asyncio.run()`** and **`asyncio.Runner.run()`** no longer call `repr` on the task results. Patch by Yilei Yang.
- gh-112962 [https://github.com/python/cpython/issues/112962]: **`dis`** module functions add cache information to the **`Instruction`** instance rather than creating fake **`Instruction`** instances to represent the cache entries.
- gh-112989 [https://github.com/python/cpython/issues/112989]: Reduce overhead to connect sockets with **`asyncio`**

SelectorEventLoop.

- gh-112970 [https://github.com/python/cpython/issues/112970]: Use **`closefrom()`** on Linux where available (e.g. glibc-2.34), rather than only FreeBSD.
- gh-110190 [https://github.com/python/cpython/issues/110190]: Fix ctypes structs with array on PPC64LE platform by setting `MAX_STRUCT_SIZE` to 64 in stgdict. Patch by Diego Russo.
- gh-112540 [https://github.com/python/cpython/issues/112540]: The statistics.geometric_mean() function now returns zero for datasets containing a zero. Formerly, it would raise an exception.
- gh-87286 [https://github.com/python/cpython/issues/87286]: Added **`LOG_FTP`**, **`LOG_NETINFO`**, **`LOG_REMOTEAUTH`**, **`LOG_INSTALL`**, **`LOG_RAS`**, and **`LOG_LAUNCHD`** tot the **`syslog`** module, all of them constants on used on macOS.
- gh-112800 [https://github.com/python/cpython/issues/112800]: Fix **`asyncio`** `SubprocessTransport.close()` not to throw `PermissionError` when used with setuid executables.
- gh-51944 [https://github.com/python/cpython/issues/51944]: Add the following constants to the **`termios`** module. These values are present in macOS system headers: `ALTWERASE`, `B14400`, `B28800`, `B7200`, `B76800`, `CCAR_OFLOW`, `CCTS_OFLOW`, `CDSR_OFLOW`, `CDTR_IFLOW`, `CIGNORE`, `CRTS_IFLOW`, `EXTPROC`, `IUTF8`, `MDMBUF`, `NL2`, `NL3`, `NOKERNINFO`, `ONOEOT`, `OXTABS`, `VDSUSP`, `VSTATUS`.
- gh-79325 [https://github.com/python/cpython/issues/79325]: Fix an infinite recursion error in **`tempfile.TemporaryDirectory()`** cleanup on Windows.
- gh-94692 [https://github.com/python/cpython/issues/94692]: **`shutil.rmtree()`** now only catches OSError exceptions. Previously a symlink attack resistant version of `shutil.rmtree()` could ignore or pass to the error handler arbitrary exception when invalid arguments were provided.

- gh-112736 [https://github.com/python/cpython/issues/112736]: The use of del-safe symbols in `subprocess` was refactored to allow for use in cross-platform build environments.

- gh-112727 [https://github.com/python/cpython/issues/112727]: Speed up **`pathlib.Path.absolute()`**. Patch by Barney Gale.

- gh-74690 [https://github.com/python/cpython/issues/74690]: Speedup **`issubclass()`** checks against simple **`runtime-checkable protocols`** by around 6%. Patch by Alex Waygood.

- gh-74690 [https://github.com/python/cpython/issues/74690]: Speedup **`isinstance()`** checks by roughly 20% for **`runtime-checkable protocols`** that only have one callable member. Speedup **`issubclass()`** checks for these protocols by roughly 10%. Patch by Alex Waygood.

- gh-112645 [https://github.com/python/cpython/issues/112645]: Remove deprecation error on passing `onerror` to **`shutil.rmtree()`**.

- gh-112640 [https://github.com/python/cpython/issues/112640]: Add `kwdefaults` parameter to **`types.FunctionType`** to set default keyword argument values.

- gh-112622 [https://github.com/python/cpython/issues/112622]: Ensure `name` parameter is passed to event loop in **`asyncio.create_task()`**.

- gh-112618 [https://github.com/python/cpython/issues/112618]: Fix a caching bug relating to **`typing.Annotated`**. `Annotated[str, True]` is no longer identical to `Annotated[str, 1]`.

- gh-112334 [https://github.com/python/cpython/issues/112334]: Fixed a performance regression in 3.12’s **`subprocess`** on Linux where it would no longer use the fast-path `vfork()` system call when it could have due to a logic bug, instead falling back to the safe but slower `fork()`.

Also fixed a second 3.12.0 potential security bug. If a value of `extra_groups=[]` was passed to **`subprocess.Popen`** or related APIs, the underlying `setgroups(0, NULL)` system call to clear the groups list would not be made in the child process prior to `exec()`.

This was identified via code inspection in the process of fixing the first bug.

- gh-110190 [https://github.com/python/cpython/issues/110190]: Fix ctypes structs with array on Arm platform by setting `MAX_STRUCT_SIZE` to 32 in stgdict. Patch by Diego Russo.

- gh-81194 [https://github.com/python/cpython/issues/81194]: Fix a crash in **`socket.if_indextoname()`** with specific value (UINT_MAX). Fix an integer overflow in **`socket.if_indextoname()`** on 64-bit non-Windows platforms.

- gh-112578 [https://github.com/python/cpython/issues/112578]: Fix a spurious **`RuntimeWarning`** when executing the **`zipfile`** module.

- gh-112516 [https://github.com/python/cpython/issues/112516]: Update the bundled copy of pip to version 23.3.1.

- gh-112510 [https://github.com/python/cpython/issues/112510]: Add **`readline.backend`** for the backend readline uses (`editline` or `readline`)

- gh-112328 [https://github.com/python/cpython/issues/112328]: [Enum] Make `EnumDict`, `EnumDict.member_names`, `EnumType._add_alias_` and `EnumType._add_value_alias_` public.

- gh-112509 [https://github.com/python/cpython/issues/112509]: Fix edge cases that could cause a key to be present in both the `__required_keys__` and `__optional_keys__` attributes of a **`typing.TypedDict`**. Patch by Jelle Zijlstra.

- gh-101336 [https://github.com/python/cpython/issues/101336]: Add

`keep_alive` keyword parameter for **`AbstractEventLoop.create_server()`** and **`BaseEventLoop.create_server()`**.

- gh-63284 [https://github.com/python/cpython/issues/63284]: Added support for TLS-PSK (pre-shared key) mode to the **`ssl`** module.

- gh-112414 [https://github.com/python/cpython/issues/112414]: Fix regression in Python 3.12 where calling **`repr()`** on a module that had been imported using a custom loader could fail with **`AttributeError`**. Patch by Alex Waygood.

- gh-112358 [https://github.com/python/cpython/issues/112358]: Revert change to **`struct.Struct`** initialization that broke some cases of subclassing.

- gh-112405 [https://github.com/python/cpython/issues/112405]: Optimize **`pathlib.PurePath.relative_to()`**. Patch by Alex Waygood.

- gh-94722 [https://github.com/python/cpython/issues/94722]: Fix bug where comparison between instances of **`DocTest`** fails if one of them has `None` as its lineno.

- gh-112361 [https://github.com/python/cpython/issues/112361]: Speed up a small handful of **`pathlib`** methods by removing some temporary objects.

- gh-112345 [https://github.com/python/cpython/issues/112345]: Improve error message when trying to call **`issubclass()`** against a **`typing.Protocol`** that has non-method members. Patch by Randolf Scholz.

- gh-112137 [https://github.com/python/cpython/issues/112137]: Change **`dis`** output to display no-lineno as «–» instead of «None».

- gh-112332 [https://github.com/python/cpython/issues/112332]: Deprecate the `exc_type` field of **`traceback.TracebackException`**. Add `exc_type_str` to replace it.

- gh-81620 [https://github.com/python/cpython/issues/81620]: Add extra tests for `random.binomialvariate()`
- gh-112292 [https://github.com/python/cpython/issues/112292]: Fix a crash in `readline` when imported from a sub interpreter. Patch by Anthony Shaw
- gh-77621 [https://github.com/python/cpython/issues/77621]: Slightly improve the import time of the `pathlib` module by deferring some imports. Patch by Barney Gale.
- gh-112137 [https://github.com/python/cpython/issues/112137]: Change `dis` output to display logical labels for jump targets instead of offsets.
- gh-112139 [https://github.com/python/cpython/issues/112139]: Add `Signature.format()` to format signatures to string with extra options. And use it in `pydoc` to render more readable signatures that have new lines between parameters.
- gh-112105 [https://github.com/python/cpython/issues/112105]: Make `readline.set_completer_delims()` work with libedit
- gh-106922 [https://github.com/python/cpython/issues/106922]: Display multiple lines with `traceback` when errors span multiple lines.
- gh-111874 [https://github.com/python/cpython/issues/111874]: When creating a `typing.NamedTuple` class, ensure `__set_name__()` is called on all objects that define `__set_name__` and exist in the values of the `NamedTuple` class's class dictionary. Patch by Alex Waygood.
- gh-68166 [https://github.com/python/cpython/issues/68166]: Add support of the «vsapi» element type in `tkinter.ttk.Style.element_create()`.
- gh-110275 [https://github.com/python/cpython/issues/110275]: Named tuple's methods `_replace()` and `__replace__()` now raise TypeError instead of ValueError for invalid

keyword arguments.

- gh-99367 [https://github.com/python/cpython/issues/99367]: Do not mangle `sys.path[0]` in **`pdb`** if safe_path is set
- gh-111615 [https://github.com/python/cpython/issues/111615]: Fix a regression caused by a fix to gh-93162 [https://github.com/python/cpython/issues/93162] whereby you couldn't configure a **`QueueHandler`** without specifying handlers.
- gh-75666 [https://github.com/python/cpython/issues/75666]: Fix the behavior of **`tkinter`** widget's `unbind()` method with two arguments. Previously, `widget.unbind(sequence, funcid)` destroyed the current binding for *sequence*, leaving *sequence* unbound, and deleted the *funcid* command. Now it removes only *funcid* from the binding for *sequence*, keeping other commands, and deletes the *funcid* command. It leaves *sequence* unbound only if *funcid* was the last bound command.
- gh-67790 [https://github.com/python/cpython/issues/67790]: Implement basic formatting support (minimum width, alignment, fill) for **`fractions.Fraction`**.
- gh-111049 [https://github.com/python/cpython/issues/111049]: Fix crash during garbage collection of the **`io.BytesIO`** buffer object.
- gh-102980 [https://github.com/python/cpython/issues/102980]: Redirect the output of `interact` command of **`pdb`** to the same channel as the debugger. Add tests and improve docs.
- gh-102988 [https://github.com/python/cpython/issues/102988]: **`email.utils.getaddresses()`** and **`email.utils.parseaddr()`** now return `('', '')` 2-tuples in more situations where invalid email addresses are encountered instead of potentially inaccurate values. Add optional *strict* parameter to these two functions: use `strict=False` to get the old behavior, accept malformed inputs. `getattr(email.utils, 'supports_strict_parsing', False)` can be use to check if the *strict* parameter is available. Patch by Thomas

Dwyer and Victor Stinner to improve the **CVE 2023-27043** [https://www.cve.org/CVERecord?id=CVE-2023-27043] fix.

- gh-52161 [https://github.com/python/cpython/issues/52161]: **`cmd.Cmd.do_help()`** now cleans docstrings with **`inspect.cleandoc()`** before writing them. Patch by Filip Łapkiewicz.
- gh-82300 [https://github.com/python/cpython/issues/82300]: Add `track` parameter to **`multiprocessing.shared_memory.SharedMemory`** that allows using shared memory blocks without having to register with the POSIX resource tracker that automatically releases them upon process exit.
- gh-110109 [https://github.com/python/cpython/issues/110109]: Add private `pathlib._PurePathBase` class: a base class for **`pathlib.PurePath`** that omits certain magic methods. It may be made public (along with `_PathBase`) in future.
- gh-109858 [https://github.com/python/cpython/issues/109858]: Protect **`zipfile`** from «quoted-overlap» zipbomb. It now raises BadZipFile when try to read an entry that overlaps with other entry or central directory.
- gh-109786 [https://github.com/python/cpython/issues/109786]: Fix possible reference leaks and crash when re-enter the `__next__()` method of **`itertools.pairwise`**.
- gh-91539 [https://github.com/python/cpython/issues/91539]: Small (10 - 20%) and trivial performance improvement of **`urllib.request.getproxies_environment()`**, typically useful when there are many environment variables to go over.
- gh-103363 [https://github.com/python/cpython/issues/103363]: Add *follow_symlinks* keyword-only argument to **`pathlib.Path.owner()`** and **`group()`**, defaulting to `True`.
- gh-102130 [https://github.com/python/cpython/issues/102130]:

Support tab completion in `cmd` for `editline`.

- gh-99437 [https://github.com/python/cpython/issues/99437]: `runpy.run_path()` now decodes path-like objects, making sure _file_ and sys.argv[0] of the module being run are always strings.

- gh-104003 [https://github.com/python/cpython/issues/104003]: Add `warnings.deprecated()`, a decorator to mark deprecated functions to static type checkers and to warn on usage of deprecated classes and functions. See **PEP 702** [https://peps.python.org/pep-0702/]. Patch by Jelle Zijlstra.

- gh-103708 [https://github.com/python/cpython/issues/103708]: Make hardcoded python name, a configurable parameter so that different implementations of python can override it instead of making huge diffs in sysconfig.py

- gh-66515 [https://github.com/python/cpython/issues/66515]: `mailbox.MH` now supports folders that do not contain a `.mh_sequences` file (e.g. Claws Mail IMAP-cache folders). Patch by Serhiy Storchaka.

- gh-83162 [https://github.com/python/cpython/issues/83162]: Renamed `re.error` to `PatternError` for clarity, and kept `re.error` for backward compatibility. Patch by Matthias Bussonnier and Adam Chhina.

- gh-91133 [https://github.com/python/cpython/issues/91133]: Fix a bug in `tempfile.TemporaryDirectory` cleanup, which now no longer dereferences symlinks when working around file system permission errors.

- bpo-43153 [https://bugs.python.org/issue?@action=redirect&bpo=43153]: On Windows, `tempfile.TemporaryDirectory` previously masked a `PermissionError` with `NotADirectoryError` during directory cleanup. It now correctly raises `PermissionError` if errors are not ignored. Patch by Andrei Kulakov and Ken Jin.

- bpo-32731 [https://bugs.python.org/issue?@action=redirect&bpo=32731]: `getpass.getuser()` now raises `OSError` for all failures rather than `ImportError` on systems lacking the `pwd` module or `KeyError` if the password database is empty.

- bpo-34321 [https://bugs.python.org/issue?@action=redirect&bpo=34321]: `mmap.mmap` now has a *trackfd* parameter on Unix; if it is `False`, the file descriptor specified by *fileno* will not be duplicated.

- bpo-35332 [https://bugs.python.org/issue?@action=redirect&bpo=35332]: The `shutil.rmtree()` function now ignores errors when calling `os.close()` when *ignore_errors* is `True`, and `os.close()` no longer retried after error.

- bpo-35928 [https://bugs.python.org/issue?@action=redirect&bpo=35928]: `io.TextIOWrapper` now correctly handles the decoding buffer after `read()` and `write()`.

- bpo-26791 [https://bugs.python.org/issue?@action=redirect&bpo=26791]: `shutil.move()` now moves a symlink into a directory when that directory is the target of the symlink. This provides the same behavior as the mv shell command. The previous behavior raised an exception. Patch by Jeffrey Kintscher.

- bpo-41422 [https://bugs.python.org/issue?@action=redirect&bpo=41422]: Fixed memory leaks of `pickle.Pickler` and `pickle.Unpickler` involving cyclic references via the internal memo mapping.

- bpo-19821 [https://bugs.python.org/issue?@action=redirect&bpo=19821]: The `pydoc.ispackage()` function has been deprecated.

- bpo-40262 [https://bugs.python.org/issue?@action=redirect&bpo=40262]: The `ssl.SSLSocket.recv_into()` method no longer requires the *buffer* argument to implement `__len__` and supports buffers with arbitrary item size.

- bpo-39912 [https://bugs.python.org/issue?@action=redirect&bpo=39912]: **`warnings.filterwarnings()`** and **`warnings.simplefilter()`** now raise appropriate exceptions instead of `AssertionError`. Patch contributed by Rémi Lapeyre.

- bpo-37260 [https://bugs.python.org/issue?@action=redirect&bpo=37260]: Fixed a race condition in **`shutil.rmtree()`** in which directory entries removed by another process or thread while `shutil.rmtree()` is running can cause it to raise FileNotFoundError. Patch by Jeffrey Kintscher.

- bpo-36959 [https://bugs.python.org/issue?@action=redirect&bpo=36959]: Fix some error messages for invalid ISO format string combinations in `strptime()` that referred to directives not contained in the format string. Patch by Gordon P. Hemsley.

- bpo-18060 [https://bugs.python.org/issue?@action=redirect&bpo=18060]: Fixed a class inheritance issue that can cause segfaults when deriving two or more levels of subclasses from a base class of Structure or Union.

- bpo-29779 [https://bugs.python.org/issue?@action=redirect&bpo=29779]: Add a new **`PYTHON_HISTORY`** environment variable to set the location of a `.python_history` file.

- bpo-21360 [https://bugs.python.org/issue?@action=redirect&bpo=21360]: **`mailbox.Maildir`** now ignores files with a leading dot.

Documentation

- gh-111699 [https://github.com/python/cpython/issues/111699]: Relocate `smtpd` deprecation notice to its own section rather than under `locale` in What's New in Python 3.12 document

- gh-110746 [https://github.com/python/cpython/issues/110746]: Improved markup for valid options/values for methods ttk.treeview.column and ttk.treeview.heading, and for Layouts.
- gh-95649 [https://github.com/python/cpython/issues/95649]: Document that the **`asyncio`** module contains code taken from v0.16.0 of the uvloop project [https://github.com/MagicStack/uvloop/tree/v0.16.0], as well as the required MIT licensing information.

Tests

- gh-111798 [https://github.com/python/cpython/issues/111798]: Disable `test_super_deep()` from `test_call` under pydebug builds on WASI; the stack depth is too small to make the test useful.

- gh-111801 [https://github.com/python/cpython/issues/111801]: Lower the recursion limit in `test_isinstance` for `test_infinitely_many_bases()`. This prevents a stack overflow on a pydebug build of WASI.

- gh-111802 [https://github.com/python/cpython/issues/111802]: Specify a low recursion depth for `test_bad_getattr()` in `test.pickletester` to avoid exhausting the stack under a pydebug build for WASI.

- gh-44626 [https://github.com/python/cpython/issues/44626]: Fix **`os.path.isabs()`** incorrectly returning `True` when given a path that starts with exactly one (back)slash on Windows.

  Fix **`pathlib.PureWindowsPath.is_absolute()`** incorrectly returning `False` for some paths beginning with two (back)slashes.

- gh-113633 [https://github.com/python/cpython/issues/113633]: Use module state for the _testcapi extension module.

- gh-109980 [https://github.com/python/cpython/issues/109980]: Fix `test_tarfile_vs_tar` in `test_shutil` for macOS, where system tar can include more information in the archive than `shutil.make_archive`.

- gh-112769 [https://github.com/python/cpython/issues/112769]: The tests now correctly compare zlib version when `zlib.ZLIB_RUNTIME_VERSION` contains non-integer suffixes. For example zlib-ng defines the version as `1.3.0.zlib-ng`.

- gh-112334 [https://github.com/python/cpython/issues/112334]: Adds a regression test to verify that `vfork()` is used when expected by `subprocess` on vfork enabled POSIX systems (Linux).

- gh-108927 [https://github.com/python/cpython/issues/108927]: Fixed order dependence in running tests in the same process when a test that has submodules (e.g. test_importlib) follows a test that imports its submodule (e.g. test_importlib.util) and precedes a test (e.g. test_unittest or test_compileall) that uses that submodule.

- bpo-40648 [https://bugs.python.org/issue?@action=redirect&bpo=40648]: Test modes that file can get with chmod() on Windows.

Build

- gh-114013 [https://github.com/python/cpython/issues/114013]: Fix `Tools/wasm/wasi.py` to not include the path to `python.wasm` as part of `HOSTRUNNER`. The environment variable is meant to specify how to run the WASI host only, having `python.wasm` and relevant flags appended to the `HOSTRUNNER`. This fixes `make test` work.
- gh-113258 [https://github.com/python/cpython/issues/113258]: Changed the Windows build to write out generated frozen modules into the build tree instead of the source tree.
- gh-112305 [https://github.com/python/cpython/issues/112305]: Fixed the `check-clean-src` step performed on out of tree builds to detect errant `$(srcdir)/Python/frozen_modules/*.h` files and recommend appropriate source tree cleanup steps to get a working build again.

- gh-112536 [https://github.com/python/cpython/issues/112536]: Add support for thread sanitizer (TSAN)
- gh-112867 [https://github.com/python/cpython/issues/112867]: Fix the build for the case that WITH_PYMALLOC_RADIX_TREE=0 set.
- gh-103065 [https://github.com/python/cpython/issues/103065]: Introduce `Tools/wasm/wasi.py` to simplify doing a WASI build.
- bpo-11102 [https://bugs.python.org/issue?@action=redirect&bpo=11102]: The **`os.major()`**, **`os.makedev()`**, and **`os.minor()`** functions are now available on HP-UX v3.
- bpo-36351 [https://bugs.python.org/issue?@action=redirect&bpo=36351]: Do not set ipv6type when cross-compiling.

Windows

- gh-114096 [https://github.com/python/cpython/issues/114096]: Process privileges that are activated for creating directory junctions are now restored afterwards, avoiding behaviour changes in other parts of the program.
- gh-111877 [https://github.com/python/cpython/issues/111877]: **`os.stat()`** calls were returning incorrect time values for files that could not be accessed directly.
- gh-111973 [https://github.com/python/cpython/issues/111973]: Update Windows installer to use SQLite 3.44.2.
- gh-113009 [https://github.com/python/cpython/issues/113009]: **`multiprocessing`**: On Windows, fix a race condition in `Process.terminate()`: no longer set the `returncode` attribute to always call `WaitForSingleObject()` in `Process.wait()`. Previously, sometimes the process was still running after `TerminateProcess()` even if `GetExitCodeProcess()` is not `STILL_ACTIVE`. Patch by Victor Stinner.
- gh-86179 [https://github.com/python/cpython/issues/86179]: Fixes path calculations when launching Python on Windows through a symlink.
- gh-71383 [https://github.com/python/cpython/issues/71383]: Update Tcl/Tk in Windows installer to 8.6.13 with a patch to

suppress incorrect ThemeChanged warnings.

- gh-111650 [https://github.com/python/cpython/issues/111650]: Ensures the `Py_GIL_DISABLED` preprocessor variable is defined in `pyconfig.h` so that extension modules written in C are able to use it.
- gh-112278 [https://github.com/python/cpython/issues/112278]: Reduce the time cost for some functions in **`platform`** on Windows if current user has no permission to the WMI.
- gh-73427 [https://github.com/python/cpython/issues/73427]: Deprecate **`sys._enablelegacywindowsfsencoding()`**. Use **`PYTHONLEGACYWINDOWSFSENCODING`** instead. Patch by Inada Naoki.
- gh-87868 [https://github.com/python/cpython/issues/87868]: Correctly sort and remove duplicate environment variables in **`_winapi.CreateProcess()`**.
- bpo-37308 [https://bugs.python.org/issue?@action=redirect&bpo=37308]: Fix mojibake in **`mmap.mmap`** when using a non-ASCII *tagname* argument on Windows.

macOS

- gh-113666 [https://github.com/python/cpython/issues/113666]: Add the following constants to module **`stat`**: `UF_SETTABLE`, `UF_TRACKED`, `UF_DATAVAULT`, `SF_SUPPORTED`, `SF_SETTABLE`, `SF_SYNTHETIC`, `SF_RESTRICTED`, `SF_FIRMLINK` and `SF_DATALESS`. The values `UF_SETTABLE`, `SF_SUPPORTED`, `SF_SETTABLE` and `SF_SYNTHETIC` are only available on macOS.
- gh-113536 [https://github.com/python/cpython/issues/113536]: **`os.waitid()`** is now available on macOS
- gh-110459 [https://github.com/python/cpython/issues/110459]: Running `configure ... --with-openssl-rpath=X/Y/Z` no longer fails to detect OpenSSL on macOS.
- gh-74573 [https://github.com/python/cpython/issues/74573]: Document that **`dbm.ndbm`** can silently corrupt DBM files on updates when exceeding undocumented platform limits, and can crash (segmentation fault) when reading such a corrupted file. (FB8919203)
- gh-65701 [https://github.com/python/cpython/issues/65701]: The **freeze** tool doesn't work with framework builds of Python.

Document this and bail out early when running the tool with such a build.

- gh-87277 [https://github.com/python/cpython/issues/87277]: webbrowser: Don’t look for X11 browsers on macOS. Those are generally not used and probing for them can result in starting XQuartz even if it isn’t used otherwise.
- gh-111973 [https://github.com/python/cpython/issues/111973]: Update macOS installer to use SQLite 3.44.2.
- gh-108269 [https://github.com/python/cpython/issues/108269]: Set `CFBundleAllowMixedLocalizations` to true in the Info.plist for the framework, embedded Python.app and IDLE.app with framework installs on macOS. This allows applications to pick up the user’s preferred locale when that’s different from english.
- gh-102362 [https://github.com/python/cpython/issues/102362]: Make sure the result of **`sysconfig.get_plaform()`** includes at least a major and minor versions, even if `MACOSX_DEPLOYMENT_TARGET` is set to only a major version during build to match the format expected by pip.
- gh-110017 [https://github.com/python/cpython/issues/110017]: Disable a signal handling stress test on macOS due to a bug in macOS (FB13453490).
- gh-110820 [https://github.com/python/cpython/issues/110820]: Make sure the preprocessor definitions for `ALIGNOF_MAX_ALIGN_T`, `SIZEOF_LONG_DOUBLE` and `HAVE_GCC_ASM_FOR_X64` are correct for Universal 2 builds on macOS.
- gh-109981 [https://github.com/python/cpython/issues/109981]: Use `/dev/fd` on macOS to determine the number of open files in `test.support.os_helper.fd_count` to avoid a crash with «guarded» file descriptors when probing for open files.

IDLE

- gh-72284 [https://github.com/python/cpython/issues/72284]: Improve the lists of features, editor key bindings, and shell key bingings in the IDLE doc.
- gh-113903 [https://github.com/python/cpython/issues/113903]: Fix rare failure of test.test_idle, in test_configdialog.
- gh-113729 [https://github.com/python/cpython/issues/113729]: Fix

the «Help -> IDLE Doc» menu bug in 3.11.7 and 3.12.1.
- gh-113269 [https://github.com/python/cpython/issues/113269]: Fix test_editor hang on macOS Catalina.
- gh-112898 [https://github.com/python/cpython/issues/112898]: Fix processing unsaved files when quitting IDLE on macOS.
- bpo-13586 [https://bugs.python.org/issue?@action=redirect&bpo=13586]: Enter the selected text when opening the «Replace» dialog.

C API

- gh-106560 [https://github.com/python/cpython/issues/106560]: Fix redundant declarations in the public C API. Declare PyBool_Type, PyLong_Type and PySys_Audit() only once. Patch by Victor Stinner.
- gh-112438 [https://github.com/python/cpython/issues/112438]: Fix support of format units «es», «et», «es#», and «et#» in nested tuples in `PyArg_ParseTuple()`-like functions.
- gh-111545 [https://github.com/python/cpython/issues/111545]: Add `Py_HashPointer()` function to hash a pointer. Patch by Victor Stinner.
- gh-65210 [https://github.com/python/cpython/issues/65210]: Change the declaration of the *keywords* parameter of `PyArg_ParseTupleAndKeywords()` and `PyArg_VaParseTupleAndKeywords()` for better compatibility with C++.

Python 3.13.0 alpha 2

Release date: 2023-11-22

Core and Builtins

- gh-112243 [https://github.com/python/cpython/issues/112243]: Don’t include comments in f-string debug expressions. Patch by Pablo Galindo

- gh-112287 [https://github.com/python/cpython/issues/112287]: Slightly optimize the Tier 2 (uop) interpreter by only loading `oparg` and `operand` when needed. Also double the trace size limit again, to 512 this time.

- gh-112266 [https://github.com/python/cpython/issues/112266]: Change docstrings of **`__dict__`** and **`__weakref__`**.

- gh-111807 [https://github.com/python/cpython/issues/111807]: Lower the max parser stack depth to 1000 under WASI debug builds.

- gh-111798 [https://github.com/python/cpython/issues/111798]: When Python is built in debug mode, set the C recursion limit to 500 instead of 1500. A debug build is likely built with low optimization level which implies higher stack memory usage than a release build. Patch by Victor Stinner.

- gh-106529 [https://github.com/python/cpython/issues/106529]: Enable translating unspecialized `FOR_ITER` to Tier 2.

- gh-111916 [https://github.com/python/cpython/issues/111916]: Make hashlib related modules thread-safe without the GIL

- gh-81137 [https://github.com/python/cpython/issues/81137]: Deprecate assignment to a function's `__code__` field when the new code object is of a mismatched type (e.g., from a generator to a plain function).

- gh-79932 [https://github.com/python/cpython/issues/79932]: Raise exception if **`frame.clear()`** is called on a suspended frame.

- gh-81925 [https://github.com/python/cpython/issues/81925]: Implement native thread ids for GNU KFreeBSD.

- gh-111843 [https://github.com/python/cpython/issues/111843]: Use exponential backoff to reduce the number of failed tier 2 optimization attempts by over 99%.

- gh-110829 [https://github.com/python/cpython/issues/110829]: Joining a thread now ensures the underlying OS thread has exited. This is required for safer fork() in multi-threaded processes.

- gh-109369 [https://github.com/python/cpython/issues/109369]: Make sure that tier 2 traces are de-optimized if the code is instrumented

- gh-111772 [https://github.com/python/cpython/issues/111772]: Specialize slot loads and stores for _Py_T_OBJECT as well as Py_T_OBJECT_EX

- gh-111666 [https://github.com/python/cpython/issues/111666]: Speed up `BaseExceptionGroup.derive()`, `BaseExceptionGroup.subgroup()`, and `BaseExceptionGroup.split()` by changing how they parse passed arguments.

- gh-111654 [https://github.com/python/cpython/issues/111654]: Fix runtime crash when some error happens in opcode `LOAD_FROM_DICT_OR_DEREF`.

- gh-111623 [https://github.com/python/cpython/issues/111623]: Add support for sharing tuples between interpreters using the cross-interpreter API. Patch by Anthony Shaw.

- gh-111354 [https://github.com/python/cpython/issues/111354]: The oparg of `YIELD_VALUE` is now `1` if the instruction is part of a yield-from or await, and `0` otherwise.

  The SUSPENDED frame state is now split into `SUSPENDED` and `SUSPENDED_YIELD_FROM`. This simplifies the code in `_PyGen_yf`.

- gh-111520 [https://github.com/python/cpython/issues/111520]: Merge the Tier 1 (bytecode) and Tier 2 (micro-ops) interpreters together, moving the Tier 2 interpreter loop and switch into `_PyEval_EvalFrameDefault()` in `Python/ceval.c`. The `Python/executor.c` file is gone. Also the `TIER_ONE` and `TIER_TWO` macros are now handled by the code generator.

  Beware! This changes the environment variables to enable micro-ops and their debugging to `PYTHON_UOPS` and `PYTHON_LLTRACE`.

- gh-109181 [https://github.com/python/cpython/issues/109181]: Speed up `Traceback` object creation by lazily compute the line number. Patch by Pablo Galindo

- gh-111420 [https://github.com/python/cpython/issues/111420]: Allow type comments in parenthesized `with` statements

- gh-111438 [https://github.com/python/cpython/issues/111438]: Add support for sharing floats between interpreters using the cross-interpreter API. Patch by Anthony Shaw.

- gh-111435 [https://github.com/python/cpython/issues/111435]: Add support for sharing of `True` and `False` between interpreters using the cross-interpreter API. Patch by Anthony Shaw.

- gh-102388 [https://github.com/python/cpython/issues/102388]: Fix a bug where `iso2022_jp_3` and `iso2022_jp_2004` codecs read out of bounds

- gh-111366 [https://github.com/python/cpython/issues/111366]: Fix an issue in the `codeop` that was causing `SyntaxError` exceptions raised in the presence of invalid syntax to not contain precise error messages. Patch by Pablo Galindo

- gh-111380 [https://github.com/python/cpython/issues/111380]: Fix a bug that was causing `SyntaxWarning` to appear twice when parsing if invalid syntax is encountered later. Patch by Pablo galindo

- gh-111374 [https://github.com/python/cpython/issues/111374]: Added a new environment variable `PYTHON_FROZEN_MODULES`. It determines whether or not frozen modules are ignored by the import machinery, equivalent of the `-X frozen_modules` command-line option.

- gh-111354 [https://github.com/python/cpython/issues/111354]: Remove `oparg` from `YIELD_VALUE`. Change `oparg` of `RESUME` to include information about the except-depth. These changes make it possible to simplify the code in generator close.

- gh-94438 [https://github.com/python/cpython/issues/94438]: Fix a regression that prevented jumping across `is None` and `is not None` when debugging. Patch by Savannah Ostrowski.

- gh-67224 [https://github.com/python/cpython/issues/67224]: Show source lines in tracebacks when using the `-c` option when running Python. Patch by Pablo Galindo

- gh-111123 [https://github.com/python/cpython/issues/111123]: Fix a bug where a **`global`** declaration in an **`except`** block is rejected when the global is used in the **`else`** block.

- gh-110938 [https://github.com/python/cpython/issues/110938]: Fix error messages for indented blocks with functions and classes with generic type parameters. Patch by Pablo Galindo

- gh-109214 [https://github.com/python/cpython/issues/109214]: Remove unnecessary instruction pointer updates before returning from frames.

- gh-110912 [https://github.com/python/cpython/issues/110912]: Correctly display the traceback for **`MemoryError`** exceptions using the **`traceback`** module. Patch by Pablo Galindo

- gh-109894 [https://github.com/python/cpython/issues/109894]: Fixed crash due to improperly initialized static **`MemoryError`** in subinterpreter.

- gh-110892 [https://github.com/python/cpython/issues/110892]: Return `NULL` for `PyTrace_RETURN` events caused by an exception

- gh-110864 [https://github.com/python/cpython/issues/110864]: Fix argument parsing by `_PyArg_UnpackKeywordsWithVararg` for functions defining pos-or-keyword, vararg, and kw-only parameters.

- gh-109094 [https://github.com/python/cpython/issues/109094]: Replace `prev_instr` on the interpreter frame by `instr_ptr` which points to the beginning of the instruction that is currently executing (or will execute once the frame resumes).

- gh-110805 [https://github.com/python/cpython/issues/110805]: Allow the repl to show source code and complete tracebacks. Patch by Pablo Galindo
- gh-110722 [https://github.com/python/cpython/issues/110722]: Add **`PYTHON_PRESITE=package.module`** to import a module early in the interpreter lifecycle before `site.py` is executed. Python needs to be built in debug mode for this option to exist.
- gh-110481 [https://github.com/python/cpython/issues/110481]: Implement biased reference counting in `--disable-gil` builds.
- gh-110543 [https://github.com/python/cpython/issues/110543]: Fix regression in Python 3.12 where **`types.CodeType.replace()`** would produce a broken code object if called on a module or class code object that contains a comprehension. Patch by Jelle Zijlstra.
- gh-89519 [https://github.com/python/cpython/issues/89519]: Removed chained **`classmethod`** descriptors (introduced in bpo-19072 [https://bugs.python.org/issue?@action=redirect&bpo=19072]). This can no longer be used to wrap other descriptors such as **`property`**. The core design of this feature was flawed and caused a number of downstream problems. To «pass-through» a **`classmethod`**, consider using the **`__wrapped__`** attribute that was added in Python 3.10.
- gh-103615 [https://github.com/python/cpython/issues/103615]: Use local events for opcode tracing
- bpo-46657 [https://bugs.python.org/issue?@action=redirect&bpo=46657]: Add mimalloc memory allocator support.
- gh-106718 [https://github.com/python/cpython/issues/106718]: When PyConfig.stdlib_dir is explicitly set, it’s now respected and won’t be overridden by PyConfig.home.

- gh-106905 [https://github.com/python/cpython/issues/106905]: Fix incorrect SystemError about AST constructor recursion depth mismatch.

- gh-100445 [https://github.com/python/cpython/issues/100445]: Improve error message for unterminated strings with escapes.

- bpo-45759 [https://bugs.python.org/issue?@action=redirect&bpo=45759]: Improved error messages for `elif`/`else` statements not matching any valid statements. Patch by Jeremiah Vivian.

Library

- gh-111942 [https://github.com/python/cpython/issues/111942]: Fix SystemError in the TextIOWrapper constructor with non-encodable «errors» argument in non-debug mode.

- gh-111995 [https://github.com/python/cpython/issues/111995]: Added the `NI_IDN` constant to the `socket` module when present in C at build time for use with `socket.getnameinfo()`.

- gh-109538 [https://github.com/python/cpython/issues/109538]: Issue warning message instead of having `RuntimeError` be displayed when event loop has already been closed at `StreamWriter.__del__()`.

- gh-111942 [https://github.com/python/cpython/issues/111942]: Fix crashes in `io.TextIOWrapper.reconfigure()` when pass invalid arguments, e.g. non-string encoding.

- gh-111460 [https://github.com/python/cpython/issues/111460]: `curses`: restore wide character support (including `curses.unget_wch()` and `get_wch()`) on macOS, which was unavailable due to a regression in Python 3.12.

- gh-103791 [https://github.com/python/cpython/issues/103791]: `contextlib.suppress` now supports suppressing exceptions raised as part of a `BaseExceptionGroup`, in addition to the recent support for `ExceptionGroup`.

- gh-111835 [https://github.com/python/cpython/issues/111835]: The **`mmap.mmap`** class now has an **`seekable()`** method that can be used when a seekable file-like object is required. The **`seek()`** method now returns the new absolute position. Patch by Donghee Na.

- gh-111804 [https://github.com/python/cpython/issues/111804]: Remove posix.fallocate() under WASI as the underlying posix_fallocate() is not available in WASI preview2.

- gh-111841 [https://github.com/python/cpython/issues/111841]: Fix truncating arguments on an embedded null character in **`os.putenv()`** and **`os.unsetenv()`** on Windows.

- gh-111768 [https://github.com/python/cpython/issues/111768]: **`wsgiref.util.is_hop_by_hop()`** is now exposed correctly in `__all__`.

- gh-80731 [https://github.com/python/cpython/issues/80731]: Avoid executing the default function in **`cmd.Cmd`** in an except block

- gh-111541 [https://github.com/python/cpython/issues/111541]: Fix **`doctest`** for **`SyntaxError`** not-builtin subclasses.

- gh-111719 [https://github.com/python/cpython/issues/111719]: Add extra argument validation for `alias` command in **`pdb`**

- gh-111482 [https://github.com/python/cpython/issues/111482]: **`time`**: Make **`time.clock_gettime()`** and **`time.clock_gettime_ns()`** functions up to 2x faster by faster calling convention. Patch by Victor Stinner.

- gh-110894 [https://github.com/python/cpython/issues/110894]: Call loop exception handler for exceptions in `client_connected_cb` of **`asyncio.start_server()`** so that applications can handle it. Patch by Kumar Aditya.

- gh-111531 [https://github.com/python/cpython/issues/111531]: Fix reference leaks in `bind_class()` and `bind_all()` methods of **`tkinter`** widgets.

- gh-111246 [https://github.com/python/cpython/issues/111246]: `asyncio.loop.create_unix_server()` will now automatically remove the Unix socket when the server is closed.

- gh-111356 [https://github.com/python/cpython/issues/111356]: Added `io.text_encoding()`, `io.DEFAULT_BUFFER_SIZE`, and `io.IncrementalNewlineDecoder` to `io.__all__`.

- gh-66425 [https://github.com/python/cpython/issues/66425]: Remove the code to set the REMOTE_HOST header from wsgiref module, as it is unreachable. This header is used for performance reasons, which is not necessary in the wsgiref module.

- gh-111429 [https://github.com/python/cpython/issues/111429]: Speed up `pathlib.PurePath.relative_to()` and `is_relative_to()`.

- gh-111342 [https://github.com/python/cpython/issues/111342]: Fixed typo in `math.sumprod()`.

- gh-68166 [https://github.com/python/cpython/issues/68166]: Remove mention of not supported «vsapi» element type in `tkinter.ttk.Style.element_create()`. Add tests for `element_create()` and other `ttk.Style` methods. Add examples for `element_create()` in the documentation.

- gh-111388 [https://github.com/python/cpython/issues/111388]: Add `show_group` parameter to `traceback.format_exception_only()`, which allows to format `ExceptionGroup` instances.

- gh-79033 [https://github.com/python/cpython/issues/79033]: Another attempt at fixing `asyncio.Server.wait_closed()`. It now blocks until both conditions are true: the server is closed, *and* there are no more active connections. (This means that in some cases where in 3.12.0 this function would *incorrectly* have returned immediately, it will now block; in particular, when there are no active connections but the server hasn't been closed yet.)

- gh-111259 [https://github.com/python/cpython/issues/111259]: Optimize recursive wildcards in `pathlib`.
- gh-111295 [https://github.com/python/cpython/issues/111295]: Fix `time` not checking for errors when initializing.
- gh-111253 [https://github.com/python/cpython/issues/111253]: Add error checking during `_socket` module init.
- gh-111251 [https://github.com/python/cpython/issues/111251]: Fix `_blake2` not checking for errors when initializing.
- gh-111233 [https://github.com/python/cpython/issues/111233]: Fix `select` not checking for errors when initializing.
- gh-111230 [https://github.com/python/cpython/issues/111230]: Fix `ssl` not checking for errors when initializing.
- gh-111174 [https://github.com/python/cpython/issues/111174]: Fix crash in `io.BytesIO.getbuffer()` called repeatedly for empty BytesIO.
- gh-111187 [https://github.com/python/cpython/issues/111187]: Postpone removal version for locale.getdefaultlocale() to Python 3.15.
- gh-111159 [https://github.com/python/cpython/issues/111159]: Fix `doctest` output comparison for exceptions with notes.
- gh-110910 [https://github.com/python/cpython/issues/110910]: Fix invalid state handling in `asyncio.TaskGroup` and `asyncio.Timeout`. They now raise proper RuntimeError if they are improperly used and are left in consistent state after this.
- gh-111092 [https://github.com/python/cpython/issues/111092]: Make turtledemo run without default root enabled.
- gh-110944 [https://github.com/python/cpython/issues/110944]: Support alias and convenience vars for `pdb` completion
- gh-110745 [https://github.com/python/cpython/issues/110745]: Added

newline parameter to **`pathlib.Path.read_text()`**. Patch by Junya Okabe.

- gh-84583 [https://github.com/python/cpython/issues/84583]: Make **`pdb`** enter post-mortem mode even for **`SyntaxError`**
- gh-80675 [https://github.com/python/cpython/issues/80675]: Set `f_trace_lines = True` on all frames upon **`pdb.set_trace()`**
- gh-110771 [https://github.com/python/cpython/issues/110771]: Expose the setup and cleanup portions of `asyncio.run_forever()` as the standalone methods `asyncio.run_forever_setup()` and `asyncio.run_forever_cleanup()`. This allows for tighter integration with GUI event loops.
- gh-110774 [https://github.com/python/cpython/issues/110774]: Support setting the **`asyncio.Runner`** loop_factory kwarg in **`unittest.IsolatedAsyncioTestCase`**
- gh-110392 [https://github.com/python/cpython/issues/110392]: Fix **`tty.setraw()`** and **`tty.setcbreak()`**: previously they returned partially modified list of the original tty attributes. **`tty.cfmakeraw()`** and **`tty.cfmakecbreak()`** now make a copy of the list of special characters before modifying it.
- gh-59013 [https://github.com/python/cpython/issues/59013]: Make line number of function breakpoint more precise in **`pdb`**
- gh-88434 [https://github.com/python/cpython/issues/88434]: Emit deprecation warning for non-integer numbers in **`gettext`** functions and methods that consider plural forms even if the translation was not found.
- gh-110395 [https://github.com/python/cpython/issues/110395]: Ensure that **`select.kqueue()`** objects correctly appear as closed in forked children, to prevent operations on an invalid file descriptor.
- gh-110196 [https://github.com/python/cpython/issues/110196]: Add

`__reduce__` method to **`IPv6Address`** in order to keep `scope_id`

- gh-109747 [https://github.com/python/cpython/issues/109747]: Improve errors for unsupported look-behind patterns. Now re.error is raised instead of OverflowError or RuntimeError for too large width of look-behind pattern.

- gh-109466 [https://github.com/python/cpython/issues/109466]: Add the **`ipaddress.IPv4Address.ipv6_mapped`** property, which returns the IPv4-mapped IPv6 address.

- gh-85098 [https://github.com/python/cpython/issues/85098]: Implement the CLI of the **`symtable`** module and improve the repr of **`Symbol`**.

- gh-108791 [https://github.com/python/cpython/issues/108791]: Improved error handling in **`pdb`** command line interface, making it produce more concise error messages.

- gh-105931 [https://github.com/python/cpython/issues/105931]: Change **`compileall`** to only strip the stripdir prefix from the full path recorded in the compiled `.pyc` file, when the prefix matches the start of the full path in its entirety. When the prefix does not match, no stripping is performed and a warning to this effect is displayed.

  Previously all path components of the stripdir prefix that matched the full path were removed, while those that did not match were left alone (including ones interspersed between matching components).

- gh-107431 [https://github.com/python/cpython/issues/107431]: Make the `DictProxy` and `ListProxy` types in **`multiprocessing.managers`** Generic Alias Types for `[]` use in typing contexts.

- gh-72904 [https://github.com/python/cpython/issues/72904]: Add **`glob.translate()`**. This function converts a pathname with shell-style wildcards to a regular expression.

- gh-90026 [https://github.com/python/cpython/issues/90026]: Define `USE_XATTRS` on Cygwin so that XATTR-related functions in the **`os`** module become available.

- gh-90890 [https://github.com/python/cpython/issues/90890]: New methods **`mailbox.Maildir.get_info()`**, **`mailbox.Maildir.set_info()`**, **`mailbox.Maildir.get_flags()`**, **`mailbox.Maildir.set_flags()`**, **`mailbox.Maildir.add_flag()`**, **`mailbox.Maildir.remove_flag()`**. These methods speed up accessing a message's info and/or flags and are useful when it is not necessary to access the message's contents, as when iterating over a Maildir to find messages with specific flags.

- gh-102956 [https://github.com/python/cpython/issues/102956]: Fix returning of empty byte strings after seek in zipfile module

- gh-102895 [https://github.com/python/cpython/issues/102895]: Added a parameter `local_exit` for **`code.interact()`** to prevent `exit()` and `quit` from closing `sys.stdin` and raise `SystemExit`.

- gh-97928 [https://github.com/python/cpython/issues/97928]: Change the behavior of **`tkinter.Text.count()`**. It now always returns an integer if one or less counting options are specified. Previously it could return a single count as a 1-tuple, an integer (only if option `"update"` was specified) or `None` if no items found. The result is now the same if `wantobjects` is set to `0`.

- gh-96954 [https://github.com/python/cpython/issues/96954]: Switch the storage of the unicode codepoint names to use a different data-structure, a directed acyclic word graph [https://en.wikipedia.org/wiki/Deterministic_acyclic_finite_state_automaton]. This makes the unicodedata shared library about 440 KiB smaller. Contributed by Carl Friedrich Bolz-Tereick using code from the PyPy project.

- gh-73561 [https://github.com/python/cpython/issues/73561]: Omit the interface scope from an IPv6 address when used as Host header by `http.client`.

- gh-86826 [https://github.com/python/cpython/issues/86826]: `zipinfo` now supports the full range of values in the TZ string determined by RFC 8536 and detects all invalid formats. Both Python and C implementations now raise exceptions of the same type on invalid data.

Tests

- gh-111808 [https://github.com/python/cpython/issues/111808]: Make the default value of `test.support.infinite_recursion()` to be conditional based on whether optimizations were used when compiling the interpreter. This helps with platforms like WASI whose stack size is greatly restricted in debug builds.
- gh-110722 [https://github.com/python/cpython/issues/110722]: Gathering line coverage of standard libraries within the regression test suite is now precise, as well as much faster. Patch by Łukasz Langa.
- gh-110367 [https://github.com/python/cpython/issues/110367]: Make regrtest `--verbose3` option compatible with `--huntrleaks -jN` options. The `./python -m test -j1 -R 3:3 --verbose3` command now works as expected. Patch by Victor Stinner.
- gh-111165 [https://github.com/python/cpython/issues/111165]: Remove no longer used functions `run_unittest()` and `run_doctest()` from the `test.support` module.
- gh-110932 [https://github.com/python/cpython/issues/110932]: Fix regrtest if the `SOURCE_DATE_EPOCH` environment variable is defined: use the variable value as the random seed. Patch by Victor Stinner.
- gh-110995 [https://github.com/python/cpython/issues/110995]: test_gdb: Fix detection of gdb built without Python scripting support. Patch by Victor Stinner.
- gh-110918 [https://github.com/python/cpython/issues/110918]: Test case matching patterns specified by options `--match`, `--ignore`, `--matchfile` and `--ignorefile` are now tested in the order of specification, and the last match determines

whether the test case be run or ignored.

- gh-108747 [https://github.com/python/cpython/issues/108747]: Add unit test for `usercustomize` and `sitecustomize` hooks from **`site`**.

Build

- gh-96954 [https://github.com/python/cpython/issues/96954]: Make `make regen-unicodedata` work for out-of-tree builds of CPython.
- gh-112088 [https://github.com/python/cpython/issues/112088]: Add `Tools/build/regen-configure.sh` script to regenerate the `configure` with an Ubuntu container image. The `quay.io/tiran/cpython_autoconf:271` container image (tiran/cpython_autoconf [https://github.com/tiran/cpython_autoconf]) is no longer used. Patch by Victor Stinner.
- gh-111046 [https://github.com/python/cpython/issues/111046]: For wasi-threads, memory is now exported to fix compatibility issues with some wasm runtimes.
- gh-110828 [https://github.com/python/cpython/issues/110828]: AIX 32bit needs `-latomic` to build the **`_testcapi`** extension module.
- gh-85283 [https://github.com/python/cpython/issues/85283]: The `errno`, `md5`, `resource`, `winsound`, `_ctypes_test`, `_multiprocessing.posixshmem`, `_scproxy`, `_stat`, `_testimportmultiple` and `_uuid` C extensions are now built with the limited C API. Patch by Victor Stinner.

Windows

- gh-111856 [https://github.com/python/cpython/issues/111856]: Fixes **`fstat()`** on file systems that do not support file ID requests. This includes FAT32 and exFAT.
- gh-111293 [https://github.com/python/cpython/issues/111293]: Fix **`os.DirEntry.inode`** dropping higher 64 bits of a file id on some filesystems on Windows.
- gh-110913 [https://github.com/python/cpython/issues/110913]: WindowsConsoleIO now correctly chunks large buffers without splitting up UTF-8 sequences.

macOS

- gh-59703 [https://github.com/python/cpython/issues/59703]: For macOS framework builds, in `getpath.c` use the system `dladdr` function to find the path to the shared library rather than depending on deprecated macOS APIs.
- gh-110950 [https://github.com/python/cpython/issues/110950]: Update macOS installer to include an upstream Tcl/Tk fix for the `Secure coding is not enabled for restorable state!` warning encountered in Tkinter on macOS 14 Sonoma.
- gh-111015 [https://github.com/python/cpython/issues/111015]: Ensure that IDLE.app and Python Launcher.app are installed with appropriate permissions on macOS builds.
- gh-71383 [https://github.com/python/cpython/issues/71383]: Update macOS installer to include an upstream Tcl/Tk fix for the `ttk::ThemeChanged` error encountered in Tkinter.
- gh-92603 [https://github.com/python/cpython/issues/92603]: Update macOS installer to include a fix accepted by upstream Tcl/Tk for a crash encountered after the first **`tkinter.Tk()`** instance is destroyed.

IDLE

- bpo-35668 [https://bugs.python.org/issue?@action=redirect&bpo=35668]: Add docstrings to the IDLE debugger module. Fix two bugs: initialize `Idb.botframe` (should be in Bdb); in `Idb.in_rpc_code`, check whether `prev_frame` is `None` before trying to use it. Greatly expand test_debugger.

Tools/Demos

- gh-111903 [https://github.com/python/cpython/issues/111903]: Argument Clinic now supports the `@critical_section` directive that instructs Argument Clinic to generate a critical section around the function call, which locks the `self` object in `--disable-gil` builds. Patch by Sam Gross.

C API

- gh-112026 [https://github.com/python/cpython/issues/112026]: Add again the private `_PyThreadState_UncheckedGet()` function as an alias to the new public **`PyThreadState_GetUnchecked()`** function. Patch by Victor Stinner.

- gh-112026 [https://github.com/python/cpython/issues/112026]: Restore the removed `_PyDict_GetItemStringWithError()` function. It is used by numpy. Patch by Victor Stinner.

- gh-112026 [https://github.com/python/cpython/issues/112026]: Restore removed private C API functions, macros and structures which have no simple replacement for now:
  - _PyDict_GetItem_KnownHash()
  - _PyDict_NewPresized()
  - _PyHASH_BITS
  - _PyHASH_IMAG
  - _PyHASH_INF
  - _PyHASH_MODULUS
  - _PyHASH_MULTIPLIER
  - _PyLong_Copy()
  - _PyLong_FromDigits()
  - _PyLong_New()
  - _PyLong_Sign()
  - _PyObject_CallMethodId()
  - _PyObject_CallMethodNoArgs()
  - _PyObject_CallMethodOneArg()
  - _PyObject_CallOneArg()
  - _PyObject_EXTRA_INIT
  - _PyObject_FastCallDict()
  - _PyObject_GetAttrId()
  - _PyObject_Vectorcall()
  - _PyObject_VectorcallMethod()
  - _PyStack_AsDict()
  - _PyThread_CurrentFrames()
  - _PyUnicodeWriter structure

    - _PyUnicodeWriter_Dealloc()
    - _PyUnicodeWriter_Finish()
    - _PyUnicodeWriter_Init()
    - _PyUnicodeWriter_Prepare()
    - _PyUnicodeWriter_PrepareKind()
    - _PyUnicodeWriter_WriteASCIIString()
    - _PyUnicodeWriter_WriteChar()
    - _PyUnicodeWriter_WriteLatin1String()
    - _PyUnicodeWriter_WriteStr()
    - _PyUnicodeWriter_WriteSubstring()
    - _PyUnicode_AsString()
    - _PyUnicode_FromId()
    - _PyVectorcall_Function()
    - _Py_IDENTIFIER()
    - _Py_c_abs()
    - _Py_c_diff()
    - _Py_c_neg()
    - _Py_c_pow()
    - _Py_c_prod()
    - _Py_c_quot()
    - _Py_c_sum()
    - _Py_static_string()
    - _Py_static_string_init()

  Patch by Victor Stinner.

- gh-112026 [https://github.com/python/cpython/issues/112026]: Add again `<ctype.h>` and `<unistd.h>` includes in `Python.h`, but don't include them in the limited C API version 3.13 and newer. Patch by Victor Stinner.

- gh-111956 [https://github.com/python/cpython/issues/111956]: Add internal-only one-time initialization API: `_PyOnceFlag` and `_PyOnceFlag_CallOnce`.

- gh-111262 [https://github.com/python/cpython/issues/111262]: Add **`PyDict_Pop()`** and **`PyDict_PopString()`** functions: remove a key from a dictionary and optionally return the removed value. This is similar to **`dict.pop()`**, but without the default value and not raising **`KeyError`** if the key missing. Patch by Stefan Behnel and Victor Stinner.

- gh-111863 [https://github.com/python/cpython/issues/111863]: Rename `Py_NOGIL` to `Py_GIL_DISABLED`. Patch by Hugo van Kemenade.

- gh-111138 [https://github.com/python/cpython/issues/111138]: Add **`PyList_Extend()`** and **`PyList_Clear()`** functions: similar to Python `list.extend()` and `list.clear()` methods. Patch by Victor Stinner.

- gh-108765 [https://github.com/python/cpython/issues/108765]: On Windows, `Python.h` no longer includes the `<stddef.h>` standard header file. If needed, it should now be included explicitly. Patch by Victor Stinner.

- gh-111569 [https://github.com/python/cpython/issues/111569]: Implement «Python Critical Sections» from **PEP 703** [https://peps.python.org/pep-0703/]. These are macros to help replace the GIL with per-object locks in the `--disable-gil` build of CPython. The macros are no-ops in the default build.

- gh-111506 [https://github.com/python/cpython/issues/111506]: In the limited C API version 3.13, **`Py_SET_REFCNT()`** function is now implemented as an opaque function call. Patch by Victor Stinner.

- gh-108082 [https://github.com/python/cpython/issues/108082]: Add **`PyErr_FormatUnraisable()`** function.

- gh-110964 [https://github.com/python/cpython/issues/110964]: Move the undocumented private _PyArg functions and _PyArg_Parser structure to internal C API (`pycore_modsupport.h`). Patch by Victor Stinner.

- gh-110815 [https://github.com/python/cpython/issues/110815]: Support non-ASCII keyword names in **`PyArg_ParseTupleAndKeywords()`**.

- gh-109587 [https://github.com/python/cpython/issues/109587]: Introduced **`PyUnstable_PerfTrampoline_CompileCode()`**, **`PyUnstable_PerfTrampoline_SetPersistAfterFork()`**

and `PyUnstable_CopyPerfMapFile()`. These functions allow extension modules to initialize trampolines eagerly, after the application is «warmed up». This makes it possible to have perf-trampolines running in an always-enabled fashion.

- gh-85283 [https://github.com/python/cpython/issues/85283]: Add the `PySys_Audit()` function to the limited C API. Patch by Victor Stinner.

- gh-85283 [https://github.com/python/cpython/issues/85283]: Add `PyMem_RawMalloc()`, `PyMem_RawCalloc()`, `PyMem_RawRealloc()` and `PyMem_RawFree()` to the limited C API. Patch by Victor Stinner.

- gh-106672 [https://github.com/python/cpython/issues/106672]: Functions `PyDict_GetItem()`, `PyDict_GetItemString()`, `PyMapping_HasKey()`, `PyMapping_HasKeyString()`, `PyObject_HasAttr()`, `PyObject_HasAttrString()`, and `PySys_GetObject()`, which clear all errors occurred during calling the function, report now them using `sys.unraisablehook()`.

- gh-67565 [https://github.com/python/cpython/issues/67565]: Remove redundant C-contiguity check in `getargs.c`, `binascii`, `ssl` and Argument Clinic. Patched by Stefan Krah and Furkan Onder

Python 3.13.0 alpha 1

Release date: 2023-10-13

Security

- gh-108310 [https://github.com/python/cpython/issues/108310]: Fixed an issue where instances of `ssl.SSLSocket` were vulnerable to a bypass of the TLS handshake and included protections (like certificate verification) and treating sent unencrypted data as if it were post-handshake TLS encrypted data. Security issue reported as **CVE 2023-40217** [https://www.cve.org/CVERecord?id=CVE-2023-40217] by Aapo Oksman. Patch by Gregory P. Smith.

- gh-107774 [https://github.com/python/cpython/issues/107774]: PEP 669 specifies that `sys.monitoring.register_callback` will generate an audit event. Pre-releases of Python 3.12 did not generate the audit event. This is now fixed.
- gh-102988 [https://github.com/python/cpython/issues/102988]: Reverted the **`email.utils`** security improvement change released in 3.12beta4 that unintentionally caused **`email.utils.getaddresses`** to fail to parse email addresses with a comma in the quoted name field. See gh-106669 [https://github.com/python/cpython/issues/106669].
- gh-99108 [https://github.com/python/cpython/issues/99108]: Refresh our new HACL* built-in **`hashlib`** code from upstream. Built-in SHA2 should be faster and an issue with SHA3 on 32-bit platforms is fixed.
- gh-102509 [https://github.com/python/cpython/issues/102509]: Start initializing `ob_digit` during creation of **`PyLongObject`** objects. Patch by Illia Volochii.

Core and Builtins

- gh-110782 [https://github.com/python/cpython/issues/110782]: Fix crash when **`typing.TypeVar`** is constructed with a keyword argument. Patch by Jelle Zijlstra.

- gh-110752 [https://github.com/python/cpython/issues/110752]: Reset `ceval.eval_breaker` in **`interpreter_clear()`**

- gh-110721 [https://github.com/python/cpython/issues/110721]: Use the **`traceback`** implementation for the default **`PyErr_Display()`** functionality. Patch by Pablo Galindo

- gh-110696 [https://github.com/python/cpython/issues/110696]: Fix incorrect error message for invalid argument unpacking. Patch by Pablo Galindo

- gh-104169 [https://github.com/python/cpython/issues/104169]: Split the tokenizer into two separate directories: - One part includes the actual lexeme producing logic and lives in `Parser/lexer`. - The second part wraps the lexer according to the different tokenization modes we have (string, utf-8, file, interactive, readline) and lives in `Parser/tokenizer`.

- gh-110688 [https://github.com/python/cpython/issues/110688]: Remove undocumented `test_c_api` method from `set`, which was only defined for testing purposes under `Py_DEBUG`. Now we have proper CAPI tests.

- gh-104584 [https://github.com/python/cpython/issues/104584]: Fix a reference leak when running with **`PYTHONUOPS`** or `-X uops` enabled.

- gh-110514 [https://github.com/python/cpython/issues/110514]: Add `PY_THROW` to `sys.setprofile()` events

- gh-110489 [https://github.com/python/cpython/issues/110489]: Optimise `math.ceil()` when the input is exactly a float, resulting in about a 10% improvement.

- gh-110455 [https://github.com/python/cpython/issues/110455]: Guard `assert(tstate->thread_id > 0)` with `#ifndef HAVE_PTHREAD_STUBS`. This allows for for pydebug builds to work under WASI which (currently) lacks thread support.

- gh-110309 [https://github.com/python/cpython/issues/110309]: Remove unnecessary empty constant nodes in the ast of f-string specs.

- gh-110259 [https://github.com/python/cpython/issues/110259]: Correctly identify the format spec in f-strings (with single or triple quotes) that have multiple lines in the expression part and include a formatting spec. Patch by Pablo Galindo

- gh-110237 [https://github.com/python/cpython/issues/110237]: Fix missing error checks for calls to `PyList_Append` in `_PyEval_MatchClass`.

- gh-110164 [https://github.com/python/cpython/issues/110164]: regrtest: If the `SOURCE_DATE_EPOCH` environment variable is defined, regrtest now disables tests randomization. Patch by Victor Stinner.

- gh-109889 [https://github.com/python/cpython/issues/109889]: Fix the compiler's redundant NOP detection algorithm to skip over NOPs with no line number when looking for the next instruction's lineno.
- gh-109853 [https://github.com/python/cpython/issues/109853]: `sys.path[0]` is now set correctly for subinterpreters.
- gh-109923 [https://github.com/python/cpython/issues/109923]: Set line number on the `POP_TOP` that follows a `RETURN_GENERATOR`.
- gh-105716 [https://github.com/python/cpython/issues/105716]: Subinterpreters now correctly handle the case where they have threads running in the background. Before, such threads would interfere with cleaning up and destroying them, as well as prevent running another script.
- gh-109369 [https://github.com/python/cpython/issues/109369]: The internal eval_breaker and supporting flags, plus the monitoring version have been merged into a single atomic integer to speed up checks.
- gh-109823 [https://github.com/python/cpython/issues/109823]: Fix bug where compiler does not adjust labels when removing an empty basic block which is a jump target.
- gh-109793 [https://github.com/python/cpython/issues/109793]: The main thread no longer exits prematurely when a subinterpreter is cleaned up during runtime finalization. The bug was a problem particularly because, when triggered, the Python process would always return with a 0 exitcode, even if it failed.
- gh-109719 [https://github.com/python/cpython/issues/109719]: Fix missing jump target labels when compiler reorders cold/warm blocks.

- gh-109595 [https://github.com/python/cpython/issues/109595]: Add `-X cpu_count` command line option to override return results of `os.cpu_count()` and `os.process_cpu_count()`. This option is useful for users who need to limit CPU resources of a container system without having to modify the container (application code). Patch by Donghee Na.

- gh-109627 [https://github.com/python/cpython/issues/109627]: Fix bug where the compiler does not assign a new jump target label to a duplicated small exit block.

- gh-109596 [https://github.com/python/cpython/issues/109596]: Fix some tokens in the grammar that were incorrectly marked as soft keywords. Also fix some repeated rule names and ensure that repeated rules are not allowed. Patch by Pablo Galindo

- gh-109496 [https://github.com/python/cpython/issues/109496]: On a Python built in debug mode, `Py_DECREF()` now calls `_Py_NegativeRefcount()` if the object is a dangling pointer to deallocated memory: memory filled with `0xDD` «dead byte» by the debug hook on memory allocators. The fix is to check the reference count *before* checking for `_Py_IsImmortal()`. Patch by Victor Stinner.

- gh-107265 [https://github.com/python/cpython/issues/107265]: Deopt opcodes hidden by the executor when base opcode is needed

- gh-109371 [https://github.com/python/cpython/issues/109371]: Deopted instructions correctly for tool initialization and modified the incorrect assertion in instrumentation, when a previous tool already sets INSTRUCTION events

- gh-105658 [https://github.com/python/cpython/issues/105658]: Fix bug where the line trace of an except block ending with a conditional includes an excess event with the line of the conditional expression.

- gh-109219 [https://github.com/python/cpython/issues/109219]: Fix compiling type param scopes that use a name which is also free in an inner scope.

- gh-109351 [https://github.com/python/cpython/issues/109351]: Fix crash when compiling an invalid AST involving a named (walrus) expression.

- gh-109341 [https://github.com/python/cpython/issues/109341]: Fix crash when compiling an invalid AST involving a **`ast.TypeAlias`**.

- gh-109195 [https://github.com/python/cpython/issues/109195]: Fix source location for the `LOAD_*` instruction preceding a `LOAD_SUPER_ATTR` to load the `super` global (or shadowing variable) so that it encompasses only the name `super` and not the following parentheses.

- gh-109256 [https://github.com/python/cpython/issues/109256]: Opcode IDs for specialized opcodes are allocated in their own range to improve stability of the IDs for the “real” opcodes.

- gh-109216 [https://github.com/python/cpython/issues/109216]: Fix possible memory leak in **`BUILD_MAP`**.

- gh-109207 [https://github.com/python/cpython/issues/109207]: Fix a SystemError in `__repr__` of symtable entry object.

- gh-109179 [https://github.com/python/cpython/issues/109179]: Fix bug where the C traceback display drops notes from **`SyntaxError`**.

- gh-109118 [https://github.com/python/cpython/issues/109118]: Disallow nested scopes (lambdas, generator expressions, and comprehensions) within PEP 695 annotation scopes that are nested within classes.

- gh-109156 [https://github.com/python/cpython/issues/109156]: Add tests for de-instrumenting instructions while keeping the instrumentation for lines

- gh-109114 [https://github.com/python/cpython/issues/109114]: Relax the detection of the error message for invalid lambdas inside f-strings to not search for arbitrary replacement fields to avoid false positives. Patch by Pablo Galindo

- gh-105848 [https://github.com/python/cpython/issues/105848]: Add a new **`CALL_KW`** opcode, used for calls containing keyword arguments. Also, fix a possible crash when jumping over method calls in a debugger.

- gh-109052 [https://github.com/python/cpython/issues/109052]: Use the base opcode when comparing code objects to avoid interference from instrumentation

- gh-109118 [https://github.com/python/cpython/issues/109118]: Fix interpreter crash when a NameError is raised inside the type parameters of a generic class.

- gh-88943 [https://github.com/python/cpython/issues/88943]: Improve syntax error for non-ASCII character that follows a numerical literal. It now points on the invalid non-ASCII character, not on the valid numerical literal.

- gh-108976 [https://github.com/python/cpython/issues/108976]: Fix crash that occurs after de-instrumenting a code object in a monitoring callback.

- gh-108732 [https://github.com/python/cpython/issues/108732]: Make iteration variables of module- and class-scoped comprehensions visible to pdb and other tools that use `frame.f_locals` again.

- gh-108959 [https://github.com/python/cpython/issues/108959]: Fix caret placement for error locations for subscript and binary operations that involve non-semantic parentheses and spaces. Patch by Pablo Galindo

- gh-104584 [https://github.com/python/cpython/issues/104584]: Fix a crash when running with **`PYTHONUOPS`** or **`-X uops`** enabled and an error occurs during optimization.

- gh-108727 [https://github.com/python/cpython/issues/108727]: Define `tp_dealloc` for `CounterOptimizer_Type`. This fixes a segfault on deallocation.

- gh-108520 [https://github.com/python/cpython/issues/108520]: Fix

`multiprocessing.synchronize.SemLock.__setstate__()` to properly initialize **`multiprocessing.synchronize.SemLock._is_fork_ctx`**. This fixes a regression when passing a SemLock across nested processes.

Rename **`multiprocessing.synchronize.SemLock.is_fork_ctx`** to **`multiprocessing.synchronize.SemLock._is_fork_ctx`** to avoid exposing it as public API.

- gh-108654 [https://github.com/python/cpython/issues/108654]: Restore locals shadowed by an inlined comprehension if the comprehension raises an exception.

- gh-108488 [https://github.com/python/cpython/issues/108488]: Change the initialization of inline cache entries so that the cache entry for `JUMP_BACKWARD` is initialized to zero, instead of the `adaptive_counter_warmup()` value used for all other instructions. This counter, unique among instructions, counts up from zero.

- gh-108716 [https://github.com/python/cpython/issues/108716]: Turn off deep-freezing of code objects. Modules are still frozen, so that a file system search is not needed for common modules.

- gh-108614 [https://github.com/python/cpython/issues/108614]: Add RESUME_CHECK instruction, to avoid having to handle instrumentation, signals, and contexts switches in the tier 2 execution engine.

- gh-108487 [https://github.com/python/cpython/issues/108487]: Move an assert that would cause a spurious crash in a devious case that should only trigger deoptimization.

- gh-106176 [https://github.com/python/cpython/issues/106176]: Use a `WeakValueDictionary` to track the lists containing the modules each thread is currently importing. This helps avoid a reference leak from keeping the list around longer than necessary. Weakrefs are used as GC can't interrupt the cleanup.

- gh-105481 [https://github.com/python/cpython/issues/105481]: The regen-opcode build stage was removed and its work is now done in regen-cases.
- gh-107901 [https://github.com/python/cpython/issues/107901]: Fix missing line number on **`JUMP_BACKWARD`** at the end of a for loop.
- gh-108113 [https://github.com/python/cpython/issues/108113]: The **`compile()`** built-in can now accept a new flag, `ast.PyCF_OPTIMIZED_AST`, which is similar to `ast.PyCF_ONLY_AST` except that the returned `AST` is optimized according to the value of the `optimize` argument.

  `ast.parse()` now accepts an optional argument `optimize` which is passed on to the **`compile()`** built-in. This makes it possible to obtain an optimized `AST`.
- gh-107971 [https://github.com/python/cpython/issues/107971]: Opcode IDs are generated from bytecodes.c instead of being hard coded in opcode.py.
- gh-107944 [https://github.com/python/cpython/issues/107944]: Improve error message for function calls with bad keyword arguments. Patch by Pablo Galindo
- gh-108390 [https://github.com/python/cpython/issues/108390]: Raise an exception when setting a non-local event (`RAISE`, `EXCEPTION_HANDLED`, etc.) in `sys.monitoring.set_local_events`.

  Fixes crash when tracing in recursive calls to Python classes.
- gh-108035 [https://github.com/python/cpython/issues/108035]: Remove the `_PyCFrame` struct, moving the pointer to the current interpreter frame back to the threadstate, as it was for 3.10 and earlier. The `_PyCFrame` existed as a performance optimization for tracing. Since PEP 669 has been

implemented, this optimization no longer applies.

- gh-91051 [https://github.com/python/cpython/issues/91051]: Fix abort / segfault when using all eight type watcher slots, on platforms where `char` is signed by default.

- gh-106581 [https://github.com/python/cpython/issues/106581]: Fix possible assertion failures and missing instrumentation events when **`PYTHONUOPS`** or **`-X uops`** is enabled.

- gh-107526 [https://github.com/python/cpython/issues/107526]: Revert converting `vars`, `dir`, `next`, `getattr`, and `iter` to argument clinic.

- gh-84805 [https://github.com/python/cpython/issues/84805]: Autogenerate signature for **`METH_NOARGS`** and **`METH_O`** extension functions.

- gh-107758 [https://github.com/python/cpython/issues/107758]: Make the `dump_stack()` routine used by the `lltrace` feature (low-level interpreter debugging) robust against recursion by ensuring that it never calls a `__repr__` method implemented in Python. Also make the similar output for Tier-2 uops appear on `stdout` (instead of `stderr`), to match the `lltrace` code in ceval.c.

- gh-107659 [https://github.com/python/cpython/issues/107659]: Add docstrings for **`ctypes.pointer()`** and **`ctypes.POINTER()`**.

- gh-105848 [https://github.com/python/cpython/issues/105848]: Modify the bytecode so that the actual callable for a **`CALL`** is at a consistent position on the stack (regardless of whether or not bound-method-calling optimizations are active).

- gh-107674 [https://github.com/python/cpython/issues/107674]: Fixed performance regression in `sys.settrace`.

- gh-107724 [https://github.com/python/cpython/issues/107724]: In pre-release versions of 3.12, up to rc1, the sys.monitoring callback function for the `PY_THROW` event was missing the third, exception argument. That is now fixed.

- gh-84436 [https://github.com/python/cpython/issues/84436]: Skip reference count modifications for many known immortal objects.

- gh-107596 [https://github.com/python/cpython/issues/107596]: Specialize subscripting `str` objects by `int` indexes.

- gh-107080 [https://github.com/python/cpython/issues/107080]: Trace refs builds (`--with-trace-refs`) were crashing when used with isolated subinterpreters. The problematic global state has been isolated to each interpreter. Other fixing the crashes, this change does not affect users.

- gh-107557 [https://github.com/python/cpython/issues/107557]: Generate the cases needed for the barebones tier 2 abstract interpreter for optimization passes in CPython.

- gh-106608 [https://github.com/python/cpython/issues/106608]: Make `_PyUOpExecutorObject` variable length.

- gh-100964 [https://github.com/python/cpython/issues/100964]: Clear generators” exception state after `return` to break reference cycles.

- gh-107455 [https://github.com/python/cpython/issues/107455]: Improve error messages when converting an incompatible type to `ctypes.c_char_p`, `ctypes.c_wchar_p` and `ctypes.c_void_p`.

- gh-107263 [https://github.com/python/cpython/issues/107263]: Increase C recursion limit for functions other than the main interpreter from 800 to 1500. This should allow functions like `list.__repr__` and `json.dumps` to handle all the inputs that they could prior to 3.12

- gh-104584 [https://github.com/python/cpython/issues/104584]: Fix an issue which caused incorrect inline caches to be read when running with **`PYTHONUOPS`** or `-X uops` enabled.

- gh-104432 [https://github.com/python/cpython/issues/104432]: Fix

potential unaligned memory access on C APIs involving returned sequences of `char *` pointers within the **`grp`** and **`socket`** modules. These were revealed using a `-fsaniziter=alignment` build on ARM macOS. Patch by Christopher Chavez.

- gh-106078 [https://github.com/python/cpython/issues/106078]: Isolate **`_decimal`** (apply **PEP 687** [https://peps.python.org/pep-0687/]). Patch by Charlie Zhao.

- gh-106898 [https://github.com/python/cpython/issues/106898]: Add the exception as the third argument to `PY_UNIND` callbacks in `sys.monitoring`. This makes the `PY_UNWIND` callback consistent with the other exception handling callbacks.

- gh-106895 [https://github.com/python/cpython/issues/106895]: Raise a `ValueError` when a monitoring callback function returns `DISABLE` for events that cannot be disabled locally.

- gh-106897 [https://github.com/python/cpython/issues/106897]: Add a `RERAISE` event to `sys.monitoring`, which occurs when an exception is reraise, either explicitly by a plain `raise` statement, or implicitly in an `except` or `finally` block.

- gh-77377 [https://github.com/python/cpython/issues/77377]: Ensure that multiprocessing synchronization objects created in a fork context are not sent to a different process created in a spawn context. This changes a segfault into an actionable RuntimeError in the parent process.

- gh-106931 [https://github.com/python/cpython/issues/106931]: Statically allocated string objects are now interned globally instead of per-interpreter. This fixes a situation where such a string would only be interned in a single interpreter. Normal string objects are unaffected.

- gh-104621 [https://github.com/python/cpython/issues/104621]: Unsupported modules now always fail to be imported.

- gh-107122 [https://github.com/python/cpython/issues/107122]: Add **`dbm.ndbm.ndbm.clear()`** to **`dbm.ndbm`**. Patch By

Donghee Na.

- gh-107122 [https://github.com/python/cpython/issues/107122]: Add **`dbm.gnu.gdbm.clear()`** to **`dbm.gnu`**. Patch By Donghee Na.

- gh-107015 [https://github.com/python/cpython/issues/107015]: The ASYNC and AWAIT tokens are removed from the Grammar, which removes the possibility of making `async` and `await` soft keywords when using `feature_version<7` in **`ast.parse()`**.

- gh-106917 [https://github.com/python/cpython/issues/106917]: Fix classmethod-style **`super()`** method calls (i.e., where the second argument to **`super()`**, or the implied second argument drawn from `self/cls` in the case of zero-arg super, is a type) when the target of the call is not a classmethod.

- gh-105699 [https://github.com/python/cpython/issues/105699]: Python no longer crashes due an infrequent race when initializing per-interpreter interned strings. The crash would manifest when the interpreter was finalized.

- gh-105699 [https://github.com/python/cpython/issues/105699]: Python no longer crashes due to an infrequent race in setting `Py_FileSystemDefaultEncoding` and `Py_FileSystemDefaultEncodeErrors` (both deprecated), when simultaneously initializing two isolated subinterpreters. Now they are only set during runtime initialization.

- gh-106908 [https://github.com/python/cpython/issues/106908]: Fix various hangs, reference leaks, test failures, and tracing/introspection bugs when running with **`PYTHONUOPS`** or **`-X uops`** enabled.

- gh-106092 [https://github.com/python/cpython/issues/106092]: Fix a segmentation fault caused by a use-after-free bug in `frame_dealloc` when the trashcan delays the deallocation of a `PyFrameObject`.

- gh-106485 [https://github.com/python/cpython/issues/106485]: Reduce the number of materialized instances dictionaries by dematerializing them when possible.

- gh-106719 [https://github.com/python/cpython/issues/106719]: No longer suppress arbitrary errors in the `__annotations__` getter and setter in the type and module types.

- gh-106723 [https://github.com/python/cpython/issues/106723]: Propagate `frozen_modules` to multiprocessing spawned process interpreters.

- gh-104909 [https://github.com/python/cpython/issues/104909]: Split **`LOAD_ATTR_INSTANCE_VALUE`** into micro-ops.

- gh-104909 [https://github.com/python/cpython/issues/104909]: Split **`LOAD_GLOBAL`** specializations into micro-ops.

- gh-106597 [https://github.com/python/cpython/issues/106597]: A new debug structure of offsets has been added to the `_PyRuntimeState` that will help out-of-process debuggers and profilers to obtain the offsets to relevant interpreter structures in a way that is agnostic of how Python was compiled and that doesn't require copying the headers. Patch by Pablo Galindo

- gh-106487 [https://github.com/python/cpython/issues/106487]: Allow the *count* argument of **`str.replace()`** to be a keyword. Patch by Hugo van Kemenade.

- gh-96844 [https://github.com/python/cpython/issues/96844]: Improve error message of **`list.remove()`**. Patch by Donghee Na.

- gh-81283 [https://github.com/python/cpython/issues/81283]: Compiler now strips indents from docstrings. It reduces `pyc` file size 5% when the module is heavily documented. This change affects to `__doc__` so tools like doctest will be affected.

- gh-106396 [https://github.com/python/cpython/issues/106396]: When the format specification of an f-string expression is empty, the parser now generates an empty `ast.JoinedStr` node for it instead of an one-element **`ast.JoinedStr`** with an empty string `ast.Constant`.

- gh-100288 [https://github.com/python/cpython/issues/100288]: Specialize `LOAD_ATTR` for non-descriptors on the class. Adds **`LOAD_ATTR_NONDESCRIPTOR_WITH_VALUES`** and **`LOAD_ATTR_NONDESCRIPTOR_NO_DICT`**.
- gh-106008 [https://github.com/python/cpython/issues/106008]: Fix possible reference leaks when failing to optimize comparisons with `None` in the bytecode compiler.
- gh-106145 [https://github.com/python/cpython/issues/106145]: Make `end_lineno` and `end_col_offset` required on `type_param` ast nodes.
- gh-106213 [https://github.com/python/cpython/issues/106213]: Changed the way that Emscripten call trampolines work for compatibility with Wasm/JS Promise integration.
- gh-106182 [https://github.com/python/cpython/issues/106182]: `sys.getfilesystemencoding()` and `sys.getfilesystemencodeerrors` now return interned Unicode object.
- gh-106210 [https://github.com/python/cpython/issues/106210]: Removed Emscripten import trampoline as it was no longer necessary for Pyodide.
- gh-104584 [https://github.com/python/cpython/issues/104584]: Added a new, experimental, tracing optimizer and interpreter (a.k.a. «tier 2»). This currently pessimizes, so don’t use yet – this is infrastructure so we can experiment with optimizing passes. To enable it, pass `-Xuops` or set `PYTHONUOPS=1`. To get debug output, set `PYTHONUOPSDEBUG=N` where `N` is a debug level (0-4, where 0 is no debug output and 4 is excessively verbose).
- gh-105775 [https://github.com/python/cpython/issues/105775]: `LOAD_CLOSURE` is now a pseudo-op.

- gh-105730 [https://github.com/python/cpython/issues/105730]: Allow any callable other than type objects as the condition predicate in `BaseExceptionGroup.split()` and `BaseExceptionGroup.subgroup()`.

- gh-105979 [https://github.com/python/cpython/issues/105979]: Fix crash in `_imp.get_frozen_object()` due to improper exception handling.

- gh-106003 [https://github.com/python/cpython/issues/106003]: Add a new `TO_BOOL` instruction, which performs boolean conversions for `POP_JUMP_IF_TRUE`, `POP_JUMP_IF_FALSE`, and `UNARY_NOT` (which all expect exact `bool` values now). Also, modify the oparg of `COMPARE_OP` to include an optional «boolean conversion» flag.

- gh-98931 [https://github.com/python/cpython/issues/98931]: Ensure custom `SyntaxError` error messages are raised for invalid imports with multiple targets. Patch by Pablo Galindo

- gh-105724 [https://github.com/python/cpython/issues/105724]: Improve `assert` error messages by providing exact error range.

- gh-105908 [https://github.com/python/cpython/issues/105908]: Fixed bug where gh-99111 [https://github.com/python/cpython/issues/99111] breaks future import `barry_as_FLUFL` in the Python REPL.

- gh-105840 [https://github.com/python/cpython/issues/105840]: Fix possible crashes when specializing function calls with too many `__defaults__`.

- gh-105831 [https://github.com/python/cpython/issues/105831]: Fix an f-string bug, where using a debug expression (the `=` sign) that appears in the last line of a file results to the debug buffer that holds the expression text being one character too small.

- gh-105800 [https://github.com/python/cpython/issues/105800]: Correctly issue `SyntaxWarning` in f-strings if invalid sequences are used. Patch by Pablo Galindo

- gh-105340 [https://github.com/python/cpython/issues/105340]: Include the comprehension iteration variable in `locals()` inside a module- or class-scope comprehension.
- gh-105331 [https://github.com/python/cpython/issues/105331]: Raise **`ValueError`** if the `delay` argument to **`asyncio.sleep()`** is a NaN (matching **`time.sleep()`**).
- gh-105587 [https://github.com/python/cpython/issues/105587]: The runtime can't guarantee that immortal objects will not be mutated by Extensions. Thus, this modifies _PyStaticObject_CheckRefcnt to warn instead of asserting.
- gh-105564 [https://github.com/python/cpython/issues/105564]: Don't include artificil newlines in the `line` attribute of tokens in the APIs of the **`tokenize`** module. Patch by Pablo Galindo
- gh-105549 [https://github.com/python/cpython/issues/105549]: Tokenize separately `NUMBER` and `NAME` tokens that are not ambiguous. Patch by Pablo Galindo.
- gh-105588 [https://github.com/python/cpython/issues/105588]: Fix an issue that could result in crashes when compiling malformed **`ast`** nodes.
- gh-100987 [https://github.com/python/cpython/issues/100987]: Allow objects other than code objects as the «executable» in internal frames. In the long term, this can help tools like Cython and PySpy interact more efficiently. In the shorter term, it allows us to perform some optimizations more simply.
- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix bugs in the **`builtins`** module where exceptions could end up being overwritten.
- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix bug in the compiler where an exception could end up being overwritten.
- gh-105375 [https://github.com/python/cpython/issues/105375]:

Improve error handling in `PyUnicode_BuildEncodingMap()` where an exception could end up being overwritten.

- gh-105486 [https://github.com/python/cpython/issues/105486]: Change the repr of `ParamSpec` list of args in `types.GenericAlias`.

- gh-105678 [https://github.com/python/cpython/issues/105678]: Break the `MAKE_FUNCTION` instruction into two parts, `MAKE_FUNCTION` which makes the function and `SET_FUNCTION_ATTRIBUTE` which sets the attributes on the function. This makes the stack effect of `MAKE_FUNCTION` regular to ease optimization and code generation.

- gh-105435 [https://github.com/python/cpython/issues/105435]: Fix spurious newline character if file ends on a comment without a newline. Patch by Pablo Galindo

- gh-105390 [https://github.com/python/cpython/issues/105390]: Correctly raise `tokenize.TokenError` exceptions instead of `SyntaxError` for tokenize errors such as incomplete input. Patch by Pablo Galindo

- gh-105259 [https://github.com/python/cpython/issues/105259]: Don't include newline character for trailing `NEWLINE` tokens emitted in the `tokenize` module. Patch by Pablo Galindo

- gh-104635 [https://github.com/python/cpython/issues/104635]: Eliminate redundant `STORE_FAST` instructions in the compiler. Patch by Donghee Na and Carl Meyer.

- gh-105324 [https://github.com/python/cpython/issues/105324]: Fix the main function of the `tokenize` module when reading from `sys.stdin`. Patch by Pablo Galindo

- gh-33092 [https://github.com/python/cpython/issues/33092]: Simplify and speed up interpreter for f-strings. Removes `FORMAT_VALUE` opcode. Add `CONVERT_VALUE`, `FORMAT_SIMPLE` and `FORMAT_WITH_SPEC` opcode. Compiler emits more efficient sequence for each format expression.

- gh-105229 [https://github.com/python/cpython/issues/105229]: Remove remaining two-codeunit superinstructions. All remaining superinstructions only take a single codeunit, simplifying instrumentation and quickening.
- gh-105235 [https://github.com/python/cpython/issues/105235]: Prevent out-of-bounds memory access during `mmap.find()` calls.
- gh-98963 [https://github.com/python/cpython/issues/98963]: Restore the ability for a subclass of **`property`** to define `__slots__` or otherwise be dict-less by ignoring failures to set a docstring on such a class. This behavior had regressed in 3.12beta1. An **`AttributeError`** where there had not previously been one was disruptive to existing code.
- gh-104812 [https://github.com/python/cpython/issues/104812]: The «pending call» machinery now works for all interpreters, not just the main interpreter, and runs in all threads, not just the main thread. Some calls are still only done in the main thread, ergo in the main interpreter. This change does not affect signal handling nor the existing public C-API (`Py_AddPendingCall()`), which both still only target the main thread. The new functionality is meant strictly for internal use for now, since consequences of its use are not well understood yet outside some very restricted cases. This change brings the capability in line with the intention when the state was made per-interpreter several years ago.
- gh-105194 [https://github.com/python/cpython/issues/105194]: Do not escape with backslashes f-string format specifiers. Patch by Pablo Galindo
- gh-105229 [https://github.com/python/cpython/issues/105229]: Replace some dynamic superinstructions with single instruction equivalents.
- gh-105162 [https://github.com/python/cpython/issues/105162]: Fixed bug in generator.close()/throw() where an inner iterator

would be ignored when the outer iterator was instrumented.

- gh-105164 [https://github.com/python/cpython/issues/105164]: Ensure annotations are set up correctly if the only annotation in a block is within a **`match`** block. Patch by Jelle Zijlstra.
- gh-105148 [https://github.com/python/cpython/issues/105148]: Make `_PyASTOptimizeState` internal to ast_opt.c. Make `_PyAST_Optimize` take two integers instead of a pointer to this struct. This avoids the need to include pycore_compile.h in ast_opt.c.
- gh-104799 [https://github.com/python/cpython/issues/104799]: Attributes of **`ast`** nodes that are lists now default to the empty list if omitted. This means that some code that previously raised **`TypeError`** when the AST node was used will now proceed with the empty list instead. Patch by Jelle Zijlstra.
- gh-105111 [https://github.com/python/cpython/issues/105111]: Remove the old trashcan macros `Py_TRASHCAN_SAFE_BEGIN` and `Py_TRASHCAN_SAFE_END`. They should be replaced by the new macros `Py_TRASHCAN_BEGIN` and `Py_TRASHCAN_END`.
- gh-105035 [https://github.com/python/cpython/issues/105035]: Fix **`super()`** calls on types with custom **`tp_getattro`** implementation (e.g. meta-types.)
- gh-105017 [https://github.com/python/cpython/issues/105017]: Show CRLF lines in the tokenize string attribute in both NL and NEWLINE tokens. Patch by Marta Gómez.
- gh-105013 [https://github.com/python/cpython/issues/105013]: Fix handling of multiline parenthesized lambdas in **`inspect.getsource()`**. Patch by Pablo Galindo
- gh-105017 [https://github.com/python/cpython/issues/105017]: Do not include an additional final `NL` token when parsing files having CRLF lines. Patch by Marta Gómez.

- gh-104976 [https://github.com/python/cpython/issues/104976]: Ensure that trailing `DEDENT` **`tokenize.TokenInfo`** objects emitted by the **`tokenize`** module are reported as in Python 3.11. Patch by Pablo Galindo

- gh-104972 [https://github.com/python/cpython/issues/104972]: Ensure that the `line` attribute in **`tokenize.TokenInfo`** objects in the **`tokenize`** module are always correct. Patch by Pablo Galindo

- gh-104955 [https://github.com/python/cpython/issues/104955]: Fix signature for the new **`__release_buffer__()`** slot. Patch by Jelle Zijlstra.

- gh-104690 [https://github.com/python/cpython/issues/104690]: Starting new threads and process creation through **`os.fork()`** during interpreter shutdown (such as from **`atexit`** handlers) is no longer supported. It can lead to race condition between the main Python runtime thread freeing thread states while internal **`threading`** routines are trying to allocate and use the state of just created threads. Or forked children trying to use the mid-shutdown runtime and thread state in the child process.

- gh-104879 [https://github.com/python/cpython/issues/104879]: Fix crash when accessing the `__module__` attribute of type aliases defined outside a module. Patch by Jelle Zijlstra.

- gh-104825 [https://github.com/python/cpython/issues/104825]: Tokens emitted by the **`tokenize`** module do not include an implicit `\n` character in the `line` attribute anymore. Patch by Pablo Galindo

- gh-104770 [https://github.com/python/cpython/issues/104770]: If a generator returns a value upon being closed, the value is now returned by **`generator.close()`**.

- gh-89091 [https://github.com/python/cpython/issues/89091]: Raise **`RuntimeWarning`** for unawaited async generator methods like **`asend()`**, **`athrow()`** and **`aclose()`**. Patch by Kumar Aditya.

- gh-96663 [https://github.com/python/cpython/issues/96663]: Add a better, more introspect-able error message when setting attributes on classes without a `__dict__` and no slot member for the attribute.

- gh-93627 [https://github.com/python/cpython/issues/93627]: Update the Python pickle module implementation to match the C implementation of the pickle module. For objects setting reduction methods like **`__reduce_ex__()`** or **`__reduce__()`** to `None`, pickling will result in a **`TypeError`**.

- gh-101006 [https://github.com/python/cpython/issues/101006]: Improve error handling when read **`marshal`** data.

- gh-91095 [https://github.com/python/cpython/issues/91095]: Specializes calls to most Python classes. Specifically, any class that inherits from `object`, or another Python class, and does not override `__new__`.

  The specialized instruction does the following:

  1. Creates the object (by calling `object.__new__`)
  2. Pushes a shim frame to the frame stack (to cleanup after `__init__`)
  3. Pushes the frame for `__init__` to the frame stack

  Speeds up the instantiation of most Python classes.

Library

- gh-110786 [https://github.com/python/cpython/issues/110786]: **`sysconfig`**'s CLI now ignores **`BrokenPipeError`**, making it exit normally if its output is being piped and the pipe closes.

- gh-103480 [https://github.com/python/cpython/issues/103480]: The **`sysconfig`** module is now a package, instead of a single-file module.

- gh-110733 [https://github.com/python/cpython/issues/110733]: Microoptimization: Avoid calling `min()`, `max()` in **`BaseEventLoop._run_once()`**.

- gh-94597 [https://github.com/python/cpython/issues/94597]: Added **`asyncio.EventLoop`** for use with the **`asyncio.run()`** *loop_factory* kwarg to avoid calling the asyncio policy system.
- gh-110682 [https://github.com/python/cpython/issues/110682]: **`runtime-checkable protocols`** used to consider `__match_args__` a protocol member in `__instancecheck__` if it was present on the protocol. Now, this attribute is ignored if it is present.
- gh-110488 [https://github.com/python/cpython/issues/110488]: Fix a couple of issues in **`pathlib.PurePath.with_name()`**: a single dot was incorrectly considered a valid name, and in **`PureWindowsPath`**, a name with an NTFS alternate data stream, like `a:b`, was incorrectly considered invalid.
- gh-110590 [https://github.com/python/cpython/issues/110590]: Fix a bug in **`_sre.compile()`** where **`TypeError`** would be overwritten by **`OverflowError`** when the *code* argument was a list of non-ints.
- gh-65052 [https://github.com/python/cpython/issues/65052]: Prevent **`pdb`** from crashing when trying to display undisplayable objects
- gh-110519 [https://github.com/python/cpython/issues/110519]: Deprecation warning about non-integer number in **`gettext`** now always refers to the line in the user code where gettext function or method is used. Previously it could refer to a line in `gettext` code.
- gh-89902 [https://github.com/python/cpython/issues/89902]: Deprecate non-standard format specifier «N» for **`decimal.Decimal`**. It was not documented and only supported in the C implementation.
- gh-110378 [https://github.com/python/cpython/issues/110378]: **`contextmanager()`** and **`asynccontextmanager()`**

context managers now close an invalid underlying generator object that yields more then one value.

- gh-106670 [https://github.com/python/cpython/issues/106670]: In **`pdb`**, set convenience variable `$_exception` for post mortem debugging.

- gh-110365 [https://github.com/python/cpython/issues/110365]: Fix **`termios.tcsetattr()`** bug that was overwriting existing errors during parsing integers from `term` list.

- gh-109653 [https://github.com/python/cpython/issues/109653]: Slightly improve the import time of several standard-library modules by deferring imports of **`warnings`** within those modules. Patch by Alex Waygood.

- gh-110273 [https://github.com/python/cpython/issues/110273]: **`dataclasses.replace()`** now raises TypeError instead of ValueError if specify keyword argument for a field declared with init=False or miss keyword argument for required InitVar field.

- gh-110249 [https://github.com/python/cpython/issues/110249]: Add `--inline-caches` flag to `dis` command line.

- gh-109653 [https://github.com/python/cpython/issues/109653]: Fix a Python 3.12 regression in the import time of **`random`**. Patch by Alex Waygood.

- gh-110222 [https://github.com/python/cpython/issues/110222]: Add support of struct sequence objects in **`copy.replace()`**. Patched by Xuehai Pan.

- gh-109649 [https://github.com/python/cpython/issues/109649]: **`multiprocessing`**, **`concurrent.futures`**, **`compileall`**: Replace **`os.cpu_count()`** with **`os.process_cpu_count()`** to select the default number of worker threads and processes. Get the CPU affinity if supported. Patch by Victor Stinner.

- gh-110150 [https://github.com/python/cpython/issues/110150]: Fix

base case handling in statistics.quantiles. Now allows a single data point.

- gh-110036 [https://github.com/python/cpython/issues/110036]: On Windows, multiprocessing `Popen.terminate()` now catches **`PermissionError`** and get the process exit code. If the process is still running, raise again the **`PermissionError`**. Otherwise, the process terminated as expected: store its exit code. Patch by Victor Stinner.

- gh-110038 [https://github.com/python/cpython/issues/110038]: Fixed an issue that caused **`KqueueSelector.select()`** to not return all the ready events in some cases when a file descriptor is registered for both read and write.

- gh-110045 [https://github.com/python/cpython/issues/110045]: Update the **`symtable`** module to support the new scopes introduced by **PEP 695** [https://peps.python.org/pep-0695/].

- gh-88402 [https://github.com/python/cpython/issues/88402]: Add new variables to **`sysconfig.get_config_vars()`** on Windows: `LIBRARY`, `LDLIBRARY`, `LIBDIR`, `SOABI`, and `Py_NOGIL`.

- gh-109631 [https://github.com/python/cpython/issues/109631]: **`re`** functions such as **`re.findall()`**, **`re.split()`**, **`re.search()`** and **`re.sub()`** which perform short repeated matches can now be interrupted by user.

- gh-109653 [https://github.com/python/cpython/issues/109653]: Reduce the import time of **`email.utils`** by around 43%. This results in the import time of **`email.message`** falling by around 18%, which in turn reduces the import time of **`importlib.metadata`** by around 6%. Patch by Alex Waygood.

- gh-109818 [https://github.com/python/cpython/issues/109818]: Fix **`reprlib.recursive_repr()`** not copying `__type_params__` from decorated function.

- gh-109047 [https://github.com/python/cpython/issues/109047]:

`concurrent.futures`: The *executor manager thread* now catches exceptions when adding an item to the *call queue*. During Python finalization, creating a new thread can now raise `RuntimeError`. Catch the exception and call `terminate_broken()` in this case. Patch by Victor Stinner.

- gh-109782 [https://github.com/python/cpython/issues/109782]: Ensure the signature of `os.path.isdir()` is identical on all platforms. Patch by Amin Alaee.

- gh-109653 [https://github.com/python/cpython/issues/109653]: Improve import time of `functools` by around 13%. Patch by Alex Waygood.

- gh-109590 [https://github.com/python/cpython/issues/109590]: `shutil.which()` will prefer files with an extension in `PATHEXT` if the given mode includes `os.X_OK` on win32. If no `PATHEXT` match is found, a file without an extension in `PATHEXT` can be returned. This change will have **`shutil.which()`** act more similarly to previous behavior in Python 3.11.

- gh-109653 [https://github.com/python/cpython/issues/109653]: Reduce the import time of `enum` by over 50%. Patch by Alex Waygood.

- gh-109593 [https://github.com/python/cpython/issues/109593]: Avoid deadlocking on a reentrant call to the multiprocessing resource tracker. Such a reentrant call, though unlikely, can happen if a GC pass invokes the finalizer for a multiprocessing object such as SemLock.

- gh-109653 [https://github.com/python/cpython/issues/109653]: Reduce the import time of `typing` by around a third. Patch by Alex Waygood.

- gh-109649 [https://github.com/python/cpython/issues/109649]: Add `os.process_cpu_count()` function to get the number of logical CPUs usable by the calling thread of the current process. Patch by Victor Stinner.

- gh-74481 [https://github.com/python/cpython/issues/74481]: Add `set_error_mode` related constants in `msvcrt` module in Python debug build.

- gh-109613 [https://github.com/python/cpython/issues/109613]: Fix **`os.stat()`** and **`os.DirEntry.stat()`**: check for exceptions. Previously, on Python built in debug mode, these functions could trigger a fatal Python error (and abort the process) when a function succeeded with an exception set. Patch by Victor Stinner.

- gh-109599 [https://github.com/python/cpython/issues/109599]: Expose the type of PyCapsule objects as `types.CapsuleType`.

- gh-109109 [https://github.com/python/cpython/issues/109109]: You can now get the raw TLS certificate chains from TLS connections via **`ssl.SSLSocket.get_verified_chain()`** and **`ssl.SSLSocket.get_unverified_chain()`** methods.

  Contributed by Mateusz Nowak.

- gh-109559 [https://github.com/python/cpython/issues/109559]: Update **`unicodedata`** database to Unicode 15.1.0.

- gh-109543 [https://github.com/python/cpython/issues/109543]: Remove unnecessary **`hasattr()`** check during **`typing.TypedDict`** creation.

- gh-109495 [https://github.com/python/cpython/issues/109495]: Remove unnecessary extra `__slots__` in **`datetime`**'s pure python implementation to reduce memory size, as they are defined in the superclass. Patch by James Hilton-Balfe

- gh-109461 [https://github.com/python/cpython/issues/109461]: **`logging`**: Use a context manager for lock acquisition.

- gh-109096 [https://github.com/python/cpython/issues/109096]: **`http.server.CGIHTTPRequestHandler`** has been deprecated for removal in 3.15. Its design is old and the web world has long since moved beyond CGI.

- gh-109409 [https://github.com/python/cpython/issues/109409]: Fix error when it was possible to inherit a frozen dataclass from multiple parents some of which were possibly not frozen.

- gh-109375 [https://github.com/python/cpython/issues/109375]: The **`pdb`** `alias` command now prevents registering aliases without arguments.

- gh-109319 [https://github.com/python/cpython/issues/109319]: Deprecate the `dis.HAVE_ARGUMENT` field in favour of `dis.hasarg`.

- gh-107219 [https://github.com/python/cpython/issues/107219]: Fix a race condition in `concurrent.futures`. When a process in the process pool was terminated abruptly (while the future was running or pending), close the connection write end. If the call queue is blocked on sending bytes to a worker process, closing the connection write end interrupts the send, so the queue can be closed. Patch by Victor Stinner.

- gh-66143 [https://github.com/python/cpython/issues/66143]: The **`codecs.CodecInfo`** object has been made copyable and pickleable. Patched by Robert Lehmann and Furkan Onder.

- gh-109187 [https://github.com/python/cpython/issues/109187]: **`pathlib.Path.resolve()`** now treats symlink loops like other errors: in strict mode, **`OSError`** is raised, and in non-strict mode, no exception is raised.

- gh-50644 [https://github.com/python/cpython/issues/50644]: Attempts to pickle or create a shallow or deep copy of **`codecs`** streams now raise a TypeError. Previously, copying failed with a RecursionError, while pickling produced wrong results that eventually caused unpickling to fail with a RecursionError.

- gh-109174 [https://github.com/python/cpython/issues/109174]: Add support of **`types.SimpleNamespace`** in **`copy.replace()`**.

- gh-109164 [https://github.com/python/cpython/issues/109164]: **`pdb`**: Replace **`getopt`** with **`argparse`** for parsing command line arguments.

- gh-109151 [https://github.com/python/cpython/issues/109151]: Enable `readline` editing features in the sqlite3 command-line interface (`python -m sqlite3`).

- gh-108987 [https://github.com/python/cpython/issues/108987]: Fix **`_thread.start_new_thread()`** race condition. If a thread is created during Python finalization, the newly spawned thread now exits immediately instead of trying to access freed memory and lead to a crash. Patch by Victor Stinner.

- gh-108682 [https://github.com/python/cpython/issues/108682]: Enum: require `names=()` or `type=...` to create an empty enum using the functional syntax.

- gh-109033 [https://github.com/python/cpython/issues/109033]: Exceptions raised by os.utime builtin function now include the related filename

- gh-108843 [https://github.com/python/cpython/issues/108843]: Fix an issue in **`ast.unparse()`** when unparsing f-strings containing many quote types.

- gh-108469 [https://github.com/python/cpython/issues/108469]: **`ast.unparse()`** now supports new f-string syntax introduced in Python 3.12. Note that the f-string quotes are reselected for simplicity under the new syntax. (Patch by Steven Sun)

- gh-108751 [https://github.com/python/cpython/issues/108751]: Add **`copy.replace()`** function which allows to create a modified copy of an object. It supports named tuples, dataclasses, and many other objects.

- gh-108682 [https://github.com/python/cpython/issues/108682]: Enum: raise **`TypeError`** if `super().__new__()` is called from a custom `__new__`.

- gh-108278 [https://github.com/python/cpython/issues/108278]: Deprecate passing the callback callable by keyword for the following `sqlite3.Connection` APIs:

    - `set_authorizer()`
    - `set_progress_handler()`
    - `set_trace_callback()`

  The affected parameters will become positional-only in Python 3.15.

  Patch by Erlend E. Aasland.

- gh-105829 [https://github.com/python/cpython/issues/105829]: Fix concurrent.futures.ProcessPoolExecutor deadlock

- gh-108295 [https://github.com/python/cpython/issues/108295]: Fix crashes related to use of weakrefs on `typing.TypeVar`.

- gh-108463 [https://github.com/python/cpython/issues/108463]: Make expressions/statements work as expected in pdb

- gh-108277 [https://github.com/python/cpython/issues/108277]: Add `os.timerfd_create()`, `os.timerfd_settime()`, `os.timerfd_gettime()`, `os.timerfd_settime_ns()`, and `os.timerfd_gettime_ns()` to provide a low level interface for Linux's timer notification file descriptor.

- gh-107811 [https://github.com/python/cpython/issues/107811]: `tarfile`: extraction of members with overly large UID or GID (e.g. on an OS with 32-bit `id_t`) now fails in the same way as failing to set the ID.

- gh-64662 [https://github.com/python/cpython/issues/64662]: Fix support for virtual tables in `sqlite3.Connection.iterdump()`. Patch by Aviv Palivoda.

- gh-108111 [https://github.com/python/cpython/issues/108111]: Fix a regression introduced in gh-101251 [https://github.com/python/cpython/issues/101251] for 3.12, resulting in an incorrect offset calculation in `gzip.GzipFile.seek()`.

- gh-108294 [https://github.com/python/cpython/issues/108294]: `time.sleep()` now raises an auditing event.

- gh-108278 [https://github.com/python/cpython/issues/108278]: Deprecate passing name, number of arguments, and the callable as keyword arguments, for the following `sqlite3.Connection` APIs:

  - `create_function()`
  - `create_aggregate()`

  The affected parameters will become positional-only in Python 3.15.

  Patch by Erlend E. Aasland.

- gh-108322 [https://github.com/python/cpython/issues/108322]: Speed-up NormalDist.samples() by using the inverse CDF method instead of calling random.gauss().

- gh-83417 [https://github.com/python/cpython/issues/83417]: Add the ability for venv to create a `.gitignore` file which causes the created environment to be ignored by Git. It is on by default when venv is called via its CLI.

- gh-105736 [https://github.com/python/cpython/issues/105736]: Harmonized the pure Python version of `OrderedDict` with the C version. Now, both versions set up their internal state in `__new__`. Formerly, the pure Python version did the set up in `__init__`.

- gh-108083 [https://github.com/python/cpython/issues/108083]: Fix bugs in the constructor of `sqlite3.Connection` and `sqlite3.Connection.close()` where exceptions could be leaked. Patch by Erlend E. Aasland.

- gh-107932 [https://github.com/python/cpython/issues/107932]: Fix `dis` module to properly report and display bytecode that do not have source lines.

- gh-105539 [https://github.com/python/cpython/issues/105539]: `sqlite3` now emits an `ResourceWarning` if a

`sqlite3.Connection` object is not **`closed`** explicitly. Patch by Erlend E. Aasland.

- gh-107995 [https://github.com/python/cpython/issues/107995]: The `__module__` attribute on instances of **`functools.cached_property`** is now set to the name of the module in which the cached_property is defined, rather than «functools». This means that doctests in `cached_property` docstrings are now properly collected by the **`doctest`** module. Patch by Tyler Smart.
- gh-107963 [https://github.com/python/cpython/issues/107963]: Fix **`multiprocessing.set_forkserver_preload()`** to check the given list of modules names. Patch by Donghee Na.
- gh-106242 [https://github.com/python/cpython/issues/106242]: Fixes **`os.path.normpath()`** to handle embedded null characters without truncating the path.
- gh-81555 [https://github.com/python/cpython/issues/81555]: **`xml.dom.minidom`** now only quotes **`"`** in attributes.
- gh-50002 [https://github.com/python/cpython/issues/50002]: **`xml.dom.minidom`** now preserves whitespaces in attributes.
- gh-93057 [https://github.com/python/cpython/issues/93057]: Passing more than one positional argument to **`sqlite3.connect()`** and the **`sqlite3.Connection`** constructor is deprecated. The remaining parameters will become keyword-only in Python 3.15. Patch by Erlend E. Aasland.
- gh-76913 [https://github.com/python/cpython/issues/76913]: Add *merge_extra* parameter/feature to **`logging.LoggerAdapter`**
- gh-107913 [https://github.com/python/cpython/issues/107913]: Fix possible losses of `errno` and `winerror` values in **`OSError`** exceptions if they were cleared or modified by the cleanup code before creating the exception object.
- gh-107845 [https://github.com/python/cpython/issues/107845]:

`tarfile.data_filter()` now takes the location of symlinks into account when determining their target, so it will no longer reject some valid tarballs with `LinkOutsideDestinationError`.

- gh-107812 [https://github.com/python/cpython/issues/107812]: Extend socket's netlink support to the FreeBSD platform.
- gh-107805 [https://github.com/python/cpython/issues/107805]: Fix signatures of module-level generated functions in **`turtle`**.
- gh-107782 [https://github.com/python/cpython/issues/107782]: **`pydoc`** is now able to show signatures which are not representable in Python, e.g. for `getattr` and `dict.pop`.
- gh-56166 [https://github.com/python/cpython/issues/56166]: Deprecate passing optional arguments *maxsplit*, *count* and *flags* in module-level functions **`re.split()`**, **`re.sub()`** and **`re.subn()`** as positional. They should only be passed by keyword.
- gh-107710 [https://github.com/python/cpython/issues/107710]: Speed up **`logging.getHandlerNames()`**.
- gh-107715 [https://github.com/python/cpython/issues/107715]: Fix **`doctest.DocTestFinder.find()`** in presence of class names with special characters. Patch by Gertjan van Zwieten.
- gh-100814 [https://github.com/python/cpython/issues/100814]: Passing a callable object as an option value to a Tkinter image now raises the expected TclError instead of an AttributeError.
- gh-72684 [https://github.com/python/cpython/issues/72684]: Add **`tkinter`** widget methods: **`tk_busy_hold()`**, **`tk_busy_configure()`**, **`tk_busy_cget()`**, **`tk_busy_forget()`**, **`tk_busy_current()`**, and **`tk_busy_status()`**.
- gh-106684 [https://github.com/python/cpython/issues/106684]: Raise **`ResourceWarning`** when **`asyncio.StreamWriter`** is not

closed leading to memory leaks. Patch by Kumar Aditya.

- gh-107465 [https://github.com/python/cpython/issues/107465]: Add **`pathlib.Path.from_uri()`** classmethod.

- gh-107077 [https://github.com/python/cpython/issues/107077]: Seems that in some conditions, OpenSSL will return `SSL_ERROR_SYSCALL` instead of `SSL_ERROR_SSL` when a certification verification has failed, but the error parameters will still contain `ERR_LIB_SSL` and `SSL_R_CERTIFICATE_VERIFY_FAILED`. We are now detecting this situation and raising the appropriate `ssl.SSLCertVerificationError`. Patch by Pablo Galindo

- gh-107576 [https://github.com/python/cpython/issues/107576]: Fix **`types.get_original_bases()`** to only return **`__orig_bases__`** if it is present on `cls` directly. Patch by James Hilton-Balfe.

- gh-105481 [https://github.com/python/cpython/issues/105481]: Remove `opcode.is_pseudo`, `opcode.MIN_PSEUDO_OPCODE` and `opcode.MAX_PSEUDO_OPCODE`, which were added in 3.12, were never documented and were not intended to be used externally.

- gh-105481 [https://github.com/python/cpython/issues/105481]: **`opcode.ENABLE_SPECIALIZATION`** (which was added in 3.12 but never documented or intended for external usage) is moved to **`_opcode.ENABLE_SPECIALIZATION`** where tests can access it.

- gh-107396 [https://github.com/python/cpython/issues/107396]: tarfiles; Fixed use before assignment of self.exception for gzip decompression

- gh-107409 [https://github.com/python/cpython/issues/107409]: Set **`__wrapped__`** attribute in **`reprlib.recursive_repr()`**.

- gh-107406 [https://github.com/python/cpython/issues/107406]:

Implement new `__repr__()` method for `struct.Struct`. Now it returns `Struct(<format repr>)`.

- gh-107369 [https://github.com/python/cpython/issues/107369]: Optimize `textwrap.indent()`. It is ~30% faster for large input. Patch by Inada Naoki.
- gh-78722 [https://github.com/python/cpython/issues/78722]: Fix issue where `pathlib.Path.iterdir()` did not raise `OSError` until iterated.
- gh-105578 [https://github.com/python/cpython/issues/105578]: Deprecate `typing.AnyStr` in favor of the new Type Parameter syntax. See PEP 695.
- gh-62519 [https://github.com/python/cpython/issues/62519]: Make `gettext.pgettext()` search plural definitions when translation is not found.
- gh-107089 [https://github.com/python/cpython/issues/107089]: Shelves opened with `shelve.open()` have a much faster `clear()` method. Patch by James Cave.
- gh-82500 [https://github.com/python/cpython/issues/82500]: Fix overflow on 32-bit systems with `asyncio` `os.sendfile()` implementation.
- gh-83006 [https://github.com/python/cpython/issues/83006]: Document behavior of `shutil.disk_usage()` for non-mounted filesystems on Unix.
- gh-65495 [https://github.com/python/cpython/issues/65495]: Use lowercase `mail from` and `rcpt to` in `smptlib.SMTP`.
- gh-106186 [https://github.com/python/cpython/issues/106186]: Do not report `MultipartInvariantViolationDefect` defect when the `email.parser.Parser` class is used to parse emails with `headersonly=True`.
- gh-105002 [https://github.com/python/cpython/issues/105002]: Fix invalid result from `PurePath.relative_to()` method when attempting to walk a «`..`» segment in *other* with

walk_up enabled. A **`ValueError`** exception is now raised in this case.

- gh-106739 [https://github.com/python/cpython/issues/106739]: Add the `rtype_cache` to the warning message (as an addition to the type of leaked objects and the number of leaked objects already included in the message) to make debugging leaked objects easier when the multiprocessing resource tracker process finds leaked objects at shutdown. This helps more quickly identify what was leaked and/or why the leaked object was not properly cleaned up.

- gh-106751 [https://github.com/python/cpython/issues/106751]: Optimize **`SelectSelector.select()`** for many iteration case. Patch By Donghee Na.

- gh-106751 [https://github.com/python/cpython/issues/106751]: Optimize **`_PollLikeSelector.select()`** for many iteration case.

- gh-106751 [https://github.com/python/cpython/issues/106751]: Optimize **`KqueueSelector.select()`** for many iteration case. Patch By Donghee Na.

- gh-106831 [https://github.com/python/cpython/issues/106831]: Fix potential missing `NULL` check of `d2i_SSL_SESSION` result in `_ssl.c`.

- gh-105481 [https://github.com/python/cpython/issues/105481]: The various opcode lists in the **`dis`** module are now generated from bytecodes.c instead of explicitly constructed in opcode.py.

- gh-106727 [https://github.com/python/cpython/issues/106727]: Make **`inspect.getsource()`** smarter for class for same name definitions

- gh-106789 [https://github.com/python/cpython/issues/106789]: Remove import of **`pprint`** from **`sysconfig`**.

- gh-105726 [https://github.com/python/cpython/issues/105726]: Added

`__slots__` to **`contextlib.AbstractContextManager`** and **`contextlib.AbstractAsyncContextManager`** so that child classes can use `__slots__`.

- gh-106774 [https://github.com/python/cpython/issues/106774]: Update the bundled copy of pip to version 23.2.1.

- gh-106751 [https://github.com/python/cpython/issues/106751]: **`selectors`**: Optimize `EpollSelector.select()` code by moving some code outside of the loop.

- gh-106752 [https://github.com/python/cpython/issues/106752]: Fixed several bugs in zipfile.Path, including: in **`zipfile.Path.match()`**, Windows separators are no longer honored (and never were meant to be); Fixed `name/suffix/suffixes/stem` operations when no filename is present and the Path is not at the root of the zipfile; Reworked glob for performance and more correct matching behavior.

- gh-105293 [https://github.com/python/cpython/issues/105293]: Remove call to `SSL_CTX_set_session_id_context` during client side context creation in the **`ssl`** module.

- gh-106734 [https://github.com/python/cpython/issues/106734]: Disable tab completion in multiline mode of **`pdb`**

- gh-105481 [https://github.com/python/cpython/issues/105481]: Expose opcode metadata through **`_opcode`**.

- gh-106670 [https://github.com/python/cpython/issues/106670]: Add the new `exceptions` command to the Pdb debugger. It makes it possible to move between chained exceptions when using post mortem debugging.

- gh-106602 [https://github.com/python/cpython/issues/106602]: Add _copy_ and _deepcopy_ in **`enum`**

- gh-106664 [https://github.com/python/cpython/issues/106664]: **`selectors`**: Add `_SelectorMapping.get()` method and optimize `_SelectorMapping.__getitem__()`.

- gh-106628 [https://github.com/python/cpython/issues/106628]: Speed up parsing of emails by about 20% by not compiling a new regular expression for every single email.
- gh-89427 [https://github.com/python/cpython/issues/89427]: Set the environment variable `VIRTUAL_ENV_PROMPT` at **`venv`** activation, even when `VIRTUAL_ENV_DISABLE_PROMPT` is set.
- gh-106530 [https://github.com/python/cpython/issues/106530]: Revert a change to **`colorsys.rgb_to_hls()`** that caused division by zero for certain almost-white inputs. Patch by Terry Jan Reedy.
- gh-106584 [https://github.com/python/cpython/issues/106584]: Fix exit code for `unittest` if all tests are skipped. Patch by Egor Eliseev.
- gh-106566 [https://github.com/python/cpython/issues/106566]: Optimize `(?!)` (pattern which always fails) in regular expressions.
- gh-106554 [https://github.com/python/cpython/issues/106554]: **`selectors`**: Reduce Selector overhead by using a `dict.get()` to lookup file descriptors.
- gh-106558 [https://github.com/python/cpython/issues/106558]: Remove ref cycle in callers of **`convert_to_error()`** by deleting `result` from scope in a `finally` block.
- gh-100502 [https://github.com/python/cpython/issues/100502]: Add **`pathlib.PurePath.pathmod`** class attribute that stores the implementation of **`os.path`** used for low-level path operations: either `posixpath` or `ntpath`.
- gh-106527 [https://github.com/python/cpython/issues/106527]: Reduce overhead to add and remove **`asyncio`** readers and writers.
- gh-106524 [https://github.com/python/cpython/issues/106524]: Fix crash in **`_sre.template()`** with templates containing

invalid group indices.

- gh-106531 [https://github.com/python/cpython/issues/106531]: Removed `_legacy` and the names it provided from `importlib.resources`: `Resource`, `contents`, `is_resource`, `open_binary`, `open_text`, `path`, `read_binary`, and `read_text`.

- gh-106052 [https://github.com/python/cpython/issues/106052]: **`re`** module: fix the matching of possessive quantifiers in the case of a subpattern containing backtracking.

- gh-106510 [https://github.com/python/cpython/issues/106510]: Improve debug output for atomic groups in regular expressions.

- gh-106503 [https://github.com/python/cpython/issues/106503]: Fix ref cycle in **`asyncio._SelectorSocketTransport`** by removing `_write_ready` in `close`.

- gh-105497 [https://github.com/python/cpython/issues/105497]: Fix flag mask inversion when unnamed flags exist.

- gh-90876 [https://github.com/python/cpython/issues/90876]: Prevent **`multiprocessing.spawn`** from failing to *import* in environments where `sys.executable` is `None`. This regressed in 3.11 with the addition of support for path-like objects in multiprocessing.

- gh-106403 [https://github.com/python/cpython/issues/106403]: Instances of **`typing.TypeVar`**, **`typing.ParamSpec`**, **`typing.ParamSpecArgs`**, **`typing.ParamSpecKwargs`**, and **`typing.TypeVarTuple`** once again support weak references, fixing a regression introduced in Python 3.12.0 beta 1. Patch by Jelle Zijlstra.

- gh-89812 [https://github.com/python/cpython/issues/89812]: Add private `pathlib._PathBase` class, which provides experimental support for virtual filesystems, and may be made public in a future version of Python.

- gh-106292 [https://github.com/python/cpython/issues/106292]: Check for an instance-dict cached value in the `__get__()` method of `functools.cached_property()`. This better matches the pre-3.12 behavior and improves compatibility for users subclassing `functools.cached_property()` and adding a `__set__()` method.

- gh-106350 [https://github.com/python/cpython/issues/106350]: Detect possible memory allocation failure in the libtommath function `mp_init()` used by the `_tkinter` module.

- gh-106330 [https://github.com/python/cpython/issues/106330]: Fix incorrect matching of empty paths in `pathlib.PurePath.match()`. This bug was introduced in Python 3.12.0 beta 1.

- gh-106309 [https://github.com/python/cpython/issues/106309]: Deprecate `typing.no_type_check_decorator()`. No major type checker ever added support for this decorator. Patch by Alex Waygood.

- gh-102541 [https://github.com/python/cpython/issues/102541]: Make `pydoc.doc()` catch bad module `ImportError` when output stream is not `None`.

- gh-106263 [https://github.com/python/cpython/issues/106263]: Fix crash when calling `repr` with a manually constructed SignalDict object. Patch by Charlie Zhao.

- gh-106236 [https://github.com/python/cpython/issues/106236]: Replace `assert` statements with `raise RuntimeError` in `threading`, so that `_DummyThread` cannot be joined even with `-OO`.

- gh-106238 [https://github.com/python/cpython/issues/106238]: Fix rare concurrency bug in lock acquisition by the logging package.

- gh-106152 [https://github.com/python/cpython/issues/106152]: Added PY_THROW event hook for `cProfile` for generators

- gh-106075 [https://github.com/python/cpython/issues/106075]: Added `asyncio.taskgroups.__all__` to `asyncio.__all__` for export in star imports.

- gh-104527 [https://github.com/python/cpython/issues/104527]: Zipapp will now skip over appending an archive to itself.

- gh-106046 [https://github.com/python/cpython/issues/106046]: Improve the error message from **`os.fspath()`** if called on an object where `__fspath__` is set to `None`. Patch by Alex Waygood.

- gh-105987 [https://github.com/python/cpython/issues/105987]: Fix crash due to improper reference counting in **`asyncio`** eager task factory internal routines.

- gh-105974 [https://github.com/python/cpython/issues/105974]: Fix bug where a **`typing.Protocol`** class that had one or more non-callable members would raise **`TypeError`** when **`issubclass()`** was called against it, even if it defined a custom `__subclasshook__` method. The behaviour in Python 3.11 and lower – which has now been restored – was not to raise **`TypeError`** in these situations if a custom `__subclasshook__` method was defined. Patch by Alex Waygood.

- gh-96145 [https://github.com/python/cpython/issues/96145]: Reverted addition of `json.AttrDict`.

- gh-89812 [https://github.com/python/cpython/issues/89812]: Add **`pathlib.UnsupportedOperation`**, which is raised instead of **`NotImplementedError`** when a path operation isn't supported.

- gh-105808 [https://github.com/python/cpython/issues/105808]: Fix a regression introduced in gh-101251 [https://github.com/python/cpython/issues/101251] for 3.12, causing **`gzip.GzipFile.flush()`** to not flush the compressor (nor pass along the `zip_mode` argument).

- gh-105481 [https://github.com/python/cpython/issues/105481]:

`stack_effect()` no longer raises an exception if an `oparg` is provided for an `opcode` that doesn't use its arg, or when it is not provided for an `opcode` that does use it. In the latter case, the stack effect is returned for `oparg=0`.

- gh-104799 [https://github.com/python/cpython/issues/104799]: Enable **`ast.unparse()`** to unparse function and class definitions created without the new `type_params` field from **PEP 695** [https://peps.python.org/pep-0695/]. Patch by Jelle Zijlstra.

- gh-105793 [https://github.com/python/cpython/issues/105793]: Add *follow_symlinks* keyword-only argument to **`pathlib.Path.is_dir()`** and **`is_file()`**, defaulting to `True`.

- gh-105570 [https://github.com/python/cpython/issues/105570]: Deprecate two methods of creating **`typing.TypedDict`** classes with 0 fields using the functional syntax: `TD = TypedDict("TD")` and `TD = TypedDict("TD", None)`. Both will be disallowed in Python 3.15. To create a `TypedDict` class with 0 fields, either use `class TD(TypedDict): pass` or `TD = TypedDict("TD", {})`.

- gh-105745 [https://github.com/python/cpython/issues/105745]: Fix `webbrowser.Konqueror.open` method.

- gh-105733 [https://github.com/python/cpython/issues/105733]: **`ctypes`**: Deprecate undocumented **`ctypes.SetPointerType()`** and **`ctypes.ARRAY()`** functions. Patch by Victor Stinner.

- gh-105687 [https://github.com/python/cpython/issues/105687]: Remove deprecated `re.template`, `re.T`, `re.TEMPLATE`, `sre_constans.SRE_FLAG_TEMPLATE`.

- gh-105684 [https://github.com/python/cpython/issues/105684]: Supporting **`asyncio.Task.set_name()`** is now mandatory for third party task implementations. The undocumented **`_set_task_name()`** function (deprecated since 3.8) has been removed. Patch by Kumar Aditya.

- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix a bug in `_Unpickler_SetInputStream()` where an exception could end up being overwritten in case of failure.

- gh-105626 [https://github.com/python/cpython/issues/105626]: Change the default return value of `http.client.HTTPConnection.get_proxy_response_heade` to be `None` and not `{}`.

- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix bugs in `sys` where exceptions could end up being overwritten because of deferred error handling.

- gh-105605 [https://github.com/python/cpython/issues/105605]: Harden `pyexpat` error handling during module initialisation to prevent exceptions from possibly being overwritten, and objects from being dereferenced twice.

- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix bug in `decimal` where an exception could end up being overwritten.

- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix bugs in `_datetime` where exceptions could be overwritten in case of module initialisation failure.

- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix bugs in `_ssl` initialisation which could lead to leaked references and overwritten exceptions.

- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix a bug in `array.array` where an exception could end up being overwritten.

- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix bugs in `_ctypes` where exceptions could end up being overwritten.

- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix a bug in the `posix` module where an exception could be overwritten.

- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix bugs in **`_elementtree`** where exceptions could be overwritten.

- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix bugs in **`zoneinfo`** where exceptions could be overwritten.

- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix bugs in **`errno`** where exceptions could be overwritten.

- gh-105566 [https://github.com/python/cpython/issues/105566]: Deprecate creating a **`typing.NamedTuple`** class using keyword arguments to denote the fields (`NT = NamedTuple("NT", x=int, y=str)`). This will be disallowed in Python 3.15. Use the class-based syntax or the functional syntax instead.

  Two methods of creating `NamedTuple` classes with 0 fields using the functional syntax are also deprecated, and will be disallowed in Python 3.15: `NT = NamedTuple("NT")` and `NT = NamedTuple("NT", None)`. To create a `NamedTuple` class with 0 fields, either use `class NT(NamedTuple): pass` or `NT = NamedTuple("NT", [])`.

- gh-105545 [https://github.com/python/cpython/issues/105545]: Remove deprecated in 3.11 `webbrowser.MacOSXOSAScript._name` attribute.

- gh-105497 [https://github.com/python/cpython/issues/105497]: Fix flag inversion when alias/mask members exist.

- gh-105509 [https://github.com/python/cpython/issues/105509]: **`typing.Annotated`** is now implemented as an instance of `typing._SpecialForm` rather than a class. This should have no user-facing impact for users of the **`typing`** module public API.

- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix

bugs in **`pickle`** where exceptions could be overwritten.

- gh-70303 [https://github.com/python/cpython/issues/70303]: Emit **`FutureWarning`** from **`pathlib.Path.glob()`** and **`rglob()`** if the given pattern ends with «`**`». In a future Python release, patterns with this ending will match both files and directories. Add a trailing slash to only match directories.

- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix a bug in **`sqlite3`** where an exception could be overwritten in the **`collation`** callback.

- gh-105382 [https://github.com/python/cpython/issues/105382]: Remove *cafile*, *capath* and *cadefault* parameters of the **`urllib.request.urlopen()`** function, deprecated in Python 3.6. Patch by Victor Stinner.

- gh-105376 [https://github.com/python/cpython/issues/105376]: **`logging`**: Remove undocumented and untested `Logger.warn()` and `LoggerAdapter.warn()` methods and `logging.warn()` function. Deprecated since Python 3.3, they were aliases to the **`logging.Logger.warning()`** method, **`logging.LoggerAdapter.warning()`** method and **`logging.warning()`** function. Patch by Victor Stinner.

- gh-105332 [https://github.com/python/cpython/issues/105332]: Revert pickling method from by-name back to by-value.

- gh-104554 [https://github.com/python/cpython/issues/104554]: Add RTSPS scheme support in urllib.parse

- gh-105292 [https://github.com/python/cpython/issues/105292]: Add option to **`traceback.format_exception_only()`** to recurse into the nested exception of a **`BaseExceptionGroup`**.

- gh-105280 [https://github.com/python/cpython/issues/105280]: Fix bug where `isinstance([], collections.abc.Mapping)` could evaluate to `True` if garbage collection happened at the wrong time. The bug was caused by changes to the implementation of

`typing.Protocol` in Python 3.12.

- gh-80480 [https://github.com/python/cpython/issues/80480]: **`array`**: Add `'w'` typecode that represents `Py_UCS4`.

- gh-105239 [https://github.com/python/cpython/issues/105239]: Fix longstanding bug where `issubclass(object, typing.Protocol)` would evaluate to `True` in some edge cases. Patch by Alex Waygood.

- gh-104310 [https://github.com/python/cpython/issues/104310]: In the beta 1 release we added a utility function for extension module authors, to use when testing their module for support in multiple interpreters or under a per-interpreter GIL. The name of that function has changed from `allowing_all_extensions` to `_incompatible_extension_module_restrictions`. The default for the «disable_check» argument has change from `True` to `False`, to better match the new function name.

- gh-105080 [https://github.com/python/cpython/issues/105080]: Fixed inconsistent signature on derived classes for **`inspect.signature()`**

- gh-105144 [https://github.com/python/cpython/issues/105144]: Fix a recent regression in the **`typing`** module. The regression meant that doing `class Foo(X, typing.Protocol)`, where `X` was a class that had **`abc.ABCMeta`** as its metaclass, would then cause subsequent `isinstance(1, X)` calls to erroneously raise **`TypeError`**. Patch by Alex Waygood.

- gh-62948 [https://github.com/python/cpython/issues/62948]: The **`io.IOBase`** finalizer now logs the `close()` method errors with **`sys.unraisablehook`**. Previously, errors were ignored silently by default, and only logged in Python Development Mode or on Python built on debug mode. Patch by Victor Stinner.

- gh-105096 [https://github.com/python/cpython/issues/105096]: **`wave`**: Deprecate the `getmark()`, `setmark()` and `getmarkers()` methods of the **`wave.Wave_read`** and

`wave.Wave_write` classes. They will be removed in Python 3.15. Patch by Victor Stinner.

- gh-104992 [https://github.com/python/cpython/issues/104992]: Remove the untested and undocumented **`unittest.TestProgram.usageExit()`** method, deprecated in Python 3.11. Patch by Hugo van Kemenade.
- gh-104996 [https://github.com/python/cpython/issues/104996]: Improve performance of **`pathlib.PurePath`** initialisation by deferring joining of paths when multiple arguments are given.
- gh-101588 [https://github.com/python/cpython/issues/101588]: Deprecate undocumented copy/deepcopy/pickle support for itertools.
- gh-103631 [https://github.com/python/cpython/issues/103631]: Fix `pathlib.PurePosixPath(pathlib.PureWindowsPath(...))` not converting path separators to restore 3.11 compatible behavior.
- gh-104947 [https://github.com/python/cpython/issues/104947]: Make comparisons between **`pathlib.PureWindowsPath`** objects consistent across Windows and Posix to match 3.11 behavior.
- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the **`audioop`** module, deprecated in Python 3.11. Patch by Victor Stinner.
- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the **`aifc`** module, deprecated in Python 3.11. Patch by Victor Stinner.
- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the **`uu`** module, deprecated in Python 3.11. Patch by Victor Stinner.
- gh-104935 [https://github.com/python/cpython/issues/104935]: Fix bugs with the interaction between **`typing.runtime_checkable()`** and **`typing.Generic`**

that were introduced by the **PEP 695** [https://peps.python.org/pep-0695/] implementation. Patch by Jelle Zijlstra.

- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the `crypt` module and its private `_crypt` extension, deprecated in Python 3.11. Patch by Victor Stinner.
- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the `nis` module, deprecated in Python 3.11. Patch by Victor Stinner.
- gh-104898 [https://github.com/python/cpython/issues/104898]: Add missing `__slots__` to `os.PathLike`.
- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the `xdrlib` module, deprecated in Python 3.11. Patch by Victor Stinner.
- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the `nntplib` module, deprecated in Python 3.11. Patch by Victor Stinner.
- gh-104886 [https://github.com/python/cpython/issues/104886]: Remove the undocumented `configparser.LegacyInterpolation` class, deprecated in the docstring since Python 3.2, and with a deprecation warning since Python 3.11. Patch by Hugo van Kemenade.
- gh-104786 [https://github.com/python/cpython/issues/104786]: Remove kwargs-based `typing.TypedDict` creation
- gh-104876 [https://github.com/python/cpython/issues/104876]: Remove the `turtle.RawTurtle.settiltangle()` method, deprecated in docs since Python 3.1 and with a deprecation warning since Python 3.11. Patch by Hugo van Kemenade.
- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Removed the `msilib` package, deprecated in Python 3.11.

- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the `spwd` module, deprecated in Python 3.11: the python-pam [https://pypi.org/project/python-pam/] project can be used instead. Patch by Victor Stinner.

- gh-75552 [https://github.com/python/cpython/issues/75552]: Removed the `tkinter.tix` module, deprecated since Python 3.6.

- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the `chunk` module, deprecated in Python 3.11. Patch by Victor Stinner.

- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the `mailcap` module, deprecated in Python 3.11. Patch by Victor Stinner.

- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the `sunau` module, deprecated in Python 3.11. Patch by Victor Stinner.

- gh-104780 [https://github.com/python/cpython/issues/104780]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the `ossaudiodev` module, deprecated in Python 3.11. Patch Victor Stinner.

- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the `pipes` module, deprecated in Python 3.11. Patch by Victor Stinner.

- gh-104873 [https://github.com/python/cpython/issues/104873]: Add `typing.get_protocol_members()` to return the set of members defining a `typing.Protocol`. Add `typing.is_protocol()` to check whether a class is a `typing.Protocol`. Patch by Jelle Zijlstra.

- gh-104874 [https://github.com/python/cpython/issues/104874]: Document the `__name__` and `__supertype__` attributes of `typing.NewType`. Patch by Jelle Zijlstra.

- gh-104835 [https://github.com/python/cpython/issues/104835]: Removed the following `unittest` functions, deprecated in Python 3.11:

    - ○ `unittest.findTestCases()`
    - ○ `unittest.makeSuite()`
    - ○ `unittest.getTestCaseNames()`

  Use `TestLoader` methods instead:

    - ○ `unittest.TestLoader.loadTestsFromModule()`
    - ○ `unittest.TestLoader.loadTestsFromTestCase()`
    - ○ `unittest.TestLoader.getTestCaseNames()`

  Patch by Hugo van Kemenade.

- gh-104804 [https://github.com/python/cpython/issues/104804]: Remove the untested and undocumented `webbrowser` `MacOSX` class, deprecated in Python 3.11. Patch by Hugo van Kemenade.

- gh-83863 [https://github.com/python/cpython/issues/83863]: Support for using `pathlib.Path` objects as context managers has been removed. Before Python 3.9, exiting the context manager marked a path as «closed», which caused some (but not all!) methods to raise when called. Since Python 3.9, using a path as a context manager does nothing.

- gh-104799 [https://github.com/python/cpython/issues/104799]: Adjust the location of the (see **PEP 695** [https://peps.python.org/pep-0695/]) `type_params` field on `ast.ClassDef`, `ast.AsyncFunctionDef`, and `ast.FunctionDef` to better preserve backward compatibility. Patch by Jelle Zijlstra

- gh-104797 [https://github.com/python/cpython/issues/104797]: Allow `typing.Protocol` classes to inherit from `collections.abc.Buffer`. Patch by Jelle Zijlstra.

- gh-104783 [https://github.com/python/cpython/issues/104783]: Remove `locale.resetlocale()` function deprecated in Python 3.11. Patch by Victor Stinner.

- gh-104780 [https://github.com/python/cpython/issues/104780]: Remove the `2to3` program and the **`lib2to3`** module, deprecated in Python 3.11. Patch by Victor Stinner.

- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the **`telnetlib`** module, deprecated in Python 3.11. Patch by Victor Stinner.

- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the **`imghdr`** module, deprecated in Python 3.11. Patch by Victor Stinner.

- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the **`cgi`** and **`cgitb`** modules, deprecated in Python 3.11. Patch by Victor Stinner.

- gh-104773 [https://github.com/python/cpython/issues/104773]: **PEP 594** [https://peps.python.org/pep-0594/]: Remove the **`sndhdr`** module, deprecated in Python 3.11. Patch by Victor Stinner.

- gh-104372 [https://github.com/python/cpython/issues/104372]: On Linux where **`subprocess`** can use the `vfork()` syscall for faster spawning, prevent the parent process from blocking other threads by dropping the GIL while it waits for the vfork'ed child process `exec()` outcome. This prevents spawning a binary from a slow filesystem from blocking the rest of the application.

- gh-99108 [https://github.com/python/cpython/issues/99108]: We now release the GIL around built-in **`hashlib`** computations of reasonable size for the SHA families and MD5 hash functions, matching what our OpenSSL backed hash computations already does.

- gh-102613 [https://github.com/python/cpython/issues/102613]: Improve performance of **`pathlib.Path.glob()`** when expanding a pattern with a non-terminal «`**`» component by filtering walked paths through a regular expression, rather than calling **`os.scandir()`** more than once on each directory.

- gh-104399 [https://github.com/python/cpython/issues/104399]: Prepare the `_tkinter` module for building with Tcl 9.0 and future libtommath by replacing usage of deprecated functions **`mp_to_unsigned_bin_n()`** and **`mp_unsigned_bin_size()`** when necessary.

- gh-102676 [https://github.com/python/cpython/issues/102676]: Add fields `start_offset`, `cache_offset`, `end_offset`, `baseopname`, `baseopcode`, `jump_target` and `oparg` to **`dis.Instruction`**.

- gh-103558 [https://github.com/python/cpython/issues/103558]: Fixed `parent` argument validation mechanism of **`argparse`**. Improved test coverage.

- gh-103464 [https://github.com/python/cpython/issues/103464]: Provide helpful usage messages when parsing incorrect **`pdb`** commands.

- gh-103384 [https://github.com/python/cpython/issues/103384]: Generalize the regex pattern `BaseConfigurator.INDEX_PATTERN` to allow spaces and non-alphanumeric characters in keys.

- gh-103124 [https://github.com/python/cpython/issues/103124]: Added multiline statement support for **`pdb`**

- gh-101162 [https://github.com/python/cpython/issues/101162]: Forbid using **`builtins.issubclass()`** with **`types.GenericAlias`** as the first argument.

- gh-103200 [https://github.com/python/cpython/issues/103200]: Fix cache repopulation semantics of zipimport.invalidate_caches(). The cache is now repopulated upon retrieving files with an invalid cache, not when the cache is invalidated.

- gh-100061 [https://github.com/python/cpython/issues/100061]: Fix a bug that causes wrong matches for regular expressions with possessive qualifier.

- gh-77609 [https://github.com/python/cpython/issues/77609]: Add *follow_symlinks* argument to `pathlib.Path.glob()` and `rglob()`, defaulting to false.

- gh-102541 [https://github.com/python/cpython/issues/102541]: Hide traceback in `help()` prompt, when import failed.

- gh-102120 [https://github.com/python/cpython/issues/102120]: Added a stream mode to `tarfile` that allows for reading archives without caching info about the inner files.

- gh-102029 [https://github.com/python/cpython/issues/102029]: Deprecate passing any arguments to `threading.RLock()`.

- gh-88233 [https://github.com/python/cpython/issues/88233]: Refactored `zipfile._strip_extra` to use higher level abstractions for extras instead of a heavy-state loop.

- gh-102024 [https://github.com/python/cpython/issues/102024]: Reduce calls of `_idle_semaphore.release()` in `concurrent.futures.thread._worker()`.

- gh-73435 [https://github.com/python/cpython/issues/73435]: Add support for recursive wildcards in `pathlib.PurePath.match()`.

- gh-84867 [https://github.com/python/cpython/issues/84867]: `unittest.TestLoader` no longer loads test cases from exact `unittest.TestCase` and `unittest.FunctionTestCase` classes.

- gh-99203 [https://github.com/python/cpython/issues/99203]: Restore following CPython <= 3.10.5 behavior of `shutil.make_archive()`: do not create an empty archive if `root_dir` is not a directory, and, in that case, raise `FileNotFoundError` or `NotADirectoryError` regardless of `format` choice. Beyond the brought-back behavior, the function may now also raise these exceptions in `dry_run` mode.

- gh-80480 [https://github.com/python/cpython/issues/80480]: Emit

`DeprecationWarning` for `array`'s `'u'` type code, deprecated in docs since Python 3.3.

- gh-94924 [https://github.com/python/cpython/issues/94924]: `unittest.mock.create_autospec()` now properly returns coroutine functions compatible with `inspect.iscoroutinefunction()`
- gh-94777 [https://github.com/python/cpython/issues/94777]: Fix hanging `multiprocessing` `ProcessPoolExecutor` when a child process crashes while data is being written in the call queue.
- gh-92871 [https://github.com/python/cpython/issues/92871]: Remove the `typing.io` and `typing.re` namespaces, deprecated since Python 3.8. All items are still available from the main `typing` module.
- bpo-43633 [https://bugs.python.org/issue?@action=redirect&bpo=43633]: Improve the textual representation of IPv4-mapped IPv6 addresses (**RFC 4291** [https://datatracker.ietf.org/doc/html/rfc4291.html] Sections 2.2, 2.5.5.2) in `ipaddress`. Patch by Oleksandr Pavliuk.
- bpo-44850 [https://bugs.python.org/issue?@action=redirect&bpo=44850]: Improve performance of `operator.methodcaller()` using the **PEP 590** [https://peps.python.org/pep-0590/] `vectorcall` convention. Patch by Anthony Lee and Pieter Eendebak.
- bpo-44185 [https://bugs.python.org/issue?@action=redirect&bpo=44185]: `unittest.mock.mock_open()` will call the `close()` method of the file handle mock when it is exiting from the context manager. Patch by Samet Yaslan.
- bpo-40988 [https://bugs.python.org/issue?@action=redirect&bpo=40988]: Improve performance of `functools.singledispatchmethod` by caching the generated dispatch wrapper. Optimization suggested by frederico. Patch by @mental32, Alex Waygood and Pieter

Eendebak.

- bpo-41768 [https://bugs.python.org/issue?@action=redirect&bpo=41768]: `unittest.mock` speccing no longer calls class properties. Patch by Melanie Witt.

- bpo-18319 [https://bugs.python.org/issue?@action=redirect&bpo=18319]: Ensure `gettext(msg)` retrieve translations even if a plural form exists. In other words: `gettext(msg) == ngettext(msg, '', 1)`.

- bpo-17013 [https://bugs.python.org/issue?@action=redirect&bpo=17013]: Add `ThreadingMock` to `unittest.mock` that can be used to create Mock objects that can wait until they are called. Patch by Karthikeyan Singaravelan and Mario Corchero.

Documentation

- gh-109209 [https://github.com/python/cpython/issues/109209]: The minimum Sphinx version required for the documentation is now 4.2.
- gh-108826 [https://github.com/python/cpython/issues/108826]: `dis` module command-line interface is now mentioned in documentation.
- gh-107305 [https://github.com/python/cpython/issues/107305]: Add documentation for `PyInterpreterConfig` and `Py_NewInterpreterFromConfig()`. Also clarify some of the nearby docs relative to per-interpreter GIL.
- gh-107008 [https://github.com/python/cpython/issues/107008]: Document the `curses` module variables `LINES` and `COLS`.
- gh-106948 [https://github.com/python/cpython/issues/106948]: Add a number of standard external names to `nitpick_ignore`.
- gh-106232 [https://github.com/python/cpython/issues/106232]: Make timeit doc command lines compatible with Windows by using double quotes for arguments. This works on linux and macOS also.
- gh-105172 [https://github.com/python/cpython/issues/105172]: Fixed `functools.lru_cache()` docstring accounting for `typed` argument's different handling of str and int. Patch by Bar

Harel.

- gh-105052 [https://github.com/python/cpython/issues/105052]: Update `timeit` doc to specify that time in seconds is just the default.
- gh-89455 [https://github.com/python/cpython/issues/89455]: Add missing documentation for the `max_group_depth` and `max_group_width` parameters and the `exceptions` attribute of the **`traceback.TracebackException`** class.
- gh-89412 [https://github.com/python/cpython/issues/89412]: Add missing documentation for the `end_lineno` and `end_offset` attributes of the **`traceback.TracebackException`** class.
- gh-104943 [https://github.com/python/cpython/issues/104943]: Remove mentions of old Python versions in **`typing.NamedTuple`**.
- gh-54738 [https://github.com/python/cpython/issues/54738]: Add documentation on how to localize the **`argparse`** module.
- gh-102823 [https://github.com/python/cpython/issues/102823]: Document the return type of `x // y` when `x` and `y` have type **`float`**.
- gh-102759 [https://github.com/python/cpython/issues/102759]: Align function signature for `functools.reduce` in documentation and docstring with the C implementation.

Tests

- gh-110647 [https://github.com/python/cpython/issues/110647]: Fix test_stress_modifying_handlers() of test_signal. Patch by Victor Stinner.
- gh-103053 [https://github.com/python/cpython/issues/103053]: Fix test_tools.test_freeze on FreeBSD: run «make distclean» instead of «make clean» in the copied source directory to remove also the «python» program. Patch by Victor Stinner.
- gh-110167 [https://github.com/python/cpython/issues/110167]: Fix a deadlock in test_socket when server fails with a timeout but the client is still running in its thread. Don’t hold a lock to call cleanup functions in doCleanups(). One of the cleanup function waits until the client completes, whereas the client could deadlock if it called addCleanup() in such situation. Patch by Victor Stinner.

- gh-110388 [https://github.com/python/cpython/issues/110388]: Add tests for `tty`.
- gh-81002 [https://github.com/python/cpython/issues/81002]: Add tests for `termios`.
- gh-110367 [https://github.com/python/cpython/issues/110367]: regrtest: When using worker processes (-jN) with –verbose3 option, regrtest can now display the worker output even if a worker process does crash. Previously, sys.stdout and sys.stderr were replaced and so the worker output was lost on a crash. Patch by Victor Stinner.
- gh-110267 [https://github.com/python/cpython/issues/110267]: Add tests for pickling and copying PyStructSequence objects. Patched by Xuehai Pan.
- gh-110171 [https://github.com/python/cpython/issues/110171]: `libregrtest` now always sets and shows `random.seed`, so tests are more reproducible. Use `--randseed` flag to pass the explicit random seed for tests.
- gh-110152 [https://github.com/python/cpython/issues/110152]: Remove `Tools/scripts/run_tests.py` and `make hostrunnertest`. Just run `./python -m test --slow-ci`, `make buildbottest` or `make test` instead. Python test runner (regrtest) now handles cross-compilation and HOSTRUNNER. It also adds options to Python such fast `-u -E -W default -bb` when `--fast-ci` or `--slow-ci` option is used. Patch by Victor Stinner.
- gh-110031 [https://github.com/python/cpython/issues/110031]: Skip test_threading tests using thread+fork if Python is built with Address Sanitizer (ASAN). Patch by Victor Stinner.
- gh-110088 [https://github.com/python/cpython/issues/110088]: Fix test_asyncio timeouts: don’t measure the maximum duration, a test should not measure a CI performance. Only measure the minimum duration when a task has a timeout or delay. Add `CLOCK_RES` to `test_asyncio.utils`. Patch by Victor Stinner.
- gh-109974 [https://github.com/python/cpython/issues/109974]: Fix race conditions in test_threading lock tests. Wait until a condition is met rather than using `time.sleep()` with a hardcoded number of seconds. Patch by Victor Stinner.
- gh-110033 [https://github.com/python/cpython/issues/110033]: Fix `test_interprocess_signal()` of `test_signal`. Make

sure that the `subprocess.Popen` object is deleted before the test raising an exception in a signal handler. Otherwise, `Popen.__del__()` can get the exception which is logged as `Exception ignored in: ...` and the test fails. Patch by Victor Stinner.

- gh-109594 [https://github.com/python/cpython/issues/109594]: Fix test_timeout() of test_concurrent_futures.test_wait. Remove the future which may or may not complete depending if it takes longer than the timeout or not. Keep the second future which does not complete before wait() timeout. Patch by Victor Stinner.
- gh-109972 [https://github.com/python/cpython/issues/109972]: Split test_gdb.py file into a test_gdb package made of multiple tests, so tests can now be run in parallel. Patch by Victor Stinner.
- gh-109566 [https://github.com/python/cpython/issues/109566]: regrtest: When `--fast-ci` or `--slow-ci` option is used, regrtest now replaces the current process with a new process to add `-u -W default -bb -E` options to Python. Patch by Victor Stinner.
- gh-109748 [https://github.com/python/cpython/issues/109748]: Fix `test_zippath_from_non_installed_posix()` of test_venv: don't copy `__pycache__/` sub-directories, because they can be modified by other Python tests running in parallel. Patch by Victor Stinner.
- gh-109739 [https://github.com/python/cpython/issues/109739]: regrtest: Fix reference leak check on Windows. Disable the load tracker on Windows in the reference leak check mode (-R option). Patch by Victor Stinner.
- gh-109276 [https://github.com/python/cpython/issues/109276]: regrtest: When a test fails with «env changed» and the –rerun option is used, the test is now re-run in verbose mode in a fresh process. Patch by Victor Stinner.
- gh-103053 [https://github.com/python/cpython/issues/103053]: Skip test_freeze_simple_script() of test_tools.test_freeze if Python is built with `./configure --enable-optimizations`, which means with Profile Guided Optimization (PGO): it just makes the test too slow. The freeze tool is tested by many other CIs with other (faster) compiler flags. Patch by Victor Stinner.
- gh-109580 [https://github.com/python/cpython/issues/109580]: Skip

`test_perf_profiler` if Python is built with ASAN, MSAN or UBSAN sanitizer. Python does crash randomly in this test on such build. Patch by Victor Stinner.

- gh-109566 [https://github.com/python/cpython/issues/109566]: regrtest: Add `--fast-ci` and `--slow-ci` options. `--fast-ci` uses a default timeout of 10 minutes and `-u all,-cpu` (skip slowest tests). `--slow-ci` uses a default timeout of 20 minutes and `-u all` (run all tests). Patch by Victor Stinner.
- gh-109425 [https://github.com/python/cpython/issues/109425]: libregrtest now decodes stdout of test worker processes with the «backslashreplace» error handler to log corrupted stdout, instead of failing with an error and not logging the stdout. Patch by Victor Stinner.
- gh-109396 [https://github.com/python/cpython/issues/109396]: Fix `test_socket.test_hmac_sha1()` in FIPS mode. Use a longer key: FIPS mode requires at least of at least 112 bits. The previous key was only 32 bits. Patch by Victor Stinner.
- gh-104736 [https://github.com/python/cpython/issues/104736]: Fix test_gdb on Python built with LLVM clang 16 on Linux ppc64le (ex: Fedora 38). Search patterns in gdb «bt» command output to detect when gdb fails to retrieve the traceback. For example, skip a test if `Backtrace stopped: frame did not save the PC` is found. Patch by Victor Stinner.
- gh-109276 [https://github.com/python/cpython/issues/109276]: libregrtest now calls **`random.seed()`** before running each test file when `-r/--randomize` command line option is used. Moreover, it's also called in worker processes. It should help to make tests more deterministic. Previously, it was only called once in the main process before running all test files and it was not called in worker processes. Patch by Victor Stinner.
- gh-109276 [https://github.com/python/cpython/issues/109276]: libregrtest now uses a separated file descriptor to write test result as JSON. Previously, if a test wrote debug messages late around the JSON, the main test process failed to parse JSON. Patch by Victor Stinner.
- gh-108996 [https://github.com/python/cpython/issues/108996]: Fix and enable `test_msvcrt`.

- gh-109237 [https://github.com/python/cpython/issues/109237]: Fix `test_site.test_underpth_basic()` when the working directory contains at least one non-ASCII character: encode the `._pth` file to UTF-8 and enable the UTF-8 Mode to use UTF-8 for the child process stdout. Patch by Victor Stinner.
- gh-109230 [https://github.com/python/cpython/issues/109230]: Fix `test_pyexpat.test_exception()`: it can now be run from a directory different than Python source code directory. Before, the test failed in this case. Skip the test if Modules/pyexpat.c source is not available. Skip also the test on Python implementations other than CPython. Patch by Victor Stinner.
- gh-108996 [https://github.com/python/cpython/issues/108996]: Add tests for `msvcrt`.
- gh-109015 [https://github.com/python/cpython/issues/109015]: Fix test_asyncio, test_imaplib and test_socket tests on FreeBSD if the TCP blackhole is enabled (`sysctl net.inet.tcp.blackhole`). Skip the few tests which failed with `ETIMEDOUT` which such non standard configuration. Currently, the FreeBSD GCP image enables TCP and UDP blackhole [https://reviews.freebsd.org/D41751] (`sysctl net.inet.tcp.blackhole=2` and `sysctl net.inet.udp.blackhole=1`). Patch by Victor Stinner.
- gh-91960 [https://github.com/python/cpython/issues/91960]: Skip `test_gdb` if gdb is unable to retrieve Python frame objects: if a frame is `<optimized out>`. When Python is built with «clang -Og», gdb can fail to retrieve the *frame* parameter of `_PyEval_EvalFrameDefault()`. In this case, tests like `py_bt()` are likely to fail. Without getting access to Python frames, `python-gdb.py` is mostly clueless on retrieving the Python traceback. Moreover, `test_gdb` is no longer skipped on macOS if Python is built with Clang. Patch by Victor Stinner.
- gh-108962 [https://github.com/python/cpython/issues/108962]: Skip `test_tempfile.test_flags()` if `chflags()` fails with «OSError: [Errno 45] Operation not supported» (ex: on FreeBSD 13). Patch by Victor Stinner.
- gh-91960 [https://github.com/python/cpython/issues/91960]: FreeBSD 13.2 CI coverage for pull requests is now provided by Cirrus-CI (a hosted CI service that supports Linux, macOS, Windows, and FreeBSD).

- gh-89392 [https://github.com/python/cpython/issues/89392]: Removed support of `test_main()` function in tests. They now always use normal unittest test runner.
- gh-108851 [https://github.com/python/cpython/issues/108851]: Fix `test_tomllib` recursion tests for WASI buildbots: reduce the recursion limit and compute the maximum nested array/dict depending on the current available recursion limit. Patch by Victor Stinner.
- gh-108851 [https://github.com/python/cpython/issues/108851]: Add `get_recursion_available()` and `get_recursion_depth()` functions to the **`test.support`** module. Patch by Victor Stinner.
- gh-108834 [https://github.com/python/cpython/issues/108834]: Add `--fail-rerun option` option to regrtest: if a test failed when then passed when rerun in verbose mode, exit the process with exit code 2 (error), instead of exit code 0 (success). Patch by Victor Stinner.
- gh-108834 [https://github.com/python/cpython/issues/108834]: Rename regrtest `--verbose2` option (`-w`) to `--rerun`. Keep `--verbose2` as a deprecated alias. Patch by Victor Stinner.
- gh-108834 [https://github.com/python/cpython/issues/108834]: When regrtest reruns failed tests in verbose mode (`./python -m test --rerun`), tests are now rerun in fresh worker processes rather than being executed in the main process. If a test does crash or is killed by a timeout, the main process can detect and handle the killed worker process. Tests are rerun in parallel if the `-jN` option is used to run tests in parallel. Patch by Victor Stinner.
- gh-108822 [https://github.com/python/cpython/issues/108822]: `regrtest` now computes statistics on all tests: successes, failures and skipped. `test_netrc`, `test_pep646_syntax` and `test_xml_etree` now return results in their `test_main()` function. Patch by Victor Stinner and Alex Waygood.
- gh-108794 [https://github.com/python/cpython/issues/108794]: The **`doctest.DocTestRunner.run()`** method now counts the number of skipped tests. Add **`doctest.DocTestRunner.skips`** and **`doctest.TestResults.skipped`** attributes. Patch by

Victor Stinner.

- gh-108388 [https://github.com/python/cpython/issues/108388]: Convert test_concurrent_futures to a package of 7 sub-tests. Patch by Victor Stinner.
- gh-108388 [https://github.com/python/cpython/issues/108388]: Split test_multiprocessing_fork, test_multiprocessing_forkserver and test_multiprocessing_spawn into test packages. Each package is made of 4 sub-tests: processes, threads, manager and misc. It allows running more tests in parallel and so reduce the total test duration. Patch by Victor Stinner.
- gh-105776 [https://github.com/python/cpython/issues/105776]: Fix test_cppext when the C compiler command `-std=c11` option: remove `-std=` options from the compiler command. Patch by Victor Stinner.
- gh-107652 [https://github.com/python/cpython/issues/107652]: Set up CIFuzz to run fuzz targets in GitHub Actions. Patch by Illia Volochii.
- gh-107237 [https://github.com/python/cpython/issues/107237]: `test_logging`: Fix `test_udp_reconnection()` by increasing the timeout from 100 ms to 5 minutes (LONG_TIMEOUT). Patch by Victor Stinner.
- gh-107178 [https://github.com/python/cpython/issues/107178]: Add the C API test for functions in the Mapping Protocol, the Sequence Protocol and some functions in the Object Protocol.
- gh-106714 [https://github.com/python/cpython/issues/106714]: test_capi: Fix test_no_FatalError_infinite_loop() to no longer write a coredump, by using test.support.SuppressCrashReport. Patch by Victor Stinner.
- gh-104090 [https://github.com/python/cpython/issues/104090]: Avoid creating a reference to the test object in **`collectedDurations()`**.
- gh-106752 [https://github.com/python/cpython/issues/106752]: Moved tests for `zipfile.Path` into `Lib/test/test_zipfile/_path`. Made `zipfile._path` a package.
- gh-106690 [https://github.com/python/cpython/issues/106690]: Add .coveragerc to cpython repository for use with coverage package.
- gh-101634 [https://github.com/python/cpython/issues/101634]: When running the Python test suite with `-jN` option, if a worker stdout cannot be decoded from the locale encoding report a

failed testn so the exitcode is non-zero. Patch by Victor Stinner.
- gh-105084 [https://github.com/python/cpython/issues/105084]: When the Python build is configured `--with-wheel-pkg-dir`, tests requiring the `setuptools` and `wheel` wheels will search for the wheels in `WHEEL_PKG_DIR`.
- gh-81005 [https://github.com/python/cpython/issues/81005]: String tests are modified to reflect that `str` and `unicode` are merged in Python 3. Patch by Daniel Fortunov.
- gh-103186 [https://github.com/python/cpython/issues/103186]: Suppress and assert expected RuntimeWarnings in test_sys_settrace.py
- gh-69714 [https://github.com/python/cpython/issues/69714]: Add additional tests to **`calendar`** to achieve full test coverage.

Build

- gh-103053 [https://github.com/python/cpython/issues/103053]: «make check-clean-src» now also checks if the «python» program is found in the source directory: fail with an error if it does exist. Patch by Victor Stinner.

- gh-109191 [https://github.com/python/cpython/issues/109191]: Fix compile error when building with recent versions of libedit.

- gh-110276 [https://github.com/python/cpython/issues/110276]: No longer ignore **`PROFILE_TASK`** failure silently: command used by Profile Guided Optimization (PGO). Patch by Victor Stinner.

- gh-109566 [https://github.com/python/cpython/issues/109566]: Remove `make testall` target: use `make buildbottest` instead. Patch by Victor Stinner.

- gh-109740 [https://github.com/python/cpython/issues/109740]: The experimental `--disable-gil` configure flag now includes «t» (for «threaded») in extension ABI tags.

- gh-109054 [https://github.com/python/cpython/issues/109054]: Fix building the `_testcapi` extension on Linux AArch64 which requires linking to libatomic when `<cpython/`

`pyatomic.h>` is used: the `_Py_atomic_or_uint64()` function requires libatomic `__atomic_fetch_or_8()` on this platform. The configure script now checks if linking to libatomic is needed and generates a new LIBATOMIC variable used to build the _testcapi extension. Patch by Victor Stinner.

- gh-63760 [https://github.com/python/cpython/issues/63760]: Fix Solaris build: no longer redefine the `gethostname()` function. Solaris defines the function since 2005. Patch by Victor Stinner, original patch by Jakub Kulík.

- gh-108740 [https://github.com/python/cpython/issues/108740]: Fix a race condition in `make regen-all`. The `deepfreeze.c` source and files generated by Argument Clinic are now generated or updated before generating «global objects». Previously, some identifiers may miss depending on the order in which these files were generated. Patch by Victor Stinner.

- gh-108634 [https://github.com/python/cpython/issues/108634]: Python built with `configure` **`--with-trace-refs`** (tracing references) is now ABI compatible with Python release build and debug build. Patch by Victor Stinner.

- gh-85283 [https://github.com/python/cpython/issues/85283]: The `_stat` C extension is now built with the limited C API. Patch by Victor Stinner.

- gh-108447 [https://github.com/python/cpython/issues/108447]: Fix x86_64 GNU/Hurd build

- gh-107814 [https://github.com/python/cpython/issues/107814]: When calling `find_python.bat` with `-q` it did not properly silence the output of nuget. That is now fixed.

- gh-105481 [https://github.com/python/cpython/issues/105481]: Remove the make target `regen-opcode-targets`, merge its work into `regen-opcode` which repeats most of the calculation. This simplifies the code for the build and reduces code duplication.

- gh-106881 [https://github.com/python/cpython/issues/106881]: Check for `linux/limits.h` before including it in `Modules/posixmodule.c`.

- gh-95855 [https://github.com/python/cpython/issues/95855]: Refactor platform triplet detection code and add detection for MIPS soft float and musl libc.

- gh-106962 [https://github.com/python/cpython/issues/106962]: Detect MPI compilers in `configure`.

- gh-106118 [https://github.com/python/cpython/issues/106118]: Fix compilation for platforms without **`O_CLOEXEC`**. The issue was introduced with Python 3.12b1 in gh-103295 [https://github.com/python/cpython/issues/103295]. Patch by Erlend Aasland.

- gh-105875 [https://github.com/python/cpython/issues/105875]: SQLite 3.15.2 or newer is required to build the **`sqlite3`** extension module. Patch by Erlend Aasland.

- gh-90005 [https://github.com/python/cpython/issues/90005]: Fix a regression in `configure` where we could end up unintentionally linking with `libbsd`.

- gh-102404 [https://github.com/python/cpython/issues/102404]: Document how to perform a WASI build on Linux. Also add Tools/wasm/build_wasi.sh as a reference implementation of the docs.

- gh-89886 [https://github.com/python/cpython/issues/89886]: Autoconf 2.71 and aclocal 1.16.4 is now required to regenerate `configure`.

- gh-104692 [https://github.com/python/cpython/issues/104692]: Include `commoninstall` as a prerequisite for `bininstall`

  This ensures that `commoninstall` is completed before `bininstall` is started when parallel builds are used (`make -j install`), and so the `python3` symlink is only installed after all standard library modules are installed.

- gh-101538 [https://github.com/python/cpython/issues/101538]: Add

experimental wasi-threads support. Patch by Takashi Yamamoto.

Windows

- gh-110437 [https://github.com/python/cpython/issues/110437]: Allows overriding the source of VC redistributables so that releases can be guaranteed to never downgrade between updates.
- gh-109286 [https://github.com/python/cpython/issues/109286]: Update Windows installer to use SQLite 3.43.1.
- gh-82367 [https://github.com/python/cpython/issues/82367]: **`os.path.realpath()`** now resolves MS-DOS style file names even if the file is not accessible. Patch by Moonsik Park.
- gh-109991 [https://github.com/python/cpython/issues/109991]: Update Windows build to use OpenSSL 3.0.11.
- gh-106242 [https://github.com/python/cpython/issues/106242]: Fixes **`realpath()`** to behave consistently when passed a path containing an embedded null character on Windows. In strict mode, it now raises **`OSError`** instead of the unexpected **`ValueError`**, and in non-strict mode will make the path absolute.
- gh-83180 [https://github.com/python/cpython/issues/83180]: Changes the Python install manager to prefer an active virtual environment when the launched script has a shebang line using a Unix-like virtual command, even if the command requests a specific version of Python.
- gh-106844 [https://github.com/python/cpython/issues/106844]: Fix integer overflow and truncating by the null character in **`_winapi.LCMapStringEx()`** which affects **`ntpath.normcase()`**.
- gh-105436 [https://github.com/python/cpython/issues/105436]: Ensure that an empty environment block is terminated by two null characters, as is required by Windows.
- gh-105146 [https://github.com/python/cpython/issues/105146]: Updated the links at the end of the installer to point to Discourse rather than the mailing lists.
- gh-103646 [https://github.com/python/cpython/issues/103646]: When installed from the Microsoft Store, `pip` no longer defaults to

per-user installs. However, as the install directory is unwritable, it should automatically decide to do a per-user install anyway. This should resolve issues when `pip` is passed an option that conflicts with `--user`.

- gh-88745 [https://github.com/python/cpython/issues/88745]: Improve performance of `shutil.copy2()` by using the operating system's `CopyFile2` function. This may result in subtle changes to metadata copied along with some files, bringing them in line with normal OS behavior.
- gh-104820 [https://github.com/python/cpython/issues/104820]: Fixes `stat()` and related functions on file systems that do not support file ID requests. This includes FAT32 and exFAT.
- gh-104803 [https://github.com/python/cpython/issues/104803]: Add `os.path.isdevdrive()` to detect whether a path is on a Windows Dev Drive. Returns `False` on platforms that do not support Dev Drive, and is absent on non-Windows platforms.

macOS

- gh-109286 [https://github.com/python/cpython/issues/109286]: Update macOS installer to use SQLite 3.43.1.
- gh-109991 [https://github.com/python/cpython/issues/109991]: Update macOS installer to use OpenSSL 3.0.11.
- gh-99079 [https://github.com/python/cpython/issues/99079]: Update macOS installer to use OpenSSL 3.0.9.

IDLE

- gh-104719 [https://github.com/python/cpython/issues/104719]: Remove IDLE's modification of tokenize.tabsize and test other uses of tokenize data and methods.

Tools/Demos

- gh-109991 [https://github.com/python/cpython/issues/109991]: Update GitHub CI workflows to use OpenSSL 3.0.11 and multissltests to use 1.1.1w, 3.0.11, and 3.1.3.
- gh-108494 [https://github.com/python/cpython/issues/108494]: Argument Clinic [https://devguide.python.org/development-tools/clinic/] now has a partial support of the Limited API: see

documentation in the Python Developer's Guide [https://devguide.python.org/development-tools/clinic/#clinic-howto-limited-capi] Patch by Victor Stinner.

- gh-107704 [https://github.com/python/cpython/issues/107704]: It is now possible to deprecate passing keyword arguments for keyword-or-positional parameters with Argument Clinic, using the new `/ [from X.Y]` syntax. (To be read as *«positional-only from Python version X.Y»*.) See documentation in the Python Developer's Guide [https://devguide.python.org/development-tools/clinic/#clinic-howto-deprecate-keyword] for more information.
- gh-107880 [https://github.com/python/cpython/issues/107880]: Argument Clinic can now clone **`__init__()`** and **`__new__()`** methods.
- gh-104683 [https://github.com/python/cpython/issues/104683]: Add `--exclude` option to Argument Clinic CLI.
- gh-95065 [https://github.com/python/cpython/issues/95065]: Argument Clinic now supports overriding automatically generated signature by using directive `@text_signature`. See documentation in the Python Developer's Guide [https://devguide.python.org/development-tools/clinic/#clinic-howto-override-signature]
- gh-107609 [https://github.com/python/cpython/issues/107609]: Fix duplicate module check in Argument Clinic. Previously, a duplicate definition would incorrectly be silently accepted. Patch by Erlend E. Aasland.
- gh-107467 [https://github.com/python/cpython/issues/107467]: The Argument Clinic command-line tool now prints to stderr instead of stdout on failure.
- gh-106970 [https://github.com/python/cpython/issues/106970]: Fix bugs in the Argument Clinic `destination <name> clear` command; the destination buffers would never be cleared, and the `destination` directive parser would simply continue to the fault handler after processing the command. Patch by Erlend E. Aasland.
- gh-106706 [https://github.com/python/cpython/issues/106706]: Change bytecode syntax for families to remove redundant name matching pseudo syntax.
- gh-106359 [https://github.com/python/cpython/issues/106359]: Argument Clinic now explicitly forbids «kwarg splats» in

function calls used as annotations.

- gh-103186 [https://github.com/python/cpython/issues/103186]: `freeze` now fetches `CONFIG_ARGS` from the original CPython instance the Makefile uses to call utility scripts. Patch by Ijtaba Hussain.
- gh-95065 [https://github.com/python/cpython/issues/95065]: It is now possible to deprecate passing parameters positionally with Argument Clinic, using the new `* [from X.Y]` syntax. (To be read as *«keyword-only from Python version X.Y»*.) See documentation in the Python Developer's Guide [https://devguide.python.org/development-tools/clinic/#clinic-howto-deprecate-positional] for more information. Patch by Erlend E. Aasland with help from Alex Waygood, Nikita Sobolev, and Serhiy Storchaka.

C API

- gh-85283 [https://github.com/python/cpython/issues/85283]: If the **`Py_LIMITED_API`** macro is defined, **`Py_BUILD_CORE`**, **`Py_BUILD_CORE_BUILTIN`** and **`Py_BUILD_CORE_MODULE`** macros are now undefined by `<Python.h>`. Patch by Victor Stinner.

- gh-110289 [https://github.com/python/cpython/issues/110289]: Add **`PyUnicode_EqualToUTF8AndSize()`** and **`PyUnicode_EqualToUTF8()`** functions.

- gh-110235 [https://github.com/python/cpython/issues/110235]: Raise **`TypeError`** for duplicate/unknown fields in `PyStructSequence` constructor. Patched by Xuehai Pan.

- gh-110014 [https://github.com/python/cpython/issues/110014]: Remove undocumented `PY_TIMEOUT_MAX` constant from the limited C API. Patch by Victor Stinner.

- gh-109521 [https://github.com/python/cpython/issues/109521]: **`PyImport_GetImporter()`** now sets RuntimeError if it fails to get **`sys.path_hooks`** or **`sys.path_importer_cache`** or they are not list and dict correspondingly. Previously it could return NULL without setting error in obscure cases, crash or raise SystemError if these attributes have wrong type.

- gh-108724 [https://github.com/python/cpython/issues/108724]: Add **`PyMutex`** internal-only lightweight locking API.

- gh-85283 [https://github.com/python/cpython/issues/85283]: Add **`PySys_AuditTuple()`** function: similar to **`PySys_Audit()`**, but pass event arguments as a Python **`tuple`** object. Patch by Victor Stinner.

- gh-108867 [https://github.com/python/cpython/issues/108867]: Add **`PyThreadState_GetUnchecked()`** function: similar to **`PyThreadState_Get()`**, but don't kill the process with a fatal error if it is NULL. The caller is responsible to check if the result is NULL. Previously, the function was private and known as `_PyThreadState_UncheckedGet()`. Patch by Victor Stinner.

- gh-108765 [https://github.com/python/cpython/issues/108765]: `Python.h` no longer includes the `<ctype.h>` standard header file. If needed, it should now be included explicitly. For example, it provides `isalpha()` and `tolower()` functions which are locale dependent. Python provides locale independent functions, like **`Py_ISALPHA()`** and **`Py_TOLOWER()`**. Patch by Victor Stinner.

- gh-108765 [https://github.com/python/cpython/issues/108765]: `Python.h` no longer includes the `<unistd.h>` standard header file. If needed, it should now be included explicitly. For example, it provides the functions: `close()`, `getpagesize()`, `getpid()` and `sysconf()`. Patch by Victor Stinner.

- gh-108765 [https://github.com/python/cpython/issues/108765]: `Python.h` no longer includes the `<ieeefp.h>` standard header. It was included for the `finite()` function which is now provided by the `<math.h>` header. It should now be included explicitly if needed. Remove also the `HAVE_IEEEFP_H` macro. Patch by Victor Stinner.

- gh-108765 [https://github.com/python/cpython/issues/108765]:

`Python.h` no longer includes these standard header files: `<time.h>`, `<sys/select.h>` and `<sys/time.h>`. If needed, they should now be included explicitly. For example, `<time.h>` provides the `clock()` and `gmtime()` functions, `<sys/select.h>` provides the `select()` function, and `<sys/time.h>` provides the `futimes()`, `gettimeofday()` and `setitimer()` functions. Patch by Victor Stinner.

- gh-108511 [https://github.com/python/cpython/issues/108511]: Add functions **`PyObject_HasAttrWithError()`**, **`PyObject_HasAttrStringWithError()`**, **`PyMapping_HasKeyWithError()`** and **`PyMapping_HasKeyStringWithError()`**.

- gh-107073 [https://github.com/python/cpython/issues/107073]: Add **`PyObject_VisitManagedDict()`** and **`PyObject_ClearManagedDict()`** functions which must be called by the traverse and clear functions of a type using **`Py_TPFLAGS_MANAGED_DICT`** flag. Patch by Victor Stinner.

- gh-108634 [https://github.com/python/cpython/issues/108634]: Python built with `configure` **`--with-trace-refs`** (tracing references) now supports the Limited API. Patch by Victor Stinner.

- gh-108014 [https://github.com/python/cpython/issues/108014]: Add **`PyLong_AsInt()`** function: similar to **`PyLong_AsLong()`**, but store the result in a C int instead of a C long. Previously, it was known as the private function **`_PyLong_AsInt()`** (with an underscore prefix). Patch by Victor Stinner.

- gh-108314 [https://github.com/python/cpython/issues/108314]: Add **`PyDict_ContainsString()`** function: same as **`PyDict_Contains()`**, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*. Patch by Victor Stinner.

- gh-108337 [https://github.com/python/cpython/issues/108337]: Add atomic operations on additional data types in pyatomic.h.

- gh-108014 [https://github.com/python/cpython/issues/108014]: Add **`Py_IsFinalizing()`** function: check if the main Python interpreter is shutting down. Patch by Victor Stinner.

- gh-107916 [https://github.com/python/cpython/issues/107916]: C API functions **`PyErr_SetFromErrnoWithFilename()`**, **`PyErr_SetExcFromWindowsErrWithFilename()`** and **`PyErr_SetFromWindowsErrWithFilename()`** save now the error code before calling **`PyUnicode_DecodeFSDefault()`**.

- gh-107915 [https://github.com/python/cpython/issues/107915]: Such C API functions as `PyErr_SetString()`, `PyErr_Format()`, `PyErr_SetFromErrnoWithFilename()` and many others no longer crash or ignore errors if it failed to format the error message or decode the filename. Instead, they keep a corresponding error.

- gh-107810 [https://github.com/python/cpython/issues/107810]: Improve **`DeprecationWarning`** for uses of **`PyType_Spec`** with metaclasses that have custom `tp_new`.

- gh-107249 [https://github.com/python/cpython/issues/107249]: Implement the **`Py_UNUSED`** macro for Windows MSVC compiler. Patch by Victor Stinner.

- gh-107226 [https://github.com/python/cpython/issues/107226]: **`PyModule_AddObjectRef()`** is now only available in the limited API version 3.10 or later.

- gh-106320 [https://github.com/python/cpython/issues/106320]: Remove private `_PyUnicode_AsString()` alias to **`PyUnicode_AsUTF8()`**. It was kept for backward compatibility with Python 3.0 - 3.2. The **`PyUnicode_AsUTF8()`** is available since Python 3.3. The **`PyUnicode_AsUTF8String()`** function can be used to keep compatibility with Python 3.2 and older. Patch by Victor Stinner.

- gh-106572 [https://github.com/python/cpython/issues/106572]:

Convert **`PyObject_DelAttr()`** and **`PyObject_DelAttrString()`** macros to functions. Patch by Victor Stinner.

- gh-106307 [https://github.com/python/cpython/issues/106307]: Add **`PyMapping_GetOptionalItem()`** function.
- gh-106521 [https://github.com/python/cpython/issues/106521]: Add **`PyObject_GetOptionalAttr()`** and **`PyObject_GetOptionalAttrString()`** functions.
- gh-106320 [https://github.com/python/cpython/issues/106320]: Remove `_PyInterpreterState_Get()` alias to **`PyInterpreterState_Get()`** which was kept for backward compatibility with Python 3.8. Patch by Victor Stinner.
- gh-106316 [https://github.com/python/cpython/issues/106316]: Remove `cpython/pytime.h` header file: it only contained private functions. Patch by Victor Stinner.
- gh-106023 [https://github.com/python/cpython/issues/106023]: Remove private `_PyObject_FastCall()` function: use `PyObject_Vectorcall()` which is available since Python 3.8 (**PEP 590** [https://peps.python.org/pep-0590/]). Patch by Victor Stinner.
- gh-106168 [https://github.com/python/cpython/issues/106168]: If Python is built in debug mode or **`with assertions`**, **`PyTuple_SET_ITEM()`** and **`PyList_SET_ITEM()`** now check the index argument with an assertion. If the assertion fails, make sure that the size is set before. Patch by Victor Stinner.
- gh-106084 [https://github.com/python/cpython/issues/106084]: Remove the old aliases to functions calling functions which were kept for backward compatibility with Python 3.8 provisional API:
    - `_PyObject_CallMethodNoArgs()`: use `PyObject_CallMethodNoArgs()`

  - `_PyObject_CallMethodOneArg()`: use `PyObject_CallMethodOneArg()`
  - `_PyObject_CallOneArg()`: use `PyObject_CallOneArg()`
  - `_PyObject_FastCallDict()`: use `PyObject_VectorcallDict()`
  - `_PyObject_Vectorcall()`: use `PyObject_Vectorcall()`
  - `_PyObject_VectorcallMethod()`: use `PyObject_VectorcallMethod()`
  - `_PyVectorcall_Function()`: use `PyVectorcall_Function()`

  Just remove the underscore prefix to update your code. Patch by Victor Stinner.

- gh-106004 [https://github.com/python/cpython/issues/106004]: Adds **`PyDict_GetItemRef()`** and **`PyDict_GetItemStringRef()`** functions: similar to **`PyDict_GetItemWithError()`** but returning a strong reference instead of a borrowed reference. Patch by Victor Stinner.

- gh-105927 [https://github.com/python/cpython/issues/105927]: Deprecate the **`PyWeakref_GetObject()`** and **`PyWeakref_GET_OBJECT()`** functions: use the new **`PyWeakref_GetRef()`** function instead. Patch by Victor Stinner.

- gh-105927 [https://github.com/python/cpython/issues/105927]: Add **`PyWeakref_GetRef()`** function: similar to **`PyWeakref_GetObject()`** but returns a strong reference, or `NULL` if the referent is no longer live. Patch by Victor Stinner.

- gh-105922 [https://github.com/python/cpython/issues/105922]: Add **`PyImport_AddModuleRef()`**: similar to **`PyImport_AddModule()`**, but return a strong reference instead of a borrowed reference. Patch by Victor Stinner.

- gh-105227 [https://github.com/python/cpython/issues/105227]: The new **`PyType_GetDict()`** provides the dictionary for the given type object that is normally exposed by `cls.__dict__`. Normally it's sufficient to use **`tp_dict`**, but for the static builtin types **`tp_dict`** is now always `NULL`. **`PyType_GetDict()`** provides the correct dict object instead.

- gh-105375 [https://github.com/python/cpython/issues/105375]: Fix a bug in **`PyErr_WarnExplicit()`** where an exception could end up being overwritten if the API failed internally.

- gh-105603 [https://github.com/python/cpython/issues/105603]: We've renamed the new (in 3.12) `PyInterpreterConfig.own_gil` to `PyInterpreterConfig.gil` and changed the meaning of the value from «bool» to an integer with supported values of `PyInterpreterConfig_DEFAULT_GIL`, `PyInterpreterConfig_SHARED_GIL`, and `PyInterpreterConfig_OWN_GIL`. The default is «shared».

- gh-105387 [https://github.com/python/cpython/issues/105387]: In the limited C API version 3.12, **`Py_INCREF()`** and **`Py_DECREF()`** functions are now implemented as opaque function calls to hide implementation details. Patch by Victor Stinner.

- gh-105396 [https://github.com/python/cpython/issues/105396]: Deprecate the **`PyImport_ImportModuleNoBlock()`** function which is just an alias to **`PyImport_ImportModule()`** since Python 3.3. Patch by Victor Stinner.

- gh-103968 [https://github.com/python/cpython/issues/103968]: **`PyType_FromMetaclass()`** now allows metaclasses with `tp_new` set to `NULL`.

- gh-105268 [https://github.com/python/cpython/issues/105268]: Remove the old private, undocumented and untested `_PyGC_FINALIZED()` macro which was kept for backward compatibility with Python 3.8 and older. Patch by Victor Stinner.

- gh-105182 [https://github.com/python/cpython/issues/105182]: Remove `PyEval_AcquireLock()` and `PyEval_ReleaseLock()` functions, deprecated in Python 3.2. Patch by Victor Stinner.

- gh-105182 [https://github.com/python/cpython/issues/105182]: Remove `PyEval_InitThreads()` and `PyEval_ThreadsInitialized()` functions, deprecated in Python 3.9. Patch by Victor Stinner.

- gh-105145 [https://github.com/python/cpython/issues/105145]: Deprecate old Python initialization functions:

    - `PySys_ResetWarnOptions()`
    - `Py_GetExecPrefix()`
    - `Py_GetPath()`
    - `Py_GetPrefix()`
    - `Py_GetProgramFullPath()`
    - `Py_GetProgramName()`
    - `Py_GetPythonHome()`

  Patch by Victor Stinner.

- gh-85275 [https://github.com/python/cpython/issues/85275]: `PyObject_AsCharBuffer()`, `PyObject_AsReadBuffer()`, `PyObject_CheckReadBuffer()`, and `PyObject_AsWriteBuffer()` are removed. Please migrate to new buffer protocol; `PyObject_GetBuffer()` and `PyBuffer_Release()`.

- gh-105156 [https://github.com/python/cpython/issues/105156]: Deprecate the old `Py_UNICODE` and `PY_UNICODE_TYPE` types: use directly the **`wchar_t`** type instead. Since Python 3.3, `Py_UNICODE` and `PY_UNICODE_TYPE` are just aliases to **`wchar_t`**. Patch by Victor Stinner.

- gh-105145 [https://github.com/python/cpython/issues/105145]: Remove the following old functions to configure the Python initialization, deprecated in Python 3.11:

    - `PySys_AddWarnOptionUnicode()`
    - `PySys_AddWarnOption()`
    - `PySys_AddXOption()`
    - `PySys_HasWarnOptions()`
    - `PySys_SetArgvEx()`
    - `PySys_SetArgv()`
    - `PySys_SetPath()`
    - `Py_SetPath()`
    - `Py_SetProgramName()`
    - `Py_SetPythonHome()`
    - `Py_SetStandardStreamEncoding()`
    - `_Py_SetProgramFullPath()`

  Patch by Victor Stinner.

- gh-105107 [https://github.com/python/cpython/issues/105107]: Remove functions deprecated in Python 3.9.

    - `PyEval_CallObject()`, `PyEval_CallObjectWithKeywords()`: use **`PyObject_CallNoArgs()`** and **`PyObject_Call()`** (positional arguments must not be *NULL*) instead.
    - `PyEval_CallFunction()`: use **`PyObject_CallFunction()`** instead.
    - `PyEval_CallMethod()`: use **`PyObject_CallMethod()`** instead.
    - `PyCFunction_Call()`: use **`PyObject_Call()`** instead.

  Patch by Victor Stinner.

- gh-105115 [https://github.com/python/cpython/issues/105115]: `PyTypeObject.tp_bases` (and `tp_mro`) for builtin static types are now shared by all interpreters, whereas in 3.12-beta1 they were stored on `PyInterpreterState`. Also note that now the tuples are immortal objects.

- gh-105071 [https://github.com/python/cpython/issues/105071]: Add `PyUnstable_Exc_PrepReraiseStar` to the unstable C api to expose the implementation of **`except*`**.

- gh-104922 [https://github.com/python/cpython/issues/104922]: `PY_SSIZE_T_CLEAN` is no longer required to use `'#'` formats in APIs like **`PyArg_ParseTuple()`** and **`Py_BuildValue()`**. They uses `Py_ssize_t` for `'#'` regardless `PY_SSIZE_T_CLEAN`.

- gh-104584 [https://github.com/python/cpython/issues/104584]: Add an unstable C API for hooking in an optimizer. This is mainly internal, but marked «unstable» to allow third-party experimentation.

- gh-104668 [https://github.com/python/cpython/issues/104668]: Don't call **`PyOS_InputHook`** or **`PyOS_ReadlineFunctionPointer`** in subinterpreters, since it's generally difficult to avoid using global state in their registered callbacks. This also avoids situations where extensions may find themselves running in a subinterpreter they don't support (or haven't yet been loaded in).

- bpo-42327 [https://bugs.python.org/issue?@action=redirect&bpo=42327]: Add **`PyModule_Add()`** function: similar to **`PyModule_AddObjectRef()`** and **`PyModule_AddObject()`**, but always steals a reference to the value.

- bpo-40309 [https://bugs.python.org/issue?@action=redirect&bpo=40309]: Properly handle trailing spaces before closing parenthesis in **`Py_BuildValue()`** format strings.

Python 3.12.0 beta 1

Release date: 2023-05-22

Security

- gh-99889 [https://github.com/python/cpython/issues/99889]: Fixed a security in flaw in **`uu.decode()`** that could allow for directory traversal based on the input if no `out_file` was specified.

- gh-104049 [https://github.com/python/cpython/issues/104049]: Do not expose the local on-disk location in directory indexes produced by **`http.client.SimpleHTTPRequestHandler`**.
- gh-99108 [https://github.com/python/cpython/issues/99108]: Upgrade built-in **`hashlib`** SHA3 implementation to a verified implementation from the `HACL*` project. Used when OpenSSL is not present or lacks SHA3.
- gh-102153 [https://github.com/python/cpython/issues/102153]: **`urllib.parse.urlsplit()`** now strips leading C0 control and space characters following the specification for URLs defined by WHATWG in response to **CVE 2023-24329** [https://www.cve.org/CVERecord?id=CVE-2023-24329]. Patch by Illia Volochii.

Library

- gh-102856 [https://github.com/python/cpython/issues/102856]: Implement PEP 701 changes in the **`tokenize`** module. Patch by Marta Gómez Macías and Pablo Galindo Salgado

Core and Builtins

- gh-104615 [https://github.com/python/cpython/issues/104615]: Fix wrong ordering of assignments in code like `a, a = x, y`. Contributed by Carl Meyer.

- gh-104572 [https://github.com/python/cpython/issues/104572]: Improve syntax error message for invalid constructs in **PEP 695** [https://peps.python.org/pep-0695/] contexts and in annotations when `from __future__ import annotations` is active.

- gh-104482 [https://github.com/python/cpython/issues/104482]: Fix three error handling bugs in ast.c's validation of pattern matching statements.

- gh-102818 [https://github.com/python/cpython/issues/102818]: Do not add a frame to the traceback in the `sys.setprofile` and `sys.settrace` trampoline functions. This ensures that frames are not duplicated if an exception is raised in the callback function, and ensures that frames are not omitted if a C callback is used and that does not add the frame.

- gh-104405 [https://github.com/python/cpython/issues/104405]: Fix an issue where some bytecode instructions could ignore **PEP 523** [https://peps.python.org/pep-0523/] when «inlining» calls.

- gh-103082 [https://github.com/python/cpython/issues/103082]: Change behavior of `sys.monitoring.events.LINE` events in `sys.monitoring`: Line events now occur when a new line is reached dynamically, instead of using a static approximation, as before. This makes the behavior very similar to that of «line» events in `sys.settrace`. This should ease porting of tools from 3.11 to 3.12.

- gh-104263 [https://github.com/python/cpython/issues/104263]: Fix `float("nan")` to produce a quiet NaN on platforms (like MIPS) where the meaning of the signalling / quiet bit is inverted from its usual meaning. Also introduce a new macro `Py_INFINITY` matching C99's `INFINITY`, and refactor internals to rely on C99's `NAN` and `INFINITY` macros instead of hard-coding bit patterns for infinities and NaNs. Thanks Sebastian Berg.

- gh-99113 [https://github.com/python/cpython/issues/99113]: Multi-phase init extension modules may now indicate that they support running in subinterpreters that have their own GIL. This is done by using `Py_MOD_PER_INTERPRETER_GIL_SUPPORTED` as the value for the `Py_mod_multiple_interpreters` module def slot. Otherwise the module, by default, cannot be imported in such subinterpreters. (This does not affect the main interpreter or subinterpreters that do not have their own GIL.) In addition to the isolation that multi-phase init already normally requires, support for per-interpreter GIL involves one additional constraint: thread-safety. If the module has external (linked) dependencies and those libraries have any state that isn't thread-safe then the module must do the additional work to add thread-safety. This should be an uncommon case.

- gh-99113 [https://github.com/python/cpython/issues/99113]: The GIL is now (optionally) per-interpreter. This is the fundamental change for PEP 684. This is all made possible by virtue of the isolated state of each interpreter in the process. The behavior of the main interpreter remains unchanged. Likewise, interpreters created using `Py_NewInterpreter()` are not affected. To get an interpreter with its own GIL, call `Py_NewInterpreterFromConfig()`.

- gh-104108 [https://github.com/python/cpython/issues/104108]: Multi-phase init extension modules may now indicate whether or not they actually support multiple interpreters. By default such modules are expected to support use in multiple interpreters. In the uncommon case that one does not, it may use the new `Py_mod_multiple_interpreters` module def slot. A value of `0` means the module does not support them. `1` means it does. The default is `1`.

- gh-104142 [https://github.com/python/cpython/issues/104142]: Fix an issue where **`list`** or **`tuple`** repetition could fail to respect **PEP 683** [https://peps.python.org/pep-0683/].

- gh-104078 [https://github.com/python/cpython/issues/104078]: Improve the performance of **`PyObject_HasAttrString()`**

- gh-104066 [https://github.com/python/cpython/issues/104066]: Improve the performance of **`hasattr()`** for module objects with a missing attribute.

- gh-104028 [https://github.com/python/cpython/issues/104028]: Reduce object creation while calling callback function from gc. Patch by Donghee Na.

- gh-104018 [https://github.com/python/cpython/issues/104018]: Disallow the «z» format specifier in %-format of bytes objects.

- gh-102213 [https://github.com/python/cpython/issues/102213]: Fix performance loss when accessing an object's attributes with `__getattr__` defined.

- gh-103895 [https://github.com/python/cpython/issues/103895]:

Improve handling of edge cases in showing `Exception.__notes__`. Ensures that the messages always end with a newline and that string/bytes are not exploded over multiple lines. Patch by Carey Metcalfe.

- gh-103907 [https://github.com/python/cpython/issues/103907]: Don't modify the refcounts of known immortal objects (**`True`**, **`False`**, and **`None`**) in the main interpreter loop.

- gh-103899 [https://github.com/python/cpython/issues/103899]: Provide a helpful hint in the **`TypeError`** message when accidentally calling a module object that has a callable attribute of the same name (such as **`dis.dis()`** or **`datetime.datetime`**).

- gh-103845 [https://github.com/python/cpython/issues/103845]: Remove both line and instruction instrumentation before adding new ones for monitoring, to avoid newly added instrumentation being removed immediately.

- gh-103763 [https://github.com/python/cpython/issues/103763]: Implement **PEP 695** [https://peps.python.org/pep-0695/], adding syntactic support for generic classes, generic functions, and type aliases.

  A new `type X = ...` syntax is added for type aliases, which resolves at runtime to an instance of the new class `typing.TypeAliasType`. The value is lazily evaluated and is accessible through the `.__value__` attribute. This is implemented as a new AST node `ast.TypeAlias`.

  New syntax (`class X[T]: ...`, `def func[T](): ...`) is added for defining generic functions and classes. This is implemented as a new `type_params` attribute on the AST nodes for classes and functions. This node holds instances of the new AST classes `ast.TypeVar`, `ast.ParamSpec`, and `ast.TypeVarTuple`.

  `typing.TypeVar`, `typing.ParamSpec`, `typing.ParamSpecArgs`, `typing.ParamSpecKwargs`, `typing.TypeVarTuple`, and `typing.Generic` are now implemented in C rather than Python.

There are new bytecode instructions `LOAD_LOCALS`, `LOAD_CLASSDICT_OR_GLOBAL`, and `LOAD_CLASSDICT_OR_DEREF` to support correct resolution of names in class namespaces.

Patch by Eric Traut, Larry Hastings, and Jelle Zijlstra.

Library

- gh-103801 [https://github.com/python/cpython/issues/103801]: Adds three minor linting fixes to the wasm module caught that were caught by ruff.
- gh-103793 [https://github.com/python/cpython/issues/103793]: Optimized asyncio Task creation by deferring expensive string formatting (task name generation) from Task creation to the first time `get_name` is called. This makes asyncio benchmarks up to 5% faster.

Core and Builtins

- gh-102310 [https://github.com/python/cpython/issues/102310]: Change the error range for invalid bytes literals.
- gh-103590 [https://github.com/python/cpython/issues/103590]: Do not wrap a single exception raised from a `try-except*` construct in an **`ExceptionGroup`**.
- gh-103650 [https://github.com/python/cpython/issues/103650]: Change the perf map format to remove the “0x” prefix from the addresses
- gh-102856 [https://github.com/python/cpython/issues/102856]: Implement the required C tokenizer changes for PEP 701. Patch by Pablo Galindo Salgado, Lysandros Nikolaou, Batuhan Taskaya, Marta Gómez Macías and sunmy2019.
- gh-100530 [https://github.com/python/cpython/issues/100530]: Clarify the error message raised when the called part of a class pattern isn’t actually a class.
- gh-101517 [https://github.com/python/cpython/issues/101517]: Fix bug in line numbers of instructions emitted for **`except*`**.
- gh-103492 [https://github.com/python/cpython/issues/103492]:

Clarify **`SyntaxWarning`** with literal `is` comparison by specifying which literal is problematic, since comparisons using `is` with e.g. `None` and bool literals are idiomatic.

- gh-87729 [https://github.com/python/cpython/issues/87729]: Add **`LOAD_SUPER_ATTR`** (and a specialization for `super().method()`) to speed up `super().method()` and `super().attr`. This makes `super().method()` roughly 2.3x faster and brings it within 20% of the performance of a simple method call. Patch by Vladimir Matveev and Carl Meyer.
- gh-103488 [https://github.com/python/cpython/issues/103488]: Change the internal offset distinguishing yield and return target addresses, so that the instruction pointer is correct for exception handling and other stack unwinding.
- gh-82012 [https://github.com/python/cpython/issues/82012]: The bitwise inversion operator (~) on bool is deprecated. It returns the bitwise inversion of the underlying `int` representation such that `bool(~True) == True`, which can be confusing. Use `not` for logical negation of bools. In the rare case that you really need the bitwise inversion of the underlying `int`, convert to int explicitly `~int(x)`.
- gh-77757 [https://github.com/python/cpython/issues/77757]: Exceptions raised in a typeobject's `__set_name__` method are no longer wrapped by a **`RuntimeError`**. Context information is added to the exception as a **PEP 678** [https://peps.python.org/pep-0678/] note.
- gh-103333 [https://github.com/python/cpython/issues/103333]: **`AttributeError`** now retains the `name` attribute when pickled and unpickled.
- gh-103242 [https://github.com/python/cpython/issues/103242]: Migrate **`set_ecdh_curve()`** method not to use deprecated OpenSSL APIs. Patch by Donghee Na.
- gh-103323 [https://github.com/python/cpython/issues/103323]: We've replaced our use of `_PyRuntime.tstate_current` with a thread-local variable. This is a fairly low-level implementation detail, and there should be no change in behavior.
- gh-84436 [https://github.com/python/cpython/issues/84436]: The implementation of PEP-683 which adds Immortal Objects by using a fixed reference count that skips reference counting to

make objects truly immutable.

- gh-102700 [https://github.com/python/cpython/issues/102700]: Allow built-in modules to be submodules. This allows submodules to be statically linked into a CPython binary.
- gh-103082 [https://github.com/python/cpython/issues/103082]: Implement **PEP 669** [https://peps.python.org/pep-0669/] Low Impact Monitoring for CPython.
- gh-88691 [https://github.com/python/cpython/issues/88691]: Reduce the number of inline **`CACHE`** entries for **`CALL`**.
- gh-102500 [https://github.com/python/cpython/issues/102500]: Make the buffer protocol accessible in Python code using the new `__buffer__` and `__release_buffer__` magic methods. See **PEP 688** [https://peps.python.org/pep-0688/] for details. Patch by Jelle Zijlstra.
- gh-97933 [https://github.com/python/cpython/issues/97933]: **PEP 709** [https://peps.python.org/pep-0709/]: inline list, dict and set comprehensions to improve performance and reduce bytecode size.
- gh-99184 [https://github.com/python/cpython/issues/99184]: Bypass instance attribute access of `__name__` in `repr` of **`weakref.ref`**.
- gh-98003 [https://github.com/python/cpython/issues/98003]: Complex function calls are now faster and consume no C stack space.
- bpo-39610 [https://bugs.python.org/issue?@action=redirect&bpo=39610]: `len()` for 0-dimensional **`memoryview`** objects (such as `memoryview(ctypes.c_uint8(42))`) now raises a **`TypeError`**. Previously this returned `1`, which was not consistent with `mem_0d[0]` raising an **`IndexError`**.

Library

- bpo-31821 [https://bugs.python.org/issue?@action=redirect&bpo=31821]: Fix **`pause_reading()`** to work when called from **`connection_made()`** in **`asyncio`**.
- gh-104600 [https://github.com/python/cpython/issues/104600]: **`functools.update_wrapper()`** now sets the `__type_params__` attribute (added by **PEP 695** [https://peps.python.org/pep-0695/]).
- gh-104340 [https://github.com/python/cpython/issues/104340]: When

an `asyncio` pipe protocol loses its connection due to an error, and the caller doesn't await `wait_closed()` on the corresponding `StreamWriter`, don't log a warning about an exception that was never retrieved. After all, according to the `StreamWriter.close()` docs, the `wait_closed()` call is optional («not mandatory»).

- gh-104555 [https://github.com/python/cpython/issues/104555]: Fix issue where an **`issubclass()`** check comparing a class `X` against a **`runtime-checkable protocol`** `Y` with non-callable members would not cause **`TypeError`** to be raised if an **`isinstance()`** call had previously been made comparing an instance of `X` to `Y`. This issue was present in edge cases on Python 3.11, but became more prominent in 3.12 due to some unrelated changes that were made to runtime-checkable protocols. Patch by Alex Waygood.
- gh-104372 [https://github.com/python/cpython/issues/104372]: Refactored the `_posixsubprocess` internals to avoid Python C API usage between fork and exec when marking `pass_fds=` file descriptors inheritable.
- gh-104484 [https://github.com/python/cpython/issues/104484]: Added *case_sensitive* argument to **`pathlib.PurePath.match()`**
- gh-75367 [https://github.com/python/cpython/issues/75367]: Fix data descriptor detection in **`inspect.getattr_static()`**.
- gh-104536 [https://github.com/python/cpython/issues/104536]: Fix a race condition in the internal **`multiprocessing.process`** cleanup logic that could manifest as an unintended `AttributeError` when calling `process.close()`.
- gh-103857 [https://github.com/python/cpython/issues/103857]: Update datetime deprecations" stracktrace to point to the calling line
- gh-101520 [https://github.com/python/cpython/issues/101520]: Move the core functionality of the `tracemalloc` module in the `Python/` folder, leaving just the module wrapper in `Modules/`.
- gh-104392 [https://github.com/python/cpython/issues/104392]: Remove undocumented and unused `_paramspec_tvars` attribute from some classes in **`typing`**.
- gh-102613 [https://github.com/python/cpython/issues/102613]: Fix issue where **`pathlib.Path.glob()`** raised **`RecursionError`** when walking deep directory trees.

- gh-103000 [https://github.com/python/cpython/issues/103000]: Improve performance of `dataclasses.asdict()` for the common case where *dict_factory* is `dict`. Patch by David C Ellis.
- gh-104301 [https://github.com/python/cpython/issues/104301]: Allow leading whitespace in disambiguated statements in `pdb`.
- gh-104139 [https://github.com/python/cpython/issues/104139]: Teach **`urllib.parse.unsplit()`** to retain the `"//"` when assembling `itms-services://?action=generate-bugs` style Apple Platform Deployment [https://support.apple.com/en-gb/guide/deployment/depce7cefc4d/web] URLs.
- gh-104307 [https://github.com/python/cpython/issues/104307]: `socket.getnameinfo()` now releases the GIL while contacting the DNS server
- gh-104310 [https://github.com/python/cpython/issues/104310]: Users may now use `importlib.util.allowing_all_extensions()` (a context manager) to temporarily disable the strict compatibility checks for importing extension modules in subinterpreters.
- gh-87695 [https://github.com/python/cpython/issues/87695]: Fix issue where `pathlib.Path.glob()` raised `OSError` when it encountered a symlink to an overly long path.
- gh-104265 [https://github.com/python/cpython/issues/104265]: Prevent possible crash by disallowing instantiation of the **`_csv.Reader`** and **`_csv.Writer`** types. The regression was introduced in 3.10.0a4 with PR 23224 (bpo-14935 [https://bugs.python.org/issue?@action=redirect&bpo=14935]). Patch by Radislav Chugunov.
- gh-102613 [https://github.com/python/cpython/issues/102613]: Improve performance of `pathlib.Path.glob()` when expanding recursive wildcards (»`**`») by merging adjacent wildcards and de-duplicating results only when necessary.
- gh-65772 [https://github.com/python/cpython/issues/65772]: Remove unneeded comments and code in turtle.py.
- gh-90208 [https://github.com/python/cpython/issues/90208]: Fixed issue where `pathlib.Path.glob()` returned incomplete results when it encountered a `PermissionError`. This method now suppresses all `OSError` exceptions, except those raised from calling `is_dir()` on the top-level path.

- gh-104144 [https://github.com/python/cpython/issues/104144]: Optimize `asyncio.TaskGroup` when using `asyncio.eager_task_factory()`. Skip scheduling a done callback if a TaskGroup task completes eagerly.
- gh-104144 [https://github.com/python/cpython/issues/104144]: Optimize `asyncio.gather()` when using `asyncio.eager_task_factory()` to complete eagerly if all fututres completed eagerly. Avoid scheduling done callbacks for futures that complete eagerly.
- gh-104114 [https://github.com/python/cpython/issues/104114]: Fix issue where `pathlib.Path.glob()` returns paths using the case of non-wildcard segments for corresponding path segments, rather than the real filesystem case.
- gh-104104 [https://github.com/python/cpython/issues/104104]: Improve performance of `pathlib.Path.glob()` by using `re.IGNORECASE` to implement case-insensitive matching.
- gh-104102 [https://github.com/python/cpython/issues/104102]: Improve performance of `pathlib.Path.glob()` when evaluating patterns that contain `'../'` segments.
- gh-103822 [https://github.com/python/cpython/issues/103822]: Update the return type of `weekday` to the newly added Day attribute
- gh-103629 [https://github.com/python/cpython/issues/103629]: Update the `repr` of `typing.Unpack` according to **PEP 692** [https://peps.python.org/pep-0692/].
- gh-103963 [https://github.com/python/cpython/issues/103963]: Make `dis` display the names of the args for `CALL_INTRINSIC_*`.
- gh-104035 [https://github.com/python/cpython/issues/104035]: Do not ignore user-defined `__getstate__` and `__setstate__` methods for slotted frozen dataclasses.
- gh-103987 [https://github.com/python/cpython/issues/103987]: In `mmap`, fix several bugs that could lead to access to memory-mapped files after they have been invalidated.
- gh-103977 [https://github.com/python/cpython/issues/103977]: Improve import time of `platform` module.
- gh-88773 [https://github.com/python/cpython/issues/88773]: Added `turtle.teleport()` to the `turtle` module to move a turtle to a new point without tracing a line, visible or invisible. Patch by Liam Gersten.
- gh-103935 [https://github.com/python/cpython/issues/103935]: Use

`io.open_code()` for files to be executed instead of raw `open()`

- gh-68968 [https://github.com/python/cpython/issues/68968]: Fixed garbled output of `assertEqual()` when an input lacks final newline.
- gh-100370 [https://github.com/python/cpython/issues/100370]: Fix potential `OverflowError` in `sqlite3.Connection.blobopen()` for 32-bit builds. Patch by Erlend E. Aasland.
- gh-102628 [https://github.com/python/cpython/issues/102628]: Substitute CTRL-D with CTRL-Z in `sqlite3` CLI banner when running on Windows.
- gh-103636 [https://github.com/python/cpython/issues/103636]: Module-level attributes `January` and `February` are deprecated from `calendar`.
- gh-103583 [https://github.com/python/cpython/issues/103583]: Isolate `_multibytecodec` and codecs extension modules. Patches by Erlend E. Aasland.
- gh-103848 [https://github.com/python/cpython/issues/103848]: Add checks to ensure that `[` bracketed `]` hosts found by `urllib.parse.urlsplit()` are of IPv6 or IPvFuture format.
- gh-103872 [https://github.com/python/cpython/issues/103872]: Update the bundled copy of pip to version 23.1.2.
- gh-99944 [https://github.com/python/cpython/issues/99944]: Make `dis` display the value of oparg of `KW_NAMES`.
- gh-74940 [https://github.com/python/cpython/issues/74940]: The C.UTF-8 locale is no longer converted to en_US.UTF-8, enabling the use of UTF-8 encoding on systems which have no locales installed.
- gh-103861 [https://github.com/python/cpython/issues/103861]: Fix `zipfile.Zipfile` creating invalid zip files when `force_zip64` was used to add files to them. Patch by Carey Metcalfe.
- gh-103857 [https://github.com/python/cpython/issues/103857]: Deprecated `datetime.datetime.utcnow()` and `datetime.datetime.utcfromtimestamp()`. (Patch by Paul Ganssle)
- gh-103839 [https://github.com/python/cpython/issues/103839]: Avoid compilation error due to tommath.h not being found when

building Tkinter against Tcl 8.7 built with bundled libtommath.

- gh-103791 [https://github.com/python/cpython/issues/103791]: **`contextlib.suppress`** now supports suppressing exceptions raised as part of an **`ExceptionGroup`**. If other exceptions exist on the group, they are re-raised in a group that does not contain the suppressed exceptions.
- gh-90750 [https://github.com/python/cpython/issues/90750]: Use **`datetime.datetime.fromisocalendar()`** in the implementation of **`datetime.datetime.strptime()`**, which should now accept only valid ISO dates. (Patch by Paul Ganssle)
- gh-103685 [https://github.com/python/cpython/issues/103685]: Prepare **`tkinter.Menu.index()`** for Tk 8.7 so that it does not raise `TclError: expected integer but got ""` when it should return `None`.
- gh-81403 [https://github.com/python/cpython/issues/81403]: **`urllib.request.CacheFTPHandler`** no longer raises **`URLError`** if a cached FTP instance is reused. ftplib's endtransfer method calls voidresp to drain the connection to handle FTP instance reuse properly.
- gh-103699 [https://github.com/python/cpython/issues/103699]: Add `__orig_bases__` to non-generic TypedDicts, call-based TypedDicts, and call-based NamedTuples. Other TypedDicts and NamedTuples already had the attribute.
- gh-103693 [https://github.com/python/cpython/issues/103693]: Add convenience variable feature to **`pdb`**
- gh-92248 [https://github.com/python/cpython/issues/92248]: Deprecate `type`, `choices`, and `metavar` parameters of `argparse.BooleanOptionalAction`.
- gh-89415 [https://github.com/python/cpython/issues/89415]: Add **`socket`** constants for source-specific multicast. Patch by Reese Hyde.
- gh-103673 [https://github.com/python/cpython/issues/103673]: **`socketserver`** gains `ForkingUnixStreamServer` and `ForkingUnixDatagramServer` classes. Patch by Jay Berry.
- gh-103636 [https://github.com/python/cpython/issues/103636]: Added Enum for months and days in the calendar module.
- gh-84976 [https://github.com/python/cpython/issues/84976]: Create a new `Lib/_pydatetime.py` file that defines the Python

version of the `datetime` module, and make `datetime` import the contents of the new library only if the C implementation is missing. Currently, the full Python implementation is defined and then deleted if the C implementation is not available, slowing down `import datetime` unnecessarily.

- gh-103596 [https://github.com/python/cpython/issues/103596]: Attributes/methods are no longer shadowed by same-named enum members, although they may be shadowed by enum.property’s.
- gh-103584 [https://github.com/python/cpython/issues/103584]: Updated `importlib.metadata` with changes from `importlib_metadata` 5.2 through 6.5.0, including: Support `installed-files.txt` for `Distribution.files` when present. `PackageMetadata` now stipulates an additional `get` method allowing for easy querying of metadata keys that may not be present. `packages_distributions` now honors packages and modules with Python modules that not `.py` sources (e.g. `.pyc`, `.so`). Expand protocol for `PackageMetadata.get_all` to match the upstream implementation of `email.message.Message.get_all` in python/typeshed#9620. Deprecated use of `Distribution` without defining abstract methods. Deprecated expectation that `PackageMetadata.__getitem__` will return `None` for missing keys. In the future, it will raise a `KeyError`.
- gh-103578 [https://github.com/python/cpython/issues/103578]: Fixed a bug where **`pdb`** crashes when reading source file with different encoding by replacing **`io.open()`** with **`io.open_code()`**. The new method would also call into the hook set by **`PyFile_SetOpenCodeHook()`**.
- gh-103556 [https://github.com/python/cpython/issues/103556]: Now creating **`inspect.Signature`** objects with positional-only parameter with a default followed by a positional-or-keyword parameter without one is impossible.
- gh-103559 [https://github.com/python/cpython/issues/103559]: Update the bundled copy of pip to version 23.1.1.
- gh-103548 [https://github.com/python/cpython/issues/103548]: Improve performance of **`pathlib.Path.absolute()`** and **`cwd()`** by joining paths only when necessary. Also improve

performance of **`pathlib.PurePath.is_absolute()`** on Posix by skipping path parsing and normalization.

- gh-103538 [https://github.com/python/cpython/issues/103538]: Remove `_tkinter` module code guarded by definition of the `TK_AQUA` macro which was only needed for Tk 8.4.7 or earlier and was never actually defined by any build system or documented for manual use.
- gh-103533 [https://github.com/python/cpython/issues/103533]: Update **`cProfile`** to use PEP 669 API
- gh-103525 [https://github.com/python/cpython/issues/103525]: Fix misleading exception message when mixed `str` and `bytes` arguments are supplied to **`pathlib.PurePath`** and **`Path`**.
- gh-103489 [https://github.com/python/cpython/issues/103489]: Add **`getconfig()`** and **`setconfig()`** to **`Connection`** to make configuration changes to a database connection. Patch by Erlend E. Aasland.
- gh-103365 [https://github.com/python/cpython/issues/103365]: Set default Flag boundary to `STRICT` and fix bitwise operations.
- gh-103472 [https://github.com/python/cpython/issues/103472]: Avoid a potential **`ResourceWarning`** in **`http.client.HTTPConnection`** by closing the proxy / tunnel's CONNECT response explicitly.
- gh-103462 [https://github.com/python/cpython/issues/103462]: Fixed an issue with using **`writelines()`** in **`asyncio`** to send very large payloads that exceed the amount of data that can be written in one call to **`socket.socket.send()`** or **`socket.socket.sendmsg()`**, resulting in the remaining buffer being left unwritten.
- gh-103449 [https://github.com/python/cpython/issues/103449]: Fix a bug in doc string generation in **`dataclasses.dataclass()`**.
- gh-103092 [https://github.com/python/cpython/issues/103092]: Isolate **`_collections`** (apply **PEP 687** [https://peps.python.org/pep-0687/]). Patch by Erlend E. Aasland.
- gh-103357 [https://github.com/python/cpython/issues/103357]: Added support for **`logging.Formatter`** `defaults` parameter to **`logging.config.dictConfig()`** and **`logging.config.fileConfig()`**. Patch by Bar Harel.
- gh-103092 [https://github.com/python/cpython/issues/103092]: Adapt the **`winreg`** extension module to **PEP 687** [https://

peps.python.org/pep-0687/].

- gh-74690 [https://github.com/python/cpython/issues/74690]: The performance of **`isinstance()`** checks against **`runtime-checkable protocols`** has been considerably improved for protocols that only have a few members. To achieve this improvement, several internal implementation details of the **`typing`** module have been refactored, including `typing._ProtocolMeta.__instancecheck__`, `typing._is_callable_members_only`, and `typing._get_protocol_attrs`. Patches by Alex Waygood.
- gh-74690 [https://github.com/python/cpython/issues/74690]: The members of a runtime-checkable protocol are now considered «frozen» at runtime as soon as the class has been created. See «What's new in Python 3.12» for more details.
- gh-103256 [https://github.com/python/cpython/issues/103256]: Fixed a bug that caused **`hmac`** to raise an exception when the requested hash algorithm was not available in OpenSSL despite being available separately as part of `hashlib` itself. It now falls back properly to the built-in. This could happen when, for example, your OpenSSL does not include SHA3 support and you want to compute `hmac.digest(b'K', b'M', 'sha3_256')`.
- gh-102778 [https://github.com/python/cpython/issues/102778]: Support `sys.last_exc` in **`idlelib`**.
- gh-103285 [https://github.com/python/cpython/issues/103285]: Improve performance of **`ast.get_source_segment()`**.
- gh-103225 [https://github.com/python/cpython/issues/103225]: Fix a bug in **`pdb`** when displaying line numbers of module-level source code.
- gh-103092 [https://github.com/python/cpython/issues/103092]: Adapt the **`msvcrt`** extension module to **PEP 687** [https://peps.python.org/pep-0687/].
- gh-103092 [https://github.com/python/cpython/issues/103092]: Adapt the **`winsound`** extension module to **PEP 687** [https://peps.python.org/pep-0687/].
- gh-93910 [https://github.com/python/cpython/issues/93910]: Remove deprecation of enum `member.member` access.
- gh-102978 [https://github.com/python/cpython/issues/102978]: Fixes **`unittest.mock.patch()`** not enforcing function signatures

for methods decorated with `@classmethod` or `@staticmethod` when patch is called with `autospec=True`.

- gh-103092 [https://github.com/python/cpython/issues/103092]: Isolate **`_socket`** (apply **PEP 687** [https://peps.python.org/pep-0687/]). Patch by Erlend E. Aasland.
- gh-100479 [https://github.com/python/cpython/issues/100479]: Add **`pathlib.PurePath.with_segments()`**, which creates a path object from arguments. This method is called whenever a derivative path is created, such as from **`pathlib.PurePath.parent`**. Subclasses may override this method to share information between path objects.
- gh-103220 [https://github.com/python/cpython/issues/103220]: Fix issue where **`os.path.join()`** added a slash when joining onto an incomplete UNC drive with a trailing slash on Windows.
- gh-103204 [https://github.com/python/cpython/issues/103204]: Fixes **`http.server`** accepting HTTP requests with HTTP version numbers preceded by "+", or "-", or with digit-separating "_" characters. The length of the version numbers is also constrained.
- gh-75586 [https://github.com/python/cpython/issues/75586]: Fix various Windows-specific issues with `shutil.which`.
- gh-103193 [https://github.com/python/cpython/issues/103193]: Improve performance of **`inspect.getattr_static()`**. Patch by Alex Waygood.
- gh-103176 [https://github.com/python/cpython/issues/103176]: **`sys._current_exceptions()`** now returns a mapping from thread-id to an exception instance, rather than to a `(typ, exc, tb)` tuple.
- gh-103143 [https://github.com/python/cpython/issues/103143]: Polish the help messages and docstrings of **`pdb`**.
- gh-103015 [https://github.com/python/cpython/issues/103015]: Add *entrypoint* keyword-only parameter to **`sqlite3.Connection.load_extension()`**, for overriding the SQLite extension entry point. Patch by Erlend E. Aasland.
- gh-103000 [https://github.com/python/cpython/issues/103000]: Improve performance of **`dataclasses.astuple()`** and **`dataclasses.asdict()`** in cases where the contents are

common Python types.

- gh-102953 [https://github.com/python/cpython/issues/102953]: The extraction methods in `tarfile`, and `shutil.unpack_archive()`, have a new a *filter* argument that allows limiting tar features than may be surprising or dangerous, such as creating files outside the destination directory. See Extraction filters for details.
- gh-97696 [https://github.com/python/cpython/issues/97696]: Implemented an eager task factory in asyncio. When used as a task factory on an event loop, it performs eager execution of coroutines. Coroutines that are able to complete synchronously (e.g. return or raise without blocking) are returned immediately as a finished task, and the task is never scheduled to the event loop. If the coroutine blocks, the (pending) task is scheduled and returned.
- gh-81079 [https://github.com/python/cpython/issues/81079]: Add *case_sensitive* keyword-only argument to `pathlib.Path.glob()` and `rglob()`.
- gh-101819 [https://github.com/python/cpython/issues/101819]: Isolate the `io` extension module by applying **PEP 687** [https://peps.python.org/pep-0687/]. Patch by Kumar Aditya, Victor Stinner, and Erlend E. Aasland.
- gh-91896 [https://github.com/python/cpython/issues/91896]: Deprecate `collections.abc.ByteString`
- gh-101362 [https://github.com/python/cpython/issues/101362]: Speed up `pathlib.Path` construction by omitting the path anchor from the internal list of path parts.
- gh-102114 [https://github.com/python/cpython/issues/102114]: Functions in the `dis` module that accept a source code string as argument now print a more concise traceback when the string contains a syntax or indentation error.
- gh-62432 [https://github.com/python/cpython/issues/62432]: The `unittest` runner will now exit with status code 5 if no tests were run. It is common for test runner misconfiguration to fail to find any tests, this should be an error.
- gh-78079 [https://github.com/python/cpython/issues/78079]: Fix incorrect normalization of UNC device path roots, and partial UNC share path roots, in `pathlib.PurePath`. Pathlib no longer appends a trailing slash to such paths.
- gh-85984 [https://github.com/python/cpython/issues/85984]: Add

`tty.cfmakeraw()` and `tty.cfmakecbreak()` to `tty` and modernize, the behavior of `tty.setraw()` and `tty.setcbreak()` to use POSIX.1-2017 Chapter 11 «General Terminal Interface» flag masks by default.

- gh-101688 [https://github.com/python/cpython/issues/101688]: Implement `types.get_original_bases()` to provide further introspection for types.
- gh-101640 [https://github.com/python/cpython/issues/101640]: `argparse.ArgumentParser` now catches errors when writing messages, such as when `sys.stderr` is `None`. Patch by Oleg Iarygin.
- gh-83861 [https://github.com/python/cpython/issues/83861]: Fix datetime.astimezone method return value when invoked on a naive datetime instance that represents local time falling in a timezone transition gap. PEP 495 requires that instances with fold=1 produce earlier times than those with fold=0 in this case.
- gh-89550 [https://github.com/python/cpython/issues/89550]: Decrease execution time of some `gzip` file writes by 15% by adding more appropriate buffering.
- gh-95299 [https://github.com/python/cpython/issues/95299]: Remove the bundled setuptools wheel from `ensurepip`, and stop installing setuptools in environments created by `venv`.
- gh-99353 [https://github.com/python/cpython/issues/99353]: Respect the `http.client.HTTPConnection` `.debuglevel` flag in **`urllib.request.AbstractHTTPHandler`** when its constructor parameter `debuglevel` is not set. And do the same for `*HTTPS*`.
- gh-98040 [https://github.com/python/cpython/issues/98040]: Remove the long-deprecated `imp` module.
- gh-97850 [https://github.com/python/cpython/issues/97850]: Deprecate **`pkgutil.find_loader()`** and **`pkgutil.get_loader()`** in favor of `importlib.util.find_spec()`.
- gh-94473 [https://github.com/python/cpython/issues/94473]: Flatten arguments in **`tkinter.Canvas.coords()`**. It now accepts not only `x1, y1, x2, y2, ...` and `[x1, y1, x2, y2, ...]`, but also `(x1, y1), (x2, y2), ...` and `[(x1, y1), (x2, y2), ...]`.
- gh-98040 [https://github.com/python/cpython/issues/98040]: Remove

more deprecated importlib APIs: `find_loader()`, `find_module()`, `importlib.abc.Finder`, `pkgutil.ImpImporter`, `pkgutil.ImpLoader`.

- gh-96522 [https://github.com/python/cpython/issues/96522]: Fix potential deadlock in pty.spawn()
- gh-96534 [https://github.com/python/cpython/issues/96534]: Support divert(4) added in FreeBSD 14.
- gh-87474 [https://github.com/python/cpython/issues/87474]: Fix potential file descriptor leaks in **`subprocess.Popen`**.
- gh-94906 [https://github.com/python/cpython/issues/94906]: Support multiple steps in **`math.nextafter()`**. Patch by Shantanu Jain and Matthias Gorgens.
- gh-51574 [https://github.com/python/cpython/issues/51574]: Make **`tempfile.mkdtemp()`** return absolute paths when its *dir* parameter is relative.
- gh-94518 [https://github.com/python/cpython/issues/94518]: Convert private **`_posixsubprocess.fork_exec()`** to use Argument Clinic.
- gh-92184 [https://github.com/python/cpython/issues/92184]: When creating zip files using **`zipfile`**, `os.altsep`, if not `None`, will always be treated as a path separator even when it is not `/`. Patch by Carey Metcalfe.
- bpo-46797 [https://bugs.python.org/issue?@action=redirect&bpo=46797]: Deprecation warnings are now emitted for **`ast.Num`**, **`ast.Bytes`**, **`ast.Str`**, **`ast.NameConstant`** and **`ast.Ellipsis`**. These have been documented as deprecated since Python 3.8, and will be removed in Python 3.14.
- bpo-44844 [https://bugs.python.org/issue?@action=redirect&bpo=44844]: Enables **`webbrowser`** to detect and launch Microsoft Edge browser.
- bpo-45606 [https://bugs.python.org/issue?@action=redirect&bpo=45606]: Fixed the bug in **`pathlib.Path.glob()`** – previously a dangling symlink would not be found by this method when the pattern is an exact match, but would be found when the pattern contains a wildcard or the recursive wildcard (`**`). With this change, a dangling symlink will be found in both cases.
- bpo-23041 [https://bugs.python.org/issue?@action=redirect&bpo=23041]: Add **`QUOTE_STRINGS`** and

`QUOTE_NOTNULL` to the suite of **`csv`** module quoting styles.

- bpo-24964 [https://bugs.python.org/issue?@action=redirect&bpo=24964]: Added **`http.client.HTTPConnection.get_proxy_response_heade`** that provides access to the HTTP headers on a proxy server response to the `CONNECT` request.
- bpo-17258 [https://bugs.python.org/issue?@action=redirect&bpo=17258]: **`multiprocessing`** now supports stronger HMAC algorithms for inter-process connection authentication rather than only HMAC-MD5.
- bpo-39744 [https://bugs.python.org/issue?@action=redirect&bpo=39744]: Make **`asyncio.subprocess.Process.communicate()`** close the subprocess's stdin even when called with `input=None`.
- bpo-22708 [https://bugs.python.org/issue?@action=redirect&bpo=22708]: http.client CONNECT method tunnel improvements: Use HTTP 1.1 protocol; send a matching Host: header with CONNECT, if one is not provided; convert IDN domain names to Punycode. Patch by Michael Handler.

Documentation

- gh-67056 [https://github.com/python/cpython/issues/67056]: Document that the effect of registering or unregistering an **`atexit`** cleanup function from within a registered cleanup function is undefined.
- gh-103629 [https://github.com/python/cpython/issues/103629]: Mention the new way of typing `**kwargs` with `Unpack` and `TypedDict` introduced in **PEP 692** [https://peps.python.org/pep-0692/].
- gh-48241 [https://github.com/python/cpython/issues/48241]: Clarifying documentation about the url parameter to urllib.request.urlopen and urllib.request.Request needing to be encoded properly.
- gh-86094 [https://github.com/python/cpython/issues/86094]: Add support for Unicode Path Extra Field in ZipFile. Patch by Yeojin Kim and Andrea Giudiceandrea
- gh-99202 [https://github.com/python/cpython/issues/99202]: Fix extension type from documentation for compiling in C++20

mode

Tests

- gh-104494 [https://github.com/python/cpython/issues/104494]: Update `test_pack_configure_in` and `test_place_configure_in` for changes to error message formatting in Tk 8.7.
- gh-104461 [https://github.com/python/cpython/issues/104461]: Run test_configure_screen on X11 only, since the `DISPLAY` environment variable and `-screen` option for toplevels are not useful on Tk for Win32 or Aqua.
- gh-86275 [https://github.com/python/cpython/issues/86275]: Added property-based tests to the **`zoneinfo`** tests, along with stubs for the `hypothesis` interface. (Patch by Paul Ganssle)
- gh-103329 [https://github.com/python/cpython/issues/103329]: Regression tests for the behaviour of `unittest.mock.PropertyMock` were added.
- gh-102795 [https://github.com/python/cpython/issues/102795]: fix use of poll in test_epoll's test_control_and_wait
- gh-75729 [https://github.com/python/cpython/issues/75729]: Fix the **`os.spawn*`** tests failing on Windows when the working directory or interpreter path contains spaces.

Build

- gh-101282 [https://github.com/python/cpython/issues/101282]: BOLT optimization is now applied to the libpython shared library if building a shared library. BOLT instrumentation and application settings can now be influenced via the `BOLT_INSTRUMENT_FLAGS` and `BOLT_APPLY_FLAGS` configure variables.

- gh-99017 [https://github.com/python/cpython/issues/99017]: `PYTHON_FOR_REGEN` now require Python 3.10 or newer.

- gh-104490 [https://github.com/python/cpython/issues/104490]: Define `.PHONY` / virtual make targets consistently and properly.

- gh-104106 [https://github.com/python/cpython/issues/104106]: Add

gcc fallback of mkfifoat/mknodat for macOS. Patch by Donghee Na.

- gh-103532 [https://github.com/python/cpython/issues/103532]: The `TKINTER_PROTECT_LOADTK` macro is no longer defined or used in the `_tkinter` module. It was previously only defined when building against Tk 8.4.13 and older, but Tk older than 8.5.12 has been unsupported since gh-91152 [https://github.com/python/cpython/issues/91152].

- gh-99069 [https://github.com/python/cpython/issues/99069]: Extended workaround defining `static_assert` when missing from the libc headers to all clang and gcc builds. In particular, this fixes building on macOS <= 10.10.

- gh-100220 [https://github.com/python/cpython/issues/100220]: Changed the default value of the `SHELL` Makefile variable from `/bin/sh` to `/bin/sh -e` to ensure that complex recipes correctly fail after an error. Previously, `make install` could fail to install some files and yet return a successful result.

- gh-90656 [https://github.com/python/cpython/issues/90656]: Add platform triplets for 64-bit LoongArch:

  ○ loongarch64-linux-gnusf
  ○ loongarch64-linux-gnuf32
  ○ loongarch64-linux-gnu

  Patch by Zhang Na.

Windows

- gh-104623 [https://github.com/python/cpython/issues/104623]: Update Windows installer to use SQLite 3.42.0.
- gh-82814 [https://github.com/python/cpython/issues/82814]: Fix a potential `[Errno 13] Permission denied` when using **`shutil.copystat()`** within Windows Subsystem for Linux (WSL) on a mounted filesystem by adding `errno.EACCES` to the list of ignored errors within the internal implementation.
- gh-103088 [https://github.com/python/cpython/issues/103088]: Fix

virtual environment `activate` script having incorrect line endings for Cygwin.
- gh-103088 [https://github.com/python/cpython/issues/103088]: Fixes venvs not working in bash on Windows across different disks
- gh-102997 [https://github.com/python/cpython/issues/102997]: Update Windows installer to use SQLite 3.41.2.
- gh-88013 [https://github.com/python/cpython/issues/88013]: Fixed a bug where **`TypeError`** was raised when calling **`ntpath.realpath()`** with a bytes parameter in some cases.

macOS

- gh-99834 [https://github.com/python/cpython/issues/99834]: Update macOS installer to Tcl/Tk 8.6.13.
- gh-104623 [https://github.com/python/cpython/issues/104623]: Update macOS installer to SQLite 3.42.0.
- gh-103545 [https://github.com/python/cpython/issues/103545]: Add `os.PRIO_DARWIN_THREAD`, `os.PRIO_DARWIN_PROCESS`, `os.PRIO_DARWIN_BG` and `os.PRIO_DARWIN_NONUI`. These can be used with `os.setpriority` to run the process at a lower priority and make use of the efficiency cores on Apple Silicon systems.
- gh-104180 [https://github.com/python/cpython/issues/104180]: Support reading SOCKS proxy configuration from macOS System Configuration. Patch by Sam Schott.
- gh-60436 [https://github.com/python/cpython/issues/60436]: update curses textbox to additionally handle backspace using the `curses.ascii.DEL` key press.
- gh-102997 [https://github.com/python/cpython/issues/102997]: Update macOS installer to SQLite 3.41.2.

IDLE

- gh-104499 [https://github.com/python/cpython/issues/104499]: Fix completions for Tk Aqua 8.7 (currently blank).
- gh-104496 [https://github.com/python/cpython/issues/104496]: About prints both tcl and tk versions if different (expected someday).
- gh-88496 [https://github.com/python/cpython/issues/88496]: Fix IDLE test hang on macOS.

Tools/Demos

- gh-104389 [https://github.com/python/cpython/issues/104389]: Argument Clinic C converters now accept the `unused` keyword, for wrapping a parameter with **`Py_UNUSED`**. Patch by Erlend E. Aasland.

C API

- gh-101291 [https://github.com/python/cpython/issues/101291]: Added unstable C API for extracting the value of «compact» integers: **`PyUnstable_Long_IsCompact()`** and **`PyUnstable_Long_CompactValue()`**.
- gh-104109 [https://github.com/python/cpython/issues/104109]: We've added `Py_NewInterpreterFromConfig()` and `PyInterpreterConfig` to the public C-API (but not the stable ABI; not yet at least). The new function may be used to create a new interpreter with various features configured. The function was added to support PEP 684 (per-interpreter GIL).
- gh-103968 [https://github.com/python/cpython/issues/103968]: **`PyType_FromSpec()`** and its variants now allow creating classes whose metaclass overrides **`tp_new`**. The `tp_new` is ignored. This behavior is deprecated and will be disallowed in 3.14+. The new **`PyType_FromMetaclass()`** already disallows it.
- gh-103743 [https://github.com/python/cpython/issues/103743]: Add **`PyUnstable_Object_GC_NewWithExtraData()`** function that can be used to allocate additional memory after an object for data not managed by Python.
- gh-103295 [https://github.com/python/cpython/issues/103295]: Introduced **`PyUnstable_WritePerfMapEntry()`**, **`PyUnstable_PerfMapState_Init()`** and **`PyUnstable_PerfMapState_Fini()`**. These allow extension modules (JIT compilers in particular) to write to perf-map files in a thread safe manner. The Soporte de Python para el perfilador perf de Linux also uses these APIs to write entries in the perf-map file.
- gh-103509 [https://github.com/python/cpython/issues/103509]: Added C API for extending types whose instance memory layout is opaque: **`PyType_Spec.basicsize`** can now be zero or

negative, **`PyObject_GetTypeData()`** can be used to get subclass-specific data, and **`Py_TPFLAGS_ITEMS_AT_END`** can be used to safely extend variable-size objects. See **PEP 697** [https://peps.python.org/pep-0697/] for details.

- gh-103091 [https://github.com/python/cpython/issues/103091]: Add a new C-API function to eagerly assign a version tag to a PyTypeObject: `PyUnstable_Type_AssignVersionTag()`.
- gh-101408 [https://github.com/python/cpython/issues/101408]: **`PyObject_GC_Resize`** should calculate preheader size if needed. Patch by Donghee Na.
- gh-98836 [https://github.com/python/cpython/issues/98836]: Add support of more formatting options (left aligning, octals, uppercase hexadecimals, **`intmax_t`**, **`ptrdiff_t`**, **`wchar_t`** C strings, variable width and precision) in **`PyUnicode_FromFormat()`** and **`PyUnicode_FromFormatV()`**.
- gh-96803 [https://github.com/python/cpython/issues/96803]: Add unstable C-API functions to get the code object, lasti and line number from the internal `_PyInterpreterFrame` in the limited API. The functions are:
    - `PyCodeObject * PyUnstable_InterpreterFrame_GetCode(struct _PyInterpreterFrame *frame)`
    - `int PyUnstable_InterpreterFrame_GetLasti(struct _PyInterpreterFrame *frame)`
    - `int PyUnstable_InterpreterFrame_GetLine(struct _PyInterpreterFrame *frame)`

Python 3.12.0 alpha 7

Release date: 2023-04-04

Core and Builtins

- gh-102192 [https://github.com/python/cpython/issues/102192]: Deprecated `_PyErr_ChainExceptions` in favour of

`_PyErr_ChainExceptions1`.

- gh-89987 [https://github.com/python/cpython/issues/89987]: Reduce the number of inline **`CACHE`** entries for **`BINARY_SUBSCR`**.
- gh-102859 [https://github.com/python/cpython/issues/102859]: Removed **`JUMP_IF_FALSE_OR_POP`** and **`JUMP_IF_TRUE_OR_POP`** instructions.
- gh-101975 [https://github.com/python/cpython/issues/101975]: Fixed `stacktop` value on tracing entries to avoid corruption on garbage collection.
- gh-102778 [https://github.com/python/cpython/issues/102778]: Add **`sys.last_exc`** and deprecate **`sys.last_type`**, **`sys.last_value`** and **`sys.last_traceback`**, which hold the same information in its legacy form.
- gh-100982 [https://github.com/python/cpython/issues/100982]: Replace all occurrences of `COMPARE_AND_BRANCH` with **`COMPARE_OP`**.
- gh-102701 [https://github.com/python/cpython/issues/102701]: Fix overflow when creating very large dict.
- gh-102755 [https://github.com/python/cpython/issues/102755]: Add **`PyErr_DisplayException()`** which takes just an exception instance, to replace the legacy **`PyErr_Display()`** which takes the `(typ, exc, tb)` triplet.
- gh-102594 [https://github.com/python/cpython/issues/102594]: Add note to exception raised in `PyErr_SetObject` when normalization fails.
- gh-90997 [https://github.com/python/cpython/issues/90997]: Shrink the number of inline **`CACHE`** entries used by **`LOAD_GLOBAL`**.

Library

- gh-102491 [https://github.com/python/cpython/issues/102491]: Improve import time of `platform` by removing IronPython version parsing. The IronPython version parsing was not functional (see https://github.com/IronLanguages/ironpython3/issues/1667).

Core and Builtins

- gh-101291 [https://github.com/python/cpython/issues/101291]: Rearrage bits in first field (after header) of PyLongObject. *

Bits 0 and 1: 1 - sign. I.e. 0 for positive numbers, 1 for zero and 2 for negative numbers. * Bit 2 reserved (probably for the immortal bit) * Bits 3+ the unsigned size.

This makes a few operations slightly more efficient, and will enable a more compact and faster 2s-complement representation of most ints in future.

- gh-102397 [https://github.com/python/cpython/issues/102397]: Fix segfault from race condition in signal handling during garbage collection. Patch by Kumar Aditya.
- gh-102406 [https://github.com/python/cpython/issues/102406]: **`codecs`** encoding/decoding errors now get the context information (which operation and which codecs) attached as **PEP 678** [https://peps.python.org/pep-0678/] notes instead of through chaining a new instance of the exception.
- gh-102281 [https://github.com/python/cpython/issues/102281]: Fix potential nullptr dereference and use of uninitialized memory in fileutils. Patch by Max Bachmann.
- gh-102300 [https://github.com/python/cpython/issues/102300]: Reuse operands with refcount of 1 in float specializations of BINARY_OP.
- gh-102213 [https://github.com/python/cpython/issues/102213]: Fix performance loss when accessing an object's attributes with `__getattr__` defined.
- gh-102255 [https://github.com/python/cpython/issues/102255]: Improve build support for the Xbox. Patch by Max Bachmann.

Build

- gh-102027 [https://github.com/python/cpython/issues/102027]: Fix SSE2 and SSE3 detection in `_blake2` internal module. Patch by Max Bachmann.

Core and Builtins

- gh-101865 [https://github.com/python/cpython/issues/101865]: Deprecate `co_lnotab` in code objects, schedule it for removal in Python 3.14

Library

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Adapt **`_pickle`** to **PEP 687** [https://peps.python.org/pep-0687/]. Patch by Mohamed Koubaa and Erlend Aasland.

- gh-103085 [https://github.com/python/cpython/issues/103085]: Pure python **`locale.getencoding()`** will not warn deprecation.

- gh-103068 [https://github.com/python/cpython/issues/103068]: It's no longer possible to register conditional breakpoints in **`Pdb`** that raise **`SyntaxError`**. Patch by Tian Gao.

- gh-102549 [https://github.com/python/cpython/issues/102549]: Don't ignore exceptions in member type creation.

- gh-103056 [https://github.com/python/cpython/issues/103056]: Ensure final `_generate_next_value_` is a `staticmethod`.

- gh-103046 [https://github.com/python/cpython/issues/103046]: Display current line label correctly in **`dis`** when `show_caches` is False and `lasti` points to a CACHE entry.

- gh-102433 [https://github.com/python/cpython/issues/102433]: **`isinstance()`** checks against **`runtime-checkable protocols`** now use **`inspect.getattr_static()`** rather than **`hasattr()`** to lookup whether attributes exist. This means that descriptors and **`__getattr__()`** methods are no longer unexpectedly evaluated during `isinstance()` checks against runtime-checkable protocols. However, it may also mean that some objects which used to be considered instances of a runtime-checkable protocol may no longer be considered instances of that protocol on Python 3.12+, and vice versa. Most users are unlikely to be affected by this change. Patch by Alex Waygood.

- gh-103023 [https://github.com/python/cpython/issues/103023]: It’s no longer possible to register expressions to display in **`Pdb`** that raise **`SyntaxError`**. Patch by Tian Gao.

- gh-102947 [https://github.com/python/cpython/issues/102947]: Improve traceback when **`dataclasses.fields()`** is called on a non-dataclass. Patch by Alex Waygood

- gh-102780 [https://github.com/python/cpython/issues/102780]: The **`asyncio.Timeout`** context manager now works reliably even when performing cleanup due to task cancellation. Previously it could raise a **`CancelledError`** instead of an **`TimeoutError`** in such cases.

- gh-102871 [https://github.com/python/cpython/issues/102871]: Remove support for obsolete browsers from **`webbrowser`**. Removed browsers include Grail, Mosaic, Netscape, Galeon, Skipstone, Iceape, Firebird, and Firefox versions 35 and below.

- gh-102839 [https://github.com/python/cpython/issues/102839]: Improve performance of **`math.log()`** arguments handling by removing the argument clinic.

- gh-102828 [https://github.com/python/cpython/issues/102828]: Add the `onexc` arg to **`shutil.rmtree()`**, which is like `onerror` but expects an exception instance rather than an exc_info tuple. Deprecate `onerror`.

- gh-88965 [https://github.com/python/cpython/issues/88965]: typing: Fix a bug relating to substitution in custom classes generic over a **`ParamSpec`**. Previously, if the `ParamSpec` was substituted with a parameters list that itself contained a **`TypeVar`**, the `TypeVar` in the parameters list could not be subsequently substituted. This is now fixed.

  Patch by Nikita Sobolev.

- gh-76846 [https://github.com/python/cpython/issues/76846]: Fix issue where `__new__()` and `__init__()` methods of **`pathlib.PurePath`** and **`Path`** subclasses were not called in some circumstances.

- gh-78530 [https://github.com/python/cpython/issues/78530]: **`asyncio.wait()`** now accepts generators yielding tasks. Patch by Kumar Aditya.
- gh-102748 [https://github.com/python/cpython/issues/102748]: **`asyncio.iscoroutine()`** now returns `False` for generators as **`asyncio`** does not support legacy generator-based coroutines. Patch by Kumar Aditya.
- gh-102670 [https://github.com/python/cpython/issues/102670]: Optimized fmean(), correlation(), covariance(), and linear_regression() using the new math.sumprod() function.
- gh-102615 [https://github.com/python/cpython/issues/102615]: Typing: Improve the `repr` of generic aliases for classes generic over a **`ParamSpec`**. (Use square brackets to represent a parameter list.)
- gh-100112 [https://github.com/python/cpython/issues/100112]: **`asyncio.Task.get_coro()`** now always returns a coroutine when wrapping an awaitable object. Patch by Kumar Aditya.
- gh-102578 [https://github.com/python/cpython/issues/102578]: Speed up setting or deleting mutable attributes on non-dataclass subclasses of frozen dataclasses. Due to the implementation of `__setattr__` and `__delattr__` for frozen dataclasses, this previously had a time complexity of $O(n)$. It now has a time complexity of $O(1)$.
- gh-102519 [https://github.com/python/cpython/issues/102519]: Add **`os.listdrives()`**, **`os.listvolumes()`** and **`os.listmounts()`** functions on Windows for enumerating drives, volumes and mount points
- gh-74468 [https://github.com/python/cpython/issues/74468]: Attribute name of the extracted **`tarfile`** file object now

holds filename of itself rather than of the archive it is contained in. Patch by Oleg Iarygin.

- gh-102378 [https://github.com/python/cpython/issues/102378]: Private helper method `inspect._signature_strip_non_python_syntax` will no longer strip `/` from the input string.
- gh-79940 [https://github.com/python/cpython/issues/79940]: Add **`inspect.getasyncgenstate()`** and **`inspect.getasyncgenlocals()`**. Patch by Thomas Krennwallner.
- gh-102103 [https://github.com/python/cpython/issues/102103]: Add `module` argument to **`dataclasses.make_dataclass()`** and make classes produced by it pickleable.
- gh-102069 [https://github.com/python/cpython/issues/102069]: Fix `__weakref__` descriptor generation for custom dataclasses.
- gh-102038 [https://github.com/python/cpython/issues/102038]: Skip a `stat` in **`site`** if we have already found a `pyvenv.cfg`
- gh-98886 [https://github.com/python/cpython/issues/98886]: Fix issues when defining dataclasses that have fields with specific underscore names that aren't clearly reserved by **`dataclasses`**.
- gh-101673 [https://github.com/python/cpython/issues/101673]: Fix a **`pdb`** bug where `ll` clears the changes to local variables.
- gh-101313 [https://github.com/python/cpython/issues/101313]: Added -h and –help arguments to the webbrowser CLI
- gh-100372 [https://github.com/python/cpython/issues/100372]: **`ssl.SSLContext.load_verify_locations()`** no longer incorrectly accepts some cases of trailing data when parsing DER.
- gh-89727 [https://github.com/python/cpython/issues/89727]: Fix pathlib.Path.walk RecursionError on deep directory trees by rewriting it using iteration instead of recursion.

- gh-100131 [https://github.com/python/cpython/issues/100131]: Added an optional `delete` keyword argument to **`tempfile.TemporaryDirectory`**.

- gh-48330 [https://github.com/python/cpython/issues/48330]: Added `--durations` command line option, showing the N slowest test cases. **`unittest.TextTestRunner`** and **`unittest.TextTestResult`** constructors accept a new *durations* keyword argument. Subclasses should take this into account or accept `**kwargs`. Added **`unittest.TestResult.addDuration()`** method and **`unittest.TestResult.collectedDurations`** attribute.

  (Contributed by Giampaolo Rodola)

- gh-98169 [https://github.com/python/cpython/issues/98169]: Fix **`dataclasses.astuple()`** crash when **`collections.defaultdict`** is present in the attributes.

- gh-96931 [https://github.com/python/cpython/issues/96931]: Fix incorrect results from **`ssl.SSLSocket.shared_ciphers()`**

- gh-95495 [https://github.com/python/cpython/issues/95495]: When built against OpenSSL 3.0, the **`ssl`** module had a bug where it reported unauthenticated EOFs (i.e. without close_notify) as a clean TLS-level EOF. It now raises **`SSLEOFError`**, matching the behavior in previous versions of OpenSSL. The **`options`** attribute on **`SSLContext`** also no longer includes **`OP_IGNORE_UNEXPECTED_EOF`** by default. This option may be set to specify the previous OpenSSL 3.0 behavior.

- gh-94684 [https://github.com/python/cpython/issues/94684]: Now **`uuid.uuid3()`** and **`uuid.uuid5()`** functions support **`bytes`** objects as their *name* argument.

- gh-94440 [https://github.com/python/cpython/issues/94440]: Fix a **`concurrent.futures.process`** bug where `ProcessPoolExecutor` shutdown could hang after a future has been quickly submitted and canceled.

- gh-72346 [https://github.com/python/cpython/issues/72346]: Added deprecation warning to *isdst* parameter of **`email.utils.localtime()`**.

- bpo-36305 [https://bugs.python.org/issue?@action=redirect&bpo=36305]: Fix handling of Windows filenames that resemble drives, such as `./a:b`, in **`pathlib`**.

Documentation

- gh-103112 [https://github.com/python/cpython/issues/103112]: Add docstring to **`http.client.HTTPResponse.read()`** to fix `pydoc` output.

Tests

- gh-102980 [https://github.com/python/cpython/issues/102980]: Improve test coverage on **`pdb`**.
- gh-102537 [https://github.com/python/cpython/issues/102537]: Adjust the error handling strategy in `test_zoneinfo.TzPathTest.python_tzpath_context`. Patch by Paul Ganssle.
- gh-101377 [https://github.com/python/cpython/issues/101377]: Improved test_locale_calendar_formatweekday of calendar.

Build

- gh-102973 [https://github.com/python/cpython/issues/102973]: Add a dev container (along with accompanying Dockerfile) for development purposes.
- gh-102711 [https://github.com/python/cpython/issues/102711]: Fix `-Wstrict-prototypes` compiler warnings.

Windows

- gh-102690 [https://github.com/python/cpython/issues/102690]: Update **`webbrowser`** to fall back to Microsoft Edge instead of Internet Explorer.
- gh-99726 [https://github.com/python/cpython/issues/99726]: Improves correctness of stat results for Windows, and uses

faster API when available

Tools/Demos

- gh-102809 [https://github.com/python/cpython/issues/102809]: `Misc/gdbinit` was removed.

C API

- gh-102013 [https://github.com/python/cpython/issues/102013]: Add a new (unstable) C-API function for iterating over GC'able objects using a callback: `PyUnstable_VisitObjects`.

Python 3.12.0 alpha 6

Release date: 2023-03-07

Security

- gh-99108 [https://github.com/python/cpython/issues/99108]: Replace builtin hashlib implementations of MD5 and SHA1 with verified ones from the HACL* project.
- gh-101727 [https://github.com/python/cpython/issues/101727]: Updated the OpenSSL version used in Windows and macOS binary release builds to 1.1.1t to address **CVE 2023-0286** [https://www.cve.org/CVERecord?id=CVE-2023-0286], **CVE 2022-4303** [https://www.cve.org/CVERecord?id=CVE-2022-4303], and **CVE 2022-4303** [https://www.cve.org/CVERecord?id=CVE-2022-4303] per the OpenSSL 2023-02-07 security advisory [https://openssl-library.org/news/secadv/20230207.txt].
- gh-99108 [https://github.com/python/cpython/issues/99108]: Replace the builtin **`hashlib`** implementations of SHA2-384 and SHA2-512 originally from LibTomCrypt with formally verified, side-channel resistant code from the HACL* [https://github.com/hacl-star/hacl-star/] project. The builtins remain a fallback only used when OpenSSL does not provide them.
- gh-101283 [https://github.com/python/cpython/issues/101283]: **`subprocess.Popen`** now uses a safer approach to find `cmd.exe` when launching with `shell=True`. Patch by Eryk

Sun, based on a patch by Oleg Iarygin.

Core and Builtins

- gh-102493 [https://github.com/python/cpython/issues/102493]: Fix regression in semantics of normalisation in `PyErr_SetObject`.
- gh-102416 [https://github.com/python/cpython/issues/102416]: Do not memoize incorrectly automatically generated loop rules in the parser. Patch by Pablo Galindo.
- gh-102356 [https://github.com/python/cpython/issues/102356]: Fix a bug that caused a crash when deallocating deeply nested filter objects. Patch by Marta Gómez Macías.
- gh-102336 [https://github.com/python/cpython/issues/102336]: Cleanup Windows 7 specific special handling. Patch by Max Bachmann.
- gh-102250 [https://github.com/python/cpython/issues/102250]: Fixed a segfault occurring when the interpreter calls a `__bool__` method that raises.
- gh-102126 [https://github.com/python/cpython/issues/102126]: Fix deadlock at shutdown when clearing thread states if any finalizer tries to acquire the runtime head lock. Patch by Kumar Aditya.
- gh-102027 [https://github.com/python/cpython/issues/102027]: Use `GetCurrentProcessId` on Windows when `getpid` is unavailable. Patch by Max Bachmann.
- gh-102056 [https://github.com/python/cpython/issues/102056]: Fix error handling bugs in interpreter's exception printing code, which could cause a crash on infinite recursion.
- gh-100982 [https://github.com/python/cpython/issues/100982]: Restrict the scope of the **`FOR_ITER_RANGE`** instruction to the scope of the original **`FOR_ITER`** instruction, to allow instrumentation.
- gh-101967 [https://github.com/python/cpython/issues/101967]: Fix possible segfault in `positional_only_passed_as_keyword` function, when new list created.
- gh-101952 [https://github.com/python/cpython/issues/101952]: Fix possible segfault in `BUILD_SET` opcode, when new set created.

- gh-74895 [https://github.com/python/cpython/issues/74895]: `socket.getaddrinfo` no longer raises `OverflowError` for `int` **port** values outside of the C long range. Out of range values are left up to the underlying string based C library API to report. A `socket.gaierror` `SAI_SERVICE` may occur instead, or no error at all as not all platform C libraries generate an error.
- gh-101799 [https://github.com/python/cpython/issues/101799]: Add `CALL_INTRINSIC_2` and use it instead of `PREP_RERAISE_STAR`.
- gh-101857 [https://github.com/python/cpython/issues/101857]: Fix xattr support detection on Linux systems by widening the check to linux, not just glibc. This fixes support for musl.
- gh-84783 [https://github.com/python/cpython/issues/84783]: Make the slice object hashable. Patch by Will Bradshaw and Furkan Onder.
- gh-87849 [https://github.com/python/cpython/issues/87849]: Change the `SEND` instruction to leave the receiver on the stack. This allows the specialized form of `SEND` to skip the chain of C calls and jump directly to the `RESUME` in the generator or coroutine.
- gh-101765 [https://github.com/python/cpython/issues/101765]: Fix SystemError / segmentation fault in iter `__reduce__` when internal access of `builtins.__dict__` keys mutates the iter object.
- gh-101430 [https://github.com/python/cpython/issues/101430]: Update `tracemalloc` to handle presize of object properly. Patch by Donghee Na.
- gh-101696 [https://github.com/python/cpython/issues/101696]: Invalidate type version tag in `_PyStaticType_Dealloc` for static types, avoiding bug where a false cache hit could crash the interpreter. Patch by Kumar Aditya.
- gh-101632 [https://github.com/python/cpython/issues/101632]: Adds a new `RETURN_CONST` instruction.
- gh-100719 [https://github.com/python/cpython/issues/100719]: Remove gi_code field from generator (and coroutine and async generator) objects as it is redundant. The frame already includes a reference to the code object.
- gh-98627 [https://github.com/python/cpython/issues/98627]: When an interpreter is configured to check (and only then),

importing an extension module will now fail when the extension does not support multiple interpreters (i.e. doesn't implement PEP 489 multi-phase init). This does not apply to the main interpreter, nor to subinterpreters created with `Py_NewInterpreter()`.

Library

- gh-102302 [https://github.com/python/cpython/issues/102302]: Micro-optimise hashing of **`inspect.Parameter`**, reducing the time it takes to hash an instance by around 40%.

- gh-101979 [https://github.com/python/cpython/issues/101979]: Fix a bug where parentheses in the `metavar` argument to **`argparse.ArgumentParser.add_argument()`** were dropped. Patch by Yeojin Kim.

- gh-91038 [https://github.com/python/cpython/issues/91038]: **`platform.platform()`** now has boolean default arguments.

- gh-81652 [https://github.com/python/cpython/issues/81652]: Add **`mmap.MAP_ALIGNED_SUPER`** FreeBSD and **`mmap.MAP_CONCEAL`** OpenBSD constants to **`mmap`**. Patch by Yeojin Kim.

- gh-102179 [https://github.com/python/cpython/issues/102179]: Fix **`os.dup2()`** error message for negative fds.

- gh-101961 [https://github.com/python/cpython/issues/101961]: For the binary mode, **`fileinput.hookcompressed()`** doesn't set the `encoding` value even if the value is `None`. Patch by Gihwan Kim.

- gh-101936 [https://github.com/python/cpython/issues/101936]: The default value of `fp` becomes **`io.BytesIO`** if **`HTTPError`** is initialized without a designated `fp` parameter. Patch by Long Vo.

- gh-101566 [https://github.com/python/cpython/issues/101566]: In zipfile, sync Path with zipp 3.14 [https://zipp.readthedocs.io/en/

latest/history.html#v3-14-0], including fix for extractall on the underlying zipfile after being wrapped in `Path`.

- gh-97930 [https://github.com/python/cpython/issues/97930]: Apply changes from importlib_resources 5.12 [https://importlib-resources.readthedocs.io/en/latest/history.html#v5-12-0], including fix for `MultiplexedPath` to support directories in multiple namespaces (python/importlib_resources#265).

- gh-101997 [https://github.com/python/cpython/issues/101997]: Upgrade pip wheel bundled with ensurepip (pip 23.0.1)

- gh-99108 [https://github.com/python/cpython/issues/99108]: The built-in extension modules for **`hashlib`** SHA2 algorithms, used when OpenSSL does not provide them, now live in a single internal `_sha2` module instead of separate `_sha256` and `_sha512` modules.

- gh-101892 [https://github.com/python/cpython/issues/101892]: Callable iterators no longer raise **`SystemError`** when the callable object exhausts the iterator but forgets to either return a sentinel value or raise **`StopIteration`**.

- gh-87634 [https://github.com/python/cpython/issues/87634]: Remove locking behavior from **`functools.cached_property()`**.

- gh-97786 [https://github.com/python/cpython/issues/97786]: Fix potential undefined behaviour in corner cases of floating-point-to-time conversions.

- gh-101517 [https://github.com/python/cpython/issues/101517]: Fixed bug where **`bdb`** looks up the source line with **`linecache`** with a `lineno=None`, which causes it to fail with an unhandled exception.

- gh-101773 [https://github.com/python/cpython/issues/101773]: Optimize **`fractions.Fraction`** for small components. The private argument `_normalize` of the **`fractions.Fraction`** constructor has been removed.

- gh-101693 [https://github.com/python/cpython/issues/101693]: In `sqlite3.Cursor.execute()`, `DeprecationWarning` is now emitted when named placeholders are used together with parameters supplied as a sequence instead of as a `dict`. Starting from Python 3.14, using named placeholders with parameters supplied as a sequence will raise a `ProgrammingError`. Patch by Erlend E. Aasland.

- gh-101446 [https://github.com/python/cpython/issues/101446]: Change repr of `collections.OrderedDict` to use regular dictionary formatting instead of pairs of keys and values.

- gh-101362 [https://github.com/python/cpython/issues/101362]: Speed up `pathlib.PurePath` construction by handling arguments more uniformly. When a `pathlib.Path` argument is supplied, we use its string representation rather than joining its parts with `os.path.join()`.

- gh-101362 [https://github.com/python/cpython/issues/101362]: Speed up `pathlib.PurePath` construction by calling `os.path.join()` only when two or more arguments are given.

- gh-101362 [https://github.com/python/cpython/issues/101362]: Speed up `pathlib.Path` construction by running the path flavour compatibility check only when pathlib is imported.

- gh-85984 [https://github.com/python/cpython/issues/85984]: Refactored the implementation of `pty.fork()` to use `os.login_tty()`.

  A `DeprecationWarning` is now raised by `pty.master_open()` and `pty.slave_open()`. They were undocumented and deprecated long long ago in the docstring in favor of `pty.openpty()`.

- gh-101561 [https://github.com/python/cpython/issues/101561]: Add a new decorator `typing.override()`. See **PEP 698** [https://peps.python.org/pep-0698/] for details. Patch by Steven Troxler.

- gh-63301 [https://github.com/python/cpython/issues/63301]: Set exit code when `tabnanny` CLI exits on error.

- gh-101360 [https://github.com/python/cpython/issues/101360]: Fix anchor matching in `pathlib.PureWindowsPath.match()`. Path and pattern anchors are now matched with `fnmatch`, just like other path parts. This allows patterns such as `"*:/Users/*"` to be matched.

- gh-101277 [https://github.com/python/cpython/issues/101277]: Remove global state from `itertools` module (**PEP 687** [https://peps.python.org/pep-0687/]). Patches by Erlend E. Aasland.

- gh-100809 [https://github.com/python/cpython/issues/100809]: Fix handling of drive-relative paths (like “C:” and “C:foo”) in `pathlib.Path.absolute()`. This method now uses the OS API to retrieve the correct current working directory for the drive.

- gh-99138 [https://github.com/python/cpython/issues/99138]: Apply **PEP 687** [https://peps.python.org/pep-0687/] to `zoneinfo`. Patch by Erlend E. Aasland.

- gh-96764 [https://github.com/python/cpython/issues/96764]: `asyncio.wait_for()` now uses `asyncio.timeout()` as its underlying implementation. Patch by Kumar Aditya.

- gh-88233 [https://github.com/python/cpython/issues/88233]: Correctly preserve «extra» fields in `zipfile` regardless of their ordering relative to a zip64 «extra.»

- bpo-23224 [https://bugs.python.org/issue?@action=redirect&bpo=23224]: Fix segfaults when creating `lzma.LZMADecompressor` and `bz2.BZ2Decompressor` objects without calling `__init__()`, and fix leakage of locks and internal buffers when calling the `__init__()` methods of `lzma.LZMADecompressor`, `lzma.LZMACompressor`, `bz2.BZ2Compressor`, and `bz2.BZ2Decompressor` objects multiple times.

Documentation

- gh-85417 [https://github.com/python/cpython/issues/85417]: Update

`cmath` documentation to clarify behaviour on branch cuts.
- gh-97725 [https://github.com/python/cpython/issues/97725]: Fix `asyncio.Task.print_stack()` description for `file=None`. Patch by Oleg Iarygin.

Tests

- gh-102019 [https://github.com/python/cpython/issues/102019]: Fix deadlock on shutdown if `test_current_{exception,frames}` fails. Patch by Jacob Bower.
- gh-85984 [https://github.com/python/cpython/issues/85984]: Utilize new «winsize» functions from termios in pty tests.
- gh-89792 [https://github.com/python/cpython/issues/89792]: `test_tools` now copies up to 10x less source data to a temporary directory during the `freeze` test by ignoring git metadata and other artifacts. It also limits its python build parallelism based on os.cpu_count instead of hard coding it as 8 cores.

Build

- gh-99942 [https://github.com/python/cpython/issues/99942]: On Android, in a static build, python-config in embed mode no longer incorrectly reports a library to link to.
- gh-99942 [https://github.com/python/cpython/issues/99942]: On Android, python.pc now correctly reports the library to link to, the same as python-config.sh.
- gh-100221 [https://github.com/python/cpython/issues/100221]: Fix creating install directories in `make sharedinstall` if they exist outside `DESTDIR` already.
- gh-96821 [https://github.com/python/cpython/issues/96821]: Explicitly mark C extension modules that need defined signed integer overflow, and add a configure option `--with-strict-overflow`. Patch by Matthias Görgens and Shantanu Jain.

Windows

- gh-102344 [https://github.com/python/cpython/issues/102344]:

Implement `winreg.QueryValue` using `QueryValueEx` and `winreg.SetValue` using `SetValueEx`. Patch by Max Bachmann.

- gh-101881 [https://github.com/python/cpython/issues/101881]: Handle read and write operations on non-blocking pipes properly on Windows.
- gh-101881 [https://github.com/python/cpython/issues/101881]: Add support for the os.get_blocking() and os.set_blocking() functions on Windows.
- gh-101849 [https://github.com/python/cpython/issues/101849]: Ensures installer will correctly upgrade existing `py.exe` launcher installs.
- gh-101763 [https://github.com/python/cpython/issues/101763]: Updates copy of libffi bundled with Windows installs to 3.4.4.
- gh-101759 [https://github.com/python/cpython/issues/101759]: Update Windows installer to SQLite 3.40.1.
- gh-101614 [https://github.com/python/cpython/issues/101614]: Correctly handle extensions built against debug binaries that reference `python3_d.dll`.
- gh-101196 [https://github.com/python/cpython/issues/101196]: The functions `os.path.isdir`, `os.path.isfile`, `os.path.islink` and `os.path.exists` are now 13% to 28% faster on Windows, by making fewer Win32 API calls.

macOS

- gh-101759 [https://github.com/python/cpython/issues/101759]: Update macOS installer to SQLite 3.40.1.

C API

- gh-101907 [https://github.com/python/cpython/issues/101907]: Removes use of non-standard C++ extension in public header files.

- gh-99293 [https://github.com/python/cpython/issues/99293]: Document that the Py_TPFLAGS_VALID_VERSION_TAG is an internal feature, should not be used, and will be removed.

- gh-101578 [https://github.com/python/cpython/issues/101578]: Add

`PyErr_GetRaisedException()` and **`PyErr_SetRaisedException()`** for saving and restoring the current exception. These functions return and accept a single exception object, rather than the triple arguments of the now-deprecated **`PyErr_Fetch()`** and **`PyErr_Restore()`**. This is less error prone and a bit more efficient.

Add **`PyException_GetArgs()`** and **`PyException_SetArgs()`** as convenience functions for retrieving and modifying the **`args`** passed to the exception's constructor.

- gh-91744 [https://github.com/python/cpython/issues/91744]: Introduced the *Unstable C API tier*, marking APi that is allowed to change in minor releases without a deprecation period. See **PEP 689** [https://peps.python.org/pep-0689/] for details.

Python 3.12.0 alpha 5

Release date: 2023-02-07

Security

- gh-99108 [https://github.com/python/cpython/issues/99108]: Replace the builtin **`hashlib`** implementations of SHA2-224 and SHA2-256 originally from LibTomCrypt with formally verified, side-channel resistant code from the HACL* [https://github.com/hacl-star/hacl-star/] project. The builtins remain a fallback only used when OpenSSL does not provide them.

Core and Builtins

- gh-92173 [https://github.com/python/cpython/issues/92173]: Fix the `defs` and `kwdefs` arguments to **`PyEval_EvalCodeEx()`** and a reference leak in that function.
- gh-59956 [https://github.com/python/cpython/issues/59956]: The GILState API is now partially compatible with subinterpreters.

Previously, `PyThreadState_GET()` and `PyGILState_GetThisThreadState()` would get out of sync, causing inconsistent behavior and crashes.

- gh-101400 [https://github.com/python/cpython/issues/101400]: Fix wrong lineno in exception message on `continue` or `break` which are not in a loop. Patch by Donghee Na.
- gh-101372 [https://github.com/python/cpython/issues/101372]: Fix `is_normalized()` to properly handle the UCD 3.2.0 cases. Patch by Donghee Na.
- gh-101266 [https://github.com/python/cpython/issues/101266]: Fix `sys.getsizeof()` reporting for `int` subclasses.
- gh-101291 [https://github.com/python/cpython/issues/101291]: Refactor the `PyLongObject` struct into a normal Python object header and a `PyLongValue` struct.
- gh-101046 [https://github.com/python/cpython/issues/101046]: Fix a possible memory leak in the parser when raising `MemoryError`. Patch by Pablo Galindo
- gh-101037 [https://github.com/python/cpython/issues/101037]: Fix potential memory underallocation issue for instances of `int` subclasses with value zero.
- gh-100762 [https://github.com/python/cpython/issues/100762]: Record the (virtual) exception block depth in the oparg of `YIELD_VALUE`. Use this to avoid the expensive `throw()` when closing generators (and coroutines) that can be closed trivially.
- gh-100982 [https://github.com/python/cpython/issues/100982]: Adds a new **COMPARE_AND_BRANCH** instruction. This is a bit more efficient when performing a comparison immediately followed by a branch, and restores the design intent of PEP 659 that specializations are local to a single instruction.
- gh-100942 [https://github.com/python/cpython/issues/100942]: Fixed segfault in property.getter/setter/deleter that occurred when a property subclass overrode the `__new__` method to return a non-property instance.
- gh-100923 [https://github.com/python/cpython/issues/100923]: Remove the `mask` cache entry for the `COMPARE_OP` instruction and embed the mask into the oparg.
- gh-100892 [https://github.com/python/cpython/issues/100892]: Fix race while iterating over thread states in clearing `threading.local`. Patch by Kumar Aditya.

- gh-91351 [https://github.com/python/cpython/issues/91351]: Fix a case where re-entrant imports could corrupt the import deadlock detection code and cause a **`KeyError`** to be raised out of **`importlib/_bootstrap`**. In addition to the straightforward cases, this could also happen when garbage collection leads to a warning being emitted – as happens when it collects an open socket or file)
- gh-100726 [https://github.com/python/cpython/issues/100726]: Optimize construction of `range` object for medium size integers.
- gh-100712 [https://github.com/python/cpython/issues/100712]: Added option to build cpython with specialization disabled, by setting `ENABLE_SPECIALIZATION=False` in **`opcode`**, followed by `make regen-all`.

Library

- bpo-32780 [https://bugs.python.org/issue?@action=redirect&bpo=32780]: Inter-field padding is now inserted into the PEP3118 format strings obtained from **`ctypes.Structure`** objects, reflecting their true representation in memory.
- gh-101541 [https://github.com/python/cpython/issues/101541]: [Enum] - fix psuedo-flag creation
- gh-101570 [https://github.com/python/cpython/issues/101570]: Upgrade pip wheel bundled with ensurepip (pip 23.0)
- gh-101323 [https://github.com/python/cpython/issues/101323]: Fix a bug where errors where not thrown by zlib._ZlibDecompressor if encountered during decompressing.
- gh-101317 [https://github.com/python/cpython/issues/101317]: Add *ssl_shutdown_timeout* parameter for **`asyncio.StreamWriter.start_tls()`**.
- gh-101326 [https://github.com/python/cpython/issues/101326]: Fix regression when passing `None` as second or third argument to `FutureIter.throw`.
- gh-92123 [https://github.com/python/cpython/issues/92123]: Adapt the `_elementtree` extension module to multi-phase init (**PEP 489** [https://peps.python.org/pep-0489/]). Patches by Erlend E. Aasland.
- gh-100795 [https://github.com/python/cpython/issues/100795]: Avoid

potential unexpected `freeaddrinfo` call (double free) in **`socket`** when when a libc `getaddrinfo()` implementation leaves garbage in an output pointer when returning an error. Original patch by Sergey G. Brester.

- gh-101143 [https://github.com/python/cpython/issues/101143]: Remove unused references to **`TimerHandle`** in `asyncio.base_events.BaseEventLoop._add_callback`.
- gh-101144 [https://github.com/python/cpython/issues/101144]: Make **`zipfile.Path.open()`** and **`zipfile.Path.read_text()`** also accept `encoding` as a positional argument. This was the behavior in Python 3.9 and earlier. 3.10 introduced a regression where supplying it as a positional argument would lead to a **`TypeError`**.
- gh-94518 [https://github.com/python/cpython/issues/94518]: Group-related variables of `_posixsubprocess` module are renamed to stress that supplementary group affinity is added to a fork, not replace the inherited ones. Patch by Oleg Iarygin.
- gh-101015 [https://github.com/python/cpython/issues/101015]: Fix **`typing.get_type_hints()`** on `'*tuple[...]'` and `*tuple[...]`. It must not drop the `Unpack` part.
- gh-101000 [https://github.com/python/cpython/issues/101000]: Add **`os.path.splitroot()`**, which splits a path into a 3-item tuple `(drive, root, tail)`. This new function is used by **`pathlib`** to improve the performance of path construction by up to a third.
- gh-100573 [https://github.com/python/cpython/issues/100573]: Fix a Windows **`asyncio`** bug with named pipes where a client doing `os.stat()` on the pipe would cause an error in the server that disabled serving future requests.
- gh-39615 [https://github.com/python/cpython/issues/39615]: **`warnings.warn()`** now has the ability to skip stack frames based on code filename prefix rather than only a numeric `stacklevel` via the new `skip_file_prefixes` keyword argument.
- gh-100750 [https://github.com/python/cpython/issues/100750]: pass encoding kwarg to subprocess in platform
- gh-100160 [https://github.com/python/cpython/issues/100160]: Emit a deprecation warning in **`asyncio.DefaultEventLoopPolicy.get_event_loop()`**

if there is no current event loop set and it decides to create one.

- gh-96290 [https://github.com/python/cpython/issues/96290]: Fix handling of partial and invalid UNC drives in `ntpath.splitdrive()`, and in `ntpath.normpath()` on non-Windows systems. Paths such as “\server” and “\” are now considered by `splitdrive()` to contain only a drive, and consequently are not modified by `normpath()` on non-Windows systems. The behaviour of `normpath()` on Windows systems is unaffected, as native OS APIs are used. Patch by Eryk Sun, with contributions by Barney Gale.
- gh-99952 [https://github.com/python/cpython/issues/99952]: Fix a reference undercounting issue in **`ctypes.Structure`** with `from_param()` results larger than a C pointer.
- gh-67790 [https://github.com/python/cpython/issues/67790]: Add float-style formatting support for **`fractions.Fraction`** instances.
- gh-99266 [https://github.com/python/cpython/issues/99266]: Preserve more detailed error messages in **`ctypes`**.
- gh-86682 [https://github.com/python/cpython/issues/86682]: Ensure runtime-created collections have the correct module name using the newly added (internal) **`sys._getframemodulename()`**.
- gh-88597 [https://github.com/python/cpython/issues/88597]: **`uuid`** now has a command line interface. Try `python -m uuid -h`.
- gh-60580 [https://github.com/python/cpython/issues/60580]: **`ctypes.wintypes.BYTE`** definition changed from **`c_byte`** to **`c_ubyte`** to match Windows SDK. Patch by Anatoly Techtonik and Oleg Iarygin.
- gh-94518 [https://github.com/python/cpython/issues/94518]: `_posixsubprocess` now initializes all UID and GID variables using a reserved `-1` value instead of a separate flag. Patch by Oleg Iarygin.
- bpo-38941 [https://bugs.python.org/issue?@action=redirect&bpo=38941]: The **`xml.etree.ElementTree`** module now emits **`DeprecationWarning`** when testing the truth value of an **`xml.etree.ElementTree.Element`**. Before, the Python implementation emitted **`FutureWarning`**, and the C implementation emitted nothing.

- bpo-40077 [https://bugs.python.org/issue?@action=redirect&bpo=40077]: Convert **`elementtree`** types to heap types. Patch by Erlend E. Aasland.
- bpo-29847 [https://bugs.python.org/issue?@action=redirect&bpo=29847]: Fix a bug where **`pathlib.Path`** accepted and ignored keyword arguments. Patch provided by Yurii Karabas.
- gh-77772 [https://github.com/python/cpython/issues/77772]: **`ctypes.CDLL`**, **`ctypes.OleDLL`**, **`ctypes.WinDLL`**, and **`ctypes.PyDLL`** now accept path-like objects as their `name` argument. Patch by Robert Hoelzl.

Documentation

- gh-88324 [https://github.com/python/cpython/issues/88324]: Reword **`subprocess`** to emphasize default behavior of *stdin*, *stdout*, and *stderr* arguments. Remove inaccurate statement about child file handle inheritance.

Tests

- gh-101334 [https://github.com/python/cpython/issues/101334]: `test_tarfile` has been updated to pass when run as a high UID.

Build

- gh-101282 [https://github.com/python/cpython/issues/101282]: Update BOLT configuration not to use deprecated usage of `--split functions`. Patch by Donghee Na.
- gh-101522 [https://github.com/python/cpython/issues/101522]: Allow overriding Windows dependencies versions and paths using MSBuild properties.
- gh-77532 [https://github.com/python/cpython/issues/77532]: Minor fixes to allow building with `PlatformToolset=ClangCL` on Windows.
- gh-101152 [https://github.com/python/cpython/issues/101152]: In accordance with **PEP 699** [https://peps.python.org/pep-0699/], the `ma_version_tag` field in **`PyDictObject`** is deprecated for extension modules. Accessing this field will generate a

compiler warning at compile time. This field will be removed in Python 3.14.

- gh-100340 [https://github.com/python/cpython/issues/100340]: Allows -Wno-int-conversion for wasm-sdk 17 and onwards, thus enables building WASI builds once against the latest sdk.
- gh-101060 [https://github.com/python/cpython/issues/101060]: Conditionally add `-fno-reorder-blocks-and-partition` in configure. Effectively fixes `--enable-bolt` when using Clang, as this appears to be a GCC-only flag.
- gh-98705 [https://github.com/python/cpython/issues/98705]: `__bool__` is defined in AIX system header files which breaks the build in AIX, so undefine it.
- gh-98636 [https://github.com/python/cpython/issues/98636]: Fix a regression in detecting `gdbm_compat` library for the `_gdbm` module build.
- gh-96305 [https://github.com/python/cpython/issues/96305]: `_aix_support` now uses a simple code to get platform details rather than the now non-existent `_bootsubprocess` during bootstrap.

Windows

- gh-101543 [https://github.com/python/cpython/issues/101543]: Ensure the install path in the registry is only used when the standard library hasn't been located in any other way.
- gh-101467 [https://github.com/python/cpython/issues/101467]: The `py.exe` launcher now correctly filters when only a single runtime is installed. It also correctly handles prefix matches on tags so that `-3.1` does not match `3.11`, but would still match `3.1-32`.
- gh-99834 [https://github.com/python/cpython/issues/99834]: Updates bundled copy of Tcl/Tk to 8.6.13.0
- gh-101135 [https://github.com/python/cpython/issues/101135]: Restore ability to launch older 32-bit versions from the `py.exe` launcher when both 32-bit and 64-bit installs of the same version are available.
- gh-82052 [https://github.com/python/cpython/issues/82052]: Fixed an issue where writing more than 32K of Unicode output to the console screen in one go can result in mojibake.
- gh-100320 [https://github.com/python/cpython/issues/100320]:

Ensures the `PythonPath` registry key from an install is used when launching from a different copy of Python that relies on an existing install to provide a copy of its modules and standard library.

- gh-100247 [https://github.com/python/cpython/issues/100247]: Restores support for the `py.exe` launcher finding shebang commands in its configuration file using the full command name.

Python 3.12.0 alpha 4

Release date: 2023-01-10

Core and Builtins

- gh-100776 [https://github.com/python/cpython/issues/100776]: Fix misleading default value in **`input()`**"s `__text_signature__`.

- gh-99005 [https://github.com/python/cpython/issues/99005]: Remove **`UNARY_POSITIVE`**, **`ASYNC_GEN_WRAP`** and **`LIST_TO_TUPLE`**, replacing them with intrinsics.

- gh-99005 [https://github.com/python/cpython/issues/99005]: Add new **`CALL_INTRINSIC_1`** instruction. Remove **`IMPORT_STAR`**, **`PRINT_EXPR`** and **`STOPITERATION_ERROR`**, replacing them with the **`CALL_INTRINSIC_1`** instruction.

- gh-100288 [https://github.com/python/cpython/issues/100288]: Remove the LOAD_ATTR_METHOD_WITH_DICT specialized instruction. Stats show it is not useful.

- gh-100720 [https://github.com/python/cpython/issues/100720]: Added `_PyFrame_NumSlotsForCodeObject`, which returns the number of slots needed in a frame for a given code object.

- gh-100719 [https://github.com/python/cpython/issues/100719]: Removed the co_nplaincellvars field from the code object, as it is redundant.

- gh-100637 [https://github.com/python/cpython/issues/100637]: Fix **`int.__sizeof__()`** calculation to include the 1-element `ob_digit` array for `0` and `False`.

- gh-100649 [https://github.com/python/cpython/issues/100649]: Update the native_thread_id field of PyThreadState after fork.

- gh-100126 [https://github.com/python/cpython/issues/100126]: Fix an issue where «incomplete» frames could be briefly visible to C code while other frames are being torn down, possibly resulting in corruption or hard crashes of the interpreter while running finalizers.

- gh-87447 [https://github.com/python/cpython/issues/87447]: Fix **`SyntaxError`** on comprehension rebind checking with names that are not actually redefined.

  Now reassigning `b` in `[(b := 1) for a, b.prop in some_iter]` is allowed. Reassigning `a` is still disallowed as per **PEP 572** [https://peps.python.org/pep-0572/].

- gh-100268 [https://github.com/python/cpython/issues/100268]: Add **`int.is_integer()`** to improve duck type compatibility between **`int`** and **`float`**.

- gh-100425 [https://github.com/python/cpython/issues/100425]: Improve the accuracy of `sum()` with compensated summation.

Library

- gh-100374 [https://github.com/python/cpython/issues/100374]: Fix incorrect result and delay in **`socket.getfqdn()`**. Patch by Dominic Socular.

Core and Builtins

- gh-100357 [https://github.com/python/cpython/issues/100357]: Convert `vars`, `dir`, `next`, `getattr`, and `iter` to argument clinic.
- gh-100117 [https://github.com/python/cpython/issues/100117]:

Improve the output of `codeobject.co_lines()` by emitting only one entry for each line range.

- gh-90043 [https://github.com/python/cpython/issues/90043]: Handle NaNs when specializing `COMPARE_OP` for `float` values.
- gh-100222 [https://github.com/python/cpython/issues/100222]: Redefine the `_Py_CODEUNIT` typedef as a union to describe its layout to the C compiler, avoiding type punning and improving clarity.
- gh-99955 [https://github.com/python/cpython/issues/99955]: Internal compiler functions (in compile.c) now consistently return -1 on error and 0 on success.
- gh-100188 [https://github.com/python/cpython/issues/100188]: The `BINARY_SUBSCR_LIST_INT` and `BINARY_SUBSCR_TUPLE_INT` instructions are no longer used for negative integers because those instructions always miss when encountering negative integers.
- gh-99110 [https://github.com/python/cpython/issues/99110]: Initialize frame->previous in frameobject.c to fix a segmentation fault when accessing frames created by `PyFrame_New()`.
- gh-94155 [https://github.com/python/cpython/issues/94155]: Improved the hashing algorithm for code objects, mitigating some hash collisions.
- gh-99540 [https://github.com/python/cpython/issues/99540]: `None` now hashes to a constant value. This is not a requirements change.
- gh-100143 [https://github.com/python/cpython/issues/100143]: When built with `--enable-pystats`, stats collection is now off by default. To enable it early at startup, pass the `-Xpystats` flag. Stats are now always dumped, even if switched off.
- gh-100146 [https://github.com/python/cpython/issues/100146]: Improve `BUILD_LIST` opcode so that it works similarly to the `BUILD_TUPLE` opcode, by stealing references from the stack rather than repeatedly using stack operations to set list elements. Implementation details are in a new private API **`_PyList_FromArraySteal()`**.
- gh-100110 [https://github.com/python/cpython/issues/100110]: Specialize `FOR_ITER` for tuples.
- gh-100050 [https://github.com/python/cpython/issues/100050]: Honor existing errors obtained when searching for mismatching parentheses in the tokenizer. Patch by Pablo Galindo

- gh-92216 [https://github.com/python/cpython/issues/92216]: Improve the performance of **`hasattr()`** for type objects with a missing attribute.
- gh-99582 [https://github.com/python/cpython/issues/99582]: Freeze **`zipimport`** module into `_bootstrap_python`.
- gh-99554 [https://github.com/python/cpython/issues/99554]: Pack debugging location tables more efficiently during bytecode compilation.
- gh-98522 [https://github.com/python/cpython/issues/98522]: Add an internal version number to code objects, to give better versioning of inner functions and comprehensions, and thus better specialization of those functions. This change is invisible to both Python and C extensions.
- gh-94603 [https://github.com/python/cpython/issues/94603]: Improve performance of `list.pop` for small lists.

Library

- gh-89051 [https://github.com/python/cpython/issues/89051]: Add **`ssl.OP_LEGACY_SERVER_CONNECT`**

Core and Builtins

- bpo-32782 [https://bugs.python.org/issue?@action=redirect&bpo=32782]: `ctypes` arrays of length 0 now report a correct itemsize when a `memoryview` is constructed from them, rather than always giving a value of 0.

Library

- gh-100833 [https://github.com/python/cpython/issues/100833]: Speed up **`math.fsum()`** by removing defensive `volatile` qualifiers.
- gh-100805 [https://github.com/python/cpython/issues/100805]: Modify **`random.choice()`** implementation to once again work with NumPy arrays.
- gh-100813 [https://github.com/python/cpython/issues/100813]: Add **`socket.IP_PKTINFO`** constant.
- gh-100792 [https://github.com/python/cpython/issues/100792]: Make **`email.message.Message.__contains__()`** twice as fast.

- gh-91851 [https://github.com/python/cpython/issues/91851]: Microoptimizations for `fractions.Fraction.__round__()`, `fractions.Fraction.__ceil__()` and `fractions.Fraction.__floor__()`.
- gh-90104 [https://github.com/python/cpython/issues/90104]: Avoid RecursionError on `repr` if a dataclass field definition has a cyclic reference.
- gh-100689 [https://github.com/python/cpython/issues/100689]: Fix crash in **`pyexpat`** by statically allocating `PyExpat_CAPI` capsule.
- gh-100740 [https://github.com/python/cpython/issues/100740]: Fix `unittest.mock.Mock` not respecting the spec for attribute names prefixed with `assert`.
- gh-91219 [https://github.com/python/cpython/issues/91219]: Change `SimpleHTTPRequestHandler` to support subclassing to provide a different set of index file names instead of using `__init__` parameters.
- gh-100690 [https://github.com/python/cpython/issues/100690]: `Mock` objects which are not unsafe will now raise an `AttributeError` when accessing an attribute that matches the name of an assertion but without the prefix `assert_`, e.g. accessing `called_once` instead of `assert_called_once`. This is in addition to this already happening for accessing attributes with prefixes `assert`, `assret`, `asert`, `aseert`, and `assrt`.
- gh-89727 [https://github.com/python/cpython/issues/89727]: Simplify and optimize `os.walk()` by using `isinstance()` checks to check the top of the stack.
- gh-100485 [https://github.com/python/cpython/issues/100485]: Add math.sumprod() to compute the sum of products.
- gh-86508 [https://github.com/python/cpython/issues/86508]: Fix `asyncio.open_connection()` to skip binding to local addresses of different family. Patch by Kumar Aditya.
- gh-97930 [https://github.com/python/cpython/issues/97930]: `importlib.resources.files` now accepts a module as an anchor instead of only accepting packages. If a module is passed, resources are resolved adjacent to that module (in the same package or at the package root). The parameter was renamed from `package` to `anchor` with a compatibility

shim for those passing by keyword. Additionally, the new `anchor` parameter is now optional and will default to the caller's module.

- gh-100585 [https://github.com/python/cpython/issues/100585]: Fixed a bug where importlib.resources.as_file was leaving file pointers open
- gh-100562 [https://github.com/python/cpython/issues/100562]: Improve performance of **`pathlib.Path.absolute()`** by nearly 2x. This comes at the cost of a performance regression in **`pathlib.Path.cwd()`**, which is generally used less frequently in user code.
- gh-100519 [https://github.com/python/cpython/issues/100519]: Small simplification of **`http.cookiejar.eff_request_host()`** that improves readability and better matches the RFC wording.
- gh-100287 [https://github.com/python/cpython/issues/100287]: Fix the interaction of **`unittest.mock.seal()`** with **`unittest.mock.AsyncMock`**.
- gh-100488 [https://github.com/python/cpython/issues/100488]: Add **`Fraction.is_integer()`** to check whether a **`fractions.Fraction`** is an integer. This improves duck type compatibility with **`float`** and **`int`**.
- gh-100474 [https://github.com/python/cpython/issues/100474]: **`http.server`** now checks that an index page is actually a regular file before trying to serve it. This avoids issues with directories named `index.html`.
- gh-100363 [https://github.com/python/cpython/issues/100363]: Speed up **`asyncio.get_running_loop()`** by removing redundant `getpid` checks. Patch by Kumar Aditya.
- gh-78878 [https://github.com/python/cpython/issues/78878]: Fix crash when creating an instance of **`_ctypes.CField`**.
- gh-100348 [https://github.com/python/cpython/issues/100348]: Fix ref cycle in **`asyncio._SelectorSocketTransport`** by removing `_read_ready_cb` in `close`.
- gh-100344 [https://github.com/python/cpython/issues/100344]: Provide C implementation for **`asyncio.current_task()`** for a 4x-6x speedup.
- gh-100272 [https://github.com/python/cpython/issues/100272]: Fix JSON serialization of OrderedDict. It now preserves the order of keys.

- gh-83076 [https://github.com/python/cpython/issues/83076]: Instantiation of `Mock()` and `AsyncMock()` is now 3.8x faster.
- gh-100234 [https://github.com/python/cpython/issues/100234]: Set a default value of 1.0 for the `lambd` parameter in random.expovariate().
- gh-100228 [https://github.com/python/cpython/issues/100228]: A **`DeprecationWarning`** may be raised when **`os.fork()`** or **`os.forkpty()`** is called from multi-threaded processes. Forking with threads is unsafe and can cause deadlocks, crashes and subtle problems. Lack of a warning does not indicate that the fork call was actually safe, as Python may not be aware of all threads.
- gh-100039 [https://github.com/python/cpython/issues/100039]: Improve signatures for enums and flags.
- gh-100133 [https://github.com/python/cpython/issues/100133]: Fix regression in **`asyncio`** where a subprocess would sometimes lose data received from pipe.
- bpo-44592 [https://bugs.python.org/issue?@action=redirect&bpo=44592]: Fixes inconsistent handling of case sensitivity of *extrasaction* arg in **`csv.DictWriter`**.
- gh-100098 [https://github.com/python/cpython/issues/100098]: Fix `tuple` subclasses being cast to `tuple` when used as enum values.
- gh-85432 [https://github.com/python/cpython/issues/85432]: Rename the *fmt* parameter of the pure-Python implementation of **`datetime.time.strftime()`** to *format*. Rename the *t* parameter of **`datetime.datetime.fromtimestamp()`** to *timestamp*. These changes mean the parameter names in the pure-Python implementation now match the parameter names in the C implementation. Patch by Alex Waygood.
- gh-98778 [https://github.com/python/cpython/issues/98778]: Update **`HTTPError`** to be initialized properly, even if the `fp` is `None`. Patch by Donghee Na.
- gh-99925 [https://github.com/python/cpython/issues/99925]: Unify error messages in JSON serialization between `json.dumps(float('nan'), allow_nan=False)` and `json.dumps(float('nan'), allow_nan=False, indent=<SOMETHING>)`. Now both include the representation of the value that could not be serialized.

- gh-89727 [https://github.com/python/cpython/issues/89727]: Fix issue with `os.walk()` where a `RecursionError` would occur on deep directory structures by adjusting the implementation of `os.walk()` to be iterative instead of recursive.
- gh-94943 [https://github.com/python/cpython/issues/94943]: Add Soporte de Dataclass to the `Enum` `__repr__()`. When inheriting from a `dataclass`, only show the field names in the value section of the member `repr()`, and not the dataclass" class name.
- gh-83035 [https://github.com/python/cpython/issues/83035]: Fix `inspect.getsource()` handling of decorator calls with nested parentheses.
- gh-99576 [https://github.com/python/cpython/issues/99576]: Fix `.save()` method for `LWPCookieJar` and `MozillaCookieJar`: saved file was not truncated on repeated save.
- gh-94912 [https://github.com/python/cpython/issues/94912]: Add `inspect.markcoroutinefunction()` decorator which manually marks a function as a coroutine for the benefit of `iscoroutinefunction()`.
- gh-99509 [https://github.com/python/cpython/issues/99509]: Add **PEP 585** [https://peps.python.org/pep-0585/] support for `multiprocessing.queues.Queue`.
- gh-99482 [https://github.com/python/cpython/issues/99482]: Remove `Jython` partial compatibility code from several stdlib modules.
- gh-99433 [https://github.com/python/cpython/issues/99433]: Fix `doctest` failure on `types.MethodWrapperType` in modules.
- gh-85267 [https://github.com/python/cpython/issues/85267]: Several improvements to `inspect.signature()`"s handling of `__text_signature`. - Fixes a case where `inspect.signature()` dropped parameters - Fixes a case where `inspect.signature()` raised `tokenize.TokenError` - Allows `inspect.signature()` to understand defaults involving binary operations of constants - `inspect.signature()` is documented as only raising `TypeError` or `ValueError`, but sometimes raised `RuntimeError`. These cases now raise `ValueError` - Removed a dead code path

- gh-91166 [https://github.com/python/cpython/issues/91166]: **`asyncio`** is optimized to avoid excessive copying when writing to socket and use **`sendmsg()`** if the platform supports it. Patch by Kumar Aditya.
- gh-98030 [https://github.com/python/cpython/issues/98030]: Add missing TCP socket options from Linux: `TCP_MD5SIG`, `TCP_THIN_LINEAR_TIMEOUTS`, `TCP_THIN_DUPACK`, `TCP_REPAIR`, `TCP_REPAIR_QUEUE`, `TCP_QUEUE_SEQ`, `TCP_REPAIR_OPTIONS`, `TCP_TIMESTAMP`, `TCP_CC_INFO`, `TCP_SAVE_SYN`, `TCP_SAVED_SYN`, `TCP_REPAIR_WINDOW`, `TCP_FASTOPEN_CONNECT`, `TCP_ULP`, `TCP_MD5SIG_EXT`, `TCP_FASTOPEN_KEY`, `TCP_FASTOPEN_NO_COOKIE`, `TCP_ZEROCOPY_RECEIVE`, `TCP_INQ`, `TCP_TX_DELAY`.
- gh-88500 [https://github.com/python/cpython/issues/88500]: Reduced the memory usage of **`urllib.parse.unquote()`** and **`urllib.parse.unquote_to_bytes()`** on large values.
- gh-96127 [https://github.com/python/cpython/issues/96127]: `inspect.signature` was raising `TypeError` on call with mock objects. Now it correctly returns `(*args, **kwargs)` as inferred signature.
- gh-95882 [https://github.com/python/cpython/issues/95882]: Fix a 3.11 regression in **`asynccontextmanager()`**, which caused it to propagate exceptions with incorrect tracebacks and fix a 3.11 regression in **`contextmanager()`**, which caused it to propagate exceptions with incorrect tracebacks for **`StopIteration`**.
- gh-78707 [https://github.com/python/cpython/issues/78707]: Deprecate passing more than one positional argument to **`pathlib.PurePath.relative_to()`** and **`is_relative_to()`**.
- gh-92122 [https://github.com/python/cpython/issues/92122]: Fix reStructuredText syntax errors in docstrings in the **`enum`** module.
- gh-91851 [https://github.com/python/cpython/issues/91851]: Optimize the **`Fraction`** arithmetics for small components.
- bpo-24132 [https://bugs.python.org/issue?@action=redirect&bpo=24132]: Make **`pathlib.PurePath`** and **`Path`** subclassable (private to start). Previously, attempting to instantiate a subclass resulted in an **`AttributeError`** being raised. Patch by Barney Gale.

- bpo-40447 [https://bugs.python.org/issue?@action=redirect&bpo=40447]: Accept **`os.PathLike`** (such as **`pathlib.Path`**) in the `stripdir` arguments of **`compileall.compile_file()`** and **`compileall.compile_dir()`**.
- bpo-36880 [https://bugs.python.org/issue?@action=redirect&bpo=36880]: Fix a reference counting issue when a **`ctypes`** callback with return type **`py_object`** returns `None`, which could cause crashes.

Documentation

- gh-100616 [https://github.com/python/cpython/issues/100616]: Document existing `attr` parameter to **`curses.window.vline()`** function in **`curses`**.
- gh-100472 [https://github.com/python/cpython/issues/100472]: Remove claim in documentation that the `stripdir`, `prependdir` and `limit_sl_dest` parameters of **`compileall.compile_dir()`** and **`compileall.compile_file()`** could be **`bytes`**.
- bpo-25377 [https://bugs.python.org/issue?@action=redirect&bpo=25377]: Clarify use of octal format of mode argument in help(os.chmod) as well as help(os.fchmod)

Tests

- gh-100454 [https://github.com/python/cpython/issues/100454]: Start running SSL tests with OpenSSL 3.1.0-beta1.
- gh-100086 [https://github.com/python/cpython/issues/100086]: The Python test runner (libregrtest) now logs Python build information like «debug» vs «release» build, or LTO and PGO optimizations. Patch by Victor Stinner.
- gh-93018 [https://github.com/python/cpython/issues/93018]: Make two tests forgiving towards host system libexpat with backported security fixes applied.

Build

- gh-100540 [https://github.com/python/cpython/issues/100540]: Removed the `--with-system-ffi` `configure` option;

`libffi` must now always be supplied by the system on all non-Windows platforms. The option has had no effect on non-Darwin platforms for several releases, and in 3.11 only had the non-obvious effect of invoking `pkg-config` to find `libffi` and never setting `-DUSING_APPLE_OS_LIBFFI`. Now on Darwin platforms `configure` will first check for the OS `libffi` and then fall back to the same processing as other platforms if it is not found.

- gh-88267 [https://github.com/python/cpython/issues/88267]: Avoid exporting Python symbols in linked Windows applications when the core is built as static.
- bpo-41916 [https://bugs.python.org/issue?@action=redirect&bpo=41916]: Allow override of ac_cv_cxx_thread so that cross compiled python can set -pthread for CXX.

Windows

- gh-100180 [https://github.com/python/cpython/issues/100180]: Update Windows installer to OpenSSL 1.1.1s
- gh-99191 [https://github.com/python/cpython/issues/99191]: Use `_MSVC_LANG >= 202002L` instead of less-precise `_MSC_VER >=1929` to more accurately test for C++20 support in `PC/_wmimodule.cpp`.
- gh-79218 [https://github.com/python/cpython/issues/79218]: Define `MS_WIN64` for Mingw-w64 64bit, fix cython compilation failure.
- gh-99941 [https://github.com/python/cpython/issues/99941]: Ensure that **`asyncio.Protocol.data_received()`** receives an immutable **`bytes`** object (as documented), instead of **`bytearray`**.
- bpo-43984 [https://bugs.python.org/issue?@action=redirect&bpo=43984]: **`winreg.SetValueEx()`** now leaves the target value untouched in the case of conversion errors. Previously, `-1` would be written in case of such errors.
- bpo-34816 [https://bugs.python.org/issue?@action=redirect&bpo=34816]: `hasattr(ctypes.windll, 'nonexistant')` now returns `False` instead of raising **`OSError`**.

macOS

- gh-100180 [https://github.com/python/cpython/issues/100180]: Update macOS installer to OpenSSL 1.1.1s
- gh-100540 [https://github.com/python/cpython/issues/100540]: Removed obsolete `dlfcn.h` shim from the `_ctypes` extension module, which has not been necessary since Mac OS X 10.2.

Tools/Demos

- bpo-45256 [https://bugs.python.org/issue?@action=redirect&bpo=45256]: Fix a bug that caused an **`AttributeError`** to be raised in `python-gdb.py` when `py-locals` is used without a frame.
- gh-100342 [https://github.com/python/cpython/issues/100342]: Add missing `NULL` check for possible allocation failure in `*args` parsing in Argument Clinic.

C API

- gh-99947 [https://github.com/python/cpython/issues/99947]: Raising SystemError on import will now have its cause be set to the original unexpected exception.
- gh-99240 [https://github.com/python/cpython/issues/99240]: In argument parsing, after deallocating newly allocated memory, reset its pointer to NULL.
- gh-98724 [https://github.com/python/cpython/issues/98724]: The **`Py_CLEAR`**, **`Py_SETREF`** and **`Py_XSETREF`** macros now only evaluate their arguments once. If an argument has side effects, these side effects are no longer duplicated. Patch by Victor Stinner.

Python 3.12.0 alpha 3

Release date: 2022-12-06

Security

- gh-100001 [https://github.com/python/cpython/issues/100001]: `python -m http.server` no longer allows terminal control characters sent within a garbage request to be printed to the stderr server log.

  This is done by changing the **`http.server`** **`BaseHTTPRequestHandler`** `.log_message` method to replace control characters with a `\x`*`HH`* hex escape before printing.

- gh-87604 [https://github.com/python/cpython/issues/87604]: Avoid publishing list of active per-interpreter audit hooks via the **`gc`** module

Core and Builtins

- gh-99891 [https://github.com/python/cpython/issues/99891]: Fix a bug in the tokenizer that could cause infinite recursion when showing syntax warnings that happen in the first line of the source. Patch by Pablo Galindo
- gh-91054 [https://github.com/python/cpython/issues/91054]: Add **`PyCode_AddWatcher()`** and **`PyCode_ClearWatcher()`** APIs to register callbacks to receive notification on creation and destruction of code objects.
- gh-99729 [https://github.com/python/cpython/issues/99729]: Fix an issue that could cause frames to be visible to Python code as they are being torn down, possibly leading to memory corruption or hard crashes of the interpreter.
- gh-99708 [https://github.com/python/cpython/issues/99708]: Fix bug where compiler crashes on an if expression with an empty body block.
- gh-99578 [https://github.com/python/cpython/issues/99578]: Fix a reference bug in **`_imp.create_builtin()`** after the creation of the first sub-interpreter for modules `builtins` and `sys`. Patch by Victor Stinner.
- gh-99581 [https://github.com/python/cpython/issues/99581]: Fixed a bug that was causing a buffer overflow if the tokenizer copies a line missing the newline character from a file that is as long as the available tokenizer buffer. Patch by Pablo galindo
- gh-99553 [https://github.com/python/cpython/issues/99553]: Fix bug

where an **`ExceptionGroup`** subclass can wrap a **`BaseException`**.

Library

- gh-99547 [https://github.com/python/cpython/issues/99547]: Add a function to os.path to check if a path is a junction: isjunction. Add similar functionality to pathlib.Path as is_junction.
- gh-99370 [https://github.com/python/cpython/issues/99370]: Fix zip path for venv created from a non-installed python on POSIX platforms.

Core and Builtins

- gh-99377 [https://github.com/python/cpython/issues/99377]: Add audit events for thread creation and clear operations.
- gh-98686 [https://github.com/python/cpython/issues/98686]: Remove the `BINARY_OP_GENERIC` and `COMPARE_OP_GENERIC` «specializations».
- gh-99298 [https://github.com/python/cpython/issues/99298]: Remove the remaining error paths for attribute specializations, and refuse to specialize attribute accesses on types that haven't had **`PyType_Ready()`** called on them yet.
- gh-99127 [https://github.com/python/cpython/issues/99127]: Allow some features of **`syslog`** to the main interpreter only. Patch by Donghee Na.
- gh-91053 [https://github.com/python/cpython/issues/91053]: Optimizing interpreters and JIT compilers may need to invalidate internal metadata when functions are modified. This change adds the ability to provide a callback that will be invoked each time a function is created, modified, or destroyed.
- gh-90994 [https://github.com/python/cpython/issues/90994]: Improve error messages when there's a syntax error with call arguments. The following three cases are covered: - No value is assigned to a named argument, eg `foo(a=)`. - A value is assigned to a star argument, eg `foo(*args=[0])`. - A value is assigned to a double-star keyword argument, eg `foo(**kwarg={'a': 0})`.
- bpo-45026 [https://bugs.python.org/issue?

@action=redirect&bpo=45026]: Optimize the **`range`** object iterator. It is now smaller, faster iteration of ranges containing large numbers. Smaller pickles, faster unpickling.

- bpo-31718 [https://bugs.python.org/issue?@action=redirect&bpo=31718]: Raise **`ValueError`** instead of **`SystemError`** when methods of uninitialized **`io.IncrementalNewlineDecoder`** objects are called. Patch by Oren Milman.
- bpo-38031 [https://bugs.python.org/issue?@action=redirect&bpo=38031]: Fix a possible assertion failure in **`io.FileIO`** when the opener returns an invalid file descriptor.

Library

- gh-100001 [https://github.com/python/cpython/issues/100001]: Also escape s in the http.server BaseHTTPRequestHandler.log_message so that it is technically possible to parse the line and reconstruct what the original data was. Without this a xHH is ambiguous as to if it is a hex replacement we put in or the characters r»x» came through in the original request line.

- gh-99957 [https://github.com/python/cpython/issues/99957]: Add `frozen_default` parameter to **`typing.dataclass_transform()`**.

- gh-79033 [https://github.com/python/cpython/issues/79033]: Fix **`asyncio.Server.wait_closed()`** to actually do what the docs promise – wait for all existing connections to complete, after closing the server.

- gh-51524 [https://github.com/python/cpython/issues/51524]: Fix bug when calling trace.CoverageResults with valid infile.

- gh-99645 [https://github.com/python/cpython/issues/99645]: Fix a bug in handling class cleanups in **`unittest.TestCase`**. Now `addClassCleanup()` uses separate lists for different `TestCase` subclasses, and `doClassCleanups()` only cleans up the particular class.

- gh-99508 [https://github.com/python/cpython/issues/99508]: Fix `TypeError` in `Lib/importlib/_bootstrap_external.py` while calling `_imp.source_hash()`.
- gh-66285 [https://github.com/python/cpython/issues/66285]: Fix **`asyncio`** to not share event loop and signal wakeupfd in forked processes. Patch by Kumar Aditya.
- gh-97001 [https://github.com/python/cpython/issues/97001]: Release the GIL when calling termios APIs to avoid blocking threads.
- gh-92647 [https://github.com/python/cpython/issues/92647]: Use final status of an enum to determine lookup or creation branch of functional API.
- gh-99388 [https://github.com/python/cpython/issues/99388]: Add *loop_factory* parameter to **`asyncio.run()`** to allow specifying a custom event loop factory. Patch by Kumar Aditya.
- gh-99341 [https://github.com/python/cpython/issues/99341]: Fix **`ast.increment_lineno()`** to also cover **`ast.TypeIgnore`** when changing line numbers.
- gh-99382 [https://github.com/python/cpython/issues/99382]: Check the number of arguments in substitution in user generics containing a **`TypeVarTuple`** and one or more **`TypeVar`**.
- gh-99379 [https://github.com/python/cpython/issues/99379]: Fix substitution of **`ParamSpec`** followed by **`TypeVarTuple`** in generic aliases.
- gh-99344 [https://github.com/python/cpython/issues/99344]: Fix substitution of **`TypeVarTuple`** and **`ParamSpec`** together in user generics.
- gh-99284 [https://github.com/python/cpython/issues/99284]: Remove `_use_broken_old_ctypes_structure_semantics_` old untested and undocumented hack from **`ctypes`**.
- gh-99201 [https://github.com/python/cpython/issues/99201]: Fix

`IndexError` when initializing the config variables on Windows if `HAVE_DYNAMIC_LOADING` is not set.

- gh-99240 [https://github.com/python/cpython/issues/99240]: Fix double-free bug in Argument Clinic `str_converter` by extracting memory clean up to a new `post_parsing` section.

- gh-64490 [https://github.com/python/cpython/issues/64490]: Fix refcount error when arguments are packed to tuple in Argument Clinic.

- gh-99029 [https://github.com/python/cpython/issues/99029]: **`pathlib.PurePath.relative_to()`** now treats naked Windows drive paths as relative. This brings its behaviour in line with other parts of pathlib.

- gh-98253 [https://github.com/python/cpython/issues/98253]: The implementation of the typing module is now more resilient to reference leaks in binary extension modules.

  Previously, a reference leak in a typed C API-based extension module could leak internals of the typing module, which could in turn introduce leaks in essentially any other package with typed function signatures. Although the typing package is not the original source of the problem, such non-local dependences exacerbate debugging of large-scale projects, and the implementation was therefore changed to reduce harm by providing better isolation.

- gh-98458 [https://github.com/python/cpython/issues/98458]: Fix infinite loop in unittest when a self-referencing chained exception is raised

- gh-93453 [https://github.com/python/cpython/issues/93453]: **`asyncio.get_event_loop()`** and many other **`asyncio`** functions like **`asyncio.ensure_future()`**, **`asyncio.shield()`** or **`asyncio.gather()`**, and also the **`get_event_loop()`** method of **`asyncio.BaseDefaultEventLoopPolicy`** now raise a **`RuntimeError`** if called when there is no running event loop and the current event loop was not set. Previously they implicitly created and set a new current event loop. `DeprecationWarning` is no longer emitted if there is no running event loop but the current event loop was set.

- gh-97966 [https://github.com/python/cpython/issues/97966]: On `uname_result`, restored expectation that `_fields` and `_asdict` would include all six properties including `processor`.

- gh-98248 [https://github.com/python/cpython/issues/98248]: Provide informative error messages in `struct.pack()` when its integral arguments are not in range.

- gh-98108 [https://github.com/python/cpython/issues/98108]: `zipfile.Path` is now pickleable if its initialization parameters were pickleable (e.g. for file system paths).

- gh-98098 [https://github.com/python/cpython/issues/98098]: Created packages from zipfile and test_zipfile modules, separating `zipfile.Path` functionality.

- gh-82836 [https://github.com/python/cpython/issues/82836]: Fix `is_private` properties in the `ipaddress` module. Previously non-private networks (0.0.0.0/0) would return `True` from this method; now they correctly return `False`.

- gh-96828 [https://github.com/python/cpython/issues/96828]: Add an `OP_ENABLE_KTLS` option for enabling the use of the kernel TLS (kTLS). Patch by Illia Volochii.

- gh-88863 [https://github.com/python/cpython/issues/88863]: To avoid apparent memory leaks when `asyncio.open_connection()` raises, break reference cycles generated by local exception and future instances (which has exception instance as its member var). Patch by Dong Uk, Kang.

- gh-91078 [https://github.com/python/cpython/issues/91078]: `TarFile.next()` now returns `None` when called on an empty tarfile.

- bpo-47220 [https://bugs.python.org/issue?@action=redirect&bpo=47220]: Document the optional *callback* parameter of **`WeakMethod`**. Patch by Géry Ogam.

- bpo-44817 [https://bugs.python.org/issue?@action=redirect&bpo=44817]: Ignore WinError 53 (ERROR_BAD_NETPATH), 65 (ERROR_NETWORK_ACCESS_DENIED) and 161 (ERROR_BAD_PATHNAME) when using ntpath.realpath().

- bpo-41260 [https://bugs.python.org/issue?@action=redirect&bpo=41260]: Rename the *fmt* parameter of the pure Python implementation of **`datetime.date.strftime()`** to *format*.

- bpo-15999 [https://bugs.python.org/issue?@action=redirect&bpo=15999]: All built-in functions now accept arguments of any type instead of just `bool` and `int` for boolean parameters.

Documentation

- gh-99931 [https://github.com/python/cpython/issues/99931]: Use sphinxext-opengraph [https://sphinxext-opengraph.readthedocs.io/] to generate OpenGraph metadata [https://ogp.me/].
- gh-89682 [https://github.com/python/cpython/issues/89682]: Reworded docstring of the default `__contains__` to clarify that it returns a **`bool`**.
- gh-88330 [https://github.com/python/cpython/issues/88330]: Improved the description of what a resource is in importlib.resources docs.
- gh-92892 [https://github.com/python/cpython/issues/92892]: Document that calling variadic functions with ctypes requires special care on macOS/arm64 (and possibly other platforms).
- bpo-41825 [https://bugs.python.org/issue?@action=redirect&bpo=41825]: Restructured the documentation for the **`os.wait*`** family of functions, and improved the docs for **`os.waitid()`** with more explanation of the possible argument constants.

Tests

- gh-99892 [https://github.com/python/cpython/issues/99892]: Skip test_normalization() of test_unicodedata if it fails to download NormalizationTest.txt file from pythontest.net. Patch by Victor Stinner.
- gh-99934 [https://github.com/python/cpython/issues/99934]: Correct test_marsh on (32 bit) x86: test_deterministic sets was failing.
- gh-99741 [https://github.com/python/cpython/issues/99741]: We've implemented multi-phase init (PEP 489/630/687) for the internal (for testing) _xxsubinterpreters module.
- gh-99659 [https://github.com/python/cpython/issues/99659]: Optional big memory tests in `test_sqlite3` now catch the correct **`sqlite.DataError`** exception type in case of too large strings and/or blobs passed.
- gh-99593 [https://github.com/python/cpython/issues/99593]: Cover the Unicode C API with tests.
- gh-96002 [https://github.com/python/cpython/issues/96002]: Add functional test for Argument Clinic.

Build

- gh-99086 [https://github.com/python/cpython/issues/99086]: Fix `-Wimplicit-int`, `-Wstrict-prototypes`, and `-Wimplicit-function-declaration` compiler warnings in **configure** checks.
- gh-99337 [https://github.com/python/cpython/issues/99337]: Fix a compilation issue with GCC 12 on macOS.
- gh-99289 [https://github.com/python/cpython/issues/99289]: Add a `COMPILEALL_OPTS` variable in Makefile to override **`compileall`** options (default: `-j0`) in `make install`. Also merged the `compileall` commands into a single command building .pyc files for the all optimization levels (0, 1, 2) at once. Patch by Victor Stinner.
- gh-98872 [https://github.com/python/cpython/issues/98872]: Fix a possible fd leak in `Programs/_freeze_module.c` introduced in Python 3.11.
- gh-88226 [https://github.com/python/cpython/issues/88226]: Always define `TARGET_*` labels in `Python/ceval.c`, even if `USE_COMPUTED_GOTOS` is disabled. This allows breakpoints

to be set at those labels in (for instance) `gdb`.

Windows

- gh-99345 [https://github.com/python/cpython/issues/99345]: Use faster initialization functions to detect install location for Windows Store package
- gh-98629 [https://github.com/python/cpython/issues/98629]: Fix initialization of **`sys.version`** and `sys._git` on Windows
- gh-99442 [https://github.com/python/cpython/issues/99442]: Fix handling in Python install manager when `argv[0]` does not include a file extension.
- bpo-40882 [https://bugs.python.org/issue?@action=redirect&bpo=40882]: Fix a memory leak in **`multiprocessing.shared_memory.SharedMemory`** on Windows.

macOS

- gh-87235 [https://github.com/python/cpython/issues/87235]: On macOS `python3 /dev/fd/9 9</path/to/script.py` failed for any script longer than a couple of bytes.
- gh-98940 [https://github.com/python/cpython/issues/98940]: Fix `Mac/Extras.install.py` file filter bug.

Tools/Demos

- gh-64490 [https://github.com/python/cpython/issues/64490]: Argument Clinic varargs bugfixes
    - Fix out-of-bounds error in **`_PyArg_UnpackKeywordsWithVararg()`**.
    - Fix incorrect check which allowed more than one varargs in clinic.py.
    - Fix miscalculation of `noptargs` in generated code.
    - Do not generate `noptargs` when there is a vararg argument and no optional argument.

C API

- gh-98680 [https://github.com/python/cpython/issues/98680]: `PyBUF_*` constants were marked as part of Limited API of Python 3.11+. These were available in 3.11.0 with **`Py_LIMITED_API`** defined for 3.11, and are necessary to use the buffer API.
- gh-99612 [https://github.com/python/cpython/issues/99612]: Fix **`PyUnicode_DecodeUTF8Stateful()`** for ASCII-only data: `*consumed` was not set.
- gh-47146 [https://github.com/python/cpython/issues/47146]: The `structmember.h` header is deprecated. Its non-deprecated contents are now available just by including `Python.h`, with a `Py_` prefix added if it was missing. (Deprecated contents are **`T_OBJECT`**, **`T_NONE`**, and no-op flags.) Patch by Petr Viktorin, based on earlier work by Alexander Belopolsky and Matthias Braun.

Python 3.12.0 alpha 2

Release date: 2022-11-14

Security

- gh-98433 [https://github.com/python/cpython/issues/98433]: The IDNA codec decoder used on DNS hostnames by **`socket`** or **`asyncio`** related name resolution functions no longer involves a quadratic algorithm. This prevents a potential CPU denial of service if an out-of-spec excessive length hostname involving bidirectional characters were decoded. Some protocols such as **`urllib`** http `3xx` redirects potentially allow for an attacker to supply such a name.

  Individual labels within an IDNA encoded DNS name will now raise an error early during IDNA decoding if they are longer than 1024 unicode characters given that each decoded DNS label must be 63 or fewer characters and the entire decoded DNS name is limited to 255. Only an application presenting a hostname or label consisting primarily of **RFC 3454** [https://datatracker.ietf.org/doc/html/rfc3454.html] section 3.1 «Nothing» characters to be removed would run into of this new limit. See also **RFC 5894** [https://datatracker.ietf.org/doc/html/rfc5894.html] section 6 and **RFC 3491** [https://datatracker.ietf.org/doc/html/rfc3491.html].

- gh-98739 [https://github.com/python/cpython/issues/98739]: Update bundled libexpat to 2.5.0

Core and Builtins

- gh-81057 [https://github.com/python/cpython/issues/81057]: The docs clearly say that `PyImport_Inittab`, `PyImport_AppendInittab()`, and `PyImport_ExtendInittab()` should not be used after `Py_Initialize()` has been called. We now enforce this for the two functions. Additionally, the runtime now uses an internal copy of `PyImport_Inittab`, to guard against modification.
- gh-99298 [https://github.com/python/cpython/issues/99298]: Fix an issue that could potentially cause incorrect error handling for some bytecode instructions.
- gh-99254 [https://github.com/python/cpython/issues/99254]: The compiler now removes all unused constants from code objects (except the first one, which may be a docstring).
- gh-99205 [https://github.com/python/cpython/issues/99205]: Fix an issue that prevented `PyThreadState` and `PyInterpreterState` memory from being freed properly.
- gh-81057 [https://github.com/python/cpython/issues/81057]: The 18 global C variables holding the state of the allocators have been moved to `_PyRuntimeState`. This is a strictly internal change with no change in behavior.
- gh-99181 [https://github.com/python/cpython/issues/99181]: Fix failure in `except*` with unhashable exceptions.
- gh-99204 [https://github.com/python/cpython/issues/99204]: Fix calculation of `sys._base_executable` when inside a POSIX virtual environment using copies of the python binary

when the base installation does not provide the executable name used by the venv. Calculation will fall back to alternative names («python<MAJOR>», «python<MAJOR>.<MINOR>»).

- gh-96055 [https://github.com/python/cpython/issues/96055]: Update `faulthandler` to emit an error message with the proper unexpected signal number. Patch by Donghee Na.

- gh-99153 [https://github.com/python/cpython/issues/99153]: Fix location of `SyntaxError` for a `try` block with both `except` and `except*`.

- gh-98686 [https://github.com/python/cpython/issues/98686]: Merge the adaptive opcode logic into each instruction's unquickened variant, and merge the logic in `EXTENDED_ARG_QUICK` into `EXTENDED_ARG`. With these changes, the quickening that happens at code object creation is now only responsible for initializing warmup counters and inserting superinstructions.

- gh-99103 [https://github.com/python/cpython/issues/99103]: Fix the error reporting positions of specialized traceback anchors when the source line contains Unicode characters.

- gh-99139 [https://github.com/python/cpython/issues/99139]: Improve the error suggestion for `NameError` exceptions for instances. Now if a `NameError` is raised in a method and the instance has an attribute that's exactly equal to the name in the exception, the suggestion will include `self.<NAME>` instead of the closest match in the method scope. Patch by Pablo Galindo

- gh-98401 [https://github.com/python/cpython/issues/98401]: Octal escapes with value larger than `0o377` (ex: `"\477"`), deprecated in Python 3.11, now produce a `SyntaxWarning`, instead of `DeprecationWarning`. In a future Python version they will be eventually a `SyntaxError`. Patch by Victor Stinner.

- gh-98401 [https://github.com/python/cpython/issues/98401]: A backslash-character pair that is not a valid escape sequence now generates a **`SyntaxWarning`**, instead of **`DeprecationWarning`**. For example, `re.compile("\d+\.\d+")` now emits a **`SyntaxWarning`** (`"\d"` is an invalid escape sequence), use raw strings for regular expression: `re.compile(r"\d+\.\d+")`. In a future Python version, **`SyntaxError`** will eventually be raised, instead of **`SyntaxWarning`**. Patch by Victor Stinner.

- gh-96793 [https://github.com/python/cpython/issues/96793]: Handle StopIteration and StopAsyncIteration raised in generator or coroutines in the bytecode, rather than in wrapping C code.

- gh-98931 [https://github.com/python/cpython/issues/98931]: Improve the **`SyntaxError`** error message when the user types `import x from y` instead of `from y import x`. Patch by Pablo Galindo

- gh-98852 [https://github.com/python/cpython/issues/98852]: Fix subscription of type aliases containing bare generic types or types like **`TypeVar`**: for example `tuple[A, T][int]` and `tuple[TypeVar, T][int]`, where `A` is a generic type, and `T` is a type variable.

- gh-98925 [https://github.com/python/cpython/issues/98925]: Lower the recursion depth for marshal on WASI to support (in-development) wasmtime 2.0.

- gh-98783 [https://github.com/python/cpython/issues/98783]: Fix multiple crashes in debug mode when `str` subclasses are used instead of `str` itself.

- gh-98811 [https://github.com/python/cpython/issues/98811]: Use complete source locations to simplify detection of `__future__` imports which are not at the beginning of the file. Also corrects the offset in the exception raised in one case, which was off by one and impeded highlighting.

- gh-96793 [https://github.com/python/cpython/issues/96793]: Add specialization of **`FOR_ITER`** for generators. Saves multiple layers of dispatch and checking to get from the **`FOR_ITER`** instruction in the caller to the **`RESUME`** in the generator.

- gh-98762 [https://github.com/python/cpython/issues/98762]: Fix source locations of `match` sub-patterns.
- gh-98586 [https://github.com/python/cpython/issues/98586]: Added the methods `PyObject_Vectorcall()` and `PyObject_VectorcallMethod()` to the Limited API along with the auxiliary macro constant `PY_VECTORCALL_ARGUMENTS_OFFSET`.

  The availability of these functions enables more efficient **PEP 590** [https://peps.python.org/pep-0590/] vector calls from binary extension modules that avoid argument boxing/unboxing overheads.
- gh-99257 [https://github.com/python/cpython/issues/99257]: Fix an issue where member descriptors (such as those for `__slots__`) could behave incorrectly or crash instead of raising a `TypeError` when accessed via an instance of an invalid type.
- gh-93143 [https://github.com/python/cpython/issues/93143]: Rather than changing `co_code`, the interpreter will now display a `RuntimeWarning` and assign `None` to any fast locals that are left unbound after jumps or `del` statements executed while tracing.
- gh-96421 [https://github.com/python/cpython/issues/96421]: When calling into Python code from C code, through `PyEval_EvalFrameEx()` or a related C-API function, a shim frame in inserted into the call stack. This occurs in the `_PyEval_EvalFrameDefault()` function. The extra frame should be invisible to all Python and most C extensions, but out-of-process profilers and debuggers need to be aware of it. These shim frames can be detected by checking `frame->owner == FRAME_OWNED_BY_CSTACK`.

  Extensions implementing their own interpreters using PEP 523 need to be aware of this shim frame and the changes to the semantics of `RETURN_VALUE`, `YIELD_VALUE`, and `RETURN_GENERATOR`, which now clear the frame.

Build

- gh-98415 [https://github.com/python/cpython/issues/98415]: Fix detection of MAC addresses for **`uuid`** on certain OSs. Patch by Chaim Sanders

Core and Builtins

- gh-98686 [https://github.com/python/cpython/issues/98686]: Quicken all code objects, and specialize adaptive bytecode instructions more aggressively.
- gh-92119 [https://github.com/python/cpython/issues/92119]: Print exception class name instead of its string representation when raising errors from **`ctypes`** calls.
- gh-91058 [https://github.com/python/cpython/issues/91058]: **`ImportError`** raised from failed `from <module> import <name>` now include suggestions for the value of `<name>` based on the available names in `<module>`. Patch by Pablo Galindo
- gh-96793 [https://github.com/python/cpython/issues/96793]: The **`FOR_ITER`** now leaves the iterator on the stack on termination of the loop. This is to assist specialization of loops for generators.
- gh-90716 [https://github.com/python/cpython/issues/90716]: Add _pylong.py module. It includes asymptotically faster algorithms that can be used for operations on integers with many digits. It is used by longobject.c to speed up some operations.
- gh-95389 [https://github.com/python/cpython/issues/95389]: Expose **`ETH_P_ALL`** and some of the ETHERTYPE_* constants in **`socket`**. Patch by Noam Cohen.

Library

- gh-93696 [https://github.com/python/cpython/issues/93696]: Allow **`pdb`** to locate source for frozen modules in the standard library.
- gh-99418 [https://github.com/python/cpython/issues/99418]: Fix bug in **`urllib.parse.urlparse()`** that causes URL schemes that begin with a digit, a plus sign, or a minus sign to be

parsed incorrectly.

- gh-94597 [https://github.com/python/cpython/issues/94597]: Deprecate **`asyncio.AbstractChildWatcher`** to be removed in Python 3.14. Patch by Kumar Aditya.
- gh-99305 [https://github.com/python/cpython/issues/99305]: Improve performance of **`secrets.token_hex()`**.
- gh-74044 [https://github.com/python/cpython/issues/74044]: Fixed bug where **`inspect.signature()`** reported incorrect arguments for decorated methods.
- gh-99275 [https://github.com/python/cpython/issues/99275]: Fix `SystemError` in **`ctypes`** when exception was not set during `__initsubclass__`.
- gh-99277 [https://github.com/python/cpython/issues/99277]: Remove older version of `_SSLProtocolTransport.get_write_buffer_limits` in **`asyncio.sslproto`**
- gh-99248 [https://github.com/python/cpython/issues/99248]: fix negative numbers failing in verify()
- gh-99155 [https://github.com/python/cpython/issues/99155]: Fix **`statistics.NormalDist`** pickle with `0` and `1` protocols.
- gh-93464 [https://github.com/python/cpython/issues/93464]: `enum.auto()` is now correctly activated when combined with other assignment values. E.g. `ONE = auto(), 'some text'` will now evaluate as `(1, 'some text')`.
- gh-99134 [https://github.com/python/cpython/issues/99134]: Update the bundled copy of pip to version 22.3.1.
- gh-92584 [https://github.com/python/cpython/issues/92584]: Remove the `distutils` package. It was deprecated in Python 3.10 by **PEP 632** [https://peps.python.org/pep-0632/] «Deprecate distutils module». For projects still using `distutils` and cannot be updated to something else, the `setuptools` project can be installed: it still provides `distutils`. Patch by Victor Stinner.
- gh-98999 [https://github.com/python/cpython/issues/98999]: Now **`_pyio`** is consistent with **`_io`** in raising `ValueError` when executing methods over closed buffers.
- gh-83004 [https://github.com/python/cpython/issues/83004]: Clean up refleak on failed module initialisation in **`_zoneinfo`**
- gh-83004 [https://github.com/python/cpython/issues/83004]: Clean up refleaks on failed module initialisation in **`_pickle`**

- gh-83004 [https://github.com/python/cpython/issues/83004]: Clean up refleak on failed module initialisation in **`_io`**.
- gh-98897 [https://github.com/python/cpython/issues/98897]: Fix memory leak in **`math.dist()`** when both points don't have the same dimension. Patch by Kumar Aditya.
- gh-98878 [https://github.com/python/cpython/issues/98878]: Use the frame bound builtins when offering a name suggestion in **`traceback`** to prevent crashing when `__builtins__` is not a dict.
- gh-98139 [https://github.com/python/cpython/issues/98139]: In **`importlib._bootstrap`**, enhance namespace package repr to `<module 'x' (namespace) from ['path']>`.
- gh-90352 [https://github.com/python/cpython/issues/90352]: Fix `_SelectorDatagramTransport` to inherit from **`DatagramTransport`** in **`asyncio`**. Patch by Kumar Aditya.
- gh-98793 [https://github.com/python/cpython/issues/98793]: Fix argument typechecks in **`_overlapped.WSAConnect()`** and **`_overlapped.Overlapped.WSASendTo()`** functions.
- gh-98744 [https://github.com/python/cpython/issues/98744]: Prevent crashing in **`traceback`** when retrieving the byte-offset for some source files that contain certain unicode characters.
- gh-98740 [https://github.com/python/cpython/issues/98740]: Fix internal error in the **`re`** module which in very rare circumstances prevented compilation of a regular expression containing a conditional expression without the «else» branch.
- gh-98703 [https://github.com/python/cpython/issues/98703]: Fix **`asyncio.StreamWriter.drain()`** to call `protocol.connection_lost` callback only once on Windows.
- gh-98624 [https://github.com/python/cpython/issues/98624]: Add a mutex to unittest.mock.NonCallableMock to protect concurrent access to mock attributes.
- gh-98658 [https://github.com/python/cpython/issues/98658]: The **`array.array`** class now supports subscripting, making it a generic type.
- gh-98284 [https://github.com/python/cpython/issues/98284]: Improved **`TypeError`** message for undefined abstract methods of a **`abc.ABC`** instance. The names of the missing methods are surrounded by single-quotes to highlight them.

- gh-96151 [https://github.com/python/cpython/issues/96151]: Allow `BUILTINS` to be a valid field name for frozen dataclasses.
- gh-98086 [https://github.com/python/cpython/issues/98086]: Make sure `patch.dict()` can be applied on async functions.
- gh-72719 [https://github.com/python/cpython/issues/72719]: Remove modules **`asyncore`** and **`asynchat`**, which were deprecated by **PEP 594** [https://peps.python.org/pep-0594/].
- gh-96192 [https://github.com/python/cpython/issues/96192]: Fix handling of `bytes` path-like objects in **`os.ismount()`**.
- gh-94172 [https://github.com/python/cpython/issues/94172]: **`ftplib`**: Remove the `FTP_TLS.ssl_version` class attribute: use the *context* parameter instead. Patch by Victor Stinner
- gh-94172 [https://github.com/python/cpython/issues/94172]: Remove the *keyfile* and *certfile* parameters from the **`ftplib`**, **`imaplib`**, **`poplib`** and **`smtplib`** modules, and the *key_file*, *cert_file* and *check_hostname* parameters from the **`http.client`** module, all deprecated since Python 3.6. Use the *context* parameter (*ssl_context* in **`imaplib`**) instead. Patch by Victor Stinner.
- gh-83638 [https://github.com/python/cpython/issues/83638]: Add the **`autocommit`** attribute to **`sqlite3.Connection`** and the *autocommit* parameter to **`sqlite3.connect()`** to control **PEP 249** [https://peps.python.org/pep-0249/]-compliant transaction handling. Patch by Erlend E. Aasland.
- gh-92452 [https://github.com/python/cpython/issues/92452]: Fixed a race condition that could cause **`sysconfig.get_config_var()`** to incorrectly return **`None`** in multi-threaded programs.
- gh-91803 [https://github.com/python/cpython/issues/91803]: Fix an error when using a method of objects mocked with **`unittest.mock.create_autospec()`** after it was sealed with **`unittest.mock.seal()`** function.
- bpo-38523 [https://bugs.python.org/issue?@action=redirect&bpo=38523]: **`shutil.copytree()`** now applies the *ignore_dangling_symlinks* argument recursively.
- bpo-40358 [https://bugs.python.org/issue?@action=redirect&bpo=40358]: Add walk_up argument in **`pathlib.PurePath.relative_to()`**.
- bpo-36267 [https://bugs.python.org/issue?

@action=redirect&bpo=36267]: Fix IndexError in `argparse.ArgumentParser` when a `store_true` action is given an explicit argument.

Documentation

- gh-98832 [https://github.com/python/cpython/issues/98832]: Changes wording of docstring for `pathlib.Path.iterdir()`.
- gh-97966 [https://github.com/python/cpython/issues/97966]: Update uname docs to clarify the special nature of the platform attribute and to indicate when it became late-bound.

Tests

- gh-98903 [https://github.com/python/cpython/issues/98903]: The Python test suite now fails with exit code 4 if no tests ran. It should help detecting typos in test names and test methods.
- gh-98713 [https://github.com/python/cpython/issues/98713]: Fix a bug in the `typing` tests where a test relying on CPython-specific implementation details was not decorated with `@cpython_only` and was not skipped on other implementations.
- gh-87390 [https://github.com/python/cpython/issues/87390]: Add tests for star-unpacking with PEP 646, and some other miscellaneous PEP 646 tests.
- gh-96853 [https://github.com/python/cpython/issues/96853]: Added explicit coverage of `Py_Initialize` (and hence `Py_InitializeEx`) back to the embedding tests (all other embedding tests migrated to `Py_InitializeFromConfig` in Python 3.11)
- bpo-34272 [https://bugs.python.org/issue?@action=redirect&bpo=34272]: Some C API tests were moved into the new Lib/test/test_capi/ directory.

Build

- gh-99086 [https://github.com/python/cpython/issues/99086]: Fix `-Wimplicit-int` compiler warning in **configure** check for `PTHREAD_SCOPE_SYSTEM`.
- gh-99016 [https://github.com/python/cpython/issues/99016]: Fix

build with `PYTHON_FOR_REGEN=python3.8`.

- gh-97731 [https://github.com/python/cpython/issues/97731]: Specify the full path to the source location for `make docclean` (needed for cross-builds).
- gh-98949 [https://github.com/python/cpython/issues/98949]: Drop unused build dependency on `readelf`.
- gh-98989 [https://github.com/python/cpython/issues/98989]: Use `python3.11`, if available, for regeneration and freezing.
- gh-98831 [https://github.com/python/cpython/issues/98831]: Add new tooling, in `Tools/cases_generator`, to generate the interpreter switch statement from a list of opcode definitions. This only affects adding, modifying or removing instruction definitions. The instruction definitions now live in `Python/bytecodes.c`, in the form of a custom DSL (under development) [https://github.com/faster-cpython/ideas/blob/main/3.12/interpreter_definition.md]. The tooling reads this file and writes `Python/generated_cases.c.h`, which is then included by `Python/ceval.c` to provide most of the cases of the main interpreter switch.
- gh-98817 [https://github.com/python/cpython/issues/98817]: Remove PCbuild/lib.pyproj: it's not used for anything, is only a minor convenience for Visual Studio users (who probably mostly don't even know about it), and it takes a lot of maintenance effort to keep updated.
- gh-98776 [https://github.com/python/cpython/issues/98776]: Fix `make regen-test-levenshtein` for out-of-tree builds.
- gh-98707 [https://github.com/python/cpython/issues/98707]: Don't use vendored `libmpdec` headers if **`--with-system-libmpdec`** is passed to **configure**. Don't use vendored `libexpat` headers if **`--with-system-expat`** is passed to **configure**.

Windows

- gh-98689 [https://github.com/python/cpython/issues/98689]: Update Windows builds to zlib v1.2.13. v1.2.12 has **CVE 2022-37434** [https://www.cve.org/CVERecord?id=CVE-2022-37434], but the vulnerable `inflateGetHeader` API is not used by Python.
- gh-98790 [https://github.com/python/cpython/issues/98790]: Assumes that a missing `DLLs` directory means that standard extension

modules are in the executable's directory.

- gh-98745 [https://github.com/python/cpython/issues/98745]: Update `py.exe` launcher to install 3.11 by default and 3.12 on request.
- gh-98692 [https://github.com/python/cpython/issues/98692]: Fix the Python install manager ignoring unrecognized shebang lines instead of treating them as local paths
- gh-94328 [https://github.com/python/cpython/issues/94328]: Update Windows installer to use SQLite 3.39.4.

macOS

- gh-94328 [https://github.com/python/cpython/issues/94328]: Update macOS installer to SQLite 3.39.4.

C API

- gh-98724 [https://github.com/python/cpython/issues/98724]: The **`Py_CLEAR`**, **`Py_SETREF`** and **`Py_XSETREF`** macros now only evaluate their argument once. If the argument has side effects, these side effects are no longer duplicated. Patch by Victor Stinner.
- gh-98978 [https://github.com/python/cpython/issues/98978]: Fix use-after-free in `Py_SetPythonHome(NULL)`, `Py_SetProgramName(NULL)` and `_Py_SetProgramFullPath(NULL)` function calls. Issue reported by Benedikt Reinartz. Patch by Victor Stinner.
- gh-98410 [https://github.com/python/cpython/issues/98410]: Add `getbufferproc` and `releasebufferproc` to the stable API.
- gh-98610 [https://github.com/python/cpython/issues/98610]: Some configurable capabilities of sub-interpreters have changed. They always allow subprocesses (**`subprocess`**) now, whereas before subprocesses could be optionally disallowed for a sub-interpreter. Instead **`os.exec()`** can now be disallowed. Disallowing daemon threads is now supported. Disallowing all threads is still allowed, but is never done by default. Note that the optional restrictions are only available through `_Py_NewInterpreterFromConfig()`, which isn't a public API. They do not affect the main interpreter, nor

`Py_NewInterpreter()`.

- gh-98608 [https://github.com/python/cpython/issues/98608]: A `_PyInterpreterConfig` has been added and `_Py_NewInterpreter()` has been renamed to `_Py_NewInterpreterFromConfig()`. The «isolated_subinterpreters» argument is now a granular config that captures the previous behavior. Note that this is all «private» API.
- gh-96853 [https://github.com/python/cpython/issues/96853]: `Py_InitializeEx` now correctly calls `PyConfig_Clear` after initializing the interpreter (the omission didn't cause a memory leak only because none of the dynamically allocated config fields are populated by the wrapper function)
- gh-91248 [https://github.com/python/cpython/issues/91248]: Add **`PyFrame_GetVar()`** and **`PyFrame_GetVarString()`** functions to get a frame variable by its name. Patch by Victor Stinner.

Python 3.12.0 alpha 1

Release date: 2022-10-25

Security

- gh-97616 [https://github.com/python/cpython/issues/97616]: Fix multiplying a list by an integer (`list *= int`): detect the integer overflow when the new allocated length is close to the maximum size. Issue reported by Jordan Limor. Patch by Victor Stinner.

- gh-97514 [https://github.com/python/cpython/issues/97514]: On Linux the **`multiprocessing`** module returns to using filesystem backed unix domain sockets for communication with the *forkserver* process instead of the Linux abstract socket namespace. Only code that chooses to use the «forkserver» start method is affected.

  Abstract sockets have no permissions and could allow any user on the system in the same network namespace [https://man7.org/linux/man-pages/man7/network_namespaces.7.html] (often the whole system) to inject code into the multiprocessing *forkserver* process. This was a potential privilege escalation. Filesystem based socket permissions restrict this to the *forkserver* process user as was the default in Python 3.8 and earlier.

This prevents Linux **CVE 2022-42919** [https://www.cve.org/CVERecord?id=CVE-2022-42919].

- gh-87389 [https://github.com/python/cpython/issues/87389]: **`http.server`**: Fix an open redirection vulnerability in the HTTP server when an URI path starts with `//`. Vulnerability discovered, and initial fix proposed, by Hamza Avvan.

- gh-79096 [https://github.com/python/cpython/issues/79096]: LWPCookieJar and MozillaCookieJar create files with file mode 600 instead of 644 (Microsoft Windows is not affected)

- gh-92888 [https://github.com/python/cpython/issues/92888]: Fix `memoryview` use after free when accessing the backing buffer in certain cases.

- gh-68966 [https://github.com/python/cpython/issues/68966]: The deprecated mailcap module now refuses to inject unsafe text (filenames, MIME types, parameters) into shell commands. Instead of using such text, it will warn and act as if a match was not found (or for test commands, as if the test failed).

Core and Builtins

- gh-98374 [https://github.com/python/cpython/issues/98374]: Suppress ImportError for invalid query for help() command. Patch by Donghee Na.
- gh-98461 [https://github.com/python/cpython/issues/98461]: Fix source location in bytecode for list, set and dict comprehensions as well as generator expressions.
- gh-98354 [https://github.com/python/cpython/issues/98354]: Added unicode check for `name` attribute of `spec` argument passed in **`_imp.create_builtin()`** function.
- gh-98398 [https://github.com/python/cpython/issues/98398]: Fix source location of “assert” bytecodes.

- gh-98390 [https://github.com/python/cpython/issues/98390]: Fix location of sub-expressions of boolean expressions, by reducing their scope to that of the sub-expression.
- gh-98254 [https://github.com/python/cpython/issues/98254]: Modules from the standard library are now potentially suggested as part of the error messages displayed by the interpreter when an `NameError` is raised to the top level. Patch by Pablo Galindo
- gh-97997 [https://github.com/python/cpython/issues/97997]: Add running column offset to the tokenizer state to avoid calculating AST column information with pointer arithmetic.
- gh-97973 [https://github.com/python/cpython/issues/97973]: Modify the tokenizer to return all necessary information the parser needs to set location information in the AST nodes, so that the parser does not have to calculate those doing pointer arithmetic.

Library

- gh-96078 [https://github.com/python/cpython/issues/96078]: `os.sched_yield()` now release the GIL while calling sched_yield(2). Patch by Donghee Na.
- gh-97955 [https://github.com/python/cpython/issues/97955]: Migrate `zoneinfo` to Argument Clinic.

Core and Builtins

- gh-97912 [https://github.com/python/cpython/issues/97912]: The compiler now avoids quadratic behavior when finding which instructions should use the `LOAD_FAST_CHECK` opcode.

- gh-97002 [https://github.com/python/cpython/issues/97002]: Fix an issue where several frame objects could be backed by the same interpreter frame, possibly leading to corrupted memory and hard crashes of the interpreter.

- gh-97943 [https://github.com/python/cpython/issues/97943]: Bugfix: `PyFunction_GetAnnotations()` should return a borrowed reference. It was returning a new reference.

- gh-97922 [https://github.com/python/cpython/issues/97922]: The Garbage Collector now runs only on the eval breaker mechanism of the Python bytecode evaluation loop instead on object allocations. The GC can also run when **`PyErr_CheckSignals()`** is called so C extensions that need to run for a long time without executing any Python code also have a chance to execute the GC periodically.

- gh-65961 [https://github.com/python/cpython/issues/65961]: When `__package__` is different than `__spec__.parent`, raise a `DeprecationWarning` instead of `ImportWarning`.

  Also remove `importlib.util.set_package()` which was scheduled for removal.

- gh-97850 [https://github.com/python/cpython/issues/97850]: Long deprecated, `module_repr()` should now be completely eradicated.

- gh-86298 [https://github.com/python/cpython/issues/86298]: In cases where `warnings.warn_explicit()` consults the module's loader, an `DeprecationWarning` is issued when `m.__loader__` differs from `m.__spec__.loader`.

- gh-97779 [https://github.com/python/cpython/issues/97779]: Ensure that all Python frame objects are backed by «complete» frames.

- gh-91052 [https://github.com/python/cpython/issues/91052]: Add API for subscribing to modification events on selected dictionaries.

- gh-97752 [https://github.com/python/cpython/issues/97752]: Fix possible data corruption or crashes when accessing the `f_back` member of newly-created generator or coroutine frames.

- gh-97591 [https://github.com/python/cpython/issues/97591]: Fixed a missing incref/decref pair in `Exception.__setstate__()`. Patch by Ofey Chan.

- gh-97670 [https://github.com/python/cpython/issues/97670]: Remove the **`sys.getdxp()`** function and the `Tools/scripts/analyze_dxp.py` script. DXP stands for «dynamic execution pairs». They were related to `DYNAMIC_EXECUTION_PROFILE` and `DXPAIRS` macros which have been removed in Python 3.11. Python can now be built with **`./configure --enable-pystats`** to gather statistics on Python opcodes. Patch by Victor Stinner.

- gh-94526 [https://github.com/python/cpython/issues/94526]: Fix the Python path configuration used to initialized **`sys.path`** at Python startup. Paths are no longer encoded to UTF-8/strict to avoid encoding errors if it contains surrogate characters (bytes paths are decoded with the surrogateescape error handler). Patch by Victor Stinner.

- gh-96670 [https://github.com/python/cpython/issues/96670]: The parser now raises **`SyntaxError`** when parsing source code containing null bytes. Patch by Pablo Galindo

- gh-96975 [https://github.com/python/cpython/issues/96975]: Fix a crash occurring when **`PyEval_GetFrame()`** is called while the topmost Python frame is in a partially-initialized state.

- gh-96848 [https://github.com/python/cpython/issues/96848]: Fix command line parsing: reject **`-X int_max_str_digits`** option with no value (invalid) when the **`PYTHONINTMAXSTRDIGITS`** environment variable is set to a valid limit. Patch by Victor Stinner.

- gh-95921 [https://github.com/python/cpython/issues/95921]: Fix overly-broad source position information for chained comparisons used as branching conditions.

Library

- gh-96821 [https://github.com/python/cpython/issues/96821]: Fix undefined behaviour in `audioop.c`.

Core and Builtins

- gh-96821 [https://github.com/python/cpython/issues/96821]: Fix undefined behaviour in `_testcapimodule.c`.
- gh-95778 [https://github.com/python/cpython/issues/95778]: When **`ValueError`** is raised if an integer is larger than the limit, mention the **`sys.set_int_max_str_digits()`** function in the error message. Patch by Victor Stinner.
- gh-96387 [https://github.com/python/cpython/issues/96387]: At Python exit, sometimes a thread holding the GIL can wait forever for a thread (usually a daemon thread) which requested to drop the GIL, whereas the thread already exited. To fix the race condition, the thread which requested the GIL drop now resets its request before exiting. Issue discovered and analyzed by Mingliang ZHAO. Patch by Victor Stinner.
- gh-96864 [https://github.com/python/cpython/issues/96864]: Fix a possible assertion failure, fatal error, or **`SystemError`** if a line tracing event raises an exception while opcode tracing is enabled.
- gh-95778 [https://github.com/python/cpython/issues/95778]: The `PyLong_FromString` function was refactored to make it more maintainable and extensible.
- gh-96678 [https://github.com/python/cpython/issues/96678]: Fix undefined behaviour in C code of null pointer arithmetic.
- gh-96754 [https://github.com/python/cpython/issues/96754]: Make sure that all frame objects created are created from valid interpreter frames. Prevents the possibility of invalid frames in backtraces and signal handlers.
- gh-90997 [https://github.com/python/cpython/issues/90997]: Improve the performance of reading and writing inline bytecode caches on some platforms.
- gh-96751 [https://github.com/python/cpython/issues/96751]: Remove dead code from `CALL_FUNCTION_EX` opcode.
- gh-90751 [https://github.com/python/cpython/issues/90751]: **`memoryview`** now supports half-floats. Patch by Donghee Na and Antoine Pitrou.
- gh-96678 [https://github.com/python/cpython/issues/96678]: Fix case of undefined behavior in ceval.c

Library

- gh-64373 [https://github.com/python/cpython/issues/64373]: Convert

`_functools` to argument clinic.
- gh-96641 [https://github.com/python/cpython/issues/96641]: Do not expose `KeyWrapper` in **`_functools`**.

Core and Builtins

- gh-96636 [https://github.com/python/cpython/issues/96636]: Ensure that tracing, `sys.settrace()`, is turned on immediately. In pre-release versions of 3.11, some tracing events might have been lost when turning on tracing in a `__del__` method or interrupt.
- gh-96572 [https://github.com/python/cpython/issues/96572]: Fix use after free in trace refs build mode. Patch by Kumar Aditya.
- gh-96611 [https://github.com/python/cpython/issues/96611]: When loading a file with invalid UTF-8 inside a multi-line string, a correct SyntaxError is emitted.
- gh-96612 [https://github.com/python/cpython/issues/96612]: Make sure that incomplete frames do not show up in tracemalloc traces.
- gh-90230 [https://github.com/python/cpython/issues/90230]: Fix compiler warnings and test failures when building with `--enable-pystats`.
- gh-96587 [https://github.com/python/cpython/issues/96587]: Correctly raise `SyntaxError` on exception groups (**PEP 654** [https://peps.python.org/pep-0654/]) on python versions prior to 3.11
- gh-96569 [https://github.com/python/cpython/issues/96569]: Remove two cases of undefined behavior, by adding NULL checks.
- gh-96582 [https://github.com/python/cpython/issues/96582]: Fix possible `NULL` pointer dereference in `_PyThread_CurrentFrames`. Patch by Kumar Aditya.
- gh-91079 [https://github.com/python/cpython/issues/91079]: Separate Python recursion checking from C recursion checking which

reduces the chance of C stack overflow and allows the recursion limit to be increased safely.

- gh-93911 [https://github.com/python/cpython/issues/93911]: Fix an issue that could prevent **`LOAD_ATTR`** from specializing properly when accessing properties.
- gh-96348 [https://github.com/python/cpython/issues/96348]: Emit a DeprecationWarning when **`throw()`**, **`throw()`** or **`athrow()`** are called with more than one argument.
- gh-95196 [https://github.com/python/cpython/issues/95196]: Disable incorrect pickling of the C implemented classmethod descriptors.
- gh-96364 [https://github.com/python/cpython/issues/96364]: Fix text signatures of `list.__getitem__` and `dict.__getitem__`.
- gh-96352 [https://github.com/python/cpython/issues/96352]: Fix **`AttributeError`** missing `name` and `obj` attributes in **`object.__getattribute__()`**. Patch by Philip Georgi.
- gh-93554 [https://github.com/python/cpython/issues/93554]: Change the jump opcodes so that all conditional jumps are forward jumps. Backward jumps are converted by the assembler into a conditional forward jump whose target is the fallthrough block (and with a reversed condition), followed by an unconditional backward jump. For example:

  `POP_JUMP_IF_TRUE BACKWARD_TARGET` becomes `POP_JUMP_IF_FALSE NEXT_BLOCK; JUMP BACKWARD_TARGET`.

  All the directed conditional jump opcodes were removed: `POP_JUMP_FORWARD_IF_TRUE`, `POP_JUMP_BACKWARD_IF_TRUE`, `POP_JUMP_FORWARD_IF_FALSE`, `POP_JUMP_BACKWARD_IF_FALSE`, `POP_JUMP_FORWARD_IF_NONE`, `POP_JUMP_BACKWARD_IF_NONE`, `POP_JUMP_FORWARD_IF_NOT_NONE`, `POP_JUMP_BACKWARD_IF_NOT_NONE`.

The corresponding opcodes without direction are no longer pseudo-instructions, and they implement the forward conditional jumps.

- gh-96268 [https://github.com/python/cpython/issues/96268]: Loading a file with invalid UTF-8 will now report the broken character at the correct location.
- gh-96237 [https://github.com/python/cpython/issues/96237]: The internal field `_PyInterpreterFrame.f_func` is renamed to `_PyInterpreterFrame.f_funcobj` and may be any object. The `f_globals` and `f_builtin` fields may hold junk values.

  It is safest to treat the `_PyInterpreterFrame` struct as opaque.
- gh-96187 [https://github.com/python/cpython/issues/96187]: Fixed a bug that caused `_PyCode_GetExtra` to return garbage for negative indexes. Patch by Pablo Galindo
- gh-96143 [https://github.com/python/cpython/issues/96143]: Add a new `-X perf` Python command line option as well as **`sys.activate_stack_trampoline()`** and **`sys.deactivate_stack_trampoline()`** function in the **`sys`** module that allows to set/unset the interpreter in a way that the Linux `perf` profiler can detect Python calls. The new **`sys.is_stack_trampoline_active()`** function allows to query the state of the perf trampoline. Design by Pablo Galindo. Patch by Pablo Galindo and Christian Heimes with contributions from Gregory P. Smith [Google] and Mark Shannon.
- gh-96071 [https://github.com/python/cpython/issues/96071]: Fix a deadlock in **`PyGILState_Ensure()`** when allocating new thread state. Patch by Kumar Aditya.
- gh-96046 [https://github.com/python/cpython/issues/96046]:

`PyType_Ready()` now initializes `ht_cached_keys` and performs additional checks to ensure that type objects are properly configured. This avoids crashes in 3rd party packages that don't use regular API to create new types.

- gh-96005 [https://github.com/python/cpython/issues/96005]: On WASI `ENOTCAPABLE` is now mapped to `PermissionError`. The `errno` modules exposes the new error number. `getpath.py` now ignores **`PermissionError`** when it cannot open landmark files `pybuilddir.txt` and `pyenv.cfg`.

- gh-93678 [https://github.com/python/cpython/issues/93678]: Added test a harness for direct unit tests of the compiler's optimization stage. The `_testinternalcapi.optimize_cfg()` function runs the optimiser on a sequence of instructions. The `CfgOptimizationTestCase` class in `test.support` has utilities for invoking the optimizer and checking the output.

- gh-95245 [https://github.com/python/cpython/issues/95245]: Reduces the size of a «simple» Python object from 8 to 6 words by moving the weakreflist pointer into the pre-header directly before the object's dict/values pointer.

- gh-90997 [https://github.com/python/cpython/issues/90997]: Compile virtual `try`/`except` blocks to handle exceptions raised during `close()` or `throw()` calls through a suspended frame.

- gh-95977 [https://github.com/python/cpython/issues/95977]: Optimized calling `__get__()` with vectorcall. Patch by Kumar Aditya.

- gh-91210 [https://github.com/python/cpython/issues/91210]: Improve error message when a parameter without a default value follows one with a default value, and show the same message, even when the non-default/default sequence is preceded by positional-only parameters.

- gh-95922 [https://github.com/python/cpython/issues/95922]: Fixed

bug where the compiler's `eliminate_empty_basic_blocks` function ignores the last block of the code unit.

- gh-95818 [https://github.com/python/cpython/issues/95818]: Skip over incomplete frames in `PyThreadState_GetFrame()`.
- gh-95876 [https://github.com/python/cpython/issues/95876]: Fix format string in `_PyPegen_raise_error_known_location` that can lead to memory corruption on some 64bit systems. The function was building a tuple with `i` (int) instead of `n` (Py_ssize_t) for Py_ssize_t arguments.
- gh-95605 [https://github.com/python/cpython/issues/95605]: Fix misleading contents of error message when converting an all-whitespace string to `float`.
- gh-95150 [https://github.com/python/cpython/issues/95150]: Update code object hashing and equality to consider all debugging and exception handling tables. This fixes an issue where certain non-identical code objects could be «deduplicated» during compilation.
- gh-91146 [https://github.com/python/cpython/issues/91146]: Reduce allocation size of `list` from `str.split()` and `str.rsplit()`. Patch by Donghee Na and Inada Naoki.
- gh-87092 [https://github.com/python/cpython/issues/87092]: Create a "jump target label" abstraction in the compiler so that the compiler's codegen stage does not work directly with basic blocks. This prepares the code for changes to the underlying CFG generation mechanism.
- gh-95355 [https://github.com/python/cpython/issues/95355]: `_PyPegen_Parser_New` now properly detects token memory allocation errors. Patch by Honglin Zhu.
- gh-90081 [https://github.com/python/cpython/issues/90081]: Run Python code in tracer/profiler function at full speed. Fixes slowdown in earlier versions of 3.11.

- gh-95324 [https://github.com/python/cpython/issues/95324]: Emit a warning in debug mode if an object does not call **`PyObject_GC_UnTrack()`** before deallocation. Patch by Pablo Galindo.

- gh-95245 [https://github.com/python/cpython/issues/95245]: Merge managed dict and values pointer into a single tagged pointer to save one word in the pre-header.

- gh-93678 [https://github.com/python/cpython/issues/93678]: Add cfg_builder struct and refactor the relevant code so that a cfg can be constructed without an instance of the compiler struct.

- gh-95185 [https://github.com/python/cpython/issues/95185]: Prevented crashes in the AST constructor when compiling some absurdly long expressions like `"+0"*1000000`. **`RecursionError`** is now raised instead. Patch by Pablo Galindo

- gh-93351 [https://github.com/python/cpython/issues/93351]: **`ast.AST`** node positions are now validated when provided to **`compile()`** and other related functions. If invalid positions are detected, a **`ValueError`** will be raised.

- gh-94438 [https://github.com/python/cpython/issues/94438]: Fix an issue that caused extended opcode arguments and some conditional pops to be ignored when calculating valid jump targets for assignments to the `f_lineno` attribute of frame objects. In some cases, this could cause inconsistent internal state, resulting in a hard crash of the interpreter.

- gh-95060 [https://github.com/python/cpython/issues/95060]: Undocumented `PyCode_Addr2Location` function now properly returns when `addrq` argument is less than zero.

- gh-95113 [https://github.com/python/cpython/issues/95113]: Replace all `EXTENDED_ARG_QUICK` instructions with basic **`EXTENDED_ARG`** instructions in unquickened code. Consumers of non-adaptive bytecode should be able to handle extended arguments the same way they were handled in CPython 3.10 and older.

- gh-91409 [https://github.com/python/cpython/issues/91409]: Fix incorrect source location info caused by certain optimizations in the bytecode compiler.

Library

- gh-95023 [https://github.com/python/cpython/issues/95023]: Implement **`os.setns()`** and **`os.unshare()`** for Linux. Patch by Noam Cohen.

Core and Builtins

- gh-94036 [https://github.com/python/cpython/issues/94036]: Fix incorrect source location info for some multi-line attribute accesses and method calls.
- gh-94938 [https://github.com/python/cpython/issues/94938]: Fix error detection in some builtin functions when keyword argument name is an instance of a str subclass with overloaded `__eq__` and `__hash__`. Previously it could cause SystemError or other undesired behavior.

Library

- gh-94996 [https://github.com/python/cpython/issues/94996]: **`ast.parse()`** will no longer parse function definitions with positional-only params when passed `feature_version` less than `(3, 8)`. Patch by Shantanu Jain.

Core and Builtins

- gh-94739 [https://github.com/python/cpython/issues/94739]: Allow jumping within, out of, and across exception handlers in the debugger.

Library

- gh-94949 [https://github.com/python/cpython/issues/94949]: **`ast.parse()`** will no longer parse parenthesized context managers when passed `feature_version` less than `(3,` `9)`. Patch by Shantanu Jain.

- gh-94947 [https://github.com/python/cpython/issues/94947]: **`ast.parse()`** will no longer parse assignment expressions when passed `feature_version` less than `(3, 8)`. Patch by Shantanu Jain.

Core and Builtins

- gh-91256 [https://github.com/python/cpython/issues/91256]: Ensures the program name is known for help text during interpreter startup.
- gh-94869 [https://github.com/python/cpython/issues/94869]: Fix the column offsets for some expressions in multi-line f-strings **`ast`** nodes. Patch by Pablo Galindo.
- gh-94893 [https://github.com/python/cpython/issues/94893]: Fix an issue where frame object manipulations could corrupt inline bytecode caches.
- gh-94822 [https://github.com/python/cpython/issues/94822]: Fix an issue where lookups of metaclass descriptors may be ignored when an identically-named attribute also exists on the class itself.
- gh-91153 [https://github.com/python/cpython/issues/91153]: Fix an issue where a **`bytearray`** item assignment could crash if it's resized by the new value's **`__index__()`** method.
- gh-90699 [https://github.com/python/cpython/issues/90699]: Fix reference counting bug in **`bool.__repr__()`**. Patch by Kumar Aditya.
- gh-94694 [https://github.com/python/cpython/issues/94694]: Fix an issue that could cause code with multi-line method lookups to have misleading or incorrect column offset information. In some cases (when compiling a hand-built AST) this could have resulted in a hard crash of the interpreter.
- gh-93252 [https://github.com/python/cpython/issues/93252]: Fix an issue that caused internal frames to outlive failed Python function calls, possibly resulting in memory leaks or hard interpreter crashes.
- gh-94215 [https://github.com/python/cpython/issues/94215]: Fix an issue where exceptions raised by line-tracing events would cause frames to be left in an invalid state, possibly resulting in a hard crash of the interpreter.

- gh-92228 [https://github.com/python/cpython/issues/92228]: Disable the compiler's inline-small-exit-blocks optimization for exit blocks that are associated with source code lines. This fixes a bug where the debugger cannot tell where an exception handler ends and the following code block begins.
- gh-94485 [https://github.com/python/cpython/issues/94485]: Line number of a module's `RESUME` instruction is set to 0 as specified in **PEP 626** [https://peps.python.org/pep-0626/].
- gh-94438 [https://github.com/python/cpython/issues/94438]: Account for instructions that can push NULL to the stack when setting line number in a frame. Prevents some (unlikely) crashes.
- gh-91719 [https://github.com/python/cpython/issues/91719]: Reload `opcode` when raising `unknown opcode error` in the interpreter main loop, for C compilers to generate dispatching code independently.
- gh-94329 [https://github.com/python/cpython/issues/94329]: Compile and run code with unpacking of extremely large sequences (1000s of elements). Such code failed to compile. It now compiles and runs correctly.
- gh-94360 [https://github.com/python/cpython/issues/94360]: Fixed a tokenizer crash when reading encoded files with syntax errors from `stdin` with non utf-8 encoded text. Patch by Pablo Galindo
- gh-88116 [https://github.com/python/cpython/issues/88116]: Fix an issue when reading line numbers from code objects if the encoded line numbers are close to `INT_MIN`. Patch by Pablo Galindo
- gh-94262 [https://github.com/python/cpython/issues/94262]: Don't create frame objects for incomplete frames. Prevents the creation of generators and closures from being observable to Python and C extensions, restoring the behavior of 3.10 and earlier.
- gh-94192 [https://github.com/python/cpython/issues/94192]: Fix error for dictionary literals with invalid expression as value.
- gh-87995 [https://github.com/python/cpython/issues/87995]: **`types.MappingProxyType`** instances are now hashable if the underlying mapping is hashable.
- gh-93883 [https://github.com/python/cpython/issues/93883]: Revise the display strategy of traceback enhanced error locations. The indicators are only shown when the location doesn't span

the whole line.

- gh-94163 [https://github.com/python/cpython/issues/94163]: Add **`BINARY_SLICE`** and **`STORE_SLICE`** instructions for more efficient handling and better specialization of slicing operations, where the slice is explicit in the source code.
- gh-94021 [https://github.com/python/cpython/issues/94021]: Fix unreachable code warning in `Python/specialize.c`.
- gh-93911 [https://github.com/python/cpython/issues/93911]: Specialize `LOAD_ATTR` for objects with custom `__getattribute__`.
- gh-93955 [https://github.com/python/cpython/issues/93955]: Improve performance of attribute lookups on objects with custom `__getattribute__` and `__getattr__`. Patch by Ken Jin.
- gh-93911 [https://github.com/python/cpython/issues/93911]: Specialize `LOAD_ATTR` for `property()` attributes.
- gh-93678 [https://github.com/python/cpython/issues/93678]: Refactor compiler optimisation code so that it no longer needs the `struct assembler` and `struct compiler` passed around. Instead, each function takes the CFG and other data that it actually needs. This will make it possible to test this code directly.
- gh-93841 [https://github.com/python/cpython/issues/93841]: When built with `-enable-pystats`, `sys._stats_on()`, `sys._stats_off()`, `sys._stats_clear()` and `sys._stats_dump()` functions have been added to enable gathering stats for parts of programs.
- gh-93516 [https://github.com/python/cpython/issues/93516]: Store offset of first traceable instruction in code object to avoid having to recompute it for each instruction when tracing.
- gh-93516 [https://github.com/python/cpython/issues/93516]: Lazily create a table mapping bytecode offsets to line numbers to speed up calculation of line numbers when tracing.
- gh-89828 [https://github.com/python/cpython/issues/89828]: **`types.GenericAlias`** no longer relays the `__class__` attribute. For example, `isinstance(list[int], type)` no longer returns `True`.
- gh-93678 [https://github.com/python/cpython/issues/93678]: Refactor the compiler to reduce boilerplate and repetition.
- gh-93671 [https://github.com/python/cpython/issues/93671]: Fix some exponential backtrace case happening with deeply

nested sequence patterns in match statements. Patch by Pablo Galindo

- gh-93662 [https://github.com/python/cpython/issues/93662]: Make sure that the end column offsets are correct in multi-line method calls. Previously, the end column could precede the column offset.

Library

- gh-93461 [https://github.com/python/cpython/issues/93461]: **`importlib.invalidate_caches()`** now drops entries from **`sys.path_importer_cache`** with a relative path as name. This solves a caching issue when a process changes its current working directory.

  `FileFinder` no longer inserts a dot in the path, e.g. `/egg/./spam` is now `/egg/spam`.

Core and Builtins

- gh-93621 [https://github.com/python/cpython/issues/93621]: Change order of bytecode instructions emitted for **`with`** and **`async with`** to reduce the number of entries in the exception table.

- gh-93533 [https://github.com/python/cpython/issues/93533]: Reduce the size of the inline cache for `LOAD_METHOD` by 2 bytes.

- gh-93444 [https://github.com/python/cpython/issues/93444]: Removed redundant fields from the compiler's basicblock struct: `b_nofallthrough`, `b_exit`, `b_return`. They can be easily calculated from the opcode of the last instruction of the block.

- gh-93429 [https://github.com/python/cpython/issues/93429]: `LOAD_METHOD` instruction has been removed. It was merged back into `LOAD_ATTR`.

- gh-93418 [https://github.com/python/cpython/issues/93418]: Fixed an assert where an f-string has an equal sign "=" following an expression, but there's no trailing brace. For example, f»{i=».

- gh-93382 [https://github.com/python/cpython/issues/93382]: Cache the result of `PyCode_GetCode()` function to restore the $O(1)$ lookup of the **`co_code`** attribute.

- gh-93359 [https://github.com/python/cpython/issues/93359]: Ensure that custom `ast` nodes without explicit end positions can be compiled. Patch by Pablo Galindo.

- gh-93356 [https://github.com/python/cpython/issues/93356]: Code for exception handlers is emitted at the end of the code unit's bytecode. This avoids one jump when no exception is raised.

- gh-93354 [https://github.com/python/cpython/issues/93354]: Use exponential backoff for specialization counters in the interpreter. Can reduce the number of failed specializations significantly and avoid slowdown for those parts of a program that are not suitable for specialization.

- gh-93283 [https://github.com/python/cpython/issues/93283]: Improve error message for invalid syntax of conversion character in f-string expressions.

- gh-93345 [https://github.com/python/cpython/issues/93345]: Fix a crash in substitution of a `TypeVar` in nested generic alias after `TypeVarTuple`.

- gh-93223 [https://github.com/python/cpython/issues/93223]: When a bytecode instruction jumps to an unconditional jump instruction, the first instruction can often be optimized to target the unconditional jump's target directly. For tracing reasons, this would previously only occur if both instructions have the same line number. This also now occurs if the unconditional jump is artificial, i.e., if it has no associated line number.

- gh-84694 [https://github.com/python/cpython/issues/84694]: The `--experimental-isolated-subinterpreters` configure option and `EXPERIMENTAL_ISOLATED_SUBINTERPRETERS` macro have been removed.

- gh-91924 [https://github.com/python/cpython/issues/91924]: Fix

`__lltrace__` debug feature if the stdout encoding is not UTF-8. Patch by Victor Stinner.

- gh-93040 [https://github.com/python/cpython/issues/93040]: Wraps unused parameters in `Objects/obmalloc.c` with `Py_UNUSED`.

- gh-93143 [https://github.com/python/cpython/issues/93143]: Avoid `NULL` checks for uninitialized local variables by determining at compile time which variables must be initialized.

- gh-93061 [https://github.com/python/cpython/issues/93061]: Backward jumps after `async for` loops are no longer given dubious line numbers.

- gh-93065 [https://github.com/python/cpython/issues/93065]: Fix contextvars HAMT implementation to handle iteration over deep trees.

  The bug was discovered and fixed by Eli Libman. See MagicStack/immutables#84 [https://github.com/MagicStack/immutables/issues/84] for more details.

- gh-93012 [https://github.com/python/cpython/issues/93012]: Added the new function **`PyType_FromMetaclass()`**, which generalizes the existing **`PyType_FromModuleAndSpec()`** using an additional metaclass argument. This is useful for language binding tools, where it can be used to intercept type-related operations like subclassing or static attribute access by specifying a metaclass with custom slots.

  Importantly, **`PyType_FromMetaclass()`** is available in the Limited API, which provides a path towards migrating more binding tools onto the Stable ABI.

- gh-93021 [https://github.com/python/cpython/issues/93021]: Fix the **`__text_signature__`** for **`__get__()`** methods implemented in C. Patch by Jelle Zijlstra.

- gh-89914 [https://github.com/python/cpython/issues/89914]: The operand of the `YIELD_VALUE` instruction is set to the stack depth. This is done to help frame handling on `yield` and may assist debuggers.

- gh-92955 [https://github.com/python/cpython/issues/92955]: Fix memory leak in code object’s lines and positions iterators as they were not finalized at exit. Patch by Kumar Aditya.
- gh-92930 [https://github.com/python/cpython/issues/92930]: Fixed a crash in `_pickle.c` from mutating collections during `__reduce__` or `persistent_id`.
- gh-90690 [https://github.com/python/cpython/issues/90690]: The PRECALL instruction has been removed. It offered only a small advantage for specialization and is not needed in the vast majority of cases.
- gh-92914 [https://github.com/python/cpython/issues/92914]: Always round the allocated size for lists up to the nearest even number.
- gh-92858 [https://github.com/python/cpython/issues/92858]: Improve error message for some suites with syntax error before “:”
- gh-90473 [https://github.com/python/cpython/issues/90473]: Decrease default recursion limit on WASI to address limited call stack size.
- gh-92804 [https://github.com/python/cpython/issues/92804]: Fix memory leak in `memoryview` iterator as it was not finalized at exit. Patch by Kumar Aditya.
- gh-92777 [https://github.com/python/cpython/issues/92777]: Specialize `LOAD_METHOD` for objects with lazy dictionaries. Patch by Ken Jin.
- gh-92658 [https://github.com/python/cpython/issues/92658]: Add support for connecting and binding to Hyper-V sockets on Windows Hyper-V hosts and guests.
- gh-92236 [https://github.com/python/cpython/issues/92236]: Remove spurious «LINE» event when starting a generator or coroutine, visible tracing functions implemented in C.

Library

- gh-91102 [https://github.com/python/cpython/issues/91102]: **`_warnings.warn_explicit()`** is ported to Argument Clinic.

Core and Builtins

- gh-92619 [https://github.com/python/cpython/issues/92619]: Make the compiler duplicate an exit block only if none of its instructions have a lineno (previously only the first instruction in the block was checked, leading to unnecessarily duplicated blocks).
- gh-88750 [https://github.com/python/cpython/issues/88750]: The deprecated debug build only `PYTHONTHREADDEBUG` environment variable no longer does anything.

Library

- gh-92261 [https://github.com/python/cpython/issues/92261]: Fix hang when trying to iterate over a `typing.Union`.

Core and Builtins

- gh-91432 [https://github.com/python/cpython/issues/91432]: Specialized the **`FOR_ITER`** opcode using the PEP 659 machinery
- gh-91399 [https://github.com/python/cpython/issues/91399]: Removed duplicate “{0, 0, 0, 0, 0, 0}” entry in “Objects/unicodetype_db.h”.
- gh-91578 [https://github.com/python/cpython/issues/91578]: Updates the error message for abstract class.
- bpo-47091 [https://bugs.python.org/issue?@action=redirect&bpo=47091]: Improve performance of repetition of **`list`** and **`tuple`** by using `memcpy` to copy data and performing the reference increments in one step.
- bpo-46142 [https://bugs.python.org/issue?@action=redirect&bpo=46142]: Make `--help` output shorter by moving some info to the new `--help-env` and `--help-`

`xoptions` command-line options. Also add `--help-all` option to print complete usage.
- bpo-42316 [https://bugs.python.org/issue?@action=redirect&bpo=42316]: Document some places where an assignment expression needs parentheses.

Library

- gh-89237 [https://github.com/python/cpython/issues/89237]: Fix hang on Windows in `subprocess.wait_closed()` in **`asyncio`** with **`ProactorEventLoop`**. Patch by Kumar Aditya.

- gh-97928 [https://github.com/python/cpython/issues/97928]: **`tkinter.Text.count()`** raises now an exception for options starting with «-» instead of silently ignoring them.

- gh-98393 [https://github.com/python/cpython/issues/98393]: The **`os`** module no longer accepts bytes-like paths, like **`bytearray`** and **`memoryview`** types: only the exact **`bytes`** type is accepted for bytes strings. Patch by Victor Stinner.

- gh-98363 [https://github.com/python/cpython/issues/98363]: Added itertools.batched() to batch data into lists of a given length with the last list possibly being shorter than the others.

- gh-98331 [https://github.com/python/cpython/issues/98331]: Update the bundled copies of pip and setuptools to versions 22.3 and 65.5.0 respectively.

- gh-98307 [https://github.com/python/cpython/issues/98307]: A **`createSocket()`** method was added to **`SysLogHandler`**.

- gh-96035 [https://github.com/python/cpython/issues/96035]: Fix bug in **`urllib.parse.urlparse()`** that causes certain port numbers containing whitespace, underscores, plus and minus signs, or non-ASCII digits to be incorrectly accepted.

- gh-98257 [https://github.com/python/cpython/issues/98257]: Make **`sys.setprofile()`** and **`sys.settrace()`** functions reentrant. They can no long fail with: `RuntimeError("Cannot install a trace function` `while another trace function is being installed")`. Patch by Victor Stinner.

- gh-98251 [https://github.com/python/cpython/issues/98251]: Allow **`venv`** to pass along **`PYTHON*`** variables to `ensurepip` and `pip` when they do not impact path resolution

- gh-94597 [https://github.com/python/cpython/issues/94597]: Deprecated **`asyncio.AbstractEventLoopPolicy.get_child_watcher()`** and **`asyncio.AbstractEventLoopPolicy.set_child_watcher()`** methods to be removed in Python 3.14. Patch by Kumar Aditya.

- gh-98178 [https://github.com/python/cpython/issues/98178]: On macOS, fix a crash in **`syslog.syslog()`** in multi-threaded applications. On macOS, the libc `syslog()` function is not thread-safe, so **`syslog.syslog()`** no longer releases the GIL to call it. Patch by Victor Stinner.

- gh-44098 [https://github.com/python/cpython/issues/44098]: Release the GIL when creating **`mmap.mmap`** objects on Unix.

- gh-87730 [https://github.com/python/cpython/issues/87730]: Wrap network errors consistently in urllib FTP support, so the test suite doesn't fail when a network is available but the public internet is not reachable.

- gh-94597 [https://github.com/python/cpython/issues/94597]: The child watcher classes **`MultiLoopChildWatcher`**, **`FastChildWatcher`** and **`SafeChildWatcher`** are deprecated and will be removed in Python 3.14. Patch by Kumar Aditya.

- gh-98023 [https://github.com/python/cpython/issues/98023]: Change default child watcher to **`PidfdChildWatcher`** on Linux systems which supports it. Patch by Kumar Aditya.

- gh-90985 [https://github.com/python/cpython/issues/90985]: Earlier in 3.11 we deprecated `asyncio.Task.cancel("message")`. We realized we were too harsh, and have undeprecated it.

- gh-65961 [https://github.com/python/cpython/issues/65961]: Do not rely solely on `__cached__` on modules; code will also support `__spec__.cached`.

- gh-97646 [https://github.com/python/cpython/issues/97646]: Replace deprecated `application/javascript` with `text/javascript` in **`mimetypes`**. See **RFC 9239** [https://datatracker.ietf.org/doc/html/rfc9239.html]. Patch by Noam Cohen.

- gh-97930 [https://github.com/python/cpython/issues/97930]: Apply changes from importlib_resources 5.8 and 5.9: `Traversable.joinpath` provides a concrete implementation. `as_file` now supports directories of resources.

- gh-97850 [https://github.com/python/cpython/issues/97850]: Remove deprecated **`importlib.util.set_loader()`** and **`importlib.util.module_for_loader()`** from **`importlib.util`**.

- gh-97837 [https://github.com/python/cpython/issues/97837]: Change deprecate warning message in **`unittest`** from

  ```
  It is deprecated to return a value!=None
  ```

  to

  ```
  It is deprecated to return a value that is not
  None from a test case
  ```

- gh-97825 [https://github.com/python/cpython/issues/97825]: Fixes **`AttributeError`** when **`subprocess.check_output()`** is used with argument `input=None` and either of the arguments *encoding* or *errors* are used.

- gh-97008 [https://github.com/python/cpython/issues/97008]: **`NameError`** and **`AttributeError`** spelling suggestions provided since gh-82711 [https://github.com/python/cpython/issues/82711] are now also emitted by the pure Python `traceback` module. Tests for those suggestions now exercise both implementations to ensure they are equivalent. Patch by Carl Friedrich Bolz-Tereick and Łukasz Langa.

- gh-97799 [https://github.com/python/cpython/issues/97799]: **`dataclass`** now uses `inspect.get_annotations()` to examine the annotations on class objects.

- gh-97781 [https://github.com/python/cpython/issues/97781]: Removed deprecated interfaces in `importlib.metadata` (entry points accessed as dictionary, implicit dictionary construction of sequence of `EntryPoint` objects, mutablility of `EntryPoints` result, access of entry point by index). `entry_points` now has a simpler, more straightforward API (returning `EntryPoints`).

- gh-96827 [https://github.com/python/cpython/issues/96827]: Avoid spurious tracebacks from `asyncio` when default executor cleanup is delayed until after the event loop is closed (e.g. as the result of a keyboard interrupt).

- gh-95534 [https://github.com/python/cpython/issues/95534]: **`gzip.GzipFile.read()`** reads 10% faster.

- gh-97592 [https://github.com/python/cpython/issues/97592]: Avoid a crash in the C version of `asyncio.Future.remove_done_callback()` when an evil argument is passed.

- gh-97639 [https://github.com/python/cpython/issues/97639]: Remove `tokenize.NL` check from `tabnanny`.

- gh-97545 [https://github.com/python/cpython/issues/97545]: Make Semaphore run faster.

- gh-73588 [https://github.com/python/cpython/issues/73588]: Fix generation of the default name of **`tkinter.Checkbutton`**. Previously, checkbuttons in different parent widgets could have the same short name and share the same state if arguments «name» and «variable» are not specified. Now they are globally unique.

- gh-96865 [https://github.com/python/cpython/issues/96865]: fix Flag to use boundary CONFORM

  This restores previous Flag behavior of allowing flags with non-sequential values to be combined; e.g.

  class Skip(Flag): TWO = 2 EIGHT = 8

  Skip.TWO | Skip.EIGHT -> <Skip.TWO|EIGHT: 10>

- gh-97005 [https://github.com/python/cpython/issues/97005]: Update bundled libexpat to 2.4.9

- gh-85760 [https://github.com/python/cpython/issues/85760]: Fix race condition in **`asyncio`** where **`process_exited()`** called before the **`pipe_data_received()`** leading to inconsistent output. Patch by Kumar Aditya.

- gh-96704 [https://github.com/python/cpython/issues/96704]: Pass the correct `contextvars.Context` when a `asyncio` exception handler is called on behalf of a task or callback handle. This adds a new `Task` method, `get_context`, and also a new `Handle` method with the same name. If this method is not found on a task object (perhaps because it is a third-party library that does not yet provide this method), the context prevailing at the time the exception handler is called is used.

- gh-96819 [https://github.com/python/cpython/issues/96819]: Fixed check in **`multiprocessing.resource_tracker`** that guarantees that the length of a write to a pipe is not greater than `PIPE_BUF`.

- gh-95865 [https://github.com/python/cpython/issues/95865]: Reduce **`urllib.parse.quote_from_bytes()`** memory use on large values.

  Contributed by Dennis Sweeney.

- gh-96741 [https://github.com/python/cpython/issues/96741]: Corrected type annotation for dataclass attribute `pstats.FunctionProfile.ncalls` to be `str`.

- gh-96734 [https://github.com/python/cpython/issues/96734]: Update **`unicodedata`** database to Unicode 15.0.0.

- gh-96735 [https://github.com/python/cpython/issues/96735]: Fix undefined behaviour in **`struct.unpack()`**.

- gh-46412 [https://github.com/python/cpython/issues/46412]: Improve performance of `bool(db)` for large ndb/gdb databases. Previously this would call `len(db)` which would iterate over all keys – the answer (empty or not) is known after the first key.

- gh-96652 [https://github.com/python/cpython/issues/96652]: Fix the faulthandler implementation of `faulthandler.register(signal, chain=True)` if the `sigaction()` function is not available: don't call the previous signal handler if it's NULL. Patch by Victor Stinner.

- gh-68163 [https://github.com/python/cpython/issues/68163]: Correct conversion of **`numbers.Rational`**'s to **`float`**.

- gh-96538 [https://github.com/python/cpython/issues/96538]: Speed up `bisect.bisect()` functions by taking advantage of type-stability.

- gh-96465 [https://github.com/python/cpython/issues/96465]: Fraction hashes are now cached.

- gh-96079 [https://github.com/python/cpython/issues/96079]: In **`typing`**, fix missing field `name` and incorrect `__module__` in _AnnotatedAlias.

- gh-96415 [https://github.com/python/cpython/issues/96415]: Remove `types._cell_factory` from module namespace.

- gh-95987 [https://github.com/python/cpython/issues/95987]: Fix `repr` of `Any` subclasses.

- gh-96388 [https://github.com/python/cpython/issues/96388]: Work around missing socket functions in **`socket`**'s `__repr__`.

- gh-96385 [https://github.com/python/cpython/issues/96385]: Fix

`TypeVarTuple.__typing_prepare_subst__`. `TypeError` was not raised when using more than one `TypeVarTuple`, like `[*T, *V]` in type alias substitutions.

- gh-96142 [https://github.com/python/cpython/issues/96142]: Add `match_args`, `kw_only`, `slots`, and `weakref_slot` to `_DataclassParams`.
- gh-96073 [https://github.com/python/cpython/issues/96073]: In **`inspect`**, fix overeager replacement of «`typing.`» in formatting annotations.
- gh-89258 [https://github.com/python/cpython/issues/89258]: Added a **`getChildren()`** method to **`logging.Logger`**, to get the immediate child loggers of a logger.
- gh-96346 [https://github.com/python/cpython/issues/96346]: Use double caching for compiled RE patterns.
- gh-96349 [https://github.com/python/cpython/issues/96349]: Fixed a minor performance regression in **`threading.Event.__init__()`**
- gh-90467 [https://github.com/python/cpython/issues/90467]: Fix **`asyncio.streams.StreamReaderProtocol`** to keep a strong reference to the created task, so that it's not garbage collected
- gh-96172 [https://github.com/python/cpython/issues/96172]: Fix a bug in `unicodedata: east_asian_width` used to return the wrong value for unassigned characters; and for yet unassigned, but reserved characters.
- gh-96159 [https://github.com/python/cpython/issues/96159]: Fix a performance regression in logging TimedRotatingFileHandler. Only check for special files when the rollover time has passed.
- gh-96175 [https://github.com/python/cpython/issues/96175]: Fix unused `localName` parameter in the `Attr` class in **`xml.dom.minidom`**.

- gh-96145 [https://github.com/python/cpython/issues/96145]: Add AttrDict to JSON module for use with object_hook.

- gh-96052 [https://github.com/python/cpython/issues/96052]: Fix handling compiler warnings (SyntaxWarning and DeprecationWarning) in **`codeop.compile_command()`** when checking for incomplete input. Previously it emitted warnings and raised a SyntaxError. Now it always returns `None` for incomplete input without emitting any warnings.

- gh-96125 [https://github.com/python/cpython/issues/96125]: Fix incorrect condition that causes `sys.thread_info.name` to be wrong on pthread platforms.

- gh-96019 [https://github.com/python/cpython/issues/96019]: Fix a bug in the `makeunicodedata.py` script leading to about 13 KiB of space saving in the `unicodedata` module, specifically the character decomposition data.

- gh-95463 [https://github.com/python/cpython/issues/95463]: Remove an incompatible change from bpo-28080 [https://bugs.python.org/issue?@action=redirect&bpo=28080] that caused a regression that ignored the utf8 in `ZipInfo.flag_bits`. Patch by Pablo Galindo.

- gh-69142 [https://github.com/python/cpython/issues/69142]: Add `%:z` strftime format code (generates tzoffset with colons as separator), see strftime() and strptime() behavior.

- gh-95899 [https://github.com/python/cpython/issues/95899]: Fix **`asyncio.Runner`** to call **`asyncio.set_event_loop()`** only once to avoid calling **`attach_loop()`** multiple times on child watchers. Patch by Kumar Aditya.

- gh-95736 [https://github.com/python/cpython/issues/95736]: Fix **`unittest.IsolatedAsyncioTestCase`** to set event loop before calling setup functions. Patch by Kumar Aditya.

- gh-95865 [https://github.com/python/cpython/issues/95865]: Speed up **`urllib.parse.quote_from_bytes()`** by replacing a list comprehension with `map()`.

- gh-95861 [https://github.com/python/cpython/issues/95861]: Add support for computing Spearman's correlation coefficient to the existing statistics.correlation() function.

- gh-95804 [https://github.com/python/cpython/issues/95804]: Fix `logging` shutdown handler so it respects `MemoryHandler.flushOnClose`.

- gh-95704 [https://github.com/python/cpython/issues/95704]: When a task catches **`asyncio.CancelledError`** and raises some other error, the other error should generally not silently be suppressed.

- gh-95149 [https://github.com/python/cpython/issues/95149]: The **`HTTPStatus`** enum offers a couple of properties to indicate the HTTP status category e.g. `HTTPStatus.OK.is_success`.

- gh-95609 [https://github.com/python/cpython/issues/95609]: Update bundled pip to 22.2.2.

- gh-95289 [https://github.com/python/cpython/issues/95289]: Fix **`asyncio.TaskGroup`** to propagate exception when **`asyncio.CancelledError`** was replaced with another exception by a context manager. Patch by Kumar Aditya and Guido van Rossum.

- gh-94909 [https://github.com/python/cpython/issues/94909]: Fix incorrect joining of relative Windows paths with drives in **`pathlib.PurePath`** initializer.

- gh-95385 [https://github.com/python/cpython/issues/95385]: Faster `json.dumps()` when sorting of keys is not requested (default).

- gh-83901 [https://github.com/python/cpython/issues/83901]: Improve **`Signature.bind`** error message for missing keyword-only arguments.

- gh-95339 [https://github.com/python/cpython/issues/95339]: Update bundled pip to 22.2.1.

- gh-95045 [https://github.com/python/cpython/issues/95045]: Fix GC crash when deallocating `_lsprof.Profiler` by untracking it before calling any callbacks. Patch by Kumar Aditya.

- gh-95231 [https://github.com/python/cpython/issues/95231]: Fail gracefully if **`EPERM`** or **`ENOSYS`** is raised when loading **`crypt`** methods. This may happen when trying to load `MD5` on a Linux kernel with FIPS enabled.

- gh-95097 [https://github.com/python/cpython/issues/95097]: Fix **`asyncio.run()`** for **`asyncio.Task`** implementations without **`uncancel()`** method. Patch by Kumar Aditya.

- gh-95087 [https://github.com/python/cpython/issues/95087]: Fix IndexError in parsing invalid date in the **`email`** module.

- gh-95199 [https://github.com/python/cpython/issues/95199]: Upgrade bundled setuptools to 63.2.0.

- gh-95194 [https://github.com/python/cpython/issues/95194]: Upgrade bundled pip to 22.2.

- gh-93899 [https://github.com/python/cpython/issues/93899]: Fix check for existence of **`os.EFD_CLOEXEC`**, **`os.EFD_NONBLOCK`** and **`os.EFD_SEMAPHORE`** flags on older kernel versions where these flags are not present. Patch by Kumar Aditya.

- gh-95166 [https://github.com/python/cpython/issues/95166]: Fix **`concurrent.futures.Executor.map()`** to cancel the currently waiting on future on an error - e.g. TimeoutError or KeyboardInterrupt.

- gh-95132 [https://github.com/python/cpython/issues/95132]: Fix a **`sqlite3`** regression where `*args` and `**kwds` were incorrectly relayed from **`connect()`** to the **`Connection`** factory. The regression was introduced in 3.11a1 with PR 24421 (gh-85128 [https://github.com/python/cpython/issues/85128]). Patch by Erlend E. Aasland.

- gh-93157 [https://github.com/python/cpython/issues/93157]: Fix

`fileinput` module didn't support `errors` option when `inplace` is true.

- gh-91212 [https://github.com/python/cpython/issues/91212]: Fixed flickering of the turtle window when the tracer is turned off. Patch by Shin-myoung-serp.

- gh-95077 [https://github.com/python/cpython/issues/95077]: Add deprecation warning for enum `member.member` access (e.g. `Color.RED.BLUE`). Remove `EnumMeta.__getattr__`.

- gh-95109 [https://github.com/python/cpython/issues/95109]: Ensure that timeouts scheduled with **`asyncio.Timeout`** that have already expired are delivered promptly.

- gh-95105 [https://github.com/python/cpython/issues/95105]: **`wsgiref.types.InputStream.__iter__()`** should return `Iterator[bytes]`, not `Iterable[bytes]`. Patch by Shantanu Jain.

- gh-95066 [https://github.com/python/cpython/issues/95066]: Replaced assert with exception in **`ast.parse()`**, when `feature_version` has an invalid major version. Patch by Shantanu Jain.

- gh-77617 [https://github.com/python/cpython/issues/77617]: Add **`sqlite3`** command-line interface. Patch by Erlend Aasland.

- gh-95005 [https://github.com/python/cpython/issues/95005]: Replace _PyAccu with _PyUnicodeWriter in JSON encoder and StringIO and remove the _PyAccu implementation.

- gh-90085 [https://github.com/python/cpython/issues/90085]: Remove `-c/--clock` and `-t/--time` CLI options of **`timeit`**. The options had been deprecated since Python 3.3 and the functionality was removed in Python 3.7. Patch by Shantanu Jain.

- gh-94857 [https://github.com/python/cpython/issues/94857]: Fix refleak in `_io.TextIOWrapper.reconfigure`. Patch by Kumar Aditya.

- gh-94821 [https://github.com/python/cpython/issues/94821]: Fix binding of unix socket to empty address on Linux to use an available address from the abstract namespace, instead of «0».

- gh-94736 [https://github.com/python/cpython/issues/94736]: Fix crash when deallocating an instance of a subclass of `_multiprocessing.SemLock`. Patch by Kumar Aditya.

- gh-81620 [https://github.com/python/cpython/issues/81620]: Add random.binomialvariate().

- gh-74116 [https://github.com/python/cpython/issues/74116]: Allow **`asyncio.StreamWriter.drain()`** to be awaited concurrently by multiple tasks. Patch by Kumar Aditya.

- gh-87822 [https://github.com/python/cpython/issues/87822]: When called with `capture_locals=True`, the **`traceback`** module functions swallow exceptions raised from calls to `repr()` on local variables of frames. This is in order to prioritize the original exception over rendering errors. An indication of the failure is printed in place of the missing value. (Patch by Simon-Martin Schroeder).

- gh-88050 [https://github.com/python/cpython/issues/88050]: Fix **`asyncio`** subprocess transport to kill process cleanly when process is blocked and avoid `RuntimeError` when loop is closed. Patch by Kumar Aditya.

- gh-94637 [https://github.com/python/cpython/issues/94637]: **`SSLContext.set_default_verify_paths()`** now releases the GIL around `SSL_CTX_set_default_verify_paths` call. The function call performs I/O and CPU intensive work.

- gh-94309 [https://github.com/python/cpython/issues/94309]: Deprecate aliases **`typing.Hashable`** and **`typing.Sized`**

- gh-92546 [https://github.com/python/cpython/issues/92546]: An undocumented `python -m pprint` benchmark is moved into `pprint` suite of pyperformance. Patch by Oleg Iarygin.

- gh-94607 [https://github.com/python/cpython/issues/94607]: Fix subclassing complex generics with type variables in **`typing`**. Previously an error message saying `Some type variables ... are not listed in Generic[...]` was shown. **`typing`** no longer populates `__parameters__` with the `__parameters__` of a Python class.

- gh-94619 [https://github.com/python/cpython/issues/94619]: Remove the long-deprecated `module_repr()` from **`importlib`**.

- gh-93910 [https://github.com/python/cpython/issues/93910]: The ability to access the other values of an enum on an enum (e.g. `Color.RED.BLUE`) has been restored in order to fix a performance regression.

- gh-93896 [https://github.com/python/cpython/issues/93896]: Fix **`asyncio.run()`** and **`unittest.IsolatedAsyncioTestCase`** to always the set event loop as it was done in Python 3.10 and earlier. Patch by Kumar Aditya.

- gh-94343 [https://github.com/python/cpython/issues/94343]: Allow setting the attributes of `reprlib.Repr` during object initialization

- gh-94382 [https://github.com/python/cpython/issues/94382]: Port static types of `_multiprocessing` module to heap types. Patch by Kumar Aditya.

- gh-78724 [https://github.com/python/cpython/issues/78724]: Fix crash in **`struct.Struct`** when it was not completely initialized by initializing it in **`__new__()`**. Patch by Kumar Aditya.

- gh-94510 [https://github.com/python/cpython/issues/94510]: Re-entrant calls to **`sys.setprofile()`** and **`sys.settrace()`** now raise **`RuntimeError`**. Patch by Pablo Galindo.

- gh-92336 [https://github.com/python/cpython/issues/92336]: Fix bug where **`linecache.getline()`** fails on bad files with

`UnicodeDecodeError` or `SyntaxError`. It now returns an empty string as per the documentation.

- gh-94398 [https://github.com/python/cpython/issues/94398]: Once a `asyncio.TaskGroup` has started shutting down (i.e., at least one task has failed and the task group has started cancelling the remaining tasks), it should not be possible to add new tasks to the task group.

- gh-94383 [https://github.com/python/cpython/issues/94383]: **`xml.etree`**: Remove the `ElementTree.Element.copy()` method of the pure Python implementation, deprecated in Python 3.10, use the `copy.copy()` function instead. The C implementation of **`xml.etree`** has no `copy()` method, only a `__copy__()` method. Patch by Victor Stinner.

- gh-94379 [https://github.com/python/cpython/issues/94379]: `zipimport`: Remove `find_loader()` and `find_module()` methods, deprecated in Python 3.10: use the `find_spec()` method instead. See **PEP 451** [https://peps.python.org/pep-0451/] for the rationale. Patch by Victor Stinner.

- gh-94352 [https://github.com/python/cpython/issues/94352]: `shlex.split()`: Passing `None` for *s* argument now raises an exception, rather than reading `sys.stdin`. The feature was deprecated in Python 3.9. Patch by Victor Stinner.

- gh-94318 [https://github.com/python/cpython/issues/94318]: Strip trailing spaces in `pydoc` text output.

- gh-89988 [https://github.com/python/cpython/issues/89988]: Fix memory leak in `pickle.Pickler` when looking up **`dispatch_table`**. Patch by Kumar Aditya.

- gh-90016 [https://github.com/python/cpython/issues/90016]: Deprecate `sqlite3` default adapters and converters. Patch by Erlend E. Aasland.

- gh-94254 [https://github.com/python/cpython/issues/94254]: Fixed types of `struct` module to be immutable. Patch by Kumar

Aditya.

- gh-93259 [https://github.com/python/cpython/issues/93259]: Now raise `ValueError` when `None` or an empty string are passed to `Distribution.from_name` (and other callers).

- gh-74696 [https://github.com/python/cpython/issues/74696]: `shutil.make_archive()` now passes the *root_dir* argument to custom archivers which support it.

- gh-94216 [https://github.com/python/cpython/issues/94216]: The `dis` module now has the opcodes for pseudo instructions (those which are used by the compiler during code generation but then removed or replaced by real opcodes before the final bytecode is emitted).

- gh-93096 [https://github.com/python/cpython/issues/93096]: Removed undocumented `python -m codecs`. Use `python -m unittest test.test_codecs.EncodedFileTest` instead.

- gh-94207 [https://github.com/python/cpython/issues/94207]: Made `_struct.Struct` GC-tracked in order to fix a reference leak in the `_struct` module.

- gh-93096 [https://github.com/python/cpython/issues/93096]: Removed undocumented `-t` argument of `python -m base64`. Use `python -m unittest test.test_base64.LegacyBase64TestCase.test_encodeby` instead.

- gh-94226 [https://github.com/python/cpython/issues/94226]: Remove the `locale.format()` function, deprecated in Python 3.7: use `locale.format_string()` instead. Patch by Victor Stinner.

- gh-94199 [https://github.com/python/cpython/issues/94199]: Remove the `ssl.match_hostname()` function. The `ssl.match_hostname()` was deprecated in Python 3.7. OpenSSL performs hostname matching since Python 3.7, Python no longer uses the `ssl.match_hostname()`

function. Patch by Victor Stinner.

- gh-94214 [https://github.com/python/cpython/issues/94214]: Document the `context` object used in the `venv.EnvBuilder` class, and add the new environment's library path to it.

- gh-94199 [https://github.com/python/cpython/issues/94199]: Remove the **`ssl.wrap_socket()`** function, deprecated in Python 3.7: instead, create a **`ssl.SSLContext`** object and call its **`ssl.SSLContext.wrap_socket`** method. Any package that still uses **`ssl.wrap_socket()`** is broken and insecure. The function neither sends a SNI TLS extension nor validates server hostname. Code is subject to **CWE 295** [https://cwe.mitre.org/data/definitions/295.html] Improper Certificate Validation. Patch by Victor Stinner.

- gh-94199 [https://github.com/python/cpython/issues/94199]: Remove the **`ssl.RAND_pseudo_bytes()`** function, deprecated in Python 3.6: use **`os.urandom()`** or **`ssl.RAND_bytes()`** instead. Patch by Victor Stinner.

- gh-94199 [https://github.com/python/cpython/issues/94199]: **`hashlib`**: Remove the pure Python implementation of **`hashlib.pbkdf2_hmac()`**, deprecated in Python 3.10. Python 3.10 and newer requires OpenSSL 1.1.1 (**PEP 644** [https://peps.python.org/pep-0644/]): this OpenSSL version provides a C implementation of **`pbkdf2_hmac()`** which is faster. Patch by Victor Stinner.

- gh-94196 [https://github.com/python/cpython/issues/94196]: **`gzip`**: Remove the `filename` attribute of **`gzip.GzipFile`**, deprecated since Python 2.6, use the **`name`** attribute instead. In write mode, the `filename` attribute added `'.gz'` file extension if it was not present. Patch by Victor Stinner.

- gh-94182 [https://github.com/python/cpython/issues/94182]: run the **`asyncio.PidfdChildWatcher`** on the running loop, this allows event loops to run subprocesses when there is no default event loop running on the main thread

- gh-94169 [https://github.com/python/cpython/issues/94169]: Remove `io.OpenWrapper` and `_pyio.OpenWrapper`, deprecated in Python 3.10: just use **`open()`** instead. The **`open()`** (**`io.open()`**) function is a built-in function. Since Python 3.10, **`_pyio.open()`** is also a static method. Patch by Victor Stinner.

- gh-91742 [https://github.com/python/cpython/issues/91742]: Fix **`pdb`** crash after jump caused by a null pointer dereference. Patch by Kumar Aditya.

- gh-94101 [https://github.com/python/cpython/issues/94101]: Manual instantiation of **`ssl.SSLSession`** objects is no longer allowed as it lead to misconfigured instances that crashed the interpreter when attributes where accessed on them.

- gh-84753 [https://github.com/python/cpython/issues/84753]: **`inspect.iscoroutinefunction()`**, **`inspect.isgeneratorfunction()`**, and **`inspect.isasyncgenfunction()`** now properly return `True` for duck-typed function-like objects like instances of **`unittest.mock.AsyncMock`**.

  This makes **`inspect.iscoroutinefunction()`** consistent with the behavior of **`asyncio.iscoroutinefunction()`**. Patch by Mehdi ABAAKOUK.

- gh-94028 [https://github.com/python/cpython/issues/94028]: Fix a regression in the **`sqlite3`** where statement objects were not properly cleared and reset after use in cursor iters. The regression was introduced by PR 27884 in Python 3.11a1. Patch by Erlend E. Aasland.

- gh-93973 [https://github.com/python/cpython/issues/93973]: Add keyword argument `all_errors` to `asyncio.create_connection` so that multiple connection errors can be raised as an `ExceptionGroup`.

- gh-93963 [https://github.com/python/cpython/issues/93963]: Officially deprecate from `importlib.abc` classes moved to `importlib.resources.abc`.

- gh-93858 [https://github.com/python/cpython/issues/93858]: Prevent error when activating venv in nested fish instances.
- gh-93820 [https://github.com/python/cpython/issues/93820]: Pickle **`enum.Flag`** by name.
- gh-93847 [https://github.com/python/cpython/issues/93847]: Fix repr of enum of generic aliases.
- gh-91404 [https://github.com/python/cpython/issues/91404]: Revert the **`re`** memory leak when a match is terminated by a signal or memory allocation failure as the implemented fix caused a major performance regression.
- gh-83499 [https://github.com/python/cpython/issues/83499]: Fix double closing of file description in **`tempfile`**.
- gh-93820 [https://github.com/python/cpython/issues/93820]: Fixed a regression when **`copy.copy()`**-ing **`enum.Flag`** with multiple flag members.
- gh-79512 [https://github.com/python/cpython/issues/79512]: Fixed names and `__module__` value of **`weakref`** classes **`ReferenceType`**, **`ProxyType`**, **`CallableProxyType`**. It makes them pickleable.
- gh-91389 [https://github.com/python/cpython/issues/91389]: Fix an issue where **`dis`** utilities could report missing or incorrect position information in the presence of `CACHE` entries.
- gh-93626 [https://github.com/python/cpython/issues/93626]: Set `__future__.annotations` to have a `None` mandatoryRelease to indicate that it is currently “TBD”.
- gh-90473 [https://github.com/python/cpython/issues/90473]: Emscripten and WASI have no home directory and cannot provide **PEP 370** [https://peps.python.org/pep-0370/] user site directory.
- gh-90494 [https://github.com/python/cpython/issues/90494]:

`copy.copy()` and **`copy.deepcopy()`** now always raise a TypeError if `__reduce__()` returns a tuple with length 6 instead of silently ignore the 6th item or produce incorrect result.

- gh-90549 [https://github.com/python/cpython/issues/90549]: Fix a multiprocessing bug where a global named resource (such as a semaphore) could leak when a child process is spawned (as opposed to forked).
- gh-93521 [https://github.com/python/cpython/issues/93521]: Fixed a case where dataclasses would try to add `__weakref__` into the `__slots__` for a dataclass that specified `weakref_slot=True` when it was already defined in one of its bases. This resulted in a `TypeError` upon the new class being created.
- gh-79579 [https://github.com/python/cpython/issues/79579]: **`sqlite3`** now correctly detects DML queries with leading comments. Patch by Erlend E. Aasland.
- gh-93421 [https://github.com/python/cpython/issues/93421]: Update **`sqlite3.Cursor.rowcount`** when a DML statement has run to completion. This fixes the row count for SQL queries like `UPDATE ... RETURNING`. Patch by Erlend E. Aasland.
- gh-93475 [https://github.com/python/cpython/issues/93475]: Expose `FICLONE` and `FICLONERANGE` constants in **`fcntl`**. Patch by Illia Volochii.
- gh-93370 [https://github.com/python/cpython/issues/93370]: Deprecate **`version`** and **`version_info`**.
- gh-91810 [https://github.com/python/cpython/issues/91810]: Suppress writing an XML declaration in open files in `ElementTree.write()` with `encoding='unicode'` and `xml_declaration=None`.
- gh-91162 [https://github.com/python/cpython/issues/91162]: Support splitting of unpacked arbitrary-length tuple over `TypeVar` and `TypeVarTuple` parameters. For example:

    - `A[T, *Ts][*tuple[int, ...]]` -> `A[int, *tuple[int, ...]]`
    - `A[*Ts, T][*tuple[int, ...]]` -> `A[*tuple[int, ...], int]`

- gh-93353 [https://github.com/python/cpython/issues/93353]: Fix the **`importlib.resources.as_file()`** context manager to remove the temporary file if destroyed late during Python finalization: keep a local reference to the **`os.remove()`** function. Patch by Victor Stinner.

- gh-83658 [https://github.com/python/cpython/issues/83658]: Make **`multiprocessing.Pool`** raise an exception if `maxtasksperchild` is not `None` or a positive int.

- gh-93312 [https://github.com/python/cpython/issues/93312]: Add **`os.PIDFD_NONBLOCK`** flag to open a file descriptor for a process with **`os.pidfd_open()`** in non-blocking mode. Patch by Kumar Aditya.

- gh-88123 [https://github.com/python/cpython/issues/88123]: Implement `Enum.__contains__` that returns `True` or `False` to replace the deprecated behaviour that would sometimes raise a **`TypeError`**.

- gh-93297 [https://github.com/python/cpython/issues/93297]: Make asyncio task groups prevent child tasks from being GCed

- gh-85308 [https://github.com/python/cpython/issues/85308]: Changed **`argparse.ArgumentParser`** to use filesystem encoding and error handler instead of default text encoding to read arguments from file (e.g. `fromfile_prefix_chars` option). This change affects Windows; argument file should be encoded with UTF-8 instead of ANSI Codepage.

- gh-93156 [https://github.com/python/cpython/issues/93156]: Accessing the **`pathlib.PurePath.parents`** sequence of an absolute path using negative index values produced incorrect results.

- gh-93162 [https://github.com/python/cpython/issues/93162]: Add the ability for `logging.config.dictConfig()` to usefully configure `QueueHandler` and `QueueListener` as a pair, and add `logging.getHandlerByName()` and `logging.getHandlerNames()` APIs to allow access to handlers by name.

- gh-93243 [https://github.com/python/cpython/issues/93243]: The **`smtpd`** module was removed per the schedule in PEP 594 [https://peps.python.org/pep-0594/].

- gh-92886 [https://github.com/python/cpython/issues/92886]: Replace `assert` statements with `raise AssertionError()` in **`BaseHandler`** so that the tested behaviour is maintained running with optimizations `(-O)`.

- gh-90155 [https://github.com/python/cpython/issues/90155]: Fix broken `asyncio.Semaphore` when acquire is cancelled.

- gh-90817 [https://github.com/python/cpython/issues/90817]: The **`locale.resetlocale()`** function is deprecated and will be removed in Python 3.13. Use `locale.setlocale(locale.LC_ALL, "")` instead. Patch by Victor Stinner.

- gh-91513 [https://github.com/python/cpython/issues/91513]: Added `taskName` attribute to `logging` module for use with `asyncio` tasks.

- gh-74696 [https://github.com/python/cpython/issues/74696]: `shutil.make_archive()` no longer temporarily changes the current working directory during creation of standard `.zip` or tar archives.

- gh-92728 [https://github.com/python/cpython/issues/92728]: The **`re.template()`** function and the corresponding **`re.TEMPLATE`** and **`re.T`** flags are restored after they were removed in 3.11.0b1, but they are now deprecated, so they might be removed from Python 3.13.

- gh-93033 [https://github.com/python/cpython/issues/93033]: Search in some strings (platform dependent i.e [U+0xFFFF, U

+0x0100] on Windows or [U+0xFFFFFFFF, U +0x00010000] on Linux 64-bit) are now up to 10 times faster.

- gh-89973 [https://github.com/python/cpython/issues/89973]: Fix **`re.error`** raised in **`fnmatch`** if the pattern contains a character range with upper bound lower than lower bound (e.g. `[c-a]`). Now such ranges are interpreted as empty ranges.

- gh-93044 [https://github.com/python/cpython/issues/93044]: No longer convert the database argument of **`sqlite3.connect()`** to bytes before passing it to the factory.

- gh-93010 [https://github.com/python/cpython/issues/93010]: In a very special case, the email package tried to append the nonexistent `InvalidHeaderError` to the defect list. It should have been `InvalidHeaderDefect`.

- gh-92986 [https://github.com/python/cpython/issues/92986]: Fix **`ast.unparse()`** when `ImportFrom.level` is `None`

- gh-92932 [https://github.com/python/cpython/issues/92932]: Now **`dis()`** and **`get_instructions()`** handle operand values for instructions prefixed by `EXTENDED_ARG_QUICK`. Patch by Sam Gross and Donghee Na.

- gh-92675 [https://github.com/python/cpython/issues/92675]: Fix **`venv.ensure_directories()`** to accept **`pathlib.Path`** arguments in addition to **`str`** paths. Patch by David Foster.

- gh-87901 [https://github.com/python/cpython/issues/87901]: Removed the `encoding` argument from **`os.popen()`** that was added in 3.11b1.

- gh-91922 [https://github.com/python/cpython/issues/91922]: Fix function **`sqlite.connect()`** and the **`sqlite.Connection`** constructor on non-UTF-8 locales. Also, they now support bytes paths non-decodable with the current FS encoding.

- gh-92869 [https://github.com/python/cpython/issues/92869]: Added **`c_time_t`** to **`ctypes`**, which has the same size as the **`time_t`** type in C.

- gh-92839 [https://github.com/python/cpython/issues/92839]: Fixed crash resulting from calling bisect.insort() or bisect.insort_left() with the key argument not equal to `None`.

- gh-90473 [https://github.com/python/cpython/issues/90473]: **`subprocess`** now fails early on Emscripten and WASI platforms to work around missing **`os.pipe()`** on WASI.

- gh-89325 [https://github.com/python/cpython/issues/89325]: Removed many old deprecated **`unittest`** features: **`TestCase`** method aliases, undocumented and broken **`TestCase`** method `assertDictContainsSubset`, undocumented **`TestLoader.loadTestsFromModule`** parameter *use_load_tests*, and an underscored alias of the **`TextTestResult`** class.

- gh-92734 [https://github.com/python/cpython/issues/92734]: Allow multi-element reprs emitted by **`reprlib`** to be pretty-printed using configurable indentation.

- gh-92671 [https://github.com/python/cpython/issues/92671]: Fixed **`ast.unparse()`** for empty tuples in the assignment target context.

- gh-91581 [https://github.com/python/cpython/issues/91581]: **`utcfromtimestamp()`** no longer attempts to resolve `fold` in the pure Python implementation, since the fold is never 1 in UTC. In addition to being slightly faster in the common case, this also prevents some errors when the timestamp is close to **`datetime.min`**. Patch by Paul Ganssle.

- gh-86388 [https://github.com/python/cpython/issues/86388]: Removed randrange() functionality deprecated since Python 3.10. Formerly, randrange(10.0) losslessly converted to randrange(10). Now, it raises a TypeError. Also, the exception raised for non-integral values such as randrange(10.5) or randrange(“10”) has been changed from ValueError to

TypeError.

- gh-90385 [https://github.com/python/cpython/issues/90385]: Add `pathlib.Path.walk()` as an alternative to `os.walk()`.
- gh-92550 [https://github.com/python/cpython/issues/92550]: Fix `pathlib.Path.rglob()` for empty pattern.
- gh-92591 [https://github.com/python/cpython/issues/92591]: Allow `logging` filters to return a `logging.LogRecord` instance so that filters attached to `logging.Handler`s can enrich records without side effects on other handlers.
- gh-92445 [https://github.com/python/cpython/issues/92445]: Fix a bug in `argparse` where `nargs="*"` would raise an error instead of returning an empty list when 0 arguments were supplied if choice was also defined in `parser.add_argument`.
- gh-92547 [https://github.com/python/cpython/issues/92547]: Remove undocumented `sqlite3` features deprecated in Python 3.10:
  - `sqlite3.enable_shared_cache()`
  - `sqlite3.OptimizedUnicode`

  Patch by Erlend E. Aasland.
- gh-92530 [https://github.com/python/cpython/issues/92530]: Fix an issue that occurred after interrupting `threading.Condition.notify()`.
- gh-92531 [https://github.com/python/cpython/issues/92531]: The statistics.median_grouped() function now always return a float. Formerly, it did not convert the input type when for sequences of length one.
- gh-84131 [https://github.com/python/cpython/issues/84131]: The `pathlib.Path` deprecated method `link_to` has been removed. Use 3.10’s `hardlink_to()` method instead as its semantics are consistent with that of `symlink_to()`.
- gh-89336 [https://github.com/python/cpython/issues/89336]:

Removed **`configparser`** module APIs: the `SafeConfigParser` class alias, the `ParsingError.filename` property and parameter, and the `ConfigParser.readfp` method, all of which were deprecated since Python 3.2.

- gh-92391 [https://github.com/python/cpython/issues/92391]: Add **`__class_getitem__()`** to **`csv.DictReader`** and **`csv.DictWriter`**, allowing them to be parameterized at runtime. Patch by Marc Mueller.

- gh-91968 [https://github.com/python/cpython/issues/91968]: Add `SO_RTABLE` and `SO_USER_COOKIE` constants to **`socket`**.

- gh-91810 [https://github.com/python/cpython/issues/91810]: **`ElementTree`** method **`write()`** and function **`tostring()`** now use the text file's encoding («UTF-8» if not available) instead of locale encoding in XML declaration when `encoding="unicode"` is specified.

- gh-81790 [https://github.com/python/cpython/issues/81790]: **`os.path.splitdrive()`** now understands DOS device paths with UNC links (beginning `\\?\UNC\`). Contributed by Barney Gale.

- gh-91760 [https://github.com/python/cpython/issues/91760]: Apply more strict rules for numerical group references and group names in regular expressions. Only sequence of ASCII digits is now accepted as a numerical reference. The group name in bytes patterns and replacement strings can now only contain ASCII letters and digits and underscore.

- gh-90622 [https://github.com/python/cpython/issues/90622]: Worker processes for **`concurrent.futures.ProcessPoolExecutor`** are no longer spawned on demand (a feature added in 3.9) when the multiprocessing context start method is `"fork"` as that can lead to deadlocks in the child processes due to a fork happening while threads are running.

- gh-91577 [https://github.com/python/cpython/issues/91577]: Move imports in `SharedMemory` methods to module level so that they can be executed late in python finalization.

- gh-91581 [https://github.com/python/cpython/issues/91581]: Remove an unhandled error case in the C implementation of calls to `datetime.fromtimestamp` with no time zone (i.e. getting a local time from an epoch timestamp). This should have no user-facing effect other than giving a possibly more accurate error message when called with timestamps that fall on 10000-01-01 in the local time. Patch by Paul Ganssle.

- gh-91539 [https://github.com/python/cpython/issues/91539]: Improve performance of `urllib.request.getproxies_environment` when there are many environment variables

- gh-91524 [https://github.com/python/cpython/issues/91524]: Speed up the regular expression substitution (functions `re.sub()` and `re.subn()` and corresponding `re.Pattern` methods) for replacement strings containing group references by 2–3 times.

- gh-91447 [https://github.com/python/cpython/issues/91447]: Fix findtext in the xml module to only give an empty string when the text attribute is set to `None`.

- gh-91456 [https://github.com/python/cpython/issues/91456]: Deprecate current default auto() behavior: In 3.13 the default will be for for auto() to always return the largest member value incremented by 1, and to raise if incompatible value types are used.

- bpo-47231 [https://bugs.python.org/issue?@action=redirect&bpo=47231]: Fixed an issue with inconsistent trailing slashes in tarfile longname directories.

- bpo-39064 [https://bugs.python.org/issue?@action=redirect&bpo=39064]: `zipfile.ZipFile` now raises `zipfile.BadZipFile` instead of `ValueError` when reading a corrupt zip file in which the central directory offset is negative.

- bpo-41287 [https://bugs.python.org/issue?@action=redirect&bpo=41287]: Fix handling of the `doc` argument in subclasses of **`property()`**.

- gh-90005 [https://github.com/python/cpython/issues/90005]: **`ctypes`** dependency `libffi` is now detected with `pkg-config`.

- bpo-32547 [https://bugs.python.org/issue?@action=redirect&bpo=32547]: The constructors for **`DictWriter`** and **`DictReader`** now coerce the `fieldnames` argument to a **`list`** if it is an iterator.

- bpo-35540 [https://bugs.python.org/issue?@action=redirect&bpo=35540]: Fix **`dataclasses.asdict()`** crash when **`collections.defaultdict`** is present in the attributes.

- bpo-47063 [https://bugs.python.org/issue?@action=redirect&bpo=47063]: Add an index_pages parameter to support using non-default index page names.

- bpo-47025 [https://bugs.python.org/issue?@action=redirect&bpo=47025]: Drop support for **`bytes`** on **`sys.path`**.

- bpo-46951 [https://bugs.python.org/issue?@action=redirect&bpo=46951]: Order the contents of zipapp archives, to make builds more reproducible.

- bpo-42777 [https://bugs.python.org/issue?@action=redirect&bpo=42777]: Implement **`pathlib.Path.is_mount()`** for Windows paths.

- bpo-46755 [https://bugs.python.org/issue?@action=redirect&bpo=46755]: In **`QueueHandler`**, clear `stack_info` from **`LogRecord`** to prevent stack trace from being written twice.

- bpo-45393 [https://bugs.python.org/issue?@action=redirect&bpo=45393]: Fix the formatting for `await x` and `not x` in the operator precedence table when using the

`help()` system.

- bpo-46642 [https://bugs.python.org/issue?@action=redirect&bpo=46642]: Improve error message when trying to subclass an instance of **`typing.TypeVar`**, **`typing.ParamSpec`**, **`typing.TypeVarTuple`**, etc. Based on patch by Gregory Beauregard.

- bpo-46364 [https://bugs.python.org/issue?@action=redirect&bpo=46364]: Restrict use of sockets instead of pipes for stdin of subprocesses created by **`asyncio`** to AIX platform only.

- bpo-28249 [https://bugs.python.org/issue?@action=redirect&bpo=28249]: Set **`doctest.DocTest.lineno`** to `None` when an object does not have **`__doc__`**.

- bpo-46197 [https://bugs.python.org/issue?@action=redirect&bpo=46197]: Fix **`ensurepip`** environment isolation for subprocess running `pip`.

- bpo-45924 [https://bugs.python.org/issue?@action=redirect&bpo=45924]: Fix **`asyncio`** incorrect traceback when future's exception is raised multiple times. Patch by Kumar Aditya.

- bpo-45046 [https://bugs.python.org/issue?@action=redirect&bpo=45046]: Add support of context managers in **`unittest`**: methods **`enterContext()`** and **`enterClassContext()`** of class **`TestCase`**, method **`enterAsyncContext()`** of class **`IsolatedAsyncioTestCase`** and function **`unittest.enterModuleContext()`**.

- bpo-44173 [https://bugs.python.org/issue?@action=redirect&bpo=44173]: Enable fast seeking of uncompressed unencrypted **`zipfile.ZipExtFile`**

- bpo-42627 [https://bugs.python.org/issue?@action=redirect&bpo=42627]: Fix incorrect parsing of Windows registry proxy settings

- bpo-42047 [https://bugs.python.org/issue?@action=redirect&bpo=42047]: Add `threading.get_native_id()` support for DragonFly BSD. Patch by David Carlier.

- bpo-14243 [https://bugs.python.org/issue?@action=redirect&bpo=14243]: The `tempfile.NamedTemporaryFile` function has a new optional parameter *delete_on_close*

- bpo-41246 [https://bugs.python.org/issue?@action=redirect&bpo=41246]: Give the same callback function for when the overlapped operation is done to the functions `recv`, `recv_into`, `recvfrom`, `sendto`, `send` and `sendfile` inside `IocpProactor`.

- bpo-39264 [https://bugs.python.org/issue?@action=redirect&bpo=39264]: Fixed `collections.UserDict.get()` to not call `__missing__()` when a value is not found. This matches the behavior of `dict`. Patch by Bar Harel.

- bpo-38693 [https://bugs.python.org/issue?@action=redirect&bpo=38693]: `importlib` now uses f-strings internally instead of `str.format`.

- bpo-38267 [https://bugs.python.org/issue?@action=redirect&bpo=38267]: Add *timeout* parameter to `asyncio.loop.shutdown_default_executor()`. The default value is `None`, which means the executor will be given an unlimited amount of time. When called from `asyncio.Runner` or `asyncio.run()`, the default timeout is 5 minutes.

- bpo-34828 [https://bugs.python.org/issue?@action=redirect&bpo=34828]: `sqlite3.Connection.iterdump()` now handles databases that use `AUTOINCREMENT` in one or more tables.

- bpo-32990 [https://bugs.python.org/issue?@action=redirect&bpo=32990]: Support reading wave files with the `WAVE_FORMAT_EXTENSIBLE` format in the `wave` module.

- bpo-26253 [https://bugs.python.org/issue?@action=redirect&bpo=26253]: Allow adjustable compression level for tarfile streams in `tarfile.open()`.

Documentation

- gh-85525 [https://github.com/python/cpython/issues/85525]: Remove extra row
- gh-86404 [https://github.com/python/cpython/issues/86404]: Deprecated tools `make suspicious` and `rstlint.py` are now removed. They have been replaced by sphinx-lint [https://pypi.org/project/sphinx-lint/].
- gh-97741 [https://github.com/python/cpython/issues/97741]: Fix `!` in c domain ref target syntax via a `conf.py` patch, so it works as intended to disable ref target resolution.
- gh-96432 [https://github.com/python/cpython/issues/96432]: Fraction literals now support whitespace around the forward slash, `Fraction('2 / 3')`.
- gh-96098 [https://github.com/python/cpython/issues/96098]: Improve discoverability of the higher level concurrent.futures module by providing clearer links from the lower level threading and multiprocessing modules.
- gh-95957 [https://github.com/python/cpython/issues/95957]: What's New 3.11 now has instructions for how to provide compiler and linker flags for Tcl/Tk and OpenSSL on RHEL 7 and CentOS 7.
- gh-95588 [https://github.com/python/cpython/issues/95588]: Clarified the conflicting advice given in the `ast` documentation about `ast.literal_eval()` being «safe» for use on untrusted input while at the same time warning that it can crash the process. The latter statement is true and is deemed unfixable without a large amount of work unsuitable for a bugfix. So we keep the warning and no longer claim that `literal_eval` is safe.
- gh-91207 [https://github.com/python/cpython/issues/91207]: Fix stylesheet not working in Windows CHM htmlhelp docs and add warning that they are deprecated. Contributed by C.A.M.

Gerlach.

- gh-95454 [https://github.com/python/cpython/issues/95454]: Replaced incorrectly written true/false values in documentation. Patch by Robert O'Shea
- gh-95451 [https://github.com/python/cpython/issues/95451]: Update library documentation with availability information on WebAssembly platforms `wasm32-emscripten` and `wasm32-wasi`.
- gh-95415 [https://github.com/python/cpython/issues/95415]: Use consistent syntax for platform availability. The directive now supports a content body and emits a warning when it encounters an unknown platform.
- gh-94321 [https://github.com/python/cpython/issues/94321]: Document the **PEP 246** [https://peps.python.org/pep-0246/] style protocol type **`sqlite3.PrepareProtocol`**.
- gh-86128 [https://github.com/python/cpython/issues/86128]: Document a limitation in ThreadPoolExecutor where its exit handler is executed before any handlers in atexit.
- gh-61162 [https://github.com/python/cpython/issues/61162]: Clarify **`sqlite3`** behavior when Como usar la conexión con un administrador de contexto.
- gh-87260 [https://github.com/python/cpython/issues/87260]: Align **`sqlite3`** argument specs with the actual implementation.
- gh-86986 [https://github.com/python/cpython/issues/86986]: The minimum Sphinx version required to build the documentation is now 3.2.
- gh-88831 [https://github.com/python/cpython/issues/88831]: Augmented documentation of asyncio.create_task(). Clarified the need to keep strong references to tasks and added a code snippet detailing how to do this.
- gh-86438 [https://github.com/python/cpython/issues/86438]: Clarify that **`-W`** and **`PYTHONWARNINGS`** are matched literally and case-insensitively, rather than as regular expressions, in **`warnings`**.
- gh-93031 [https://github.com/python/cpython/issues/93031]: Update tutorial introduction output to use 3.10+ SyntaxError invalid range.
- gh-92240 [https://github.com/python/cpython/issues/92240]: Added release dates for «What's New in Python 3.X» for 3.0, 3.1, 3.2, 3.8 and 3.10

- bpo-47161 [https://bugs.python.org/issue?@action=redirect&bpo=47161]: Document that **`pathlib.PurePath`** does not collapse initial double slashes because they denote UNC paths.
- bpo-40838 [https://bugs.python.org/issue?@action=redirect&bpo=40838]: Document that **`inspect.getdoc()`**, **`inspect.getmodule()`**, and **`inspect.getsourcefile()`** might return `None`.
- bpo-43689 [https://bugs.python.org/issue?@action=redirect&bpo=43689]: The `Differ` documentation now also mentions other whitespace characters, which make it harder to understand the diff output.
- bpo-38056 [https://bugs.python.org/issue?@action=redirect&bpo=38056]: Overhaul the Manejadores de errores documentation in **`codecs`**.
- bpo-13553 [https://bugs.python.org/issue?@action=redirect&bpo=13553]: Document tkinter.Tk args.

Tests

- gh-95027 [https://github.com/python/cpython/issues/95027]: On Windows, when the Python test suite is run with the `-jN` option, the ANSI code page is now used as the encoding for the stdout temporary file, rather than using UTF-8 which can lead to decoding errors. Patch by Victor Stinner.
- gh-96624 [https://github.com/python/cpython/issues/96624]: Fixed the failure of repeated runs of `test.test_unittest` caused by side effects in `test_dotted_but_module_not_loaded`.
- gh-95243 [https://github.com/python/cpython/issues/95243]: Mitigate the inherent race condition from using find_unused_port() in testSockName() by trying to find an unused port a few times before failing. Patch by Ross Burton.
- gh-95573 [https://github.com/python/cpython/issues/95573]: Lib/test/test_asyncio/test_ssl.py [https://github.com/python/cpython/tree/3.14/Lib/test/test_asyncio/test_ssl.py] exposed a bug in the macOS kernel where intense concurrent load on non-blocking sockets occasionally causes **`errno.ENOBUFS`** («No buffer space available») to be emitted. FB11063974 filed with Apple, in the mean time as a workaround buffer size used in

tests on macOS is decreased to avoid intermittent failures. Patch by Fantix King.

- gh-95280 [https://github.com/python/cpython/issues/95280]: Fix problem with `test_ssl` `test_get_ciphers` on systems that require perfect forward secrecy (PFS) ciphers.
- gh-95212 [https://github.com/python/cpython/issues/95212]: Make multiprocessing test case `test_shared_memory_recreate` parallel-safe.
- gh-95218 [https://github.com/python/cpython/issues/95218]: Move tests for importlib.resources into test_importlib.resources.
- gh-93963 [https://github.com/python/cpython/issues/93963]: Updated tests to use preferred location for `importlib.resources` ABCs.
- gh-94675 [https://github.com/python/cpython/issues/94675]: Add a regression test for **`re`** exponentional slowdown when using rjsmin.
- gh-91330 [https://github.com/python/cpython/issues/91330]: Added more tests for **`dataclasses`** to cover behavior with data descriptor-based fields.
- gh-94208 [https://github.com/python/cpython/issues/94208]: `test_ssl` is now checking for supported TLS version and protocols in more tests.
- gh-94315 [https://github.com/python/cpython/issues/94315]: Tests now check for DAC override capability instead of relying on **`os.geteuid()`**.
- gh-54781 [https://github.com/python/cpython/issues/54781]: Rename test_tk to test_tkinter, and rename test_ttk_guionly to test_ttk. Patch by Victor Stinner.
- gh-93839 [https://github.com/python/cpython/issues/93839]: Move `Lib/ctypes/test/` to `Lib/test/test_ctypes/`. Patch by Victor Stinner.
- gh-93951 [https://github.com/python/cpython/issues/93951]: In test_bdb.StateTestCase.test_skip, avoid including auxiliary importers.
- gh-93957 [https://github.com/python/cpython/issues/93957]: Provide nicer error reporting from subprocesses in test_venv.EnsurePipTest.test_with_pip.
- gh-93884 [https://github.com/python/cpython/issues/93884]: Add test cases for **`PyNumber_ToBase()`** that take a large number or a non-int object as parameter.

- gh-93852 [https://github.com/python/cpython/issues/93852]: test_asyncio, test_logging, test_socket and test_socketserver now create AF_UNIX domains in the current directory to no longer fail with `OSError("AF_UNIX path too long")` if the temporary directory (the **`TMPDIR`** environment variable) is too long. Patch by Victor Stinner.
- gh-93353 [https://github.com/python/cpython/issues/93353]: regrtest now checks if a test leaks temporary files or directories if run with -jN option. Patch by Victor Stinner.
- gh-84461 [https://github.com/python/cpython/issues/84461]: `run_tests.py` now handles cross compiling env vars correctly and pass `HOSTRUNNER` to regression tests.
- gh-93616 [https://github.com/python/cpython/issues/93616]: `test_modulefinder` now creates a temporary directory in `ModuleFinderTest.setUp()` instead of module scope.
- gh-93575 [https://github.com/python/cpython/issues/93575]: Fix issue with test_unicode test_raiseMemError. The test case now use `test.support.calcobjsize` to calculate size of PyUnicode structs. **`sys.getsizeof()`** may return different size when string has UTF-8 memory.
- gh-90473 [https://github.com/python/cpython/issues/90473]: WASI does not have a `chmod(2)` syscall. **`os.chmod()`** is now a dummy function on WASI. Skip all tests that depend on working **`os.chmod()`**.
- gh-90473 [https://github.com/python/cpython/issues/90473]: Skip tests on WASI that require symlinks with absolute paths.
- gh-57539 [https://github.com/python/cpython/issues/57539]: Increase calendar test coverage for **`calendar.LocaleTextCalendar.formatweekday()`**.
- gh-90473 [https://github.com/python/cpython/issues/90473]: Skip symlink tests on WASI. wasmtime uses `openat2(2)` with `RESOLVE_BENEATH` flag, which prevents symlinks with absolute paths.
- gh-89858 [https://github.com/python/cpython/issues/89858]: Fix `test_embed` for out-of-tree builds. Patch by Kumar Aditya.
- gh-92886 [https://github.com/python/cpython/issues/92886]: Fixing tests that fail when running with optimizations (`-O`) in `test_imaplib.py`.
- gh-92886 [https://github.com/python/cpython/issues/92886]: Fixing tests that fail when running with optimizations (`-O`) in

`test_zipimport.py`

- gh-92886 [https://github.com/python/cpython/issues/92886]: Fixing tests that fail when running with optimizations (`-O`) in `test_py_compile.py`
- gh-92886 [https://github.com/python/cpython/issues/92886]: Fixing tests that fail when running with optimizations (`-O`) in `test_sys_settrace.py`.
- gh-92886 [https://github.com/python/cpython/issues/92886]: Fixing tests that fail when running with optimizations (`-O`) in `_test_multiprocessing.py`
- gh-92670 [https://github.com/python/cpython/issues/92670]: Skip `test_shutil.TestCopy.test_copyfile_nonexistent_dir` test on AIX as the test uses a trailing slash to force the OS consider the path as a directory, but on AIX the trailing slash has no effect and is considered as a file.
- gh-92514 [https://github.com/python/cpython/issues/92514]: Remove unused `test.support.BasicTestRunner`. Patch by Jelle Zijlstra.
- bpo-47016 [https://bugs.python.org/issue?@action=redirect&bpo=47016]: Create a GitHub Actions workflow for verifying bundled pip and setuptools. Patch by Illia Volochii and Adam Turner.

Build

- gh-96761 [https://github.com/python/cpython/issues/96761]: Fix the build process of clang compiler for **_bootstrap_python** if LTO optimization is applied. Patch by Matthias Görgens and Donghee Na.
- gh-96883 [https://github.com/python/cpython/issues/96883]: `wasm32-emscripten` builds for browsers now include **`concurrent.futures`** for **`asyncio`** and **`unittest.mock`**.
- gh-85936 [https://github.com/python/cpython/issues/85936]: CPython now uses the ThinLTO option as the default policy if the Clang compiler accepts the flag. Patch by Donghee Na.
- gh-96729 [https://github.com/python/cpython/issues/96729]: Ensure that Windows releases built with `Tools\msi\buildrelease.bat` are upgradable to and from official Python releases.

- gh-96269 [https://github.com/python/cpython/issues/96269]: Shared module targets now depend on new `MODULE_DEPS` variable, which includes `EXPORTSYMS`. This fixes a build order issue on unsupported AIX platform.
- gh-84461 [https://github.com/python/cpython/issues/84461]: `wasm32-emscripten` platform no longer builds **`resource`** module, **`getresuid()`**, **`getresgid()`**, and their setters. The APIs are stubs and not functional.
- gh-95973 [https://github.com/python/cpython/issues/95973]: Add a new `--with-dsymutil` configure option to link debug information in macOS. Patch by Pablo Galindo.
- gh-90536 [https://github.com/python/cpython/issues/90536]: Use the BOLT post-link optimizer to improve performance, particularly on medium-to-large applications.
- gh-93744 [https://github.com/python/cpython/issues/93744]: Remove the `configure --with-cxx-main` build option: it didn't work for many years. Remove the `MAINCC` variable from `configure` and `Makefile`. Patch by Victor Stinner.
- gh-94801 [https://github.com/python/cpython/issues/94801]: Fix a regression in `configure` script that caused some header checks to ignore custom `CPPFLAGS`. The regression was introduced in gh-94802 [https://github.com/python/cpython/issues/94802].
- gh-95145 [https://github.com/python/cpython/issues/95145]: wasm32-wasi builds no longer depend on WASIX's pthread stubs. Python now has its own stubbed pthread API.
- gh-95174 [https://github.com/python/cpython/issues/95174]: Python now detects missing `dup` function in WASI and works around some missing **`errno`**, **`select`**, and **`socket`** constants.
- gh-95174 [https://github.com/python/cpython/issues/95174]: Python now skips missing **`socket`** functions and methods on WASI. WASI can only create sockets from existing fd / accept and has no netdb.
- gh-95085 [https://github.com/python/cpython/issues/95085]: Platforms `wasm32-unknown-emscripten` and `wasm32-unknown-wasi` have been promoted to **PEP 11** [https://peps.python.org/pep-0011/] tier 3 platform support.
- gh-94847 [https://github.com/python/cpython/issues/94847]: Fixed `_decimal` module build issue on GCC when compiling with LTO and pydebug. Debug builds no longer force inlining of

functions.

- gh-94841 [https://github.com/python/cpython/issues/94841]: Fix the possible performance regression of **`PyObject_Free()`** compiled with MSVC version 1932.
- gh-94801 [https://github.com/python/cpython/issues/94801]: `configure` now uses custom flags like `ZLIB_CFLAGS` and `ZLIB_LIBS` when searching for headers and libraries.
- gh-94773 [https://github.com/python/cpython/issues/94773]: `deepfreeze.py` now supports code object with frozensets that contain incompatible, unsortable types.
- gh-94682 [https://github.com/python/cpython/issues/94682]: Build and test with OpenSSL 1.1.1q
- gh-90005 [https://github.com/python/cpython/issues/90005]: Dependencies of **`readline`** and **`curses`** module are now detected in `configure` script with `pkg-config`. Only `ncurses` / `ncursesw` are detected automatically. The old `curses` library is not configured automatically. Workaround for missing `termcap` or `tinfo` library has been removed.
- gh-90005 [https://github.com/python/cpython/issues/90005]: Fix building `_ctypes` extension without `pkg-config`.
- gh-90005 [https://github.com/python/cpython/issues/90005]: `_dbm` module dependencies are now detected by configure.
- gh-94404 [https://github.com/python/cpython/issues/94404]: `makesetup` now works around an issue with sed on macOS and uses correct CFLAGS for object files that end up in a shared extension. Module CFLAGS are used before PY_STDMODULE_CFLAGS to avoid clashes with system headers.
- gh-93939 [https://github.com/python/cpython/issues/93939]: C extension modules are now built by `configure` and `make` instead of `distutils` and `setup.py`.
- gh-93939 [https://github.com/python/cpython/issues/93939]: The `2to3`, `idle`, and `pydoc` scripts are now generated and installed by `Makefile` instead of `setup.py`.
- gh-94280 [https://github.com/python/cpython/issues/94280]: Updated pegen regeneration script on Windows to find and use Python 3.9 or higher. Prior to this, pegen regeneration already required 3.9 or higher, but the script may have used lower versions of Python.
- gh-93584 [https://github.com/python/cpython/issues/93584]: Address

race condition in `Makefile` when installing a PGO build. All `test` and `install` targets now depend on `all` target.
- gh-93491 [https://github.com/python/cpython/issues/93491]: `configure` now detects and reports **PEP 11** [https://peps.python.org/pep-0011/] support tiers.
- gh-69093 [https://github.com/python/cpython/issues/69093]: Fix `Modules/Setup.stdlib.in` rule for `_sqlite3` extension.
- gh-93207 [https://github.com/python/cpython/issues/93207]: `va_start()` with two parameters, like `va_start(args, format),` is now required to build Python. `va_start()` is no longer called with a single parameter. Patch by Kumar Aditya.
- gh-93202 [https://github.com/python/cpython/issues/93202]: Python now always use the `%zu` and `%zd` printf formats to format a **`size_t`** or `Py_ssize_t` number. Building Python 3.12 requires a C11 compiler, so these printf formats are now always supported. Patch by Victor Stinner.
- gh-90473 [https://github.com/python/cpython/issues/90473]: Disable pymalloc and increase stack size on `wasm32-wasi`.
- bpo-34449 [https://bugs.python.org/issue?@action=redirect&bpo=34449]: Drop invalid compiler switch `-fPIC` for HP aCC on HP-UX. Patch by Michael Osipov.

Windows

- gh-98360 [https://github.com/python/cpython/issues/98360]: Fixes **`multiprocessing`** spawning child processes on Windows from a virtual environment to ensure that child processes that also use **`multiprocessing`** to spawn more children will recognize that they are in a virtual environment.
- gh-98414 [https://github.com/python/cpython/issues/98414]: Fix `py.exe` launcher handling of `-V:<company>/` option when default preferences have been set in environment variables or configuration files.
- gh-97728 [https://github.com/python/cpython/issues/97728]: Fix possible crashes caused by the use of uninitialized variables when pass invalid arguments in **`os.system()`** on Windows and in Windows-specific modules (like `winreg`).
- gh-90989 [https://github.com/python/cpython/issues/90989]: Made Python install manager install per-user by default (unless an

all users install already exists), and clarify some text in the installer.

- gh-97649 [https://github.com/python/cpython/issues/97649]: The `Tools` directory is no longer installed on Windows
- gh-96965 [https://github.com/python/cpython/issues/96965]: Update libffi to 3.4.3
- gh-96577 [https://github.com/python/cpython/issues/96577]: Fixes a potential buffer overrun in **`msilib`**.
- gh-96559 [https://github.com/python/cpython/issues/96559]: Fixes the Windows launcher not using the compatible interpretation of default tags found in configuration files when no tag was passed to the command.
- gh-94781 [https://github.com/python/cpython/issues/94781]: Fix `pcbuild.proj` to clean previous instances of output files in `Python\deepfreeze` and `Python\frozen_modules` directories on Windows. Patch by Charlie Zhao.
- gh-89545 [https://github.com/python/cpython/issues/89545]: Updates **`platform`** code getting the Windows version to use native Windows Management Instrumentation (WMI) queries to determine OS version, type, and architecture.
- gh-95733 [https://github.com/python/cpython/issues/95733]: Make certain requirements of the Windows Store package optional to allow installing on earlier updates of Windows.
- gh-95656 [https://github.com/python/cpython/issues/95656]: Enable the **`enable_load_extension()`** **`sqlite3`** API.
- gh-95587 [https://github.com/python/cpython/issues/95587]: Fixes some issues where the Windows installer would incorrectly detect certain features of an existing install when upgrading.
- gh-94399 [https://github.com/python/cpython/issues/94399]: Restores the behaviour of Python install manager for `/usr/bin/env` shebang lines, which will now search **`PATH`** for an executable matching the given command. If none is found, the usual search process is used.
- gh-95445 [https://github.com/python/cpython/issues/95445]: Fixes the unsuccessful removal of the HTML document directory when uninstalling with Windows msi.
- gh-95359 [https://github.com/python/cpython/issues/95359]: Fix Python install manager handling of `py.ini` commands (it was incorrectly expecting a `py_` prefix on keys) and crashes when reading per-user configuration file.

- gh-95285 [https://github.com/python/cpython/issues/95285]: Fix Python install manager handling of command lines where it is only passed a short executable name.
- gh-90844 [https://github.com/python/cpython/issues/90844]: Allow virtual environments to correctly launch when they have spaces in the path.
- gh-94772 [https://github.com/python/cpython/issues/94772]: Fix incorrect handling of shebang lines in py.exe launcher
- gh-94018 [https://github.com/python/cpython/issues/94018]: **`zipfile`** will now remove trailing spaces from path components when extracting files on Windows.
- gh-93824 [https://github.com/python/cpython/issues/93824]: Drag and drop of files onto Python files in Windows Explorer has been enabled for Windows ARM64.
- gh-43414 [https://github.com/python/cpython/issues/43414]: **`os.get_terminal_size()`** now attempts to read the size from any provided handle, rather than only supporting file descriptors 0, 1 and 2.
- gh-92817 [https://github.com/python/cpython/issues/92817]: Ensures that `py.exe` will prefer an active virtual environment over default tags specified with environment variables or through a `py.ini` file.
- gh-92984 [https://github.com/python/cpython/issues/92984]: Explicitly disable incremental linking for non-Debug builds
- gh-92841 [https://github.com/python/cpython/issues/92841]: **`asyncio`** no longer throws `RuntimeError: Event loop is closed` on interpreter exit after asynchronous socket activity. Patch by Oleg Iarygin.
- bpo-46907 [https://bugs.python.org/issue?@action=redirect&bpo=46907]: Update Windows installer to use SQLite 3.38.4.
- gh-91061 [https://github.com/python/cpython/issues/91061]: Accept os.PathLike for the argument to winsound.PlaySound
- bpo-42658 [https://bugs.python.org/issue?@action=redirect&bpo=42658]: Support native Windows case-insensitive path comparisons by using `LCMapStringEx` instead of **`str.lower()`** in **`ntpath.normcase()`**. Add `LCMapStringEx` to the **`_winapi`** module.
- bpo-38704 [https://bugs.python.org/issue?@action=redirect&bpo=38704]: Prevent installation on

unsupported Windows versions.

macOS

- gh-97897 [https://github.com/python/cpython/issues/97897]: The macOS 13 SDK includes support for the `mkfifoat` and `mknodat` system calls. Using the `dir_fd` option with either **`os.mkfifo()`** or **`os.mknod()`** could result in a segfault if cpython is built with the macOS 13 SDK but run on an earlier version of macOS. Prevent this by adding runtime support for detection of these system calls («weaklinking») as is done for other newer syscalls on macOS.

IDLE

- gh-97527 [https://github.com/python/cpython/issues/97527]: Fix a bug in the previous bugfix that caused IDLE to not start when run with 3.10.8, 3.12.0a1, and at least Microsoft Python 3.10.2288.0 installed without the Lib/test package. 3.11.0 was never affected.
- gh-65802 [https://github.com/python/cpython/issues/65802]: Document handling of extensions in Save As dialogs.
- gh-95191 [https://github.com/python/cpython/issues/95191]: Include prompts when saving Shell (interactive input and output).
- gh-95511 [https://github.com/python/cpython/issues/95511]: Fix the Shell context menu copy-with-prompts bug of copying an extra line when one selects whole lines.
- gh-95471 [https://github.com/python/cpython/issues/95471]: In the Edit menu, move `Select All` and add a new separator.
- gh-95411 [https://github.com/python/cpython/issues/95411]: Enable using IDLE’s module browser with .pyw files.
- gh-89610 [https://github.com/python/cpython/issues/89610]: Add .pyi as a recognized extension for IDLE on macOS. This allows opening stub files by double clicking on them in the Finder.

Tools/Demos

- gh-68686 [https://github.com/python/cpython/issues/68686]: Remove ptags and eptags scripts.
- gh-97681 [https://github.com/python/cpython/issues/97681]: Remove

the `Tools/demo/` directory which contained old demo scripts. A copy can be found in the old-demos project [https://github.com/gvanrossum/old-demos]. Patch by Victor Stinner.

- gh-97669 [https://github.com/python/cpython/issues/97669]: Remove outdated example scripts of the `Tools/scripts/` directory. A copy can be found in the old-demos project [https://github.com/gvanrossum/old-demos]. Patch by Victor Stinner.
- gh-95853 [https://github.com/python/cpython/issues/95853]: The `wasm_build.py` script now pre-builds Emscripten ports, checks for broken EMSDK versions, and warns about pkg-config env vars.
- gh-95853 [https://github.com/python/cpython/issues/95853]: The new tool `Tools/wasm/wasm_builder.py` automates configure, compile, and test steps for building CPython on WebAssembly platforms.
- gh-95731 [https://github.com/python/cpython/issues/95731]: Fix handling of module docstrings in `Tools/i18n/pygettext.py`.
- gh-93939 [https://github.com/python/cpython/issues/93939]: Add script `Tools/scripts/check_modules.py` to check and validate builtin and shared extension modules. The script also handles `Modules/Setup` and will eventually replace `setup.py`.
- gh-94538 [https://github.com/python/cpython/issues/94538]: Fix Argument Clinic output to custom file destinations. Patch by Erlend E. Aasland.
- gh-94430 [https://github.com/python/cpython/issues/94430]: Allow parameters named `module` and `self` with custom C names in Argument Clinic. Patch by Erlend E. Aasland
- gh-86087 [https://github.com/python/cpython/issues/86087]: The `Tools/scripts/parseentities.py` script used to parse HTML4 entities has been removed.

C API

- gh-98393 [https://github.com/python/cpython/issues/98393]: The **`PyUnicode_FSDecoder()`** function no longer accepts bytes-like paths, like **`bytearray`** and **`memoryview`** types: only the exact **`bytes`** type is accepted for bytes strings. Patch by Victor Stinner.

- gh-91051 [https://github.com/python/cpython/issues/91051]: Add **`PyType_Watch()`** and related APIs to allow callbacks on **`PyType_Modified()`**.

- gh-95756 [https://github.com/python/cpython/issues/95756]: Lazily create and cache `co_` attributes for better performance for code getters.

- gh-96512 [https://github.com/python/cpython/issues/96512]: Configuration for the integer string conversion length limitation now lives in the PyConfig C API struct.

- gh-95589 [https://github.com/python/cpython/issues/95589]: Extensions classes that set `tp_dictoffset` and `tp_weaklistoffset` lose the support for multiple inheritance, but are now safe. Extension classes should use **`Py_TPFLAGS_MANAGED_DICT`** and **`Py_TPFLAGS_MANAGED_WEAKREF`** instead.

- gh-95781 [https://github.com/python/cpython/issues/95781]: An unrecognized format character in **`PyUnicode_FromFormat()`** and **`PyUnicode_FromFormatV()`** now sets a **`SystemError`**. In previous versions it caused all the rest of the format string to be copied as-is to the result string, and any extra arguments discarded.

- gh-92678 [https://github.com/python/cpython/issues/92678]: Restore the 3.10 behavior for multiple inheritance of C extension classes that store their dictionary at the end of the struct.

- gh-92678 [https://github.com/python/cpython/issues/92678]: Support C extensions using managed dictionaries by setting the `Py_TPFLAGS_MANAGED_DICT` flag.

- gh-93274 [https://github.com/python/cpython/issues/93274]: API for implementing vectorcall (**`Py_TPFLAGS_HAVE_VECTORCALL`**, **`PyVectorcall_NARGS()`** and **`PyVectorcall_Call()`**) was added to the limited API and stable ABI.

- gh-95504 [https://github.com/python/cpython/issues/95504]: Fix sign placement when specifying width or precision in `PyUnicode_FromFormat()` and `PyUnicode_FromFormatV()`. Patch by Philip Georgi.

- gh-93012 [https://github.com/python/cpython/issues/93012]: The `Py_TPFLAGS_HAVE_VECTORCALL` flag is now removed from a class when the class's `__call__()` method is reassigned. This makes vectorcall safe to use with mutable types (i.e. heap types without the `immutable` flag). Mutable types that do not override `tp_call` now inherit the `Py_TPFLAGS_HAVE_VECTORCALL` flag.

- gh-95388 [https://github.com/python/cpython/issues/95388]: Creating `immutable types` with mutable bases is deprecated and is planned to be disabled in Python 3.14.

- gh-92678 [https://github.com/python/cpython/issues/92678]: Adds unstable C-API functions `_PyObject_VisitManagedDict` and `_PyObject_ClearManagedDict` to allow C extensions to allow the VM to manage their object's dictionaries.

- gh-94936 [https://github.com/python/cpython/issues/94936]: Added `PyCode_GetVarnames()`, `PyCode_GetCellvars()` and `PyCode_GetFreevars()` for accessing `co_varnames`, `co_cellvars` and `co_freevars` respectively via the C API.

- gh-94930 [https://github.com/python/cpython/issues/94930]: Fix `SystemError` raised when `PyArg_ParseTupleAndKeywords()` is used with `#` in `(...)` but without `PY_SSIZE_T_CLEAN` defined.

- gh-94731 [https://github.com/python/cpython/issues/94731]: Python again uses C-style casts for most casting operations when compiled with C++. This may trigger compiler warnings, if they are enabled with e.g. `-Wold-style-cast` or `-Wzero-as-null-pointer-constant` options for `g++`.

- gh-93937 [https://github.com/python/cpython/issues/93937]: The following frame functions and type are now directly available with `#include <Python.h>`, it's no longer needed to add

`#include <frameobject.h>`:

  - **`PyFrame_Check()`**
  - **`PyFrame_GetBack()`**
  - **`PyFrame_GetBuiltins()`**
  - **`PyFrame_GetGenerator()`**
  - **`PyFrame_GetGlobals()`**
  - **`PyFrame_GetLasti()`**
  - **`PyFrame_GetLocals()`**
  - **`PyFrame_Type`**

  Patch by Victor Stinner.

- gh-91321 [https://github.com/python/cpython/issues/91321]: Fix the compatibility of the Python C API with C++ older than C++11. Patch by Victor Stinner.

- gh-91731 [https://github.com/python/cpython/issues/91731]: Avoid defining the `static_assert` when compiling with C++ 11, where this is a keyword and redefining it can lead to undefined behavior. Patch by Pablo Galindo

- gh-89546 [https://github.com/python/cpython/issues/89546]: **`PyType_FromMetaclass()`** (and other `PyType_From*` functions) now check that offsets and the base class's **`tp_basicsize`** fit in the new class's `tp_basicsize`.

- gh-93503 [https://github.com/python/cpython/issues/93503]: Add two new public functions to the public C-API, **`PyEval_SetProfileAllThreads()`** and **`PyEval_SetTraceAllThreads()`**, that allow to set tracing and profiling functions in all running threads in addition to the calling one. Also, two analogous functions to the **`threading`** module (**`threading.setprofile_all_threads()`** and **`threading.settrace_all_threads()`**) that allow to do the same from Python. Patch by Pablo Galindo

- gh-93442 [https://github.com/python/cpython/issues/93442]: Add C++ overloads for _Py_CAST_impl() to handle 0/NULL. This will allow C++ extensions that pass 0 or NULL to macros using

_Py_CAST() to continue to compile.

- gh-93466 [https://github.com/python/cpython/issues/93466]: Slot IDs in PyType_Spec may not be repeated. The documentation was updated to mention this. For some cases of repeated slots, PyType_FromSpec and related functions will now raise an exception.

- gh-92898 [https://github.com/python/cpython/issues/92898]: Fix C++ compiler warnings when casting function arguments to `PyObject*`. Patch by Serge Guelton.

- gh-93103 [https://github.com/python/cpython/issues/93103]: Deprecate global configuration variables, like **`Py_IgnoreEnvironmentFlag`**, in the documentation: the **`Py_InitializeFromConfig()`** API should be instead. Patch by Victor Stinner.

- gh-77782 [https://github.com/python/cpython/issues/77782]: Deprecate global configuration variable like **`Py_IgnoreEnvironmentFlag`**: the **`Py_InitializeFromConfig()`** API should be instead. Patch by Victor Stinner.

- gh-92913 [https://github.com/python/cpython/issues/92913]: Ensures changes to **`PyConfig.module_search_paths`** are ignored unless **`PyConfig.module_search_paths_set`** is set

- gh-92781 [https://github.com/python/cpython/issues/92781]: Avoid mixing declarations and code in the C API to fix the compiler warning: «ISO C90 forbids mixed declarations and code» [-Werror=declaration-after-statement]. Patch by Victor Stinner.

- gh-92651 [https://github.com/python/cpython/issues/92651]: Remove the `token.h` header file. There was never any public tokenizer C API. The `token.h` header file was only designed to be used by Python internals. Patch by Victor Stinner.

- gh-92536 [https://github.com/python/cpython/issues/92536]: Remove legacy Unicode APIs based on `Py_UNICODE*`.

- gh-85858 [https://github.com/python/cpython/issues/85858]: Remove the `PyUnicode_InternImmortal()` function and the `SSTATE_INTERNED_IMMORTAL` macro. Patch by Victor Stinner.
- gh-92193 [https://github.com/python/cpython/issues/92193]: Add new function **`PyFunction_SetVectorcall()`** to the C API which sets the vectorcall field of a given **`PyFunctionObject`**.

  Warning: extensions using this API must preserve the behavior of the unaltered function!
- gh-59121 [https://github.com/python/cpython/issues/59121]: Fixed an assert that prevented `PyRun_InteractiveOne` from providing tracebacks when parsing from the provided FD.
- bpo-45383 [https://bugs.python.org/issue?@action=redirect&bpo=45383]: The **`PyType_FromSpec()`** API will now find and use a metaclass based on the provided bases. An error will be raised if there is a metaclass conflict.

Python 3.11.0 beta 1

Release date: 2022-05-06

Security

- gh-57684 [https://github.com/python/cpython/issues/57684]: Add the **`-P`** command line option and the **`PYTHONSAFEPATH`** environment variable to not prepend a potentially unsafe path to **`sys.path`**. Patch by Victor Stinner.

Core and Builtins

- gh-89519 [https://github.com/python/cpython/issues/89519]: Chaining classmethod descriptors (introduced in bpo-19072 [https://bugs.python.org/issue?@action=redirect&bpo=19072]) is deprecated. It can no longer be used to wrap other descriptors such as property(). The core design of this feature was flawed, and it caused a number of downstream problems.

- gh-92345 [https://github.com/python/cpython/issues/92345]: `pymain_run_python()` now imports `readline` and `rlcompleter` before sys.path is extended to include the current working directory of an interactive interpreter. Non-interactive interpreters are not affected.
- bpo-43857 [https://bugs.python.org/issue?@action=redirect&bpo=43857]: Improve the **`AttributeError`** message when deleting a missing attribute. Patch by Géry Ogam.
- gh-92245 [https://github.com/python/cpython/issues/92245]: Make sure that PEP 523 is respected in all cases. In 3.11a7, specialization may have prevented Python-to-Python calls respecting PEP 523.
- gh-92203 [https://github.com/python/cpython/issues/92203]: Add a closure keyword-only parameter to **`exec()`**. It can only be specified when exec-ing a code object that uses free variables. When specified, it must be a tuple, with exactly the number of cell variables referenced by the code object. closure has a default value of `None`, and it must be `None` if the code object doesn't refer to any free variables.
- gh-91173 [https://github.com/python/cpython/issues/91173]: Disable frozen modules in debug builds. Patch by Kumar Aditya.
- gh-92114 [https://github.com/python/cpython/issues/92114]: Improve error message when subscript a type with `__class_getitem__` set to `None`.
- gh-92112 [https://github.com/python/cpython/issues/92112]: Fix crash triggered by an evil custom `mro()` on a metaclass.
- gh-92063 [https://github.com/python/cpython/issues/92063]: The `PRECALL_METHOD_DESCRIPTOR_FAST_WITH_KEYWORDS` instruction now ensures methods are called only on objects of the correct type.
- gh-92031 [https://github.com/python/cpython/issues/92031]: Deoptimize statically allocated code objects during `Py_FINALIZE()` so that future `_PyCode_Quicken` calls always start with unquickened code.
- gh-92036 [https://github.com/python/cpython/issues/92036]: Fix a crash in subinterpreters related to the garbage collector. When a subinterpreter is deleted, untrack all objects tracked by its GC. To prevent a crash in deallocator functions

expecting objects to be tracked by the GC, leak a strong reference to these objects on purpose, so they are never deleted and their deallocator functions are not called. Patch by Victor Stinner.

- gh-92032 [https://github.com/python/cpython/issues/92032]: The interpreter can now autocomplete soft keywords, as of now `match`, `case`, and `_` (wildcard pattern) from **PEP 634** [https://peps.python.org/pep-0634/].
- gh-87999 [https://github.com/python/cpython/issues/87999]: The warning emitted by the Python parser for a numeric literal immediately followed by keyword has been changed from deprecation warning to syntax warning.
- gh-91869 [https://github.com/python/cpython/issues/91869]: Fix an issue where specialized opcodes with extended arguments could produce incorrect tracing output or lead to assertion failures.
- gh-91603 [https://github.com/python/cpython/issues/91603]: Speed up **`types.UnionType`** instantiation. Based on patch provided by Yurii Karabas.
- gh-89373 [https://github.com/python/cpython/issues/89373]: If Python is built in debug mode, Python now ensures that deallocator functions leave the current exception unchanged. Patch by Victor Stinner.
- gh-91632 [https://github.com/python/cpython/issues/91632]: Fix a minor memory leak at exit: release the memory of the **`generic_alias_iterator`** type. Patch by Donghee Na.
- gh-81548 [https://github.com/python/cpython/issues/81548]: Octal escapes with value larger than `0o377` now produce a **`DeprecationWarning`**. In a future Python version they will be a **`SyntaxWarning`** and eventually a **`SyntaxError`**.
- bpo-43950 [https://bugs.python.org/issue?@action=redirect&bpo=43950]: Use a single compact table for line starts, ends and column offsets. Reduces memory consumption for location info by half
- gh-91102 [https://github.com/python/cpython/issues/91102]: Use Argument Clinic for **`EncodingMap`**. Patch by Oleg Iarygin.
- gh-91636 [https://github.com/python/cpython/issues/91636]: Fixed a crash in a garbage-collection edge-case, in which a `PyFunction_Type.tp_clear` function could leave a python function object in an inconsistent state.

- gh-91603 [https://github.com/python/cpython/issues/91603]: Speed up **`isinstance()`** and **`issubclass()`** checks for **`types.UnionType`**. Patch by Yurii Karabas.
- gh-91625 [https://github.com/python/cpython/issues/91625]: Fixed a bug in which adaptive opcodes ignored any preceding `EXTENDED_ARG`s on specialization failure.
- gh-78607 [https://github.com/python/cpython/issues/78607]: The LLTRACE special build now looks for the name `__lltrace__` defined in module globals, rather than the name `__ltrace__`, which had been introduced as a typo.
- gh-91576 [https://github.com/python/cpython/issues/91576]: Speed up iteration of ascii strings by 50%. Patch by Kumar Aditya.
- gh-89279 [https://github.com/python/cpython/issues/89279]: Improve interpreter performance on Windows by inlining a few specific macros.
- gh-91502 [https://github.com/python/cpython/issues/91502]: Add a new **`_PyFrame_IsEntryFrame()`** API function, to check if a **`PyFrameObject`** is an entry frame. Patch by Pablo Galindo.
- gh-91266 [https://github.com/python/cpython/issues/91266]: Refactor the `bytearray` strip methods `strip`, `lstrip` and `rstrip` to use a common implementation.
- gh-91479 [https://github.com/python/cpython/issues/91479]: Replaced the `__note__` field of **`BaseException`** (added in an earlier version of 3.11) with the final design of PEP 678 [https://peps.python.org/pep-0678/]. Namely, **`BaseException`** gets an **`add_note()`** method, and its `__notes__` field is created when necessary.
- gh-46055 [https://github.com/python/cpython/issues/46055]: Speed up right shift of negative integers, by removing unnecessary creation of temporaries. Original patch by Xinhang Xu, reworked by Mark Dickinson.
- gh-91462 [https://github.com/python/cpython/issues/91462]: Make the interpreter's low-level tracing (lltrace) feature output more readable by displaying opcode names (rather than just numbers), and by displaying stack contents before each opcode.
- gh-89455 [https://github.com/python/cpython/issues/89455]: Fixed an uninitialized bool value in the traceback printing code path that was introduced by the initial bpo-45292 [https://

bugs.python.org/issue?@action=redirect&bpo=45292] exception groups work.

- gh-91421 [https://github.com/python/cpython/issues/91421]: Fix a potential integer overflow in _Py_DecodeUTF8Ex.
- gh-91428 [https://github.com/python/cpython/issues/91428]: Add `static const char *const _PyOpcode_OpName[256] = {...};` to `opcode.h` for debug builds to assist in debugging the Python interpreter. It is now more convenient to make various forms of debugging output more human-readable by including opcode names rather than just the corresponding decimal digits.
- bpo-47120 [https://bugs.python.org/issue?@action=redirect&bpo=47120]: Make **`POP_JUMP_IF_TRUE`**, **`POP_JUMP_IF_FALSE`**, **`POP_JUMP_IF_NONE`** and **`POP_JUMP_IF_NOT_NONE`** virtual, mapping to new relative jump opcodes.
- bpo-45317 [https://bugs.python.org/issue?@action=redirect&bpo=45317]: Add internal documentation explaining design of new (for 3.11) frame stack.

Library

- bpo-47197 [https://bugs.python.org/issue?@action=redirect&bpo=47197]: ctypes used to mishandle `void` return types, so that for instance a function declared like `ctypes.CFUNCTYPE(None, ctypes.c_int)` would be called with signature `int f(int)` instead of `void f(int)`. Wasm targets require function pointers to be called with the correct signatures so this led to crashes. The problem is now fixed.

Core and Builtins

- bpo-47120 [https://bugs.python.org/issue?@action=redirect&bpo=47120]: Make opcodes **`JUMP_IF_TRUE_OR_POP`** and **`JUMP_IF_FALSE_OR_POP`** relative rather than absolute.

- bpo-47177 [https://bugs.python.org/issue?@action=redirect&bpo=47177]: Replace the `f_lasti` member of the internal `_PyInterpreterFrame` structure with a `prev_instr` pointer, which reduces overhead in the main interpreter loop. The `f_lasti` attribute of Python-layer frame objects is preserved for backward-compatibility.

- bpo-46961 [https://bugs.python.org/issue?@action=redirect&bpo=46961]: Integer mod/remainder operations, including the three-argument form of **`pow()`**, now consistently return ints from the global small integer cache when applicable.

- bpo-46962 [https://bugs.python.org/issue?@action=redirect&bpo=46962]: Classes and functions that unconditionally declared their docstrings ignoring the `--without-doc-strings` compilation flag no longer do so.

  The classes affected are **`ctypes.UnionType`**, **`pickle.PickleBuffer`**, **`testcapi.RecursingInfinitelyError`**, and **`types.GenericAlias`**.

  The functions affected are 24 methods in **`ctypes`**.

  Patch by Oleg Iarygin.

- bpo-46942 [https://bugs.python.org/issue?@action=redirect&bpo=46942]: Use Argument Clinic for the **`types.MethodType`** constructor. Patch by Oleg Iarygin.

- bpo-46764 [https://bugs.python.org/issue?@action=redirect&bpo=46764]: Fix wrapping bound methods with @classmethod

- bpo-43464 [https://bugs.python.org/issue?@action=redirect&bpo=43464]: Optimize **`set.intersection()`** for non-set arguments.

- bpo-46721 [https://bugs.python.org/issue?@action=redirect&bpo=46721]: Optimize **`set.issuperset()`** for non-set argument.

- bpo-46509 [https://bugs.python.org/issue?

@action=redirect&bpo=46509]: Add type-specialized versions of the `Py_DECREF()`, and use them for `float`, `int`, `str`, `bool`, and `None` to avoid pointer-chasing at runtime where types are known at C compile time.

Build

- bpo-46045 [https://bugs.python.org/issue?@action=redirect&bpo=46045]: Do not use POSIX semaphores on NetBSD

Core and Builtins

- bpo-36819 [https://bugs.python.org/issue?@action=redirect&bpo=36819]: Fix crashes in built-in encoders with error handlers that return position less or equal than the starting position of non-encodable characters.
- bpo-34093 [https://bugs.python.org/issue?@action=redirect&bpo=34093]: `marshal.dumps()` uses `FLAG_REF` for all interned strings. This makes output more deterministic and helps reproducible build.
- bpo-26579 [https://bugs.python.org/issue?@action=redirect&bpo=26579]: Added `object.__getstate__` which provides the default implementation of the `__getstate__()` method.

  Copying and pickling instances of subclasses of builtin types bytearray, set, frozenset, collections.OrderedDict, collections.deque, weakref.WeakSet, and datetime.tzinfo now copies and pickles instance attributes implemented as slots.

Library

- gh-87901 [https://github.com/python/cpython/issues/87901]: Add the *encoding* parameter to **`os.popen()`**.
- gh-90997 [https://github.com/python/cpython/issues/90997]: Fix an issue where **`dis`** utilities may interpret populated inline

cache entries as valid instructions.

- gh-92332 [https://github.com/python/cpython/issues/92332]: Deprecate `typing.Text` (removal of the class is currently not planned). Patch by Alex Waygood.

- gh-78157 [https://github.com/python/cpython/issues/78157]: Deprecate nested classes in enum definitions becoming members – in 3.13 they will be normal classes; add `member` and `nonmember` functions to allow control over results now.

- gh-92356 [https://github.com/python/cpython/issues/92356]: Fixed a performance regression in ctypes function calls.

- gh-90997 [https://github.com/python/cpython/issues/90997]: Show the actual named values stored in inline caches when `show_caches=True` is passed to `dis` utilities.

- gh-92301 [https://github.com/python/cpython/issues/92301]: Prefer `close_range()` to iterating over procfs for file descriptor closing in `subprocess` for better performance.

- gh-67248 [https://github.com/python/cpython/issues/67248]: Sort the miscellaneous topics in Cmd.do_help()

- gh-92210 [https://github.com/python/cpython/issues/92210]: Port `socket.__init__` to Argument Clinic. Patch by Cinder.

- gh-80010 [https://github.com/python/cpython/issues/80010]: Add support for generalized ISO 8601 parsing to `datetime.datetime.fromisoformat()`, `datetime.date.fromisoformat()` and `datetime.time.fromisoformat()`. Patch by Paul Ganssle.

- gh-92118 [https://github.com/python/cpython/issues/92118]: Fix a 3.11 regression in `contextmanager()`, which caused it to propagate exceptions with incorrect tracebacks.

- gh-90887 [https://github.com/python/cpython/issues/90887]: Adding `COPYFILE_STAT`, `COPYFILE_ACL` and `COPYFILE_XATTR` constants for `os.fcopyfile()` available in macOs.

- gh-91215 [https://github.com/python/cpython/issues/91215]: For **`@~dataclasses.dataclass`**, add *weakref_slot*. The new parameter defaults to `False`. If true, and if `slots=True`, add a slot named `"__weakref__"`, which will allow instances to be weakref'd. Contributed by Eric V. Smith

- gh-85984 [https://github.com/python/cpython/issues/85984]: New function os.login_tty() for Unix.

- gh-92128 [https://github.com/python/cpython/issues/92128]: Add **`__class_getitem__()`** to **`logging.LoggerAdapter`** and **`logging.StreamHandler`**, allowing them to be parameterized at runtime. Patch by Alex Waygood.

- gh-92049 [https://github.com/python/cpython/issues/92049]: Forbid pickling constants `re._constants.SUCCESS` etc. Previously, pickling did not fail, but the result could not be unpickled.

- gh-92062 [https://github.com/python/cpython/issues/92062]: **`inspect.Parameter`** now raises **`ValueError`** if `name` is a keyword, in addition to the existing check that it is an identifier.

- gh-87390 [https://github.com/python/cpython/issues/87390]: Add an `__unpacked__` attribute to **`types.GenericAlias`**. Patch by Jelle Zijlstra.

- gh-88089 [https://github.com/python/cpython/issues/88089]: Add support for generic **`typing.NamedTuple`**.

- gh-91996 [https://github.com/python/cpython/issues/91996]: New http.HTTPMethod enum to represent all the available HTTP request methods in a convenient way

- gh-91984 [https://github.com/python/cpython/issues/91984]: Modified test strings in test_argparse.py to not contain trailing spaces before end of line.

- gh-91952 [https://github.com/python/cpython/issues/91952]: Add `encoding="locale"` support to

`TextIOWrapper.reconfigure()`.

- gh-91954 [https://github.com/python/cpython/issues/91954]: Add *encoding* and *errors* arguments to **`subprocess.getoutput()`** and **`subprocess.getstatusoutput()`**.
- bpo-47029 [https://bugs.python.org/issue?@action=redirect&bpo=47029]: Always close the read end of the pipe used by **`multiprocessing.Queue`** *after* the last write of buffered data to the write end of the pipe to avoid **`BrokenPipeError`** at garbage collection and at **`multiprocessing.Queue.close()`** calls. Patch by Géry Ogam.
- gh-91928 [https://github.com/python/cpython/issues/91928]: Add `datetime.UTC` alias for `datetime.timezone.utc`.

  Patch by Kabir Kwatra.
- gh-68966 [https://github.com/python/cpython/issues/68966]: The **`mailcap`** module is now deprecated and will be removed in Python 3.13. See **PEP 594** [https://peps.python.org/pep-0594/] for the rationale and the **`mimetypes`** module for an alternative. Patch by Victor Stinner.
- gh-91401 [https://github.com/python/cpython/issues/91401]: Provide a way to disable **`subprocess`** use of `vfork()` just in case it is ever needed and document the existing mechanism for `posix_spawn()`.
- gh-64783 [https://github.com/python/cpython/issues/64783]: Fix **`signal.NSIG`** value on FreeBSD to accept signal numbers greater than 32, like **`signal.SIGRTMIN`** and **`signal.SIGRTMAX`**. Patch by Victor Stinner.
- gh-91910 [https://github.com/python/cpython/issues/91910]: Add missing f prefix to f-strings in error messages from the **`multiprocessing`** and **`asyncio`** modules.
- gh-91860 [https://github.com/python/cpython/issues/91860]: Add

`typing.dataclass_transform()`, implementing **PEP 681** [https://peps.python.org/pep-0681/]. Patch by Jelle Zijlstra.

- gh-91832 [https://github.com/python/cpython/issues/91832]: Add `required` attribute to **`argparse.Action`** repr output.

- gh-91827 [https://github.com/python/cpython/issues/91827]: In the **`tkinter`** module add method `info_patchlevel()` which returns the exact version of the Tcl library as a named tuple similar to **`sys.version_info`**.

- gh-84461 [https://github.com/python/cpython/issues/84461]: Add **`--enable-wasm-pthreads`** to enable pthreads support for WASM builds. `Emscripten/node` no longer has threading enabled by default. Include additional file systems.

- gh-91821 [https://github.com/python/cpython/issues/91821]: Fix unstable `test_from_tuple` test in `test_decimal.py`.

- gh-91217 [https://github.com/python/cpython/issues/91217]: Deprecate the xdrlib module.

- gh-91217 [https://github.com/python/cpython/issues/91217]: Deprecate the uu module.

- gh-91760 [https://github.com/python/cpython/issues/91760]: More strict rules will be applied for numerical group references and group names in regular expressions. For now, a deprecation warning is emitted for group references and group names which will be errors in future Python versions.

- gh-84461 [https://github.com/python/cpython/issues/84461]: Add provisional **`sys._emscripten_info`** named tuple with build-time and run-time information about Emscripten platform.

- gh-90623 [https://github.com/python/cpython/issues/90623]: **`signal.raise_signal()`** and **`os.kill()`** now check immediately for pending signals. Patch by Victor Stinner.

- gh-91734 [https://github.com/python/cpython/issues/91734]: Fix OSS audio support on Solaris.

- gh-90633 [https://github.com/python/cpython/issues/90633]: Include the passed value in the exception thrown by `typing.assert_never()`. Patch by Jelle Zijlstra.

- gh-91700 [https://github.com/python/cpython/issues/91700]: Compilation of regular expression containing a conditional expression `(?(group)...)` now raises an appropriate `re.error` if the group number refers to not defined group. Previously an internal RuntimeError was raised.

- gh-91231 [https://github.com/python/cpython/issues/91231]: Add an optional keyword *shutdown_timeout* parameter to the `multiprocessing.BaseManager` constructor. Kill the process if terminate() takes longer than the timeout. Patch by Victor Stinner.

- gh-91621 [https://github.com/python/cpython/issues/91621]: Fix `typing.get_type_hints()` for `collections.abc.Callable`. Patch by Shantanu Jain.

- gh-90568 [https://github.com/python/cpython/issues/90568]: Parsing `\N` escapes of Unicode Named Character Sequences in a `regular expression` raises now `re.error` instead of `TypeError`.

- gh-91670 [https://github.com/python/cpython/issues/91670]: Remove deprecated `SO` config variable in `sysconfig`.

- gh-91217 [https://github.com/python/cpython/issues/91217]: Deprecate the telnetlib module.

- gh-91217 [https://github.com/python/cpython/issues/91217]: Deprecate the sunau module.

- gh-91217 [https://github.com/python/cpython/issues/91217]: Deprecate the spwd module.

- gh-91217 [https://github.com/python/cpython/issues/91217]: Deprecate the sndhdr module, as well as inline needed functionality for `email.mime.MIMEAudio`.

- gh-91616 [https://github.com/python/cpython/issues/91616]: `re` module, fix `fullmatch()` mismatch when using Atomic Grouping or Possessive Quantifiers.

- gh-91217 [https://github.com/python/cpython/issues/91217]: Deprecate the “pipes” module.
- gh-91217 [https://github.com/python/cpython/issues/91217]: Deprecate the ossaudiodev module.
- bpo-47256 [https://bugs.python.org/issue?@action=redirect&bpo=47256]: `re` module, limit the maximum capturing group to 1,073,741,823 in 64-bit build, this increases the depth of backtracking.
- gh-91217 [https://github.com/python/cpython/issues/91217]: Deprecate the nis module.
- gh-91595 [https://github.com/python/cpython/issues/91595]: Fix the comparison of character and integer inside **`Tools.gdb.libpython.write_repr()`**. Patch by Yu Liu.
- gh-74166 [https://github.com/python/cpython/issues/74166]: Add option to raise all errors from `create_connection()` in an `ExceptionGroup` when it fails to create a connection. The default remains to raise only the last error that had occurred when multiple addresses were tried.
- gh-91487 [https://github.com/python/cpython/issues/91487]: Optimize asyncio UDP speed, over 100 times faster when transferring a large file.
- gh-91575 [https://github.com/python/cpython/issues/91575]: Update case-insensitive matching in the `re` module to the latest Unicode version.
- gh-90622 [https://github.com/python/cpython/issues/90622]: In `concurrent.futures.process.ProcessPoolExecutor` disallow the «fork» multiprocessing start method when the new `max_tasks_per_child` feature is used as the mix of threads+fork can hang the child processes. Default to using the safe «spawn» start method in that circumstance if no

`mp_context` was supplied.

- gh-89022 [https://github.com/python/cpython/issues/89022]: In **`sqlite3`**, `SQLITE_MISUSE` result codes are now mapped to **`InterfaceError`** instead of **`ProgrammingError`**. Also, more accurate exceptions are raised when binding parameters fail. Patch by Erlend E. Aasland.

- gh-91526 [https://github.com/python/cpython/issues/91526]: Stop calling `os.device_encoding(file.fileno())` in **`TextIOWrapper`**. It was complex, never documented, and didn't work for most cases. (Patch by Inada Naoki.)

- gh-88116 [https://github.com/python/cpython/issues/88116]: Change the frame-related functions in the **`inspect`** module to return a regular object (that is backwards compatible with the old tuple-like interface) that include the extended **PEP 657** [https://peps.python.org/pep-0657/] position information (end line number, column and end column). The affected functions are: **`inspect.getframeinfo()`**, **`inspect.getouterframes()`**, **`inspect.getinnerframes()`**, **`inspect.stack()`** and **`inspect.trace()`**. Patch by Pablo Galindo.

- gh-69093 [https://github.com/python/cpython/issues/69093]: Add indexing and slicing support to **`sqlite3.Blob`**. Patch by Aviv Palivoda and Erlend E. Aasland.

- gh-69093 [https://github.com/python/cpython/issues/69093]: Add context manager support to **`sqlite3.Blob`**. Patch by Aviv Palivoda and Erlend E. Aasland.

- gh-91217 [https://github.com/python/cpython/issues/91217]: Deprecate nntplib.

- gh-91217 [https://github.com/python/cpython/issues/91217]: Deprecate msilib.

- gh-91404 [https://github.com/python/cpython/issues/91404]: Improve the performance of **`re`** matching by using computed gotos (or «threaded code») on supported platforms and removing expensive pointer indirections.

- gh-91217 [https://github.com/python/cpython/issues/91217]: Deprecate the imghdr module.
- gh-91217 [https://github.com/python/cpython/issues/91217]: Deprecate the crypt module.
- gh-91276 [https://github.com/python/cpython/issues/91276]: Make space for longer opcodes in `dis` output.
- bpo-47000 [https://bugs.python.org/issue?@action=redirect&bpo=47000]: Make `TextIOWrapper` uses locale encoding when `encoding="locale"` is specified even in UTF-8 mode.
- gh-91230 [https://github.com/python/cpython/issues/91230]: `warnings.catch_warnings()` now accepts arguments for `warnings.simplefilter()`, providing a more concise way to locally ignore warnings or convert them to errors.
- gh-91217 [https://github.com/python/cpython/issues/91217]: Deprecate the chunk module.
- gh-91498 [https://github.com/python/cpython/issues/91498]: Add the `TCP_CONNECTION_INFO` option (available on macOS) to `socket`.
- bpo-47260 [https://bugs.python.org/issue?@action=redirect&bpo=47260]: Fix `os.closerange()` potentially being a no-op in a Linux seccomp sandbox.
- bpo-47087 [https://bugs.python.org/issue?@action=redirect&bpo=47087]: Implement `typing.Required` and `typing.NotRequired` (**PEP 655** [https://peps.python.org/pep-0655/]). Patch by David Foster and Jelle Zijlstra.
- bpo-47061 [https://bugs.python.org/issue?@action=redirect&bpo=47061]: Deprecate cgi and cgitb.
- bpo-47061 [https://bugs.python.org/issue?@action=redirect&bpo=47061]: Deprecate audioop.

- bpo-47000 [https://bugs.python.org/issue?@action=redirect&bpo=47000]: Add `locale.getencoding()` to get the current locale encoding. It is similar to `locale.getpreferredencoding(False)` but ignores the Python UTF-8 Mode.

- bpo-42012 [https://bugs.python.org/issue?@action=redirect&bpo=42012]: Add `wsgiref.types`, containing WSGI-specific types for static type checking.

- bpo-47227 [https://bugs.python.org/issue?@action=redirect&bpo=47227]: Suppress expression chaining for more `re` parsing errors.

- bpo-47211 [https://bugs.python.org/issue?@action=redirect&bpo=47211]: Remove undocumented and never working function `re.template()` and flag `re.TEMPLATE`. This was later reverted in 3.11.0b2 and deprecated instead.

- bpo-47135 [https://bugs.python.org/issue?@action=redirect&bpo=47135]: `decimal.localcontext()` now accepts context attributes via keyword arguments

- bpo-43323 [https://bugs.python.org/issue?@action=redirect&bpo=43323]: Fix errors in the `email` module if the charset itself contains undecodable/unencodable characters.

- bpo-46841 [https://bugs.python.org/issue?@action=redirect&bpo=46841]: Disassembly of quickened code.

- bpo-46681 [https://bugs.python.org/issue?@action=redirect&bpo=46681]: Forward gzip.compress() compresslevel to zlib.

- bpo-45100 [https://bugs.python.org/issue?@action=redirect&bpo=45100]: Add `typing.get_overloads()` and `typing.clear_overloads()`. Patch by Jelle Zijlstra.

- bpo-44807 [https://bugs.python.org/issue?

@action=redirect&bpo=44807]: `typing.Protocol` no longer silently replaces `__init__()` methods defined on subclasses. Patch by Adrian Garcia Badaracco.

- bpo-46787 [https://bugs.python.org/issue?@action=redirect&bpo=46787]: Fix `concurrent.futures.ProcessPoolExecutor` exception memory leak

- bpo-46720 [https://bugs.python.org/issue?@action=redirect&bpo=46720]: Add support for path-like objects to `multiprocessing.set_executable()` for Windows to be on a par with Unix-like systems. Patch by Géry Ogam.

- bpo-46696 [https://bugs.python.org/issue?@action=redirect&bpo=46696]: Add `SO_INCOMING_CPU` constant to `socket`.

- bpo-46053 [https://bugs.python.org/issue?@action=redirect&bpo=46053]: Fix OSS audio support on NetBSD.

- bpo-45639 [https://bugs.python.org/issue?@action=redirect&bpo=45639]: `image/avif` and `image/webp` were added to `mimetypes`.

- bpo-46285 [https://bugs.python.org/issue?@action=redirect&bpo=46285]: Add command-line option `-p/--protocol` to module `http.server` which specifies the HTTP version to which the server is conformant (HTTP/1.1 conformant servers can now be run from the command-line interface of module `http.server`). Patch by Géry Ogam.

- bpo-44791 [https://bugs.python.org/issue?@action=redirect&bpo=44791]: Accept ellipsis as the last argument of `typing.Concatenate`.

- bpo-46547 [https://bugs.python.org/issue?@action=redirect&bpo=46547]: Remove variables leaking into `pydoc.Helper` class namespace.

- bpo-46415 [https://bugs.python.org/issue?

@action=redirect&bpo=46415]: Fix ipaddress.ip_{address,interface,network} raising TypeError instead of ValueError if given invalid tuple as address parameter.

- bpo-46075 [https://bugs.python.org/issue?@action=redirect&bpo=46075]: `CookieJar` with `DefaultCookiePolicy` now can process cookies from localhost with domain=localhost explicitly specified in Set-Cookie header.

- bpo-45995 [https://bugs.python.org/issue?@action=redirect&bpo=45995]: Add a «z» option to the string formatting specification that coerces negative zero floating-point values to positive zero after rounding to the format precision. Contributed by John Belmonte.

- bpo-26175 [https://bugs.python.org/issue?@action=redirect&bpo=26175]: Fully implement the **`io.BufferedIOBase`** or **`io.TextIOBase`** interface for **`tempfile.SpooledTemporaryFile`** objects. This lets them work correctly with higher-level layers (like compression modules). Patch by Carey Metcalfe.

- bpo-45138 [https://bugs.python.org/issue?@action=redirect&bpo=45138]: Fix a regression in the **`sqlite3`** trace callback where bound parameters were not expanded in the passed statement string. The regression was introduced in Python 3.10 by bpo-40318 [https://bugs.python.org/issue?@action=redirect&bpo=40318]. Patch by Erlend E. Aasland.

- bpo-44863 [https://bugs.python.org/issue?@action=redirect&bpo=44863]: Allow **`TypedDict`** subclasses to also include **`Generic`** as a base class in class based syntax. Thereby allowing the user to define a generic `TypedDict`, just like a user-defined generic but with `TypedDict` semantics.

- bpo-44587 [https://bugs.python.org/issue?@action=redirect&bpo=44587]: Fix BooleanOptionalAction to not automatically add a default string. If a default string is desired, use a formatter to add it.

- bpo-43827 [https://bugs.python.org/issue?@action=redirect&bpo=43827]: All positional-or-keyword parameters to `ABCMeta.__new__` are now positional-only to avoid conflicts with keyword arguments to be passed to **`__init_subclass__()`**.

- bpo-43218 [https://bugs.python.org/issue?@action=redirect&bpo=43218]: Prevent creation of a venv whose path contains the PATH separator. This could affect the usage of the activate script. Patch by Dustin Rodrigues.

- bpo-38435 [https://bugs.python.org/issue?@action=redirect&bpo=38435]: Add a `process_group` parameter to **`subprocess.Popen`** to help move more things off of the unsafe `preexec_fn` parameter.

- bpo-42066 [https://bugs.python.org/issue?@action=redirect&bpo=42066]: Fix cookies getting sorted in **`CookieJar.__iter__()`** which is an extra behavior and not mentioned in RFC 2965 or Netscape cookie protocol. Now the cookies in `CookieJar` follows the order of the `Set-Cookie` header. Patch by Iman Kermani.

- bpo-40617 [https://bugs.python.org/issue?@action=redirect&bpo=40617]: Add **`create_window_function()`** to **`sqlite3.Connection`** for creating aggregate window functions. Patch by Erlend E. Aasland.

- bpo-40676 [https://bugs.python.org/issue?@action=redirect&bpo=40676]: Convert **`csv`** to use Argument Clinic for **`csv.field_size_limit()`**, **`csv.get_dialect()`**, **`csv.unregister_dialect()`** and **`csv.list_dialects()`**.

- bpo-39716 [https://bugs.python.org/issue?@action=redirect&bpo=39716]: Raise an `ArgumentError` when the same subparser name is added twice to an `argparse.ArgumentParser`. This is consistent with the (default) behavior when the same option string is added twice to an `ArgumentParser`.

- bpo-36073 [https://bugs.python.org/issue?@action=redirect&bpo=36073]: Raise **`ProgrammingError`** instead of segfaulting on recursive usage of cursors in **`sqlite3`** converters. Patch by Sergey Fedoseev.

- bpo-34975 [https://bugs.python.org/issue?@action=redirect&bpo=34975]: Adds a `start_tls()` method to **`StreamWriter`**, which upgrades the connection with TLS using the given **`SSLContext`**.

- bpo-22276 [https://bugs.python.org/issue?@action=redirect&bpo=22276]: **`Path`** methods **`glob()`** and **`rglob()`** return only directories if *pattern* ends with a pathname components separator (/ or **`sep`**). Patch by Eisuke Kawashima.

- bpo-24905 [https://bugs.python.org/issue?@action=redirect&bpo=24905]: Add **`blobopen()`** to **`sqlite3.Connection`**. **`sqlite3.Blob`** allows incremental I/O operations on blobs. Patch by Aviv Palivoda and Erlend E. Aasland.

Documentation

- gh-91888 [https://github.com/python/cpython/issues/91888]: Add a new `gh` role to the documentation to link to GitHub issues.
- gh-91783 [https://github.com/python/cpython/issues/91783]: Document security issues concerning the use of the function **`shutil.unpack_archive()`**
- gh-91547 [https://github.com/python/cpython/issues/91547]: Remove «Undocumented modules» page.
- gh-91298 [https://github.com/python/cpython/issues/91298]: In `importlib.resources.abc`, refined the documentation of the Traversable Protocol, applying changes from importlib_resources 5.7.1.
- bpo-44347 [https://bugs.python.org/issue?@action=redirect&bpo=44347]: Clarify the meaning of *dirs_exist_ok*, a kwarg of **`shutil.copytree()`**.

- bpo-36329 [https://bugs.python.org/issue?@action=redirect&bpo=36329]: Remove “make -C Doc serve” in favour of “make -C Doc htmlview”
- bpo-47189 [https://bugs.python.org/issue?@action=redirect&bpo=47189]: Add a What’s New in Python 3.11 entry for the Faster CPython project. Documentation by Ken Jin and Kumar Aditya.
- bpo-38668 [https://bugs.python.org/issue?@action=redirect&bpo=38668]: Update the introduction to documentation for `os.path` to remove warnings that became irrelevant after the implementations of **PEP 383** [https://peps.python.org/pep-0383/] and **PEP 529** [https://peps.python.org/pep-0529/].
- bpo-47115 [https://bugs.python.org/issue?@action=redirect&bpo=47115]: The documentation now lists which members of C structs are part of the Limited API/Stable ABI.
- bpo-46962 [https://bugs.python.org/issue?@action=redirect&bpo=46962]: All docstrings in code snippets are now wrapped into `PyDoc_STR` to follow the guideline of **PEP 7’s Documentation Strings paragraph** [https://peps.python.org/pep-0007/#documentation-strings]. Patch by Oleg Iarygin.
- bpo-26792 [https://bugs.python.org/issue?@action=redirect&bpo=26792]: Improve the docstrings of `runpy.run_module()` and `runpy.run_path()`. Original patch by Andrew Brezovsky.

Tests

- gh-92169 [https://github.com/python/cpython/issues/92169]: Use `warnings_helper.import_deprecated()` to import deprecated modules uniformly in tests. Patch by Hugo van Kemenade.
- gh-84461 [https://github.com/python/cpython/issues/84461]: When multiprocessing is enabled, libregrtest can now use a Python executable other than `sys.executable` via the `--python` flag.
- gh-91904 [https://github.com/python/cpython/issues/91904]: Fix initialization of `PYTHONREGRTEST_UNICODE_GUARD` which

prevented running regression tests on non-UTF-8 locale.

- gh-91752 [https://github.com/python/cpython/issues/91752]: Added @requires_zlib to test.test_tools.test_freeze.TestFreeze.
- gh-91607 [https://github.com/python/cpython/issues/91607]: Fix `test_concurrent_futures` to test the correct multiprocessing start method context in several cases where the test logic mixed this up.
- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: Threading tests are now skipped on WASM targets without pthread support.
- bpo-47109 [https://bugs.python.org/issue?@action=redirect&bpo=47109]: Test for **`ctypes.macholib.dyld`**, **`ctypes.macholib.dylib`**, and **`ctypes.macholib.framework`** are brought from manual pre-**`unittest`** times to **`ctypes.test`** location and structure. Patch by Oleg Iarygin.
- bpo-29890 [https://bugs.python.org/issue?@action=redirect&bpo=29890]: Add tests for **`ipaddress.IPv4Interface`** and **`ipaddress.IPv6Interface`** construction with tuple arguments. Original patch and tests by louisom.

Build

- gh-89452 [https://github.com/python/cpython/issues/89452]: gdbm-compat is now preferred over ndbm if both are available on the system. This allows avoiding the problematic ndbm.h on macOS.
- gh-91731 [https://github.com/python/cpython/issues/91731]: Python is now built with `-std=c11` compiler option, rather than `-std=c99`. Patch by Victor Stinner.
- bpo-47152 [https://bugs.python.org/issue?@action=redirect&bpo=47152]: Add script and make target for generating `sre_constants.h`.
- bpo-47103 [https://bugs.python.org/issue?@action=redirect&bpo=47103]: Windows `PGInstrument` builds now copy a required DLL into the output directory, making it easier to run the profile stage of a PGO build.

Windows

- bpo-46907 [https://bugs.python.org/issue?@action=redirect&bpo=46907]: Update Windows installer to use SQLite 3.38.3.
- bpo-47239 [https://bugs.python.org/issue?@action=redirect&bpo=47239]: Fixed –list and –list-paths output for Python install manager when used in an active virtual environment.
- bpo-46907 [https://bugs.python.org/issue?@action=redirect&bpo=46907]: Update Windows installer to use SQLite 3.38.2.
- bpo-46785 [https://bugs.python.org/issue?@action=redirect&bpo=46785]: Fix race condition between **`os.stat()`** and unlinking a file on Windows, by using errors codes returned by `FindFirstFileW()` when appropriate in `win32_xstat_impl`.
- bpo-40859 [https://bugs.python.org/issue?@action=redirect&bpo=40859]: Update Windows build to use xz-5.2.5

macOS

- bpo-46907 [https://bugs.python.org/issue?@action=redirect&bpo=46907]: Update macOS installer to SQLite 3.38.4.

Tools/Demos

- gh-91583 [https://github.com/python/cpython/issues/91583]: Fix regression in the code generated by Argument Clinic for functions with the `defining_class` parameter.
- gh-91575 [https://github.com/python/cpython/issues/91575]: Add script `Tools/scripts/generate_re_casefix.py` and the make target `regen-re` for generating additional data for case-insensitive matching according to the current Unicode version.
- gh-91551 [https://github.com/python/cpython/issues/91551]: Remove the ancient Pynche color editor. It has moved to https://gitlab.com/warsaw/pynche

C API

- gh-88279 [https://github.com/python/cpython/issues/88279]: Deprecate the C functions: **`PySys_SetArgv()`**, **`PySys_SetArgvEx()`**, **`PySys_SetPath()`**. Patch by Victor Stinner.

- gh-92154 [https://github.com/python/cpython/issues/92154]: Added the **`PyCode_GetCode()`** function. This function does the equivalent of the Python code `getattr(code_object, 'co_code')`.

- gh-92173 [https://github.com/python/cpython/issues/92173]: Fix the `closure` argument to **`PyEval_EvalCodeEx()`**.

- gh-91320 [https://github.com/python/cpython/issues/91320]: Fix C++ compiler warnings about «old-style cast» (`g++ -Wold-style-cast`) in the Python C API. Use C++ `reinterpret_cast<>` and `static_cast<>` casts when the Python C API is used in C++. Patch by Victor Stinner.

- gh-80527 [https://github.com/python/cpython/issues/80527]: Mark functions as deprecated by **PEP 623** [https://peps.python.org/pep-0623/]: **`PyUnicode_AS_DATA()`**, **`PyUnicode_AS_UNICODE()`**, **`PyUnicode_GET_DATA_SIZE()`**, **`PyUnicode_GET_SIZE()`**. Patch by Victor Stinner.

- gh-91768 [https://github.com/python/cpython/issues/91768]: **`Py_REFCNT()`**, **`Py_TYPE()`**, **`Py_SIZE()`** and **`Py_IS_TYPE()`** functions argument type is now `PyObject*`, rather than `const PyObject*`. Patch by Victor Stinner.

- gh-91020 [https://github.com/python/cpython/issues/91020]: Add `PyBytes_Type.tp_alloc` to initialize `PyBytesObject.ob_shash` for bytes subclasses.

- bpo-40421 [https://bugs.python.org/issue?@action=redirect&bpo=40421]: Add `PyFrame_GetLasti` C-API function to access frame object's `f_lasti` attribute safely from C code.

- bpo-35134 [https://bugs.python.org/issue?@action=redirect&bpo=35134]: Remove the `Include/code.h` header file. C extensions should only include the main `<Python.h>` header file. Patch by Victor Stinner.

- bpo-47169 [https://bugs.python.org/issue?@action=redirect&bpo=47169]: **`PyOS_CheckStack()`** is now exported in the Stable ABI on Windows.

- bpo-47169 [https://bugs.python.org/issue?@action=redirect&bpo=47169]: **`PyThread_get_thread_native_id()`** is excluded from the stable ABI on platforms where it doesn't exist (like Solaris).

- bpo-46343 [https://bugs.python.org/issue?@action=redirect&bpo=46343]: Added **`PyErr_GetHandledException()`** and **`PyErr_SetHandledException()`** as simpler alternatives to **`PyErr_GetExcInfo()`** and **`PyErr_SetExcInfo()`**.

  They are included in the stable ABI.

Python 3.11.0 alpha 7

Release date: 2022-04-05

Core and Builtins

- bpo-47212 [https://bugs.python.org/issue?@action=redirect&bpo=47212]: Raise **`IndentationError`** instead of **`SyntaxError`** for a bare `except` with no following indent. Improve **`SyntaxError`** locations for an un-parenthesized generator used as arguments. Patch by Matthieu Dartiailh.
- bpo-47186 [https://bugs.python.org/issue?@action=redirect&bpo=47186]: Replace **`JUMP_IF_NOT_EG_MATCH`** by **`CHECK_EG_MATCH`** + jump.
- bpo-47176 [https://bugs.python.org/issue?@action=redirect&bpo=47176]: Emscripten builds cannot handle

signals in the usual way due to platform limitations. Python can now handle signals. To use, set Module.Py_EmscriptenSignalBuffer to be a single byte SharedArrayBuffer and set Py_EMSCRIPTEN_SIGNAL_HANDLING to 1. Writing a number into the SharedArrayBuffer will cause the corresponding signal to be raised into the Python thread.

- bpo-47186 [https://bugs.python.org/issue?@action=redirect&bpo=47186]: Replace `JUMP_IF_NOT_EXC_MATCH` by `CHECK_EXC_MATCH` + jump.
- bpo-47120 [https://bugs.python.org/issue?@action=redirect&bpo=47120]: Replace the absolute jump opcode `JUMP_NO_INTERRUPT` by the relative `JUMP_BACKWARD_NO_INTERRUPT`.
- bpo-46841 [https://bugs.python.org/issue?@action=redirect&bpo=46841]: Avoid unnecessary allocations when comparing code objects.
- bpo-47182 [https://bugs.python.org/issue?@action=redirect&bpo=47182]: Fix a crash when using a named unicode character like `"\N{digit nine}"` after the main interpreter has been initialized a second time.
- bpo-47162 [https://bugs.python.org/issue?@action=redirect&bpo=47162]: WebAssembly cannot deal with bad function pointer casts (different count or types of arguments). Python can now use call trampolines to mitigate the problem. Define `PY_CALL_TRAMPOLINE` to enable call trampolines.
- bpo-46775 [https://bugs.python.org/issue?@action=redirect&bpo=46775]: Some Windows system error codes(>= 10000) are now mapped into the correct errno and may now raise a subclass of `OSError`. Patch by Donghee Na.
- bpo-47129 [https://bugs.python.org/issue?@action=redirect&bpo=47129]: Improve error messages in f-string syntax errors concerning empty expressions.
- bpo-47117 [https://bugs.python.org/issue?@action=redirect&bpo=47117]: Fix a crash if we fail to decode characters in interactive mode if the tokenizer buffers are uninitialized. Patch by Pablo Galindo.

- bpo-47127 [https://bugs.python.org/issue?@action=redirect&bpo=47127]: Speed up calls to c functions with keyword arguments by 25% with specialization. Patch by Kumar Aditya.
- bpo-47120 [https://bugs.python.org/issue?@action=redirect&bpo=47120]: Replaced **`JUMP_ABSOLUTE`** by the relative jump **`JUMP_BACKWARD`**.
- bpo-42197 [https://bugs.python.org/issue?@action=redirect&bpo=42197]: **`PyFrame_FastToLocalsWithError()`** and **`PyFrame_LocalsToFast()`** are no longer called during profiling nor tracing. C code can access the `f_locals` attribute of **`PyFrameObject`** by calling **`PyFrame_GetLocals()`**.
- bpo-47070 [https://bugs.python.org/issue?@action=redirect&bpo=47070]: Improve performance of `array_inplace_repeat` by reducing the number of invocations of `memcpy`. Refactor the `repeat` and inplace `repeat` methods of `array`, `bytes`, `bytearray` and `unicodeobject` to use the common `_PyBytes_Repeat`.
- bpo-47053 [https://bugs.python.org/issue?@action=redirect&bpo=47053]: Reduce de-optimization in the specialized `BINARY_OP_INPLACE_ADD_UNICODE` opcode.
- bpo-47045 [https://bugs.python.org/issue?@action=redirect&bpo=47045]: Remove the `f_state` field from the _PyInterpreterFrame struct. Add the `owner` field to the _PyInterpreterFrame struct to make ownership explicit to simplify clearing and deallocing frames and generators.

Library

- bpo-46968 [https://bugs.python.org/issue?@action=redirect&bpo=46968]: Check for the existence of the «sys/auxv.h» header in **`faulthandler`** to avoid compilation problems in systems where this header doesn't exist. Patch by Pablo Galindo

Core and Builtins

- bpo-46329 [https://bugs.python.org/issue?

@action=redirect&bpo=46329]: Use low bit of `LOAD_GLOBAL` to indicate whether to push a `NULL` before the global. Helps streamline the call sequence a bit.
- bpo-46841 [https://bugs.python.org/issue?@action=redirect&bpo=46841]: Quicken bytecode in-place by storing it as part of the corresponding `PyCodeObject`.
- bpo-47012 [https://bugs.python.org/issue?@action=redirect&bpo=47012]: Speed up iteration of **`bytes`** and **`bytearray`** by 30%. Patch by Kumar Aditya.
- bpo-47009 [https://bugs.python.org/issue?@action=redirect&bpo=47009]: Improved the performance of **`list.append()`** and list comprehensions by optimizing for the common case, where no resize is needed. Patch by Dennis Sweeney.
- bpo-47005 [https://bugs.python.org/issue?@action=redirect&bpo=47005]: Improve performance of `bytearray_repeat` and `bytearray_irepeat` by reducing the number of invocations of `memcpy`.

Library

- bpo-46829 [https://bugs.python.org/issue?@action=redirect&bpo=46829]: Deprecate passing a message into **`asyncio.Future.cancel()`** and **`asyncio.Task.cancel()`**

Core and Builtins

- bpo-46993 [https://bugs.python.org/issue?@action=redirect&bpo=46993]: Speed up **`bytearray`** creation from **`list`** and **`tuple`** by 40%. Patch by Kumar Aditya.
- bpo-39829 [https://bugs.python.org/issue?@action=redirect&bpo=39829]: Removed the `__len__()` call when initializing a list and moved initializing to `list_extend`. Patch by Jeremiah Pascual.
- bpo-46944 [https://bugs.python.org/issue?@action=redirect&bpo=46944]: Speed up throwing exception in generator with **`METH_FASTCALL`** calling convention. Patch by Kumar Aditya.
- bpo-46841 [https://bugs.python.org/issue?

@action=redirect&bpo=46841]: Modify `STORE_SUBSCR` to use an inline cache entry (rather than its oparg) as an adaptive counter.

- bpo-46841 [https://bugs.python.org/issue?@action=redirect&bpo=46841]: Use inline caching for **PRECALL** and `CALL`, and remove the internal machinery for managing the (now unused) non-inline caches.
- bpo-46881 [https://bugs.python.org/issue?@action=redirect&bpo=46881]: Statically allocate and initialize the latin1 characters.
- bpo-46838 [https://bugs.python.org/issue?@action=redirect&bpo=46838]: Improve syntax errors for incorrect function definitions. Patch by Pablo Galindo
- bpo-43721 [https://bugs.python.org/issue?@action=redirect&bpo=43721]: Fix docstrings of `getter`, `setter`, and `deleter` to clarify that they create a new copy of the property.
- bpo-43224 [https://bugs.python.org/issue?@action=redirect&bpo=43224]: Make grammar changes required for PEP 646.

Library

- bpo-47208 [https://bugs.python.org/issue?@action=redirect&bpo=47208]: Allow vendors to override **CTYPES_MAX_ARGCOUNT**.

- bpo-23689 [https://bugs.python.org/issue?@action=redirect&bpo=23689]: `re` module: fix memory leak when a match is terminated by a signal or memory allocation failure. Patch by Ma Lin.

- bpo-47167 [https://bugs.python.org/issue?@action=redirect&bpo=47167]: Allow overriding a future compliance check in `asyncio.Task`.

- bpo-47151 [https://bugs.python.org/issue?@action=redirect&bpo=47151]: When subprocess tries to use vfork, it now falls back to fork if vfork returns an error. This allows use in situations where vfork isn’t allowed by the OS kernel.

- bpo-47152 [https://bugs.python.org/issue?@action=redirect&bpo=47152]: Convert the `re` module into a package. Deprecate modules `sre_compile`, `sre_constants` and `sre_parse`.

- bpo-4833 [https://bugs.python.org/issue?@action=redirect&bpo=4833]: Add `ZipFile.mkdir()`

- bpo-27929 [https://bugs.python.org/issue?@action=redirect&bpo=27929]: Fix `asyncio.loop.sock_connect()` to only resolve names for `socket.AF_INET` or `socket.AF_INET6` families. Resolution may not make sense for other families, like `socket.AF_BLUETOOTH` and `socket.AF_UNIX`.

- bpo-14265 [https://bugs.python.org/issue?@action=redirect&bpo=14265]: Adds the fully qualified test name to unittest output

- bpo-47061 [https://bugs.python.org/issue?@action=redirect&bpo=47061]: Deprecate the aifc module.

- bpo-39622 [https://bugs.python.org/issue?@action=redirect&bpo=39622]: Handle Ctrl+C in asyncio programs to interrupt the main task.

- bpo-47101 [https://bugs.python.org/issue?@action=redirect&bpo=47101]: `hashlib.algorithms_available` now lists only algorithms that are provided by activated crypto providers on OpenSSL 3.0. Legacy algorithms are not listed unless the legacy provider has been loaded into the default OSSL context.

- bpo-47099 [https://bugs.python.org/issue?@action=redirect&bpo=47099]: All `URLError` exception messages raised in `urllib.request.URLopener` now contain a colon between `ftp error` and the rest of the message. Previously, `open_ftp()` missed the colon. Patch by Oleg Iarygin.

- bpo-47099 [https://bugs.python.org/issue?@action=redirect&bpo=47099]: Exception chaining is changed from **`Exception.with_traceback()`**/`sys.exc_info()` to **PEP 3134** [https://peps.python.org/pep-3134/]. Patch by Oleg Iarygin.
- bpo-47095 [https://bugs.python.org/issue?@action=redirect&bpo=47095]: `hashlib`'s internal `_blake2` module now prefers `libb2` from https://www.blake2.net/ over Python's vendored copy of blake2.
- bpo-47098 [https://bugs.python.org/issue?@action=redirect&bpo=47098]: The Keccak Code Package for `hashlib`'s internal `_sha3` module has been replaced with tiny_sha3. The module is used as fallback when Python is built without OpenSSL.
- bpo-47088 [https://bugs.python.org/issue?@action=redirect&bpo=47088]: Implement `typing.LiteralString`, part of **PEP 675** [https://peps.python.org/pep-0675/]. Patch by Jelle Zijlstra.
- bpo-42885 [https://bugs.python.org/issue?@action=redirect&bpo=42885]: Optimize `re.search()`, `re.split()`, `re.findall()`, `re.finditer()` and `re.sub()` for regular expressions starting with `\A` or `^`.
- bpo-23691 [https://bugs.python.org/issue?@action=redirect&bpo=23691]: Protect the `re.finditer()` iterator from re-entering.
- bpo-47067 [https://bugs.python.org/issue?@action=redirect&bpo=47067]: Optimize calling `GenericAlias` objects by using **PEP 590** [https://peps.python.org/pep-0590/] `vectorcall` and by replacing `PyObject_SetAttrString` with `PyObject_SetAttr`.
- bpo-28080 [https://bugs.python.org/issue?@action=redirect&bpo=28080]: Add the *metadata_encoding*

parameter in the `zipfile.ZipFile` constructor and the `--metadata-encoding` option in the `zipfile` CLI to allow reading zipfiles using non-standard codecs to encode the filenames within the archive.

- bpo-47000 [https://bugs.python.org/issue?@action=redirect&bpo=47000]: Make `io.text_encoding()` returns «utf-8» when UTF-8 mode is enabled.

- bpo-42369 [https://bugs.python.org/issue?@action=redirect&bpo=42369]: Fix thread safety of **`zipfile._SharedFile.tell()`** to avoid a «zipfile.BadZipFile: Bad CRC-32 for file» exception when reading a **`ZipFile`** from multiple threads.

- bpo-38256 [https://bugs.python.org/issue?@action=redirect&bpo=38256]: Fix `binascii.crc32()` when it is compiled to use zlib'c crc32 to work properly on inputs 4+GiB in length instead of returning the wrong result. The workaround prior to this was to always feed the function data in increments smaller than 4GiB or to just call the zlib module function.

  We also have `binascii.crc32()` release the GIL when computing on larger inputs as `zlib.crc32()` and `hashlib` do.

  This also boosts performance on Windows as it now uses the zlib crc32 implementation for `binascii.crc32()` for a 2-3x speedup.

  That the stdlib has a crc32 API in two modules is a known historical oddity. This moves us closer to a single implementation behind them.

- bpo-47066 [https://bugs.python.org/issue?@action=redirect&bpo=47066]: Global inline flags (e.g. `(?i)`) can now only be used at the start of the regular expressions. Using them not at the start of expression was deprecated since Python 3.6.

- bpo-39394 [https://bugs.python.org/issue?@action=redirect&bpo=39394]: A warning about inline flags not at the start of the regular expression now contains the position of the flag.

- bpo-433030 [https://bugs.python.org/issue?@action=redirect&bpo=433030]: Add support of atomic grouping (`(?>...)`) and possessive quantifiers (`*+, ++, ?+, {m,n}+`) in **`regular expressions`**.

- bpo-47062 [https://bugs.python.org/issue?@action=redirect&bpo=47062]: Implement **`asyncio.Runner`** context manager.

- bpo-46382 [https://bugs.python.org/issue?@action=redirect&bpo=46382]: **`dataclass()`** `slots=True` now correctly omits slots already defined in base classes. Patch by Arie Bovenberg.

- bpo-47057 [https://bugs.python.org/issue?@action=redirect&bpo=47057]: Use FASTCALL convention for `FutureIter.throw()`

- bpo-47061 [https://bugs.python.org/issue?@action=redirect&bpo=47061]: Deprecate the various modules listed by **PEP 594** [https://peps.python.org/pep-0594/]:

  aifc, asynchat, asyncore, audioop, cgi, cgitb, chunk, crypt, imghdr, msilib, nntplib, nis, ossaudiodev, pipes, smtpd, sndhdr, spwd, sunau, telnetlib, uu, xdrlib

- bpo-34790 [https://bugs.python.org/issue?@action=redirect&bpo=34790]: Remove passing coroutine objects to **`asyncio.wait()`**.

- bpo-47039 [https://bugs.python.org/issue?@action=redirect&bpo=47039]: Normalize `repr()` of asyncio future and task objects.

- bpo-2604 [https://bugs.python.org/issue?@action=redirect&bpo=2604]: Fix bug where doctests using globals would fail when run multiple times.

- bpo-45150 [https://bugs.python.org/issue?@action=redirect&bpo=45150]: Add `hashlib.file_digest()` helper for efficient hashing of file object.

- bpo-34861 [https://bugs.python.org/issue?@action=redirect&bpo=34861]: Made cumtime the default sorting key for cProfile

- bpo-45997 [https://bugs.python.org/issue?@action=redirect&bpo=45997]: Fix `asyncio.Semaphore` re-aquiring FIFO order.

- bpo-47022 [https://bugs.python.org/issue?@action=redirect&bpo=47022]: The **`asynchat`**, **`asyncore`** and **`smtpd`** modules have been deprecated since at least Python 3.6. Their documentation and deprecation warnings and have now been updated to note they will removed in Python 3.12 (**PEP 594** [https://peps.python.org/pep-0594/]).

- bpo-43253 [https://bugs.python.org/issue?@action=redirect&bpo=43253]: Fix a crash when closing transports where the underlying socket handle is already invalid on the Proactor event loop.

- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: `select.select()` now passes `NULL` to `select` for each empty fdset.

- bpo-47004 [https://bugs.python.org/issue?@action=redirect&bpo=47004]: Apply bugfixes from importlib_metadata 4.11.3, including bugfix for EntryPoint.extras, which was returning match objects and not the extras strings.

- bpo-46998 [https://bugs.python.org/issue?@action=redirect&bpo=46998]: Allow subclassing of `typing.Any`. Patch by Shantanu Jain.

- bpo-46995 [https://bugs.python.org/issue?

@action=redirect&bpo=46995]: Deprecate missing **`asyncio.Task.set_name()`** for third-party task implementations, schedule making it mandatory in Python 3.13.

- bpo-46994 [https://bugs.python.org/issue?@action=redirect&bpo=46994]: Accept explicit contextvars.Context in **`asyncio.create_task()`** and **`asyncio.loop.create_task()`**.

- bpo-46981 [https://bugs.python.org/issue?@action=redirect&bpo=46981]: `typing.get_args(typing.Tuple[()])` now returns `()` instead of `((),)`.

- bpo-46968 [https://bugs.python.org/issue?@action=redirect&bpo=46968]: Add `os.sysconf_names['SC_MINSIGSTKSZ']`.

- bpo-46985 [https://bugs.python.org/issue?@action=redirect&bpo=46985]: Upgrade pip wheel bundled with ensurepip (pip 22.0.4)

- bpo-46968 [https://bugs.python.org/issue?@action=redirect&bpo=46968]: **`faulthandler`**: On Linux 5.14 and newer, dynamically determine size of signal handler stack size CPython allocates using `getauxval(AT_MINSIGSTKSZ)`. This changes allows for Python extension's request to Linux kernel to use AMX_TILE instruction set on Sapphire Rapids Xeon processor to succeed, unblocking use of the ISA in frameworks.

- bpo-46917 [https://bugs.python.org/issue?@action=redirect&bpo=46917]: The **`math.nan`** value is now always available. Patch by Victor Stinner.

- bpo-46955 [https://bugs.python.org/issue?@action=redirect&bpo=46955]: Expose **`asyncio.base_events.Server`** as **`asyncio.Server`**. Patch by Stefan Zabka.

- bpo-23325 [https://bugs.python.org/issue?@action=redirect&bpo=23325]: The `signal` module no longer assumes that `SIG_IGN` and `SIG_DFL` are small int singletons.

- bpo-46932 [https://bugs.python.org/issue?@action=redirect&bpo=46932]: Update bundled libexpat to 2.4.7

- bpo-46933 [https://bugs.python.org/issue?@action=redirect&bpo=46933]: The `pwd` module is now optional. `os.path.expanduser()` returns the path when the `pwd` module is not available.

- bpo-40059 [https://bugs.python.org/issue?@action=redirect&bpo=40059]: **PEP 680** [https://peps.python.org/pep-0680/], the `tomllib` module. Adds support for parsing TOML.

- bpo-464471 [https://bugs.python.org/issue?@action=redirect&bpo=464471]: `asyncio.timeout()` and `asyncio.timeout_at()` context managers added. Patch by Tin Tvrtković and Andrew Svetlov.

- bpo-46805 [https://bugs.python.org/issue?@action=redirect&bpo=46805]: Added raw datagram socket functions for asyncio: `sock_sendto()`, `sock_recvfrom()` and `sock_recvfrom_into()`.

- bpo-46644 [https://bugs.python.org/issue?@action=redirect&bpo=46644]: No longer require valid typeforms to be callable. This allows `typing.Annotated` to wrap `typing.ParamSpecArgs` and `dataclasses.InitVar`. Patch by Gregory Beauregard.

- bpo-46581 [https://bugs.python.org/issue?@action=redirect&bpo=46581]: Brings `ParamSpec` propagation for `GenericAlias` in line with `Concatenate` (and others).

- bpo-45413 [https://bugs.python.org/issue?@action=redirect&bpo=45413]: Define *posix_venv* and *nt_venv* sysconfig installation schemes to be used for bootstrapping new virtual environments. Add *venv* sysconfig installation scheme to get the appropriate one of the above. The schemes are identical to the pre-existing *posix_prefix* and *nt* install schemes. The `venv` module now uses the *venv* scheme to create new virtual environments instead of hardcoding the paths depending only on the platform. Downstream Python distributors customizing the *posix_prefix* or *nt* install scheme in a way that is not compatible with the install scheme used in virtual environments are encouraged not to customize the *venv* schemes. When Python itself runs in a virtual environment, `sysconfig.get_default_scheme()` and `sysconfig.get_preferred_scheme()` with `key="prefix"` returns *venv*.

- bpo-43224 [https://bugs.python.org/issue?@action=redirect&bpo=43224]: Implement support for PEP 646 in typing.py.

- bpo-43224 [https://bugs.python.org/issue?@action=redirect&bpo=43224]: Allow unpacking types.GenericAlias objects, e.g. `*tuple[int, str]`.

- bpo-46557 [https://bugs.python.org/issue?@action=redirect&bpo=46557]: Warnings captured by the logging module are now logged without a format string to prevent systems that group logs by the msg argument from grouping captured warnings together.

- bpo-41370 [https://bugs.python.org/issue?@action=redirect&bpo=41370]: `typing.get_type_hints()` now supports evaluating strings as forward references in PEP 585 generic aliases.

- bpo-46607 [https://bugs.python.org/issue?@action=redirect&bpo=46607]: Add `DeprecationWarning` to **`LegacyInterpolation`**, deprecated in the docstring since Python 3.2. Will be removed in Python 3.13. Use **`BasicInterpolation`** or **`ExtendedInterpolation`** instead.

- bpo-26120 [https://bugs.python.org/issue?@action=redirect&bpo=26120]: `pydoc` now excludes __future__ imports from the module's data items.

- bpo-46480 [https://bugs.python.org/issue?@action=redirect&bpo=46480]: Add `typing.assert_type()`. Patch by Jelle Zijlstra.

- bpo-46421 [https://bugs.python.org/issue?@action=redirect&bpo=46421]: Fix a unittest issue where if the command was invoked as `python -m unittest` and the filename(s) began with a dot (.), a `ValueError` is returned.

- bpo-46245 [https://bugs.python.org/issue?@action=redirect&bpo=46245]: Add optional parameter *dir_fd* in `shutil.rmtree()`.

- bpo-22859 [https://bugs.python.org/issue?@action=redirect&bpo=22859]: `unittest.TestProgram.usageExit()` is marked as deprecated, to be removed in Python 3.13.

- bpo-46170 [https://bugs.python.org/issue?@action=redirect&bpo=46170]: Improve the error message when you try to subclass an instance of `typing.NewType`.

- bpo-40296 [https://bugs.python.org/issue?@action=redirect&bpo=40296]: Fix supporting generic aliases in `pydoc`.

- bpo-20392 [https://bugs.python.org/issue?@action=redirect&bpo=20392]: Fix inconsistency with uppercase file extensions in `MimeTypes.guess_type()`. Patch by Kumar Aditya.

- bpo-46030 [https://bugs.python.org/issue?@action=redirect&bpo=46030]: Add `LOCAL_CREDS`, `LOCAL_CREDS_PERSISTENT` and `SCM_CREDS2` FreeBSD constants to the socket module.

- bpo-44439 [https://bugs.python.org/issue?

@action=redirect&bpo=44439]: Fix `.write()` method of a member file in `ZipFile`, when the input data is an object that supports the buffer protocol, the file length may be wrong.

- bpo-45171 [https://bugs.python.org/issue?@action=redirect&bpo=45171]: Fix handling of the `stacklevel` argument to logging functions in the **`logging`** module so that it is consistent across all logging functions and, as advertised, similar to the `stacklevel` argument used in **`warn()`**.

- bpo-24959 [https://bugs.python.org/issue?@action=redirect&bpo=24959]: Fix bug where **`unittest`** sometimes drops frames from tracebacks of exceptions raised in tests.

- bpo-44859 [https://bugs.python.org/issue?@action=redirect&bpo=44859]: Raise more accurate and **PEP 249** [https://peps.python.org/pep-0249/] compatible exceptions in **`sqlite3`**.

    - Raise **`InterfaceError`** instead of **`ProgrammingError`** for `SQLITE_MISUSE` errors.
    - Don't overwrite **`BufferError`** with **`ValueError`** when conversion to BLOB fails.
    - Raise **`ProgrammingError`** instead of **`Warning`** if user tries to **`execute()`** more than one SQL statement.
    - Raise **`ProgrammingError`** instead of **`ValueError`** if an SQL query contains null characters.

- bpo-44493 [https://bugs.python.org/issue?@action=redirect&bpo=44493]: Add missing terminated NUL in sockaddr_un's length

  This was potentially observable when using non-abstract AF_UNIX datagram sockets to processes written in another programming language.

- bpo-41930 [https://bugs.python.org/issue?

@action=redirect&bpo=41930]: Add `serialize()` and `deserialize()` support to `sqlite3`. Patch by Erlend E. Aasland.

- bpo-33178 [https://bugs.python.org/issue?@action=redirect&bpo=33178]: Added `ctypes.BigEndianUnion` and `ctypes.LittleEndianUnion` classes, as originally documented in the library docs but not yet implemented.

- bpo-43352 [https://bugs.python.org/issue?@action=redirect&bpo=43352]: Add an Barrier object in synchronization primitives of *asyncio* Lib in order to be consistent with Barrier from *threading* and *multiprocessing* libs*

- bpo-35859 [https://bugs.python.org/issue?@action=redirect&bpo=35859]: `re` module, fix a few bugs about capturing group. In rare cases, capturing group gets an incorrect string. Patch by Ma Lin.

Documentation

- bpo-45099 [https://bugs.python.org/issue?@action=redirect&bpo=45099]: Document internal `asyncio` API.
- bpo-47126 [https://bugs.python.org/issue?@action=redirect&bpo=47126]: Update PEP URLs to **PEP 676** [https://peps.python.org/pep-0676/]'s new canonical form.
- bpo-47040 [https://bugs.python.org/issue?@action=redirect&bpo=47040]: Clarified the old Python versions compatibility note of `binascii.crc32()` / `zlib.adler32()` / `zlib.crc32()` functions.
- bpo-46033 [https://bugs.python.org/issue?@action=redirect&bpo=46033]: Clarify `for` statement execution in its doc.
- bpo-45790 [https://bugs.python.org/issue?@action=redirect&bpo=45790]: Adjust inaccurate phrasing in Definición de tipos de extensión: Tutorial about the `ob_base` field and the macros used to access its contents.
- bpo-42340 [https://bugs.python.org/issue?@action=redirect&bpo=42340]: Document that in some

circumstances `KeyboardInterrupt` may cause the code to enter an inconsistent state. Provided a sample workaround to avoid it if needed.

- bpo-41233 [https://bugs.python.org/issue?@action=redirect&bpo=41233]: Link the errnos referenced in `Doc/library/exceptions.rst` to their respective section in `Doc/library/errno.rst`, and vice versa. Previously this was only done for EINTR and InterruptedError. Patch by Yan «yyyyyyyan» Orestes.

Tests

- bpo-47205 [https://bugs.python.org/issue?@action=redirect&bpo=47205]: Skip test for `sched_getaffinity()` and `sched_setaffinity()` error case on FreeBSD.
- bpo-46126 [https://bugs.python.org/issue?@action=redirect&bpo=46126]: Restore "descriptions" when running tests internally.
- bpo-47104 [https://bugs.python.org/issue?@action=redirect&bpo=47104]: Rewrite `asyncio.to_thread()` tests to use `unittest.IsolatedAsyncioTestCase`.
- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: The test suite is now passing on the Emscripten platform. All fork, socket, and subprocess-based tests are skipped.
- bpo-47037 [https://bugs.python.org/issue?@action=redirect&bpo=47037]: Skip `strftime("%4Y")` feature test on Windows. It can cause an assertion error in debug builds.
- bpo-46587 [https://bugs.python.org/issue?@action=redirect&bpo=46587]: Skip tests if platform's `strftime` does not support non-portable glibc extensions.
- bpo-47015 [https://bugs.python.org/issue?@action=redirect&bpo=47015]: A test case for `os.sendfile()` is converted from deprecated **asyncore** (see PEP 594 [https://peps.python.org/pep-0594/]) to `asyncio`. Patch by Oleg Iarygin.

Build

- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: Add configure option **`--enable-wasm-dynamic-linking`** to enable `dlopen` and MAIN_MODULE / SIDE_MODULE on `wasm32-emscripten`.
- bpo-46023 [https://bugs.python.org/issue?@action=redirect&bpo=46023]: `makesetup` now detects and skips all duplicated module definitions. The first entry wins.
- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: Add SOABI `wasm32-emscripten` for Emscripten and `wasm32-wasi` for WASI on 32bit WASM as well as `wasm64` counter parts.
- bpo-47032 [https://bugs.python.org/issue?@action=redirect&bpo=47032]: Ensure Windows install builds fail correctly with a non-zero exit code when part of the build fails.
- bpo-47024 [https://bugs.python.org/issue?@action=redirect&bpo=47024]: Update OpenSSL to 1.1.1n for macOS installers and all Windows builds.
- bpo-46996 [https://bugs.python.org/issue?@action=redirect&bpo=46996]: The **`tkinter`** package now requires Tcl/Tk version 8.5.12 or newer.
- bpo-46973 [https://bugs.python.org/issue?@action=redirect&bpo=46973]: Add `regen-configure` make target to regenerate configure script with Christian’s container image `quay.io/tiran/cpython_autoconf:269`.
- bpo-46917 [https://bugs.python.org/issue?@action=redirect&bpo=46917]: Building Python now requires support of IEEE 754 floating-point numbers. Patch by Victor Stinner.
- bpo-45774 [https://bugs.python.org/issue?@action=redirect&bpo=45774]: `configure` now verifies that all SQLite C APIs needed for the **`sqlite3`** extension module are found.

Windows

- bpo-47194 [https://bugs.python.org/issue?

@action=redirect&bpo=47194]: Update `zlib` to v1.2.12 to resolve **CVE 2018-25032** [https://www.cve.org/CVERecord?id=CVE-2018-25032].

- bpo-47171 [https://bugs.python.org/issue?@action=redirect&bpo=47171]: Enables installing the `py.exe` launcher on Windows ARM64.
- bpo-46566 [https://bugs.python.org/issue?@action=redirect&bpo=46566]: Upgraded Python install manager to support a new `-V:company/tag` argument for full **PEP 514** [https://peps.python.org/pep-0514/] support and to detect ARM64 installs. The `-64` suffix on arguments is deprecated, but still selects any non-32-bit install. Setting **`PYLAUNCHER_ALLOW_INSTALL`** and specifying a version that is not installed will attempt to install the requested version from the Microsoft Store.
- bpo-47086 [https://bugs.python.org/issue?@action=redirect&bpo=47086]: The installer for Windows now includes documentation as loose HTML files rather than a single compiled `.chm` file.
- bpo-46907 [https://bugs.python.org/issue?@action=redirect&bpo=46907]: Update Windows installer to use SQLite 3.38.1.
- bpo-44549 [https://bugs.python.org/issue?@action=redirect&bpo=44549]: Update bzip2 to 1.0.8 in Windows builds to mitigate **CVE 2016-3189** [https://www.cve.org/CVERecord?id=CVE-2016-3189] and **CVE 2019-12900** [https://www.cve.org/CVERecord?id=CVE-2019-12900].
- bpo-46948 [https://bugs.python.org/issue?@action=redirect&bpo=46948]: Prevent **CVE 2022-26488** [https://www.cve.org/CVERecord?id=CVE-2022-26488] by ensuring the Add to PATH option in the Windows installer uses the correct path when being repaired.

macOS

- bpo-46890 [https://bugs.python.org/issue?@action=redirect&bpo=46890]: Fix a regression in the setting of `sys._base_executable` in framework builds, and thereby fix a regression in **`venv`** virtual environments with such builds.

- bpo-46907 [https://bugs.python.org/issue?@action=redirect&bpo=46907]: Update macOS installer to SQLite 3.38.1.

Tools/Demos

- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: Replace Emscripten's limited shell with Katie Bell's browser-ui REPL from python-wasm project.

C API

- bpo-40421 [https://bugs.python.org/issue?@action=redirect&bpo=40421]: Add `PyFrame_GetBuiltins`, `PyFrame_GetGenerator` and `PyFrame_GetGlobals` C-API functions to access frame object attributes safely from C code.

- bpo-46850 [https://bugs.python.org/issue?@action=redirect&bpo=46850]: Move the private `_PyFrameEvalFunction` type, and private `_PyInterpreterState_GetEvalFrameFunc()` and `_PyInterpreterState_SetEvalFrameFunc()` functions to the internal C API. The `_PyFrameEvalFunction` callback function type now uses the `_PyInterpreterFrame` type which is part of the internal C API. Patch by Victor Stinner.

- bpo-46850 [https://bugs.python.org/issue?@action=redirect&bpo=46850]: Move the private undocumented `_PyEval_EvalFrameDefault()` function to the internal C API. The function now uses the `_PyInterpreterFrame` type which is part of the internal C API. Patch by Victor Stinner.

- bpo-46850 [https://bugs.python.org/issue?@action=redirect&bpo=46850]: Remove the private undocumented function `_PyEval_CallTracing()` from the C API. Call the public **`sys.call_tracing()`** function instead. Patch by Victor Stinner.

- bpo-46850 [https://bugs.python.org/issue?@action=redirect&bpo=46850]: Remove the private undocumented function `_PyEval_GetCoroutineOriginTrackingDepth()` from the C API. Call the public **`sys.get_coroutine_origin_tracking_depth()`** function instead. Patch by Victor Stinner.

- bpo-46850 [https://bugs.python.org/issue?@action=redirect&bpo=46850]: Remove the following private undocumented functions from the C API:

    ○ `_PyEval_GetAsyncGenFirstiter()`
    ○ `_PyEval_GetAsyncGenFinalizer()`
    ○ `_PyEval_SetAsyncGenFirstiter()`
    ○ `_PyEval_SetAsyncGenFinalizer()`

  Call the public **`sys.get_asyncgen_hooks()`** and **`sys.set_asyncgen_hooks()`** functions instead. Patch by Victor Stinner.

- bpo-46987 [https://bugs.python.org/issue?@action=redirect&bpo=46987]: Remove private functions `_PySys_GetObjectId()` and `_PySys_SetObjectId()`. Patch by Donghee Na.

- bpo-46906 [https://bugs.python.org/issue?@action=redirect&bpo=46906]: Add new functions to pack and unpack C double (serialize and deserialize): **`PyFloat_Pack2()`**, **`PyFloat_Pack4()`**, **`PyFloat_Pack8()`**, **`PyFloat_Unpack2()`**, **`PyFloat_Unpack4()`** and **`PyFloat_Unpack8()`**. Patch by Victor Stinner.

Python 3.11.0 alpha 6

Release date: 2022-03-07

Core and Builtins

- bpo-46940 [https://bugs.python.org/issue?@action=redirect&bpo=46940]: Avoid overriding `AttributeError` metadata information for nested attribute access calls. Patch by Pablo Galindo.
- bpo-46927 [https://bugs.python.org/issue?@action=redirect&bpo=46927]: Include the type's name in the error message for subscripting non-generic types.
- bpo-46921 [https://bugs.python.org/issue?@action=redirect&bpo=46921]: Support vectorcall for `super()`. Patch by Ken Jin.
- bpo-46841 [https://bugs.python.org/issue?@action=redirect&bpo=46841]: Fix incorrect handling of inline cache entries when specializing `BINARY_OP`.
- bpo-46841 [https://bugs.python.org/issue?@action=redirect&bpo=46841]: Use an oparg to simplify the construction of helpful error messages in `GET_AWAITABLE`.
- bpo-46903 [https://bugs.python.org/issue?@action=redirect&bpo=46903]: Make sure that str subclasses can be used as attribute names for instances with virtual dictionaries. Fixes regression in 3.11alpha
- bpo-46841 [https://bugs.python.org/issue?@action=redirect&bpo=46841]: Add more detailed specialization failure stats for `COMPARE_OP` followed by `EXTENDED_ARG`.
- bpo-46891 [https://bugs.python.org/issue?@action=redirect&bpo=46891]: Fix bug introduced during 3.11alpha where subclasses of `types.ModuleType` with `__slots__` were not initialized correctly, resulting in an interpreter crash.
- bpo-46841 [https://bugs.python.org/issue?@action=redirect&bpo=46841]: Use inline caching for `LOAD_ATTR`, `LOAD_METHOD`, and `STORE_ATTR`.
- bpo-46841 [https://bugs.python.org/issue?@action=redirect&bpo=46841]: Use inline cache for `BINARY_SUBSCR`.
- bpo-46841 [https://bugs.python.org/issue?@action=redirect&bpo=46841]: Use inline caching for `COMPARE_OP`.
- bpo-46864 [https://bugs.python.org/issue?@action=redirect&bpo=46864]: Deprecate `PyBytesObject.ob_shash`. It will be removed in Python

3.13.

- bpo-46841 [https://bugs.python.org/issue?@action=redirect&bpo=46841]: Use inline caching for **`UNPACK_SEQUENCE`**.
- bpo-46845 [https://bugs.python.org/issue?@action=redirect&bpo=46845]: Reduces dict size by removing hash value from hash table when all inserted keys are Unicode. For example, `sys.getsizeof(dict.fromkeys("abcdefg"))` becomes 272 bytes from 352 bytes on 64bit platform.
- bpo-46841 [https://bugs.python.org/issue?@action=redirect&bpo=46841]: Use inline cache for **`LOAD_GLOBAL`**.
- bpo-46852 [https://bugs.python.org/issue?@action=redirect&bpo=46852]: Rename the private undocumented `float.__set_format__()` method to `float.__setformat__()` to fix a typo introduced in Python 3.7. The method is only used by test_float. Patch by Victor Stinner.
- bpo-46852 [https://bugs.python.org/issue?@action=redirect&bpo=46852]: Remove the undocumented private `float.__set_format__()` method, previously known as `float.__setformat__()` in Python 3.7. Its docstring said: «You probably don't want to use this function. It exists mainly to be used in Python's test suite.» Patch by Victor Stinner.
- bpo-40116 [https://bugs.python.org/issue?@action=redirect&bpo=40116]: Fix regression that dict.update(other) may don't respect iterate order of other when other is key sharing dict.
- bpo-46712 [https://bugs.python.org/issue?@action=redirect&bpo=46712]: Share global string identifiers in deep-frozen modules.
- bpo-46430 [https://bugs.python.org/issue?@action=redirect&bpo=46430]: Fix memory leak in interned strings of deep-frozen modules.
- bpo-46841 [https://bugs.python.org/issue?@action=redirect&bpo=46841]: Store **`BINARY_OP`** caches inline using a new **`CACHE`** instruction.
- bpo-45107 [https://bugs.python.org/issue?

@action=redirect&bpo=45107]: Specialize `LOAD_METHOD` for instances with a dict.
- bpo-44337 [https://bugs.python.org/issue?@action=redirect&bpo=44337]: Reduce the memory usage of specialized **`LOAD_ATTR`** and **`STORE_ATTR`** instructions.
- bpo-46729 [https://bugs.python.org/issue?@action=redirect&bpo=46729]: Add number of sub-exceptions to **`BaseException.__str__()`**.
- bpo-45885 [https://bugs.python.org/issue?@action=redirect&bpo=45885]: Don't un-adapt **`COMPARE_OP`** when collecting specialization stats.
- bpo-46329 [https://bugs.python.org/issue?@action=redirect&bpo=46329]: Fix specialization stats gathering for **`PRECALL`** instructions.

Library

- bpo-46794 [https://bugs.python.org/issue?@action=redirect&bpo=46794]: Bump up the libexpat version into 2.4.6

Core and Builtins

- bpo-46823 [https://bugs.python.org/issue?@action=redirect&bpo=46823]: Implement a specialized combined opcode `LOAD_FAST__LOAD_ATTR_INSTANCE_VALUE`. Patch by Dennis Sweeney.

- bpo-46820 [https://bugs.python.org/issue?@action=redirect&bpo=46820]: Fix parsing a numeric literal immediately (without spaces) followed by «not in» keywords, like in `1not in x`. Now the parser only emits a warning, not a syntax error.

- bpo-46329 [https://bugs.python.org/issue?@action=redirect&bpo=46329]: Move `KW_NAMES` before `PRECALL` instruction in call sequence. Change `operand` of `CALL` to match `PRECALL` for easier specialization.

- bpo-46808 [https://bugs.python.org/issue?@action=redirect&bpo=46808]: Remove the `NEXT_BLOCK` macro from compile.c, and make the compiler automatically generate implicit blocks when they are needed.
- bpo-46329 [https://bugs.python.org/issue?@action=redirect&bpo=46329]: Add `PUSH_NULL` instruction. This is used as a prefix when evaluating a callable, so that the stack has the same shape for methods and other calls. `PRECALL_FUNCTION` and `PRECALL_METHOD` are merged into a single `PRECALL` instruction.

  There is no change in semantics.
- bpo-46762 [https://bugs.python.org/issue?@action=redirect&bpo=46762]: Fix an assert failure in debug builds when a "<", ">", or "=" is the last character in an f-string that's missing a closing right brace.
- bpo-46730 [https://bugs.python.org/issue?@action=redirect&bpo=46730]: Message of AttributeError caused by getting, setting or deleting a property without the corresponding function now mentions that the attribute is in fact a property and also specifies type of the class that it belongs to.
- bpo-46724 [https://bugs.python.org/issue?@action=redirect&bpo=46724]: Make sure that all backwards jumps use the `JUMP_ABSOLUTE` instruction, rather than `JUMP_FORWARD` with an argument of `(2**32)+offset`.
- bpo-46732 [https://bugs.python.org/issue?@action=redirect&bpo=46732]: Correct the docstring for the **`__bool__()`** method. Patch by Jelle Zijlstra.
- bpo-46072 [https://bugs.python.org/issue?@action=redirect&bpo=46072]: Add more detailed specialization failure statistics for **`BINARY_OP`**.
- bpo-46707 [https://bugs.python.org/issue?@action=redirect&bpo=46707]: Avoid potential exponential

backtracking when producing some syntax errors involving lots of brackets. Patch by Pablo Galindo.

Library

- bpo-46323 [https://bugs.python.org/issue?@action=redirect&bpo=46323]: `ctypes` now allocates memory on the stack instead of on the heap to pass arguments while calling a Python callback function. Patch by Donghee Na.

Core and Builtins

- bpo-45923 [https://bugs.python.org/issue?@action=redirect&bpo=45923]: Add a quickened form of `RESUME` that skips quickening checks.
- bpo-46702 [https://bugs.python.org/issue?@action=redirect&bpo=46702]: Specialize `UNPACK_SEQUENCE` for `tuple` and `list` unpackings.
- bpo-46072 [https://bugs.python.org/issue?@action=redirect&bpo=46072]: Opcode pair stats are now gathered with `--enable-pystats`. Defining `DYNAMIC_EXECUTION_PROFILE` or `DXPAIRS` no longer has any effect.
- bpo-46675 [https://bugs.python.org/issue?@action=redirect&bpo=46675]: Allow more than 16 items in a split dict before it is combined. The limit is now 254.

Library

- bpo-40479 [https://bugs.python.org/issue?@action=redirect&bpo=40479]: Add a missing call to `va_end()` in `Modules/_hashopenssl.c`.
- bpo-46323 [https://bugs.python.org/issue?@action=redirect&bpo=46323]: Use `PyObject_Vectorcall()` while calling ctypes callback function. Patch by Donghee Na.

Core and Builtins

- bpo-46615 [https://bugs.python.org/issue?

@action=redirect&bpo=46615]: When iterating over sets internally in `setobject.c`, acquire strong references to the resulting items from the set. This prevents crashes in corner-cases of various set operations where the set gets mutated.

- bpo-45828 [https://bugs.python.org/issue?@action=redirect&bpo=45828]: The bytecode compiler now attempts to apply runtime stack manipulations at compile-time (whenever it is feasible to do so).
- bpo-30496 [https://bugs.python.org/issue?@action=redirect&bpo=30496]: Fixed a minor portability issue in the implementation of `PyLong_FromLong()`, and added a fast path for single-digit integers to `PyLong_FromLongLong()`.

Library

- bpo-25707 [https://bugs.python.org/issue?@action=redirect&bpo=25707]: Fixed a file leak in `xml.etree.ElementTree.iterparse()` when the iterator is not exhausted. Patch by Jacob Walls.

- bpo-46877 [https://bugs.python.org/issue?@action=redirect&bpo=46877]: Export `unittest.doModuleCleanups()` in `unittest`. Patch by Kumar Aditya.

- bpo-46848 [https://bugs.python.org/issue?@action=redirect&bpo=46848]: For performance, use the optimized string-searching implementations from `find()` and `rfind()` for `find()` and `rfind()`.

- bpo-46736 [https://bugs.python.org/issue?@action=redirect&bpo=46736]: `SimpleHTTPRequestHandler` now uses HTML5 grammar. Patch by Donghee Na.

- bpo-44886 [https://bugs.python.org/issue?@action=redirect&bpo=44886]: Inherit asyncio proactor datagram transport from `asyncio.DatagramTransport`.

- bpo-46827 [https://bugs.python.org/issue?@action=redirect&bpo=46827]: Support UDP sockets in `asyncio.loop.sock_connect()` for selector-based event loops. Patch by Thomas Grainger.

- bpo-46811 [https://bugs.python.org/issue?@action=redirect&bpo=46811]: Make test suite support Expat >=2.4.5

- bpo-46252 [https://bugs.python.org/issue?@action=redirect&bpo=46252]: Raise `TypeError` if `ssl.SSLSocket` is passed to transport-based APIs.

- bpo-46784 [https://bugs.python.org/issue?@action=redirect&bpo=46784]: Fix libexpat symbols collisions with user dynamically loaded or statically linked libexpat in embedded Python.

- bpo-46786 [https://bugs.python.org/issue?@action=redirect&bpo=46786]: The HTML serialisation in xml.etree.ElementTree now writes `embed`, `source`, `track` and `wbr` as empty tags, as defined in HTML 5.

- bpo-39327 [https://bugs.python.org/issue?@action=redirect&bpo=39327]: `shutil.rmtree()` can now work with VirtualBox shared folders when running from the guest operating-system.

- bpo-45390 [https://bugs.python.org/issue?@action=redirect&bpo=45390]: Propagate `asyncio.CancelledError` message from inner task to outer awaiter.

- bpo-46756 [https://bugs.python.org/issue?@action=redirect&bpo=46756]: Fix a bug in `urllib.request.HTTPPasswordMgr.find_user_password()` and `urllib.request.HTTPPasswordMgrWithPriorAuth.is_auth` which allowed to bypass authorization. For example, access to URI `example.org/foobar` was allowed if the user was authorized for URI `example.org/foo`.

- bpo-46737 [https://bugs.python.org/issue?

@action=redirect&bpo=46737]: `random.gauss()` and `random.normalvariate()` now have default arguments.

- bpo-46752 [https://bugs.python.org/issue?@action=redirect&bpo=46752]: Add task groups to asyncio (structured concurrency, inspired by Trio's nurseries). This also introduces a change to task cancellation, where a cancelled task can't be cancelled again until it calls .uncancel().
- bpo-46724 [https://bugs.python.org/issue?@action=redirect&bpo=46724]: Fix `dis` behavior on negative jump offsets.
- bpo-46333 [https://bugs.python.org/issue?@action=redirect&bpo=46333]: The `__repr__()` method of `typing.ForwardRef` now includes the `module` parameter of `typing.ForwardRef` when it is set.
- bpo-46643 [https://bugs.python.org/issue?@action=redirect&bpo=46643]: In `typing.get_type_hints()`, support evaluating stringified `ParamSpecArgs` and `ParamSpecKwargs` annotations. Patch by Gregory Beauregard.
- bpo-45863 [https://bugs.python.org/issue?@action=redirect&bpo=45863]: When the `tarfile` module creates a pax format archive, it will put an integer representation of timestamps in the ustar header (if possible) for the benefit of older unarchivers, in addition to the existing full-precision timestamps in the pax extended header.
- bpo-46066 [https://bugs.python.org/issue?@action=redirect&bpo=46066]: Deprecate kwargs-based syntax for `typing.TypedDict` definitions. It had confusing semantics when specifying totality, and was largely unused. Patch by Jingchen Ye.
- bpo-46676 [https://bugs.python.org/issue?@action=redirect&bpo=46676]: Make `typing.ParamSpec` args and kwargs equal to themselves. Patch by Gregory

Beauregard.

- bpo-46323 [https://bugs.python.org/issue?@action=redirect&bpo=46323]: `ctypes.CFUNCTYPE()` and `ctypes.WINFUNCTYPE()` now fail to create the type if its `_argtypes_` member contains too many arguments. Previously, the error was only raised when calling a function. Patch by Victor Stinner.

- bpo-46672 [https://bugs.python.org/issue?@action=redirect&bpo=46672]: Fix `NameError` in `asyncio.gather()` when initial type check fails.

- bpo-46659 [https://bugs.python.org/issue?@action=redirect&bpo=46659]: The `calendar.LocaleTextCalendar` and `calendar.LocaleHTMLCalendar` classes now use `locale.getlocale()`, instead of using `locale.getdefaultlocale()`, if no locale is specified. Patch by Victor Stinner.

- bpo-46659 [https://bugs.python.org/issue?@action=redirect&bpo=46659]: The `locale.getdefaultlocale()` function is deprecated and will be removed in Python 3.13. Use `locale.setlocale()`, `locale.getpreferredencoding(False)` and `locale.getlocale()` functions instead. Patch by Victor Stinner.

- bpo-46655 [https://bugs.python.org/issue?@action=redirect&bpo=46655]: In `typing.get_type_hints()`, support evaluating bare stringified `TypeAlias` annotations. Patch by Gregory Beauregard.

- bpo-45948 [https://bugs.python.org/issue?@action=redirect&bpo=45948]: Fixed a discrepancy in the C implementation of the `xml.etree.ElementTree` module. Now, instantiating an `xml.etree.ElementTree.XMLParser` with a `target=None` keyword provides a default **`xml.etree.ElementTree.TreeBuilder`** target as the Python implementation does.

- bpo-46626 [https://bugs.python.org/issue?@action=redirect&bpo=46626]: Expose Linux's `IP_BIND_ADDRESS_NO_PORT` option in **`socket`**.
- bpo-46521 [https://bugs.python.org/issue?@action=redirect&bpo=46521]: Fix a bug in the **`codeop`** module that was incorrectly identifying invalid code involving string quotes as valid code.
- bpo-46571 [https://bugs.python.org/issue?@action=redirect&bpo=46571]: Improve **`typing.no_type_check()`**.

  Now it does not modify external classes and functions. We also now correctly mark classmethods as not to be type checked.
- bpo-46400 [https://bugs.python.org/issue?@action=redirect&bpo=46400]: expat: Update libexpat from 2.4.1 to 2.4.4
- bpo-46556 [https://bugs.python.org/issue?@action=redirect&bpo=46556]: Deprecate undocumented support for using a **`pathlib.Path`** object as a context manager.
- bpo-46534 [https://bugs.python.org/issue?@action=redirect&bpo=46534]: Implement **PEP 673** [https://peps.python.org/pep-0673/] **`typing.Self`**. Patch by James Hilton-Balfe.
- bpo-46522 [https://bugs.python.org/issue?@action=redirect&bpo=46522]: Make various module `__getattr__` AttributeErrors more closely match a typical AttributeError
- bpo-46475 [https://bugs.python.org/issue?@action=redirect&bpo=46475]: Add **`typing.Never`** and

`typing.assert_never()`. Patch by Jelle Zijlstra.

- bpo-46333 [https://bugs.python.org/issue?@action=redirect&bpo=46333]: The `__eq__()` and `__hash__()` methods of `typing.ForwardRef` now honor the `module` parameter of `typing.ForwardRef`. Forward references from different modules are now differentiated.

- bpo-46246 [https://bugs.python.org/issue?@action=redirect&bpo=46246]: Add missing `__slots__` to `importlib.metadata.DeprecatedList`. Patch by Arie Bovenberg.

- bpo-46232 [https://bugs.python.org/issue?@action=redirect&bpo=46232]: The `ssl` module now handles certificates with bit strings in DN correctly.

- bpo-46195 [https://bugs.python.org/issue?@action=redirect&bpo=46195]: `typing.get_type_hints()` no longer adds `Optional` to parameters with `None` as a default. This aligns to changes to PEP 484 in https://github.com/python/peps/pull/689

- bpo-31369 [https://bugs.python.org/issue?@action=redirect&bpo=31369]: Add `RegexFlag` to `re.__all__` and documented it. Add `NOFLAG` to indicate no flags being set.

- bpo-45898 [https://bugs.python.org/issue?@action=redirect&bpo=45898]: `ctypes` no longer defines `ffi_type_*` symbols in `cfield.c`. The symbols have been provided by libffi for over a decade.

- bpo-44953 [https://bugs.python.org/issue?@action=redirect&bpo=44953]: Calling `operator.itemgetter` objects and `operator.attrgetter` objects is now faster due to use of the vectorcall calling convention.

- bpo-44289 [https://bugs.python.org/issue?@action=redirect&bpo=44289]: Fix an issue with `is_tarfile()` method when using *fileobj* argument: position in the *fileobj* was advanced forward which made it unreadable with `tarfile.TarFile.open()`.

- bpo-44011 [https://bugs.python.org/issue?@action=redirect&bpo=44011]: Reimplement SSL/TLS support in asyncio, borrow the implementation from uvloop library.

- bpo-41086 [https://bugs.python.org/issue?@action=redirect&bpo=41086]: Make the `configparser.ConfigParser` constructor raise `TypeError` if the `interpolation` parameter is not of type **`configparser.Interpolation`**

- bpo-29418 [https://bugs.python.org/issue?@action=redirect&bpo=29418]: Implement `inspect.ismethodwrapper()` and fix `inspect.isroutine()` for cases where methodwrapper is given. Patch by Hakan Çelik.

- bpo-14156 [https://bugs.python.org/issue?@action=redirect&bpo=14156]: argparse.FileType now supports an argument of "-" in binary mode, returning the .buffer attribute of sys.stdin/sys.stdout as appropriate. Modes including "x" and "a" are treated equivalently to "w" when argument is "-". Patch contributed by Josh Rosenberg

Documentation

- bpo-42238 [https://bugs.python.org/issue?@action=redirect&bpo=42238]: `Doc/tools/rstlint.py` has moved to its own repository and is now packaged on PyPI as `sphinx-lint`.

Tests

- bpo-46913 [https://bugs.python.org/issue?@action=redirect&bpo=46913]: Fix test_faulthandler.test_sigfpe() if Python is built with undefined behavior sanitizer (UBSAN): disable UBSAN on the faulthandler_sigfpe() function. Patch by Victor Stinner.

- bpo-46760 [https://bugs.python.org/issue?@action=redirect&bpo=46760]: Remove bytecode offsets from expected values in test.test_dis module. Reduces the obstacles to modifying the VM or compiler.
- bpo-46708 [https://bugs.python.org/issue?@action=redirect&bpo=46708]: Prevent default asyncio event loop policy modification warning after `test_asyncio` execution.
- bpo-46678 [https://bugs.python.org/issue?@action=redirect&bpo=46678]: The function `make_legacy_pyc` in `Lib/test/support/import_helper.py` no longer fails when `PYTHONPYCACHEPREFIX` is set to a directory on a different device from where tempfiles are stored.
- bpo-46623 [https://bugs.python.org/issue?@action=redirect&bpo=46623]: Skip test_pair() and test_speech128() of test_zlib on s390x since they fail if zlib uses the s390x hardware accelerator. Patch by Victor Stinner.

Build

- bpo-46860 [https://bugs.python.org/issue?@action=redirect&bpo=46860]: Respect `--with-suffix` when building on case-insensitive file systems.
- bpo-46656 [https://bugs.python.org/issue?@action=redirect&bpo=46656]: Building Python now requires a C11 compiler. Optional C11 features are not required. Patch by Victor Stinner.
- bpo-46656 [https://bugs.python.org/issue?@action=redirect&bpo=46656]: Building Python now requires support for floating-point Not-a-Number (NaN): remove the `Py_NO_NAN` macro. Patch by Victor Stinner.
- bpo-46640 [https://bugs.python.org/issue?@action=redirect&bpo=46640]: Building Python now requires a C99 `<math.h>` header file providing a `NAN` constant, or the `__builtin_nan()` built-in function. Patch by Victor Stinner.
- bpo-46608 [https://bugs.python.org/issue?@action=redirect&bpo=46608]: Exclude marshalled-frozen data if deep-freezing to save 300 KB disk space. This includes adding a new `is_package` field to **`_frozen`**. Patch by Kumar

Aditya.

- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: Fix wasm32-emscripten test failures and platform issues. - Disable syscalls that are not supported or don't work, e.g. wait, getrusage, prlimit, mkfifo, mknod, setres[gu]id, setgroups. - Use fd_count to count open fds. - Add more checks for subprocess and fork. - Add workarounds for missing _multiprocessing and failing socket.accept(). - Enable bzip2. - Disable large file support. - Disable signal.alarm.
- bpo-46430 [https://bugs.python.org/issue?@action=redirect&bpo=46430]: Intern strings in deep-frozen modules. Patch by Kumar Aditya.

Windows

- bpo-46744 [https://bugs.python.org/issue?@action=redirect&bpo=46744]: The default all users install directory for ARM64 is now under the native `Program Files` folder, rather than `Program Files (Arm)` which is intended for ARM (32-bit) files.
- bpo-46567 [https://bugs.python.org/issue?@action=redirect&bpo=46567]: Adds Tcl and Tk support for Windows ARM64. This also adds IDLE to the installation.
- bpo-46638 [https://bugs.python.org/issue?@action=redirect&bpo=46638]: Ensures registry virtualization is consistently disabled. For 3.10 and earlier, it remains enabled (some registry writes are protected), while for 3.11 and later it is disabled (registry modifications affect all applications).

IDLE

- bpo-46630 [https://bugs.python.org/issue?@action=redirect&bpo=46630]: Make query dialogs on Windows start with a cursor in the entry box.
- bpo-45447 [https://bugs.python.org/issue?@action=redirect&bpo=45447]: Apply IDLE syntax highlighting to `.pyi` files. Patch by Alex Waygood and Terry Jan Reedy.

C API

- bpo-46748 [https://bugs.python.org/issue?@action=redirect&bpo=46748]: Python’s public headers no longer import `<stdbool.h>`, leaving code that embeds/extends Python free to define `bool`, `true` and `false`.
- bpo-46836 [https://bugs.python.org/issue?@action=redirect&bpo=46836]: Move the **`PyFrameObject`** type definition (`struct _frame`) to the internal C API `pycore_frame.h` header file. Patch by Victor Stinner.
- bpo-45459 [https://bugs.python.org/issue?@action=redirect&bpo=45459]: Rename `Include/buffer.h` header file to `Include/pybuffer.h` to avoid conflicts with projects having an existing `buffer.h` header file. Patch by Victor Stinner.
- bpo-45412 [https://bugs.python.org/issue?@action=redirect&bpo=45412]: Remove the `HAVE_PY_SET_53BIT_PRECISION` macro (moved to the internal C API). Patch by Victor Stinner.
- bpo-46613 [https://bugs.python.org/issue?@action=redirect&bpo=46613]: Added function **`PyType_GetModuleByDef()`**, which allows access to module state when a method’s defining class is not available.

Python 3.11.0 alpha 5

Release date: 2022-02-03

Core and Builtins

- bpo-45773 [https://bugs.python.org/issue?@action=redirect&bpo=45773]: Remove two invalid «peephole» optimizations from the bytecode compiler.

- bpo-46564 [https://bugs.python.org/issue?@action=redirect&bpo=46564]: Do not create frame objects when creating **`super`** object. Patch by Kumar Aditya.

- bpo-45885 [https://bugs.python.org/issue?@action=redirect&bpo=45885]: Added more fined-grained specialization failure stats regarding the `COMPARE_OP` bytecode.

- bpo-44977 [https://bugs.python.org/issue?@action=redirect&bpo=44977]: The delegation of `int()` to `__trunc__()` is now deprecated. Calling `int(a)` when `type(a)` implements `__trunc__()` but not `__int__()` or `__index__()` now raises a `DeprecationWarning`.

- bpo-46458 [https://bugs.python.org/issue?@action=redirect&bpo=46458]: Reorder code emitted by the compiler for a `try`-`except` block so that the `else` block's code immediately follows the `try` body (without a jump). This is more optimal for the happy path.

- bpo-46527 [https://bugs.python.org/issue?@action=redirect&bpo=46527]: Allow passing `iterable` as a keyword argument to `enumerate()` again. Patch by Jelle Zijlstra.

- bpo-46528 [https://bugs.python.org/issue?@action=redirect&bpo=46528]: Replace several stack manipulation instructions (`DUP_TOP`, `DUP_TOP_TWO`, `ROT_TWO`, `ROT_THREE`, `ROT_FOUR`, and `ROT_N`) with new `COPY` and `SWAP` instructions.

- bpo-46329 [https://bugs.python.org/issue?@action=redirect&bpo=46329]: Use two or three bytecodes to implement most calls.

  Calls without named arguments are implemented as a sequence of two instructions: `PRECALL; CALL`. Calls with named arguments are implemented as a sequence of three instructions: `PRECALL; KW_NAMES; CALL`. There are two different `PRECALL` instructions: `PRECALL_FUNTION` and `PRECALL_METHOD`. The latter pairs with `LOAD_METHOD`.

  This partition into pre-call and call allows better specialization, and thus better performance ultimately.

  There is no change in semantics.

- bpo-46503 [https://bugs.python.org/issue?@action=redirect&bpo=46503]: Fix an assert when parsing some invalid N escape sequences in f-strings.

- bpo-46431 [https://bugs.python.org/issue?@action=redirect&bpo=46431]: Improve error message on invalid calls to **`BaseExceptionGroup.__new__()`**.

- bpo-46476 [https://bugs.python.org/issue?@action=redirect&bpo=46476]: Fix memory leak in code objects generated by deepfreeze. Patch by Kumar Aditya.

- bpo-46481 [https://bugs.python.org/issue?@action=redirect&bpo=46481]: Speed up calls to **`weakref.ref.__call__()`** by using the **PEP 590** [https://peps.python.org/pep-0590/] `vectorcall` calling convention. Patch by Donghee Na.

- bpo-46417 [https://bugs.python.org/issue?@action=redirect&bpo=46417]: Fix a race condition on setting a type `__bases__` attribute: the internal function `add_subclass()` now gets the `PyTypeObject.tp_subclasses` member after calling **`PyWeakref_NewRef()`** which can trigger a garbage collection which can indirectly modify `PyTypeObject.tp_subclasses`. Patch by Victor Stinner.

- bpo-46417 [https://bugs.python.org/issue?@action=redirect&bpo=46417]: `python -X showrefcount` now shows the total reference count after clearing and destroyed the main Python interpreter. Previously, it was shown before. Patch by Victor Stinner.

- bpo-43683 [https://bugs.python.org/issue?@action=redirect&bpo=43683]: Add ASYNC_GEN_WRAP opcode to wrap the value to be yielded in async generators. Removes the need to special case async generators in the `YIELD_VALUE` instruction.

- bpo-46407 [https://bugs.python.org/issue?@action=redirect&bpo=46407]: Optimize some modulo operations in `Objects/longobject.c`. Patch by Jeremiah Vivian.

- bpo-46409 [https://bugs.python.org/issue?@action=redirect&bpo=46409]: Add new `RETURN_GENERATOR` bytecode to make generators. Simplifies calling Python functions in the VM, as they no longer any need to special case generator functions.

  Also add `JUMP_NO_INTERRUPT` bytecode that acts like `JUMP_ABSOLUTE`, but does not check for interrupts.

- bpo-46406 [https://bugs.python.org/issue?@action=redirect&bpo=46406]: The integer division `//` implementation has been optimized to better let the compiler understand its constraints. It can be 20% faster on the amd64 platform when dividing an int by a value smaller than `2**30`.

Library

- bpo-46383 [https://bugs.python.org/issue?@action=redirect&bpo=46383]: Fix invalid signature of `_zoneinfo`'s `module_free` function to resolve a crash on wasm32-emscripten platform.

Core and Builtins

- bpo-46361 [https://bugs.python.org/issue?@action=redirect&bpo=46361]: Ensure that «small» integers created by **`int.from_bytes()`** and **`decimal.Decimal`** are properly cached.
- bpo-46161 [https://bugs.python.org/issue?@action=redirect&bpo=46161]: Fix the class building error when the arguments are constants and CALL_FUNCTION_EX is used.
- bpo-46028 [https://bugs.python.org/issue?@action=redirect&bpo=46028]: Fixes calculation of **`sys._base_executable`** when inside a virtual environment that uses symlinks with different binary names than the base environment provides.
- bpo-46091 [https://bugs.python.org/issue?@action=redirect&bpo=46091]: Correctly calculate indentation levels for lines with whitespace character that are ended by

line continuation characters. Patch by Pablo Galindo

Library

- bpo-30512 [https://bugs.python.org/issue?@action=redirect&bpo=30512]: Add CAN Socket support for NetBSD.

Build

- bpo-46045 [https://bugs.python.org/issue?@action=redirect&bpo=46045]: Do not use POSIX semaphores on NetBSD

Core and Builtins

- bpo-44024 [https://bugs.python.org/issue?@action=redirect&bpo=44024]: Improve the `TypeError` message for non-string second arguments passed to the built-in functions `getattr()` and `hasattr()`. Patch by Géry Ogam.

Library

- bpo-46624 [https://bugs.python.org/issue?@action=redirect&bpo=46624]: Restore support for non-integer arguments of `random.randrange()` and `random.randint()`.
- bpo-46591 [https://bugs.python.org/issue?@action=redirect&bpo=46591]: Make the IDLE doc URL on the About IDLE dialog clickable.
- bpo-46565 [https://bugs.python.org/issue?@action=redirect&bpo=46565]: Remove loop variables that are leaking into modules” namespaces.
- bpo-46553 [https://bugs.python.org/issue?@action=redirect&bpo=46553]: In `typing.get_type_hints()`, support evaluating bare stringified `ClassVar` annotations. Patch by Gregory Beauregard.

- bpo-46544 [https://bugs.python.org/issue?@action=redirect&bpo=46544]: Don’t leak `x` & `uspace` intermediate vars in `textwrap.TextWrapper`.
- bpo-46487 [https://bugs.python.org/issue?@action=redirect&bpo=46487]: Add the `get_write_buffer_limits` method to `asyncio.transports.WriteTransport` and to the SSL transport.
- bpo-45173 [https://bugs.python.org/issue?@action=redirect&bpo=45173]: Note the configparser deprecations will be removed in Python 3.12.
- bpo-45162 [https://bugs.python.org/issue?@action=redirect&bpo=45162]: The deprecated `unittest` APIs removed in 3.11a1 have been temporarily restored to be removed in 3.12 while cleanups in external projects go in.
- bpo-46539 [https://bugs.python.org/issue?@action=redirect&bpo=46539]: In `typing.get_type_hints()`, support evaluating stringified `ClassVar` and `Final` annotations inside `Annotated`. Patch by Gregory Beauregard.
- bpo-46510 [https://bugs.python.org/issue?@action=redirect&bpo=46510]: Add missing test for `types.TracebackType` and `types.FrameType`. Calculate them directly from the caught exception without calling `sys.exc_info()`.
- bpo-46491 [https://bugs.python.org/issue?@action=redirect&bpo=46491]: Allow `typing.Annotated` to wrap `typing.Final` and `typing.ClassVar`. Patch by Gregory Beauregard.
- bpo-46483 [https://bugs.python.org/issue?@action=redirect&bpo=46483]: Remove `__class_getitem__()` from `pathlib.PurePath` as this class was not supposed to be generic.
- bpo-46436 [https://bugs.python.org/issue?@action=redirect&bpo=46436]: Fix command-line option `-d/--directory` in module `http.server` which is ignored when combined with command-line option `--cgi`. Patch by Géry Ogam.
- bpo-41403 [https://bugs.python.org/issue?@action=redirect&bpo=41403]: Make `mock.patch()` raise a

`TypeError` with a relevant error message on invalid arg. Previously it allowed a cryptic **`AttributeError`** to escape.

- bpo-46474 [https://bugs.python.org/issue?@action=redirect&bpo=46474]: In `importlib.metadata.EntryPoint.pattern`, avoid potential REDoS by limiting ambiguity in consecutive whitespace.
- bpo-46474 [https://bugs.python.org/issue?@action=redirect&bpo=46474]: Removed private method from `importlib.metadata.Path`. Sync with importlib_metadata 4.10.0.
- bpo-46470 [https://bugs.python.org/issue?@action=redirect&bpo=46470]: Remove unused branch from `typing._remove_dups_flatten`
- bpo-46469 [https://bugs.python.org/issue?@action=redirect&bpo=46469]: **`asyncio`** generic classes now return **`types.GenericAlias`** in `__class_getitem__` instead of the same class.
- bpo-41906 [https://bugs.python.org/issue?@action=redirect&bpo=41906]: Support passing filter instances in the `filters` values of `handlers` and `loggers` in the dictionary passed to **`logging.config.dictConfig()`**.
- bpo-46422 [https://bugs.python.org/issue?@action=redirect&bpo=46422]: Use `dis.Positions` in `dis.Instruction` instead of a regular `tuple`.
- bpo-46434 [https://bugs.python.org/issue?@action=redirect&bpo=46434]: **`pdb`** now gracefully handles `help` when **`__doc__`** is missing, for example when run with pregenerated optimized `.pyc` files.
- bpo-43869 [https://bugs.python.org/issue?@action=redirect&bpo=43869]: Python uses the same time Epoch on all platforms. Add an explicit unit test to ensure that it's the case. Patch by Victor Stinner.
- bpo-46414 [https://bugs.python.org/issue?@action=redirect&bpo=46414]: Add **`typing.reveal_type()`**. Patch by Jelle Zijlstra.
- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: **`subprocess`** now imports Windows-specific imports when `msvcrt` module is available, and POSIX-specific imports on all other platforms. This gives

a clean exception when `_posixsubprocess` is not available (e.g. Emscripten browser target).

- bpo-40066 [https://bugs.python.org/issue?@action=redirect&bpo=40066]: `IntEnum`, `IntFlag`, and `StrEnum` use the mixed-in type for their `str()` and `format()` output.
- bpo-46316 [https://bugs.python.org/issue?@action=redirect&bpo=46316]: Optimize **`pathlib.Path.iterdir()`** by removing an unnecessary check for special entries.
- bpo-29688 [https://bugs.python.org/issue?@action=redirect&bpo=29688]: Document **`pathlib.Path.absolute()`** (which has always existed).
- bpo-43012 [https://bugs.python.org/issue?@action=redirect&bpo=43012]: The pathlib module's obsolete and internal `_Accessor` class has been removed to prepare the terrain for upcoming enhancements to the module.
- bpo-46258 [https://bugs.python.org/issue?@action=redirect&bpo=46258]: Speed up **`math.isqrt()`** for small positive integers by replacing two division steps with a lookup table.
- bpo-46242 [https://bugs.python.org/issue?@action=redirect&bpo=46242]: Improve error message when creating a new **`enum.Enum`** type subclassing an existing `Enum` with `_member_names_` using **`enum.Enum.__call__()`**.
- bpo-43118 [https://bugs.python.org/issue?@action=redirect&bpo=43118]: Fix a bug in **`inspect.signature()`** that was causing it to fail on some subclasses of classes with a `__text_signature__` referencing module globals. Patch by Weipeng Hong.
- bpo-26552 [https://bugs.python.org/issue?@action=redirect&bpo=26552]: Fixed case where failing **`asyncio.ensure_future()`** did not close the coroutine. Patch by Kumar Aditya.
- bpo-21987 [https://bugs.python.org/issue?@action=redirect&bpo=21987]: Fix an issue with **`tarfile.TarFile.getmember()`** getting a directory name with a trailing slash.
- bpo-46124 [https://bugs.python.org/issue?

@action=redirect&bpo=46124]: Update `zoneinfo` to rely on importlib.resources traversable API.
- bpo-46103 [https://bugs.python.org/issue?@action=redirect&bpo=46103]: Now `inspect.getmembers()` only gets `__bases__` attribute from class type. Patch by Weipeng Hong.
- bpo-46080 [https://bugs.python.org/issue?@action=redirect&bpo=46080]: Fix exception in argparse help text generation if a `argparse.BooleanOptionalAction` argument's default is `argparse.SUPPRESS` and it has `help` specified. Patch by Felix Fontein.
- bpo-44791 [https://bugs.python.org/issue?@action=redirect&bpo=44791]: Fix substitution of `ParamSpec` in `Concatenate` with different parameter expressions. Substitution with a list of types returns now a tuple of types. Substitution with `Concatenate` returns now a `Concatenate` with concatenated lists of arguments.

Documentation

- bpo-46463 [https://bugs.python.org/issue?@action=redirect&bpo=46463]: Fixes `escape4chm.py` script used when building the CHM documentation file

Tests

- bpo-43478 [https://bugs.python.org/issue?@action=redirect&bpo=43478]: Mocks can no longer be provided as the specs for other Mocks. As a result, an already-mocked object cannot be passed to `mock.Mock()`. This can uncover bugs in tests since these Mock-derived Mocks will always pass certain tests (e.g. isinstance) and builtin assert functions (e.g. assert_called_once_with) will unconditionally pass.
- bpo-46616 [https://bugs.python.org/issue?@action=redirect&bpo=46616]: Ensures `test_importlib.test_windows` cleans up registry keys after completion.
- bpo-44359 [https://bugs.python.org/issue?@action=redirect&bpo=44359]: test_ftplib now silently ignores socket errors to prevent logging unhandled threading

exceptions. Patch by Victor Stinner.

- bpo-46600 [https://bugs.python.org/issue?@action=redirect&bpo=46600]: Fix test_gdb.test_pycfunction() for Python built with `clang -Og`. Tolerate inlined functions in the gdb traceback. Patch by Victor Stinner.
- bpo-46542 [https://bugs.python.org/issue?@action=redirect&bpo=46542]: Fix a Python crash in test_lib2to3 when using Python built in debug mode: limit the recursion limit. Patch by Victor Stinner.
- bpo-46576 [https://bugs.python.org/issue?@action=redirect&bpo=46576]: test_peg_generator now disables compiler optimization when testing compilation of its own C extensions to significantly speed up the testing on non-debug builds of CPython.
- bpo-46542 [https://bugs.python.org/issue?@action=redirect&bpo=46542]: Fix `test_json` tests checking for **`RecursionError`**: modify these tests to use `support.infinite_recursion()`. Patch by Victor Stinner.
- bpo-13886 [https://bugs.python.org/issue?@action=redirect&bpo=13886]: Skip test_builtin PTY tests on non-ASCII characters if the readline module is loaded. The readline module changes input() behavior, but test_builtin is not intended to test the readline module. Patch by Victor Stinner.
- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: Add **`test.support.requires_fork()`** decorators to mark tests that require a working **`os.fork()`**.
- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: Add **`test.support.requires_subprocess()`** decorator to mark tests which require working **`subprocess`** module or `os.spawn*`. The wasm32-emscripten platform has no support for processes.
- bpo-46126 [https://bugs.python.org/issue?@action=redirect&bpo=46126]: Disable “descriptions” when running tests internally.

Build

- bpo-46602 [https://bugs.python.org/issue?@action=redirect&bpo=46602]: Tidied up configure.ac so that conftest.c is truncated rather than appended. This assists in the case where the “rm” of conftest.c fails to happen between tests. Downstream issues such as a clobbered SOABI can result.
- bpo-46600 [https://bugs.python.org/issue?@action=redirect&bpo=46600]: Fix the test checking if the C compiler supports `-Og` option in the `./configure` script to also use `-Og` on clang which supports it. Patch by Victor Stinner.
- bpo-38472 [https://bugs.python.org/issue?@action=redirect&bpo=38472]: Fix GCC detection in setup.py when cross-compiling. The C compiler is now run with LC_ALL=C. Previously, the detection failed with a German locale.
- bpo-46513 [https://bugs.python.org/issue?@action=redirect&bpo=46513]: **configure** no longer uses `AC_C_CHAR_UNSIGNED` macro and `pyconfig.h` no longer defines reserved symbol `__CHAR_UNSIGNED__`.
- bpo-46471 [https://bugs.python.org/issue?@action=redirect&bpo=46471]: Use global singletons for single byte bytes objects in deepfreeze.
- bpo-46443 [https://bugs.python.org/issue?@action=redirect&bpo=46443]: Deepfreeze now uses cached small integers as it saves some space for common small integers.
- bpo-46429 [https://bugs.python.org/issue?@action=redirect&bpo=46429]: Merge all deep-frozen files into one for space savings. Patch by Kumar Aditya.
- bpo-45569 [https://bugs.python.org/issue?@action=redirect&bpo=45569]: The build now defaults to using 30-bit digits for Python integers. Previously either 15-bit or 30-bit digits would be selected, depending on the platform. 15-bit digits may still be selected using the `--enable-big-digits=15` option to the `configure` script, or by defining `PYLONG_BITS_IN_DIGIT` in `pyconfig.h`.
- bpo-45925 [https://bugs.python.org/issue?@action=redirect&bpo=45925]: Update Windows installer to use SQLite 3.37.2.
- bpo-43112 [https://bugs.python.org/issue?

@action=redirect&bpo=43112]: Detect musl libc as a separate SOABI (tagged as `linux-musl`).

Windows

- bpo-33125 [https://bugs.python.org/issue?@action=redirect&bpo=33125]: The traditional EXE/MSI based installer for Windows is now available for ARM64
- bpo-46362 [https://bugs.python.org/issue?@action=redirect&bpo=46362]: os.path.abspath(«C:CON») is now fixed to return «\.CON», not the same path. The regression was true of all legacy DOS devices such as COM1, LPT1, or NUL.
- bpo-44934 [https://bugs.python.org/issue?@action=redirect&bpo=44934]: The installer now offers a command-line only option to add the installation directory to the end of **`PATH`** instead of at the start.

macOS

- bpo-45925 [https://bugs.python.org/issue?@action=redirect&bpo=45925]: Update macOS installer to SQLite 3.37.2.

IDLE

- bpo-45296 [https://bugs.python.org/issue?@action=redirect&bpo=45296]: Clarify close, quit, and exit in IDLE. In the File menu, “Close” and “Exit” are now “Close Window” (the current one) and “Exit” is now “Exit IDLE” (by closing all windows). In Shell, “quit()” and “exit()” mean “close Shell”. If there are no other windows, this also exits IDLE.

C API

- bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170]: Remove the `PyHeapType_GET_MEMBERS()` macro. It was exposed in the public C API by mistake, it must only be used by Python internally. Use the `PyTypeObject.tp_members` member instead. Patch by Victor Stinner.

- bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170]: Move _Py_GetAllocatedBlocks() and _PyObject_DebugMallocStats() private functions to the internal C API. Patch by Victor Stinner.
- bpo-46433 [https://bugs.python.org/issue?@action=redirect&bpo=46433]: The internal function _PyType_GetModuleByDef now correctly handles inheritance patterns involving static types.
- bpo-45459 [https://bugs.python.org/issue?@action=redirect&bpo=45459]: **`Py_buffer`** and various `Py_buffer` related functions are now part of the limited API and stable ABI.
- bpo-14916 [https://bugs.python.org/issue?@action=redirect&bpo=14916]: Fixed bug in the tokenizer that prevented `PyRun_InteractiveOne` from parsing from the provided FD.

Python 3.11.0 alpha 4

Release date: 2022-01-13

Core and Builtins

- bpo-46070 [https://bugs.python.org/issue?@action=redirect&bpo=46070]: **`Py_EndInterpreter()`** now explicitly untracks all objects currently tracked by the GC. Previously, if an object was used later by another interpreter, calling **`PyObject_GC_UnTrack()`** on the object crashed if the previous or the next object of the **`PyGC_Head`** structure became a dangling pointer. Patch by Victor Stinner.
- bpo-46347 [https://bugs.python.org/issue?@action=redirect&bpo=46347]: Fix memory leak in PyEval_EvalCodeEx.
- bpo-46339 [https://bugs.python.org/issue?@action=redirect&bpo=46339]: Fix a crash in the parser when retrieving the error text for multi-line f-strings expressions

that do not start in the first line of the string. Patch by Pablo Galindo

- bpo-46331 [https://bugs.python.org/issue?@action=redirect&bpo=46331]: Do not set line number of instruction storing doc-string. Fixes regression introduced in 3.11 alpha.
- bpo-46314 [https://bugs.python.org/issue?@action=redirect&bpo=46314]: Remove spurious «call» event when creating a lambda function that was accidentally introduced in 3.11a4.
- bpo-46289 [https://bugs.python.org/issue?@action=redirect&bpo=46289]: ASDL declaration of `FormattedValue` has changed to reflect `conversion` field is not optional.
- bpo-46297 [https://bugs.python.org/issue?@action=redirect&bpo=46297]: Fixed an interpreter crash on bootup with multiple PythonPaths set in the Windows registry. Patch by Derzsi Dániel.
- bpo-46237 [https://bugs.python.org/issue?@action=redirect&bpo=46237]: Fix the line number of tokenizer errors inside f-strings. Patch by Pablo Galindo.
- bpo-46263 [https://bugs.python.org/issue?@action=redirect&bpo=46263]: We always expect the «use_frozen_modules» config to be set, now that getpath.c was rewritten in pure Python and the logic improved.
- bpo-46006 [https://bugs.python.org/issue?@action=redirect&bpo=46006]: Fix a regression when a type method like `__init__()` is modified in a subinterpreter. Fix a regression in `_PyUnicode_EqualToASCIIId()` and type `update_slot()`. Revert the change which made the Unicode dictionary of interned strings compatible with subinterpreters: the internal interned dictionary is shared again by all interpreters. Patch by Victor Stinner.
- bpo-45923 [https://bugs.python.org/issue?@action=redirect&bpo=45923]: Add RESUME opcode. This is a logical no-op. It is emitted by the compiler anywhere a Python function can be entered. It is used by the interpreter to perform tracing and optimizer checks.

Library

- bpo-46208 [https://bugs.python.org/issue?@action=redirect&bpo=46208]: Fix the regression of os.path.normpath(«A/../../B») not returning expected «../B» but «B».

Core and Builtins

- bpo-46240 [https://bugs.python.org/issue?@action=redirect&bpo=46240]: Correct the error message for unclosed parentheses when the tokenizer doesn't reach the end of the source when the error is reported. Patch by Pablo Galindo
- bpo-46009 [https://bugs.python.org/issue?@action=redirect&bpo=46009]: Remove the `GEN_START` opcode.
- bpo-46235 [https://bugs.python.org/issue?@action=redirect&bpo=46235]: Certain sequence multiplication operations like `[0] * 1_000` are now faster due to reference-counting optimizations. Patch by Dennis Sweeney.
- bpo-46221 [https://bugs.python.org/issue?@action=redirect&bpo=46221]: **`PREP_RERAISE_STAR`** no longer pushes `lasti` to the stack.
- bpo-46202 [https://bugs.python.org/issue?@action=redirect&bpo=46202]: Remove **`POP_EXCEPT_AND_RERAISE`** and replace it by an equivalent sequence of other opcodes.

Library

- bpo-46085 [https://bugs.python.org/issue?@action=redirect&bpo=46085]: Fix iterator cache mechanism of **`OrderedDict`**.

Core and Builtins

- bpo-46055 [https://bugs.python.org/issue?@action=redirect&bpo=46055]: Speed up shifting operation involving integers less than **`PyLong_BASE`**. Patch by Xinhang Xu.

- bpo-46110 [https://bugs.python.org/issue?

@action=redirect&bpo=46110]: Add a maximum recursion check to the PEG parser to avoid stack overflow. Patch by Pablo Galindo

- bpo-46107 [https://bugs.python.org/issue?@action=redirect&bpo=46107]: Fix bug where **`ExceptionGroup.split()`** and **`ExceptionGroup.subgroup()`** did not copy the exception group's `__note__` field to the parts.

- bpo-45711 [https://bugs.python.org/issue?@action=redirect&bpo=45711]: The interpreter state's representation of handled exceptions (a.k.a exc_info, or _PyErr_StackItem) now has only the `exc_value` field, `exc_type` and `exc_traceback` have been removed as their values can be derived from `exc_value`.

- bpo-44525 [https://bugs.python.org/issue?@action=redirect&bpo=44525]: Replace the four call bytecode instructions which one pre-call instruction and two call instructions.

  Removes `CALL_FUNCTION`, `CALL_FUNCTION_KW`, `CALL_METHOD` and `CALL_METHOD_KW`.

  Adds `CALL_NO_KW` and `CALL_KW` call instructions, and `PRECALL_METHOD` prefix for pairing with `LOAD_METHOD`.

- bpo-46039 [https://bugs.python.org/issue?@action=redirect&bpo=46039]: Remove the `YIELD_FROM` instruction and replace it with the `SEND` instruction which performs the same operation, but without the loop.

- bpo-45635 [https://bugs.python.org/issue?@action=redirect&bpo=45635]: The code called from **`_PyErr_Display()`** was refactored to improve error handling. It now exits immediately upon an unrecoverable error.

- bpo-46054 [https://bugs.python.org/issue?@action=redirect&bpo=46054]: Fix parser error when parsing non-utf8 characters in source files. Patch by Pablo Galindo.

- bpo-46042 [https://bugs.python.org/issue?@action=redirect&bpo=46042]: Improve the location of the caret in `SyntaxError` exceptions emitted by the symbol table. Patch by Pablo Galindo.
- bpo-46049 [https://bugs.python.org/issue?@action=redirect&bpo=46049]: Ensure `._pth` files work as intended on platforms other than Windows.
- bpo-46048 [https://bugs.python.org/issue?@action=redirect&bpo=46048]: Fixes parsing of `._pth` files on startup so that single-character paths are correctly read.
- bpo-37971 [https://bugs.python.org/issue?@action=redirect&bpo=37971]: Fix a bug where the line numbers given in a traceback when a decorator application raised an exception were wrong.
- bpo-46031 [https://bugs.python.org/issue?@action=redirect&bpo=46031]: Add `POP_JUMP_IF_NOT_NONE` and `POP_JUMP_IF_NONE` opcodes to speed up conditional jumps.
- bpo-45654 [https://bugs.python.org/issue?@action=redirect&bpo=45654]: Deepfreeze `runpy`, patch by Kumar Aditya.
- bpo-46025 [https://bugs.python.org/issue?@action=redirect&bpo=46025]: Fix a crash in the `atexit` module involving functions that unregister themselves before raising exceptions. Patch by Pablo Galindo.

Library

- bpo-46000 [https://bugs.python.org/issue?@action=redirect&bpo=46000]: Improve compatibility of the `curses` module with NetBSD curses.

Core and Builtins

- bpo-44525 [https://bugs.python.org/issue?@action=redirect&bpo=44525]: Specialize the CALL_FUNCTION instruction for calls to builtin types with a single argument. Speeds up `range(x)`, `list(x)`, and specifically `type(obj)`.
- bpo-42918 [https://bugs.python.org/issue?@action=redirect&bpo=42918]: Fix bug where the built-in **`compile()`** function did not always raise a **`SyntaxError`** when passed multiple statements in “single” mode. Patch by Weipeng Hong.
- bpo-45953 [https://bugs.python.org/issue?@action=redirect&bpo=45953]: The main interpreter in _PyRuntimeState.interpreters is now statically allocated (as part of _PyRuntime). Likewise for the initial thread state of each interpreter. This means less allocation during runtime init, as well as better memory locality for these key state objects.
- bpo-45292 [https://bugs.python.org/issue?@action=redirect&bpo=45292]: Complete the **PEP 654** [https://peps.python.org/pep-0654/] implementation: add `except*`.
- bpo-43413 [https://bugs.python.org/issue?@action=redirect&bpo=43413]: Revert changes in `set.__init__`. Subclass of **`set`** needs to define a `__init__()` method if it defines a `__new__()` method with additional keyword parameters.
- bpo-43931 [https://bugs.python.org/issue?@action=redirect&bpo=43931]: Added the **`Py_Version`** constant which bears the same value as **`PY_VERSION_HEX`**. Patch by Gabriele N. Tornetta.

Library

- bpo-46342 [https://bugs.python.org/issue?@action=redirect&bpo=46342]: The `@typing.final` decorator now sets the `__final__` attribute on the decorated object to allow runtime introspection. Patch by Jelle Zijlstra.

- bpo-46328 [https://bugs.python.org/issue?@action=redirect&bpo=46328]: Added the **`sys.exception()`** method which returns the active exception instance.

- bpo-46307 [https://bugs.python.org/issue?@action=redirect&bpo=46307]: Add `string.Template.is_valid()` and `string.Template.get_identifiers()` methods.

- bpo-46306 [https://bugs.python.org/issue?@action=redirect&bpo=46306]: Assume that `types.CodeType` always has `types.CodeType.co_firstlineno` in `doctest`.

- bpo-40479 [https://bugs.python.org/issue?@action=redirect&bpo=40479]: Fix `hashlib` *usedforsecurity* option to work correctly with OpenSSL 3.0.0 in FIPS mode.

- bpo-46070 [https://bugs.python.org/issue?@action=redirect&bpo=46070]: Fix possible segfault when importing the `asyncio` module from different sub-interpreters in parallel. Patch by Erlend E. Aasland.

- bpo-46244 [https://bugs.python.org/issue?@action=redirect&bpo=46244]: Removed `__slots__` from `typing.ParamSpec` and `typing.TypeVar`. They served no purpose. Patch by Arie Bovenberg.

- bpo-46278 [https://bugs.python.org/issue?@action=redirect&bpo=46278]: Reflect `context` argument in `AbstractEventLoop.call_*()` methods. Loop implementations already support it.

- bpo-46269 [https://bugs.python.org/issue?@action=redirect&bpo=46269]: Remove special-casing of `__new__` in `enum.Enum.__dir__()`.

- bpo-46266 [https://bugs.python.org/issue?@action=redirect&bpo=46266]: Improve day constants in `calendar`.

  Now all constants (`MONDAY` … `SUNDAY`) are documented, tested, and added to `__all__`.

- bpo-46257 [https://bugs.python.org/issue?

@action=redirect&bpo=46257]: Optimized the mean, variance, and stdev functions in the statistics module. If the input is an iterator, it is consumed in a single pass rather than eating memory by conversion to a list. The single pass algorithm is about twice as fast as the previous two pass code.

- bpo-41011 [https://bugs.python.org/issue?@action=redirect&bpo=41011]: Added two new variables to *pyvenv.cfg* which is generated by **`venv`** module: *executable* for the executable and *command* for the command line used to create the environment.
- bpo-46239 [https://bugs.python.org/issue?@action=redirect&bpo=46239]: Improve error message when importing **`asyncio.windows_events`** on non-Windows.
- bpo-46238 [https://bugs.python.org/issue?@action=redirect&bpo=46238]: Reuse `_winapi` constants in `asyncio.windows_events`.
- bpo-46222 [https://bugs.python.org/issue?@action=redirect&bpo=46222]: Adding `SF_NOCACHE` sendfile constant for FreeBSD for the posixmodule.
- bpo-37295 [https://bugs.python.org/issue?@action=redirect&bpo=37295]: Add fast path for `0 <= k <= n <= 67` for **`math.comb()`**.
- bpo-46176 [https://bugs.python.org/issue?@action=redirect&bpo=46176]: Adding the `MAP_STACK` constant for the mmap module.
- bpo-43424 [https://bugs.python.org/issue?@action=redirect&bpo=43424]: Deprecate **`webbrowser.MacOSXOSAScript._name`** and use `name` instead.
- bpo-45321 [https://bugs.python.org/issue?@action=redirect&bpo=45321]: Added missing error codes to module `xml.parsers.expat.errors`.

- bpo-46125 [https://bugs.python.org/issue?@action=redirect&bpo=46125]: Refactor tests to test traversable API directly. Includes changes from importlib 5.4.0.
- bpo-46118 [https://bugs.python.org/issue?@action=redirect&bpo=46118]: Moved importlib.resources and its related functionality to a package.
- bpo-37578 [https://bugs.python.org/issue?@action=redirect&bpo=37578]: Add *include_hidden* parameter to `glob()` and `iglob()` to match hidden files and directories when using special characters like `*`, `**`, `?` and `[]`.
- bpo-20369 [https://bugs.python.org/issue?@action=redirect&bpo=20369]: `concurrent.futures.wait()` no longer blocks forever when given duplicate Futures. Patch by Kumar Aditya.
- bpo-46105 [https://bugs.python.org/issue?@action=redirect&bpo=46105]: Honor spec when generating requirement specs with urls and extras (importlib_metadata 4.8.3).
- bpo-44893 [https://bugs.python.org/issue?@action=redirect&bpo=44893]: EntryPoint objects are no longer tuples. Recommended means to access is by attribute (".name", ".group") or accessor (".load()"). Access by index is deprecated and will raise deprecation warning.
- bpo-22815 [https://bugs.python.org/issue?@action=redirect&bpo=22815]: Print unexpected successes together with failures and errors in summary in `unittest.TextTestResult`.
- bpo-22047 [https://bugs.python.org/issue?@action=redirect&bpo=22047]: Calling `add_argument_group()` on an argument group is deprecated. Calling `add_argument_group()` or `add_mutually_exclusive_group()` on a mutually exclusive group is deprecated.

These features were never supported and do not always work correctly. The functions exist on the API by accident through inheritance and will be removed in the future.

- bpo-26952 [https://bugs.python.org/issue?@action=redirect&bpo=26952]: **`argparse`** raises **`ValueError`** with clear message when trying to render usage for an empty mutually exclusive group. Previously it raised a cryptic **`IndexError`**.
- bpo-45615 [https://bugs.python.org/issue?@action=redirect&bpo=45615]: Functions in the **`traceback`** module raise **`TypeError`** rather than **`AttributeError`** when an exception argument is not of type **`BaseException`**.
- bpo-16594 [https://bugs.python.org/issue?@action=redirect&bpo=16594]: Add allow allow_reuse_port flag in socketserver.
- bpo-27718 [https://bugs.python.org/issue?@action=redirect&bpo=27718]: Fix help for the **`signal`** module. Some functions (e.g. `signal()` and `getsignal()`) were omitted.
- bpo-46032 [https://bugs.python.org/issue?@action=redirect&bpo=46032]: The `registry()` method of **`functools.singledispatch()`** functions checks now the first argument or the first parameter annotation and raises a TypeError if it is not supported. Previously unsupported «types» were ignored (e.g. `typing.List[int]`) or caused an error at calling time (e.g. `list[int]`).
- bpo-46014 [https://bugs.python.org/issue?@action=redirect&bpo=46014]: Add ability to use `typing.Union` and `types.UnionType` as dispatch argument to `functools.singledispatch`. Patch provided by Yurii Karabas.
- bpo-27062 [https://bugs.python.org/issue?@action=redirect&bpo=27062]: Add **`__all__`** to **`inspect`**,

patch by Kumar Aditya.

- bpo-46018 [https://bugs.python.org/issue?@action=redirect&bpo=46018]: Ensure that `math.expm1()` does not raise on underflow.

- bpo-46016 [https://bugs.python.org/issue?@action=redirect&bpo=46016]: Adding **`F_DUP2FD`** and **`F_DUP2FD_CLOEXEC`** constants from FreeBSD into the `fcntl` module.

- bpo-45755 [https://bugs.python.org/issue?@action=redirect&bpo=45755]: `typing` generic aliases now reveal the class attributes of the original generic class when passed to `dir()`. This was the behavior up to Python 3.6, but was changed in 3.7-3.9.

- bpo-45874 [https://bugs.python.org/issue?@action=redirect&bpo=45874]: The empty query string, consisting of no query arguments, is now handled correctly in `urllib.parse.parse_qsl`. This caused problems before when strict parsing was enabled.

- bpo-44674 [https://bugs.python.org/issue?@action=redirect&bpo=44674]: Change how dataclasses disallows mutable default values. It used to use a list of known types (list, dict, set). Now it disallows unhashable objects to be defaults. It's using unhashability as a proxy for mutability. Patch by Eric V. Smith, idea by Raymond Hettinger.

- bpo-23882 [https://bugs.python.org/issue?@action=redirect&bpo=23882]: Remove namespace package (PEP 420) support from unittest discovery. It was introduced in Python 3.4 but has been broken since Python 3.7.

- bpo-25066 [https://bugs.python.org/issue?@action=redirect&bpo=25066]: Added a **`__repr__()`** method to `multiprocessing.Event` objects, patch by Kumar Aditya.

- bpo-45643 [https://bugs.python.org/issue?@action=redirect&bpo=45643]: Added `signal.SIGSTKFLT` on platforms where this signal is defined.

- bpo-44092 [https://bugs.python.org/issue?@action=redirect&bpo=44092]: Fetch across rollback no longer raises `InterfaceError`. Instead we leave it to the SQLite library to handle these cases. Patch by Erlend E. Aasland.

- bpo-42413 [https://bugs.python.org/issue?@action=redirect&bpo=42413]: Replace `concurrent.futures.TimeoutError` and `asyncio.TimeoutError` with builtin `TimeoutError`, keep these names as deprecated aliases.

Documentation

- bpo-46196 [https://bugs.python.org/issue?@action=redirect&bpo=46196]: Document method `cmd.Cmd.columnize()`.
- bpo-46120 [https://bugs.python.org/issue?@action=redirect&bpo=46120]: State that `|` is preferred for readability over `Union` in the `typing` docs.
- bpo-46109 [https://bugs.python.org/issue?@action=redirect&bpo=46109]: Extracted `importlib.resources` and `importlib.resources.abc` documentation into separate files.
- bpo-19737 [https://bugs.python.org/issue?@action=redirect&bpo=19737]: Update the documentation for the `globals()` function.

Tests

- bpo-46296 [https://bugs.python.org/issue?@action=redirect&bpo=46296]: Add a test case for `enum` with `_use_args_ == True` and `_member_type_ == object`.
- bpo-46205 [https://bugs.python.org/issue?@action=redirect&bpo=46205]: Fix hang in runtest_mp due to race condition
- bpo-46263 [https://bugs.python.org/issue?@action=redirect&bpo=46263]: Fix test_capi on FreeBSD 14-dev:

instruct jemalloc to not fill freed memory with junk byte.

- bpo-46262 [https://bugs.python.org/issue?@action=redirect&bpo=46262]: Cover `ValueError` path in tests for **`enum.Flag._missing_()`**.
- bpo-46150 [https://bugs.python.org/issue?@action=redirect&bpo=46150]: Now `fakename` in `test_pathlib.PosixPathTest.test_expanduser` is checked to be non-existent.
- bpo-46129 [https://bugs.python.org/issue?@action=redirect&bpo=46129]: Rewrite `asyncio.locks` tests with **`unittest.IsolatedAsyncioTestCase`** usage.
- bpo-23819 [https://bugs.python.org/issue?@action=redirect&bpo=23819]: Fixed **`asyncio`** tests in python optimized mode. Patch by Kumar Aditya.
- bpo-46114 [https://bugs.python.org/issue?@action=redirect&bpo=46114]: Fix test case for OpenSSL 3.0.1 version. OpenSSL 3.0 uses `0xMNN00PP0L`.

Build

- bpo-44133 [https://bugs.python.org/issue?@action=redirect&bpo=44133]: When Python is configured with **`--without-static-libpython`**, the Python static library (libpython.a) is no longer built. Patch by Victor Stinner.
- bpo-44133 [https://bugs.python.org/issue?@action=redirect&bpo=44133]: When Python is built without **`--enable-shared`**, the `python` program is now linked to object files, rather than being linked to the Python static library (libpython.a), to make sure that all symbols are exported. Previously, the linker omitted some symbols like the **`Py_FrozenMain()`** function. Patch by Victor Stinner.
- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: The `configure` script has a new option `--with-emscripten-target` to select browser or node as Emscripten build target.
- bpo-46315 [https://bugs.python.org/issue?@action=redirect&bpo=46315]: Added and fixed `#ifdef HAVE_FEATURE` checks for functionality that is not available on WASI platform.
- bpo-45723 [https://bugs.python.org/issue?

@action=redirect&bpo=45723]: Fixed a regression in `configure` check for **`select.epoll()`**.
- bpo-46263 [https://bugs.python.org/issue?@action=redirect&bpo=46263]: `configure` no longer sets `MULTIARCH` on FreeBSD platforms.
- bpo-46106 [https://bugs.python.org/issue?@action=redirect&bpo=46106]: Updated OpenSSL to 1.1.1m in Windows builds, macOS installer builds, and CI. Patch by Kumar Aditya.
- bpo-46088 [https://bugs.python.org/issue?@action=redirect&bpo=46088]: Automatically detect or install bootstrap Python runtime when building from Visual Studio.
- bpo-46072 [https://bugs.python.org/issue?@action=redirect&bpo=46072]: Add a –with-pystats configure option to turn on internal statistics gathering.
- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: A new directory `Tools/wasm` contains WebAssembly-related helpers like `config.site` override for wasm32-emscripten, wasm assets generator to bundle the stdlib, and a README.
- bpo-46023 [https://bugs.python.org/issue?@action=redirect&bpo=46023]: **makesetup** no longer builds extensions that have been marked as *disabled*. This allows users to disable modules in `Modules/Setup.local`.
- bpo-45949 [https://bugs.python.org/issue?@action=redirect&bpo=45949]: Use pure Python `freeze_module` for all but importlib bootstrap files. `--with-freeze-module` **configure** option is no longer needed for cross builds.

Windows

- bpo-46217 [https://bugs.python.org/issue?@action=redirect&bpo=46217]: Removed parameter that is unsupported on Windows 8.1 and early Windows 10 and may have caused build or runtime failures.

macOS

- bpo-40477 [https://bugs.python.org/issue?

@action=redirect&bpo=40477]: The Python Launcher app for macOS now properly launches scripts and, if necessary, the Terminal app when running on recent macOS releases.

C API

- bpo-46236 [https://bugs.python.org/issue?@action=redirect&bpo=46236]: Fix a bug in **`PyFunction_GetAnnotations()`** that caused it to return a `tuple` instead of a `dict`.
- bpo-46140 [https://bugs.python.org/issue?@action=redirect&bpo=46140]: **`PyBuffer_GetPointer()`**, **`PyBuffer_FromContiguous()`**, **`PyBuffer_ToContiguous()`** and **`PyMemoryView_FromBuffer()`** now take buffer info by `const Py_buffer *` instead of `Py_buffer *`, as they do not need mutability. **`PyBuffer_FromContiguous()`** also now takes the source buffer as `const void *`, and similarly **`PyBuffer_GetPointer()`** takes the strides as `const Py_ssize_t *`.
- bpo-45855 [https://bugs.python.org/issue?@action=redirect&bpo=45855]: Document that the *no_block* argument to **`PyCapsule_Import()`** is a no-op now.
- bpo-45855 [https://bugs.python.org/issue?@action=redirect&bpo=45855]: Replaced deprecated usage of **`PyImport_ImportModuleNoBlock()`** with **`PyImport_ImportModule()`** in stdlib modules. Patch by Kumar Aditya.
- bpo-46007 [https://bugs.python.org/issue?@action=redirect&bpo=46007]: The **`PyUnicode_CHECK_INTERNED()`** macro has been excluded from the limited C API. It was never usable there, because it used internal structures which are not available in the limited C API. Patch by Victor Stinner.

Python 3.11.0 alpha 3

Release date: 2021-12-08

Core and Builtins

- bpo-46009 [https://bugs.python.org/issue?@action=redirect&bpo=46009]: Restore behavior from 3.9 and earlier when sending non-None to newly started generator. In 3.9 this did not affect the state of the generator. In 3.10.0 and 3.10.1 `gen_func().send(0)` is equivalent to `gen_func().throw(TypeError(...)` which exhausts the generator. In 3.10.2 onward, the behavior has been reverted to that of 3.9.
- bpo-46004 [https://bugs.python.org/issue?@action=redirect&bpo=46004]: Fix the **`SyntaxError`** location for errors involving for loops with invalid targets. Patch by Pablo Galindo
- bpo-45711 [https://bugs.python.org/issue?@action=redirect&bpo=45711]: **`_PyErr_ChainStackItem()`** no longer normalizes `exc_info` (including setting the traceback on the exception instance) because `exc_info` is always normalized.
- bpo-45607 [https://bugs.python.org/issue?@action=redirect&bpo=45607]: The `__note__` field was added to **`BaseException`**. It is `None` by default but can be set to a string which is added to the exception’s traceback.
- bpo-45947 [https://bugs.python.org/issue?@action=redirect&bpo=45947]: Place pointers to dict and values immediately before GC header. This reduces number of dependent memory loads to access either dict or values from 3 to 1.
- bpo-45915 [https://bugs.python.org/issue?@action=redirect&bpo=45915]: `is_valid_fd` now uses faster `fcntl(fd, F_GETFD)` on Linux, macOS, and Windows.
- bpo-44530 [https://bugs.python.org/issue?@action=redirect&bpo=44530]: Reverts a change to the `code.__new__` audit event from an earlier prerelease.
- bpo-42268 [https://bugs.python.org/issue?@action=redirect&bpo=42268]: Fail the configure step if the selected compiler doesn’t support memory sanitizer. Patch by Pablo Galindo
- bpo-45711 [https://bugs.python.org/issue?@action=redirect&bpo=45711]: The three values of `exc_info`

are now always consistent with each other. In particular, the `type` and `traceback` fields are now derived from the exception instance. This impacts the return values of **`sys.exc_info()`** and **`PyErr_GetExcInfo()`** if the exception instance is modified while the exception is handled, as well as **`PyErr_SetExcInfo()`**, which now ignores the `type` and `traceback` arguments provided to it.

- bpo-45727 [https://bugs.python.org/issue?@action=redirect&bpo=45727]: Refine the custom syntax error that suggests that a comma may be missing to trigger only when the expressions are detected between parentheses or brackets. Patch by Pablo Galindo
- bpo-45885 [https://bugs.python.org/issue?@action=redirect&bpo=45885]: Specialized the `COMPARE_OP` opcode using the PEP 659 machinery.
- bpo-45786 [https://bugs.python.org/issue?@action=redirect&bpo=45786]: Allocate space for the interpreter frame in the frame object, to avoid an additional allocation when the frame object outlives the frame activation.

Library

- bpo-45614 [https://bugs.python.org/issue?@action=redirect&bpo=45614]: Fix **`traceback`** display for exceptions with invalid module name.

Core and Builtins

- bpo-45813 [https://bugs.python.org/issue?@action=redirect&bpo=45813]: Fix crash when calling coro.cr_frame.clear() after coroutine has been freed.

- bpo-45811 [https://bugs.python.org/issue?@action=redirect&bpo=45811]: Improve the tokenizer errors when encountering invisible control characters in the parser. Patch by Pablo Galindo

- bpo-45848 [https://bugs.python.org/issue?@action=redirect&bpo=45848]: Allow the parser to obtain error lines directly from encoded files. Patch by Pablo Galindo

- bpo-45709 [https://bugs.python.org/issue?@action=redirect&bpo=45709]: Restore behavior from 3.10 when tracing an exception raised within a with statement.

- bpo-44525 [https://bugs.python.org/issue?@action=redirect&bpo=44525]: Adds new `COPY_FREE_VARS` opcode, to make copying of free variables from function to frame explicit. Helps optimization of calls to Python function.

- bpo-45829 [https://bugs.python.org/issue?@action=redirect&bpo=45829]: Specialize `BINARY_SUBSCR` for classes with a `__getitem__` method implemented in Python

- bpo-45826 [https://bugs.python.org/issue?@action=redirect&bpo=45826]: Fixed a crash when calling `.with_traceback(None)` on `NameError`. This occurs internally in `unittest.TestCase.assertRaises()`.

- bpo-45822 [https://bugs.python.org/issue?@action=redirect&bpo=45822]: Fixed a bug in the parser that was causing it to not respect **PEP 263** [https://peps.python.org/pep-0263/] coding cookies when no flags are provided. Patch by Pablo Galindo

- bpo-45820 [https://bugs.python.org/issue?@action=redirect&bpo=45820]: Fix a segfault when the parser fails without reading any input. Patch by Pablo Galindo

- bpo-45636 [https://bugs.python.org/issue?@action=redirect&bpo=45636]: Simplify the implementation of `BINARY_OP` by indexing into an array of function pointers (rather than switching on the oparg).

- bpo-42540 [https://bugs.python.org/issue?@action=redirect&bpo=42540]: Fix crash when `os.fork()` is called with an active non-default memory allocator.

- bpo-45738 [https://bugs.python.org/issue?@action=redirect&bpo=45738]: Fix computation of error location for invalid continuation characters in the parser. Patch by Pablo Galindo.

- bpo-45636 [https://bugs.python.org/issue?@action=redirect&bpo=45636]: Remove an existing «fast path» for old-style string formatting, since it no longer appears to have any measurable impact.

- bpo-45753 [https://bugs.python.org/issue?@action=redirect&bpo=45753]: Make recursion checks a bit more efficient by tracking amount of calls left before overflow.

- bpo-45773 [https://bugs.python.org/issue?@action=redirect&bpo=45773]: Fix a compiler hang when attempting to optimize certain jump patterns.

- bpo-45764 [https://bugs.python.org/issue?@action=redirect&bpo=45764]: The parser now gives a better error message when leaving out the opening parenthesis `(` after a `def`-statement:

  ```
  >>> def f:
    File "<stdin>", line 1
      def f:
           ^
  SyntaxError: expected '('
  ```

- bpo-45609 [https://bugs.python.org/issue?@action=redirect&bpo=45609]: Specialized the `STORE_SUBSCR` opcode using the PEP 659 machinery.

- bpo-45636 [https://bugs.python.org/issue?@action=redirect&bpo=45636]: Replace all numeric `BINARY_*` and `INPLACE_*` instructions with a single **`BINARY_OP`** implementation.

- bpo-45582 [https://bugs.python.org/issue?@action=redirect&bpo=45582]: Path calculation (known as `getpath`) has been reimplemented as a frozen Python module. This should have no visible impact, but may affect calculation of all paths referenced in **`sys`** and **`sysconfig`**.

- bpo-45450 [https://bugs.python.org/issue?@action=redirect&bpo=45450]: Improve the syntax error message for parenthesized arguments. Patch by Pablo Galindo.

Library

- bpo-27946 [https://bugs.python.org/issue?@action=redirect&bpo=27946]: Fix possible crash when getting an attribute of **`xml.etree.ElementTree.Element`** simultaneously with replacing the `attrib` dict.

- bpo-45711 [https://bugs.python.org/issue?@action=redirect&bpo=45711]: Make **`asyncio`** normalize exceptions as soon as they are captured with **`PyErr_Fetch()`**, and before they are stored as an exc_info triplet. This brings **`asyncio`** in line with the rest of the codebase, where an exc_info triplet is always normalized.

- bpo-23819 [https://bugs.python.org/issue?@action=redirect&bpo=23819]: Replaced asserts with exceptions in asyncio, patch by Kumar Aditya.

- bpo-13236 [https://bugs.python.org/issue?@action=redirect&bpo=13236]: **`unittest.TextTestResult`** and **`unittest.TextTestRunner`** flush now the output stream more often.

- bpo-45917 [https://bugs.python.org/issue?@action=redirect&bpo=45917]: Added **`math.exp2()`**:, which returns 2 raised to the power of x.

- bpo-37658 [https://bugs.python.org/issue?@action=redirect&bpo=37658]: Fix issue when on certain conditions `asyncio.wait_for()` may allow a coroutine to complete successfully, but fail to return the result, potentially causing memory leaks or other issues.

- bpo-45876 [https://bugs.python.org/issue?@action=redirect&bpo=45876]: Improve the accuracy of stdev() and pstdev() in the statistics module. When the inputs are floats or fractions, the output is a correctly rounded float

- bpo-44649 [https://bugs.python.org/issue?

@action=redirect&bpo=44649]: Handle dataclass(slots=True) with a field that has default a default value, but for which init=False.

- bpo-45803 [https://bugs.python.org/issue?@action=redirect&bpo=45803]: Added missing kw_only parameter to dataclasses.make_dataclass().

- bpo-45837 [https://bugs.python.org/issue?@action=redirect&bpo=45837]: The **`turtle.RawTurtle.settiltangle()`** is deprecated since Python 3.1, it now emits a deprecation warning and will be removed in Python 3.13.

  Use **`turtle.RawTurtle.tiltangle()`** instead.

  `turtle.RawTurtle.tiltangle()` was earlier incorrectly marked as deprecated, its docstring has been corrected.

  Patch by Hugo van Kemenade.

- bpo-45831 [https://bugs.python.org/issue?@action=redirect&bpo=45831]: **`faulthandler`** can now write ASCII-only strings (like filenames and function names) with a single write() syscall when dumping a traceback. It reduces the risk of getting an unreadable dump when two threads or two processes dump a traceback to the same file (like stderr) at the same time. Patch by Victor Stinner.

- bpo-45828 [https://bugs.python.org/issue?@action=redirect&bpo=45828]: **`sqlite`** C callbacks now use unraisable exceptions if callback tracebacks are enabled. Patch by Erlend E. Aasland.

- bpo-41735 [https://bugs.python.org/issue?@action=redirect&bpo=41735]: Fix thread lock in `zlib.Decompress.flush()` method before `PyObject_GetBuffer`.

- bpo-45235 [https://bugs.python.org/issue?@action=redirect&bpo=45235]: Reverted an argparse bugfix that caused regression in the handling of default arguments for subparsers. This prevented leaf level arguments from taking precedence over root level arguments.

- bpo-45754 [https://bugs.python.org/issue?@action=redirect&bpo=45754]: Fix a regression in Python 3.11a1 and 3.11a2 where `sqlite3` incorrectly would use `SQLITE_LIMIT_LENGTH` when checking SQL statement lengths. Now, `SQLITE_LIMIT_SQL_LENGTH` is used. Patch by Erlend E. Aasland.
- bpo-45766 [https://bugs.python.org/issue?@action=redirect&bpo=45766]: Added *proportional* option to `statistics.linear_regression()`.
- bpo-45765 [https://bugs.python.org/issue?@action=redirect&bpo=45765]: In importlib.metadata, fix distribution discovery for an empty path.
- bpo-45757 [https://bugs.python.org/issue?@action=redirect&bpo=45757]: Fix bug where `dis` produced an incorrect oparg when `EXTENDED_ARG` is followed by an opcode that does not use its argument.
- bpo-45644 [https://bugs.python.org/issue?@action=redirect&bpo=45644]: In-place JSON file formatting using `python3 -m json.tool infile infile` now works correctly, previously it left the file empty. Patch by Chris Wesseling.
- bpo-45703 [https://bugs.python.org/issue?@action=redirect&bpo=45703]: When a namespace package is imported before another module from the same namespace is created/installed in a different `sys.path` location while the program is running, calling the `importlib.invalidate_caches()` function will now also guarantee the new module is noticed.
- bpo-45535 [https://bugs.python.org/issue?@action=redirect&bpo=45535]: Improve output of `dir()` with Enums.

- bpo-45664 [https://bugs.python.org/issue?@action=redirect&bpo=45664]: Fix `types.resolve_bases()` and `types.new_class()` for `types.GenericAlias` instance as a base.
- bpo-45663 [https://bugs.python.org/issue?@action=redirect&bpo=45663]: Fix `dataclasses.is_dataclass()` for dataclasses which are subclasses of `types.GenericAlias`.
- bpo-45662 [https://bugs.python.org/issue?@action=redirect&bpo=45662]: Fix the repr of `dataclasses.InitVar` with a type alias to the built-in class, e.g. `InitVar[list[int]]`.
- bpo-43137 [https://bugs.python.org/issue?@action=redirect&bpo=43137]: Launch GNOME web browsers via gio tool instead of obsolete gvfs-open
- bpo-45429 [https://bugs.python.org/issue?@action=redirect&bpo=45429]: On Windows, `time.sleep()` now uses a waitable timer which supports high-resolution timers. Patch by Donghee Na and Eryk Sun.
- bpo-37295 [https://bugs.python.org/issue?@action=redirect&bpo=37295]: Optimize `math.comb()` and `math.perm()`.
- bpo-45514 [https://bugs.python.org/issue?@action=redirect&bpo=45514]: Deprecated legacy functions in `importlib.resources`.
- bpo-45507 [https://bugs.python.org/issue?@action=redirect&bpo=45507]: Add tests for truncated/missing trailers in gzip.decompress implementation.
- bpo-45359 [https://bugs.python.org/issue?@action=redirect&bpo=45359]: Implement **PEP 585** [https://peps.python.org/pep-0585/] for `graphlib.TopologicalSorter`.

- bpo-44733 [https://bugs.python.org/issue?@action=redirect&bpo=44733]: Add `max_tasks_per_child` to **`concurrent.futures.ProcessPoolExecutor`**. This allows users to specify the maximum number of tasks a single process should execute before the process needs to be restarted.

- bpo-28806 [https://bugs.python.org/issue?@action=redirect&bpo=28806]: Improve netrc library. netrc file no longer needs to contain all tokens. And if the login name is anonymous, security check is no longer need.

- bpo-43498 [https://bugs.python.org/issue?@action=redirect&bpo=43498]: Avoid a possible *«RuntimeError: dictionary changed size during iteration»* when adjusting the process count of **`ProcessPoolExecutor`**.

- bpo-42158 [https://bugs.python.org/issue?@action=redirect&bpo=42158]: Add MIME types for N-quads, N-triples, Notation3 and TriG to `mimetypes`.

- bpo-30533 [https://bugs.python.org/issue?@action=redirect&bpo=30533]: Add **`inspect.getmembers_static()`** , it return all members without triggering dynamic lookup via the descriptor protocol. Patch by Weipeng Hong.

Documentation

- bpo-42238 [https://bugs.python.org/issue?@action=redirect&bpo=42238]: `make -C Doc suspicious` will be removed soon in favor of `make -C Doc check`, mark it as deprecated.
- bpo-45840 [https://bugs.python.org/issue?@action=redirect&bpo=45840]: Improve cross-references in the documentation for the data model.
- bpo-45640 [https://bugs.python.org/issue?@action=redirect&bpo=45640]: Properly marked-up grammar tokens in the documentation are now clickable and take you to the definition of a given piece of grammar. Patch by Arthur Milchior.

- bpo-45788 [https://bugs.python.org/issue?@action=redirect&bpo=45788]: Link doc for sys.prefix to sysconfig doc on installation paths.
- bpo-45772 [https://bugs.python.org/issue?@action=redirect&bpo=45772]: `socket.socket` documentation is corrected to a class from a function.
- bpo-45392 [https://bugs.python.org/issue?@action=redirect&bpo=45392]: Update the docstring of the **`type`** built-in to remove a redundant line and to mention keyword arguments for the constructor.
- bpo-45250 [https://bugs.python.org/issue?@action=redirect&bpo=45250]: Update the documentation to note that CPython does not consistently require iterators to define `__iter__`.
- bpo-25381 [https://bugs.python.org/issue?@action=redirect&bpo=25381]: In the extending chapter of the extending doc, update a paragraph about the global variables containing exception information.
- bpo-43905 [https://bugs.python.org/issue?@action=redirect&bpo=43905]: Expanded **`astuple()`** and **`asdict()`** docs, warning about deepcopy being applied and providing a workaround.

Tests

- bpo-45695 [https://bugs.python.org/issue?@action=redirect&bpo=45695]: Out-of-tree builds with a read-only source directory are now tested by CI.
- bpo-19460 [https://bugs.python.org/issue?@action=redirect&bpo=19460]: Add new Test for `Lib/email/mime/nonmultipart.py::MIMENonMultipart`.
- bpo-45835 [https://bugs.python.org/issue?@action=redirect&bpo=45835]: Fix race condition in test_queue tests with multiple «feeder» threads.
- bpo-45783 [https://bugs.python.org/issue?@action=redirect&bpo=45783]: The test for the freeze tool now handles file moves and deletions.
- bpo-45745 [https://bugs.python.org/issue?@action=redirect&bpo=45745]: Remove the `--findleaks` command line option of regrtest: use the `--fail-env-`

`changed` option instead. Since Python 3.7, it was a deprecated alias to the `--fail-env-changed` option.

- bpo-45701 [https://bugs.python.org/issue?@action=redirect&bpo=45701]: Add tests with `tuple` type with `functools.lru_cache()` to `test_functools`.

Build

- bpo-44035 [https://bugs.python.org/issue?@action=redirect&bpo=44035]: CI now verifies that autoconf files have been regenerated with a current and unpatched autoconf package.

- bpo-45950 [https://bugs.python.org/issue?@action=redirect&bpo=45950]: The build system now uses a **_bootstrap_python** interpreter for freezing and deepfreezing again. To speed up build process the build tools **_bootstrap_python** and **_freeze_module** are no longer build with LTO.

- bpo-45881 [https://bugs.python.org/issue?@action=redirect&bpo=45881]: The **configure** script now accepts `--with-build-python` and `--with-freeze-module` options to make cross compiling easier.

- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: Emscripten platform now uses `.wasm` suffix by default.

- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: Disable unusable core extension modules on WASM/Emscripten targets.

- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: `configure` now checks for socket `shutdown` function. The check makes it possible to disable `SYS_shutdown` with `ac_cv_func_shutdown=no` in CONFIG_SITE.

- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: `configure` now checks for functions `fork1`, `getegid`, `geteuid`, `getgid`, `getppid`, `getuid`, `opendir`, `pipe`, `system`, `wait`, `ttyname`.

- bpo-33393 [https://bugs.python.org/issue?@action=redirect&bpo=33393]: Update `config.guess` to 2021-06-03 and `config.sub` to 2021-08-14. `Makefile` now has an `update-config` target to make updating more convenient.

- bpo-45866 [https://bugs.python.org/issue?@action=redirect&bpo=45866]: `make regen-all` now produces the same output when run from a directory other than the source tree: when building Python out of the source tree. pegen now strips directory of the «generated by pygen from <FILENAME>» header Patch by Victor Stinner.

- bpo-40280 [https://bugs.python.org/issue?@action=redirect&bpo=40280]: `configure` now accepts machine `wasm32` or `wasm64` and OS `wasi` or `emscripten` for cross building, e.g. `wasm32-unknown-emscripten`, `wasm32-wasi`, or `wasm32-unknown-wasi`.

- bpo-41498 [https://bugs.python.org/issue?@action=redirect&bpo=41498]: Python now compiles on platforms without `sigset_t`. Several functions in **`signal`** are not available when `sigset_t` is missing.

  Based on patch by Roman Yurchak for pyodide.

- bpo-45881 [https://bugs.python.org/issue?@action=redirect&bpo=45881]: `setup.py` now uses `CC` from environment first to discover multiarch and cross compile paths.

- bpo-45886 [https://bugs.python.org/issue?@action=redirect&bpo=45886]: The `_freeze_module` program path can now be overridden on the command line, e.g. `make FREEZE_MODULE=../x86_64/Program/_freeze_module`.

- bpo-45873 [https://bugs.python.org/issue?@action=redirect&bpo=45873]: Get rid of the `_bootstrap_python` build step. The deepfreeze.py script is now run using `$(PYTHON_FOR_REGEN)` which can be Python 3.7 or newer (on Windows, 3.8 or newer).

- bpo-45847 [https://bugs.python.org/issue?@action=redirect&bpo=45847]: Port builtin hashlib extensions to `PY_STDLIB_MOD` macro and `addext()`.

- bpo-45723 [https://bugs.python.org/issue?@action=redirect&bpo=45723]: Add `autoconf` helpers for saving and restoring environment variables:

  - `SAVE_ENV`: Save `$CFLAGS`, `$LDFLAGS`, `$LIBS`, and `$CPPFLAGS`.
  - `RESTORE_ENV`: Restore `$CFLAGS`, `$LDFLAGS`, `$LIBS`, and `$CPPFLAGS`.
  - `WITH_SAVE_ENV([SCRIPT])`: Run `SCRIPT` wrapped with `SAVE_ENV` and `RESTORE_ENV`.

  Patch by Erlend E. Aasland.

- bpo-45573 [https://bugs.python.org/issue?@action=redirect&bpo=45573]: Mandatory core modules, that are required to bootstrap Python, are now in `Modules/Setup.bootstrap`.

- bpo-45573 [https://bugs.python.org/issue?@action=redirect&bpo=45573]: `configure` now creates `Modules/Setup.stdlib` with conditionally enabled/disabled extension module lines. The file is not used, yet.

- bpo-45573 [https://bugs.python.org/issue?@action=redirect&bpo=45573]: `configure` now uses a unified format to set state, compiler flags, and linker flags in Makefile. The new macro `PY_STDLIB_MOD` sets three variables that are consumed by `Modules/Setup` and `setup.py`.

- bpo-45816 [https://bugs.python.org/issue?

@action=redirect&bpo=45816]: Python now supports building with Visual Studio 2022 (MSVC v143, VS Version 17.0). Patch by Jeremiah Vivian.

- bpo-45800 [https://bugs.python.org/issue?@action=redirect&bpo=45800]: Settings for **`pyexpat`** C extension are now detected by `configure`. The bundled `expat` library is built in `Makefile`.
- bpo-45798 [https://bugs.python.org/issue?@action=redirect&bpo=45798]: Settings for **`decimal`** internal C extension are now detected by `configure`. The bundled `libmpdec` library is built in `Makefile`.
- bpo-45723 [https://bugs.python.org/issue?@action=redirect&bpo=45723]: **configure** has a new option `--with-pkg-config` to disable or require pkg-config.
- bpo-45774 [https://bugs.python.org/issue?@action=redirect&bpo=45774]: The build dependencies for **`sqlite3`** are now detected by `configure` and `pkg-config`. Patch by Erlend E. Aasland.
- bpo-45763 [https://bugs.python.org/issue?@action=redirect&bpo=45763]: The build dependencies for **`zlib`**, **`bz2`**, and **`lzma`** are now detected by `configure`.
- bpo-45747 [https://bugs.python.org/issue?@action=redirect&bpo=45747]: gdbm and dbm build dependencies are now detected by `configure`.
- bpo-45743 [https://bugs.python.org/issue?@action=redirect&bpo=45743]: On macOS, the build system no longer passes `search_paths_first` to the linker. The flag has been the default since Xcode 4 / macOS 10.6.
- bpo-45723 [https://bugs.python.org/issue?@action=redirect&bpo=45723]: `configure.ac` is now compatible with autoconf 2.71. Deprecated checks `STDC_HEADERS` and `AC_HEADER_TIME` have been removed.

- bpo-45723 [https://bugs.python.org/issue?@action=redirect&bpo=45723]: `configure` now prints a warning when pkg-config is missing.

- bpo-45731 [https://bugs.python.org/issue?@action=redirect&bpo=45731]: `configure --enable-loadable-sqlite-extensions` is now handled by new `PY_SQLITE_ENABLE_LOAD_EXTENSION` macro instead of logic in setup.py.

- bpo-45723 [https://bugs.python.org/issue?@action=redirect&bpo=45723]: configure.ac now uses custom helper macros and `AC_CACHE_CHECK` to simplify and speed up configure runs.

- bpo-45696 [https://bugs.python.org/issue?@action=redirect&bpo=45696]: Skip the marshal step for frozen modules by generating C code that produces a set of ready-to-use code objects. This speeds up startup time by another 10% or more.

- bpo-45561 [https://bugs.python.org/issue?@action=redirect&bpo=45561]: Run smelly.py tool from $(srcdir).

Windows

- bpo-46105 [https://bugs.python.org/issue?@action=redirect&bpo=46105]: Fixed calculation of **`sys.path`** in a venv on Windows.
- bpo-45901 [https://bugs.python.org/issue?@action=redirect&bpo=45901]: When installed through the Microsoft Store and set as the default app for `*.py` files, command line arguments will now be passed to Python when invoking a script without explicitly launching Python (that is, `script.py args` rather than `python script.py args`).
- bpo-45616 [https://bugs.python.org/issue?@action=redirect&bpo=45616]: Fix Python Launcher's ability to distinguish between versions 3.1 and 3.10 when either one is explicitly requested. Previously, 3.1 would be used if 3.10 was requested but not installed, and 3.10 would be used if 3.1 was requested but 3.10 was installed.

- bpo-45850 [https://bugs.python.org/issue?@action=redirect&bpo=45850]: Implement changes to build with deep-frozen modules on Windows. Note that we now require Python 3.10 as the «bootstrap» or «host» Python.
- bpo-45732 [https://bugs.python.org/issue?@action=redirect&bpo=45732]: Updates bundled Tcl/Tk to 8.6.12.
- bpo-45720 [https://bugs.python.org/issue?@action=redirect&bpo=45720]: Internal reference to `shlwapi.dll` was dropped to help improve startup time. This DLL will no longer be loaded at the start of every Python process.

macOS

- bpo-45732 [https://bugs.python.org/issue?@action=redirect&bpo=45732]: Update python.org macOS installer to use Tcl/Tk 8.6.12.

C API

- bpo-39026 [https://bugs.python.org/issue?@action=redirect&bpo=39026]: Fix Python.h to build C extensions with Xcode: remove a relative include from `Include/cpython/pystate.h`.

Python 3.11.0 alpha 2

Release date: 2021-11-05

Core and Builtins

- bpo-45716 [https://bugs.python.org/issue?@action=redirect&bpo=45716]: Improve the **`SyntaxError`** message when using `True`, `None` or `False` as keywords in a function call. Patch by Pablo Galindo.

- bpo-45688 [https://bugs.python.org/issue?@action=redirect&bpo=45688]: **`sys.stdlib_module_names`** now contains the macOS-specific module **`_scproxy`**.

- bpo-45379 [https://bugs.python.org/issue?@action=redirect&bpo=45379]: Clarify **`ImportError`** message when we try to explicitly import a frozen module but frozen modules are disabled.
- bpo-44525 [https://bugs.python.org/issue?@action=redirect&bpo=44525]: Specialize simple calls to Python functions (no starargs, keyword dict, or closure)
- bpo-45530 [https://bugs.python.org/issue?@action=redirect&bpo=45530]: Cases of sorting using tuples as keys may now be significantly faster in some cases. Patch by Tim Peters.

  The order of the result may differ from earlier releases if the tuple elements don't define a total ordering (see Comparaciones de valor for information on total ordering). It's generally true that the result of sorting simply isn't well-defined in the absence of a total ordering on list elements.
- bpo-45526 [https://bugs.python.org/issue?@action=redirect&bpo=45526]: In obmalloc, set ADDRESS_BITS to not ignore any bits (ignored 16 before). That is safer in the case that the kernel gives user-space virtual addresses that span a range greater than 48 bits.
- bpo-30570 [https://bugs.python.org/issue?@action=redirect&bpo=30570]: Fixed a crash in `issubclass()` from infinite recursion when searching pathological `__bases__` tuples.
- bpo-45521 [https://bugs.python.org/issue?@action=redirect&bpo=45521]: Fix a bug in the obmalloc radix tree code. On 64-bit machines, the bug causes the tree to hold 46-bits of virtual addresses, rather than the intended 48-bits.
- bpo-45494 [https://bugs.python.org/issue?@action=redirect&bpo=45494]: Fix parser crash when reporting errors involving invalid continuation characters. Patch by

Pablo Galindo.

- bpo-45445 [https://bugs.python.org/issue?@action=redirect&bpo=45445]: Python now fails to initialize if it finds an invalid `-X` option in the command line. Patch by Pablo Galindo.

- bpo-45340 [https://bugs.python.org/issue?@action=redirect&bpo=45340]: Object attributes are held in an array instead of a dictionary. An object's dictionary are created lazily, only when needed. Reduces the memory consumption of a typical Python object by about 30%. Patch by Mark Shannon.

- bpo-45408 [https://bugs.python.org/issue?@action=redirect&bpo=45408]: Fix a crash in the parser when reporting tokenizer errors that occur at the same time unclosed parentheses are detected. Patch by Pablo Galindo.

- bpo-29410 [https://bugs.python.org/issue?@action=redirect&bpo=29410]: Add SipHash13 for string hash algorithm and use it by default.

- bpo-45385 [https://bugs.python.org/issue?@action=redirect&bpo=45385]: Fix reference leak from descr_check. Patch by Donghee Na.

- bpo-45367 [https://bugs.python.org/issue?@action=redirect&bpo=45367]: Specialized the `BINARY_MULTIPLY` opcode to `BINARY_MULTIPLY_INT` and `BINARY_MULTIPLY_FLOAT` using the PEP 659 machinery.

- bpo-21736 [https://bugs.python.org/issue?@action=redirect&bpo=21736]: Frozen stdlib modules now have `__file__` to the .py file they would otherwise be loaded from, if possible. For packages, `__path__` now has the correct entry instead of being an empty list, which allows unfrozen submodules to be imported. These are set only if the stdlib directory is known when the runtime is initialized. Note that the file at `__file__` is not guaranteed to exist. None of this affects non-stdlib frozen modules nor, for now, frozen modules imported using `PyImport_ImportFrozenModule()`. Also, at the moment `co_filename` is not updated for the module.

- bpo-45020 [https://bugs.python.org/issue?@action=redirect&bpo=45020]: For frozen stdlib modules, record the original module name as `module.__spec__.loader_state.origname`. If the value is different than `module.__spec__.name` then the module was defined as an alias in Tools/scripts/freeze_modules.py. If it is `None` then the module comes from a source file outside the stdlib.

- bpo-45324 [https://bugs.python.org/issue?@action=redirect&bpo=45324]: In FrozenImporter.find_spec(), we now preserve the information needed in exec_module() to load the module. This change mostly impacts internal details, rather than changing the importer's behavior.

- bpo-45292 [https://bugs.python.org/issue?@action=redirect&bpo=45292]: Implement **PEP 654** [https://peps.python.org/pep-0654/]. Add **`ExceptionGroup`** and **`BaseExceptionGroup`**. Update traceback display code.

- bpo-40116 [https://bugs.python.org/issue?@action=redirect&bpo=40116]: Change to the implementation of split dictionaries. Classes where the instances differ either in the exact set of attributes, or in the order in which those attributes are set, can still share keys. This should have no observable effect on users of Python or the C-API. Patch by Mark Shannon.

- bpo-44050 [https://bugs.python.org/issue?@action=redirect&bpo=44050]: Extensions that indicate they use global state (by setting `m_size` to -1) can again be used in multiple interpreters. This reverts to behavior of Python 3.8.

- bpo-44525 [https://bugs.python.org/issue?@action=redirect&bpo=44525]: Setup initial specialization infrastructure for the `CALL_FUNCTION` opcode. Implemented initial specializations for C function calls:

    - `CALL_FUNCTION_BUILTIN_O` for `METH_O` flag.
    - `CALL_FUNCTION_BUILTIN_FAST` for `METH_FASTCALL` flag without keywords.
    - `CALL_FUNCTION_LEN` for `len(o)`.
    - `CALL_FUNCTION_ISINSTANCE` for `isinstance(o, t)`.

- bpo-44511 [https://bugs.python.org/issue?@action=redirect&bpo=44511]: Improve the generated bytecode for class and mapping patterns.

- bpo-43706 [https://bugs.python.org/issue?@action=redirect&bpo=43706]: Speed up calls to `enumerate()` by using the **PEP 590** [https://peps.python.org/pep-0590/] `vectorcall` calling convention. Patch by Donghee Na.

Library

- bpo-45679 [https://bugs.python.org/issue?@action=redirect&bpo=45679]: Fix caching of multi-value **`typing.Literal`**. `Literal[True, 2]` is no longer equal to `Literal[1, 2]`.

- bpo-42064 [https://bugs.python.org/issue?@action=redirect&bpo=42064]: Convert **`sqlite3`** to multi-phase initialisation (PEP 489). Patches by Erlend E. Aasland.

- bpo-45438 [https://bugs.python.org/issue?@action=redirect&bpo=45438]: Fix typing.Signature string representation for generic builtin types.

- bpo-45613 [https://bugs.python.org/issue?@action=redirect&bpo=45613]: **`sqlite3`** now sets **`sqlite3.threadsafety`** based on the default threading mode the underlying SQLite library has been compiled with. Patch by Erlend E. Aasland.

- bpo-45574 [https://bugs.python.org/issue?@action=redirect&bpo=45574]: Fix warning about `print_escape` being unused.

- bpo-45581 [https://bugs.python.org/issue?@action=redirect&bpo=45581]: `sqlite3.connect()` now correctly raises `MemoryError` if the underlying SQLite API signals memory error. Patch by Erlend E. Aasland.

- bpo-45557 [https://bugs.python.org/issue?@action=redirect&bpo=45557]: `pprint.pprint()` now handles *underscore_numbers* correctly. Previously it was always setting it to `False`.

- bpo-44019 [https://bugs.python.org/issue?@action=redirect&bpo=44019]: Add `operator.call()` to `operator.__all__`. Patch by Kreusada.

- bpo-42174 [https://bugs.python.org/issue?@action=redirect&bpo=42174]: `shutil.get_terminal_size()` now falls back to sane values if the column or line count are 0.

- bpo-35673 [https://bugs.python.org/issue?@action=redirect&bpo=35673]: Improve the introspectability of the `__loader__` attribute for namespace packages. `importlib.machinery.NamespaceLoader` is now public, and implements the `importlib.abc.InspectLoader` interface. `_NamespaceLoader` is kept for backward compatibility.

- bpo-45515 [https://bugs.python.org/issue?@action=redirect&bpo=45515]: Add references to `zoneinfo` in the `datetime` documentation, mostly replacing outdated references to `dateutil.tz`. Change by Paul Ganssle.

- bpo-45475 [https://bugs.python.org/issue?@action=redirect&bpo=45475]: Reverted optimization of iterating `gzip.GzipFile`, `bz2.BZ2File`, and `lzma.LZMAFile` (see bpo-43787 [https://bugs.python.org/issue?@action=redirect&bpo=43787]) because it caused regression when user iterate them without having reference of them. Patch by Inada Naoki.

- bpo-45489 [https://bugs.python.org/issue?

@action=redirect&bpo=45489]: Update **`ForwardRef`** to support `|` operator. Patch by Donghee Na.

- bpo-42222 [https://bugs.python.org/issue?@action=redirect&bpo=42222]: Removed deprecated support for float arguments in *randrange()*.
- bpo-45428 [https://bugs.python.org/issue?@action=redirect&bpo=45428]: Fix a regression in py_compile when reading filenames from standard input.
- bpo-45467 [https://bugs.python.org/issue?@action=redirect&bpo=45467]: Fix incremental decoder and stream reader in the «raw-unicode-escape» codec. Previously they failed if the escape sequence was split.
- bpo-45461 [https://bugs.python.org/issue?@action=redirect&bpo=45461]: Fix incremental decoder and stream reader in the «unicode-escape» codec. Previously they failed if the escape sequence was split.
- bpo-45239 [https://bugs.python.org/issue?@action=redirect&bpo=45239]: Fixed **`email.utils.parsedate_tz()`** crashing with **`UnboundLocalError`** on certain invalid input instead of returning `None`. Patch by Ben Hoyt.
- bpo-45417 [https://bugs.python.org/issue?@action=redirect&bpo=45417]: Fix quadratic behaviour in the enum module: Creation of enum classes with a lot of entries was quadratic.
- bpo-45249 [https://bugs.python.org/issue?@action=redirect&bpo=45249]: Fix the behaviour of **`traceback.print_exc()`** when displaying the caret when the `end_offset` in the exception is set to 0. Patch by Pablo Galindo
- bpo-45416 [https://bugs.python.org/issue?@action=redirect&bpo=45416]: Fix use of **`asyncio.Condition`** with explicit **`asyncio.Lock`** objects, which was a regression

due to removal of explicit loop arguments. Patch by Joongi Kim.

- bpo-20028 [https://bugs.python.org/issue?@action=redirect&bpo=20028]: Empty escapechar/quotechar is not allowed when initializing `csv.Dialect`. Patch by Vajrasky Kok and Donghee Na.

- bpo-44904 [https://bugs.python.org/issue?@action=redirect&bpo=44904]: Fix bug in the `doctest` module that caused it to fail if a docstring included an example with a `classmethod` `property`. Patch by Alex Waygood.

- bpo-45406 [https://bugs.python.org/issue?@action=redirect&bpo=45406]: Make `inspect.getmodule()` catch `FileNotFoundError` raised by **`inspect.getabsfile()`**, and return `None` to indicate that the module could not be determined.

- bpo-45411 [https://bugs.python.org/issue?@action=redirect&bpo=45411]: Add extensions for files containing subtitles - .srt & .vtt - to the mimetypes.py module.

- bpo-10716 [https://bugs.python.org/issue?@action=redirect&bpo=10716]: Migrated pydoc to HTML5 (without changing the look of it). Side effect is to update xmlrpc's `ServerHTMLDoc` which now uses the CSS too. cgitb now relies less on pydoc (as it can't use the CSS file).

- bpo-27580 [https://bugs.python.org/issue?@action=redirect&bpo=27580]: Add support of null characters in `csv`.

- bpo-45262 [https://bugs.python.org/issue?@action=redirect&bpo=45262]: Prevent use-after-free in asyncio. Make sure the cached running loop holder gets cleared on dealloc to prevent use-after-free in get_running_loop

- bpo-45386 [https://bugs.python.org/issue?@action=redirect&bpo=45386]: Make `xmlrpc.client` more robust to C runtimes where the underlying C `strftime` function results in a `ValueError` when testing for year formatting options.

- bpo-20028 [https://bugs.python.org/issue?@action=redirect&bpo=20028]: Improve error message of **`csv.Dialect`** when initializing. Patch by Vajrasky Kok and Donghee Na.
- bpo-45343 [https://bugs.python.org/issue?@action=redirect&bpo=45343]: Update bundled pip to 21.2.4 and setuptools to 58.1.0
- bpo-45328 [https://bugs.python.org/issue?@action=redirect&bpo=45328]: Fixed **`http.client.HTTPConnection`** to work properly in OSs that don't support the `TCP_NODELAY` socket option.
- bpo-45243 [https://bugs.python.org/issue?@action=redirect&bpo=45243]: Add **`setlimit()`** and **`getlimit()`** to **`sqlite3.Connection`** for setting and getting SQLite limits by connection basis. Patch by Erlend E. Aasland.
- bpo-45320 [https://bugs.python.org/issue?@action=redirect&bpo=45320]: Removed from the **`inspect`** module:

  ○ the `getargspec` function, deprecated since Python 3.0;

  use **`inspect.signature()`** or **`inspect.getfullargspec()`** instead.

  ○ the `formatargspec` function, deprecated since Python 3.5; use the **`inspect.signature()`** function and **`Signature`** object directly.

  ○ the undocumented `Signature.from_builtin` and `Signature.from_function` functions, deprecated since Python 3.5; use the **`Signature.from_callable()`** method instead.

  Patch by Hugo van Kemenade.

- bpo-45192 [https://bugs.python.org/issue?@action=redirect&bpo=45192]: Fix the `tempfile._infer_return_type` function so that the `dir` argument of the **`tempfile`** functions accepts an object implementing the `os.PathLike` protocol.

  Patch by Kyungmin Lee.

- bpo-45160 [https://bugs.python.org/issue?@action=redirect&bpo=45160]: When tracing a tkinter variable used by a ttk OptionMenu, callbacks are no longer made twice.

- bpo-25625 [https://bugs.python.org/issue?@action=redirect&bpo=25625]: Added non parallel-safe **`chdir()`** context manager to change the current working directory and then restore it on exit. Simple wrapper around **`chdir()`**.

- bpo-24139 [https://bugs.python.org/issue?@action=redirect&bpo=24139]: Add support for SQLite extended result codes in **`sqlite3.Error`**. Patch by Erlend E. Aasland.

- bpo-24444 [https://bugs.python.org/issue?@action=redirect&bpo=24444]: Fixed an error raised in **`argparse`** help display when help for an option is set to 1+ blank spaces or when *choices* arg is an empty container.

- bpo-44547 [https://bugs.python.org/issue?@action=redirect&bpo=44547]: Implement `Fraction.__int__`, so that a **`fractions.Fraction`** instance `f` passes an `isinstance(f, typing.SupportsInt)` check.

- bpo-40321 [https://bugs.python.org/issue?@action=redirect&bpo=40321]: Adds support for HTTP 308 redirects to **`urllib`**. See **RFC 7538** [https://datatracker.ietf.org/doc/html/rfc7538.html] for details. Patch by Jochem Schulenklopper.

- bpo-41374 [https://bugs.python.org/issue?

@action=redirect&bpo=41374]: Ensure that `socket.TCP_*` constants are exposed on Cygwin 3.1.6 and greater.

- bpo-35970 [https://bugs.python.org/issue?@action=redirect&bpo=35970]: Add help flag to the base64 module's command line interface. Patch contributed by Robert Kuska.

Documentation

- bpo-45726 [https://bugs.python.org/issue?@action=redirect&bpo=45726]: Improve documentation for **`functools.singledispatch()`** and **`functools.singledispatchmethod`**.
- bpo-45680 [https://bugs.python.org/issue?@action=redirect&bpo=45680]: Amend the docs on `GenericAlias` objects to clarify that non-container classes can also implement `__class_getitem__`. Patch contributed by Alex Waygood.
- bpo-45618 [https://bugs.python.org/issue?@action=redirect&bpo=45618]: Update Sphinx version used to build the documentation to 4.2.0. Patch by Maciej Olko.
- bpo-45655 [https://bugs.python.org/issue?@action=redirect&bpo=45655]: Add a new «relevant PEPs» section to the top of the documentation for the `typing` module. Patch by Alex Waygood.
- bpo-45604 [https://bugs.python.org/issue?@action=redirect&bpo=45604]: Add `level` argument to `multiprocessing.log_to_stderr` function docs.
- bpo-45516 [https://bugs.python.org/issue?@action=redirect&bpo=45516]: Add protocol description to the **`importlib.abc.TraversableResources`** documentation.
- bpo-45464 [https://bugs.python.org/issue?@action=redirect&bpo=45464]: Mention in the documentation of Built-in Exceptions that inheriting from multiple exception types in a single subclass is not recommended due to possible memory layout incompatibility.
- bpo-45449 [https://bugs.python.org/issue?@action=redirect&bpo=45449]: Add note about **PEP 585** [https://peps.python.org/pep-0585/] in **`collections.abc`**.

- bpo-45516 [https://bugs.python.org/issue?@action=redirect&bpo=45516]: Add protocol description to the **`importlib.abc.Traversable`** documentation.
- bpo-20692 [https://bugs.python.org/issue?@action=redirect&bpo=20692]: Add Programming FAQ entry explaining that int literal attribute access requires either a space after or parentheses around the literal.

Tests

- bpo-45678 [https://bugs.python.org/issue?@action=redirect&bpo=45678]: Add tests for scenarios in which **`functools.singledispatchmethod`** is stacked on top of a method that has already been wrapped by two other decorators. Patch by Alex Waygood.
- bpo-45578 [https://bugs.python.org/issue?@action=redirect&bpo=45578]: Add tests for **`dis.distb()`**
- bpo-45678 [https://bugs.python.org/issue?@action=redirect&bpo=45678]: Add tests to ensure that `functools.singledispatchmethod` correctly wraps the attributes of the target function.
- bpo-45668 [https://bugs.python.org/issue?@action=redirect&bpo=45668]: PGO tests now pass when Python is built without test extension modules.
- bpo-45577 [https://bugs.python.org/issue?@action=redirect&bpo=45577]: Add subtests for all `pickle` protocols in `test_zoneinfo`.
- bpo-45566 [https://bugs.python.org/issue?@action=redirect&bpo=45566]: Fix `test_frozen_pickle` in `test_dataclasses` to check all `pickle` versions.
- bpo-43592 [https://bugs.python.org/issue?@action=redirect&bpo=43592]: **`test.libregrtest`** now raises the soft resource limit for the maximum number of file descriptors when the default is too low for our test suite as was often the case on macOS.
- bpo-39679 [https://bugs.python.org/issue?@action=redirect&bpo=39679]: Add more test cases for `@functools.singledispatchmethod` when combined with `@classmethod` or `@staticmethod`.
- bpo-45410 [https://bugs.python.org/issue?

@action=redirect&bpo=45410]: When libregrtest spawns a worker process, stderr is now written into stdout to keep messages order. Use a single pipe for stdout and stderr, rather than two pipes. Previously, messages were out of order which made analysis of buildbot logs harder Patch by Victor Stinner.

- bpo-45402 [https://bugs.python.org/issue?@action=redirect&bpo=45402]: Fix test_tools.test_sundry() when Python is built out of tree: fix how the freeze_modules.py tool locates the _freeze_module program. Patch by Victor Stinner.
- bpo-45403 [https://bugs.python.org/issue?@action=redirect&bpo=45403]: Fix test_sys.test_stdlib_dir() when Python is built outside the source tree: compare normalized paths. Patch by Victor Stinner.
- bpo-45400 [https://bugs.python.org/issue?@action=redirect&bpo=45400]: Fix test_name_error_suggestions_do_not_trigger_for_too_many_locals() of test_exceptions if a directory name contains «a1» (like «Python-3.11.0a1»): use a stricter regular expression. Patch by Victor Stinner.
- bpo-10572 [https://bugs.python.org/issue?@action=redirect&bpo=10572]: Rename **`sqlite3`** tests from `test_sqlite` to `test_sqlite3`, and relocate them to `Lib/test/test_sqlite3`. Patch by Erlend E. Aasland.

Build

- bpo-43158 [https://bugs.python.org/issue?@action=redirect&bpo=43158]: `setup.py` now uses values from configure script to build the `_uuid` extension module. Configure now detects util-linux's `libuuid`, too.
- bpo-45666 [https://bugs.python.org/issue?@action=redirect&bpo=45666]: Fix warning of `swprintf` and `%s` usage in `_testembed.c`
- bpo-45548 [https://bugs.python.org/issue?@action=redirect&bpo=45548]: `Modules/Setup` and `Modules/makesetup` have been improved. The `Setup` file now contains working rules for all extensions. Outdated comments have been removed. Rules defined by `makesetup` track dependencies correctly.
- bpo-45548 [https://bugs.python.org/issue?

@action=redirect&bpo=45548]: The **`math`** and **`cmath`** implementation now require a C99 compatible `libm` and no longer ship with workarounds for missing acosh, asinh, atanh, expm1, and log1p functions.

- bpo-45595 [https://bugs.python.org/issue?@action=redirect&bpo=45595]: `setup.py` and `makesetup` now track build dependencies on all Python header files and module specific header files.
- bpo-45571 [https://bugs.python.org/issue?@action=redirect&bpo=45571]: `Modules/Setup` now use `PY_CFLAGS_NODIST` instead of `PY_CFLAGS` to compile shared modules.
- bpo-45570 [https://bugs.python.org/issue?@action=redirect&bpo=45570]: **`pyexpat`** and **`_elementtree`** no longer define obsolete macros `HAVE_EXPAT_CONFIG_H` and `USE_PYEXPAT_CAPI`. `XML_POOR_ENTROPY` is now defined in `expat_config.h`.
- bpo-43974 [https://bugs.python.org/issue?@action=redirect&bpo=43974]: `setup.py` no longer defines `Py_BUILD_CORE_MODULE`. Instead every module, that uses the internal API, defines the macro.
- bpo-45548 [https://bugs.python.org/issue?@action=redirect&bpo=45548]: Fill in missing entries in Modules/Setup.
- bpo-45532 [https://bugs.python.org/issue?@action=redirect&bpo=45532]: Update **`sys.version`** to use `main` as fallback information. Patch by Jeong YunWon.
- bpo-45536 [https://bugs.python.org/issue?@action=redirect&bpo=45536]: The `configure` script now checks whether OpenSSL headers and libraries provide required APIs. Most common APIs are verified. The check detects outdated or missing OpenSSL. Failures do not stop configure.
- bpo-45221 [https://bugs.python.org/issue?@action=redirect&bpo=45221]: Fixed regression in handling of `LDFLAGS` and `CPPFLAGS` options where **`argparse.parse_known_args()`** could interpret an option as one of the built-in command line argument, for example `-h` for help.
- bpo-45440 [https://bugs.python.org/issue?

@action=redirect&bpo=45440]: Building Python now requires a C99 `<math.h>` header file providing the following functions: `copysign()`, `hypot()`, `isfinite()`, `isinf()`, `isnan()`, `round()`. Patch by Victor Stinner.
- bpo-45405 [https://bugs.python.org/issue?@action=redirect&bpo=45405]: Prevent `internal configure error` when running `configure` with recent versions of non-Apple clang. Patch by David Bohman.
- bpo-45433 [https://bugs.python.org/issue?@action=redirect&bpo=45433]: Avoid linking libpython with libcrypt.

Windows

- bpo-43652 [https://bugs.python.org/issue?@action=redirect&bpo=43652]: Update Tcl/Tk to 8.6.11, actually this time. The previous update incorrectly included 8.6.10.
- bpo-45337 [https://bugs.python.org/issue?@action=redirect&bpo=45337]: venv now warns when the created environment may need to be accessed at a different path, due to redirections, links or junctions. It also now correctly installs or upgrades components when the alternate path is required.
- bpo-43851 [https://bugs.python.org/issue?@action=redirect&bpo=43851]: Build SQLite `SQLITE_OMIT_AUTOINIT` on Windows. Patch by Erlend E. Aasland.

macOS

- bpo-44828 [https://bugs.python.org/issue?@action=redirect&bpo=44828]: Avoid tkinter file dialog failure on macOS 12 Monterey when using the Tk 8.6.11 provided by python.org macOS installers. Patch by Marc Culler of the Tk project.

IDLE

- bpo-45495 [https://bugs.python.org/issue?@action=redirect&bpo=45495]: Add context keywords “case” and "match" to completions list.

C API

- bpo-29103 [https://bugs.python.org/issue?@action=redirect&bpo=29103]: **`PyType_FromSpec*`** now copies the class name from the spec to a buffer owned by the class, so the original can be safely deallocated. Patch by Petr Viktorin.

- bpo-45522 [https://bugs.python.org/issue?@action=redirect&bpo=45522]: The internal freelists for frame, float, list, dict, async generators, and context objects can now be disabled.

- bpo-35134 [https://bugs.python.org/issue?@action=redirect&bpo=35134]: Exclude **`PyWeakref_GET_OBJECT()`** from the limited C API. It never worked since the **`PyWeakReference`** structure is opaque in the limited C API.

- bpo-35081 [https://bugs.python.org/issue?@action=redirect&bpo=35081]: Move the `interpreteridobject.h` header file from `Include/` to `Include/internal/`. It only provides private functions. Patch by Victor Stinner.

- bpo-35134 [https://bugs.python.org/issue?@action=redirect&bpo=35134]: The non-limited API files `cellobject.h`, `classobject.h`, `context.h`, `funcobject.h`, `genobject.h` and `longintrepr.h` have been moved to the `Include/cpython` directory. Moreover, the `eval.h` header file was removed. These files must not be included directly, as they are already included in `Python.h`: Include Files. If they have been included directly, consider including `Python.h` instead. Patch by Victor Stinner.

- bpo-45474 [https://bugs.python.org/issue?@action=redirect&bpo=45474]: The following items are no longer available when `Py_LIMITED_API` is defined:

    - **`PyMarshal_WriteLongToFile()`**
    - **`PyMarshal_WriteObjectToFile()`**
    - **`PyMarshal_ReadObjectFromString()`**
    - **`PyMarshal_WriteObjectToString()`**
    - the `Py_MARSHAL_VERSION` macro

  These are not part of the limited API.

  Patch by Victor Stinner.

- bpo-45434 [https://bugs.python.org/issue?@action=redirect&bpo=45434]: Remove the `pystrhex.h` header file. It only contains private functions. C extensions should only include the main `<Python.h>` header file. Patch by Victor Stinner.

- bpo-45440 [https://bugs.python.org/issue?@action=redirect&bpo=45440]: Remove the `Py_FORCE_DOUBLE()` macro. It was used by the `Py_IS_INFINITY()` macro. Patch by Victor Stinner.

- bpo-45434 [https://bugs.python.org/issue?@action=redirect&bpo=45434]: `<Python.h>` no longer includes the header files `<stdlib.h>`, `<stdio.h>`, `<errno.h>` and `<string.h>` when the `Py_LIMITED_API` macro is set to `0x030b0000` (Python 3.11) or higher. C extensions should explicitly include the header files after `#include <Python.h>`. Patch by Victor Stinner.

- bpo-41123 [https://bugs.python.org/issue?@action=redirect&bpo=41123]: Remove `Py_UNICODE_COPY()` and `Py_UNICODE_FILL()` macros, deprecated since Python 3.3. Use `PyUnicode_CopyCharacters()` or `memcpy()` (`wchar_t*` string), and `PyUnicode_Fill()` functions instead. Patch by Victor Stinner.

- bpo-45412 [https://bugs.python.org/issue?@action=redirect&bpo=45412]: Remove the following math macros using the `errno` variable:

    - `Py_ADJUST_ERANGE1()`

  - `Py_ADJUST_ERANGE2()`
  - `Py_OVERFLOWED()`
  - `Py_SET_ERANGE_IF_OVERFLOW()`
  - `Py_SET_ERRNO_ON_MATH_ERROR()`

  Patch by Victor Stinner.

- bpo-45395 [https://bugs.python.org/issue?@action=redirect&bpo=45395]: Custom frozen modules (the array set to `PyImport_FrozenModules`) are now treated as additions, rather than replacing all the default frozen modules. Frozen stdlib modules can still be disabled by setting the «code» field of the custom array entry to NULL.

- bpo-43760 [https://bugs.python.org/issue?@action=redirect&bpo=43760]: Add new **`PyThreadState_EnterTracing()`**, and **`PyThreadState_LeaveTracing()`** functions to the limited C API to suspend and resume tracing and profiling. Patch by Victor Stinner.

- bpo-44220 [https://bugs.python.org/issue?@action=redirect&bpo=44220]: **`PyStructSequence_UnnamedField`** is added to the Stable ABI.

Python 3.11.0 alpha 1

Release date: 2021-10-05

Security

- bpo-42278 [https://bugs.python.org/issue?@action=redirect&bpo=42278]: Replaced usage of **`tempfile.mktemp()`** with **`TemporaryDirectory`** to avoid a potential race condition.
- bpo-44600 [https://bugs.python.org/issue?@action=redirect&bpo=44600]: Fix incorrect line numbers while tracing some failed patterns in match statements. Patch by Charles Burkland.

- bpo-41180 [https://bugs.python.org/issue?@action=redirect&bpo=41180]: Add auditing events to the **`marshal`** module, and stop raising `code.__init__` events for every unmarshalled code object. Directly instantiated code objects will continue to raise an event, and audit event handlers should inspect or collect the raw marshal data. This reduces a significant performance overhead when loading from `.pyc` files.
- bpo-44394 [https://bugs.python.org/issue?@action=redirect&bpo=44394]: Update the vendored copy of libexpat to 2.4.1 (from 2.2.8) to get the fix for the **CVE 2013-0340** [https://www.cve.org/CVERecord?id=CVE-2013-0340] «Billion Laughs» vulnerability. This copy is most used on Windows and macOS.
- bpo-43124 [https://bugs.python.org/issue?@action=redirect&bpo=43124]: Made the internal `putcmd` function in **`smtplib`** sanitize input for presence of `\r` and `\n` characters to avoid (unlikely) command injection.
- bpo-44022 [https://bugs.python.org/issue?@action=redirect&bpo=44022]: **`http.client`** now avoids infinitely reading potential HTTP headers after a `100 Continue` status response from the server.

Core and Builtins

- bpo-43760 [https://bugs.python.org/issue?@action=redirect&bpo=43760]: The number of hardware branches per instruction dispatch is reduced from two to one by adding a special instruction for tracing. Patch by Mark Shannon.
- bpo-45061 [https://bugs.python.org/issue?@action=redirect&bpo=45061]: Add a deallocator to the bool type to detect refcount bugs in C extensions which call Py_DECREF(Py_True) or Py_DECREF(Py_False) by mistake. Detect also refcount bugs when the empty tuple singleton or the Unicode empty string singleton is destroyed by mistake. Patch by Victor Stinner.
- bpo-24076 [https://bugs.python.org/issue?@action=redirect&bpo=24076]: sum() was further optimised for summing up single digit integers.
- bpo-45190 [https://bugs.python.org/issue?

@action=redirect&bpo=45190]: Update Unicode databases to Unicode 14.0.0.

- bpo-45167 [https://bugs.python.org/issue?@action=redirect&bpo=45167]: Fix deepcopying of `types.GenericAlias` objects.
- bpo-45155 [https://bugs.python.org/issue?@action=redirect&bpo=45155]: `int.to_bytes()` and `int.from_bytes()` now take a default value of `"big"` for the `byteorder` argument. **`int.to_bytes()`** also takes a default value of `1` for the `length` argument.

Library

- bpo-44219 [https://bugs.python.org/issue?@action=redirect&bpo=44219]: Release the GIL while performing `isatty` system calls on arbitrary file descriptors. In particular, this affects `os.isatty()`, `os.device_encoding()` and `io.TextIOWrapper`. By extension, `io.open()` in text mode is also affected. This change solves a deadlock in **`os.isatty()`**. Patch by Vincent Michel in bpo-44219 [https://bugs.python.org/issue?@action=redirect&bpo=44219].

Core and Builtins

- bpo-44959 [https://bugs.python.org/issue?@action=redirect&bpo=44959]: Added fallback to extension modules with ".sl" suffix on HP-UX

Library

- bpo-45121 [https://bugs.python.org/issue?@action=redirect&bpo=45121]: Fix issue where `Protocol.__init__` raises `RecursionError` when it's called directly or via `super()`. Patch provided by Yurii Karabas.

Core and Builtins

- bpo-44348 [https://bugs.python.org/issue?@action=redirect&bpo=44348]: The deallocator function of the `BaseException` type now uses the trashcan mechanism to prevent stack overflow. For example, when a `RecursionError` instance is raised, it can be linked to another RecursionError through the `__context__` attribute or the `__traceback__` attribute, and then a chain of exceptions is created. When the chain is destroyed, nested deallocator function calls can crash with a stack overflow if the chain is too long compared to the available stack memory. Patch by Victor Stinner.

- bpo-45123 [https://bugs.python.org/issue?@action=redirect&bpo=45123]: Fix PyAiter_Check to only check for the __anext__ presence (not for __aiter__). Rename PyAiter_Check to PyAIter_Check, PyObject_GetAiter -> PyObject_GetAIter.

- bpo-1514420 [https://bugs.python.org/issue?@action=redirect&bpo=1514420]: Interpreter no longer attempts to open files with names in angle brackets (like «<string>» or «<stdin>») when formatting an exception.

- bpo-41031 [https://bugs.python.org/issue?@action=redirect&bpo=41031]: Match C and Python code formatting of unprintable exceptions and exceptions in the `__main__` module.

- bpo-37330 [https://bugs.python.org/issue?@action=redirect&bpo=37330]: `open()`, `io.open()`, `codecs.open()` and `fileinput.FileInput` no longer accept `'U'` («universal newline») in the file mode. This flag was deprecated since Python 3.3. Patch by Victor Stinner.

- bpo-45083 [https://bugs.python.org/issue?@action=redirect&bpo=45083]: When the interpreter renders an exception, its name now has a complete qualname. Previously only the class name was concatenated to the module name, which sometimes resulted in an incorrect full name being displayed.

(This issue impacted only the C code exception rendering, the **`traceback`** module was using qualname already).

- bpo-34561 [https://bugs.python.org/issue?@action=redirect&bpo=34561]: List sorting now uses the merge-ordering strategy from Munro and Wild's `powersort()`. Unlike the former strategy, this is provably near-optimal in the entropy of the distribution of run lengths. Most uses of `list.sort()` probably won't see a significant time difference, but may see significant improvements in cases where the former strategy was exceptionally poor. However, as these are all fast linear-time approximations to a problem that's inherently at best quadratic-time to solve truly optimally, it's also possible to contrive cases where the former strategy did better.

- bpo-45056 [https://bugs.python.org/issue?@action=redirect&bpo=45056]: Compiler now removes trailing unused constants from co_consts.

- bpo-45020 [https://bugs.python.org/issue?@action=redirect&bpo=45020]: Add a new command line option, «-X frozen_modules=[on|off]» to opt out of (or into) using optional frozen modules. This defaults to «on» (or «off» if it's running out of the source tree).

Library

- bpo-45012 [https://bugs.python.org/issue?@action=redirect&bpo=45012]: In **`posix`**, release GIL during `stat()`, `lstat()`, and `fstatat()` syscalls made by **`os.DirEntry.stat()`**. Patch by Stanisław Skonieczny.

Core and Builtins

- bpo-45018 [https://bugs.python.org/issue?@action=redirect&bpo=45018]: Fixed pickling of range iterators that iterated for over `2**32` times.
- bpo-45000 [https://bugs.python.org/issue?@action=redirect&bpo=45000]: A **`SyntaxError`** is now raised

when trying to delete **`__debug__`**. Patch by Donghee Na.

- bpo-44963 [https://bugs.python.org/issue?@action=redirect&bpo=44963]: Implement `send()` and `throw()` methods for `anext_awaitable` objects. Patch by Pablo Galindo.

Library

- bpo-44962 [https://bugs.python.org/issue?@action=redirect&bpo=44962]: Fix a race in WeakKeyDictionary, WeakValueDictionary and WeakSet when two threads attempt to commit the last pending removal. This fixes asyncio.create_task and fixes a data loss in asyncio.run where shutdown_asyncgens is not run

Core and Builtins

- bpo-24234 [https://bugs.python.org/issue?@action=redirect&bpo=24234]: Implement the **`__bytes__()`** special method on the **`bytes`** type, so a bytes object `b` passes an `isinstance(b, typing.SupportsBytes)` check.
- bpo-24234 [https://bugs.python.org/issue?@action=redirect&bpo=24234]: Implement the **`__complex__()`** special method on the **`complex`** type, so a complex number `z` passes an `isinstance(z, typing.SupportsComplex)` check.
- bpo-44954 [https://bugs.python.org/issue?@action=redirect&bpo=44954]: Fixed a corner case bug where the result of `float.fromhex('0x.8p-1074')` was rounded the wrong way.
- bpo-44947 [https://bugs.python.org/issue?@action=redirect&bpo=44947]: Refine the syntax error for trailing commas in import statements. Patch by Pablo Galindo.
- bpo-44945 [https://bugs.python.org/issue?@action=redirect&bpo=44945]: Specialize the BINARY_ADD instruction using the PEP 659 machinery. Adds five new instructions:
  - BINARY_ADD_ADAPTIVE
  - BINARY_ADD_FLOAT

    - BINARY_ADD_INT
    - BINARY_ADD_UNICODE
    - BINARY_ADD_UNICODE_INPLACE_FAST

Library

- bpo-44929 [https://bugs.python.org/issue?@action=redirect&bpo=44929]: Fix some edge cases of `enum.Flag` string representation in the REPL. Patch by Pablo Galindo.

Core and Builtins

- bpo-44914 [https://bugs.python.org/issue?@action=redirect&bpo=44914]: Class version tags are no longer recycled.

  This means that a version tag serves as a unique identifier for the state of a class. We rely on this for effective specialization of the LOAD_ATTR and other instructions.

- bpo-44698 [https://bugs.python.org/issue?@action=redirect&bpo=44698]: Restore behaviour of complex exponentiation with integer-valued exponent of type **`float`** or **`complex`**.

- bpo-44895 [https://bugs.python.org/issue?@action=redirect&bpo=44895]: A debug variable **`PYTHONDUMPREFSFILE`** is added for creating a dump file which is generated by **`--with-trace-refs`**. Patch by Donghee Na.

- bpo-44900 [https://bugs.python.org/issue?@action=redirect&bpo=44900]: Add five superinstructions for PEP 659 quickening:

    - LOAD_FAST LOAD_FAST
    - STORE_FAST LOAD_FAST
    - LOAD_FAST LOAD_CONST
    - LOAD_CONST LOAD_FAST
    - STORE_FAST STORE_FAST

- bpo-44889 [https://bugs.python.org/issue?@action=redirect&bpo=44889]: Initial implementation of adaptive specialization of `LOAD_METHOD`. The following specialized forms were added:

  - `LOAD_METHOD_CACHED`
  - `LOAD_METHOD_MODULE`
  - `LOAD_METHOD_CLASS`

- bpo-44890 [https://bugs.python.org/issue?@action=redirect&bpo=44890]: Specialization stats are always collected in debug builds.

- bpo-44885 [https://bugs.python.org/issue?@action=redirect&bpo=44885]: Correct the ast locations of f-strings with format specs and repeated expressions. Patch by Pablo Galindo

- bpo-44878 [https://bugs.python.org/issue?@action=redirect&bpo=44878]: Remove the loop from the bytecode interpreter. All instructions end with a DISPATCH macro, so the loop is now redundant.

- bpo-44878 [https://bugs.python.org/issue?@action=redirect&bpo=44878]: Remove switch statement for interpreter loop when using computed gotos. This makes sure that we only have one dispatch table in the interpreter.

- bpo-44874 [https://bugs.python.org/issue?@action=redirect&bpo=44874]: Deprecate the old trashcan macros (`Py_TRASHCAN_SAFE_BEGIN`/`Py_TRASHCAN_SAFE_END`). They should be replaced by the new macros `Py_TRASHCAN_BEGIN` and `Py_TRASHCAN_END`.

- bpo-44872 [https://bugs.python.org/issue?@action=redirect&bpo=44872]: Use new trashcan macros (Py_TRASHCAN_BEGIN/END) in frameobject.c instead of the old ones (Py_TRASHCAN_SAFE_BEGIN/END).

- bpo-33930 [https://bugs.python.org/issue?

@action=redirect&bpo=33930]: Fix segmentation fault with deep recursion when cleaning method objects. Patch by Augusto Goulart and Pablo Galindo.

- bpo-25782 [https://bugs.python.org/issue?@action=redirect&bpo=25782]: Fix bug where `PyErr_SetObject` hangs when the current exception has a cycle in its context chain.

- bpo-44856 [https://bugs.python.org/issue?@action=redirect&bpo=44856]: Fix reference leaks in the error paths of `update_bases()` and `__build_class__`. Patch by Pablo Galindo.

- bpo-44826 [https://bugs.python.org/issue?@action=redirect&bpo=44826]: Initial implementation of adaptive specialization of STORE_ATTR

  Three specialized forms of STORE_ATTR are added:

  ○ STORE_ATTR_SLOT
  ○ STORE_ATTR_SPLIT_KEYS
  ○ STORE_ATTR_WITH_HINT

- bpo-44838 [https://bugs.python.org/issue?@action=redirect&bpo=44838]: Fixed a bug that was causing the parser to raise an incorrect custom **`SyntaxError`** for invalid “if” expressions. Patch by Pablo Galindo.

- bpo-44821 [https://bugs.python.org/issue?@action=redirect&bpo=44821]: Create instance dictionaries (_dict_) eagerly, to improve regularity of object layout and assist specialization.

- bpo-44792 [https://bugs.python.org/issue?@action=redirect&bpo=44792]: Improve syntax errors for if expressions. Patch by Miguel Brito

- bpo-34013 [https://bugs.python.org/issue?@action=redirect&bpo=34013]: Generalize the invalid legacy statement custom error message (like the one generated when «print» is called without parentheses) to include more generic expressions. Patch by Pablo Galindo

- bpo-44732 [https://bugs.python.org/issue?@action=redirect&bpo=44732]: Rename `types.Union` to `types.UnionType`.

- bpo-44725 [https://bugs.python.org/issue?@action=redirect&bpo=44725]: Expose specialization stats in python via **`_opcode.get_specialization_stats()`**.

- bpo-44717 [https://bugs.python.org/issue?@action=redirect&bpo=44717]: Improve AttributeError on circular imports of submodules.

- bpo-44698 [https://bugs.python.org/issue?@action=redirect&bpo=44698]: Fix undefined behaviour in complex object exponentiation.

- bpo-44653 [https://bugs.python.org/issue?@action=redirect&bpo=44653]: Support **`typing`** types in parameter substitution in the union type.

- bpo-44676 [https://bugs.python.org/issue?@action=redirect&bpo=44676]: Add ability to serialise `types.Union` objects. Patch provided by Yurii Karabas.

- bpo-44633 [https://bugs.python.org/issue?@action=redirect&bpo=44633]: Parameter substitution of the union type with wrong types now raises `TypeError` instead of returning **`NotImplemented`**.

- bpo-44661 [https://bugs.python.org/issue?@action=redirect&bpo=44661]: Update `property_descr_set` to use vectorcall if possible. Patch by Donghee Na.

- bpo-44662 [https://bugs.python.org/issue?@action=redirect&bpo=44662]: Add `__module__` to `types.Union`. This also fixes `types.Union` issues with `typing.Annotated`. Patch provided by Yurii Karabas.

- bpo-44655 [https://bugs.python.org/issue?

@action=redirect&bpo=44655]: Include the name of the type in unset _slots_ attribute errors. Patch by Pablo Galindo

- bpo-44655 [https://bugs.python.org/issue?@action=redirect&bpo=44655]: Don't include a missing attribute with the same name as the failing one when offering suggestions for missing attributes. Patch by Pablo Galindo
- bpo-44646 [https://bugs.python.org/issue?@action=redirect&bpo=44646]: Fix the hash of the union type: it no longer depends on the order of arguments.
- bpo-44636 [https://bugs.python.org/issue?@action=redirect&bpo=44636]: Collapse union of equal types. E.g. the result of `int | int` is now `int`. Fix comparison of the union type with non-hashable objects. E.g. `int | str == {}` no longer raises a TypeError.

Library

- bpo-44611 [https://bugs.python.org/issue?@action=redirect&bpo=44611]: On Windows, **`os.urandom()`**: uses BCryptGenRandom API instead of CryptGenRandom API which is deprecated from Microsoft Windows API. Patch by Donghee Na.

Core and Builtins

- bpo-44635 [https://bugs.python.org/issue?@action=redirect&bpo=44635]: Convert `None` to `type(None)` in the union type constructor.
- bpo-26280 [https://bugs.python.org/issue?@action=redirect&bpo=26280]: Implement adaptive specialization for BINARY_SUBSCR

  Three specialized forms of BINARY_SUBSCR are added:

  - BINARY_SUBSCR_LIST_INT
  - BINARY_SUBSCR_TUPLE_INT
  - BINARY_SUBSCR_DICT

- bpo-44589 [https://bugs.python.org/issue?@action=redirect&bpo=44589]: Mapping patterns in `match` statements with two or more equal literal keys will now raise a **`SyntaxError`** at compile-time.

- bpo-44606 [https://bugs.python.org/issue?@action=redirect&bpo=44606]: Fix `__instancecheck__` and `__subclasscheck__` for the union type.

- bpo-42073 [https://bugs.python.org/issue?@action=redirect&bpo=42073]: The `@classmethod` decorator can now wrap other classmethod-like descriptors.

- bpo-41972 [https://bugs.python.org/issue?@action=redirect&bpo=41972]: Tuned the string-searching algorithm of fastsearch.h to have a shorter inner loop for most cases.

- bpo-44590 [https://bugs.python.org/issue?@action=redirect&bpo=44590]: All necessary data for executing a Python function (local variables, stack, etc) is now kept in a per-thread stack. Frame objects are lazily allocated on demand. This increases performance by about 7% on the standard benchmark suite. Introspection and debugging are unaffected as frame objects are always available when needed. Patch by Mark Shannon.

- bpo-44584 [https://bugs.python.org/issue?@action=redirect&bpo=44584]: The threading debug (**`PYTHONTHREADDEBUG`** environment variable) is deprecated in Python 3.10 and will be removed in Python 3.12. This feature requires a debug build of Python. Patch by Victor Stinner.

- bpo-43895 [https://bugs.python.org/issue?@action=redirect&bpo=43895]: An obsolete internal cache of shared object file handles added in 1995 that attempted, but did not guarantee, that a .so would not be dlopen'ed twice to work around flaws in mid-1990s posix-ish operating systems has been removed from dynload_shlib.c.

Library

- bpo-44490 [https://bugs.python.org/issue?@action=redirect&bpo=44490]: **`typing`** now searches for type parameters in `types.Union` objects. `get_type_hints` will also properly resolve annotations with nested `types.Union` objects. Patch provided by Yurii Karabas.

Core and Builtins

- bpo-43950 [https://bugs.python.org/issue?@action=redirect&bpo=43950]: Code objects can now provide the column information for instructions when available. This is levaraged during traceback printing to show the expressions responsible for errors.

  Contributed by Pablo Galindo, Batuhan Taskaya and Ammar Askar as part of **PEP 657** [https://peps.python.org/pep-0657/].

- bpo-44562 [https://bugs.python.org/issue?@action=redirect&bpo=44562]: Remove uses of **`PyObject_GC_Del()`** in error path when initializing **`types.GenericAlias`**.

Library

- bpo-41486 [https://bugs.python.org/issue?@action=redirect&bpo=41486]: Fix a memory consumption and copying performance regression in earlier 3.10 beta releases if someone used an output buffer larger than 4GiB with zlib.decompress on input data that expands that large.

Core and Builtins

- bpo-43908 [https://bugs.python.org/issue?@action=redirect&bpo=43908]: Heap types with the **`Py_TPFLAGS_IMMUTABLETYPE`** flag can now inherit the **PEP 590** [https://peps.python.org/pep-0590/] vectorcall protocol. Previously, this was only possible for static types. Patch by Erlend E. Aasland.

- bpo-44553 [https://bugs.python.org/issue?@action=redirect&bpo=44553]: Implement GC methods for `types.Union` to break reference cycles and prevent memory leaks.

- bpo-44490 [https://bugs.python.org/issue?@action=redirect&bpo=44490]: Add `__parameters__` attribute and `__getitem__` operator to `types.Union`. Patch provided by Yurii Karabas.

- bpo-44523 [https://bugs.python.org/issue?@action=redirect&bpo=44523]: Remove the pass-through for **`hash()`** of **`weakref.proxy`** objects to prevent unintended consequences when the original referred object dies while the proxy is part of a hashable object. Patch by Pablo Galindo.

- bpo-44483 [https://bugs.python.org/issue?@action=redirect&bpo=44483]: Fix a crash in `types.Union` objects when creating a union of an object with bad `__module__` field.

- bpo-44486 [https://bugs.python.org/issue?@action=redirect&bpo=44486]: Modules will always have a dictionary, even when created by `types.ModuleType.__new__()`

- bpo-44472 [https://bugs.python.org/issue?@action=redirect&bpo=44472]: Fix ltrace functionality when exceptions are raised. Patch by Pablo Galindo

- bpo-12022 [https://bugs.python.org/issue?@action=redirect&bpo=12022]: A **`TypeError`** is now raised instead of an **`AttributeError`** in **`with`** and **`async with`** statements for objects which do not support the context manager or asynchronous context manager protocols correspondingly.

- bpo-44297 [https://bugs.python.org/issue?@action=redirect&bpo=44297]: Make sure that the line number is set when entering a comprehension scope. This ensures that backtraces including generator expressions show the correct line number.

- bpo-44456 [https://bugs.python.org/issue?@action=redirect&bpo=44456]: Improve the syntax error when mixing positional and keyword patterns. Patch by Pablo Galindo.

- bpo-44409 [https://bugs.python.org/issue?@action=redirect&bpo=44409]: Fix error location information for tokenizer errors raised on initialization of the tokenizer. Patch by Pablo Galindo.

- bpo-44396 [https://bugs.python.org/issue?@action=redirect&bpo=44396]: Fix a possible crash in the tokenizer when raising syntax errors for unclosed strings. Patch by Pablo Galindo.

- bpo-44376 [https://bugs.python.org/issue?@action=redirect&bpo=44376]: Exact integer exponentiation (like `i**2` or `pow(i, 2)`) with a small exponent is much faster, due to reducing overhead in such cases.

- bpo-44313 [https://bugs.python.org/issue?@action=redirect&bpo=44313]: Directly imported objects and modules (through import and from import statements) don't generate `LOAD_METHOD/CALL_METHOD` for directly accessed objects on their namespace. They now use the regular `LOAD_ATTR/CALL_FUNCTION`.

- bpo-44338 [https://bugs.python.org/issue?@action=redirect&bpo=44338]: Implement adaptive specialization for LOAD_GLOBAL

  Two specialized forms of LOAD_GLOBAL are added:

  - LOAD_GLOBAL_MODULE
  - LOAD_GLOBAL_BUILTIN

- bpo-44368 [https://bugs.python.org/issue?@action=redirect&bpo=44368]: Improve syntax errors for invalid «as» targets. Patch by Pablo Galindo

- bpo-44349 [https://bugs.python.org/issue?@action=redirect&bpo=44349]: Fix an edge case when displaying text from files with encoding in syntax errors. Patch by Pablo Galindo.

- bpo-44337 [https://bugs.python.org/issue?@action=redirect&bpo=44337]: Initial implementation of adaptive specialization of LOAD_ATTR

  Four specialized forms of LOAD_ATTR are added:

  ○ LOAD_ATTR_SLOT
  ○ LOAD_ATTR_SPLIT_KEYS
  ○ LOAD_ATTR_WITH_HINT
  ○ LOAD_ATTR_MODULE

- bpo-44335 [https://bugs.python.org/issue?@action=redirect&bpo=44335]: Fix a regression when identifying incorrect characters in syntax errors. Patch by Pablo Galindo

- bpo-43693 [https://bugs.python.org/issue?@action=redirect&bpo=43693]: Computation of the offsets of cell variables is done in the compiler instead of at runtime. This reduces the overhead of handling cell and free variables, especially in the case where a variable is both an argument and cell variable.

- bpo-44317 [https://bugs.python.org/issue?@action=redirect&bpo=44317]: Improve tokenizer error with improved locations. Patch by Pablo Galindo.

Library

- bpo-44304 [https://bugs.python.org/issue?@action=redirect&bpo=44304]: Fix a crash in the `sqlite3` module that happened when the garbage collector clears `sqlite.Statement` objects. Patch by Pablo Galindo

Core and Builtins

- bpo-44305 [https://bugs.python.org/issue?

@action=redirect&bpo=44305]: Improve error message for `try` blocks without `except` or `finally` blocks. Patch by Pablo Galindo.

- bpo-43413 [https://bugs.python.org/issue?@action=redirect&bpo=43413]: Constructors of subclasses of some builtin classes (e.g. **`tuple`**, **`list`**, **`frozenset`**) no longer accept arbitrary keyword arguments. [reverted in 3.11a4] Subclass of **`set`** can now define a `__new__()` method with additional keyword parameters without overriding also `__init__()`.

- bpo-43667 [https://bugs.python.org/issue?@action=redirect&bpo=43667]: Improve Unicode support in non-UTF locales on Oracle Solaris. This issue does not affect other Solaris systems.

- bpo-43693 [https://bugs.python.org/issue?@action=redirect&bpo=43693]: A new opcode MAKE_CELL has been added that effectively moves some of the work done on function entry into the compiler and into the eval loop. In addition to creating the required cell objects, the new opcode converts relevant arguments (and other locals) to cell variables on function entry.

- bpo-44232 [https://bugs.python.org/issue?@action=redirect&bpo=44232]: Fix a regression in **`type()`** when a metaclass raises an exception. The C function **`type_new()`** must properly report the exception when a metaclass constructor raises an exception and the winner class is not the metaclass. Patch by Victor Stinner.

- bpo-44201 [https://bugs.python.org/issue?@action=redirect&bpo=44201]: Avoid side effects of checking for specialized syntax errors in the REPL that was causing it to ask for extra tokens after a syntax error had been detected. Patch by Pablo Galindo

- bpo-43693 [https://bugs.python.org/issue?@action=redirect&bpo=43693]: `PyCodeObject` gained `co_fastlocalnames` and `co_fastlocalkinds` as the authoritative source of fast locals info. Marshaled code objects have changed accordingly.

- bpo-44184 [https://bugs.python.org/issue?@action=redirect&bpo=44184]: Fix a crash at Python exit when a deallocator function removes the last strong reference to a heap type. Patch by Victor Stinner.
- bpo-44187 [https://bugs.python.org/issue?@action=redirect&bpo=44187]: Implement quickening in the interpreter. This offers no advantages as yet, but is an enabler of future optimizations. See PEP 659 for full explanation.
- bpo-44180 [https://bugs.python.org/issue?@action=redirect&bpo=44180]: The parser doesn't report generic syntax errors that happen in a position further away that the one it reached in the first pass. Patch by Pablo Galindo
- bpo-44168 [https://bugs.python.org/issue?@action=redirect&bpo=44168]: Fix error message in the parser involving keyword arguments with invalid expressions. Patch by Pablo Galindo
- bpo-44156 [https://bugs.python.org/issue?@action=redirect&bpo=44156]: String caches in `compile.c` are now subinterpreter compatible.
- bpo-44143 [https://bugs.python.org/issue?@action=redirect&bpo=44143]: Fixed a crash in the parser that manifest when raising tokenizer errors when an existing exception was present. Patch by Pablo Galindo.
- bpo-44032 [https://bugs.python.org/issue?@action=redirect&bpo=44032]: Move “fast” locals and other variables from the frame object to a per-thread datastack.
- bpo-44114 [https://bugs.python.org/issue?@action=redirect&bpo=44114]: Fix incorrect dictkeys_reversed and dictitems_reversed function signatures in C code, which broke webassembly builds.

- bpo-44110 [https://bugs.python.org/issue?@action=redirect&bpo=44110]: Improve **`str.__getitem__()`** error message

- bpo-26110 [https://bugs.python.org/issue?@action=redirect&bpo=26110]: Add `CALL_METHOD_KW` opcode to speed up method calls with keyword arguments. Idea originated from PyPy. A side effect is executing `CALL_METHOD` is now branchless in the evaluation loop.

- bpo-28307 [https://bugs.python.org/issue?@action=redirect&bpo=28307]: Compiler now optimizes simple C-style formatting with literal format containing only format codes %s, %r and %a by converting them to f-string expressions.

- bpo-43149 [https://bugs.python.org/issue?@action=redirect&bpo=43149]: Correct the syntax error message regarding multiple exception types to not refer to «exception groups». Patch by Pablo Galindo

- bpo-43822 [https://bugs.python.org/issue?@action=redirect&bpo=43822]: The parser will prioritize tokenizer errors over custom syntax errors when raising exceptions. Patch by Pablo Galindo.

- bpo-40222 [https://bugs.python.org/issue?@action=redirect&bpo=40222]: «Zero cost» exception handling.

    - Uses a lookup table to determine how to handle exceptions.
    - Removes SETUP_FINALLY and POP_TOP block instructions, eliminating the runtime overhead of try statements.
    - Reduces the size of the frame object by about 60%.

  Patch by Mark Shannon

- bpo-43918 [https://bugs.python.org/issue?@action=redirect&bpo=43918]: Document the signature and `default` argument in the docstring of the new `anext` builtin.

- bpo-43833 [https://bugs.python.org/issue?@action=redirect&bpo=43833]: Emit a deprecation warning if the numeric literal is immediately followed by one of keywords: and, else, for, if, in, is, or. Raise a syntax error with more informative message if it is immediately followed by other keyword or identifier.

- bpo-43879 [https://bugs.python.org/issue?@action=redirect&bpo=43879]: Add native_thread_id to PyThreadState. Patch by Gabriele N. Tornetta.

- bpo-43693 [https://bugs.python.org/issue?@action=redirect&bpo=43693]: Compute cell offsets relative to locals in compiler. Allows the interpreter to treats locals and cells a single array, which is slightly more efficient. Also make the LOAD_CLOSURE opcode an alias for LOAD_FAST. Preserving LOAD_CLOSURE helps keep bytecode a bit more readable.

- bpo-17792 [https://bugs.python.org/issue?@action=redirect&bpo=17792]: More accurate error messages for access of unbound locals or free vars.

- bpo-28146 [https://bugs.python.org/issue?@action=redirect&bpo=28146]: Fix a confusing error message in **`str.format()`**.

- bpo-11105 [https://bugs.python.org/issue?@action=redirect&bpo=11105]: When compiling **`ast.AST`** objects with recursive references through **`compile()`**, the interpreter doesn't crash anymore instead it raises a **`RecursionError`**.

- bpo-39091 [https://bugs.python.org/issue?@action=redirect&bpo=39091]: Fix crash when using passing a non-exception to a generator's `throw()` method. Patch by Noah Oxer

- bpo-33346 [https://bugs.python.org/issue?

@action=redirect&bpo=33346]: Asynchronous comprehensions are now allowed inside comprehensions in asynchronous functions. Outer comprehensions implicitly become asynchronous.

Library

- bpo-45371 [https://bugs.python.org/issue?@action=redirect&bpo=45371]: Fix clang rpath issue in `distutils`. The UnixCCompiler now uses correct clang option to add a runtime library directory (rpath) to a shared library.
- bpo-45329 [https://bugs.python.org/issue?@action=redirect&bpo=45329]: Fix freed memory access in **`pyexpat.xmlparser`** when building it with an installed expat library <= 2.2.0.
- bpo-41710 [https://bugs.python.org/issue?@action=redirect&bpo=41710]: On Unix, if the `sem_clockwait()` function is available in the C library (glibc 2.30 and newer), the **`threading.Lock.acquire()`** method now uses the monotonic clock (**`time.CLOCK_MONOTONIC`**) for the timeout, rather than using the system clock (**`time.CLOCK_REALTIME`**), to not be affected by system clock changes. Patch by Victor Stinner.
- bpo-1596321 [https://bugs.python.org/issue?@action=redirect&bpo=1596321]: Fix the **`threading._shutdown()`** function when the **`threading`** module was imported first from a thread different than the main thread: no longer log an error at Python exit.
- bpo-45274 [https://bugs.python.org/issue?@action=redirect&bpo=45274]: Fix a race condition in the **`Thread.join()`** method of the **`threading`** module. If the function is interrupted by a signal and the signal handler raises an exception, make sure that the thread remains in a consistent state to prevent a deadlock. Patch by Victor Stinner.

- bpo-21302 [https://bugs.python.org/issue?@action=redirect&bpo=21302]: In Unix operating systems, `time.sleep()` now uses the `nanosleep()` function, if `clock_nanosleep()` is not available but `nanosleep()` is available. `nanosleep()` allows to sleep with nanosecond precision.

- bpo-21302 [https://bugs.python.org/issue?@action=redirect&bpo=21302]: On Windows, `time.sleep()` now uses a waitable timer which has a resolution of 100 nanoseconds (10^{-7} seconds). Previously, it had a resolution of 1 millisecond (10^{-3} seconds). Patch by Benjamin Szőke and Victor Stinner.

- bpo-45238 [https://bugs.python.org/issue?@action=redirect&bpo=45238]: Fix **`unittest.IsolatedAsyncioTestCase.debug()`**: it runs now asynchronous methods and callbacks.

- bpo-36674 [https://bugs.python.org/issue?@action=redirect&bpo=36674]: `unittest.TestCase.debug()` raises now a `unittest.SkipTest` if the class or the test method are decorated with the skipping decorator.

- bpo-45235 [https://bugs.python.org/issue?@action=redirect&bpo=45235]: Fix an issue where argparse would not preserve values in a provided namespace when using a subparser with defaults.

- bpo-45183 [https://bugs.python.org/issue?@action=redirect&bpo=45183]: Have zipimport.zipimporter.find_spec() not raise an exception when the underlying zip file has been deleted and the internal cache has been reset via invalidate_cache().

- bpo-45234 [https://bugs.python.org/issue?@action=redirect&bpo=45234]: Fixed a regression in `copyfile()`, `copy()`, `copy2()` raising `FileNotFoundError` when source is a directory, which should raise `IsADirectoryError`

- bpo-45228 [https://bugs.python.org/issue?@action=redirect&bpo=45228]: Fix stack buffer overflow in parsing J1939 network address.

- bpo-45225 [https://bugs.python.org/issue?@action=redirect&bpo=45225]: use map function instead of genexpr in capwords.

- bpo-42135 [https://bugs.python.org/issue?@action=redirect&bpo=42135]: Fix typo: `importlib.find_loader` is really slated for removal in Python 3.12 not 3.10, like the others in PR 25169.

  Patch by Hugo van Kemenade.

- bpo-20524 [https://bugs.python.org/issue?@action=redirect&bpo=20524]: Improves error messages on `.format()` operation for `str`, `float`, `int`, and `complex`. New format now shows the problematic pattern and the object type.

- bpo-45168 [https://bugs.python.org/issue?@action=redirect&bpo=45168]: Change **`dis.dis()`** output to omit op arg values that cannot be resolved due to `co_consts`, `co_names` etc not being provided. Previously the oparg itself was repeated in the value field, which is not useful and can be confusing.

- bpo-21302 [https://bugs.python.org/issue?@action=redirect&bpo=21302]: In Unix operating systems, **`time.sleep()`** now uses the `clock_nanosleep()` function, if available, which allows to sleep for an interval specified with nanosecond precision.

- bpo-45173 [https://bugs.python.org/issue?@action=redirect&bpo=45173]: Remove from the **`configparser`** module: the **`SafeConfigParser`** class, the **`filename`** property of the **`ParsingError`** class, the **`readfp()`** method of the **`ConfigParser`** class, deprecated since Python 3.2.

Patch by Hugo van Kemenade.

- bpo-44987 [https://bugs.python.org/issue?@action=redirect&bpo=44987]: Pure ASCII strings are now normalized in constant time by **`unicodedata.normalize()`**. Patch by Donghee Na.

- bpo-35474 [https://bugs.python.org/issue?@action=redirect&bpo=35474]: Calling **`mimetypes.guess_all_extensions()`** with `strict=False` no longer affects the result of the following call with `strict=True`. Also, mutating the returned list no longer affects the global state.

- bpo-45166 [https://bugs.python.org/issue?@action=redirect&bpo=45166]: **`typing.get_type_hints()`** now works with **`Final`** wrapped in **`ForwardRef`**.

- bpo-45162 [https://bugs.python.org/issue?@action=redirect&bpo=45162]: Remove many old deprecated **`unittest`** features:

    - «`fail*`» and «`assert*`» aliases of **`TestCase`** methods.
    - Broken from start **`TestCase`** method `assertDictContainsSubset()`.
    - Ignored **`<unittest.TestLoader.loadTestsFromModule> TestLoader.loadTestsFromModule()`** parameter *use_load_tests*.
    - Old alias `_TextTestResult` of **`TextTestResult`**.

- bpo-38371 [https://bugs.python.org/issue?@action=redirect&bpo=38371]: Remove the deprecated `split()` method of **`_tkinter.TkappType`**. Patch by Erlend E. Aasland.

- bpo-20499 [https://bugs.python.org/issue?@action=redirect&bpo=20499]: Improve the speed and accuracy of statistics.pvariance().

- bpo-45132 [https://bugs.python.org/issue?@action=redirect&bpo=45132]: Remove `__getitem__()` methods of `xml.dom.pulldom.DOMEventStream`, `wsgiref.util.FileWrapper` and `fileinput.FileInput`, deprecated since Python 3.9.

  Patch by Hugo van Kemenade.

- bpo-45129 [https://bugs.python.org/issue?@action=redirect&bpo=45129]: Due to significant security concerns, the *reuse_address* parameter of `asyncio.loop.create_datagram_endpoint()`, disabled in Python 3.9, is now entirely removed. This is because of the behavior of the socket option `SO_REUSEADDR` in UDP.

  Patch by Hugo van Kemenade.

- bpo-45124 [https://bugs.python.org/issue?@action=redirect&bpo=45124]: The `bdist_msi` command, deprecated in Python 3.9, is now removed.

  Use `bdist_wheel` (wheel packages) instead.

  Patch by Hugo van Kemenade.

- bpo-30856 [https://bugs.python.org/issue?@action=redirect&bpo=30856]: `unittest.TestResult` methods `addFailure()`, `addError()`, `addSkip()` and `addSubTest()` are now called immediately after raising an exception in test or finishing a subtest. Previously they were called only after finishing the test clean up.

- bpo-45034 [https://bugs.python.org/issue?@action=redirect&bpo=45034]: Changes how error is formatted for `struct.pack` with `'H'` and `'h'` modes and too large / small numbers. Now it shows the actual numeric limits, while previously it was showing arithmetic expressions.

- bpo-25894 [https://bugs.python.org/issue?@action=redirect&bpo=25894]: `unittest` now always reports skipped and failed subtests separately: separate characters in default mode and separate lines in verbose mode. Also the test description is now output for errors in test method, class and module cleanups.

- bpo-45081 [https://bugs.python.org/issue?@action=redirect&bpo=45081]: Fix issue when dataclasses that inherit from `typing.Protocol` subclasses have wrong `__init__`. Patch provided by Yurii Karabas.

- bpo-45085 [https://bugs.python.org/issue?@action=redirect&bpo=45085]: The `binhex` module, deprecated in Python 3.9, is now removed. The following **`binascii`** functions, deprecated in Python 3.9, are now also removed:

  - `a2b_hqx()`, `b2a_hqx()`;
  - `rlecode_hqx()`, `rledecode_hqx()`.

  The **`binascii.crc_hqx()`** function remains available.

  Patch by Victor Stinner.

- bpo-40360 [https://bugs.python.org/issue?@action=redirect&bpo=40360]: The **`lib2to3`** package is now deprecated and may not be able to parse Python 3.10 or newer. See the **PEP 617** [https://peps.python.org/pep-0617/] (New PEG parser for CPython). Patch by Victor Stinner.

- bpo-45075 [https://bugs.python.org/issue?@action=redirect&bpo=45075]: Rename **`traceback.StackSummary.format_frame()`** to **`traceback.StackSummary.format_frame_summary()`**. This method was added for 3.11 so it was not released yet.

  Updated code and docs to better distinguish frame and FrameSummary.

- bpo-31299 [https://bugs.python.org/issue?@action=redirect&bpo=31299]: Add option to completely drop frames from a traceback by returning `None` from a **`format_frame()`** override.

- bpo-41620 [https://bugs.python.org/issue?@action=redirect&bpo=41620]: `run()` now always return a `TestResult` instance. Previously it returned `None` if the test class or method was decorated with a skipping decorator.

- bpo-45021 [https://bugs.python.org/issue?@action=redirect&bpo=45021]: Fix a potential deadlock at shutdown of forked children when using `concurrent.futures` module

- bpo-43913 [https://bugs.python.org/issue?@action=redirect&bpo=43913]: Fix bugs in cleaning up classes and modules in `unittest`:

    - Functions registered with `addModuleCleanup()` were not called unless the user defines `tearDownModule()` in their test module.
    - Functions registered with `addClassCleanup()` were not called if `tearDownClass` is set to `None`.
    - Buffering in `TestResult` did not work with functions registered with `addClassCleanup()` and `addModuleCleanup()`.
    - Errors in functions registered with `addClassCleanup()` and `addModuleCleanup()` were not handled correctly in buffered and debug modes.
    - Errors in `setUpModule()` and functions registered with `addModuleCleanup()` were reported in wrong order.
    - And several lesser bugs.

- bpo-45030 [https://bugs.python.org/issue?@action=redirect&bpo=45030]: Fix integer overflow in pickling and copying the range iterator.

- bpo-45001 [https://bugs.python.org/issue?@action=redirect&bpo=45001]: Made email date parsing more robust against malformed input, namely a whitespace-only `Date:` header. Patch by Wouter Bolsterlee.

- bpo-45010 [https://bugs.python.org/issue?

@action=redirect&bpo=45010]: Remove support of special method `__div__` in **`unittest.mock`**. It is not used in Python 3.

- bpo-39218 [https://bugs.python.org/issue?@action=redirect&bpo=39218]: Improve accuracy of variance calculations by using `x*x` instead of `x**2`.

- bpo-43613 [https://bugs.python.org/issue?@action=redirect&bpo=43613]: Improve the speed of **`gzip.compress()`** and **`gzip.decompress()`** by compressing and decompressing at once in memory instead of in a streamed fashion.

- bpo-37596 [https://bugs.python.org/issue?@action=redirect&bpo=37596]: Ensure that **`set`** and **`frozenset`** objects are always **`marshalled`** reproducibly.

- bpo-44019 [https://bugs.python.org/issue?@action=redirect&bpo=44019]: A new function `operator.call` has been added, such that `operator.call(obj, *args, **kwargs) == obj(*args, **kwargs)`.

- bpo-42255 [https://bugs.python.org/issue?@action=redirect&bpo=42255]: **`webbrowser.MacOSX`** is deprecated and will be removed in Python 3.13. It is untested and undocumented and also not used by **`webbrowser`** itself. Patch by Donghee Na.

- bpo-44955 [https://bugs.python.org/issue?@action=redirect&bpo=44955]: Method **`stopTestRun()`** is now always called in pair with method **`startTestRun()`** for **`TestResult`** objects implicitly created in **`run()`**. Previously it was not called for test methods and classes decorated with a skipping decorator.

- bpo-39039 [https://bugs.python.org/issue?@action=redirect&bpo=39039]: tarfile.open raises **`ReadError`** when a zlib error occurs during file extraction.

- bpo-44935 [https://bugs.python.org/issue?

@action=redirect&bpo=44935]: `subprocess` on Solaris now also uses `os.posix_spawn()` for better performance.

- bpo-44911 [https://bugs.python.org/issue?@action=redirect&bpo=44911]: `IsolatedAsyncioTestCase` will no longer throw an exception while cancelling leaked tasks. Patch by Bar Harel.
- bpo-41322 [https://bugs.python.org/issue?@action=redirect&bpo=41322]: Added `DeprecationWarning` for tests and async tests that return a value!=None (as this may indicate an improperly written test, for example a test written as a generator function).
- bpo-44524 [https://bugs.python.org/issue?@action=redirect&bpo=44524]: Make exception message more useful when subclass from typing special form alias. Patch provided by Yurii Karabas.
- bpo-38956 [https://bugs.python.org/issue?@action=redirect&bpo=38956]: `argparse.BooleanOptionalAction`'s default value is no longer printed twice when used with `argparse.ArgumentDefaultsHelpFormatter`.
- bpo-44860 [https://bugs.python.org/issue?@action=redirect&bpo=44860]: Fix the `posix_user` scheme in `sysconfig` to not depend on `sys.platlibdir`.
- bpo-44859 [https://bugs.python.org/issue?@action=redirect&bpo=44859]: Improve error handling in `sqlite3` and raise more accurate exceptions.
  - `MemoryError` is now raised instead of `sqlite3.Warning` when memory is not enough for encoding a statement to UTF-8 in `Connection.__call__()` and `Cursor.execute()`.
  - `UnicodEncodeError` is now raised instead of `sqlite3.Warning` when the statement contains surrogate characters in `Connection.__call__()`

  and `Cursor.execute()`.
  - **`TypeError`** is now raised instead of **`ValueError`** for non-string script argument in `Cursor.executescript()`.
  - **`ValueError`** is now raised for script containing the null character instead of truncating it in `Cursor.executescript()`.
  - Correctly handle exceptions raised when getting boolean value of the result of the progress handler.
  - Add many tests covering different corner cases.

- bpo-44581 [https://bugs.python.org/issue?@action=redirect&bpo=44581]: Upgrade bundled pip to 21.2.3 and setuptools to 57.4.0

- bpo-44849 [https://bugs.python.org/issue?@action=redirect&bpo=44849]: Fix the **`os.set_inheritable()`** function on FreeBSD 14 for file descriptor opened with the **`O_PATH`** flag: ignore the **`EBADF`** error on `ioctl()`, fallback on the `fcntl()` implementation. Patch by Victor Stinner.

- bpo-44605 [https://bugs.python.org/issue?@action=redirect&bpo=44605]: The @functools.total_ordering() decorator now works with metaclasses.

- bpo-44524 [https://bugs.python.org/issue?@action=redirect&bpo=44524]: Fixed an issue wherein the `__name__` and `__qualname__` attributes of subscribed specialforms could be `None`.

- bpo-44839 [https://bugs.python.org/issue?@action=redirect&bpo=44839]: **`MemoryError`** raised in user-defined functions will now produce a `MemoryError` in **`sqlite3`**. **`OverflowError`** will now be converted to **`DataError`**. Previously **`OperationalError`** was produced in these cases.

- bpo-44822 [https://bugs.python.org/issue?@action=redirect&bpo=44822]: **`sqlite3`** user-defined functions and aggregators returning **`strings`** with embedded NUL characters are no longer truncated. Patch by Erlend E. Aasland.

- bpo-44801 [https://bugs.python.org/issue?@action=redirect&bpo=44801]: Ensure that the `ParamSpec` variable in Callable can only be substituted with a parameters expression (a list of types, an ellipsis, ParamSpec or Concatenate).

- bpo-44806 [https://bugs.python.org/issue?@action=redirect&bpo=44806]: Non-protocol subclasses of `typing.Protocol` ignore now the `__init__` method inherited from protocol base classes.

- bpo-27275 [https://bugs.python.org/issue?@action=redirect&bpo=27275]: `collections.OrderedDict.popitem()` and `collections.OrderedDict.pop()` no longer call `__getitem__` and `__delitem__` methods of the OrderedDict subclasses.

- bpo-44793 [https://bugs.python.org/issue?@action=redirect&bpo=44793]: Fix checking the number of arguments when subscribe a generic type with `ParamSpec` parameter.

- bpo-44784 [https://bugs.python.org/issue?@action=redirect&bpo=44784]: In importlib.metadata tests, override warnings behavior under expected DeprecationWarnings (importlib_metadata 4.6.3).

- bpo-44667 [https://bugs.python.org/issue?@action=redirect&bpo=44667]: The `tokenize.tokenize()` doesn't incorrectly generate a `NEWLINE` token if the source doesn't end with a new line character but the last line is a comment, as the function is already generating a `NL` token. Patch by Pablo Galindo

- bpo-44771 [https://bugs.python.org/issue?@action=redirect&bpo=44771]: Added `importlib.simple` module implementing adapters from a low-level resources reader interface to a `TraversableResources` interface. Legacy API (`path`, `contents`, …) is now supported entirely by the `.files()` API with a compatibility shim supplied for resource loaders without that functionality. Feature parity with `importlib_resources` 5.2.

- bpo-44752 [https://bugs.python.org/issue?@action=redirect&bpo=44752]: **`rcompleter`** does not call **`getattr()`** on **`property`** objects to avoid the side-effect of evaluating the corresponding method.

- bpo-44747 [https://bugs.python.org/issue?@action=redirect&bpo=44747]: Refactor usage of `sys._getframe` in `typing` module. Patch provided by Yurii Karabas.

- bpo-42378 [https://bugs.python.org/issue?@action=redirect&bpo=42378]: Fixes the issue with log file being overwritten when **`logging.FileHandler`** is used in **`atexit`** with *filemode* set to `'w'`. Note this will cause the message in *atexit* not being logged if the log stream is already closed due to shutdown of logging.

- bpo-44720 [https://bugs.python.org/issue?@action=redirect&bpo=44720]: `weakref.proxy` objects referencing non-iterators now raise `TypeError` rather than dereferencing the null `tp_iternext` slot and crashing.

- bpo-44704 [https://bugs.python.org/issue?@action=redirect&bpo=44704]: The implementation of `collections.abc.Set._hash()` now matches that of `frozenset.__hash__()`.

- bpo-44666 [https://bugs.python.org/issue?@action=redirect&bpo=44666]: Fixed issue in **`compileall.compile_file()`** when `sys.stdout` is redirected. Patch by Stefan Hölzl.

- bpo-44688 [https://bugs.python.org/issue?@action=redirect&bpo=44688]: **`sqlite3.Connection.create_collation()`** now accepts non-ASCII collation names. Patch by Erlend E. Aasland.

- bpo-44690 [https://bugs.python.org/issue?@action=redirect&bpo=44690]: Adopt *binacii.a2b_base64*'s strict mode in *base64.b64decode*.

- bpo-42854 [https://bugs.python.org/issue?@action=redirect&bpo=42854]: Fixed a bug in the **`_ssl`** module that was throwing **`OverflowError`** when using **`_ssl._SSLSocket.write()`** and **`_ssl._SSLSocket.read()`** for a big value of the `len` parameter. Patch by Pablo Galindo.

- bpo-44686 [https://bugs.python.org/issue?@action=redirect&bpo=44686]: Replace `unittest.mock._importer` with `pkgutil.resolve_name`.

- bpo-44353 [https://bugs.python.org/issue?@action=redirect&bpo=44353]: Make `NewType.__call__` faster by implementing it in C. Patch provided by Yurii Karabas.

- bpo-44682 [https://bugs.python.org/issue?@action=redirect&bpo=44682]: Change the **`pdb`** *commands* directive to disallow setting commands for an invalid breakpoint and to display an appropriate error.

- bpo-44353 [https://bugs.python.org/issue?@action=redirect&bpo=44353]: Refactor `typing.NewType` from function into callable class. Patch provided by Yurii Karabas.

- bpo-44678 [https://bugs.python.org/issue?@action=redirect&bpo=44678]: Added a separate error message for discontinuous padding in *binascii.a2b_base64* strict mode.

- bpo-44524 [https://bugs.python.org/issue?@action=redirect&bpo=44524]: Add missing `__name__` and `__qualname__` attributes to `typing` module classes. Patch provided by Yurii Karabas.

- bpo-40897 [https://bugs.python.org/issue?@action=redirect&bpo=40897]: Give priority to using the current class constructor in `inspect.signature()`. Patch by Weipeng Hong.

- bpo-44638 [https://bugs.python.org/issue?@action=redirect&bpo=44638]: Add a reference to the zipp project and hint as to how to use it.

- bpo-44648 [https://bugs.python.org/issue?@action=redirect&bpo=44648]: Fixed wrong error being thrown by `inspect.getsource()` when examining a class in the interactive session. Instead of `TypeError`, it should be `OSError` with appropriate error message.

- bpo-44608 [https://bugs.python.org/issue?@action=redirect&bpo=44608]: Fix memory leak in `_tkinter._flatten()` if it is called with a sequence or set, but not list or tuple.

- bpo-44594 [https://bugs.python.org/issue?@action=redirect&bpo=44594]: Fix an edge case of `ExitStack` and `AsyncExitStack` exception chaining. They will now match `with` block behavior when `__context__` is explicitly set to `None` when the exception is in flight.

- bpo-42799 [https://bugs.python.org/issue?@action=redirect&bpo=42799]: In `fnmatch`, the cache size for compiled regex patterns (`functools.lru_cache()`) was bumped up from 256 to 32768, affecting functions: `fnmatch.fnmatch()`, `fnmatch.fnmatchcase()`, `fnmatch.filter()`.

- bpo-41928 [https://bugs.python.org/issue?@action=redirect&bpo=41928]: Update `shutil.copyfile()` to raise `FileNotFoundError` instead of confusing `IsADirectoryError` when a path ending with a `os.path.sep` does not exist; `shutil.copy()` and `shutil.copy2()` are also affected.

- bpo-44569 [https://bugs.python.org/issue?

@action=redirect&bpo=44569]: Added the **`StackSummary.format_frame()`** function in **`traceback`**. This allows users to customize the way individual lines are formatted in tracebacks without re-implementing logic to handle recursive tracebacks.

- bpo-44566 [https://bugs.python.org/issue?@action=redirect&bpo=44566]: handle StopIteration subclass raised from @contextlib.contextmanager generator

- bpo-44558 [https://bugs.python.org/issue?@action=redirect&bpo=44558]: Make the implementation consistency of **`indexOf()`** between C and Python versions. Patch by Donghee Na.

- bpo-41249 [https://bugs.python.org/issue?@action=redirect&bpo=41249]: Fixes `TypedDict` to work with `typing.get_type_hints()` and postponed evaluation of annotations across modules.

- bpo-44554 [https://bugs.python.org/issue?@action=redirect&bpo=44554]: Refactor argument processing in **`pdb.main()`** to simplify detection of errors in input loading and clarify behavior around module or script invocation.

- bpo-34798 [https://bugs.python.org/issue?@action=redirect&bpo=34798]: Break up paragraph about **`pprint.PrettyPrinter`** construction parameters to make it easier to read.

- bpo-44539 [https://bugs.python.org/issue?@action=redirect&bpo=44539]: Added support for recognizing JPEG files without JFIF or Exif markers.

- bpo-44461 [https://bugs.python.org/issue?@action=redirect&bpo=44461]: Fix bug with **`pdb`**'s handling of import error due to a package which does not have a `__main__` module

- bpo-43625 [https://bugs.python.org/issue?@action=redirect&bpo=43625]: Fix a bug in the detection of CSV file headers by `csv.Sniffer.has_header()` and improve documentation of same.

- bpo-44516 [https://bugs.python.org/issue?@action=redirect&bpo=44516]: Update vendored pip to 21.1.3

- bpo-42892 [https://bugs.python.org/issue?@action=redirect&bpo=42892]: Fixed an exception thrown while parsing a malformed multipart email by `email.message.EmailMessage`.

- bpo-44468 [https://bugs.python.org/issue?@action=redirect&bpo=44468]: `typing.get_type_hints()` now finds annotations in classes and base classes with unexpected `__module__`. Previously, it skipped those MRO elements.

- bpo-44491 [https://bugs.python.org/issue?@action=redirect&bpo=44491]: Allow clearing the `sqlite3` authorizer callback by passing `None` to `set_authorizer()`. Patch by Erlend E. Aasland.

- bpo-43977 [https://bugs.python.org/issue?@action=redirect&bpo=43977]: Set the proper `Py_TPFLAGS_MAPPING` and `Py_TPFLAGS_SEQUENCE` flags for subclasses created before a parent has been registered as a `collections.abc.Mapping` or `collections.abc.Sequence`.

- bpo-44482 [https://bugs.python.org/issue?@action=redirect&bpo=44482]: Fix very unlikely resource leak in `glob` in alternate Python implementations.

- bpo-44466 [https://bugs.python.org/issue?@action=redirect&bpo=44466]: The `faulthandler` module now detects if a fatal error occurs during a garbage collector collection. Patch by Victor Stinner.

- bpo-44471 [https://bugs.python.org/issue?@action=redirect&bpo=44471]: A `TypeError` is now raised instead of an `AttributeError` in

`contextlib.ExitStack.enter_context()` and **`contextlib.AsyncExitStack.enter_async_context()`** for objects which do not support the context manager or asynchronous context manager protocols correspondingly.

- bpo-44404 [https://bugs.python.org/issue?@action=redirect&bpo=44404]: **`tkinter`**'s `after()` method now supports callables without the `__name__` attribute.

- bpo-41546 [https://bugs.python.org/issue?@action=redirect&bpo=41546]: Make **`pprint`** (like the builtin `print`) not attempt to write to `stdout` when it is `None`.

- bpo-44458 [https://bugs.python.org/issue?@action=redirect&bpo=44458]: `BUFFER_BLOCK_SIZE` is now declared static, to avoid linking collisions when bz2, lmza or zlib are statically linked.

- bpo-44464 [https://bugs.python.org/issue?@action=redirect&bpo=44464]: Remove exception for flake8 in deprecated importlib.metadata interfaces. Sync with importlib_metadata 4.6.

- bpo-44446 [https://bugs.python.org/issue?@action=redirect&bpo=44446]: Take into account that `lineno` might be `None` in **`traceback.FrameSummary`**.

- bpo-44439 [https://bugs.python.org/issue?@action=redirect&bpo=44439]: Fix in **`bz2.BZ2File.write()`** / **`lzma.LZMAFile.write()`** methods, when the input data is an object that supports the buffer protocol, the file length may be wrong.

- bpo-44434 [https://bugs.python.org/issue?@action=redirect&bpo=44434]: _thread.start_new_thread() no longer calls PyThread_exit_thread() explicitly at the thread exit, the call was redundant. On Linux with the glibc, pthread_exit() aborts the whole process if dlopen() fails to open libgcc_s.so file (ex: EMFILE error). Patch by Victor Stinner.

- bpo-42972 [https://bugs.python.org/issue?@action=redirect&bpo=42972]: The _thread.RLock type now fully implement the GC protocol: add a traverse function and the `Py_TPFLAGS_HAVE_GC` flag. Patch by Victor Stinner.

- bpo-44422 [https://bugs.python.org/issue?@action=redirect&bpo=44422]: The `threading.enumerate()` function now uses a reentrant lock to prevent a hang on reentrant call. Patch by Victor Stinner.

- bpo-38291 [https://bugs.python.org/issue?@action=redirect&bpo=38291]: Importing typing.io or typing.re now prints a `DeprecationWarning`.

- bpo-37880 [https://bugs.python.org/issue?@action=redirect&bpo=37880]: argparse actions store_const and append_const each receive a default value of `None` when the `const` kwarg is not provided. Previously, this raised a `TypeError`.

- bpo-44389 [https://bugs.python.org/issue?@action=redirect&bpo=44389]: Fix deprecation of `ssl.OP_NO_TLSv1_3`

- bpo-27827 [https://bugs.python.org/issue?@action=redirect&bpo=27827]: `pathlib.PureWindowsPath.is_reserved()` now identifies a greater range of reserved filenames, including those with trailing spaces or colons.

- bpo-44395 [https://bugs.python.org/issue?@action=redirect&bpo=44395]: Fix `as_string()` to pass unixfrom properly. Patch by Donghee Na.

- bpo-34266 [https://bugs.python.org/issue?@action=redirect&bpo=34266]: Handle exceptions from parsing the arg of `pdb`'s run/restart command.

- bpo-44362 [https://bugs.python.org/issue?@action=redirect&bpo=44362]: Improve `ssl` module's deprecation messages, error reporting, and documentation for deprecations.

- bpo-44342 [https://bugs.python.org/issue?@action=redirect&bpo=44342]: [Enum] Change pickling from by-value to by-name.

- bpo-44356 [https://bugs.python.org/issue?@action=redirect&bpo=44356]: [Enum] Allow multiple data-type mixins if they are all the same.

- bpo-44351 [https://bugs.python.org/issue?@action=redirect&bpo=44351]: Restore back **`parse_makefile()`** in `distutils.sysconfig` because it behaves differently than the similar implementation in **`sysconfig`**.

- bpo-35800 [https://bugs.python.org/issue?@action=redirect&bpo=35800]: **`smtpd.MailmanProxy`** is now removed as it is unusable without an external module, `mailman`. Patch by Donghee Na.

- bpo-44357 [https://bugs.python.org/issue?@action=redirect&bpo=44357]: Added a function that returns cube root of the given number **`math.cbrt()`**

- bpo-44339 [https://bugs.python.org/issue?@action=redirect&bpo=44339]: Change `math.pow(±0.0, -math.inf)` to return `inf` instead of raising `ValueError`. This brings the special-case handling of `math.pow` into compliance with the IEEE 754 standard.

- bpo-44242 [https://bugs.python.org/issue?@action=redirect&bpo=44242]: Remove missing flag check from Enum creation and move into a `verify` decorator.

- bpo-44246 [https://bugs.python.org/issue?@action=redirect&bpo=44246]: In `importlib.metadata`, restore compatibility in the result from `Distribution.entry_points` (`EntryPoints`) to honor expectations in older implementations and issuing deprecation warnings for these cases: A. `EntryPoints` objects are once again mutable, allowing for `sort()` and other list-based mutation operations. Avoid deprecation warnings by casting to a mutable sequence (e.g. `list(dist.entry_points).sort()`). B. `EntryPoints` results once again allow for access by index. To avoid deprecation warnings, cast the result to a Sequence first (e.g. `tuple(dist.entry_points)[0]`).

- bpo-44246 [https://bugs.python.org/issue?@action=redirect&bpo=44246]: In importlib.metadata.entry_points, de-duplication of distributions no longer requires loading the full metadata for PathDistribution objects, improving entry point loading performance by ~10x.

- bpo-43858 [https://bugs.python.org/issue?@action=redirect&bpo=43858]: Added a function that returns a copy of a dict of logging levels: **`logging.getLevelNamesMapping()`**

- bpo-44260 [https://bugs.python.org/issue?@action=redirect&bpo=44260]: The **`random.Random`** constructor no longer reads system entropy without need.

- bpo-44254 [https://bugs.python.org/issue?@action=redirect&bpo=44254]: On Mac, give turtledemo button text a color that works on both light or dark background. Programmers cannot control the latter.

- bpo-44258 [https://bugs.python.org/issue?@action=redirect&bpo=44258]: Support PEP 515 for Fraction's initialization from string.

- bpo-44235 [https://bugs.python.org/issue?@action=redirect&bpo=44235]: Remove deprecated functions in the **`gettext`**. Patch by Donghee Na.

- bpo-38693 [https://bugs.python.org/issue?@action=redirect&bpo=38693]: Prefer f-strings to `.format` in importlib.resources.

- bpo-33693 [https://bugs.python.org/issue?@action=redirect&bpo=33693]: Importlib.metadata now prefers f-strings to .format.

- bpo-44241 [https://bugs.python.org/issue?@action=redirect&bpo=44241]: Incorporate minor tweaks from importlib_metadata 4.1: SimplePath protocol, support for Metadata 2.2.

- bpo-43216 [https://bugs.python.org/issue?@action=redirect&bpo=43216]: Remove the **`@asyncio.coroutine`** decorator enabling legacy generator-based coroutines to be compatible with async/await code; remove **`asyncio.coroutines.CoroWrapper`** used for wrapping legacy coroutine objects in the debug mode. The decorator has been deprecated since Python 3.8 and the removal was initially scheduled for Python 3.10. Patch by Illia Volochii.

- bpo-44210 [https://bugs.python.org/issue?@action=redirect&bpo=44210]: Make importlib.metadata._meta.PackageMetadata public.

- bpo-43643 [https://bugs.python.org/issue?@action=redirect&bpo=43643]: Declare readers.MultiplexedPath.name as a property per the spec.

- bpo-27334 [https://bugs.python.org/issue?@action=redirect&bpo=27334]: The **`sqlite3`** context manager now performs a rollback (thus releasing the database lock) if commit failed. Patch by Luca Citi and Erlend E. Aasland.

- bpo-4928 [https://bugs.python.org/issue?@action=redirect&bpo=4928]: Documented existing behavior on POSIX: NamedTemporaryFiles are not deleted when creating process is killed with SIGKILL

- bpo-44154 [https://bugs.python.org/issue?@action=redirect&bpo=44154]: Optimize **`fractions.Fraction`** pickling for large components.

- bpo-33433 [https://bugs.python.org/issue?@action=redirect&bpo=33433]: For IPv4 mapped IPv6 addresses (**RFC 4291** [https://datatracker.ietf.org/doc/html/rfc4291.html] Section 2.5.5.2), the `ipaddress.IPv6Address.is_private` check is deferred to the mapped IPv4 address. This solves a bug where public mapped IPv4 addresses were considered private by the IPv6 check.

- bpo-44150 [https://bugs.python.org/issue?@action=redirect&bpo=44150]: Add optional *weights* argument to statistics.fmean().

- bpo-44142 [https://bugs.python.org/issue?@action=redirect&bpo=44142]: `ast.unparse()` will now drop the redundant parentheses when tuples used as assignment targets (e.g in for loops).

- bpo-44145 [https://bugs.python.org/issue?@action=redirect&bpo=44145]: `hmac` computations were not releasing the GIL while calling the OpenSSL `HMAC_Update` C API (a new feature in 3.9). This unintentionally prevented parallel computation as other `hashlib` algorithms support.

- bpo-44095 [https://bugs.python.org/issue?@action=redirect&bpo=44095]: `zipfile.Path` now supports `zipfile.Path.stem`, `zipfile.Path.suffixes`, and `zipfile.Path.suffix` attributes.

- bpo-44077 [https://bugs.python.org/issue?@action=redirect&bpo=44077]: It’s now possible to receive the type of service (ToS), a.k.a. differentiated services (DS), a.k.a. differentiated services code point (DSCP) and explicit congestion notification (ECN) IP header fields with `socket.IP_RECVTOS`.

- bpo-37788 [https://bugs.python.org/issue?@action=redirect&bpo=37788]: Fix a reference leak when a Thread object is never joined.

- bpo-38908 [https://bugs.python.org/issue?

@action=redirect&bpo=38908]: Subclasses of `typing.Protocol` which only have data variables declared will now raise a `TypeError` when checked with `isinstance` unless they are decorated with **`runtime_checkable()`**. Previously, these checks passed silently. Patch provided by Yurii Karabas.

- bpo-44098 [https://bugs.python.org/issue?@action=redirect&bpo=44098]: `typing.ParamSpec` will no longer be found in the `__parameters__` of most **`typing`** generics except in valid use locations specified by **PEP 612** [https://peps.python.org/pep-0612/]. This prevents incorrect usage like `typing.List[P][int]`. This change means incorrect usage which may have passed silently in 3.10 beta 1 and earlier will now error.

- bpo-44089 [https://bugs.python.org/issue?@action=redirect&bpo=44089]: Allow subclassing `csv.Error` in 3.10 (it was allowed in 3.9 and earlier but was disallowed in early versions of 3.10).

- bpo-44081 [https://bugs.python.org/issue?@action=redirect&bpo=44081]: **`ast.unparse()`** now doesn’t use redundant spaces to separate `lambda` and the `:` if there are no parameters.

- bpo-44061 [https://bugs.python.org/issue?@action=redirect&bpo=44061]: Fix regression in previous release when calling **`pkgutil.iter_modules()`** with a list of **`pathlib.Path`** objects

- bpo-44059 [https://bugs.python.org/issue?@action=redirect&bpo=44059]: Register the SerenityOS Browser in the **`webbrowser`** module.

- bpo-36515 [https://bugs.python.org/issue?@action=redirect&bpo=36515]: The **`hashlib`** module no longer does unaligned memory accesses when compiled for ARM platforms.

- bpo-40465 [https://bugs.python.org/issue?

@action=redirect&bpo=40465]: Remove random module features deprecated in Python 3.9.

- bpo-44018 [https://bugs.python.org/issue?@action=redirect&bpo=44018]: random.seed() no longer mutates bytearray inputs.
- bpo-38352 [https://bugs.python.org/issue?@action=redirect&bpo=38352]: Add `IO`, `BinaryIO`, `TextIO`, `Match`, and `Pattern` to `typing.__all__`. Patch by Jelle Zijlstra.
- bpo-44002 [https://bugs.python.org/issue?@action=redirect&bpo=44002]: **`urllib.parse`** now uses **`functool.lru_cache()`** for its internal URL splitting and quoting caches instead of rolling its own like its the "90s.

  The undocumented internal **`urllib.parse`** `Quoted` class API is now deprecated, for removal in 3.14.
- bpo-43972 [https://bugs.python.org/issue?@action=redirect&bpo=43972]: When **`http.server.SimpleHTTPRequestHandler`** sends a `301 (Moved Permanently)` for a directory path not ending with `/`, add a `Content-Length: 0` header. This improves the behavior for certain clients.
- bpo-28528 [https://bugs.python.org/issue?@action=redirect&bpo=28528]: Fix a bug in **`pdb`** where **`checkline()`** raises **`AttributeError`** if it is called after **`reset()`**.
- bpo-43853 [https://bugs.python.org/issue?@action=redirect&bpo=43853]: Improved string handling for **`sqlite3`** user-defined functions and aggregates:
  - ○ It is now possible to pass strings with embedded null characters to UDFs
  - ○ Conversion failures now correctly raise **`MemoryError`**

  Patch by Erlend E. Aasland.

- bpo-43666 [https://bugs.python.org/issue?@action=redirect&bpo=43666]: AIX: `Lib/_aix_support.get_platform()` may fail in an AIX WPAR. The fileset bos.rte appears to have a builddate in both LPAR and WPAR so this fileset is queried rather than bos.mp64. To prevent a similar situation (no builddate in ODM) a value (9988) sufficient for completing a build is provided. Patch by M Felt.

- bpo-43650 [https://bugs.python.org/issue?@action=redirect&bpo=43650]: Fix **`MemoryError`** in **`shutil.unpack_archive()`** which fails inside **`shutil._unpack_zipfile()`** on large files. Patch by Igor Bolshakov.

- bpo-43612 [https://bugs.python.org/issue?@action=redirect&bpo=43612]: **`zlib.compress()`** now accepts a wbits parameter which allows users to compress data as a raw deflate block without zlib headers and trailers in one go. Previously this required instantiating a `zlib.compressobj`. It also provides a faster alternative to `gzip.compress` when wbits=31 is used.

- bpo-43392 [https://bugs.python.org/issue?@action=redirect&bpo=43392]: **`importlib._bootstrap._find_and_load()`** now implements a two-step check to avoid locking when modules have been already imported and are ready. This improves performance of repeated calls to **`importlib.import_module()`** and **`importlib.__import__()`**.

- bpo-43318 [https://bugs.python.org/issue?@action=redirect&bpo=43318]: Fix a bug where **`pdb`** does not always echo cleared breakpoints.

- bpo-43234 [https://bugs.python.org/issue?@action=redirect&bpo=43234]: Prohibit passing non-**`concurrent.futures.ThreadPoolExecutor`** executors to **`loop.set_default_executor()`** following a deprecation in Python 3.8. Patch by Illia Volochii.

- bpo-43232 [https://bugs.python.org/issue?@action=redirect&bpo=43232]: Prohibit previously deprecated potentially disruptive operations on **`asyncio.trsock.TransportSocket`**. Patch by Illia Volochii.

- bpo-30077 [https://bugs.python.org/issue?@action=redirect&bpo=30077]: Added support for Apple's aifc/sowt pseudo-compression

- bpo-42971 [https://bugs.python.org/issue?@action=redirect&bpo=42971]: Add definition of `errno.EQFULL` for platforms that define this constant (such as macOS).

- bpo-43086 [https://bugs.python.org/issue?@action=redirect&bpo=43086]: Added a new optional `strict_mode` parameter to *binascii.a2b_base64*. When `scrict_mode` is set to `True`, the *a2b_base64* function will accept only valid base64 content. More details about what «valid base64 content» is, can be found in the function's documentation.

- bpo-43024 [https://bugs.python.org/issue?@action=redirect&bpo=43024]: Improve the help signature of **`traceback.print_exception()`**, **`traceback.format_exception()`** and **`traceback.format_exception_only()`**.

- bpo-33809 [https://bugs.python.org/issue?@action=redirect&bpo=33809]: Add the **`traceback.TracebackException.print()`** method which prints the formatted exception information.

- bpo-42862 [https://bugs.python.org/issue?@action=redirect&bpo=42862]: **`sqlite3`** now utilizes **`functools.lru_cache()`** to implement the connection statement cache. As a small optimisation, the default statement cache size has been increased from 100 to 128. Patch by Erlend E. Aasland.

- bpo-41818 [https://bugs.python.org/issue?@action=redirect&bpo=41818]: Soumendra Ganguly: add termios.tcgetwinsize(), termios.tcsetwinsize().

- bpo-40497 [https://bugs.python.org/issue?@action=redirect&bpo=40497]: **`subprocess.check_output()`** now raises **`ValueError`** when the invalid keyword argument *check* is passed by user code. Previously such use would fail later with a **`TypeError`**. Patch by Rémi Lapeyre.

- bpo-37449 [https://bugs.python.org/issue?@action=redirect&bpo=37449]: `ensurepip` now uses `importlib.resources.files()` traversable APIs

- bpo-40956 [https://bugs.python.org/issue?@action=redirect&bpo=40956]: Use Argument Clinic in **`sqlite3`**. Patches by Erlend E. Aasland.

- bpo-41730 [https://bugs.python.org/issue?@action=redirect&bpo=41730]: `DeprecationWarning` is now raised when importing **`tkinter.tix`**, which has been deprecated in documentation since Python 3.6.

- bpo-20684 [https://bugs.python.org/issue?@action=redirect&bpo=20684]: Remove unused `_signature_get_bound_param` function from **`inspect`** - by Anthony Sottile.

- bpo-41402 [https://bugs.python.org/issue?@action=redirect&bpo=41402]: Fix **`email.message.EmailMessage.set_content()`** when called with binary data and `7bit` content transfer encoding.

- bpo-32695 [https://bugs.python.org/issue?@action=redirect&bpo=32695]: The *compresslevel* and *preset* keyword arguments of **`tarfile.open()`** are now both documented and tested.

- bpo-41137 [https://bugs.python.org/issue?@action=redirect&bpo=41137]: Use utf-8 encoding while reading .pdbrc files. Patch by Srinivas Reddy Thatiparthy

- bpo-24391 [https://bugs.python.org/issue?@action=redirect&bpo=24391]: Improved reprs of `threading` synchronization objects: `Semaphore`, `BoundedSemaphore`, `Event` and `Barrier`.

- bpo-5846 [https://bugs.python.org/issue?@action=redirect&bpo=5846]: Deprecated the following `unittest` functions, scheduled for removal in Python 3.13:

  - `findTestCases()`
  - `makeSuite()`
  - `getTestCaseNames()`

  Use `TestLoader` methods instead:

  - `unittest.TestLoader.loadTestsFromModule()`
  - `unittest.TestLoader.loadTestsFromTestCase()`
  - `unittest.TestLoader.getTestCaseNames()`

  Patch by Erlend E. Aasland.

- bpo-40563 [https://bugs.python.org/issue?@action=redirect&bpo=40563]: Support pathlike objects on dbm/shelve. Patch by Hakan Çelik and Henry-Joseph Audéoud.

- bpo-34990 [https://bugs.python.org/issue?@action=redirect&bpo=34990]: Fixed a Y2k38 bug in the compileall module where it would fail to compile files with a modification time after the year 2038.

- bpo-39549 [https://bugs.python.org/issue?@action=redirect&bpo=39549]: Whereas the code for reprlib.Repr had previously used a hardcoded string value of "…", this PR updates it to use of a "fillvalue" attribute, whose value defaults to "…" and can be reset in either individual reprlib.Repr instances or in subclasses thereof.

- bpo-37022 [https://bugs.python.org/issue?@action=redirect&bpo=37022]: `pdb` now displays exceptions from `repr()` with its `p` and `pp` commands.

- bpo-38840 [https://bugs.python.org/issue?

@action=redirect&bpo=38840]: Fix `test___all__` on platforms lacking a shared memory implementation.

- bpo-39359 [https://bugs.python.org/issue?@action=redirect&bpo=39359]: Add one missing check that the password is a bytes object for an encrypted zipfile.
- bpo-38741 [https://bugs.python.org/issue?@action=redirect&bpo=38741]: **`configparser`**: using "]" inside a section header will no longer cut the section name short at the "]"
- bpo-38415 [https://bugs.python.org/issue?@action=redirect&bpo=38415]: Added missing behavior to **`contextlib.asynccontextmanager()`** to match **`contextlib.contextmanager()`** so decorated functions can themselves be decorators.
- bpo-30256 [https://bugs.python.org/issue?@action=redirect&bpo=30256]: Pass multiprocessing BaseProxy argument `manager_owned` through AutoProxy.
- bpo-27513 [https://bugs.python.org/issue?@action=redirect&bpo=27513]: **`email.utils.getaddresses()`** now accepts **`email.header.Header`** objects along with string values. Patch by Zackery Spytz.
- bpo-16379 [https://bugs.python.org/issue?@action=redirect&bpo=16379]: Add SQLite error code and name to **`sqlite3`** exceptions. Patch by Aviv Palivoda, Daniel Shahaf, and Erlend E. Aasland.
- bpo-26228 [https://bugs.python.org/issue?@action=redirect&bpo=26228]: pty.spawn no longer hangs on FreeBSD, macOS, and Solaris.
- bpo-33349 [https://bugs.python.org/issue?@action=redirect&bpo=33349]: lib2to3 now recognizes async generators everywhere.

- bpo-29298 [https://bugs.python.org/issue?@action=redirect&bpo=29298]: Fix `TypeError` when required subparsers without `dest` do not receive arguments. Patch by Anthony Sottile.

Documentation

- bpo-45216 [https://bugs.python.org/issue?@action=redirect&bpo=45216]: Remove extra documentation listing methods in `difflib`. It was rendering twice in pydoc and was outdated in some places.
- bpo-45024 [https://bugs.python.org/issue?@action=redirect&bpo=45024]: **`collections.abc`** documentation has been expanded to explicitly cover how instance and subclass checks work, with additional doctest examples and an exhaustive list of ABCs which test membership purely by presence of the right special methods. Patch by Raymond Hettinger.
- bpo-44957 [https://bugs.python.org/issue?@action=redirect&bpo=44957]: Promote PEP 604 union syntax by using it where possible. Also, mention `X | Y` more prominently in section about `Union` and mention `X | None` at all in section about `Optional`.
- bpo-16580 [https://bugs.python.org/issue?@action=redirect&bpo=16580]: Added code equivalents for the **`int.to_bytes()`** and **`int.from_bytes()`** methods, as well as tests ensuring that these code equivalents are valid.
- bpo-44903 [https://bugs.python.org/issue?@action=redirect&bpo=44903]: Removed the `othergui.rst` file, any references to it, and the list of GUI frameworks in the FAQ. In their place I've added links to the Python Wiki page on GUI frameworks [https://wiki.python.org/moin/GuiProgramming].
- bpo-33479 [https://bugs.python.org/issue?@action=redirect&bpo=33479]: Tkinter documentation has been greatly expanded with new «Architecture» and «Threading model» sections.
- bpo-36700 [https://bugs.python.org/issue?@action=redirect&bpo=36700]: **`base64`** RFC references were updated to point to **RFC 4648** [https://datatracker.ietf.org/doc/html/rfc4648.html]; a section was added to point users to the

new «security considerations» section of the RFC.

- bpo-44740 [https://bugs.python.org/issue?@action=redirect&bpo=44740]: Replaced occurrences of uppercase «Web» and «Internet» with lowercase versions per the 2016 revised Associated Press Style Book.
- bpo-44693 [https://bugs.python.org/issue?@action=redirect&bpo=44693]: Update the definition of __future__ in the glossary by replacing the confusing word «pseudo-module» with a more accurate description.
- bpo-35183 [https://bugs.python.org/issue?@action=redirect&bpo=35183]: Add typical examples to os.path.splitext docs
- bpo-30511 [https://bugs.python.org/issue?@action=redirect&bpo=30511]: Clarify that **`shutil.make_archive()`** is not thread-safe due to reliance on changing the current working directory.
- bpo-44561 [https://bugs.python.org/issue?@action=redirect&bpo=44561]: Update of three expired hyperlinks in Doc/distributing/index.rst: «Project structure», «Building and packaging the project», and «Uploading the project to the Python Packaging Index».
- bpo-44651 [https://bugs.python.org/issue?@action=redirect&bpo=44651]: Delete entry «coercion» in Doc/glossary.rst for its outdated definition.
- bpo-42958 [https://bugs.python.org/issue?@action=redirect&bpo=42958]: Updated the docstring and docs of **`filecmp.cmp()`** to be more accurate and less confusing especially in respect to *shallow* arg.
- bpo-44631 [https://bugs.python.org/issue?@action=redirect&bpo=44631]: Refactored the `repr()` code of the `_Environ` (os module).
- bpo-44613 [https://bugs.python.org/issue?@action=redirect&bpo=44613]: importlib.metadata is no longer provisional.
- bpo-44558 [https://bugs.python.org/issue?@action=redirect&bpo=44558]: Match the docstring and python implementation of **`countOf()`** to the behavior of its c implementation.
- bpo-44544 [https://bugs.python.org/issue?@action=redirect&bpo=44544]: List all kwargs for

`textwrap.wrap()`, `textwrap.fill()`, and `textwrap.shorten()`. Now, there are nav links to attributes of `TextWrap`, which makes navigation much easier while minimizing duplication in the documentation.

- bpo-38062 [https://bugs.python.org/issue?@action=redirect&bpo=38062]: Clarify that atexit uses equality comparisons internally.
- bpo-40620 [https://bugs.python.org/issue?@action=redirect&bpo=40620]: Convert examples in tutorial controlflow.rst section 4.3 to be interpreter-demo style.
- bpo-43066 [https://bugs.python.org/issue?@action=redirect&bpo=43066]: Added a warning to `zipfile` docs: filename arg with a leading slash may cause archive to be un-openable on Windows systems.
- bpo-39452 [https://bugs.python.org/issue?@action=redirect&bpo=39452]: Rewrote `Doc/library/__main__.rst`. Broadened scope of the document to explicitly discuss and differentiate between `__main__.py` in packages versus the `__name__ == '__main__'` expression (and the idioms that surround it).
- bpo-13814 [https://bugs.python.org/issue?@action=redirect&bpo=13814]: In the Design FAQ, answer «Why don't generators support the with statement?»
- bpo-27752 [https://bugs.python.org/issue?@action=redirect&bpo=27752]: Documentation of csv.Dialect is more descriptive.
- bpo-44453 [https://bugs.python.org/issue?@action=redirect&bpo=44453]: Fix documentation for the return type of `sysconfig.get_path()`.
- bpo-44392 [https://bugs.python.org/issue?@action=redirect&bpo=44392]: Added a new section in the C API documentation for types used in type hinting. Documented `Py_GenericAlias` and `Py_GenericAliasType`.
- bpo-38291 [https://bugs.python.org/issue?@action=redirect&bpo=38291]: Mark `typing.io` and `typing.re` as deprecated since Python 3.8 in the documentation. They were never properly supported by type checkers.
- bpo-44322 [https://bugs.python.org/issue?@action=redirect&bpo=44322]: Document that SyntaxError args

have a details tuple and that details are adjusted for errors in f-string field replacement expressions.

- bpo-42392 [https://bugs.python.org/issue?@action=redirect&bpo=42392]: Document the deprecation and removal of the `loop` parameter for many functions and classes in **`asyncio`**.
- bpo-44195 [https://bugs.python.org/issue?@action=redirect&bpo=44195]: Corrected references to `TraversableResources` in docs. There is no `TraversableReader`.
- bpo-41963 [https://bugs.python.org/issue?@action=redirect&bpo=41963]: Document that `ConfigParser` strips off comments when reading configuration files.
- bpo-44072 [https://bugs.python.org/issue?@action=redirect&bpo=44072]: Correct where in the numeric ABC hierarchy `**` support is added, i.e., in numbers.Complex, not numbers.Integral.
- bpo-43558 [https://bugs.python.org/issue?@action=redirect&bpo=43558]: Add the remark to **`dataclasses`** documentation that the **`__init__()`** of any base class has to be called in **`__post_init__()`**, along with a code example.
- bpo-44025 [https://bugs.python.org/issue?@action=redirect&bpo=44025]: Clarify when "_" in match statements is a keyword, and when not.
- bpo-41706 [https://bugs.python.org/issue?@action=redirect&bpo=41706]: Fix docs about how methods like `__add__` are invoked when evaluating operator expressions.
- bpo-41621 [https://bugs.python.org/issue?@action=redirect&bpo=41621]: Document that **`collections.defaultdict`** parameter `default_factory` defaults to `None` and is positional-only.
- bpo-41576 [https://bugs.python.org/issue?@action=redirect&bpo=41576]: document BaseException in favor of bare except
- bpo-21760 [https://bugs.python.org/issue?@action=redirect&bpo=21760]: The description for _file_ fixed. Patch by Furkan Onder
- bpo-39498 [https://bugs.python.org/issue?@action=redirect&bpo=39498]: Add a «Security Considerations»

index which links to standard library modules that have explicitly documented security considerations.
- bpo-33479 [https://bugs.python.org/issue?@action=redirect&bpo=33479]: Remove the unqualified claim that tkinter is threadsafe. It has not been true for several years and likely never was. An explanation of what is true may be added later, after more discussion, and possibly after patching _tkinter.c,

Tests

- bpo-40173 [https://bugs.python.org/issue?@action=redirect&bpo=40173]: Fix `test.support.import_helper.import_fresh_module()`.

- bpo-45280 [https://bugs.python.org/issue?@action=redirect&bpo=45280]: Add a test case for empty `typing.NamedTuple`.

- bpo-45269 [https://bugs.python.org/issue?@action=redirect&bpo=45269]: Cover case when invalid `markers` type is supplied to `c_make_encoder`.

- bpo-45128 [https://bugs.python.org/issue?@action=redirect&bpo=45128]: Fix `test_multiprocessing_fork` failure due to `test_logging` and `sys.modules` manipulation.

- bpo-45209 [https://bugs.python.org/issue?@action=redirect&bpo=45209]: Fix `UserWarning: resource_tracker` warning in `_test_multiprocessing._TestSharedMemory.test_shared`

- bpo-45185 [https://bugs.python.org/issue?@action=redirect&bpo=45185]: Enables `TestEnumerations` test cases in `test_ssl` suite.

- bpo-45195 [https://bugs.python.org/issue?@action=redirect&bpo=45195]: Fix test_readline.test_nonascii(): sometimes, the newline character is not written at the end, so don’t expect it in the output. Patch by Victor Stinner.

- bpo-45156 [https://bugs.python.org/issue?@action=redirect&bpo=45156]: Fixes infinite loop on `unittest.mock.seal()` of mocks created by `create_autospec()`.

- bpo-45125 [https://bugs.python.org/issue?@action=redirect&bpo=45125]: Improves pickling tests and docs of `SharedMemory` and `SharableList` objects.

- bpo-44860 [https://bugs.python.org/issue?@action=redirect&bpo=44860]: Update `test_sysconfig.test_user_similar()` for the posix_user scheme: `platlib` doesn't use `sys.platlibdir`. Patch by Victor Stinner.

- bpo-45052 [https://bugs.python.org/issue?@action=redirect&bpo=45052]: `WithProcessesTestSharedMemory.test_shared_memory_ba` test was ignored, because `self.assertEqual(sms.size, sms2.size)` line was failing. It is now removed and test is unskipped.

  The main motivation for this line to be removed from the test is that the `size` of `SharedMemory` is not ever guaranteed to be the same. It is decided by the platform.

- bpo-44895 [https://bugs.python.org/issue?@action=redirect&bpo=44895]: libregrtest now clears the type cache later to reduce the risk of false alarm when checking for reference leaks. Previously, the type cache was cleared too early and libregrtest raised a false alarm about reference leaks under very specific conditions. Patch by Irit Katriel and Victor Stinner.

- bpo-45042 [https://bugs.python.org/issue?@action=redirect&bpo=45042]: Fixes that test classes decorated with `@hashlib_helper.requires_hashdigest` were skipped all the time.

- bpo-25130 [https://bugs.python.org/issue?@action=redirect&bpo=25130]: Add calls of `gc.collect()` in tests to support PyPy.

- bpo-45011 [https://bugs.python.org/issue?@action=redirect&bpo=45011]: Made tests relying on the **`_asyncio`** C extension module optional to allow running on alternative Python implementations. Patch by Serhiy Storchaka.
- bpo-44949 [https://bugs.python.org/issue?@action=redirect&bpo=44949]: Fix auto history tests of test_readline: sometimes, the newline character is not written at the end, so don't expect it in the output.
- bpo-44891 [https://bugs.python.org/issue?@action=redirect&bpo=44891]: Tests were added to clarify **`id()`** is preserved when `obj * 1` is used on **`str`** and **`bytes`** objects. Patch by Nikita Sobolev.
- bpo-44852 [https://bugs.python.org/issue?@action=redirect&bpo=44852]: Add ability to wholesale silence DeprecationWarnings while running the regression test suite.
- bpo-40928 [https://bugs.python.org/issue?@action=redirect&bpo=40928]: Notify users running test_decimal regression tests on macOS of potential harmless «malloc can't allocate region» messages spewed by test_decimal.
- bpo-44734 [https://bugs.python.org/issue?@action=redirect&bpo=44734]: Fixed floating-point precision issue in turtle tests.
- bpo-44708 [https://bugs.python.org/issue?@action=redirect&bpo=44708]: Regression tests, when run with -w, are now re-running only the affected test methods instead of re-running the entire test file.
- bpo-42095 [https://bugs.python.org/issue?@action=redirect&bpo=42095]: Added interop tests for Apple plists: generate plist files with Python plistlib and parse with Apple plutil; and the other way round.

- bpo-44647 [https://bugs.python.org/issue?@action=redirect&bpo=44647]: Added a permanent Unicode-valued environment variable to regression tests to ensure they handle this use case in the future. If your test environment breaks because of that, report a bug to us, and temporarily set PYTHONREGRTEST_UNICODE_GUARD=0 in your test environment.

- bpo-44515 [https://bugs.python.org/issue?@action=redirect&bpo=44515]: Adjust recently added contextlib tests to avoid assuming the use of a refcounted GC

- bpo-44287 [https://bugs.python.org/issue?@action=redirect&bpo=44287]: Fix asyncio test_popen() of test_windows_utils by using a longer timeout. Use military grade battle-tested `test.support.SHORT_TIMEOUT` timeout rather than a hardcoded timeout of 10 seconds: it's 30 seconds by default, but it is made longer on slow buildbots. Patch by Victor Stinner.

- bpo-44451 [https://bugs.python.org/issue?@action=redirect&bpo=44451]: Reset `DeprecationWarning` filters in `test.test_importlib.test_metadata_api.APITests.test_` to avoid `StopIteration` error if `DeprecationWarnings` are ignored.

- bpo-44363 [https://bugs.python.org/issue?@action=redirect&bpo=44363]: Account for address sanitizer in test_capi. test_capi now passes when run GCC address sanitizer.

- bpo-44364 [https://bugs.python.org/issue?@action=redirect&bpo=44364]: Add non integral tests for `math.sqrt()` function.

- bpo-43921 [https://bugs.python.org/issue?@action=redirect&bpo=43921]: Fix test_ssl.test_wrong_cert_tls13(): use `suppress_ragged_eofs=False`, since `read()` can raise `ssl.SSLEOFError` on Windows. Patch by Victor Stinner.

- bpo-43921 [https://bugs.python.org/issue?@action=redirect&bpo=43921]: Fix test_pha_required_nocert() of test_ssl: catch two more EOF cases (when the `recv()` method returns an empty string). Patch by Victor Stinner.

- bpo-44131 [https://bugs.python.org/issue?@action=redirect&bpo=44131]: Add test_frozenmain to test_embed to test the **`Py_FrozenMain()`** C function. Patch by Victor Stinner.

- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Ignore error string case in test_file_not_exists().

- bpo-42083 [https://bugs.python.org/issue?@action=redirect&bpo=42083]: Add test to check that `PyStructSequence_NewType` accepts a `PyStructSequence_Desc` with `doc` field set to `NULL`.

- bpo-35753 [https://bugs.python.org/issue?@action=redirect&bpo=35753]: Fix crash in doctest when doctest parses modules that include unwrappable functions by skipping those functions.

- bpo-30256 [https://bugs.python.org/issue?@action=redirect&bpo=30256]: Add test for nested queues when using `multiprocessing` shared objects `AutoProxy[Queue]` inside `ListProxy` and `DictProxy`

Build

- bpo-45220 [https://bugs.python.org/issue?@action=redirect&bpo=45220]: Avoid building with the Windows 11 SDK previews automatically. This may be overridden by setting the `DefaultWindowsSDKVersion` environment variable before building.

- bpo-45020 [https://bugs.python.org/issue?@action=redirect&bpo=45020]: Freeze stdlib modules that are imported during startup. This provides significant performance improvements to startup. If necessary, use the previously added «-X frozen_modules=off» commandline option to force importing the source modules.

- bpo-45188 [https://bugs.python.org/issue?@action=redirect&bpo=45188]: Windows builds now regenerate frozen modules as the first part of the build. Previously the regeneration was later in the build, which would require it to be restarted if any modules had changed.

- bpo-45163 [https://bugs.python.org/issue?@action=redirect&bpo=45163]: Fixes Haiku platform build.

- bpo-45067 [https://bugs.python.org/issue?@action=redirect&bpo=45067]: The ncurses function extended_color_content was introduced in 2017

  (https://invisible-island.net/ncurses/NEWS.html#index-t20170401). The

  ncurses-devel package in CentOS 7 had a older version ncurses resulted in compilation error. For compiling ncurses with extended color support, we verify the version of the ncurses library >= 20170401.

- bpo-45019 [https://bugs.python.org/issue?@action=redirect&bpo=45019]: Generate lines in relevant files for frozen modules. Up until now each of the files had to be edited manually. This change makes it easier to add to and modify the frozen modules.

- bpo-44340 [https://bugs.python.org/issue?@action=redirect&bpo=44340]: Add support for building with clang thin lto via –with-lto=thin/full. Patch by Donghee Na and Brett Holman.

- bpo-44535 [https://bugs.python.org/issue?@action=redirect&bpo=44535]: Enable building using a Visual Studio 2022 install on Windows.

- bpo-43298 [https://bugs.python.org/issue?@action=redirect&bpo=43298]: Improved error message when building without a Windows SDK installed.

- bpo-44381 [https://bugs.python.org/issue?@action=redirect&bpo=44381]: The Windows build now accepts **`EnableControlFlowGuard`** set to `guard` to enable CFG.

- bpo-41282 [https://bugs.python.org/issue?@action=redirect&bpo=41282]: Fix broken `make install` that caused standard library extension modules to be unnecessarily and incorrectly rebuilt during the install phase of cpython.

Windows

- bpo-45375 [https://bugs.python.org/issue?@action=redirect&bpo=45375]: Fixes an assertion failure due to searching for the standard library in unnormalised paths.
- bpo-45022 [https://bugs.python.org/issue?@action=redirect&bpo=45022]: Update Windows release to include libffi 3.4.2
- bpo-45007 [https://bugs.python.org/issue?@action=redirect&bpo=45007]: Update to OpenSSL 1.1.1l in Windows build
- bpo-44848 [https://bugs.python.org/issue?@action=redirect&bpo=44848]: Upgrade Windows installer to use SQLite 3.36.0.
- bpo-44572 [https://bugs.python.org/issue?@action=redirect&bpo=44572]: Avoid consuming standard input in the **`platform`** module
- bpo-44582 [https://bugs.python.org/issue?@action=redirect&bpo=44582]: Accelerate speed of **`mimetypes`** initialization using a native implementation of the registry scan.
- bpo-41299 [https://bugs.python.org/issue?@action=redirect&bpo=41299]: Fix 16 milliseconds jitter when using timeouts in **`threading`**, such as with **`threading.Lock.acquire()`** or **`threading.Condition.wait()`**.
- bpo-42686 [https://bugs.python.org/issue?@action=redirect&bpo=42686]: Build **`sqlite3`** with math

functions enabled. Patch by Erlend E. Aasland.

- bpo-40263 [https://bugs.python.org/issue?@action=redirect&bpo=40263]: This is a follow-on bug from https://bugs.python.org/issue26903. Once that is applied we run into an off-by-one assertion problem. The assert was not correct.

macOS

- bpo-45007 [https://bugs.python.org/issue?@action=redirect&bpo=45007]: Update macOS installer builds to use OpenSSL 1.1.1l.
- bpo-34602 [https://bugs.python.org/issue?@action=redirect&bpo=34602]: When building CPython on macOS with `./configure --with-undefined-behavior-sanitizer --with-pydebug`, the stack size is now quadrupled to allow for the entire test suite to pass.
- bpo-44848 [https://bugs.python.org/issue?@action=redirect&bpo=44848]: Update macOS installer to use SQLite 3.36.0.
- bpo-44689 [https://bugs.python.org/issue?@action=redirect&bpo=44689]: **`ctypes.util.find_library()`** now works correctly on macOS 11 Big Sur even if Python is built on an older version of macOS. Previously, when built on older macOS systems, `find_library` was not able to find macOS system libraries when running on Big Sur due to changes in how system libraries are stored.
- bpo-41972 [https://bugs.python.org/issue?@action=redirect&bpo=41972]: The framework build’s user header path in sysconfig is changed to add a “pythonX.Y” component to match distutils’s behavior.
- bpo-43109 [https://bugs.python.org/issue?@action=redirect&bpo=43109]: Allow –with-lto configure option to work with Apple-supplied Xcode or Command Line Tools.
- bpo-34932 [https://bugs.python.org/issue?@action=redirect&bpo=34932]: Add socket.TCP_KEEPALIVE support for macOS. Patch by Shane Harvey.

IDLE

- bpo-45296 [https://bugs.python.org/issue?@action=redirect&bpo=45296]: On Windows, change exit/quit message to suggest Ctrl-D, which works, instead of <Ctrl-Z Return>, which does not work in IDLE.
- bpo-45193 [https://bugs.python.org/issue?@action=redirect&bpo=45193]: Make completion boxes appear on Ubuntu again.
- bpo-40128 [https://bugs.python.org/issue?@action=redirect&bpo=40128]: Mostly fix completions on macOS when not using tcl/tk 8.6.11 (as with 3.9). The added update_idletask call should be harmless and possibly helpful otherwise.
- bpo-33962 [https://bugs.python.org/issue?@action=redirect&bpo=33962]: Move the indent space setting from the Font tab to the new Windows tab. Patch by Mark Roseman and Terry Jan Reedy.
- bpo-40468 [https://bugs.python.org/issue?@action=redirect&bpo=40468]: Split the settings dialog General tab into Windows and Shell/ED tabs. Move help sources, which extend the Help menu, to the Extensions tab. Make space for new options and shorten the dialog. The latter makes the dialog better fit small screens.
- bpo-41611 [https://bugs.python.org/issue?@action=redirect&bpo=41611]: Avoid uncaught exceptions in `AutoCompleteWindow.winconfig_event()`.
- bpo-41611 [https://bugs.python.org/issue?@action=redirect&bpo=41611]: Fix IDLE sometimes freezing upon tab-completion on macOS.
- bpo-44010 [https://bugs.python.org/issue?@action=redirect&bpo=44010]: Highlight the new match statement's soft keywords: **`match`**, **`case`**, and **`_`**. However, this highlighting is not perfect and will be incorrect in some rare cases, including some `_`-s in `case` patterns.
- bpo-44026 [https://bugs.python.org/issue?@action=redirect&bpo=44026]: Include interpreter's typo fix suggestions in message line for NameErrors and AttributeErrors. Patch by E. Paine.

Tools/Demos

- bpo-44786 [https://bugs.python.org/issue?@action=redirect&bpo=44786]: Fix a warning in regular expression in the c-analyzer script.
- bpo-44967 [https://bugs.python.org/issue?@action=redirect&bpo=44967]: pydoc now returns a non-zero status code when a module cannot be found.
- bpo-44978 [https://bugs.python.org/issue?@action=redirect&bpo=44978]: Allow the Argument Clinic tool to handle `__complex__` special methods.
- bpo-43425 [https://bugs.python.org/issue?@action=redirect&bpo=43425]: Removed the "test2to3" demo project that demonstrated using lib2to3 to support Python 2.x and Python 3.x from a single source in a distutils package. Patch by Donghee Na
- bpo-44074 [https://bugs.python.org/issue?@action=redirect&bpo=44074]: Make patchcheck automatically detect the correct base branch name (previously it was hardcoded to "master")
- bpo-20291 [https://bugs.python.org/issue?@action=redirect&bpo=20291]: Added support for variadic positional parameters in Argument Clinic.

C API

- bpo-41710 [https://bugs.python.org/issue?@action=redirect&bpo=41710]: The PyThread_acquire_lock_timed() function now clamps the timeout if it is too large, rather than aborting the process. Patch by Victor Stinner.

- bpo-44687 [https://bugs.python.org/issue?@action=redirect&bpo=44687]: **`BufferedReader.peek()`** no longer raises **`ValueError`** when the entire file has already been buffered.

- bpo-45116 [https://bugs.python.org/issue?@action=redirect&bpo=45116]: Add the **`Py_ALWAYS_INLINE`** macro to ask the compiler to always inline a static inline function. The compiler can ignore it and decides to not inline the function. Patch by Victor Stinner.

- bpo-45094 [https://bugs.python.org/issue?@action=redirect&bpo=45094]: Add the **`Py_NO_INLINE`** macro to disable inlining on a function. Patch by Victor Stinner.

- bpo-45061 [https://bugs.python.org/issue?@action=redirect&bpo=45061]: Add a deallocator to the **`bool`** type to detect refcount bugs in C extensions which call `Py_DECREF(Py_True);` or `Py_DECREF(Py_False);` by mistake. Patch by Victor Stinner.

- bpo-42035 [https://bugs.python.org/issue?@action=redirect&bpo=42035]: Add a new **`PyType_GetQualName()`** function to get type's qualified name.

- bpo-41103 [https://bugs.python.org/issue?@action=redirect&bpo=41103]: Reverts removal of the old buffer protocol because they are part of stable ABI.

- bpo-44751 [https://bugs.python.org/issue?@action=redirect&bpo=44751]: Remove `crypt.h` include from the public `Python.h` header.

- bpo-42747 [https://bugs.python.org/issue?@action=redirect&bpo=42747]: The `Py_TPFLAGS_HAVE_VERSION_TAG` type flag now does nothing. The `Py_TPFLAGS_HAVE_AM_SEND` flag (which was added in 3.10) is removed. Both were unnecessary because it is not possible to have type objects with the relevant fields missing.

- bpo-44530 [https://bugs.python.org/issue?@action=redirect&bpo=44530]: Added the `co_qualname` to the `PyCodeObject` structure to propagate the qualified name from the compiler to code objects.

  Patch by Gabriele N. Tornetta

- bpo-44441 [https://bugs.python.org/issue?@action=redirect&bpo=44441]: **`Py_RunMain()`** now resets **`PyImport_Inittab`** to its initial value at exit. It must be possible to call **`PyImport_AppendInittab()`** or **`PyImport_ExtendInittab()`** at each Python initialization. Patch by Victor Stinner.

- bpo-39947 [https://bugs.python.org/issue?@action=redirect&bpo=39947]: Remove 4 private trashcan C API functions which were only kept for the backward compatibility of the stable ABI with Python 3.8 and older, since the trashcan API was not usable with the limited C API on Python 3.8 and older. The trashcan API was excluded from the limited C API in Python 3.9.

  Removed functions:

  - ○ _PyTrash_deposit_object()
  - ○ _PyTrash_destroy_chain()
  - ○ _PyTrash_thread_deposit_object()
  - ○ _PyTrash_thread_destroy_chain()

  The trashcan C API was never usable with the limited C API, since old trashcan macros accessed directly **`PyThreadState`** members like `_tstate->trash_delete_nesting`, whereas the **`PyThreadState`** structure is opaque in the limited C API.

  Exclude also the `PyTrash_UNWIND_LEVEL` constant from the C API.

  Patch by Victor Stinner.

- bpo-40939 [https://bugs.python.org/issue?@action=redirect&bpo=40939]: Removed documentation for the removed `PyParser_*` C API.

- bpo-43795 [https://bugs.python.org/issue?@action=redirect&bpo=43795]: The list in Contenido de la API limitada now shows the public name **`PyFrameObject`** rather than `_frame`. The non-existing entry `_node` no longer appears in the list.

- bpo-44378 [https://bugs.python.org/issue?

@action=redirect&bpo=44378]: `Py_IS_TYPE()` no longer uses `Py_TYPE()` to avoid a compiler warning: no longer cast `const PyObject*` to `PyObject*`. Patch by Victor Stinner.

- bpo-39573 [https://bugs.python.org/issue?@action=redirect&bpo=39573]: Convert the `Py_TYPE()` and `Py_SIZE()` macros to static inline functions. The `Py_SET_TYPE()` and `Py_SET_SIZE()` functions must now be used to set an object type and size. Patch by Victor Stinner.

- bpo-44263 [https://bugs.python.org/issue?@action=redirect&bpo=44263]: The `PyType_Ready()` function now raises an error if a type is defined with the `Py_TPFLAGS_HAVE_GC` flag set but has no traverse function (`PyTypeObject.tp_traverse`). Patch by Victor Stinner.

- bpo-43795 [https://bugs.python.org/issue?@action=redirect&bpo=43795]: The undocumented function `Py_FrozenMain()` is removed from the Limited API.

- bpo-44113 [https://bugs.python.org/issue?@action=redirect&bpo=44113]: Deprecate the following functions to configure the Python initialization:

  - ○ `PySys_AddWarnOptionUnicode()`
  - ○ `PySys_AddWarnOption()`
  - ○ `PySys_AddXOption()`
  - ○ `PySys_HasWarnOptions()`
  - ○ `Py_SetPath()`
  - ○ `Py_SetProgramName()`
  - ○ `Py_SetPythonHome()`
  - ○ `Py_SetStandardStreamEncoding()`
  - ○ `_Py_SetProgramFullPath()`

  Use the new `PyConfig` API of the Python Initialization Configuration instead (**PEP 587** [https://peps.python.org/pep-0587/]).

- bpo-44094 [https://bugs.python.org/issue?@action=redirect&bpo=44094]: Remove

`PyErr_SetFromErrnoWithUnicodeFilename()`, `PyErr_SetFromWindowsErrWithUnicodeFilename()`, and `PyErr_SetExcFromWindowsErrWithUnicodeFilename()`. They are not documented and have been deprecated since Python 3.3.

- bpo-43795 [https://bugs.python.org/issue?@action=redirect&bpo=43795]: **`PyCodec_Unregister()`** is now properly exported as a function in the Windows Stable ABI DLL.

- bpo-44029 [https://bugs.python.org/issue?@action=redirect&bpo=44029]: Remove deprecated `Py_UNICODE` APIs: `PyUnicode_Encode`, `PyUnicode_EncodeUTF7`, `PyUnicode_EncodeUTF8`, `PyUnicode_EncodeUTF16`, `PyUnicode_EncodeUTF32`, `PyUnicode_EncodeLatin1`, `PyUnicode_EncodeMBCS`, `PyUnicode_EncodeDecimal`, `PyUnicode_EncodeRawUnicodeEscape`, `PyUnicode_EncodeCharmap`, `PyUnicode_EncodeUnicodeEscape`, `PyUnicode_TransformDecimalToASCII`, `PyUnicode_TranslateCharmap`, `PyUnicodeEncodeError_Create`, `PyUnicodeTranslateError_Create`. See **PEP 393** [https://peps.python.org/pep-0393/] and **PEP 624** [https://peps.python.org/pep-0624/] for reference.

- bpo-42035 [https://bugs.python.org/issue?@action=redirect&bpo=42035]: Add a new **`PyType_GetName()`** function to get type’s short name.

Python 3.10.0 beta 1

Release date: 2021-05-03

Security

- bpo-43434 [https://bugs.python.org/issue?

@action=redirect&bpo=43434]: Creating **`sqlite3.Connection`** objects now also produces `sqlite3.connect` and `sqlite3.connect/handle` auditing events. Previously these events were only produced by **`sqlite3.connect()`** calls. Patch by Erlend E. Aasland.

- bpo-43998 [https://bugs.python.org/issue?@action=redirect&bpo=43998]: The **`ssl`** module sets more secure cipher suites defaults. Ciphers without forward secrecy and with SHA-1 MAC are disabled by default. Security level 2 prohibits weak RSA, DH, and ECC keys with less than 112 bits of security. **`SSLContext`** defaults to minimum protocol version TLS 1.2. Settings are based on Hynek Schlawack's research.
- bpo-43882 [https://bugs.python.org/issue?@action=redirect&bpo=43882]: The presence of newline or tab characters in parts of a URL could allow some forms of attacks.

  Following the controlling specification for URLs defined by WHATWG **`urllib.parse()`** now removes ASCII newlines and tabs from URLs, preventing such attacks.
- bpo-43472 [https://bugs.python.org/issue?@action=redirect&bpo=43472]: Ensures interpreter-level audit hooks receive the `cpython.PyInterpreterState_New` event when called through the `_xxsubinterpreters` module.
- bpo-43362 [https://bugs.python.org/issue?@action=redirect&bpo=43362]: Fix invalid free in _sha3 module. The issue was introduced in 3.10.0a1. Python 3.9 and earlier are not affected.
- bpo-43762 [https://bugs.python.org/issue?@action=redirect&bpo=43762]: Add audit events for **`sqlite3.connect/handle()`**, **`sqlite3.Connection.enable_load_extension()`**, and **`sqlite3.Connection.load_extension()`**. Patch by Erlend E. Aasland.

- bpo-43756 [https://bugs.python.org/issue?@action=redirect&bpo=43756]: Add new audit event `glob.glob/2` to incorporate the new *root_dir* and *dir_fd* arguments added to **`glob.glob()`** and **`glob.iglob()`**.

- bpo-36384 [https://bugs.python.org/issue?@action=redirect&bpo=36384]: **`ipaddress`** module no longer accepts any leading zeros in IPv4 address strings. Leading zeros are ambiguous and interpreted as octal notation by some libraries. For example the legacy function **`socket.inet_aton()`** treats leading zeros as octal notation. glibc implementation of modern **`inet_pton()`** does not accept any leading zeros. For a while the **`ipaddress`** module used to accept ambiguous leading zeros.

- bpo-43075 [https://bugs.python.org/issue?@action=redirect&bpo=43075]: Fix Regular Expression Denial of Service (ReDoS) vulnerability in **`urllib.request.AbstractBasicAuthHandler`**. The ReDoS-vulnerable regex has quadratic worst-case complexity and it allows cause a denial of service when identifying crafted invalid RFCs. This ReDoS issue is on the client side and needs remote attackers to control the HTTP server.

- bpo-42800 [https://bugs.python.org/issue?@action=redirect&bpo=42800]: Audit hooks are now fired for frame.f_code, traceback.tb_frame, and generator code/frame attribute access.

- bpo-37363 [https://bugs.python.org/issue?@action=redirect&bpo=37363]: Add audit events to the **`http.client`** module.

Core and Builtins

- bpo-43977 [https://bugs.python.org/issue?@action=redirect&bpo=43977]: Prevent classes being both a sequence and a mapping when pattern matching.
- bpo-43977 [https://bugs.python.org/issue?@action=redirect&bpo=43977]: Use **`tp_flags`** on the class object to determine if the subject is a sequence or mapping

when pattern matching. Avoids the need to import `collections.abc` when pattern matching.

- bpo-43892 [https://bugs.python.org/issue?@action=redirect&bpo=43892]: Restore proper validation of complex literal value patterns when parsing `match` blocks.
- bpo-43933 [https://bugs.python.org/issue?@action=redirect&bpo=43933]: Set frame.f_lineno to the line number of the “with” kweyword when executing the call to `__exit__`.
- bpo-43933 [https://bugs.python.org/issue?@action=redirect&bpo=43933]: If the current position in a frame has no line number then set the f_lineno attribute to None, instead of -1, to conform to PEP 626. This should not normally be possible, but might occur in some unusual circumstances.
- bpo-43963 [https://bugs.python.org/issue?@action=redirect&bpo=43963]: Importing the `_signal` module in a subinterpreter has no longer side effects.
- bpo-42739 [https://bugs.python.org/issue?@action=redirect&bpo=42739]: The internal representation of line number tables is changed to not use sentinels, and an explicit length parameter is added to the out of process API function `PyLineTable_InitAddressRange`. This makes the handling of line number tables more robust in some circumstances.

Library

- bpo-43908 [https://bugs.python.org/issue?@action=redirect&bpo=43908]: Make `re` types immutable. Patch by Erlend E. Aasland.
- bpo-43908 [https://bugs.python.org/issue?@action=redirect&bpo=43908]: Make the `array.array` type immutable. Patch by Erlend E. Aasland.

Core and Builtins

- bpo-43901 [https://bugs.python.org/issue?@action=redirect&bpo=43901]: Change class and module objects to lazy-create empty annotations dicts on demand. The annotations dicts are stored in the object's _dict_ for backwards compatibility.

- bpo-43892 [https://bugs.python.org/issue?@action=redirect&bpo=43892]: Match patterns now use new dedicated AST nodes (`MatchValue`, `MatchSingleton`, `MatchSequence`, `MatchStar`, `MatchMapping`, `MatchClass`) rather than reusing expression AST nodes. `MatchAs` and `MatchOr` are now defined as pattern nodes rather than as expression nodes. Patch by Nick Coghlan.
- bpo-42725 [https://bugs.python.org/issue?@action=redirect&bpo=42725]: Usage of `await/yield/yield from` and named expressions within an annotation is now forbidden when PEP 563 is activated.
- bpo-43754 [https://bugs.python.org/issue?@action=redirect&bpo=43754]: When performing structural pattern matching (**PEP 634** [https://peps.python.org/pep-0634/]), captured names are now left unbound until the *entire* pattern has matched successfully.
- bpo-42737 [https://bugs.python.org/issue?@action=redirect&bpo=42737]: Annotations for complex targets (everything beside simple names) no longer cause any runtime effects with `from __future__ import annotations`.
- bpo-43914 [https://bugs.python.org/issue?@action=redirect&bpo=43914]: **`SyntaxError`** exceptions raised by the interpreter will highlight the full error range of the expression that constitutes the syntax error itself, instead of just where the problem is detected. Patch by Pablo Galindo.
- bpo-38605 [https://bugs.python.org/issue?@action=redirect&bpo=38605]: Revert making `from __future__ import annotations` the default. This follows the Steering Council decision to postpone PEP 563 changes to at least Python 3.11. See the original email for more information regarding the decision: https://mail.python.org/archives/list/python-dev@python.org/thread/CLVXXPQ2T2LQ5MP2Y53VVQFCXYWQJHKZ/. Patch by Pablo Galindo.
- bpo-43475 [https://bugs.python.org/issue?@action=redirect&bpo=43475]: Hashes of NaN values now depend on object identity. Formerly, they always hashed to 0

even though NaN values are not equal to one another. Having the same hash for unequal values caused pile-ups in hash tables.

- bpo-43859 [https://bugs.python.org/issue?@action=redirect&bpo=43859]: Improve the error message for `IndentationError` exceptions. Patch by Pablo Galindo
- bpo-41323 [https://bugs.python.org/issue?@action=redirect&bpo=41323]: Constant tuple folding in bytecode optimizer now reuses tuple in constant table.
- bpo-43846 [https://bugs.python.org/issue?@action=redirect&bpo=43846]: Data stack usage is much reduced for large literal and call expressions.
- bpo-38530 [https://bugs.python.org/issue?@action=redirect&bpo=38530]: When printing `NameError` raised by the interpreter, `PyErr_Display()` will offer suggestions of similar variable names in the function that the exception was raised from. Patch by Pablo Galindo
- bpo-43823 [https://bugs.python.org/issue?@action=redirect&bpo=43823]: Improve syntax errors for invalid dictionary literals. Patch by Pablo Galindo.
- bpo-43822 [https://bugs.python.org/issue?@action=redirect&bpo=43822]: Improve syntax errors in the parser for missing commas between expressions. Patch by Pablo Galindo.
- bpo-43798 [https://bugs.python.org/issue?@action=redirect&bpo=43798]: `ast.alias` nodes now include source location metadata attributes e.g. lineno, col_offset.
- bpo-43797 [https://bugs.python.org/issue?@action=redirect&bpo=43797]: Improve `SyntaxError` error messages for invalid comparisons. Patch by Pablo Galindo.
- bpo-43760 [https://bugs.python.org/issue?@action=redirect&bpo=43760]: Move the flag for checking whether tracing is enabled to the C stack, from the heap. Should speed up dispatch in the interpreter.
- bpo-43682 [https://bugs.python.org/issue?@action=redirect&bpo=43682]: Static methods (`@staticmethod`) and class methods (`@classmethod`) now inherit the method attributes (`__module__`, `__name__`, `__qualname__`, `__doc__`, `__annotations__`) and have a new `__wrapped__` attribute. Patch by Victor Stinner.

- bpo-43751 [https://bugs.python.org/issue?@action=redirect&bpo=43751]: Fixed a bug where `anext(ait, default)` would erroneously return None.
- bpo-42128 [https://bugs.python.org/issue?@action=redirect&bpo=42128]: **`__match_args__`** is no longer allowed to be a list.
- bpo-43683 [https://bugs.python.org/issue?@action=redirect&bpo=43683]: Add GEN_START opcode. Marks start of generator, including async, or coroutine and handles sending values to a newly created generator or coroutine.

Library

- bpo-43105 [https://bugs.python.org/issue?@action=redirect&bpo=43105]: Importlib now resolves relative paths when creating module spec objects from file locations.

Core and Builtins

- bpo-43682 [https://bugs.python.org/issue?@action=redirect&bpo=43682]: Static methods (**`@staticmethod`**) are now callable as regular functions. Patch by Victor Stinner.
- bpo-42609 [https://bugs.python.org/issue?@action=redirect&bpo=42609]: Prevented crashes in the AST validator and optimizer when compiling some absurdly long expressions like `"+0"*1000000`. **`RecursionError`** is now raised instead.
- bpo-38530 [https://bugs.python.org/issue?@action=redirect&bpo=38530]: When printing **`AttributeError`**, **`PyErr_Display()`** will offer suggestions of similar attribute names in the object that the exception was raised from. Patch by Pablo Galindo

Library

- bpo-44015 [https://bugs.python.org/issue?@action=redirect&bpo=44015]: In @dataclass(), raise a TypeError if KW_ONLY is specified more than once.

- bpo-25478 [https://bugs.python.org/issue?@action=redirect&bpo=25478]: Added a *total()* method to collections.Counter() to compute the sum of the counts.

- bpo-43733 [https://bugs.python.org/issue?@action=redirect&bpo=43733]: Change `netrc.netrc` to use UTF-8 encoding before using locale encoding.

- bpo-43979 [https://bugs.python.org/issue?@action=redirect&bpo=43979]: Removed an unnecessary list comprehension before looping from `urllib.parse.parse_qsl()`. Patch by Christoph Zwerschke and Donghee Na.

- bpo-43993 [https://bugs.python.org/issue?@action=redirect&bpo=43993]: Update bundled pip to 21.1.1.

- bpo-43957 [https://bugs.python.org/issue?@action=redirect&bpo=43957]: [Enum] Deprecate `TypeError` when non-member is used in a containment check; In 3.12 `True` or `False` will be returned instead, and containment will return `True` if the value is either a member of that enum or one of its members" value.

- bpo-42904 [https://bugs.python.org/issue?@action=redirect&bpo=42904]: For backwards compatibility with previous minor versions of Python, if `typing.get_type_hints()` receives no namespace dictionary arguments, `typing.get_type_hints()` will search through the global then local namespaces during evaluation of stringized type annotations (string forward references) inside a class.

- bpo-43945 [https://bugs.python.org/issue?@action=redirect&bpo=43945]: [Enum] Deprecate non-standard mixin format() behavior: in 3.12 the enum member, not the member's value, will be used for format() calls.

- bpo-41139 [https://bugs.python.org/issue?@action=redirect&bpo=41139]: Deprecate undocumented `cgi.log()` API.

- bpo-43937 [https://bugs.python.org/issue?@action=redirect&bpo=43937]: Fixed the `turtle` module working with non-default root window.

- bpo-43930 [https://bugs.python.org/issue?@action=redirect&bpo=43930]: Update bundled pip to 21.1 and setuptools to 56.0.0

- bpo-43907 [https://bugs.python.org/issue?@action=redirect&bpo=43907]: Fix a bug in the pure-Python pickle implementation when using protocol 5, where bytearray instances that occur several time in the pickled object graph would incorrectly unpickle into repeated copies of the bytearray object.

- bpo-43926 [https://bugs.python.org/issue?@action=redirect&bpo=43926]: In `importlib.metadata`, provide a uniform interface to `Description`, allow for any field to be encoded with multiline values, remove continuation lines from multiline values, and add a `.json` property for easy access to the PEP 566 JSON-compatible form. Sync with `importlib_metadata 4.0`.

- bpo-43920 [https://bugs.python.org/issue?@action=redirect&bpo=43920]: OpenSSL 3.0.0: `load_verify_locations()` now returns a consistent error message when cadata contains no valid certificate.

- bpo-43607 [https://bugs.python.org/issue?@action=redirect&bpo=43607]: `urllib` can now convert Windows paths with `\\?\` prefixes into URL paths.

- bpo-43817 [https://bugs.python.org/issue?@action=redirect&bpo=43817]: Add `inspect.get_annotations()`, which safely computes the annotations defined on an object. It works around the quirks of accessing the annotations from various types of objects, and makes very few assumptions about the object passed in. `inspect.get_annotations()` can also correctly un-stringize stringized annotations.

`inspect.signature()`, `inspect.from_callable()`, and `inspect.from_function()` now call `inspect.get_annotations()` to retrieve annotations. This means `inspect.signature()` and `inspect.from_callable()` can now un-stringize stringized annotations, too.

- bpo-43284 [https://bugs.python.org/issue?@action=redirect&bpo=43284]: platform.win32_ver derives the windows version from sys.getwindowsversion().platform_version which in turn derives the version from kernel32.dll (which can be of a different version than Windows itself). Therefore change the platform.win32_ver to determine the version using the platform module's _syscmd_ver private function to return an accurate version.

- bpo-42854 [https://bugs.python.org/issue?@action=redirect&bpo=42854]: The `ssl` module now uses `SSL_read_ex` and `SSL_write_ex` internally. The functions support reading and writing of data larger than 2 GB. Writing zero-length data no longer fails with a protocol violation error.

- bpo-42333 [https://bugs.python.org/issue?@action=redirect&bpo=42333]: Port `_ssl` extension module to multiphase initialization.

- bpo-43880 [https://bugs.python.org/issue?@action=redirect&bpo=43880]: `ssl` now raises DeprecationWarning for OP_NO_SSL/TLS* options, old TLS versions, old protocols, and other features that have been deprecated since Python 3.6, 3.7, or OpenSSL 1.1.0.

- bpo-41559 [https://bugs.python.org/issue?@action=redirect&bpo=41559]: **PEP 612** [https://peps.python.org/pep-0612/] is now implemented purely in Python; builtin `types.GenericAlias` objects no longer include `typing.ParamSpec` in `__parameters__` (with the exception of `collections.abc.Callable`"s `GenericAlias`). This means previously invalid uses of

`ParamSpec` (such as `list[P]`) which worked in earlier versions of Python 3.10 alpha, will now raise `TypeError` during substitution.

- bpo-43867 [https://bugs.python.org/issue?@action=redirect&bpo=43867]: The **`multiprocessing`** `Server` class now explicitly catches **`SystemExit`** and closes the client connection in this case. It happens when the `Server.serve_client()` method reaches the end of file (EOF).

- bpo-40443 [https://bugs.python.org/issue?@action=redirect&bpo=40443]: Remove unused imports: pyclbr no longer uses copy, and typing no longer uses ast. Patch by Victor Stinner.

- bpo-43820 [https://bugs.python.org/issue?@action=redirect&bpo=43820]: Remove an unneeded copy of the namespace passed to dataclasses.make_dataclass().

- bpo-43787 [https://bugs.python.org/issue?@action=redirect&bpo=43787]: Add `__iter__()` method to **`bz2.BZ2File`**, **`gzip.GzipFile`**, and **`lzma.LZMAFile`**. It makes iterating them about 2x faster. Patch by Inada Naoki.

- bpo-43680 [https://bugs.python.org/issue?@action=redirect&bpo=43680]: Deprecate io.OpenWrapper and _pyio.OpenWrapper: use io.open and _pyio.open instead. Until Python 3.9, _pyio.open was not a static method and builtins.open was set to OpenWrapper to not become a bound method when set to a class variable. _io.open is a built-in function whereas _pyio.open is a Python function. In Python 3.10, _pyio.open() is now a static method, and builtins.open() is now io.open().

- bpo-43680 [https://bugs.python.org/issue?@action=redirect&bpo=43680]: The Python **`_pyio.open()`** function becomes a static method to behave as **`io.open()`** built-in function: don’t become a bound method when stored as a class variable. It becomes possible since static methods are now callable in Python 3.10. Moreover,

`_pyio.OpenWrapper()` becomes a simple alias to **`_pyio.open()`**. Patch by Victor Stinner.

- bpo-41515 [https://bugs.python.org/issue?@action=redirect&bpo=41515]: Fix **`KeyError`** raised in **`typing.get_type_hints()`** due to synthetic modules that don't appear in `sys.modules`.

- bpo-43776 [https://bugs.python.org/issue?@action=redirect&bpo=43776]: When **`subprocess.Popen`** args are provided as a string or as **`pathlib.Path`**, the Popen instance repr now shows the right thing.

- bpo-42248 [https://bugs.python.org/issue?@action=redirect&bpo=42248]: [Enum] ensure exceptions raised in `_missing__` are released

- bpo-43744 [https://bugs.python.org/issue?@action=redirect&bpo=43744]: fix issue with enum member name matching the start of a private variable name

- bpo-43772 [https://bugs.python.org/issue?@action=redirect&bpo=43772]: Fixed the return value of `TypeVar.__ror__`. Patch by Jelle Zijlstra.

- bpo-43764 [https://bugs.python.org/issue?@action=redirect&bpo=43764]: Add match_args parameter to @dataclass decorator to allow suppression of _match_args_ generation.

- bpo-43799 [https://bugs.python.org/issue?@action=redirect&bpo=43799]: OpenSSL 3.0.0: define `OPENSSL_API_COMPAT` 1.1.1 to suppress deprecation warnings. Python requires OpenSSL 1.1.1 APIs.

- bpo-43478 [https://bugs.python.org/issue?@action=redirect&bpo=43478]: Mocks can no longer be used as the specs for other Mocks. As a result, an already-mocked object cannot have an attribute mocked using `autospec=True` or be the subject of a `create_autospec(...)` call. This can uncover bugs in tests since these Mock-derived Mocks will always pass certain tests (e.g. `isinstance()`) and builtin assert functions (e.g. assert_called_once_with) will unconditionally pass.

- bpo-43794 [https://bugs.python.org/issue?@action=redirect&bpo=43794]: Add `ssl.OP_IGNORE_UNEXPECTED_EOF` constants (OpenSSL 3.0.0)

- bpo-43785 [https://bugs.python.org/issue?@action=redirect&bpo=43785]: Improve `bz2.BZ2File` performance by removing the RLock from BZ2File. This makes BZ2File thread unsafe in the face of multiple simultaneous readers or writers, just like its equivalent classes in `gzip` and `lzma` have always been. Patch by Inada Naoki.

- bpo-43789 [https://bugs.python.org/issue?@action=redirect&bpo=43789]: OpenSSL 3.0.0: Don't call the password callback function a second time when first call has signaled an error condition.

- bpo-43788 [https://bugs.python.org/issue?@action=redirect&bpo=43788]: The header files for `ssl` error codes are now OpenSSL version-specific. Exceptions will now show correct reason and library codes. The `make_ssl_data.py` script has been rewritten to use OpenSSL's text file with error codes.

- bpo-43766 [https://bugs.python.org/issue?@action=redirect&bpo=43766]: Implement **PEP 647** [https://peps.python.org/pep-0647/] in the `typing` module by adding **`TypeGuard`**.

- bpo-25264 [https://bugs.python.org/issue?@action=redirect&bpo=25264]: `os.path.realpath()` now accepts a *strict* keyword-only argument. When set to `True`, `OSError` is raised if a path doesn't exist or a symlink loop is encountered.

- bpo-43780 [https://bugs.python.org/issue?@action=redirect&bpo=43780]: In `importlib.metadata`, incorporate changes from importlib_metadata 3.10: Add mtime-based caching during distribution discovery. Flagged use of dict result from `entry_points()` as deprecated.

- gh-47383 [https://github.com/python/cpython/issues/47383]: The `P.args` and `P.kwargs` attributes of **`typing.ParamSpec`** are now instances of the new classes **`typing.ParamSpecArgs`** and **`typing.ParamSpecKwargs`**, which enables a more useful `repr()`. Patch by Jelle Zijlstra.

- bpo-43731 [https://bugs.python.org/issue?@action=redirect&bpo=43731]: Add an `encoding` parameter **`logging.fileConfig()`**.

- bpo-43712 [https://bugs.python.org/issue?@action=redirect&bpo=43712]: Add `encoding` and `errors` parameters to **`fileinput.input()`** and **`fileinput.FileInput`**.

- bpo-38659 [https://bugs.python.org/issue?@action=redirect&bpo=38659]: A `simple_enum` decorator is added to the `enum` module to convert a normal class into an Enum. `test_simple_enum` added to test simple enums against a corresponding normal Enum. Standard library modules updated to use `simple_enum`.

- bpo-43764 [https://bugs.python.org/issue?@action=redirect&bpo=43764]: Fix an issue where **`__match_args__`** generation could fail for some **`dataclasses`**.

- bpo-43752 [https://bugs.python.org/issue?@action=redirect&bpo=43752]: Fix **`sqlite3`** regression for zero-sized blobs with converters, where `b""` was returned instead of `None`. The regression was introduced by PR 24723. Patch by Erlend E. Aasland.

- bpo-43655 [https://bugs.python.org/issue?@action=redirect&bpo=43655]: **`tkinter`** dialog windows are now recognized as dialogs by window managers on macOS and X Window.

- bpo-43723 [https://bugs.python.org/issue?@action=redirect&bpo=43723]: The following `threading` methods are now deprecated and should be replaced:

  - `currentThread` => **`threading.current_thread()`**
  - `activeCount` => **`threading.active_count()`**
  - `Condition.notifyAll` => **`threading.Condition.notify_all()`**
  - `Event.isSet` => **`threading.Event.is_set()`**
  - `Thread.setName` => **`threading.Thread.name`**
  - `thread.getName` => **`threading.Thread.name`**
  - `Thread.isDaemon` => **`threading.Thread.daemon`**
  - `Thread.setDaemon` => **`threading.Thread.daemon`**

  Patch by Jelle Zijlstra.

- bpo-2135 [https://bugs.python.org/issue?@action=redirect&bpo=2135]: Deprecate find_module() and find_loader() implementations in importlib and zipimport.

- bpo-43534 [https://bugs.python.org/issue?@action=redirect&bpo=43534]: **`turtle.textinput()`** and **`turtle.numinput()`** create now a transient window working on behalf of the canvas window.

- bpo-43532 [https://bugs.python.org/issue?@action=redirect&bpo=43532]: Add the ability to specify keyword-only fields to dataclasses. These fields will become keyword-only arguments to the generated _init_.

- bpo-43522 [https://bugs.python.org/issue?@action=redirect&bpo=43522]: Fix problem with **`hostname_checks_common_name`**. OpenSSL does not copy hostflags from *struct SSL_CTX* to *struct SSL*.

- bpo-8978 [https://bugs.python.org/issue?@action=redirect&bpo=8978]:

Improve error message for **`tarfile.open()`** when **`lzma`** / **`bz2`** are unavailable. Patch by Anthony Sottile.

- bpo-42967 [https://bugs.python.org/issue?@action=redirect&bpo=42967]: Allow **`bytes`** `separator` argument in `urllib.parse.parse_qs` and `urllib.parse.parse_qsl` when parsing **`str`** query strings. Previously, this raised a `TypeError`.

- bpo-43296 [https://bugs.python.org/issue?@action=redirect&bpo=43296]: Improve **`sqlite3`** error handling: `sqlite3_value_blob()` errors that set `SQLITE_NOMEM` now raise **`MemoryError`**. Patch by Erlend E. Aasland.

- bpo-43312 [https://bugs.python.org/issue?@action=redirect&bpo=43312]: New functions **`sysconfig.get_preferred_scheme()`** and **`sysconfig.get_default_scheme()`** are added to query a platform for its preferred «user», «home», and «prefix» (default) scheme names.

- bpo-43265 [https://bugs.python.org/issue?@action=redirect&bpo=43265]: Improve **`sqlite3.Connection.backup()`** error handling. The error message for non-existent target database names is now `unknown database <database name>` instead of `SQL logic error`. Patch by Erlend E. Aasland.

- bpo-41282 [https://bugs.python.org/issue?@action=redirect&bpo=41282]: Install schemes in `distutils.command.install` are now loaded from **`sysconfig`**.

- bpo-41282 [https://bugs.python.org/issue?@action=redirect&bpo=41282]: `distutils.sysconfig` has been merged to **`sysconfig`**.

- bpo-43176 [https://bugs.python.org/issue?@action=redirect&bpo=43176]: Fixed processing of a dataclass that inherits from a frozen dataclass with no fields. It is now correctly detected as an error.

- bpo-43080 [https://bugs.python.org/issue?@action=redirect&bpo=43080]: `pprint` now has support for `dataclasses.dataclass`. Patch by Lewis Gaul.

- bpo-39950 [https://bugs.python.org/issue?@action=redirect&bpo=39950]: Add `pathlib.Path.hardlink_to()` method that supersedes `link_to()`. The new method has the same argument order as `symlink_to()`.

- bpo-42904 [https://bugs.python.org/issue?@action=redirect&bpo=42904]: `typing.get_type_hints()` now checks the local namespace of a class when evaluating **PEP 563** [https://peps.python.org/pep-0563/] annotations inside said class.

- bpo-42269 [https://bugs.python.org/issue?@action=redirect&bpo=42269]: Add `slots` parameter to `dataclasses.dataclass` decorator to automatically generate `__slots__` for class. Patch provided by Yurii Karabas.

- bpo-39529 [https://bugs.python.org/issue?@action=redirect&bpo=39529]: Deprecated use of `asyncio.get_event_loop()` without running event loop. Emit deprecation warning for `asyncio` functions which implicitly create a `Future` or `Task` objects if there is no running event loop and no explicit *loop* argument is passed: `ensure_future()`, `wrap_future()`, `gather()`, `shield()`, `as_completed()` and constructors of **`Future`**, **`Task`**, `StreamReader`, **`StreamReaderProtocol`**.

- bpo-18369 [https://bugs.python.org/issue?@action=redirect&bpo=18369]: Certificate and PrivateKey classes were added to the ssl module. Certificates and keys can now be loaded from memory buffer, too.

- bpo-41486 [https://bugs.python.org/issue?@action=redirect&bpo=41486]: Use a new output buffer management code for `bz2` / `lzma` / `zlib` modules, and add `.readall()` function to `_compression.DecompressReader` class. These bring some performance improvements. Patch by Ma Lin.

- bpo-31870 [https://bugs.python.org/issue?@action=redirect&bpo=31870]: The `ssl.get_server_certificate()` function now has a *timeout* parameter.

- bpo-41735 [https://bugs.python.org/issue?@action=redirect&bpo=41735]: Fix thread locks in zlib module may go wrong in rare case. Patch by Ma Lin.

- bpo-36470 [https://bugs.python.org/issue?@action=redirect&bpo=36470]: Fix dataclasses with `InitVars` and `replace()`. Patch by Claudiu Popa.

- bpo-40849 [https://bugs.python.org/issue?@action=redirect&bpo=40849]: Expose X509_V_FLAG_PARTIAL_CHAIN ssl flag

- bpo-35114 [https://bugs.python.org/issue?@action=redirect&bpo=35114]: `ssl.RAND_status()` now returns a boolean value (as documented) instead of `1` or `0`.

- bpo-39906 [https://bugs.python.org/issue?@action=redirect&bpo=39906]: `pathlib.Path.stat()` and `chmod()` now accept a *follow_symlinks* keyword-only argument for consistency with corresponding functions in the `os` module.

- bpo-39899 [https://bugs.python.org/issue?@action=redirect&bpo=39899]: `os.path.expanduser()` now refuses to guess Windows home directories if the basename of current user's home directory does not match their username.

  `pathlib.Path.expanduser()` and `home()` now consistently raise `RuntimeError` exception when a home directory cannot be resolved. Previously a `KeyError` exception could be raised on Windows when the

`"USERNAME"` environment variable was unset.

- bpo-36076 [https://bugs.python.org/issue?@action=redirect&bpo=36076]: Added SNI support to `ssl.get_server_certificate()`.

- bpo-38490 [https://bugs.python.org/issue?@action=redirect&bpo=38490]: Covariance, Pearson's correlation, and simple linear regression functionality was added to statistics module. Patch by Tymoteusz Wołodźko.

- bpo-33731 [https://bugs.python.org/issue?@action=redirect&bpo=33731]: Provide a locale.localize() function, which converts a normalized number string into a locale format.

- bpo-32745 [https://bugs.python.org/issue?@action=redirect&bpo=32745]: Fix a regression in the handling of ctypes" `ctypes.c_wchar_p` type: embedded null characters would cause a `ValueError` to be raised. Patch by Zackery Spytz.

Documentation

- bpo-43987 [https://bugs.python.org/issue?@action=redirect&bpo=43987]: Add «Annotations Best Practices» document as a new HOWTO.
- bpo-43977 [https://bugs.python.org/issue?@action=redirect&bpo=43977]: Document the new `Py_TPFLAGS_MAPPING` and `Py_TPFLAGS_SEQUENCE` type flags.
- bpo-43959 [https://bugs.python.org/issue?@action=redirect&bpo=43959]: The documentation on the PyContextVar C-API was clarified.
- bpo-43938 [https://bugs.python.org/issue?@action=redirect&bpo=43938]: Update dataclasses documentation to express that FrozenInstanceError is derived from AttributeError.
- bpo-43778 [https://bugs.python.org/issue?@action=redirect&bpo=43778]: Fix the Sphinx glossary_search extension: create the _static/ sub-directory if it doesn't exist.

- bpo-43755 [https://bugs.python.org/issue?@action=redirect&bpo=43755]: Update documentation to reflect that unparenthesized lambda expressions can no longer be the expression part in an `if` clause in comprehensions and generator expressions since Python 3.9.
- bpo-43739 [https://bugs.python.org/issue?@action=redirect&bpo=43739]: Fixing the example code in Doc/extending/extending.rst to declare and initialize the pmodule variable to be of the right type.

Tests

- bpo-43961 [https://bugs.python.org/issue?@action=redirect&bpo=43961]: Fix test_logging.test_namer_rotator_inheritance() on Windows: use **`os.replace()`** rather than **`os.rename()`**. Patch by Victor Stinner.
- bpo-43842 [https://bugs.python.org/issue?@action=redirect&bpo=43842]: Fix a race condition in the SMTP test of test_logging. Don't close a file descriptor (socket) from a different thread while asyncore.loop() is polling the file descriptor. Patch by Victor Stinner.
- bpo-43843 [https://bugs.python.org/issue?@action=redirect&bpo=43843]: **`test.libregrtest`** now marks a test as ENV_CHANGED (altered the execution environment) if a thread raises an exception but does not catch it. It sets a hook on **`threading.excepthook()`**. Use `--fail-env-changed` option to mark the test as failed. Patch by Victor Stinner.
- bpo-43811 [https://bugs.python.org/issue?@action=redirect&bpo=43811]: Tests multiple OpenSSL versions on GitHub Actions. Use ccache to speed up testing.
- bpo-43791 [https://bugs.python.org/issue?@action=redirect&bpo=43791]: OpenSSL 3.0.0: Disable testing of legacy protocols TLS 1.0 and 1.1. Tests are failing with TLSV1_ALERT_INTERNAL_ERROR.

Build

- bpo-43567 [https://bugs.python.org/issue?

@action=redirect&bpo=43567]: Improved generated code refresh (AST/tokens/opcodes/keywords) on Windows.
- bpo-43669 [https://bugs.python.org/issue?@action=redirect&bpo=43669]: Implement **PEP 644** [https://peps.python.org/pep-0644/]. Python now requires OpenSSL 1.1.1 or newer.

Windows

- bpo-35306 [https://bugs.python.org/issue?@action=redirect&bpo=35306]: Adds additional arguments to `os.startfile()` function.
- bpo-43538 [https://bugs.python.org/issue?@action=redirect&bpo=43538]: Avoid raising errors from `pathlib.Path.exists()` when passed an invalid filename.
- bpo-38822 [https://bugs.python.org/issue?@action=redirect&bpo=38822]: Fixed `os.stat()` failing on inaccessible directories with a trailing slash, rather than falling back to the parent directory's metadata. This implicitly affected `os.path.exists()` and `os.path.isdir()`.
- bpo-26227 [https://bugs.python.org/issue?@action=redirect&bpo=26227]: Fixed decoding of host names in `socket.gethostbyaddr()` and `socket.gethostbyname_ex()`.
- bpo-40432 [https://bugs.python.org/issue?@action=redirect&bpo=40432]: Updated pegen regeneration script on Windows to find and use Python 3.8 or higher. Prior to this, pegen regeneration already required 3.8 or higher, but the script may have used lower versions of Python.
- bpo-43745 [https://bugs.python.org/issue?@action=redirect&bpo=43745]: Actually updates Windows release to OpenSSL 1.1.1k. Earlier releases were mislabelled and actually included 1.1.1i again.
- bpo-43652 [https://bugs.python.org/issue?@action=redirect&bpo=43652]: Update Tcl and Tk to 8.6.11 in Windows installer.
- bpo-43492 [https://bugs.python.org/issue?@action=redirect&bpo=43492]: Upgrade Windows installer to use SQLite 3.35.5.

- bpo-30555 [https://bugs.python.org/issue?@action=redirect&bpo=30555]: Fix `WindowsConsoleIO` errors in the presence of fd redirection. Patch by Segev Finer.

macOS

- bpo-42119 [https://bugs.python.org/issue?@action=redirect&bpo=42119]: Fix check for macOS SDK paths when building Python. Narrow search to match contents of SDKs, namely only files in `/System/Library`, `/System/IOSSupport`, and `/usr` other than `/usr/local`. Previously, anything under `/System` was assumed to be in an SDK which causes problems with the new file system layout in 10.15+ where user file systems may appear to be mounted under `/System`. Paths in `/Library` were also incorrectly treated as SDK locations.
- bpo-43568 [https://bugs.python.org/issue?@action=redirect&bpo=43568]: Drop support for MACOSX_DEPLOYMENT_TARGET < 10.3
- bpo-44009 [https://bugs.python.org/issue?@action=redirect&bpo=44009]: Provide «python3.x-intel64» executable to allow reliably forcing macOS universal2 framework builds to run under Rosetta 2 Intel-64 emulation on Apple Silicon Macs. This can be useful for testing or when universal2 wheels are not yet available.
- bpo-43851 [https://bugs.python.org/issue?@action=redirect&bpo=43851]: Build SQLite with `SQLITE_OMIT_AUTOINIT` on macOS. Patch by Erlend E. Aasland.
- bpo-43492 [https://bugs.python.org/issue?@action=redirect&bpo=43492]: Update macOS installer to use SQLite 3.35.4.
- bpo-42235 [https://bugs.python.org/issue?@action=redirect&bpo=42235]: `Mac/BuildScript/build-installer.py` will now use «–enable-optimizations» and `--with-lto` when building on macOS 10.15 or later.

IDLE

- bpo-37903 [https://bugs.python.org/issue?

@action=redirect&bpo=37903]: Add mouse actions to the shell sidebar. Left click and optional drag selects one or more lines, as with the editor line number sidebar. Right click after selecting raises a context menu with “copy with prompts”. This zips together prompts from the sidebar with lines from the selected text.

- bpo-43981 [https://bugs.python.org/issue?@action=redirect&bpo=43981]: Fix reference leak in test_sidebar and test_squeezer. Patches by Terry Jan Reedy and Pablo Galindo
- bpo-37892 [https://bugs.python.org/issue?@action=redirect&bpo=37892]: Indent IDLE Shell input with spaces instead of tabs
- bpo-43655 [https://bugs.python.org/issue?@action=redirect&bpo=43655]: IDLE dialog windows are now recognized as dialogs by window managers on macOS and X Window.
- bpo-37903 [https://bugs.python.org/issue?@action=redirect&bpo=37903]: IDLE’s shell now shows prompts in a separate side-bar.

C API

- bpo-43916 [https://bugs.python.org/issue?@action=redirect&bpo=43916]: Add a new **`Py_TPFLAGS_DISALLOW_INSTANTIATION`** type flag to disallow creating type instances. Patch by Victor Stinner.
- bpo-43774 [https://bugs.python.org/issue?@action=redirect&bpo=43774]: Remove the now unused `PYMALLOC_DEBUG` macro. Debug hooks on memory allocators are now installed by default if Python is built in debug mode (if `Py_DEBUG` macro is defined). Moreover, they can now be used on Python build in release mode (ex: using `PYTHONMALLOC=debug` environment variable).
- bpo-43962 [https://bugs.python.org/issue?@action=redirect&bpo=43962]: _PyInterpreterState_IDIncref() now calls _PyInterpreterState_IDInitref() and always increments id_refcount. Previously, calling _xxsubinterpreters.get_current() could create an id_refcount inconsistency when a _xxsubinterpreters.InterpreterID object

was deallocated. Patch by Victor Stinner.

- bpo-28254 [https://bugs.python.org/issue?@action=redirect&bpo=28254]: Add new C-API functions to control the state of the garbage collector: **`PyGC_Enable()`**, **`PyGC_Disable()`**, **`PyGC_IsEnabled()`**, corresponding to the functions in the **`gc`** module.
- bpo-43908 [https://bugs.python.org/issue?@action=redirect&bpo=43908]: Introduce **`Py_TPFLAGS_IMMUTABLETYPE`** flag for immutable type objects, and modify **`PyType_Ready()`** to set it for static types. Patch by Erlend E. Aasland.
- bpo-43795 [https://bugs.python.org/issue?@action=redirect&bpo=43795]: **`PyMem_Calloc()`** is now available in the limited C API (`Py_LIMITED_API`).
- bpo-43868 [https://bugs.python.org/issue?@action=redirect&bpo=43868]: **`PyOS_ReadlineFunctionPointer()`** is no longer exported by limited C API headers and by `python3.dll` on Windows. Like any function that takes `FILE*`, it is not part of the stable ABI.
- bpo-43795 [https://bugs.python.org/issue?@action=redirect&bpo=43795]: Stable ABI and limited API definitions are generated from a central manifest (**PEP 652** [https://peps.python.org/pep-0652/]).
- bpo-43753 [https://bugs.python.org/issue?@action=redirect&bpo=43753]: Add the **`Py_Is(x, y)`** function to test if the *x* object is the *y* object, the same as `x is y` in Python. Add also the **`Py_IsNone()`**, **`Py_IsTrue()`**, **`Py_IsFalse()`** functions to test if an object is, respectively, the `None` singleton, the `True` singleton or the `False` singleton. Patch by Victor Stinner.

Python 3.10.0 alpha 7

Release date: 2021-04-05

Security

- bpo-42988 [https://bugs.python.org/issue?

@action=redirect&bpo=42988]: **CVE 2021-3426** [https://www.cve.org/CVERecord?id=CVE-2021-3426]: Remove the `getfile` feature of the **`pydoc`** module which could be abused to read arbitrary files on the disk (directory traversal vulnerability). Moreover, even source code of Python modules can contain sensitive data like passwords. Vulnerability reported by David Schwörer.

- bpo-43285 [https://bugs.python.org/issue?@action=redirect&bpo=43285]: **`ftplib`** no longer trusts the IP address value returned from the server in response to the PASV command by default. This prevents a malicious FTP server from using the response to probe IPv4 address and port combinations on the client network.

  Code that requires the former vulnerable behavior may set a `trust_server_pasv_ipv4_address` attribute on their **`ftplib.FTP`** instances to `True` to re-enable it.

- bpo-43439 [https://bugs.python.org/issue?@action=redirect&bpo=43439]: Add audit hooks for **`gc.get_objects()`**, **`gc.get_referrers()`** and **`gc.get_referents()`**. Patch by Pablo Galindo.

Core and Builtins

- bpo-27129 [https://bugs.python.org/issue?@action=redirect&bpo=27129]: Update CPython bytecode magic number.
- bpo-43672 [https://bugs.python.org/issue?@action=redirect&bpo=43672]: Raise ImportWarning when calling find_loader().
- bpo-43660 [https://bugs.python.org/issue?@action=redirect&bpo=43660]: Fix crash that happens when replacing `sys.stderr` with a callable that can remove the object while an exception is being printed. Patch by Pablo Galindo.
- bpo-27129 [https://bugs.python.org/issue?@action=redirect&bpo=27129]: The bytecode interpreter uses instruction, rather byte, offsets internally. This reduces the

number of EXTENDED_ARG instructions needed and streamlines instruction dispatch a bit.

Library

- bpo-40645 [https://bugs.python.org/issue?@action=redirect&bpo=40645]: Fix reference leak in the **`_hashopenssl`** extension. Patch by Pablo Galindo.

Core and Builtins

- bpo-42134 [https://bugs.python.org/issue?@action=redirect&bpo=42134]: Calls to find_module() by the import system now raise ImportWarning.

- bpo-41064 [https://bugs.python.org/issue?@action=redirect&bpo=41064]: Improve the syntax error for invalid usage of double starred elements ("**") in f-strings. Patch by Pablo Galindo.

- bpo-43575 [https://bugs.python.org/issue?@action=redirect&bpo=43575]: Speed up calls to `map()` by using the **PEP 590** [https://peps.python.org/pep-0590/] `vectorcall` calling convention. Patch by Donghee Na.

- bpo-42137 [https://bugs.python.org/issue?@action=redirect&bpo=42137]: The import system now prefers using `__spec__` for `ModuleType.__repr__` over `module_repr()`.

- bpo-43452 [https://bugs.python.org/issue?@action=redirect&bpo=43452]: Added micro-optimizations to `_PyType_Lookup()` to improve cache lookup performance in the common case of cache hits.

- bpo-43555 [https://bugs.python.org/issue?@action=redirect&bpo=43555]: Report the column offset for **`SyntaxError`** for invalid line continuation characters. Patch by Pablo Galindo.

- bpo-43517 [https://bugs.python.org/issue?

@action=redirect&bpo=43517]: Fix misdetection of circular imports when using `from pkg.mod import attr`, which caused false positives in non-trivial multi-threaded code.

- bpo-43497 [https://bugs.python.org/issue?@action=redirect&bpo=43497]: Emit SyntaxWarnings for assertions with tuple constants, this is a regression introduced in python3.7
- bpo-39316 [https://bugs.python.org/issue?@action=redirect&bpo=39316]: Tracing now has correct line numbers for attribute accesses when the attribute is on a different line from the object. Improves debugging and profiling for multi-line method chains.
- bpo-35883 [https://bugs.python.org/issue?@action=redirect&bpo=35883]: Python no longer fails at startup with a fatal error if a command line argument contains an invalid Unicode character. The **`Py_DecodeLocale()`** function now escapes byte sequences which would be decoded as Unicode characters outside the [U+0000; U+10ffff] range.
- bpo-43410 [https://bugs.python.org/issue?@action=redirect&bpo=43410]: Fix a bug that was causing the parser to crash when emitting syntax errors when reading input from stdin. Patch by Pablo Galindo
- bpo-43406 [https://bugs.python.org/issue?@action=redirect&bpo=43406]: Fix a possible race condition where `PyErr_CheckSignals` tries to execute a non-Python signal handler.
- bpo-42128 [https://bugs.python.org/issue?@action=redirect&bpo=42128]: Add `__match_args__` to struct sequence objects. Patch by Pablo Galindo.
- bpo-43390 [https://bugs.python.org/issue?@action=redirect&bpo=43390]: CPython now sets the `SA_ONSTACK` flag in `PyOS_setsig` for the VM's default signal handlers. This is friendlier to other in-process code that

an extension module or embedding use could pull in (such as Golang's cgo) where tiny thread stacks are the norm and `sigaltstack()` has been used to provide for signal handlers. This is a no-op change for the vast majority of processes that don't use sigaltstack.

- bpo-43287 [https://bugs.python.org/issue?@action=redirect&bpo=43287]: Speed up calls to `filter()` by using the **PEP 590** [https://peps.python.org/pep-0590/] `vectorcall` calling convention. Patch by Donghee Na.
- bpo-37448 [https://bugs.python.org/issue?@action=redirect&bpo=37448]: Add a radix tree based memory map to track in-use obmalloc arenas. Use to replace the old implementation of address_in_range(). The radix tree approach makes it easy to increase pool sizes beyond the OS page size. Boosting the pool and arena size allows obmalloc to handle a significantly higher percentage of requests from its ultra-fast paths.

  It also has the advantage of eliminating the memory unsanitary behavior of the previous address_in_range(). The old address_in_range() was marked with the annotations _Py_NO_SANITIZE_ADDRESS, _Py_NO_SANITIZE_THREAD, and _Py_NO_SANITIZE_MEMORY. Those annotations are no longer needed.

  To disable the radix tree map, set a preprocessor flag as follows: `-DWITH_PYMALLOC_RADIX_TREE=0`.

  Co-authored-by: Tim Peters <tim.peters@gmail.com>
- bpo-29988 [https://bugs.python.org/issue?@action=redirect&bpo=29988]: Only handle asynchronous exceptions and requests to drop the GIL when returning from a call or on the back edges of loops. Makes sure that **`__exit__()`** is always called in with statements, even for interrupts.

Library

- bpo-43720 [https://bugs.python.org/issue?@action=redirect&bpo=43720]: Document various stdlib deprecations in imp, pkgutil, and importlib.util for removal in Python 3.12.

- bpo-43433 [https://bugs.python.org/issue?@action=redirect&bpo=43433]: `xmlrpc.client.ServerProxy` no longer ignores query and fragment in the URL of the server.

- bpo-31956 [https://bugs.python.org/issue?@action=redirect&bpo=31956]: The `index()` method of `array.array` now has optional *start* and *stop* parameters.

- bpo-40066 [https://bugs.python.org/issue?@action=redirect&bpo=40066]: Enum: adjust `repr()` to show only enum and member name (not value, nor angle brackets) and `str()` to show only member name. Update and improve documentation to match.

- bpo-42136 [https://bugs.python.org/issue?@action=redirect&bpo=42136]: Deprecate all module_repr() methods found in importlib as their use is being phased out by Python 3.12.

- bpo-35930 [https://bugs.python.org/issue?@action=redirect&bpo=35930]: Raising an exception raised in a «future» instance will create reference cycles.

- bpo-41369 [https://bugs.python.org/issue?@action=redirect&bpo=41369]: Finish updating the vendored libmpdec to version 2.5.1. Patch by Stefan Krah.

- bpo-43422 [https://bugs.python.org/issue?@action=redirect&bpo=43422]: Revert the _decimal C API which was added in bpo-41324 [https://bugs.python.org/issue?@action=redirect&bpo=41324].

- bpo-43577 [https://bugs.python.org/issue?@action=redirect&bpo=43577]: Fix deadlock when using `ssl.SSLContext` debug callback with `ssl.SSLContext.sni_callback()`.

- bpo-43571 [https://bugs.python.org/issue?@action=redirect&bpo=43571]: It's now possible to create MPTCP sockets with IPPROTO_MPTCP

- bpo-43542 [https://bugs.python.org/issue?@action=redirect&bpo=43542]: `image/heic` and `image/heif` were added to `mimetypes`.

- bpo-40645 [https://bugs.python.org/issue?@action=redirect&bpo=40645]: The `hmac` module now uses OpenSSL's HMAC implementation when digestmod argument is a hash name or builtin hash function.

- bpo-43510 [https://bugs.python.org/issue?@action=redirect&bpo=43510]: Implement **PEP 597** [https://peps.python.org/pep-0597/]: Add `EncodingWarning` warning, `-X warn_default_encoding` option, `PYTHONWARNDEFAULTENCODING` environment variable and `encoding="locale"` argument value.

- bpo-43521 [https://bugs.python.org/issue?@action=redirect&bpo=43521]: `ast.unparse` can now render NaNs and empty sets.

- bpo-42914 [https://bugs.python.org/issue?@action=redirect&bpo=42914]: `pprint.pprint()` gains a new boolean `underscore_numbers` optional argument to emit integers with thousands separated by an underscore character for improved readability (for example `1_000_000` instead of `1000000`).

- bpo-41361 [https://bugs.python.org/issue?@action=redirect&bpo=41361]: `rotate()` calls are now slightly faster due to faster argument parsing.

- bpo-43423 [https://bugs.python.org/issue?@action=redirect&bpo=43423]: `subprocess.communicate()` no longer raises an IndexError when there is an empty stdout or stderr IO buffer during a timeout on Windows.

- bpo-27820 [https://bugs.python.org/issue?@action=redirect&bpo=27820]: Fixed long-standing bug of smtplib.SMTP where doing AUTH LOGIN with initial_response_ok=False will fail.

  The cause is that SMTP.auth_login _always_ returns a password if provided with a challenge string, thus non-compliant with the standard for AUTH LOGIN.

  Also fixes bug with the test for smtpd.

- bpo-43445 [https://bugs.python.org/issue?@action=redirect&bpo=43445]: Add frozen modules to **`sys.stdlib_module_names`**. For example, add `"_frozen_importlib"` and `"_frozen_importlib_external"` names.

- bpo-43245 [https://bugs.python.org/issue?@action=redirect&bpo=43245]: Add keyword arguments support to `ChainMap.new_child()`.

- bpo-29982 [https://bugs.python.org/issue?@action=redirect&bpo=29982]: Add optional parameter *ignore_cleanup_errors* to **`tempfile.TemporaryDirectory()`** and allow multiple **`cleanup()`** attempts. Contributed by C.A.M. Gerlach.

- bpo-43428 [https://bugs.python.org/issue?@action=redirect&bpo=43428]: Include changes from importlib_metadata 3.7 [https://importlib-metadata.readthedocs.io/en/latest/history.html#v3-7-0]:

  Performance enhancements to distribution discovery.

  `entry_points` only returns unique distributions.

  Introduces new `EntryPoints` object for containing a set of entry points with convenience methods for selecting entry points by group or name. `entry_points` now returns this object if selection parameters are supplied but continues to return a dict object for compatibility. Users are encouraged to rely on the selection interface. The dict object result is likely to be deprecated in the future.

Added packages_distributions function to return a mapping of packages to the distributions that provide them.

- bpo-43332 [https://bugs.python.org/issue?@action=redirect&bpo=43332]: Improves the networking efficiency of **`http.client`** when using a proxy via **`set_tunnel()`**. Fewer small send calls are made during connection setup.

- bpo-43420 [https://bugs.python.org/issue?@action=redirect&bpo=43420]: Improve performance of **`fractions.Fraction`** arithmetics for large components. Contributed by Sergey B. Kirpichev.

- bpo-43356 [https://bugs.python.org/issue?@action=redirect&bpo=43356]: Allow passing a signal number to `_thread.interrupt_main()`.

- bpo-43399 [https://bugs.python.org/issue?@action=redirect&bpo=43399]: Fix `ElementTree.extend` not working on iterators when using the Python implementation

- bpo-43369 [https://bugs.python.org/issue?@action=redirect&bpo=43369]: Improve **`sqlite3`** error handling: If `sqlite3_column_text()` and `sqlite3_column_blob()` set `SQLITE_NOMEM`, **`MemoryError`** is now raised. Patch by Erlend E. Aasland.

- bpo-43368 [https://bugs.python.org/issue?@action=redirect&bpo=43368]: Fix a regression introduced in PR 24562, where an empty bytestring was fetched as `None` instead of `b''` in **`sqlite3`**. Patch by Mariusz Felisiak.

- bpo-41282 [https://bugs.python.org/issue?@action=redirect&bpo=41282]: Fixed stacklevel of `DeprecationWarning` emitted from `import distutils`.

- bpo-42129 [https://bugs.python.org/issue?

@action=redirect&bpo=42129]: `importlib.resources` now honors namespace packages, merging resources from each location in the namespace as introduced in `importlib_resources` 3.2 and including incidental changes through 5.0.3.

- bpo-43295 [https://bugs.python.org/issue?@action=redirect&bpo=43295]: **`datetime.datetime.strptime()`** now raises `ValueError` instead of `IndexError` when matching `'z'` with the `%z` format specifier.

- bpo-43125 [https://bugs.python.org/issue?@action=redirect&bpo=43125]: Return empty string if base64mime.body_encode receive empty bytes

- bpo-43084 [https://bugs.python.org/issue?@action=redirect&bpo=43084]: **`curses.window.enclose()`** returns now `True` or `False` (as was documented) instead of `1` or `0`.

- bpo-42994 [https://bugs.python.org/issue?@action=redirect&bpo=42994]: Add MIME types for opus, AAC, 3gpp and 3gpp2

- bpo-14678 [https://bugs.python.org/issue?@action=redirect&bpo=14678]: Add an invalidate_caches() method to the zipimport.zipimporter class to support importlib.invalidate_caches(). Patch by Desmond Cheong.

- bpo-42782 [https://bugs.python.org/issue?@action=redirect&bpo=42782]: Fail fast in **`shutil.move()`** to avoid creating destination directories on failure.

- bpo-40066 [https://bugs.python.org/issue?@action=redirect&bpo=40066]: Enum's `repr()` and `str()` have changed: `repr()` is now *EnumClass.MemberName* and `str()` is *MemberName*. Additionally, stdlib Enum's whose contents are available as module attributes, such as `RegexFlag.IGNORECASE`, have their `repr()` as *module.name*, e.g. `re.IGNORECASE`.

- bpo-26053 [https://bugs.python.org/issue?@action=redirect&bpo=26053]: Fixed bug where the `pdb` interactive run command echoed the args from the shell command line, even if those have been overridden at the pdb prompt.

- bpo-24160 [https://bugs.python.org/issue?@action=redirect&bpo=24160]: Fixed bug where breakpoints did not persist across multiple debugger sessions in `pdb`'s interactive mode.

- bpo-40701 [https://bugs.python.org/issue?@action=redirect&bpo=40701]: When the `tempfile.tempdir` global variable is set to a value of type bytes, it is now handled consistently. Previously exceptions could be raised from some tempfile APIs when the directory did not already exist in this situation. Also ensures that the `tempfile.gettempdir()` and `tempfile.gettempdirb()` functions *always* return `str` and `bytes` respectively.

- bpo-39342 [https://bugs.python.org/issue?@action=redirect&bpo=39342]: Expose `X509_V_FLAG_ALLOW_PROXY_CERTS` as `VERIFY_ALLOW_PROXY_CERTS` to allow proxy certificate validation as explained in https://docs.openssl.org/1.1.1/man7/proxy-certificates/.

- bpo-31861 [https://bugs.python.org/issue?@action=redirect&bpo=31861]: Add builtins.aiter and builtins.anext. Patch by Joshua Bronson (@jab), Daniel Pope (@lordmauve), and Justin Wang (@justin39).

Documentation

- bpo-43199 [https://bugs.python.org/issue?@action=redirect&bpo=43199]: Answer «Why is there no goto?» in the Design and History FAQ.
- bpo-43407 [https://bugs.python.org/issue?@action=redirect&bpo=43407]: Clarified that a result from `time.monotonic()`, `time.perf_counter()`,

`time.process_time()`, or `time.thread_time()` can be compared with the result from any following call to the same function - not just the next immediate call.
- bpo-43354 [https://bugs.python.org/issue?@action=redirect&bpo=43354]: Fix type documentation for `Fault.faultCode`; the type has to be `int` instead of `str`.
- bpo-41933 [https://bugs.python.org/issue?@action=redirect&bpo=41933]: Clarified wording of s * n in the Common Sequence Operations

Tests

- bpo-37945 [https://bugs.python.org/issue?@action=redirect&bpo=37945]: Fix test_getsetlocale_issue1813() of test_locale: skip the test if `setlocale()` fails. Patch by Victor Stinner.
- bpo-41561 [https://bugs.python.org/issue?@action=redirect&bpo=41561]: Add workaround for Ubuntu's custom OpenSSL security level policy.

Build

- bpo-43179 [https://bugs.python.org/issue?@action=redirect&bpo=43179]: Introduce and correctly use ALIGNOF_X in place of SIZEOF_X for alignment-related code in optimized string routines. Patch by Jessica Clarke.
- bpo-43631 [https://bugs.python.org/issue?@action=redirect&bpo=43631]: Update macOS, Windows, and CI to OpenSSL 1.1.1k.
- bpo-43617 [https://bugs.python.org/issue?@action=redirect&bpo=43617]: Improve configure.ac: Check for presence of autoconf-archive package and remove our copies of M4 macros.
- bpo-43466 [https://bugs.python.org/issue?@action=redirect&bpo=43466]: The `configure` script now supports `--with-openssl-rpath` option.
- bpo-43372 [https://bugs.python.org/issue?@action=redirect&bpo=43372]: Use `_freeze_importlib` to generate code for the `__hello__` module. This approach ensures the code matches the interpreter version. Previously,

PYTHON_FOR_REGEN was used to generate the code, which might be wrong. The marshal format for code objects has changed with bpo-42246 [https://bugs.python.org/issue?@action=redirect&bpo=42246], commit 877df851. Update the code and the expected code sizes in ctypes test_frozentable.

Windows

- bpo-43440 [https://bugs.python.org/issue?@action=redirect&bpo=43440]: Build **`sqlite3`** with the `R*Tree` module enabled. Patch by Erlend E. Aasland.

IDLE

- bpo-42225 [https://bugs.python.org/issue?@action=redirect&bpo=42225]: Document that IDLE can fail on Unix either from misconfigured IP masquerade rules or failure displaying complex colored (non-ascii) characters.

C API

- bpo-43688 [https://bugs.python.org/issue?@action=redirect&bpo=43688]: The limited C API is now supported if Python is built in debug mode (if the `Py_DEBUG` macro is defined). In the limited C API, the **`Py_INCREF()`** and **`Py_DECREF()`** functions are now implemented as opaque function calls, rather than accessing directly the **`PyObject.ob_refcnt`** member, if Python is built in debug mode and the `Py_LIMITED_API` macro targets Python 3.10 or newer. It became possible to support the limited C API in debug mode because the **`PyObject`** structure is the same in release and debug mode since Python 3.8 (see bpo-36465 [https://bugs.python.org/issue?@action=redirect&bpo=36465]).

  The limited C API is still not supported in the `--with-trace-refs` special build (`Py_TRACE_REFS` macro).

  Patch by Victor Stinner.

- bpo-43244 [https://bugs.python.org/issue?

@action=redirect&bpo=43244]: Remove the `pyarena.h` header file with functions:

  - `PyArena_New()`
  - `PyArena_Free()`
  - `PyArena_Malloc()`
  - `PyArena_AddPyObject()`

  These functions were undocumented, excluded from the limited C API, and were only used internally by the compiler. Patch by Victor Stinner.

- bpo-43244 [https://bugs.python.org/issue?@action=redirect&bpo=43244]: Remove the compiler and parser functions using `struct _mod` type, because the public AST C API was removed:

  - `PyAST_Compile()`
  - `PyAST_CompileEx()`
  - `PyAST_CompileObject()`
  - `PyFuture_FromAST()`
  - `PyFuture_FromASTObject()`
  - `PyParser_ASTFromFile()`
  - `PyParser_ASTFromFileObject()`
  - `PyParser_ASTFromFilename()`
  - `PyParser_ASTFromString()`
  - `PyParser_ASTFromStringObject()`

  These functions were undocumented and excluded from the limited C API. Patch by Victor Stinner.

- bpo-43244 [https://bugs.python.org/issue?@action=redirect&bpo=43244]: Remove `ast.h`, `asdl.h`, and `Python-ast.h` header files. These functions were undocumented and excluded from the limited C API. Most names defined by these header files were not prefixed by `Py` and so could create names conflicts. For example, `Python-ast.h` defined a `Yield` macro which was conflict with the `Yield` name used by the Windows `<winbase.h>` header. Use the Python **`ast`** module instead. Patch by Victor Stinner.

- bpo-43541 [https://bugs.python.org/issue?@action=redirect&bpo=43541]: Fix a `PyEval_EvalCodeEx()` regression: fix reference counting on builtins. Patch by Victor Stinner.
- bpo-43244 [https://bugs.python.org/issue?@action=redirect&bpo=43244]: Remove the `symtable.h` header file and the undocumented functions:

  - ○ `PyST_GetScope()`
  - ○ `PySymtable_Build()`
  - ○ `PySymtable_BuildObject()`
  - ○ `PySymtable_Free()`
  - ○ `Py_SymtableString()`
  - ○ `Py_SymtableStringObject()`

  The `Py_SymtableString()` function was part the stable ABI by mistake but it could not be used, because the `symtable.h` header file was excluded from the limited C API.

  The Python **`symtable`** module remains available and is unchanged.

  Patch by Victor Stinner.

- bpo-43244 [https://bugs.python.org/issue?@action=redirect&bpo=43244]: Remove the `PyAST_Validate()` function. It is no longer possible to build a AST object (`mod_ty` type) with the public C API. The function was already excluded from the limited C API (**PEP 384** [https://peps.python.org/pep-0384/]). Patch by Victor Stinner.

Python 3.10.0 alpha 6

Release date: 2021-03-01

Security

- bpo-42967 [https://bugs.python.org/issue?

@action=redirect&bpo=42967]: Fix web cache poisoning vulnerability by defaulting the query args separator to `&`, and allowing the user to choose a custom separator.

Core and Builtins

- bpo-43321 [https://bugs.python.org/issue?@action=redirect&bpo=43321]: Fix `SystemError` raised when `PyArg_Parse*()` is used with `#` but without `PY_SSIZE_T_CLEAN` defined.
- bpo-36346 [https://bugs.python.org/issue?@action=redirect&bpo=36346]: `PyArg_Parse*()` functions now emits `DeprecationWarning` when `u` or `Z` format is used. See **PEP 623** [https://peps.python.org/pep-0623/] for detail.
- bpo-43277 [https://bugs.python.org/issue?@action=redirect&bpo=43277]: Add a new **`PySet_CheckExact()`** function to the C-API to check if an object is an instance of **`set`** but not an instance of a subtype. Patch by Pablo Galindo.
- bpo-42990 [https://bugs.python.org/issue?@action=redirect&bpo=42990]: The **`types.FunctionType`** constructor now inherits the current builtins if the *globals* dictionary has no `"__builtins__"` key, rather than using `{"None": None}` as builtins: same behavior as **`eval()`** and **`exec()`** functions. Defining a function with `def function(...): ...` in Python is not affected, globals cannot be overridden with this syntax: it also inherits the current builtins. Patch by Victor Stinner.
- bpo-42990 [https://bugs.python.org/issue?@action=redirect&bpo=42990]: Functions have a new `__builtins__` attribute which is used to look for builtin symbols when a function is executed, instead of looking into `__globals__['__builtins__']`. Patch by Mark Shannon and Victor Stinner.
- bpo-43149 [https://bugs.python.org/issue?@action=redirect&bpo=43149]: Improve the error message in the parser for exception groups without parentheses. Patch by Pablo Galindo.
- bpo-43121 [https://bugs.python.org/issue?@action=redirect&bpo=43121]: Fixed an incorrect **`SyntaxError`**

message for missing comma in literals. Patch by Pablo Galindo.

Library

- bpo-42819 [https://bugs.python.org/issue?@action=redirect&bpo=42819]: **`readline`**: Explicitly disable bracketed paste in the interactive interpreter, even if it's set in the inputrc, is enabled by default (eg GNU Readline 8.1), or a user calls `readline.read_init_file()`. The Python REPL has not implemented bracketed paste support. Also, bracketed mode writes the `"\x1b[?2004h"` escape sequence into stdout which causes test failures in applications that don't support it. It can still be explicitly enabled by calling `readline.parse_and_bind("set enable-bracketed-paste on")`. Patch by Dustin Rodrigues.

Core and Builtins

- bpo-42808 [https://bugs.python.org/issue?@action=redirect&bpo=42808]: Simple calls to `type(object)` are now faster due to the `vectorcall` calling convention. Patch by Dennis Sweeney.
- bpo-42217 [https://bugs.python.org/issue?@action=redirect&bpo=42217]: Make the compiler merges same co_code and co_linetable objects in a module like already did for co_consts.
- bpo-41972 [https://bugs.python.org/issue?@action=redirect&bpo=41972]: Substring search functions such as `str1 in str2` and `str2.find(str1)` now sometimes use the «Two-Way» string comparison algorithm to avoid quadratic behavior on long strings.
- bpo-42128 [https://bugs.python.org/issue?@action=redirect&bpo=42128]: Implement **PEP 634** [https://peps.python.org/pep-0634/] (structural pattern matching). Patch by Brandt Bucher.

Library

- bpo-40692 [https://bugs.python.org/issue?

@action=redirect&bpo=40692]: In the `concurrent.futures.ProcessPoolExecutor`, validate that `multiprocess.synchronize()` is available on a given platform and rely on that check in the `concurrent.futures` test suite so we can run tests that are unrelated to `ProcessPoolExecutor` on those platforms.

Core and Builtins

- bpo-38302 [https://bugs.python.org/issue?@action=redirect&bpo=38302]: If `object.__ipow__()` returns `NotImplemented`, the operator will correctly fall back to `object.__pow__()` and `object.__rpow__()` as expected.

Library

- bpo-43316 [https://bugs.python.org/issue?@action=redirect&bpo=43316]: The `python -m gzip` command line application now properly fails when detecting an unsupported extension. It exits with a non-zero exit code and prints an error message to stderr.
- bpo-43317 [https://bugs.python.org/issue?@action=redirect&bpo=43317]: Set the chunk size for the `gzip` module main function to io.DEFAULT_BUFFER_SIZE. This is slightly faster than the 1024 bytes constant that was used previously.
- bpo-43146 [https://bugs.python.org/issue?@action=redirect&bpo=43146]: Handle None in single-arg versions of `print_exception()` and `format_exception()`.
- bpo-43260 [https://bugs.python.org/issue?@action=redirect&bpo=43260]: Fix TextIOWrapper can not flush internal buffer forever after very large text is written.
- bpo-43258 [https://bugs.python.org/issue?@action=redirect&bpo=43258]: Prevent needless allocation of `sqlite3` aggregate function context when no rows match an aggregate query. Patch by Erlend E. Aasland.
- bpo-43251 [https://bugs.python.org/issue?@action=redirect&bpo=43251]: Improve `sqlite3` error

handling: `sqlite3_column_name()` failures now result in **`MemoryError`**. Patch by Erlend E. Aasland.

- bpo-40956 [https://bugs.python.org/issue?@action=redirect&bpo=40956]: Fix segfault in **`sqlite3.Connection.backup()`** if no argument was provided. The regression was introduced by PR 23838. Patch by Erlend E. Aasland.
- bpo-43172 [https://bugs.python.org/issue?@action=redirect&bpo=43172]: The readline module now passes its tests when built directly against libedit. Existing irreconcilable API differences remain in **`readline.get_begidx()`** and **`readline.get_endidx()`** behavior based on libreadline vs libedit use.
- bpo-43163 [https://bugs.python.org/issue?@action=redirect&bpo=43163]: Fix a bug in **`codeop`** that was causing it to not ask for more input when multi-line snippets have unclosed parentheses. Patch by Pablo Galindo
- bpo-43162 [https://bugs.python.org/issue?@action=redirect&bpo=43162]: deprecate unsupported ability to access enum members as attributes of other enum members
- bpo-43146 [https://bugs.python.org/issue?@action=redirect&bpo=43146]: Fix recent regression in None argument handling in **`traceback`** module functions.
- bpo-43102 [https://bugs.python.org/issue?@action=redirect&bpo=43102]: The namedtuple __new__ method had its __builtins__ set to None instead of an actual dictionary. This created problems for introspection tools.
- bpo-43106 [https://bugs.python.org/issue?@action=redirect&bpo=43106]: Added **`O_EVTONLY`**, **`O_FSYNC`**, **`O_SYMLINK`** and **`O_NOFOLLOW_ANY`** for macOS. Patch by Donghee Na.
- bpo-42960 [https://bugs.python.org/issue?@action=redirect&bpo=42960]: Adds **`resource.RLIMIT_KQUEUES`** constant from FreeBSD to the **`resource`** module.
- bpo-42151 [https://bugs.python.org/issue?@action=redirect&bpo=42151]: Make the pure Python implementation of **`xml.etree.ElementTree`** behave the same as the C implementation (**`_elementree`**) regarding

default attribute values (by not setting `specified_attributes=1`).

- bpo-29753 [https://bugs.python.org/issue?@action=redirect&bpo=29753]: In ctypes, now packed bitfields are calculated properly and the first item of packed bitfields is now shrank correctly.

Documentation

- bpo-27646 [https://bugs.python.org/issue?@action=redirect&bpo=27646]: Clarify that "yield from <expr>" works with any iterable, not just iterators.
- bpo-36346 [https://bugs.python.org/issue?@action=redirect&bpo=36346]: Update some deprecated unicode APIs which are documented as «will be removed in 4.0» to «3.12». See **PEP 623** [https://peps.python.org/pep-0623/] for detail.

Tests

- bpo-43288 [https://bugs.python.org/issue?@action=redirect&bpo=43288]: Fix test_importlib to correctly skip Unicode file tests if the filesystem does not support them.

Build

- bpo-43174 [https://bugs.python.org/issue?@action=redirect&bpo=43174]: Windows build now uses `/utf-8` compiler option.

- bpo-43103 [https://bugs.python.org/issue?@action=redirect&bpo=43103]: Add a new configure `--without-static-libpython` option to not build the `libpythonMAJOR.MINOR.a` static library and not install the `python.o` object file.

- bpo-13501 [https://bugs.python.org/issue?@action=redirect&bpo=13501]: The configure script can now use *libedit* instead of *readline* with the command line option `--with-readline=editline`.

- bpo-42603 [https://bugs.python.org/issue?@action=redirect&bpo=42603]: Make configure script use pkg-config to detect the location of Tcl/Tk headers and libraries, used to build tkinter.

  On macOS, a Tcl/Tk configuration provided by pkg-config will be preferred over Tcl/Tk frameworks installed in `/{System/,}Library/Frameworks`. If both exist and the latter is preferred, the appropriate `--with-tcltk-*` configuration options need to be explicitly set.

- bpo-39448 [https://bugs.python.org/issue?@action=redirect&bpo=39448]: Add the «regen-frozen» makefile target that regenerates the code for the frozen `__hello__` module.

Windows

- bpo-43155 [https://bugs.python.org/issue?@action=redirect&bpo=43155]: `PyCMethod_New()` is now present in `python3.lib`.

macOS

- bpo-41837 [https://bugs.python.org/issue?@action=redirect&bpo=41837]: Update macOS installer build to use OpenSSL 1.1.1j.

IDLE

- bpo-43283 [https://bugs.python.org/issue?@action=redirect&bpo=43283]: Document why printing to IDLE's Shell is often slower than printing to a system terminal and that it can be made faster by pre-formatting a single string before printing.

C API

- bpo-43278 [https://bugs.python.org/issue?@action=redirect&bpo=43278]: Always put compiler and system information on the first line of the REPL welcome message.

- bpo-43270 [https://bugs.python.org/issue?@action=redirect&bpo=43270]: Remove the private `_PyErr_OCCURRED()` macro: use the public **`PyErr_Occurred()`** function instead.
- bpo-35134 [https://bugs.python.org/issue?@action=redirect&bpo=35134]: Move odictobject.h, parser_interface.h, picklebufobject.h, pydebug.h, and pyfpe.h into the cpython/ directory. They must not be included directly, as they are already included by Python.h: Include Files.
- bpo-35134 [https://bugs.python.org/issue?@action=redirect&bpo=35134]: Move pyarena.h, pyctype.h, and pytime.h into the cpython/ directory. They must not be included directly, as they are already included by Python.h: Include Files.
- bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170]: **`PyExceptionClass_Name()`** is now always declared as a function, in order to hide implementation details. The macro accessed **`PyTypeObject.tp_name`** directly. Patch by Erlend E. Aasland.
- bpo-43239 [https://bugs.python.org/issue?@action=redirect&bpo=43239]: The **`PyCFunction_New()`** function is now exported in the ABI when compiled with `-fvisibility=hidden`.
- bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170]: **`PyIter_Check()`** is now always declared as a function, in order to hide implementation details. The macro accessed **`PyTypeObject.tp_iternext`** directly. Patch by Erlend E. Aasland.
- bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170]: Convert **`PyDescr_IsData()`** macro to a function to hide implementation details: The macro accessed **`PyTypeObject.tp_descr_set`** directly. Patch by Erlend E. Aasland.
- bpo-43181 [https://bugs.python.org/issue?@action=redirect&bpo=43181]: Convert **`PyObject_TypeCheck()`** macro to a static inline function. Patch by Erlend E. Aasland.

Python 3.10.0 alpha 5

Release date: 2021-02-02

Security

- bpo-42938 [https://bugs.python.org/issue?@action=redirect&bpo=42938]: Avoid static buffers when computing the repr of **`ctypes.c_double`** and **`ctypes.c_longdouble`** values.

Core and Builtins

- bpo-42990 [https://bugs.python.org/issue?@action=redirect&bpo=42990]: Refactor the `PyEval_` family of functions.
    - An new function `_PyEval_Vector` is added to simplify calls to Python from C.
    - `_PyEval_EvalCodeWithName` is removed
    - `PyEval_EvalCodeEx` is retained as part of the API, but is not used internally
- bpo-38631 [https://bugs.python.org/issue?@action=redirect&bpo=38631]: Replace **`Py_FatalError()`** calls in the compiler with regular **`SystemError`** exceptions. Patch by Victor Stinner.
- bpo-42997 [https://bugs.python.org/issue?@action=redirect&bpo=42997]: Improve error message for missing «:» before blocks. Patch by Pablo Galindo.
- bpo-43017 [https://bugs.python.org/issue?@action=redirect&bpo=43017]: Improve error message in the parser when using un-parenthesised tuples in comprehensions. Patch by Pablo Galindo.
- bpo-42986 [https://bugs.python.org/issue?@action=redirect&bpo=42986]: Fix parser crash when reporting syntax errors in f-string with newlines. Patch by Pablo Galindo.
- bpo-40176 [https://bugs.python.org/issue?@action=redirect&bpo=40176]: Syntax errors for unterminated string literals now point to the start of the string instead of

reporting EOF/EOL.

- bpo-42927 [https://bugs.python.org/issue?@action=redirect&bpo=42927]: The inline cache for `LOAD_ATTR` now also optimizes access to attributes defined by `__slots__`. This makes reading such attribute up to 30% faster.
- bpo-42864 [https://bugs.python.org/issue?@action=redirect&bpo=42864]: Improve error messages in the parser when parentheses are not closed. Patch by Pablo Galindo.
- bpo-42924 [https://bugs.python.org/issue?@action=redirect&bpo=42924]: Fix `bytearray` repetition incorrectly copying data from the start of the buffer, even if the data is offset within the buffer (e.g. after reassigning a slice at the start of the `bytearray` to a shorter byte string).
- bpo-42882 [https://bugs.python.org/issue?@action=redirect&bpo=42882]: Fix the **`_PyUnicode_FromId()`** function (_Py_IDENTIFIER(var) API) when **`Py_Initialize()`** / **`Py_Finalize()`** is called multiple times: preserve `_PyRuntime.unicode_ids.next_index` value.
- bpo-42827 [https://bugs.python.org/issue?@action=redirect&bpo=42827]: Fix a crash when working out the error line of a **`SyntaxError`** in some multi-line expressions.
- bpo-42823 [https://bugs.python.org/issue?@action=redirect&bpo=42823]: frame.f_lineno is correct even if frame.f_trace is set to True

Library

- bpo-37324 [https://bugs.python.org/issue?@action=redirect&bpo=37324]: Remove deprecated aliases to Colecciones clases base abstractas from the **`collections`** module.

Core and Builtins

- bpo-41994 [https://bugs.python.org/issue?@action=redirect&bpo=41994]: Fixed possible leak in `import` when `sys.modules` is not a `dict`.

- bpo-27772 [https://bugs.python.org/issue?@action=redirect&bpo=27772]: In string formatting, preceding the *width* field by `'0'` no longer affects the default alignment for strings.

Library

- bpo-43108 [https://bugs.python.org/issue?@action=redirect&bpo=43108]: Fixed a reference leak in the **`curses`** module. Patch by Pablo Galindo
- bpo-43077 [https://bugs.python.org/issue?@action=redirect&bpo=43077]: Update the bundled pip to 21.0.1 and setuptools to 52.0.0.
- bpo-41282 [https://bugs.python.org/issue?@action=redirect&bpo=41282]: Deprecate `distutils` in documentation and add warning on import.
- bpo-43014 [https://bugs.python.org/issue?@action=redirect&bpo=43014]: Improve performance of **`tokenize`** by 20-30%. Patch by Anthony Sottile.
- bpo-42323 [https://bugs.python.org/issue?@action=redirect&bpo=42323]: Fix **`math.nextafter()`** for NaN on AIX.
- bpo-42955 [https://bugs.python.org/issue?@action=redirect&bpo=42955]: Add **`sys.stdlib_module_names`**, containing the list of the standard library module names. Patch by Victor Stinner.
- bpo-42944 [https://bugs.python.org/issue?@action=redirect&bpo=42944]: Fix `random.Random.sample` when `counts` argument is not `None`.
- bpo-42934 [https://bugs.python.org/issue?@action=redirect&bpo=42934]: Use **`TracebackException`**'s new `compact` param in **`TestResult`** to reduce time and memory consumed by traceback formatting.
- bpo-42931 [https://bugs.python.org/issue?@action=redirect&bpo=42931]: Add **`randbytes()`** to `random.__all__`.
- bpo-38250 [https://bugs.python.org/issue?@action=redirect&bpo=38250]: [Enum] Flags consisting of a single bit are now considered canonical, and will be the only flags returned from listing and iterating over a Flag class or a

Flag member. Multi-bit flags are considered aliases; they will be returned from lookups and operations that result in their value. Iteration for both Flag and Flag members is in definition order.

- bpo-42877 [https://bugs.python.org/issue?@action=redirect&bpo=42877]: Added the `compact` parameter to the constructor of **`traceback.TracebackException`** to reduce time and memory for use cases that only need to call **`TracebackException.format()`** and **`TracebackException.format_exception_only()`**.
- bpo-42923 [https://bugs.python.org/issue?@action=redirect&bpo=42923]: The **`Py_FatalError()`** function and the **`faulthandler`** module now dump the list of extension modules on a fatal error.
- bpo-42848 [https://bugs.python.org/issue?@action=redirect&bpo=42848]: Removed recursion from **`TracebackException`** to allow it to handle long exception chains.
- bpo-42901 [https://bugs.python.org/issue?@action=redirect&bpo=42901]: [Enum] move member creation from `EnumMeta.__new__` to `_proto_member.__set_name__`, allowing members to be created and visible in `__init_subclass__`.
- bpo-42780 [https://bugs.python.org/issue?@action=redirect&bpo=42780]: Fix os.set_inheritable() for O_PATH file descriptors on Linux.
- bpo-42866 [https://bugs.python.org/issue?@action=redirect&bpo=42866]: Fix a reference leak in the `getcodec()` function of CJK codecs. Patch by Victor Stinner.
- bpo-42846 [https://bugs.python.org/issue?@action=redirect&bpo=42846]: Convert the 6 CJK codec extension modules (_codecs_cn, _codecs_hk, _codecs_iso2022, _codecs_jp, _codecs_kr and _codecs_tw) to the multiphase initialization API (**PEP 489** [https://peps.python.org/pep-0489/]). Patch by Victor Stinner.
- bpo-42851 [https://bugs.python.org/issue?@action=redirect&bpo=42851]: remove _init_subclass_ support for Enum members
- bpo-42834 [https://bugs.python.org/issue?@action=redirect&bpo=42834]: Make internal caches of the

`_json` module compatible with subinterpreters.

- bpo-41748 [https://bugs.python.org/issue?@action=redirect&bpo=41748]: Fix HTMLParser parsing rules for element attributes containing commas with spaces. Patch by Karl Dubost.
- bpo-40810 [https://bugs.python.org/issue?@action=redirect&bpo=40810]: Require SQLite 3.7.15 or newer. Patch by Erlend E. Aasland.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Convert the _multibytecodec extension module (CJK codecs) to multi-phase initialization (**PEP 489** [https://peps.python.org/pep-0489/]). Patch by Erlend E. Aasland.
- bpo-42802 [https://bugs.python.org/issue?@action=redirect&bpo=42802]: The distutils `bdist_wininst` command deprecated in Python 3.8 has been removed. The distutils `bdist_wheel` command is now recommended to distribute binary packages on Windows.
- bpo-24464 [https://bugs.python.org/issue?@action=redirect&bpo=24464]: The undocumented built-in function `sqlite3.enable_shared_cache` is now deprecated, scheduled for removal in Python 3.12. Its use is strongly discouraged by the SQLite3 documentation. Patch by Erlend E. Aasland.
- bpo-42384 [https://bugs.python.org/issue?@action=redirect&bpo=42384]: Make pdb populate sys.path[0] exactly the same as regular python execution.
- bpo-42383 [https://bugs.python.org/issue?@action=redirect&bpo=42383]: Fix pdb: previously pdb would fail to restart the debugging target if it was specified using a relative path and the current directory changed.
- bpo-42005 [https://bugs.python.org/issue?@action=redirect&bpo=42005]: Fix CLI of **`cProfile`** and **`profile`** to catch **`BrokenPipeError`**.
- bpo-41604 [https://bugs.python.org/issue?@action=redirect&bpo=41604]: Don’t decrement the reference count of the previous user_ptr when set_panel_userptr fails.
- bpo-41149 [https://bugs.python.org/issue?@action=redirect&bpo=41149]: Allow executing callables that have a boolean value of `False` when passed to

`Threading.thread` as the target. Patch contributed by Barney Stratford.

- bpo-38307 [https://bugs.python.org/issue?@action=redirect&bpo=38307]: Add an “end_lineno” attribute to the Class and Function objects that appear in the tree returned by pyclbr functions. This and the existing “lineno” attribute define the extent of class and def statements. Patch by Aviral Srivastava.
- bpo-39273 [https://bugs.python.org/issue?@action=redirect&bpo=39273]: The `BUTTON5_*` constants are now exposed in the **`curses`** module if available.
- bpo-33289 [https://bugs.python.org/issue?@action=redirect&bpo=33289]: Correct call to **`tkinter.colorchooser`** to return RGB triplet of ints instead of floats. Patch by Cheryl Sabella.

Documentation

- bpo-40304 [https://bugs.python.org/issue?@action=redirect&bpo=40304]: Fix doc for type(name, bases, dict). Patch by Boris Verkhovskiy and Éric Araujo.
- bpo-42811 [https://bugs.python.org/issue?@action=redirect&bpo=42811]: Updated importlib.util.resolve_name() doc to use __spec__.parent instead of __package__. (Thanks Yair Frid.)

Tests

- bpo-40823 [https://bugs.python.org/issue?@action=redirect&bpo=40823]: Use **`unittest.TestLoader().loadTestsFromTestCase()`** instead of **`unittest.makeSuite()`** in **`sqlite3`** tests. Patch by Erlend E. Aasland.
- bpo-40810 [https://bugs.python.org/issue?@action=redirect&bpo=40810]: In **`sqlite3`**, fix `CheckTraceCallbackContent` for SQLite pre 3.7.15.

Build

- bpo-43031 [https://bugs.python.org/issue?

@action=redirect&bpo=43031]: Pass `--timeout=$(TESTTIMEOUT)` option to the default profile task `./python -m test --pgo` command.

- bpo-36143 [https://bugs.python.org/issue?@action=redirect&bpo=36143]: `make regen-all` now also runs `regen-keyword`. Patch by Victor Stinner.

- bpo-42874 [https://bugs.python.org/issue?@action=redirect&bpo=42874]: Removed the grep -q and -E flags in the tzpath validation section of the configure script to better accommodate users of some platforms (specifically Solaris 10).

- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Add library search path by wr-cc in add_cross_compiling_paths() for VxWorks.

- bpo-42856 [https://bugs.python.org/issue?@action=redirect&bpo=42856]: Add `--with-wheel-pkg-dir=PATH` option to the `./configure` script. If specified, the **`ensurepip`** module looks for `setuptools` and `pip` wheel packages in this directory: if both are present, these wheel packages are used instead of ensurepip bundled wheel packages.

  Some Linux distribution packaging policies recommend against bundling dependencies. For example, Fedora installs wheel packages in the `/usr/share/python-wheels/` directory and don't install the `ensurepip._bundled` package.

Windows

- bpo-41837 [https://bugs.python.org/issue?@action=redirect&bpo=41837]: Updated Windows installer to include OpenSSL 1.1.1i
- bpo-42584 [https://bugs.python.org/issue?@action=redirect&bpo=42584]: Upgrade Windows installer to use SQLite 3.34.0.

macOS

- bpo-42504 [https://bugs.python.org/issue?@action=redirect&bpo=42504]: Ensure that the value of sysconfig.get_config_var("MACOSX_DEPLOYMENT_TARGET") is always a string, even in when the value is parsable as an integer.

IDLE

- bpo-43008 [https://bugs.python.org/issue?@action=redirect&bpo=43008]: Make IDLE invoke **`sys.excepthook()`** in normal, 2-process mode. Patch by Ken Hilton.
- bpo-33065 [https://bugs.python.org/issue?@action=redirect&bpo=33065]: Fix problem debugging user classes with _repr_ method.
- bpo-23544 [https://bugs.python.org/issue?@action=redirect&bpo=23544]: Disable Debug=>Stack Viewer when user code is running or Debugger is active, to prevent hang or crash. Patch by Zackery Spytz.
- bpo-32631 [https://bugs.python.org/issue?@action=redirect&bpo=32631]: Finish zzdummy example extension module: make menu entries work; add docstrings and tests with 100% coverage.

C API

- bpo-42979 [https://bugs.python.org/issue?@action=redirect&bpo=42979]: When Python is built in debug mode (with C assertions), calling a type slot like `sq_length` (`__len__()` in Python) now fails with a fatal error if the slot succeeded with an exception set, or failed with no exception set. The error message contains the slot, the type name, and the current exception (if an exception is set). Patch by Victor Stinner.
- bpo-43030 [https://bugs.python.org/issue?@action=redirect&bpo=43030]: Fixed a compiler warning in **`Py_UNICODE_ISSPACE()`** on platforms with signed **`wchar_t`**.

Python 3.10.0 alpha 4

Release date: 2021-01-04

Core and Builtins

- bpo-42814 [https://bugs.python.org/issue?@action=redirect&bpo=42814]: Fix undefined behavior in `Objects/genericaliasobject.c`.
- bpo-42806 [https://bugs.python.org/issue?@action=redirect&bpo=42806]: Fix the column offsets for f-strings **`ast`** nodes surrounded by parentheses and for nodes that spawn multiple lines. Patch by Pablo Galindo.
- bpo-40631 [https://bugs.python.org/issue?@action=redirect&bpo=40631]: Fix regression where a single parenthesized starred expression was a valid assignment target.
- bpo-27794 [https://bugs.python.org/issue?@action=redirect&bpo=27794]: Improve the error message for failed writes/deletes to property objects. When possible, the attribute name is now shown. Patch provided by Yurii Karabas.
- bpo-42745 [https://bugs.python.org/issue?@action=redirect&bpo=42745]: Make the type attribute lookup cache per-interpreter. Patch by Victor Stinner.
- bpo-42246 [https://bugs.python.org/issue?@action=redirect&bpo=42246]: Jumps to jumps are not eliminated when it would break PEP 626.
- bpo-42246 [https://bugs.python.org/issue?@action=redirect&bpo=42246]: Make sure that the `f_lasti` and `f_lineno` attributes of a frame are set correctly when an exception is raised or re-raised. Required for PEP 626.
- bpo-32381 [https://bugs.python.org/issue?@action=redirect&bpo=32381]: The coding cookie (ex: `# coding: latin1`) is now ignored in the command passed to the **`-c`** command line option. Patch by Victor Stinner.
- bpo-30858 [https://bugs.python.org/issue?@action=redirect&bpo=30858]: Improve error location in expressions that contain assignments. Patch by Pablo Galindo

and Lysandros Nikolaou.
- bpo-42615 [https://bugs.python.org/issue?@action=redirect&bpo=42615]: Remove jump commands made redundant by the deletion of unreachable bytecode blocks

Library

- bpo-42639 [https://bugs.python.org/issue?@action=redirect&bpo=42639]: Make the **`atexit`** module state per-interpreter. It is now safe have more than one **`atexit`** module instance. Patch by Donghee Na and Victor Stinner.

Core and Builtins

- bpo-32381 [https://bugs.python.org/issue?@action=redirect&bpo=32381]: Fix encoding name when running a `.pyc` file on Windows: **`PyRun_SimpleFileExFlags()`** now uses the correct encoding to decode the filename.

Library

- bpo-42195 [https://bugs.python.org/issue?@action=redirect&bpo=42195]: The `__args__` of the parameterized generics for **`typing.Callable`** and **`collections.abc.Callable`** are now consistent. The `__args__` for **`collections.abc.Callable`** are now flattened while **`typing.Callable`**’s have not changed. To allow this change, **`types.GenericAlias`** can now be subclassed and `collections.abc.Callable`’s `__class_getitem__` will now return a subclass of `types.GenericAlias`. Tests for typing were also updated to not subclass things like `Callable[..., T]` as that is not a valid base class. Finally, both `Callable`s no longer validate their `argtypes`, in `Callable[[argtypes], resulttype]` to prepare for **PEP 612** [https://peps.python.org/pep-0612/]. Patch by Ken Jin.
- bpo-40137 [https://bugs.python.org/issue?@action=redirect&bpo=40137]: Convert functools module to use **`PyType_FromModuleAndSpec()`**.
- bpo-40077 [https://bugs.python.org/issue?

@action=redirect&bpo=40077]: Convert `array` to use heap types, and establish module state for these.
- bpo-42008 [https://bugs.python.org/issue?@action=redirect&bpo=42008]: Fix _random.Random() seeding.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the `pyexpat` extension module to multi-phase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).

Core and Builtins

- bpo-40521 [https://bugs.python.org/issue?@action=redirect&bpo=40521]: Make the Unicode dictionary of interned strings compatible with subinterpreters. Patch by Victor Stinner.
- bpo-39465 [https://bugs.python.org/issue?@action=redirect&bpo=39465]: Make `_PyUnicode_FromId()` function compatible with subinterpreters. Each interpreter now has an array of identifier objects (interned strings decoded from UTF-8). Patch by Victor Stinner.

Library

- bpo-42257 [https://bugs.python.org/issue?@action=redirect&bpo=42257]: Handle empty string in variable executable in platform.libc_ver()

- bpo-42772 [https://bugs.python.org/issue?@action=redirect&bpo=42772]: randrange() now raises a TypeError when step is specified without a stop argument. Formerly, it silently ignored the step argument.

- bpo-42759 [https://bugs.python.org/issue?@action=redirect&bpo=42759]: Fixed equality comparison of `tkinter.Variable` and `tkinter.font.Font`. Objects which belong to different Tcl interpreters are now always different, even if they have the same name.

- bpo-42756 [https://bugs.python.org/issue?@action=redirect&bpo=42756]: Configure LMTP Unix-domain socket to use socket global default timeout when a timeout is not explicitly provided.

- bpo-23328 [https://bugs.python.org/issue?@action=redirect&bpo=23328]: Allow / character in username, password fields on _PROXY envars.

- bpo-42740 [https://bugs.python.org/issue?@action=redirect&bpo=42740]: `typing.get_args()` and `typing.get_origin()` now support **PEP 604** [https://peps.python.org/pep-0604/] union types and **PEP 612** [https://peps.python.org/pep-0612/] additions to `Callable`.

- bpo-42655 [https://bugs.python.org/issue?@action=redirect&bpo=42655]: `subprocess` *extra_groups* is now correctly passed into setgroups() system call.

- bpo-42727 [https://bugs.python.org/issue?@action=redirect&bpo=42727]: `EnumMeta.__prepare__` now accepts `**kwds` to properly support `__init_subclass__`

- bpo-38308 [https://bugs.python.org/issue?@action=redirect&bpo=38308]: Add optional *weights* to *statistics.harmonic_mean()*.

- bpo-42721 [https://bugs.python.org/issue?@action=redirect&bpo=42721]: When simple query dialogs (`tkinter.simpledialog`), message boxes (`tkinter.messagebox`) or color choose dialog (`tkinter.colorchooser`) are created without arguments *master* and *parent*, and the default root window is not yet created, and **`NoDefaultRoot()`** was not called, a new temporal hidden root window will be created automatically. It will not be set as the default root window and will be destroyed right after closing the dialog window. It will help to use these simple dialog windows in programs which do not need other GUI.

- bpo-25246 [https://bugs.python.org/issue?@action=redirect&bpo=25246]: Optimized `collections.deque.remove()`.

- bpo-35728 [https://bugs.python.org/issue?@action=redirect&bpo=35728]: Added a root parameter to `tkinter.font.nametofont()`.

- bpo-15303 [https://bugs.python.org/issue?@action=redirect&bpo=15303]: `tkinter` supports now widgets with boolean value False.

- bpo-42681 [https://bugs.python.org/issue?@action=redirect&bpo=42681]: Fixed range checks for color and pair numbers in `curses`.

- bpo-42685 [https://bugs.python.org/issue?@action=redirect&bpo=42685]: Improved placing of simple query windows in Tkinter (such as `tkinter.simpledialog.askinteger()`). They are now centered at the center of the parent window if it is specified and shown, otherwise at the center of the screen.

- bpo-9694 [https://bugs.python.org/issue?@action=redirect&bpo=9694]: Argparse help no longer uses the confusing phrase, «optional arguments». It uses «options» instead.

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the `_thread` extension module to the multiphase initialization API (**PEP 489** [https://peps.python.org/pep-0489/]) and convert its static types to heap types.

- bpo-37961 [https://bugs.python.org/issue?@action=redirect&bpo=37961]: Fix crash in `tracemalloc.Traceback.__repr__()` (regressed in Python 3.9).

- bpo-42630 [https://bugs.python.org/issue?@action=redirect&bpo=42630]: `tkinter` functions and constructors which need a default root window raise now `RuntimeError` with descriptive message instead of obscure `AttributeError` or `NameError` if it is not created yet or cannot be created automatically.

- bpo-42639 [https://bugs.python.org/issue?@action=redirect&bpo=42639]: **`atexit._run_exitfuncs()`** now logs callback exceptions using **`sys.unraisablehook`**, rather than logging them directly into **`sys.stderr`** and raise the last exception.

- bpo-42644 [https://bugs.python.org/issue?@action=redirect&bpo=42644]: `logging.disable` will now validate the types and value of its parameter. It also now accepts strings representing the levels (as does `logging.setLevel`) instead of only the numerical values.

- bpo-42639 [https://bugs.python.org/issue?@action=redirect&bpo=42639]: At Python exit, if a callback registered with **`atexit.register()`** fails, its exception is now logged. Previously, only some exceptions were logged, and the last exception was always silently ignored.

- bpo-36541 [https://bugs.python.org/issue?@action=redirect&bpo=36541]: Fixed lib2to3.pgen2 to be able to parse PEP-570 positional only argument syntax.

- bpo-42382 [https://bugs.python.org/issue?@action=redirect&bpo=42382]: In `importlib.metadata`: - `EntryPoint` objects now expose a `.dist` object referencing the `Distribution` when constructed from a `Distribution`. - Add support for package discovery under package normalization rules. - The object returned by `metadata()` now has a formally defined protocol called `PackageMetadata` with declared support for the `.get_all()` method. - Synced with importlib_metadata 3.3.

- bpo-41877 [https://bugs.python.org/issue?@action=redirect&bpo=41877]: A check is added against misspellings of autospect, auto_spec and set_spec being passed as arguments to patch, patch.object and create_autospec.

- bpo-39717 [https://bugs.python.org/issue?@action=redirect&bpo=39717]: [tarfile] update nested exception raising to use `from None` or `from e`

- bpo-41877 [https://bugs.python.org/issue?@action=redirect&bpo=41877]: AttributeError for suspected misspellings of assertions on mocks are now pointing out that the cause are misspelled assertions and also what to do if the misspelling is actually an intended attribute name. The unittest.mock document is also updated to reflect the current set of recognised misspellings.

- bpo-41559 [https://bugs.python.org/issue?@action=redirect&bpo=41559]: Implemented **PEP 612** [https://peps.python.org/pep-0612/]: added `ParamSpec` and `Concatenate` to **`typing`**. Patch by Ken Jin.

- bpo-42385 [https://bugs.python.org/issue?@action=redirect&bpo=42385]: StrEnum: fix _generate_next_value_ to return a str

- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Define THREAD_STACK_SIZE for VxWorks.

- bpo-34750 [https://bugs.python.org/issue?@action=redirect&bpo=34750]: [Enum] `_EnumDict.update()` is now supported.

- bpo-42517 [https://bugs.python.org/issue?@action=redirect&bpo=42517]: Enum: private names do not become members / do not generate errors – they remain normal attributes

- bpo-42678 [https://bugs.python.org/issue?@action=redirect&bpo=42678]: `Enum`: call `__init_subclass__` after members have been added

- bpo-28964 [https://bugs.python.org/issue?@action=redirect&bpo=28964]: **`ast.literal_eval()`** adds line number information (if available) in error message for malformed nodes.

- bpo-42470 [https://bugs.python.org/issue?@action=redirect&bpo=42470]: **`random.sample()`** no longer warns on a sequence which is also a set.

- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: **`posixpath.expanduser()`** returns the input *path* unchanged if user home directory is None on VxWorks.
- bpo-42388 [https://bugs.python.org/issue?@action=redirect&bpo=42388]: Fix subprocess.check_output(..., input=None) behavior when text=True to be consistent with that of the documentation and universal_newlines=True.
- bpo-34463 [https://bugs.python.org/issue?@action=redirect&bpo=34463]: Fixed discrepancy between **`traceback`** and the interpreter in formatting of SyntaxError with lineno not set (**`traceback`** was changed to match interpreter).
- bpo-42393 [https://bugs.python.org/issue?@action=redirect&bpo=42393]: Raise **`OverflowError`** instead of silent truncation in **`socket.ntohs()`** and **`socket.htons()`**. Silent truncation was deprecated in Python 3.7. Patch by Erlend E. Aasland
- bpo-42222 [https://bugs.python.org/issue?@action=redirect&bpo=42222]: Harmonized **`random.randrange()`** argument handling to match **`range()`**.
  - The integer test and conversion in `randrange()` now uses **`operator.index()`**.
  - Non-integer arguments to `randrange()` are deprecated.
  - The `ValueError` is deprecated in favor of a `TypeError`.
  - It now runs a little faster than before.

  (Contributed by Raymond Hettinger and Serhiy Storchaka.)
- bpo-42163 [https://bugs.python.org/issue?@action=redirect&bpo=42163]: Restore compatibility for

`uname_result` around deepcopy and _replace.

- bpo-42090 [https://bugs.python.org/issue?@action=redirect&bpo=42090]: `zipfile.Path.joinpath` now accepts arbitrary arguments, same as `pathlib.Path.joinpath`.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the _csv module to the multi-phase initialization API (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-42059 [https://bugs.python.org/issue?@action=redirect&bpo=42059]: **`typing.TypedDict`** types created using the alternative call-style syntax now correctly respect the `total` keyword argument when setting their `__required_keys__` and `__optional_keys__` class attributes.
- bpo-41960 [https://bugs.python.org/issue?@action=redirect&bpo=41960]: Add `globalns` and `localns` parameters to the **`inspect.signature()`** and **`inspect.Signature.from_callable()`**.
- bpo-41907 [https://bugs.python.org/issue?@action=redirect&bpo=41907]: fix `format()` behavior for `IntFlag`
- bpo-41891 [https://bugs.python.org/issue?@action=redirect&bpo=41891]: Ensure asyncio.wait_for waits for task completion
- bpo-24792 [https://bugs.python.org/issue?@action=redirect&bpo=24792]: Fixed bug where **`zipimporter`** sometimes reports an incorrect cause of import errors.
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Fix site and sysconfig modules for VxWorks RTOS which has no home directories.
- bpo-41462 [https://bugs.python.org/issue?

@action=redirect&bpo=41462]: Add `os.set_blocking()` support for VxWorks RTOS.

- bpo-40219 [https://bugs.python.org/issue?@action=redirect&bpo=40219]: Lowered `tkinter.ttk.LabeledScale` dummy widget to prevent hiding part of the content label.

- bpo-37193 [https://bugs.python.org/issue?@action=redirect&bpo=37193]: Fixed memory leak in `socketserver.ThreadingMixIn` introduced in Python 3.7.

- bpo-39068 [https://bugs.python.org/issue?@action=redirect&bpo=39068]: Fix initialization race condition in `a85encode()` and `b85encode()` in `base64`. Patch by Brandon Stansbury.

Documentation

- bpo-17140 [https://bugs.python.org/issue?@action=redirect&bpo=17140]: Add documentation for the `multiprocessing.pool.ThreadPool` class.
- bpo-34398 [https://bugs.python.org/issue?@action=redirect&bpo=34398]: Prominently feature listings from the glossary in documentation search results. Patch by Ammar Askar.

Tests

- bpo-42794 [https://bugs.python.org/issue?@action=redirect&bpo=42794]: Update test_nntplib to use official group name of news.aioe.org for testing. Patch by Donghee Na.
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Skip some asyncio tests on VxWorks.
- bpo-42641 [https://bugs.python.org/issue?@action=redirect&bpo=42641]: Enhance `test_select.test_select()`: it now takes 500

milliseconds rather than 10 seconds. Use Python rather than a shell to make the test more portable.

- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Skip some tests in _test_all_chown_common() on VxWorks.
- bpo-42199 [https://bugs.python.org/issue?@action=redirect&bpo=42199]: Fix bytecode helper assertNotInBytecode.
- bpo-41443 [https://bugs.python.org/issue?@action=redirect&bpo=41443]: Add more attribute checking in test_posix.py
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Disable os.popen and impacted tests on VxWorks
- bpo-41439 [https://bugs.python.org/issue?@action=redirect&bpo=41439]: Port test_ssl and test_uuid to VxWorks RTOS.

Build

- bpo-42692 [https://bugs.python.org/issue?@action=redirect&bpo=42692]: Fix __builtin_available check on older compilers. Patch by Joshua Root.
- bpo-27640 [https://bugs.python.org/issue?@action=redirect&bpo=27640]: Added `--disable-test-modules` option to the `configure` script: don’t build nor install test modules. Patch by Xavier de Gaye, Thomas Petazzoni and Peixing Xin.
- bpo-42604 [https://bugs.python.org/issue?@action=redirect&bpo=42604]: Now all platforms use a value for the «EXT_SUFFIX» build variable derived from SOABI (for instance in freeBSD, «EXT_SUFFIX» is now «.cpython-310d.so» instead of «.so»). Previously only Linux, Mac and VxWorks were using a value for «EXT_SUFFIX» that included «SOABI».
- bpo-42598 [https://bugs.python.org/issue?@action=redirect&bpo=42598]: Fix implicit function declarations in configure which could have resulted in incorrect configuration checks. Patch contributed by Joshua Root.
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Enable libpython3.so for

VxWorks.

- bpo-29076 [https://bugs.python.org/issue?@action=redirect&bpo=29076]: Add fish shell support to macOS installer.

macOS

- bpo-42361 [https://bugs.python.org/issue?@action=redirect&bpo=42361]: Update macOS installer build to use Tcl/Tk 8.6.11 (rc2, expected to be final release).
- bpo-41837 [https://bugs.python.org/issue?@action=redirect&bpo=41837]: Update macOS installer build to use OpenSSL 1.1.1i.
- bpo-42584 [https://bugs.python.org/issue?@action=redirect&bpo=42584]: Update macOS installer to use SQLite 3.34.0.

Tools/Demos

- bpo-42726 [https://bugs.python.org/issue?@action=redirect&bpo=42726]: Fixed Python 3 compatibility issue with gdb/libpython.py handling of attribute dictionaries.
- bpo-42613 [https://bugs.python.org/issue?@action=redirect&bpo=42613]: Fix `freeze.py` tool to use the prope config and library directories. Patch by Victor Stinner.

C API

- bpo-42591 [https://bugs.python.org/issue?@action=redirect&bpo=42591]: Export the **`Py_FrozenMain()`** function: fix a Python 3.9.0 regression. Python 3.9 uses `-fvisibility=hidden` and the function was not exported explicitly and so not exported.
- bpo-32381 [https://bugs.python.org/issue?@action=redirect&bpo=32381]: Remove the private **`_Py_fopen()`** function which is no longer needed. Use **`_Py_wfopen()`** or **`_Py_fopen_obj()`** instead. Patch by Victor Stinner.
- bpo-1635741 [https://bugs.python.org/issue?

@action=redirect&bpo=1635741]: Port **`resource`** extension module to module state

- bpo-42111 [https://bugs.python.org/issue?@action=redirect&bpo=42111]: Update the `xxlimited` module to be a better example of how to use the limited C API.
- bpo-40052 [https://bugs.python.org/issue?@action=redirect&bpo=40052]: Fix an alignment build warning/ error in function `PyVectorcall_Function()`. Patch by Andreas Schneider, Antoine Pitrou and Petr Viktorin.

Python 3.10.0 alpha 3

Release date: 2020-12-07

Security

- bpo-40791 [https://bugs.python.org/issue?@action=redirect&bpo=40791]: Add `volatile` to the accumulator variable in `hmac.compare_digest`, making constant-time-defeating optimizations less likely.

Core and Builtins

- bpo-42576 [https://bugs.python.org/issue?@action=redirect&bpo=42576]: `types.GenericAlias` will now raise a `TypeError` when attempting to initialize with a keyword argument. Previously, this would cause the interpreter to crash if the interpreter was compiled with debug symbols. This does not affect interpreters compiled for release. Patch by Ken Jin.

- bpo-42536 [https://bugs.python.org/issue?@action=redirect&bpo=42536]: Several built-in and standard library types now ensure that their internal result tuples are always tracked by the garbage collector:

  - **`collections.OrderedDict.items()`**
  - **`dict.items()`**
  - **`enumerate()`**

  - ○ `functools.reduce()`
  - ○ `itertools.combinations()`
  - ○ `itertools.combinations_with_replacement()`
  - ○ `itertools.permutations()`
  - ○ `itertools.product()`
  - ○ `itertools.zip_longest()`
  - ○ `zip()`

  Previously, they could have become untracked by a prior garbage collection. Patch by Brandt Bucher.

- bpo-42500 [https://bugs.python.org/issue?@action=redirect&bpo=42500]: Improve handling of exceptions near recursion limit. Converts a number of Fatal Errors in RecursionErrors.

- bpo-42246 [https://bugs.python.org/issue?@action=redirect&bpo=42246]: PEP 626: After a return, the f_lineno attribute of a frame is always the last line executed.

- bpo-42435 [https://bugs.python.org/issue?@action=redirect&bpo=42435]: Speed up comparison of bytes objects with non-bytes objects when option `-b` is specified. Speed up comparison of bytarray objects with non-buffer object.

Library

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the `_warnings` extension module to the multi-phase initialization API (**PEP 489** [https://peps.python.org/pep-0489/]). Patch by Victor Stinner.

Core and Builtins

- bpo-41686 [https://bugs.python.org/issue?@action=redirect&bpo=41686]: On Windows, the `SIGINT` event, `_PyOS_SigintEvent()`, is now created even if Python is configured to not install signal handlers (if `PyConfig.install_signal_handlers` equals to 0, or

`Py_InitializeEx(0)`).

- bpo-42381 [https://bugs.python.org/issue?@action=redirect&bpo=42381]: Allow assignment expressions in set literals and set comprehensions as per PEP 572. Patch by Pablo Galindo.
- bpo-42202 [https://bugs.python.org/issue?@action=redirect&bpo=42202]: Change function parameters annotations internal representation to tuple of strings. Patch provided by Yurii Karabas.
- bpo-42374 [https://bugs.python.org/issue?@action=redirect&bpo=42374]: Fix a regression introduced by the new parser, where an unparenthesized walrus operator was not allowed within generator expressions.
- bpo-42316 [https://bugs.python.org/issue?@action=redirect&bpo=42316]: Allow an unparenthesized walrus in subscript indexes.
- bpo-42349 [https://bugs.python.org/issue?@action=redirect&bpo=42349]: Make sure that the compiler front-end produces a well-formed control flow graph. Be more aggressive in the compiler back-end, as it is now safe to do so.
- bpo-42296 [https://bugs.python.org/issue?@action=redirect&bpo=42296]: On Windows, fix a regression in signal handling which prevented to interrupt a program using CTRL+C. The signal handler can be run in a thread different than the Python thread, in which case the test deciding if the thread can handle signals is wrong.
- bpo-42332 [https://bugs.python.org/issue?@action=redirect&bpo=42332]: **`types.GenericAlias`** objects can now be the targets of weakrefs.
- bpo-42282 [https://bugs.python.org/issue?@action=redirect&bpo=42282]: Optimise constant subexpressions that appear as part of named expressions (previously the AST optimiser did not descend into named expressions). Patch by Nick Coghlan.
- bpo-42266 [https://bugs.python.org/issue?@action=redirect&bpo=42266]: Fixed a bug with the LOAD_ATTR opcode cache that was not respecting monkey-patching a class-level attribute to make it a descriptor. Patch by Pablo Galindo.

Library

- bpo-40077 [https://bugs.python.org/issue?@action=redirect&bpo=40077]: Convert `queue` to use heap types.

Core and Builtins

- bpo-42246 [https://bugs.python.org/issue?@action=redirect&bpo=42246]: Improved accuracy of line tracing events and f_lineno attribute of Frame objects. See PEP 626 for details.

Library

- bpo-40077 [https://bugs.python.org/issue?@action=redirect&bpo=40077]: Convert `mmap` to use heap types.

Core and Builtins

- bpo-42233 [https://bugs.python.org/issue?@action=redirect&bpo=42233]: Allow `GenericAlias` objects to use union type expressions. This allows expressions like `list[int] | dict[float, str]` where previously a `TypeError` would have been thrown. This also fixes union type expressions not de-duplicating `GenericAlias` objects. (Contributed by Ken Jin in bpo-42233 [https://bugs.python.org/issue?@action=redirect&bpo=42233].)
- bpo-26131 [https://bugs.python.org/issue?@action=redirect&bpo=26131]: The import system triggers a `ImportWarning` when it falls back to using `load_module()`.

Library

- bpo-5054 [https://bugs.python.org/issue?@action=redirect&bpo=5054]: CGIHTTPRequestHandler.run_cgi() HTTP_ACCEPT improperly parsed. Replace the special purpose getallmatchingheaders with generic get_all method and add relevant tests.

Original Patch by Martin Panter. Modified by Senthil Kumaran.

- bpo-42562 [https://bugs.python.org/issue?@action=redirect&bpo=42562]: Fix issue when dis failed to parse function that has no line numbers. Patch provided by Yurii Karabas.

- bpo-17735 [https://bugs.python.org/issue?@action=redirect&bpo=17735]: **`inspect.findsource()`** now raises **`OSError`** instead of **`IndexError`** when **`co_lineno`** of a code object is greater than the file length. This can happen, for example, when a file is edited after it was imported. PR by Irit Katriel.

- bpo-42116 [https://bugs.python.org/issue?@action=redirect&bpo=42116]: Fix handling of trailing comments by **`inspect.getsource()`**.

- bpo-42532 [https://bugs.python.org/issue?@action=redirect&bpo=42532]: Remove unexpected call of `__bool__` when passing a `spec_arg` argument to a Mock.

- bpo-38200 [https://bugs.python.org/issue?@action=redirect&bpo=38200]: Added itertools.pairwise()

- bpo-41818 [https://bugs.python.org/issue?@action=redirect&bpo=41818]: Fix test_master_read() so that it succeeds on all platforms that either raise OSError or return b»» upon reading from master.

- bpo-42487 [https://bugs.python.org/issue?@action=redirect&bpo=42487]: ChainMap._iter_ no longer calls _getitem_ on underlying maps

- bpo-42482 [https://bugs.python.org/issue?@action=redirect&bpo=42482]: **`TracebackException`** no longer holds a reference to the exception's traceback object. Consequently, instances of TracebackException for equivalent but non-equal exceptions now compare as equal.

- bpo-41818 [https://bugs.python.org/issue?@action=redirect&bpo=41818]: Make test_openpty() avoid unexpected success due to number of rows and/or number of columns being == 0.

- bpo-42392 [https://bugs.python.org/issue?@action=redirect&bpo=42392]: Remove loop parameter from `asyncio.subprocess` and `asyncio.tasks` functions. Patch provided by Yurii Karabas.

- bpo-42392 [https://bugs.python.org/issue?@action=redirect&bpo=42392]: Remove loop parameter from `asyncio.open_connection` and `asyncio.start_server` functions. Patch provided by Yurii Karabas.

- bpo-28468 [https://bugs.python.org/issue?@action=redirect&bpo=28468]: Add **`platform.freedesktop_os_release()`** function to parse freedesktop.org `os-release` files.

- bpo-42299 [https://bugs.python.org/issue?@action=redirect&bpo=42299]: Removed the `formatter` module, which was deprecated in Python 3.4. It is somewhat obsolete, little used, and not tested. It was originally scheduled to be removed in Python 3.6, but such removals were delayed until after Python 2.7 EOL. Existing users should copy whatever classes they use into their code. Patch by Donghee Na and and Terry J. Reedy.

- bpo-26131 [https://bugs.python.org/issue?@action=redirect&bpo=26131]: Deprecate zipimport.zipimporter.load_module() in favour of exec_module().

- bpo-41818 [https://bugs.python.org/issue?@action=redirect&bpo=41818]: Updated tests for the pty library. test_basic() has been changed to test_openpty(); this additionally checks if slave termios and slave winsize are being set properly by pty.openpty(). In order to add support for FreeBSD, NetBSD, OpenBSD, and Darwin, this also adds test_master_read(), which demonstrates that pty.spawn() should not depend on an OSError to exit from its copy loop.

- bpo-42392 [https://bugs.python.org/issue?@action=redirect&bpo=42392]: Remove loop parameter from `__init__` in all `asyncio.locks` and `asyncio.Queue` classes. Patch provided by Yurii Karabas.

- bpo-15450 [https://bugs.python.org/issue?@action=redirect&bpo=15450]: Make **`filecmp.dircmp`** respect subclassing. Now the **`filecmp.dircmp.subdirs`** behaves as expected when subclassing dircmp.

- bpo-42413 [https://bugs.python.org/issue?@action=redirect&bpo=42413]: The exception **`socket.timeout`** is now an alias of **`TimeoutError`**.

- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Support signal module on VxWorks.

- bpo-42406 [https://bugs.python.org/issue?@action=redirect&bpo=42406]: We fixed an issue in `pickle.whichmodule` in which importing `multiprocessing` could change the how pickle identifies which module an object belongs to, potentially breaking the unpickling of those objects.

- bpo-42403 [https://bugs.python.org/issue?@action=redirect&bpo=42403]: Simplify the **`importlib`** external bootstrap code: `importlib._bootstrap_external` now uses regular imports to import builtin modules. When it is imported, the builtin **`__import__()`** function is already fully working and so can be used to import builtin modules like **`sys`**. Patch by Victor Stinner.

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Convert _sre module types to heap types (PEP 384). Patch by Erlend E. Aasland.

- bpo-42375 [https://bugs.python.org/issue?

@action=redirect&bpo=42375]: subprocess module update for DragonFlyBSD support.

- bpo-41713 [https://bugs.python.org/issue?@action=redirect&bpo=41713]: Port the `_signal` extension module to the multi-phase initialization API (**PEP 489** [https://peps.python.org/pep-0489/]). Patch by Victor Stinner and Mohamed Koubaa.

- bpo-37205 [https://bugs.python.org/issue?@action=redirect&bpo=37205]: `time.time()`, `time.perf_counter()` and `time.monotonic()` functions can no longer fail with a Python fatal error, instead raise a regular Python exception on failure.

- bpo-42328 [https://bugs.python.org/issue?@action=redirect&bpo=42328]: Fixed `tkinter.ttk.Style.map()`. The function accepts now the representation of the default state as empty sequence (as returned by `Style.map()`). The structure of the result is now the same on all platform and does not depend on the value of `wantobjects`.

- bpo-42345 [https://bugs.python.org/issue?@action=redirect&bpo=42345]: Fix various issues with `typing.Literal` parameter handling (flatten, deduplicate, use type to cache key). Patch provided by Yurii Karabas.

- bpo-37205 [https://bugs.python.org/issue?@action=redirect&bpo=37205]: `time.perf_counter()` on Windows and `time.monotonic()` on macOS are now system-wide. Previously, they used an offset computed at startup to reduce the precision loss caused by the float type. Use `time.perf_counter_ns()` and `time.monotonic_ns()` added in Python 3.7 to avoid this precision loss.

- bpo-42318 [https://bugs.python.org/issue?@action=redirect&bpo=42318]: Fixed support of non-BMP characters in `tkinter` on macOS.

- bpo-42350 [https://bugs.python.org/issue?@action=redirect&bpo=42350]: Fix the **`threading.Thread`** class at fork: do nothing if the thread is already stopped (ex: fork called at Python exit). Previously, an error was logged in the child process.

- bpo-42333 [https://bugs.python.org/issue?@action=redirect&bpo=42333]: Port _ssl extension module to heap types.

- bpo-42014 [https://bugs.python.org/issue?@action=redirect&bpo=42014]: The `onerror` callback from `shutil.rmtree` now receives correct function when `os.open` fails.

- bpo-42237 [https://bugs.python.org/issue?@action=redirect&bpo=42237]: Fix `os.sendfile()` on illumos.

- bpo-42308 [https://bugs.python.org/issue?@action=redirect&bpo=42308]: Add **`threading.__excepthook__`** to allow retrieving the original value of **`threading.excepthook()`** in case it is set to a broken or a different value. Patch by Mario Corchero.

- bpo-42131 [https://bugs.python.org/issue?@action=redirect&bpo=42131]: Implement PEP 451/spec methods on zipimport.zipimporter: find_spec(), create_module(), and exec_module().

  This also allows for the documented deprecation of find_loader(), find_module(), and load_module().

- bpo-41877 [https://bugs.python.org/issue?@action=redirect&bpo=41877]: Mock objects which are not unsafe will now raise an AttributeError if an attribute with the prefix asert, aseert, or assrt is accessed, in addition to this already happening for the prefixes assert or assret.

- bpo-42264 [https://bugs.python.org/issue?@action=redirect&bpo=42264]: `sqlite3.OptimizedUnicode` has been undocumented and obsolete since Python 3.3, when it was made an alias to `str`. It is now deprecated, scheduled for removal in Python 3.12.

- bpo-42251 [https://bugs.python.org/issue?@action=redirect&bpo=42251]: Added `threading.gettrace()` and `threading.getprofile()` to retrieve the functions set by `threading.settrace()` and `threading.setprofile()` respectively. Patch by Mario Corchero.

- bpo-42249 [https://bugs.python.org/issue?@action=redirect&bpo=42249]: Fixed writing binary Plist files larger than 4 GiB.

- bpo-42236 [https://bugs.python.org/issue?@action=redirect&bpo=42236]: On Unix, the `os.device_encoding()` function now returns `'UTF-8'` rather than the device encoding if the Python UTF-8 Mode is enabled.

- bpo-41754 [https://bugs.python.org/issue?@action=redirect&bpo=41754]: webbrowser: Ignore *NotADirectoryError* when calling `xdg-settings`.

- bpo-42183 [https://bugs.python.org/issue?@action=redirect&bpo=42183]: Fix a stack overflow error for asyncio Task or Future repr().

  The overflow occurs under some circumstances when a Task or Future recursively returns itself.

- bpo-42140 [https://bugs.python.org/issue?@action=redirect&bpo=42140]: Improve asyncio.wait function to create the futures set just one time.

- bpo-42133 [https://bugs.python.org/issue?@action=redirect&bpo=42133]: Update various modules in the stdlib to fall back on `__spec__.loader` when `__loader__` isn't defined on a module.

- bpo-26131 [https://bugs.python.org/issue?@action=redirect&bpo=26131]: The `load_module()` methods found in `importlib` now trigger a `DeprecationWarning`.

- bpo-39825 [https://bugs.python.org/issue?@action=redirect&bpo=39825]: Windows: Change `sysconfig.get_config_var('EXT_SUFFIX')` to the expected full `platform_tag.extension` format. Previously it was hard-coded to `.pyd`, now it is compatible with `distutils.sysconfig` and will result in something like `.cp38-win_amd64.pyd`. This brings windows into conformance with the other platforms.

- bpo-26389 [https://bugs.python.org/issue?@action=redirect&bpo=26389]: The **`traceback.format_exception()`**, **`traceback.format_exception_only()`**, and **`traceback.print_exception()`** functions can now take an exception object as a positional-only argument.

- bpo-41889 [https://bugs.python.org/issue?@action=redirect&bpo=41889]: Enum: fix regression involving inheriting a multiply inherited enum

- bpo-41861 [https://bugs.python.org/issue?@action=redirect&bpo=41861]: Convert **`sqlite3`** to use heap types (PEP 384). Patch by Erlend E. Aasland.

- bpo-40624 [https://bugs.python.org/issue?@action=redirect&bpo=40624]: Added support for the XPath `!=` operator in xml.etree

- bpo-28850 [https://bugs.python.org/issue?@action=redirect&bpo=28850]: Fix **`pprint.PrettyPrinter.format()`** overrides being ignored for contents of small containers. The **`pprint._safe_repr()`** function was removed.

- bpo-41625 [https://bugs.python.org/issue?@action=redirect&bpo=41625]: Expose the **`splice()`** as

`os.splice()` in the **`os`** module. Patch by Pablo Galindo

- bpo-34215 [https://bugs.python.org/issue?@action=redirect&bpo=34215]: Clarify the error message for **`asyncio.IncompleteReadError`** when `expected` is `None`.

- bpo-41543 [https://bugs.python.org/issue?@action=redirect&bpo=41543]: Add async context manager support for contextlib.nullcontext.

- bpo-21041 [https://bugs.python.org/issue?@action=redirect&bpo=21041]: **`pathlib.PurePath.parents`** now supports negative indexing. Patch contributed by Yaroslav Pankovych.

- bpo-41332 [https://bugs.python.org/issue?@action=redirect&bpo=41332]: Added missing connect_accepted_socket() method to `asyncio.AbstractEventLoop`.

- bpo-12800 [https://bugs.python.org/issue?@action=redirect&bpo=12800]: Extracting a symlink from a tarball should succeed and overwrite the symlink if it already exists. The fix is to remove the existing file or symlink before extraction. Based on patch by Chris AtLee, Jeffrey Kintscher, and Senthil Kumaran.

- bpo-40968 [https://bugs.python.org/issue?@action=redirect&bpo=40968]: **`urllib.request`** and **`http.client`** now send `http/1.1` ALPN extension during TLS handshake when no custom context is supplied.

- bpo-41001 [https://bugs.python.org/issue?@action=redirect&bpo=41001]: Add **`os.eventfd()`** to provide a low level interface for Linux's event notification file descriptor.

- bpo-40816 [https://bugs.python.org/issue?@action=redirect&bpo=40816]: Add AsyncContextDecorator to contextlib to support async context manager as a decorator.

- bpo-40550 [https://bugs.python.org/issue?@action=redirect&bpo=40550]: Fix time-of-check/time-of-action issue in subprocess.Popen.send_signal.

- bpo-39411 [https://bugs.python.org/issue?@action=redirect&bpo=39411]: Add an `is_async` identifier to **`pyclbr`**'s `Function` objects. Patch by Batuhan Taskaya

- bpo-35498 [https://bugs.python.org/issue?@action=redirect&bpo=35498]: Add slice support to **`pathlib.PurePath.parents`**.

Documentation

- bpo-42238 [https://bugs.python.org/issue?@action=redirect&bpo=42238]: Tentative to deprecate `make suspicious` by first removing it from the CI and documentation builds, but keeping it around for manual uses.
- bpo-42153 [https://bugs.python.org/issue?@action=redirect&bpo=42153]: Fix the URL for the IMAP protocol documents.
- bpo-41028 [https://bugs.python.org/issue?@action=redirect&bpo=41028]: Language and version switchers, previously maintained in every cpython branches, are now handled by docsbuild-script.

Tests

- bpo-41473 [https://bugs.python.org/issue?@action=redirect&bpo=41473]: Re-enable test_gdb on gdb 9.2 and newer: https://bugzilla.redhat.com/show_bug.cgi?id=1866884 bug is fixed in gdb 10.1.
- bpo-42553 [https://bugs.python.org/issue?@action=redirect&bpo=42553]: Fix `test_asyncio.test_call_later()` race condition: don't measure asyncio performance in the `call_later()` unit test. The test failed randomly on the CI.
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Fix test_netrc on VxWorks: create temporary directories using temp_cwd().

- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: skip test_getaddrinfo_ipv6_scopeid_symbolic and test_getnameinfo_ipv6_scopeid_symbolic on VxWorks
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: skip test_test of test_mailcap on VxWorks
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: add shell requirement for test_pipes
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: skip some tests related to fifo on VxWorks
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Fix test_doctest.py failures for VxWorks.
- bpo-40754 [https://bugs.python.org/issue?@action=redirect&bpo=40754]: Include `_testinternalcapi` module in Windows installer for test suite
- bpo-41561 [https://bugs.python.org/issue?@action=redirect&bpo=41561]: test_ssl: skip test_min_max_version_mismatch when TLS 1.0 is not available
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Fix os module failures for VxWorks RTOS.
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Fix fifo test cases for VxWorks RTOS.

Build

- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: remove libnet dependency from detect_socket() for VxWorks
- bpo-42398 [https://bugs.python.org/issue?@action=redirect&bpo=42398]: Fix a race condition in «make regen-all» when make -jN option is used to run jobs in parallel. The clinic.py script now only use atomic write to write files. Moveover, generated files are now left unchanged if the content does not change, to not change the file

modification time.

- bpo-41617 [https://bugs.python.org/issue?@action=redirect&bpo=41617]: Fix building `pycore_bitutils.h` internal header on old clang version without `__builtin_bswap16()` (ex: Xcode 4.6.3 on Mac OS X 10.7). Patch by Joshua Root and Victor Stinner.
- bpo-38823 [https://bugs.python.org/issue?@action=redirect&bpo=38823]: It is no longer possible to build the `_ctypes` extension module without **`wchar_t`** type: remove `CTYPES_UNICODE` macro. Anyway, the **`wchar_t`** type is required to build Python. Patch by Victor Stinner.
- bpo-42087 [https://bugs.python.org/issue?@action=redirect&bpo=42087]: Support was removed for AIX 5.3 and below. See bpo-40680 [https://bugs.python.org/issue?@action=redirect&bpo=40680].
- bpo-40998 [https://bugs.python.org/issue?@action=redirect&bpo=40998]: Addressed three compiler warnings found by undefined behavior sanitizer (ubsan).

Windows

- bpo-42120 [https://bugs.python.org/issue?@action=redirect&bpo=42120]: Remove macro definition of `copysign` (to `_copysign`) in headers.
- bpo-38506 [https://bugs.python.org/issue?@action=redirect&bpo=38506]: The Windows launcher now properly handles Python 3.10 when listing installed Python versions.

macOS

- bpo-42504 [https://bugs.python.org/issue?@action=redirect&bpo=42504]: Fix build on macOS Big Sur when MACOSX_DEPLOYMENT_TARGET=11
- bpo-41116 [https://bugs.python.org/issue?@action=redirect&bpo=41116]: Ensure distutils.unixxcompiler.find_library_file can find system provided libraries on macOS 11.

- bpo-41100 [https://bugs.python.org/issue?@action=redirect&bpo=41100]: Add support for macOS 11 and Apple Silicon systems.

  It is now possible to build «Universal 2» binaries using «–enable-universalsdk –with-universal-archs=universal2».

  Binaries build on later macOS versions can be deployed back to older versions (tested up to macOS 10.9), when using the correct deployment target. This is tested using Xcode 11 and later.

- bpo-42232 [https://bugs.python.org/issue?@action=redirect&bpo=42232]: Added Darwin specific madvise options to mmap module.

- bpo-38443 [https://bugs.python.org/issue?@action=redirect&bpo=38443]: The `--enable-universalsdk` and `--with-universal-archs` options for the configure script now check that the specified architectures can be used.

IDLE

- bpo-42508 [https://bugs.python.org/issue?@action=redirect&bpo=42508]: Keep IDLE running on macOS. Remove obsolete workaround that prevented running files with shortcuts when using new universal2 installers built on macOS 11.
- bpo-42426 [https://bugs.python.org/issue?@action=redirect&bpo=42426]: Fix reporting offset of the RE error in searchengine.
- bpo-42415 [https://bugs.python.org/issue?@action=redirect&bpo=42415]: Get docstrings for IDLE calltips more often by using inspect.getdoc.

Tools/Demos

- bpo-42212 [https://bugs.python.org/issue?@action=redirect&bpo=42212]: The smelly.py script now also checks the Python dynamic library and extension modules, not only the Python static library. Make also the script more verbose: explain what it does.

- bpo-36310 [https://bugs.python.org/issue?@action=redirect&bpo=36310]: Allow `Tools/i18n/pygettext.py` to detect calls to `gettext` in f-strings.

C API

- bpo-42423 [https://bugs.python.org/issue?@action=redirect&bpo=42423]: The **`PyType_FromSpecWithBases()`** and **`PyType_FromModuleAndSpec()`** functions now accept a single class as the *bases* argument.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port **`select`** extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _posixsubprocess extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _posixshmem extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/])
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _struct extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/])
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port **`spwd`** extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/])
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port **`gc`** extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/])
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _queue extension module to multiphase initialization (**PEP 489** [https://peps.python.org/

pep-0489/])

- bpo-39573 [https://bugs.python.org/issue?@action=redirect&bpo=39573]: Convert **`Py_TYPE()`** and **`Py_SIZE()`** back to macros to allow using them as an l-value. Many third party C extension modules rely on the ability of using Py_TYPE() and Py_SIZE() to set an object type and size: `Py_TYPE(obj) = type;` and `Py_SIZE(obj) = size;`.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port **`symtable`** extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/])
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port **`grp`** and **`pwd`** extension modules to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/])
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _random extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/])
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _hashlib extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/])
- bpo-41713 [https://bugs.python.org/issue?@action=redirect&bpo=41713]: Removed the undocumented `PyOS_InitInterrupts()` function. Initializing Python already implicitly installs signal handlers: see **`PyConfig.install_signal_handlers`**. Patch by Victor Stinner.
- bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170]: The `Py_TRASHCAN_BEGIN` macro no longer accesses PyTypeObject attributes, but now can get the condition by calling the new private **`_PyTrash_cond()`** function which hides implementation details.
- bpo-42260 [https://bugs.python.org/issue?@action=redirect&bpo=42260]: **`Py_GetPath()`**, **`Py_GetPrefix()`**, **`Py_GetExecPrefix()`**, **`Py_GetProgramFullPath()`**, **`Py_GetPythonHome()`** and

`Py_GetProgramName()` functions now return `NULL` if called before `Py_Initialize()` (before Python is initialized). Use the new Python Initialization Configuration API to get the Python Path Configuration.. Patch by Victor Stinner.

- bpo-42260 [https://bugs.python.org/issue?@action=redirect&bpo=42260]: The `PyConfig_Read()` function now only parses `PyConfig.argv` arguments once: `PyConfig.parse_argv` is set to `2` after arguments are parsed. Since Python arguments are strippped from **`PyConfig.argv`**, parsing arguments twice would parse the application options as Python options.
- bpo-42262 [https://bugs.python.org/issue?@action=redirect&bpo=42262]: Added `Py_NewRef()` and `Py_XNewRef()` functions to increment the reference count of an object and return the object. Patch by Victor Stinner.
- bpo-42260 [https://bugs.python.org/issue?@action=redirect&bpo=42260]: When `Py_Initialize()` is called twice, the second call now updates more `sys` attributes for the configuration, rather than only `sys.argv`. Patch by Victor Stinner.
- bpo-41832 [https://bugs.python.org/issue?@action=redirect&bpo=41832]: The `PyType_FromModuleAndSpec()` function now accepts NULL `tp_doc` slot.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Added `PyModule_AddObjectRef()` function: similar to `PyModule_AddObject()` but don’t steal a reference to the value on success. Patch by Victor Stinner.
- bpo-42171 [https://bugs.python.org/issue?@action=redirect&bpo=42171]: The `METH_FASTCALL` calling convention is added to the limited API. The functions `PyModule_AddType()`, `PyType_FromModuleAndSpec()`, `PyType_GetModule()` and `PyType_GetModuleState()` are added to the limited API on Windows.
- bpo-42085 [https://bugs.python.org/issue?@action=redirect&bpo=42085]: Add dedicated entry to PyAsyncMethods for sending values
- bpo-41073 [https://bugs.python.org/issue?

@action=redirect&bpo=41073]: **`PyType_GetSlot()`** can now accept static types.
- bpo-30459 [https://bugs.python.org/issue?@action=redirect&bpo=30459]: **`PyList_SET_ITEM()`**, **`PyTuple_SET_ITEM()`** and **`PyCell_SET()`** macros can no longer be used as l-value or r-value. For example, `x = PyList_SET_ITEM(a, b, c)` and `PyList_SET_ITEM(a, b, c) = x` now fail with a compiler error. It prevents bugs like `if (PyList_SET_ITEM (a, b, c) < 0) ...` test. Patch by Zackery Spytz and Victor Stinner.

Python 3.10.0 alpha 2

Release date: 2020-11-03

Security

- bpo-42103 [https://bugs.python.org/issue?@action=redirect&bpo=42103]: Prevented potential DoS attack via CPU and RAM exhaustion when processing malformed Apple Property List files in binary format.
- bpo-42051 [https://bugs.python.org/issue?@action=redirect&bpo=42051]: The **`plistlib`** module no longer accepts entity declarations in XML plist files to avoid XML vulnerabilities. This should not affect users as entity declarations are not used in regular plist files.

Core and Builtins

- bpo-42236 [https://bugs.python.org/issue?@action=redirect&bpo=42236]: If the `nl_langinfo(CODESET)` function returns an empty string, Python now uses UTF-8 as the filesystem encoding. Patch by Victor Stinner.
- bpo-42218 [https://bugs.python.org/issue?@action=redirect&bpo=42218]: Fixed a bug in the PEG parser that was causing crashes in debug mode. Now errors are checked in left-recursive rules to avoid cases where such errors do not get handled in time and appear as long-distance crashes in

other places.

- bpo-42214 [https://bugs.python.org/issue?@action=redirect&bpo=42214]: Fixed a possible crash in the PEG parser when checking for the “!=” token in the `barry_as_flufl` rule. Patch by Pablo Galindo.
- bpo-42206 [https://bugs.python.org/issue?@action=redirect&bpo=42206]: Propagate and raise the errors caused by **`PyAST_Validate()`** in the parser.
- bpo-41796 [https://bugs.python.org/issue?@action=redirect&bpo=41796]: The **`ast`** module internal state is now per interpreter. Patch by Victor Stinner.
- bpo-42143 [https://bugs.python.org/issue?@action=redirect&bpo=42143]: Fix handling of errors during creation of `PyFunctionObject`, which resulted in operations on uninitialized memory. Patch by Yonatan Goldschmidt.
- bpo-41659 [https://bugs.python.org/issue?@action=redirect&bpo=41659]: Fix a bug in the parser, where a curly brace following a `primary` didn’t fail immediately. This led to invalid expressions like `a {b}` to throw a **`SyntaxError`** with a wrong offset, or invalid expressions ending with a curly brace like `a {` to not fail immediately in the REPL.
- bpo-42150 [https://bugs.python.org/issue?@action=redirect&bpo=42150]: Fix possible buffer overflow in the new parser when checking for continuation lines. Patch by Pablo Galindo.
- bpo-42123 [https://bugs.python.org/issue?@action=redirect&bpo=42123]: Run the parser two times. On the first run, disable all the rules that only generate better error messages to gain performance. If there’s a parse failure, run the parser a second time with those enabled.
- bpo-42093 [https://bugs.python.org/issue?@action=redirect&bpo=42093]: The `LOAD_ATTR` instruction now uses new «per opcode cache» mechanism and it is about 36% faster now. Patch by Pablo Galindo and Yury Selivanov.
- bpo-42030 [https://bugs.python.org/issue?@action=redirect&bpo=42030]: Support for the legacy AIX-specific shared library loading support has been removed. All versions of AIX since 4.3 have supported and defaulted to

using the common Unix mechanism instead.

- bpo-41984 [https://bugs.python.org/issue?@action=redirect&bpo=41984]: The garbage collector now tracks all user-defined classes. Patch by Brandt Bucher.
- bpo-41993 [https://bugs.python.org/issue?@action=redirect&bpo=41993]: Fixed potential issues with removing not completely initialized module from `sys.modules` when import fails.
- bpo-41979 [https://bugs.python.org/issue?@action=redirect&bpo=41979]: Star-unpacking is now allowed for with item's targets in the PEG parser.
- bpo-41974 [https://bugs.python.org/issue?@action=redirect&bpo=41974]: Removed special methods `__int__`, `__float__`, `__floordiv__`, `__mod__`, `__divmod__`, `__rfloordiv__`, `__rmod__` and `__rdivmod__` of the **`complex`** class. They always raised a **`TypeError`**.
- bpo-41902 [https://bugs.python.org/issue?@action=redirect&bpo=41902]: Micro optimization when compute **`sq_item`** and **`mp_subscript`** of **`range`**. Patch by Donghee Na.
- bpo-41894 [https://bugs.python.org/issue?@action=redirect&bpo=41894]: When loading a native module and a load failure occurs, prevent a possible UnicodeDecodeError when not running in a UTF-8 locale by decoding the load error message using the current locale's encoding.
- bpo-41902 [https://bugs.python.org/issue?@action=redirect&bpo=41902]: Micro optimization for range.index if step is 1. Patch by Donghee Na.

Library

- bpo-41435 [https://bugs.python.org/issue?@action=redirect&bpo=41435]: Add `sys._current_exceptions()` function to retrieve a dictionary mapping each thread's identifier to the topmost exception currently active in that thread at the time the function is called.

Core and Builtins

- bpo-38605 [https://bugs.python.org/issue?@action=redirect&bpo=38605]: Enable `from __future__ import annotations` (**PEP 563** [https://peps.python.org/pep-0563/]) by default. The values found in **`__annotations__`** dicts are now strings, for example `{"x": "int"}` instead of `{"x": int}`.

Library

- bpo-35455 [https://bugs.python.org/issue?@action=redirect&bpo=35455]: On Solaris, **`thread_time()`** is now implemented with `gethrvtime()` because `clock_gettime(CLOCK_THREAD_CPUTIME_ID)` is not always available. Patch by Jakub Kulik.

- bpo-42233 [https://bugs.python.org/issue?@action=redirect&bpo=42233]: The **`repr()`** of **`typing`** types containing Generic Alias Types previously did not show the parameterized types in the `GenericAlias`. They have now been changed to do so.

- bpo-29566 [https://bugs.python.org/issue?@action=redirect&bpo=29566]: `binhex.binhex()` consistently writes macOS 9 line endings.

- bpo-26789 [https://bugs.python.org/issue?@action=redirect&bpo=26789]: The **`logging.FileHandler`** class now keeps a reference to the builtin **`open()`** function to be able to open or reopen the file during Python finalization. Fix errors like: `NameError: name 'open' is not defined`. Patch by Victor Stinner.

- bpo-42157 [https://bugs.python.org/issue?@action=redirect&bpo=42157]: Removed the `unicodedata.ucnhash_CAPI` attribute which was an internal PyCapsule object. The related private `_PyUnicode_Name_CAPI` structure was moved to the internal C API. Patch by Victor Stinner.

- bpo-42157 [https://bugs.python.org/issue?@action=redirect&bpo=42157]: Convert the **`unicodedata`** extension module to the multiphase initialization API (**PEP 489** [https://peps.python.org/pep-0489/]) and convert the `unicodedata.UCD` static type to a heap type. Patch by Mohamed Koubaa and Victor Stinner.

- bpo-42146 [https://bugs.python.org/issue?@action=redirect&bpo=42146]: Fix memory leak in **`subprocess.Popen()`** in case an uid (gid) specified in `user` (`group`, `extra_groups`) overflows `uid_t` (`gid_t`).

- bpo-42103 [https://bugs.python.org/issue?@action=redirect&bpo=42103]: **`InvalidFileException`** and **`RecursionError`** are now the only errors caused by loading malformed binary Plist file (previously ValueError and TypeError could be raised in some specific cases).

- bpo-41490 [https://bugs.python.org/issue?@action=redirect&bpo=41490]: In `importlib.resources`, `.path` method is more aggressive about releasing handles to zipfile objects early, enabling use-cases like certifi to leave the context open but delete the underlying zip file.

- bpo-41052 [https://bugs.python.org/issue?@action=redirect&bpo=41052]: Pickling heap types implemented in C with protocols 0 and 1 raises now an error instead of producing incorrect data.

- bpo-42089 [https://bugs.python.org/issue?@action=redirect&bpo=42089]: In `importlib.metadata.PackageNotFoundError`, make reference to the package metadata being missing to improve the user experience.

- bpo-41491 [https://bugs.python.org/issue?@action=redirect&bpo=41491]: plistlib: fix parsing XML plists with hexadecimal integer values

- bpo-42065 [https://bugs.python.org/issue?@action=redirect&bpo=42065]: Fix an incorrectly formatted error from `_codecs.charmap_decode()` when called with a mapped value outside the range of valid Unicode code points. PR by Max Bernstein.

- bpo-41966 [https://bugs.python.org/issue?@action=redirect&bpo=41966]: Fix pickling pure Python `datetime.time` subclasses. Patch by Dean Inwood.

- bpo-19270 [https://bugs.python.org/issue?@action=redirect&bpo=19270]: `sched.scheduler.cancel()` will now cancel the correct event, if two events with same priority are scheduled for the same time. Patch by Bar Harel.

- bpo-28660 [https://bugs.python.org/issue?@action=redirect&bpo=28660]: `textwrap.wrap()` now attempts to break long words after hyphens when `break_long_words=True` and `break_on_hyphens=True`.

- bpo-35823 [https://bugs.python.org/issue?@action=redirect&bpo=35823]: Use `vfork()` instead of `fork()` for `subprocess.Popen()` on Linux to improve performance in cases where it is deemed safe.

- bpo-42043 [https://bugs.python.org/issue?@action=redirect&bpo=42043]: Add support for `zipfile.Path` inheritance. `zipfile.Path.is_file()` now returns False for non-existent names. `zipfile.Path` objects now expose a `.filename` attribute and rely on that to resolve `.name` and `.parent` when the `Path` object is at the root of the zipfile.

- bpo-42021 [https://bugs.python.org/issue?@action=redirect&bpo=42021]: Fix possible ref leaks in `sqlite3` module init.

- bpo-39101 [https://bugs.python.org/issue?@action=redirect&bpo=39101]: Fixed tests using IsolatedAsyncioTestCase from hanging on BaseExceptions.

- bpo-41976 [https://bugs.python.org/issue?

@action=redirect&bpo=41976]: Fixed a bug that was causing **`ctypes.util.find_library()`** to return `None` when triying to locate a library in an environment when gcc>=9 is available and `ldconfig` is not. Patch by Pablo Galindo

- bpo-41943 [https://bugs.python.org/issue?@action=redirect&bpo=41943]: Fix bug where TestCase.assertLogs doesn't correctly filter messages by level.

- bpo-41923 [https://bugs.python.org/issue?@action=redirect&bpo=41923]: Implement **PEP 613** [https://peps.python.org/pep-0613/], introducing **`typing.TypeAlias`** annotation.

- bpo-41905 [https://bugs.python.org/issue?@action=redirect&bpo=41905]: A new function in abc: *update_abstractmethods* to re-calculate an abstract class's abstract status. In addition, *dataclass* has been changed to call this function.

- bpo-23706 [https://bugs.python.org/issue?@action=redirect&bpo=23706]: Added *newline* parameter to `pathlib.Path.write_text()`.

- bpo-41876 [https://bugs.python.org/issue?@action=redirect&bpo=41876]: Tkinter font class repr uses font name

- bpo-41831 [https://bugs.python.org/issue?@action=redirect&bpo=41831]: `str()` for the `type` attribute of the `tkinter.Event` object always returns now the numeric code returned by Tk instead of the name of the event type.

- bpo-39337 [https://bugs.python.org/issue?@action=redirect&bpo=39337]: **`encodings.normalize_encoding()`** now ignores non-ASCII characters.

- bpo-41747 [https://bugs.python.org/issue?@action=redirect&bpo=41747]: Ensure all methods that generated from **`dataclasses.dataclass()`** objects now have the proper `__qualname__` attribute referring to the class they belong to. Patch by Batuhan Taskaya.

- bpo-30681 [https://bugs.python.org/issue?@action=redirect&bpo=30681]: Handle exceptions caused by unparsable date headers when using email «default» policy. Patch by Tim Bell, Georges Toth

- bpo-41586 [https://bugs.python.org/issue?@action=redirect&bpo=41586]: Add F_SETPIPE_SZ and F_GETPIPE_SZ to fcntl module. Allow setting pipesize on subprocess.Popen.

- bpo-41229 [https://bugs.python.org/issue?@action=redirect&bpo=41229]: Add `contextlib.aclosing` for deterministic cleanup of async generators which is analogous to `contextlib.closing` for non-async generators. Patch by Joongi Kim and John Belmonte.

- bpo-16396 [https://bugs.python.org/issue?@action=redirect&bpo=16396]: Allow `ctypes.wintypes` to be imported on non-Windows systems.

- bpo-4356 [https://bugs.python.org/issue?@action=redirect&bpo=4356]: Add a key function to the bisect module.

- bpo-40592 [https://bugs.python.org/issue?@action=redirect&bpo=40592]: **`shutil.which()`** now ignores empty entries in **`PATHEXT`** instead of treating them as a match.

- bpo-40492 [https://bugs.python.org/issue?@action=redirect&bpo=40492]: Fix `--outfile` for **`cProfile`** / **`profile`** not writing the output file in the original directory when the program being profiled changes the working directory. PR by Anthony Sottile.

- bpo-34204 [https://bugs.python.org/issue?@action=redirect&bpo=34204]: The **`shelve`** module now uses **`pickle.DEFAULT_PROTOCOL`** by default instead of **`pickle`** protocol `3`.

- bpo-27321 [https://bugs.python.org/issue?@action=redirect&bpo=27321]: Fixed KeyError exception when flattening an email to a string attempts to replace a non-existent Content-Transfer-Encoding header.

- bpo-38976 [https://bugs.python.org/issue?@action=redirect&bpo=38976]: The **`http.cookiejar`** module now supports the parsing of cookies in CURL-style cookiejar files through MozillaCookieJar on all platforms. Previously, such cookie entries would be silently ignored when loading a cookiejar with such entries.

  Additionally, the HTTP Only attribute is persisted in the object, and will be correctly written to file if the MozillaCookieJar object is subsequently dumped.

Documentation

- bpo-42061 [https://bugs.python.org/issue?@action=redirect&bpo=42061]: Document _format_ functionality for IP addresses.
- bpo-41910 [https://bugs.python.org/issue?@action=redirect&bpo=41910]: Document the default implementation of `object.__eq__`.
- bpo-42010 [https://bugs.python.org/issue?@action=redirect&bpo=42010]: Clarify that subscription expressions are also valid for certain classes and types in the standard library, and for user-defined classes and types if the classmethod **`__class_getitem__()`** is provided.
- bpo-41805 [https://bugs.python.org/issue?@action=redirect&bpo=41805]: Documented generic alias type and **`types.GenericAlias`**. Also added an entry in glossary for generic types.
- bpo-39693 [https://bugs.python.org/issue?@action=redirect&bpo=39693]: Fix tarfile's extractfile documentation
- bpo-39416 [https://bugs.python.org/issue?@action=redirect&bpo=39416]: Document some restrictions on the default string representations of numeric classes.

Tests

- bpo-41739 [https://bugs.python.org/issue?@action=redirect&bpo=41739]: Fix test_logging.test_race_between_set_target_and_flush(): the test now waits until all threads complete to avoid leaking running threads.
- bpo-41970 [https://bugs.python.org/issue?@action=redirect&bpo=41970]: Avoid a test failure in `test_lib2to3` if the module has already imported at the time the test executes. Patch by Pablo Galindo.
- bpo-41944 [https://bugs.python.org/issue?@action=redirect&bpo=41944]: Tests for CJK codecs no longer call `eval()` on content received via HTTP.
- bpo-41306 [https://bugs.python.org/issue?@action=redirect&bpo=41306]: Fixed a failure in `test_tk.test_widgets.ScaleTest` happening when executing the test with Tk 8.6.10.

Build

- bpo-38980 [https://bugs.python.org/issue?@action=redirect&bpo=38980]: Add `-fno-semantic-interposition` to both the compile and link line when building with `--enable-optimizations`. Patch by Victor Stinner and Pablo Galindo.

Windows

- bpo-38439 [https://bugs.python.org/issue?@action=redirect&bpo=38439]: Updates the icons for IDLE in the Windows Store package.
- bpo-38252 [https://bugs.python.org/issue?@action=redirect&bpo=38252]: Use 8-byte step to detect ASCII sequence in 64-bit Windows build.
- bpo-39107 [https://bugs.python.org/issue?@action=redirect&bpo=39107]: Update Tcl and Tk to 8.6.10 in Windows installer.
- bpo-41557 [https://bugs.python.org/issue?@action=redirect&bpo=41557]: Update Windows installer to use

SQLite 3.33.0.

- bpo-38324 [https://bugs.python.org/issue?@action=redirect&bpo=38324]: Avoid Unicode errors when accessing certain locale data on Windows.

macOS

- bpo-41471 [https://bugs.python.org/issue?@action=redirect&bpo=41471]: Ignore invalid prefix lengths in system proxy excludes.

IDLE

- bpo-33987 [https://bugs.python.org/issue?@action=redirect&bpo=33987]: Mostly finish using ttk widgets, mainly for editor, settings, and searches. Some patches by Mark Roseman.
- bpo-40511 [https://bugs.python.org/issue?@action=redirect&bpo=40511]: Typing opening and closing parentheses inside the parentheses of a function call will no longer cause unnecessary «flashing» off and on of an existing open call-tip, e.g. when typed in a string literal.
- bpo-38439 [https://bugs.python.org/issue?@action=redirect&bpo=38439]: Add a 256 × 256 pixel IDLE icon to the Windows .ico file. Created by Andrew Clover. Remove the low-color gif variations from the .ico file.

C API

- bpo-42157 [https://bugs.python.org/issue?@action=redirect&bpo=42157]: The private `_PyUnicode_Name_CAPI` structure of the PyCapsule API `unicodedata.ucnhash_CAPI` has been moved to the internal C API. Patch by Victor Stinner.
- bpo-42015 [https://bugs.python.org/issue?@action=redirect&bpo=42015]: Fix potential crash in deallocating method objects when dynamically allocated **`PyMethodDef`**'s lifetime is managed through the `self` argument of a **`PyCFunction`**.
- bpo-40423 [https://bugs.python.org/issue?

@action=redirect&bpo=40423]: The **`subprocess`** module and `os.closerange` will now use the `close_range(low, high, flags)` syscall when it is available for more efficient closing of ranges of descriptors.
- bpo-41845 [https://bugs.python.org/issue?@action=redirect&bpo=41845]: **`PyObject_GenericGetDict()`** is available again in the limited API when targeting 3.10 or later.
- bpo-40422 [https://bugs.python.org/issue?@action=redirect&bpo=40422]: Add `_Py_closerange` function to provide performant closing of a range of file descriptors.
- bpo-41986 [https://bugs.python.org/issue?@action=redirect&bpo=41986]: **`Py_FileSystemDefaultEncodeErrors`** and **`Py_UTF8Mode`** are available again in limited API.
- bpo-41756 [https://bugs.python.org/issue?@action=redirect&bpo=41756]: Add `PyIter_Send` function to allow sending value into generator/coroutine/iterator without raising StopIteration exception to signal return.
- bpo-41784 [https://bugs.python.org/issue?@action=redirect&bpo=41784]: Added `PyUnicode_AsUTF8AndSize` to the limited C API.

Python 3.10.0 alpha 1

Release date: 2020-10-05

Security

- bpo-41304 [https://bugs.python.org/issue?@action=redirect&bpo=41304]: Fixes `python3x._pth` being ignored on Windows, caused by the fix for bpo-29778 [https://bugs.python.org/issue?@action=redirect&bpo=29778] (**CVE 2020-15801** [https://www.cve.org/CVERecord?id=CVE-2020-15801]).
- bpo-41162 [https://bugs.python.org/issue?@action=redirect&bpo=41162]: Audit hooks are now cleared later during finalization to avoid missing events.
- bpo-29778 [https://bugs.python.org/issue?@action=redirect&bpo=29778]: Ensure `python3.dll` is loaded

from correct locations when Python is embedded (**CVE 2020-15523** [https://www.cve.org/CVERecord?id=CVE-2020-15523]).
- bpo-41004 [https://bugs.python.org/issue?@action=redirect&bpo=41004]: The _hash_() methods of ipaddress.IPv4Interface and ipaddress.IPv6Interface incorrectly generated constant hash values of 32 and 128 respectively. This resulted in always causing hash collisions. The fix uses hash() to generate hash values for the tuple of (address, mask length, network address).
- bpo-39603 [https://bugs.python.org/issue?@action=redirect&bpo=39603]: Prevent http header injection by rejecting control characters in http.client.putrequest(…).

Core and Builtins

- bpo-41909 [https://bugs.python.org/issue?@action=redirect&bpo=41909]: Fixed stack overflow in **`issubclass()`** and **`isinstance()`** when getting the `__bases__` attribute leads to infinite recursion.
- bpo-41922 [https://bugs.python.org/issue?@action=redirect&bpo=41922]: Speed up calls to `reversed()` by using the **PEP 590** [https://peps.python.org/pep-0590/] `vectorcall` calling convention. Patch by Donghee Na.
- bpo-41873 [https://bugs.python.org/issue?@action=redirect&bpo=41873]: Calls to `float()` are now faster due to the `vectorcall` calling convention. Patch by Dennis Sweeney.
- bpo-41870 [https://bugs.python.org/issue?@action=redirect&bpo=41870]: Speed up calls to `bool()` by using the **PEP 590** [https://peps.python.org/pep-0590/] `vectorcall` calling convention. Patch by Donghee Na.

Library

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the **`_bisect`** module to the multi-phase initialization API (**PEP 489** [https://peps.python.org/pep-0489/]).

Core and Builtins

- bpo-39934 [https://bugs.python.org/issue?@action=redirect&bpo=39934]: Correctly count control blocks in "except" in compiler. Ensures that a syntax error, rather a fatal error, occurs for deeply nested, named exception handlers.
- bpo-41780 [https://bugs.python.org/issue?@action=redirect&bpo=41780]: Fix `__dir__()` of `types.GenericAlias`. Patch by Batuhan Taskaya.

Library

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the `_lsprof` extension module to multi-phase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the `cmath` extension module to multi-phase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the `_scproxy` extension module to multi-phase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the `termios` extension module to multi-phase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Convert the `_sha256` extension module types to heap types.

Core and Builtins

- bpo-41690 [https://bugs.python.org/issue?@action=redirect&bpo=41690]: Fix a possible stack overflow in the parser when parsing functions and classes with a huge amount of arguments. Patch by Pablo Galindo.

Library

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the `_overlapped` extension module to multi-phase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the `_curses_panel` extension module to multi-phase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the `_opcode` extension module to multi-phase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).

Core and Builtins

- bpo-41681 [https://bugs.python.org/issue?@action=redirect&bpo=41681]: Fixes the wrong error description in the error raised by using 2 `,` in format string in f-string and `str.format()`.

Library

- bpo-41675 [https://bugs.python.org/issue?@action=redirect&bpo=41675]: The implementation of `signal.siginterrupt()` now uses `sigaction()` (if it is available in the system) instead of the deprecated `siginterrupt()`. Patch by Pablo Galindo.

Core and Builtins

- bpo-41670 [https://bugs.python.org/issue?@action=redirect&bpo=41670]: Prevent line trace being skipped on platforms not compiled with `USE_COMPUTED_GOTOS`. Fixes issue where some lines nested within a try-except block were not being traced on Windows.
- bpo-41654 [https://bugs.python.org/issue?@action=redirect&bpo=41654]: Fix a crash that occurred when destroying subclasses of `MemoryError`. Patch by Pablo Galindo.

Library

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the **`zlib`** extension module to multi-phase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).

Core and Builtins

- bpo-41631 [https://bugs.python.org/issue?@action=redirect&bpo=41631]: The `_ast` module uses again a global state. Using a module state per module instance is causing subtle practical problems. For example, the Mercurial project replaces the `__import__()` function to implement lazy import, whereas Python expected that `import _ast` always return a fully initialized `_ast` module.

Library

- bpo-40077 [https://bugs.python.org/issue?@action=redirect&bpo=40077]: Convert **`_operator`** to use **`PyType_FromSpec()`**.

Core and Builtins

- bpo-1653741 [https://bugs.python.org/issue?@action=redirect&bpo=1653741]: Port **`_sha3`** to multi-phase init. Convert static types to heap types.

Library

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the **`_blake2`** extension module to the multi-phase initialization API (**PEP 489** [https://peps.python.org/pep-0489/]).

Core and Builtins

- bpo-41533 [https://bugs.python.org/issue?

@action=redirect&bpo=41533]: Free the stack allocated in `va_build_stack` if `do_mkstack` fails and the stack is not a `small_stack`.

- bpo-41531 [https://bugs.python.org/issue?@action=redirect&bpo=41531]: Fix a bug that was dropping keys when compiling dict literals with more than 0xFFFF elements. Patch by Pablo Galindo.
- bpo-41525 [https://bugs.python.org/issue?@action=redirect&bpo=41525]: The output of `python --help` contains now only ASCII characters.

Library

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the **`_sha1`**, **`_sha512`**, and **`_md5`** extension modules to multi-phase initialization API (**PEP 489** [https://peps.python.org/pep-0489/]).

Core and Builtins

- bpo-41431 [https://bugs.python.org/issue?@action=redirect&bpo=41431]: Optimize `dict_merge()` for copying dict (e.g. `dict(d)` and `{}.update(d)`).
- bpo-41428 [https://bugs.python.org/issue?@action=redirect&bpo=41428]: Implement PEP 604. This supports (int | str) etc. in place of Union[str, int].
- bpo-41340 [https://bugs.python.org/issue?@action=redirect&bpo=41340]: Removed fallback implementation for `strdup`.
- bpo-38156 [https://bugs.python.org/issue?@action=redirect&bpo=38156]: Handle interrupts that come after EOF correctly in `PyOS_StdioReadline`.
- bpo-41342 [https://bugs.python.org/issue?@action=redirect&bpo=41342]: **`round()`** with integer argument is now faster (9–60%).
- bpo-41334 [https://bugs.python.org/issue?@action=redirect&bpo=41334]: Constructors **`str()`**, **`bytes()`** and **`bytearray()`** are now faster (around 30–40% for small objects).
- bpo-41295 [https://bugs.python.org/issue?

@action=redirect&bpo=41295]: Resolve a regression in CPython 3.8.4 where defining «_setattr_» in a multi-inheritance setup and calling up the hierarchy chain could fail if builtins/ extension types were involved in the base types.
- bpo-41323 [https://bugs.python.org/issue?@action=redirect&bpo=41323]: Bytecode optimizations are performed directly on the control flow graph. This will result in slightly more compact code objects in some circumstances.
- bpo-41247 [https://bugs.python.org/issue?@action=redirect&bpo=41247]: Always cache the running loop holder when running `asyncio.set_running_loop`.
- bpo-41252 [https://bugs.python.org/issue?@action=redirect&bpo=41252]: Fix incorrect refcounting in _ssl.c's `_servername_callback()`.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port **`multiprocessing`** to multi-phase initialization

Library

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port **`winapi`** to multiphase initialization

Core and Builtins

- bpo-41215 [https://bugs.python.org/issue?@action=redirect&bpo=41215]: Use non-NULL default values in the PEG parser keyword list to overcome a bug that was preventing Python from being properly compiled when using the XLC compiler. Patch by Pablo Galindo.
- bpo-41218 [https://bugs.python.org/issue?@action=redirect&bpo=41218]: Python 3.8.3 had a regression where compiling with ast.PyCF_ALLOW_TOP_LEVEL_AWAIT would aggressively mark list comprehension with CO_COROUTINE. Now only list comprehension making use of async/await will tagged as so.

Library

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port `faulthandler` to multiphase initialization.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port `sha256` to multiphase initialization

Core and Builtins

- bpo-41175 [https://bugs.python.org/issue?@action=redirect&bpo=41175]: Guard against a NULL pointer dereference within bytearrayobject triggered by the `bytearray() + bytearray()` operation.
- bpo-41100 [https://bugs.python.org/issue?@action=redirect&bpo=41100]: add arm64 to the allowable Mac OS arches in mpdecimal.h
- bpo-41094 [https://bugs.python.org/issue?@action=redirect&bpo=41094]: Fix decoding errors with audit when open files with non-ASCII names on non-UTF-8 locale.
- bpo-39960 [https://bugs.python.org/issue?@action=redirect&bpo=39960]: The «hackcheck» that prevents sneaking around a type's _setattr_() by calling the superclass method was rewritten to allow C implemented heap types.
- bpo-41084 [https://bugs.python.org/issue?@action=redirect&bpo=41084]: Prefix the error message with "f-string: ", when parsing an f-string expression which throws a `SyntaxError`.
- bpo-40521 [https://bugs.python.org/issue?@action=redirect&bpo=40521]: Empty frozensets are no longer singletons.
- bpo-41076 [https://bugs.python.org/issue?@action=redirect&bpo=41076]: Pre-feed the parser with the location of the f-string expression, not the f-string itself, which allows us to skip the shifting of the AST node locations after the parsing is completed.
- bpo-41056 [https://bugs.python.org/issue?@action=redirect&bpo=41056]: Fixes a reference to deallocated stack space during startup when constructing sys.path involving a relative symlink when code was supplied via -c. (discovered via Coverity)

- bpo-41061 [https://bugs.python.org/issue?@action=redirect&bpo=41061]: Fix incorrect expressions and asserts in hashtable code and tests.
- bpo-41052 [https://bugs.python.org/issue?@action=redirect&bpo=41052]: Opt out serialization/deserialization for _random.Random
- bpo-40939 [https://bugs.python.org/issue?@action=redirect&bpo=40939]: Rename `PyPegen*` functions to `PyParser*`, so that we can remove the old set of `PyParser*` functions that were using the old parser, but keep everything backwards-compatible.
- bpo-35975 [https://bugs.python.org/issue?@action=redirect&bpo=35975]: Stefan Behnel reported that cf_feature_version is used even when PyCF_ONLY_AST is not set. This is against the intention and against the documented behavior, so it's been fixed.
- bpo-40939 [https://bugs.python.org/issue?@action=redirect&bpo=40939]: Remove the remaining files from the old parser and the **`symbol`** module.

Library

- bpo-40077 [https://bugs.python.org/issue?@action=redirect&bpo=40077]: Convert **`_bz2`** to use **`PyType_FromSpec()`**.

Core and Builtins

- bpo-41006 [https://bugs.python.org/issue?@action=redirect&bpo=41006]: The `encodings.latin_1` module is no longer imported at startup. Now it is only imported when it is the filesystem encoding or the stdio encoding.
- bpo-40636 [https://bugs.python.org/issue?@action=redirect&bpo=40636]: **`zip()`** now supports **PEP 618** [https://peps.python.org/pep-0618/]'s `strict` parameter, which raises a **`ValueError`** if the arguments are exhausted at different lengths. Patch by Brandt Bucher.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port **`_gdbm`** to multiphase

initialization.

- bpo-40985 [https://bugs.python.org/issue?@action=redirect&bpo=40985]: Fix a bug that caused the `SyntaxError` text to be empty when a file ends with a line ending in a line continuation character (i.e. backslash). The error text should contain the text of the last line.
- bpo-40958 [https://bugs.python.org/issue?@action=redirect&bpo=40958]: Fix a possible buffer overflow in the PEG parser when gathering information for emitting syntax errors. Patch by Pablo Galindo.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port `_dbm` to multiphase initialization.
- bpo-40957 [https://bugs.python.org/issue?@action=redirect&bpo=40957]: Fix refleak in _Py_fopen_obj() when PySys_Audit() fails

Library

- bpo-40950 [https://bugs.python.org/issue?@action=redirect&bpo=40950]: Add a state to the `nis` module (**PEP 3121** [https://peps.python.org/pep-3121/]) and apply the multiphase initialization. Patch by Donghee Na.

Core and Builtins

- bpo-40947 [https://bugs.python.org/issue?@action=redirect&bpo=40947]: The Python Path Configuration now takes `PyConfig.platlibdir` in account.

- bpo-40939 [https://bugs.python.org/issue?@action=redirect&bpo=40939]: Remove the old parser, the `parser` module and all associated support code, command-line options and environment variables. Patch by Pablo Galindo.

- bpo-40847 [https://bugs.python.org/issue?@action=redirect&bpo=40847]: Fix a bug where a line with only a line continuation character is not considered a blank line at tokenizer level. In such cases, more than a single `NEWLINE` token was emitted. The old parser was working around the issue, but the new parser threw a `SyntaxError` for valid input due to this. For example, an empty line following a line continuation character was interpreted as a `SyntaxError`.

- bpo-40890 [https://bugs.python.org/issue?@action=redirect&bpo=40890]: Each dictionary view now has a `mapping` attribute that provides a `types.MappingProxyType` wrapping the original dictionary. Patch contributed by Dennis Sweeney.

- bpo-40889 [https://bugs.python.org/issue?@action=redirect&bpo=40889]: Improved the performance of symmetric difference operations on dictionary item views. Patch by Dennis Sweeney.

- bpo-40904 [https://bugs.python.org/issue?@action=redirect&bpo=40904]: Fix possible segfault in the new PEG parser when parsing f-string containing yield statements with no value (`f"{yield}"`). Patch by Pablo Galindo

- bpo-40903 [https://bugs.python.org/issue?@action=redirect&bpo=40903]: Fixed a possible segfault in the new PEG parser when producing error messages for invalid assignments of the form `p=p=`. Patch by Pablo Galindo

- bpo-40880 [https://bugs.python.org/issue?@action=redirect&bpo=40880]: Fix invalid memory read in the new parser when checking newlines in string literals. Patch by Pablo Galindo.

- bpo-40883 [https://bugs.python.org/issue?@action=redirect&bpo=40883]: Fix memory leak in when parsing f-strings in the new parser. Patch by Pablo Galindo

- bpo-40870 [https://bugs.python.org/issue?@action=redirect&bpo=40870]: Raise `ValueError` when validating custom AST's where the constants `True`, `False` and `None` are used within a `ast.Name` node.

- bpo-40854 [https://bugs.python.org/issue?

@action=redirect&bpo=40854]: Allow overriding `sys.platlibdir` via a new `PYTHONPLATLIBDIR` environment variable.

- bpo-40826 [https://bugs.python.org/issue?@action=redirect&bpo=40826]: Fix GIL usage in **`PyOS_Readline()`**: lock the GIL to set an exception and pass the Python thread state when checking if there is a pending signal.

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port `fcntl` to multiphase initialization.

- bpo-19468 [https://bugs.python.org/issue?@action=redirect&bpo=19468]: Delete unnecessary instance check in importlib.reload(). Patch by Furkan Önder.

- bpo-40824 [https://bugs.python.org/issue?@action=redirect&bpo=40824]: Unexpected errors in calling the `__iter__` method are no longer masked by `TypeError` in the `in` operator and functions `contains()`, `indexOf()` and `countOf()` of the `operator` module.

- bpo-40792 [https://bugs.python.org/issue?@action=redirect&bpo=40792]: Attributes `start`, `stop` and `step` of the `range` object now always has exact type `int`. Previously, they could have been an instance of a subclass of `int`.

- bpo-40780 [https://bugs.python.org/issue?@action=redirect&bpo=40780]: Fix a corner case where g-style string formatting of a float failed to remove trailing zeros.

- bpo-38964 [https://bugs.python.org/issue?@action=redirect&bpo=38964]: When there's a `SyntaxError` in the expression part of an fstring, the filename attribute of the **`SyntaxError`** gets correctly set to the name of the file the fstring resides in.

- bpo-40750 [https://bugs.python.org/issue?

@action=redirect&bpo=40750]: Support the «-d» debug flag in the new PEG parser. Patch by Pablo Galindo

- bpo-40217 [https://bugs.python.org/issue?@action=redirect&bpo=40217]: Instances of types created with `PyType_FromSpecWithBases()` will no longer automatically visit their class object when traversing references in the garbage collector. The user is expected to manually visit the object's class. Patch by Pablo Galindo.

- bpo-39573 [https://bugs.python.org/issue?@action=redirect&bpo=39573]: `Py_TYPE()` is changed to the inline static function. Patch by Donghee Na.

- bpo-40696 [https://bugs.python.org/issue?@action=redirect&bpo=40696]: Fix a hang that can arise after `generator.throw()` due to a cycle in the exception context chain.

- bpo-40521 [https://bugs.python.org/issue?@action=redirect&bpo=40521]: Each interpreter now its has own free lists, singletons and caches:

    - Free lists: float, tuple, list, dict, frame, context, asynchronous generator, MemoryError.
    - Singletons: empty tuple, empty bytes string, empty Unicode string, single byte character, single Unicode (latin1) character.
    - Slice cache.

  They are no longer shared by all interpreters.

- bpo-40679 [https://bugs.python.org/issue?@action=redirect&bpo=40679]: Certain `TypeError` messages about missing or extra arguments now include the function's qualified name. Patch by Dennis Sweeney.

- bpo-29590 [https://bugs.python.org/issue?@action=redirect&bpo=29590]: Make the stack trace correct after calling `generator.throw()` on a generator that has yielded from a `yield from`.

- bpo-4022 [https://bugs.python.org/issue?@action=redirect&bpo=4022]: Improve performance of generators by not raising internal StopIteration.

Library

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port `mmap` to multiphase initialization.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port `_lzma` to multiphase initialization.

Core and Builtins

- bpo-37999 [https://bugs.python.org/issue?@action=redirect&bpo=37999]: Builtin and extension functions that take integer arguments no longer accept `Decimal`s, `Fraction`s and other objects that can be converted to integers only with a loss (e.g. that have the `__int__()` method but do not have the `__index__()` method).
- bpo-29882 [https://bugs.python.org/issue?@action=redirect&bpo=29882]: Add `int.bit_count()`, counting the number of ones in the binary representation of an integer. Patch by Niklas Fiekas.

Library

- bpo-36982 [https://bugs.python.org/issue?@action=redirect&bpo=36982]: Use ncurses extended color functions when available to support terminals with 256 colors, and add the new function `curses.has_extended_color_support()` to indicate whether extended color support is provided by the underlying ncurses library.

Core and Builtins

- bpo-19569 [https://bugs.python.org/issue?

@action=redirect&bpo=19569]: Add the private macros `_Py_COMP_DIAG_PUSH`, `_Py_COMP_DIAG_IGNORE_DEPR_DECLS`, and `_Py_COMP_DIAG_POP`.

- bpo-26680 [https://bugs.python.org/issue?@action=redirect&bpo=26680]: The int type now supports the x.is_integer() method for compatibility with float.

Library

- bpo-41900 [https://bugs.python.org/issue?@action=redirect&bpo=41900]: C14N 2.0 serialisation in xml.etree.ElementTree failed for unprefixed attributes when a default namespace was defined.

- bpo-41887 [https://bugs.python.org/issue?@action=redirect&bpo=41887]: Strip leading spaces and tabs on `ast.literal_eval()`. Also document stripping of spaces and tabs for `eval()`.

- bpo-41773 [https://bugs.python.org/issue?@action=redirect&bpo=41773]: Note in documentation that `random.choices()` doesn't support non-finite weights, raise `ValueError` when given non-finite weights.

- bpo-41840 [https://bugs.python.org/issue?@action=redirect&bpo=41840]: Fix a bug in the `symtable` module that was causing module-scope global variables to not be reported as both local and global. Patch by Pablo Galindo.

- bpo-41842 [https://bugs.python.org/issue?@action=redirect&bpo=41842]: Add `codecs.unregister()` function to unregister a codec search function.

- bpo-40564 [https://bugs.python.org/issue?@action=redirect&bpo=40564]: In `zipfile.Path`, mutate the passed ZipFile object type instead of making a copy. Prevents issues when both the local copy and the caller's copy attempt to close the same file handle.

- bpo-40670 [https://bugs.python.org/issue?

@action=redirect&bpo=40670]: More reliable validation of statements in `timeit.Timer`. It now accepts «empty» statements (only whitespaces and comments) and rejects misindentent statements.

- bpo-41833 [https://bugs.python.org/issue?@action=redirect&bpo=41833]: The `threading.Thread` constructor now uses the target name if the *target* argument is specified but the *name* argument is omitted.

- bpo-41817 [https://bugs.python.org/issue?@action=redirect&bpo=41817]: fix `tkinter.EventType` Enum so all members are strings, and none are tuples

- bpo-41810 [https://bugs.python.org/issue?@action=redirect&bpo=41810]: `types.EllipsisType`, `types.NotImplementedType` and `types.NoneType` have been reintroduced, providing a new set of types readily interpretable by static type checkers.

- bpo-41815 [https://bugs.python.org/issue?@action=redirect&bpo=41815]: Fix SQLite3 segfault when backing up closed database. Patch contributed by Peter David McCormick.

- bpo-41816 [https://bugs.python.org/issue?@action=redirect&bpo=41816]: StrEnum added: it ensures that all members are already strings or string candidates

- bpo-41517 [https://bugs.python.org/issue?@action=redirect&bpo=41517]: fix bug allowing Enums to be extended via multiple inheritance

- bpo-39587 [https://bugs.python.org/issue?@action=redirect&bpo=39587]: use the correct mix-in data type when constructing Enums

- bpo-41792 [https://bugs.python.org/issue?@action=redirect&bpo=41792]: Add is_typeddict function to typing.py to check if a type is a TypedDict class

Previously there was no way to check that without using private API. See the relevant issue in python/typing [https://github.com/python/typing/issues/751].

- bpo-41789 [https://bugs.python.org/issue?@action=redirect&bpo=41789]: Honor `object` overrides in `Enum` class creation (specifically, `__str__`, `__repr__`, `__format__`, and `__reduce_ex__`).

- bpo-32218 [https://bugs.python.org/issue?@action=redirect&bpo=32218]: `enum.Flag` and `enum.IntFlag` members are now iterable.

- bpo-39651 [https://bugs.python.org/issue?@action=redirect&bpo=39651]: Fix a race condition in the `call_soon_threadsafe()` method of `asyncio.ProactorEventLoop`: do nothing if the self-pipe socket has been closed.

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the `mashal` extension module to the multi-phase initialization API (**PEP 489** [https://peps.python.org/pep-0489/]).

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port the `_string` extension module to the multi-phase initialization API (**PEP 489** [https://peps.python.org/pep-0489/]).

- bpo-41732 [https://bugs.python.org/issue?@action=redirect&bpo=41732]: Added an iterator to **`memoryview`**.

- bpo-41720 [https://bugs.python.org/issue?@action=redirect&bpo=41720]: Fixed **`turtle.Vec2D.__rmul__()`** for arguments which are not int or float.

- bpo-41696 [https://bugs.python.org/issue?@action=redirect&bpo=41696]: Fix handling of debug mode in **`asyncio.run()`**. This allows setting

`PYTHONASYNCIODEBUG` or `-X dev` to enable asyncio debug mode when using **`asyncio.run()`**.

- bpo-41687 [https://bugs.python.org/issue?@action=redirect&bpo=41687]: Fix implementation of sendfile to be compatible with Solaris.
- bpo-41662 [https://bugs.python.org/issue?@action=redirect&bpo=41662]: No longer override exceptions raised in `__len__()` of a sequence of parameters in **`sqlite3`** with **`ProgrammingError`**.
- bpo-39010 [https://bugs.python.org/issue?@action=redirect&bpo=39010]: Restarting a `ProactorEventLoop` on Windows no longer logs spurious `ConnectionResetErrors`.
- bpo-41638 [https://bugs.python.org/issue?@action=redirect&bpo=41638]: **`ProgrammingError`** message for absent parameter in **`sqlite3`** contains now the name of the parameter instead of its index when parameters are supplied as a dict.
- bpo-41662 [https://bugs.python.org/issue?@action=redirect&bpo=41662]: Fixed crash when mutate list of parameters during iteration in **`sqlite3`**.
- bpo-41513 [https://bugs.python.org/issue?@action=redirect&bpo=41513]: Improved the accuracy of math.hypot(). Internally, each step is computed with extra precision so that the result is now almost always correctly rounded.
- bpo-41609 [https://bugs.python.org/issue?@action=redirect&bpo=41609]: The pdb whatis command correctly reports instance methods as “Method” rather than “Function”.
- bpo-39994 [https://bugs.python.org/issue?@action=redirect&bpo=39994]: Fixed pprint’s handling of dict subclasses that override _repr_.

- bpo-32751 [https://bugs.python.org/issue?@action=redirect&bpo=32751]: When cancelling the task due to a timeout, `asyncio.wait_for()` will now wait until the cancellation is complete also in the case when *timeout* is <= 0, like it does with positive timeouts.

- bpo-37658 [https://bugs.python.org/issue?@action=redirect&bpo=37658]: `asyncio.wait_for()` now properly handles races between cancellation of itself and the completion of the wrapped awaitable.

- bpo-40782 [https://bugs.python.org/issue?@action=redirect&bpo=40782]: Change the method asyncio.AbstractEventLoop.run_in_executor to not be a coroutine.

- bpo-41520 [https://bugs.python.org/issue?@action=redirect&bpo=41520]: Fix `codeop` regression that prevented turning compile warnings into errors.

- bpo-41528 [https://bugs.python.org/issue?@action=redirect&bpo=41528]: turtle uses math module functions to convert degrees to radians and vice versa and to calculate vector norm

- bpo-41513 [https://bugs.python.org/issue?@action=redirect&bpo=41513]: Minor algorithmic improvement to math.hypot() and math.dist() giving small gains in speed and accuracy.

- bpo-41503 [https://bugs.python.org/issue?@action=redirect&bpo=41503]: Fixed a race between setTarget and flush in logging.handlers.MemoryHandler.

- bpo-41497 [https://bugs.python.org/issue?@action=redirect&bpo=41497]: Fix potential UnicodeDecodeError in dis module.

- bpo-41467 [https://bugs.python.org/issue?@action=redirect&bpo=41467]: On Windows, fix asyncio `recv_into()` return value when the socket/pipe is closed

(`BrokenPipeError`): return `0` rather than an empty byte string (`b''`).

- bpo-41425 [https://bugs.python.org/issue?@action=redirect&bpo=41425]: Make tkinter doc example runnable.
- bpo-41421 [https://bugs.python.org/issue?@action=redirect&bpo=41421]: Make an algebraic simplification to random.paretovariate(). It now is slightly less subject to round-off error and is slightly faster. Inputs that used to cause ZeroDivisionError now cause an OverflowError instead.
- bpo-41440 [https://bugs.python.org/issue?@action=redirect&bpo=41440]: Add `os.cpu_count()` support for VxWorks RTOS.
- bpo-41316 [https://bugs.python.org/issue?@action=redirect&bpo=41316]: Fix the `tarfile` module to write only basename of TAR file to GZIP compression header.
- bpo-41384 [https://bugs.python.org/issue?@action=redirect&bpo=41384]: Raise TclError instead of TypeError when an unknown option is passed to tkinter.OptionMenu.
- bpo-41317 [https://bugs.python.org/issue?@action=redirect&bpo=41317]: Use add_done_callback() in asyncio.loop.sock_accept() to unsubscribe reader early on cancellation.
- bpo-41364 [https://bugs.python.org/issue?@action=redirect&bpo=41364]: Reduce import overhead of `uuid`.
- bpo-35328 [https://bugs.python.org/issue?@action=redirect&bpo=35328]: Set the environment variable `VIRTUAL_ENV_PROMPT` at `venv` activation.
- bpo-41341 [https://bugs.python.org/issue?@action=redirect&bpo=41341]: Recursive evaluation of

`typing.ForwardRef` in `get_type_hints`.

- bpo-41344 [https://bugs.python.org/issue?@action=redirect&bpo=41344]: Prevent creating **`shared_memory.SharedMemory`** objects with `size=0`.

- bpo-41333 [https://bugs.python.org/issue?@action=redirect&bpo=41333]: **`collections.OrderedDict.pop()`** is now 2 times faster.

- bpo-41288 [https://bugs.python.org/issue?@action=redirect&bpo=41288]: Unpickling invalid NEWOBJ_EX opcode with the C implementation raises now UnpicklingError instead of crashing.

- bpo-39017 [https://bugs.python.org/issue?@action=redirect&bpo=39017]: Avoid infinite loop when reading specially crafted TAR files using the tarfile module (**CVE 2019-20907** [https://www.cve.org/CVERecord?id=CVE-2019-20907]).

- bpo-41273 [https://bugs.python.org/issue?@action=redirect&bpo=41273]: Speed up any transport using `_ProactorReadPipeTransport` by calling `recv_into` instead of `recv`, thus not creating a new buffer for each `recv` call in the transport's read loop.

- bpo-41235 [https://bugs.python.org/issue?@action=redirect&bpo=41235]: Fix the error handling in **`ssl.SSLContext.load_dh_params()`**.

- bpo-41207 [https://bugs.python.org/issue?@action=redirect&bpo=41207]: In distutils.spawn, restore expectation that DistutilsExecError is raised when the command is not found.

- bpo-29727 [https://bugs.python.org/issue?@action=redirect&bpo=29727]: Register **`array.array`** as a **`MutableSequence`**. Patch by Pablo Galindo.

- bpo-39168 [https://bugs.python.org/issue?@action=redirect&bpo=39168]: Remove the `__new__` method of `typing.Generic`.

- bpo-41194 [https://bugs.python.org/issue?@action=redirect&bpo=41194]: Fix a crash in the `_ast` module: it can no longer be loaded more than once. It now uses a global state rather than a module state.

- bpo-41195 [https://bugs.python.org/issue?@action=redirect&bpo=41195]: Add read-only ssl.SSLContext.security_level attribute to retrieve the context's security level.

- bpo-41193 [https://bugs.python.org/issue?@action=redirect&bpo=41193]: The `write_history()` atexit function of the readline completer now ignores any **`OSError`** to ignore error if the filesystem is read-only, instead of only ignoring **`FileNotFoundError`** and **`PermissionError`**.

- bpo-41182 [https://bugs.python.org/issue?@action=redirect&bpo=41182]: selector: use DefaultSelector based upon implementation

- bpo-41161 [https://bugs.python.org/issue?@action=redirect&bpo=41161]: The decimal module now requires libmpdec-2.5.0. Users of –with-system-libmpdec should update their system library.

- bpo-40874 [https://bugs.python.org/issue?@action=redirect&bpo=40874]: The decimal module now requires libmpdec-2.5.0.

- bpo-41138 [https://bugs.python.org/issue?@action=redirect&bpo=41138]: Fixed the **`trace`** module CLI for Python source files with non-UTF-8 encoding.

- bpo-31082 [https://bugs.python.org/issue?@action=redirect&bpo=31082]: Use the term «iterable» in the docstring for **`functools.reduce()`**.

- bpo-40521 [https://bugs.python.org/issue?@action=redirect&bpo=40521]: Remove freelist from collections.deque().

- bpo-31938 [https://bugs.python.org/issue?@action=redirect&bpo=31938]: Fix default-value signatures of several functions in the `select` module - by Anthony Sottile.

- bpo-41068 [https://bugs.python.org/issue?@action=redirect&bpo=41068]: Fixed reading files with non-ASCII names from ZIP archive directly after writing them.

- bpo-41058 [https://bugs.python.org/issue?@action=redirect&bpo=41058]: `pdb.find_function()` now correctly determines the source file encoding.

- bpo-41056 [https://bugs.python.org/issue?@action=redirect&bpo=41056]: Invalid file descriptor values are now prevented from being passed to os.fpathconf. (discovered by Coverity)

- bpo-41056 [https://bugs.python.org/issue?@action=redirect&bpo=41056]: Fix a NULL pointer dereference within the ssl module during a MemoryError in the keylog callback. (discovered by Coverity)

- bpo-41056 [https://bugs.python.org/issue?@action=redirect&bpo=41056]: Fixed an instance where a MemoryError within the zoneinfo module might not be reported or not reported at its source. (found by Coverity)

- bpo-41048 [https://bugs.python.org/issue?@action=redirect&bpo=41048]: `mimetypes.read_mime_types()` function reads the rule file using UTF-8 encoding, not the locale encoding. Patch by Srinivas Reddy Thatiparthy.

- bpo-41043 [https://bugs.python.org/issue?@action=redirect&bpo=41043]: Fixed the use of `glob()` in the stdlib: literal part of the path is now always correctly escaped.

- bpo-41025 [https://bugs.python.org/issue?

@action=redirect&bpo=41025]: Fixed an issue preventing the C implementation of `zoneinfo.ZoneInfo` from being subclassed.

- bpo-35018 [https://bugs.python.org/issue?@action=redirect&bpo=35018]: Add the `xml.sax.handler.LexicalHandler` class that is present in other SAX XML implementations.
- bpo-41002 [https://bugs.python.org/issue?@action=redirect&bpo=41002]: Improve performance of HTTPResponse.read with a given amount. Patch by Bruce Merry.
- bpo-40448 [https://bugs.python.org/issue?@action=redirect&bpo=40448]: `ensurepip` now disables the use of `pip` cache when installing the bundled versions of `pip` and `setuptools`. Patch by Krzysztof Konopko.
- bpo-40967 [https://bugs.python.org/issue?@action=redirect&bpo=40967]: Removed `asyncio.Task.current_task()` and `asyncio.Task.all_tasks()`. Patch contributed by Rémi Lapeyre.
- bpo-40924 [https://bugs.python.org/issue?@action=redirect&bpo=40924]: Ensure `importlib.resources.path` returns an extant path for the SourceFileLoader's resource reader. Avoids the regression identified in master while a long-term solution is devised.
- bpo-40955 [https://bugs.python.org/issue?@action=redirect&bpo=40955]: Fix a minor memory leak in `subprocess` module when extra_groups was specified.
- bpo-40855 [https://bugs.python.org/issue?@action=redirect&bpo=40855]: The standard deviation and variance functions in the statistics module were ignoring their mu and xbar arguments.
- bpo-40939 [https://bugs.python.org/issue?

@action=redirect&bpo=40939]: Use the new PEG parser when generating the stdlib `keyword` module.

- bpo-23427 [https://bugs.python.org/issue?@action=redirect&bpo=23427]: Add `sys.orig_argv` attribute: the list of the original command line arguments passed to the Python executable.

- bpo-33689 [https://bugs.python.org/issue?@action=redirect&bpo=33689]: Ignore empty or whitespace-only lines in .pth files. This matches the documentated behavior. Before, empty lines caused the site-packages dir to appear multiple times in sys.path. By Ido Michael, contributors Malcolm Smith and Tal Einat.

- bpo-40884 [https://bugs.python.org/issue?@action=redirect&bpo=40884]: Added a `defaults` parameter to `logging.Formatter`, to allow specifying default values for custom fields. Patch by Asaf Alon and Bar Harel.

- bpo-40876 [https://bugs.python.org/issue?@action=redirect&bpo=40876]: Clarify error message in the `csv` module.

- bpo-39791 [https://bugs.python.org/issue?@action=redirect&bpo=39791]: Refresh importlib.metadata from importlib_metadata 1.6.1.

- bpo-40807 [https://bugs.python.org/issue?@action=redirect&bpo=40807]: Stop codeop._maybe_compile, used by code.InteractiveInterpreter (and IDLE). from emitting each warning three times.

- bpo-32604 [https://bugs.python.org/issue?@action=redirect&bpo=32604]: Fix reference leak in the `select` module when the module is imported in a subinterpreter.

- bpo-39791 [https://bugs.python.org/issue?@action=redirect&bpo=39791]: Built-in loaders (SourceFileLoader and ZipImporter) now supply `TraversableResources` implementations for `ResourceReader`, and the fallback function has been removed.

- bpo-39314 [https://bugs.python.org/issue?@action=redirect&bpo=39314]: `rlcompleter.Completer` and the standard Python shell now close the parenthesis for functions that take no arguments. Patch contributed by Rémi Lapeyre.

- bpo-17005 [https://bugs.python.org/issue?@action=redirect&bpo=17005]: The topological sort functionality that was introduced initially in the `functools` module has been moved to a new `graphlib` module to better accommodate the new tools and keep the original scope of the `functools` module. Patch by Pablo Galindo

- bpo-40834 [https://bugs.python.org/issue?@action=redirect&bpo=40834]: Fix truncate when sending str object with_xxsubinterpreters.channel_send.

- bpo-40755 [https://bugs.python.org/issue?@action=redirect&bpo=40755]: Add rich comparisons to collections.Counter().

- bpo-26407 [https://bugs.python.org/issue?@action=redirect&bpo=26407]: Unexpected errors in calling the `__iter__` method are no longer masked by `TypeError` in `csv.reader()`, `csv.writer.writerow()` and `csv.writer.writerows()`.

- bpo-39384 [https://bugs.python.org/issue?@action=redirect&bpo=39384]: Fixed email.contentmanager to allow set_content() to set a null string.

- bpo-40744 [https://bugs.python.org/issue?@action=redirect&bpo=40744]: The `sqlite3` module uses SQLite API functions that require SQLite v3.7.3 or higher. This patch removes support for older SQLite versions, and explicitly requires SQLite 3.7.3 both at build, compile and runtime. Patch by Sergey Fedoseev and Erlend E. Aasland.

- bpo-40777 [https://bugs.python.org/issue?

@action=redirect&bpo=40777]: Initialize PyDateTime_IsoCalendarDateType.tp_base at run-time to avoid errors on some compilers.

- bpo-38488 [https://bugs.python.org/issue?@action=redirect&bpo=38488]: Update ensurepip to install pip 20.1.1 and setuptools 47.1.0.

- bpo-40792 [https://bugs.python.org/issue?@action=redirect&bpo=40792]: The result of **`operator.index()`** now always has exact type **`int`**. Previously, the result could have been an instance of a subclass of `int`.

- bpo-40767 [https://bugs.python.org/issue?@action=redirect&bpo=40767]: **`webbrowser`** now properly finds the default browser in pure Wayland systems by checking the WAYLAND_DISPLAY environment variable. Patch contributed by Jérémy Attali.

- bpo-40791 [https://bugs.python.org/issue?@action=redirect&bpo=40791]: **`hashlib.compare_digest()`** uses OpenSSL's `CRYPTO_memcmp()` function when OpenSSL is available.

- bpo-40795 [https://bugs.python.org/issue?@action=redirect&bpo=40795]: **`ctypes`** module: If ctypes fails to convert the result of a callback or if a ctypes callback function raises an exception, sys.unraisablehook is now called with an exception set. Previously, the error was logged into stderr by **`PyErr_Print()`**.

- bpo-16995 [https://bugs.python.org/issue?@action=redirect&bpo=16995]: Add **`base64.b32hexencode()`** and **`base64.b32hexdecode()`** to support the Base32 Encoding with Extended Hex Alphabet.

- bpo-30008 [https://bugs.python.org/issue?@action=redirect&bpo=30008]: Fix **`ssl`** code to be compatible with OpenSSL 1.1.x builds that use `no-deprecated` and `--api=1.1.0`.

- bpo-30064 [https://bugs.python.org/issue?@action=redirect&bpo=30064]: Fix asyncio `loop.sock_*` race condition issue

- bpo-40759 [https://bugs.python.org/issue?@action=redirect&bpo=40759]: Deprecate the **`symbol`** module.

- bpo-40756 [https://bugs.python.org/issue?@action=redirect&bpo=40756]: The second argument (extra) of `LoggerAdapter.__init__` now defaults to None.

- bpo-37129 [https://bugs.python.org/issue?@action=redirect&bpo=37129]: Add a new **`os.RWF_APPEND`** flag for **`os.pwritev()`**.

- bpo-40737 [https://bugs.python.org/issue?@action=redirect&bpo=40737]: Fix possible reference leak for **`sqlite3`** initialization.

- bpo-40726 [https://bugs.python.org/issue?@action=redirect&bpo=40726]: Handle cases where the `end_lineno` is `None` on **`ast.increment_lineno()`**.

- bpo-40698 [https://bugs.python.org/issue?@action=redirect&bpo=40698]: `distutils` upload creates SHA2-256 and Blake2b-256 digests. MD5 digests is skipped if platform blocks MD5.

- bpo-40695 [https://bugs.python.org/issue?@action=redirect&bpo=40695]: **`hashlib`** no longer falls back to builtin hash implementations when OpenSSL provides a hash digest and the algorithm is blocked by security policy.

- bpo-9216 [https://bugs.python.org/issue?@action=redirect&bpo=9216]: **`hashlib.new()`** passed `usedforsecurity` to OpenSSL EVP constructor `_hashlib.new()`. test_hashlib and test_smtplib handle strict security policy better.

- bpo-40614 [https://bugs.python.org/issue?@action=redirect&bpo=40614]: **`ast.parse()`** will not parse self documenting expressions in f-strings when passed `feature_version` is less than `(3, 8)`.

- bpo-40626 [https://bugs.python.org/issue?@action=redirect&bpo=40626]: Add h5 file extension as MIME Type application/x-hdf5, as per HDF Group recommendation for HDF5 formatted data files. Patch contributed by Mark Schwab.

- bpo-25920 [https://bugs.python.org/issue?@action=redirect&bpo=25920]: On macOS, when building Python for macOS 10.4 and older, which wasn't the case for python.org macOS installer, **`socket.getaddrinfo()`** no longer uses an internal lock to prevent race conditions when calling `getaddrinfo()` which is thread-safe since macOS 10.5. Python 3.9 requires macOS 10.6 or newer. The internal lock caused random hang on fork when another thread was calling **`socket.getaddrinfo()`**. The lock was also used on FreeBSD older than 5.3, OpenBSD older than 201311 and NetBSD older than 4.

- bpo-40671 [https://bugs.python.org/issue?@action=redirect&bpo=40671]: Prepare `_hashlib` for **PEP 489** [https://peps.python.org/pep-0489/] and use **`PyModule_AddType()`**.

- bpo-32309 [https://bugs.python.org/issue?@action=redirect&bpo=32309]: Added a new coroutine **`asyncio.to_thread()`**. It is mainly used for running IO-bound functions in a separate thread to avoid blocking the event loop, and essentially works as a high-level version of **`run_in_executor()`** that can directly take keyword arguments.

- bpo-36543 [https://bugs.python.org/issue?@action=redirect&bpo=36543]: Restored the deprecated **`xml.etree.cElementTree`** module.

- bpo-40611 [https://bugs.python.org/issue?@action=redirect&bpo=40611]: **`MAP_POPULATE`** constant has now been added to the list of exported **`mmap`** module flags.

- bpo-39881 [https://bugs.python.org/issue?@action=redirect&bpo=39881]: PEP 554 for use in the test suite. (Patch By Joannah Nanjekye)

- bpo-13097 [https://bugs.python.org/issue?@action=redirect&bpo=13097]: `ctypes` now raises an `ArgumentError` when a callback is invoked with more than 1024 arguments.

- bpo-39385 [https://bugs.python.org/issue?@action=redirect&bpo=39385]: A new test assertion context-manager, **`unittest.assertNoLogs()`** will ensure a given block of code emits no log messages using the logging module. Contributed by Kit Yan Choi.

- bpo-23082 [https://bugs.python.org/issue?@action=redirect&bpo=23082]: Updated the error message and docs of PurePath.relative_to() to better reflect the function behaviour.

- bpo-40318 [https://bugs.python.org/issue?@action=redirect&bpo=40318]: Use SQLite3 trace v2 API, if it is available.

- bpo-40105 [https://bugs.python.org/issue?@action=redirect&bpo=40105]: ZipFile truncates files to avoid corruption when a shorter comment is provided in append («a») mode. Patch by Jan Mazur.

- bpo-40084 [https://bugs.python.org/issue?@action=redirect&bpo=40084]: Fix `Enum.__dir__`: dir(Enum.member) now includes attributes as well as methods.

- bpo-31122 [https://bugs.python.org/issue?@action=redirect&bpo=31122]: ssl.wrap_socket() now raises ssl.SSLEOFError rather than OSError when peer closes connection during TLS negotiation

- bpo-39728 [https://bugs.python.org/issue?@action=redirect&bpo=39728]: fix default `_missing_` so a duplicate `ValueError` is not set as the `__context__` of the original `ValueError`.

- bpo-39244 [https://bugs.python.org/issue?@action=redirect&bpo=39244]: Fixed **`multiprocessing.context.get_all_start_methods`** to properly return the default method first on macOS.

- bpo-39040 [https://bugs.python.org/issue?@action=redirect&bpo=39040]: Fix parsing of invalid mime headers parameters by collapsing whitespace between encoded words in a bare-quote-string.

- bpo-38731 [https://bugs.python.org/issue?@action=redirect&bpo=38731]: Add `--quiet` option to command-line interface of **`py_compile`**. Patch by Gregory Schevchenko.

- bpo-35714 [https://bugs.python.org/issue?@action=redirect&bpo=35714]: **`struct.error`** is now raised if there is a null character in a **`struct`** format string.

- bpo-38144 [https://bugs.python.org/issue?@action=redirect&bpo=38144]: Added the *root_dir* and *dir_fd* parameters in **`glob.glob()`**.

- bpo-26543 [https://bugs.python.org/issue?@action=redirect&bpo=26543]: Fix **`IMAP4.noop()`** when debug mode is enabled (ex: `imaplib.Debug = 3`).

- bpo-12178 [https://bugs.python.org/issue?@action=redirect&bpo=12178]: **`csv.writer()`** now correctly escapes *escapechar* when input contains *escapechar*. Patch by Catalin Iacob, Berker Peksag, and Itay Elbirt.

- bpo-36290 [https://bugs.python.org/issue?@action=redirect&bpo=36290]: AST nodes are now raising **`TypeError`** on conflicting keyword arguments. Patch contributed by Rémi Lapeyre.

- bpo-33944 [https://bugs.python.org/issue?

@action=redirect&bpo=33944]: Added site.py site-packages tracing in verbose mode.

- bpo-35078 [https://bugs.python.org/issue?@action=redirect&bpo=35078]: Refactor formatweekday, formatmonthname methods in LocaleHTMLCalendar and LocaleTextCalendar classes in calendar module to call the base class methods.This enables customizable CSS classes for LocaleHTMLCalendar. Patch by Srinivas Reddy Thatiparthy
- bpo-29620 [https://bugs.python.org/issue?@action=redirect&bpo=29620]: **`assertWarns()`** no longer raises a `RuntimeException` when accessing a module's `__warningregistry__` causes importation of a new module, or when a new module is imported in another thread. Patch by Kernc.
- bpo-31844 [https://bugs.python.org/issue?@action=redirect&bpo=31844]: Remove `ParserBase.error()` method from the private and undocumented `_markupbase` module. **`html.parser.HTMLParser`** is the only subclass of `ParserBase` and its `error()` implementation was deprecated in Python 3.4 and removed in Python 3.5.
- bpo-34226 [https://bugs.python.org/issue?@action=redirect&bpo=34226]: Fix `cgi.parse_multipart` without content_length. Patch by Roger Duran
- bpo-33660 [https://bugs.python.org/issue?@action=redirect&bpo=33660]: Fix pathlib.PosixPath to resolve a relative path located on the root directory properly.
- bpo-28557 [https://bugs.python.org/issue?@action=redirect&bpo=28557]: Improve the error message for a misbehaving `rawio.readinto`
- bpo-26680 [https://bugs.python.org/issue?@action=redirect&bpo=26680]: The d.is_integer() method is added to the Decimal type, for compatibility with other number types.

- bpo-26680 [https://bugs.python.org/issue?@action=redirect&bpo=26680]: The x.is_integer() method is incorporated into the abstract types of the numeric tower, Real, Rational and Integral, with appropriate default implementations.

Documentation

- bpo-41428 [https://bugs.python.org/issue?@action=redirect&bpo=41428]: Add documentation for **PEP 604** [https://peps.python.org/pep-0604/] (Allow writing union types as `X | Y`).
- bpo-41774 [https://bugs.python.org/issue?@action=redirect&bpo=41774]: In Programming FAQ «Sequences (Tuples/Lists)» section, add «How do you remove multiple items from a list».
- bpo-35293 [https://bugs.python.org/issue?@action=redirect&bpo=35293]: Fix RemovedInSphinx40Warning when building the documentation. Patch by Donghee Na.
- bpo-37149 [https://bugs.python.org/issue?@action=redirect&bpo=37149]: Change Shipman tkinter doc link from archive.org to TkDocs. (The doc has been removed from the NMT server.) The new link responds much faster and includes a short explanatory note.
- bpo-41726 [https://bugs.python.org/issue?@action=redirect&bpo=41726]: Update the refcounts info of `PyType_FromModuleAndSpec`.
- bpo-41624 [https://bugs.python.org/issue?@action=redirect&bpo=41624]: Fix the signature of **`typing.Coroutine`**.
- bpo-40204 [https://bugs.python.org/issue?@action=redirect&bpo=40204]: Enable Sphinx 3.2 `c_allow_pre_v3` option and disable `c_warn_on_allowed_pre_v3` option to make the documentation compatible with Sphinx 2 and Sphinx 3.
- bpo-41045 [https://bugs.python.org/issue?@action=redirect&bpo=41045]: Add documentation for debug feature of f-strings.
- bpo-41314 [https://bugs.python.org/issue?@action=redirect&bpo=41314]: Changed the release when `from`

`__future__ import annotations` becomes the default from `4.0` to `3.10` (following a change in PEP 563).

- bpo-40979 [https://bugs.python.org/issue?@action=redirect&bpo=40979]: Refactored typing.rst, arranging more than 70 classes, functions, and decorators into new sub-sections.
- bpo-40552 [https://bugs.python.org/issue?@action=redirect&bpo=40552]: Fix in tutorial section 4.2. Code snippet is now correct.
- bpo-39883 [https://bugs.python.org/issue?@action=redirect&bpo=39883]: Make code, examples, and recipes in the Python documentation be licensed under the more permissive BSD0 license in addition to the existing Python 2.0 license.
- bpo-37703 [https://bugs.python.org/issue?@action=redirect&bpo=37703]: Updated Documentation to comprehensively elaborate on the behaviour of gather.cancel()

Tests

- bpo-41939 [https://bugs.python.org/issue?@action=redirect&bpo=41939]: Fix test_site.test_license_exists_at_url(): call `urllib.request.urlcleanup()` to reset the global `urllib.request._opener`. Patch by Victor Stinner.
- bpo-41731 [https://bugs.python.org/issue?@action=redirect&bpo=41731]: Make test_cmd_line_script pass with option "-vv".
- bpo-41602 [https://bugs.python.org/issue?@action=redirect&bpo=41602]: Add tests for SIGINT handling in the runpy module.
- bpo-41521 [https://bugs.python.org/issue?@action=redirect&bpo=41521]: **`test.support`**: Rename `blacklist` parameter of **`check__all__()`** to `not_exported`.
- bpo-41477 [https://bugs.python.org/issue?@action=redirect&bpo=41477]: Make ctypes optional in test_genericalias.
- bpo-41085 [https://bugs.python.org/issue?

@action=redirect&bpo=41085]: Fix integer overflow in the **`array.array.index()`** method on 64-bit Windows for index larger than `2**31`.

- bpo-41069 [https://bugs.python.org/issue?@action=redirect&bpo=41069]: **`test.support.TESTFN`** and the current directory for tests when run via `test.regrtest` contain now non-ascii characters if possible.
- bpo-38377 [https://bugs.python.org/issue?@action=redirect&bpo=38377]: On Linux, skip tests using multiprocessing if the current user cannot create a file in `/dev/shm/` directory. Add the **`skip_if_broken_multiprocessing_synchronize()`** function to the **`test.support`** module.
- bpo-41009 [https://bugs.python.org/issue?@action=redirect&bpo=41009]: Fix use of `support.require_{linux|mac|freebsd}_version()` decorators as class decorator.
- bpo-41003 [https://bugs.python.org/issue?@action=redirect&bpo=41003]: Fix `test_copyreg` when `numpy` is installed: `test.pickletester` now saves/restores warnings filters when importing `numpy`, to ignore filters installed by `numpy`.
- bpo-40964 [https://bugs.python.org/issue?@action=redirect&bpo=40964]: Disable remote **`imaplib`** tests, host cyrus.andrew.cmu.edu is blocking incoming connections.
- bpo-40927 [https://bugs.python.org/issue?@action=redirect&bpo=40927]: Fix test_binhex when run twice: it now uses import_fresh_module() to ensure that it raises DeprecationWarning each time.
- bpo-17258 [https://bugs.python.org/issue?@action=redirect&bpo=17258]: Skip some **`multiprocessing`** tests when MD5 hash digest is blocked.
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Increase LOOPBACK_TIMEOUT to 10 for VxWorks RTOS.
- bpo-38169 [https://bugs.python.org/issue?@action=redirect&bpo=38169]: Increase code coverage for SharedMemory and ShareableList
- bpo-34401 [https://bugs.python.org/issue?@action=redirect&bpo=34401]: Make test_gdb properly run on

HP-UX. Patch by Michael Osipov.

Build

- bpo-38249 [https://bugs.python.org/issue?@action=redirect&bpo=38249]: Update `Py_UNREACHABLE` to use _builtin_unreachable() if only the compiler is able to use it. Patch by Donghee Na.
- bpo-41617 [https://bugs.python.org/issue?@action=redirect&bpo=41617]: Fix `pycore_bitutils.h` header file to support old clang versions: `__builtin_bswap16()` is not available in LLVM clang 3.0.
- bpo-40204 [https://bugs.python.org/issue?@action=redirect&bpo=40204]: Pin Sphinx version to 2.3.1 in `Doc/Makefile`.
- bpo-36020 [https://bugs.python.org/issue?@action=redirect&bpo=36020]: The C99 functions **`snprintf()`** and **`vsnprintf()`** are now required to build Python.
- bpo-40684 [https://bugs.python.org/issue?@action=redirect&bpo=40684]: `make install` now uses the `PLATLIBDIR` variable for the destination `lib-dynload/` directory when `./configure --with-platlibdir` is used.
- bpo-40683 [https://bugs.python.org/issue?@action=redirect&bpo=40683]: Fixed an issue where the `zoneinfo` module and its tests were not included when Python is installed with `make`.

Windows

- bpo-41744 [https://bugs.python.org/issue?@action=redirect&bpo=41744]: Fixes automatic import of props file when using the Nuget package.
- bpo-41627 [https://bugs.python.org/issue?@action=redirect&bpo=41627]: The user site directory for 32-bit now includes a `-32` suffix to distinguish it from the 64-bit interpreter's directory.
- bpo-41526 [https://bugs.python.org/issue?@action=redirect&bpo=41526]: Fixed layout of final page of the installer by removing the special thanks to Mark Hammond

(with his permission).

- bpo-41492 [https://bugs.python.org/issue?@action=redirect&bpo=41492]: Fixes the description that appears in UAC prompts.
- bpo-40948 [https://bugs.python.org/issue?@action=redirect&bpo=40948]: Improve post-install message to direct people to the «py» command.
- bpo-41412 [https://bugs.python.org/issue?@action=redirect&bpo=41412]: The installer will now fail to install on Windows 7 and Windows 8. Further, the UCRT dependency is now always downloaded on demand.
- bpo-40741 [https://bugs.python.org/issue?@action=redirect&bpo=40741]: Update Windows release to include SQLite 3.32.3.
- bpo-41142 [https://bugs.python.org/issue?@action=redirect&bpo=41142]: **`msilib`** now supports creating CAB files with non-ASCII file path and adding files with non-ASCII file path to them.
- bpo-41074 [https://bugs.python.org/issue?@action=redirect&bpo=41074]: Fixed support of non-ASCII names in functions **`msilib.OpenDatabase()`** and **`msilib.init_database()`** and non-ASCII SQL in method **`msilib.Database.OpenView()`**.
- bpo-41039 [https://bugs.python.org/issue?@action=redirect&bpo=41039]: Stable ABI redirection DLL (python3.dll) now uses `#pragma comment(linker)` for re-exporting.
- bpo-40164 [https://bugs.python.org/issue?@action=redirect&bpo=40164]: Updates Windows OpenSSL to 1.1.1g
- bpo-39631 [https://bugs.python.org/issue?@action=redirect&bpo=39631]: Changes the registered MIME type for `.py` files on Windows to `text/x-python` instead of `text/plain`.
- bpo-40677 [https://bugs.python.org/issue?@action=redirect&bpo=40677]: Manually define IO_REPARSE_TAG_APPEXECLINK in case some old Windows SDK doesn't have it.
- bpo-37556 [https://bugs.python.org/issue?@action=redirect&bpo=37556]: Extend py.exe help to mention

overrides via venv, shebang, environmental variables & ini files.

macOS

- bpo-41557 [https://bugs.python.org/issue?@action=redirect&bpo=41557]: Update macOS installer to use SQLite 3.33.0.
- bpo-39580 [https://bugs.python.org/issue?@action=redirect&bpo=39580]: Avoid opening Finder window if running installer from the command line. Patch contributed by Rick Heil.
- bpo-41100 [https://bugs.python.org/issue?@action=redirect&bpo=41100]: Fix configure error when building on macOS 11. Note that the current Python release was released shortly after the first developer preview of macOS 11 (Big Sur); there are other known issues with building and running on the developer preview. Big Sur is expected to be fully supported in a future bugfix release of Python 3.8.x and with 3.9.0.
- bpo-40741 [https://bugs.python.org/issue?@action=redirect&bpo=40741]: Update macOS installer to use SQLite 3.32.3.
- bpo-41005 [https://bugs.python.org/issue?@action=redirect&bpo=41005]: fixed an XDG settings issue not allowing macos to open browser in webbrowser.py
- bpo-40741 [https://bugs.python.org/issue?@action=redirect&bpo=40741]: Update macOS installer to use SQLite 3.32.2.

IDLE

- bpo-41775 [https://bugs.python.org/issue?@action=redirect&bpo=41775]: Use “IDLE Shell” as shell title
- bpo-35764 [https://bugs.python.org/issue?@action=redirect&bpo=35764]: Rewrite the Calltips doc section.
- bpo-40181 [https://bugs.python.org/issue?@action=redirect&bpo=40181]: In calltips, stop reminding that “/” marks the end of positional-only arguments.
- bpo-41468 [https://bugs.python.org/issue?

@action=redirect&bpo=41468]: Improve IDLE run crash error message (which users should never see).
- bpo-41373 [https://bugs.python.org/issue?@action=redirect&bpo=41373]: Save files loaded with no line ending, as when blank, or different line endings, by setting its line ending to the system default. Fix regression in 3.8.4 and 3.9.0b4.
- bpo-41300 [https://bugs.python.org/issue?@action=redirect&bpo=41300]: Save files with non-ascii chars. Fix regression released in 3.9.0b4 and 3.8.4.
- bpo-37765 [https://bugs.python.org/issue?@action=redirect&bpo=37765]: Add keywords to module name completion list. Rewrite Completions section of IDLE doc.
- bpo-41152 [https://bugs.python.org/issue?@action=redirect&bpo=41152]: The encoding of `stdin`, `stdout` and `stderr` in IDLE is now always UTF-8.
- bpo-41144 [https://bugs.python.org/issue?@action=redirect&bpo=41144]: Make Open Module open a special module such as os.path.
- bpo-39885 [https://bugs.python.org/issue?@action=redirect&bpo=39885]: Make context menu Cut and Copy work again when right-clicking within a selection.
- bpo-40723 [https://bugs.python.org/issue?@action=redirect&bpo=40723]: Make test_idle pass when run after import.

C API

- bpo-41936 [https://bugs.python.org/issue?@action=redirect&bpo=41936]: Removed undocumented macros `Py_ALLOW_RECURSION` and `Py_END_ALLOW_RECURSION` and the `recursion_critical` field of the **`PyInterpreterState`** structure.
- bpo-41692 [https://bugs.python.org/issue?@action=redirect&bpo=41692]: The `PyUnicode_InternImmortal()` function is now deprecated and will be removed in Python 3.12: use **`PyUnicode_InternInPlace()`** instead. Patch by Victor Stinner.
- bpo-41842 [https://bugs.python.org/issue?

@action=redirect&bpo=41842]: Add **`PyCodec_Unregister()`** function to unregister a codec search function.

- bpo-41834 [https://bugs.python.org/issue?@action=redirect&bpo=41834]: Remove the `_Py_CheckRecursionLimit` variable: it has been replaced by `ceval.recursion_limit` of the **`PyInterpreterState`** structure. Patch by Victor Stinner.
- bpo-41689 [https://bugs.python.org/issue?@action=redirect&bpo=41689]: Types created with **`PyType_FromSpec()`** now make any signature in their `tp_doc` slot accessible from `__text_signature__`.
- bpo-41524 [https://bugs.python.org/issue?@action=redirect&bpo=41524]: Fix bug in **`PyOS_mystrnicmp()`** and **`PyOS_mystricmp()`** that incremented pointers beyond the end of a string.
- bpo-41324 [https://bugs.python.org/issue?@action=redirect&bpo=41324]: Add a minimal decimal capsule API. The API supports fast conversions between Decimals up to 38 digits and their triple representation as a C struct.
- bpo-30155 [https://bugs.python.org/issue?@action=redirect&bpo=30155]: Add **`PyDateTime_DATE_GET_TZINFO()`** and **`PyDateTime_TIME_GET_TZINFO()`** macros for accessing the `tzinfo` attributes of **`datetime.datetime`** and **`datetime.time`** objects.
- bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170]: Revert **`PyType_HasFeature()`** change: it reads again directly the **`PyTypeObject.tp_flags`** member when the limited C API is not used, rather than always calling **`PyType_GetFlags()`** which hides implementation details.
- bpo-41123 [https://bugs.python.org/issue?@action=redirect&bpo=41123]: Remove `PyUnicode_AsUnicodeCopy`.
- bpo-41123 [https://bugs.python.org/issue?@action=redirect&bpo=41123]: Removed `PyLong_FromUnicode()`.
- bpo-41123 [https://bugs.python.org/issue?@action=redirect&bpo=41123]: Removed `PyUnicode_GetMax()`.

- bpo-41123 [https://bugs.python.org/issue?@action=redirect&bpo=41123]: Removed `Py_UNICODE_str*` functions manipulating `Py_UNICODE*` strings.
- bpo-41103 [https://bugs.python.org/issue?@action=redirect&bpo=41103]: `PyObject_AsCharBuffer()`, `PyObject_AsReadBuffer()`, `PyObject_CheckReadBuffer()`, and `PyObject_AsWriteBuffer()` are removed. Please migrate to new buffer protocol; **`PyObject_GetBuffer()`** and **`PyBuffer_Release()`**.
- bpo-36346 [https://bugs.python.org/issue?@action=redirect&bpo=36346]: Raises DeprecationWarning for `PyUnicode_FromUnicode(NULL, size)` and `PyUnicode_FromStringAndSize(NULL, size)` with `size > 0`.
- bpo-36346 [https://bugs.python.org/issue?@action=redirect&bpo=36346]: Mark `Py_UNICODE_COPY`, `Py_UNICODE_FILL`, `PyUnicode_WSTR_LENGTH`, `PyUnicode_FromUnicode`, `PyUnicode_AsUnicode`, and `PyUnicode_AsUnicodeAndSize` as deprecated in C. Remove `Py_UNICODE_MATCH` which was deprecated and broken since Python 3.3.
- bpo-40989 [https://bugs.python.org/issue?@action=redirect&bpo=40989]: The **`PyObject_INIT()`** and **`PyObject_INIT_VAR()`** macros become aliases to, respectively, **`PyObject_Init()`** and **`PyObject_InitVar()`** functions.
- bpo-36020 [https://bugs.python.org/issue?@action=redirect&bpo=36020]: On Windows, `#include "pyerrors.h"` no longer defines `snprintf` and `vsnprintf` macros.
- bpo-40943 [https://bugs.python.org/issue?@action=redirect&bpo=40943]: The `PY_SSIZE_T_CLEAN` macro must now be defined to use **`PyArg_ParseTuple()`** and **`Py_BuildValue()`** formats which use `#`: `es#`, `et#`, `s#`, `u#`, `y#`, `z#`, `U#` and `Z#`. See Parsing arguments and building values and the **PEP 353** [https://peps.python.org/pep-0353/].
- bpo-40910 [https://bugs.python.org/issue?@action=redirect&bpo=40910]: Export explicitly the

`Py_GetArgcArgv()` function to the C API and document the function. Previously, it was exported implicitly which no longer works since Python is built with `-fvisibility=hidden`.

- bpo-40724 [https://bugs.python.org/issue?@action=redirect&bpo=40724]: Allow defining buffer slots in type specs.
- bpo-40679 [https://bugs.python.org/issue?@action=redirect&bpo=40679]: Fix a `_PyEval_EvalCode()` crash if *qualname* argument is NULL.
- bpo-40839 [https://bugs.python.org/issue?@action=redirect&bpo=40839]: Calling `PyDict_GetItem()` without GIL held had been allowed for historical reason. It is no longer allowed.
- bpo-40826 [https://bugs.python.org/issue?@action=redirect&bpo=40826]: `PyOS_InterruptOccurred()` now fails with a fatal error if it is called with the GIL released.
- bpo-40792 [https://bugs.python.org/issue?@action=redirect&bpo=40792]: The result of `PyNumber_Index()` now always has exact type `int`. Previously, the result could have been an instance of a subclass of `int`.
- bpo-39573 [https://bugs.python.org/issue?@action=redirect&bpo=39573]: Convert `Py_REFCNT()` and `Py_SIZE()` macros to static inline functions. They cannot be used as l-value anymore: use `Py_SET_REFCNT()` and `Py_SET_SIZE()` to set an object reference count and size. This change is backward incompatible on purpose, to prepare the C API for an opaque `PyObject` structure.
- bpo-40703 [https://bugs.python.org/issue?@action=redirect&bpo=40703]: The PyType_FromSpec*() functions no longer overwrite the type's «__module__» attribute if it is set via «Py_tp_members» or «Py_tp_getset».
- bpo-39583 [https://bugs.python.org/issue?@action=redirect&bpo=39583]: Remove superfluous «extern C» declarations from `Include/cpython/*.h`.

Python 3.9.0 beta 1

Release date: 2020-05-19

Security

- bpo-40501 [https://bugs.python.org/issue?@action=redirect&bpo=40501]: **`uuid`** no longer uses **`ctypes`** to load `libuuid` or `rpcrt4.dll` at runtime.

Core and Builtins

- bpo-40663 [https://bugs.python.org/issue?@action=redirect&bpo=40663]: Correctly generate annotations where parentheses are omitted but required (e.g: `Type[(str, int, *other))]`.
- bpo-40596 [https://bugs.python.org/issue?@action=redirect&bpo=40596]: Fixed **`str.isidentifier()`** for non-canonicalized strings containing non-BMP characters on Windows.
- bpo-40593 [https://bugs.python.org/issue?@action=redirect&bpo=40593]: Improved syntax errors for invalid characters in source code.
- bpo-40585 [https://bugs.python.org/issue?@action=redirect&bpo=40585]: Fixed a bug when using **`codeop.compile_command()`** that was causing exceptions to be swallowed with the new parser. Patch by Pablo Galindo
- bpo-40566 [https://bugs.python.org/issue?@action=redirect&bpo=40566]: Apply **PEP 573** [https://peps.python.org/pep-0573/] to **`abc`**.
- bpo-40502 [https://bugs.python.org/issue?@action=redirect&bpo=40502]: Initialize `n->n_col_offset`. (Patch by Joannah Nanjekye)
- bpo-40527 [https://bugs.python.org/issue?@action=redirect&bpo=40527]: Fix command line argument parsing: no longer write errors multiple times into stderr.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port **`errno`** to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-40523 [https://bugs.python.org/issue?@action=redirect&bpo=40523]: Add pass-throughs for **`hash()`** and **`reversed()`** to **`weakref.proxy`** objects. Patch by

Pablo Galindo.

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port `syslog` to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-40246 [https://bugs.python.org/issue?@action=redirect&bpo=40246]: Reporting a specialised error message for invalid string prefixes, which was introduced in bpo-40246 [https://bugs.python.org/issue?@action=redirect&bpo=40246], is being reverted due to backwards compatibility concerns for strings that immediately follow a reserved keyword without whitespace between them. Constructs like `bg="#d00" if clear else"#fca"` were failing to parse, which is not an acceptable breakage on such short notice.
- bpo-40417 [https://bugs.python.org/issue?@action=redirect&bpo=40417]: Fix imp module deprecation warning when PyImport_ReloadModule is called. Patch by Robert Rouhani.
- bpo-40408 [https://bugs.python.org/issue?@action=redirect&bpo=40408]: Fixed support of nested type variables in GenericAlias (e.g. `list[list[T]]`).
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _stat module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-29587 [https://bugs.python.org/issue?@action=redirect&bpo=29587]: Enable implicit exception chaining when calling `generator.throw()`.
- bpo-40328 [https://bugs.python.org/issue?@action=redirect&bpo=40328]: Add tools for generating mappings headers for CJKCodecs.
- bpo-40228 [https://bugs.python.org/issue?@action=redirect&bpo=40228]: Setting frame.f_lineno is now robust w.r.t. changes in the source-to-bytecode compiler
- bpo-38880 [https://bugs.python.org/issue?@action=redirect&bpo=38880]: Added the ability to list interpreters associated with channel ends in the internal subinterpreters module.
- bpo-37986 [https://bugs.python.org/issue?@action=redirect&bpo=37986]: Improve performance of `PyLong_FromDouble()` for values that fit into long.

Library

- bpo-40662 [https://bugs.python.org/issue?@action=redirect&bpo=40662]: Fixed `ast.get_source_segment()` for ast nodes that have incomplete location information. Patch by Irit Katriel.
- bpo-40665 [https://bugs.python.org/issue?@action=redirect&bpo=40665]: Convert `bisect` to use Argument Clinic.
- bpo-40536 [https://bugs.python.org/issue?@action=redirect&bpo=40536]: Added the `available_timezones()` function to the `zoneinfo` module. Patch by Paul Ganssle.
- bpo-40645 [https://bugs.python.org/issue?@action=redirect&bpo=40645]: The `hmac.HMAC` exposes internal implementation details. The attributes `digest_cons`, `inner`, and `outer` are deprecated and will be removed in the future.
- bpo-40645 [https://bugs.python.org/issue?@action=redirect&bpo=40645]: The internal module `_hashlib` wraps and exposes OpenSSL's HMAC API. The new code will be used in Python 3.10 after the internal implementation details of the pure Python HMAC module are no longer part of the public API.
- bpo-40637 [https://bugs.python.org/issue?@action=redirect&bpo=40637]: Builtin hash modules can now be disabled or selectively enabled with `configure --with-builtin-hashlib-hashes=sha3,blake1` or `--without-builtin-hashlib-hashes`.
- bpo-37630 [https://bugs.python.org/issue?@action=redirect&bpo=37630]: The `hashlib` module can now use SHA3 hashes and SHAKE XOF from OpenSSL when available.
- bpo-40479 [https://bugs.python.org/issue?@action=redirect&bpo=40479]: The `hashlib` now compiles with OpenSSL 3.0.0-alpha2.
- bpo-40257 [https://bugs.python.org/issue?@action=redirect&bpo=40257]: Revert changes to `inspect.getdoc()`.
- bpo-40607 [https://bugs.python.org/issue?

@action=redirect&bpo=40607]: When cancelling a task due to timeout, **`asyncio.wait_for()`** will now propagate the exception if an error happens during cancellation. Patch by Roman Skurikhin.

- bpo-40612 [https://bugs.python.org/issue?@action=redirect&bpo=40612]: Fix edge cases in SyntaxError formatting. If the offset is <= 0, no caret is printed. If the offset is > line length, the caret is printed pointing just after the last character.
- bpo-40597 [https://bugs.python.org/issue?@action=redirect&bpo=40597]: If text content lines are longer than policy.max_line_length, always use a content-encoding to make sure they are wrapped.
- bpo-40571 [https://bugs.python.org/issue?@action=redirect&bpo=40571]: Added functools.cache() as a simpler, more discoverable way to access the unbounded cache variant of lru_cache(maxsize=None).
- bpo-40503 [https://bugs.python.org/issue?@action=redirect&bpo=40503]: **PEP 615** [https://peps.python.org/pep-0615/], the **`zoneinfo`** module. Adds support for the IANA time zone database.
- bpo-40397 [https://bugs.python.org/issue?@action=redirect&bpo=40397]: Removed attributes `__args__` and `__parameters__` from special generic aliases like `typing.List` (not subscripted).
- bpo-40549 [https://bugs.python.org/issue?@action=redirect&bpo=40549]: Convert posixmodule.c («posix» or «nt» module) to the multiphase initialization (PEP 489).
- bpo-31033 [https://bugs.python.org/issue?@action=redirect&bpo=31033]: Add a `msg` argument to **`Future.cancel()`** and **`Task.cancel()`**.
- bpo-40541 [https://bugs.python.org/issue?@action=redirect&bpo=40541]: Added an optional *counts* parameter to random.sample().
- bpo-40515 [https://bugs.python.org/issue?@action=redirect&bpo=40515]: The **`ssl`** and **`hashlib`** modules now actively check that OpenSSL is build with thread support. Python 3.7.0 made thread support mandatory and no longer works safely with a no-thread builds.
- bpo-31033 [https://bugs.python.org/issue?

@action=redirect&bpo=31033]: When a `asyncio.Task` is cancelled, the exception traceback now chains all the way back to where the task was first interrupted.

- bpo-40504 [https://bugs.python.org/issue?@action=redirect&bpo=40504]: `functools.lru_cache()` objects can now be the targets of weakrefs.
- bpo-40559 [https://bugs.python.org/issue?@action=redirect&bpo=40559]: Fix possible memory leak in the C implementation of `asyncio.Task`.
- bpo-40480 [https://bugs.python.org/issue?@action=redirect&bpo=40480]: `fnmatch.fnmatch()` could take exponential time in the presence of multiple `*` pattern characters. This was repaired by generating more elaborate regular expressions to avoid futile backtracking.
- bpo-40495 [https://bugs.python.org/issue?@action=redirect&bpo=40495]: `compileall` is now able to use hardlinks to prevent duplicates in a case when `.pyc` files for different optimization levels have the same content.
- bpo-40457 [https://bugs.python.org/issue?@action=redirect&bpo=40457]: The ssl module now support OpenSSL builds without TLS 1.0 and 1.1 methods.
- bpo-40355 [https://bugs.python.org/issue?@action=redirect&bpo=40355]: Improve error reporting in `ast.literal_eval()` in the presence of malformed `ast.Dict` nodes instead of silently ignoring any non-conforming elements. Patch by Curtis Bucher.
- bpo-40465 [https://bugs.python.org/issue?@action=redirect&bpo=40465]: Deprecated the optional *random* argument to *random.shuffle()*.
- bpo-40459 [https://bugs.python.org/issue?@action=redirect&bpo=40459]: `platform.win32_ver()` now produces correct *ptype* strings instead of empty strings.
- bpo-39435 [https://bugs.python.org/issue?@action=redirect&bpo=39435]: The first argument of `pickle.loads()` is now positional-only.
- bpo-39305 [https://bugs.python.org/issue?@action=redirect&bpo=39305]: Update **`nntplib`** to merge **`nntplib.NNTP`** and **`nntplib._NNTPBase`**. Patch by Donghee Na.
- bpo-32494 [https://bugs.python.org/issue?

@action=redirect&bpo=32494]: Update **`dbm.gnu`** to use gdbm_count if possible when calling **`len()`**. Patch by Donghee Na.

- bpo-40453 [https://bugs.python.org/issue?@action=redirect&bpo=40453]: Add `isolated=True` keyword-only parameter to `_xxsubinterpreters.create()`. An isolated subinterpreter cannot spawn threads, spawn a child process or call `os.fork()`.
- bpo-40286 [https://bugs.python.org/issue?@action=redirect&bpo=40286]: Remove `_random.Random.randbytes()`: the C implementation of `randbytes()`. Implement the method in Python to ease subclassing: `randbytes()` now directly reuses `getrandbits()`.
- bpo-40394 [https://bugs.python.org/issue?@action=redirect&bpo=40394]: Added default arguments to **`difflib.SequenceMatcher.find_longest_match()`**.
- bpo-39995 [https://bugs.python.org/issue?@action=redirect&bpo=39995]: Fix a race condition in concurrent.futures._ThreadWakeup: access to _ThreadWakeup is now protected with the shutdown lock.
- bpo-30966 [https://bugs.python.org/issue?@action=redirect&bpo=30966]: `Process.shutdown(wait=True)` of **`concurrent.futures`** now closes explicitly the result queue.
- bpo-30966 [https://bugs.python.org/issue?@action=redirect&bpo=30966]: Add a new **`close()`** method to the **`SimpleQueue`** class to explicitly close the queue.
- bpo-39966 [https://bugs.python.org/issue?@action=redirect&bpo=39966]: Revert bpo-25597 [https://bugs.python.org/issue?@action=redirect&bpo=25597]. **`unittest.mock.MagicMock`** with wraps” set uses default return values for magic methods.
- bpo-39791 [https://bugs.python.org/issue?@action=redirect&bpo=39791]: Added `files()` function to importlib.resources with support for subdirectories in package data, matching backport in importlib_resources 1.5.
- bpo-40375 [https://bugs.python.org/issue?@action=redirect&bpo=40375]: **`imaplib.IMAP4.unselect()`**

is added. Patch by Donghee Na.

- bpo-40389 [https://bugs.python.org/issue?@action=redirect&bpo=40389]: `repr()` now returns `typing.Optional[T]` when called for `typing.Union` of two types, one of which is `NoneType`.
- bpo-40291 [https://bugs.python.org/issue?@action=redirect&bpo=40291]: Add support for CAN_J1939 sockets (available on Linux 5.4+)
- bpo-40273 [https://bugs.python.org/issue?@action=redirect&bpo=40273]: **`types.MappingProxyType`** is now reversible.
- bpo-39075 [https://bugs.python.org/issue?@action=redirect&bpo=39075]: The repr for **`types.SimpleNamespace`** is now insertion ordered rather than alphabetical.
- bpo-40192 [https://bugs.python.org/issue?@action=redirect&bpo=40192]: On AIX, **`thread_time()`** is now implemented with `thread_cputime()` which has nanosecond resolution, rather than `clock_gettime(CLOCK_THREAD_CPUTIME_ID)` which has a resolution of 10 milliseconds. Patch by Batuhan Taskaya.
- bpo-40025 [https://bugs.python.org/issue?@action=redirect&bpo=40025]: Raise TypeError when _generate_next_value_ is defined after members. Patch by Ethan Onstott.
- bpo-39058 [https://bugs.python.org/issue?@action=redirect&bpo=39058]: In the argparse module, the repr for Namespace() and other argument holders now displayed in the order attributes were added. Formerly, it displayed in alphabetical order even though argument order is preserved the user visible parts of the module.
- bpo-24416 [https://bugs.python.org/issue?@action=redirect&bpo=24416]: The `isocalendar()` methods of **`datetime.date`** and **`datetime.datetime`** now return a named tuple instead of a **`tuple`**.

Documentation

- bpo-34790 [https://bugs.python.org/issue?@action=redirect&bpo=34790]: Add version of removal for explicit passing of coros to `asyncio.wait()`”s documentation

- bpo-40561 [https://bugs.python.org/issue?@action=redirect&bpo=40561]: Provide docstrings for webbrowser open functions.
- bpo-40499 [https://bugs.python.org/issue?@action=redirect&bpo=40499]: Mention that **`asyncio.wait()`** requires a non-empty set of awaitables.
- bpo-39705 [https://bugs.python.org/issue?@action=redirect&bpo=39705]: Tutorial example for sorted() in the Loop Techniques section is given a better explanation. Also a new example is included to explain sorted()”s basic behavior.
- bpo-39435 [https://bugs.python.org/issue?@action=redirect&bpo=39435]: Fix an incorrect signature for **`pickle.loads()`** in the docs

Tests

- bpo-40055 [https://bugs.python.org/issue?@action=redirect&bpo=40055]: distutils.tests now saves/restores warnings filters to leave them unchanged. Importing tests imports docutils which imports pkg_resources which adds a warnings filter.
- bpo-40436 [https://bugs.python.org/issue?@action=redirect&bpo=40436]: test_gdb and test.pythoninfo now check gdb command exit code.

Build

- bpo-40653 [https://bugs.python.org/issue?@action=redirect&bpo=40653]: Move _dirnameW out of HAVE_SYMLINK to fix a potential compiling issue.
- bpo-40514 [https://bugs.python.org/issue?@action=redirect&bpo=40514]: Add `--with-experimental-isolated-subinterpreters` build option to `configure`: better isolate subinterpreters, experimental build mode.

Windows

- bpo-40650 [https://bugs.python.org/issue?@action=redirect&bpo=40650]: Include winsock2.h in pytime.c for timeval.
- bpo-40458 [https://bugs.python.org/issue?@action=redirect&bpo=40458]: Increase reserved stack space to prevent overflow crash on Windows.
- bpo-39148 [https://bugs.python.org/issue?@action=redirect&bpo=39148]: Add IPv6 support to `asyncio` datagram endpoints in ProactorEventLoop. Change the raised exception for unknown address families to ValueError as it's not coming from Windows API.

macOS

- bpo-34956 [https://bugs.python.org/issue?@action=redirect&bpo=34956]: When building Python on macOS from source, `_tkinter` now links with non-system Tcl and Tk frameworks if they are installed in `/Library/Frameworks`, as had been the case on older releases of macOS. If a macOS SDK is explicitly configured, by using `--enable-universalsdk=` or `-isysroot`, only the SDK itself is searched. The default behavior can still be overridden with `--with-tcltk-includes` and `--with-tcltk-libs`.
- bpo-35569 [https://bugs.python.org/issue?@action=redirect&bpo=35569]: Expose RFC 3542 IPv6 socket options.

Tools/Demos

- bpo-40479 [https://bugs.python.org/issue?@action=redirect&bpo=40479]: Update multissltest helper to test with latest OpenSSL 1.0.2, 1.1.0, 1.1.1, and 3.0.0-alpha.
- bpo-40431 [https://bugs.python.org/issue?@action=redirect&bpo=40431]: Fix a syntax typo in `turtledemo` that now raises a `SyntaxError`.
- bpo-40163 [https://bugs.python.org/issue?@action=redirect&bpo=40163]: Fix multissltest tool. OpenSSL has changed download URL for old releases. The multissltest tool now tries to download from current and old download URLs.

C API

- bpo-39465 [https://bugs.python.org/issue?@action=redirect&bpo=39465]: Remove the `_PyUnicode_ClearStaticStrings()` function from the C API.
- bpo-38787 [https://bugs.python.org/issue?@action=redirect&bpo=38787]: Add PyCFunction_CheckExact() macro for exact type checks now that we allow subtypes of PyCFunction, as well as PyCMethod_CheckExact() and PyCMethod_Check() for the new PyCMethod subtype.
- bpo-40545 [https://bugs.python.org/issue?@action=redirect&bpo=40545]: Declare `_PyErr_GetTopmostException()` with `PyAPI_FUNC()` to properly export the function in the C API. The function remains private (`_Py`) prefix.
- bpo-40412 [https://bugs.python.org/issue?@action=redirect&bpo=40412]: Nullify inittab_copy during finalization, preventing future interpreter initializations in an embedded situation from crashing. Patch by Gregory Szorc.
- bpo-40429 [https://bugs.python.org/issue?@action=redirect&bpo=40429]: The **`PyThreadState_GetFrame()`** function now returns a strong reference to the frame.
- bpo-40428 [https://bugs.python.org/issue?@action=redirect&bpo=40428]: Remove the following functions from the C API. Call **`PyGC_Collect()`** explicitly to free all free lists.
    - `PyAsyncGen_ClearFreeLists()`
    - `PyContext_ClearFreeList()`
    - `PyDict_ClearFreeList()`
    - `PyFloat_ClearFreeList()`
    - `PyFrame_ClearFreeList()`
    - `PyList_ClearFreeList()`
    - `PySet_ClearFreeList()`
    - `PyTuple_ClearFreeList()`
- bpo-40421 [https://bugs.python.org/issue?@action=redirect&bpo=40421]: New **`PyFrame_GetBack()`** function: get the frame next outer frame.
- bpo-40421 [https://bugs.python.org/issue?

@action=redirect&bpo=40421]: New `PyFrame_GetCode()` function: return a borrowed reference to the frame code.
- bpo-40217 [https://bugs.python.org/issue?@action=redirect&bpo=40217]: Ensure that instances of types created with `PyType_FromSpecWithBases()` will visit its class object when traversing references in the garbage collector (implemented as an extension of the provided `tp_traverse`). Patch by Pablo Galindo.
- bpo-38787 [https://bugs.python.org/issue?@action=redirect&bpo=38787]: Module C state is now accessible from C-defined heap type methods (**PEP 573** [https://peps.python.org/pep-0573/]). Patch by Marcel Plch and Petr Viktorin.

Python 3.9.0 alpha 6

Release date: 2020-04-27

Security

- bpo-40121 [https://bugs.python.org/issue?@action=redirect&bpo=40121]: Fixes audit events raised on creating a new socket.
- bpo-39073 [https://bugs.python.org/issue?@action=redirect&bpo=39073]: Disallow CR or LF in email.headerregistry.Address arguments to guard against header injection attacks.
- bpo-39503 [https://bugs.python.org/issue?@action=redirect&bpo=39503]: **CVE 2020-8492** [https://www.cve.org/CVERecord?id=CVE-2020-8492]: The `AbstractBasicAuthHandler` class of the `urllib.request` module uses an inefficient regular expression which can be exploited by an attacker to cause a denial of service. Fix the regex to prevent the catastrophic backtracking. Vulnerability reported by Ben Caller and Matt Schwager.

Core and Builtins

- bpo-40313 [https://bugs.python.org/issue?@action=redirect&bpo=40313]: Improve the performance of bytes.hex().
- bpo-40334 [https://bugs.python.org/issue?@action=redirect&bpo=40334]: Switch to a new parser, based on PEG. For more details see PEP 617. To temporarily switch back to the old parser, use `-X oldparser` or `PYTHONOLDPARSER=1`. In Python 3.10 we will remove the old parser completely, including the `parser` module (already deprecated) and anything that depends on it.
- bpo-40267 [https://bugs.python.org/issue?@action=redirect&bpo=40267]: Fix the tokenizer to display the correct error message, when there is a `SyntaxError` on the last input character and no newline follows. It used to be `unexpected EOF while parsing`, while it should be `invalid syntax`.
- bpo-39522 [https://bugs.python.org/issue?@action=redirect&bpo=39522]: Correctly unparse explicit `u` prefix for strings when postponed evaluation for annotations activated. Patch by Batuhan Taskaya.
- bpo-40246 [https://bugs.python.org/issue?@action=redirect&bpo=40246]: Report a specialized error message, `invalid string prefix`, when the tokenizer encounters a string with an invalid prefix.
- bpo-40082 [https://bugs.python.org/issue?@action=redirect&bpo=40082]: Fix the signal handler: it now always uses the main interpreter, rather than trying to get the current Python thread state.
- bpo-37388 [https://bugs.python.org/issue?@action=redirect&bpo=37388]: str.encode() and str.decode() no longer check the encoding and errors in development mode or in debug mode during Python finalization. The codecs machinery can no longer work on very late calls to str.encode() and str.decode().
- bpo-40077 [https://bugs.python.org/issue?@action=redirect&bpo=40077]: Fix possible refleaks in **`_json`**, memo of PyScannerObject should be traversed.
- bpo-37207 [https://bugs.python.org/issue?@action=redirect&bpo=37207]: Speed up calls to `dict()` by using the **PEP 590** [https://peps.python.org/pep-0590/]

`vectorcall` calling convention.

- bpo-40141 [https://bugs.python.org/issue?@action=redirect&bpo=40141]: Add column and line information to `ast.keyword` nodes. Patch by Pablo Galindo.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port **`resource`** to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port **`math`** to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _uuid module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-40077 [https://bugs.python.org/issue?@action=redirect&bpo=40077]: Convert json module to use **`PyType_FromSpec()`**.
- bpo-40067 [https://bugs.python.org/issue?@action=redirect&bpo=40067]: Improve the error message for multiple star expressions in an assignment. Patch by Furkan Onder
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _functools module to multiphase initialization (PEP 489). Patch by Paulo Henrique Silva.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port operator module to multiphase initialization (PEP 489). Patch by Paulo Henrique Silva.
- bpo-20526 [https://bugs.python.org/issue?@action=redirect&bpo=20526]: Fix **`PyThreadState_Clear()`**. `PyThreadState.frame` is a borrowed reference, not a strong reference: `PyThreadState_Clear()` must not call `Py_CLEAR(tstate->frame)`.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port time module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]). Patch by Paulo Henrique Silva.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _weakref extension module

to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-40020 [https://bugs.python.org/issue?@action=redirect&bpo=40020]: Fix a leak and subsequent crash in parsetok.c caused by realloc misuse on a rare codepath.
- bpo-39939 [https://bugs.python.org/issue?@action=redirect&bpo=39939]: Added str.removeprefix and str.removesuffix methods and corresponding bytes, bytearray, and collections.UserString methods to remove affixes from a string if present. See **PEP 616** [https://peps.python.org/pep-0616/] for a full description. Patch by Dennis Sweeney.
- bpo-39481 [https://bugs.python.org/issue?@action=redirect&bpo=39481]: Implement PEP 585. This supports list[int], tuple[str, …] etc.
- bpo-32894 [https://bugs.python.org/issue?@action=redirect&bpo=32894]: Support unparsing of infinity numbers in postponed annotations. Patch by Batuhan Taşkaya.
- bpo-37207 [https://bugs.python.org/issue?@action=redirect&bpo=37207]: Speed up calls to `list()` by using the **PEP 590** [https://peps.python.org/pep-0590/] `vectorcall` calling convention. Patch by Mark Shannon.

Library

- bpo-40398 [https://bugs.python.org/issue?@action=redirect&bpo=40398]: **`typing.get_args()`** now always returns an empty tuple for special generic aliases.

- bpo-40396 [https://bugs.python.org/issue?@action=redirect&bpo=40396]: Functions **`typing.get_origin()`**, **`typing.get_args()`** and **`typing.get_type_hints()`** support now generic aliases like `list[int]`.

- bpo-38061 [https://bugs.python.org/issue?@action=redirect&bpo=38061]: Optimize the **`subprocess`** module on FreeBSD using `closefrom()`. A single `close(fd)` syscall is cheap, but when `sysconf(_SC_OPEN_MAX)` is high, the loop calling `close(fd)` on each file descriptor can take several milliseconds.

The workaround on FreeBSD to improve performance was to load and mount the fdescfs kernel module, but this is not enabled by default.

Initial patch by Ed Maste (emaste), Conrad Meyer (cem), Kyle Evans (kevans) and Kubilay Kocak (koobs): https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=242274

- bpo-38061 [https://bugs.python.org/issue?@action=redirect&bpo=38061]: On FreeBSD, `os.closerange(fd_low, fd_high)` now calls `closefrom(fd_low)` if *fd_high* is greater than or equal to `sysconf(_SC_OPEN_MAX)`.

  Initial patch by Ed Maste (emaste), Conrad Meyer (cem), Kyle Evans (kevans) and Kubilay Kocak (koobs): https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=242274

- bpo-40360 [https://bugs.python.org/issue?@action=redirect&bpo=40360]: The **`lib2to3`** module is pending deprecation due to **PEP 617** [https://peps.python.org/pep-0617/].

- bpo-40138 [https://bugs.python.org/issue?@action=redirect&bpo=40138]: Fix the Windows implementation of **`os.waitpid()`** for exit code larger than `INT_MAX >> 8`. The exit status is now interpreted as an unsigned number.

- bpo-39942 [https://bugs.python.org/issue?@action=redirect&bpo=39942]: Set «_main_» as the default module name when «_name_» is missing in **`typing.TypeVar`**. Patch by Weipeng Hong.

- bpo-40275 [https://bugs.python.org/issue?@action=redirect&bpo=40275]: The **`logging`** package is now imported lazily in **`unittest`** only when the **`assertLogs()`** assertion is used.

- bpo-40275 [https://bugs.python.org/issue?

@action=redirect&bpo=40275]: The `asyncio` package is now imported lazily in `unittest` only when the `IsolatedAsyncioTestCase` class is used.

- bpo-40330 [https://bugs.python.org/issue?@action=redirect&bpo=40330]: In **`ShareableList.__setitem__()`**, check the size of a new string item after encoding it to utf-8, not before.

- bpo-40148 [https://bugs.python.org/issue?@action=redirect&bpo=40148]: Added **`pathlib.Path.with_stem()`** to create a new Path with the stem replaced.

- bpo-40325 [https://bugs.python.org/issue?@action=redirect&bpo=40325]: Deprecated support for set objects in random.sample().

- bpo-40257 [https://bugs.python.org/issue?@action=redirect&bpo=40257]: Improved help for the `typing` module. Docstrings are now shown for all special forms and special generic aliases (like `Union` and `List`). Using `help()` with generic alias like `List[int]` will show the help for the correspondent concrete type (`list` in this case).

- bpo-40257 [https://bugs.python.org/issue?@action=redirect&bpo=40257]: `inspect.getdoc()` no longer returns docstring inherited from the type of the object or from parent class if it is a class if it is not defined in the object itself. In `pydoc` the documentation string is now shown not only for class, function, method etc, but for any object that has its own `__doc__` attribute.

- bpo-40287 [https://bugs.python.org/issue?@action=redirect&bpo=40287]: Fixed `SpooledTemporaryFile.seek()` to return the position.

- bpo-40290 [https://bugs.python.org/issue?@action=redirect&bpo=40290]: Added zscore() to statistics.NormalDist().

- bpo-40282 [https://bugs.python.org/issue?@action=redirect&bpo=40282]: Allow `random.getrandbits(0)` to succeed and to return 0.
- bpo-40286 [https://bugs.python.org/issue?@action=redirect&bpo=40286]: Add **`random.randbytes()`** function and **`random.Random.randbytes()`** method to generate random bytes.
- bpo-40277 [https://bugs.python.org/issue?@action=redirect&bpo=40277]: **`collections.namedtuple()`** now provides a human-readable repr for its field accessors.
- bpo-40270 [https://bugs.python.org/issue?@action=redirect&bpo=40270]: The included copy of sqlite3 on Windows is now compiled with the json extension. This allows the use of functions such as `json_object`.
- bpo-29255 [https://bugs.python.org/issue?@action=redirect&bpo=29255]: Wait in `KqueueSelector.select` when no fds are registered
- bpo-40260 [https://bugs.python.org/issue?@action=redirect&bpo=40260]: Ensure **`modulefinder`** uses **`io.open_code()`** and respects coding comments.
- bpo-40234 [https://bugs.python.org/issue?@action=redirect&bpo=40234]: Allow again to spawn daemon threads in subinterpreters (revert change which denied them).
- bpo-39207 [https://bugs.python.org/issue?@action=redirect&bpo=39207]: Workers in **`ProcessPoolExecutor`** are now spawned on demand, only when there are no available idle workers to reuse. This optimizes startup overhead and reduces the amount of lost CPU time to idle workers. Patch by Kyle Stanley.
- bpo-40091 [https://bugs.python.org/issue?@action=redirect&bpo=40091]: Fix a hang at fork in the logging module: the new private _at_fork_reinit() method is now used to reinitialize locks at fork in the child process.

- bpo-40149 [https://bugs.python.org/issue?@action=redirect&bpo=40149]: Implement traverse and clear slots in _abc._abc_data type.

- bpo-40208 [https://bugs.python.org/issue?@action=redirect&bpo=40208]: Remove deprecated **`symtable.SymbolTable.has_exec()`**.

- bpo-40196 [https://bugs.python.org/issue?@action=redirect&bpo=40196]: Fix a bug in the **`symtable`** module that was causing incorrectly report global variables as local. Patch by Pablo Galindo.

- bpo-40190 [https://bugs.python.org/issue?@action=redirect&bpo=40190]: Add support for `_SC_AIX_REALMEM` to **`posix.sysconf()`**.

- bpo-40182 [https://bugs.python.org/issue?@action=redirect&bpo=40182]: Removed the `_field_types` attribute of the **`typing.NamedTuple`** class.

- bpo-36517 [https://bugs.python.org/issue?@action=redirect&bpo=36517]: Multiple inheritance with **`typing.NamedTuple`** now raises an error instead of silently ignoring other types.

- bpo-40126 [https://bugs.python.org/issue?@action=redirect&bpo=40126]: Fixed reverting multiple patches in unittest.mock. Patcher's `__exit__()` is now never called if its `__enter__()` is failed. Returning true from `__exit__()` silences now the exception.

- bpo-40094 [https://bugs.python.org/issue?@action=redirect&bpo=40094]: CGIHTTPRequestHandler of http.server now logs the CGI script exit code, rather than the CGI script exit status of os.waitpid(). For example, if the script is killed by signal 11, it now logs: «CGI script exit code -11.»

- bpo-40108 [https://bugs.python.org/issue?@action=redirect&bpo=40108]: Improve the error message when triying to import a module using `runpy` and incorrectly using the «.py» extension at the end of the module name. Patch by Pablo Galindo.

- bpo-40094 [https://bugs.python.org/issue?@action=redirect&bpo=40094]: Add `os.waitstatus_to_exitcode()` function: convert a wait status to an exit code.

- bpo-40089 [https://bugs.python.org/issue?@action=redirect&bpo=40089]: Fix threading._after_fork(): if fork was not called by a thread spawned by threading.Thread, threading._after_fork() now creates a _MainThread instance for _main_thread, instead of a _DummyThread instance.

- bpo-40089 [https://bugs.python.org/issue?@action=redirect&bpo=40089]: Add a private `_at_fork_reinit()` method to **`_thread.Lock`**, **`_thread.RLock`**, `threading.RLock` and `threading.Condition` classes: reinitialize the lock at fork in the child process, reset the lock to the unlocked state. Rename also the private `_reset_internal_locks()` method of `threading.Event` to `_at_fork_reinit()`.

- bpo-25780 [https://bugs.python.org/issue?@action=redirect&bpo=25780]: Expose `CAN_RAW_JOIN_FILTERS` in the `socket` module.

- bpo-39503 [https://bugs.python.org/issue?@action=redirect&bpo=39503]: `AbstractBasicAuthHandler` of `urllib.request` now parses all WWW-Authenticate HTTP headers and accepts multiple challenges per header: use the realm of the first Basic challenge.

- bpo-39812 [https://bugs.python.org/issue?@action=redirect&bpo=39812]: Removed daemon threads from `concurrent.futures` by adding an internal `threading._register_atexit()`, which calls registered functions prior to joining all non-daemon threads. This allows for compatibility with subinterpreters, which don’t support daemon threads.

- bpo-40050 [https://bugs.python.org/issue?@action=redirect&bpo=40050]: Fix `importlib._bootstrap_external`: avoid creating a new `winreg` builtin module if it's already available in **`sys.modules`**, and remove redundant imports.

- bpo-40014 [https://bugs.python.org/issue?@action=redirect&bpo=40014]: Fix `os.getgrouplist()`: if `getgrouplist()` function fails because the group list is too small, retry with a larger group list. On failure, the glibc implementation of `getgrouplist()` sets `ngroups` to the total number of groups. For other implementations, double the group list size.

- bpo-40017 [https://bugs.python.org/issue?@action=redirect&bpo=40017]: Add **`time.CLOCK_TAI`** constant if the operating system support it.

- bpo-40016 [https://bugs.python.org/issue?@action=redirect&bpo=40016]: In re docstring, clarify the relationship between inline and argument compile flags.

- bpo-39953 [https://bugs.python.org/issue?@action=redirect&bpo=39953]: Update internal table of OpenSSL error codes in the `ssl` module.

- bpo-36144 [https://bugs.python.org/issue?@action=redirect&bpo=36144]: Added **PEP 584** [https://peps.python.org/pep-0584/] operators to **`weakref.WeakValueDictionary`**.

- bpo-36144 [https://bugs.python.org/issue?@action=redirect&bpo=36144]: Added **PEP 584** [https://peps.python.org/pep-0584/] operators to **`weakref.WeakKeyDictionary`**.

- bpo-38891 [https://bugs.python.org/issue?@action=redirect&bpo=38891]: Fix linear runtime behaviour of the `__getitem__` and `__setitem__` methods in **`multiprocessing.shared_memory.ShareableList`**. This avoids quadratic performance when iterating a

`ShareableList`. Patch by Thomas Krennwallner.

- bpo-39682 [https://bugs.python.org/issue?@action=redirect&bpo=39682]: Remove undocumented support for *closing* a `pathlib.Path` object via its context manager. The context manager magic methods remain, but they are now a no-op, making `Path` objects immutable.

- bpo-36144 [https://bugs.python.org/issue?@action=redirect&bpo=36144]: Added **PEP 584** [https://peps.python.org/pep-0584/] operators (`|` and `|=`) to **`collections.ChainMap`**.

- bpo-39011 [https://bugs.python.org/issue?@action=redirect&bpo=39011]: Normalization of line endings in ElementTree attributes was removed, as line endings which were replaced by entity numbers should be preserved in original form.

- bpo-38410 [https://bugs.python.org/issue?@action=redirect&bpo=38410]: Properly handle **`sys.audit()`** failures in **`sys.set_asyncgen_hooks()`**.

- bpo-36541 [https://bugs.python.org/issue?@action=redirect&bpo=36541]: lib2to3 now recognizes named assignment expressions (the walrus operator, `:=`)

- bpo-35967 [https://bugs.python.org/issue?@action=redirect&bpo=35967]: In platform, delay the invocation of “uname -p” until the processor attribute is requested.

- bpo-35113 [https://bugs.python.org/issue?@action=redirect&bpo=35113]: **`inspect.getsource()`** now returns correct source code for inner class with same name as module level class. Decorators are also returned as part of source of the class. Patch by Karthikeyan Singaravelan.

- bpo-33262 [https://bugs.python.org/issue?@action=redirect&bpo=33262]: Deprecate passing None as an argument for **`shlex.split()`**”s `s` parameter. Patch by Zackery Spytz.

- bpo-31758 [https://bugs.python.org/issue?@action=redirect&bpo=31758]: Prevent crashes when using an uninitialized `_elementtree.XMLParser` object. Patch by Oren Milman.

Documentation

- bpo-27635 [https://bugs.python.org/issue?@action=redirect&bpo=27635]: The pickle documentation incorrectly claimed that `__new__` isn't called by default when unpickling.
- bpo-39879 [https://bugs.python.org/issue?@action=redirect&bpo=39879]: Updated Modelo de datos docs to include **`dict()`** insertion order preservation. Patch by Furkan Onder and Samy Lahfa.
- bpo-38387 [https://bugs.python.org/issue?@action=redirect&bpo=38387]: Document **`PyDoc_STRVAR`** macro in the C-API reference.
- bpo-13743 [https://bugs.python.org/issue?@action=redirect&bpo=13743]: Some methods within xml.dom.minidom.Element class are now better documented.

Tests

- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Set expected default encoding in test_c_locale_coercion.py for VxWorks RTOS.
- bpo-40162 [https://bugs.python.org/issue?@action=redirect&bpo=40162]: Update Travis CI configuration to OpenSSL 1.1.1f.
- bpo-40146 [https://bugs.python.org/issue?@action=redirect&bpo=40146]: Update OpenSSL to 1.1.1f in Azure Pipelines.
- bpo-40094 [https://bugs.python.org/issue?@action=redirect&bpo=40094]: Add **`test.support.wait_process()`** function.
- bpo-40003 [https://bugs.python.org/issue?@action=redirect&bpo=40003]: `test.bisect_cmd` now copies Python command line options like `-O` or `-W`. Moreover, emit a warning if `test.bisect_cmd` is used with `-w/--`

`verbose2` option.

- bpo-39380 [https://bugs.python.org/issue?@action=redirect&bpo=39380]: Add the encoding in **`ftplib.FTP`** and **`ftplib.FTP_TLS`** to the constructor as keyword-only and change the default from `latin-1` to `utf-8` to follow **RFC 2640** [https://datatracker.ietf.org/doc/html/rfc2640.html].
- bpo-39793 [https://bugs.python.org/issue?@action=redirect&bpo=39793]: Use the same domain when testing `make_msgid`. Patch by Batuhan Taskaya.
- bpo-1812 [https://bugs.python.org/issue?@action=redirect&bpo=1812]: Fix newline handling in doctest.testfile when loading from a package whose loader has a get_data method. Patch by Peter Donis.

Build

- bpo-38360 [https://bugs.python.org/issue?@action=redirect&bpo=38360]: Support single-argument form of macOS -isysroot flag.
- bpo-40158 [https://bugs.python.org/issue?@action=redirect&bpo=40158]: Fix CPython MSBuild Properties in NuGet Package (build/native/python.props)
- bpo-38527 [https://bugs.python.org/issue?@action=redirect&bpo=38527]: Fix configure check on Solaris for «float word ordering»: sometimes, the correct «grep» command was not being used. Patch by Arnon Yaari.

Windows

- bpo-40164 [https://bugs.python.org/issue?@action=redirect&bpo=40164]: Updates Windows to OpenSSL 1.1.1f
- bpo-8901 [https://bugs.python.org/issue?@action=redirect&bpo=8901]: Ignore the Windows registry when the `-E` option is used.

macOS

- bpo-38329 [https://bugs.python.org/issue?@action=redirect&bpo=38329]: python.org macOS installers now

update the Current version symlink of /Library/Frameworks/Python.framework/Versions for 3.9 installs. Previously, Current was only updated for Python 2.x installs. This should make it easier to embed Python 3 into other macOS applications.
- bpo-40164 [https://bugs.python.org/issue?@action=redirect&bpo=40164]: Update macOS installer builds to use OpenSSL 1.1.1g.

IDLE

- bpo-38439 [https://bugs.python.org/issue?@action=redirect&bpo=38439]: Add a 256 × 256 pixel IDLE icon to support more modern environments. Created by Andrew Clover. Delete the unused macOS idle.icns icon file.
- bpo-38689 [https://bugs.python.org/issue?@action=redirect&bpo=38689]: IDLE will no longer freeze when inspect.signature fails when fetching a calltip.

Tools/Demos

- bpo-40385 [https://bugs.python.org/issue?@action=redirect&bpo=40385]: Removed the checkpyc.py tool. Please see compileall without force mode as a potential alternative.
- bpo-40179 [https://bugs.python.org/issue?@action=redirect&bpo=40179]: Fixed translation of `#elif` in Argument Clinic.
- bpo-40094 [https://bugs.python.org/issue?@action=redirect&bpo=40094]: Fix `which.py` script exit code: it now uses **`os.waitstatus_to_exitcode()`** to convert **`os.system()`** exit status into an exit code.

C API

- bpo-40241 [https://bugs.python.org/issue?@action=redirect&bpo=40241]: Move the **`PyGC_Head`** structure to the internal C API.
- bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170]: Convert **`PyObject_IS_GC()`**

macro to a function to hide implementation details.

- bpo-40241 [https://bugs.python.org/issue?@action=redirect&bpo=40241]: Add the functions `PyObject_GC_IsTracked()` and `PyObject_GC_IsFinalized()` to the public API to allow to query if Python objects are being currently tracked or have been already finalized by the garbage collector respectively. Patch by Pablo Galindo.
- bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170]: The `PyObject_NEW()` macro becomes an alias to the `PyObject_New()` macro, and the `PyObject_NEW_VAR()` macro becomes an alias to the `PyObject_NewVar()` macro, to hide implementation details. They no longer access directly the `PyTypeObject.tp_basicsize` member.
- bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170]: `PyType_HasFeature()` now always calls `PyType_GetFlags()` to hide implementation details. Previously, it accessed directly the `PyTypeObject.tp_flags` member when the limited C API was not used.
- bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170]: Convert the `PyObject_GET_WEAKREFS_LISTPTR()` macro to a function to hide implementation details: the macro accessed directly to the `PyTypeObject.tp_weaklistoffset` member.
- bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170]: Convert `PyObject_CheckBuffer()` macro to a function to hide implementation details: the macro accessed directly the `PyTypeObject.tp_as_buffer` member.
- bpo-40170 [https://bugs.python.org/issue?@action=redirect&bpo=40170]: Always declare `PyIndex_Check()` as an opaque function to hide implementation details: remove `PyIndex_Check()` macro. The macro accessed directly the `PyTypeObject.tp_as_number` member.
- bpo-39947 [https://bugs.python.org/issue?@action=redirect&bpo=39947]: Add `PyThreadState_GetID()` function: get the unique identifier of a Python thread state.

Python 3.9.0 alpha 5

Release date: 2020-03-23

Security

- bpo-38576 [https://bugs.python.org/issue?@action=redirect&bpo=38576]: Disallow control characters in hostnames in http.client, addressing **CVE 2019-18348** [https://www.cve.org/CVERecord?id=CVE-2019-18348]. Such potentially malicious header injection URLs now cause a InvalidURL to be raised.

Core and Builtins

- bpo-40010 [https://bugs.python.org/issue?@action=redirect&bpo=40010]: Optimize pending calls in multithreaded applications. If a thread different than the main thread schedules a pending call (`Py_AddPendingCall()`), the bytecode evaluation loop is no longer interrupted at each bytecode instruction to check for pending calls which cannot be executed. Only the main thread can execute pending calls.

  Previously, the bytecode evaluation loop was interrupted at each instruction until the main thread executes pending calls.

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _weakref extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _collections module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).

- bpo-40010 [https://bugs.python.org/issue?@action=redirect&bpo=40010]: Optimize signal handling in multithreaded applications. If a thread different than the main thread gets a signal, the bytecode evaluation loop is no longer interrupted at each bytecode instruction to check for pending signals which cannot be handled. Only the main thread of the main interpreter can handle signals.

Previously, the bytecode evaluation loop was interrupted at each instruction until the main thread handles signals.

- bpo-39984 [https://bugs.python.org/issue?@action=redirect&bpo=39984]: If `Py_AddPendingCall()` is called in a subinterpreter, the function is now scheduled to be called from the subinterpreter, rather than being called from the main interpreter. Each subinterpreter now has its own list of scheduled calls.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _heapq module to multiphase initialization.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port itertools module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-37207 [https://bugs.python.org/issue?@action=redirect&bpo=37207]: Speed up calls to `frozenset()` by using the **PEP 590** [https://peps.python.org/pep-0590/] `vectorcall` calling convention. Patch by Donghee Na.
- bpo-39984 [https://bugs.python.org/issue?@action=redirect&bpo=39984]: subinterpreters: Move `_PyRuntimeState.ceval.tracing_possible` to `PyInterpreterState.ceval.tracing_possible`: each interpreter now has its own variable.
- bpo-37207 [https://bugs.python.org/issue?@action=redirect&bpo=37207]: Speed up calls to `set()` by using the **PEP 590** [https://peps.python.org/pep-0590/] `vectorcall` calling convention. Patch by Donghee Na.
- bpo-1635741 [https://bugs.python.org/issue?

@action=redirect&bpo=1635741]: Port _statistics module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).

- bpo-39968 [https://bugs.python.org/issue?@action=redirect&bpo=39968]: Use inline function to replace extension modules" get_module_state macros.
- bpo-39965 [https://bugs.python.org/issue?@action=redirect&bpo=39965]: Correctly raise `SyntaxError` if *await* is used inside non-async functions and `PyCF_ALLOW_TOP_LEVEL_AWAIT` is set (like in the asyncio REPL). Patch by Pablo Galindo.
- bpo-39562 [https://bugs.python.org/issue?@action=redirect&bpo=39562]: Allow executing asynchronous comprehensions on the top level when the `PyCF_ALLOW_TOP_LEVEL_AWAIT` flag is given. Patch by Batuhan Taskaya.
- bpo-37207 [https://bugs.python.org/issue?@action=redirect&bpo=37207]: Speed up calls to `tuple()` by using the **PEP 590** [https://peps.python.org/pep-0590/] `vectorcall` calling convention. Patch by Donghee Na.
- bpo-38373 [https://bugs.python.org/issue?@action=redirect&bpo=38373]: Changed list overallocation strategy. It no longer overallocates if the new size is closer to overallocated size than to the old size and adds padding.
- bpo-39926 [https://bugs.python.org/issue?@action=redirect&bpo=39926]: Update Unicode database to Unicode version 13.0.0.
- bpo-19466 [https://bugs.python.org/issue?@action=redirect&bpo=19466]: Clear the frames of daemon threads earlier during the Python shutdown to call objects destructors. So «unclosed file» resource warnings are now emitted for daemon threads in a more reliable way.
- bpo-38894 [https://bugs.python.org/issue?

@action=redirect&bpo=38894]: Fix a bug that was causing incomplete results when calling `pathlib.Path.glob` in the presence of symlinks that point to files where the user does not have read access. Patch by Pablo Galindo and Matt Wozniski.

- bpo-39877 [https://bugs.python.org/issue?@action=redirect&bpo=39877]: Fix **`PyEval_RestoreThread()`** random crash at exit with daemon threads. It now accesses the `_PyRuntime` variable directly instead of using `tstate->interp->runtime`, since `tstate` can be a dangling pointer after **`Py_Finalize()`** has been called. Moreover, the daemon thread now exits before trying to take the GIL.

- bpo-39871 [https://bugs.python.org/issue?@action=redirect&bpo=39871]: Fix a possible **`SystemError`** in `math.{atan2,copysign,remainder}()` when the first argument cannot be converted to a **`float`**. Patch by Zackery Spytz.

- bpo-39776 [https://bugs.python.org/issue?@action=redirect&bpo=39776]: Fix race condition where threads created by PyGILState_Ensure() could get a duplicate id.

  This affects consumers of tstate->id like the contextvar caching machinery, which could return invalid cached objects under heavy thread load (observed in embedded scenarios).

- bpo-39778 [https://bugs.python.org/issue?@action=redirect&bpo=39778]: Fixed a crash due to incorrect handling of weak references in `collections.OrderedDict` classes. Patch by Pablo Galindo.

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port audioop extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).

- bpo-39702 [https://bugs.python.org/issue?@action=redirect&bpo=39702]: Relax decorator grammar restrictions to allow any valid expression (**PEP 614** [https://peps.python.org/pep-0614/]).

- bpo-38091 [https://bugs.python.org/issue?@action=redirect&bpo=38091]: Tweak import deadlock detection code to not deadlock itself.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _locale extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-39087 [https://bugs.python.org/issue?@action=redirect&bpo=39087]: Optimize `PyUnicode_AsUTF8()` and `PyUnicode_AsUTF8AndSize()` slightly when they need to create internal UTF-8 cache.
- bpo-39520 [https://bugs.python.org/issue?@action=redirect&bpo=39520]: Fix unparsing of ext slices with no items (`foo[:,]`). Patch by Batuhan Taskaya.
- bpo-39220 [https://bugs.python.org/issue?@action=redirect&bpo=39220]: Do not optimize annotations if “from __future__ import annotations” is used. Patch by Pablo Galindo.
- bpo-35712 [https://bugs.python.org/issue?@action=redirect&bpo=35712]: Using `NotImplemented` in a boolean context has been deprecated. Patch contributed by Josh Rosenberg.
- bpo-22490 [https://bugs.python.org/issue?@action=redirect&bpo=22490]: Don’t leak environment variable `__PYVENV_LAUNCHER__` into the interpreter session on macOS.

Library

- bpo-39830 [https://bugs.python.org/issue?@action=redirect&bpo=39830]: Add `zipfile.Path` to `__all__` in the **`zipfile`** module.

- bpo-40000 [https://bugs.python.org/issue?@action=redirect&bpo=40000]: Improved error messages for validation of `ast.Constant` nodes. Patch by Batuhan Taskaya.
- bpo-39999 [https://bugs.python.org/issue?@action=redirect&bpo=39999]: `__module__` of the AST node classes is now set to «ast» instead of «_ast». Added docstrings for dummy AST node classes and deprecated attributes.
- bpo-39991 [https://bugs.python.org/issue?@action=redirect&bpo=39991]: **`uuid.getnode()`** now skips IPv6 addresses with the same string length than a MAC address (17 characters): only use MAC addresses.
- bpo-39988 [https://bugs.python.org/issue?@action=redirect&bpo=39988]: Deprecated `ast.AugLoad` and `ast.AugStore` node classes because they are no longer used.
- bpo-39656 [https://bugs.python.org/issue?@action=redirect&bpo=39656]: Ensure `bin/python3.#` is always present in virtual environments on POSIX platforms - by Anthony Sottile.
- bpo-39969 [https://bugs.python.org/issue?@action=redirect&bpo=39969]: Deprecated `ast.Param` node class because it's no longer used. Patch by Batuhan Taskaya.
- bpo-39360 [https://bugs.python.org/issue?@action=redirect&bpo=39360]: Ensure all workers exit when finalizing a **`multiprocessing.Pool`** implicitly via the module finalization handlers of multiprocessing. This fixes a deadlock situation that can be experienced when the Pool is not properly finalized via the context manager or a call to `multiprocessing.Pool.terminate`. Patch by Batuhan Taskaya and Pablo Galindo.
- bpo-35370 [https://bugs.python.org/issue?@action=redirect&bpo=35370]: sys.settrace(), sys.setprofile() and _lsprof.Profiler.enable() now properly report **`PySys_Audit()`** error if «sys.setprofile» or «sys.settrace» audit event is denied.
- bpo-39936 [https://bugs.python.org/issue?@action=redirect&bpo=39936]: AIX: Fix _aix_support module when the subprocess is not available, when building Python

from scratch. It now uses new private _bootsubprocess module, rather than having two implementations depending if subprocess is available or not. So _aix_support.aix_platform() result is now the same if subprocess is available or not.

- bpo-36144 [https://bugs.python.org/issue?@action=redirect&bpo=36144]: **`collections.OrderedDict`** now implements `|` and `|=` (**PEP 584** [https://peps.python.org/pep-0584/]).
- bpo-39652 [https://bugs.python.org/issue?@action=redirect&bpo=39652]: The column name found in `sqlite3.Cursor.description` is now truncated on the first "[" only if the PARSE_COLNAMES option is set.
- bpo-39915 [https://bugs.python.org/issue?@action=redirect&bpo=39915]: Ensure **`unittest.mock.AsyncMock.await_args_list`** has call objects in the order of awaited arguments instead of using **`unittest.mock.Mock.call_args`** which has the last value of the call. Patch by Karthikeyan Singaravelan.
- bpo-36144 [https://bugs.python.org/issue?@action=redirect&bpo=36144]: Updated **`os.environ`** and **`os.environb`** to support **PEP 584** [https://peps.python.org/pep-0584/]'s merge (`|`) and update (`|=`) operators.
- bpo-38662 [https://bugs.python.org/issue?@action=redirect&bpo=38662]: The `ensurepip` module now invokes `pip` via the `runpy` module. Hence it is no longer tightly coupled with the internal API of the bundled `pip` version, allowing easier updates to a newer `pip` version both internally and for distributors.
- bpo-38075 [https://bugs.python.org/issue?@action=redirect&bpo=38075]: Fix the **`random.Random.seed()`** method when a **`bool`** is passed as the seed.
- bpo-39916 [https://bugs.python.org/issue?@action=redirect&bpo=39916]: More reliable use of `os.scandir()` in `Path.glob()`. It no longer emits a ResourceWarning when interrupted.
- bpo-39850 [https://bugs.python.org/issue?@action=redirect&bpo=39850]: **`multiprocessing`** now supports abstract socket addresses (if abstract sockets are

supported in the running platform). When creating arbitrary addresses (like when default-constructing **`multiprocessing.connection.Listener`** objects) abstract sockets are preferred to avoid the case when the temporary-file-generated address is too large for an AF_UNIX socket address. Patch by Pablo Galindo.

- bpo-36287 [https://bugs.python.org/issue?@action=redirect&bpo=36287]: **`ast.dump()`** no longer outputs optional fields and attributes with default values. The default values for optional fields and attributes of AST nodes are now set as class attributes (e.g. `Constant.kind` is set to `None`).
- bpo-39889 [https://bugs.python.org/issue?@action=redirect&bpo=39889]: Fixed **`ast.unparse()`** for extended slices containing a single element (e.g. `a[i:j,]`). Remove redundant tuples when index with a tuple (e.g. `a[i, j]`).
- bpo-39828 [https://bugs.python.org/issue?@action=redirect&bpo=39828]: Fix **`json.tool`** to catch **`BrokenPipeError`**. Patch by Donghee Na.
- bpo-13487 [https://bugs.python.org/issue?@action=redirect&bpo=13487]: Avoid a possible *«RuntimeError: dictionary changed size during iteration»* from **`inspect.getmodule()`** when it tried to loop through **`sys.modules`**.
- bpo-39674 [https://bugs.python.org/issue?@action=redirect&bpo=39674]: Revert «bpo-37330 [https://bugs.python.org/issue?@action=redirect&bpo=37330]: open() no longer accept "U" in file mode». The «U» mode of open() is kept in Python 3.9 to ease transition from Python 2.7, but will be removed in Python 3.10.
- bpo-28577 [https://bugs.python.org/issue?@action=redirect&bpo=28577]: The hosts method on 32-bit prefix length IPv4Networks and 128-bit prefix IPv6Networks now returns a list containing the single Address instead of an empty list.
- bpo-39826 [https://bugs.python.org/issue?@action=redirect&bpo=39826]: Add getConnection method to logging HTTPHandler to enable custom connections.
- bpo-39763 [https://bugs.python.org/issue?@action=redirect&bpo=39763]: Reimplement

`distutils.spawn.spawn` function with the `subprocess` module.

- bpo-39794 [https://bugs.python.org/issue?@action=redirect&bpo=39794]: Add –without-decimal-contextvar build option. This enables a thread-local rather than a coroutine local context.
- bpo-36144 [https://bugs.python.org/issue?@action=redirect&bpo=36144]: `collections.defaultdict` now implements `|` (**PEP 584** [https://peps.python.org/pep-0584/]).
- bpo-39517 [https://bugs.python.org/issue?@action=redirect&bpo=39517]: Fix runpy.run_path() when using pathlike objects
- bpo-39775 [https://bugs.python.org/issue?@action=redirect&bpo=39775]: Change `inspect.Signature.parameters` back to `collections.OrderedDict`. This was changed to `dict` in Python 3.9.0a4.
- bpo-39678 [https://bugs.python.org/issue?@action=redirect&bpo=39678]: Refactor queue_manager in `concurrent.futures.ProcessPoolExecutor` to make it easier to maintain.
- bpo-39764 [https://bugs.python.org/issue?@action=redirect&bpo=39764]: Fix AttributeError when calling get_stack on a PyAsyncGenObject Task
- bpo-39769 [https://bugs.python.org/issue?@action=redirect&bpo=39769]: The `compileall.compile_dir()` function's *ddir* parameter and the compileall command line flag `-d` no longer write the wrong pathname to the generated pyc file for submodules beneath the root of the directory tree being compiled. This fixes a regression introduced with Python 3.5.
- bpo-36144 [https://bugs.python.org/issue?@action=redirect&bpo=36144]: `types.MappingProxyType` objects now support the merge (`|`) operator from **PEP 584** [https://peps.python.org/pep-0584/].
- bpo-38691 [https://bugs.python.org/issue?@action=redirect&bpo=38691]: The `importlib` module now ignores the `PYTHONCASEOK` environment variable when the `-E` or `-I` command line options are being used.
- bpo-39719 [https://bugs.python.org/issue?

@action=redirect&bpo=39719]: Remove **`tempfile.SpooledTemporaryFile.softspace()`** as files no longer have the `softspace` attribute in Python 3. Patch by Shantanu.

- bpo-39667 [https://bugs.python.org/issue?@action=redirect&bpo=39667]: Improve pathlib.Path compatibility on zipfile.Path and correct performance degradation as found in zipp 3.0.
- bpo-39638 [https://bugs.python.org/issue?@action=redirect&bpo=39638]: Keep ASDL signatures in the docstrings for `AST` nodes. Patch by Batuhan Taskaya
- bpo-39639 [https://bugs.python.org/issue?@action=redirect&bpo=39639]: Deprecated `ast.Suite` node class because it's no longer used. Patch by Batuhan Taskaya.
- bpo-39609 [https://bugs.python.org/issue?@action=redirect&bpo=39609]: Add thread_name_prefix to default asyncio executor
- bpo-39548 [https://bugs.python.org/issue?@action=redirect&bpo=39548]: Fix handling of header in **`urllib.request.AbstractDigestAuthHandler`** when the optional `qop` parameter is not present.
- bpo-39509 [https://bugs.python.org/issue?@action=redirect&bpo=39509]: HTTP status codes `103 EARLY_HINTS` and `425 TOO_EARLY` are added to **`http.HTTPStatus`**. Patch by Donghee Na.
- bpo-39507 [https://bugs.python.org/issue?@action=redirect&bpo=39507]: Adding HTTP status 418 «I'm a Teapot» to HTTPStatus in http library. Patch by Ross Rhodes.
- bpo-39495 [https://bugs.python.org/issue?@action=redirect&bpo=39495]: Remove default value from *attrs* parameter of **`xml.etree.ElementTree.TreeBuilder.start()`** for consistency between Python and C implementations.
- bpo-38971 [https://bugs.python.org/issue?@action=redirect&bpo=38971]: Open issue in the BPO indicated a desire to make the implementation of codecs.open() at parity with io.open(), which implements a try/except to assure file stream gets closed before an exception is raised.
- bpo-38641 [https://bugs.python.org/issue?@action=redirect&bpo=38641]: Added starred expressions support

to `return` and `yield` statements for `lib2to3`. Patch by Vlad Emelianov.

- bpo-37534 [https://bugs.python.org/issue?@action=redirect&bpo=37534]: When using minidom module to generate XML documents the ability to add Standalone Document Declaration is added. All the changes are made to generate a document in compliance with Extensible Markup Language (XML) 1.0 (Fifth Edition) W3C Recommendation (available here: https://www.w3.org/TR/xml/#sec-prolog-dtd).
- bpo-34788 [https://bugs.python.org/issue?@action=redirect&bpo=34788]: Add support for scoped IPv6 addresses to **`ipaddress`**. Patch by Oleksandr Pavliuk.
- bpo-34822 [https://bugs.python.org/issue?@action=redirect&bpo=34822]: Simplified AST for subscription. Simple indices are now represented by their value, extended slices are represented as tuples. **`ast`** classes `Index` and `ExtSlice` are considered deprecated and will be removed in future Python versions. In the meantime, `Index(value)` now returns a `value` itself, `ExtSlice(slices)` returns `Tuple(slices, Load())`.

Documentation

- bpo-39868 [https://bugs.python.org/issue?@action=redirect&bpo=39868]: Updated the Language Reference for **PEP 572** [https://peps.python.org/pep-0572/].
- bpo-13790 [https://bugs.python.org/issue?@action=redirect&bpo=13790]: Change “string” to “specification” in format doc.
- bpo-17422 [https://bugs.python.org/issue?@action=redirect&bpo=17422]: The language reference no longer restricts default class namespaces to dicts only.
- bpo-39530 [https://bugs.python.org/issue?@action=redirect&bpo=39530]: Fix misleading documentation about mixed-type numeric comparisons.
- bpo-39718 [https://bugs.python.org/issue?@action=redirect&bpo=39718]: Update **`token`** documentation to reflect additions in Python 3.8
- bpo-39677 [https://bugs.python.org/issue?

@action=redirect&bpo=39677]: Changed operand name of **MAKE_FUNCTION** from *argc* to *flags* for module `dis`

Tests

- bpo-40019 [https://bugs.python.org/issue?@action=redirect&bpo=40019]: test_gdb now skips tests if it detects that gdb failed to read debug information because the Python binary is optimized.
- bpo-27807 [https://bugs.python.org/issue?@action=redirect&bpo=27807]: `test_site.test_startup_imports()` is now skipped if a path of `sys.path` contains a `.pth` file.
- bpo-26067 [https://bugs.python.org/issue?@action=redirect&bpo=26067]: Do not fail test_shutil test_chown test when uid or gid of user cannot be resolved to a name.
- bpo-39855 [https://bugs.python.org/issue?@action=redirect&bpo=39855]: test_subprocess.test_user() now skips the test on an user name if the user name doesn't exist. For example, skip the test if the user «nobody» doesn't exist on Linux.

Build

- bpo-39761 [https://bugs.python.org/issue?@action=redirect&bpo=39761]: Fix build with DTrace but without additional DFLAGS.
- bpo-39763 [https://bugs.python.org/issue?@action=redirect&bpo=39763]: setup.py now uses a basic implementation of the `subprocess` module if the **`subprocess`** module is not available: before required C extension modules are built.
- bpo-1294959 [https://bugs.python.org/issue?@action=redirect&bpo=1294959]: Add `--with-platlibdir` option to the configure script: name of the platform-specific library directory, stored in the new `sys.platlibdir` attribute. It is used to build the path of platform-specific extension modules and the path of the standard library. It is equal to `"lib"` on most platforms. On Fedora and SuSE, it is equal to `"lib64"` on 64-bit platforms. Patch by Jan Matějek,

Matěj Cepl, Charalampos Stratakis and Victor Stinner.

Windows

- bpo-39930 [https://bugs.python.org/issue?@action=redirect&bpo=39930]: Ensures the required `vcruntime140.dll` is included in install packages.
- bpo-39847 [https://bugs.python.org/issue?@action=redirect&bpo=39847]: Avoid hang when computer is hibernated whilst waiting for a mutex (for lock-related objects from `threading`) around 49-day uptime.
- bpo-38597 [https://bugs.python.org/issue?@action=redirect&bpo=38597]: `distutils` will no longer statically link `vcruntime140.dll` when a redistributable version is unavailable. All future releases of CPython will include a copy of this DLL to ensure distributed extensions can continue to load.
- bpo-38380 [https://bugs.python.org/issue?@action=redirect&bpo=38380]: Update Windows builds to use SQLite 3.31.1
- bpo-39789 [https://bugs.python.org/issue?@action=redirect&bpo=39789]: Update Windows release build machines to Visual Studio 2019 (MSVC 14.2).
- bpo-34803 [https://bugs.python.org/issue?@action=redirect&bpo=34803]: Package for nuget.org now includes repository reference and bundled icon image.

macOS

- bpo-38380 [https://bugs.python.org/issue?@action=redirect&bpo=38380]: Update macOS builds to use SQLite 3.31.1

IDLE

- bpo-27115 [https://bugs.python.org/issue?@action=redirect&bpo=27115]: For “Go to Line”, use a Query box subclass with IDLE standard behavior and improved error checking.
- bpo-39885 [https://bugs.python.org/issue?

@action=redirect&bpo=39885]: Since clicking to get an IDLE context menu moves the cursor, any text selection should be and now is cleared.
- bpo-39852 [https://bugs.python.org/issue?@action=redirect&bpo=39852]: Edit «Go to line» now clears any selection, preventing accidental deletion. It also updates Ln and Col on the status bar.
- bpo-39781 [https://bugs.python.org/issue?@action=redirect&bpo=39781]: Selecting code context lines no longer causes a jump.

Tools/Demos

- bpo-36184 [https://bugs.python.org/issue?@action=redirect&bpo=36184]: Port python-gdb.py to FreeBSD. python-gdb.py now checks for «take_gil» function name to check if a frame tries to acquire the GIL, instead of checking for «pthread_cond_timedwait» which is specific to Linux and can be a different condition than the GIL.
- bpo-38080 [https://bugs.python.org/issue?@action=redirect&bpo=38080]: Added support to fix `getproxies` in the **`lib2to3.fixes.fix_urllib`** module. Patch by José Roberto Meza Cabrera.

C API

- bpo-40024 [https://bugs.python.org/issue?@action=redirect&bpo=40024]: Add **`PyModule_AddType()`** helper function: add a type to a module. Patch by Donghee Na.

- bpo-39946 [https://bugs.python.org/issue?@action=redirect&bpo=39946]: Remove `_PyRuntime.getframe` hook and remove `_PyThreadState_GetFrame` macro which was an alias to `_PyRuntime.getframe`. They were only exposed by the internal C API. Remove also `PyThreadFrameGetter` type.

- bpo-39947 [https://bugs.python.org/issue?@action=redirect&bpo=39947]: Add

`PyThreadState_GetFrame()` function: get the current frame of a Python thread state.

- bpo-37207 [https://bugs.python.org/issue?@action=redirect&bpo=37207]: Add _PyArg_NoKwnames helper function. Patch by Donghee Na.
- bpo-39947 [https://bugs.python.org/issue?@action=redirect&bpo=39947]: Add `PyThreadState_GetInterpreter()`: get the interpreter of a Python thread state.
- bpo-39947 [https://bugs.python.org/issue?@action=redirect&bpo=39947]: Add `PyInterpreterState_Get()` function to the limited C API.
- bpo-35370 [https://bugs.python.org/issue?@action=redirect&bpo=35370]: If `PySys_Audit()` fails in `PyEval_SetProfile()` or `PyEval_SetTrace()`, log the error as an unraisable exception.
- bpo-39947 [https://bugs.python.org/issue?@action=redirect&bpo=39947]: Move the static inline function flavor of Py_EnterRecursiveCall() and Py_LeaveRecursiveCall() to the internal C API: they access PyThreadState attributes. The limited C API provides regular functions which hide implementation details.
- bpo-39947 [https://bugs.python.org/issue?@action=redirect&bpo=39947]: Py_TRASHCAN_BEGIN_CONDITION and Py_TRASHCAN_END macro no longer access PyThreadState attributes, but call new private _PyTrash_begin() and _PyTrash_end() functions which hide implementation details.
- bpo-39884 [https://bugs.python.org/issue?@action=redirect&bpo=39884]: `PyDescr_NewMethod()` and `PyCFunction_NewEx()` now include the method name in the SystemError «bad call flags» error message to ease debug.

- bpo-39877 [https://bugs.python.org/issue?@action=redirect&bpo=39877]: Deprecated **`PyEval_InitThreads()`** and **`PyEval_ThreadsInitialized()`**. Calling **`PyEval_InitThreads()`** now does nothing.

- bpo-38249 [https://bugs.python.org/issue?@action=redirect&bpo=38249]: **`Py_UNREACHABLE`** is now implemented with `__builtin_unreachable()` and analogs in release mode.

- bpo-38643 [https://bugs.python.org/issue?@action=redirect&bpo=38643]: **`PyNumber_ToBase()`** now raises a **`SystemError`** instead of crashing when called with invalid base.

- bpo-39882 [https://bugs.python.org/issue?@action=redirect&bpo=39882]: The **`Py_FatalError()`** function is replaced with a macro which logs automatically the name of the current function, unless the `Py_LIMITED_API` macro is defined.

- bpo-39824 [https://bugs.python.org/issue?@action=redirect&bpo=39824]: Extension modules: **`m_traverse`**, **`m_clear`** and **`m_free`** functions of **`PyModuleDef`** are no longer called if the module state was requested but is not allocated yet. This is the case immediately after the module is created and before the module is executed (**`Py_mod_exec`** function). More precisely, these functions are not called if **`m_size`** is greater than 0 and the module state (as returned by **`PyModule_GetState()`**) is `NULL`.

  Extension modules without module state (`m_size <= 0`) are not affected.

- bpo-38913 [https://bugs.python.org/issue?@action=redirect&bpo=38913]: Fixed segfault in `Py_BuildValue()` called with a format containing «#» and undefined PY_SSIZE_T_CLEAN whwn an exception is set.

- bpo-38500 [https://bugs.python.org/issue?@action=redirect&bpo=38500]: Add a private API to get and set the frame evaluation function: add `_PyInterpreterState_GetEvalFrameFunc()` and `_PyInterpreterState_SetEvalFrameFunc()` C functions. The `_PyFrameEvalFunction` function type now takes a *tstate* parameter.

Python 3.9.0 alpha 4

Release date: 2020-02-25

Security

- bpo-39184 [https://bugs.python.org/issue?@action=redirect&bpo=39184]: Add audit events to functions in `fcntl`, `msvcrt`, `os`, `resource`, `shutil`, `signal` and `syslog`.
- bpo-39401 [https://bugs.python.org/issue?@action=redirect&bpo=39401]: Avoid unsafe DLL load at startup on Windows 7 and earlier.
- bpo-39184 [https://bugs.python.org/issue?@action=redirect&bpo=39184]: Add audit events to command execution functions in os and pty modules.

Core and Builtins

- bpo-39382 [https://bugs.python.org/issue?@action=redirect&bpo=39382]: Fix a use-after-free in the single inheritance path of `issubclass()`, when the `__bases__` of an object has a single reference, and so does its first item. Patch by Yonatan Goldschmidt.
- bpo-39573 [https://bugs.python.org/issue?@action=redirect&bpo=39573]: Update clinic tool to use `Py_IS_TYPE()`. Patch by Donghee Na.
- bpo-39619 [https://bugs.python.org/issue?@action=redirect&bpo=39619]: Enable use of `os.chroot()` on HP-UX systems.
- bpo-39573 [https://bugs.python.org/issue?

@action=redirect&bpo=39573]: Add **`Py_IS_TYPE()`** static inline function to check whether the object *o* type is *type*.
- bpo-39606 [https://bugs.python.org/issue?@action=redirect&bpo=39606]: Fix regression caused by fix for bpo-39386 [https://bugs.python.org/issue?@action=redirect&bpo=39386], that prevented calling `aclose` on an async generator that had already been closed or exhausted.
- bpo-39579 [https://bugs.python.org/issue?@action=redirect&bpo=39579]: Change the ending column offset of `Attribute` nodes constructed in `ast_for_dotted_name` to point at the end of the current node and not at the end of the last `NAME` node.
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _crypt extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _contextvars extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-39510 [https://bugs.python.org/issue?@action=redirect&bpo=39510]: Fix segfault in `readinto()` method on closed BufferedReader.
- bpo-39502 [https://bugs.python.org/issue?@action=redirect&bpo=39502]: Fix **`time.localtime()`** on 64-bit AIX to support years before 1902 and after 2038. Patch by M Felt.
- bpo-39492 [https://bugs.python.org/issue?@action=redirect&bpo=39492]: Fix a reference cycle in the C Pickler that was preventing the garbage collection of deleted, pickled objects.
- bpo-39453 [https://bugs.python.org/issue?@action=redirect&bpo=39453]: Fixed a possible crash in **`list.__contains__()`** when a list is changed during comparing items. Patch by Donghee Na.
- bpo-39434 [https://bugs.python.org/issue?@action=redirect&bpo=39434]: floor division of float operation now has a better performance. Also the message of **`ZeroDivisionError`** for this operation is updated. Patch by

Donghee Na.

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _codecs extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _bz2 extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _abc extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).
- bpo-39320 [https://bugs.python.org/issue?@action=redirect&bpo=39320]: Replace two complex bytecodes for building dicts with two simpler ones. The new bytecodes `DICT_MERGE` and `DICT_UPDATE` have been added The old bytecodes `BUILD_MAP_UNPACK` and `BUILD_MAP_UNPACK_WITH_CALL` have been removed.
- bpo-39219 [https://bugs.python.org/issue?@action=redirect&bpo=39219]: Syntax errors raised in the tokenizer now always set correct «text» and «offset» attributes.
- bpo-36051 [https://bugs.python.org/issue?@action=redirect&bpo=36051]: Drop the GIL during large `bytes.join` operations. Patch by Bruce Merry.
- bpo-38960 [https://bugs.python.org/issue?@action=redirect&bpo=38960]: Fix DTrace build issues on FreeBSD. Patch by David Carlier.
- bpo-37207 [https://bugs.python.org/issue?@action=redirect&bpo=37207]: Speed up calls to `range()` by about 30%, by using the PEP 590 `vectorcall` calling convention. Patch by Mark Shannon.
- bpo-36144 [https://bugs.python.org/issue?@action=redirect&bpo=36144]: **`dict`** (and **`collections.UserDict`**) objects now support PEP 584's merge (|) and update (|=) operators. Patch by Brandt Bucher.
- bpo-32856 [https://bugs.python.org/issue?@action=redirect&bpo=32856]: Optimized the idiom for

assignment a temporary variable in comprehensions. Now `for y in [expr]` in comprehensions is as fast as a simple assignment `y = expr`.

Library

- bpo-30566 [https://bugs.python.org/issue?@action=redirect&bpo=30566]: Fix **`IndexError`** when trying to decode an invalid string with punycode codec.
- bpo-39649 [https://bugs.python.org/issue?@action=redirect&bpo=39649]: Remove obsolete check for `__args__` in `bdb.Bdb.format_stack_entry`.
- bpo-39648 [https://bugs.python.org/issue?@action=redirect&bpo=39648]: Expanded **`math.gcd()`** and **`math.lcm()`** to handle multiple arguments.
- bpo-39681 [https://bugs.python.org/issue?@action=redirect&bpo=39681]: Fix a regression where the C pickle module wouldn't allow unpickling from a file-like object that doesn't expose a readinto() method.
- bpo-35950 [https://bugs.python.org/issue?@action=redirect&bpo=35950]: Raise **`io.UnsupportedOperation`** in **`io.BufferedReader.truncate()`** when it is called on a read-only **`io.BufferedReader`** instance.
- bpo-39479 [https://bugs.python.org/issue?@action=redirect&bpo=39479]: Add **`math.lcm()`** function: least common multiple.
- bpo-39674 [https://bugs.python.org/issue?@action=redirect&bpo=39674]: Revert «Do not expose abstract collection classes in the collections module» change (bpo-25988 [https://bugs.python.org/issue?@action=redirect&bpo=25988]). Aliases to ABC like collections.Mapping are kept in Python 3.9 to ease transition from Python 2.7, but will be removed in Python 3.10.
- bpo-39104 [https://bugs.python.org/issue?@action=redirect&bpo=39104]: Fix hanging ProcessPoolExcutor on `shutdown(wait=False)` when a task has failed pickling.
- bpo-39627 [https://bugs.python.org/issue?@action=redirect&bpo=39627]: Fixed TypedDict totality check for

inherited keys.

- bpo-39474 [https://bugs.python.org/issue?@action=redirect&bpo=39474]: Fixed starting position of AST for expressions like `(a)(b)`, `(a)[b]` and `(a).b`.
- bpo-21016 [https://bugs.python.org/issue?@action=redirect&bpo=21016]: The **`pydoc`** and **`trace`** modules now use the **`sysconfig`** module to get the path to the Python standard library, to support uncommon installation path like `/usr/lib64/python3.9/` on Fedora. Patch by Jan Matějek.
- bpo-39590 [https://bugs.python.org/issue?@action=redirect&bpo=39590]: Collections.deque now holds strong references during deque.__contains__ and deque.count, fixing crashes.
- bpo-39586 [https://bugs.python.org/issue?@action=redirect&bpo=39586]: The distutils `bdist_msi` command is deprecated in Python 3.9, use `bdist_wheel` (wheel packages) instead.
- bpo-39595 [https://bugs.python.org/issue?@action=redirect&bpo=39595]: Improved performance of zipfile.Path for files with a large number of entries. Also improved performance and fixed minor issue as published with importlib_metadata 1.5 [https://importlib-metadata.readthedocs.io/en/latest/history.html#v1-5-0].
- bpo-39350 [https://bugs.python.org/issue?@action=redirect&bpo=39350]: Fix regression in **`fractions.Fraction`** if the numerator and/or the denominator is an **`int`** subclass. The **`math.gcd()`** function is now used to normalize the *numerator* and *denominator*. **`math.gcd()`** always return a **`int`** type. Previously, the GCD type depended on *numerator* and *denominator*.
- bpo-39567 [https://bugs.python.org/issue?@action=redirect&bpo=39567]: Added audit for **`os.walk()`**, **`os.fwalk()`**, **`pathlib.Path.glob()`** and **`pathlib.Path.rglob()`**.
- bpo-39559 [https://bugs.python.org/issue?@action=redirect&bpo=39559]: Remove unused, undocumented argument `getters` from **`uuid.getnode()`**
- bpo-38149 [https://bugs.python.org/issue?@action=redirect&bpo=38149]: **`sys.audit()`** is now called only

once per call of `glob.glob()` and `glob.iglob()`.

- bpo-39546 [https://bugs.python.org/issue?@action=redirect&bpo=39546]: Fix a regression in `ArgumentParser` where `allow_abbrev=False` was ignored for long options that used a prefix character other than «-«.
- bpo-39450 [https://bugs.python.org/issue?@action=redirect&bpo=39450]: Striped whitespace from docstring before returning it from **`unittest.case.shortDescription()`**.
- bpo-12915 [https://bugs.python.org/issue?@action=redirect&bpo=12915]: A new function `resolve_name` has been added to the `pkgutil` module. This resolves a string of the form `'a.b.c.d'` or `'a.b:c.d'` to an object. In the example, `a.b` is a package/module and `c.d` is an object within that package/module reached via recursive attribute access.
- bpo-39353 [https://bugs.python.org/issue?@action=redirect&bpo=39353]: The `binascii.crc_hqx()` function is no longer deprecated.
- bpo-39493 [https://bugs.python.org/issue?@action=redirect&bpo=39493]: Mark `typing.IO.closed` as a property
- bpo-39491 [https://bugs.python.org/issue?@action=redirect&bpo=39491]: Add `typing.Annotated` and `include_extras` parameter to `typing.get_type_hints()` as part of PEP 593 [https://peps.python.org/pep-0593/]. Patch by Till Varoquaux, documentation by Till Varoquaux and Konstantin Kashin.
- bpo-39485 [https://bugs.python.org/issue?@action=redirect&bpo=39485]: Fix a bug in `unittest.mock.create_autospec()` that would complain about the wrong number of arguments for custom descriptors defined in an extension module returning functions.
- bpo-38932 [https://bugs.python.org/issue?@action=redirect&bpo=38932]: Mock fully resets child objects on reset_mock(). Patch by Vegard Stikbakke
- bpo-39082 [https://bugs.python.org/issue?@action=redirect&bpo=39082]: Allow AsyncMock to correctly

patch static/class methods

- bpo-39432 [https://bugs.python.org/issue?@action=redirect&bpo=39432]: Implement PEP-489 algorithm for non-ascii «PyInit_…» symbol names in distutils to make it export the correct init symbol also on Windows.
- bpo-18819 [https://bugs.python.org/issue?@action=redirect&bpo=18819]: Omit `devmajor` and `devminor` fields for non-device files in **`tarfile`** archives, enabling bit-for-bit compatibility with GNU `tar(1)`.
- bpo-39349 [https://bugs.python.org/issue?@action=redirect&bpo=39349]: Added a new *cancel_futures* parameter to **`concurrent.futures.Executor.shutdown()`** that cancels all pending futures which have not started running, instead of waiting for them to complete before shutting down the executor.
- bpo-39274 [https://bugs.python.org/issue?@action=redirect&bpo=39274]: `bool(fraction.Fraction)` now returns a boolean even if (numerator != 0) does not return a boolean (ex: numpy number).
- bpo-34793 [https://bugs.python.org/issue?@action=redirect&bpo=34793]: Remove support for `with (await asyncio.lock):` and `with (yield from asyncio.lock):`. The same is correct for `asyncio.Condition` and `asyncio.Semaphore`.
- bpo-25597 [https://bugs.python.org/issue?@action=redirect&bpo=25597]: Ensure, if `wraps` is supplied to **`unittest.mock.MagicMock`**, it is used to calculate return values for the magic methods instead of using the default return values. Patch by Karthikeyan Singaravelan.
- bpo-36350 [https://bugs.python.org/issue?@action=redirect&bpo=36350]: `inspect.Signature.parameters` and `inspect.BoundArguments.arguments` are now dicts instead of OrderedDicts. Patch contributed by Rémi Lapeyre.
- bpo-35727 [https://bugs.python.org/issue?@action=redirect&bpo=35727]: Fix sys.exit() and sys.exit(None) exit code propagation when used in multiprocessing.Process.
- bpo-32173 [https://bugs.python.org/issue?@action=redirect&bpo=32173]: * Add `lazycache` function to

`__all__`. * Use `dict.clear` to clear the cache. * Refactoring `getline` function and `checkcache` function.

Documentation

- bpo-17422 [https://bugs.python.org/issue?@action=redirect&bpo=17422]: The language reference now specifies restrictions on class namespaces. Adapted from a patch by Ethan Furman.
- bpo-39572 [https://bugs.python.org/issue?@action=redirect&bpo=39572]: Updated documentation of `total` flag of `TypedDict`.
- bpo-39654 [https://bugs.python.org/issue?@action=redirect&bpo=39654]: In pyclbr doc, update “class” to “module” where appropriate and add readmodule comment. Patch by Hakan Çelik.
- bpo-39153 [https://bugs.python.org/issue?@action=redirect&bpo=39153]: Clarify refcounting semantics for the following functions: - PyObject_SetItem - PyMapping_SetItemString - PyDict_SetItem - PyDict_SetItemString
- bpo-39392 [https://bugs.python.org/issue?@action=redirect&bpo=39392]: Explain that when filling with turtle, overlap regions may be left unfilled.
- bpo-39369 [https://bugs.python.org/issue?@action=redirect&bpo=39369]: Update mmap readline method description. The fact that the readline method does update the file position should not be ignored since this might give the impression for the programmer that it doesn’t update it.
- bpo-9056 [https://bugs.python.org/issue?@action=redirect&bpo=9056]: Include subsection in TOC for PDF version of docs.

Tests

- bpo-38325 [https://bugs.python.org/issue?@action=redirect&bpo=38325]: Skip tests on non-BMP characters of test_winconsoleio.
- bpo-39502 [https://bugs.python.org/issue?@action=redirect&bpo=39502]: Skip test_zipfile.test_add_file_after_2107() if **`time.localtime()`**

fails with `OverflowError`. It is the case on AIX 6.1 for example.

Build

- bpo-39489 [https://bugs.python.org/issue?@action=redirect&bpo=39489]: Remove `COUNT_ALLOCS` special build.

Windows

- bpo-39553 [https://bugs.python.org/issue?@action=redirect&bpo=39553]: Delete unused code related to SxS manifests.
- bpo-39439 [https://bugs.python.org/issue?@action=redirect&bpo=39439]: Honor the Python path when a virtualenv is active on Windows.
- bpo-39393 [https://bugs.python.org/issue?@action=redirect&bpo=39393]: Improve the error message when attempting to load a DLL with unresolved dependencies.
- bpo-38883 [https://bugs.python.org/issue?@action=redirect&bpo=38883]: `home()` and `expanduser()` on Windows now prefer **`USERPROFILE`** and no longer use **`HOME`**, which is not normally set for regular user accounts. This makes them again behave like `os.path.expanduser()`, which was changed to ignore **`HOME`** in 3.8, see bpo-36264 [https://bugs.python.org/issue?@action=redirect&bpo=36264].
- bpo-39185 [https://bugs.python.org/issue?@action=redirect&bpo=39185]: The build.bat script has additional options for very-quiet output (-q) and very-verbose output (-vv)

IDLE

- bpo-39663 [https://bugs.python.org/issue?@action=redirect&bpo=39663]: Add tests for pyparse find_good_parse_start().
- bpo-39600 [https://bugs.python.org/issue?@action=redirect&bpo=39600]: In the font configuration window,

remove duplicated font names.
- bpo-30780 [https://bugs.python.org/issue?@action=redirect&bpo=30780]: Add remaining configdialog tests for buttons and highlights and keys tabs.
- bpo-39388 [https://bugs.python.org/issue?@action=redirect&bpo=39388]: IDLE Settings Cancel button now cancels pending changes
- bpo-38792 [https://bugs.python.org/issue?@action=redirect&bpo=38792]: Close an IDLE shell calltip if a **`KeyboardInterrupt`** or shell restart occurs. Patch by Zackery Spytz.

C API

- bpo-35081 [https://bugs.python.org/issue?@action=redirect&bpo=35081]: Move the `bytes_methods.h` header file to the internal C API as `pycore_bytes_methods.h`: it only contains private symbols (prefixed by `_Py`), except of the `PyDoc_STRVAR_shared()` macro.
- bpo-35081 [https://bugs.python.org/issue?@action=redirect&bpo=35081]: Move the `dtoa.h` header file to the internal C API as `pycore_dtoa.h`: it only contains private functions (prefixed by `_Py`). The **`math`** and **`cmath`** modules must now be compiled with the `Py_BUILD_CORE` macro defined.
- bpo-39573 [https://bugs.python.org/issue?@action=redirect&bpo=39573]: Add **`Py_SET_SIZE()`** function to set the size of an object.
- bpo-39500 [https://bugs.python.org/issue?@action=redirect&bpo=39500]: **`PyUnicode_IsIdentifier()`** does not call **`Py_FatalError()`** anymore if the string is not ready.
- bpo-39573 [https://bugs.python.org/issue?@action=redirect&bpo=39573]: Add **`Py_SET_TYPE()`** function to set the type of an object.
- bpo-39573 [https://bugs.python.org/issue?@action=redirect&bpo=39573]: Add a **`Py_SET_REFCNT()`** function to set the reference counter of an object.
- bpo-39542 [https://bugs.python.org/issue?

@action=redirect&bpo=39542]: Convert **`PyType_HasFeature()`**, **`PyType_Check()`** and **`PyType_CheckExact()`** macros to static inline functions.
- bpo-39542 [https://bugs.python.org/issue?@action=redirect&bpo=39542]: In the limited C API, `PyObject_INIT()` and `PyObject_INIT_VAR()` are now defined as aliases to **`PyObject_Init()`** and **`PyObject_InitVar()`** to make their implementation opaque. It avoids to leak implementation details in the limited C API. Exclude the following functions from the limited C API: `_Py_NewReference()`, `_Py_ForgetReference()`, `_PyTraceMalloc_NewReference()` and `_Py_GetRefTotal()`.
- bpo-39542 [https://bugs.python.org/issue?@action=redirect&bpo=39542]: Exclude trashcan mechanism from the limited C API: it requires access to PyTypeObject and PyThreadState structure fields, whereas these structures are opaque in the limited C API.
- bpo-39511 [https://bugs.python.org/issue?@action=redirect&bpo=39511]: The **`PyThreadState_Clear()`** function now calls the **`PyThreadState.on_delete`** callback. Previously, that happened in **`PyThreadState_Delete()`**.
- bpo-38076 [https://bugs.python.org/issue?@action=redirect&bpo=38076]: Fix to clear the interpreter state only after clearing module globals to guarantee module state access from C Extensions during runtime destruction
- bpo-39245 [https://bugs.python.org/issue?@action=redirect&bpo=39245]: The Vectorcall API (PEP 590) was made public, adding the functions `PyObject_Vectorcall`, `PyObject_VectorcallMethod`, `PyVectorcall_Function`, `PyObject_CallOneArg`, `PyObject_CallMethodNoArgs`, `PyObject_CallMethodOneArg`, `PyObject_FastCallDict`, and the flag `Py_TPFLAGS_HAVE_VECTORCALL`.

Python 3.9.0 alpha 3

Release date: 2020-01-24

Core and Builtins

- bpo-39427 [https://bugs.python.org/issue?@action=redirect&bpo=39427]: Document all possibilities for the `-X` options in the command line help section. Patch by Pablo Galindo.

- bpo-39421 [https://bugs.python.org/issue?@action=redirect&bpo=39421]: Fix possible crashes when operating with the functions in the **`heapq`** module and custom comparison operators.

- bpo-39386 [https://bugs.python.org/issue?@action=redirect&bpo=39386]: Prevent double awaiting of async iterator.

- bpo-17005 [https://bugs.python.org/issue?@action=redirect&bpo=17005]: Add **`functools.TopologicalSorter`** to the **`functools`** module to offers functionality to perform topological sorting of graphs. Patch by Pablo Galindo, Tim Peters and Larry Hastings.

- bpo-39320 [https://bugs.python.org/issue?@action=redirect&bpo=39320]: Replace four complex bytecodes for building sequences with three simpler ones.

  The following four bytecodes have been removed:

  - BUILD_LIST_UNPACK
  - BUILD_TUPLE_UNPACK
  - BUILD_SET_UNPACK
  - BUILD_TUPLE_UNPACK_WITH_CALL

  The following three bytecodes have been added:

  - LIST_TO_TUPLE
  - LIST_EXTEND
  - SET_UPDATE

- bpo-39336 [https://bugs.python.org/issue?@action=redirect&bpo=39336]: Import loaders which publish immutable module objects can now publish immutable packages in addition to individual modules.

- bpo-39322 [https://bugs.python.org/issue?@action=redirect&bpo=39322]: Added a new function **`gc.is_finalized()`** to check if an object has been finalized by the garbage collector. Patch by Pablo Galindo.

- bpo-39048 [https://bugs.python.org/issue?@action=redirect&bpo=39048]: Improve the displayed error message when incorrect types are passed to `async with` statements by looking up the **`__aenter__()`** special method before the **`__aexit__()`** special method when entering an asynchronous context manager. Patch by Géry Ogam.

- bpo-39235 [https://bugs.python.org/issue?@action=redirect&bpo=39235]: Fix AST end location for lone generator expression in function call, e.g. f(i for i in a).

- bpo-39209 [https://bugs.python.org/issue?@action=redirect&bpo=39209]: Correctly handle multi-line tokens in interactive mode. Patch by Pablo Galindo.

- bpo-1635741 [https://bugs.python.org/issue?@action=redirect&bpo=1635741]: Port _json extension module to multiphase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).

- bpo-39216 [https://bugs.python.org/issue?@action=redirect&bpo=39216]: Fix constant folding optimization for positional only arguments - by Anthony Sottile.

- bpo-39215 [https://bugs.python.org/issue?@action=redirect&bpo=39215]: Fix `SystemError` when nested function has annotation on positional-only argument - by Anthony Sottile.

- bpo-39200 [https://bugs.python.org/issue?@action=redirect&bpo=39200]: Correct the error message when calling the `min()` or `max()` with no arguments. Patch by Donghee Na.

- bpo-39200 [https://bugs.python.org/issue?@action=redirect&bpo=39200]: Correct the error message when trying to construct `range` objects with no arguments. Patch by Pablo Galindo.

- bpo-39166 [https://bugs.python.org/issue?@action=redirect&bpo=39166]: Fix incorrect line execution reporting in trace functions when tracing the last iteration of asynchronous for loops. Patch by Pablo Galindo.

- bpo-39114 [https://bugs.python.org/issue?@action=redirect&bpo=39114]: Fix incorrect line execution reporting in trace functions when tracing exception handlers with name binding. Patch by Pablo Galindo.

- bpo-39156 [https://bugs.python.org/issue?@action=redirect&bpo=39156]: Split the COMPARE_OP bytecode instruction into four distinct instructions.

  - COMPARE_OP for rich comparisons
  - IS_OP for “is” and “is not” tests
  - CONTAINS_OP for “in” and “is not” tests
  - JUMP_IF_NOT_EXC_MATCH for checking exceptions in “try-except” statements.

  This improves the clarity of the interpreter and should provide a modest speedup.

- bpo-38588 [https://bugs.python.org/issue?@action=redirect&bpo=38588]: Fix possible crashes in dict and list when calling `PyObject_RichCompareBool()`.

- bpo-13601 [https://bugs.python.org/issue?@action=redirect&bpo=13601]: By default, `sys.stderr` is line-buffered now, even if `stderr` is redirected to a file. You can still make `sys.stderr` unbuffered by passing the `-u` command-line option or setting the `PYTHONUNBUFFERED` environment variable.

(Contributed by Jendrik Seipp in bpo-13601 [https://bugs.python.org/issue?@action=redirect&bpo=13601].)

- bpo-38610 [https://bugs.python.org/issue?@action=redirect&bpo=38610]: Fix possible crashes in several list methods by holding strong references to list elements when calling **`PyObject_RichCompareBool()`**.

- bpo-32021 [https://bugs.python.org/issue?@action=redirect&bpo=32021]: Include brotli .br encoding in mimetypes encodings_map

Library

- bpo-39430 [https://bugs.python.org/issue?@action=redirect&bpo=39430]: Fixed race condition in lazy imports in **`tarfile`**.
- bpo-39413 [https://bugs.python.org/issue?@action=redirect&bpo=39413]: The **`os.unsetenv()`** function is now also available on Windows.
- bpo-39390 [https://bugs.python.org/issue?@action=redirect&bpo=39390]: Fixed a regression with the `ignore` callback of **`shutil.copytree()`**. The argument types are now `str` and `List[str]` again.
- bpo-39395 [https://bugs.python.org/issue?@action=redirect&bpo=39395]: The **`os.putenv()`** and **`os.unsetenv()`** functions are now always available.
- bpo-39406 [https://bugs.python.org/issue?@action=redirect&bpo=39406]: If `setenv()` C function is available, **`os.putenv()`** is now implemented with `setenv()` instead of `putenv()`, so Python doesn't have to handle the environment variable memory.
- bpo-39396 [https://bugs.python.org/issue?@action=redirect&bpo=39396]: Fix `math.nextafter(-0.0, +0.0)` on AIX 7.1.
- bpo-29435 [https://bugs.python.org/issue?@action=redirect&bpo=29435]: Allow **`tarfile.is_tarfile()`** to be used with file and file-like objects, like **`zipfile.is_zipfile()`**. Patch by William Woodruff.
- bpo-39377 [https://bugs.python.org/issue?

@action=redirect&bpo=39377]: Removed `encoding` option from **`json.loads()`**. It has been deprecated since Python 3.1.

- bpo-39389 [https://bugs.python.org/issue?@action=redirect&bpo=39389]: Write accurate compression level metadata in **`gzip`** archives, rather than always signaling maximum compression.
- bpo-39366 [https://bugs.python.org/issue?@action=redirect&bpo=39366]: The previously deprecated `xpath()` and `xgtitle()` methods of **`nntplib.NNTP`** have been removed.
- bpo-39357 [https://bugs.python.org/issue?@action=redirect&bpo=39357]: Remove the *buffering* parameter of **`bz2.BZ2File`**. Since Python 3.0, it was ignored and using it was emitting **`DeprecationWarning`**. Pass an open file object, to control how the file is opened. The *compresslevel* parameter becomes keyword-only.
- bpo-39353 [https://bugs.python.org/issue?@action=redirect&bpo=39353]: Deprecate binhex4 and hexbin4 standards. Deprecate the **`binhex`** module and the following **`binascii`** functions: **`b2a_hqx()`**, **`a2b_hqx()`**, **`rlecode_hqx()`**, **`rledecode_hqx()`**, **`crc_hqx()`**.
- bpo-39351 [https://bugs.python.org/issue?@action=redirect&bpo=39351]: Remove `base64.encodestring()` and `base64.decodestring()`, aliases deprecated since Python 3.1: use **`base64.encodebytes()`** and **`base64.decodebytes()`** instead.
- bpo-39350 [https://bugs.python.org/issue?@action=redirect&bpo=39350]: Remove `fractions.gcd()` function, deprecated since Python 3.5 (bpo-22486 [https://bugs.python.org/issue?@action=redirect&bpo=22486]): use **`math.gcd()`** instead.
- bpo-39329 [https://bugs.python.org/issue?@action=redirect&bpo=39329]: **`LMTP`** constructor now has an optional *timeout* parameter. Patch by Donghee Na.
- bpo-39313 [https://bugs.python.org/issue?@action=redirect&bpo=39313]: Add a new `exec_function` option (*–exec-function* in the CLI) to `RefactoringTool` for making `exec` a function. Patch by Batuhan Taskaya.
- bpo-39259 [https://bugs.python.org/issue?

@action=redirect&bpo=39259]: `FTP_TLS` and **`FTP_TLS`** now raise a `ValueError` if the given timeout for their constructor is zero to prevent the creation of a non-blocking socket. Patch by Donghee Na.

- bpo-39259 [https://bugs.python.org/issue?@action=redirect&bpo=39259]: `SMTP` and `SMTP_SSL` now raise a `ValueError` if the given timeout for their constructor is zero to prevent the creation of a non-blocking socket. Patch by Donghee Na.
- bpo-39310 [https://bugs.python.org/issue?@action=redirect&bpo=39310]: Add `math.ulp()`: return the value of the least significant bit of a float.
- bpo-39297 [https://bugs.python.org/issue?@action=redirect&bpo=39297]: Improved performance of importlib.metadata distribution discovery and resilients to inaccessible sys.path entries (importlib_metadata v1.4.0).
- bpo-39259 [https://bugs.python.org/issue?@action=redirect&bpo=39259]: **`NNTP`** and **`NNTP_SSL`** now raise a `ValueError` if the given timeout for their constructor is zero to prevent the creation of a non-blocking socket. Patch by Donghee Na.
- bpo-38901 [https://bugs.python.org/issue?@action=redirect&bpo=38901]: When you specify prompt=”.” or equivalently python -m venv –prompt . … the basename of the current directory is used to set the created venv’s prompt when it’s activated.
- bpo-39288 [https://bugs.python.org/issue?@action=redirect&bpo=39288]: Add `math.nextafter()`: return the next floating-point value after *x* towards *y*.
- bpo-39259 [https://bugs.python.org/issue?@action=redirect&bpo=39259]: `POP3` and `POP3_SSL` now raise a `ValueError` if the given timeout for their constructor is zero to prevent the creation of a non-blocking socket. Patch by Donghee Na.
- bpo-39242 [https://bugs.python.org/issue?@action=redirect&bpo=39242]: Updated the Gmane domain from news.gmane.org to news.gmane.io which is used for examples of **`NNTP`** news reader server and nntplib tests.
- bpo-35292 [https://bugs.python.org/issue?@action=redirect&bpo=35292]: Proxy the

`SimpleHTTPRequestHandler.guess_type` to `mimetypes.guess_type` so the `mimetypes.init` is called lazily to avoid unnecessary costs when **`http.server`** module is imported.

- bpo-39239 [https://bugs.python.org/issue?@action=redirect&bpo=39239]: The **`select.epoll.unregister()`** method no longer ignores the **`EBADF`** error.
- bpo-38907 [https://bugs.python.org/issue?@action=redirect&bpo=38907]: In http.server script, restore binding to IPv4 on Windows.
- bpo-39152 [https://bugs.python.org/issue?@action=redirect&bpo=39152]: Fix ttk.Scale.configure([name]) to return configuration tuple for name or all options. Giovanni Lombardo contributed part of the patch.
- bpo-39198 [https://bugs.python.org/issue?@action=redirect&bpo=39198]: If an exception were to be thrown in `Logger.isEnabledFor` (say, by asyncio timeouts or stopit) , the `logging` global lock may not be released appropriately, resulting in deadlock. This change wraps that block of code with `try...finally` to ensure the lock is released.
- bpo-39191 [https://bugs.python.org/issue?@action=redirect&bpo=39191]: Perform a check for running loop before starting a new task in `loop.run_until_complete()` to fail fast; it prevents the side effect of new task spawning before exception raising.
- bpo-38871 [https://bugs.python.org/issue?@action=redirect&bpo=38871]: Correctly parenthesize filter-based statements that contain lambda expressions in **`lib2to3`**. Patch by Donghee Na.
- bpo-39142 [https://bugs.python.org/issue?@action=redirect&bpo=39142]: A change was made to logging.config.dictConfig to avoid converting instances of named tuples to ConvertingTuple. It's assumed that named tuples are too specialised to be treated like ordinary tuples; if a user of named tuples requires ConvertingTuple functionality, they will have to implement that themselves in their named tuple class.
- bpo-39158 [https://bugs.python.org/issue?

@action=redirect&bpo=39158]: ast.literal_eval() now supports empty sets.

- bpo-39129 [https://bugs.python.org/issue?@action=redirect&bpo=39129]: Fix import path for `asyncio.TimeoutError`
- bpo-39057 [https://bugs.python.org/issue?@action=redirect&bpo=39057]: **`urllib.request.proxy_bypass_environment()`** now ignores leading dots and no longer ignores a trailing newline.
- bpo-39056 [https://bugs.python.org/issue?@action=redirect&bpo=39056]: Fixed handling invalid warning category in the -W option. No longer import the re module if it is not needed.
- bpo-39055 [https://bugs.python.org/issue?@action=redirect&bpo=39055]: **`base64.b64decode()`** with `validate=True` raises now a binascii.Error if the input ends with a single `\n`.
- bpo-21600 [https://bugs.python.org/issue?@action=redirect&bpo=21600]: Fix **`mock.patch.stopall()`** to stop active patches that were created with **`mock.patch.dict()`**.
- bpo-39019 [https://bugs.python.org/issue?@action=redirect&bpo=39019]: Implement dummy `__class_getitem__` for **`tempfile.SpooledTemporaryFile`**.
- bpo-39019 [https://bugs.python.org/issue?@action=redirect&bpo=39019]: Implement dummy `__class_getitem__` for `subprocess.Popen`, `subprocess.CompletedProcess`
- bpo-38914 [https://bugs.python.org/issue?@action=redirect&bpo=38914]: Adjusted the wording of the warning issued by distutils” `check` command when the `author` and `maintainer` fields are supplied but no corresponding e-mail field (`author_email` or `maintainer_email`) is found. The wording now reflects the fact that these fields are suggested, but not required. Patch by Juergen Gmach.
- bpo-38878 [https://bugs.python.org/issue?@action=redirect&bpo=38878]: Fixed _subclasshook_ of **`os.PathLike`** to return a correct result upon inheritance.

Patch by Bar Harel.

- bpo-38615 [https://bugs.python.org/issue?@action=redirect&bpo=38615]: `IMAP4` and `IMAP4_SSL` now have an optional *timeout* parameter for their constructors. Also, the `open()` method now has an optional *timeout* parameter with this change. The overridden methods of **`IMAP4_SSL`** and `IMAP4_stream` were applied to this change. Patch by Donghee Na.
- bpo-35182 [https://bugs.python.org/issue?@action=redirect&bpo=35182]: Fixed **`Popen.communicate()`** subsequent call crash when the child process has already closed any piped standard stream, but still continues to be running. Patch by Andriy Maletsky.
- bpo-38630 [https://bugs.python.org/issue?@action=redirect&bpo=38630]: On Unix, `subprocess.Popen.send_signal()` now polls the process status. Polling reduces the risk of sending a signal to the wrong process if the process completed, the `subprocess.Popen.returncode` attribute is still `None`, and the pid has been reassigned (recycled) to a new different process.
- bpo-38536 [https://bugs.python.org/issue?@action=redirect&bpo=38536]: Removes trailing space in formatted currency with `international=True` and a locale with symbol following value. E.g. `locale.currency(12.34, international=True)` returned `'12,34 EUR '` instead of `'12,34 EUR'`.
- bpo-38473 [https://bugs.python.org/issue?@action=redirect&bpo=38473]: Use signature from inner mock for autospecced methods attached with **`unittest.mock.attach_mock()`**. Patch by Karthikeyan Singaravelan.
- bpo-38361 [https://bugs.python.org/issue?@action=redirect&bpo=38361]: Fixed an issue where `ident` could include a leading path separator when `syslog.openlog()` was called without arguments.
- bpo-38293 [https://bugs.python.org/issue?@action=redirect&bpo=38293]: Add `copy.copy()` and `copy.deepcopy()` support to `property()` objects.
- bpo-37958 [https://bugs.python.org/issue?

@action=redirect&bpo=37958]: Added the pstats.Stats.get_profile_dict() method to return the profile data as a StatsProfile instance.
- bpo-28367 [https://bugs.python.org/issue?@action=redirect&bpo=28367]: Termios magic constants for the following baud rates: - B500000 - B576000 - B921600 - B1000000 - B1152000 - B1500000 - B2000000 - B2500000 - B3000000 - B3500000 - B4000000 Patch by Andrey Smirnov

Documentation

- bpo-39381 [https://bugs.python.org/issue?@action=redirect&bpo=39381]: Mention in docs that **`asyncio.get_event_loop()`** implicitly creates new event loop only if called from the main thread.
- bpo-38918 [https://bugs.python.org/issue?@action=redirect&bpo=38918]: Add an entry for `__module__` in the «function» & «method» sections of the **`inspect`** docs" Tipos y miembros table.
- bpo-3530 [https://bugs.python.org/issue?@action=redirect&bpo=3530]: In the **`ast`** module documentation, fix a misleading `NodeTransformer` example and add advice on when to use the `fix_missing_locations` function.

Build

- bpo-39395 [https://bugs.python.org/issue?@action=redirect&bpo=39395]: On non-Windows platforms, the **`setenv()`** and **`unsetenv()`** functions are now required to build Python.
- bpo-39160 [https://bugs.python.org/issue?@action=redirect&bpo=39160]: Updated the documentation in `./configure --help` to show default values, reference documentation where required and add additional explanation where needed.
- bpo-39144 [https://bugs.python.org/issue?@action=redirect&bpo=39144]: The ctags and etags build targets both include Modules/_ctypes and Python standard library source files.

IDLE

- bpo-39050 [https://bugs.python.org/issue?@action=redirect&bpo=39050]: Make IDLE Settings dialog Help button work again.
- bpo-34118 [https://bugs.python.org/issue?@action=redirect&bpo=34118]: Tag memoryview, range, and tuple as classes, the same as list, etcetera, in the library manual built-in functions list.
- bpo-32989 [https://bugs.python.org/issue?@action=redirect&bpo=32989]: Add tests for editor newline_and_indent_event method. Remove dead code from pyparse find_good_parse_start method.

C API

- bpo-39372 [https://bugs.python.org/issue?@action=redirect&bpo=39372]: Clean header files of interfaces defined but with no implementation. The public API symbols being removed are: `_PyBytes_InsertThousandsGroupingLocale`, `_PyBytes_InsertThousandsGrouping`, `_Py_InitializeFromArgs`, `_Py_InitializeFromWideArgs`, `_PyFloat_Repr`, `_PyFloat_Digits`, `_PyFloat_DigitsInit`, `PyFrame_ExtendStack`, `_PyAIterWrapper_Type`, `PyNullImporter_Type`, `PyCmpWrapper_Type`, `PySortWrapper_Type`, `PyNoArgsFunction`.
- bpo-39164 [https://bugs.python.org/issue?@action=redirect&bpo=39164]: Add a private `_PyErr_GetExcInfo()` function to retrieve exception information of the specified Python thread state.

Python 3.9.0 alpha 2

Release date: 2019-12-18

Security

- bpo-38945 [https://bugs.python.org/issue?@action=redirect&bpo=38945]: Newline characters have been escaped when performing uu encoding to prevent them from overflowing into to content section of the encoded file. This prevents malicious or accidental modification of data during the decoding process.
- bpo-37228 [https://bugs.python.org/issue?@action=redirect&bpo=37228]: Due to significant security concerns, the *reuse_address* parameter of **`asyncio.loop.create_datagram_endpoint()`** is no longer supported. This is because of the behavior of `SO_REUSEADDR` in UDP. For more details, see the documentation for `loop.create_datagram_endpoint()`. (Contributed by Kyle Stanley, Antoine Pitrou, and Yury Selivanov in bpo-37228 [https://bugs.python.org/issue?@action=redirect&bpo=37228].)
- bpo-38804 [https://bugs.python.org/issue?@action=redirect&bpo=38804]: Fixes a ReDoS vulnerability in **`http.cookiejar`**. Patch by Ben Caller.

Core and Builtins

- bpo-39028 [https://bugs.python.org/issue?@action=redirect&bpo=39028]: Slightly improve the speed of keyword argument parsing with many kwargs by strengthening the assumption that kwargs are interned strings.
- bpo-39080 [https://bugs.python.org/issue?@action=redirect&bpo=39080]: Fix the value of *end_col_offset* for Starred Expression AST nodes when they are among the elements in the *args* attribute of Call AST nodes.
- bpo-39031 [https://bugs.python.org/issue?@action=redirect&bpo=39031]: When parsing an «elif» node, lineno and col_offset of the node now point to the «elif» keyword and not to its condition, making it consistent with the «if» node. Patch by Lysandros Nikolaou.
- bpo-20443 [https://bugs.python.org/issue?@action=redirect&bpo=20443]: In Python 3.9.0a1, sys.argv[0] was made an absolute path if a filename was specified on the

command line. Revert this change, since most users expect sys.argv to be unmodified.

- bpo-39008 [https://bugs.python.org/issue?@action=redirect&bpo=39008]: **`PySys_Audit()`** now requires `Py_ssize_t` to be used for size arguments in the format string, regardless of whether `PY_SSIZE_T_CLEAN` was defined at include time.
- bpo-38673 [https://bugs.python.org/issue?@action=redirect&bpo=38673]: In REPL mode, don't switch to PS2 if the line starts with comment or whitespace. Based on work by Batuhan Taşkaya.
- bpo-38922 [https://bugs.python.org/issue?@action=redirect&bpo=38922]: Calling `replace` on a code object now raises the `code.__new__` audit event.
- bpo-38920 [https://bugs.python.org/issue?@action=redirect&bpo=38920]: Add audit hooks for when **`sys.excepthook()`** and **`sys.unraisablehook()`** are invoked.
- bpo-38892 [https://bugs.python.org/issue?@action=redirect&bpo=38892]: Improve documentation for audit events table and functions.
- bpo-38852 [https://bugs.python.org/issue?@action=redirect&bpo=38852]: Set the thread stack size to 8 Mb for debug builds on android platforms.
- bpo-38858 [https://bugs.python.org/issue?@action=redirect&bpo=38858]: Each Python subinterpreter now has its own «small integer singletons»: numbers in [-5; 257] range. It is no longer possible to change the number of small integers at build time by overriding `NSMALLNEGINTS` and `NSMALLPOSINTS` macros: macros should now be modified manually in `pycore_pystate.h` header file.
- bpo-36854 [https://bugs.python.org/issue?@action=redirect&bpo=36854]: The garbage collector state becomes per interpreter (`PyInterpreterState.gc`), rather than being global (`_PyRuntimeState.gc`).
- bpo-38835 [https://bugs.python.org/issue?@action=redirect&bpo=38835]: The `PyFPE_START_PROTECT()` and `PyFPE_END_PROTECT()` macros are empty: they have been doing nothing for the last year, so stop using them.
- bpo-38328 [https://bugs.python.org/issue?

@action=redirect&bpo=38328]: Sped up the creation time of constant **`list`** and **`set`** displays. Patch by Brandt Bucher.
- bpo-38707 [https://bugs.python.org/issue?@action=redirect&bpo=38707]: `MainThread.native_id` is now correctly reset in child processes spawned using **`multiprocessing.Process`**, instead of retaining the parent's value.
- bpo-38629 [https://bugs.python.org/issue?@action=redirect&bpo=38629]: Added `__floor__` and `__ceil__` methods to float object. Patch by Batuhan Taşkaya.
- bpo-27145 [https://bugs.python.org/issue?@action=redirect&bpo=27145]: int + int and int - int operators can now return small integer singletons. Patch by hongweipeng.
- bpo-38021 [https://bugs.python.org/issue?@action=redirect&bpo=38021]: Provide a platform tag for AIX that is sufficient for PEP425 binary distribution identification. Patch by Michael Felt.
- bpo-35409 [https://bugs.python.org/issue?@action=redirect&bpo=35409]: Ignore GeneratorExit exceptions when throwing an exception into the aclose coroutine of an asynchronous generator.
- bpo-33387 [https://bugs.python.org/issue?@action=redirect&bpo=33387]: Removed WITH_CLEANUP_START, WITH_CLEANUP_FINISH, BEGIN_FINALLY, END_FINALLY, CALL_FINALLY and POP_FINALLY bytecodes. Replaced with RERAISE and WITH_EXCEPT_START bytecodes. The compiler now generates different code for exceptional and non-exceptional branches for "with" and "try-except" statements. For "try-finally" statements the "finally" block is replicated for each exit from the "try" body.

Library

- bpo-39033 [https://bugs.python.org/issue?@action=redirect&bpo=39033]: Fix **`NameError`** in **`zipimport`**. Patch by Karthikeyan Singaravelan.

- bpo-39022 [https://bugs.python.org/issue?@action=redirect&bpo=39022]: Update importlib.metadata to include improvements from importlib_metadata 1.3 including better serialization of EntryPoints and improved documentation for custom finders.

- bpo-39006 [https://bugs.python.org/issue?@action=redirect&bpo=39006]: Fix asyncio when the ssl module is missing: only check for ssl.SSLSocket instance if the ssl module is available.

- bpo-38708 [https://bugs.python.org/issue?@action=redirect&bpo=38708]: Fix a potential IndexError in email parser when parsing an empty msg-id.

- bpo-38698 [https://bugs.python.org/issue?@action=redirect&bpo=38698]: Add a new `InvalidMessageID` token to email parser to represent invalid Message-ID headers. Also, add defects when there is remaining value after parsing the header.

- bpo-38994 [https://bugs.python.org/issue?@action=redirect&bpo=38994]: Implement `__class_getitem__` for `os.PathLike`, `pathlib.Path`.

- bpo-38979 [https://bugs.python.org/issue?@action=redirect&bpo=38979]: Return class from `ContextVar.__class_getitem__` to simplify subclassing.

- bpo-38978 [https://bugs.python.org/issue?@action=redirect&bpo=38978]: Implement `__class_getitem__` on asyncio objects (Future, Task, Queue). Patch by Batuhan Taskaya.

- bpo-38916 [https://bugs.python.org/issue?@action=redirect&bpo=38916]: **`array.array`**: Remove `tostring()` and `fromstring()` methods. They were aliases to `tobytes()` and `frombytes()`, deprecated since Python 3.2.

- bpo-38986 [https://bugs.python.org/issue?

@action=redirect&bpo=38986]: Make repr of C accelerated TaskWakeupMethWrapper the same as of pure Python version.

- bpo-38982 [https://bugs.python.org/issue?@action=redirect&bpo=38982]: Fix asyncio `PidfdChildWatcher`: handle `waitpid()` error. If `waitpid()` is called elsewhere, `waitpid()` call fails with **`ChildProcessError`**: use return code 255 in this case, and log a warning. It ensures that the pidfd file descriptor is closed if this error occurs.

- bpo-38529 [https://bugs.python.org/issue?@action=redirect&bpo=38529]: Drop too noisy asyncio warning about deletion of a stream without explicit `.close()` call.

- bpo-27413 [https://bugs.python.org/issue?@action=redirect&bpo=27413]: Added ability to pass through `ensure_ascii` options to json.dumps in the `json.tool` command-line interface.

- bpo-38634 [https://bugs.python.org/issue?@action=redirect&bpo=38634]: The **`readline`** module now detects if Python is linked to libedit at runtime on all platforms. Previously, the check was only done on macOS.

- bpo-33684 [https://bugs.python.org/issue?@action=redirect&bpo=33684]: Fix `json.tool` failed to read a JSON file with non-ASCII characters when locale encoding is not UTF-8.

- bpo-38698 [https://bugs.python.org/issue?@action=redirect&bpo=38698]: Prevent UnboundLocalError to pop up in parse_message_id.

  parse_message_id() was improperly using a token defined inside an exception handler, which was raising `UnboundLocalError` on parsing an invalid value. Patch by Claudiu Popa.

- bpo-38927 [https://bugs.python.org/issue?

@action=redirect&bpo=38927]: Use `python -m pip` instead of `pip` to upgrade dependencies in venv.

- bpo-26730 [https://bugs.python.org/issue?@action=redirect&bpo=26730]: Fix `SpooledTemporaryFile.rollover()` might corrupt the file when it is in text mode. Patch by Serhiy Storchaka.
- bpo-38881 [https://bugs.python.org/issue?@action=redirect&bpo=38881]: random.choices() now raises a ValueError when all the weights are zero.
- bpo-38876 [https://bugs.python.org/issue?@action=redirect&bpo=38876]: Raise pickle.UnpicklingError when loading an item from memo for invalid input.

  The previous code was raising a `KeyError` for both the Python and C implementation. This was caused by the specified index of an invalid input which did not exist in the memo structure, where the pickle stores what objects it has seen. The malformed input would have caused either a `BINGET` or `LONG_BINGET` load from the memo, leading to a `KeyError` as the determined index was bogus. Patch by Claudiu Popa.
- bpo-38688 [https://bugs.python.org/issue?@action=redirect&bpo=38688]: Calling func:`shutil.copytree` to copy a directory tree from one directory to another subdirectory resulted in an endless loop and a RecursionError. A fix was added to consume an iterator and create the list of the entries to be copied, avoiding the recursion for newly created directories. Patch by Bruno P. Kinoshita.
- bpo-38863 [https://bugs.python.org/issue?@action=redirect&bpo=38863]: Improve **`is_cgi()`** function in **`http.server`**, which enables processing the case that cgi directory is a child of another directory other than root.
- bpo-37838 [https://bugs.python.org/issue?@action=redirect&bpo=37838]: **`typing.get_type_hints()`** properly handles functions decorated with

`functools.wraps()`.

- bpo-38870 [https://bugs.python.org/issue?@action=redirect&bpo=38870]: Expose `ast.unparse()` as a function of the `ast` module that can be used to unparse an `ast.AST` object and produce a string with code that would produce an equivalent `ast.AST` object when parsed. Patch by Pablo Galindo and Batuhan Taskaya.

- bpo-38859 [https://bugs.python.org/issue?@action=redirect&bpo=38859]: AsyncMock now returns StopAsyncIteration on the exhaustion of a side_effects iterable. Since PEP-479 its Impossible to raise a StopIteration exception from a coroutine.

- bpo-38857 [https://bugs.python.org/issue?@action=redirect&bpo=38857]: AsyncMock fix for return values that are awaitable types. This also covers side_effect iterable values that happened to be awaitable, and wraps callables that return an awaitable type. Before these awaitables were being awaited instead of being returned as is.

- bpo-38834 [https://bugs.python.org/issue?@action=redirect&bpo=38834]: `typing.TypedDict` subclasses now track which keys are optional using the `__required_keys__` and `__optional_keys__` attributes, to enable runtime validation by downstream projects. Patch by Zac Hatfield-Dodds.

- bpo-38821 [https://bugs.python.org/issue?@action=redirect&bpo=38821]: Fix unhandled exceptions in `argparse` when internationalizing error messages for arguments with `nargs` set to special (non-integer) values. Patch by Federico Bond.

- bpo-38820 [https://bugs.python.org/issue?@action=redirect&bpo=38820]: Make Python compatible with OpenSSL 3.0.0. `ssl.SSLSocket.getpeercert()` no longer returns IPv6 addresses with a trailing new line.

- bpo-38811 [https://bugs.python.org/issue?

@action=redirect&bpo=38811]: Fix an unhandled exception in **`pathlib`** when **`os.link()`** is missing. Patch by Toke Høiland-Jørgensen.

- bpo-38686 [https://bugs.python.org/issue?@action=redirect&bpo=38686]: Added support for multiple `qop` values in **`urllib.request.AbstractDigestAuthHandler`**.

- bpo-38712 [https://bugs.python.org/issue?@action=redirect&bpo=38712]: Add the Linux-specific **`signal.pidfd_send_signal()`** function, which allows sending a signal to a process identified by a file descriptor rather than a pid.

- bpo-38348 [https://bugs.python.org/issue?@action=redirect&bpo=38348]: Add `-i` and `--indent` (indentation level), and `--no-type-comments` (type comments) command line options to ast parsing tool.

- bpo-37523 [https://bugs.python.org/issue?@action=redirect&bpo=37523]: Change **`zipfile.ZipExtFile`** to raise `ValueError` when trying to access the underlying file object after it has been closed. This new behavior is consistent with how accessing closed files is handled in other parts of Python.

- bpo-38045 [https://bugs.python.org/issue?@action=redirect&bpo=38045]: Improve the performance of **`enum._decompose()`** in **`enum`**. Patch by hongweipeng.

- bpo-36820 [https://bugs.python.org/issue?@action=redirect&bpo=36820]: Break cycle generated when saving an exception in socket.py, codeop.py and dyld.py as they keep alive not only the exception but user objects through the `__traceback__` attribute. Patch by Mario Corchero.

- bpo-36406 [https://bugs.python.org/issue?@action=redirect&bpo=36406]: Handle namespace packages in **`doctest`**. Patch by Karthikeyan Singaravelan.

- bpo-34776 [https://bugs.python.org/issue?@action=redirect&bpo=34776]: Fix dataclasses to support forward references in type annotations

- bpo-20928 [https://bugs.python.org/issue?@action=redirect&bpo=20928]: ElementTree supports recursive XInclude processing. Patch by Stefan Behnel.

- bpo-29636 [https://bugs.python.org/issue?@action=redirect&bpo=29636]: Add whitespace options for formatting JSON with the `json.tool` CLI. The following mutually exclusive options are now supported: `--indent` for setting the indent level in spaces; `--tab` for indenting with tabs; `--no-indent` for suppressing newlines; and `--compact` for suppressing all whitespace. The default behavior remains the same as `--indent=4`.

Documentation

- bpo-38928 [https://bugs.python.org/issue?@action=redirect&bpo=38928]: Correct when venv's `upgrade_dependencies()` and `--upgrade-deps` are added.
- bpo-38899 [https://bugs.python.org/issue?@action=redirect&bpo=38899]: Update documentation to state that to activate virtual environments under fish one should use `source`, not `.` as documented at https://fishshell.com/docs/current/cmds/source.html.
- bpo-22377 [https://bugs.python.org/issue?@action=redirect&bpo=22377]: Improves documentation of the values that **`datetime.datetime.strptime()`** accepts for `%Z`. Patch by Karl Dubost.

Tests

- bpo-38546 [https://bugs.python.org/issue?@action=redirect&bpo=38546]: Fix test_ressources_gced_in_workers() of test_concurrent_futures: explicitly stop the manager to prevent leaking a child process running in the background after the test completes.

- bpo-38546 [https://bugs.python.org/issue?@action=redirect&bpo=38546]: Multiprocessing and concurrent.futures tests now stop the resource tracker process when tests complete.
- bpo-38614 [https://bugs.python.org/issue?@action=redirect&bpo=38614]: Replace hardcoded timeout constants in tests with new `test.support` constants: `LOOPBACK_TIMEOUT`, `INTERNET_TIMEOUT`, `SHORT_TIMEOUT` and `LONG_TIMEOUT`. It becomes easier to adjust these four timeout constants for all tests at once, rather than having to adjust every single test file.
- bpo-38547 [https://bugs.python.org/issue?@action=redirect&bpo=38547]: Fix test_pty: if the process is the session leader, closing the master file descriptor raises a SIGHUP signal: simply ignore SIGHUP when running the tests.
- bpo-38992 [https://bugs.python.org/issue?@action=redirect&bpo=38992]: Fix a test for `math.fsum()` that was failing due to constant folding.
- bpo-38991 [https://bugs.python.org/issue?@action=redirect&bpo=38991]: `test.support`: **`run_python_until_end()`**, **`assert_python_ok()`** and **`assert_python_failure()`** functions no longer strip whitespaces from stderr. Remove `test.support.strip_python_stderr()` function.
- bpo-38965 [https://bugs.python.org/issue?@action=redirect&bpo=38965]: Fix test_faulthandler on GCC 10. Use the «volatile» keyword in `faulthandler._stack_overflow()` to prevent tail call optimization on any compiler, rather than relying on compiler specific pragma.
- bpo-38875 [https://bugs.python.org/issue?@action=redirect&bpo=38875]: test_capi: trashcan tests now require the test «cpu» resource.
- bpo-38841 [https://bugs.python.org/issue?@action=redirect&bpo=38841]: Skip asyncio test_create_datagram_endpoint_existing_sock_unix on platforms lacking a functional bind() for named unix domain sockets.
- bpo-38692 [https://bugs.python.org/issue?@action=redirect&bpo=38692]: Skip the

test_posix.test_pidfd_open() test if `os.pidfd_open()` fails with a `PermissionError`. This situation can happen in a Linux sandbox using a syscall whitelist which doesn't allow the `pidfd_open()` syscall yet.

- bpo-38839 [https://bugs.python.org/issue?@action=redirect&bpo=38839]: Fix some unused functions in tests. Patch by Adam Johnson.
- bpo-38669 [https://bugs.python.org/issue?@action=redirect&bpo=38669]: Raise `TypeError` when passing target as a string with `unittest.mock.patch.object()`.
- bpo-37957 [https://bugs.python.org/issue?@action=redirect&bpo=37957]: test.regrtest now can receive a list of test patterns to ignore (using the -i/–ignore argument) or a file with a list of patterns to ignore (using the –ignore-file argument). Patch by Pablo Galindo.

Build

- bpo-37404 [https://bugs.python.org/issue?@action=redirect&bpo=37404]: `asyncio` now raises `TypeError` when calling incompatible methods with an `ssl.SSLSocket` socket. Patch by Ido Michael.
- bpo-36500 [https://bugs.python.org/issue?@action=redirect&bpo=36500]: Added an optional «regen» project to the Visual Studio solution that will regenerate all grammar, tokens, and opcodes.

Windows

- bpo-39007 [https://bugs.python.org/issue?@action=redirect&bpo=39007]: Add auditing events to functions in `winreg`.
- bpo-33125 [https://bugs.python.org/issue?@action=redirect&bpo=33125]: Add support for building and releasing Windows ARM64 packages.

macOS

- bpo-37931 [https://bugs.python.org/issue?@action=redirect&bpo=37931]: Fixed a crash on OSX dynamic builds that occurred when re-initializing the posix module after a Py_Finalize if the environment had changed since the previous `import posix`. Patch by Benoît Hudson.

IDLE

- bpo-38944 [https://bugs.python.org/issue?@action=redirect&bpo=38944]: Escape key now closes IDLE completion windows. Patch by Johnny Najera.
- bpo-38943 [https://bugs.python.org/issue?@action=redirect&bpo=38943]: Fix IDLE autocomplete windows not always appearing on some systems. Patch by Johnny Najera.
- bpo-38862 [https://bugs.python.org/issue?@action=redirect&bpo=38862]: “Strip Trailing Whitespace” on the Format menu removes extra newlines at the end of non-shell files.
- bpo-38636 [https://bugs.python.org/issue?@action=redirect&bpo=38636]: Fix IDLE Format menu tab toggle and file indent width. These functions (default shortcuts Alt-T and Alt-U) were mistakenly disabled in 3.7.5 and 3.8.0.

C API

- bpo-38896 [https://bugs.python.org/issue?@action=redirect&bpo=38896]: Remove `PyUnicode_ClearFreeList()` function: the Unicode free list has been removed in Python 3.3.
- bpo-37340 [https://bugs.python.org/issue?@action=redirect&bpo=37340]: Remove `PyMethod_ClearFreeList()` and `PyCFunction_ClearFreeList()` functions: the free lists of bound method objects have been removed.
- bpo-38835 [https://bugs.python.org/issue?@action=redirect&bpo=38835]: Exclude `PyFPE_START_PROTECT()` and `PyFPE_END_PROTECT()` macros of `pyfpe.h` from `Py_LIMITED_API` (stable API).

Python 3.9.0 alpha 1

Release date: 2019-11-19

Security

- bpo-38722 [https://bugs.python.org/issue?@action=redirect&bpo=38722]: **`runpy`** now uses **`io.open_code()`** to open code files. Patch by Jason Killen.
- bpo-38622 [https://bugs.python.org/issue?@action=redirect&bpo=38622]: Add additional audit events for the **`ctypes`** module.
- bpo-38418 [https://bugs.python.org/issue?@action=redirect&bpo=38418]: Fixes audit event for **`os.system()`** to be named `os.system`.
- bpo-38243 [https://bugs.python.org/issue?@action=redirect&bpo=38243]: Escape the server title of **`xmlrpc.server.DocXMLRPCServer`** when rendering the document page as HTML. (Contributed by Donghee Na in bpo-38243 [https://bugs.python.org/issue?@action=redirect&bpo=38243].)
- bpo-38174 [https://bugs.python.org/issue?@action=redirect&bpo=38174]: Update vendorized expat library version to 2.2.8, which resolves **CVE 2019-15903** [https://www.cve.org/CVERecord?id=CVE-2019-15903].
- bpo-37764 [https://bugs.python.org/issue?@action=redirect&bpo=37764]: Fixes email._header_value_parser.get_unstructured going into an infinite loop for a specific case in which the email header does not have trailing whitespace, and the case in which it contains an invalid encoded word. Patch by Ashwin Ramaswami.
- bpo-37461 [https://bugs.python.org/issue?@action=redirect&bpo=37461]: Fix an infinite loop when parsing specially crafted email headers. Patch by Abhilash Raj.
- bpo-37363 [https://bugs.python.org/issue?@action=redirect&bpo=37363]: Adds audit events for the range of supported run commands (see Línea de comandos y entorno).
- bpo-37463 [https://bugs.python.org/issue?@action=redirect&bpo=37463]: ssl.match_hostname() no longer accepts IPv4 addresses with additional text after the address and only quad-dotted notation without trailing whitespaces.

Some inet_aton() implementations ignore whitespace and all data after whitespace, e.g. “127.0.0.1 whatever”.

- bpo-37363 [https://bugs.python.org/issue?@action=redirect&bpo=37363]: Adds audit events for `ensurepip`, `ftplib`, `glob`, `imaplib`, `nntplib`, `pdb`, `poplib`, `shutil`, `smtplib`, `sqlite3`, `subprocess`, `telnetlib`, `tempfile` and `webbrowser`, as well as `os.listdir()`, `os.scandir()` and `breakpoint()`.
- bpo-37364 [https://bugs.python.org/issue?@action=redirect&bpo=37364]: `io.open_code()` is now used when reading `.pth` files.
- bpo-34631 [https://bugs.python.org/issue?@action=redirect&bpo=34631]: Updated OpenSSL to 1.1.1c in Windows installer
- bpo-34155 [https://bugs.python.org/issue?@action=redirect&bpo=34155]: Fix parsing of invalid email addresses with more than one `@` (e.g. a@b@c.com.) to not return the part before 2nd `@` as valid email address. Patch by maxking & jpic.

Core and Builtins

- bpo-38631 [https://bugs.python.org/issue?@action=redirect&bpo=38631]: Replace `Py_FatalError()` call with a regular `RuntimeError` exception in `float.__getformat__()`.

- bpo-38639 [https://bugs.python.org/issue?@action=redirect&bpo=38639]: Optimized `math.floor()`, `math.ceil()` and `math.trunc()` for floats.

- bpo-38640 [https://bugs.python.org/issue?@action=redirect&bpo=38640]: Fixed a bug in the compiler that was causing to raise in the presence of break statements and continue statements inside always false while loops. Patch by Pablo Galindo.

- bpo-38613 [https://bugs.python.org/issue?@action=redirect&bpo=38613]: Optimized some set operations (e.g. `|`, `^`, and `-`) of `dict_keys`. `d.keys() | other` was slower than `set(d) | other` but they are almost same performance for now.

- bpo-28029 [https://bugs.python.org/issue?@action=redirect&bpo=28029]: `"".replace("", s, n)` now returns `s` instead of an empty string for all non-zero `n`. There are similar changes for **`bytes`** and **`bytearray`** objects.

- bpo-38535 [https://bugs.python.org/issue?@action=redirect&bpo=38535]: Fixed line numbers and column offsets for AST nodes for calls without arguments in decorators.

- bpo-38525 [https://bugs.python.org/issue?@action=redirect&bpo=38525]: Fix a segmentation fault when using reverse iterators of empty `dict` objects. Patch by Donghee Na and Inada Naoki.

- bpo-38465 [https://bugs.python.org/issue?@action=redirect&bpo=38465]: **`bytearray`**, **`array`** and **`mmap`** objects allow now to export more than `2**31` buffers at a time.

- bpo-38469 [https://bugs.python.org/issue?@action=redirect&bpo=38469]: Fixed a bug where the scope of named expressions was not being resolved correctly in the presence of the *global* keyword. Patch by Pablo Galindo.

- bpo-38437 [https://bugs.python.org/issue?@action=redirect&bpo=38437]: Activate the `GC_DEBUG` macro for debug builds of the interpreter (when `Py_DEBUG` is set). Patch by Pablo Galindo.

- bpo-38379 [https://bugs.python.org/issue?@action=redirect&bpo=38379]: When the garbage collector makes a collection in which some objects resurrect (they are reachable from outside the isolated cycles after the finalizers have been executed), do not block the collection of all objects that are still unreachable. Patch by Pablo Galindo and Tim Peters.

- bpo-38379 [https://bugs.python.org/issue?@action=redirect&bpo=38379]: When cyclic garbage collection (gc) runs finalizers that resurrect unreachable objects, the current gc run ends, without collecting any cyclic trash. However, the statistics reported by `collect()` and `get_stats()` claimed that all cyclic trash found was collected, and that the resurrected objects were collected. Changed the stats to report that none were collected.

- bpo-38392 [https://bugs.python.org/issue?@action=redirect&bpo=38392]: In debug mode, **`PyObject_GC_Track()`** now calls `tp_traverse()` of the object type to ensure that the object is valid: test that objects visited by `tp_traverse()` are valid.

- bpo-38210 [https://bugs.python.org/issue?@action=redirect&bpo=38210]: Remove unnecessary intersection and update set operation in dictview with empty set. (Contributed by Donghee Na in bpo-38210 [https://bugs.python.org/issue?@action=redirect&bpo=38210].)

- bpo-38402 [https://bugs.python.org/issue?@action=redirect&bpo=38402]: Check the error from the system's underlying `crypt` or `crypt_r`.

- bpo-37474 [https://bugs.python.org/issue?@action=redirect&bpo=37474]: On FreeBSD, Python no longer calls `fedisableexcept()` at startup to control the floating-point control mode. The call became useless since FreeBSD 6: it became the default mode.

- bpo-38006 [https://bugs.python.org/issue?@action=redirect&bpo=38006]: Fix a bug due to the interaction of weakrefs and the cyclic garbage collector. We must clear any weakrefs in garbage in order to prevent their callbacks from executing and causing a crash.

- bpo-38317 [https://bugs.python.org/issue?@action=redirect&bpo=38317]: Fix warnings options priority: `PyConfig.warnoptions` has the highest priority, as stated in the **PEP 587** [https://peps.python.org/pep-0587/].

- bpo-38310 [https://bugs.python.org/issue?@action=redirect&bpo=38310]: Predict `BUILD_MAP_UNPACK_WITH_CALL` -> `CALL_FUNCTION_EX` opcode pairs in the main interpreter loop. Patch by Brandt Bucher.

- bpo-36871 [https://bugs.python.org/issue?@action=redirect&bpo=36871]: Improve error handling for the assert_has_calls and assert_has_awaits methods of mocks. Fixed a bug where any errors encountered while binding the expected calls to the mock's spec were silently swallowed, leading to misleading error output.

- bpo-11410 [https://bugs.python.org/issue?@action=redirect&bpo=11410]: Better control over symbol visibility is provided through use of the visibility attributes available in gcc >= 4.0, provided in a uniform way across POSIX and Windows. The POSIX build files have been updated to compile with -fvisibility=hidden, minimising exported symbols.

- bpo-38219 [https://bugs.python.org/issue?@action=redirect&bpo=38219]: Optimized the `dict` constructor and the `update()` method for the case when the argument is a dict.

- bpo-38236 [https://bugs.python.org/issue?@action=redirect&bpo=38236]: Python now dumps path configuration if it fails to import the Python codecs of the filesystem and stdio encodings.

- bpo-38013 [https://bugs.python.org/issue?@action=redirect&bpo=38013]: Allow to call `async_generator_athrow().throw(...)` even for non-started async generator helper. It fixes annoying warning at the end of `asyncio.run()` call.

- bpo-38124 [https://bugs.python.org/issue?@action=redirect&bpo=38124]: Fix an off-by-one error in PyState_AddModule that could cause out-of-bounds memory access.

- bpo-38116 [https://bugs.python.org/issue?@action=redirect&bpo=38116]: The select module is now PEP-384 compliant and no longer has static state
- bpo-38113 [https://bugs.python.org/issue?@action=redirect&bpo=38113]: ast module updated to PEP-384 and all statics removed
- bpo-38076 [https://bugs.python.org/issue?@action=redirect&bpo=38076]: The struct module is now PEP-384 compatible
- bpo-38075 [https://bugs.python.org/issue?@action=redirect&bpo=38075]: The random module is now PEP-384 compatible
- bpo-38074 [https://bugs.python.org/issue?@action=redirect&bpo=38074]: zlib module made PEP-384 compatible
- bpo-38073 [https://bugs.python.org/issue?@action=redirect&bpo=38073]: Make pwd extension module PEP-384 compatible
- bpo-38072 [https://bugs.python.org/issue?@action=redirect&bpo=38072]: grp module made PEP-384 compatible
- bpo-38069 [https://bugs.python.org/issue?@action=redirect&bpo=38069]: Make _posixsubprocess PEP-384 compatible
- bpo-38071 [https://bugs.python.org/issue?@action=redirect&bpo=38071]: Make termios extension module PEP-384 compatible
- bpo-38005 [https://bugs.python.org/issue?@action=redirect&bpo=38005]: Fixed comparing and creating of InterpreterID and ChannelID.
- bpo-36946 [https://bugs.python.org/issue?@action=redirect&bpo=36946]: Fix possible signed integer

overflow when handling slices. Patch by hongweipeng.

- bpo-37994 [https://bugs.python.org/issue?@action=redirect&bpo=37994]: Fixed silencing arbitrary errors if an attribute lookup fails in several sites. Only AttributeError should be silenced.

- bpo-8425 [https://bugs.python.org/issue?@action=redirect&bpo=8425]: Optimize set difference_update for the case when the other set is much larger than the base set. (Suggested by Evgeny Kapun with code contributed by Michele Orrù).

- bpo-37966 [https://bugs.python.org/issue?@action=redirect&bpo=37966]: The implementation of `is_normalized()` has been greatly sped up on strings that aren't normalized, by implementing the full normalization-quick-check algorithm from the Unicode standard.

- bpo-37947 [https://bugs.python.org/issue?@action=redirect&bpo=37947]: Adjust correctly the recursion level in the symtable generation for named expressions. Patch by Pablo Galindo.

- bpo-37812 [https://bugs.python.org/issue?@action=redirect&bpo=37812]: The `CHECK_SMALL_INT` macro used inside `Object/longobject.c` has been replaced with an explicit `return` at each call site.

- bpo-37751 [https://bugs.python.org/issue?@action=redirect&bpo=37751]: Fix `codecs.lookup()` to normalize the encoding name the same way than `encodings.normalize_encoding()`, except that `codecs.lookup()` also converts the name to lower case.

- bpo-37830 [https://bugs.python.org/issue?@action=redirect&bpo=37830]: Fixed compilation of `break` and `continue` in the `finally` block when the corresponding `try` block contains `return` with a non-constant value.

- bpo-20490 [https://bugs.python.org/issue?@action=redirect&bpo=20490]: Improve import error message for partially initialized module on circular `from` imports - by Anthony Sottile.

- bpo-37840 [https://bugs.python.org/issue?@action=redirect&bpo=37840]: Fix handling of negative indices in **`sq_item`** of **`bytearray`**. Patch by Sergey Fedoseev.
- bpo-37802 [https://bugs.python.org/issue?@action=redirect&bpo=37802]: Slightly improve performance of **`PyLong_FromUnsignedLong()`**, **`PyLong_FromUnsignedLongLong()`** and **`PyLong_FromSize_t()`**. Patch by Sergey Fedoseev.
- bpo-37409 [https://bugs.python.org/issue?@action=redirect&bpo=37409]: Ensure explicit relative imports from interactive sessions and scripts (having no parent package) always raise ImportError, rather than treating the current module as the package. Patch by Ben Lewis.
- bpo-32912 [https://bugs.python.org/issue?@action=redirect&bpo=32912]: Reverted bpo-32912 [https://bugs.python.org/issue?@action=redirect&bpo=32912]: emitting **`SyntaxWarning`** instead of **`DeprecationWarning`** for invalid escape sequences in string and bytes literals.
- bpo-37757 [https://bugs.python.org/issue?@action=redirect&bpo=37757]: **PEP 572** [https://peps.python.org/pep-0572/]: As described in the PEP, assignment expressions now raise **`SyntaxError`** when their interaction with comprehension scoping results in an ambiguous target scope.

  The `TargetScopeError` subclass originally proposed by the PEP has been removed in favour of just raising regular syntax errors for the disallowed cases.
- bpo-36279 [https://bugs.python.org/issue?@action=redirect&bpo=36279]: Fix potential use of uninitialized memory in **`os.wait3()`**.
- bpo-36311 [https://bugs.python.org/issue?@action=redirect&bpo=36311]: Decoding bytes objects larger

than 2GiB is faster and no longer fails when a multibyte characters spans a chunk boundary.

- bpo-34880 [https://bugs.python.org/issue?@action=redirect&bpo=34880]: The **`assert`** statement now works properly if the **`AssertionError`** exception is being shadowed. Patch by Zackery Spytz.

- bpo-37340 [https://bugs.python.org/issue?@action=redirect&bpo=37340]: Removed object cache (`free_list`) for bound method objects. Temporary bound method objects are less used than before thanks to the `LOAD_METHOD` opcode and the `_PyObject_VectorcallMethod` C API.

- bpo-37648 [https://bugs.python.org/issue?@action=redirect&bpo=37648]: Fixed minor inconsistency in **`list.__contains__()`**, **`tuple.__contains__()`** and a few other places. The collection's item is now always at the left and the needle is on the right of `==`.

- bpo-37444 [https://bugs.python.org/issue?@action=redirect&bpo=37444]: Update differing exception between **`builtins.__import__()`** and **`importlib.__import__()`**.

- bpo-37619 [https://bugs.python.org/issue?@action=redirect&bpo=37619]: When adding a wrapper descriptor from one class to a different class (for example, setting `__add__ = str.__add__` on an `int` subclass), an exception is correctly raised when the operator is called.

- bpo-37593 [https://bugs.python.org/issue?@action=redirect&bpo=37593]: Swap the positions of the *posonlyargs* and *args* parameters in the constructor of **`ast.parameters`** nodes.

- bpo-37543 [https://bugs.python.org/issue?@action=redirect&bpo=37543]: Optimized pymalloc for non PGO build.

- bpo-37537 [https://bugs.python.org/issue?@action=redirect&bpo=37537]: Compute allocated pymalloc blocks inside _Py_GetAllocatedBlocks(). This slows down _Py_GetAllocatedBlocks() but gives a small speedup to _PyObject_Malloc() and _PyObject_Free().

- bpo-37467 [https://bugs.python.org/issue?@action=redirect&bpo=37467]: Fix **`sys.excepthook()`** and **`PyErr_Display()`** if a filename is a bytes string. For example, for a SyntaxError exception where the filename attribute is a bytes string.

- bpo-37433 [https://bugs.python.org/issue?@action=redirect&bpo=37433]: Fix `SyntaxError` indicator printing too many spaces for multi-line strings - by Anthony Sottile.

- bpo-37417 [https://bugs.python.org/issue?@action=redirect&bpo=37417]: **`bytearray.extend()`** now correctly handles errors that arise during iteration. Patch by Brandt Bucher.

- bpo-37414 [https://bugs.python.org/issue?@action=redirect&bpo=37414]: The undocumented `sys.callstats()` function has been removed. Since Python 3.7, it was deprecated and always returned `None`. It required a special build option `CALL_PROFILE` which was already removed in Python 3.7.

- bpo-37392 [https://bugs.python.org/issue?@action=redirect&bpo=37392]: Remove `sys.getcheckinterval()` and `sys.setcheckinterval()` functions. They were deprecated since Python 3.2. Use **`sys.getswitchinterval()`** and **`sys.setswitchinterval()`** instead. Remove also `check_interval` field of the `PyInterpreterState` structure.

- bpo-37388 [https://bugs.python.org/issue?@action=redirect&bpo=37388]: In development mode and in debug build, *encoding* and *errors* arguments are now checked on string encoding and decoding operations. Examples: **`open()`**, **`str.encode()`** and **`bytes.decode()`**.

By default, for best performances, the *errors* argument is only checked at the first encoding/decoding error, and the *encoding* argument is sometimes ignored for empty strings.

- bpo-37348 [https://bugs.python.org/issue?@action=redirect&bpo=37348]: Optimized decoding short ASCII string with UTF-8 and ascii codecs. `b"foo".decode()` is about 15% faster. Patch by Inada Naoki.
- bpo-24214 [https://bugs.python.org/issue?@action=redirect&bpo=24214]: Improved support of the surrogatepass error handler in the UTF-8 and UTF-16 incremental decoders.
- bpo-37330 [https://bugs.python.org/issue?@action=redirect&bpo=37330]: **`open()`**, **`io.open()`**, **`codecs.open()`** and **`fileinput.FileInput`** no longer accept `'U'` («universal newline») in the file mode. This flag was deprecated since Python 3.3.
- bpo-35224 [https://bugs.python.org/issue?@action=redirect&bpo=35224]: Reverse evaluation order of key: value in dict comprehensions as proposed in PEP 572. I.e. in `{k: v for ...}`, `k` will be evaluated before `v`.
- bpo-37316 [https://bugs.python.org/issue?@action=redirect&bpo=37316]: Fix the **`PySys_Audit()`** call in **`mmap.mmap`**.
- bpo-37300 [https://bugs.python.org/issue?@action=redirect&bpo=37300]: Remove an unnecessary Py_XINCREF in classobject.c.
- bpo-37269 [https://bugs.python.org/issue?@action=redirect&bpo=37269]: Fix a bug in the peephole optimizer that was not treating correctly constant conditions with binary operators. Patch by Pablo Galindo.

- bpo-20443 [https://bugs.python.org/issue?@action=redirect&bpo=20443]: Python now gets the absolute path of the script filename specified on the command line (ex: «python3 script.py»): the _file_ attribute of the _main_ module and sys.path[0] become an absolute path, rather than a relative path.

- bpo-37257 [https://bugs.python.org/issue?@action=redirect&bpo=37257]: Python's small object allocator (`obmalloc.c`) now allows (no more than) one empty arena to remain available for immediate reuse, without returning it to the OS. This prevents thrashing in simple loops where an arena could be created and destroyed anew on each iteration.

- bpo-37231 [https://bugs.python.org/issue?@action=redirect&bpo=37231]: The dispatching of type slots to special methods (for example calling `__mul__` when doing `x * y`) has been made faster.

- bpo-36974 [https://bugs.python.org/issue?@action=redirect&bpo=36974]: Implemented separate vectorcall functions for every calling convention of builtin functions and methods. This improves performance for calls.

- bpo-37213 [https://bugs.python.org/issue?@action=redirect&bpo=37213]: Handle correctly negative line offsets in the peephole optimizer. Patch by Pablo Galindo.

- bpo-37219 [https://bugs.python.org/issue?@action=redirect&bpo=37219]: Remove erroneous optimization for empty set differences.

- bpo-15913 [https://bugs.python.org/issue?@action=redirect&bpo=15913]: Implement **`PyBuffer_SizeFromFormat()`** function (previously documented but not implemented): call **`struct.calcsize()`**. Patch by Joannah Nanjekye.

- bpo-36922 [https://bugs.python.org/issue?@action=redirect&bpo=36922]: Slot functions optimize any callable with `Py_TPFLAGS_METHOD_DESCRIPTOR` instead of only instances of `function`.

- bpo-36974 [https://bugs.python.org/issue?@action=redirect&bpo=36974]: The slot `tp_vectorcall_offset` is inherited unconditionally to support `super().__call__()` when the base class uses vectorcall.

- bpo-37160 [https://bugs.python.org/issue?@action=redirect&bpo=37160]: **`threading.get_native_id()`** now also supports NetBSD.

- bpo-37077 [https://bugs.python.org/issue?@action=redirect&bpo=37077]: Add **`threading.get_native_id()`** support for AIX. Patch by M. Felt

- bpo-36781 [https://bugs.python.org/issue?@action=redirect&bpo=36781]: **`sum()`** has been optimized for boolean values.

- bpo-34556 [https://bugs.python.org/issue?@action=redirect&bpo=34556]: Add `--upgrade-deps` to venv module. Patch by Cooper Ry Lees

- bpo-20523 [https://bugs.python.org/issue?@action=redirect&bpo=20523]: `pdb.Pdb` supports ~/.pdbrc in Windows 7. Patch by Tim Hopper and Dan Lidral-Porter.

- bpo-35551 [https://bugs.python.org/issue?@action=redirect&bpo=35551]: Updated encodings: - Removed the «tis260» encoding, which was an alias for the nonexistent «tactis» codec. - Added «mac_centeuro» as an alias for the mac_latin2 encoding.

- bpo-19072 [https://bugs.python.org/issue?@action=redirect&bpo=19072]: The **`classmethod`** decorator can now wrap other descriptors such as property objects. Adapted from a patch written by Graham Dumpleton.

- bpo-27575 [https://bugs.python.org/issue?

@action=redirect&bpo=27575]: Improve speed of dictview intersection by directly using set intersection logic. Patch by David Su.

- bpo-30773 [https://bugs.python.org/issue?@action=redirect&bpo=30773]: Prohibit parallel running of aclose() / asend() / athrow(). Fix ag_running to reflect the actual running status of the AG.

Library

- bpo-36589 [https://bugs.python.org/issue?@action=redirect&bpo=36589]: The **`curses.update_lines_cols()`** function now returns `None` instead of `1` on success.
- bpo-38807 [https://bugs.python.org/issue?@action=redirect&bpo=38807]: Update **`TypeError`** messages for **`os.path.join()`** to include **`os.PathLike`** objects as acceptable input types.
- bpo-38724 [https://bugs.python.org/issue?@action=redirect&bpo=38724]: Add a repr for `subprocess.Popen` objects. Patch by Andrey Doroschenko.
- bpo-38786 [https://bugs.python.org/issue?@action=redirect&bpo=38786]: pydoc now recognizes and parses HTTPS URLs. Patch by python273.
- bpo-38785 [https://bugs.python.org/issue?@action=redirect&bpo=38785]: Prevent asyncio from crashing if parent `__init__` is not called from a constructor of object derived from `asyncio.Future`.
- bpo-38723 [https://bugs.python.org/issue?@action=redirect&bpo=38723]: **`pdb`** now uses **`io.open_code()`** to trigger auditing events.
- bpo-27805 [https://bugs.python.org/issue?@action=redirect&bpo=27805]: Allow opening pipes and other non-seekable files in append mode with **`open()`**.

- bpo-38438 [https://bugs.python.org/issue?@action=redirect&bpo=38438]: Simplify the `argparse` usage message for `nargs="*"`.

- bpo-38761 [https://bugs.python.org/issue?@action=redirect&bpo=38761]: WeakSet is now registered as a collections.abc.MutableSet.

- bpo-38716 [https://bugs.python.org/issue?@action=redirect&bpo=38716]: logging: change RotatingHandler namer and rotator to class-level attributes. This stops _init_ from setting them to None in the case where a subclass defines them with eponymous methods.

- bpo-38713 [https://bugs.python.org/issue?@action=redirect&bpo=38713]: Add `os.P_PIDFD` constant, which may be passed to `os.waitid()` to wait on a Linux process file descriptor.

- bpo-38692 [https://bugs.python.org/issue?@action=redirect&bpo=38692]: Add **`asyncio.PidfdChildWatcher`**, a Linux-specific child watcher implementation that polls process file descriptors.

- bpo-38692 [https://bugs.python.org/issue?@action=redirect&bpo=38692]: Expose the Linux `pidfd_open` syscall as `os.pidfd_open()`.

- bpo-38602 [https://bugs.python.org/issue?@action=redirect&bpo=38602]: Added constants **`F_OFD_GETLK`**, **`F_OFD_SETLK`** and **`F_OFD_SETLKW`** to the `fcntl` module. Patch by Donghee Na.

- bpo-38334 [https://bugs.python.org/issue?@action=redirect&bpo=38334]: Fixed seeking backward on an encrypted **`zipfile.ZipExtFile`**.

- bpo-38312 [https://bugs.python.org/issue?@action=redirect&bpo=38312]: Add `curses.get_escdelay()`, `curses.set_escdelay()`, `curses.get_tabsize()`, and `curses.set_tabsize()` functions - by Anthony

Sottile.

- bpo-38586 [https://bugs.python.org/issue?@action=redirect&bpo=38586]: Now `fileConfig()` correctly sets the .name of handlers loaded.
- bpo-38565 [https://bugs.python.org/issue?@action=redirect&bpo=38565]: Add new cache_parameters() method for functools.lru_cache() to better support pickling.
- bpo-34679 [https://bugs.python.org/issue?@action=redirect&bpo=34679]: asynci.ProactorEventLoop.close() now only calls signal.set_wakeup_fd() in the main thread.
- bpo-31202 [https://bugs.python.org/issue?@action=redirect&bpo=31202]: The case the result of **`pathlib.WindowsPath.glob()`** matches now the case of the pattern for literal parts.
- bpo-36321 [https://bugs.python.org/issue?@action=redirect&bpo=36321]: Remove misspelled attribute. The 3.8 changelog noted that this would be removed in 3.9.
- bpo-38521 [https://bugs.python.org/issue?@action=redirect&bpo=38521]: Fixed erroneous equality comparison in statistics.NormalDist().
- bpo-38493 [https://bugs.python.org/issue?@action=redirect&bpo=38493]: Added `CLD_KILLED` and `CLD_STOPPED` for **`si_code`**. Patch by Donghee Na.
- bpo-38478 [https://bugs.python.org/issue?@action=redirect&bpo=38478]: Fixed a bug in **`inspect.signature.bind()`** that was causing it to fail when handling a keyword argument with same name as positional-only parameter. Patch by Pablo Galindo.
- bpo-33604 [https://bugs.python.org/issue?@action=redirect&bpo=33604]: Fixed `hmac.new` and `hmac.HMAC` to raise TypeError instead of ValueError when the digestmod parameter, now required in 3.8, is omitted.

Also clarified the hmac module documentation and docstrings.

- bpo-38378 [https://bugs.python.org/issue?@action=redirect&bpo=38378]: Parameters *out* and *in* of `os.sendfile()` was renamed to *out_fd* and *in_fd*.

- bpo-38417 [https://bugs.python.org/issue?@action=redirect&bpo=38417]: Added support for setting the umask in the child process to the subprocess module on POSIX systems.

- bpo-38449 [https://bugs.python.org/issue?@action=redirect&bpo=38449]: Revert PR 15522, which introduces a regression in `mimetypes.guess_type()` due to improper handling of filenames as urls.

- bpo-38431 [https://bugs.python.org/issue?@action=redirect&bpo=38431]: Fix `__repr__` method for `dataclasses.InitVar` to support typing objects, patch by Samuel Colvin.

- bpo-38109 [https://bugs.python.org/issue?@action=redirect&bpo=38109]: Add missing `stat.S_IFDOOR`, `stat.S_IFPORT`, `stat.S_IFWHT`, `stat.S_ISDOOR()`, `stat.S_ISPORT()`, and `stat.S_ISWHT()` values to the Python implementation of `stat`.

- bpo-38422 [https://bugs.python.org/issue?@action=redirect&bpo=38422]: Clarify docstrings of pathlib suffix(es)

- bpo-38405 [https://bugs.python.org/issue?@action=redirect&bpo=38405]: Nested subclasses of `typing.NamedTuple` are now pickleable.

- bpo-38332 [https://bugs.python.org/issue?@action=redirect&bpo=38332]: Prevent `KeyError` thrown by `_encoded_words.decode()` when given an encoded-word with invalid content-type encoding from propagating all the way to `email.message.get()`.

- bpo-38371 [https://bugs.python.org/issue?@action=redirect&bpo=38371]: Deprecated the `split()` method in **`_tkinter.TkappType`** in favour of the `splitlist()` method which has more consistent and predictable behavior.

- bpo-38341 [https://bugs.python.org/issue?@action=redirect&bpo=38341]: Add **`smtplib.SMTPNotSupportedError`** to the **`smtplib`** exported names.

- bpo-38319 [https://bugs.python.org/issue?@action=redirect&bpo=38319]: sendfile() used in socket and shutil modules was raising OverflowError for files >= 2GiB on 32-bit architectures. (patch by Giampaolo Rodola)

- bpo-38242 [https://bugs.python.org/issue?@action=redirect&bpo=38242]: Revert the new asyncio Streams API

- bpo-13153 [https://bugs.python.org/issue?@action=redirect&bpo=13153]: OS native encoding is now used for converting between Python strings and Tcl objects. This allows to display, copy and paste to clipboard emoji and other non-BMP characters. Converting strings from Tcl to Python and back now never fails (except MemoryError).

- bpo-38019 [https://bugs.python.org/issue?@action=redirect&bpo=38019]: Correctly handle pause/resume reading of closed asyncio unix pipe.

- bpo-38163 [https://bugs.python.org/issue?@action=redirect&bpo=38163]: Child mocks will now detect their type as either synchronous or asynchronous, asynchronous child mocks will be AsyncMocks and synchronous child mocks will be either MagicMock or Mock (depending on their parent type).

- bpo-38161 [https://bugs.python.org/issue?@action=redirect&bpo=38161]: Removes _AwaitEvent from AsyncMock.

- bpo-38216 [https://bugs.python.org/issue?@action=redirect&bpo=38216]: Allow the rare code that wants to send invalid http requests from the `http.client` library a way to do so. The fixes for bpo-30458 [https://bugs.python.org/issue?@action=redirect&bpo=30458] led to breakage for some projects that were relying on this ability to test their own behavior in the face of bad requests.

- bpo-28286 [https://bugs.python.org/issue?@action=redirect&bpo=28286]: Deprecate opening **`GzipFile`** for writing implicitly. Always specify the *mode* argument for writing.

- bpo-38108 [https://bugs.python.org/issue?@action=redirect&bpo=38108]: Any synchronous magic methods on an AsyncMock now return a MagicMock. Any asynchronous magic methods on a MagicMock now return an AsyncMock.

- bpo-38265 [https://bugs.python.org/issue?@action=redirect&bpo=38265]: Update the *length* parameter of **`os.pread()`** to accept **`Py_ssize_t`** instead of int.

- bpo-38112 [https://bugs.python.org/issue?@action=redirect&bpo=38112]: **`compileall`** has a higher default recursion limit and new command-line arguments for path manipulation, symlinks handling, and multiple optimization levels.

- bpo-38248 [https://bugs.python.org/issue?@action=redirect&bpo=38248]: asyncio: Fix inconsistent immediate Task cancellation

- bpo-38237 [https://bugs.python.org/issue?@action=redirect&bpo=38237]: The arguments for the builtin pow function are more descriptive. They can now also be passed in as keywords.

- bpo-34002 [https://bugs.python.org/issue?@action=redirect&bpo=34002]: Improve efficiency in parts of email package by changing while-pop to a for loop, using isdisjoint instead of set intersections.

- bpo-38191 [https://bugs.python.org/issue?@action=redirect&bpo=38191]: Constructors of **`NamedTuple`** and **`TypedDict`** types now accept arbitrary keyword argument names, including «cls», «self», «typename», «_typename», «fields» and «_fields».

- bpo-38155 [https://bugs.python.org/issue?@action=redirect&bpo=38155]: Add `__all__` to **`datetime`**. Patch by Tahia Khan.

- bpo-38185 [https://bugs.python.org/issue?@action=redirect&bpo=38185]: Fixed case-insensitive string comparison in **`sqlite3.Row`** indexing.

- bpo-38136 [https://bugs.python.org/issue?@action=redirect&bpo=38136]: Changes AsyncMock call count and await count to be two different counters. Now await count only counts when a coroutine has been awaited, not when it has been called, and vice-versa. Update the documentation around this.

- bpo-37828 [https://bugs.python.org/issue?@action=redirect&bpo=37828]: Fix default mock name in **`unittest.mock.Mock.assert_called()`** exceptions. Patch by Abraham Toriz Cruz.

- bpo-38175 [https://bugs.python.org/issue?@action=redirect&bpo=38175]: Fix a memory leak in comparison of **`sqlite3.Row`** objects.

- bpo-33936 [https://bugs.python.org/issue?@action=redirect&bpo=33936]: _hashlib no longer calls obsolete OpenSSL initialization function with OpenSSL 1.1.0+.

- bpo-34706 [https://bugs.python.org/issue?@action=redirect&bpo=34706]: Preserve subclassing in inspect.Signature.from_callable.

- bpo-38153 [https://bugs.python.org/issue?

@action=redirect&bpo=38153]: Names of hashing algorithms from OpenSSL are now normalized to follow Python’s naming conventions. For example OpenSSL uses sha3-512 instead of sha3_512 or blake2b512 instead of blake2b.

- bpo-38115 [https://bugs.python.org/issue?@action=redirect&bpo=38115]: Fix a bug in dis.findlinestarts() where it would return invalid bytecode offsets. Document that a code object’s co_lnotab can contain invalid bytecode offsets.

- bpo-38148 [https://bugs.python.org/issue?@action=redirect&bpo=38148]: Add slots to `asyncio` transport classes, which can reduce memory usage.

- bpo-38142 [https://bugs.python.org/issue?@action=redirect&bpo=38142]: The _hashlib OpenSSL wrapper extension module is now PEP-384 compliant.

- bpo-9216 [https://bugs.python.org/issue?@action=redirect&bpo=9216]: hashlib constructors now support usedforsecurity flag to signal that a hashing algorithm is not used in a security context.

- bpo-36991 [https://bugs.python.org/issue?@action=redirect&bpo=36991]: Fixes a potential incorrect AttributeError exception escaping ZipFile.extract() in some unsupported input error situations.

- bpo-38134 [https://bugs.python.org/issue?@action=redirect&bpo=38134]: Remove obsolete copy of PBKDF2_HMAC_fast. All supported OpenSSL versions contain a fast implementation.

- bpo-38132 [https://bugs.python.org/issue?@action=redirect&bpo=38132]: The OpenSSL hashlib wrapper uses a simpler implementation. Several Macros and pointless caches are gone. The hash name now comes from OpenSSL’s EVP. The algorithm name stays the same, except it is now always lower case.

- bpo-38008 [https://bugs.python.org/issue?@action=redirect&bpo=38008]: Fix parent class check in protocols to correctly identify the module that provides a builtin protocol, instead of assuming they all come from the `collections.abc` module

- bpo-34037 [https://bugs.python.org/issue?@action=redirect&bpo=34037]: For `asyncio`, add a new coroutine `loop.shutdown_default_executor()`. The new coroutine provides an API to schedule an executor shutdown that waits on the threadpool to finish closing. Also, `asyncio.run()` has been updated to utilize the new coroutine. Patch by Kyle Stanley.

- bpo-37405 [https://bugs.python.org/issue?@action=redirect&bpo=37405]: Fixed regression bug for socket.getsockname() for non-CAN_ISOTP AF_CAN address family sockets by returning a 1-tuple instead of string.

- bpo-38121 [https://bugs.python.org/issue?@action=redirect&bpo=38121]: Update parameter names on functions in importlib.metadata matching the changes in the 0.22 release of importlib_metadata.

- bpo-38110 [https://bugs.python.org/issue?@action=redirect&bpo=38110]: The os.closewalk() implementation now uses the libc fdwalk() API on platforms where it is available.

- bpo-38093 [https://bugs.python.org/issue?@action=redirect&bpo=38093]: Fixes AsyncMock so it doesn't crash when used with AsyncContextManagers or AsyncIterators.

- bpo-37488 [https://bugs.python.org/issue?@action=redirect&bpo=37488]: Add warning to `datetime.utctimetuple()`, `datetime.utcnow()` and `datetime.utcfromtimestamp()` .

- bpo-35640 [https://bugs.python.org/issue?@action=redirect&bpo=35640]: Allow passing a path-like object as `directory` argument to the **`http.server.SimpleHTTPRequestHandler`** class. Patch by Géry Ogam.

- bpo-38086 [https://bugs.python.org/issue?@action=redirect&bpo=38086]: Update importlib.metadata with changes from importlib_metadata 0.21 [https://gitlab.com/python-devs/importlib_metadata/blob/0.21/importlib_metadata/docs/changelog.rst].

- bpo-37251 [https://bugs.python.org/issue?@action=redirect&bpo=37251]: Remove `__code__` check in AsyncMock that incorrectly evaluated function specs as async objects but failed to evaluate classes with `__await__` but no `__code__` attribute defined as async objects.

- bpo-38037 [https://bugs.python.org/issue?@action=redirect&bpo=38037]: Fix reference counters in the **`signal`** module.

- bpo-38066 [https://bugs.python.org/issue?@action=redirect&bpo=38066]: Hide internal asyncio.Stream methods: feed_eof(), feed_data(), set_exception() and set_transport().

- bpo-38059 [https://bugs.python.org/issue?@action=redirect&bpo=38059]: inspect.py now uses sys.exit() instead of exit()

- bpo-38049 [https://bugs.python.org/issue?@action=redirect&bpo=38049]: Added command-line interface for the **`ast`** module.

- bpo-37953 [https://bugs.python.org/issue?@action=redirect&bpo=37953]: In **`typing`**, improved the `__hash__` and `__eq__` methods for **`ForwardReferences`**.

- bpo-38026 [https://bugs.python.org/issue?@action=redirect&bpo=38026]: Fixed **`inspect.getattr_static()`** used `isinstance` while it should avoid dynamic lookup.

- bpo-35923 [https://bugs.python.org/issue?@action=redirect&bpo=35923]: Update **`importlib.machinery.BuiltinImporter`** to use `loader._ORIGIN` instead of a hardcoded value. Patch by Donghee Na.

- bpo-38010 [https://bugs.python.org/issue?@action=redirect&bpo=38010]: In `importlib.metadata` sync with `importlib_metadata` 0.20, clarifying behavior of `files()` and fixing issue where only one requirement was returned for `requires()` on `dist-info` packages.

- bpo-38006 [https://bugs.python.org/issue?@action=redirect&bpo=38006]: weakref.WeakValueDictionary defines a local remove() function used as callback for weak references. This function was created with a closure. Modify the implementation to avoid the closure.

- bpo-37995 [https://bugs.python.org/issue?@action=redirect&bpo=37995]: Added the *indent* option to **`ast.dump()`** which allows it to produce a multiline indented output.

- bpo-34410 [https://bugs.python.org/issue?@action=redirect&bpo=34410]: Fixed a crash in the **`tee()`** iterator when re-enter it. RuntimeError is now raised in this case.

- bpo-37140 [https://bugs.python.org/issue?@action=redirect&bpo=37140]: Fix a ctypes regression of Python 3.8. When a ctypes.Structure is passed by copy to a function, ctypes internals created a temporary object which had the side effect of calling the structure finalizer (_del_) twice. The Python semantics requires a finalizer to be called exactly once. Fix ctypes internals to no longer call the finalizer twice.

- bpo-37587 [https://bugs.python.org/issue?@action=redirect&bpo=37587]: `_json.scanstring` is now up to 3x faster when there are many backslash escaped characters in the JSON string.

- bpo-37834 [https://bugs.python.org/issue?@action=redirect&bpo=37834]: Prevent shutil.rmtree exception when built on non-Windows system without fd system call support, like older versions of macOS.

- bpo-10978 [https://bugs.python.org/issue?@action=redirect&bpo=10978]: Semaphores and BoundedSemaphores can now release more than one waiting thread at a time.

- bpo-37972 [https://bugs.python.org/issue?@action=redirect&bpo=37972]: Subscripts to the `unittest.mock.call` objects now receive the same chaining mechanism as any other custom attributes, so that the following usage no longer raises a `TypeError`:

  ```
  call().foo().__getitem__('bar')
  ```

  Patch by blhsing

- bpo-37965 [https://bugs.python.org/issue?@action=redirect&bpo=37965]: Fix C compiler warning caused by distutils.ccompiler.CCompiler.has_function.

- bpo-37964 [https://bugs.python.org/issue?@action=redirect&bpo=37964]: Add `F_GETPATH` command to **`fcntl`**.

- bpo-37960 [https://bugs.python.org/issue?@action=redirect&bpo=37960]: `repr()` of buffered and text streams now silences only expected exceptions when get the value of «name» and «mode» attributes.

- bpo-37961 [https://bugs.python.org/issue?@action=redirect&bpo=37961]: Add a `total_nframe` field to the traces collected by the tracemalloc module. This field indicates the original number of frames before it was truncated.

- bpo-37951 [https://bugs.python.org/issue?@action=redirect&bpo=37951]: Most features of the subprocess module now work again in subinterpreters. Only *preexec_fn* is restricted in subinterpreters.

- bpo-36205 [https://bugs.python.org/issue?@action=redirect&bpo=36205]: Fix the rusage implementation of time.process_time() to correctly report the sum of the system and user CPU time.

- bpo-37950 [https://bugs.python.org/issue?@action=redirect&bpo=37950]: Fix **`ast.dump()`** when call with incompletely initialized node.

- bpo-34679 [https://bugs.python.org/issue?@action=redirect&bpo=34679]: Restores instantiation of Windows IOCP event loops from the non-main thread.

- bpo-36917 [https://bugs.python.org/issue?@action=redirect&bpo=36917]: Add default implementation of the **`ast.NodeVisitor.visit_Constant()`** method which emits a deprecation warning and calls corresponding methods `visit_Num()`, `visit_Str()`, etc.

- bpo-37798 [https://bugs.python.org/issue?@action=redirect&bpo=37798]: Update test_statistics.py to verify that the statistics module works well for both C and Python implementations. Patch by Donghee Na

- bpo-26589 [https://bugs.python.org/issue?@action=redirect&bpo=26589]: Added a new status code to the http module: 451 UNAVAILABLE_FOR_LEGAL_REASONS

- bpo-37915 [https://bugs.python.org/issue?@action=redirect&bpo=37915]: Fix a segmentation fault that appeared when comparing instances of `datetime.timezone` and `datetime.tzinfo` objects. Patch by Pablo Galindo.

- bpo-32554 [https://bugs.python.org/issue?@action=redirect&bpo=32554]: Deprecate having random.seed() call hash on arbitrary types.

- bpo-9938 [https://bugs.python.org/issue?@action=redirect&bpo=9938]: Add optional keyword argument `exit_on_error` for **`ArgumentParser`**.

- bpo-37851 [https://bugs.python.org/issue?@action=redirect&bpo=37851]: The **`faulthandler`** module no longer allocates its alternative stack at Python startup. Now the stack is only allocated at the first faulthandler usage.

- bpo-32793 [https://bugs.python.org/issue?@action=redirect&bpo=32793]: Fix a duplicated debug message when **`smtplib.SMTP.connect()`** is called.

- bpo-37885 [https://bugs.python.org/issue?@action=redirect&bpo=37885]: venv: Don't generate unset variable warning on deactivate.

- bpo-37868 [https://bugs.python.org/issue?@action=redirect&bpo=37868]: Fix dataclasses.is_dataclass when given an instance that never raises AttributeError in _getattr_. That is, an object that returns something for _dataclass_fields_ even if it's not a dataclass.

- bpo-37811 [https://bugs.python.org/issue?@action=redirect&bpo=37811]: Fix `socket` module's `socket.connect(address)` function being unable to establish connection in case of interrupted system call. The problem was observed on all OSes which `poll(2)` system call can take only non-negative integers and -1 as a timeout value.

- bpo-37863 [https://bugs.python.org/issue?@action=redirect&bpo=37863]: Optimizations for Fraction._hash_ suggested by Tim Peters.

- bpo-21131 [https://bugs.python.org/issue?@action=redirect&bpo=21131]: Fix `faulthandler.register(chain=True)` stack. faulthandler now allocates a dedicated stack of `SIGSTKSZ*2` bytes, instead of just `SIGSTKSZ` bytes. Calling the previous signal handler in faulthandler signal handler uses more than

`SIGSTKSZ` bytes of stack memory on some platforms.

- bpo-37798 [https://bugs.python.org/issue?@action=redirect&bpo=37798]: Add C fastpath for statistics.NormalDist.inv_cdf() Patch by Donghee Na

- bpo-37804 [https://bugs.python.org/issue?@action=redirect&bpo=37804]: Remove the deprecated method `threading.Thread.isAlive()`. Patch by Donghee Na.

- bpo-37819 [https://bugs.python.org/issue?@action=redirect&bpo=37819]: Add Fraction.as_integer_ratio() to match the corresponding methods in bool, int, float, and decimal.

- bpo-14465 [https://bugs.python.org/issue?@action=redirect&bpo=14465]: Add an xml.etree.ElementTree.indent() function for pretty-printing XML trees. Contributed by Stefan Behnel.

- bpo-37810 [https://bugs.python.org/issue?@action=redirect&bpo=37810]: Fix **`difflib`** `?` hint in diff output when dealing with tabs. Patch by Anthony Sottile.

- bpo-37772 [https://bugs.python.org/issue?@action=redirect&bpo=37772]: In `zipfile.Path`, when adding implicit dirs, ensure that ancestral directories are added and that duplicates are excluded.

- bpo-18578 [https://bugs.python.org/issue?@action=redirect&bpo=18578]: Renamed and documented `test.bytecode_helper` as `test.support.bytecode_helper`. Patch by Joannah Nanjekye.

- bpo-37785 [https://bugs.python.org/issue?@action=redirect&bpo=37785]: Fix xgettext warnings in **`argparse`**.

- bpo-34488 [https://bugs.python.org/issue?@action=redirect&bpo=34488]: **`writelines()`** method of

`io.BytesIO` is now slightly faster when many small lines are passed. Patch by Sergey Fedoseev.

- bpo-37449 [https://bugs.python.org/issue?@action=redirect&bpo=37449]: `ensurepip` now uses `importlib.resources.read_binary()` to read data instead of `pkgutil.get_data()`. Patch by Joannah Nanjekye.
- bpo-28292 [https://bugs.python.org/issue?@action=redirect&bpo=28292]: Mark calendar.py helper functions as being private. The follows PEP 8 guidance to maintain the style conventions in the module and it addresses a known case of user confusion.
- bpo-18049 [https://bugs.python.org/issue?@action=redirect&bpo=18049]: Add definition of THREAD_STACK_SIZE for AIX in Python/thread_pthread.h The default thread stacksize caused crashes with the default recursion limit Patch by M Felt
- bpo-37742 [https://bugs.python.org/issue?@action=redirect&bpo=37742]: The logging.getLogger() API now returns the root logger when passed the name "root", whereas previously it returned a non-root logger named "root". This could affect cases where user code explicitly wants a non-root logger named "root", or instantiates a logger using logging.getLogger(_name_) in some top-level module called "root.py".
- bpo-37738 [https://bugs.python.org/issue?@action=redirect&bpo=37738]: Fix the implementation of curses `addch(str, color_pair)`: pass the color pair to `setcchar()`, instead of always passing 0 as the color pair.
- bpo-37723 [https://bugs.python.org/issue?@action=redirect&bpo=37723]: Fix performance regression on regular expression parsing with huge character sets. Patch by Yann Vaginay.
- bpo-35943 [https://bugs.python.org/issue?

@action=redirect&bpo=35943]: The function `PyImport_GetModule()` now ensures any module it returns is fully initialized. Patch by Joannah Nanjekye.

- bpo-32178 [https://bugs.python.org/issue?@action=redirect&bpo=32178]: Fix IndexError in `email` package when trying to parse invalid address fields starting with `:`.
- bpo-37268 [https://bugs.python.org/issue?@action=redirect&bpo=37268]: The `parser` module is deprecated and will be removed in future versions of Python.
- bpo-11953 [https://bugs.python.org/issue?@action=redirect&bpo=11953]: Completing WSA* error codes in `socket`.
- bpo-37685 [https://bugs.python.org/issue?@action=redirect&bpo=37685]: Fixed comparisons of `datetime.timedelta` and `datetime.timezone`.
- bpo-37697 [https://bugs.python.org/issue?@action=redirect&bpo=37697]: Synchronize `importlib.metadata` with importlib_metadata 0.19 [https://gitlab.com/python-devs/importlib_metadata/-/milestones/20], improving handling of EGG-INFO files and fixing a crash when entry point names contained colons.
- bpo-37695 [https://bugs.python.org/issue?@action=redirect&bpo=37695]: Correct `curses.unget_wch()` error message. Patch by Anthony Sottile.
- bpo-37689 [https://bugs.python.org/issue?@action=redirect&bpo=37689]: Add `is_relative_to()` in `PurePath` to determine whether or not one path is relative to another.
- bpo-29553 [https://bugs.python.org/issue?@action=redirect&bpo=29553]: Fixed `argparse.ArgumentParser.format_usage()` for mutually exclusive groups. Patch by Andrew Nester.

- bpo-37691 [https://bugs.python.org/issue?@action=redirect&bpo=37691]: Let math.dist() accept coordinates as sequences (or iterables) rather than just tuples.

- bpo-37685 [https://bugs.python.org/issue?@action=redirect&bpo=37685]: Fixed `__eq__`, `__lt__` etc implementations in some classes. They now return **`NotImplemented`** for unsupported type of the other operand. This allows the other operand to play role (for example the equality comparison with **`ANY`** will return `True`).

- bpo-37354 [https://bugs.python.org/issue?@action=redirect&bpo=37354]: Make Activate.ps1 Powershell script static to allow for signing it.

- bpo-37664 [https://bugs.python.org/issue?@action=redirect&bpo=37664]: Update wheels bundled with ensurepip (pip 19.2.3 and setuptools 41.2.0)

- bpo-37663 [https://bugs.python.org/issue?@action=redirect&bpo=37663]: Bring consistency to venv shell activation scripts by always using _VENV_PROMPT_.

- bpo-37642 [https://bugs.python.org/issue?@action=redirect&bpo=37642]: Allowed the pure Python implementation of **`datetime.timezone`** to represent sub-minute offsets close to minimum and maximum boundaries, specifically in the ranges (23:59, 24:00) and (-23:59, 24:00). Patch by Ngalim Siregar

- bpo-36161 [https://bugs.python.org/issue?@action=redirect&bpo=36161]: In **`posix`**, use `ttyname_r` instead of `ttyname` for thread safety.

- bpo-36324 [https://bugs.python.org/issue?@action=redirect&bpo=36324]: Make internal attributes for statistics.NormalDist() private.

- bpo-37555 [https://bugs.python.org/issue?@action=redirect&bpo=37555]: Fix

`NonCallableMock._call_matcher` returning tuple instead of `_Call` object when `self._spec_signature` exists. Patch by Elizabeth Uselton

- bpo-29446 [https://bugs.python.org/issue?@action=redirect&bpo=29446]: Make `from tkinter import *` import only the expected objects.

- bpo-16970 [https://bugs.python.org/issue?@action=redirect&bpo=16970]: Adding a value error when an invalid value in passed to nargs Patch by Robert Leenders

- bpo-34443 [https://bugs.python.org/issue?@action=redirect&bpo=34443]: Exceptions from **`enum`** now use the `__qualname` of the enum class in the exception message instead of the `__name__`.

- bpo-37491 [https://bugs.python.org/issue?@action=redirect&bpo=37491]: Fix `IndexError` when parsing email headers with unexpectedly ending bare-quoted string value. Patch by Abhilash Raj.

- bpo-37587 [https://bugs.python.org/issue?@action=redirect&bpo=37587]: Make json.loads faster for long strings. (Patch by Marco Paolini)

- bpo-18378 [https://bugs.python.org/issue?@action=redirect&bpo=18378]: Recognize «UTF-8» as a valid value for LC_CTYPE in locale._parse_localename.

- bpo-37579 [https://bugs.python.org/issue?@action=redirect&bpo=37579]: Return **`NotImplemented`** in Python implementation of `__eq__` for **`timedelta`** and **`time`** when the other object being compared is not of the same type to match C implementation. Patch by Karthikeyan Singaravelan.

- bpo-21478 [https://bugs.python.org/issue?@action=redirect&bpo=21478]: Record calls to parent when autospecced object is attached to a mock using **`unittest.mock.attach_mock()`**. Patch by Karthikeyan

Singaravelan.

- bpo-37531 [https://bugs.python.org/issue?@action=redirect&bpo=37531]: «python3 -m test -jN –timeout=TIMEOUT» now kills a worker process if it runs longer than *TIMEOUT* seconds.
- bpo-37482 [https://bugs.python.org/issue?@action=redirect&bpo=37482]: Fix serialization of display name in originator or destination address fields with both encoded words and special chars.
- bpo-36993 [https://bugs.python.org/issue?@action=redirect&bpo=36993]: Improve error reporting for corrupt zip files with bad zip64 extra data. Patch by Daniel Hillier.
- bpo-37502 [https://bugs.python.org/issue?@action=redirect&bpo=37502]: pickle.loads() no longer raises TypeError when the buffers argument is set to None
- bpo-37520 [https://bugs.python.org/issue?@action=redirect&bpo=37520]: Correct behavior for zipfile.Path.parent when the path object identifies a subdirectory.
- bpo-18374 [https://bugs.python.org/issue?@action=redirect&bpo=18374]: Fix the `.col_offset` attribute of nested **`ast.BinOp`** instances which had a too large value in some situations.
- bpo-37424 [https://bugs.python.org/issue?@action=redirect&bpo=37424]: Fixes a possible hang when using a timeout on `subprocess.run()` while capturing output. If the child process spawned its own children or otherwise connected its stdout or stderr handles with another process, we could hang after the timeout was reached and our child was killed when attempting to read final output from the pipes.
- bpo-37421 [https://bugs.python.org/issue?

@action=redirect&bpo=37421]: Fix **`multiprocessing.util.get_temp_dir()`** finalizer: clear also the "tempdir" configuration of the current process, so next call to `get_temp_dir()` will create a new temporary directory, rather than reusing the removed temporary directory.

- bpo-37481 [https://bugs.python.org/issue?@action=redirect&bpo=37481]: The distutils `bdist_wininst` command is deprecated in Python 3.8, use `bdist_wheel` (wheel packages) instead.

- bpo-37479 [https://bugs.python.org/issue?@action=redirect&bpo=37479]: When `Enum.__str__` is overridden in a derived class, the override will be used by `Enum.__format__` regardless of whether mixin classes are present.

- bpo-37440 [https://bugs.python.org/issue?@action=redirect&bpo=37440]: http.client now enables TLS 1.3 post-handshake authentication for default context or if a cert_file is passed to HTTPSConnection.

- bpo-37437 [https://bugs.python.org/issue?@action=redirect&bpo=37437]: Update vendorized expat version to 2.2.7.

- bpo-37428 [https://bugs.python.org/issue?@action=redirect&bpo=37428]: SSLContext.post_handshake_auth = True no longer sets SSL_VERIFY_POST_HANDSHAKE verify flag for client connections. Although the option is documented as ignored for clients, OpenSSL implicitly enables cert chain validation when the flag is set.

- bpo-37420 [https://bugs.python.org/issue?@action=redirect&bpo=37420]: **`os.sched_setaffinity()`** now correctly handles errors that arise during iteration over its `mask` argument. Patch by Brandt Bucher.

- bpo-37412 [https://bugs.python.org/issue?@action=redirect&bpo=37412]: The **`os.getcwdb()`** function now uses the UTF-8 encoding on Windows, rather than the ANSI code page: see **PEP 529** [https://peps.python.org/pep-0529/] for the rationale. The function is no longer deprecated on Windows.

- bpo-37406 [https://bugs.python.org/issue?@action=redirect&bpo=37406]: The sqlite3 module now raises TypeError, rather than ValueError, if operation argument type is not str: execute(), executemany() and calling a connection.
- bpo-29412 [https://bugs.python.org/issue?@action=redirect&bpo=29412]: Fix IndexError in parsing a header value ending unexpectedly. Patch by Abhilash Raj.
- bpo-36546 [https://bugs.python.org/issue?@action=redirect&bpo=36546]: The *dist* argument for statistics.quantiles() is now positional only. The current name doesn't reflect that the argument can be either a dataset or a distribution. Marking the parameter as positional avoids confusion and makes it possible to change the name later.
- bpo-37394 [https://bugs.python.org/issue?@action=redirect&bpo=37394]: Fix a bug that was causing the `queue` module to fail if the accelerator module was not available. Patch by Pablo Galindo.
- bpo-37376 [https://bugs.python.org/issue?@action=redirect&bpo=37376]: `pprint` now has support for `types.SimpleNamespace`. Patch by Carl Bordum Hansen.
- bpo-26967 [https://bugs.python.org/issue?@action=redirect&bpo=26967]: An `ArgumentParser` with `allow_abbrev=False` no longer disables grouping of short flags, such as `-vv`, but only disables abbreviation of long flags as documented. Patch by Zac Hatfield-Dodds.
- bpo-37212 [https://bugs.python.org/issue?@action=redirect&bpo=37212]: `unittest.mock.call()` now preserves the order of keyword arguments in repr output. Patch by Karthikeyan Singaravelan.

- bpo-37372 [https://bugs.python.org/issue?@action=redirect&bpo=37372]: Fix error unpickling datetime.time objects from Python 2 with seconds>=24. Patch by Justin Blanchard.

- bpo-37345 [https://bugs.python.org/issue?@action=redirect&bpo=37345]: Add formal support for UDPLITE sockets. Support was present before, but it is now easier to detect support with `hasattr(socket, 'IPPROTO_UDPLITE')` and there are constants defined for each of the values needed: `socket.IPPROTO_UDPLITE`, `UDPLITE_SEND_CSCOV`, and `UDPLITE_RECV_CSCOV`. Patch by Gabe Appleton.

- bpo-37358 [https://bugs.python.org/issue?@action=redirect&bpo=37358]: Optimized `functools.partial` by using vectorcall.

- bpo-37347 [https://bugs.python.org/issue?@action=redirect&bpo=37347]: **`sqlite3.Connection.create_aggregate()`**, **`sqlite3.Connection.create_function()`**, **`sqlite3.Connection.set_authorizer()`**, **`sqlite3.Connection.set_progress_handler()`** **`sqlite3.Connection.set_trace_callback()`** methods lead to segfaults if some of these methods are called twice with an equal object but not the same. Now callbacks are stored more carefully. Patch by Aleksandr Balezin.

- bpo-37163 [https://bugs.python.org/issue?@action=redirect&bpo=37163]: The *obj* argument of **`dataclasses.replace()`** is positional-only now.

- bpo-37085 [https://bugs.python.org/issue?@action=redirect&bpo=37085]: Add the optional Linux SocketCAN Broadcast Manager constants, used as flags to configure the BCM behaviour, in the socket module. Patch by Karl Ding.

- bpo-37328 [https://bugs.python.org/issue?@action=redirect&bpo=37328]: `HTMLParser.unescape` is removed. It was undocumented and deprecated since Python 3.4.

- bpo-37305 [https://bugs.python.org/issue?@action=redirect&bpo=37305]: Add .webmanifest -> application/manifest+json to list of recognized file types and content type headers

- bpo-37320 [https://bugs.python.org/issue?@action=redirect&bpo=37320]: `aifc.openfp()` alias to `aifc.open()`, `sunau.openfp()` alias to `sunau.open()`, and `wave.openfp()` alias to `wave.open()` have been removed. They were deprecated since Python 3.7.

- bpo-37315 [https://bugs.python.org/issue?@action=redirect&bpo=37315]: Deprecated accepting floats with integral value (like `5.0`) in **`math.factorial()`**.

- bpo-37312 [https://bugs.python.org/issue?@action=redirect&bpo=37312]: `_dummy_thread` and `dummy_threading` modules have been removed. These modules were deprecated since Python 3.7 which requires threading support.

- bpo-33972 [https://bugs.python.org/issue?@action=redirect&bpo=33972]: Email with single part but content-type set to `multipart/*` doesn't raise AttributeError anymore.

- bpo-37280 [https://bugs.python.org/issue?@action=redirect&bpo=37280]: Use threadpool for reading from file for sendfile fallback mode.

- bpo-37279 [https://bugs.python.org/issue?@action=redirect&bpo=37279]: Fix asyncio sendfile support when sendfile sends extra data in fallback mode.

- bpo-19865 [https://bugs.python.org/issue?@action=redirect&bpo=19865]: **`ctypes.create_unicode_buffer()`** now also supports non-BMP characters on platforms with 16-bit **`wchar_t`** (for example, Windows and AIX).

- bpo-37266 [https://bugs.python.org/issue?@action=redirect&bpo=37266]: In a subinterpreter, spawning a daemon thread now raises an exception. Daemon threads were never supported in subinterpreters. Previously, the subinterpreter finalization crashed with a Python fatal error if a daemon thread was still running.

- bpo-37210 [https://bugs.python.org/issue?@action=redirect&bpo=37210]: Allow pure Python implementation of `pickle` to work even when the C `_pickle` module is unavailable.

- bpo-21872 [https://bugs.python.org/issue?@action=redirect&bpo=21872]: Fix `lzma`: module decompresses data incompletely. When decompressing a FORMAT_ALONE format file, and it doesn't have the end marker, sometimes the last one to dozens bytes can't be output. Patch by Ma Lin.

- bpo-35922 [https://bugs.python.org/issue?@action=redirect&bpo=35922]: Fix `RobotFileParser.crawl_delay()` and `RobotFileParser.request_rate()` to return `None` rather than raise `AttributeError` when no relevant rule is defined in the robots.txt file. Patch by Rémi Lapeyre.

- bpo-35766 [https://bugs.python.org/issue?@action=redirect&bpo=35766]: Change the format of feature_version to be a (major, minor) tuple.

- bpo-36607 [https://bugs.python.org/issue?@action=redirect&bpo=36607]: Eliminate `RuntimeError` raised by `asyncio.all_tasks()` if internal tasks weak set is changed by another thread during iteration.

- bpo-18748 [https://bugs.python.org/issue?@action=redirect&bpo=18748]: `_pyio.IOBase` destructor now does nothing if getting the `closed` attribute fails to better mimic `_io.IOBase` finalizer.

- bpo-36402 [https://bugs.python.org/issue?@action=redirect&bpo=36402]: Fix a race condition at Python shutdown when waiting for threads. Wait until the Python thread state of all non-daemon threads get deleted (join all non-daemon threads), rather than just wait until non-daemon Python threads complete.

- bpo-37206 [https://bugs.python.org/issue?@action=redirect&bpo=37206]: Default values which cannot be represented as Python objects no longer improperly represented as `None` in function signatures.

- bpo-37111 [https://bugs.python.org/issue?@action=redirect&bpo=37111]: Added `encoding` and `errors` keyword parameters to `logging.basicConfig`.

- bpo-12144 [https://bugs.python.org/issue?@action=redirect&bpo=12144]: Ensure cookies with `expires` attribute are handled in **`CookieJar.make_cookies()`**.

- bpo-34886 [https://bugs.python.org/issue?@action=redirect&bpo=34886]: Fix an unintended ValueError from **`subprocess.run()`** when checking for conflicting *input* and *stdin* or *capture_output* and *stdout* or *stderr* args when they were explicitly provided but with `None` values within a passed in `**kwargs` dict rather than as passed directly by name. Patch contributed by Rémi Lapeyre.

- bpo-37173 [https://bugs.python.org/issue?@action=redirect&bpo=37173]: The exception message for `inspect.getfile()` now correctly reports the passed class rather than the builtins module.

- bpo-37178 [https://bugs.python.org/issue?@action=redirect&bpo=37178]: Give math.perm() a one argument form that means the same as math.factorial().

- bpo-37178 [https://bugs.python.org/issue?@action=redirect&bpo=37178]: For math.perm(n, k), let k default to n, giving the same result as factorial.

- bpo-37165 [https://bugs.python.org/issue?@action=redirect&bpo=37165]: Converted _collections._count_elements to use the Argument Clinic.

- bpo-34767 [https://bugs.python.org/issue?@action=redirect&bpo=34767]: Do not always create a `collections.deque` in `asyncio.Lock`.

- bpo-37158 [https://bugs.python.org/issue?@action=redirect&bpo=37158]: Speed-up statistics.fmean() by switching from a function to a generator.

- bpo-34282 [https://bugs.python.org/issue?@action=redirect&bpo=34282]: Remove `Enum._convert` method, deprecated in 3.8.

- bpo-37150 [https://bugs.python.org/issue?@action=redirect&bpo=37150]: `argparse._ActionsContainer.add_argument` now throws error, if someone accidentally pass FileType class object instead of instance of FileType as `type` argument.

- bpo-28724 [https://bugs.python.org/issue?@action=redirect&bpo=28724]: The socket module now has the `socket.send_fds()` and `socket.recv.fds()` methods. Contributed by Joannah Nanjekye, Shinya Okano and Victor Stinner.

- bpo-35621 [https://bugs.python.org/issue?@action=redirect&bpo=35621]: Support running asyncio subprocesses when execution event loop in a thread on UNIX.

- bpo-36520 [https://bugs.python.org/issue?@action=redirect&bpo=36520]: Lengthy email headers with UTF-8 characters are now properly encoded when they are folded. Patch by Jeffrey Kintscher.

- bpo-30835 [https://bugs.python.org/issue?@action=redirect&bpo=30835]: Fixed a bug in email parsing where a message with invalid bytes in content-transfer-encoding of a multipart message can cause an AttributeError.

Patch by Andrew Donnellan.

- bpo-31163 [https://bugs.python.org/issue?@action=redirect&bpo=31163]: pathlib.Path instance's rename and replace methods now return the new Path instance.
- bpo-25068 [https://bugs.python.org/issue?@action=redirect&bpo=25068]: `urllib.request.ProxyHandler` now lowercases the keys of the passed dictionary.
- bpo-26185 [https://bugs.python.org/issue?@action=redirect&bpo=26185]: Fix `repr()` on empty `ZipInfo` object. Patch by Mickaël Schoentgen.
- bpo-21315 [https://bugs.python.org/issue?@action=redirect&bpo=21315]: Email headers containing RFC2047 encoded words are parsed despite the missing whitespace, and a defect registered. Also missing trailing whitespace after encoded words is now registered as a defect.
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Port test_datetime to VxWorks: skip zoneinfo tests on VxWorks
- bpo-35805 [https://bugs.python.org/issue?@action=redirect&bpo=35805]: Add parser for Message-ID header and add it to default HeaderRegistry. This should prevent folding of Message-ID using RFC 2048 encoded words.
- bpo-36871 [https://bugs.python.org/issue?@action=redirect&bpo=36871]: Ensure method signature is used instead of constructor signature of a class while asserting mock object against method calls. Patch by Karthikeyan Singaravelan.
- bpo-35070 [https://bugs.python.org/issue?@action=redirect&bpo=35070]: posix.getgrouplist() now works correctly when the user belongs to NGROUPS_MAX supplemental groups. Patch by Jeffrey Kintscher.

- bpo-31783 [https://bugs.python.org/issue?@action=redirect&bpo=31783]: Fix race condition in ThreadPoolExecutor when worker threads are created during interpreter shutdown.

- bpo-36582 [https://bugs.python.org/issue?@action=redirect&bpo=36582]: Fix `UserString.encode()` to correctly return `bytes` rather than a `UserString` instance.

- bpo-32424 [https://bugs.python.org/issue?@action=redirect&bpo=32424]: Deprecate xml.etree.ElementTree.Element.copy() in favor of copy.copy().

  Patch by Gordon P. Hemsley

- bpo-36564 [https://bugs.python.org/issue?@action=redirect&bpo=36564]: Fix infinite loop in email header folding logic that would be triggered when an email policy's max_line_length is not long enough to include the required markup and any values in the message. Patch by Paul Ganssle

- bpo-36543 [https://bugs.python.org/issue?@action=redirect&bpo=36543]: Removed methods Element.getchildren(), Element.getiterator() and ElementTree.getiterator() and the xml.etree.cElementTree module.

- bpo-36409 [https://bugs.python.org/issue?@action=redirect&bpo=36409]: Remove the old plistlib API deprecated in Python 3.4

- bpo-36302 [https://bugs.python.org/issue?@action=redirect&bpo=36302]: distutils sorts source file lists so that Extension .so files build more reproducibly by default

- bpo-36250 [https://bugs.python.org/issue?@action=redirect&bpo=36250]: Ignore `ValueError` from `signal` with `interaction` in non-main thread.

- bpo-36046 [https://bugs.python.org/issue?

@action=redirect&bpo=36046]: Added `user`, `group` and `extra_groups` parameters to the subprocess.Popen constructor. Patch by Patrick McLean.

- bpo-32627 [https://bugs.python.org/issue?@action=redirect&bpo=32627]: Fix compile error when `_uuid` headers conflicting included.
- bpo-35800 [https://bugs.python.org/issue?@action=redirect&bpo=35800]: Deprecate `smtpd.MailmanProxy` ready for future removal.
- bpo-35168 [https://bugs.python.org/issue?@action=redirect&bpo=35168]: **`shlex.shlex.punctuation_chars`** is now a read-only property.
- bpo-8538 [https://bugs.python.org/issue?@action=redirect&bpo=8538]: Add support for boolean actions like `--foo` and `--no-foo` to argparse. Patch contributed by Rémi Lapeyre.
- bpo-20504 [https://bugs.python.org/issue?@action=redirect&bpo=20504]: Fixes a bug in **`cgi`** module when a multipart/form-data request has no `Content-Length` header.
- bpo-25988 [https://bugs.python.org/issue?@action=redirect&bpo=25988]: The abstract base classes in **`collections.abc`** no longer are exposed in the regular **`collections`** module.
- bpo-11122 [https://bugs.python.org/issue?@action=redirect&bpo=11122]: Distutils won't check for rpmbuild in specified paths only.
- bpo-34775 [https://bugs.python.org/issue?@action=redirect&bpo=34775]: Division handling of PurePath now returns NotImplemented instead of raising a TypeError when passed something other than an instance of str or PurePath. Patch by Roger Aiudi.

- bpo-34749 [https://bugs.python.org/issue?@action=redirect&bpo=34749]: **`binascii.a2b_base64()`** is now up to 2 times faster. Patch by Sergey Fedoseev.

- bpo-34519 [https://bugs.python.org/issue?@action=redirect&bpo=34519]: Add additional aliases for HP Roman 8. Patch by Michael Osipov.

- bpo-28009 [https://bugs.python.org/issue?@action=redirect&bpo=28009]: Fix uuid.getnode() on platforms with "." as MAC Addr delimiter as well fix for MAC Addr format that omits a leading 0 in MAC Addr values. Currently, AIX is the only know platform with these settings. Patch by Michael Felt.

- bpo-30618 [https://bugs.python.org/issue?@action=redirect&bpo=30618]: Add **`readlink()`**. Patch by Girts Folkmanis.

- bpo-32498 [https://bugs.python.org/issue?@action=redirect&bpo=32498]: Made **`urllib.parse.unquote()`** accept bytes in addition to strings. Patch by Stein Karlsen.

- bpo-33348 [https://bugs.python.org/issue?@action=redirect&bpo=33348]: lib2to3 now recognizes expressions after `*` and `**` like in `f(*[] or [])`.

- bpo-32689 [https://bugs.python.org/issue?@action=redirect&bpo=32689]: Update **`shutil.move()`** function to allow for Path objects to be used as source argument. Patch by Emily Morehouse and Maxwell «5.13b» McKinnon.

- bpo-32820 [https://bugs.python.org/issue?@action=redirect&bpo=32820]: Added `__format__` to IPv4 and IPv6 classes. Always outputs a fully zero-padded string. Supports b/x/n modifiers (bin/hex/native format). Native format for IPv4 is bin, native format for IPv6 is hex. Also supports "#" and "_" modifiers.

- bpo-27657 [https://bugs.python.org/issue?@action=redirect&bpo=27657]: Fix urllib.parse.urlparse() with numeric paths. A string like «path:80» is no longer parsed as a path but as a scheme («path») and a path («80»).

- bpo-4963 [https://bugs.python.org/issue?@action=redirect&bpo=4963]: Fixed non-deterministic behavior related to mimetypes extension mapping and module reinitialization.

Documentation

- bpo-21767 [https://bugs.python.org/issue?@action=redirect&bpo=21767]: Explicitly mention abc support in functools.singledispatch

- bpo-38816 [https://bugs.python.org/issue?@action=redirect&bpo=38816]: Provides more details about the interaction between `fork()` and CPython's runtime, focusing just on the C-API. This includes cautions about where `fork()` should and shouldn't be called.

- bpo-38351 [https://bugs.python.org/issue?@action=redirect&bpo=38351]: Modernize `email` examples from %-formatting to f-strings.

- bpo-38778 [https://bugs.python.org/issue?@action=redirect&bpo=38778]: Document the fact that `RuntimeError` is raised if `os.fork()` is called in a subinterpreter.

- bpo-38592 [https://bugs.python.org/issue?@action=redirect&bpo=38592]: Add Brazilian Portuguese to the language switcher at Python Documentation website.

- bpo-38294 [https://bugs.python.org/issue?@action=redirect&bpo=38294]: Add list of no-longer-escaped chars to re.escape documentation

- bpo-38053 [https://bugs.python.org/issue?@action=redirect&bpo=38053]: Modernized the plistlib documentation

- bpo-26868 [https://bugs.python.org/issue?@action=redirect&bpo=26868]: Fix example usage of **`PyModule_AddObject()`** to properly handle errors.

- bpo-36797 [https://bugs.python.org/issue?@action=redirect&bpo=36797]: Fix a dead link in the distutils API Reference.

- bpo-37977 [https://bugs.python.org/issue?@action=redirect&bpo=37977]: Warn more strongly and clearly about pickle insecurity

- bpo-37979 [https://bugs.python.org/issue?@action=redirect&bpo=37979]: Added a link to dateutil.parser.isoparse in the datetime.fromisoformat documentation. Patch by Paul Ganssle

- bpo-12707 [https://bugs.python.org/issue?@action=redirect&bpo=12707]: Deprecate info(), geturl(), getcode() methods in favor of the headers, url, and status properties, respectively, for HTTPResponse and addinfourl. Also deprecate the code attribute of addinfourl in favor of the status attribute. Patch by Ashwin Ramaswami

- bpo-37937 [https://bugs.python.org/issue?@action=redirect&bpo=37937]: Mention `frame.f_trace` in **`sys.settrace()`** docs.

- bpo-37878 [https://bugs.python.org/issue?@action=redirect&bpo=37878]: Make **`PyThreadState_DeleteCurrent()`** Internal.

- bpo-37759 [https://bugs.python.org/issue?@action=redirect&bpo=37759]: Beginning edits to Whatsnew 3.8

- bpo-37726 [https://bugs.python.org/issue?@action=redirect&bpo=37726]: Stop recommending getopt in the tutorial for command line argument parsing and promote argparse.

- bpo-32910 [https://bugs.python.org/issue?

@action=redirect&bpo=32910]: Remove implementation-specific behaviour of how venv's Deactivate works.

- bpo-37256 [https://bugs.python.org/issue?@action=redirect&bpo=37256]: Fix wording of arguments for **`Request`** in **`urllib.request`**
- bpo-37284 [https://bugs.python.org/issue?@action=redirect&bpo=37284]: Add a brief note to indicate that any new `sys.implementation` required attributes must go through the PEP process.
- bpo-30088 [https://bugs.python.org/issue?@action=redirect&bpo=30088]: Documented that **`mailbox.Maildir`** constructor doesn't attempt to verify the maildir folder layout correctness. Patch by Sviatoslav Sydorenko.
- bpo-37521 [https://bugs.python.org/issue?@action=redirect&bpo=37521]: Fix `importlib` examples to insert any newly created modules via importlib.util.module_from_spec() immediately into sys.modules instead of after calling loader.exec_module().

  Thanks to Benjamin Mintz for finding the bug.
- bpo-37456 [https://bugs.python.org/issue?@action=redirect&bpo=37456]: Slash ("/") is now part of syntax.
- bpo-37487 [https://bugs.python.org/issue?@action=redirect&bpo=37487]: Fix PyList_GetItem index description to include 0.
- bpo-37149 [https://bugs.python.org/issue?@action=redirect&bpo=37149]: Replace the dead link to the Tkinter 8.5 reference by John Shipman, New Mexico Tech, with a link to the archive.org copy.
- bpo-37478 [https://bugs.python.org/issue?@action=redirect&bpo=37478]: Added possible exceptions to the description of os.chdir().

- bpo-34903 [https://bugs.python.org/issue?@action=redirect&bpo=34903]: Documented that in `datetime.datetime.strptime()`, the leading zero in some two-digit formats is optional. Patch by Mike Gleen.

- bpo-36260 [https://bugs.python.org/issue?@action=redirect&bpo=36260]: Add decompression pitfalls to zipfile module documentation.

- bpo-37004 [https://bugs.python.org/issue?@action=redirect&bpo=37004]: In the documentation for difflib, a note was added explicitly warning that the results of SequenceMatcher's ratio method may depend on the order of the input strings.

- bpo-36960 [https://bugs.python.org/issue?@action=redirect&bpo=36960]: Restructured the `datetime` docs in the interest of making them more user-friendly and improving readability. Patch by Brad Solomon.

- bpo-36487 [https://bugs.python.org/issue?@action=redirect&bpo=36487]: Make C-API docs clear about what the «main» interpreter is.

- bpo-23460 [https://bugs.python.org/issue?@action=redirect&bpo=23460]: The documentation for decimal string formatting using the `:g` specifier has been updated to reflect the correct exponential notation cutoff point. Original patch contributed by Tuomas Suutari.

- bpo-35803 [https://bugs.python.org/issue?@action=redirect&bpo=35803]: Document and test that `tempfile` functions may accept a path-like object for the `dir` argument. Patch by Anthony Sottile.

- bpo-33944 [https://bugs.python.org/issue?@action=redirect&bpo=33944]: Added a note about the intended use of code in .pth files.

- bpo-34293 [https://bugs.python.org/issue?@action=redirect&bpo=34293]: Fix the Doc/Makefile regarding

PAPER environment variable and PDF builds

- bpo-25237 [https://bugs.python.org/issue?@action=redirect&bpo=25237]: Add documentation for tkinter modules

Tests

- bpo-38614 [https://bugs.python.org/issue?@action=redirect&bpo=38614]: Fix test_communicate() of test_asyncio.test_subprocess: use `support.LONG_TIMEOUT` (5 minutes), instead of just 1 minute.

- bpo-38614 [https://bugs.python.org/issue?@action=redirect&bpo=38614]: Add timeout constants to **`test.support`**: **`LOOPBACK_TIMEOUT`**, **`INTERNET_TIMEOUT`**, **`SHORT_TIMEOUT`** and **`LONG_TIMEOUT`**.

- bpo-38502 [https://bugs.python.org/issue?@action=redirect&bpo=38502]: test.regrtest now uses process groups in the multiprocessing mode (-jN command line option) if process groups are available: if **`os.setsid()`** and **`os.killpg()`** functions are available.

- bpo-35998 [https://bugs.python.org/issue?@action=redirect&bpo=35998]: Fix a race condition in test_asyncio.test_start_tls_server_1(). Previously, there was a race condition between the test main() function which replaces the protocol and the test ServerProto protocol which sends ANSWER once it gets HELLO. Now, only the test main() function is responsible to send data, ServerProto no longer sends data.

- bpo-38470 [https://bugs.python.org/issue?@action=redirect&bpo=38470]: Fix `test_compileall.test_compile_dir_maxlevels()` on Windows without long path support: only create 3 subdirectories instead of between 20 and 100 subdirectories.

- bpo-37531 [https://bugs.python.org/issue?

@action=redirect&bpo=37531]: On timeout, regrtest no longer attempts to call `popen.communicate()` again: it can hang until all child processes using stdout and stderr pipes completes. Kill the worker process and ignores its output. Change also the faulthandler timeout of the main process from 1 minute to 5 minutes, for Python slowest buildbots.

- bpo-38239 [https://bugs.python.org/issue?@action=redirect&bpo=38239]: Fix test_gdb for Link Time Optimization (LTO) builds.
- bpo-38275 [https://bugs.python.org/issue?@action=redirect&bpo=38275]: test_ssl now handles disabled TLS/SSL versions better. OpenSSL's crypto policy and run-time settings are recognized and tests for disabled versions are skipped. Tests also accept more TLS minimum_versions for platforms that override OpenSSL's default with strict settings.
- bpo-38271 [https://bugs.python.org/issue?@action=redirect&bpo=38271]: The private keys for test_ssl were encrypted with 3DES in traditional PKCS#5 format. 3DES and the digest algorithm of PKCS#5 are blocked by some strict crypto policies. Use PKCS#8 format with AES256 encryption instead.
- bpo-38270 [https://bugs.python.org/issue?@action=redirect&bpo=38270]: test.support now has a helper function to check for availability of a hash digest function. Several tests are refactored avoid MD5 and use SHA256 instead. Other tests are marked to use MD5 and skipped when MD5 is disabled.
- bpo-37123 [https://bugs.python.org/issue?@action=redirect&bpo=37123]: Multiprocessing test test_mymanager() now also expects -SIGTERM, not only exitcode 0. BaseManager._finalize_manager() sends SIGTERM to the manager process if it takes longer than 1 second to stop, which happens on slow buildbots.
- bpo-38212 [https://bugs.python.org/issue?@action=redirect&bpo=38212]: Multiprocessing tests: increase

test_queue_feeder_donot_stop_onexc() timeout from 1 to 60 seconds.

- bpo-38117 [https://bugs.python.org/issue?@action=redirect&bpo=38117]: Test with OpenSSL 1.1.1d
- bpo-38018 [https://bugs.python.org/issue?@action=redirect&bpo=38018]: Increase code coverage for multiprocessing.shared_memory.
- bpo-37805 [https://bugs.python.org/issue?@action=redirect&bpo=37805]: Add tests for json.dump(…, skipkeys=True). Patch by Donghee Na.
- bpo-37531 [https://bugs.python.org/issue?@action=redirect&bpo=37531]: Enhance regrtest multiprocess timeout: write a message when killing a worker process, catch popen.kill() and popen.wait() exceptions, put a timeout on the second call to popen.communicate().
- bpo-37876 [https://bugs.python.org/issue?@action=redirect&bpo=37876]: Add tests for ROT-13 codec.
- bpo-36833 [https://bugs.python.org/issue?@action=redirect&bpo=36833]: Added tests for `PyDateTime_xxx_GET_xxx()` macros of the C API of the **`datetime`** module. Patch by Joannah Nanjekye.
- bpo-37558 [https://bugs.python.org/issue?@action=redirect&bpo=37558]: Fix test_shared_memory_cleaned_after_process_termination name handling
- bpo-37526 [https://bugs.python.org/issue?@action=redirect&bpo=37526]: Add **`test.support.catch_threading_exception()`**: context manager catching **`threading.Thread`** exception using **`threading.excepthook()`**.
- bpo-37421 [https://bugs.python.org/issue?@action=redirect&bpo=37421]: test_concurrent_futures now

explicitly stops the ForkServer instance if it's running.

- bpo-37421 [https://bugs.python.org/issue?@action=redirect&bpo=37421]: multiprocessing tests now stop the ForkServer instance if it's running: close the «alive» file descriptor to ask the server to stop and then remove its UNIX address.
- bpo-37421 [https://bugs.python.org/issue?@action=redirect&bpo=37421]: test_distutils.test_build_ext() is now able to remove the temporary directory on Windows: don't import the newly built C extension («xx») in the current process, but test it in a separated process.
- bpo-37421 [https://bugs.python.org/issue?@action=redirect&bpo=37421]: test_concurrent_futures now cleans up multiprocessing to remove immediately temporary directories created by multiprocessing.util.get_temp_dir().
- bpo-37421 [https://bugs.python.org/issue?@action=redirect&bpo=37421]: test_winconsoleio doesn't leak a temporary file anymore: use tempfile.TemporaryFile() to remove it when the test completes.
- bpo-37421 [https://bugs.python.org/issue?@action=redirect&bpo=37421]: multiprocessing tests now explicitly call `_run_finalizers()` to immediately remove temporary directories created by tests.
- bpo-37421 [https://bugs.python.org/issue?@action=redirect&bpo=37421]: urllib.request tests now call **`urlcleanup()`** to remove temporary files created by `urlretrieve()` tests and to clear the `_opener` global variable set by `urlopen()` and functions calling indirectly `urlopen()`.
- bpo-37472 [https://bugs.python.org/issue?@action=redirect&bpo=37472]: Remove `Lib/test/outstanding_bugs.py`.
- bpo-37199 [https://bugs.python.org/issue?

@action=redirect&bpo=37199]: Fix test failures when IPv6 is unavailable or disabled.

- bpo-19696 [https://bugs.python.org/issue?@action=redirect&bpo=19696]: Replace deprecated method «random.choose» with «random.choice» in «test_pkg_import.py».
- bpo-37335 [https://bugs.python.org/issue?@action=redirect&bpo=37335]: Remove no longer necessary code from c locale coercion tests
- bpo-37421 [https://bugs.python.org/issue?@action=redirect&bpo=37421]: Fix test_shutil to no longer leak temporary files.
- bpo-37411 [https://bugs.python.org/issue?@action=redirect&bpo=37411]: Fix test_wsgiref.testEnviron() to no longer depend on the environment variables (don't fail if «X» variable is set).
- bpo-37400 [https://bugs.python.org/issue?@action=redirect&bpo=37400]: Fix test_os.test_chown(): use os.getgroups() rather than grp.getgrall() to get groups. Rename also the test to test_chown_gid().
- bpo-37359 [https://bugs.python.org/issue?@action=redirect&bpo=37359]: Add –cleanup option to python3 -m test to remove `test_python_*` directories of previous failed jobs. Add «make cleantest» to run `python3 -m test --cleanup`.
- bpo-37362 [https://bugs.python.org/issue?@action=redirect&bpo=37362]: test_gdb no longer fails if it gets an «unexpected» message on stderr: it now ignores stderr. The purpose of test_gdb is to test that python-gdb.py commands work as expected, not to test gdb.
- bpo-35998 [https://bugs.python.org/issue?@action=redirect&bpo=35998]: Avoid TimeoutError in test_asyncio: test_start_tls_server_1()

- bpo-37278 [https://bugs.python.org/issue?@action=redirect&bpo=37278]: Fix test_asyncio ProactorLoopCtrlC: join the thread to prevent leaking a running thread and leaking a reference.
- bpo-37261 [https://bugs.python.org/issue?@action=redirect&bpo=37261]: Fix **`test.support.catch_unraisable_exception()`**: its _exit_() method now ignores unraisable exception raised when clearing its `unraisable` attribute.
- bpo-37069 [https://bugs.python.org/issue?@action=redirect&bpo=37069]: regrtest now uses **`sys.unraisablehook()`** to mark a test as «environment altered» (ENV_CHANGED) if it emits an «unraisable exception». Moreover, regrtest logs a warning in this case.

  Use `python3 -m test --fail-env-changed` to catch unraisable exceptions in tests.
- bpo-37252 [https://bugs.python.org/issue?@action=redirect&bpo=37252]: Fix assertions in `test_close` and `test_events_mask_overflow` devpoll tests.
- bpo-37169 [https://bugs.python.org/issue?@action=redirect&bpo=37169]: Rewrite `_PyObject_IsFreed()` unit tests.
- bpo-37153 [https://bugs.python.org/issue?@action=redirect&bpo=37153]: `test_venv.test_multiprocessing()` now explicitly calls `pool.terminate()` to wait until the pool completes.
- bpo-34001 [https://bugs.python.org/issue?@action=redirect&bpo=34001]: Make test_ssl pass with LibreSSL. LibreSSL handles minimum and maximum TLS version differently than OpenSSL.
- bpo-36919 [https://bugs.python.org/issue?@action=redirect&bpo=36919]: Make `test_source_encoding.test_issue2301`

implementation independent. The test will work now for both CPython and IronPython.

- bpo-30202 [https://bugs.python.org/issue?@action=redirect&bpo=30202]: Update `test.test_importlib.test_abc` to test `find_spec()`.

- bpo-28009 [https://bugs.python.org/issue?@action=redirect&bpo=28009]: Modify the test_uuid logic to test when a program is available AND can be used to obtain a MACADDR as basis for an UUID. Patch by M. Felt

- bpo-34596 [https://bugs.python.org/issue?@action=redirect&bpo=34596]: Fallback to a default reason when **`unittest.skip()`** is uncalled. Patch by Naitree Zhu.

Build

- bpo-38809 [https://bugs.python.org/issue?@action=redirect&bpo=38809]: On Windows, build scripts will now recognize and use python.exe from an active virtual env.

- bpo-38684 [https://bugs.python.org/issue?@action=redirect&bpo=38684]: Fix _hashlib build when Blake2 is disabled, but OpenSSL supports it.

- bpo-38468 [https://bugs.python.org/issue?@action=redirect&bpo=38468]: Misc/python-config.in now uses `getvar()` for all still existing `sysconfig.get_config_var()` calls. Patch by Joannah Nanjekye.

- bpo-37415 [https://bugs.python.org/issue?@action=redirect&bpo=37415]: Fix stdatomic.h header check for ICC compiler: the ICC implementation lacks atomic_uintptr_t type which is needed by Python.

- bpo-38301 [https://bugs.python.org/issue?@action=redirect&bpo=38301]: In Solaris family, we must be sure to use `-D_REENTRANT`. Patch by Jesús Cea Avión.

- bpo-36002 [https://bugs.python.org/issue?@action=redirect&bpo=36002]: Locate `llvm-profdata` and `llvm-ar` binaries using `AC_PATH_TOOL` rather than `AC_PATH_TARGET_TOOL`.

- bpo-37936 [https://bugs.python.org/issue?@action=redirect&bpo=37936]: The `.gitignore` file systematically keeps «rooted», with a non-trailing slash, all the rules that are meant to apply to files in a specific place in the repo. Previously, when the intended file to ignore happened to be at the root of the repo, we'd most often accidentally also ignore files and directories with the same name anywhere in the tree.

- bpo-37760 [https://bugs.python.org/issue?@action=redirect&bpo=37760]: The `Tools/unicode/makeunicodedata.py` script, which is used for converting information from the Unicode Character Database into generated code and data used by the methods of **`str`** and by the **`unicodedata`** module, now handles each character's data as a `dataclass` with named attributes, rather than a length-18 list of different fields.

- bpo-37936 [https://bugs.python.org/issue?@action=redirect&bpo=37936]: The `.gitignore` file no longer applies to any files that are in fact tracked in the Git repository. Patch by Greg Price.

- bpo-37725 [https://bugs.python.org/issue?@action=redirect&bpo=37725]: Change «clean» makefile target to also clean the program guided optimization (PGO) data. Previously you would have to use «make clean» and «make profile-removal», or «make clobber».

- bpo-37707 [https://bugs.python.org/issue?@action=redirect&bpo=37707]: Mark some individual tests to skip when –pgo is used. The tests marked increase the PGO task time significantly and likely don't help improve optimization of the final executable.

- bpo-36044 [https://bugs.python.org/issue?

@action=redirect&bpo=36044]: Reduce the number of unit tests run for the PGO generation task. This speeds up the task by a factor of about 15x. Running the full unit test suite is slow. This change may result in a slightly less optimized build since not as many code branches will be executed. If you are willing to wait for the much slower build, the old behavior can be restored using “./configure [..] PROFILE_TASK=»-m test –pgo-extended»”. We make no guarantees as to which PGO task set produces a faster build. Users who care should run their own relevant benchmarks as results can depend on the environment, workload, and compiler tool chain.

- bpo-37468 [https://bugs.python.org/issue?@action=redirect&bpo=37468]: `make install` no longer installs `wininst-*.exe` files used by distutils bdist_wininst: bdist_wininst only works on Windows.

- bpo-37189 [https://bugs.python.org/issue?@action=redirect&bpo=37189]: Many `PyRun_XXX()` functions like **`PyRun_String()`** were no longer exported in `libpython38.dll` by mistake. Export them again to fix the ABI compatibility.

- bpo-25361 [https://bugs.python.org/issue?@action=redirect&bpo=25361]: Enables use of SSE2 instructions in Windows 32-bit build.

- bpo-36210 [https://bugs.python.org/issue?@action=redirect&bpo=36210]: Update optional extension module detection for AIX. ossaudiodev and spwd are not applicable for AIX, and are no longer reported as missing. 3rd-party packaging of ncurses (with ASIS support) conflicts with officially supported AIX curses library, so configure AIX to use libcurses.a. However, skip trying to build _curses_panel.

  patch by M Felt

Windows

- bpo-38589 [https://bugs.python.org/issue?@action=redirect&bpo=38589]: Fixes HTML Help shortcut when

Windows is not installed to C drive
- bpo-38453 [https://bugs.python.org/issue?@action=redirect&bpo=38453]: Ensure ntpath.realpath() correctly resolves relative paths.
- bpo-38519 [https://bugs.python.org/issue?@action=redirect&bpo=38519]: Restores the internal C headers that were missing from the nuget.org and Microsoft Store packages.
- bpo-38492 [https://bugs.python.org/issue?@action=redirect&bpo=38492]: Remove `pythonw.exe` dependency on the Microsoft C++ runtime.
- bpo-38344 [https://bugs.python.org/issue?@action=redirect&bpo=38344]: Fix error message in activate.bat
- bpo-38359 [https://bugs.python.org/issue?@action=redirect&bpo=38359]: Ensures `pyw.exe` launcher reads correct registry key.
- bpo-38355 [https://bugs.python.org/issue?@action=redirect&bpo=38355]: Fixes `ntpath.realpath` failing on `sys.executable`.
- bpo-38117 [https://bugs.python.org/issue?@action=redirect&bpo=38117]: Update bundled OpenSSL to 1.1.1d
- bpo-38092 [https://bugs.python.org/issue?@action=redirect&bpo=38092]: Reduce overhead when using multiprocessing in a Windows virtual environment.
- bpo-38133 [https://bugs.python.org/issue?@action=redirect&bpo=38133]: Allow py.exe launcher to locate installations from the Microsoft Store and improve display of active virtual environments.
- bpo-38114 [https://bugs.python.org/issue?@action=redirect&bpo=38114]: The `pip.ini` is no longer included in the Nuget package.
- bpo-32592 [https://bugs.python.org/issue?@action=redirect&bpo=32592]: Set Windows 8 as the minimum required version for API support
- bpo-36634 [https://bugs.python.org/issue?@action=redirect&bpo=36634]: **`os.cpu_count()`** now returns active processors rather than maximum processors.
- bpo-36634 [https://bugs.python.org/issue?@action=redirect&bpo=36634]: venv activate.bat now works

when the existing variables contain double quote characters.

- bpo-38081 [https://bugs.python.org/issue?@action=redirect&bpo=38081]: Prevent error calling `os.path.realpath()` on `'NUL'`.
- bpo-38087 [https://bugs.python.org/issue?@action=redirect&bpo=38087]: Fix case sensitivity in test_pathlib and test_ntpath.
- bpo-38088 [https://bugs.python.org/issue?@action=redirect&bpo=38088]: Fixes distutils not finding vcruntime140.dll with only the v142 toolset installed.
- bpo-37283 [https://bugs.python.org/issue?@action=redirect&bpo=37283]: Ensure command-line and unattend.xml setting override previously detected states in Windows installer.
- bpo-38030 [https://bugs.python.org/issue?@action=redirect&bpo=38030]: Fixes `os.stat()` failing for block devices on Windows
- bpo-38020 [https://bugs.python.org/issue?@action=redirect&bpo=38020]: Fixes potential crash when calling `os.readlink()` (or indirectly through `realpath()`) on a file that is not a supported link.
- bpo-37705 [https://bugs.python.org/issue?@action=redirect&bpo=37705]: Improve the implementation of `winerror_to_errno()`.
- bpo-37549 [https://bugs.python.org/issue?@action=redirect&bpo=37549]: `os.dup()` no longer fails for standard streams on Windows 7.
- bpo-1311 [https://bugs.python.org/issue?@action=redirect&bpo=1311]: The `nul` file on Windows now returns True from `exists()` and a valid result from `os.stat()` with `S_IFCHR` set.
- bpo-9949 [https://bugs.python.org/issue?@action=redirect&bpo=9949]: Enable support for following symlinks in `os.realpath()`.
- bpo-37834 [https://bugs.python.org/issue?@action=redirect&bpo=37834]: Treat all name surrogate reparse points on Windows in `os.lstat()` and other reparse points as regular files in `os.stat()`.
- bpo-36266 [https://bugs.python.org/issue?@action=redirect&bpo=36266]: Add the module name in the formatted error message when DLL load fail happens during module import in

`_PyImport_FindSharedFuncptrWindows()`. Patch by Srinivas Nyayapati.

- bpo-25172 [https://bugs.python.org/issue?@action=redirect&bpo=25172]: Trying to import the **`crypt`** module on Windows will result in an **`ImportError`** with a message explaining that the module isn't supported on Windows. On other platforms, if the underlying `_crypt` module is not available, the ImportError will include a message explaining the problem.
- bpo-37778 [https://bugs.python.org/issue?@action=redirect&bpo=37778]: Fixes the icons used for file associations to the Microsoft Store package.
- bpo-37734 [https://bugs.python.org/issue?@action=redirect&bpo=37734]: Fix use of registry values to launch Python from Microsoft Store app.
- bpo-37702 [https://bugs.python.org/issue?@action=redirect&bpo=37702]: Fix memory leak on Windows in creating an SSLContext object or running `urllib.request.urlopen('https://...')`.
- bpo-37672 [https://bugs.python.org/issue?@action=redirect&bpo=37672]: Switch Windows Store package's pip to use bundled `pip.ini` instead of **`PIP_USER`** variable.
- bpo-10945 [https://bugs.python.org/issue?@action=redirect&bpo=10945]: Officially drop support for creating bdist_wininst installers on non-Windows systems.
- bpo-37445 [https://bugs.python.org/issue?@action=redirect&bpo=37445]: Include the `FORMAT_MESSAGE_IGNORE_INSERTS` flag in `FormatMessageW()` calls.
- bpo-37369 [https://bugs.python.org/issue?@action=redirect&bpo=37369]: Fixes path for **`sys.executable`** when running from the Microsoft Store.
- bpo-37380 [https://bugs.python.org/issue?@action=redirect&bpo=37380]: Don't collect unfinished processes with `subprocess._active` on Windows to cleanup later. Patch by Ruslan Kuprieiev.
- bpo-37351 [https://bugs.python.org/issue?@action=redirect&bpo=37351]: Removes libpython38.a from standard Windows distribution.
- bpo-35360 [https://bugs.python.org/issue?

@action=redirect&bpo=35360]: Update Windows builds to use SQLite 3.28.0.
- bpo-37267 [https://bugs.python.org/issue?@action=redirect&bpo=37267]: On Windows, `os.dup()` no longer creates an inheritable fd when handling a character file.
- bpo-36779 [https://bugs.python.org/issue?@action=redirect&bpo=36779]: Ensure `time.tzname` is correct on Windows when the active code page is set to CP_UTF7 or CP_UTF8.
- bpo-32587 [https://bugs.python.org/issue?@action=redirect&bpo=32587]: Make `winreg.REG_MULTI_SZ` support zero-length strings.
- bpo-28269 [https://bugs.python.org/issue?@action=redirect&bpo=28269]: Replace use of `strcasecmp()` for the system function `_stricmp()`. Patch by Minmin Gong.
- bpo-36590 [https://bugs.python.org/issue?@action=redirect&bpo=36590]: Add native Bluetooth RFCOMM support to socket module.

macOS

- bpo-38117 [https://bugs.python.org/issue?@action=redirect&bpo=38117]: Updated OpenSSL to 1.1.1d in macOS installer.
- bpo-38089 [https://bugs.python.org/issue?@action=redirect&bpo=38089]: Move Azure Pipelines to latest VM versions and make macOS tests optional
- bpo-18049 [https://bugs.python.org/issue?@action=redirect&bpo=18049]: Increase the default stack size of threads from 5MB to 16MB on macOS, to match the stack size of the main thread. This avoids crashes on deep recursion in threads.
- bpo-34602 [https://bugs.python.org/issue?@action=redirect&bpo=34602]: Avoid test suite failures on macOS by no longer calling resource.setrlimit to increase the process stack size limit at runtime. The runtime change is no longer needed since the interpreter is being built with a larger default stack size.

- bpo-35360 [https://bugs.python.org/issue?@action=redirect&bpo=35360]: Update macOS installer to use SQLite 3.28.0.
- bpo-34631 [https://bugs.python.org/issue?@action=redirect&bpo=34631]: Updated OpenSSL to 1.1.1c in macOS installer.

IDLE

- bpo-26353 [https://bugs.python.org/issue?@action=redirect&bpo=26353]: Stop adding newline when saving an IDLE shell window.
- bpo-4630 [https://bugs.python.org/issue?@action=redirect&bpo=4630]: Add an option to toggle IDLE's cursor blink for shell, editor, and output windows. See Settings, General, Window Preferences, Cursor Blink. Patch by Zackery Spytz.
- bpo-38598 [https://bugs.python.org/issue?@action=redirect&bpo=38598]: Do not try to compile IDLE shell or output windows
- bpo-36698 [https://bugs.python.org/issue?@action=redirect&bpo=36698]: IDLE no longer fails when write non-encodable characters to stderr. It now escapes them with a backslash, as the regular Python interpreter. Added the `errors` field to the standard streams.
- bpo-35379 [https://bugs.python.org/issue?@action=redirect&bpo=35379]: When exiting IDLE, catch any AttributeError. One happens when EditorWindow.close is called twice. Printing a traceback, when IDLE is run from a terminal, is useless and annoying.
- bpo-38183 [https://bugs.python.org/issue?@action=redirect&bpo=38183]: To avoid problems, test_idle ignores the user config directory. It no longer tries to create or access .idlerc or any files within. Users must run IDLE to discover problems with saving settings.
- bpo-38077 [https://bugs.python.org/issue?@action=redirect&bpo=38077]: IDLE no longer adds "argv" to the user namespace when initializing it. This bug only affected 3.7.4 and 3.8.0b2 to 3.8.0b4.
- bpo-38041 [https://bugs.python.org/issue?@action=redirect&bpo=38041]: Shell restart lines now fill the

window width, always start with “=”, and avoid wrapping unnecessarily. The line will still wrap if the included file name is long relative to the width.

- bpo-35771 [https://bugs.python.org/issue?@action=redirect&bpo=35771]: To avoid occasional spurious test_idle failures on slower machines, increase the `hover_delay` in test_tooltip.
- bpo-37824 [https://bugs.python.org/issue?@action=redirect&bpo=37824]: Properly handle user input warnings in IDLE shell. Cease turning SyntaxWarnings into SyntaxErrors.
- bpo-37929 [https://bugs.python.org/issue?@action=redirect&bpo=37929]: IDLE Settings dialog now closes properly when there is no shell window.
- bpo-37902 [https://bugs.python.org/issue?@action=redirect&bpo=37902]: Add mousewheel scrolling for IDLE module, path, and stack browsers. Patch by George Zhang.
- bpo-37849 [https://bugs.python.org/issue?@action=redirect&bpo=37849]: Fixed completions list appearing too high or low when shown above the current line.
- bpo-36419 [https://bugs.python.org/issue?@action=redirect&bpo=36419]: Refactor IDLE autocomplete and improve testing.
- bpo-37748 [https://bugs.python.org/issue?@action=redirect&bpo=37748]: Reorder the Run menu. Put the most common choice, Run Module, at the top.
- bpo-37692 [https://bugs.python.org/issue?@action=redirect&bpo=37692]: Improve highlight config sample with example shell interaction and better labels for shell elements.
- bpo-37628 [https://bugs.python.org/issue?@action=redirect&bpo=37628]: Settings dialog no longer expands with font size.
- bpo-37627 [https://bugs.python.org/issue?@action=redirect&bpo=37627]: Initialize the Customize Run dialog with the command line arguments most recently entered before. The user can optionally edit before submitting them.
- bpo-33610 [https://bugs.python.org/issue?

@action=redirect&bpo=33610]: Fix code context not showing the correct context when first toggled on.

- bpo-37530 [https://bugs.python.org/issue?@action=redirect&bpo=37530]: Optimize code context to reduce unneeded background activity. Font and highlight changes now occur along with text changes instead of after a random delay.
- bpo-27452 [https://bugs.python.org/issue?@action=redirect&bpo=27452]: Cleanup `config.py` by inlining `RemoveFile` and simplifying the handling of `file` in `CreateConfigHandlers`.
- bpo-37325 [https://bugs.python.org/issue?@action=redirect&bpo=37325]: Fix tab focus traversal order for help source and custom run dialogs.
- bpo-37321 [https://bugs.python.org/issue?@action=redirect&bpo=37321]: Both subprocess connection error messages now refer to the “Startup failure” section of the IDLE doc.
- bpo-17535 [https://bugs.python.org/issue?@action=redirect&bpo=17535]: Add optional line numbers for IDLE editor windows. Windows open without line numbers unless set otherwise in the General tab of the configuration dialog.
- bpo-26806 [https://bugs.python.org/issue?@action=redirect&bpo=26806]: To compensate for stack frames added by IDLE and avoid possible problems with low recursion limits, add 30 to limits in the user code execution process. Subtract 30 when reporting recursion limits to make this addition mostly transparent.
- bpo-37177 [https://bugs.python.org/issue?@action=redirect&bpo=37177]: Properly “attach” search dialogs to their main window so that they behave like other dialogs and do not get hidden behind their main window.
- bpo-37039 [https://bugs.python.org/issue?@action=redirect&bpo=37039]: Adjust «Zoom Height» to individual screens by momentarily maximizing the window on first use with a particular screen. Changing screen settings may invalidate the saved height. While a window is maximized, «Zoom Height» has no effect.
- bpo-35763 [https://bugs.python.org/issue?

@action=redirect&bpo=35763]: Make calltip reminder about “/” meaning positional-only less obtrusive by only adding it when there is room on the first line.
- bpo-5680 [https://bugs.python.org/issue?@action=redirect&bpo=5680]: Add “Run… Customized” to the Run menu to run a module with customized settings. Any “command line arguments” entered are added to sys.argv. One can suppress the normal Shell main module restart.
- bpo-36390 [https://bugs.python.org/issue?@action=redirect&bpo=36390]: Gather Format menu functions into format.py. Combine paragraph.py, rstrip.py, and format methods from editor.py.

Tools/Demos

- bpo-38118 [https://bugs.python.org/issue?@action=redirect&bpo=38118]: Update Valgrind suppression file to ignore a false alarm in `PyUnicode_Decode()` when using GCC builtin strcmp().
- bpo-38347 [https://bugs.python.org/issue?@action=redirect&bpo=38347]: pathfix.py: Assume all files that end on “.py” are Python scripts when working recursively.
- bpo-37803 [https://bugs.python.org/issue?@action=redirect&bpo=37803]: pdb’s `--help` and `--version` long options now work.
- bpo-37942 [https://bugs.python.org/issue?@action=redirect&bpo=37942]: Improve ArgumentClinic converter for floats.
- bpo-37704 [https://bugs.python.org/issue?@action=redirect&bpo=37704]: Remove `Tools/scripts/h2py.py`: use cffi to access a C API in Python.
- bpo-37675 [https://bugs.python.org/issue?@action=redirect&bpo=37675]: 2to3 now works when run from a zipped standard library.
- bpo-37034 [https://bugs.python.org/issue?@action=redirect&bpo=37034]: Argument Clinic now uses the argument name on errors with keyword-only argument instead of their position. Patch contributed by Rémi Lapeyre.
- bpo-37064 [https://bugs.python.org/issue?@action=redirect&bpo=37064]: Add option -k to pathscript.py

script: preserve shebang flags. Add option -a to pathscript.py script: add flags.

C API

- bpo-37633 [https://bugs.python.org/issue?@action=redirect&bpo=37633]: Re-export some function compatibility wrappers for macros in `pythonrun.h`.
- bpo-38644 [https://bugs.python.org/issue?@action=redirect&bpo=38644]: Provide **`Py_EnterRecursiveCall()`** and **`Py_LeaveRecursiveCall()`** as regular functions for the limited API. Previously, there were defined as macros, but these macros didn't work with the limited API which cannot access `PyThreadState.recursion_depth` field. Remove `_Py_CheckRecursionLimit` from the stable ABI.
- bpo-38650 [https://bugs.python.org/issue?@action=redirect&bpo=38650]: The global variable **`PyStructSequence_UnnamedField`** is now a constant and refers to a constant string.
- bpo-38540 [https://bugs.python.org/issue?@action=redirect&bpo=38540]: Fixed possible leak in **`PyArg_Parse()`** and similar functions for format units `"es#"` and `"et#"` when the macro **`PY_SSIZE_T_CLEAN`** is not defined.
- bpo-38395 [https://bugs.python.org/issue?@action=redirect&bpo=38395]: Fix a crash in **`weakref.proxy`** objects due to incorrect lifetime management when calling some associated methods that may delete the last reference to object being referenced by the proxy. Patch by Pablo Galindo.
- bpo-36389 [https://bugs.python.org/issue?@action=redirect&bpo=36389]: The `_PyObject_CheckConsistency()` function is now also available in release mode. For example, it can be used to debug a crash in the `visit_decref()` function of the GC.
- bpo-38266 [https://bugs.python.org/issue?@action=redirect&bpo=38266]: Revert the removal of PyThreadState_DeleteCurrent() with documentation.
- bpo-38303 [https://bugs.python.org/issue?@action=redirect&bpo=38303]: Update audioop extension module

to use the stable ABI (PEP-384). Patch by Tyler Kieft.

- bpo-38234 [https://bugs.python.org/issue?@action=redirect&bpo=38234]: **`Py_SetPath()`** now sets `sys.executable` to the program full path (`Py_GetProgramFullPath()`) rather than to the program name (`Py_GetProgramName()`).
- bpo-38234 [https://bugs.python.org/issue?@action=redirect&bpo=38234]: Python ignored arguments passed to **`Py_SetPath()`**, **`Py_SetPythonHome()`** and **`Py_SetProgramName()`**: fix Python initialization to use specified arguments.
- bpo-38205 [https://bugs.python.org/issue?@action=redirect&bpo=38205]: The `Py_UNREACHABLE()` macro now calls `Py_FatalError()`.
- bpo-38140 [https://bugs.python.org/issue?@action=redirect&bpo=38140]: Make dict and weakref offsets opaque for C heap types by passing the offsets through PyMemberDef
- bpo-15088 [https://bugs.python.org/issue?@action=redirect&bpo=15088]: The C function `PyGen_NeedsFinalizing` has been removed. It was not documented, tested or used anywhere within CPython after the implementation of **PEP 442** [https://peps.python.org/pep-0442/]. Patch by Joannah Nanjekye. (Patch by Joannah Nanjekye)
- bpo-36763 [https://bugs.python.org/issue?@action=redirect&bpo=36763]: Options added by `PySys_AddXOption()` are now handled the same way than `PyConfig.xoptions` and command line `-X` options.
- bpo-37926 [https://bugs.python.org/issue?@action=redirect&bpo=37926]: Fix a crash in `PySys_SetArgvEx(0, NULL, 0)`.
- bpo-37879 [https://bugs.python.org/issue?@action=redirect&bpo=37879]: Fix subtype_dealloc to suppress the type decref when the base type is a C heap type
- bpo-37645 [https://bugs.python.org/issue?@action=redirect&bpo=37645]: Add **`_PyObject_FunctionStr()`** to get a user-friendly string representation of a function-like object. Patch by Jeroen Demeyer.

- bpo-29548 [https://bugs.python.org/issue?@action=redirect&bpo=29548]: The functions `PyEval_CallObject`, `PyEval_CallFunction`, `PyEval_CallMethod` and `PyEval_CallObjectWithKeywords` are deprecated. Use **`PyObject_Call()`** and its variants instead.
- bpo-37151 [https://bugs.python.org/issue?@action=redirect&bpo=37151]: `PyCFunction_Call` is now a deprecated alias of **`PyObject_Call()`**.
- bpo-37540 [https://bugs.python.org/issue?@action=redirect&bpo=37540]: The vectorcall protocol now requires that the caller passes only strings as keyword names.
- bpo-37207 [https://bugs.python.org/issue?@action=redirect&bpo=37207]: The vectorcall protocol is now enabled for `type` objects: set `tp_vectorcall` to a vectorcall function to be used instead of `tp_new` and `tp_init` when calling the class itself.
- bpo-21120 [https://bugs.python.org/issue?@action=redirect&bpo=21120]: Exclude Python-ast.h, ast.h and asdl.h from the limited API.
- bpo-37483 [https://bugs.python.org/issue?@action=redirect&bpo=37483]: Add new function `_PyObject_CallOneArg` for calling an object with one positional argument.
- bpo-36763 [https://bugs.python.org/issue?@action=redirect&bpo=36763]: Add **`PyConfig_SetWideStringList()`** function.
- bpo-37337 [https://bugs.python.org/issue?@action=redirect&bpo=37337]: Add fast functions for calling methods: **`_PyObject_VectorcallMethod()`**, **`_PyObject_CallMethodNoArgs()`** and **`_PyObject_CallMethodOneArg()`**.
- bpo-28805 [https://bugs.python.org/issue?@action=redirect&bpo=28805]: The **`METH_FASTCALL`** calling convention has been documented.
- bpo-37221 [https://bugs.python.org/issue?@action=redirect&bpo=37221]: The new function **`PyCode_NewWithPosOnlyArgs()`** allows to create code objects like **`PyCode_New()`**, but with an extra *posonlyargcount* parameter for indicating the number of

positonal-only arguments.

- bpo-37215 [https://bugs.python.org/issue?@action=redirect&bpo=37215]: Fix dtrace issue introduce by bpo-36842 [https://bugs.python.org/issue?@action=redirect&bpo=36842]
- bpo-37194 [https://bugs.python.org/issue?@action=redirect&bpo=37194]: Add a new public **`PyObject_CallNoArgs()`** function to the C API: call a callable Python object without any arguments. It is the most efficient way to call a callback without any argument. On x86-64, for example, `PyObject_CallFunctionObjArgs(func, NULL)` allocates 960 bytes on the stack per call, whereas `PyObject_CallNoArgs(func)` only allocates 624 bytes per call.
- bpo-37170 [https://bugs.python.org/issue?@action=redirect&bpo=37170]: Fix the cast on error in **`PyLong_AsUnsignedLongLongMask()`**.
- bpo-35381 [https://bugs.python.org/issue?@action=redirect&bpo=35381]: Convert posixmodule.c statically allocated types `DirEntryType` and `ScandirIteratorType` to heap-allocated types.
- bpo-34331 [https://bugs.python.org/issue?@action=redirect&bpo=34331]: Use singular/plural noun in error message when instantiating an abstract class with non-overridden abstract method(s).

Python 3.8.0 beta 1

Release date: 2019-06-04

Security

- bpo-35907 [https://bugs.python.org/issue?@action=redirect&bpo=35907]: **CVE 2019-9948** [https://www.cve.org/CVERecord?id=CVE-2019-9948]: Avoid file reading by disallowing `local-file://` and `local_file://` URL schemes in `URLopener().open()` and `URLopener().retrieve()` of **`urllib.request`**.

- bpo-33529 [https://bugs.python.org/issue?@action=redirect&bpo=33529]: Prevent fold function used in email header encoding from entering infinite loop when there are too many non-ASCII characters in a header.
- bpo-33164 [https://bugs.python.org/issue?@action=redirect&bpo=33164]: Updated blake2 implementation which uses secure memset implementation provided by platform.

Core and Builtins

- bpo-35814 [https://bugs.python.org/issue?@action=redirect&bpo=35814]: Allow unpacking in the right hand side of annotated assignments. In particular, `t: Tuple[int, ...] = x, y, *z` is now allowed.
- bpo-37126 [https://bugs.python.org/issue?@action=redirect&bpo=37126]: All structseq objects are now tracked by the garbage collector. Patch by Pablo Galindo.
- bpo-37122 [https://bugs.python.org/issue?@action=redirect&bpo=37122]: Make the *co_argcount* attribute of code objects represent the total number of positional arguments (including positional-only arguments). The value of *co_posonlyargcount* can be used to distinguish which arguments are positional only, and the difference (*co_argcount* - *co_posonlyargcount*) is the number of positional-or-keyword arguments. Patch by Pablo Galindo.
- bpo-20092 [https://bugs.python.org/issue?@action=redirect&bpo=20092]: Constructors of **`int`**, **`float`** and **`complex`** will now use the **`__index__()`** special method, if available and the corresponding method **`__int__()`**, **`__float__()`** or **`__complex__()`** is not available.
- bpo-37087 [https://bugs.python.org/issue?@action=redirect&bpo=37087]: Add native thread ID (TID) support to OpenBSD.
- bpo-26219 [https://bugs.python.org/issue?@action=redirect&bpo=26219]: Implemented per opcode cache mechanism and `LOAD_GLOBAL` instruction use it. `LOAD_GLOBAL` is now about 40% faster. Contributed by Yury Selivanov, and Inada Naoki.
- bpo-37072 [https://bugs.python.org/issue?

@action=redirect&bpo=37072]: Fix crash in PyAST_FromNodeObject() when flags is NULL.

- bpo-37029 [https://bugs.python.org/issue?@action=redirect&bpo=37029]: Freeing a great many small objects could take time quadratic in the number of arenas, due to using linear search to keep `obmalloc.c`'s list of usable arenas sorted by order of number of free memory pools. This is accomplished without search now, leaving the worst-case time linear in the number of arenas. For programs where this quite visibly matters (typically with more than 100 thousand small objects alive simultaneously), this can greatly reduce the time needed to release their memory.
- bpo-26423 [https://bugs.python.org/issue?@action=redirect&bpo=26423]: Fix possible overflow in `wrap_lenfunc()` when `sizeof(long) < sizeof(Py_ssize_t)` (e.g., 64-bit Windows).
- bpo-37050 [https://bugs.python.org/issue?@action=redirect&bpo=37050]: Improve the AST for «debug» f-strings, which use "=" to print out the source of the expression being evaluated. Delete expr_text from the FormattedValue node, and instead use a Constant string node (possibly merged with adjacent constant expressions inside the f-string).
- bpo-22385 [https://bugs.python.org/issue?@action=redirect&bpo=22385]: The `bytes.hex`, `bytearray.hex`, and `memoryview.hex` methods as well as the `binascii.hexlify` and `b2a_hex` functions now have the ability to include an optional separator between hex bytes. This functionality was inspired by MicroPython's hexlify implementation.
- bpo-26836 [https://bugs.python.org/issue?@action=redirect&bpo=26836]: Add **`os.memfd_create()`**.
- bpo-37032 [https://bugs.python.org/issue?@action=redirect&bpo=37032]: Added new `replace()` method to the code type (**`types.CodeType`**).
- bpo-37007 [https://bugs.python.org/issue?@action=redirect&bpo=37007]: Implement **`socket.if_nameindex()`**, **`socket.if_nametoindex()`**, and **`socket.if_indextoname()`** on Windows.
- bpo-36829 [https://bugs.python.org/issue?

@action=redirect&bpo=36829]: **`PyErr_WriteUnraisable()`** now creates a traceback object if there is no current traceback. Moreover, call **`PyErr_NormalizeException()`** and **`PyException_SetTraceback()`** to normalize the exception value. Ignore any error.

- bpo-36878 [https://bugs.python.org/issue?@action=redirect&bpo=36878]: Only accept text after `# type: ignore` if the first character is ASCII. This is to disallow things like `# type: ignoreé`.
- bpo-36878 [https://bugs.python.org/issue?@action=redirect&bpo=36878]: Store text appearing after a `# type: ignore` comment in the AST. For example a type ignore like `# type: ignore[E1000]` will have the string `"[E1000]"` stored in its AST node.
- bpo-2180 [https://bugs.python.org/issue?@action=redirect&bpo=2180]: Treat line continuation at EOF as a `SyntaxError` by Anthony Sottile.
- bpo-36907 [https://bugs.python.org/issue?@action=redirect&bpo=36907]: Fix a crash when calling a C function with a keyword dict (`f(**kwargs)`) and changing the dict `kwargs` while that function is running.
- bpo-36946 [https://bugs.python.org/issue?@action=redirect&bpo=36946]: Fix possible signed integer overflow when handling slices.
- bpo-36826 [https://bugs.python.org/issue?@action=redirect&bpo=36826]: Add NamedExpression kind support to ast_unparse.c
- bpo-1875 [https://bugs.python.org/issue?@action=redirect&bpo=1875]: A **`SyntaxError`** is now raised if a code blocks that will be optimized away (e.g. if conditions that are always false) contains syntax errors. Patch by Pablo Galindo.
- bpo-36027 [https://bugs.python.org/issue?@action=redirect&bpo=36027]: Allow computation of modular inverses via three-argument `pow`: the second argument is now permitted to be negative in the case where the first and third arguments are relatively prime.
- bpo-36861 [https://bugs.python.org/issue?@action=redirect&bpo=36861]: Update the Unicode database to version 12.1.0.
- bpo-28866 [https://bugs.python.org/issue?

@action=redirect&bpo=28866]: Avoid caching attributes of classes which type defines mro() to avoid a hard cache invalidation problem.

- bpo-36851 [https://bugs.python.org/issue?@action=redirect&bpo=36851]: The `FrameType` stack is now correctly cleaned up if the execution ends with a return and the stack is not empty.
- bpo-34616 [https://bugs.python.org/issue?@action=redirect&bpo=34616]: The `compile()` builtin functions now support the `ast.PyCF_ALLOW_TOP_LEVEL_AWAIT` flag, which allow to compile sources that contains top-level `await`, `async with` or `async for`. This is useful to evaluate async-code from with an already async functions; for example in a custom REPL.
- bpo-36842 [https://bugs.python.org/issue?@action=redirect&bpo=36842]: Implement PEP 578, adding sys.audit, io.open_code and related APIs.
- bpo-27639 [https://bugs.python.org/issue?@action=redirect&bpo=27639]: Correct return type for UserList slicing operations. Patch by Michael Blahay, Erick Cervantes, and vaultah
- bpo-36737 [https://bugs.python.org/issue?@action=redirect&bpo=36737]: Move PyRuntimeState.warnings into per-interpreter state (via «module state»).
- bpo-36793 [https://bugs.python.org/issue?@action=redirect&bpo=36793]: Removed `__str__` implementations from builtin types **`bool`**, **`int`**, **`float`**, **`complex`** and few classes from the standard library. They now inherit `__str__()` from **`object`**.
- bpo-36817 [https://bugs.python.org/issue?@action=redirect&bpo=36817]: Add a `=` feature f-strings for debugging. This can precede `!s`, `!r`, or `!a`. It produces the text of the expression, followed by an equal sign, followed by the repr of the value of the expression. So `f'{3*9+15=}'` would be equal to the string `'3*9+15=42'`. If `=` is specified, the default conversion is set to `!r`, unless a format spec is given, in which case the formatting behavior is unchanged, and _format_ will be used.
- bpo-24048 [https://bugs.python.org/issue?@action=redirect&bpo=24048]: Save the live exception during

import.c’s `remove_module()`.

- bpo-27987 [https://bugs.python.org/issue?@action=redirect&bpo=27987]: pymalloc returns memory blocks aligned by 16 bytes, instead of 8 bytes, on 64-bit platforms to conform x86-64 ABI. Recent compilers assume this alignment more often. Patch by Inada Naoki.
- bpo-36601 [https://bugs.python.org/issue?@action=redirect&bpo=36601]: A long-since-meaningless check for `getpid() == main_pid` was removed from Python’s internal C signal handler.
- bpo-36594 [https://bugs.python.org/issue?@action=redirect&bpo=36594]: Fix incorrect use of `%p` in format strings. Patch by Zackery Spytz.
- bpo-36045 [https://bugs.python.org/issue?@action=redirect&bpo=36045]: `builtins.help()` now prefixes `async` for async functions.
- bpo-36084 [https://bugs.python.org/issue?@action=redirect&bpo=36084]: Add native thread ID (TID) to threading.Thread objects (supported platforms: Windows, FreeBSD, Linux, macOS)
- bpo-36035 [https://bugs.python.org/issue?@action=redirect&bpo=36035]: Added fix for broken symlinks in combination with pathlib
- bpo-35983 [https://bugs.python.org/issue?@action=redirect&bpo=35983]: Added new trashcan macros to deal with a double deallocation that could occur when the `tp_dealloc` of a subclass calls the `tp_dealloc` of a base class and that base class uses the trashcan mechanism. Patch by Jeroen Demeyer.
- bpo-20602 [https://bugs.python.org/issue?@action=redirect&bpo=20602]: Do not clear **`sys.flags`** and **`sys.float_info`** during shutdown. Patch by Zackery Spytz.
- bpo-26826 [https://bugs.python.org/issue?@action=redirect&bpo=26826]: Expose **`copy_file_range()`** as a low level API in the **`os`** module.
- bpo-32388 [https://bugs.python.org/issue?@action=redirect&bpo=32388]: Remove cross-version binary compatibility requirement in tp_flags.
- bpo-31862 [https://bugs.python.org/issue?@action=redirect&bpo=31862]: Port binascii to PEP 489

multiphase initialization. Patch by Marcel Plch.

Library

- bpo-37128 [https://bugs.python.org/issue?@action=redirect&bpo=37128]: Added **`math.perm()`**.
- bpo-37120 [https://bugs.python.org/issue?@action=redirect&bpo=37120]: Add SSLContext.num_tickets to control the number of TLSv1.3 session tickets.
- bpo-12202 [https://bugs.python.org/issue?@action=redirect&bpo=12202]: Fix the error handling in **`msilib.SummaryInformation.GetProperty()`**. Patch by Zackery Spytz.
- bpo-26835 [https://bugs.python.org/issue?@action=redirect&bpo=26835]: The fcntl module now contains file sealing constants for sealing of memfds.
- bpo-29262 [https://bugs.python.org/issue?@action=redirect&bpo=29262]: Add `get_origin()` and `get_args()` introspection helpers to `typing` module.
- bpo-12639 [https://bugs.python.org/issue?@action=redirect&bpo=12639]: **`msilib.Directory.start_component()`** no longer fails if *keyfile* is not `None`.
- bpo-36999 [https://bugs.python.org/issue?@action=redirect&bpo=36999]: Add the `asyncio.Task.get_coro()` method to publicly expose the tasks's coroutine object.
- bpo-35246 [https://bugs.python.org/issue?@action=redirect&bpo=35246]: Make **`asyncio.create_subprocess_exec()`** accept path-like arguments.
- bpo-35279 [https://bugs.python.org/issue?@action=redirect&bpo=35279]: Change default *max_workers* of

`ThreadPoolExecutor` from `cpu_count() * 5` to `min(32, cpu_count() + 4)`. Previous value was unreasonably large on many cores machines.

- bpo-37076 [https://bugs.python.org/issue?@action=redirect&bpo=37076]: **`_thread.start_new_thread()`** now logs uncaught exception raised by the function using **`sys.unraisablehook()`**, rather than **`sys.excepthook()`**, so the hook gets access to the function which raised the exception.

- bpo-33725 [https://bugs.python.org/issue?@action=redirect&bpo=33725]: On macOS, the **`multiprocessing`** module now uses *spawn* start method by default.

- bpo-37054 [https://bugs.python.org/issue?@action=redirect&bpo=37054]: Fix destructor **`_pyio.BytesIO`** and **`_pyio.TextIOWrapper`**: initialize their `_buffer` attribute as soon as possible (in the class body), because it's used by `__del__()` which calls `close()`.

- bpo-37058 [https://bugs.python.org/issue?@action=redirect&bpo=37058]: PEP 544: Add `Protocol` and `@runtime_checkable` to the `typing` module.

- bpo-36933 [https://bugs.python.org/issue?@action=redirect&bpo=36933]: The functions `sys.set_coroutine_wrapper` and `sys.get_coroutine_wrapper` that were deprecated and marked for removal in 3.8 have been removed.

- bpo-37047 [https://bugs.python.org/issue?@action=redirect&bpo=37047]: Handle late binding and attribute access in **`unittest.mock.AsyncMock`** setup for autospeccing. Document newly implemented async methods in **`unittest.mock.MagicMock`**.

- bpo-37049 [https://bugs.python.org/issue?@action=redirect&bpo=37049]: PEP 589: Add `TypedDict` to the `typing` module.

- bpo-37046 [https://bugs.python.org/issue?@action=redirect&bpo=37046]: PEP 586: Add `Literal` to the `typing` module.

- bpo-37045 [https://bugs.python.org/issue?@action=redirect&bpo=37045]: PEP 591: Add `Final` qualifier and `@final` decorator to the `typing` module.

- bpo-37035 [https://bugs.python.org/issue?@action=redirect&bpo=37035]: Don’t log OSError based exceptions if a fatal error has occurred in asyncio transport. Peer can generate almost any OSError, user cannot avoid these exceptions by fixing own code. Errors are still propagated to user code, it’s just logging them is pointless and pollute asyncio logs.

- bpo-37001 [https://bugs.python.org/issue?@action=redirect&bpo=37001]: **`symtable.symtable()`** now accepts the same input types for source code as the built-in **`compile()`** function. Patch by Dino Viehland.

- bpo-37028 [https://bugs.python.org/issue?@action=redirect&bpo=37028]: Implement asyncio REPL

- bpo-37027 [https://bugs.python.org/issue?@action=redirect&bpo=37027]: Return safe to use proxy socket object from transport.get_extra_info(“socket”)

- bpo-32528 [https://bugs.python.org/issue?@action=redirect&bpo=32528]: Make asyncio.CancelledError a BaseException.

  This will address the common mistake many asyncio users make: an «except Exception» clause breaking Tasks cancellation.

  In addition to this change, we stop inheriting asyncio.TimeoutError and asyncio.InvalidStateError from their concurrent.futures.* counterparts. There’s no point for these exceptions to share the inheritance chain.

- bpo-1230540 [https://bugs.python.org/issue?@action=redirect&bpo=1230540]: Add a new `threading.excepthook()` function which handles uncaught `threading.Thread.run()` exception. It can be overridden to control how uncaught `threading.Thread.run()` exceptions are handled.
- bpo-36996 [https://bugs.python.org/issue?@action=redirect&bpo=36996]: Handle `unittest.mock.patch()` used as a decorator on async functions.
- bpo-37008 [https://bugs.python.org/issue?@action=redirect&bpo=37008]: Add support for calling `next()` with the mock resulting from `unittest.mock.mock_open()`
- bpo-27737 [https://bugs.python.org/issue?@action=redirect&bpo=27737]: Allow whitespace only header encoding in `email.header` - by Batuhan Taskaya
- bpo-36969 [https://bugs.python.org/issue?@action=redirect&bpo=36969]: PDB command `args` now display positional only arguments. Patch contributed by Rémi Lapeyre.
- bpo-36969 [https://bugs.python.org/issue?@action=redirect&bpo=36969]: PDB command `args` now display keyword only arguments. Patch contributed by Rémi Lapeyre.
- bpo-36983 [https://bugs.python.org/issue?@action=redirect&bpo=36983]: Add missing names to `typing.__all__`: `ChainMap`, `ForwardRef`, `OrderedDict` - by Anthony Sottile.
- bpo-36972 [https://bugs.python.org/issue?@action=redirect&bpo=36972]: Add SupportsIndex protocol to the typing module to allow type checking to detect classes that can be passed to `hex()`, `oct()` and `bin()`.

- bpo-32972 [https://bugs.python.org/issue?@action=redirect&bpo=32972]: Implement `unittest.IsolatedAsyncioTestCase` to help testing asyncio-based code.

- bpo-36952 [https://bugs.python.org/issue?@action=redirect&bpo=36952]: `fileinput.input()` and `fileinput.FileInput` **bufsize** argument has been removed (was deprecated and ignored since Python 3.6), and as a result the **mode** and **openhook** arguments have been made keyword-only.

- bpo-36952 [https://bugs.python.org/issue?@action=redirect&bpo=36952]: Starting with Python 3.3, importing ABCs from `collections` is deprecated, and import should be done from `collections.abc`. Still being able to import from `collections` was marked for removal in 3.8, but has been delayed to 3.9; documentation and `DeprecationWarning` clarified.

- bpo-36949 [https://bugs.python.org/issue?@action=redirect&bpo=36949]: Implement _repr_ for WeakSet objects.

- bpo-36948 [https://bugs.python.org/issue?@action=redirect&bpo=36948]: Fix `NameError` in `urllib.request.URLopener.retrieve()`. Patch by Karthikeyan Singaravelan.

- bpo-33524 [https://bugs.python.org/issue?@action=redirect&bpo=33524]: Fix the folding of email header when the max_line_length is 0 or None and the header contains non-ascii characters. Contributed by Licht Takeuchi (@Licht-T).

- bpo-24564 [https://bugs.python.org/issue?@action=redirect&bpo=24564]: `shutil.copystat()` now ignores `errno.EINVAL` on `os.setxattr()` which may occur when copying files on filesystems without extended attributes support.

Original patch by Giampaolo Rodola, updated by Ying Wang.

- bpo-36888 [https://bugs.python.org/issue?@action=redirect&bpo=36888]: Python child processes can now access the status of their parent process using multiprocessing.process.parent_process

- bpo-36921 [https://bugs.python.org/issue?@action=redirect&bpo=36921]: Deprecate `@coroutine` for sake of `async def`.

- bpo-25652 [https://bugs.python.org/issue?@action=redirect&bpo=25652]: Fix bug in `__rmod__` of `UserString` - by Batuhan Taskaya.

- bpo-36916 [https://bugs.python.org/issue?@action=redirect&bpo=36916]: Remove a message about an unhandled exception in a task when writer.write() is used without await and writer.drain() fails with an exception.

- bpo-36889 [https://bugs.python.org/issue?@action=redirect&bpo=36889]: Introduce **`asyncio.Stream`** class that merges **`asyncio.StreamReader`** and **`asyncio.StreamWriter`** functionality. **`asyncio.Stream`** can work in readonly, writeonly and readwrite modes. Provide **`asyncio.connect()`**, **`asyncio.connect_unix()`**, **`asyncio.connect_read_pipe()`** and **`asyncio.connect_write_pipe()`** factories to open **`asyncio.Stream`** connections. Provide **`asyncio.StreamServer`** and **`UnixStreamServer`** to serve servers with asyncio.Stream API. Modify **`asyncio.create_subprocess_shell()`** and **`asyncio.create_subprocess_exec()`** to use **`asyncio.Stream`** instead of deprecated **`StreamReader`** and **`StreamWriter`**. Deprecate **`asyncio.StreamReader`** and **`asyncio.StreamWriter`**. Deprecate usage of private classes, e.g. **`asyncio.FlowControlMixing`** and **`asyncio.StreamReaderProtocol`** outside of asyncio package.

- bpo-36845 [https://bugs.python.org/issue?@action=redirect&bpo=36845]: Added validation of integer prefixes to the construction of IP networks and interfaces in the ipaddress module.

- bpo-23378 [https://bugs.python.org/issue?@action=redirect&bpo=23378]: Add an extend action to argparser.

- bpo-36867 [https://bugs.python.org/issue?@action=redirect&bpo=36867]: Fix a bug making a SharedMemoryManager instance and its parent process use two separate resource_tracker processes.

- bpo-23896 [https://bugs.python.org/issue?@action=redirect&bpo=23896]: Adds a grammar to lib2to3.pygram that contains exec as a function not as statement.

- bpo-36895 [https://bugs.python.org/issue?@action=redirect&bpo=36895]: The function `time.clock()` was deprecated in 3.3 in favor of `time.perf_counter()` and marked for removal in 3.8, it has removed.

- bpo-35545 [https://bugs.python.org/issue?@action=redirect&bpo=35545]: Fix asyncio discarding IPv6 scopes when ensuring hostname resolutions internally

- bpo-36887 [https://bugs.python.org/issue?@action=redirect&bpo=36887]: Add new function **`math.isqrt()`** to compute integer square roots.

- bpo-34632 [https://bugs.python.org/issue?@action=redirect&bpo=34632]: Introduce the `importlib.metadata` module with (provisional) support for reading metadata from third-party packages.

- bpo-36878 [https://bugs.python.org/issue?@action=redirect&bpo=36878]: When using `type_comments=True` in `ast.parse`, treat `# type: ignore` followed by a non-alphanumeric character and then arbitrary text as a type ignore, instead of requiring nothing but whitespace or another comment. This is to permit formations such as `# type: ignore[E1000]`.

- bpo-36778 [https://bugs.python.org/issue?@action=redirect&bpo=36778]: `cp65001` encoding (Windows code page 65001) becomes an alias to `utf_8` encoding.

- bpo-36867 [https://bugs.python.org/issue?@action=redirect&bpo=36867]: The multiprocessing.resource_tracker replaces the multiprocessing.semaphore_tracker module. Other than semaphores, resource_tracker also tracks shared_memory segments.

- bpo-30262 [https://bugs.python.org/issue?@action=redirect&bpo=30262]: The `Cache` and `Statement` objects of the **`sqlite3`** module are not exposed to the user. Patch by Aviv Palivoda.

- bpo-24538 [https://bugs.python.org/issue?@action=redirect&bpo=24538]: In `shutil.copystat()`, first copy extended file attributes and then file permissions, since extended attributes can only be set on the destination while it is still writeable.

- bpo-36829 [https://bugs.python.org/issue?@action=redirect&bpo=36829]: Add new **`sys.unraisablehook()`** function which can be overridden to control how «unraisable exceptions» are handled. It is called when an exception has occurred but there is no way for Python to handle it. For example, when a destructor raises an exception or during garbage collection (**`gc.collect()`**).

- bpo-36832 [https://bugs.python.org/issue?@action=redirect&bpo=36832]: Introducing `zipfile.Path`, a pathlib-compatible wrapper for traversing zip files.

- bpo-36814 [https://bugs.python.org/issue?@action=redirect&bpo=36814]: Fix an issue where os.posix_spawnp() would incorrectly raise a TypeError when file_actions is None.

- bpo-33110 [https://bugs.python.org/issue?@action=redirect&bpo=33110]: Handle exceptions raised by functions added by concurrent.futures add_done_callback correctly when the Future has already completed.
- bpo-26903 [https://bugs.python.org/issue?@action=redirect&bpo=26903]: Limit `max_workers` in `ProcessPoolExecutor` to 61 to work around a WaitForMultipleObjects limitation.
- bpo-36813 [https://bugs.python.org/issue?@action=redirect&bpo=36813]: Fix **`QueueListener`** to call `queue.task_done()` upon stopping. Patch by Bar Harel.
- bpo-36806 [https://bugs.python.org/issue?@action=redirect&bpo=36806]: Forbid creation of asyncio stream objects like StreamReader, StreamWriter, Process, and their protocols outside of asyncio package.
- bpo-36802 [https://bugs.python.org/issue?@action=redirect&bpo=36802]: Provide both sync and async calls for StreamWriter.write() and StreamWriter.close()
- bpo-36801 [https://bugs.python.org/issue?@action=redirect&bpo=36801]: Properly handle SSL connection closing in asyncio StreamWriter.drain() call.
- bpo-36785 [https://bugs.python.org/issue?@action=redirect&bpo=36785]: Implement PEP 574 (pickle protocol 5 with out-of-band buffers).
- bpo-36772 [https://bugs.python.org/issue?@action=redirect&bpo=36772]: functools.lru_cache() can now be used as a straight decorator in addition to its existing usage as a function that returns a decorator.
- bpo-6584 [https://bugs.python.org/issue?@action=redirect&bpo=6584]: Add a **`BadGzipFile`** exception to the **`gzip`** module.
- bpo-36748 [https://bugs.python.org/issue?

@action=redirect&bpo=36748]: Optimized write buffering in C implementation of `TextIOWrapper`. Writing ASCII string to `TextIOWrapper` with ascii, latin1, or utf-8 encoding is about 20% faster. Patch by Inada Naoki.

- bpo-8138 [https://bugs.python.org/issue?@action=redirect&bpo=8138]: Don't mark `wsgiref.simple_server.SimpleServer` as multi-threaded since `wsgiref.simple_server.WSGIServer` is single-threaded.

- bpo-22640 [https://bugs.python.org/issue?@action=redirect&bpo=22640]: **`py_compile.compile()`** now supports silent mode. Patch by Joannah Nanjekye

- bpo-29183 [https://bugs.python.org/issue?@action=redirect&bpo=29183]: Fix double exceptions in **`wsgiref.handlers.BaseHandler`** by calling its **`close()`** method only when no exception is raised.

- bpo-36548 [https://bugs.python.org/issue?@action=redirect&bpo=36548]: Improved the repr of regular expression flags.

- bpo-36542 [https://bugs.python.org/issue?@action=redirect&bpo=36542]: The signature of Python functions can now be overridden by specifying the `__text_signature__` attribute.

- bpo-36533 [https://bugs.python.org/issue?@action=redirect&bpo=36533]: Reinitialize logging.Handler locks in forked child processes instead of attempting to acquire them all in the parent before forking only to be released in the child process. The acquire/release pattern was leading to deadlocks in code that has implemented any form of chained logging handlers that depend upon one another as the lock acquisition order cannot be guaranteed.

- bpo-35252 [https://bugs.python.org/issue?@action=redirect&bpo=35252]: Throw a TypeError instead of an AssertionError when using an invalid type annotation with singledispatch.

- bpo-35900 [https://bugs.python.org/issue?@action=redirect&bpo=35900]: Allow reduction methods to return a 6-item tuple where the 6th item specifies a custom state-setting method that's called instead of the regular `__setstate__` method.

- bpo-35900 [https://bugs.python.org/issue?@action=redirect&bpo=35900]: enable custom reduction callback registration for functions and classes in _pickle.c, using the new Pickler's attribute `reducer_override`

- bpo-36368 [https://bugs.python.org/issue?@action=redirect&bpo=36368]: Fix a bug crashing SharedMemoryManager instances in interactive sessions after a ctrl-c (KeyboardInterrupt) was sent

- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Fix mmap fail for VxWorks

- bpo-27497 [https://bugs.python.org/issue?@action=redirect&bpo=27497]: **`csv.DictWriter.writeheader()`** now returns the return value of the underlying **`csv.Writer.writerow()`** method. Patch contributed by Ashish Nitin Patil.

- bpo-36239 [https://bugs.python.org/issue?@action=redirect&bpo=36239]: Parsing .mo files now ignores comments starting and ending with #-#-#-#-#.

- bpo-26707 [https://bugs.python.org/issue?@action=redirect&bpo=26707]: Enable plistlib to read and write binary plist files that were created as a KeyedArchive file. Specifically, this allows the plistlib to process 0x80 tokens as UID objects.

- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Add posix module support for VxWorks.

- bpo-35125 [https://bugs.python.org/issue?@action=redirect&bpo=35125]: Asyncio: Remove inner callback on outer cancellation in shield

- bpo-35721 [https://bugs.python.org/issue?@action=redirect&bpo=35721]: Fix **`asyncio.SelectorEventLoop.subprocess_exec()`** leaks file descriptors if `Popen` fails and called with `stdin=subprocess.PIPE`. Patch by Niklas Fiekas.

- bpo-31855 [https://bugs.python.org/issue?@action=redirect&bpo=31855]: **`unittest.mock.mock_open()`** results now respects the argument of read([size]). Patch contributed by Rémi Lapeyre.

- bpo-35431 [https://bugs.python.org/issue?@action=redirect&bpo=35431]: Implement **`math.comb()`** that returns binomial coefficient, that computes the number of ways to choose k items from n items without repetition and without order. Patch by Yash Aggarwal and Keller Fuchs.

- bpo-26660 [https://bugs.python.org/issue?@action=redirect&bpo=26660]: Fixed permission errors in **`TemporaryDirectory`** clean up. Previously `TemporaryDirectory.cleanup()` failed when non-writeable or non-searchable files or directories were created inside a temporary directory.

- bpo-34271 [https://bugs.python.org/issue?@action=redirect&bpo=34271]: Add debugging helpers to ssl module. It’s now possible to dump key material and to trace TLS protocol. The default and stdlib contexts also support SSLKEYLOGFILE env var.

- bpo-26467 [https://bugs.python.org/issue?@action=redirect&bpo=26467]: Added AsyncMock to support using unittest to mock asyncio coroutines. Patch by Lisa Roach.

- bpo-33569 [https://bugs.python.org/issue?@action=redirect&bpo=33569]: dataclasses.InitVar: Exposes the type used to create the init var.

- bpo-34424 [https://bugs.python.org/issue?@action=redirect&bpo=34424]: Fix serialization of messages containing encoded strings when the policy.linesep is set to a multi-character string. Patch by Jens Troeger.

- bpo-34303 [https://bugs.python.org/issue?@action=redirect&bpo=34303]: Performance of `functools.reduce()` is slightly improved. Patch by Sergey Fedoseev.

- bpo-33361 [https://bugs.python.org/issue?@action=redirect&bpo=33361]: Fix a bug in `codecs.StreamRecoder` where seeking might leave old data in a buffer and break subsequent read calls. Patch by Ammar Askar.

- bpo-22454 [https://bugs.python.org/issue?@action=redirect&bpo=22454]: The `shlex` module now exposes `shlex.join()`, the inverse of `shlex.split()`. Patch by Bo Bayles.

- bpo-31922 [https://bugs.python.org/issue?@action=redirect&bpo=31922]: `asyncio.AbstractEventLoop.create_datagram_endpoint(` Do not connect UDP socket when broadcast is allowed. This allows to receive replies after a UDP broadcast.

- bpo-24882 [https://bugs.python.org/issue?@action=redirect&bpo=24882]: Change ThreadPoolExecutor to use existing idle threads before spinning up new ones.

- bpo-31961 [https://bugs.python.org/issue?@action=redirect&bpo=31961]: Added support for bytes and path-like objects in `subprocess.Popen()` on Windows. The *args* parameter now accepts a path-like object if *shell* is `False` and a sequence containing bytes and path-like objects. The *executable* parameter now accepts a bytes and path-like object. The *cwd* parameter now accepts a bytes object. Based on patch by Anders Lorentsen.

- bpo-33123 [https://bugs.python.org/issue?

@action=redirect&bpo=33123]: `pathlib.Path.unlink` now accepts a *missing_ok* parameter to avoid a `FileNotFoundError` from being raised. Patch by Robert Buchholz.

- bpo-32941 [https://bugs.python.org/issue?@action=redirect&bpo=32941]: Allow `mmap.mmap` objects to access the madvise() system call (through `mmap.mmap.madvise()`).

- bpo-22102 [https://bugs.python.org/issue?@action=redirect&bpo=22102]: Added support for ZIP files with disks set to 0. Such files are commonly created by builtin tools on Windows when use ZIP64 extension. Patch by Francisco Facioni.

- bpo-32515 [https://bugs.python.org/issue?@action=redirect&bpo=32515]: trace.py can now run modules via python3 -m trace -t –module module_name

- bpo-32299 [https://bugs.python.org/issue?@action=redirect&bpo=32299]: Changed `unittest.mock.patch.dict()` to return the patched dictionary when used as context manager. Patch by Vadim Tsander.

- bpo-27141 [https://bugs.python.org/issue?@action=redirect&bpo=27141]: Added a `__copy__()` to `collections.UserList` and `collections.UserDict` in order to correctly implement shallow copying of the objects. Patch by Bar Harel.

- bpo-31829 [https://bugs.python.org/issue?@action=redirect&bpo=31829]: `\r`, `\0` and `\x1a` (end-of-file on Windows) are now escaped in protocol 0 pickles of Unicode strings. This allows to load them without loss from files open in text mode in Python 2.

- bpo-23395 [https://bugs.python.org/issue?@action=redirect&bpo=23395]: `_thread.interrupt_main()` now avoids setting the Python error status if the `SIGINT` signal is ignored or not handled by Python.

Documentation

- bpo-36896 [https://bugs.python.org/issue?@action=redirect&bpo=36896]: Clarify that some types have unstable constructor signature between Python versions.
- bpo-36686 [https://bugs.python.org/issue?@action=redirect&bpo=36686]: Improve documentation of the stdin, stdout, and stderr arguments of the `asyncio.subprocess_exec` function to specify which values are supported. Also mention that decoding as text is not supported.

  Add a few tests to verify that the various values passed to the std* arguments actually work.
- bpo-36984 [https://bugs.python.org/issue?@action=redirect&bpo=36984]: Improve version added references in `typing` module - by Anthony Sottile.
- bpo-36868 [https://bugs.python.org/issue?@action=redirect&bpo=36868]: What's new now mentions SSLContext.hostname_checks_common_name instead of SSLContext.host_flags.
- bpo-35924 [https://bugs.python.org/issue?@action=redirect&bpo=35924]: Add a note to the `curses.addstr()` documentation to warn that multiline strings can cause segfaults because of an ncurses bug.
- bpo-36783 [https://bugs.python.org/issue?@action=redirect&bpo=36783]: Added C API Documentation for Time_FromTimeAndFold and PyDateTime_FromDateAndTimeAndFold as per PEP 495. Patch by Edison Abahurire.
- bpo-36797 [https://bugs.python.org/issue?@action=redirect&bpo=36797]: More of the legacy distutils documentation has been either pruned, or else more clearly

marked as being retained solely until the setuptools documentation covers it independently.

- bpo-22865 [https://bugs.python.org/issue?@action=redirect&bpo=22865]: Add detail to the documentation on the `pty.spawn` function.

- bpo-35397 [https://bugs.python.org/issue?@action=redirect&bpo=35397]: Remove deprecation and document urllib.parse.unwrap(). Patch contributed by Rémi Lapeyre.

- bpo-32995 [https://bugs.python.org/issue?@action=redirect&bpo=32995]: Added the context variable in glossary.

- bpo-33519 [https://bugs.python.org/issue?@action=redirect&bpo=33519]: Clarify that `copy()` is not part of the `MutableSequence` ABC.

- bpo-33482 [https://bugs.python.org/issue?@action=redirect&bpo=33482]: Make `codecs.StreamRecoder.writelines` take a list of bytes.

- bpo-25735 [https://bugs.python.org/issue?@action=redirect&bpo=25735]: Added documentation for func factorial to indicate that returns integer values

- bpo-20285 [https://bugs.python.org/issue?@action=redirect&bpo=20285]: Expand object.__doc__ (docstring) to make it clearer. Modify pydoc.py so that help(object) lists object methods (for other classes, help omits methods of the object base class.)

Tests

- bpo-37069 [https://bugs.python.org/issue?@action=redirect&bpo=37069]: Modify test_coroutines, test_cprofile, test_generators, test_raise, test_ssl and test_yield_from to use **`test.support.catch_unraisable_exception()`** rather than `test.support.captured_stderr()`.

- bpo-37098 [https://bugs.python.org/issue?@action=redirect&bpo=37098]: Fix test_memfd_create on older Linux Kernels.
- bpo-37081 [https://bugs.python.org/issue?@action=redirect&bpo=37081]: Test with OpenSSL 1.1.1c
- bpo-36829 [https://bugs.python.org/issue?@action=redirect&bpo=36829]: Add `test.support.catch_unraisable_exception()`: context manager catching unraisable exception using `sys.unraisablehook()`.
- bpo-36915 [https://bugs.python.org/issue?@action=redirect&bpo=36915]: The main regrtest process now always removes all temporary directories of worker processes even if they crash or if they are killed on KeyboardInterrupt (CTRL+c).
- bpo-36719 [https://bugs.python.org/issue?@action=redirect&bpo=36719]: «python3 -m test -jN …» now continues the execution of next tests when a worker process crash (CHILD_ERROR state). Previously, the test suite stopped immediately. Use –failfast to stop at the first error.
- bpo-36816 [https://bugs.python.org/issue?@action=redirect&bpo=36816]: Update Lib/test/selfsigned_pythontestdotnet.pem to match self-signed.pythontest.net's new TLS certificate.
- bpo-35925 [https://bugs.python.org/issue?@action=redirect&bpo=35925]: Skip httplib and nntplib networking tests when they would otherwise fail due to a modern OS or distro with a default OpenSSL policy of rejecting connections to servers with weak certificates.
- bpo-36782 [https://bugs.python.org/issue?@action=redirect&bpo=36782]: Add tests for several C API functions in the `datetime` module. Patch by Edison Abahurire.
- bpo-36342 [https://bugs.python.org/issue?@action=redirect&bpo=36342]: Fix test_multiprocessing in test_venv if platform lacks functioning sem_open.

Build

- bpo-36721 [https://bugs.python.org/issue?@action=redirect&bpo=36721]: To embed Python into an application, a new `--embed` option must be passed to `python3-config --libs --embed` to get `-lpython3.8` (link the application to libpython). To support both 3.8 and older, try `python3-config --libs --embed` first and fallback to `python3-config --libs` (without `--embed`) if the previous command fails.

  Add a pkg-config `python-3.8-embed` module to embed Python into an application: `pkg-config python-3.8-embed --libs` includes `-lpython3.8`. To support both 3.8 and older, try `pkg-config python-X.Y-embed --libs` first and fallback to `pkg-config python-X.Y --libs` (without `--embed`) if the previous command fails (replace `X.Y` with the Python version).

  On the other hand, `pkg-config python3.8 --libs` no longer contains `-lpython3.8`. C extensions must not be linked to libpython (except on Android, case handled by the script); this change is backward incompatible on purpose.

- bpo-36786 [https://bugs.python.org/issue?@action=redirect&bpo=36786]: «make install» now runs compileall in parallel.

Windows

- bpo-36965 [https://bugs.python.org/issue?@action=redirect&bpo=36965]: include of STATUS_CONTROL_C_EXIT without depending on MSC compiler
- bpo-35926 [https://bugs.python.org/issue?@action=redirect&bpo=35926]: Update to OpenSSL 1.1.1b for Windows.
- bpo-29883 [https://bugs.python.org/issue?@action=redirect&bpo=29883]: Add Windows support for UDP transports for the Proactor Event Loop. Patch by Adam Meily.
- bpo-33407 [https://bugs.python.org/issue?@action=redirect&bpo=33407]: The **`Py_DEPRECATED()`** macro

has been implemented for MSVC.

macOS

- bpo-36231 [https://bugs.python.org/issue?@action=redirect&bpo=36231]: Support building Python on macOS without /usr/include installed. As of macOS 10.14, system header files are only available within an SDK provided by either the Command Line Tools or the Xcode app.

IDLE

- bpo-35610 [https://bugs.python.org/issue?@action=redirect&bpo=35610]: Replace now redundant .context_use_ps1 with .prompt_last_line. This finishes change started in bpo-31858 [https://bugs.python.org/issue?@action=redirect&bpo=31858].
- bpo-37038 [https://bugs.python.org/issue?@action=redirect&bpo=37038]: Make idlelib.run runnable; add test clause.
- bpo-36958 [https://bugs.python.org/issue?@action=redirect&bpo=36958]: Print any argument other than None or int passed to SystemExit or sys.exit().
- bpo-36807 [https://bugs.python.org/issue?@action=redirect&bpo=36807]: When saving a file, call os.fsync() so bits are flushed to e.g. USB drive.
- bpo-32411 [https://bugs.python.org/issue?@action=redirect&bpo=32411]: In browser.py, remove extraneous sorting by line number since dictionary was created in line number order.

Tools/Demos

- bpo-37053 [https://bugs.python.org/issue?@action=redirect&bpo=37053]: Handle strings like u»bar» correctly in Tools/parser/unparse.py. Patch by Chih-Hsuan Yen.

C API

- bpo-36763 [https://bugs.python.org/issue?@action=redirect&bpo=36763]: Implement the **PEP 587** [https://peps.python.org/pep-0587/] «Python Initialization Configuration».
- bpo-36379 [https://bugs.python.org/issue?@action=redirect&bpo=36379]: Fix crashes when attempting to use the *modulo* parameter when `__ipow__` is implemented in C.
- bpo-37107 [https://bugs.python.org/issue?@action=redirect&bpo=37107]: Update **`PyObject_CallMethodObjArgs()`** and `_PyObject_CallMethodIdObjArgs` to use `_PyObject_GetMethod` to avoid creating a bound method object in many cases. Patch by Michael J. Sullivan.
- bpo-36974 [https://bugs.python.org/issue?@action=redirect&bpo=36974]: Implement **PEP 590** [https://peps.python.org/pep-0590/]: Vectorcall: a fast calling protocol for CPython. This is a new protocol to optimize calls of custom callable objects.
- bpo-36763 [https://bugs.python.org/issue?@action=redirect&bpo=36763]: `Py_Main()` now returns the exitcode rather than calling `Py_Exit(exitcode)` when calling `PyErr_Print()` if the current exception type is `SystemExit`.
- bpo-36922 [https://bugs.python.org/issue?@action=redirect&bpo=36922]: Add new type flag `Py_TPFLAGS_METHOD_DESCRIPTOR` for objects behaving like unbound methods. These are objects supporting the optimization given by the `LOAD_METHOD/CALL_METHOD` opcodes. See PEP 590.
- bpo-36728 [https://bugs.python.org/issue?@action=redirect&bpo=36728]: The **`PyEval_ReInitThreads()`** function has been removed from the C API. It should not be called explicitly: use **`PyOS_AfterFork_Child()`** instead.

Python 3.8.0 alpha 4

Release date: 2019-05-06

Security

- bpo-36742 [https://bugs.python.org/issue?@action=redirect&bpo=36742]: Fixes mishandling of pre-normalization characters in urlsplit().
- bpo-30458 [https://bugs.python.org/issue?@action=redirect&bpo=30458]: Address **CVE 2019-9740** [https://www.cve.org/CVERecord?id=CVE-2019-9740] by disallowing URL paths with embedded whitespace or control characters through into the underlying http client request. Such potentially malicious header injection URLs now cause an http.client.InvalidURL exception to be raised.
- bpo-35755 [https://bugs.python.org/issue?@action=redirect&bpo=35755]: **`shutil.which()`** now uses `os.confstr("CS_PATH")` if available and if the **`PATH`** environment variable is not set. Remove also the current directory from **`posixpath.defpath`**. On Unix, **`shutil.which()`** and the **`subprocess`** module no longer search the executable in the current directory if the **`PATH`** environment variable is not set.

Core and Builtins

- bpo-36722 [https://bugs.python.org/issue?@action=redirect&bpo=36722]: In debug build, import now also looks for C extensions compiled in release mode and for C extensions compiled in the stable ABI.
- bpo-32849 [https://bugs.python.org/issue?@action=redirect&bpo=32849]: Fix Python Initialization code on FreeBSD to detect properly when stdin file descriptor (fd 0) is invalid.
- bpo-36623 [https://bugs.python.org/issue?@action=redirect&bpo=36623]: Remove parser headers and related function declarations that lack implementations after the removal of pgen.
- bpo-20180 [https://bugs.python.org/issue?@action=redirect&bpo=20180]: `dict.pop()` is now up to 33% faster thanks to Argument Clinic. Patch by Inada Naoki.
- bpo-36611 [https://bugs.python.org/issue?@action=redirect&bpo=36611]: Debug memory allocators: disable

serialno field by default from debug hooks on Python memory allocators to reduce the memory footprint by 5%. Enable **`tracemalloc`** to get the traceback where a memory block has been allocated when a fatal memory error is logged to decide where to put a breakpoint. Compile Python with `PYMEM_DEBUG_SERIALNO` defined to get back the field.

- bpo-36588 [https://bugs.python.org/issue?@action=redirect&bpo=36588]: On AIX, **`sys.platform`** doesn't contain the major version anymore. Always return `'aix'`, instead of `'aix3'` .. `'aix7'`. Since older Python versions include the version number, it is recommended to always use `sys.platform.startswith('aix')`. Contributed by M. Felt.
- bpo-36549 [https://bugs.python.org/issue?@action=redirect&bpo=36549]: Change str.capitalize to use titlecase for the first character instead of uppercase.
- bpo-36540 [https://bugs.python.org/issue?@action=redirect&bpo=36540]: Implement **PEP 570** [https://peps.python.org/pep-0570/] (Python positional-only parameters). Patch by Pablo Galindo.
- bpo-36475 [https://bugs.python.org/issue?@action=redirect&bpo=36475]: **`PyEval_AcquireLock()`** and **`PyEval_AcquireThread()`** now terminate the current thread if called while the interpreter is finalizing, making them consistent with **`PyEval_RestoreThread()`**, **`Py_END_ALLOW_THREADS()`**, and **`PyGILState_Ensure()`**.
- bpo-36504 [https://bugs.python.org/issue?@action=redirect&bpo=36504]: Fix signed integer overflow in _ctypes.c's `PyCArrayType_new()`.
- bpo-20844 [https://bugs.python.org/issue?@action=redirect&bpo=20844]: Fix running script with encoding cookie and LF line ending may fail on Windows.
- bpo-24214 [https://bugs.python.org/issue?@action=redirect&bpo=24214]: Fixed support of the surrogatepass error handler in the UTF-8 incremental decoder.
- bpo-36452 [https://bugs.python.org/issue?@action=redirect&bpo=36452]: Changing `dict` keys during iteration of the dict itself, `keys()`, `values()`, or `items()` will now be detected in certain corner cases where keys are

deleted/added so that the number of keys isn't changed. A `RuntimeError` will be raised after `len(dict)` iterations. Contributed by Thomas Perl.

- bpo-36459 [https://bugs.python.org/issue?@action=redirect&bpo=36459]: Fix a possible double `PyMem_FREE()` due to tokenizer.c's `tok_nextc()`.
- bpo-36433 [https://bugs.python.org/issue?@action=redirect&bpo=36433]: Fixed TypeError message in classmethoddescr_call.
- bpo-36430 [https://bugs.python.org/issue?@action=redirect&bpo=36430]: Fix a possible reference leak in **`itertools.count()`**.
- bpo-36440 [https://bugs.python.org/issue?@action=redirect&bpo=36440]: Include node names in `ParserError` messages, instead of numeric IDs. Patch by A. Skrobov.
- bpo-36143 [https://bugs.python.org/issue?@action=redirect&bpo=36143]: Regenerate **`keyword`** from the Grammar and Tokens file using pgen. Patch by Pablo Galindo.
- bpo-18372 [https://bugs.python.org/issue?@action=redirect&bpo=18372]: Add missing **`PyObject_GC_Track()`** calls in the **`pickle`** module. Patch by Zackery Spytz.

Library

- bpo-35952 [https://bugs.python.org/issue?@action=redirect&bpo=35952]: Fix pythoninfo when the compiler is missing.
- bpo-28238 [https://bugs.python.org/issue?@action=redirect&bpo=28238]: The `.find*()` methods of xml.etree.ElementTree can now search for wildcards like `{*}tag` and `{ns}*` that match a tag in any namespace or all tags in a namespace. Patch by Stefan Behnel.
- bpo-26978 [https://bugs.python.org/issue?@action=redirect&bpo=26978]: `pathlib.path.link_to()` is now implemented. It creates a hard link pointing to a path.
- bpo-1613500 [https://bugs.python.org/issue?@action=redirect&bpo=1613500]: **`fileinput.FileInput`** now uses the input file mode to correctly set the output file mode

(previously it was hardcoded to `'w'`) when `inplace=True` is passed to its constructor.

- bpo-36734 [https://bugs.python.org/issue?@action=redirect&bpo=36734]: Fix compilation of `faulthandler.c` on HP-UX. Initialize `stack_t current_stack` to zero using `memset()`.
- bpo-13611 [https://bugs.python.org/issue?@action=redirect&bpo=13611]: The xml.etree.ElementTree packages gained support for C14N 2.0 serialisation. Patch by Stefan Behnel.
- bpo-36669 [https://bugs.python.org/issue?@action=redirect&bpo=36669]: Add missing matrix multiplication operator support to weakref.proxy.
- bpo-36676 [https://bugs.python.org/issue?@action=redirect&bpo=36676]: The XMLParser() in xml.etree.ElementTree provides namespace prefix context to the parser target if it defines the callback methods «start_ns()» and/or «end_ns()». Patch by Stefan Behnel.
- bpo-36673 [https://bugs.python.org/issue?@action=redirect&bpo=36673]: The TreeBuilder and XMLPullParser in xml.etree.ElementTree gained support for parsing comments and processing instructions. Patch by Stefan Behnel.
- bpo-36650 [https://bugs.python.org/issue?@action=redirect&bpo=36650]: The C version of functools.lru_cache() was treating calls with an empty `**kwargs` dictionary as being distinct from calls with no keywords at all. This did not result in an incorrect answer, but it did trigger an unexpected cache miss.
- bpo-28552 [https://bugs.python.org/issue?@action=redirect&bpo=28552]: Fix `distutils.sysconfig` if **`sys.executable`** is `None` or an empty string: use **`os.getcwd()`** to initialize `project_base`. Fix also the distutils build command: don't use **`sys.executable`** if it is `None` or an empty string.
- bpo-35755 [https://bugs.python.org/issue?@action=redirect&bpo=35755]: **`shutil.which()`** and `distutils.spawn.find_executable` now use `os.confstr("CS_PATH")` if available instead of **`os.defpath`**, if the `PATH` environment variable is not set.

Moreover, don't use `os.confstr("CS_PATH")` nor **`os.defpath`** if the `PATH` environment variable is set to an empty string.

- bpo-25430 [https://bugs.python.org/issue?@action=redirect&bpo=25430]: improve performance of `IPNetwork.__contains__()`
- bpo-30485 [https://bugs.python.org/issue?@action=redirect&bpo=30485]: Path expressions in xml.etree.ElementTree can now avoid explicit namespace prefixes for tags (or the «{namespace}tag» notation) by passing a default namespace with an empty string prefix.
- bpo-36613 [https://bugs.python.org/issue?@action=redirect&bpo=36613]: Fix **`asyncio`** wait() not removing callback if exception
- bpo-36598 [https://bugs.python.org/issue?@action=redirect&bpo=36598]: Fix `isinstance` check for Mock objects with spec when the code is executed under tracing. Patch by Karthikeyan Singaravelan.
- bpo-18748 [https://bugs.python.org/issue?@action=redirect&bpo=18748]: In development mode (**`-X`** `dev`) and in debug build, the **`io.IOBase`** destructor now logs `close()` exceptions. These exceptions are silent by default in release mode.
- bpo-36575 [https://bugs.python.org/issue?@action=redirect&bpo=36575]: The `_lsprof` module now uses internal timer same to `time.perf_counter()` by default. `gettimeofday(2)` was used on Unix. New timer has better resolution on most Unix platforms and timings are no longer impacted by system clock updates since `perf_counter()` is monotonic. Patch by Inada Naoki.
- bpo-33461 [https://bugs.python.org/issue?@action=redirect&bpo=33461]: `json.loads` now emits `DeprecationWarning` when `encoding` option is specified. Patch by Matthias Bussonnier.
- bpo-36559 [https://bugs.python.org/issue?@action=redirect&bpo=36559]: The random module now prefers the lean internal _sha512 module over hashlib for seed(version=2) to optimize import time.
- bpo-17561 [https://bugs.python.org/issue?@action=redirect&bpo=17561]: Set backlog=None as the default

for socket.create_server.

- bpo-34373 [https://bugs.python.org/issue?@action=redirect&bpo=34373]: Fix **`time.mktime()`** error handling on AIX for year before 1970.
- bpo-36232 [https://bugs.python.org/issue?@action=redirect&bpo=36232]: Improve error message when trying to open existing DBM database that actually doesn't exist. Patch by Marco Rougeth.
- bpo-36546 [https://bugs.python.org/issue?@action=redirect&bpo=36546]: Add statistics.quantiles()
- bpo-36050 [https://bugs.python.org/issue?@action=redirect&bpo=36050]: Optimized `http.client.HTTPResponse.read()` for large response. Patch by Inada Naoki.
- bpo-36522 [https://bugs.python.org/issue?@action=redirect&bpo=36522]: If *debuglevel* is set to >0 in **`http.client`**, print all values for headers with multiple values for the same header name. Patch by Matt Houglum.
- bpo-36492 [https://bugs.python.org/issue?@action=redirect&bpo=36492]: Deprecated passing required arguments like *func* as keyword arguments in functions which should accept arbitrary keyword arguments and pass them to other function. Arbitrary keyword arguments (even with names «self» and «func») can now be passed to these functions if the required arguments are passed as positional arguments.
- bpo-27181 [https://bugs.python.org/issue?@action=redirect&bpo=27181]: Add statistics.geometric_mean().
- bpo-30427 [https://bugs.python.org/issue?@action=redirect&bpo=30427]: `os.path.normcase()` relies on `os.fspath()` to check the type of its argument. Redundant checks have been removed from its `posixpath.normcase()` and `ntpath.normcase()` implementations. Patch by Wolfgang Maier.
- bpo-36385 [https://bugs.python.org/issue?@action=redirect&bpo=36385]: Stop rejecting IPv4 octets for being ambiguously octal. Leading zeros are ignored, and no longer are assumed to specify octal octets. Octets are always decimal numbers. Octets must still be no more than three digits, including leading zeroes.
- bpo-36434 [https://bugs.python.org/issue?

@action=redirect&bpo=36434]: Errors during writing to a ZIP file no longer prevent to properly close it.

- bpo-36407 [https://bugs.python.org/issue?@action=redirect&bpo=36407]: Fixed wrong indentation writing for CDATA section in xml.dom.minidom. Patch by Vladimir Surjaninov.
- bpo-36326 [https://bugs.python.org/issue?@action=redirect&bpo=36326]: inspect.getdoc() can now find docstrings for member objects when _slots_ is a dictionary.
- bpo-36366 [https://bugs.python.org/issue?@action=redirect&bpo=36366]: Calling `stop()` on an unstarted or stopped **`unittest.mock.patch()`** object will now return `None` instead of raising **`RuntimeError`**, making the method idempotent. Patch by Karthikeyan Singaravelan.
- bpo-36348 [https://bugs.python.org/issue?@action=redirect&bpo=36348]: The **`imap.IMAP4.logout()`** method no longer ignores silently arbitrary exceptions.
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Add time module support and fix test_time failures for VxWorks.
- bpo-36227 [https://bugs.python.org/issue?@action=redirect&bpo=36227]: Added support for keyword arguments `default_namespace` and `xml_declaration` in functions `ElementTree.tostring()` and `ElementTree.tostringlist()`.
- bpo-36004 [https://bugs.python.org/issue?@action=redirect&bpo=36004]: Added new alternate constructors **`datetime.date.fromisocalendar()`** and **`datetime.datetime.fromisocalendar()`**, which construct date objects from ISO year, week number and weekday; these are the inverse of each class's `isocalendar` method. Patch by Paul Ganssle.
- bpo-35936 [https://bugs.python.org/issue?@action=redirect&bpo=35936]: **`modulefinder`** no longer depends on the deprecated **`imp`** module, and the initializer for **`modulefinder.ModuleFinder`** now has immutable default arguments. Patch by Brandt Bucher.
- bpo-35376 [https://bugs.python.org/issue?@action=redirect&bpo=35376]: **`modulefinder`** correctly handles modules that have the same name as a bad package.

Patch by Brandt Bucher.

- bpo-17396 [https://bugs.python.org/issue?@action=redirect&bpo=17396]: `modulefinder` no longer crashes when encountering syntax errors in followed imports. Patch by Brandt Bucher.
- bpo-35934 [https://bugs.python.org/issue?@action=redirect&bpo=35934]: Added `create_server()` and `has_dualstack_ipv6()` convenience functions to automate the necessary tasks usually involved when creating a server socket, including accepting both IPv4 and IPv6 connections on the same socket. (Contributed by Giampaolo Rodola in bpo-17561 [https://bugs.python.org/issue?@action=redirect&bpo=17561].)
- bpo-23078 [https://bugs.python.org/issue?@action=redirect&bpo=23078]: Add support for `classmethod()` and `staticmethod()` to `unittest.mock.create_autospec()`. Initial patch by Felipe Ochoa.
- bpo-35416 [https://bugs.python.org/issue?@action=redirect&bpo=35416]: Fix potential resource warnings in distutils. Patch by Mickaël Schoentgen.
- bpo-25451 [https://bugs.python.org/issue?@action=redirect&bpo=25451]: Add transparency methods to `tkinter.PhotoImage`. Patch by Zackery Spytz.
- bpo-35082 [https://bugs.python.org/issue?@action=redirect&bpo=35082]: Don’t return deleted attributes when calling dir on a `unittest.mock.Mock`.
- bpo-34547 [https://bugs.python.org/issue?@action=redirect&bpo=34547]: `wsgiref.handlers.BaseHandler` now handles abrupt client connection terminations gracefully. Patch by Petter Strandmark.
- bpo-31658 [https://bugs.python.org/issue?@action=redirect&bpo=31658]: `xml.sax.parse()` now supports path-like. Patch by Mickaël Schoentgen.
- bpo-34139 [https://bugs.python.org/issue?@action=redirect&bpo=34139]: Remove stale unix datagram socket before binding
- bpo-33530 [https://bugs.python.org/issue?@action=redirect&bpo=33530]: Implemented Happy Eyeballs in

`asyncio.create_connection()`. Added two new arguments, *happy_eyeballs_delay* and *interleave*, to specify Happy Eyeballs behavior.
- bpo-33291 [https://bugs.python.org/issue?@action=redirect&bpo=33291]: Do not raise AttributeError when calling the inspect functions isgeneratorfunction, iscoroutinefunction, isasyncgenfunction on a method created from an arbitrary callable. Instead, return False.
- bpo-31310 [https://bugs.python.org/issue?@action=redirect&bpo=31310]: Fix the multiprocessing.semaphore_tracker so it is reused by child processes
- bpo-31292 [https://bugs.python.org/issue?@action=redirect&bpo=31292]: Fix `setup.py check --restructuredtext` for files containing `include` directives.

Documentation

- bpo-36625 [https://bugs.python.org/issue?@action=redirect&bpo=36625]: Remove obsolete comments from docstrings in fractions.Fraction
- bpo-30840 [https://bugs.python.org/issue?@action=redirect&bpo=30840]: Document relative imports
- bpo-36523 [https://bugs.python.org/issue?@action=redirect&bpo=36523]: Add docstring for io.IOBase.writelines().
- bpo-36425 [https://bugs.python.org/issue?@action=redirect&bpo=36425]: New documentation translation: Simplified Chinese [https://docs.python.org/zh-cn/].
- bpo-36345 [https://bugs.python.org/issue?@action=redirect&bpo=36345]: Avoid the duplication of code from `Tools/scripts/serve.py` in using the **`literalinclude`** directive for the basic wsgiref-based web server in the documentation of **`wsgiref`**. Contributed by Stéphane Wirtel.
- bpo-36345 [https://bugs.python.org/issue?@action=redirect&bpo=36345]: Using the code of the `Tools/scripts/serve.py` script as an example in the **`wsgiref`** documentation. Contributed by Stéphane Wirtel.

- bpo-36157 [https://bugs.python.org/issue?@action=redirect&bpo=36157]: Added documentation for PyInterpreterState_Main().
- bpo-33043 [https://bugs.python.org/issue?@action=redirect&bpo=33043]: Updates the docs.python.org page with the addition of a “Contributing to Docs” link at the end of the page (between “Reporting Bugs” and “About Documentation”). Updates the “Found a Bug” page with additional links and information in the Documentation Bugs section.
- bpo-35581 [https://bugs.python.org/issue?@action=redirect&bpo=35581]: @typing.type_check_only now allows type stubs to mark functions and classes not available during runtime.
- bpo-33832 [https://bugs.python.org/issue?@action=redirect&bpo=33832]: Add glossary entry for “magic method”.
- bpo-32913 [https://bugs.python.org/issue?@action=redirect&bpo=32913]: Added re.Match.groupdict example to regex HOWTO.

Tests

- bpo-36719 [https://bugs.python.org/issue?@action=redirect&bpo=36719]: regrtest now always detects uncollectable objects. Previously, the check was only enabled by `--findleaks`. The check now also works with `-jN/--multiprocess N`. `--findleaks` becomes a deprecated alias to `--fail-env-changed`.
- bpo-36725 [https://bugs.python.org/issue?@action=redirect&bpo=36725]: When using multiprocessing mode (-jN), regrtest now better reports errors if a worker process fails, and it exits immediately on a worker thread failure or when interrupted.
- bpo-36454 [https://bugs.python.org/issue?@action=redirect&bpo=36454]: Change test_time.test_monotonic() to test only the lower bound of elapsed time after a sleep command rather than the upper bound. This prevents unnecessary test failures on slow buildbots. Patch by Victor Stinner.

- bpo-32424 [https://bugs.python.org/issue?@action=redirect&bpo=32424]: Improve test coverage for xml.etree.ElementTree. Patch by Gordon P. Hemsley.
- bpo-32424 [https://bugs.python.org/issue?@action=redirect&bpo=32424]: Fix typo in test_cyclic_gc() test for xml.etree.ElementTree. Patch by Gordon P. Hemsley.
- bpo-36635 [https://bugs.python.org/issue?@action=redirect&bpo=36635]: Add a new **`_testinternalcapi`** module to test the internal C API.
- bpo-36629 [https://bugs.python.org/issue?@action=redirect&bpo=36629]: Fix `test_imap4_host_default_value()` of `test_imaplib`: catch also **`errno.ENETUNREACH`** error.
- bpo-36611 [https://bugs.python.org/issue?@action=redirect&bpo=36611]: Fix `test_sys.test_getallocatedblocks()` when **`tracemalloc`** is enabled.
- bpo-36560 [https://bugs.python.org/issue?@action=redirect&bpo=36560]: Fix reference leak hunting in regrtest: compute also deltas (of reference count, allocated memory blocks, file descriptor count) during warmup, to ensure that everything is initialized before starting to hunt reference leaks.
- bpo-36565 [https://bugs.python.org/issue?@action=redirect&bpo=36565]: Fix reference hunting (`python3 -m test -R 3:3`) when Python has no built-in abc module.
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Port test_resource to VxWorks: skip tests cases setting RLIMIT_FSIZE and RLIMIT_CPU.
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Fix test_tabnanny on VxWorks: adjust ENOENT error message.
- bpo-36436 [https://bugs.python.org/issue?@action=redirect&bpo=36436]: Fix `_testcapi.pymem_buffer_overflow()`: handle memory allocation failure.
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Fix test_utf8_mode on VxWorks: Python always use UTF-8 on VxWorks.
- bpo-36341 [https://bugs.python.org/issue?

@action=redirect&bpo=36341]: Fix tests that may fail with PermissionError upon calling bind() on AF_UNIX sockets.

Build

- bpo-36747 [https://bugs.python.org/issue?@action=redirect&bpo=36747]: Remove the stale scriptsinstall Makefile target.
- bpo-21536 [https://bugs.python.org/issue?@action=redirect&bpo=21536]: On Unix, C extensions are no longer linked to libpython except on Android and Cygwin.

  It is now possible for a statically linked Python to load a C extension built using a shared library Python.

  When Python is embedded, `libpython` must not be loaded with `RTLD_LOCAL`, but `RTLD_GLOBAL` instead. Previously, using `RTLD_LOCAL`, it was already not possible to load C extensions which were not linked to `libpython`, such as C extensions of the standard library built by the `*shared*` section of `Modules/Setup`.

  distutils, python-config and python-config.py have been modified.
- bpo-36707 [https://bugs.python.org/issue?@action=redirect&bpo=36707]: `./configure --with-pymalloc` no longer adds the `m` flag to SOABI (sys.implementation.cache_tag). Enabling or disabling pymalloc has no impact on the ABI.
- bpo-36635 [https://bugs.python.org/issue?@action=redirect&bpo=36635]: Change `PyAPI_FUNC(type)`, `PyAPI_DATA(type)` and `PyMODINIT_FUNC` macros of `pyport.h` when `Py_BUILD_CORE_MODULE` is defined. The `Py_BUILD_CORE_MODULE` define must be now be used to build a C extension as a dynamic library accessing Python internals: export the `PyInit_xxx()` function in DLL exports on Windows.

- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Don't build the `_crypt` extension on VxWorks.

- bpo-36618 [https://bugs.python.org/issue?@action=redirect&bpo=36618]: Add `-fmax-type-align=8` to CFLAGS when clang compiler is detected. The pymalloc memory allocator aligns memory on 8 bytes. On x86-64, clang expects alignment on 16 bytes by default and so uses MOVAPS instruction which can lead to segmentation fault. Instruct clang that Python is limited to alignment on 8 bytes to use MOVUPS instruction instead: slower but don't trigger a SIGSEGV if the memory is not aligned on 16 bytes. Sadly, the flag must be added to `CFLAGS` and not just `CFLAGS_NODIST`, since third party C extensions can have the same issue.

- bpo-36605 [https://bugs.python.org/issue?@action=redirect&bpo=36605]: `make tags` and `make TAGS` now also parse `Modules/_io/*.c` and `Modules/_io/*.h`.

- bpo-36465 [https://bugs.python.org/issue?@action=redirect&bpo=36465]: Release builds and debug builds are now ABI compatible: defining the `Py_DEBUG` macro no longer implies the `Py_TRACE_REFS` macro, which introduces the only ABI incompatibility. The `Py_TRACE_REFS` macro, which adds the **`sys.getobjects()`** function and the **`PYTHONDUMPREFS`** environment variable, can be set using the new `./configure --with-trace-refs` build option.

- bpo-36577 [https://bugs.python.org/issue?@action=redirect&bpo=36577]: setup.py now correctly reports missing OpenSSL headers and libraries again.

- bpo-36544 [https://bugs.python.org/issue?@action=redirect&bpo=36544]: Fix regression introduced in bpo-36146 [https://bugs.python.org/issue?@action=redirect&bpo=36146] refactoring setup.py

- bpo-36508 [https://bugs.python.org/issue?@action=redirect&bpo=36508]: `python-config --ldflags` no longer includes flags of the `LINKFORSHARED` variable. The `LINKFORSHARED` variable must only be used to build executables.

- bpo-36503 [https://bugs.python.org/issue?@action=redirect&bpo=36503]: Remove references to «aix3» and «aix4». Patch by M. Felt.

Windows

- bpo-35920 [https://bugs.python.org/issue?@action=redirect&bpo=35920]: Added platform.win32_edition() and platform.win32_is_iot(). Added support for cross-compiling packages for Windows ARM32. Skip tests that are not expected to work on Windows IoT Core ARM32.
- bpo-36649 [https://bugs.python.org/issue?@action=redirect&bpo=36649]: Remove trailing spaces for registry keys when installed via the Store.
- bpo-34144 [https://bugs.python.org/issue?@action=redirect&bpo=34144]: Fixed activate.bat to correctly update codepage when chcp.com returns dots in output. Patch by Lorenz Mende.
- bpo-36509 [https://bugs.python.org/issue?@action=redirect&bpo=36509]: Added preset-iot layout for Windows IoT ARM containers. This layout doesn't contain UI components like tkinter or IDLE. It also doesn't contain files to support on-target builds since Windows ARM32 builds must be cross-compiled when using MSVC.
- bpo-35941 [https://bugs.python.org/issue?@action=redirect&bpo=35941]: enum_certificates function of the ssl module now returns certificates from all available certificate stores inside windows in a query instead of returning only certificates from the system wide certificate store. This includes certificates from these certificate stores: local machine, local machine enterprise, local machine group policy, current user, current user group policy, services, users. ssl.enum_crls() function is changed in the same way to return all certificate revocation lists inside the windows certificate

revocation list stores.

- bpo-36441 [https://bugs.python.org/issue?@action=redirect&bpo=36441]: Fixes creating a venv when debug binaries are installed.
- bpo-36085 [https://bugs.python.org/issue?@action=redirect&bpo=36085]: Enable better DLL resolution on Windows by using safe DLL search paths and adding `os.add_dll_directory()`.
- bpo-36010 [https://bugs.python.org/issue?@action=redirect&bpo=36010]: Add the venv standard library module to the nuget distribution for Windows.
- bpo-29515 [https://bugs.python.org/issue?@action=redirect&bpo=29515]: Add the following socket module constants on Windows: IPPROTO_AH IPPROTO_CBT IPPROTO_DSTOPTS IPPROTO_EGP IPPROTO_ESP IPPROTO_FRAGMENT IPPROTO_GGP IPPROTO_HOPOPTS IPPROTO_ICLFXBM IPPROTO_ICMPV6 IPPROTO_IDP IPPROTO_IGMP IPPROTO_IGP IPPROTO_IPV4 IPPROTO_IPV6 IPPROTO_L2TP IPPROTO_MAX IPPROTO_ND IPPROTO_NONE IPPROTO_PGM IPPROTO_PIM IPPROTO_PUP IPPROTO_RDP IPPROTO_ROUTING IPPROTO_SCTP IPPROTO_ST
- bpo-35947 [https://bugs.python.org/issue?@action=redirect&bpo=35947]: Added current version of libffi to cpython-source-deps. Change _ctypes to use current version of libffi on Windows.
- bpo-34060 [https://bugs.python.org/issue?@action=redirect&bpo=34060]: Report system load when running test suite on Windows. Patch by Ammar Askar. Based on prior work by Jeremy Kloth.
- bpo-31512 [https://bugs.python.org/issue?@action=redirect&bpo=31512]: With the Windows 10 Creators Update, non-elevated users can now create symlinks as long as the computer has Developer Mode enabled.

macOS

- bpo-34602 [https://bugs.python.org/issue?@action=redirect&bpo=34602]: Avoid failures setting macOS stack resource limit with resource.setrlimit. This reverts an earlier fix for bpo-18075 [https://bugs.python.org/issue?@action=redirect&bpo=18075] which forced a non-default stack size when building the interpreter executable on macOS.

IDLE

- bpo-36429 [https://bugs.python.org/issue?@action=redirect&bpo=36429]: Fix starting IDLE with pyshell. Add idlelib.pyshell alias at top; remove pyshell alias at bottom. Remove obsolete _name_ = =”_main_” command.

Tools/Demos

- bpo-14546 [https://bugs.python.org/issue?@action=redirect&bpo=14546]: Fix the argument handling in Tools/scripts/lll.py.

C API

- bpo-36763 [https://bugs.python.org/issue?@action=redirect&bpo=36763]: Fix memory leak in **`Py_SetStandardStreamEncoding()`**: release memory if the function is called twice.
- bpo-36641 [https://bugs.python.org/issue?@action=redirect&bpo=36641]: PyDoc_VAR(name) and PyDoc_STRVAR(name, str) now create `static const char name[]` instead of `static char name[]`. Patch by Inada Naoki.
- bpo-36389 [https://bugs.python.org/issue?@action=redirect&bpo=36389]: Change the value of `CLEANBYTE`, `DEADDYTE` and `FORBIDDENBYTE` internal constants used by debug hooks on Python memory allocators (**`PyMem_SetupDebugHooks()`** function). Byte patterns `0xCB`, `0xDB` and `0xFB` have been replaced with `0xCD`, `0xDD` and `0xFD` to use the same values than Windows CRT debug `malloc()` and `free()`.
- bpo-36443 [https://bugs.python.org/issue?@action=redirect&bpo=36443]: Since Python 3.7.0, calling **`Py_DecodeLocale()`** before **`Py_Initialize()`** produces mojibake if the `LC_CTYPE` locale is coerced and/or if the

UTF-8 Mode is enabled by the user configuration. The LC_CTYPE coercion and UTF-8 Mode are now disabled by default to fix the mojibake issue. They must now be enabled explicitly (opt-in) using the new **`_Py_PreInitialize()`** API with `_PyPreConfig`.

- bpo-36025 [https://bugs.python.org/issue?@action=redirect&bpo=36025]: Fixed an accidental change to the datetime C API where the arguments to the **`PyDate_FromTimestamp()`** function were incorrectly interpreted as a single timestamp rather than an arguments tuple, which causes existing code to start raising **`TypeError`**. The backwards-incompatible change was only present in alpha releases of Python 3.8. Patch by Paul Ganssle.
- bpo-35810 [https://bugs.python.org/issue?@action=redirect&bpo=35810]: Modify `PyObject_Init` to correctly increase the refcount of heap-allocated Type objects. Also fix the refcounts of the heap-allocated types that were either doing this manually or not decreasing the type's refcount in tp_dealloc

Python 3.8.0 alpha 3

Release date: 2019-03-25

Security

- bpo-36216 [https://bugs.python.org/issue?@action=redirect&bpo=36216]: Changes urlsplit() to raise ValueError when the URL contains characters that decompose under IDNA encoding (NFKC-normalization) into characters that affect how the URL is parsed.
- bpo-35121 [https://bugs.python.org/issue?@action=redirect&bpo=35121]: Don't send cookies of domain A without Domain attribute to domain B when domain A is a suffix match of domain B while using a cookiejar with **`http.cookiejar.DefaultCookiePolicy`** policy. Patch by Karthikeyan Singaravelan.

Core and Builtins

- bpo-36421 [https://bugs.python.org/issue?@action=redirect&bpo=36421]: Fix a possible double decref in _ctypes.c's `PyCArrayType_new()`.
- bpo-36412 [https://bugs.python.org/issue?@action=redirect&bpo=36412]: Fix a possible crash when creating a new dictionary.
- bpo-36398 [https://bugs.python.org/issue?@action=redirect&bpo=36398]: Fix a possible crash in `structseq_repr()`.
- bpo-36256 [https://bugs.python.org/issue?@action=redirect&bpo=36256]: Fix bug in parsermodule when parsing a state in a DFA that has two or more arcs with labels of the same type. Patch by Pablo Galindo.
- bpo-36365 [https://bugs.python.org/issue?@action=redirect&bpo=36365]: repr(structseq) is no longer limited to 512 bytes.
- bpo-36374 [https://bugs.python.org/issue?@action=redirect&bpo=36374]: Fix a possible null pointer dereference in `merge_consts_recursive()`. Patch by Zackery Spytz.
- bpo-36236 [https://bugs.python.org/issue?@action=redirect&bpo=36236]: At Python initialization, the current directory is no longer prepended to `sys.path` if it has been removed.
- bpo-36352 [https://bugs.python.org/issue?@action=redirect&bpo=36352]: Python initialization now fails with an error, rather than silently truncating paths, if a path is too long.
- bpo-36301 [https://bugs.python.org/issue?@action=redirect&bpo=36301]: Python initialization now fails if decoding `pybuilddir.txt` configuration file fails at startup.
- bpo-36333 [https://bugs.python.org/issue?@action=redirect&bpo=36333]: Fix leak in _PyRuntimeState_Fini. Contributed by Stéphane Wirtel.
- bpo-36332 [https://bugs.python.org/issue?@action=redirect&bpo=36332]: The builtin `compile()` can now handle AST objects that contain assignment expressions.

Patch by Pablo Galindo.

- bpo-36282 [https://bugs.python.org/issue?@action=redirect&bpo=36282]: Improved error message for too much positional arguments in some builtin functions.
- bpo-30040 [https://bugs.python.org/issue?@action=redirect&bpo=30040]: New empty dict uses fewer memory for now. It used more memory than empty dict created by `dict.clear()`. And empty dict creation and deletion is about 2x faster. Patch by Inada Naoki.
- bpo-36262 [https://bugs.python.org/issue?@action=redirect&bpo=36262]: Fix an unlikely memory leak on conversion from string to float in the function `_Py_dg_strtod()` used by `float(str)`, `complex(str)`, **`pickle.load()`**, **`marshal.load()`**, etc.
- bpo-36252 [https://bugs.python.org/issue?@action=redirect&bpo=36252]: Update Unicode databases to version 12.0.0.
- bpo-36218 [https://bugs.python.org/issue?@action=redirect&bpo=36218]: Fix a segfault occurring when sorting a list of heterogeneous values. Patch contributed by Rémi Lapeyre and Elliot Gorokhovsky.
- bpo-36188 [https://bugs.python.org/issue?@action=redirect&bpo=36188]: Cleaned up left-over vestiges of Python 2 unbound method handling in method objects and documentation. Patch by Martijn Pieters
- bpo-36124 [https://bugs.python.org/issue?@action=redirect&bpo=36124]: Add a new interpreter-specific dict and expose it in the C-API via PyInterpreterState_GetDict(). This parallels PyThreadState_GetDict(). However, extension modules should continue using PyModule_GetState() for their own internal per-interpreter state.
- bpo-35975 [https://bugs.python.org/issue?@action=redirect&bpo=35975]: Add a `feature_version` flag to `ast.parse()` (documented) and `compile()` (hidden) that allows tweaking the parser to support older versions of the grammar. In particular, if `feature_version` is 5 or 6, the hacks for the `async` and `await` keyword from PEP 492 are reinstated. (For 7 or higher, these are unconditionally treated as keywords, but they are still special tokens rather

than `NAME` tokens that the parser driver recognizes.)

- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Use UTF-8 as the system encoding on VxWorks.
- bpo-36048 [https://bugs.python.org/issue?@action=redirect&bpo=36048]: The `__index__()` special method will be used instead of `__int__()` for implicit conversion of Python numbers to C integers. Using the `__int__()` method in implicit conversions has been deprecated.
- bpo-35808 [https://bugs.python.org/issue?@action=redirect&bpo=35808]: Retire pgen and use a modified version of pgen2 to generate the parser. Patch by Pablo Galindo.

Library

- bpo-36401 [https://bugs.python.org/issue?@action=redirect&bpo=36401]: The class documentation created by pydoc now has a separate section for readonly properties.
- bpo-36320 [https://bugs.python.org/issue?@action=redirect&bpo=36320]: The typing.NamedTuple() class has deprecated the _field_types attribute in favor of the _annotations_ attribute which carried the same information. Also, both attributes were converted from OrderedDict to a regular dict.
- bpo-34745 [https://bugs.python.org/issue?@action=redirect&bpo=34745]: Fix `asyncio` ssl memory issues caused by circular references
- bpo-36324 [https://bugs.python.org/issue?@action=redirect&bpo=36324]: Add method to statistics.NormalDist for computing the inverse cumulative normal distribution.
- bpo-36321 [https://bugs.python.org/issue?@action=redirect&bpo=36321]: collections.namedtuple() misspelled the name of an attribute. To be consistent with typing.NamedTuple, the attribute name should have been «_field_defaults» instead of «_fields_defaults». For backwards compatibility, both spellings are now created. The misspelled version may be removed in the future.

- bpo-36297 [https://bugs.python.org/issue?@action=redirect&bpo=36297]: «unicode_internal» codec is removed. It was deprecated since Python 3.3. Patch by Inada Naoki.
- bpo-36298 [https://bugs.python.org/issue?@action=redirect&bpo=36298]: Raise ModuleNotFoundError in pyclbr when a module can't be found. Thanks to "mental" for the bug report.
- bpo-36268 [https://bugs.python.org/issue?@action=redirect&bpo=36268]: Switch the default format used for writing tars with `tarfile` to the modern POSIX.1-2001 pax standard, from the vendor-specific GNU. Contributed by C.A.M. Gerlach.
- bpo-36285 [https://bugs.python.org/issue?@action=redirect&bpo=36285]: Fix integer overflows in the array module. Patch by Stephan Hohe.
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Add _signal module support for VxWorks.
- bpo-36272 [https://bugs.python.org/issue?@action=redirect&bpo=36272]: `logging` does not silently ignore RecursionError anymore. Patch contributed by Rémi Lapeyre.
- bpo-36280 [https://bugs.python.org/issue?@action=redirect&bpo=36280]: Add a kind field to ast.Constant. It is "u" if the literal has a "u" prefix (i.e. a Python 2 style unicode literal), else None.
- bpo-35931 [https://bugs.python.org/issue?@action=redirect&bpo=35931]: The `pdb` `debug` command now gracefully handles all exceptions.
- bpo-36251 [https://bugs.python.org/issue?@action=redirect&bpo=36251]: Fix format strings used for stderrprinter and re.Match reprs. Patch by Stephan Hohe.
- bpo-36235 [https://bugs.python.org/issue?@action=redirect&bpo=36235]: Fix `CFLAGS` in `customize_compiler()` of `distutils.sysconfig`: when the `CFLAGS` environment variable is defined, don't override `CFLAGS` variable with the `OPT` variable anymore. Initial patch written by David Malcolm.
- bpo-35807 [https://bugs.python.org/issue?@action=redirect&bpo=35807]: Update ensurepip to install pip

19.0.3 and setuptools 40.8.0.

- bpo-36139 [https://bugs.python.org/issue?@action=redirect&bpo=36139]: Release GIL when closing **`mmap`** objects.
- bpo-36179 [https://bugs.python.org/issue?@action=redirect&bpo=36179]: Fix two unlikely reference leaks in _hashopenssl. The leaks only occur in out-of-memory cases.
- bpo-36169 [https://bugs.python.org/issue?@action=redirect&bpo=36169]: Add overlap() method to statistics.NormalDist. Computes the overlapping coefficient for two normal distributions.
- bpo-36103 [https://bugs.python.org/issue?@action=redirect&bpo=36103]: Default buffer size used by `shutil.copyfileobj()` is changed from 16 KiB to 64 KiB on non-Windows platform to reduce system call overhead. Contributed by Inada Naoki.
- bpo-36130 [https://bugs.python.org/issue?@action=redirect&bpo=36130]: Fix `pdb` with `skip=...` when stepping into a frame without a `__name__` global. Patch by Anthony Sottile.
- bpo-35652 [https://bugs.python.org/issue?@action=redirect&bpo=35652]: shutil.copytree(copy_function=...) erroneously pass DirEntry instead of a path string.
- bpo-35178 [https://bugs.python.org/issue?@action=redirect&bpo=35178]: Ensure custom **`warnings.formatwarning()`** function can receive `line` as positional argument. Based on patch by Tashrif Billah.
- bpo-36106 [https://bugs.python.org/issue?@action=redirect&bpo=36106]: Resolve potential name clash with libm's sinpi(). Patch by Dmitrii Pasechnik.
- bpo-36091 [https://bugs.python.org/issue?@action=redirect&bpo=36091]: Clean up reference to async generator in Lib/types. Patch by Henry Chen.
- bpo-36043 [https://bugs.python.org/issue?@action=redirect&bpo=36043]: **`FileCookieJar`** supports path-like object. Contributed by Stéphane Wirtel
- bpo-35899 [https://bugs.python.org/issue?@action=redirect&bpo=35899]: Enum has been fixed to correctly handle empty strings and strings with non-Latin characters

(ie. “α”, “א”) without crashing. Original patch contributed by Maxwell. Assisted by Stéphane Wirtel.

- bpo-21269 [https://bugs.python.org/issue?@action=redirect&bpo=21269]: Add `args` and `kwargs` properties to mock call objects. Contributed by Kumar Akshay.
- bpo-30670 [https://bugs.python.org/issue?@action=redirect&bpo=30670]: `pprint.pp` has been added to pretty-print objects with dictionary keys being sorted with their insertion order by default. Parameter *sort_dicts* has been added to `pprint.pprint`, `pprint.pformat` and `pprint.PrettyPrinter`. Contributed by Rémi Lapeyre.
- bpo-35843 [https://bugs.python.org/issue?@action=redirect&bpo=35843]: Implement `__getitem__` for `_NamespacePath`. Patch by Anthony Sottile.
- bpo-35802 [https://bugs.python.org/issue?@action=redirect&bpo=35802]: Clean up code which checked presence of `os.stat` / `os.lstat` / `os.chmod` which are always present. Patch by Anthony Sottile.
- bpo-35715 [https://bugs.python.org/issue?@action=redirect&bpo=35715]: Librates the return value of a ProcessPoolExecutor _process_worker after it’s no longer needed to free memory
- bpo-35493 [https://bugs.python.org/issue?@action=redirect&bpo=35493]: Use **`multiprocessing.connection.wait()`** instead of polling each 0.2 seconds for worker updates in **`multiprocessing.Pool`**. Patch by Pablo Galindo.
- bpo-35661 [https://bugs.python.org/issue?@action=redirect&bpo=35661]: Store the venv prompt in pyvenv.cfg.
- bpo-35121 [https://bugs.python.org/issue?@action=redirect&bpo=35121]: Don’t set cookie for a request when the request path is a prefix match of the cookie’s path attribute but doesn’t end with «/». Patch by Karthikeyan Singaravelan.
- bpo-21478 [https://bugs.python.org/issue?@action=redirect&bpo=21478]: Calls to a child function created with **`unittest.mock.create_autospec()`** should propagate to the parent. Patch by Karthikeyan Singaravelan.

- bpo-35198 [https://bugs.python.org/issue?@action=redirect&bpo=35198]: Fix C++ extension compilation on AIX

Documentation

- bpo-36329 [https://bugs.python.org/issue?@action=redirect&bpo=36329]: Declare the path of the Python binary for the usage of `Tools/scripts/serve.py` when executing `make -C Doc/ serve`. Contributed by Stéphane Wirtel
- bpo-36138 [https://bugs.python.org/issue?@action=redirect&bpo=36138]: Improve documentation about converting datetime.timedelta to scalars.
- bpo-21314 [https://bugs.python.org/issue?@action=redirect&bpo=21314]: A new entry was added to the Core Language Section of the Programming FAQ, which explaines the usage of slash(/) in the signature of a function. Patch by Lysandros Nikolaou

Tests

- bpo-36234 [https://bugs.python.org/issue?@action=redirect&bpo=36234]: test_posix.PosixUidGidTests: add tests for invalid uid/gid type (str). Initial patch written by David Malcolm.
- bpo-29571 [https://bugs.python.org/issue?@action=redirect&bpo=29571]: Fix `test_re.test_locale_flag()`: use `locale.getpreferredencoding()` rather than `locale.getlocale()` to get the locale encoding. With some locales, `locale.getlocale()` returns the wrong encoding.
- bpo-36123 [https://bugs.python.org/issue?@action=redirect&bpo=36123]: Fix race condition in test_socket.

Build

- bpo-36356 [https://bugs.python.org/issue?@action=redirect&bpo=36356]: Fix leaks that led to build failure when configured with address sanitizer.

- bpo-36146 [https://bugs.python.org/issue?@action=redirect&bpo=36146]: Add `TEST_EXTENSIONS` constant to `setup.py` to allow to not build test extensions like `_testcapi`.
- bpo-36146 [https://bugs.python.org/issue?@action=redirect&bpo=36146]: Fix setup.py on macOS: only add `/usr/include/ffi` to include directories of _ctypes, not for all extensions.
- bpo-31904 [https://bugs.python.org/issue?@action=redirect&bpo=31904]: Enable build system to cross-build for VxWorks RTOS.

Windows

- bpo-36312 [https://bugs.python.org/issue?@action=redirect&bpo=36312]: Fixed decoders for the following code pages: 50220, 50221, 50222, 50225, 50227, 50229, 57002 through 57011, 65000 and 42.
- bpo-36264 [https://bugs.python.org/issue?@action=redirect&bpo=36264]: Don't honor POSIX `HOME` in `os.path.expanduser` on windows. Patch by Anthony Sottile.
- bpo-24643 [https://bugs.python.org/issue?@action=redirect&bpo=24643]: Fix name collisions due to `#define timezone _timezone` in PC/pyconfig.h.

IDLE

- bpo-36405 [https://bugs.python.org/issue?@action=redirect&bpo=36405]: Use dict unpacking in idlelib.
- bpo-36396 [https://bugs.python.org/issue?@action=redirect&bpo=36396]: Remove fgBg param of idlelib.config.GetHighlight(). This param was only used twice and changed the return type.
- bpo-36176 [https://bugs.python.org/issue?@action=redirect&bpo=36176]: Fix IDLE autocomplete & calltip popup colors. Prevent conflicts with Linux dark themes (and slightly darken calltip background).
- bpo-23205 [https://bugs.python.org/issue?

@action=redirect&bpo=23205]: For the grep module, add tests for findfiles, refactor findfiles to be a module-level function, and refactor findfiles to use os.walk.

- bpo-23216 [https://bugs.python.org/issue?@action=redirect&bpo=23216]: Add docstrings to IDLE search modules.
- bpo-36152 [https://bugs.python.org/issue?@action=redirect&bpo=36152]: Remove colorizer.ColorDelegator.close_when_done and the corresponding argument of .close(). In IDLE, both have always been None or False since 2007.
- bpo-32129 [https://bugs.python.org/issue?@action=redirect&bpo=32129]: Avoid blurry IDLE application icon on macOS with Tk 8.6. Patch by Kevin Walzer.
- bpo-36096 [https://bugs.python.org/issue?@action=redirect&bpo=36096]: Refactor class variables to instance variables in colorizer.
- bpo-30348 [https://bugs.python.org/issue?@action=redirect&bpo=30348]: Increase test coverage of idlelib.autocomplete by 30%. Patch by Louie Lu

Tools/Demos

- bpo-35132 [https://bugs.python.org/issue?@action=redirect&bpo=35132]: Fix py-list and py-bt commands of python-gdb.py on gdb7.
- bpo-32217 [https://bugs.python.org/issue?@action=redirect&bpo=32217]: Fix freeze script on Windows.

C API

- bpo-36381 [https://bugs.python.org/issue?@action=redirect&bpo=36381]: Raise `DeprecationWarning` when “#” formats are used for building or parsing values without `PY_SSIZE_T_CLEAN`.
- bpo-36142 [https://bugs.python.org/issue?@action=redirect&bpo=36142]: The whole coreconfig.h header is now excluded from Py_LIMITED_API. Move functions definitions into a new internal pycore_coreconfig.h header.

Python 3.8.0 alpha 2

Release date: 2019-02-25

Core and Builtins

- bpo-36052 [https://bugs.python.org/issue?@action=redirect&bpo=36052]: Raise a **`SyntaxError`** when assigning a value to `__debug__` with the Assignment Operator. Contributed by Stéphane Wirtel and Pablo Galindo.
- bpo-36012 [https://bugs.python.org/issue?@action=redirect&bpo=36012]: Doubled the speed of class variable writes. When a non-dunder attribute was updated, there was an unnecessary call to update slots.
- bpo-35942 [https://bugs.python.org/issue?@action=redirect&bpo=35942]: The error message emitted when returning invalid types from `__fspath__` in interfaces that allow passing **`PathLike`** objects has been improved and now it does explain the origin of the error.
- bpo-36016 [https://bugs.python.org/issue?@action=redirect&bpo=36016]: `gc.get_objects` can now receive an optional parameter indicating a generation to get objects from. Patch by Pablo Galindo.
- bpo-1054041 [https://bugs.python.org/issue?@action=redirect&bpo=1054041]: When the main interpreter exits due to an uncaught KeyboardInterrupt, the process now exits in the appropriate manner for its parent process to detect that a SIGINT or ˆC terminated the process. This allows shells and batch scripts to understand that the user has asked them to stop.
- bpo-35992 [https://bugs.python.org/issue?@action=redirect&bpo=35992]: Fix `__class_getitem__()` not being called on a class with a custom non-subscriptable metaclass.
- bpo-35993 [https://bugs.python.org/issue?@action=redirect&bpo=35993]: Fix a crash on fork when using subinterpreters. Contributed by Stéphane Wirtel
- bpo-35991 [https://bugs.python.org/issue?@action=redirect&bpo=35991]: Fix a potential double free in

Modules/_randommodule.c.

- bpo-35961 [https://bugs.python.org/issue?@action=redirect&bpo=35961]: Fix a crash in slice_richcompare(): use strong references rather than stolen references for the two temporary internal tuples.
- bpo-35911 [https://bugs.python.org/issue?@action=redirect&bpo=35911]: Enable the creation of cell objects by adding a `cell.__new__` method, and expose the type `cell` in `Lib/types.py` under the name CellType. Patch by Pierre Glaser.
- bpo-12822 [https://bugs.python.org/issue?@action=redirect&bpo=12822]: Use monotonic clock for `pthread_cond_timedwait` when `pthread_condattr_setclock` and `CLOCK_MONOTONIC` are available.
- bpo-15248 [https://bugs.python.org/issue?@action=redirect&bpo=15248]: The compiler emits now syntax warnings in the case when a comma is likely missed before tuple or list.
- bpo-35886 [https://bugs.python.org/issue?@action=redirect&bpo=35886]: The implementation of PyInterpreterState has been moved into the internal header files (guarded by Py_BUILD_CORE).
- bpo-31506 [https://bugs.python.org/issue?@action=redirect&bpo=31506]: Clarify the errors reported when `object.__new__` and `object.__init__` receive more than one argument. Contributed by Sanyam Khurana.
- bpo-35724 [https://bugs.python.org/issue?@action=redirect&bpo=35724]: Signal-handling is now guaranteed to happen relative to the main interpreter.
- bpo-33608 [https://bugs.python.org/issue?@action=redirect&bpo=33608]: We added a new internal _Py_AddPendingCall() that operates relative to the provided interpreter. This allows us to use the existing implementation to ask another interpreter to do work that cannot be done in the current interpreter, like decref an object the other interpreter owns. The existing Py_AddPendingCall() only operates relative to the main interpreter.
- bpo-33989 [https://bugs.python.org/issue?@action=redirect&bpo=33989]: Fix a possible crash in

`list.sort()` when sorting objects with `ob_type->tp_richcompare == NULL`. Patch by Zackery Spytz.

Library

- bpo-35512 [https://bugs.python.org/issue?@action=redirect&bpo=35512]: `unittest.mock.patch.dict()` used as a decorator with string target resolves the target during function call instead of during decorator construction. Patch by Karthikeyan Singaravelan.
- bpo-36018 [https://bugs.python.org/issue?@action=redirect&bpo=36018]: Add statistics.NormalDist, a tool for creating and manipulating normal distributions of random variable. Features a composite class that treats the mean and standard deviation of measurement data as single entity.
- bpo-35904 [https://bugs.python.org/issue?@action=redirect&bpo=35904]: Added statistics.fmean() as a faster, floating-point variant of the existing mean() function.
- bpo-35918 [https://bugs.python.org/issue?@action=redirect&bpo=35918]: Removed broken `has_key` method from multiprocessing.managers.SyncManager.dict. Contributed by Rémi Lapeyre.
- bpo-18283 [https://bugs.python.org/issue?@action=redirect&bpo=18283]: Add support for bytes to `shutil.which()`.
- bpo-35960 [https://bugs.python.org/issue?@action=redirect&bpo=35960]: Fix `dataclasses.field()` throwing away empty mapping objects passed as metadata.
- bpo-35500 [https://bugs.python.org/issue?@action=redirect&bpo=35500]: Write expected and actual call parameters on separate lines in `unittest.mock.Mock.assert_called_with()` assertion errors. Contributed by Susan Su.
- bpo-35931 [https://bugs.python.org/issue?@action=redirect&bpo=35931]: The `pdb` `debug` command now gracefully handles syntax errors.
- bpo-24209 [https://bugs.python.org/issue?@action=redirect&bpo=24209]: In http.server script, rely on getaddrinfo to bind to preferred address based on the bind

parameter. Now default bind or binding to a name may bind to IPv6 or dual-stack, depending on the environment.

- bpo-35321 [https://bugs.python.org/issue?@action=redirect&bpo=35321]: Set `__spec__.origin` of `_frozen_importlib` to frozen so that it matches the behavior of `_frozen_importlib_external`. Patch by Nina Zakharenko.
- bpo-35378 [https://bugs.python.org/issue?@action=redirect&bpo=35378]: Fix a reference issue inside **`multiprocessing.Pool`** that caused the pool to remain alive if it was deleted without being closed or terminated explicitly. A new strong reference is added to the pool iterators to link the lifetime of the pool to the lifetime of its iterators so the pool does not get destroyed if a pool iterator is still alive.
- bpo-34294 [https://bugs.python.org/issue?@action=redirect&bpo=34294]: re module, fix wrong capturing groups in rare cases. `re.search()`, `re.findall()`, `re.sub()` and other functions that scan through string looking for a match, should reset capturing groups between two match attempts. Patch by Ma Lin.
- bpo-35615 [https://bugs.python.org/issue?@action=redirect&bpo=35615]: `weakref`: Fix a RuntimeError when copying a WeakKeyDictionary or a WeakValueDictionary, due to some keys or values disappearing while iterating.
- bpo-35606 [https://bugs.python.org/issue?@action=redirect&bpo=35606]: Implement `math.prod()` as analogous function to `sum()` that returns the product of a “start” value (default: 1) times an iterable of numbers. Patch by Pablo Galindo.
- bpo-32417 [https://bugs.python.org/issue?@action=redirect&bpo=32417]: Performing arithmetic between `datetime.datetime` subclasses and `datetime.timedelta` now returns an object of the same type as the **`datetime.datetime`** subclass. As a result, `datetime.datetime.astimezone()` and alternate constructors like `datetime.datetime.now()` and **`datetime.fromtimestamp()`** called with a `tz` argument now *also* retain their subclass.

- bpo-35153 [https://bugs.python.org/issue?@action=redirect&bpo=35153]: Add *headers* optional keyword-only parameter to `xmlrpc.client.ServerProxy`, **`xmlrpc.client.Transport`** and **`xmlrpc.client.SafeTransport`**. Patch by Cédric Krier.
- bpo-34572 [https://bugs.python.org/issue?@action=redirect&bpo=34572]: Fix C implementation of pickle.loads to use importlib's locking mechanisms, and thereby avoid using partially loaded modules. Patch by Tim Burgess.

Documentation

- bpo-36083 [https://bugs.python.org/issue?@action=redirect&bpo=36083]: Fix formatting of –check-hash-based-pycs options in the manpage Synopsis.
- bpo-36007 [https://bugs.python.org/issue?@action=redirect&bpo=36007]: Bump minimum sphinx version to 1.8. Patch by Anthony Sottile.
- bpo-22062 [https://bugs.python.org/issue?@action=redirect&bpo=22062]: Update documentation and docstrings for pathlib. Original patch by Mike Short.

Tests

- bpo-27313 [https://bugs.python.org/issue?@action=redirect&bpo=27313]: Avoid test_ttk_guionly ComboboxTest failure with macOS Cocoa Tk.

- bpo-36019 [https://bugs.python.org/issue?@action=redirect&bpo=36019]: Add test.support.TEST_HTTP_URL and replace references of http://www.example.com by this new constant. Contributed by Stéphane Wirtel.

- bpo-36037 [https://bugs.python.org/issue?@action=redirect&bpo=36037]: Fix test_ssl for strict OpenSSL configuration like RHEL8 strict crypto policy. Use older TLS version for minimum TLS version of the server SSL context if needed, to test TLS version older than default minimum TLS version.

- bpo-35798 [https://bugs.python.org/issue?@action=redirect&bpo=35798]: Added **`test.support.check_syntax_warning()`**.

- bpo-35505 [https://bugs.python.org/issue?@action=redirect&bpo=35505]: Make test_imap4_host_default_value independent on whether the local IMAP server is running.

- bpo-35917 [https://bugs.python.org/issue?@action=redirect&bpo=35917]: multiprocessing: provide unit tests for SyncManager and SharedMemoryManager classes + all the shareable types which are supposed to be supported by them. (patch by Giampaolo Rodola)

- bpo-35704 [https://bugs.python.org/issue?@action=redirect&bpo=35704]: Skip `test_shutil.test_unpack_archive_xztar` to prevent a MemoryError on 32-bit AIX when MAXDATA setting is less than 0x20000000.

  Patch by Michael Felt (aixtools)

- bpo-34720 [https://bugs.python.org/issue?@action=redirect&bpo=34720]: Assert m_state != NULL to mimic GC traversal functions that do not correctly handle module creation when the module state has not been created.

Windows

- bpo-35976 [https://bugs.python.org/issue?@action=redirect&bpo=35976]: Added ARM build support to Windows build files in PCBuild.
- bpo-35692 [https://bugs.python.org/issue?@action=redirect&bpo=35692]: `pathlib` no longer raises when checking file and directory existence on drives that are not ready
- bpo-35872 [https://bugs.python.org/issue?@action=redirect&bpo=35872]: Uses the base Python executable when invoking venv in a virtual environment
- bpo-35873 [https://bugs.python.org/issue?

@action=redirect&bpo=35873]: Prevents venv paths being inherited by child processes
- bpo-35299 [https://bugs.python.org/issue?@action=redirect&bpo=35299]: Fix sysconfig detection of the source directory and distutils handling of pyconfig.h during PGO profiling

IDLE

- bpo-24310 [https://bugs.python.org/issue?@action=redirect&bpo=24310]: IDLE – Document settings dialog font tab sample.
- bpo-35833 [https://bugs.python.org/issue?@action=redirect&bpo=35833]: Revise IDLE doc for control codes sent to Shell. Add a code example block.
- bpo-35689 [https://bugs.python.org/issue?@action=redirect&bpo=35689]: Add docstrings and unittests for colorizer.py.

Python 3.8.0 alpha 1

Release date: 2019-02-03

Security

- bpo-35746 [https://bugs.python.org/issue?@action=redirect&bpo=35746]: **CVE 2019-5010** [https://www.cve.org/CVERecord?id=CVE-2019-5010]: Fix a NULL pointer deref in ssl module. The cert parser did not handle CRL distribution points with empty DP or URI correctly. A malicious or buggy certificate can result into segfault. Vulnerability (TALOS-2018-0758) reported by Colin Read and Nicolas Edet of Cisco.
- bpo-34812 [https://bugs.python.org/issue?@action=redirect&bpo=34812]: The `-I` command line option (run Python in isolated mode) is now also copied by the `multiprocessing` and `distutils` modules when spawning child processes. Previously, only `-E` and `-s` options (enabled by `-I`) were copied.

- bpo-34791 [https://bugs.python.org/issue?@action=redirect&bpo=34791]: The xml.sax and xml.dom.domreg no longer use environment variables to override parser implementations when sys.flags.ignore_environment is set by -E or -I arguments.
- bpo-17239 [https://bugs.python.org/issue?@action=redirect&bpo=17239]: The xml.sax and xml.dom.minidom parsers no longer processes external entities by default. External DTD and ENTITY declarations no longer load files or create network connections.
- bpo-34623 [https://bugs.python.org/issue?@action=redirect&bpo=34623]: **CVE 2018-14647** [https://www.cve.org/CVERecord?id=CVE-2018-14647]: The C accelerated _elementtree module now initializes hash randomization salt from _Py_HashSecret instead of libexpat's default CSPRNG.
- bpo-34405 [https://bugs.python.org/issue?@action=redirect&bpo=34405]: Updated to OpenSSL 1.1.0i for Windows builds.
- bpo-33871 [https://bugs.python.org/issue?@action=redirect&bpo=33871]: Fixed sending the part of the file in `os.sendfile()` on macOS. Using the *trailers* argument could cause sending more bytes from the input file than was specified.
- bpo-32533 [https://bugs.python.org/issue?@action=redirect&bpo=32533]: Fixed thread-safety of error handling in _ssl.
- bpo-33136 [https://bugs.python.org/issue?@action=redirect&bpo=33136]: Harden ssl module against LibreSSL **CVE 2018-8970** [https://www.cve.org/CVERecord?id=CVE-2018-8970]. X509_VERIFY_PARAM_set1_host() is called with an explicit namelen. A new test ensures that NULL bytes are not allowed.
- bpo-33001 [https://bugs.python.org/issue?@action=redirect&bpo=33001]: Minimal fix to prevent buffer overrun in os.symlink on Windows
- bpo-32981 [https://bugs.python.org/issue?@action=redirect&bpo=32981]: Regexes in difflib and poplib were vulnerable to catastrophic backtracking. These regexes formed potential DOS vectors (REDOS). They have been refactored. This resolves **CVE 2018-1060** [https://www.cve.org/

CVERecord?id=CVE-2018-1060] and **CVE 2018-1061** [https://www.cve.org/CVERecord?id=CVE-2018-1061]. Patch by Jamie Davis.
- bpo-28414 [https://bugs.python.org/issue?@action=redirect&bpo=28414]: The ssl module now allows users to perform their own IDN en/decoding when using SNI.

Core and Builtins

- bpo-35877 [https://bugs.python.org/issue?@action=redirect&bpo=35877]: Make parenthesis optional for named expressions in while statement. Patch by Karthikeyan Singaravelan.

- bpo-35814 [https://bugs.python.org/issue?@action=redirect&bpo=35814]: Allow same right hand side expressions in annotated assignments as in normal ones. In particular, `x: Tuple[int, int] = 1, 2` (without parentheses on the right) is now allowed.

- bpo-35766 [https://bugs.python.org/issue?@action=redirect&bpo=35766]: Add the option to parse PEP 484 type comments in the ast module. (Off by default.) This is merging the key functionality of the third party fork thereof, [typed_ast](https://github.com/python/typed_ast).

- bpo-35713 [https://bugs.python.org/issue?@action=redirect&bpo=35713]: Reorganize Python initialization to get working exceptions and sys.stderr earlier.

- bpo-33416 [https://bugs.python.org/issue?@action=redirect&bpo=33416]: Add end line and end column position information to the Python AST nodes. This is a C-level backwards incompatible change.

- bpo-35720 [https://bugs.python.org/issue?@action=redirect&bpo=35720]: Fixed a minor memory leak in pymain_parse_cmdline_impl function in Modules/main.c

- bpo-35634 [https://bugs.python.org/issue?@action=redirect&bpo=35634]: `func(**kwargs)` will now raise an error when `kwargs` is a mapping containing multiple entries with the same key. An error was already raised when other keyword arguments are passed before `**kwargs` since Python 3.6.

- bpo-35623 [https://bugs.python.org/issue?@action=redirect&bpo=35623]: Fix a crash when sorting very long lists. Patch by Stephan Hohe.

- bpo-35214 [https://bugs.python.org/issue?@action=redirect&bpo=35214]: clang Memory Sanitizer build instrumentation was added to work around false positives from posix, socket, time, test_io, and test_faulthandler.

- bpo-35560 [https://bugs.python.org/issue?@action=redirect&bpo=35560]: Fix an assertion error in **`format()`** in debug build for floating-point formatting with «n» format, zero padding and small width. Release build is not impacted. Patch by Karthikeyan Singaravelan.

- bpo-35552 [https://bugs.python.org/issue?@action=redirect&bpo=35552]: Format characters `%s` and `%V` in **`PyUnicode_FromFormat()`** and `%s` in **`PyBytes_FromFormat()`** no longer read memory past the limit if *precision* is specified.

- bpo-35504 [https://bugs.python.org/issue?@action=redirect&bpo=35504]: Fix segfaults and **`SystemError`**s when deleting certain attributes. Patch by Zackery Spytz.

- bpo-35504 [https://bugs.python.org/issue?@action=redirect&bpo=35504]: Fixed a SystemError when delete the characters_written attribute of an OSError.

- bpo-35494 [https://bugs.python.org/issue?@action=redirect&bpo=35494]: Improved syntax error messages for unbalanced parentheses in f-string.

- bpo-35444 [https://bugs.python.org/issue?@action=redirect&bpo=35444]: Fixed error handling in pickling methods when fail to look up builtin «getattr». Sped up pickling iterators.

- bpo-35436 [https://bugs.python.org/issue?@action=redirect&bpo=35436]: Fix various issues with memory allocation error handling. Patch by Zackery Spytz.

- bpo-35423 [https://bugs.python.org/issue?@action=redirect&bpo=35423]: Separate the signal handling trigger in the eval loop from the «pending calls» machinery. There is no semantic change and the difference in performance is insignificant.

- bpo-35357 [https://bugs.python.org/issue?@action=redirect&bpo=35357]: Internal attributes" names of unittest.mock._Call and unittest.mock.MagicProxy (name, parent & from_kall) are now prefixed with _mock_ in order to prevent clashes with widely used object attributes. Fixed minor typo in test function name.

- bpo-35372 [https://bugs.python.org/issue?@action=redirect&bpo=35372]: Fixed the code page decoder for input longer than 2 GiB containing undecodable bytes.

- bpo-35336 [https://bugs.python.org/issue?@action=redirect&bpo=35336]: Fix PYTHONCOERCECLOCALE=1 environment variable: only coerce the C locale if the LC_CTYPE locale is «C».

- bpo-31241 [https://bugs.python.org/issue?@action=redirect&bpo=31241]: The *lineno* and *col_offset* attributes of AST nodes for list comprehensions, generator expressions and tuples are now point to the opening parenthesis or square brace. For tuples without parenthesis they point to the position of the first item.

- bpo-33954 [https://bugs.python.org/issue?@action=redirect&bpo=33954]: For `str.format()`, **`float.__format__()`** and **`complex.__format__()`** methods for non-ASCII decimal point when using the «n» formatter.

- bpo-35269 [https://bugs.python.org/issue?@action=redirect&bpo=35269]: Fix a possible segfault involving a newly created coroutine. Patch by Zackery Spytz.

- bpo-35224 [https://bugs.python.org/issue?@action=redirect&bpo=35224]: Implement **PEP 572** [https://peps.python.org/pep-0572/] (assignment expressions). Patch by Emily Morehouse.
- bpo-32492 [https://bugs.python.org/issue?@action=redirect&bpo=32492]: Speed up **`namedtuple()`** attribute access by 1.6x using a C fast-path for the name descriptors. Patch by Pablo Galindo.
- bpo-35214 [https://bugs.python.org/issue?@action=redirect&bpo=35214]: Fixed an out of bounds memory access when parsing a truncated unicode escape sequence at the end of a string such as `'\N'`. It would read one byte beyond the end of the memory allocation.
- bpo-35214 [https://bugs.python.org/issue?@action=redirect&bpo=35214]: The interpreter and extension modules have had annotations added so that they work properly under clang's Memory Sanitizer. A new configure flag –with-memory-sanitizer has been added to make test builds of this nature easier to perform.
- bpo-35193 [https://bugs.python.org/issue?@action=redirect&bpo=35193]: Fix an off by one error in the bytecode peephole optimizer where it could read bytes beyond the end of bounds of an array when removing unreachable code. This bug was present in every release of Python 3.6 and 3.7 until now.
- bpo-35169 [https://bugs.python.org/issue?@action=redirect&bpo=35169]: Improved error messages for forbidden assignments.
- bpo-34022 [https://bugs.python.org/issue?@action=redirect&bpo=34022]: Fix handling of hash-based bytecode files in **`zipimport`**. Patch by Elvis Pranskevichus.
- bpo-28401 [https://bugs.python.org/issue?

@action=redirect&bpo=28401]: Debug builds will no longer to attempt to import extension modules built for the ABI as they were never compatible to begin with. Patch by Stefano Rivera.

- bpo-29341 [https://bugs.python.org/issue?@action=redirect&bpo=29341]: Clarify in the docstrings of **`os`** methods that path-like objects are also accepted as input parameters.

- bpo-35050 [https://bugs.python.org/issue?@action=redirect&bpo=35050]: **`socket`**: Fix off-by-one bug in length check for `AF_ALG` name and type.

- bpo-29743 [https://bugs.python.org/issue?@action=redirect&bpo=29743]: Raise **`ValueError`** instead of **`OverflowError`** in case of a negative `_length_` in a **`ctypes.Array`** subclass. Also raise **`TypeError`** instead of **`AttributeError`** for non-integer `_length_`. Original patch by Oren Milman.

- bpo-16806 [https://bugs.python.org/issue?@action=redirect&bpo=16806]: Fix `lineno` and `col_offset` for multi-line string tokens.

- bpo-35029 [https://bugs.python.org/issue?@action=redirect&bpo=35029]: **`SyntaxWarning`** raised as an exception at code generation time will be now replaced with a **`SyntaxError`** for better error reporting.

- bpo-34983 [https://bugs.python.org/issue?@action=redirect&bpo=34983]: Expose **`symtable.Symbol.is_nonlocal()`** in the symtable module. Patch by Pablo Galindo.

- bpo-34974 [https://bugs.python.org/issue?@action=redirect&bpo=34974]: **`bytes`** and **`bytearray`** constructors no longer convert unexpected exceptions (e.g. **`MemoryError`** and **`KeyboardInterrupt`**) to **`TypeError`**.

- bpo-34939 [https://bugs.python.org/issue?

@action=redirect&bpo=34939]: Allow annotated names in module namespace that are declared global before the annotation happens. Patch by Pablo Galindo.

- bpo-34973 [https://bugs.python.org/issue?@action=redirect&bpo=34973]: Fixed crash in `bytes()` when the `list` argument is mutated while it is iterated.
- bpo-34876 [https://bugs.python.org/issue?@action=redirect&bpo=34876]: The *lineno* and *col_offset* attributes of the AST for decorated function and class refer now to the position of the corresponding `def`, `async def` and `class` instead of the position of the first decorator. This leads to more correct line reporting in tracing. This is the only case when the position of child AST nodes can precede the position of the parent AST node.
- bpo-34879 [https://bugs.python.org/issue?@action=redirect&bpo=34879]: Fix a possible null pointer dereference in bytesobject.c. Patch by Zackery Spytz.
- bpo-34784 [https://bugs.python.org/issue?@action=redirect&bpo=34784]: Fix the implementation of PyStructSequence_NewType in order to create heap allocated StructSequences.
- bpo-32912 [https://bugs.python.org/issue?@action=redirect&bpo=32912]: A `SyntaxWarning` is now emitted instead of a `DeprecationWarning` for invalid escape sequences in string and bytes literals.
- bpo-34854 [https://bugs.python.org/issue?@action=redirect&bpo=34854]: Fixed a crash in compiling string annotations containing a lambda with a keyword-only argument that doesn’t have a default value.
- bpo-34850 [https://bugs.python.org/issue?@action=redirect&bpo=34850]: The compiler now produces a `SyntaxWarning` when identity checks (`is` and `is not`) are used with certain types of literals (e.g. strings, ints). These can often work by accident in CPython, but are not

guaranteed by the language spec. The warning advises users to use equality tests (== and `!=`) instead.

- bpo-34824 [https://bugs.python.org/issue?@action=redirect&bpo=34824]: Fix a possible null pointer dereference in Modules/_ssl.c. Patch by Zackery Spytz.

- bpo-30156 [https://bugs.python.org/issue?@action=redirect&bpo=30156]: The C function `property_descr_get()` uses a «cached» tuple to optimize function calls. But this tuple can be discovered in debug mode with **`sys.getobjects()`**. Remove the optimization, it's not really worth it and it causes 3 different crashes last years.

- bpo-34762 [https://bugs.python.org/issue?@action=redirect&bpo=34762]: Fix contextvars C API to use PyObject* pointer types.

- bpo-34751 [https://bugs.python.org/issue?@action=redirect&bpo=34751]: The hash function for tuples is now based on xxHash which gives better collision results on (formerly) pathological cases. Additionally, on 64-bit systems it improves tuple hashes in general. Patch by Jeroen Demeyer with substantial contributions by Tim Peters.

- bpo-34735 [https://bugs.python.org/issue?@action=redirect&bpo=34735]: Fix a memory leak in Modules/timemodule.c. Patch by Zackery Spytz.

- bpo-34683 [https://bugs.python.org/issue?@action=redirect&bpo=34683]: Fixed a bug where some SyntaxError error pointed to locations that were off-by-one.

- bpo-34651 [https://bugs.python.org/issue?@action=redirect&bpo=34651]: Only allow the main interpreter to fork. The avoids the possibility of affecting the main interpreter, which is critical to operation of the runtime.

- bpo-34653 [https://bugs.python.org/issue?@action=redirect&bpo=34653]: Remove unused function PyParser_SimpleParseStringFilename.

- bpo-32236 [https://bugs.python.org/issue?@action=redirect&bpo=32236]: Warn that line buffering is not supported if `open()` is called with binary mode and `buffering=1`.

- bpo-34641 [https://bugs.python.org/issue?@action=redirect&bpo=34641]: Further restrict the syntax of the left-hand side of keyword arguments in function calls. In particular, `f((keyword)=arg)` is now disallowed.

- bpo-34637 [https://bugs.python.org/issue?@action=redirect&bpo=34637]: Make the *start* argument to *sum()* visible as a keyword argument.

- bpo-1621 [https://bugs.python.org/issue?@action=redirect&bpo=1621]: Do not assume signed integer overflow behavior (C undefined behavior) when performing set hash table resizing.

- bpo-34588 [https://bugs.python.org/issue?@action=redirect&bpo=34588]: Fix an off-by-one in the recursive call pruning feature of traceback formatting.

- bpo-34485 [https://bugs.python.org/issue?@action=redirect&bpo=34485]: On Windows, the LC_CTYPE is now set to the user preferred locale at startup. Previously, the LC_CTYPE locale was «C» at startup, but changed when calling setlocale(LC_CTYPE, «») or setlocale(LC_ALL, «»).

- bpo-34485 [https://bugs.python.org/issue?@action=redirect&bpo=34485]: Standard streams like sys.stdout now use the «surrogateescape» error handler, instead of «strict», on the POSIX locale (when the C locale is not coerced and the UTF-8 Mode is disabled).

- bpo-34485 [https://bugs.python.org/issue?@action=redirect&bpo=34485]: Fix the error handler of standard streams like sys.stdout: PYTHONIOENCODING=»:» is now ignored instead of setting the error handler to «strict».

- bpo-34485 [https://bugs.python.org/issue?@action=redirect&bpo=34485]: Python now gets the locale encoding with C code to initialize the encoding of standard streams like sys.stdout. Moreover, the encoding is now initialized to the Python codec name to get a normalized encoding name and to ensure that the codec is loaded. The change avoids importing _bootlocale and _locale modules at startup by default.

- bpo-34527 [https://bugs.python.org/issue?@action=redirect&bpo=34527]: On FreeBSD, Py_DecodeLocale() and Py_EncodeLocale() now also forces the ASCII encoding if the LC_CTYPE locale is «POSIX», not only if the LC_CTYPE locale is «C».
- bpo-34527 [https://bugs.python.org/issue?@action=redirect&bpo=34527]: The UTF-8 Mode is now also enabled by the «POSIX» locale, not only by the «C» locale.
- bpo-34403 [https://bugs.python.org/issue?@action=redirect&bpo=34403]: On HP-UX with C or POSIX locale, sys.getfilesystemencoding() now returns «ascii» instead of «roman8» (when the UTF-8 Mode is disabled and the C locale is not coerced).
- bpo-34523 [https://bugs.python.org/issue?@action=redirect&bpo=34523]: The Python filesystem encoding is now read earlier during the Python initialization.
- bpo-12458 [https://bugs.python.org/issue?@action=redirect&bpo=12458]: Tracebacks show now correct line number for subexpressions in multiline expressions. Tracebacks show now the line number of the first line for multiline expressions instead of the line number of the last subexpression.
- bpo-34408 [https://bugs.python.org/issue?@action=redirect&bpo=34408]: Prevent a null pointer dereference and resource leakage in `PyInterpreterState_New()`.
- bpo-34400 [https://bugs.python.org/issue?@action=redirect&bpo=34400]: Fix undefined behavior in parsetok.c. Patch by Zackery Spytz.

- bpo-33073 [https://bugs.python.org/issue?@action=redirect&bpo=33073]: Added as_integer_ratio to ints to make them more interoperable with floats.
- bpo-34377 [https://bugs.python.org/issue?@action=redirect&bpo=34377]: Update valgrind suppression list to use `_PyObject_Free/_PyObject_Realloc` instead of `PyObject_Free/PyObject_Realloc`.
- bpo-34353 [https://bugs.python.org/issue?@action=redirect&bpo=34353]: Added the «socket» option in the `stat.filemode()` Python implementation to match the C implementation.
- bpo-34320 [https://bugs.python.org/issue?@action=redirect&bpo=34320]: Fix `dict(od)` didn't copy iteration order of OrderedDict.
- bpo-34113 [https://bugs.python.org/issue?@action=redirect&bpo=34113]: Fixed crash on debug builds when opcode stack was adjusted with negative numbers. Patch by Constantin Petrisor.
- bpo-34100 [https://bugs.python.org/issue?@action=redirect&bpo=34100]: Compiler now merges constants in tuples and frozensets recursively. Code attributes like `co_names` are merged too.
- bpo-34151 [https://bugs.python.org/issue?@action=redirect&bpo=34151]: Performance of list concatenation, repetition and slicing operations is slightly improved. Patch by Sergey Fedoseev.
- bpo-34170 [https://bugs.python.org/issue?@action=redirect&bpo=34170]: -X dev: it is now possible to override the memory allocator using PYTHONMALLOC even if the developer mode is enabled.
- bpo-33237 [https://bugs.python.org/issue?@action=redirect&bpo=33237]: Improved **`AttributeError`** message for partially initialized module.

- bpo-34149 [https://bugs.python.org/issue?@action=redirect&bpo=34149]: Fix min and max functions to get default behavior when key is None.

- bpo-34125 [https://bugs.python.org/issue?@action=redirect&bpo=34125]: Profiling of unbound built-in methods now works when `**kwargs` is given.

- bpo-34141 [https://bugs.python.org/issue?@action=redirect&bpo=34141]: Optimized pickling atomic types (None, bool, int, float, bytes, str).

- bpo-34126 [https://bugs.python.org/issue?@action=redirect&bpo=34126]: Fix crashes when profiling certain invalid calls of unbound methods. Patch by Jeroen Demeyer.

- bpo-24618 [https://bugs.python.org/issue?@action=redirect&bpo=24618]: Fixed reading invalid memory when create the code object with too small varnames tuple or too large argument counts.

- bpo-34068 [https://bugs.python.org/issue?@action=redirect&bpo=34068]: In **`io.IOBase.close()`**, ensure that the **`closed`** attribute is not set with a live exception. Patch by Zackery Spytz and Serhiy Storchaka.

- bpo-34087 [https://bugs.python.org/issue?@action=redirect&bpo=34087]: Fix buffer overflow while converting unicode to numeric values.

- bpo-34080 [https://bugs.python.org/issue?@action=redirect&bpo=34080]: Fixed a memory leak in the compiler when it raised some uncommon errors during tokenizing.

- bpo-34066 [https://bugs.python.org/issue?@action=redirect&bpo=34066]: Disabled interruption by Ctrl-C between calling `open()` and entering a **with** block in `with open()`.

- bpo-34042 [https://bugs.python.org/issue?

@action=redirect&bpo=34042]: Fix dict.copy() to maintain correct total refcount (as reported by sys.gettotalrefcount()).

- bpo-33418 [https://bugs.python.org/issue?@action=redirect&bpo=33418]: Fix potential memory leak in function object when it creates reference cycle.

- bpo-33985 [https://bugs.python.org/issue?@action=redirect&bpo=33985]: Implement contextvars.ContextVar.name attribute.

- bpo-33956 [https://bugs.python.org/issue?@action=redirect&bpo=33956]: Update vendored Expat library copy to version 2.2.5.

- bpo-24596 [https://bugs.python.org/issue?@action=redirect&bpo=24596]: Decref the module object in **`PyRun_SimpleFileExFlags()`** before calling **`PyErr_Print()`**. Patch by Zackery Spytz.

- bpo-33451 [https://bugs.python.org/issue?@action=redirect&bpo=33451]: Close directly executed pyc files before calling `PyEval_EvalCode()`.

- bpo-1617161 [https://bugs.python.org/issue?@action=redirect&bpo=1617161]: The hash of **`BuiltinMethodType`** instances (methods of built-in classes) now depends on the hash of the identity of *_self_* instead of its value. The hash and equality of **`ModuleType`** and **`MethodWrapperType`** instances (methods of user-defined classes and some methods of built-in classes like `str.__add__`) now depend on the hash and equality of the identity of *_self_* instead of its value. **`MethodWrapperType`** instances no longer support ordering.

- bpo-33824 [https://bugs.python.org/issue?@action=redirect&bpo=33824]: Fix «LC_ALL=C python3.7 -V»: reset properly the command line parser when the encoding changes after reading the Python configuration.

- bpo-33803 [https://bugs.python.org/issue?

@action=redirect&bpo=33803]: Fix a crash in hamt.c caused by enabling GC tracking for an object that hadn't all of its fields set to NULL.

- bpo-33738 [https://bugs.python.org/issue?@action=redirect&bpo=33738]: Seven macro incompatibilities with the Limited API were fixed, and the macros **`PyIter_Check()`**, **`PyIndex_Check()`** and **`PyExceptionClass_Name()`** were added as functions. A script for automatic macro checks was added.

- bpo-33786 [https://bugs.python.org/issue?@action=redirect&bpo=33786]: Fix asynchronous generators to handle GeneratorExit in athrow() correctly

- bpo-30167 [https://bugs.python.org/issue?@action=redirect&bpo=30167]: `PyRun_SimpleFileExFlags` removes `__cached__` from module in addition to `__file__`.

- bpo-33706 [https://bugs.python.org/issue?@action=redirect&bpo=33706]: Fix a crash in Python initialization when parsing the command line options. Thanks Christoph Gohlke for the bug report and the fix!

- bpo-33597 [https://bugs.python.org/issue?@action=redirect&bpo=33597]: Reduce `PyGC_Head` size from 3 words to 2 words.

- bpo-30654 [https://bugs.python.org/issue?@action=redirect&bpo=30654]: Fixed reset of the SIGINT handler to SIG_DFL on interpreter shutdown even when there was a custom handler set previously. Patch by Philipp Kerling.

- bpo-33622 [https://bugs.python.org/issue?@action=redirect&bpo=33622]: Fixed a leak when the garbage collector fails to add an object with the `__del__` method or referenced by it into the **`gc.garbage`** list. **`PyGC_Collect()`** can now be called when an exception is set and preserves it.

- bpo-33462 [https://bugs.python.org/issue?@action=redirect&bpo=33462]: Make dict and dict views reversible. Patch by Rémi Lapeyre.

- bpo-23722 [https://bugs.python.org/issue?@action=redirect&bpo=23722]: A `RuntimeError` is now raised when the custom metaclass doesn't provide the `__classcell__` entry in the namespace passed to `type.__new__`. A `DeprecationWarning` was emitted in Python 3.6–3.7.

- bpo-33499 [https://bugs.python.org/issue?@action=redirect&bpo=33499]: Add `PYTHONPYCACHEPREFIX` environment variable and `-X` `pycache_prefix` command-line option to set an alternate root directory for writing module bytecode cache files.

- bpo-25711 [https://bugs.python.org/issue?@action=redirect&bpo=25711]: The `zipimport` module has been rewritten in pure Python.

- bpo-33509 [https://bugs.python.org/issue?@action=redirect&bpo=33509]: Fix module_globals parameter of warnings.warn_explicit(): don't crash if module_globals is not a dict.

- bpo-31849 [https://bugs.python.org/issue?@action=redirect&bpo=31849]: Fix signed/unsigned comparison warning in pyhash.c.

- bpo-33475 [https://bugs.python.org/issue?@action=redirect&bpo=33475]: Fixed miscellaneous bugs in converting annotations to strings and optimized parentheses in the string representation.

- bpo-20104 [https://bugs.python.org/issue?@action=redirect&bpo=20104]: Added support for the `setpgroup`, `resetids`, `setsigmask`, `setsigdef` and `scheduler` parameters of `posix_spawn`. Patch by Pablo Galindo.

- bpo-33391 [https://bugs.python.org/issue?@action=redirect&bpo=33391]: Fix a leak in set_symmetric_difference().
- bpo-33363 [https://bugs.python.org/issue?@action=redirect&bpo=33363]: Raise a SyntaxError for `async with` and `async for` statements outside of async functions.
- bpo-28055 [https://bugs.python.org/issue?@action=redirect&bpo=28055]: Fix unaligned accesses in siphash24(). Patch by Rolf Eike Beer.
- bpo-33128 [https://bugs.python.org/issue?@action=redirect&bpo=33128]: Fix a bug that causes PathFinder to appear twice on sys.meta_path. Patch by Pablo Galindo Salgado.
- bpo-33331 [https://bugs.python.org/issue?@action=redirect&bpo=33331]: Modules imported last are now cleared first at interpreter shutdown.
- bpo-33312 [https://bugs.python.org/issue?@action=redirect&bpo=33312]: Fixed clang ubsan (undefined behavior sanitizer) warnings in dictobject.c by adjusting how the internal struct _dictkeysobject shared keys structure is declared.
- bpo-33305 [https://bugs.python.org/issue?@action=redirect&bpo=33305]: Improved syntax error messages for invalid numerical literals.
- bpo-33306 [https://bugs.python.org/issue?@action=redirect&bpo=33306]: Improved syntax error messages for unbalanced parentheses.
- bpo-33234 [https://bugs.python.org/issue?@action=redirect&bpo=33234]: The list constructor will pre-size and not over-allocate when the input length is known.
- bpo-33270 [https://bugs.python.org/issue?@action=redirect&bpo=33270]: Intern the names for all

anonymous code objects. Patch by Zackery Spytz.

- bpo-30455 [https://bugs.python.org/issue?@action=redirect&bpo=30455]: The C and Python code and the documentation related to tokens are now generated from a single source file `Grammar/Tokens`.
- bpo-33176 [https://bugs.python.org/issue?@action=redirect&bpo=33176]: Add a `toreadonly()` method to memoryviews.
- bpo-33231 [https://bugs.python.org/issue?@action=redirect&bpo=33231]: Fix potential memory leak in `normalizestring()`.
- bpo-33205 [https://bugs.python.org/issue?@action=redirect&bpo=33205]: Change dict growth function from `round_up_to_power_2(used*2+hashtable_size/2)` to `round_up_to_power_2(used*3)`. Previously, dict is shrunk only when `used == 0`. Now dict has more chance to be shrunk.
- bpo-29922 [https://bugs.python.org/issue?@action=redirect&bpo=29922]: Improved error messages in “async with” when `__aenter__()` or `__aexit__()` return non-awaitable object.
- bpo-33199 [https://bugs.python.org/issue?@action=redirect&bpo=33199]: Fix `ma_version_tag` in dict implementation is uninitialized when copying from key-sharing dict.
- bpo-33053 [https://bugs.python.org/issue?@action=redirect&bpo=33053]: When using the -m switch, sys.path[0] is now explicitly expanded as the *starting* working directory, rather than being left as the empty path (which allows imports from the current working directory at the time of the import)
- bpo-33138 [https://bugs.python.org/issue?@action=redirect&bpo=33138]: Changed standard error message

for non-pickleable and non-copyable types. It now says «cannot pickle» instead of «can't pickle» or «cannot serialize».

- bpo-33018 [https://bugs.python.org/issue?@action=redirect&bpo=33018]: Improve consistency of errors raised by `issubclass()` when called with a non-class and an abstract base class as the first and second arguments, respectively. Patch by Josh Bronson.

- bpo-33083 [https://bugs.python.org/issue?@action=redirect&bpo=33083]: `math.factorial` no longer accepts arguments that are not int-like. Patch by Pablo Galindo.

- bpo-33041 [https://bugs.python.org/issue?@action=redirect&bpo=33041]: Added new opcode **`END_ASYNC_FOR`** and fixes the following issues:

    - Setting global **`StopAsyncIteration`** no longer breaks `async for` loops.
    - Jumping into an `async for` loop is now disabled.
    - Jumping out of an `async for` loop no longer corrupts the stack.

- bpo-25750 [https://bugs.python.org/issue?@action=redirect&bpo=25750]: Fix rare Python crash due to bad refcounting in `type_getattro()` if a descriptor deletes itself from the class. Patch by Jeroen Demeyer.

- bpo-33041 [https://bugs.python.org/issue?@action=redirect&bpo=33041]: Fixed bytecode generation for «async for» with a complex target. A StopAsyncIteration raised on assigning or unpacking will be now propagated instead of stopping the iteration.

- bpo-33026 [https://bugs.python.org/issue?@action=redirect&bpo=33026]: Fixed jumping out of «with» block by setting f_lineno.

- bpo-33005 [https://bugs.python.org/issue?@action=redirect&bpo=33005]: Fix a crash on fork when using a custom memory allocator (ex: using PYTHONMALLOC env var). _PyGILState_Reinit() and _PyInterpreterState_Enable() now use the default RAW memory allocator to allocate a new interpreters mutex on fork.

- bpo-32911 [https://bugs.python.org/issue?@action=redirect&bpo=32911]: Due to unexpected compatibility issues discovered during downstream beta testing, reverted bpo-29463 [https://bugs.python.org/issue?@action=redirect&bpo=29463]. `docstring` field is removed from Module, ClassDef, FunctionDef, and AsyncFunctionDef ast nodes which was added in 3.7a1. Docstring expression is restored as a first statement in their body. Based on patch by Inada Naoki.

- bpo-17288 [https://bugs.python.org/issue?@action=redirect&bpo=17288]: Prevent jumps from “return” and “exception” trace events.

- bpo-32946 [https://bugs.python.org/issue?@action=redirect&bpo=32946]: Importing names from already imported module with «from … import …» is now 30% faster if the module is not a package.

- bpo-32932 [https://bugs.python.org/issue?@action=redirect&bpo=32932]: Make error message more revealing when there are non-str objects in `__all__`.

- bpo-32925 [https://bugs.python.org/issue?@action=redirect&bpo=32925]: Optimized iterating and containing test for literal lists consisting of non-constants: `x in [a, b]` and `for x in [a, b]`. The case of all constant elements already was optimized.

- bpo-32889 [https://bugs.python.org/issue?@action=redirect&bpo=32889]: Update Valgrind suppression list to account for the rename of `Py_ADDRESS_IN_RANG` to `address_in_range`.

- bpo-32836 [https://bugs.python.org/issue?@action=redirect&bpo=32836]: Don’t use temporary variables in cases of list/dict/set comprehensions

- bpo-31356 [https://bugs.python.org/issue?@action=redirect&bpo=31356]: Remove the new API added in bpo-31356 [https://bugs.python.org/issue?@action=redirect&bpo=31356] (gc.ensure_disabled() context manager).
- bpo-32305 [https://bugs.python.org/issue?@action=redirect&bpo=32305]: For namespace packages, ensure that both `__file__` and `__spec__.origin` are set to None.
- bpo-32303 [https://bugs.python.org/issue?@action=redirect&bpo=32303]: Make sure `__spec__.loader` matches `__loader__` for namespace packages.
- bpo-32711 [https://bugs.python.org/issue?@action=redirect&bpo=32711]: Fix the warning messages for Python/ast_unparse.c. Patch by Stéphane Wirtel
- bpo-32583 [https://bugs.python.org/issue?@action=redirect&bpo=32583]: Fix possible crashing in builtin Unicode decoders caused by write out-of-bound errors when using customized decode error handlers.
- bpo-32489 [https://bugs.python.org/issue?@action=redirect&bpo=32489]: A `continue` statement is now allowed in the `finally` clause.
- bpo-17611 [https://bugs.python.org/issue?@action=redirect&bpo=17611]: Simplified the interpreter loop by moving the logic of unrolling the stack of blocks into the compiler. The compiler emits now explicit instructions for adjusting the stack of values and calling the cleaning up code for `break`, `continue` and `return`.

  Removed opcodes **`BREAK_LOOP`**, **`CONTINUE_LOOP`**, **`SETUP_LOOP`** and **`SETUP_EXCEPT`**. Added new opcodes **`ROT_FOUR`**, **`BEGIN_FINALLY`** and **`CALL_FINALLY`** and **`POP_FINALLY`**. Changed the behavior of **`END_FINALLY`** and

`WITH_CLEANUP_START`.

- bpo-32285 [https://bugs.python.org/issue?@action=redirect&bpo=32285]: New function unicodedata.is_normalized, which can check whether a string is in a specific normal form.

- bpo-10544 [https://bugs.python.org/issue?@action=redirect&bpo=10544]: Yield expressions are now disallowed in comprehensions and generator expressions except the expression for the outermost iterable.

- bpo-32117 [https://bugs.python.org/issue?@action=redirect&bpo=32117]: Iterable unpacking is now allowed without parentheses in yield and return statements, e.g. `yield 1, 2, 3, *rest`. Thanks to David Cuthbert for the change and Jordan Chapman for added tests.

- bpo-31902 [https://bugs.python.org/issue?@action=redirect&bpo=31902]: Fix the `col_offset` attribute for ast nodes `ast.AsyncFor`, `ast.AsyncFunctionDef`, and `ast.AsyncWith`. Previously, `col_offset` pointed to the keyword after `async`.

- bpo-25862 [https://bugs.python.org/issue?@action=redirect&bpo=25862]: Fix assertion failures in the `tell()` method of `io.TextIOWrapper`. Patch by Zackery Spytz.

- bpo-21983 [https://bugs.python.org/issue?@action=redirect&bpo=21983]: Fix a crash in `ctypes.cast()` in case the type argument is a ctypes structured data type. Patch by Eryk Sun and Oren Milman.

- bpo-31577 [https://bugs.python.org/issue?@action=redirect&bpo=31577]: Fix a crash in `os.utime()` in case of a bad ns argument. Patch by Oren Milman.

- bpo-29832 [https://bugs.python.org/issue?@action=redirect&bpo=29832]: Remove references to “getsockaddrarg” from various socket error messages. Patch by Oren Milman.

Library

- bpo-35845 [https://bugs.python.org/issue?@action=redirect&bpo=35845]: Add "order" parameter to memoryview.tobytes().

- bpo-35864 [https://bugs.python.org/issue?@action=redirect&bpo=35864]: The _asdict() method for collections.namedtuple now returns a regular dict instead of an OrderedDict.

- bpo-35537 [https://bugs.python.org/issue?@action=redirect&bpo=35537]: An ExitStack is now used internally within subprocess.Popen to clean up pipe file handles. No behavior change in normal operation. But if closing one handle were ever to cause an exception, the others will now be closed instead of leaked. (patch by Giampaolo Rodola)

- bpo-35847 [https://bugs.python.org/issue?@action=redirect&bpo=35847]: RISC-V needed the CTYPES_PASS_BY_REF_HACK. Fixes ctypes Structure test_pass_by_value.

- bpo-35813 [https://bugs.python.org/issue?@action=redirect&bpo=35813]: Shared memory submodule added to multiprocessing to avoid need for serialization between processes

- bpo-35780 [https://bugs.python.org/issue?@action=redirect&bpo=35780]: Fix lru_cache() errors arising in recursive, reentrant, or multi-threaded code. These errors could result in orphan links and in the cache being trapped in a state with fewer than the specified maximum number of links. Fix handling of negative maxsize which should have been treated as zero. Fix errors in toggling the «full» status flag. Fix misordering of links when errors are encountered. Sync-up the C code and pure Python code for the space saving path in functions with a single positional argument. In this common case, the space overhead of an lru cache entry is reduced by almost half. Fix counting of cache misses. In error cases, the miss count was out of sync with the actual number of times the underlying user function was called.

- bpo-35537 [https://bugs.python.org/issue?@action=redirect&bpo=35537]: `os.posix_spawn()` and `os.posix_spawnp()` now have a *setsid* parameter.

- bpo-23846 [https://bugs.python.org/issue?@action=redirect&bpo=23846]: `asyncio.ProactorEventLoop` now catches and logs send errors when the self-pipe is full.

- bpo-34323 [https://bugs.python.org/issue?@action=redirect&bpo=34323]: `asyncio`: Enhance `IocpProactor.close()` log: wait 1 second before the first log, then log every second. Log also the number of seconds since `close()` was called.

- bpo-35674 [https://bugs.python.org/issue?@action=redirect&bpo=35674]: Add a new `os.posix_spawnp()` function. Patch by Joannah Nanjekye.

- bpo-35733 [https://bugs.python.org/issue?@action=redirect&bpo=35733]: `ast.Constant(boolean)` no longer an instance of `ast.Num`. Patch by Anthony Sottile.

- bpo-35726 [https://bugs.python.org/issue?@action=redirect&bpo=35726]: QueueHandler.prepare() now makes a copy of the record before modifying and enqueueing it, to avoid affecting other handlers in the chain.

- bpo-35719 [https://bugs.python.org/issue?@action=redirect&bpo=35719]: Sped up multi-argument `math` functions atan2(), copysign(), remainder() and hypot() by 1.3–2.5 times.

- bpo-35717 [https://bugs.python.org/issue?@action=redirect&bpo=35717]: Fix KeyError exception raised when using enums and compile. Patch contributed by Rémi Lapeyre.

- bpo-35699 [https://bugs.python.org/issue?@action=redirect&bpo=35699]: Fixed detection of Visual Studio Build Tools 2017 in distutils

- bpo-32710 [https://bugs.python.org/issue?@action=redirect&bpo=32710]: Fix memory leaks in asyncio ProactorEventLoop on overlapped operation failure.

- bpo-35702 [https://bugs.python.org/issue?@action=redirect&bpo=35702]: The **`time.CLOCK_UPTIME_RAW`** constant is now available for macOS 10.12.

- bpo-32710 [https://bugs.python.org/issue?@action=redirect&bpo=32710]: Fix a memory leak in asyncio in the ProactorEventLoop when `ReadFile()` or `WSASend()` overlapped operation fail immediately: release the internal buffer.

- bpo-35682 [https://bugs.python.org/issue?@action=redirect&bpo=35682]: Fix `asyncio.ProactorEventLoop.sendfile()`: don't attempt to set the result of an internal future if it's already done.

- bpo-35283 [https://bugs.python.org/issue?@action=redirect&bpo=35283]: Add a deprecated warning for the **`threading.Thread.isAlive()`** method. Patch by Donghee Na.

- bpo-35664 [https://bugs.python.org/issue?@action=redirect&bpo=35664]: Improve operator.itemgetter() performance by 33% with optimized argument handling and with adding a fast path for the common case of a single non-negative integer index into a tuple (which is the typical use case in the standard library).

- bpo-35643 [https://bugs.python.org/issue?@action=redirect&bpo=35643]: Fixed a SyntaxWarning: invalid escape sequence in Modules/_sha3/cleanup.py. Patch by Mickaël Schoentgen.

- bpo-35619 [https://bugs.python.org/issue?@action=redirect&bpo=35619]: Improved support of custom data descriptors in `help()` and `pydoc`.

- bpo-28503 [https://bugs.python.org/issue?@action=redirect&bpo=28503]: The `crypt` module now internally uses the `crypt_r()` library function instead of `crypt()` when available.

- bpo-35614 [https://bugs.python.org/issue?@action=redirect&bpo=35614]: Fixed help() on metaclasses. Patch by Sanyam Khurana.

- bpo-35568 [https://bugs.python.org/issue?@action=redirect&bpo=35568]: Expose `raise(signum)` as `raise_signal`

- bpo-35588 [https://bugs.python.org/issue?@action=redirect&bpo=35588]: The floor division and modulo operations and the `divmod()` function on `fractions.Fraction` types are 2–4x faster. Patch by Stefan Behnel.

- bpo-35585 [https://bugs.python.org/issue?@action=redirect&bpo=35585]: Speed-up building enums by value, e.g. http.HTTPStatus(200).

- bpo-30561 [https://bugs.python.org/issue?@action=redirect&bpo=30561]: random.gammavariate(1.0, beta) now computes the same result as random.expovariate(1.0 / beta). This synchronizes the two algorithms and eliminates some idiosyncrasies in the old implementation. It does however produce a difference stream of random variables than it used to.

- bpo-35537 [https://bugs.python.org/issue?@action=redirect&bpo=35537]: The `subprocess` module can now use the `os.posix_spawn()` function in some cases for better performance.

- bpo-35526 [https://bugs.python.org/issue?

@action=redirect&bpo=35526]: Delaying the “joke” of barry_as_FLUFL.mandatory to Python version 4.0

- bpo-35523 [https://bugs.python.org/issue?@action=redirect&bpo=35523]: Remove `ctypes` callback workaround: no longer create a callback at startup. Avoid SELinux alert on `import ctypes` and `import uuid`.

- bpo-31784 [https://bugs.python.org/issue?@action=redirect&bpo=31784]: `uuid.uuid1()` now calls `time.time_ns()` rather than `int(time.time() * 1e9)`.

- bpo-35513 [https://bugs.python.org/issue?@action=redirect&bpo=35513]: **`TextTestRunner`** of **`unittest.runner`** now uses `time.perf_counter()` rather than `time.time()` to measure the execution time of a test: **`time.time()`** can go backwards, whereas **`time.perf_counter()`** is monotonic.

- bpo-35502 [https://bugs.python.org/issue?@action=redirect&bpo=35502]: Fixed reference leaks in `xml.etree.ElementTree.TreeBuilder` in case of unfinished building of the tree (in particular when an error was raised during parsing XML).

- bpo-35348 [https://bugs.python.org/issue?@action=redirect&bpo=35348]: Make `platform.architecture()` parsing of `file` command output more reliable: add the `-b` option to the `file` command to omit the filename, force the usage of the C locale, and search also the «shared object» pattern.

- bpo-35491 [https://bugs.python.org/issue?@action=redirect&bpo=35491]: `multiprocessing`: Add `Pool.__repr__()` and enhance `BaseProcess.__repr__()` (add pid and parent pid) to ease debugging. Pool state constant values are now strings instead of integers, for example `RUN` value becomes `'RUN'` instead of `0`.

- bpo-35477 [https://bugs.python.org/issue?@action=redirect&bpo=35477]: **`multiprocessing.Pool.__enter__()`** now fails if the pool is not running: `with pool:` fails if used more than once.

- bpo-31446 [https://bugs.python.org/issue?@action=redirect&bpo=31446]: Copy command line that was passed to CreateProcessW since this function can change the content of the input buffer.

- bpo-35471 [https://bugs.python.org/issue?@action=redirect&bpo=35471]: Python 2.4 dropped MacOS 9 support. The macpath module was deprecated in Python 3.7. The module is now removed.

- bpo-23057 [https://bugs.python.org/issue?@action=redirect&bpo=23057]: Unblock Proactor event loop when keyboard interrupt is received on Windows

- bpo-35052 [https://bugs.python.org/issue?@action=redirect&bpo=35052]: Fix xml.dom.minidom cloneNode() on a document with an entity: pass the correct arguments to the user data handler of an entity.

- bpo-20239 [https://bugs.python.org/issue?@action=redirect&bpo=20239]: Allow repeated assignment deletion of **`unittest.mock.Mock`** attributes. Patch by Pablo Galindo.

- bpo-17185 [https://bugs.python.org/issue?@action=redirect&bpo=17185]: Set `__signature__` on mock for **`inspect`** to get signature. Patch by Karthikeyan Singaravelan.

- bpo-35445 [https://bugs.python.org/issue?@action=redirect&bpo=35445]: Memory errors during creating posix.environ no longer ignored.

- bpo-35415 [https://bugs.python.org/issue?@action=redirect&bpo=35415]: Validate fileno= argument to socket.socket().

- bpo-35424 [https://bugs.python.org/issue?@action=redirect&bpo=35424]: **`multiprocessing.Pool`** destructor now emits **`ResourceWarning`** if the pool is still running.

- bpo-35330 [https://bugs.python.org/issue?@action=redirect&bpo=35330]: When a **`Mock`** instance was used to wrap an object, if `side_effect` is used in one of the mocks of it methods, don't call the original implementation and return the result of using the side effect the same way that it is done with return_value.

- bpo-35346 [https://bugs.python.org/issue?@action=redirect&bpo=35346]: Drop Mac OS 9 and Rhapsody support from the **`platform`** module. Rhapsody last release was in 2000. Mac OS 9 last release was in 2001.

- bpo-10496 [https://bugs.python.org/issue?@action=redirect&bpo=10496]: **`check_environ()`** of `distutils.utils` now catches **`KeyError`** on calling **`pwd.getpwuid()`**: don't create the `HOME` environment variable in this case.

- bpo-10496 [https://bugs.python.org/issue?@action=redirect&bpo=10496]: **`posixpath.expanduser()`** now returns the input *path* unchanged if the `HOME` environment variable is not set and the current user has no home directory (if the current user identifier doesn't exist in the password database). This change fix the **`site`** module if the current user doesn't exist in the password database (if the user has no home directory).

- bpo-35389 [https://bugs.python.org/issue?@action=redirect&bpo=35389]: **`platform.libc_ver()`** now uses `os.confstr('CS_GNU_LIBC_VERSION')` if available and the *executable* parameter is not set.

- bpo-35394 [https://bugs.python.org/issue?@action=redirect&bpo=35394]: Add empty slots to asyncio abstract protocols.

- bpo-35310 [https://bugs.python.org/issue?@action=redirect&bpo=35310]: Fix a bug in **`select.select()`** where, in some cases, the file descriptor sequences were returned unmodified after a signal interruption, even though the file descriptors might not be ready yet. **`select.select()`** will now always return empty lists if a timeout has occurred. Patch by Oran Avraham.

- bpo-35380 [https://bugs.python.org/issue?@action=redirect&bpo=35380]: Enable TCP_NODELAY on Windows for proactor asyncio event loop.

- bpo-35341 [https://bugs.python.org/issue?@action=redirect&bpo=35341]: Add generic version of `collections.OrderedDict` to the `typing` module. Patch by Ismo Toijala.

- bpo-35371 [https://bugs.python.org/issue?@action=redirect&bpo=35371]: Fixed possible crash in `os.utime()` on Windows when pass incorrect arguments.

- bpo-35346 [https://bugs.python.org/issue?@action=redirect&bpo=35346]: **`platform.uname()`** now redirects `stderr` to **`os.devnull`** when running external programs like `cmd /c ver`.

- bpo-35066 [https://bugs.python.org/issue?@action=redirect&bpo=35066]: Previously, calling the strftime() method on a datetime object with a trailing "%" in the format string would result in an exception. However, this only occurred when the datetime C module was being used; the python implementation did not match this behavior. Datetime is now PEP-399 compliant, and will not throw an exception on a trailing "%".

- bpo-35345 [https://bugs.python.org/issue?@action=redirect&bpo=35345]: The function `platform.popen` has been removed, it was deprecated since Python 3.3: use **`os.popen()`** instead.

- bpo-35344 [https://bugs.python.org/issue?@action=redirect&bpo=35344]: On macOS, `platform.platform()` now uses `platform.mac_ver()`, if it returns a non-empty release string, to get the macOS version rather than the darwin version.

- bpo-35312 [https://bugs.python.org/issue?@action=redirect&bpo=35312]: Make `lib2to3.pgen2.parse.ParseError` round-trip pickle-able. Patch by Anthony Sottile.

- bpo-35308 [https://bugs.python.org/issue?@action=redirect&bpo=35308]: Fix regression in `webbrowser` where default browsers may be preferred over browsers in the `BROWSER` environment variable.

- bpo-24746 [https://bugs.python.org/issue?@action=redirect&bpo=24746]: Avoid stripping trailing whitespace in doctest fancy diff. Original patch by R. David Murray & Jairo Trad. Enhanced by Sanyam Khurana.

- bpo-28604 [https://bugs.python.org/issue?@action=redirect&bpo=28604]: `locale.localeconv()` now sets temporarily the `LC_CTYPE` locale to the `LC_MONETARY` locale if the two locales are different and monetary strings are non-ASCII. This temporary change affects other threads.

- bpo-35277 [https://bugs.python.org/issue?@action=redirect&bpo=35277]: Update ensurepip to install pip 18.1 and setuptools 40.6.2.

- bpo-24209 [https://bugs.python.org/issue?@action=redirect&bpo=24209]: Adds IPv6 support when invoking http.server directly.

- bpo-35226 [https://bugs.python.org/issue?@action=redirect&bpo=35226]: Recursively check arguments when testing for equality of `unittest.mock.call` objects and add note that tracking of parameters used to create ancestors of mocks in `mock_calls` is not possible.

- bpo-29564 [https://bugs.python.org/issue?@action=redirect&bpo=29564]: The warnings module now suggests to enable tracemalloc if the source is specified, the tracemalloc module is available, but tracemalloc is not tracing memory allocations.

- bpo-35189 [https://bugs.python.org/issue?@action=redirect&bpo=35189]: Modify the following fnctl function to retry if interrupted by a signal (EINTR): flock, lockf, fnctl

- bpo-30064 [https://bugs.python.org/issue?@action=redirect&bpo=30064]: Use add_done_callback() in sock_* asyncio API to unsubscribe reader/writer early on calcellation.

- bpo-35186 [https://bugs.python.org/issue?@action=redirect&bpo=35186]: Removed the «built with» comment added when `setup.py upload` is used with either `bdist_rpm` or `bdist_dumb`.

- bpo-35152 [https://bugs.python.org/issue?@action=redirect&bpo=35152]: Allow sending more than 2 GB at once on a multiprocessing connection on non-Windows systems.

- bpo-35062 [https://bugs.python.org/issue?@action=redirect&bpo=35062]: Fix incorrect parsing of **`io.IncrementalNewlineDecoder`**'s *translate* argument.

- bpo-35065 [https://bugs.python.org/issue?@action=redirect&bpo=35065]: Remove `StreamReaderProtocol._untrack_reader`. The call to `_untrack_reader` is currently performed too soon, causing the protocol to forget about the reader before `connection_lost` can run and feed the EOF to the reader.

- bpo-34160 [https://bugs.python.org/issue?@action=redirect&bpo=34160]: ElementTree and minidom now preserve the attribute order specified by the user.

- bpo-35079 [https://bugs.python.org/issue?@action=redirect&bpo=35079]: Improve difflib.SequenceManager.get_matching_blocks doc by adding "non-overlapping" and changing "!=" to "<".

- bpo-33710 [https://bugs.python.org/issue?@action=redirect&bpo=33710]: Deprecated `l*gettext()` functions and methods in the **`gettext`** module. They return encoded bytes instead of Unicode strings and are artifacts from Python 2 times. Also deprecated functions and methods related to setting the charset for `l*gettext()` functions and methods.

- bpo-35017 [https://bugs.python.org/issue?@action=redirect&bpo=35017]: **`socketserver.BaseServer.serve_forever()`** now exits immediately if it's **`shutdown()`** method is called while it is polling for new events.

- bpo-35024 [https://bugs.python.org/issue?@action=redirect&bpo=35024]: `importlib` no longer logs `wrote <bytecode path>` redundantly after `(created|could not create) <bytecode path>` is already logged. Patch by Quentin Agren.

- bpo-35047 [https://bugs.python.org/issue?@action=redirect&bpo=35047]: `unittest.mock` now includes mock calls in exception messages if `assert_not_called`, `assert_called_once`, or `assert_called_once_with` fails. Patch by Petter Strandmark.

- bpo-31047 [https://bugs.python.org/issue?@action=redirect&bpo=31047]: Fix `ntpath.abspath` regression where it didn't remove a trailing separator on Windows. Patch by Tim Graham.

- bpo-35053 [https://bugs.python.org/issue?@action=redirect&bpo=35053]: tracemalloc now tries to update the traceback when an object is reused from a «free list» (optimization for faster object creation, used by the builtin list type for example).

- bpo-31553 [https://bugs.python.org/issue?@action=redirect&bpo=31553]: Add the –json-lines option to json.tool. Patch by hongweipeng.

- bpo-34794 [https://bugs.python.org/issue?@action=redirect&bpo=34794]: Fixed a leak in Tkinter when pass the Python wrapper around Tcl_Obj back to Tcl/Tk.

- bpo-34909 [https://bugs.python.org/issue?@action=redirect&bpo=34909]: Enum: fix grandchildren subclassing when parent mixed with concrete data types.

- bpo-35022 [https://bugs.python.org/issue?@action=redirect&bpo=35022]: **`unittest.mock.MagicMock`** now supports the `__fspath__` method (from **`os.PathLike`**).

- bpo-35008 [https://bugs.python.org/issue?@action=redirect&bpo=35008]: Fixed references leaks when call the `__setstate__()` method of **`xml.etree.ElementTree.Element`** in the C implementation for already initialized element.

- bpo-23420 [https://bugs.python.org/issue?@action=redirect&bpo=23420]: Verify the value for the parameter "-s" of the cProfile CLI. Patch by Robert Kuska

- bpo-33947 [https://bugs.python.org/issue?@action=redirect&bpo=33947]: dataclasses now handle recursive reprs without raising RecursionError.

- bpo-34890 [https://bugs.python.org/issue?@action=redirect&bpo=34890]: Make **`inspect.iscoroutinefunction()`**, **`inspect.isgeneratorfunction()`** and **`inspect.isasyncgenfunction()`** work with **`functools.partial()`**. Patch by Pablo Galindo.

- bpo-34521 [https://bugs.python.org/issue?@action=redirect&bpo=34521]: Use **`socket.CMSG_SPACE()`** to calculate ancillary data size instead of

`socket.CMSG_LEN()` in **`multiprocessing.reduction.recvfds()`** as RFC 3542 [https://datatracker.ietf.org/doc/html/rfc3542.html] requires the use of the former for portable applications.

- bpo-31522 [https://bugs.python.org/issue?@action=redirect&bpo=31522]: The `mailbox.mbox.get_string` function *from_* parameter can now successfully be set to a non-default value.

- bpo-34970 [https://bugs.python.org/issue?@action=redirect&bpo=34970]: Protect tasks weak set manipulation in `asyncio.all_tasks()`

- bpo-34969 [https://bugs.python.org/issue?@action=redirect&bpo=34969]: gzip: Add –fast, –best on the gzip CLI, these parameters will be used for the fast compression method (quick) or the best method compress (slower, but smaller file). Also, change the default compression level to 6 (tradeoff).

- bpo-16965 [https://bugs.python.org/issue?@action=redirect&bpo=16965]: The 2to3 `execfile` fixer now opens the file with mode `'rb'`. Patch by Zackery Spytz.

- bpo-34966 [https://bugs.python.org/issue?@action=redirect&bpo=34966]: **`pydoc`** now supports aliases not only to methods defined in the end class, but also to inherited methods. The docstring is not duplicated for aliases.

- bpo-34926 [https://bugs.python.org/issue?@action=redirect&bpo=34926]: **`mimetypes.MimeTypes.guess_type()`** now accepts path-like object in addition to url strings. Patch by Mayank Asthana.

- bpo-23831 [https://bugs.python.org/issue?@action=redirect&bpo=23831]: Add `moveto()` method to the `tkinter.Canvas` widget. Patch by Juliette Monsel.

- bpo-34941 [https://bugs.python.org/issue?

@action=redirect&bpo=34941]: Methods `find()`, `findtext()` and `findall()` of the `Element` class in the **`xml.etree.ElementTree`** module are now able to find children which are instances of `Element` subclasses.

- bpo-32680 [https://bugs.python.org/issue?@action=redirect&bpo=32680]: **`smtplib.SMTP`** objects now always have a `sock` attribute present
- bpo-34769 [https://bugs.python.org/issue?@action=redirect&bpo=34769]: Fix for async generators not finalizing when event loop is in debug mode and garbage collector runs in another thread.
- bpo-34936 [https://bugs.python.org/issue?@action=redirect&bpo=34936]: Fix `TclError` in `tkinter.Spinbox.selection_element()`. Patch by Juliette Monsel.
- bpo-34829 [https://bugs.python.org/issue?@action=redirect&bpo=34829]: Add methods `selection_from`, `selection_range`, `selection_present` and `selection_to` to the `tkinter.Spinbox` for consistency with the `tkinter.Entry` widget. Patch by Juliette Monsel.
- bpo-34911 [https://bugs.python.org/issue?@action=redirect&bpo=34911]: Added *secure_protocols* argument to *http.cookiejar.DefaultCookiePolicy* to allow for tweaking of protocols and also to add support by default for *wss*, the secure websocket protocol.
- bpo-34922 [https://bugs.python.org/issue?@action=redirect&bpo=34922]: Fixed integer overflow in the **`digest()`** and **`hexdigest()`** methods for the SHAKE algorithm in the **`hashlib`** module.
- bpo-34925 [https://bugs.python.org/issue?@action=redirect&bpo=34925]: 25% speedup in argument parsing for the functions in the bisect module.
- bpo-34900 [https://bugs.python.org/issue?

@action=redirect&bpo=34900]: Fixed `unittest.TestCase.debug()` when used to call test methods with subtests. Patch by Bruno Oliveira.

- bpo-34844 [https://bugs.python.org/issue?@action=redirect&bpo=34844]: logging.Formatter enhancement - Ensure styles and fmt matches in logging.Formatter - Added validate method in each format style class: StrFormatStyle, PercentStyle, StringTemplateStyle. - This method is called in the constructor of logging.Formatter class - Also re-raise the KeyError in the format method of each style class, so it would a bit clear that it’s an error with the invalid format fields.

- bpo-34897 [https://bugs.python.org/issue?@action=redirect&bpo=34897]: Adjust test.support.missing_compiler_executable check so that a nominal command name of «» is ignored. Patch by Michael Felt.

- bpo-34871 [https://bugs.python.org/issue?@action=redirect&bpo=34871]: Fix inspect module polluted `sys.modules` when parsing `__text_signature__` of callable.

- bpo-34898 [https://bugs.python.org/issue?@action=redirect&bpo=34898]: Add `mtime` argument to `gzip.compress` for reproducible output. Patch by Guo Ci Teo.

- bpo-28441 [https://bugs.python.org/issue?@action=redirect&bpo=28441]: On Cygwin and MinGW, ensure that `sys.executable` always includes the full filename in the path, including the `.exe` suffix (unless it is a symbolic link).

- bpo-34866 [https://bugs.python.org/issue?@action=redirect&bpo=34866]: Adding `max_num_fields` to `cgi.FieldStorage` to make DOS attacks harder by limiting the number of `MiniFieldStorage` objects created by `FieldStorage`.

- bpo-34711 [https://bugs.python.org/issue?@action=redirect&bpo=34711]: http.server ensures it reports HTTPStatus.NOT_FOUND when the local path ends with «/» and is not a directory, even if the underlying OS (e.g. AIX) accepts such paths as a valid file reference. Patch by Michael Felt.

- bpo-34872 [https://bugs.python.org/issue?@action=redirect&bpo=34872]: Fix self-cancellation in C implementation of asyncio.Task

- bpo-34849 [https://bugs.python.org/issue?@action=redirect&bpo=34849]: Don't log waiting for `selector.select` in asyncio loop iteration. The waiting is pretty normal for any asyncio program, logging its time just adds a noise to logs without any useful information provided.

- bpo-34022 [https://bugs.python.org/issue?@action=redirect&bpo=34022]: The **`SOURCE_DATE_EPOCH`** environment variable no longer overrides the value of the *invalidation_mode* argument to **`py_compile.compile()`**, and determines its default value instead.

- bpo-34819 [https://bugs.python.org/issue?@action=redirect&bpo=34819]: Use a monotonic clock to compute timeouts in **`Executor.map()`** and **`as_completed()`**, in order to prevent timeouts from deviating when the system clock is adjusted.

- bpo-34758 [https://bugs.python.org/issue?@action=redirect&bpo=34758]: Add .wasm -> application/wasm to list of recognized file types and content type headers

- bpo-34789 [https://bugs.python.org/issue?@action=redirect&bpo=34789]: **`xml.sax.make_parser()`** now accepts any iterable as its *parser_list* argument. Patch by Andrés Delfino.

- bpo-34334 [https://bugs.python.org/issue?@action=redirect&bpo=34334]: In **`QueueHandler`**, clear `exc_text` from **`LogRecord`** to prevent traceback from being written twice.

- bpo-34687 [https://bugs.python.org/issue?@action=redirect&bpo=34687]: On Windows, asyncio now uses ProactorEventLoop, instead of SelectorEventLoop, by default.

- bpo-5950 [https://bugs.python.org/issue?@action=redirect&bpo=5950]: Support reading zip files with archive comments in **`zipimport`**.

- bpo-32892 [https://bugs.python.org/issue?@action=redirect&bpo=32892]: The parser now represents all constants as **`ast.Constant`** instead of using specific constant AST types (`Num`, `Str`, `Bytes`, `NameConstant` and `Ellipsis`). These classes are considered deprecated and will be removed in future Python versions.

- bpo-34728 [https://bugs.python.org/issue?@action=redirect&bpo=34728]: Add deprecation warning when `loop` is used in methods: `asyncio.sleep`, `asyncio.wait` and `asyncio.wait_for`.

- bpo-34738 [https://bugs.python.org/issue?@action=redirect&bpo=34738]: ZIP files created by `distutils` will now include entries for directories.

- bpo-34659 [https://bugs.python.org/issue?@action=redirect&bpo=34659]: Add an optional *initial* argument to itertools.accumulate().

- bpo-29577 [https://bugs.python.org/issue?@action=redirect&bpo=29577]: Support multiple mixin classes when creating Enums.

- bpo-34670 [https://bugs.python.org/issue?@action=redirect&bpo=34670]: Add SSLContext.post_handshake_auth and SSLSocket.verify_client_post_handshake for TLS 1.3's post handshake authentication feature.

- bpo-32718 [https://bugs.python.org/issue?

@action=redirect&bpo=32718]: The Activate.ps1 script from venv works with PowerShell Core 6.1 and is now available under all operating systems.

- bpo-31177 [https://bugs.python.org/issue?@action=redirect&bpo=31177]: Fix bug that prevented using **`reset_mock`** on mock instances with deleted attributes

- bpo-34672 [https://bugs.python.org/issue?@action=redirect&bpo=34672]: Add a workaround, so the `'Z'` **`time.strftime()`** specifier on the musl C library can work in some cases.

- bpo-34666 [https://bugs.python.org/issue?@action=redirect&bpo=34666]: Implement `asyncio.StreamWriter.awrite` and `asyncio.StreamWriter.aclose()` coroutines. Methods are needed for providing a consistent stream API with control flow switched on by default.

- bpo-6721 [https://bugs.python.org/issue?@action=redirect&bpo=6721]: Acquire the logging module's commonly used internal locks while fork()ing to avoid deadlocks in the child process.

- bpo-34658 [https://bugs.python.org/issue?@action=redirect&bpo=34658]: Fix a rare interpreter unhandled exception state SystemError only seen when using subprocess with a preexec_fn while an after_parent handler has been registered with os.register_at_fork and the fork system call fails.

- bpo-34652 [https://bugs.python.org/issue?@action=redirect&bpo=34652]: Ensure **`os.lchmod()`** is never defined on Linux.

- bpo-34638 [https://bugs.python.org/issue?@action=redirect&bpo=34638]: Store a weak reference to stream reader to break strong references loop between reader and protocol. It allows to detect and close the socket if the stream is deleted (garbage collected) without `close()` call.

- bpo-34536 [https://bugs.python.org/issue?@action=redirect&bpo=34536]: `Enum._missing_`: raise `ValueError` if None returned and `TypeError` if non-member is returned.

- bpo-34636 [https://bugs.python.org/issue?@action=redirect&bpo=34636]: Speed up re scanning of many non-matching characters for s w and d within bytes objects. (microoptimization)

- bpo-24412 [https://bugs.python.org/issue?@action=redirect&bpo=24412]: Add **`addModuleCleanup()`** and **`addClassCleanup()`** to unittest to support cleanups for **`setUpModule()`** and **`setUpClass()`**. Patch by Lisa Roach.

- bpo-34630 [https://bugs.python.org/issue?@action=redirect&bpo=34630]: Don’t log SSL certificate errors in asyncio code (connection error logging is skipped already).

- bpo-32490 [https://bugs.python.org/issue?@action=redirect&bpo=32490]: Prevent filename duplication in **`subprocess`** exception messages. Patch by Zackery Spytz.

- bpo-34363 [https://bugs.python.org/issue?@action=redirect&bpo=34363]: dataclasses.asdict() and .astuple() now handle namedtuples correctly.

- bpo-34625 [https://bugs.python.org/issue?@action=redirect&bpo=34625]: Update vendorized expat library version to 2.2.6.

- bpo-32270 [https://bugs.python.org/issue?@action=redirect&bpo=32270]: The subprocess module no longer mistakenly closes redirected fds even when they were in pass_fds when outside of the default {0, 1, 2} set.

- bpo-34622 [https://bugs.python.org/issue?@action=redirect&bpo=34622]: Create a dedicated `asyncio.CancelledError`, `asyncio.InvalidStateError` and `asyncio.TimeoutError` exception classes. Inherit them from corresponding exceptions from `concurrent.futures` package. Extract `asyncio` exceptions into a separate file.

- bpo-34610 [https://bugs.python.org/issue?@action=redirect&bpo=34610]: Fixed iterator of **`multiprocessing.managers.DictProxy`**.

- bpo-34421 [https://bugs.python.org/issue?@action=redirect&bpo=34421]: Fix distutils logging for non-ASCII strings. This caused installation issues on Windows.

- bpo-34604 [https://bugs.python.org/issue?@action=redirect&bpo=34604]: Fix possible mojibake in the error message of `pwd.getpwnam` and `grp.getgrnam` using string representation because of invisible characters or trailing whitespaces. Patch by William Grzybowski.

- bpo-30977 [https://bugs.python.org/issue?@action=redirect&bpo=30977]: Make uuid.UUID use `__slots__` to reduce its memory footprint. Based on original patch by Wouter Bolsterlee.

- bpo-34574 [https://bugs.python.org/issue?@action=redirect&bpo=34574]: OrderedDict iterators are not exhausted during pickling anymore. Patch by Sergey Fedoseev.

- bpo-8110 [https://bugs.python.org/issue?@action=redirect&bpo=8110]: Refactored **`subprocess`** to check for Windows-specific modules rather than `sys.platform == 'win32'`.

- bpo-34530 [https://bugs.python.org/issue?@action=redirect&bpo=34530]: `distutils.spawn.find_executable()` now falls back on **`os.defpath`** if the `PATH` environment variable is not set.

- bpo-34563 [https://bugs.python.org/issue?@action=redirect&bpo=34563]: On Windows, fix multiprocessing.Connection for very large read: fix _winapi.PeekNamedPipe() and _winapi.ReadFile() for read larger than INT_MAX (usually `2**31-1`).

- bpo-34558 [https://bugs.python.org/issue?@action=redirect&bpo=34558]: Correct typo in Lib/ctypes/_aix.py
- bpo-34282 [https://bugs.python.org/issue?@action=redirect&bpo=34282]: Move `Enum._convert` to `EnumMeta._convert_` and fix enum members getting shadowed by parent attributes.
- bpo-22872 [https://bugs.python.org/issue?@action=redirect&bpo=22872]: When the queue is closed, **`ValueError`** is now raised by **`multiprocessing.Queue.put()`** and **`multiprocessing.Queue.get()`** instead of **`AssertionError`** and **`OSError`**, respectively. Patch by Zackery Spytz.
- bpo-34515 [https://bugs.python.org/issue?@action=redirect&bpo=34515]: Fix parsing non-ASCII identifiers in **`lib2to3.pgen2.tokenize`** (**PEP 3131** [https://peps.python.org/pep-3131/]).
- bpo-13312 [https://bugs.python.org/issue?@action=redirect&bpo=13312]: Avoids a possible integer underflow (undefined behavior) in the time module's year handling code when passed a very low negative year value.
- bpo-34472 [https://bugs.python.org/issue?@action=redirect&bpo=34472]: Improved compatibility for streamed files in **`zipfile`**. Previously an optional signature was not being written and certain ZIP applications were not supported. Patch by Silas Sewell.
- bpo-34454 [https://bugs.python.org/issue?@action=redirect&bpo=34454]: Fix the .fromisoformat() methods of datetime types crashing when given unicode with non-UTF-8-encodable code points. Specifically, datetime.fromisoformat() now accepts surrogate unicode code points used as the separator. Report and tests by Alexey Izbyshev, patch by Paul Ganssle.

- bpo-6700 [https://bugs.python.org/issue?@action=redirect&bpo=6700]: Fix inspect.getsourcelines for module level frames/tracebacks. Patch by Vladimir Matveev.

- bpo-34171 [https://bugs.python.org/issue?@action=redirect&bpo=34171]: Running the `trace` module no longer creates the `trace.cover` file.

- bpo-34441 [https://bugs.python.org/issue?@action=redirect&bpo=34441]: Fix crash when an `ABC`-derived class with invalid `__subclasses__` is passed as the second argument to `issubclass()`. Patch by Alexey Izbyshev.

- bpo-34427 [https://bugs.python.org/issue?@action=redirect&bpo=34427]: Fix infinite loop in `a.extend(a)` for `MutableSequence` subclasses.

- bpo-34412 [https://bugs.python.org/issue?@action=redirect&bpo=34412]: Make `signal.strsignal()` work on HP-UX. Patch by Michael Osipov.

- bpo-20849 [https://bugs.python.org/issue?@action=redirect&bpo=20849]: shutil.copytree now accepts a new `dirs_exist_ok` keyword argument. Patch by Josh Bronson.

- bpo-31715 [https://bugs.python.org/issue?@action=redirect&bpo=31715]: Associate `.mjs` file extension with `application/javascript` MIME Type.

- bpo-34384 [https://bugs.python.org/issue?@action=redirect&bpo=34384]: `os.readlink()` now accepts path-like and `bytes` objects on Windows.

- bpo-22602 [https://bugs.python.org/issue?@action=redirect&bpo=22602]: The UTF-7 decoder now raises `UnicodeDecodeError` for ill-formed sequences starting with «+» (as specified in RFC 2152). Patch by Zackery Spytz.

- bpo-2122 [https://bugs.python.org/issue?@action=redirect&bpo=2122]: The `mmap.flush()` method now returns `None` on success, raises an exception on error under all platforms.

- bpo-34341 [https://bugs.python.org/issue?@action=redirect&bpo=34341]: Appending to the ZIP archive with the ZIP64 extension no longer grows the size of extra fields of existing entries.

- bpo-34333 [https://bugs.python.org/issue?@action=redirect&bpo=34333]: Fix %-formatting in **`pathlib.PurePath.with_suffix()`** when formatting an error message.

- bpo-18540 [https://bugs.python.org/issue?@action=redirect&bpo=18540]: The **`imaplib.IMAP4`** and **`imaplib.IMAP4_SSL`** classes now resolve to the local host IP correctly when the default value of *host* parameter (`''`) is used.

- bpo-26502 [https://bugs.python.org/issue?@action=redirect&bpo=26502]: Implement `traceback.FrameSummary.__len__()` method to preserve compatibility with the old tuple API.

- bpo-34318 [https://bugs.python.org/issue?@action=redirect&bpo=34318]: **`assertRaises()`**, **`assertRaisesRegex()`**, **`assertWarns()`** and **`assertWarnsRegex()`** no longer success if the passed callable is None. They no longer ignore unknown keyword arguments in the context manager mode. A DeprecationWarning was raised in these cases since Python 3.5.

- bpo-9372 [https://bugs.python.org/issue?@action=redirect&bpo=9372]: Deprecate **`__getitem__()`** methods of **`xml.dom.pulldom.DOMEventStream`**, **`wsgiref.util.FileWrapper`** and **`fileinput.FileInput`**.

- bpo-33613 [https://bugs.python.org/issue?@action=redirect&bpo=33613]: Fix a race condition in `multiprocessing.semaphore_tracker` when the tracker receives SIGINT before it can register signal handlers for ignoring it.

- bpo-34248 [https://bugs.python.org/issue?@action=redirect&bpo=34248]: Report filename in the exception raised when the database file cannot be opened by `dbm.gnu.open()` and `dbm.ndbm.open()` due to OS-related error. Patch by Zsolt Cserna.

- bpo-33089 [https://bugs.python.org/issue?@action=redirect&bpo=33089]: Add math.dist() to compute the Euclidean distance between two points.

- bpo-34246 [https://bugs.python.org/issue?@action=redirect&bpo=34246]: `smtplib.SMTP.send_message()` no longer modifies the content of the *mail_options* argument. Patch by Pablo S. Blum de Aguiar.

- bpo-31047 [https://bugs.python.org/issue?@action=redirect&bpo=31047]: Fix `ntpath.abspath` for invalid paths on windows. Patch by Franz Woellert.

- bpo-32321 [https://bugs.python.org/issue?@action=redirect&bpo=32321]: Add pure Python fallback for functools.reduce. Patch by Robert Wright.

- bpo-34270 [https://bugs.python.org/issue?@action=redirect&bpo=34270]: The default asyncio task class now always has a name which can be get or set using two new methods (`get_name()` and `set_name()`) and is visible in the `repr()` output. An initial name can also be set using the new `name` keyword argument to `asyncio.create_task()` or the `create_task()` method of the event loop. If no initial name is set, the default Task implementation generates a name like `Task-1` using a monotonic counter.

- bpo-34263 [https://bugs.python.org/issue?@action=redirect&bpo=34263]: asyncio's event loop will not pass timeouts longer than one day to epoll/select etc.

- bpo-34035 [https://bugs.python.org/issue?@action=redirect&bpo=34035]: Fix several AttributeError in zipfile seek() methods. Patch by Mickaël Schoentgen.

- bpo-32215 [https://bugs.python.org/issue?@action=redirect&bpo=32215]: Fix performance regression in **`sqlite3`** when a DML statement appeared in a different line than the rest of the SQL query.

- bpo-34075 [https://bugs.python.org/issue?@action=redirect&bpo=34075]: Deprecate passing non-ThreadPoolExecutor instances to **`AbstractEventLoop.set_default_executor()`**.

- bpo-34251 [https://bugs.python.org/issue?@action=redirect&bpo=34251]: Restore `msilib.Win64` to preserve backwards compatibility since it's already used by `distutils” bdist_msi` command.

- bpo-19891 [https://bugs.python.org/issue?@action=redirect&bpo=19891]: Ignore errors caused by missing / non-writable homedir while writing history during exit of an interactive session. Patch by Anthony Sottile.

- bpo-33089 [https://bugs.python.org/issue?@action=redirect&bpo=33089]: Enhanced math.hypot() to support more than two dimensions.

- bpo-34228 [https://bugs.python.org/issue?@action=redirect&bpo=34228]: tracemalloc: PYTHONTRACEMALLOC=0 environment variable and -X tracemalloc=0 command line option are now allowed to disable explicitly tracemalloc at startup.

- bpo-13041 [https://bugs.python.org/issue?@action=redirect&bpo=13041]: Use **`shutil.get_terminal_size()`** to calculate the terminal width correctly in the `argparse.HelpFormatter` class. Initial patch by Zbyszek Jędrzejewski-Szmek.

- bpo-34213 [https://bugs.python.org/issue?@action=redirect&bpo=34213]: Allow frozen dataclasses to have a field named «object». Previously this conflicted with an internal use of «object».

- bpo-34052 [https://bugs.python.org/issue?@action=redirect&bpo=34052]: **`sqlite3.Connection.create_aggregate()`**, **`sqlite3.Connection.create_function()`**, **`sqlite3.Connection.set_authorizer()`**, **`sqlite3.Connection.set_progress_handler()`** methods raises TypeError when unhashable objects are passed as callable. These methods now don't pass such objects to SQLite API. Previous behavior could lead to segfaults. Patch by Sergey Fedoseev.

- bpo-34197 [https://bugs.python.org/issue?@action=redirect&bpo=34197]: Attributes *skipinitialspace*, *doublequote* and *strict* of the *dialect* attribute of the **`csv`** reader are now **`bool`** instances instead of integers 0 or 1.

- bpo-32788 [https://bugs.python.org/issue?@action=redirect&bpo=32788]: Errors other than **`TypeError`** raised in methods `__adapt__()` and `__conform__()` in the **`sqlite3`** module are now propagated to the user.

- bpo-21446 [https://bugs.python.org/issue?@action=redirect&bpo=21446]: The `reload` fixer now uses **`importlib.reload()`** instead of deprecated **`imp.reload()`**.

- bpo-940286 [https://bugs.python.org/issue?@action=redirect&bpo=940286]: pydoc's `Helper.showtopic()` method now prints the cross references of a topic correctly.

- bpo-34164 [https://bugs.python.org/issue?@action=redirect&bpo=34164]: **`base64.b32decode()`** could raise UnboundLocalError or OverflowError for incorrect padding. Now it always raises **`base64.Error`** in these cases.

- bpo-33729 [https://bugs.python.org/issue?@action=redirect&bpo=33729]: Fixed issues with arguments parsing in **`hashlib`**.

- bpo-34097 [https://bugs.python.org/issue?@action=redirect&bpo=34097]: ZipFile can zip files older than 1980-01-01 and newer than 2107-12-31 using a new `strict_timestamps` parameter at the cost of setting the timestamp to the limit.

- bpo-34108 [https://bugs.python.org/issue?@action=redirect&bpo=34108]: Remove extraneous CR in 2to3 refactor.

- bpo-34070 [https://bugs.python.org/issue?@action=redirect&bpo=34070]: Make sure to only check if the handle is a tty, when opening a file with `buffering=-1`.

- bpo-27494 [https://bugs.python.org/issue?@action=redirect&bpo=27494]: Reverted bpo-27494 [https://bugs.python.org/issue?@action=redirect&bpo=27494]. 2to3 rejects now a trailing comma in generator expressions.

- bpo-33967 [https://bugs.python.org/issue?@action=redirect&bpo=33967]: functools.singledispatch now raises TypeError instead of IndexError when no positional arguments are passed.

- bpo-34041 [https://bugs.python.org/issue?@action=redirect&bpo=34041]: Add the parameter *deterministic* to the **`sqlite3.Connection.create_function()`** method. Patch by Sergey Fedoseev.

- bpo-34056 [https://bugs.python.org/issue?@action=redirect&bpo=34056]: Ensure the loader shim created by `imp.load_module` always returns bytes from its `get_data()` function. This fixes using `imp.load_module` with **PEP 552** [https://peps.python.org/pep-0552/] hash-based pycs.

- bpo-34054 [https://bugs.python.org/issue?@action=redirect&bpo=34054]: The multiprocessing module now uses the monotonic clock **`time.monotonic()`** instead of the system clock **`time.time()`** to implement timeout.

- bpo-34043 [https://bugs.python.org/issue?

@action=redirect&bpo=34043]: Optimize tarfile uncompress performance about 15% when gzip is used.

- bpo-34044 [https://bugs.python.org/issue?@action=redirect&bpo=34044]: `subprocess.Popen` now copies the *startupinfo* argument to leave it unchanged: it will modify the copy, so that the same `STARTUPINFO` object can be used multiple times.
- bpo-34010 [https://bugs.python.org/issue?@action=redirect&bpo=34010]: Fixed a performance regression for reading streams with tarfile. The buffered read should use a list, instead of appending to a bytes object.
- bpo-34019 [https://bugs.python.org/issue?@action=redirect&bpo=34019]: webbrowser: Correct the arguments passed to Opera Browser when opening a new URL using the `webbrowser` module. Patch by Bumsik Kim.
- bpo-34003 [https://bugs.python.org/issue?@action=redirect&bpo=34003]: csv.DictReader now creates dicts instead of OrderedDicts. Patch by Michael Selik.
- bpo-33978 [https://bugs.python.org/issue?@action=redirect&bpo=33978]: Closed existing logging handlers before reconfiguration via fileConfig and dictConfig. Patch by Karthikeyan Singaravelan.
- bpo-14117 [https://bugs.python.org/issue?@action=redirect&bpo=14117]: Make minor tweaks to turtledemo. The “wikipedia” example is now “rosette”, describing what it draws. The “penrose” print output is reduced. The’1024” output of “tree” is eliminated.
- bpo-33974 [https://bugs.python.org/issue?@action=redirect&bpo=33974]: Fixed passing lists and tuples of strings containing special characters `"`, `\`, `{`, `}` and `\n` as options to **`ttk`** widgets.
- bpo-27500 [https://bugs.python.org/issue?@action=redirect&bpo=27500]: Fix getaddrinfo to resolve IPv6

addresses correctly.

- bpo-24567 [https://bugs.python.org/issue?@action=redirect&bpo=24567]: Improve random.choices() to handle subnormal input weights that could occasionally trigger an IndexError.

- bpo-33871 [https://bugs.python.org/issue?@action=redirect&bpo=33871]: Fixed integer overflow in `os.readv()`, `os.writev()`, `os.preadv()` and `os.pwritev()` and in `os.sendfile()` with *headers* or *trailers* arguments (on BSD-based OSes and macOS).

- bpo-25007 [https://bugs.python.org/issue?@action=redirect&bpo=25007]: Add `copy.copy()` and `copy.deepcopy()` support to zlib compressors and decompressors. Patch by Zackery Spytz.

- bpo-33929 [https://bugs.python.org/issue?@action=redirect&bpo=33929]: multiprocessing: Fix a race condition in Popen of multiprocessing.popen_spawn_win32. The child process now duplicates the read end of pipe instead of «stealing» it. Previously, the read end of pipe was «stolen» by the child process, but it leaked a handle if the child process had been terminated before it could steal the handle from the parent process.

- bpo-33899 [https://bugs.python.org/issue?@action=redirect&bpo=33899]: Tokenize module now implicitly emits a NEWLINE when provided with input that does not have a trailing new line. This behavior now matches what the C tokenizer does internally. Contributed by Ammar Askar.

- bpo-33897 [https://bugs.python.org/issue?@action=redirect&bpo=33897]: Added a “force” keyword argument to logging.basicConfig().

- bpo-33695 [https://bugs.python.org/issue?@action=redirect&bpo=33695]: `shutil.copytree()` uses `os.scandir()` function and all copy functions depending from it use cached `os.stat()` values. The speedup for copying a directory with 8000 files is around +9% on Linux, +20% on Windows and + 30% on a Windows SMB share. Also the number of **`os.stat()`** syscalls is reduced by 38% making **`shutil.copytree()`** especially faster on network filesystems. (Contributed by Giampaolo Rodola" in bpo-33695 [https://bugs.python.org/issue?@action=redirect&bpo=33695].)

- bpo-33916 [https://bugs.python.org/issue?@action=redirect&bpo=33916]: bz2 and lzma: When Decompressor.__init__() is called twice, free the old lock to not leak memory.

- bpo-32568 [https://bugs.python.org/issue?@action=redirect&bpo=32568]: Make select.epoll() and its documentation consistent regarding *sizehint* and *flags*.

- bpo-33833 [https://bugs.python.org/issue?@action=redirect&bpo=33833]: Fixed bug in asyncio where ProactorSocketTransport logs AssertionError if force closed during write.

- bpo-33663 [https://bugs.python.org/issue?@action=redirect&bpo=33663]: Convert content length to string before putting to header.

- bpo-33721 [https://bugs.python.org/issue?@action=redirect&bpo=33721]: **`os.path`** functions that return a boolean result like **`exists()`**, **`lexists()`**, **`isdir()`**, **`isfile()`**, **`islink()`**, and **`ismount()`**, and **`pathlib.Path`** methods that return a boolean result like **`exists()`**, **`is_dir()`**, **`is_file()`**, **`is_mount()`**, **`is_symlink()`**, **`is_block_device()`**, **`is_char_device()`**, **`is_fifo()`**, **`is_socket()`** now return `False` instead of raising **`ValueError`** or its subclasses **`UnicodeEncodeError`** and **`UnicodeDecodeError`** for paths that contain characters or bytes unrepresentable at the OS level.

- bpo-26544 [https://bugs.python.org/issue?@action=redirect&bpo=26544]: Fixed implementation of

`platform.libc_ver()`. It almost always returned version "2.9" for glibc.

- bpo-33843 [https://bugs.python.org/issue?@action=redirect&bpo=33843]: Remove deprecated `cgi.escape`, `cgi.parse_qs` and `cgi.parse_qsl`.
- bpo-33842 [https://bugs.python.org/issue?@action=redirect&bpo=33842]: Remove `tarfile.filemode` which is deprecated since Python 3.3.
- bpo-30167 [https://bugs.python.org/issue?@action=redirect&bpo=30167]: Prevent site.main() exception if PYTHONSTARTUP is set. Patch by Steve Weber.
- bpo-33805 [https://bugs.python.org/issue?@action=redirect&bpo=33805]: Improve error message of dataclasses.replace() when an InitVar is not specified
- bpo-33687 [https://bugs.python.org/issue?@action=redirect&bpo=33687]: Fix the call to `os.chmod()` for `uu.decode()` if a mode is given or decoded. Patch by Timo Furrer.
- bpo-33812 [https://bugs.python.org/issue?@action=redirect&bpo=33812]: Datetime instance d with non-None tzinfo, but with d.tzinfo.utcoffset(d) returning None is now treated as naive by the astimezone() method.
- bpo-32108 [https://bugs.python.org/issue?@action=redirect&bpo=32108]: In configparser, don't clear section when it is assigned to itself.
- bpo-27397 [https://bugs.python.org/issue?@action=redirect&bpo=27397]: Make email module properly handle invalid-length base64 strings.
- bpo-33578 [https://bugs.python.org/issue?@action=redirect&bpo=33578]: Implement multibyte encoder/decoder state methods
- bpo-30805 [https://bugs.python.org/issue?

@action=redirect&bpo=30805]: Avoid race condition with debug logging

- bpo-33476 [https://bugs.python.org/issue?@action=redirect&bpo=33476]: Fix _header_value_parser.py when address group is missing final “;”. Contributed by Enrique Perez-Terron
- bpo-33694 [https://bugs.python.org/issue?@action=redirect&bpo=33694]: asyncio: Fix a race condition causing data loss on pause_reading()/resume_reading() when using the ProactorEventLoop.
- bpo-32493 [https://bugs.python.org/issue?@action=redirect&bpo=32493]: Correct test for `uuid_enc_be` availability in `configure.ac`. Patch by Michael Felt.
- bpo-33792 [https://bugs.python.org/issue?@action=redirect&bpo=33792]: Add asyncio.WindowsSelectorEventLoopPolicy and asyncio.WindowsProactorEventLoopPolicy.
- bpo-33274 [https://bugs.python.org/issue?@action=redirect&bpo=33274]: W3C DOM Level 1 specifies return value of Element.removeAttributeNode() as «The Attr node that was removed.» xml.dom.minidom now complies with this requirement.
- bpo-33778 [https://bugs.python.org/issue?@action=redirect&bpo=33778]: Update `unicodedata`’s database to Unicode version 11.0.0.
- bpo-33165 [https://bugs.python.org/issue?@action=redirect&bpo=33165]: Added a stacklevel parameter to logging calls to allow use of wrapper/helper functions for logging APIs.
- bpo-33770 [https://bugs.python.org/issue?@action=redirect&bpo=33770]: improve base64 exception message for encoded inputs of invalid length

- bpo-33769 [https://bugs.python.org/issue?@action=redirect&bpo=33769]: asyncio/start_tls: Fix error message; cancel callbacks in case of an unhandled error; mark SSLTransport as closed if it is aborted.

- bpo-33767 [https://bugs.python.org/issue?@action=redirect&bpo=33767]: The concatenation (+) and repetition (*) sequence operations now raise **`TypeError`** instead of **`SystemError`** when performed on **`mmap.mmap`** objects. Patch by Zackery Spytz.

- bpo-33734 [https://bugs.python.org/issue?@action=redirect&bpo=33734]: asyncio/ssl: Fix AttributeError, increase default handshake timeout

- bpo-31014 [https://bugs.python.org/issue?@action=redirect&bpo=31014]: Fixed creating a controller for **`webbrowser`** when a user specifies a path to an entry in the BROWSER environment variable. Based on patch by John Still.

- bpo-2504 [https://bugs.python.org/issue?@action=redirect&bpo=2504]: Add gettext.pgettext() and variants.

- bpo-33197 [https://bugs.python.org/issue?@action=redirect&bpo=33197]: Add description property for _ParameterKind

- bpo-32751 [https://bugs.python.org/issue?@action=redirect&bpo=32751]: When cancelling the task due to a timeout, **`asyncio.wait_for()`** will now wait until the cancellation is complete.

- bpo-32684 [https://bugs.python.org/issue?@action=redirect&bpo=32684]: Fix gather to propagate cancellation of itself even with return_exceptions.

- bpo-33654 [https://bugs.python.org/issue?@action=redirect&bpo=33654]: Support protocol type switching in SSLTransport.set_protocol().

- bpo-33674 [https://bugs.python.org/issue?@action=redirect&bpo=33674]: Pause the transport as early as possible to further reduce the risk of data_received() being called before connection_made().

- bpo-33671 [https://bugs.python.org/issue?@action=redirect&bpo=33671]: `shutil.copyfile()`, `shutil.copy()`, `shutil.copy2()`, `shutil.copytree()` and `shutil.move()` use platform-specific fast-copy syscalls on Linux and macOS in order to copy the file more efficiently. On Windows `shutil.copyfile()` uses a bigger default buffer size (1 MiB instead of 16 KiB) and a `memoryview()`-based variant of `shutil.copyfileobj()` is used. The speedup for copying a 512MiB file is about +26% on Linux, +50% on macOS and +40% on Windows. Also, much less CPU cycles are consumed. (Contributed by Giampaolo Rodola" in bpo-25427 [https://bugs.python.org/issue?@action=redirect&bpo=25427].)

- bpo-33674 [https://bugs.python.org/issue?@action=redirect&bpo=33674]: Fix a race condition in SSLProtocol.connection_made() of asyncio.sslproto: start immediately the handshake instead of using call_soon(). Previously, data_received() could be called before the handshake started, causing the handshake to hang or fail.

- bpo-31647 [https://bugs.python.org/issue?@action=redirect&bpo=31647]: Fixed bug where calling write_eof() on a _SelectorSocketTransport after it's already closed raises AttributeError.

- bpo-32610 [https://bugs.python.org/issue?@action=redirect&bpo=32610]: Make asyncio.all_tasks() return only pending tasks.

- bpo-32410 [https://bugs.python.org/issue?@action=redirect&bpo=32410]: Avoid blocking on file IO in sendfile fallback code

- bpo-33469 [https://bugs.python.org/issue?

@action=redirect&bpo=33469]: Fix RuntimeError after closing loop that used run_in_executor

- bpo-33672 [https://bugs.python.org/issue?@action=redirect&bpo=33672]: Fix Task.__repr__ crash with Cython's bogus coroutines
- bpo-33654 [https://bugs.python.org/issue?@action=redirect&bpo=33654]: Fix transport.set_protocol() to support switching between asyncio.Protocol and asyncio.BufferedProtocol. Fix loop.start_tls() to work with asyncio.BufferedProtocols.
- bpo-33652 [https://bugs.python.org/issue?@action=redirect&bpo=33652]: Pickles of type variables and subscripted generics are now future-proof and compatible with older Python versions.
- bpo-32493 [https://bugs.python.org/issue?@action=redirect&bpo=32493]: Fixed **`uuid.uuid1()`** on FreeBSD.
- bpo-33238 [https://bugs.python.org/issue?@action=redirect&bpo=33238]: Add `InvalidStateError` to **`concurrent.futures`**. `Future.set_result` and `Future.set_exception` now raise `InvalidStateError` if the futures are not pending or running. Patch by Jason Haydaman.
- bpo-33618 [https://bugs.python.org/issue?@action=redirect&bpo=33618]: Finalize and document preliminary and experimental TLS 1.3 support with OpenSSL 1.1.1
- bpo-33625 [https://bugs.python.org/issue?@action=redirect&bpo=33625]: Release GIL on `grp.getgrnam`, `grp.getgrgid`, `pwd.getpwnam` and `pwd.getpwuid` if reentrant variants of these functions are available. Patch by William Grzybowski.
- bpo-33623 [https://bugs.python.org/issue?

@action=redirect&bpo=33623]: Fix possible SIGSGV when asyncio.Future is created in _del_

- bpo-11874 [https://bugs.python.org/issue?@action=redirect&bpo=11874]: Use a better regex when breaking usage into wrappable parts. Avoids bogus assertion errors from custom metavar strings.

- bpo-30877 [https://bugs.python.org/issue?@action=redirect&bpo=30877]: Fixed a bug in the Python implementation of the JSON decoder that prevented the cache of parsed strings from clearing after finishing the decoding. Based on patch by c-fos.

- bpo-33604 [https://bugs.python.org/issue?@action=redirect&bpo=33604]: Remove HMAC default to md5 marked for removal in 3.8 (removal originally planned in 3.6, bump to 3.8 in PR 7062).

- bpo-33582 [https://bugs.python.org/issue?@action=redirect&bpo=33582]: Emit a deprecation warning for inspect.formatargspec

- bpo-21145 [https://bugs.python.org/issue?@action=redirect&bpo=21145]: Add `functools.cached_property` decorator, for computed properties cached for the life of the instance.

- bpo-33570 [https://bugs.python.org/issue?@action=redirect&bpo=33570]: Change TLS 1.3 cipher suite settings for compatibility with OpenSSL 1.1.1-pre6 and newer. OpenSSL 1.1.1 will have TLS 1.3 ciphers enabled by default.

- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Do not simplify arguments to `typing.Union`. Now `Union[Manager, Employee]` is not simplified to `Employee` at runtime. Such simplification previously caused several bugs and limited possibilities for introspection.

- bpo-12486 [https://bugs.python.org/issue?@action=redirect&bpo=12486]: `tokenize.generate_tokens()` is now documented as a public API to tokenize unicode strings. It was previously present but undocumented.

- bpo-33540 [https://bugs.python.org/issue?@action=redirect&bpo=33540]: Add a new `block_on_close` class attribute to `ForkingMixIn` and `ThreadingMixIn` classes of `socketserver`.

- bpo-33548 [https://bugs.python.org/issue?@action=redirect&bpo=33548]: tempfile._candidate_tempdir_list should consider common TEMP locations

- bpo-33109 [https://bugs.python.org/issue?@action=redirect&bpo=33109]: argparse subparsers are once again not required by default, reverting the change in behavior introduced by bpo-26510 [https://bugs.python.org/issue?@action=redirect&bpo=26510] in 3.7.0a2.

- bpo-33541 [https://bugs.python.org/issue?@action=redirect&bpo=33541]: Remove unused private method `_strptime.LocaleTime.__pad` (a.k.a. `_LocaleTime__pad`).

- bpo-33536 [https://bugs.python.org/issue?@action=redirect&bpo=33536]: dataclasses.make_dataclass now checks for invalid field names and duplicate fields. Also, added a check for invalid field specifications.

- bpo-33542 [https://bugs.python.org/issue?@action=redirect&bpo=33542]: Prevent `uuid.get_node` from using a DUID instead of a MAC on Windows. Patch by Zvi Effron

- bpo-26819 [https://bugs.python.org/issue?@action=redirect&bpo=26819]: Fix race condition with `ReadTransport.resume_reading` in Windows proactor event loop.

- Fix failure in `typing.get_type_hints()` when ClassVar was provided as a string forward reference.

- bpo-33516 [https://bugs.python.org/issue?@action=redirect&bpo=33516]: **`unittest.mock.MagicMock`** now supports the `__round__` magic method.

- bpo-28612 [https://bugs.python.org/issue?@action=redirect&bpo=28612]: Added support for Site Maps to urllib's `RobotFileParser` as **`RobotFileParser.site_maps()`**. Patch by Lady Red, based on patch by Peter Wirtz.

- bpo-28167 [https://bugs.python.org/issue?@action=redirect&bpo=28167]: Remove platform.linux_distribution, which was deprecated since 3.5.

- bpo-33504 [https://bugs.python.org/issue?@action=redirect&bpo=33504]: Switch the default dictionary implementation for **`configparser`** from **`collections.OrderedDict`** to the standard **`dict`** type.

- bpo-33505 [https://bugs.python.org/issue?@action=redirect&bpo=33505]: Optimize asyncio.ensure_future() by reordering if checks: 1.17x faster.

- bpo-33497 [https://bugs.python.org/issue?@action=redirect&bpo=33497]: Add errors param to cgi.parse_multipart and make an encoding in FieldStorage use the given errors (needed for Twisted). Patch by Amber Brown.

- bpo-29235 [https://bugs.python.org/issue?@action=redirect&bpo=29235]: The **`cProfile.Profile`** class can now be used as a context manager. Patch by Scott Sanderson.

- bpo-33495 [https://bugs.python.org/issue?@action=redirect&bpo=33495]: Change dataclasses.Fields repr to use the repr of each of its members, instead of str. This makes it more clear what each field actually represents. This is especially true for the “type” member.

- bpo-26103 [https://bugs.python.org/issue?@action=redirect&bpo=26103]: Correct `inspect.isdatadescriptor` to look for `__set__` or `__delete__`. Patch by Aaron Hall.

- bpo-29209 [https://bugs.python.org/issue?@action=redirect&bpo=29209]: Removed the `doctype()` method and the *html* parameter of the constructor of **`XMLParser`**. The `doctype()` method defined in a subclass will no longer be called. Deprecated methods `getchildren()` and `getiterator()` in the **`ElementTree`** module emit now a **`DeprecationWarning`** instead of **`PendingDeprecationWarning`**.

- bpo-33453 [https://bugs.python.org/issue?@action=redirect&bpo=33453]: Fix dataclasses to work if using literal string type annotations or if using PEP 563 «Postponed Evaluation of Annotations». Only specific string prefixes are detected for both ClassVar («ClassVar» and «typing.ClassVar») and InitVar («InitVar» and «dataclasses.InitVar»).

- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Minor fixes in typing module: add annotations to `NamedTuple.__new__`, pass `*args` and `**kwds` in `Generic.__new__`. Original PRs by Paulius Šarka and Chad Dombrova.

- bpo-33365 [https://bugs.python.org/issue?@action=redirect&bpo=33365]: Print the header values besides the header keys instead just the header keys if *debuglevel* is set to >0 in **`http.client`**. Patch by Marco Strigl.

- bpo-20087 [https://bugs.python.org/issue?@action=redirect&bpo=20087]: Updated alias mapping with glibc 2.27 supported locales.

- bpo-33422 [https://bugs.python.org/issue?@action=redirect&bpo=33422]: Fix trailing quotation marks getting deleted when looking up byte/string literals on pydoc.

Patch by Andrés Delfino.

- bpo-28167 [https://bugs.python.org/issue?@action=redirect&bpo=28167]: The function `platform.linux_distribution` and `platform.dist` now trigger a `DeprecationWarning` and have been marked for removal in Python 3.8
- bpo-33281 [https://bugs.python.org/issue?@action=redirect&bpo=33281]: Fix ctypes.util.find_library regression on macOS.
- bpo-33311 [https://bugs.python.org/issue?@action=redirect&bpo=33311]: Text and html output generated by cgitb does not display parentheses if the current call is done directly in the module. Patch by Stéphane Blondon.
- bpo-27300 [https://bugs.python.org/issue?@action=redirect&bpo=27300]: The file classes in *tempfile* now accept an *errors* parameter that complements the already existing *encoding*. Patch by Stephan Hohe.
- bpo-32933 [https://bugs.python.org/issue?@action=redirect&bpo=32933]: **`unittest.mock.mock_open()`** now supports iteration over the file contents. Patch by Tony Flury.
- bpo-33217 [https://bugs.python.org/issue?@action=redirect&bpo=33217]: Raise **`TypeError`** when looking up non-Enum objects in Enum classes and Enum members.
- bpo-33197 [https://bugs.python.org/issue?@action=redirect&bpo=33197]: Update error message when constructing invalid inspect.Parameters Patch by Donghee Na.
- bpo-33383 [https://bugs.python.org/issue?@action=redirect&bpo=33383]: Fixed crash in the get() method of the **`dbm.ndbm`** database object when it is called with a single argument.
- bpo-33375 [https://bugs.python.org/issue?

@action=redirect&bpo=33375]: The warnings module now finds the Python file associated with a warning from the code object, rather than the frame's global namespace. This is consistent with how tracebacks and pdb find filenames, and should work better for dynamically executed code.

- bpo-33336 [https://bugs.python.org/issue?@action=redirect&bpo=33336]: `imaplib` now allows `MOVE` command in `IMAP4.uid()` (RFC 6851: IMAP MOVE Extension) and potentially as a name of supported method of `IMAP4` object.
- bpo-32455 [https://bugs.python.org/issue?@action=redirect&bpo=32455]: Added *jump* parameter to **`dis.stack_effect()`**.
- bpo-27485 [https://bugs.python.org/issue?@action=redirect&bpo=27485]: Rename and deprecate undocumented functions in **`urllib.parse()`**.
- bpo-33332 [https://bugs.python.org/issue?@action=redirect&bpo=33332]: Add `signal.valid_signals()` to expose the POSIX sigfillset() functionality.
- bpo-33251 [https://bugs.python.org/issue?@action=redirect&bpo=33251]: `ConfigParser.items()` was fixed so that key-value pairs passed in via **`vars()`** are not included in the resulting output.
- bpo-33329 [https://bugs.python.org/issue?@action=redirect&bpo=33329]: Fix multiprocessing regression on newer glibcs
- bpo-33334 [https://bugs.python.org/issue?@action=redirect&bpo=33334]: **`dis.stack_effect()`** now supports all defined opcodes including NOP and EXTENDED_ARG.
- bpo-991266 [https://bugs.python.org/issue?@action=redirect&bpo=991266]: Fix quoting of the `Comment`

attribute of `http.cookies.SimpleCookie`.

- bpo-33131 [https://bugs.python.org/issue?@action=redirect&bpo=33131]: Upgrade bundled version of pip to 10.0.1.

- bpo-33308 [https://bugs.python.org/issue?@action=redirect&bpo=33308]: Fixed a crash in the **`parser`** module when converting an ST object to a tree of tuples or lists with `line_info=False` and `col_info=True`.

- bpo-23403 [https://bugs.python.org/issue?@action=redirect&bpo=23403]: lib2to3 now uses pickle protocol 4 for pre-computed grammars.

- bpo-33266 [https://bugs.python.org/issue?@action=redirect&bpo=33266]: lib2to3 now recognizes `rf'...'` strings.

- bpo-11594 [https://bugs.python.org/issue?@action=redirect&bpo=11594]: Ensure line-endings are respected when using lib2to3.

- bpo-33254 [https://bugs.python.org/issue?@action=redirect&bpo=33254]: Have `importlib.resources.contents()` and **`importlib.abc.ResourceReader.contents()`** return an iterable instead of an iterator.

- bpo-33265 [https://bugs.python.org/issue?@action=redirect&bpo=33265]: `contextlib.ExitStack` and `contextlib.AsyncExitStack` now use a method instead of a wrapper function for exit callbacks.

- bpo-33263 [https://bugs.python.org/issue?@action=redirect&bpo=33263]: Fix FD leak in `_SelectorSocketTransport` Patch by Vlad Starostin.

- bpo-33256 [https://bugs.python.org/issue?@action=redirect&bpo=33256]: Fix display of `<module>` call in the html produced by `cgitb.html()`. Patch by Stéphane

Blondon.

- bpo-33144 [https://bugs.python.org/issue?@action=redirect&bpo=33144]: `random.Random()` and its subclassing mechanism got optimized to check only once at class/subclass instantiation time whether its `getrandbits()` method can be relied on by other methods, including `randrange()`, for the generation of arbitrarily large random integers. Patch by Wolfgang Maier.
- bpo-33185 [https://bugs.python.org/issue?@action=redirect&bpo=33185]: Fixed regression when running pydoc with the **`-m`** switch. (The regression was introduced in 3.7.0b3 by the resolution of bpo-33053 [https://bugs.python.org/issue?@action=redirect&bpo=33053])

  This fix also changed pydoc to add `os.getcwd()` to **`sys.path`** when necessary, rather than adding `"."`.
- bpo-29613 [https://bugs.python.org/issue?@action=redirect&bpo=29613]: Added support for the `SameSite` cookie flag to the `http.cookies` module.
- bpo-33169 [https://bugs.python.org/issue?@action=redirect&bpo=33169]: Delete entries of `None` in **`sys.path_importer_cache`** when **`importlib.machinery.invalidate_caches()`** is called.
- bpo-33203 [https://bugs.python.org/issue?@action=redirect&bpo=33203]: `random.Random.choice()` now raises `IndexError` for empty sequences consistently even when called from subclasses without a `getrandbits()` implementation.
- bpo-33224 [https://bugs.python.org/issue?@action=redirect&bpo=33224]: Update difflib.mdiff() for **PEP 479** [https://peps.python.org/pep-0479/]. Convert an uncaught StopIteration in a generator into a return-statement.
- bpo-33209 [https://bugs.python.org/issue?@action=redirect&bpo=33209]: End framing at the end of C

implementation of `pickle.Pickler.dump()`.

- bpo-32861 [https://bugs.python.org/issue?@action=redirect&bpo=32861]: The urllib.robotparser's `__str__` representation now includes wildcard entries and the «Crawl-delay» and «Request-rate» fields. Also removes extra newlines that were being appended to the end of the string. Patch by Michael Lazar.

- bpo-23403 [https://bugs.python.org/issue?@action=redirect&bpo=23403]: `DEFAULT_PROTOCOL` in `pickle` was bumped to 4. Protocol 4 is described in **PEP 3154** [https://peps.python.org/pep-3154/] and available since Python 3.4. It offers better performance and smaller size compared to protocol 3 introduced in Python 3.0.

- bpo-20104 [https://bugs.python.org/issue?@action=redirect&bpo=20104]: Improved error handling and fixed a reference leak in `os.posix_spawn()`.

- bpo-33106 [https://bugs.python.org/issue?@action=redirect&bpo=33106]: Deleting a key from a read-only dbm database raises module specific error instead of KeyError.

- bpo-33175 [https://bugs.python.org/issue?@action=redirect&bpo=33175]: In dataclasses, Field._set_name_ now looks up the _set_name_ special method on the class, not the instance, of the default value.

- bpo-32380 [https://bugs.python.org/issue?@action=redirect&bpo=32380]: Create functools.singledispatchmethod to support generic single dispatch on descriptors and methods.

- bpo-33141 [https://bugs.python.org/issue?@action=redirect&bpo=33141]: Have Field objects pass through _set_name_ to their default values, if they have their own _set_name_.

- bpo-33096 [https://bugs.python.org/issue?

@action=redirect&bpo=33096]: Allow ttk.Treeview.insert to insert iid that has a false boolean value. Note iid=0 and iid=False would be same. Patch by Garvit Khatri.

- bpo-32873 [https://bugs.python.org/issue?@action=redirect&bpo=32873]: Treat type variables and special typing forms as immutable by copy and pickle. This fixes several minor issues and inconsistencies, and improves backwards compatibility with Python 3.6.
- bpo-33134 [https://bugs.python.org/issue?@action=redirect&bpo=33134]: When computing dataclass's _hash_, use the lookup table to contain the function which returns the _hash_ value. This is an improvement over looking up a string, and then testing that string to see what to do.
- bpo-33127 [https://bugs.python.org/issue?@action=redirect&bpo=33127]: The ssl module now compiles with LibreSSL 2.7.1.
- bpo-32505 [https://bugs.python.org/issue?@action=redirect&bpo=32505]: Raise TypeError if a member variable of a dataclass is of type Field, but doesn't have a type annotation.
- bpo-33078 [https://bugs.python.org/issue?@action=redirect&bpo=33078]: Fix the failure on OSX caused by the tests relying on sem_getvalue
- bpo-33116 [https://bugs.python.org/issue?@action=redirect&bpo=33116]: Add "Field" to dataclasses._all_.
- bpo-32896 [https://bugs.python.org/issue?@action=redirect&bpo=32896]: Fix an error where subclassing a dataclass with a field that uses a default_factory would generate an incorrect class.
- bpo-33100 [https://bugs.python.org/issue?@action=redirect&bpo=33100]: Dataclasses: If a field has a default value that's a MemberDescriptorType, then it's from that field

being in _slots_, not an actual default value.

- bpo-32953 [https://bugs.python.org/issue?@action=redirect&bpo=32953]: If a non-dataclass inherits from a frozen dataclass, allow attributes to be added to the derived class. Only attributes from the frozen dataclass cannot be assigned to. Require all dataclasses in a hierarchy to be either all frozen or all non-frozen.
- bpo-33097 [https://bugs.python.org/issue?@action=redirect&bpo=33097]: Raise RuntimeError when `executor.submit` is called during interpreter shutdown.
- bpo-32968 [https://bugs.python.org/issue?@action=redirect&bpo=32968]: Modulo and floor division involving Fraction and float should return float.
- bpo-33061 [https://bugs.python.org/issue?@action=redirect&bpo=33061]: Add missing `NoReturn` to `__all__` in typing.py
- bpo-33078 [https://bugs.python.org/issue?@action=redirect&bpo=33078]: Fix the size handling in multiprocessing.Queue when a pickling error occurs.
- bpo-33064 [https://bugs.python.org/issue?@action=redirect&bpo=33064]: lib2to3 now properly supports trailing commas after `*args` and `**kwargs` in function signatures.
- bpo-33056 [https://bugs.python.org/issue?@action=redirect&bpo=33056]: FIX properly close leaking fds in concurrent.futures.ProcessPoolExecutor.
- bpo-33021 [https://bugs.python.org/issue?@action=redirect&bpo=33021]: Release the GIL during fstat() calls, avoiding hang of all threads when calling mmap.mmap(), os.urandom(), and random.seed(). Patch by Nir Soffer.
- bpo-31804 [https://bugs.python.org/issue?

@action=redirect&bpo=31804]: Avoid failing in multiprocessing.Process if the standard streams are closed or None at exit.

- bpo-33034 [https://bugs.python.org/issue?@action=redirect&bpo=33034]: Providing an explicit error message when casting the port property to anything that is not an integer value using `urlparse()` and `urlsplit()`. Patch by Matt Eaton.
- bpo-30249 [https://bugs.python.org/issue?@action=redirect&bpo=30249]: Improve struct.unpack_from() exception messages for problems with the buffer size and offset.
- bpo-33037 [https://bugs.python.org/issue?@action=redirect&bpo=33037]: Skip sending/receiving data after SSL transport closing.
- bpo-27683 [https://bugs.python.org/issue?@action=redirect&bpo=27683]: Fix a regression in **`ipaddress`** that result of **`hosts()`** is empty when the network is constructed by a tuple containing an integer mask and only 1 bit left for addresses.
- bpo-22674 [https://bugs.python.org/issue?@action=redirect&bpo=22674]: Add the strsignal() function in the signal module that returns the system description of the given signal, as returned by strsignal(3).
- bpo-32999 [https://bugs.python.org/issue?@action=redirect&bpo=32999]: Fix C implementation of `ABC.__subclasscheck__(cls, subclass)` crashed when `subclass` is not a type object.
- bpo-33009 [https://bugs.python.org/issue?@action=redirect&bpo=33009]: Fix inspect.signature() for single-parameter partialmethods.
- bpo-32969 [https://bugs.python.org/issue?@action=redirect&bpo=32969]: Expose several missing constants

in zlib and fix corresponding documentation.

- bpo-32056 [https://bugs.python.org/issue?@action=redirect&bpo=32056]: Improved exceptions raised for invalid number of channels and sample width when read an audio file in modules `aifc`, `wave` and `sunau`.
- bpo-32970 [https://bugs.python.org/issue?@action=redirect&bpo=32970]: Improved disassembly of the MAKE_FUNCTION instruction.
- bpo-32844 [https://bugs.python.org/issue?@action=redirect&bpo=32844]: Fix wrong redirection of a low descriptor (0 or 1) to stderr in subprocess if another low descriptor is closed.
- bpo-32960 [https://bugs.python.org/issue?@action=redirect&bpo=32960]: For dataclasses, disallow inheriting frozen from non-frozen classes, and also disallow inheriting non-frozen from frozen classes. This restriction will be relaxed at a future date.
- bpo-32713 [https://bugs.python.org/issue?@action=redirect&bpo=32713]: Fixed tarfile.itn handling of out-of-bounds float values. Patch by Joffrey Fuhrer.
- bpo-32257 [https://bugs.python.org/issue?@action=redirect&bpo=32257]: The ssl module now contains OP_NO_RENEGOTIATION constant, available with OpenSSL 1.1.0h or 1.1.1.
- bpo-32951 [https://bugs.python.org/issue?@action=redirect&bpo=32951]: Direct instantiation of SSLSocket and SSLObject objects is now prohibited. The constructors were never documented, tested, or designed as public constructors. Users were suppose to use ssl.wrap_socket() or SSLContext.
- bpo-32929 [https://bugs.python.org/issue?@action=redirect&bpo=32929]: Remove the tri-state parameter «hash», and add the boolean «unsafe_hash». If unsafe_hash is

True, add a _hash_ function, but if a _hash_ exists, raise TypeError. If unsafe_hash is False, add a _hash_ based on the values of eq= and frozen=. The unsafe_hash=False behavior is the same as the old hash=None behavior. unsafe_hash=False is the default, just as hash=None used to be.

- bpo-32947 [https://bugs.python.org/issue?@action=redirect&bpo=32947]: Add OP_ENABLE_MIDDLEBOX_COMPAT and test workaround for TLSv1.3 for future compatibility with OpenSSL 1.1.1.

- bpo-32146 [https://bugs.python.org/issue?@action=redirect&bpo=32146]: Document the interaction between frozen executables and the spawn and forkserver start methods in multiprocessing.

- bpo-30622 [https://bugs.python.org/issue?@action=redirect&bpo=30622]: The ssl module now detects missing NPN support in LibreSSL.

- bpo-32922 [https://bugs.python.org/issue?@action=redirect&bpo=32922]: dbm.open() now encodes filename with the filesystem encoding rather than default encoding.

- bpo-32759 [https://bugs.python.org/issue?@action=redirect&bpo=32759]: Free unused arenas in multiprocessing.heap.

- bpo-32859 [https://bugs.python.org/issue?@action=redirect&bpo=32859]: In `os.dup2`, don't check every call whether the `dup3` syscall exists or not.

- bpo-32556 [https://bugs.python.org/issue?@action=redirect&bpo=32556]: nt._getfinalpathname, nt._getvolumepathname and nt._getdiskusage now correctly convert from bytes.

- bpo-21060 [https://bugs.python.org/issue?@action=redirect&bpo=21060]: Rewrite confusing message from setup.py upload from «No dist file created in earlier command» to the more helpful «Must create and upload files in one command».

- bpo-32857 [https://bugs.python.org/issue?@action=redirect&bpo=32857]: In **`tkinter`**, `after_cancel(None)` now raises a **`ValueError`** instead of canceling the first scheduled function. Patch by Cheryl Sabella.

- bpo-32852 [https://bugs.python.org/issue?@action=redirect&bpo=32852]: Make sure sys.argv remains as a list when running trace.

- bpo-31333 [https://bugs.python.org/issue?@action=redirect&bpo=31333]: `_abc` module is added. It is a speedup module with C implementations for various functions and methods in `abc`. Creating an ABC subclass and calling `isinstance` or `issubclass` with an ABC subclass are up to 1.5x faster. In addition, this makes Python start-up up to 10% faster.

  Note that the new implementation hides internal registry and caches, previously accessible via private attributes `_abc_registry`, `_abc_cache`, and `_abc_negative_cache`. There are three debugging helper methods that can be used instead `_dump_registry`, `_abc_registry_clear`, and `_abc_caches_clear`.

- bpo-32841 [https://bugs.python.org/issue?@action=redirect&bpo=32841]: Fixed `asyncio.Condition` issue which silently ignored cancellation after notifying and cancelling a conditional lock. Patch by Bar Harel.

- bpo-32819 [https://bugs.python.org/issue?@action=redirect&bpo=32819]: ssl.match_hostname() has been simplified and no longer depends on re and ipaddress module for wildcard and IP addresses. Error reporting for invalid wildcards has been improved.

- bpo-19675 [https://bugs.python.org/issue?@action=redirect&bpo=19675]: `multiprocessing.Pool` no longer leaks processes if its initialization fails.

- bpo-32394 [https://bugs.python.org/issue?@action=redirect&bpo=32394]: socket: Remove TCP_FASTOPEN,TCP_KEEPCNT,TCP_KEEPIDLE,TCP_KEEPINTVL flags on older version Windows during run-time.

- bpo-31787 [https://bugs.python.org/issue?@action=redirect&bpo=31787]: Fixed refleaks of `__init__()` methods in various modules. (Contributed by Oren Milman)

- bpo-30157 [https://bugs.python.org/issue?@action=redirect&bpo=30157]: Fixed guessing quote and delimiter in csv.Sniffer.sniff() when only the last field is quoted. Patch by Jake Davis.

- bpo-30688 [https://bugs.python.org/issue?@action=redirect&bpo=30688]: Added support of `\N{name}` escapes in regular expressions. Based on patch by Jonathan Eunice.

- bpo-32792 [https://bugs.python.org/issue?@action=redirect&bpo=32792]: collections.ChainMap() preserves the order of the underlying mappings.

- bpo-32775 [https://bugs.python.org/issue?@action=redirect&bpo=32775]: **`fnmatch.translate()`** no longer produces patterns which contain set operations. Sets starting with “[” or containing “–”, “&&”, “~~” or “||” will be interpreted differently in regular expressions in future versions. Currently they emit warnings. fnmatch.translate() now avoids producing patterns containing such sets by accident.

- bpo-32622 [https://bugs.python.org/issue?@action=redirect&bpo=32622]: Implement native fast sendfile for Windows proactor event loop.

- bpo-32777 [https://bugs.python.org/issue?@action=redirect&bpo=32777]: Fix a rare but potential pre-exec child process deadlock in subprocess on POSIX systems when marking file descriptors inheritable on exec in the child process. This bug appears to have been introduced in 3.4.

- bpo-32647 [https://bugs.python.org/issue?@action=redirect&bpo=32647]: The ctypes module used to depend on indirect linking for dlopen. The shared extension is now explicitly linked against libdl on platforms with dl.
- bpo-32749 [https://bugs.python.org/issue?@action=redirect&bpo=32749]: A **`dbm.dumb`** database opened with flags “r” is now read-only. **`dbm.dumb.open()`** with flags “r” and “w” no longer creates a database if it does not exist.
- bpo-32741 [https://bugs.python.org/issue?@action=redirect&bpo=32741]: Implement `asyncio.TimerHandle.when()` method.
- bpo-32691 [https://bugs.python.org/issue?@action=redirect&bpo=32691]: Use mod_spec.parent when running modules with pdb
- bpo-32734 [https://bugs.python.org/issue?@action=redirect&bpo=32734]: Fixed `asyncio.Lock()` safety issue which allowed acquiring and locking the same lock multiple times, without it being free. Patch by Bar Harel.
- bpo-32727 [https://bugs.python.org/issue?@action=redirect&bpo=32727]: Do not include name field in SMTP envelope from address. Patch by Stéphane Wirtel
- bpo-31453 [https://bugs.python.org/issue?@action=redirect&bpo=31453]: Add TLSVersion constants and SSLContext.maximum_version / minimum_version attributes. The new API wraps OpenSSL 1.1 https://web.archive.org/web/20180309043602/https://www.openssl.org/docs/man1.1.0/ssl/SSL_CTX_set_min_proto_version.html feature.
- bpo-24334 [https://bugs.python.org/issue?@action=redirect&bpo=24334]: Internal implementation details of ssl module were cleaned up. The SSLSocket has one less layer

of indirection. Owner and session information are now handled by the SSLSocket and SSLObject constructor. Channel binding implementation has been simplified.

- bpo-31848 [https://bugs.python.org/issue?@action=redirect&bpo=31848]: Fix the error handling in Aifc_read.initfp() when the SSND chunk is not found. Patch by Zackery Spytz.

- bpo-32585 [https://bugs.python.org/issue?@action=redirect&bpo=32585]: Add Ttk spinbox widget to **`tkinter.ttk`**. Patch by Alan D Moore.

- bpo-32512 [https://bugs.python.org/issue?@action=redirect&bpo=32512]: **`profile`** CLI accepts `-m module_name` as an alternative to script path.

- bpo-8525 [https://bugs.python.org/issue?@action=redirect&bpo=8525]: help() on a type now displays builtin subclasses. This is intended primarily to help with notification of more specific exception subclasses.

  Patch by Sanyam Khurana.

- bpo-31639 [https://bugs.python.org/issue?@action=redirect&bpo=31639]: http.server now exposes a ThreadingHTTPServer class and uses it when the module is run with `-m` to cope with web browsers pre-opening sockets.

- bpo-29877 [https://bugs.python.org/issue?@action=redirect&bpo=29877]: compileall: import ProcessPoolExecutor only when needed, preventing hangs on low resource platforms

- bpo-32221 [https://bugs.python.org/issue?@action=redirect&bpo=32221]: Various functions returning tuple containing IPv6 addresses now omit `%scope` part since the same information is already encoded in *scopeid* tuple item. Especially this speeds up **`socket.recvfrom()`** when it receives multicast packet since useless resolving of network interface name is omitted.

- bpo-32147 [https://bugs.python.org/issue?@action=redirect&bpo=32147]: `binascii.unhexlify()` is now up to 2 times faster. Patch by Sergey Fedoseev.

- bpo-30693 [https://bugs.python.org/issue?@action=redirect&bpo=30693]: The TarFile class now recurses directories in a reproducible way.

- bpo-30693 [https://bugs.python.org/issue?@action=redirect&bpo=30693]: The ZipFile class now recurses directories in a reproducible way.

- bpo-31680 [https://bugs.python.org/issue?@action=redirect&bpo=31680]: Added `curses.ncurses_version`.

- bpo-31908 [https://bugs.python.org/issue?@action=redirect&bpo=31908]: Fix output of cover files for `trace` module command-line tool. Previously emitted cover files only when `--missing` option was used. Patch by Michael Selik.

- bpo-31608 [https://bugs.python.org/issue?@action=redirect&bpo=31608]: Raise a `TypeError` instead of crashing if a `collections.deque` subclass returns a non-deque from `__new__`. Patch by Oren Milman.

- bpo-31425 [https://bugs.python.org/issue?@action=redirect&bpo=31425]: Add support for sockets of the AF_QIPCRTR address family, supported by the Linux kernel. This is used to communicate with services, such as GPS or radio, running on Qualcomm devices. Patch by Bjorn Andersson.

- bpo-22005 [https://bugs.python.org/issue?@action=redirect&bpo=22005]: Implemented unpickling instances of `datetime`, `date` and `time` pickled by Python 2. `encoding='latin1'` should be used for successful decoding.

- bpo-27645 [https://bugs.python.org/issue?

@action=redirect&bpo=27645]: `sqlite3.Connection` now exposes a `backup` method, if the underlying SQLite library is at version 3.6.11 or higher. Patch by Lele Gaifax.

- bpo-16865 [https://bugs.python.org/issue?@action=redirect&bpo=16865]: Support arrays >=2GiB in `ctypes`. Patch by Segev Finer.

- bpo-31508 [https://bugs.python.org/issue?@action=redirect&bpo=31508]: Removed support of arguments in `tkinter.ttk.Treeview.selection`. It was deprecated in 3.6. Use specialized methods like `selection_set` for changing the selection.

- bpo-29456 [https://bugs.python.org/issue?@action=redirect&bpo=29456]: Fix bugs in hangul normalization: u1176, u11a7 and u11c3

Documentation

- bpo-21257 [https://bugs.python.org/issue?@action=redirect&bpo=21257]: Document `http.client.parse_headers()`.
- bpo-34764 [https://bugs.python.org/issue?@action=redirect&bpo=34764]: Improve example of iter() with 2nd sentinel argument.
- bpo-35564 [https://bugs.python.org/issue?@action=redirect&bpo=35564]: Explicitly set master_doc variable in conf.py for compliance with Sphinx 2.0
- bpo-35511 [https://bugs.python.org/issue?@action=redirect&bpo=35511]: Specified that profile.Profile class doesn’t not support enable or disable methods. Also, elaborated that Profile object as a context manager is only supported in cProfile module.
- bpo-10536 [https://bugs.python.org/issue?@action=redirect&bpo=10536]: Enhance the gettext docs. Patch by Éric Araujo
- bpo-35089 [https://bugs.python.org/issue?@action=redirect&bpo=35089]: Remove mention of `typing.io` and `typing.re`. Their types should be imported from

`typing` directly.

- bpo-35038 [https://bugs.python.org/issue?@action=redirect&bpo=35038]: Fix the documentation about an unexisting `f_restricted` attribute in the frame object. Patch by Stéphane Wirtel
- bpo-35042 [https://bugs.python.org/issue?@action=redirect&bpo=35042]: Replace PEP XYZ by the pep role and allow to use the direct links to the PEPs.
- bpo-35044 [https://bugs.python.org/issue?@action=redirect&bpo=35044]: Fix the documentation with the role `exc` for the appropriated exception. Patch by Stéphane Wirtel
- bpo-35035 [https://bugs.python.org/issue?@action=redirect&bpo=35035]: Rename documentation for **`email.utils`** to `email.utils.rst`.
- bpo-34967 [https://bugs.python.org/issue?@action=redirect&bpo=34967]: Use app.add_object_type() instead of the deprecated Sphinx function app.description_unit()
- bpo-34913 [https://bugs.python.org/issue?@action=redirect&bpo=34913]: Add documentation about the new command line interface of the gzip module.
- bpo-32174 [https://bugs.python.org/issue?@action=redirect&bpo=32174]: chm document displays non-ASCII characters properly on some MBCS Windows systems.
- bpo-11233 [https://bugs.python.org/issue?@action=redirect&bpo=11233]: Create availability directive for documentation. Original patch by Georg Brandl.
- bpo-34790 [https://bugs.python.org/issue?@action=redirect&bpo=34790]: Document how passing coroutines to asyncio.wait() can be confusing.
- bpo-34552 [https://bugs.python.org/issue?@action=redirect&bpo=34552]: Make clear that `==` operator sometimes is equivalent to `is`. The `<`, `<=`, `>` and `>=` operators are only defined where they make sense.
- bpo-28617 [https://bugs.python.org/issue?@action=redirect&bpo=28617]: Fixed info in the stdtypes docs concerning the types that support membership tests.
- bpo-20177 [https://bugs.python.org/issue?@action=redirect&bpo=20177]: Migrate datetime.date.fromtimestamp to Argument Clinic. Patch by

Tim Hoffmann.

- bpo-34065 [https://bugs.python.org/issue?@action=redirect&bpo=34065]: Fix wrongly written basicConfig documentation markup syntax
- bpo-33460 [https://bugs.python.org/issue?@action=redirect&bpo=33460]: replaced ellipsis with correct error codes in tutorial chapter 3.
- bpo-33847 [https://bugs.python.org/issue?@action=redirect&bpo=33847]: Add "@" operator entry to index.
- bpo-33409 [https://bugs.python.org/issue?@action=redirect&bpo=33409]: Clarified the relationship between **PEP 538** [https://peps.python.org/pep-0538/]'s PYTHONCOERCECLOCALE and PEP 540's PYTHONUTF8 mode.
- bpo-33197 [https://bugs.python.org/issue?@action=redirect&bpo=33197]: Add versionadded tag to the documentation of ParameterKind.description
- bpo-17045 [https://bugs.python.org/issue?@action=redirect&bpo=17045]: Improve the C-API doc for PyTypeObject. This includes adding several quick-reference tables and a lot of missing slot/typedef entries. The existing entries were also cleaned up with a slightly more consistent format.
- bpo-33736 [https://bugs.python.org/issue?@action=redirect&bpo=33736]: Improve the documentation of **`asyncio.open_connection()`**, **`asyncio.start_server()`** and their UNIX socket counterparts.
- bpo-23859 [https://bugs.python.org/issue?@action=redirect&bpo=23859]: Document that `asyncio.wait()` does not cancel its futures on timeout.
- bpo-32436 [https://bugs.python.org/issue?@action=redirect&bpo=32436]: Document **PEP 567** [https://peps.python.org/pep-0567/] changes to asyncio.
- bpo-33604 [https://bugs.python.org/issue?@action=redirect&bpo=33604]: Update HMAC md5 default to a DeprecationWarning, bump removal to 3.8.
- bpo-33594 [https://bugs.python.org/issue?@action=redirect&bpo=33594]: Document `getargspec`, `from_function` and `from_builtin` as deprecated in their

respective docstring, and include version since deprecation in DeprecationWarning message.

- bpo-33503 [https://bugs.python.org/issue?@action=redirect&bpo=33503]: Fix broken pypi link
- bpo-33421 [https://bugs.python.org/issue?@action=redirect&bpo=33421]: Add missing documentation for `typing.AsyncContextManager`.
- bpo-33487 [https://bugs.python.org/issue?@action=redirect&bpo=33487]: BZ2file now emit a DeprecationWarning when buffering=None is passed, the deprecation message and documentation also now explicitly state it is deprecated since 3.0.
- bpo-33378 [https://bugs.python.org/issue?@action=redirect&bpo=33378]: Add Korean language switcher for https://docs.python.org/3/
- bpo-33276 [https://bugs.python.org/issue?@action=redirect&bpo=33276]: Clarify that the `__path__` attribute on modules cannot be just any value.
- bpo-33201 [https://bugs.python.org/issue?@action=redirect&bpo=33201]: Modernize documentation for writing C extension types.
- bpo-33195 [https://bugs.python.org/issue?@action=redirect&bpo=33195]: Deprecate `Py_UNICODE` usage in `c-api/arg` document. `Py_UNICODE` related APIs are deprecated since Python 3.3, but it is missed in the document.
- bpo-33126 [https://bugs.python.org/issue?@action=redirect&bpo=33126]: Document PyBuffer_ToContiguous().
- bpo-27212 [https://bugs.python.org/issue?@action=redirect&bpo=27212]: Modify documentation for the **`islice()`** recipe to consume initial values up to the start index.
- bpo-28247 [https://bugs.python.org/issue?@action=redirect&bpo=28247]: Update **`zipapp`** documentation to describe how to make standalone applications.
- bpo-18802 [https://bugs.python.org/issue?@action=redirect&bpo=18802]: Documentation changes for ipaddress. Patch by Jon Foster and Berker Peksag.
- bpo-27428 [https://bugs.python.org/issue?@action=redirect&bpo=27428]: Update documentation to clarify

that `WindowsRegistryFinder` implements `MetaPathFinder`. (Patch by Himanshu Lakhara)

- bpo-28124 [https://bugs.python.org/issue?@action=redirect&bpo=28124]: The ssl module function ssl.wrap_socket() has been de-emphasized and deprecated in favor of the more secure and efficient SSLContext.wrap_socket() method.
- bpo-17232 [https://bugs.python.org/issue?@action=redirect&bpo=17232]: Clarify docs for -O and -OO. Patch by Terry Reedy.
- bpo-32436 [https://bugs.python.org/issue?@action=redirect&bpo=32436]: Add documentation for the contextvars module (PEP 567).
- bpo-32800 [https://bugs.python.org/issue?@action=redirect&bpo=32800]: Update link to w3c doc for xml default namespaces.
- bpo-11015 [https://bugs.python.org/issue?@action=redirect&bpo=11015]: Update **`test.support`** documentation.
- bpo-32613 [https://bugs.python.org/issue?@action=redirect&bpo=32613]: Update the faq/windows.html to use the py command from PEP 397 instead of python.
- bpo-8722 [https://bugs.python.org/issue?@action=redirect&bpo=8722]: Document **`__getattr__()`** behavior when property **`get()`** method raises **`AttributeError`**.
- bpo-32614 [https://bugs.python.org/issue?@action=redirect&bpo=32614]: Modify RE examples in documentation to use raw strings to prevent **`DeprecationWarning`** and add text to REGEX HOWTO to highlight the deprecation.
- bpo-20709 [https://bugs.python.org/issue?@action=redirect&bpo=20709]: Remove the paragraph where we explain that os.utime() does not support a directory as path under Windows. Patch by Jan-Philip Gehrcke
- bpo-32722 [https://bugs.python.org/issue?@action=redirect&bpo=32722]: Remove the bad example in the tutorial of the Generator Expression. Patch by Stéphane Wirtel
- bpo-31972 [https://bugs.python.org/issue?@action=redirect&bpo=31972]: Improve docstrings for

`pathlib.PurePath` subclasses.

- bpo-30607 [https://bugs.python.org/issue?@action=redirect&bpo=30607]: Use the externalized `python-docs-theme` package when building the documentation.
- bpo-8243 [https://bugs.python.org/issue?@action=redirect&bpo=8243]: Add a note about curses.addch and curses.addstr exception behavior when writing outside a window, or pad.
- bpo-32337 [https://bugs.python.org/issue?@action=redirect&bpo=32337]: Update documentation related with `dict` order.
- bpo-25041 [https://bugs.python.org/issue?@action=redirect&bpo=25041]: Document `AF_PACKET` in the **`socket`** module.
- bpo-31432 [https://bugs.python.org/issue?@action=redirect&bpo=31432]: Clarify meaning of CERT_NONE, CERT_OPTIONAL, and CERT_REQUIRED flags for ssl.SSLContext.verify_mode.

Tests

- bpo-35772 [https://bugs.python.org/issue?@action=redirect&bpo=35772]: Fix sparse file tests of test_tarfile on ppc64 with the tmpfs filesystem. Fix the function testing if the filesystem supports sparse files: create a file which contains data and «holes», instead of creating a file which contains no data. tmpfs effective block size is a page size (tmpfs lives in the page cache). RHEL uses 64 KiB pages on aarch64, ppc64, ppc64le, only s390x and x86_64 use 4 KiB pages, whereas the test punch holes of 4 KiB.

- bpo-35045 [https://bugs.python.org/issue?@action=redirect&bpo=35045]: Make ssl tests less strict and also accept TLSv1 as system default. The changes unbreaks test_min_max_version on Fedora 29.

- bpo-32710 [https://bugs.python.org/issue?@action=redirect&bpo=32710]: `test_asyncio/test_sendfile.py` now resets the event loop policy using **`tearDownModule()`** as done in other tests, to prevent a warning when running tests on Windows.

- bpo-33717 [https://bugs.python.org/issue?@action=redirect&bpo=33717]: test.pythoninfo now logs information of all clocks, not only time.time() and time.perf_counter().

- bpo-35488 [https://bugs.python.org/issue?@action=redirect&bpo=35488]: Add a test to pathlib's Path.match() to verify it does not support glob-style ** recursive pattern matching.

- bpo-31731 [https://bugs.python.org/issue?@action=redirect&bpo=31731]: Fix a race condition in `check_interrupted_write()` of test_io: create directly the thread with SIGALRM signal blocked, rather than blocking the signal later from the thread. Previously, it was possible that the thread gets the signal before the signal is blocked.

- bpo-35424 [https://bugs.python.org/issue?@action=redirect&bpo=35424]: Fix test_multiprocessing_main_handling: use **`multiprocessing.Pool`** with a context manager and then explicitly join the pool.

- bpo-35519 [https://bugs.python.org/issue?@action=redirect&bpo=35519]: Rename **`test.bisect`** module to **`test.bisect_cmd`** to avoid conflict with **`bisect`** module when running directly a test like `./python Lib/test/test_xmlrpc.py`.

- bpo-35513 [https://bugs.python.org/issue?@action=redirect&bpo=35513]: Replace **`time.time()`** with **`time.monotonic()`** in tests to measure time delta.

- bpo-34279 [https://bugs.python.org/issue?@action=redirect&bpo=34279]: **`test.support.run_unittest()`** no longer raise **`TestDidNotRun`** if the test result contains skipped tests. The exception is now only raised if no test have been run and no test have been skipped.

- bpo-35412 [https://bugs.python.org/issue?@action=redirect&bpo=35412]: Add testcase to `test_future4`: check unicode literal.

- bpo-26704 [https://bugs.python.org/issue?@action=redirect&bpo=26704]: Added test demonstrating double-patching of an instance method. Patch by Anthony Sottile.

- bpo-33725 [https://bugs.python.org/issue?@action=redirect&bpo=33725]: test_multiprocessing_fork may crash on recent versions of macOS. Until the issue is resolved, skip the test on macOS.

- bpo-35352 [https://bugs.python.org/issue?@action=redirect&bpo=35352]: Modify test_asyncio to use the certificate set from the test directory.

- bpo-35317 [https://bugs.python.org/issue?@action=redirect&bpo=35317]: Fix `mktime()` overflow error in `test_email`: run `test_localtime_daylight_true_dst_true()` and `test_localtime_daylight_false_dst_true()` with a specific timezone.

- bpo-21263 [https://bugs.python.org/issue?@action=redirect&bpo=21263]: After several reports that test_gdb does not work properly on macOS and since gdb is not shipped by default anymore, test_gdb is now skipped on macOS when LLVM Clang has been used to compile Python. Patch by Lysandros Nikolaou

- bpo-34279 [https://bugs.python.org/issue?@action=redirect&bpo=34279]: regrtest issue a warning when no tests have been executed in a particular test file. Also, a new final result state is issued if no test have been executed across all test files. Patch by Pablo Galindo.

- bpo-34962 [https://bugs.python.org/issue?@action=redirect&bpo=34962]: make docstest in Doc now passes., and is enforced in CI

- bpo-23596 [https://bugs.python.org/issue?@action=redirect&bpo=23596]: Use argparse for the command line of the gzip module. Patch by Antony Lee

- bpo-34537 [https://bugs.python.org/issue?@action=redirect&bpo=34537]: Fix `test_gdb.test_strings()` when `LC_ALL=C` and GDB was compiled with Python 3.6 or earlier.

- bpo-34587 [https://bugs.python.org/issue?@action=redirect&bpo=34587]: test_socket: Remove RDSTest.testCongestion(). The test tries to fill the receiver's socket buffer and expects an error. But the RDS protocol doesn't require that. Moreover, the Linux implementation of RDS expects that the producer of the messages reduces its rate, it's not the role of the receiver to trigger an error. The test fails on Fedora 28 by design, so just remove it.

- bpo-34661 [https://bugs.python.org/issue?@action=redirect&bpo=34661]: Fix test_shutil if unzip doesn't support -t.

- bpo-34200 [https://bugs.python.org/issue?@action=redirect&bpo=34200]: Fixed non-deterministic flakiness of test_pkg by not using the scary test.support.module_cleanup() logic to save and restore sys.modules contents between test cases.

- bpo-34569 [https://bugs.python.org/issue?@action=redirect&bpo=34569]: The experimental PEP 554 data channels now correctly pass negative PyLong objects between subinterpreters on 32-bit systems. Patch by Michael Felt.

- bpo-34594 [https://bugs.python.org/issue?@action=redirect&bpo=34594]: Fix usage of hardcoded `errno` values in the tests.

- bpo-34579 [https://bugs.python.org/issue?@action=redirect&bpo=34579]: Fix test_embed for AIX Patch by Michael Felt

- bpo-34542 [https://bugs.python.org/issue?@action=redirect&bpo=34542]: Use 3072 RSA keys and SHA-256 signature for test certs and keys.

- bpo-11193 [https://bugs.python.org/issue?@action=redirect&bpo=11193]: Remove special condition for AIX in `test_subprocess.test_undecodable_env`

- bpo-34347 [https://bugs.python.org/issue?@action=redirect&bpo=34347]: Fix `test_utf8_mode.test_cmd_line` for AIX

- bpo-34490 [https://bugs.python.org/issue?@action=redirect&bpo=34490]: On AIX with AF_UNIX family sockets getsockname() does not provide "sockname", so skip calls to transport.get_extra_info("sockname")

- bpo-34391 [https://bugs.python.org/issue?@action=redirect&bpo=34391]: Fix ftplib test for TLS 1.3 by reading from data socket.

- bpo-11192 [https://bugs.python.org/issue?@action=redirect&bpo=11192]: Fix `test_socket` on AIX 6.1 and later IPv6 zone id supports only supported by `inet_pton6_zone()`. Switch to runtime-based `platform.system()` to establish current platform rather than build-time based `sys.platform()`

- bpo-34399 [https://bugs.python.org/issue?@action=redirect&bpo=34399]: Update all RSA keys and DH params to use at least 2048 bits.

- bpo-34373 [https://bugs.python.org/issue?@action=redirect&bpo=34373]: Fix `test_mktime` and `test_pthread_getcpuclickid` tests for AIX Add range checking for `_PyTime_localtime` for AIX Patch by Michael Felt

- bpo-11191 [https://bugs.python.org/issue?@action=redirect&bpo=11191]: Skip the distutils test "test_search_cpp" when using XLC as compiler patch by aixtools (Michael Felt)

- Improved an error message when mock assert_has_calls fails.

- bpo-33746 [https://bugs.python.org/issue?@action=redirect&bpo=33746]: Fix test_unittest when run in verbose mode.

- bpo-33901 [https://bugs.python.org/issue?@action=redirect&bpo=33901]: Fix test_dbm_gnu on macOS with gdbm 1.15: add a larger value to make sure that the file size changes.

- bpo-33873 [https://bugs.python.org/issue?@action=redirect&bpo=33873]: Fix a bug in `regrtest` that caused an extra test to run if –huntrleaks/-R was used. Exit with error in case that invalid parameters are specified to –huntrleaks/-R (at least one warmup run and one repetition must be used).

- bpo-33562 [https://bugs.python.org/issue?@action=redirect&bpo=33562]: Check that a global asyncio event loop policy is not left behind by any tests.

- bpo-33655 [https://bugs.python.org/issue?@action=redirect&bpo=33655]: Ignore test_posix_fallocate failures on BSD platforms that might be due to running on ZFS.

- bpo-32962 [https://bugs.python.org/issue?@action=redirect&bpo=32962]: Fixed test_gdb when Python is compiled with flags -mcet -fcf-protection -O0.

- bpo-33358 [https://bugs.python.org/issue?@action=redirect&bpo=33358]: Fix `test_embed.test_pre_initialization_sys_options()` when the interpreter is built with `--enable-shared`.

- bpo-32872 [https://bugs.python.org/issue?@action=redirect&bpo=32872]: Avoid regrtest compatibility issue with namespace packages.

- bpo-32517 [https://bugs.python.org/issue?

@action=redirect&bpo=32517]: Fix failing `test_asyncio` on macOS 10.12.2+ due to transport of `KqueueSelector` loop was not being closed.

- bpo-32663 [https://bugs.python.org/issue?@action=redirect&bpo=32663]: Making sure the `SMTPUTF8SimTests` class of tests gets run in `test_smtplib.py`.

- bpo-27643 [https://bugs.python.org/issue?@action=redirect&bpo=27643]: Test_C test case needs «signed short» bitfields, but the IBM XLC compiler (on AIX) does not support this Skip the code and test when AIX and XLC are used

  Applicable to Python2-2.7 and later

- bpo-19417 [https://bugs.python.org/issue?@action=redirect&bpo=19417]: Add test_bdb.py.

- bpo-31809 [https://bugs.python.org/issue?@action=redirect&bpo=31809]: Add tests to verify connection with secp ECDH curves.

Build

- bpo-34691 [https://bugs.python.org/issue?@action=redirect&bpo=34691]: The _contextvars module is now built into the core Python library on Windows.
- bpo-35683 [https://bugs.python.org/issue?@action=redirect&bpo=35683]: Improved Azure Pipelines build steps and now verifying layouts correctly
- bpo-35642 [https://bugs.python.org/issue?@action=redirect&bpo=35642]: Remove asynciomodule.c from pythoncore.vcxproj
- bpo-35550 [https://bugs.python.org/issue?@action=redirect&bpo=35550]: Fix incorrect Solaris #ifdef checks to look for __sun && __SVR4 instead of sun when compiling.
- bpo-35499 [https://bugs.python.org/issue?@action=redirect&bpo=35499]: `make profile-opt` no longer replaces `CFLAGS_NODIST` with `CFLAGS`. It now adds

profile-guided optimization (PGO) flags to `CFLAGS_NODIST`: existing `CFLAGS_NODIST` flags are kept.

- bpo-35257 [https://bugs.python.org/issue?@action=redirect&bpo=35257]: Avoid leaking the linker flags from Link Time Optimizations (LTO) into distutils when compiling C extensions.
- bpo-35351 [https://bugs.python.org/issue?@action=redirect&bpo=35351]: When building Python with clang and LTO, LTO flags are no longer passed into CFLAGS to build third-party C extensions through distutils.
- bpo-35139 [https://bugs.python.org/issue?@action=redirect&bpo=35139]: Fix a compiler error when statically linking `pyexpat` in `Modules/Setup`.
- bpo-35059 [https://bugs.python.org/issue?@action=redirect&bpo=35059]: PCbuild: Set InlineFunctionExpansion to OnlyExplicitInline («/Ob1» option) in pyproject.props in Debug mode to expand functions marked as inline. This change should make Python compiled in Debug mode a little bit faster on Windows.
- bpo-35011 [https://bugs.python.org/issue?@action=redirect&bpo=35011]: Restores the use of pyexpatns.h to isolate our embedded copy of the expat C library so that its symbols do not conflict at link or dynamic loading time with an embedding application or other extension modules with their own version of libexpat.
- bpo-28015 [https://bugs.python.org/issue?@action=redirect&bpo=28015]: Have –with-lto works correctly with clang.
- bpo-34765 [https://bugs.python.org/issue?@action=redirect&bpo=34765]: Update the outdated install-sh file to the latest revision from automake v1.16.1
- bpo-34585 [https://bugs.python.org/issue?@action=redirect&bpo=34585]: Check for floating-point byte order in configure.ac using compilation tests instead of executing code, so that these checks work in cross-compiled builds.
- bpo-34710 [https://bugs.python.org/issue?@action=redirect&bpo=34710]: Fixed SSL module build with OpenSSL & pedantic CFLAGS.
- bpo-34582 [https://bugs.python.org/issue?

@action=redirect&bpo=34582]: Add JUnit XML output for regression tests and update Azure DevOps builds.
- bpo-34081 [https://bugs.python.org/issue?@action=redirect&bpo=34081]: Make Sphinx warnings as errors in the Docs Makefile.
- bpo-34555 [https://bugs.python.org/issue?@action=redirect&bpo=34555]: Fix for case where it was not possible to have both `HAVE_LINUX_VM_SOCKETS_H` and `HAVE_SOCKADDR_ALG` be undefined.
- bpo-33015 [https://bugs.python.org/issue?@action=redirect&bpo=33015]: Fix an undefined behaviour in the pthread implementation of **`PyThread_start_new_thread()`**: add a function wrapper to always return `NULL`.
- bpo-34245 [https://bugs.python.org/issue?@action=redirect&bpo=34245]: The Python shared library is now installed with write permission (mode 0755), which is the standard way of installing such libraries.
- bpo-34121 [https://bugs.python.org/issue?@action=redirect&bpo=34121]: Fix detection of C11 atomic support on clang.
- bpo-32430 [https://bugs.python.org/issue?@action=redirect&bpo=32430]: Rename Modules/Setup.dist to Modules/Setup, and remove the necessity to copy the former manually to the latter when updating the local source tree.
- bpo-30345 [https://bugs.python.org/issue?@action=redirect&bpo=30345]: Add -g to LDFLAGS when compiling with LTO to get debug symbols.
- bpo-5755 [https://bugs.python.org/issue?@action=redirect&bpo=5755]: Move `-Wstrict-prototypes` option to `CFLAGS_NODIST` from `OPT`. This option emitted annoying warnings when building extension modules written in C++.
- bpo-33614 [https://bugs.python.org/issue?@action=redirect&bpo=33614]: Ensures module definition files for the stable ABI on Windows are correctly regenerated.
- bpo-33648 [https://bugs.python.org/issue?@action=redirect&bpo=33648]: The –with-c-locale-warning configuration flag has been removed. It has had no effect for about a year.
- bpo-33522 [https://bugs.python.org/issue?

@action=redirect&bpo=33522]: Enable CI builds on Visual Studio Team Services at https://python.visualstudio.com/cpython
- bpo-33512 [https://bugs.python.org/issue?@action=redirect&bpo=33512]: configure's check for «long double» has been simplified
- bpo-33483 [https://bugs.python.org/issue?@action=redirect&bpo=33483]: C compiler is now correctly detected from the standard environment variables. –without-gcc and –with-icc options have been removed.
- bpo-33394 [https://bugs.python.org/issue?@action=redirect&bpo=33394]: Enable the verbose build for extension modules, when GNU make is passed macros on the command line.
- bpo-33393 [https://bugs.python.org/issue?@action=redirect&bpo=33393]: Update config.guess and config.sub files.
- bpo-33377 [https://bugs.python.org/issue?@action=redirect&bpo=33377]: Add new triplets for mips r6 and riscv variants (used in extension suffixes).
- bpo-32232 [https://bugs.python.org/issue?@action=redirect&bpo=32232]: By default, modules configured in `Modules/Setup` are no longer built with `-DPy_BUILD_CORE`. Instead, modules that specifically need that preprocessor definition include it in their individual entries.
- bpo-33182 [https://bugs.python.org/issue?@action=redirect&bpo=33182]: The embedding tests can once again be built with clang 6.0
- bpo-33163 [https://bugs.python.org/issue?@action=redirect&bpo=33163]: Upgrade pip to 9.0.3 and setuptools to v39.0.1.
- bpo-33012 [https://bugs.python.org/issue?@action=redirect&bpo=33012]: gcc 8 has added a new warning heuristic to detect invalid function casts and a stock python build seems to hit that warning quite often. The most common is the cast of a METH_NOARGS function (that uses just one argument) to a PyCFunction. Fix this by adding a dummy argument to all functions that implement METH_NOARGS.
- bpo-32898 [https://bugs.python.org/issue?

@action=redirect&bpo=32898]: Fix the python debug build when using COUNT_ALLOCS.
- bpo-29442 [https://bugs.python.org/issue?@action=redirect&bpo=29442]: Replace optparse with argparse in setup.py

Windows

- bpo-35890 [https://bugs.python.org/issue?@action=redirect&bpo=35890]: Fix API calling consistency of GetVersionEx and wcstok.
- bpo-32560 [https://bugs.python.org/issue?@action=redirect&bpo=32560]: The `py` launcher now forwards its `STARTUPINFO` structure to child processes.
- bpo-35854 [https://bugs.python.org/issue?@action=redirect&bpo=35854]: Fix EnvBuilder and –symlinks in venv on Windows
- bpo-35811 [https://bugs.python.org/issue?@action=redirect&bpo=35811]: Avoid propagating venv settings when launching via py.exe
- bpo-35797 [https://bugs.python.org/issue?@action=redirect&bpo=35797]: Fix default executable used by the multiprocessing module
- bpo-35758 [https://bugs.python.org/issue?@action=redirect&bpo=35758]: Allow building on ARM with MSVC.
- bpo-29734 [https://bugs.python.org/issue?@action=redirect&bpo=29734]: Fix handle leaks in os.stat on Windows.
- bpo-35596 [https://bugs.python.org/issue?@action=redirect&bpo=35596]: Use unchecked PYCs for the embeddable distro to avoid zipimport restrictions.
- bpo-35596 [https://bugs.python.org/issue?@action=redirect&bpo=35596]: Fix vcruntime140.dll being added to embeddable distro multiple times.
- bpo-35402 [https://bugs.python.org/issue?@action=redirect&bpo=35402]: Update Windows build to use Tcl and Tk 8.6.9
- bpo-35401 [https://bugs.python.org/issue?@action=redirect&bpo=35401]: Updates Windows build to

OpenSSL 1.1.0j

- bpo-34977 [https://bugs.python.org/issue?@action=redirect&bpo=34977]: venv on Windows will now use a python.exe redirector rather than copying the actual binaries from the base environment.
- bpo-34977 [https://bugs.python.org/issue?@action=redirect&bpo=34977]: Adds support for building a Windows App Store package
- bpo-35067 [https://bugs.python.org/issue?@action=redirect&bpo=35067]: Remove _distutils_findvs module and use vswhere.exe instead.
- bpo-32557 [https://bugs.python.org/issue?@action=redirect&bpo=32557]: Allow shutil.disk_usage to take a file path on Windows
- bpo-34770 [https://bugs.python.org/issue?@action=redirect&bpo=34770]: Fix a possible null pointer dereference in pyshellext.cpp.
- bpo-34603 [https://bugs.python.org/issue?@action=redirect&bpo=34603]: Fix returning structs from functions produced by MSVC
- bpo-34581 [https://bugs.python.org/issue?@action=redirect&bpo=34581]: Guard MSVC-specific code in socketmodule.c with `#ifdef _MSC_VER`.
- bpo-34532 [https://bugs.python.org/issue?@action=redirect&bpo=34532]: Fixes exit code of list version arguments for py.exe.
- bpo-34062 [https://bugs.python.org/issue?@action=redirect&bpo=34062]: Fixed the “–list” and “–list-paths” arguments for the py.exe launcher
- bpo-34225 [https://bugs.python.org/issue?@action=redirect&bpo=34225]: Ensure INCLUDE and LIB directories do not end with a backslash.
- bpo-34011 [https://bugs.python.org/issue?@action=redirect&bpo=34011]: A suite of code has been changed which copied across DLLs and init.tcl from the running Python location into a venv being created. These copies are needed only when running from a Python source build, and the copying code is now only run when that is the case, rather than whenever a venv is created.
- bpo-34006 [https://bugs.python.org/issue?

@action=redirect&bpo=34006]: Revert line length limit for Windows help docs. The line-length limit is not needed because the pages appear in a separate app rather than on a browser tab. It can also interact badly with the DPI setting.

- bpo-31546 [https://bugs.python.org/issue?@action=redirect&bpo=31546]: Restore running PyOS_InputHook while waiting for user input at the prompt. The restores integration of interactive GUI windows (such as Matplotlib figures) with the prompt on Windows.
- bpo-30237 [https://bugs.python.org/issue?@action=redirect&bpo=30237]: Output error when ReadConsole is canceled by CancelSynchronousIo instead of crashing.
- bpo-33895 [https://bugs.python.org/issue?@action=redirect&bpo=33895]: GIL is released while calling functions that acquire Windows loader lock.
- bpo-33720 [https://bugs.python.org/issue?@action=redirect&bpo=33720]: Reduces maximum marshal recursion depth on release builds.
- bpo-29097 [https://bugs.python.org/issue?@action=redirect&bpo=29097]: Fix bug where **`datetime.fromtimestamp()`** erroneously throws an `OSError` on Windows for values between 0 and 86400. Patch by Ammar Askar.
- bpo-33316 [https://bugs.python.org/issue?@action=redirect&bpo=33316]: PyThread_release_lock always fails
- bpo-33184 [https://bugs.python.org/issue?@action=redirect&bpo=33184]: Update Windows installer to use OpenSSL 1.1.0h.
- bpo-32890 [https://bugs.python.org/issue?@action=redirect&bpo=32890]: Fix usage of GetLastError() instead of errno in os.execve() and os.truncate().
- bpo-33016 [https://bugs.python.org/issue?@action=redirect&bpo=33016]: Fix potential use of uninitialized memory in nt._getfinalpathname
- bpo-32903 [https://bugs.python.org/issue?@action=redirect&bpo=32903]: Fix a memory leak in os.chdir() on Windows if the current directory is set to a UNC path.
- bpo-32901 [https://bugs.python.org/issue?@action=redirect&bpo=32901]: Update Tcl and Tk versions to 8.6.8

- bpo-31966 [https://bugs.python.org/issue?@action=redirect&bpo=31966]: Fixed WindowsConsoleIO.write() for writing empty data.
- bpo-32409 [https://bugs.python.org/issue?@action=redirect&bpo=32409]: Ensures activate.bat can handle Unicode contents.
- bpo-32457 [https://bugs.python.org/issue?@action=redirect&bpo=32457]: Improves handling of denormalized executable path when launching Python.
- bpo-32370 [https://bugs.python.org/issue?@action=redirect&bpo=32370]: Use the correct encoding for ipconfig output in the uuid module. Patch by Segev Finer.
- bpo-29248 [https://bugs.python.org/issue?@action=redirect&bpo=29248]: Fix **`os.readlink()`** on Windows, which was mistakenly treating the `PrintNameOffset` field of the reparse data buffer as a number of characters instead of bytes. Patch by Craig Holmquist and SSE4.
- bpo-1104 [https://bugs.python.org/issue?@action=redirect&bpo=1104]: Correctly handle string length in `msilib.SummaryInfo.GetProperty()` to prevent it from truncating the last character.

macOS

- bpo-35401 [https://bugs.python.org/issue?@action=redirect&bpo=35401]: Update macOS installer to use OpenSSL 1.1.0j.
- bpo-35025 [https://bugs.python.org/issue?@action=redirect&bpo=35025]: Properly guard the use of the `CLOCK_GETTIME` et al. macros in `timemodule` on macOS.
- bpo-24658 [https://bugs.python.org/issue?@action=redirect&bpo=24658]: On macOS, fix reading from and writing into a file with a size larger than 2 GiB.
- bpo-34405 [https://bugs.python.org/issue?@action=redirect&bpo=34405]: Update to OpenSSL 1.1.0i for macOS installer builds.
- bpo-33635 [https://bugs.python.org/issue?@action=redirect&bpo=33635]: In macOS stat on some file descriptors (/dev/fd/3 f.e) will result in bad file descriptor

OSError. Guard against this exception was added in is_dir, is_file and similar methods. DirEntry.is_dir can also throw this exception so _RecursiveWildcardSelector._iterate_directories was also extended with the same error ignoring pattern.

- bpo-13631 [https://bugs.python.org/issue?@action=redirect&bpo=13631]: The .editrc file in user's home directory is now processed correctly during the readline initialization through editline emulation on macOS.
- bpo-33184 [https://bugs.python.org/issue?@action=redirect&bpo=33184]: Update macOS installer build to use OpenSSL 1.1.0h.
- bpo-32726 [https://bugs.python.org/issue?@action=redirect&bpo=32726]: Build and link with private copy of Tcl/Tk 8.6 for the macOS 10.6+ installer. The 10.9+ installer variant already does this. This means that the Python 3.7 provided by the python.org macOS installers no longer need or use any external versions of Tcl/Tk, either system-provided or user-installed, such as ActiveTcl.
- bpo-32901 [https://bugs.python.org/issue?@action=redirect&bpo=32901]: Update macOS 10.9+ installer to Tcl/Tk 8.6.8.
- bpo-31903 [https://bugs.python.org/issue?@action=redirect&bpo=31903]: In **`_scproxy`**, drop the GIL when calling into `SystemConfiguration` to avoid deadlocks.

IDLE

- bpo-35770 [https://bugs.python.org/issue?@action=redirect&bpo=35770]: IDLE macosx deletes Options => Configure IDLE. It previously deleted Window => Zoom Height by mistake. (Zoom Height is now on the Options menu). On Mac, the settings dialog is accessed via Preferences on the IDLE menu.
- bpo-35769 [https://bugs.python.org/issue?@action=redirect&bpo=35769]: Change IDLE's new file name from "Untitled" to "untitled"
- bpo-35660 [https://bugs.python.org/issue?@action=redirect&bpo=35660]: Fix imports in idlelib.window.
- bpo-35641 [https://bugs.python.org/issue?@action=redirect&bpo=35641]: Proper format `calltip` when

the function has no docstring.

- bpo-33987 [https://bugs.python.org/issue?@action=redirect&bpo=33987]: Use ttk Frame for ttk widgets.
- bpo-34055 [https://bugs.python.org/issue?@action=redirect&bpo=34055]: Fix erroneous “smart” indents and newlines in IDLE Shell.
- bpo-35591 [https://bugs.python.org/issue?@action=redirect&bpo=35591]: Find Selection now works when selection not found.
- bpo-35196 [https://bugs.python.org/issue?@action=redirect&bpo=35196]: Speed up squeezer line counting.
- bpo-35598 [https://bugs.python.org/issue?@action=redirect&bpo=35598]: Update config_key: use PEP 8 names and ttk widgets, make some objects global, and add tests.
- bpo-28097 [https://bugs.python.org/issue?@action=redirect&bpo=28097]: Add Previous/Next History entries to Shell menu.
- bpo-35208 [https://bugs.python.org/issue?@action=redirect&bpo=35208]: Squeezer now properly counts wrapped lines before newlines.
- bpo-35555 [https://bugs.python.org/issue?@action=redirect&bpo=35555]: Gray out Code Context menu entry when it’s not applicable.
- bpo-35521 [https://bugs.python.org/issue?@action=redirect&bpo=35521]: Document the IDLE editor code context feature. Add some internal references within the IDLE doc.
- bpo-22703 [https://bugs.python.org/issue?@action=redirect&bpo=22703]: The Code Context menu label now toggles between Show/Hide Code Context. The Zoom Height menu now toggles between Zoom/Restore Height. Zoom Height has moved from the Window menu to the Options menu.
- bpo-35213 [https://bugs.python.org/issue?@action=redirect&bpo=35213]: Where appropriate, use “macOS” in idlelib.
- bpo-34864 [https://bugs.python.org/issue?@action=redirect&bpo=34864]: On macOS, warn if the system preference «Prefer tabs when opening documents» is set to

«Always».

- bpo-34864 [https://bugs.python.org/issue?@action=redirect&bpo=34864]: Document two IDLE on MacOS issues. The System Preferences Dock «prefer tabs always» setting disables some IDLE features. Menus are a bit different than as described for Windows and Linux.
- bpo-35202 [https://bugs.python.org/issue?@action=redirect&bpo=35202]: Remove unused imports from lib/idlelib
- bpo-33000 [https://bugs.python.org/issue?@action=redirect&bpo=33000]: Document that IDLE's shell has no line limit. A program that runs indefinitely can overfill memory.
- bpo-23220 [https://bugs.python.org/issue?@action=redirect&bpo=23220]: Explain how IDLE's Shell displays output.
- bpo-35099 [https://bugs.python.org/issue?@action=redirect&bpo=35099]: Improve the doc about IDLE running user code. The section is renamed from «IDLE – console differences» is renamed «Running user code». It mostly covers the implications of using custom `sys.stdxxx` objects.
- bpo-35097 [https://bugs.python.org/issue?@action=redirect&bpo=35097]: Add IDLE doc subsection explaining editor windows. Topics include opening, title and status bar, .py* extension, and running.
- bpo-35093 [https://bugs.python.org/issue?@action=redirect&bpo=35093]: Document the IDLE document viewer in the IDLE doc. Add a paragraph in «Help and preferences», «Help sources» subsection.
- bpo-35088 [https://bugs.python.org/issue?@action=redirect&bpo=35088]: Update idlelib.help.copy_string docstring. We now use git and backporting instead of hg and forward merging.
- bpo-35087 [https://bugs.python.org/issue?@action=redirect&bpo=35087]: Update idlelib help files for the current doc build. The main change is the elimination of chapter-section numbers.
- bpo-34548 [https://bugs.python.org/issue?@action=redirect&bpo=34548]: Use configured color theme for

read-only text views.

- bpo-1529353 [https://bugs.python.org/issue?@action=redirect&bpo=1529353]: Enable «squeezing» of long outputs in the shell, to avoid performance degradation and to clean up the history without losing it. Squeezed outputs may be copied, viewed in a separate window, and «unsqueezed».
- bpo-34047 [https://bugs.python.org/issue?@action=redirect&bpo=34047]: Fixed mousewheel scrolling direction on macOS.
- bpo-34275 [https://bugs.python.org/issue?@action=redirect&bpo=34275]: Make IDLE calltips always visible on Mac. Some MacOS-tk combinations need .update_idletasks(). Patch by Kevin Walzer.
- bpo-34120 [https://bugs.python.org/issue?@action=redirect&bpo=34120]: Fix unresponsiveness after closing certain windows and dialogs.
- bpo-33975 [https://bugs.python.org/issue?@action=redirect&bpo=33975]: Avoid small type when running htests. Since part of the purpose of human-viewed tests is to determine that widgets look right, it is important that they look the same for testing as when running IDLE.
- bpo-33905 [https://bugs.python.org/issue?@action=redirect&bpo=33905]: Add test for idlelib.stackview.StackBrowser.
- bpo-33924 [https://bugs.python.org/issue?@action=redirect&bpo=33924]: Change mainmenu.menudefs key "windows" to "window". Every other menudef key is lowercase version of main menu entry.
- bpo-33906 [https://bugs.python.org/issue?@action=redirect&bpo=33906]: Rename idlelib.windows as window Match Window on the main menu and remove last plural module name.
- bpo-33917 [https://bugs.python.org/issue?@action=redirect&bpo=33917]: Fix and document idlelib/idle_test/template.py. The revised file compiles, runs, and tests OK. idle_test/README.txt explains how to use it to create new IDLE test files.
- bpo-33904 [https://bugs.python.org/issue?@action=redirect&bpo=33904]: IDLE: In rstrip, rename class RstripExtension as Rstrip

- bpo-33907 [https://bugs.python.org/issue?@action=redirect&bpo=33907]: For consistency and clarity, rename an IDLE module and classes. Module calltips and its class CallTips are now calltip and Calltip. In module calltip_w, class CallTip is now CalltipWindow.
- bpo-33856 [https://bugs.python.org/issue?@action=redirect&bpo=33856]: Add «help» in the welcome message of IDLE
- bpo-33839 [https://bugs.python.org/issue?@action=redirect&bpo=33839]: IDLE: refactor ToolTip and CallTip and add documentation and tests
- bpo-33855 [https://bugs.python.org/issue?@action=redirect&bpo=33855]: Minimally test all IDLE modules. Add missing files, import module, instantiate classes, and check coverage. Check existing files.
- bpo-33656 [https://bugs.python.org/issue?@action=redirect&bpo=33656]: On Windows, add API call saying that tk scales for DPI. On Windows 8.1+ or 10, with DPI compatibility properties of the Python binary unchanged, and a monitor resolution greater than 96 DPI, this should make text and lines sharper. It should otherwise have no effect.
- bpo-33768 [https://bugs.python.org/issue?@action=redirect&bpo=33768]: Clicking on a context line moves that line to the top of the editor window.
- bpo-33763 [https://bugs.python.org/issue?@action=redirect&bpo=33763]: IDLE: Use read-only text widget for code context instead of label widget.
- bpo-33664 [https://bugs.python.org/issue?@action=redirect&bpo=33664]: Scroll IDLE editor text by lines. Previously, the mouse wheel and scrollbar slider moved text by a fixed number of pixels, resulting in partial lines at the top of the editor box. The change also applies to the shell and grep output windows, but not to read-only text views.
- bpo-33679 [https://bugs.python.org/issue?@action=redirect&bpo=33679]: Enable theme-specific color configuration for Code Context. Use the Highlights tab to see the setting for built-in themes or add settings to custom themes.
- bpo-33642 [https://bugs.python.org/issue?@action=redirect&bpo=33642]: Display up to maxlines non-blank

lines for Code Context. If there is no current context, show a single blank line.

- bpo-33628 [https://bugs.python.org/issue?@action=redirect&bpo=33628]: IDLE: Cleanup codecontext.py and its test.
- bpo-33564 [https://bugs.python.org/issue?@action=redirect&bpo=33564]: IDLE's code context now recognizes async as a block opener.
- bpo-21474 [https://bugs.python.org/issue?@action=redirect&bpo=21474]: Update word/identifier definition from ascii to unicode. In text and entry boxes, this affects selection by double-click, movement left/right by control-left/right, and deletion left/right by control-BACKSPACE/DEL.
- bpo-33204 [https://bugs.python.org/issue?@action=redirect&bpo=33204]: IDLE: consistently color invalid string prefixes. A "u" string prefix cannot be paired with either "r" or "f". Consistently color as much of the prefix, starting at the right, as is valid. Revise and extend colorizer test.
- bpo-32984 [https://bugs.python.org/issue?@action=redirect&bpo=32984]: Set `__file__` while running a startup file. Like Python, IDLE optionally runs one startup file in the Shell window before presenting the first interactive input prompt. For IDLE, `-s` runs a file named in environmental variable **`IDLESTARTUP`** or **`PYTHONSTARTUP`**; `-r file` runs `file`. Python sets `__file__` to the startup file name before running the file and unsets it before the first prompt. IDLE now does the same when run normally, without the `-n` option.
- bpo-32940 [https://bugs.python.org/issue?@action=redirect&bpo=32940]: Simplify and rename StringTranslatePseudoMapping in pyparse.
- bpo-32916 [https://bugs.python.org/issue?@action=redirect&bpo=32916]: Change `str` to `code` in pyparse.
- bpo-32905 [https://bugs.python.org/issue?@action=redirect&bpo=32905]: Remove unused code in pyparse module.
- bpo-32874 [https://bugs.python.org/issue?@action=redirect&bpo=32874]: Add tests for pyparse.

- bpo-32837 [https://bugs.python.org/issue?@action=redirect&bpo=32837]: Using the system and place-dependent default encoding for open() is a bad idea for IDLE’s system and location-independent files.
- bpo-32826 [https://bugs.python.org/issue?@action=redirect&bpo=32826]: Add «encoding=utf-8» to open() in IDLE’s test_help_about. GUI test test_file_buttons() only looks at initial ascii-only lines, but failed on systems where open() defaults to “ascii” because readline() internally reads and decodes far enough ahead to encounter a non-ascii character in CREDITS.txt.
- bpo-32831 [https://bugs.python.org/issue?@action=redirect&bpo=32831]: Add docstrings and tests for codecontext.
- bpo-32765 [https://bugs.python.org/issue?@action=redirect&bpo=32765]: Update configdialog General tab docstring to add new widgets to the widget list.

Tools/Demos

- bpo-35884 [https://bugs.python.org/issue?@action=redirect&bpo=35884]: Add a benchmark script for timing various ways to access variables: `Tools/scripts/var_access_benchmark.py`.
- bpo-34989 [https://bugs.python.org/issue?@action=redirect&bpo=34989]: python-gdb.py now handles errors on computing the line number of a Python frame.
- bpo-20260 [https://bugs.python.org/issue?@action=redirect&bpo=20260]: Argument Clinic now has non-bitwise unsigned int converters.
- bpo-32962 [https://bugs.python.org/issue?@action=redirect&bpo=32962]: python-gdb now catches `UnicodeDecodeError` exceptions when calling `string()`.
- bpo-32962 [https://bugs.python.org/issue?@action=redirect&bpo=32962]: python-gdb now catches ValueError on read_var(): when Python has no debug symbols for example.
- bpo-33189 [https://bugs.python.org/issue?@action=redirect&bpo=33189]: **pygettext.py** now recognizes only literal strings as docstrings and translatable strings, and

rejects bytes literals and f-string expressions.

- bpo-31920 [https://bugs.python.org/issue?@action=redirect&bpo=31920]: Fixed handling directories as arguments in the `pygettext` script. Based on patch by Oleg Krasnikov.
- bpo-29673 [https://bugs.python.org/issue?@action=redirect&bpo=29673]: Fix pystackv and pystack gdbinit macros.
- bpo-25427 [https://bugs.python.org/issue?@action=redirect&bpo=25427]: Remove the pyvenv script in favor of `python3 -m venv` in order to lower confusion as to what Python interpreter a virtual environment will be created for.
- bpo-32885 [https://bugs.python.org/issue?@action=redirect&bpo=32885]: Add an `-n` flag for `Tools/scripts/pathfix.py` to disable automatic backup creation (files with `~` suffix).
- bpo-32222 [https://bugs.python.org/issue?@action=redirect&bpo=32222]: Fix pygettext not extracting docstrings for functions with type annotated arguments. Patch by Toby Harradine.
- bpo-31583 [https://bugs.python.org/issue?@action=redirect&bpo=31583]: Fix 2to3 for using with –add-suffix option but without –output-dir option for relative path to files in current directory.

C API

- bpo-35713 [https://bugs.python.org/issue?@action=redirect&bpo=35713]: The **`PyByteArray_Init()`** and **`PyByteArray_Fini()`** functions have been removed. They did nothing since Python 2.7.4 and Python 3.2.0, were excluded from the limited API (stable ABI), and were not documented.
- bpo-33817 [https://bugs.python.org/issue?@action=redirect&bpo=33817]: Fixed **`_PyBytes_Resize()`** for empty bytes objects.
- bpo-35322 [https://bugs.python.org/issue?@action=redirect&bpo=35322]: Fix memory leak in **`PyUnicode_EncodeLocale()`** and

`PyUnicode_EncodeFSDefault()` on error handling.

- bpo-35059 [https://bugs.python.org/issue?@action=redirect&bpo=35059]: The following C macros have been converted to static inline functions: `Py_INCREF()`, `Py_DECREF()`, `Py_XINCREF()`, `Py_XDECREF()`, `PyObject_INIT()`, `PyObject_INIT_VAR()`.
- bpo-35296 [https://bugs.python.org/issue?@action=redirect&bpo=35296]: `make install` now also installs the internal API: `Include/internal/*.h` header files.
- bpo-35081 [https://bugs.python.org/issue?@action=redirect&bpo=35081]: Internal APIs surrounded by `#ifdef Py_BUILD_CORE` have been moved from `Include/*.h` headers to new header files `Include/internal/pycore_*.h`.
- bpo-35259 [https://bugs.python.org/issue?@action=redirect&bpo=35259]: Conditionally declare `Py_FinalizeEx()` (new in 3.6) based on Py_LIMITED_API. Patch by Arthur Neufeld.
- bpo-35081 [https://bugs.python.org/issue?@action=redirect&bpo=35081]: The `_PyObject_GC_TRACK()` and `_PyObject_GC_UNTRACK()` macros have been removed from the public C API.
- bpo-35134 [https://bugs.python.org/issue?@action=redirect&bpo=35134]: Creation of a new `Include/cpython/` subdirectory.
- bpo-34725 [https://bugs.python.org/issue?@action=redirect&bpo=34725]: Adds _Py_SetProgramFullPath so embedders may override sys.executable
- bpo-34910 [https://bugs.python.org/issue?@action=redirect&bpo=34910]: Ensure that `PyObject_Print()` always returns `-1` on error. Patch by Zackery Spytz.
- bpo-34523 [https://bugs.python.org/issue?@action=redirect&bpo=34523]: Py_DecodeLocale() and Py_EncodeLocale() now use the UTF-8 encoding on Windows if Py_LegacyWindowsFSEncodingFlag is zero.
- bpo-34193 [https://bugs.python.org/issue?@action=redirect&bpo=34193]: Fix pluralization in TypeError messages in getargs.c and typeobject.c: “1 argument” instead of “1 arguments” and “1 element” instead of “1 elements”.
- bpo-34127 [https://bugs.python.org/issue?

@action=redirect&bpo=34127]: Return grammatically correct error message based on argument count. Patch by Karthikeyan Singaravelan.

- bpo-23927 [https://bugs.python.org/issue?@action=redirect&bpo=23927]: Fixed `SystemError` in `PyArg_ParseTupleAndKeywords()` when the `w*` format unit is used for optional parameter.
- bpo-32455 [https://bugs.python.org/issue?@action=redirect&bpo=32455]: Added `PyCompile_OpcodeStackEffectWithJump()`.
- bpo-34008 [https://bugs.python.org/issue?@action=redirect&bpo=34008]: Py_Main() can again be called after Py_Initialize(), as in Python 3.6.
- bpo-32500 [https://bugs.python.org/issue?@action=redirect&bpo=32500]: Fixed error messages for `PySequence_Size()`, `PySequence_GetItem()`, `PySequence_SetItem()` and `PySequence_DelItem()` called with a mapping and `PyMapping_Size()` called with a sequence.
- bpo-33818 [https://bugs.python.org/issue?@action=redirect&bpo=33818]: `PyExceptionClass_Name()` will now return `const char *` instead of `char *`.
- bpo-33042 [https://bugs.python.org/issue?@action=redirect&bpo=33042]: Embedding applications may once again call PySys_ResetWarnOptions, PySys_AddWarnOption, and PySys_AddXOption prior to calling Py_Initialize.
- bpo-32374 [https://bugs.python.org/issue?@action=redirect&bpo=32374]: Document that m_traverse for multi-phase initialized modules can be called with m_state=NULL, and add a sanity check
- bpo-30863 [https://bugs.python.org/issue?@action=redirect&bpo=30863]: `PyUnicode_AsWideChar()` and `PyUnicode_AsWideCharString()` no longer cache the `wchar_t*` representation of string objects.

Python 3.7.0 final

Release date: 2018-06-27

Library

- bpo-33851 [https://bugs.python.org/issue?@action=redirect&bpo=33851]: Fix `ast.get_docstring()` for a node that lacks a docstring.

C API

- bpo-33932 [https://bugs.python.org/issue?@action=redirect&bpo=33932]: Calling Py_Initialize() twice does nothing, instead of failing with a fatal error: restore the Python 3.6 behaviour.

Python 3.7.0 release candidate 1

Release date: 2018-06-12

Core and Builtins

- bpo-33803 [https://bugs.python.org/issue?@action=redirect&bpo=33803]: Fix a crash in hamt.c caused by enabling GC tracking for an object that hadn't all of its fields set to NULL.
- bpo-33706 [https://bugs.python.org/issue?@action=redirect&bpo=33706]: Fix a crash in Python initialization when parsing the command line options. Thanks Christoph Gohlke for the bug report and the fix!
- bpo-30654 [https://bugs.python.org/issue?@action=redirect&bpo=30654]: Fixed reset of the SIGINT handler to SIG_DFL on interpreter shutdown even when there was a custom handler set previously. Patch by Philipp Kerling.
- bpo-31849 [https://bugs.python.org/issue?@action=redirect&bpo=31849]: Fix signed/unsigned comparison warning in pyhash.c.

Library

- bpo-30167 [https://bugs.python.org/issue?@action=redirect&bpo=30167]: Prevent site.main() exception if

PYTHONSTARTUP is set. Patch by Steve Weber.

- bpo-33812 [https://bugs.python.org/issue?@action=redirect&bpo=33812]: Datetime instance d with non-None tzinfo, but with d.tzinfo.utcoffset(d) returning None is now treated as naive by the astimezone() method.
- bpo-30805 [https://bugs.python.org/issue?@action=redirect&bpo=30805]: Avoid race condition with debug logging
- bpo-33694 [https://bugs.python.org/issue?@action=redirect&bpo=33694]: asyncio: Fix a race condition causing data loss on pause_reading()/resume_reading() when using the ProactorEventLoop.
- bpo-32493 [https://bugs.python.org/issue?@action=redirect&bpo=32493]: Correct test for `uuid_enc_be` availability in `configure.ac`. Patch by Michael Felt.
- bpo-33792 [https://bugs.python.org/issue?@action=redirect&bpo=33792]: Add asyncio.WindowsSelectorEventLoopPolicy and asyncio.WindowsProactorEventLoopPolicy.
- bpo-33778 [https://bugs.python.org/issue?@action=redirect&bpo=33778]: Update `unicodedata`’s database to Unicode version 11.0.0.
- bpo-33770 [https://bugs.python.org/issue?@action=redirect&bpo=33770]: improve base64 exception message for encoded inputs of invalid length
- bpo-33769 [https://bugs.python.org/issue?@action=redirect&bpo=33769]: asyncio/start_tls: Fix error message; cancel callbacks in case of an unhandled error; mark SSLTransport as closed if it is aborted.
- bpo-33767 [https://bugs.python.org/issue?@action=redirect&bpo=33767]: The concatenation (`+`) and repetition (`*`) sequence operations now raise **`TypeError`** instead of **`SystemError`** when performed on **`mmap.mmap`** objects. Patch by Zackery Spytz.
- bpo-33734 [https://bugs.python.org/issue?@action=redirect&bpo=33734]: asyncio/ssl: Fix AttributeError, increase default handshake timeout
- bpo-11874 [https://bugs.python.org/issue?@action=redirect&bpo=11874]: Use a better regex when breaking usage into wrappable parts. Avoids bogus assertion errors

from custom metavar strings.

- bpo-33582 [https://bugs.python.org/issue?@action=redirect&bpo=33582]: Emit a deprecation warning for inspect.formatargspec

Documentation

- bpo-33409 [https://bugs.python.org/issue?@action=redirect&bpo=33409]: Clarified the relationship between **PEP 538** [https://peps.python.org/pep-0538/]'s PYTHONCOERCECLOCALE and PEP 540's PYTHONUTF8 mode.
- bpo-33736 [https://bugs.python.org/issue?@action=redirect&bpo=33736]: Improve the documentation of `asyncio.open_connection()`, `asyncio.start_server()` and their UNIX socket counterparts.
- bpo-31432 [https://bugs.python.org/issue?@action=redirect&bpo=31432]: Clarify meaning of CERT_NONE, CERT_OPTIONAL, and CERT_REQUIRED flags for ssl.SSLContext.verify_mode.

Build

- bpo-5755 [https://bugs.python.org/issue?@action=redirect&bpo=5755]: Move `-Wstrict-prototypes` option to `CFLAGS_NODIST` from `OPT`. This option emitted annoying warnings when building extension modules written in C++.

Windows

- bpo-33720 [https://bugs.python.org/issue?@action=redirect&bpo=33720]: Reduces maximum marshal recursion depth on release builds.

IDLE

- bpo-33656 [https://bugs.python.org/issue?@action=redirect&bpo=33656]: On Windows, add API call saying that tk scales for DPI. On Windows 8.1+ or 10, with DPI compatibility properties of the Python binary unchanged, and a monitor resolution greater than 96 DPI, this should make text and lines sharper. It should otherwise have no effect.

- bpo-33768 [https://bugs.python.org/issue?@action=redirect&bpo=33768]: Clicking on a context line moves that line to the top of the editor window.
- bpo-33763 [https://bugs.python.org/issue?@action=redirect&bpo=33763]: IDLE: Use read-only text widget for code context instead of label widget.
- bpo-33664 [https://bugs.python.org/issue?@action=redirect&bpo=33664]: Scroll IDLE editor text by lines. Previously, the mouse wheel and scrollbar slider moved text by a fixed number of pixels, resulting in partial lines at the top of the editor box. The change also applies to the shell and grep output windows, but not to read-only text views.
- bpo-33679 [https://bugs.python.org/issue?@action=redirect&bpo=33679]: Enable theme-specific color configuration for Code Context. Use the Highlights tab to see the setting for built-in themes or add settings to custom themes.
- bpo-33642 [https://bugs.python.org/issue?@action=redirect&bpo=33642]: Display up to maxlines non-blank lines for Code Context. If there is no current context, show a single blank line.

Python 3.7.0 beta 5

Release date: 2018-05-30

Core and Builtins

- bpo-33622 [https://bugs.python.org/issue?@action=redirect&bpo=33622]: Fixed a leak when the garbage collector fails to add an object with the `__del__` method or referenced by it into the **`gc.garbage`** list. **`PyGC_Collect()`** can now be called when an exception is set and preserves it.
- bpo-33509 [https://bugs.python.org/issue?

@action=redirect&bpo=33509]: Fix module_globals parameter of warnings.warn_explicit(): don't crash if module_globals is not a dict.

- bpo-20104 [https://bugs.python.org/issue?@action=redirect&bpo=20104]: The new `os.posix_spawn` added in 3.7.0b1 was removed as we are still working on what the API should look like. Expect this in 3.8 instead.
- bpo-33475 [https://bugs.python.org/issue?@action=redirect&bpo=33475]: Fixed miscellaneous bugs in converting annotations to strings and optimized parentheses in the string representation.
- bpo-33391 [https://bugs.python.org/issue?@action=redirect&bpo=33391]: Fix a leak in set_symmetric_difference().
- bpo-28055 [https://bugs.python.org/issue?@action=redirect&bpo=28055]: Fix unaligned accesses in siphash24(). Patch by Rolf Eike Beer.
- bpo-32911 [https://bugs.python.org/issue?@action=redirect&bpo=32911]: Due to unexpected compatibility issues discovered during downstream beta testing, reverted bpo-29463 [https://bugs.python.org/issue?@action=redirect&bpo=29463]. `docstring` field is removed from Module, ClassDef, FunctionDef, and AsyncFunctionDef ast nodes which was added in 3.7a1. Docstring expression is restored as a first statement in their body. Based on patch by Inada Naoki.
- bpo-21983 [https://bugs.python.org/issue?@action=redirect&bpo=21983]: Fix a crash in `ctypes.cast()` in case the type argument is a ctypes structured data type. Patch by Eryk Sun and Oren Milman.

Library

- bpo-32751 [https://bugs.python.org/issue?@action=redirect&bpo=32751]: When cancelling the task due to a timeout, **`asyncio.wait_for()`** will now wait until the cancellation is complete.
- bpo-32684 [https://bugs.python.org/issue?@action=redirect&bpo=32684]: Fix gather to propagate cancellation of itself even with return_exceptions.

- bpo-33654 [https://bugs.python.org/issue?@action=redirect&bpo=33654]: Support protocol type switching in SSLTransport.set_protocol().
- bpo-33674 [https://bugs.python.org/issue?@action=redirect&bpo=33674]: Pause the transport as early as possible to further reduce the risk of data_received() being called before connection_made().
- bpo-33674 [https://bugs.python.org/issue?@action=redirect&bpo=33674]: Fix a race condition in SSLProtocol.connection_made() of asyncio.sslproto: start immediately the handshake instead of using call_soon(). Previously, data_received() could be called before the handshake started, causing the handshake to hang or fail.
- bpo-31647 [https://bugs.python.org/issue?@action=redirect&bpo=31647]: Fixed bug where calling write_eof() on a _SelectorSocketTransport after it's already closed raises AttributeError.
- bpo-32610 [https://bugs.python.org/issue?@action=redirect&bpo=32610]: Make asyncio.all_tasks() return only pending tasks.
- bpo-32410 [https://bugs.python.org/issue?@action=redirect&bpo=32410]: Avoid blocking on file IO in sendfile fallback code
- bpo-33469 [https://bugs.python.org/issue?@action=redirect&bpo=33469]: Fix RuntimeError after closing loop that used run_in_executor
- bpo-33672 [https://bugs.python.org/issue?@action=redirect&bpo=33672]: Fix Task.__repr__ crash with Cython's bogus coroutines
- bpo-33654 [https://bugs.python.org/issue?@action=redirect&bpo=33654]: Fix transport.set_protocol() to support switching between asyncio.Protocol and asyncio.BufferedProtocol. Fix loop.start_tls() to work with asyncio.BufferedProtocols.
- bpo-33652 [https://bugs.python.org/issue?@action=redirect&bpo=33652]: Pickles of type variables and subscripted generics are now future-proof and compatible with older Python versions.
- bpo-32493 [https://bugs.python.org/issue?@action=redirect&bpo=32493]: Fixed `uuid.uuid1()` on

FreeBSD.

- bpo-33618 [https://bugs.python.org/issue?@action=redirect&bpo=33618]: Finalize and document preliminary and experimental TLS 1.3 support with OpenSSL 1.1.1
- bpo-33623 [https://bugs.python.org/issue?@action=redirect&bpo=33623]: Fix possible SIGSGV when asyncio.Future is created in _del_
- bpo-30877 [https://bugs.python.org/issue?@action=redirect&bpo=30877]: Fixed a bug in the Python implementation of the JSON decoder that prevented the cache of parsed strings from clearing after finishing the decoding. Based on patch by c-fos.
- bpo-33570 [https://bugs.python.org/issue?@action=redirect&bpo=33570]: Change TLS 1.3 cipher suite settings for compatibility with OpenSSL 1.1.1-pre6 and newer. OpenSSL 1.1.1 will have TLS 1.3 ciphers enabled by default.
- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Do not simplify arguments to `typing.Union`. Now `Union[Manager, Employee]` is not simplified to `Employee` at runtime. Such simplification previously caused several bugs and limited possibilities for introspection.
- bpo-33540 [https://bugs.python.org/issue?@action=redirect&bpo=33540]: Add a new `block_on_close` class attribute to `ForkingMixIn` and `ThreadingMixIn` classes of **`socketserver`**.
- bpo-33548 [https://bugs.python.org/issue?@action=redirect&bpo=33548]: tempfile._candidate_tempdir_list should consider common TEMP locations
- bpo-33109 [https://bugs.python.org/issue?@action=redirect&bpo=33109]: argparse subparsers are once again not required by default, reverting the change in behavior introduced by bpo-26510 [https://bugs.python.org/issue?@action=redirect&bpo=26510] in 3.7.0a2.
- bpo-33536 [https://bugs.python.org/issue?@action=redirect&bpo=33536]: dataclasses.make_dataclass now checks for invalid field names and duplicate fields. Also, added a check for invalid field specifications.

- bpo-33542 [https://bugs.python.org/issue?@action=redirect&bpo=33542]: Prevent `uuid.get_node` from using a DUID instead of a MAC on Windows. Patch by Zvi Effron
- bpo-26819 [https://bugs.python.org/issue?@action=redirect&bpo=26819]: Fix race condition with `ReadTransport.resume_reading` in Windows proactor event loop.
- Fix failure in `typing.get_type_hints()` when ClassVar was provided as a string forward reference.
- bpo-33505 [https://bugs.python.org/issue?@action=redirect&bpo=33505]: Optimize asyncio.ensure_future() by reordering if checks: 1.17x faster.
- bpo-33497 [https://bugs.python.org/issue?@action=redirect&bpo=33497]: Add errors param to cgi.parse_multipart and make an encoding in FieldStorage use the given errors (needed for Twisted). Patch by Amber Brown.
- bpo-33495 [https://bugs.python.org/issue?@action=redirect&bpo=33495]: Change dataclasses.Fields repr to use the repr of each of its members, instead of str. This makes it more clear what each field actually represents. This is especially true for the “type” member.
- bpo-33453 [https://bugs.python.org/issue?@action=redirect&bpo=33453]: Fix dataclasses to work if using literal string type annotations or if using PEP 563 «Postponed Evaluation of Annotations». Only specific string prefixes are detected for both ClassVar («ClassVar» and «typing.ClassVar») and InitVar («InitVar» and «dataclasses.InitVar»).
- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Minor fixes in typing module: add annotations to `NamedTuple.__new__`, pass `*args` and `**kwds` in `Generic.__new__`. Original PRs by Paulius Šarka and Chad Dombrova.
- bpo-20087 [https://bugs.python.org/issue?@action=redirect&bpo=20087]: Updated alias mapping with glibc 2.27 supported locales.
- bpo-33422 [https://bugs.python.org/issue?@action=redirect&bpo=33422]: Fix trailing quotation marks getting deleted when looking up byte/string literals on pydoc.

Patch by Andrés Delfino.

- bpo-28167 [https://bugs.python.org/issue?@action=redirect&bpo=28167]: The function `platform.linux_distribution` and `platform.dist` now trigger a `DeprecationWarning` and have been marked for removal in Python 3.8
- bpo-33197 [https://bugs.python.org/issue?@action=redirect&bpo=33197]: Update error message when constructing invalid inspect.Parameters Patch by Donghee Na.
- bpo-33263 [https://bugs.python.org/issue?@action=redirect&bpo=33263]: Fix FD leak in `_SelectorSocketTransport` Patch by Vlad Starostin.
- bpo-32861 [https://bugs.python.org/issue?@action=redirect&bpo=32861]: The urllib.robotparser's `__str__` representation now includes wildcard entries and the «Crawl-delay» and «Request-rate» fields. Patch by Michael Lazar.
- bpo-32257 [https://bugs.python.org/issue?@action=redirect&bpo=32257]: The ssl module now contains OP_NO_RENEGOTIATION constant, available with OpenSSL 1.1.0h or 1.1.1.
- bpo-16865 [https://bugs.python.org/issue?@action=redirect&bpo=16865]: Support arrays >=2GiB in **`ctypes`**. Patch by Segev Finer.

Documentation

- bpo-23859 [https://bugs.python.org/issue?@action=redirect&bpo=23859]: Document that `asyncio.wait()` does not cancel its futures on timeout.
- bpo-32436 [https://bugs.python.org/issue?@action=redirect&bpo=32436]: Document **PEP 567** [https://peps.python.org/pep-0567/] changes to asyncio.
- bpo-33604 [https://bugs.python.org/issue?@action=redirect&bpo=33604]: Update HMAC md5 default to a DeprecationWarning, bump removal to 3.8.
- bpo-33503 [https://bugs.python.org/issue?@action=redirect&bpo=33503]: Fix broken pypi link
- bpo-33421 [https://bugs.python.org/issue?@action=redirect&bpo=33421]: Add missing documentation for `typing.AsyncContextManager`.

Tests

- bpo-33655 [https://bugs.python.org/issue?@action=redirect&bpo=33655]: Ignore test_posix_fallocate failures on BSD platforms that might be due to running on ZFS.
- bpo-32604 [https://bugs.python.org/issue?@action=redirect&bpo=32604]: Remove the _xxsubinterpreters module (meant for testing) and associated helpers. This module was originally added recently in 3.7b1.

Build

- bpo-33614 [https://bugs.python.org/issue?@action=redirect&bpo=33614]: Ensures module definition files for the stable ABI on Windows are correctly regenerated.
- bpo-33522 [https://bugs.python.org/issue?@action=redirect&bpo=33522]: Enable CI builds on Visual Studio Team Services at https://python.visualstudio.com/cpython
- bpo-33012 [https://bugs.python.org/issue?@action=redirect&bpo=33012]: Add `-Wno-cast-function-type` for gcc 8 for silencing warnings about function casts like casting to PyCFunction in method definition lists.

macOS

- bpo-13631 [https://bugs.python.org/issue?@action=redirect&bpo=13631]: The .editrc file in user's home directory is now processed correctly during the readline initialization through editline emulation on macOS.

IDLE

- bpo-33628 [https://bugs.python.org/issue?@action=redirect&bpo=33628]: IDLE: Cleanup codecontext.py and its test.
- bpo-33564 [https://bugs.python.org/issue?@action=redirect&bpo=33564]: IDLE's code context now recognizes async as a block opener.
- bpo-32831 [https://bugs.python.org/issue?

@action=redirect&bpo=32831]: Add docstrings and tests for codecontext.

Python 3.7.0 beta 4

Release date: 2018-05-02

Core and Builtins

- bpo-33363 [https://bugs.python.org/issue?@action=redirect&bpo=33363]: Raise a SyntaxError for `async with` and `async for` statements outside of async functions.
- bpo-33128 [https://bugs.python.org/issue?@action=redirect&bpo=33128]: Fix a bug that causes PathFinder to appear twice on sys.meta_path. Patch by Pablo Galindo Salgado.
- bpo-33312 [https://bugs.python.org/issue?@action=redirect&bpo=33312]: Fixed clang ubsan (undefined behavior sanitizer) warnings in dictobject.c by adjusting how the internal struct _dictkeysobject shared keys structure is declared.
- bpo-33231 [https://bugs.python.org/issue?@action=redirect&bpo=33231]: Fix potential memory leak in `normalizestring()`.
- bpo-33205 [https://bugs.python.org/issue?@action=redirect&bpo=33205]: Change dict growth function from `round_up_to_power_2(used*2+hashtable_size/2)` to `round_up_to_power_2(used*3)`. Previously, dict is shrunk only when `used == 0`. Now dict has more chance to be shrunk.
- bpo-29922 [https://bugs.python.org/issue?@action=redirect&bpo=29922]: Improved error messages in “async with” when `__aenter__()` or `__aexit__()` return non-awaitable object.
- bpo-33199 [https://bugs.python.org/issue?@action=redirect&bpo=33199]: Fix `ma_version_tag` in dict implementation is uninitialized when copying from key-sharing dict.

Library

- bpo-33281 [https://bugs.python.org/issue?@action=redirect&bpo=33281]: Fix ctypes.util.find_library regression on macOS.
- bpo-33383 [https://bugs.python.org/issue?@action=redirect&bpo=33383]: Fixed crash in the get() method of the `dbm.ndbm` database object when it is called with a single argument.
- bpo-33329 [https://bugs.python.org/issue?@action=redirect&bpo=33329]: Fix multiprocessing regression on newer glibcs
- bpo-991266 [https://bugs.python.org/issue?@action=redirect&bpo=991266]: Fix quoting of the `Comment` attribute of `http.cookies.SimpleCookie`.
- bpo-33131 [https://bugs.python.org/issue?@action=redirect&bpo=33131]: Upgrade bundled version of pip to 10.0.1.
- bpo-33308 [https://bugs.python.org/issue?@action=redirect&bpo=33308]: Fixed a crash in the `parser` module when converting an ST object to a tree of tuples or lists with `line_info=False` and `col_info=True`.
- bpo-33266 [https://bugs.python.org/issue?@action=redirect&bpo=33266]: lib2to3 now recognizes `rf'...'` strings.
- bpo-11594 [https://bugs.python.org/issue?@action=redirect&bpo=11594]: Ensure line-endings are respected when using lib2to3.
- bpo-33254 [https://bugs.python.org/issue?@action=redirect&bpo=33254]: Have `importlib.resources.contents()` and `importlib.abc.ResourceReader.contents()` return an iterable instead of an iterator.
- bpo-33256 [https://bugs.python.org/issue?@action=redirect&bpo=33256]: Fix display of `<module>` call in the html produced by `cgitb.html()`. Patch by Stéphane Blondon.
- bpo-33185 [https://bugs.python.org/issue?@action=redirect&bpo=33185]: Fixed regression when running pydoc with the `-m` switch. (The regression was introduced in

3.7.0b3 by the resolution of bpo-33053 [https://bugs.python.org/issue?@action=redirect&bpo=33053]) This fix also changed pydoc to add `os.getcwd()` to **`sys.path`** when necessary, rather than adding `"."`.

- bpo-33169 [https://bugs.python.org/issue?@action=redirect&bpo=33169]: Delete entries of `None` in **`sys.path_importer_cache`** when **`importlib.machinery.invalidate_caches()`** is called.
- bpo-33217 [https://bugs.python.org/issue?@action=redirect&bpo=33217]: Deprecate looking up non-Enum objects in Enum classes and Enum members (will raise **`TypeError`** in 3.8+).
- bpo-33203 [https://bugs.python.org/issue?@action=redirect&bpo=33203]: `random.Random.choice()` now raises `IndexError` for empty sequences consistently even when called from subclasses without a `getrandbits()` implementation.
- bpo-33224 [https://bugs.python.org/issue?@action=redirect&bpo=33224]: Update difflib.mdiff() for **PEP 479** [https://peps.python.org/pep-0479/]. Convert an uncaught StopIteration in a generator into a return-statement.
- bpo-33209 [https://bugs.python.org/issue?@action=redirect&bpo=33209]: End framing at the end of C implementation of **`pickle.Pickler.dump()`**.
- bpo-20104 [https://bugs.python.org/issue?@action=redirect&bpo=20104]: Improved error handling and fixed a reference leak in **`os.posix_spawn()`**.
- bpo-33175 [https://bugs.python.org/issue?@action=redirect&bpo=33175]: In dataclasses, Field.__set_name__ now looks up the __set_name__ special method on the class, not the instance, of the default value.
- bpo-33097 [https://bugs.python.org/issue?@action=redirect&bpo=33097]: Raise RuntimeError when `executor.submit` is called during interpreter shutdown.
- bpo-31908 [https://bugs.python.org/issue?@action=redirect&bpo=31908]: Fix output of cover files for `trace` module command-line tool. Previously emitted cover files only when `--missing` option was used. Patch by Michael Selik.

Documentation

- bpo-33378 [https://bugs.python.org/issue?@action=redirect&bpo=33378]: Add Korean language switcher for https://docs.python.org/3/
- bpo-33276 [https://bugs.python.org/issue?@action=redirect&bpo=33276]: Clarify that the `__path__` attribute on modules cannot be just any value.
- bpo-33201 [https://bugs.python.org/issue?@action=redirect&bpo=33201]: Modernize documentation for writing C extension types.
- bpo-33195 [https://bugs.python.org/issue?@action=redirect&bpo=33195]: Deprecate `Py_UNICODE` usage in `c-api/arg` document. `Py_UNICODE` related APIs are deprecated since Python 3.3, but it is missed in the document.
- bpo-8243 [https://bugs.python.org/issue?@action=redirect&bpo=8243]: Add a note about curses.addch and curses.addstr exception behavior when writing outside a window, or pad.
- bpo-32337 [https://bugs.python.org/issue?@action=redirect&bpo=32337]: Update documentation related with `dict` order.

Tests

- bpo-33358 [https://bugs.python.org/issue?@action=redirect&bpo=33358]: Fix `test_embed.test_pre_initialization_sys_options()` when the interpreter is built with `--enable-shared`.

Build

- bpo-33394 [https://bugs.python.org/issue?@action=redirect&bpo=33394]: Enable the verbose build for extension modules, when GNU make is passed macros on the command line.
- bpo-33393 [https://bugs.python.org/issue?@action=redirect&bpo=33393]: Update config.guess and config.sub files.
- bpo-33377 [https://bugs.python.org/issue?@action=redirect&bpo=33377]: Add new triplets for mips r6 and

riscv variants (used in extension suffixes).

- bpo-32232 [https://bugs.python.org/issue?@action=redirect&bpo=32232]: By default, modules configured in `Modules/Setup` are no longer built with `-DPy_BUILD_CORE`. Instead, modules that specifically need that preprocessor definition include it in their individual entries.
- bpo-33182 [https://bugs.python.org/issue?@action=redirect&bpo=33182]: The embedding tests can once again be built with clang 6.0

Windows

- bpo-33184 [https://bugs.python.org/issue?@action=redirect&bpo=33184]: Update Windows installer to use OpenSSL 1.1.0h.

macOS

- bpo-33184 [https://bugs.python.org/issue?@action=redirect&bpo=33184]: Update macOS installer build to use OpenSSL 1.1.0h.

IDLE

- bpo-21474 [https://bugs.python.org/issue?@action=redirect&bpo=21474]: Update word/identifier definition from ascii to unicode. In text and entry boxes, this affects selection by double-click, movement left/right by control-left/right, and deletion left/right by control-BACKSPACE/DEL.
- bpo-33204 [https://bugs.python.org/issue?@action=redirect&bpo=33204]: IDLE: consistently color invalid string prefixes. A “u” string prefix cannot be paired with either “r” or “f”. Consistently color as much of the prefix, starting at the right, as is valid. Revise and extend colorizer test.

Tools/Demos

- bpo-33189 [https://bugs.python.org/issue?@action=redirect&bpo=33189]: **pygettext.py** now recognizes only literal strings as docstrings and translatable strings, and rejects bytes literals and f-string expressions.
- bpo-31920 [https://bugs.python.org/issue?@action=redirect&bpo=31920]: Fixed handling directories as arguments in the `pygettext` script. Based on patch by Oleg Krasnikov.
- bpo-29673 [https://bugs.python.org/issue?@action=redirect&bpo=29673]: Fix pystackv and pystack gdbinit macros.
- bpo-31583 [https://bugs.python.org/issue?@action=redirect&bpo=31583]: Fix 2to3 for using with –add-suffix option but without –output-dir option for relative path to files in current directory.

Python 3.7.0 beta 3

Release date: 2018-03-29

Security

- bpo-33136 [https://bugs.python.org/issue?@action=redirect&bpo=33136]: Harden ssl module against LibreSSL **CVE 2018-8970** [https://www.cve.org/CVERecord?id=CVE-2018-8970]. X509_VERIFY_PARAM_set1_host() is called with an explicit namelen. A new test ensures that NULL bytes are not allowed.
- bpo-33001 [https://bugs.python.org/issue?@action=redirect&bpo=33001]: Minimal fix to prevent buffer overrun in os.symlink on Windows
- bpo-32981 [https://bugs.python.org/issue?@action=redirect&bpo=32981]: Regexes in difflib and poplib were vulnerable to catastrophic backtracking. These regexes formed potential DOS vectors (REDOS). They have been refactored. This resolves **CVE 2018-1060** [https://www.cve.org/CVERecord?id=CVE-2018-1060] and **CVE 2018-1061** [https://www.cve.org/CVERecord?id=CVE-2018-1061]. Patch by Jamie Davis.

Core and Builtins

- bpo-33053 [https://bugs.python.org/issue?@action=redirect&bpo=33053]: When using the -m switch, sys.path[0] is now explicitly expanded as the *starting* working directory, rather than being left as the empty path (which allows imports from the current working directory at the time of the import)
- bpo-33018 [https://bugs.python.org/issue?@action=redirect&bpo=33018]: Improve consistency of errors raised by `issubclass()` when called with a non-class and an abstract base class as the first and second arguments, respectively. Patch by Josh Bronson.
- bpo-33041 [https://bugs.python.org/issue?@action=redirect&bpo=33041]: Fixed jumping when the function contains an `async for` loop.
- bpo-33026 [https://bugs.python.org/issue?@action=redirect&bpo=33026]: Fixed jumping out of «with» block by setting f_lineno.
- bpo-33005 [https://bugs.python.org/issue?@action=redirect&bpo=33005]: Fix a crash on fork when using a custom memory allocator (ex: using PYTHONMALLOC env var). _PyGILState_Reinit() and _PyInterpreterState_Enable() now use the default RAW memory allocator to allocate a new interpreters mutex on fork.
- bpo-17288 [https://bugs.python.org/issue?@action=redirect&bpo=17288]: Prevent jumps from "return" and "exception" trace events.
- bpo-32836 [https://bugs.python.org/issue?@action=redirect&bpo=32836]: Don't use temporary variables in cases of list/dict/set comprehensions

Library

- bpo-33141 [https://bugs.python.org/issue?@action=redirect&bpo=33141]: Have Field objects pass through __set_name__ to their default values, if they have their own __set_name__.
- bpo-33096 [https://bugs.python.org/issue?@action=redirect&bpo=33096]: Allow ttk.Treeview.insert to insert

iid that has a false boolean value. Note iid=0 and iid=False would be same. Patch by Garvit Khatri.

- bpo-32873 [https://bugs.python.org/issue?@action=redirect&bpo=32873]: Treat type variables and special typing forms as immutable by copy and pickle. This fixes several minor issues and inconsistencies, and improves backwards compatibility with Python 3.6.
- bpo-33134 [https://bugs.python.org/issue?@action=redirect&bpo=33134]: When computing dataclass's _hash_, use the lookup table to contain the function which returns the _hash_ value. This is an improvement over looking up a string, and then testing that string to see what to do.
- bpo-33127 [https://bugs.python.org/issue?@action=redirect&bpo=33127]: The ssl module now compiles with LibreSSL 2.7.1.
- bpo-32505 [https://bugs.python.org/issue?@action=redirect&bpo=32505]: Raise TypeError if a member variable of a dataclass is of type Field, but doesn't have a type annotation.
- bpo-33078 [https://bugs.python.org/issue?@action=redirect&bpo=33078]: Fix the failure on OSX caused by the tests relying on sem_getvalue
- bpo-33116 [https://bugs.python.org/issue?@action=redirect&bpo=33116]: Add "Field" to dataclasses._all_.
- bpo-32896 [https://bugs.python.org/issue?@action=redirect&bpo=32896]: Fix an error where subclassing a dataclass with a field that uses a default_factory would generate an incorrect class.
- bpo-33100 [https://bugs.python.org/issue?@action=redirect&bpo=33100]: Dataclasses: If a field has a default value that's a MemberDescriptorType, then it's from that field being in _slots_, not an actual default value.
- bpo-32953 [https://bugs.python.org/issue?@action=redirect&bpo=32953]: If a non-dataclass inherits from a frozen dataclass, allow attributes to be added to the derived class. Only attributes from the frozen dataclass cannot be assigned to. Require all dataclasses in a hierarchy to be either all frozen or all non-frozen.
- bpo-33061 [https://bugs.python.org/issue?

@action=redirect&bpo=33061]: Add missing `NoReturn` to `__all__` in typing.py
- bpo-33078 [https://bugs.python.org/issue?@action=redirect&bpo=33078]: Fix the size handling in multiprocessing.Queue when a pickling error occurs.
- bpo-33064 [https://bugs.python.org/issue?@action=redirect&bpo=33064]: lib2to3 now properly supports trailing commas after `*args` and `**kwargs` in function signatures.
- bpo-33056 [https://bugs.python.org/issue?@action=redirect&bpo=33056]: FIX properly close leaking fds in concurrent.futures.ProcessPoolExecutor.
- bpo-33021 [https://bugs.python.org/issue?@action=redirect&bpo=33021]: Release the GIL during fstat() calls, avoiding hang of all threads when calling mmap.mmap(), os.urandom(), and random.seed(). Patch by Nir Soffer.
- bpo-31804 [https://bugs.python.org/issue?@action=redirect&bpo=31804]: Avoid failing in multiprocessing.Process if the standard streams are closed or None at exit.
- bpo-33037 [https://bugs.python.org/issue?@action=redirect&bpo=33037]: Skip sending/receiving data after SSL transport closing.
- bpo-27683 [https://bugs.python.org/issue?@action=redirect&bpo=27683]: Fix a regression in **`ipaddress`** that result of **`hosts()`** is empty when the network is constructed by a tuple containing an integer mask and only 1 bit left for addresses.
- bpo-32999 [https://bugs.python.org/issue?@action=redirect&bpo=32999]: Fix C implementation of `ABC.__subclasscheck__(cls, subclass)` crashed when `subclass` is not a type object.
- bpo-33009 [https://bugs.python.org/issue?@action=redirect&bpo=33009]: Fix inspect.signature() for single-parameter partialmethods.
- bpo-32969 [https://bugs.python.org/issue?@action=redirect&bpo=32969]: Expose several missing constants in zlib and fix corresponding documentation.
- bpo-32056 [https://bugs.python.org/issue?

@action=redirect&bpo=32056]: Improved exceptions raised for invalid number of channels and sample width when read an audio file in modules `aifc`, `wave` and `sunau`.

- bpo-32844 [https://bugs.python.org/issue?@action=redirect&bpo=32844]: Fix wrong redirection of a low descriptor (0 or 1) to stderr in subprocess if another low descriptor is closed.
- bpo-32857 [https://bugs.python.org/issue?@action=redirect&bpo=32857]: In `tkinter`, `after_cancel(None)` now raises a `ValueError` instead of canceling the first scheduled function. Patch by Cheryl Sabella.
- bpo-31639 [https://bugs.python.org/issue?@action=redirect&bpo=31639]: http.server now exposes a ThreadedHTTPServer class and uses it when the module is run with `-m` to cope with web browsers pre-opening sockets.
- bpo-27645 [https://bugs.python.org/issue?@action=redirect&bpo=27645]: `sqlite3.Connection` now exposes a `backup` method, if the underlying SQLite library is at version 3.6.11 or higher. Patch by Lele Gaifax.

Documentation

- bpo-33126 [https://bugs.python.org/issue?@action=redirect&bpo=33126]: Document PyBuffer_ToContiguous().
- bpo-27212 [https://bugs.python.org/issue?@action=redirect&bpo=27212]: Modify documentation for the `islice()` recipe to consume initial values up to the start index.
- bpo-28247 [https://bugs.python.org/issue?@action=redirect&bpo=28247]: Update `zipapp` documentation to describe how to make standalone applications.
- bpo-18802 [https://bugs.python.org/issue?@action=redirect&bpo=18802]: Documentation changes for ipaddress. Patch by Jon Foster and Berker Peksag.
- bpo-27428 [https://bugs.python.org/issue?@action=redirect&bpo=27428]: Update documentation to clarify that `WindowsRegistryFinder` implements `MetaPathFinder`. (Patch by Himanshu Lakhara)

Tests

- bpo-32872 [https://bugs.python.org/issue?@action=redirect&bpo=32872]: Avoid regrtest compatibility issue with namespace packages.
- bpo-32517 [https://bugs.python.org/issue?@action=redirect&bpo=32517]: Fix failing `test_asyncio` on macOS 10.12.2+ due to transport of `KqueueSelector` loop was not being closed.
- bpo-19417 [https://bugs.python.org/issue?@action=redirect&bpo=19417]: Add test_bdb.py.

Build

- bpo-33163 [https://bugs.python.org/issue?@action=redirect&bpo=33163]: Upgrade pip to 9.0.3 and setuptools to v39.0.1.

Windows

- bpo-33016 [https://bugs.python.org/issue?@action=redirect&bpo=33016]: Fix potential use of uninitialized memory in nt._getfinalpathname
- bpo-32903 [https://bugs.python.org/issue?@action=redirect&bpo=32903]: Fix a memory leak in os.chdir() on Windows if the current directory is set to a UNC path.

macOS

- bpo-32726 [https://bugs.python.org/issue?@action=redirect&bpo=32726]: Build and link with private copy of Tcl/Tk 8.6 for the macOS 10.6+ installer. The 10.9+ installer variant already does this. This means that the Python 3.7 provided by the python.org macOS installers no longer need or use any external versions of Tcl/Tk, either system-provided or user-installed, such as ActiveTcl.

IDLE

- bpo-32984 [https://bugs.python.org/issue?@action=redirect&bpo=32984]: Set `__file__` while running a startup file. Like Python, IDLE optionally runs one startup file in the Shell window before presenting the first interactive input prompt. For IDLE, `-s` runs a file named in environmental variable **`IDLESTARTUP`** or **`PYTHONSTARTUP`**; `-r file` runs `file`. Python sets `__file__` to the startup file name before running the file and unsets it before the first prompt. IDLE now does the same when run normally, without the `-n` option.
- bpo-32940 [https://bugs.python.org/issue?@action=redirect&bpo=32940]: Simplify and rename StringTranslatePseudoMapping in pyparse.

Tools/Demos

- bpo-32885 [https://bugs.python.org/issue?@action=redirect&bpo=32885]: Add an `-n` flag for `Tools/scripts/pathfix.py` to disable automatic backup creation (files with `~` suffix).

C API

- bpo-33042 [https://bugs.python.org/issue?@action=redirect&bpo=33042]: Embedding applications may once again call PySys_ResetWarnOptions, PySys_AddWarnOption, and PySys_AddXOption prior to calling Py_Initialize.
- bpo-32374 [https://bugs.python.org/issue?@action=redirect&bpo=32374]: Document that m_traverse for multi-phase initialized modules can be called with m_state=NULL, and add a sanity check

Python 3.7.0 beta 2

Release date: 2018-02-27

Security

- bpo-28414 [https://bugs.python.org/issue?

@action=redirect&bpo=28414]: The ssl module now allows users to perform their own IDN en/decoding when using SNI.

Core and Builtins

- bpo-32889 [https://bugs.python.org/issue?@action=redirect&bpo=32889]: Update Valgrind suppression list to account for the rename of `Py_ADDRESS_IN_RANG` to `address_in_range`.
- bpo-31356 [https://bugs.python.org/issue?@action=redirect&bpo=31356]: Remove the new API added in bpo-31356 [https://bugs.python.org/issue?@action=redirect&bpo=31356] (gc.ensure_disabled() context manager).
- bpo-32305 [https://bugs.python.org/issue?@action=redirect&bpo=32305]: For namespace packages, ensure that both `__file__` and `__spec__.origin` are set to None.
- bpo-32303 [https://bugs.python.org/issue?@action=redirect&bpo=32303]: Make sure `__spec__.loader` matches `__loader__` for namespace packages.
- bpo-32711 [https://bugs.python.org/issue?@action=redirect&bpo=32711]: Fix the warning messages for Python/ast_unparse.c. Patch by Stéphane Wirtel
- bpo-32583 [https://bugs.python.org/issue?@action=redirect&bpo=32583]: Fix possible crashing in builtin Unicode decoders caused by write out-of-bound errors when using customized decode error handlers.

Library

- bpo-32960 [https://bugs.python.org/issue?@action=redirect&bpo=32960]: For dataclasses, disallow inheriting frozen from non-frozen classes, and also disallow inheriting non-frozen from frozen classes. This restriction will be relaxed at a future date.
- bpo-32713 [https://bugs.python.org/issue?@action=redirect&bpo=32713]: Fixed tarfile.itn handling of out-of-bounds float values. Patch by Joffrey Fuhrer.
- bpo-32951 [https://bugs.python.org/issue?

@action=redirect&bpo=32951]: Direct instantiation of SSLSocket and SSLObject objects is now prohibited. The constructors were never documented, tested, or designed as public constructors. Users were suppose to use ssl.wrap_socket() or SSLContext.

- bpo-32929 [https://bugs.python.org/issue?@action=redirect&bpo=32929]: Remove the tri-state parameter «hash», and add the boolean «unsafe_hash». If unsafe_hash is True, add a _hash_ function, but if a _hash_ exists, raise TypeError. If unsafe_hash is False, add a _hash_ based on the values of eq= and frozen=. The unsafe_hash=False behavior is the same as the old hash=None behavior. unsafe_hash=False is the default, just as hash=None used to be.
- bpo-32947 [https://bugs.python.org/issue?@action=redirect&bpo=32947]: Add OP_ENABLE_MIDDLEBOX_COMPAT and test workaround for TLSv1.3 for future compatibility with OpenSSL 1.1.1.
- bpo-30622 [https://bugs.python.org/issue?@action=redirect&bpo=30622]: The ssl module now detects missing NPN support in LibreSSL.
- bpo-32922 [https://bugs.python.org/issue?@action=redirect&bpo=32922]: dbm.open() now encodes filename with the filesystem encoding rather than default encoding.
- bpo-32859 [https://bugs.python.org/issue?@action=redirect&bpo=32859]: In `os.dup2`, don’t check every call whether the `dup3` syscall exists or not.
- bpo-32556 [https://bugs.python.org/issue?@action=redirect&bpo=32556]: nt._getfinalpathname, nt._getvolumepathname and nt._getdiskusage now correctly convert from bytes.
- bpo-25988 [https://bugs.python.org/issue?@action=redirect&bpo=25988]: Emit a **`DeprecationWarning`** when using or importing an ABC directly from **`collections`** rather than from **`collections.abc`**.
- bpo-21060 [https://bugs.python.org/issue?@action=redirect&bpo=21060]: Rewrite confusing message from setup.py upload from «No dist file created in earlier command» to the more helpful «Must create and upload files in one command».

- bpo-32852 [https://bugs.python.org/issue?@action=redirect&bpo=32852]: Make sure sys.argv remains as a list when running trace.
- bpo-31333 [https://bugs.python.org/issue?@action=redirect&bpo=31333]: `_abc` module is added. It is a speedup module with C implementations for various functions and methods in `abc`. Creating an ABC subclass and calling `isinstance` or `issubclass` with an ABC subclass are up to 1.5x faster. In addition, this makes Python start-up up to 10% faster. Note that the new implementation hides internal registry and caches, previously accessible via private attributes `_abc_registry`, `_abc_cache`, and `_abc_negative_cache`. There are three debugging helper methods that can be used instead `_dump_registry`, `_abc_registry_clear`, and `_abc_caches_clear`.
- bpo-32841 [https://bugs.python.org/issue?@action=redirect&bpo=32841]: Fixed `asyncio.Condition` issue which silently ignored cancellation after notifying and cancelling a conditional lock. Patch by Bar Harel.
- bpo-32819 [https://bugs.python.org/issue?@action=redirect&bpo=32819]: ssl.match_hostname() has been simplified and no longer depends on re and ipaddress module for wildcard and IP addresses. Error reporting for invalid wildcards has been improved.
- bpo-32394 [https://bugs.python.org/issue?@action=redirect&bpo=32394]: socket: Remove TCP_FASTOPEN,TCP_KEEPCNT,TCP_KEEPIDLE,TCP_KEEPINTVL flags on older version Windows during run-time.
- bpo-31787 [https://bugs.python.org/issue?@action=redirect&bpo=31787]: Fixed refleaks of `__init__()` methods in various modules. (Contributed by Oren Milman)
- bpo-30157 [https://bugs.python.org/issue?@action=redirect&bpo=30157]: Fixed guessing quote and delimiter in csv.Sniffer.sniff() when only the last field is quoted. Patch by Jake Davis.
- bpo-32792 [https://bugs.python.org/issue?@action=redirect&bpo=32792]: collections.ChainMap() preserves the order of the underlying mappings.
- bpo-32775 [https://bugs.python.org/issue?@action=redirect&bpo=32775]: **`fnmatch.translate()`** no

longer produces patterns which contain set operations. Sets starting with “[” or containing “–”, “&&”, “~~” or “||” will be interpreted differently in regular expressions in future versions. Currently they emit warnings. fnmatch.translate() now avoids producing patterns containing such sets by accident.

- bpo-32622 [https://bugs.python.org/issue?@action=redirect&bpo=32622]: Implement native fast sendfile for Windows proactor event loop.
- bpo-32777 [https://bugs.python.org/issue?@action=redirect&bpo=32777]: Fix a rare but potential pre-exec child process deadlock in subprocess on POSIX systems when marking file descriptors inheritable on exec in the child process. This bug appears to have been introduced in 3.4.
- bpo-32647 [https://bugs.python.org/issue?@action=redirect&bpo=32647]: The ctypes module used to depend on indirect linking for dlopen. The shared extension is now explicitly linked against libdl on platforms with dl.
- bpo-32741 [https://bugs.python.org/issue?@action=redirect&bpo=32741]: Implement `asyncio.TimerHandle.when()` method.
- bpo-32691 [https://bugs.python.org/issue?@action=redirect&bpo=32691]: Use mod_spec.parent when running modules with pdb
- bpo-32734 [https://bugs.python.org/issue?@action=redirect&bpo=32734]: Fixed `asyncio.Lock()` safety issue which allowed acquiring and locking the same lock multiple times, without it being free. Patch by Bar Harel.
- bpo-32727 [https://bugs.python.org/issue?@action=redirect&bpo=32727]: Do not include name field in SMTP envelope from address. Patch by Stéphane Wirtel
- bpo-31453 [https://bugs.python.org/issue?@action=redirect&bpo=31453]: Add TLSVersion constants and SSLContext.maximum_version / minimum_version attributes. The new API wraps OpenSSL 1.1 https://web.archive.org/web/20180309043602/https://www.openssl.org/docs/man1.1.0/ssl/SSL_CTX_set_min_proto_version.html feature.
- bpo-24334 [https://bugs.python.org/issue?@action=redirect&bpo=24334]: Internal implementation details of ssl module were cleaned up. The SSLSocket has one less layer

of indirection. Owner and session information are now handled by the SSLSocket and SSLObject constructor. Channel binding implementation has been simplified.

- bpo-31848 [https://bugs.python.org/issue?@action=redirect&bpo=31848]: Fix the error handling in Aifc_read.initfp() when the SSND chunk is not found. Patch by Zackery Spytz.
- bpo-32585 [https://bugs.python.org/issue?@action=redirect&bpo=32585]: Add Ttk spinbox widget to **`tkinter.ttk`**. Patch by Alan D Moore.
- bpo-32221 [https://bugs.python.org/issue?@action=redirect&bpo=32221]: Various functions returning tuple containing IPv6 addresses now omit `%scope` part since the same information is already encoded in *scopeid* tuple item. Especially this speeds up **`socket.recvfrom()`** when it receives multicast packet since useless resolving of network interface name is omitted.
- bpo-30693 [https://bugs.python.org/issue?@action=redirect&bpo=30693]: The TarFile class now recurses directories in a reproducible way.
- bpo-30693 [https://bugs.python.org/issue?@action=redirect&bpo=30693]: The ZipFile class now recurses directories in a reproducible way.

Documentation

- bpo-28124 [https://bugs.python.org/issue?@action=redirect&bpo=28124]: The ssl module function ssl.wrap_socket() has been de-emphasized and deprecated in favor of the more secure and efficient SSLContext.wrap_socket() method.
- bpo-17232 [https://bugs.python.org/issue?@action=redirect&bpo=17232]: Clarify docs for -O and -OO. Patch by Terry Reedy.
- bpo-32436 [https://bugs.python.org/issue?@action=redirect&bpo=32436]: Add documentation for the contextvars module (PEP 567).
- bpo-32800 [https://bugs.python.org/issue?@action=redirect&bpo=32800]: Update link to w3c doc for xml default namespaces.

- bpo-11015 [https://bugs.python.org/issue?@action=redirect&bpo=11015]: Update `test.support` documentation.
- bpo-8722 [https://bugs.python.org/issue?@action=redirect&bpo=8722]: Document `__getattr__()` behavior when property `get()` method raises `AttributeError`.
- bpo-32614 [https://bugs.python.org/issue?@action=redirect&bpo=32614]: Modify RE examples in documentation to use raw strings to prevent `DeprecationWarning` and add text to REGEX HOWTO to highlight the deprecation.
- bpo-31972 [https://bugs.python.org/issue?@action=redirect&bpo=31972]: Improve docstrings for `pathlib.PurePath` subclasses.

Tests

- bpo-31809 [https://bugs.python.org/issue?@action=redirect&bpo=31809]: Add tests to verify connection with secp ECDH curves.

Build

- bpo-32898 [https://bugs.python.org/issue?@action=redirect&bpo=32898]: Fix the python debug build when using COUNT_ALLOCS.

Windows

- bpo-32901 [https://bugs.python.org/issue?@action=redirect&bpo=32901]: Update Tcl and Tk versions to 8.6.8
- bpo-31966 [https://bugs.python.org/issue?@action=redirect&bpo=31966]: Fixed WindowsConsoleIO.write() for writing empty data.
- bpo-32409 [https://bugs.python.org/issue?@action=redirect&bpo=32409]: Ensures activate.bat can handle Unicode contents.
- bpo-32457 [https://bugs.python.org/issue?@action=redirect&bpo=32457]: Improves handling of

denormalized executable path when launching Python.

- bpo-32370 [https://bugs.python.org/issue?@action=redirect&bpo=32370]: Use the correct encoding for ipconfig output in the uuid module. Patch by Segev Finer.
- bpo-29248 [https://bugs.python.org/issue?@action=redirect&bpo=29248]: Fix `os.readlink()` on Windows, which was mistakenly treating the `PrintNameOffset` field of the reparse data buffer as a number of characters instead of bytes. Patch by Craig Holmquist and SSE4.

macOS

- bpo-32901 [https://bugs.python.org/issue?@action=redirect&bpo=32901]: Update macOS 10.9+ installer to Tcl/Tk 8.6.8.

IDLE

- bpo-32916 [https://bugs.python.org/issue?@action=redirect&bpo=32916]: Change `str` to `code` in pyparse.
- bpo-32905 [https://bugs.python.org/issue?@action=redirect&bpo=32905]: Remove unused code in pyparse module.
- bpo-32874 [https://bugs.python.org/issue?@action=redirect&bpo=32874]: Add tests for pyparse.
- bpo-32837 [https://bugs.python.org/issue?@action=redirect&bpo=32837]: Using the system and place-dependent default encoding for open() is a bad idea for IDLE’s system and location-independent files.
- bpo-32826 [https://bugs.python.org/issue?@action=redirect&bpo=32826]: Add «encoding=utf-8» to open() in IDLE’s test_help_about. GUI test test_file_buttons() only looks at initial ascii-only lines, but failed on systems where open() defaults to “ascii” because readline() internally reads and decodes far enough ahead to encounter a non-ascii character in CREDITS.txt.
- bpo-32765 [https://bugs.python.org/issue?@action=redirect&bpo=32765]: Update configdialog General tab docstring to add new widgets to the widget list.

Tools/Demos

- bpo-32222 [https://bugs.python.org/issue?@action=redirect&bpo=32222]: Fix pygettext not extracting docstrings for functions with type annotated arguments. Patch by Toby Harradine.

Python 3.7.0 beta 1

Release date: 2018-01-30

Core and Builtins

- bpo-32703 [https://bugs.python.org/issue?@action=redirect&bpo=32703]: Fix coroutine's ResourceWarning when there's an active error set when it's being finalized.
- bpo-32650 [https://bugs.python.org/issue?@action=redirect&bpo=32650]: Pdb and other debuggers dependent on bdb.py will correctly step over (next command) native coroutines. Patch by Pablo Galindo.
- bpo-28685 [https://bugs.python.org/issue?@action=redirect&bpo=28685]: Optimize list.sort() and sorted() by using type specialized comparisons when possible.
- bpo-32685 [https://bugs.python.org/issue?@action=redirect&bpo=32685]: Improve suggestion when the Python 2 form of print statement is either present on the same line as the header of a compound statement or else terminated by a semi-colon instead of a newline. Patch by Nitish Chandra.
- bpo-32697 [https://bugs.python.org/issue?@action=redirect&bpo=32697]: Python now explicitly preserves the definition order of keyword-only parameters. It's always preserved their order, but this behavior was never guaranteed before; this behavior is now guaranteed and tested.
- bpo-32690 [https://bugs.python.org/issue?@action=redirect&bpo=32690]: The locals() dictionary now displays in the lexical order that variables were defined. Previously, the order was reversed.
- bpo-32677 [https://bugs.python.org/issue?

@action=redirect&bpo=32677]: Add `.isascii()` method to `str`, `bytes` and `bytearray`. It can be used to test that string contains only ASCII characters.

- bpo-32670 [https://bugs.python.org/issue?@action=redirect&bpo=32670]: Enforce **PEP 479** [https://peps.python.org/pep-0479/] for all code. This means that manually raising a StopIteration exception from a generator is prohibited for all code, regardless of whether "from _future_ import generator_stop" was used or not.
- bpo-32591 [https://bugs.python.org/issue?@action=redirect&bpo=32591]: Added built-in support for tracking the origin of coroutine objects; see sys.set_coroutine_origin_tracking_depth and CoroutineType.cr_origin. This replaces the asyncio debug mode's use of coroutine wrapping for native coroutine objects.
- bpo-31368 [https://bugs.python.org/issue?@action=redirect&bpo=31368]: Expose preadv and pwritev system calls in the os module. Patch by Pablo Galindo
- bpo-32544 [https://bugs.python.org/issue?@action=redirect&bpo=32544]: `hasattr(obj, name)` and `getattr(obj, name, default)` are about 4 times faster than before when `name` is not found and `obj` doesn't override `__getattr__` or `__getattribute__`.
- bpo-26163 [https://bugs.python.org/issue?@action=redirect&bpo=26163]: Improved frozenset() hash to create more distinct hash values when faced with datasets containing many similar values.
- bpo-32550 [https://bugs.python.org/issue?@action=redirect&bpo=32550]: Remove the STORE_ANNOTATION bytecode.
- bpo-20104 [https://bugs.python.org/issue?@action=redirect&bpo=20104]: Expose posix_spawn as a low level API in the os module. (removed before 3.7.0rc1)
- bpo-24340 [https://bugs.python.org/issue?@action=redirect&bpo=24340]: Fixed estimation of the code stack size.
- bpo-32436 [https://bugs.python.org/issue?@action=redirect&bpo=32436]: Implement **PEP 567** [https://peps.python.org/pep-0567/] Context Variables.

- bpo-18533 [https://bugs.python.org/issue?@action=redirect&bpo=18533]: `repr()` on a dict containing its own `values()` or `items()` no longer raises `RecursionError`; OrderedDict similarly. Instead, use `...`, as for other recursive structures. Patch by Ben North.
- bpo-20891 [https://bugs.python.org/issue?@action=redirect&bpo=20891]: Py_Initialize() now creates the GIL. The GIL is no longer created «on demand» to fix a race condition when PyGILState_Ensure() is called in a non-Python thread.
- bpo-32028 [https://bugs.python.org/issue?@action=redirect&bpo=32028]: Leading whitespace is now correctly ignored when generating suggestions for converting Py2 print statements to Py3 builtin print function calls. Patch by Sanyam Khurana.
- bpo-31179 [https://bugs.python.org/issue?@action=redirect&bpo=31179]: Make dict.copy() up to 5.5 times faster.
- bpo-31113 [https://bugs.python.org/issue?@action=redirect&bpo=31113]: Get rid of recursion in the compiler for normal control flow.

Library

- bpo-25988 [https://bugs.python.org/issue?@action=redirect&bpo=25988]: Deprecate exposing the contents of collections.abc in the regular collections module.
- bpo-31429 [https://bugs.python.org/issue?@action=redirect&bpo=31429]: The default cipher suite selection of the ssl module now uses a blacklist approach rather than a hard-coded whitelist. Python no longer re-enables ciphers that have been blocked by OpenSSL security update. Default cipher suite selection can be configured on compile time.
- bpo-30306 [https://bugs.python.org/issue?@action=redirect&bpo=30306]: contextlib.contextmanager now releases the arguments passed to the underlying generator as soon as the context manager is entered. Previously it would keep them alive for as long as the context manager was alive, even when not being used as a function decorator. Patch by Martin Teichmann.

- bpo-21417 [https://bugs.python.org/issue?@action=redirect&bpo=21417]: Added support for setting the compression level for zipfile.ZipFile.
- bpo-32251 [https://bugs.python.org/issue?@action=redirect&bpo=32251]: Implement asyncio.BufferedProtocol (provisional API).
- bpo-32513 [https://bugs.python.org/issue?@action=redirect&bpo=32513]: In dataclasses, allow easier overriding of dunder methods without specifying decorator parameters.
- bpo-32660 [https://bugs.python.org/issue?@action=redirect&bpo=32660]: **`termios`** makes available `FIONREAD`, `FIONCLEX`, `FIOCLEX`, `FIOASYNC` and `FIONBIO` also under Solaris/derivatives.
- bpo-27931 [https://bugs.python.org/issue?@action=redirect&bpo=27931]: Fix email address header parsing error when the username is an empty quoted string. Patch by Xiang Zhang.
- bpo-32659 [https://bugs.python.org/issue?@action=redirect&bpo=32659]: Under Solaris and derivatives, **`os.stat_result`** provides a st_fstype attribute.
- bpo-32662 [https://bugs.python.org/issue?@action=redirect&bpo=32662]: Implement Server.start_serving(), Server.serve_forever(), and Server.is_serving() methods. Add "start_serving" keyword parameter to loop.create_server() and loop.create_unix_server().
- bpo-32391 [https://bugs.python.org/issue?@action=redirect&bpo=32391]: Implement **`asyncio.StreamWriter.wait_closed()`** and **`asyncio.StreamWriter.is_closing()`** methods
- bpo-32643 [https://bugs.python.org/issue?@action=redirect&bpo=32643]: Make Task._step, Task._wakeup and Future._schedule_callbacks methods private.
- bpo-32630 [https://bugs.python.org/issue?@action=redirect&bpo=32630]: Refactor decimal module to use contextvars to store decimal context.
- bpo-32622 [https://bugs.python.org/issue?@action=redirect&bpo=32622]: Add **`asyncio.AbstractEventLoop.sendfile()`** method.
- bpo-32304 [https://bugs.python.org/issue?

@action=redirect&bpo=32304]: distutils” upload command no longer corrupts tar files ending with a CR byte, and no longer tries to convert CR to CRLF in any of the upload text fields.

- bpo-32502 [https://bugs.python.org/issue?@action=redirect&bpo=32502]: uuid.uuid1 no longer raises an exception if a 64-bit hardware address is encountered.
- bpo-32596 [https://bugs.python.org/issue?@action=redirect&bpo=32596]: `concurrent.futures` imports `ThreadPoolExecutor` and `ProcessPoolExecutor` lazily (using **PEP 562** [https://peps.python.org/pep-0562/]). It makes `import asyncio` about 15% faster because asyncio uses only `ThreadPoolExecutor` by default.
- bpo-31801 [https://bugs.python.org/issue?@action=redirect&bpo=31801]: Add `_ignore_` to `Enum` so temporary variables can be used during class construction without being turned into members.
- bpo-32576 [https://bugs.python.org/issue?@action=redirect&bpo=32576]: Use queue.SimpleQueue() in places where it can be invoked from a weakref callback.
- bpo-32574 [https://bugs.python.org/issue?@action=redirect&bpo=32574]: Fix memory leak in asyncio.Queue, when the queue has limited size and it is full, the cancellation of queue.put() can cause a memory leak. Patch by: José Melero.
- bpo-32521 [https://bugs.python.org/issue?@action=redirect&bpo=32521]: The nis module is now compatible with new libnsl and headers location.
- bpo-32467 [https://bugs.python.org/issue?@action=redirect&bpo=32467]: collections.abc.ValuesView now inherits from collections.abc.Collection.
- bpo-32473 [https://bugs.python.org/issue?@action=redirect&bpo=32473]: Improve ABCMeta._dump_registry() output readability
- bpo-32102 [https://bugs.python.org/issue?@action=redirect&bpo=32102]: New argument `capture_output` for subprocess.run
- bpo-32521 [https://bugs.python.org/issue?@action=redirect&bpo=32521]: glibc has removed Sun RPC. Use replacement libtirpc headers and library in nis module.
- bpo-32493 [https://bugs.python.org/issue?

@action=redirect&bpo=32493]: UUID module fixes build for FreeBSD/OpenBSD

- bpo-32503 [https://bugs.python.org/issue?@action=redirect&bpo=32503]: Pickling with protocol 4 no longer creates too small frames.
- bpo-29237 [https://bugs.python.org/issue?@action=redirect&bpo=29237]: Create enum for pstats sorting options
- bpo-32454 [https://bugs.python.org/issue?@action=redirect&bpo=32454]: Add close(fd) function to the socket module.
- bpo-25942 [https://bugs.python.org/issue?@action=redirect&bpo=25942]: The subprocess module is now more graceful when handling a Ctrl-C KeyboardInterrupt during subprocess.call, subprocess.run, or a Popen context manager. It now waits a short amount of time for the child (presumed to have also gotten the SIGINT) to exit, before continuing the KeyboardInterrupt exception handling. This still includes a SIGKILL in the call() and run() APIs, but at least the child had a chance first.
- bpo-32433 [https://bugs.python.org/issue?@action=redirect&bpo=32433]: The hmac module now has hmac.digest(), which provides an optimized HMAC digest.
- bpo-28134 [https://bugs.python.org/issue?@action=redirect&bpo=28134]: Sockets now auto-detect family, type and protocol from file descriptor by default.
- bpo-32404 [https://bugs.python.org/issue?@action=redirect&bpo=32404]: Fix bug where **`datetime.datetime.fromtimestamp()`** did not call __new__ in **`datetime.datetime`** subclasses.
- bpo-32403 [https://bugs.python.org/issue?@action=redirect&bpo=32403]: Improved speed of **`datetime.date`** and **`datetime.datetime`** alternate constructors.
- bpo-32228 [https://bugs.python.org/issue?@action=redirect&bpo=32228]: Ensure that `truncate()` preserves the file position (as reported by `tell()`) after writes longer than the buffer size.
- bpo-32410 [https://bugs.python.org/issue?@action=redirect&bpo=32410]: Implement

`loop.sock_sendfile` for asyncio event loop.

- bpo-22908 [https://bugs.python.org/issue?@action=redirect&bpo=22908]: Added seek and tell to the ZipExtFile class. This only works if the file object used to open the zipfile is seekable.
- bpo-32373 [https://bugs.python.org/issue?@action=redirect&bpo=32373]: Add socket.getblocking() method.
- bpo-32248 [https://bugs.python.org/issue?@action=redirect&bpo=32248]: Add **`importlib.resources`** and **`importlib.abc.ResourceReader`** as the unified API for reading resources contained within packages. Loaders wishing to support resource reading must implement the **`get_resource_reader()`** method. File-based and zipimport-based loaders both implement these APIs. **`importlib.abc.ResourceLoader`** is deprecated in favor of these new APIs.
- bpo-32320 [https://bugs.python.org/issue?@action=redirect&bpo=32320]: collections.namedtuple() now supports default values.
- bpo-29302 [https://bugs.python.org/issue?@action=redirect&bpo=29302]: Add contextlib.AsyncExitStack. Patch by Alexander Mohr and Ilya Kulakov.
- bpo-31961 [https://bugs.python.org/issue?@action=redirect&bpo=31961]: *Removed in Python 3.7.0b2.* The *args* argument of subprocess.Popen can now be a path-like object. If *args* is given as a sequence, it's first element can now be a path-like object as well.
- bpo-31900 [https://bugs.python.org/issue?@action=redirect&bpo=31900]: The **`locale.localeconv()`** function now sets temporarily the `LC_CTYPE` locale to the `LC_NUMERIC` locale to decode `decimal_point` and `thousands_sep` byte strings if they are non-ASCII or longer than 1 byte, and the `LC_NUMERIC` locale is different than the `LC_CTYPE` locale. This temporary change affects other threads. Same change for the **`str.format()`** method when formatting a number (**`int`**, **`float`**, **`float`** and subclasses) with the `n` type (ex: `'{:n}'.format(1234)`).
- bpo-31853 [https://bugs.python.org/issue?@action=redirect&bpo=31853]: Use super().method instead of socket.method in SSLSocket. They were there most likely for

legacy reasons.

- bpo-31399 [https://bugs.python.org/issue?@action=redirect&bpo=31399]: The ssl module now uses OpenSSL's X509_VERIFY_PARAM_set1_host() and X509_VERIFY_PARAM_set1_ip() API to verify hostname and IP addresses. Subject common name fallback can be disabled with SSLContext.hostname_checks_common_name.
- bpo-14976 [https://bugs.python.org/issue?@action=redirect&bpo=14976]: Add a queue.SimpleQueue class, an unbounded FIFO queue with a reentrant C implementation of put().

Documentation

- bpo-32724 [https://bugs.python.org/issue?@action=redirect&bpo=32724]: Add references to some commands in the documentation of Pdb. Patch by Stéphane Wirtel
- bpo-32649 [https://bugs.python.org/issue?@action=redirect&bpo=32649]: Complete the C API documentation, profiling and tracing part with the newly added per-opcode events.
- bpo-17799 [https://bugs.python.org/issue?@action=redirect&bpo=17799]: Explain real behaviour of sys.settrace and sys.setprofile and their C-API counterparts regarding which type of events are received in each function. Patch by Pablo Galindo Salgado.

Tests

- bpo-32721 [https://bugs.python.org/issue?@action=redirect&bpo=32721]: Fix test_hashlib to not fail if the _md5 module is not built.
- bpo-28414 [https://bugs.python.org/issue?@action=redirect&bpo=28414]: Add test cases for IDNA 2003 and 2008 host names. IDNA 2003 internationalized host names are working since bpo-31399 [https://bugs.python.org/issue?@action=redirect&bpo=31399] has landed. IDNA 2008 are still broken.
- bpo-32604 [https://bugs.python.org/issue?

@action=redirect&bpo=32604]: Add a new «_xxsubinterpreters» extension module that exposes the existing subinterpreter C-API and a new cross-interpreter data sharing mechanism. The module is primarily intended for more thorough testing of the existing subinterpreter support. Note that the _xxsubinterpreters module has been removed in 3.7.0rc1.
- bpo-32602 [https://bugs.python.org/issue?@action=redirect&bpo=32602]: Add test certs and test for ECDSA cert and EC/RSA dual mode.
- bpo-32549 [https://bugs.python.org/issue?@action=redirect&bpo=32549]: On Travis CI, Python now Compiles and uses a local copy of OpenSSL 1.1.0g for testing.

Build

- bpo-32635 [https://bugs.python.org/issue?@action=redirect&bpo=32635]: Fix segfault of the crypt module when libxcrypt is provided instead of libcrypt at the system.
- bpo-32598 [https://bugs.python.org/issue?@action=redirect&bpo=32598]: Use autoconf to detect OpenSSL libs, headers and supported features. The ax_check_openssl M4 macro uses pkg-config to locate OpenSSL and falls back to manual search.
- bpo-32593 [https://bugs.python.org/issue?@action=redirect&bpo=32593]: Drop support of FreeBSD 9 and older.
- bpo-29708 [https://bugs.python.org/issue?@action=redirect&bpo=29708]: If the **`SOURCE_DATE_EPOCH`** environment variable is set, **`py_compile`** will always create hash-based `.pyc` files.

Windows

- bpo-32588 [https://bugs.python.org/issue?@action=redirect&bpo=32588]: Create standalone _distutils_findvs module and add missing _queue module to installer.
- bpo-29911 [https://bugs.python.org/issue?@action=redirect&bpo=29911]: Ensure separate Modify and Uninstall buttons are displayed.
- bpo-32507 [https://bugs.python.org/issue?

@action=redirect&bpo=32507]: Use app-local UCRT install rather than the proper update for old versions of Windows.

macOS

- bpo-32726 [https://bugs.python.org/issue?@action=redirect&bpo=32726]: Provide an additional, more modern macOS installer variant that supports macOS 10.9+ systems in 64-bit mode only. Upgrade the supplied third-party libraries to OpenSSL 1.1.0g and to SQLite 3.22.0. The 10.9+ installer now links with and supplies its own copy of Tcl/Tk 8.6.
- bpo-28440 [https://bugs.python.org/issue?@action=redirect&bpo=28440]: No longer add /Library/Python/3.x/site-packages to sys.path for macOS framework builds to avoid future conflicts.

C API

- bpo-32681 [https://bugs.python.org/issue?@action=redirect&bpo=32681]: Fix uninitialized variable “res” in the C implementation of os.dup2. Patch by Stéphane Wirtel
- bpo-10381 [https://bugs.python.org/issue?@action=redirect&bpo=10381]: Add C API access to the `datetime.timezone` constructor and `datetime.timezone.UTC` singleton.

Python 3.7.0 alpha 4

Release date: 2018-01-08

Core and Builtins

- bpo-31975 [https://bugs.python.org/issue?@action=redirect&bpo=31975]: The default warning filter list now starts with a «default::DeprecationWarning:__main__» entry, so deprecation warnings are once again shown by default in single-file scripts and at the interactive prompt.
- bpo-32226 [https://bugs.python.org/issue?

@action=redirect&bpo=32226]: `__class_getitem__` is now an automatic class method.

- bpo-32399 [https://bugs.python.org/issue?@action=redirect&bpo=32399]: Add AIX uuid library support for RFC4122 using uuid_create() in libc.a
- bpo-32390 [https://bugs.python.org/issue?@action=redirect&bpo=32390]: Fix the compilation failure on AIX after the f_fsid field has been added to the object returned by os.statvfs() (bpo-32143 [https://bugs.python.org/issue?@action=redirect&bpo=32143]). Original patch by Michael Felt.
- bpo-32379 [https://bugs.python.org/issue?@action=redirect&bpo=32379]: Make MRO computation faster when a class inherits from a single base.
- bpo-32259 [https://bugs.python.org/issue?@action=redirect&bpo=32259]: The error message of a TypeError raised when unpack non-iterable is now more specific.
- bpo-27169 [https://bugs.python.org/issue?@action=redirect&bpo=27169]: The `__debug__` constant is now optimized out at compile time. This fixes also bpo-22091 [https://bugs.python.org/issue?@action=redirect&bpo=22091].
- bpo-32329 [https://bugs.python.org/issue?@action=redirect&bpo=32329]: The **`-R`** option now turns on hash randomization when the **`PYTHONHASHSEED`** environment variable is set to `0`. Previously, the option was ignored. Moreover, `sys.flags.hash_randomization` is now properly set to 0 when hash randomization is turned off by `PYTHONHASHSEED=0`.
- bpo-30416 [https://bugs.python.org/issue?@action=redirect&bpo=30416]: The optimizer is now protected from spending much time doing complex calculations and consuming much memory for creating large constants in constant folding. Increased limits for constants that can be produced in constant folding.
- bpo-32282 [https://bugs.python.org/issue?@action=redirect&bpo=32282]: Fix an unnecessary ifdef in the include of VersionHelpers.h in socketmodule on Windows.
- bpo-30579 [https://bugs.python.org/issue?@action=redirect&bpo=30579]: Implement TracebackType.__new__ to allow Python-level creation of traceback objects, and make TracebackType.tb_next mutable.

- bpo-32260 [https://bugs.python.org/issue?@action=redirect&bpo=32260]: Don't byte swap the input keys to the SipHash algorithm on big-endian platforms. This should ensure siphash gives consistent results across platforms.
- bpo-31506 [https://bugs.python.org/issue?@action=redirect&bpo=31506]: Improve the error message logic for object._new_ and object._init_. Patch by Sanyam Khurana.
- bpo-20361 [https://bugs.python.org/issue?@action=redirect&bpo=20361]: `-b` and `-bb` now inject `'default::BytesWarning'` and `error::BytesWarning` entries into `sys.warnoptions`, ensuring that they take precedence over any other warning filters configured via the `-W` option or the `PYTHONWARNINGS` environment variable.
- bpo-32230 [https://bugs.python.org/issue?@action=redirect&bpo=32230]: `-X dev` now injects a `'default'` entry into sys.warnoptions, ensuring that it behaves identically to actually passing `-Wdefault` at the command line.
- bpo-29240 [https://bugs.python.org/issue?@action=redirect&bpo=29240]: Add a new UTF-8 mode: implementation of the **PEP 540** [https://peps.python.org/pep-0540/].
- bpo-32226 [https://bugs.python.org/issue?@action=redirect&bpo=32226]: **PEP 560** [https://peps.python.org/pep-0560/]: Add support for `__mro_entries__` and `__class_getitem__`. Implemented by Ivan Levkivskyi.
- bpo-32225 [https://bugs.python.org/issue?@action=redirect&bpo=32225]: **PEP 562** [https://peps.python.org/pep-0562/]: Add support for module `__getattr__` and `__dir__`. Implemented by Ivan Levkivskyi.
- bpo-31901 [https://bugs.python.org/issue?@action=redirect&bpo=31901]: The `atexit` module now has its callback stored per interpreter.
- bpo-31650 [https://bugs.python.org/issue?@action=redirect&bpo=31650]: Implement **PEP 552** [https://peps.python.org/pep-0552/] (Deterministic pycs). Python now supports invalidating bytecode cache files bashed on a source content hash rather than source last-modified time.

- bpo-29469 [https://bugs.python.org/issue?@action=redirect&bpo=29469]: Move constant folding from bytecode layer to AST layer. Original patch by Eugene Toder.

Library

- bpo-32506 [https://bugs.python.org/issue?@action=redirect&bpo=32506]: Now that dict is defined as keeping insertion order, drop OrderedDict and just use plain dict.
- bpo-32279 [https://bugs.python.org/issue?@action=redirect&bpo=32279]: Add params to dataclasses.make_dataclasses(): init, repr, eq, order, hash, and frozen. Pass them through to dataclass().
- bpo-32278 [https://bugs.python.org/issue?@action=redirect&bpo=32278]: Make type information optional on dataclasses.make_dataclass(). If omitted, the string “typing.Any” is used.
- bpo-32499 [https://bugs.python.org/issue?@action=redirect&bpo=32499]: Add dataclasses.is_dataclass(obj), which returns True if obj is a dataclass or an instance of one.
- bpo-32468 [https://bugs.python.org/issue?@action=redirect&bpo=32468]: Improve frame repr() to mention filename, code name and current line number.
- bpo-23749 [https://bugs.python.org/issue?@action=redirect&bpo=23749]: asyncio: Implement loop.start_tls()
- bpo-32441 [https://bugs.python.org/issue?@action=redirect&bpo=32441]: Return the new file descriptor (i.e., the second argument) from `os.dup2`. Previously, `None` was always returned.
- bpo-32422 [https://bugs.python.org/issue?@action=redirect&bpo=32422]: `functools.lru_cache` uses less memory (3 words for each cached key) and takes about 1/3 time for cyclic GC.
- bpo-31721 [https://bugs.python.org/issue?@action=redirect&bpo=31721]: Prevent Python crash from happening when Future._log_traceback is set to True manually. Now it can only be set to False, or a ValueError is raised.
- bpo-32415 [https://bugs.python.org/issue?

@action=redirect&bpo=32415]: asyncio: Add Task.get_loop() and Future.get_loop()

- bpo-26133 [https://bugs.python.org/issue?@action=redirect&bpo=26133]: Don’t unsubscribe signals in asyncio UNIX event loop on interpreter shutdown.
- bpo-32363 [https://bugs.python.org/issue?@action=redirect&bpo=32363]: Make asyncio.Task.set_exception() and set_result() raise NotImplementedError. Task._step() and Future.__await__() raise proper exceptions when they are in an invalid state, instead of raising an AssertionError.
- bpo-32357 [https://bugs.python.org/issue?@action=redirect&bpo=32357]: Optimize asyncio.iscoroutine() and loop.create_task() for non-native coroutines (e.g. async/await compiled with Cython).
  “loop.create_task(python_coroutine)” used to be 20% faster than “loop.create_task(cython_coroutine)”. Now, the latter is as fast.
- bpo-32356 [https://bugs.python.org/issue?@action=redirect&bpo=32356]: asyncio.transport.resume_reading() and pause_reading() are now idempotent. New transport.is_reading() method is added.
- bpo-32355 [https://bugs.python.org/issue?@action=redirect&bpo=32355]: Optimize asyncio.gather(); now up to 15% faster.
- bpo-32351 [https://bugs.python.org/issue?@action=redirect&bpo=32351]: Use fastpath in asyncio.sleep if delay<0 (2x boost)
- bpo-32348 [https://bugs.python.org/issue?@action=redirect&bpo=32348]: Optimize asyncio.Future schedule/add/remove callback. The optimization shows 3-6% performance improvements of async/await code.
- bpo-32331 [https://bugs.python.org/issue?@action=redirect&bpo=32331]: Fix socket.settimeout() and socket.setblocking() to keep socket.type as is. Fix socket.socket() constructor to reset any bit flags applied to socket’s type. This change only affects OSes that have SOCK_NONBLOCK and/or SOCK_CLOEXEC.
- bpo-32248 [https://bugs.python.org/issue?@action=redirect&bpo=32248]: Add **`importlib.abc.ResourceReader`** as an ABC for loaders

to provide a unified API for reading resources contained within packages. Also add `importlib.resources` as the port of `importlib_resources`.

- bpo-32311 [https://bugs.python.org/issue?@action=redirect&bpo=32311]: Implement asyncio.create_task(coro) shortcut
- bpo-32327 [https://bugs.python.org/issue?@action=redirect&bpo=32327]: Convert asyncio functions that were documented as coroutines to coroutines. Affected functions: loop.sock_sendall, loop.sock_recv, loop.sock_accept, loop.getaddrinfo, loop.getnameinfo.
- bpo-32323 [https://bugs.python.org/issue?@action=redirect&bpo=32323]: `urllib.parse.urlsplit()` does not convert zone-id (scope) to lower case for scoped IPv6 addresses in hostnames now.
- bpo-32302 [https://bugs.python.org/issue?@action=redirect&bpo=32302]: Fix bdist_wininst of distutils for CRT v142: it binary compatible with CRT v140.
- bpo-29711 [https://bugs.python.org/issue?@action=redirect&bpo=29711]: Fix `stop_serving` in asyncio proactor loop kill all listening servers
- bpo-32308 [https://bugs.python.org/issue?@action=redirect&bpo=32308]: `re.sub()` now replaces empty matches adjacent to a previous non-empty match.
- bpo-29970 [https://bugs.python.org/issue?@action=redirect&bpo=29970]: Abort asyncio SSLProtocol connection if handshake not complete within 10 seconds.
- bpo-32314 [https://bugs.python.org/issue?@action=redirect&bpo=32314]: Implement asyncio.run().
- bpo-17852 [https://bugs.python.org/issue?@action=redirect&bpo=17852]: Revert incorrect fix based on misunderstanding of _Py_PyAtExit() semantics.
- bpo-32296 [https://bugs.python.org/issue?@action=redirect&bpo=32296]: Implement asyncio._get_running_loop() and get_event_loop() in C. This makes them 4x faster.
- bpo-32250 [https://bugs.python.org/issue?@action=redirect&bpo=32250]: Implement `asyncio.current_task()` and `asyncio.all_tasks()`. Add helpers intended to be used by alternative task

implementations: `asyncio._register_task`, `asyncio._enter_task`, `asyncio._leave_task` and `asyncio._unregister_task`. Deprecate `asyncio.Task.current_task()` and `asyncio.Task.all_tasks()`.

- bpo-32255 [https://bugs.python.org/issue?@action=redirect&bpo=32255]: A single empty field is now always quoted when written into a CSV file. This allows to distinguish an empty row from a row consisting of a single empty field. Patch by Licht Takeuchi.
- bpo-32277 [https://bugs.python.org/issue?@action=redirect&bpo=32277]: Raise `NotImplementedError` instead of `SystemError` on platforms where `chmod(..., follow_symlinks=False)` is not supported. Patch by Anthony Sottile.
- bpo-30050 [https://bugs.python.org/issue?@action=redirect&bpo=30050]: New argument warn_on_full_buffer to signal.set_wakeup_fd lets you control whether Python prints a warning on stderr when the wakeup fd buffer overflows.
- bpo-29137 [https://bugs.python.org/issue?@action=redirect&bpo=29137]: The `fpectl` library has been removed. It was never enabled by default, never worked correctly on x86-64, and it changed the Python ABI in ways that caused unexpected breakage of C extensions.
- bpo-32273 [https://bugs.python.org/issue?@action=redirect&bpo=32273]: Move asyncio.test_utils to test.test_asyncio.
- bpo-32272 [https://bugs.python.org/issue?@action=redirect&bpo=32272]: Remove asyncio.async() function.
- bpo-32269 [https://bugs.python.org/issue?@action=redirect&bpo=32269]: Add asyncio.get_running_loop() function.
- bpo-32265 [https://bugs.python.org/issue?@action=redirect&bpo=32265]: All class and static methods of builtin types now are correctly classified by inspect.classify_class_attrs() and grouped in pydoc output. Added types.ClassMethodDescriptorType for unbound class methods of builtin types.
- bpo-32253 [https://bugs.python.org/issue?

@action=redirect&bpo=32253]: Deprecate `yield from lock`, `await lock`, `with (yield from lock)` and `with await lock` for asyncio synchronization primitives.

- bpo-22589 [https://bugs.python.org/issue?@action=redirect&bpo=22589]: Changed MIME type of .bmp from “image/x-ms-bmp” to “image/bmp”
- bpo-32193 [https://bugs.python.org/issue?@action=redirect&bpo=32193]: Convert asyncio to use *async/await* syntax. Old styled `yield from` is still supported too.
- bpo-32206 [https://bugs.python.org/issue?@action=redirect&bpo=32206]: Add support to run modules with pdb
- bpo-32227 [https://bugs.python.org/issue?@action=redirect&bpo=32227]: `functools.singledispatch` now supports registering implementations using type annotations.
- bpo-15873 [https://bugs.python.org/issue?@action=redirect&bpo=15873]: Added new alternate constructors **`datetime.datetime.fromisoformat()`**, **`datetime.time.fromisoformat()`** and **`datetime.date.fromisoformat()`** as the inverse operation of each classes’s respective `isoformat` methods.
- bpo-32199 [https://bugs.python.org/issue?@action=redirect&bpo=32199]: The getnode() ip getter now uses “ip link” instead of “ip link list”.
- bpo-32143 [https://bugs.python.org/issue?@action=redirect&bpo=32143]: os.statvfs() includes the f_fsid field from statvfs(2)
- bpo-26439 [https://bugs.python.org/issue?@action=redirect&bpo=26439]: Fix ctypes.util.find_library() for AIX by implementing ctypes._aix.find_library() Patch by: Michael Felt
- bpo-31993 [https://bugs.python.org/issue?@action=redirect&bpo=31993]: The pickler now uses less memory when serializing large bytes and str objects into a file. Pickles created with protocol 4 will require less memory for unpickling large bytes and str objects.
- bpo-27456 [https://bugs.python.org/issue?@action=redirect&bpo=27456]: Ensure TCP_NODELAY is set on Linux. Tests by Victor Stinner.

- bpo-31778 [https://bugs.python.org/issue?@action=redirect&bpo=31778]: ast.literal_eval() is now more strict. Addition and subtraction of arbitrary numbers no longer allowed.
- bpo-31802 [https://bugs.python.org/issue?@action=redirect&bpo=31802]: Importing native path module (`posixpath`, `ntpath`) now works even if the `os` module still is not imported.
- bpo-30241 [https://bugs.python.org/issue?@action=redirect&bpo=30241]: Add contextlib.AbstractAsyncContextManager. Patch by Jelle Zijlstra.
- bpo-31699 [https://bugs.python.org/issue?@action=redirect&bpo=31699]: Fix deadlocks in **`concurrent.futures.ProcessPoolExecutor`** when task arguments or results cause pickling or unpickling errors. This should make sure that calls to the **`ProcessPoolExecutor`** API always eventually return.
- bpo-15216 [https://bugs.python.org/issue?@action=redirect&bpo=15216]: `TextIOWrapper.reconfigure()` supports changing *encoding*, *errors*, and *newline*.

Documentation

- bpo-32418 [https://bugs.python.org/issue?@action=redirect&bpo=32418]: Add get_loop() method to Server and AbstractServer classes.

Tests

- bpo-32252 [https://bugs.python.org/issue?@action=redirect&bpo=32252]: Fix faulthandler_suppress_crash_report() used to prevent core dump files when testing crashes. getrlimit() returns zero on success.
- bpo-32002 [https://bugs.python.org/issue?@action=redirect&bpo=32002]: Adjust C locale coercion testing for the empty locale and POSIX locale cases to more readily adjust to platform dependent behaviour.

Windows

- bpo-19764 [https://bugs.python.org/issue?@action=redirect&bpo=19764]: Implement support for `subprocess.Popen(close_fds=True)` on Windows. Patch by Segev Finer.

Tools/Demos

- bpo-24960 [https://bugs.python.org/issue?@action=redirect&bpo=24960]: 2to3 and lib2to3 can now read pickled grammar files using pkgutil.get_data() rather than probing the filesystem. This lets 2to3 and lib2to3 work when run from a zipfile.

C API

- bpo-32030 [https://bugs.python.org/issue?@action=redirect&bpo=32030]: Py_Initialize() doesn't reset the memory allocators to default if the `PYTHONMALLOC` environment variable is not set.
- bpo-29084 [https://bugs.python.org/issue?@action=redirect&bpo=29084]: Undocumented C API for OrderedDict has been excluded from the limited C API. It was added by mistake and actually never worked in the limited C API.
- bpo-32264 [https://bugs.python.org/issue?@action=redirect&bpo=32264]: Moved the pygetopt.h header into internal/, since it has no public APIs.
- bpo-32241 [https://bugs.python.org/issue?@action=redirect&bpo=32241]: **`Py_SetProgramName()`** and **`Py_SetPythonHome()`** now take the `const wchar *` arguments instead of `wchar *`.

Python 3.7.0 alpha 3

Release date: 2017-12-05

Core and Builtins

- bpo-32176 [https://bugs.python.org/issue?@action=redirect&bpo=32176]: co_flags.CO_NOFREE is now always set correctly by the code object constructor based on freevars and cellvars, rather than needing to be set correctly by the caller. This ensures it will be cleared automatically when additional cell references are injected into a modified code object and function.
- bpo-10544 [https://bugs.python.org/issue?@action=redirect&bpo=10544]: Yield expressions are now deprecated in comprehensions and generator expressions. They are still permitted in the definition of the outermost iterable, as that is evaluated directly in the enclosing scope.
- bpo-32137 [https://bugs.python.org/issue?@action=redirect&bpo=32137]: The repr of deeply nested dict now raises a RecursionError instead of crashing due to a stack overflow.
- bpo-32096 [https://bugs.python.org/issue?@action=redirect&bpo=32096]: Revert memory allocator changes in the C API: move structures back from _PyRuntime to Objects/obmalloc.c. The memory allocators are once again initialized statically, and so PyMem_RawMalloc() and Py_DecodeLocale() can be called before _PyRuntime_Initialize().
- bpo-32043 [https://bugs.python.org/issue?@action=redirect&bpo=32043]: Add a new «developer mode»: new «-X dev» command line option to enable debug checks at runtime.
- bpo-32023 [https://bugs.python.org/issue?@action=redirect&bpo=32023]: SyntaxError is now correctly raised when a generator expression without parenthesis is used instead of an inheritance list in a class definition. The duplication of the parentheses can be omitted only on calls.
- bpo-32012 [https://bugs.python.org/issue?@action=redirect&bpo=32012]: SyntaxError is now correctly raised when a generator expression without parenthesis is passed as an argument, but followed by a trailing comma. A generator expression always needs to be directly inside a set of parentheses and cannot have a comma on either side.
- bpo-28180 [https://bugs.python.org/issue?@action=redirect&bpo=28180]: A new internal

`_Py_SetLocaleFromEnv(category)` helper function has been added in order to improve the consistency of behaviour across different `libc` implementations (e.g. Android doesn't support setting the locale from the environment by default).

- bpo-31949 [https://bugs.python.org/issue?@action=redirect&bpo=31949]: Fixed several issues in printing tracebacks (PyTraceBack_Print()). Setting sys.tracebacklimit to 0 or less now suppresses printing tracebacks. Setting sys.tracebacklimit to None now causes using the default limit. Setting sys.tracebacklimit to an integer larger than LONG_MAX now means using the limit LONG_MAX rather than the default limit. Fixed integer overflows in the case of more than `2**31` traceback items on Windows. Fixed output errors handling.
- bpo-30696 [https://bugs.python.org/issue?@action=redirect&bpo=30696]: Fix the interactive interpreter looping endlessly when no memory.
- bpo-20047 [https://bugs.python.org/issue?@action=redirect&bpo=20047]: Bytearray methods partition() and rpartition() now accept only bytes-like objects as separator, as documented. In particular they now raise TypeError rather of returning a bogus result when an integer is passed as a separator.
- bpo-21720 [https://bugs.python.org/issue?@action=redirect&bpo=21720]: BytesWarning no longer emitted when the *fromlist* argument of `__import__()` or the `__all__` attribute of the module contain bytes instances.
- bpo-31845 [https://bugs.python.org/issue?@action=redirect&bpo=31845]: Environment variables are once more read correctly at interpreter startup.
- bpo-28936 [https://bugs.python.org/issue?@action=redirect&bpo=28936]: Ensure that lexically first syntax error involving a parameter and `global` or `nonlocal` is detected first at a given scope. Patch by Ivan Levkivskyi.
- bpo-31825 [https://bugs.python.org/issue?@action=redirect&bpo=31825]: Fixed OverflowError in the "unicode-escape" codec and in codecs.escape_decode() when decode an escaped non-ascii byte.
- bpo-31618 [https://bugs.python.org/issue?@action=redirect&bpo=31618]: The per-frame tracing logic added

in 3.7a1 has been altered so that `frame->f_lineno` is updated before either `"line"` or `"opcode"` events are emitted. Previously, opcode events were emitted first, and therefore would occasionally see stale line numbers on the frame. The behavior of this feature has changed slightly as a result: when both `f_trace_lines` and `f_trace_opcodes` are enabled, line events now occur first.

- bpo-28603 [https://bugs.python.org/issue?@action=redirect&bpo=28603]: Print the full context/cause chain of exceptions on interpreter exit, even if an exception in the chain is unhashable or compares equal to later ones. Patch by Zane Bitter.
- bpo-31786 [https://bugs.python.org/issue?@action=redirect&bpo=31786]: Fix timeout rounding in the select module to round correctly negative timeouts between -1.0 and 0.0. The functions now block waiting for events as expected. Previously, the call was incorrectly non-blocking. Patch by Pablo Galindo.
- bpo-31781 [https://bugs.python.org/issue?@action=redirect&bpo=31781]: Prevent crashes when calling methods of an uninitialized `zipimport.zipimporter` object. Patch by Oren Milman.
- bpo-30399 [https://bugs.python.org/issue?@action=redirect&bpo=30399]: Standard repr() of BaseException with a single argument no longer contains redundant trailing comma.
- bpo-31626 [https://bugs.python.org/issue?@action=redirect&bpo=31626]: Fixed a bug in debug memory allocator. There was a write to freed memory after shrinking a memory block.
- bpo-30817 [https://bugs.python.org/issue?@action=redirect&bpo=30817]: `PyErr_PrintEx()` clears now the ignored exception that may be raised by `_PySys_SetObjectId()`, for example when no memory.

Library

- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Two minor fixes for `typing` module: allow shallow copying instances of generic classes, improve interaction of `__init_subclass__` with generics. Original PRs by Ivan Levkivskyi.

- bpo-32214 [https://bugs.python.org/issue?@action=redirect&bpo=32214]: PEP 557, Data Classes. Provides a decorator which adds boilerplate methods to classes which use type annotations so specify fields.
- bpo-27240 [https://bugs.python.org/issue?@action=redirect&bpo=27240]: The header folding algorithm for the new email policies has been rewritten, which also fixes bpo-30788 [https://bugs.python.org/issue?@action=redirect&bpo=30788], bpo-31831 [https://bugs.python.org/issue?@action=redirect&bpo=31831], and bpo-32182 [https://bugs.python.org/issue?@action=redirect&bpo=32182]. In particular, RFC2231 folding is now done correctly.
- bpo-32186 [https://bugs.python.org/issue?@action=redirect&bpo=32186]: io.FileIO.readall() and io.FileIO.read() now release the GIL when getting the file size. Fixed hang of all threads with inaccessible NFS server. Patch by Nir Soffer.
- bpo-32101 [https://bugs.python.org/issue?@action=redirect&bpo=32101]: Add **`sys.flags.dev_mode`** flag
- bpo-32154 [https://bugs.python.org/issue?@action=redirect&bpo=32154]: The `asyncio.windows_utils.socketpair()` function has been removed: use directly **`socket.socketpair()`** which is available on all platforms since Python 3.5 (before, it wasn’t available on Windows). `asyncio.windows_utils.socketpair()` was just an alias to `socket.socketpair` on Python 3.5 and newer.
- bpo-32089 [https://bugs.python.org/issue?@action=redirect&bpo=32089]: warnings: In development (-X dev) and debug mode (pydebug build), use the «default» action for ResourceWarning, rather than the «always» action, in the default warnings filters.
- bpo-32107 [https://bugs.python.org/issue?@action=redirect&bpo=32107]: `uuid.getnode()` now preferentially returns universally administered MAC addresses if available, over locally administered MAC addresses. This makes a better guarantee for global uniqueness of UUIDs returned from `uuid.uuid1()`. If only locally administered

MAC addresses are available, the first such one found is returned.

- bpo-23033 [https://bugs.python.org/issue?@action=redirect&bpo=23033]: Wildcard is now supported in hostname when it is one and only character in the left most segment of hostname in second argument of **`ssl.match_hostname()`**. Patch by Mandeep Singh.
- bpo-12239 [https://bugs.python.org/issue?@action=redirect&bpo=12239]: Make **`msilib.SummaryInformation.GetProperty()`** return `None` when the value of property is `VT_EMPTY`. Initial patch by Mark Mc Mahon.
- bpo-28334 [https://bugs.python.org/issue?@action=redirect&bpo=28334]: Use **`os.path.expanduser()`** to find the `~/.netrc` file in **`netrc.netrc`**. If it does not exist, **`FileNotFoundError`** is raised. Patch by Dimitri Merejkowsky.
- bpo-32121 [https://bugs.python.org/issue?@action=redirect&bpo=32121]: Made `tracemalloc.Traceback` behave more like the traceback module, sorting the frames from oldest to most recent. `Traceback.format()` now accepts negative *limit*, truncating the result to the `abs(limit)` oldest frames. To get the old behaviour, one can use the new *most_recent_first* argument to `Traceback.format()`. (Patch by Jesse Bakker.)
- bpo-31325 [https://bugs.python.org/issue?@action=redirect&bpo=31325]: Fix wrong usage of **`collections.namedtuple()`** in the **`RobotFileParser.parse()`** method. Initial patch by Robin Wellner.
- bpo-12382 [https://bugs.python.org/issue?@action=redirect&bpo=12382]: **`msilib.OpenDatabase()`** now raises a better exception message when it couldn't open or create an MSI file. Initial patch by William Tisäter.
- bpo-19610 [https://bugs.python.org/issue?@action=redirect&bpo=19610]: `setup()` now warns about invalid types for some fields. The `distutils.dist.Distribution` class now warns when `classifiers`, `keywords` and `platforms` fields are not

specified as a list or a string.

- bpo-32071 [https://bugs.python.org/issue?@action=redirect&bpo=32071]: Added the `-k` command-line option to `python -m unittest` to run only tests that match the given pattern(s).
- bpo-10049 [https://bugs.python.org/issue?@action=redirect&bpo=10049]: Added *nullcontext* no-op context manager to contextlib. This provides a simpler and faster alternative to ExitStack() when handling optional context managers.
- bpo-28684 [https://bugs.python.org/issue?@action=redirect&bpo=28684]: The new test.support.skip_unless_bind_unix_socket() decorator is used here to skip asyncio tests that fail because the platform lacks a functional bind() function for unix domain sockets (as it is the case for non root users on the recent Android versions that run now SELinux in enforcing mode).
- bpo-32110 [https://bugs.python.org/issue?@action=redirect&bpo=32110]: `codecs.StreamReader.read(n)` now returns not more than *n* characters/bytes for non-negative *n*. This makes it compatible with `read()` methods of other file-like objects.
- bpo-27535 [https://bugs.python.org/issue?@action=redirect&bpo=27535]: The warnings module doesn’t leak memory anymore in the hidden warnings registry for the «ignore» action of warnings filters. warn_explicit() function doesn’t add the warning key to the registry anymore for the «ignore» action.
- bpo-32088 [https://bugs.python.org/issue?@action=redirect&bpo=32088]: warnings: When Python is build is debug mode (`Py_DEBUG`), **`DeprecationWarning`**, **`PendingDeprecationWarning`** and **`ImportWarning`** warnings are now displayed by default.
- bpo-1647489 [https://bugs.python.org/issue?@action=redirect&bpo=1647489]: Fixed searching regular expression patterns that could match an empty string. Non-empty string can now be correctly found after matching an empty string.
- bpo-25054 [https://bugs.python.org/issue?@action=redirect&bpo=25054]: Added support of splitting on a

pattern that could match an empty string.

- bpo-32072 [https://bugs.python.org/issue?@action=redirect&bpo=32072]: Fixed issues with binary plists: Fixed saving bytearrays. Identical objects will be saved only once. Equal references will be load as identical objects. Added support for saving and loading recursive data structures.
- bpo-32069 [https://bugs.python.org/issue?@action=redirect&bpo=32069]: Drop legacy SSL transport from asyncio, ssl.MemoryBIO is always used anyway.
- bpo-32066 [https://bugs.python.org/issue?@action=redirect&bpo=32066]: asyncio: Support pathlib.Path in create_unix_connection; sock arg should be optional
- bpo-32046 [https://bugs.python.org/issue?@action=redirect&bpo=32046]: Updates 2to3 to convert from operator.isCallable(obj) to callable(obj). Patch by Donghee Na.
- bpo-32018 [https://bugs.python.org/issue?@action=redirect&bpo=32018]: inspect.signature should follow **PEP 8** [https://peps.python.org/pep-0008/], if the parameter has an annotation and a default value. Patch by Donghee Na.
- bpo-32025 [https://bugs.python.org/issue?@action=redirect&bpo=32025]: Add time.thread_time() and time.thread_time_ns()
- bpo-32037 [https://bugs.python.org/issue?@action=redirect&bpo=32037]: Integers that fit in a signed 32-bit integer will be now pickled with protocol 0 using the INT opcode. This will decrease the size of a pickle, speed up pickling and unpickling, and make these integers be unpickled as int instances in Python 2.
- bpo-32034 [https://bugs.python.org/issue?@action=redirect&bpo=32034]: Make asyncio.IncompleteReadError and LimitOverrunError pickleable.
- bpo-32015 [https://bugs.python.org/issue?@action=redirect&bpo=32015]: Fixed the looping of asyncio in the case of reconnection the socket during waiting async read/write from/to the socket.
- bpo-32011 [https://bugs.python.org/issue?@action=redirect&bpo=32011]: Restored support of loading marshal files with the TYPE_INT64 code. These files can be

produced in Python 2.7.

- bpo-28369 [https://bugs.python.org/issue?@action=redirect&bpo=28369]: Enhance add_reader/writer check that socket is not used by some transport. Before, only cases when add_reader/writer were called with an int FD were supported. Now the check is implemented correctly for all file-like objects.
- bpo-31976 [https://bugs.python.org/issue?@action=redirect&bpo=31976]: Fix race condition when flushing a file is slow, which can cause a segfault if closing the file from another thread.
- bpo-31985 [https://bugs.python.org/issue?@action=redirect&bpo=31985]: Formally deprecated aifc.openfp, sunau.openfp, and wave.openfp. Since change 7bc817d5ba917528e8bd07ec461c635291e7b06a in 1993, openfp in each of the three modules had been pointing to that module's open function as a matter of backwards compatibility, though it had been both untested and undocumented.
- bpo-21862 [https://bugs.python.org/issue?@action=redirect&bpo=21862]: cProfile command line now accepts `-m module_name` as an alternative to script path. Patch by Sanyam Khurana.
- bpo-31970 [https://bugs.python.org/issue?@action=redirect&bpo=31970]: Reduce performance overhead of asyncio debug mode.
- bpo-31843 [https://bugs.python.org/issue?@action=redirect&bpo=31843]: *database* argument of sqlite3.connect() now accepts a path-like object, instead of just a string.
- bpo-31945 [https://bugs.python.org/issue?@action=redirect&bpo=31945]: Add Configurable *blocksize* to `HTTPConnection` and `HTTPSConnection` for improved upload throughput. Patch by Nir Soffer.
- bpo-31943 [https://bugs.python.org/issue?@action=redirect&bpo=31943]: Add a `cancelled()` method to **`asyncio.Handle`**. Patch by Marat Sharafutdinov.
- bpo-9678 [https://bugs.python.org/issue?@action=redirect&bpo=9678]: Fixed determining the MAC address in the uuid module: Using ifconfig on NetBSD and OpenBSD. Using arp on Linux,

FreeBSD, NetBSD and OpenBSD. Based on patch by Takayuki Shimizukawa.

- bpo-30057 [https://bugs.python.org/issue?@action=redirect&bpo=30057]: Fix potential missed signal in signal.signal().
- bpo-31933 [https://bugs.python.org/issue?@action=redirect&bpo=31933]: Fix Blake2 params leaf_size and node_offset on big endian platforms. Patch by Jack O'Connor.
- bpo-21423 [https://bugs.python.org/issue?@action=redirect&bpo=21423]: Add an initializer argument to {Process,Thread}PoolExecutor
- bpo-31927 [https://bugs.python.org/issue?@action=redirect&bpo=31927]: Fixed compilation of the socket module on NetBSD 8. Fixed assertion failure or reading arbitrary data when parse a AF_BLUETOOTH address on NetBSD and DragonFly BSD.
- bpo-27666 [https://bugs.python.org/issue?@action=redirect&bpo=27666]: Fixed stack corruption in curses.box() and curses.ungetmouse() when the size of types chtype or mmask_t is less than the size of C long. curses.box() now accepts characters as arguments. Based on patch by Steve Fink.
- bpo-31917 [https://bugs.python.org/issue?@action=redirect&bpo=31917]: Add 3 new clock identifiers: `time.CLOCK_BOOTTIME`, `time.CLOCK_PROF` and `time.CLOCK_UPTIME`.
- bpo-31897 [https://bugs.python.org/issue?@action=redirect&bpo=31897]: plistlib now catches more errors when read binary plists and raises InvalidFileException instead of unexpected exceptions.
- bpo-25720 [https://bugs.python.org/issue?@action=redirect&bpo=25720]: Fix the method for checking pad state of curses WINDOW. Patch by Masayuki Yamamoto.
- bpo-31893 [https://bugs.python.org/issue?@action=redirect&bpo=31893]: Fixed the layout of the kqueue_event structure on OpenBSD and NetBSD. Fixed the comparison of the kqueue_event objects.
- bpo-31891 [https://bugs.python.org/issue?@action=redirect&bpo=31891]: Fixed building the curses module on NetBSD.

- bpo-31884 [https://bugs.python.org/issue?@action=redirect&bpo=31884]: added required constants to subprocess module for setting priority on windows
- bpo-28281 [https://bugs.python.org/issue?@action=redirect&bpo=28281]: Remove year (1-9999) limits on the Calendar.weekday() function. Patch by Mark Gollahon.
- bpo-31702 [https://bugs.python.org/issue?@action=redirect&bpo=31702]: crypt.mksalt() now allows to specify the number of rounds for SHA-256 and SHA-512 hashing.
- bpo-30639 [https://bugs.python.org/issue?@action=redirect&bpo=30639]: **`inspect.getfile()`** no longer computes the repr of unknown objects to display in an error message, to protect against badly behaved custom reprs.
- bpo-30768 [https://bugs.python.org/issue?@action=redirect&bpo=30768]: Fix the pthread+semaphore implementation of PyThread_acquire_lock_timed() when called with timeout > 0 and intr_flag=0: recompute the timeout if sem_timedwait() is interrupted by a signal (EINTR). See also the **PEP 475** [https://peps.python.org/pep-0475/].
- bpo-31854 [https://bugs.python.org/issue?@action=redirect&bpo=31854]: Add `mmap.ACCESS_DEFAULT` constant.
- bpo-31834 [https://bugs.python.org/issue?@action=redirect&bpo=31834]: Use optimized code for BLAKE2 only with SSSE3+. The pure SSE2 implementation is slower than the pure C reference implementation.
- bpo-28292 [https://bugs.python.org/issue?@action=redirect&bpo=28292]: Calendar.itermonthdates() will now consistently raise an exception when a date falls outside of the 0001-01-01 through 9999-12-31 range. To support applications that cannot tolerate such exceptions, the new methods itermonthdays3() and itermonthdays4() are added. The new methods return tuples and are not restricted by the range supported by datetime.date.
- bpo-28564 [https://bugs.python.org/issue?@action=redirect&bpo=28564]: The shutil.rmtree() function has been sped up to 20–40%. This was done using the os.scandir() function.
- bpo-28416 [https://bugs.python.org/issue?

@action=redirect&bpo=28416]: Instances of pickle.Pickler subclass with the persistent_id() method and pickle.Unpickler subclass with the persistent_load() method no longer create reference cycles.
- bpo-31653 [https://bugs.python.org/issue?@action=redirect&bpo=31653]: Don’t release the GIL if we can acquire a multiprocessing semaphore immediately.
- bpo-28326 [https://bugs.python.org/issue?@action=redirect&bpo=28326]: Fix multiprocessing.Process when stdout and/or stderr is closed or None.
- bpo-20825 [https://bugs.python.org/issue?@action=redirect&bpo=20825]: Add `subnet_of` and `superset_of` containment tests to **`ipaddress.IPv6Network`** and **`ipaddress.IPv4Network`**. Patch by Michel Albert and Cheryl Sabella.
- bpo-31827 [https://bugs.python.org/issue?@action=redirect&bpo=31827]: Remove the os.stat_float_times() function. It was introduced in Python 2.3 for backward compatibility with Python 2.2, and was deprecated since Python 3.1.
- bpo-31756 [https://bugs.python.org/issue?@action=redirect&bpo=31756]: Add a `subprocess.Popen(text=False)` keyword argument to `subprocess` functions to be more explicit about when the library should attempt to decode outputs into text. Patch by Andrew Clegg.
- bpo-31819 [https://bugs.python.org/issue?@action=redirect&bpo=31819]: Add AbstractEventLoop.sock_recv_into().
- bpo-31457 [https://bugs.python.org/issue?@action=redirect&bpo=31457]: If nested log adapters are used, the inner `process()` methods are no longer omitted.
- bpo-31457 [https://bugs.python.org/issue?@action=redirect&bpo=31457]: The `manager` property on LoggerAdapter objects is now properly settable.
- bpo-31806 [https://bugs.python.org/issue?@action=redirect&bpo=31806]: Fix timeout rounding in time.sleep(), threading.Lock.acquire() and socket.socket.settimeout() to round correctly negative

timeouts between -1.0 and 0.0. The functions now block waiting for events as expected. Previously, the call was incorrectly non-blocking. Patch by Pablo Galindo.
- bpo-31803 [https://bugs.python.org/issue?@action=redirect&bpo=31803]: time.clock() and time.get_clock_info("clock") now emit a DeprecationWarning warning.
- bpo-31800 [https://bugs.python.org/issue?@action=redirect&bpo=31800]: Extended support for parsing UTC offsets. strptime "%z" can now parse the output generated by datetime.isoformat, including seconds and microseconds.
- bpo-28603 [https://bugs.python.org/issue?@action=redirect&bpo=28603]: traceback: Fix a TypeError that occurred during printing of exception tracebacks when either the current exception or an exception in its context/cause chain is unhashable. Patch by Zane Bitter.
- bpo-30541 [https://bugs.python.org/issue?@action=redirect&bpo=30541]: Add new function to seal a mock and prevent the automatically creation of child mocks. Patch by Mario Corchero.
- bpo-31784 [https://bugs.python.org/issue?@action=redirect&bpo=31784]: Implement the **PEP 564** [https://peps.python.org/pep-0564/], add new 6 new functions with nanosecond resolution to the `time` module: `clock_gettime_ns()`, `clock_settime_ns()`, `monotonic_ns()`, `perf_counter_ns()`, `process_time_ns()`, `time_ns()`.
- bpo-30143 [https://bugs.python.org/issue?@action=redirect&bpo=30143]: 2to3 now generates a code that uses abstract collection classes from collections.abc rather than collections.
- bpo-31770 [https://bugs.python.org/issue?@action=redirect&bpo=31770]: Prevent a crash when calling the `__init__()` method of a `sqlite3.Cursor` object more than once. Patch by Oren Milman.
- bpo-31764 [https://bugs.python.org/issue?@action=redirect&bpo=31764]: Prevent a crash in `sqlite3.Cursor.close()` in case the `Cursor` object is uninitialized. Patch by Oren Milman.
- bpo-31752 [https://bugs.python.org/issue?

@action=redirect&bpo=31752]: Fix possible crash in timedelta constructor called with custom integers.

- bpo-31620 [https://bugs.python.org/issue?@action=redirect&bpo=31620]: an empty asyncio.Queue now doesn't leak memory when queue.get pollers timeout
- bpo-31690 [https://bugs.python.org/issue?@action=redirect&bpo=31690]: Allow the flags re.ASCII, re.LOCALE, and re.UNICODE to be used as group flags for regular expressions.
- bpo-30349 [https://bugs.python.org/issue?@action=redirect&bpo=30349]: FutureWarning is now emitted if a regular expression contains character set constructs that will change semantically in the future (nested sets and set operations).
- bpo-31664 [https://bugs.python.org/issue?@action=redirect&bpo=31664]: Added support for the Blowfish hashing in the crypt module.
- bpo-31632 [https://bugs.python.org/issue?@action=redirect&bpo=31632]: Fix method set_protocol() of class _SSLProtocolTransport in asyncio module. This method was previously modifying a wrong reference to the protocol.
- bpo-15037 [https://bugs.python.org/issue?@action=redirect&bpo=15037]: Added a workaround for getkey() in curses for ncurses 5.7 and earlier.
- bpo-31307 [https://bugs.python.org/issue?@action=redirect&bpo=31307]: Allow use of bytes objects for arguments to **`configparser.ConfigParser.read()`**. Patch by Vincent Michel.
- bpo-31334 [https://bugs.python.org/issue?@action=redirect&bpo=31334]: Fix `poll.poll([timeout])` in the `select` module for arbitrary negative timeouts on all OSes where it can only be a non-negative integer or -1. Patch by Riccardo Coccioli.
- bpo-31310 [https://bugs.python.org/issue?@action=redirect&bpo=31310]: multiprocessing's semaphore tracker should be launched again if crashed.
- bpo-31308 [https://bugs.python.org/issue?@action=redirect&bpo=31308]: Make multiprocessing's forkserver process immune to Ctrl-C and other user interruptions. If it crashes, restart it when necessary.

- bpo-31245 [https://bugs.python.org/issue?@action=redirect&bpo=31245]: Added support for AF_UNIX socket in asyncio `create_datagram_endpoint`.
- bpo-30553 [https://bugs.python.org/issue?@action=redirect&bpo=30553]: Add HTTP/2 status code 421 (Misdirected Request) to **`http.HTTPStatus`**. Patch by Vitor Pereira.

Documentation

- bpo-32105 [https://bugs.python.org/issue?@action=redirect&bpo=32105]: Added asyncio.BaseEventLoop.connect_accepted_socket versionadded marker.

Tests

- bpo-31380 [https://bugs.python.org/issue?@action=redirect&bpo=31380]: Skip test_httpservers test_undecodable_file on macOS: fails on APFS.
- bpo-31705 [https://bugs.python.org/issue?@action=redirect&bpo=31705]: Skip test_socket.test_sha256() on Linux kernel older than 4.5. The test fails with ENOKEY on kernel 3.10 (on ppc64le). A fix was merged into the kernel 4.5.
- bpo-32138 [https://bugs.python.org/issue?@action=redirect&bpo=32138]: Skip on Android test_faulthandler tests that raise SIGSEGV and remove the test.support.requires_android_level decorator.
- bpo-32136 [https://bugs.python.org/issue?@action=redirect&bpo=32136]: The runtime embedding tests have been split out from `Lib/test/test_capi.py` into a new `Lib/test/test_embed.py` file.
- bpo-28668 [https://bugs.python.org/issue?@action=redirect&bpo=28668]: test.support.requires_multiprocessing_queue is removed. Skip tests with test.support.import_module(“multiprocessing.synchronize”) instead when the semaphore implementation is broken or missing.

- bpo-32126 [https://bugs.python.org/issue?@action=redirect&bpo=32126]: Skip test_get_event_loop_new_process in test.test_asyncio.test_events when sem_open() is not functional.
- bpo-31174 [https://bugs.python.org/issue?@action=redirect&bpo=31174]: Fix test_tools.test_unparse: DirectoryTestCase now stores the names sample to always test the same files. It prevents false alarms when hunting reference leaks.

Build

- bpo-28538 [https://bugs.python.org/issue?@action=redirect&bpo=28538]: Revert the previous changes, the if_nameindex structure is defined by Unified Headers.
- bpo-28762 [https://bugs.python.org/issue?@action=redirect&bpo=28762]: Revert the last commit, the F_LOCK macro is defined by Android Unified Headers.
- bpo-29040 [https://bugs.python.org/issue?@action=redirect&bpo=29040]: Support building Android with Unified Headers. The first NDK release to support Unified Headers is android-ndk-r14.
- bpo-32059 [https://bugs.python.org/issue?@action=redirect&bpo=32059]: `detect_modules()` in `setup.py` now also searches the sysroot paths when cross-compiling.
- bpo-31957 [https://bugs.python.org/issue?@action=redirect&bpo=31957]: Fixes Windows SDK version detection when building for Windows.
- bpo-31609 [https://bugs.python.org/issue?@action=redirect&bpo=31609]: Fixes quotes in PCbuild/clean.bat
- bpo-31934 [https://bugs.python.org/issue?@action=redirect&bpo=31934]: Abort the build when building out of a not clean source tree.
- bpo-31926 [https://bugs.python.org/issue?@action=redirect&bpo=31926]: Fixed Argument Clinic sometimes causing compilation errors when there was more than one function and/or method in a .c file with the same name.
- bpo-28791 [https://bugs.python.org/issue?@action=redirect&bpo=28791]: Update Windows builds to use

SQLite 3.21.0.
- bpo-28791 [https://bugs.python.org/issue?@action=redirect&bpo=28791]: Update OS X installer to use SQLite 3.21.0.
- bpo-28643 [https://bugs.python.org/issue?@action=redirect&bpo=28643]: Record profile-opt build progress with stamp files.
- bpo-31866 [https://bugs.python.org/issue?@action=redirect&bpo=31866]: Finish removing support for AtheOS.

Windows

- bpo-1102 [https://bugs.python.org/issue?@action=redirect&bpo=1102]: Return `None` when `View.Fetch()` returns `ERROR_NO_MORE_ITEMS` instead of raising `MSIError`. Initial patch by Anthony Tuininga.
- bpo-31944 [https://bugs.python.org/issue?@action=redirect&bpo=31944]: Fixes Modify button in Apps and Features dialog.
- bpo-20486 [https://bugs.python.org/issue?@action=redirect&bpo=20486]: Implement the `Database.Close()` method to help closing MSI database objects.
- bpo-31857 [https://bugs.python.org/issue?@action=redirect&bpo=31857]: Make the behavior of USE_STACKCHECK deterministic in a multi-threaded environment.

macOS

- bpo-31392 [https://bugs.python.org/issue?@action=redirect&bpo=31392]: Update macOS installer to use OpenSSL 1.0.2m

IDLE

- bpo-32207 [https://bugs.python.org/issue?@action=redirect&bpo=32207]: Improve tk event exception tracebacks in IDLE. When tk event handling is driven by

IDLE's run loop, a confusing and distracting queue.EMPTY traceback context is no longer added to tk event exception tracebacks. The traceback is now the same as when event handling is driven by user code. Patch based on a suggestion by Serhiy Storchaka.

- bpo-32164 [https://bugs.python.org/issue?@action=redirect&bpo=32164]: Delete unused file idlelib/tabbedpages.py. Use of TabbedPageSet in configdialog was replaced by ttk.Notebook.
- bpo-32100 [https://bugs.python.org/issue?@action=redirect&bpo=32100]: IDLE: Fix old and new bugs in pathbrowser; improve tests. Patch mostly by Cheryl Sabella.
- bpo-31858 [https://bugs.python.org/issue?@action=redirect&bpo=31858]: IDLE – Restrict shell prompt manipulation to the shell. Editor and output windows only see an empty last prompt line. This simplifies the code and fixes a minor bug when newline is inserted. Sys.ps1, if present, is read on Shell start-up, but is not set or changed.
- bpo-31860 [https://bugs.python.org/issue?@action=redirect&bpo=31860]: The font sample in the IDLE configuration dialog is now editable. Changes persist while IDLE remains open
- bpo-31836 [https://bugs.python.org/issue?@action=redirect&bpo=31836]: Test_code_module now passes if run after test_idle, which sets ps1. The code module uses sys.ps1 if present or sets it to "> > > " if not. Test_code_module now properly tests both behaviors. Ditto for ps2.
- bpo-28603 [https://bugs.python.org/issue?@action=redirect&bpo=28603]: Fix a TypeError that caused a shell restart when printing a traceback that includes an exception that is unhashable. Patch by Zane Bitter.
- bpo-13802 [https://bugs.python.org/issue?@action=redirect&bpo=13802]: Use non-Latin characters in the IDLE's Font settings sample. Even if one selects a font that defines a limited subset of the unicode Basic Multilingual Plane, tcl/tk will use other fonts that define a character. The expanded example give users of non-Latin characters a better idea of what they might see in IDLE's shell and editors. To make room for the expanded sample, frames on the Font tab

are re-arranged. The Font/Tabs help explains a bit about the additions.

Tools/Demos

- bpo-32159 [https://bugs.python.org/issue?@action=redirect&bpo=32159]: Remove CVS and Subversion tools: remove svneol.py and treesync.py scripts. CPython migrated from CVS to Subversion, to Mercurial, and then to Git. CVS and Subversion are no longer used to develop CPython.
- bpo-30722 [https://bugs.python.org/issue?@action=redirect&bpo=30722]: Make redemo work with Python 3.6 and newer versions. Also, remove the `LOCALE` option since it doesn't work with string patterns in Python 3. Patch by Christoph Sarnowski.

C API

- bpo-20891 [https://bugs.python.org/issue?@action=redirect&bpo=20891]: Fix PyGILState_Ensure(). When PyGILState_Ensure() is called in a non-Python thread before PyEval_InitThreads(), only call PyEval_InitThreads() after calling PyThreadState_New() to fix a crash.
- bpo-32125 [https://bugs.python.org/issue?@action=redirect&bpo=32125]: The `Py_UseClassExceptionsFlag` flag has been removed. It was deprecated and wasn't used anymore since Python 2.0.
- bpo-25612 [https://bugs.python.org/issue?@action=redirect&bpo=25612]: Move the current exception state from the frame object to the co-routine. This simplifies the interpreter and fixes a couple of obscure bugs caused by having swap exception state when entering or exiting a generator.
- bpo-23699 [https://bugs.python.org/issue?@action=redirect&bpo=23699]: Add Py_RETURN_RICHCOMPARE macro to reduce boilerplate code in rich comparison functions.
- bpo-30697 [https://bugs.python.org/issue?@action=redirect&bpo=30697]: The

`PyExc_RecursionErrorInst` singleton is removed and `PyErr_NormalizeException()` does not use it anymore. This singleton is persistent and its members being never cleared may cause a segfault during finalization of the interpreter. See also bpo-22898 [https://bugs.python.org/issue?@action=redirect&bpo=22898].

Python 3.7.0 alpha 2

Release date: 2017-10-16

Core and Builtins

- bpo-31558 [https://bugs.python.org/issue?@action=redirect&bpo=31558]: `gc.freeze()` is a new API that allows for moving all objects currently tracked by the garbage collector to a permanent generation, effectively removing them from future collection events. This can be used to protect those objects from having their PyGC_Head mutated. In effect, this enables great copy-on-write stability at fork().
- bpo-31642 [https://bugs.python.org/issue?@action=redirect&bpo=31642]: Restored blocking «from package import module» by setting sys.modules[«package.module»] to None.
- bpo-31708 [https://bugs.python.org/issue?@action=redirect&bpo=31708]: Allow use of asynchronous generator expressions in synchronous functions.
- bpo-31709 [https://bugs.python.org/issue?@action=redirect&bpo=31709]: Drop support of asynchronous _aiter_.
- bpo-30404 [https://bugs.python.org/issue?@action=redirect&bpo=30404]: The -u option now makes the stdout and stderr streams unbuffered rather than line-buffered.
- bpo-31619 [https://bugs.python.org/issue?@action=redirect&bpo=31619]: Fixed a ValueError when convert a string with large number of underscores to integer with binary base.
- bpo-31602 [https://bugs.python.org/issue?

@action=redirect&bpo=31602]: Fix an assertion failure in `zipimporter.get_source()` in case of a bad `zlib.decompress()`. Patch by Oren Milman.

- bpo-31592 [https://bugs.python.org/issue?@action=redirect&bpo=31592]: Fixed an assertion failure in Python parser in case of a bad `unicodedata.normalize()`. Patch by Oren Milman.
- bpo-31588 [https://bugs.python.org/issue?@action=redirect&bpo=31588]: Raise a `TypeError` with a helpful error message when class creation fails due to a metaclass with a bad `__prepare__()` method. Patch by Oren Milman.
- bpo-31574 [https://bugs.python.org/issue?@action=redirect&bpo=31574]: Importlib was instrumented with two dtrace probes to profile import timing.
- bpo-31566 [https://bugs.python.org/issue?@action=redirect&bpo=31566]: Fix an assertion failure in `_warnings.warn()` in case of a bad `__name__` global. Patch by Oren Milman.
- bpo-31506 [https://bugs.python.org/issue?@action=redirect&bpo=31506]: Improved the error message logic for `object.__new__` and `object.__init__`.
- bpo-31505 [https://bugs.python.org/issue?@action=redirect&bpo=31505]: Fix an assertion failure in `json`, in case `_json.make_encoder()` received a bad `encoder()` argument. Patch by Oren Milman.
- bpo-31492 [https://bugs.python.org/issue?@action=redirect&bpo=31492]: Fix assertion failures in case of failing to import from a module with a bad `__name__` attribute, and in case of failing to access an attribute of such a module. Patch by Oren Milman.
- bpo-31478 [https://bugs.python.org/issue?@action=redirect&bpo=31478]: Fix an assertion failure in `_random.Random.seed()` in case the argument has a bad `__abs__()` method. Patch by Oren Milman.
- bpo-31336 [https://bugs.python.org/issue?@action=redirect&bpo=31336]: Speed up class creation by 10-20% by reducing the overhead in the necessary special method lookups. Patch by Stefan Behnel.
- bpo-31415 [https://bugs.python.org/issue?

@action=redirect&bpo=31415]: Add `-X importtime` option to show how long each import takes. It can be used to optimize application's startup time. Support the **`PYTHONPROFILEIMPORTTIME`** as an equivalent way to enable this.

- bpo-31410 [https://bugs.python.org/issue?@action=redirect&bpo=31410]: Optimized calling wrapper and classmethod descriptors.
- bpo-31353 [https://bugs.python.org/issue?@action=redirect&bpo=31353]: **PEP 553** [https://peps.python.org/pep-0553/] - Add a new built-in called `breakpoint()` which calls `sys.breakpointhook()`. By default this imports `pdb` and calls `pdb.set_trace()`, but users may override `sys.breakpointhook()` to call whatever debugger they want. The original value of the hook is saved in `sys.__breakpointhook__`.
- bpo-17852 [https://bugs.python.org/issue?@action=redirect&bpo=17852]: Maintain a list of open buffered files, flush them before exiting the interpreter. Based on a patch from Armin Rigo.
- bpo-31315 [https://bugs.python.org/issue?@action=redirect&bpo=31315]: Fix an assertion failure in imp.create_dynamic(), when spec.name is not a string. Patch by Oren Milman.
- bpo-31311 [https://bugs.python.org/issue?@action=redirect&bpo=31311]: Fix a crash in the `__setstate__()` method of `ctypes._CData`, in case of a bad `__dict__`. Patch by Oren Milman.
- bpo-31293 [https://bugs.python.org/issue?@action=redirect&bpo=31293]: Fix crashes in true division and multiplication of a timedelta object by a float with a bad as_integer_ratio() method. Patch by Oren Milman.
- bpo-31285 [https://bugs.python.org/issue?@action=redirect&bpo=31285]: Fix an assertion failure in `warnings.warn_explicit`, when the return value of the received loader's `get_source()` has a bad `splitlines()` method. Patch by Oren Milman.
- bpo-30406 [https://bugs.python.org/issue?@action=redirect&bpo=30406]: Make `async` and `await` proper keywords, as specified in **PEP 492** [https://peps.python.org/

pep-0492/].

Library

- bpo-30058 [https://bugs.python.org/issue?@action=redirect&bpo=30058]: Fixed buffer overflow in select.kqueue.control().
- bpo-31672 [https://bugs.python.org/issue?@action=redirect&bpo=31672]: `idpattern` in `string.Template` matched some non-ASCII characters. Now it uses `-i` regular expression local flag to avoid non-ASCII characters.
- bpo-31701 [https://bugs.python.org/issue?@action=redirect&bpo=31701]: On Windows, faulthandler.enable() now ignores MSC and COM exceptions.
- bpo-31728 [https://bugs.python.org/issue?@action=redirect&bpo=31728]: Prevent crashes in `_elementtree` due to unsafe cleanup of `Element.text` and `Element.tail`. Patch by Oren Milman.
- bpo-31671 [https://bugs.python.org/issue?@action=redirect&bpo=31671]: Now `re.compile()` converts passed RegexFlag to normal int object before compiling. bm_regex_compile benchmark shows 14% performance improvements.
- bpo-30397 [https://bugs.python.org/issue?@action=redirect&bpo=30397]: The types of compiled regular objects and match objects are now exposed as `re.Pattern` and `re.Match`. This adds information in pydoc output for the `re` module.
- bpo-31675 [https://bugs.python.org/issue?@action=redirect&bpo=31675]: Fixed memory leaks in Tkinter's methods splitlist() and split() when pass a string larger than 2 GiB.
- bpo-31673 [https://bugs.python.org/issue?@action=redirect&bpo=31673]: Fixed typo in the name of Tkinter's method adderrorinfo().
- bpo-31648 [https://bugs.python.org/issue?@action=redirect&bpo=31648]: Improvements to path predicates in ElementTree: Allow whitespace around predicate parts, i.e. «[a = “text”]» instead of requiring the less readable

«[a=”text”]». Add support for text comparison of the current node, like «[.=”text”]». Patch by Stefan Behnel.

- bpo-30806 [https://bugs.python.org/issue?@action=redirect&bpo=30806]: Fix the string representation of a netrc object.
- bpo-31638 [https://bugs.python.org/issue?@action=redirect&bpo=31638]: Add optional argument `compressed` to `zipapp.create_archive`, and add option `--compress` to the command line interface of `zipapp`.
- bpo-25351 [https://bugs.python.org/issue?@action=redirect&bpo=25351]: Avoid venv activate failures with undefined variables
- bpo-20519 [https://bugs.python.org/issue?@action=redirect&bpo=20519]: Avoid ctypes use (if possible) and improve import time for uuid.
- bpo-28293 [https://bugs.python.org/issue?@action=redirect&bpo=28293]: The regular expression cache is no longer completely dumped when it is full.
- bpo-31596 [https://bugs.python.org/issue?@action=redirect&bpo=31596]: Added pthread_getcpuclockid() to the time module
- bpo-27494 [https://bugs.python.org/issue?@action=redirect&bpo=27494]: Make 2to3 accept a trailing comma in generator expressions. For example, `set(x for x in [],)` is now allowed.
- bpo-30347 [https://bugs.python.org/issue?@action=redirect&bpo=30347]: Stop crashes when concurrently iterate over itertools.groupby() iterators.
- bpo-30346 [https://bugs.python.org/issue?@action=redirect&bpo=30346]: An iterator produced by itertools.groupby() iterator now becomes exhausted after advancing the groupby iterator.
- bpo-31556 [https://bugs.python.org/issue?@action=redirect&bpo=31556]: Cancel asyncio.wait_for future faster if timeout $<= 0$
- bpo-31540 [https://bugs.python.org/issue?@action=redirect&bpo=31540]: Allow passing a context object in **`concurrent.futures.ProcessPoolExecutor`** constructor. Also, free job resources in

`concurrent.futures.ProcessPoolExecutor` earlier to improve memory usage when a worker waits for new jobs.

- bpo-31516 [https://bugs.python.org/issue?@action=redirect&bpo=31516]: `threading.current_thread()` should not return a dummy thread at shutdown.
- bpo-31525 [https://bugs.python.org/issue?@action=redirect&bpo=31525]: In the sqlite module, require the sqlite3_prepare_v2 API. Thus, the sqlite module now requires sqlite version at least 3.3.9.
- bpo-26510 [https://bugs.python.org/issue?@action=redirect&bpo=26510]: argparse subparsers are now required by default. This matches behaviour in Python 2. For optional subparsers, use the new parameter `add_subparsers(required=False)`. Patch by Anthony Sottile. (As of 3.7.0rc1, the default was changed to not required as had been the case since Python 3.3.)
- bpo-27541 [https://bugs.python.org/issue?@action=redirect&bpo=27541]: Reprs of subclasses of some collection and iterator classes (`bytearray`, `array.array`, `collections.deque`, `collections.defaultdict`, `itertools.count`, `itertools.repeat`) now contain actual type name instead of hardcoded names of the base class.
- bpo-31351 [https://bugs.python.org/issue?@action=redirect&bpo=31351]: python -m ensurepip now exits with non-zero exit code if pip bootstrapping has failed.
- bpo-31389 [https://bugs.python.org/issue?@action=redirect&bpo=31389]: `pdb.set_trace()` now takes an optional keyword-only argument `header`. If given, this is printed to the console just before debugging begins.

Documentation

- bpo-31537 [https://bugs.python.org/issue?@action=redirect&bpo=31537]: Fix incorrect usage of `get_history_length` in readline documentation example code. Patch by Brad Smith.
- bpo-30085 [https://bugs.python.org/issue?@action=redirect&bpo=30085]: The operator functions without

double underscores are preferred for clarity. The one with underscores are only kept for back-compatibility.

Build

- bpo-31696 [https://bugs.python.org/issue?@action=redirect&bpo=31696]: Improve compiler version information in `sys.version` when Python is built with Clang.
- bpo-31625 [https://bugs.python.org/issue?@action=redirect&bpo=31625]: Stop using ranlib on static libraries. Instead, we assume ar supports the “s” flag.
- bpo-31624 [https://bugs.python.org/issue?@action=redirect&bpo=31624]: Remove support for BSD/OS.
- bpo-22140 [https://bugs.python.org/issue?@action=redirect&bpo=22140]: Prevent double substitution of prefix in python-config.sh.
- bpo-31569 [https://bugs.python.org/issue?@action=redirect&bpo=31569]: Correct PCBuild/ case to PCbuild/ in build scripts and documentation.
- bpo-31536 [https://bugs.python.org/issue?@action=redirect&bpo=31536]: Avoid wholesale rebuild after `make regen-all` if nothing changed.

IDLE

- bpo-31460 [https://bugs.python.org/issue?@action=redirect&bpo=31460]: Simplify the API of IDLE’s Module Browser. Passing a widget instead of an flist with a root widget opens the option of creating a browser frame that is only part of a window. Passing a full file name instead of pieces assumed to come from a .py file opens the possibility of browsing python files that do not end in .py.
- bpo-31649 [https://bugs.python.org/issue?@action=redirect&bpo=31649]: IDLE - Make _htest, _utest parameters keyword only.
- bpo-31559 [https://bugs.python.org/issue?@action=redirect&bpo=31559]: Remove test order dependence in idle_test.test_browser.
- bpo-31459 [https://bugs.python.org/issue?

@action=redirect&bpo=31459]: Rename IDLE's module browser from Class Browser to Module Browser. The original module-level class and method browser became a module browser, with the addition of module-level functions, years ago. Nested classes and functions were added yesterday. For back-compatibility, the virtual event <<open-class-browser>>, which appears on the Keys tab of the Settings dialog, is not changed. Patch by Cheryl Sabella.
- bpo-31500 [https://bugs.python.org/issue?@action=redirect&bpo=31500]: Default fonts now are scaled on HiDPI displays.
- bpo-1612262 [https://bugs.python.org/issue?@action=redirect&bpo=1612262]: IDLE module browser now shows nested classes and functions. Original patches for code and tests by Guilherme Polo and Cheryl Sabella, respectively.

C API

- bpo-28280 [https://bugs.python.org/issue?@action=redirect&bpo=28280]: Make `PyMapping_Keys()`, `PyMapping_Values()` and `PyMapping_Items()` always return a `list` (rather than a `list` or a `tuple`). Patch by Oren Milman.
- bpo-31532 [https://bugs.python.org/issue?@action=redirect&bpo=31532]: Fix memory corruption due to allocator mix in getpath.c between Py_GetPath() and Py_SetPath()
- bpo-25658 [https://bugs.python.org/issue?@action=redirect&bpo=25658]: Implement **PEP 539** [https://peps.python.org/pep-0539/] for Thread Specific Storage (TSS) API: it is a new Thread Local Storage (TLS) API to CPython which would supersede use of the existing TLS API within the CPython interpreter, while deprecating the existing API. PEP written by Erik M. Bray, patch by Masayuki Yamamoto.

Python 3.7.0 alpha 1

Release date: 2017-09-19

Security

- bpo-29781 [https://bugs.python.org/issue?@action=redirect&bpo=29781]: SSLObject.version() now correctly returns None when handshake over BIO has not been performed yet.
- bpo-29505 [https://bugs.python.org/issue?@action=redirect&bpo=29505]: Add fuzz tests for float(str), int(str), unicode(str); for oss-fuzz.
- bpo-30947 [https://bugs.python.org/issue?@action=redirect&bpo=30947]: Upgrade libexpat embedded copy from version 2.2.1 to 2.2.3 to get security fixes.
- bpo-30730 [https://bugs.python.org/issue?@action=redirect&bpo=30730]: Prevent environment variables injection in subprocess on Windows. Prevent passing other environment variables and command arguments.
- bpo-30694 [https://bugs.python.org/issue?@action=redirect&bpo=30694]: Upgrade expat copy from 2.2.0 to 2.2.1 to get fixes of multiple security vulnerabilities including: **CVE 2017-9233** [https://www.cve.org/CVERecord?id=CVE-2017-9233] (External entity infinite loop DoS), **CVE 2016-9063** [https://www.cve.org/CVERecord?id=CVE-2016-9063] (Integer overflow, re-fix), **CVE 2016-0718** [https://www.cve.org/CVERecord?id=CVE-2016-0718] (Fix regression bugs from 2.2.0's fix to **CVE 2016-0718** [https://www.cve.org/CVERecord?id=CVE-2016-0718]) and **CVE 2012-0876** [https://www.cve.org/CVERecord?id=CVE-2012-0876] (Counter hash flooding with SipHash). Note: the **CVE 2016-5300** [https://www.cve.org/CVERecord?id=CVE-2016-5300] (Use os-specific entropy sources like getrandom) doesn't impact Python, since Python already gets entropy from the OS to set the expat secret using `XML_SetHashSalt()`.
- bpo-30500 [https://bugs.python.org/issue?@action=redirect&bpo=30500]: Fix urllib.parse.splithost() to correctly parse fragments. For example, `splithost('//127.0.0.1#@evil.com/')` now correctly returns the `127.0.0.1` host, instead of treating `@evil.com` as the host in an authentication (`login@host`).
- bpo-29591 [https://bugs.python.org/issue?@action=redirect&bpo=29591]: Update expat copy from 2.1.1 to

2.2.0 to get fixes of **CVE 2016-0718** [https://www.cve.org/CVERecord?id=CVE-2016-0718] and **CVE 2016-4472** [https://www.cve.org/CVERecord?id=CVE-2016-4472]. See https://sourceforge.net/p/expat/bugs/537/ for more information.

Core and Builtins

- bpo-31490 [https://bugs.python.org/issue?@action=redirect&bpo=31490]: Fix an assertion failure in `ctypes` class definition, in case the class has an attribute whose name is specified in `_anonymous_` but not in `_fields_`. Patch by Oren Milman.
- bpo-31471 [https://bugs.python.org/issue?@action=redirect&bpo=31471]: Fix an assertion failure in `subprocess.Popen()` on Windows, in case the env argument has a bad `keys()` method. Patch by Oren Milman.
- bpo-31418 [https://bugs.python.org/issue?@action=redirect&bpo=31418]: Fix an assertion failure in `PyErr_WriteUnraisable()` in case of an exception with a bad `__module__` attribute. Patch by Oren Milman.
- bpo-31416 [https://bugs.python.org/issue?@action=redirect&bpo=31416]: Fix assertion failures in case of a bad warnings.filters or warnings.defaultaction. Patch by Oren Milman.
- bpo-28411 [https://bugs.python.org/issue?@action=redirect&bpo=28411]: Change direct usage of PyInterpreterState.modules to PyImport_GetModuleDict(). Also introduce more uniformity in other code that deals with sys.modules. This helps reduce complications when working on sys.modules.
- bpo-28411 [https://bugs.python.org/issue?@action=redirect&bpo=28411]: Switch to the abstract API when dealing with `PyInterpreterState.modules`. This allows later support for all dict subclasses and other Mapping implementations. Also add a `PyImport_GetModule()` function to reduce a bunch of duplicated code.
- bpo-31411 [https://bugs.python.org/issue?@action=redirect&bpo=31411]: Raise a TypeError instead of SystemError in case warnings.onceregistry is not a dictionary. Patch by Oren Milman.

- bpo-31344 [https://bugs.python.org/issue?@action=redirect&bpo=31344]: For finer control of tracing behaviour when testing the interpreter, two new frame attributes have been added to control the emission of particular trace events: `f_trace_lines` (`True` by default) to turn off per-line trace events; and `f_trace_opcodes` (`False` by default) to turn on per-opcode trace events.
- bpo-31373 [https://bugs.python.org/issue?@action=redirect&bpo=31373]: Fix several possible instances of undefined behavior due to floating-point demotions.
- bpo-30465 [https://bugs.python.org/issue?@action=redirect&bpo=30465]: Location information (`lineno` and `col_offset`) in f-strings is now (mostly) correct. This fixes tools like flake8 from showing warnings on the wrong line (typically the first line of the file).
- bpo-30860 [https://bugs.python.org/issue?@action=redirect&bpo=30860]: Consolidate CPython's global runtime state under a single struct. This improves discoverability of the runtime state.
- bpo-31347 [https://bugs.python.org/issue?@action=redirect&bpo=31347]: Fix possible undefined behavior in _PyObject_FastCall_Prepend.
- bpo-31343 [https://bugs.python.org/issue?@action=redirect&bpo=31343]: Include sys/sysmacros.h for major(), minor(), and makedev(). GNU C library plans to remove the functions from sys/types.h.
- bpo-31291 [https://bugs.python.org/issue?@action=redirect&bpo=31291]: Fix an assertion failure in `zipimport.zipimporter.get_data` on Windows, when the return value of `pathname.replace('/','\\')` isn't a string. Patch by Oren Milman.
- bpo-31271 [https://bugs.python.org/issue?@action=redirect&bpo=31271]: Fix an assertion failure in the `write()` method of `io.TextIOWrapper`, when the encoder doesn't return a bytes object. Patch by Oren Milman.
- bpo-31243 [https://bugs.python.org/issue?@action=redirect&bpo=31243]: Fix a crash in some methods of `io.TextIOWrapper`, when the decoder's state is invalid. Patch by Oren Milman.
- bpo-30721 [https://bugs.python.org/issue?

@action=redirect&bpo=30721]: `print` now shows correct usage hint for using Python 2 redirection syntax. Patch by Sanyam Khurana.
- bpo-31070 [https://bugs.python.org/issue?@action=redirect&bpo=31070]: Fix a race condition in importlib _get_module_lock().
- bpo-30747 [https://bugs.python.org/issue?@action=redirect&bpo=30747]: Add a non-dummy implementation of _Py_atomic_store and _Py_atomic_load on MSVC.
- bpo-31095 [https://bugs.python.org/issue?@action=redirect&bpo=31095]: Fix potential crash during GC caused by `tp_dealloc` which doesn't call `PyObject_GC_UnTrack()`.
- bpo-31071 [https://bugs.python.org/issue?@action=redirect&bpo=31071]: Avoid masking original TypeError in call with * unpacking when other arguments are passed.
- bpo-30978 [https://bugs.python.org/issue?@action=redirect&bpo=30978]: str.format_map() now passes key lookup exceptions through. Previously any exception was replaced with a KeyError exception.
- bpo-30808 [https://bugs.python.org/issue?@action=redirect&bpo=30808]: Use _Py_atomic API for concurrency-sensitive signal state.
- bpo-30876 [https://bugs.python.org/issue?@action=redirect&bpo=30876]: Relative import from unloaded package now reimports the package instead of failing with SystemError. Relative import from non-package now fails with ImportError rather than SystemError.
- bpo-30703 [https://bugs.python.org/issue?@action=redirect&bpo=30703]: Improve signal delivery. Avoid using Py_AddPendingCall from signal handler, to avoid calling signal-unsafe functions. The tests I'm adding here fail without the rest of the patch, on Linux and OS X. This means our signal delivery logic had defects (some signals could be lost).
- bpo-30765 [https://bugs.python.org/issue?@action=redirect&bpo=30765]: Avoid blocking in pthread_mutex_lock() when PyThread_acquire_lock() is asked not to block.

- bpo-31161 [https://bugs.python.org/issue?@action=redirect&bpo=31161]: Make sure the “Missing parentheses” syntax error message is only applied to SyntaxError, not to subclasses. Patch by Martijn Pieters.
- bpo-30814 [https://bugs.python.org/issue?@action=redirect&bpo=30814]: Fixed a race condition when import a submodule from a package.
- bpo-30736 [https://bugs.python.org/issue?@action=redirect&bpo=30736]: The internal unicodedata database has been upgraded to Unicode 10.0.
- bpo-30604 [https://bugs.python.org/issue?@action=redirect&bpo=30604]: Move co_extra_freefuncs from per-thread to per-interpreter to avoid crashes.
- bpo-30597 [https://bugs.python.org/issue?@action=redirect&bpo=30597]: `print` now shows expected input in custom error message when used as a Python 2 statement. Patch by Sanyam Khurana.
- bpo-30682 [https://bugs.python.org/issue?@action=redirect&bpo=30682]: Removed a too-strict assertion that failed for certain f-strings, such as eval(«f”\n”») and eval(«f”\r”»).
- bpo-30501 [https://bugs.python.org/issue?@action=redirect&bpo=30501]: The compiler now produces more optimal code for complex condition expressions in the «if», «while» and «assert» statement, the «if» expression, and generator expressions and comprehensions.
- bpo-28180 [https://bugs.python.org/issue?@action=redirect&bpo=28180]: Implement **PEP 538** [https://peps.python.org/pep-0538/] (legacy C locale coercion). This means that when a suitable coercion target locale is available, both the core interpreter and locale-aware C extensions will assume the use of UTF-8 as the default text encoding, rather than ASCII.
- bpo-30486 [https://bugs.python.org/issue?@action=redirect&bpo=30486]: Allows setting cell values for _closure_. Patch by Lisa Roach.
- bpo-30537 [https://bugs.python.org/issue?@action=redirect&bpo=30537]: itertools.islice now accepts integer-like objects (having an _index_ method) as start, stop, and slice arguments

- bpo-25324 [https://bugs.python.org/issue?@action=redirect&bpo=25324]: Tokens needed for parsing in Python moved to C. `COMMENT`, `NL` and `ENCODING`. This way the tokens and tok_names in the token module don't get changed when you import the tokenize module.
- bpo-29104 [https://bugs.python.org/issue?@action=redirect&bpo=29104]: Fixed parsing backslashes in f-strings.
- bpo-27945 [https://bugs.python.org/issue?@action=redirect&bpo=27945]: Fixed various segfaults with dict when input collections are mutated during searching, inserting or comparing. Based on patches by Duane Griffin and Tim Mitchell.
- bpo-25794 [https://bugs.python.org/issue?@action=redirect&bpo=25794]: Fixed type.__setattr__() and type.__delattr__() for non-interned attribute names. Based on patch by Eryk Sun.
- bpo-30039 [https://bugs.python.org/issue?@action=redirect&bpo=30039]: If a KeyboardInterrupt happens when the interpreter is in the middle of resuming a chain of nested "yield from" or "await" calls, it's now correctly delivered to the innermost frame.
- bpo-28974 [https://bugs.python.org/issue?@action=redirect&bpo=28974]: `object.__format__(x, '')` is now equivalent to `str(x)` rather than `format(str(self), '')`.
- bpo-30024 [https://bugs.python.org/issue?@action=redirect&bpo=30024]: Circular imports involving absolute imports with binding a submodule to a name are now supported.
- bpo-12414 [https://bugs.python.org/issue?@action=redirect&bpo=12414]: sys.getsizeof() on a code object now returns the sizes which includes the code struct and sizes of objects which it references. Patch by Donghee Na.
- bpo-29839 [https://bugs.python.org/issue?@action=redirect&bpo=29839]: len() now raises ValueError rather than OverflowError if __len__() returned a large negative integer.
- bpo-11913 [https://bugs.python.org/issue?@action=redirect&bpo=11913]: README.rst is now included in

the list of distutils standard READMEs and therefore included in source distributions.

- bpo-29914 [https://bugs.python.org/issue?@action=redirect&bpo=29914]: Fixed default implementations of _reduce_ and _reduce_ex_(). object._reduce_() no longer takes arguments, object._reduce_ex_() now requires one argument.
- bpo-29949 [https://bugs.python.org/issue?@action=redirect&bpo=29949]: Fix memory usage regression of set and frozenset object.
- bpo-29935 [https://bugs.python.org/issue?@action=redirect&bpo=29935]: Fixed error messages in the index() method of tuple, list and deque when pass indices of wrong type.
- bpo-29816 [https://bugs.python.org/issue?@action=redirect&bpo=29816]: Shift operation now has less opportunity to raise OverflowError. ValueError always is raised rather than OverflowError for negative counts. Shifting zero with non-negative count always returns zero.
- bpo-24821 [https://bugs.python.org/issue?@action=redirect&bpo=24821]: Fixed the slowing down to 25 times in the searching of some unlucky Unicode characters.
- bpo-29102 [https://bugs.python.org/issue?@action=redirect&bpo=29102]: Add a unique ID to PyInterpreterState. This makes it easier to identify each subinterpreter.
- bpo-29894 [https://bugs.python.org/issue?@action=redirect&bpo=29894]: The deprecation warning is emitted if _complex_ returns an instance of a strict subclass of complex. In a future versions of Python this can be an error.
- bpo-29859 [https://bugs.python.org/issue?@action=redirect&bpo=29859]: Show correct error messages when any of the pthread_* calls in thread_pthread.h fails.
- bpo-29849 [https://bugs.python.org/issue?@action=redirect&bpo=29849]: Fix a memory leak when an ImportError is raised during from import.
- bpo-28856 [https://bugs.python.org/issue?@action=redirect&bpo=28856]: Fix an oversight that %b format for bytes should support objects follow the buffer protocol.

- bpo-29723 [https://bugs.python.org/issue?@action=redirect&bpo=29723]: The `sys.path[0]` initialization change for bpo-29139 [https://bugs.python.org/issue?@action=redirect&bpo=29139] caused a regression by revealing an inconsistency in how sys.path is initialized when executing `__main__` from a zipfile, directory, or other import location. The interpreter now consistently avoids ever adding the import location's parent directory to `sys.path`, and ensures no other `sys.path` entries are inadvertently modified when inserting the import location named on the command line.
- bpo-29568 [https://bugs.python.org/issue?@action=redirect&bpo=29568]: Escaped percent «%%» in the format string for classic string formatting no longer allows any characters between two percents.
- bpo-29714 [https://bugs.python.org/issue?@action=redirect&bpo=29714]: Fix a regression that bytes format may fail when containing zero bytes inside.
- bpo-29695 [https://bugs.python.org/issue?@action=redirect&bpo=29695]: bool(), float(), list() and tuple() no longer take keyword arguments. The first argument of int() can now be passes only as positional argument.
- bpo-28893 [https://bugs.python.org/issue?@action=redirect&bpo=28893]: Set correct _cause_ for errors about invalid awaitables returned from _aiter_ and _anext_.
- bpo-28876 [https://bugs.python.org/issue?@action=redirect&bpo=28876]: `bool(range)` works even if `len(range)` raises **`OverflowError`**.
- bpo-29683 [https://bugs.python.org/issue?@action=redirect&bpo=29683]: Fixes to memory allocation in _PyCode_SetExtra. Patch by Brian Coleman.
- bpo-29684 [https://bugs.python.org/issue?@action=redirect&bpo=29684]: Fix minor regression of PyEval_CallObjectWithKeywords. It should raise TypeError when kwargs is not a dict. But it might cause segv when args=NULL and kwargs is not a dict.
- bpo-28598 [https://bugs.python.org/issue?@action=redirect&bpo=28598]: Support _rmod_ for subclasses of str being called before str._mod_. Patch by Martijn Pieters.
- bpo-29607 [https://bugs.python.org/issue?@action=redirect&bpo=29607]: Fix stack_effect computation for

CALL_FUNCTION_EX. Patch by Matthieu Dartiailh.

- bpo-29602 [https://bugs.python.org/issue?@action=redirect&bpo=29602]: Fix incorrect handling of signed zeros in complex constructor for complex subclasses and for inputs having a _complex_ method. Patch by Serhiy Storchaka.
- bpo-29347 [https://bugs.python.org/issue?@action=redirect&bpo=29347]: Fixed possibly dereferencing undefined pointers when creating weakref objects.
- bpo-29463 [https://bugs.python.org/issue?@action=redirect&bpo=29463]: Add `docstring` field to Module, ClassDef, FunctionDef, and AsyncFunctionDef ast nodes. docstring is not first stmt in their body anymore. It affects `co_firstlineno` and `co_lnotab` of code object for module and class. (Reverted in bpo-32911 [https://bugs.python.org/issue?@action=redirect&bpo=32911].)
- bpo-29438 [https://bugs.python.org/issue?@action=redirect&bpo=29438]: Fixed use-after-free problem in key sharing dict.
- bpo-29546 [https://bugs.python.org/issue?@action=redirect&bpo=29546]: Set the “path” and “name” attribute on ImportError for `from ... import ...`.
- bpo-29546 [https://bugs.python.org/issue?@action=redirect&bpo=29546]: Improve from-import error message with location
- bpo-29478 [https://bugs.python.org/issue?@action=redirect&bpo=29478]: If max_line_length=None is specified while using the Compat32 policy, it is no longer ignored. Patch by Mircea Cosbuc.
- bpo-29319 [https://bugs.python.org/issue?@action=redirect&bpo=29319]: Prevent RunMainFromImporter overwriting sys.path[0].
- bpo-29337 [https://bugs.python.org/issue?@action=redirect&bpo=29337]: Fixed possible BytesWarning when compare the code objects. Warnings could be emitted at compile time.
- bpo-29327 [https://bugs.python.org/issue?@action=redirect&bpo=29327]: Fixed a crash when pass the iterable keyword argument to sorted().
- bpo-29034 [https://bugs.python.org/issue?

@action=redirect&bpo=29034]: Fix memory leak and use-after-free in os module (path_converter).

- bpo-29159 [https://bugs.python.org/issue?@action=redirect&bpo=29159]: Fix regression in bytes(x) when x.__index__() raises Exception.
- bpo-29049 [https://bugs.python.org/issue?@action=redirect&bpo=29049]: Call _PyObject_GC_TRACK() lazily when calling Python function. Calling function is up to 5% faster.
- bpo-28927 [https://bugs.python.org/issue?@action=redirect&bpo=28927]: bytes.fromhex() and bytearray.fromhex() now ignore all ASCII whitespace, not only spaces. Patch by Robert Xiao.
- bpo-28932 [https://bugs.python.org/issue?@action=redirect&bpo=28932]: Do not include <sys/random.h> if it does not exist.
- bpo-25677 [https://bugs.python.org/issue?@action=redirect&bpo=25677]: Correct the positioning of the syntax error caret for indented blocks. Based on patch by Michael Layzell.
- bpo-29000 [https://bugs.python.org/issue?@action=redirect&bpo=29000]: Fixed bytes formatting of octals with zero padding in alternate form.
- bpo-18896 [https://bugs.python.org/issue?@action=redirect&bpo=18896]: Python function can now have more than 255 parameters. collections.namedtuple() now supports tuples with more than 255 elements.
- bpo-28596 [https://bugs.python.org/issue?@action=redirect&bpo=28596]: The preferred encoding is UTF-8 on Android. Patch written by Chi Hsuan Yen.
- bpo-22257 [https://bugs.python.org/issue?@action=redirect&bpo=22257]: Clean up interpreter startup (see **PEP 432** [https://peps.python.org/pep-0432/]).
- bpo-26919 [https://bugs.python.org/issue?@action=redirect&bpo=26919]: On Android, operating system data is now always encoded/decoded to/from UTF-8, instead of the locale encoding to avoid inconsistencies with os.fsencode() and os.fsdecode() which are already using UTF-8.
- bpo-28991 [https://bugs.python.org/issue?

@action=redirect&bpo=28991]: functools.lru_cache() was susceptible to an obscure reentrancy bug triggerable by a monkey-patched len() function.

- bpo-28147 [https://bugs.python.org/issue?@action=redirect&bpo=28147]: Fix a memory leak in split-table dictionaries: setattr() must not convert combined table into split table. Patch written by INADA Naoki.
- bpo-28739 [https://bugs.python.org/issue?@action=redirect&bpo=28739]: f-string expressions are no longer accepted as docstrings and by ast.literal_eval() even if they do not include expressions.
- bpo-28512 [https://bugs.python.org/issue?@action=redirect&bpo=28512]: Fixed setting the offset attribute of SyntaxError by PyErr_SyntaxLocationEx() and PyErr_SyntaxLocationObject().
- bpo-28918 [https://bugs.python.org/issue?@action=redirect&bpo=28918]: Fix the cross compilation of xxlimited when Python has been built with Py_DEBUG defined.
- bpo-23722 [https://bugs.python.org/issue?@action=redirect&bpo=23722]: Rather than silently producing a class that doesn't support zero-argument `super()` in methods, failing to pass the new `__classcell__` namespace entry up to `type.__new__` now results in a `DeprecationWarning` and a class that supports zero-argument `super()`.
- bpo-28797 [https://bugs.python.org/issue?@action=redirect&bpo=28797]: Modifying the class _dict_ inside the _set_name_ method of a descriptor that is used inside that class no longer prevents calling the _set_name_ method of other descriptors.
- bpo-28799 [https://bugs.python.org/issue?@action=redirect&bpo=28799]: Remove the `PyEval_GetCallStats()` function and deprecate the untested and undocumented `sys.callstats()` function. Remove the `CALL_PROFILE` special build: use the **`sys.setprofile()`** function, **`cProfile`** or **`profile`** to profile function calls.
- bpo-12844 [https://bugs.python.org/issue?@action=redirect&bpo=12844]: More than 255 arguments can

now be passed to a function.

- bpo-28782 [https://bugs.python.org/issue?@action=redirect&bpo=28782]: Fix a bug in the implementation `yield from` when checking if the next instruction is YIELD_FROM. Regression introduced by WORDCODE (bpo-26647 [https://bugs.python.org/issue?@action=redirect&bpo=26647]).
- bpo-28774 [https://bugs.python.org/issue?@action=redirect&bpo=28774]: Fix error position of the unicode error in ASCII and Latin1 encoders when a string returned by the error handler contains multiple non-encodable characters (non-ASCII for the ASCII codec, characters out of the U+0000-U+00FF range for Latin1).
- bpo-28731 [https://bugs.python.org/issue?@action=redirect&bpo=28731]: Optimize _PyDict_NewPresized() to create correct size dict. Improve speed of dict literal with constant keys up to 30%.
- bpo-28532 [https://bugs.python.org/issue?@action=redirect&bpo=28532]: Show sys.version when -V option is supplied twice.
- bpo-27100 [https://bugs.python.org/issue?@action=redirect&bpo=27100]: The with-statement now checks for _enter_ before it checks for _exit_. This gives less confusing error messages when both methods are missing. Patch by Jonathan Ellington.
- bpo-28746 [https://bugs.python.org/issue?@action=redirect&bpo=28746]: Fix the set_inheritable() file descriptor method on platforms that do not have the ioctl FIOCLEX and FIONCLEX commands.
- bpo-26920 [https://bugs.python.org/issue?@action=redirect&bpo=26920]: Fix not getting the locale's charset upon initializing the interpreter, on platforms that do not have langinfo.
- bpo-28648 [https://bugs.python.org/issue?@action=redirect&bpo=28648]: Fixed crash in Py_DecodeLocale() in debug build on Mac OS X when decode astral characters. Patch by Xiang Zhang.
- bpo-28665 [https://bugs.python.org/issue?@action=redirect&bpo=28665]: Improve speed of the STORE_DEREF opcode by 40%.

- bpo-19398 [https://bugs.python.org/issue?@action=redirect&bpo=19398]: Extra slash no longer added to sys.path components in case of empty compile-time PYTHONPATH components.
- bpo-28621 [https://bugs.python.org/issue?@action=redirect&bpo=28621]: Sped up converting int to float by reusing faster bits counting implementation. Patch by Adrian Wielgosik.
- bpo-28580 [https://bugs.python.org/issue?@action=redirect&bpo=28580]: Optimize iterating split table values. Patch by Xiang Zhang.
- bpo-28583 [https://bugs.python.org/issue?@action=redirect&bpo=28583]: PyDict_SetDefault didn't combine split table when needed. Patch by Xiang Zhang.
- bpo-28128 [https://bugs.python.org/issue?@action=redirect&bpo=28128]: Deprecation warning for invalid str and byte escape sequences now prints better information about where the error occurs. Patch by Serhiy Storchaka and Eric Smith.
- bpo-28509 [https://bugs.python.org/issue?@action=redirect&bpo=28509]: dict.update() no longer allocate unnecessary large memory.
- bpo-28426 [https://bugs.python.org/issue?@action=redirect&bpo=28426]: Fixed potential crash in PyUnicode_AsDecodedObject() in debug build.
- bpo-28517 [https://bugs.python.org/issue?@action=redirect&bpo=28517]: Fixed of-by-one error in the peephole optimizer that caused keeping unreachable code.
- bpo-28214 [https://bugs.python.org/issue?@action=redirect&bpo=28214]: Improved exception reporting for problematic __set_name__ attributes.
- bpo-23782 [https://bugs.python.org/issue?@action=redirect&bpo=23782]: Fixed possible memory leak in _PyTraceback_Add() and exception loss in PyTraceBack_Here().
- bpo-28183 [https://bugs.python.org/issue?@action=redirect&bpo=28183]: Optimize and cleanup dict iteration.
- bpo-26081 [https://bugs.python.org/issue?@action=redirect&bpo=26081]: Added C implementation of

asyncio.Future. Original patch by Yury Selivanov.

- bpo-28379 [https://bugs.python.org/issue?@action=redirect&bpo=28379]: Added sanity checks and tests for PyUnicode_CopyCharacters(). Patch by Xiang Zhang.
- bpo-28376 [https://bugs.python.org/issue?@action=redirect&bpo=28376]: The type of long range iterator is now registered as Iterator. Patch by Oren Milman.
- bpo-28376 [https://bugs.python.org/issue?@action=redirect&bpo=28376]: Creating instances of range_iterator by calling range_iterator type now is disallowed. Calling iter() on range instance is the only way. Patch by Oren Milman.
- bpo-26906 [https://bugs.python.org/issue?@action=redirect&bpo=26906]: Resolving special methods of uninitialized type now causes implicit initialization of the type instead of a fail.
- bpo-18287 [https://bugs.python.org/issue?@action=redirect&bpo=18287]: PyType_Ready() now checks that tp_name is not NULL. Original patch by Niklas Koep.
- bpo-24098 [https://bugs.python.org/issue?@action=redirect&bpo=24098]: Fixed possible crash when AST is changed in process of compiling it.
- bpo-28201 [https://bugs.python.org/issue?@action=redirect&bpo=28201]: Dict reduces possibility of 2nd conflict in hash table when hashes have same lower bits.
- bpo-28350 [https://bugs.python.org/issue?@action=redirect&bpo=28350]: String constants with null character no longer interned.
- bpo-26617 [https://bugs.python.org/issue?@action=redirect&bpo=26617]: Fix crash when GC runs during weakref callbacks.
- bpo-27942 [https://bugs.python.org/issue?@action=redirect&bpo=27942]: String constants now interned recursively in tuples and frozensets.
- bpo-28289 [https://bugs.python.org/issue?@action=redirect&bpo=28289]: ImportError.__init__ now resets not specified attributes.
- bpo-21578 [https://bugs.python.org/issue?@action=redirect&bpo=21578]: Fixed misleading error message when ImportError called with invalid keyword args.

- bpo-28203 [https://bugs.python.org/issue?@action=redirect&bpo=28203]: Fix incorrect type in complex(1.0, {2:3}) error message. Patch by Soumya Sharma.
- bpo-28086 [https://bugs.python.org/issue?@action=redirect&bpo=28086]: Single var-positional argument of tuple subtype was passed unscathed to the C-defined function. Now it is converted to exact tuple.
- bpo-28214 [https://bugs.python.org/issue?@action=redirect&bpo=28214]: Now __set_name__ is looked up on the class instead of the instance.
- bpo-27955 [https://bugs.python.org/issue?@action=redirect&bpo=27955]: Fallback on reading /dev/urandom device when the getrandom() syscall fails with EPERM, for example when blocked by SECCOMP.
- bpo-28192 [https://bugs.python.org/issue?@action=redirect&bpo=28192]: Don't import readline in isolated mode.
- bpo-27441 [https://bugs.python.org/issue?@action=redirect&bpo=27441]: Remove some redundant assignments to ob_size in longobject.c. Thanks Oren Milman.
- bpo-27222 [https://bugs.python.org/issue?@action=redirect&bpo=27222]: Clean up redundant code in long_rshift function. Thanks Oren Milman.
- Upgrade internal unicode databases to Unicode version 9.0.0.
- bpo-28131 [https://bugs.python.org/issue?@action=redirect&bpo=28131]: Fix a regression in zipimport's compile_source(). zipimport should use the same optimization level as the interpreter.
- bpo-28126 [https://bugs.python.org/issue?@action=redirect&bpo=28126]: Replace Py_MEMCPY with memcpy(). Visual Studio can properly optimize memcpy().
- bpo-28120 [https://bugs.python.org/issue?@action=redirect&bpo=28120]: Fix dict.pop() for split dictionary when trying to remove a «pending key» (Not yet inserted in split-table). Patch by Xiang Zhang.
- bpo-26182 [https://bugs.python.org/issue?@action=redirect&bpo=26182]: Raise DeprecationWarning when async and await keywords are used as variable/attribute/class/function name.
- bpo-26182 [https://bugs.python.org/issue?

@action=redirect&bpo=26182]: Fix a refleak in code that raises DeprecationWarning.
- bpo-28721 [https://bugs.python.org/issue?@action=redirect&bpo=28721]: Fix asynchronous generators aclose() and athrow() to handle StopAsyncIteration propagation properly.
- bpo-26110 [https://bugs.python.org/issue?@action=redirect&bpo=26110]: Speed-up method calls: add LOAD_METHOD and CALL_METHOD opcodes.

Library

- bpo-31499 [https://bugs.python.org/issue?@action=redirect&bpo=31499]: xml.etree: Fix a crash when a parser is part of a reference cycle.
- bpo-31482 [https://bugs.python.org/issue?@action=redirect&bpo=31482]: `random.seed()` now works with bytes in version=1
- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: typing.get_type_hints now finds the right globalns for classes and modules by default (when no `globalns` was specified by the caller).
- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Speed improvements to the `typing` module. Original PRs by Ivan Levkivskyi and Mitar.
- bpo-31544 [https://bugs.python.org/issue?@action=redirect&bpo=31544]: The C accelerator module of ElementTree ignored exceptions raised when looking up TreeBuilder target methods in XMLParser().
- bpo-31234 [https://bugs.python.org/issue?@action=redirect&bpo=31234]: socket.create_connection() now fixes manually a reference cycle: clear the variable storing the last exception on success.
- bpo-31457 [https://bugs.python.org/issue?@action=redirect&bpo=31457]: LoggerAdapter objects can now be nested.
- bpo-31431 [https://bugs.python.org/issue?@action=redirect&bpo=31431]: SSLContext.check_hostname now automatically sets SSLContext.verify_mode to ssl.CERT_REQUIRED instead of failing with a ValueError.

- bpo-31233 [https://bugs.python.org/issue?@action=redirect&bpo=31233]: socketserver.ThreadingMixIn now keeps a list of non-daemonic threads to wait until all these threads complete in server_close().
- bpo-28638 [https://bugs.python.org/issue?@action=redirect&bpo=28638]: Changed the implementation strategy for collections.namedtuple() to substantially reduce the use of exec() in favor of precomputed methods. As a result, the *verbose* parameter and *_source* attribute are no longer supported. The benefits include 1) having a smaller memory footprint for applications using multiple named tuples, 2) faster creation of the named tuple class (approx 4x to 6x depending on how it is measured), and 3) minor speed-ups for instance creation using __new__, _make, and _replace. (The primary patch contributor is Jelle Zijlstra with further improvements by INADA Naoki, Serhiy Storchaka, and Raymond Hettinger.)
- bpo-31400 [https://bugs.python.org/issue?@action=redirect&bpo=31400]: Improves SSL error handling to avoid losing error numbers.
- bpo-27629 [https://bugs.python.org/issue?@action=redirect&bpo=27629]: Make return types of SSLContext.wrap_bio() and SSLContext.wrap_socket() customizable.
- bpo-28958 [https://bugs.python.org/issue?@action=redirect&bpo=28958]: ssl.SSLContext() now uses OpenSSL error information when a context cannot be instantiated.
- bpo-28182 [https://bugs.python.org/issue?@action=redirect&bpo=28182]: The SSL module now raises SSLCertVerificationError when OpenSSL fails to verify the peer’s certificate. The exception contains more information about the error.
- bpo-27340 [https://bugs.python.org/issue?@action=redirect&bpo=27340]: SSLSocket.sendall() now uses memoryview to create slices of data. This fixes support for all bytes-like object. It is also more efficient and avoids costly copies.
- bpo-14191 [https://bugs.python.org/issue?@action=redirect&bpo=14191]: A new function

`argparse.ArgumentParser.parse_intermixed_args` provides the ability to parse command lines where there user intermixes options and positional arguments.

- bpo-31178 [https://bugs.python.org/issue?@action=redirect&bpo=31178]: Fix string concatenation bug in rare error path in the subprocess module
- bpo-31350 [https://bugs.python.org/issue?@action=redirect&bpo=31350]: Micro-optimize **`asyncio._get_running_loop()`** to become up to 10% faster.
- bpo-31170 [https://bugs.python.org/issue?@action=redirect&bpo=31170]: expat: Update libexpat from 2.2.3 to 2.2.4. Fix copying of partial characters for UTF-8 input (libexpat bug 115): https://github.com/libexpat/libexpat/issues/115
- bpo-29136 [https://bugs.python.org/issue?@action=redirect&bpo=29136]: Add TLS 1.3 cipher suites and OP_NO_TLSv1_3.
- bpo-1198569 [https://bugs.python.org/issue?@action=redirect&bpo=1198569]: `string.Template` subclasses can optionally define `braceidpattern` if they want to specify different placeholder patterns inside and outside the braces. If None (the default) it falls back to `idpattern`.
- bpo-31326 [https://bugs.python.org/issue?@action=redirect&bpo=31326]: concurrent.futures.ProcessPoolExecutor.shutdown() now explicitly closes the call queue. Moreover, shutdown(wait=True) now also join the call queue thread, to prevent leaking a dangling thread.
- bpo-27144 [https://bugs.python.org/issue?@action=redirect&bpo=27144]: The `map()` and `as_completed()` iterators in `concurrent.futures` now avoid keeping a reference to yielded objects.
- bpo-31281 [https://bugs.python.org/issue?@action=redirect&bpo=31281]: Fix `fileinput.FileInput(files, inplace=True)` when `files` contain `pathlib.Path` objects.
- bpo-10746 [https://bugs.python.org/issue?@action=redirect&bpo=10746]: Fix ctypes producing wrong **PEP 3118** [https://peps.python.org/pep-3118/] type codes for integer

types.

- bpo-27584 [https://bugs.python.org/issue?@action=redirect&bpo=27584]: `AF_VSOCK` has been added to the socket interface which allows communication between virtual machines and their host.
- bpo-22536 [https://bugs.python.org/issue?@action=redirect&bpo=22536]: The subprocess module now sets the filename when FileNotFoundError is raised on POSIX systems due to the executable or cwd not being found.
- bpo-29741 [https://bugs.python.org/issue?@action=redirect&bpo=29741]: Update some methods in the _pyio module to also accept integer types. Patch by Oren Milman.
- bpo-31249 [https://bugs.python.org/issue?@action=redirect&bpo=31249]: concurrent.futures: WorkItem.run() used by ThreadPoolExecutor now breaks a reference cycle between an exception object and the WorkItem object.
- bpo-31247 [https://bugs.python.org/issue?@action=redirect&bpo=31247]: xmlrpc.server now explicitly breaks reference cycles when using sys.exc_info() in code handling exceptions.
- bpo-23835 [https://bugs.python.org/issue?@action=redirect&bpo=23835]: configparser: reading defaults in the `ConfigParser()` constructor is now using `read_dict()`, making its behavior consistent with the rest of the parser. Non-string keys and values in the defaults dictionary are now being implicitly converted to strings. Patch by James Tocknell.
- bpo-31238 [https://bugs.python.org/issue?@action=redirect&bpo=31238]: pydoc: the stop() method of the private ServerThread class now waits until DocServer.serve_until_quit() completes and then explicitly sets its docserver attribute to None to break a reference cycle.
- bpo-5001 [https://bugs.python.org/issue?@action=redirect&bpo=5001]: Many asserts in `multiprocessing` are now more informative, and some error types have been changed to more specific ones.
- bpo-31109 [https://bugs.python.org/issue?@action=redirect&bpo=31109]: Convert zipimport to use Argument Clinic.

- bpo-30102 [https://bugs.python.org/issue?@action=redirect&bpo=30102]: The ssl and hashlib modules now call OPENSSL_add_all_algorithms_noconf() on OpenSSL < 1.1.0. The function detects CPU features and enables optimizations on some CPU architectures such as POWER8. Patch is based on research from Gustavo Serra Scalet.
- bpo-18966 [https://bugs.python.org/issue?@action=redirect&bpo=18966]: Non-daemonic threads created by a multiprocessing.Process are now joined on child exit.
- bpo-31183 [https://bugs.python.org/issue?@action=redirect&bpo=31183]: `dis` now works with asynchronous generator and coroutine objects. Patch by George Collins based on diagnosis by Luciano Ramalho.
- bpo-5001 [https://bugs.python.org/issue?@action=redirect&bpo=5001]: There are a number of uninformative asserts in the `multiprocessing` module, as noted in issue 5001. This change fixes two of the most potentially problematic ones, since they are in error-reporting code, in the `multiprocessing.managers.convert_to_error` function. (It also makes more informative a ValueError message.) The only potentially problematic change is that the AssertionError is now a TypeError; however, this should also help distinguish it from an AssertionError being *reported* by the function/its caller (such as in issue 31169). - Patch by Allen W. Smith (drallensmith on github).
- bpo-31185 [https://bugs.python.org/issue?@action=redirect&bpo=31185]: Fixed miscellaneous errors in asyncio speedup module.
- bpo-31151 [https://bugs.python.org/issue?@action=redirect&bpo=31151]: socketserver.ForkingMixIn.server_close() now waits until all child processes completed to prevent leaking zombie processes.
- bpo-31072 [https://bugs.python.org/issue?@action=redirect&bpo=31072]: Add an `include_file` parameter to `zipapp.create_archive()`
- bpo-24700 [https://bugs.python.org/issue?@action=redirect&bpo=24700]: Optimize array.array comparison. It is now from 10x up to 70x faster when comparing arrays holding values of the same integer type.

- bpo-31135 [https://bugs.python.org/issue?@action=redirect&bpo=31135]: ttk: fix the destroy() method of LabeledScale and OptionMenu classes. Call the parent destroy() method even if the used attribute doesn't exist. The LabeledScale.destroy() method now also explicitly clears label and scale attributes to help the garbage collector to destroy all widgets.
- bpo-31107 [https://bugs.python.org/issue?@action=redirect&bpo=31107]: Fix `copyreg._slotnames()` mangled attribute calculation for classes whose name begins with an underscore. Patch by Shane Harvey.
- bpo-31080 [https://bugs.python.org/issue?@action=redirect&bpo=31080]: Allow `logging.config.fileConfig` to accept kwargs and/or args.
- bpo-30897 [https://bugs.python.org/issue?@action=redirect&bpo=30897]: `pathlib.Path` objects now include an `is_mount()` method (only implemented on POSIX). This is similar to `os.path.ismount(p)`. Patch by Cooper Ry Lees.
- bpo-31061 [https://bugs.python.org/issue?@action=redirect&bpo=31061]: Fixed a crash when using asyncio and threads.
- bpo-30987 [https://bugs.python.org/issue?@action=redirect&bpo=30987]: Added support for CAN ISO-TP protocol in the socket module.
- bpo-30522 [https://bugs.python.org/issue?@action=redirect&bpo=30522]: Added a `setStream` method to `logging.StreamHandler` to allow the stream to be set after creation.
- bpo-30502 [https://bugs.python.org/issue?@action=redirect&bpo=30502]: Fix handling of long oids in ssl. Based on patch by Christian Heimes.
- bpo-5288 [https://bugs.python.org/issue?@action=redirect&bpo=5288]: Support tzinfo objects with sub-minute offsets.
- bpo-30919 [https://bugs.python.org/issue?@action=redirect&bpo=30919]: Fix shared memory performance regression in multiprocessing in 3.x. Shared memory used anonymous memory mappings in 2.x, while 3.x mmaps actual files. Try to be careful to do as little disk I/O as possible.

- bpo-26732 [https://bugs.python.org/issue?@action=redirect&bpo=26732]: Fix too many fds in processes started with the «forkserver» method. A child process would inherit as many fds as the number of still-running children.
- bpo-29403 [https://bugs.python.org/issue?@action=redirect&bpo=29403]: Fix `unittest.mock`'s autospec to not fail on method-bound builtin functions. Patch by Aaron Gallagher.
- bpo-30961 [https://bugs.python.org/issue?@action=redirect&bpo=30961]: Fix decrementing a borrowed reference in tracemalloc.
- bpo-19896 [https://bugs.python.org/issue?@action=redirect&bpo=19896]: Fix multiprocessing.sharedctypes to recognize typecodes `'q'` and `'Q'`.
- bpo-30946 [https://bugs.python.org/issue?@action=redirect&bpo=30946]: Remove obsolete code in readline module for platforms where GNU readline is older than 2.1 or where select() is not available.
- bpo-25684 [https://bugs.python.org/issue?@action=redirect&bpo=25684]: Change `ttk.OptionMenu` radiobuttons to be unique across instances of `OptionMenu`.
- bpo-30886 [https://bugs.python.org/issue?@action=redirect&bpo=30886]: Fix multiprocessing.Queue.join_thread(): it now waits until the thread completes, even if the thread was started by the same process which created the queue.
- bpo-29854 [https://bugs.python.org/issue?@action=redirect&bpo=29854]: Fix segfault in readline when using readline's history-size option. Patch by Nir Soffer.
- bpo-30794 [https://bugs.python.org/issue?@action=redirect&bpo=30794]: Added multiprocessing.Process.kill method to terminate using the SIGKILL signal on Unix.
- bpo-30319 [https://bugs.python.org/issue?@action=redirect&bpo=30319]: socket.close() now ignores ECONNRESET error.
- bpo-30828 [https://bugs.python.org/issue?@action=redirect&bpo=30828]: Fix out of bounds write in `asyncio.CFuture.remove_done_callback()`.
- bpo-30302 [https://bugs.python.org/issue?

@action=redirect&bpo=30302]: Use keywords in the `repr` of `datetime.timedelta`.

- bpo-30807 [https://bugs.python.org/issue?@action=redirect&bpo=30807]: signal.setitimer() may disable the timer when passed a tiny value. Tiny values (such as 1e-6) are valid non-zero values for setitimer(), which is specified as taking microsecond-resolution intervals. However, on some platform, our conversion routine could convert 1e-6 into a zero interval, therefore disabling the timer instead of (re-)scheduling it.
- bpo-30441 [https://bugs.python.org/issue?@action=redirect&bpo=30441]: Fix bug when modifying os.environ while iterating over it
- bpo-29585 [https://bugs.python.org/issue?@action=redirect&bpo=29585]: Avoid importing `sysconfig` from `site` to improve startup speed. Python startup is about 5% faster on Linux and 30% faster on macOS.
- bpo-29293 [https://bugs.python.org/issue?@action=redirect&bpo=29293]: Add missing parameter «n» on multiprocessing.Condition.notify(). The doc claims multiprocessing.Condition behaves like threading.Condition, but its notify() method lacked the optional «n» argument (to specify the number of sleepers to wake up) that threading.Condition.notify() accepts.
- bpo-30532 [https://bugs.python.org/issue?@action=redirect&bpo=30532]: Fix email header value parser dropping folding white space in certain cases.
- bpo-30596 [https://bugs.python.org/issue?@action=redirect&bpo=30596]: Add a `close()` method to `multiprocessing.Process`.
- bpo-9146 [https://bugs.python.org/issue?@action=redirect&bpo=9146]: Fix a segmentation fault in _hashopenssl when standard hash functions such as md5 are not available in the linked OpenSSL library. As in some special FIPS-140 build environments.
- bpo-29169 [https://bugs.python.org/issue?@action=redirect&bpo=29169]: Update zlib to 1.2.11.
- bpo-30119 [https://bugs.python.org/issue?@action=redirect&bpo=30119]: ftplib.FTP.putline() now throws ValueError on commands that contains CR or LF. Patch by

Donghee Na.

- bpo-30879 [https://bugs.python.org/issue?@action=redirect&bpo=30879]: os.listdir() and os.scandir() now emit bytes names when called with bytes-like argument.
- bpo-30746 [https://bugs.python.org/issue?@action=redirect&bpo=30746]: Prohibited the "=" character in environment variable names in `os.putenv()` and `os.spawn*()`.
- bpo-30664 [https://bugs.python.org/issue?@action=redirect&bpo=30664]: The description of a unittest subtest now preserves the order of keyword arguments of TestCase.subTest().
- bpo-21071 [https://bugs.python.org/issue?@action=redirect&bpo=21071]: struct.Struct.format type is now **`str`** instead of **`bytes`**.
- bpo-29212 [https://bugs.python.org/issue?@action=redirect&bpo=29212]: Fix concurrent.futures.thread.ThreadPoolExecutor threads to have a non repr() based thread name by default when no thread_name_prefix is supplied. They will now identify themselves as «ThreadPoolExecutor-y_n».
- bpo-29755 [https://bugs.python.org/issue?@action=redirect&bpo=29755]: Fixed the lgettext() family of functions in the gettext module. They now always return bytes.
- bpo-30616 [https://bugs.python.org/issue?@action=redirect&bpo=30616]: Functional API of enum allows to create empty enums. Patched by Donghee Na
- bpo-30038 [https://bugs.python.org/issue?@action=redirect&bpo=30038]: Fix race condition between signal delivery and wakeup file descriptor. Patch by Nathaniel Smith.
- bpo-23894 [https://bugs.python.org/issue?@action=redirect&bpo=23894]: lib2to3 now recognizes `rb'...'` and `f'...'` strings.
- bpo-24744 [https://bugs.python.org/issue?@action=redirect&bpo=24744]: pkgutil.walk_packages function now raises ValueError if *path* is a string. Patch by Sanyam Khurana.
- bpo-24484 [https://bugs.python.org/issue?

@action=redirect&bpo=24484]: Avoid race condition in multiprocessing cleanup.

- bpo-30589 [https://bugs.python.org/issue?@action=redirect&bpo=30589]: Fix multiprocessing.Process.exitcode to return the opposite of the signal number when the process is killed by a signal (instead of 255) when using the «forkserver» method.
- bpo-28994 [https://bugs.python.org/issue?@action=redirect&bpo=28994]: The traceback no longer displayed for SystemExit raised in a callback registered by atexit.
- bpo-30508 [https://bugs.python.org/issue?@action=redirect&bpo=30508]: Don't log exceptions if Task/Future «cancel()» method was called.
- bpo-30645 [https://bugs.python.org/issue?@action=redirect&bpo=30645]: Fix path calculation in `imp.load_package()`, fixing it for cases when a package is only shipped with bytecodes. Patch by Alexandru Ardelean.
- bpo-11822 [https://bugs.python.org/issue?@action=redirect&bpo=11822]: The dis.dis() function now is able to disassemble nested code objects.
- bpo-30624 [https://bugs.python.org/issue?@action=redirect&bpo=30624]: selectors does not take KeyboardInterrupt and SystemExit into account, leaving a fd in a bad state in case of error. Patch by Giampaolo Rodola".
- bpo-30595 [https://bugs.python.org/issue?@action=redirect&bpo=30595]: multiprocessing.Queue.get() with a timeout now polls its reader in non-blocking mode if it succeeded to acquire the lock but the acquire took longer than the timeout.
- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Updates to typing module: Add generic AsyncContextManager, add support for ContextManager on all versions. Original PRs by Jelle Zijlstra and Ivan Levkivskyi
- bpo-30605 [https://bugs.python.org/issue?@action=redirect&bpo=30605]: re.compile() no longer raises a BytesWarning when compiling a bytes instance with misplaced inline modifier. Patch by Roy Williams.
- bpo-29870 [https://bugs.python.org/issue?@action=redirect&bpo=29870]: Fix ssl sockets leaks when

connection is aborted in asyncio/ssl implementation. Patch by Michaël Sghaïer.

- bpo-29743 [https://bugs.python.org/issue?@action=redirect&bpo=29743]: Closing transport during handshake process leaks open socket. Patch by Nikolay Kim
- bpo-27585 [https://bugs.python.org/issue?@action=redirect&bpo=27585]: Fix waiter cancellation in asyncio.Lock. Patch by Mathieu Sornay.
- bpo-30014 [https://bugs.python.org/issue?@action=redirect&bpo=30014]: modify() method of poll(), epoll() and devpoll() based classes of selectors module is around 10% faster. Patch by Giampaolo Rodola".
- bpo-30418 [https://bugs.python.org/issue?@action=redirect&bpo=30418]: On Windows, subprocess.Popen.communicate() now also ignore EINVAL on stdin.write() if the child process is still running but closed the pipe.
- bpo-30463 [https://bugs.python.org/issue?@action=redirect&bpo=30463]: Added empty __slots__ to abc.ABC. This allows subclassers to deny __dict__ and __weakref__ creation. Patch by Aaron Hall.
- bpo-30520 [https://bugs.python.org/issue?@action=redirect&bpo=30520]: Loggers are now pickleable.
- bpo-30557 [https://bugs.python.org/issue?@action=redirect&bpo=30557]: faulthandler now correctly filters and displays exception codes on Windows
- bpo-30526 [https://bugs.python.org/issue?@action=redirect&bpo=30526]: Add TextIOWrapper.reconfigure() and a TextIOWrapper.write_through attribute.
- bpo-30245 [https://bugs.python.org/issue?@action=redirect&bpo=30245]: Fix possible overflow when organize struct.pack_into error message. Patch by Yuan Liu.
- bpo-30378 [https://bugs.python.org/issue?@action=redirect&bpo=30378]: Fix the problem that logging.handlers.SysLogHandler cannot handle IPv6 addresses.
- bpo-16500 [https://bugs.python.org/issue?@action=redirect&bpo=16500]: Allow registering at-fork handlers.
- bpo-30470 [https://bugs.python.org/issue?@action=redirect&bpo=30470]: Deprecate invalid ctypes call

protection on Windows. Patch by Mariatta Wijaya.

- bpo-30414 [https://bugs.python.org/issue?@action=redirect&bpo=30414]: multiprocessing.Queue._feed background running thread do not break from main loop on exception.
- bpo-30003 [https://bugs.python.org/issue?@action=redirect&bpo=30003]: Fix handling escape characters in HZ codec. Based on patch by Ma Lin.
- bpo-30149 [https://bugs.python.org/issue?@action=redirect&bpo=30149]: inspect.signature() now supports callables with variable-argument parameters wrapped with partialmethod. Patch by Donghee Na.
- bpo-30436 [https://bugs.python.org/issue?@action=redirect&bpo=30436]: importlib.find_spec() raises ModuleNotFoundError instead of AttributeError if the specified parent module is not a package (i.e. lacks a __path__ attribute).
- bpo-30301 [https://bugs.python.org/issue?@action=redirect&bpo=30301]: Fix AttributeError when using SimpleQueue.empty() under *spawn* and *forkserver* start methods.
- bpo-30375 [https://bugs.python.org/issue?@action=redirect&bpo=30375]: Warnings emitted when compile a regular expression now always point to the line in the user code. Previously they could point into inners of the re module if emitted from inside of groups or conditionals.
- bpo-30329 [https://bugs.python.org/issue?@action=redirect&bpo=30329]: imaplib and poplib now catch the Windows socket WSAEINVAL error (code 10022) on shutdown(SHUT_RDWR): An invalid operation was attempted. This error occurs sometimes on SSL connections.
- bpo-29196 [https://bugs.python.org/issue?@action=redirect&bpo=29196]: Removed previously deprecated in Python 2.4 classes Plist, Dict and _InternalDict in the plistlib module. Dict values in the result of functions readPlist() and readPlistFromBytes() are now normal dicts. You no longer can use attribute access to access items of these dictionaries.
- bpo-9850 [https://bugs.python.org/issue?@action=redirect&bpo=9850]: The `macpath` is now deprecated and will be removed in

Python 3.8.

- bpo-30299 [https://bugs.python.org/issue?@action=redirect&bpo=30299]: Compiling regular expression in debug mode on CPython now displays the compiled bytecode in human readable form.
- bpo-30048 [https://bugs.python.org/issue?@action=redirect&bpo=30048]: Fixed `Task.cancel()` can be ignored when the task is running coroutine and the coroutine returned without any more `await`.
- bpo-30266 [https://bugs.python.org/issue?@action=redirect&bpo=30266]: contextlib.AbstractContextManager now supports anti-registration by setting __enter__ = None or __exit__ = None, following the pattern introduced in bpo-25958 [https://bugs.python.org/issue?@action=redirect&bpo=25958]. Patch by Jelle Zijlstra.
- bpo-30340 [https://bugs.python.org/issue?@action=redirect&bpo=30340]: Enhanced regular expressions optimization. This increased the performance of matching some patterns up to 25 times.
- bpo-30298 [https://bugs.python.org/issue?@action=redirect&bpo=30298]: Weaken the condition of deprecation warnings for inline modifiers. Now allowed several subsequential inline modifiers at the start of the pattern (e.g. `'(?i)(?s)...'`). In verbose mode whitespaces and comments now are allowed before and between inline modifiers (e.g. `'(?x) (?i) (?s)...'`).
- bpo-30285 [https://bugs.python.org/issue?@action=redirect&bpo=30285]: Optimized case-insensitive matching and searching of regular expressions.
- bpo-29990 [https://bugs.python.org/issue?@action=redirect&bpo=29990]: Fix range checking in GB18030 decoder. Original patch by Ma Lin.
- bpo-29979 [https://bugs.python.org/issue?@action=redirect&bpo=29979]: rewrite cgi.parse_multipart, reusing the FieldStorage class and making its results consistent with those of FieldStorage for multipart/form-data requests. Patch by Pierre Quentel.
- bpo-30243 [https://bugs.python.org/issue?@action=redirect&bpo=30243]: Removed the __init__ methods of

_json’s scanner and encoder. Misusing them could cause memory leaks or crashes. Now scanner and encoder objects are completely initialized in the __new__ methods.

- bpo-30215 [https://bugs.python.org/issue?@action=redirect&bpo=30215]: Compiled regular expression objects with the re.LOCALE flag no longer depend on the locale at compile time. Only the locale at matching time affects the result of matching.
- bpo-30185 [https://bugs.python.org/issue?@action=redirect&bpo=30185]: Avoid KeyboardInterrupt tracebacks in forkserver helper process when Ctrl-C is received.
- bpo-30103 [https://bugs.python.org/issue?@action=redirect&bpo=30103]: binascii.b2a_uu() and uu.encode() now support using `` '`' `` as zero instead of space.
- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Various updates to typing module: add typing.NoReturn type, use WrapperDescriptorType, minor bug-fixes. Original PRs by Jim Fasarakis-Hilliard and Ivan Levkivskyi.
- bpo-30205 [https://bugs.python.org/issue?@action=redirect&bpo=30205]: Fix getsockname() for unbound AF_UNIX sockets on Linux.
- bpo-30228 [https://bugs.python.org/issue?@action=redirect&bpo=30228]: The seek() and tell() methods of io.FileIO now set the internal seekable attribute to avoid one syscall on open() (in buffered or text mode).
- bpo-30190 [https://bugs.python.org/issue?@action=redirect&bpo=30190]: unittest’s assertAlmostEqual and assertNotAlmostEqual provide a better message in case of failure which includes the difference between left and right arguments. (patch by Giampaolo Rodola”)
- bpo-30101 [https://bugs.python.org/issue?@action=redirect&bpo=30101]: Add support for curses.A_ITALIC.
- bpo-29822 [https://bugs.python.org/issue?@action=redirect&bpo=29822]: inspect.isabstract() now works during __init_subclass__. Patch by Nate Soares.
- bpo-29960 [https://bugs.python.org/issue?@action=redirect&bpo=29960]: Preserve generator state when _random.Random.setstate() raises an exception. Patch by

Bryan Olson.

- bpo-30070 [https://bugs.python.org/issue?@action=redirect&bpo=30070]: Fixed leaks and crashes in errors handling in the parser module.
- bpo-22352 [https://bugs.python.org/issue?@action=redirect&bpo=22352]: Column widths in the output of dis.dis() are now adjusted for large line numbers and instruction offsets.
- bpo-30061 [https://bugs.python.org/issue?@action=redirect&bpo=30061]: Fixed crashes in IOBase methods __next__() and readlines() when readline() or __next__() respectively return non-sizeable object. Fixed possible other errors caused by not checking results of PyObject_Size(), PySequence_Size(), or PyMapping_Size().
- bpo-30218 [https://bugs.python.org/issue?@action=redirect&bpo=30218]: Fix PathLike support for shutil.unpack_archive. Patch by Jelle Zijlstra.
- bpo-10076 [https://bugs.python.org/issue?@action=redirect&bpo=10076]: Compiled regular expression and match objects in the re module now support copy.copy() and copy.deepcopy() (they are considered atomic).
- bpo-30068 [https://bugs.python.org/issue?@action=redirect&bpo=30068]: _io._IOBase.readlines will check if it's closed first when hint is present.
- bpo-29694 [https://bugs.python.org/issue?@action=redirect&bpo=29694]: Fixed race condition in pathlib mkdir with flags parents=True. Patch by Armin Rigo.
- bpo-29692 [https://bugs.python.org/issue?@action=redirect&bpo=29692]: Fixed arbitrary unchaining of RuntimeError exceptions in contextlib.contextmanager. Patch by Siddharth Velankar.
- bpo-26187 [https://bugs.python.org/issue?@action=redirect&bpo=26187]: Test that sqlite3 trace callback is not called multiple times when schema is changing. Indirectly fixed by switching to use sqlite3_prepare_v2() in bpo-9303 [https://bugs.python.org/issue?@action=redirect&bpo=9303]. Patch by Aviv Palivoda.
- bpo-30017 [https://bugs.python.org/issue?@action=redirect&bpo=30017]: Allowed calling the close() method of the zip entry writer object multiple times. Writing

to a closed writer now always produces a ValueError.

- bpo-29998 [https://bugs.python.org/issue?@action=redirect&bpo=29998]: Pickling and copying ImportError now preserves name and path attributes.
- bpo-29995 [https://bugs.python.org/issue?@action=redirect&bpo=29995]: re.escape() now escapes only regex special characters.
- bpo-29962 [https://bugs.python.org/issue?@action=redirect&bpo=29962]: Add math.remainder operation, implementing remainder as specified in IEEE 754.
- bpo-29649 [https://bugs.python.org/issue?@action=redirect&bpo=29649]: Improve struct.pack_into() exception messages for problems with the buffer size and offset. Patch by Andrew Nester.
- bpo-29654 [https://bugs.python.org/issue?@action=redirect&bpo=29654]: Support If-Modified-Since HTTP header (browser cache). Patch by Pierre Quentel.
- bpo-29931 [https://bugs.python.org/issue?@action=redirect&bpo=29931]: Fixed comparison check for ipaddress.ip_interface objects. Patch by Sanjay Sundaresan.
- bpo-29953 [https://bugs.python.org/issue?@action=redirect&bpo=29953]: Fixed memory leaks in the replace() method of datetime and time objects when pass out of bound fold argument.
- bpo-29942 [https://bugs.python.org/issue?@action=redirect&bpo=29942]: Fix a crash in itertools.chain.from_iterable when encountering long runs of empty iterables.
- bpo-10030 [https://bugs.python.org/issue?@action=redirect&bpo=10030]: Sped up reading encrypted ZIP files by 2 times.
- bpo-29204 [https://bugs.python.org/issue?@action=redirect&bpo=29204]: Element.getiterator() and the html parameter of XMLParser() were deprecated only in the documentation (since Python 3.2 and 3.4 correspondingly). Now using them emits a deprecation warning.
- bpo-27863 [https://bugs.python.org/issue?@action=redirect&bpo=27863]: Fixed multiple crashes in ElementTree caused by race conditions and wrong types.
- bpo-25996 [https://bugs.python.org/issue?

@action=redirect&bpo=25996]: Added support of file descriptors in os.scandir() on Unix. os.fwalk() is sped up by 2 times by using os.scandir().

- bpo-28699 [https://bugs.python.org/issue?@action=redirect&bpo=28699]: Fixed a bug in pools in multiprocessing.pool that raising an exception at the very first of an iterable may swallow the exception or make the program hang. Patch by Davin Potts and Xiang Zhang.
- bpo-23890 [https://bugs.python.org/issue?@action=redirect&bpo=23890]: unittest.TestCase.assertRaises() now manually breaks a reference cycle to not keep objects alive longer than expected.
- bpo-29901 [https://bugs.python.org/issue?@action=redirect&bpo=29901]: The zipapp module now supports general path-like objects, not just pathlib.Path.
- bpo-25803 [https://bugs.python.org/issue?@action=redirect&bpo=25803]: Avoid incorrect errors raised by Path.mkdir(exist_ok=True) when the OS gives priority to errors such as EACCES over EEXIST.
- bpo-29861 [https://bugs.python.org/issue?@action=redirect&bpo=29861]: Release references to tasks, their arguments and their results as soon as they are finished in multiprocessing.Pool.
- bpo-19930 [https://bugs.python.org/issue?@action=redirect&bpo=19930]: The mode argument of os.makedirs() no longer affects the file permission bits of newly created intermediate-level directories.
- bpo-29884 [https://bugs.python.org/issue?@action=redirect&bpo=29884]: faulthandler: Restore the old sigaltstack during teardown. Patch by Christophe Zeitouny.
- bpo-25455 [https://bugs.python.org/issue?@action=redirect&bpo=25455]: Fixed crashes in repr of recursive buffered file-like objects.
- bpo-29800 [https://bugs.python.org/issue?@action=redirect&bpo=29800]: Fix crashes in partial.__repr__ if the keys of partial.keywords are not strings. Patch by Michael Seifert.
- bpo-8256 [https://bugs.python.org/issue?@action=redirect&bpo=8256]: Fixed possible failing or crashing input() if attributes «encoding» or «errors» of sys.stdin or sys.stdout are not set or

are not strings.

- bpo-28692 [https://bugs.python.org/issue?@action=redirect&bpo=28692]: Using non-integer value for selecting a plural form in gettext is now deprecated.
- bpo-26121 [https://bugs.python.org/issue?@action=redirect&bpo=26121]: Use C library implementation for math functions erf() and erfc().
- bpo-29619 [https://bugs.python.org/issue?@action=redirect&bpo=29619]: os.stat() and os.DirEntry.inode() now convert inode (st_ino) using unsigned integers.
- bpo-28298 [https://bugs.python.org/issue?@action=redirect&bpo=28298]: Fix a bug that prevented array "Q", "L" and "I" from accepting big intables (objects that have _int_) as elements.
- bpo-29645 [https://bugs.python.org/issue?@action=redirect&bpo=29645]: Speed up importing the webbrowser module. webbrowser.register() is now thread-safe.
- bpo-28231 [https://bugs.python.org/issue?@action=redirect&bpo=28231]: The zipfile module now accepts path-like objects for external paths.
- bpo-26915 [https://bugs.python.org/issue?@action=redirect&bpo=26915]: index() and count() methods of collections.abc.Sequence now check identity before checking equality when do comparisons.
- bpo-28682 [https://bugs.python.org/issue?@action=redirect&bpo=28682]: Added support for bytes paths in os.fwalk().
- bpo-29728 [https://bugs.python.org/issue?@action=redirect&bpo=29728]: Add new **`socket.TCP_NOTSENT_LOWAT`** (Linux 3.12) constant. Patch by Nathaniel J. Smith.
- bpo-29623 [https://bugs.python.org/issue?@action=redirect&bpo=29623]: Allow use of path-like object as a single argument in ConfigParser.read(). Patch by David Ellis.
- bpo-9303 [https://bugs.python.org/issue?@action=redirect&bpo=9303]: Migrate sqlite3 module to _v2 API. Patch by Aviv Palivoda.
- bpo-28963 [https://bugs.python.org/issue?@action=redirect&bpo=28963]: Fix out of bound iteration in asyncio.Future.remove_done_callback implemented in C.

- bpo-29704 [https://bugs.python.org/issue?@action=redirect&bpo=29704]: asyncio.subprocess.SubprocessStreamProtocol no longer closes before all pipes are closed.
- bpo-29271 [https://bugs.python.org/issue?@action=redirect&bpo=29271]: Fix Task.current_task and Task.all_tasks implemented in C to accept None argument as their pure Python implementation.
- bpo-29703 [https://bugs.python.org/issue?@action=redirect&bpo=29703]: Fix asyncio to support instantiation of new event loops in child processes.
- bpo-29615 [https://bugs.python.org/issue?@action=redirect&bpo=29615]: SimpleXMLRPCDispatcher no longer chains KeyError (or any other exception) to exception(s) raised in the dispatched methods. Patch by Petr Motejlek.
- bpo-7769 [https://bugs.python.org/issue?@action=redirect&bpo=7769]: Method register_function() of xmlrpc.server.SimpleXMLRPCDispatcher and its subclasses can now be used as a decorator.
- bpo-29376 [https://bugs.python.org/issue?@action=redirect&bpo=29376]: Fix assertion error in threading._DummyThread.is_alive().
- bpo-28624 [https://bugs.python.org/issue?@action=redirect&bpo=28624]: Add a test that checks that cwd parameter of Popen() accepts PathLike objects. Patch by Sayan Chowdhury.
- bpo-28518 [https://bugs.python.org/issue?@action=redirect&bpo=28518]: Start a transaction implicitly before a DML statement. Patch by Aviv Palivoda.
- bpo-29742 [https://bugs.python.org/issue?@action=redirect&bpo=29742]: get_extra_info() raises exception if get called on closed ssl transport. Patch by Nikolay Kim.
- bpo-16285 [https://bugs.python.org/issue?@action=redirect&bpo=16285]: urllib.parse.quote is now based on RFC 3986 and hence includes “~” in the set of characters that is not quoted by default. Patch by Christian Theune and Ratnadeep Debnath.
- bpo-29532 [https://bugs.python.org/issue?@action=redirect&bpo=29532]: Altering a kwarg dictionary

passed to functools.partial() no longer affects a partial object after creation.

- bpo-29110 [https://bugs.python.org/issue?@action=redirect&bpo=29110]: Fix file object leak in aifc.open() when file is given as a filesystem path and is not in valid AIFF format. Patch by Anthony Zhang.
- bpo-22807 [https://bugs.python.org/issue?@action=redirect&bpo=22807]: Add uuid.SafeUUID and uuid.UUID.is_safe to relay information from the platform about whether generated UUIDs are generated with a multiprocessing safe method.
- bpo-29576 [https://bugs.python.org/issue?@action=redirect&bpo=29576]: Improve some deprecations in importlib. Some deprecated methods now emit DeprecationWarnings and have better descriptive messages.
- bpo-29534 [https://bugs.python.org/issue?@action=redirect&bpo=29534]: Fixed different behaviour of Decimal.from_float() for _decimal and _pydecimal. Thanks Andrew Nester.
- bpo-10379 [https://bugs.python.org/issue?@action=redirect&bpo=10379]: locale.format_string now supports the “monetary” keyword argument, and locale.format is deprecated.
- bpo-29851 [https://bugs.python.org/issue?@action=redirect&bpo=29851]: importlib.reload() now raises ModuleNotFoundError if the module lacks a spec.
- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Various updates to typing module: typing.Counter, typing.ChainMap, improved ABC caching, etc. Original PRs by Jelle Zijlstra, Ivan Levkivskyi, Manuel Krebber, and Łukasz Langa.
- bpo-29100 [https://bugs.python.org/issue?@action=redirect&bpo=29100]: Fix datetime.fromtimestamp() regression introduced in Python 3.6.0: check minimum and maximum years.
- bpo-29416 [https://bugs.python.org/issue?@action=redirect&bpo=29416]: Prevent infinite loop in pathlib.Path.mkdir
- bpo-29444 [https://bugs.python.org/issue?@action=redirect&bpo=29444]: Fixed out-of-bounds buffer access

in the group() method of the match object. Based on patch by WGH.

- bpo-29377 [https://bugs.python.org/issue?@action=redirect&bpo=29377]: Add WrapperDescriptorType, MethodWrapperType, and MethodDescriptorType built-in types to types module. Original patch by Manuel Krebber.
- bpo-29218 [https://bugs.python.org/issue?@action=redirect&bpo=29218]: Unused install_misc command is now removed. It has been documented as unused since 2000. Patch by Eric N. Vander Weele.
- bpo-29368 [https://bugs.python.org/issue?@action=redirect&bpo=29368]: The extend() method is now called instead of the append() method when unpickle collections.deque and other list-like objects. This can speed up unpickling to 2 times.
- bpo-29338 [https://bugs.python.org/issue?@action=redirect&bpo=29338]: The help of a builtin or extension class now includes the constructor signature if __text_signature__ is provided for the class.
- bpo-29335 [https://bugs.python.org/issue?@action=redirect&bpo=29335]: Fix subprocess.Popen.wait() when the child process has exited to a stopped instead of terminated state (ex: when under ptrace).
- bpo-29290 [https://bugs.python.org/issue?@action=redirect&bpo=29290]: Fix a regression in argparse that help messages would wrap at non-breaking spaces.
- bpo-28735 [https://bugs.python.org/issue?@action=redirect&bpo=28735]: Fixed the comparison of mock.MagickMock with mock.ANY.
- bpo-29197 [https://bugs.python.org/issue?@action=redirect&bpo=29197]: Removed deprecated function ntpath.splitunc().
- bpo-29210 [https://bugs.python.org/issue?@action=redirect&bpo=29210]: Removed support of deprecated argument «exclude» in tarfile.TarFile.add().
- bpo-29219 [https://bugs.python.org/issue?@action=redirect&bpo=29219]: Fixed infinite recursion in the repr of uninitialized ctypes.CDLL instances.
- bpo-29192 [https://bugs.python.org/issue?@action=redirect&bpo=29192]: Removed deprecated features in

the http.cookies module.

- bpo-29193 [https://bugs.python.org/issue?@action=redirect&bpo=29193]: A format string argument for string.Formatter.format() is now positional-only.
- bpo-29195 [https://bugs.python.org/issue?@action=redirect&bpo=29195]: Removed support of deprecated undocumented keyword arguments in methods of regular expression objects.
- bpo-28969 [https://bugs.python.org/issue?@action=redirect&bpo=28969]: Fixed race condition in C implementation of functools.lru_cache. KeyError could be raised when cached function with full cache was simultaneously called from different threads with the same uncached arguments.
- bpo-20804 [https://bugs.python.org/issue?@action=redirect&bpo=20804]: The unittest.mock.sentinel attributes now preserve their identity when they are copied or pickled.
- bpo-29142 [https://bugs.python.org/issue?@action=redirect&bpo=29142]: In urllib.request, suffixes in no_proxy environment variable with leading dots could match related hostnames again (e.g. .b.c matches a.b.c). Patch by Milan Oberkirch.
- bpo-28961 [https://bugs.python.org/issue?@action=redirect&bpo=28961]: Fix unittest.mock._Call helper: don’t ignore the name parameter anymore. Patch written by Jiajun Huang.
- bpo-15812 [https://bugs.python.org/issue?@action=redirect&bpo=15812]: inspect.getframeinfo() now correctly shows the first line of a context. Patch by Sam Breese.
- bpo-28985 [https://bugs.python.org/issue?@action=redirect&bpo=28985]: Update authorizer constants in sqlite3 module. Patch by Dingyuan Wang.
- bpo-29079 [https://bugs.python.org/issue?@action=redirect&bpo=29079]: Prevent infinite loop in pathlib.resolve() on Windows
- bpo-13051 [https://bugs.python.org/issue?@action=redirect&bpo=13051]: Fixed recursion errors in large or resized curses.textpad.Textbox. Based on patch by Tycho

Andersen.

- bpo-9770 [https://bugs.python.org/issue?@action=redirect&bpo=9770]: curses.ascii predicates now work correctly with negative integers.
- bpo-28427 [https://bugs.python.org/issue?@action=redirect&bpo=28427]: old keys should not remove new values from WeakValueDictionary when collecting from another thread.
- bpo-28923 [https://bugs.python.org/issue?@action=redirect&bpo=28923]: Remove editor artifacts from Tix.py.
- bpo-28871 [https://bugs.python.org/issue?@action=redirect&bpo=28871]: Fixed a crash when deallocate deep ElementTree.
- bpo-19542 [https://bugs.python.org/issue?@action=redirect&bpo=19542]: Fix bugs in WeakValueDictionary.setdefault() and WeakValueDictionary.pop() when a GC collection happens in another thread.
- bpo-20191 [https://bugs.python.org/issue?@action=redirect&bpo=20191]: Fixed a crash in resource.prlimit() when passing a sequence that doesn't own its elements as limits.
- bpo-16255 [https://bugs.python.org/issue?@action=redirect&bpo=16255]: subprocess.Popen uses /system/bin/sh on Android as the shell, instead of /bin/sh.
- bpo-28779 [https://bugs.python.org/issue?@action=redirect&bpo=28779]: multiprocessing.set_forkserver_preload() would crash the forkserver process if a preloaded module instantiated some multiprocessing objects such as locks.
- bpo-26937 [https://bugs.python.org/issue?@action=redirect&bpo=26937]: The chown() method of the tarfile.TarFile class does not fail now when the grp module cannot be imported, as for example on Android platforms.
- bpo-28847 [https://bugs.python.org/issue?@action=redirect&bpo=28847]: dbm.dumb now supports reading read-only files and no longer writes the index file when it is not changed. A deprecation warning is now emitted if the index file is missed and recreated in the “r” and “w” modes

(will be an error in future Python releases).

- bpo-27030 [https://bugs.python.org/issue?@action=redirect&bpo=27030]: Unknown escapes consisting of `'\'` and an ASCII letter in re.sub() replacement templates regular expressions now are errors.
- bpo-28835 [https://bugs.python.org/issue?@action=redirect&bpo=28835]: Fix a regression introduced in warnings.catch_warnings(): call warnings.showwarning() if it was overridden inside the context manager.
- bpo-27172 [https://bugs.python.org/issue?@action=redirect&bpo=27172]: To assist with upgrades from 2.7, the previously documented deprecation of `inspect.getfullargspec()` has been reversed. This decision may be revisited again after the Python 2.7 branch is no longer officially supported.
- bpo-28740 [https://bugs.python.org/issue?@action=redirect&bpo=28740]: Add sys.getandroidapilevel(): return the build time API version of Android as an integer. Function only available on Android.
- bpo-26273 [https://bugs.python.org/issue?@action=redirect&bpo=26273]: Add new **`socket.TCP_CONGESTION`** (Linux 2.6.13) and **`socket.TCP_USER_TIMEOUT`** (Linux 2.6.37) constants. Patch written by Omar Sandoval.
- bpo-28752 [https://bugs.python.org/issue?@action=redirect&bpo=28752]: Restored the __reduce__() methods of datetime objects.
- bpo-28727 [https://bugs.python.org/issue?@action=redirect&bpo=28727]: Regular expression patterns, _sre.SRE_Pattern objects created by re.compile(), become comparable (only x==y and x!=y operators). This change should fix the bpo-18383 [https://bugs.python.org/issue?@action=redirect&bpo=18383]: don't duplicate warning filters when the warnings module is reloaded (thing usually only done in unit tests).
- bpo-20572 [https://bugs.python.org/issue?@action=redirect&bpo=20572]: Remove the subprocess.Popen.wait endtime parameter. It was deprecated in 3.4 and undocumented prior to that.
- bpo-25659 [https://bugs.python.org/issue?

@action=redirect&bpo=25659]: In ctypes, prevent a crash calling the from_buffer() and from_buffer_copy() methods on abstract classes like Array.

- bpo-28548 [https://bugs.python.org/issue?@action=redirect&bpo=28548]: In the «http.server» module, parse the protocol version if possible, to avoid using HTTP 0.9 in some error responses.
- bpo-19717 [https://bugs.python.org/issue?@action=redirect&bpo=19717]: Makes Path.resolve() succeed on paths that do not exist. Patch by Vajrasky Kok
- bpo-28563 [https://bugs.python.org/issue?@action=redirect&bpo=28563]: Fixed possible DoS and arbitrary code execution when handle plural form selections in the gettext module. The expression parser now supports exact syntax supported by GNU gettext.
- bpo-28387 [https://bugs.python.org/issue?@action=redirect&bpo=28387]: Fixed possible crash in _io.TextIOWrapper deallocator when the garbage collector is invoked in other thread. Based on patch by Sebastian Cufre.
- bpo-27517 [https://bugs.python.org/issue?@action=redirect&bpo=27517]: LZMA compressor and decompressor no longer raise exceptions if given empty data twice. Patch by Benjamin Fogle.
- bpo-28549 [https://bugs.python.org/issue?@action=redirect&bpo=28549]: Fixed segfault in curses's addch() with ncurses6.
- bpo-28449 [https://bugs.python.org/issue?@action=redirect&bpo=28449]: tarfile.open() with mode «r» or «r:» now tries to open a tar file with compression before trying to open it without compression. Otherwise it had 50% chance failed with ignore_zeros=True.
- bpo-23262 [https://bugs.python.org/issue?@action=redirect&bpo=23262]: The webbrowser module now supports Firefox 36+ and derived browsers. Based on patch by Oleg Broytman.
- bpo-24241 [https://bugs.python.org/issue?@action=redirect&bpo=24241]: The webbrowser in an X environment now prefers using the default browser directly. Also, the webbrowser register() function now has a documented “preferred” argument, to specify browsers to be

returned by get() with no arguments. Patch by David Steele

- bpo-27939 [https://bugs.python.org/issue?@action=redirect&bpo=27939]: Fixed bugs in tkinter.ttk.LabeledScale and tkinter.Scale caused by representing the scale as float value internally in Tk. tkinter.IntVar now works if float value is set to underlying Tk variable.
- bpo-28255 [https://bugs.python.org/issue?@action=redirect&bpo=28255]: calendar.TextCalendar.prweek() no longer prints a space after a weeks's calendar. calendar.TextCalendar.pryear() no longer prints redundant newline after a year's calendar. Based on patch by Xiang Zhang.
- bpo-28255 [https://bugs.python.org/issue?@action=redirect&bpo=28255]: calendar.TextCalendar.prmonth() no longer prints a space at the start of new line after printing a month's calendar. Patch by Xiang Zhang.
- bpo-20491 [https://bugs.python.org/issue?@action=redirect&bpo=20491]: The textwrap.TextWrapper class now honors non-breaking spaces. Based on patch by Kaarle Ritvanen.
- bpo-28353 [https://bugs.python.org/issue?@action=redirect&bpo=28353]: os.fwalk() no longer fails on broken links.
- bpo-28430 [https://bugs.python.org/issue?@action=redirect&bpo=28430]: Fix iterator of C implemented asyncio.Future doesn't accept non-None value is passed to it.send(val).
- bpo-27025 [https://bugs.python.org/issue?@action=redirect&bpo=27025]: Generated names for Tkinter widgets now start by the «!» prefix for readability.
- bpo-25464 [https://bugs.python.org/issue?@action=redirect&bpo=25464]: Fixed HList.header_exists() in tkinter.tix module by addin a workaround to Tix library bug.
- bpo-28488 [https://bugs.python.org/issue?@action=redirect&bpo=28488]: shutil.make_archive() no longer adds entry «./» to ZIP archive.
- bpo-25953 [https://bugs.python.org/issue?@action=redirect&bpo=25953]: re.sub() now raises an error for invalid numerical group reference in replacement template

even if the pattern is not found in the string. Error message for invalid group reference now includes the group index and the position of the reference. Based on patch by SilentGhost.

- bpo-28469 [https://bugs.python.org/issue?@action=redirect&bpo=28469]: timeit now uses the sequence 1, 2, 5, 10, 20, 50,… instead of 1, 10, 100,… for autoranging.
- bpo-28115 [https://bugs.python.org/issue?@action=redirect&bpo=28115]: Command-line interface of the zipfile module now uses argparse. Added support of long options.
- bpo-18219 [https://bugs.python.org/issue?@action=redirect&bpo=18219]: Optimize csv.DictWriter for large number of columns. Patch by Mariatta Wijaya.
- bpo-28448 [https://bugs.python.org/issue?@action=redirect&bpo=28448]: Fix C implemented asyncio.Future didn’t work on Windows.
- bpo-23214 [https://bugs.python.org/issue?@action=redirect&bpo=23214]: In the «io» module, the argument to BufferedReader and BytesIO’s read1() methods is now optional and can be -1, matching the BufferedIOBase specification.
- bpo-28480 [https://bugs.python.org/issue?@action=redirect&bpo=28480]: Fix error building socket module when multithreading is disabled.
- bpo-28240 [https://bugs.python.org/issue?@action=redirect&bpo=28240]: timeit: remove `-c/--clock` and `-t/--time` command line options which were deprecated since Python 3.3.
- bpo-28240 [https://bugs.python.org/issue?@action=redirect&bpo=28240]: timeit now repeats the benchmarks 5 times instead of only 3 to make benchmarks more reliable.
- bpo-28240 [https://bugs.python.org/issue?@action=redirect&bpo=28240]: timeit autorange now uses a single loop iteration if the benchmark takes less than 10 seconds, instead of 10 iterations. «python3 -m timeit -s “import time” “time.sleep(1)”» now takes 4 seconds instead of 40 seconds.
- Distutils.sdist now looks for README and setup.py files with case sensitivity. This behavior matches that found in

Setuptools 6.0 and later. See setuptools 100 [https://github.com/pypa/setuptools/issues/100] for rationale.

- bpo-24452 [https://bugs.python.org/issue?@action=redirect&bpo=24452]: Make webbrowser support Chrome on Mac OS X. Patch by Ned Batchelder.
- bpo-20766 [https://bugs.python.org/issue?@action=redirect&bpo=20766]: Fix references leaked by pdb in the handling of SIGINT handlers.
- bpo-27998 [https://bugs.python.org/issue?@action=redirect&bpo=27998]: Fixed bytes path support in os.scandir() on Windows. Patch by Eryk Sun.
- bpo-28317 [https://bugs.python.org/issue?@action=redirect&bpo=28317]: The disassembler now decodes FORMAT_VALUE argument.
- bpo-28380 [https://bugs.python.org/issue?@action=redirect&bpo=28380]: unittest.mock Mock autospec functions now properly support assert_called, assert_not_called, and assert_called_once.
- bpo-28229 [https://bugs.python.org/issue?@action=redirect&bpo=28229]: lzma module now supports pathlib.
- bpo-28321 [https://bugs.python.org/issue?@action=redirect&bpo=28321]: Fixed writing non-BMP characters with binary format in plistlib.
- bpo-28225 [https://bugs.python.org/issue?@action=redirect&bpo=28225]: bz2 module now supports pathlib. Initial patch by Ethan Furman.
- bpo-28227 [https://bugs.python.org/issue?@action=redirect&bpo=28227]: gzip now supports pathlib. Patch by Ethan Furman.
- bpo-28332 [https://bugs.python.org/issue?@action=redirect&bpo=28332]: Deprecated silent truncations in socket.htons and socket.ntohs. Original patch by Oren Milman.
- bpo-27358 [https://bugs.python.org/issue?@action=redirect&bpo=27358]: Optimized merging var-keyword arguments and improved error message when passing a non-mapping as a var-keyword argument.
- bpo-28257 [https://bugs.python.org/issue?@action=redirect&bpo=28257]: Improved error message when

passing a non-iterable as a var-positional argument. Added opcode BUILD_TUPLE_UNPACK_WITH_CALL.

- bpo-28322 [https://bugs.python.org/issue?@action=redirect&bpo=28322]: Fixed possible crashes when unpickle itertools objects from incorrect pickle data. Based on patch by John Leitch.
- bpo-28228 [https://bugs.python.org/issue?@action=redirect&bpo=28228]: imghdr now supports pathlib.
- bpo-28226 [https://bugs.python.org/issue?@action=redirect&bpo=28226]: compileall now supports pathlib.
- bpo-28314 [https://bugs.python.org/issue?@action=redirect&bpo=28314]: Fix function declaration (C flags) for the getiterator() method of xml.etree.ElementTree.Element.
- bpo-28148 [https://bugs.python.org/issue?@action=redirect&bpo=28148]: Stop using localtime() and gmtime() in the time module. Introduced platform independent _PyTime_localtime API that is similar to POSIX localtime_r, but available on all platforms. Patch by Ed Schouten.
- bpo-28253 [https://bugs.python.org/issue?@action=redirect&bpo=28253]: Fixed calendar functions for extreme months: 0001-01 and 9999-12. Methods itermonthdays() and itermonthdays2() are reimplemented so that they don’t call itermonthdates() which can cause datetime.date under/overflow.
- bpo-28275 [https://bugs.python.org/issue?@action=redirect&bpo=28275]: Fixed possible use after free in the decompress() methods of the LZMADecompressor and BZ2Decompressor classes. Original patch by John Leitch.
- bpo-27897 [https://bugs.python.org/issue?@action=redirect&bpo=27897]: Fixed possible crash in sqlite3.Connection.create_collation() if pass invalid string-like object as a name. Patch by Xiang Zhang.
- bpo-18844 [https://bugs.python.org/issue?@action=redirect&bpo=18844]: random.choices() now has k as a keyword-only argument to improve the readability of common cases and come into line with the signature used in other languages.
- bpo-18893 [https://bugs.python.org/issue?

@action=redirect&bpo=18893]: Fix invalid exception handling in Lib/ctypes/macholib/dyld.py. Patch by Madison May.

- bpo-27611 [https://bugs.python.org/issue?@action=redirect&bpo=27611]: Fixed support of default root window in the tkinter.tix module. Added the master parameter in the DisplayStyle constructor.
- bpo-27348 [https://bugs.python.org/issue?@action=redirect&bpo=27348]: In the traceback module, restore the formatting of exception messages like «Exception: None». This fixes a regression introduced in 3.5a2.
- bpo-25651 [https://bugs.python.org/issue?@action=redirect&bpo=25651]: Allow false values to be used for msg parameter of subTest().
- bpo-27778 [https://bugs.python.org/issue?@action=redirect&bpo=27778]: Fix a memory leak in os.getrandom() when the getrandom() is interrupted by a signal and a signal handler raises a Python exception.
- bpo-28200 [https://bugs.python.org/issue?@action=redirect&bpo=28200]: Fix memory leak on Windows in the os module (fix path_converter() function).
- bpo-25400 [https://bugs.python.org/issue?@action=redirect&bpo=25400]: RobotFileParser now correctly returns default values for crawl_delay and request_rate. Initial patch by Peter Wirtz.
- bpo-27932 [https://bugs.python.org/issue?@action=redirect&bpo=27932]: Prevent memory leak in win32_ver().
- Fix UnboundLocalError in socket._sendfile_use_sendfile.
- bpo-28075 [https://bugs.python.org/issue?@action=redirect&bpo=28075]: Check for ERROR_ACCESS_DENIED in Windows implementation of os.stat(). Patch by Eryk Sun.
- bpo-22493 [https://bugs.python.org/issue?@action=redirect&bpo=22493]: Warning message emitted by using inline flags in the middle of regular expression now contains a (truncated) regex pattern. Patch by Tim Graham.
- bpo-25270 [https://bugs.python.org/issue?@action=redirect&bpo=25270]: Prevent codecs.escape_encode() from raising SystemError when an empty bytestring is passed.
- bpo-28181 [https://bugs.python.org/issue?

@action=redirect&bpo=28181]: Get antigravity over HTTPS. Patch by Kaartic Sivaraam.

- bpo-25895 [https://bugs.python.org/issue?@action=redirect&bpo=25895]: Enable WebSocket URL schemes in urllib.parse.urljoin. Patch by Gergely Imreh and Markus Holtermann.
- bpo-28114 [https://bugs.python.org/issue?@action=redirect&bpo=28114]: Fix a crash in parse_envlist() when env contains byte strings. Patch by Eryk Sun.
- bpo-27599 [https://bugs.python.org/issue?@action=redirect&bpo=27599]: Fixed buffer overrun in binascii.b2a_qp() and binascii.a2b_qp().
- bpo-27906 [https://bugs.python.org/issue?@action=redirect&bpo=27906]: Fix socket accept exhaustion during high TCP traffic. Patch by Kevin Conway.
- bpo-28174 [https://bugs.python.org/issue?@action=redirect&bpo=28174]: Handle when SO_REUSEPORT isn't properly supported. Patch by Seth Michael Larson.
- bpo-26654 [https://bugs.python.org/issue?@action=redirect&bpo=26654]: Inspect functools.partial in asyncio.Handle.__repr__. Patch by iceboy.
- bpo-26909 [https://bugs.python.org/issue?@action=redirect&bpo=26909]: Fix slow pipes IO in asyncio. Patch by INADA Naoki.
- bpo-28176 [https://bugs.python.org/issue?@action=redirect&bpo=28176]: Fix callbacks race in asyncio.SelectorLoop.sock_connect.
- bpo-27759 [https://bugs.python.org/issue?@action=redirect&bpo=27759]: Fix selectors incorrectly retain invalid file descriptors. Patch by Mark Williams.
- bpo-28325 [https://bugs.python.org/issue?@action=redirect&bpo=28325]: Remove vestigial MacOS 9 macurl2path module and its tests.
- bpo-28368 [https://bugs.python.org/issue?@action=redirect&bpo=28368]: Refuse monitoring processes if the child watcher has no loop attached. Patch by Vincent Michel.
- bpo-28369 [https://bugs.python.org/issue?@action=redirect&bpo=28369]: Raise RuntimeError when transport's FD is used with add_reader, add_writer, etc.
- bpo-28370 [https://bugs.python.org/issue?

@action=redirect&bpo=28370]: Speedup asyncio.StreamReader.readexactly. Patch by Коренберг Марк.
- bpo-28371 [https://bugs.python.org/issue?@action=redirect&bpo=28371]: Deprecate passing asyncio.Handles to run_in_executor.
- bpo-28372 [https://bugs.python.org/issue?@action=redirect&bpo=28372]: Fix asyncio to support formatting of non-python coroutines.
- bpo-28399 [https://bugs.python.org/issue?@action=redirect&bpo=28399]: Remove UNIX socket from FS before binding. Patch by Коренберг Марк.
- bpo-27972 [https://bugs.python.org/issue?@action=redirect&bpo=27972]: Prohibit Tasks to await on themselves.
- bpo-24142 [https://bugs.python.org/issue?@action=redirect&bpo=24142]: Reading a corrupt config file left configparser in an invalid state. Original patch by Florian Höch.
- bpo-29581 [https://bugs.python.org/issue?@action=redirect&bpo=29581]: ABCMeta.__new__ now accepts `**kwargs`, allowing abstract base classes to use keyword parameters in __init_subclass__. Patch by Nate Soares.
- bpo-25532 [https://bugs.python.org/issue?@action=redirect&bpo=25532]: inspect.unwrap() will now only try to unwrap an object sys.getrecursionlimit() times, to protect against objects which create a new object on every attribute access.
- bpo-30177 [https://bugs.python.org/issue?@action=redirect&bpo=30177]: path.resolve(strict=False) no longer cuts the path after the first element not present in the filesystem. Patch by Antoine Pietri.

Documentation

- bpo-31294 [https://bugs.python.org/issue?@action=redirect&bpo=31294]: Fix incomplete code snippet in the ZeroMQSocketListener and ZeroMQSocketHandler examples and adapt them to Python 3.
- bpo-21649 [https://bugs.python.org/issue?

@action=redirect&bpo=21649]: Add RFC 7525 and Mozilla server side TLS links to SSL documentation.

- bpo-31128 [https://bugs.python.org/issue?@action=redirect&bpo=31128]: Allow the pydoc server to bind to arbitrary hostnames.
- bpo-30803 [https://bugs.python.org/issue?@action=redirect&bpo=30803]: Clarify doc on truth value testing. Original patch by Peter Thomassen.
- bpo-30176 [https://bugs.python.org/issue?@action=redirect&bpo=30176]: Add missing attribute related constants in curses documentation.
- bpo-30052 [https://bugs.python.org/issue?@action=redirect&bpo=30052]: the link targets for **`bytes()`** and **`bytearray()`** are now their respective type definitions, rather than the corresponding builtin function entries. Use bytes and bytearray to reference the latter. In order to ensure this and future cross-reference updates are applied automatically, the daily documentation builds now disable the default output caching features in Sphinx.
- bpo-26985 [https://bugs.python.org/issue?@action=redirect&bpo=26985]: Add missing info of code object in inspect documentation.
- bpo-19824 [https://bugs.python.org/issue?@action=redirect&bpo=19824]: Improve the documentation for, and links to, template strings by emphasizing their utility for internationalization, and by clarifying some usage constraints. (See also: bpo-20314 [https://bugs.python.org/issue?@action=redirect&bpo=20314], bpo-12518 [https://bugs.python.org/issue?@action=redirect&bpo=12518])
- bpo-28929 [https://bugs.python.org/issue?@action=redirect&bpo=28929]: Link the documentation to its source file on GitHub.
- bpo-25008 [https://bugs.python.org/issue?@action=redirect&bpo=25008]: Document smtpd.py as effectively deprecated and add a pointer to aiosmtpd, a third-party asyncio-based replacement.
- bpo-26355 [https://bugs.python.org/issue?@action=redirect&bpo=26355]: Add canonical header link on each page to corresponding major version of the documentation. Patch by Matthias Bussonnier.

- bpo-29349 [https://bugs.python.org/issue?@action=redirect&bpo=29349]: Fix Python 2 syntax in code for building the documentation.
- bpo-23722 [https://bugs.python.org/issue?@action=redirect&bpo=23722]: The data model reference and the porting section in the 3.6 What's New guide now cover the additional `__classcell__` handling needed for custom metaclasses to fully support **PEP 487** [https://peps.python.org/pep-0487/] and zero-argument `super()`.
- bpo-28513 [https://bugs.python.org/issue?@action=redirect&bpo=28513]: Documented command-line interface of zipfile.

Tests

- bpo-29639 [https://bugs.python.org/issue?@action=redirect&bpo=29639]: test.support.HOST is now «localhost», a new HOSTv4 constant has been added for your `127.0.0.1` needs, similar to the existing HOSTv6 constant.
- bpo-31320 [https://bugs.python.org/issue?@action=redirect&bpo=31320]: Silence traceback in test_ssl
- bpo-31346 [https://bugs.python.org/issue?@action=redirect&bpo=31346]: Prefer PROTOCOL_TLS_CLIENT and PROTOCOL_TLS_SERVER protocols for SSLContext.
- bpo-25674 [https://bugs.python.org/issue?@action=redirect&bpo=25674]: Remove sha256.tbs-internet.com ssl test
- bpo-30715 [https://bugs.python.org/issue?@action=redirect&bpo=30715]: Address ALPN callback changes for OpenSSL 1.1.0f. The latest version behaves like OpenSSL 1.0.2 and no longer aborts handshake.
- bpo-30822 [https://bugs.python.org/issue?@action=redirect&bpo=30822]: regrtest: Exclude tzdata from regrtest –all. When running the test suite using –use=all / -u all, exclude tzdata since it makes test_datetime too slow (15-20 min on some buildbots) which then times out on some buildbots. Fix also regrtest command line parser to allow passing -u extralargefile to run test_zipfile64.
- bpo-30695 [https://bugs.python.org/issue?@action=redirect&bpo=30695]: Add the

`set_nomemory(start, stop)` and `remove_mem_hooks()` functions to the `_testcapi` module.

- bpo-30357 [https://bugs.python.org/issue?@action=redirect&bpo=30357]: test_thread: setUp() now uses support.threading_setup() and support.threading_cleanup() to wait until threads complete to avoid random side effects on following tests. Initial patch written by Grzegorz Grzywacz.
- bpo-30197 [https://bugs.python.org/issue?@action=redirect&bpo=30197]: Enhanced functions swap_attr() and swap_item() in the test.support module. They now work when delete replaced attribute or item inside the with statement. The old value of the attribute or item (or None if it doesn't exist) now will be assigned to the target of the «as» clause, if there is one.
- bpo-24932 [https://bugs.python.org/issue?@action=redirect&bpo=24932]: Use proper command line parsing in _testembed
- bpo-28950 [https://bugs.python.org/issue?@action=redirect&bpo=28950]: Disallow -j0 to be combined with -T/-l in regrtest command line arguments.
- bpo-28683 [https://bugs.python.org/issue?@action=redirect&bpo=28683]: Fix the tests that bind() a unix socket and raise PermissionError on Android for a non-root user.
- bpo-26936 [https://bugs.python.org/issue?@action=redirect&bpo=26936]: Fix the test_socket failures on Android - getservbyname(), getservbyport() and getaddrinfo() are broken on some Android API levels.
- bpo-28666 [https://bugs.python.org/issue?@action=redirect&bpo=28666]: Now test.support.rmtree is able to remove unwritable or unreadable directories.
- bpo-23839 [https://bugs.python.org/issue?@action=redirect&bpo=23839]: Various caches now are cleared before running every test file.
- bpo-26944 [https://bugs.python.org/issue?@action=redirect&bpo=26944]: Fix test_posix for Android where “id -G” is entirely wrong or missing the effective gid.
- bpo-28409 [https://bugs.python.org/issue?@action=redirect&bpo=28409]: regrtest: fix the parser of

command line arguments.
- bpo-28217 [https://bugs.python.org/issue?@action=redirect&bpo=28217]: Adds _testconsole module to test console input.
- bpo-26939 [https://bugs.python.org/issue?@action=redirect&bpo=26939]: Add the support.setswitchinterval() function to fix test_functools hanging on the Android armv7 qemu emulator.

Build

- bpo-31354 [https://bugs.python.org/issue?@action=redirect&bpo=31354]: Allow `--with-lto` to be used on all builds, not just `make profile-opt`.
- bpo-31370 [https://bugs.python.org/issue?@action=redirect&bpo=31370]: Remove support for building –without-threads. This option is not really useful anymore in the 21st century. Removing lots of conditional paths allows us to simplify the code base, including in difficult to maintain low-level internal code.
- bpo-31341 [https://bugs.python.org/issue?@action=redirect&bpo=31341]: Per **PEP 11** [https://peps.python.org/pep-0011/], support for the IRIX operating system was removed.
- bpo-30854 [https://bugs.python.org/issue?@action=redirect&bpo=30854]: Fix compile error when compiling –without-threads. Patch by Masayuki Yamamoto.
- bpo-30687 [https://bugs.python.org/issue?@action=redirect&bpo=30687]: Locate msbuild.exe on Windows when building rather than vcvarsall.bat
- bpo-20210 [https://bugs.python.org/issue?@action=redirect&bpo=20210]: Support the *disabled* marker in Setup files. Extension modules listed after this marker are not built at all, neither by the Makefile nor by setup.py.
- bpo-29941 [https://bugs.python.org/issue?@action=redirect&bpo=29941]: Add `--with-assertions` configure flag to explicitly enable C `assert()` checks. Defaults to off. `--with-pydebug` implies `--with-assertions`.
- bpo-28787 [https://bugs.python.org/issue?

@action=redirect&bpo=28787]: Fix out-of-tree builds of Python when configured with `--with--dtrace`.

- bpo-29243 [https://bugs.python.org/issue?@action=redirect&bpo=29243]: Prevent unnecessary rebuilding of Python during `make test`, `make install` and some other make targets when configured with `--enable-optimizations`.
- bpo-23404 [https://bugs.python.org/issue?@action=redirect&bpo=23404]: Don't regenerate generated files based on file modification time anymore: the action is now explicit. Replace `make touch` with `make regen-all`.
- bpo-29643 [https://bugs.python.org/issue?@action=redirect&bpo=29643]: Fix `--enable-optimization` didn't work.
- bpo-27593 [https://bugs.python.org/issue?@action=redirect&bpo=27593]: sys.version and the platform module python_build(), python_branch(), and python_revision() functions now use git information rather than hg when building from a repo.
- bpo-29572 [https://bugs.python.org/issue?@action=redirect&bpo=29572]: Update Windows build and OS X installers to use OpenSSL 1.0.2k.
- bpo-27659 [https://bugs.python.org/issue?@action=redirect&bpo=27659]: Prohibit implicit C function declarations: use `-Werror=implicit-function-declaration` when possible (GCC and Clang, but it depends on the compiler version). Patch written by Chi Hsuan Yen.
- bpo-29384 [https://bugs.python.org/issue?@action=redirect&bpo=29384]: Remove old Be OS helper scripts.
- bpo-26851 [https://bugs.python.org/issue?@action=redirect&bpo=26851]: Set Android compilation and link flags.
- bpo-28768 [https://bugs.python.org/issue?@action=redirect&bpo=28768]: Fix implicit declaration of function _setmode. Patch by Masayuki Yamamoto
- bpo-29080 [https://bugs.python.org/issue?@action=redirect&bpo=29080]: Removes hard dependency on hg.exe from PCBuild/build.bat
- bpo-23903 [https://bugs.python.org/issue?@action=redirect&bpo=23903]: Added missed names to PC/

python3.def.

- bpo-28762 [https://bugs.python.org/issue?@action=redirect&bpo=28762]: lockf() is available on Android API level 24, but the F_LOCK macro is not defined in android-ndk-r13.
- bpo-28538 [https://bugs.python.org/issue?@action=redirect&bpo=28538]: Fix the compilation error that occurs because if_nameindex() is available on Android API level 24, but the if_nameindex structure is not defined.
- bpo-20211 [https://bugs.python.org/issue?@action=redirect&bpo=20211]: Do not add the directory for installing C header files and the directory for installing object code libraries to the cross compilation search paths. Original patch by Thomas Petazzoni.
- bpo-28849 [https://bugs.python.org/issue?@action=redirect&bpo=28849]: Do not define sys.implementation._multiarch on Android.
- bpo-10656 [https://bugs.python.org/issue?@action=redirect&bpo=10656]: Fix out-of-tree building on AIX. Patch by Tristan Carel and Michael Haubenwallner.
- bpo-26359 [https://bugs.python.org/issue?@action=redirect&bpo=26359]: Rename –with-optimiations to –enable-optimizations.
- bpo-28444 [https://bugs.python.org/issue?@action=redirect&bpo=28444]: Fix missing extensions modules when cross compiling.
- bpo-28208 [https://bugs.python.org/issue?@action=redirect&bpo=28208]: Update Windows build and OS X installers to use SQLite 3.14.2.
- bpo-28248 [https://bugs.python.org/issue?@action=redirect&bpo=28248]: Update Windows build and OS X installers to use OpenSSL 1.0.2j.
- bpo-21124 [https://bugs.python.org/issue?@action=redirect&bpo=21124]: Fix building the _struct module on Cygwin by passing `NULL` instead of `&PyType_Type` to PyVarObject_HEAD_INIT. Patch by Masayuki Yamamoto.
- bpo-13756 [https://bugs.python.org/issue?@action=redirect&bpo=13756]: Fix building extensions modules on Cygwin. Patch by Roumen Petrov, based on original patch by Jason Tishler.

- bpo-21085 [https://bugs.python.org/issue?@action=redirect&bpo=21085]: Add configure check for siginfo_t.si_band, which Cygwin does not provide. Patch by Masayuki Yamamoto with review and rebase by Erik Bray.
- bpo-28258 [https://bugs.python.org/issue?@action=redirect&bpo=28258]: Fixed build with Estonian locale (python-config and distclean targets in Makefile). Patch by Arfrever Frehtes Taifersar Arahesis.
- bpo-26661 [https://bugs.python.org/issue?@action=redirect&bpo=26661]: setup.py now detects system libffi with multiarch wrapper.
- bpo-27979 [https://bugs.python.org/issue?@action=redirect&bpo=27979]: A full copy of libffi is no longer bundled for use when building _ctypes on non-OSX UNIX platforms. An installed copy of libffi is now required when building _ctypes on such platforms.
- bpo-15819 [https://bugs.python.org/issue?@action=redirect&bpo=15819]: Remove redundant include search directory option for building outside the source tree.
- bpo-28676 [https://bugs.python.org/issue?@action=redirect&bpo=28676]: Prevent missing “getentropy” declaration warning on macOS. Patch by Gareth Rees.

Windows

- bpo-31392 [https://bugs.python.org/issue?@action=redirect&bpo=31392]: Update Windows build to use OpenSSL 1.1.0f
- bpo-30389 [https://bugs.python.org/issue?@action=redirect&bpo=30389]: Adds detection of Visual Studio 2017 to distutils on Windows.
- bpo-31358 [https://bugs.python.org/issue?@action=redirect&bpo=31358]: zlib is no longer bundled in the CPython source, instead it is downloaded on demand just like bz2, lzma, OpenSSL, Tcl/Tk, and SQLite.
- bpo-31340 [https://bugs.python.org/issue?@action=redirect&bpo=31340]: Change to building with MSVC v141 (included with Visual Studio 2017)
- bpo-30581 [https://bugs.python.org/issue?@action=redirect&bpo=30581]: os.cpu_count() now returns the

correct number of processors on Windows when the number of logical processors is greater than 64.

- bpo-30916 [https://bugs.python.org/issue?@action=redirect&bpo=30916]: Pre-build OpenSSL, Tcl and Tk and include the binaries in the build.
- bpo-30731 [https://bugs.python.org/issue?@action=redirect&bpo=30731]: Add a missing xmlns to python.manifest so that it matches the schema.
- bpo-30291 [https://bugs.python.org/issue?@action=redirect&bpo=30291]: Allow requiring 64-bit interpreters from py.exe using -64 suffix. Contributed by Steve (Gadget) Barnes.
- bpo-30362 [https://bugs.python.org/issue?@action=redirect&bpo=30362]: Adds list options (-0, -0p) to py.exe launcher. Contributed by Steve Barnes.
- bpo-23451 [https://bugs.python.org/issue?@action=redirect&bpo=23451]: Fix socket deprecation warnings in socketmodule.c. Patch by Segev Finer.
- bpo-30450 [https://bugs.python.org/issue?@action=redirect&bpo=30450]: The build process on Windows no longer depends on Subversion, instead pulling external code from GitHub via a Python script. If Python 3.6 is not found on the system (via `py -3.6`), NuGet is used to download a copy of 32-bit Python.
- bpo-29579 [https://bugs.python.org/issue?@action=redirect&bpo=29579]: Removes readme.txt from the installer.
- bpo-25778 [https://bugs.python.org/issue?@action=redirect&bpo=25778]: winreg does not truncate string correctly (Patch by Eryk Sun)
- bpo-28896 [https://bugs.python.org/issue?@action=redirect&bpo=28896]: Deprecate WindowsRegistryFinder and disable it by default
- bpo-28522 [https://bugs.python.org/issue?@action=redirect&bpo=28522]: Fixes mishandled buffer reallocation in getpathp.c
- bpo-28402 [https://bugs.python.org/issue?@action=redirect&bpo=28402]: Adds signed catalog files for stdlib on Windows.
- bpo-28333 [https://bugs.python.org/issue?

@action=redirect&bpo=28333]: Enables Unicode for ps1/ps2 and input() prompts. (Patch by Eryk Sun)
- bpo-28251 [https://bugs.python.org/issue?@action=redirect&bpo=28251]: Improvements to help manuals on Windows.
- bpo-28110 [https://bugs.python.org/issue?@action=redirect&bpo=28110]: launcher.msi has different product codes between 32-bit and 64-bit
- bpo-28161 [https://bugs.python.org/issue?@action=redirect&bpo=28161]: Opening CON for write access fails
- bpo-28162 [https://bugs.python.org/issue?@action=redirect&bpo=28162]: WindowsConsoleIO readall() fails if first line starts with Ctrl+Z
- bpo-28163 [https://bugs.python.org/issue?@action=redirect&bpo=28163]: WindowsConsoleIO fileno() passes wrong flags to _open_osfhandle
- bpo-28164 [https://bugs.python.org/issue?@action=redirect&bpo=28164]: _PyIO_get_console_type fails for various paths
- bpo-28137 [https://bugs.python.org/issue?@action=redirect&bpo=28137]: Renames Windows path file to ._pth
- bpo-28138 [https://bugs.python.org/issue?@action=redirect&bpo=28138]: Windows ._pth file should allow import site

IDLE

- bpo-31493 [https://bugs.python.org/issue?@action=redirect&bpo=31493]: IDLE code context – fix code update and font update timers. Canceling timers prevents a warning message when test_idle completes.
- bpo-31488 [https://bugs.python.org/issue?@action=redirect&bpo=31488]: IDLE - Update non-key options in former extension classes. When applying configdialog changes, call .reload for each feature class. Change ParenMatch so updated options affect existing instances attached to existing editor windows.
- bpo-31477 [https://bugs.python.org/issue?

@action=redirect&bpo=31477]: IDLE - Improve rstrip entry in doc. Strip trailing whitespace strips more than blank spaces. Multiline string literals are not skipped.
- bpo-31480 [https://bugs.python.org/issue?@action=redirect&bpo=31480]: IDLE - make tests pass with zzdummy extension disabled by default.
- bpo-31421 [https://bugs.python.org/issue?@action=redirect&bpo=31421]: Document how IDLE runs tkinter programs. IDLE calls tcl/tk update in the background in order to make live interaction and experimentation with tkinter applications much easier.
- bpo-31414 [https://bugs.python.org/issue?@action=redirect&bpo=31414]: IDLE – fix tk entry box tests by deleting first. Adding to an int entry is not the same as deleting and inserting because int(“”) will fail.
- bpo-31051 [https://bugs.python.org/issue?@action=redirect&bpo=31051]: Rearrange IDLE configdialog GenPage into Window, Editor, and Help sections.
- bpo-30617 [https://bugs.python.org/issue?@action=redirect&bpo=30617]: IDLE - Add docstrings and tests for outwin subclass of editor. Move some data and functions from the class to module level. Patch by Cheryl Sabella.
- bpo-31287 [https://bugs.python.org/issue?@action=redirect&bpo=31287]: IDLE - Do not modify tkinter.message in test_configdialog.
- bpo-27099 [https://bugs.python.org/issue?@action=redirect&bpo=27099]: Convert IDLE’s built-in “extensions” to regular features. About 10 IDLE features were implemented as supposedly optional extensions. Their different behavior could be confusing or worse for users and not good for maintenance. Hence the conversion. The main difference for users is that user configurable key bindings for builtin features are now handled uniformly. Now, editing a binding in a keyset only affects its value in the keyset. All bindings are defined together in the system-specific default keysets in config-extensions.def. All custom keysets are saved as a whole in config-extension.cfg. All take effect as soon as one clicks Apply or Ok. The affected events are “<<force-open-completions>>”, “<<expand-word>>”, “<<force-open-calltip>>”, “<<flash-paren>>”, “<<format-

paragraph> >”, “< <run-module> >”, “< <check-module> >”, and “< <zoom-height> >”. Any (global) customizations made before 3.6.3 will not affect their keyset-specific customization after 3.6.3. and vice versa. Initial patch by Charles Wohlganger.

- bpo-31206 [https://bugs.python.org/issue?@action=redirect&bpo=31206]: IDLE: Factor HighPage(Frame) class from ConfigDialog. Patch by Cheryl Sabella.
- bpo-31001 [https://bugs.python.org/issue?@action=redirect&bpo=31001]: Add tests for configdialog highlight tab. Patch by Cheryl Sabella.
- bpo-31205 [https://bugs.python.org/issue?@action=redirect&bpo=31205]: IDLE: Factor KeysPage(Frame) class from ConfigDialog. The slightly modified tests continue to pass. Patch by Cheryl Sabella.
- bpo-31130 [https://bugs.python.org/issue?@action=redirect&bpo=31130]: IDLE – stop leaks in test_configdialog. Initial patch by Victor Stinner.
- bpo-31002 [https://bugs.python.org/issue?@action=redirect&bpo=31002]: Add tests for configdialog keys tab. Patch by Cheryl Sabella.
- bpo-19903 [https://bugs.python.org/issue?@action=redirect&bpo=19903]: IDLE: Calltips use `inspect.signature` instead of `inspect.getfullargspec`. This improves calltips for builtins converted to use Argument Clinic. Patch by Louie Lu.
- bpo-31083 [https://bugs.python.org/issue?@action=redirect&bpo=31083]: IDLE - Add an outline of a TabPage class in configdialog. Update existing classes to match outline. Initial patch by Cheryl Sabella.
- bpo-31050 [https://bugs.python.org/issue?@action=redirect&bpo=31050]: Factor GenPage(Frame) class from ConfigDialog. The slightly modified tests continue to pass. Patch by Cheryl Sabella.
- bpo-31004 [https://bugs.python.org/issue?@action=redirect&bpo=31004]: IDLE - Factor FontPage(Frame) class from ConfigDialog. Slightly modified tests continue to pass. Fix General tests. Patch mostly by Cheryl Sabella.
- bpo-30781 [https://bugs.python.org/issue?@action=redirect&bpo=30781]: IDLE - Use ttk widgets in

ConfigDialog. Patches by Terry Jan Reedy and Cheryl Sabella.

- bpo-31060 [https://bugs.python.org/issue?@action=redirect&bpo=31060]: IDLE - Finish rearranging methods of ConfigDialog Grouping methods pertaining to each tab and the buttons will aid writing tests and improving the tabs and will enable splitting the groups into classes.
- bpo-30853 [https://bugs.python.org/issue?@action=redirect&bpo=30853]: IDLE – Factor a VarTrace class out of ConfigDialog. Instance tracers manages pairs consisting of a tk variable and a callback function. When tracing is turned on, setting the variable calls the function. Test coverage for the new class is 100%.
- bpo-31003 [https://bugs.python.org/issue?@action=redirect&bpo=31003]: IDLE: Add more tests for General tab.
- bpo-30993 [https://bugs.python.org/issue?@action=redirect&bpo=30993]: IDLE - Improve configdialog font page and tests. In configdialog: Document causal pathways in create_font_tab docstring. Simplify some attribute names. Move set_samples calls to var_changed_font (idea from Cheryl Sabella). Move related functions to positions after the create widgets function. In test_configdialog: Fix test_font_set so not order dependent. Fix renamed test_indent_scale so it tests the widget. Adjust tests for movement of set_samples call. Add tests for load functions. Put all font tests in one class and tab indent tests in another. Except for two lines, these tests completely cover the related functions.
- bpo-30981 [https://bugs.python.org/issue?@action=redirect&bpo=30981]: IDLE – Add more configdialog font page tests.
- bpo-28523 [https://bugs.python.org/issue?@action=redirect&bpo=28523]: IDLE: replace “colour” with “color” in configdialog.
- bpo-30917 [https://bugs.python.org/issue?@action=redirect&bpo=30917]: Add tests for idlelib.config.IdleConf. Increase coverage from 46% to 96%. Patch by Louie Lu.
- bpo-30934 [https://bugs.python.org/issue?@action=redirect&bpo=30934]: Document coverage details for idlelib tests. Add section to idlelib/idle-test/README.txt.

Include check that branches are taken both ways. Exclude IDLE-specific code that does not run during unit tests.

- bpo-30913 [https://bugs.python.org/issue?@action=redirect&bpo=30913]: IDLE: Document ConfigDialog tk Vars, methods, and widgets in docstrings This will facilitate improving the dialog and splitting up the class. Original patch by Cheryl Sabella.
- bpo-30899 [https://bugs.python.org/issue?@action=redirect&bpo=30899]: IDLE: Add tests for ConfigParser subclasses in config. Patch by Louie Lu.
- bpo-30881 [https://bugs.python.org/issue?@action=redirect&bpo=30881]: IDLE: Add docstrings to browser.py. Patch by Cheryl Sabella.
- bpo-30851 [https://bugs.python.org/issue?@action=redirect&bpo=30851]: IDLE: Remove unused variables in configdialog. One is a duplicate, one is set but cannot be altered by users. Patch by Cheryl Sabella.
- bpo-30870 [https://bugs.python.org/issue?@action=redirect&bpo=30870]: IDLE: In Settings dialog, select font with Up, Down keys as well as mouse. Initial patch by Louie Lu.
- bpo-8231 [https://bugs.python.org/issue?@action=redirect&bpo=8231]: IDLE: call config.IdleConf.GetUserCfgDir only once.
- bpo-30779 [https://bugs.python.org/issue?@action=redirect&bpo=30779]: IDLE: Factor ConfigChanges class from configdialog, put in config; test. * In config, put dump test code in a function; run it and unittest in “if _name_ = = “_main_”. * Add class config.ConfigChanges based on changes_class_v4.py on bpo issue. * Add class test_config.ChangesTest, partly using configdialog_tests_v1.py. * Revise configdialog to use ConfigChanges; see tracker msg297804. * Revise test_configdialog to match configdialog changes. * Remove configdialog functions unused or moved to ConfigChanges. Cheryl Sabella contributed parts of the patch.
- bpo-30777 [https://bugs.python.org/issue?@action=redirect&bpo=30777]: IDLE: configdialog - Add docstrings and fix comments. Patch by Cheryl Sabella.
- bpo-30495 [https://bugs.python.org/issue?@action=redirect&bpo=30495]: IDLE: Improve textview with

docstrings, PEP8 names, and more tests. Patch by Cheryl Sabella.

- bpo-30723 [https://bugs.python.org/issue?@action=redirect&bpo=30723]: IDLE: Make several improvements to parenmatch. Add “parens” style to highlight both opener and closer. Make “default” style, which is not default, a synonym for “opener”. Make time-delay work the same with all styles. Add help for config dialog extensions tab, including help for parenmatch. Add new tests. Original patch by Charles Wohlganger.
- bpo-30674 [https://bugs.python.org/issue?@action=redirect&bpo=30674]: IDLE: add docstrings to grep module. Patch by Cheryl Sabella
- bpo-21519 [https://bugs.python.org/issue?@action=redirect&bpo=21519]: IDLE’s basic custom key entry dialog now detects duplicates properly. Original patch by Saimadhav Heblikar.
- bpo-29910 [https://bugs.python.org/issue?@action=redirect&bpo=29910]: IDLE no longer deletes a character after commenting out a region by a key shortcut. Add `return 'break'` for this and other potential conflicts between IDLE and default key bindings.
- bpo-30728 [https://bugs.python.org/issue?@action=redirect&bpo=30728]: Review and change idlelib.configdialog names. Lowercase method and attribute names. Replace “colour” with “color”, expand overly cryptic names, delete unneeded underscores. Replace `import *` with specific imports. Patches by Cheryl Sabella.
- bpo-6739 [https://bugs.python.org/issue?@action=redirect&bpo=6739]: IDLE: Verify user-entered key sequences by trying to bind them with tk. Add tests for all 3 validation functions. Original patch by G Polo. Tests added by Cheryl Sabella.
- bpo-15786 [https://bugs.python.org/issue?@action=redirect&bpo=15786]: Fix several problems with IDLE’s autocompletion box. The following should now work: clicking on selection box items; using the scrollbar; selecting an item by hitting Return. Hangs on MacOSX should no longer happen. Patch by Louie Lu.
- bpo-25514 [https://bugs.python.org/issue?@action=redirect&bpo=25514]: Add doc subsubsection about

IDLE failure to start. Popup no-connection message directs users to this section.

- bpo-30642 [https://bugs.python.org/issue?@action=redirect&bpo=30642]: Fix reference leaks in IDLE tests. Patches by Louie Lu and Terry Jan Reedy.
- bpo-30495 [https://bugs.python.org/issue?@action=redirect&bpo=30495]: Add docstrings for textview.py and use PEP8 names. Patches by Cheryl Sabella and Terry Jan Reedy.
- bpo-30290 [https://bugs.python.org/issue?@action=redirect&bpo=30290]: Help-about: use pep8 names and add tests. Increase coverage to 100%. Patches by Louie Lu, Cheryl Sabella, and Terry Jan Reedy.
- bpo-30303 [https://bugs.python.org/issue?@action=redirect&bpo=30303]: Add _utest option to textview; add new tests. Increase coverage to 100%. Patches by Louie Lu and Terry Jan Reedy.
- bpo-29071 [https://bugs.python.org/issue?@action=redirect&bpo=29071]: IDLE colors f-string prefixes (but not invalid ur prefixes).
- bpo-28572 [https://bugs.python.org/issue?@action=redirect&bpo=28572]: Add 10% to coverage of IDLE's test_configdialog. Update and augment description of the configuration system.

Tools/Demos

- bpo-30983 [https://bugs.python.org/issue?@action=redirect&bpo=30983]: gdb integration commands (py-bt, etc.) work on optimized shared builds now, too. **PEP 523** [https://peps.python.org/pep-0523/] introduced _PyEval_EvalFrameDefault which inlines PyEval_EvalFrameEx on non-debug shared builds. This broke the ability to use py-bt, py-up, and a few other Python-specific gdb integrations. The problem is fixed by only looking for _PyEval_EvalFrameDefault frames in python-gdb.py. Original patch by Bruno «Polaco» Penteado.
- bpo-29748 [https://bugs.python.org/issue?@action=redirect&bpo=29748]: Added the slice index converter in Argument Clinic.

- bpo-24037 [https://bugs.python.org/issue?@action=redirect&bpo=24037]: Argument Clinic now uses the converter `bool(accept={int})` rather than `int` for semantical booleans. This avoids repeating the default value for Python and C and will help in converting to `bool` in future.
- bpo-29367 [https://bugs.python.org/issue?@action=redirect&bpo=29367]: python-gdb.py now supports also `method-wrapper` (`wrapperobject`) objects.
- bpo-28023 [https://bugs.python.org/issue?@action=redirect&bpo=28023]: Fix python-gdb.py didn't support new dict implementation.
- bpo-15369 [https://bugs.python.org/issue?@action=redirect&bpo=15369]: The pybench and pystone microbenchmark have been removed from Tools. Please use the new Python benchmark suite https://github.com/python/pyperformance which is more reliable and includes a portable version of pybench working on Python 2 and Python 3.
- bpo-28102 [https://bugs.python.org/issue?@action=redirect&bpo=28102]: The zipfile module CLI now prints usage to stderr. Patch by Stephen J. Turnbull.

C API

- bpo-31338 [https://bugs.python.org/issue?@action=redirect&bpo=31338]: Added the `Py_UNREACHABLE()` macro for code paths which are never expected to be reached. This and a few other useful macros are now documented in the C API manual.
- bpo-30832 [https://bugs.python.org/issue?@action=redirect&bpo=30832]: Remove own implementation for thread-local storage. CPython has provided the own implementation for thread-local storage (TLS) on Python/thread.c, it's used in the case which a platform has not supplied native TLS. However, currently all supported platforms (Windows and pthreads) have provided native TLS and defined the Py_HAVE_NATIVE_TLS macro with unconditional in any case.
- bpo-30708 [https://bugs.python.org/issue?@action=redirect&bpo=30708]: PyUnicode_AsWideCharString()

now raises a ValueError if the second argument is NULL and the wchar_t* string contains null characters.

- bpo-16500 [https://bugs.python.org/issue?@action=redirect&bpo=16500]: Deprecate PyOS_AfterFork() and add PyOS_BeforeFork(), PyOS_AfterFork_Parent() and PyOS_AfterFork_Child().
- bpo-6532 [https://bugs.python.org/issue?@action=redirect&bpo=6532]: The type of results of PyThread_start_new_thread() and PyThread_get_thread_ident(), and the id parameter of PyThreadState_SetAsyncExc() changed from «long» to «unsigned long».
- bpo-27867 [https://bugs.python.org/issue?@action=redirect&bpo=27867]: Function PySlice_GetIndicesEx() is deprecated and replaced with a macro if Py_LIMITED_API is not set or set to the value between 0x03050400 and 0x03060000 (not including) or 0x03060100 or higher. Added functions PySlice_Unpack() and PySlice_AdjustIndices().
- bpo-29083 [https://bugs.python.org/issue?@action=redirect&bpo=29083]: Fixed the declaration of some public API functions. PyArg_VaParse() and PyArg_VaParseTupleAndKeywords() were not available in limited API. PyArg_ValidateKeywordArguments(), PyArg_UnpackTuple() and Py_BuildValue() were not available in limited API of version < 3.3 when PY_SSIZE_T_CLEAN is defined.
- bpo-28769 [https://bugs.python.org/issue?@action=redirect&bpo=28769]: The result of PyUnicode_AsUTF8AndSize() and PyUnicode_AsUTF8() is now of type `const char *` rather of `char *`.
- bpo-29058 [https://bugs.python.org/issue?@action=redirect&bpo=29058]: All stable API extensions added after Python 3.2 are now available only when Py_LIMITED_API is set to the PY_VERSION_HEX value of the minimum Python version supporting this API.
- bpo-28822 [https://bugs.python.org/issue?@action=redirect&bpo=28822]: The index parameters *start* and *end* of PyUnicode_FindChar() are now adjusted to behave like `str[start:end]`.
- bpo-28808 [https://bugs.python.org/issue?@action=redirect&bpo=28808]:

PyUnicode_CompareWithASCIIString() now never raises exceptions.
- bpo-28761 [https://bugs.python.org/issue?@action=redirect&bpo=28761]: The fields name and doc of structures PyMemberDef, PyGetSetDef, PyStructSequence_Field, PyStructSequence_Desc, and wrapperbase are now of type `const char *` rather of `char *`.
- bpo-28748 [https://bugs.python.org/issue?@action=redirect&bpo=28748]: Private variable _Py_PackageContext is now of type `const char *` rather of `char *`.
- bpo-19569 [https://bugs.python.org/issue?@action=redirect&bpo=19569]: Compiler warnings are now emitted if use most of deprecated functions.
- bpo-28426 [https://bugs.python.org/issue?@action=redirect&bpo=28426]: Deprecated undocumented functions PyUnicode_AsEncodedObject(), PyUnicode_AsDecodedObject(), PyUnicode_AsDecodedUnicode() and PyUnicode_AsEncodedUnicode().

Python 3.6.6 final

Release date: 2018-06-27

There were no new changes in version 3.6.6.

Python 3.6.6 release candidate 1

Release date: 2018-06-11

Core and Builtins

- bpo-33786 [https://bugs.python.org/issue?@action=redirect&bpo=33786]: Fix asynchronous generators to handle GeneratorExit in athrow() correctly
- bpo-30654 [https://bugs.python.org/issue?@action=redirect&bpo=30654]: Fixed reset of the SIGINT handler

to SIG_DFL on interpreter shutdown even when there was a custom handler set previously. Patch by Philipp Kerling.

- bpo-33622 [https://bugs.python.org/issue?@action=redirect&bpo=33622]: Fixed a leak when the garbage collector fails to add an object with the `__del__` method or referenced by it into the `gc.garbage` list. `PyGC_Collect()` can now be called when an exception is set and preserves it.
- bpo-31849 [https://bugs.python.org/issue?@action=redirect&bpo=31849]: Fix signed/unsigned comparison warning in pyhash.c.
- bpo-33391 [https://bugs.python.org/issue?@action=redirect&bpo=33391]: Fix a leak in set_symmetric_difference().
- bpo-28055 [https://bugs.python.org/issue?@action=redirect&bpo=28055]: Fix unaligned accesses in siphash24(). Patch by Rolf Eike Beer.
- bpo-33231 [https://bugs.python.org/issue?@action=redirect&bpo=33231]: Fix potential memory leak in `normalizestring()`.
- bpo-29922 [https://bugs.python.org/issue?@action=redirect&bpo=29922]: Improved error messages in “async with” when `__aenter__()` or `__aexit__()` return non-awaitable object.
- bpo-33199 [https://bugs.python.org/issue?@action=redirect&bpo=33199]: Fix `ma_version_tag` in dict implementation is uninitialized when copying from key-sharing dict.
- bpo-33041 [https://bugs.python.org/issue?@action=redirect&bpo=33041]: Fixed jumping when the function contains an `async for` loop.
- bpo-32282 [https://bugs.python.org/issue?@action=redirect&bpo=32282]: Fix an unnecessary ifdef in the include of VersionHelpers.h in socketmodule on Windows.
- bpo-21983 [https://bugs.python.org/issue?@action=redirect&bpo=21983]: Fix a crash in `ctypes.cast()` in case the type argument is a ctypes structured data type. Patch by Eryk Sun and Oren Milman.

Library

- bpo-30167 [https://bugs.python.org/issue?@action=redirect&bpo=30167]: Prevent site.main() exception if PYTHONSTARTUP is set. Patch by Steve Weber.
- bpo-33812 [https://bugs.python.org/issue?@action=redirect&bpo=33812]: Datetime instance d with non-None tzinfo, but with d.tzinfo.utcoffset(d) returning None is now treated as naive by the astimezone() method.
- bpo-30805 [https://bugs.python.org/issue?@action=redirect&bpo=30805]: Avoid race condition with debug logging
- bpo-33767 [https://bugs.python.org/issue?@action=redirect&bpo=33767]: The concatenation (+) and repetition (*) sequence operations now raise `TypeError` instead of `SystemError` when performed on `mmap.mmap` objects. Patch by Zackery Spytz.
- bpo-32684 [https://bugs.python.org/issue?@action=redirect&bpo=32684]: Fix gather to propagate cancellation of itself even with return_exceptions.
- bpo-33674 [https://bugs.python.org/issue?@action=redirect&bpo=33674]: Fix a race condition in SSLProtocol.connection_made() of asyncio.sslproto: start immediately the handshake instead of using call_soon(). Previously, data_received() could be called before the handshake started, causing the handshake to hang or fail.
- bpo-31647 [https://bugs.python.org/issue?@action=redirect&bpo=31647]: Fixed bug where calling write_eof() on a _SelectorSocketTransport after it's already closed raises AttributeError.
- bpo-33672 [https://bugs.python.org/issue?@action=redirect&bpo=33672]: Fix Task.__repr__ crash with Cython's bogus coroutines
- bpo-33469 [https://bugs.python.org/issue?@action=redirect&bpo=33469]: Fix RuntimeError after closing loop that used run_in_executor
- bpo-11874 [https://bugs.python.org/issue?@action=redirect&bpo=11874]: Use a better regex when breaking usage into wrappable parts. Avoids bogus assertion errors from custom metavar strings.
- bpo-30877 [https://bugs.python.org/issue?@action=redirect&bpo=30877]: Fixed a bug in the Python

implementation of the JSON decoder that prevented the cache of parsed strings from clearing after finishing the decoding. Based on patch by c-fos.

- bpo-33548 [https://bugs.python.org/issue?@action=redirect&bpo=33548]: tempfile._candidate_tempdir_list should consider common TEMP locations
- bpo-33542 [https://bugs.python.org/issue?@action=redirect&bpo=33542]: Prevent `uuid.get_node` from using a DUID instead of a MAC on Windows. Patch by Zvi Effron
- bpo-26819 [https://bugs.python.org/issue?@action=redirect&bpo=26819]: Fix race condition with `ReadTransport.resume_reading` in Windows proactor event loop.
- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Minor fixes in typing module: add annotations to `NamedTuple.__new__`, pass `*args` and `**kwds` in `Generic.__new__`. Original PRs by Paulius Šarka and Chad Dombrova.
- bpo-20087 [https://bugs.python.org/issue?@action=redirect&bpo=20087]: Updated alias mapping with glibc 2.27 supported locales.
- bpo-33422 [https://bugs.python.org/issue?@action=redirect&bpo=33422]: Fix trailing quotation marks getting deleted when looking up byte/string literals on pydoc. Patch by Andrés Delfino.
- bpo-33197 [https://bugs.python.org/issue?@action=redirect&bpo=33197]: Update error message when constructing invalid inspect.Parameters Patch by Donghee Na.
- bpo-33383 [https://bugs.python.org/issue?@action=redirect&bpo=33383]: Fixed crash in the get() method of the **`dbm.ndbm`** database object when it is called with a single argument.
- bpo-33329 [https://bugs.python.org/issue?@action=redirect&bpo=33329]: Fix multiprocessing regression on newer glibcs
- bpo-991266 [https://bugs.python.org/issue?@action=redirect&bpo=991266]: Fix quoting of the `Comment` attribute of **`http.cookies.SimpleCookie`**.
- bpo-33131 [https://bugs.python.org/issue?

@action=redirect&bpo=33131]: Upgrade bundled version of pip to 10.0.1.

- bpo-33308 [https://bugs.python.org/issue?@action=redirect&bpo=33308]: Fixed a crash in the **parser** module when converting an ST object to a tree of tuples or lists with `line_info=False` and `col_info=True`.
- bpo-33263 [https://bugs.python.org/issue?@action=redirect&bpo=33263]: Fix FD leak in `_SelectorSocketTransport` Patch by Vlad Starostin.
- bpo-33256 [https://bugs.python.org/issue?@action=redirect&bpo=33256]: Fix display of `<module>` call in the html produced by `cgitb.html()`. Patch by Stéphane Blondon.
- bpo-33203 [https://bugs.python.org/issue?@action=redirect&bpo=33203]: `random.Random.choice()` now raises `IndexError` for empty sequences consistently even when called from subclasses without a `getrandbits()` implementation.
- bpo-33224 [https://bugs.python.org/issue?@action=redirect&bpo=33224]: Update difflib.mdiff() for **PEP 479** [https://peps.python.org/pep-0479/]. Convert an uncaught StopIteration in a generator into a return-statement.
- bpo-33209 [https://bugs.python.org/issue?@action=redirect&bpo=33209]: End framing at the end of C implementation of **`pickle.Pickler.dump()`**.
- bpo-32861 [https://bugs.python.org/issue?@action=redirect&bpo=32861]: The urllib.robotparser's `__str__` representation now includes wildcard entries and the «Crawl-delay» and «Request-rate» fields. Patch by Michael Lazar.
- bpo-33096 [https://bugs.python.org/issue?@action=redirect&bpo=33096]: Allow ttk.Treeview.insert to insert iid that has a false boolean value. Note iid=0 and iid=False would be same. Patch by Garvit Khatri.
- bpo-33127 [https://bugs.python.org/issue?@action=redirect&bpo=33127]: The ssl module now compiles with LibreSSL 2.7.1.
- bpo-33021 [https://bugs.python.org/issue?@action=redirect&bpo=33021]: Release the GIL during fstat() calls, avoiding hang of all threads when calling mmap.mmap(), os.urandom(), and random.seed(). Patch by

Nir Soffer.

- bpo-27683 [https://bugs.python.org/issue?@action=redirect&bpo=27683]: Fix a regression in **`ipaddress`** that result of **`hosts()`** is empty when the network is constructed by a tuple containing an integer mask and only 1 bit left for addresses.
- bpo-32844 [https://bugs.python.org/issue?@action=redirect&bpo=32844]: Fix wrong redirection of a low descriptor (0 or 1) to stderr in subprocess if another low descriptor is closed.
- bpo-31908 [https://bugs.python.org/issue?@action=redirect&bpo=31908]: Fix output of cover files for `trace` module command-line tool. Previously emitted cover files only when `--missing` option was used. Patch by Michael Selik.
- bpo-31457 [https://bugs.python.org/issue?@action=redirect&bpo=31457]: If nested log adapters are used, the inner `process()` methods are no longer omitted.
- bpo-16865 [https://bugs.python.org/issue?@action=redirect&bpo=16865]: Support arrays >=2GiB in **`ctypes`**. Patch by Segev Finer.
- bpo-31238 [https://bugs.python.org/issue?@action=redirect&bpo=31238]: pydoc: the stop() method of the private ServerThread class now waits until DocServer.serve_until_quit() completes and then explicitly sets its docserver attribute to None to break a reference cycle.

Documentation

- bpo-33503 [https://bugs.python.org/issue?@action=redirect&bpo=33503]: Fix broken pypi link
- bpo-33421 [https://bugs.python.org/issue?@action=redirect&bpo=33421]: Add missing documentation for `typing.AsyncContextManager`.
- bpo-33378 [https://bugs.python.org/issue?@action=redirect&bpo=33378]: Add Korean language switcher for https://docs.python.org/3/
- bpo-33276 [https://bugs.python.org/issue?@action=redirect&bpo=33276]: Clarify that the `__path__` attribute on modules cannot be just any value.

- bpo-33201 [https://bugs.python.org/issue?@action=redirect&bpo=33201]: Modernize documentation for writing C extension types.
- bpo-33195 [https://bugs.python.org/issue?@action=redirect&bpo=33195]: Deprecate `Py_UNICODE` usage in `c-api/arg` document. `Py_UNICODE` related APIs are deprecated since Python 3.3, but it is missed in the document.
- bpo-33126 [https://bugs.python.org/issue?@action=redirect&bpo=33126]: Document PyBuffer_ToContiguous().
- bpo-27212 [https://bugs.python.org/issue?@action=redirect&bpo=27212]: Modify documentation for the **`islice()`** recipe to consume initial values up to the start index.
- bpo-28247 [https://bugs.python.org/issue?@action=redirect&bpo=28247]: Update **`zipapp`** documentation to describe how to make standalone applications.
- bpo-18802 [https://bugs.python.org/issue?@action=redirect&bpo=18802]: Documentation changes for ipaddress. Patch by Jon Foster and Berker Peksag.
- bpo-27428 [https://bugs.python.org/issue?@action=redirect&bpo=27428]: Update documentation to clarify that `WindowsRegistryFinder` implements `MetaPathFinder`. (Patch by Himanshu Lakhara)
- bpo-8243 [https://bugs.python.org/issue?@action=redirect&bpo=8243]: Add a note about curses.addch and curses.addstr exception behavior when writing outside a window, or pad.
- bpo-31432 [https://bugs.python.org/issue?@action=redirect&bpo=31432]: Clarify meaning of CERT_NONE, CERT_OPTIONAL, and CERT_REQUIRED flags for ssl.SSLContext.verify_mode.

Tests

- bpo-33655 [https://bugs.python.org/issue?@action=redirect&bpo=33655]: Ignore test_posix_fallocate failures on BSD platforms that might be due to running on ZFS.
- bpo-19417 [https://bugs.python.org/issue?@action=redirect&bpo=19417]: Add test_bdb.py.

Build

- bpo-5755 [https://bugs.python.org/issue?@action=redirect&bpo=5755]: Move `-Wstrict-prototypes` option to `CFLAGS_NODIST` from `OPT`. This option emitted annoying warnings when building extension modules written in C++.
- bpo-33614 [https://bugs.python.org/issue?@action=redirect&bpo=33614]: Ensures module definition files for the stable ABI on Windows are correctly regenerated.
- bpo-33522 [https://bugs.python.org/issue?@action=redirect&bpo=33522]: Enable CI builds on Visual Studio Team Services at https://python.visualstudio.com/cpython
- bpo-33012 [https://bugs.python.org/issue?@action=redirect&bpo=33012]: Add `-Wno-cast-function-type` for gcc 8 for silencing warnings about function casts like casting to PyCFunction in method definition lists.
- bpo-33394 [https://bugs.python.org/issue?@action=redirect&bpo=33394]: Enable the verbose build for extension modules, when GNU make is passed macros on the command line.

Windows

- bpo-33184 [https://bugs.python.org/issue?@action=redirect&bpo=33184]: Update Windows installer to OpenSSL 1.0.2o.

macOS

- bpo-33184 [https://bugs.python.org/issue?@action=redirect&bpo=33184]: Update macOS installer build to use OpenSSL 1.0.2o.

IDLE

- bpo-33656 [https://bugs.python.org/issue?@action=redirect&bpo=33656]: On Windows, add API call saying that tk scales for DPI. On Windows 8.1+ or 10, with DPI compatibility properties of the Python binary unchanged, and a monitor resolution greater than 96 DPI, this should make text and lines sharper. It should otherwise have no effect.

- bpo-33768 [https://bugs.python.org/issue?@action=redirect&bpo=33768]: Clicking on a context line moves that line to the top of the editor window.
- bpo-33763 [https://bugs.python.org/issue?@action=redirect&bpo=33763]: IDLE: Use read-only text widget for code context instead of label widget.
- bpo-33664 [https://bugs.python.org/issue?@action=redirect&bpo=33664]: Scroll IDLE editor text by lines. Previously, the mouse wheel and scrollbar slider moved text by a fixed number of pixels, resulting in partial lines at the top of the editor box. The change also applies to the shell and grep output windows, but not to read-only text views.
- bpo-33679 [https://bugs.python.org/issue?@action=redirect&bpo=33679]: Enable theme-specific color configuration for Code Context. Use the Highlights tab to see the setting for built-in themes or add settings to custom themes.
- bpo-33642 [https://bugs.python.org/issue?@action=redirect&bpo=33642]: Display up to maxlines non-blank lines for Code Context. If there is no current context, show a single blank line.
- bpo-33628 [https://bugs.python.org/issue?@action=redirect&bpo=33628]: IDLE: Cleanup codecontext.py and its test.
- bpo-33564 [https://bugs.python.org/issue?@action=redirect&bpo=33564]: IDLE's code context now recognizes async as a block opener.
- bpo-29706 [https://bugs.python.org/issue?@action=redirect&bpo=29706]: IDLE now colors async and await as keywords in 3.6. They become full keywords in 3.7.
- bpo-21474 [https://bugs.python.org/issue?@action=redirect&bpo=21474]: Update word/identifier definition from ascii to unicode. In text and entry boxes, this affects selection by double-click, movement left/right by control-left/right, and deletion left/right by control-BACKSPACE/DEL.
- bpo-33204 [https://bugs.python.org/issue?@action=redirect&bpo=33204]: IDLE: consistently color invalid

string prefixes. A “u” string prefix cannot be paired with either “r” or “f”. Consistently color as much of the prefix, starting at the right, as is valid. Revise and extend colorizer test.
- bpo-32831 [https://bugs.python.org/issue?@action=redirect&bpo=32831]: Add docstrings and tests for codecontext.

Tools/Demos

- bpo-33189 [https://bugs.python.org/issue?@action=redirect&bpo=33189]: **pygettext.py** now recognizes only literal strings as docstrings and translatable strings, and rejects bytes literals and f-string expressions.
- bpo-31920 [https://bugs.python.org/issue?@action=redirect&bpo=31920]: Fixed handling directories as arguments in the `pygettext` script. Based on patch by Oleg Krasnikov.
- bpo-29673 [https://bugs.python.org/issue?@action=redirect&bpo=29673]: Fix pystackv and pystack gdbinit macros.
- bpo-32885 [https://bugs.python.org/issue?@action=redirect&bpo=32885]: Add an `-n` flag for `Tools/scripts/pathfix.py` to disable automatic backup creation (files with `~` suffix).
- bpo-31583 [https://bugs.python.org/issue?@action=redirect&bpo=31583]: Fix 2to3 for using with –add-suffix option but without –output-dir option for relative path to files in current directory.

C API

- bpo-32374 [https://bugs.python.org/issue?@action=redirect&bpo=32374]: Document that m_traverse for multi-phase initialized modules can be called with m_state=NULL, and add a sanity check

Python 3.6.5 final

Release date: 2018-03-28

Tests

- bpo-32872 [https://bugs.python.org/issue?@action=redirect&bpo=32872]: Avoid regrtest compatibility issue with namespace packages.

Build

- bpo-33163 [https://bugs.python.org/issue?@action=redirect&bpo=33163]: Upgrade pip to 9.0.3 and setuptools to v39.0.1.

Python 3.6.5 release candidate 1

Release date: 2018-03-13

Security

- bpo-33001 [https://bugs.python.org/issue?@action=redirect&bpo=33001]: Minimal fix to prevent buffer overrun in os.symlink on Windows
- bpo-32981 [https://bugs.python.org/issue?@action=redirect&bpo=32981]: Regexes in difflib and poplib were vulnerable to catastrophic backtracking. These regexes formed potential DOS vectors (REDOS). They have been refactored. This resolves **CVE 2018-1060** [https://www.cve.org/CVERecord?id=CVE-2018-1060] and **CVE 2018-1061** [https://www.cve.org/CVERecord?id=CVE-2018-1061]. Patch by Jamie Davis.

Core and Builtins

- bpo-33026 [https://bugs.python.org/issue?@action=redirect&bpo=33026]: Fixed jumping out of «with» block by setting f_lineno.
- bpo-17288 [https://bugs.python.org/issue?@action=redirect&bpo=17288]: Prevent jumps from “return” and “exception” trace events.

- bpo-32889 [https://bugs.python.org/issue?@action=redirect&bpo=32889]: Update Valgrind suppression list to account for the rename of `Py_ADDRESS_IN_RANG` to `address_in_range`.
- bpo-32650 [https://bugs.python.org/issue?@action=redirect&bpo=32650]: Pdb and other debuggers dependent on bdb.py will correctly step over (next command) native coroutines. Patch by Pablo Galindo.
- bpo-32685 [https://bugs.python.org/issue?@action=redirect&bpo=32685]: Improve suggestion when the Python 2 form of print statement is either present on the same line as the header of a compound statement or else terminated by a semi-colon instead of a newline. Patch by Nitish Chandra.
- bpo-32583 [https://bugs.python.org/issue?@action=redirect&bpo=32583]: Fix possible crashing in builtin Unicode decoders caused by write out-of-bound errors when using customized decode error handlers.
- bpo-26163 [https://bugs.python.org/issue?@action=redirect&bpo=26163]: Improved frozenset() hash to create more distinct hash values when faced with datasets containing many similar values.
- bpo-27169 [https://bugs.python.org/issue?@action=redirect&bpo=27169]: The `__debug__` constant is now optimized out at compile time. This fixes also bpo-22091 [https://bugs.python.org/issue?@action=redirect&bpo=22091].
- bpo-32329 [https://bugs.python.org/issue?@action=redirect&bpo=32329]: `sys.flags.hash_randomization` is now properly set to 0 when hash randomization is turned off by `PYTHONHASHSEED=0`.
- bpo-30416 [https://bugs.python.org/issue?@action=redirect&bpo=30416]: The optimizer is now protected from spending much time doing complex calculations and consuming much memory for creating large constants in constant folding.
- bpo-18533 [https://bugs.python.org/issue?@action=redirect&bpo=18533]: `repr()` on a dict containing its own `values()` or `items()` no longer raises `RecursionError`; OrderedDict similarly. Instead, use `...`,

as for other recursive structures. Patch by Ben North.

- bpo-32028 [https://bugs.python.org/issue?@action=redirect&bpo=32028]: Leading whitespace is now correctly ignored when generating suggestions for converting Py2 print statements to Py3 builtin print function calls. Patch by Sanyam Khurana.
- bpo-32137 [https://bugs.python.org/issue?@action=redirect&bpo=32137]: The repr of deeply nested dict now raises a RecursionError instead of crashing due to a stack overflow.

Library

- bpo-33064 [https://bugs.python.org/issue?@action=redirect&bpo=33064]: lib2to3 now properly supports trailing commas after `*args` and `**kwargs` in function signatures.
- bpo-31804 [https://bugs.python.org/issue?@action=redirect&bpo=31804]: Avoid failing in multiprocessing.Process if the standard streams are closed or None at exit.
- bpo-33037 [https://bugs.python.org/issue?@action=redirect&bpo=33037]: Skip sending/receiving data after SSL transport closing.
- bpo-30353 [https://bugs.python.org/issue?@action=redirect&bpo=30353]: Fix ctypes pass-by-value for structs on 64-bit Cygwin/MinGW.
- bpo-33009 [https://bugs.python.org/issue?@action=redirect&bpo=33009]: Fix inspect.signature() for single-parameter partialmethods.
- bpo-32969 [https://bugs.python.org/issue?@action=redirect&bpo=32969]: Expose several missing constants in zlib and fix corresponding documentation.
- bpo-32713 [https://bugs.python.org/issue?@action=redirect&bpo=32713]: Fixed tarfile.itn handling of out-of-bounds float values. Patch by Joffrey Fuhrer.
- bpo-30622 [https://bugs.python.org/issue?@action=redirect&bpo=30622]: The ssl module now detects missing NPN support in LibreSSL.
- bpo-32922 [https://bugs.python.org/issue?

@action=redirect&bpo=32922]: dbm.open() now encodes filename with the filesystem encoding rather than default encoding.

- bpo-32859 [https://bugs.python.org/issue?@action=redirect&bpo=32859]: In `os.dup2`, don't check every call whether the `dup3` syscall exists or not.
- bpo-21060 [https://bugs.python.org/issue?@action=redirect&bpo=21060]: Rewrite confusing message from setup.py upload from «No dist file created in earlier command» to the more helpful «Must create and upload files in one command».
- bpo-32857 [https://bugs.python.org/issue?@action=redirect&bpo=32857]: In **`tkinter`**, `after_cancel(None)` now raises a **`ValueError`** instead of canceling the first scheduled function. Patch by Cheryl Sabella.
- bpo-32852 [https://bugs.python.org/issue?@action=redirect&bpo=32852]: Make sure sys.argv remains as a list when running trace.
- bpo-32841 [https://bugs.python.org/issue?@action=redirect&bpo=32841]: Fixed `asyncio.Condition` issue which silently ignored cancellation after notifying and cancelling a conditional lock. Patch by Bar Harel.
- bpo-31787 [https://bugs.python.org/issue?@action=redirect&bpo=31787]: Fixed refleaks of `__init__()` methods in various modules. (Contributed by Oren Milman)
- bpo-30157 [https://bugs.python.org/issue?@action=redirect&bpo=30157]: Fixed guessing quote and delimiter in csv.Sniffer.sniff() when only the last field is quoted. Patch by Jake Davis.
- bpo-32394 [https://bugs.python.org/issue?@action=redirect&bpo=32394]: socket: Remove TCP_FASTOPEN, TCP_KEEPCNT flags on older version Windows during runtime.
- bpo-32777 [https://bugs.python.org/issue?@action=redirect&bpo=32777]: Fix a rare but potential pre-exec child process deadlock in subprocess on POSIX systems when marking file descriptors inheritable on exec in the child process. This bug appears to have been introduced in 3.4.
- bpo-32647 [https://bugs.python.org/issue?@action=redirect&bpo=32647]: The ctypes module used to

depend on indirect linking for dlopen. The shared extension is now explicitly linked against libdl on platforms with dl.

- bpo-32734 [https://bugs.python.org/issue?@action=redirect&bpo=32734]: Fixed `asyncio.Lock()` safety issue which allowed acquiring and locking the same lock multiple times, without it being free. Patch by Bar Harel.
- bpo-32727 [https://bugs.python.org/issue?@action=redirect&bpo=32727]: Do not include name field in SMTP envelope from address. Patch by Stéphane Wirtel
- bpo-27931 [https://bugs.python.org/issue?@action=redirect&bpo=27931]: Fix email address header parsing error when the username is an empty quoted string. Patch by Xiang Zhang.
- bpo-32304 [https://bugs.python.org/issue?@action=redirect&bpo=32304]: distutils" upload command no longer corrupts tar files ending with a CR byte, and no longer tries to convert CR to CRLF in any of the upload text fields.
- bpo-32502 [https://bugs.python.org/issue?@action=redirect&bpo=32502]: uuid.uuid1 no longer raises an exception if a 64-bit hardware address is encountered.
- bpo-31848 [https://bugs.python.org/issue?@action=redirect&bpo=31848]: Fix the error handling in Aifc_read.initfp() when the SSND chunk is not found. Patch by Zackery Spytz.
- bpo-32555 [https://bugs.python.org/issue?@action=redirect&bpo=32555]: On FreeBSD and Solaris, os.strerror() now always decode the byte string from the current locale encoding, rather than using ASCII/surrogateescape in some cases.
- bpo-32521 [https://bugs.python.org/issue?@action=redirect&bpo=32521]: The nis module is now compatible with new libnsl and headers location.
- bpo-32473 [https://bugs.python.org/issue?@action=redirect&bpo=32473]: Improve ABCMeta._dump_registry() output readability
- bpo-32521 [https://bugs.python.org/issue?@action=redirect&bpo=32521]: glibc has removed Sun RPC. Use replacement libtirpc headers and library in nis module.
- bpo-32228 [https://bugs.python.org/issue?@action=redirect&bpo=32228]: Ensure that `truncate()`

preserves the file position (as reported by `tell()`) after writes longer than the buffer size.

- bpo-26133 [https://bugs.python.org/issue?@action=redirect&bpo=26133]: Don’t unsubscribe signals in asyncio UNIX event loop on interpreter shutdown.
- bpo-32185 [https://bugs.python.org/issue?@action=redirect&bpo=32185]: The SSL module no longer sends IP addresses in SNI TLS extension on platforms with OpenSSL 1.0.2+ or inet_pton.
- bpo-32323 [https://bugs.python.org/issue?@action=redirect&bpo=32323]: **`urllib.parse.urlsplit()`** does not convert zone-id (scope) to lower case for scoped IPv6 addresses in hostnames now.
- bpo-32302 [https://bugs.python.org/issue?@action=redirect&bpo=32302]: Fix bdist_wininst of distutils for CRT v142: it binary compatible with CRT v140.
- bpo-32255 [https://bugs.python.org/issue?@action=redirect&bpo=32255]: A single empty field is now always quoted when written into a CSV file. This allows to distinguish an empty row from a row consisting of a single empty field. Patch by Licht Takeuchi.
- bpo-32277 [https://bugs.python.org/issue?@action=redirect&bpo=32277]: Raise `NotImplementedError` instead of `SystemError` on platforms where `chmod(..., follow_symlinks=False)` is not supported. Patch by Anthony Sottile.
- bpo-32199 [https://bugs.python.org/issue?@action=redirect&bpo=32199]: The getnode() ip getter now uses “ip link” instead of “ip link list”.
- bpo-27456 [https://bugs.python.org/issue?@action=redirect&bpo=27456]: Ensure TCP_NODELAY is set on Linux. Tests by Victor Stinner.
- bpo-31900 [https://bugs.python.org/issue?@action=redirect&bpo=31900]: The **`locale.localeconv()`** function now sets temporarily the `LC_CTYPE` locale to the `LC_NUMERIC` locale to decode `decimal_point` and `thousands_sep` byte strings if they are non-ASCII or longer than 1 byte, and the `LC_NUMERIC` locale is different than the `LC_CTYPE` locale. This temporary change affects other threads. Same change for the **`str.format()`** method when

formatting a number (**`int`**, **`float`**, **`float`** and subclasses) with the `n` type (ex: `'{:n}'.format(1234)`).

- bpo-31802 [https://bugs.python.org/issue?@action=redirect&bpo=31802]: Importing native path module (`posixpath`, `ntpath`) now works even if the `os` module still is not imported.

Documentation

- bpo-17232 [https://bugs.python.org/issue?@action=redirect&bpo=17232]: Clarify docs for -O and -OO. Patch by Terry Reedy.
- bpo-32800 [https://bugs.python.org/issue?@action=redirect&bpo=32800]: Update link to w3c doc for xml default namespaces.
- bpo-8722 [https://bugs.python.org/issue?@action=redirect&bpo=8722]: Document **`__getattr__()`** behavior when property **`get()`** method raises **`AttributeError`**.
- bpo-32614 [https://bugs.python.org/issue?@action=redirect&bpo=32614]: Modify RE examples in documentation to use raw strings to prevent **`DeprecationWarning`** and add text to REGEX HOWTO to highlight the deprecation.
- bpo-31972 [https://bugs.python.org/issue?@action=redirect&bpo=31972]: Improve docstrings for `pathlib.PurePath` subclasses.
- bpo-17799 [https://bugs.python.org/issue?@action=redirect&bpo=17799]: Explain real behaviour of sys.settrace and sys.setprofile and their C-API counterparts regarding which type of events are received in each function. Patch by Pablo Galindo Salgado.

Tests

- bpo-32517 [https://bugs.python.org/issue?@action=redirect&bpo=32517]: Fix failing `test_asyncio` on macOS 10.12.2+ due to transport of `KqueueSelector` loop was not being closed.
- bpo-32721 [https://bugs.python.org/issue?@action=redirect&bpo=32721]: Fix test_hashlib to not fail if the

_md5 module is not built.

- bpo-32252 [https://bugs.python.org/issue?@action=redirect&bpo=32252]: Fix faulthandler_suppress_crash_report() used to prevent core dump files when testing crashes. getrlimit() returns zero on success.
- bpo-31518 [https://bugs.python.org/issue?@action=redirect&bpo=31518]: Debian Unstable has disabled TLS 1.0 and 1.1 for SSLv23_METHOD(). Change TLS/SSL protocol of some tests to PROTOCOL_TLS or PROTOCOL_TLSv1_2 to make them pass on Debian.

Build

- bpo-32635 [https://bugs.python.org/issue?@action=redirect&bpo=32635]: Fix segfault of the crypt module when libxcrypt is provided instead of libcrypt at the system.

Windows

- bpo-33016 [https://bugs.python.org/issue?@action=redirect&bpo=33016]: Fix potential use of uninitialized memory in nt._getfinalpathname
- bpo-32903 [https://bugs.python.org/issue?@action=redirect&bpo=32903]: Fix a memory leak in os.chdir() on Windows if the current directory is set to a UNC path.
- bpo-31966 [https://bugs.python.org/issue?@action=redirect&bpo=31966]: Fixed WindowsConsoleIO.write() for writing empty data.
- bpo-32409 [https://bugs.python.org/issue?@action=redirect&bpo=32409]: Ensures activate.bat can handle Unicode contents.
- bpo-32457 [https://bugs.python.org/issue?@action=redirect&bpo=32457]: Improves handling of denormalized executable path when launching Python.
- bpo-32370 [https://bugs.python.org/issue?@action=redirect&bpo=32370]: Use the correct encoding for ipconfig output in the uuid module. Patch by Segev Finer.
- bpo-29248 [https://bugs.python.org/issue?@action=redirect&bpo=29248]: Fix `os.readlink()` on

Windows, which was mistakenly treating the `PrintNameOffset` field of the reparse data buffer as a number of characters instead of bytes. Patch by Craig Holmquist and SSE4.

- bpo-32588 [https://bugs.python.org/issue?@action=redirect&bpo=32588]: Create standalone _distutils_findvs module.

macOS

- bpo-32726 [https://bugs.python.org/issue?@action=redirect&bpo=32726]: Provide an additional, more modern macOS installer variant that supports macOS 10.9+ systems in 64-bit mode only. Upgrade the supplied third-party libraries to OpenSSL 1.0.2n, XZ 5.2.3, and SQLite 3.22.0. The 10.9+ installer now links with and supplies its own copy of Tcl/Tk 8.6.8.

IDLE

- bpo-32984 [https://bugs.python.org/issue?@action=redirect&bpo=32984]: Set `__file__` while running a startup file. Like Python, IDLE optionally runs one startup file in the Shell window before presenting the first interactive input prompt. For IDLE, `-s` runs a file named in environmental variable **`IDLESTARTUP`** or **`PYTHONSTARTUP`**; `-r file` runs `file`. Python sets `__file__` to the startup file name before running the file and unsets it before the first prompt. IDLE now does the same when run normally, without the `-n` option.
- bpo-32940 [https://bugs.python.org/issue?@action=redirect&bpo=32940]: Simplify and rename StringTranslatePseudoMapping in pyparse.
- bpo-32916 [https://bugs.python.org/issue?@action=redirect&bpo=32916]: Change `str` to `code` in pyparse.
- bpo-32905 [https://bugs.python.org/issue?@action=redirect&bpo=32905]: Remove unused code in pyparse module.
- bpo-32874 [https://bugs.python.org/issue?@action=redirect&bpo=32874]: Add tests for pyparse.

- bpo-32837 [https://bugs.python.org/issue?@action=redirect&bpo=32837]: Using the system and place-dependent default encoding for open() is a bad idea for IDLE's system and location-independent files.
- bpo-32826 [https://bugs.python.org/issue?@action=redirect&bpo=32826]: Add «encoding=utf-8» to open() in IDLE's test_help_about. GUI test test_file_buttons() only looks at initial ascii-only lines, but failed on systems where open() defaults to "ascii" because readline() internally reads and decodes far enough ahead to encounter a non-ascii character in CREDITS.txt.
- bpo-32765 [https://bugs.python.org/issue?@action=redirect&bpo=32765]: Update configdialog General tab docstring to add new widgets to the widget list.

Tools/Demos

- bpo-24960 [https://bugs.python.org/issue?@action=redirect&bpo=24960]: 2to3 and lib2to3 can now read pickled grammar files using pkgutil.get_data() rather than probing the filesystem. This lets 2to3 and lib2to3 work when run from a zipfile.
- bpo-32222 [https://bugs.python.org/issue?@action=redirect&bpo=32222]: Fix pygettext not extracting docstrings for functions with type annotated arguments. Patch by Toby Harradine.

C API

- bpo-29084 [https://bugs.python.org/issue?@action=redirect&bpo=29084]: Undocumented C API for OrderedDict has been excluded from the limited C API. It was added by mistake and actually never worked in the limited C API.

Python 3.6.4 final

Release date: 2017-12-18

There were no new code changes in version 3.6.4 since v3.6.4rc1.

Python 3.6.4 release candidate 1

Release date: 2017-12-05

Core and Builtins

- bpo-32176 [https://bugs.python.org/issue?@action=redirect&bpo=32176]: co_flags.CO_NOFREE is now always set correctly by the code object constructor based on freevars and cellvars, rather than needing to be set correctly by the caller. This ensures it will be cleared automatically when additional cell references are injected into a modified code object and function.
- bpo-31949 [https://bugs.python.org/issue?@action=redirect&bpo=31949]: Fixed several issues in printing tracebacks (PyTraceBack_Print()). Setting sys.tracebacklimit to 0 or less now suppresses printing tracebacks. Setting sys.tracebacklimit to None now causes using the default limit. Setting sys.tracebacklimit to an integer larger than LONG_MAX now means using the limit LONG_MAX rather than the default limit. Fixed integer overflows in the case of more than `2**31` traceback items on Windows. Fixed output errors handling.
- bpo-30696 [https://bugs.python.org/issue?@action=redirect&bpo=30696]: Fix the interactive interpreter looping endlessly when no memory.
- bpo-20047 [https://bugs.python.org/issue?@action=redirect&bpo=20047]: Bytearray methods partition() and rpartition() now accept only bytes-like objects as separator, as documented. In particular they now raise TypeError rather of returning a bogus result when an integer is passed as a separator.
- bpo-31852 [https://bugs.python.org/issue?@action=redirect&bpo=31852]: Fix a segmentation fault caused by a combination of the async soft keyword and continuation lines.
- bpo-21720 [https://bugs.python.org/issue?

@action=redirect&bpo=21720]: BytesWarning no longer emitted when the *fromlist* argument of `__import__()` or the `__all__` attribute of the module contain bytes instances.

- bpo-31825 [https://bugs.python.org/issue?@action=redirect&bpo=31825]: Fixed OverflowError in the "unicode-escape" codec and in codecs.escape_decode() when decode an escaped non-ascii byte.
- bpo-28603 [https://bugs.python.org/issue?@action=redirect&bpo=28603]: Print the full context/cause chain of exceptions on interpreter exit, even if an exception in the chain is unhashable or compares equal to later ones. Patch by Zane Bitter.
- bpo-31786 [https://bugs.python.org/issue?@action=redirect&bpo=31786]: Fix timeout rounding in the select module to round correctly negative timeouts between -1.0 and 0.0. The functions now block waiting for events as expected. Previously, the call was incorrectly non-blocking. Patch by Pablo Galindo.
- bpo-31642 [https://bugs.python.org/issue?@action=redirect&bpo=31642]: Restored blocking «from package import module» by setting sys.modules[«package.module»] to None.
- bpo-31626 [https://bugs.python.org/issue?@action=redirect&bpo=31626]: Fixed a bug in debug memory allocator. There was a write to freed memory after shrinking a memory block.
- bpo-31619 [https://bugs.python.org/issue?@action=redirect&bpo=31619]: Fixed a ValueError when convert a string with large number of underscores to integer with binary base.
- bpo-31592 [https://bugs.python.org/issue?@action=redirect&bpo=31592]: Fixed an assertion failure in Python parser in case of a bad `unicodedata.normalize()`. Patch by Oren Milman.
- bpo-31588 [https://bugs.python.org/issue?@action=redirect&bpo=31588]: Raise a `TypeError` with a helpful error message when class creation fails due to a metaclass with a bad `__prepare__()` method. Patch by Oren Milman.
- bpo-31566 [https://bugs.python.org/issue?

@action=redirect&bpo=31566]: Fix an assertion failure in `_warnings.warn()` in case of a bad `__name__` global. Patch by Oren Milman.

- bpo-31505 [https://bugs.python.org/issue?@action=redirect&bpo=31505]: Fix an assertion failure in `json`, in case `_json.make_encoder()` received a bad `encoder()` argument. Patch by Oren Milman.
- bpo-31492 [https://bugs.python.org/issue?@action=redirect&bpo=31492]: Fix assertion failures in case of failing to import from a module with a bad `__name__` attribute, and in case of failing to access an attribute of such a module. Patch by Oren Milman.
- bpo-31490 [https://bugs.python.org/issue?@action=redirect&bpo=31490]: Fix an assertion failure in `ctypes` class definition, in case the class has an attribute whose name is specified in `_anonymous_` but not in `_fields_`. Patch by Oren Milman.
- bpo-31478 [https://bugs.python.org/issue?@action=redirect&bpo=31478]: Fix an assertion failure in `_random.Random.seed()` in case the argument has a bad `__abs__()` method. Patch by Oren Milman.
- bpo-31315 [https://bugs.python.org/issue?@action=redirect&bpo=31315]: Fix an assertion failure in imp.create_dynamic(), when spec.name is not a string. Patch by Oren Milman.
- bpo-31311 [https://bugs.python.org/issue?@action=redirect&bpo=31311]: Fix a crash in the `__setstate__()` method of `ctypes._CData`, in case of a bad `__dict__`. Patch by Oren Milman.
- bpo-31293 [https://bugs.python.org/issue?@action=redirect&bpo=31293]: Fix crashes in true division and multiplication of a timedelta object by a float with a bad as_integer_ratio() method. Patch by Oren Milman.
- bpo-31285 [https://bugs.python.org/issue?@action=redirect&bpo=31285]: Fix an assertion failure in `warnings.warn_explicit`, when the return value of the received loader's `get_source()` has a bad `splitlines()` method. Patch by Oren Milman.
- bpo-30817 [https://bugs.python.org/issue?@action=redirect&bpo=30817]: `PyErr_PrintEx()` clears now

the ignored exception that may be raised by `_PySys_SetObjectId()`, for example when no memory.

Library

- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Two minor fixes for `typing` module: allow shallow copying instances of generic classes, improve interaction of `__init_subclass__` with generics. Original PRs by Ivan Levkivskyi.
- bpo-27240 [https://bugs.python.org/issue?@action=redirect&bpo=27240]: The header folding algorithm for the new email policies has been rewritten, which also fixes bpo-30788 [https://bugs.python.org/issue?@action=redirect&bpo=30788], bpo-31831 [https://bugs.python.org/issue?@action=redirect&bpo=31831], and bpo-32182 [https://bugs.python.org/issue?@action=redirect&bpo=32182]. In particular, RFC2231 folding is now done correctly.
- bpo-32186 [https://bugs.python.org/issue?@action=redirect&bpo=32186]: io.FileIO.readall() and io.FileIO.read() now release the GIL when getting the file size. Fixed hang of all threads with inaccessible NFS server. Patch by Nir Soffer.
- bpo-12239 [https://bugs.python.org/issue?@action=redirect&bpo=12239]: Make **`msilib.SummaryInformation.GetProperty()`** return `None` when the value of property is `VT_EMPTY`. Initial patch by Mark Mc Mahon.
- bpo-31325 [https://bugs.python.org/issue?@action=redirect&bpo=31325]: Fix wrong usage of **`collections.namedtuple()`** in the **`RobotFileParser.parse()`** method. Initial patch by Robin Wellner.
- bpo-12382 [https://bugs.python.org/issue?@action=redirect&bpo=12382]: **`msilib.OpenDatabase()`** now raises a better exception message when it couldn’t open or create an MSI file. Initial patch by William Tisäter.
- bpo-32110 [https://bugs.python.org/issue?@action=redirect&bpo=32110]: `codecs.StreamReader.read(n)` now returns not more

than *n* characters/bytes for non-negative *n*. This makes it compatible with `read()` methods of other file-like objects.

- bpo-32072 [https://bugs.python.org/issue?@action=redirect&bpo=32072]: Fixed issues with binary plists: Fixed saving bytearrays. Identical objects will be saved only once. Equal references will be load as identical objects. Added support for saving and loading recursive data structures.
- bpo-32034 [https://bugs.python.org/issue?@action=redirect&bpo=32034]: Make asyncio.IncompleteReadError and LimitOverrunError pickleable.
- bpo-32015 [https://bugs.python.org/issue?@action=redirect&bpo=32015]: Fixed the looping of asyncio in the case of reconnection the socket during waiting async read/write from/to the socket.
- bpo-32011 [https://bugs.python.org/issue?@action=redirect&bpo=32011]: Restored support of loading marshal files with the TYPE_INT64 code. These files can be produced in Python 2.7.
- bpo-31970 [https://bugs.python.org/issue?@action=redirect&bpo=31970]: Reduce performance overhead of asyncio debug mode.
- bpo-9678 [https://bugs.python.org/issue?@action=redirect&bpo=9678]: Fixed determining the MAC address in the uuid module: Using ifconfig on NetBSD and OpenBSD. Using arp on Linux, FreeBSD, NetBSD and OpenBSD. Based on patch by Takayuki Shimizukawa.
- bpo-30057 [https://bugs.python.org/issue?@action=redirect&bpo=30057]: Fix potential missed signal in signal.signal().
- bpo-31933 [https://bugs.python.org/issue?@action=redirect&bpo=31933]: Fix Blake2 params leaf_size and node_offset on big endian platforms. Patch by Jack O’Connor.
- bpo-31927 [https://bugs.python.org/issue?@action=redirect&bpo=31927]: Fixed compilation of the socket module on NetBSD 8. Fixed assertion failure or reading arbitrary data when parse a AF_BLUETOOTH address on NetBSD and DragonFly BSD.
- bpo-27666 [https://bugs.python.org/issue?@action=redirect&bpo=27666]: Fixed stack corruption in

curses.box() and curses.ungetmouse() when the size of types chtype or mmask_t is less than the size of C long. curses.box() now accepts characters as arguments. Based on patch by Steve Fink.

- bpo-31897 [https://bugs.python.org/issue?@action=redirect&bpo=31897]: plistlib now catches more errors when read binary plists and raises InvalidFileException instead of unexpected exceptions.
- bpo-25720 [https://bugs.python.org/issue?@action=redirect&bpo=25720]: Fix the method for checking pad state of curses WINDOW. Patch by Masayuki Yamamoto.
- bpo-31893 [https://bugs.python.org/issue?@action=redirect&bpo=31893]: Fixed the layout of the kqueue_event structure on OpenBSD and NetBSD. Fixed the comparison of the kqueue_event objects.
- bpo-31891 [https://bugs.python.org/issue?@action=redirect&bpo=31891]: Fixed building the curses module on NetBSD.
- bpo-28416 [https://bugs.python.org/issue?@action=redirect&bpo=28416]: Instances of pickle.Pickler subclass with the persistent_id() method and pickle.Unpickler subclass with the persistent_load() method no longer create reference cycles.
- bpo-28326 [https://bugs.python.org/issue?@action=redirect&bpo=28326]: Fix multiprocessing.Process when stdout and/or stderr is closed or None.
- bpo-31457 [https://bugs.python.org/issue?@action=redirect&bpo=31457]: If nested log adapters are used, the inner `process()` methods are no longer omitted.
- bpo-31457 [https://bugs.python.org/issue?@action=redirect&bpo=31457]: The `manager` property on LoggerAdapter objects is now properly settable.
- bpo-31806 [https://bugs.python.org/issue?@action=redirect&bpo=31806]: Fix timeout rounding in time.sleep(), threading.Lock.acquire() and socket.socket.settimeout() to round correctly negative timeouts between -1.0 and 0.0. The functions now block waiting for events as expected. Previously, the call was incorrectly non-blocking. Patch by Pablo Galindo.
- bpo-28603 [https://bugs.python.org/issue?

@action=redirect&bpo=28603]: traceback: Fix a TypeError that occurred during printing of exception tracebacks when either the current exception or an exception in its context/cause chain is unhashable. Patch by Zane Bitter.

- bpo-30058 [https://bugs.python.org/issue?@action=redirect&bpo=30058]: Fixed buffer overflow in select.kqueue.control().
- bpo-31770 [https://bugs.python.org/issue?@action=redirect&bpo=31770]: Prevent a crash when calling the `__init__()` method of a `sqlite3.Cursor` object more than once. Patch by Oren Milman.
- bpo-31672 [https://bugs.python.org/issue?@action=redirect&bpo=31672]: `idpattern` in `string.Template` matched some non-ASCII characters. Now it uses `-i` regular expression local flag to avoid non-ASCII characters.
- bpo-31764 [https://bugs.python.org/issue?@action=redirect&bpo=31764]: Prevent a crash in `sqlite3.Cursor.close()` in case the `Cursor` object is uninitialized. Patch by Oren Milman.
- bpo-31752 [https://bugs.python.org/issue?@action=redirect&bpo=31752]: Fix possible crash in timedelta constructor called with custom integers.
- bpo-31701 [https://bugs.python.org/issue?@action=redirect&bpo=31701]: On Windows, faulthandler.enable() now ignores MSC and COM exceptions.
- bpo-31728 [https://bugs.python.org/issue?@action=redirect&bpo=31728]: Prevent crashes in `_elementtree` due to unsafe cleanup of `Element.text` and `Element.tail`. Patch by Oren Milman.
- bpo-31620 [https://bugs.python.org/issue?@action=redirect&bpo=31620]: an empty asyncio.Queue now doesn't leak memory when queue.get pollers timeout
- bpo-31632 [https://bugs.python.org/issue?@action=redirect&bpo=31632]: Fix method set_protocol() of class _SSLProtocolTransport in asyncio module. This method was previously modifying a wrong reference to the protocol.
- bpo-31675 [https://bugs.python.org/issue?@action=redirect&bpo=31675]: Fixed memory leaks in Tkinter's methods splitlist() and split() when pass a string larger than 2

GiB.

- bpo-31673 [https://bugs.python.org/issue?@action=redirect&bpo=31673]: Fixed typo in the name of Tkinter's method adderrorinfo().
- bpo-30806 [https://bugs.python.org/issue?@action=redirect&bpo=30806]: Fix the string representation of a netrc object.
- bpo-15037 [https://bugs.python.org/issue?@action=redirect&bpo=15037]: Added a workaround for getkey() in curses for ncurses 5.7 and earlier.
- bpo-25351 [https://bugs.python.org/issue?@action=redirect&bpo=25351]: Avoid venv activate failures with undefined variables
- bpo-25532 [https://bugs.python.org/issue?@action=redirect&bpo=25532]: inspect.unwrap() will now only try to unwrap an object sys.getrecursionlimit() times, to protect against objects which create a new object on every attribute access.
- bpo-30347 [https://bugs.python.org/issue?@action=redirect&bpo=30347]: Stop crashes when concurrently iterate over itertools.groupby() iterators.
- bpo-31516 [https://bugs.python.org/issue?@action=redirect&bpo=31516]: `threading.current_thread()` should not return a dummy thread at shutdown.
- bpo-31351 [https://bugs.python.org/issue?@action=redirect&bpo=31351]: python -m ensurepip now exits with non-zero exit code if pip bootstrapping has failed.
- bpo-31482 [https://bugs.python.org/issue?@action=redirect&bpo=31482]: `random.seed()` now works with bytes in version=1
- bpo-31334 [https://bugs.python.org/issue?@action=redirect&bpo=31334]: Fix `poll.poll([timeout])` in the `select` module for arbitrary negative timeouts on all OSes where it can only be a non-negative integer or -1. Patch by Riccardo Coccioli.
- bpo-31310 [https://bugs.python.org/issue?@action=redirect&bpo=31310]: multiprocessing's semaphore tracker should be launched again if crashed.
- bpo-31308 [https://bugs.python.org/issue?

@action=redirect&bpo=31308]: Make multiprocessing's forkserver process immune to Ctrl-C and other user interruptions. If it crashes, restart it when necessary.

Documentation

- bpo-32105 [https://bugs.python.org/issue?@action=redirect&bpo=32105]: Added asyncio.BaseEventLoop.connect_accepted_socket versionadded marker.
- bpo-31537 [https://bugs.python.org/issue?@action=redirect&bpo=31537]: Fix incorrect usage of `get_history_length` in readline documentation example code. Patch by Brad Smith.
- bpo-30085 [https://bugs.python.org/issue?@action=redirect&bpo=30085]: The operator functions without double underscores are preferred for clarity. The one with underscores are only kept for back-compatibility.

Tests

- bpo-31380 [https://bugs.python.org/issue?@action=redirect&bpo=31380]: Skip test_httpservers test_undecodable_file on macOS: fails on APFS.
- bpo-31705 [https://bugs.python.org/issue?@action=redirect&bpo=31705]: Skip test_socket.test_sha256() on Linux kernel older than 4.5. The test fails with ENOKEY on kernel 3.10 (on ppc64le). A fix was merged into the kernel 4.5.
- bpo-31174 [https://bugs.python.org/issue?@action=redirect&bpo=31174]: Fix test_tools.test_unparse: DirectoryTestCase now stores the names sample to always test the same files. It prevents false alarms when hunting reference leaks.
- bpo-30695 [https://bugs.python.org/issue?@action=redirect&bpo=30695]: Add the `set_nomemory(start, stop)` and `remove_mem_hooks()` functions to the `_testcapi` module.

Build

- bpo-32059 [https://bugs.python.org/issue?@action=redirect&bpo=32059]: `detect_modules()` in `setup.py` now also searches the sysroot paths when cross-compiling.
- bpo-31957 [https://bugs.python.org/issue?@action=redirect&bpo=31957]: Fixes Windows SDK version detection when building for Windows.
- bpo-31609 [https://bugs.python.org/issue?@action=redirect&bpo=31609]: Fixes quotes in PCbuild/clean.bat
- bpo-31934 [https://bugs.python.org/issue?@action=redirect&bpo=31934]: Abort the build when building out of a not clean source tree.
- bpo-31926 [https://bugs.python.org/issue?@action=redirect&bpo=31926]: Fixed Argument Clinic sometimes causing compilation errors when there was more than one function and/or method in a .c file with the same name.
- bpo-28791 [https://bugs.python.org/issue?@action=redirect&bpo=28791]: Update Windows builds to use SQLite 3.21.0.
- bpo-28791 [https://bugs.python.org/issue?@action=redirect&bpo=28791]: Update OS X installer to use SQLite 3.21.0.
- bpo-22140 [https://bugs.python.org/issue?@action=redirect&bpo=22140]: Prevent double substitution of prefix in python-config.sh.
- bpo-31536 [https://bugs.python.org/issue?@action=redirect&bpo=31536]: Avoid wholesale rebuild after `make regen-all` if nothing changed.

Windows

- bpo-1102 [https://bugs.python.org/issue?@action=redirect&bpo=1102]: Return `None` when `View.Fetch()` returns `ERROR_NO_MORE_ITEMS` instead of raising `MSIError`. Initial patch by Anthony Tuininga.
- bpo-31944 [https://bugs.python.org/issue?@action=redirect&bpo=31944]: Fixes Modify button in Apps and Features dialog.

macOS

- bpo-31392 [https://bugs.python.org/issue?@action=redirect&bpo=31392]: Update macOS installer to use OpenSSL 1.0.2m

IDLE

- bpo-32207 [https://bugs.python.org/issue?@action=redirect&bpo=32207]: Improve tk event exception tracebacks in IDLE. When tk event handling is driven by IDLE's run loop, a confusing and distracting queue.EMPTY traceback context is no longer added to tk event exception tracebacks. The traceback is now the same as when event handling is driven by user code. Patch based on a suggestion by Serhiy Storchaka.
- bpo-32164 [https://bugs.python.org/issue?@action=redirect&bpo=32164]: Delete unused file idlelib/tabbedpages.py. Use of TabbedPageSet in configdialog was replaced by ttk.Notebook.
- bpo-32100 [https://bugs.python.org/issue?@action=redirect&bpo=32100]: IDLE: Fix old and new bugs in pathbrowser; improve tests. Patch mostly by Cheryl Sabella.
- bpo-31858 [https://bugs.python.org/issue?@action=redirect&bpo=31858]: IDLE – Restrict shell prompt manipulation to the shell. Editor and output windows only see an empty last prompt line. This simplifies the code and fixes a minor bug when newline is inserted. Sys.ps1, if present, is read on Shell start-up, but is not set or changed.
- bpo-31860 [https://bugs.python.org/issue?@action=redirect&bpo=31860]: The font sample in the IDLE configuration dialog is now editable. Changes persist while IDLE remains open
- bpo-31836 [https://bugs.python.org/issue?@action=redirect&bpo=31836]: Test_code_module now passes if run after test_idle, which sets ps1. The code module uses sys.ps1 if present or sets it to “> > > “ if not. Test_code_module now properly tests both behaviors. Ditto for ps2.
- bpo-28603 [https://bugs.python.org/issue?

@action=redirect&bpo=28603]: Fix a TypeError that caused a shell restart when printing a traceback that includes an exception that is unhashable. Patch by Zane Bitter.

- bpo-13802 [https://bugs.python.org/issue?@action=redirect&bpo=13802]: Use non-Latin characters in the IDLE's Font settings sample. Even if one selects a font that defines a limited subset of the unicode Basic Multilingual Plane, tcl/tk will use other fonts that define a character. The expanded example give users of non-Latin characters a better idea of what they might see in IDLE's shell and editors. To make room for the expanded sample, frames on the Font tab are re-arranged. The Font/Tabs help explains a bit about the additions.
- bpo-31460 [https://bugs.python.org/issue?@action=redirect&bpo=31460]: Simplify the API of IDLE's Module Browser. Passing a widget instead of an flist with a root widget opens the option of creating a browser frame that is only part of a window. Passing a full file name instead of pieces assumed to come from a .py file opens the possibility of browsing python files that do not end in .py.
- bpo-31649 [https://bugs.python.org/issue?@action=redirect&bpo=31649]: IDLE - Make _htest, _utest parameters keyword only.
- bpo-31559 [https://bugs.python.org/issue?@action=redirect&bpo=31559]: Remove test order dependence in idle_test.test_browser.
- bpo-31459 [https://bugs.python.org/issue?@action=redirect&bpo=31459]: Rename IDLE's module browser from Class Browser to Module Browser. The original module-level class and method browser became a module browser, with the addition of module-level functions, years ago. Nested classes and functions were added yesterday. For back-compatibility, the virtual event <<open-class-browser>>, which appears on the Keys tab of the Settings dialog, is not changed. Patch by Cheryl Sabella.
- bpo-31500 [https://bugs.python.org/issue?@action=redirect&bpo=31500]: Default fonts now are scaled on HiDPI displays.
- bpo-1612262 [https://bugs.python.org/issue?@action=redirect&bpo=1612262]: IDLE module browser now

shows nested classes and functions. Original patches for code and tests by Guilherme Polo and Cheryl Sabella, respectively.

Tools/Demos

- bpo-30722 [https://bugs.python.org/issue?@action=redirect&bpo=30722]: Make redemo work with Python 3.6 and newer versions. Also, remove the `LOCALE` option since it doesn't work with string patterns in Python 3. Patch by Christoph Sarnowski.

C API

- bpo-20891 [https://bugs.python.org/issue?@action=redirect&bpo=20891]: Fix PyGILState_Ensure(). When PyGILState_Ensure() is called in a non-Python thread before PyEval_InitThreads(), only call PyEval_InitThreads() after calling PyThreadState_New() to fix a crash.
- bpo-31532 [https://bugs.python.org/issue?@action=redirect&bpo=31532]: Fix memory corruption due to allocator mix in getpath.c between Py_GetPath() and Py_SetPath()
- bpo-30697 [https://bugs.python.org/issue?@action=redirect&bpo=30697]: The `PyExc_RecursionErrorInst` singleton is removed and `PyErr_NormalizeException()` does not use it anymore. This singleton is persistent and its members being never cleared may cause a segfault during finalization of the interpreter. See also bpo-22898 [https://bugs.python.org/issue?@action=redirect&bpo=22898].

Python 3.6.3 final

Release date: 2017-10-03

Library

- bpo-31641 [https://bugs.python.org/issue?@action=redirect&bpo=31641]: Re-allow arbitrary iterables in `concurrent.futures.as_completed()`. Fixes regression in 3.6.3rc1.

Build

- bpo-31662 [https://bugs.python.org/issue?@action=redirect&bpo=31662]: Fix typos in Windows `uploadrelease.bat` script. Fix Windows Doc build issues in `Doc/make.bat`.
- bpo-31423 [https://bugs.python.org/issue?@action=redirect&bpo=31423]: Fix building the PDF documentation with newer versions of Sphinx.

Python 3.6.3 release candidate 1

Release date: 2017-09-18

Security

- bpo-29781 [https://bugs.python.org/issue?@action=redirect&bpo=29781]: SSLObject.version() now correctly returns None when handshake over BIO has not been performed yet.
- bpo-30947 [https://bugs.python.org/issue?@action=redirect&bpo=30947]: Upgrade libexpat embedded copy from version 2.2.1 to 2.2.3 to get security fixes.

Core and Builtins

- bpo-31471 [https://bugs.python.org/issue?@action=redirect&bpo=31471]: Fix an assertion failure in `subprocess.Popen()` on Windows, in case the env argument has a bad `keys()` method. Patch by Oren Milman.
- bpo-31418 [https://bugs.python.org/issue?@action=redirect&bpo=31418]: Fix an assertion failure in `PyErr_WriteUnraisable()` in case of an exception with a bad `__module__` attribute. Patch by Oren Milman.
- bpo-31416 [https://bugs.python.org/issue?@action=redirect&bpo=31416]: Fix assertion failures in case of a

bad warnings.filters or warnings.defaultaction. Patch by Oren Milman.

- bpo-31411 [https://bugs.python.org/issue?@action=redirect&bpo=31411]: Raise a TypeError instead of SystemError in case warnings.onceregistry is not a dictionary. Patch by Oren Milman.
- bpo-31373 [https://bugs.python.org/issue?@action=redirect&bpo=31373]: Fix several possible instances of undefined behavior due to floating-point demotions.
- bpo-30465 [https://bugs.python.org/issue?@action=redirect&bpo=30465]: Location information (`lineno` and `col_offset`) in f-strings is now (mostly) correct. This fixes tools like flake8 from showing warnings on the wrong line (typically the first line of the file).
- bpo-31343 [https://bugs.python.org/issue?@action=redirect&bpo=31343]: Include sys/sysmacros.h for major(), minor(), and makedev(). GNU C library plans to remove the functions from sys/types.h.
- bpo-31291 [https://bugs.python.org/issue?@action=redirect&bpo=31291]: Fix an assertion failure in `zipimport.zipimporter.get_data` on Windows, when the return value of `pathname.replace('/','\\')` isn't a string. Patch by Oren Milman.
- bpo-31271 [https://bugs.python.org/issue?@action=redirect&bpo=31271]: Fix an assertion failure in the `write()` method of `io.TextIOWrapper`, when the encoder doesn't return a bytes object. Patch by Oren Milman.
- bpo-31243 [https://bugs.python.org/issue?@action=redirect&bpo=31243]: Fix a crash in some methods of `io.TextIOWrapper`, when the decoder's state is invalid. Patch by Oren Milman.
- bpo-30721 [https://bugs.python.org/issue?@action=redirect&bpo=30721]: `print` now shows correct usage hint for using Python 2 redirection syntax. Patch by Sanyam Khurana.
- bpo-31070 [https://bugs.python.org/issue?@action=redirect&bpo=31070]: Fix a race condition in importlib _get_module_lock().
- bpo-31095 [https://bugs.python.org/issue?@action=redirect&bpo=31095]: Fix potential crash during GC

caused by `tp_dealloc` which doesn't call `PyObject_GC_UnTrack()`.

- bpo-31071 [https://bugs.python.org/issue?@action=redirect&bpo=31071]: Avoid masking original TypeError in call with * unpacking when other arguments are passed.
- bpo-30978 [https://bugs.python.org/issue?@action=redirect&bpo=30978]: str.format_map() now passes key lookup exceptions through. Previously any exception was replaced with a KeyError exception.
- bpo-30808 [https://bugs.python.org/issue?@action=redirect&bpo=30808]: Use _Py_atomic API for concurrency-sensitive signal state.
- bpo-30876 [https://bugs.python.org/issue?@action=redirect&bpo=30876]: Relative import from unloaded package now reimports the package instead of failing with SystemError. Relative import from non-package now fails with ImportError rather than SystemError.
- bpo-30703 [https://bugs.python.org/issue?@action=redirect&bpo=30703]: Improve signal delivery. Avoid using Py_AddPendingCall from signal handler, to avoid calling signal-unsafe functions. The tests I'm adding here fail without the rest of the patch, on Linux and OS X. This means our signal delivery logic had defects (some signals could be lost).
- bpo-30765 [https://bugs.python.org/issue?@action=redirect&bpo=30765]: Avoid blocking in pthread_mutex_lock() when PyThread_acquire_lock() is asked not to block.
- bpo-31161 [https://bugs.python.org/issue?@action=redirect&bpo=31161]: Make sure the "Missing parentheses" syntax error message is only applied to SyntaxError, not to subclasses. Patch by Martijn Pieters.
- bpo-30814 [https://bugs.python.org/issue?@action=redirect&bpo=30814]: Fixed a race condition when import a submodule from a package.
- bpo-30597 [https://bugs.python.org/issue?@action=redirect&bpo=30597]: `print` now shows expected input in custom error message when used as a Python 2 statement. Patch by Sanyam Khurana.

Library

- bpo-31499 [https://bugs.python.org/issue?@action=redirect&bpo=31499]: xml.etree: Fix a crash when a parser is part of a reference cycle.
- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: typing.get_type_hints now finds the right globalns for classes and modules by default (when no `globalns` was specified by the caller).
- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Speed improvements to the `typing` module. Original PRs by Ivan Levkivskyi and Mitar.
- bpo-31544 [https://bugs.python.org/issue?@action=redirect&bpo=31544]: The C accelerator module of ElementTree ignored exceptions raised when looking up TreeBuilder target methods in XMLParser().
- bpo-31234 [https://bugs.python.org/issue?@action=redirect&bpo=31234]: socket.create_connection() now fixes manually a reference cycle: clear the variable storing the last exception on success.
- bpo-31457 [https://bugs.python.org/issue?@action=redirect&bpo=31457]: LoggerAdapter objects can now be nested.
- bpo-31400 [https://bugs.python.org/issue?@action=redirect&bpo=31400]: Improves SSL error handling to avoid losing error numbers.
- bpo-28958 [https://bugs.python.org/issue?@action=redirect&bpo=28958]: ssl.SSLContext() now uses OpenSSL error information when a context cannot be instantiated.
- bpo-27340 [https://bugs.python.org/issue?@action=redirect&bpo=27340]: SSLSocket.sendall() now uses memoryview to create slices of data. This fixes support for all bytes-like object. It is also more efficient and avoids costly copies.
- bpo-31178 [https://bugs.python.org/issue?@action=redirect&bpo=31178]: Fix string concatenation bug in rare error path in the subprocess module
- bpo-31350 [https://bugs.python.org/issue?@action=redirect&bpo=31350]: Micro-optimize

`asyncio._get_running_loop()` to become up to 10% faster.

- bpo-31170 [https://bugs.python.org/issue?@action=redirect&bpo=31170]: expat: Update libexpat from 2.2.3 to 2.2.4. Fix copying of partial characters for UTF-8 input (libexpat bug 115): https://github.com/libexpat/libexpat/issues/115
- bpo-29136 [https://bugs.python.org/issue?@action=redirect&bpo=29136]: Add TLS 1.3 cipher suites and OP_NO_TLSv1_3.
- bpo-29212 [https://bugs.python.org/issue?@action=redirect&bpo=29212]: Fix concurrent.futures.thread.ThreadPoolExecutor threads to have a non repr() based thread name by default when no thread_name_prefix is supplied. They will now identify themselves as «ThreadPoolExecutor-y_n».
- bpo-9146 [https://bugs.python.org/issue?@action=redirect&bpo=9146]: Fix a segmentation fault in _hashopenssl when standard hash functions such as md5 are not available in the linked OpenSSL library. As in some special FIPS-140 build environments.
- bpo-27144 [https://bugs.python.org/issue?@action=redirect&bpo=27144]: The `map()` and `as_completed()` iterators in `concurrent.futures` now avoid keeping a reference to yielded objects.
- bpo-10746 [https://bugs.python.org/issue?@action=redirect&bpo=10746]: Fix ctypes producing wrong **PEP 3118** [https://peps.python.org/pep-3118/] type codes for integer types.
- bpo-22536 [https://bugs.python.org/issue?@action=redirect&bpo=22536]: The subprocess module now sets the filename when FileNotFoundError is raised on POSIX systems due to the executable or cwd not being found.
- bpo-31249 [https://bugs.python.org/issue?@action=redirect&bpo=31249]: concurrent.futures: WorkItem.run() used by ThreadPoolExecutor now breaks a reference cycle between an exception object and the WorkItem object.
- bpo-31247 [https://bugs.python.org/issue?@action=redirect&bpo=31247]: xmlrpc.server now explicitly

breaks reference cycles when using sys.exc_info() in code handling exceptions.

- bpo-30102 [https://bugs.python.org/issue?@action=redirect&bpo=30102]: The ssl and hashlib modules now call OPENSSL_add_all_algorithms_noconf() on OpenSSL < 1.1.0. The function detects CPU features and enables optimizations on some CPU architectures such as POWER8. Patch is based on research from Gustavo Serra Scalet.
- bpo-31185 [https://bugs.python.org/issue?@action=redirect&bpo=31185]: Fixed miscellaneous errors in asyncio speedup module.
- bpo-31135 [https://bugs.python.org/issue?@action=redirect&bpo=31135]: ttk: fix the destroy() method of LabeledScale and OptionMenu classes. Call the parent destroy() method even if the used attribute doesn’t exist. The LabeledScale.destroy() method now also explicitly clears label and scale attributes to help the garbage collector to destroy all widgets.
- bpo-31107 [https://bugs.python.org/issue?@action=redirect&bpo=31107]: Fix `copyreg._slotnames()` mangled attribute calculation for classes whose name begins with an underscore. Patch by Shane Harvey.
- bpo-31061 [https://bugs.python.org/issue?@action=redirect&bpo=31061]: Fixed a crash when using asyncio and threads.
- bpo-30502 [https://bugs.python.org/issue?@action=redirect&bpo=30502]: Fix handling of long oids in ssl. Based on patch by Christian Heimes.
- bpo-30119 [https://bugs.python.org/issue?@action=redirect&bpo=30119]: ftplib.FTP.putline() now throws ValueError on commands that contains CR or LF. Patch by Donghee Na.
- bpo-30595 [https://bugs.python.org/issue?@action=redirect&bpo=30595]: multiprocessing.Queue.get() with a timeout now polls its reader in non-blocking mode if it succeeded to acquire the lock but the acquire took longer than the timeout.
- bpo-29403 [https://bugs.python.org/issue?@action=redirect&bpo=29403]: Fix `unittest.mock`’s autospec to not fail on method-bound builtin functions. Patch by Aaron

Gallagher.

- bpo-30961 [https://bugs.python.org/issue?@action=redirect&bpo=30961]: Fix decrementing a borrowed reference in tracemalloc.
- bpo-25684 [https://bugs.python.org/issue?@action=redirect&bpo=25684]: Change `ttk.OptionMenu` radiobuttons to be unique across instances of `OptionMenu`.
- bpo-30886 [https://bugs.python.org/issue?@action=redirect&bpo=30886]: Fix multiprocessing.Queue.join_thread(): it now waits until the thread completes, even if the thread was started by the same process which created the queue.
- bpo-29854 [https://bugs.python.org/issue?@action=redirect&bpo=29854]: Fix segfault in readline when using readline's history-size option. Patch by Nir Soffer.
- bpo-30319 [https://bugs.python.org/issue?@action=redirect&bpo=30319]: socket.close() now ignores ECONNRESET error.
- bpo-30828 [https://bugs.python.org/issue?@action=redirect&bpo=30828]: Fix out of bounds write in `asyncio.CFuture.remove_done_callback()`.
- bpo-30807 [https://bugs.python.org/issue?@action=redirect&bpo=30807]: signal.setitimer() may disable the timer when passed a tiny value. Tiny values (such as 1e-6) are valid non-zero values for setitimer(), which is specified as taking microsecond-resolution intervals. However, on some platform, our conversion routine could convert 1e-6 into a zero interval, therefore disabling the timer instead of (re-)scheduling it.
- bpo-30441 [https://bugs.python.org/issue?@action=redirect&bpo=30441]: Fix bug when modifying os.environ while iterating over it
- bpo-30532 [https://bugs.python.org/issue?@action=redirect&bpo=30532]: Fix email header value parser dropping folding white space in certain cases.
- bpo-30879 [https://bugs.python.org/issue?@action=redirect&bpo=30879]: os.listdir() and os.scandir() now emit bytes names when called with bytes-like argument.
- bpo-30746 [https://bugs.python.org/issue?@action=redirect&bpo=30746]: Prohibited the “=” character in

environment variable names in `os.putenv()` and `os.spawn*()`.
- bpo-29755 [https://bugs.python.org/issue?@action=redirect&bpo=29755]: Fixed the lgettext() family of functions in the gettext module. They now always return bytes.

Documentation

- bpo-31294 [https://bugs.python.org/issue?@action=redirect&bpo=31294]: Fix incomplete code snippet in the ZeroMQSocketListener and ZeroMQSocketHandler examples and adapt them to Python 3.
- bpo-21649 [https://bugs.python.org/issue?@action=redirect&bpo=21649]: Add RFC 7525 and Mozilla server side TLS links to SSL documentation.
- bpo-30803 [https://bugs.python.org/issue?@action=redirect&bpo=30803]: Clarify doc on truth value testing. Original patch by Peter Thomassen.

Tests

- bpo-31320 [https://bugs.python.org/issue?@action=redirect&bpo=31320]: Silence traceback in test_ssl
- bpo-25674 [https://bugs.python.org/issue?@action=redirect&bpo=25674]: Remove sha256.tbs-internet.com ssl test
- bpo-30715 [https://bugs.python.org/issue?@action=redirect&bpo=30715]: Address ALPN callback changes for OpenSSL 1.1.0f. The latest version behaves like OpenSSL 1.0.2 and no longer aborts handshake.
- bpo-30822 [https://bugs.python.org/issue?@action=redirect&bpo=30822]: regrtest: Exclude tzdata from regrtest –all. When running the test suite using –use=all / -u all, exclude tzdata since it makes test_datetime too slow (15-20 min on some buildbots) which then times out on some buildbots. Fix also regrtest command line parser to allow passing -u extralargefile to run test_zipfile64.

Build

- bpo-30854 [https://bugs.python.org/issue?@action=redirect&bpo=30854]: Fix compile error when compiling –without-threads. Patch by Masayuki Yamamoto.

Windows

- bpo-30389 [https://bugs.python.org/issue?@action=redirect&bpo=30389]: Adds detection of Visual Studio 2017 to distutils on Windows.
- bpo-31340 [https://bugs.python.org/issue?@action=redirect&bpo=31340]: Change to building with MSVC v141 (included with Visual Studio 2017)
- bpo-30581 [https://bugs.python.org/issue?@action=redirect&bpo=30581]: os.cpu_count() now returns the correct number of processors on Windows when the number of logical processors is greater than 64.
- bpo-30731 [https://bugs.python.org/issue?@action=redirect&bpo=30731]: Add a missing xmlns to python.manifest so that it matches the schema.

IDLE

- bpo-31493 [https://bugs.python.org/issue?@action=redirect&bpo=31493]: IDLE code context – fix code update and font update timers. Canceling timers prevents a warning message when test_idle completes.
- bpo-31488 [https://bugs.python.org/issue?@action=redirect&bpo=31488]: IDLE - Update non-key options in former extension classes. When applying configdialog changes, call .reload for each feature class. Change ParenMatch so updated options affect existing instances attached to existing editor windows.
- bpo-31477 [https://bugs.python.org/issue?@action=redirect&bpo=31477]: IDLE - Improve rstrip entry in doc. Strip trailing whitespace strips more than blank spaces. Multiline string literals are not skipped.
- bpo-31480 [https://bugs.python.org/issue?@action=redirect&bpo=31480]: IDLE - make tests pass with zzdummy extension disabled by default.
- bpo-31421 [https://bugs.python.org/issue?

@action=redirect&bpo=31421]: Document how IDLE runs tkinter programs. IDLE calls tcl/tk update in the background in order to make live interaction and experimentation with tkinter applications much easier.

- bpo-31414 [https://bugs.python.org/issue?@action=redirect&bpo=31414]: IDLE – fix tk entry box tests by deleting first. Adding to an int entry is not the same as deleting and inserting because int(“”) will fail.
- bpo-31051 [https://bugs.python.org/issue?@action=redirect&bpo=31051]: Rearrange IDLE configdialog GenPage into Window, Editor, and Help sections.
- bpo-30617 [https://bugs.python.org/issue?@action=redirect&bpo=30617]: IDLE - Add docstrings and tests for outwin subclass of editor. Move some data and functions from the class to module level. Patch by Cheryl Sabella.
- bpo-31287 [https://bugs.python.org/issue?@action=redirect&bpo=31287]: IDLE - Do not modify tkinter.message in test_configdialog.
- bpo-27099 [https://bugs.python.org/issue?@action=redirect&bpo=27099]: Convert IDLE’s built-in “extensions” to regular features. About 10 IDLE features were implemented as supposedly optional extensions. Their different behavior could be confusing or worse for users and not good for maintenance. Hence the conversion. The main difference for users is that user configurable key bindings for builtin features are now handled uniformly. Now, editing a binding in a keyset only affects its value in the keyset. All bindings are defined together in the system-specific default keysets in config-extensions.def. All custom keysets are saved as a whole in config-extension.cfg. All take effect as soon as one clicks Apply or Ok. The affected events are “<<force-open-completions>>”, “<<expand-word>>”, “<<force-open-calltip>>”, “<<flash-paren>>”, “<<format-paragraph>>”, “<<run-module>>”, “<<check-module>>”, and “<<zoom-height>>”. Any (global) customizations made before 3.6.3 will not affect their keyset-specific customization after 3.6.3. and vice versa. Initial patch by Charles Wohlganger.
- bpo-31206 [https://bugs.python.org/issue?@action=redirect&bpo=31206]: IDLE: Factor HighPage(Frame)

class from ConfigDialog. Patch by Cheryl Sabella.

- bpo-31001 [https://bugs.python.org/issue?@action=redirect&bpo=31001]: Add tests for configdialog highlight tab. Patch by Cheryl Sabella.
- bpo-31205 [https://bugs.python.org/issue?@action=redirect&bpo=31205]: IDLE: Factor KeysPage(Frame) class from ConfigDialog. The slightly modified tests continue to pass. Patch by Cheryl Sabella.
- bpo-31130 [https://bugs.python.org/issue?@action=redirect&bpo=31130]: IDLE – stop leaks in test_configdialog. Initial patch by Victor Stinner.
- bpo-31002 [https://bugs.python.org/issue?@action=redirect&bpo=31002]: Add tests for configdialog keys tab. Patch by Cheryl Sabella.
- bpo-19903 [https://bugs.python.org/issue?@action=redirect&bpo=19903]: IDLE: Calltips use `inspect.signature` instead of `inspect.getfullargspec`. This improves calltips for builtins converted to use Argument Clinic. Patch by Louie Lu.
- bpo-31083 [https://bugs.python.org/issue?@action=redirect&bpo=31083]: IDLE - Add an outline of a TabPage class in configdialog. Update existing classes to match outline. Initial patch by Cheryl Sabella.
- bpo-31050 [https://bugs.python.org/issue?@action=redirect&bpo=31050]: Factor GenPage(Frame) class from ConfigDialog. The slightly modified tests continue to pass. Patch by Cheryl Sabella.
- bpo-31004 [https://bugs.python.org/issue?@action=redirect&bpo=31004]: IDLE - Factor FontPage(Frame) class from ConfigDialog. Slightly modified tests continue to pass. Fix General tests. Patch mostly by Cheryl Sabella.
- bpo-30781 [https://bugs.python.org/issue?@action=redirect&bpo=30781]: IDLE - Use ttk widgets in ConfigDialog. Patches by Terry Jan Reedy and Cheryl Sabella.
- bpo-31060 [https://bugs.python.org/issue?@action=redirect&bpo=31060]: IDLE - Finish rearranging methods of ConfigDialog Grouping methods pertaining to each tab and the buttons will aid writing tests and improving the tabs and will enable splitting the groups into classes.
- bpo-30853 [https://bugs.python.org/issue?

@action=redirect&bpo=30853]: IDLE – Factor a VarTrace class out of ConfigDialog. Instance tracers manages pairs consisting of a tk variable and a callback function. When tracing is turned on, setting the variable calls the function. Test coverage for the new class is 100%.

- bpo-31003 [https://bugs.python.org/issue?@action=redirect&bpo=31003]: IDLE: Add more tests for General tab.
- bpo-30993 [https://bugs.python.org/issue?@action=redirect&bpo=30993]: IDLE - Improve configdialog font page and tests. In configdialog: Document causal pathways in create_font_tab docstring. Simplify some attribute names. Move set_samples calls to var_changed_font (idea from Cheryl Sabella). Move related functions to positions after the create widgets function. In test_configdialog: Fix test_font_set so not order dependent. Fix renamed test_indent_scale so it tests the widget. Adjust tests for movement of set_samples call. Add tests for load functions. Put all font tests in one class and tab indent tests in another. Except for two lines, these tests completely cover the related functions.
- bpo-30981 [https://bugs.python.org/issue?@action=redirect&bpo=30981]: IDLE – Add more configdialog font page tests.
- bpo-28523 [https://bugs.python.org/issue?@action=redirect&bpo=28523]: IDLE: replace “colour” with “color” in configdialog.
- bpo-30917 [https://bugs.python.org/issue?@action=redirect&bpo=30917]: Add tests for idlelib.config.IdleConf. Increase coverage from 46% to 96%. Patch by Louie Lu.
- bpo-30934 [https://bugs.python.org/issue?@action=redirect&bpo=30934]: Document coverage details for idlelib tests. Add section to idlelib/idle-test/README.txt. Include check that branches are taken both ways. Exclude IDLE-specific code that does not run during unit tests.
- bpo-30913 [https://bugs.python.org/issue?@action=redirect&bpo=30913]: IDLE: Document ConfigDialog tk Vars, methods, and widgets in docstrings This will facilitate improving the dialog and splitting up the class. Original patch by Cheryl Sabella.

- bpo-30899 [https://bugs.python.org/issue?@action=redirect&bpo=30899]: IDLE: Add tests for ConfigParser subclasses in config. Patch by Louie Lu.
- bpo-30881 [https://bugs.python.org/issue?@action=redirect&bpo=30881]: IDLE: Add docstrings to browser.py. Patch by Cheryl Sabella.
- bpo-30851 [https://bugs.python.org/issue?@action=redirect&bpo=30851]: IDLE: Remove unused variables in configdialog. One is a duplicate, one is set but cannot be altered by users. Patch by Cheryl Sabella.
- bpo-30870 [https://bugs.python.org/issue?@action=redirect&bpo=30870]: IDLE: In Settings dialog, select font with Up, Down keys as well as mouse. Initial patch by Louie Lu.
- bpo-8231 [https://bugs.python.org/issue?@action=redirect&bpo=8231]: IDLE: call config.IdleConf.GetUserCfgDir only once.
- bpo-30779 [https://bugs.python.org/issue?@action=redirect&bpo=30779]: IDLE: Factor ConfigChanges class from configdialog, put in config; test. * In config, put dump test code in a function; run it and unittest in “if _name_ == “_main_”. * Add class config.ConfigChanges based on changes_class_v4.py on bpo issue. * Add class test_config.ChangesTest, partly using configdialog_tests_v1.py. * Revise configdialog to use ConfigChanges; see tracker msg297804. * Revise test_configdialog to match configdialog changes. * Remove configdialog functions unused or moved to ConfigChanges. Cheryl Sabella contributed parts of the patch.
- bpo-30777 [https://bugs.python.org/issue?@action=redirect&bpo=30777]: IDLE: configdialog - Add docstrings and fix comments. Patch by Cheryl Sabella.
- bpo-30495 [https://bugs.python.org/issue?@action=redirect&bpo=30495]: IDLE: Improve textview with docstrings, PEP8 names, and more tests. Patch by Cheryl Sabella.
- bpo-30723 [https://bugs.python.org/issue?@action=redirect&bpo=30723]: IDLE: Make several improvements to parenmatch. Add “parens” style to highlight both opener and closer. Make “default” style, which is not default, a synonym for “opener”. Make time-delay work the same with

all styles. Add help for config dialog extensions tab, including help for parenmatch. Add new tests. Original patch by Charles Wohlganger.

- bpo-30674 [https://bugs.python.org/issue?@action=redirect&bpo=30674]: IDLE: add docstrings to grep module. Patch by Cheryl Sabella
- bpo-21519 [https://bugs.python.org/issue?@action=redirect&bpo=21519]: IDLE's basic custom key entry dialog now detects duplicates properly. Original patch by Saimadhav Heblikar.
- bpo-29910 [https://bugs.python.org/issue?@action=redirect&bpo=29910]: IDLE no longer deletes a character after commenting out a region by a key shortcut. Add `return 'break'` for this and other potential conflicts between IDLE and default key bindings.
- bpo-30728 [https://bugs.python.org/issue?@action=redirect&bpo=30728]: Review and change idlelib.configdialog names. Lowercase method and attribute names. Replace "colour" with "color", expand overly cryptic names, delete unneeded underscores. Replace `import *` with specific imports. Patches by Cheryl Sabella.
- bpo-6739 [https://bugs.python.org/issue?@action=redirect&bpo=6739]: IDLE: Verify user-entered key sequences by trying to bind them with tk. Add tests for all 3 validation functions. Original patch by G Polo. Tests added by Cheryl Sabella.

Tools/Demos

- bpo-30983 [https://bugs.python.org/issue?@action=redirect&bpo=30983]: gdb integration commands (py-bt, etc.) work on optimized shared builds now, too. **PEP 523** [https://peps.python.org/pep-0523/] introduced _PyEval_EvalFrameDefault which inlines PyEval_EvalFrameEx on non-debug shared builds. This broke the ability to use py-bt, py-up, and a few other Python-specific gdb integrations. The problem is fixed by only looking for _PyEval_EvalFrameDefault frames in python-gdb.py. Original patch by Bruno «Polaco» Penteado.

Python 3.6.2 final

Release date: 2017-07-17

No changes since release candidate 2

Python 3.6.2 release candidate 2

Release date: 2017-07-07

Security

- bpo-30730 [https://bugs.python.org/issue?@action=redirect&bpo=30730]: Prevent environment variables injection in subprocess on Windows. Prevent passing other environment variables and command arguments.
- bpo-30694 [https://bugs.python.org/issue?@action=redirect&bpo=30694]: Upgrade expat copy from 2.2.0 to 2.2.1 to get fixes of multiple security vulnerabilities including: **CVE 2017-9233** [https://www.cve.org/CVERecord?id=CVE-2017-9233] (External entity infinite loop DoS), **CVE 2016-9063** [https://www.cve.org/CVERecord?id=CVE-2016-9063] (Integer overflow, re-fix), **CVE 2016-0718** [https://www.cve.org/CVERecord?id=CVE-2016-0718] (Fix regression bugs from 2.2.0's fix to **CVE 2016-0718** [https://www.cve.org/CVERecord?id=CVE-2016-0718]) and **CVE 2012-0876** [https://www.cve.org/CVERecord?id=CVE-2012-0876] (Counter hash flooding with SipHash). Note: the **CVE 2016-5300** [https://www.cve.org/CVERecord?id=CVE-2016-5300] (Use os-specific entropy sources like getrandom) doesn't impact Python, since Python already gets entropy from the OS to set the expat secret using `XML_SetHashSalt()`.
- bpo-30500 [https://bugs.python.org/issue?@action=redirect&bpo=30500]: Fix urllib.parse.splithost() to correctly parse fragments. For example, `splithost('//127.0.0.1#@evil.com/')` now correctly returns the `127.0.0.1` host, instead of treating `@evil.com` as the host in an authentication (`login@host`).

Python 3.6.2 release candidate 1

Release date: 2017-06-17

Security

- bpo-29591 [https://bugs.python.org/issue?@action=redirect&bpo=29591]: Update expat copy from 2.1.1 to 2.2.0 to get fixes of **CVE 2016-0718** [https://www.cve.org/CVERecord?id=CVE-2016-0718] and **CVE 2016-4472** [https://www.cve.org/CVERecord?id=CVE-2016-4472]. See https://sourceforge.net/p/expat/bugs/537/ for more information.

Core and Builtins

- bpo-30682 [https://bugs.python.org/issue?@action=redirect&bpo=30682]: Removed a too-strict assertion that failed for certain f-strings, such as eval(«f”\n”») and eval(«f”\r”»).
- bpo-30604 [https://bugs.python.org/issue?@action=redirect&bpo=30604]: Move co_extra_freefuncs to not be per-thread to avoid crashes
- bpo-29104 [https://bugs.python.org/issue?@action=redirect&bpo=29104]: Fixed parsing backslashes in f-strings.
- bpo-27945 [https://bugs.python.org/issue?@action=redirect&bpo=27945]: Fixed various segfaults with dict when input collections are mutated during searching, inserting or comparing. Based on patches by Duane Griffin and Tim Mitchell.
- bpo-25794 [https://bugs.python.org/issue?@action=redirect&bpo=25794]: Fixed type.__setattr__() and type.__delattr__() for non-interned attribute names. Based on patch by Eryk Sun.
- bpo-30039 [https://bugs.python.org/issue?@action=redirect&bpo=30039]: If a KeyboardInterrupt happens when the interpreter is in the middle of resuming a chain of nested “yield from” or “await” calls, it’s now correctly delivered to the innermost frame.

- bpo-12414 [https://bugs.python.org/issue?@action=redirect&bpo=12414]: sys.getsizeof() on a code object now returns the sizes which includes the code struct and sizes of objects which it references. Patch by Donghee Na.
- bpo-29949 [https://bugs.python.org/issue?@action=redirect&bpo=29949]: Fix memory usage regression of set and frozenset object.
- bpo-29935 [https://bugs.python.org/issue?@action=redirect&bpo=29935]: Fixed error messages in the index() method of tuple, list and deque when pass indices of wrong type.
- bpo-29859 [https://bugs.python.org/issue?@action=redirect&bpo=29859]: Show correct error messages when any of the pthread_* calls in thread_pthread.h fails.
- bpo-28876 [https://bugs.python.org/issue?@action=redirect&bpo=28876]: `bool(range)` works even if `len(range)` raises **`OverflowError`**.
- bpo-29600 [https://bugs.python.org/issue?@action=redirect&bpo=29600]: Fix wrapping coroutine return values in StopIteration.
- bpo-28856 [https://bugs.python.org/issue?@action=redirect&bpo=28856]: Fix an oversight that %b format for bytes should support objects follow the buffer protocol.
- bpo-29714 [https://bugs.python.org/issue?@action=redirect&bpo=29714]: Fix a regression that bytes format may fail when containing zero bytes inside.
- bpo-29478 [https://bugs.python.org/issue?@action=redirect&bpo=29478]: If max_line_length=None is specified while using the Compat32 policy, it is no longer ignored. Patch by Mircea Cosbuc.

Library

- bpo-30616 [https://bugs.python.org/issue?@action=redirect&bpo=30616]: Functional API of enum allows to create empty enums. Patched by Donghee Na
- bpo-30038 [https://bugs.python.org/issue?@action=redirect&bpo=30038]: Fix race condition between signal delivery and wakeup file descriptor. Patch by Nathaniel Smith.

- bpo-23894 [https://bugs.python.org/issue?@action=redirect&bpo=23894]: lib2to3 now recognizes `rb'...'` and `f'...'` strings.
- bpo-23890 [https://bugs.python.org/issue?@action=redirect&bpo=23890]: unittest.TestCase.assertRaises() now manually breaks a reference cycle to not keep objects alive longer than expected.
- bpo-30149 [https://bugs.python.org/issue?@action=redirect&bpo=30149]: inspect.signature() now supports callables with variable-argument parameters wrapped with partialmethod. Patch by Donghee Na.
- bpo-30645 [https://bugs.python.org/issue?@action=redirect&bpo=30645]: Fix path calculation in imp.load_package(), fixing it for cases when a package is only shipped with bytecodes. Patch by Alexandru Ardelean.
- bpo-29931 [https://bugs.python.org/issue?@action=redirect&bpo=29931]: Fixed comparison check for ipaddress.ip_interface objects. Patch by Sanjay Sundaresan.
- bpo-30605 [https://bugs.python.org/issue?@action=redirect&bpo=30605]: re.compile() no longer raises a BytesWarning when compiling a bytes instance with misplaced inline modifier. Patch by Roy Williams.
- bpo-24484 [https://bugs.python.org/issue?@action=redirect&bpo=24484]: Avoid race condition in multiprocessing cleanup (#2159)
- bpo-28994 [https://bugs.python.org/issue?@action=redirect&bpo=28994]: The traceback no longer displayed for SystemExit raised in a callback registered by atexit.
- bpo-30508 [https://bugs.python.org/issue?@action=redirect&bpo=30508]: Don’t log exceptions if Task/Future «cancel()» method was called.
- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Updates to typing module: Add generic AsyncContextManager, add support for ContextManager on all versions. Original PRs by Jelle Zijlstra and Ivan Levkivskyi
- bpo-29870 [https://bugs.python.org/issue?@action=redirect&bpo=29870]: Fix ssl sockets leaks when connection is aborted in asyncio/ssl implementation. Patch by Michaël Sghaïer.

- bpo-29743 [https://bugs.python.org/issue?@action=redirect&bpo=29743]: Closing transport during handshake process leaks open socket. Patch by Nikolay Kim
- bpo-27585 [https://bugs.python.org/issue?@action=redirect&bpo=27585]: Fix waiter cancellation in asyncio.Lock. Patch by Mathieu Sornay.
- bpo-30418 [https://bugs.python.org/issue?@action=redirect&bpo=30418]: On Windows, subprocess.Popen.communicate() now also ignore EINVAL on stdin.write() if the child process is still running but closed the pipe.
- bpo-29822 [https://bugs.python.org/issue?@action=redirect&bpo=29822]: inspect.isabstract() now works during __init_subclass__. Patch by Nate Soares.
- bpo-29581 [https://bugs.python.org/issue?@action=redirect&bpo=29581]: ABCMeta.__new__ now accepts `**kwargs`, allowing abstract base classes to use keyword parameters in __init_subclass__. Patch by Nate Soares.
- bpo-30557 [https://bugs.python.org/issue?@action=redirect&bpo=30557]: faulthandler now correctly filters and displays exception codes on Windows
- bpo-30378 [https://bugs.python.org/issue?@action=redirect&bpo=30378]: Fix the problem that logging.handlers.SysLogHandler cannot handle IPv6 addresses.
- bpo-29960 [https://bugs.python.org/issue?@action=redirect&bpo=29960]: Preserve generator state when _random.Random.setstate() raises an exception. Patch by Bryan Olson.
- bpo-30414 [https://bugs.python.org/issue?@action=redirect&bpo=30414]: multiprocessing.Queue._feed background running thread do not break from main loop on exception.
- bpo-30003 [https://bugs.python.org/issue?@action=redirect&bpo=30003]: Fix handling escape characters in HZ codec. Based on patch by Ma Lin.
- bpo-30301 [https://bugs.python.org/issue?@action=redirect&bpo=30301]: Fix AttributeError when using SimpleQueue.empty() under *spawn* and *forkserver* start methods.

- bpo-30329 [https://bugs.python.org/issue?@action=redirect&bpo=30329]: imaplib and poplib now catch the Windows socket WSAEINVAL error (code 10022) on shutdown(SHUT_RDWR): An invalid operation was attempted. This error occurs sometimes on SSL connections.
- bpo-30375 [https://bugs.python.org/issue?@action=redirect&bpo=30375]: Warnings emitted when compile a regular expression now always point to the line in the user code. Previously they could point into inners of the re module if emitted from inside of groups or conditionals.
- bpo-30048 [https://bugs.python.org/issue?@action=redirect&bpo=30048]: Fixed `Task.cancel()` can be ignored when the task is running coroutine and the coroutine returned without any more `await`.
- bpo-30266 [https://bugs.python.org/issue?@action=redirect&bpo=30266]: contextlib.AbstractContextManager now supports anti-registration by setting _enter_ = None or _exit_ = None, following the pattern introduced in bpo-25958 [https://bugs.python.org/issue?@action=redirect&bpo=25958]. Patch by Jelle Zijlstra.
- bpo-30298 [https://bugs.python.org/issue?@action=redirect&bpo=30298]: Weaken the condition of deprecation warnings for inline modifiers. Now allowed several subsequential inline modifiers at the start of the pattern (e.g. `'(?i)(?s)...'`). In verbose mode whitespaces and comments now are allowed before and between inline modifiers (e.g. `'(?x) (?i) (?s)...'`).
- bpo-29990 [https://bugs.python.org/issue?@action=redirect&bpo=29990]: Fix range checking in GB18030 decoder. Original patch by Ma Lin.
- bpo-26293 [https://bugs.python.org/issue?@action=redirect&bpo=26293]: Change resulted because of zipfile breakage. (See also: bpo-29094 [https://bugs.python.org/issue?@action=redirect&bpo=29094])
- bpo-30243 [https://bugs.python.org/issue?@action=redirect&bpo=30243]: Removed the _init_ methods of _json's scanner and encoder. Misusing them could cause memory leaks or crashes. Now scanner and encoder objects are completely initialized in the _new_ methods.

- bpo-30185 [https://bugs.python.org/issue?@action=redirect&bpo=30185]: Avoid KeyboardInterrupt tracebacks in forkserver helper process when Ctrl-C is received.
- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Various updates to typing module: add typing.NoReturn type, use WrapperDescriptorType, minor bug-fixes. Original PRs by Jim Fasarakis-Hilliard and Ivan Levkivskyi.
- bpo-30205 [https://bugs.python.org/issue?@action=redirect&bpo=30205]: Fix getsockname() for unbound AF_UNIX sockets on Linux.
- bpo-30070 [https://bugs.python.org/issue?@action=redirect&bpo=30070]: Fixed leaks and crashes in errors handling in the parser module.
- bpo-30061 [https://bugs.python.org/issue?@action=redirect&bpo=30061]: Fixed crashes in IOBase methods _next_() and readlines() when readline() or _next_() respectively return non-sizeable object. Fixed possible other errors caused by not checking results of PyObject_Size(), PySequence_Size(), or PyMapping_Size().
- bpo-30017 [https://bugs.python.org/issue?@action=redirect&bpo=30017]: Allowed calling the close() method of the zip entry writer object multiple times. Writing to a closed writer now always produces a ValueError.
- bpo-30068 [https://bugs.python.org/issue?@action=redirect&bpo=30068]: _io._IOBase.readlines will check if it’s closed first when hint is present.
- bpo-29694 [https://bugs.python.org/issue?@action=redirect&bpo=29694]: Fixed race condition in pathlib mkdir with flags parents=True. Patch by Armin Rigo.
- bpo-29692 [https://bugs.python.org/issue?@action=redirect&bpo=29692]: Fixed arbitrary unchaining of RuntimeError exceptions in contextlib.contextmanager. Patch by Siddharth Velankar.
- bpo-29998 [https://bugs.python.org/issue?@action=redirect&bpo=29998]: Pickling and copying ImportError now preserves name and path attributes.
- bpo-29953 [https://bugs.python.org/issue?@action=redirect&bpo=29953]: Fixed memory leaks in the

replace() method of datetime and time objects when pass out of bound fold argument.

- bpo-29942 [https://bugs.python.org/issue?@action=redirect&bpo=29942]: Fix a crash in itertools.chain.from_iterable when encountering long runs of empty iterables.
- bpo-27863 [https://bugs.python.org/issue?@action=redirect&bpo=27863]: Fixed multiple crashes in ElementTree caused by race conditions and wrong types.
- bpo-28699 [https://bugs.python.org/issue?@action=redirect&bpo=28699]: Fixed a bug in pools in multiprocessing.pool that raising an exception at the very first of an iterable may swallow the exception or make the program hang. Patch by Davin Potts and Xiang Zhang.
- bpo-25803 [https://bugs.python.org/issue?@action=redirect&bpo=25803]: Avoid incorrect errors raised by Path.mkdir(exist_ok=True) when the OS gives priority to errors such as EACCES over EEXIST.
- bpo-29861 [https://bugs.python.org/issue?@action=redirect&bpo=29861]: Release references to tasks, their arguments and their results as soon as they are finished in multiprocessing.Pool.
- bpo-29884 [https://bugs.python.org/issue?@action=redirect&bpo=29884]: faulthandler: Restore the old sigaltstack during teardown. Patch by Christophe Zeitouny.
- bpo-25455 [https://bugs.python.org/issue?@action=redirect&bpo=25455]: Fixed crashes in repr of recursive buffered file-like objects.
- bpo-29800 [https://bugs.python.org/issue?@action=redirect&bpo=29800]: Fix crashes in partial.__repr__ if the keys of partial.keywords are not strings. Patch by Michael Seifert.
- bpo-29742 [https://bugs.python.org/issue?@action=redirect&bpo=29742]: get_extra_info() raises exception if get called on closed ssl transport. Patch by Nikolay Kim.
- bpo-8256 [https://bugs.python.org/issue?@action=redirect&bpo=8256]: Fixed possible failing or crashing input() if attributes «encoding» or «errors» of sys.stdin or sys.stdout are not set or are not strings.
- bpo-28298 [https://bugs.python.org/issue?

@action=redirect&bpo=28298]: Fix a bug that prevented array “Q”, “L” and “I” from accepting big intables (objects that have _int_) as elements. Patch by Oren Milman.

- bpo-28231 [https://bugs.python.org/issue?@action=redirect&bpo=28231]: The zipfile module now accepts path-like objects for external paths.
- bpo-26915 [https://bugs.python.org/issue?@action=redirect&bpo=26915]: index() and count() methods of collections.abc.Sequence now check identity before checking equality when do comparisons.
- bpo-29615 [https://bugs.python.org/issue?@action=redirect&bpo=29615]: SimpleXMLRPCDispatcher no longer chains KeyError (or any other exception) to exception(s) raised in the dispatched methods. Patch by Petr Motejlek.
- bpo-30177 [https://bugs.python.org/issue?@action=redirect&bpo=30177]: path.resolve(strict=False) no longer cuts the path after the first element not present in the filesystem. Patch by Antoine Pietri.

IDLE

- bpo-15786 [https://bugs.python.org/issue?@action=redirect&bpo=15786]: Fix several problems with IDLE’s autocompletion box. The following should now work: clicking on selection box items; using the scrollbar; selecting an item by hitting Return. Hangs on MacOSX should no longer happen. Patch by Louie Lu.
- bpo-25514 [https://bugs.python.org/issue?@action=redirect&bpo=25514]: Add doc subsubsection about IDLE failure to start. Popup no-connection message directs users to this section.
- bpo-30642 [https://bugs.python.org/issue?@action=redirect&bpo=30642]: Fix reference leaks in IDLE tests. Patches by Louie Lu and Terry Jan Reedy.
- bpo-30495 [https://bugs.python.org/issue?@action=redirect&bpo=30495]: Add docstrings for textview.py and use PEP8 names. Patches by Cheryl Sabella and Terry Jan Reedy.
- bpo-30290 [https://bugs.python.org/issue?

@action=redirect&bpo=30290]: Help-about: use pep8 names and add tests. Increase coverage to 100%. Patches by Louie Lu, Cheryl Sabella, and Terry Jan Reedy.
- bpo-30303 [https://bugs.python.org/issue?@action=redirect&bpo=30303]: Add _utest option to textview; add new tests. Increase coverage to 100%. Patches by Louie Lu and Terry Jan Reedy.

C API

- bpo-27867 [https://bugs.python.org/issue?@action=redirect&bpo=27867]: Function PySlice_GetIndicesEx() no longer replaced with a macro if Py_LIMITED_API is not set.

Build

- bpo-29941 [https://bugs.python.org/issue?@action=redirect&bpo=29941]: Add `--with-assertions` configure flag to explicitly enable C `assert()` checks. Defaults to off. `--with-pydebug` implies `--with-assertions`.
- bpo-28787 [https://bugs.python.org/issue?@action=redirect&bpo=28787]: Fix out-of-tree builds of Python when configured with `--with--dtrace`.
- bpo-29243 [https://bugs.python.org/issue?@action=redirect&bpo=29243]: Prevent unnecessary rebuilding of Python during `make test`, `make install` and some other make targets when configured with `--enable-optimizations`.
- bpo-23404 [https://bugs.python.org/issue?@action=redirect&bpo=23404]: Don’t regenerate generated files based on file modification time anymore: the action is now explicit. Replace `make touch` with `make regen-all`.
- bpo-29643 [https://bugs.python.org/issue?@action=redirect&bpo=29643]: Fix `--enable-optimization` didn’t work.

Documentation

- bpo-30176 [https://bugs.python.org/issue?

@action=redirect&bpo=30176]: Add missing attribute related constants in curses documentation.

- bpo-30052 [https://bugs.python.org/issue?@action=redirect&bpo=30052]: the link targets for **`bytes()`** and **`bytearray()`** are now their respective type definitions, rather than the corresponding builtin function entries. Use bytes and bytearray to reference the latter. In order to ensure this and future cross-reference updates are applied automatically, the daily documentation builds now disable the default output caching features in Sphinx.
- bpo-26985 [https://bugs.python.org/issue?@action=redirect&bpo=26985]: Add missing info of code object in inspect documentation.

Tools/Demos

- bpo-29367 [https://bugs.python.org/issue?@action=redirect&bpo=29367]: python-gdb.py now supports also `method-wrapper` (`wrapperobject`) objects.

Tests

- bpo-30357 [https://bugs.python.org/issue?@action=redirect&bpo=30357]: test_thread: setUp() now uses support.threading_setup() and support.threading_cleanup() to wait until threads complete to avoid random side effects on following tests. Initial patch written by Grzegorz Grzywacz.
- bpo-30197 [https://bugs.python.org/issue?@action=redirect&bpo=30197]: Enhanced functions swap_attr() and swap_item() in the test.support module. They now work when delete replaced attribute or item inside the with statement. The old value of the attribute or item (or None if it doesn’t exist) now will be assigned to the target of the «as» clause, if there is one.

Windows

- bpo-30687 [https://bugs.python.org/issue?@action=redirect&bpo=30687]: Locate msbuild.exe on Windows when building rather than vcvarsall.bat

- bpo-30450 [https://bugs.python.org/issue?@action=redirect&bpo=30450]: The build process on Windows no longer depends on Subversion, instead pulling external code from GitHub via a Python script. If Python 3.6 is not found on the system (via `py -3.6`), NuGet is used to download a copy of 32-bit Python.

Python 3.6.1 final

Release date: 2017-03-21

Core and Builtins

- bpo-29723 [https://bugs.python.org/issue?@action=redirect&bpo=29723]: The `sys.path[0]` initialization change for bpo-29139 [https://bugs.python.org/issue?@action=redirect&bpo=29139] caused a regression by revealing an inconsistency in how sys.path is initialized when executing `__main__` from a zipfile, directory, or other import location. The interpreter now consistently avoids ever adding the import location's parent directory to `sys.path`, and ensures no other `sys.path` entries are inadvertently modified when inserting the import location named on the command line.

Build

- bpo-27593 [https://bugs.python.org/issue?@action=redirect&bpo=27593]: fix format of git information used in sys.version
- Fix incompatible comment in python.h

Python 3.6.1 release candidate 1

Release date: 2017-03-04

Core and Builtins

- bpo-28893 [https://bugs.python.org/issue?

@action=redirect&bpo=28893]: Set correct _cause_ for errors about invalid awaitables returned from _aiter_ and _anext_.
- bpo-29683 [https://bugs.python.org/issue?@action=redirect&bpo=29683]: Fixes to memory allocation in _PyCode_SetExtra. Patch by Brian Coleman.
- bpo-29684 [https://bugs.python.org/issue?@action=redirect&bpo=29684]: Fix minor regression of PyEval_CallObjectWithKeywords. It should raise TypeError when kwargs is not a dict. But it might cause segv when args=NULL and kwargs is not a dict.
- bpo-28598 [https://bugs.python.org/issue?@action=redirect&bpo=28598]: Support _rmod_ for subclasses of str being called before str._mod_. Patch by Martijn Pieters.
- bpo-29607 [https://bugs.python.org/issue?@action=redirect&bpo=29607]: Fix stack_effect computation for CALL_FUNCTION_EX. Patch by Matthieu Dartiailh.
- bpo-29602 [https://bugs.python.org/issue?@action=redirect&bpo=29602]: Fix incorrect handling of signed zeros in complex constructor for complex subclasses and for inputs having a _complex_ method. Patch by Serhiy Storchaka.
- bpo-29347 [https://bugs.python.org/issue?@action=redirect&bpo=29347]: Fixed possibly dereferencing undefined pointers when creating weakref objects.
- bpo-29438 [https://bugs.python.org/issue?@action=redirect&bpo=29438]: Fixed use-after-free problem in key sharing dict.
- bpo-29319 [https://bugs.python.org/issue?@action=redirect&bpo=29319]: Prevent RunMainFromImporter overwriting sys.path[0].
- bpo-29337 [https://bugs.python.org/issue?@action=redirect&bpo=29337]: Fixed possible BytesWarning when compare the code objects. Warnings could be emitted at compile time.
- bpo-29327 [https://bugs.python.org/issue?@action=redirect&bpo=29327]: Fixed a crash when pass the iterable keyword argument to sorted().
- bpo-29034 [https://bugs.python.org/issue?@action=redirect&bpo=29034]: Fix memory leak and use-after-free in os module (path_converter).

- bpo-29159 [https://bugs.python.org/issue?@action=redirect&bpo=29159]: Fix regression in bytes(x) when x.__index__() raises Exception.
- bpo-28932 [https://bugs.python.org/issue?@action=redirect&bpo=28932]: Do not include <sys/random.h> if it does not exist.
- bpo-25677 [https://bugs.python.org/issue?@action=redirect&bpo=25677]: Correct the positioning of the syntax error caret for indented blocks. Based on patch by Michael Layzell.
- bpo-29000 [https://bugs.python.org/issue?@action=redirect&bpo=29000]: Fixed bytes formatting of octals with zero padding in alternate form.
- bpo-26919 [https://bugs.python.org/issue?@action=redirect&bpo=26919]: On Android, operating system data is now always encoded/decoded to/from UTF-8, instead of the locale encoding to avoid inconsistencies with os.fsencode() and os.fsdecode() which are already using UTF-8.
- bpo-28991 [https://bugs.python.org/issue?@action=redirect&bpo=28991]: functools.lru_cache() was susceptible to an obscure reentrancy bug triggerable by a monkey-patched len() function.
- bpo-28739 [https://bugs.python.org/issue?@action=redirect&bpo=28739]: f-string expressions are no longer accepted as docstrings and by ast.literal_eval() even if they do not include expressions.
- bpo-28512 [https://bugs.python.org/issue?@action=redirect&bpo=28512]: Fixed setting the offset attribute of SyntaxError by PyErr_SyntaxLocationEx() and PyErr_SyntaxLocationObject().
- bpo-28918 [https://bugs.python.org/issue?@action=redirect&bpo=28918]: Fix the cross compilation of xxlimited when Python has been built with Py_DEBUG defined.
- bpo-28731 [https://bugs.python.org/issue?@action=redirect&bpo=28731]: Optimize _PyDict_NewPresized() to create correct size dict. Improve speed of dict literal with constant keys up to 30%.

Library

- bpo-29169 [https://bugs.python.org/issue?@action=redirect&bpo=29169]: Update zlib to 1.2.11.
- bpo-29623 [https://bugs.python.org/issue?@action=redirect&bpo=29623]: Allow use of path-like object as a single argument in ConfigParser.read(). Patch by David Ellis.
- bpo-28963 [https://bugs.python.org/issue?@action=redirect&bpo=28963]: Fix out of bound iteration in asyncio.Future.remove_done_callback implemented in C.
- bpo-29704 [https://bugs.python.org/issue?@action=redirect&bpo=29704]: asyncio.subprocess.SubprocessStreamProtocol no longer closes before all pipes are closed.
- bpo-29271 [https://bugs.python.org/issue?@action=redirect&bpo=29271]: Fix Task.current_task and Task.all_tasks implemented in C to accept None argument as their pure Python implementation.
- bpo-29703 [https://bugs.python.org/issue?@action=redirect&bpo=29703]: Fix asyncio to support instantiation of new event loops in child processes.
- bpo-29376 [https://bugs.python.org/issue?@action=redirect&bpo=29376]: Fix assertion error in threading._DummyThread.is_alive().
- bpo-28624 [https://bugs.python.org/issue?@action=redirect&bpo=28624]: Add a test that checks that cwd parameter of Popen() accepts PathLike objects. Patch by Sayan Chowdhury.
- bpo-28518 [https://bugs.python.org/issue?@action=redirect&bpo=28518]: Start a transaction implicitly before a DML statement. Patch by Aviv Palivoda.
- bpo-29532 [https://bugs.python.org/issue?@action=redirect&bpo=29532]: Altering a kwarg dictionary passed to functools.partial() no longer affects a partial object after creation.
- bpo-29110 [https://bugs.python.org/issue?@action=redirect&bpo=29110]: Fix file object leak in aifc.open() when file is given as a filesystem path and is not in valid AIFF format. Patch by Anthony Zhang.
- bpo-28556 [https://bugs.python.org/issue?

@action=redirect&bpo=28556]: Various updates to typing module: typing.Counter, typing.ChainMap, improved ABC caching, etc. Original PRs by Jelle Zijlstra, Ivan Levkivskyi, Manuel Krebber, and Łukasz Langa.

- bpo-29100 [https://bugs.python.org/issue?@action=redirect&bpo=29100]: Fix datetime.fromtimestamp() regression introduced in Python 3.6.0: check minimum and maximum years.
- bpo-29519 [https://bugs.python.org/issue?@action=redirect&bpo=29519]: Fix weakref spewing exceptions during interpreter shutdown when used with a rare combination of multiprocessing and custom codecs.
- bpo-29416 [https://bugs.python.org/issue?@action=redirect&bpo=29416]: Prevent infinite loop in pathlib.Path.mkdir
- bpo-29444 [https://bugs.python.org/issue?@action=redirect&bpo=29444]: Fixed out-of-bounds buffer access in the group() method of the match object. Based on patch by WGH.
- bpo-29335 [https://bugs.python.org/issue?@action=redirect&bpo=29335]: Fix subprocess.Popen.wait() when the child process has exited to a stopped instead of terminated state (ex: when under ptrace).
- bpo-29290 [https://bugs.python.org/issue?@action=redirect&bpo=29290]: Fix a regression in argparse that help messages would wrap at non-breaking spaces.
- bpo-28735 [https://bugs.python.org/issue?@action=redirect&bpo=28735]: Fixed the comparison of mock.MagickMock with mock.ANY.
- bpo-29316 [https://bugs.python.org/issue?@action=redirect&bpo=29316]: Restore the provisional status of typing module, add corresponding note to documentation. Patch by Ivan L.
- bpo-29219 [https://bugs.python.org/issue?@action=redirect&bpo=29219]: Fixed infinite recursion in the repr of uninitialized ctypes.CDLL instances.
- bpo-29011 [https://bugs.python.org/issue?@action=redirect&bpo=29011]: Fix an important omission by adding Deque to the typing module.
- bpo-28969 [https://bugs.python.org/issue?

@action=redirect&bpo=28969]: Fixed race condition in C implementation of functools.lru_cache. KeyError could be raised when cached function with full cache was simultaneously called from different threads with the same uncached arguments.

- bpo-29142 [https://bugs.python.org/issue?@action=redirect&bpo=29142]: In urllib.request, suffixes in no_proxy environment variable with leading dots could match related hostnames again (e.g. .b.c matches a.b.c). Patch by Milan Oberkirch.
- bpo-28961 [https://bugs.python.org/issue?@action=redirect&bpo=28961]: Fix unittest.mock._Call helper: don’t ignore the name parameter anymore. Patch written by Jiajun Huang.
- bpo-29203 [https://bugs.python.org/issue?@action=redirect&bpo=29203]: functools.lru_cache() now respects **PEP 468** [https://peps.python.org/pep-0468/] and preserves the order of keyword arguments. f(a=1, b=2) is now cached separately from f(b=2, a=1) since both calls could potentially give different results.
- bpo-15812 [https://bugs.python.org/issue?@action=redirect&bpo=15812]: inspect.getframeinfo() now correctly shows the first line of a context. Patch by Sam Breese.
- bpo-29094 [https://bugs.python.org/issue?@action=redirect&bpo=29094]: Offsets in a ZIP file created with extern file object and modes «w» and «x» now are relative to the start of the file.
- bpo-29085 [https://bugs.python.org/issue?@action=redirect&bpo=29085]: Allow random.Random.seed() to use high quality OS randomness rather than the pid and time.
- bpo-29061 [https://bugs.python.org/issue?@action=redirect&bpo=29061]: Fixed bug in secrets.randbelow() which would hang when given a negative input. Patch by Brendan Donegan.
- bpo-29079 [https://bugs.python.org/issue?@action=redirect&bpo=29079]: Prevent infinite loop in pathlib.resolve() on Windows
- bpo-13051 [https://bugs.python.org/issue?@action=redirect&bpo=13051]: Fixed recursion errors in large or

resized curses.textpad.Textbox. Based on patch by Tycho Andersen.

- bpo-29119 [https://bugs.python.org/issue?@action=redirect&bpo=29119]: Fix weakrefs in the pure python version of collections.OrderedDict move_to_end() method. Contributed by Andra Bogildea.
- bpo-9770 [https://bugs.python.org/issue?@action=redirect&bpo=9770]: curses.ascii predicates now work correctly with negative integers.
- bpo-28427 [https://bugs.python.org/issue?@action=redirect&bpo=28427]: old keys should not remove new values from WeakValueDictionary when collecting from another thread.
- bpo-28923 [https://bugs.python.org/issue?@action=redirect&bpo=28923]: Remove editor artifacts from Tix.py.
- bpo-29055 [https://bugs.python.org/issue?@action=redirect&bpo=29055]: Neaten-up empty population error on random.choice() by suppressing the upstream exception.
- bpo-28871 [https://bugs.python.org/issue?@action=redirect&bpo=28871]: Fixed a crash when deallocate deep ElementTree.
- bpo-19542 [https://bugs.python.org/issue?@action=redirect&bpo=19542]: Fix bugs in WeakValueDictionary.setdefault() and WeakValueDictionary.pop() when a GC collection happens in another thread.
- bpo-20191 [https://bugs.python.org/issue?@action=redirect&bpo=20191]: Fixed a crash in resource.prlimit() when passing a sequence that doesn’t own its elements as limits.
- bpo-28779 [https://bugs.python.org/issue?@action=redirect&bpo=28779]: multiprocessing.set_forkserver_preload() would crash the forkserver process if a preloaded module instantiated some multiprocessing objects such as locks.
- bpo-28847 [https://bugs.python.org/issue?@action=redirect&bpo=28847]: dbm.dumb now supports reading read-only files and no longer writes the index file when it is not changed.

- bpo-26937 [https://bugs.python.org/issue?@action=redirect&bpo=26937]: The chown() method of the tarfile.TarFile class does not fail now when the grp module cannot be imported, as for example on Android platforms.

IDLE

- bpo-29071 [https://bugs.python.org/issue?@action=redirect&bpo=29071]: IDLE colors f-string prefixes (but not invalid ur prefixes).
- bpo-28572 [https://bugs.python.org/issue?@action=redirect&bpo=28572]: Add 10% to coverage of IDLE's test_configdialog. Update and augment description of the configuration system.

Windows

- bpo-29579 [https://bugs.python.org/issue?@action=redirect&bpo=29579]: Removes readme.txt from the installer
- bpo-29326 [https://bugs.python.org/issue?@action=redirect&bpo=29326]: Ignores blank lines in ._pth files (Patch by Alexey Izbyshev)
- bpo-28164 [https://bugs.python.org/issue?@action=redirect&bpo=28164]: Correctly handle special console filenames (patch by Eryk Sun)
- bpo-29409 [https://bugs.python.org/issue?@action=redirect&bpo=29409]: Implement **PEP 529** [https://peps.python.org/pep-0529/] for io.FileIO (Patch by Eryk Sun)
- bpo-29392 [https://bugs.python.org/issue?@action=redirect&bpo=29392]: Prevent crash when passing invalid arguments into msvcrt module.
- bpo-25778 [https://bugs.python.org/issue?@action=redirect&bpo=25778]: winreg does not truncate string correctly (Patch by Eryk Sun)
- bpo-28896 [https://bugs.python.org/issue?@action=redirect&bpo=28896]: Deprecate WindowsRegistryFinder and disable it by default.

C API

- bpo-27867 [https://bugs.python.org/issue?@action=redirect&bpo=27867]: Function PySlice_GetIndicesEx() is replaced with a macro if Py_LIMITED_API is not set or set to the value between 0x03050400 and 0x03060000 (not including) or 0x03060100 or higher.
- bpo-29083 [https://bugs.python.org/issue?@action=redirect&bpo=29083]: Fixed the declaration of some public API functions. PyArg_VaParse() and PyArg_VaParseTupleAndKeywords() were not available in limited API. PyArg_ValidateKeywordArguments(), PyArg_UnpackTuple() and Py_BuildValue() were not available in limited API of version < 3.3 when PY_SSIZE_T_CLEAN is defined.
- bpo-29058 [https://bugs.python.org/issue?@action=redirect&bpo=29058]: All stable API extensions added after Python 3.2 are now available only when Py_LIMITED_API is set to the PY_VERSION_HEX value of the minimum Python version supporting this API.

Documentation

- bpo-28929 [https://bugs.python.org/issue?@action=redirect&bpo=28929]: Link the documentation to its source file on GitHub.
- bpo-25008 [https://bugs.python.org/issue?@action=redirect&bpo=25008]: Document smtpd.py as effectively deprecated and add a pointer to aiosmtpd, a third-party asyncio-based replacement.
- bpo-26355 [https://bugs.python.org/issue?@action=redirect&bpo=26355]: Add canonical header link on each page to corresponding major version of the documentation. Patch by Matthias Bussonnier.
- bpo-29349 [https://bugs.python.org/issue?@action=redirect&bpo=29349]: Fix Python 2 syntax in code for building the documentation.

Tests

- bpo-28087 [https://bugs.python.org/issue?@action=redirect&bpo=28087]: Skip test_asyncore and test_eintr

poll failures on macOS. Skip some tests of select.poll when running on macOS due to unresolved issues with the underlying system poll function on some macOS versions.
- bpo-29571 [https://bugs.python.org/issue?@action=redirect&bpo=29571]: to match the behaviour of the `re.LOCALE` flag, test_re.test_locale_flag now uses `locale.getpreferredencoding(False)` to determine the candidate encoding for the test regex (allowing it to correctly skip the test when the default locale encoding is a multi-byte encoding)
- bpo-28950 [https://bugs.python.org/issue?@action=redirect&bpo=28950]: Disallow -j0 to be combined with -T/-l in regrtest command line arguments.
- bpo-28683 [https://bugs.python.org/issue?@action=redirect&bpo=28683]: Fix the tests that bind() a unix socket and raise PermissionError on Android for a non-root user.
- bpo-26939 [https://bugs.python.org/issue?@action=redirect&bpo=26939]: Add the support.setswitchinterval() function to fix test_functools hanging on the Android armv7 qemu emulator.

Build

- bpo-27593 [https://bugs.python.org/issue?@action=redirect&bpo=27593]: sys.version and the platform module python_build(), python_branch(), and python_revision() functions now use git information rather than hg when building from a repo.
- bpo-29572 [https://bugs.python.org/issue?@action=redirect&bpo=29572]: Update Windows build and OS X installers to use OpenSSL 1.0.2k.
- bpo-26851 [https://bugs.python.org/issue?@action=redirect&bpo=26851]: Set Android compilation and link flags.
- bpo-28768 [https://bugs.python.org/issue?@action=redirect&bpo=28768]: Fix implicit declaration of function _setmode. Patch by Masayuki Yamamoto
- bpo-29080 [https://bugs.python.org/issue?@action=redirect&bpo=29080]: Removes hard dependency on

hg.exe from PCBuild/build.bat
- bpo-23903 [https://bugs.python.org/issue?@action=redirect&bpo=23903]: Added missed names to PC/python3.def.
- bpo-28762 [https://bugs.python.org/issue?@action=redirect&bpo=28762]: lockf() is available on Android API level 24, but the F_LOCK macro is not defined in android-ndk-r13.
- bpo-28538 [https://bugs.python.org/issue?@action=redirect&bpo=28538]: Fix the compilation error that occurs because if_nameindex() is available on Android API level 24, but the if_nameindex structure is not defined.
- bpo-20211 [https://bugs.python.org/issue?@action=redirect&bpo=20211]: Do not add the directory for installing C header files and the directory for installing object code libraries to the cross compilation search paths. Original patch by Thomas Petazzoni.
- bpo-28849 [https://bugs.python.org/issue?@action=redirect&bpo=28849]: Do not define sys.implementation._multiarch on Android.

Python 3.6.0 final

Release date: 2016-12-23

No changes since release candidate 2

Python 3.6.0 release candidate 2

Release date: 2016-12-16

Core and Builtins

- bpo-28147 [https://bugs.python.org/issue?@action=redirect&bpo=28147]: Fix a memory leak in split-table dictionaries: setattr() must not convert combined table into split table. Patch written by INADA Naoki.
- bpo-28990 [https://bugs.python.org/issue?@action=redirect&bpo=28990]: Fix asyncio SSL hanging if

connection is closed before handshake is completed. (Patch by HoHo-Ho)

Tools/Demos

- bpo-28770 [https://bugs.python.org/issue?@action=redirect&bpo=28770]: Fix python-gdb.py for fastcalls.

Windows

- bpo-28896 [https://bugs.python.org/issue?@action=redirect&bpo=28896]: Deprecate WindowsRegistryFinder.

Build

- bpo-28898 [https://bugs.python.org/issue?@action=redirect&bpo=28898]: Prevent gdb build errors due to HAVE_LONG_LONG redefinition.

Python 3.6.0 release candidate 1

Release date: 2016-12-06

Core and Builtins

- bpo-23722 [https://bugs.python.org/issue?@action=redirect&bpo=23722]: Rather than silently producing a class that doesn't support zero-argument `super()` in methods, failing to pass the new `__classcell__` namespace entry up to `type.__new__` now results in a `DeprecationWarning` and a class that supports zero-argument `super()`.
- bpo-28797 [https://bugs.python.org/issue?@action=redirect&bpo=28797]: Modifying the class _dict_ inside the _set_name_ method of a descriptor that is used inside that class no longer prevents calling the _set_name_ method of other descriptors.
- bpo-28782 [https://bugs.python.org/issue?

@action=redirect&bpo=28782]: Fix a bug in the implementation `yield from` when checking if the next instruction is YIELD_FROM. Regression introduced by WORDCODE (bpo-26647 [https://bugs.python.org/issue?@action=redirect&bpo=26647]).

Library

- bpo-27030 [https://bugs.python.org/issue?@action=redirect&bpo=27030]: Unknown escapes in re.sub() replacement template are allowed again. But they still are deprecated and will be disabled in 3.7.
- bpo-28835 [https://bugs.python.org/issue?@action=redirect&bpo=28835]: Fix a regression introduced in warnings.catch_warnings(): call warnings.showwarning() if it was overridden inside the context manager.
- bpo-27172 [https://bugs.python.org/issue?@action=redirect&bpo=27172]: To assist with upgrades from 2.7, the previously documented deprecation of `inspect.getfullargspec()` has been reversed. This decision may be revisited again after the Python 2.7 branch is no longer officially supported.
- bpo-26273 [https://bugs.python.org/issue?@action=redirect&bpo=26273]: Add new **`socket.TCP_CONGESTION`** (Linux 2.6.13) and **`socket.TCP_USER_TIMEOUT`** (Linux 2.6.37) constants. Patch written by Omar Sandoval.
- bpo-24142 [https://bugs.python.org/issue?@action=redirect&bpo=24142]: Reading a corrupt config file left configparser in an invalid state. Original patch by Florian Höch.
- bpo-28843 [https://bugs.python.org/issue?@action=redirect&bpo=28843]: Fix asyncio C Task to handle exceptions _traceback_.

C API

- bpo-28808 [https://bugs.python.org/issue?@action=redirect&bpo=28808]: PyUnicode_CompareWithASCIIString() now never raises exceptions.

Documentation

- bpo-23722 [https://bugs.python.org/issue?@action=redirect&bpo=23722]: The data model reference and the porting section in the What's New guide now cover the additional `__classcell__` handling needed for custom metaclasses to fully support **PEP 487** [https://peps.python.org/pep-0487/] and zero-argument `super()`.

Tools/Demos

- bpo-28023 [https://bugs.python.org/issue?@action=redirect&bpo=28023]: Fix python-gdb.py didn't support new dict implementation.

Python 3.6.0 beta 4

Release date: 2016-11-21

Core and Builtins

- bpo-28532 [https://bugs.python.org/issue?@action=redirect&bpo=28532]: Show sys.version when -V option is supplied twice.
- bpo-27100 [https://bugs.python.org/issue?@action=redirect&bpo=27100]: The with-statement now checks for _enter_ before it checks for _exit_. This gives less confusing error messages when both methods are missing. Patch by Jonathan Ellington.
- bpo-28746 [https://bugs.python.org/issue?@action=redirect&bpo=28746]: Fix the set_inheritable() file descriptor method on platforms that do not have the ioctl FIOCLEX and FIONCLEX commands.
- bpo-26920 [https://bugs.python.org/issue?@action=redirect&bpo=26920]: Fix not getting the locale's charset upon initializing the interpreter, on platforms that do not have langinfo.

- bpo-28648 [https://bugs.python.org/issue?@action=redirect&bpo=28648]: Fixed crash in Py_DecodeLocale() in debug build on Mac OS X when decode astral characters. Patch by Xiang Zhang.
- bpo-19398 [https://bugs.python.org/issue?@action=redirect&bpo=19398]: Extra slash no longer added to sys.path components in case of empty compile-time PYTHONPATH components.
- bpo-28665 [https://bugs.python.org/issue?@action=redirect&bpo=28665]: Improve speed of the STORE_DEREF opcode by 40%.
- bpo-28583 [https://bugs.python.org/issue?@action=redirect&bpo=28583]: PyDict_SetDefault didn't combine split table when needed. Patch by Xiang Zhang.
- bpo-27243 [https://bugs.python.org/issue?@action=redirect&bpo=27243]: Change PendingDeprecationWarning -> DeprecationWarning. As it was agreed in the issue, _aiter_ returning an awaitable should result in PendingDeprecationWarning in 3.5 and in DeprecationWarning in 3.6.
- bpo-26182 [https://bugs.python.org/issue?@action=redirect&bpo=26182]: Fix a refleak in code that raises DeprecationWarning.
- bpo-28721 [https://bugs.python.org/issue?@action=redirect&bpo=28721]: Fix asynchronous generators aclose() and athrow() to handle StopAsyncIteration propagation properly.

Library

- bpo-28752 [https://bugs.python.org/issue?@action=redirect&bpo=28752]: Restored the _reduce_() methods of datetime objects.
- bpo-28727 [https://bugs.python.org/issue?@action=redirect&bpo=28727]: Regular expression patterns, _sre.SRE_Pattern objects created by re.compile(), become comparable (only x==y and x!=y operators). This change should fix the bpo-18383 [https://bugs.python.org/issue?@action=redirect&bpo=18383]: don't duplicate warning filters when the warnings module is reloaded (thing usually only

done in unit tests).

- bpo-20572 [https://bugs.python.org/issue?@action=redirect&bpo=20572]: The subprocess.Popen.wait method's undocumented endtime parameter now raises a DeprecationWarning.
- bpo-25659 [https://bugs.python.org/issue?@action=redirect&bpo=25659]: In ctypes, prevent a crash calling the from_buffer() and from_buffer_copy() methods on abstract classes like Array.
- bpo-19717 [https://bugs.python.org/issue?@action=redirect&bpo=19717]: Makes Path.resolve() succeed on paths that do not exist. Patch by Vajrasky Kok
- bpo-28563 [https://bugs.python.org/issue?@action=redirect&bpo=28563]: Fixed possible DoS and arbitrary code execution when handle plural form selections in the gettext module. The expression parser now supports exact syntax supported by GNU gettext.
- bpo-28387 [https://bugs.python.org/issue?@action=redirect&bpo=28387]: Fixed possible crash in _io.TextIOWrapper deallocator when the garbage collector is invoked in other thread. Based on patch by Sebastian Cufre.
- bpo-28600 [https://bugs.python.org/issue?@action=redirect&bpo=28600]: Optimize loop.call_soon.
- bpo-28613 [https://bugs.python.org/issue?@action=redirect&bpo=28613]: Fix get_event_loop() return the current loop if called from coroutines/callbacks.
- bpo-28634 [https://bugs.python.org/issue?@action=redirect&bpo=28634]: Fix asyncio.isfuture() to support unittest.Mock.
- bpo-26081 [https://bugs.python.org/issue?@action=redirect&bpo=26081]: Fix refleak in _asyncio.Future.__iter__().throw.
- bpo-28639 [https://bugs.python.org/issue?@action=redirect&bpo=28639]: Fix inspect.isawaitable to always return bool Patch by Justin Mayfield.
- bpo-28652 [https://bugs.python.org/issue?@action=redirect&bpo=28652]: Make loop methods reject socket kinds they do not support.
- bpo-28653 [https://bugs.python.org/issue?@action=redirect&bpo=28653]: Fix a refleak in

functools.lru_cache.

- bpo-28703 [https://bugs.python.org/issue?@action=redirect&bpo=28703]: Fix asyncio.iscoroutinefunction to handle Mock objects.
- bpo-28704 [https://bugs.python.org/issue?@action=redirect&bpo=28704]: Fix create_unix_server to support Path-like objects (PEP 519).
- bpo-28720 [https://bugs.python.org/issue?@action=redirect&bpo=28720]: Add collections.abc.AsyncGenerator.

Documentation

- bpo-28513 [https://bugs.python.org/issue?@action=redirect&bpo=28513]: Documented command-line interface of zipfile.

Tests

- bpo-28666 [https://bugs.python.org/issue?@action=redirect&bpo=28666]: Now test.support.rmtree is able to remove unwritable or unreadable directories.
- bpo-23839 [https://bugs.python.org/issue?@action=redirect&bpo=23839]: Various caches now are cleared before running every test file.

Build

- bpo-10656 [https://bugs.python.org/issue?@action=redirect&bpo=10656]: Fix out-of-tree building on AIX. Patch by Tristan Carel and Michael Haubenwallner.
- bpo-26359 [https://bugs.python.org/issue?@action=redirect&bpo=26359]: Rename –with-optimiations to –enable-optimizations.
- bpo-28676 [https://bugs.python.org/issue?@action=redirect&bpo=28676]: Prevent missing “getentropy” declaration warning on macOS. Patch by Gareth Rees.

Python 3.6.0 beta 3

Release date: 2016-10-31

Core and Builtins

- bpo-28128 [https://bugs.python.org/issue?@action=redirect&bpo=28128]: Deprecation warning for invalid str and byte escape sequences now prints better information about where the error occurs. Patch by Serhiy Storchaka and Eric Smith.
- bpo-28509 [https://bugs.python.org/issue?@action=redirect&bpo=28509]: dict.update() no longer allocate unnecessary large memory.
- bpo-28426 [https://bugs.python.org/issue?@action=redirect&bpo=28426]: Fixed potential crash in PyUnicode_AsDecodedObject() in debug build.
- bpo-28517 [https://bugs.python.org/issue?@action=redirect&bpo=28517]: Fixed of-by-one error in the peephole optimizer that caused keeping unreachable code.
- bpo-28214 [https://bugs.python.org/issue?@action=redirect&bpo=28214]: Improved exception reporting for problematic __set_name__ attributes.
- bpo-23782 [https://bugs.python.org/issue?@action=redirect&bpo=23782]: Fixed possible memory leak in _PyTraceback_Add() and exception loss in PyTraceBack_Here().
- bpo-28471 [https://bugs.python.org/issue?@action=redirect&bpo=28471]: Fix «Python memory allocator called without holding the GIL» crash in socket.setblocking.

Library

- bpo-27517 [https://bugs.python.org/issue?@action=redirect&bpo=27517]: LZMA compressor and decompressor no longer raise exceptions if given empty data twice. Patch by Benjamin Fogle.
- bpo-28549 [https://bugs.python.org/issue?@action=redirect&bpo=28549]: Fixed segfault in curses's addch() with ncurses6.
- bpo-28449 [https://bugs.python.org/issue?@action=redirect&bpo=28449]: tarfile.open() with mode «r» or

«r:» now tries to open a tar file with compression before trying to open it without compression. Otherwise it had 50% chance failed with ignore_zeros=True.

- bpo-23262 [https://bugs.python.org/issue?@action=redirect&bpo=23262]: The webbrowser module now supports Firefox 36+ and derived browsers. Based on patch by Oleg Broytman.
- bpo-27939 [https://bugs.python.org/issue?@action=redirect&bpo=27939]: Fixed bugs in tkinter.ttk.LabeledScale and tkinter.Scale caused by representing the scale as float value internally in Tk. tkinter.IntVar now works if float value is set to underlying Tk variable.
- bpo-18844 [https://bugs.python.org/issue?@action=redirect&bpo=18844]: The various ways of specifying weights for random.choices() now produce the same result sequences.
- bpo-28255 [https://bugs.python.org/issue?@action=redirect&bpo=28255]: calendar.TextCalendar().prmonth() no longer prints a space at the start of new line after printing a month's calendar. Patch by Xiang Zhang.
- bpo-20491 [https://bugs.python.org/issue?@action=redirect&bpo=20491]: The textwrap.TextWrapper class now honors non-breaking spaces. Based on patch by Kaarle Ritvanen.
- bpo-28353 [https://bugs.python.org/issue?@action=redirect&bpo=28353]: os.fwalk() no longer fails on broken links.
- bpo-28430 [https://bugs.python.org/issue?@action=redirect&bpo=28430]: Fix iterator of C implemented asyncio.Future doesn't accept non-None value is passed to it.send(val).
- bpo-27025 [https://bugs.python.org/issue?@action=redirect&bpo=27025]: Generated names for Tkinter widgets now start by the «!» prefix for readability.
- bpo-25464 [https://bugs.python.org/issue?@action=redirect&bpo=25464]: Fixed HList.header_exists() in tkinter.tix module by addin a workaround to Tix library bug.
- bpo-28488 [https://bugs.python.org/issue?

@action=redirect&bpo=28488]: shutil.make_archive() no longer adds entry «./» to ZIP archive.

- bpo-25953 [https://bugs.python.org/issue?@action=redirect&bpo=25953]: re.sub() now raises an error for invalid numerical group reference in replacement template even if the pattern is not found in the string. Error message for invalid group reference now includes the group index and the position of the reference. Based on patch by SilentGhost.
- bpo-18219 [https://bugs.python.org/issue?@action=redirect&bpo=18219]: Optimize csv.DictWriter for large number of columns. Patch by Mariatta Wijaya.
- bpo-28448 [https://bugs.python.org/issue?@action=redirect&bpo=28448]: Fix C implemented asyncio.Future didn’t work on Windows.
- bpo-28480 [https://bugs.python.org/issue?@action=redirect&bpo=28480]: Fix error building socket module when multithreading is disabled.
- bpo-24452 [https://bugs.python.org/issue?@action=redirect&bpo=24452]: Make webbrowser support Chrome on Mac OS X.
- bpo-20766 [https://bugs.python.org/issue?@action=redirect&bpo=20766]: Fix references leaked by pdb in the handling of SIGINT handlers.
- bpo-28492 [https://bugs.python.org/issue?@action=redirect&bpo=28492]: Fix how StopIteration exception is raised in _asyncio.Future.
- bpo-28500 [https://bugs.python.org/issue?@action=redirect&bpo=28500]: Fix asyncio to handle async gens GC from another thread.
- bpo-26923 [https://bugs.python.org/issue?@action=redirect&bpo=26923]: Fix asyncio.Gather to refuse being cancelled once all children are done. Patch by Johannes Ebke.
- bpo-26796 [https://bugs.python.org/issue?@action=redirect&bpo=26796]: Don’t configure the number of workers for default threadpool executor. Initial patch by Hans Lawrenz.
- bpo-28544 [https://bugs.python.org/issue?@action=redirect&bpo=28544]: Implement asyncio.Task in C.

Windows

- bpo-28522 [https://bugs.python.org/issue?@action=redirect&bpo=28522]: Fixes mishandled buffer reallocation in getpathp.c

Build

- bpo-28444 [https://bugs.python.org/issue?@action=redirect&bpo=28444]: Fix missing extensions modules when cross compiling.
- bpo-28208 [https://bugs.python.org/issue?@action=redirect&bpo=28208]: Update Windows build and OS X installers to use SQLite 3.14.2.
- bpo-28248 [https://bugs.python.org/issue?@action=redirect&bpo=28248]: Update Windows build and OS X installers to use OpenSSL 1.0.2j.

Tests

- bpo-26944 [https://bugs.python.org/issue?@action=redirect&bpo=26944]: Fix test_posix for Android where “id -G” is entirely wrong or missing the effective gid.
- bpo-28409 [https://bugs.python.org/issue?@action=redirect&bpo=28409]: regrtest: fix the parser of command line arguments.

Python 3.6.0 beta 2

Release date: 2016-10-10

Core and Builtins

- bpo-28183 [https://bugs.python.org/issue?@action=redirect&bpo=28183]: Optimize and cleanup dict iteration.
- bpo-26081 [https://bugs.python.org/issue?@action=redirect&bpo=26081]: Added C implementation of asyncio.Future. Original patch by Yury Selivanov.
- bpo-28379 [https://bugs.python.org/issue?@action=redirect&bpo=28379]: Added sanity checks and tests for

PyUnicode_CopyCharacters(). Patch by Xiang Zhang.

- bpo-28376 [https://bugs.python.org/issue?@action=redirect&bpo=28376]: The type of long range iterator is now registered as Iterator. Patch by Oren Milman.
- bpo-28376 [https://bugs.python.org/issue?@action=redirect&bpo=28376]: Creating instances of range_iterator by calling range_iterator type now is deprecated. Patch by Oren Milman.
- bpo-28376 [https://bugs.python.org/issue?@action=redirect&bpo=28376]: The constructor of range_iterator now checks that step is not 0. Patch by Oren Milman.
- bpo-26906 [https://bugs.python.org/issue?@action=redirect&bpo=26906]: Resolving special methods of uninitialized type now causes implicit initialization of the type instead of a fail.
- bpo-18287 [https://bugs.python.org/issue?@action=redirect&bpo=18287]: PyType_Ready() now checks that tp_name is not NULL. Original patch by Niklas Koep.
- bpo-24098 [https://bugs.python.org/issue?@action=redirect&bpo=24098]: Fixed possible crash when AST is changed in process of compiling it.
- bpo-28201 [https://bugs.python.org/issue?@action=redirect&bpo=28201]: Dict reduces possibility of 2nd conflict in hash table when hashes have same lower bits.
- bpo-28350 [https://bugs.python.org/issue?@action=redirect&bpo=28350]: String constants with null character no longer interned.
- bpo-26617 [https://bugs.python.org/issue?@action=redirect&bpo=26617]: Fix crash when GC runs during weakref callbacks.
- bpo-27942 [https://bugs.python.org/issue?@action=redirect&bpo=27942]: String constants now interned recursively in tuples and frozensets.
- bpo-21578 [https://bugs.python.org/issue?@action=redirect&bpo=21578]: Fixed misleading error message when ImportError called with invalid keyword args.
- bpo-28203 [https://bugs.python.org/issue?@action=redirect&bpo=28203]: Fix incorrect type in complex(1.0, {2:3}) error message. Patch by Soumya Sharma.
- bpo-28086 [https://bugs.python.org/issue?

@action=redirect&bpo=28086]: Single var-positional argument of tuple subtype was passed unscathed to the C-defined function. Now it is converted to exact tuple.
- bpo-28214 [https://bugs.python.org/issue?@action=redirect&bpo=28214]: Now __set_name__ is looked up on the class instead of the instance.
- bpo-27955 [https://bugs.python.org/issue?@action=redirect&bpo=27955]: Fallback on reading /dev/urandom device when the getrandom() syscall fails with EPERM, for example when blocked by SECCOMP.
- bpo-28192 [https://bugs.python.org/issue?@action=redirect&bpo=28192]: Don't import readline in isolated mode.
- Upgrade internal unicode databases to Unicode version 9.0.0.
- bpo-28131 [https://bugs.python.org/issue?@action=redirect&bpo=28131]: Fix a regression in zipimport's compile_source(). zipimport should use the same optimization level as the interpreter.
- bpo-28126 [https://bugs.python.org/issue?@action=redirect&bpo=28126]: Replace Py_MEMCPY with memcpy(). Visual Studio can properly optimize memcpy().
- bpo-28120 [https://bugs.python.org/issue?@action=redirect&bpo=28120]: Fix dict.pop() for split dictionary when trying to remove a «pending key» (Not yet inserted in split-table). Patch by Xiang Zhang.
- bpo-26182 [https://bugs.python.org/issue?@action=redirect&bpo=26182]: Raise DeprecationWarning when async and await keywords are used as variable/attribute/class/function name.

Library

- bpo-27998 [https://bugs.python.org/issue?@action=redirect&bpo=27998]: Fixed bytes path support in os.scandir() on Windows. Patch by Eryk Sun.
- bpo-28317 [https://bugs.python.org/issue?@action=redirect&bpo=28317]: The disassembler now decodes FORMAT_VALUE argument.
- bpo-26293 [https://bugs.python.org/issue?@action=redirect&bpo=26293]: Fixed writing ZIP files that starts

not from the start of the file. Offsets in ZIP file now are relative to the start of the archive in conforming to the specification.

- bpo-28380 [https://bugs.python.org/issue?@action=redirect&bpo=28380]: unittest.mock Mock autospec functions now properly support assert_called, assert_not_called, and assert_called_once.
- bpo-27181 [https://bugs.python.org/issue?@action=redirect&bpo=27181]: remove statistics.geometric_mean and defer until 3.7.
- bpo-28229 [https://bugs.python.org/issue?@action=redirect&bpo=28229]: lzma module now supports pathlib.
- bpo-28321 [https://bugs.python.org/issue?@action=redirect&bpo=28321]: Fixed writing non-BMP characters with binary format in plistlib.
- bpo-28225 [https://bugs.python.org/issue?@action=redirect&bpo=28225]: bz2 module now supports pathlib. Initial patch by Ethan Furman.
- bpo-28227 [https://bugs.python.org/issue?@action=redirect&bpo=28227]: gzip now supports pathlib. Patch by Ethan Furman.
- bpo-27358 [https://bugs.python.org/issue?@action=redirect&bpo=27358]: Optimized merging var-keyword arguments and improved error message when passing a non-mapping as a var-keyword argument.
- bpo-28257 [https://bugs.python.org/issue?@action=redirect&bpo=28257]: Improved error message when passing a non-iterable as a var-positional argument. Added opcode BUILD_TUPLE_UNPACK_WITH_CALL.
- bpo-28322 [https://bugs.python.org/issue?@action=redirect&bpo=28322]: Fixed possible crashes when unpickle itertools objects from incorrect pickle data. Based on patch by John Leitch.
- bpo-28228 [https://bugs.python.org/issue?@action=redirect&bpo=28228]: imghdr now supports pathlib.
- bpo-28226 [https://bugs.python.org/issue?@action=redirect&bpo=28226]: compileall now supports pathlib.
- bpo-28314 [https://bugs.python.org/issue?@action=redirect&bpo=28314]: Fix function declaration (C flags)

for the getiterator() method of xml.etree.ElementTree.Element.

- bpo-28148 [https://bugs.python.org/issue?@action=redirect&bpo=28148]: Stop using localtime() and gmtime() in the time module. Introduced platform independent _PyTime_localtime API that is similar to POSIX localtime_r, but available on all platforms. Patch by Ed Schouten.
- bpo-28253 [https://bugs.python.org/issue?@action=redirect&bpo=28253]: Fixed calendar functions for extreme months: 0001-01 and 9999-12. Methods itermonthdays() and itermonthdays2() are reimplemented so that they don't call itermonthdates() which can cause datetime.date under/overflow.
- bpo-28275 [https://bugs.python.org/issue?@action=redirect&bpo=28275]: Fixed possible use after free in the decompress() methods of the LZMADecompressor and BZ2Decompressor classes. Original patch by John Leitch.
- bpo-27897 [https://bugs.python.org/issue?@action=redirect&bpo=27897]: Fixed possible crash in sqlite3.Connection.create_collation() if pass invalid string-like object as a name. Patch by Xiang Zhang.
- bpo-18844 [https://bugs.python.org/issue?@action=redirect&bpo=18844]: random.choices() now has k as a keyword-only argument to improve the readability of common cases and come into line with the signature used in other languages.
- bpo-18893 [https://bugs.python.org/issue?@action=redirect&bpo=18893]: Fix invalid exception handling in Lib/ctypes/macholib/dyld.py. Patch by Madison May.
- bpo-27611 [https://bugs.python.org/issue?@action=redirect&bpo=27611]: Fixed support of default root window in the tkinter.tix module. Added the master parameter in the DisplayStyle constructor.
- bpo-27348 [https://bugs.python.org/issue?@action=redirect&bpo=27348]: In the traceback module, restore the formatting of exception messages like «Exception: None». This fixes a regression introduced in 3.5a2.
- bpo-25651 [https://bugs.python.org/issue?@action=redirect&bpo=25651]: Allow false values to be used for

msg parameter of subTest().

- bpo-27778 [https://bugs.python.org/issue?@action=redirect&bpo=27778]: Fix a memory leak in os.getrandom() when the getrandom() is interrupted by a signal and a signal handler raises a Python exception.
- bpo-28200 [https://bugs.python.org/issue?@action=redirect&bpo=28200]: Fix memory leak on Windows in the os module (fix path_converter() function).
- bpo-25400 [https://bugs.python.org/issue?@action=redirect&bpo=25400]: RobotFileParser now correctly returns default values for crawl_delay and request_rate. Initial patch by Peter Wirtz.
- bpo-27932 [https://bugs.python.org/issue?@action=redirect&bpo=27932]: Prevent memory leak in win32_ver().
- Fix UnboundLocalError in socket._sendfile_use_sendfile.
- bpo-28075 [https://bugs.python.org/issue?@action=redirect&bpo=28075]: Check for ERROR_ACCESS_DENIED in Windows implementation of os.stat(). Patch by Eryk Sun.
- bpo-22493 [https://bugs.python.org/issue?@action=redirect&bpo=22493]: Warning message emitted by using inline flags in the middle of regular expression now contains a (truncated) regex pattern. Patch by Tim Graham.
- bpo-25270 [https://bugs.python.org/issue?@action=redirect&bpo=25270]: Prevent codecs.escape_encode() from raising SystemError when an empty bytestring is passed.
- bpo-28181 [https://bugs.python.org/issue?@action=redirect&bpo=28181]: Get antigravity over HTTPS. Patch by Kaartic Sivaraam.
- bpo-25895 [https://bugs.python.org/issue?@action=redirect&bpo=25895]: Enable WebSocket URL schemes in urllib.parse.urljoin. Patch by Gergely Imreh and Markus Holtermann.
- bpo-28114 [https://bugs.python.org/issue?@action=redirect&bpo=28114]: Fix a crash in parse_envlist() when env contains byte strings. Patch by Eryk Sun.
- bpo-27599 [https://bugs.python.org/issue?@action=redirect&bpo=27599]: Fixed buffer overrun in binascii.b2a_qp() and binascii.a2b_qp().

- bpo-27906 [https://bugs.python.org/issue?@action=redirect&bpo=27906]: Fix socket accept exhaustion during high TCP traffic. Patch by Kevin Conway.
- bpo-28174 [https://bugs.python.org/issue?@action=redirect&bpo=28174]: Handle when SO_REUSEPORT isn’t properly supported. Patch by Seth Michael Larson.
- bpo-26654 [https://bugs.python.org/issue?@action=redirect&bpo=26654]: Inspect functools.partial in asyncio.Handle.__repr__. Patch by iceboy.
- bpo-26909 [https://bugs.python.org/issue?@action=redirect&bpo=26909]: Fix slow pipes IO in asyncio. Patch by INADA Naoki.
- bpo-28176 [https://bugs.python.org/issue?@action=redirect&bpo=28176]: Fix callbacks race in asyncio.SelectorLoop.sock_connect.
- bpo-27759 [https://bugs.python.org/issue?@action=redirect&bpo=27759]: Fix selectors incorrectly retain invalid file descriptors. Patch by Mark Williams.
- bpo-28368 [https://bugs.python.org/issue?@action=redirect&bpo=28368]: Refuse monitoring processes if the child watcher has no loop attached. Patch by Vincent Michel.
- bpo-28369 [https://bugs.python.org/issue?@action=redirect&bpo=28369]: Raise RuntimeError when transport’s FD is used with add_reader, add_writer, etc.
- bpo-28370 [https://bugs.python.org/issue?@action=redirect&bpo=28370]: Speedup asyncio.StreamReader.readexactly. Patch by Коренберг Марк.
- bpo-28371 [https://bugs.python.org/issue?@action=redirect&bpo=28371]: Deprecate passing asyncio.Handles to run_in_executor.
- bpo-28372 [https://bugs.python.org/issue?@action=redirect&bpo=28372]: Fix asyncio to support formatting of non-python coroutines.
- bpo-28399 [https://bugs.python.org/issue?@action=redirect&bpo=28399]: Remove UNIX socket from FS before binding. Patch by Коренберг Марк.
- bpo-27972 [https://bugs.python.org/issue?@action=redirect&bpo=27972]: Prohibit Tasks to await on themselves.

Windows

- bpo-28402 [https://bugs.python.org/issue?@action=redirect&bpo=28402]: Adds signed catalog files for stdlib on Windows.
- bpo-28333 [https://bugs.python.org/issue?@action=redirect&bpo=28333]: Enables Unicode for ps1/ps2 and input() prompts. (Patch by Eryk Sun)
- bpo-28251 [https://bugs.python.org/issue?@action=redirect&bpo=28251]: Improvements to help manuals on Windows.
- bpo-28110 [https://bugs.python.org/issue?@action=redirect&bpo=28110]: launcher.msi has different product codes between 32-bit and 64-bit
- bpo-28161 [https://bugs.python.org/issue?@action=redirect&bpo=28161]: Opening CON for write access fails
- bpo-28162 [https://bugs.python.org/issue?@action=redirect&bpo=28162]: WindowsConsoleIO readall() fails if first line starts with Ctrl+Z
- bpo-28163 [https://bugs.python.org/issue?@action=redirect&bpo=28163]: WindowsConsoleIO fileno() passes wrong flags to _open_osfhandle
- bpo-28164 [https://bugs.python.org/issue?@action=redirect&bpo=28164]: _PyIO_get_console_type fails for various paths
- bpo-28137 [https://bugs.python.org/issue?@action=redirect&bpo=28137]: Renames Windows path file to ._pth
- bpo-28138 [https://bugs.python.org/issue?@action=redirect&bpo=28138]: Windows ._pth file should allow import site

C API

- bpo-28426 [https://bugs.python.org/issue?@action=redirect&bpo=28426]: Deprecated undocumented functions PyUnicode_AsEncodedObject(), PyUnicode_AsDecodedObject(), PyUnicode_AsDecodedUnicode() and

PyUnicode_AsEncodedUnicode().

Build

- bpo-28258 [https://bugs.python.org/issue?@action=redirect&bpo=28258]: Fixed build with Estonian locale (python-config and distclean targets in Makefile). Patch by Arfrever Frehtes Taifersar Arahesis.
- bpo-26661 [https://bugs.python.org/issue?@action=redirect&bpo=26661]: setup.py now detects system libffi with multiarch wrapper.
- bpo-15819 [https://bugs.python.org/issue?@action=redirect&bpo=15819]: Remove redundant include search directory option for building outside the source tree.

Tests

- bpo-28217 [https://bugs.python.org/issue?@action=redirect&bpo=28217]: Adds _testconsole module to test console input.

Python 3.6.0 beta 1

Release date: 2016-09-12

Core and Builtins

- bpo-23722 [https://bugs.python.org/issue?@action=redirect&bpo=23722]: The _class_ cell used by zero-argument super() is now initialized from type._new_ rather than _build_class_, so class methods relying on that will now work correctly when called from metaclass methods during class creation. Patch by Martin Teichmann.
- bpo-25221 [https://bugs.python.org/issue?@action=redirect&bpo=25221]: Fix corrupted result from PyLong_FromLong(0) when Python is compiled with NSMALLPOSINTS = 0.
- bpo-27080 [https://bugs.python.org/issue?@action=redirect&bpo=27080]: Implement formatting support for

PEP 515 [https://peps.python.org/pep-0515/]. Initial patch by Chris Angelico.

- bpo-27199 [https://bugs.python.org/issue?@action=redirect&bpo=27199]: In tarfile, expose copyfileobj bufsize to improve throughput. Patch by Jason Fried.
- bpo-27948 [https://bugs.python.org/issue?@action=redirect&bpo=27948]: In f-strings, only allow backslashes inside the braces (where the expressions are). This is a breaking change from the 3.6 alpha releases, where backslashes are allowed anywhere in an f-string. Also, require that expressions inside f-strings be enclosed within literal braces, and not escapes like `f'\x7b"hi"\x7d'`.
- bpo-28046 [https://bugs.python.org/issue?@action=redirect&bpo=28046]: Remove platform-specific directories from sys.path.
- bpo-28071 [https://bugs.python.org/issue?@action=redirect&bpo=28071]: Add early-out for differencing from an empty set.
- bpo-25758 [https://bugs.python.org/issue?@action=redirect&bpo=25758]: Prevents zipimport from unnecessarily encoding a filename (patch by Eryk Sun)
- bpo-25856 [https://bugs.python.org/issue?@action=redirect&bpo=25856]: The _module_ attribute of extension classes and functions now is interned. This leads to more compact pickle data with protocol 4.
- bpo-27213 [https://bugs.python.org/issue?@action=redirect&bpo=27213]: Rework CALL_FUNCTION* opcodes to produce shorter and more efficient bytecode. Patch by Demur Rumed, design by Serhiy Storchaka, reviewed by Serhiy Storchaka and Victor Stinner.
- bpo-26331 [https://bugs.python.org/issue?@action=redirect&bpo=26331]: Implement tokenizing support for **PEP 515** [https://peps.python.org/pep-0515/]. Patch by Georg Brandl.
- bpo-27999 [https://bugs.python.org/issue?@action=redirect&bpo=27999]: Make «global after use» a SyntaxError, and ditto for nonlocal. Patch by Ivan Levkivskyi.
- bpo-28003 [https://bugs.python.org/issue?@action=redirect&bpo=28003]: Implement **PEP 525** [https://peps.python.org/pep-0525/] – Asynchronous Generators.

- bpo-27985 [https://bugs.python.org/issue?@action=redirect&bpo=27985]: Implement **PEP 526** [https://peps.python.org/pep-0526/] – Syntax for Variable Annotations. Patch by Ivan Levkivskyi.
- bpo-26058 [https://bugs.python.org/issue?@action=redirect&bpo=26058]: Add a new private version to the builtin dict type, incremented at each dictionary creation and at each dictionary change. Implementation of the PEP 509.
- bpo-27364 [https://bugs.python.org/issue?@action=redirect&bpo=27364]: A backslash-character pair that is not a valid escape sequence now generates a DeprecationWarning. Patch by Emanuel Barry.
- bpo-27350 [https://bugs.python.org/issue?@action=redirect&bpo=27350]: `dict` implementation is changed like PyPy. It is more compact and preserves insertion order. (Concept developed by Raymond Hettinger and patch by Inada Naoki.)
- bpo-27911 [https://bugs.python.org/issue?@action=redirect&bpo=27911]: Remove unnecessary error checks in `exec_builtin_or_dynamic()`.
- bpo-27078 [https://bugs.python.org/issue?@action=redirect&bpo=27078]: Added BUILD_STRING opcode. Optimized f-strings evaluation.
- bpo-17884 [https://bugs.python.org/issue?@action=redirect&bpo=17884]: Python now requires systems with inttypes.h and stdint.h
- bpo-27961 [https://bugs.python.org/issue?@action=redirect&bpo=27961]: Require platforms to support `long long`. Python hasn’t compiled without `long long` for years, so this is basically a formality.
- bpo-27355 [https://bugs.python.org/issue?@action=redirect&bpo=27355]: Removed support for Windows CE. It was never finished, and Windows CE is no longer a relevant platform for Python.
- Implement **PEP 523** [https://peps.python.org/pep-0523/].
- bpo-27870 [https://bugs.python.org/issue?@action=redirect&bpo=27870]: A left shift of zero by a large integer no longer attempts to allocate large amounts of memory.
- bpo-25402 [https://bugs.python.org/issue?

@action=redirect&bpo=25402]: In int-to-decimal-string conversion, improve the estimate of the intermediate memory required, and remove an unnecessarily strict overflow check. Patch by Serhiy Storchaka.
- bpo-27214 [https://bugs.python.org/issue?@action=redirect&bpo=27214]: In long_invert, be more careful about modifying object returned by long_add, and remove an unnecessary check for small longs. Thanks Oren Milman for analysis and patch.
- bpo-27506 [https://bugs.python.org/issue?@action=redirect&bpo=27506]: Support passing the bytes/bytearray.translate() «delete» argument by keyword.
- bpo-27812 [https://bugs.python.org/issue?@action=redirect&bpo=27812]: Properly clear out a generator’s frame’s backreference to the generator to prevent crashes in frame.clear().
- bpo-27811 [https://bugs.python.org/issue?@action=redirect&bpo=27811]: Fix a crash when a coroutine that has not been awaited is finalized with warnings-as-errors enabled.
- bpo-27587 [https://bugs.python.org/issue?@action=redirect&bpo=27587]: Fix another issue found by PVS-Studio: Null pointer check after use of “def” in _PyState_AddModule(). Initial patch by Christian Heimes.
- bpo-27792 [https://bugs.python.org/issue?@action=redirect&bpo=27792]: The modulo operation applied to `bool` and other `int` subclasses now always returns an `int`. Previously the return type depended on the input values. Patch by Xiang Zhang.
- bpo-26984 [https://bugs.python.org/issue?@action=redirect&bpo=26984]: int() now always returns an instance of exact int.
- bpo-25604 [https://bugs.python.org/issue?@action=redirect&bpo=25604]: Fix a minor bug in integer true division; this bug could potentially have caused off-by-one-ulp results on platforms with unreliable ldexp implementations.
- bpo-24254 [https://bugs.python.org/issue?@action=redirect&bpo=24254]: Make class definition namespace ordered by default.

- bpo-27662 [https://bugs.python.org/issue?@action=redirect&bpo=27662]: Fix an overflow check in `List_New`: the original code was checking against `Py_SIZE_MAX` instead of the correct upper bound of `Py_SSIZE_T_MAX`. Patch by Xiang Zhang.
- bpo-27782 [https://bugs.python.org/issue?@action=redirect&bpo=27782]: Multi-phase extension module import now correctly allows the `m_methods` field to be used to add module level functions to instances of non-module types returned from `Py_create_mod`. Patch by Xiang Zhang.
- bpo-27936 [https://bugs.python.org/issue?@action=redirect&bpo=27936]: The round() function accepted a second None argument for some types but not for others. Fixed the inconsistency by accepting None for all numeric types.
- bpo-27487 [https://bugs.python.org/issue?@action=redirect&bpo=27487]: Warn if a submodule argument to «python -m» or runpy.run_module() is found in sys.modules after parent packages are imported, but before the submodule is executed.
- bpo-27157 [https://bugs.python.org/issue?@action=redirect&bpo=27157]: Make only type() itself accept the one-argument form. Patch by Eryk Sun and Emanuel Barry.
- bpo-27558 [https://bugs.python.org/issue?@action=redirect&bpo=27558]: Fix a SystemError in the implementation of «raise» statement. In a brand new thread, raise a RuntimeError since there is no active exception to reraise. Patch written by Xiang Zhang.
- bpo-28008 [https://bugs.python.org/issue?@action=redirect&bpo=28008]: Implement **PEP 530** [https://peps.python.org/pep-0530/] – asynchronous comprehensions.
- bpo-27942 [https://bugs.python.org/issue?@action=redirect&bpo=27942]: Fix memory leak in codeobject.c

Library

- bpo-28732 [https://bugs.python.org/issue?@action=redirect&bpo=28732]: Fix crash in os.spawnv() with no elements in args

- bpo-28485 [https://bugs.python.org/issue?@action=redirect&bpo=28485]: Always raise ValueError for negative compileall.compile_dir(workers=...) parameter, even when multithreading is unavailable.
- bpo-28037 [https://bugs.python.org/issue?@action=redirect&bpo=28037]: Use sqlite3_get_autocommit() instead of setting Connection->inTransaction manually.
- bpo-25283 [https://bugs.python.org/issue?@action=redirect&bpo=25283]: Attributes tm_gmtoff and tm_zone are now available on all platforms in the return values of time.localtime() and time.gmtime().
- bpo-24454 [https://bugs.python.org/issue?@action=redirect&bpo=24454]: Regular expression match object groups are now accessible using __getitem__. «mo[x]» is equivalent to «mo.group(x)».
- bpo-10740 [https://bugs.python.org/issue?@action=redirect&bpo=10740]: sqlite3 no longer implicitly commit an open transaction before DDL statements.
- bpo-17941 [https://bugs.python.org/issue?@action=redirect&bpo=17941]: Add a *module* parameter to collections.namedtuple().
- bpo-22493 [https://bugs.python.org/issue?@action=redirect&bpo=22493]: Inline flags now should be used only at the start of the regular expression. Deprecation warning is emitted if uses them in the middle of the regular expression.
- bpo-26885 [https://bugs.python.org/issue?@action=redirect&bpo=26885]: xmlrpc now supports unmarshalling additional data types used by Apache XML-RPC implementation for numerics and None.
- bpo-28070 [https://bugs.python.org/issue?@action=redirect&bpo=28070]: Fixed parsing inline verbose flag in regular expressions.
- bpo-19500 [https://bugs.python.org/issue?@action=redirect&bpo=19500]: Add client-side SSL session resumption to the ssl module.
- bpo-28022 [https://bugs.python.org/issue?@action=redirect&bpo=28022]: Deprecate ssl-related arguments in favor of SSLContext. The deprecation include manual creation of SSLSocket and certfile/keyfile (or similar) in

ftplib, httplib, imaplib, smtplib, poplib and urllib.

- bpo-28043 [https://bugs.python.org/issue?@action=redirect&bpo=28043]: SSLContext has improved default settings: OP_NO_SSLv2, OP_NO_SSLv3, OP_NO_COMPRESSION, OP_CIPHER_SERVER_PREFERENCE, OP_SINGLE_DH_USE, OP_SINGLE_ECDH_USE and HIGH ciphers without MD5.
- bpo-24693 [https://bugs.python.org/issue?@action=redirect&bpo=24693]: Changed some RuntimeError's in the zipfile module to more appropriate types. Improved some error messages and debugging output.
- bpo-17909 [https://bugs.python.org/issue?@action=redirect&bpo=17909]: `json.load` and `json.loads` now support binary input encoded as UTF-8, UTF-16 or UTF-32. Patch by Serhiy Storchaka.
- bpo-27137 [https://bugs.python.org/issue?@action=redirect&bpo=27137]: the pure Python fallback implementation of `functools.partial` now matches the behaviour of its accelerated C counterpart for subclassing, pickling and text representation purposes. Patch by Emanuel Barry and Serhiy Storchaka.
- Fix possible integer overflows and crashes in the mmap module with unusual usage patterns.
- bpo-1703178 [https://bugs.python.org/issue?@action=redirect&bpo=1703178]: Fix the ability to pass the –link-objects option to the distutils build_ext command.
- bpo-28019 [https://bugs.python.org/issue?@action=redirect&bpo=28019]: itertools.count() no longer rounds non-integer step in range between 1.0 and 2.0 to 1.
- bpo-18401 [https://bugs.python.org/issue?@action=redirect&bpo=18401]: Pdb now supports the “readrc” keyword argument to control whether .pdbrc files should be read. Patch by Martin Matusiak and Sam Kimbrel.
- bpo-25969 [https://bugs.python.org/issue?@action=redirect&bpo=25969]: Update the lib2to3 grammar to handle the unpacking generalizations added in 3.5.
- bpo-14977 [https://bugs.python.org/issue?@action=redirect&bpo=14977]: mailcap now respects the order of the lines in the mailcap files («first match»), as required by RFC 1542. Patch by Michael Lazar.

- bpo-28082 [https://bugs.python.org/issue?@action=redirect&bpo=28082]: Convert re flag constants to IntFlag.
- bpo-28025 [https://bugs.python.org/issue?@action=redirect&bpo=28025]: Convert all ssl module constants to IntEnum and IntFlags. SSLContext properties now return flags and enums.
- bpo-23591 [https://bugs.python.org/issue?@action=redirect&bpo=23591]: Add Flag, IntFlag, and auto() to enum module.
- bpo-433028 [https://bugs.python.org/issue?@action=redirect&bpo=433028]: Added support of modifier spans in regular expressions.
- bpo-24594 [https://bugs.python.org/issue?@action=redirect&bpo=24594]: Validates persist parameter when opening MSI database
- bpo-17582 [https://bugs.python.org/issue?@action=redirect&bpo=17582]: xml.etree.ElementTree nows preserves whitespaces in attributes (Patch by Duane Griffin. Reviewed and approved by Stefan Behnel.)
- bpo-28047 [https://bugs.python.org/issue?@action=redirect&bpo=28047]: Fixed calculation of line length used for the base64 CTE in the new email policies.
- bpo-27576 [https://bugs.python.org/issue?@action=redirect&bpo=27576]: Fix call order in OrderedDict.__init__().
- email.generator.DecodedGenerator now supports the policy keyword.
- bpo-28027 [https://bugs.python.org/issue?@action=redirect&bpo=28027]: Remove undocumented modules from `Lib/plat-*`: IN, CDROM, DLFCN, TYPES, CDIO, and STROPTS.
- bpo-27445 [https://bugs.python.org/issue?@action=redirect&bpo=27445]: Don’t pass str(_charset) to MIMEText.set_payload(). Patch by Claude Paroz.
- bpo-24277 [https://bugs.python.org/issue?@action=redirect&bpo=24277]: The new email API is no longer provisional, and the docs have been reorganized and rewritten to emphasize the new API.
- bpo-22450 [https://bugs.python.org/issue?

@action=redirect&bpo=22450]: urllib now includes an `Accept: */*` header among the default headers. This makes the results of REST API requests more consistent and predictable especially when proxy servers are involved.
- lib2to3.pgen3.driver.load_grammar() now creates a stable cache file between runs given the same Grammar.txt input regardless of the hash randomization setting.
- bpo-28005 [https://bugs.python.org/issue?@action=redirect&bpo=28005]: Allow ImportErrors in encoding implementation to propagate.
- bpo-26667 [https://bugs.python.org/issue?@action=redirect&bpo=26667]: Support path-like objects in importlib.util.
- bpo-27570 [https://bugs.python.org/issue?@action=redirect&bpo=27570]: Avoid zero-length memcpy() etc calls with null source pointers in the «ctypes» and «array» modules.
- bpo-22233 [https://bugs.python.org/issue?@action=redirect&bpo=22233]: Break email header lines *only* on the RFC specified CR and LF characters, not on arbitrary unicode line breaks. This also fixes a bug in HTTP header parsing.
- bpo-27331 [https://bugs.python.org/issue?@action=redirect&bpo=27331]: The email.mime classes now all accept an optional policy keyword.
- bpo-27988 [https://bugs.python.org/issue?@action=redirect&bpo=27988]: Fix email iter_attachments incorrect mutation of payload list.
- bpo-16113 [https://bugs.python.org/issue?@action=redirect&bpo=16113]: Add SHA-3 and SHAKE support to hashlib module.
- Eliminate a tautological-pointer-compare warning in _scproxy.c.
- bpo-27776 [https://bugs.python.org/issue?@action=redirect&bpo=27776]: The **`os.urandom()`** function does now block on Linux 3.17 and newer until the system urandom entropy pool is initialized to increase the security. This change is part of the **PEP 524** [https://peps.python.org/pep-0524/].
- bpo-27778 [https://bugs.python.org/issue?

@action=redirect&bpo=27778]: Expose the Linux `getrandom()` syscall as a new **`os.getrandom()`** function. This change is part of the **PEP 524** [https://peps.python.org/pep-0524/].

- bpo-27691 [https://bugs.python.org/issue?@action=redirect&bpo=27691]: Fix ssl module's parsing of GEN_RID subject alternative name fields in X.509 certs.
- bpo-18844 [https://bugs.python.org/issue?@action=redirect&bpo=18844]: Add random.choices().
- bpo-25761 [https://bugs.python.org/issue?@action=redirect&bpo=25761]: Improved error reporting about truncated pickle data in C implementation of unpickler. UnpicklingError is now raised instead of AttributeError and ValueError in some cases.
- bpo-26798 [https://bugs.python.org/issue?@action=redirect&bpo=26798]: Add BLAKE2 (blake2b and blake2s) to hashlib.
- bpo-26032 [https://bugs.python.org/issue?@action=redirect&bpo=26032]: Optimized globbing in pathlib by using os.scandir(); it is now about 1.5–4 times faster.
- bpo-25596 [https://bugs.python.org/issue?@action=redirect&bpo=25596]: Optimized glob() and iglob() functions in the glob module; they are now about 3–6 times faster.
- bpo-27928 [https://bugs.python.org/issue?@action=redirect&bpo=27928]: Add scrypt (password-based key derivation function) to hashlib module (requires OpenSSL 1.1.0).
- bpo-27850 [https://bugs.python.org/issue?@action=redirect&bpo=27850]: Remove 3DES from ssl module's default cipher list to counter measure sweet32 attack (**CVE 2016-2183** [https://www.cve.org/CVERecord?id=CVE-2016-2183]).
- bpo-27766 [https://bugs.python.org/issue?@action=redirect&bpo=27766]: Add ChaCha20 Poly1305 to ssl module's default cipher list. (Required OpenSSL 1.1.0 or LibreSSL).
- bpo-25387 [https://bugs.python.org/issue?@action=redirect&bpo=25387]: Check return value of winsound.MessageBeep.
- bpo-27866 [https://bugs.python.org/issue?@action=redirect&bpo=27866]: Add SSLContext.get_ciphers()

method to get a list of all enabled ciphers.

- bpo-27744 [https://bugs.python.org/issue?@action=redirect&bpo=27744]: Add AF_ALG (Linux Kernel crypto) to socket module.
- bpo-26470 [https://bugs.python.org/issue?@action=redirect&bpo=26470]: Port ssl and hashlib module to OpenSSL 1.1.0.
- bpo-11620 [https://bugs.python.org/issue?@action=redirect&bpo=11620]: Fix support for SND_MEMORY in winsound.PlaySound. Based on a patch by Tim Lesher.
- bpo-11734 [https://bugs.python.org/issue?@action=redirect&bpo=11734]: Add support for IEEE 754 half-precision floats to the struct module. Based on a patch by Eli Stevens.
- bpo-27919 [https://bugs.python.org/issue?@action=redirect&bpo=27919]: Deprecated `extra_path` distribution option in distutils packaging.
- bpo-23229 [https://bugs.python.org/issue?@action=redirect&bpo=23229]: Add new `cmath` constants: `cmath.inf` and `cmath.nan` to match `math.inf` and `math.nan`, and also `cmath.infj` and `cmath.nanj` to match the format used by complex repr.
- bpo-27842 [https://bugs.python.org/issue?@action=redirect&bpo=27842]: The csv.DictReader now returns rows of type OrderedDict. (Contributed by Steve Holden.)
- Remove support for passing a file descriptor to os.access. It never worked but previously didn’t raise.
- bpo-12885 [https://bugs.python.org/issue?@action=redirect&bpo=12885]: Fix error when distutils encounters symlink.
- bpo-27881 [https://bugs.python.org/issue?@action=redirect&bpo=27881]: Fixed possible bugs when setting sqlite3.Connection.isolation_level. Based on patch by Xiang Zhang.
- bpo-27861 [https://bugs.python.org/issue?@action=redirect&bpo=27861]: Fixed a crash in sqlite3.Connection.cursor() when a factory creates not a cursor. Patch by Xiang Zhang.
- bpo-19884 [https://bugs.python.org/issue?@action=redirect&bpo=19884]: Avoid spurious output on OS X

with Gnu Readline.

- bpo-27706 [https://bugs.python.org/issue?@action=redirect&bpo=27706]: Restore deterministic behavior of random.Random().seed() for string seeds using seeding version 1. Allows sequences of calls to random() to exactly match those obtained in Python 2. Patch by Nofar Schnider.
- bpo-10513 [https://bugs.python.org/issue?@action=redirect&bpo=10513]: Fix a regression in Connection.commit(). Statements should not be reset after a commit.
- bpo-12319 [https://bugs.python.org/issue?@action=redirect&bpo=12319]: Chunked transfer encoding support added to http.client.HTTPConnection requests. The urllib.request.AbstractHTTPHandler class does not enforce a Content-Length header any more. If a HTTP request has a file or iterable body, but no Content-Length header, the library now falls back to use chunked transfer-encoding.
- A new version of typing.py from https://github.com/python/typing: - Collection (only for 3.6) (bpo-27598 [https://bugs.python.org/issue?@action=redirect&bpo=27598]) - Add FrozenSet to _all_ (upstream #261) - fix crash in _get_type_vars() (upstream #259) - Remove the dict constraint in ForwardRef._eval_type (upstream #252)
- bpo-27832 [https://bugs.python.org/issue?@action=redirect&bpo=27832]: Make `_normalize` parameter to `Fraction` constructor keyword-only, so that `Fraction(2, 3, 4)` now raises `TypeError`.
- bpo-27539 [https://bugs.python.org/issue?@action=redirect&bpo=27539]: Fix unnormalised `Fraction.__pow__` result in the case of negative exponent and negative base.
- bpo-21718 [https://bugs.python.org/issue?@action=redirect&bpo=21718]: cursor.description is now available for queries using CTEs.
- bpo-27819 [https://bugs.python.org/issue?@action=redirect&bpo=27819]: In distutils sdists, simply produce the «gztar» (gzipped tar format) distributions on all platforms unless «formats» is supplied.
- bpo-2466 [https://bugs.python.org/issue?@action=redirect&bpo=2466]: posixpath.ismount now correctly recognizes mount points

which the user does not have permission to access.

- bpo-9998 [https://bugs.python.org/issue?@action=redirect&bpo=9998]: On Linux, ctypes.util.find_library now looks in LD_LIBRARY_PATH for shared libraries.
- bpo-27573 [https://bugs.python.org/issue?@action=redirect&bpo=27573]: exit message for code.interact is now configurable.
- bpo-27930 [https://bugs.python.org/issue?@action=redirect&bpo=27930]: Improved behaviour of logging.handlers.QueueListener. Thanks to Paulo Andrade and Petr Viktorin for the analysis and patch.
- bpo-6766 [https://bugs.python.org/issue?@action=redirect&bpo=6766]: Distributed reference counting added to multiprocessing to support nesting of shared values / proxy objects.
- bpo-21201 [https://bugs.python.org/issue?@action=redirect&bpo=21201]: Improves readability of multiprocessing error message. Thanks to Wojciech Walczak for patch.
- asyncio: Add set_protocol / get_protocol to Transports.
- bpo-27456 [https://bugs.python.org/issue?@action=redirect&bpo=27456]: asyncio: Set TCP_NODELAY by default.

IDLE

- bpo-15308 [https://bugs.python.org/issue?@action=redirect&bpo=15308]: Add “interrupt execution” (^C) to Shell menu. Patch by Roger Serwy, updated by Bayard Randel.
- bpo-27922 [https://bugs.python.org/issue?@action=redirect&bpo=27922]: Stop IDLE tests from “flashing” gui widgets on the screen.
- bpo-27891 [https://bugs.python.org/issue?@action=redirect&bpo=27891]: Consistently group and sort imports within idlelib modules.
- bpo-17642 [https://bugs.python.org/issue?@action=redirect&bpo=17642]: add larger font sizes for classroom projection.
- Add version to title of IDLE help window.
- bpo-25564 [https://bugs.python.org/issue?

@action=redirect&bpo=25564]: In section on IDLE – console differences, mention that using exec means that _builtins_ is defined for each statement.
- bpo-27821 [https://bugs.python.org/issue?@action=redirect&bpo=27821]: Fix 3.6.0a3 regression that prevented custom key sets from being selected when no custom theme was defined.

C API

- bpo-26900 [https://bugs.python.org/issue?@action=redirect&bpo=26900]: Excluded underscored names and other private API from limited API.
- bpo-26027 [https://bugs.python.org/issue?@action=redirect&bpo=26027]: Add support for path-like objects in PyUnicode_FSConverter() & PyUnicode_FSDecoder().

Tests

- bpo-27427 [https://bugs.python.org/issue?@action=redirect&bpo=27427]: Additional tests for the math module. Patch by Francisco Couzo.
- bpo-27953 [https://bugs.python.org/issue?@action=redirect&bpo=27953]: Skip math and cmath tests that fail on OS X 10.4 due to a poor libm implementation of tan.
- bpo-26040 [https://bugs.python.org/issue?@action=redirect&bpo=26040]: Improve test_math and test_cmath coverage and rigour. Patch by Jeff Allen.
- bpo-27787 [https://bugs.python.org/issue?@action=redirect&bpo=27787]: Call gc.collect() before checking each test for «dangling threads», since the dangling threads are weak references.

Build

- bpo-27566 [https://bugs.python.org/issue?@action=redirect&bpo=27566]: Fix clean target in freeze makefile (patch by Lisa Roach)
- bpo-27705 [https://bugs.python.org/issue?@action=redirect&bpo=27705]: Update message in

validate_ucrtbase.py

- bpo-27976 [https://bugs.python.org/issue?@action=redirect&bpo=27976]: Deprecate building _ctypes with the bundled copy of libffi on non-OSX UNIX platforms.
- bpo-27983 [https://bugs.python.org/issue?@action=redirect&bpo=27983]: Cause lack of llvm-profdata tool when using clang as required for PGO linking to be a configure time error rather than make time when `--with-optimizations` is enabled. Also improve our ability to find the llvm-profdata tool on MacOS and some Linuxes.
- bpo-21590 [https://bugs.python.org/issue?@action=redirect&bpo=21590]: Support for DTrace and SystemTap probes.
- bpo-26307 [https://bugs.python.org/issue?@action=redirect&bpo=26307]: The profile-opt build now applies PGO to the built-in modules.
- bpo-26359 [https://bugs.python.org/issue?@action=redirect&bpo=26359]: Add the –with-optimizations flag to turn on LTO and PGO build support when available.
- bpo-27917 [https://bugs.python.org/issue?@action=redirect&bpo=27917]: Set platform triplets for Android builds.
- bpo-25825 [https://bugs.python.org/issue?@action=redirect&bpo=25825]: Update references to the $(LIBPL) installation path on AIX. This path was changed in 3.2a4.
- Update OS X installer to use SQLite 3.14.1 and XZ 5.2.2.
- bpo-21122 [https://bugs.python.org/issue?@action=redirect&bpo=21122]: Fix LTO builds on OS X.
- bpo-17128 [https://bugs.python.org/issue?@action=redirect&bpo=17128]: Build OS X installer with a private copy of OpenSSL. Also provide a sample Install Certificates command script to install a set of root certificates from the third-party certifi module.

Tools/Demos

- bpo-27952 [https://bugs.python.org/issue?@action=redirect&bpo=27952]: Get Tools/scripts/fixcid.py working with Python 3 and the current «re» module, avoid invalid Python backslash escapes, and fix a bug parsing escaped C quote signs.

Windows

- bpo-28065 [https://bugs.python.org/issue?@action=redirect&bpo=28065]: Update xz dependency to 5.2.2 and build it from source.
- bpo-25144 [https://bugs.python.org/issue?@action=redirect&bpo=25144]: Ensures TargetDir is set before continuing with custom install.
- bpo-1602 [https://bugs.python.org/issue?@action=redirect&bpo=1602]: Windows console doesn't input or print Unicode (PEP 528)
- bpo-27781 [https://bugs.python.org/issue?@action=redirect&bpo=27781]: Change file system encoding on Windows to UTF-8 (PEP 529)
- bpo-27731 [https://bugs.python.org/issue?@action=redirect&bpo=27731]: Opt-out of MAX_PATH on Windows 10
- bpo-6135 [https://bugs.python.org/issue?@action=redirect&bpo=6135]: Adds encoding and errors parameters to subprocess.
- bpo-27959 [https://bugs.python.org/issue?@action=redirect&bpo=27959]: Adds oem encoding, alias ansi to mbcs, move aliasmbcs to codec lookup.
- bpo-27982 [https://bugs.python.org/issue?@action=redirect&bpo=27982]: The functions of the winsound module now accept keyword arguments.
- bpo-20366 [https://bugs.python.org/issue?@action=redirect&bpo=20366]: Build full text search support into SQLite on Windows.
- bpo-27756 [https://bugs.python.org/issue?@action=redirect&bpo=27756]: Adds new icons for Python files and processes on Windows. Designs by Cherry Wang.
- bpo-27883 [https://bugs.python.org/issue?@action=redirect&bpo=27883]: Update sqlite to 3.14.1.0 on Windows.

Python 3.6.0 alpha 4

Release date: 2016-08-15

Core and Builtins

- bpo-27704 [https://bugs.python.org/issue?@action=redirect&bpo=27704]: Optimized creating bytes and bytearray from byte-like objects and iterables. Speed up to 3 times for short objects. Original patch by Naoki Inada.
- bpo-26823 [https://bugs.python.org/issue?@action=redirect&bpo=26823]: Large sections of repeated lines in tracebacks are now abbreviated as «[Previous line repeated {count} more times]» by the builtin traceback rendering. Patch by Emanuel Barry.
- bpo-27574 [https://bugs.python.org/issue?@action=redirect&bpo=27574]: Decreased an overhead of parsing keyword arguments in functions implemented with using Argument Clinic.
- bpo-22557 [https://bugs.python.org/issue?@action=redirect&bpo=22557]: Now importing already imported modules is up to 2.5 times faster.
- bpo-17596 [https://bugs.python.org/issue?@action=redirect&bpo=17596]: Include <wincrypt.h> to help with Min GW building.
- bpo-17599 [https://bugs.python.org/issue?@action=redirect&bpo=17599]: On Windows, rename the privately defined REPARSE_DATA_BUFFER structure to avoid conflicting with the definition from Min GW.
- bpo-27507 [https://bugs.python.org/issue?@action=redirect&bpo=27507]: Add integer overflow check in bytearray.extend(). Patch by Xiang Zhang.
- bpo-27581 [https://bugs.python.org/issue?@action=redirect&bpo=27581]: Don’t rely on wrapping for overflow check in PySequence_Tuple(). Patch by Xiang Zhang.
- bpo-1621 [https://bugs.python.org/issue?@action=redirect&bpo=1621]: Avoid signed integer overflow in list and tuple operations. Patch by Xiang Zhang.
- bpo-27419 [https://bugs.python.org/issue?@action=redirect&bpo=27419]: Standard _import_() no longer look up «_import_» in globals or builtins for importing submodules or «from import». Fixed a crash if raise a warning about unabling to resolve package from _spec_ or _package_.

- bpo-27083 [https://bugs.python.org/issue?@action=redirect&bpo=27083]: Respect the PYTHONCASEOK environment variable under Windows.
- bpo-27514 [https://bugs.python.org/issue?@action=redirect&bpo=27514]: Make having too many statically nested blocks a SyntaxError instead of SystemError.
- bpo-27366 [https://bugs.python.org/issue?@action=redirect&bpo=27366]: Implemented **PEP 487** [https://peps.python.org/pep-0487/] (Simpler customization of class creation). Upon subclassing, the _init_subclass_ classmethod is called on the base class. Descriptors are initialized with _set_name_ after class creation.

Library

- bpo-26027 [https://bugs.python.org/issue?@action=redirect&bpo=26027]: Add **PEP 519** [https://peps.python.org/pep-0519/]/_fspath_() support to the os and os.path modules. Includes code from Jelle Zijlstra. (See also: bpo-27524 [https://bugs.python.org/issue?@action=redirect&bpo=27524])
- bpo-27598 [https://bugs.python.org/issue?@action=redirect&bpo=27598]: Add Collections to collections.abc. Patch by Ivan Levkivskyi, docs by Neil Girdhar.
- bpo-25958 [https://bugs.python.org/issue?@action=redirect&bpo=25958]: Support «anti-registration» of special methods from various ABCs, like _hash_, _iter_ or _len_. All these (and several more) can be set to None in an implementation class and the behavior will be as if the method is not defined at all. (Previously, this mechanism existed only for _hash_, to make mutable classes unhashable.) Code contributed by Andrew Barnert and Ivan Levkivskyi.
- bpo-16764 [https://bugs.python.org/issue?@action=redirect&bpo=16764]: Support keyword arguments to zlib.decompress(). Patch by Xiang Zhang.
- bpo-27736 [https://bugs.python.org/issue?@action=redirect&bpo=27736]: Prevent segfault after interpreter re-initialization due to ref count problem introduced in code for bpo-27038 [https://bugs.python.org/issue?

@action=redirect&bpo=27038] in 3.6.0a3. Patch by Xiang Zhang.

- bpo-25628 [https://bugs.python.org/issue?@action=redirect&bpo=25628]: The *verbose* and *rename* parameters for collections.namedtuple are now keyword-only.
- bpo-12345 [https://bugs.python.org/issue?@action=redirect&bpo=12345]: Add mathematical constant tau to math and cmath. See also **PEP 628** [https://peps.python.org/pep-0628/].
- bpo-26823 [https://bugs.python.org/issue?@action=redirect&bpo=26823]: traceback.StackSummary.format now abbreviates large sections of repeated lines as «[Previous line repeated {count} more times]» (this change then further affects other traceback display operations in the module). Patch by Emanuel Barry.
- bpo-27664 [https://bugs.python.org/issue?@action=redirect&bpo=27664]: Add to concurrent.futures.thread.ThreadPoolExecutor() the ability to specify a thread name prefix.
- bpo-27181 [https://bugs.python.org/issue?@action=redirect&bpo=27181]: Add geometric_mean and harmonic_mean to statistics module.
- bpo-27573 [https://bugs.python.org/issue?@action=redirect&bpo=27573]: code.interact now prints an message when exiting.
- bpo-6422 [https://bugs.python.org/issue?@action=redirect&bpo=6422]: Add autorange method to timeit.Timer objects.
- bpo-27773 [https://bugs.python.org/issue?@action=redirect&bpo=27773]: Correct some memory management errors server_hostname in _ssl.wrap_socket().
- bpo-26750 [https://bugs.python.org/issue?@action=redirect&bpo=26750]: unittest.mock.create_autospec() now works properly for subclasses of property() and other data descriptors. Removes the never publicly used, never documented unittest.mock.DescriptorTypes tuple.
- bpo-26754 [https://bugs.python.org/issue?@action=redirect&bpo=26754]: Undocumented support of general bytes-like objects as path in compile() and similar functions is now deprecated.
- bpo-26800 [https://bugs.python.org/issue?@action=redirect&bpo=26800]: Undocumented support of general

bytes-like objects as paths in os functions is now deprecated.

- bpo-26981 [https://bugs.python.org/issue?@action=redirect&bpo=26981]: Add _order_ compatibility shim to enum.Enum for Python 2/3 code bases.
- bpo-27661 [https://bugs.python.org/issue?@action=redirect&bpo=27661]: Added tzinfo keyword argument to datetime.combine.
- In the curses module, raise an error if window.getstr() or window.instr() is passed a negative value.
- bpo-27783 [https://bugs.python.org/issue?@action=redirect&bpo=27783]: Fix possible usage of uninitialized memory in operator.methodcaller.
- bpo-27774 [https://bugs.python.org/issue?@action=redirect&bpo=27774]: Fix possible Py_DECREF on unowned object in _sre.
- bpo-27760 [https://bugs.python.org/issue?@action=redirect&bpo=27760]: Fix possible integer overflow in binascii.b2a_qp.
- bpo-27758 [https://bugs.python.org/issue?@action=redirect&bpo=27758]: Fix possible integer overflow in the _csv module for large record lengths.
- bpo-27568 [https://bugs.python.org/issue?@action=redirect&bpo=27568]: Prevent HTTPoxy attack (**CVE 2016-1000110** [https://www.cve.org/CVERecord?id=CVE-2016-1000110]). Ignore the HTTP_PROXY variable when REQUEST_METHOD environment is set, which indicates that the script is in CGI mode.
- bpo-7063 [https://bugs.python.org/issue?@action=redirect&bpo=7063]: Remove dead code from the «array» module's slice handling. Patch by Chuck.
- bpo-27656 [https://bugs.python.org/issue?@action=redirect&bpo=27656]: Do not assume sched.h defines any SCHED_* constants.
- bpo-27130 [https://bugs.python.org/issue?@action=redirect&bpo=27130]: In the «zlib» module, fix handling of large buffers (typically 4 GiB) when compressing and decompressing. Previously, inputs were limited to 4 GiB, and compression and decompression operations did not properly handle results of 4 GiB.
- bpo-24773 [https://bugs.python.org/issue?

@action=redirect&bpo=24773]: Implemented **PEP 495** [https://peps.python.org/pep-0495/] (Local Time Disambiguation).
- Expose the EPOLLEXCLUSIVE constant (when it is defined) in the select module.
- bpo-27567 [https://bugs.python.org/issue?@action=redirect&bpo=27567]: Expose the EPOLLRDHUP and POLLRDHUP constants in the select module.
- bpo-1621 [https://bugs.python.org/issue?@action=redirect&bpo=1621]: Avoid signed int negation overflow in the «audioop» module.
- bpo-27533 [https://bugs.python.org/issue?@action=redirect&bpo=27533]: Release GIL in nt._isdir
- bpo-17711 [https://bugs.python.org/issue?@action=redirect&bpo=17711]: Fixed unpickling by the persistent ID with protocol 0. Original patch by Alexandre Vassalotti.
- bpo-27522 [https://bugs.python.org/issue?@action=redirect&bpo=27522]: Avoid an unintentional reference cycle in email.feedparser.
- bpo-27512 [https://bugs.python.org/issue?@action=redirect&bpo=27512]: Fix a segfault when os.fspath() called an __fspath__() method that raised an exception. Patch by Xiang Zhang.

IDLE

- bpo-27714 [https://bugs.python.org/issue?@action=redirect&bpo=27714]: text_textview and test_autocomplete now pass when re-run in the same process. This occurs when test_idle fails when run with the -w option but without -jn. Fix warning from test_config.
- bpo-27621 [https://bugs.python.org/issue?@action=redirect&bpo=27621]: Put query response validation error messages in the query box itself instead of in a separate messagebox. Redo tests to match. Add Mac OSX refinements. Original patch by Mark Roseman.
- bpo-27620 [https://bugs.python.org/issue?@action=redirect&bpo=27620]: Escape key now closes Query box as cancelled.
- bpo-27609 [https://bugs.python.org/issue?@action=redirect&bpo=27609]: IDLE: tab after initial whitespace should tab, not autocomplete. This fixes problem with writing

docstrings at least twice indented.

- bpo-27609 [https://bugs.python.org/issue?@action=redirect&bpo=27609]: Explicitly return None when there are also non-None returns. In a few cases, reverse a condition and eliminate a return.
- bpo-25507 [https://bugs.python.org/issue?@action=redirect&bpo=25507]: IDLE no longer runs buggy code because of its tkinter imports. Users must include the same imports required to run directly in Python.
- bpo-27173 [https://bugs.python.org/issue?@action=redirect&bpo=27173]: Add “IDLE Modern Unix” to the built-in key sets. Make the default key set depend on the platform. Add tests for the changes to the config module.
- bpo-27452 [https://bugs.python.org/issue?@action=redirect&bpo=27452]: add line counter and crc to IDLE configHandler test dump.

Tests

- bpo-25805 [https://bugs.python.org/issue?@action=redirect&bpo=25805]: Skip a test in test_pkgutil as needed that doesn’t work when `__name__ == __main__`. Patch by SilentGhost.
- bpo-27472 [https://bugs.python.org/issue?@action=redirect&bpo=27472]: Add test.support.unix_shell as the path to the default shell.
- bpo-27369 [https://bugs.python.org/issue?@action=redirect&bpo=27369]: In test_pyexpat, avoid testing an error message detail that changed in Expat 2.2.0.
- bpo-27594 [https://bugs.python.org/issue?@action=redirect&bpo=27594]: Prevent assertion error when running test_ast with coverage enabled: ensure code object has a valid first line number. Patch suggested by Ivan Levkivskyi.

Windows

- bpo-27647 [https://bugs.python.org/issue?@action=redirect&bpo=27647]: Update bundled Tcl/Tk to 8.6.6.
- bpo-27610 [https://bugs.python.org/issue?

@action=redirect&bpo=27610]: Adds **PEP 514** [https://peps.python.org/pep-0514/] metadata to Windows installer
- bpo-27469 [https://bugs.python.org/issue?@action=redirect&bpo=27469]: Adds a shell extension to the launcher so that drag and drop works correctly.
- bpo-27309 [https://bugs.python.org/issue?@action=redirect&bpo=27309]: Enables proper Windows styles in python[w].exe manifest.

Build

- bpo-27713 [https://bugs.python.org/issue?@action=redirect&bpo=27713]: Suppress spurious build warnings when updating importlib's bootstrap files. Patch by Xiang Zhang
- bpo-25825 [https://bugs.python.org/issue?@action=redirect&bpo=25825]: Correct the references to Modules/python.exp, which is required on AIX. The references were accidentally changed in 3.5.0a1.
- bpo-27453 [https://bugs.python.org/issue?@action=redirect&bpo=27453]: CPP invocation in configure must use CPPFLAGS. Patch by Chi Hsuan Yen.
- bpo-27641 [https://bugs.python.org/issue?@action=redirect&bpo=27641]: The configure script now inserts comments into the makefile to prevent the pgen and _freeze_importlib executables from being cross-compiled.
- bpo-26662 [https://bugs.python.org/issue?@action=redirect&bpo=26662]: Set PYTHON_FOR_GEN in configure as the Python program to be used for file generation during the build.
- bpo-10910 [https://bugs.python.org/issue?@action=redirect&bpo=10910]: Avoid C++ compilation errors on FreeBSD and OS X. Also update FreedBSD version checks for the original ctype UTF-8 workaround.

Python 3.6.0 alpha 3

Release date: 2016-07-11

Security

- bpo-27278 [https://bugs.python.org/issue?@action=redirect&bpo=27278]: Fix os.urandom() implementation using getrandom() on Linux. Truncate size to INT_MAX and loop until we collected enough random bytes, instead of casting a directly Py_ssize_t to int.
- bpo-22636 [https://bugs.python.org/issue?@action=redirect&bpo=22636]: Avoid shell injection problems with ctypes.util.find_library().

Core and Builtins

- bpo-27473 [https://bugs.python.org/issue?@action=redirect&bpo=27473]: Fixed possible integer overflow in bytes and bytearray concatenations. Patch by Xiang Zhang.
- bpo-23034 [https://bugs.python.org/issue?@action=redirect&bpo=23034]: The output of a special Python build with defined COUNT_ALLOCS, SHOW_ALLOC_COUNT or SHOW_TRACK_COUNT macros is now off by default. It can be re-enabled using the «-X showalloccount» option. It now outputs to stderr instead of stdout.
- bpo-27443 [https://bugs.python.org/issue?@action=redirect&bpo=27443]: __length_hint__() of bytearray iterators no longer return a negative integer for a resized bytearray.
- bpo-27007 [https://bugs.python.org/issue?@action=redirect&bpo=27007]: The fromhex() class methods of bytes and bytearray subclasses now return an instance of corresponding subclass.

Library

- bpo-26844 [https://bugs.python.org/issue?@action=redirect&bpo=26844]: Fix error message for imp.find_module() to refer to “path” instead of “name”. Patch by Lev Maximov.
- bpo-23804 [https://bugs.python.org/issue?@action=redirect&bpo=23804]: Fix SSL zero-length recv() calls to not block and not raise an error about unclean EOF.

- bpo-27466 [https://bugs.python.org/issue?@action=redirect&bpo=27466]: Change time format returned by http.cookie.time2netscape, confirming the netscape cookie format and making it consistent with documentation.
- bpo-21708 [https://bugs.python.org/issue?@action=redirect&bpo=21708]: Deprecated dbm.dumb behavior that differs from common dbm behavior: creating a database in "r" and "w" modes and modifying a database in "r" mode.
- bpo-26721 [https://bugs.python.org/issue?@action=redirect&bpo=26721]: Change the socketserver.StreamRequestHandler.wfile attribute to implement BufferedIOBase. In particular, the write() method no longer does partial writes.
- bpo-22115 [https://bugs.python.org/issue?@action=redirect&bpo=22115]: Added methods trace_add, trace_remove and trace_info in the tkinter.Variable class. They replace old methods trace_variable, trace, trace_vdelete and trace_vinfo that use obsolete Tcl commands and might not work in future versions of Tcl. Fixed old tracing methods: trace_vdelete() with wrong mode no longer break tracing, trace_vinfo() now always returns a list of pairs of strings, tracing in the «u» mode now works.
- bpo-26243 [https://bugs.python.org/issue?@action=redirect&bpo=26243]: Only the level argument to zlib.compress() is keyword argument now. The first argument is positional-only.
- bpo-27038 [https://bugs.python.org/issue?@action=redirect&bpo=27038]: Expose the DirEntry type as os.DirEntry. Code patch by Jelle Zijlstra.
- bpo-27186 [https://bugs.python.org/issue?@action=redirect&bpo=27186]: Update os.fspath()/PyOS_FSPath() to check the return value of __fspath__() to be either str or bytes.
- bpo-18726 [https://bugs.python.org/issue?@action=redirect&bpo=18726]: All optional parameters of the dump(), dumps(), load() and loads() functions and JSONEncoder and JSONDecoder class constructors in the json module are now keyword-only.
- bpo-27319 [https://bugs.python.org/issue?@action=redirect&bpo=27319]: Methods selection_set(),

selection_add(), selection_remove() and selection_toggle() of ttk.TreeView now allow passing multiple items as multiple arguments instead of passing them as a tuple. Deprecated undocumented ability of calling the selection() method with arguments.

- bpo-27079 [https://bugs.python.org/issue?@action=redirect&bpo=27079]: Fixed curses.ascii functions isblank(), iscntrl() and ispunct().
- bpo-27294 [https://bugs.python.org/issue?@action=redirect&bpo=27294]: Numerical state in the repr for Tkinter event objects is now represented as a combination of known flags.
- bpo-27177 [https://bugs.python.org/issue?@action=redirect&bpo=27177]: Match objects in the re module now support index-like objects as group indices. Based on patches by Jeroen Demeyer and Xiang Zhang.
- bpo-26754 [https://bugs.python.org/issue?@action=redirect&bpo=26754]: Some functions (compile() etc) accepted a filename argument encoded as an iterable of integers. Now only strings and byte-like objects are accepted.
- bpo-26536 [https://bugs.python.org/issue?@action=redirect&bpo=26536]: socket.ioctl now supports SIO_LOOPBACK_FAST_PATH. Patch by Daniel Stokes.
- bpo-27048 [https://bugs.python.org/issue?@action=redirect&bpo=27048]: Prevents distutils failing on Windows when environment variables contain non-ASCII characters
- bpo-27330 [https://bugs.python.org/issue?@action=redirect&bpo=27330]: Fixed possible leaks in the ctypes module.
- bpo-27238 [https://bugs.python.org/issue?@action=redirect&bpo=27238]: Got rid of bare excepts in the turtle module. Original patch by Jelle Zijlstra.
- bpo-27122 [https://bugs.python.org/issue?@action=redirect&bpo=27122]: When an exception is raised within the context being managed by a contextlib.ExitStack() and one of the exit stack generators catches and raises it in a chain, do not re-raise the original exception when exiting, let the new chained one through. This avoids the **PEP 479** [https://peps.python.org/pep-0479/] bug described in issue25782.

- bpo-16864 [https://bugs.python.org/issue?@action=redirect&bpo=16864]: sqlite3.Cursor.lastrowid now supports REPLACE statement. Initial patch by Alex LordThorsen.
- bpo-26386 [https://bugs.python.org/issue?@action=redirect&bpo=26386]: Fixed ttk.TreeView selection operations with item id’s containing spaces.
- bpo-8637 [https://bugs.python.org/issue?@action=redirect&bpo=8637]: Honor a pager set by the env var MANPAGER (in preference to one set by the env var PAGER).
- bpo-16182 [https://bugs.python.org/issue?@action=redirect&bpo=16182]: Fix various functions in the «readline» module to use the locale encoding, and fix get_begidx() and get_endidx() to return code point indexes.
- bpo-27392 [https://bugs.python.org/issue?@action=redirect&bpo=27392]: Add loop.connect_accepted_socket(). Patch by Jim Fulton.

IDLE

- bpo-27477 [https://bugs.python.org/issue?@action=redirect&bpo=27477]: IDLE search dialogs now use ttk widgets.
- bpo-27173 [https://bugs.python.org/issue?@action=redirect&bpo=27173]: Add “IDLE Modern Unix” to the built-in key sets. Make the default key set depend on the platform. Add tests for the changes to the config module.
- bpo-27452 [https://bugs.python.org/issue?@action=redirect&bpo=27452]: make command line «idle-test> python test_help.py» work. _file_ is relative when python is started in the file’s directory.
- bpo-27452 [https://bugs.python.org/issue?@action=redirect&bpo=27452]: add line counter and crc to IDLE configHandler test dump.
- bpo-27380 [https://bugs.python.org/issue?@action=redirect&bpo=27380]: IDLE: add query.py with base Query dialog and ttk widgets. Module had subclasses SectionName, ModuleName, and HelpSource, which are used to get information from users by configdialog and file =>Load Module. Each subclass has itw own validity checks.

Using ModuleName allows users to edit bad module names instead of starting over. Add tests and delete the two files combined into the new one.

- bpo-27372 [https://bugs.python.org/issue?@action=redirect&bpo=27372]: Test_idle no longer changes the locale.
- bpo-27365 [https://bugs.python.org/issue?@action=redirect&bpo=27365]: Allow non-ascii chars in IDLE NEWS.txt, for contributor names.
- bpo-27245 [https://bugs.python.org/issue?@action=redirect&bpo=27245]: IDLE: Cleanly delete custom themes and key bindings. Previously, when IDLE was started from a console or by import, a cascade of warnings was emitted. Patch by Serhiy Storchaka.
- bpo-24137 [https://bugs.python.org/issue?@action=redirect&bpo=24137]: Run IDLE, test_idle, and htest with tkinter default root disabled. Fix code and tests that fail with this restriction. Fix htests to not create a second and redundant root and mainloop.
- bpo-27310 [https://bugs.python.org/issue?@action=redirect&bpo=27310]: Fix IDLE.app failure to launch on OS X due to vestigial import.

C API

- bpo-26754 [https://bugs.python.org/issue?@action=redirect&bpo=26754]: PyUnicode_FSDecoder() accepted a filename argument encoded as an iterable of integers. Now only strings and byte-like objects are accepted.

Build

- bpo-28066 [https://bugs.python.org/issue?@action=redirect&bpo=28066]: Fix the logic that searches build directories for generated include files when building outside the source tree.
- bpo-27442 [https://bugs.python.org/issue?@action=redirect&bpo=27442]: Expose the Android API level that python was built against, in sysconfig.get_config_vars() as “ANDROID_API_LEVEL”.

- bpo-27434 [https://bugs.python.org/issue?@action=redirect&bpo=27434]: The interpreter that runs the cross-build, found in PATH, must now be of the same feature version (e.g. 3.6) as the source being built.
- bpo-26930 [https://bugs.python.org/issue?@action=redirect&bpo=26930]: Update Windows builds to use OpenSSL 1.0.2h.
- bpo-23968 [https://bugs.python.org/issue?@action=redirect&bpo=23968]: Rename the platform directory from plat-$(MACHDEP) to plat-$(PLATFORM_TRIPLET). Rename the config directory (LIBPL) from config-$(LDVERSION) to config-$(LDVERSION)-$(PLATFORM_TRIPLET). Install the platform specific _sysconfigdata module into the platform directory and rename it to include the ABIFLAGS.
- Don't use largefile support for GNU/Hurd.

Tools/Demos

- bpo-27332 [https://bugs.python.org/issue?@action=redirect&bpo=27332]: Fixed the type of the first argument of module-level functions generated by Argument Clinic. Patch by Petr Viktorin.
- bpo-27418 [https://bugs.python.org/issue?@action=redirect&bpo=27418]: Fixed Tools/importbench/importbench.py.

Documentation

- bpo-19489 [https://bugs.python.org/issue?@action=redirect&bpo=19489]: Moved the search box from the sidebar to the header and footer of each page. Patch by Ammar Askar.
- bpo-27285 [https://bugs.python.org/issue?@action=redirect&bpo=27285]: Update documentation to reflect the deprecation of `pyvenv` and normalize on the term «virtual environment». Patch by Steve Piercy.

Tests

- bpo-27027 [https://bugs.python.org/issue?@action=redirect&bpo=27027]: Added test.support.is_android that is True when this is an Android build.

Python 3.6.0 alpha 2

Release date: 2016-06-13

Security

- bpo-26556 [https://bugs.python.org/issue?@action=redirect&bpo=26556]: Update expat to 2.1.1, fixes **CVE 2015-1283** [https://www.cve.org/CVERecord?id=CVE-2015-1283].
- Fix TLS stripping vulnerability in smtplib, **CVE 2016-0772** [https://www.cve.org/CVERecord?id=CVE-2016-0772]. Reported by Team Oststrom.
- bpo-26839 [https://bugs.python.org/issue?@action=redirect&bpo=26839]: On Linux, **`os.urandom()`** now calls `getrandom()` with `GRND_NONBLOCK` to fall back on reading `/dev/urandom` if the urandom entropy pool is not initialized yet. Patch written by Colm Buckley.

Core and Builtins

- bpo-27095 [https://bugs.python.org/issue?@action=redirect&bpo=27095]: Simplified MAKE_FUNCTION and removed MAKE_CLOSURE opcodes. Patch by Demur Rumed.
- bpo-27190 [https://bugs.python.org/issue?@action=redirect&bpo=27190]: Raise NotSupportedError if sqlite3 is older than 3.3.1. Patch by Dave Sawyer.
- bpo-27286 [https://bugs.python.org/issue?@action=redirect&bpo=27286]: Fixed compiling BUILD_MAP_UNPACK_WITH_CALL opcode. Calling function with generalized unpacking (PEP 448) and conflicting keyword names could cause undefined behavior.
- bpo-27140 [https://bugs.python.org/issue?@action=redirect&bpo=27140]: Added BUILD_CONST_KEY_MAP opcode.
- bpo-27186 [https://bugs.python.org/issue?

@action=redirect&bpo=27186]: Add support for os.PathLike objects to open() (part of **PEP 519** [https://peps.python.org/pep-0519/]).

- bpo-27066 [https://bugs.python.org/issue?@action=redirect&bpo=27066]: Fixed SystemError if a custom opener (for open()) returns a negative number without setting an exception.
- bpo-26983 [https://bugs.python.org/issue?@action=redirect&bpo=26983]: float() now always return an instance of exact float. The deprecation warning is emitted if _float_ returns an instance of a strict subclass of float. In a future versions of Python this can be an error.
- bpo-27097 [https://bugs.python.org/issue?@action=redirect&bpo=27097]: Python interpreter is now about 7% faster due to optimized instruction decoding. Based on patch by Demur Rumed.
- bpo-26647 [https://bugs.python.org/issue?@action=redirect&bpo=26647]: Python interpreter now uses 16-bit wordcode instead of bytecode. Patch by Demur Rumed.
- bpo-23275 [https://bugs.python.org/issue?@action=redirect&bpo=23275]: Allow assigning to an empty target list in round brackets: () = iterable.
- bpo-27243 [https://bugs.python.org/issue?@action=redirect&bpo=27243]: Update the _aiter_ protocol: instead of returning an awaitable that resolves to an asynchronous iterator, the asynchronous iterator should be returned directly. Doing the former will trigger a PendingDeprecationWarning.

Library

- Comment out socket (SO_REUSEPORT) and posix (O_SHLOCK, O_EXLOCK) constants exposed on the API which are not implemented on GNU/Hurd. They would not work at runtime anyway.
- bpo-27025 [https://bugs.python.org/issue?@action=redirect&bpo=27025]: Generated names for Tkinter widgets are now more meaningful and recognizable.
- bpo-25455 [https://bugs.python.org/issue?@action=redirect&bpo=25455]: Fixed crashes in repr of recursive

ElementTree.Element and functools.partial objects.

- bpo-27294 [https://bugs.python.org/issue?@action=redirect&bpo=27294]: Improved repr for Tkinter event objects.
- bpo-20508 [https://bugs.python.org/issue?@action=redirect&bpo=20508]: Improve exception message of IPv{4,6}Network.__getitem__. Patch by Gareth Rees.
- bpo-21386 [https://bugs.python.org/issue?@action=redirect&bpo=21386]: Implement missing IPv4Address.is_global property. It was documented since 07a5610bae9d. Initial patch by Roger Luethi.
- bpo-27029 [https://bugs.python.org/issue?@action=redirect&bpo=27029]: Removed deprecated support of universal newlines mode from ZipFile.open().
- bpo-27030 [https://bugs.python.org/issue?@action=redirect&bpo=27030]: Unknown escapes consisting of `'\'` and an ASCII letter in regular expressions now are errors. The re.LOCALE flag now can be used only with bytes patterns.
- bpo-27186 [https://bugs.python.org/issue?@action=redirect&bpo=27186]: Add os.PathLike support to DirEntry (part of **PEP 519** [https://peps.python.org/pep-0519/]). Initial patch by Jelle Zijlstra.
- bpo-20900 [https://bugs.python.org/issue?@action=redirect&bpo=20900]: distutils register command now decodes HTTP responses correctly. Initial patch by ingrid.
- bpo-27186 [https://bugs.python.org/issue?@action=redirect&bpo=27186]: Add os.PathLike support to pathlib, removing its provisional status (part of PEP 519). Initial patch by Dusty Phillips.
- bpo-27186 [https://bugs.python.org/issue?@action=redirect&bpo=27186]: Add support for os.PathLike objects to os.fsencode() and os.fsdecode() (part of **PEP 519** [https://peps.python.org/pep-0519/]).
- bpo-27186 [https://bugs.python.org/issue?@action=redirect&bpo=27186]: Introduce os.PathLike and os.fspath() (part of **PEP 519** [https://peps.python.org/pep-0519/]).
- A new version of typing.py provides several new classes and features: @overload outside stubs, Reversible, DefaultDict, Text, ContextManager, Type[], NewType(), TYPE_CHECKING,

and numerous bug fixes (note that some of the new features are not yet implemented in mypy or other static analyzers). Also classes for **PEP 492** [https://peps.python.org/pep-0492/] (Awaitable, AsyncIterable, AsyncIterator) have been added (in fact they made it into 3.5.1 but were never mentioned).

- bpo-25738 [https://bugs.python.org/issue?@action=redirect&bpo=25738]: Stop http.server.BaseHTTPRequestHandler.send_error() from sending a message body for 205 Reset Content. Also, don't send Content header fields in responses that don't have a body. Patch by Susumu Koshiba.
- bpo-21313 [https://bugs.python.org/issue?@action=redirect&bpo=21313]: Fix the «platform» module to tolerate when sys.version contains truncated build information.
- bpo-23883 [https://bugs.python.org/issue?@action=redirect&bpo=23883]: Added missing APIs to __all__ to match the documented APIs for the following modules: cgi, mailbox, mimetypes, plistlib and smtpd. Patches by Jacek Kołodziej.
- bpo-27164 [https://bugs.python.org/issue?@action=redirect&bpo=27164]: In the zlib module, allow decompressing raw Deflate streams with a predefined zdict. Based on patch by Xiang Zhang.
- bpo-24291 [https://bugs.python.org/issue?@action=redirect&bpo=24291]: Fix wsgiref.simple_server.WSGIRequestHandler to completely write data to the client. Previously it could do partial writes and truncate data. Also, wsgiref.handler.ServerHandler can now handle stdout doing partial writes, but this is deprecated.
- bpo-21272 [https://bugs.python.org/issue?@action=redirect&bpo=21272]: Use _sysconfigdata.py to initialize distutils.sysconfig.
- bpo-19611 [https://bugs.python.org/issue?@action=redirect&bpo=19611]: **`inspect`** now reports the implicit `.0` parameters generated by the compiler for comprehension and generator expression scopes as if they were positional-only parameters called `implicit0`. Patch by Jelle Zijlstra.
- bpo-26809 [https://bugs.python.org/issue?

@action=redirect&bpo=26809]: Add `__all__` to `string`. Patch by Emanuel Barry.
- bpo-26373 [https://bugs.python.org/issue?@action=redirect&bpo=26373]: subprocess.Popen.communicate now correctly ignores BrokenPipeError when the child process dies before .communicate() is called in more/all circumstances.
- signal, socket, and ssl module IntEnum constant name lookups now return a consistent name for values having multiple names. Ex: signal.Signals(6) now refers to itself as signal.SIGALRM rather than flipping between that and signal.SIGIOT based on the interpreter's hash randomization seed.
- bpo-27167 [https://bugs.python.org/issue?@action=redirect&bpo=27167]: Clarify the subprocess.CalledProcessError error message text when the child process died due to a signal.
- bpo-25931 [https://bugs.python.org/issue?@action=redirect&bpo=25931]: Don't define socketserver.Forking* names on platforms such as Windows that do not support os.fork().
- bpo-21776 [https://bugs.python.org/issue?@action=redirect&bpo=21776]: distutils.upload now correctly handles HTTPError. Initial patch by Claudiu Popa.
- bpo-26526 [https://bugs.python.org/issue?@action=redirect&bpo=26526]: Replace custom parse tree validation in the parser module with a simple DFA validator.
- bpo-27114 [https://bugs.python.org/issue?@action=redirect&bpo=27114]: Fix SSLContext._load_windows_store_certs fails with PermissionError
- bpo-18383 [https://bugs.python.org/issue?@action=redirect&bpo=18383]: Avoid creating duplicate filters when using filterwarnings and simplefilter. Based on patch by Alex Shkop.
- bpo-23026 [https://bugs.python.org/issue?@action=redirect&bpo=23026]: winreg.QueryValueEx() now return an integer for REG_QWORD type.
- bpo-26741 [https://bugs.python.org/issue?@action=redirect&bpo=26741]: subprocess.Popen destructor now

emits a ResourceWarning warning if the child process is still running.

- bpo-27056 [https://bugs.python.org/issue?@action=redirect&bpo=27056]: Optimize pickle.load() and pickle.loads(), up to 10% faster to deserialize a lot of small objects.
- bpo-21271 [https://bugs.python.org/issue?@action=redirect&bpo=21271]: New keyword only parameters in reset_mock call.

IDLE

- bpo-5124 [https://bugs.python.org/issue?@action=redirect&bpo=5124]: Paste with text selected now replaces the selection on X11. This matches how paste works on Windows, Mac, most modern Linux apps, and ttk widgets. Original patch by Serhiy Storchaka.
- bpo-24750 [https://bugs.python.org/issue?@action=redirect&bpo=24750]: Switch all scrollbars in IDLE to ttk versions. Where needed, minimal tests are added to cover changes.
- bpo-24759 [https://bugs.python.org/issue?@action=redirect&bpo=24759]: IDLE requires tk 8.5 and availability ttk widgets. Delete now unneeded tk version tests and code for older versions. Add test for IDLE syntax colorizer.
- bpo-27239 [https://bugs.python.org/issue?@action=redirect&bpo=27239]: idlelib.macosx.isXyzTk functions initialize as needed.
- bpo-27262 [https://bugs.python.org/issue?@action=redirect&bpo=27262]: move Aqua unbinding code, which enable context menus, to macosx.
- bpo-24759 [https://bugs.python.org/issue?@action=redirect&bpo=24759]: Make clear in idlelib.idle_test._init_ that the directory is a private implementation of test.test_idle and tool for maintainers.
- bpo-27196 [https://bugs.python.org/issue?@action=redirect&bpo=27196]: Stop “ThemeChanged” warnings when running IDLE tests. These persisted after other warnings were suppressed in #20567. Apply Serhiy Storchaka’s

update_idletasks solution to four test files. Record this additional advice in idle_test/README.txt

- bpo-20567 [https://bugs.python.org/issue?@action=redirect&bpo=20567]: Revise idle_test/README.txt with advice about avoiding tk warning messages from tests. Apply advice to several IDLE tests.
- bpo-24225 [https://bugs.python.org/issue?@action=redirect&bpo=24225]: Update idlelib/README.txt with new file names and event handlers.
- bpo-27156 [https://bugs.python.org/issue?@action=redirect&bpo=27156]: Remove obsolete code not used by IDLE.
- bpo-27117 [https://bugs.python.org/issue?@action=redirect&bpo=27117]: Make colorizer htest and turtledemo work with dark themes. Move code for configuring text widget colors to a new function.
- bpo-24225 [https://bugs.python.org/issue?@action=redirect&bpo=24225]: Rename many `idlelib/*.py` and `idle_test/test_*.py` files. Edit files to replace old names with new names when the old name referred to the module rather than the class it contained. See the issue and IDLE section in What’s New in 3.6 for more.
- bpo-26673 [https://bugs.python.org/issue?@action=redirect&bpo=26673]: When tk reports font size as 0, change to size 10. Such fonts on Linux prevented the configuration dialog from opening.
- bpo-21939 [https://bugs.python.org/issue?@action=redirect&bpo=21939]: Add test for IDLE’s percolator. Original patch by Saimadhav Heblikar.
- bpo-21676 [https://bugs.python.org/issue?@action=redirect&bpo=21676]: Add test for IDLE’s replace dialog. Original patch by Saimadhav Heblikar.
- bpo-18410 [https://bugs.python.org/issue?@action=redirect&bpo=18410]: Add test for IDLE’s search dialog. Original patch by Westley Martínez.
- bpo-21703 [https://bugs.python.org/issue?@action=redirect&bpo=21703]: Add test for undo delegator. Patch mostly by Saimadhav Heblikar .
- bpo-27044 [https://bugs.python.org/issue?@action=redirect&bpo=27044]: Add

ConfigDialog.remove_var_callbacks to stop memory leaks.

- bpo-23977 [https://bugs.python.org/issue?@action=redirect&bpo=23977]: Add more asserts to test_delegator.

Documentation

- bpo-16484 [https://bugs.python.org/issue?@action=redirect&bpo=16484]: Change the default PYTHONDOCS URL to «https:», and fix the resulting links to use lowercase. Patch by Sean Rodman, test by Kaushik Nadikuditi.
- bpo-24136 [https://bugs.python.org/issue?@action=redirect&bpo=24136]: Document the new **PEP 448** [https://peps.python.org/pep-0448/] unpacking syntax of 3.5.
- bpo-22558 [https://bugs.python.org/issue?@action=redirect&bpo=22558]: Add remaining doc links to source code for Python-coded modules. Patch by Yoni Lavi.

Tests

- bpo-25285 [https://bugs.python.org/issue?@action=redirect&bpo=25285]: regrtest now uses subprocesses when the -j1 command line option is used: each test file runs in a fresh child process. Before, the -j1 option was ignored.
- bpo-25285 [https://bugs.python.org/issue?@action=redirect&bpo=25285]: Tools/buildbot/test.bat script now uses -j1 by default to run each test file in fresh child process.

Windows

- bpo-27064 [https://bugs.python.org/issue?@action=redirect&bpo=27064]: The py.exe launcher now defaults to Python 3. The Windows launcher `py.exe` no longer prefers an installed Python 2 version over Python 3 by default when used interactively.
- bpo-17500 [https://bugs.python.org/issue?@action=redirect&bpo=17500]: Remove unused and outdated icons. (See also: https://github.com/python/pythondotorg/issues/945)

Build

- bpo-27229 [https://bugs.python.org/issue?@action=redirect&bpo=27229]: Fix the cross-compiling pgen rule for in-tree builds. Patch by Xavier de Gaye.
- bpo-26930 [https://bugs.python.org/issue?@action=redirect&bpo=26930]: Update OS X 10.5+ 32-bit-only installer to build and link with OpenSSL 1.0.2h.

C API

- bpo-27186 [https://bugs.python.org/issue?@action=redirect&bpo=27186]: Add the PyOS_FSPath() function (part of **PEP 519** [https://peps.python.org/pep-0519/]).
- bpo-26282 [https://bugs.python.org/issue?@action=redirect&bpo=26282]: PyArg_ParseTupleAndKeywords() now supports positional-only parameters.

Tools/Demos

- bpo-26282 [https://bugs.python.org/issue?@action=redirect&bpo=26282]: Argument Clinic now supports positional-only and keyword parameters in the same function.

Python 3.6.0 alpha 1

Release date: 2016-05-16

Security

- bpo-26657 [https://bugs.python.org/issue?@action=redirect&bpo=26657]: Fix directory traversal vulnerability with http.server on Windows. This fixes a regression that was introduced in 3.3.4rc1 and 3.4.0rc1. Based on patch by Philipp Hagemeister.
- bpo-26313 [https://bugs.python.org/issue?@action=redirect&bpo=26313]: ssl.py _load_windows_store_certs fails if windows cert store is empty. Patch by Baji.
- bpo-25939 [https://bugs.python.org/issue?

@action=redirect&bpo=25939]: On Windows open the cert store readonly in ssl.enum_certificates.

Core and Builtins

- bpo-20041 [https://bugs.python.org/issue?@action=redirect&bpo=20041]: Fixed TypeError when frame.f_trace is set to None. Patch by Xavier de Gaye.
- bpo-26168 [https://bugs.python.org/issue?@action=redirect&bpo=26168]: Fixed possible refleaks in failing Py_BuildValue() with the «N» format unit.
- bpo-26991 [https://bugs.python.org/issue?@action=redirect&bpo=26991]: Fix possible refleak when creating a function with annotations.
- bpo-27039 [https://bugs.python.org/issue?@action=redirect&bpo=27039]: Fixed bytearray.remove() for values greater than 127. Based on patch by Joe Jevnik.
- bpo-23640 [https://bugs.python.org/issue?@action=redirect&bpo=23640]: int.from_bytes() no longer bypasses constructors for subclasses.
- bpo-27005 [https://bugs.python.org/issue?@action=redirect&bpo=27005]: Optimized the float.fromhex() class method for exact float. It is now 2 times faster.
- bpo-18531 [https://bugs.python.org/issue?@action=redirect&bpo=18531]: Single var-keyword argument of dict subtype was passed unscathed to the C-defined function. Now it is converted to exact dict.
- bpo-26811 [https://bugs.python.org/issue?@action=redirect&bpo=26811]: gc.get_objects() no longer contains a broken tuple with NULL pointer.
- bpo-20120 [https://bugs.python.org/issue?@action=redirect&bpo=20120]: Use RawConfigParser for .pypirc parsing, removing support for interpolation unintentionally added with move to Python 3. Behavior no longer does any interpolation in .pypirc files, matching behavior in Python 2.7 and Setuptools 19.0.
- bpo-26249 [https://bugs.python.org/issue?@action=redirect&bpo=26249]: Memory functions of the `PyMem_Malloc()` domain (`PYMEM_DOMAIN_MEM`) now use the pymalloc allocator rather than system `malloc()`.

Applications calling **`PyMem_Malloc()`** without holding the GIL can now crash: use `PYTHONMALLOC=debug` environment variable to validate the usage of memory allocators in your application.

- bpo-26802 [https://bugs.python.org/issue?@action=redirect&bpo=26802]: Optimize function calls only using unpacking like `func(*tuple)` (no other positional argument, no keyword): avoid copying the tuple. Patch written by Joe Jevnik.
- bpo-26659 [https://bugs.python.org/issue?@action=redirect&bpo=26659]: Make the builtin slice type support cycle collection.
- bpo-26718 [https://bugs.python.org/issue?@action=redirect&bpo=26718]: super._init_ no longer leaks memory if called multiple times. NOTE: A direct call of super._init_ is not endorsed!
- bpo-27138 [https://bugs.python.org/issue?@action=redirect&bpo=27138]: Fix the doc comment for FileFinder.find_spec().
- bpo-27147 [https://bugs.python.org/issue?@action=redirect&bpo=27147]: Mention **PEP 420** [https://peps.python.org/pep-0420/] in the importlib docs.
- bpo-25339 [https://bugs.python.org/issue?@action=redirect&bpo=25339]: PYTHONIOENCODING now has priority over locale in setting the error handler for stdin and stdout.
- bpo-26494 [https://bugs.python.org/issue?@action=redirect&bpo=26494]: Fixed crash on iterating exhausting iterators. Affected classes are generic sequence iterators, iterators of str, bytes, bytearray, list, tuple, set, frozenset, dict, OrderedDict, corresponding views and os.scandir() iterator.
- bpo-26574 [https://bugs.python.org/issue?@action=redirect&bpo=26574]: Optimize `bytes.replace(b'', b'.')` and `bytearray.replace(b'', b'.')`. Patch written by Josh Snider.
- bpo-26581 [https://bugs.python.org/issue?@action=redirect&bpo=26581]: If coding cookie is specified multiple times on a line in Python source code file, only the

first one is taken to account.

- bpo-19711 [https://bugs.python.org/issue?@action=redirect&bpo=19711]: Add tests for reloading namespace packages.
- bpo-21099 [https://bugs.python.org/issue?@action=redirect&bpo=21099]: Switch applicable importlib tests to use **PEP 451** [https://peps.python.org/pep-0451/] API.
- bpo-26563 [https://bugs.python.org/issue?@action=redirect&bpo=26563]: Debug hooks on Python memory allocators now raise a fatal error if functions of the `PyMem_Malloc()` family are called without holding the GIL.
- bpo-26564 [https://bugs.python.org/issue?@action=redirect&bpo=26564]: On error, the debug hooks on Python memory allocators now use the `tracemalloc` module to get the traceback where a memory block was allocated.
- bpo-26558 [https://bugs.python.org/issue?@action=redirect&bpo=26558]: The debug hooks on Python memory allocator `PyObject_Malloc()` now detect when functions are called without holding the GIL.
- bpo-26516 [https://bugs.python.org/issue?@action=redirect&bpo=26516]: Add `PYTHONMALLOC` environment variable to set the Python memory allocators and/or install debug hooks.
- bpo-26516 [https://bugs.python.org/issue?@action=redirect&bpo=26516]: The `PyMem_SetupDebugHooks()` function can now also be used on Python compiled in release mode.
- bpo-26516 [https://bugs.python.org/issue?@action=redirect&bpo=26516]: The `PYTHONMALLOCSTATS` environment variable can now also be used on Python compiled in release mode. It now has no effect if set to an empty string.
- bpo-26516 [https://bugs.python.org/issue?@action=redirect&bpo=26516]: In debug mode, debug hooks are now also installed on Python memory allocators when Python is configured without pymalloc.
- bpo-26464 [https://bugs.python.org/issue?@action=redirect&bpo=26464]: Fix str.translate() when string is ASCII and first replacements removes character, but next

replacement uses a non-ASCII character or a string longer than 1 character. Regression introduced in Python 3.5.0.

- bpo-22836 [https://bugs.python.org/issue?@action=redirect&bpo=22836]: Ensure exception reports from PyErr_Display() and PyErr_WriteUnraisable() are sensible even when formatting them produces secondary errors. This affects the reports produced by sys.__excepthook__() and when __del__() raises an exception.
- bpo-26302 [https://bugs.python.org/issue?@action=redirect&bpo=26302]: Correct behavior to reject comma as a legal character for cookie names.
- bpo-26136 [https://bugs.python.org/issue?@action=redirect&bpo=26136]: Upgrade the warning when a generator raises StopIteration from PendingDeprecationWarning to DeprecationWarning. Patch by Anish Shah.
- bpo-26204 [https://bugs.python.org/issue?@action=redirect&bpo=26204]: The compiler now ignores all constant statements: bytes, str, int, float, complex, name constants (None, False, True), Ellipsis and ast.Constant; not only str and int. For example, `1.0` is now ignored in `def f(): 1.0`.
- bpo-4806 [https://bugs.python.org/issue?@action=redirect&bpo=4806]: Avoid masking the original TypeError exception when using star (*) unpacking in function calls. Based on patch by Hagen Fürstenau and Daniel Urban.
- bpo-26146 [https://bugs.python.org/issue?@action=redirect&bpo=26146]: Add a new kind of AST node: `ast.Constant`. It can be used by external AST optimizers, but the compiler does not emit directly such node.
- bpo-23601 [https://bugs.python.org/issue?@action=redirect&bpo=23601]: Sped-up allocation of dict key objects by using Python’s small object allocator. (Contributed by Julian Taylor.)
- bpo-18018 [https://bugs.python.org/issue?@action=redirect&bpo=18018]: Import raises ImportError instead of SystemError if a relative import is attempted without a known parent package.
- bpo-25843 [https://bugs.python.org/issue?@action=redirect&bpo=25843]: When compiling code, don’t

merge constants if they are equal but have a different types. For example, `f1, f2 = lambda: 1, lambda: 1.0` is now correctly compiled to two different functions: `f1()` returns `1` (`int`) and `f2()` returns `1.0` (`float`), even if `1` and `1.0` are equal.

- bpo-26107 [https://bugs.python.org/issue?@action=redirect&bpo=26107]: The format of the `co_lnotab` attribute of code objects changes to support negative line number delta.
- bpo-26154 [https://bugs.python.org/issue?@action=redirect&bpo=26154]: Add a new private _PyThreadState_UncheckedGet() function to get the current Python thread state, but don't issue a fatal error if it is NULL. This new function must be used instead of accessing directly the _PyThreadState_Current variable. The variable is no more exposed since Python 3.5.1 to hide the exact implementation of atomic C types, to avoid compiler issues.
- bpo-25791 [https://bugs.python.org/issue?@action=redirect&bpo=25791]: If _package_ != _spec_.parent or if neither _package_ or _spec_ are defined then ImportWarning is raised.
- bpo-22995 [https://bugs.python.org/issue?@action=redirect&bpo=22995]: [UPDATE] Comment out the one of the pickleability tests in _PyObject_GetState() due to regressions observed in Cython-based projects.
- bpo-25961 [https://bugs.python.org/issue?@action=redirect&bpo=25961]: Disallowed null characters in the type name.
- bpo-25973 [https://bugs.python.org/issue?@action=redirect&bpo=25973]: Fix segfault when an invalid nonlocal statement binds a name starting with two underscores.
- bpo-22995 [https://bugs.python.org/issue?@action=redirect&bpo=22995]: Instances of extension types with a state that aren't subclasses of list or dict and haven't implemented any pickle-related methods (_reduce_, _reduce_ex_, _getnewargs_, _getnewargs_ex_, or _getstate_), can no longer be pickled. Including memoryview.
- bpo-20440 [https://bugs.python.org/issue?

@action=redirect&bpo=20440]: Massive replacing unsafe attribute setting code with special macro Py_SETREF.

- bpo-25766 [https://bugs.python.org/issue?@action=redirect&bpo=25766]: Special method __bytes__() now works in str subclasses.
- bpo-25421 [https://bugs.python.org/issue?@action=redirect&bpo=25421]: __sizeof__ methods of builtin types now use dynamic basic size. This allows sys.getsize() to work correctly with their subclasses with __slots__ defined.
- bpo-25709 [https://bugs.python.org/issue?@action=redirect&bpo=25709]: Fixed problem with in-place string concatenation and utf-8 cache.
- bpo-5319 [https://bugs.python.org/issue?@action=redirect&bpo=5319]: New Py_FinalizeEx() API allowing Python to set an exit status of 120 on failure to flush buffered streams.
- bpo-25485 [https://bugs.python.org/issue?@action=redirect&bpo=25485]: telnetlib.Telnet is now a context manager.
- bpo-24097 [https://bugs.python.org/issue?@action=redirect&bpo=24097]: Fixed crash in object.__reduce__() if slot name is freed inside __getattr__.
- bpo-24731 [https://bugs.python.org/issue?@action=redirect&bpo=24731]: Fixed crash on converting objects with special methods __bytes__, __trunc__, and __float__ returning instances of subclasses of bytes, int, and float to subclasses of bytes, int, and float correspondingly.
- bpo-25630 [https://bugs.python.org/issue?@action=redirect&bpo=25630]: Fix a possible segfault during argument parsing in functions that accept filesystem paths.
- bpo-23564 [https://bugs.python.org/issue?@action=redirect&bpo=23564]: Fixed a partially broken sanity check in the _posixsubprocess internals regarding how fds_to_pass were passed to the child. The bug had no actual impact as subprocess.py already avoided it.
- bpo-25388 [https://bugs.python.org/issue?@action=redirect&bpo=25388]: Fixed tokenizer crash when processing undecodable source code with a null byte.
- bpo-25462 [https://bugs.python.org/issue?@action=redirect&bpo=25462]: The hash of the key now is calculated only once in most operations in C implementation

of OrderedDict.

- bpo-22995 [https://bugs.python.org/issue?@action=redirect&bpo=22995]: Default implementation of _reduce_ and _reduce_ex_ now rejects builtin types with not defined _new_.
- bpo-24802 [https://bugs.python.org/issue?@action=redirect&bpo=24802]: Avoid buffer overreads when int(), float(), compile(), exec() and eval() are passed bytes-like objects. These objects are not necessarily terminated by a null byte, but the functions assumed they were.
- bpo-25555 [https://bugs.python.org/issue?@action=redirect&bpo=25555]: Fix parser and AST: fill lineno and col_offset of «arg» node when compiling AST from Python objects.
- bpo-24726 [https://bugs.python.org/issue?@action=redirect&bpo=24726]: Fixed a crash and leaking NULL in repr() of OrderedDict that was mutated by direct calls of dict methods.
- bpo-25449 [https://bugs.python.org/issue?@action=redirect&bpo=25449]: Iterating OrderedDict with keys with unstable hash now raises KeyError in C implementations as well as in Python implementation.
- bpo-25395 [https://bugs.python.org/issue?@action=redirect&bpo=25395]: Fixed crash when highly nested OrderedDict structures were garbage collected.
- bpo-25401 [https://bugs.python.org/issue?@action=redirect&bpo=25401]: Optimize bytes.fromhex() and bytearray.fromhex(): they are now between 2x and 3.5x faster.
- bpo-25399 [https://bugs.python.org/issue?@action=redirect&bpo=25399]: Optimize bytearray % args using the new private _PyBytesWriter API. Formatting is now between 2.5 and 5 times faster.
- bpo-25274 [https://bugs.python.org/issue?@action=redirect&bpo=25274]: sys.setrecursionlimit() now raises a RecursionError if the new recursion limit is too low depending at the current recursion depth. Modify also the «lower-water mark» formula to make it monotonic. This mark is used to decide when the overflowed flag of the thread state is reset.

- bpo-24402 [https://bugs.python.org/issue?@action=redirect&bpo=24402]: Fix input() to prompt to the redirected stdout when sys.stdout.fileno() fails.
- bpo-25349 [https://bugs.python.org/issue?@action=redirect&bpo=25349]: Optimize bytes % args using the new private _PyBytesWriter API. Formatting is now up to 2 times faster.
- bpo-24806 [https://bugs.python.org/issue?@action=redirect&bpo=24806]: Prevent builtin types that are not allowed to be subclassed from being subclassed through multiple inheritance.
- bpo-25301 [https://bugs.python.org/issue?@action=redirect&bpo=25301]: The UTF-8 decoder is now up to 15 times as fast for error handlers: `ignore`, `replace` and `surrogateescape`.
- bpo-24848 [https://bugs.python.org/issue?@action=redirect&bpo=24848]: Fixed a number of bugs in UTF-7 decoding of misformed data.
- bpo-25267 [https://bugs.python.org/issue?@action=redirect&bpo=25267]: The UTF-8 encoder is now up to 75 times as fast for error handlers: `ignore`, `replace`, `surrogateescape`, `surrogatepass`. Patch co-written with Serhiy Storchaka.
- bpo-25280 [https://bugs.python.org/issue?@action=redirect&bpo=25280]: Import trace messages emitted in verbose (-v) mode are no longer formatted twice.
- bpo-25227 [https://bugs.python.org/issue?@action=redirect&bpo=25227]: Optimize ASCII and latin1 encoders with the `surrogateescape` error handler: the encoders are now up to 3 times as fast. Initial patch written by Serhiy Storchaka.
- bpo-25003 [https://bugs.python.org/issue?@action=redirect&bpo=25003]: On Solaris 11.3 or newer, os.urandom() now uses the getrandom() function instead of the getentropy() function. The getentropy() function is blocking to generate very good quality entropy, os.urandom() doesn't need such high-quality entropy.
- bpo-9232 [https://bugs.python.org/issue?@action=redirect&bpo=9232]: Modify Python's grammar to allow trailing commas in the argument list of a function declaration. For example, «def f(*,

a = 3,): pass» is now legal. Patch from Mark Dickinson.

- bpo-24965 [https://bugs.python.org/issue?@action=redirect&bpo=24965]: Implement **PEP 498** [https://peps.python.org/pep-0498/] «Literal String Interpolation». This allows you to embed expressions inside f-strings, which are converted to normal strings at run time. Given x=3, then f'value={x}" == "value=3". Patch by Eric V. Smith.
- bpo-26478 [https://bugs.python.org/issue?@action=redirect&bpo=26478]: Fix semantic bugs when using binary operators with dictionary views and tuples.
- bpo-26171 [https://bugs.python.org/issue?@action=redirect&bpo=26171]: Fix possible integer overflow and heap corruption in zipimporter.get_data().
- bpo-25660 [https://bugs.python.org/issue?@action=redirect&bpo=25660]: Fix TAB key behaviour in REPL with readline.
- bpo-26288 [https://bugs.python.org/issue?@action=redirect&bpo=26288]: Optimize PyLong_AsDouble.
- bpo-26289 [https://bugs.python.org/issue?@action=redirect&bpo=26289]: Optimize floor and modulo division for single-digit longs. Microbenchmarks show 2-2.5x improvement. Built-in "divmod" function is now also ~10% faster. (See also: bpo-26315 [https://bugs.python.org/issue?@action=redirect&bpo=26315])
- bpo-25887 [https://bugs.python.org/issue?@action=redirect&bpo=25887]: Raise a RuntimeError when a coroutine object is awaited more than once.

Library

- bpo-27057 [https://bugs.python.org/issue?@action=redirect&bpo=27057]: Fix os.set_inheritable() on Android, ioctl() is blocked by SELinux and fails with EACCESS. The function now falls back to fcntl(). Patch written by Michał Bednarski.
- bpo-27014 [https://bugs.python.org/issue?@action=redirect&bpo=27014]: Fix infinite recursion using typing.py. Thanks to Kalle Tuure!
- bpo-27031 [https://bugs.python.org/issue?@action=redirect&bpo=27031]: Removed dummy methods in

Tkinter widget classes: tk_menuBar() and tk_bindForTraversal().

- bpo-14132 [https://bugs.python.org/issue?@action=redirect&bpo=14132]: Fix urllib.request redirect handling when the target only has a query string. Original fix by Ján Janech.
- bpo-17214 [https://bugs.python.org/issue?@action=redirect&bpo=17214]: The «urllib.request» module now percent-encodes non-ASCII bytes found in redirect target URLs. Some servers send Location header fields with non-ASCII bytes, but «http.client» requires the request target to be ASCII-encodable, otherwise a UnicodeEncodeError is raised. Based on patch by Christian Heimes.
- bpo-27033 [https://bugs.python.org/issue?@action=redirect&bpo=27033]: The default value of the decode_data parameter for smtpd.SMTPChannel and smtpd.SMTPServer constructors is changed to False.
- bpo-27034 [https://bugs.python.org/issue?@action=redirect&bpo=27034]: Removed deprecated class asynchat.fifo.
- bpo-26870 [https://bugs.python.org/issue?@action=redirect&bpo=26870]: Added readline.set_auto_history(), which can stop entries being automatically added to the history list. Based on patch by Tyler Crompton.
- bpo-26039 [https://bugs.python.org/issue?@action=redirect&bpo=26039]: zipfile.ZipFile.open() can now be used to write data into a ZIP file, as well as for extracting data. Patch by Thomas Kluyver.
- bpo-26892 [https://bugs.python.org/issue?@action=redirect&bpo=26892]: Honor debuglevel flag in urllib.request.HTTPHandler. Patch contributed by Chi Hsuan Yen.
- bpo-22274 [https://bugs.python.org/issue?@action=redirect&bpo=22274]: In the subprocess module, allow stderr to be redirected to stdout even when stdout is not redirected. Patch by Akira Li.
- bpo-26807 [https://bugs.python.org/issue?@action=redirect&bpo=26807]: mock_open “files” no longer error on readline at end of file. Patch from Yolanda Robla.
- bpo-25745 [https://bugs.python.org/issue?

@action=redirect&bpo=25745]: Fixed leaking a userptr in curses panel destructor.
- bpo-26977 [https://bugs.python.org/issue?@action=redirect&bpo=26977]: Removed unnecessary, and ignored, call to sum of squares helper in statistics.pvariance.
- bpo-26002 [https://bugs.python.org/issue?@action=redirect&bpo=26002]: Use bisect in statistics.median instead of a linear search. Patch by Upendra Kuma.
- bpo-25974 [https://bugs.python.org/issue?@action=redirect&bpo=25974]: Make use of new Decimal.as_integer_ratio() method in statistics module. Patch by Stefan Krah.
- bpo-26996 [https://bugs.python.org/issue?@action=redirect&bpo=26996]: Add secrets module as described in **PEP 506** [https://peps.python.org/pep-0506/].
- bpo-26881 [https://bugs.python.org/issue?@action=redirect&bpo=26881]: The modulefinder module now supports extended opcode arguments.
- bpo-23815 [https://bugs.python.org/issue?@action=redirect&bpo=23815]: Fixed crashes related to directly created instances of types in _tkinter and curses.panel modules.
- bpo-17765 [https://bugs.python.org/issue?@action=redirect&bpo=17765]: weakref.ref() no longer silently ignores keyword arguments. Patch by Georg Brandl.
- bpo-26873 [https://bugs.python.org/issue?@action=redirect&bpo=26873]: xmlrpc now raises ResponseError on unsupported type tags instead of silently return incorrect result.
- bpo-26915 [https://bugs.python.org/issue?@action=redirect&bpo=26915]: The _contains_ methods in the collections ABCs now check for identity before checking equality. This better matches the behavior of the concrete classes, allows sensible handling of NaNs, and makes it easier to reason about container invariants.
- bpo-26711 [https://bugs.python.org/issue?@action=redirect&bpo=26711]: Fixed the comparison of plistlib.Data with other types.
- bpo-24114 [https://bugs.python.org/issue?@action=redirect&bpo=24114]: Fix an uninitialized variable in

`ctypes.util`. The bug only occurs on SunOS when the ctypes implementation searches for the `crle` program. Patch by Xiang Zhang. Tested on SunOS by Kees Bos.

- bpo-26864 [https://bugs.python.org/issue?@action=redirect&bpo=26864]: In urllib.request, change the proxy bypass host checking against no_proxy to be case-insensitive, and to not match unrelated host names that happen to have a bypassed hostname as a suffix. Patch by Xiang Zhang.
- bpo-24902 [https://bugs.python.org/issue?@action=redirect&bpo=24902]: Print server URL on http.server startup. Initial patch by Felix Kaiser.
- bpo-25788 [https://bugs.python.org/issue?@action=redirect&bpo=25788]: fileinput.hook_encoded() now supports an «errors» argument for passing to open. Original patch by Joseph Hackman.
- bpo-26634 [https://bugs.python.org/issue?@action=redirect&bpo=26634]: recursive_repr() now sets __qualname__ of wrapper. Patch by Xiang Zhang.
- bpo-26804 [https://bugs.python.org/issue?@action=redirect&bpo=26804]: urllib.request will prefer lower_case proxy environment variables over UPPER_CASE or Mixed_Case ones. Patch contributed by Hans-Peter Jansen.
- bpo-26837 [https://bugs.python.org/issue?@action=redirect&bpo=26837]: assertSequenceEqual() now correctly outputs non-stringified differing items (like bytes in the -b mode). This affects assertListEqual() and assertTupleEqual().
- bpo-26041 [https://bugs.python.org/issue?@action=redirect&bpo=26041]: Remove «will be removed in Python 3.7» from deprecation messages of platform.dist() and platform.linux_distribution(). Patch by Kumaripaba Miyurusara Athukorala.
- bpo-26822 [https://bugs.python.org/issue?@action=redirect&bpo=26822]: itemgetter, attrgetter and methodcaller objects no longer silently ignore keyword arguments.
- bpo-26733 [https://bugs.python.org/issue?@action=redirect&bpo=26733]: Disassembling a class now disassembles class and static methods. Patch by Xiang Zhang.

- bpo-26801 [https://bugs.python.org/issue?@action=redirect&bpo=26801]: Fix error handling in **`shutil.get_terminal_size()`**, catch **`AttributeError`** instead of **`NameError`**. Patch written by Emanuel Barry.
- bpo-24838 [https://bugs.python.org/issue?@action=redirect&bpo=24838]: tarfile's ustar and gnu formats now correctly calculate name and link field limits for multibyte character encodings like utf-8.
- bpo-26717 [https://bugs.python.org/issue?@action=redirect&bpo=26717]: Stop encoding Latin-1-ized WSGI paths with UTF-8. Patch by Anthony Sottile.
- bpo-26782 [https://bugs.python.org/issue?@action=redirect&bpo=26782]: Add STARTUPINFO to subprocess._all_ on Windows.
- bpo-26404 [https://bugs.python.org/issue?@action=redirect&bpo=26404]: Add context manager to socketserver. Patch by Aviv Palivoda.
- bpo-26735 [https://bugs.python.org/issue?@action=redirect&bpo=26735]: Fix **`os.urandom()`** on Solaris 11.3 and newer when reading more than 1,024 bytes: call `getrandom()` multiple times with a limit of 1024 bytes per call.
- bpo-26585 [https://bugs.python.org/issue?@action=redirect&bpo=26585]: Eliminate http.server._quote_html() and use html.escape(quote=False). Patch by Xiang Zhang.
- bpo-26685 [https://bugs.python.org/issue?@action=redirect&bpo=26685]: Raise OSError if closing a socket fails.
- bpo-16329 [https://bugs.python.org/issue?@action=redirect&bpo=16329]: Add .webm to mimetypes.types_map. Patch by Giampaolo Rodola".
- bpo-13952 [https://bugs.python.org/issue?@action=redirect&bpo=13952]: Add .csv to mimetypes.types_map. Patch by Geoff Wilson.
- bpo-26587 [https://bugs.python.org/issue?@action=redirect&bpo=26587]: the site module now allows .pth files to specify files to be added to sys.path (e.g. zip files).
- bpo-25609 [https://bugs.python.org/issue?@action=redirect&bpo=25609]: Introduce

contextlib.AbstractContextManager and typing.ContextManager.

- bpo-26709 [https://bugs.python.org/issue?@action=redirect&bpo=26709]: Fixed Y2038 problem in loading binary PLists.
- bpo-23735 [https://bugs.python.org/issue?@action=redirect&bpo=23735]: Handle terminal resizing with Readline 6.3+ by installing our own SIGWINCH handler. Patch by Eric Price.
- bpo-25951 [https://bugs.python.org/issue?@action=redirect&bpo=25951]: Change SSLSocket.sendall() to return None, as explicitly documented for plain socket objects. Patch by Aviv Palivoda.
- bpo-26586 [https://bugs.python.org/issue?@action=redirect&bpo=26586]: In http.server, respond with «413 Request header fields too large» if there are too many header fields to parse, rather than killing the connection and raising an unhandled exception. Patch by Xiang Zhang.
- bpo-26676 [https://bugs.python.org/issue?@action=redirect&bpo=26676]: Added missing XMLPullParser to ElementTree._all_.
- bpo-22854 [https://bugs.python.org/issue?@action=redirect&bpo=22854]: Change BufferedReader.writable() and BufferedWriter.readable() to always return False.
- bpo-26492 [https://bugs.python.org/issue?@action=redirect&bpo=26492]: Exhausted iterator of array.array now conforms with the behavior of iterators of other mutable sequences: it lefts exhausted even if iterated array is extended.
- bpo-26641 [https://bugs.python.org/issue?@action=redirect&bpo=26641]: doctest.DocFileTest and doctest.testfile() now support packages (module split into multiple directories) for the package parameter.
- bpo-25195 [https://bugs.python.org/issue?@action=redirect&bpo=25195]: Fix a regression in mock.MagicMock. _Call is a subclass of tuple (changeset 3603bae63c13 only works for classes) so we need to implement _ne_ ourselves. Patch by Andrew Plummer.
- bpo-26644 [https://bugs.python.org/issue?@action=redirect&bpo=26644]: Raise ValueError rather than

SystemError when a negative length is passed to SSLSocket.recv() or read().

- bpo-23804 [https://bugs.python.org/issue?@action=redirect&bpo=23804]: Fix SSL recv(0) and read(0) methods to return zero bytes instead of up to 1024.
- bpo-26616 [https://bugs.python.org/issue?@action=redirect&bpo=26616]: Fixed a bug in datetime.astimezone() method.
- bpo-26637 [https://bugs.python.org/issue?@action=redirect&bpo=26637]: The `importlib` module now emits an `ImportError` rather than a `TypeError` if `__import__()` is tried during the Python shutdown process but `sys.path` is already cleared (set to `None`).
- bpo-21925 [https://bugs.python.org/issue?@action=redirect&bpo=21925]: `warnings.formatwarning()` now catches exceptions when calling `linecache.getline()` and `tracemalloc.get_object_traceback()` to be able to log `ResourceWarning` emitted late during the Python shutdown process.
- bpo-23848 [https://bugs.python.org/issue?@action=redirect&bpo=23848]: On Windows, faulthandler.enable() now also installs an exception handler to dump the traceback of all Python threads on any Windows exception, not only on UNIX signals (SIGSEGV, SIGFPE, SIGABRT).
- bpo-26530 [https://bugs.python.org/issue?@action=redirect&bpo=26530]: Add C functions `_PyTraceMalloc_Track()` and `_PyTraceMalloc_Untrack()` to track memory blocks using the `tracemalloc` module. Add `_PyTraceMalloc_GetTraceback()` to get the traceback of an object.
- bpo-26588 [https://bugs.python.org/issue?@action=redirect&bpo=26588]: The _tracemalloc now supports tracing memory allocations of multiple address spaces (domains).
- bpo-24266 [https://bugs.python.org/issue?@action=redirect&bpo=24266]: Ctrl+C during Readline history search now cancels the search mode when compiled with

Readline 7.

- bpo-26590 [https://bugs.python.org/issue?@action=redirect&bpo=26590]: Implement a safe finalizer for the _socket.socket type. It now releases the GIL to close the socket.
- bpo-18787 [https://bugs.python.org/issue?@action=redirect&bpo=18787]: spwd.getspnam() now raises a PermissionError if the user doesn’t have privileges.
- bpo-26560 [https://bugs.python.org/issue?@action=redirect&bpo=26560]: Avoid potential ValueError in BaseHandler.start_response. Initial patch by Peter Inglesby.
- bpo-26567 [https://bugs.python.org/issue?@action=redirect&bpo=26567]: Add a new function `PyErr_ResourceWarning()` function to pass the destroyed object. Add a *source* attribute to **`warnings.WarningMessage`**. Add warnings._showwarnmsg() which uses tracemalloc to get the traceback where source object was allocated.
- bpo-26569 [https://bugs.python.org/issue?@action=redirect&bpo=26569]: Fix `pyclbr.readmodule()` and `pyclbr.readmodule_ex()` to support importing packages.
- bpo-26499 [https://bugs.python.org/issue?@action=redirect&bpo=26499]: Account for remaining Content-Length in HTTPResponse.readline() and read1(). Based on patch by Silent Ghost. Also document that HTTPResponse now supports these methods.
- bpo-25320 [https://bugs.python.org/issue?@action=redirect&bpo=25320]: Handle sockets in directories unittest discovery is scanning. Patch from Victor van den Elzen.
- bpo-16181 [https://bugs.python.org/issue?@action=redirect&bpo=16181]: cookiejar.http2time() now returns None if year is higher than datetime.MAXYEAR.
- bpo-26513 [https://bugs.python.org/issue?@action=redirect&bpo=26513]: Fixes platform module detection of Windows Server
- bpo-23718 [https://bugs.python.org/issue?@action=redirect&bpo=23718]: Fixed parsing time in week 0 before Jan 1. Original patch by Tamás Bence Gedai.
- bpo-26323 [https://bugs.python.org/issue?

@action=redirect&bpo=26323]: Add Mock.assert_called() and Mock.assert_called_once() methods to unittest.mock. Patch written by Amit Saha.

- bpo-20589 [https://bugs.python.org/issue?@action=redirect&bpo=20589]: Invoking Path.owner() and Path.group() on Windows now raise NotImplementedError instead of ImportError.
- bpo-26177 [https://bugs.python.org/issue?@action=redirect&bpo=26177]: Fixed the keys() method for Canvas and Scrollbar widgets.
- bpo-15068 [https://bugs.python.org/issue?@action=redirect&bpo=15068]: Got rid of excessive buffering in fileinput. The bufsize parameter is now deprecated and ignored.
- bpo-19475 [https://bugs.python.org/issue?@action=redirect&bpo=19475]: Added an optional argument timespec to the datetime isoformat() method to choose the precision of the time component.
- bpo-2202 [https://bugs.python.org/issue?@action=redirect&bpo=2202]: Fix UnboundLocalError in AbstractDigestAuthHandler.get_algorithm_impls. Initial patch by Mathieu Dupuy.
- bpo-26167 [https://bugs.python.org/issue?@action=redirect&bpo=26167]: Minimized overhead in copy.copy() and copy.deepcopy(). Optimized copying and deepcopying bytearrays, NotImplemented, slices, short lists, tuples, dicts, sets.
- bpo-25718 [https://bugs.python.org/issue?@action=redirect&bpo=25718]: Fixed pickling and copying the accumulate() iterator with total is None.
- bpo-26475 [https://bugs.python.org/issue?@action=redirect&bpo=26475]: Fixed debugging output for regular expressions with the (?x) flag.
- bpo-26482 [https://bugs.python.org/issue?@action=redirect&bpo=26482]: Allowed pickling recursive dequeues.
- bpo-26335 [https://bugs.python.org/issue?@action=redirect&bpo=26335]: Make mmap.write() return the number of bytes written like other write methods. Patch by Jakub Stasiak.

- bpo-26457 [https://bugs.python.org/issue?@action=redirect&bpo=26457]: Fixed the subnets() methods in IP network classes for the case when resulting prefix length is equal to maximal prefix length. Based on patch by Xiang Zhang.
- bpo-26385 [https://bugs.python.org/issue?@action=redirect&bpo=26385]: Remove the file if the internal open() call in NamedTemporaryFile() fails. Patch by Silent Ghost.
- bpo-26402 [https://bugs.python.org/issue?@action=redirect&bpo=26402]: Fix XML-RPC client to retry when the server shuts down a persistent connection. This was a regression related to the new http.client.RemoteDisconnected exception in 3.5.0a4.
- bpo-25913 [https://bugs.python.org/issue?@action=redirect&bpo=25913]: Leading `<~` is optional now in base64.a85decode() with adobe=True. Patch by Swati Jaiswal.
- bpo-26186 [https://bugs.python.org/issue?@action=redirect&bpo=26186]: Remove an invalid type check in importlib.util.LazyLoader.
- bpo-26367 [https://bugs.python.org/issue?@action=redirect&bpo=26367]: importlib.__import__() raises ImportError like builtins.__import__() when `level` is specified but without an accompanying package specified.
- bpo-26309 [https://bugs.python.org/issue?@action=redirect&bpo=26309]: In the «socketserver» module, shut down the request (closing the connected socket) when verify_request() returns false. Patch by Aviv Palivoda.
- bpo-23430 [https://bugs.python.org/issue?@action=redirect&bpo=23430]: Change the socketserver module to only catch exceptions raised from a request handler that are derived from Exception (instead of BaseException). Therefore SystemExit and KeyboardInterrupt no longer trigger the handle_error() method, and will now to stop a single-threaded server.
- bpo-25995 [https://bugs.python.org/issue?@action=redirect&bpo=25995]: os.walk() no longer uses FDs proportional to the tree depth.
- bpo-25994 [https://bugs.python.org/issue?

@action=redirect&bpo=25994]: Added the close() method and the support of the context manager protocol for the os.scandir() iterator.

- bpo-23992 [https://bugs.python.org/issue?@action=redirect&bpo=23992]: multiprocessing: make MapResult not fail-fast upon exception.
- bpo-26243 [https://bugs.python.org/issue?@action=redirect&bpo=26243]: Support keyword arguments to zlib.compress(). Patch by Aviv Palivoda.
- bpo-26117 [https://bugs.python.org/issue?@action=redirect&bpo=26117]: The os.scandir() iterator now closes file descriptor not only when the iteration is finished, but when it was failed with error.
- bpo-25949 [https://bugs.python.org/issue?@action=redirect&bpo=25949]: __dict__ for an OrderedDict instance is now created only when needed.
- bpo-25911 [https://bugs.python.org/issue?@action=redirect&bpo=25911]: Restored support of bytes paths in os.walk() on Windows.
- bpo-26045 [https://bugs.python.org/issue?@action=redirect&bpo=26045]: Add UTF-8 suggestion to error message when posting a non-Latin-1 string with http.client.
- bpo-26039 [https://bugs.python.org/issue?@action=redirect&bpo=26039]: Added zipfile.ZipInfo.from_file() and zipinfo.ZipInfo.is_dir(). Patch by Thomas Kluyver.
- bpo-12923 [https://bugs.python.org/issue?@action=redirect&bpo=12923]: Reset FancyURLopener's redirect counter even if there is an exception. Based on patches by Brian Brazil and Daniel Rocco.
- bpo-25945 [https://bugs.python.org/issue?@action=redirect&bpo=25945]: Fixed a crash when unpickle the functools.partial object with wrong state. Fixed a leak in failed functools.partial constructor. «args» and «keywords» attributes of functools.partial have now always types tuple and dict correspondingly.
- bpo-26202 [https://bugs.python.org/issue?@action=redirect&bpo=26202]: copy.deepcopy() now correctly copies range() objects with non-atomic attributes.
- bpo-23076 [https://bugs.python.org/issue?@action=redirect&bpo=23076]: Path.glob() now raises a

ValueError if it’s called with an invalid pattern. Patch by Thomas Nyberg.

- bpo-19883 [https://bugs.python.org/issue?@action=redirect&bpo=19883]: Fixed possible integer overflows in zipimport.
- bpo-26227 [https://bugs.python.org/issue?@action=redirect&bpo=26227]: On Windows, getnameinfo(), gethostbyaddr() and gethostbyname_ex() functions of the socket module now decode the hostname from the ANSI code page rather than UTF-8.
- bpo-26099 [https://bugs.python.org/issue?@action=redirect&bpo=26099]: The site module now writes an error into stderr if sitecustomize module can be imported but executing the module raise an ImportError. Same change for usercustomize.
- bpo-26147 [https://bugs.python.org/issue?@action=redirect&bpo=26147]: xmlrpc now works with strings not encodable with used non-UTF-8 encoding.
- bpo-25935 [https://bugs.python.org/issue?@action=redirect&bpo=25935]: Garbage collector now breaks reference loops with OrderedDict.
- bpo-16620 [https://bugs.python.org/issue?@action=redirect&bpo=16620]: Fixed AttributeError in msilib.Directory.glob().
- bpo-26013 [https://bugs.python.org/issue?@action=redirect&bpo=26013]: Added compatibility with broken protocol 2 pickles created in old Python 3 versions (3.4.3 and lower).
- bpo-26129 [https://bugs.python.org/issue?@action=redirect&bpo=26129]: Deprecated accepting non-integers in grp.getgrgid().
- bpo-25850 [https://bugs.python.org/issue?@action=redirect&bpo=25850]: Use cross-compilation by default for 64-bit Windows.
- bpo-25822 [https://bugs.python.org/issue?@action=redirect&bpo=25822]: Add docstrings to the fields of urllib.parse results. Patch contributed by Swati Jaiswal.
- bpo-22642 [https://bugs.python.org/issue?@action=redirect&bpo=22642]: Convert trace module option parsing mechanism to argparse. Patch contributed by

SilentGhost.

- bpo-24705 [https://bugs.python.org/issue?@action=redirect&bpo=24705]: Fix sysconfig._parse_makefile not expanding ${} vars appearing before $() vars.
- bpo-26069 [https://bugs.python.org/issue?@action=redirect&bpo=26069]: Remove the deprecated apis in the trace module.
- bpo-22138 [https://bugs.python.org/issue?@action=redirect&bpo=22138]: Fix mock.patch behavior when patching descriptors. Restore original values after patching. Patch contributed by Sean McCully.
- bpo-25672 [https://bugs.python.org/issue?@action=redirect&bpo=25672]: In the ssl module, enable the SSL_MODE_RELEASE_BUFFERS mode option if it is safe to do so.
- bpo-26012 [https://bugs.python.org/issue?@action=redirect&bpo=26012]: Don't traverse into symlinks for ** pattern in pathlib.Path.[r]glob().
- bpo-24120 [https://bugs.python.org/issue?@action=redirect&bpo=24120]: Ignore PermissionError when traversing a tree with pathlib.Path.[r]glob(). Patch by Ulrich Petri.
- bpo-21815 [https://bugs.python.org/issue?@action=redirect&bpo=21815]: Accept] characters in the data portion of imap responses, in order to handle the flags with square brackets accepted and produced by servers such as gmail.
- bpo-25447 [https://bugs.python.org/issue?@action=redirect&bpo=25447]: fileinput now uses sys.stdin as-is if it does not have a buffer attribute (restores backward compatibility).
- bpo-25971 [https://bugs.python.org/issue?@action=redirect&bpo=25971]: Optimized creating Fractions from floats by 2 times and from Decimals by 3 times.
- bpo-25802 [https://bugs.python.org/issue?@action=redirect&bpo=25802]: Document as deprecated the remaining implementations of importlib.abc.Loader.load_module().
- bpo-25928 [https://bugs.python.org/issue?@action=redirect&bpo=25928]: Add Decimal.as_integer_ratio().

- bpo-25447 [https://bugs.python.org/issue?@action=redirect&bpo=25447]: Copying the lru_cache() wrapper object now always works, independently from the type of the wrapped object (by returning the original object unchanged).
- bpo-25768 [https://bugs.python.org/issue?@action=redirect&bpo=25768]: Have the functions in compileall return booleans instead of ints and add proper documentation and tests for the return values.
- bpo-24103 [https://bugs.python.org/issue?@action=redirect&bpo=24103]: Fixed possible use after free in ElementTree.XMLPullParser.
- bpo-25860 [https://bugs.python.org/issue?@action=redirect&bpo=25860]: os.fwalk() no longer skips remaining directories when error occurs. Original patch by Samson Lee.
- bpo-25914 [https://bugs.python.org/issue?@action=redirect&bpo=25914]: Fixed and simplified OrderedDict.__sizeof__.
- bpo-25869 [https://bugs.python.org/issue?@action=redirect&bpo=25869]: Optimized deepcopying ElementTree; it is now 20 times faster.
- bpo-25873 [https://bugs.python.org/issue?@action=redirect&bpo=25873]: Optimized iterating ElementTree. Iterating elements Element.iter() is now 40% faster, iterating text Element.itertext() is now up to 2.5 times faster.
- bpo-25902 [https://bugs.python.org/issue?@action=redirect&bpo=25902]: Fixed various refcount issues in ElementTree iteration.
- bpo-22227 [https://bugs.python.org/issue?@action=redirect&bpo=22227]: The TarFile iterator is reimplemented using generator. This implementation is simpler that using class.
- bpo-25638 [https://bugs.python.org/issue?@action=redirect&bpo=25638]: Optimized ElementTree.iterparse(); it is now 2x faster. Optimized ElementTree parsing; it is now 10% faster.
- bpo-25761 [https://bugs.python.org/issue?@action=redirect&bpo=25761]: Improved detecting errors in broken pickle data.
- bpo-25717 [https://bugs.python.org/issue?

@action=redirect&bpo=25717]: Restore the previous behaviour of tolerating most fstat() errors when opening files. This was a regression in 3.5a1, and stopped anonymous temporary files from working in special cases.

- bpo-24903 [https://bugs.python.org/issue?@action=redirect&bpo=24903]: Fix regression in number of arguments compileall accepts when "-d" is specified. The check on the number of arguments has been dropped completely as it never worked correctly anyway.
- bpo-25764 [https://bugs.python.org/issue?@action=redirect&bpo=25764]: In the subprocess module, preserve any exception caused by fork() failure when preexec_fn is used.
- bpo-25771 [https://bugs.python.org/issue?@action=redirect&bpo=25771]: Tweak the exception message for importlib.util.resolve_name() when "package" isn't specified but necessary.
- bpo-6478 [https://bugs.python.org/issue?@action=redirect&bpo=6478]: _strptime's regexp cache now is reset after changing timezone with time.tzset().
- bpo-14285 [https://bugs.python.org/issue?@action=redirect&bpo=14285]: When executing a package with the «python -m package» option, and package initialization fails, a proper traceback is now reported. The «runpy» module now lets exceptions from package initialization pass back to the caller, rather than raising ImportError.
- bpo-19771 [https://bugs.python.org/issue?@action=redirect&bpo=19771]: Also in runpy and the «-m» option, omit the irrelevant message «… is a package and cannot be directly executed» if the package could not even be initialized (e.g. due to a bad `*.pyc` file).
- bpo-25177 [https://bugs.python.org/issue?@action=redirect&bpo=25177]: Fixed problem with the mean of very small and very large numbers. As a side effect, statistics.mean and statistics.variance should be significantly faster.
- bpo-25718 [https://bugs.python.org/issue?@action=redirect&bpo=25718]: Fixed copying object with state with boolean value is false.
- bpo-10131 [https://bugs.python.org/issue?

@action=redirect&bpo=10131]: Fixed deep copying of minidom documents. Based on patch by Marian Ganisin.
- bpo-7990 [https://bugs.python.org/issue?@action=redirect&bpo=7990]: dir() on ElementTree.Element now lists properties: «tag», «text», «tail» and «attrib». Original patch by Santoso Wijaya.
- bpo-25725 [https://bugs.python.org/issue?@action=redirect&bpo=25725]: Fixed a reference leak in pickle.loads() when unpickling invalid data including tuple instructions.
- bpo-25663 [https://bugs.python.org/issue?@action=redirect&bpo=25663]: In the Readline completer, avoid listing duplicate global names, and search the global namespace before searching builtins.
- bpo-25688 [https://bugs.python.org/issue?@action=redirect&bpo=25688]: Fixed file leak in ElementTree.iterparse() raising an error.
- bpo-23914 [https://bugs.python.org/issue?@action=redirect&bpo=23914]: Fixed SystemError raised by unpickler on broken pickle data.
- bpo-25691 [https://bugs.python.org/issue?@action=redirect&bpo=25691]: Fixed crash on deleting ElementTree.Element attributes.
- bpo-25624 [https://bugs.python.org/issue?@action=redirect&bpo=25624]: ZipFile now always writes a ZIP_STORED header for directory entries. Patch by Dingyuan Wang.
- bpo-25626 [https://bugs.python.org/issue?@action=redirect&bpo=25626]: Change three zlib functions to accept sizes that fit in Py_ssize_t, but internally cap those sizes to UINT_MAX. This resolves a regression in 3.5 where GzipFile.read() failed to read chunks larger than 2 or 4 GiB. The change affects the zlib.Decompress.decompress() max_length parameter, the zlib.decompress() bufsize parameter, and the zlib.Decompress.flush() length parameter.
- bpo-25583 [https://bugs.python.org/issue?@action=redirect&bpo=25583]: Avoid incorrect errors raised by os.makedirs(exist_ok=True) when the OS gives priority to errors such as EACCES over EEXIST.
- bpo-25593 [https://bugs.python.org/issue?@action=redirect&bpo=25593]: Change semantics of

EventLoop.stop() in asyncio.

- bpo-6973 [https://bugs.python.org/issue?@action=redirect&bpo=6973]: When we know a subprocess.Popen process has died, do not allow the send_signal(), terminate(), or kill() methods to do anything as they could potentially signal a different process.
- bpo-23883 [https://bugs.python.org/issue?@action=redirect&bpo=23883]: Added missing APIs to _all_ to match the documented APIs for the following modules: calendar, csv, enum, fileinput, ftplib, logging, optparse, tarfile, threading and wave. Also added a test.support.check_all_() helper. Patches by Jacek Kołodziej, Mauro S. M. Rodrigues and Joel Taddei.
- bpo-25590 [https://bugs.python.org/issue?@action=redirect&bpo=25590]: In the Readline completer, only call getattr() once per attribute. Also complete names of attributes such as properties and slots which are listed by dir() but not yet created on an instance.
- bpo-25498 [https://bugs.python.org/issue?@action=redirect&bpo=25498]: Fix a crash when garbage-collecting ctypes objects created by wrapping a memoryview. This was a regression made in 3.5a1. Based on patch by Eryksun.
- bpo-25584 [https://bugs.python.org/issue?@action=redirect&bpo=25584]: Added «escape» to the _all_ list in the glob module.
- bpo-25584 [https://bugs.python.org/issue?@action=redirect&bpo=25584]: Fixed recursive glob() with patterns starting with **.
- bpo-25446 [https://bugs.python.org/issue?@action=redirect&bpo=25446]: Fix regression in smtplib’s AUTH LOGIN support.
- bpo-18010 [https://bugs.python.org/issue?@action=redirect&bpo=18010]: Fix the pydoc web server’s module search function to handle exceptions from importing packages.
- bpo-25554 [https://bugs.python.org/issue?@action=redirect&bpo=25554]: Got rid of circular references in regular expression parsing.
- bpo-18973 [https://bugs.python.org/issue?@action=redirect&bpo=18973]: Command-line interface of the

calendar module now uses argparse instead of optparse.

- bpo-25510 [https://bugs.python.org/issue?@action=redirect&bpo=25510]: fileinput.FileInput.readline() now returns b”” instead of “” at the end if the FileInput was opened with binary mode. Patch by Ryosuke Ito.
- bpo-25503 [https://bugs.python.org/issue?@action=redirect&bpo=25503]: Fixed inspect.getdoc() for inherited docstrings of properties. Original patch by John Mark Vandenberg.
- bpo-25515 [https://bugs.python.org/issue?@action=redirect&bpo=25515]: Always use os.urandom as a source of randomness in uuid.uuid4.
- bpo-21827 [https://bugs.python.org/issue?@action=redirect&bpo=21827]: Fixed textwrap.dedent() for the case when largest common whitespace is a substring of smallest leading whitespace. Based on patch by Robert Li.
- bpo-25447 [https://bugs.python.org/issue?@action=redirect&bpo=25447]: The lru_cache() wrapper objects now can be copied and pickled (by returning the original object unchanged).
- bpo-25390 [https://bugs.python.org/issue?@action=redirect&bpo=25390]: typing: Don’t crash on Union[str, Pattern].
- bpo-25441 [https://bugs.python.org/issue?@action=redirect&bpo=25441]: asyncio: Raise error from drain() when socket is closed.
- bpo-25410 [https://bugs.python.org/issue?@action=redirect&bpo=25410]: Cleaned up and fixed minor bugs in C implementation of OrderedDict.
- bpo-25411 [https://bugs.python.org/issue?@action=redirect&bpo=25411]: Improved Unicode support in SMTPHandler through better use of the email package. Thanks to user simon04 for the patch.
- Move the imp module from a PendingDeprecationWarning to DeprecationWarning.
- bpo-25407 [https://bugs.python.org/issue?@action=redirect&bpo=25407]: Remove mentions of the formatter module being removed in Python 3.6.
- bpo-25406 [https://bugs.python.org/issue?@action=redirect&bpo=25406]: Fixed a bug in C implementation

of OrderedDict.move_to_end() that caused segmentation fault or hang in iterating after moving several items to the start of ordered dict.

- bpo-25382 [https://bugs.python.org/issue?@action=redirect&bpo=25382]: pickletools.dis() now outputs implicit memo index for the MEMOIZE opcode.
- bpo-25357 [https://bugs.python.org/issue?@action=redirect&bpo=25357]: Add an optional newline parameter to binascii.b2a_base64(). base64.b64encode() uses it to avoid a memory copy.
- bpo-24164 [https://bugs.python.org/issue?@action=redirect&bpo=24164]: Objects that need calling `__new__` with keyword arguments, can now be pickled using pickle protocols older than protocol version 4.
- bpo-25364 [https://bugs.python.org/issue?@action=redirect&bpo=25364]: zipfile now works in threads disabled builds.
- bpo-25328 [https://bugs.python.org/issue?@action=redirect&bpo=25328]: smtpd's SMTPChannel now correctly raises a ValueError if both decode_data and enable_SMTPUTF8 are set to true.
- bpo-16099 [https://bugs.python.org/issue?@action=redirect&bpo=16099]: RobotFileParser now supports Crawl-delay and Request-rate extensions. Patch by Nikolay Bogoychev.
- bpo-25316 [https://bugs.python.org/issue?@action=redirect&bpo=25316]: distutils raises OSError instead of DistutilsPlatformError when MSVC is not installed.
- bpo-25380 [https://bugs.python.org/issue?@action=redirect&bpo=25380]: Fixed protocol for the STACK_GLOBAL opcode in pickletools.opcodes.
- bpo-23972 [https://bugs.python.org/issue?@action=redirect&bpo=23972]: Updates asyncio datagram create method allowing reuseport and reuseaddr socket options to be set prior to binding the socket. Mirroring the existing asyncio create_server method the reuseaddr option for datagram sockets defaults to True if the O/S is “posix” (except if the platform is Cygwin). Patch by Chris Laws.
- bpo-25304 [https://bugs.python.org/issue?@action=redirect&bpo=25304]: Add

asyncio.run_coroutine_threadsafe(). This lets you submit a coroutine to a loop from another thread, returning a concurrent.futures.Future. By Vincent Michel.

- bpo-25232 [https://bugs.python.org/issue?@action=redirect&bpo=25232]: Fix CGIRequestHandler to split the query from the URL at the first question mark (?) rather than the last. Patch from Xiang Zhang.
- bpo-24657 [https://bugs.python.org/issue?@action=redirect&bpo=24657]: Prevent CGIRequestHandler from collapsing slashes in the query part of the URL as if it were a path. Patch from Xiang Zhang.
- bpo-25287 [https://bugs.python.org/issue?@action=redirect&bpo=25287]: Don’t add crypt.METHOD_CRYPT to crypt.methods if it’s not supported. Check if it is supported, it may not be supported on OpenBSD for example.
- bpo-23600 [https://bugs.python.org/issue?@action=redirect&bpo=23600]: Default implementation of tzinfo.fromutc() was returning wrong results in some cases.
- bpo-25203 [https://bugs.python.org/issue?@action=redirect&bpo=25203]: Failed readline.set_completer_delims() no longer left the module in inconsistent state.
- bpo-25011 [https://bugs.python.org/issue?@action=redirect&bpo=25011]: rlcompleter now omits private and special attribute names unless the prefix starts with underscores.
- bpo-25209 [https://bugs.python.org/issue?@action=redirect&bpo=25209]: rlcompleter now can add a space or a colon after completed keyword.
- bpo-22241 [https://bugs.python.org/issue?@action=redirect&bpo=22241]: timezone.utc name is now plain “UTC”, not “UTC-00:00”.
- bpo-23517 [https://bugs.python.org/issue?@action=redirect&bpo=23517]: fromtimestamp() and utcfromtimestamp() methods of datetime.datetime now round microseconds to nearest with ties going to nearest even integer (ROUND_HALF_EVEN), as round(float), instead of rounding towards -Infinity (ROUND_FLOOR).
- bpo-23552 [https://bugs.python.org/issue?@action=redirect&bpo=23552]: Timeit now warns when there is

substantial (4x) variance between best and worst times. Patch from Serhiy Storchaka.

- bpo-24633 [https://bugs.python.org/issue?@action=redirect&bpo=24633]: site-packages/README -> README.txt.
- bpo-24879 [https://bugs.python.org/issue?@action=redirect&bpo=24879]: help() and pydoc can now list named tuple fields in the order they were defined rather than alphabetically. The ordering is determined by the _fields attribute if present.
- bpo-24874 [https://bugs.python.org/issue?@action=redirect&bpo=24874]: Improve speed of itertools.cycle() and make its pickle more compact.
- Fix crash in itertools.cycle._setstate_() when the first argument wasn't a list.
- bpo-20059 [https://bugs.python.org/issue?@action=redirect&bpo=20059]: urllib.parse raises ValueError on all invalid ports. Patch by Martin Panter.
- bpo-24360 [https://bugs.python.org/issue?@action=redirect&bpo=24360]: Improve _repr_ of argparse.Namespace() for invalid identifiers. Patch by Matthias Bussonnier.
- bpo-23426 [https://bugs.python.org/issue?@action=redirect&bpo=23426]: run_setup was broken in distutils. Patch from Alexander Belopolsky.
- bpo-13938 [https://bugs.python.org/issue?@action=redirect&bpo=13938]: 2to3 converts StringTypes to a tuple. Patch from Mark Hammond.
- bpo-2091 [https://bugs.python.org/issue?@action=redirect&bpo=2091]: open() accepted a “U” mode string containing “+”, but “U” can only be used with “r”. Patch from Jeff Balogh and John O'Connor.
- bpo-8585 [https://bugs.python.org/issue?@action=redirect&bpo=8585]: improved tests for zipimporter2. Patch from Mark Lawrence.
- bpo-18622 [https://bugs.python.org/issue?@action=redirect&bpo=18622]: unittest.mock.mock_open().reset_mock would recurse infinitely. Patch from Nicola Palumbo and Laurent De Buyst.
- bpo-24426 [https://bugs.python.org/issue?@action=redirect&bpo=24426]: Fast searching optimization in

regular expressions now works for patterns that starts with capturing groups. Fast searching optimization now can't be disabled at compile time.

- bpo-23661 [https://bugs.python.org/issue?@action=redirect&bpo=23661]: unittest.mock side_effects can now be exceptions again. This was a regression vs Python 3.4. Patch from Ignacio Rossi
- bpo-13248 [https://bugs.python.org/issue?@action=redirect&bpo=13248]: Remove deprecated inspect.getmoduleinfo function.
- bpo-25578 [https://bugs.python.org/issue?@action=redirect&bpo=25578]: Fix (another) memory leak in SSLSocket.getpeercer().
- bpo-25530 [https://bugs.python.org/issue?@action=redirect&bpo=25530]: Disable the vulnerable SSLv3 protocol by default when creating ssl.SSLContext.
- bpo-25569 [https://bugs.python.org/issue?@action=redirect&bpo=25569]: Fix memory leak in SSLSocket.getpeercert().
- bpo-25471 [https://bugs.python.org/issue?@action=redirect&bpo=25471]: Sockets returned from accept() shouldn't appear to be nonblocking.
- bpo-25319 [https://bugs.python.org/issue?@action=redirect&bpo=25319]: When threading.Event is reinitialized, the underlying condition should use a regular lock rather than a recursive lock.
- Skip getaddrinfo if host is already resolved. Patch by A. Jesse Jiryu Davis.
- bpo-26050 [https://bugs.python.org/issue?@action=redirect&bpo=26050]: Add asyncio.StreamReader.readuntil() method. Patch by Марк Коренберг.
- bpo-25924 [https://bugs.python.org/issue?@action=redirect&bpo=25924]: Avoid unnecessary serialization of getaddrinfo(3) calls on OS X versions 10.5 or higher. Original patch by A. Jesse Jiryu Davis.
- bpo-26406 [https://bugs.python.org/issue?@action=redirect&bpo=26406]: Avoid unnecessary serialization of getaddrinfo(3) calls on current versions of OpenBSD and NetBSD. Patch by A. Jesse Jiryu Davis.

- bpo-26848 [https://bugs.python.org/issue?@action=redirect&bpo=26848]: Fix asyncio/subprocess.communicate() to handle empty input. Patch by Jack O'Connor.
- bpo-27040 [https://bugs.python.org/issue?@action=redirect&bpo=27040]: Add loop.get_exception_handler method
- bpo-27041 [https://bugs.python.org/issue?@action=redirect&bpo=27041]: asyncio: Add loop.create_future method

IDLE

- bpo-20640 [https://bugs.python.org/issue?@action=redirect&bpo=20640]: Add tests for idlelib.configHelpSourceEdit. Patch by Saimadhav Heblikar.
- In the "IDLE-console differences" section of the IDLE doc, clarify how running with IDLE affects sys.modules and the standard streams.
- bpo-25507 [https://bugs.python.org/issue?@action=redirect&bpo=25507]: fix incorrect change in IOBinding that prevented printing. Augment IOBinding htest to include all major IOBinding functions.
- bpo-25905 [https://bugs.python.org/issue?@action=redirect&bpo=25905]: Revert unwanted conversion of " to ' RIGHT SINGLE QUOTATION MARK in README.txt and open this and NEWS.txt with "ascii". Re-encode CREDITS.txt to utf-8 and open it with "utf-8".
- bpo-15348 [https://bugs.python.org/issue?@action=redirect&bpo=15348]: Stop the debugger engine (normally in a user process) before closing the debugger window (running in the IDLE process). This prevents the RuntimeErrors that were being caught and ignored.
- bpo-24455 [https://bugs.python.org/issue?@action=redirect&bpo=24455]: Prevent IDLE from hanging when a) closing the shell while the debugger is active (15347); b) closing the debugger with the [X] button (15348); and c) activating the debugger when already active (24455). The patch by Mark Roseman does this by making two changes. 1. Suspend and resume the gui.interaction method with the tcl

vwait mechanism intended for this purpose (instead of root.mainloop & .quit). 2. In gui.run, allow any existing interaction to terminate first.

- Change "The program" to "Your program" in an IDLE "kill program?" message to make it clearer that the program referred to is the currently running user program, not IDLE itself.
- bpo-24750 [https://bugs.python.org/issue?@action=redirect&bpo=24750]: Improve the appearance of the IDLE editor window status bar. Patch by Mark Roseman.
- bpo-25313 [https://bugs.python.org/issue?@action=redirect&bpo=25313]: Change the handling of new built-in text color themes to better address the compatibility problem introduced by the addition of IDLE Dark. Consistently use the revised idleConf.CurrentTheme everywhere in idlelib.
- bpo-24782 [https://bugs.python.org/issue?@action=redirect&bpo=24782]: Extension configuration is now a tab in the IDLE Preferences dialog rather than a separate dialog. The former tabs are now a sorted list. Patch by Mark Roseman.
- bpo-22726 [https://bugs.python.org/issue?@action=redirect&bpo=22726]: Re-activate the config dialog help button with some content about the other buttons and the new IDLE Dark theme.
- bpo-24820 [https://bugs.python.org/issue?@action=redirect&bpo=24820]: IDLE now has an "IDLE Dark" built-in text color theme. It is more or less IDLE Classic inverted, with a cobalt blue background. Strings, comments, keywords, … are still green, red, orange, … . To use it with IDLEs released before November 2015, hit the "Save as New Custom Theme" button and enter a new name, such as "Custom Dark". The custom theme will work with any IDLE release, and can be modified.
- bpo-25224 [https://bugs.python.org/issue?@action=redirect&bpo=25224]: README.txt is now an idlelib index for IDLE developers and curious users. The previous user content is now in the IDLE doc chapter. "IDLE" now means "Integrated Development and Learning Environment".
- bpo-24820 [https://bugs.python.org/issue?

@action=redirect&bpo=24820]: Users can now set breakpoint colors in Settings -> Custom Highlighting. Original patch by Mark Roseman.
- bpo-24972 [https://bugs.python.org/issue?@action=redirect&bpo=24972]: Inactive selection background now matches active selection background, as configured by users, on all systems. Found items are now always highlighted on Windows. Initial patch by Mark Roseman.
- bpo-24570 [https://bugs.python.org/issue?@action=redirect&bpo=24570]: Idle: make calltip and completion boxes appear on Macs affected by a tk regression. Initial patch by Mark Roseman.
- bpo-24988 [https://bugs.python.org/issue?@action=redirect&bpo=24988]: Idle ScrolledList context menus (used in debugger) now work on Mac Aqua. Patch by Mark Roseman.
- bpo-24801 [https://bugs.python.org/issue?@action=redirect&bpo=24801]: Make right-click for context menu work on Mac Aqua. Patch by Mark Roseman.
- bpo-25173 [https://bugs.python.org/issue?@action=redirect&bpo=25173]: Associate tkinter messageboxes with a specific widget. For Mac OSX, make them a “sheet”. Patch by Mark Roseman.
- bpo-25198 [https://bugs.python.org/issue?@action=redirect&bpo=25198]: Enhance the initial html viewer now used for Idle Help. Properly indent fixed-pitch text (patch by Mark Roseman). Give code snippet a very Sphinx-like light blueish-gray background. Re-use initial width and height set by users for shell and editor. When the Table of Contents (TOC) menu is used, put the section header at the top of the screen.
- bpo-25225 [https://bugs.python.org/issue?@action=redirect&bpo=25225]: Condense and rewrite Idle doc section on text colors.
- bpo-21995 [https://bugs.python.org/issue?@action=redirect&bpo=21995]: Explain some differences between IDLE and console Python.
- bpo-22820 [https://bugs.python.org/issue?@action=redirect&bpo=22820]: Explain need for *print* when running file from Idle editor.

- bpo-25224 [https://bugs.python.org/issue?@action=redirect&bpo=25224]: Doc: augment Idle feature list and no-subprocess section.
- bpo-25219 [https://bugs.python.org/issue?@action=redirect&bpo=25219]: Update doc for Idle command line options. Some were missing and notes were not correct.
- bpo-24861 [https://bugs.python.org/issue?@action=redirect&bpo=24861]: Most of idlelib is private and subject to change. Use idleib.idle.* to start Idle. See idlelib.__init__.__doc__.
- bpo-25199 [https://bugs.python.org/issue?@action=redirect&bpo=25199]: Idle: add synchronization comments for future maintainers.
- bpo-16893 [https://bugs.python.org/issue?@action=redirect&bpo=16893]: Replace help.txt with help.html for Idle doc display. The new idlelib/help.html is rstripped Doc/build/html/library/idle.html. It looks better than help.txt and will better document Idle as released. The tkinter html viewer that works for this file was written by Rose Roseman. The now unused EditorWindow.HelpDialog class and helt.txt file are deprecated.
- bpo-24199 [https://bugs.python.org/issue?@action=redirect&bpo=24199]: Deprecate unused idlelib.idlever with possible removal in 3.6.
- bpo-24790 [https://bugs.python.org/issue?@action=redirect&bpo=24790]: Remove extraneous code (which also create 2 & 3 conflicts).

Documentation

- bpo-26736 [https://bugs.python.org/issue?@action=redirect&bpo=26736]: Used HTTPS for external links in the documentation if possible.
- bpo-6953 [https://bugs.python.org/issue?@action=redirect&bpo=6953]: Rework the Readline module documentation to group related functions together, and add more details such as what underlying Readline functions and variables are accessed.
- bpo-23606 [https://bugs.python.org/issue?@action=redirect&bpo=23606]: Adds note to ctypes documentation regarding cdll.msvcrt.

- bpo-24952 [https://bugs.python.org/issue?@action=redirect&bpo=24952]: Clarify the default size argument of stack_size() in the «threading» and «_thread» modules. Patch from Mattip.
- bpo-26014 [https://bugs.python.org/issue?@action=redirect&bpo=26014]: Update 3.x packaging documentation: * «See also» links to the new docs are now provided in the legacy pages * links to setuptools documentation have been updated

Tests

- bpo-21916 [https://bugs.python.org/issue?@action=redirect&bpo=21916]: Added tests for the turtle module. Patch by ingrid, Gregory Loyse and Jelle Zijlstra.
- bpo-26295 [https://bugs.python.org/issue?@action=redirect&bpo=26295]: When using «python3 -m test –testdir=TESTDIR», regrtest doesn't add «test.» prefix to test module names.
- bpo-26523 [https://bugs.python.org/issue?@action=redirect&bpo=26523]: The multiprocessing thread pool (multiprocessing.dummy.Pool) was untested.
- bpo-26015 [https://bugs.python.org/issue?@action=redirect&bpo=26015]: Added new tests for pickling iterators of mutable sequences.
- bpo-26325 [https://bugs.python.org/issue?@action=redirect&bpo=26325]: Added test.support.check_no_resource_warning() to check that no ResourceWarning is emitted.
- bpo-25940 [https://bugs.python.org/issue?@action=redirect&bpo=25940]: Changed test_ssl to use its internal local server more. This avoids relying on svn.python.org, which recently changed root certificate.
- bpo-25616 [https://bugs.python.org/issue?@action=redirect&bpo=25616]: Tests for OrderedDict are extracted from test_collections into separate file test_ordered_dict.
- bpo-25449 [https://bugs.python.org/issue?@action=redirect&bpo=25449]: Added tests for OrderedDict subclasses.

- bpo-25188 [https://bugs.python.org/issue?@action=redirect&bpo=25188]: Add -P/–pgo to test.regrtest to suppress error output when running the test suite for the purposes of a PGO build. Initial patch by Alecsandru Patrascu.
- bpo-22806 [https://bugs.python.org/issue?@action=redirect&bpo=22806]: Add `python -m test --list-tests` command to list tests.
- bpo-18174 [https://bugs.python.org/issue?@action=redirect&bpo=18174]: `python -m test --huntrleaks ...` now also checks for leak of file descriptors. Patch written by Richard Oudkerk.
- bpo-25260 [https://bugs.python.org/issue?@action=redirect&bpo=25260]: Fix `python -m test --coverage` on Windows. Remove the list of ignored directories.
- `PCbuild\rt.bat` now accepts an unlimited number of arguments to pass along to regrtest.py. Previously there was a limit of 9.
- bpo-26583 [https://bugs.python.org/issue?@action=redirect&bpo=26583]: Skip test_timestamp_overflow in test_import if bytecode files cannot be written.

Build

- bpo-21277 [https://bugs.python.org/issue?@action=redirect&bpo=21277]: Don’t try to link _ctypes with a ffi_convenience library.
- bpo-26884 [https://bugs.python.org/issue?@action=redirect&bpo=26884]: Fix linking extension modules for cross builds. Patch by Xavier de Gaye.
- bpo-26932 [https://bugs.python.org/issue?@action=redirect&bpo=26932]: Fixed support of RTLD_* constants defined as enum values, not via macros (in particular on Android). Patch by Chi Hsuan Yen.
- bpo-22359 [https://bugs.python.org/issue?@action=redirect&bpo=22359]: Disable the rules for running _freeze_importlib and pgen when cross-compiling. The output of these programs is normally saved with the source code anyway, and is still regenerated when doing a native build.

Patch by Xavier de Gaye.

- bpo-21668 [https://bugs.python.org/issue?@action=redirect&bpo=21668]: Link audioop, _datetime, _ctypes_test modules to libm, except on Mac OS X. Patch written by Chi Hsuan Yen.
- bpo-25702 [https://bugs.python.org/issue?@action=redirect&bpo=25702]: A –with-lto configure option has been added that will enable link time optimizations at build time during a make profile-opt. Some compilers and toolchains are known to not produce stable code when using LTO, be sure to test things thoroughly before relying on it. It can provide a few % speed up over profile-opt alone.
- bpo-26624 [https://bugs.python.org/issue?@action=redirect&bpo=26624]: Adds validation of ucrtbase[d].dll version with warning for old versions.
- bpo-17603 [https://bugs.python.org/issue?@action=redirect&bpo=17603]: Avoid error about nonexistent fileblocks.o file by using a lower-level check for st_blocks in struct stat.
- bpo-26079 [https://bugs.python.org/issue?@action=redirect&bpo=26079]: Fixing the build output folder for tix-8.4.3.6. Patch by Bjoern Thiel.
- bpo-26465 [https://bugs.python.org/issue?@action=redirect&bpo=26465]: Update Windows builds to use OpenSSL 1.0.2g.
- bpo-25348 [https://bugs.python.org/issue?@action=redirect&bpo=25348]: Added `--pgo` and `--pgo-job` arguments to `PCbuild\build.bat` for building with Profile-Guided Optimization. The old `PCbuild\build_pgo.bat` script is removed.
- bpo-25827 [https://bugs.python.org/issue?@action=redirect&bpo=25827]: Add support for building with ICC to `configure`, including a new `--with-icc` flag.
- bpo-25696 [https://bugs.python.org/issue?@action=redirect&bpo=25696]: Fix installation of Python on UNIX with make -j9.
- bpo-24986 [https://bugs.python.org/issue?@action=redirect&bpo=24986]: It is now possible to build Python on Windows without errors when external libraries are not available.

- bpo-24421 [https://bugs.python.org/issue?@action=redirect&bpo=24421]: Compile Modules/_math.c once, before building extensions. Previously it could fail to compile properly if the math and cmath builds were concurrent.
- bpo-26465 [https://bugs.python.org/issue?@action=redirect&bpo=26465]: Update OS X 10.5+ 32-bit-only installer to build and link with OpenSSL 1.0.2g.
- bpo-26268 [https://bugs.python.org/issue?@action=redirect&bpo=26268]: Update Windows builds to use OpenSSL 1.0.2f.
- bpo-25136 [https://bugs.python.org/issue?@action=redirect&bpo=25136]: Support Apple Xcode 7's new textual SDK stub libraries.
- bpo-24324 [https://bugs.python.org/issue?@action=redirect&bpo=24324]: Do not enable unreachable code warnings when using gcc as the option does not work correctly in older versions of gcc and has been silently removed as of gcc-4.5.

Windows

- bpo-27053 [https://bugs.python.org/issue?@action=redirect&bpo=27053]: Updates make_zip.py to correctly generate library ZIP file.
- bpo-26268 [https://bugs.python.org/issue?@action=redirect&bpo=26268]: Update the prepare_ssl.py script to handle OpenSSL releases that don't include the contents of the include directory (that is, 1.0.2e and later).
- bpo-26071 [https://bugs.python.org/issue?@action=redirect&bpo=26071]: bdist_wininst created binaries fail to start and find 32bit Python
- bpo-26073 [https://bugs.python.org/issue?@action=redirect&bpo=26073]: Update the list of magic numbers in launcher
- bpo-26065 [https://bugs.python.org/issue?@action=redirect&bpo=26065]: Excludes venv from library when generating embeddable distro.
- bpo-25022 [https://bugs.python.org/issue?@action=redirect&bpo=25022]: Removed very outdated PC/example_nt/ directory.

Tools/Demos

- bpo-26799 [https://bugs.python.org/issue?@action=redirect&bpo=26799]: Fix python-gdb.py: don't get C types once when the Python code is loaded, but get C types on demand. The C types can change if python-gdb.py is loaded before the Python executable. Patch written by Thomas Ilsche.
- bpo-26271 [https://bugs.python.org/issue?@action=redirect&bpo=26271]: Fix the Freeze tool to properly use flags passed through configure. Patch by Daniel Shaulov.
- bpo-26489 [https://bugs.python.org/issue?@action=redirect&bpo=26489]: Add dictionary unpacking support to Tools/parser/unparse.py. Patch by Guo Ci Teo.
- bpo-26316 [https://bugs.python.org/issue?@action=redirect&bpo=26316]: Fix variable name typo in Argument Clinic.
- bpo-25440 [https://bugs.python.org/issue?@action=redirect&bpo=25440]: Fix output of python-config –extension-suffix.
- bpo-25154 [https://bugs.python.org/issue?@action=redirect&bpo=25154]: The pyvenv script has been deprecated in favour of `python3 -m venv`.

C API

- bpo-26312 [https://bugs.python.org/issue?@action=redirect&bpo=26312]: SystemError is now raised in all programming bugs with using PyArg_ParseTupleAndKeywords(). RuntimeError did raised before in some programming bugs.
- bpo-26198 [https://bugs.python.org/issue?@action=redirect&bpo=26198]: ValueError is now raised instead of TypeError on buffer overflow in parsing «es#» and «et#» format units. SystemError is now raised instead of TypeError on programmatical error in parsing format string.

Python 3.5.5 final

Release date: 2018-02-04

There were no new changes in version 3.5.5.

Python 3.5.5 release candidate 1

Release date: 2018-01-23

Security

- bpo-32551 [https://bugs.python.org/issue?@action=redirect&bpo=32551]: The `sys.path[0]` initialization change for bpo-29139 [https://bugs.python.org/issue?@action=redirect&bpo=29139] caused a regression by revealing an inconsistency in how sys.path is initialized when executing `__main__` from a zipfile, directory, or other import location. This is considered a potential security issue, as it may lead to privileged processes unexpectedly loading code from user controlled directories in situations where that was not previously the case. The interpreter now consistently avoids ever adding the import location's parent directory to `sys.path`, and ensures no other `sys.path` entries are inadvertently modified when inserting the import location named on the command line. (Originally reported as bpo-29723 [https://bugs.python.org/issue?@action=redirect&bpo=29723] against Python 3.6rc1, but it was missed at the time that the then upcoming Python 3.5.4 release would also be affected)
- bpo-30657 [https://bugs.python.org/issue?@action=redirect&bpo=30657]: Fixed possible integer overflow in PyBytes_DecodeEscape, **CVE 2017-1000158** [https://www.cve.org/CVERecord?id=CVE-2017-1000158]. Original patch by Jay Bosamiya; rebased to Python 3 by Miro Hrončok.
- bpo-30947 [https://bugs.python.org/issue?@action=redirect&bpo=30947]: Upgrade libexpat embedded copy from version 2.2.1 to 2.2.3 to get security fixes.

Core and Builtins

- bpo-31095 [https://bugs.python.org/issue?@action=redirect&bpo=31095]: Fix potential crash during GC caused by `tp_dealloc` which doesn't call `PyObject_GC_UnTrack()`.

Library

- bpo-32072 [https://bugs.python.org/issue?@action=redirect&bpo=32072]: Fixed issues with binary plists: Fixed saving bytearrays. Identical objects will be saved only once. Equal references will be load as identical objects. Added support for saving and loading recursive data structures.
- bpo-31170 [https://bugs.python.org/issue?@action=redirect&bpo=31170]: expat: Update libexpat from 2.2.3 to 2.2.4. Fix copying of partial characters for UTF-8 input (libexpat bug 115): https://github.com/libexpat/libexpat/issues/115

Python 3.5.4 final

Release date: 2017-08-07

Library

- bpo-30119 [https://bugs.python.org/issue?@action=redirect&bpo=30119]: ftplib.FTP.putline() now throws ValueError on commands that contains CR or LF. Patch by Donghee Na.

Python 3.5.4 release candidate 1

Release date: 2017-07-23

Security

- bpo-30730 [https://bugs.python.org/issue?@action=redirect&bpo=30730]: Prevent environment variables injection in subprocess on Windows. Prevent passing other environment variables and command arguments.

- bpo-30694 [https://bugs.python.org/issue?@action=redirect&bpo=30694]: Upgrade expat copy from 2.2.0 to 2.2.1 to get fixes of multiple security vulnerabilities including: **CVE 2017-9233** [https://www.cve.org/CVERecord?id=CVE-2017-9233] (External entity infinite loop DoS), **CVE 2016-9063** [https://www.cve.org/CVERecord?id=CVE-2016-9063] (Integer overflow, re-fix), **CVE 2016-0718** [https://www.cve.org/CVERecord?id=CVE-2016-0718] (Fix regression bugs from 2.2.0's fix to **CVE 2016-0718** [https://www.cve.org/CVERecord?id=CVE-2016-0718]) and **CVE 2012-0876** [https://www.cve.org/CVERecord?id=CVE-2012-0876] (Counter hash flooding with SipHash). Note: the **CVE 2016-5300** [https://www.cve.org/CVERecord?id=CVE-2016-5300] (Use os-specific entropy sources like getrandom) doesn't impact Python, since Python already gets entropy from the OS to set the expat secret using `XML_SetHashSalt()`.
- bpo-30500 [https://bugs.python.org/issue?@action=redirect&bpo=30500]: Fix urllib.parse.splithost() to correctly parse fragments. For example, `splithost('//127.0.0.1#@evil.com/')` now correctly returns the `127.0.0.1` host, instead of treating `@evil.com` as the host in an authentication (`login@host`).
- bpo-29591 [https://bugs.python.org/issue?@action=redirect&bpo=29591]: Update expat copy from 2.1.1 to 2.2.0 to get fixes of **CVE 2016-0718** [https://www.cve.org/CVERecord?id=CVE-2016-0718] and **CVE 2016-4472** [https://www.cve.org/CVERecord?id=CVE-2016-4472]. See https://sourceforge.net/p/expat/bugs/537/ for more information.

Core and Builtins

- bpo-30876 [https://bugs.python.org/issue?@action=redirect&bpo=30876]: Relative import from unloaded package now reimports the package instead of failing with SystemError. Relative import from non-package now fails with ImportError rather than SystemError.
- bpo-30765 [https://bugs.python.org/issue?@action=redirect&bpo=30765]: Avoid blocking in pthread_mutex_lock() when PyThread_acquire_lock() is asked

not to block.

- bpo-27945 [https://bugs.python.org/issue?@action=redirect&bpo=27945]: Fixed various segfaults with dict when input collections are mutated during searching, inserting or comparing. Based on patches by Duane Griffin and Tim Mitchell.
- bpo-25794 [https://bugs.python.org/issue?@action=redirect&bpo=25794]: Fixed type.__setattr__() and type.__delattr__() for non-interned attribute names. Based on patch by Eryk Sun.
- bpo-29935 [https://bugs.python.org/issue?@action=redirect&bpo=29935]: Fixed error messages in the index() method of tuple, list and deque when pass indices of wrong type.
- bpo-28876 [https://bugs.python.org/issue?@action=redirect&bpo=28876]: `bool(range)` works even if `len(range)` raises **`OverflowError`**.
- bpo-29600 [https://bugs.python.org/issue?@action=redirect&bpo=29600]: Fix wrapping coroutine return values in StopIteration.
- bpo-29537 [https://bugs.python.org/issue?@action=redirect&bpo=29537]: Restore runtime compatibility with bytecode files generated by CPython 3.5.0 to 3.5.2, and adjust the eval loop to avoid the problems that could be caused by the malformed variant of the BUILD_MAP_UNPACK_WITH_CALL opcode that they may contain. Patch by Petr Viktorin, Serhiy Storchaka, and Nick Coghlan.
- bpo-28598 [https://bugs.python.org/issue?@action=redirect&bpo=28598]: Support __rmod__ for subclasses of str being called before str.__mod__. Patch by Martijn Pieters.
- bpo-29602 [https://bugs.python.org/issue?@action=redirect&bpo=29602]: Fix incorrect handling of signed zeros in complex constructor for complex subclasses and for inputs having a __complex__ method. Patch by Serhiy Storchaka.
- bpo-29347 [https://bugs.python.org/issue?@action=redirect&bpo=29347]: Fixed possibly dereferencing undefined pointers when creating weakref objects.
- bpo-29438 [https://bugs.python.org/issue?

@action=redirect&bpo=29438]: Fixed use-after-free problem in key sharing dict.
- bpo-29319 [https://bugs.python.org/issue?@action=redirect&bpo=29319]: Prevent RunMainFromImporter overwriting sys.path[0].
- bpo-29337 [https://bugs.python.org/issue?@action=redirect&bpo=29337]: Fixed possible BytesWarning when compare the code objects. Warnings could be emitted at compile time.
- bpo-29478 [https://bugs.python.org/issue?@action=redirect&bpo=29478]: If max_line_length=None is specified while using the Compat32 policy, it is no longer ignored. Patch by Mircea Cosbuc.

Library

- bpo-29403 [https://bugs.python.org/issue?@action=redirect&bpo=29403]: Fix `unittest.mock`'s autospec to not fail on method-bound builtin functions. Patch by Aaron Gallagher.
- bpo-30961 [https://bugs.python.org/issue?@action=redirect&bpo=30961]: Fix decrementing a borrowed reference in tracemalloc.
- bpo-30886 [https://bugs.python.org/issue?@action=redirect&bpo=30886]: Fix multiprocessing.Queue.join_thread(): it now waits until the thread completes, even if the thread was started by the same process which created the queue.
- bpo-29854 [https://bugs.python.org/issue?@action=redirect&bpo=29854]: Fix segfault in readline when using readline's history-size option. Patch by Nir Soffer.
- bpo-30807 [https://bugs.python.org/issue?@action=redirect&bpo=30807]: signal.setitimer() may disable the timer when passed a tiny value. Tiny values (such as 1e-6) are valid non-zero values for setitimer(), which is specified as taking microsecond-resolution intervals. However, on some platform, our conversion routine could convert 1e-6 into a zero interval, therefore disabling the timer instead of (re-)scheduling it.
- bpo-30441 [https://bugs.python.org/issue?

@action=redirect&bpo=30441]: Fix bug when modifying os.environ while iterating over it

- bpo-30532 [https://bugs.python.org/issue?@action=redirect&bpo=30532]: Fix email header value parser dropping folding white space in certain cases.
- bpo-29169 [https://bugs.python.org/issue?@action=redirect&bpo=29169]: Update zlib to 1.2.11.
- bpo-30879 [https://bugs.python.org/issue?@action=redirect&bpo=30879]: os.listdir() and os.scandir() now emit bytes names when called with bytes-like argument.
- bpo-30746 [https://bugs.python.org/issue?@action=redirect&bpo=30746]: Prohibited the "=" character in environment variable names in `os.putenv()` and `os.spawn*()`.
- bpo-29755 [https://bugs.python.org/issue?@action=redirect&bpo=29755]: Fixed the lgettext() family of functions in the gettext module. They now always return bytes.
- bpo-30645 [https://bugs.python.org/issue?@action=redirect&bpo=30645]: Fix path calculation in imp.load_package(), fixing it for cases when a package is only shipped with bytecodes. Patch by Alexandru Ardelean.
- bpo-23890 [https://bugs.python.org/issue?@action=redirect&bpo=23890]: unittest.TestCase.assertRaises() now manually breaks a reference cycle to not keep objects alive longer than expected.
- bpo-30149 [https://bugs.python.org/issue?@action=redirect&bpo=30149]: inspect.signature() now supports callables with variable-argument parameters wrapped with partialmethod. Patch by Donghee Na.
- bpo-29931 [https://bugs.python.org/issue?@action=redirect&bpo=29931]: Fixed comparison check for ipaddress.ip_interface objects. Patch by Sanjay Sundaresan.
- bpo-24484 [https://bugs.python.org/issue?@action=redirect&bpo=24484]: Avoid race condition in multiprocessing cleanup.
- bpo-28994 [https://bugs.python.org/issue?@action=redirect&bpo=28994]: The traceback no longer displayed for SystemExit raised in a callback registered by atexit.
- bpo-30508 [https://bugs.python.org/issue?

@action=redirect&bpo=30508]: Don't log exceptions if Task/Future «cancel()» method was called.

- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Updates to typing module: Add generic AsyncContextManager, add support for ContextManager on all versions. Original PRs by Jelle Zijlstra and Ivan Levkivskyi
- bpo-29870 [https://bugs.python.org/issue?@action=redirect&bpo=29870]: Fix ssl sockets leaks when connection is aborted in asyncio/ssl implementation. Patch by Michaël Sghaïer.
- bpo-29743 [https://bugs.python.org/issue?@action=redirect&bpo=29743]: Closing transport during handshake process leaks open socket. Patch by Nikolay Kim
- bpo-27585 [https://bugs.python.org/issue?@action=redirect&bpo=27585]: Fix waiter cancellation in asyncio.Lock. Patch by Mathieu Sornay.
- bpo-30418 [https://bugs.python.org/issue?@action=redirect&bpo=30418]: On Windows, subprocess.Popen.communicate() now also ignore EINVAL on stdin.write() if the child process is still running but closed the pipe.
- bpo-30378 [https://bugs.python.org/issue?@action=redirect&bpo=30378]: Fix the problem that logging.handlers.SysLogHandler cannot handle IPv6 addresses.
- bpo-29960 [https://bugs.python.org/issue?@action=redirect&bpo=29960]: Preserve generator state when _random.Random.setstate() raises an exception. Patch by Bryan Olson.
- bpo-30414 [https://bugs.python.org/issue?@action=redirect&bpo=30414]: multiprocessing.Queue._feed background running thread do not break from main loop on exception.
- bpo-30003 [https://bugs.python.org/issue?@action=redirect&bpo=30003]: Fix handling escape characters in HZ codec. Based on patch by Ma Lin.
- bpo-30301 [https://bugs.python.org/issue?@action=redirect&bpo=30301]: Fix AttributeError when using SimpleQueue.empty() under *spawn* and *forkserver* start

methods.

- bpo-30329 [https://bugs.python.org/issue?@action=redirect&bpo=30329]: imaplib and poplib now catch the Windows socket WSAEINVAL error (code 10022) on shutdown(SHUT_RDWR): An invalid operation was attempted. This error occurs sometimes on SSL connections.
- bpo-30375 [https://bugs.python.org/issue?@action=redirect&bpo=30375]: Warnings emitted when compile a regular expression now always point to the line in the user code. Previously they could point into inners of the re module if emitted from inside of groups or conditionals.
- bpo-30048 [https://bugs.python.org/issue?@action=redirect&bpo=30048]: Fixed `Task.cancel()` can be ignored when the task is running coroutine and the coroutine returned without any more `await`.
- bpo-29990 [https://bugs.python.org/issue?@action=redirect&bpo=29990]: Fix range checking in GB18030 decoder. Original patch by Ma Lin.
- bpo-26293 [https://bugs.python.org/issue?@action=redirect&bpo=26293]: Change resulted because of zipfile breakage. (See also: bpo-29094 [https://bugs.python.org/issue?@action=redirect&bpo=29094])
- bpo-30243 [https://bugs.python.org/issue?@action=redirect&bpo=30243]: Removed the _init_ methods of _json's scanner and encoder. Misusing them could cause memory leaks or crashes. Now scanner and encoder objects are completely initialized in the _new_ methods.
- bpo-30185 [https://bugs.python.org/issue?@action=redirect&bpo=30185]: Avoid KeyboardInterrupt tracebacks in forkserver helper process when Ctrl-C is received.
- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Various updates to typing module: add typing.NoReturn type, use WrapperDescriptorType, minor bug-fixes. Original PRs by Jim Fasarakis-Hilliard and Ivan Levkivskyi.
- bpo-30205 [https://bugs.python.org/issue?@action=redirect&bpo=30205]: Fix getsockname() for unbound AF_UNIX sockets on Linux.
- bpo-30070 [https://bugs.python.org/issue?

@action=redirect&bpo=30070]: Fixed leaks and crashes in errors handling in the parser module.

- bpo-30061 [https://bugs.python.org/issue?@action=redirect&bpo=30061]: Fixed crashes in IOBase methods __next__() and readlines() when readline() or __next__() respectively return non-sizeable object. Fixed possible other errors caused by not checking results of PyObject_Size(), PySequence_Size(), or PyMapping_Size().
- bpo-30068 [https://bugs.python.org/issue?@action=redirect&bpo=30068]: _io._IOBase.readlines will check if it's closed first when hint is present.
- bpo-29694 [https://bugs.python.org/issue?@action=redirect&bpo=29694]: Fixed race condition in pathlib mkdir with flags parents=True. Patch by Armin Rigo.
- bpo-29692 [https://bugs.python.org/issue?@action=redirect&bpo=29692]: Fixed arbitrary unchaining of RuntimeError exceptions in contextlib.contextmanager. Patch by Siddharth Velankar.
- bpo-29998 [https://bugs.python.org/issue?@action=redirect&bpo=29998]: Pickling and copying ImportError now preserves name and path attributes.
- bpo-29942 [https://bugs.python.org/issue?@action=redirect&bpo=29942]: Fix a crash in itertools.chain.from_iterable when encountering long runs of empty iterables.
- bpo-27863 [https://bugs.python.org/issue?@action=redirect&bpo=27863]: Fixed multiple crashes in ElementTree caused by race conditions and wrong types.
- bpo-28699 [https://bugs.python.org/issue?@action=redirect&bpo=28699]: Fixed a bug in pools in multiprocessing.pool that raising an exception at the very first of an iterable may swallow the exception or make the program hang. Patch by Davin Potts and Xiang Zhang.
- bpo-25803 [https://bugs.python.org/issue?@action=redirect&bpo=25803]: Avoid incorrect errors raised by Path.mkdir(exist_ok=True) when the OS gives priority to errors such as EACCES over EEXIST.
- bpo-29861 [https://bugs.python.org/issue?@action=redirect&bpo=29861]: Release references to tasks, their arguments and their results as soon as they are finished in

multiprocessing.Pool.

- bpo-29884 [https://bugs.python.org/issue?@action=redirect&bpo=29884]: faulthandler: Restore the old sigaltstack during teardown. Patch by Christophe Zeitouny.
- bpo-25455 [https://bugs.python.org/issue?@action=redirect&bpo=25455]: Fixed crashes in repr of recursive buffered file-like objects.
- bpo-29800 [https://bugs.python.org/issue?@action=redirect&bpo=29800]: Fix crashes in partial.__repr__ if the keys of partial.keywords are not strings. Patch by Michael Seifert.
- bpo-29742 [https://bugs.python.org/issue?@action=redirect&bpo=29742]: get_extra_info() raises exception if get called on closed ssl transport. Patch by Nikolay Kim.
- bpo-8256 [https://bugs.python.org/issue?@action=redirect&bpo=8256]: Fixed possible failing or crashing input() if attributes «encoding» or «errors» of sys.stdin or sys.stdout are not set or are not strings.
- bpo-28298 [https://bugs.python.org/issue?@action=redirect&bpo=28298]: Fix a bug that prevented array "Q", "L" and "I" from accepting big intables (objects that have __int__) as elements. Patch by Oren Milman.
- bpo-29615 [https://bugs.python.org/issue?@action=redirect&bpo=29615]: SimpleXMLRPCDispatcher no longer chains KeyError (or any other exception) to exception(s) raised in the dispatched methods. Patch by Petr Motejlek.
- bpo-29704 [https://bugs.python.org/issue?@action=redirect&bpo=29704]: asyncio.subprocess.SubprocessStreamProtocol no longer closes before all pipes are closed.
- bpo-29703 [https://bugs.python.org/issue?@action=redirect&bpo=29703]: Fix asyncio to support instantiation of new event loops in child processes.
- bpo-29376 [https://bugs.python.org/issue?@action=redirect&bpo=29376]: Fix assertion error in threading._DummyThread.is_alive().
- bpo-29110 [https://bugs.python.org/issue?@action=redirect&bpo=29110]: Fix file object leak in aifc.open() when file is given as a filesystem path and is not in valid AIFF

format. Patch by Anthony Zhang.

- bpo-28961 [https://bugs.python.org/issue?@action=redirect&bpo=28961]: Fix unittest.mock._Call helper: don’t ignore the name parameter anymore. Patch written by Jiajun Huang.
- bpo-29532 [https://bugs.python.org/issue?@action=redirect&bpo=29532]: Altering a kwarg dictionary passed to functools.partial() no longer affects a partial object after creation.
- bpo-28556 [https://bugs.python.org/issue?@action=redirect&bpo=28556]: Various updates to typing module: typing.Counter, typing.ChainMap, improved ABC caching, etc. Original PRs by Jelle Zijlstra, Ivan Levkivskyi, Manuel Krebber, and Łukasz Langa.
- bpo-29100 [https://bugs.python.org/issue?@action=redirect&bpo=29100]: Fix datetime.fromtimestamp() regression introduced in Python 3.6.0: check minimum and maximum years.
- bpo-29519 [https://bugs.python.org/issue?@action=redirect&bpo=29519]: Fix weakref spewing exceptions during interpreter shutdown when used with a rare combination of multiprocessing and custom codecs.
- bpo-29416 [https://bugs.python.org/issue?@action=redirect&bpo=29416]: Prevent infinite loop in pathlib.Path.mkdir
- bpo-29444 [https://bugs.python.org/issue?@action=redirect&bpo=29444]: Fixed out-of-bounds buffer access in the group() method of the match object. Based on patch by WGH.
- bpo-29335 [https://bugs.python.org/issue?@action=redirect&bpo=29335]: Fix subprocess.Popen.wait() when the child process has exited to a stopped instead of terminated state (ex: when under ptrace).
- bpo-29290 [https://bugs.python.org/issue?@action=redirect&bpo=29290]: Fix a regression in argparse that help messages would wrap at non-breaking spaces.
- bpo-28735 [https://bugs.python.org/issue?@action=redirect&bpo=28735]: Fixed the comparison of mock.MagickMock with mock.ANY.
- bpo-29011 [https://bugs.python.org/issue?

@action=redirect&bpo=29011]: Fix an important omission by adding Deque to the typing module.
- bpo-29219 [https://bugs.python.org/issue?@action=redirect&bpo=29219]: Fixed infinite recursion in the repr of uninitialized ctypes.CDLL instances.
- bpo-28969 [https://bugs.python.org/issue?@action=redirect&bpo=28969]: Fixed race condition in C implementation of functools.lru_cache. KeyError could be raised when cached function with full cache was simultaneously called from different threads with the same uncached arguments.
- bpo-29142 [https://bugs.python.org/issue?@action=redirect&bpo=29142]: In urllib.request, suffixes in no_proxy environment variable with leading dots could match related hostnames again (e.g. .b.c matches a.b.c). Patch by Milan Oberkirch.

Documentation

- bpo-30176 [https://bugs.python.org/issue?@action=redirect&bpo=30176]: Add missing attribute related constants in curses documentation.
- bpo-26985 [https://bugs.python.org/issue?@action=redirect&bpo=26985]: Add missing info of code object in inspect documentation.
- bpo-28929 [https://bugs.python.org/issue?@action=redirect&bpo=28929]: Link the documentation to its source file on GitHub.
- bpo-25008 [https://bugs.python.org/issue?@action=redirect&bpo=25008]: Document smtpd.py as effectively deprecated and add a pointer to aiosmtpd, a third-party asyncio-based replacement.
- bpo-26355 [https://bugs.python.org/issue?@action=redirect&bpo=26355]: Add canonical header link on each page to corresponding major version of the documentation. Patch by Matthias Bussonnier.
- bpo-29349 [https://bugs.python.org/issue?@action=redirect&bpo=29349]: Fix Python 2 syntax in code for building the documentation.

Tests

- bpo-30822 [https://bugs.python.org/issue?@action=redirect&bpo=30822]: Fix regrtest command line parser to allow passing -u extralargefile to run test_zipfile64.
- bpo-30383 [https://bugs.python.org/issue?@action=redirect&bpo=30383]: regrtest: Enhance regrtest and backport features from the master branch. Add options: –coverage, –testdir, –list-tests (list test files, don't run them), –list-cases (list test identifiers, don't run them, bpo-30523 [https://bugs.python.org/issue?@action=redirect&bpo=30523]), –matchfile (load a list of test filters from a text file, bpo-30540 [https://bugs.python.org/issue?@action=redirect&bpo=30540]), –slowest (alias to –slow). Enhance output: add timestamp, test result, currently running tests, «Tests result: xxx» summary with total duration, etc. Fix reference leak hunting in regrtest, –huntrleaks: regrtest now warms up caches, create explicitly all internal singletons which are created on demand to prevent false positives when checking for reference leaks. (bpo-30675 [https://bugs.python.org/issue?@action=redirect&bpo=30675]).
- bpo-30357 [https://bugs.python.org/issue?@action=redirect&bpo=30357]: test_thread: setUp() now uses support.threading_setup() and support.threading_cleanup() to wait until threads complete to avoid random side effects on following tests. Initial patch written by Grzegorz Grzywacz.
- bpo-28087 [https://bugs.python.org/issue?@action=redirect&bpo=28087]: Skip test_asyncore and test_eintr poll failures on macOS. Skip some tests of select.poll when running on macOS due to unresolved issues with the underlying system poll function on some macOS versions.
- bpo-30197 [https://bugs.python.org/issue?@action=redirect&bpo=30197]: Enhanced functions swap_attr() and swap_item() in the test.support module. They now work when delete replaced attribute or item inside the with statement. The old value of the attribute or item (or None if it doesn't exist) now will be assigned to the target of the «as» clause, if there is one.
- bpo-29571 [https://bugs.python.org/issue?@action=redirect&bpo=29571]: to match the behaviour of the

`re.LOCALE` flag, test_re.test_locale_flag now uses `locale.getpreferredencoding(False)` to determine the candidate encoding for the test regex (allowing it to correctly skip the test when the default locale encoding is a multi-byte encoding)

Build

- bpo-29243 [https://bugs.python.org/issue?@action=redirect&bpo=29243]: Prevent unnecessary rebuilding of Python during `make test`, `make install` and some other make targets when configured with `--enable-optimizations`.
- bpo-23404 [https://bugs.python.org/issue?@action=redirect&bpo=23404]: Don't regenerate generated files based on file modification time anymore: the action is now explicit. Replace `make touch` with `make regen-all`.
- bpo-29643 [https://bugs.python.org/issue?@action=redirect&bpo=29643]: Fix `--enable-optimization` didn't work.

Windows

- bpo-30687 [https://bugs.python.org/issue?@action=redirect&bpo=30687]: Locate msbuild.exe on Windows when building rather than vcvarsall.bat
- bpo-29392 [https://bugs.python.org/issue?@action=redirect&bpo=29392]: Prevent crash when passing invalid arguments into msvcrt module.

C API

- bpo-27867 [https://bugs.python.org/issue?@action=redirect&bpo=27867]: Function PySlice_GetIndicesEx() is replaced with a macro if Py_LIMITED_API is set to the value between 0x03050400 and 0x03060000 (not including) or 0x03060100 or higher.
- bpo-29083 [https://bugs.python.org/issue?@action=redirect&bpo=29083]: Fixed the declaration of some public API functions. PyArg_VaParse() and

PyArg_VaParseTupleAndKeywords() were not available in limited API. PyArg_ValidateKeywordArguments(), PyArg_UnpackTuple() and Py_BuildValue() were not available in limited API of version < 3.3 when PY_SSIZE_T_CLEAN is defined.

Python 3.5.3 final

Release date: 2017-01-17

There were no code changes between 3.5.3rc1 and 3.5.3 final.

Python 3.5.3 release candidate 1

Release date: 2017-01-02

Security

- bpo-27278 [https://bugs.python.org/issue?@action=redirect&bpo=27278]: Fix os.urandom() implementation using getrandom() on Linux. Truncate size to INT_MAX and loop until we collected enough random bytes, instead of casting a directly Py_ssize_t to int.
- bpo-22636 [https://bugs.python.org/issue?@action=redirect&bpo=22636]: Avoid shell injection problems with ctypes.util.find_library().

Core and Builtins

- bpo-29073 [https://bugs.python.org/issue?@action=redirect&bpo=29073]: bytearray formatting no longer truncates on first null byte.
- bpo-28932 [https://bugs.python.org/issue?@action=redirect&bpo=28932]: Do not include <sys/random.h> if it does not exist.
- bpo-28147 [https://bugs.python.org/issue?@action=redirect&bpo=28147]: Fix a memory leak in split-table dictionaries: setattr() must not convert combined table into split table.

- bpo-25677 [https://bugs.python.org/issue?@action=redirect&bpo=25677]: Correct the positioning of the syntax error caret for indented blocks. Based on patch by Michael Layzell.
- bpo-29000 [https://bugs.python.org/issue?@action=redirect&bpo=29000]: Fixed bytes formatting of octals with zero padding in alternate form.
- bpo-28512 [https://bugs.python.org/issue?@action=redirect&bpo=28512]: Fixed setting the offset attribute of SyntaxError by PyErr_SyntaxLocationEx() and PyErr_SyntaxLocationObject().
- bpo-28991 [https://bugs.python.org/issue?@action=redirect&bpo=28991]: functools.lru_cache() was susceptible to an obscure reentrancy bug caused by a monkey-patched len() function.
- bpo-28648 [https://bugs.python.org/issue?@action=redirect&bpo=28648]: Fixed crash in Py_DecodeLocale() in debug build on Mac OS X when decode astral characters. Patch by Xiang Zhang.
- bpo-19398 [https://bugs.python.org/issue?@action=redirect&bpo=19398]: Extra slash no longer added to sys.path components in case of empty compile-time PYTHONPATH components.
- bpo-28426 [https://bugs.python.org/issue?@action=redirect&bpo=28426]: Fixed potential crash in PyUnicode_AsDecodedObject() in debug build.
- bpo-23782 [https://bugs.python.org/issue?@action=redirect&bpo=23782]: Fixed possible memory leak in _PyTraceback_Add() and exception loss in PyTraceBack_Here().
- bpo-28379 [https://bugs.python.org/issue?@action=redirect&bpo=28379]: Added sanity checks and tests for PyUnicode_CopyCharacters(). Patch by Xiang Zhang.
- bpo-28376 [https://bugs.python.org/issue?@action=redirect&bpo=28376]: The type of long range iterator is now registered as Iterator. Patch by Oren Milman.
- bpo-28376 [https://bugs.python.org/issue?@action=redirect&bpo=28376]: The constructor of range_iterator now checks that step is not 0. Patch by Oren Milman.
- bpo-26906 [https://bugs.python.org/issue?

@action=redirect&bpo=26906]: Resolving special methods of uninitialized type now causes implicit initialization of the type instead of a fail.

- bpo-18287 [https://bugs.python.org/issue?@action=redirect&bpo=18287]: PyType_Ready() now checks that tp_name is not NULL. Original patch by Niklas Koep.
- bpo-24098 [https://bugs.python.org/issue?@action=redirect&bpo=24098]: Fixed possible crash when AST is changed in process of compiling it.
- bpo-28350 [https://bugs.python.org/issue?@action=redirect&bpo=28350]: String constants with null character no longer interned.
- bpo-26617 [https://bugs.python.org/issue?@action=redirect&bpo=26617]: Fix crash when GC runs during weakref callbacks.
- bpo-27942 [https://bugs.python.org/issue?@action=redirect&bpo=27942]: String constants now interned recursively in tuples and frozensets.
- bpo-21578 [https://bugs.python.org/issue?@action=redirect&bpo=21578]: Fixed misleading error message when ImportError called with invalid keyword args.
- bpo-28203 [https://bugs.python.org/issue?@action=redirect&bpo=28203]: Fix incorrect type in error message from `complex(1.0, {2:3})`. Patch by Soumya Sharma.
- bpo-27955 [https://bugs.python.org/issue?@action=redirect&bpo=27955]: Fallback on reading /dev/urandom device when the getrandom() syscall fails with EPERM, for example when blocked by SECCOMP.
- bpo-28131 [https://bugs.python.org/issue?@action=redirect&bpo=28131]: Fix a regression in zipimport's compile_source(). zipimport should use the same optimization level as the interpreter.
- bpo-25221 [https://bugs.python.org/issue?@action=redirect&bpo=25221]: Fix corrupted result from PyLong_FromLong(0) when Python is compiled with NSMALLPOSINTS = 0.
- bpo-25758 [https://bugs.python.org/issue?@action=redirect&bpo=25758]: Prevents zipimport from unnecessarily encoding a filename (patch by Eryk Sun)

- bpo-28189 [https://bugs.python.org/issue?@action=redirect&bpo=28189]: dictitems_contains no longer swallows compare errors. (Patch by Xiang Zhang)
- bpo-27812 [https://bugs.python.org/issue?@action=redirect&bpo=27812]: Properly clear out a generator's frame's backreference to the generator to prevent crashes in frame.clear().
- bpo-27811 [https://bugs.python.org/issue?@action=redirect&bpo=27811]: Fix a crash when a coroutine that has not been awaited is finalized with warnings-as-errors enabled.
- bpo-27587 [https://bugs.python.org/issue?@action=redirect&bpo=27587]: Fix another issue found by PVS-Studio: Null pointer check after use of "def" in _PyState_AddModule(). Initial patch by Christian Heimes.
- bpo-26020 [https://bugs.python.org/issue?@action=redirect&bpo=26020]: set literal evaluation order did not match documented behaviour.
- bpo-27782 [https://bugs.python.org/issue?@action=redirect&bpo=27782]: Multi-phase extension module import now correctly allows the `m_methods` field to be used to add module level functions to instances of non-module types returned from `Py_create_mod`. Patch by Xiang Zhang.
- bpo-27936 [https://bugs.python.org/issue?@action=redirect&bpo=27936]: The round() function accepted a second None argument for some types but not for others. Fixed the inconsistency by accepting None for all numeric types.
- bpo-27487 [https://bugs.python.org/issue?@action=redirect&bpo=27487]: Warn if a submodule argument to «python -m» or runpy.run_module() is found in sys.modules after parent packages are imported, but before the submodule is executed.
- bpo-27558 [https://bugs.python.org/issue?@action=redirect&bpo=27558]: Fix a SystemError in the implementation of «raise» statement. In a brand new thread, raise a RuntimeError since there is no active exception to reraise. Patch written by Xiang Zhang.
- bpo-27419 [https://bugs.python.org/issue?

@action=redirect&bpo=27419]: Standard _import_() no longer look up «_import_» in globals or builtins for importing submodules or «from import». Fixed handling an error of non-string package name.

- bpo-27083 [https://bugs.python.org/issue?@action=redirect&bpo=27083]: Respect the PYTHONCASEOK environment variable under Windows.
- bpo-27514 [https://bugs.python.org/issue?@action=redirect&bpo=27514]: Make having too many statically nested blocks a SyntaxError instead of SystemError.
- bpo-27473 [https://bugs.python.org/issue?@action=redirect&bpo=27473]: Fixed possible integer overflow in bytes and bytearray concatenations. Patch by Xiang Zhang.
- bpo-27507 [https://bugs.python.org/issue?@action=redirect&bpo=27507]: Add integer overflow check in bytearray.extend(). Patch by Xiang Zhang.
- bpo-27581 [https://bugs.python.org/issue?@action=redirect&bpo=27581]: Don't rely on wrapping for overflow check in PySequence_Tuple(). Patch by Xiang Zhang.
- bpo-27443 [https://bugs.python.org/issue?@action=redirect&bpo=27443]: _length_hint_() of bytearray iterators no longer return a negative integer for a resized bytearray.
- bpo-27942 [https://bugs.python.org/issue?@action=redirect&bpo=27942]: Fix memory leak in codeobject.c

Library

- bpo-15812 [https://bugs.python.org/issue?@action=redirect&bpo=15812]: inspect.getframeinfo() now correctly shows the first line of a context. Patch by Sam Breese.
- bpo-29094 [https://bugs.python.org/issue?@action=redirect&bpo=29094]: Offsets in a ZIP file created with extern file object and modes «w» and «x» now are relative to the start of the file.
- bpo-13051 [https://bugs.python.org/issue?@action=redirect&bpo=13051]: Fixed recursion errors in large or resized curses.textpad.Textbox. Based on patch by Tycho Andersen.

- bpo-29119 [https://bugs.python.org/issue?@action=redirect&bpo=29119]: Fix weakrefs in the pure python version of collections.OrderedDict move_to_end() method. Contributed by Andra Bogildea.
- bpo-9770 [https://bugs.python.org/issue?@action=redirect&bpo=9770]: curses.ascii predicates now work correctly with negative integers.
- bpo-28427 [https://bugs.python.org/issue?@action=redirect&bpo=28427]: old keys should not remove new values from WeakValueDictionary when collecting from another thread.
- bpo-28923 [https://bugs.python.org/issue?@action=redirect&bpo=28923]: Remove editor artifacts from Tix.py.
- bpo-28871 [https://bugs.python.org/issue?@action=redirect&bpo=28871]: Fixed a crash when deallocate deep ElementTree.
- bpo-19542 [https://bugs.python.org/issue?@action=redirect&bpo=19542]: Fix bugs in WeakValueDictionary.setdefault() and WeakValueDictionary.pop() when a GC collection happens in another thread.
- bpo-20191 [https://bugs.python.org/issue?@action=redirect&bpo=20191]: Fixed a crash in resource.prlimit() when pass a sequence that doesn't own its elements as limits.
- bpo-28779 [https://bugs.python.org/issue?@action=redirect&bpo=28779]: multiprocessing.set_forkserver_preload() would crash the forkserver process if a preloaded module instantiated some multiprocessing objects such as locks.
- bpo-28847 [https://bugs.python.org/issue?@action=redirect&bpo=28847]: dbm.dumb now supports reading read-only files and no longer writes the index file when it is not changed.
- bpo-25659 [https://bugs.python.org/issue?@action=redirect&bpo=25659]: In ctypes, prevent a crash calling the from_buffer() and from_buffer_copy() methods on abstract classes like Array.
- bpo-28732 [https://bugs.python.org/issue?@action=redirect&bpo=28732]: Fix crash in os.spawnv() with no

elements in args

- bpo-28485 [https://bugs.python.org/issue?@action=redirect&bpo=28485]: Always raise ValueError for negative compileall.compile_dir(workers=…) parameter, even when multithreading is unavailable.
- bpo-28387 [https://bugs.python.org/issue?@action=redirect&bpo=28387]: Fixed possible crash in _io.TextIOWrapper deallocator when the garbage collector is invoked in other thread. Based on patch by Sebastian Cufre.
- bpo-27517 [https://bugs.python.org/issue?@action=redirect&bpo=27517]: LZMA compressor and decompressor no longer raise exceptions if given empty data twice. Patch by Benjamin Fogle.
- bpo-28549 [https://bugs.python.org/issue?@action=redirect&bpo=28549]: Fixed segfault in curses's addch() with ncurses6.
- bpo-28449 [https://bugs.python.org/issue?@action=redirect&bpo=28449]: tarfile.open() with mode «r» or «r:» now tries to open a tar file with compression before trying to open it without compression. Otherwise it had 50% chance failed with ignore_zeros=True.
- bpo-23262 [https://bugs.python.org/issue?@action=redirect&bpo=23262]: The webbrowser module now supports Firefox 36+ and derived browsers. Based on patch by Oleg Broytman.
- bpo-27939 [https://bugs.python.org/issue?@action=redirect&bpo=27939]: Fixed bugs in tkinter.ttk.LabeledScale and tkinter.Scale caused by representing the scale as float value internally in Tk. tkinter.IntVar now works if float value is set to underlying Tk variable.
- bpo-28255 [https://bugs.python.org/issue?@action=redirect&bpo=28255]: calendar.TextCalendar().prmonth() no longer prints a space at the start of new line after printing a month's calendar. Patch by Xiang Zhang.
- bpo-20491 [https://bugs.python.org/issue?@action=redirect&bpo=20491]: The textwrap.TextWrapper class now honors non-breaking spaces. Based on patch by Kaarle Ritvanen.

- bpo-28353 [https://bugs.python.org/issue?@action=redirect&bpo=28353]: os.fwalk() no longer fails on broken links.
- bpo-25464 [https://bugs.python.org/issue?@action=redirect&bpo=25464]: Fixed HList.header_exists() in tkinter.tix module by addin a workaround to Tix library bug.
- bpo-28488 [https://bugs.python.org/issue?@action=redirect&bpo=28488]: shutil.make_archive() no longer add entry «./» to ZIP archive.
- bpo-24452 [https://bugs.python.org/issue?@action=redirect&bpo=24452]: Make webbrowser support Chrome on Mac OS X.
- bpo-20766 [https://bugs.python.org/issue?@action=redirect&bpo=20766]: Fix references leaked by pdb in the handling of SIGINT handlers.
- bpo-26293 [https://bugs.python.org/issue?@action=redirect&bpo=26293]: Fixed writing ZIP files that starts not from the start of the file. Offsets in ZIP file now are relative to the start of the archive in conforming to the specification.
- bpo-28321 [https://bugs.python.org/issue?@action=redirect&bpo=28321]: Fixed writing non-BMP characters with binary format in plistlib.
- bpo-28322 [https://bugs.python.org/issue?@action=redirect&bpo=28322]: Fixed possible crashes when unpickle itertools objects from incorrect pickle data. Based on patch by John Leitch.
- Fix possible integer overflows and crashes in the mmap module with unusual usage patterns.
- bpo-1703178 [https://bugs.python.org/issue?@action=redirect&bpo=1703178]: Fix the ability to pass the –link-objects option to the distutils build_ext command.
- bpo-28253 [https://bugs.python.org/issue?@action=redirect&bpo=28253]: Fixed calendar functions for extreme months: 0001-01 and 9999-12. Methods itermonthdays() and itermonthdays2() are reimplemented so that they don’t call itermonthdates() which can cause datetime.date under/overflow.
- bpo-28275 [https://bugs.python.org/issue?@action=redirect&bpo=28275]: Fixed possible use after free in the

decompress() methods of the LZMADecompressor and BZ2Decompressor classes. Original patch by John Leitch.

- bpo-27897 [https://bugs.python.org/issue?@action=redirect&bpo=27897]: Fixed possible crash in sqlite3.Connection.create_collation() if pass invalid string-like object as a name. Patch by Xiang Zhang.
- bpo-18893 [https://bugs.python.org/issue?@action=redirect&bpo=18893]: Fix invalid exception handling in Lib/ctypes/macholib/dyld.py. Patch by Madison May.
- bpo-27611 [https://bugs.python.org/issue?@action=redirect&bpo=27611]: Fixed support of default root window in the tkinter.tix module.
- bpo-27348 [https://bugs.python.org/issue?@action=redirect&bpo=27348]: In the traceback module, restore the formatting of exception messages like «Exception: None». This fixes a regression introduced in 3.5a2.
- bpo-25651 [https://bugs.python.org/issue?@action=redirect&bpo=25651]: Allow false values to be used for msg parameter of subTest().
- bpo-27932 [https://bugs.python.org/issue?@action=redirect&bpo=27932]: Prevent memory leak in win32_ver().
- Fix UnboundLocalError in socket._sendfile_use_sendfile.
- bpo-28075 [https://bugs.python.org/issue?@action=redirect&bpo=28075]: Check for ERROR_ACCESS_DENIED in Windows implementation of os.stat(). Patch by Eryk Sun.
- bpo-25270 [https://bugs.python.org/issue?@action=redirect&bpo=25270]: Prevent codecs.escape_encode() from raising SystemError when an empty bytestring is passed.
- bpo-28181 [https://bugs.python.org/issue?@action=redirect&bpo=28181]: Get antigravity over HTTPS. Patch by Kaartic Sivaraam.
- bpo-25895 [https://bugs.python.org/issue?@action=redirect&bpo=25895]: Enable WebSocket URL schemes in urllib.parse.urljoin. Patch by Gergely Imreh and Markus Holtermann.
- bpo-27599 [https://bugs.python.org/issue?@action=redirect&bpo=27599]: Fixed buffer overrun in binascii.b2a_qp() and binascii.a2b_qp().

- bpo-19003 [https://bugs.python.org/issue?@action=redirect&bpo=19003]: m email.generator now replaces only `\r` and/or `\n` line endings, per the RFC, instead of all unicode line endings.
- bpo-28019 [https://bugs.python.org/issue?@action=redirect&bpo=28019]: itertools.count() no longer rounds non-integer step in range between 1.0 and 2.0 to 1.
- bpo-25969 [https://bugs.python.org/issue?@action=redirect&bpo=25969]: Update the lib2to3 grammar to handle the unpacking generalizations added in 3.5.
- bpo-14977 [https://bugs.python.org/issue?@action=redirect&bpo=14977]: mailcap now respects the order of the lines in the mailcap files («first match»), as required by RFC 1542. Patch by Michael Lazar.
- bpo-24594 [https://bugs.python.org/issue?@action=redirect&bpo=24594]: Validates persist parameter when opening MSI database
- bpo-17582 [https://bugs.python.org/issue?@action=redirect&bpo=17582]: xml.etree.ElementTree nows preserves whitespaces in attributes (Patch by Duane Griffin. Reviewed and approved by Stefan Behnel.)
- bpo-28047 [https://bugs.python.org/issue?@action=redirect&bpo=28047]: Fixed calculation of line length used for the base64 CTE in the new email policies.
- bpo-27445 [https://bugs.python.org/issue?@action=redirect&bpo=27445]: Don't pass str(_charset) to MIMEText.set_payload(). Patch by Claude Paroz.
- bpo-22450 [https://bugs.python.org/issue?@action=redirect&bpo=22450]: urllib now includes an `Accept: */*` header among the default headers. This makes the results of REST API requests more consistent and predictable especially when proxy servers are involved.
- lib2to3.pgen3.driver.load_grammar() now creates a stable cache file between runs given the same Grammar.txt input regardless of the hash randomization setting.
- bpo-27570 [https://bugs.python.org/issue?@action=redirect&bpo=27570]: Avoid zero-length memcpy() etc calls with null source pointers in the «ctypes» and «array» modules.
- bpo-22233 [https://bugs.python.org/issue?

@action=redirect&bpo=22233]: Break email header lines *only* on the RFC specified CR and LF characters, not on arbitrary unicode line breaks. This also fixes a bug in HTTP header parsing.

- bpo-27988 [https://bugs.python.org/issue?@action=redirect&bpo=27988]: Fix email iter_attachments incorrect mutation of payload list.
- bpo-27691 [https://bugs.python.org/issue?@action=redirect&bpo=27691]: Fix ssl module's parsing of GEN_RID subject alternative name fields in X.509 certs.
- bpo-27850 [https://bugs.python.org/issue?@action=redirect&bpo=27850]: Remove 3DES from ssl module's default cipher list to counter measure sweet32 attack (**CVE 2016-2183** [https://www.cve.org/CVERecord?id=CVE-2016-2183]).
- bpo-27766 [https://bugs.python.org/issue?@action=redirect&bpo=27766]: Add ChaCha20 Poly1305 to ssl module's default cipher list. (Required OpenSSL 1.1.0 or LibreSSL).
- bpo-26470 [https://bugs.python.org/issue?@action=redirect&bpo=26470]: Port ssl and hashlib module to OpenSSL 1.1.0.
- Remove support for passing a file descriptor to os.access. It never worked but previously didn't raise.
- bpo-12885 [https://bugs.python.org/issue?@action=redirect&bpo=12885]: Fix error when distutils encounters symlink.
- bpo-27881 [https://bugs.python.org/issue?@action=redirect&bpo=27881]: Fixed possible bugs when setting sqlite3.Connection.isolation_level. Based on patch by Xiang Zhang.
- bpo-27861 [https://bugs.python.org/issue?@action=redirect&bpo=27861]: Fixed a crash in sqlite3.Connection.cursor() when a factory creates not a cursor. Patch by Xiang Zhang.
- bpo-19884 [https://bugs.python.org/issue?@action=redirect&bpo=19884]: Avoid spurious output on OS X with Gnu Readline.
- bpo-27706 [https://bugs.python.org/issue?@action=redirect&bpo=27706]: Restore deterministic behavior of random.Random().seed() for string seeds using seeding

version 1. Allows sequences of calls to random() to exactly match those obtained in Python 2. Patch by Nofar Schnider.

- bpo-10513 [https://bugs.python.org/issue?@action=redirect&bpo=10513]: Fix a regression in Connection.commit(). Statements should not be reset after a commit.
- A new version of typing.py from https://github.com/python/typing: Collection (only for 3.6) (bpo-27598 [https://bugs.python.org/issue?@action=redirect&bpo=27598]). Add FrozenSet to _all_ (upstream #261). Fix crash in _get_type_vars() (upstream #259). Remove the dict constraint in ForwardRef._eval_type (upstream #252).
- bpo-27539 [https://bugs.python.org/issue?@action=redirect&bpo=27539]: Fix unnormalised `Fraction.__pow__` result in the case of negative exponent and negative base.
- bpo-21718 [https://bugs.python.org/issue?@action=redirect&bpo=21718]: cursor.description is now available for queries using CTEs.
- bpo-2466 [https://bugs.python.org/issue?@action=redirect&bpo=2466]: posixpath.ismount now correctly recognizes mount points which the user does not have permission to access.
- bpo-27773 [https://bugs.python.org/issue?@action=redirect&bpo=27773]: Correct some memory management errors server_hostname in _ssl.wrap_socket().
- bpo-26750 [https://bugs.python.org/issue?@action=redirect&bpo=26750]: unittest.mock.create_autospec() now works properly for subclasses of property() and other data descriptors.
- In the curses module, raise an error if window.getstr() or window.instr() is passed a negative value.
- bpo-27783 [https://bugs.python.org/issue?@action=redirect&bpo=27783]: Fix possible usage of uninitialized memory in operator.methodcaller.
- bpo-27774 [https://bugs.python.org/issue?@action=redirect&bpo=27774]: Fix possible Py_DECREF on unowned object in _sre.
- bpo-27760 [https://bugs.python.org/issue?@action=redirect&bpo=27760]: Fix possible integer overflow in binascii.b2a_qp.

- bpo-27758 [https://bugs.python.org/issue?@action=redirect&bpo=27758]: Fix possible integer overflow in the _csv module for large record lengths.
- bpo-27568 [https://bugs.python.org/issue?@action=redirect&bpo=27568]: Prevent HTTPoxy attack (**CVE 2016-1000110** [https://www.cve.org/CVERecord?id=CVE-2016-1000110]). Ignore the HTTP_PROXY variable when REQUEST_METHOD environment is set, which indicates that the script is in CGI mode.
- bpo-27656 [https://bugs.python.org/issue?@action=redirect&bpo=27656]: Do not assume sched.h defines any SCHED_* constants.
- bpo-27130 [https://bugs.python.org/issue?@action=redirect&bpo=27130]: In the «zlib» module, fix handling of large buffers (typically 4 GiB) when compressing and decompressing. Previously, inputs were limited to 4 GiB, and compression and decompression operations did not properly handle results of 4 GiB.
- bpo-27533 [https://bugs.python.org/issue?@action=redirect&bpo=27533]: Release GIL in nt._isdir
- bpo-17711 [https://bugs.python.org/issue?@action=redirect&bpo=17711]: Fixed unpickling by the persistent ID with protocol 0. Original patch by Alexandre Vassalotti.
- bpo-27522 [https://bugs.python.org/issue?@action=redirect&bpo=27522]: Avoid an unintentional reference cycle in email.feedparser.
- bpo-26844 [https://bugs.python.org/issue?@action=redirect&bpo=26844]: Fix error message for imp.find_module() to refer to “path” instead of “name”. Patch by Lev Maximov.
- bpo-23804 [https://bugs.python.org/issue?@action=redirect&bpo=23804]: Fix SSL zero-length recv() calls to not block and not raise an error about unclean EOF.
- bpo-27466 [https://bugs.python.org/issue?@action=redirect&bpo=27466]: Change time format returned by http.cookie.time2netscape, confirming the netscape cookie format and making it consistent with documentation.
- bpo-26664 [https://bugs.python.org/issue?@action=redirect&bpo=26664]: Fix activate.fish by removing mis-use of `$`.

- bpo-22115 [https://bugs.python.org/issue?@action=redirect&bpo=22115]: Fixed tracing Tkinter variables: trace_vdelete() with wrong mode no longer break tracing, trace_vinfo() now always returns a list of pairs of strings, tracing in the «u» mode now works.
- Fix a scoping issue in importlib.util.LazyLoader which triggered an UnboundLocalError when lazy-loading a module that was already put into sys.modules.
- bpo-27079 [https://bugs.python.org/issue?@action=redirect&bpo=27079]: Fixed curses.ascii functions isblank(), iscntrl() and ispunct().
- bpo-26754 [https://bugs.python.org/issue?@action=redirect&bpo=26754]: Some functions (compile() etc) accepted a filename argument encoded as an iterable of integers. Now only strings and byte-like objects are accepted.
- bpo-27048 [https://bugs.python.org/issue?@action=redirect&bpo=27048]: Prevents distutils failing on Windows when environment variables contain non-ASCII characters
- bpo-27330 [https://bugs.python.org/issue?@action=redirect&bpo=27330]: Fixed possible leaks in the ctypes module.
- bpo-27238 [https://bugs.python.org/issue?@action=redirect&bpo=27238]: Got rid of bare excepts in the turtle module. Original patch by Jelle Zijlstra.
- bpo-27122 [https://bugs.python.org/issue?@action=redirect&bpo=27122]: When an exception is raised within the context being managed by a contextlib.ExitStack() and one of the exit stack generators catches and raises it in a chain, do not re-raise the original exception when exiting, let the new chained one through. This avoids the **PEP 479** [https://peps.python.org/pep-0479/] bug described in issue25782.
- bpo-26386 [https://bugs.python.org/issue?@action=redirect&bpo=26386]: Fixed ttk.TreeView selection operations with item id’s containing spaces.
- bpo-16182 [https://bugs.python.org/issue?@action=redirect&bpo=16182]: Fix various functions in the «readline» module to use the locale encoding, and fix get_begidx() and get_endidx() to return code point indexes.
- bpo-27392 [https://bugs.python.org/issue?

@action=redirect&bpo=27392]: Add loop.connect_accepted_socket(). Patch by Jim Fulton.
- bpo-27930 [https://bugs.python.org/issue?@action=redirect&bpo=27930]: Improved behaviour of logging.handlers.QueueListener. Thanks to Paulo Andrade and Petr Viktorin for the analysis and patch.
- bpo-21201 [https://bugs.python.org/issue?@action=redirect&bpo=21201]: Improves readability of multiprocessing error message. Thanks to Wojciech Walczak for patch.
- bpo-27456 [https://bugs.python.org/issue?@action=redirect&bpo=27456]: asyncio: Set TCP_NODELAY by default.
- bpo-27906 [https://bugs.python.org/issue?@action=redirect&bpo=27906]: Fix socket accept exhaustion during high TCP traffic. Patch by Kevin Conway.
- bpo-28174 [https://bugs.python.org/issue?@action=redirect&bpo=28174]: Handle when SO_REUSEPORT isn’t properly supported. Patch by Seth Michael Larson.
- bpo-26654 [https://bugs.python.org/issue?@action=redirect&bpo=26654]: Inspect functools.partial in asyncio.Handle.__repr__. Patch by iceboy.
- bpo-26909 [https://bugs.python.org/issue?@action=redirect&bpo=26909]: Fix slow pipes IO in asyncio. Patch by INADA Naoki.
- bpo-28176 [https://bugs.python.org/issue?@action=redirect&bpo=28176]: Fix callbacks race in asyncio.SelectorLoop.sock_connect.
- bpo-27759 [https://bugs.python.org/issue?@action=redirect&bpo=27759]: Fix selectors incorrectly retain invalid file descriptors. Patch by Mark Williams.
- bpo-28368 [https://bugs.python.org/issue?@action=redirect&bpo=28368]: Refuse monitoring processes if the child watcher has no loop attached. Patch by Vincent Michel.
- bpo-28369 [https://bugs.python.org/issue?@action=redirect&bpo=28369]: Raise RuntimeError when transport’s FD is used with add_reader, add_writer, etc.
- bpo-28370 [https://bugs.python.org/issue?@action=redirect&bpo=28370]: Speedup asyncio.StreamReader.readexactly. Patch by Коренберг

Марк.

- bpo-28371 [https://bugs.python.org/issue?@action=redirect&bpo=28371]: Deprecate passing asyncio.Handles to run_in_executor.
- bpo-28372 [https://bugs.python.org/issue?@action=redirect&bpo=28372]: Fix asyncio to support formatting of non-python coroutines.
- bpo-28399 [https://bugs.python.org/issue?@action=redirect&bpo=28399]: Remove UNIX socket from FS before binding. Patch by Коренберг Марк.
- bpo-27972 [https://bugs.python.org/issue?@action=redirect&bpo=27972]: Prohibit Tasks to await on themselves.
- bpo-26923 [https://bugs.python.org/issue?@action=redirect&bpo=26923]: Fix asyncio.Gather to refuse being cancelled once all children are done. Patch by Johannes Ebke.
- bpo-26796 [https://bugs.python.org/issue?@action=redirect&bpo=26796]: Don't configure the number of workers for default threadpool executor. Initial patch by Hans Lawrenz.
- bpo-28600 [https://bugs.python.org/issue?@action=redirect&bpo=28600]: Optimize loop.call_soon().
- bpo-28613 [https://bugs.python.org/issue?@action=redirect&bpo=28613]: Fix get_event_loop() return the current loop if called from coroutines/callbacks.
- bpo-28639 [https://bugs.python.org/issue?@action=redirect&bpo=28639]: Fix inspect.isawaitable to always return bool Patch by Justin Mayfield.
- bpo-28652 [https://bugs.python.org/issue?@action=redirect&bpo=28652]: Make loop methods reject socket kinds they do not support.
- bpo-28653 [https://bugs.python.org/issue?@action=redirect&bpo=28653]: Fix a refleak in functools.lru_cache.
- bpo-28703 [https://bugs.python.org/issue?@action=redirect&bpo=28703]: Fix asyncio.iscoroutinefunction to handle Mock objects.
- bpo-24142 [https://bugs.python.org/issue?@action=redirect&bpo=24142]: Reading a corrupt config file left the parser in an invalid state. Original patch by Florian Höch.

- bpo-28990 [https://bugs.python.org/issue?@action=redirect&bpo=28990]: Fix SSL hanging if connection is closed before handshake completed. (Patch by HoHo-Ho)

IDLE

- bpo-15308 [https://bugs.python.org/issue?@action=redirect&bpo=15308]: Add “interrupt execution” (^C) to Shell menu. Patch by Roger Serwy, updated by Bayard Randel.
- bpo-27922 [https://bugs.python.org/issue?@action=redirect&bpo=27922]: Stop IDLE tests from “flashing” gui widgets on the screen.
- Add version to title of IDLE help window.
- bpo-25564 [https://bugs.python.org/issue?@action=redirect&bpo=25564]: In section on IDLE – console differences, mention that using exec means that _builtins_ is defined for each statement.
- bpo-27714 [https://bugs.python.org/issue?@action=redirect&bpo=27714]: text_textview and test_autocomplete now pass when re-run in the same process. This occurs when test_idle fails when run with the -w option but without -jn. Fix warning from test_config.
- bpo-25507 [https://bugs.python.org/issue?@action=redirect&bpo=25507]: IDLE no longer runs buggy code because of its tkinter imports. Users must include the same imports required to run directly in Python.
- bpo-27452 [https://bugs.python.org/issue?@action=redirect&bpo=27452]: add line counter and crc to IDLE configHandler test dump.
- bpo-27365 [https://bugs.python.org/issue?@action=redirect&bpo=27365]: Allow non-ascii chars in IDLE NEWS.txt, for contributor names.
- bpo-27245 [https://bugs.python.org/issue?@action=redirect&bpo=27245]: IDLE: Cleanly delete custom themes and key bindings. Previously, when IDLE was started from a console or by import, a cascade of warnings was emitted. Patch by Serhiy Storchaka.

C API

- bpo-28808 [https://bugs.python.org/issue?@action=redirect&bpo=28808]: PyUnicode_CompareWithASCIIString() now never raises exceptions.
- bpo-26754 [https://bugs.python.org/issue?@action=redirect&bpo=26754]: PyUnicode_FSDecoder() accepted a filename argument encoded as an iterable of integers. Now only strings and bytes-like objects are accepted.

Documentation

- bpo-28513 [https://bugs.python.org/issue?@action=redirect&bpo=28513]: Documented command-line interface of zipfile.

Tests

- bpo-28950 [https://bugs.python.org/issue?@action=redirect&bpo=28950]: Disallow -j0 to be combined with -T/-l/-M in regrtest command line arguments.
- bpo-28666 [https://bugs.python.org/issue?@action=redirect&bpo=28666]: Now test.support.rmtree is able to remove unwritable or unreadable directories.
- bpo-23839 [https://bugs.python.org/issue?@action=redirect&bpo=23839]: Various caches now are cleared before running every test file.
- bpo-28409 [https://bugs.python.org/issue?@action=redirect&bpo=28409]: regrtest: fix the parser of command line arguments.
- bpo-27787 [https://bugs.python.org/issue?@action=redirect&bpo=27787]: Call gc.collect() before checking each test for «dangling threads», since the dangling threads are weak references.
- bpo-27369 [https://bugs.python.org/issue?@action=redirect&bpo=27369]: In test_pyexpat, avoid testing an error message detail that changed in Expat 2.2.0.

Tools/Demos

- bpo-27952 [https://bugs.python.org/issue?

@action=redirect&bpo=27952]: Get Tools/scripts/fixcid.py working with Python 3 and the current «re» module, avoid invalid Python backslash escapes, and fix a bug parsing escaped C quote signs.
- bpo-27332 [https://bugs.python.org/issue?@action=redirect&bpo=27332]: Fixed the type of the first argument of module-level functions generated by Argument Clinic. Patch by Petr Viktorin.
- bpo-27418 [https://bugs.python.org/issue?@action=redirect&bpo=27418]: Fixed Tools/importbench/importbench.py.

Windows

- bpo-28251 [https://bugs.python.org/issue?@action=redirect&bpo=28251]: Improvements to help manuals on Windows.
- bpo-28110 [https://bugs.python.org/issue?@action=redirect&bpo=28110]: launcher.msi has different product codes between 32-bit and 64-bit
- bpo-25144 [https://bugs.python.org/issue?@action=redirect&bpo=25144]: Ensures TargetDir is set before continuing with custom install.
- bpo-27469 [https://bugs.python.org/issue?@action=redirect&bpo=27469]: Adds a shell extension to the launcher so that drag and drop works correctly.
- bpo-27309 [https://bugs.python.org/issue?@action=redirect&bpo=27309]: Enabled proper Windows styles in python[w].exe manifest.

Build

- bpo-29080 [https://bugs.python.org/issue?@action=redirect&bpo=29080]: Removes hard dependency on hg.exe from PCBuild/build.bat
- bpo-23903 [https://bugs.python.org/issue?@action=redirect&bpo=23903]: Added missed names to PC/python3.def.
- bpo-10656 [https://bugs.python.org/issue?@action=redirect&bpo=10656]: Fix out-of-tree building on AIX.

Patch by Tristan Carel and Michael Haubenwallner.

- bpo-26359 [https://bugs.python.org/issue?@action=redirect&bpo=26359]: Rename –with-optimiations to –enable-optimizations.
- bpo-28444 [https://bugs.python.org/issue?@action=redirect&bpo=28444]: Fix missing extensions modules when cross compiling.
- bpo-28248 [https://bugs.python.org/issue?@action=redirect&bpo=28248]: Update Windows build and OS X installers to use OpenSSL 1.0.2j.
- bpo-28258 [https://bugs.python.org/issue?@action=redirect&bpo=28258]: Fixed build with Estonian locale (python-config and distclean targets in Makefile). Patch by Arfrever Frehtes Taifersar Arahesis.
- bpo-26661 [https://bugs.python.org/issue?@action=redirect&bpo=26661]: setup.py now detects system libffi with multiarch wrapper.
- bpo-28066 [https://bugs.python.org/issue?@action=redirect&bpo=28066]: Fix the logic that searches build directories for generated include files when building outside the source tree.
- bpo-15819 [https://bugs.python.org/issue?@action=redirect&bpo=15819]: Remove redundant include search directory option for building outside the source tree.
- bpo-27566 [https://bugs.python.org/issue?@action=redirect&bpo=27566]: Fix clean target in freeze makefile (patch by Lisa Roach)
- bpo-27705 [https://bugs.python.org/issue?@action=redirect&bpo=27705]: Update message in validate_ucrtbase.py
- bpo-27983 [https://bugs.python.org/issue?@action=redirect&bpo=27983]: Cause lack of llvm-profdata tool when using clang as required for PGO linking to be a configure time error rather than make time when –with-optimizations is enabled. Also improve our ability to find the llvm-profdata tool on MacOS and some Linuxes.
- bpo-26307 [https://bugs.python.org/issue?@action=redirect&bpo=26307]: The profile-opt build now applies PGO to the built-in modules.
- bpo-26359 [https://bugs.python.org/issue?

@action=redirect&bpo=26359]: Add the –with-optimizations configure flag.

- bpo-27713 [https://bugs.python.org/issue?@action=redirect&bpo=27713]: Suppress spurious build warnings when updating importlib's bootstrap files. Patch by Xiang Zhang
- bpo-25825 [https://bugs.python.org/issue?@action=redirect&bpo=25825]: Correct the references to Modules/python.exp and ld_so_aix, which are required on AIX. This updates references to an installation path that was changed in 3.2a4, and undoes changed references to the build tree that were made in 3.5.0a1.
- bpo-27453 [https://bugs.python.org/issue?@action=redirect&bpo=27453]: CPP invocation in configure must use CPPFLAGS. Patch by Chi Hsuan Yen.
- bpo-27641 [https://bugs.python.org/issue?@action=redirect&bpo=27641]: The configure script now inserts comments into the makefile to prevent the pgen and _freeze_importlib executables from being cross-compiled.
- bpo-26662 [https://bugs.python.org/issue?@action=redirect&bpo=26662]: Set PYTHON_FOR_GEN in configure as the Python program to be used for file generation during the build.
- bpo-10910 [https://bugs.python.org/issue?@action=redirect&bpo=10910]: Avoid C++ compilation errors on FreeBSD and OS X. Also update FreedBSD version checks for the original ctype UTF-8 workaround.
- bpo-28676 [https://bugs.python.org/issue?@action=redirect&bpo=28676]: Prevent missing "getentropy" declaration warning on macOS. Patch by Gareth Rees.

Python 3.5.2 final

Release date: 2016-06-26

Core and Builtins

- bpo-26930 [https://bugs.python.org/issue?@action=redirect&bpo=26930]: Update Windows builds to use

OpenSSL 1.0.2h.

Tests

- bpo-26867 [https://bugs.python.org/issue?@action=redirect&bpo=26867]: Ubuntu's openssl OP_NO_SSLv3 is forced on by default; fix test.

IDLE

- bpo-27365 [https://bugs.python.org/issue?@action=redirect&bpo=27365]: Allow non-ascii in idlelib/NEWS.txt - minimal part for 3.5.2.

Python 3.5.2 release candidate 1

Release date: 2016-06-12

Security

- bpo-26556 [https://bugs.python.org/issue?@action=redirect&bpo=26556]: Update expat to 2.1.1, fixes **CVE 2015-1283** [https://www.cve.org/CVERecord?id=CVE-2015-1283].
- Fix TLS stripping vulnerability in smtplib, **CVE 2016-0772** [https://www.cve.org/CVERecord?id=CVE-2016-0772]. Reported by Team Oststrom.
- bpo-26839 [https://bugs.python.org/issue?@action=redirect&bpo=26839]: On Linux, **`os.urandom()`** now calls `getrandom()` with `GRND_NONBLOCK` to fall back on reading `/dev/urandom` if the urandom entropy pool is not initialized yet. Patch written by Colm Buckley.
- bpo-26657 [https://bugs.python.org/issue?@action=redirect&bpo=26657]: Fix directory traversal vulnerability with http.server on Windows. This fixes a regression that was introduced in 3.3.4rc1 and 3.4.0rc1. Based on patch by Philipp Hagemeister.
- bpo-26313 [https://bugs.python.org/issue?@action=redirect&bpo=26313]: ssl.py _load_windows_store_certs fails if windows cert store is empty. Patch by Baji.

- bpo-25939 [https://bugs.python.org/issue?@action=redirect&bpo=25939]: On Windows open the cert store readonly in ssl.enum_certificates.

Core and Builtins

- bpo-27066 [https://bugs.python.org/issue?@action=redirect&bpo=27066]: Fixed SystemError if a custom opener (for open()) returns a negative number without setting an exception.
- bpo-20041 [https://bugs.python.org/issue?@action=redirect&bpo=20041]: Fixed TypeError when frame.f_trace is set to None. Patch by Xavier de Gaye.
- bpo-26168 [https://bugs.python.org/issue?@action=redirect&bpo=26168]: Fixed possible refleaks in failing Py_BuildValue() with the «N» format unit.
- bpo-26991 [https://bugs.python.org/issue?@action=redirect&bpo=26991]: Fix possible refleak when creating a function with annotations.
- bpo-27039 [https://bugs.python.org/issue?@action=redirect&bpo=27039]: Fixed bytearray.remove() for values greater than 127. Patch by Joe Jevnik.
- bpo-23640 [https://bugs.python.org/issue?@action=redirect&bpo=23640]: int.from_bytes() no longer bypasses constructors for subclasses.
- bpo-26811 [https://bugs.python.org/issue?@action=redirect&bpo=26811]: gc.get_objects() no longer contains a broken tuple with NULL pointer.
- bpo-20120 [https://bugs.python.org/issue?@action=redirect&bpo=20120]: Use RawConfigParser for .pypirc parsing, removing support for interpolation unintentionally added with move to Python 3. Behavior no longer does any interpolation in .pypirc files, matching behavior in Python 2.7 and Setuptools 19.0.
- bpo-26659 [https://bugs.python.org/issue?@action=redirect&bpo=26659]: Make the builtin slice type support cycle collection.
- bpo-26718 [https://bugs.python.org/issue?@action=redirect&bpo=26718]: super.__init__ no longer leaks memory if called multiple times. NOTE: A direct call of

super.__init__ is not endorsed!

- bpo-25339 [https://bugs.python.org/issue?@action=redirect&bpo=25339]: PYTHONIOENCODING now has priority over locale in setting the error handler for stdin and stdout.
- bpo-26494 [https://bugs.python.org/issue?@action=redirect&bpo=26494]: Fixed crash on iterating exhausting iterators. Affected classes are generic sequence iterators, iterators of str, bytes, bytearray, list, tuple, set, frozenset, dict, OrderedDict, corresponding views and os.scandir() iterator.
- bpo-26581 [https://bugs.python.org/issue?@action=redirect&bpo=26581]: If coding cookie is specified multiple times on a line in Python source code file, only the first one is taken to account.
- bpo-26464 [https://bugs.python.org/issue?@action=redirect&bpo=26464]: Fix str.translate() when string is ASCII and first replacements removes character, but next replacement uses a non-ASCII character or a string longer than 1 character. Regression introduced in Python 3.5.0.
- bpo-22836 [https://bugs.python.org/issue?@action=redirect&bpo=22836]: Ensure exception reports from PyErr_Display() and PyErr_WriteUnraisable() are sensible even when formatting them produces secondary errors. This affects the reports produced by sys.__excepthook__() and when __del__() raises an exception.
- bpo-26302 [https://bugs.python.org/issue?@action=redirect&bpo=26302]: Correct behavior to reject comma as a legal character for cookie names.
- bpo-4806 [https://bugs.python.org/issue?@action=redirect&bpo=4806]: Avoid masking the original TypeError exception when using star (*) unpacking in function calls. Based on patch by Hagen Fürstenau and Daniel Urban.
- bpo-27138 [https://bugs.python.org/issue?@action=redirect&bpo=27138]: Fix the doc comment for FileFinder.find_spec().
- bpo-26154 [https://bugs.python.org/issue?@action=redirect&bpo=26154]: Add a new private _PyThreadState_UncheckedGet() function to get the current Python thread state, but don't issue a fatal error if it is NULL.

This new function must be used instead of accessing directly the _PyThreadState_Current variable. The variable is no more exposed since Python 3.5.1 to hide the exact implementation of atomic C types, to avoid compiler issues.

- bpo-26194 [https://bugs.python.org/issue?@action=redirect&bpo=26194]: Deque.insert() gave odd results for bounded deques that had reached their maximum size. Now an IndexError will be raised when attempting to insert into a full deque.
- bpo-25843 [https://bugs.python.org/issue?@action=redirect&bpo=25843]: When compiling code, don’t merge constants if they are equal but have a different types. For example, `f1, f2 = lambda: 1, lambda: 1.0` is now correctly compiled to two different functions: `f1()` returns `1` (`int`) and `f2()` returns `1.0` (`int`), even if `1` and `1.0` are equal.
- bpo-22995 [https://bugs.python.org/issue?@action=redirect&bpo=22995]: [UPDATE] Comment out the one of the pickleability tests in _PyObject_GetState() due to regressions observed in Cython-based projects.
- bpo-25961 [https://bugs.python.org/issue?@action=redirect&bpo=25961]: Disallowed null characters in the type name.
- bpo-25973 [https://bugs.python.org/issue?@action=redirect&bpo=25973]: Fix segfault when an invalid nonlocal statement binds a name starting with two underscores.
- bpo-22995 [https://bugs.python.org/issue?@action=redirect&bpo=22995]: Instances of extension types with a state that aren’t subclasses of list or dict and haven’t implemented any pickle-related methods (_reduce_, _reduce_ex_, _getnewargs_, _getnewargs_ex_, or _getstate_), can no longer be pickled. Including memoryview.
- bpo-20440 [https://bugs.python.org/issue?@action=redirect&bpo=20440]: Massive replacing unsafe attribute setting code with special macro Py_SETREF.
- bpo-25766 [https://bugs.python.org/issue?@action=redirect&bpo=25766]: Special method _bytes_() now works in str subclasses.

- bpo-25421 [https://bugs.python.org/issue?@action=redirect&bpo=25421]: _sizeof_ methods of builtin types now use dynamic basic size. This allows sys.getsize() to work correctly with their subclasses with _slots_ defined.
- bpo-25709 [https://bugs.python.org/issue?@action=redirect&bpo=25709]: Fixed problem with in-place string concatenation and utf-8 cache.
- bpo-27147 [https://bugs.python.org/issue?@action=redirect&bpo=27147]: Mention **PEP 420** [https://peps.python.org/pep-0420/] in the importlib docs.
- bpo-24097 [https://bugs.python.org/issue?@action=redirect&bpo=24097]: Fixed crash in object._reduce_() if slot name is freed inside _getattr_.
- bpo-24731 [https://bugs.python.org/issue?@action=redirect&bpo=24731]: Fixed crash on converting objects with special methods _bytes_, _trunc_, and _float_ returning instances of subclasses of bytes, int, and float to subclasses of bytes, int, and float correspondingly.
- bpo-26478 [https://bugs.python.org/issue?@action=redirect&bpo=26478]: Fix semantic bugs when using binary operators with dictionary views and tuples.
- bpo-26171 [https://bugs.python.org/issue?@action=redirect&bpo=26171]: Fix possible integer overflow and heap corruption in zipimporter.get_data().
- bpo-25660 [https://bugs.python.org/issue?@action=redirect&bpo=25660]: Fix TAB key behaviour in REPL with readline.
- bpo-25887 [https://bugs.python.org/issue?@action=redirect&bpo=25887]: Raise a RuntimeError when a coroutine object is awaited more than once.
- bpo-27243 [https://bugs.python.org/issue?@action=redirect&bpo=27243]: Update the _aiter_ protocol: instead of returning an awaitable that resolves to an asynchronous iterator, the asynchronous iterator should be returned directly. Doing the former will trigger a PendingDeprecationWarning.

Library

- bpo-21386 [https://bugs.python.org/issue?

@action=redirect&bpo=21386]: Implement missing IPv4Address.is_global property. It was documented since 07a5610bae9d. Initial patch by Roger Luethi.

- bpo-20900 [https://bugs.python.org/issue?@action=redirect&bpo=20900]: distutils register command now decodes HTTP responses correctly. Initial patch by ingrid.
- A new version of typing.py provides several new classes and features: @overload outside stubs, Reversible, DefaultDict, Text, ContextManager, Type[], NewType(), TYPE_CHECKING, and numerous bug fixes (note that some of the new features are not yet implemented in mypy or other static analyzers). Also classes for **PEP 492** [https://peps.python.org/pep-0492/] (Awaitable, AsyncIterable, AsyncIterator) have been added (in fact they made it into 3.5.1 but were never mentioned).
- bpo-25738 [https://bugs.python.org/issue?@action=redirect&bpo=25738]: Stop http.server.BaseHTTPRequestHandler.send_error() from sending a message body for 205 Reset Content. Also, don’t send Content header fields in responses that don’t have a body. Patch by Susumu Koshiba.
- bpo-21313 [https://bugs.python.org/issue?@action=redirect&bpo=21313]: Fix the «platform» module to tolerate when sys.version contains truncated build information.
- bpo-27164 [https://bugs.python.org/issue?@action=redirect&bpo=27164]: In the zlib module, allow decompressing raw Deflate streams with a predefined zdict. Based on patch by Xiang Zhang.
- bpo-24291 [https://bugs.python.org/issue?@action=redirect&bpo=24291]: Fix wsgiref.simple_server.WSGIRequestHandler to completely write data to the client. Previously it could do partial writes and truncate data. Also, wsgiref.handler.ServerHandler can now handle stdout doing partial writes, but this is deprecated.
- bpo-26809 [https://bugs.python.org/issue?@action=redirect&bpo=26809]: Add `__all__` to **`string`**. Patch by Emanuel Barry.
- bpo-26373 [https://bugs.python.org/issue?@action=redirect&bpo=26373]: subprocess.Popen.communicate now correctly ignores BrokenPipeError when the child

process dies before .communicate() is called in more/all circumstances.

- bpo-21776 [https://bugs.python.org/issue?@action=redirect&bpo=21776]: distutils.upload now correctly handles HTTPError. Initial patch by Claudiu Popa.
- bpo-27114 [https://bugs.python.org/issue?@action=redirect&bpo=27114]: Fix SSLContext._load_windows_store_certs fails with PermissionError
- bpo-18383 [https://bugs.python.org/issue?@action=redirect&bpo=18383]: Avoid creating duplicate filters when using filterwarnings and simplefilter. Based on patch by Alex Shkop.
- bpo-27057 [https://bugs.python.org/issue?@action=redirect&bpo=27057]: Fix os.set_inheritable() on Android, ioctl() is blocked by SELinux and fails with EACCESS. The function now falls back to fcntl(). Patch written by Michał Bednarski.
- bpo-27014 [https://bugs.python.org/issue?@action=redirect&bpo=27014]: Fix infinite recursion using typing.py. Thanks to Kalle Tuure!
- bpo-14132 [https://bugs.python.org/issue?@action=redirect&bpo=14132]: Fix urllib.request redirect handling when the target only has a query string. Original fix by Ján Janech.
- bpo-17214 [https://bugs.python.org/issue?@action=redirect&bpo=17214]: The «urllib.request» module now percent-encodes non-ASCII bytes found in redirect target URLs. Some servers send Location header fields with non-ASCII bytes, but «http.client» requires the request target to be ASCII-encodable, otherwise a UnicodeEncodeError is raised. Based on patch by Christian Heimes.
- bpo-26892 [https://bugs.python.org/issue?@action=redirect&bpo=26892]: Honor debuglevel flag in urllib.request.HTTPHandler. Patch contributed by Chi Hsuan Yen.
- bpo-22274 [https://bugs.python.org/issue?@action=redirect&bpo=22274]: In the subprocess module, allow stderr to be redirected to stdout even when stdout is not redirected. Patch by Akira Li.

- bpo-26807 [https://bugs.python.org/issue?@action=redirect&bpo=26807]: mock_open “files” no longer error on readline at end of file. Patch from Yolanda Robla.
- bpo-25745 [https://bugs.python.org/issue?@action=redirect&bpo=25745]: Fixed leaking a userptr in curses panel destructor.
- bpo-26977 [https://bugs.python.org/issue?@action=redirect&bpo=26977]: Removed unnecessary, and ignored, call to sum of squares helper in statistics.pvariance.
- bpo-26881 [https://bugs.python.org/issue?@action=redirect&bpo=26881]: The modulefinder module now supports extended opcode arguments.
- bpo-23815 [https://bugs.python.org/issue?@action=redirect&bpo=23815]: Fixed crashes related to directly created instances of types in _tkinter and curses.panel modules.
- bpo-17765 [https://bugs.python.org/issue?@action=redirect&bpo=17765]: weakref.ref() no longer silently ignores keyword arguments. Patch by Georg Brandl.
- bpo-26873 [https://bugs.python.org/issue?@action=redirect&bpo=26873]: xmlrpc now raises ResponseError on unsupported type tags instead of silently return incorrect result.
- bpo-26711 [https://bugs.python.org/issue?@action=redirect&bpo=26711]: Fixed the comparison of plistlib.Data with other types.
- bpo-24114 [https://bugs.python.org/issue?@action=redirect&bpo=24114]: Fix an uninitialized variable in `ctypes.util`. The bug only occurs on SunOS when the ctypes implementation searches for the `crle` program. Patch by Xiang Zhang. Tested on SunOS by Kees Bos.
- bpo-26864 [https://bugs.python.org/issue?@action=redirect&bpo=26864]: In urllib.request, change the proxy bypass host checking against no_proxy to be case-insensitive, and to not match unrelated host names that happen to have a bypassed hostname as a suffix. Patch by Xiang Zhang.
- bpo-26634 [https://bugs.python.org/issue?@action=redirect&bpo=26634]: recursive_repr() now sets _qualname_ of wrapper. Patch by Xiang Zhang.

- bpo-26804 [https://bugs.python.org/issue?@action=redirect&bpo=26804]: urllib.request will prefer lower_case proxy environment variables over UPPER_CASE or Mixed_Case ones. Patch contributed by Hans-Peter Jansen.
- bpo-26837 [https://bugs.python.org/issue?@action=redirect&bpo=26837]: assertSequenceEqual() now correctly outputs non-stringified differing items (like bytes in the -b mode). This affects assertListEqual() and assertTupleEqual().
- bpo-26041 [https://bugs.python.org/issue?@action=redirect&bpo=26041]: Remove «will be removed in Python 3.7» from deprecation messages of platform.dist() and platform.linux_distribution(). Patch by Kumaripaba Miyurusara Athukorala.
- bpo-26822 [https://bugs.python.org/issue?@action=redirect&bpo=26822]: itemgetter, attrgetter and methodcaller objects no longer silently ignore keyword arguments.
- bpo-26733 [https://bugs.python.org/issue?@action=redirect&bpo=26733]: Disassembling a class now disassembles class and static methods. Patch by Xiang Zhang.
- bpo-26801 [https://bugs.python.org/issue?@action=redirect&bpo=26801]: Fix error handling in **`shutil.get_terminal_size()`**, catch **`AttributeError`** instead of **`NameError`**. Patch written by Emanuel Barry.
- bpo-24838 [https://bugs.python.org/issue?@action=redirect&bpo=24838]: tarfile's ustar and gnu formats now correctly calculate name and link field limits for multibyte character encodings like utf-8.
- bpo-26717 [https://bugs.python.org/issue?@action=redirect&bpo=26717]: Stop encoding Latin-1-ized WSGI paths with UTF-8. Patch by Anthony Sottile.
- bpo-26735 [https://bugs.python.org/issue?@action=redirect&bpo=26735]: Fix **`os.urandom()`** on Solaris 11.3 and newer when reading more than 1,024 bytes: call `getrandom()` multiple times with a limit of 1024 bytes per call.
- bpo-16329 [https://bugs.python.org/issue?@action=redirect&bpo=16329]: Add .webm to mimetypes.types_map. Patch by Giampaolo Rodola'.

- bpo-13952 [https://bugs.python.org/issue?@action=redirect&bpo=13952]: Add .csv to mimetypes.types_map. Patch by Geoff Wilson.
- bpo-26709 [https://bugs.python.org/issue?@action=redirect&bpo=26709]: Fixed Y2038 problem in loading binary PLists.
- bpo-23735 [https://bugs.python.org/issue?@action=redirect&bpo=23735]: Handle terminal resizing with Readline 6.3+ by installing our own SIGWINCH handler. Patch by Eric Price.
- bpo-26586 [https://bugs.python.org/issue?@action=redirect&bpo=26586]: In http.server, respond with «413 Request header fields too large» if there are too many header fields to parse, rather than killing the connection and raising an unhandled exception. Patch by Xiang Zhang.
- bpo-22854 [https://bugs.python.org/issue?@action=redirect&bpo=22854]: Change BufferedReader.writable() and BufferedWriter.readable() to always return False.
- bpo-25195 [https://bugs.python.org/issue?@action=redirect&bpo=25195]: Fix a regression in mock.MagicMock. _Call is a subclass of tuple (changeset 3603bae63c13 only works for classes) so we need to implement _ne_ ourselves. Patch by Andrew Plummer.
- bpo-26644 [https://bugs.python.org/issue?@action=redirect&bpo=26644]: Raise ValueError rather than SystemError when a negative length is passed to SSLSocket.recv() or read().
- bpo-23804 [https://bugs.python.org/issue?@action=redirect&bpo=23804]: Fix SSL recv(0) and read(0) methods to return zero bytes instead of up to 1024.
- bpo-26616 [https://bugs.python.org/issue?@action=redirect&bpo=26616]: Fixed a bug in datetime.astimezone() method.
- bpo-21925 [https://bugs.python.org/issue?@action=redirect&bpo=21925]: **`warnings.formatwarning()`** now catches exceptions on `linecache.getline(...)` to be able to log **`ResourceWarning`** emitted late during the Python shutdown process.
- bpo-24266 [https://bugs.python.org/issue?@action=redirect&bpo=24266]: Ctrl+C during Readline history

search now cancels the search mode when compiled with Readline 7.

- bpo-26560 [https://bugs.python.org/issue?@action=redirect&bpo=26560]: Avoid potential ValueError in BaseHandler.start_response. Initial patch by Peter Inglesby.
- bpo-26569 [https://bugs.python.org/issue?@action=redirect&bpo=26569]: Fix `pyclbr.readmodule()` and `pyclbr.readmodule_ex()` to support importing packages.
- bpo-26499 [https://bugs.python.org/issue?@action=redirect&bpo=26499]: Account for remaining Content-Length in HTTPResponse.readline() and read1(). Based on patch by Silent Ghost. Also document that HTTPResponse now supports these methods.
- bpo-25320 [https://bugs.python.org/issue?@action=redirect&bpo=25320]: Handle sockets in directories unittest discovery is scanning. Patch from Victor van den Elzen.
- bpo-16181 [https://bugs.python.org/issue?@action=redirect&bpo=16181]: cookiejar.http2time() now returns None if year is higher than datetime.MAXYEAR.
- bpo-26513 [https://bugs.python.org/issue?@action=redirect&bpo=26513]: Fixes platform module detection of Windows Server
- bpo-23718 [https://bugs.python.org/issue?@action=redirect&bpo=23718]: Fixed parsing time in week 0 before Jan 1. Original patch by Tamás Bence Gedai.
- bpo-20589 [https://bugs.python.org/issue?@action=redirect&bpo=20589]: Invoking Path.owner() and Path.group() on Windows now raise NotImplementedError instead of ImportError.
- bpo-26177 [https://bugs.python.org/issue?@action=redirect&bpo=26177]: Fixed the keys() method for Canvas and Scrollbar widgets.
- bpo-15068 [https://bugs.python.org/issue?@action=redirect&bpo=15068]: Got rid of excessive buffering in the fileinput module. The bufsize parameter is no longer used.
- bpo-2202 [https://bugs.python.org/issue?@action=redirect&bpo=2202]: Fix UnboundLocalError in AbstractDigestAuthHandler.get_algorithm_impls. Initial patch by Mathieu Dupuy.

- bpo-25718 [https://bugs.python.org/issue?@action=redirect&bpo=25718]: Fixed pickling and copying the accumulate() iterator with total is None.
- bpo-26475 [https://bugs.python.org/issue?@action=redirect&bpo=26475]: Fixed debugging output for regular expressions with the (?x) flag.
- bpo-26457 [https://bugs.python.org/issue?@action=redirect&bpo=26457]: Fixed the subnets() methods in IP network classes for the case when resulting prefix length is equal to maximal prefix length. Based on patch by Xiang Zhang.
- bpo-26385 [https://bugs.python.org/issue?@action=redirect&bpo=26385]: Remove the file if the internal open() call in NamedTemporaryFile() fails. Patch by Silent Ghost.
- bpo-26402 [https://bugs.python.org/issue?@action=redirect&bpo=26402]: Fix XML-RPC client to retry when the server shuts down a persistent connection. This was a regression related to the new http.client.RemoteDisconnected exception in 3.5.0a4.
- bpo-25913 [https://bugs.python.org/issue?@action=redirect&bpo=25913]: Leading `<~` is optional now in base64.a85decode() with adobe=True. Patch by Swati Jaiswal.
- bpo-26186 [https://bugs.python.org/issue?@action=redirect&bpo=26186]: Remove an invalid type check in importlib.util.LazyLoader.
- bpo-26367 [https://bugs.python.org/issue?@action=redirect&bpo=26367]: importlib.__import__() raises SystemError like builtins.__import__() when `level` is specified but without an accompanying package specified.
- bpo-26309 [https://bugs.python.org/issue?@action=redirect&bpo=26309]: In the «socketserver» module, shut down the request (closing the connected socket) when verify_request() returns false. Patch by Aviv Palivoda.
- bpo-25995 [https://bugs.python.org/issue?@action=redirect&bpo=25995]: os.walk() no longer uses FDs proportional to the tree depth.
- bpo-26117 [https://bugs.python.org/issue?@action=redirect&bpo=26117]: The os.scandir() iterator now

closes file descriptor not only when the iteration is finished, but when it was failed with error.

- bpo-25911 [https://bugs.python.org/issue?@action=redirect&bpo=25911]: Restored support of bytes paths in os.walk() on Windows.
- bpo-26045 [https://bugs.python.org/issue?@action=redirect&bpo=26045]: Add UTF-8 suggestion to error message when posting a non-Latin-1 string with http.client.
- bpo-12923 [https://bugs.python.org/issue?@action=redirect&bpo=12923]: Reset FancyURLopener's redirect counter even if there is an exception. Based on patches by Brian Brazil and Daniel Rocco.
- bpo-25945 [https://bugs.python.org/issue?@action=redirect&bpo=25945]: Fixed a crash when unpickle the functools.partial object with wrong state. Fixed a leak in failed functools.partial constructor. «args» and «keywords» attributes of functools.partial have now always types tuple and dict correspondingly.
- bpo-26202 [https://bugs.python.org/issue?@action=redirect&bpo=26202]: copy.deepcopy() now correctly copies range() objects with non-atomic attributes.
- bpo-23076 [https://bugs.python.org/issue?@action=redirect&bpo=23076]: Path.glob() now raises a ValueError if it's called with an invalid pattern. Patch by Thomas Nyberg.
- bpo-19883 [https://bugs.python.org/issue?@action=redirect&bpo=19883]: Fixed possible integer overflows in zipimport.
- bpo-26227 [https://bugs.python.org/issue?@action=redirect&bpo=26227]: On Windows, getnameinfo(), gethostbyaddr() and gethostbyname_ex() functions of the socket module now decode the hostname from the ANSI code page rather than UTF-8.
- bpo-26147 [https://bugs.python.org/issue?@action=redirect&bpo=26147]: xmlrpc now works with strings not encodable with used non-UTF-8 encoding.
- bpo-25935 [https://bugs.python.org/issue?@action=redirect&bpo=25935]: Garbage collector now breaks reference loops with OrderedDict.
- bpo-16620 [https://bugs.python.org/issue?

@action=redirect&bpo=16620]: Fixed AttributeError in msilib.Directory.glob().

- bpo-26013 [https://bugs.python.org/issue?@action=redirect&bpo=26013]: Added compatibility with broken protocol 2 pickles created in old Python 3 versions (3.4.3 and lower).
- bpo-25850 [https://bugs.python.org/issue?@action=redirect&bpo=25850]: Use cross-compilation by default for 64-bit Windows.
- bpo-17633 [https://bugs.python.org/issue?@action=redirect&bpo=17633]: Improve zipimport's support for namespace packages.
- bpo-24705 [https://bugs.python.org/issue?@action=redirect&bpo=24705]: Fix sysconfig._parse_makefile not expanding ${} vars appearing before $() vars.
- bpo-22138 [https://bugs.python.org/issue?@action=redirect&bpo=22138]: Fix mock.patch behavior when patching descriptors. Restore original values after patching. Patch contributed by Sean McCully.
- bpo-25672 [https://bugs.python.org/issue?@action=redirect&bpo=25672]: In the ssl module, enable the SSL_MODE_RELEASE_BUFFERS mode option if it is safe to do so.
- bpo-26012 [https://bugs.python.org/issue?@action=redirect&bpo=26012]: Don't traverse into symlinks for ** pattern in pathlib.Path.[r]glob().
- bpo-24120 [https://bugs.python.org/issue?@action=redirect&bpo=24120]: Ignore PermissionError when traversing a tree with pathlib.Path.[r]glob(). Patch by Ulrich Petri.
- bpo-25447 [https://bugs.python.org/issue?@action=redirect&bpo=25447]: fileinput now uses sys.stdin as-is if it does not have a buffer attribute (restores backward compatibility).
- bpo-25447 [https://bugs.python.org/issue?@action=redirect&bpo=25447]: Copying the lru_cache() wrapper object now always works, independently from the type of the wrapped object (by returning the original object unchanged).
- bpo-24103 [https://bugs.python.org/issue?@action=redirect&bpo=24103]: Fixed possible use after free in

ElementTree.XMLPullParser.

- bpo-25860 [https://bugs.python.org/issue?@action=redirect&bpo=25860]: os.fwalk() no longer skips remaining directories when error occurs. Original patch by Samson Lee.
- bpo-25914 [https://bugs.python.org/issue?@action=redirect&bpo=25914]: Fixed and simplified OrderedDict.__sizeof__.
- bpo-25902 [https://bugs.python.org/issue?@action=redirect&bpo=25902]: Fixed various refcount issues in ElementTree iteration.
- bpo-25717 [https://bugs.python.org/issue?@action=redirect&bpo=25717]: Restore the previous behaviour of tolerating most fstat() errors when opening files. This was a regression in 3.5a1, and stopped anonymous temporary files from working in special cases.
- bpo-24903 [https://bugs.python.org/issue?@action=redirect&bpo=24903]: Fix regression in number of arguments compileall accepts when "-d" is specified. The check on the number of arguments has been dropped completely as it never worked correctly anyway.
- bpo-25764 [https://bugs.python.org/issue?@action=redirect&bpo=25764]: In the subprocess module, preserve any exception caused by fork() failure when preexec_fn is used.
- bpo-6478 [https://bugs.python.org/issue?@action=redirect&bpo=6478]: _strptime's regexp cache now is reset after changing timezone with time.tzset().
- bpo-14285 [https://bugs.python.org/issue?@action=redirect&bpo=14285]: When executing a package with the «python -m package» option, and package initialization fails, a proper traceback is now reported. The «runpy» module now lets exceptions from package initialization pass back to the caller, rather than raising ImportError.
- bpo-19771 [https://bugs.python.org/issue?@action=redirect&bpo=19771]: Also in runpy and the «-m» option, omit the irrelevant message «… is a package and cannot be directly executed» if the package could not even be initialized (e.g. due to a bad `*.pyc` file).
- bpo-25177 [https://bugs.python.org/issue?

@action=redirect&bpo=25177]: Fixed problem with the mean of very small and very large numbers. As a side effect, statistics.mean and statistics.variance should be significantly faster.
- bpo-25718 [https://bugs.python.org/issue?@action=redirect&bpo=25718]: Fixed copying object with state with boolean value is false.
- bpo-10131 [https://bugs.python.org/issue?@action=redirect&bpo=10131]: Fixed deep copying of minidom documents. Based on patch by Marian Ganisin.
- bpo-25725 [https://bugs.python.org/issue?@action=redirect&bpo=25725]: Fixed a reference leak in pickle.loads() when unpickling invalid data including tuple instructions.
- bpo-25663 [https://bugs.python.org/issue?@action=redirect&bpo=25663]: In the Readline completer, avoid listing duplicate global names, and search the global namespace before searching builtins.
- bpo-25688 [https://bugs.python.org/issue?@action=redirect&bpo=25688]: Fixed file leak in ElementTree.iterparse() raising an error.
- bpo-23914 [https://bugs.python.org/issue?@action=redirect&bpo=23914]: Fixed SystemError raised by unpickler on broken pickle data.
- bpo-25691 [https://bugs.python.org/issue?@action=redirect&bpo=25691]: Fixed crash on deleting ElementTree.Element attributes.
- bpo-25624 [https://bugs.python.org/issue?@action=redirect&bpo=25624]: ZipFile now always writes a ZIP_STORED header for directory entries. Patch by Dingyuan Wang.
- Skip getaddrinfo if host is already resolved. Patch by A. Jesse Jiryu Davis.
- bpo-26050 [https://bugs.python.org/issue?@action=redirect&bpo=26050]: Add asyncio.StreamReader.readuntil() method. Patch by Марк Коренберг.
- bpo-25924 [https://bugs.python.org/issue?@action=redirect&bpo=25924]: Avoid unnecessary serialization of getaddrinfo(3) calls on OS X versions 10.5 or higher. Original

patch by A. Jesse Jiryu Davis.

- bpo-26406 [https://bugs.python.org/issue?@action=redirect&bpo=26406]: Avoid unnecessary serialization of getaddrinfo(3) calls on current versions of OpenBSD and NetBSD. Patch by A. Jesse Jiryu Davis.
- bpo-26848 [https://bugs.python.org/issue?@action=redirect&bpo=26848]: Fix asyncio/subprocess.communicate() to handle empty input. Patch by Jack O'Connor.
- bpo-27040 [https://bugs.python.org/issue?@action=redirect&bpo=27040]: Add loop.get_exception_handler method
- bpo-27041 [https://bugs.python.org/issue?@action=redirect&bpo=27041]: asyncio: Add loop.create_future method
- bpo-27223 [https://bugs.python.org/issue?@action=redirect&bpo=27223]: asyncio: Fix _read_ready and _write_ready to respect _conn_lost. Patch by Łukasz Langa.
- bpo-22970 [https://bugs.python.org/issue?@action=redirect&bpo=22970]: asyncio: Fix inconsistency cancelling Condition.wait. Patch by David Coles.

IDLE

- bpo-5124 [https://bugs.python.org/issue?@action=redirect&bpo=5124]: Paste with text selected now replaces the selection on X11. This matches how paste works on Windows, Mac, most modern Linux apps, and ttk widgets. Original patch by Serhiy Storchaka.
- bpo-24759 [https://bugs.python.org/issue?@action=redirect&bpo=24759]: Make clear in idlelib.idle_test.__init__ that the directory is a private implementation of test.test_idle and tool for maintainers.
- bpo-27196 [https://bugs.python.org/issue?@action=redirect&bpo=27196]: Stop "ThemeChanged" warnings when running IDLE tests. These persisted after other warnings were suppressed in #20567. Apply Serhiy Storchaka's update_idletasks solution to four test files. Record this additional advice in idle_test/README.txt
- bpo-20567 [https://bugs.python.org/issue?

@action=redirect&bpo=20567]: Revise idle_test/README.txt with advice about avoiding tk warning messages from tests. Apply advice to several IDLE tests.

- bpo-27117 [https://bugs.python.org/issue?@action=redirect&bpo=27117]: Make colorizer htest and turtledemo work with dark themes. Move code for configuring text widget colors to a new function.
- bpo-26673 [https://bugs.python.org/issue?@action=redirect&bpo=26673]: When tk reports font size as 0, change to size 10. Such fonts on Linux prevented the configuration dialog from opening.
- bpo-21939 [https://bugs.python.org/issue?@action=redirect&bpo=21939]: Add test for IDLE's percolator. Original patch by Saimadhav Heblikar.
- bpo-21676 [https://bugs.python.org/issue?@action=redirect&bpo=21676]: Add test for IDLE's replace dialog. Original patch by Saimadhav Heblikar.
- bpo-18410 [https://bugs.python.org/issue?@action=redirect&bpo=18410]: Add test for IDLE's search dialog. Original patch by Westley Martínez.
- bpo-21703 [https://bugs.python.org/issue?@action=redirect&bpo=21703]: Add test for IDLE's undo delegator. Original patch by Saimadhav Heblikar .
- bpo-27044 [https://bugs.python.org/issue?@action=redirect&bpo=27044]: Add ConfigDialog.remove_var_callbacks to stop memory leaks.
- bpo-23977 [https://bugs.python.org/issue?@action=redirect&bpo=23977]: Add more asserts to test_delegator.
- bpo-20640 [https://bugs.python.org/issue?@action=redirect&bpo=20640]: Add tests for idlelib.configHelpSourceEdit. Patch by Saimadhav Heblikar.
- In the "IDLE-console differences" section of the IDLE doc, clarify how running with IDLE affects sys.modules and the standard streams.
- bpo-25507 [https://bugs.python.org/issue?@action=redirect&bpo=25507]: fix incorrect change in IOBinding that prevented printing. Augment IOBinding htest to include all major IOBinding functions.
- bpo-25905 [https://bugs.python.org/issue?

@action=redirect&bpo=25905]: Revert unwanted conversion of ‘ to ’ RIGHT SINGLE QUOTATION MARK in README.txt and open this and NEWS.txt with “ascii”. Re-encode CREDITS.txt to utf-8 and open it with “utf-8”.

Documentation

- bpo-19489 [https://bugs.python.org/issue?@action=redirect&bpo=19489]: Moved the search box from the sidebar to the header and footer of each page. Patch by Ammar Askar.
- bpo-24136 [https://bugs.python.org/issue?@action=redirect&bpo=24136]: Document the new **PEP 448** [https://peps.python.org/pep-0448/] unpacking syntax of 3.5.
- bpo-26736 [https://bugs.python.org/issue?@action=redirect&bpo=26736]: Used HTTPS for external links in the documentation if possible.
- bpo-6953 [https://bugs.python.org/issue?@action=redirect&bpo=6953]: Rework the Readline module documentation to group related functions together, and add more details such as what underlying Readline functions and variables are accessed.
- bpo-23606 [https://bugs.python.org/issue?@action=redirect&bpo=23606]: Adds note to ctypes documentation regarding cdll.msvcrt.
- bpo-25500 [https://bugs.python.org/issue?@action=redirect&bpo=25500]: Fix documentation to not claim that _import_ is searched for in the global scope.
- bpo-26014 [https://bugs.python.org/issue?@action=redirect&bpo=26014]: Update 3.x packaging documentation: * «See also» links to the new docs are now provided in the legacy pages * links to setuptools documentation have been updated

Tests

- bpo-21916 [https://bugs.python.org/issue?@action=redirect&bpo=21916]: Added tests for the turtle module. Patch by ingrid, Gregory Loyse and Jelle Zijlstra.
- bpo-26523 [https://bugs.python.org/issue?@action=redirect&bpo=26523]: The multiprocessing thread pool

(multiprocessing.dummy.Pool) was untested.

- bpo-26015 [https://bugs.python.org/issue?@action=redirect&bpo=26015]: Added new tests for pickling iterators of mutable sequences.
- bpo-26325 [https://bugs.python.org/issue?@action=redirect&bpo=26325]: Added test.support.check_no_resource_warning() to check that no ResourceWarning is emitted.
- bpo-25940 [https://bugs.python.org/issue?@action=redirect&bpo=25940]: Changed test_ssl to use self-signed.pythontest.net. This avoids relying on svn.python.org, which recently changed root certificate.
- bpo-25616 [https://bugs.python.org/issue?@action=redirect&bpo=25616]: Tests for OrderedDict are extracted from test_collections into separate file test_ordered_dict.
- bpo-26583 [https://bugs.python.org/issue?@action=redirect&bpo=26583]: Skip test_timestamp_overflow in test_import if bytecode files cannot be written.

Build

- bpo-26884 [https://bugs.python.org/issue?@action=redirect&bpo=26884]: Fix linking extension modules for cross builds. Patch by Xavier de Gaye.
- bpo-22359 [https://bugs.python.org/issue?@action=redirect&bpo=22359]: Disable the rules for running _freeze_importlib and pgen when cross-compiling. The output of these programs is normally saved with the source code anyway, and is still regenerated when doing a native build. Patch by Xavier de Gaye.
- bpo-27229 [https://bugs.python.org/issue?@action=redirect&bpo=27229]: Fix the cross-compiling pgen rule for in-tree builds. Patch by Xavier de Gaye.
- bpo-21668 [https://bugs.python.org/issue?@action=redirect&bpo=21668]: Link audioop, _datetime, _ctypes_test modules to libm, except on Mac OS X. Patch written by Xavier de Gaye.
- bpo-25702 [https://bugs.python.org/issue?@action=redirect&bpo=25702]: A –with-lto configure option has

been added that will enable link time optimizations at build time during a make profile-opt. Some compilers and toolchains are known to not produce stable code when using LTO, be sure to test things thoroughly before relying on it. It can provide a few % speed up over profile-opt alone.

- bpo-26624 [https://bugs.python.org/issue?@action=redirect&bpo=26624]: Adds validation of ucrtbase[d].dll version with warning for old versions.
- bpo-17603 [https://bugs.python.org/issue?@action=redirect&bpo=17603]: Avoid error about nonexistent fileblocks.o file by using a lower-level check for st_blocks in struct stat.
- bpo-26079 [https://bugs.python.org/issue?@action=redirect&bpo=26079]: Fixing the build output folder for tix-8.4.3.6. Patch by Bjoern Thiel.
- bpo-26465 [https://bugs.python.org/issue?@action=redirect&bpo=26465]: Update Windows builds to use OpenSSL 1.0.2g.
- bpo-24421 [https://bugs.python.org/issue?@action=redirect&bpo=24421]: Compile Modules/_math.c once, before building extensions. Previously it could fail to compile properly if the math and cmath builds were concurrent.
- bpo-25348 [https://bugs.python.org/issue?@action=redirect&bpo=25348]: Added `--pgo` and `--pgo-job` arguments to `PCbuild\build.bat` for building with Profile-Guided Optimization. The old `PCbuild\build_pgo.bat` script is now deprecated, and simply calls `PCbuild\build.bat --pgo %*`.
- bpo-25827 [https://bugs.python.org/issue?@action=redirect&bpo=25827]: Add support for building with ICC to `configure`, including a new `--with-icc` flag.
- bpo-25696 [https://bugs.python.org/issue?@action=redirect&bpo=25696]: Fix installation of Python on UNIX with make -j9.
- bpo-26930 [https://bugs.python.org/issue?@action=redirect&bpo=26930]: Update OS X 10.5+ 32-bit-only installer to build and link with OpenSSL 1.0.2h.
- bpo-26268 [https://bugs.python.org/issue?@action=redirect&bpo=26268]: Update Windows builds to use OpenSSL 1.0.2f.

- bpo-25136 [https://bugs.python.org/issue?@action=redirect&bpo=25136]: Support Apple Xcode 7's new textual SDK stub libraries.
- bpo-24324 [https://bugs.python.org/issue?@action=redirect&bpo=24324]: Do not enable unreachable code warnings when using gcc as the option does not work correctly in older versions of gcc and has been silently removed as of gcc-4.5.

Windows

- bpo-27053 [https://bugs.python.org/issue?@action=redirect&bpo=27053]: Updates make_zip.py to correctly generate library ZIP file.
- bpo-26268 [https://bugs.python.org/issue?@action=redirect&bpo=26268]: Update the prepare_ssl.py script to handle OpenSSL releases that don't include the contents of the include directory (that is, 1.0.2e and later).
- bpo-26071 [https://bugs.python.org/issue?@action=redirect&bpo=26071]: bdist_wininst created binaries fail to start and find 32bit Python
- bpo-26073 [https://bugs.python.org/issue?@action=redirect&bpo=26073]: Update the list of magic numbers in launcher
- bpo-26065 [https://bugs.python.org/issue?@action=redirect&bpo=26065]: Excludes venv from library when generating embeddable distro.
- bpo-17500 [https://bugs.python.org/issue?@action=redirect&bpo=17500]: Remove unused and outdated icons. (See also: https://github.com/python/pythondotorg/issues/945)

Tools/Demos

- bpo-26799 [https://bugs.python.org/issue?@action=redirect&bpo=26799]: Fix python-gdb.py: don't get C types once when the Python code is loaded, but get C types on demand. The C types can change if python-gdb.py is loaded before the Python executable. Patch written by Thomas Ilsche.

- bpo-26271 [https://bugs.python.org/issue?@action=redirect&bpo=26271]: Fix the Freeze tool to properly use flags passed through configure. Patch by Daniel Shaulov.
- bpo-26489 [https://bugs.python.org/issue?@action=redirect&bpo=26489]: Add dictionary unpacking support to Tools/parser/unparse.py. Patch by Guo Ci Teo.
- bpo-26316 [https://bugs.python.org/issue?@action=redirect&bpo=26316]: Fix variable name typo in Argument Clinic.

Python 3.5.1 final

Release date: 2015-12-06

Core and Builtins

- bpo-25709 [https://bugs.python.org/issue?@action=redirect&bpo=25709]: Fixed problem with in-place string concatenation and utf-8 cache.

Windows

- bpo-25715 [https://bugs.python.org/issue?@action=redirect&bpo=25715]: Python 3.5.1 installer shows wrong upgrade path and incorrect logic for launcher detection.

Python 3.5.1 release candidate 1

Release date: 2015-11-22

Core and Builtins

- bpo-25630 [https://bugs.python.org/issue?@action=redirect&bpo=25630]: Fix a possible segfault during argument parsing in functions that accept filesystem paths.
- bpo-23564 [https://bugs.python.org/issue?@action=redirect&bpo=23564]: Fixed a partially broken sanity

check in the _posixsubprocess internals regarding how fds_to_pass were passed to the child. The bug had no actual impact as subprocess.py already avoided it.

- bpo-25388 [https://bugs.python.org/issue?@action=redirect&bpo=25388]: Fixed tokenizer crash when processing undecodable source code with a null byte.
- bpo-25462 [https://bugs.python.org/issue?@action=redirect&bpo=25462]: The hash of the key now is calculated only once in most operations in C implementation of OrderedDict.
- bpo-22995 [https://bugs.python.org/issue?@action=redirect&bpo=22995]: Default implementation of _reduce_ and _reduce_ex_ now rejects builtin types with not defined _new_.
- bpo-25555 [https://bugs.python.org/issue?@action=redirect&bpo=25555]: Fix parser and AST: fill lineno and col_offset of «arg» node when compiling AST from Python objects.
- bpo-24802 [https://bugs.python.org/issue?@action=redirect&bpo=24802]: Avoid buffer overreads when int(), float(), compile(), exec() and eval() are passed bytes-like objects. These objects are not necessarily terminated by a null byte, but the functions assumed they were.
- bpo-24726 [https://bugs.python.org/issue?@action=redirect&bpo=24726]: Fixed a crash and leaking NULL in repr() of OrderedDict that was mutated by direct calls of dict methods.
- bpo-25449 [https://bugs.python.org/issue?@action=redirect&bpo=25449]: Iterating OrderedDict with keys with unstable hash now raises KeyError in C implementations as well as in Python implementation.
- bpo-25395 [https://bugs.python.org/issue?@action=redirect&bpo=25395]: Fixed crash when highly nested OrderedDict structures were garbage collected.
- bpo-25274 [https://bugs.python.org/issue?@action=redirect&bpo=25274]: sys.setrecursionlimit() now raises a RecursionError if the new recursion limit is too low depending at the current recursion depth. Modify also the «lower-water mark» formula to make it monotonic. This mark is used to decide when the overflowed flag of the thread state

is reset.

- bpo-24402 [https://bugs.python.org/issue?@action=redirect&bpo=24402]: Fix input() to prompt to the redirected stdout when sys.stdout.fileno() fails.
- bpo-24806 [https://bugs.python.org/issue?@action=redirect&bpo=24806]: Prevent builtin types that are not allowed to be subclassed from being subclassed through multiple inheritance.
- bpo-24848 [https://bugs.python.org/issue?@action=redirect&bpo=24848]: Fixed a number of bugs in UTF-7 decoding of misformed data.
- bpo-25280 [https://bugs.python.org/issue?@action=redirect&bpo=25280]: Import trace messages emitted in verbose (-v) mode are no longer formatted twice.
- bpo-25003 [https://bugs.python.org/issue?@action=redirect&bpo=25003]: On Solaris 11.3 or newer, os.urandom() now uses the getrandom() function instead of the getentropy() function. The getentropy() function is blocking to generate very good quality entropy, os.urandom() doesn't need such high-quality entropy.
- bpo-25182 [https://bugs.python.org/issue?@action=redirect&bpo=25182]: The stdprinter (used as sys.stderr before the io module is imported at startup) now uses the backslashreplace error handler.
- bpo-25131 [https://bugs.python.org/issue?@action=redirect&bpo=25131]: Make the line number and column offset of set/dict literals and comprehensions correspond to the opening brace.
- bpo-25150 [https://bugs.python.org/issue?@action=redirect&bpo=25150]: Hide the private `_Py_atomic_xxx` symbols from the public Python.h header to fix a compilation error with OpenMP. PyThreadState_GET() becomes an alias to PyThreadState_Get() to avoid ABI incompatibilities.

Library

- bpo-25626 [https://bugs.python.org/issue?@action=redirect&bpo=25626]: Change three zlib functions to accept sizes that fit in Py_ssize_t, but internally cap those sizes to UINT_MAX. This resolves a regression in 3.5 where GzipFile.read() failed to read chunks larger than 2 or 4 GiB. The change affects the zlib.Decompress.decompress() max_length parameter, the zlib.decompress() bufsize parameter, and the zlib.Decompress.flush() length parameter.

- bpo-25583 [https://bugs.python.org/issue?@action=redirect&bpo=25583]: Avoid incorrect errors raised by os.makedirs(exist_ok=True) when the OS gives priority to errors such as EACCES over EEXIST.
- bpo-25593 [https://bugs.python.org/issue?@action=redirect&bpo=25593]: Change semantics of EventLoop.stop() in asyncio.
- bpo-6973 [https://bugs.python.org/issue?@action=redirect&bpo=6973]: When we know a subprocess.Popen process has died, do not allow the send_signal(), terminate(), or kill() methods to do anything as they could potentially signal a different process.
- bpo-25590 [https://bugs.python.org/issue?@action=redirect&bpo=25590]: In the Readline completer, only call getattr() once per attribute.
- bpo-25498 [https://bugs.python.org/issue?@action=redirect&bpo=25498]: Fix a crash when garbage-collecting ctypes objects created by wrapping a memoryview. This was a regression made in 3.5a1. Based on patch by Eryksun.
- bpo-25584 [https://bugs.python.org/issue?@action=redirect&bpo=25584]: Added «escape» to the _all_ list in the glob module.
- bpo-25584 [https://bugs.python.org/issue?@action=redirect&bpo=25584]: Fixed recursive glob() with patterns starting with **.
- bpo-25446 [https://bugs.python.org/issue?@action=redirect&bpo=25446]: Fix regression in smtplib’s AUTH LOGIN support.
- bpo-18010 [https://bugs.python.org/issue?@action=redirect&bpo=18010]: Fix the pydoc web server’s module search function to handle exceptions from importing packages.
- bpo-25554 [https://bugs.python.org/issue?@action=redirect&bpo=25554]: Got rid of circular references in regular expression parsing.

- bpo-25510 [https://bugs.python.org/issue?@action=redirect&bpo=25510]: fileinput.FileInput.readline() now returns b”” instead of “” at the end if the FileInput was opened with binary mode. Patch by Ryosuke Ito.
- bpo-25503 [https://bugs.python.org/issue?@action=redirect&bpo=25503]: Fixed inspect.getdoc() for inherited docstrings of properties. Original patch by John Mark Vandenberg.
- bpo-25515 [https://bugs.python.org/issue?@action=redirect&bpo=25515]: Always use os.urandom as a source of randomness in uuid.uuid4.
- bpo-21827 [https://bugs.python.org/issue?@action=redirect&bpo=21827]: Fixed textwrap.dedent() for the case when largest common whitespace is a substring of smallest leading whitespace. Based on patch by Robert Li.
- bpo-25447 [https://bugs.python.org/issue?@action=redirect&bpo=25447]: The lru_cache() wrapper objects now can be copied and pickled (by returning the original object unchanged).
- bpo-25390 [https://bugs.python.org/issue?@action=redirect&bpo=25390]: typing: Don’t crash on Union[str, Pattern].
- bpo-25441 [https://bugs.python.org/issue?@action=redirect&bpo=25441]: asyncio: Raise error from drain() when socket is closed.
- bpo-25410 [https://bugs.python.org/issue?@action=redirect&bpo=25410]: Cleaned up and fixed minor bugs in C implementation of OrderedDict.
- bpo-25411 [https://bugs.python.org/issue?@action=redirect&bpo=25411]: Improved Unicode support in SMTPHandler through better use of the email package. Thanks to user simon04 for the patch.
- bpo-25407 [https://bugs.python.org/issue?@action=redirect&bpo=25407]: Remove mentions of the formatter module being removed in Python 3.6.
- bpo-25406 [https://bugs.python.org/issue?@action=redirect&bpo=25406]: Fixed a bug in C implementation of OrderedDict.move_to_end() that caused segmentation fault or hang in iterating after moving several items to the start of ordered dict.

- bpo-25364 [https://bugs.python.org/issue?@action=redirect&bpo=25364]: zipfile now works in threads disabled builds.
- bpo-25328 [https://bugs.python.org/issue?@action=redirect&bpo=25328]: smtpd's SMTPChannel now correctly raises a ValueError if both decode_data and enable_SMTPUTF8 are set to true.
- bpo-25316 [https://bugs.python.org/issue?@action=redirect&bpo=25316]: distutils raises OSError instead of DistutilsPlatformError when MSVC is not installed.
- bpo-25380 [https://bugs.python.org/issue?@action=redirect&bpo=25380]: Fixed protocol for the STACK_GLOBAL opcode in pickletools.opcodes.
- bpo-23972 [https://bugs.python.org/issue?@action=redirect&bpo=23972]: Updates asyncio datagram create method allowing reuseport and reuseaddr socket options to be set prior to binding the socket. Mirroring the existing asyncio create_server method the reuseaddr option for datagram sockets defaults to True if the O/S is "posix" (except if the platform is Cygwin). Patch by Chris Laws.
- bpo-25304 [https://bugs.python.org/issue?@action=redirect&bpo=25304]: Add asyncio.run_coroutine_threadsafe(). This lets you submit a coroutine to a loop from another thread, returning a concurrent.futures.Future. By Vincent Michel.
- bpo-25232 [https://bugs.python.org/issue?@action=redirect&bpo=25232]: Fix CGIRequestHandler to split the query from the URL at the first question mark (?) rather than the last. Patch from Xiang Zhang.
- bpo-24657 [https://bugs.python.org/issue?@action=redirect&bpo=24657]: Prevent CGIRequestHandler from collapsing slashes in the query part of the URL as if it were a path. Patch from Xiang Zhang.
- bpo-24483 [https://bugs.python.org/issue?@action=redirect&bpo=24483]: C implementation of functools.lru_cache() now calculates key's hash only once.
- bpo-22958 [https://bugs.python.org/issue?@action=redirect&bpo=22958]: Constructor and update method of weakref.WeakValueDictionary now accept the self and the dict keyword arguments.

- bpo-22609 [https://bugs.python.org/issue?@action=redirect&bpo=22609]: Constructor of collections.UserDict now accepts the self keyword argument.
- bpo-25111 [https://bugs.python.org/issue?@action=redirect&bpo=25111]: Fixed comparison of traceback.FrameSummary.
- bpo-25262 [https://bugs.python.org/issue?@action=redirect&bpo=25262]: Added support for BINBYTES8 opcode in Python implementation of unpickler. Highest 32 bits of 64-bit size for BINUNICODE8 and BINBYTES8 opcodes no longer silently ignored on 32-bit platforms in C implementation.
- bpo-25034 [https://bugs.python.org/issue?@action=redirect&bpo=25034]: Fix string.Formatter problem with auto-numbering and nested format_specs. Patch by Anthon van der Neut.
- bpo-25233 [https://bugs.python.org/issue?@action=redirect&bpo=25233]: Rewrite the guts of asyncio.Queue and asyncio.Semaphore to be more understandable and correct.
- bpo-25203 [https://bugs.python.org/issue?@action=redirect&bpo=25203]: Failed readline.set_completer_delims() no longer left the module in inconsistent state.
- bpo-23600 [https://bugs.python.org/issue?@action=redirect&bpo=23600]: Default implementation of tzinfo.fromutc() was returning wrong results in some cases.
- bpo-23329 [https://bugs.python.org/issue?@action=redirect&bpo=23329]: Allow the ssl module to be built with older versions of LibreSSL.
- Prevent overflow in _Unpickler_Read.
- bpo-25047 [https://bugs.python.org/issue?@action=redirect&bpo=25047]: The XML encoding declaration written by Element Tree now respects the letter case given by the user. This restores the ability to write encoding names in uppercase like «UTF-8», which worked in Python 2.
- bpo-25135 [https://bugs.python.org/issue?@action=redirect&bpo=25135]: Make deque_clear() safer by emptying the deque before clearing. This helps avoid possible reentrancy issues.

- bpo-19143 [https://bugs.python.org/issue?@action=redirect&bpo=19143]: platform module now reads Windows version from kernel32.dll to avoid compatibility shims.
- bpo-25092 [https://bugs.python.org/issue?@action=redirect&bpo=25092]: Fix datetime.strftime() failure when errno was already set to EINVAL.
- bpo-23517 [https://bugs.python.org/issue?@action=redirect&bpo=23517]: Fix rounding in fromtimestamp() and utcfromtimestamp() methods of datetime.datetime: microseconds are now rounded to nearest with ties going to nearest even integer (ROUND_HALF_EVEN), instead of being rounding towards minus infinity (ROUND_FLOOR). It's important that these methods use the same rounding mode than datetime.timedelta to keep the property: (datetime(1970,1,1) + timedelta(seconds=t)) == datetime.utcfromtimestamp(t). It also the rounding mode used by round(float) for example.
- bpo-25155 [https://bugs.python.org/issue?@action=redirect&bpo=25155]: Fix datetime.datetime.now() and datetime.datetime.utcnow() on Windows to support date after year 2038. It was a regression introduced in Python 3.5.0.
- bpo-25108 [https://bugs.python.org/issue?@action=redirect&bpo=25108]: Omitted internal frames in traceback functions print_stack(), format_stack(), and extract_stack() called without arguments.
- bpo-25118 [https://bugs.python.org/issue?@action=redirect&bpo=25118]: Fix a regression of Python 3.5.0 in os.waitpid() on Windows.
- bpo-24684 [https://bugs.python.org/issue?@action=redirect&bpo=24684]: socket.socket.getaddrinfo() now calls PyUnicode_AsEncodedString() instead of calling the encode() method of the host, to handle correctly custom string with an encode() method which doesn't return a byte string. The encoder of the IDNA codec is now called directly instead of calling the encode() method of the string.
- bpo-25060 [https://bugs.python.org/issue?@action=redirect&bpo=25060]: Correctly compute stack usage of the BUILD_MAP opcode.
- bpo-24857 [https://bugs.python.org/issue?

@action=redirect&bpo=24857]: Comparing call_args to a long sequence now correctly returns a boolean result instead of raising an exception. Patch by A Kaptur.

- bpo-23144 [https://bugs.python.org/issue?@action=redirect&bpo=23144]: Make sure that HTMLParser.feed() returns all the data, even when convert_charrefs is True.
- bpo-24982 [https://bugs.python.org/issue?@action=redirect&bpo=24982]: shutil.make_archive() with the «zip» format now adds entries for directories (including empty directories) in ZIP file.
- bpo-25019 [https://bugs.python.org/issue?@action=redirect&bpo=25019]: Fixed a crash caused by setting non-string key of expat parser. Based on patch by John Leitch.
- bpo-16180 [https://bugs.python.org/issue?@action=redirect&bpo=16180]: Exit pdb if file has syntax error, instead of trapping user in an infinite loop. Patch by Xavier de Gaye.
- bpo-24891 [https://bugs.python.org/issue?@action=redirect&bpo=24891]: Fix a race condition at Python startup if the file descriptor of stdin (0), stdout (1) or stderr (2) is closed while Python is creating sys.stdin, sys.stdout and sys.stderr objects. These attributes are now set to None if the creation of the object failed, instead of raising an OSError exception. Initial patch written by Marco Paolini.
- bpo-24992 [https://bugs.python.org/issue?@action=redirect&bpo=24992]: Fix error handling and a race condition (related to garbage collection) in collections.OrderedDict constructor.
- bpo-24881 [https://bugs.python.org/issue?@action=redirect&bpo=24881]: Fixed setting binary mode in Python implementation of FileIO on Windows and Cygwin. Patch from Akira Li.
- bpo-25578 [https://bugs.python.org/issue?@action=redirect&bpo=25578]: Fix (another) memory leak in SSLSocket.getpeercer().
- bpo-25530 [https://bugs.python.org/issue?@action=redirect&bpo=25530]: Disable the vulnerable SSLv3 protocol by default when creating ssl.SSLContext.

- bpo-25569 [https://bugs.python.org/issue?@action=redirect&bpo=25569]: Fix memory leak in SSLSocket.getpeercert().
- bpo-25471 [https://bugs.python.org/issue?@action=redirect&bpo=25471]: Sockets returned from accept() shouldn't appear to be nonblocking.
- bpo-25319 [https://bugs.python.org/issue?@action=redirect&bpo=25319]: When threading.Event is reinitialized, the underlying condition should use a regular lock rather than a recursive lock.
- bpo-21112 [https://bugs.python.org/issue?@action=redirect&bpo=21112]: Fix regression in unittest.expectedFailure on subclasses. Patch from Berker Peksag.
- bpo-24764 [https://bugs.python.org/issue?@action=redirect&bpo=24764]: cgi.FieldStorage.read_multi() now ignores the Content-Length header in part headers. Patch written by Peter Landry and reviewed by Pierre Quentel.
- bpo-24913 [https://bugs.python.org/issue?@action=redirect&bpo=24913]: Fix overrun error in deque.index(). Found by John Leitch and Bryce Darling.
- bpo-24774 [https://bugs.python.org/issue?@action=redirect&bpo=24774]: Fix docstring in http.server.test. Patch from Chiu-Hsiang Hsu.
- bpo-21159 [https://bugs.python.org/issue?@action=redirect&bpo=21159]: Improve message in configparser.InterpolationMissingOptionError. Patch from Łukasz Langa.
- bpo-20362 [https://bugs.python.org/issue?@action=redirect&bpo=20362]: Honour TestCase.longMessage correctly in assertRegex. Patch from Ilia Kurenkov.
- bpo-23572 [https://bugs.python.org/issue?@action=redirect&bpo=23572]: Fixed functools.singledispatch on classes with false metaclasses. Patch by Ethan Furman.
- asyncio: ensure_future() now accepts awaitable objects.

IDLE

- bpo-15348 [https://bugs.python.org/issue?@action=redirect&bpo=15348]: Stop the debugger engine (normally in a user process) before closing the debugger window (running in the IDLE process). This prevents the RuntimeErrors that were being caught and ignored.

- bpo-24455 [https://bugs.python.org/issue?@action=redirect&bpo=24455]: Prevent IDLE from hanging when a) closing the shell while the debugger is active (15347); b) closing the debugger with the [X] button (15348); and c) activating the debugger when already active (24455). The patch by Mark Roseman does this by making two changes. 1. Suspend and resume the gui.interaction method with the tcl vwait mechanism intended for this purpose (instead of root.mainloop & .quit). 2. In gui.run, allow any existing interaction to terminate first.
- Change “The program” to “Your program” in an IDLE “kill program?” message to make it clearer that the program referred to is the currently running user program, not IDLE itself.
- bpo-24750 [https://bugs.python.org/issue?@action=redirect&bpo=24750]: Improve the appearance of the IDLE editor window status bar. Patch by Mark Roseman.
- bpo-25313 [https://bugs.python.org/issue?@action=redirect&bpo=25313]: Change the handling of new built-in text color themes to better address the compatibility problem introduced by the addition of IDLE Dark. Consistently use the revised idleConf.CurrentTheme everywhere in idlelib.
- bpo-24782 [https://bugs.python.org/issue?@action=redirect&bpo=24782]: Extension configuration is now a tab in the IDLE Preferences dialog rather than a separate dialog. The former tabs are now a sorted list. Patch by Mark Roseman.
- bpo-22726 [https://bugs.python.org/issue?@action=redirect&bpo=22726]: Re-activate the config dialog help button with some content about the other buttons and the new IDLE Dark theme.
- bpo-24820 [https://bugs.python.org/issue?@action=redirect&bpo=24820]: IDLE now has an “IDLE Dark” built-in text color theme. It is more or less IDLE Classic inverted, with a cobalt blue background. Strings, comments, keywords, … are still green, red, orange, … . To use it with

IDLEs released before November 2015, hit the “Save as New Custom Theme” button and enter a new name, such as “Custom Dark”. The custom theme will work with any IDLE release, and can be modified.

- bpo-25224 [https://bugs.python.org/issue?@action=redirect&bpo=25224]: README.txt is now an idlelib index for IDLE developers and curious users. The previous user content is now in the IDLE doc chapter. “IDLE” now means “Integrated Development and Learning Environment”.
- bpo-24820 [https://bugs.python.org/issue?@action=redirect&bpo=24820]: Users can now set breakpoint colors in Settings -> Custom Highlighting. Original patch by Mark Roseman.
- bpo-24972 [https://bugs.python.org/issue?@action=redirect&bpo=24972]: Inactive selection background now matches active selection background, as configured by users, on all systems. Found items are now always highlighted on Windows. Initial patch by Mark Roseman.
- bpo-24570 [https://bugs.python.org/issue?@action=redirect&bpo=24570]: Idle: make calltip and completion boxes appear on Macs affected by a tk regression. Initial patch by Mark Roseman.
- bpo-24988 [https://bugs.python.org/issue?@action=redirect&bpo=24988]: Idle ScrolledList context menus (used in debugger) now work on Mac Aqua. Patch by Mark Roseman.
- bpo-24801 [https://bugs.python.org/issue?@action=redirect&bpo=24801]: Make right-click for context menu work on Mac Aqua. Patch by Mark Roseman.
- bpo-25173 [https://bugs.python.org/issue?@action=redirect&bpo=25173]: Associate tkinter messageboxes with a specific widget. For Mac OSX, make them a “sheet”. Patch by Mark Roseman.
- bpo-25198 [https://bugs.python.org/issue?@action=redirect&bpo=25198]: Enhance the initial html viewer now used for Idle Help. Properly indent fixed-pitch text (patch by Mark Roseman). Give code snippet a very Sphinx-like light blueish-gray background. Re-use initial width and height set by users for shell and editor. When the Table of Contents (TOC) menu is used, put the section header at the

top of the screen.

- bpo-25225 [https://bugs.python.org/issue?@action=redirect&bpo=25225]: Condense and rewrite Idle doc section on text colors.
- bpo-21995 [https://bugs.python.org/issue?@action=redirect&bpo=21995]: Explain some differences between IDLE and console Python.
- bpo-22820 [https://bugs.python.org/issue?@action=redirect&bpo=22820]: Explain need for *print* when running file from Idle editor.
- bpo-25224 [https://bugs.python.org/issue?@action=redirect&bpo=25224]: Doc: augment Idle feature list and no-subprocess section.
- bpo-25219 [https://bugs.python.org/issue?@action=redirect&bpo=25219]: Update doc for Idle command line options. Some were missing and notes were not correct.
- bpo-24861 [https://bugs.python.org/issue?@action=redirect&bpo=24861]: Most of idlelib is private and subject to change. Use idleib.idle.* to start Idle. See idlelib._init_._doc_.
- bpo-25199 [https://bugs.python.org/issue?@action=redirect&bpo=25199]: Idle: add synchronization comments for future maintainers.
- bpo-16893 [https://bugs.python.org/issue?@action=redirect&bpo=16893]: Replace help.txt with help.html for Idle doc display. The new idlelib/help.html is rstripped Doc/build/html/library/idle.html. It looks better than help.txt and will better document Idle as released. The tkinter html viewer that works for this file was written by Mark Roseman. The now unused EditorWindow.HelpDialog class and helt.txt file are deprecated.
- bpo-24199 [https://bugs.python.org/issue?@action=redirect&bpo=24199]: Deprecate unused idlelib.idlever with possible removal in 3.6.
- bpo-24790 [https://bugs.python.org/issue?@action=redirect&bpo=24790]: Remove extraneous code (which also create 2 & 3 conflicts).

Documentation

- bpo-22558 [https://bugs.python.org/issue?@action=redirect&bpo=22558]: Add remaining doc links to source code for Python-coded modules. Patch by Yoni Lavi.
- bpo-12067 [https://bugs.python.org/issue?@action=redirect&bpo=12067]: Rewrite Comparisons section in the Expressions chapter of the language reference. Some of the details of comparing mixed types were incorrect or ambiguous. NotImplemented is only relevant at a lower level than the Expressions chapter. Added details of comparing range() objects, and default behaviour and consistency suggestions for user-defined classes. Patch from Andy Maier.
- bpo-24952 [https://bugs.python.org/issue?@action=redirect&bpo=24952]: Clarify the default size argument of stack_size() in the «threading» and «_thread» modules. Patch from Mattip.
- bpo-23725 [https://bugs.python.org/issue?@action=redirect&bpo=23725]: Overhaul tempfile docs. Note deprecated status of mktemp. Patch from Zbigniew Jędrzejewski-Szmek.
- bpo-24808 [https://bugs.python.org/issue?@action=redirect&bpo=24808]: Update the types of some PyTypeObject fields. Patch by Joseph Weston.
- bpo-22812 [https://bugs.python.org/issue?@action=redirect&bpo=22812]: Fix unittest discovery examples. Patch from Pam McA'Nulty.

Tests

- bpo-25449 [https://bugs.python.org/issue?@action=redirect&bpo=25449]: Added tests for OrderedDict subclasses.
- bpo-25099 [https://bugs.python.org/issue?@action=redirect&bpo=25099]: Make test_compileall not fail when an entry on sys.path cannot be written to (commonly seen in administrative installs on Windows).
- bpo-23919 [https://bugs.python.org/issue?@action=redirect&bpo=23919]: Prevents assert dialogs appearing in the test suite.
- `PCbuild\rt.bat` now accepts an unlimited number of arguments to pass along to regrtest.py. Previously there was a

limit of 9.

Build

- bpo-24915 [https://bugs.python.org/issue?@action=redirect&bpo=24915]: Add LLVM support for PGO builds and use the test suite to generate the profile data. Initial patch by Alecsandru Patrascu of Intel.
- bpo-24910 [https://bugs.python.org/issue?@action=redirect&bpo=24910]: Windows MSIs now have unique display names.
- bpo-24986 [https://bugs.python.org/issue?@action=redirect&bpo=24986]: It is now possible to build Python on Windows without errors when external libraries are not available.

Windows

- bpo-25450 [https://bugs.python.org/issue?@action=redirect&bpo=25450]: Updates shortcuts to start Python in installation directory.
- bpo-25164 [https://bugs.python.org/issue?@action=redirect&bpo=25164]: Changes default all-users install directory to match per-user directory.
- bpo-25143 [https://bugs.python.org/issue?@action=redirect&bpo=25143]: Improves installer error messages for unsupported platforms.
- bpo-25163 [https://bugs.python.org/issue?@action=redirect&bpo=25163]: Display correct directory in installer when using non-default settings.
- bpo-25361 [https://bugs.python.org/issue?@action=redirect&bpo=25361]: Disables use of SSE2 instructions in Windows 32-bit build
- bpo-25089 [https://bugs.python.org/issue?@action=redirect&bpo=25089]: Adds logging to installer for case where launcher is not selected on upgrade.
- bpo-25165 [https://bugs.python.org/issue?@action=redirect&bpo=25165]: Windows uninstallation should not remove launcher if other versions remain
- bpo-25112 [https://bugs.python.org/issue?

@action=redirect&bpo=25112]: py.exe launcher is missing icons
- bpo-25102 [https://bugs.python.org/issue?@action=redirect&bpo=25102]: Windows installer does not precompile for -O or -OO.
- bpo-25081 [https://bugs.python.org/issue?@action=redirect&bpo=25081]: Makes Back button in installer go back to upgrade page when upgrading.
- bpo-25091 [https://bugs.python.org/issue?@action=redirect&bpo=25091]: Increases font size of the installer.
- bpo-25126 [https://bugs.python.org/issue?@action=redirect&bpo=25126]: Clarifies that the non-web installer will download some components.
- bpo-25213 [https://bugs.python.org/issue?@action=redirect&bpo=25213]: Restores requestedExecutionLevel to manifest to disable UAC virtualization.
- bpo-25022 [https://bugs.python.org/issue?@action=redirect&bpo=25022]: Removed very outdated PC/example_nt/ directory.

Tools/Demos

- bpo-25440 [https://bugs.python.org/issue?@action=redirect&bpo=25440]: Fix output of python-config –extension-suffix.

Python 3.5.0 final

Release date: 2015-09-13

Build

- bpo-25071 [https://bugs.python.org/issue?@action=redirect&bpo=25071]: Windows installer should not require TargetDir parameter when installing quietly.

Python 3.5.0 release candidate 4

Release date: 2015-09-09

Library

- bpo-25029 [https://bugs.python.org/issue?@action=redirect&bpo=25029]: Fixes MemoryError in test_strptime.

Build

- bpo-25027 [https://bugs.python.org/issue?@action=redirect&bpo=25027]: Reverts partial-static build options and adds vcruntime140.dll to Windows installation.

Python 3.5.0 release candidate 3

Release date: 2015-09-07

Core and Builtins

- bpo-24305 [https://bugs.python.org/issue?@action=redirect&bpo=24305]: Prevent import subsystem stack frames from being counted by the warnings.warn(stacklevel=) parameter.
- bpo-24912 [https://bugs.python.org/issue?@action=redirect&bpo=24912]: Prevent _class_ assignment to immutable built-in objects.
- bpo-24975 [https://bugs.python.org/issue?@action=redirect&bpo=24975]: Fix AST compilation for **PEP 448** [https://peps.python.org/pep-0448/] syntax.

Library

- bpo-24917 [https://bugs.python.org/issue?@action=redirect&bpo=24917]: time_strftime() buffer over-read.
- bpo-24748 [https://bugs.python.org/issue?@action=redirect&bpo=24748]: To resolve a compatibility problem found with py2exe and pywin32, imp.load_dynamic() once again ignores previously loaded modules to support Python modules replacing themselves with extension modules. Patch by Petr Viktorin.

- bpo-24635 [https://bugs.python.org/issue?@action=redirect&bpo=24635]: Fixed a bug in typing.py where isinstance([], typing.Iterable) would return True once, then False on subsequent calls.
- bpo-24989 [https://bugs.python.org/issue?@action=redirect&bpo=24989]: Fixed buffer overread in BytesIO.readline() if a position is set beyond size. Based on patch by John Leitch.
- bpo-24913 [https://bugs.python.org/issue?@action=redirect&bpo=24913]: Fix overrun error in deque.index(). Found by John Leitch and Bryce Darling.

Python 3.5.0 release candidate 2

Release date: 2015-08-25

Core and Builtins

- bpo-24769 [https://bugs.python.org/issue?@action=redirect&bpo=24769]: Interpreter now starts properly when dynamic loading is disabled. Patch by Petr Viktorin.
- bpo-21167 [https://bugs.python.org/issue?@action=redirect&bpo=21167]: NAN operations are now handled correctly when python is compiled with ICC even if -fp-model strict is not specified.
- bpo-24492 [https://bugs.python.org/issue?@action=redirect&bpo=24492]: A «package» lacking a __name__ attribute when trying to perform a `from .. import ...` statement will trigger an ImportError instead of an AttributeError.

Library

- bpo-24847 [https://bugs.python.org/issue?@action=redirect&bpo=24847]: Removes vcruntime140.dll dependency from Tcl/Tk.
- bpo-24839 [https://bugs.python.org/issue?@action=redirect&bpo=24839]: platform._syscmd_ver raises DeprecationWarning

- bpo-24867 [https://bugs.python.org/issue?@action=redirect&bpo=24867]: Fix Task.get_stack() for "async def" coroutines

Python 3.5.0 release candidate 1

Release date: 2015-08-09

Core and Builtins

- bpo-24667 [https://bugs.python.org/issue?@action=redirect&bpo=24667]: Resize odict in all cases that the underlying dict resizes.

Library

- bpo-24824 [https://bugs.python.org/issue?@action=redirect&bpo=24824]: Signatures of codecs.encode() and codecs.decode() now are compatible with pydoc.
- bpo-24634 [https://bugs.python.org/issue?@action=redirect&bpo=24634]: Importing uuid should not try to load libc on Windows
- bpo-24798 [https://bugs.python.org/issue?@action=redirect&bpo=24798]: _msvccompiler.py doesn't properly support manifests
- bpo-4395 [https://bugs.python.org/issue?@action=redirect&bpo=4395]: Better testing and documentation of binary operators. Patch by Martin Panter.
- bpo-23973 [https://bugs.python.org/issue?@action=redirect&bpo=23973]: Update typing.py from GitHub repo.
- bpo-23004 [https://bugs.python.org/issue?@action=redirect&bpo=23004]: mock_open() now reads binary data correctly when the type of read_data is bytes. Initial patch by Aaron Hill.
- bpo-23888 [https://bugs.python.org/issue?@action=redirect&bpo=23888]: Handle fractional time in cookie expiry. Patch by ssh.
- bpo-23652 [https://bugs.python.org/issue?

@action=redirect&bpo=23652]: Make it possible to compile the select module against the libc headers from the Linux Standard Base, which do not include some EPOLL macros. Patch by Matt Frank.

- bpo-22932 [https://bugs.python.org/issue?@action=redirect&bpo=22932]: Fix timezones in email.utils.formatdate. Patch from Dmitry Shachnev.
- bpo-23779 [https://bugs.python.org/issue?@action=redirect&bpo=23779]: imaplib raises TypeError if authenticator tries to abort. Patch from Craig Holmquist.
- bpo-23319 [https://bugs.python.org/issue?@action=redirect&bpo=23319]: Fix ctypes.BigEndianStructure, swap correctly bytes. Patch written by Matthieu Gautier.
- bpo-23254 [https://bugs.python.org/issue?@action=redirect&bpo=23254]: Document how to close the TCPServer listening socket. Patch from Martin Panter.
- bpo-19450 [https://bugs.python.org/issue?@action=redirect&bpo=19450]: Update Windows and OS X installer builds to use SQLite 3.8.11.
- bpo-17527 [https://bugs.python.org/issue?@action=redirect&bpo=17527]: Add PATCH to wsgiref.validator. Patch from Luca Sbardella.
- bpo-24791 [https://bugs.python.org/issue?@action=redirect&bpo=24791]: Fix grammar regression for call syntax: “g(*a or b)”.

IDLE

- bpo-23672 [https://bugs.python.org/issue?@action=redirect&bpo=23672]: Allow Idle to edit and run files with astral chars in name. Patch by Mohd Sanad Zaki Rizvi.
- bpo-24745 [https://bugs.python.org/issue?@action=redirect&bpo=24745]: Idle editor default font. Switch from Courier to platform-sensitive TkFixedFont. This should not affect current customized font selections. If there is a problem, edit $HOME/.idlerc/config-main.cfg and remove “`font`*xxx*” entries from [Editor Window]. Patch by Mark Roseman.
- bpo-21192 [https://bugs.python.org/issue?@action=redirect&bpo=21192]: Idle editor. When a file is run, put

its name in the restart bar. Do not print false prompts. Original patch by Adnan Umer.
- bpo-13884 [https://bugs.python.org/issue?@action=redirect&bpo=13884]: Idle menus. Remove tearoff lines. Patch by Roger Serwy.

Documentation

- bpo-24129 [https://bugs.python.org/issue?@action=redirect&bpo=24129]: Clarify the reference documentation for name resolution. This includes removing the assumption that readers will be familiar with the name resolution scheme Python used prior to the introduction of lexical scoping for function namespaces. Patch by Ivan Levkivskyi.
- bpo-20769 [https://bugs.python.org/issue?@action=redirect&bpo=20769]: Improve reload() docs. Patch by Dorian Pula.
- bpo-23589 [https://bugs.python.org/issue?@action=redirect&bpo=23589]: Remove duplicate sentence from the FAQ. Patch by Yongzhi Pan.
- bpo-24729 [https://bugs.python.org/issue?@action=redirect&bpo=24729]: Correct IO tutorial to match implementation regarding encoding parameter to open function.

Tests

- bpo-24751 [https://bugs.python.org/issue?@action=redirect&bpo=24751]: When running regrtest with the `-w` command line option, a test run is no longer marked as a failure if all tests succeed when re-run.

Python 3.5.0 beta 4

Release date: 2015-07-26

Core and Builtins

- bpo-23573 [https://bugs.python.org/issue?@action=redirect&bpo=23573]: Restored optimization of bytes.rfind() and bytearray.rfind() for single-byte argument on Linux.
- bpo-24569 [https://bugs.python.org/issue?@action=redirect&bpo=24569]: Make **PEP 448** [https://peps.python.org/pep-0448/] dictionary evaluation more consistent.
- bpo-24583 [https://bugs.python.org/issue?@action=redirect&bpo=24583]: Fix crash when set is mutated while being updated.
- bpo-24407 [https://bugs.python.org/issue?@action=redirect&bpo=24407]: Fix crash when dict is mutated while being updated.
- bpo-24619 [https://bugs.python.org/issue?@action=redirect&bpo=24619]: New approach for tokenizing async/await. As a consequence, it is now possible to have one-line “async def foo(): await ..” functions.
- bpo-24687 [https://bugs.python.org/issue?@action=redirect&bpo=24687]: Plug refleak on SyntaxError in function parameters annotations.
- bpo-15944 [https://bugs.python.org/issue?@action=redirect&bpo=15944]: memoryview: Allow arbitrary formats when casting to bytes. Patch by Martin Panter.

Library

- bpo-23441 [https://bugs.python.org/issue?@action=redirect&bpo=23441]: rcompleter now prints a tab character instead of displaying possible completions for an empty word. Initial patch by Martin Sekera.
- bpo-24683 [https://bugs.python.org/issue?@action=redirect&bpo=24683]: Fixed crashes in _json functions called with arguments of inappropriate type.
- bpo-21697 [https://bugs.python.org/issue?@action=redirect&bpo=21697]: shutil.copytree() now correctly handles symbolic links that point to directories. Patch by Eduardo Seabra and Thomas Kluyver.
- bpo-14373 [https://bugs.python.org/issue?@action=redirect&bpo=14373]: Fixed segmentation fault when gc.collect() is called during constructing lru_cache (C

implementation).

- bpo-24695 [https://bugs.python.org/issue?@action=redirect&bpo=24695]: Fix a regression in traceback.print_exception(). If exc_traceback is None we shouldn't print a traceback header like described in the documentation.
- bpo-24620 [https://bugs.python.org/issue?@action=redirect&bpo=24620]: Random.setstate() now validates the value of state last element.
- bpo-22485 [https://bugs.python.org/issue?@action=redirect&bpo=22485]: Fixed an issue that caused `inspect.getsource` to return incorrect results on nested functions.
- bpo-22153 [https://bugs.python.org/issue?@action=redirect&bpo=22153]: Improve unittest docs. Patch from Martin Panter and evilzero.
- bpo-24580 [https://bugs.python.org/issue?@action=redirect&bpo=24580]: Symbolic group references to open group in re patterns now are explicitly forbidden as well as numeric group references.
- bpo-24206 [https://bugs.python.org/issue?@action=redirect&bpo=24206]: Fixed _eq_ and _ne_ methods of inspect classes.
- bpo-24631 [https://bugs.python.org/issue?@action=redirect&bpo=24631]: Fixed regression in the timeit module with multiline setup.
- bpo-18622 [https://bugs.python.org/issue?@action=redirect&bpo=18622]: unittest.mock.mock_open().reset_mock would recurse infinitely. Patch from Nicola Palumbo and Laurent De Buyst.
- bpo-23661 [https://bugs.python.org/issue?@action=redirect&bpo=23661]: unittest.mock side_effects can now be exceptions again. This was a regression vs Python 3.4. Patch from Ignacio Rossi
- bpo-24608 [https://bugs.python.org/issue?@action=redirect&bpo=24608]: chunk.Chunk.read() now always returns bytes, not str.
- bpo-18684 [https://bugs.python.org/issue?@action=redirect&bpo=18684]: Fixed reading out of the buffer in the re module.

- bpo-24259 [https://bugs.python.org/issue?@action=redirect&bpo=24259]: tarfile now raises a ReadError if an archive is truncated inside a data segment.
- bpo-15014 [https://bugs.python.org/issue?@action=redirect&bpo=15014]: SMTP.auth() and SMTP.login() now support RFC 4954’s optional initial-response argument to the SMTP AUTH command.
- bpo-24669 [https://bugs.python.org/issue?@action=redirect&bpo=24669]: Fix inspect.getsource() for “async def” functions. Patch by Kai Groner.
- bpo-24688 [https://bugs.python.org/issue?@action=redirect&bpo=24688]: ast.get_docstring() for “async def” functions.

Build

- bpo-24603 [https://bugs.python.org/issue?@action=redirect&bpo=24603]: Update Windows builds and OS X 10.5 installer to use OpenSSL 1.0.2d.

Python 3.5.0 beta 3

Release date: 2015-07-05

Core and Builtins

- bpo-24467 [https://bugs.python.org/issue?@action=redirect&bpo=24467]: Fixed possible buffer over-read in bytearray. The bytearray object now always allocates place for trailing null byte and it’s buffer now is always null-terminated.
- Upgrade to Unicode 8.0.0.
- bpo-24345 [https://bugs.python.org/issue?@action=redirect&bpo=24345]: Add Py_tp_finalize slot for the stable ABI.
- bpo-24400 [https://bugs.python.org/issue?@action=redirect&bpo=24400]: Introduce a distinct type for **PEP 492** [https://peps.python.org/pep-0492/] coroutines; add types.CoroutineType, inspect.getcoroutinestate,

inspect.getcoroutinelocals; coroutines no longer use CO_GENERATOR flag; sys.set_coroutine_wrapper works only for “async def” coroutines; inspect.iscoroutine no longer uses collections.abc.Coroutine, it’s intended to test for pure “async def” coroutines only; add new opcode: GET_YIELD_FROM_ITER; fix generators wrapper used in types.coroutine to be instance of collections.abc.Generator; collections.abc.Awaitable and collections.abc.Coroutine can no longer be used to detect generator-based coroutines–use inspect.isawaitable instead.

- bpo-24450 [https://bugs.python.org/issue?@action=redirect&bpo=24450]: Add gi_yieldfrom to generators and cr_await to coroutines. Contributed by Benno Leslie and Yury Selivanov.
- bpo-19235 [https://bugs.python.org/issue?@action=redirect&bpo=19235]: Add new RecursionError exception. Patch by Georg Brandl.

Library

- bpo-21750 [https://bugs.python.org/issue?@action=redirect&bpo=21750]: mock_open.read_data can now be read from each instance, as it could in Python 3.3.
- bpo-24552 [https://bugs.python.org/issue?@action=redirect&bpo=24552]: Fix use after free in an error case of the _pickle module.
- bpo-24514 [https://bugs.python.org/issue?@action=redirect&bpo=24514]: tarfile now tolerates number fields consisting of only whitespace.
- bpo-19176 [https://bugs.python.org/issue?@action=redirect&bpo=19176]: Fixed doctype() related bugs in C implementation of ElementTree. A deprecation warning no longer issued by XMLParser subclass with default doctype() method. Direct call of doctype() now issues a warning. Parser’s doctype() now is not called if target’s doctype() is called. Based on patch by Martin Panter.
- bpo-20387 [https://bugs.python.org/issue?@action=redirect&bpo=20387]: Restore semantic round-trip correctness in tokenize/untokenize for tab-indented blocks.
- bpo-24456 [https://bugs.python.org/issue?

@action=redirect&bpo=24456]: Fixed possible buffer over-read in adpcm2lin() and lin2adpcm() functions of the audioop module.

- bpo-24336 [https://bugs.python.org/issue?@action=redirect&bpo=24336]: The contextmanager decorator now works with functions with keyword arguments called «func» and «self». Patch by Martin Panter.
- bpo-24522 [https://bugs.python.org/issue?@action=redirect&bpo=24522]: Fix possible integer overflow in json accelerator module.
- bpo-24489 [https://bugs.python.org/issue?@action=redirect&bpo=24489]: ensure a previously set C errno doesn't disturb cmath.polar().
- bpo-24408 [https://bugs.python.org/issue?@action=redirect&bpo=24408]: Fixed AttributeError in measure() and metrics() methods of tkinter.Font.
- bpo-14373 [https://bugs.python.org/issue?@action=redirect&bpo=14373]: C implementation of functools.lru_cache() now can be used with methods.
- bpo-24347 [https://bugs.python.org/issue?@action=redirect&bpo=24347]: Set KeyError if PyDict_GetItemWithError returns NULL.
- bpo-24348 [https://bugs.python.org/issue?@action=redirect&bpo=24348]: Drop superfluous incref/decref.
- bpo-24359 [https://bugs.python.org/issue?@action=redirect&bpo=24359]: Check for changed OrderedDict size during iteration.
- bpo-24368 [https://bugs.python.org/issue?@action=redirect&bpo=24368]: Support keyword arguments in OrderedDict methods.
- bpo-24362 [https://bugs.python.org/issue?@action=redirect&bpo=24362]: Simplify the C OrderedDict fast nodes resize logic.
- bpo-24377 [https://bugs.python.org/issue?@action=redirect&bpo=24377]: Fix a ref leak in OrderedDict.__repr__.
- bpo-24369 [https://bugs.python.org/issue?@action=redirect&bpo=24369]: Defend against key-changes during iteration.

Tests

- bpo-24373 [https://bugs.python.org/issue?@action=redirect&bpo=24373]: _testmultiphase and xxlimited now use tp_traverse and tp_finalize to avoid reference leaks encountered when combining tp_dealloc with PyType_FromSpec (see bpo-16690 [https://bugs.python.org/issue?@action=redirect&bpo=16690] for details)

Documentation

- bpo-24458 [https://bugs.python.org/issue?@action=redirect&bpo=24458]: Update documentation to cover multi-phase initialization for extension modules (PEP 489). Patch by Petr Viktorin.
- bpo-24351 [https://bugs.python.org/issue?@action=redirect&bpo=24351]: Clarify what is meant by «identifier» in the context of string.Template instances.

Build

- bpo-24432 [https://bugs.python.org/issue?@action=redirect&bpo=24432]: Update Windows builds and OS X 10.5 installer to use OpenSSL 1.0.2c.

Python 3.5.0 beta 2

Release date: 2015-05-31

Core and Builtins

- bpo-24284 [https://bugs.python.org/issue?@action=redirect&bpo=24284]: The startswith and endswith methods of the str class no longer return True when finding the empty string and the indexes are completely out of range.
- bpo-24115 [https://bugs.python.org/issue?@action=redirect&bpo=24115]: Update uses of PyObject_IsTrue(), PyObject_Not(), PyObject_IsInstance(), PyObject_RichCompareBool() and _PyDict_Contains() to check

for and handle errors correctly.

- bpo-24328 [https://bugs.python.org/issue?@action=redirect&bpo=24328]: Fix importing one character extension modules.
- bpo-11205 [https://bugs.python.org/issue?@action=redirect&bpo=11205]: In dictionary displays, evaluate the key before the value.
- bpo-24285 [https://bugs.python.org/issue?@action=redirect&bpo=24285]: Fixed regression that prevented importing extension modules from inside packages. Patch by Petr Viktorin.

Library

- bpo-23247 [https://bugs.python.org/issue?@action=redirect&bpo=23247]: Fix a crash in the StreamWriter.reset() of CJK codecs.
- bpo-24270 [https://bugs.python.org/issue?@action=redirect&bpo=24270]: Add math.isclose() and cmath.isclose() functions as per **PEP 485** [https://peps.python.org/pep-0485/]. Contributed by Chris Barker and Tal Einat.
- bpo-5633 [https://bugs.python.org/issue?@action=redirect&bpo=5633]: Fixed timeit when the statement is a string and the setup is not.
- bpo-24326 [https://bugs.python.org/issue?@action=redirect&bpo=24326]: Fixed audioop.ratecv() with non-default weightB argument. Original patch by David Moore.
- bpo-16991 [https://bugs.python.org/issue?@action=redirect&bpo=16991]: Add a C implementation of OrderedDict.
- bpo-23934 [https://bugs.python.org/issue?@action=redirect&bpo=23934]: Fix inspect.signature to fail correctly for builtin types lacking signature information. Initial patch by James Powell.

Python 3.5.0 beta 1

Release date: 2015-05-24

Core and Builtins

- bpo-24276 [https://bugs.python.org/issue?@action=redirect&bpo=24276]: Fixed optimization of property descriptor getter.
- bpo-24268 [https://bugs.python.org/issue?@action=redirect&bpo=24268]: PEP 489: Multi-phase extension module initialization. Patch by Petr Viktorin.
- bpo-23955 [https://bugs.python.org/issue?@action=redirect&bpo=23955]: Add pyvenv.cfg option to suppress registry/environment lookup for generating sys.path on Windows.
- bpo-24257 [https://bugs.python.org/issue?@action=redirect&bpo=24257]: Fixed system error in the comparison of faked types.SimpleNamespace.
- bpo-22939 [https://bugs.python.org/issue?@action=redirect&bpo=22939]: Fixed integer overflow in iterator object. Patch by Clement Rouault.
- bpo-23985 [https://bugs.python.org/issue?@action=redirect&bpo=23985]: Fix a possible buffer overrun when deleting a slice from the front of a bytearray and then appending some other bytes data.
- bpo-24102 [https://bugs.python.org/issue?@action=redirect&bpo=24102]: Fixed exception type checking in standard error handlers.
- bpo-15027 [https://bugs.python.org/issue?@action=redirect&bpo=15027]: The UTF-32 encoder is now 3x to 7x faster.
- bpo-23290 [https://bugs.python.org/issue?@action=redirect&bpo=23290]: Optimize set_merge() for cases where the target is empty. (Contributed by Serhiy Storchaka.)
- bpo-2292 [https://bugs.python.org/issue?@action=redirect&bpo=2292]: PEP 448: Additional Unpacking Generalizations.
- bpo-24096 [https://bugs.python.org/issue?@action=redirect&bpo=24096]: Make warnings.warn_explicit more robust against mutation of the warnings.filters list.
- bpo-23996 [https://bugs.python.org/issue?@action=redirect&bpo=23996]: Avoid a crash when a delegated generator raises an unnormalized StopIteration exception. Patch by Stefan Behnel.

- bpo-23910 [https://bugs.python.org/issue?@action=redirect&bpo=23910]: Optimize property() getter calls. Patch by Joe Jevnik.
- bpo-23911 [https://bugs.python.org/issue?@action=redirect&bpo=23911]: Move path-based importlib bootstrap code to a separate frozen module.
- bpo-24192 [https://bugs.python.org/issue?@action=redirect&bpo=24192]: Fix namespace package imports.
- bpo-24022 [https://bugs.python.org/issue?@action=redirect&bpo=24022]: Fix tokenizer crash when processing undecodable source code.
- bpo-9951 [https://bugs.python.org/issue?@action=redirect&bpo=9951]: Added a hex() method to bytes, bytearray, and memoryview.
- bpo-22906 [https://bugs.python.org/issue?@action=redirect&bpo=22906]: PEP 479: Change StopIteration handling inside generators.
- bpo-24017 [https://bugs.python.org/issue?@action=redirect&bpo=24017]: PEP 492: Coroutines with async and await syntax.

Library

- bpo-14373 [https://bugs.python.org/issue?@action=redirect&bpo=14373]: Added C implementation of functools.lru_cache(). Based on patches by Matt Joiner and Alexey Kachayev.
- bpo-24230 [https://bugs.python.org/issue?@action=redirect&bpo=24230]: The tempfile module now accepts bytes for prefix, suffix and dir parameters and returns bytes in such situations (matching the os module APIs).
- bpo-22189 [https://bugs.python.org/issue?@action=redirect&bpo=22189]: collections.UserString now supports __getnewargs__(), __rmod__(), casefold(), format_map(), isprintable(), and maketrans(). Patch by Joe Jevnik.
- bpo-24244 [https://bugs.python.org/issue?@action=redirect&bpo=24244]: Prevents termination when an invalid format string is encountered on Windows in strftime.
- bpo-23973 [https://bugs.python.org/issue?@action=redirect&bpo=23973]: PEP 484: Add the typing module.

- bpo-23086 [https://bugs.python.org/issue?@action=redirect&bpo=23086]: The collections.abc.Sequence() abstract base class added *start* and *stop* parameters to the index() mixin. Patch by Devin Jeanpierre.
- bpo-20035 [https://bugs.python.org/issue?@action=redirect&bpo=20035]: Replaced the `tkinter._fix` module used for setting up the Tcl/Tk environment on Windows with a private function in the `_tkinter` module that makes no permanent changes to the environment.
- bpo-24257 [https://bugs.python.org/issue?@action=redirect&bpo=24257]: Fixed segmentation fault in sqlite3.Row constructor with faked cursor type.
- bpo-15836 [https://bugs.python.org/issue?@action=redirect&bpo=15836]: assertRaises(), assertRaisesRegex(), assertWarns() and assertWarnsRegex() assertments now check the type of the first argument to prevent possible user error. Based on patch by Daniel Wagner-Hall.
- bpo-9858 [https://bugs.python.org/issue?@action=redirect&bpo=9858]: Add missing method stubs to _io.RawIOBase. Patch by Laura Rupprecht.
- bpo-22955 [https://bugs.python.org/issue?@action=redirect&bpo=22955]: attrgetter, itemgetter and methodcaller objects in the operator module now support pickling. Added readable and evaluable repr for these objects. Based on patch by Josh Rosenberg.
- bpo-22107 [https://bugs.python.org/issue?@action=redirect&bpo=22107]: tempfile.gettempdir() and tempfile.mkdtemp() now try again when a directory with the chosen name already exists on Windows as well as on Unix. tempfile.mkstemp() now fails early if parent directory is not valid (not exists or is a file) on Windows.
- bpo-23780 [https://bugs.python.org/issue?@action=redirect&bpo=23780]: Improved error message in os.path.join() with single argument.
- bpo-6598 [https://bugs.python.org/issue?@action=redirect&bpo=6598]: Increased time precision and random number range in email.utils.make_msgid() to strengthen the uniqueness of the message ID.
- bpo-24091 [https://bugs.python.org/issue?

@action=redirect&bpo=24091]: Fixed various crashes in corner cases in C implementation of ElementTree.
- bpo-21931 [https://bugs.python.org/issue?@action=redirect&bpo=21931]: msilib.FCICreate() now raises TypeError in the case of a bad argument instead of a ValueError with a bogus FCI error number. Patch by Jeffrey Armstrong.
- bpo-13866 [https://bugs.python.org/issue?@action=redirect&bpo=13866]: *quote_via* argument added to urllib.parse.urlencode.
- bpo-20098 [https://bugs.python.org/issue?@action=redirect&bpo=20098]: New mangle_from policy option for email, default True for compat32, but False for all other policies.
- bpo-24211 [https://bugs.python.org/issue?@action=redirect&bpo=24211]: The email library now supports RFC 6532: it can generate headers using utf-8 instead of encoded words.
- bpo-16314 [https://bugs.python.org/issue?@action=redirect&bpo=16314]: Added support for the LZMA compression in distutils.
- bpo-21804 [https://bugs.python.org/issue?@action=redirect&bpo=21804]: poplib now supports RFC 6856 (UTF8).
- bpo-18682 [https://bugs.python.org/issue?@action=redirect&bpo=18682]: Optimized pprint functions for builtin scalar types.
- bpo-22027 [https://bugs.python.org/issue?@action=redirect&bpo=22027]: smtplib now supports RFC 6531 (SMTPUTF8).
- bpo-23488 [https://bugs.python.org/issue?@action=redirect&bpo=23488]: Random generator objects now consume 2x less memory on 64-bit.
- bpo-1322 [https://bugs.python.org/issue?@action=redirect&bpo=1322]: platform.dist() and platform.linux_distribution() functions are now deprecated. Initial patch by Vajrasky Kok.
- bpo-22486 [https://bugs.python.org/issue?@action=redirect&bpo=22486]: Added the math.gcd() function. The fractions.gcd() function now is deprecated. Based on patch by Mark Dickinson.

- bpo-24064 [https://bugs.python.org/issue?@action=redirect&bpo=24064]: Property() docstrings are now writeable. (Patch by Berker Peksag.)
- bpo-22681 [https://bugs.python.org/issue?@action=redirect&bpo=22681]: Added support for the koi8_t encoding.
- bpo-22682 [https://bugs.python.org/issue?@action=redirect&bpo=22682]: Added support for the kz1048 encoding.
- bpo-23796 [https://bugs.python.org/issue?@action=redirect&bpo=23796]: peek and read1 methods of BufferedReader now raise ValueError if they called on a closed object. Patch by John Hergenroeder.
- bpo-21795 [https://bugs.python.org/issue?@action=redirect&bpo=21795]: smtpd now supports the 8BITMIME extension whenever the new *decode_data* constructor argument is set to False.
- bpo-24155 [https://bugs.python.org/issue?@action=redirect&bpo=24155]: optimize heapq.heapify() for better cache performance when heapifying large lists.
- bpo-21800 [https://bugs.python.org/issue?@action=redirect&bpo=21800]: imaplib now supports RFC 5161 (enable), RFC 6855 (utf8/internationalized email) and automatically encodes non-ASCII usernames and passwords to UTF8.
- bpo-20274 [https://bugs.python.org/issue?@action=redirect&bpo=20274]: When calling a _sqlite.Connection, it now complains if passed any keyword arguments. Previously it silently ignored them.
- bpo-20274 [https://bugs.python.org/issue?@action=redirect&bpo=20274]: Remove ignored and erroneous «kwargs» parameters from three METH_VARARGS methods on _sqlite.Connection.
- bpo-24134 [https://bugs.python.org/issue?@action=redirect&bpo=24134]: assertRaises(), assertRaisesRegex(), assertWarns() and assertWarnsRegex() checks now emits a deprecation warning when callable is None or keyword arguments except msg is passed in the context manager mode.
- bpo-24018 [https://bugs.python.org/issue?

@action=redirect&bpo=24018]: Add a collections.abc.Generator abstract base class. Contributed by Stefan Behnel.

- bpo-23880 [https://bugs.python.org/issue?@action=redirect&bpo=23880]: Tkinter's getint() and getdouble() now support Tcl_Obj. Tkinter's getdouble() now supports any numbers (in particular int).
- bpo-22619 [https://bugs.python.org/issue?@action=redirect&bpo=22619]: Added negative limit support in the traceback module. Based on patch by Dmitry Kazakov.
- bpo-24094 [https://bugs.python.org/issue?@action=redirect&bpo=24094]: Fix possible crash in json.encode with poorly behaved dict subclasses.
- bpo-9246 [https://bugs.python.org/issue?@action=redirect&bpo=9246]: On POSIX, os.getcwd() now supports paths longer than 1025 bytes. Patch written by William Orr.
- bpo-17445 [https://bugs.python.org/issue?@action=redirect&bpo=17445]: add difflib.diff_bytes() to support comparison of byte strings (fixes a regression from Python 2).
- bpo-23917 [https://bugs.python.org/issue?@action=redirect&bpo=23917]: Fall back to sequential compilation when ProcessPoolExecutor doesn't exist. Patch by Claudiu Popa.
- bpo-23008 [https://bugs.python.org/issue?@action=redirect&bpo=23008]: Fixed resolving attributes with boolean value is False in pydoc.
- Fix asyncio issue 235: LifoQueue and PriorityQueue's put didn't increment unfinished tasks (this bug was introduced when JoinableQueue was merged with Queue).
- bpo-23908 [https://bugs.python.org/issue?@action=redirect&bpo=23908]: os functions now reject paths with embedded null character on Windows instead of silently truncating them.
- bpo-23728 [https://bugs.python.org/issue?@action=redirect&bpo=23728]: binascii.crc_hqx() could return an integer outside of the range 0-0xffff for empty data.
- bpo-23887 [https://bugs.python.org/issue?@action=redirect&bpo=23887]: urllib.error.HTTPError now has a proper repr() representation. Patch by Berker Peksag.
- asyncio: New event loop APIs: set_task_factory() and get_task_factory().

- asyncio: async() function is deprecated in favour of ensure_future().
- bpo-24178 [https://bugs.python.org/issue?@action=redirect&bpo=24178]: asyncio.Lock, Condition, Semaphore, and BoundedSemaphore support new “async with” syntax. Contributed by Yury Selivanov.
- bpo-24179 [https://bugs.python.org/issue?@action=redirect&bpo=24179]: Support “async for” for asyncio.StreamReader. Contributed by Yury Selivanov.
- bpo-24184 [https://bugs.python.org/issue?@action=redirect&bpo=24184]: Add AsyncIterator and AsyncIterable ABCs to collections.abc. Contributed by Yury Selivanov.
- bpo-22547 [https://bugs.python.org/issue?@action=redirect&bpo=22547]: Implement informative _repr_ for inspect.BoundArguments. Contributed by Yury Selivanov.
- bpo-24190 [https://bugs.python.org/issue?@action=redirect&bpo=24190]: Implement inspect.BoundArgument.apply_defaults() method. Contributed by Yury Selivanov.
- bpo-20691 [https://bugs.python.org/issue?@action=redirect&bpo=20691]: Add “follow_wrapped” argument to inspect.Signature.from_callable() and inspect.signature(). Contributed by Yury Selivanov.
- bpo-24248 [https://bugs.python.org/issue?@action=redirect&bpo=24248]: Deprecate inspect.Signature.from_function() and inspect.Signature.from_builtin().
- bpo-23898 [https://bugs.python.org/issue?@action=redirect&bpo=23898]: Fix inspect.classify_class_attrs() to support attributes with overloaded _eq_ and _bool_. Patch by Mike Bayer.
- bpo-24298 [https://bugs.python.org/issue?@action=redirect&bpo=24298]: Fix inspect.signature() to correctly unwrap wrappers around bound methods.

IDLE

- bpo-23184 [https://bugs.python.org/issue?@action=redirect&bpo=23184]: remove unused names and imports in idlelib. Initial patch by Al Sweigart.

Tests

- bpo-21520 [https://bugs.python.org/issue?@action=redirect&bpo=21520]: test_zipfile no longer fails if the word "bad" appears anywhere in the name of the current directory.
- bpo-9517 [https://bugs.python.org/issue?@action=redirect&bpo=9517]: Move script_helper into the support package. Patch by Christie Wilson.

Documentation

- bpo-22155 [https://bugs.python.org/issue?@action=redirect&bpo=22155]: Add File Handlers subsection with createfilehandler to tkinter doc. Remove obsolete example from FAQ. Patch by Martin Panter.
- bpo-24029 [https://bugs.python.org/issue?@action=redirect&bpo=24029]: Document the name binding behavior for submodule imports.
- bpo-24077 [https://bugs.python.org/issue?@action=redirect&bpo=24077]: Fix typo in man page for -I command option: -s, not -S

Tools/Demos

- bpo-24000 [https://bugs.python.org/issue?@action=redirect&bpo=24000]: Improved Argument Clinic's mapping of converters to legacy «format units». Updated the documentation to match.
- bpo-24001 [https://bugs.python.org/issue?@action=redirect&bpo=24001]: Argument Clinic converters now use accept={type} instead of types={"type"} to specify the types the converter accepts.
- bpo-23330 [https://bugs.python.org/issue?@action=redirect&bpo=23330]: h2py now supports arbitrary filenames in #include.
- bpo-24031 [https://bugs.python.org/issue?@action=redirect&bpo=24031]: make patchcheck now supports

git checkouts, too.

Python 3.5.0 alpha 4

Release date: 2015-04-19

Core and Builtins

- bpo-22980 [https://bugs.python.org/issue?@action=redirect&bpo=22980]: Under Linux, GNU/KFreeBSD and the Hurd, C extensions now include the architecture triplet in the extension name, to make it easy to test builds for different ABIs in the same working tree. Under OS X, the extension name now includes **PEP 3149** [https://peps.python.org/pep-3149/]-style information.
- bpo-22631 [https://bugs.python.org/issue?@action=redirect&bpo=22631]: Added Linux-specific socket constant CAN_RAW_FD_FRAMES. Patch courtesy of Joe Jevnik.
- bpo-23731 [https://bugs.python.org/issue?@action=redirect&bpo=23731]: Implement **PEP 488** [https://peps.python.org/pep-0488/]: removal of .pyo files.
- bpo-23726 [https://bugs.python.org/issue?@action=redirect&bpo=23726]: Don't enable GC for user subclasses of non-GC types that don't add any new fields. Patch by Eugene Toder.
- bpo-23309 [https://bugs.python.org/issue?@action=redirect&bpo=23309]: Avoid a deadlock at shutdown if a daemon thread is aborted while it is holding a lock to a buffered I/O object, and the main thread tries to use the same I/O object (typically stdout or stderr). A fatal error is emitted instead.
- bpo-22977 [https://bugs.python.org/issue?@action=redirect&bpo=22977]: Fixed formatting Windows error messages on Wine. Patch by Martin Panter.
- bpo-23466 [https://bugs.python.org/issue?@action=redirect&bpo=23466]: %c, %o, %x, and %X in bytes formatting now raise TypeError on non-integer input.
- bpo-24044 [https://bugs.python.org/issue?

@action=redirect&bpo=24044]: Fix possible null pointer dereference in list.sort in out of memory conditions.
- bpo-21354 [https://bugs.python.org/issue?@action=redirect&bpo=21354]: PyCFunction_New function is exposed by python DLL again.

Library

- bpo-23840 [https://bugs.python.org/issue?@action=redirect&bpo=23840]: tokenize.open() now closes the temporary binary file on error to fix a resource warning.
- bpo-16914 [https://bugs.python.org/issue?@action=redirect&bpo=16914]: new debuglevel 2 in smtplib adds timestamps to debug output.
- bpo-7159 [https://bugs.python.org/issue?@action=redirect&bpo=7159]: urllib.request now supports sending auth credentials automatically after the first 401. This enhancement is a superset of the enhancement from bpo-19494 [https://bugs.python.org/issue?@action=redirect&bpo=19494] and supersedes that change.
- bpo-23703 [https://bugs.python.org/issue?@action=redirect&bpo=23703]: Fix a regression in urljoin() introduced in 901e4e52b20a. Patch by Demian Brecht.
- bpo-4254 [https://bugs.python.org/issue?@action=redirect&bpo=4254]: Adds _curses.update_lines_cols(). Patch by Arnon Yaari
- bpo-19933 [https://bugs.python.org/issue?@action=redirect&bpo=19933]: Provide default argument for ndigits in round. Patch by Vajrasky Kok.
- bpo-23193 [https://bugs.python.org/issue?@action=redirect&bpo=23193]: Add a numeric_owner parameter to tarfile.TarFile.extract and tarfile.TarFile.extractall. Patch by Michael Vogt and Eric Smith.
- bpo-23342 [https://bugs.python.org/issue?@action=redirect&bpo=23342]: Add a subprocess.run() function than returns a CalledProcess instance for a more consistent API than the existing call* functions.
- bpo-21217 [https://bugs.python.org/issue?@action=redirect&bpo=21217]: inspect.getsourcelines() now tries to compute the start and end lines from the code object, fixing an issue when a lambda function is used as decorator

argument. Patch by Thomas Ballinger and Allison Kaptur.

- bpo-24521 [https://bugs.python.org/issue?@action=redirect&bpo=24521]: Fix possible integer overflows in the pickle module.
- bpo-22931 [https://bugs.python.org/issue?@action=redirect&bpo=22931]: Allow “[” and “]” in cookie values.
- The keywords attribute of functools.partial is now always a dictionary.
- bpo-23811 [https://bugs.python.org/issue?@action=redirect&bpo=23811]: Add missing newline to the PyCompileError error message. Patch by Alex Shkop.
- bpo-21116 [https://bugs.python.org/issue?@action=redirect&bpo=21116]: Avoid blowing memory when allocating a multiprocessing shared array that’s larger than 50% of the available RAM. Patch by Médéric Boquien.
- bpo-22982 [https://bugs.python.org/issue?@action=redirect&bpo=22982]: Improve BOM handling when seeking to multiple positions of a writable text file.
- bpo-23464 [https://bugs.python.org/issue?@action=redirect&bpo=23464]: Removed deprecated asyncio JoinableQueue.
- bpo-23529 [https://bugs.python.org/issue?@action=redirect&bpo=23529]: Limit the size of decompressed data when reading from GzipFile, BZ2File or LZMAFile. This defeats denial of service attacks using compressed bombs (i.e. compressed payloads which decompress to a huge size). Patch by Martin Panter and Nikolaus Rath.
- bpo-21859 [https://bugs.python.org/issue?@action=redirect&bpo=21859]: Added Python implementation of io.FileIO.
- bpo-23865 [https://bugs.python.org/issue?@action=redirect&bpo=23865]: close() methods in multiple modules now are idempotent and more robust at shutdown. If they need to release multiple resources, all are released even if errors occur.
- bpo-23400 [https://bugs.python.org/issue?@action=redirect&bpo=23400]: Raise same exception on both Python 2 and 3 if sem_open is not available. Patch by Davin Potts.

- bpo-10838 [https://bugs.python.org/issue?@action=redirect&bpo=10838]: The subprocess now module includes SubprocessError and TimeoutError in its list of exported names for the users wild enough to use `from subprocess import *`.
- bpo-23411 [https://bugs.python.org/issue?@action=redirect&bpo=23411]: Added DefragResult, ParseResult, SplitResult, DefragResultBytes, ParseResultBytes, and SplitResultBytes to urllib.parse._all_. Patch by Martin Panter.
- bpo-23881 [https://bugs.python.org/issue?@action=redirect&bpo=23881]: urllib.request.ftpwrapper constructor now closes the socket if the FTP connection failed to fix a ResourceWarning.
- bpo-23853 [https://bugs.python.org/issue?@action=redirect&bpo=23853]: **`socket.socket.sendall()`** does no more reset the socket timeout each time data is sent successfully. The socket timeout is now the maximum total duration to send all data.
- bpo-22721 [https://bugs.python.org/issue?@action=redirect&bpo=22721]: An order of multiline pprint output of set or dict containing orderable and non-orderable elements no longer depends on iteration order of set or dict.
- bpo-15133 [https://bugs.python.org/issue?@action=redirect&bpo=15133]: _tkinter.tkapp.getboolean() now supports Tcl_Obj and always returns bool. tkinter.BooleanVar now validates input values (accepted bool, int, str, and Tcl_Obj). tkinter.BooleanVar.get() now always returns bool.
- bpo-10590 [https://bugs.python.org/issue?@action=redirect&bpo=10590]: xml.sax.parseString() now supports string argument.
- bpo-23338 [https://bugs.python.org/issue?@action=redirect&bpo=23338]: Fixed formatting ctypes error messages on Cygwin. Patch by Makoto Kato.
- bpo-15582 [https://bugs.python.org/issue?@action=redirect&bpo=15582]: inspect.getdoc() now follows inheritance chains.
- bpo-2175 [https://bugs.python.org/issue?@action=redirect&bpo=2175]: SAX parsers now support a character stream of InputSource object.
- bpo-16840 [https://bugs.python.org/issue?

@action=redirect&bpo=16840]: Tkinter now supports 64-bit integers added in Tcl 8.4 and arbitrary precision integers added in Tcl 8.5.

- bpo-23834 [https://bugs.python.org/issue?@action=redirect&bpo=23834]: Fix socket.sendto(), use the C Py_ssize_t type to store the result of sendto() instead of the C int type.
- bpo-23618 [https://bugs.python.org/issue?@action=redirect&bpo=23618]: `socket.socket.connect()` now waits until the connection completes instead of raising `InterruptedError` if the connection is interrupted by signals, signal handlers don't raise an exception and the socket is blocking or has a timeout. **`socket.socket.connect()`** still raise **`InterruptedError`** for non-blocking sockets.
- bpo-21526 [https://bugs.python.org/issue?@action=redirect&bpo=21526]: Tkinter now supports new boolean type in Tcl 8.5.
- bpo-23836 [https://bugs.python.org/issue?@action=redirect&bpo=23836]: Fix the faulthandler module to handle reentrant calls to its signal handlers.
- bpo-23838 [https://bugs.python.org/issue?@action=redirect&bpo=23838]: linecache now clears the cache and returns an empty result on MemoryError.
- bpo-10395 [https://bugs.python.org/issue?@action=redirect&bpo=10395]: Added os.path.commonpath(). Implemented in posixpath and ntpath. Based on patch by Rafik Draoui.
- bpo-23611 [https://bugs.python.org/issue?@action=redirect&bpo=23611]: Serializing more «lookupable» objects (such as unbound methods or nested classes) now are supported with pickle protocols < 4.
- bpo-13583 [https://bugs.python.org/issue?@action=redirect&bpo=13583]: sqlite3.Row now supports slice indexing.
- bpo-18473 [https://bugs.python.org/issue?@action=redirect&bpo=18473]: Fixed 2to3 and 3to2 compatible pickle mappings. Fixed ambiguous reverse mappings. Added many new mappings. Import mapping is no longer applied to modules already mapped with full name mapping.

- bpo-23485 [https://bugs.python.org/issue?@action=redirect&bpo=23485]: select.select() is now retried automatically with the recomputed timeout when interrupted by a signal, except if the signal handler raises an exception. This change is part of the **PEP 475** [https://peps.python.org/pep-0475/].
- bpo-23752 [https://bugs.python.org/issue?@action=redirect&bpo=23752]: When built from an existing file descriptor, io.FileIO() now only calls fstat() once. Before fstat() was called twice, which was not necessary.
- bpo-23704 [https://bugs.python.org/issue?@action=redirect&bpo=23704]: collections.deque() objects now support __add__, __mul__, and __imul__().
- bpo-23171 [https://bugs.python.org/issue?@action=redirect&bpo=23171]: csv.Writer.writerow() now supports arbitrary iterables.
- bpo-23745 [https://bugs.python.org/issue?@action=redirect&bpo=23745]: The new email header parser now handles duplicate MIME parameter names without error, similar to how get_param behaves.
- bpo-22117 [https://bugs.python.org/issue?@action=redirect&bpo=22117]: Fix os.utime(), it now rounds the timestamp towards minus infinity (-inf) instead of rounding towards zero.
- bpo-23310 [https://bugs.python.org/issue?@action=redirect&bpo=23310]: Fix MagicMock's initializer to work with __methods__, just like configure_mock(). Patch by Kasia Jachim.

Build

- bpo-23817 [https://bugs.python.org/issue?@action=redirect&bpo=23817]: FreeBSD now uses «1.0» in the SOVERSION as other operating systems, instead of just «1».
- bpo-23501 [https://bugs.python.org/issue?@action=redirect&bpo=23501]: Argument Clinic now generates code into separate files by default.

Tests

- bpo-23799 [https://bugs.python.org/issue?@action=redirect&bpo=23799]: Added test.support.start_threads() for running and cleaning up multiple threads.
- bpo-22390 [https://bugs.python.org/issue?@action=redirect&bpo=22390]: test.regrtest now emits a warning if temporary files or directories are left after running a test.

Tools/Demos

- bpo-18128 [https://bugs.python.org/issue?@action=redirect&bpo=18128]: pygettext now uses standard +NNNN format in the POT-Creation-Date header.
- bpo-23935 [https://bugs.python.org/issue?@action=redirect&bpo=23935]: Argument Clinic's understanding of format units accepting bytes, bytearrays, and buffers is now consistent with both the documentation and the implementation.
- bpo-23944 [https://bugs.python.org/issue?@action=redirect&bpo=23944]: Argument Clinic now wraps long impl prototypes at column 78.
- bpo-20586 [https://bugs.python.org/issue?@action=redirect&bpo=20586]: Argument Clinic now ensures that functions without docstrings have signatures.
- bpo-23492 [https://bugs.python.org/issue?@action=redirect&bpo=23492]: Argument Clinic now generates argument parsing code with PyArg_Parse instead of PyArg_ParseTuple if possible.
- bpo-23500 [https://bugs.python.org/issue?@action=redirect&bpo=23500]: Argument Clinic is now smarter about generating the «#ifndef» (empty) definition of the methoddef macro: it's only generated once, even if Argument Clinic processes the same symbol multiple times, and it's emitted at the end of all processing rather than immediately after the first use.

C API

- bpo-23998 [https://bugs.python.org/issue?@action=redirect&bpo=23998]: PyImport_ReInitLock() now checks for lock allocation error

Python 3.5.0 alpha 3

Release date: 2015-03-28

Core and Builtins

- bpo-23573 [https://bugs.python.org/issue?@action=redirect&bpo=23573]: Increased performance of string search operations (str.find, str.index, str.count, the in operator, str.split, str.partition) with arguments of different kinds (UCS1, UCS2, UCS4).
- bpo-23753 [https://bugs.python.org/issue?@action=redirect&bpo=23753]: Python doesn't support anymore platforms without stat() or fstat(), these functions are always required.
- bpo-23681 [https://bugs.python.org/issue?@action=redirect&bpo=23681]: The -b option now affects comparisons of bytes with int.
- bpo-23632 [https://bugs.python.org/issue?@action=redirect&bpo=23632]: Memoryviews now allow tuple indexing (including for multi-dimensional memoryviews).
- bpo-23192 [https://bugs.python.org/issue?@action=redirect&bpo=23192]: Fixed generator lambdas. Patch by Bruno Cauet.
- bpo-23629 [https://bugs.python.org/issue?@action=redirect&bpo=23629]: Fix the default _sizeof_ implementation for variable-sized objects.

Library

- bpo-14260 [https://bugs.python.org/issue?@action=redirect&bpo=14260]: The groupindex attribute of regular expression pattern object now is non-modifiable mapping.
- bpo-23792 [https://bugs.python.org/issue?@action=redirect&bpo=23792]: Ignore KeyboardInterrupt when the pydoc pager is active. This mimics the behavior of the standard unix pagers, and prevents pipepager from shutting down while the pager itself is still running.

- bpo-23775 [https://bugs.python.org/issue?@action=redirect&bpo=23775]: pprint() of OrderedDict now outputs the same representation as repr().
- bpo-23765 [https://bugs.python.org/issue?@action=redirect&bpo=23765]: Removed IsBadStringPtr calls in ctypes
- bpo-22364 [https://bugs.python.org/issue?@action=redirect&bpo=22364]: Improved some re error messages using regex for hints.
- bpo-23742 [https://bugs.python.org/issue?@action=redirect&bpo=23742]: ntpath.expandvars() no longer loses unbalanced single quotes.
- bpo-21717 [https://bugs.python.org/issue?@action=redirect&bpo=21717]: The zipfile.ZipFile.open function now supports “x” (exclusive creation) mode.
- bpo-21802 [https://bugs.python.org/issue?@action=redirect&bpo=21802]: The reader in BufferedRWPair now is closed even when closing writer failed in BufferedRWPair.close().
- bpo-23622 [https://bugs.python.org/issue?@action=redirect&bpo=23622]: Unknown escapes in regular expressions that consist of `'\'` and ASCII letter now raise a deprecation warning and will be forbidden in Python 3.6.
- bpo-23671 [https://bugs.python.org/issue?@action=redirect&bpo=23671]: string.Template now allows specifying the «self» parameter as a keyword argument. string.Formatter now allows specifying the «self» and the «format_string» parameters as keyword arguments.
- bpo-23502 [https://bugs.python.org/issue?@action=redirect&bpo=23502]: The pprint module now supports mapping proxies.
- bpo-17530 [https://bugs.python.org/issue?@action=redirect&bpo=17530]: pprint now wraps long bytes objects and bytearrays.
- bpo-22687 [https://bugs.python.org/issue?@action=redirect&bpo=22687]: Fixed some corner cases in breaking words in tetxtwrap. Got rid of quadratic complexity in breaking long words.
- bpo-4727 [https://bugs.python.org/issue?@action=redirect&bpo=4727]: The copy module now uses pickle protocol 4 (PEP 3154) and

supports copying of instances of classes whose _new_ method takes keyword-only arguments.

- bpo-23491 [https://bugs.python.org/issue?@action=redirect&bpo=23491]: Added a zipapp module to support creating executable zip file archives of Python code. Registered «.pyz» and «.pyzw» extensions on Windows for these archives (PEP 441).
- bpo-23657 [https://bugs.python.org/issue?@action=redirect&bpo=23657]: Avoid explicit checks for str in zipapp, adding support for pathlib.Path objects as arguments.
- bpo-23688 [https://bugs.python.org/issue?@action=redirect&bpo=23688]: Added support of arbitrary bytes-like objects and avoided unnecessary copying of memoryview in gzip.GzipFile.write(). Original patch by Wolfgang Maier.
- bpo-23252 [https://bugs.python.org/issue?@action=redirect&bpo=23252]: Added support for writing ZIP files to unseekable streams.
- bpo-23647 [https://bugs.python.org/issue?@action=redirect&bpo=23647]: Increase imaplib's MAXLINE to accommodate modern mailbox sizes.
- bpo-23539 [https://bugs.python.org/issue?@action=redirect&bpo=23539]: If body is None, http.client.HTTPConnection.request now sets Content-Length to 0 for PUT, POST, and PATCH headers to avoid 411 errors from some web servers.
- bpo-22351 [https://bugs.python.org/issue?@action=redirect&bpo=22351]: The nntplib.NNTP constructor no longer leaves the connection and socket open until the garbage collector cleans them up. Patch by Martin Panter.
- bpo-23704 [https://bugs.python.org/issue?@action=redirect&bpo=23704]: collections.deque() objects now support methods for index(), insert(), and copy(). This allows deques to be registered as a MutableSequence and it improves their substitutability for lists.
- bpo-23715 [https://bugs.python.org/issue?@action=redirect&bpo=23715]: **`signal.sigwaitinfo()`** and **`signal.sigtimedwait()`** are now retried when interrupted by a signal not in the *sigset* parameter, if the signal handler does not raise an exception. signal.sigtimedwait() recomputes the timeout with a

monotonic clock when it is retried.

- bpo-23001 [https://bugs.python.org/issue?@action=redirect&bpo=23001]: Few functions in modules mmap, ossaudiodev, socket, ssl, and codecs, that accepted only read-only bytes-like object now accept writable bytes-like object too.
- bpo-23646 [https://bugs.python.org/issue?@action=redirect&bpo=23646]: If time.sleep() is interrupted by a signal, the sleep is now retried with the recomputed delay, except if the signal handler raises an exception (PEP 475).
- bpo-23136 [https://bugs.python.org/issue?@action=redirect&bpo=23136]: _strptime now uniformly handles all days in week 0, including Dec 30 of previous year. Based on patch by Jim Carroll.
- bpo-23700 [https://bugs.python.org/issue?@action=redirect&bpo=23700]: Iterator of NamedTemporaryFile now keeps a reference to NamedTemporaryFile instance. Patch by Bohuslav Kabrda.
- bpo-22903 [https://bugs.python.org/issue?@action=redirect&bpo=22903]: The fake test case created by unittest.loader when it fails importing a test module is now picklable.
- bpo-22181 [https://bugs.python.org/issue?@action=redirect&bpo=22181]: On Linux, os.urandom() now uses the new getrandom() syscall if available, syscall introduced in the Linux kernel 3.17. It is more reliable and more secure, because it avoids the need of a file descriptor and waits until the kernel has enough entropy.
- bpo-2211 [https://bugs.python.org/issue?@action=redirect&bpo=2211]: Updated the implementation of the http.cookies.Morsel class. Setting attributes key, value and coded_value directly now is deprecated. update() and setdefault() now transform and check keys. Comparing for equality now takes into account attributes key, value and coded_value. copy() now returns a Morsel, not a dict. repr() now contains all attributes. Optimized checking keys and quoting values. Added new tests. Original patch by Demian Brecht.
- bpo-18983 [https://bugs.python.org/issue?@action=redirect&bpo=18983]: Allow selection of output units in timeit. Patch by Julian Gindi.

- bpo-23631 [https://bugs.python.org/issue?@action=redirect&bpo=23631]: Fix traceback.format_list when a traceback has been mutated.
- bpo-23568 [https://bugs.python.org/issue?@action=redirect&bpo=23568]: Add rdivmod support to MagicMock() objects. Patch by Håkan Lövdahl.
- bpo-2052 [https://bugs.python.org/issue?@action=redirect&bpo=2052]: Add charset parameter to HtmlDiff.make_file().
- bpo-23668 [https://bugs.python.org/issue?@action=redirect&bpo=23668]: Support os.truncate and os.ftruncate on Windows.
- bpo-23138 [https://bugs.python.org/issue?@action=redirect&bpo=23138]: Fixed parsing cookies with absent keys or values in cookiejar. Patch by Demian Brecht.
- bpo-23051 [https://bugs.python.org/issue?@action=redirect&bpo=23051]: multiprocessing.Pool methods imap() and imap_unordered() now handle exceptions raised by an iterator. Patch by Alon Diamant and Davin Potts.
- bpo-23581 [https://bugs.python.org/issue?@action=redirect&bpo=23581]: Add matmul support to MagicMock. Patch by Håkan Lövdahl.
- bpo-23566 [https://bugs.python.org/issue?@action=redirect&bpo=23566]: enable(), register(), dump_traceback() and dump_traceback_later() functions of faulthandler now accept file descriptors. Patch by Wei Wu.
- bpo-22928 [https://bugs.python.org/issue?@action=redirect&bpo=22928]: Disabled HTTP header injections in http.client. Original patch by Demian Brecht.
- bpo-23615 [https://bugs.python.org/issue?@action=redirect&bpo=23615]: Modules bz2, tarfile and tokenize now can be reloaded with imp.reload(). Patch by Thomas Kluyver.
- bpo-23605 [https://bugs.python.org/issue?@action=redirect&bpo=23605]: os.walk() now calls os.scandir() instead of os.listdir(). The usage of os.scandir() reduces the number of calls to os.stat(). Initial patch written by Ben Hoyt.

Build

- bpo-23585 [https://bugs.python.org/issue?

@action=redirect&bpo=23585]: make patchcheck will ensure the interpreter is built.

Tests

- bpo-23583 [https://bugs.python.org/issue?@action=redirect&bpo=23583]: Added tests for standard IO streams in IDLE.
- bpo-22289 [https://bugs.python.org/issue?@action=redirect&bpo=22289]: Prevent test_urllib2net failures due to ftp connection timeout.

Tools/Demos

- bpo-22826 [https://bugs.python.org/issue?@action=redirect&bpo=22826]: The result of open() in Tools/freeze/bkfile.py is now better compatible with regular files (in particular it now supports the context management protocol).

Python 3.5.0 alpha 2

Release date: 2015-03-09

Core and Builtins

- bpo-23571 [https://bugs.python.org/issue?@action=redirect&bpo=23571]: PyObject_Call() and PyCFunction_Call() now raise a SystemError if a function returns a result and raises an exception. The SystemError is chained to the previous exception.

Library

- bpo-22524 [https://bugs.python.org/issue?@action=redirect&bpo=22524]: New os.scandir() function, part of the **PEP 471** [https://peps.python.org/pep-0471/]: «os.scandir() function – a better and faster directory iterator». Patch written by Ben Hoyt.

- bpo-23103 [https://bugs.python.org/issue?@action=redirect&bpo=23103]: Reduced the memory consumption of IPv4Address and IPv6Address.
- bpo-21793 [https://bugs.python.org/issue?@action=redirect&bpo=21793]: BaseHTTPRequestHandler again logs response code as numeric, not as stringified enum. Patch by Demian Brecht.
- bpo-23476 [https://bugs.python.org/issue?@action=redirect&bpo=23476]: In the ssl module, enable OpenSSL's X509_V_FLAG_TRUSTED_FIRST flag on certificate stores when it is available.
- bpo-23576 [https://bugs.python.org/issue?@action=redirect&bpo=23576]: Avoid stalling in SSL reads when EOF has been reached in the SSL layer but the underlying connection hasn't been closed.
- bpo-23504 [https://bugs.python.org/issue?@action=redirect&bpo=23504]: Added an _all_ to the types module.
- bpo-23563 [https://bugs.python.org/issue?@action=redirect&bpo=23563]: Optimized utility functions in urllib.parse.
- bpo-7830 [https://bugs.python.org/issue?@action=redirect&bpo=7830]: Flatten nested functools.partial.
- bpo-20204 [https://bugs.python.org/issue?@action=redirect&bpo=20204]: Added the _module_ attribute to _tkinter classes.
- bpo-19980 [https://bugs.python.org/issue?@action=redirect&bpo=19980]: Improved help() for non-recognized strings. help("") now shows the help on str. help("help") now shows the help on help(). Original patch by Mark Lawrence.
- bpo-23521 [https://bugs.python.org/issue?@action=redirect&bpo=23521]: Corrected pure python implementation of timedelta division. Eliminated OverflowError from `timedelta * float` for some floats; Corrected rounding in timedelta true division.
- bpo-21619 [https://bugs.python.org/issue?@action=redirect&bpo=21619]: Popen objects no longer leave a zombie after exit in the with statement if the pipe was broken. Patch by Martin Panter.

- bpo-22936 [https://bugs.python.org/issue?@action=redirect&bpo=22936]: Make it possible to show local variables in tracebacks for both the traceback module and unittest.
- bpo-15955 [https://bugs.python.org/issue?@action=redirect&bpo=15955]: Add an option to limit the output size in bz2.decompress(). Patch by Nikolaus Rath.
- bpo-6639 [https://bugs.python.org/issue?@action=redirect&bpo=6639]: Module-level turtle functions no longer raise TclError after closing the window.
- bpo-814253 [https://bugs.python.org/issue?@action=redirect&bpo=814253]: Group references and conditional group references now work in lookbehind assertions in regular expressions. (See also: bpo-9179 [https://bugs.python.org/issue?@action=redirect&bpo=9179])
- bpo-23215 [https://bugs.python.org/issue?@action=redirect&bpo=23215]: Multibyte codecs with custom error handlers that ignores errors consumed too much memory and raised SystemError or MemoryError. Original patch by Aleksi Torhamo.
- bpo-5700 [https://bugs.python.org/issue?@action=redirect&bpo=5700]: io.FileIO() called flush() after closing the file. flush() was not called in close() if closefd=False.
- bpo-23374 [https://bugs.python.org/issue?@action=redirect&bpo=23374]: Fixed pydoc failure with non-ASCII files when stdout encoding differs from file system encoding (e.g. on Mac OS).
- bpo-23481 [https://bugs.python.org/issue?@action=redirect&bpo=23481]: Remove RC4 from the SSL module's default cipher list.
- bpo-21548 [https://bugs.python.org/issue?@action=redirect&bpo=21548]: Fix pydoc.synopsis() and pydoc.apropos() on modules with empty docstrings.
- bpo-22885 [https://bugs.python.org/issue?@action=redirect&bpo=22885]: Fixed arbitrary code execution vulnerability in the dbm.dumb module. Original patch by Claudiu Popa.
- bpo-23239 [https://bugs.python.org/issue?@action=redirect&bpo=23239]: ssl.match_hostname() now supports matching of IP addresses.

- bpo-23146 [https://bugs.python.org/issue?@action=redirect&bpo=23146]: Fix mishandling of absolute Windows paths with forward slashes in pathlib.
- bpo-23096 [https://bugs.python.org/issue?@action=redirect&bpo=23096]: Pickle representation of floats with protocol 0 now is the same for both Python and C implementations.
- bpo-19105 [https://bugs.python.org/issue?@action=redirect&bpo=19105]: pprint now more efficiently uses free space at the right.
- bpo-14910 [https://bugs.python.org/issue?@action=redirect&bpo=14910]: Add allow_abbrev parameter to argparse.ArgumentParser. Patch by Jonathan Paugh, Steven Bethard, paul j3 and Daniel Eriksson.
- bpo-21717 [https://bugs.python.org/issue?@action=redirect&bpo=21717]: tarfile.open() now supports “x” (exclusive creation) mode.
- bpo-23344 [https://bugs.python.org/issue?@action=redirect&bpo=23344]: marshal.dumps() is now 20-25% faster on average.
- bpo-20416 [https://bugs.python.org/issue?@action=redirect&bpo=20416]: marshal.dumps() with protocols 3 and 4 is now 40-50% faster on average.
- bpo-23421 [https://bugs.python.org/issue?@action=redirect&bpo=23421]: Fixed compression in tarfile CLI. Patch by wdv4758h.
- bpo-23367 [https://bugs.python.org/issue?@action=redirect&bpo=23367]: Fix possible overflows in the unicodedata module.
- bpo-23361 [https://bugs.python.org/issue?@action=redirect&bpo=23361]: Fix possible overflow in Windows subprocess creation code.
- logging.handlers.QueueListener now takes a respect_handler_level keyword argument which, if set to True, will pass messages to handlers taking handler levels into account.
- bpo-19705 [https://bugs.python.org/issue?@action=redirect&bpo=19705]: turtledemo now has a visual sorting algorithm demo. Original patch from Jason Yeo.
- bpo-23801 [https://bugs.python.org/issue?

@action=redirect&bpo=23801]: Fix issue where cgi.FieldStorage did not always ignore the entire preamble to a multipart body.

Build

- bpo-23445 [https://bugs.python.org/issue?@action=redirect&bpo=23445]: pydebug builds now use «gcc -Og» where possible, to make the resulting executable faster.
- bpo-23686 [https://bugs.python.org/issue?@action=redirect&bpo=23686]: Update OS X 10.5 installer build to use OpenSSL 1.0.2a.

C API

- bpo-20204 [https://bugs.python.org/issue?@action=redirect&bpo=20204]: Deprecation warning is now raised for builtin types without the _module_ attribute.

Windows

- bpo-23465 [https://bugs.python.org/issue?@action=redirect&bpo=23465]: Implement **PEP 486** [https://peps.python.org/pep-0486/] - Make the Python Launcher aware of virtual environments. Patch by Paul Moore.
- bpo-23437 [https://bugs.python.org/issue?@action=redirect&bpo=23437]: Make user scripts directory versioned on Windows. Patch by Paul Moore.

Python 3.5.0 alpha 1

Release date: 2015-02-08

Core and Builtins

- bpo-23285 [https://bugs.python.org/issue?@action=redirect&bpo=23285]: PEP 475 - EINTR handling.
- bpo-22735 [https://bugs.python.org/issue?@action=redirect&bpo=22735]: Fix many edge cases (including

crashes) involving custom mro() implementations.

- bpo-22896 [https://bugs.python.org/issue?@action=redirect&bpo=22896]: Avoid using PyObject_AsCharBuffer(), PyObject_AsReadBuffer() and PyObject_AsWriteBuffer().
- bpo-21295 [https://bugs.python.org/issue?@action=redirect&bpo=21295]: Revert some changes (bpo-16795 [https://bugs.python.org/issue?@action=redirect&bpo=16795]) to AST line numbers and column offsets that constituted a regression.
- bpo-22986 [https://bugs.python.org/issue?@action=redirect&bpo=22986]: Allow changing an object's _class_ between a dynamic type and static type in some cases.
- bpo-15859 [https://bugs.python.org/issue?@action=redirect&bpo=15859]: PyUnicode_EncodeFSDefault(), PyUnicode_EncodeMBCS() and PyUnicode_EncodeCodePage() now raise an exception if the object is not a Unicode object. For PyUnicode_EncodeFSDefault(), it was already the case on platforms other than Windows. Patch written by Campbell Barton.
- bpo-21408 [https://bugs.python.org/issue?@action=redirect&bpo=21408]: The default _ne_() now returns NotImplemented if _eq_() returned NotImplemented. Original patch by Martin Panter.
- bpo-23321 [https://bugs.python.org/issue?@action=redirect&bpo=23321]: Fixed a crash in str.decode() when error handler returned replacement string longer than malformed input data.
- bpo-22286 [https://bugs.python.org/issue?@action=redirect&bpo=22286]: The «backslashreplace» error handlers now works with decoding and translating.
- bpo-23253 [https://bugs.python.org/issue?@action=redirect&bpo=23253]: Delay-load ShellExecute[AW] in os.startfile for reduced startup overhead on Windows.
- bpo-22038 [https://bugs.python.org/issue?@action=redirect&bpo=22038]: pyatomic.h now uses stdatomic.h or GCC built-in functions for atomic memory access if available. Patch written by Vitor de Lima and Gustavo Temple.
- bpo-20284 [https://bugs.python.org/issue?

@action=redirect&bpo=20284]: %-interpolation (aka printf) formatting added for bytes and bytearray.

- bpo-23048 [https://bugs.python.org/issue?@action=redirect&bpo=23048]: Fix jumping out of an infinite while loop in the pdb.
- bpo-20335 [https://bugs.python.org/issue?@action=redirect&bpo=20335]: bytes constructor now raises TypeError when encoding or errors is specified with non-string argument. Based on patch by Renaud Blanch.
- bpo-22834 [https://bugs.python.org/issue?@action=redirect&bpo=22834]: If the current working directory ends up being set to a non-existent directory then import will no longer raise FileNotFoundError.
- bpo-22869 [https://bugs.python.org/issue?@action=redirect&bpo=22869]: Move the interpreter startup & shutdown code to a new dedicated pylifecycle.c module
- bpo-22847 [https://bugs.python.org/issue?@action=redirect&bpo=22847]: Improve method cache efficiency.
- bpo-22335 [https://bugs.python.org/issue?@action=redirect&bpo=22335]: Fix crash when trying to enlarge a bytearray to 0x7fffffff bytes on a 32-bit platform.
- bpo-22653 [https://bugs.python.org/issue?@action=redirect&bpo=22653]: Fix an assertion failure in debug mode when doing a reentrant dict insertion in debug mode.
- bpo-22643 [https://bugs.python.org/issue?@action=redirect&bpo=22643]: Fix integer overflow in Unicode case operations (upper, lower, title, swapcase, casefold).
- bpo-17636 [https://bugs.python.org/issue?@action=redirect&bpo=17636]: Circular imports involving relative imports are now supported.
- bpo-22604 [https://bugs.python.org/issue?@action=redirect&bpo=22604]: Fix assertion error in debug mode when dividing a complex number by (nan+0j).
- bpo-21052 [https://bugs.python.org/issue?@action=redirect&bpo=21052]: Do not raise ImportWarning when sys.path_hooks or sys.meta_path are set to None.
- bpo-16518 [https://bugs.python.org/issue?@action=redirect&bpo=16518]: Use “bytes-like object required” in error messages that previously used the far more cryptic «“x” does not support the buffer protocol.

- bpo-22470 [https://bugs.python.org/issue?@action=redirect&bpo=22470]: Fixed integer overflow issues in «backslashreplace», «xmlcharrefreplace», and «surrogatepass» error handlers.
- bpo-22540 [https://bugs.python.org/issue?@action=redirect&bpo=22540]: speed up `PyObject_IsInstance` and `PyObject_IsSubclass` in the common case that the second argument has metaclass `type`.
- bpo-18711 [https://bugs.python.org/issue?@action=redirect&bpo=18711]: Add a new `PyErr_FormatV` function, similar to `PyErr_Format` but accepting a `va_list` argument.
- bpo-22520 [https://bugs.python.org/issue?@action=redirect&bpo=22520]: Fix overflow checking when generating the repr of a unicode object.
- bpo-22519 [https://bugs.python.org/issue?@action=redirect&bpo=22519]: Fix overflow checking in PyBytes_Repr.
- bpo-22518 [https://bugs.python.org/issue?@action=redirect&bpo=22518]: Fix integer overflow issues in latin-1 encoding.
- bpo-16324 [https://bugs.python.org/issue?@action=redirect&bpo=16324]: _charset parameter of MIMEText now also accepts email.charset.Charset instances. Initial patch by Claude Paroz.
- bpo-1764286 [https://bugs.python.org/issue?@action=redirect&bpo=1764286]: Fix inspect.getsource() to support decorated functions. Patch by Claudiu Popa.
- bpo-18554 [https://bugs.python.org/issue?@action=redirect&bpo=18554]: os.__all__ includes posix functions.
- bpo-21391 [https://bugs.python.org/issue?@action=redirect&bpo=21391]: Use os.path.abspath in the shutil module.
- bpo-11471 [https://bugs.python.org/issue?@action=redirect&bpo=11471]: avoid generating a JUMP_FORWARD instruction at the end of an if-block if there is no else-clause. Original patch by Eugene Toder.
- bpo-22215 [https://bugs.python.org/issue?@action=redirect&bpo=22215]: Now ValueError is raised instead

of TypeError when str or bytes argument contains not permitted null character or byte.

- bpo-22258 [https://bugs.python.org/issue?@action=redirect&bpo=22258]: Fix the internal function set_inheritable() on Illumos. This platform exposes the function `ioctl(FIOCLEX)`, but calling it fails with errno is ENOTTY: «Inappropriate ioctl for device». set_inheritable() now falls back to the slower `fcntl()` (`F_GETFD` and then `F_SETFD`).
- bpo-21389 [https://bugs.python.org/issue?@action=redirect&bpo=21389]: Displaying the _qualname_ of the underlying function in the repr of a bound method.
- bpo-22206 [https://bugs.python.org/issue?@action=redirect&bpo=22206]: Using pthread, PyThread_create_key() now sets errno to ENOMEM and returns -1 (error) on integer overflow.
- bpo-20184 [https://bugs.python.org/issue?@action=redirect&bpo=20184]: Argument Clinic based signature introspection added for 30 of the builtin functions.
- bpo-22116 [https://bugs.python.org/issue?@action=redirect&bpo=22116]: C functions and methods (of the “builtin_function_or_method” type) can now be weakref’ed. Patch by Wei Wu.
- bpo-22077 [https://bugs.python.org/issue?@action=redirect&bpo=22077]: Improve index error messages for bytearrays, bytes, lists, and tuples by adding “or slices”. Added “, not <typename>” for bytearrays. Original patch by Claudiu Popa.
- bpo-20179 [https://bugs.python.org/issue?@action=redirect&bpo=20179]: Apply Argument Clinic to bytes and bytearray. Patch by Tal Einat.
- bpo-22082 [https://bugs.python.org/issue?@action=redirect&bpo=22082]: Clear interned strings in slotdefs.
- Upgrade Unicode database to Unicode 7.0.0.
- bpo-21897 [https://bugs.python.org/issue?@action=redirect&bpo=21897]: Fix a crash with the f_locals attribute with closure variables when frame.clear() has been called.
- bpo-21205 [https://bugs.python.org/issue?@action=redirect&bpo=21205]: Add a new `__qualname__`

attribute to generator, the qualified name, and use it in the representation of a generator (`repr(gen)`). The default name of the generator (`__name__` attribute) is now get from the function instead of the code. Use `gen.gi_code.co_name` to get the name of the code.

- bpo-21669 [https://bugs.python.org/issue?@action=redirect&bpo=21669]: With the aid of heuristics in SyntaxError.__init__, the parser now attempts to generate more meaningful (or at least more search engine friendly) error messages when «exec» and «print» are used as statements.
- bpo-21642 [https://bugs.python.org/issue?@action=redirect&bpo=21642]: In the conditional if-else expression, allow an integer written with no space between itself and the `else` keyword (e.g. `True if 42else False`) to be valid syntax.
- bpo-21523 [https://bugs.python.org/issue?@action=redirect&bpo=21523]: Fix over-pessimistic computation of the stack effect of some opcodes in the compiler. This also fixes a quadratic compilation time issue noticeable when compiling code with a large number of «and» and «or» operators.
- bpo-21418 [https://bugs.python.org/issue?@action=redirect&bpo=21418]: Fix a crash in the builtin function super() when called without argument and without current frame (ex: embedded Python).
- bpo-21425 [https://bugs.python.org/issue?@action=redirect&bpo=21425]: Fix flushing of standard streams in the interactive interpreter.
- bpo-21435 [https://bugs.python.org/issue?@action=redirect&bpo=21435]: In rare cases, when running finalizers on objects in cyclic trash a bad pointer dereference could occur due to a subtle flaw in internal iteration logic.
- bpo-21377 [https://bugs.python.org/issue?@action=redirect&bpo=21377]: PyBytes_Concat() now tries to concatenate in-place when the first argument has a reference count of 1. Patch by Nikolaus Rath.
- bpo-20355 [https://bugs.python.org/issue?@action=redirect&bpo=20355]: -W command line options now have higher priority than the PYTHONWARNINGS environment variable. Patch by Arfrever.

- bpo-21274 [https://bugs.python.org/issue?@action=redirect&bpo=21274]: Define PATH_MAX for GNU/Hurd in Python/pythonrun.c.
- bpo-20904 [https://bugs.python.org/issue?@action=redirect&bpo=20904]: Support setting FPU precision on m68k.
- bpo-21209 [https://bugs.python.org/issue?@action=redirect&bpo=21209]: Fix sending tuples to custom generator objects with the yield from syntax.
- bpo-21193 [https://bugs.python.org/issue?@action=redirect&bpo=21193]: pow(a, b, c) now raises ValueError rather than TypeError when b is negative. Patch by Josh Rosenberg.
- bpo-21176 [https://bugs.python.org/issue?@action=redirect&bpo=21176]: PEP 465: Add the “@” operator for matrix multiplication.
- bpo-21134 [https://bugs.python.org/issue?@action=redirect&bpo=21134]: Fix segfault when str is called on an uninitialized UnicodeEncodeError, UnicodeDecodeError, or UnicodeTranslateError object.
- bpo-19537 [https://bugs.python.org/issue?@action=redirect&bpo=19537]: Fix PyUnicode_DATA() alignment under m68k. Patch by Andreas Schwab.
- bpo-20929 [https://bugs.python.org/issue?@action=redirect&bpo=20929]: Add a type cast to avoid shifting a negative number.
- bpo-20731 [https://bugs.python.org/issue?@action=redirect&bpo=20731]: Properly position in source code files even if they are opened in text mode. Patch by Serhiy Storchaka.
- bpo-20637 [https://bugs.python.org/issue?@action=redirect&bpo=20637]: Key-sharing now also works for instance dictionaries of subclasses. Patch by Peter Ingebretson.
- bpo-8297 [https://bugs.python.org/issue?@action=redirect&bpo=8297]: Attributes missing from modules now include the module name in the error text. Original patch by ysj.ray.
- bpo-19995 [https://bugs.python.org/issue?@action=redirect&bpo=19995]: %c, %o, %x, and %X now raise TypeError on non-integer input.

- bpo-19655 [https://bugs.python.org/issue?@action=redirect&bpo=19655]: The ASDL parser - used by the build process to generate code for managing the Python AST in C - was rewritten. The new parser is self contained and does not require to carry long the spark.py parser-generator library; spark.py was removed from the source base.
- bpo-12546 [https://bugs.python.org/issue?@action=redirect&bpo=12546]: Allow `\x00` to be used as a fill character when using str, int, float, and complex _format_ methods.
- bpo-20480 [https://bugs.python.org/issue?@action=redirect&bpo=20480]: Add ipaddress.reverse_pointer. Patch by Leon Weber.
- bpo-13598 [https://bugs.python.org/issue?@action=redirect&bpo=13598]: Modify string.Formatter to support auto-numbering of replacement fields. It now matches the behavior of str.format() in this regard. Patches by Phil Elson and Ramchandra Apte.
- bpo-8931 [https://bugs.python.org/issue?@action=redirect&bpo=8931]: Make alternate formatting ("#") for type "c" raise an exception. In versions prior to 3.5, "#" with "c" had no effect. Now specifying it is an error. Patch by Torsten Landschoff.
- bpo-23165 [https://bugs.python.org/issue?@action=redirect&bpo=23165]: Perform overflow checks before allocating memory in the _Py_char2wchar function.

Library

- bpo-23399 [https://bugs.python.org/issue?@action=redirect&bpo=23399]: pyvenv creates relative symlinks where possible.
- bpo-20289 [https://bugs.python.org/issue?@action=redirect&bpo=20289]: cgi.FieldStorage() now supports the context management protocol.
- bpo-13128 [https://bugs.python.org/issue?@action=redirect&bpo=13128]: Print response headers for CONNECT requests when debuglevel > 0. Patch by Demian Brecht.
- bpo-15381 [https://bugs.python.org/issue?@action=redirect&bpo=15381]: Optimized io.BytesIO to make less

allocations and copyings.

- bpo-22818 [https://bugs.python.org/issue?@action=redirect&bpo=22818]: Splitting on a pattern that could match an empty string now raises a warning. Patterns that can only match empty strings are now rejected.
- bpo-23099 [https://bugs.python.org/issue?@action=redirect&bpo=23099]: Closing io.BytesIO with exported buffer is rejected now to prevent corrupting exported buffer.
- bpo-23326 [https://bugs.python.org/issue?@action=redirect&bpo=23326]: Removed __ne__ implementations. Since fixing default __ne__ implementation in bpo-21408 [https://bugs.python.org/issue?@action=redirect&bpo=21408] they are redundant.
- bpo-23363 [https://bugs.python.org/issue?@action=redirect&bpo=23363]: Fix possible overflow in itertools.permutations.
- bpo-23364 [https://bugs.python.org/issue?@action=redirect&bpo=23364]: Fix possible overflow in itertools.product.
- bpo-23366 [https://bugs.python.org/issue?@action=redirect&bpo=23366]: Fixed possible integer overflow in itertools.combinations.
- bpo-23369 [https://bugs.python.org/issue?@action=redirect&bpo=23369]: Fixed possible integer overflow in _json.encode_basestring_ascii.
- bpo-23353 [https://bugs.python.org/issue?@action=redirect&bpo=23353]: Fix the exception handling of generators in PyEval_EvalFrameEx(). At entry, save or swap the exception state even if PyEval_EvalFrameEx() is called with throwflag=0. At exit, the exception state is now always restored or swapped, not only if why is WHY_YIELD or WHY_RETURN. Patch co-written with Antoine Pitrou.
- bpo-14099 [https://bugs.python.org/issue?@action=redirect&bpo=14099]: Restored support of writing ZIP files to tellable but non-seekable streams.
- bpo-14099 [https://bugs.python.org/issue?@action=redirect&bpo=14099]: Writing to ZipFile and reading multiple ZipExtFiles is threadsafe now.
- bpo-19361 [https://bugs.python.org/issue?@action=redirect&bpo=19361]: JSON decoder now raises

JSONDecodeError instead of ValueError.

- bpo-18518 [https://bugs.python.org/issue?@action=redirect&bpo=18518]: timeit now rejects statements which can't be compiled outside a function or a loop (e.g. «return» or «break»).
- bpo-23094 [https://bugs.python.org/issue?@action=redirect&bpo=23094]: Fixed readline with frames in Python implementation of pickle.
- bpo-23268 [https://bugs.python.org/issue?@action=redirect&bpo=23268]: Fixed bugs in the comparison of ipaddress classes.
- bpo-21408 [https://bugs.python.org/issue?@action=redirect&bpo=21408]: Removed incorrect implementations of __ne__() which didn't returned NotImplemented if __eq__() returned NotImplemented. The default __ne__() now works correctly.
- bpo-19996 [https://bugs.python.org/issue?@action=redirect&bpo=19996]: **`email.feedparser.FeedParser`** now handles (malformed) headers with no key rather than assuming the body has started.
- bpo-20188 [https://bugs.python.org/issue?@action=redirect&bpo=20188]: Support Application-Layer Protocol Negotiation (ALPN) in the ssl module.
- bpo-23133 [https://bugs.python.org/issue?@action=redirect&bpo=23133]: Pickling of ipaddress objects now produces more compact and portable representation.
- bpo-23248 [https://bugs.python.org/issue?@action=redirect&bpo=23248]: Update ssl error codes from latest OpenSSL git master.
- bpo-23266 [https://bugs.python.org/issue?@action=redirect&bpo=23266]: Much faster implementation of ipaddress.collapse_addresses() when there are many non-consecutive addresses.
- bpo-23098 [https://bugs.python.org/issue?@action=redirect&bpo=23098]: 64-bit dev_t is now supported in the os module.
- bpo-21817 [https://bugs.python.org/issue?@action=redirect&bpo=21817]: When an exception is raised in a task submitted to a ProcessPoolExecutor, the remote

traceback is now displayed in the parent process. Patch by Claudiu Popa.

- bpo-15955 [https://bugs.python.org/issue?@action=redirect&bpo=15955]: Add an option to limit output size when decompressing LZMA data. Patch by Nikolaus Rath and Martin Panter.
- bpo-23250 [https://bugs.python.org/issue?@action=redirect&bpo=23250]: In the http.cookies module, capitalize «HttpOnly» and «Secure» as they are written in the standard.
- bpo-23063 [https://bugs.python.org/issue?@action=redirect&bpo=23063]: In the distutils" check command, fix parsing of reST with code or code-block directives.
- bpo-23209 [https://bugs.python.org/issue?@action=redirect&bpo=23209]: selectors.BaseSelector.get_key() now raises a RuntimeError if the selector is closed. And selectors.BaseSelector.close() now clears its internal reference to the selector mapping to break a reference cycle. Initial patch written by Martin Richard. (See also: bpo-23225 [https://bugs.python.org/issue?@action=redirect&bpo=23225])
- bpo-17911 [https://bugs.python.org/issue?@action=redirect&bpo=17911]: Provide a way to seed the linecache for a PEP-302 module without actually loading the code.
- bpo-17911 [https://bugs.python.org/issue?@action=redirect&bpo=17911]: Provide a new object API for traceback, including the ability to not lookup lines at all until the traceback is actually rendered, without any trace of the original objects being kept alive.
- bpo-19777 [https://bugs.python.org/issue?@action=redirect&bpo=19777]: Provide a home() classmethod on Path objects. Contributed by Victor Salgado and Mayank Tripathi.
- bpo-23206 [https://bugs.python.org/issue?@action=redirect&bpo=23206]: Make `json.dumps(..., ensure_ascii=False)` as fast as the default case of `ensure_ascii=True`. Patch by Naoki Inada.
- bpo-23185 [https://bugs.python.org/issue?@action=redirect&bpo=23185]: Add math.inf and math.nan constants.

- bpo-23186 [https://bugs.python.org/issue?@action=redirect&bpo=23186]: Add ssl.SSLObject.shared_ciphers() and ssl.SSLSocket.shared_ciphers() to fetch the client's list ciphers sent at handshake.
- bpo-23143 [https://bugs.python.org/issue?@action=redirect&bpo=23143]: Remove compatibility with OpenSSLs older than 0.9.8.
- bpo-23132 [https://bugs.python.org/issue?@action=redirect&bpo=23132]: Improve performance and introspection support of comparison methods created by functool.total_ordering.
- bpo-19776 [https://bugs.python.org/issue?@action=redirect&bpo=19776]: Add an expanduser() method on Path objects.
- bpo-23112 [https://bugs.python.org/issue?@action=redirect&bpo=23112]: Fix SimpleHTTPServer to correctly carry the query string and fragment when it redirects to add a trailing slash.
- bpo-21793 [https://bugs.python.org/issue?@action=redirect&bpo=21793]: Added http.HTTPStatus enums (i.e. HTTPStatus.OK, HTTPStatus.NOT_FOUND). Patch by Demian Brecht.
- bpo-23093 [https://bugs.python.org/issue?@action=redirect&bpo=23093]: In the io, module allow more operations to work on detached streams.
- bpo-23111 [https://bugs.python.org/issue?@action=redirect&bpo=23111]: In the ftplib, make ssl.PROTOCOL_SSLv23 the default protocol version.
- bpo-22585 [https://bugs.python.org/issue?@action=redirect&bpo=22585]: On OpenBSD 5.6 and newer, os.urandom() now calls getentropy(), instead of reading /dev/urandom, to get pseudo-random bytes.
- bpo-19104 [https://bugs.python.org/issue?@action=redirect&bpo=19104]: pprint now produces evaluable output for wrapped strings.
- bpo-23071 [https://bugs.python.org/issue?@action=redirect&bpo=23071]: Added missing names to codecs.__all__. Patch by Martin Panter.
- bpo-22783 [https://bugs.python.org/issue?

@action=redirect&bpo=22783]: Pickling now uses the NEWOBJ opcode instead of the NEWOBJ_EX opcode if possible.

- bpo-15513 [https://bugs.python.org/issue?@action=redirect&bpo=15513]: Added a _sizeof_ implementation for pickle classes.
- bpo-19858 [https://bugs.python.org/issue?@action=redirect&bpo=19858]: pickletools.optimize() now aware of the MEMOIZE opcode, can produce more compact result and no longer produces invalid output if input data contains MEMOIZE opcodes together with PUT or BINPUT opcodes.
- bpo-22095 [https://bugs.python.org/issue?@action=redirect&bpo=22095]: Fixed HTTPConnection.set_tunnel with default port. The port value in the host header was set to «None». Patch by Demian Brecht.
- bpo-23016 [https://bugs.python.org/issue?@action=redirect&bpo=23016]: A warning no longer produces an AttributeError when the program is run with pythonw.exe.
- bpo-21775 [https://bugs.python.org/issue?@action=redirect&bpo=21775]: shutil.copytree(): fix crash when copying to VFAT. An exception handler assumed that OSError objects always have a "winerror" attribute. That is not the case, so the exception handler itself raised AttributeError when run on Linux (and, presumably, any other non-Windows OS). Patch by Greg Ward.
- bpo-1218234 [https://bugs.python.org/issue?@action=redirect&bpo=1218234]: Fix inspect.getsource() to load updated source of reloaded module. Initial patch by Berker Peksag.
- bpo-21740 [https://bugs.python.org/issue?@action=redirect&bpo=21740]: Support wrapped callables in doctest. Patch by Claudiu Popa.
- bpo-23009 [https://bugs.python.org/issue?@action=redirect&bpo=23009]: Make sure selectors.EpollSelector.select() works when no FD is registered.
- bpo-22959 [https://bugs.python.org/issue?@action=redirect&bpo=22959]: In the constructor of http.client.HTTPSConnection, prefer the context's check_hostname attribute over the *check_hostname* parameter.
- bpo-22696 [https://bugs.python.org/issue?

@action=redirect&bpo=22696]: Add function **`sys.is_finalizing()`** to know about interpreter shutdown.

- bpo-16043 [https://bugs.python.org/issue?@action=redirect&bpo=16043]: Add a default limit for the amount of data xmlrpclib.gzip_decode will return. This resolves **CVE 2013-1753** [https://www.cve.org/CVERecord?id=CVE-2013-1753].
- bpo-14099 [https://bugs.python.org/issue?@action=redirect&bpo=14099]: ZipFile.open() no longer reopen the underlying file. Objects returned by ZipFile.open() can now operate independently of the ZipFile even if the ZipFile was created by passing in a file-like object as the first argument to the constructor.
- bpo-22966 [https://bugs.python.org/issue?@action=redirect&bpo=22966]: Fix _pycache_ pyc file name clobber when pyc_compile is asked to compile a source file containing multiple dots in the source file name.
- bpo-21971 [https://bugs.python.org/issue?@action=redirect&bpo=21971]: Update turtledemo doc and add module to the index.
- bpo-21032 [https://bugs.python.org/issue?@action=redirect&bpo=21032]: Fixed socket leak if HTTPConnection.getresponse() fails. Original patch by Martin Panter.
- bpo-22407 [https://bugs.python.org/issue?@action=redirect&bpo=22407]: Deprecated the use of re.LOCALE flag with str patterns or re.ASCII. It was newer worked.
- bpo-22902 [https://bugs.python.org/issue?@action=redirect&bpo=22902]: The «ip» command is now used on Linux to determine MAC address in uuid.getnode(). Patch by Bruno Cauet.
- bpo-22960 [https://bugs.python.org/issue?@action=redirect&bpo=22960]: Add a context argument to xmlrpclib.ServerProxy constructor.
- bpo-22389 [https://bugs.python.org/issue?@action=redirect&bpo=22389]: Add contextlib.redirect_stderr().
- bpo-21356 [https://bugs.python.org/issue?@action=redirect&bpo=21356]: Make ssl.RAND_egd() optional to support LibreSSL. The availability of the function is checked during the compilation. Patch written by Bernard Spil.

- bpo-22915 [https://bugs.python.org/issue?@action=redirect&bpo=22915]: SAX parser now supports files opened with file descriptor or bytes path.
- bpo-22609 [https://bugs.python.org/issue?@action=redirect&bpo=22609]: Constructors and update methods of mapping classes in the collections module now accept the self keyword argument.
- bpo-22940 [https://bugs.python.org/issue?@action=redirect&bpo=22940]: Add readline.append_history_file.
- bpo-19676 [https://bugs.python.org/issue?@action=redirect&bpo=19676]: Added the «namereplace» error handler.
- bpo-22788 [https://bugs.python.org/issue?@action=redirect&bpo=22788]: Add *context* parameter to logging.handlers.HTTPHandler.
- bpo-22921 [https://bugs.python.org/issue?@action=redirect&bpo=22921]: Allow SSLContext to take the *hostname* parameter even if OpenSSL doesn't support SNI.
- bpo-22894 [https://bugs.python.org/issue?@action=redirect&bpo=22894]: TestCase.subTest() would cause the test suite to be stopped when in failfast mode, even in the absence of failures.
- bpo-22796 [https://bugs.python.org/issue?@action=redirect&bpo=22796]: HTTP cookie parsing is now stricter, in order to protect against potential injection attacks.
- bpo-22370 [https://bugs.python.org/issue?@action=redirect&bpo=22370]: Windows detection in pathlib is now more robust.
- bpo-22841 [https://bugs.python.org/issue?@action=redirect&bpo=22841]: Reject coroutines in asyncio add_signal_handler(). Patch by Ludovic.Gasc.
- bpo-19494 [https://bugs.python.org/issue?@action=redirect&bpo=19494]: Added urllib.request.HTTPBasicPriorAuthHandler. Patch by Matej Cepl.
- bpo-22578 [https://bugs.python.org/issue?@action=redirect&bpo=22578]: Added attributes to the re.error class.
- bpo-22849 [https://bugs.python.org/issue?@action=redirect&bpo=22849]: Fix possible double free in the

io.TextIOWrapper constructor.

- bpo-12728 [https://bugs.python.org/issue?@action=redirect&bpo=12728]: Different Unicode characters having the same uppercase but different lowercase are now matched in case-insensitive regular expressions.
- bpo-22821 [https://bugs.python.org/issue?@action=redirect&bpo=22821]: Fixed fcntl() with integer argument on 64-bit big-endian platforms.
- bpo-21650 [https://bugs.python.org/issue?@action=redirect&bpo=21650]: Add an `--sort-keys` option to `json.tool` CLI.
- bpo-22824 [https://bugs.python.org/issue?@action=redirect&bpo=22824]: Updated reprlib output format for sets to use set literals. Patch contributed by Berker Peksag.
- bpo-22824 [https://bugs.python.org/issue?@action=redirect&bpo=22824]: Updated reprlib output format for arrays to display empty arrays without an unnecessary empty list. Suggested by Serhiy Storchaka.
- bpo-22406 [https://bugs.python.org/issue?@action=redirect&bpo=22406]: Fixed the uu_codec codec incorrectly ported to 3.x. Based on patch by Martin Panter.
- bpo-17293 [https://bugs.python.org/issue?@action=redirect&bpo=17293]: uuid.getnode() now determines MAC address on AIX using netstat. Based on patch by Aivars Kalvāns.
- bpo-22769 [https://bugs.python.org/issue?@action=redirect&bpo=22769]: Fixed ttk.Treeview.tag_has() when called without arguments.
- bpo-22417 [https://bugs.python.org/issue?@action=redirect&bpo=22417]: Verify certificates by default in httplib (PEP 476).
- bpo-22775 [https://bugs.python.org/issue?@action=redirect&bpo=22775]: Fixed unpickling of http.cookies.SimpleCookie with protocol 2 and above. Patch by Tim Graham.
- bpo-22776 [https://bugs.python.org/issue?@action=redirect&bpo=22776]: Brought excluded code into the scope of a try block in SysLogHandler.emit().
- bpo-22665 [https://bugs.python.org/issue?@action=redirect&bpo=22665]: Add missing get_terminal_size and

SameFileError to shutil.__all__.

- bpo-6623 [https://bugs.python.org/issue?@action=redirect&bpo=6623]: Remove deprecated Netrc class in the ftplib module. Patch by Matt Chaput.
- bpo-17381 [https://bugs.python.org/issue?@action=redirect&bpo=17381]: Fixed handling of case-insensitive ranges in regular expressions.
- bpo-22410 [https://bugs.python.org/issue?@action=redirect&bpo=22410]: Module level functions in the re module now cache compiled locale-dependent regular expressions taking into account the locale.
- bpo-22759 [https://bugs.python.org/issue?@action=redirect&bpo=22759]: Query methods on pathlib.Path() (exists(), is_dir(), etc.) now return False when the underlying stat call raises NotADirectoryError.
- bpo-8876 [https://bugs.python.org/issue?@action=redirect&bpo=8876]: distutils now falls back to copying files when hard linking doesn’t work. This allows use with special filesystems such as VirtualBox shared folders.
- bpo-22217 [https://bugs.python.org/issue?@action=redirect&bpo=22217]: Implemented reprs of classes in the zipfile module.
- bpo-22457 [https://bugs.python.org/issue?@action=redirect&bpo=22457]: Honour load_tests in the start_dir of discovery.
- bpo-18216 [https://bugs.python.org/issue?@action=redirect&bpo=18216]: gettext now raises an error when a .mo file has an unsupported major version number. Patch by Aaron Hill.
- bpo-13918 [https://bugs.python.org/issue?@action=redirect&bpo=13918]: Provide a locale.delocalize() function which can remove locale-specific number formatting from a string representing a number, without then converting it to a specific type. Patch by Cédric Krier.
- bpo-22676 [https://bugs.python.org/issue?@action=redirect&bpo=22676]: Make the pickling of global objects which don’t have a __module__ attribute less slow.
- bpo-18853 [https://bugs.python.org/issue?@action=redirect&bpo=18853]: Fixed ResourceWarning in shlex.__nain__.

- bpo-9351 [https://bugs.python.org/issue?@action=redirect&bpo=9351]: Defaults set with set_defaults on an argparse subparser are no longer ignored when also set on the parent parser.
- bpo-7559 [https://bugs.python.org/issue?@action=redirect&bpo=7559]: unittest test loading ImportErrors are reported as import errors with their import exception rather than as attribute errors after the import has already failed.
- bpo-19746 [https://bugs.python.org/issue?@action=redirect&bpo=19746]: Make it possible to examine the errors from unittest discovery without executing the test suite. The new `errors` attribute on `TestLoader` exposes these non-fatal errors encountered during discovery.
- bpo-21991 [https://bugs.python.org/issue?@action=redirect&bpo=21991]: Make email.headerregistry's header "params" attributes be read-only (MappingProxyType). Previously the dictionary was modifiable but a new one was created on each access of the attribute.
- bpo-22638 [https://bugs.python.org/issue?@action=redirect&bpo=22638]: SSLv3 is now disabled throughout the standard library. It can still be enabled by instantiating a SSLContext manually.
- bpo-22641 [https://bugs.python.org/issue?@action=redirect&bpo=22641]: In asyncio, the default SSL context for client connections is now created using ssl.create_default_context(), for stronger security.
- bpo-17401 [https://bugs.python.org/issue?@action=redirect&bpo=17401]: Include closefd in io.FileIO repr.
- bpo-21338 [https://bugs.python.org/issue?@action=redirect&bpo=21338]: Add silent mode for compileall. quiet parameters of compile_{dir, file, path} functions now have a multilevel value. Also, -q option of the CLI now have a multilevel value. Patch by Thomas Kluyver.
- bpo-20152 [https://bugs.python.org/issue?@action=redirect&bpo=20152]: Convert the array and cmath modules to Argument Clinic.
- bpo-18643 [https://bugs.python.org/issue?@action=redirect&bpo=18643]: Add socket.socketpair() on Windows.
- bpo-22435 [https://bugs.python.org/issue?

@action=redirect&bpo=22435]: Fix a file descriptor leak when socketserver bind fails.

- bpo-13096 [https://bugs.python.org/issue?@action=redirect&bpo=13096]: Fixed segfault in CTypes POINTER handling of large values.
- bpo-11694 [https://bugs.python.org/issue?@action=redirect&bpo=11694]: Raise ConversionError in xdrlib as documented. Patch by Filip Gruszczyński and Claudiu Popa.
- bpo-19380 [https://bugs.python.org/issue?@action=redirect&bpo=19380]: Optimized parsing of regular expressions.
- bpo-1519638 [https://bugs.python.org/issue?@action=redirect&bpo=1519638]: Now unmatched groups are replaced with empty strings in re.sub() and re.subn().
- bpo-18615 [https://bugs.python.org/issue?@action=redirect&bpo=18615]: sndhdr.what/whathdr now return a namedtuple.
- bpo-22462 [https://bugs.python.org/issue?@action=redirect&bpo=22462]: Fix pyexpat's creation of a dummy frame to make it appear in exception tracebacks.
- bpo-21965 [https://bugs.python.org/issue?@action=redirect&bpo=21965]: Add support for in-memory SSL to the ssl module. Patch by Geert Jansen.
- bpo-21173 [https://bugs.python.org/issue?@action=redirect&bpo=21173]: Fix len() on a WeakKeyDictionary when .clear() was called with an iterator alive.
- bpo-11866 [https://bugs.python.org/issue?@action=redirect&bpo=11866]: Eliminated race condition in the computation of names for new threads.
- bpo-21905 [https://bugs.python.org/issue?@action=redirect&bpo=21905]: Avoid RuntimeError in pickle.whichmodule() when sys.modules is mutated while iterating. Patch by Olivier Grisel.
- bpo-11271 [https://bugs.python.org/issue?@action=redirect&bpo=11271]: concurrent.futures.Executor.map() now takes a *chunksize* argument to allow batching of tasks in child processes and improve performance of ProcessPoolExecutor. Patch by Dan O'Reilly.
- bpo-21883 [https://bugs.python.org/issue?@action=redirect&bpo=21883]: os.path.join() and

os.path.relpath() now raise a TypeError with more helpful error message for unsupported or mismatched types of arguments.

- bpo-22219 [https://bugs.python.org/issue?@action=redirect&bpo=22219]: The zipfile module CLI now adds entries for directories (including empty directories) in ZIP file.
- bpo-22449 [https://bugs.python.org/issue?@action=redirect&bpo=22449]: In the ssl.SSLContext.load_default_certs, consult the environmental variables SSL_CERT_DIR and SSL_CERT_FILE on Windows.
- bpo-22508 [https://bugs.python.org/issue?@action=redirect&bpo=22508]: The email.__version__ variable has been removed; the email code is no longer shipped separately from the stdlib, and __version__ hasn't been updated in several releases.
- bpo-20076 [https://bugs.python.org/issue?@action=redirect&bpo=20076]: Added non derived UTF-8 aliases to locale aliases table.
- bpo-20079 [https://bugs.python.org/issue?@action=redirect&bpo=20079]: Added locales supported in glibc 2.18 to locale alias table.
- bpo-20218 [https://bugs.python.org/issue?@action=redirect&bpo=20218]: Added convenience methods read_text/write_text and read_bytes/ write_bytes to pathlib.Path objects.
- bpo-22396 [https://bugs.python.org/issue?@action=redirect&bpo=22396]: On 32-bit AIX platform, don't expose os.posix_fadvise() nor os.posix_fallocate() because their prototypes in system headers are wrong.
- bpo-22517 [https://bugs.python.org/issue?@action=redirect&bpo=22517]: When an io.BufferedRWPair object is deallocated, clear its weakrefs.
- bpo-22437 [https://bugs.python.org/issue?@action=redirect&bpo=22437]: Number of capturing groups in regular expression is no longer limited by 100.
- bpo-17442 [https://bugs.python.org/issue?@action=redirect&bpo=17442]: InteractiveInterpreter now displays the full chained traceback in its showtraceback method, to match the built in interactive interpreter.

- bpo-23392 [https://bugs.python.org/issue?@action=redirect&bpo=23392]: Added tests for marshal C API that works with FILE*.
- bpo-10510 [https://bugs.python.org/issue?@action=redirect&bpo=10510]: distutils register and upload methods now use HTML standards compliant CRLF line endings.
- bpo-9850 [https://bugs.python.org/issue?@action=redirect&bpo=9850]: Fixed macpath.join() for empty first component. Patch by Oleg Oshmyan.
- bpo-5309 [https://bugs.python.org/issue?@action=redirect&bpo=5309]: distutils” build and build_ext commands now accept a `-j` option to enable parallel building of extension modules.
- bpo-22448 [https://bugs.python.org/issue?@action=redirect&bpo=22448]: Improve canceled timer handles cleanup to prevent unbound memory usage. Patch by Joshua Moore-Oliva.
- bpo-22427 [https://bugs.python.org/issue?@action=redirect&bpo=22427]: TemporaryDirectory no longer attempts to clean up twice when used in the with statement in generator.
- bpo-22362 [https://bugs.python.org/issue?@action=redirect&bpo=22362]: Forbidden ambiguous octal escapes out of range 0-0o377 in regular expressions.
- bpo-20912 [https://bugs.python.org/issue?@action=redirect&bpo=20912]: Now directories added to ZIP file have correct Unix and MS-DOS directory attributes.
- bpo-21866 [https://bugs.python.org/issue?@action=redirect&bpo=21866]: ZipFile.close() no longer writes ZIP64 central directory records if allowZip64 is false.
- bpo-22278 [https://bugs.python.org/issue?@action=redirect&bpo=22278]: Fix urljoin problem with relative urls, a regression observed after changes to issue22118 were submitted.
- bpo-22415 [https://bugs.python.org/issue?@action=redirect&bpo=22415]: Fixed debugging output of the GROUPREF_EXISTS opcode in the re module. Removed trailing spaces in debugging output.
- bpo-22423 [https://bugs.python.org/issue?@action=redirect&bpo=22423]: Unhandled exception in thread no

longer causes unhandled AttributeError when sys.stderr is None.

- bpo-21332 [https://bugs.python.org/issue?@action=redirect&bpo=21332]: Ensure that `bufsize=1` in subprocess.Popen() selects line buffering, rather than block buffering. Patch by Akira Li.
- bpo-21091 [https://bugs.python.org/issue?@action=redirect&bpo=21091]: Fix API bug: email.message.EmailMessage.is_attachment is now a method.
- bpo-21079 [https://bugs.python.org/issue?@action=redirect&bpo=21079]: Fix email.message.EmailMessage.is_attachment to return the correct result when the header has parameters as well as a value.
- bpo-22247 [https://bugs.python.org/issue?@action=redirect&bpo=22247]: Add NNTPError to nntplib._all_.
- bpo-22366 [https://bugs.python.org/issue?@action=redirect&bpo=22366]: urllib.request.urlopen will accept a context object (SSLContext) as an argument which will then be used for HTTPS connection. Patch by Alex Gaynor.
- bpo-4180 [https://bugs.python.org/issue?@action=redirect&bpo=4180]: The warnings registries are now reset when the filters are modified.
- bpo-22419 [https://bugs.python.org/issue?@action=redirect&bpo=22419]: Limit the length of incoming HTTP request in wsgiref server to 65536 bytes and send a 414 error code for higher lengths. Patch contributed by Devin Cook.
- Lax cookie parsing in http.cookies could be a security issue when combined with non-standard cookie handling in some web browsers. Reported by Sergey Bobrov.
- bpo-20537 [https://bugs.python.org/issue?@action=redirect&bpo=20537]: logging methods now accept an exception instance as well as a Boolean value or exception tuple. Thanks to Yury Selivanov for the patch.
- bpo-22384 [https://bugs.python.org/issue?@action=redirect&bpo=22384]: An exception in Tkinter callback no longer crashes the program when it is run with pythonw.exe.
- bpo-22168 [https://bugs.python.org/issue?

@action=redirect&bpo=22168]: Prevent turtle AttributeError with non-default Canvas on OS X.
- bpo-21147 [https://bugs.python.org/issue?@action=redirect&bpo=21147]: sqlite3 now raises an exception if the request contains a null character instead of truncating it. Based on patch by Victor Stinner.
- bpo-13968 [https://bugs.python.org/issue?@action=redirect&bpo=13968]: The glob module now supports recursive search in subdirectories using the `**` pattern.
- bpo-21951 [https://bugs.python.org/issue?@action=redirect&bpo=21951]: Fixed a crash in Tkinter on AIX when called Tcl command with empty string or tuple argument.
- bpo-21951 [https://bugs.python.org/issue?@action=redirect&bpo=21951]: Tkinter now most likely raises MemoryError instead of crash if the memory allocation fails.
- bpo-22338 [https://bugs.python.org/issue?@action=redirect&bpo=22338]: Fix a crash in the json module on memory allocation failure.
- bpo-12410 [https://bugs.python.org/issue?@action=redirect&bpo=12410]: imaplib.IMAP4 now supports the context management protocol. Original patch by Tarek Ziadé.
- bpo-21270 [https://bugs.python.org/issue?@action=redirect&bpo=21270]: We now override tuple methods in mock.call objects so that they can be used as normal call attributes.
- bpo-16662 [https://bugs.python.org/issue?@action=redirect&bpo=16662]: `load_tests()` is now unconditionally run when it is present in a package's `__init__.py`. `TestLoader.loadTestsFromModule()` still accepts use_load_tests, but it is deprecated and ignored. A new keyword-only attribute `pattern` is added and documented. Patch given by Robert Collins, tweaked by Barry Warsaw.
- bpo-22226 [https://bugs.python.org/issue?@action=redirect&bpo=22226]: First letter no longer is stripped from the «status» key in the result of Treeview.heading().
- bpo-19524 [https://bugs.python.org/issue?@action=redirect&bpo=19524]: Fixed resource leak in the HTTP connection when an invalid response is received. Patch by

Martin Panter.

- bpo-20421 [https://bugs.python.org/issue?@action=redirect&bpo=20421]: Add a .version() method to SSL sockets exposing the actual protocol version in use.
- bpo-19546 [https://bugs.python.org/issue?@action=redirect&bpo=19546]: configparser exceptions no longer expose implementation details. Chained KeyErrors are removed, which leads to cleaner tracebacks. Patch by Claudiu Popa.
- bpo-22051 [https://bugs.python.org/issue?@action=redirect&bpo=22051]: turtledemo no longer reloads examples to re-run them. Initialization of variables and gui setup should be done in main(), which is called each time a demo is run, but not on import.
- bpo-21933 [https://bugs.python.org/issue?@action=redirect&bpo=21933]: Turtledemo users can change the code font size with a menu selection or control(command) "-" or "+" or control-mousewheel. Original patch by Lita Cho.
- bpo-21597 [https://bugs.python.org/issue?@action=redirect&bpo=21597]: The separator between the turtledemo text pane and the drawing canvas can now be grabbed and dragged with a mouse. The code text pane can be widened to easily view or copy the full width of the text. The canvas can be widened on small screens. Original patches by Jan Kanis and Lita Cho.
- bpo-18132 [https://bugs.python.org/issue?@action=redirect&bpo=18132]: Turtledemo buttons no longer disappear when the window is shrunk. Original patches by Jan Kanis and Lita Cho.
- bpo-22043 [https://bugs.python.org/issue?@action=redirect&bpo=22043]: time.monotonic() is now always available. `threading.Lock.acquire()`, `threading.RLock.acquire()` and socket operations now use a monotonic clock, instead of the system clock, when a timeout is used.
- bpo-21527 [https://bugs.python.org/issue?@action=redirect&bpo=21527]: Add a default number of workers to ThreadPoolExecutor equal to 5 times the number of CPUs. Patch by Claudiu Popa.
- bpo-22216 [https://bugs.python.org/issue?

@action=redirect&bpo=22216]: smtplib now resets its state more completely after a quit. The most obvious consequence of the previous behavior was a STARTTLS failure during a connect/starttls/quit/connect/starttls sequence.

- bpo-22098 [https://bugs.python.org/issue?@action=redirect&bpo=22098]: ctypes” BigEndianStructure and LittleEndianStructure now define an empty _slots_ so that subclasses don’t always get an instance dict. Patch by Claudiu Popa.
- bpo-22185 [https://bugs.python.org/issue?@action=redirect&bpo=22185]: Fix an occasional RuntimeError in threading.Condition.wait() caused by mutation of the waiters queue without holding the lock. Patch by Doug Zongker.
- bpo-22287 [https://bugs.python.org/issue?@action=redirect&bpo=22287]: On UNIX, _PyTime_gettimeofday() now uses clock_gettime(CLOCK_REALTIME) if available. As a side effect, Python now depends on the librt library on Solaris and on Linux (only with glibc older than 2.17).
- bpo-22182 [https://bugs.python.org/issue?@action=redirect&bpo=22182]: Use e.args to unpack exceptions correctly in distutils.file_util.move_file. Patch by Claudiu Popa.
- The webbrowser module now uses subprocess’s start_new_session=True rather than a potentially risky preexec_fn=os.setsid call.
- bpo-22042 [https://bugs.python.org/issue?@action=redirect&bpo=22042]: signal.set_wakeup_fd(fd) now raises an exception if the file descriptor is in blocking mode.
- bpo-16808 [https://bugs.python.org/issue?@action=redirect&bpo=16808]: inspect.stack() now returns a named tuple instead of a tuple. Patch by Daniel Shahaf.
- bpo-22236 [https://bugs.python.org/issue?@action=redirect&bpo=22236]: Fixed Tkinter images copying operations in NoDefaultRoot mode.
- bpo-2527 [https://bugs.python.org/issue?@action=redirect&bpo=2527]: Add a *globals* argument to timeit functions, in order to override the globals namespace in which the timed code is executed. Patch by Ben Roberts.
- bpo-22118 [https://bugs.python.org/issue?@action=redirect&bpo=22118]: Switch urllib.parse to use RFC

3986 semantics for the resolution of relative URLs, rather than RFCs 1808 and 2396. Patch by Demian Brecht.

- bpo-21549 [https://bugs.python.org/issue?@action=redirect&bpo=21549]: Added the «members» parameter to TarFile.list().
- bpo-19628 [https://bugs.python.org/issue?@action=redirect&bpo=19628]: Allow compileall recursion depth to be specified with a -r option.
- bpo-15696 [https://bugs.python.org/issue?@action=redirect&bpo=15696]: Add a __sizeof__ implementation for mmap objects on Windows.
- bpo-22068 [https://bugs.python.org/issue?@action=redirect&bpo=22068]: Avoided reference loops with Variables and Fonts in Tkinter.
- bpo-22165 [https://bugs.python.org/issue?@action=redirect&bpo=22165]: SimpleHTTPRequestHandler now supports undecodable file names.
- bpo-15381 [https://bugs.python.org/issue?@action=redirect&bpo=15381]: Optimized line reading in io.BytesIO.
- bpo-8797 [https://bugs.python.org/issue?@action=redirect&bpo=8797]: Raise HTTPError on failed Basic Authentication immediately. Initial patch by Sam Bull.
- bpo-20729 [https://bugs.python.org/issue?@action=redirect&bpo=20729]: Restored the use of lazy iterkeys()/itervalues()/iteritems() in the mailbox module.
- bpo-21448 [https://bugs.python.org/issue?@action=redirect&bpo=21448]: Changed FeedParser feed() to avoid $O(n^2)$ behavior when parsing long line. Original patch by Raymond Hettinger.
- bpo-22184 [https://bugs.python.org/issue?@action=redirect&bpo=22184]: The functools LRU Cache decorator factory now gives an earlier and clearer error message when the user forgets the required parameters.
- bpo-17923 [https://bugs.python.org/issue?@action=redirect&bpo=17923]: glob() patterns ending with a slash no longer match non-dirs on AIX. Based on patch by Delhallt.
- bpo-21725 [https://bugs.python.org/issue?@action=redirect&bpo=21725]: Added support for RFC 6531

(SMTPUTF8) in smtpd.

- bpo-22176 [https://bugs.python.org/issue?@action=redirect&bpo=22176]: Update the ctypes module's libffi to v3.1. This release adds support for the Linux AArch64 and POWERPC ELF ABIv2 little endian architectures.
- bpo-5411 [https://bugs.python.org/issue?@action=redirect&bpo=5411]: Added support for the «xztar» format in the shutil module.
- bpo-21121 [https://bugs.python.org/issue?@action=redirect&bpo=21121]: Don't force 3rd party C extensions to be built with -Werror=declaration-after-statement.
- bpo-21975 [https://bugs.python.org/issue?@action=redirect&bpo=21975]: Fixed crash when using uninitialized sqlite3.Row (in particular when unpickling pickled sqlite3.Row). sqlite3.Row is now initialized in the __new__() method.
- bpo-20170 [https://bugs.python.org/issue?@action=redirect&bpo=20170]: Convert posixmodule to use Argument Clinic.
- bpo-21539 [https://bugs.python.org/issue?@action=redirect&bpo=21539]: Add an *exists_ok* argument to `Pathlib.mkdir()` to mimic `mkdir -p` and `os.makedirs()` functionality. When true, ignore `FileExistsErrors`. Patch by Berker Peksag.
- bpo-22127 [https://bugs.python.org/issue?@action=redirect&bpo=22127]: Bypass IDNA for pure-ASCII host names in the socket module (in particular for numeric IPs).
- bpo-21047 [https://bugs.python.org/issue?@action=redirect&bpo=21047]: set the default value for the *convert_charrefs* argument of HTMLParser to True. Patch by Berker Peksag.
- Add an __all__ to html.entities.
- bpo-15114 [https://bugs.python.org/issue?@action=redirect&bpo=15114]: the strict mode and argument of HTMLParser, HTMLParser.error, and the HTMLParserError exception have been removed.
- bpo-22085 [https://bugs.python.org/issue?@action=redirect&bpo=22085]: Dropped support of Tk 8.3 in Tkinter.
- bpo-21580 [https://bugs.python.org/issue?@action=redirect&bpo=21580]: Now Tkinter correctly handles

bytes arguments passed to Tk. In particular this allows initializing images from binary data.

- bpo-22003 [https://bugs.python.org/issue?@action=redirect&bpo=22003]: When initialized from a bytes object, io.BytesIO() now defers making a copy until it is mutated, improving performance and memory use on some use cases. Patch by David Wilson.
- bpo-22018 [https://bugs.python.org/issue?@action=redirect&bpo=22018]: On Windows, signal.set_wakeup_fd() now also supports sockets. A side effect is that Python depends to the WinSock library.
- bpo-22054 [https://bugs.python.org/issue?@action=redirect&bpo=22054]: Add os.get_blocking() and os.set_blocking() functions to get and set the blocking mode of a file descriptor (False if the O_NONBLOCK flag is set, True otherwise). These functions are not available on Windows.
- bpo-17172 [https://bugs.python.org/issue?@action=redirect&bpo=17172]: Make turtledemo start as active on OS X even when run with subprocess. Patch by Lita Cho.
- bpo-21704 [https://bugs.python.org/issue?@action=redirect&bpo=21704]: Fix build error for _multiprocessing when semaphores are not available. Patch by Arfrever Frehtes Taifersar Arahesis.
- bpo-20173 [https://bugs.python.org/issue?@action=redirect&bpo=20173]: Convert sha1, sha256, sha512 and md5 to ArgumentClinic. Patch by Vajrasky Kok.
- Fix repr(_socket.socket) on Windows 64-bit: don't fail with OverflowError on closed socket. repr(socket.socket) already works fine.
- bpo-22033 [https://bugs.python.org/issue?@action=redirect&bpo=22033]: Reprs of most Python implemented classes now contain actual class name instead of hardcoded one.
- bpo-21947 [https://bugs.python.org/issue?@action=redirect&bpo=21947]: The dis module can now disassemble generator-iterator objects based on their gi_code attribute. Patch by Clement Rouault.
- bpo-16133 [https://bugs.python.org/issue?@action=redirect&bpo=16133]: The asynchat.async_chat.handle_read() method now ignores

BlockingIOError exceptions.

- bpo-22044 [https://bugs.python.org/issue?@action=redirect&bpo=22044]: Fixed premature DECREF in call_tzinfo_method. Patch by Tom Flanagan.
- bpo-19884 [https://bugs.python.org/issue?@action=redirect&bpo=19884]: readline: Disable the meta modifier key if stdout is not a terminal to not write the ANSI sequence `"\033[1034h"` into stdout. This sequence is used on some terminal (ex: TERM=xterm-256color») to enable support of 8 bit characters.
- bpo-4350 [https://bugs.python.org/issue?@action=redirect&bpo=4350]: Removed a number of out-of-dated and non-working for a long time Tkinter methods.
- bpo-6167 [https://bugs.python.org/issue?@action=redirect&bpo=6167]: Scrollbar.activate() now returns the name of active element if the argument is not specified. Scrollbar.set() now always accepts only 2 arguments.
- bpo-15275 [https://bugs.python.org/issue?@action=redirect&bpo=15275]: Clean up and speed up the ntpath module.
- bpo-21888 [https://bugs.python.org/issue?@action=redirect&bpo=21888]: plistlib's load() and loads() now work if the fmt parameter is specified.
- bpo-22032 [https://bugs.python.org/issue?@action=redirect&bpo=22032]: _qualname_ instead of _name_ is now always used to format fully qualified class names of Python implemented classes.
- bpo-22031 [https://bugs.python.org/issue?@action=redirect&bpo=22031]: Reprs now always use hexadecimal format with the «0x» prefix when contain an id in form « at 0x…».
- bpo-22018 [https://bugs.python.org/issue?@action=redirect&bpo=22018]: signal.set_wakeup_fd() now raises an OSError instead of a ValueError on `fstat()` failure.
- bpo-21044 [https://bugs.python.org/issue?@action=redirect&bpo=21044]: tarfile.open() now handles fileobj with an integer "name" attribute. Based on patch by Antoine Pietri.
- bpo-21966 [https://bugs.python.org/issue?@action=redirect&bpo=21966]: Respect -q command-line option

when code module is ran.

- bpo-19076 [https://bugs.python.org/issue?@action=redirect&bpo=19076]: Don’t pass the redundant “file” argument to self.error().
- bpo-16382 [https://bugs.python.org/issue?@action=redirect&bpo=16382]: Improve exception message of warnings.warn() for bad category. Initial patch by Phil Elson.
- bpo-21932 [https://bugs.python.org/issue?@action=redirect&bpo=21932]: os.read() now uses a **`Py_ssize_t()`** type instead of int for the size to support reading more than 2 GB at once. On Windows, the size is truncated to INT_MAX. As any call to os.read(), the OS may read less bytes than the number of requested bytes.
- bpo-21942 [https://bugs.python.org/issue?@action=redirect&bpo=21942]: Fixed source file viewing in pydoc’s server mode on Windows.
- bpo-11259 [https://bugs.python.org/issue?@action=redirect&bpo=11259]: asynchat.async_chat().set_terminator() now raises a ValueError if the number of received bytes is negative.
- bpo-12523 [https://bugs.python.org/issue?@action=redirect&bpo=12523]: asynchat.async_chat.push() now raises a TypeError if it doesn’t get a bytes string
- bpo-21707 [https://bugs.python.org/issue?@action=redirect&bpo=21707]: Add missing kwonlyargcount argument to ModuleFinder.replace_paths_in_code().
- bpo-20639 [https://bugs.python.org/issue?@action=redirect&bpo=20639]: calling Path.with_suffix(“”) allows removing the suffix again. Patch by July Tikhonov.
- bpo-21714 [https://bugs.python.org/issue?@action=redirect&bpo=21714]: Disallow the construction of invalid paths using Path.with_name(). Original patch by Antony Lee.
- bpo-15014 [https://bugs.python.org/issue?@action=redirect&bpo=15014]: Added “auth” method to smtplib to make implementing auth mechanisms simpler, and used it internally in the login method.
- bpo-21151 [https://bugs.python.org/issue?@action=redirect&bpo=21151]: Fixed a segfault in the winreg module when `None` is passed as a `REG_BINARY` value to

SetValueEx. Patch by John Ehresman.

- bpo-21090 [https://bugs.python.org/issue?@action=redirect&bpo=21090]: io.FileIO.readall() does not ignore I/O errors anymore. Before, it ignored I/O errors if at least the first C call read() succeed.
- bpo-5800 [https://bugs.python.org/issue?@action=redirect&bpo=5800]: headers parameter of wsgiref.headers.Headers is now optional. Initial patch by Pablo Torres Navarrete and SilentGhost.
- bpo-21781 [https://bugs.python.org/issue?@action=redirect&bpo=21781]: ssl.RAND_add() now supports strings longer than 2 GB.
- bpo-21679 [https://bugs.python.org/issue?@action=redirect&bpo=21679]: Prevent extraneous fstat() calls during open(). Patch by Bohuslav Kabrda.
- bpo-21863 [https://bugs.python.org/issue?@action=redirect&bpo=21863]: cProfile now displays the module name of C extension functions, in addition to their own name.
- bpo-11453 [https://bugs.python.org/issue?@action=redirect&bpo=11453]: asyncore: emit a ResourceWarning when an unclosed file_wrapper object is destroyed. The destructor now closes the file if needed. The close() method can now be called twice: the second call does nothing.
- bpo-21858 [https://bugs.python.org/issue?@action=redirect&bpo=21858]: Better handling of Python exceptions in the sqlite3 module.
- bpo-21476 [https://bugs.python.org/issue?@action=redirect&bpo=21476]: Make sure the email.parser.BytesParser TextIOWrapper is discarded after parsing, so the input file isn't unexpectedly closed.
- bpo-20295 [https://bugs.python.org/issue?@action=redirect&bpo=20295]: imghdr now recognizes OpenEXR format images.
- bpo-21729 [https://bugs.python.org/issue?@action=redirect&bpo=21729]: Used the «with» statement in the dbm.dumb module to ensure files closing. Patch by Claudiu Popa.
- bpo-21491 [https://bugs.python.org/issue?@action=redirect&bpo=21491]: socketserver: Fix a race condition

in child processes reaping.

- bpo-21719 [https://bugs.python.org/issue?@action=redirect&bpo=21719]: Added the `st_file_attributes` field to os.stat_result on Windows.
- bpo-21832 [https://bugs.python.org/issue?@action=redirect&bpo=21832]: Require named tuple inputs to be exact strings.
- bpo-21722 [https://bugs.python.org/issue?@action=redirect&bpo=21722]: The distutils «upload» command now exits with a non-zero return code when uploading fails. Patch by Martin Dengler.
- bpo-21723 [https://bugs.python.org/issue?@action=redirect&bpo=21723]: asyncio.Queue: support any type of number (ex: float) for the maximum size. Patch written by Vajrasky Kok.
- bpo-21711 [https://bugs.python.org/issue?@action=redirect&bpo=21711]: support for «site-python» directories has now been removed from the site module (it was deprecated in 3.4).
- bpo-17552 [https://bugs.python.org/issue?@action=redirect&bpo=17552]: new socket.sendfile() method allowing a file to be sent over a socket by using high-performance os.sendfile() on UNIX. Patch by Giampaolo Rodola".
- bpo-18039 [https://bugs.python.org/issue?@action=redirect&bpo=18039]: dbm.dump.open() now always creates a new database when the flag has the value "n". Patch by Claudiu Popa.
- bpo-21326 [https://bugs.python.org/issue?@action=redirect&bpo=21326]: Add a new is_closed() method to asyncio.BaseEventLoop. run_forever() and run_until_complete() methods of asyncio.BaseEventLoop now raise an exception if the event loop was closed.
- bpo-21766 [https://bugs.python.org/issue?@action=redirect&bpo=21766]: Prevent a security hole in CGIHTTPServer by URL unquoting paths before checking for a CGI script at that path.
- bpo-21310 [https://bugs.python.org/issue?@action=redirect&bpo=21310]: Fixed possible resource leak in failed open().

- bpo-21256 [https://bugs.python.org/issue?@action=redirect&bpo=21256]: Printout of keyword args should be in deterministic order in a mock function call. This will help to write better doctests.
- bpo-21677 [https://bugs.python.org/issue?@action=redirect&bpo=21677]: Fixed chaining nonnormalized exceptions in io close() methods.
- bpo-11709 [https://bugs.python.org/issue?@action=redirect&bpo=11709]: Fix the pydoc.help function to not fail when sys.stdin is not a valid file.
- bpo-21515 [https://bugs.python.org/issue?@action=redirect&bpo=21515]: tempfile.TemporaryFile now uses os.O_TMPFILE flag is available.
- bpo-13223 [https://bugs.python.org/issue?@action=redirect&bpo=13223]: Fix pydoc.writedoc so that the HTML documentation for methods that use “self” in the example code is generated correctly.
- bpo-21463 [https://bugs.python.org/issue?@action=redirect&bpo=21463]: In urllib.request, fix pruning of the FTP cache.
- bpo-21618 [https://bugs.python.org/issue?@action=redirect&bpo=21618]: The subprocess module could fail to close open fds that were inherited by the calling process and already higher than POSIX resource limits would otherwise allow. On systems with a functioning /proc/self/fd or /dev/fd interface the max is now ignored and all fds are closed.
- bpo-20383 [https://bugs.python.org/issue?@action=redirect&bpo=20383]: Introduce importlib.util.module_from_spec() as the preferred way to create a new module.
- bpo-21552 [https://bugs.python.org/issue?@action=redirect&bpo=21552]: Fixed possible integer overflow of too long string lengths in the tkinter module on 64-bit platforms.
- bpo-14315 [https://bugs.python.org/issue?@action=redirect&bpo=14315]: The zipfile module now ignores extra fields in the central directory that are too short to be parsed instead of letting a struct.unpack error bubble up as this «bad data» appears in many real world zip files in the

wild and is ignored by other zip tools.

- bpo-13742 [https://bugs.python.org/issue?@action=redirect&bpo=13742]: Added «key» and «reverse» parameters to heapq.merge(). (First draft of patch contributed by Simon Sapin.)
- bpo-21402 [https://bugs.python.org/issue?@action=redirect&bpo=21402]: tkinter.ttk now works when default root window is not set.
- bpo-3015 [https://bugs.python.org/issue?@action=redirect&bpo=3015]: `_tkinter.create()` now creates `tkapp` object with `wantobjects=1` by default.
- bpo-10203 [https://bugs.python.org/issue?@action=redirect&bpo=10203]: sqlite3.Row now truly supports sequence protocol. In particular it supports reverse() and negative indices. Original patch by Claudiu Popa.
- bpo-18807 [https://bugs.python.org/issue?@action=redirect&bpo=18807]: If copying (no symlinks) specified for a venv, then the python interpreter aliases (python, python3) are now created by copying rather than symlinking.
- bpo-20197 [https://bugs.python.org/issue?@action=redirect&bpo=20197]: Added support for the WebP image type in the imghdr module. Patch by Fabrice Aneche and Claudiu Popa.
- bpo-21513 [https://bugs.python.org/issue?@action=redirect&bpo=21513]: Speedup some properties of IP addresses (IPv4Address, IPv6Address) such as .is_private or .is_multicast.
- bpo-21137 [https://bugs.python.org/issue?@action=redirect&bpo=21137]: Improve the repr for threading.Lock() and its variants by showing the «locked» or «unlocked» status. Patch by Berker Peksag.
- bpo-21538 [https://bugs.python.org/issue?@action=redirect&bpo=21538]: The plistlib module now supports loading of binary plist files when reference or offset size is not a power of two.
- bpo-21455 [https://bugs.python.org/issue?@action=redirect&bpo=21455]: Add a default backlog to socket.listen().
- bpo-21525 [https://bugs.python.org/issue?@action=redirect&bpo=21525]: Most Tkinter methods which

accepted tuples now accept lists too.

- bpo-22166 [https://bugs.python.org/issue?@action=redirect&bpo=22166]: With the assistance of a new internal _codecs._forget_codec helping function, test_codecs now clears the encoding caches to avoid the appearance of a reference leak
- bpo-22236 [https://bugs.python.org/issue?@action=redirect&bpo=22236]: Tkinter tests now don't reuse default root window. New root window is created for every test class.
- bpo-10744 [https://bugs.python.org/issue?@action=redirect&bpo=10744]: Fix **PEP 3118** [https://peps.python.org/pep-3118/] format strings on ctypes objects with a nontrivial shape.
- bpo-20826 [https://bugs.python.org/issue?@action=redirect&bpo=20826]: Optimize ipaddress.collapse_addresses().
- bpo-21487 [https://bugs.python.org/issue?@action=redirect&bpo=21487]: Optimize ipaddress.summarize_address_range() and ipaddress.{IPv4Network,IPv6Network}.subnets().
- bpo-21486 [https://bugs.python.org/issue?@action=redirect&bpo=21486]: Optimize parsing of netmasks in ipaddress.IPv4Network and ipaddress.IPv6Network.
- bpo-13916 [https://bugs.python.org/issue?@action=redirect&bpo=13916]: Disallowed the surrogatepass error handler for non UTF-* encodings.
- bpo-20998 [https://bugs.python.org/issue?@action=redirect&bpo=20998]: Fixed re.fullmatch() of repeated single character pattern with ignore case. Original patch by Matthew Barnett.
- bpo-21075 [https://bugs.python.org/issue?@action=redirect&bpo=21075]: fileinput.FileInput now reads bytes from standard stream if binary mode is specified. Patch by Sam Kimbrel.
- bpo-19775 [https://bugs.python.org/issue?@action=redirect&bpo=19775]: Add a samefile() method to pathlib Path objects. Initial patch by Vajrasky Kok.
- bpo-21226 [https://bugs.python.org/issue?@action=redirect&bpo=21226]: Set up modules properly in

PyImport_ExecCodeModuleObject (and friends).

- bpo-21398 [https://bugs.python.org/issue?@action=redirect&bpo=21398]: Fix a unicode error in the pydoc pager when the documentation contains characters not encodable to the stdout encoding.
- bpo-16531 [https://bugs.python.org/issue?@action=redirect&bpo=16531]: ipaddress.IPv4Network and ipaddress.IPv6Network now accept an (address, netmask) tuple argument, so as to easily construct network objects from existing addresses.
- bpo-21156 [https://bugs.python.org/issue?@action=redirect&bpo=21156]: importlib.abc.InspectLoader.source_to_code() is now a staticmethod.
- bpo-21424 [https://bugs.python.org/issue?@action=redirect&bpo=21424]: Simplified and optimized heaqp.nlargest() and nmsmallest() to make fewer tuple comparisons.
- bpo-21396 [https://bugs.python.org/issue?@action=redirect&bpo=21396]: Fix TextIOWrapper(…, write_through=True) to not force a flush() on the underlying binary stream. Patch by akira.
- bpo-18314 [https://bugs.python.org/issue?@action=redirect&bpo=18314]: Unlink now removes junctions on Windows. Patch by Kim Gräsman
- bpo-21088 [https://bugs.python.org/issue?@action=redirect&bpo=21088]: Bugfix for curses.window.addch() regression in 3.4.0. In porting to Argument Clinic, the first two arguments were reversed.
- bpo-21407 [https://bugs.python.org/issue?@action=redirect&bpo=21407]: _decimal: The module now supports function signatures.
- bpo-10650 [https://bugs.python.org/issue?@action=redirect&bpo=10650]: Remove the non-standard “watchexp” parameter from the Decimal.quantize() method in the Python version. It had never been present in the C version.
- bpo-21469 [https://bugs.python.org/issue?@action=redirect&bpo=21469]: Reduced the risk of false positives in robotparser by checking to make sure that robots.txt has

been read or does not exist prior to returning True in can_fetch().

- bpo-19414 [https://bugs.python.org/issue?@action=redirect&bpo=19414]: Have the OrderedDict mark deleted links as unusable. This gives an early failure if the link is deleted during iteration.
- bpo-21421 [https://bugs.python.org/issue?@action=redirect&bpo=21421]: Add _slots_ to the MappingViews ABC. Patch by Josh Rosenberg.
- bpo-21101 [https://bugs.python.org/issue?@action=redirect&bpo=21101]: Eliminate double hashing in the C speed-up code for collections.Counter().
- bpo-21321 [https://bugs.python.org/issue?@action=redirect&bpo=21321]: itertools.islice() now releases the reference to the source iterator when the slice is exhausted. Patch by Anton Afanasyev.
- bpo-21057 [https://bugs.python.org/issue?@action=redirect&bpo=21057]: TextIOWrapper now allows the underlying binary stream's read() or read1() method to return an arbitrary bytes-like object (such as a memoryview). Patch by Nikolaus Rath.
- bpo-20951 [https://bugs.python.org/issue?@action=redirect&bpo=20951]: SSLSocket.send() now raises either SSLWantReadError or SSLWantWriteError on a non-blocking socket if the operation would block. Previously, it would return 0. Patch by Nikolaus Rath.
- bpo-13248 [https://bugs.python.org/issue?@action=redirect&bpo=13248]: removed previously deprecated asyncore.dispatcher _getattr_ cheap inheritance hack.
- bpo-9815 [https://bugs.python.org/issue?@action=redirect&bpo=9815]: assertRaises now tries to clear references to local variables in the exception's traceback.
- bpo-19940 [https://bugs.python.org/issue?@action=redirect&bpo=19940]: ssl.cert_time_to_seconds() now interprets the given time string in the UTC timezone (as specified in RFC 5280), not the local timezone.
- bpo-13204 [https://bugs.python.org/issue?@action=redirect&bpo=13204]: Calling sys.flags._new_ would crash the interpreter, now it raises a TypeError.
- bpo-19385 [https://bugs.python.org/issue?

@action=redirect&bpo=19385]: Make operations on a closed dbm.dumb database always raise the same exception.

- bpo-21207 [https://bugs.python.org/issue?@action=redirect&bpo=21207]: Detect when the os.urandom cached fd has been closed or replaced, and open it anew.
- bpo-21291 [https://bugs.python.org/issue?@action=redirect&bpo=21291]: subprocess's Popen.wait() is now thread safe so that multiple threads may be calling wait() or poll() on a Popen instance at the same time without losing the Popen.returncode value.
- bpo-21127 [https://bugs.python.org/issue?@action=redirect&bpo=21127]: Path objects can now be instantiated from str subclass instances (such as `numpy.str_`).
- bpo-15002 [https://bugs.python.org/issue?@action=redirect&bpo=15002]: urllib.response object to use _TemporaryFileWrapper (and _TemporaryFileCloser) facility. Provides a better way to handle file descriptor close. Patch contributed by Christian Theune.
- bpo-12220 [https://bugs.python.org/issue?@action=redirect&bpo=12220]: mindom now raises a custom ValueError indicating it doesn't support spaces in URIs instead of letting a "split" ValueError bubble up.
- bpo-21068 [https://bugs.python.org/issue?@action=redirect&bpo=21068]: The ssl.PROTOCOL* constants are now enum members.
- bpo-21276 [https://bugs.python.org/issue?@action=redirect&bpo=21276]: posixmodule: Don't define USE_XATTRS on KFreeBSD and the Hurd.
- bpo-21262 [https://bugs.python.org/issue?@action=redirect&bpo=21262]: New method assert_not_called for Mock. It raises AssertionError if the mock has been called.
- bpo-21238 [https://bugs.python.org/issue?@action=redirect&bpo=21238]: New keyword argument `unsafe` to Mock. It raises `AttributeError` in case of an attribute startswith assert or assret.
- bpo-20896 [https://bugs.python.org/issue?@action=redirect&bpo=20896]: ssl.get_server_certificate() now uses PROTOCOL_SSLv23, not PROTOCOL_SSLv3, for maximum compatibility.

- bpo-21239 [https://bugs.python.org/issue?@action=redirect&bpo=21239]: patch.stopall() didn't work deterministically when the same name was patched more than once.
- bpo-21203 [https://bugs.python.org/issue?@action=redirect&bpo=21203]: Updated fileConfig and dictConfig to remove inconsistencies. Thanks to Jure Koren for the patch.
- bpo-21222 [https://bugs.python.org/issue?@action=redirect&bpo=21222]: Passing name keyword argument to mock.create_autospec now works.
- bpo-21197 [https://bugs.python.org/issue?@action=redirect&bpo=21197]: Add lib64 -> lib symlink in venvs on 64-bit non-OS X POSIX.
- bpo-17498 [https://bugs.python.org/issue?@action=redirect&bpo=17498]: Some SMTP servers disconnect after certain errors, violating strict RFC conformance. Instead of losing the error code when we issue the subsequent RSET, smtplib now returns the error code and defers raising the SMTPServerDisconnected error until the next command is issued.
- bpo-17826 [https://bugs.python.org/issue?@action=redirect&bpo=17826]: setting an iterable side_effect on a mock function created by create_autospec now works. Patch by Kushal Das.
- bpo-7776 [https://bugs.python.org/issue?@action=redirect&bpo=7776]: Fix `Host:` header and reconnection when using http.client.HTTPConnection.set_tunnel(). Patch by Nikolaus Rath.
- bpo-20968 [https://bugs.python.org/issue?@action=redirect&bpo=20968]: unittest.mock.MagicMock now supports division. Patch by Johannes Baiter.
- bpo-21529 [https://bugs.python.org/issue?@action=redirect&bpo=21529]: Fix arbitrary memory access in JSONDecoder.raw_decode with a negative second parameter. Bug reported by Guido Vranken. (See also: **CVE 2014-4616** [https://www.cve.org/CVERecord?id=CVE-2014-4616])
- bpo-21169 [https://bugs.python.org/issue?@action=redirect&bpo=21169]: getpass now handles non-ascii characters that the input stream encoding cannot encode by

re-encoding using the replace error handler.

- bpo-21171 [https://bugs.python.org/issue?@action=redirect&bpo=21171]: Fixed undocumented filter API of the rot13 codec. Patch by Berker Peksag.
- bpo-20539 [https://bugs.python.org/issue?@action=redirect&bpo=20539]: Improved math.factorial error message for large positive inputs and changed exception type (OverflowError -> ValueError) for large negative inputs.
- bpo-21172 [https://bugs.python.org/issue?@action=redirect&bpo=21172]: isinstance check relaxed from dict to collections.Mapping.
- bpo-21155 [https://bugs.python.org/issue?@action=redirect&bpo=21155]: asyncio.EventLoop.create_unix_server() now raises a ValueError if path and sock are specified at the same time.
- bpo-21136 [https://bugs.python.org/issue?@action=redirect&bpo=21136]: Avoid unnecessary normalization of Fractions resulting from power and other operations. Patch by Raymond Hettinger.
- bpo-17621 [https://bugs.python.org/issue?@action=redirect&bpo=17621]: Introduce importlib.util.LazyLoader.
- bpo-21076 [https://bugs.python.org/issue?@action=redirect&bpo=21076]: signal module constants were turned into enums. Patch by Giampaolo Rodola”.
- bpo-20636 [https://bugs.python.org/issue?@action=redirect&bpo=20636]: Improved the repr of Tkinter widgets.
- bpo-19505 [https://bugs.python.org/issue?@action=redirect&bpo=19505]: The items, keys, and values views of OrderedDict now support reverse iteration using reversed().
- bpo-21149 [https://bugs.python.org/issue?@action=redirect&bpo=21149]: Improved thread-safety in logging cleanup during interpreter shutdown. Thanks to Devin Jeanpierre for the patch.
- bpo-21058 [https://bugs.python.org/issue?@action=redirect&bpo=21058]: Fix a leak of file descriptor in `tempfile.NamedTemporaryFile()`, close the file descriptor if `io.open()` fails

- bpo-21200 [https://bugs.python.org/issue?@action=redirect&bpo=21200]: Return None from pkgutil.get_loader() when _spec_ is missing.
- bpo-21013 [https://bugs.python.org/issue?@action=redirect&bpo=21013]: Enhance ssl.create_default_context() when used for server side sockets to provide better security by default.
- bpo-20145 [https://bugs.python.org/issue?@action=redirect&bpo=20145]: `assertRaisesRegex` and `assertWarnsRegex` now raise a `TypeError` if the second argument is not a string or compiled regex.
- bpo-20633 [https://bugs.python.org/issue?@action=redirect&bpo=20633]: Replace relative import by absolute import.
- bpo-20980 [https://bugs.python.org/issue?@action=redirect&bpo=20980]: Stop wrapping exception when using ThreadPool.
- bpo-21082 [https://bugs.python.org/issue?@action=redirect&bpo=21082]: In os.makedirs, do not set the process-wide umask. Note this changes behavior of makedirs when exist_ok=True.
- bpo-20990 [https://bugs.python.org/issue?@action=redirect&bpo=20990]: Fix issues found by pyflakes for multiprocessing.
- bpo-21015 [https://bugs.python.org/issue?@action=redirect&bpo=21015]: SSL contexts will now automatically select an elliptic curve for ECDH key exchange on OpenSSL 1.0.2 and later, and otherwise default to «prime256v1».
- bpo-21000 [https://bugs.python.org/issue?@action=redirect&bpo=21000]: Improve the command-line interface of json.tool.
- bpo-20995 [https://bugs.python.org/issue?@action=redirect&bpo=20995]: Enhance default ciphers used by the ssl module to enable better security and prioritize perfect forward secrecy.
- bpo-20884 [https://bugs.python.org/issue?@action=redirect&bpo=20884]: Don’t assume that _file_ is defined on importlib._init_.
- bpo-21499 [https://bugs.python.org/issue?

@action=redirect&bpo=21499]: Ignore _builtins_ in several test_importlib.test_api tests.
- bpo-20627 [https://bugs.python.org/issue?@action=redirect&bpo=20627]: xmlrpc.client.ServerProxy is now a context manager.
- bpo-19165 [https://bugs.python.org/issue?@action=redirect&bpo=19165]: The formatter module now raises DeprecationWarning instead of PendingDeprecationWarning.
- bpo-13936 [https://bugs.python.org/issue?@action=redirect&bpo=13936]: Remove the ability of datetime.time instances to be considered false in boolean contexts.
- bpo-18931 [https://bugs.python.org/issue?@action=redirect&bpo=18931]: selectors module now supports /dev/poll on Solaris. Patch by Giampaolo Rodola".
- bpo-19977 [https://bugs.python.org/issue?@action=redirect&bpo=19977]: When the `LC_TYPE` locale is the POSIX locale (`C` locale), **`sys.stdin`** and **`sys.stdout`** are now using the `surrogateescape` error handler, instead of the `strict` error handler.
- bpo-20574 [https://bugs.python.org/issue?@action=redirect&bpo=20574]: Implement incremental decoder for cp65001 code (Windows code page 65001, Microsoft UTF-8).
- bpo-20879 [https://bugs.python.org/issue?@action=redirect&bpo=20879]: Delay the initialization of encoding and decoding tables for base32, ascii85 and base85 codecs in the base64 module, and delay the initialization of the unquote_to_bytes() table of the urllib.parse module, to not waste memory if these modules are not used.
- bpo-19157 [https://bugs.python.org/issue?@action=redirect&bpo=19157]: Include the broadcast address in the usable hosts for IPv6 in ipaddress.
- bpo-11599 [https://bugs.python.org/issue?@action=redirect&bpo=11599]: When an external command (e.g. compiler) fails, distutils now prints out the whole command line (instead of just the command name) if the environment variable DISTUTILS_DEBUG is set.
- bpo-4931 [https://bugs.python.org/issue?@action=redirect&bpo=4931]: distutils should not produce unhelpful «error: None» messages

anymore. distutils.util.grok_environment_error is kept but doc-deprecated.

- bpo-20875 [https://bugs.python.org/issue?@action=redirect&bpo=20875]: Prevent possible gzip «“read” is not defined» NameError. Patch by Claudiu Popa.
- bpo-11558 [https://bugs.python.org/issue?@action=redirect&bpo=11558]: `email.message.Message.attach` now returns a more useful error message if `attach` is called on a message for which `is_multipart` is False.
- bpo-20283 [https://bugs.python.org/issue?@action=redirect&bpo=20283]: RE pattern methods now accept the string keyword parameters as documented. The pattern and source keyword parameters are left as deprecated aliases.
- bpo-20778 [https://bugs.python.org/issue?@action=redirect&bpo=20778]: Fix modulefinder to work with bytecode-only modules.
- bpo-20791 [https://bugs.python.org/issue?@action=redirect&bpo=20791]: copy.copy() now doesn’t make a copy when the input is a bytes object. Initial patch by Peter Otten.
- bpo-19748 [https://bugs.python.org/issue?@action=redirect&bpo=19748]: On AIX, time.mktime() now raises an OverflowError for year outsize range [1902; 2037].
- bpo-19573 [https://bugs.python.org/issue?@action=redirect&bpo=19573]: inspect.signature: Use enum for parameter kind constants.
- bpo-20726 [https://bugs.python.org/issue?@action=redirect&bpo=20726]: inspect.signature: Make Signature and Parameter picklable.
- bpo-17373 [https://bugs.python.org/issue?@action=redirect&bpo=17373]: Add inspect.Signature.from_callable method.
- bpo-20378 [https://bugs.python.org/issue?@action=redirect&bpo=20378]: Improve repr of inspect.Signature and inspect.Parameter.
- bpo-20816 [https://bugs.python.org/issue?@action=redirect&bpo=20816]: Fix inspect.getcallargs() to raise correct TypeError for missing keyword-only arguments. Patch by Jeremiah Lowin.

- bpo-20817 [https://bugs.python.org/issue?@action=redirect&bpo=20817]: Fix inspect.getcallargs() to fail correctly if more than 3 arguments are missing. Patch by Jeremiah Lowin.
- bpo-6676 [https://bugs.python.org/issue?@action=redirect&bpo=6676]: Ensure a meaningful exception is raised when attempting to parse more than one XML document per pyexpat xmlparser instance. (Original patches by Hirokazu Yamamoto and Amaury Forgeot d'Arc, with suggested wording by David Gutteridge)
- bpo-21117 [https://bugs.python.org/issue?@action=redirect&bpo=21117]: Fix inspect.signature to better support functools.partial. Due to the specifics of functools.partial implementation, positional-or-keyword arguments passed as keyword arguments become keyword-only.
- bpo-20334 [https://bugs.python.org/issue?@action=redirect&bpo=20334]: inspect.Signature and inspect.Parameter are now hashable. Thanks to Antony Lee for bug reports and suggestions.
- bpo-15916 [https://bugs.python.org/issue?@action=redirect&bpo=15916]: doctest.DocTestSuite returns an empty unittest.TestSuite instead of raising ValueError if it finds no tests
- bpo-21209 [https://bugs.python.org/issue?@action=redirect&bpo=21209]: Fix asyncio.tasks.CoroWrapper to workaround a bug in yield-from implementation in CPythons prior to 3.4.1.
- asyncio: Add gi_{frame,running,code} properties to CoroWrapper (upstream bpo-163 [https://bugs.python.org/issue?@action=redirect&bpo=163]).
- bpo-21311 [https://bugs.python.org/issue?@action=redirect&bpo=21311]: Avoid exception in _osx_support with non-standard compiler configurations. Patch by John Szakmeister.
- bpo-11571 [https://bugs.python.org/issue?@action=redirect&bpo=11571]: Ensure that the turtle window becomes the topmost window when launched on OS X.
- bpo-21801 [https://bugs.python.org/issue?@action=redirect&bpo=21801]: Validate that __signature__ is None

or an instance of Signature.

- bpo-21923 [https://bugs.python.org/issue?@action=redirect&bpo=21923]: Prevent AttributeError in distutils.sysconfig.customize_compiler due to possible uninitialized _config_vars.
- bpo-21323 [https://bugs.python.org/issue?@action=redirect&bpo=21323]: Fix http.server to again handle scripts in CGI subdirectories, broken by the fix for security bpo-19435 [https://bugs.python.org/issue?@action=redirect&bpo=19435]. Patch by Zach Byrne.
- bpo-22733 [https://bugs.python.org/issue?@action=redirect&bpo=22733]: Fix ffi_prep_args not zero-extending argument values correctly on 64-bit Windows.
- bpo-23302 [https://bugs.python.org/issue?@action=redirect&bpo=23302]: Default to TCP_NODELAY=1 upon establishing an HTTPConnection. Removed use of hard-coded MSS as it's an optimization that's no longer needed with Nagle disabled.

IDLE

- bpo-20577 [https://bugs.python.org/issue?@action=redirect&bpo=20577]: Configuration of the max line length for the FormatParagraph extension has been moved from the General tab of the Idle preferences dialog to the FormatParagraph tab of the Config Extensions dialog. Patch by Tal Einat.
- bpo-16893 [https://bugs.python.org/issue?@action=redirect&bpo=16893]: Update Idle doc chapter to match current Idle and add new information.
- bpo-3068 [https://bugs.python.org/issue?@action=redirect&bpo=3068]: Add Idle extension configuration dialog to Options menu. Changes are written to HOME/.idlerc/config-extensions.cfg. Original patch by Tal Einat.
- bpo-16233 [https://bugs.python.org/issue?@action=redirect&bpo=16233]: A module browser (File : Class Browser, Alt+C) requires an editor window with a filename. When Class Browser is requested otherwise, from a shell, output window, or “Untitled” editor, Idle no longer displays an error box. It now pops up an Open Module box (Alt+M).

If a valid name is entered and a module is opened, a corresponding browser is also opened.

- bpo-4832 [https://bugs.python.org/issue?@action=redirect&bpo=4832]: Save As to type Python files automatically adds .py to the name you enter (even if your system does not display it). Some systems automatically add .txt when type is Text files.
- bpo-21986 [https://bugs.python.org/issue?@action=redirect&bpo=21986]: Code objects are not normally pickled by the pickle module. To match this, they are no longer pickled when running under Idle.
- bpo-17390 [https://bugs.python.org/issue?@action=redirect&bpo=17390]: Adjust Editor window title; remove “Python”, move version to end.
- bpo-14105 [https://bugs.python.org/issue?@action=redirect&bpo=14105]: Idle debugger breakpoints no longer disappear when inserting or deleting lines.
- bpo-17172 [https://bugs.python.org/issue?@action=redirect&bpo=17172]: Turtledemo can now be run from Idle. Currently, the entry is on the Help menu, but it may move to Run. Patch by Ramchandra Apt and Lita Cho.
- bpo-21765 [https://bugs.python.org/issue?@action=redirect&bpo=21765]: Add support for non-ascii identifiers to HyperParser.
- bpo-21940 [https://bugs.python.org/issue?@action=redirect&bpo=21940]: Add unittest for WidgetRedirector. Initial patch by Saimadhav Heblikar.
- bpo-18592 [https://bugs.python.org/issue?@action=redirect&bpo=18592]: Add unittest for SearchDialogBase. Patch by Phil Webster.
- bpo-21694 [https://bugs.python.org/issue?@action=redirect&bpo=21694]: Add unittest for ParenMatch. Patch by Saimadhav Heblikar.
- bpo-21686 [https://bugs.python.org/issue?@action=redirect&bpo=21686]: add unittest for HyperParser. Original patch by Saimadhav Heblikar.
- bpo-12387 [https://bugs.python.org/issue?@action=redirect&bpo=12387]: Add missing upper(lower)case versions of default Windows key bindings for Idle so Caps Lock does not disable them. Patch by Roger Serwy.
- bpo-21695 [https://bugs.python.org/issue?

@action=redirect&bpo=21695]: Closing a Find-in-files output window while the search is still in progress no longer closes Idle.
- bpo-18910 [https://bugs.python.org/issue?@action=redirect&bpo=18910]: Add unittest for textView. Patch by Phil Webster.
- bpo-18292 [https://bugs.python.org/issue?@action=redirect&bpo=18292]: Add unittest for AutoExpand. Patch by Saihadhav Heblikar.
- bpo-18409 [https://bugs.python.org/issue?@action=redirect&bpo=18409]: Add unittest for AutoComplete. Patch by Phil Webster.
- bpo-21477 [https://bugs.python.org/issue?@action=redirect&bpo=21477]: htest.py - Improve framework, complete set of tests. Patches by Saimadhav Heblikar
- bpo-18104 [https://bugs.python.org/issue?@action=redirect&bpo=18104]: Add idlelib/idle_test/htest.py with a few sample tests to begin consolidating and improving human-validated tests of Idle. Change other files as needed to work with htest. Running the module as _main_ runs all tests.
- bpo-21139 [https://bugs.python.org/issue?@action=redirect&bpo=21139]: Change default paragraph width to 72, the **PEP 8** [https://peps.python.org/pep-0008/] recommendation.
- bpo-21284 [https://bugs.python.org/issue?@action=redirect&bpo=21284]: Paragraph reformat test passes after user changes reformat width.
- bpo-17654 [https://bugs.python.org/issue?@action=redirect&bpo=17654]: Ensure IDLE menus are customized properly on OS X for non-framework builds and for all variants of Tk.
- bpo-23180 [https://bugs.python.org/issue?@action=redirect&bpo=23180]: Rename IDLE «Windows» menu item to «Window». Patch by Al Sweigart.

Build

- bpo-15506 [https://bugs.python.org/issue?@action=redirect&bpo=15506]: Use standard

PKG_PROG_PKG_CONFIG autoconf macro in the configure script.

- bpo-22935 [https://bugs.python.org/issue?@action=redirect&bpo=22935]: Allow the ssl module to be compiled if openssl doesn't support SSL 3.
- bpo-22592 [https://bugs.python.org/issue?@action=redirect&bpo=22592]: Drop support of the Borland C compiler to build Python. The distutils module still supports it to build extensions.
- bpo-22591 [https://bugs.python.org/issue?@action=redirect&bpo=22591]: Drop support of MS-DOS, especially of the DJGPP compiler (MS-DOS port of GCC).
- bpo-16537 [https://bugs.python.org/issue?@action=redirect&bpo=16537]: Check whether self.extensions is empty in setup.py. Patch by Jonathan Hosmer.
- bpo-22359 [https://bugs.python.org/issue?@action=redirect&bpo=22359]: Remove incorrect uses of recursive make. Patch by Jonas Wagner.
- bpo-21958 [https://bugs.python.org/issue?@action=redirect&bpo=21958]: Define HAVE_ROUND when building with Visual Studio 2013 and above. Patch by Zachary Turner.
- bpo-18093 [https://bugs.python.org/issue?@action=redirect&bpo=18093]: the programs that embed the CPython runtime are now in a separate «Programs» directory, rather than being kept in the Modules directory.
- bpo-15759 [https://bugs.python.org/issue?@action=redirect&bpo=15759]: «make suspicious», «make linkcheck» and «make doctest» in Doc/ now display special message when and only when there are failures.
- bpo-21141 [https://bugs.python.org/issue?@action=redirect&bpo=21141]: The Windows build process no longer attempts to find Perl, instead relying on OpenSSL source being configured and ready to build. The `PCbuild\build_ssl.py` script has been re-written and re-named to `PCbuild\prepare_ssl.py`, and takes care of configuring OpenSSL source for both 32 and 64 bit platforms. OpenSSL sources obtained from svn.python.org will always be pre-configured and ready to build.
- bpo-21037 [https://bugs.python.org/issue?

@action=redirect&bpo=21037]: Add a build option to enable AddressSanitizer support.
- bpo-19962 [https://bugs.python.org/issue?@action=redirect&bpo=19962]: The Windows build process now creates «python.bat» in the root of the source tree, which passes all arguments through to the most recently built interpreter.
- bpo-21285 [https://bugs.python.org/issue?@action=redirect&bpo=21285]: Refactor and fix curses configure check to always search in a ncursesw directory.
- bpo-15234 [https://bugs.python.org/issue?@action=redirect&bpo=15234]: For BerkeleyDB and Sqlite, only add the found library and include directories if they aren't already being searched. This avoids an explicit runtime library dependency.
- bpo-17861 [https://bugs.python.org/issue?@action=redirect&bpo=17861]: Tools/scripts/generate_opcode_h.py automatically regenerates Include/opcode.h from Lib/opcode.py if the latter gets any change.
- bpo-20644 [https://bugs.python.org/issue?@action=redirect&bpo=20644]: OS X installer build support for documentation build changes in 3.4.1: assume externally supplied sphinx-build is available in /usr/bin.
- bpo-20022 [https://bugs.python.org/issue?@action=redirect&bpo=20022]: Eliminate use of deprecated bundlebuilder in OS X builds.
- bpo-15968 [https://bugs.python.org/issue?@action=redirect&bpo=15968]: Incorporated Tcl, Tk, and Tix builds into the Windows build solution.
- bpo-17095 [https://bugs.python.org/issue?@action=redirect&bpo=17095]: Fix Modules/Setup *shared* support.
- bpo-21811 [https://bugs.python.org/issue?@action=redirect&bpo=21811]: Anticipated fixes to support OS X versions > 10.9.
- bpo-21166 [https://bugs.python.org/issue?@action=redirect&bpo=21166]: Prevent possible segfaults and other random failures of python –generate-posix-vars in pybuilddir.txt build target.
- bpo-18096 [https://bugs.python.org/issue?

@action=redirect&bpo=18096]: Fix library order returned by python-config.
- bpo-17219 [https://bugs.python.org/issue?@action=redirect&bpo=17219]: Add library build dir for Python extension cross-builds.
- bpo-22919 [https://bugs.python.org/issue?@action=redirect&bpo=22919]: Windows build updated to support VC 14.0 (Visual Studio 2015), which will be used for the official release.
- bpo-21236 [https://bugs.python.org/issue?@action=redirect&bpo=21236]: Build _msi.pyd with cabinet.lib instead of fci.lib
- bpo-17128 [https://bugs.python.org/issue?@action=redirect&bpo=17128]: Use private version of OpenSSL for OS X 10.5+ installer.

C API

- bpo-14203 [https://bugs.python.org/issue?@action=redirect&bpo=14203]: Remove obsolete support for view==NULL in PyBuffer_FillInfo(), bytearray_getbuffer(), bytesiobuf_getbuffer() and array_buffer_getbuf(). All functions now raise BufferError in that case.
- bpo-22445 [https://bugs.python.org/issue?@action=redirect&bpo=22445]: PyBuffer_IsContiguous() now implements precise contiguity tests, compatible with NumPy's NPY_RELAXED_STRIDES_CHECKING compilation flag. Previously the function reported false negatives for corner cases.
- bpo-22079 [https://bugs.python.org/issue?@action=redirect&bpo=22079]: PyType_Ready() now checks that statically allocated type has no dynamically allocated bases.
- bpo-22453 [https://bugs.python.org/issue?@action=redirect&bpo=22453]: Removed non-documented macro PyObject_REPR().
- bpo-18395 [https://bugs.python.org/issue?@action=redirect&bpo=18395]: Rename `_Py_char2wchar()` to **`Py_DecodeLocale()`**, rename `_Py_wchar2char()` to **`Py_EncodeLocale()`**, and document these functions.
- bpo-21233 [https://bugs.python.org/issue?

@action=redirect&bpo=21233]: Add new C functions: PyMem_RawCalloc(), PyMem_Calloc(), PyObject_Calloc(), _PyObject_GC_Calloc(). bytes(int) is now using `calloc()` instead of `malloc()` for large objects which is faster and use less memory.

- bpo-20942 [https://bugs.python.org/issue?@action=redirect&bpo=20942]: PyImport_ImportFrozenModuleObject() no longer sets _file_ to match what importlib does; this affects _frozen_importlib as well as any module loaded using imp.init_frozen().

Documentation

- bpo-19548 [https://bugs.python.org/issue?@action=redirect&bpo=19548]: Update the codecs module documentation to better cover the distinction between text encodings and other codecs, together with other clarifications. Patch by Martin Panter.
- bpo-22394 [https://bugs.python.org/issue?@action=redirect&bpo=22394]: Doc/Makefile now supports `make venv PYTHON=../python` to create a venv for generating the documentation, e.g., `make html PYTHON=venv/bin/python3`.
- bpo-21514 [https://bugs.python.org/issue?@action=redirect&bpo=21514]: The documentation of the json module now refers to new JSON RFC 7159 instead of obsoleted RFC 4627.
- bpo-21777 [https://bugs.python.org/issue?@action=redirect&bpo=21777]: The binary sequence methods on bytes and bytearray are now documented explicitly, rather than assuming users will be able to derive the expected behaviour from the behaviour of the corresponding str methods.
- bpo-6916 [https://bugs.python.org/issue?@action=redirect&bpo=6916]: undocument deprecated asynchat.fifo class.
- bpo-17386 [https://bugs.python.org/issue?@action=redirect&bpo=17386]: Expanded functionality of the `Doc/make.bat` script to make it much more comparable to `Doc/Makefile`.
- bpo-21312 [https://bugs.python.org/issue?

@action=redirect&bpo=21312]: Update the thread_foobar.h template file to include newer threading APIs. Patch by Jack McCracken.
- bpo-21043 [https://bugs.python.org/issue?@action=redirect&bpo=21043]: Remove the recommendation for specific CA organizations and to mention the ability to load the OS certificates.
- bpo-20765 [https://bugs.python.org/issue?@action=redirect&bpo=20765]: Add missing documentation for PurePath.with_name() and PurePath.with_suffix().
- bpo-19407 [https://bugs.python.org/issue?@action=redirect&bpo=19407]: New package installation and distribution guides based on the Python Packaging Authority tools. Existing guides have been retained as legacy links from the distutils docs, as they still contain some required reference material for tool developers that isn't recorded anywhere else.
- bpo-19697 [https://bugs.python.org/issue?@action=redirect&bpo=19697]: Document cases where __main__.__spec__ is None.

Tests

- bpo-18982 [https://bugs.python.org/issue?@action=redirect&bpo=18982]: Add tests for CLI of the calendar module.
- bpo-19548 [https://bugs.python.org/issue?@action=redirect&bpo=19548]: Added some additional checks to test_codecs to ensure that statements in the updated documentation remain accurate. Patch by Martin Panter.
- bpo-22838 [https://bugs.python.org/issue?@action=redirect&bpo=22838]: All test_re tests now work with unittest test discovery.
- bpo-22173 [https://bugs.python.org/issue?@action=redirect&bpo=22173]: Update lib2to3 tests to use unittest test discovery.
- bpo-16000 [https://bugs.python.org/issue?@action=redirect&bpo=16000]: Convert test_curses to use unittest.
- bpo-21456 [https://bugs.python.org/issue?@action=redirect&bpo=21456]: Skip two tests in test_urllib2net.py

if _ssl module not present. Patch by Remi Pointel.

- bpo-20746 [https://bugs.python.org/issue?@action=redirect&bpo=20746]: Fix test_pdb to run in refleak mode (-R). Patch by Xavier de Gaye.
- bpo-22060 [https://bugs.python.org/issue?@action=redirect&bpo=22060]: test_ctypes has been somewhat cleaned up and simplified; it now uses unittest test discovery to find its tests.
- bpo-22104 [https://bugs.python.org/issue?@action=redirect&bpo=22104]: regrtest.py no longer holds a reference to the suite of tests loaded from test modules that don’t define test_main().
- bpo-22111 [https://bugs.python.org/issue?@action=redirect&bpo=22111]: Assorted cleanups in test_imaplib. Patch by Milan Oberkirch.
- bpo-22002 [https://bugs.python.org/issue?@action=redirect&bpo=22002]: Added `load_package_tests` function to test.support and used it to implement/augment test discovery in test_asyncio, test_email, test_importlib, test_json, and test_tools.
- bpo-21976 [https://bugs.python.org/issue?@action=redirect&bpo=21976]: Fix test_ssl to accept LibreSSL version strings. Thanks to William Orr.
- bpo-21918 [https://bugs.python.org/issue?@action=redirect&bpo=21918]: Converted test_tools from a module to a package containing separate test files for each tested script.
- bpo-9554 [https://bugs.python.org/issue?@action=redirect&bpo=9554]: Use modern unittest features in test_argparse. Initial patch by Denver Coneybeare and Radu Voicilas.
- bpo-20155 [https://bugs.python.org/issue?@action=redirect&bpo=20155]: Changed HTTP method names in failing tests in test_httpservers so that packet filtering software (specifically Windows Base Filtering Engine) does not interfere with the transaction semantics expected by the tests.
- bpo-19493 [https://bugs.python.org/issue?@action=redirect&bpo=19493]: Refactored the ctypes test package to skip tests explicitly rather than silently.
- bpo-18492 [https://bugs.python.org/issue?

@action=redirect&bpo=18492]: All resources are now allowed when tests are not run by regrtest.py.

- bpo-21634 [https://bugs.python.org/issue?@action=redirect&bpo=21634]: Fix pystone micro-benchmark: use floor division instead of true division to benchmark integers instead of floating-point numbers. Set pystone version to 1.2. Patch written by Lennart Regebro.
- bpo-21605 [https://bugs.python.org/issue?@action=redirect&bpo=21605]: Added tests for Tkinter images.
- bpo-21493 [https://bugs.python.org/issue?@action=redirect&bpo=21493]: Added test for ntpath.expanduser(). Original patch by Claudiu Popa.
- bpo-19925 [https://bugs.python.org/issue?@action=redirect&bpo=19925]: Added tests for the spwd module. Original patch by Vajrasky Kok.
- bpo-21522 [https://bugs.python.org/issue?@action=redirect&bpo=21522]: Added Tkinter tests for Listbox.itemconfigure(), PanedWindow.paneconfigure(), and Menu.entryconfigure().
- bpo-17756 [https://bugs.python.org/issue?@action=redirect&bpo=17756]: Fix test_code test when run from the installed location.
- bpo-17752 [https://bugs.python.org/issue?@action=redirect&bpo=17752]: Fix distutils tests when run from the installed location.
- bpo-18604 [https://bugs.python.org/issue?@action=redirect&bpo=18604]: Consolidated checks for GUI availability. All platforms now at least check whether Tk can be instantiated when the GUI resource is requested.
- bpo-21275 [https://bugs.python.org/issue?@action=redirect&bpo=21275]: Fix a socket test on KFreeBSD.
- bpo-21223 [https://bugs.python.org/issue?@action=redirect&bpo=21223]: Pass test_site/test_startup_imports when some of the extensions are built as builtins.
- bpo-20635 [https://bugs.python.org/issue?@action=redirect&bpo=20635]: Added tests for Tk geometry managers.
- Add test case for freeze.
- bpo-20743 [https://bugs.python.org/issue?@action=redirect&bpo=20743]: Fix a reference leak in test_tcl.

- bpo-21097 [https://bugs.python.org/issue?@action=redirect&bpo=21097]: Move test_namespace_pkgs into test_importlib.
- bpo-21503 [https://bugs.python.org/issue?@action=redirect&bpo=21503]: Use test_both() consistently in test_importlib.
- bpo-20939 [https://bugs.python.org/issue?@action=redirect&bpo=20939]: Avoid various network test failures due to new redirect of http://www.python.org/ to https://www.python.org: use http://www.example.com instead.
- bpo-20668 [https://bugs.python.org/issue?@action=redirect&bpo=20668]: asyncio tests no longer rely on tests.txt file. (Patch by Vajrasky Kok)
- bpo-21093 [https://bugs.python.org/issue?@action=redirect&bpo=21093]: Prevent failures of ctypes test_macholib on OS X if a copy of libz exists in $HOME/lib or /usr/local/lib.
- bpo-22770 [https://bugs.python.org/issue?@action=redirect&bpo=22770]: Prevent some Tk segfaults on OS X when running gui tests.
- bpo-23211 [https://bugs.python.org/issue?@action=redirect&bpo=23211]: Workaround test_logging failure on some OS X 10.6 systems.
- bpo-23345 [https://bugs.python.org/issue?@action=redirect&bpo=23345]: Prevent test_ssl failures with large OpenSSL patch level values (like 0.9.8zc).

Tools/Demos

- bpo-22314 [https://bugs.python.org/issue?@action=redirect&bpo=22314]: pydoc now works when the LINES environment variable is set.
- bpo-22615 [https://bugs.python.org/issue?@action=redirect&bpo=22615]: Argument Clinic now supports the «type» argument for the int converter. This permits using the int converter with enums and typedefs.
- bpo-20076 [https://bugs.python.org/issue?@action=redirect&bpo=20076]: The makelocalealias.py script no longer ignores UTF-8 mapping.

- bpo-20079 [https://bugs.python.org/issue?@action=redirect&bpo=20079]: The makelocalealias.py script now can parse the SUPPORTED file from glibc sources and supports command line options for source paths.
- bpo-22201 [https://bugs.python.org/issue?@action=redirect&bpo=22201]: Command-line interface of the zipfile module now correctly extracts ZIP files with directory entries. Patch by Ryan Wilson.
- bpo-22120 [https://bugs.python.org/issue?@action=redirect&bpo=22120]: For functions using an unsigned integer return converter, Argument Clinic now generates a cast to that type for the comparison to -1 in the generated code. (This suppresses a compilation warning.)
- bpo-18974 [https://bugs.python.org/issue?@action=redirect&bpo=18974]: Tools/scripts/diff.py now uses argparse instead of optparse.
- bpo-21906 [https://bugs.python.org/issue?@action=redirect&bpo=21906]: Make Tools/scripts/md5sum.py work in Python 3. Patch by Zachary Ware.
- bpo-21629 [https://bugs.python.org/issue?@action=redirect&bpo=21629]: Fix Argument Clinic's «–converters» feature.
- Add support for `yield from` to 2to3.
- Add support for the **PEP 465** [https://peps.python.org/pep-0465/] matrix multiplication operator to 2to3.
- bpo-16047 [https://bugs.python.org/issue?@action=redirect&bpo=16047]: Fix module exception list and __file__ handling in freeze. Patch by Meador Inge.
- bpo-11824 [https://bugs.python.org/issue?@action=redirect&bpo=11824]: Consider ABI tags in freeze. Patch by Meador Inge.
- bpo-20535 [https://bugs.python.org/issue?@action=redirect&bpo=20535]: PYTHONWARNING no longer affects the run_tests.py script. Patch by Arfrever Frehtes Taifersar Arahesis.

Windows

- bpo-23260 [https://bugs.python.org/issue?@action=redirect&bpo=23260]: Update Windows installer

- The bundled version of Tcl/Tk has been updated to 8.6.3. The most visible result of this change is the addition of new native file dialogs when running on Windows Vista or newer. See Tcl/Tk's TIP 432 for more information. Also, this version of Tcl/Tk includes support for Windows 10.
- bpo-17896 [https://bugs.python.org/issue?@action=redirect&bpo=17896]: The Windows build scripts now expect external library sources to be in `PCbuild\..\externals` rather than `PCbuild\..\..`.
- bpo-17717 [https://bugs.python.org/issue?@action=redirect&bpo=17717]: The Windows build scripts now use a copy of NASM pulled from svn.python.org to build OpenSSL.
- bpo-21907 [https://bugs.python.org/issue?@action=redirect&bpo=21907]: Improved the batch scripts provided for building Python.
- bpo-22644 [https://bugs.python.org/issue?@action=redirect&bpo=22644]: The bundled version of OpenSSL has been updated to 1.0.1j.
- bpo-10747 [https://bugs.python.org/issue?@action=redirect&bpo=10747]: Use versioned labels in the Windows start menu. Patch by Olive Kilburn.
- bpo-22980 [https://bugs.python.org/issue?@action=redirect&bpo=22980]: .pyd files with a version and platform tag (for example, «.cp35-win32.pyd») will now be loaded in preference to those without tags.

(For information about older versions, consult the HISTORY file.)

El tutorial de Python

Truco

This tutorial is designed for *programmers* that are new to the Python language, **not** *beginners* who are new to programming.

Python es un lenguaje de programación potente y fácil de aprender. Tiene estructuras de datos de alto nivel eficientes y un simple pero efectivo sistema de programación orientado a objetos. La elegante sintaxis de Python y su tipado dinámico, junto a su naturaleza interpretada lo convierten en un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en muchas áreas, para la mayoría de plataformas.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python website, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

El intérprete de Python es fácilmente extensible con funciones y tipos de datos implementados en C o C++ (u otros lenguajes que permitan ser llamados desde C). Python también es apropiado como un lenguaje para extender aplicaciones modificables.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. Be aware that it expects you to have a basic understanding of programming in general. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

Para una descripción de los objetos estándar y de los módulos, ver La biblioteca estándar de Python. Referencia del Lenguaje Python dónde se ofrece una definición más formal del lenguaje. Para escribir extensiones en C o C++, leer Ampliación e incrustación del intérprete de Python y Python/C API reference manual. Existen diversos libros que cubren Python en detalle.

Este tutorial no pretende ser exhaustivo ni cubrir cada una de las características del lenguaje, ni siquiera las más utilizadas. En vez de eso, pretende introducir muchas de las funcionalidades más notables y brindar una idea clara acerca del estilo y el tipo de lenguaje que es Python. Después de leerlo podrás leer y escribir módulos y programas en Python, y estarás listo para aprender más acerca de las diversas librerías y módulos descritos en La biblioteca estándar de Python.

Es interesante leer el Glosario.

- 1. Abriendo el apetito
- 2. Usando el intérprete de Python
    - 2.1. Invocar el intérprete
        - 2.1.1. Paso de argumentos
        - 2.1.2. Modo interactivo
    - 2.2. El intérprete y su entorno
        - 2.2.1. Codificación del código fuente
- 3. Una introducción informal a Python
    - 3.1. Usando Python como una calculadora
        - 3.1.1. Números
        - 3.1.2. Texto
        - 3.1.3. Listas
    - 3.2. Primeros pasos hacia la programación
- 4. Más herramientas para control de flujo
    - 4.1. La sentencia `if`
    - 4.2. La sentencia `for`

  - 4.3. La función `range()`
  - 4.4. Las sentencias `break` y `continue`
  - 4.5. Cláusulas `else` en bucles
  - 4.6. La sentencia `pass`
  - 4.7. La sentencia `match`
  - 4.8. Definir funciones
  - 4.9. Más sobre definición de funciones
    - 4.9.1. Argumentos con valores por omisión
    - 4.9.2. Palabras claves como argumentos
    - 4.9.3. Parámetros especiales
      - 4.9.3.1. Argumentos posicionales o de palabras claves
      - 4.9.3.2. Parámetros únicamente posicionales
      - 4.9.3.3. Argumentos únicamente de palabras clave
      - 4.9.3.4. Ejemplos de Funciones
      - 4.9.3.5. Resumen
    - 4.9.4. Listas de argumentos arbitrarios
    - 4.9.5. Desempaquetando una lista de argumentos
    - 4.9.6. Expresiones lambda
    - 4.9.7. Cadenas de texto de documentación
    - 4.9.8. Anotación de funciones
  - 4.10. Intermezzo: Estilo de programación
- 5. Estructuras de datos
  - 5.1. Más sobre listas
    - 5.1.1. Usar listas como pilas
    - 5.1.2. Usar listas como colas
    - 5.1.3. Comprensión de listas
    - 5.1.4. Listas por comprensión anidadas
  - 5.2. La instrucción `del`
  - 5.3. Tuplas y secuencias
  - 5.4. Conjuntos

    - 5.5. Diccionarios
    - 5.6. Técnicas de iteración
    - 5.7. Más acerca de condiciones
    - 5.8. Comparando secuencias y otros tipos
- 6. Módulos
    - 6.1. Más sobre los módulos
        - 6.1.1. Ejecutar módulos como scripts
        - 6.1.2. El camino de búsqueda de los módulos
        - 6.1.3. Archivos «compilados» de Python
    - 6.2. Módulos estándar
    - 6.3. La función **`dir()`**
    - 6.4. Paquetes
        - 6.4.1. Importar * desde un paquete
        - 6.4.2. Referencias internas en paquetes
        - 6.4.3. Paquetes en múltiples directorios
- 7. Entrada y salida
    - 7.1. Formateo elegante de la salida
        - 7.1.1. Formatear cadenas literales
        - 7.1.2. El método format() de cadenas
        - 7.1.3. Formateo manual de cadenas
        - 7.1.4. Viejo formateo de cadenas
    - 7.2. Leyendo y escribiendo archivos
        - 7.2.1. Métodos de los objetos Archivo
        - 7.2.2. Guardar datos estructurados con **`json`**
- 8. Errores y excepciones
    - 8.1. Errores de sintaxis
    - 8.2. Excepciones
    - 8.3. Gestionando excepciones
    - 8.4. Lanzando excepciones
    - 8.5. Encadenamiento de excepciones

  - 8.6. Excepciones definidas por el usuario
  - 8.7. Definiendo acciones de limpieza
  - 8.8. Acciones predefinidas de limpieza
  - 8.9. Lanzando y gestionando múltiples excepciones no relacionadas
  - 8.10. Enriqueciendo excepciones con notas

- 9. Clases

  - 9.1. Unas palabras sobre nombres y objetos
  - 9.2. Ámbitos y espacios de nombres en Python

    - 9.2.1. Ejemplo de ámbitos y espacios de nombre

  - 9.3. Un primer vistazo a las clases

    - 9.3.1. Sintaxis de definición de clases
    - 9.3.2. Objetos clase
    - 9.3.3. Objetos instancia
    - 9.3.4. Objetos método
    - 9.3.5. Variables de clase y de instancia

  - 9.4. Algunas observaciones
  - 9.5. Herencia

    - 9.5.1. Herencia múltiple

  - 9.6. Variables privadas
  - 9.7. Detalles y Cuestiones Varias
  - 9.8. Iteradores
  - 9.9. Generadores
  - 9.10. Expresiones generadoras

- 10. Brief tour of the standard library

  - 10.1. Operating system interface
  - 10.2. File wildcards
  - 10.3. Command-line arguments
  - 10.4. Error output redirection and program termination
  - 10.5. String pattern matching
  - 10.6. Matemáticas
  - 10.7. Internet access

    - 10.8. Dates and times
    - 10.9. Data compression
    - 10.10. Performance measurement
    - 10.11. Quality control
    - 10.12. Batteries included

- 11. Brief tour of the standard library — part II

    - 11.1. Output formatting
    - 11.2. Plantillas
    - 11.3. Working with binary data record layouts
    - 11.4. Multi-hilos
    - 11.5. Registrando
    - 11.6. Weak references
    - 11.7. Tools for working with lists
    - 11.8. Decimal floating-point arithmetic

- 12. Entornos virtuales y paquetes

    - 12.1. Introducción
    - 12.2. Creando entornos virtuales
    - 12.3. Manejando paquetes con pip

- 13. ¿Y ahora qué?
- 14. Edición de entrada interactiva y sustitución de historial

    - 14.1. Autocompletado con tab e historial de edición
    - 14.2. Alternativas al intérprete interactivo

- 15. Floating-Point Arithmetic: Issues and Limitations

    - 15.1. Error de Representación

- 16. Apéndice

    - 16.1. Modo interactivo

        - 16.1.1. Manejo de errores
        - 16.1.2. Programas ejecutables de Python
        - 16.1.3. El archivo de inicio interactivo
        - 16.1.4. Los módulos de customización

1. Abriendo el apetito

Si trabajas mucho con ordenadores, en algún momento encontrarás que hay alguna tarea que quieres automatizar. Por ejemplo, quizás quieres buscar y remplazar un texto en muchos ficheros o renombrar y reordenar un montón de imágenes de forma complicada. Quizás lo que quieres es escribir una pequeña base de datos personalizada, una interfaz gráfica o un juego simple.

Si eres un desarrollador profesional, quizás quieres trabajar con varias librerías de C/C++/Java pero encuentras el ciclo de escribir/compilar/probar/recompilar bastante lento. Quizás estás escribiendo una serie de pruebas para éstas librerías y te parece tedioso escribir el código de pruebas. O quizás has escrito un programa que puede utilizar un lenguaje como extensión y no quieres diseñar e implementar un lenguaje entero para tu aplicación.

Python es justo el lenguaje para ti.

Puedes escribir un script de shell de Unix o archivos por lotes de Windows para algunas de estas tareas. Los scripts de shell son mejores para mover archivos y cambiar datos de texto, pero no son adecuados para juegos o aplicaciones GUI. Podrías escribir un programa C / C ++ / Java, pero llevaría mucho tiempo de desarrollo obtener incluso un primer programa de borrador. Python es más fácil de usar, está disponible en los sistemas operativos Windows, macOS y Unix, y te ayudará a hacer el trabajo más rápidamente.

Python es fácil de utilizar y es un verdadero lenguaje de programación ofreciendo mucha más estructura y soporte para programas grandes que la que ofrecen scripts de shell o archivos por lotes. Por otro lado, Python también ofrece mayor comprobación de errores que C y siendo un *lenguaje de muy alto nivel* tiene tipos de datos de alto nivel incorporados como listas flexibles y diccionarios. Debido a sus tipos de datos más generales, Python es aplicable a más dominios que Awk o Perl, aunque hay muchas cosas que son tan sencillas en Python como en esos lenguajes.

Python te permite dividir tu programa en módulos que pueden reutilizarse en otros programas de Python. Tiene una gran colección de módulos estándar que puedes utilizar como la base de tus programas o como ejemplos para empezar a aprender Python. Algunos de estos módulos proporcionan cosas como entrada/salida de ficheros, llamadas a sistema, sockets e incluso interfaces a herramientas de interfaz gráfica como Tk.

Python es un lenguaje interpretado, lo cual puede ahorrarte mucho tiempo durante el desarrollo ya que no es necesario compilar ni enlazar. El intérprete puede usarse interactivamente, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones cuando se hace desarrollo de programas de abajo hacia arriba. Es también una calculadora de escritorio práctica.

Python permite escribir programas compactos y legibles. Los programas en Python son típicamente más cortos que sus programas equivalentes en C, C++ o Java por varios motivos:

- los tipos de datos de alto nivel permiten expresar operaciones complejas en una sola instrucción;
- la agrupación de instrucciones se hace mediante indentación en vez de llaves de apertura y cierre;
- no es necesario declarar variables ni argumentos.

Python es *extensible*: si ya sabes programar en C es fácil añadir nuevas funciones o módulos al intérprete, ya sea para realizar operaciones críticas a velocidad máxima, o para enlazar programas de Python con bibliotecas que tal vez sólo estén disponibles de forma binaria (por ejemplo bibliotecas gráficas específicas de un fabricante). Una vez estés realmente entusiasmado, puedes enlazar el intérprete Python en una aplicación hecha en C y usarlo como lenguaje de extensión o de comando para esa aplicación.

Por cierto, el lenguaje recibe su nombre del programa de televisión de la BBC «Monty Python's Flying Circus» y no tiene nada que ver con reptiles. Hacer referencias sobre Monty Python en la documentación no sólo esta permitido, ¡sino que también está bien visto!

Ahora que estás emocionado con Python, querrás verlo en más detalle. Como la mejor forma de aprender un lenguaje es usarlo, el tutorial te invita a que juegues con el intérprete de Python a medida que vas leyendo.

En el próximo capítulo se explicará la mecánica de uso del intérprete. Esta es información bastante mundana, pero es esencial para poder probar los ejemplos que aparecerán más adelante.

El resto del tutorial introduce varias características del lenguaje y el sistema Python a través de ejemplos, empezando con expresiones, instrucciones y tipos de datos simples, pasando por funciones y módulos, y finalmente tocando conceptos avanzados como excepciones y clases definidas por el usuario.

[1]

2. Usando el intérprete de Python

2.1. Invocar el intérprete

The Python interpreter is usually installed as `/usr/local/bin/python3.14` on those machines where it is available; putting `/usr/local/bin` in your Unix shell's search path makes it possible to start it by typing the command:

```
python3.14
```

en el terminal [1]. Ya que la elección del directorio dónde vivirá el intérprete es una opción del proceso de instalación, puede estar en otros lugares; consulta a tu gurú de Python local o administrador de sistemas. (Por ejemplo, `/usr/local/python` es una alternativa popular).

On Windows machines where you have installed Python from the Microsoft Store, the `python3.14` command will be available. If you have the py.exe launcher installed, you can use the `py` command. See Python install manager for other ways to launch Python.

Se puede salir del intérprete con estado de salida cero ingresando el carácter de fin de archivo (`Control-D` en Unix, `Control-Z` en Windows). Si eso no funciona, puedes salir del intérprete escribiendo el comando: `quit()`.

The interpreter's line-editing features include interactive editing, history substitution and code completion on most systems. Perhaps the quickest check to see whether command line editing is supported is typing a word in on the Python prompt, then pressing Left arrow (or `Control-b`). If the cursor moves, you have command line editing; see Appendix Edición de entrada interactiva y sustitución de historial for an introduction to the keys. If nothing appears to happen, or if a sequence like `^[[D` or `^B` appears, command line editing isn't available; you'll only be able to use backspace to remove characters from the current line.

El intérprete funciona de manera similar al shell de Unix: cuando se le llama con una entrada estándar conectada a un terminal, lee y ejecuta comandos de manera interactiva; cuando se le llama con un argumento de nombre de archivo o con un archivo como entrada estándar, lee y ejecuta un *script* desde ese archivo.

Una segunda forma de iniciar el intérprete es `python -c comando [arg] ...`, que ejecuta las sentencias en *comando*, similar a la opción de shell **`-c`** . Como las sentencias de Python a menudo contienen espacios u otros caracteres que son especiales para el shell, generalmente se recomienda citar *comando* en su totalidad.

Algunos módulos de Python también son útiles como scripts. Estos pueden invocarse utilizando `python -m módulo [arg] ...`, que ejecuta el archivo fuente para *módulo* como si se hubiera escrito el nombre completo en la línea de comandos.

Cuando se usa un script, a veces es útil poder ejecutar el script y luego ingresar al modo interactivo. Esto se puede hacer pasando la **`-i`** antes del nombre del script.

Todas las opciones de la línea de comandos se describen en Línea de comandos y entorno.

2.1.1. Paso de argumentos

Cuando son conocidos por el intérprete, el nombre del script y los argumentos adicionales se convierten a una lista de cadenas de texto asignada a la variable `argv` del módulo `sys`. Puedes acceder a esta lista haciendo `import sys`. La longitud de la lista es al menos uno; cuando no se utiliza ningún script o argumento, `sys.argv[0]` es una cadena vacía. Cuando se pasa el nombre del script con `'-'` (lo que significa la entrada estándar), `sys.argv[0]` vale `'-'`. Cuando se usa **`-c`** *comando*, `sys.argv[0]` vale `'-c'`. Cuando se usa **`-m`** *módulo*, `sys.argv[0]` contiene el valor del nombre completo del módulo. Las opciones encontradas después de **`-c`** *comando* o **`-m`** *módulo* no son consumidas por el procesador de opciones de Python pero de todas formas se almacenan en `sys.argv` para ser manejadas por el comando o módulo.

2.1.2. Modo interactivo

Cuando se leen los comandos desde un terminal, se dice que el intérprete está en *modo interactivo*. En este modo, espera el siguiente comando con el *prompt primario*, generalmente tres signos de mayor que (>>>); para las líneas de continuación, aparece el *prompt secundario*, por defecto tres puntos (`...`). El intérprete imprime un mensaje de bienvenida que indica su número de versión y un aviso de copyright antes de imprimir el primer *prompt primario*:

```
$ python3.14
Python 3.14 (default, April 4 2024, 09:25:04)
[GCC 10.2.0] on linux
Type "help", "copyright", "credits" or "license" for mor
>>>
```

Las líneas de continuación son necesarias cuando se ingresa una construcción multilínea. Como ejemplo, echa un vistazo a la sentencia **`if`**:

```
>>> the_world_is_flat = True
>>> if the_world_is_flat:
...     print("Be careful not to fall off!")
...
Be careful not to fall off!
```

Para más información sobre el modo interactivo, ver Modo interactivo.

2.2. El intérprete y su entorno

2.2.1. Codificación del código fuente

De forma predeterminada, los archivos fuente de Python se tratan como codificados en UTF-8. En esa codificación, los caracteres de la mayoría de los idiomas del mundo se pueden usar simultáneamente en literales, identificadores y comentarios, aunque la biblioteca estándar solo usa caracteres ASCII para los identificadores, una convención que debería seguir cualquier código que sea portable.Para mostrar todos estos caracteres correctamente, tu editor debe reconocer que el archivo es UTF-8, y debe usar una fuente que admita todos los caracteres del archivo.

Para declarar una codificación que no sea la predeterminada, se debe agregar una línea de comentario especial como la *primera* línea del archivo. La sintaxis es la siguiente:

```
# -*- coding: encoding -*-
```

donde *encoding* es uno de los **`codecs`** soportados por Python.

Por ejemplo, para declarar que se utilizará la codificación de Windows-1252, la primera línea del archivo de código fuente debe ser:

```
# -*- coding: cp1252 -*-
```

Una excepción a la regla de *primera línea* es cuando el código fuente comienza con una linea UNIX «shebang». En ese caso, la declaración de codificación debe agregarse como la segunda línea del archivo. Por ejemplo:

```
#!/usr/bin/env python3
# -*- coding: cp1252 -*-
```

Notas al pie

En Unix, el intérprete de Python 3.x no está instalado por defecto con el ejecutable llamado `python`, por lo que no entra en conflicto con un ejecutable de Python 2.x instalado simultáneamente.

[1] [2]

3. Una introducción informal a Python

En los siguientes ejemplos, la entrada y la salida se distinguen por la presencia o ausencia de prompts (>>> y ...): para repetir el ejemplo, escribe todo después del prompt, cuando aparece; las líneas que no comienzan con un prompt son emitidas desde el intérprete. Ten en cuenta que en un ejemplo un prompt secundario en una linea en solitario significa que debes escribir una línea en blanco. Esto se utiliza para finalizar un comando multilínea.

You can use the «Copy» button (it appears in the upper-right corner when hovering over or tapping a code example), which strips prompts and omits output, to copy and paste the input lines into your interpreter.

Muchos de los ejemplos de este manual, incluso aquellos ingresados en el prompt interactivo, incluyen comentarios. Los comentarios en Python comienzan con el carácter numeral, `#`, y se extienden hasta el final visible de la línea. Un comentario quizás aparezca al comienzo de la línea o seguido de espacios en blanco o código, pero no dentro de una cadena de caracteres. Un carácter numeral dentro de una cadena de caracteres es sólo un carácter numeral. Ya que los comentarios son para aclarar código y no son interpretados por Python, pueden omitirse cuando se escriben los ejemplos.

Algunos ejemplos:

```
# this is the first comment
spam = 1  # and this is the second comment
          # ... and now a third!
text = "# This is not a comment because it's inside quot
```

3.1. Usando Python como una calculadora

Probemos algunos comandos simples de Python. Inicia el intérprete y espera el prompt primario, `>>>`. (No debería tardar mucho.)

3.1.1. Números

The interpreter acts as a simple calculator: you can type an expression into it and it will write the value. Expression syntax is straightforward: the operators `+`, `-`, `*` and `/` can be used to perform arithmetic; parentheses (`()`) can be used for grouping. For example:

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # division always returns a floating-point nu
1.6
```

Los números enteros (ej. `2`, `4`, `20`) tienen tipo **`int`**, los que tienen una parte fraccionaria (por ejemplo `5.0`, `1.6`) tiene el tipo **`float`**. Vamos a ver más acerca de los tipos numéricos más adelante en el tutorial.

La división (`/`) siempre retorna un número decimal de punto flotante. Para hacer floor division y obtener un número entero como resultado puede usarse el operador `//`; para calcular el resto puedes usar `%`:

```
>>> 17 / 3  # classic division returns a float
5.666666666666667
>>>
>>> 17 // 3  # floor division discards the fractional pa
5
>>> 17 % 3  # the % operator returns the remainder of th
2
>>> 5 * 3 + 2  # floored quotient * divisor + remainder
17
```

Con Python, es posible usar el operador `**` para calcular potencias [1]:

```
>>> 5 ** 2  # 5 squared
25
>>> 2 ** 7  # 2 to the power of 7
128
```

El signo igual (=) se usa para asignar un valor a una variable. Después, no se muestra ningún resultado antes del siguiente prompt interactivo:

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

Si una variable no está «definida» (no se le ha asignado un valor), al intentar usarla dará un error:

```
>>> n  # try to access an undefined variable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

Hay soporte completo de punto flotante; operadores con operando mezclados convertirán los enteros a punto flotante:

```
>>> 4 * 3.75 - 1
14.0
```

En el modo interactivo, la última expresión impresa se asigna a la variable `_`. Esto significa que cuando se está utilizando Python como calculadora, es más fácil seguir calculando, por ejemplo:

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
```

```
>>> round(_, 2)
113.06
```

Esta variable debe ser tratada como de sólo lectura por el usuario. No le asignes explícitamente un valor; crearás una variable local independiente con el mismo nombre enmascarando la variable con el comportamiento mágico.

Además de **`int`** y **`float`**, Python admite otros tipos de números, como **`Decimal`** y **`Fraction`**. Python también tiene soporte incorporado para complex numbers, y usa el sufijo `j` o `J` para indicar la parte imaginaria (por ejemplo, `3+5j`).

3.1.2. Texto

Python puede manipular texto (representado por el tipo **`str`**, conocido como «cadenas de caracteres») al igual que números. Esto incluye caracteres «`!`», palabras «`conejo`», nombres «`París`», oraciones «`¡Te tengo a la vista!`», etc. «`Yay! :)`». Se pueden encerrar en comillas simples (`'...'`) o comillas dobles (`"..."`) con el mismo resultado [2].

```
>>> 'spam eggs'  # single quotes
'spam eggs'
>>> "Paris rabbit got your back :)! Yay!"  # double quot
'Paris rabbit got your back :)! Yay!'
>>> '1975'  # digits and numerals enclosed in quotes are
'1975'
```

Para citar una cita, debemos «escapar» la cita precediéndola con `\`. Alternativamente, podemos usar el otro tipo de comillas:

```
>>> 'doesn\'t'  # use \' to escape the single quote...
"doesn't"
>>> "doesn't"  # ...or use double quotes instead
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
```

```
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```

En el intérprete de Python, la definición de cadena y la cadena de salida pueden verse diferentes. La función **`print()`** produce una salida más legible, omitiendo las comillas de encuadre e imprimiendo caracteres escapados y especiales:

```
>>> s = 'First line.\nSecond line.'  # \n means newline
>>> s  # without print(), special characters are include
'First line.\nSecond line.'
>>> print(s)  # with print(), special characters are int
First line.
Second line.
```

Si no quieres que los caracteres precedidos por `\` se interpreten como caracteres especiales, puedes usar *cadenas sin formato* agregando una `r` antes de la primera comilla:

```
>>> print('C:\this\name')  # here \t means tab, \n means
C:      his
ame
>>> print(r'C:\this\name')  # note the r before the quot
C:\this\name
```

Hay un aspecto sutil en las cadenas sin formato: una cadena sin formato no puede terminar en un número impar de caracteres `\`; consultar en preguntas frequentes para obtener más información y soluciones.

String literals can span multiple lines. One way is using triple-quotes: `"""..."""` or `'''...'''`. End-of-line characters are automatically included in the string, but it’s possible to prevent this by adding a `\` at the end of the line. In the following example, the initial newline is not included:

```
>>> print("""\
... Usage: thingy [OPTIONS]
...      -h                        Display this usage me
...      -H hostname               Hostname to connect t
... """)
Usage: thingy [OPTIONS]
     -h                        Display this usage messag
     -H hostname               Hostname to connect to

>>>
```

Las cadenas se pueden concatenar (pegar juntas) con el operador `+` y se pueden repetir con `*`:

```
>>> # 3 times 'un', followed by 'ium'
>>> 3 * 'un' + 'ium'
'unununium'
```

Dos o más *cadenas literales* (es decir, las encerradas entre comillas) una al lado de la otra se concatenan automáticamente.

```
>>> 'Py' 'thon'
'Python'
```

Esta característica es particularmente útil cuando quieres dividir cadenas largas:

```
>>> text = ('Put several strings within parentheses '
...         'to have them joined together.')
>>> text
'Put several strings within parentheses to have them joi
```

Esto solo funciona con dos literales, no con variables ni expresiones:

```
>>> prefix = 'Py'
>>> prefix 'thon'  # can't concatenate a variable and a
  File "<stdin>", line 1
    prefix 'thon'
           ^^^^^^
SyntaxError: invalid syntax
>>> ('un' * 3) 'ium'
  File "<stdin>", line 1
    ('un' * 3) 'ium'
               ^^^^^
```

```
SyntaxError: invalid syntax
```

Si quieres concatenar variables o una variable y un literal, usa `+`:

```
>>> prefix + 'thon'
'Python'
```

Las cadenas de texto se pueden *indexar* (subíndices), el primer carácter de la cadena tiene el índice 0. No hay un tipo de dato diferente para los caracteres; un carácter es simplemente una cadena de longitud uno:

```
>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'
```

Los índices también pueden ser números negativos, para empezar a contar desde la derecha:

```
>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
>>> word[-6]
'P'
```

Nótese que -0 es lo mismo que 0, los índice negativos comienzan desde -1.

Además de los índices, las *rebanadas* (slicing) también están soportadas. Mientras que la indexación se utiliza para obtener caracteres individuales, *rebanar* te permite obtener una subcadena:

```
>>> word[0:2]  # characters from position 0 (included) t
'Py'
>>> word[2:5]  # characters from position 2 (included) t
'tho'
```

Los índices de las rebanadas tienen valores por defecto útiles; el valor por defecto para el primer índice es cero, el valor por defecto para el segundo índice es la longitud de la cadena a rebanar.

```
>>> word[:2]   # character from the beginning to positio
'Py'
>>> word[4:]   # characters from position 4 (included) t
'on'
>>> word[-2:]  # characters from the second-last (incluc
'on'
```

Nótese cómo el inicio siempre se incluye y el final siempre se excluye. Esto asegura que `s[:i] + s[i:]` siempre sea igual a `s`:

```
>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

Una forma de recordar cómo funcionan las rebanadas es pensar que los índices apuntan *entre* caracteres, con el borde izquierdo del primer carácter numerado 0. Luego, el punto derecho del último carácter de una cadena de *n* caracteres tiene un índice *n*, por ejemplo

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

La primera fila de números da la posición de los índices 0…6 en la cadena; La segunda fila da los correspondientes indices negativos. La rebanada desde *i* hasta *j* consta de todos los caracteres entre los bordes etiquetados *i* y *j*, respectivamente.

Para índices no negativos, la longitud de la rebanada es la diferencia de los índices, si ambos están dentro de los límites. Por ejemplo, la longitud de `word[1:3]` es 2.

Intentar usar un índice que es muy grande resultará en un error:

```
>>> word[42]  # the word only has 6 characters
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Sin embargo, los índices de rebanadas fuera de rango se manejan satisfactoriamente cuando se usan para rebanar:

```
>>> word[4:42]
'on'
>>> word[42:]
''
```

Las cadenas de Python no se pueden modificar, son immutable. Por eso, asignar a una posición indexada de la cadena resulta en un error:

```
>>> word[0] = 'J'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> word[2:] = 'py'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Si necesitas una cadena diferente, deberías crear una nueva:

```
>>> 'J' + word[1:]
'Jython'
>>> word[:2] + 'py'
'Pypy'
```

La función incorporada **`len()`** retorna la longitud de una cadena:

```
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)
34
```

Ver también

Cadenas de caracteres — str

Las cadenas de texto son ejemplos de *tipos secuencias* y soportan las operaciones comunes para esos tipos.

Métodos de las cadenas de caracteres

Las cadenas de texto soportan una gran cantidad de métodos para transformaciones básicas y búsqueda.

f-strings

Literales de cadena que tienen expresiones embebidas.

Format string syntax

Aquí se da información sobre formateo de cadenas de texto con `str.format()`.

Formateo de cadenas al estilo *printf*

Aquí se describen con más detalle las antiguas operaciones para formateo utilizadas cuando una cadena de texto está a la izquierda del operador `%`.

3.1.3. Listas

Python tiene varios tipos de datos *compuestos*, utilizados para agrupar otros valores. El más versátil es la *lista*, la cual puede ser escrita como una lista de valores separados por coma (ítems) entre corchetes. Las listas pueden contener ítems de diferentes tipos, pero usualmente los ítems son del mismo tipo.

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
```

Al igual que las cadenas (y todas las demás tipos integrados sequence), las listas se pueden indexar y segmentar:

```
>>> squares[0]  # indexing returns the item
1
>>> squares[-1]
25
>>> squares[-3:]  # slicing returns a new list
```

```
[9, 16, 25]
```

Las listas también admiten operaciones como concatenación:

```
>>> squares + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

A diferencia de las cadenas, que son immutable, las listas son de tipo mutable, es decir, es posible cambiar su contenido:

```
>>> cubes = [1, 8, 27, 65, 125]  # something's wrong her
>>> 4 ** 3  # the cube of 4 is 64, not 65!
64
>>> cubes[3] = 64  # replace the wrong value
>>> cubes
[1, 8, 27, 64, 125]
```

You can also add new items at the end of the list, by using the **`list.append()`** *method* (we will see more about methods later):

```
>>> cubes.append(216)  # add the cube of 6
>>> cubes.append(7 ** 3)  # and the cube of 7
>>> cubes
[1, 8, 27, 64, 125, 216, 343]
```

Simple assignment in Python never copies data. When you assign a list to a variable, the variable refers to the *existing list*. Any changes you make to the list through one variable will be seen through all other variables that refer to it.:

```
>>> rgb = ["Red", "Green", "Blue"]
>>> rgba = rgb
>>> id(rgb) == id(rgba)  # they reference the same objec
True
>>> rgba.append("Alph")
>>> rgb
["Red", "Green", "Blue", "Alph"]
```

Todas las operaciones de rebanado retornan una nueva lista que contiene los elementos pedidos. Esto significa que la siguiente rebanada retorna una shallow copy de la lista:

```
>>> correct_rgba = rgba[:]
>>> correct_rgba[-1] = "Alpha"
>>> correct_rgba
["Red", "Green", "Blue", "Alpha"]
>>> rgba
["Red", "Green", "Blue", "Alph"]
```

También es posible asignar a una rebanada, y esto incluso puede cambiar la longitud de la lista o vaciarla totalmente:

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # replace some values
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # now remove them
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> # clear the list by replacing all the elements with
>>> letters[:] = []
>>> letters
[]
```

La función predefinida **`len()`** también sirve para las listas

```
>>> letters = ['a', 'b', 'c', 'd']
>>> len(letters)
4
```

Es posible anidar listas (crear listas que contengan otras listas), por ejemplo:

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
```

```
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

3.2. Primeros pasos hacia la programación

Of course, we can use Python for more complicated tasks than adding two and two together. For instance, we can write an initial sub-sequence of the Fibonacci series [https://en.wikipedia.org/wiki/Fibonacci_sequence] as follows:

```
>>> # Fibonacci series:
>>> # the sum of two elements defines the next
>>> a, b = 0, 1
>>> while a < 10:
...     print(a)
...     a, b = b, a+b
...
0
1
1
2
3
5
8
```

Este ejemplo introduce varias características nuevas.

- La primera línea contiene una *asignación múltiple*: las variables `a` y `b` obtienen simultáneamente los nuevos valores 0 y 1. En la última línea esto se usa nuevamente, demostrando que las expresiones de la derecha son evaluadas primero antes de que se realice cualquiera de las asignaciones. Las expresiones del lado derecho se evalúan de izquierda a derecha.

- El bucle **`while`** se ejecuta mientras la condición (aquí: `a < 10`) sea verdadera. En Python, como en C, cualquier valor entero que no sea cero es verdadero; cero es falso. La condición también puede ser una cadena de texto o una lista, de hecho, cualquier secuencia; cualquier cosa con una longitud distinta de cero es verdadera, las secuencias vacías son falsas. La prueba utilizada en el ejemplo es una comparación simple. Los operadores de comparación estándar se escriben igual que en C: `<` (menor que), `>` (mayor que), `==` (igual a), `<=` (menor que o igual a), `>=` (mayor que o igual a) y `!=` (distinto a).

- El cuerpo del bucle está *indentado*: la indentación es la forma que usa Python para agrupar declaraciones. En el intérprete interactivo debes teclear un tabulador o espacio(s) para cada línea indentada. En la práctica vas a preparar entradas más complicadas para Python con un editor de texto; todos los editores de texto modernos tienen la facilidad de agregar la indentación automáticamente. Cuando se ingresa una instrucción compuesta de forma interactiva, se debe finalizar con una línea en blanco para indicar que está completa (ya que el analizador no puede adivinar cuando tecleaste la última línea). Nota que cada línea de un bloque básico debe estar sangrada de la misma forma.

- The **`print()`** function writes the value of the argument(s) it is given. It differs from just writing the expression you want to write (as we did earlier in the calculator examples) in the way it handles multiple arguments, floating-point quantities, and strings. Strings are printed without quotes, and a space is inserted between items, so you can format things nicely, like this:

  ```
  >>> i = 256*256
  >>> print('The value of i is', i)
  The value of i is 65536
  ```

  El parámetro nombrado *end* puede usarse para evitar el salto de linea al final de la salida, o terminar la salida con una cadena diferente:

  ```
  >>> a, b = 0, 1
  >>> while a < 1000:
  ```

```
...     print(a, end=',')
...     a, b = b, a+b
...
0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,
```

Notas al pie

Debido a que `**` tiene una prioridad mayor que `-`, `-3**2` se interpretará como `-(3**2)`, por lo tanto dará como resultado `-9`. Para evitar esto y obtener `9`, puedes usar `(-3)**2`.

A diferencia de otros lenguajes, caracteres especiales como `\n` tienen el mismo significado con simple(`'...'`) y dobles (`"..."`) comillas. La única diferencia entre las dos es que dentro de las comillas simples no existe la necesidad de escapar `"` (pero tienes que escapar `\'`) y viceversa.

[1]

4. Más herramientas para control de flujo

Además de la sentencia `while` que acabamos de introducir, Python utiliza algunas más que encontraremos en este capítulo.

4.1. La sentencia `if`

Tal vez el tipo más conocido de sentencia sea el `if`. Por ejemplo:

```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
...
More
```

Puede haber cero o más bloques `elif`, y el bloque `else` es opcional. La palabra reservada `elif` es una abreviación de “else if”, y es útil para evitar un sangrado excesivo. Una secuencia `if` … `elif` … `elif` … sustituye las sentencias `switch` o `case` encontradas en otros lenguajes.

Si necesitas comparar un mismo valor con muchas constantes, o comprobar que tenga un tipo o atributos específicos puede que encuentres útil la sentencia `match`. Para más detalles véase La sentencia match.

4.2. La sentencia `for`

La sentencia `for` en Python difiere un poco de lo que uno puede estar acostumbrado en lenguajes como C o Pascal. En lugar de siempre iterar sobre una progresión aritmética de números (como en Pascal) o darle al usuario la posibilidad de definir tanto el paso de la iteración como la condición de fin (como en C), la sentencia **for** de Python itera sobre los ítems de cualquier secuencia (una lista o una cadena de texto), en el orden que aparecen en la secuencia. Por ejemplo:

```
>>> # Measure some strings:
>>> words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
...
cat 3
window 6
defenestrate 12
```

Código que modifica una colección mientras se itera sobre la misma colección puede ser complejo de hacer bien. Sin embargo, suele ser más directo iterar sobre una copia de la colección o crear una nueva colección:

```
# Create a sample collection
users = {'Hans': 'active', 'Éléonore': 'inactive', '景太

# Strategy:  Iterate over a copy
for user, status in users.copy().items():
    if status == 'inactive':
        del users[user]

# Strategy:  Create a new collection
active_users = {}
for user, status in users.items():
    if status == 'active':
        active_users[user] = status
```

4.3. La función `range()`

Si se necesita iterar sobre una secuencia de números, es apropiado utilizar la función integrada `range()`, la cual genera progresiones aritméticas:

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

El valor final dado nunca es parte de la secuencia; `range(10)` genera 10 valores, los índices correspondientes para los ítems de una secuencia de longitud 10. Es posible hacer que el rango empiece con otro número, o especificar un incremento diferente (incluso negativo; algunas veces se lo llama "paso"):

```
>>> list(range(5, 10))
[5, 6, 7, 8, 9]

>>> list(range(0, 10, 3))
[0, 3, 6, 9]

>>> list(range(-10, -100, -30))
[-10, -40, -70]
```

Para iterar sobre los índices de una secuencia, puedes combinar `range()` y `len()` así:

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print(i, a[i])
...
0 Mary
1 had
```

```
2 a
3 little
4 lamb
```

En la mayoría de los casos, sin embargo, conviene usar la función **`enumerate()`**, ver Técnicas de iteración.

Algo extraño sucede si tan sólo muestras un `range`:

```
>>> range(10)
range(0, 10)
```

El objeto retornado por **`range()`** se comporta de muchas maneras como si fuera una lista, pero no lo es. Es un objeto que retorna los ítems sucesivos de la secuencia deseada cuando iteras sobre él, pero realmente no construye la lista, ahorrando entonces espacio.

Decimos que tal objeto es iterable; esto es, que se puede usar en funciones y construcciones que esperan algo de lo cual obtener ítems sucesivos hasta que se termine. Hemos visto que la declaración **`for`** es una de esas construcciones, mientras que un ejemplo de función que toma un iterable es la función **`sum()`**:

```
>>> sum(range(4))  # 0 + 1 + 2 + 3
6
```

Later we will see more functions that return iterables and take iterables as arguments. In chapter Estructuras de datos, we will discuss **`list()`** in more detail.

4.4. Las sentencias `break` y `continue`

La sentencia **`break`** termina el bucle **`for`** o **`while`** envolvente más interno:

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(f"{n} equals {x} * {n//x}")
...             break
```

```
...
4 equals 2 * 2
6 equals 2 * 3
8 equals 2 * 4
9 equals 3 * 3
```

La sentencia `continue`, también tomada de C, continúa con la siguiente iteración del bucle:

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print(f"Found an even number {num}")
...         continue
...     print(f"Found an odd number {num}")
...
Found an even number 2
Found an odd number 3
Found an even number 4
Found an odd number 5
Found an even number 6
Found an odd number 7
Found an even number 8
Found an odd number 9
```

4.5. Cláusulas `else` en bucles

In a **`for`** or **`while`** loop the **`break`** statement may be paired with an **`else`** clause. If the loop finishes without executing the **`break`**, the **`else`** clause executes.

En un bucle `for`, la cláusula **`else`** se ejecuta después de que el bucle termine su iteración final, es decir, si no ocurrió un break.

En un bucle `while`, se ejecuta después de que la condición del bucle se vuelva falsa.

En cualquier tipo de bucle, la cláusula **`else`** **no** se ejecuta si el bucle terminó con un `break`. Por supuesto, otras formas de terminar el bucle anticipadamente, como un `return` o una excepción lanzada, también omitirán la ejecución de la cláusula **`else`**.

Esto se ejemplifica en el siguiente bucle **`for`**, que busca números primos:

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         # loop fell through without finding a factor
...         print(n, 'is a prime number')
...
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

(Sí, este es el código correcto. Fíjate bien: la cláusula `else` pertenece al bucle `for`, **no** a la sentencia `if`.)

One way to think of the else clause is to imagine it paired with the `if` inside the loop. As the loop executes, it will run a sequence like if/if/if/else. The `if` is inside the loop, encountered a number of times. If the condition is ever true, a `break` will happen. If the condition is never true, the `else` clause outside the loop will execute.

Cuando se usa con un bucle, la cláusula `else` tiene más en común con la cláusula `else` de una sentencia **`try`** que con la de sentencias `if` la cláusula `else` de una sentencia `try` se ejecuta cuando no ocurre ninguna excepción, y la cláusula `else` de un bucle se ejecuta cuando no ocurre ningún `break`. Para más información sobre la sentencia **`try`** y las excepciones, consulta

Gestionando excepciones.

4.6. La sentencia `pass`

La sentencia **`pass`** no hace nada. Se puede usar cuando una sentencia es requerida por la sintaxis pero el programa no requiere ninguna acción. Por ejemplo:

```
>>> while True:
...     pass  # Busy-wait for keyboard interrupt (Ctrl+C
...
```

Se usa normalmente para crear clases en su mínima expresión:

```
>>> class MyEmptyClass:
...     pass
...
```

Otro lugar donde se puede usar **`pass`** es como una marca de lugar para una función o un cuerpo condicional cuando estás trabajando en código nuevo, lo cual te permite pensar a un nivel de abstracción mayor. El **`pass`** se ignora silenciosamente:

```
>>> def initlog(*args):
...     pass   # Remember to implement this!
...
```

For this last case, many people use the ellipsis literal `...` instead of `pass`. This use has no special meaning to Python, and is not part of the language definition (you could use any constant expression here), but `...` is used conventionally as a placeholder body as well. See El objeto puntos suspensivos (Ellipsis).

4.7. La sentencia `match`

A **`match`** statement takes an expression and compares its value to successive patterns given as one or more case blocks. This is superficially similar to a switch statement in C, Java or JavaScript (and many other languages), but it's more similar to pattern matching in languages like Rust or Haskell. Only the first pattern that matches gets executed and it can also extract components (sequence elements or object attributes) from the value into variables. If no case matches, none of the branches is executed.

La forma más simple compara un valor expuesto con uno o más literales:

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
        case _:
            return "Something's wrong with the internet"
```

Note the last block: the «variable name» `_` acts as a *wildcard* and never fails to match.

Se pueden combinar varios literales en un solo patrón usando `|` («o»):

```
case 401 | 403 | 404:
    return "Not allowed"
```

Los patrones pueden también verse como asignaciones que desempaquetan, y pueden usarse para ligar variables:

```
# point is an (x, y) tuple
match point:
    case (0, 0):
        print("Origin")
    case (0, y):
        print(f"Y={y}")
    case (x, 0):
        print(f"X={x}")
    case (x, y):
        print(f"X={x}, Y={y}")
    case _:
        raise ValueError("Not a point")
```

¡Observa éste caso con cuidado! El primer patrón tiene dos literales y puede considerarse una extensión del patrón literal que se mostró anteriormente. Pero los siguientes dos patrones combinan un literal y una variable, y la variable *liga* uno de los elementos del sujeto (`point`). El cuarto patrón captura ambos elementos, lo que lo hace conceptualmente similar a la asignación que desempaqueta `(x, y) = point`.

Si estás usando clases para estructurar tus datos, puedes usar el nombre de la clase seguida de una lista de argumentos similar a la de un constructor, pero con la capacidad de capturar atributos en variables:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

def where_is(point):
    match point:
        case Point(x=0, y=0):
            print("Origin")
        case Point(x=0, y=y):
            print(f"Y={y}")
        case Point(x=x, y=0):
            print(f"X={x}")
        case Point():
            print("Somewhere else")
        case _:
            print("Not a point")
```

Puedes usar argumentos posicionales en algunas clases incorporadas que proveen un orden para sus atributos (por ej. dataclasses). También puedes definir una posición especifica para los atributos de los patrones si asignas en tu clase el atributo especial `__match_args__`. Si le asignas («x», «y»), los siguientes patrones son todos equivalentes entre sí (y todos ligan el atributo `y` a la variable `var`):

```
Point(1, var)
Point(1, y=var)
Point(x=1, y=var)
Point(y=var, x=1)
```

Una recomendación para leer patrones es verlos como una forma extendida de lo que pondrías en el lado izquierdo de una asignación, para así entender cuáles variables tomarían qué valores. Sólo los nombres que aparecen por si solos (cómo `var` arriba) son asignados por una sentencia match. Nunca se asigna a los nombres con puntos (como `foo.bar`), nombres de atributos (los `x=` e `y=` arriba) o nombres de clases (reconocidos por los «(…)» junto a ellos, como `Point` arriba).

Los patrones pueden anidarse arbitrariamente. Por ejemplo, si tuviéramos una lista corta de puntos, con `__match_args__` añadido, podríamos aplicar match así:

```
class Point:
    __match_args__ = ('x', 'y')
    def __init__(self, x, y):
        self.x = x
        self.y = y

match points:
    case []:
        print("No points")
    case [Point(0, 0)]:
        print("The origin")
    case [Point(x, y)]:
        print(f"Single point {x}, {y}")
    case [Point(0, y1), Point(0, y2)]:
        print(f"Two on the Y axis at {y1}, {y2}")
    case _:
        print("Something else")
```

Podemos añadir una clausula `if` a un patrón, conocida como «guarda». Si la guarda es falsa, `match` pasa a intentar el siguiente bloque case. Obsérvese que la captura de valores sucede antes de que la guarda sea evaluada:

```
match point:
    case Point(x, y) if x == y:
        print(f"Y=X at {x}")
    case Point(x, y):
        print(f"Not on the diagonal")
```

Algunas otras propiedades importantes de esta sentencia:

- Al igual que las asignaciones con desempaquetado, los patrones de lista o tupla tienen exactamente el mismo sentido y realmente coinciden con cualquier secuencia arbitraria. Una excepción importante es que no coinciden ni con iteradores ni con cadenas de caracteres.

- Los patrones de secuencia soportan desempaquetado extendido: `[x, y, *otros]` y `(x, y, *otros)` funcionan de manera similar a las asignaciones con desempaquetado. El nombre luego de `*` también puede ser _, con lo cual `(x, y, *_)` coincide con cualquier secuencia de al menos dos elementos, sin ligar ninguno de los demás elementos.

- Los patrones de mapeo: `{"ancho_de_banda": c, "latencia": l}` capturan los valores `"ancho_de_banda"` y `"latencia"` de un diccionario. A diferencia de los patrones de secuencia, las claves adicionales son ignoradas. Puede usarse un desempaquetado como `**otros` . (Aunque `**_` sería redundante, con lo cual no está permitido)

- Pueden capturarse subpatrones usando la palabra clave `as`:

  ```
  case (Point(x1, y1), Point(x2, y2) as p2): ...
  ```

  capturará el segundo elemento de la entrada en `p2` (siempre y cuando la entrada sea una secuencia de dos puntos)

- La mayoría de los literales se comparan por igualdad, pero las instancias únicas `True`, `False` y `None` se comparan por identidad.

- Patterns may use named constants. These must be dotted names to prevent them from being interpreted as capture variables:

  ```
  from enum import Enum
  class Color(Enum):
      RED = 'red'
      GREEN = 'green'
      BLUE = 'blue'

  color = Color(input("Enter your choice of 'red', 'b

  match color:
      case Color.RED:
          print("I see red!")
      case Color.GREEN:
          print("Grass is green")
      case Color.BLUE:
          print("I'm feeling the blues :(")
  ```

Para una explicación más detallada y más ejemplos, puede leerse **PEP 636** [https://peps.python.org/pep-0636/] que está escrita en un formato de tutorial.

4.8. Definir funciones

Podemos crear una función que escriba la serie de Fibonacci hasta un límite determinado:

```
>>> def fib(n):    # write Fibonacci series less than n
...     """Print a Fibonacci series less than n."""
...     a, b = 0, 1
...     while a < n:
...         print(a, end=' ')
...         a, b = b, a+b
```

```
...     print()
...
>>> # Now call the function we just defined:
>>> fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

La palabra reservada **`def`** se usa para definir funciones. Debe seguirle el nombre de la función y la lista de parámetros formales entre paréntesis. Las sentencias que forman el cuerpo de la función empiezan en la línea siguiente, y deben estar con sangría.

La primera sentencia del cuerpo de la función puede ser opcionalmente una cadena de texto literal; esta es la cadena de texto de documentación de la función, o *docstring*. (Puedes encontrar más acerca de docstrings en la sección Cadenas de texto de documentación.). Existen herramientas que usan las `docstrings` para producir documentación imprimible o disponible en línea, o para dejar que los usuarios busquen interactivamente a través del código; es una buena práctica incluir `docstrings` en el código que escribes, así que acostúmbrate a hacerlo.

La *ejecución* de una función introduce una nueva tabla de símbolos usada para las variables locales de la función. Más precisamente, todas las asignaciones de variables en la función almacenan el valor en la tabla de símbolos local; así mismo la referencia a variables primero mira la tabla de símbolos local, luego en la tabla de símbolos local de las funciones externas, luego la tabla de símbolos global, y finalmente la tabla de nombres predefinidos. Así, a variables globales y a variables de funciones que engloban a una función no se les puede asignar directamente un valor dentro de una función (a menos que se las nombre en la sentencia **`global`**, o mediante la sentencia **`nonlocal`** para variables de funciones que engloban la función local), aunque si pueden ser referenciadas.

Los parámetros reales (argumentos) para una llamada de función se introducen en la tabla de símbolos local de la función llamada cuando ésta se llama; por lo tanto, los argumentos se pasan usando *llamada por valor* (donde el *valor* es siempre una *referencia* al objeto, no el valor del objeto). [1] Cuando una función llama a otra función, o se llama a sí misma de forma recursiva, se crea una nueva tabla de símbolos locales para esa llamada.

Una definición de función asocia el nombre de la función con el objeto de función en la tabla de símbolos actual. El intérprete reconoce el objeto al que apunta ese nombre como una función definida por el usuario. Otros nombres también pueden apuntar a ese mismo objeto de función y también se pueden usar para acceder a la función:

```
>>> fib
<function fib at 10042ed0>
>>> f = fib
>>> f(100)
0 1 1 2 3 5 8 13 21 34 55 89
```

Viniendo de otros lenguajes, puedes objetar que `fib` no es una función, sino un procedimiento, porque no retorna un valor. De hecho, técnicamente hablando, los procedimientos sin **`return`** sí retornan un valor, aunque uno bastante aburrido. Este valor se llama `None` (es un nombre predefinido). El intérprete por lo general no escribe el valor `None` si va a ser el único valor escrito. Puede verlo si realmente lo desea utilizando **`print()`**:

```
>>> fib(0)
>>> print(fib(0))
None
```

Es simple escribir una función que retorne una lista con los números de la serie de Fibonacci en lugar de imprimirlos:

```
>>> def fib2(n):  # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)    # see below
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100)    # call it
>>> f100                # write the result
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Este ejemplo, como es usual, demuestra algunas características más de Python:

- La sentencia **`return`** retorna un valor en una función. **`return`** sin una expresión como argumento retorna `None`. Si se alcanza el final de una función, también se retorna `None`.
- The statement `result.append(a)` calls a *method* of the list object `result`. A method is a function that "belongs" to an object and is named `obj.methodname`, where `obj` is some object (this may be an expression), and `methodname` is the name of a method that is defined by the object's type. Different types define different methods. Methods of different types may have the same name without causing ambiguity. (It is possible to define your own object types and methods, using *classes*, see Clases) The method **`append()`** shown in the example is defined for list objects; it adds a new element at the end of the list. In this example it is equivalent to `result = result + [a]`, but more efficient.

4.9. Más sobre definición de funciones

También es posible definir funciones con un número variable de argumentos. Hay tres formas que pueden ser combinadas.

4.9.1. Argumentos con valores por omisión

La forma más útil es especificar un valor por omisión para uno o más argumentos. Esto crea una función que puede ser llamada con menos argumentos que los que permite. Por ejemplo:

```
def ask_ok(prompt, retries=4, reminder='Please try agai
    while True:
        reply = input(prompt)
        if reply in {'y', 'ye', 'yes'}:
            return True
        if reply in {'n', 'no', 'nop', 'nope'}:
            return False
        retries = retries - 1
        if retries < 0:
            raise ValueError('invalid user response')
        print(reminder)
```

Esta función puede ser llamada de distintas maneras:

- pasando sólo el argumento obligatorio: `ask_ok('Do you really want to quit?')`
- pasando uno de los argumentos opcionales: `ask_ok('OK to overwrite the file?', 2)`
- o pasando todos los argumentos: `ask_ok('OK to overwrite the file?', 2, 'Come on, only yes or no!')`

Este ejemplo también introduce la palabra reservada **`in`**, la cual prueba si una secuencia contiene o no un determinado valor.

Los valores por omisión son evaluados en el momento de la definición de la función en el ámbito de la definición, entonces:

```
i = 5

def f(arg=i):
    print(arg)

i = 6
f()
```

imprimirá `5`.

Advertencia importante: El valor por omisión es evaluado solo una vez. Existe una diferencia cuando el valor por omisión es un objeto mutable como una lista, diccionario, o instancia de la mayoría de las clases. Por ejemplo, la siguiente función acumula los argumentos que se le pasan en subsiguientes llamadas:

```
def f(a, L=[]):
    L.append(a)
    return L
```

```
print(f(1))
print(f(2))
print(f(3))
```

Imprimirá

```
[1]
[1, 2]
[1, 2, 3]
```

Si no se quiere que el valor por omisión sea compartido entre subsiguientes llamadas, se pueden escribir la función así:

```
def f(a, L=None):
    if L is None:
        L = []
    L.append(a)
    return L
```

4.9.2. Palabras claves como argumentos

Las funciones también puede ser llamadas usando argumentos de palabras clave (o argumentos nombrados) de la forma `kwarg=value`. Por ejemplo, la siguiente función:

```
def parrot(voltage, state='a stiff', action='voom', type
    print("-- This parrot wouldn't", action, end=' ')
    print("if you put", voltage, "volts through it.")
    print("-- Lovely plumage, the", type)
    print("-- It's", state, "!")
```

...acepta un argumento obligatorio (`voltage`)) y tres argumentos opcionales (`state`, `action`, y `type`). Esta función puede llamarse de cualquiera de las siguientes maneras:

```
parrot(1000)                                           #
parrot(voltage=1000)                                   #
parrot(voltage=1000000, action='VOOOOOM')              #
parrot(action='VOOOOOM', voltage=1000000)              #
parrot('a million', 'bereft of life', 'jump')         #
parrot('a thousand', state='pushing up the daisies')  #
```

…pero estas otras llamadas serían todas inválidas:

```
parrot()                     # required argument missing
parrot(voltage=5.0, 'dead')  # non-keyword argument afte
parrot(110, voltage=220)     # duplicate value for the s
parrot(actor='John Cleese')  # unknown keyword argument
```

En una llamada a una función, los argumentos nombrados deben seguir a los argumentos posicionales. Cada uno de los argumentos nombrados pasados deben coincidir con un argumento aceptado por la función (por ejemplo, `actor` no es un argumento válido para la función `parrot`), y el orden de los mismos no es importante. Esto también se aplica a los argumentos obligatorios (por ejemplo, `parrot(voltage=1000)` también es válido). Ningún argumento puede recibir más de un valor al mismo tiempo. Aquí hay un ejemplo que falla debido a esta restricción:

```
>>> def function(a):
...     pass
...
>>> function(0, a=0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: function() got multiple values for argument '
```

Cuando un parámetro formal de la forma `**nombre` está presente al final, recibe un diccionario (ver Tipos mapa — dict) conteniendo todos los argumentos nombrados excepto aquellos correspondientes a un parámetro formal. Esto puede ser combinado con un parámetro formal de la forma `*nombre` (descrito en la siguiente sección) que recibe una tupla conteniendo los argumentos posicionales además de la lista de parámetros formales. (`*nombre` debe ocurrir antes de `**nombre`). Por ejemplo, si definimos una función así:

```
def cheeseshop(kind, *arguments, **keywords):
    print("-- Do you have any", kind, "?")
    print("-- I'm sorry, we're all out of", kind)
    for arg in arguments:
        print(arg)
    print("-" * 40)
    for kw in keywords:
        print(kw, ":", keywords[kw])
```

Puede ser llamada así:

```
cheeseshop("Limburger", "It's very runny, sir.",
           "It's really very, VERY runny, sir.",
           shopkeeper="Michael Palin",
           client="John Cleese",
           sketch="Cheese Shop Sketch")
```

…y por supuesto imprimirá:

```
-- Do you have any Limburger ?
-- I'm sorry, we're all out of Limburger
It's very runny, sir.
It's really very, VERY runny, sir.
----------------------------------------
shopkeeper : Michael Palin
client : John Cleese
sketch : Cheese Shop Sketch
```

Se debe notar que el orden en el cual los argumentos nombrados son impresos está garantizado para coincidir con el orden en el cual fueron provistos en la llamada a la función.

4.9.3. Parámetros especiales

Por defecto, los argumentos pueden enviarse a una función Python o bien por posición o explícitamente por clave. Para legibilidad y rendimiento tiene sentido restringir como se pueden enviar los argumentos, así un desarrollador necesitará mirar solamente la definición de la función para determinar si los argumentos se deben enviar por posición, por posición o clave, o por clave.

La definición de una función puede ser como la siguiente:

```
def f(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
      -----------    ----------     ----------
        |             |                  |
        |        Positional or keyword   |
        |                                - Keyword only
         -- Positional only
```

donde `/` y `*` son posicionales. Si se utilizan, esos símbolos indican el tipo de parámetro según la forma en que los argumentos deben enviarse a la función: solo por posición (*positional-only*), por posición o clave (*positional-or-keyword*) y solo por clave (*keyword-only*). Parámetros por clave pueden también denominarse parámetros por nombre o nombrados.

4.9.3.1. Argumentos posicionales o de palabras claves

Si `/` y `*` no están presentes en la definición de la función, los parámetros pueden ser pasados a una función posicionalmente o por palabra clave.

4.9.3.2. Parámetros únicamente posicionales

Mirando esto con un poco más de detalle, es posible señalar algunos parámetros como *únicamente posicionales*. En ese caso el orden de los parámetros es importante, y los parámetros no pueden ser indicados utilizando palabras claves. Parámetros únicamente posicionales son ubicados antes de una `/` (barra). La `/` es utilizada para separar lógicamente parámetros únicamente posicionales del resto. Si no existe una `/` en la definición de la función, no existen parámetros únicamente posicionales.

Los parámetros luego de una `/` pueden ser *únicamente posicionales* o *unicamente de palabras claves*.

4.9.3.3. Argumentos únicamente de palabras clave

Para señalar parámetros como *unicamente de palabras clave*, indicando que los parámetros deben ser pasados con una palabra clave, indiqué un `*` en la lista de argumentos antes del primer parámetro *únicamente de palabras clave*.

4.9.3.4. Ejemplos de Funciones

Considere el siguiente ejemplo de definiciones de funciones prestando especial atención a los marcadores `/` y `*`:

```
>>> def standard_arg(arg):
...     print(arg)
...
>>> def pos_only_arg(arg, /):
...     print(arg)
...
>>> def kwd_only_arg(*, arg):
...     print(arg)
...
>>> def combined_example(pos_only, /, standard, *, kwd_o
...     print(pos_only, standard, kwd_only)
```

La primer definición de función, `standard_arg`, la forma mas familiar, no indica ninguna restricción en las condiciones para llamarla y los parámetros deben ser pasados por posición o utilizando palabras clave:

```
>>> standard_arg(2)
2

>>> standard_arg(arg=2)
2
```

La segunda función `pos_only_arg` está restringida a utilizar únicamente parámetros posicionales ya que existe una `/` en la definición de la función:

```
>>> pos_only_arg(1)
1

>>> pos_only_arg(arg=1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

```
TypeError: pos_only_arg() got some positional-only argum
```

The third function `kwd_only_arg` only allows keyword arguments as indicated by a `*` in the function definition:

```
>>> kwd_only_arg(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: kwd_only_arg() takes 0 positional arguments b

>>> kwd_only_arg(arg=3)
3
```

La última utiliza las tres convenciones en una misma definición de función:

```
>>> combined_example(1, 2, 3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: combined_example() takes 2 positional argumen

>>> combined_example(1, 2, kwd_only=3)
1 2 3

>>> combined_example(1, standard=2, kwd_only=3)
1 2 3

>>> combined_example(pos_only=1, standard=2, kwd_only=3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: combined_example() got some positional-only a
```

Finalmente, considere esta definición de función que contiene una colisión potencial entre los parámetros posicionales `name` y `**kwds` que incluye `name` como una clave:

```
def foo(name, **kwds):
    return 'name' in kwds
```

No hay una llamada posible que lo haga retornar `True` ya que la palabra clave `'name'` siempre se vinculará al primer parámetro. Por ejemplo:

```
>>> foo(1, **{'name': 2})
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: foo() got multiple values for argument 'name'
>>>
```

Pero utilizando `/` (parámetros únicamente posicionales), es posible ya que permite utilizar `name` como un parámetro posicional y `name` como un parámetro de palabras clave:

```
>>> def foo(name, /, **kwds):
...     return 'name' in kwds
...
>>> foo(1, **{'name': 2})
True
```

En otras palabras, los nombres de parámetros únicamente posicionales pueden ser utilizados en `**kwds` sin ambigüedad.

4.9.3.5. Resumen

El caso de uso determinará qué parámetros utilizar en una definición de función:

```
def f(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
```

A modo de guía:

- Utilice únicamente posicionales si quiere que el nombre del parámetro esté disponible para el usuario. Esto es útil cuando el nombre del parámetro no tiene un significado real, si se quiere imponer el orden de los parámetros cuando una función es llamada o si necesita tomar algunos parámetros posicionales y palabras claves arbitrarias.
- Utilice parámetros únicamente de palabras clave cuando los nombres de los parámetros tienen un significado y la definición de la función será más entendible usando nombres

explícitos o cuando desea evitar que los usuarios dependan de la posición de los parámetros que se pasan.
- En el caso de una API, use solo posicional para evitar que se rompan los cambios de la API si el nombre del parámetro se modifica en el futuro.

4.9.4. Listas de argumentos arbitrarios

Finalmente, la opción menos frecuentemente usada es especificar que una función puede ser llamada con un número arbitrario de argumentos. Estos argumentos serán organizados en una tupla (ver Tuplas y secuencias). Antes del número variable de argumentos, cero o más argumentos normales pueden estar presentes.:

```
def write_multiple_items(file, separator, *args):
    file.write(separator.join(args))
```

Normalmente estos argumentos *variádicos* serán los últimos en la lista de parámetros formales, porque toman todo el remanente de argumentos que se pasan a la función. Cualquier parámetro que suceda luego del `*args` será "sólo de palabra clave", o sea que sólo se pueden usar como argumentos nombrados y no como posicionales.

```
>>> def concat(*args, sep="/"):
...     return sep.join(args)
...
>>> concat("earth", "mars", "venus")
'earth/mars/venus'
>>> concat("earth", "mars", "venus", sep=".")
'earth.mars.venus'
```

4.9.5. Desempaquetando una lista de argumentos

La situación inversa ocurre cuando los argumentos ya están en una lista o tupla pero necesitan ser desempaquetados para llamar a una función que requiere argumentos posicionales separados. Por ejemplo, la función predefinida **`range()`** espera los parámetros *inicio* y *fin*. Si estos no están disponibles en forma separada, se puede escribir la llamada a la función con el operador `*` para desempaquetar argumentos desde una lista o una tupla:

```
>>> list(range(3, 6))            # normal call with sepa
[3, 4, 5]
>>> args = [3, 6]
>>> list(range(*args))            # call with arguments
[3, 4, 5]
```

Del mismo modo, los diccionarios pueden entregar argumentos nombrados con el operador `**`:

```
>>> def parrot(voltage, state='a stiff', action='voom'):
...     print("-- This parrot wouldn't", action, end=' '
...     print("if you put", voltage, "volts through it."
...     print("E's", state, "!")
...
>>> d = {"voltage": "four million", "state": "bleedin' d
>>> parrot(**d)
-- This parrot wouldn't VOOM if you put four million vol
```

4.9.6. Expresiones lambda

Pequeñas funciones anónimas pueden ser creadas con la palabra reservada **`lambda`**. Esta función retorna la suma de sus dos argumentos: `lambda a, b: a+b` Las funciones Lambda pueden ser usadas en cualquier lugar donde sea requerido un objeto de tipo función. Están sintácticamente restringidas a una sola expresión. Semánticamente, son solo azúcar sintáctica para definiciones normales de funciones. Al igual que las funciones anidadas, las funciones lambda pueden hacer referencia a variables desde el ámbito que la contiene:

```
>>> def make_incrementor(n):
...     return lambda x: x + n
...
>>> f = make_incrementor(42)
>>> f(0)
42
>>> f(1)
43
```

The above example uses a lambda expression to return a function. Another use is to pass a small function as an argument. For instance, **`list.sort()`** takes a sorting key function *key* which can be a lambda function:

```
>>> pairs = [(1, 'one'), (2, 'two'), (3, 'three'), (4, '
>>> pairs.sort(key=lambda pair: pair[1])
>>> pairs
[(4, 'four'), (1, 'one'), (3, 'three'), (2, 'two')]
```

4.9.7. Cadenas de texto de documentación

Acá hay algunas convenciones sobre el contenido y formato de las cadenas de texto de documentación.

La primera línea debe ser siempre un resumen corto y conciso del propósito del objeto. Para ser breve, no se debe mencionar explícitamente el nombre o tipo del objeto, ya que estos están disponibles de otros modos (excepto si el nombre es un verbo que describe el funcionamiento de la función). Esta línea debe empezar con una letra mayúscula y terminar con un punto.

Si hay más líneas en la cadena de texto de documentación, la segunda línea debe estar en blanco, separando visualmente el resumen del resto de la descripción. Las líneas siguientes deben ser uno o más párrafos describiendo las convenciones para llamar al objeto, efectos secundarios, etc.

The Python parser strips indentation from multi-line string literals when they serve as module, class, or function docstrings.

Este es un ejemplo de un `docstring` multi-línea:

```
>>> def my_function():
...     """Do nothing, but document it.
...
...     No, really, it doesn't do anything:
...
...         >>> my_function()
...         >>>
...     """
...     pass
...
>>> print(my_function.__doc__)
Do nothing, but document it.

No, really, it doesn't do anything:

    >>> my_function()
    >>>
```

4.9.8. Anotación de funciones

Las anotaciones de funciones son información completamente opcional sobre los tipos usadas en funciones definidas por el usuario (ver **PEP 3107** [https://peps.python.org/pep-3107/] y **PEP 484** [https://peps.python.org/pep-0484/] para más información).

Annotations are stored in the `__annotations__` attribute of the function as a dictionary and have no effect on any other part of the function. Parameter annotations are defined by a colon after the parameter name, followed by an expression evaluating to the value of the annotation. Return annotations are defined by a literal `->`, followed by an expression, between the parameter list and the colon denoting the end of the `def` statement. The following example has a required argument, an optional argument, and the return value annotated:

```
>>> def f(ham: str, eggs: str = 'eggs') -> str:
...     print("Annotations:", f.__annotations__)
...     print("Arguments:", ham, eggs)
...     return ham + ' and ' + eggs
...
>>> f('spam')
Annotations: {'ham': <class 'str'>, 'return': <class 'st
Arguments: spam eggs
'spam and eggs'
```

4.10. Intermezzo: Estilo de programación

Now that you are about to write longer, more complex pieces of Python, it is a good time to talk about *coding style*. Most languages can be written (or more concisely, *formatted*) in different styles; some are more readable than others. Making it easy for others to read your code is always a good idea, and adopting a nice coding style helps tremendously for that.

Para Python, **PEP 8** [https://peps.python.org/pep-0008/] se erigió como la guía de estilo a la que más proyectos adhirieron; promueve un estilo de programación fácil de leer y visualmente agradable. Todos los desarrolladores Python deben leerlo en algún momento; aquí están extraídos los puntos más importantes:

- Usar sangrías de 4 espacios, no tabuladores.

  4 espacios son un buen compromiso entre una sangría pequeña (permite mayor nivel de sangrado)y una sangría grande (más fácil de leer). Los tabuladores introducen confusión y es mejor dejarlos de lado.

- Recortar las líneas para que no superen los 79 caracteres.

  Esto ayuda a los usuarios con pantallas pequeñas y hace posible tener varios archivos de código abiertos, uno al lado del otro, en pantallas grandes.

- Usar líneas en blanco para separar funciones y clases, y bloques grandes de código dentro de funciones.

- Cuando sea posible, poner comentarios en una sola línea.

- Usar `docstrings`.

- Usar espacios alrededor de operadores y luego de las comas, pero no directamente dentro de paréntesis: `a = f(1, 2) + g(3, 4)`.

- Nombrar las clases y funciones consistentemente; la convención es usar `NotacionCamello` para clases y `minusculas_con_guiones_bajos` para funciones y métodos. Siempre usa `self` como el nombre para el primer argumento en los métodos (ver Un primer vistazo a las clases para más información sobre clases y métodos).

- No uses codificaciones estrafalarias si esperas usar el código en entornos internacionales. El default de Python, UTF-8, o incluso ASCII plano funcionan bien en la mayoría de los casos.
- De la misma manera, no uses caracteres no-ASCII en los identificadores si hay incluso una pequeñísima chance de que gente que hable otro idioma tenga que leer o mantener el código.

Notas al pie

En realidad, *llamadas por referencia de objeto* sería una mejor descripción, ya que si se pasa un objeto mutable, quien realiza la llamada verá cualquier cambio que se realice sobre el mismo (por ejemplo ítems insertados en una lista).

[1]

5. Estructuras de datos

Este capítulo describe en más detalle algunas cosas que ya has aprendido y agrega algunas cosas nuevas también.

5.1. Más sobre listas

The list data type has some more methods. Here are all of the methods of list objects:

list.append(*value*, */*)

> Add an item to the end of the list. Similar to `a[len(a):] = [x]`.

list.extend(*iterable*, */*)

> Extend the list by appending all the items from the iterable. Similar to `a[len(a):] = iterable`.

list.insert(*index*, *value*, */*)

> Inserta un ítem en una posición dada. El primer argumento es el índice del ítem delante del cual se insertará, por lo tanto `a.insert(0, x)` inserta al principio de la lista y `a.insert(len(a), x)` equivale a `a.append(x)`.

list.remove(*value*, */*)

> Remove the first item from the list whose value is equal to *value*. It raises a **`ValueError`** if there is no such item.

list.pop(*index=-1*, */*)

> Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. It raises an **`IndexError`** if the list is empty or the index is outside the list range.

list.clear()

Remove all items from the list. Similar to `del a[:]`.

list.index(*value*[, *start*[, *stop*]])

Return zero-based index of the first occurrence of *value* in the list. Raises a **`ValueError`** if there is no such item.

Los argumentos opcionales *start* y *end* son interpretados como la notación de rebanadas y se usan para limitar la búsqueda a un segmento particular de la lista. El índice retornado se calcula de manera relativa al inicio de la secuencia completa en lugar de hacerlo con respecto al argumento *start*.

list.count(*value*, */*)

Return the number of times *value* appears in the list.

list.sort(*, *key=None*, *reverse=False*)

Ordena los elementos de la lista in situ (los argumentos pueden ser usados para personalizar el orden de la lista, ver **`sorted()`** para su explicación).

list.reverse()

Invierte los elementos de la lista in situ.

list.copy()

Return a shallow copy of the list. Similar to `a[:]`.

Un ejemplo que usa la mayoría de los métodos de la lista:

```
>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi
>>> fruits.count('apple')
2
>>> fruits.count('tangerine')
0
>>> fruits.index('banana')
3
```

```
>>> fruits.index('banana', 4)  # Find next banana starti
6
>>> fruits.reverse()
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', '
>>> fruits.append('grape')
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', '
>>> fruits.sort()
>>> fruits
['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi',
>>> fruits.pop()
'pear'
```

Quizás hayas notado que métodos como `insert`, `remove` o `sort` que únicamente modifican la lista no tienen un valor de retorno impreso – retornan el valor por defecto `None`. [1] Esto es un principio de diseño para todas las estructuras de datos mutables en Python.

Another thing you might notice is that not all data can be sorted or compared. For instance, `[None, 'hello', 10]` doesn't sort because integers can't be compared to strings and `None` can't be compared to other types. Also, there are some types that don't have a defined ordering relation. For example, `3+4j < 5+7j` isn't a valid comparison.

5.1.1. Usar listas como pilas

Los métodos de lista hacen que resulte muy fácil usar una lista como una pila, donde el último elemento añadido es el primer elemento retirado («último en entrar, primero en salir»). Para agregar un elemento a la cima de la pila, utiliza **`append()`**. Para retirar un elemento de la cima de la pila, utiliza **`pop()`** sin un índice explícito. Por ejemplo:

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
```

```
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

5.1.2. Usar listas como colas

También es posible usar una lista como una cola, donde el primer elemento añadido es el primer elemento retirado («primero en entrar, primero en salir»); sin embargo, las listas no son eficientes para este propósito. Agregar y sacar del final de la lista es rápido, pero insertar o sacar del comienzo de una lista es lento (porque todos los otros elementos tienen que ser desplazados en uno).

Para implementar una cola, utiliza `collections.deque` el cual fue diseñado para añadir y quitar de ambas puntas de forma rápida. Por ejemplo:

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")          # Graham arrives
>>> queue.popleft()                 # The first to arriv
'Eric'
>>> queue.popleft()                 # The second to arri
'John'
>>> queue                           # Remaining queue ir
deque(['Michael', 'Terry', 'Graham'])
```

5.1.3. Comprensión de listas

Las comprensiones de listas ofrecen una manera concisa de crear listas. Sus usos comunes son para hacer nuevas listas donde cada elemento es el resultado de algunas operaciones aplicadas a cada miembro de otra secuencia o iterable, o para crear un segmento de la secuencia de esos elementos para satisfacer una condición determinada.

Por ejemplo, asumamos que queremos crear una lista de cuadrados, como:

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Nótese que esto crea (o sobreescribe) una variable llamada `x` que sigue existiendo luego de que el bucle haya terminado. Podemos calcular la lista de cuadrados sin ningún efecto secundario haciendo:

```
squares = list(map(lambda x: x**2, range(10)))
```

o, un equivalente:

```
squares = [x**2 for x in range(10)]
```

que es más conciso y legible.

Una lista de comprensión consiste de corchetes rodeando una expresión seguida de la declaración **`for`** y luego cero o más declaraciones **`for`** o **`if`**. El resultado será una nueva lista que sale de evaluar la expresión en el contexto de los **`for`** o **`if`** que le siguen. Por ejemplo, esta lista de comprensión combina los elementos de dos listas si no son iguales:

```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

y es equivalente a:

```
>>> combs = []
>>> for x in [1,2,3]:
...     for y in [3,1,4]:
...         if x != y:
...             combs.append((x, y))
...
>>> combs
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Nótese como el orden de los `for` y `if` es el mismo en ambos pedacitos de código.

Si la expresión es una tupla (como el `(x, y)` en el ejemplo anterior), debe estar entre paréntesis.

```
>>> vec = [-4, -2, 0, 2, 4]
>>> # create a new list with the values doubled
>>> [x*2 for x in vec]
[-8, -4, 0, 4, 8]
>>> # filter the list to exclude negative numbers
>>> [x for x in vec if x >= 0]
[0, 2, 4]
>>> # apply a function to all the elements
>>> [abs(x) for x in vec]
[4, 2, 0, 2, 4]
>>> # call a method on each element
>>> freshfruit = ['  banana', '  loganberry ', 'passion
>>> [weapon.strip() for weapon in freshfruit]
['banana', 'loganberry', 'passion fruit']
>>> # create a list of 2-tuples like (number, square)
>>> [(x, x**2) for x in range(6)]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
>>> # the tuple must be parenthesized, otherwise an erro
>>> [x, x**2 for x in range(6)]
  File "<stdin>", line 1
    [x, x**2 for x in range(6)]
     ^^^^^^^
SyntaxError: did you forget parentheses around the compr
>>> # flatten a list using a listcomp with two 'for'
>>> vec = [[1,2,3], [4,5,6], [7,8,9]]
>>> [num for elem in vec for num in elem]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Las comprensiones de listas pueden contener expresiones complejas y funciones anidadas:

```
>>> from math import pi
>>> [str(round(pi, i)) for i in range(1, 6)]
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

5.1.4. Listas por comprensión anidadas

La expresión inicial de una comprensión de listas puede ser cualquier expresión arbitraria, incluyendo otra comprensión de listas.

Considerá el siguiente ejemplo de una matriz de 3x4 implementada como una lista de tres listas de largo 4:

```
>>> matrix = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
```

La siguiente comprensión de lista transpondrá las filas y columnas:

```
>>> [[row[i] for row in matrix] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Como vimos en la sección anterior, la lista de comprensión anidada se evalúa en el contexto del `for` que lo sigue, por lo que este ejemplo equivale a:

```
>>> transposed = []
>>> for i in range(4):
...     transposed.append([row[i] for row in matrix])
...
>>> transposed
```

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

el cual, a la vez, es lo mismo que:

```
>>> transposed = []
>>> for i in range(4):
...     # the following 3 lines implement the nested lis
...     transposed_row = []
...     for row in matrix:
...         transposed_row.append(row[i])
...     transposed.append(transposed_row)
...
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

En el mundo real, deberías preferir funciones predefinidas a declaraciones con flujo complejo. La función `zip()` haría un buen trabajo para este caso de uso:

```
>>> list(zip(*matrix))
[(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]
```

Ver Desempaquetando una lista de argumentos para detalles sobre el asterisco de esta línea.

5.2. La instrucción `del`

Hay una manera de quitar un ítem de una lista dado su índice en lugar de su valor: la instrucción `del`. Esta es diferente del método `pop()`, el cual retorna un valor. La instrucción `del` también puede usarse para quitar secciones de una lista o vaciar la lista completa (lo que hacíamos antes asignando una lista vacía a la rebanada). Por ejemplo:

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.25, 333, 333, 1234.5]
>>> del a[2:4]
>>> a
```

```
[1, 66.25, 1234.5]
>>> del a[:]
>>> a
[]
```

`del` puede usarse también para eliminar variables:

```
>>> del a
```

Hacer referencia al nombre `a` de aquí en más es un error (al menos hasta que se le asigne otro valor). Veremos otros usos para `del` más adelante.

5.3. Tuplas y secuencias

Vimos que las listas y cadenas tienen propiedades en común, como el indexado y las operaciones de rebanado. Estas son dos ejemplos de datos de tipo *secuencia* (ver Tipos secuencia — list, tuple, range). Como Python es un lenguaje en evolución, otros datos de tipo secuencia pueden agregarse. Existe otro dato de tipo secuencia estándar: la *tupla*.

Una tupla está formada por un número de valores separados por comas, por ejemplo:

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
>>> u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
>>> t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignme
```

```
>>> # but they can contain mutable objects:
>>> v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

Como puedes ver, en la salida las tuplas siempre se encierran entre paréntesis para que las tuplas anidadas puedan interpretarse correctamente; pueden ingresarse con o sin paréntesis, aunque a menudo los paréntesis son necesarios de todas formas (si la tupla es parte de una expresión más grande). No es posible asignar a los ítems individuales de una tupla, pero sin embargo sí se puede crear tuplas que contengan objetos mutables, como las listas.

A pesar de que las tuplas puedan parecerse a las listas, frecuentemente se utilizan en distintas situaciones y para distintos propósitos. Las tuplas son immutable y normalmente contienen una secuencia heterogénea de elementos que son accedidos al desempaquetar (ver más adelante en esta sección) o indizar (o incluso acceder por atributo en el caso de las **`namedtuples`**). Las listas son mutable, y sus elementos son normalmente homogéneos y se acceden iterando a la lista.

Un problema particular es la construcción de tuplas que contengan 0 o 1 ítem: la sintaxis presenta algunas peculiaridades para estos casos. Las tuplas vacías se construyen mediante un par de paréntesis vacío; una tupla con un ítem se construye poniendo una coma a continuación del valor (no alcanza con encerrar un único valor entre paréntesis). Feo, pero efectivo. Por ejemplo:

```
>>> empty = ()
>>> singleton = 'hello',    # <-- note trailing comma
>>> len(empty)
0
>>> len(singleton)
1
>>> singleton
('hello',)
```

La declaración `t = 12345, 54321, 'hola!'` es un ejemplo de *empaquetado de tuplas*: los valores `12345`, `54321` y `'hola!'` se empaquetan juntos en una tupla. La operación inversa también es posible:

```
>>> x, y, z = t
```

Esto se llama, apropiadamente, *desempaquetado de secuencias*, y funciona para cualquier secuencia en el lado derecho del igual. El desempaquetado de secuencias requiere que la cantidad de variables a la izquierda del signo igual sea el tamaño de la secuencia. Nótese que la asignación múltiple es en realidad sólo una combinación de empaquetado de tuplas y desempaquetado de secuencias.

5.4. Conjuntos

Python also includes a data type for sets. A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Las llaves o la función **`set()`** pueden usarse para crear conjuntos. Notá que para crear un conjunto vacío tenés que usar `set()`, no `{}`; esto último crea un diccionario vacío, una estructura de datos que discutiremos en la sección siguiente.

Because sets are unordered, iterating over them or printing them can produce the elements in a different order than you expect.

Una pequeña demostración:

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orang
>>> print(basket)                      # show that dupli
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket                 # fast membership
True
>>> 'crabgrass' in basket
False

>>> # Demonstrate set operations on unique letters from
```

```
>>>
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                  # unique letters
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                              # letters in a bu
{'r', 'd', 'b'}
>>> a | b                              # letters in a or
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                              # letters in both
{'a', 'c'}
>>> a ^ b                              # letters in a or
{'r', 'd', 'b', 'm', 'z', 'l'}
```

De forma similar a las comprensiones de listas, la comprensión de conjuntos está también soportada:

```
>>> a = {x for x in 'abracadabra' if x not in 'abc'}
>>> a
{'r', 'd'}
```

5.5. Diccionarios

Otro tipo de dato útil incluido en Python es el *diccionario* (ver Tipos mapa — dict). Los diccionarios se encuentran a veces en otros lenguajes como «memorias asociativas» o «arreglos asociativos». A diferencia de las secuencias, que se indexan mediante un rango numérico, los diccionarios se indexan con *claves*, que pueden ser cualquier tipo inmutable; las cadenas y números siempre pueden ser claves. Las tuplas pueden usarse como claves si solamente contienen cadenas, números o tuplas; si una tupla contiene cualquier objeto mutable directa o indirectamente, no puede usarse como clave. No podés usar listas como claves, ya que las listas pueden modificarse usando asignación por índice, asignación por sección, o métodos como **`append()`** y **`extend()`**.

Es mejor pensar en un diccionario como un conjunto de pares *clave:valor* con el requerimiento de que las claves sean únicas (dentro de un diccionario). Un par de llaves crean un diccionario vacío: `{}`. Colocar una lista de pares clave:valor separada por comas dentro de las llaves añade pares clave:valor iniciales al diccionario; esta es también la forma en que los diccionarios se muestran en la salida.

The main operations on a dictionary are storing a value with some key and extracting the value given the key. It is also possible to delete a key:value pair with `del`. If you store using a key that is already in use, the old value associated with that key is forgotten.

Extracting a value for a non-existent key by subscripting (`d[key]`) raises a **`KeyError`**. To avoid getting this error when trying to access a possibly non-existent key, use the **`get()`** method instead, which returns `None` (or a specified default value) if the key is not in the dictionary.

Ejecutando `list(d)` en un diccionario retornará una lista con todas las claves usadas en el diccionario, en el orden de inserción (si deseas que esté ordenada simplemente usa `sorted(d)` en su lugar). Para comprobar si una clave está en el diccionario usa la palabra clave **`in`**.

Un pequeño ejemplo de uso de un diccionario:

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> tel['jack']
4098
>>> tel['irv']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'irv'
>>> print(tel.get('irv'))
None
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'guido': 4127, 'irv': 4127}
```

```
>>> list(tel)
['jack', 'guido', 'irv']
>>> sorted(tel)
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

El constructor `dict()` crea un diccionario directamente desde secuencias de pares clave-valor:

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

Además, las comprensiones de diccionarios se pueden usar para crear diccionarios desde expresiones arbitrarias de clave y valor:

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

Cuando las claves son cadenas simples, a veces resulta más fácil especificar los pares usando argumentos por palabra clave:

```
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

5.6. Técnicas de iteración

Cuando iteramos sobre diccionarios, se pueden obtener al mismo tiempo la clave y su valor correspondiente usando el método `items()`.

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the bra
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad the pure
robin the brave
```

Cuando se itera sobre una secuencia, se puede obtener el índice de posición junto a su valor correspondiente usando la función `enumerate()`.

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
...
0 tic
1 tac
2 toe
```

Para iterar sobre dos o más secuencias al mismo tiempo, los valores pueden emparejarse con la función `zip()`.

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print('What is your {0}?  It is {1}.'.format(q,
...
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

Para iterar sobre una secuencia en orden inverso, se especifica primero la secuencia al derecho y luego se llama a la función `reversed()`.

```
>>> for i in reversed(range(1, 10, 2)):
...     print(i)
...
9
7
5
3
1
```

Para iterar sobre una secuencia ordenada, se utiliza la función `sorted()` la cual retorna una nueva lista ordenada dejando a la original intacta.

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orang
>>> for i in sorted(basket):
...     print(i)
...
apple
apple
banana
orange
orange
pear
```

El uso de `set()` en una secuencia elimina los elementos duplicados. El uso de `sorted()` en combinación con `set()` sobre una secuencia es una forma idiomática de recorrer elementos únicos de la secuencia ordenada.

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orang
>>> for f in sorted(set(basket)):
...     print(f)
...
apple
banana
orange
pear
```

A veces uno intenta cambiar una lista mientras la está iterando; sin embargo, a menudo es más simple y seguro crear una nueva lista:

```
>>> import math
>>> raw_data = [56.2, float('NaN'), 51.7, 55.3, 52.5, fl
>>> filtered_data = []
>>> for value in raw_data:
...     if not math.isnan(value):
...         filtered_data.append(value)
...
>>> filtered_data
[56.2, 51.7, 55.3, 52.5, 47.8]
```

5.7. Más acerca de condiciones

Las condiciones usadas en las instrucciones `while` e `if` pueden contener cualquier operador, no sólo comparaciones.

Los operadores de comparación `in` y `not in` verifican si un valor ocurre (o no ocurre) en una secuencia. Los operadores `is` e `is not` comparan si dos objetos son realmente el mismo objeto. Todos los operadores de comparación tienen la misma prioridad, que es menor que la de todos los operadores numéricos.

Las comparaciones pueden encadenarse. Por ejemplo, `a < b == c` verifica si `a` es menor que `b` y además si `b` es igual a `c`.

Las comparaciones pueden combinarse mediante los operadores booleanos `and` y `or`, y el resultado de una comparación (o de cualquier otra expresión booleana) puede negarse con `not`. Estos tienen prioridades menores que los operadores de comparación; entre ellos `not` tiene la mayor prioridad y `or` la menor, o sea que `A and not B or C` equivale a `(A and (not B)) or C`. Como siempre, los paréntesis pueden usarse para expresar la composición deseada.

Los operadores booleanos `and` y `or` son los llamados operadores *cortocircuito*: sus argumentos se evalúan de izquierda a derecha, y la evaluación se detiene en el momento en que se determina su resultado. Por ejemplo, si `A` y `C` son verdaderas pero `B` es falsa, en `A and B and C` no se evalúa la expresión `C`. Cuando se usa como un valor general y no como un booleano, el valor retornado de un operador cortocircuito es el último argumento evaluado.

Es posible asignar el resultado de una comparación u otra expresión booleana a una variable. Por ejemplo,

```
>>> string1, string2, string3 = '', 'Trondheim', 'Hammer
>>> non_null = string1 or string2 or string3
>>> non_null
'Trondheim'
```

Nótese que en Python, a diferencia de C, asignaciones dentro de expresiones deben realizarse explícitamente con el operador walrus `:=`. Esto soluciona algunos problemas comunes encontrados en C: escribiendo `=` en una expresión cuando se intentaba escribir `==`.

5.8. Comparando secuencias y otros tipos

Las secuencias pueden compararse con otros objetos del mismo tipo de secuencia. La comparación usa orden *lexicográfico*: primero se comparan los dos primeros ítems, si son diferentes esto ya determina el resultado de la comparación; si son iguales, se comparan los siguientes dos ítems, y así sucesivamente hasta llegar al final de alguna de las secuencias. Si dos ítems a comparar son ambos secuencias del mismo tipo, la comparación lexicográfica es recursiva. Si todos los ítems de dos secuencias resultan iguales, se considera que las secuencias son iguales. Si una secuencia es la parte inicial de la otra, la secuencia más corta es la más pequeña. El orden lexicográfico de las cadenas de caracteres utiliza el punto de código Unicode para ordenar caracteres individuales. Algunos ejemplos de comparación entre secuencias del mismo tipo:

```
(1, 2, 3)              < (1, 2, 4)
[1, 2, 3]              < [1, 2, 4]
'ABC' < 'C' < 'Pascal' < 'Python'
(1, 2, 3, 4)           < (1, 2, 4)
(1, 2)                 < (1, 2, -1)
(1, 2, 3)             == (1.0, 2.0, 3.0)
(1, 2, ('aa', 'ab'))   < (1, 2, ('abc', 'a'), 4)
```

Observá que comparar objetos de diferentes tipos con `<` o `>` es legal siempre y cuando los objetos tenga los métodos de comparación apropiados. Por ejemplo, los tipos de números mezclados son comparados de acuerdo a su valor numérico, o sea 0 es igual a 0.0, etc. Si no es el caso, en lugar de proveer un ordenamiento arbitrario, el intérprete lanzará una excepción **`TypeError`**.

Notas al pie

Otros lenguajes podrían retornar un objeto mutado, que permite encadenamiento de métodos como `d->insert("a")->remove("b")->sort();`.

[1]

6. Módulos

Si sales del intérprete de Python y vuelves a entrar, las definiciones que habías hecho (funciones y variables) se pierden. Por lo tanto, si quieres escribir un programa más o menos largo, es mejor que utilices un editor de texto para preparar la entrada para el intérprete y ejecutarlo con ese archivo como entrada. Esto se conoce como crear un *script*. A medida que tu programa crezca, quizás quieras separarlo en varios archivos para que el mantenimiento sea más sencillo. Quizás también quieras usar una función útil que has escrito en distintos programas sin copiar su definición en cada programa.

Para soportar esto, Python tiene una manera de poner definiciones en un archivo y usarlos en un script o en una instancia del intérprete. Este tipo de ficheros se llama *módulo*; las definiciones de un módulo pueden ser *importadas* a otros módulos o al módulo *principal* (la colección de variables a las que tienes acceso en un script ejecutado en el nivel superior y en el modo calculadora).

Un módulo es un fichero conteniendo definiciones y declaraciones de Python. El nombre de archivo es el nombre del módulo con el sufijo `.py` agregado. Dentro de un módulo, el nombre del mismo módulo (como cadena) está disponible en el valor de la variable global `__name__`. Por ejemplo, utiliza tu editor de texto favorito para crear un archivo llamado `fibo.py` en el directorio actual, con el siguiente contenido:

```
# Fibonacci numbers module

def fib(n):
    """Write Fibonacci series up to n."""
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):
    """Return Fibonacci series up to n."""
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

Ahora entra en el intérprete de Python e importa este modulo con el siguiente comando:

```
>>> import fibo
```

Esto no añade los nombres de las funciones definidas en `fibo` directamente al actual namespace (ver Ámbitos y espacios de nombres en Python para más detalles); sólo añade el nombre del módulo `fibo` allí. Usando el nombre del módulo puedes acceder a las funciones:

```
>>> fibo.fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

Si pretendes utilizar una función frecuentemente puedes asignarla a un nombre local:

```
>>> fib = fibo.fib
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

6.1. Más sobre los módulos

Un módulo puede contener tanto declaraciones ejecutables como definiciones de funciones. Estas declaraciones están pensadas para inicializar el módulo. Se ejecutan únicamente la *primera* vez que el módulo se encuentra en una declaración import. [1] (También se ejecutan si el archivo se ejecuta como script.)

Cada módulo tiene su propio espacio de nombres privado, que es utilizado como espacio de nombres global por todas las funciones definidas en el módulo. De este modo, el autor de un módulo puede utilizar variables globales en el módulo sin preocuparse por choques accidentales con las variables globales de un usuario. Por otro lado, si sabes lo que estás haciendo puedes tocar las variables globales de un módulo con la misma notación que se utiliza para referirse a sus funciones, `modname.itemname`.

Los módulos pueden importar otros módulos. Es costumbre pero no obligatorio ubicar todas las declaraciones **`import`** al principio del módulo (o script, para el caso). Los nombres de los módulos importados, si se colocan en el nivel superior de un módulo (fuera de cualquier función o clase), se añaden al espacio de nombres global del módulo.

Hay una variante de la declaración **`import`** que importa los nombres de un módulo directamente al espacio de nombres del módulo que hace la importación. Por ejemplo:

```
>>> from fibo import fib, fib2
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Esto no introduce en el espacio de nombres local el nombre del módulo desde el cual se está importando (por lo tanto, en el ejemplo, `fibo` no esta definido).

Hay incluso una variante para importar todos los nombres que un módulo define:

```
>>> from fibo import *
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Esto importa todos los nombres excepto los que inician con un guión bajo (_). La mayoría de las veces los programadores de

Python no usan esto ya que introduce en el intérprete un conjunto de nombres desconocido, posiblemente escondiendo algunas de las definiciones previas.

Nótese que en general la práctica de importar `*` de un módulo o paquete está muy mal vista, ya que frecuentemente genera código poco legible. Sin embargo, está bien usarlo para ahorrar tecleo en sesiones interactivas.

Si el nombre del módulo es seguido por **`as`**, el nombre siguiendo **`as`** queda ligado directamente al módulo importado.

```
>>> import fibo as fib
>>> fib.fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Esto es básicamente importar el módulo de la misma forma que se haría con `import fibo`, con la única diferencia en que se encuentra accesible como `fib`.

También se puede utilizar cuando se utiliza **`from`** con efectos similares:

```
>>> from fibo import fib as fibonacci
>>> fibonacci(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Nota

Por razones de eficiencia, cada módulo es importado solo una vez por sesión del intérprete. Por lo tanto, si cambias tus módulos, debes reiniciar el interprete – ó, si es un solo módulo que quieres probar de forma interactiva, usa **`importlib.reload()`**, por ejemplo: `import importlib; importlib.reload(modulename)`.

6.1.1. Ejecutar módulos como scripts

Cuando ejecutes un módulo de Python con

```
python fibo.py <arguments>
```

el código en el módulo será ejecutado, tal como si lo hubieses importado, pero con `__name__` con el valor de `"__main__"`. Eso significa que agregando este código al final de tu módulo:

```
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

puedes hacer que el archivo sea utilizable tanto como script, como módulo importable, porque el código que analiza la línea de órdenes sólo se ejecuta si el módulo es ejecutado como archivo principal:

```
$ python fibo.py 50
0 1 1 2 3 5 8 13 21 34
```

Si el módulo se importa, ese código no se ejecuta:

```
>>> import fibo
>>>
```

Esto es frecuentemente usado para proveer al módulo una interfaz de usuario conveniente, o para fines de prueba (ejecutar el módulo como un script que ejecuta un conjunto de pruebas).

6.1.2. El camino de búsqueda de los módulos

Cuando se importa un módulo llamado **`spam`**, el intérprete busca primero por un módulo con ese nombre que esté integrado en el intérprete. Estos nombres de módulos están listados en **`sys.builtin_module_names`**. Si no lo encuentra, entonces busca un archivo llamado `spam.py` en una lista de directorios especificada por la variable **`sys.path`**. **`sys.path`** se inicializa con las siguientes ubicaciones:

- El directorio que contiene el script de entrada (o el directorio actual cuando no se especifica archivo).
- **`PYTHONPATH`** (una lista de nombres de directorios, con la misma sintaxis que la variable de la terminal **`PATH`**).

- El valor predeterminado dependiente de la instalación (por convención incluye un directorio `site-packages`, manejado por el módulo **`site`**).

Más detalles en La inicialización de la ruta de búsqueda de módulo de sys.path.

Nota

En los sistemas de archivo que soportan enlaces simbólicos, el directorio que contiene el script de entrada es calculado luego de seguir el enlace simbólico. En otras palabras, el directorio que contiene el enlace simbólico **no** es agregado al camino de búsqueda del módulo.

Luego de la inicialización, los programas Python pueden modificar **`sys.path`**. El directorio que contiene el script que se está ejecutando se ubica al principio de la búsqueda, adelante de la biblioteca estándar. Esto significa que se cargarán scripts en ese directorio en lugar de módulos de la biblioteca estándar con el mismo nombre. Esto es un error a menos que se esté reemplazando intencionalmente. Mirá la sección Módulos estándar para más información.

6.1.3. Archivos «compilados» de Python

Para acelerar la carga de módulos, Python cachea las versiones compiladas de cada módulo en el directorio `__pycache__` bajo el nombre `module.version.pyc`, dónde la versión codifica el formato del archivo compilado; generalmente contiene el número de versión de Python. Por ejemplo, en CPython *release* 3.3 la versión compilada de spam.py sería cacheada como `__pycache__/spam.cpython-33.pyc`. Este convención de nombre permite compilar módulos desde diferentes *releases* y versiones de Python para coexistir.

Python chequea la fecha de modificación de la fuente contra la versión compilada para ver si esta es obsoleta y necesita ser recompilada. Esto es un proceso completamente automático. También, los módulos compilados son independientes de la plataforma, así que la misma biblioteca puede ser compartida a través de sistemas con diferentes arquitecturas.

Python no chequea el caché en dos circunstancias. Primero, siempre recompila y no graba el resultado del módulo que es cargado directamente desde la línea de comando. Segundo, no chequea el caché si no hay módulo fuente. Para soportar una distribución sin fuente (solo compilada), el módulo compilado debe estar en el directorio origen, y no debe haber un módulo fuente.

Algunos consejos para expertos:

- Puedes usar los modificadores **`-O`** o **`-OO`** en el comando de Python para reducir el tamaño del módulo compilado. El modificador `-O` remueve las declaraciones *assert*, el modificador `-OO` remueve declaraciones *assert* y cadenas _doc_. Dado que algunos programas pueden confiar en tenerlos disponibles, solo deberías usar esta opción si conoces lo que estás haciendo. Los módulos «optimizados» tienen una etiqueta `opt-` y generalmente son mas pequeños. *Releases* futuras pueden cambiar los efectos de la optimización.
- Un programa no se ejecuta mas rápido cuando es leído de un archivo `.pyc` que cuando es leído de un archivo `.py`; la única cosa que es mas rápida en los archivos `.pyc` es la velocidad con la cual son cargados.
- El módulo **`compileall`** puede crear archivos .pyc para todos los módulos en un directorio.
- Hay mas detalle de este proceso, incluyendo un diagrama de flujo de decisiones, en **PEP 3147** [https://peps.python.org/pep-3147/].

6.2. Módulos estándar

Python viene con una biblioteca de módulos estándar, descrita en un documento separado, la Referencia de la Biblioteca de Python (de aquí en más, «Referencia de la Biblioteca»). Algunos módulos se integran en el intérprete; estos proveen acceso a operaciones que no son parte del núcleo del lenguaje pero que sin embargo están integrados, tanto por eficiencia como para proveer acceso a primitivas del sistema operativo, como llamadas al sistema. El conjunto de tales módulos es una opción de configuración que también depende de la plataforma subyacente. Por ejemplo, el módulo **`winreg`** sólo se provee en sistemas Windows. Un módulo en particular merece algo de atención: **`sys`**, el que está integrado en todos los intérpretes de Python. Las variables `sys.ps1` y `sys.ps2` definen las cadenas usadas como cursores primarios y secundarios:

```
>>> import sys
>>> sys.ps1
'>>> '
>>> sys.ps2
'... '
>>> sys.ps1 = 'C> '
C> print('Yuck!')
Yuck!
C>
```

Estas dos variables están solamente definidas si el intérprete está en modo interactivo.

La variable `sys.path` es una lista de cadenas que determinan el camino de búsqueda del intérprete para los módulos. Se inicializa por omisión a un camino tomado de la variable de entorno **`PYTHONPATH`**, o a un valor predefinido en el intérprete si **`PYTHONPATH`** no está configurada. Lo puedes modificar usando las operaciones estándar de listas:

```
>>> import sys
>>> sys.path.append('/ufs/guido/lib/python')
```

6.3. La función `dir()`

La función integrada **`dir()`** se usa para encontrar qué nombres define un módulo. Retorna una lista ordenada de cadenas:

```
>>> import fibo, sys
>>> dir(fibo)
['__name__', 'fib', 'fib2']
>>> dir(sys)
```

```
['__breakpointhook__', '__displayhook__', '__doc__', '__
 '__interactivehook__', '__loader__', '__name__', '__pac
 '__stderr__', '__stdin__', '__stdout__', '__unraisableh
 '_clear_type_cache', '_current_frames', '_debugmallocst
 '_getframe', '_git', '_home', '_xoptions', 'abiflags',
 'api_version', 'argv', 'audit', 'base_exec_prefix', 'ba
 'breakpointhook', 'builtin_module_names', 'byteorder',
 'callstats', 'copyright', 'displayhook', 'dont_write_by
 'excepthook', 'exec_prefix', 'executable', 'exit', 'fla
 'float_repr_style', 'get_asyncgen_hooks', 'get_corouti
 'getallocatedblocks', 'getdefaultencoding', 'getdlopenf
 'getfilesystemencodeerrors', 'getfilesystemencoding', '
 'getrecursionlimit', 'getrefcount', 'getsizeof', 'getsw
 'gettrace', 'hash_info', 'hexversion', 'implementation'
 'intern', 'is_finalizing', 'last_traceback', 'last_type
 'maxsize', 'maxunicode', 'meta_path', 'modules', 'path'
 'path_importer_cache', 'platform', 'prefix', 'ps1', 'ps
 'set_asyncgen_hooks', 'set_coroutine_origin_tracking_de
 'setprofile', 'setrecursionlimit', 'setswitchinterval',
 'stdin', 'stdout', 'thread_info', 'unraisablehook', 've
 'warnoptions']
```

Sin argumentos, `dir()` lista los nombres que tienes actualmente definidos:

```
>>> a = [1, 2, 3, 4, 5]
>>> import fibo
>>> fib = fibo.fib
>>> dir()
['__builtins__', '__name__', 'a', 'fib', 'fibo', 'sys']
```

Nótese que lista todos los tipos de nombres: variables, módulos, funciones, etc.

`dir()` no lista los nombres de las funciones y variables integradas. Si quieres una lista de esos, están definidos en el módulo estándar `builtins`:

```
>>> import builtins
>>> dir(builtins)
```

```
['ArithmeticError', 'AssertionError', 'AttributeError',
 'BlockingIOError', 'BrokenPipeError', 'BufferError', '
 'ChildProcessError', 'ConnectionAbortedError', 'Connect
 'ConnectionRefusedError', 'ConnectionResetError', 'Depr
 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception'
 'FileExistsError', 'FileNotFoundError', 'FloatingPoint
 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportErr
 'ImportWarning', 'IndentationError', 'IndexError', 'Int
 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', '
 'MemoryError', 'NameError', 'None', 'NotADirectoryError
 'NotImplementedError', 'OSError', 'OverflowError',
 'PendingDeprecationWarning', 'PermissionError', 'Proces
 'ReferenceError', 'ResourceWarning', 'RuntimeError', '
 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'Syste
 'SystemExit', 'TabError', 'TimeoutError', 'True', 'Type
 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEnc
 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarni
 'ValueError', 'Warning', 'ZeroDivisionError', '_', '__b
 '__debug__', '__doc__', '__import__', '__name__', '__pa
 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'byt
 'chr', 'classmethod', 'compile', 'complex', 'copyright'
 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval'
 'filter', 'float', 'format', 'frozenset', 'getattr', '
 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstan
 'iter', 'len', 'license', 'list', 'locals', 'map', 'ma
 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', '
 'quit', 'range', 'repr', 'reversed', 'round', 'set', '
 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple
 'zip']
```

6.4. Paquetes

Los Paquetes son una forma de estructurar el espacio de nombres de módulos de Python usando «nombres de módulo con puntos». Por ejemplo, el nombre del módulo **`A.B`** designa un submódulo `B` en un paquete llamado `A`. Así como el uso de módulos salva a los autores de diferentes módulos de tener que preocuparse por los nombres de las variables globales de los demás, el uso de nombres de módulo con puntos evita que los autores de paquetes multimódulos, como NumPy o Pillow, tengan que preocuparse por los nombres de los módulos de los demás.

Supongamos que quieres designar una colección de módulos (un «paquete») para el manejo uniforme de archivos y datos de sonidos. Hay diferentes formatos de archivos de sonido (normalmente reconocidos por su extensión, por ejemplo: `.wav`, `.aiff`, `.au`), por lo que tienes que crear y mantener una colección siempre creciente de módulos para la conversión entre los distintos formatos de archivos. Hay muchas operaciones diferentes que quizás quieras ejecutar en los datos de sonido (como mezclarlos, añadir eco, aplicar una función ecualizadora, crear un efecto estéreo artificial), por lo que además estarás escribiendo una lista sin fin de módulos para realizar estas operaciones. Aquí hay una posible estructura para tu paquete (expresados en términos de un sistema jerárquico de archivos):

```
sound/                          Top-level package
      __init__.py               Initialize the sound pac
      formats/                  Subpackage for file form
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound eff
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

Al importar el paquete, Python busca a través de los directorios en `sys.path`, buscando el sub-directorio del paquete.

The `__init__.py` files are required to make Python treat directories containing the file as packages (unless using a namespace package, a relatively advanced feature). This prevents directories with a common name, such as `string`, from unintentionally hiding valid modules that occur later on the module search path. In the simplest case, `__init__.py` can just be an empty file, but it can also execute initialization code for the package or set the `__all__` variable, described later.

Los usuarios del paquete pueden importar módulos individuales del mismo, por ejemplo:

```
import sound.effects.echo
```

Esto carga el submódulo **`sound.effects.echo`**. Debe hacerse referencia al mismo con el nombre completo.

```
sound.effects.echo.echofilter(input, output, delay=0.7,
```

Otra alternativa para importar el submódulo es:

```
from sound.effects import echo
```

Esto también carga el submódulo **`echo`**, y lo deja disponible sin su prefijo de paquete, por lo que puede usarse así:

```
echo.echofilter(input, output, delay=0.7, atten=4)
```

Otra variación más es importar la función o variable deseadas directamente:

```
from sound.effects.echo import echofilter
```

De nuevo, esto carga el submódulo **`echo`**, pero deja directamente disponible a la función **`echofilter()`**:

```
echofilter(input, output, delay=0.7, atten=4)
```

Nótese que al usar `from package import item`, el ítem puede ser tanto un submódulo (o subpaquete) del paquete, o algún otro nombre definido en el paquete, como una función, clase, o variable. La declaración `import` primero verifica si el ítem está definido en el paquete; si no, asume que es un módulo y trata de cargarlo. Si no lo puede encontrar, se genera una excepción **`ImportError`**.

Por otro lado, cuando se usa la sintaxis como `import item.subitem.subsubitem`, cada ítem excepto el último debe ser un paquete; el mismo puede ser un módulo o un paquete pero no puede ser una clase, función o variable definida en el ítem previo.

6.4.1. Importar * desde un paquete

Ahora, ¿qué sucede cuando el usuario escribe `from sound.effects import *`? Idealmente, uno esperaría que esto de alguna manera vaya al sistema de archivos, encuentre cuales submódulos están presentes en el paquete, y los importe a todos. Esto puede tardar mucho y el importar sub-módulos puede tener efectos secundarios no deseados que sólo deberían ocurrir cuando se importe explícitamente el sub-módulo.

La única solución es que el autor del paquete provea un índice explícito del paquete. La declaración **`import`** usa la siguiente convención: si el código del `__init__.py` de un paquete define una lista llamada `__all__`, se toma como la lista de los nombres de módulos que deberían ser importados cuando se hace `from package import *`. Es tarea del autor del paquete mantener actualizada esta lista cuando se libera una nueva versión del paquete. Los autores de paquetes podrían decidir no soportarlo, si no ven un uso para importar * en sus paquetes. Por ejemplo, el archivo `sound/effects/__init__.py` podría contener el siguiente código:

```
__all__ = ["echo", "surround", "reverse"]
```

Esto significaría que `from sound.effects import *` importaría esos tres submódulos del paquete **`sound.effects`**.

Ten en cuenta que los submódulos pueden quedar ocultos por nombres definidos localmente. Por ejemplo, si agregaste una función llamada `reverse` al archivo `sound/effects/__init__.py`, `from sound.effects import *` solo importaría los dos submódulos `echo` y `surround`, pero *no* el submódulo `reverse` porque queda oculto por la función `reverse` definida localmente:

```
__all__ = [
    "echo",      # refers to the 'echo.py' file
    "surround",  # refers to the 'surround.py' file
    "reverse",   # !!! refers to the 'reverse' function
]

def reverse(msg: str):  # <-- this name shadows the 'rev
    return msg[::-1]    #     in the case of a 'from sou
```

Si no se define `__all__`, la declaración `from sound.effects import *` *no* importa todos los submódulos del paquete **`sound.effects`** al espacio de nombres actual; sólo se asegura que se haya importado el paquete **`sound.effects`** (posiblemente ejecutando algún código de inicialización que haya en `__init__.py`) y luego importa aquellos nombres que estén definidos en el paquete. Esto incluye cualquier nombre definido (y submódulos explícitamente cargados) por `__init__.py`. También incluye cualquier submódulo del paquete que pudiera haber sido explícitamente cargado por declaraciones **`import`** previas. Considere este código:

```
import sound.effects.echo
import sound.effects.surround
from sound.effects import *
```

En este ejemplo, los módulos **`echo`** y **`surround`** se importan en el espacio de nombre actual porque están definidos en el paquete **`sound.effects`** cuando se ejecuta la declaración `from...import`. (Esto también funciona cuando se define `__all__`).

A pesar de que ciertos módulos están diseñados para exportar solo nombres que siguen ciertos patrones cuando uses `import *`, también se considera una mala práctica en código de producción.

Recuerda, ¡no hay nada malo al usar `from package import specific_submodule`! De hecho, esta es la notación recomendada a menos que el módulo que importamos necesite usar submódulos con el mismo nombre desde un paquete diferente.

6.4.2. Referencias internas en paquetes

Cuando se estructuran los paquetes en sub-paquetes (como en el ejemplo **`sound`**), puedes usar imports absolutos para referirte a submódulos de paquetes hermanos. Por ejemplo, si el módulo **`sound.filters.vocoder`** necesita usar el módulo **`echo`** en el paquete **`sound.effects`**, puede hacer `from sound.effects import echo`.

También puedes escribir imports relativos con la forma `from module import name`. Estos imports usan puntos adelante para indicar los paquetes actuales o paquetes padres involucrados en el import relativo. En el ejemplo **`surround`**, podrías hacer:

```
from . import echo
from .. import formats
from ..filters import equalizer
```

Note that relative imports are based on the name of the current module's package. Since the main module does not have a package, modules intended for use as the main module of a Python application must always use absolute imports.

6.4.3. Paquetes en múltiples directorios

Packages support one more special attribute, **`__path__`**. This is initialized to be a sequence of strings containing the name of the directory holding the package's `__init__.py` before the code in that file is executed. This variable can be modified; doing so affects future searches for modules and subpackages contained in the package.

Aunque esta característica no se necesita frecuentemente, puede usarse para extender el conjunto de módulos que se encuentran en el paquete.

Notas al pie

De hecho, las definiciones de funciones también son «declaraciones» que se «ejecutan»; la ejecución de una definición de función a nivel de módulo, añade el nombre de la función en el espacio de nombres global del módulo.

7. Entrada y salida

Hay diferentes métodos de presentar la salida de un programa; los datos pueden ser impresos de una forma legible por humanos, o escritos a un archivo para uso futuro. Este capítulo discutirá algunas de las posibilidades.

7.1. Formateo elegante de la salida

Hasta ahora encontramos dos maneras de escribir valores: *declaraciones de expresión* y la función `print()`. (Una tercera manera es usando el método `write()` de los objetos tipo archivo; el archivo de salida estándar puede referenciarse como `sys.stdout`. Mirá la Referencia de la Biblioteca para más información sobre esto).

A menudo se querrá tener más control sobre el formato de la salida, y no simplemente imprimir valores separados por espacios. Para ello, hay varias maneras de dar formato a la salida.

- Para usar literales de cadena formateados, comience una cadena con `f` o `F` antes de la comilla de apertura o comillas triples. Dentro de esta cadena, se puede escribir una expresión de Python entre los caracteres `{` y `}` que pueden hacer referencia a variables o valores literales.

  ```
  >>> year = 2016
  >>> event = 'Referendum'
  >>> f'Results of the {year} {event}'
  'Results of the 2016 Referendum'
  ```

- The `str.format()` method of strings requires more manual effort. You’ll still use `{` and `}` to mark where a variable will be substituted and can provide detailed formatting directives, but you’ll also need to provide the information to be formatted. In the following code block there are two examples of how to format variables:

```
>>> yes_votes = 42_572_654
>>> total_votes = 85_705_149
>>> percentage = yes_votes / total_votes
>>> '{:-9} YES votes  {:2.2%}'.format(yes_votes, pe
' 42572654 YES votes  49.67%'
```

Notice how the `yes_votes` are padded with spaces and a negative sign only for negative numbers. The example also prints `percentage` multiplied by 100, with 2 decimal places and followed by a percent sign (see Format specification mini-language for details).

- Por último, puede realizar todo el control de cadenas usted mismo mediante operaciones de concatenación y segmentación de cadenas para crear cualquier diseño que se pueda imaginar. El tipo de cadena tiene algunos métodos que realizan operaciones útiles para rellenar cadenas a un ancho de columna determinado.

Cuando no necesita una salida elegante, pero solo desea una visualización rápida de algunas variables con fines de depuración, puede convertir cualquier valor en una cadena con las funciones `repr()` o `str()`.

La función `str()` retorna representaciones de los valores que son bastante legibles por humanos, mientras que `repr()` genera representaciones que pueden ser leídas por el intérprete (o forzarían un `SyntaxError` si no hay sintaxis equivalente). Para objetos que no tienen una representación en particular para consumo humano, `str()` retornará el mismo valor que `repr()`. Muchos valores, como números o estructuras como listas y diccionarios, tienen la misma representación usando cualquiera de las dos funciones. Las cadenas, en particular, tienen dos representaciones distintas.

Algunos ejemplos:

```
>>> s = 'Hello, world.'
>>> str(s)
'Hello, world.'
```

```
>>> repr(s)
"'Hello, world.'"
>>> str(1/7)
'0.14285714285714285'
>>> x = 10 * 3.25
>>> y = 200 * 200
>>> s = 'The value of x is ' + repr(x) + ', and y is ' +
>>> print(s)
The value of x is 32.5, and y is 40000...
>>> # The repr() of a string adds string quotes and back
>>> hello = 'hello, world\n'
>>> hellos = repr(hello)
>>> print(hellos)
'hello, world\n'
>>> # The argument to repr() may be any Python object:
>>> repr((x, y, ('spam', 'eggs')))
"(32.5, 40000, ('spam', 'eggs'))"
```

The **`string`** module contains support for a simple templating approach based upon regular expressions, via **`string.Template`**. This offers yet another way to substitute values into strings, using placeholders like `$x` and replacing them with values from a dictionary. This syntax is easy to use, although it offers much less control for formatting.

7.1.1. Formatear cadenas literales

Literales de cadena formateados (también llamados f-strings para abreviar) le permiten incluir el valor de las expresiones de Python dentro de una cadena prefijando la cadena con `f` o `F` y escribiendo expresiones como `{expresion}`.

La expresión puede ir seguida de un especificador de formato opcional . Esto permite un mayor control sobre cómo se formatea el valor. En el ejemplo siguiente se redondea pi a tres lugares después del decimal:

```
>>> import math
>>> print(f'The value of pi is approximately {math.pi:.
The value of pi is approximately 3.142.
```

Pasar un entero después de `':'` hará que ese campo sea un número mínimo de caracteres de ancho. Esto es útil para hacer que las columnas se alineen.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
>>> for name, phone in table.items():
...     print(f'{name:10} ==> {phone:10d}')
...
Sjoerd     ==>       4127
Jack       ==>       4098
Dcab       ==>       7678
```

Se pueden utilizar otros modificadores para convertir el valor antes de formatearlo. `'!a'` se aplica `ascii()`, `'!s'` se aplica `str()`, y `'!r'` se aplica `repr()`:

```
>>> animals = 'eels'
>>> print(f'My hovercraft is full of {animals}.')
My hovercraft is full of eels.
>>> print(f'My hovercraft is full of {animals!r}.')
My hovercraft is full of 'eels'.
```

El especificador `=` puede utilizarse para expandir una expresión al texto de la expresión, un signo igual y, a continuación, la representación de la expresión evaluada:

```
>>> bugs = 'roaches'
>>> count = 13
>>> area = 'living room'
>>> print(f'Debugging {bugs=} {count=} {area=}')
Debugging bugs='roaches' count=13 area='living room'
```

Véase expresiones auto-documentadas para más información en el especificador `=`. Para obtener una referencia sobre estas especificaciones de formato, consulte la guía de referencia para Format specification mini-language.

7.1.2. El método format() de cadenas

El uso básico del método `str.format()` es como esto:

```
>>> print('We are the {} who say "{}!"'.format('knights'
We are the knights who say "Ni!"
```

Las llaves y caracteres dentro de las mismas (llamados campos de formato) son reemplazadas con los objetos pasados en el método **`str.format()`**. Un número en las llaves se refiere a la posición del objeto pasado en el método **`str.format()`**.

```
>>> print('{0} and {1}'.format('spam', 'eggs'))
spam and eggs
>>> print('{1} and {0}'.format('spam', 'eggs'))
eggs and spam
```

Si se usan argumentos nombrados en el método **`str.format()`**, sus valores se referencian usando el nombre del argumento.

```
>>> print('This {food} is {adjective}.'.format(
...       food='spam', adjective='absolutely horrible'))
This spam is absolutely horrible.
```

Se pueden combinar arbitrariamente argumentos posicionales y nombrados:

```
>>> print('The story of {0}, {1}, and {other}.'.format('
...
The story of Bill, Manfred, and Georg.
```

Si tiene una cadena de caracteres de formato realmente larga que no desea dividir, sería bueno si pudiera hacer referencia a las variables que se formatearán por nombre en lugar de por posición. Esto se puede hacer simplemente pasando el diccionario y usando corchetes `'[]'` para acceder a las claves.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637
>>> print('Jack: {0[Jack]:d}; Sjoerd: {0[Sjoerd]:d}; '
...       'Dcab: {0[Dcab]:d}'.format(table))
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
```

Esto se podría hacer, también, pasando el diccionario `table` como argumentos por palabra clave con la notación “**”.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 86376
>>> print('Jack: {Jack:d}; Sjoerd: {Sjoerd:d}; Dcab: {Dc
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
```

This is particularly useful in combination with the built-in function **`vars()`**, which returns a dictionary containing all local variables:

```
>>> table = {k: str(v) for k, v in vars().items()}
>>> message = " ".join([f'{k}: ' + '{' + k +'};' for k i
>>> print(message.format(**table))
__name__: __main__; __doc__: None; __package__: None; __
```

Como ejemplo, las siguientes líneas producen un conjunto ordenado de columnas que dan enteros y sus cuadrados y cubos:

```
>>> for x in range(1, 11):
...     print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*
...
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

Para una completa descripción del formateo de cadenas con **`str.format()`**, ver Custom string formatting.

7.1.3. Formateo manual de cadenas

Aquí está la misma tabla de cuadrados y cubos, formateados manualmente:

```
>>> for x in range(1, 11):
...     print(repr(x).rjust(2), repr(x*x).rjust(3), end=
...     # Note use of 'end' on previous line
...     print(repr(x*x*x).rjust(4))
...
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

(Nótese que el espacio existente entre cada columna es añadido debido a como funciona `print()`: siempre añade espacios entre sus argumentos.)

El método `str.rjust()` de los objetos cadena justifica a la derecha en un campo de anchura predeterminada rellenando con espacios a la izquierda. Métodos similares a este son `str.ljust()` y `str.center()`. Estos métodos no escriben nada, simplemente retornan una nueva cadena. Si la cadena de entrada es demasiado larga no la truncarán sino que la retornarán sin cambios; esto desordenará la disposición de la columna que es, normalmente, mejor que la alternativa, la cual podría falsear un valor. (Si realmente deseas truncar siempre puedes añadir una operación de rebanado, como en `x.ljust(n)[:n]`.)

Hay otro método, `str.zfill()`, el cual rellena una cadena numérica a la izquierda con ceros. Entiende signos positivos y negativos:

```
>>> '12'.zfill(5)
'00012'
>>> '-3.14'.zfill(7)
'-003.14'
>>> '3.14159265359'.zfill(5)
'3.14159265359'
```

7.1.4. Viejo formateo de cadenas

The % operator (modulo) can also be used for string formatting. Given `format % values` (where *format* is a string), `%` conversion specifications in *format* are replaced with zero or more elements of *values*. This operation is commonly known as string interpolation. For example:

```
>>> import math
>>> print('The value of pi is approximately %5.3f.' % ma
The value of pi is approximately 3.142.
```

Podés encontrar más información en la sección Formateo de cadenas al estilo *printf*.

7.2. Leyendo y escribiendo archivos

La función **`open()`** retorna un file object, y se usa normalmente con dos argumentos posicionales y un argumento nombrado: `open(nombre_de_archivo, modo, encoding=None)`

```
>>> f = open('workfile', 'w', encoding="utf-8")
```

El primer argumento es una cadena que contiene el nombre del fichero. El segundo argumento es otra cadena que contiene unos pocos caracteres describiendo la forma en que el fichero será usado. *mode* puede ser `'r'` cuando el fichero solo se leerá, `'w'` para solo escritura (un fichero existente con el mismo nombre se borrará) y `'a'` abre el fichero para agregar; cualquier dato que se escribe en el fichero se añade automáticamente al final. `'r+'` abre el fichero tanto para lectura como para escritura. El argumento *mode* es opcional; se asume que se usará `'r'` si se omite.

Normalmente, los ficheros se abren en *modo texto*, es decir, se leen y escriben cadenas desde y hacia el fichero, que están codificadas en una *codificación* específica. Si no se especifica *codificación*, el valor por defecto depende de la plataforma (véase **`open()`**). Dado que UTF-8 es el estándar moderno de facto, se recomienda `encoding="utf-8"` a menos que sepa que necesita usar una codificación diferente. Añadiendo `'b'` al modo se abre el fichero en *modo binario*. Los datos en modo binario se leen y escriben como objetos **`bytes`**. No se puede especificar *codificación* al abrir un fichero en modo binario.

Cuando se lee en modo texto, por defecto se convierten los fin de lineas que son específicos a las plataformas (`\n` en Unix, `\r\n` en Windows) a solamente `\n`. Cuando se escribe en modo texto, por defecto se convierten los `\n` a los fin de linea específicos de la plataforma. Este cambio automático está bien para archivos de texto, pero corrompería datos binarios como los de archivos `JPEG` o `EXE`. Asegúrese de usar modo binario cuando lea y escriba tales archivos.

Es una buena práctica usar la declaración `with` cuando manejamos objetos archivo. Tiene la ventaja de que el archivo es cerrado apropiadamente luego de que el bloque termina, incluso si se generó una excepción. También es mucho más corto que escribir los equivalentes bloques `try`-`finally`

```
>>> with open('workfile', encoding="utf-8") as f:
...     read_data = f.read()

>>> # We can check that the file has been automatically
>>> f.closed
True
```

Si no está utilizando la palabra clave `with`, entonces debe llamar a `f.close()` para cerrar el archivo y liberar inmediatamente los recursos del sistema utilizados por él.

Advertencia

Llamar a `f.write()` sin usar la palabra clave **`with`** o sin llamar a `f.close()` **podría** dar como resultado que los argumentos de `f.write()` no se escriban completamente en disco, incluso si el programa se termina correctamente.

Después de que un objeto de archivo es cerrado, ya sea por `with` o llamando a `f.close()`, intentar volver a utilizarlo fallará automáticamente:

```
>>> f.close()
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

7.2.1. Métodos de los objetos Archivo

El resto de los ejemplos en esta sección asumirán que ya se creó un objeto archivo llamado `f`.

Para leer el contenido de una archivo utiliza `f.read(size)`, el cual lee alguna cantidad de datos y los retorna como una cadena (en modo texto) o un objeto de bytes (en modo binario). *size* es un argumento numérico opcional. Cuando se omite *size* o es negativo, el contenido entero del archivo será leído y retornado; es tu problema si el archivo es el doble de grande que la memoria de tu máquina. De otra manera, son leídos y retornados como máximo *size* caracteres (en modo texto) o *size* bytes (en modo binario). Si se alcanzó el fin del archivo, `f.read()` retornará una cadena vacía (`''`).

```
>>> f.read()
'This is the entire file.\n'
>>> f.read()
''
```

`f.readline()` lee una sola linea del archivo; el carácter de fin de linea (`\n`) se deja al final de la cadena, y sólo se omite en la última linea del archivo si el mismo no termina en un fin de linea. Esto hace que el valor de retorno no sea ambiguo; si `f.readline()` retorna una cadena vacía, es que se alcanzó el fin del archivo, mientras que una linea en blanco es representada por `'\n'`, una cadena conteniendo sólo un único fin de linea.

```
>>> f.readline()
'This is the first line of the file.\n'
>>> f.readline()
'Second line of the file\n'
>>> f.readline()
''
```

Para leer líneas de un archivo, puedes iterar sobre el objeto archivo. Esto es eficiente en memoria, rápido, y conduce a un código más simple:

```
>>> for line in f:
...     print(line, end='')
...
This is the first line of the file.
Second line of the file
```

Si querés leer todas las líneas de un archivo en una lista también podés usar `list(f)` o `f.readlines()`.

`f.write(cadena)` escribe el contenido de la *cadena* al archivo, retornando la cantidad de caracteres escritos.

```
>>> f.write('This is a test\n')
15
```

Otros tipos de objetos necesitan ser convertidos – tanto a una cadena (en modo texto) o a un objeto de bytes (en modo binario) – antes de escribirlos:

```
>>> value = ('the answer', 42)
>>> s = str(value)  # convert the tuple to string
>>> f.write(s)
18
```

`f.tell()` retorna un entero que indica la posición actual en el archivo representada como número de bytes desde el comienzo del archivo en modo binario y un número opaco en modo texto.

Para cambiar la posición del objeto archivo, utiliza `f.seek(offset, whence)`. La posición es calculada agregando el *offset* a un punto de referencia; el punto de referencia se selecciona del argumento *whence*. Un valor *whence* de 0 mide desde el comienzo del archivo, 1 usa la posición actual del archivo, y 2 usa el fin del archivo como punto de referencia. *whence* puede omitirse, el valor por defecto es 0, usando el comienzo del archivo como punto de referencia.

```
>>> f = open('workfile', 'rb+')
>>> f.write(b'0123456789abcdef')
16
>>> f.seek(5)      # Go to the 6th byte in the file
5
>>> f.read(1)
b'5'
>>> f.seek(-3, 2)  # Go to the 3rd byte before the end
13
>>> f.read(1)
b'd'
```

En los archivos de texto (aquellos que se abrieron sin una `b` en el modo), se permiten solamente desplazamientos con `seek` relativos al comienzo (con la excepción de ir justo al final con `seek(0, 2)`) y los únicos valores de *desplazamiento* válidos son aquellos retornados por `f.tell()`, o cero. Cualquier otro valor de *desplazamiento* produce un comportamiento indefinido.

Los objetos archivo tienen algunos métodos más, como **`isatty()`** y **`truncate()`** que son usados menos frecuentemente; consultá la Referencia de la Biblioteca para una guía completa sobre los objetos archivo.

7.2.2. Guardar datos estructurados con `json`

Las cadenas pueden fácilmente escribirse y leerse de un archivo. Los números toman algo más de esfuerzo, ya que el método **`read()`** sólo retorna cadenas, que tendrán que ser pasadas a una función como **`int()`**, que toma una cadena como `'123'` y retorna su valor numérico 123. Sin embargo, cuando querés guardar tipos de datos más complejos como listas, diccionarios, o instancias de clases, las cosas se ponen más complicadas.

En lugar de tener a los usuarios constantemente escribiendo y debugueando código para guardar tipos de datos complicados, Python te permite usar el popular formato intercambiable de datos llamado JSON (JavaScript Object Notation) [http://json.org]. El módulo estándar llamado **`json`** puede tomar datos de Python con una jerarquía, y convertirlo a representaciones de cadena de caracteres; este proceso es llamado *serialización*. Reconstruir los datos desde la representación de cadena de caracteres es llamado *deserialización*. Entre serialización y deserialización, la cadena de caracteres representando el objeto quizás haya sido guardado en un archivo o datos, o enviado a una máquina distante por una conexión de red.

Nota

El formato JSON es comúnmente usado por aplicaciones modernas para permitir el intercambio de datos. Muchos programadores ya están familiarizados con él, lo cual lo convierte en una buena opción para la interoperabilidad.

Si tienes un objeto `x`, puedes ver su representación JSON con una simple línea de código:

```
>>> import json
>>> x = [1, 'simple', 'list']
>>> json.dumps(x)
'[1, "simple", "list"]'
```

Otra variante de la función **`dumps()`**, llamada **`dump()`**, simplemente serializa el objeto a un archivo de texto. Así que, si `f` es un objeto archivo de texto abierto para escritura, podemos hacer:

```
json.dump(x, f)
```

Para decodificar un objeto nuevamente, si `f` es un objeto binary file o text file que fue abierto para lectura:

```
x = json.load(f)
```

Nota

Los archivos JSON deben estar codificados en UTF-8. Utilice `encoding="utf-8"` al abrir un archivo JSON como text file tanto para lectura como para escritura.

La simple técnica de serialización puede manejar listas y diccionarios, pero serializar instancias de clases arbitrarias en JSON requiere un poco de esfuerzo extra. La referencia del módulo `json` contiene una explicación de esto.

Ver también

`pickle` - El módulo *pickle*

Contrariamente a JSON, *pickle* es un protocolo que permite la serialización de objetos Python arbitrariamente complejos. Como tal, es específico de Python y no se puede utilizar para comunicarse con aplicaciones escritas en otros lenguajes. También es inseguro de forma predeterminada: deserializar los datos de *pickle* procedentes de un origen que no es de confianza puede ejecutar código arbitrario, si los datos fueron creados por un atacante experto.

8. Errores y excepciones

Hasta ahora los mensajes de error apenas habían sido mencionados, pero si has probado los ejemplos anteriores probablemente hayas visto algunos. Hay (al menos) dos tipos diferentes de errores: *errores de sintaxis* y *excepciones*.

8.1. Errores de sintaxis

Los errores de sintaxis, también conocidos como errores de interpretación, son quizás el tipo de queja más común que tenés cuando todavía estás aprendiendo Python:

```
>>> while True print('Hello world')
  File "<stdin>", line 1
    while True print('Hello world')
               ^^^^^
SyntaxError: invalid syntax
```

The parser repeats the offending line and displays little arrows pointing at the place where the error was detected. Note that this is not always the place that needs to be fixed. In the example, the error is detected at the function **`print()`**, since a colon (`':'`) is missing just before it.

The file name (`<stdin>` in our example) and line number are printed so you know where to look in case the input came from a file.

8.2. Excepciones

Incluso si una declaración o expresión es sintácticamente correcta, puede generar un error cuando se intenta ejecutar. Los errores detectados durante la ejecución se llaman *excepciones*, y no son incondicionalmente fatales: pronto aprenderás a gestionarlos en programas Python. Sin embargo, la mayoría de las excepciones no son gestionadas por el código, y resultan en mensajes de error como los mostrados aquí:

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    10 * (1/0)
          ~^~
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    4 + spam*3
        ^^^^
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    '2' + 2
    ~~~~^~~
TypeError: can only concatenate str (not "int") to str
```

La última línea de los mensajes de error indica qué ha sucedido. Hay excepciones de diferentes tipos, y el tipo se imprime como parte del mensaje: los tipos en el ejemplo son: **`ZeroDivisionError`**, **`NameError`** y **`TypeError`**. La cadena mostrada como tipo de la excepción es el nombre de la excepción predefinida que ha ocurrido. Esto es válido para todas las excepciones predefinidas del intérprete, pero no tiene por que ser así para excepciones definidas por el usuario (aunque es una convención útil). Los nombres de las excepciones estándar son identificadores incorporados al intérprete (no son palabras clave reservadas).

El resto de la línea provee información basado en el tipo de la excepción y qué la causó.

La parte anterior del mensaje de error muestra el contexto donde ocurrió la excepción, en forma de seguimiento de pila. En general, contiene un seguimiento de pila que enumera las líneas de origen; sin embargo, no mostrará las líneas leídas desde la entrada estándar.

Excepciones incorporadas lista las excepciones predefinidas y sus significados.

8.3. Gestionando excepciones

Es posible escribir programas que gestionen determinadas excepciones. Véase el siguiente ejemplo, que le pide al usuario una entrada hasta que ingrese un entero válido, pero permite al usuario interrumpir el programa (usando `Control-C` o lo que soporte el sistema operativo); nótese que una interrupción generada por el usuario es señalizada generando la excepción **`KeyboardInterrupt`**.

```
>>> while True:
...     try:
...         x = int(input("Please enter a number: "))
...         break
...     except ValueError:
...         print("Oops!  That was no valid number.  Try
...
```

La sentencia **`try`** funciona de la siguiente manera.

- Primero, se ejecuta la cláusula *try* (la(s) linea(s) entre las palabras reservadas **`try`** y la **`except`**).
- Si no ocurre ninguna excepción, la cláusula *except* se omite y la ejecución de la cláusula **`try`** finaliza.
- Si ocurre una excepción durante la ejecución de la cláusula **`try`**, se omite el resto de la cláusula. Luego, si su tipo coincide con la excepción nombrada después de la palabra clave **`except`**, se ejecuta la *cláusula except*, y luego la ejecución continúa después del bloque try/except.
- If an exception occurs which does not match the exception named in the *except clause*, it is passed on to outer **`try`** statements; if no handler is found, it is an *unhandled exception* and execution stops with an error message.

A **`try`** statement may have more than one *except clause*, to specify handlers for different exceptions. At most one handler will be executed. Handlers only handle exceptions that occur in the corresponding *try clause*, not in other handlers of the same **`try`** statement. An *except clause* may name multiple exceptions, for example:

```
... except RuntimeError, TypeError, NameError:
...     pass
```

A class in an **`except`** clause matches exceptions which are instances of the class itself or one of its derived classes (but not the other way around — an *except clause* listing a derived class does not match instances of its base classes). For example, the following code will print B, C, D in that order:

```
class B(Exception):
    pass

class C(B):
    pass

class D(C):
    pass

for cls in [B, C, D]:
    try:
        raise cls()
    except D:
        print("D")
    except C:
        print("C")
    except B:
        print("B")
```

Nótese que si las *cláusulas except* estuvieran invertidas (con `except B` primero), habría impreso B, B, B — se usa la primera *cláusula except* coincidente.

Cuando ocurre una excepción, puede tener un valor asociado, también conocido como el *argumento* de la excepción. La presencia y el tipo de argumento depende del tipo de excepción.

La *cláusula except* puede especificar una variable después del nombre de la excepción. La variable está ligada a la instancia de la excepción, que normalmente tiene un atributo `args` que almacena los argumentos. Por conveniencia, los tipos de excepción incorporados definen `__str__()` para imprimir todos los argumentos sin acceder explícitamente a `.args`.

```
>>> try:
...     raise Exception('spam', 'eggs')
... except Exception as inst:
...     print(type(inst))    # the exception type
...     print(inst.args)     # arguments stored in .args
...     print(inst)          # __str__ allows args to be
...                          # but may be overridden in
...     x, y = inst.args     # unpack args
...     print('x =', x)
...     print('y =', y)
...
<class 'Exception'>
('spam', 'eggs')
('spam', 'eggs')
x = spam
y = eggs
```

La salida `__str__()` de la excepción se imprime como la última parte ("detalle") del mensaje para las excepciones no gestionadas.

`BaseException` es la clase base común de todas las excepciones. Una de sus subclases, `Exception`, es la clase base de todas las excepciones no fatales. Las excepciones que no son subclases de `Exception` no se suelen manejar, porque se utilizan para indicar que el programa debe terminar. Entre ellas se incluyen `SystemExit`, que es lanzada por `sys.exit()` y `KeyboardInterrupt`, que se lanza cuando un usuario desea interrumpir el programa.

`Exception` se puede utilizar como un comodín que atrapa (casi) todo. Sin embargo, es una buena práctica ser lo más específico posible con los tipos de excepciones que pretendemos manejar, y permitir que cualquier excepción inesperada se propague.

El patrón más común para gestionar `Exception` es imprimir o registrar la excepción y luego volver a re-lanzarla (permitiendo a un llamador manejar la excepción también):

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("OS error:", err)
except ValueError:
    print("Could not convert data to an integer.")
except Exception as err:
    print(f"Unexpected {err=}, {type(err)=}")
    raise
```

La declaración `try` … `except` tiene una *cláusula else* opcional, que, cuando está presente, debe seguir todas las *cláusulas except*. Es útil para el código que debe ejecutarse si la *cláusula try* no lanza una excepción. Por ejemplo:

```
for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except OSError:
        print('cannot open', arg)
    else:
        print(arg, 'has', len(f.readlines()), 'lines')
        f.close()
```

El uso de la cláusula **`else`** es mejor que agregar código adicional en la cláusula `try` porque evita capturar accidentalmente una excepción que no fue generada por el código que está protegido por la declaración **`try`** … **`except`**.

Los gestores de excepciones no sólo gestionan excepciones que ocurren inmediatamente en la *cláusula try*, sino también aquellas que ocurren dentro de funciones que son llamadas (incluso indirectamente) en la *cláusula try*. Por ejemplo:

```
>>> def this_fails():
...     x = 1/0
...
>>> try:
...     this_fails()
... except ZeroDivisionError as err:
...     print('Handling run-time error:', err)
...
Handling run-time error: division by zero
```

8.4. Lanzando excepciones

La declaración `raise` permite al programador forzar a que ocurra una excepción específica. Por ejemplo:

```
>>> raise NameError('HiThere')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    raise NameError('HiThere')
NameError: HiThere
```

El único argumento de `raise` indica la excepción a lanzar. Debe ser una instancia de excepción o una clase de excepción (una clase que derive de `BaseException`, como `Exception` o una de sus subclases). Si se pasa una clase de excepción, se instanciará implícitamente llamando a su constructor sin argumentos:

```
raise ValueError  # shorthand for 'raise ValueError()'
```

Si es necesario determinar si una excepción fue lanzada pero sin intención de gestionarla, una versión simplificada de la instrucción `raise` te permite relanzarla:

```
>>> try:
...     raise NameError('HiThere')
... except NameError:
...     print('An exception flew by!')
...     raise
...
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    raise NameError('HiThere')
NameError: HiThere
```

8.5. Encadenamiento de excepciones

Si se produce una excepción no gestionada dentro de una sección `except`, se le adjuntará la excepción que se está gestionando y se incluirá en el mensaje de error:

```
>>> try:
...     open("database.sqlite")
... except OSError:
...     raise RuntimeError("unable to handle error")
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    open("database.sqlite")
    ~~~~^^^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory:

During handling of the above exception, another exceptio

Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
    raise RuntimeError("unable to handle error")
RuntimeError: unable to handle error
```

Para indicar que una excepción es consecuencia directa de otra, la sentencia `raise` permite una cláusula opcional `from`:

```
# exc must be exception instance or None.
raise RuntimeError from exc
```

Esto puede resultar útil cuando está transformando excepciones. Por ejemplo:

```
>>> def func():
...     raise ConnectionError
...
>>> try:
...     func()
... except ConnectionError as exc:
...     raise RuntimeError('Failed to open database') fr
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    func()
    ~~~~^^
  File "<stdin>", line 2, in func
ConnectionError

The above exception was the direct cause of the followin

Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
    raise RuntimeError('Failed to open database') from e
RuntimeError: Failed to open database
```

También permite deshabilitar el encadenamiento automático de excepciones utilizando el modismo `from None`:

```
>>> try:
...     open('database.sqlite')
... except OSError:
...     raise RuntimeError from None
...
Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
    raise RuntimeError from None
RuntimeError
```

Para obtener más información sobre la mecánica del encadenamiento, consulte Excepciones incorporadas.

8.6. Excepciones definidas por el usuario

Los programas pueden nombrar sus propias excepciones creando una nueva clase excepción (mirá Clases para más información sobre las clases de Python). Las excepciones, típicamente, deberán derivar de la clase `Exception`, directa o indirectamente.

Las clases de Excepción pueden ser definidas de la misma forma que cualquier otra clase, pero es habitual mantenerlas lo más simples posible, a menudo ofreciendo solo un número de atributos con información sobre el error que leerán los gestores de la excepción.

La mayoría de las excepciones se definen con nombres acabados en «Error», de manera similar a la nomenclatura de las excepciones estándar.

Muchos módulos estándar definen sus propias excepciones para reportar errores que pueden ocurrir en funciones propias.

8.7. Definiendo acciones de limpieza

La declaración `try` tiene otra cláusula opcional cuyo propósito es definir acciones de limpieza que serán ejecutadas bajo ciertas circunstancias. Por ejemplo:

```
>>> try:
...     raise KeyboardInterrupt
... finally:
...     print('Goodbye, world!')
...
Goodbye, world!
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    raise KeyboardInterrupt
KeyboardInterrupt
```

Si una cláusula `finally` está presente, el bloque **`finally`** se ejecutará al final antes de que todo el bloque `try` se complete. La cláusula **`finally`** se ejecuta independientemente de que la cláusula **`try`** produzca o no una excepción. Los siguientes puntos explican casos más complejos en los que se produce una excepción:

- Si ocurre una excepción durante la ejecución de la cláusula **`try`**, la excepción podría ser gestionada por una cláusula `except`. Si la excepción no es gestionada por una cláusula **`except`**, la excepción es relanzada después de que se ejecute el bloque de la cláusula **`finally`**.
- Podría aparecer una excepción durante la ejecución de una cláusula **`except`** o **`else`**. De nuevo, la excepción será relanzada después de que el bloque de la cláusula **`finally`** se ejecute.
- If the **`finally`** clause executes a `break`, `continue` or `return` statement, exceptions are not re-raised. This can be confusing and is therefore discouraged. From version 3.14 the compiler emits a `SyntaxWarning` for it (see PEP 765 [https://peps.python.org/pep-0765/]).
- Si el bloque **`try`** llega a una sentencia `break`, `continue` o `return`, la cláusula **`finally`** se ejecutará justo antes de la ejecución de dicha sentencia.
- If a **`finally`** clause includes a **`return`** statement, the returned value will be the one from the **`finally`** clause's **`return`** statement, not the value from the **`try`** clause's **`return`** statement. This can be confusing and is therefore discouraged. From version 3.14 the compiler emits a `SyntaxWarning` for it (see PEP 765 [https://peps.python.org/pep-0765/]).

Por ejemplo:

```
>>> def bool_return():
...     try:
...         return True
...     finally:
...         return False
...
>>> bool_return()
False
```

Un ejemplo más complicado:

```
>>> def divide(x, y):
...     try:
...         result = x / y
...     except ZeroDivisionError:
...         print("division by zero!")
...     else:
...         print("result is", result)
...     finally:
...         print("executing finally clause")
...
>>> divide(2, 1)
result is 2.0
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    divide("2", "1")
    ~~~~~~^^^^^^^^^^
  File "<stdin>", line 3, in divide
    result = x / y
             ~~^~~
TypeError: unsupported operand type(s) for /: 'str' and
```

Como se puede ver, la cláusula `finally` siempre se ejecuta. La excepción `TypeError` lanzada al dividir dos cadenas de texto no es gestionado por la cláusula `except` y por lo tanto es relanzada luego de que se ejecuta la cláusula `finally`.

En aplicaciones reales, la cláusula `finally` es útil para liberar recursos externos (como archivos o conexiones de red), sin importar si el uso del recurso fue exitoso.

8.8. Acciones predefinidas de limpieza

Algunos objetos definen acciones de limpieza estándar para llevar a cabo cuando el objeto ya no necesario, independientemente de que las operaciones sobre el objeto hayan sido exitosas o no. Véase el siguiente ejemplo, que intenta abrir un archivo e imprimir su contenido en la pantalla.

```
for line in open("myfile.txt"):
    print(line, end="")
```

El problema con este código es que deja el archivo abierto por un periodo de tiempo indeterminado luego de que esta parte termine de ejecutarse. Esto no es un problema en *scripts* simples, pero puede ser un problema en aplicaciones más grandes. La declaración **`with`** permite que los objetos como archivos sean usados de una forma que asegure que siempre se los libera rápido y en forma correcta.:

```
with open("myfile.txt") as f:
    for line in f:
        print(line, end="")
```

Una vez que la declaración se ejecuta, el fichero *f* siempre se cierra, incluso si aparece algún error durante el procesado de las líneas. Los objetos que, como los ficheros, posean acciones predefinidas de limpieza lo indicarán en su documentación.

8.9. Lanzando y gestionando múltiples excepciones no relacionadas

Hay situaciones en las que es necesario informar de varias excepciones que se han producido. Este es a menudo el caso en los marcos de concurrencia, cuando varias tareas pueden haber fallado en paralelo, pero también hay otros casos de uso en los que es deseable continuar la ejecución y recoger múltiples errores en lugar de lanzar la primera excepción.

El incorporado **`ExceptionGroup`** envuelve una lista de instancias de excepción para que puedan ser lanzadas juntas. Es una excepción en sí misma, por lo que puede capturarse como cualquier otra excepción.

```
>>> def f():
...     excs = [OSError('error 1'), SystemError('error 2
...     raise ExceptionGroup('there were problems', excs
...
>>> f()
  + Exception Group Traceback (most recent call last):
  |   File "<stdin>", line 1, in <module>
  |     f()
  |     ~^^
  |   File "<stdin>", line 3, in f
  |     raise ExceptionGroup('there were problems', excs
  | ExceptionGroup: there were problems (2 sub-exceptio
  +-+---------------- 1 ----------------
    | OSError: error 1
    +---------------- 2 ----------------
    | SystemError: error 2
    +------------------------------------
>>> try:
...     f()
... except Exception as e:
...     print(f'caught {type(e)}: {e}')
...
caught <class 'ExceptionGroup'>: there were problems (2
>>>
```

Utilizando `except*` en lugar de `except`, podemos manejar selectivamente sólo las excepciones del grupo que coincidan con un determinado tipo. En el siguiente ejemplo, que muestra un grupo de excepciones anidado, cada cláusula `except*` extrae del grupo las excepciones de un tipo determinado, mientras que deja que el resto de excepciones se propaguen a otras cláusulas y, finalmente, se vuelvan a lanzar.

```
>>> def f():
...     raise ExceptionGroup(
...         "group1",
...         [
...             OSError(1),
...             SystemError(2),
...                 ExceptionGroup(
...                     "group2",
...                     [
...                         OSError(3),
...                         RecursionError(4)
...                     ]
...                 )
...             ]
...     )
...
>>> try:
...     f()
... except* OSError as e:
...     print("There were OSErrors")
... except* SystemError as e:
...     print("There were SystemErrors")
...
There were OSErrors
There were SystemErrors
  + Exception Group Traceback (most recent call last):
  |   File "<stdin>", line 2, in <module>
  |     f()
  |     ~^^
  |   File "<stdin>", line 2, in f
  |     raise ExceptionGroup(
  |     ...<12 lines>...
  |     )
  | ExceptionGroup: group1 (1 sub-exception)
  +-+---------------- 1 ----------------
    | ExceptionGroup: group2 (1 sub-exception)
    +-+---------------- 1 ----------------
      | RecursionError: 4
      +------------------------------------
>>>
```

Tenga en cuenta que las excepciones anidadas en un grupo de excepciones deben ser instancias, no tipos. Esto se debe a que en la práctica las excepciones serían típicamente las que ya han sido planteadas y capturadas por el programa, siguiendo el siguiente patrón:

```
>>> excs = []
... for test in tests:
...     try:
...         test.run()
...     except Exception as e:
...         excs.append(e)
...
>>> if excs:
...    raise ExceptionGroup("Test Failures", excs)
...
```

8.10. Enriqueciendo excepciones con notas

Cuando se crea una excepción para ser lanzada, normalmente se inicializa con información que describe el error que se ha producido. Hay casos en los que es útil añadir información después de que la excepción haya sido capturada. Para este propósito, las excepciones tienen un método `add_note(note)` que acepta una cadena y la añade a la lista de notas de la excepción. La representación estándar del rastreo incluye todas las notas, en el orden en que fueron añadidas, después de la excepción.

```
>>> try:
...     raise TypeError('bad type')
... except Exception as e:
...     e.add_note('Add some information')
...     e.add_note('Add some more information')
...     raise
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    raise TypeError('bad type')
TypeError: bad type
Add some information
Add some more information
```

```
>>>
```

Por ejemplo, al recopilar excepciones en un grupo de excepciones, es posible que queramos añadir información de contexto para los errores individuales. A continuación, cada excepción del grupo tiene una nota que indica cuándo se ha producido ese error.

```
>>> def f():
...     raise OSError('operation failed')
...
>>> excs = []
>>> for i in range(3):
...     try:
...         f()
...     except Exception as e:
...         e.add_note(f'Happened in Iteration {i+1}')
...         excs.append(e)
...
>>> raise ExceptionGroup('We have some problems', excs)
  + Exception Group Traceback (most recent call last):
  |   File "<stdin>", line 1, in <module>
  |     raise ExceptionGroup('We have some problems', ex
  | ExceptionGroup: We have some problems (3 sub-excepti
  +-+---------------- 1 ----------------
    | Traceback (most recent call last):
    |   File "<stdin>", line 3, in <module>
    |     f()
    |     ~^^
    |   File "<stdin>", line 2, in f
    |     raise OSError('operation failed')
    | OSError: operation failed
    | Happened in Iteration 1
    +---------------- 2 ----------------
    | Traceback (most recent call last):
    |   File "<stdin>", line 3, in <module>
    |     f()
    |     ~^^
    |   File "<stdin>", line 2, in f
    |     raise OSError('operation failed')
    | OSError: operation failed
    | Happened in Iteration 2
    +---------------- 3 ----------------
    | Traceback (most recent call last):
    |   File "<stdin>", line 3, in <module>
    |     f()
    |     ~^^
    |   File "<stdin>", line 2, in f
    |     raise OSError('operation failed')
    | OSError: operation failed
    | Happened in Iteration 3
    +------------------------------------
>>>
```

[1]

9. Clases

Las clases proveen una forma de empaquetar datos y funcionalidad juntos. Al crear una nueva clase, se crea un nuevo *tipo* de objeto, permitiendo crear nuevas *instancias* de ese tipo. Cada instancia de clase puede tener atributos adjuntos para mantener su estado. Las instancias de clase también pueden tener métodos (definidos por su clase) para modificar su estado.

Comparado con otros lenguajes de programación, el mecanismo de clases de Python agrega clases con un mínimo de nuevas sintaxis y semánticas. Es una mezcla de los mecanismos de clases encontrados en C++ y Modula-3. Las clases de Python proveen todas las características normales de la Programación Orientada a Objetos: el mecanismo de la herencia de clases permite múltiples clases base, una clase derivada puede sobre escribir cualquier método de su(s) clase(s) base, y un método puede llamar al método de la clase base con el mismo nombre. Los objetos pueden tener una cantidad arbitraria de datos de cualquier tipo. Igual que con los módulos, las clases participan de la naturaleza dinámica de Python: se crean en tiempo de ejecución, y pueden modificarse luego de la creación.

En terminología de C++, normalmente los miembros de las clases (incluyendo los miembros de datos), son *públicos* (excepto ver abajo Variables privadas), y todas las funciones miembro son *virtuales*. Como en Modula-3, no hay atajos para hacer referencia a los miembros del objeto desde sus métodos: la función método se declara con un primer argumento explícito que representa al objeto, el cual se provee implícitamente por la llamada. Como en Smalltalk, las clases mismas son objetos. Esto provee una semántica para importar y renombrar. A diferencia de C++ y Modula-3, los tipos de datos integrados pueden usarse como clases base para que el usuario los extienda. También, como en C++ pero a diferencia de Modula-3, la mayoría de los operadores integrados con sintaxis especial (operadores aritméticos, de sub-índice, etc.) pueden volver a ser definidos por instancias de la clase.

(Sin haber una terminología universalmente aceptada sobre clases, haré uso ocasional de términos de Smalltalk y C++. Usaría términos de Modula-3, ya que su semántica orientada a objetos es más cercana a Python que C++, pero no espero que muchos lectores hayan escuchado hablar de él.)

9.1. Unas palabras sobre nombres y objetos

Los objetos tienen individualidad, y múltiples nombres (en muchos ámbitos) pueden vincularse al mismo objeto. Esto se conoce como *aliasing* en otros lenguajes. Normalmente no se aprecia esto a primera vista en Python, y puede ignorarse sin problemas cuando se maneja tipos básicos inmutables (números, cadenas, tuplas). Sin embargo, el *aliasing*, o renombrado, tiene un efecto posiblemente sorpresivo sobre la semántica de código Python que involucra objetos mutables como listas, diccionarios, y la mayoría de otros tipos. Esto se usa normalmente para beneficio del programa, ya que los renombres funcionan como punteros en algunos aspectos. Por ejemplo, pasar un objeto es barato ya que la implementación solamente pasa el puntero; y si una función modifica el objeto que fue pasado, el que la llama verá el cambio; esto elimina la necesidad de tener dos formas diferentes de pasar argumentos, como en Pascal.

9.2. Ámbitos y espacios de nombres en Python

Antes de ver clases, primero debo decirte algo acerca de las reglas de ámbito de Python. Las definiciones de clases hacen unos lindos trucos con los espacios de nombres, y necesitás saber cómo funcionan los alcances y espacios de nombres para entender por completo cómo es la cosa. De paso, los conocimientos en este tema son útiles para cualquier programador Python avanzado.

Comencemos con unas definiciones.

Un *espacio de nombres* es una relación de nombres a objetos. Muchos espacios de nombres están implementados en este momento como diccionarios de Python, pero eso no se nota para nada (excepto por el desempeño), y puede cambiar en el futuro. Como ejemplos de espacios de nombres tenés: el conjunto de nombres incluidos (conteniendo funciones como **`abs()`**, y los nombres de excepciones integradas); los nombres globales en un módulo; y los nombres locales en la invocación a una función. Lo que es importante saber de los espacios de nombres es que no hay relación en absoluto entre los nombres de espacios de nombres distintos; por ejemplo, dos módulos diferentes pueden tener definidos los dos una función `maximizar` sin confusión; los usuarios de los módulos deben usar el nombre del módulo como prefijo.

Por cierto, yo uso la palabra *atributo* para cualquier cosa después de un punto; por ejemplo, en la expresión `z.real`, `real` es un atributo del objeto `z`. Estrictamente hablando, las referencias a nombres en módulos son referencias a atributos: en la expresión `modulo.funcion`, `modulo` es un objeto módulo y `funcion` es un atributo de éste. En este caso hay una relación directa entre los atributos del módulo y los nombres globales definidos en el módulo: ¡están compartiendo el mismo espacio de nombres! [1]

Los atributos pueden ser de solo lectura, o de escritura. En el segundo caso, es posible asignar valores a los atributos. Los atributos de módulo son modificables: puedes escribir `modname.the_answer = 42`. Los atributos modificables también se pueden eliminar con la declaración **`del`**. Por ejemplo, `del modname.the_answer` eliminará el atributo **`the_answer`** del objeto nombrado por `modname`.

Los espacios de nombres se crean en diferentes momentos y con diferentes tiempos de vida. El espacio de nombres que contiene los nombres incluidos se crea cuando se inicia el intérprete, y nunca se borra. El espacio de nombres global de un módulo se crea cuando se lee la definición de un módulo; normalmente, los espacios de nombres de módulos también duran hasta que el intérprete finaliza. Las instrucciones ejecutadas en el nivel de llamadas superior del intérprete, ya sea desde un script o interactivamente, se consideran parte del módulo llamado **`__main__`**, por lo tanto tienen su propio espacio de nombres global. (Los nombres incluidos en realidad también viven en un módulo; este se llama `builtins`.)

El espacio de nombres local a una función se crea cuando la función es llamada, y se elimina cuando la función retorna o lanza una excepción que no se maneje dentro de la función. (Podríamos decir que lo que pasa en realidad es que ese espacio de nombres se «olvida».) Por supuesto, las llamadas recursivas tienen cada una su propio espacio de nombres local.

Un *ámbito* es una región textual de un programa en Python donde un espacio de nombres es accesible directamente. «Accesible directamente» significa que una referencia sin calificar a un nombre intenta encontrar dicho nombre dentro del espacio de nombres.

Aunque los alcances se determinan de forma estática, se utilizan de forma dinámica. En cualquier momento durante la ejecución, hay 3 o 4 ámbitos anidados cuyos espacios de nombres son directamente accesibles:

- el alcance más interno, que es inspeccionado primero, contiene los nombres locales
- los alcances de cualquier función que encierra a otra, son inspeccionados a partir del alcance más cercano, contienen nombres no locales, pero también no globales
- el penúltimo alcance contiene nombres globales del módulo actual
- el alcance más externo (el último inspeccionado) es el espacio de nombres que contiene los nombres integrados

Si un nombre se declara global, entonces todas las referencias y asignaciones se realizan directamente en el ámbito penúltimo que contiene los nombres globales del módulo. Para volver a enlazar variables encontradas fuera del ámbito más interno, se puede utilizar la declaración `nonlocal`; si no se declara nonlocal, esas variables serán de sólo lectura (un intento de escribir en una variable de este tipo simplemente creará una *nueva* variable local en el ámbito más interno, dejando sin cambios la variable con el mismo nombre en el ámbito externo).

Habitualmente, el ámbito local referencia los nombres locales de la función actual. Fuera de una función, el ámbito local referencia al mismo espacio de nombres que el ámbito global: el espacio de nombres del módulo. Las definiciones de clases crean un espacio de nombres más en el ámbito local.

Es importante notar que los alcances se determinan textualmente: el ámbito global de una función definida en un módulo es el espacio de nombres de ese módulo, no importa desde dónde o con qué alias se llame a la función. Por otro lado, la búsqueda de nombres se hace dinámicamente, en tiempo de ejecución; sin embargo, la definición del lenguaje está evolucionando a hacer resolución de nombres estáticamente, en tiempo de «compilación», ¡así que no te confíes de la resolución de nombres dinámica! (De hecho, las variables locales ya se determinan estáticamente.)

Una peculiaridad especial de Python es que, si no hay una declaración **`global`** o **`nonlocal`** en efecto, las asignaciones a nombres siempre van al ámbito interno. Las asignaciones no copian datos, solamente asocian nombres a objetos. Lo mismo cuando se borra: la declaración `del x` quita la asociación de `x` del espacio de nombres referenciado por el ámbito local. De hecho, todas las operaciones que introducen nuevos nombres usan el ámbito local: en particular, las instrucciones **`import`** y las definiciones de funciones asocian el módulo o nombre de la función al espacio de nombres en el ámbito local.

La declaración **`global`** puede usarse para indicar que ciertas variables viven en el ámbito global y deberían reasignarse allí; la declaración **`nonlocal`** indica que ciertas variables viven en un ámbito encerrado y deberían reasignarse allí.

9.2.1. Ejemplo de ámbitos y espacios de nombre

Este es un ejemplo que muestra como hacer referencia a distintos ámbitos y espacios de nombres, y cómo las declaraciones **`global`** y **`nonlocal`** afectan la asignación de variables:

```
def scope_test():
    def do_local():
        spam = "local spam"
    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

El resultado del código ejemplo es:

```
After local assignment: test spam
After nonlocal assignment: nonlocal spam
After global assignment: nonlocal spam
In global scope: global spam
```

Notá como la asignación *local* (que es el comportamiento normal) no cambió la vinculación de *spam* de *scope_test*. La asignación `nonlocal` cambió la vinculación de *spam* de *scope_test*, y la asignación `global` cambió la vinculación a nivel de módulo.

También podés ver que no había vinculación para *spam* antes de la asignación `global`.

9.3. Un primer vistazo a las clases

Las clases introducen un poquito de sintaxis nueva, tres nuevos tipos de objetos y algo de semántica nueva.

9.3.1. Sintaxis de definición de clases

La forma más sencilla de definición de una clase se ve así:

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

Las definiciones de clases, al igual que las definiciones de funciones (instrucciones `def`) deben ejecutarse antes de que tengan efecto alguno. (Es concebible poner una definición de clase dentro de una rama de un `if`, o dentro de una función.)

En la práctica, las declaraciones dentro de una clase son definiciones de funciones, pero otras declaraciones son permitidas, y a veces resultan útiles; veremos esto más adelante. Las definiciones de funciones dentro de una clase normalmente tienen una lista de argumentos peculiar, dictada por las convenciones de invocación de métodos; a esto también lo veremos más adelante.

Cuando se ingresa una definición de clase, se crea un nuevo espacio de nombres, el cual se usa como ámbito local; por lo tanto, todas las asignaciones a variables locales van a este nuevo espacio de nombres. En particular, las definiciones de funciones asocian el nombre de las funciones nuevas allí.

Cuando una definición de clase se finaliza normalmente (al llegar al final) se crea un *objeto clase*. Básicamente, este objeto envuelve los contenidos del espacio de nombres creado por la definición de la clase; aprenderemos más acerca de los objetos clase en la sección siguiente. El ámbito local original (el que tenía efecto justo antes de que ingrese la definición de la clase) es restablecido, y el objeto clase se asocia allí al nombre que se le puso a la clase en el encabezado de su definición (`ClassName` en el ejemplo).

9.3.2. Objetos clase

Los objetos clase soportan dos tipos de operaciones: hacer referencia a atributos e instanciación.

Para *hacer referencia a atributos* se usa la sintaxis estándar de todas las referencias a atributos en Python: `objeto.nombre`. Los nombres de atributo válidos son todos los nombres que estaban en el espacio de nombres de la clase cuando ésta se creó. Por lo tanto, si la definición de la clase es así:

```
class MyClass:
    """A simple example class"""
    i = 12345

    def f(self):
        return 'hello world'
```

then `MyClass.i` and `MyClass.f` are valid attribute references, returning an integer and a function object, respectively. Class attributes can also be assigned to, so you can change the value of `MyClass.i` by assignment. **`__doc__`** is also a valid attribute, returning the docstring belonging to the class: `"A simple example class"`.

La *instanciación* de clases usa la notación de funciones. Hacé de cuenta que el objeto de clase es una función sin parámetros que retorna una nueva instancia de la clase. Por ejemplo (para la clase de más arriba):

```
x = MyClass()
```

crea una nueva *instancia* de la clase y asigna este objeto a la variable local `x`.

La operación de instanciación («llamar» a un objeto clase) crea un objeto vacío. Muchas clases necesitan crear objetos con instancias en un estado inicial particular. Por lo tanto una clase puede definir un método especial llamado **`__init__()`**, de esta forma:

```
def __init__(self):
    self.data = []
```

Cuando una clase define un método `__init__()`, la instanciación de la clase automáticamente invoca a `__init__()` para la instancia recién creada. Entonces, en este ejemplo, una instancia nueva e inicializada se puede obtener haciendo:

```
x = MyClass()
```

Por supuesto, el método `__init__()` puede tener argumentos para mayor flexibilidad. En ese caso, los argumentos que se pasen al operador de instanciación de la clase van a parar al método `__init__()`. Por ejemplo,

```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

9.3.3. Objetos instancia

Ahora, ¿Qué podemos hacer con los objetos instancia? La única operación que es entendida por los objetos instancia es la referencia de atributos. Hay dos tipos de nombres de atributos válidos, atributos de datos y métodos.

Data attributes correspond to «instance variables» in Smalltalk, and to «data members» in C++. Data attributes need not be declared; like local variables, they spring into existence when they are first assigned to. For example, if `x` is the instance of **`MyClass`** created above, the following piece of code will print the value `16`, without leaving a trace:

```
x.counter = 1
while x.counter < 10:
    x.counter = x.counter * 2
print(x.counter)
del x.counter
```

The other kind of instance attribute reference is a *method*. A method is a function that «belongs to» an object.

Los nombres válidos de métodos de un objeto instancia dependen de su clase. Por definición, todos los atributos de clase que son objetos funciones definen métodos correspondientes de sus instancias. Entonces, en nuestro ejemplo, `x.f` es una referencia a un método válido, dado que `MyClass.f` es una función, pero `x.i` no lo es, dado que `MyClass.i` no lo es. Pero `x.f` no es la misma cosa que `MyClass.f`; es un *objeto método*, no un objeto función.

9.3.4. Objetos método

Generalmente, un método es llamado luego de ser vinculado:

```
x.f()
```

If `x = MyClass()`, as above, this will return the string `'hello world'`. However, it is not necessary to call a method right away: `x.f` is a method object, and can be stored away and called at a later time. For example:

```
xf = x.f
while True:
    print(xf())
```

continuará imprimiendo `hello world` hasta el fin de los días.

¿Qué sucede exactamente cuando un método es llamado? Debés haber notado que `x.f()` fue llamado más arriba sin ningún argumento, a pesar de que la definición de función de **`f()`** especificaba un argumento. ¿Qué pasó con ese argumento? Seguramente Python lanza una excepción cuando una función que requiere un argumento es llamada sin ninguno, aún si el argumento no es utilizado…

De hecho, tal vez hayas adivinado la respuesta: lo que tienen de especial los métodos es que el objeto es pasado como el primer argumento de la función. En nuestro ejemplo, la llamada `x.f()` es exactamente equivalente a `MyClass.f(x)`. En general, llamar a un método con una lista de *n* argumentos es equivalente a llamar a la función correspondiente con una lista de argumentos que es creada insertando el objeto del método antes del primer argumento.

In general, methods work as follows. When a non-data attribute of an instance is referenced, the instance's class is searched. If the name denotes a valid class attribute that is a function object, references to both the instance object and the function object are packed into a method object. When the method object is called with an argument list, a new argument list is constructed from the instance object and the argument list, and the function object is called with this new argument list.

9.3.5. Variables de clase y de instancia

En general, las variables de instancia son para datos únicos de cada instancia y las variables de clase son para atributos y métodos compartidos por todas las instancias de la clase:

```
class Dog:

    kind = 'canine'         # class variable shared by a

    def __init__(self, name):
        self.name = name    # instance variable unique t

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.kind                  # shared by all dogs
'canine'
>>> e.kind                  # shared by all dogs
'canine'
>>> d.name                  # unique to d
'Fido'
>>> e.name                  # unique to e
'Buddy'
```

As discussed in Unas palabras sobre nombres y objetos, shared data can have possibly surprising effects involving mutable objects such as lists and dictionaries. For example, the *tricks* list in the following code should not be used as a class variable because just a single list would be shared by all *Dog* instances:

```
class Dog:

    tricks = []             # mistaken use of a class va

    def __init__(self, name):
        self.name = name

    def add_trick(self, trick):
        self.tricks.append(trick)

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks                # unexpectedly shared by all
['roll over', 'play dead']
```

El diseño correcto de esta clase sería usando una variable de instancia:

```
class Dog:

    def __init__(self, name):
        self.name = name
        self.tricks = []    # creates a new empty list f

    def add_trick(self, trick):
        self.tricks.append(trick)

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks
['roll over']
>>> e.tricks
['play dead']
```

9.4. Algunas observaciones

Si el mismo nombre de atributo aparece tanto en la instancia como en la clase, la búsqueda del atributo prioriza la instancia:

```
>>> class Warehouse:
...    purpose = 'storage'
...    region = 'west'
...
>>> w1 = Warehouse()
>>> print(w1.purpose, w1.region)
storage west
>>> w2 = Warehouse()
>>> w2.region = 'east'
>>> print(w2.purpose, w2.region)
storage east
```

A los atributos de datos los pueden hacer referencia tanto los métodos como los usuarios («clientes») ordinarios de un objeto. En otras palabras, las clases no se usan para implementar tipos de datos abstractos puros. De hecho, en Python no hay nada que haga cumplir el ocultar datos; todo se basa en convención. (Por otro lado, la implementación de Python, escrita en C, puede ocultar por completo detalles de implementación y el control de acceso a un objeto si es necesario; esto se puede usar en extensiones a Python escritas en C.)

Los clientes deben usar los atributos de datos con cuidado; éstos pueden romper invariantes que mantienen los métodos si pisan los atributos de datos. Observá que los clientes pueden añadir sus propios atributos de datos a una instancia sin afectar la validez de sus métodos, siempre y cuando se eviten conflictos de nombres; de nuevo, una convención de nombres puede ahorrar un montón de dolores de cabeza.

No hay un atajo para hacer referencia a atributos de datos (¡u otros métodos!) desde dentro de un método. A mi parecer, esto en realidad aumenta la legibilidad de los métodos: no existe posibilidad alguna de confundir variables locales con variables de instancia cuando repasamos un método.

A menudo, el primer argumento de un método se llama `self` (uno mismo). Esto no es nada más que una convención: el nombre `self` no significa nada en especial para Python. Observá que, sin embargo, si no seguís la convención tu código puede resultar menos legible a otros programadores de Python, y puede llegar a pasar que un programa *navegador de clases* pueda escribirse de una manera que dependa de dicha convención.

Cualquier objeto función que es un atributo de clase define un método para instancias de esa clase. No es necesario que el la definición de la función esté textualmente dentro de la definición de la clase: asignando un objeto función a una variable local en la clase también está bien. Por ejemplo:

```
# Function defined outside the class
def f1(self, x, y):
    return min(x, x+y)

class C:
    f = f1

    def g(self):
        return 'hello world'

    h = g
```

Ahora `f`, `g` y `h` son todos atributos de la clase **`C`** que hacen referencia a objetos función, y consecuentemente son todos métodos de las instancias de **`C`** — `h` siendo exactamente equivalente a `g`. Fijate que esta práctica normalmente sólo sirve para confundir al que lea un programa.

Los métodos pueden llamar a otros métodos de la instancia usando el argumento `self`:

```
class Bag:
    def __init__(self):
        self.data = []

    def add(self, x):
        self.data.append(x)

    def addtwice(self, x):
        self.add(x)
        self.add(x)
```

Los métodos pueden hacer referencia a nombres globales de la misma manera que lo hacen las funciones comunes. El ámbito global asociado a un método es el módulo que contiene su definición. (Una clase nunca se usa como un ámbito global). Si bien es raro encontrar una buena razón para usar datos globales en un método, hay muchos usos legítimos del ámbito global: por lo menos, las funciones y módulos importados en el ámbito global pueden usarse por los métodos, al igual que las funciones y clases definidas en él. Habitualmente, la clase que contiene el método está definida en este ámbito global, y en la siguiente sección veremos algunas buenas razones por las que un método querría hacer referencia a su propia clase.

Todo valor es un objeto, y por lo tanto tiene una *clase* (también llamado su *tipo*). Ésta se almacena como `objeto.__class__`.

9.5. Herencia

Por supuesto, una característica del lenguaje no sería digna del nombre «clase» si no soportara herencia. La sintaxis para una definición de clase derivada se ve así:

```
class DerivedClassName(BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>
```

El nombre **`BaseClassName`** debe estar definido en un ámbito que contenga a la definición de la clase derivada. En el lugar del nombre de la clase base se permiten otras expresiones arbitrarias. Esto puede ser útil, por ejemplo, cuando la clase base está definida en otro módulo:

```
class DerivedClassName(modname.BaseClassName):
```

La ejecución de una definición de clase derivada procede de la misma forma que una clase base. Cuando el objeto clase se construye, se tiene en cuenta a la clase base. Esto se usa para resolver referencias a atributos: si un atributo solicitado no se encuentra en la clase, la búsqueda continúa por la clase base. Esta regla se aplica recursivamente si la clase base misma deriva de alguna otra clase.

No hay nada en especial en la instanciación de clases derivadas: `DerivedClassName()` crea una nueva instancia de la clase. Las referencias a métodos se resuelven de la siguiente manera: se busca el atributo de clase correspondiente, descendiendo por la cadena de clases base si es necesario, y la referencia al método es válida si se entrega un objeto función.

Las clases derivadas pueden redefinir métodos de su clase base. Como los métodos no tienen privilegios especiales cuando llaman a otros métodos del mismo objeto, un método de la clase base que llame a otro método definido en la misma clase base puede terminar llamando a un método de la clase derivada que lo haya redefinido. (Para los programadores de C++: en Python todos los métodos son en efecto `virtuales`.)

Un método redefinido en una clase derivada puede de hecho querer extender en vez de simplemente reemplazar al método de la clase base con el mismo nombre. Hay una manera simple de llamar al método de la clase base directamente: simplemente llamás a `BaseClassName.methodname(self, arguments)`. En ocasiones esto es útil para los clientes también. (Observá que esto sólo funciona si la clase base es accesible como `BaseClassName` en el ámbito global).

Python tiene dos funciones integradas que funcionan con herencia:

- Usar **`isinstance()`** para verificar el tipo de una instancia: `isinstance(obj, int)` será `True` sólo si `obj.__class__` es **`int`** o alguna clase derivada de **`int`**.
- Usar **`issubclass()`** para verificar la herencia de clases: `issubclass(bool, int)` es `True` ya que **`bool`** es una

subclase de **`int`**. Sin embargo, `issubclass(float, int)` es `False` ya que **`float`** no es una subclase de **`int`**.

9.5.1. Herencia múltiple

Python también soporta una forma de herencia múltiple. Una definición de clase con múltiples clases base se ve así:

```
class DerivedClassName(Base1, Base2, Base3):
    <statement-1>
    .
    .
    .
    <statement-N>
```

Para la mayoría de los propósitos, en los casos más simples, podés pensar en la búsqueda de los atributos heredados de una clase padre como una búsqueda en profundidad, de izquierda a derecha, sin repetir la misma clase cuando está dos veces en la jerarquía. Por lo tanto, si un atributo no se encuentra en **`DerivedClassName`**, se busca en **`Base1`**, luego (recursivamente) en las clases base de **`Base1`**, y solo si no se encuentra allí se lo busca en **`Base2`**, y así sucesivamente.

En realidad es un poco más complejo que eso; el orden de resolución de métodos cambia dinámicamente para soportar las llamadas cooperativas a **`super()`**. Este enfoque es conocido en otros lenguajes con herencia múltiple como «llámese al siguiente método» y es más poderoso que la llamada al superior que se encuentra en lenguajes con sólo herencia simple.

Dynamic ordering is necessary because all cases of multiple inheritance exhibit one or more diamond relationships (where at least one of the parent classes can be accessed through multiple paths from the bottommost class). For example, all classes inherit from **`object`**, so any case of multiple inheritance provides more than one path to reach **`object`**. To keep the base classes from being accessed more than once, the dynamic algorithm linearizes the search order in a way that preserves the left-to-right ordering specified in each class, that calls each parent only once, and that is monotonic (meaning that a class can be subclassed without affecting the precedence order of its parents). Taken together, these properties make it possible to design reliable and extensible classes with multiple inheritance. For more detail, see The Python 2.3 Method Resolution Order.

9.6. Variables privadas

Las variables «privadas» de instancia, que no pueden accederse excepto desde dentro de un objeto, no existen en Python. Sin embargo, hay una convención que se sigue en la mayoría del código Python: un nombre prefijado con un guión bajo (por ejemplo, `_spam`) debería tratarse como una parte no pública de la API (más allá de que sea una función, un método, o un dato). Debería considerarse un detalle de implementación y que está sujeto a cambios sin aviso.

Ya que hay un caso de uso válido para los identificadores privados de clase (a saber: colisión de nombres con nombres definidos en las subclases), hay un soporte limitado para este mecanismo. Cualquier identificador con la forma `__spam` (al menos dos guiones bajos al principio, como mucho un guión bajo al final) es textualmente reemplazado por `_nombredeclase__spam`, donde `nombredeclase` es el nombre de clase actual al que se le sacan guiones bajos del comienzo (si los tuviera). Se modifica el nombre del identificador sin importar su posición sintáctica, siempre y cuando ocurra dentro de la definición de una clase.

Ver también

The private name mangling specifications for details and special cases.

La modificación de nombres es útil para dejar que las subclases sobreescriban los métodos sin romper las llamadas a los métodos desde la misma clase. Por ejemplo:

```
class Mapping:
    def __init__(self, iterable):
        self.items_list = []
        self.__update(iterable)

    def update(self, iterable):
        for item in iterable:
            self.items_list.append(item)

    __update = update   # private copy of original updat

class MappingSubclass(Mapping):

    def update(self, keys, values):
        # provides new signature for update()
        # but does not break __init__()
        for item in zip(keys, values):
            self.items_list.append(item)
```

El ejemplo de arriba funcionaría incluso si `MappingSubclass` introdujera un identificador `__update` ya que se reemplaza con `_Mapping__update` en la clase `Mapping` y `_MappingSubclass__update` en la clase `MappingSubclass` respectivamente.

Hay que aclarar que las reglas de modificación de nombres están diseñadas principalmente para evitar accidentes; es posible acceder o modificar una variable que es considerada como privada. Esto hasta puede resultar útil en circunstancias especiales, tales como en el depurador.

Notar que el código pasado a `exec` o `eval()` no considera que el nombre de clase de la clase que invoca sea la clase actual; esto es similar al efecto de la sentencia `global`, efecto que es de similar manera restringido a código que es compilado en conjunto. La misma restricción aplica a `getattr()`, `setattr()` y `delattr()`, así como cuando se referencia a `__dict__` directamente.

9.7. Detalles y Cuestiones Varias

A veces es útil tener un tipo de datos similar al «registro» de Pascal o la «estructura» de C, que sirva para juntar algunos pocos ítems con nombre. El enfoque idiomático es utilizar `dataclasses` con este propósito:

```
from dataclasses import dataclass

@dataclass
class Employee:
    name: str
    dept: str
    salary: int

>>> john = Employee('john', 'computer lab', 1000)
>>> john.dept
'computer lab'
>>> john.salary
1000
```

Un fragmento de código en Python que espera un tipo de dato abstracto en particular a menudo puede recibir una clase que emule los métodos de ese tipo de dato en su lugar. Por ejemplo, si tienes una función que formatea algunos datos de un objeto de archivo, puedes definir una clase con los métodos `read()` y `readline()` que obtienen los datos de un búfer de cadena en su lugar, y pasarla como argumento.

Instance method objects have attributes, too: `m.__self__` is the instance object with the method **`m()`**, and `m.__func__` is the function object corresponding to the method.

9.8. Iteradores

Es probable que hayas notado que la mayoría de los objetos contenedores pueden ser recorridos usando una sentencia `for`:

```
for element in [1, 2, 3]:
    print(element)
for element in (1, 2, 3):
    print(element)
```

```
for key in {'one':1, 'two':2}:
    print(key)
for char in "123":
    print(char)
for line in open("myfile.txt"):
    print(line, end='')
```

Este estilo de acceso es limpio, conciso y conveniente. El uso de iteradores está impregnado y unifica a Python. En bambalinas, la sentencia **`for`** llama a **`iter()`** en el objeto contenedor. La función retorna un objeto iterador que define el método **`__next__()`** que accede elementos en el contenedor de a uno por vez. Cuando no hay más elementos, **`__next__()`** lanza una excepción **`StopIteration`** que le avisa al bucle del **`for`** que hay que terminar. Podés llamar al método **`__next__()`** usando la función integrada **`next()`**; este ejemplo muestra como funciona todo esto:

```
>>> s = 'abc'
>>> it = iter(s)
>>> it
<str_iterator object at 0x10c90e650>
>>> next(it)
'a'
>>> next(it)
'b'
>>> next(it)
'c'
>>> next(it)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    next(it)
StopIteration
```

Habiendo visto la mecánica del protocolo de iteración, es fácil agregar comportamiento de iterador a tus clases. Definí un método **`__iter__()`** que retorne un objeto con un método **`__next__()`**. Si la clase define **`__next__()`**, entonces alcanza con que **`__iter__()`** retorne `self`:

```
class Reverse:
    """Iterator for looping over a sequence backwards.""
    def __init__(self, data):
        self.data = data
        self.index = len(data)

    def __iter__(self):
        return self

    def __next__(self):
        if self.index == 0:
            raise StopIteration
        self.index = self.index - 1
        return self.data[self.index]
```

```
>>> rev = Reverse('spam')
>>> iter(rev)
<__main__.Reverse object at 0x00A1DB50>
>>> for char in rev:
...     print(char)
...
m
a
p
s
```

9.9. Generadores

Generators son una herramienta simple y poderosa para crear iteradores. Están escritas como funciones regulares pero usan la palabra clave **`yield`** siempre que quieran retornar datos. Cada vez que se llama a **`next()`**, el generador se reanuda donde lo dejó (recuerda todos los valores de datos y qué instrucción se ejecutó por última vez). Un ejemplo muestra que los generadores pueden ser trivialmente fáciles de crear:

```
def reverse(data):
    for index in range(len(data)-1, -1, -1):
        yield data[index]
```

```
>>> for char in reverse('golf'):
...     print(char)
...
f
l
o
g
```

Todo lo que puede ser hecho con generadores también puede ser hecho con iteradores basados en clases, como se describe en la sección anterior. Lo que hace que los generadores sean tan compactos es que los métodos **`__iter__()`** y **`__next__()`** son creados automáticamente.

Otra característica clave es que las variables locales y el estado de la ejecución son guardados automáticamente entre llamadas. Esto hace que la función sea más fácil de escribir y quede mucho más claro que hacerlo usando variables de instancia tales como `self.indice` y `self.datos`.

Además de la creación automática de métodos y el guardar el estado del programa, cuando los generadores terminan automáticamente lanzan **`StopIteration`**. Combinadas, estas características facilitan la creación de iteradores, y hacen que no sea más esfuerzo que escribir una función regular.

9.10. Expresiones generadoras

Algunos generadores simples pueden ser escritos de manera concisa como expresiones usando una sintaxis similar a las comprensiones de listas pero con paréntesis en lugar de corchetes. Estas expresiones están hechas para situaciones donde el generador es utilizado de inmediato por la función que lo encierra. Las expresiones generadoras son más compactas pero menos versátiles que las definiciones completas de generadores y tienden a ser más amigables con la memoria que sus comprensiones de listas equivalentes.

Ejemplos:

```
>>> sum(i*i for i in range(10))                 # sum of
285

>>> xvec = [10, 20, 30]
>>> yvec = [7, 5, 3]
>>> sum(x*y for x,y in zip(xvec, yvec))         # dot pr
260

>>> unique_words = set(word for line in page  for word i

>>> valedictorian = max((student.gpa, student.name) for

>>> data = 'golf'
>>> list(data[i] for i in range(len(data)-1, -1, -1))
['f', 'l', 'o', 'g']
```

Notas al pie

Except for one thing. Module objects have a secret read-only attribute called `__dict__` which returns the dictionary used to implement the module’s namespace; the name `__dict__` is an attribute but not a global name. Obviously, using this violates the abstraction of namespace implementation, and should be restricted to things like post-mortem debuggers.

10. Brief tour of the standard library

10.1. Operating system interface

El módulo **`os`** proporciona docenas de funciones para interactuar con el sistema operativo:

```
>>> import os
>>> os.getcwd()      # Return the current working direct
'C:\\Python314'
>>> os.chdir('/server/accesslogs')   # Change current wo
>>> os.system('mkdir today')   # Run the command mkdir i
0
```

Asegúrate de utilizar el estilo `import os` en lugar de `from os import *`. Esto evitará que **`os.open()`** oculte la función integrada **`open()`**, que funciona de manera muy diferente.

Las funciones integradas **`dir()`** y **`help()`** son útiles como ayudas interactivas para trabajar con módulos grandes como **`os`**:

```
>>> import os
>>> dir(os)
<returns a list of all module functions>
>>> help(os)
<returns an extensive manual page created from the modul
```

Para las tareas diarias de administración de archivos y directorios, el módulo **`shutil`** proporciona una interfaz en un nivel superior que es más fácil de usar:

```
>>> import shutil
>>> shutil.copyfile('data.db', 'archive.db')
'archive.db'
```

```
>>> shutil.move('/build/executables', 'installdir')
'installdir'
```

10.2. File wildcards

El módulo **`glob`** proporciona una función para hacer listas de archivos a partir de búsquedas con comodines en directorios:

```
>>> import glob
>>> glob.glob('*.py')
['primes.py', 'random.py', 'quote.py']
```

10.3. Command-line arguments

Los programas frecuentemente necesitan procesar argumentos de linea de comandos. Estos argumentos se almacenan en el atributo *argv* del módulo **`sys`** como una lista. Por ejemplo, consideremos el siguiente archivo :file: “demo.py”:

```
# File demo.py
import sys
print(sys.argv)
```

Este es el resultado de ejecutar `python demo.py one two three` en la línea de comandos:

```
['demo.py', 'one', 'two', 'three']
```

El modulo **`argparse`** provee un mecanismo más sofisticado para procesar argumentos recibidos vía línea de comandos. El siguiente *script* extrae uno o más nombres de archivos y un número opcional de líneas para mostrar:

```
import argparse

parser = argparse.ArgumentParser(
    prog='top',
    description='Show top lines from each file')
parser.add_argument('filenames', nargs='+')
parser.add_argument('-l', '--lines', type=int, default=1
args = parser.parse_args()
print(args)
```

Cuando se ejecuta en la línea de comandos con `python top.py --lines=5 alpha.txt beta.txt`, el *script* establece `args.lines` a `5` y `args.filenames` a `['alpha.txt', 'beta.txt']`.

10.4. Error output redirection and program termination

El módulo **`sys`** también tiene sus atributos para *stdin*, *stdout*, y *stderr*. Este último es útil para emitir mensajes de alerta y error para que se vean incluso cuando se haya redirigido *stdout*:

```
>>> sys.stderr.write('Warning, log file not found starti
Warning, log file not found starting a new one
```

La forma más directa de terminar un programa es usar `sys.exit()`.

10.5. String pattern matching

El módulo **`re`** provee herramientas de expresiones regulares para un procesamiento avanzado de cadenas. Para manipulación y coincidencias complejas, las expresiones regulares ofrecen soluciones concisas y optimizadas:

```
>>> import re
>>> re.findall(r'\bf[a-z]*', 'which foot or hand fell fa
['foot', 'fell', 'fastest']
>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat'
'cat in the hat'
```

Cuando sólo se necesitan funciones sencillas, se prefieren los métodos de cadenas porque son más fáciles de leer y depurar:

```
>>> 'tea for too'.replace('too', 'two')
```

```
'tea for two'
```

10.6. Matemáticas

The **`math`** module gives access to the underlying C library functions for floating-point math:

```
>>> import math
>>> math.cos(math.pi / 4)
0.70710678118654757
>>> math.log(1024, 2)
10.0
```

El módulo **`random`** provee herramientas para realizar selecciones aleatorias:

```
>>> import random
>>> random.choice(['apple', 'pear', 'banana'])
'apple'
>>> random.sample(range(100), 10)   # sampling without r
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
>>> random.random()    # random float from the interval
0.17970987693706186
>>> random.randrange(6)    # random integer chosen from
4
```

El módulo **`statistics`** calcula propiedades de estadística básica (la media, mediana, varianza, etc) de datos numéricos:

```
>>> import statistics
>>> data = [2.75, 1.75, 1.25, 0.25, 0.5, 1.25, 3.5]
>>> statistics.mean(data)
1.6071428571428572
>>> statistics.median(data)
1.25
>>> statistics.variance(data)
1.3720238095238095
```

El proyecto SciPy <https://scipy.org> tiene muchos otros módulos para cálculos numéricos.

10.7. Internet access

Hay varios módulos para acceder a Internet y procesar sus protocolos. Dos de los más simples son **`urllib.request`** para traer data de URLs y **`smtplib`** para enviar correos:

```
>>> from urllib.request import urlopen
>>> with urlopen('https://docs.python.org/3/') as respo
...     for line in response:
...         line = line.decode()                 # Convert b
...         if 'updated' in line:
...             print(line.rstrip())             # Remove tr
...
      Last updated on Nov 11, 2025 (20:11 UTC).

>>> import smtplib
>>> server = smtplib.SMTP('localhost')
>>> server.sendmail('soothsayer@example.org', 'jcaesar@e
... """To: jcaesar@example.org
... From: soothsayer@example.org
...
... Beware the Ides of March.
... """)
>>> server.quit()
```

(Nota que el segundo ejemplo necesita un servidor de correo corriendo en la máquina local)

10.8. Dates and times

El módulo **`datetime`** ofrece clases para gestionar fechas y tiempos tanto de manera simple como compleja. Aunque soporta aritmética sobre fechas y tiempos, la aplicación se centra en la extracción eficiente de elementos para el formateo y la manipulación de los datos de salida. El módulo también admite objetos que tienen en cuenta la zona horaria.

```
>>> # dates are easily constructed and formatted
>>> import datetime as dt
>>> now = dt.date.today()
>>> now
datetime.date(2003, 12, 2)
>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d c
'12-02-03. 02 Dec 2003 is a Tuesday on the 02 day of Dec

>>> # dates support calendar arithmetic
>>> birthday = dt.date(1964, 7, 31)
>>> age = now - birthday
>>> age.days
14368
```

10.9. Data compression

Los módulos admiten directamente los formatos más comunes de archivo y compresión de datos, entre ellos : **`zlib`**, **`gzip`**, **`bz2`**, **`lzma`**, **`zipfile`** y **`tarfile`**.

```
>>> import zlib
>>> s = b'witch which has which witches wrist watch'
>>> len(s)
41
>>> t = zlib.compress(s)
>>> len(t)
37
>>> zlib.decompress(t)
b'witch which has which witches wrist watch'
>>> zlib.crc32(s)
226805979
```

10.10. Performance measurement

Algunos usuarios de Python desarrollan un profundo interés en conocer el rendimiento relativo de las diferentes soluciones al mismo problema. Python provee una herramienta de medición que responde inmediatamente a esas preguntas.

Por ejemplo, puede ser tentador usar la característica de empaquetado y desempaquetado de las tuplas en lugar de la solución tradicional para intercambiar argumentos. El módulo `timeit` muestra rápidamente una ligera ventaja de rendimiento:

```
>>> from timeit import Timer
>>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()
0.57535828626024577
>>> Timer('a,b = b,a', 'a=1; b=2').timeit()
0.54962537085770791
```

En contraste con el fino nivel de medición del módulo `timeit`, los módulos `profile` y `pstats` proveen herramientas para identificar secciones críticas de tiempo en bloques de código más grandes.

10.11. Quality control

Una forma para desarrollar software de alta calidad consiste en escribir pruebas para cada función mientras se desarrolla y correr esas pruebas frecuentemente durante el proceso de desarrollo.

El módulo `doctest` provee una herramienta para revisar un módulo y validar las pruebas integradas en las cadenas de documentación (o *docstring*) del programa. La construcción de las pruebas es tan sencillo como cortar y pegar una ejecución típica junto con sus resultados en los docstrings. Esto mejora la documentación al proveer al usuario un ejemplo y permite que el módulo *doctest* se asegure que el código permanece fiel a la documentación:

```
def average(values):
    """Computes the arithmetic mean of a list of numbers

    >>> print(average([20, 30, 70]))
    40.0
    """
    return sum(values) / len(values)
```

```
import doctest
doctest.testmod()   # automatically validate the embedde
```

El módulo **`unittest`** no es tan sencillo como el módulo **`doctest`**, pero permite mantener un conjunto más completo de pruebas en un archivo independiente:

```
import unittest

class TestStatisticalFunctions(unittest.TestCase):

    def test_average(self):
        self.assertEqual(average([20, 30, 70]), 40.0)
        self.assertEqual(round(average([1, 5, 7]), 1), 4
        with self.assertRaises(ZeroDivisionError):
            average([])
        with self.assertRaises(TypeError):
            average(20, 30, 70)

unittest.main()  # Calling from the command line invokes
```

10.12. Batteries included

Python tiene una filosofía de «pilas incluidas». Esto se ve mejor en las capacidades robustas y sofisticadas de sus paquetes más grandes. Por ejemplo:

- Los módulos **`xmlrpc.client`** y **`xmlrpc.server`** convierten la implementación de llamadas a procedimientos remotos en una tarea casi trivial. A pesar de los nombres de los módulos, no se necesita ningún conocimiento o manejo directo de XML.
- The **`email`** package is a library for managing email messages, including MIME and other RFC 5322 [https://datatracker.ietf.org/doc/html/rfc5322.html]-based message documents. Unlike **`smtplib`** and **`poplib`** which actually send and receive messages, the email package has a complete toolset for building or decoding complex message structures (including attachments) and for implementing internet

encoding and header protocols.
- El paquete **`json`** proporciona un sólido soporte para analizar este popular formato de intercambio de datos. El módulo **`csv`** permite la lectura y escritura directa de archivos en formato de valor separado por comas, comúnmente compatible con bases de datos y hojas de cálculo. El procesamiento XML es compatible con los paquetes **`xml.etree.ElementTree`**, **`xml.dom`** y **`xml.sax`**. Juntos, estos módulos y paquetes simplifican en gran medida el intercambio de datos entre aplicaciones de Python y otras herramientas.
- El módulo **`sqlite3`** es un wrapper de la biblioteca de bases de datos SQLite, proporciona una base de datos constante que se puede actualizar y a la que se puede acceder utilizando una sintaxis SQL ligeramente no estándar.
- La internacionalización es compatible con una serie de módulos, incluyendo **`gettext`**, **`locale`**, y el paquete **`codecs`**.

11. Brief tour of the standard library — part II

Este segundo paseo cubre módulos más avanzados que facilitan necesidades de programación complejas. Estos módulos raramente se usan en scripts cortos.

11.1. Output formatting

El módulo **`reprlib`** provee una versión de **`repr()`** ajustada para mostrar contenedores grandes o profundamente anidados, en forma abreviada:

```
>>> import reprlib
>>> reprlib.repr(set('supercalifragilisticexpialidocious
"{'a', 'c', 'd', 'e', 'f', 'g', ...}"
```

El módulo **`pprint`** ofrece un control más sofisticado de la forma en que se imprimen tanto los objetos predefinidos como los objetos definidos por el usuario, de manera que sean legibles por el intérprete. Cuando el resultado ocupa más de una línea, el generador de «impresiones lindas» agrega saltos de línea y sangrías para mostrar la estructura de los datos más claramente:

```
>>> import pprint
>>> t = [[[['black', 'cyan'], 'white', ['green', 'red']]
...     'yellow'], 'blue']]]
...
>>> pprint.pprint(t, width=30)
[[[['black', 'cyan'],
   'white',
   ['green', 'red']],
  [['magenta', 'yellow'],
   'blue']]]
```

El módulo `textwrap` formatea párrafos de texto para que quepan dentro de cierto ancho de pantalla:

```
>>> import textwrap
>>> doc = """The wrap() method is just like fill() excep
... a list of strings instead of one big string with new
... the wrapped lines."""
...
>>> print(textwrap.fill(doc, width=40))
The wrap() method is just like fill()
except that it returns a list of strings
instead of one big string with newlines
to separate the wrapped lines.
```

El módulo `locale` accede a una base de datos de formatos específicos a una cultura. El atributo *grouping* de la función *format* permite una forma directa de formatear números con separadores de grupo:

```
>>> import locale
>>> locale.setlocale(locale.LC_ALL, 'English_United Stat
'English_United States.1252'
>>> conv = locale.localeconv()          # get a mapping
>>> x = 1234567.8
>>> locale.format_string("%d", x, grouping=True)
'1,234,567'
>>> locale.format_string("%s%.*f", (conv['currency_symbo
...                          conv['frac_digits'], x), groupi
'$1,234,567.80'
```

11.2. Plantillas

El módulo `string` incluye una clase versátil `Template` (plantilla) con una sintaxis simplificada apta para ser editada por usuarios finales. Esto permite que los usuarios personalicen sus aplicaciones sin necesidad de modificar la aplicación en sí.

El formato usa marcadores cuyos nombres se forman con `$` seguido de identificadores Python válidos (caracteres alfanuméricos y guión de subrayado). Si se los encierra entre llaves, pueden seguir más caracteres alfanuméricos sin necesidad de dejar espacios en blanco. `$$` genera un `$`:

```
>>> from string import Template
>>> t = Template('${village}folk send $$10 to $cause.')
>>> t.substitute(village='Nottingham', cause='the ditch
'Nottinghamfolk send $10 to the ditch fund.'
```

El método **`substitute()`** lanza **`KeyError`** cuando no se suministra ningún valor para un marcador mediante un diccionario o argumento por nombre. Para algunas aplicaciones los datos suministrados por el usuario puede ser incompletos, y el método **`safe_substitute()`** puede ser más apropiado: deja los marcadores inalterados cuando falten datos:

```
>>> t = Template('Return the $item to $owner.')
>>> d = dict(item='unladen swallow')
>>> t.substitute(d)
Traceback (most recent call last):
  ...
KeyError: 'owner'
>>> t.safe_substitute(d)
'Return the unladen swallow to $owner.'
```

Las subclases de *Template* pueden especificar un delimitador propio. Por ejemplo, una utilidad de renombrado por lotes para una galería de fotos puede escoger usar signos de porcentaje para los marcadores tales como la fecha actual, el número de secuencia de la imagen, o el formato de archivo:

```
>>> import time, os.path
>>> photofiles = ['img_1074.jpg', 'img_1076.jpg', 'img_1
>>> class BatchRename(Template):
...     delimiter = '%'
...
>>> fmt = input('Enter rename style (%d-date %n-seqnum %
Enter rename style (%d-date %n-seqnum %f-format):  Ashle

>>> t = BatchRename(fmt)
>>> date = time.strftime('%d%b%y')
>>> for i, filename in enumerate(photofiles):
...     base, ext = os.path.splitext(filename)
...     newname = t.substitute(d=date, n=i, f=ext)
...     print('{0} --> {1}'.format(filename, newname))

img_1074.jpg --> Ashley_0.jpg
img_1076.jpg --> Ashley_1.jpg
img_1077.jpg --> Ashley_2.jpg
```

Las plantillas también pueden ser usadas para separar la lógica del programa de los detalles de múltiples formatos de salida. Esto permite sustituir plantillas específicas para archivos XML, reportes en texto plano, y reportes web en HTML.

11.3. Working with binary data record layouts

El módulo **`struct`** provee las funciones **`pack()`** y **`unpack()`** para trabajar con formatos de registros binarios de longitud variable. El siguiente ejemplo muestra cómo recorrer la información de encabezado en un archivo ZIP sin usar el módulo **`zipfile`**. Los códigos `"H"` e `"I"` representan números sin signo de dos y cuatro bytes respectivamente. El `"<"` indica que son de tamaño estándar y los bytes tienen ordenamiento *little-endian*:

```
import struct

with open('myfile.zip', 'rb') as f:
    data = f.read()

start = 0
for i in range(3):                      # show the first
    start += 14
    fields = struct.unpack('<IIIHH', data[start:start+16
    crc32, comp_size, uncomp_size, filenamesize, extra_s

    start += 16
    filename = data[start:start+filenamesize]
    start += filenamesize
    extra = data[start:start+extra_size]
    print(filename, hex(crc32), comp_size, uncomp_size)

    start += extra_size + comp_size     # skip to the ne
```

11.4. Multi-hilos

La técnica de multi-hilos (o *multi-threading*) permite desacoplar tareas que no tienen dependencia secuencial. Los hilos se pueden usar para mejorar el grado de reacción de las aplicaciones que aceptan entradas del usuario mientras otras tareas se ejecutan en segundo plano. Un caso de uso relacionado es ejecutar E/S en paralelo con cálculos en otro hilo.

El código siguiente muestra cómo el módulo de alto nivel **`threading`** puede ejecutar tareas en segundo plano mientras el programa principal continúa su ejecución:

```
import threading, zipfile

class AsyncZip(threading.Thread):
    def __init__(self, infile, outfile):
        super().__init__()
        self.infile = infile
        self.outfile = outfile

    def run(self):
        with zipfile.ZipFile(self.outfile, 'w', zipfile.
            f.write(self.infile)
        print('Finished background zip of:', self.infile

background = AsyncZip('mydata.txt', 'myarchive.zip')
background.start()
print('The main program continues to run in foreground.'

background.join()    # Wait for the background task to f
print('Main program waited until background was done.')
```

El desafío principal de las aplicaciones multi-hilo es la coordinación entre los hilos que comparten datos u otros recursos. A ese fin, el módulo *threading* provee una serie de primitivas de sincronización que incluyen bloqueos, eventos, variables de condición, y semáforos.

Aún cuando esas herramientas son poderosas, pequeños errores de diseño pueden resultar en problemas difíciles de reproducir. La forma preferida de coordinar tareas es concentrar todos los accesos a un recurso en un único hilo y después usar el módulo **`queue`** para alimentar dicho hilo con pedidos desde otros hilos. Las aplicaciones que usan objetos **`Queue`** para comunicación y coordinación entre hilos son más fáciles de diseñar, más legibles, y más confiables.

11.5. Registrando

El módulo **`logging`** ofrece un sistema de registros (*logs*) completo y flexible. En su forma más simple, los mensajes de registro se envían a un archivo o a `sys.stderr`:

```
import logging
logging.debug('Debugging information')
logging.info('Informational message')
logging.warning('Warning:config file %s not found', 'ser
logging.error('Error occurred')
logging.critical('Critical error -- shutting down')
```

Ésta es la salida obtenida:

```
WARNING:root:Warning:config file server.conf not found
ERROR:root:Error occurred
CRITICAL:root:Critical error -- shutting down
```

De forma predeterminada, los mensajes de depuración e informativos se suprimen, y la salida se envía al error estándar. Otras opciones de salida incluyen mensajes de enrutamiento a través de correo electrónico, datagramas, sockets, o un servidor HTTP. Nuevos filtros pueden seleccionar diferentes rutas basadas en la prioridad del mensaje: **`DEBUG`**, **`INFO`**, **`WARNING`**, **`ERROR`**, y **`CRITICAL`** (Depuración, Informativo, Atención, Error y Crítico respectivamente)

El sistema de registro puede configurarse directamente desde Python o puede cargarse la configuración desde un archivo modificable por el usuario para personalizar el registro sin alterar la aplicación.

11.6. Weak references

Python realiza administración de memoria automática (cuenta de referencias para la mayoría de los objetos, y garbage collection para eliminar ciclos). La memoria se libera poco después de que la última referencia a la misma haya sido eliminada.

Este enfoque funciona bien para la mayoría de las aplicaciones pero de vez en cuando existe la necesidad de controlar objetos sólo mientras estén siendo utilizados por otra cosa. Desafortunadamente, el sólo hecho de controlarlos crea una referencia que los convierte en permanentes. El módulo **`weakref`** provee herramientas para controlar objetos sin crear una referencia. Cuando el objeto no se necesita mas, es removido automáticamente de una tabla de referencias débiles y se dispara una *callback* para objetos *weakref*. Comúnmente las aplicaciones incluyen cacheo de objetos que son costosos de crear:

```
>>> import weakref, gc
>>> class A:
...     def __init__(self, value):
...         self.value = value
...     def __repr__(self):
...         return str(self.value)
...
>>> a = A(10)                   # create a reference
>>> d = weakref.WeakValueDictionary()
>>> d['primary'] = a            # does not create a refe
>>> d['primary']                # fetch the object if it
10
>>> del a                       # remove the one referen
>>> gc.collect()                # run garbage collectior
```

```
0
>>> d['primary']                # entry was automaticall
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    d['primary']                # entry was automaticall
  File "C:/python314/lib/weakref.py", line 46, in __geti
    o = self.data[key]()
KeyError: 'primary'
```

11.7. Tools for working with lists

Muchas necesidades de estructuras de datos pueden ser satisfechas con el tipo integrado lista. Sin embargo, a veces se hacen necesarias implementaciones alternativas con rendimientos distintos.

The **`array`** module provides an **`array`** object that is like a list that stores only homogeneous data and stores it more compactly. The following example shows an array of numbers stored as two byte unsigned binary numbers (typecode `"H"`) rather than the usual 16 bytes per entry for regular lists of Python int objects:

```
>>> from array import array
>>> a = array('H', [4000, 10, 700, 22222])
>>> sum(a)
26932
>>> a[1:3]
array('H', [10, 700])
```

The **`collections`** module provides a **`deque`** object that is like a list with faster appends and pops from the left side but slower lookups in the middle. These objects are well suited for implementing queues and breadth first tree searches:

```
>>> from collections import deque
>>> d = deque(["task1", "task2", "task3"])
>>> d.append("task4")
>>> print("Handling", d.popleft())
Handling task1
```

```
unsearched = deque([starting_node])
def breadth_first_search(unsearched):
    node = unsearched.popleft()
    for m in gen_moves(node):
        if is_goal(m):
            return m
        unsearched.append(m)
```

Además de las implementaciones alternativas de listas, la biblioteca ofrece otras herramientas como el módulo **`bisect`** con funciones para manipular listas ordenadas:

```
>>> import bisect
>>> scores = [(100, 'perl'), (200, 'tcl'), (400, 'lua'),
>>> bisect.insort(scores, (300, 'ruby'))
>>> scores
[(100, 'perl'), (200, 'tcl'), (300, 'ruby'), (400, 'lua'
```

El módulo **`heapq`** provee funciones para implementar pilas (*heaps*) basados en listas comunes. El menor valor ingresado se mantiene en la posición cero. Esto es útil para aplicaciones que acceden a menudo al elemento más chico pero no quieren hacer un orden completo de la lista:

```
>>> from heapq import heapify, heappop, heappush
>>> data = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]
>>> heapify(data)                      # rearrange the l
>>> heappush(data, -5)                 # add a new entry
>>> [heappop(data) for i in range(3)]  # fetch the three
[-5, 0, 1]
```

11.8. Decimal floating-point arithmetic

The **`decimal`** module offers a **`Decimal`** datatype for decimal floating-point arithmetic. Compared to the built-in **`float`** implementation of binary floating point, the class is especially helpful for

- aplicaciones financieras y otros usos que requieren

representación decimal exacta,
- control sobre la precisión,
- control sobre el redondeo para cumplir requisitos legales,
- seguimiento de dígitos decimales significativos, o
- aplicaciones donde el usuario espera que los resultados coincidan con cálculos hecho a mano.

Por ejemplo, calcular un impuesto del 5% de una tarifa telefónica de 70 centavos da resultados distintos con punto flotante decimal y punto flotante binario. La diferencia se vuelve significativa si los resultados se redondean al centavo más próximo:

```
>>> from decimal import *
>>> round(Decimal('0.70') * Decimal('1.05'), 2)
Decimal('0.74')
>>> round(.70 * 1.05, 2)
0.73
```

El resultado con `Decimal` conserva un cero al final, calculando automáticamente cuatro cifras significativas a partir de los multiplicandos con dos cifras significativas. Decimal reproduce la matemática como se la hace a mano, y evita problemas que pueden surgir cuando el punto flotante binario no puede representar exactamente cantidades decimales.

La representación exacta permite a la clase `Decimal` hacer cálculos de modulo y pruebas de igualdad que son inadecuadas para punto flotante binario:

```
>>> Decimal('1.00') % Decimal('.10')
Decimal('0.00')
>>> 1.00 % 0.10
0.09999999999999995

>>> sum([Decimal('0.1')]*10) == Decimal('1.0')
True
>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
False
```

El módulo `decimal` provee aritmética con tanta precisión como haga falta:

```
>>> getcontext().prec = 36
>>> Decimal(1) / Decimal(7)
Decimal('0.142857142857142857142857142857142857')
```

12. Entornos virtuales y paquetes

12.1. Introducción

Las aplicaciones en Python usualmente hacen uso de paquetes y módulos que no forman parte de la librería estándar. Las aplicaciones a veces necesitan una versión específica de una librería, debido a que dicha aplicación requiere que un bug particular haya sido solucionado o bien la aplicación ha sido escrita usando una versión obsoleta de la interfaz de la librería.

Esto significa que tal vez no sea posible para una instalación de Python cumplir los requerimientos de todas las aplicaciones. Si la aplicación A necesita la versión 1.0 de un módulo particular y la aplicación B necesita la versión 2.0, entonces los requerimientos entran en conflicto e instalar la versión 1.0 o 2.0 dejará una de las aplicaciones sin funcionar.

La solución a este problema es crear un entorno virtual, un directorio que contiene una instalación de Python de una versión en particular, además de unos cuantos paquetes adicionales.

Diferentes aplicaciones pueden entonces usar entornos virtuales diferentes. Para resolver el ejemplo de requerimientos en conflicto citado anteriormente, la aplicación A puede tener su propio entorno virtual con la versión 1.0 instalada mientras que la aplicación B tiene otro entorno virtual con la versión 2.0. Si la aplicación B requiere que actualizar la librería a la versión 3.0, ésto no afectará el entorno virtual de la aplicación A.

12.2. Creando entornos virtuales

The module used to create and manage virtual environments is called `venv`. **`venv`** will install the Python version from which the command was run (as reported by the `--version` option). For instance, executing the command with `python3.12` will install version 3.12.

Para crear un entorno virtual, decide en que carpeta quieres crearlo y ejecuta el módulo `venv` como script con la ruta a la carpeta:

```
python -m venv tutorial-env
```

Esto creará el directorio `tutorial-env` si no existe, y también creará directorios dentro de él que contienen una copia del intérprete de Python y varios archivos de soporte.

Una ruta común para el directorio de un entorno virtual es `.venv`. Ese nombre mantiene el directorio típicamente escondido en la consola y fuera de vista mientras le da un nombre que explica cuál es el motivo de su existencia. También permite que no haya conflicto con los ficheros de definición de variables de entorno `.env` que algunas herramientas soportan.

Una vez creado el entorno virtual, podrás activarlo.

En Windows, ejecuta:

```
tutorial-env\Scripts\activate
```

En Unix o MacOS, ejecuta:

```
source tutorial-env/bin/activate
```

(Este script está escrito para la consola bash. Si usas las consolas **csh** or **fish**, hay scripts alternativos `activate.csh` y `activate.fish` que deberá usar en su lugar.)

Activar el entorno virtual cambiará el prompt de tu consola para mostrar que entorno virtual está usando, y modificará el entorno para que al ejecutar `python` sea con esa versión e instalación en particular. Por ejemplo:

```
$ source ~/envs/tutorial-env/bin/activate
(tutorial-env) $ python
```

```
Python 3.5.1 (default, May  6 2016, 10:59:36)
  ...
>>> import sys
>>> sys.path
['', '/usr/local/lib/python35.zip', ...,
'~/envs/tutorial-env/lib/python3.5/site-packages']
>>>
```

Para desactivar el entorno virtual, digita:

```
deactivate
```

en el terminal.

12.3. Manejando paquetes con pip

Puede instalar, actualizar y eliminar paquetes usando un programa llamado **pip**. De forma predeterminada, `pip` instalará paquetes desde el Indice de Paquetes de Python [https://pypi.org]. Puede navegar por el índice de paquetes de Python yendo a él en su navegador web.

`pip` tiene varios subcomandos: «install», «uninstall», «freeze», etc. (Consulte la guía Installing Python modules para obtener la documentación completa de `pip`).

Se puede instalar la última versión de un paquete especificando el nombre del paquete:

```
(tutorial-env) $ python -m pip install novas
Collecting novas
  Downloading novas-3.1.1.3.tar.gz (136kB)
Installing collected packages: novas
  Running setup.py install for novas
Successfully installed novas-3.1.1.3
```

También se puede instalar una versión específica de un paquete ingresando el nombre del paquete seguido de `==` y el número de versión:

```
(tutorial-env) $ python -m pip install requests==2.6.0
Collecting requests==2.6.0
  Using cached requests-2.6.0-py2.py3-none-any.whl
Installing collected packages: requests
Successfully installed requests-2.6.0
```

Si se ejecuta de nuevo el comando, `pip` detectará que la versión ya está instalada y no hará nada. Se puede ingresar un número de versión diferente para instalarlo, o se puede ejecutar `pip install --upgrade` para actualizar el paquete a la última versión:

```
(tutorial-env) $ python -m pip install --upgrade request
Collecting requests
Installing collected packages: requests
  Found existing installation: requests 2.6.0
    Uninstalling requests-2.6.0:
      Successfully uninstalled requests-2.6.0
Successfully installed requests-2.7.0
```

`pip -m pip uninstall` seguido de uno o varios nombres de paquetes eliminará los paquetes del entorno virtual.

`python -m pip show` mostrará información de un paquete en particular:

```
(tutorial-env) $ python -m pip show requests
---
Metadata-Version: 2.0
Name: requests
Version: 2.7.0
Summary: Python HTTP for Humans.
Home-page: http://python-requests.org
Author: Kenneth Reitz
Author-email: me@kennethreitz.com
License: Apache 2.0
Location: /Users/akuchling/envs/tutorial-env/lib/python3
Requires:
```

`python -m pip list` mostrará todos los paquetes instalados en el entorno virtual:

```
(tutorial-env) $ python -m pip list
novas (3.1.1.3)
numpy (1.9.2)
pip (7.0.3)
requests (2.7.0)
setuptools (16.0)
```

`python -m pip freeze` retorna una lista de paquetes instalados, pero el formato de salida es el requerido por `python -m pip install`. Una convención común es poner esta lista en un archivo `requirements.txt`:

```
(tutorial-env) $ python -m pip freeze > requirements.txt
(tutorial-env) $ cat requirements.txt
novas==3.1.1.3
numpy==1.9.2
requests==2.7.0
```

El archivo `requirements.txt` puede ser agregado al controlador de versiones y distribuido como parte de la aplicación. Los usuarios pueden entonces instalar todos los paquetes necesarios con `install -r`:

```
(tutorial-env) $ python -m pip install -r requirements.t
Collecting novas==3.1.1.3 (from -r requirements.txt (li
  ...
Collecting numpy==1.9.2 (from -r requirements.txt (line
  ...
Collecting requests==2.7.0 (from -r requirements.txt (li
  ...
Installing collected packages: novas, numpy, requests
  Running setup.py install for novas
Successfully installed novas-3.1.1.3 numpy-1.9.2 request
```

`pip` tiene muchas más opciones. Consulte la guía Installing Python modules para obtener documentación completa de `pip`. Cuando haya escrito un paquete y desee que esté disponible en el índice de paquetes de Python, consulte la Guía de usuario de empaquetado de

Python [https://packaging.python.org/en/latest/tutorials/packaging-projects/].

[1]

13. ¿Y ahora qué?

Leer este tutorial probablemente reforzó tu interés por usar Python, deberías estar ansioso por aplicar Python a la resolución de tus problemas reales. ¿A dónde deberías ir para aprender más?

Este tutorial forma parte del conjunto de documentación de Python. Algunos otros documentos que encontrarás en este conjunto son:

- La biblioteca estándar de Python:

  Deberías navegar a través de este manual, que da una completa (aunque breve) referencia sobre los tipos, funciones y módulos en la librería estándar. La distribución estándar de Python incluye *mucho* más código adicional. Hay módulos para leer buzones Unix, recuperar documentos vía HTTP, generar números aleatorios, analizar opciones de línea de comandos, escribir programas CGI, comprimir datos y muchas más tareas. Echar una ojeada a la Librería de Referencia te dará una idea de lo que está disponible.

- Installing Python modules explica como instalar módulos adicionales escritos por otros usuarios de Python.

- Referencia del Lenguaje Python: Una explicación detallada de la sintaxis y la semántica de Python. Es una lectura pesada, pero es muy útil como guía complete del lenguaje.

Más recursos sobre Python:

- https://www.python.org: The major Python website. It contains code, documentation, and pointers to Python-related pages around the web.
- https://docs.python.org: Acceso rápido a la documentación de Python.
- https://pypi.org: El Python Package Index, apodado previamente la Tienda de Queso [1], es un índice de módulos

de Python creados por usuarios que están disponibles para su descarga. Cuando empiezas a distribuir código, lo puedes registrar allí para que otros lo encuentren.
- https://code.activestate.com/recipes/langs/python/: El Python Cookbook es una gran colección de ejemplos de código, módulos grandes y scripts útiles. Las contribuciones más notables también están recogidas en un libro titulado Python Cookbook (O'Reilly & Associates, ISBN 0-596-00797-3.)
- http://www.pyvideo.org recoge enlaces a vídeos relacionados con Python provenientes de conferencias y reuniones de grupos de usuarios.
- https://scipy.org: El proyecto de Python científico incluye módulos para el cálculo rápido de operaciones y manipulaciones sobre arrays además de muchos paquetes para cosas como Álgebra Lineal, Transformadas de Fourier, solucionadores de sistemas no-lineales, distribuciones de números aleatorios, análisis estadísticos y otras herramientas.

Para preguntas relacionadas con Python y reportes de problemas puedes escribir al grupo de noticias *comp.lang.python*, o enviarlas a la lista de correo que hay en python-list@python.org. El grupo de noticias y la lista de correo están conectados, por lo que los mensajes enviados a uno serán retransmitidos al otro. Hay alrededor de cientos de mensajes diarios (con picos de hasta varios cientos), haciendo (y respondiendo) preguntas, sugiriendo nuevas características, y anunciando nuevos módulos. Los archivos de la lista de correos están disponibles en https://mail.python.org/pipermail/.

Antes de escribir, asegúrate de haber revisado la lista de Frequently Asked Questions (también llamado el *FAQ*). Muchas veces responde las preguntas que se hacen una y otra vez, y quizás contenga la solución a tu problema.

Notas al pie

«Cheese Shop» is a Monty Python sketch: a customer enters a cheese shop, but whatever cheese he asks for, the clerk says it's missing.

14. Edición de entrada interactiva y sustitución de historial

Algunas versiones del intérprete de Python permiten editar la línea de entrada actual, y sustituir en base al historial, de forma similar a las capacidades del intérprete de comandos *Korn* y el GNU bash. Esto se implementa con la biblioteca GNU Readline [https://tiswww.case.edu/php/chet/readline/rltop.html], que soporta varios estilos de edición. Esta biblioteca tiene su propia documentación la cuál no vamos a duplicar aquí.

14.1. Autocompletado con tab e historial de edición

El autocompletado de variables y nombres de módulos es automatically enabled al iniciar el intérprete, por lo tanto la tecla `Tab` invoca la función de autocompletado; ésta mira en los nombres de sentencia, las variables locales y los nombres de módulos disponibles. Para expresiones con puntos como `string.a`, va a evaluar la expresión hasta el `'.'` final y entonces sugerir autocompletado para los atributos del objeto resultante. Nota que esto quizás ejecute código de aplicaciones definidas si un objeto con un método **`__getattr__()`** es parte de la expresión. La configuración por omisión también guarda tu historial en un archivo llamado `.python_history` en tu directorio de usuario. El historial estará disponible durante la próxima sesión interactiva del intérprete.

14.2. Alternativas al intérprete interactivo

This facility is an enormous step forward compared to earlier versions of the interpreter; however, some wishes are left: It would be nice if the proper indentation were suggested on continuation lines (the parser knows if an `INDENT` token is required next). The completion mechanism might use the interpreter's symbol table. A command to check (or even suggest) matching parentheses, quotes, etc., would also be useful.

Un intérprete interactivo mejorado alternativo que está dando vueltas desde hace rato es IPython [https://ipython.org/], que ofrece completado por tab, exploración de objetos, y administración avanzada del historial. También puede ser configurado en profundidad, e integrarse en otras aplicaciones. Otro entorno interactivo mejorado similar es bpython [https://bpython-interpreter.org/].

15. Floating-Point Arithmetic: Issues and Limitations

Los números de punto flotante se representan en el hardware de computadoras como fracciones en base 2 (binarias). Por ejemplo, la fracción **decimal** `0.625` tiene un valor de 6/10 + 2/100 + 5/1000, y de la misma manera, la fracción binaria `0.101` tiene un valor de 1/2 + 0/4 + 1/8. Estas dos fracciones tienen valores idénticos; la única diferencia real radica en que la primera se escribe en notación fraccional en base 10, y la segunda en base 2.

Desafortunadamente, la mayoría de las fracciones decimales no pueden representarse exactamente como fracciones binarias. Como consecuencia, en general los números de punto flotante decimal que ingresás en la computadora son sólo aproximados por los números de punto flotante binario que realmente se guardan en la máquina.

El problema es más fácil de entender primero en base 10. Considerá la fracción 1/3. Podés aproximarla como una fracción de base 10

```
0.3
```

...o, mejor,

```
0.33
```

...o, mejor,

```
0.333
```

...y así. No importa cuantos dígitos desees escribir, el resultado nunca será exactamente 1/3, pero será una aproximación cada vez mejor de 1/3.

De la misma manera, no importa cuantos dígitos en base 2 quieras usar, el valor decimal 0.1 no puede representarse exactamente como una fracción en base 2. En base 2, 1/10 es la siguiente fracción que se repite infinitamente:

```
0.0001100110011001100110011001100110011001100110011...
```

Frená en cualquier número finito de bits, y tendrás una aproximación. En la mayoría de las máquinas hoy en día, los float se aproximan usando una fracción binaria con el numerador usando los primeros 53 bits con el bit más significativos y el denominador como una potencia de dos. En el caso de 1/10, la fracción binaria es `3602879701896397 / 2 ** 55` que está cerca pero no es exactamente el valor verdadero de 1/10.

La mayoría de los usuarios no son conscientes de esta aproximación por la forma en que se muestran los valores. Python solamente muestra una aproximación decimal al valor verdadero decimal de la aproximación binaria almacenada por la máquina. En la mayoría de las máquinas, si Python fuera a imprimir el verdadero valor decimal de la aproximación binaria almacenada para 0.1, debería mostrar:

```
>>> 0.1
0.100000000000000005551115123125782702118158340454101562
```

Esos son más dígitos que lo que la mayoría de la gente encuentra útil, por lo que Python mantiene manejable la cantidad de dígitos al mostrar un valor redondeado en su lugar:

```
>>> 1 / 10
0.1
```

Sólo recordá que, a pesar de que el valor mostrado resulta ser exactamente 1/10, el valor almacenado realmente es la fracción binaria más cercana posible.

Interesantemente, hay varios números decimales que comparten la misma fracción binaria más aproximada. Por ejemplo, los números `0.1`, `0.10000000000000001` y `0.100000000000000005551115123125782702118158340454101562` son todos aproximados por `3602879701896397 / 2 ** 55`. Ya que todos estos valores decimales comparten la misma aproximación, se podría mostrar cualquiera de ellos para preservar el invariante `eval(repr(x)) == x`.

Históricamente, el prompt de Python y la función integrada **`repr()`** eligieron el valor con los 17 dígitos, `0.10000000000000001`. Desde Python 3.1, en la mayoría de los sistemas Python ahora es capaz de elegir la forma más corta de ellos y mostrar `0.1`.

Note that this is in the very nature of binary floating point: this is not a bug in Python, and it is not a bug in your code either. You'll see the same kind of thing in all languages that support your hardware's floating-point arithmetic (although some languages may not *display* the difference by default, or in all output modes).

Para una salida más elegante, quizás quieras usar el formateo de cadenas de texto para generar un número limitado de dígitos significativos:

```
>>> format(math.pi, '.12g')  # give 12 significant digit
'3.14159265359'

>>> format(math.pi, '.2f')   # give 2 digits after the p
'3.14'

>>> repr(math.pi)
'3.141592653589793'
```

Es importante darse cuenta que esto es, realmente, una ilusión: estás simplemente redondeando al *mostrar* el valor verdadero de la máquina.

Una ilusión puede generar otra. Por ejemplo, ya que 0.1 no es exactamente 1/10, sumar tres veces 0.1 podría también no generar exactamente 0.3:

```
>>> 0.1 + 0.1 + 0.1 == 0.3
False
```

También, ya que 0.1 no puede acercarse más al valor exacto de 1/10 y 0.3 no puede acercarse más al valor exacto de 3/10, redondear primero con la función **`round()`** no puede ayudar:

```
>>> round(0.1, 1) + round(0.1, 1) + round(0.1, 1) == rou
False
```

Aunque los números no pueden acercarse más a sus valores exactos previstos, la función **`math.isclose()`** puede ser útil para comparar valores inexactos:

```
>>> math.isclose(0.1 + 0.1 + 0.1, 0.3)
True
```

Alternatively, the **`round()`** function can be used to compare rough approximations:

```
>>> round(math.pi, ndigits=2) == round(22 / 7, ndigits=2
True
```

Binary floating-point arithmetic holds many surprises like this. The problem with «0.1» is explained in precise detail below, in the «Representation Error» section. See Examples of Floating Point Problems [https://jvns.ca/blog/2023/01/13/examples-of-floating-point-problems/] for a pleasant summary of how binary floating point works and the kinds of problems commonly encountered in practice. Also see The Perils of Floating Point [http://www.indowsway.com/floatingpoint.htm] for a more complete account of other common surprises.

As that says near the end, «there are no easy answers.» Still, don't be unduly wary of floating point! The errors in Python float operations are inherited from the floating-point hardware, and on most machines are on the order of no more than 1 part in 2**53 per operation. That's more than adequate for most tasks, but you do need to keep in mind that it's not decimal arithmetic and that every float operation can suffer a new rounding error.

A pesar de que existen casos patológicos, para la mayoría de usos casuales de la aritmética de punto flotante al final verás el resultado que esperás si simplemente redondeás lo que mostrás de tus resultados finales al número de dígitos decimales que esperás. **`str()`** es normalmente suficiente, y para un control más fino mirá los parámetros del método de formateo **`str.format()`** en Custom string formatting.

Para los casos de uso que necesitan una representación decimal exacta, probá el módulo **`decimal`**, que implementa aritmética decimal útil para aplicaciones de contabilidad y de alta precisión.

El módulo **`fractions`** soporta otra forma de aritmética exacta, ya que implementa aritmética basada en números racionales (por lo que números como 1/3 pueden ser representados exactamente).

Si eres un usuario intensivo de operaciones de punto flotante, deberías echar un vistazo al paquete NumPy y a muchos otros paquetes para operaciones matemáticas y estadísticas proporcionados por el proyecto SciPy. Ver <https://scipy.org>.

Python provee herramientas que pueden ayudar en esas raras ocasiones cuando realmente *querés* saber el valor exacto de un punto flotante. El método **`float.as_integer_ratio()`** expresa el valor del punto flotante como una fracción:

```
>>> x = 3.14159
>>> x.as_integer_ratio()
(3537115888337719, 1125899906842624)
```

Ya que la fracción es exacta, se puede usar para recrear sin pérdidas el valor original:

```
>>> x == 3537115888337719 / 1125899906842624
True
```

El método **`float.hex()`** expresa un punto flotante en hexadecimal (base 16), nuevamente retornando el valor exacto almacenado por tu computadora:

```
>>> x.hex()
'0x1.921f9f01b866ep+1'
```

Esta representación hexadecimal precisa se puede usar para reconstruir el valor exacto del punto flotante:

```
>>> x == float.fromhex('0x1.921f9f01b866ep+1')
True
```

Ya que la representación es exacta, es útil para portar valores a través de diferentes versiones de Python de manera confiable (independencia de plataformas) e intercambiar datos con otros lenguajes que soportan el mismo formato (como Java y C99).

Otra herramienta útil es la función **`sum()`** que ayuda a mitigar la pérdida de precisión durante la suma. Utiliza precisión extendida para pasos de redondeo intermedios a medida que se agregan valores a un total en ejecución. Esto puede marcar la diferencia en la precisión general para que los errores no se acumulen hasta el punto en que afecten el total final:

```
>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
False
>>> sum([0.1] * 10) == 1.0
True
```

The **`math.fsum()`** goes further and tracks all of the «lost digits» as values are added onto a running total so that the result has only a single rounding. This is slower than **`sum()`** but will be more accurate in uncommon cases where large magnitude inputs mostly cancel each other out leaving a final sum near zero:

```
>>> arr = [-0.10430216751806065, -266310978.67179024, 14
...         -143401161400469.7, 266262841.31058735, -0.00
>>> float(sum(map(Fraction, arr)))   # Exact summation w
8.042173697819788e-13
>>> math.fsum(arr)                   # Single rounding
8.042173697819788e-13
>>> sum(arr)                         # Multiple roundin
8.042178034628478e-13
>>> total = 0.0
>>> for x in arr:
...     total += x                   # Multiple roundin
...
>>> total                            # Straight additio
-0.0051575902860057365
```

15.1. Error de Representación

Esta sección explica el ejemplo «0.1» en detalle, y muestra como en la mayoría de los casos vos mismo podés realizar un análisis exacto como este. Se asume un conocimiento básico de la representación de punto flotante binario.

Error de representación se refiere al hecho de que algunas (la mayoría) de las fracciones decimales no pueden representarse exactamente como fracciones binarias (en base 2). Esta es la razón principal de por qué Python (o Perl, C, C++, Java, Fortran, y tantos otros) frecuentemente no mostrarán el número decimal exacto que esperás.

¿Por qué sucede esto? 1/10 no es exactamente representable como una fracción binaria. Desde al menos el año 2000, casi todas las máquinas utilizan la aritmética de punto flotante binaria IEEE 754, y casi todas las plataformas asignan los números de punto flotante de Python a valores binarios de 64 bits de precisión «doble» de IEEE 754. Los valores binarios de IEEE 754 de 64 bits contienen 53 bits de precisión, por lo que en la entrada, la computadora se esfuerza por convertir 0.1 en la fracción más cercana de la forma *J*/2***N* donde *J* es un número entero que contiene exactamente 53 bits. Reescribiendo

```
1 / 10 ~= J / (2**N)
```

...como

```
J ~= 2**N / 10
```

...y recordando que *J* tiene exactamente 53 bits (es `>= 2**52` pero `< 2**53`), el mejor valor para *N* es 56:

```
>>> 2**52 <=  2**56 // 10  < 2**53
True
```

O sea, 56 es el único valor para *N* que deja *J* con exactamente 53 bits. El mejor valor posible para *J* es entonces el cociente redondeado:

```
>>> q, r = divmod(2**56, 10)
>>> r
```

Ya que el resto es más que la mitad de 10, la mejor aproximación se obtiene redondeándolo:

```
>>> q+1
7205759403792794
```

Por lo tanto la mejor aproximación a 1/10 en doble precisión IEEE 754 es:

```
7205759403792794 / 2 ** 56
```

El dividir tanto el numerador como el denominador reduce la fracción a:

```
3602879701896397 / 2 ** 55
```

Notá que como lo redondeamos, esto es un poquito más grande que 1/10; si no lo hubiéramos redondeado, el cociente hubiese sido un poquito menor que 1/10. ¡Pero no hay caso en que sea *exactamente* 1/10!

Entonces la computadora nunca «ve» 1/10: lo que ve es la fracción exacta de arriba, la mejor aproximación al flotante doble IEEE 754 que puede obtener:

```
>>> 0.1 * 2 ** 55
3602879701896397.0
```

Si multiplicamos esa fracción por 10**55, podemos ver el valor hasta los 55 dígitos decimales:

```
>>> 3602879701896397 * 10 ** 55 // 2 ** 55
1000000000000000055511151231257827021181583404541015625
```

lo que significa que el valor exacto almacenado en la computadora es igual al valor decimal 0.1000000000000000055511151231257827021181583404541015625. En lugar de mostrar el valor decimal completo, muchos lenguajes (incluyendo versiones anteriores de Python), redondean el resultado a 17 dígitos significativos:

```
>>> format(0.1, '.17f')
'0.10000000000000001'
```

Los módulos `fractions` y `decimal` hacen fácil estos cálculos:

```
>>> from decimal import Decimal
>>> from fractions import Fraction

>>> Fraction.from_float(0.1)
Fraction(3602879701896397, 36028797018963968)

>>> (0.1).as_integer_ratio()
(3602879701896397, 36028797018963968)

>>> Decimal.from_float(0.1)
Decimal('0.10000000000000000555111512312578270211815834O

>>> format(Decimal.from_float(0.1), '.17')
'0.10000000000000001'
```

[1]

16. Apéndice

16.1. Modo interactivo

There are two variants of the interactive REPL. The classic basic interpreter is supported on all platforms with minimal line control capabilities.

Since Python 3.13, a new interactive shell is used by default. This one supports color, multiline editing, history browsing, and paste mode. To disable color, see Controlling color for details. Function keys provide some additional functionality. `F1` enters the interactive help browser **`pydoc`**. `F2` allows for browsing command-line history with neither output nor the >>> and ... prompts. `F3` enters «paste mode», which makes pasting larger blocks of code easier. Press `F3` to return to the regular prompt.

When using the new interactive shell, exit the shell by typing `exit` or `quit`. Adding call parentheses after those commands is not required.

If the new interactive shell is not desired, it can be disabled via the **`PYTHON_BASIC_REPL`** environment variable.

16.1.1. Manejo de errores

When an error occurs, the interpreter prints an error message and a stack trace. In interactive mode, it then returns to the primary prompt; when input came from a file, it exits with a nonzero exit status after printing the stack trace. (Exceptions handled by an **`except`** clause in a **`try`** statement are not errors in this context.) Some errors are unconditionally fatal and cause an exit with a nonzero exit status; this applies to internal inconsistencies and some cases of running out of memory. All error messages are written to the standard error stream; normal output from executed commands is written to standard output.

Al ingresar el carácter de interrupción (por lo general `Control-C` o `Supr`) en el prompt primario o secundario, se cancela la entrada y retorna al prompt primario. [1] Tipear una interrupción mientras un comando se están ejecutando lanza la excepción **`KeyboardInterrupt`**, que puede ser manejada con una sentencia **`try`**.

16.1.2. Programas ejecutables de Python

En los sistemas Unix y tipo BSD, los programas Python pueden convertirse directamente en ejecutables, como programas del intérprete de comandos, poniendo la linea:

```
#!/usr/bin/env python3
```

…al principio del script y dándole al archivo permisos de ejecución (asumiendo que el intérprete están en la variable de entorno **`PATH`** del usuario). `#!` deben ser los primeros dos caracteres del archivo. En algunas plataformas, la primera línea debe terminar al estilo Unix (`'\n'`), no como en Windows (`'\r\n'`). Notá que el carácter numeral `'#'` se usa en Python para comenzar un comentario.

Se le puede dar permisos de ejecución al script usando el comando **chmod**.

```
$ chmod +x myscript.py
```

En sistemas Windows, no existe la noción de «modo ejecutable». El instalador de Python asocia automáticamente la extensión `.py` con `python.exe` para que al hacerle doble clic a un archivo Python se corra el script. La extensión también puede ser `.pyw`, en este caso se omite la ventana con la consola que normalmente aparece.

16.1.3. El archivo de inicio interactivo

Cuando usas Python en forma interactiva, suele ser útil que algunos comandos estándar se ejecuten cada vez que el intérprete se inicia. Puedes hacer esto configurando la variable de entorno **`PYTHONSTARTUP`** con el nombre de un archivo que contenga tus comandos de inicio. Esto es similar al archivo `.profile` en los intérpretes de comandos de Unix.

Este archivo es solo leído en las sesiones interactivas del intérprete, no cuando Python lee comandos de un script ni cuando `/dev/tty` se explicita como una fuente de comandos (que de otro modo se comporta como una sesión interactiva). Se ejecuta en el mismo espacio de nombres en el que los comandos interactivos se ejecutan, entonces los objetos que define o importa pueden ser usados sin cualificaciones en la sesión interactiva. En este archivo también puedes cambiar los prompts `sys.ps1` y `sys.ps2`.

Si quieres leer un archivo de inicio adicional desde el directorio actual, puedes programarlo en el archivo de inicio global usando algo como `if os.path.isfile('.pythonrc.py'): exec(open('.pythonrc.py').read())`. Si quieres usar el archivo de inicio en un script, tienes que hacer lo siguiente de forma explícita en el script:

```
import os
filename = os.environ.get('PYTHONSTARTUP')
if filename and os.path.isfile(filename):
    with open(filename) as fobj:
        startup_file = fobj.read()
    exec(startup_file)
```

16.1.4. Los módulos de customización

Python provee dos formas para customizarlo: sitecustomize y usercustomize. Para ver cómo funciona, necesitas primero encontrar dónde está tu directorio para tu usuario de paquetes del sistema. Inicia Python y ejecuta el siguiente código:

```
>>> import site
>>> site.getusersitepackages()
'/home/user/.local/lib/python3.x/site-packages'
```

Ahora puedes crear un archivo llamado `usercustomize.py` en ese directorio y poner lo que quieras en él. Eso afectará cada ejecución de Python, a menos que se inicie con la opción **`-s`** para deshabilitar esta importación automática.

sitecustomize funciona de la misma manera, pero normalmente lo crea el administrador de la computadora en el directorio global de paquetes del sistema, y se importa antes que usercustomize. Para más detalles, mira la documentación del módulo `site`.

Notas al pie

Un problema con el paquete GNU Readline puede prevenir esto.

Configuración y uso de Python

Esta parte de la documentación está dedicada a información general sobre la configuración del entorno Python en diferentes plataformas, la invocación del intérprete y cosas que facilitan el trabajo con Python.

- 1. Línea de comandos y entorno
    - 1.1. Línea de comando
    - 1.2. Variables de entorno
- 2. Uso de Python en plataformas Unix
    - 2.1. Obteniendo e instalando la última versión de Python
    - 2.2. Construyendo Python
    - 2.3. Rutas y archivos relacionados con Python
    - 2.4. Miscelánea
    - 2.5. OpenSSL personalizado
- 3. Configurar Python
    - 3.1. Requisitos de compilación
    - 3.2. Archivos generados
    - 3.3. Configurar opciones
    - 3.4. Sistema de compilación Python
    - 3.5. Banderas de compilador y vinculación
- 4. Uso de Python en Windows
    - 4.1. Python install manager
    - 4.2. El paquete incrustable
    - 4.3. El paquete de nuget.org
    - 4.4. Distribuciones alternativas
    - 4.5. Supported Windows versions
    - 4.6. Removing the MAX_PATH limitation

  - 4.7. Modo UTF-8
  - 4.8. Encontrar módulos
  - 4.9. Módulos adicionales
  - 4.10. Compilar Python en Windows
  - 4.11. The full installer (deprecated)
  - 4.12. Python launcher for Windows (deprecated)

- 5. Using Python on macOS

  - 5.1. Using Python for macOS from **`python.org`**
  - 5.2. Alternative Distributions
  - 5.3. Instalación de paquetes adicionales de Python
  - 5.4. GUI Programming
  - 5.5. Advanced Topics
  - 5.6. Otros recursos

- 6. Using Python on Android

  - 6.1. Adding Python to an Android app
  - 6.2. Building a Python package for Android

- 7. Using Python on iOS

  - 7.1. Python at runtime on iOS
  - 7.2. Installing Python on iOS
  - 7.3. App Store Compliance

- 8. Editores e IDEs

  - 8.1. IDLE — Python editor and shell
  - 8.2. Other Editors and IDEs

1. Línea de comandos y entorno

El intérprete de CPython analiza la línea de comandos y el entorno en busca de varias configuraciones.

Los esquemas de línea de comandos de otras implementaciones pueden diferir. Véase Implementaciones alternativas para obtener más recursos.

1.1. Línea de comando

Al invocar Python, puede especificar cualquiera de estas opciones:

```
python [-bBdEhiIOPqRsSuvVWx?] [-c command | -m module-na
```

El caso de uso más común es, por supuesto, una simple invocación de un script:

```
python myscript.py
```

1.1.1. Opciones de interfaz

La interfaz del intérprete es similar a la del shell UNIX, pero proporciona algunos métodos adicionales de invocación:

- When called with standard input connected to a tty device, it prompts for commands and executes them until an EOF (an end-of-file character, you can produce that with `Ctrl-D` on UNIX or `Ctrl-Z, Enter` on Windows) is read. For more on interactive mode, see Modo interactivo.
- Cuando se llama con un argumento de nombre de archivo o con un archivo como entrada estándar, lee y ejecuta un script de ese archivo.
- Cuando se llama con un argumento de nombre de directorio,

lee y ejecuta un script con el nombre adecuado desde ese directorio.

- Cuando se llama con `-c comando`, ejecuta las instrucciones de Python dadas como *command*. Aquí *comando* puede contener varias instrucciones separadas por nuevas líneas. ¡El espacio en blanco principal es significativo en las instrucciones de Python!
- Cuando se llama con `-m module-name`, el módulo dado se encuentra en la ruta del módulo Python y se ejecuta como un script.

En el modo no interactivo, toda la entrada se analiza antes de ejecutarse.

Una opción de interfaz termina la lista de opciones consumidas por el intérprete, todos los argumentos consecutivos terminarán en **`sys.argv`** – tenga en cuenta que el primer elemento, subíndice cero (`sys.argv[0]`), es una cadena que refleja el origen del programa.

-c <command>

Ejecute el código de Python en *comando*. *comando* puede ser una o más sentencias separadas por nuevas líneas, con espacio en blanco inicial significativo como en el código normal del módulo.

Si se proporciona esta opción, el primer elemento de **`sys.argv`** será `"-c"` y el directorio actual se agregará al inicio de **`sys.path`** (permitiendo que los módulos de ese directorio se importen como módulos de nivel superior).

Lanza un auditing event `cpython.run_command` con el argumento `command`.

Distinto en la versión 3.14: command is automatically dedented before execution.

-m <module-name>

Busque **`sys.path`** para el módulo con nombre y ejecute su contenido como el módulo **`__main__`**.

Dado que el argumento es un nombre *módulo*, no debe dar una extensión de archivo (`.py`). El nombre del módulo debe ser un nombre de módulo Python absoluto válido, pero es posible que la implementación no siempre lo aplique (por ejemplo, puede permitirle usar un nombre que incluya un guión).

También se permiten los nombres de paquetes (incluidos los paquetes de espacio de nombres). Cuando se proporciona un nombre de paquete en lugar de un módulo normal, el intérprete ejecutará `<pkg>.__main__` como módulo principal. Este comportamiento es deliberadamente similar al manejo de directorios y archivos zip que se pasan al intérprete como argumento del script.

Nota

Esta opción no se puede utilizar con módulos integrados y módulos de extensión escritos en C, ya que no tienen archivos de módulo Python. Sin embargo, todavía se puede utilizar para módulos precompilados, incluso si el archivo de origen original no está disponible.

Si se da esta opción, el primer elemento de **`sys.argv`** será la ruta de acceso completa al archivo de módulo (mientras se encuentra el archivo de módulo, el primer elemento se establecerá en `"-m"`). Al igual que con la opción **`-c`**, el directorio actual se agregará al inicio de **`sys.path`**.

`-I` option can be used to run the script in isolated mode where **`sys.path`** contains neither the current directory nor the user’s site-packages directory. All `PYTHON*` environment variables are ignored, too.

Muchos módulos de biblioteca estándar contienen código que se invoca en su ejecución como script. Un ejemplo es el módulo **`timeit`**:

```
python -m timeit -s "setup here" "benchmarked code
python -m timeit -h # for details
```

Retorna un auditing event `cpython.run_module` con el argumento `nombre-módulo`.

Ver también

`runpy.run_module()`
Funcionalidad equivalente directamente disponible para el código Python

PEP 338 [https://peps.python.org/pep-0338/] – Ejecución de módulos como scripts

Distinto en la versión 3.1: Proporcione el nombre del paquete para ejecutar un submódulo `__main__`.

Distinto en la versión 3.4: paquetes de espacio de nombres también son compatibles

-

Leer comandos de entrada estándar (**`sys.stdin`**). Si la entrada estándar es un terminal, **`-i`** está implícita.

Si se da esta opción, el primer elemento de **`sys.argv`** será `"-"` y el directorio actual se agregará al inicio de **`sys.path`**.

Lanza un evento auditing event `cpython.run_stdin` sin argumentos.

<script>

Ejecute el código Python contenido en *script*, que debe ser una ruta de acceso del sistema de archivos (absoluta o relativa) que haga referencia a un archivo Python, un directorio que contenga un archivo `__main__.py` o un archivo zip que contenga un archivo `__main__.py`.

Si se proporciona esta opción, el primer elemento de **`sys.argv`** será el nombre del script como se indica en la línea de comandos.

Si el nombre del script hace referencia directamente a un archivo Python, el directorio que contiene ese archivo se agrega al inicio de **`sys.path`**, y el archivo se ejecuta como el módulo **`__main__`**.

Si el nombre del script hace referencia a un directorio o zipfile, el nombre del script se agrega al inicio de **`sys.path`** y el archivo `__main__.py` en esa ubicación se ejecuta como el módulo **`__main__`**.

`-I` option can be used to run the script in isolated mode where **`sys.path`** contains neither the script's directory nor the user's site-packages directory. All `PYTHON*` environment variables are ignored, too.

Lanza un auditing event `cpython.run_file` con el argumento `filename`.

Ver también

`runpy.run_path()`
Funcionalidad equivalente directamente disponible para el código Python

Si no se da ninguna opción de interfaz, **`-i`** está implícita, `sys.argv[0]` es una cadena vacía (`""`) y el directorio actual se agregará al inicio de **`sys.path`**. Además, la finalización de tabulación y la edición del historial se habilitan automáticamente, si están disponibles en su plataforma (consulte Configuración de Readline).

Ver también

Invocar el intérprete

Distinto en la versión 3.4: Habilitación automática de la finalización de pestañas y edición del historial.

1.1.2. Opciones genéricas

-?
-h
--help

Imprime una breve descripción de todas las opciones de la línea de comandos.

--help-env

Imprima una breve descripción de las variables de entorno específicas de Python y salga.

Added in version 3.11.

--help-xoptions

Imprima una descripción de las opciones `-X` específicas de la implementación y salga.

Added in version 3.11.

--help-all

Imprima información completa de uso y salga.

Added in version 3.11.

-V
--version

Imprima el número de versión de Python y salga. Ejemplo de salida podría ser:

```
Python 3.8.0b2+
```

Cuando se le dé dos veces, imprima más información sobre la compilación, como:

```
Python 3.8.0b2+ (3.8:0c076caaa8, Apr 20 2019, 21:55
[GCC 6.2.0 20161005]
```

Added in version 3.6: La opción `-VV`.

1.1.3. Opciones diversas

-b

Issue a warning when converting **`bytes`** or **`bytearray`** to **`str`** without specifying encoding or comparing **`bytes`** or **`bytearray`** with **`str`** or **`bytes`** with **`int`**. Issue an error when the option is given twice (**`-bb`**).

Distinto en la versión 3.5: Affects also comparisons of **`bytes`** with **`int`**.

-B

Si se da, Python no intentará escribir archivos `.pyc` en la importación de módulos de origen. Véase también **`PYTHONDONTWRITEBYTECODE`**.

--check-hash-based-pycs default|always|never

Controle el comportamiento de validación de los archivos `.pyc` basados en hash. Véase Invalidación del código de bytes en caché. Cuando se establece en `default`, los archivos de caché de código de bytes basados en hash marcados y desmarcados se validan según su semántica predeterminada. Cuando se establece en `always`, todos los archivos basados en hash `.pyc`, ya estén marcados o desmarcados, se validan con su archivo de origen correspondiente. Cuando se establece en `never`, los archivos basados en hash `.pyc` no se validan con sus archivos de origen correspondientes.

Esta opción no afecta a la semántica de los archivos `.pyc` basados en la marca de tiempo.

-d

Activa la salida de depuración (solo para expertos, dependiendo de las opciones de compilación). Véase también la variable de ambiente **`PYTHONDEBUG`**.

Esta opción necesita una compilación de depuración de Python, de lo contrario será ignorado.

-E

Ignore all `PYTHON*` environment variables, e.g.

`PYTHONPATH` and **`PYTHONHOME`**, that might be set.

Véase también las opciones **`-P`** e **`-I`** (aisladas).

-i

Enter interactive mode after execution.

Using the **`-i`** option will enter interactive mode in any of the following circumstances:

- When a script is passed as first argument
- When the **`-c`** option is used
- When the **`-m`** option is used

Interactive mode will start even when **`sys.stdin`** does not appear to be a terminal. The **`PYTHONSTARTUP`** file is not read.

Esto puede ser útil para inspeccionar variables globales o un seguimiento de pila cuando un script lanza una excepción. Véase también **`PYTHONINSPECT`**.

-I

Ejecute Python en modo aislado. Esto también implica las opciones **`-E`**, **`-P`** y **`-s`**.

In isolated mode **`sys.path`** contains neither the script's directory nor the user's site-packages directory. All `PYTHON*` environment variables are ignored, too. Further restrictions may be imposed to prevent the user from injecting malicious code.

Added in version 3.4.

-O

Quite las instrucciones assert y cualquier código condicionado al valor de **`__debug__`**. Aumente el nombre de archivo para los archivos compilados (bytecode) agregando `.opt-1` antes de la extensión `.pyc` (consulte **PEP 488** [https://peps.python.org/pep-0488/]). Véase también **`PYTHONOPTIMIZE`**.

Distinto en la versión 3.5: Modifique los nombres de archivo `.pyc` según **PEP 488** [https://peps.python.org/pep-0488/].

-OO

Haga **`-O`** y también deseche las docstrings. Aumente el nombre de archivo para los archivos compilados (bytecode) agregando `.opt-2` antes de la extensión `.pyc` (consulte **PEP 488** [https://peps.python.org/pep-0488/]).

Distinto en la versión 3.5: Modifique los nombres de archivo `.pyc` según **PEP 488** [https://peps.python.org/pep-0488/].

-P

No anteponga una ruta potencialmente insegura a **`sys.path`**:

- `python -m module` command line: No anteponga el directorio de trabajo actual.
- `python script.py` command line: No anteponga el directorio del script. Si es un enlace simbólico, resuelva los enlaces simbólicos.
- `python -c code` y `python` (REPL) command lines: No anteponga una cadena vacía, lo que significa el directorio de trabajo actual.

Consulte también la variable de entorno **`PYTHONSAFEPATH`** y las opciones **`-E`** y **`-I`** (aisladas).

Added in version 3.11.

-q

No muestres los mensajes de copyright y versión incluso en modo interactivo.

Added in version 3.2.

-R

Turn on hash randomization. This option only has an effect if the **`PYTHONHASHSEED`** environment variable is set to anything other than `random`, since hash randomization is enabled by default.

On previous versions of Python, this option turns on hash randomization, so that the `__hash__()` values of str and bytes objects are «salted» with an unpredictable random value. Although they remain constant within an individual Python process, they are not predictable between repeated invocations of Python.

Hash randomization is intended to provide protection against a denial-of-service caused by carefully chosen inputs that exploit the worst case performance of a dict construction, $O(n^2)$ complexity. See https://ocert.org/advisories/ocert-2011-003.html for details.

`PYTHONHASHSEED` le permite establecer un valor fijo para el secreto de inicialización hash.

Added in version 3.2.3.

Distinto en la versión 3.7: La opción ya no se omite.

-s

No agregue el `user site-packages directory` a `sys.path`.

See also `PYTHONNOUSERSITE`.

Ver también

PEP 370 [https://peps.python.org/pep-0370/] – Directorio de paquetes de sitio por usuario

-S

Deshabilite la importación del módulo `site` y las manipulaciones dependientes del sitio de `sys.path` que conlleva. También deshabilite estas manipulaciones si `site` se importa explícitamente más tarde (llame a `site.main()`

si desea que se activen).

-u

Forzar que las corrientes stdout y stderr no estén en búfer. Esta opción no tiene ningún efecto en la secuencia stdin.

Véase también **`PYTHONUNBUFFERED`**.

Distinto en la versión 3.7: La capa de texto de las secuencias stdout y stderr ahora no está en búfer.

-v

Imprime un mensaje cada vez que se inicialice un módulo, mostrando el lugar (nombre de archivo o módulo integrado) desde el que se carga. Cuando se le pasa dos veces (**`-vv`**), imprime un mensaje para cada archivo que se comprueba al buscar un módulo. También proporciona información sobre la limpieza del módulo en la salida.

Distinto en la versión 3.10: El módulo **`site`** informa las rutas específicas del sitio y los archivos `.pth` que se están procesando.

Véase también **`PYTHONVERBOSE`**.

-W arg

Control de advertencia. La maquinaria de advertencia de Python por defecto imprime mensajes de advertencia en **`sys.stderr`**.

La configuración más sencilla aplica una acción determinada incondicionalmente a todas las advertencias emitidas por un proceso (incluso aquellas que de otro modo se ignoran de forma predeterminada):

```
-Wdefault  # Warn once per call location
-Werror    # Convert to exceptions
-Walways   # Warn every time
-Wall      # Same as -Walways
-Wmodule   # Warn once per calling module
```

```
-Wonce     # Warn once per Python process
-Wignore   # Never warn
```

Los nombres de acción se pueden abreviar como se desee y el intérprete los resolverá con el nombre de acción adecuado. Por ejemplo, `-Wi` es lo mismo que `-Wignore`.

La forma completa del argumento es:

```
action:message:category:module:lineno
```

Los campos vacíos cuadran con todos los valores; los campos vacíos finales pueden omitirse. Por ejemplo, `-W ignore::DeprecationWarning` ignora todas las advertencias de DeprecationWarning.

El campo *action* es como se explicó anteriormente, pero solo se aplica a las advertencias que coinciden con los campos restantes.

El campo *message* debe coincidir con el mensaje de advertencia completo; esta coincidencia no distingue entre mayúsculas y minúsculas.

El campo *category* coincide con la categoría de advertencia (por ej. `DeprecationWarning`). Debe ser un nombre de clase; la coincidencia prueba si la categoría de advertencia real del mensaje es una subclase de la categoría de advertencia especificada.

El campo *module* coincide con el nombre de dominio completo del módulo; esta coincidencia distingue entre mayúsculas y minúsculas.

El campo *lineno* coincide con el número de línea, donde cero corresponde a todos los números de línea y, por lo tanto, es equivalente a número de línea omitido.

Se pueden dar varias opciones **`-W`**; cuando una advertencia coincide con más de una opción, se realiza la acción para la última opción de coincidencia. Las opciones invalidas **`-W`** son ignoradas (aunque se imprime un mensaje de advertencia sobre opciones no válidas cuando se emite la primera advertencia).

Las advertencias también se pueden controlar utilizando la variable de entorno **`PYTHONWARNINGS`** y desde un programa Python utilizando el módulo **`warnings`**. Por ejemplo, la función **`warnings.filterwarnings()`** puede ser usarse con una expresión regular en el mensaje de advertencia.

Consulte El filtro de advertencias y Descripción de los filtros de advertencia para obtener más detalles.

-x

Omita la primera línea de la fuente, permitiendo el uso de formas que no sean de Unix de `#!cmd`. Esto está destinado a un hackeo específico de DOS solamente.

-X

Reservado para varias opciones específicas de la implementación. CPython define actualmente los siguientes valores posibles:

- `-X faulthandler` para habilitar **`faulthandler`**. Véase **`PYTHONFAULTHANDLER`** para obtener más información.

  Added in version 3.3.

- `-X showrefcount` para generar el recuento total de referencias y el número de bloques de memoria utilizados cuando finalice el programa o después de cada instrucción en el intérprete interactivo. Esto sólo funciona en compilaciones de depuración.

  Added in version 3.4.

- `-X tracemalloc` para iniciar el seguimiento de las asignaciones de memoria de Python mediante el módulo **`tracemalloc`**. De forma predeterminada, solo el marco más reciente se almacena en un seguimiento de un seguimiento. Utilice `-X tracemalloc=NFRAME` para iniciar el seguimiento con un límite de rastreo de marcos *NFRAME*. Consulte el **`tracemalloc.start()`** y **`PYTHONTRACEMALLOC`** para obtener más información.

Added in version 3.4.

- `-X int_max_str_digits` configura la limitación de longitud de conversión de cadena de tipo entero. Véase también **`PYTHONINTMAXSTRDIGITS`**.

  Added in version 3.11.

- `-X importtime` to show how long each import takes. It shows module name, cumulative time (including nested imports) and self time (excluding nested imports). Note that its output may be broken in multi-threaded application. Typical usage is `python -X importtime -c 'import asyncio'`.

  `-X importtime=2` enables additional output that indicates when an imported module has already been loaded. In such cases, the string `cached` will be printed in both time columns.

  See also **`PYTHONPROFILEIMPORTTIME`**.

  Added in version 3.7.

  Distinto en la versión 3.14: Added `-X importtime=2` to also trace imports of loaded modules, and reserved values other than `1` and `2` for future use.

- `-X dev`: enable Python Development Mode, introducing additional runtime checks that are too expensive to be enabled by default. See also **`PYTHONDEVMODE`**.

  Added in version 3.7.

- `-X utf8` habilita el modo Python UTF-8. `-X utf8=0` desactiva explícitamente el modo Python UTF-8 (incluso cuando al contrario se activaría automáticamente). Véase **`PYTHONUTF8`**.

  Added in version 3.7.

- `-X pycache_prefix=PATH` permite escribir archivos `.pyc` en un árbol paralelo enraizado en el directorio dado en lugar de en el árbol de código. Véase también **`PYTHONPYCACHEPREFIX`**.

  Added in version 3.8.

- `-X warn_default_encoding` emite un **`EncodingWarning`** cuando se usa la codificación predeterminada específica de la configuración regional para abrir archivos. Vea también **`PYTHONWARNDEFAULTENCODING`**.

  Added in version 3.10.

- `-X no_debug_ranges` deshabilita la inclusión de tablas que asignan información de ubicación adicional (línea final, desplazamiento de columna inicial y desplazamiento de columna final) a cada instrucción en los objetos de código. Esto es útil cuando se desean objetos de código más pequeños y archivos pyc, además de suprimir los indicadores de ubicación visuales adicionales cuando el intérprete muestra rastreos. Véase también **`PYTHONNODEBUGRANGES`**.

  Added in version 3.11.

- `-X frozen_modules` determines whether or not frozen modules are ignored by the import machinery. A value of `on` means they get imported and `off` means they are ignored. The default is `on` if this is an installed Python (the normal case). If it's under development (running from the source tree) then the default is `off`. Note that the

`importlib_bootstrap` and **`importlib_bootstrap_external`** frozen modules are always used, even if this flag is set to `off`. See also **`PYTHON_FROZEN_MODULES`**.

Added in version 3.11.

- `-X perf` habilita el soporte para el perfilador `perf` de Linux. Cuando se proporciona esta opción, el perfilador `perf` podrá informar llamadas de Python. Esta opción sólo está disponible en algunas plataformas y no hará nada si no es compatible con el sistema actual. El valor predeterminado es «off». Consulte también **`PYTHONPERFSUPPORT`** y Soporte de Python para el perfilador perf de Linux.

  Added in version 3.12.

- `-X perf_jit` enables support for the Linux `perf` profiler with DWARF support. When this option is provided, the `perf` profiler will be able to report Python calls using DWARF information. This option is only available on some platforms and will do nothing if is not supported on the current system. The default value is «off». See also **`PYTHON_PERF_JIT_SUPPORT`** and Soporte de Python para el perfilador perf de Linux.

  Added in version 3.13.

- `-X disable_remote_debug` disables the remote debugging support as described in **PEP 768** [https://peps.python.org/pep-0768/]. This includes both the functionality to schedule code for execution in another process and the functionality to receive code for execution in the current process.

  This option is only available on some platforms and will do nothing if is not supported on the current system. See also **`PYTHON_DISABLE_REMOTE_DEBUG`** and **PEP 768** [https://peps.python.org/pep-0768/].

Added in version 3.14.

- `-X cpu_count=n` overrides **`os.cpu_count()`**, **`os.process_cpu_count()`**, and **`multiprocessing.cpu_count()`**. *n* must be greater than or equal to 1. This option may be useful for users who need to limit CPU resources of a container system. See also **`PYTHON_CPU_COUNT`**. If *n* is `default`, nothing is overridden.

  Added in version 3.13.

- `-X presite=package.module` specifies a module that should be imported before the **`site`** module is executed and before the **`__main__`** module exists. Therefore, the imported module isn't **`__main__`**. This can be used to execute code early during Python initialization. Python needs to be built in debug mode for this option to exist. See also **`PYTHON_PRESITE`**.

  Added in version 3.13.

- `-X gil=0,1` forces the GIL to be disabled or enabled, respectively. Setting to `0` is only available in builds configured with **`--disable-gil`**. See also **`PYTHON_GIL`** and Free-threaded CPython.

  Added in version 3.13.

- `-X thread_inherit_context=0,1` causes **`Thread`** to, by default, use a copy of context of the caller of `Thread.start()` when starting. Otherwise, threads will start with an empty context. If unset, the value of this option defaults to `1` on free-threaded builds and to `0` otherwise. See also **`PYTHON_THREAD_INHERIT_CONTEXT`**.

  Added in version 3.14.

- `-X context_aware_warnings=0,1` causes the **`warnings.catch_warnings`** context manager to use

a **`ContextVar`** to store warnings filter state. If unset, the value of this option defaults to `1` on free-threaded builds and to `0` otherwise. See also **`PYTHON_CONTEXT_AWARE_WARNINGS`**.

Added in version 3.14.

- `-X tlbc=0,1` enables (1, the default) or disables (0) thread-local bytecode in builds configured with **`--disable-gil`**. When disabled, this also disables the specializing interpreter. See also **`PYTHON_TLBC`**.

  Added in version 3.14.

También permite pasar valores arbitrarios y recuperarlos a través del diccionario **`sys._xoptions`**.

Added in version 3.2.

Distinto en la versión 3.9: Removed the `-X showalloccount` option.

Distinto en la versión 3.10: Removed the `-X oldparser` option.

Removed in version 3.14: **`-J`** is no longer reserved for use by Jython [https://www.jython.org/], and now has no special meaning.

1.1.4. Controlling color

The Python interpreter is configured by default to use colors to highlight output in certain situations such as when displaying tracebacks. This behavior can be controlled by setting different environment variables.

Setting the environment variable `TERM` to `dumb` will disable color.

If the **`FORCE_COLOR`** [https://force-color.org/] environment variable is set, then color will be enabled regardless of the value of TERM. This is useful on CI systems which aren’t terminals but can still display ANSI escape sequences.

If the **`NO_COLOR`** [https://no-color.org/] environment variable is set, Python will disable all color in the output. This takes precedence over `FORCE_COLOR`.

All these environment variables are used also by other tools to control color output. To control the color output only in the Python interpreter, the **`PYTHON_COLORS`** environment variable can be used. This variable takes precedence over `NO_COLOR`, which in turn takes precedence over `FORCE_COLOR`.

1.2. Variables de entorno

Estas variables de entorno influyen en el comportamiento de Python, se procesan antes de que los modificadores de línea de comandos distintos de -E o -I. Es habitual que los modificadores de línea de comandos anulen variables de entorno donde hay un conflicto.

PYTHONHOME

Cambie la ubicación de las bibliotecas estándar de Python. De forma predeterminada, las bibliotecas se buscan en `prefix/lib/pythonversion` y `exec_prefix/lib/pythonversion`, donde `prefix` y `exec_prefix` son directorios dependientes de la instalación, ambos predeterminados `/usr/local`.

Cuando **`PYTHONHOME`** se establece en un único directorio, su valor reemplaza tanto al `prefix` como a `exec_prefix`. Para especificar valores diferentes para estos, establezca **`PYTHONHOME`** en `prefix:exec_prefix`.

PYTHONPATH

Aumente la ruta de búsqueda predeterminada para los archivos de módulo. El formato es el mismo que el de shell **`PATH`**: uno o más nombres de ruta de directorio separados por **`os.pathsep`** (por ejemplo, dos puntos en Unix o punto y coma en Windows). Los directorios inexistentes se omiten silenciosamente.

Además de los directorios normales, las entradas individuales **`PYTHONPATH`** pueden referirse a archivos zip que contienen módulos Python puros (ya sea en forma de origen o compilado). Los módulos de extensión no se pueden importar desde zipfiles.

La ruta de búsqueda predeterminada depende de la instalación, pero generalmente comienza con `prefix/lib/pythonversion` (consulte **`PYTHONHOME`** arriba). Es *always* anexado a **`PYTHONPATH`**.

Se insertará un directorio adicional en la ruta de búsqueda delante de **`PYTHONPATH`** como se describió anteriormente en Opciones de interfaz. La ruta de búsqueda se puede manipular desde un programa Python como la variable **`sys.path`**.

PYTHONSAFEPATH

Si se establece en una cadena no vacía, no anteponga una ruta potencialmente insegura a **`sys.path`**: véase la opción **`-u`** para más información.

Added in version 3.11.

PYTHONPLATLIBDIR

Si se establece en una cadena no vacía, anula el valor **`sys.platlibdir`**.

Added in version 3.9.

PYTHONSTARTUP

Si este es el nombre de un archivo legible, los comandos de Python de ese archivo se ejecutan antes de que el primer mensaje se muestre en modo interactivo. El archivo se ejecuta en el mismo espacio de nombres donde se ejecutan comandos interactivos para que los objetos definidos o importados en él se puedan usar sin calificación en la sesión interactiva. También puede cambiar las solicitudes **`sys.ps1`** y **`sys.ps2`** y el enlace **`sys.__interactivehook__`** en este archivo.

Lanza auditing event `cpython.run_startup` con el nombre de archivo como argumento cuando se llama al inicio.

PYTHONOPTIMIZE

Si se establece en una cadena no vacía, equivale a especificar la opción `-O`. Si se establece en un entero, es equivalente a especificar `-O` varias veces.

PYTHONBREAKPOINT

Si se establece, nombra un nombre que se puede llamar mediante la notación de trayecto de puntos. El módulo que contiene el invocable se importará y, a continuación, el invocable se ejecutará por la implementación predeterminada de `sys.breakpointhook()` que a su vez se llama por incorporado `breakpoint()`. Si no se establece o se establece en la cadena vacía, es equivalente al valor «pdb.set_trace». Establecer esto en la cadena «0» hace que la implementación predeterminada de `sys.breakpointhook()` no haga nada más que retornar inmediatamente.

Added in version 3.7.

PYTHONDEBUG

Si se establece en una cadena no vacía, equivale a especificar la opción `-d`. Si se establece en un entero, equivale a especificar `-d` varias veces.

Esta variable de entorno necesita una compilación de depuración de Python, de lo contrario será ignorada.

PYTHONINSPECT

Si se establece en una cadena no vacía, equivale a especificar la opción `-i`.

Esta variable también se puede modificar mediante código Python mediante `os.environ` para forzar el modo de inspección en la terminación del programa.

Lanza un evento auditing event `cpython.run_stdin` sin argumentos.

Distinto en la versión 3.12.5: (also 3.11.10, 3.10.15, 3.9.20, and 3.8.20) Emits audit events.

Distinto en la versión 3.13: Uses PyREPL if possible, in which case **`PYTHONSTARTUP`** is also executed. Emits audit events.

PYTHONUNBUFFERED

Si se establece en una cadena no vacía, equivale a especificar la opción **`-u`**.

PYTHONVERBOSE

Si se establece en una cadena no vacía, equivale a especificar la opción **`-v`**. Si se establece en un entero, equivale a especificar **`-v`** varias veces.

PYTHONCASEOK

Si se establece, Python omite mayúsculas y minúsculas en las instrucciones **`import`**. Esto sólo funciona en Windows y macOS.

PYTHONDONTWRITEBYTECODE

Si se establece en una cadena no vacía, Python no intentará escribir archivos `.pyc` en la importación de módulos de origen. Esto equivale a especificar la opción **`-B`**.

PYTHONPYCACHEPREFIX

Si se establece, Python escribirá archivos `.pyc` en un árbol de directorios reflejado en esta ruta de acceso, en lugar de en directorios `__pycache__` dentro del árbol de origen. Esto equivale a especificar la opción **`-X`** `pycache_prefix=PATH`.

Added in version 3.8.

PYTHONHASHSEED

Si esta variable no se establece o se establece en `random`, se utiliza un valor aleatorio para sembrar los hashes de los objetos str y bytes.

Si **`PYTHONHASHSEED`** se establece en un valor entero, se utiliza como una semilla fija para generar el hash() de los tipos cubiertos por la aleatorización hash.

Su propósito es permitir el hash repetible, como para las autocomprobaciónes para el propio intérprete, o permitir que un grupo de procesos python comparta valores hash.

El entero debe ser un número decimal en el intervalo [0,4294967295]. Especificar el valor 0 deshabilitará la aleatorización de hash.

Added in version 3.2.3.

PYTHONINTMAXSTRDIGITS

Si esta variable se establece en un número entero, se utiliza para configurar la limitación de longitud de conversión de cadena entera.

Added in version 3.11.

PYTHONIOENCODING

Si se establece antes de ejecutar el intérprete, invalida la codificación utilizada para stdin/stdout/stderr, en la sintaxis `encodingname:errorhandler`. Tanto las partes `encodingname` como `:errorhandler` son opcionales y tienen el mismo significado que en **`str.encode()`**.

Para stderr, se omite la parte `:errorhandler`; el manejador siempre será `'backslashreplace'`.

Distinto en la versión 3.4: La parte `encodingname` ahora es opcional.

Distinto en la versión 3.6: En Windows, la codificación especificada por esta variable se omite para los búferes de consola interactivos a menos que también se especifique

`PYTHONLEGACYWINDOWSSTDIO`. Los archivos y canalizaciones redirigidos a través de las corrientes estándar no se ven afectados.

PYTHONNOUSERSITE

Si se establece, Python no agregará **`user site-packages directory`** a **`sys.path`**.

Ver también

PEP 370 [https://peps.python.org/pep-0370/] – Directorio de paquetes de sitio por usuario

PYTHONUSERBASE

Define el **`user base directory`**, que se utiliza para calcular la ruta de acceso de **`user site-packages directory`** y installation paths para `python -m pie install --user`.

Ver también

PEP 370 [https://peps.python.org/pep-0370/] – Directorio de paquetes de sitio por usuario

PYTHONEXECUTABLE

Si se establece esta variable de entorno, `sys.argv[0]` se establecerá en su valor en lugar del valor conseguido a través del tiempo de ejecución de C. Sólo funciona en macOS.

PYTHONWARNINGS

Esto es equivalente a la opción **`-W`**. Si se establece en una cadena separada por comas, es equivalente a especificar **`-W`** varias veces, con filtros más adelante en la lista que tienen prioridad sobre los anteriores de la lista.

La configuración más sencilla aplica una acción determinada incondicionalmente a todas las advertencias emitidas por un proceso (incluso aquellas que de otro modo se ignoran de forma predeterminada):

```
PYTHONWARNINGS=default  # Warn once per call locati
PYTHONWARNINGS=error    # Convert to exceptions
PYTHONWARNINGS=always   # Warn every time
PYTHONWARNINGS=all      # Same as PYTHONWARNINGS=al
PYTHONWARNINGS=module   # Warn once per calling mod
PYTHONWARNINGS=once     # Warn once per Python proc
PYTHONWARNINGS=ignore   # Never warn
```

Consulte El filtro de advertencias y Descripción de los filtros de advertencia para obtener más detalles.

PYTHONFAULTHANDLER

If this environment variable is set to a non-empty string, **`faulthandler.enable()`** is called at startup: install a handler for **`SIGSEGV`**, **`SIGFPE`**, **`SIGABRT`**, **`SIGBUS`** and **`SIGILL`** signals to dump the Python traceback. This is equivalent to **`-X`** `faulthandler` option.

Added in version 3.3.

PYTHONTRACEMALLOC

Si esta variable de entorno se establece en una cadena no vacía, comience a trazar las asignaciones de memoria de Python mediante el módulo **`tracemalloc`**. El valor de la variable es el número máximo de marcos almacenados en un rastreo de un seguimiento. Por ejemplo, `PYTHONTRACEMALLOC=1` almacena sólo el marco más reciente. Consulte la función **`tracemalloc.start()`** para obtener más información. Esto equivale a configurar la opción **`-X`** `tracemalloc`.

Added in version 3.4.

PYTHONPROFILEIMPORTTIME

If this environment variable is set to `1`, Python will show

how long each import takes. If set to `2`, Python will include output for imported modules that have already been loaded. This is equivalent to setting the **`-X`** `importtime` option.

Added in version 3.7.

Distinto en la versión 3.14: Added `PYTHONPROFILEIMPORTTIME=2` to also trace imports of loaded modules.

PYTHONASYNCIODEBUG

Si esta variable de entorno se establece en una cadena no vacía, habilite el modo debug mode del módulo **`asyncio`**.

Added in version 3.4.

PYTHONMALLOC

Establezca los asignadores de memoria de Python y/o instale enlaces de depuración.

Establezca la familia de asignadores de memoria utilizados por Python:

- `default`: utilice default memory allocators.
- `malloc`: utilice la función **`malloc()`** de la biblioteca C para todos los dominios (**`PYMEM_DOMAIN_RAW`**, **`PYMEM_DOMAIN_MEM`**, **`PYMEM_DOMAIN_OBJ`**).
- `pymalloc`: utilice el pymalloc allocator para **`PYMEM_DOMAIN_MEM`** y dominios **`PYMEM_DOMAIN_OBJ`** y utilice la función **`malloc()`** para el dominio **`PYMEM_DOMAIN_RAW`**.
- `mimalloc`: use the mimalloc allocator for **`PYMEM_DOMAIN_MEM`** and **`PYMEM_DOMAIN_OBJ`** domains and use the **`malloc()`** function for the **`PYMEM_DOMAIN_RAW`** domain.

Instalar enlaces de depuración:

- `debug`: instale los enlaces de depuración encima de default memory allocators.

- `malloc_debug`: igual que `malloc` pero también instalar ganchos de depuración.
- `pymalloc_debug`: igual que `pymalloc` pero también instalar enlaces de depuración.
- `mimalloc_debug`: same as `mimalloc` but also install debug hooks.

Nota

In the free-threaded build, the `malloc`, `malloc_debug`, `pymalloc`, and `pymalloc_debug` values are not supported. Only `default`, `debug`, `mimalloc`, and `mimalloc_debug` are accepted.

Added in version 3.6.

Distinto en la versión 3.7: Se ha añadido el asignador `"predeterminado"`.

PYTHONMALLOCSTATS

If set to a non-empty string, Python will print statistics of the pymalloc memory allocator or the mimalloc memory allocator (whichever is in use) every time a new object arena is created, and on shutdown.

This variable is ignored if the **`PYTHONMALLOC`** environment variable is used to force the **`malloc()`** allocator of the C library, or if Python is configured without both `pymalloc` and `mimalloc` support.

Distinto en la versión 3.6: Esta variable ahora también se puede utilizar en Python compilado en modo de versión. Ahora no tiene ningún efecto si se establece en una cadena vacía.

PYTHONLEGACYWINDOWSFSENCODING

Si se establece en una cadena no vacía, el filesystem encoding and error handler predeterminado volverá a sus valores pre-3.6 de *mbcs* y *replace*, respectivamente. De lo contrario, se utilizan los nuevos valores predeterminados "utf-8" y "surrogatepass".

This may also be enabled at runtime with **`sys._enablelegacywindowsfsencoding()`**.

Availability: Windows.

Added in version 3.6: Consulte **PEP 529** [https://peps.python.org/pep-0529/] para obtener más detalles.

PYTHONLEGACYWINDOWSSTDIO

Si se establece en una cadena no vacía, no utiliza el nuevo lector y escritor de consola. Esto significa que los caracteres Unicode se codificarán de acuerdo con la página de códigos de la consola activa, en lugar de usar utf-8.

Esta variable se omite si se redirigen las secuencias estándar (a archivos o canalizaciones) en lugar de hacer referencia a búferes de consola.

Availability: Windows.

Added in version 3.6.

PYTHONCOERCECLOCALE

Si se establece en el valor `0`, hace que la aplicación principal de línea de comandos de Python omita la coerción de las configuraciones regionales C y POSIX basadas en ASCII heredadas a una alternativa basada en UTF-8 más capaz.

Si esta variable es *no* establecida (o se establece en un valor distinto de `0`), tampoco se establece la variable de entorno de invalidación local `LC_ALL`, y la configuración local actual notificada para la categoría `LC_CTYPE` es la configuración local `C` predeterminada, o bien la configuración local basada explícitamente en ASCII `POSIX`, entonces la CLI de Python intentará configurar las siguientes configuraciones locales para la categoría `LC_CTYPE` en el orden indicado antes de cargar el tiempo de ejecución del intérprete:

- `C.UTF-8`
- `C.utf8`

- `UTF-8`

Si la configuración de una de estas categorías de configuración local se realiza correctamente, la variable de entorno `LC_CTYPE` también se establecerá en consecuencia en el entorno de proceso actual antes de que se inicialice el tiempo de ejecución de Python. Esto garantiza que, además de ser visto tanto por el propio intérprete como por otros componentes compatibles con la configuración local que se ejecutan en el mismo proceso (como la biblioteca GNU `readline`), la configuración actualizada también se ve en los subprocesos (independientemente de si esos procesos están ejecutando o no un intérprete de Python), así como en las operaciones que consultan el entorno en lugar de la configuración regional de C actual (como la propia de Python **`locale.getdefaultlocale()`**).

La configuración de una de estas configuraciones regionales (ya sea explícitamente o a través de la coerción de configuración regional implícita anterior) habilita automáticamente el `surrogateescape` error handler para **`sys.stdin`** y **`sys.stdout`** (**`sys.stderr`** continúa utilizando `backslashreplace` como lo hace en cualquier otra configuración local). Este comportamiento de control de secuencias se puede invalidar mediante **`PYTHONIOENCODING`** como de costumbre.

Para fines de depuración, establecer `PYTHONCOERCECLOCALE=warn` hará que Python emita mensajes de advertencia en `stderr` si se activa la coerción de configuración regional, o si una configuración regional que *would* ha activado la coerción sigue activa cuando se inicializa el tiempo de ejecución de Python.

También tenga en cuenta que incluso cuando la coerción de configuración regional está desactivada, o cuando no puede encontrar una configuración regional de destino adecuada, **`PYTHONUTF8`** se activará de forma predeterminada en las configuraciones regionales heredadas basadas en ASCII. Ambas funciones deben estar deshabilitadas para obligar al intérprete a usar `ASCII` en lugar de `UTF-8` para las interfaces del sistema.

Availability: Unix.

Added in version 3.7: Consulte **PEP 538** [https://peps.python.org/pep-0538/] para obtener más detalles.

PYTHONDEVMODE

Si esta variable de entorno se establece en una cadena no vacía, habilite Python Development Mode, introduciendo comprobaciones de tiempo de ejecución adicionales que son demasiado caras para habilitarse de forma predeterminada. Esto equivale a configurar la opción **`-X`** `dev`.

Added in version 3.7.

PYTHONUTF8

Si se establece en `1`, habilita el modo Python UTF-8.

Si se establece en `0`, deshabilita el modo Python UTF-8.

Establecer cualquier otra cadena no vacía produce un error durante la inicialización del intérprete.

Added in version 3.7.

PYTHONWARNDEFAULTENCODING

Si esta variable de entorno se establece como una cadena de caracteres no vacía, emite un **`EncodingWarning`** cuando se utilice la codificación predeterminada específica de la configuración regional.

Ver EncodingWarning opcional para más detalles.

Added in version 3.10.

PYTHONNODEBUGRANGES

Si se establece esta variable, deshabilita la inclusión de tablas que asignan información de ubicación adicional (línea final, desplazamiento de columna inicial y desplazamiento de columna final) a cada instrucción en los objetos de código. Esto es útil cuando se desean objetos de código más pequeños y archivos pyc, además de suprimir los indicadores de ubicación visuales adicionales cuando el intérprete muestra rastreos.

Added in version 3.11.

PYTHONPERFSUPPORT

Si esta variable se establece en un valor distinto de cero, habilita la compatibilidad con el perfilador `perf` de Linux para que pueda detectar las llamadas de Python.

Si se establece en `0`, deshabilite la compatibilidad con el generador de perfiles `perf` de Linux.

Consulte también la opción de línea de comandos **`-X perf`** y Soporte de Python para el perfilador perf de Linux.

Added in version 3.12.

PYTHON_PERF_JIT_SUPPORT

If this variable is set to a nonzero value, it enables support for the Linux `perf` profiler so Python calls can be detected by it using DWARF information.

Si se establece en `0`, deshabilite la compatibilidad con el generador de perfiles `perf` de Linux.

See also the **`-X perf_jit`** command-line option and Soporte de Python para el perfilador perf de Linux.

Added in version 3.13.

PYTHON_DISABLE_REMOTE_DEBUG

If this variable is set to a non-empty string, it disables the remote debugging feature described in **PEP 768** [https://peps.python.org/pep-0768/]. This includes both the functionality to schedule code for execution in another process and the functionality to receive code for execution in the current

process.

See also the `-X disable_remote_debug` command-line option.

Added in version 3.14.

PYTHON_CPU_COUNT

If this variable is set to a positive integer, it overrides the return values of `os.cpu_count()` and `os.process_cpu_count()`.

See also the `-X cpu_count` command-line option.

Added in version 3.13.

PYTHON_FROZEN_MODULES

If this variable is set to `on` or `off`, it determines whether or not frozen modules are ignored by the import machinery. A value of `on` means they get imported and `off` means they are ignored. The default is `on` for non-debug builds (the normal case) and `off` for debug builds. Note that the **`importlib_bootstrap`** and **`importlib_bootstrap_external`** frozen modules are always used, even if this flag is set to `off`.

See also the `-X frozen_modules` command-line option.

Added in version 3.13.

PYTHON_COLORS

If this variable is set to `1`, the interpreter will colorize various kinds of output. Setting it to `0` deactivates this behavior. See also Controlling color.

Added in version 3.13.

PYTHON_BASIC_REPL

If this variable is set to any value, the interpreter will not attempt to load the Python-based REPL that requires

readline, and will instead use the traditional parser-based REPL.

Added in version 3.13.

PYTHON_HISTORY

This environment variable can be used to set the location of a `.python_history` file (by default, it is `.python_history` in the user's home directory).

Added in version 3.13.

PYTHON_GIL

If this variable is set to `1`, the global interpreter lock (GIL) will be forced on. Setting it to `0` forces the GIL off (needs Python configured with the **--disable-gil** build option).

See also the **-X gil** command-line option, which takes precedence over this variable, and Free-threaded CPython.

Added in version 3.13.

PYTHON_THREAD_INHERIT_CONTEXT

If this variable is set to `1` then **Thread** will, by default, use a copy of context of the caller of `Thread.start()` when starting. Otherwise, new threads will start with an empty context. If unset, this variable defaults to `1` on free-threaded builds and to `0` otherwise. See also **-X thread_inherit_context**.

Added in version 3.14.

PYTHON_CONTEXT_AWARE_WARNINGS

If set to `1` then the **warnings.catch_warnings** context manager will use a **ContextVar** to store warnings filter state. If unset, this variable defaults to `1` on free-threaded builds and to `0` otherwise. See **-X context_aware_warnings**.

Added in version 3.14.

PYTHON_JIT

On builds where experimental just-in-time compilation is available, this variable can force the JIT to be disabled (`0`) or enabled (`1`) at interpreter startup.

Added in version 3.13.

PYTHON_TLBC

If set to `1` enables thread-local bytecode. If set to `0` thread-local bytecode and the specializing interpreter are disabled. Only applies to builds configured with **`--disable-gil`**.

See also the **`-X tlbc`** command-line option.

Added in version 3.14.

1.2.1. Variables de modo de depuración

PYTHONDUMPREFS

Si se establece, Python volcará objetos y recuentos de referencias aún vivos después de apagar el intérprete.

Needs Python configured with the **`--with-trace-refs`** build option.

PYTHONDUMPREFSFILE

If set, Python will dump objects and reference counts still alive after shutting down the interpreter into a file under the path given as the value to this environment variable.

Needs Python configured with the **`--with-trace-refs`** build option.

Added in version 3.11.

PYTHON_PRESITE

If this variable is set to a module, that module will be imported early in the interpreter lifecycle, before the **`site`** module is executed, and before the **`__main__`** module is

created. Therefore, the imported module is not treated as **`__main__`**.

This can be used to execute code early during Python initialization.

To import a submodule, use `package.module` as the value, like in an import statement.

See also the **`-X presite`** command-line option, which takes precedence over this variable.

Needs Python configured with the **`--with-pydebug`** build option.

Added in version 3.13.

2. Uso de Python en plataformas Unix

2.1. Obteniendo e instalando la última versión de Python

2.1.1. En Linux

Python comes preinstalled on most Linux distributions, and is available as a package on all others. However there are certain features you might want to use that are not available on your distro's package. You can compile the latest version of Python from source.

In the event that the latest version of Python doesn't come preinstalled and isn't in the repositories as well, you can make packages for your own distro. Have a look at the following links:

Ver también

https://www.debian.org/doc/manuals/maint-guide/first.en.html
para usuarios de Debian

https://en.opensuse.org/Portal:Packaging
para los usuarios de OpenSuse

https://docs.fedoraproject.org/en-US/package-maintainers/Packaging_Tutorial_GNU_Hello/
para los usuarios de Fedora

https://slackbook.org/html/package-management-making-packages.html
para los usuarios de Slackware

2.1.1.1. Installing IDLE

In some cases, IDLE might not be included in your Python installation.

- For Debian and Ubuntu users:

  ```
  sudo apt update
  sudo apt install idle
  ```

- For Fedora, RHEL, and CentOS users:

  ```
  sudo dnf install python3-idle
  ```

- For SUSE and OpenSUSE users:

  ```
  sudo zypper install python3-idle
  ```

- For Alpine Linux users:

  ```
  sudo apk add python3-idle
  ```

2.1.2. En FreeBSD y OpenBSD

- Usuarios FreeBSD, para añadir al paquete use:

  ```
  pkg install python3
  ```

- Usuarios OpenBSD, para añadir al paquete use:

  ```
  pkg_add -r python
  ```

  ```
  pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packa
  ```

  Por ejemplo, los usuarios de i386 obtienen la versión 2.5.1 de Python usando:

  ```
  pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packa
  ```

2.2. Construyendo Python

Ver también

If you want to contribute to CPython, refer to the devguide [https://devguide.python.org/getting-started/setup-building/], which includes build instructions and other tips on setting up environment.

If you want to compile CPython yourself, first thing you should do is get the source [https://www.python.org/downloads/source/]. You can download either the latest release's source or grab a fresh clone [https://devguide.python.org/setup/#get-the-source-code]. You will also need to install the build requirements.

El proceso de construcción consta de los comandos habituales:

```
./configure
make
make install
```

Opciones de configuración y las advertencias para plataformas Unix específicas están ampliamente documentadas en el archivo README.rst [https://github.com/python/cpython/tree/3.14/README.rst] en la raíz del árbol de fuentes de Python.

Advertencia

`make install` puede sobreescribir o enmascarar el binario `python3`. Por lo tanto se recomienda `make altinstall` en lugar de `make install` debido a que sólo instala `exec_prefix/bin/pythonversion`.

2.3. Rutas y archivos relacionados con Python

Estos están sujetos a diferencias según las convenciones de instalación locales; **`prefix`** y **`exec_prefix`** son dependientes de la instalación y deben interpretarse como en el software GNU; pueden ser iguales.

Por ejemplo, en la mayoría de los sistemas Linux, el valor predeterminado para ambos es `/usr`.

Archivo/directorio	**Significado**
`exec_prefix/bin/python3`	Ubicación recomendada del intérprete.
`prefix/lib/pythonversion`, `exec_prefix/lib/pythonversion`	Ubicaciones recomendadas de los directorios que contienen los módulos estándar.
`prefix/include/pythonversion`, `exec_prefix/include/pythonversion`	Ubicaciones recomendadas de los directorios que contienen los archivos de inclusión necesarios para desarrollar extensiones de Python e incrustar el intérprete.

2.4. Miscelánea

Para usar fácilmente los scripts de Python en Unix, debe hacerlos ejecutables, p. ej. con

```
$ chmod +x script
```

y coloque una línea *Shebang* adecuada en la parte superior del script. Una buena opción es usualmente

```
#!/usr/bin/env python3
```

que busca el intérprete de Python en el conjunto **`PATH`**. Sin embargo, algunos Unices puede que no tengan el comando **env**, por lo que es posible que deba codificar `/usr/bin/python3` como la ruta intérprete.

Para usar comandos de shell en sus scripts de Python, mire el módulo **`subprocess`**.

2.5. OpenSSL personalizado

1. Para utilizar la configuración de OpenSSL de su proveedor y el almacén de confianza del sistema, busque el directorio con el archivo `openssl.cnf` o el enlace simbólico en `/etc`. En la mayoría de las distribuciones, el archivo está en `/etc/ssl` o `/etc/pki/tls`. El directorio también debe contener un archivo `cert.pem` y / o un directorio `certs`.

```
$ find /etc/ -name openssl.cnf -printf "%h\n"
/etc/ssl
```

2. Descargue, compile e instale OpenSSL. Asegúrese de utilizar `install_sw` y no `install`. El destino `install_sw` no anula `openssl.cnf`.

   ```
   $ curl -O https://www.openssl.org/source/openssl-VE
   $ tar xzf openssl-VERSION
   $ pushd openssl-VERSION
   $ ./config \
       --prefix=/usr/local/custom-openssl \
       --libdir=lib \
       --openssldir=/etc/ssl
   $ make -j1 depend
   $ make -j8
   $ make install_sw
   $ popd
   ```

3. Construir Python con OpenSSL personalizado (consulte las opciones configure `--with-openssl` y `--with-openssl-rpath`)

   ```
   $ pushd python-3.x.x
   $ ./configure -C \
       --with-openssl=/usr/local/custom-openssl \
       --with-openssl-rpath=auto \
       --prefix=/usr/local/python-3.x.x
   $ make -j8
   $ make altinstall
   ```

Nota

Las versiones de parche de OpenSSL tienen una ABI compatible con versiones anteriores. No es necesario volver a compilar Python para actualizar OpenSSL. Es suficiente reemplazar la instalación personalizada de OpenSSL con una versión más nueva.

[1] [2] [3] [4] [5] [6]

3. Configurar Python

3.1. Requisitos de compilación

To build CPython, you will need:

- Un compilador C11 [https://en.cppreference.com/w/c/11]. Características opcionales de C11 [https://en.wikipedia.org/wiki/C11_(C_standard_revision)#Optional_features] no son necesarias.
- En Windows, se necesita Microsoft Visual Studio 2017 o posterior.
- Support for IEEE 754 [https://en.wikipedia.org/wiki/IEEE_754] floating-point numbers and floating-point Not-a-Number (NaN) [https://en.wikipedia.org/wiki/NaN#Floating_point].
- Soporte para hilos.

Distinto en la versión 3.5: On Windows, Visual Studio 2015 or later is now required.

Distinto en la versión 3.6: Selected C99 features, like `<stdint.h>` and `static inline` functions, are now required.

Distinto en la versión 3.7: Thread support is now required.

Distinto en la versión 3.11: C11 compiler, IEEE 754 and NaN support are now required. On Windows, Visual Studio 2017 or later is required.

Ver también **PEP 7** [https://peps.python.org/pep-0007/] «Style Guide for C Code» y **PEP 11** [https://peps.python.org/pep-0011/] «CPython platform support».

3.1.1. Requirements for optional modules

Some optional modules of the standard library require third-party libraries installed for development (for example, header files must

be available).

Missing requirements are reported in the `configure` output. Modules that are missing due to missing dependencies are listed near the end of the `make` output, sometimes using an internal name, for example, `_ctypes` for **`ctypes`** module.

If you distribute a CPython interpreter without optional modules, it's best practice to advise users, who generally expect that standard library modules are available.

Dependencies to build optional modules are:

Dependency	Minimum version	Python module
libbz2		**`bz2`**
libffi	3.3.0 recommended	**`ctypes`**
liblzma		**`lzma`**
libmpdec	2.5.0	**`decimal`** [1]
libreadline or libedit [2]		**`readline`**
libuuid		_uuid [3]
ncurses [4]		**`curses`**
OpenSSL	3.0.18 recommended (1.1.1 minimum)	**`ssl`**, **`hashlib`** [5]
SQLite	3.15.2	**`sqlite3`**
Tcl/Tk	8.5.12	**`tkinter`**, IDLE, **`turtle`**
zlib	1.2.2.1	**`zlib`**, **`gzip`**, **`ensurepip`**
zstd	1.4.5	**`compression.zstd`**

If *libmpdec* is not available, the **`decimal`** module will use a pure-Python implementation. See **`--with-system-libmpdec`** for details.

See **`--with-readline`** for choosing the backend for the **`readline`** module.

The **`uuid`** module uses `_uuid` to generate «safe» UUIDs. See the module documentation for details.

The **`curses`** module requires the `libncurses` or `libncursesw` library. The **`curses.panel`** module additionally requires the `libpanel` or `libpanelw` library.

If OpenSSL is not available, the `hashlib` module will use bundled implementations of several hash functions. See `--with-builtin-hashlib-hashes` for *forcing* usage of OpenSSL.

Note that the table does not include all optional modules; in particular, platform-specific modules like `winreg` are not listed here.

Ver también

- The devguide [https://devguide.python.org/getting-started/setup-building/#install-dependencies] includes a full list of dependencies required to build all modules and instructions on how to install them on common platforms.
- `--with-system-expat` allows building with an external libexpat [https://libexpat.github.io/] library.
- Options for third-party dependencies

Distinto en la versión 3.1: Tcl/Tk version 8.3.1 is now required for `tkinter`.

Distinto en la versión 3.5: Tcl/Tk version 8.4 is now required for `tkinter`.

Distinto en la versión 3.7: OpenSSL 1.0.2 is now required for `hashlib` and `ssl`.

Distinto en la versión 3.10: OpenSSL 1.1.1 is now required for `hashlib` and `ssl`. SQLite 3.7.15 is now required for `sqlite3`.

Distinto en la versión 3.11: Tcl/Tk version 8.5.12 is now required for `tkinter`.

Distinto en la versión 3.13: SQLite 3.15.2 is now required for `sqlite3`.

3.2. Archivos generados

Para reducir las dependencias de compilación, el código fuente de Python contiene varios archivos generados. Comandos para regenerar todos los archivos generados:

```
make regen-all
make regen-stdlib-module-names
make regen-limited-abi
make regen-configure
```

El archivo `Makefile.pre.in` documenta archivos generados, sus entradas y las herramientas que se usaron para regenerarlos. Busque los objetivos make `regen-*`.

3.2.1. configure script

The `make regen-configure` command regenerates the `aclocal.m4` file and the `configure` script using the `Tools/build/regen-configure.sh` shell script which uses an Ubuntu container to get the same tools versions and have a reproducible output.

The container is optional, the following command can be run locally:

```
autoreconf -ivf -Werror
```

The generated files can change depending on the exact versions of the tools used. The container that CPython uses has Autoconf [https://gnu.org/software/autoconf] 2.72, `aclocal` from Automake [https://www.gnu.org/software/automake] 1.16.5, and pkg-config [https://www.freedesktop.org/wiki/Software/pkg-config/] 1.8.1.

Distinto en la versión 3.13: Autoconf 2.71 and aclocal 1.16.5 and are now used to regenerate `configure`.

Distinto en la versión 3.14: Autoconf 2.72 is now used to regenerate `configure`.

3.3. Configurar opciones

List all `configure` script options using:

```
./configure --help
```

Consultar también `Misc/SpecialBuilds.txt` en la distribución fuente de Python.

3.3.1. Opciones generales

--enable-loadable-sqlite-extensions

Admite extensiones cargables en el módulo de extensión **`_sqlite`** (el valor por defecto es no) del módulo **`sqlite3`**.

Consultar el método **`sqlite3.Connection.enable_load_extension()`** del módulo **`sqlite3`**.

Added in version 3.6.

--disable-ipv6

Deshabilita la compatibilidad con IPv6 (habilitada de forma predeterminada si es compatible), consulte el módulo **`socket`**.

--enable-big-digits=[15|30]

Define el tamaño en bits de los dígitos **`int`** de Python: 15 o 30 bits.

Por defecto, el tamaño del dígito es 30.

Define el `PYLONG_BITS_IN_DIGIT` en `15` o `30`.

Consultar **`sys.int_info.bits_per_digit`**.

--with-suffix=SUFFIX

Establece el sufijo ejecutable de Python en *SUFFIX*.

El sufijo predeterminado es `.exe` en Windows y macOS (ejecutable `python.exe`), `.js` en el nodo Emscripten, `.html` en el navegador Emscripten, `.wasm` en WASI y una cadena vacía en otras plataformas (ejecutable `python`).

Distinto en la versión 3.11: El sufijo predeterminado en la plataforma WASM es uno de `.js`, `.html` o `.wasm`.

--with-tzpath=<list of absolute paths separated by pathsep>

Selecciona la ruta de búsqueda de zona horaria predeterminada para **`zoneinfo.TZPATH`**. Consultar la Configuración en tiempo de compilación del módulo **`zoneinfo`**.

Por defecto: `/usr/share/zoneinfo:/usr/lib/zoneinfo:/usr/share/lib/zoneinfo:/etc/zoneinfo`.

Consultar separador de rutas **`os.pathsep`** .

Added in version 3.9.

--without-decimal-contextvar

Construye el módulo de extensión `_decimal` usando un contexto local de hilos en lugar de un contexto local de corutinas (predeterminado), consultar el módulo **`decimal`**.

Consultar **`decimal.HAVE_CONTEXTVAR`** y el módulo **`contextvars`**.

Added in version 3.9.

--with-dbmliborder=<list of backend names>

Sobrescribe el orden para verificar los de las bases datos para el módulo **`dbm`**

Un valor válido es una cadena de caracteres separada por dos puntos (`:`) con los nombres de los backends:

- `ndbm`;
- `gdbm`;
- `bdb`.

--without-c-locale-coercion

Deshabilita la coerción de configuración regional C a una configuración regional basada en UTF-8 (habilitada de forma predeterminada).

No define la macro `PY_COERCE_C_LOCALE`.

Consultar **`PYTHONCOERCECLOCALE`** y el **PEP 538** [https://peps.python.org/pep-0538/].

--with-platlibdir=DIRNAME

Nombre del directorio de la biblioteca de Python (por defecto es `lib`).

Fedora y SuSE usan `lib64` en plataformas 64-bit.

Consultar **`sys.platlibdir`**.

Added in version 3.9.

--with-wheel-pkg-dir=PATH

Directorio de los paquetes *wheel* usados por el módulo **`ensurepip`** (ninguno por defecto)

Algunas políticas de empaquetado de distribución de Linux recomiendan no empaquetar dependencias. Por ejemplo, Fedora instala paquetes *wheel* en el directorio `/usr/share/python-wheels/` y no instala el paquete **`ensurepip._bundled`**.

Added in version 3.10.

--with-pkg-config=[check|yes|no]

Si configure debe usar **pkg-config** para detectar dependencias de compilación.

- `check` (predeterminado): **pkg-config** es opcional
- `yes`: **pkg-config** es obligatorio
- `no`: configure no usa **pkg-config** incluso cuando está presente

Added in version 3.11.

--enable-pystats

Turn on internal Python performance statistics gathering.

By default, statistics gathering is off. Use `python3 -X pystats` command or set `PYTHONSTATS=1` environment variable to turn on statistics gathering at Python startup.

At Python exit, dump statistics if statistics gathering was on and not cleared.

Efectos:

- Add `-X pystats` command line option.
- Add **`PYTHONSTATS`** environment variable.
- Define the `Py_STATS` macro.
- Add functions to the `sys` module:
  - **`sys._stats_on()`**: Turns on statistics gathering.
  - **`sys._stats_off()`**: Turns off statistics gathering.
  - **`sys._stats_clear()`**: Clears the statistics.
  - **`sys._stats_dump()`**: Dump statistics to file, and clears the statistics.

The statistics will be dumped to a arbitrary (probably unique) file in `/tmp/py_stats/` (Unix) or `C:\temp\py_stats\` (Windows). If that directory does not exist, results will be printed on stderr.

Usa `Tools/scripts/summarize_stats.py` para leer las estadísticas.

Statistics:

- Opcode:
  - Specialization: success, failure, hit, deferred, miss, deopt, failures;
  - Execution count;
  - Pair count.
- Call:
  - Inlined Python calls;

    - PyEval calls;
    - Frames pushed;
    - Frame object created;
    - Eval calls: vector, generator, legacy, function VECTORCALL, build class, slot, function «ex», API, method.
- Object:
    - incref and decref;
    - interpreter incref and decref;
    - allocations: all, 512 bytes, 4 kiB, big;
    - free;
    - to/from free lists;
    - dictionary materialized/dematerialized;
    - type cache;
    - optimization attempts;
    - optimization traces created/executed;
    - uops executed.
- Garbage collector:
    - Garbage collections;
    - Objects visited;
    - Objects collected.

Added in version 3.11.

--disable-gil

Enables support for running Python without the global interpreter lock (GIL): free-threaded build.

Defines the `Py_GIL_DISABLED` macro and adds `"t"` to **`sys.abiflags`**.

See Free-threaded CPython for more detail.

Added in version 3.13.

--enable-experimental-jit=[no|yes|yes-off|interpreter]

Indicate how to integrate the experimental just-in-time compiler.

- `no`: Don't build the JIT.

- `yes`: Enable the JIT. To disable it at runtime, set the environment variable **`PYTHON_JIT=0`**.
- `yes-off`: Build the JIT, but disable it by default. To enable it at runtime, set the environment variable **`PYTHON_JIT=1`**.
- `interpreter`: Enable the «JIT interpreter» (only useful for those debugging the JIT itself). To disable it at runtime, set the environment variable **`PYTHON_JIT=0`**.

`--enable-experimental-jit=no` is the default behavior if the option is not provided, and `--enable-experimental-jit` is shorthand for `--enable-experimental-jit=yes`. See `Tools/jit/README.md` for more information, including how to install the necessary build-time dependencies.

Nota

When building CPython with JIT enabled, ensure that your system has Python 3.11 or later installed.

Added in version 3.13.

PKG_CONFIG

Path to `pkg-config` utility.

PKG_CONFIG_LIBDIR

PKG_CONFIG_PATH

`pkg-config` options.

3.3.2. C compiler options

CC

Comando del compilador C.

CFLAGS

Banderas del compilador de C.

CPP

C preprocessor command.

CPPFLAGS

C preprocessor flags, e.g. `-I`*`include_dir`*.

3.3.3. Opciones del enlazador

LDFLAGS

Linker flags, e.g. `-L`*`library_directory`*.

LIBS

Libraries to pass to the linker, e.g. `-l`*`library`*.

MACHDEP

Name for machine-dependent library files.

3.3.4. Options for third-party dependencies

Added in version 3.11.

BZIP2_CFLAGS

BZIP2_LIBS

C compiler and linker flags to link Python to `libbz2`, used by **`bz2`** module, overriding `pkg-config`.

CURSES_CFLAGS

CURSES_LIBS

C compiler and linker flags for `libncurses` or `libncursesw`, used by **`curses`** module, overriding `pkg-config`.

GDBM_CFLAGS

GDBM_LIBS

C compiler and linker flags for `gdbm`.

LIBEDIT_CFLAGS

LIBEDIT_LIBS

C compiler and linker flags for `libedit`, used by **`readline`** module, overriding `pkg-config`.

LIBFFI_CFLAGS

LIBFFI_LIBS

C compiler and linker flags for `libffi`, used by **`ctypes`** module, overriding `pkg-config`.

LIBMPDEC_CFLAGS

LIBMPDEC_LIBS

C compiler and linker flags for `libmpdec`, used by **`decimal`** module, overriding `pkg-config`.

Nota

These environment variables have no effect unless **`--with-system-libmpdec`** is specified.

LIBLZMA_CFLAGS

LIBLZMA_LIBS

C compiler and linker flags for `liblzma`, used by **`lzma`** module, overriding `pkg-config`.

LIBREADLINE_CFLAGS

LIBREADLINE_LIBS

C compiler and linker flags for `libreadline`, used by

`readline` module, overriding `pkg-config`.

LIBSQLITE3_CFLAGS

LIBSQLITE3_LIBS

C compiler and linker flags for `libsqlite3`, used by **`sqlite3`** module, overriding `pkg-config`.

LIBUUID_CFLAGS

LIBUUID_LIBS

C compiler and linker flags for `libuuid`, used by **`uuid`** module, overriding `pkg-config`.

LIBZSTD_CFLAGS

LIBZSTD_LIBS

C compiler and linker flags for `libzstd`, used by **`compression.zstd`** module, overriding `pkg-config`.

Added in version 3.14.

PANEL_CFLAGS

PANEL_LIBS

C compiler and linker flags for PANEL, overriding `pkg-config`.

C compiler and linker flags for `libpanel` or `libpanelw`, used by **`curses.panel`** module, overriding `pkg-config`.

TCLTK_CFLAGS

TCLTK_LIBS

C compiler and linker flags for TCLTK, overriding `pkg-config`.

ZLIB_CFLAGS

ZLIB_LIBS

C compiler and linker flags for `libzlib`, used by **`gzip`** module, overriding `pkg-config`.

3.3.5. Opciones de WebAssembly

--enable-wasm-dynamic-linking

Active la compatibilidad con enlaces dinámicos para WASM.

La vinculación dinámica habilita `dlopen`. El tamaño del archivo del ejecutable aumenta debido a la eliminación limitada de código muerto y características adicionales.

Added in version 3.11.

--enable-wasm-pthreads

Active la compatibilidad con pthreads para WASM.

Added in version 3.11.

3.3.6. Opciones de instalación

--prefix=PREFIX

Instala archivos independientes de la arquitectura en PREFIX. En Unix, el valor predeterminado es `/usr/local`.

Este valor se puede recuperar en tiempo de ejecución al usar **`sys.prefix`**.

Como ejemplo, se puede utilizar `--prefix="$HOME/.local/"` para instalar Python en su directorio raíz.

--exec-prefix=EPREFIX

Instala archivos independientes de la arquitectura en EPREFIX, el valor predeterminado es **`--prefix`**.

Este valor se puede recuperar en tiempo de ejecución al usar **`sys.exec_prefix`**.

--disable-test-modules

No construya ni instale módulos de prueba, como el paquete **`test`** o el módulo de extensión **`_testcapi`** (construido e instalado por defecto).

Added in version 3.10.

--with-ensurepip=[upgrade|install|no]

Selecciona el comando **`ensurepip`** que se ejecuta en la instalación de Python:

- `upgrade` (por defecto): ejecutar el comando `python -m ensurepip --altinstall --upgrade`.
- `install`: ejecutar el comando `python -m ensurepip --altinstall`;
- `no`: no ejecuta ensurepip;

Added in version 3.6.

3.3.7. Opciones de desempeño

Se recomienda configurar Python usando `--enable-optimizations --with-lto` (PGO + LTO) para obtener el mejor rendimiento. El indicador experimental `--enable-bolt` también se puede usar para mejorar el rendimiento.

--enable-optimizations

Habilite la Optimización Guiada por Perfiles (*PGO* por sus siglas en inglés) usando **`PROFILE_TASK`** (deshabilitado de forma predeterminada).

El compilador de C Clang requiere el programa `llvm-profdata` para PGO. En macOS, GCC también lo requiere: GCC es solo un alias de Clang en macOS.

Desactiva también la interposición semántica en libpython si se usa `--enable-shared` y GCC: agregar `-fno-semantic-interposition` a los flags del compilador y del enlazador.

Nota

During the build, you may encounter compiler warnings about profile data not being available for some source files. These warnings are harmless, as only a subset of the code is exercised during profile data acquisition. To disable these warnings on Clang, manually suppress them by adding `-Wno-profile-instr-unprofiled` to **CFLAGS**.

Added in version 3.6.

Distinto en la versión 3.10: Usar `-fno-semantic-interposition` en GCC.

PROFILE_TASK

Variable de entorno utilizada en el Makefile: argumentos de la línea de comando Python para la tarea de generación de PGO.

Por defecto: `-m test --pgo --timeout=$(TESTTIMEOUT)`.

Added in version 3.8.

Distinto en la versión 3.13: Task failure is no longer ignored silently.

--with-lto=[full|thin|no|yes]

Habilita la Optimización de Tiempo de Enlace (*LTO* por sus siglas en inglés) en cualquier compilación (deshabilitado de forma predeterminada).

El compilador de C Clang requiere `llvm-ar` para LTO (`ar` en macOS), así como un enlazador compatible con LTO (`ld.gold` o `lld`).

Added in version 3.6.

Added in version 3.11: Para usar la función ThinLTO, use `--with-lto=thin` en Clang.

Distinto en la versión 3.12: Utiliza ThinLTO como política de optimización predeterminada en Clang si el compilador acepta el indicador.

--enable-bolt

Habilita el uso del optimizador binario post-enlace BOLT [https://github.com/llvm/llvm-project/tree/main/bolt] (deshabilitado de forma predeterminada).

BOLT es parte del proyecto LLVM pero no siempre se incluye en sus distribuciones binarias. Este indicador necesita que `llvm-bolt` y `merge-fdata` estén disponibles.

BOLT aún es un proyecto bastante nuevo, así que este indicador debería considerarse experimental por ahora. Debido a que esta herramienta opera en código máquina, su éxito depende de una combinación del entorno de compilación + los otros argumentos de configuración de optimización + la arquitectura del CPU, y no todas las combinaciones son compatibles. Se sabe que las versiones de BOLT anteriores a LLVM 16 bloquean BOLT en algunos escenarios. Se recomienda encarecidamente utilizar LLVM 16 o posterior para la optimización de BOLT.

Las variables **configure** **`BOLT_INSTRUMENT_FLAGS`** y **`BOLT_APPLY_FLAGS`** se pueden definir para sobrescribir el conjunto predeterminado de argumentos de **llvm-bolt** para instrumentar y aplicar datos BOLT a binarios, respectivamente.

Added in version 3.12.

BOLT_APPLY_FLAGS

Arguments to `llvm-bolt` when creating a BOLT optimized binary [https://github.com/facebookarchive/BOLT].

Added in version 3.12.

BOLT_INSTRUMENT_FLAGS

Arguments to `llvm-bolt` when instrumenting binaries.

Added in version 3.12.

--with-computed-gotos

Habilita los *gotos* calculados en el ciclo de evaluación (habilitado de forma predeterminada en los compiladores compatibles).

--with-tail-call-interp

Enable interpreters using tail calls in CPython. If enabled, enabling PGO (**`--enable-optimizations`**) is highly recommended. This option specifically requires a C compiler with proper tail call support, and the preserve_none [https://clang.llvm.org/docs/AttributeReference.html#preserve-none] calling convention. For example, Clang 19 and newer supports this feature.

Added in version 3.14.

--without-mimalloc

Disable the fast mimalloc allocator (enabled by default).

This option cannot be used together with **`--disable-gil`** because the free-threaded build requires mimalloc.

Consultar también la variable de entorno **`PYTHONMALLOC`**.

--without-pymalloc

Deshabilita el asignador de memoria especializado de Python pymalloc (habilitado de forma predeterminada).

Consultar también la variable de entorno **`PYTHONMALLOC`**.

--without-doc-strings

Deshabilita las cadenas de caracteres de documentación estáticas para reducir el espacio de memoria (habilitado de forma predeterminada). Las cadenas de caracteres de documentación definidas en Python no se ven afectadas.

No define la macro `WITH_DOC_STRINGS`.

Consultar la macro `PyDoc_STRVAR()`.

--enable-profiling

Habilita el análisis de rendimiento de código (*profiling*) de nivel C con `gprof` (deshabilitado de forma predeterminada).

--with-strict-overflow

Agrega `-fstrict-overflow` a los indicadores del compilador de C (el valor predeterminado que agregamos en su lugar es `-fno-strict-overflow`).

--without-remote-debug

Deactivate remote debugging support described in **PEP 768** [https://peps.python.org/pep-0768/] (enabled by default). When this flag is provided the code that allows the interpreter to schedule the execution of a Python file in a separate process as described in **PEP 768** [https://peps.python.org/pep-0768/] is not compiled. This includes both the functionality to schedule code to be executed and the functionality to receive code to be executed.

Py_REMOTE_DEBUG

This macro is defined by default, unless Python is configured with **`--without-remote-debug`**.

Note that even if the macro is defined, remote debugging may not be available (for example, on an incompatible platform).

Added in version 3.14.

3.3.8. Compilación de depuración de Python

Una compilación de depuración es Python construido con la opción de configuración **`--with-pydebug`**.

Efectos de una compilación de depuración:

- Muestra todas las advertencias de forma predeterminada: la lista de filtros de advertencia predeterminados está vacía en el módulo `warnings`.

- Agrega `d` a `sys.abiflags`.
- Agrega la función `sys.gettotalrefcount()`.
- Agrega la opción de línea de comando `-X showrefcount`.
- Agrega la opción de línea de comando `-d` y la variable de entorno `PYTHONDEBUG` para depurar el analizador.
- Agregue soporte para la variable `__lltrace__`: habilite el seguimiento de bajo nivel en el ciclo de evaluación del código de bytes si la variable está definida.
- Instala ganchos de depuración en los asignadores de memoria para detectar el desbordamiento del búfer y otros errores de memoria.
- Define las macros `Py_DEBUG` y `Py_REF_DEBUG`.
- Agregue verificaciones de tiempo de ejecución: código rodeado por `#ifdef Py_DEBUG` y `#endif`. Habilite las aserciones `assert(...)` y `_PyObject_ASSERT(...)`: no configure la macro `NDEBUG` (vea también la opción de configuración `--with-assertions`). Comprobaciones principales de tiempo de ejecución:
    - Agregue controles de sanidad en los argumentos de la función.
    - Los objetos unicode e int se crean con su memoria completa con un patrón para detectar el uso de objetos no inicializados.
    - Asegúrese de que las funciones que pueden borrar o reemplazar la excepción actual no se invocan con una excepción lanzada.
    - Verifique que las funciones de desasignador no cambien la excepción actual.
    - El recolector de basura (función `gc.collect()`) ejecuta algunas comprobaciones básicas sobre la consistencia de los objetos.
    - La macro `Py_SAFE_DOWNCAST()` comprueba el subdesbordamiento y el desbordamiento de enteros al realizar una conversión descendente de tipos anchos a tipos estrechos.

Consultar también Modo de Desarrollo de Python y la opción de configuración `--with-trace-refs`.

Distinto en la versión 3.8: Release builds are now ABI compatible with debug builds: defining the `Py_DEBUG` macro no longer implies the `Py_TRACE_REFS` macro (see the **`--with-trace-refs`** option). However, debug builds still expose more symbols than release builds and code built against a debug build is not necessarily compatible with a release build.

3.3.9. Opciones de depuración

--with-pydebug

Compila Python en modo de depuración: define la macro `Py_DEBUG` (deshabilitada por defecto).

--with-trace-refs

Habilita las referencias de seguimiento con fines de depuración (deshabilitado de forma predeterminada).

Efectos:

- Define la macro `Py_TRACE_REFS`.
- Add **`sys.getobjects()`** function.
- Agrega la variable de entorno **`PYTHONDUMPREFS`**.

The **`PYTHONDUMPREFS`** environment variable can be used to dump objects and reference counts still alive at Python exit.

Statically allocated objects are not traced.

Added in version 3.8.

Distinto en la versión 3.13: This build is now ABI compatible with release build and debug build.

--with-assertions

Compila con las aserciones de C habilitadas (el valor predeterminado es no): `assert(...);` y `_PyObject_ASSERT(...);`.

Si se establece, la macro `NDEBUG` no está definida en la variable del compilador **`OPT`**.

Consultar también la opción **`--with-pydebug`** (compilación de depuración) que también habilita las aserciones.

Added in version 3.6.

--with-valgrind

Habilite la compatibilidad con Valgrind (el valor predeterminado es no).

--with-dtrace

Habilite la compatibilidad con DTrace (el valor predeterminado es no).

Consultar Instrumentación de CPython con DTrace y SystemTap.

Added in version 3.6.

--with-address-sanitizer

Enable AddressSanitizer memory error detector, `asan` (default is no). To improve ASan detection capabilities you may also want to combine this with **`--without-pymalloc`** to disable the specialized small-object allocator whose allocations are not tracked by ASan.

Added in version 3.6.

--with-memory-sanitizer

Habilita el detector de errores de asignación MemorySanitizer, `msan` (el valor predeterminado es no).

Added in version 3.6.

--with-undefined-behavior-sanitizer

Habilita el detector de comportamiento indefinido UndefinedBehaviorSanitizer, `ubsan` (el valor predeterminado es no).

Added in version 3.6.

--with-thread-sanitizer

Enable ThreadSanitizer data race detector, `tsan` (default is no).

Added in version 3.13.

3.3.10. Opciones del enlazador

--enable-shared

Habilita la compilación de una biblioteca compartida de Python :`libpython` (el valor predeterminado es no).

--without-static-libpython

No compila `libpythonMAJOR.MINOR.a` y no instala `python.o` (compilado y habilitado de forma predeterminada).

Added in version 3.10.

3.3.11. Opciones de bibliotecas

--with-libs='lib1 ...'

Enlace con bibliotecas adicionales (el valor predeterminado es no).

--with-system-expat

Compila el módulo **pyexpat** usando la biblioteca instalada `expat` instalada (por defecto es no).

--with-system-libmpdec

Build the `_decimal` extension module using an installed `mpdecimal` library, see the **decimal** module (default is yes).

Added in version 3.3.

Distinto en la versión 3.13: Default to using the installed `mpdecimal` library.

Distinto en la versión 3.15: A bundled copy of the library will no longer be selected implicitly if an installed `mpdecimal` library is not found. In Python 3.15 only, it can still be selected explicitly using `--with-system-libmpdec=no` or `--without-system-libmpdec`.

Deprecated since version 3.13, will be removed in version 3.16: A copy of the `mpdecimal` library sources will no longer be distributed with Python 3.16.

Ver también

`LIBMPDEC_CFLAGS` and **`LIBMPDEC_LIBS`**.

--with-readline=readline|editline

Designate a backend library for the **`readline`** module.

- readline: Use readline as the backend.
- editline: Use editline as the backend.

Added in version 3.10.

--without-readline

No cree el módulo **`readline`** (es construido por defecto).

No defina la macro `HAVE_LIBREADLINE`.

Added in version 3.10.

--with-libm=STRING

Sobreescribe la biblioteca matemática `libm` a *STRING* (el valor predeterminado es dependiente del sistema).

--with-libc=STRING

Sobreescribe la biblioteca C `libc` a *STRING* (el valor predeterminado es dependiente del sistema).

--with-openssl=DIR

Raíz del directorio OpenSSL.

Added in version 3.7.

--with-openssl-rpath=[no|auto|DIR]

Configura el directorio de la biblioteca en tiempo de ejecución (rpath) para las bibliotecas OpenSSL:

- `no` (por defecto): no establece rpath;
- `auto`: autodetecta rpath desde **`--with-openssl`** y `pkg-config`;
- *DIR*: establece un rpath explícito.

Added in version 3.10.

3.3.12. Opciones de seguridad

--with-hash-algorithm=[fnv|siphash13|siphash24]

Selecciona el algoritmo hash para usar en `Python/pyhash.c`:

- `siphash13` (por defecto);
- `siphash24`;
- `fnv`.

Added in version 3.4.

Added in version 3.11: Se agrega `siphash13` y es el nuevo valor predeterminado.

--with-builtin-hashlib-hashes=md5,sha1,sha256,sha512,sha3,blake2

Módulos hash incorporados:

- `md5`;
- `sha1`;
- `sha256`;
- `sha512`;
- `sha3` (con shake);
- `blake2`.

Added in version 3.9.

--with-ssl-default-suites=[python|openssl|STRING]

Sobreescribe la cadena de conjuntos de cifrado predeterminada de OpenSSL:

- `python` (por defecto): usa la selección principal de Python;
- `openssl`: deja intactos los valores predeterminados de OpenSSL;
- *STRING*: usa una cadena de caracteres personalizada

Consultar el módulo **`ssl`**.

Added in version 3.7.

Distinto en la versión 3.10: Las configuraciones `python` y *STRING* también establecen TLS 1.2 como versión mínima del protocolo.

--disable-safety

Disable compiler options that are recommended by OpenSSF [https://github.com/ossf/wg-best-practices-os-developers/blob/main/docs/Compiler-Hardening-Guides/Compiler-Options-Hardening-Guide-for-C-and-C++.md] for security reasons with no performance overhead. If this option is not enabled, CPython will be built based on safety compiler options with no slow down. When this option is enabled, CPython will not be built with the compiler options listed below.

The following compiler options are disabled with **`--disable-safety`**:

- -fstack-protector-strong [https://github.com/ossf/wg-best-practices-os-developers/blob/main/docs/Compiler-Hardening-Guides/Compiler-Options-Hardening-Guide-for-C-and-C++.md#enable-run-time-checks-for-stack-based-buffer-overflows]: Enable run-time checks for stack-based buffer overflows.
- -Wtrampolines [https://github.com/ossf/wg-best-practices-os-developers/blob/main/docs/Compiler-Hardening-Guides/Compiler-Options-Hardening-Guide-for-C-and-C++.md#enable-warning-

about-trampolines-that-require-executable-stacks]: Enable warnings about trampolines that require executable stacks.

Added in version 3.14.

--enable-slower-safety

Enable compiler options that are recommended by OpenSSF [https://github.com/ossf/wg-best-practices-os-developers/blob/main/docs/Compiler-Hardening-Guides/Compiler-Options-Hardening-Guide-for-C-and-C++.md] for security reasons which require overhead. If this option is not enabled, CPython will not be built based on safety compiler options which performance impact. When this option is enabled, CPython will be built with the compiler options listed below.

The following compiler options are enabled with **`--enable-slower-safety`**:

- -D_FORTIFY_SOURCE=3 [https://github.com/ossf/wg-best-practices-os-developers/blob/main/docs/Compiler-Hardening-Guides/Compiler-Options-Hardening-Guide-for-C-and-C++.md#fortify-sources-for-unsafe-libc-usage-and-buffer-overflows]: Fortify sources with compile- and run-time checks for unsafe libc usage and buffer overflows.

Added in version 3.14.

3.3.13. Opciones macOS

See Mac/README.rst [https://github.com/python/cpython/tree/3.14/Mac/README.rst].

--enable-universalsdk

--enable-universalsdk=SDKDIR

Crea una compilación binaria universal. *SDKDIR* especifica qué macOS SDK debe usarse para realizar la compilación (el valor predeterminado es no).

--enable-framework

--enable-framework=INSTALLDIR

Crear un Python.framework en lugar de una instalación Unix tradicional. Opcionalmente *INSTALLDIR* especifica la ruta de instalación (el valor predeterminado es no).

--with-universal-archs=ARCH

Especifique el tipo de binario universal que se debe crear. Esta opción solo es válida cuando se establece **`--enable-universalsdk`**.

Opciones:

- `universal2` (x86-64 and arm64);
- `32-bit` (PPC and i386);
- `64-bit` (PPC64 and x86-64);
- `3-way` (i386, PPC and x86-64);
- `intel` (i386 and x86-64);
- `intel-32` (i386);
- `intel-64` (x86-64);
- `all` (PPC, i386, PPC64 and x86-64).

Note that values for this configuration item are *not* the same as the identifiers used for universal binary wheels on macOS. See the Python Packaging User Guide for details on the packaging platform compatibility tags used on macOS [https://packaging.python.org/en/latest/specifications/platform-compatibility-tags/#macos]

--with-framework-name=FRAMEWORK

Especifica el nombre del framework de Python en macOS, solo es válido cuando **`--enable-framework`** está configurada (por defecto: `Python`).

--with-app-store-compliance

--with-app-store-compliance=PATCH-FILE

The Python standard library contains strings that are known to trigger automated inspection tool errors when submitted for distribution by the macOS and iOS App Stores. If enabled, this option will apply the list of patches that are known to correct app store compliance. A custom patch file can also be specified. This option is disabled by default.

Added in version 3.13.

3.3.14. iOS Options

See iOS/README.rst [https://github.com/python/cpython/tree/3.14/iOS/README.rst].

--enable-framework=INSTALLDIR

Create a Python.framework. Unlike macOS, the *INSTALLDIR* argument specifying the installation path is mandatory.

--with-framework-name=FRAMEWORK

Specify the name for the framework (default: `Python`).

3.3.15. Opciones de compilación cruzada

La compilación cruzada, también conocida como construcción cruzada, se puede usar para construir Python para otra plataforma o arquitectura de CPU. La compilación cruzada requiere un intérprete de Python para la plataforma de compilación. La versión de Python de compilación debe coincidir con la versión de Python host de compilación cruzada.

--build=BUILD

configure para construir en BUILD, generalmente adivinado por **config.guess**.

--host=HOST

compilación cruzada para crear programas que se ejecuten en HOST (plataforma de destino)

--with-build-python=path/to/python

ruta para construir el binario `python` para compilación cruzada

Added in version 3.11.

CONFIG_SITE=file

Una variable de entorno que apunta a un archivo con anulaciones de configuración.

Example *config.site* file:

```
# config.site-aarch64
ac_cv_buggy_getaddrinfo=no
ac_cv_file__dev_ptmx=yes
ac_cv_file__dev_ptc=no
```

HOSTRUNNER

Program to run CPython for the host platform for cross-compilation.

Added in version 3.11.

Ejemplo de compilación cruzada:

```
CONFIG_SITE=config.site-aarch64 ../configure \
    --build=x86_64-pc-linux-gnu \
    --host=aarch64-unknown-linux-gnu \
    --with-build-python=../x86_64/python
```

3.4. Sistema de compilación Python

3.4.1. Archivos principales del sistema de compilación

- `configure.ac` => `configure`;
- `Makefile.pre.in` => `Makefile` (creado por `configure`);
- `pyconfig.h` (creado por `configure`);
- `Modules/Setup`: Extensiones C creadas por Makefile usando el script de shell `Module/makesetup`;

3.4.2. Pasos principales de compilación

- Los archivos C (`.c`) se crean como archivos objeto (`.o`).
- La biblioteca estática `libpython` (`.a`) se crea a a partir de archivos de objetos.
- `python.o` y la biblioteca estática `libpython` están enlazadas al programa final `python`.
- Las extensiones C son creadas por Makefile (ver `Modules/Setup`).

3.4.3. Objetivos principales de Makefile

3.4.3.1. make

For the most part, when rebuilding after editing some code or refreshing your checkout from upstream, all you need to do is execute `make`, which (per Make's semantics) builds the default target, the first one defined in the Makefile. By tradition (including in the CPython project) this is usually the `all` target. The `configure` script expands an `autoconf` variable, `@DEF_MAKE_ALL_RULE@` to describe precisely which targets `make all` will build. The three choices are:

- `profile-opt` (configured with `--enable-optimizations`)
- `build_wasm` (chosen if the host platform matches `wasm32-wasi*` or `wasm32-emscripten`)
- `build_all` (configured without explicitly using either of the others)

Depending on the most recent source file changes, Make will rebuild any targets (object files and executables) deemed out-of-date, including running `configure` again if necessary. Source/target dependencies are many and maintained manually however, so Make sometimes doesn't have all the information necessary to correctly detect all targets which need to be rebuilt. Depending on which targets aren't rebuilt, you might experience a number of problems. If you have build or test problems which you can't otherwise explain, `make clean && make` should work around most dependency problems, at the expense of longer build times.

3.4.3.2. make platform

Build the `python` program, but don't build the standard library extension modules. This generates a file named `platform` which contains a single line describing the details of the build platform, e.g., `macosx-14.3-arm64-3.12` or `linux-x86_64-3.13`.

3.4.3.3. make profile-opt

Build Python using profile-guided optimization (PGO). You can use the configure **`--enable-optimizations`** option to make this the default target of the `make` command (`make all` or just `make`).

3.4.3.4. make clean

Remove built files.

3.4.3.5. make distclean

In addition to the work done by `make clean`, remove files created by the configure script. `configure` will have to be run before building again. [6]

3.4.3.6. make install

Build the `all` target and install Python.

3.4.3.7. make test

Build the `all` target and run the Python test suite with the `--fast-ci` option without GUI tests. Variables:

- `TESTOPTS`: additional regrtest command-line options.
- `TESTPYTHONOPTS`: additional Python command-line options.
- `TESTTIMEOUT`: timeout in seconds (default: 10 minutes).

3.4.3.8. make ci

This is similar to `make test`, but uses the `-ugui` to also run GUI tests.

Added in version 3.14.

3.4.3.9. make buildbottest

This is similar to `make test`, but uses the `--slow-ci` option and default timeout of 20 minutes, instead of `--fast-ci` option.

3.4.3.10. make regen-all

Regenerate (almost) all generated files. These include (but are not limited to) bytecode cases, and parser generator file. `make regen-stdlib-module-names` and `autoconf` must be run separately for the remaining generated files.

3.4.4. Extensiones C

Some C extensions are built as built-in modules, like the `sys` module. They are built with the `Py_BUILD_CORE_BUILTIN` macro defined. Built-in modules have no `__file__` attribute:

```
>>> import sys
>>> sys
<module 'sys' (built-in)>
>>> sys.__file__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'sys' has no attribute '__file__'
```

Other C extensions are built as dynamic libraries, like the `_asyncio` module. They are built with the `Py_BUILD_CORE_MODULE` macro defined. Example on Linux x86-64:

```
>>> import _asyncio
>>> _asyncio
<module '_asyncio' from '/usr/lib64/python3.9/lib-dynloa
>>> _asyncio.__file__
'/usr/lib64/python3.9/lib-dynload/_asyncio.cpython-39-x8
```

`Modules/Setup` se usa para generar objetivos Makefile para compilar extensiones C. Al principio de los archivos, las extensiones C se crean como módulos incorporados. Las extensiones definidas después del marcador `*shared*` se crean como bibliotecas dinámicas.

Las macros **`PyAPI_FUNC()`**, **`PyAPI_DATA()`** y **`PyMODINIT_FUNC`** de `Include/exports.h` se definen de manera diferente dependiendo si es definida la macro `Py_BUILD_CORE_MODULE`:

- Use `Py_EXPORTED_SYMBOL` si `Py_BUILD_CORE_MODULE` es definido
- Use `Py_IMPORTED_SYMBOL` de lo contrario.

Si la macro `Py_BUILD_CORE_BUILTIN` se usa por error en una extensión de C compilada como una biblioteca compartida, su función `PyInit_xxx()` no se exporta, provocando un **`ImportError`** en la importación.

3.5. Banderas de compilador y vinculación

Opciones establecidas por el script `./configure` y las variables de entorno y utilizadas por `Makefile`.

3.5.1. Banderas del preprocesador

CONFIGURE_CPPFLAGS

Valor de la variable **`CPPFLAGS`** pasado al script `./configure`.

Added in version 3.6.

CPPFLAGS

(Objetivo) Indicadores del preprocesador C/C++, p. ej. `-Iinclude_dir` si tiene encabezados en un directorio no estándar *include_dir*.

Ambos **`CPPFLAGS`** y **`LDFLAGS`** necesitan contener el valor del shell para poder compilar módulos de extensión usando los directorios especificados en las variables de entorno.

BASECPPFLAGS

Added in version 3.4.

PY_CPPFLAGS

Se agregaron indicadores de preprocesador adicionales para construir los archivos de objeto del intérprete.

Por defecto: `$(BASECPPFLAGS) -I. -I$(srcdir)/Include $(CONFIGURE_CPPFLAGS) $(CPPFLAGS)`.

Added in version 3.2.

3.5.2. Banderas del compilador

CC

Comando del compilador C.

Ejemplo: `gcc -pthread`.

CXX

Comando del compilador de C++.

Ejemplo: `g++ -pthread`.

CFLAGS

Banderas del compilador de C.

CFLAGS_NODIST

`CFLAGS_NODIST` se usa para compilar el intérprete y las extensiones stdlib C. Úselo cuando un indicador del compilador *no* deba ser parte de **`CFLAGS`** una vez que Python esté instalado (gh-65320 [https://github.com/python/cpython/issues/65320]).

En particular, **`CFLAGS`** no debe contener:

- el indicador del compilador `-I` (para configurar la ruta de búsqueda de archivos de inclusión). Los indicadores `-I` se procesan de izquierda a derecha, y cualquier indicador en **`CFLAGS`** tendrá prioridad sobre los indicadores `-I` proporcionados por el usuario y el paquete.
- banderas de endurecimiento como `-Werror` porque las distribuciones no pueden controlar si los paquetes instalados por los usuarios cumplen con estándares tan elevados.

Added in version 3.5.

COMPILEALL_OPTS

Las opciones pasadas a la línea de comando **`compileall`** al crear archivos PYC en `make install`. El valor predeterminado: `-j0`.

Added in version 3.12.

EXTRA_CFLAGS

Banderas adicionales del compilador de C.

CONFIGURE_CFLAGS

Valor de la variable **`CFLAGS`** pasada al script `./configure`.

Added in version 3.2.

CONFIGURE_CFLAGS_NODIST

Valor de la variable **`CFLAGS_NODIST`** pasada al script `./configure`.

Added in version 3.5.

BASECFLAGS

Banderas base del compilador.

OPT

Banderas de optimización.

CFLAGS_ALIASING

Banderas de alias estrictos o no estrictos que se utilizan para compilar `Python/dtoa.c`.

Added in version 3.7.

CFLAGS_CEVAL

Flags used to compile `Python/ceval.c`.

Added in version 3.14.5.

CCSHARED

Banderas del compilador que se utilizan para compilar una biblioteca compartida.

Por ejemplo, `-fPIC` se usa en Linux y BSD.

CFLAGSFORSHARED

Se agregaron banderas C adicionales para compilar los archivos de objeto del intérprete.

Por defecto: `$(CCSHARED)` cuando se usa **`--enable-shared`**, o una cadena de caracteres vacía en caso contrario.

PY_CFLAGS

Por defecto: `$(BASECFLAGS) $(OPT) $(CONFIGURE_CFLAGS) $(CFLAGS) $(EXTRA_CFLAGS)`.

PY_CFLAGS_NODIST

Por defecto: `$(CONFIGURE_CFLAGS_NODIST) $(CFLAGS_NODIST) -I$(srcdir)/Include/internal`.

Added in version 3.5.

PY_STDMODULE_CFLAGS

Banderas de C que se utilizan para compilar los archivos de objeto del intérprete.

Por defecto: `$(PY_CFLAGS) $(PY_CFLAGS_NODIST) $(PY_CPPFLAGS) $(CFLAGSFORSHARED)`.

Added in version 3.7.

PY_CORE_CFLAGS

Por defecto: `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE`.

Added in version 3.2.

PY_BUILTIN_MODULE_CFLAGS

Banderas del compilador para construir un módulo de extensión de biblioteca estándar como un módulo incorporado, como el módulo **`posix`**.

Por defecto: `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE_BUILTIN`.

Added in version 3.8.

PURIFY

Comando Purify. Purify es un programa de depuración de memoria.

Por defecto: cadena de caracteres vacía (no utilizado).

3.5.3. Banderas de vinculación

LINKCC

Comando de vinculación usado para compilar programas como `python` y `_testembed`.

Por defecto: `$(PURIFY) $(CC)`.

CONFIGURE_LDFLAGS

Valor de la variable **`LDFLAGS`** pasada al script `./configure`.

Evite asignar **`CFLAGS`**, **`LDFLAGS`**, etc. así los usuarios pueden usarlos en la línea de comando para agregar estos valores sin pisar los valores preestablecidos.

Added in version 3.2.

LDFLAGS_NODIST

`LDFLAGS_NODIST` se usa de la misma manera que **`CFLAGS_NODIST`**. Usar cuando un indicador del enlazador *no* deba ser parte de **`LDFLAGS`** una vez que Python esté instalado (gh-65320 [https://github.com/python/cpython/issues/65320]).

En particular, **`LDFLAGS`** no debe contener:

- el indicador del compilador `-L` (para establecer la ruta de búsqueda de bibliotecas). Los indicadores `-L` se procesan de izquierda a derecha, y cualquier indicador en **`LDFLAGS`** tendrá prioridad sobre los indicadores `-L` proporcionados por el usuario y el paquete.

CONFIGURE_LDFLAGS_NODIST

Valor de la variable **`LDFLAGS_NODIST`** pasado al script `./configure`.

Added in version 3.8.

LDFLAGS

Indicadores de vinculación, p. ej. `-Llib_dir` si tiene bibliotecas en un directorio no estándar *lib_dir*.

Ambos **`CPPFLAGS`** y **`LDFLAGS`** necesitan contener el valor del shell para poder compilar módulos de extensión usando los directorios especificados en las variables de entorno.

LIBS

Banderas de vinculación para pasar bibliotecas al vinculador al enlazar el ejecutable de Python.

Ejemplo: `-lrt`.

LDSHARED

Comando para construir una biblioteca compartida.

Por defecto: `@LDSHARED@ $(PY_LDFLAGS)`.

BLDSHARED

Comando para compilar la biblioteca compartida `libpython`.

Por defecto: `@BLDSHARED@ $(PY_CORE_LDFLAGS)`.

PY_LDFLAGS

Por defecto: `$(CONFIGURE_LDFLAGS) $(LDFLAGS)`.

PY_LDFLAGS_NODIST

Por defecto: `$(CONFIGURE_LDFLAGS_NODIST) $(LDFLAGS_NODIST)`.

Added in version 3.8.

PY_CORE_LDFLAGS

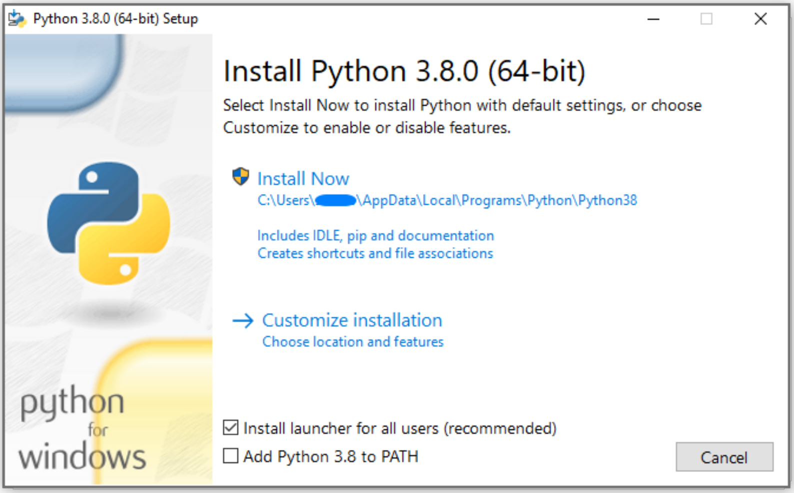
Banderas de vinculación que se utilizan para crear los archivos de objeto del intérprete.

Added in version 3.8.

Footnotes

`git clean -fdx` is an even more extreme way to «clean» your checkout. It removes all files not known to Git. When bug hunting using `git bisect`, this is recommended between probes [https://github.com/python/cpython/issues/114505#issuecomment-1907021718] to guarantee a completely clean build. **Use with care**, as it will delete all files not checked into Git, including your new, uncommitted

work.


Python 3.8.0 (64-bit) Setup

Install Python 3.8.0 (64-bit)

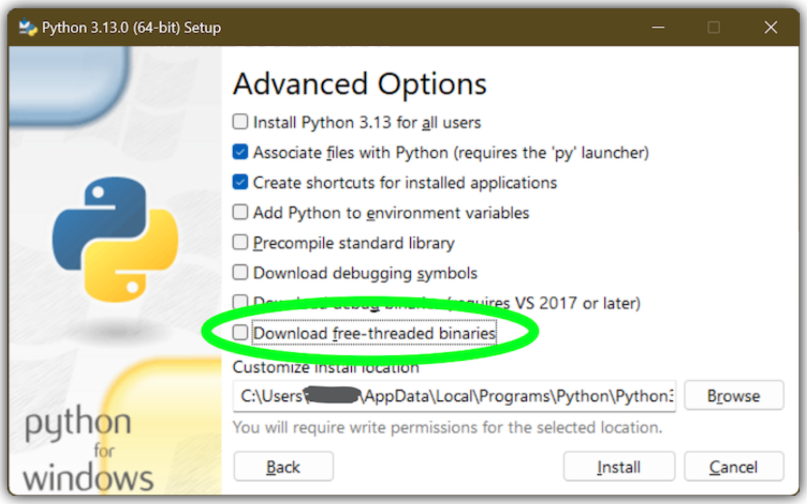
Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\[redacted]\AppData\Local\Programs\Python\Python38

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☐ Add Python 3.8 to PATH


Python 3.13.0 (64-bit) Setup

Advanced Options

☐ Install Python 3.13 for all users
☑ Associate files with Python (requires the 'py' launcher)
☑ Create shortcuts for installed applications
☐ Add Python to environment variables
☐ Precompile standard library
☐ Download debugging symbols
☐ Download debug binaries (requires VS 2017 or later)
☐ Download free-threaded binaries

Customize install location
C:\Users\[redacted]\AppData\Local\Programs\Python\Python

You will require write permissions for the selected location.

4. Uso de Python en Windows

Este documento pretende dar una visión general del comportamiento específico de Windows que se debería conocer al usar Python en Microsoft Windows.

Unlike most Unix systems and services, Windows does not include a

system supported installation of Python. Instead, Python can be obtained from a number of distributors, including directly from the CPython team. Each Python distribution will have its own benefits and drawbacks, however, consistency with other tools you are using is generally a worthwhile benefit. Before committing to the process described here, we recommend investigating your existing tools to see if they can provide Python directly.

To obtain Python from the CPython team, use the Python Install Manager. This is a standalone tool that makes Python available as global commands on your Windows machine, integrates with the system, and supports updates over time. You can download the Python Install Manager from python.org/downloads [https://www.python.org/downloads/] or through the Microsoft Store app [https://apps.microsoft.com/detail/9NQ7512CXL7T].

Once you have installed the Python Install Manager, the global `python` command can be used from any terminal to launch your current latest version of Python. This version may change over time as you add or remove different versions, and the `py list` command will show which is current.

In general, we recommend that you create a virtual environment for each project and run `<env>\Scripts\Activate` in your terminal to use it. This provides isolation between projects, consistency over time, and ensures that additional commands added by packages are also available in your session. Create a virtual environment using `python -m venv <env path>`.

If the `python` or `py` commands do not seem to be working, please see the Troubleshooting section below. There are sometimes additional manual steps required to configure your PC.

Apart from using the Python install manager, Python can also be obtained as NuGet packages. See El paquete de nuget.org below for more information on these packages.

The embeddable distros are minimal packages of Python suitable for embedding into larger applications. They can be installed using the Python install manager. See El paquete incrustable below for more information on these packages.

4.1. Python install manager

4.1.1. Installation

The Python install manager can be installed from the Microsoft Store app [https://apps.microsoft.com/detail/9NQ7512CXL7T] or downloaded and installed from python.org/downloads [https://www.python.org/downloads/]. The two versions are identical.

To install through the Store, simply click «Install». After it has completed, open a terminal and type `python` to get started.

To install the file downloaded from python.org, either double-click and select «Install», or run `Add-AppxPackage <path to MSIX>` in Windows Powershell.

After installation, the `python`, `py`, and `pymanager` commands should be available. If you have existing installations of Python, or you have modified your **PATH** variable, you may need to remove them or undo the modifications. See Troubleshooting for more help with fixing non-working commands.

When you first install a runtime, you will likely be prompted to add a directory to your **PATH**. This is optional, if you prefer to use the `py` command, but is offered for those who prefer the full range of aliases (such as `python3.14.exe`) to be available. The directory will be `%LocalAppData%\Python\bin` by default, but may be customized by an administrator. Click Start and search for «Edit environment variables for your account» for the system settings page to add the path.

Each Python runtime you install will have its own directory for scripts. These also need to be added to **PATH** if you want to use them.

The Python install manager will be automatically updated to new releases. This does not affect any installs of Python runtimes. Uninstalling the Python install manager does not uninstall any Python runtimes.

If you are not able to install an MSIX in your context, for example, you are using automated deployment software that does not support it, or are targeting Windows Server 2019, please see Advanced installation below for more information.

4.1.2. Basic use

The recommended command for launching Python is `python`, which will either launch the version requested by the script being launched, an active virtual environment, or the default installed version, which will be the latest stable release unless configured otherwise. If no version is specifically requested and no runtimes are installed at all, the current latest release will be installed automatically.

For all scenarios involving multiple runtime versions, the recommended command is `py`. This may be used anywhere in place of `python` or the older `py.exe` launcher. By default, `py` matches the behaviour of `python`, but also allows command line options to select a specific version as well as subcommands to manage installations. These are detailed below.

Because the `py` command may already be taken by the previous version, there is also an unambiguous `pymanager` command. Scripted installs that are intending to use Python install manager should consider using `pymanager`, due to the lower chance of encountering a conflict with existing installs. The only difference between the two commands is when running without any arguments: `py` will launch your default interpreter, while `pymanager` will display help (`pymanager exec ...` provides equivalent behaviour to `py ...`).

Each of these commands also has a windowed version that avoids creating a console window. These are `pyw`, `pythonw` and `pymanagerw`. A `python3` command is also included that mimics the `python` command. It is intended to catch accidental uses of the typical POSIX command on Windows, but is not meant to be widely used or recommended.

To launch your default runtime, run `python` or `py` with the arguments you want to be passed to the runtime (such as script files or the module to launch):

```
$> py
...
$> python my-script.py
...
$> py -m this
...
```

The default runtime can be overridden with the **`PYTHON_MANAGER_DEFAULT`** environment variable, or a configuration file. See Configuration for information about configuration settings.

To launch a specific runtime, the `py` command accepts a `-V:<TAG>` option. This option must be specified before any others. The tag is part or all of the identifier for the runtime; for those from the CPython team, it looks like the version, potentially with the platform. For compatibility, the `V:` may be omitted in cases where the tag refers to an official release and starts with `3`.

```
$> py -V:3.14 ...
$> py -V:3-arm64 ...
```

Runtimes from other distributors may require the *company* to be included as well. This should be separated from the tag by a slash, and may be a prefix. Specifying the company is optional when it is `PythonCore`, and specifying the tag is optional (but not the slash) when you want the latest release from a specific company.

```
$> py -V:Distributor\1.0 ...
$> py -V:distrib/ ...
```

If no version is specified, but a script file is passed, the script will be inspected for a *shebang line*. This is a special format for the first line in a file that allows overriding the command. See Shebang lines for more information. When there is no shebang line, or it cannot be resolved, the script will be launched with the default runtime.

If you are running in an active virtual environment, have not requested a particular version, and there is no shebang line, the default runtime will be that virtual environment. In this scenario, the `python` command was likely already overridden and none of these checks occurred. However, this behaviour ensures that the `py` command can be used interchangeably.

When no runtimes are installed, any launch command will try to install the requested version and launch it. However, after any version is installed, only the `py exec ...` and `pymanager exec ...` commands will install if the requested version is absent. Other forms of commands will display an error and direct you to use `py install` first.

4.1.3. Command help

The `py help` command will display the full list of supported commands, along with their options. Any command may be passed the `-?` option to display its help, or its name passed to `py help`.

```
$> py help
$> py help install
$> py install /?
```

All commands support some common options, which will be shown by `py help`. These options must be specified after any subcommand. Specifying `-v` or `--verbose` will increase the amount of output shown, and `-vv` will increase it further for debugging purposes. Passing `-q` or `--quiet` will reduce output, and `-qq` will reduce it further.

The `--config=<PATH>` option allows specifying a configuration file to override multiple settings at once. See Configuration below for more information about these files.

4.1.4. Listing runtimes

```
$> py list [-f=|--format=<FMT>] [-1|--one] [--online|-s=
```

The list of installed runtimes can be seen using `py list`. A filter may be added in the form of one or more tags (with or without company specifier), and each may include a `<`, `<=`, `>=` or `>` prefix to restrict to a range.

A range of formats are supported, and can be passed as the `--format=<FMT>` or `-f <FMT>` option. Formats include `table` (a user friendly table view), `csv` (comma-separated table), `json` (a single JSON blob), `jsonl` (one JSON blob per result), `exe` (just the executable path), `prefix` (just the prefix path).

The `--one` or `-1` option only displays a single result. If the default runtime is included, it will be the one. Otherwise, the «best» result is shown («best» is deliberately vaguely defined, but will usually be the most recent version). The result shown by `py list --one <TAG>` will match the runtime that would be launched by `py -V:<TAG>`.

The `--only-managed` option excludes results that were not installed by the Python install manager. This is useful when determining which runtimes may be updated or uninstalled through the `py` command.

The `--online` option is short for passing `--source=<URL>` with the default source. Passing either of these options will search the online index for runtimes that can be installed. The result shown by `py list --online --one <TAG>` will match the runtime that would be installed by `py install <TAG>`.

```
$> py list --online 3.14
```

For compatibility with the old launcher, the `--list`, `--list-paths`, `-0` and `-0p` commands (e.g. `py -0p`) are retained. They do not allow additional options, and will produce legacy formatted output.

4.1.5. Installing runtimes

```
$> py install [-s=|--source=<URL>] [-f|--force] [-u|--up
```

New runtime versions may be added using `py install`. One or more tags may be specified, and the special tag `default` may be used to select the default. Ranges are not supported for installation.

The `--source=<URL>` option allows overriding the online index that is used to obtain runtimes. This may be used with an offline index, as shown in Offline installs.

Passing `--force` will ignore any cached files and remove any existing install to replace it with the specified one.

Passing `--update` will replace existing installs if the new version is newer. Otherwise, they will be left. If no tags are provided with `--update`, all installs managed by the Python install manager will be updated if newer versions are available. Updates will remove any modifications made to the install, including globally installed packages, but virtual environments will continue to work.

Passing `--dry-run` will generate output and logs, but will not modify any installs.

Passing `--refresh` will update all registrations for installed runtimes. This will recreate Start menu shortcuts, registry keys, and global aliases (such as `python3.14.exe` or for any installed scripts). These are automatically refreshed on installation of any runtime, but may need to be manually refreshed after installing packages.

In addition to the above options, the `--target` option will extract the runtime to the specified directory instead of doing a normal install. This is useful for embedding runtimes into larger applications. Unlike a normal install, `py` will not be aware of the extracted runtime, and no Start menu or other shortcuts will be created. To launch the runtime, directly execute the main executable (typically `python.exe`) in the target directory.

```
$> py install ... [-t=|--target=<PATH>] <TAG>
```

The `py exec` command will install the requested runtime if it is not already present. This is controlled by the `automatic_install` configuration (**`PYTHON_MANAGER_AUTOMATIC_INSTALL`**), and is enabled by default. If no runtimes are available at all, all launch commands will do an automatic install if the configuration setting allows. This is to ensure a good experience for new users, but should not generally be relied on rather than using the `py exec` command or explicit install commands.

4.1.6. Offline installs

To perform offline installs of Python, you will need to first create an offline index on a machine that has network access.

```
$> py install --download=<PATH> ... <TAG>...
```

The `--download=<PATH>` option will download the packages for the listed tags and create a directory containing them and an `index.json` file suitable for later installation. This entire directory can be moved to the offline machine and used to install one or more of the bundled runtimes:

```
$> py install --source="<PATH>\index.json" <TAG>...
```

The Python install manager can be installed by downloading its installer and moving it to another machine before installing.

Alternatively, the ZIP files in an offline index directory can simply be transferred to another machine and extracted. This will not register the install in any way, and so it must be launched by directly referencing the executables in the extracted directory, but it is sometimes a preferable approach in cases where installing the Python install manager is not possible or convenient.

In this way, Python runtimes can be installed and managed on a machine without access to the internet.

4.1.7. Uninstalling runtimes

```
$> py uninstall [-y|--yes] <TAG>...
```

Runtimes may be removed using the `py uninstall` command. One or more tags must be specified. Ranges are not supported here.

The `--yes` option bypasses the confirmation prompt before uninstalling.

Instead of passing tags individually, the `--purge` option may be specified. This will remove all runtimes managed by the Python install manager, including cleaning up the Start menu, registry, and any download caches. Runtimes that were not installed by the Python install manager will not be impacted, and neither will manually created configuration files.

```
$> py uninstall [-y|--yes] --purge
```

The Python install manager can be uninstalled through the Windows «Installed apps» settings page. This does not remove any runtimes, and they will still be usable, though the global `python` and `py` commands will be removed. Reinstalling the Python install manager will allow you to manage these runtimes again. To completely clean up all Python runtimes, run with `--purge` before uninstalling the Python install manager.

4.1.8. Configuration

Python install manager is configured with a hierarchy of configuration files, environment variables, command-line options, and registry settings. In general, configuration files have the ability to configure everything, including the location of other configuration files, while registry settings are administrator-only and will override configuration files. Command-line options override all other settings, but not every option is available.

This section will describe the defaults, but be aware that modified or overridden installs may resolve settings differently.

A global configuration file may be configured by an administrator, and would be read first. The user configuration file is stored at `%AppData%\Python\pymanager.json` (note that this location is under `Roaming`, not `Local`) and is read next, overwriting any settings from earlier files. An additional configuration file may be specified as the `PYTHON_MANAGER_CONFIG` environment variable or the `--config` command line option (but not both). These locations may be modified by administrative customization options listed later.

The following settings are those that are considered likely to be modified in normal use. Later sections list those that are intended for administrative customization.

Standard configuration options

Config Key	Environment Variable	Description
`default_tag`	`PYTHON_MANAGER_DEFAULT`	The preferred default version to launch or install. By default, this is interpreted as the most recent non-prerelease version from the CPython team.
`default_platform`	`PYTHON_MANAGER_DEFAULT_PLATFORM`	The preferred default platform to launch or install. This is treated as a suffix to the specified tag, such that `py -V:3.14` would prefer an install for `3.14-64` if it exists (and `default_platform` is `-64`), but will use `3.14` if no tagged install exists.
`logs_dir`	`PYTHON_MANAGER_LOGS`	The location where log files are written. By default, `%TEMP%`.
`automatic_install`	`PYTHON_MANAGER_AUTOMATIC_INSTALL`	True to allow automatic installs when using `py exec` to launch (or `py` when no runtimes are installed yet). Other commands will not automatically install, regardless of this setting. By default, true.
`include_unmanaged`	`PYTHON_MANAGER_INCLUDE_UNMANAGED`	True to allow listing and launching runtimes that were not installed by the Python install manager, or false to exclude them. By default, true.
`shebang_can_run_anything`	`PYTHON_MANAGER_SHEBANG_CAN_RUN_ANYTHING`	True to allow shebangs in `.py` files to launch applications other than Python runtimes, or false to prevent it. By default, true.
`log_level`	`PYMANAGER_VERBOSE`, `PYMANAGER_DEBUG`	Set the default level of output (0-50). By default, 20. Lower values produce more output. The environment variables are boolean, and may produce additional output during startup that is later suppressed by other configuration.
`confirm`	`PYTHON_MANAGER_CONFIRM`	True to confirm certain actions before taking them (such as uninstall), or false to skip the confirmation. By default, true.
`install.source`	`PYTHON_MANAGER_SOURCE_URL`	Override the index feed to obtain new installs from.
`install.enable_entrypoints`	(none)	True to generate global commands for installed packages (such as `pip.exe`). These are defined by the packages themselves. If set to false, only the Python interpreter has global commands created. By default, true. You should run `py install --refresh` after changing this setting.
`list.format`	`PYTHON_MANAGER_LIST_FORMAT`	Specify the default format used by the `py list` command. By default, `table`.
`install_dir`	(none)	Specify the root directory that runtimes will be installed into. If

you change this setting, previously installed runtimes will not be usable unless you move them to the new location.

`global_dir`

Specify the directory where global commands (such as `python3.14.exe` and `pip.exe`) are stored. This directory should be added to your **`PATH`** to make the commands available from your terminal.

`download_dir`

Specify the directory where downloaded files are stored. This directory is a temporary cache, and can be cleaned up from time to time.

Dotted names should be nested inside JSON objects, for example, `list.format` would be specified as `{"list": {"format": "table"}}`.

4.1.9. Shebang lines

If the first line of a script file starts with `#!`, it is known as a «shebang» line. Linux and other Unix like operating systems have native support for such lines and they are commonly used on such systems to indicate how a script should be executed. The `python` and `py` commands allow the same facilities to be used with Python scripts on Windows.

To allow shebang lines in Python scripts to be portable between Unix and Windows, a number of “virtual” commands are supported to specify which interpreter to use. The supported virtual commands are:

- `/usr/bin/env <ALIAS>`
- `/usr/bin/env -S <ALIAS>`
- `/usr/bin/<ALIAS>`
- `/usr/local/bin/<ALIAS>`
- `<ALIAS>`

Por ejemplo, si la primera línea del script comienza con

```
#! /usr/bin/python
```

The default Python or an active virtual environment will be located and used. As many Python scripts written to work on Unix will already have this line, you should find these scripts can be used by the launcher without modification. If you are writing a new script on Windows which you hope will be useful on Unix, you should use one of the shebang lines starting with `/usr`.

Any of the above virtual commands can have `<ALIAS>` replaced by an alias from an installed runtime. That is, any command generated in the global aliases directory (which you may have added to your **PATH** environment variable) can be used in a shebang, even if it is not on your **PATH**. This allows the use of shebangs like `/usr/bin/python3.12` to select a particular runtime.

If no runtimes are installed, or if automatic installation is enabled, the requested runtime will be installed if necessary. See Configuration for information about configuration settings.

The `/usr/bin/env` form of shebang line will also search the **PATH** environment variable for unrecognized commands. This corresponds to the behaviour of the Unix `env` program, which performs the same search, but prefers launching known Python commands. A warning may be displayed when searching for arbitrary executables, and this search may be disabled by the `shebang_can_run_anything` configuration option.

Shebang lines that do not match any of patterns are treated as *Windows* executable paths that are absolute or relative to the directory containing the script file. This is a convenience for Windows-only scripts, such as those generated by an installer, since the behavior is not compatible with Unix-style shells. These paths may be quoted, and may include multiple arguments, after which the path to the script and any additional arguments will be appended. This functionality may be disabled by the `shebang_can_run_anything` configuration option.

Nota

The behaviour of shebangs in the Python install manager is subtly different from the previous `py.exe` launcher, and the old configuration options no longer apply. If you are specifically reliant on the old behaviour or configuration, we recommend installing the legacy launcher [https://www.python.org/ftp/

python/3.14.0/win32/launcher.msi]. The legacy launcher's `py` command will override PyManager's one by default, and you will need to use `pymanager` commands for installing and uninstalling.

4.1.10. Advanced installation

For situations where an MSIX cannot be installed, such as some older administrative distribution platforms, there is an MSI available from the python.org downloads page. This MSI has no user interface, and can only perform per-machine installs to its default location in Program Files. It will attempt to modify the system **PATH** environment variable to include this install location, but be sure to validate this on your configuration.

Nota

Windows Server 2019 is the only version of Windows that CPython supports that does not support MSIX. For Windows Server 2019, you should use the MSI.

Be aware that the MSI package does not bundle any runtimes, and so is not suitable for installs into offline environments without also creating an offline install index. See Offline installs and Administrative configuration for information on handling these scenarios.

Runtimes installed by the MSI are shared with those installed by the MSIX, and are all per-user only. The Python install manager does not support installing runtimes per-machine. To emulate a per-machine install, you can use `py install --target=<shared location>` as administrator and add your own system-wide modifications to **PATH**, the registry, or the Start menu.

When the MSIX is installed, but commands are not available in the **PATH** environment variable, they can be found under `%LocalAppData%\Microsoft\WindowsApps\PythonSoftwareFoundation.PythonManager_3847v3x7pw1km` or `%LocalAppData%\Microsoft\WindowsApps`

`\PythonSoftwareFoundation.PythonManager_qbz5n2kfra8p0`, depending on whether it was installed from python.org or through the Windows Store. Attempting to run the executable directly from Program Files is not recommended.

To programmatically install the Python install manager, it is easiest to use WinGet, which is included with all supported versions of Windows:

```
$> winget install 9NQ7512CXL7T -e --accept-package-agree

# Optionally run the configuration checker and accept al
$> py install --configure -y
```

To download the Python install manager and install on another machine, the following WinGet command will download the required files from the Store to your Downloads directory (add `-d <location>` to customize the output location). This also generates a YAML file that appears to be unnecessary, as the downloaded MSIX can be installed by launching or using the commands below.

```
$> winget download 9NQ7512CXL7T -e --skip-license --acce
```

To programmatically install or uninstall an MSIX using only PowerShell, the Add-AppxPackage [https://learn.microsoft.com/powershell/module/appx/add-appxpackage] and Remove-AppxPackage [https://learn.microsoft.com/powershell/module/appx/remove-appxpackage] PowerShell cmdlets are recommended:

```
$> Add-AppxPackage C:\Downloads\python-manager-25.0.msix
...
$> Get-AppxPackage PythonSoftwareFoundation.PythonManage
```

The latest release can be downloaded and installed by Windows by passing the AppInstaller file to the Add-AppxPackage command. This installs using the MSIX on python.org, and is only recommended for cases where installing via the Store (interactively or using WinGet) is not possible.

```
$> Add-AppxPackage -AppInstallerFile https://www.python.
```

Other tools and APIs may also be used to provision an MSIX package for all users on a machine, but Python does not consider this a supported scenario. We suggest looking into the PowerShell Add-AppxProvisionedPackage [https://learn.microsoft.com/powershell/module/dism/add-appxprovisionedpackage] cmdlet, the native Windows PackageManager [https://learn.microsoft.com/uwp/api/windows.management.deployment.packagemanager] class, or the documentation and support for your deployment tool.

Regardless of the install method, users will still need to install their own copies of Python itself, as there is no way to trigger those installs without being a logged in user. When using the MSIX, the latest version of Python will be available for all users to install without network access.

Note that the MSIX downloadable from the Store and from the Python website are subtly different and cannot be installed at the same time. Wherever possible, we suggest using the above WinGet commands to download the package from the Store to reduce the risk of setting up conflicting installs. There are no licensing restrictions on the Python install manager that would prevent using the Store package in this way.

4.1.11. Administrative configuration

There are a number of options that may be useful for administrators to override configuration of the Python install manager. These can be used to provide local caching, disable certain shortcut types, override bundled content. All of the above configuration options may be set, as well as those below.

Configuration options may be overridden in the registry by setting values under `HKEY_LOCAL_MACHINE\Software\Policies\Python\PyManager`, where the value name matches the configuration key and the value type is `REG_SZ`. Note that this key can itself be customized, but only by modifying the core config file distributed with the Python install manager. We recommend, however, that registry values are used only to set `base_config` to a JSON file containing the full set of overrides. Registry key overrides will replace any other configured setting, while `base_config` allows users to further modify settings they may need.

Note that most settings with environment variables support those variables because their default setting specifies the variable. If you override them, the environment variable will no longer work, unless you override it with another one. For example, the default value of `confirm` is literally `%PYTHON_MANAGER_CONFIRM%`, which will resolve the variable at load time. If you override the value to `yes`, then the environment variable will no longer be used. If you override the value to `%CONFIRM%`, then that environment variable will be used instead.

Configuration settings that are paths are interpreted as relative to the directory containing the configuration file that specified them.

Administrative configuration options

Config Key	Description
`base_config`	The highest priority configuration file to read. Note that only the built-in configuration file and the registry can modify this setting.
`user_config`	The second configuration file to read.
`additional_config`	The third configuration file to read.
`registry_override_key`	Registry location to check for overrides. Note that only the built-in configuration file can modify this setting.
`bundled_dir`	Read-only directory containing locally cached files.
`install.fallback_source`	Path or URL to an index to consult when the main index cannot be accessed.
`install.enable_shortcut_kinds`	Comma-separated list of shortcut kinds to allow (e.g. `"pep514,start"`). Enabled shortcuts may still be disabled by `disable_shortcut_kinds`.
`install.disable_shortcut_kinds`	Comma-separated list of shortcut kinds to exclude (e.g. `"pep514,start"`). Disabled shortcuts are not reactivated by `enable_shortcut_kinds`.
`install.hard_link_entrypoints`	True to use hard links for global shortcuts to save disk space. If false, each shortcut executable is copied instead. After changing this setting, you must run `py install --refresh --force` to update existing commands. By default, true. Disabling this

	may be necessary for troubleshooting or systems that have issues with file links.
`pep514_root`	Registry location to read and write PEP 514 entries into. By default, `HKEY_CURRENT_USER\Software\Python`.
`start_folder`	Start menu folder to write shortcuts into. By default, `Python`. This path is relative to the user's Programs folder.
`virtual_env`	Path to the active virtual environment. By default, this is `%VIRTUAL_ENV%`, but may be set empty to disable venv detection.
`shebang_can_run_anything_silently`	True to suppress visible warnings when a shebang launches an application other than a Python runtime.
`source_settings`	A mapping from source URL to settings specific to that index. When multiple configuration files include this section, URL settings are added or overwritten, but individual settings are not merged. These settings are currently only for index signatures.

4.1.12. Installing free-threaded binaries

Added in version 3.13.

Pre-built distributions of the free-threaded build are available by installing tags with the `t` suffix.

```
$> py install 3.14t
$> py install 3.14t-arm64
$> py install 3.14t-32
```

This will install and register as normal. If you have no other runtimes installed, then `python` will launch this one. Otherwise, you will need to use `py -V:3.14t ...` or, if you have added the global aliases directory to your **`PATH`** environment variable, the `python3.14t.exe` commands.

4.1.13. Index signatures

Added in version 26.2.

Index files may be signed to detect tampering. A signature is a catalog file at the same URL as the index with `.cat` added to the filename. The catalog file should contain the hash of its matching index file, and should be signed with a valid Authenticode signature. This allows standard tooling (on Windows) to generate a signature, and any certificate may be used as long as the client operating system already trusts its certification authority (root CA).

Index signatures are only downloaded and checked when the local configuration's `source_settings` section includes the index URL and `requires_signature` is true, or the index JSON contains `requires_signature` set to true. When the setting exists in local configuration, even when false, settings in the index are ignored.

As well as requiring a valid signature, the `required_root_subject` and `required_publisher_subject` settings can further restrict acceptable signatures based on the certificate Subject fields. Any attribute specified in the configuration must match the attribute in the certificate (additional attributes in the certificate are ignored). Typical attributes are `CN=` for the common name, `O=` for the organizational unit, and `C=` for the publisher's country.

Finally, the `required_publisher_eku` setting allows requiring that a specific Enhanced Key Usage (EKU) has been assigned to the publisher certificate. For example, the EKU `1.3.6.1.5.5.7.3.3` indicates that the certificate was intended for code signing (as opposed to server or client authentication). In combination with a specific root CA, this provides another mechanism to verify a legitimate signature.

This is an example `source_settings` section from a configuration file. In this case, the publisher of the feed is uniquely identified by the combination of the Microsoft Identity Verification root and the EKU assigned by that root. The signature for this case would be found at `https://www.python.org/ftp/python/index-windows.json.cat`.

```
{
  "source_settings": {
    "https://www.python.org/ftp/python/index-windows.jso
      "requires_signature": true,
      "required_root_subject": "CN=Microsoft Identity Ve
      "required_publisher_subject": "CN=Python Software
```

```
        "required_publisher_eku": "1.3.6.1.4.1.311.97.608
      }
    }
}
```

The same settings could be specified in the `index.json` file instead. In this case, the root and EKU are omitted, meaning that the signature must be valid and have a specific common name in the publisher's certificate, but no other checks are used.

```
{
  "requires_signature": true,
  "required_publisher_subject": "CN=Python Software Four
  "versions": [
    // ...
  ]
}
```

When settings from inside a feed are used, the user is notified and the settings are shown in the log file or verbose output. It is recommended to copy these settings into a local configuration file for feeds that will be used frequently, so that unauthorised modifications to the feed cannot disable verification.

It is not possible to override the location of the signature file in the feed or through a configuration file. Administrators can provide their own `source_settings` in a mandatory configuration file (see Administrative configuration).

If signature validation fails, you will be notified and prompted to continue. When interactive confirmation is not allowed (for example, because `--yes` was specified), it will always abort. To use a feed with invalid configuration in this scenario, you must provide a configuration file that disables signature checking for that feed.

```
"source_settings": {
  "https://www.example.com/feed-with-invalid-signature.
    "requires_signature": false
  }
```

}

4.1.14. Troubleshooting

If your Python install manager does not seem to be working correctly, please work through these tests and fixes to see if it helps. If not, please report an issue at our bug tracker [https://github.com/python/pymanager/issues], including any relevant log files (written to your `%TEMP%` directory by default).

Troubleshooting

Symptom	**Things to try**
`python` gives me a «command not found» error or opens the Store app when I type it in my terminal.	Did you install the Python install manager?
	Click Start, open «Manage app execution aliases», and check that the aliases for «Python (default)» are enabled. If they already are, try disabling and re-enabling to refresh the command. The «Python (default windowed)» and «Python install manager» commands may also need refreshing.
	Check that the `py` and `pymanager` commands work.
	Ensure your **`PATH`** variable contains the entry for `%UserProfile%\AppData\Local\Microsoft\WindowsApps`. The operating system includes this entry once by default, after other user paths. If removed, shortcuts will not be found.
`py` gives me a «command not found» error when I type it in my terminal.	Did you install the Python install manager?
	Click Start, open «Manage app execution aliases», and check that the aliases for «Python (default)» are enabled. If they already are, try disabling and re-enabling to refresh the command. The «Python (default windowed)» and «Python install manager» commands may also need refreshing.
	Ensure your **`PATH`** variable contains the entry for `%UserProfile%\AppData\Local\Microsoft\WindowsApps`. The operating system includes this entry once by default, after other user paths. If removed, shortcuts will not be found.
`py` gives me a «can't open file» error when I type commands in	This usually means you have the legacy launcher installed and it

my terminal.	has priority over the Python install manager. To remove, click Start, open «Installed apps», search for «Python launcher» and uninstall it.
`python` doesn't launch the same runtime as `py`	Click Start, open «Installed apps», look for any existing Python runtimes, and either remove them or Modify and disable the **`PATH`** options. Click Start, open «Manage app execution aliases», and check that your `python.exe` alias is set to «Python (default)»
`python` and `py` don't launch the runtime I expect	Check your **`PYTHON_MANAGER_DEFAULT`** environment variable or `default_tag` configuration. The `py list` command will show your default based on these settings. Installs that are managed by the Python install manager will be chosen ahead of unmanaged installs. Use `py install` to install the runtime you expect, or configure your default tag. Prerelease and experimental installs that are not managed by the Python install manager may be chosen ahead of stable releases. Configure your default tag or uninstall the prerelease runtime and reinstall it using `py install`.
`pythonw` or `pyw` don't launch the same runtime as `python` or `py`	Click Start, open «Manage app execution aliases», and check that your `pythonw.exe` and `pyw.exe` aliases are consistent with your others.
`pip` gives me a «command not found» error when I type it in my terminal.	Have you activated a virtual environment? Run the `.venv\Scripts\activate` script in your terminal to activate. The package may be available but missing the generated executable. We recommend using the `python -m pip` command instead. Running `py install --refresh` and ensuring that the global shortcuts directory is on **`PATH`** (it will be shown in the command output if it is not) should make commands such as `pip` (and other installed packages) available.
I installed a package with `pip` but its command is not found.	Have you activated a virtual environment? Run the `.venv\Scripts\activate` script in your terminal to activate. New packages do not automatically have global shortcuts created by the Python install manager. Similarly, uninstalled packages do not have their shortcuts removed. Run `py install --refresh` to update the global shortcuts for newly installed packages.
Typing `script-name.py` in the terminal opens in a new window.	This is a known limitation of the operating system. Either specify `py` before the script name, create a batch file containing `@py "%~dpn0.py" %*` with the same name as the script, or install the legacy launcher and select it as the association for scripts.

Drag-dropping files onto a script doesn't work	This is a known limitation of the operating system. It is supported with the legacy launcher, or with the Python install manager when installed from the MSI.
I have installed the Python install manager multiple times.	It is possible to install from the Store or WinGet, from the MSIX on the Python website, and from the MSI, all at once. They are all compatible and will share configuration and runtimes. See the earlier Advanced installation section for ways to uninstall the install manager other than the typical Installed Apps (Add and Remove Programs) settings page.
My old `py.ini` file no longer works.	The new Python install manager no longer supports this configuration file or its settings, and so it will be ignored. See Configuration for information about configuration settings.

4.2. El paquete incrustable

Added in version 3.5.

La distribución incrustable consiste en un archivo ZIP que contiene un mínimo entorno de Python. Está destinado a ser usado como parte de otra aplicación, en lugar de ser accedido directamente por los usuarios finales.

To install an embedded distribution, we recommend using `py install` with the `--target` option:

```
$> py install 3.14-embed --target=<directory>
```

When extracted, the embedded distribution is (almost) fully isolated from the user's system, including environment variables, system registry settings, and installed packages. The standard library is included as pre-compiled and optimized `.pyc` files in a ZIP, and `python3.dll`, `python313.dll`, `python.exe` and `pythonw.exe` are all provided. Tcl/tk (including all dependents, such as Idle), pip and the Python documentation are not included.

A default `._pth` file is included, which further restricts the default search paths (as described below in Encontrar módulos). This file is intended for embedders to modify as necessary.

Los paquetes de terceros deben ser instalados por el instalador de la aplicación junto a la distribución incrustada. El uso de pip para administrar dependencias como en una instalación de Python regular no es soportado por esta distribución, aunque con cierto cuidado es posible incluir y usar pip para automatizar las actualizaciones. En general, los paquetes de terceros deben ser tratados como parte de la aplicación («vendoring») para que el desarrollador pueda asegurar la compatibilidad con las nuevas versiones antes de proporcionar actualizaciones a los usuarios.

Los dos casos de uso recomendados para esta distribución se describen a continuación.

4.2.1. Python application

Una aplicación escrita en Python no necesariamente requiere que los usuarios sean conscientes de ese hecho. La distribución incrustada puede ser usada en este caso para incluir una versión privada de Python en un paquete de instalación. Dependiendo de lo transparente que deba ser (o por el contrario, de lo profesional que deba parecer), hay dos opciones.

El uso de un ejecutable especializado como lanzador requiere algo de código, pero proporciona la experiencia más transparente para los usuarios. Con un lanzador personalizado, no hay indicadores obvios de que el programa se ejecuta en Python: los íconos pueden ser personalizados, se puede especificar información de la compañía y de la versión, y las asociaciones de archivos se comportan correctamente. En la mayoría de los casos, un lanzador personalizado debería simplemente poder invocar `Py_Main` utilizando una línea de comandos codificada.

El enfoque más simple es proporcionar un archivo por lotes o un acceso directo generado que directamente invoque `python.exe` o `pythonw.exe` con los argumentos de línea de comandos requeridos. En este caso, la aplicación aparecerá como Python y no con su nombre real, y los usuarios podrían tener problemas para distinguirla de otros procesos Python en ejecución o asociaciones de archivos.

Con este último enfoque, los paquetes deben instalarse como directorios junto al ejecutable de Python para asegurar su disponibilidad en la ruta. Con el lanzador especializado, los paquetes pueden encontrarse en otras ubicaciones ya que hay oportunidad de especificar la ruta de búsqueda antes de iniciar la aplicación.

4.2.2. Incrustar Python

Las aplicaciones escritas en código nativo frecuentemente requieren algún tipo de lenguaje de scripting, y la distribución de Python incrustada puede ser utilizada con ese propósito. En general, la mayoría de la aplicación utiliza código nativo, y alguna parte invocará `python.exe` o usará `python3.dll` directamente. Para cualquiera de estos casos, la extracción de la distribución incrustable a un subdirectorio de la instalación de la aplicación es suficiente para proporcionar un intérprete de Python invocable.

Al igual que con el uso de la aplicación, los paquetes pueden ser instalados en cualquier ubicación, ya que existe la posibilidad de especificar rutas de búsqueda antes de inicializar el intérprete. Más allá de esto, no existen diferencias fundamentales entre el uso de la distribución incrustada y una instalación normal.

4.3. El paquete de nuget.org

Added in version 3.5.2.

El paquete de nuget.org es un entorno Python de tamaño reducido destinado a usarse en sistemas de integración continua y compilación que no posean una instalación de Python a nivel de sistema. Si bien nuget es «el administrador de paquetes para .NET», también funciona perfectamente para paquetes que contienen herramientas de tiempo de compilación.

Visite nuget.org [https://www.nuget.org/] para obtener la información más actualizada sobre cómo usar nuget. Lo que sigue es un resumen que es suficiente para desarrolladores Python.

La herramienta de línea de comandos `nuget.exe` puede ser descargada directamente desde `https://aka.ms/nugetclidl`, por ejemplo usando curl o PowerShell. Con esta herramienta, la última versión de Python para máquinas de 64 o 32 bit se instala con:

```
nuget.exe install python -ExcludeVersion -OutputDirector
nuget.exe install pythonx86 -ExcludeVersion -OutputDirec
```

Para seleccionar una versión en particular, agregue un `-Version 3.x.y`. El directorio de salida se puede cambiar de `.` y el paquete se instalará en un subdirectorio. De forma predeterminada, el subdirectorio tiene el mismo nombre que el paquete y, sin la opción `-ExcludeVersion`, este nombre incluirá la versión específica instalada. Dentro del subdirectorio hay un directorio `tools` que contiene la instalación de Python:

```
# Without -ExcludeVersion
> .\python.3.5.2\tools\python.exe -V
Python 3.5.2

# With -ExcludeVersion
> .\python\tools\python.exe -V
Python 3.5.2
```

En general, los paquetes nuget no son actualizables, y versiones más nuevas deben ser instaladas en paralelo y referenciadas usando la ruta completa. Otra opción es borrar el directorio del paquete de forma manual e instalarlo de nuevo. Muchos sistemas de CI harán esto automáticamente si no mantienen archivos entre compilaciones.

Junto al directorio `tools` está el directorio `build\native`. Este contiene un archivo de propiedades MSBuild `Python.props` que puede ser usado en un proyecto C++ para referenciar la instalación de Python. Al incluir las configuraciones, automáticamente se usarán los encabezados y se importarán las bibliotecas en la compilación.

The package information pages on nuget.org are www.nuget.org/packages/python [https://www.nuget.org/packages/python] for the 64-bit version, www.nuget.org/packages/pythonx86 [https://www.nuget.org/packages/pythonx86] for the 32-bit version, and www.nuget.org/packages/pythonarm64 [https://www.nuget.org/packages/pythonarm64] for the ARM64 version

4.3.1. Free-threaded packages

Added in version 3.13.

Packages containing free-threaded binaries are named python-freethreaded [https://www.nuget.org/packages/python-freethreaded] for the 64-bit version, pythonx86-freethreaded [https://www.nuget.org/packages/pythonx86-freethreaded] for the 32-bit version, and pythonarm64-freethreaded [https://www.nuget.org/packages/pythonarm64-freethreaded] for the ARM64 version. These packages contain both the `python3.13t.exe` and `python.exe` entry points, both of which run free threaded.

4.4. Distribuciones alternativas

Además de la distribución estándar de CPython, hay paquetes modificados que incluyen funcionalidad adicional. La siguiente es una lista de versiones populares y sus características clave:

ActivePython [https://www.activestate.com/products/python/]

> Instalador compatible con múltiples plataformas, documentación, PyWin32

Anaconda [https://www.anaconda.com/download/]

> Módulos científicos populares (como numpy, scipy y pandas) y el gestor de paquetes `conda`.

Enthought Deployment Manager [https://assets.enthought.com/downloads/edm/]

> «El administrador de paquetes y entorno de Python de próxima generación».
>
> Anteriormente, Enthought proporcionaba Canopy, pero llegó al final de su vida en2016 [https://support.enthought.com/hc/en-us/articles/360038600051-Canopy-GUI-end-of-life-transition-to-the-Enthought-Deployment-Manager-EDM-and-Visual-Studio-Code].

WinPython [https://winpython.github.io/]

Distribución específica para Windows con paquetes científicos precompilados y herramientas para construir paquetes.

Tenga en cuenta que estos paquetes pueden no incluir la última versión de Python u otras bibliotecas, y no son mantenidos ni respaldados por el equipo central de Python.

4.5. Supported Windows versions

As specified in **PEP 11** [https://peps.python.org/pep-0011/], a Python release only supports a Windows platform while Microsoft considers the platform under extended support. This means that Python 3.14 supports Windows 10 and newer. If you require Windows 7 support, please install Python 3.8. If you require Windows 8.1 support, please install Python 3.12.

4.6. Removing the MAX_PATH limitation

Windows históricamente ha limitado la longitud de las rutas a 260 caracteres. Esto significaba que rutas de mayor longitud no resolverían y se producirían errores.

In the latest versions of Windows, this limitation can be expanded to over 32,000 characters. Your administrator will need to activate the «Enable Win32 long paths» group policy, or set `LongPathsEnabled` to `1` in the registry key `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem`.

Esto permite que la función `open()`, el módulo `os` y la mayoría de las demás funciones de ruta acepten y retornen rutas de más de 260 caracteres.

After changing the above option and rebooting, no further configuration is required.

4.7. Modo UTF-8

Added in version 3.7.

Windows todavía usa codificaciones heredadas para la codificación del sistema (la página de códigos ANSI). Python lo usa para la codificación predeterminada de archivos de texto (por ejemplo, **`locale.getencoding()`**).

Esto puede causar problemas porque UTF-8 es ampliamente utilizado en internet y en la mayoría de los sistemas Unix, incluido WSL (subsistema de Windows para Linux).

Se puede utilizar el modo UTF-8 para cambiar la codificación predeterminada a UTF-8. El modo UTF-8 se puede activar mediante la opción de línea de comandos `-X utf-8`, o con la variable de entorno `PYTHONUTF8=1`. Consulte **`PYTHONUTF8`** para activar el modo UTF-8, y Python install manager para saber cómo modificar las variables de entorno.

Cuando el modo UTF-8 de Python es activado, usted puede seguir usando la codificación del sistema (La página de código ANSI) a través del códec «mbcs».

Tenga en cuenta que agregar `PYTHONUTF8=1` a las variables de entorno predeterminadas afectará a todas las aplicaciones de Python 3.7+ en el sistema. Si utiliza alguna aplicación de Python 3.7+ que depende de la codificación heredada del sistema, se recomienda que se configure la variable de entorno solo temporalmente o se use la opción de línea de comandos `-X utf8`.

Nota

Aún con el modo UTF-8 desactivado, Python utiliza UTF-8 de forma predeterminada en Windows para:

- E/S de consola, incluida la E/S estándar (consultar **PEP 528** [https://peps.python.org/pep-0528/] para más detalles).
- La codificación del sistema (vea **PEP 529** [https://peps.python.org/pep-0529/] para más detalles).

4.8. Encontrar módulos

Estas notas complementan la descripción en La inicialización de la ruta de búsqueda de módulo de sys.path con notas detalladas de Windows.

Cuando no se encuentre ningún archivo `._pth`, así es como **`sys.path`** es completado en Windows:

- Se agrega una entrada vacía al comienzo, que corresponde al directorio actual.
- Si existe la variable de entorno **`PYTHONPATH`**, de acuerdo a lo descrito en Variables de entorno, sus entradas se agregan a continuación. Tenga en cuenta que en Windows, las rutas en esta variable deben estar separadas por punto y coma (;), para distinguirlas de los dos puntos utilizados en los identificadores de disco (`C:\`, etc.).
- Se pueden agregar al registro «rutas de aplicación» adicionales como subclaves de `\SOFTWARE\Python\PythonCore{version}\PythonPath` bajo los subárboles `HKEY_CURRENT_USER` y `HKEY_LOCAL_MACHINE`. Las subclaves que contienen un valor por defecto compuesto por cadenas de ruta separadas por punto y coma causan que cada una de esas rutas sea agregada a **`sys.path`**. (Tenga en cuenta que todos los instaladores conocidos solo utilizan HKLM, por lo que HKCU comúnmente se encuentra vacío.)
- Si se configura la variable de entorno **`PYTHONHOME`**, es asumida como el «Python Home» (el directorio de origen de Python). De lo contrario, la ruta del ejecutable principal de Python es utilizada para ubicar un «archivo de referencia» (ya sea `Lib\os.py` o `pythonXY.zip`) para deducir el «Python Home». Si el directorio de origen de Python es encontrado, los subdirectorios relevantes que se agregan a **`sys.path`** (`Lib`, `plat-win`, etc.) se basan en ese directorio. Por el contrario, la ruta principal de Python se construye a partir del PythonPath guardado en el registro.
- Si el Python Home no puede ser ubicado, **`PYTHONPATH`** no está especificado en el entorno y no se encuentra ninguna entrada en el registro, se usa una ruta predeterminada con entradas relativas (por ej. `.\Lib;.\plat-win`, etc.).

Si se encuentra el archivo `pyvenv.cfg` junto al ejecutable principal o en el directorio un nivel arriba del ejecutable, se aplica la siguiente variación:

- Si `home` es una ruta absoluta y **`PYTHONHOME`** no está configurada, se usa esta ruta en lugar de la ruta al ejecutable principal para deducir la ubicación del directorio de origen.

El resultado final de todo esto es:

- Cuando se ejecuta `python.exe`, o cualquier otro .exe en el directorio principal de Python (tanto la versión instalada como directamente desde el directorio PCbuild), se deduce la ruta principal, y se ignoran las rutas principales en el registro. Siempre se leen otras «rutas de aplicación» del registro.
- Cuando se aloja Python en otro .exe (distinto directorio, incrustado mediante COM, etc.), el «Python Home» no se deduce, y se utiliza la ruta principal del registro. Siempre se leen otras «rutas de aplicación» del registro.
- Si Python no puede encontrar su directorio de origen y no hay valores en el registro (un .exe congelado, una muy rara configuración de instalación) se obtiene una ruta relativa predeterminada.

Para aquellos que quieran incluir Python en su aplicación o distribución, los siguientes consejos evitarán conflictos con otras instalaciones:

- Incluya un archivo `._pth` junto al ejecutable, que contenga los directorios a incluir. Esto hará que se ignoren las rutas enumeradas en el registro y en las variables de entorno, y que también se ignore **`site`** a menos que se especifique `import site`.
- If you are loading `python3.dll` or `python37.dll` in your own executable, explicitly set **`PyConfig.module_search_paths`** before **`Py_InitializeFromConfig()`**.
- Limpie y/o sobrescriba **`PYTHONPATH`** y configure **`PYTHONHOME`** antes de iniciar `python.exe` desde su aplicación.
- Si no se pueden utilizar las sugerencias previas (por ejemplo, en una distribución que permite a los usuarios ejecutar

`python.exe` directamente), hay que asegurarse de que el archivo de referencia (`Lib\os.py`) exista en el directorio de instalación. (Tener en cuenta que este no será detectado dentro de un archivo ZIP, pero si se detectará un ZIP correctamente nombrado.)

Esto asegura que los archivos de una instalación del sistema no tendrán precedencia por sobre la copia de la biblioteca estándar incluida en su aplicación. De otra manera, los usuarios podrían experimentar problemas al utilizar su aplicación. Tenga en cuenta que la primera sugerencia es la mejor, ya que las otras aún pueden ser afectadas por rutas no estándar en el registro y en el site-packages del usuario.

Distinto en la versión 3.6: Add `._pth` file support and removes `applocal` option from `pyvenv.cfg`.

Distinto en la versión 3.6: Add `python`*`XX`*`.zip` as a potential landmark when directly adjacent to the executable.

Obsoleto desde la versión 3.6: Los módulos especificados en el registro bajo `Modules` (no `PythonPath`) pueden ser importados por **`importlib.machinery.WindowsRegistryFinder`**. Este buscador está habilitado en Windows en la versión 3.6.0 y anteriores, pero es posible que deba agregarse explícitamente a **`sys.meta_path`** en el futuro.

4.9. Módulos adicionales

Aunque Python pretende ser portátil entre todas las plataformas, hay características que son exclusivas de Windows. Existen un par de módulos, de la biblioteca estándar y externos, y fragmentos de código para utilizar estas funciones.

Los módulos estándar específicos para Windows se encuentran documentados en Servicios Específicos para MS Windows.

4.9.1. PyWin32

The PyWin32 [https://pypi.org/project/PyWin32/] module by Mark

Hammond is a collection of modules for advanced Windows-specific support. This includes utilities for:

- Component Object Model [https://learn.microsoft.com/windows/win32/com/component-object-model--com--portal] (COM)
- Invocación de la API Win32
- Registro
- Registro de eventos
- Microsoft Foundation Classes [https://learn.microsoft.com/cpp/mfc/mfc-desktop-applications] (MFC) user interfaces

PythonWin [https://web.archive.org/web/20060524042422/https://www.python.org/windows/pythonwin/] es una aplicación MFC de muestra distribuida con PyWin32. Es un IDE incrustable con depurador incorporado.

Ver también

Win32 How Do I...? [https://timgolden.me.uk/python/win32_how_do_i.html]
por Tim Golden

Python and COM [https://www.boddie.org.uk/python/COM.html]
por David y Paul Boddie

4.9.2. cx_Freeze

cx_Freeze [https://cx-freeze.readthedocs.io/en/latest/] envuelve scripts de Python en programas ejecutables de Windows (archivos `*.exe`).Cuando hayas hecho esto, puedes distribuir tu aplicación sin requerir que tus usuarios instalen Python.

4.10. Compilar Python en Windows

Si desea compilar CPython usted mismo, lo primero que debe hacer es obtener el source [https://www.python.org/downloads/source/]. Puede descargar la fuente de la última versión o simplemente obtener un checkout [https://devguide.python.org/setup/#get-the-source-code] nuevo.

El árbol fuente contiene una solución de compilación y archivos de proyecto para Microsoft Visual Studio, que es el compilador que se usa para compilar las versiones oficiales de Python. Estos archivos están en el directorio `PCbuild`.

Consulte `PCbuild/readme.txt` para obtener información general acerca del proceso de compilación.

Para módulos de extensión, consulte Creación de extensiones C y C + + en Windows.

4.11. The full installer (deprecated)

Obsoleto desde la versión 3.14: This installer is deprecated since 3.14 and will not be produced for Python 3.16 or later. See Python install manager for the modern installer.

4.11.1. Pasos para la instalación

Cuatro instaladores de Python 3.14 están disponibles para descargar - dos por cada una de las versiones de 32-bit y 64-bit del intérprete. El *instalador web* es una pequeña descarga inicial que automáticamente descargará los componentes requeridos cuando sea necesario. El *instalador fuera de línea* incluye los componentes necesarios para una instalación por defecto y solo requiere de una conexión a internet para características opcionales. Consultar Installing without downloading para conocer otras formas de evitar descargas durante la instalación.

Luego de iniciar el instalador, se puede seleccionar una de estas dos opciones:

Si se selecciona «Install Now»:

- *No* necesitarás ser administrador (a menos que se requiera una actualización de sistema para C Runtime Library o se necesite instalar el Python install manager para todos los usuarios)
- Python será instalado en su directorio de usuario
- El Python install manager será instalado de acuerdo con la

opción en la parte inferior de la primera página
- La biblioteca estándar, conjunto de pruebas, lanzador y pip serán instalados
- Si se selecciona, el directorio de instalación se agregará a su **PATH**
- Los accesos directos solo serán visibles para al usuario actual

Si selecciona «Customize installation» podrá elegir qué funciones instalar, el destino de la instalación y otras opciones o acciones posinstalación. Para instalar símbolos de depuración o binarios, necesitará usar esta opción.

Para realizar una instalación para todos los usuarios, deberá seleccionar «Customize installation». En este caso:

- Es posible que deba proporcionar credenciales administrativas o aprobación
- Python será instalado en el directorio Program Files
- El Python install manager será instalado en el directorio Windows
- Se pueden seleccionar características opcionales durante la instalación
- La biblioteca estándar puede ser precompilada a bytecode
- Si se selecciona, el directorio de instalación será agregado al **PATH** del sistema
- Los accesos directos están disponibles para todos los usuarios

4.11.2. Removing the MAX_PATH limitation

Windows históricamente ha limitado la longitud de las rutas a 260 caracteres. Esto significaba que rutas de mayor longitud no resolverían y se producirían errores.

En las últimas versiones de Windows, esta limitación se puede ampliar a aproximadamente 32.000 caracteres. Su administrador deberá activar la política de grupo «Habilitar rutas largas de Win32» o establecer `LongPathsEnabled` en `1` en la clave de registro `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem`.

Esto permite que la función `open()`, el módulo `os` y la mayoría de las demás funciones de ruta acepten y retornen rutas de más de 260 caracteres.

Luego de cambiar la opción anterior, no es necesaria ninguna otra configuración.

Distinto en la versión 3.6: Se habilitó el soporte para rutas largas en Python.

4.11.3. Installing without UI

Todas las opciones disponibles desde la interfaz de usuario del instalador también pueden especificarse desde la línea de comandos, lo cual permite que instaladores mediante scripts repliquen una instalación en muchas máquinas sin la interacción del usuario. Estas opciones también pueden ser configuradas sin anular la interfaz de usuario con el fin de cambiar alguno de los valores predeterminados.

Las siguientes opciones (encontradas al ejecutar el instalador con `/?`) pueden ser pasadas al instalador

Nombre	**Descripción**
/passive	para mostrar el progreso sin requerir interacción del usuario
/quiet	para instalar/desinstalar sin mostrar ninguna interfaz
/simple	para prevenir la personalización del usuario
/uninstall	para eliminar Python (sin confirmación)
/layout [directorio]	para pre-descargar todos los componentes
/log [nombre de archivo]	para especificar la ubicación de los archivos de registro

Todas las otras opciones se especifican con la forma `nombre=valor`, siendo el valor usualmente `0` para deshabilitar una funcionalidad, `1` para habilitar una funcionalidad, o una ruta. La lista completa de opciones disponibles se muestra a continuación.

Nombre	**Descripción**	**Predeterminado**
InstallAllUsers	Realizar una instalación en todo el sistema.	0
TargetDir	El directorio de instalación	Seleccionado de acuerdo a InstallAllUsers
DefaultAllUsersTargetDir	El directorio predeterminado de instalación cuando se instala para todos los usuarios	`%ProgramFiles%\Python X.Y` o `%ProgramFiles(x86)%\Python X.Y`

DefaultJustForMeTargetDir	El directorio de instalación predeterminado para instalaciones del usuario actual solamente	`%LocalAppData%\Programs\Python\PythonXY` or `%LocalAppData%\Programs\Python\PythonXY-32` or `%LocalAppData%\Programs\Python\PythonXY-64`
DefaultCustomTargetDir	Valor predeterminado de directorio de instalación personalizado que se muestra en la interfaz de usuario	(vacío)
AssociateFiles	Crear asociaciones de archivos si el lanzador también es instalado.	1
CompileAll	Compilar todos los archivos `.py` a `.pyc`.	0
PrependPath	Anteponer los directorios de instalación y scripts a **`PATH`** y agregue `.PY` a **`PATHEXT`**	0
AppendPath	Agregue directorios de instalación y scripts a **`PATH`** y agregue `.PY` a **`PATHEXT`**	0
Shortcuts	Crear accesos directos para el intérprete, documentación e IDLE si está instalado.	1
Include_doc	Instalar el manual de Python	1
Include_debug	Instalar los binarios de depuración	0
Include_dev	Instale encabezados y bibliotecas de desarrollador. Omitir esto puede conducir a una instalación inutilizable.	1
Include_exe	Instale `python.exe` y archivos relacionados. Omitir esto puede conducir a una instalación inutilizable.	1
Include_launcher	Instalar Python install manager.	1
InstallLauncherAllUsers	Instala el lanzador para todos los usuarios. También requiere que `Include_launcher` se establezca en 1	1
Include_lib	Instale la biblioteca estándar y los módulos de extensión. Omitir esto puede conducir a una instalación inutilizable.	1
Include_pip	Instalar los paquetes pip y setuptools	1
Include_symbols	Instalar los símbolos de depuración (`*.pdb`)	0
Include_tcltk	Instalar IDLE y soporte para Tcl/Tk	1
Include_test	Instalar el conjunto de pruebas de la biblioteca estándar	1
Include_tools	Instalar scripts de utilidades	1
LauncherOnly	Instalar solo el lanzador. Esto anulará la mayoría de las otras opciones.	0
SimpleInstall	Deshabilitar muchas de las partes de la interfaz de usuario	0
SimpleInstallDescription	Un mensaje personalizado para mostrar cuando se use la versión simplificada de la interfaz de usuario de instalación.	(vacío)

Por ejemplo, para realizar de forma silenciosa una instalación predeterminada de Python para todo el sistema, se puede usar el siguiente comando (desde un símbolo del sistema con privilegios elevados):

```
python-3.9.0.exe /quiet InstallAllUsers=1 PrependPath=1
```

Para permitir que los usuarios instalen fácilmente una copia personal de Python sin el conjunto de pruebas, se puede proporcionar un acceso directo con el siguiente comando. Esto mostrará una página inicial simplificada y no permitirá la personalización:

```
python-3.9.0.exe InstallAllUsers=0 Include_launcher=0 In
    SimpleInstall=1 SimpleInstallDescription="Just for m
```

(Tener en cuenta que al omitir el lanzador también se omiten las asociaciones de archivos y solo es recomendable hacerlo para instalaciones por usuario cuando ya hay una instalación en todo el sistema que incluye el lanzador.)

Las opciones enumeradas anteriormente también se pueden proporcionar en un archivo de nombre `unattend.xml` junto al ejecutable. Este archivo especifica una lista de opciones y valores. Cuando un valor se proporciona como un atributo, se convertirá a número si es posible. Los valores proporcionados como elementos de texto siempre se dejan como cadenas. Este archivo de ejemplo configura las mismas opciones que el ejemplo anterior:

```
<Options>
    <Option Name="InstallAllUsers" Value="no" />
    <Option Name="Include_launcher" Value="0" />
    <Option Name="Include_test" Value="no" />
    <Option Name="SimpleInstall" Value="yes" />
    <Option Name="SimpleInstallDescription">Just for me,
</Options>
```

4.11.4. Installing without downloading

Como algunas características de Python no se incluyen con la descarga inicial del instalador, la selección de estas características podría requerir de una conexión a internet. Para evitar esta necesidad, todos los posibles componentes pueden ser descargados a pedido para crear una estructura que no necesitará una conexión a internet, independientemente de las características que se seleccionen. Tener en cuenta que esta descarga puede ser más grande de lo necesario, pero si se va a realizar un gran número de instalaciones es muy útil tener una copia en la caché local.

Ejecute el siguiente comando desde el símbolo del sistema para descargar todos los archivos necesarios posibles. Recuerde sustituir `python-3.9.0.exe` por el nombre real de su instalador y crear diseños en sus propios directorios para evitar colisiones entre archivos con el mismo nombre.

```
python-3.9.0.exe /layout [optional target directory]
```

También se puede especificar la opción `/quiet` para no mostrar el progreso.

4.11.5. Modificar una instalación

Una vez que Python ha sido instalado, se puede agregar o quitar funciones a través de la herramienta Programas y características que es parte de Windows. Seleccionar la entrada Python y elegir «Desinstalar/Cambiar» para abrir el instalador en modo mantenimiento.

«Cambiar» permite agregar o eliminar características modificando las casillas de verificación - aquellas casillas que no se cambien no agregarán ni quitarán nada. Algunas opciones no pueden cambiarse de esta forma, como el directorio de instalación; para modificarlas es necesario eliminar y reinstalar Python completamente.

«Reparar» verificará todos los archivos que deben instalarse con la configuración actual y reemplazará los que se hayan eliminado o modificado.

«Desinstalar» eliminará Python completamente, a excepción del Python install manager, el cual posee su propia entrada en Programas y características.

4.11.6. Installing free-threaded binaries

Added in version 3.13.

To install pre-built binaries with free-threading enabled (see **PEP 703** [https://peps.python.org/pep-0703/]), you should select «Customize installation». The second page of options includes the «Download free-threaded binaries» checkbox.

Selecting this option will download and install additional binaries to the same location as the main Python install. The main executable is called `python3.13t.exe`, and other binaries either receive a `t` suffix or a full ABI suffix. Python source files and bundled third-party dependencies are shared with the main install.

The free-threaded version is registered as a regular Python install with the tag `3.13t` (with a `-32` or `-arm64` suffix as normal for those platforms). This allows tools to discover it, and for the Python install manager to support `py.exe -3.13t`. Note that the launcher will interpret `py.exe -3` (or a `python3` shebang) as «the latest 3.x install», which will prefer the free-threaded binaries over the regular ones, while `py.exe -3.13` will not. If you use the short style of option, you may prefer to not install the free-threaded binaries at this time.

To specify the install option at the command line, use `Include_freethreaded=1`. See Installing without downloading for instructions on pre-emptively downloading the additional binaries for offline install. The options to include debug symbols and binaries also apply to the free-threaded builds.

Free-threaded binaries are also available on nuget.org.

4.12. Python launcher for Windows (deprecated)

Obsoleto desde la versión 3.14: The launcher and this documentation have been superseded by the Python Install Manager described above. This is preserved temporarily for historical interest.

Added in version 3.3.

El lanzador de Python para Windows es una utilidad que ayuda en la ubicación y ejecución de diferentes versiones de Python. Este

permite que los scripts (o la línea de comandos) indiquen preferencia por una versión específica de Python, y ubicará y ejecutará esa versión.

A diferencia de la variable **`PATH`**, el lanzador seleccionará correctamente la versión más apropiada de Python. Priorizará instalaciones del usuario por sobre instalaciones de todo el sistema, y ordenará por versión del lenguaje en lugar de utilizar la más recientemente instalada.

El lanzador se especificó originalmente en **PEP 397** [https://peps.python.org/pep-0397/].

4.12.1. Comenzar

4.12.1.1. Desde la línea de comandos

Distinto en la versión 3.6.

Las instalaciones de todo el sistema de Python 3.3 y versiones posteriores colocarán el lanzador en su **`PATH`**. El lanzador es compatible con todas las versiones disponibles de Python, por lo que no importa qué versión esté instalada. Para comprobar que el lanzador está disponible, ejecute el siguiente comando en el símbolo del sistema:

```
py
```

Debería suceder que se inicia la última versión de Python instalada - se puede cerrar normalmente, y todo argumento adicional especificado por línea de comandos será enviado directamente a Python.

Si tiene varias versiones de Python instaladas (por ejemplo, 3.7 y 3.14), habrá notado que Python 3.14 se inició: para iniciar Python 3.7, pruebe el comando:

```
py -3.7
```

Si quieres la última versión de Python 2 que tienes instalada, prueba el comando:

```
py -2
```

Si ve el siguiente error, no tiene instalado el lanzador:

```
'py' is not recognized as an internal or external comman
operable program or batch file.
```

El comando:

```
py --list
```

muestra la(s) versión(es) actualmente instalada(s) de Python.

El argumento `-x.y` es la forma corta del argumento `-V:Compañía/Etiqueta`, que permite seleccionar un entorno de ejecución Python específico, incluidos aquellos que pueden haber provenido de lugares diferentes a python.org. Cualquier entorno registrado siguiendo el **PEP 514** [https://peps.python.org/pep-0514/] será detectable. El comando `--list` muestra todos los entornos disponibles usando el formato `-V:`.

Al usar el argumento `-V:`, especificar la Compañía limitará la selección a entornos de ese proveedor, mientras que especificar solo la Etiqueta seleccionará de todos los proveedores. Tenga en cuenta que omitir la barra implica una etiqueta:

```
# Select any '3.*' tagged runtime
py -V:3

# Select any 'PythonCore' released runtime
py -V:PythonCore/

# Select PythonCore's latest Python 3 runtime
py -V:PythonCore/3
```

La forma corta del argumento (`-3`) solo selecciona de las versiones principales de Python y no de otras distribuciones. Sin embargo, la forma más larga (`-V:3`) seleccionará de cualquiera.

The Company is matched on the full string, case-insensitive. The Tag is matched on either the full string, or a prefix, provided the next character is a dot or a hyphen. This allows `-V:3.1` to match `3.1-32`, but not `3.10`. Tags are sorted using numerical ordering (`3.10` is newer than `3.1`), but are compared using text (`-V:3.01` does not match `3.1`).

4.12.1.2. Entornos virtuales

Added in version 3.5.

Si el lanzador es ejecutado sin explícita especificación de la versión de Python, y un entorno virtual se encuentra activo (creado con el módulo **`venv`** de la biblioteca estándar o con la herramienta externa `virtualenv`), el lanzador ejecutará el intérprete del entorno virtual en lugar del global. Para ejecutar el intérprete global, desactive el entorno virtual o especifique explícitamente la versión global de Python.

4.12.1.3. Desde un script

Vamos a crear un script de Python para una prueba - cree un archivo llamado `hello.py` con el siguiente contenido

```
#! python
import sys
sys.stdout.write("hello from Python %s\n" % (sys.versior
```

Desde el directorio en el que vive hello.py, ejecute el comando:

```
py hello.py
```

Debería notar que se imprime el número de versión de la última instalación de Python 2.x. Ahora pruebe cambiando la primera línea por:

```
#! python3
```

Volver a ejecutar el comando ahora debería imprimir la información más reciente de Python 3.x. Al igual que con los ejemplos de línea de comandos anteriores, puede especificar un calificador de versión más explícito. Suponiendo que tiene instalado

Python 3.7, intente cambiar la primera línea a `#! python3.7` y debería encontrar la información de la versión 3.7 impresa.

Tenga en cuenta que a diferencia del uso interactivo, el comando «python» (a secas) utilizará la última versión de Python 2.x que esté instalada. Esto es así por compatibilidad con versiones anteriores y por compatibilidad con Unix, donde el comando `python` usualmente refiere a Python 2.

4.12.1.4. Desde asociaciones de archivos

El lanzador debería haber sido asociado con archivos de Python (por ej. archivos `.py`, `.pyw` y `.pyc`) cuando fue instalado. Esto significa que cuando se haga doble click sobre alguno de estos archivos desde el explorador de Windows se utilizará el lanzador, por lo que se pueden utilizar las mismas funciones descritas anteriormente para que el script especifique la versión que debería usarse.

El beneficio clave de esto es que un único lanzador puede soportar múltiples versiones de Python al mismo tiempo dependiendo del contenido de la primera línea.

4.12.2. Shebang lines

Si la primera línea de un script comienza con `#!`, esta se denomina línea «shebang». Linux y otros sistemas operativos tipo Unix soportan de forma nativa este tipo de líneas y son comúnmente utilizadas en dichos sistemas para indicar cómo debería ser ejecutado un script. Este lanzador permite que la misma funcionalidad pueda ser utilizada con scripts de Python en Windows, y los ejemplos anteriores demuestran su uso.

Para permitir que las líneas shebang de scripts de Python sean trasladables entre Unix y Windows, este lanzador soporta varios comandos “virtuales” para especificar qué intérprete utilizar. Los comandos virtuales soportados son:

- `/usr/bin/env`
- `/usr/bin/python`

- `/usr/local/bin/python`
- `python`

Por ejemplo, si la primera línea del script comienza con

```
#! /usr/bin/python
```

The default Python or an active virtual environment will be located and used. As many Python scripts written to work on Unix will already have this line, you should find these scripts can be used by the launcher without modification. If you are writing a new script on Windows which you hope will be useful on Unix, you should use one of the shebang lines starting with `/usr`.

Any of the above virtual commands can be suffixed with an explicit version (either just the major version, or the major and minor version). Furthermore the 32-bit version can be requested by adding «-32» after the minor version. I.e. `/usr/bin/python3.7-32` will request usage of the 32-bit Python 3.7. If a virtual environment is active, the version will be ignored and the environment will be used.

Added in version 3.7: Desde la versión 3.7 del lanzador de Python es posible solicitar la versión de 64 bit con el sufijo «-64». Además es posible especificar una versión mayor y la arquitectura sin la versión menor (por ej. `/usr/bin/python3-64`).

Distinto en la versión 3.11: El sufijo «-64» está en desuso y ahora implica «cualquier arquitectura que comprobable no sea i386/32 bits». Para solicitar un entorno específico, use el nuevo argumento `-V:TAG` con la etiqueta completa.

Distinto en la versión 3.13: Virtual commands referencing `python` now prefer an active virtual environment rather than searching **PATH**. This handles cases where the shebang specifies `/usr/bin/env python3` but `python3.exe` is not present in the active environment.

The `/usr/bin/env` form of shebang line has one further special property. Before looking for installed Python interpreters, this form will search the executable **PATH** for a Python executable matching the name provided as the first argument. This corresponds to the behaviour of the Unix `env` program, which performs a **`PATH`** search. If an executable matching the first argument after the `env` command cannot be found, but the argument starts with `python`, it will be handled as described for the other virtual commands. The environment variable **`PYLAUNCHER_NO_SEARCH_PATH`** may be set (to any value) to skip this search of **`PATH`**.

Las líneas shebang que no coincidan con ninguno de estos patrones se buscan en la sección `[commands]` del archivo .INI del lanzador. Esto se puede usar para manejar ciertos comandos de una manera que tenga sentido para tu sistema. El nombre del comando debe ser un único argumento (sin espacios en el ejecutable shebang), y el valor sustituido es la ruta completa al ejecutable (los argumentos adicionales especificados en el .INI se citarán como parte del nombre del archivo).

```
[commands]
/bin/xpython=C:\Program Files\XPython\python.exe
```

Cualquier comando que no se encuentre en el archivo .INI se trata como rutas ejecutables de **Windows** que son absolutas o relativas al directorio que contiene el archivo de script. Esto es una comodidad para los scripts solo para Windows, como aquellos generados por un instalador, ya que el comportamiento no es compatible con los shells al estilo Unix. Estas rutas pueden estar entre comillas y pueden incluir varios argumentos, tras los cuales se añadirá la ruta al script y cualquier argumento adicional.

4.12.3. Argumentos en líneas shebang

Las líneas shebang también pueden especificar opciones adicionales para que sean pasadas al intérprete de Python. Por ej. si se tiene esta línea shebang:

```
#! /usr/bin/python -v
```

Entonces Python se iniciará con la opción `-v`

4.12.4. Personalización

4.12.4.1. Personalización con archivos INI

El lanzador buscará dos archivos .ini - `py.ini` en el directorio de datos de aplicación del usuario actual (`%LOCALAPPDATA%` o `$env:LocalAppData`) y `py.ini` en el mismo directorio que el lanzador. Los mismos archivos .ini son utilizados tanto para la versión “consola” del lanzador (es decir, py.exe) como para la version “windows” (es decir pyw.exe).

La personalización especificada en el «directorio de aplicación» tendrá precedencia por sobre la que esté junto al ejecutable, por lo que un usuario, que podría no tener acceso de escritura al archivo .ini que está junto al lanzador, puede sobrescribir comandos en ese archivo .ini global.

4.12.4.2. Personalizar las versiones de Python predeterminadas

En algunos casos, un calificador de versión puede ser incluido en un comando para dictar qué versión de Python será utilizada por dicho comando. Un calificador de versión comienza con el número mayor de la versión y pude ser seguido opcionalmente por un punto (“.”) y el número menor de la versión. Además es posible especificar si se solicita una implementación de 32 o 64 bit agregando «-32» o «-64».

Por ejemplo, una línea shebang como `#!python` no posee calificador de versión, mientras que `#!python3` sí tiene un calificador de versión el cual especifica solo el número mayor de la versión.

If no version qualifiers are found in a command, the environment variable **`PY_PYTHON`** can be set to specify the default version qualifier. If it is not set, the default is «3». The variable can specify any value that may be passed on the command line, such as «3», «3.7», «3.7-32» or «3.7-64». (Note that the «-64» option is only available with the launcher included with Python 3.7 or newer.)

Si no se encuentra ningún calificador de versión menor, la variable de entorno `PY_PYTHON{major}` (donde `{major}` es el actual calificador de versión mayor según lo determinado antes) puede ser configurada para especificar la versión completa. Si dicha opción no se encuentra, el lanzador enumerará las versiones de Python instaladas y utilizará la última versión menor encontrada para la versión mayor, la cual es probable, aunque no se garantiza, que sea la versión más recientemente instalada de esa familia.

En un Windows de 64 bit con ambas implementaciones de 32 y 64 bit de la misma versión (mayor.menor) de Python instaladas, la versión de 64 bit siempre tendrá precedencia. Esto se cumple para ambas implementaciones de 32 y 64 bit del lanzador - un lanzador de 32 bit priorizará ejecutar una instalación de Python de 64 bit de la versión especificada si está disponible. Esto es así para que el comportamiento del lanzador pueda ser predecible sabiendo solamente qué versiones están instaladas en la PC y sin importar el orden en el cual fueron instaladas (esto es, sin saber si una versión de Python de 32 o 64 bit y su correspondiente lanzador fue la última instalada). Como se especificó antes, el sufijo «-32» o «-64» puede ser utilizado en el especificador de versión para cambiar este comportamiento.

Ejemplos:

- Si no se configura ninguna opción relevante, los comandos `python` y `python2` utilizarán la última versión de Python 2.x instalada y el comando `python3` utilizará el último Python 3.x instalado.
- El comando `python3.7` no consultará ninguna opción ya que las versiones están completamente especificadas.
- Si `PY_PYTHON=3`, los comandos `python` y `python3` utilizarán ambos la última versión instalada de Python 3.
- Si es `PY_PYTHON=3.7-32`, el comando `python` usará la implementación de 32 bits de 3.7, mientras que el comando `python3` usará la última versión de Python instalada (PY_PYTHON no se consideró en absoluto porque se especificó una versión principal).
- Si `PY_PYTHON=3` y `PY_PYTHON3=3.7`, los comandos `python` y `python3` usarán específicamente 3.7

Además de las variables de entorno, las mismas configuraciones pueden realizarse desde el archivo .INI utilizado por el lanzador. La sección en el archivo INI se llama `[defaults]` y el nombre de cada clave será igual al de la variable de entorno pero sin el prefijo

`PY_` (tenga en cuenta que los nombres de clave en el archivo INI son indiferentes a mayúsculas y minúsculas). El contenido de las variables de entorno sobrescribirá los valores especificados en el archivo INI.

Por ejemplo:

- La configuración de `PY_PYTHON=3.7` es equivalente al archivo INI que contiene:

```
[defaults]
python=3.7
```

- La configuración de `PY_PYTHON=3` y `PY_PYTHON3=3.7` es equivalente al archivo INI que contiene:

```
[defaults]
python=3
python3=3.7
```

4.12.5. Diagnóstico

If an environment variable **`PYLAUNCHER_DEBUG`** is set (to any value), the launcher will print diagnostic information to stderr (i.e. to the console). While this information manages to be simultaneously verbose *and* terse, it should allow you to see what versions of Python were located, why a particular version was chosen and the exact command-line used to execute the target Python. It is primarily intended for testing and debugging.

4.12.6. Dry run

If an environment variable **`PYLAUNCHER_DRYRUN`** is set (to any value), the launcher will output the command it would have run, but will not actually launch Python. This may be useful for tools that want to use the launcher to detect and then launch Python directly. Note that the command written to standard output is always encoded using UTF-8, and may not render correctly in the console.

4.12.7. Instalación bajo demanda

If an environment variable **`PYLAUNCHER_ALLOW_INSTALL`** is set (to any value), and the requested Python version is not installed but is available on the Microsoft Store, the launcher will attempt to install it. This may require user interaction to complete, and you may need to run the command again.

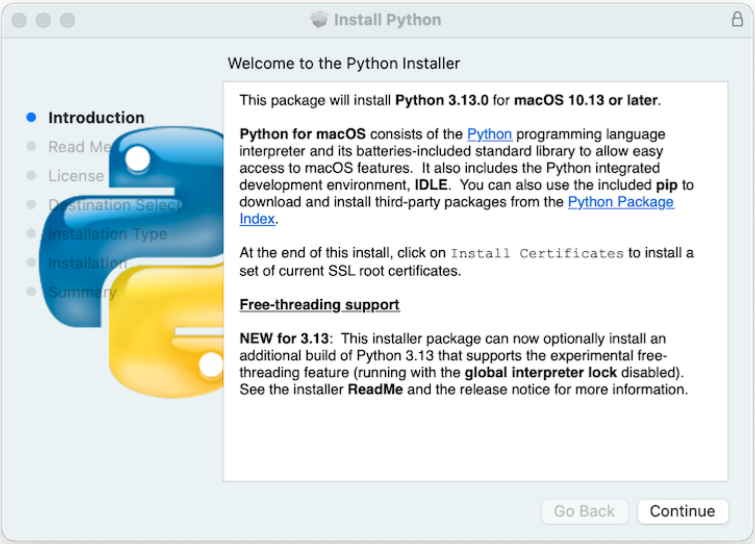
An additional **`PYLAUNCHER_ALWAYS_INSTALL`** variable causes the launcher to always try to install Python, even if it is detected. This is mainly intended for testing (and should be used with **`PYLAUNCHER_DRYRUN`**).

4.12.8. Códigos de retorno

El lanzador de Python puede retornar los siguientes códigos de salida. Desafortunadamente, no hay forma de distinguirlos del código de salida de Python.

Los nombres de los códigos son como se usan en las fuentes y son solo para referencia. No hay forma de acceder a ellos o resolverlos aparte de leer esta página. Las entradas se enumeran en orden alfabético de nombres.

Nombre	**Valor**	**Descripción**
RC_BAD_VENV_CFG	107	Se encontró un `pyvenv.cfg` pero está corrupto.
RC_CREATE_PROCESS	101	No se pudo iniciar Python.
RC_INSTALLING	111	Se inició una instalación, pero será necesario volver a ejecutar el comando una vez que se complete.
RC_INTERNAL_ERROR	109	Error inesperado. Por favor, informe de un error.
RC_NO_COMMANDLINE	108	No se puede obtener la línea de comandos del sistema operativo.
RC_NO_PYTHON	103	No se puede encontrar la versión solicitada.
RC_NO_VENV_CFG	106	Se requería un `pyvenv.cfg` pero no se encontró.


Install Python
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
Go Back
Continue

Welcome to the Python Installer

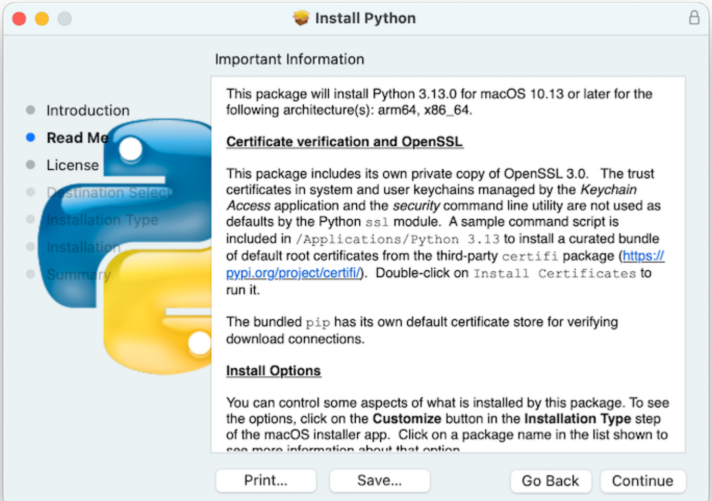
This package will install **Python 3.13.0** for **macOS 10.13 or later**.

Python for macOS consists of the Python programming language interpreter and its batteries-included standard library to allow easy access to macOS features. It also includes the Python integrated development environment, **IDLE**. You can also use the included **pip** to download and install third-party packages from the Python Package Index.

At the end of this install, click on `Install Certificates` to install a set of current SSL root certificates.

Free-threading support

NEW for 3.13: This installer package can now optionally install an additional build of Python 3.13 that supports the experimental free-threading feature (running with the **global interpreter lock** disabled). See the installer **ReadMe** and the release notice for more information.


Install Python
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
Print...
Save...
Go Back
Continue

Important Information

This package will install Python 3.13.0 for macOS 10.13 or later for the following architecture(s): arm64, x86_64.

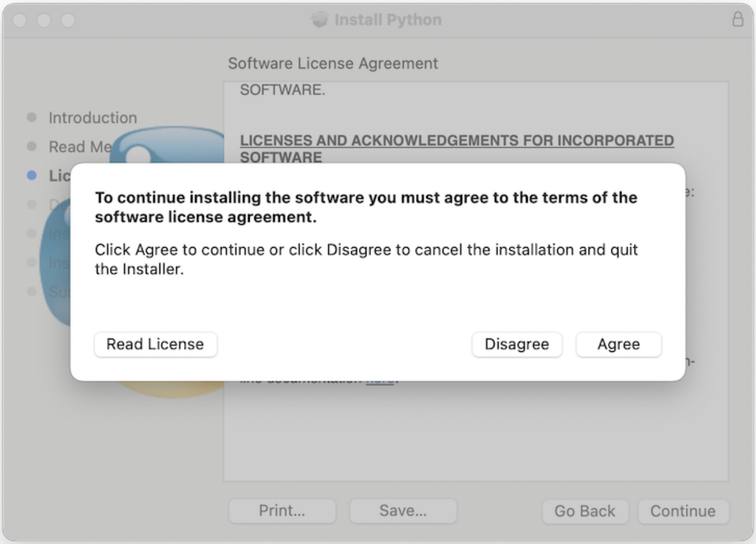
Certificate verification and OpenSSL

This package includes its own private copy of OpenSSL 3.0. The trust certificates in system and user keychains managed by the *Keychain Access* application and the *security* command line utility are not used as defaults by the Python `ssl` module. A sample command script is included in `/Applications/Python 3.13` to install a curated bundle of default root certificates from the third-party `certifi` package (https://pypi.org/project/certifi/). Double-click on `Install Certificates` to run it.

The bundled `pip` has its own default certificate store for verifying download connections.

Install Options

You can control some aspects of what is installed by this package. To see the options, click on the **Customize** button in the **Installation Type** step of the macOS installer app. Click on a package name in the list shown to see more information about that option.


Install Python
Introduction
Read Me
Lic
Print...
Save...
Go Back
Continue
Read License
Disagree
Agree

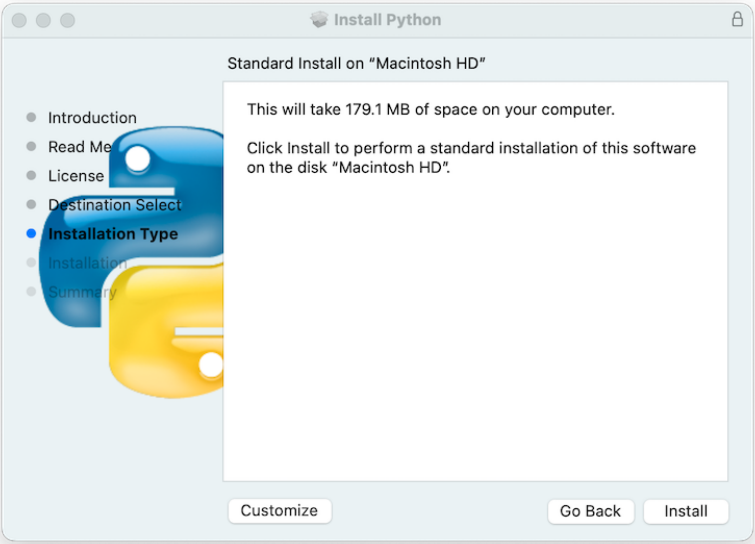
Software License Agreement

SOFTWARE.

LICENSES AND ACKNOWLEDGEMENTS FOR INCORPORATED SOFTWARE

To continue installing the software you must agree to the terms of the software license agreement.

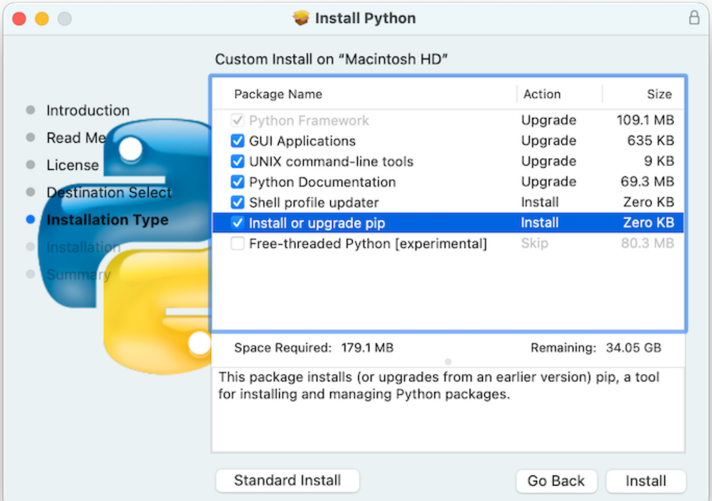
Click Agree to continue or click Disagree to cancel the installation and quit the Installer.


Install Python
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
Customize
Go Back
Install

Standard Install on “Macintosh HD”

This will take 179.1 MB of space on your computer.

Click Install to perform a standard installation of this software on the disk “Macintosh HD”.

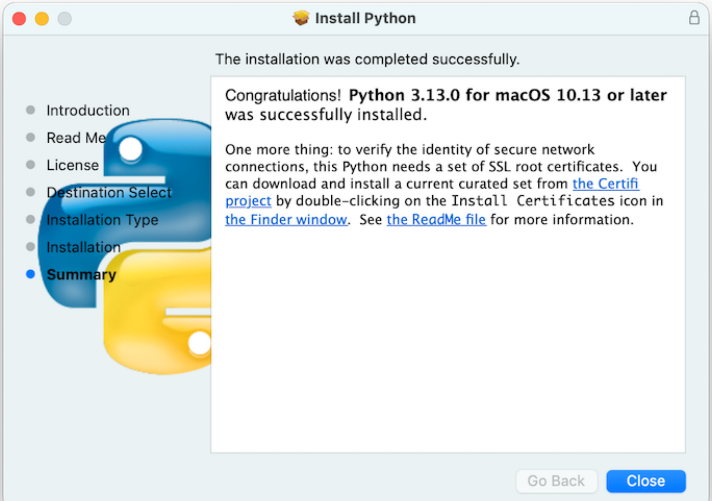

Install Python
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
Standard Install
Go Back
Install

Custom Install on “Macintosh HD”

Package Name	Action	Size
☑ Python Framework	Upgrade	109.1 MB
☑ GUI Applications	Upgrade	635 KB
☑ UNIX command-line tools	Upgrade	9 KB
☑ Python Documentation	Upgrade	69.3 MB
☑ Shell profile updater	Install	Zero KB
☑ Install or upgrade pip	Install	Zero KB
☐ Free-threaded Python [experimental]	Skip	80.3 MB

Space Required: 179.1 MB Remaining: 34.05 GB

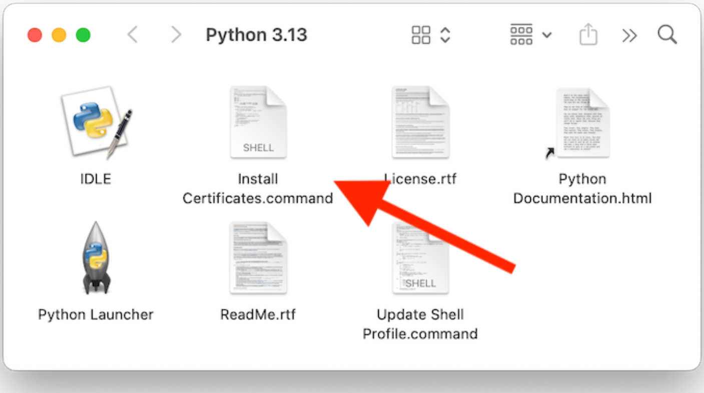
This package installs (or upgrades from an earlier version) pip, a tool for installing and managing Python packages.

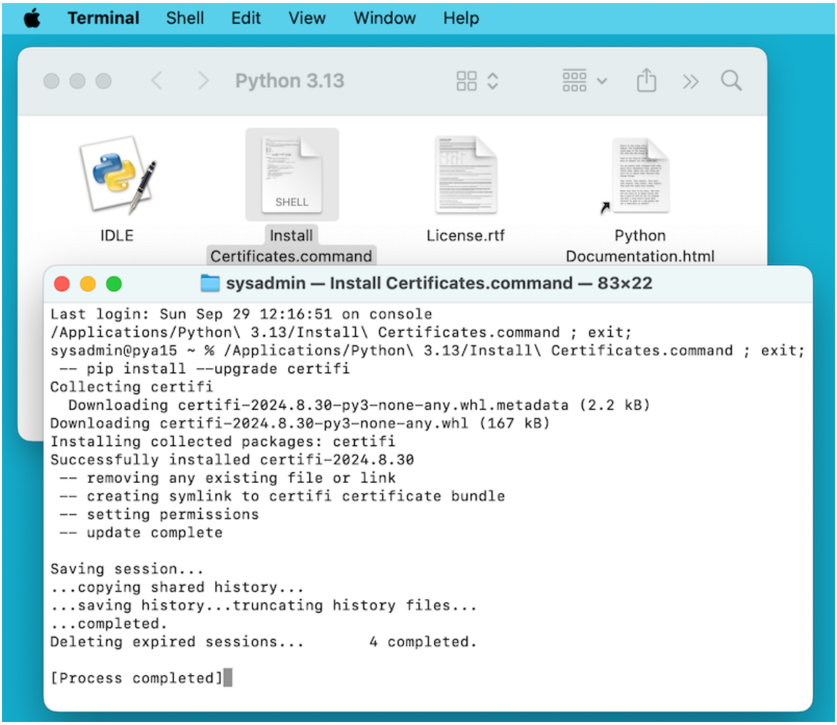

Install Python
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
Go Back
Close

The installation was completed successfully.

Congratulations! **Python 3.13.0 for macOS 10.13 or later** was successfully installed.

One more thing: to verify the identity of secure network connections, this Python needs a set of SSL root certificates. You can download and install a current curated set from the Certifi project by double-clicking on the `Install Certificates` icon in the Finder window. See the ReadMe file for more information.

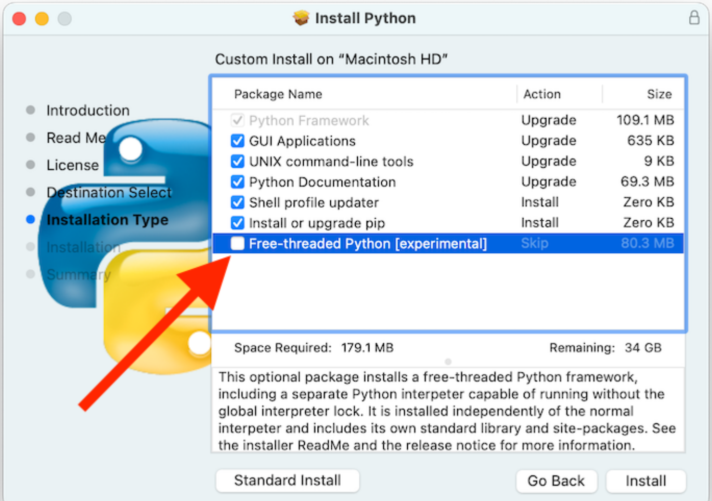

Python 3.13
IDLE
SHELL
Install Certificates.command
License.rtf
Python Documentation.html
Python Launcher
ReadMe.rtf
SHELL
Update Shell Profile.command


Terminal
Shell
Edit
View
Window
Help
Python 3.13
IDLE
SHELL
Install Certificates.command
License.rtf
Python Documentation.html
sysadmin — Install Certificates.command — 83×22

```
Last login: Sun Sep 29 12:16:51 on console
/Applications/Python\ 3.13/Install\ Certificates.command ; exit;
sysadmin@pya15 ~ % /Applications/Python\ 3.13/Install\ Certificates.command ; exit;
 -- pip install --upgrade certifi
Collecting certifi
  Downloading certifi-2024.8.30-py3-none-any.whl.metadata (2.2 kB)
Downloading certifi-2024.8.30-py3-none-any.whl (167 kB)
Installing collected packages: certifi
Successfully installed certifi-2024.8.30
 -- removing any existing file or link
 -- creating symlink to certifi certificate bundle
 -- setting permissions
 -- update complete

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.
Deleting expired sessions...       4 completed.

[Process completed]
```



Custom Install on “Macintosh HD”

Package Name	Action	Size
☑ Python Framework	Upgrade	109.1 MB
☑ GUI Applications	Upgrade	635 KB
☑ UNIX command-line tools	Upgrade	9 KB
☑ Python Documentation	Upgrade	69.3 MB
☑ Shell profile updater	Install	Zero KB
☑ Install or upgrade pip	Install	Zero KB
☐ Free-threaded Python [experimental]	Skip	80.3 MB

Space Required: 179.1 MB Remaining: 34 GB

This optional package installs a free-threaded Python framework, including a separate Python interpeter capable of running without the global interpreter lock. It is installed independently of the normal interpeter and includes its own standard library and site-packages. See the installer ReadMe and the release notice for more information.

5. Using Python on macOS

This document aims to give an overview of macOS-specific behavior you should know about to get started with Python on Mac computers. Python on a Mac running macOS is very similar to Python on other Unix-derived platforms, but there are some differences in installation and some features.

There are various ways to obtain and install Python for macOS. Pre-built versions of the most recent versions of Python are available from a number of distributors. Much of this document describes use of the Pythons provided by the CPython release team for download from the python.org website [https://www.python.org/downloads/]. See Alternative Distributions for some other options.

5.1. Using Python for macOS from `python.org`

5.1.1. Installation steps

For current Python versions [https://www.python.org/downloads/] (other than those in `security` status), the release team produces a **Python for macOS** installer package for each new release. A list of available installers is available here [https://www.python.org/downloads/macos/]. We recommend using the most recent supported Python version where possible. Current installers provide a universal2 binary [https://en.wikipedia.org/wiki/Universal_binary] build of Python which runs natively on all Macs (Apple Silicon and Intel) that are supported by a wide range of macOS versions, currently typically from at least **macOS 10.15 Catalina** on.

The downloaded file is a standard macOS installer package file (`.pkg`). File integrity information (checksum, size, sigstore signature, etc) for each file is included on the release download page. Installer packages and their contents are signed and notarized with `Python Software Foundation` Apple Developer ID certificates to meet macOS Gatekeeper requirements [https://support.apple.com/en-us/102445].

For a default installation, double-click on the downloaded installer package file. This should launch the standard macOS Installer app and display the first of several installer windows steps.

Clicking on the **Continue** button brings up the **Read Me** for this installer. Besides other important information, the **Read Me** documents which Python version is going to be installed and on what versions of macOS it is supported. You may need to scroll through to read the whole file. By default, this **Read Me** will also be installed in `/Applications/Python 3.14/` and available to read anytime.

Clicking on **Continue** proceeds to display the license for Python and for other included software. You will then need to **Agree** to the license terms before proceeding to the next step. This license file will also be installed and available to be read later.

After the license terms are accepted, the next step is the **Installation Type** display. For most uses, the standard set of installation operations is appropriate.

By pressing the **Customize** button, you can choose to omit or select certain package components of the installer. Click on each package name to see a description of what it installs. To also install support for the optional free-threaded feature, see Installing Free-threaded Binaries.

In either case, clicking **Install** will begin the install process by asking permission to install new software. A macOS user name with `Administrator` privilege is needed as the installed Python will be available to all users of the Mac.

When the installation is complete, the **Summary** window will appear.

Double-click on the **Install Certificates.command** icon or file in the `/Applications/Python 3.14/` window to complete the installation.

This will open a temporary **Terminal** shell window that will use the new Python to download and install SSL root certificates for its use.

If `Successfully installed certifi` and `update complete` appears in the terminal window, the installation is complete. Close this terminal window and the installer window.

A default install will include:

- A `Python 3.14` folder in your `Applications` folder. In here you find **IDLE**, the development environment that is a standard part of official Python distributions; and **Python Launcher**, which handles double-clicking Python scripts from the macOS Finder [https://support.apple.com/en-us/HT201732].
- A framework `/Library/Frameworks/Python.framework`, which includes the Python executable and libraries. The installer adds this location to your shell path. To uninstall Python, you can remove these three things. Symlinks to the Python executable are placed in `/usr/local/bin/`.

Nota

Recent versions of macOS include a **python3** command in `/`

`usr/bin/python3` that links to a usually older and incomplete version of Python provided by and for use by the Apple development tools, **Xcode** or the **Command Line Tools for Xcode**. You should never modify or attempt to delete this installation, as it is Apple-controlled and is used by Apple-provided or third-party software. If you choose to install a newer Python version from `python.org`, you will have two different but functional Python installations on your computer that can co-exist. The default installer options should ensure that its **python3** will be used instead of the system **python3**.

5.1.2. Cómo ejecutar un *script* de Python

There are two ways to invoke the Python interpreter. If you are familiar with using a Unix shell in a terminal window, you can invoke `python3.14` or `python3` optionally followed by one or more command line options (described in Línea de comandos y entorno). The Python tutorial also has a useful section on using Python interactively from a shell.

You can also invoke the interpreter through an integrated development environment. IDLE — Python editor and shell is a basic editor and interpreter environment which is included with the standard distribution of Python. **IDLE** includes a Help menu that allows you to access Python documentation. If you are completely new to Python, you can read the tutorial introduction in that document.

There are many other editors and IDEs available, see Editores e IDEs for more information.

To run a Python script file from the terminal window, you can invoke the interpreter with the name of the script file:

```
python3.14 myscript.py
```

To run your script from the Finder, you can either:

- Drag it to **Python Launcher**.
- Select **Python Launcher** as the default application to open

your script (or any `.py` script) through the Finder Info window and double-click it. **Python Launcher** has various preferences to control how your script is launched. Option-dragging allows you to change these for one invocation, or use its `Preferences` menu to change things globally.

Be aware that running the script directly from the macOS Finder might produce different results than when running from a terminal window as the script will not be run in the usual shell environment including any setting of environment variables in shell profiles. And, as with any other script or program, be certain of what you are about to run.

5.2. Alternative Distributions

Besides the standard `python.org` for macOS installer, there are third-party distributions for macOS that may include additional functionality. Some popular distributions and their key features:

ActivePython [https://www.activestate.com/products/python/]

Installer with multi-platform compatibility, documentation

Anaconda [https://www.anaconda.com/download/]

Popular scientific modules (such as numpy, scipy, and pandas) and the `conda` package manager.

Homebrew [https://brew.sh]

Package manager for macOS including multiple versions of Python and many third-party Python-based packages (including numpy, scipy, and pandas).

MacPorts [https://www.macports.org]

Another package manager for macOS including multiple versions of Python and many third-party Python-based packages. May include pre-built versions of Python and many packages for older versions of macOS.

Note that distributions might not include the latest versions of Python or other libraries, and are not maintained or supported by the core Python team.

5.3. Instalación de paquetes adicionales de Python

Refer to the Python Packaging User Guide [https://packaging.python.org/en/latest/tutorials/installing-packages/] for more information.

5.4. GUI Programming

Hay varias opciones para crear aplicaciones GUI en Mac con Python.

The standard Python GUI toolkit is `tkinter`, based on the cross-platform Tk toolkit (https://www.tcl.tk). A macOS-native version of Tk is included with the installer.

PyObjC is a Python binding to Apple's Objective-C/Cocoa framework. Information on PyObjC is available from pyobjc [https://pypi.org/project/pyobjc/].

A number of alternative macOS GUI toolkits are available including:

- PySide [https://www.qt.io/qt-for-python]: Official Python bindings to the Qt GUI toolkit [https://wiki.qt.io/Qt_for_Python].
- PyQt [https://riverbankcomputing.com/software/pyqt/]: Alternative Python bindings to Qt.
- Kivy [https://kivy.org]: A cross-platform GUI toolkit that supports desktop and mobile platforms.
- Toga [https://toga.readthedocs.io]: Part of the BeeWare Project [https://beeware.org]; supports desktop, mobile, web and console apps.
- wxPython [https://wxpython.org]: A cross-platform toolkit that supports desktop operating systems.

5.5. Advanced Topics

5.5.1. Installing Free-threaded Binaries

Added in version 3.13.

The `python.org` Python for macOS installer package can optionally install an additional build of Python 3.14 that supports **PEP 703** [https://peps.python.org/pep-0703/], the free-threading feature (running with the global interpreter lock disabled). Check the release page on `python.org` for possible updated information.

The free-threaded mode is working and continues to be improved, but there is some additional overhead in single-threaded workloads compared to the regular build. Additionally, third-party packages, in particular ones with an extension module, may not be ready for use in a free-threaded build, and will re-enable the GIL. Therefore, the support for free-threading is not installed by default. It is packaged as a separate install option, available by clicking the **Customize** button on the **Installation Type** step of the installer as described above.

If the box next to the **Free-threaded Python** package name is checked, a separate `PythonT.framework` will also be installed alongside the normal `Python.framework` in `/Library/Frameworks`. This configuration allows a free-threaded Python 3.14 build to co-exist on your system with a traditional (GIL only) Python 3.14 build with minimal risk while installing or testing. This installation layout may change in future releases.

Known cautions and limitations:

- The **UNIX command-line tools** package, which is selected by default, will install links in `/usr/local/bin` for `python3.14t`, the free-threaded interpreter, and `python3.14t-config`, a configuration utility which may be useful for package builders. Since `/usr/local/bin` is typically included in your shell `PATH`, in most cases no changes to your `PATH` environment variables should be needed to use `python3.14t`.
- For this release, the **Shell profile updater** package and the `Update Shell Profile.command` in `/Applications/Python 3.14/` do not support the free-threaded package.
- The free-threaded build and the traditional build have separate search paths and separate `site-packages`

directories so, by default, if you need a package available in both builds, it may need to be installed in both. The free-threaded package will install a separate instance of **pip** for use with `python3.14t`.

  - To install a package using **pip** without a **venv**:

    ```
    python3.14t -m pip install <package_name>
    ```

- When working with multiple Python environments, it is usually safest and easiest to create and use virtual environments. This can avoid possible command name conflicts and confusion about which Python is in use:

  ```
  python3.14t -m venv <venv_name>
  ```

  then **activate**.

- To run a free-threaded version of IDLE:

  ```
  python3.14t -m idlelib
  ```

- The interpreters in both builds respond to the same PYTHON environment variables which may have unexpected results, for example, if you have `PYTHONPATH` set in a shell profile. If necessary, there are command line options like `-E` to ignore these environment variables.

- The free-threaded build links to the third-party shared libraries, such as `OpenSSL` and `Tk`, installed in the traditional framework. This means that both builds also share one set of trust certificates as installed by the **Install Certificates.command** script, thus it only needs to be run once.

- If you cannot depend on the link in `/usr/local/bin` pointing to the `python.org` free-threaded `python3.14t` (for example, if you want to install your own version there or some other distribution does), you can explicitly set your shell `PATH` environment variable to include the `PythonT` framework `bin` directory:

```
export PATH="/Library/Frameworks/PythonT.framework/
```

The traditional framework installation by default does something similar, except for `Python.framework`. Be aware that having both framework `bin` directories in `PATH` can lead to confusion if there are duplicate names like `python3.14` in both; which one is actually used depends on the order they appear in `PATH`. The `which python3.x` or `which python3.xt` commands can show which path is being used. Using virtual environments can help avoid such ambiguities. Another option might be to create a shell **alias** to the desired interpreter, like:

```
alias py3.14="/Library/Frameworks/Python.framework/
alias py3.14t="/Library/Frameworks/PythonT.framewor
```

5.5.2. Installing using the command line

If you want to use automation to install the `python.org` installer package (rather than by using the familiar macOS **Installer** GUI app), the macOS command line **installer** utility lets you select non-default options, too. If you are not familiar with **installer**, it can be somewhat cryptic (see **man installer** for more information). As an example, the following shell snippet shows one way to do it, using the `3.14.0b2` release and selecting the free-threaded interpreter option:

```
RELEASE="python-3.140b2-macos11.pkg"

# download installer pkg
curl -O https://www.python.org/ftp/python/3.14.0/${RELEA

# create installer choicechanges to customize the instal
#    enable the PythonTFramework-3.14 package
#    while accepting the other defaults (install all oth
cat > ./choicechanges.plist <<EOF
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "ht
<plist version="1.0">
<array>
```

```
        <dict>
                <key>attributeSetting</key>
                <integer>1</integer>
                <key>choiceAttribute</key>
                <string>selected</string>
                <key>choiceIdentifier</key>
                <string>org.python.Python.PythonTFramewo
        </dict>
</array>
</plist>
EOF

sudo installer -pkg ./${RELEASE} -applyChoiceChangesXML
```

You can then test that both installer builds are now available with something like:

```
$ # test that the free-threaded interpreter was installe
$ /usr/local/bin/python3.14t -VV
Python 3.14.0b2 free-threading build (v3.14.0b2:3a83b172
$ #    and the traditional interpreter
$ /usr/local/bin/python3.14 -VV
Python 3.14.0b2 (v3.14.0b2:3a83b172af, Jun  5 2024, 12:5
$ # test that they are also available without the prefix
$ python3.14t -VV
Python 3.14.0b2 free-threading build (v3.14.0b2:3a83b172
$ python3.14 -VV
Python 3.14.0b2 (v3.14.0b2:3a83b172af, Jun  5 2024, 12:5
```

Nota

Current `python.org` installers only install to fixed locations like `/Library/Frameworks/`, `/Applications`, and `/usr/local/bin`. You cannot use the **installer** `-domain` option to install to other locations.

5.5.3. Distributing Python Applications

A range of tools exist for converting your Python code into a standalone distributable application:

- py2app [https://pypi.org/project/py2app/]: Supports creating macOS `.app` bundles from a Python project.
- Briefcase [https://briefcase.readthedocs.io]: Part of the BeeWare Project [https://beeware.org]; a cross-platform packaging tool that supports creation of `.app` bundles on macOS, as well as managing signing and notarization.
- PyInstaller [https://pyinstaller.org/]: A cross-platform packaging tool that creates a single file or folder as a distributable artifact.

5.5.4. App Store Compliance

Apps submitted for distribution through the macOS App Store must pass Apple's app review process. This process includes a set of automated validation rules that inspect the submitted application bundle for problematic code.

The Python standard library contains some code that is known to violate these automated rules. While these violations appear to be false positives, Apple's review rules cannot be challenged. Therefore, it is necessary to modify the Python standard library for an app to pass App Store review.

The Python source tree contains a patch file [https://github.com/python/cpython/tree/3.14/Mac/Resources/app-store-compliance.patch] that will remove all code that is known to cause issues with the App Store review process. This patch is applied automatically when CPython is configured with the **`--with-app-store-compliance`** option.

This patch is not normally required to use CPython on a Mac; nor is it required if you are distributing an app *outside* the macOS App Store. It is *only* required if you are using the macOS App Store as a distribution channel.

5.6. Otros recursos

The python.org Help page [https://www.python.org/about/help/] has links to many useful resources. The Pythonmac-SIG mailing list [https://www.python.org/community/sigs/current/pythonmac-sig/] is another support resource specifically for Python users and developers on the Mac.

6. Using Python on Android

Python on Android is unlike Python on desktop platforms. On a desktop platform, Python is generally installed as a system resource that can be used by any user of that computer. Users then interact with Python by running a **python** executable and entering commands at an interactive prompt, or by running a Python script.

On Android, there is no concept of installing as a system resource. The only unit of software distribution is an «app». There is also no console where you could run a **python** executable, or interact with a Python REPL.

As a result, the only way you can use Python on Android is in embedded mode – that is, by writing a native Android application, embedding a Python interpreter using `libpython`, and invoking Python code using the Python embedding API. The full Python interpreter, the standard library, and all your Python code is then packaged into your app for its own private use.

The Python standard library has some notable omissions and restrictions on Android. See the API availability guide for details.

6.1. Adding Python to an Android app

Most app developers should use one of the following tools, which will provide a much easier experience:

- Briefcase [https://briefcase.beeware.org], from the BeeWare project
- Buildozer [https://buildozer.readthedocs.io], from the Kivy project
- Chaquopy [https://chaquo.com/chaquopy]
- pyqtdeploy [https://www.riverbankcomputing.com/static/Docs/pyqtdeploy/]
- Termux [https://termux.dev/en/]

If you’re sure you want to do all of this manually, read on. You can use the testbed app [https://github.com/python/cpython/tree/3.14/Android/testbed] as a guide; each step below contains a link to the relevant file.

- First, acquire a build of Python for Android:
    - The easiest way is to download an Android release from python.org [https://www.python.org/downloads/android/]. The `prefix` directory mentioned below is at the top level of the package.
    - Or if you want to build it yourself, follow the instructions in Android/README.md [https://github.com/python/cpython/tree/3.14/Android/README.md]. The `prefix` directory will be created under `cross-build/HOST`.
- Add code to your build.gradle [https://github.com/python/cpython/tree/3.14/Android/testbed/app/build.gradle.kts] file to copy the following items into your project. All except your own Python code can be copied from `prefix/lib`:
    - In your JNI libraries:
        - `libpython*.*.so`
        - `lib*_python.so` (external libraries such as OpenSSL)
    - In your assets:
        - `python*.*` (the Python standard library)
        - `python*.*/site-packages` (your own Python code)
- Add code to your app to extract the assets to the filesystem [https://github.com/python/cpython/tree/3.14/Android/testbed/app/src/main/java/org/python/testbed/MainActivity.kt].
- Add code to your app to start Python in embedded mode [https://github.com/python/cpython/tree/3.14/Android/testbed/app/src/main/c/main_activity.c]. This will need to be C code called via JNI.

6.2. Building a Python package for Android

Python packages can be built for Android as wheels and released on PyPI. The recommended tool for doing this is cibuildwheel [https://cibuildwheel.pypa.io/en/stable/platforms/#android], which automates all the details of setting up a cross-compilation environment, building the wheel, and testing it on an emulator.

7. Using Python on iOS

Authors:

Russell Keith-Magee (2024-03)

Python on iOS is unlike Python on desktop platforms. On a desktop platform, Python is generally installed as a system resource that can be used by any user of that computer. Users then interact with Python by running a **python** executable and entering commands at an interactive prompt, or by running a Python script.

On iOS, there is no concept of installing as a system resource. The only unit of software distribution is an «app». There is also no console where you could run a **python** executable, or interact with a Python REPL.

As a result, the only way you can use Python on iOS is in embedded mode - that is, by writing a native iOS application, and embedding a Python interpreter using `libPython`, and invoking Python code using the Python embedding API. The full Python interpreter, the standard library, and all your Python code is then packaged as a standalone bundle that can be distributed via the iOS App Store.

If you're looking to experiment for the first time with writing an iOS app in Python, projects such as BeeWare [https://beeware.org] and Kivy [https://kivy.org] will provide a much more approachable user experience. These projects manage the complexities associated with getting an iOS project running, so you only need to deal with the Python code itself.

7.1. Python at runtime on iOS

7.1.1. iOS version compatibility

The minimum supported iOS version is specified at compile time, using the **`--host`** option to `configure`. By default, when compiled for iOS, Python will be compiled with a minimum supported iOS version of 13.0. To use a different minimum iOS version, provide the version number as part of the **`--host`** argument - for example, `--host=arm64-apple-ios15.4-simulator` would compile an ARM64 simulator build with a deployment target of 15.4.

7.1.2. Platform identification

When executing on iOS, `sys.platform` will report as `ios`. This value will be returned on an iPhone or iPad, regardless of whether the app is running on the simulator or a physical device.

Information about the specific runtime environment, including the iOS version, device model, and whether the device is a simulator, can be obtained using **`platform.ios_ver()`**. **`platform.system()`** will report `iOS` or `iPadOS`, depending on the device.

`os.uname()` reports kernel-level details; it will report a name of `Darwin`.

7.1.3. Standard library availability

The Python standard library has some notable omissions and restrictions on iOS. See the API availability guide for iOS for details.

7.1.4. Binary extension modules

One notable difference about iOS as a platform is that App Store distribution imposes hard requirements on the packaging of an application. One of these requirements governs how binary extension modules are distributed.

The iOS App Store requires that *all* binary modules in an iOS app must be dynamic libraries, contained in a framework with appropriate metadata, stored in the `Frameworks` folder of the packaged app. There can be only a single binary per framework, and there can be no executable binary material outside the `Frameworks` folder.

This conflicts with the usual Python approach for distributing binaries, which allows a binary extension module to be loaded from any location on `sys.path`. To ensure compliance with App Store policies, an iOS project must post-process any Python packages, converting `.so` binary modules into individual standalone frameworks with appropriate metadata and signing. For details on how to perform this post-processing, see the guide for adding Python to your project.

To help Python discover binaries in their new location, the original `.so` file on `sys.path` is replaced with a `.fwork` file. This file is a text file containing the location of the framework binary, relative to the app bundle. To allow the framework to resolve back to the original location, the framework must contain a `.origin` file that contains the location of the `.fwork` file, relative to the app bundle.

For example, consider the case of an import `from foo.bar import _whiz`, where `_whiz` is implemented with the binary module `sources/foo/bar/_whiz.abi3.so`, with `sources` being the location registered on `sys.path`, relative to the application bundle. This module *must* be distributed as `Frameworks/foo.bar._whiz.framework/foo.bar._whiz` (creating the framework name from the full import path of the module), with an `Info.plist` file in the `.framework` directory identifying the binary as a framework. The `foo.bar._whiz` module would be represented in the original location with a `sources/foo/bar/_whiz.abi3.fwork` marker file, containing the path `Frameworks/foo.bar._whiz/foo.bar._whiz`. The framework would also contain `Frameworks/foo.bar._whiz.framework/foo.bar._whiz.origin`, containing the path to the `.fwork` file.

When running on iOS, the Python interpreter will install an **`AppleFrameworkLoader`** that is able to read and import `.fwork` files. Once imported, the `__file__` attribute of the binary module will report as the location of the `.fwork` file. However, the **`ModuleSpec`** for the loaded module will report the `origin` as the location of the binary in the framework folder.

7.1.5. Compiler stub binaries

Xcode doesn't expose explicit compilers for iOS; instead, it uses an `xcrun` script that resolves to a full compiler path (e.g., `xcrun --sdk iphoneos clang` to get the `clang` for an iPhone device). However, using this script poses two problems:

- The output of `xcrun` includes paths that are machine specific, resulting in a sysconfig module that cannot be shared between users; and
- It results in `CC/CPP/LD/AR` definitions that include spaces. There is a lot of C ecosystem tooling that assumes that you can split a command line at the first space to get the path to the compiler executable; this isn't the case when using `xcrun`.

To avoid these problems, Python provided stubs for these tools. These stubs are shell script wrappers around the underingly `xcrun` tools, distributed in a `bin` folder distributed alongside the compiled iOS framework. These scripts are relocatable, and will always resolve to the appropriate local system paths. By including these scripts in the bin folder that accompanies a framework, the contents of the `sysconfig` module becomes useful for end-users to compile their own modules. When compiling third-party Python modules for iOS, you should ensure these stub binaries are on your path.

7.2. Installing Python on iOS

7.2.1. Tools for building iOS apps

Building for iOS requires the use of Apple's Xcode tooling. It is strongly recommended that you use the most recent stable release of Xcode. This will require the use of the most (or second-most) recently released macOS version, as Apple does not maintain Xcode for older macOS versions. The Xcode Command Line Tools are not sufficient for iOS development; you need a *full* Xcode install.

If you want to run your code on the iOS simulator, you'll also need to install an iOS Simulator Platform. You should be prompted to select an iOS Simulator Platform when you first run Xcode. Alternatively, you can add an iOS Simulator Platform by selecting from the Platforms tab of the Xcode Settings panel.

7.2.2. Adding Python to an iOS project

Python can be added to any iOS project, using either Swift or Objective C. The following examples will use Objective C; if you are using Swift, you may find a library like PythonKit [https://github.com/pvieito/PythonKit] to be helpful.

To add Python to an iOS Xcode project:

1. Build or obtain a Python `XCFramework`. See the instructions in Apple/iOS/README.md [https://github.com/python/cpython/tree/3.14/Apple/iOS/README.md] (in the CPython source distribution) for details on how to build a Python `XCFramework`. At a minimum, you will need a build that supports `arm64-apple-ios`, plus one of either `arm64-apple-ios-simulator` or `x86_64-apple-ios-simulator`.

2. Drag the `XCframework` into your iOS project. In the following instructions, we'll assume you've dropped the `XCframework` into the root of your project; however, you can use any other location that you want by adjusting paths as needed.

3. Add your application code as a folder in your Xcode project. In the following instructions, we'll assume that your user code is in a folder named `app` in the root of your project; you can use any other location by adjusting paths as needed. Ensure that this folder is associated with your app target.

4. Select the app target by selecting the root node of your Xcode project, then the target name in the sidebar that appears.

5. In the «General» settings, under «Frameworks, Libraries and Embedded Content», add `Python.xcframework`, with «Embed & Sign» selected.

6. In the «Build Settings» tab, modify the following:

- Build Options
    - User Script Sandboxing: No
    - Enable Testability: Yes
- Search Paths
    - Framework Search Paths: `$(PROJECT_DIR)`
    - Header Search Paths: `"$(BUILT_PRODUCTS_DIR)/Python.framework/Headers"`
- Apple Clang - Warnings - All languages
    - Quoted Include In Framework Header: No

15. Add a build step that processes the Python standard library, and your own Python binary dependencies. In the «Build Phases» tab, add a new «Run Script» build step *before* the «Embed Frameworks» step, but *after* the «Copy Bundle Resources» step. Name the step «Process Python libraries», disable the «Based on dependency analysis» checkbox, and set the script content to:

    ```
    set -e
    source $PROJECT_DIR/Python.xcframework/build/build_
    install_python Python.xcframework app
    ```

    If you have placed your XCframework somewhere other than the root of your project, modify the path to the first argument.

16. Add Objective C code to initialize and use a Python interpreter in embedded mode. You should ensure that:

    - UTF-8 mode (`PyPreConfig.utf8_mode`) is *enabled*;
    - Buffered stdio (`PyConfig.buffered_stdio`) is *disabled*;
    - Writing bytecode (`PyConfig.write_bytecode`) is *disabled*;
    - Signal handlers (`PyConfig.install_signal_handlers`) are *enabled*;
    - System logging (`PyConfig.use_system_logger`) is *enabled* (optional, but strongly recommended; this is enabled by default);

  - **`PYTHONHOME`** for the interpreter is configured to point at the `python` subfolder of your app's bundle; and
  - The **`PYTHONPATH`** for the interpreter includes:
    - the `python/lib/python3.X` subfolder of your app's bundle,
    - the `python/lib/python3.X/lib-dynload` subfolder of your app's bundle, and
    - the `app` subfolder of your app's bundle

  Your app's bundle location can be determined using `[[NSBundle mainBundle] resourcePath]`.

Steps 7 and 8 of these instructions assume that you have a single folder of pure Python application code, named `app`. If you have third-party binary modules in your app, some additional steps will be required:

- You need to ensure that any folders containing third-party binaries are either associated with the app target, or are explicitly copied as part of step 7. Step 7 should also purge any binaries that are not appropriate for the platform a specific build is targeting (i.e., delete any device binaries if you're building an app targeting the simulator).
- If you're using a separate folder for third-party packages, ensure that folder is added to the end of the call to `install_python` in step 7, and as part of the **`PYTHONPATH`** configuration in step 8.
- If any of the folders that contain third-party packages will contain `.pth` files, you should add that folder as a *site directory* (using **`site.addsitedir()`**), rather than adding to **`PYTHONPATH`** or **`sys.path`** directly.

7.2.3. Testing a Python package

The CPython source tree contains a testbed project [https://github.com/python/cpython/tree/3.14/Apple/iOS/testbed] that is used to run the CPython test suite on the iOS simulator. This testbed can also be used as a testbed project for running your Python library's test suite on iOS.

After building or obtaining an iOS XCFramework (see Apple/iOS/

README.md [https://github.com/python/cpython/tree/3.14/Apple/iOS/README.md] for details), create a clone of the Python iOS testbed project. If you used the `Apple` build script to build the XCframework, you can run:

```
$ python cross-build/iOS/testbed clone --app <path/to/mo
```

Or, if you’ve sourced your own XCframework, by running:

```
$ python Apple/testbed clone --platform iOS --framework
```

Any folders specified with the `--app` flag will be copied into the cloned testbed project. The resulting testbed will be created in the `app-testbed` folder. In this example, the `module1` and `module2` would be importable modules at runtime. If your project has additional dependencies, they can be installed into the `app-testbed/Testbed/app_packages` folder (using `pip install --target app-testbed/Testbed/app_packages` or similar).

You can then use the `app-testbed` folder to run the test suite for your app, For example, if `module1.tests` was the entry point to your test suite, you could run:

```
$ python app-testbed run -- module1.tests
```

This is the equivalent of running `python -m module1.tests` on a desktop Python build. Any arguments after the `--` will be passed to the testbed as if they were arguments to `python -m` on a desktop machine.

You can also open the testbed project in Xcode by running:

```
$ open app-testbed/iOSTestbed.xcodeproj
```

This will allow you to use the full Xcode suite of tools for debugging.

The arguments used to run the test suite are defined as part of the test plan. To modify the test plan, select the test plan node of the project tree (it should be the first child of the root node), and select the «Configurations» tab. Modify the «Arguments Passed On Launch» value to change the testing arguments.

The test plan also disables parallel testing, and specifies the use of the `Testbed.lldbinit` file for providing configuration of the debugger. The default debugger configuration disables automatic breakpoints on the `SIGINT`, `SIGUSR1`, `SIGUSR2`, and `SIGXFSZ` signals.

7.3. App Store Compliance

The only mechanism for distributing apps to third-party iOS devices is to submit the app to the iOS App Store; apps submitted for distribution must pass Apple's app review process. This process includes a set of automated validation rules that inspect the submitted application bundle for problematic code. There are some steps that must be taken to ensure that your app will be able to pass these validation steps.

7.3.1. Incompatible code in the standard library

The Python standard library contains some code that is known to violate these automated rules. While these violations appear to be false positives, Apple's review rules cannot be challenged; so, it is necessary to modify the Python standard library for an app to pass App Store review.

The Python source tree contains a patch file [https://github.com/python/cpython/tree/3.14/Mac/Resources/app-store-compliance.patch] that will remove all code that is known to cause issues with the App Store review process. This patch is applied automatically when building for iOS.

7.3.2. Privacy manifests

In April 2025, Apple introduced a requirement for certain third-party libraries to provide a Privacy Manifest [https://developer.apple.com/support/third-party-SDK-requirements]. As a result, if you have a binary module that uses one of the affected libraries, you must provide an `.xcprivacy` file for that library. OpenSSL is one library affected by this requirement, but there are others.

If you produce a binary module named `mymodule.so`, and use you the Xcode build script described in step 7 above, you can place a `mymodule.xcprivacy` file next to `mymodule.so`, and the privacy manifest will be installed into the required location when the binary module is converted into a framework.

8. Editores e IDEs

Existen numerosos IDEs que admiten el lenguaje de programación Python. Muchos editores e IDEs proporcionan resaltado de sintaxis, herramientas de depuración y comprobaciones de **PEP 8** [https://peps.python.org/pep-0008/].

8.1. IDLE — Python editor and shell

IDLE is Python's Integrated Development and Learning Environment and is generally bundled with Python installs. If you are on Linux and do not have IDLE installed see Installing IDLE on Linux. For more information see the IDLE docs.

8.2. Other Editors and IDEs

Python's community wiki has information submitted by the community on Editors and IDEs. Please go to Python Editors [https://wiki.python.org/moin/PythonEditors] and Integrated Development Environments [https://wiki.python.org/moin/IntegratedDevelopmentEnvironments] for a comprehensive list.

Referencia del Lenguaje Python

Este manual de referencia describe la sintaxis y la «semántica base» del lenguaje. Es conciso, pero intenta ser exacto y completo. La semántica de los tipos de objetos integrados no esenciales y de las funciones y módulos integrados están descritos en La biblioteca estándar de Python. Para obtener una introducción informal al lenguaje, consulte El tutorial de Python. Para programadores C o C + +, existen dos manuales adicionales: Ampliación e incrustación del intérprete de Python describe detalladamente cómo escribir un módulo de extensión de Python, y Python/C API reference manual describe en detalle las interfaces disponibles para los programadores C/C+ +.

- 1. Introducción
    - 1.1. Implementaciones alternativas
    - 1.2. Notación
- 2. Análisis léxico
    - 2.1. Estructura de línea
    - 2.2. Otros tokens
    - 2.3. Names (identifiers and keywords)
    - 2.4. Literales
    - 2.5. Literales de cadenas y bytes
    - 2.6. Literales numéricos
    - 2.7. Operators and delimiters
- 3. Modelo de datos
    - 3.1. Objetos, valores y tipos
    - 3.2. Jerarquía de tipos estándar
    - 3.3. Nombres especiales de método
    - 3.4. Corrutinas
- 4. Modelo de ejecución

  - 4.1. Estructura de un programa
  - 4.2. Nombres y vínculos
  - 4.3. Excepciones
  - 4.4. Runtime Components

- 5. El sistema de importación

  - 5.1. `importlib`
  - 5.2. Paquetes
  - 5.3. Buscando
  - 5.4. Cargando
  - 5.5. El buscador basado en rutas
  - 5.6. Reemplazando el sistema de importación estándar
  - 5.7. Paquete Importaciones relativas
  - 5.8. Consideraciones especiales para __main__
  - 5.9. Referencias

- 6. Expresiones

  - 6.1. Conversiones aritméticas
  - 6.2. Átomos
  - 6.3. Primarios
  - 6.4. Expresión await
  - 6.5. El operador de potencia
  - 6.6. Aritmética unaria y operaciones bit a bit
  - 6.7. Operaciones aritméticas binarias
  - 6.8. Operaciones de desplazamiento
  - 6.9. Operaciones bit a bit binarias
  - 6.10. Comparaciones
  - 6.11. Operaciones booleanas
  - 6.12. Expresiones de asignación
  - 6.13. Expresiones condicionales
  - 6.14. Lambdas
  - 6.15. Listas de expresiones
  - 6.16. Orden de evaluación
  - 6.17. Prioridad de operador

- 7. Declaraciones simples

  - 7.1. Declaraciones de tipo expresión
  - 7.2. Declaraciones de asignación

  - 7.3. La declaración `assert`
  - 7.4. La declaración `pass`
  - 7.5. La declaración `del`
  - 7.6. La declaración `return`
  - 7.7. La declaración `yield`
  - 7.8. La declaración `raise`
  - 7.9. La declaración `break`
  - 7.10. La declaración `continue`
  - 7.11. La declaración `import`
  - 7.12. La declaración `global`
  - 7.13. La declaración `nonlocal`
  - 7.14. The `type` statement

- 8. Sentencias compuestas

  - 8.1. La sentencia `if`
  - 8.2. La sentencia `while`
  - 8.3. La sentencia `for`
  - 8.4. La sentencia `try`
  - 8.5. La sentencia `with`
  - 8.6. La sentencia `match`
  - 8.7. Definiciones de funciones
  - 8.8. Definiciones de clase
  - 8.9. Corrutinas
  - 8.10. Listas de tipo parámetro
  - 8.11. Annotations

- 9. Componentes de nivel superior

  - 9.1. Programas completos de Python
  - 9.2. Entrada de archivo
  - 9.3. Entrada interactiva
  - 9.4. Entrada de expresión

- 10. Especificación completa de la gramática

1. Introducción

Este manual de referencia describe el lenguaje de programación Python. No pretende ser un tutorial.

Aunque intento ser lo más preciso posible, he optado por utilizar el Español (N. de T.: del original en inglés "I chose to use English")» en lugar de especificaciones formales para todo, excepto para la sintaxis y el análisis léxico. Esto debería hacer el documento más comprensible para el lector medio, pero dejará espacio para ambigüedades. En consecuencia, si vienes de Marte y tratas de re implementar Python sólo a partir de este documento, puede que tengas que adivinar cosas y, de hecho, probablemente acabarías implementando un lenguaje bastante diferente. Por otro lado, si estás usando Python y te preguntas cuáles son las reglas precisas sobre un área particular del lenguaje, deberías poder encontrarlas aquí. Si te gustaría ver una definición más formal del lenguaje, quizás podrías ofrecer tu tiempo — o inventar una máquina clonadora :-).

Es peligroso añadir muchos detalles de implementación en un documento de referencia: la implementación puede cambiar y otras implementaciones del lenguaje pueden funcionar de forma diferente. Por otro lado, CPython es la implementación de Python más usada (aunque implementaciones alternativas están ganando soporte), y es importante mencionar sus detalles particulares especialmente donde la implementación impone limitaciones adicionales. Por lo tanto, encontrarás pequeñas «notas sobre la implementación» repartidas por todo el texto.

Cada implementación de Python viene con un número de módulos estándar incorporados. Éstos están documentados en La biblioteca estándar de Python. Unos pocos de estos módulos son citados cuando interactúan de forma significativa con la definición del lenguaje.

1.1. Implementaciones alternativas

Aunque hay una implementación de Python que es, de lejos, la más popular, hay otras implementaciones alternativas que pueden ser de particular interés para diferentes audiencias.

Las implementaciones conocidas incluyen:

CPython

> Es la implementación original, y la más mantenida, de Python y está escrita en C. Las nuevas características del lenguaje normalmente aparecen primero aquí.

Jython

> Python implementado en Java. Esta implementación puede utilizarse como lenguaje de scripting para aplicaciones Java, o puede utilizarse para crear aplicaciones utilizando las bibliotecas de clases Java. También se utiliza a menudo para crear pruebas para bibliotecas Java. Puede encontrar más información en the Jython website [https://www.jython.org/].

Python para .NET

> Esta implementación, de hecho, usa la implementación CPython, pero es una aplicación .NET gestionada y usa librerías .NET. Ha sido creada por Brian Lloyd. Para más información ir al sitio web de Python for .NET [https://pythonnet.github.io/].

IronPython

> Una alternativa de Python para .NET. A diferencia de Python.NET, esta es una implementación completa de Python que genera IL, y compila código Python directamente a ensamblados .NET. Fue creado por Jim Hugunin, el creador original de Jython. Para más información, consulte el sitio web de IronPython [https://ironpython.net/].

PyPy

> Una implementación de Python escrita completamente en Python. Soporta varias características avanzadas que no se encuentran en otras implementaciones como compatibilidad

stackless y un compilador Just in Time. Uno de los objetivos del proyecto es fomentar la experimentación con el propio lenguaje facilitando la modificación del intérprete (ya que está escrito en Python). Puede encontrar más información en la página principal del proyecto "PyPy <https://www.pypy.org/>_".

Cada una de estas implementaciones varía de una forma u otra del lenguaje tal y como está documentado en este manual, o introduce información específica más allá de lo cubierto por la documentación estándar de Python. Por favor, consulte la documentación específica de cada implementación para saber qué tienes que saber acerca de la implementación específica que uses.

1.2. Notación

The descriptions of lexical analysis and syntax use a grammar notation that is a mixture of EBNF [https://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_form] and PEG [https://en.wikipedia.org/wiki/Parsing_expression_grammar]. For example:

```
name:   letter (letter | digit | "_")*
letter: "a"..."z" | "A"..."Z"
digit:  "0"..."9"
```

In this example, the first line says that a `name` is a `letter` followed by a sequence of zero or more `letter`s, `digit`s, and underscores. A `letter` in turn is any of the single characters `'a'` through `'z'` and `A` through `Z`; a `digit` is a single character from `0` to `9`.

Each rule begins with a name (which identifies the rule that's being defined) followed by a colon, `:`. The definition to the right of the colon uses the following syntax elements:

- `name`: A name refers to another rule. Where possible, it is a link to the rule's definition.
    - `TOKEN`: An uppercase name refers to a token. For the purposes of grammar definitions, tokens are the same

as rules.

- `"text"`, `'text'`: Text in single or double quotes must match literally (without the quotes). The type of quote is chosen according to the meaning of `text`:
    - `'if'`: A name in single quotes denotes a keyword.
    - `"case"`: A name in double quotes denotes a soft-keyword.
    - `'@'`: A non-letter symbol in single quotes denotes an **`OP`** token, that is, a delimiter or operator.
- `e1 e2`: Items separated only by whitespace denote a sequence. Here, `e1` must be followed by `e2`.
- `e1 | e2`: A vertical bar is used to separate alternatives. It denotes PEG's «ordered choice»: if `e1` matches, `e2` is not considered. In traditional PEG grammars, this is written as a slash, `/`, rather than a vertical bar. See **PEP 617** [https://peps.python.org/pep-0617/] for more background and details.
- `e*`: A star means zero or more repetitions of the preceding item.
- `e+`: Likewise, a plus means one or more repetitions.
- `[e]`: A phrase enclosed in square brackets means zero or one occurrences. In other words, the enclosed phrase is optional.
- `e?`: A question mark has exactly the same meaning as square brackets: the preceding item is optional.
- `(e)`: Parentheses are used for grouping.

The following notation is only used in lexical definitions.

- `"a"..."z"`: Two literal characters separated by three dots mean a choice of any single character in the given (inclusive) range of ASCII characters.
- `<...>`: A phrase between angular brackets gives an informal description of the matched symbol (for example, `<any ASCII character except "\">`), or an abbreviation that is defined in nearby text (for example, `<Lu>`).

Some definitions also use *lookaheads*, which indicate that an element must (or must not) match at a given position, but without consuming any input:

- `&e`: a positive lookahead (that is, `e` is required to match)
- `!e`: a negative lookahead (that is, `e` is required *not* to match)

The unary operators (`*`, `+`, `?`) bind as tightly as possible; the vertical bar (`|`) binds most loosely.

White space is only meaningful to separate tokens.

Rules are normally contained on a single line, but rules that are too long may be wrapped:

```
literal: stringliteral | bytesliteral
         | integer | floatnumber | imagnumber
```

Alternatively, rules may be formatted with the first line ending at the colon, and each alternative beginning with a vertical bar on a new line. For example:

```
literal:
   | stringliteral
   | bytesliteral
   | integer
   | floatnumber
   | imagnumber
```

This does *not* mean that there is an empty first alternative.

1.2.1. Lexical and Syntactic definitions

There is some difference between *lexical* and *syntactic* analysis: the lexical analyzer operates on the individual characters of the input source, while the *parser* (syntactic analyzer) operates on the stream of tokens generated by the lexical analysis. However, in some cases the exact boundary between the two phases is a CPython implementation detail.

The practical difference between the two is that in *lexical* definitions, all whitespace is significant. The lexical analyzer discards all whitespace that is not converted to tokens like **`token.INDENT`** or **`NEWLINE`**. *Syntactic* definitions then use these tokens, rather than source characters.

This documentation uses the same BNF grammar for both styles of definitions. All uses of BNF in the next chapter (Análisis léxico) are lexical definitions; uses in subsequent chapters are syntactic definitions.

2. Análisis léxico

A Python program is read by a *parser*. Input to the parser is a stream of tokens, generated by the *lexical analyzer* (also known as the *tokenizer*). This chapter describes how the lexical analyzer produces these tokens.

The lexical analyzer determines the program text's encoding (UTF-8 by default), and decodes the text into source characters. If the text cannot be decoded, a **`SyntaxError`** is raised.

Next, the lexical analyzer uses the source characters to generate a stream of tokens. The type of a generated token generally depends on the next source character to be processed. Similarly, other special behavior of the analyzer depends on the first source character that hasn't yet been processed. The following table gives a quick summary of these source characters, with links to sections that contain more information.

Character	Next token (or other relevant documentation)
• espacio • tabulador • formfeed	• Whitespace
• CR, LF	• New line • Indentation
• backslash (`\`)	• Explicit line joining • (Also significant in string escape sequences)
• hash (`#`)	• Comment
• quote (`'`, `"`)	• String literal
• ASCII letter (`a-z`, `A-Z`) • non-ASCII character	• Name • Prefixed string or bytes literal
• underscore (`_`)	• Name • (Can also be part of numeric literals)
• number (`0-9`)	• Numeric literal
• dot (`.`)	• Numeric literal • Operator

- question mark (?)
- dollar (`$`)
- backquote (`` ` ``)
- control character
- other printing character
- end of file

Error (outside string literals and comments)

Operator or delimiter

End marker

2.1. Estructura de línea

Un programa Python se divide en un número de *líneas lógicas*.

2.1.1. Líneas lógicas

The end of a logical line is represented by the token **`NEWLINE`**. Statements cannot cross logical line boundaries except where **`NEWLINE`** is allowed by the syntax (e.g., between statements in compound statements). A logical line is constructed from one or more *physical lines* by following the explicit or implicit *line joining* rules.

2.1.2. Líneas físicas

A physical line is a sequence of characters terminated by one the following end-of-line sequences:

- the Unix form using ASCII LF (linefeed),
- the Windows form using the ASCII sequence CR LF (return followed by linefeed),
- the “Classic Mac OS [https://en.wikipedia.org/wiki/Classic_Mac_OS]” form using the ASCII CR (return) character.

Regardless of platform, each of these sequences is replaced by a single ASCII LF (linefeed) character. (This is done even inside string literals.) Each line can use any of the sequences; they do not need to be consistent within a file.

The end of input also serves as an implicit terminator for the final physical line.

Formally:

newline: `<ASCII LF> | <ASCII CR> <ASCII LF> | <ASCII CR`

2.1.3. Comentarios

Un comentario comienza con un carácter de almohadilla (`#`) que no es parte de un literal de cadena, y termina al final de la línea física. Un comentario implica el final de la línea lógica, a menos que se invoque la regla implícita de unión de líneas. Los comentarios son ignorados por la sintaxis.

2.1.4. Declaración de Codificación

Si un comentario en la primera o segunda línea del script de Python coincide con la expresión regular `coding[=:]\s*([-\w.]+)`, este comentario se procesa como una declaración de codificación; el primer grupo de esta expresión denomina la codificación del archivo de código fuente. La declaración de codificación debe aparecer en una línea propia. Si se trata de la segunda línea, la primera línea debe ser también una línea solamente de comentario. Las formas recomendadas de una expresión de codificación son

```
# -*- coding: <encoding-name> -*-
```

que también es reconocido por GNU Emacs y

```
# vim:fileencoding=<encoding-name>
```

que es reconocido por el VIM de Bram Moolenaar.

If no encoding declaration is found, the default encoding is UTF-8. If the implicit or explicit encoding of a file is UTF-8, an initial UTF-8 byte-order mark (`b'\xef\xbb\xbf'`) is ignored rather than being a syntax error.

Si se declara una codificación, Python debe reconocer el nombre de la codificación (ver Codificaciones estándar). La codificación se utiliza para todos los análisis léxicos, incluidos las cadenas literales, los comentarios y los identificadores.

All lexical analysis, including string literals, comments and identifiers, works on Unicode text decoded using the source encoding. Any Unicode code point, except the NUL control character, can appear in Python source.

```
source_character:  <any Unicode code point, except NUL>
```

2.1.5. Unión explícita de líneas

Dos o más líneas físicas pueden unirse en líneas lógicas utilizando caracteres de barra invertida (\), de la siguiente manera: cuando una línea física termina en una barra invertida que no es parte de literal de cadena o de un comentario, se une con la siguiente formando una sola línea lógica, borrando la barra invertida y el siguiente carácter de fin de línea. Por ejemplo:

```
if 1900 < year < 2100 and 1 <= month <= 12 \
   and 1 <= day <= 31 and 0 <= hour < 24 \
   and 0 <= minute < 60 and 0 <= second < 60:   # Looks
        return 1
```

Una línea que termina en una barra invertida no puede llevar un comentario. Una barra invertida no continúa un comentario. Una barra invertida no continúa un token excepto para los literales de la cadena (es decir, los tokens que no sean literales de la cadena no pueden ser divididos a través de líneas físicas usando una barra invertida). La barra invertida es ilegal en cualquier parte de una línea fuera del literal de la cadena.

2.1.6. Unión implícita de líneas

Las expresiones entre paréntesis, entre corchetes o entre rizos pueden dividirse en más de una línea física sin usar barras invertidas. Por ejemplo:

```
month_names = ['Januari', 'Februari', 'Maart',      # Sc
               'April',   'Mei',      'Juni',       # no
               'Juli',    'Augustus', 'September',  # pa
               'Oktober', 'November', 'December']   # de
```

Las líneas continuas implícitas pueden llevar comentarios. La sangría de las líneas de continuación no es importante. Se permiten líneas de continuación en blanco. No hay ningún token NEWLINE (nueva línea) entre las líneas de continuación implícitas. Las líneas de continuación implícitas también pueden aparecer dentro de cadenas de triple comilla (ver más adelante); en ese caso no pueden llevar comentarios.

2.1.7. Líneas en blanco

A logical line that contains only spaces, tabs, formfeeds and possibly a comment, is ignored (i.e., no `NEWLINE` token is generated). During interactive input of statements, handling of a blank line may differ depending on the implementation of the read-eval-print loop. In the standard interactive interpreter, an entirely blank logical line (that is, one containing not even whitespace or a comment) terminates a multi-line statement.

2.1.8. Sangría

El espacio en blanco (espacios y tabulaciones) al principio de una línea lógica se utiliza para calcular el nivel de sangría de la línea, que a su vez se utiliza para determinar la agrupación de las declaraciones.

Los tabuladores se sustituyen (de izquierda a derecha) por uno a ocho espacios, de manera que el número total de caracteres hasta el reemplazo inclusive es un múltiplo de ocho (se pretende que sea la misma regla que la utilizada por Unix). El número total de espacios que preceden al primer carácter no en blanco determina entonces la sangría de la línea. La sangría no puede dividirse en múltiples líneas físicas utilizando barras invertidas; el espacio en blanco hasta la primera barra invertida determina la sangría.

La indentación se rechaza como inconsistente si un archivo fuente mezcla tabulaciones y espacios de manera que el significado depende del valor de una tabulación en los espacios; un `TabError` se produce en ese caso.

Nota de compatibilidad entre plataformas: debido a la naturaleza de los editores de texto en plataformas que no sean UNIX, no es aconsejable utilizar una mezcla de espacios y tabuladores para la sangría en un solo archivo de origen. También debe tenerse en cuenta que las diferentes plataformas pueden limitar explícitamente el nivel máximo de sangría.

Un carácter *formfeed* puede estar presente al comienzo de la línea; será ignorado para los cálculos de sangría anteriores. Los caracteres *formfeed* que aparecen en otras partes del espacio en blanco inicial tienen un efecto indefinido (por ejemplo, pueden poner a cero el recuento de espacio).

The indentation levels of consecutive lines are used to generate **`INDENT`** and **`DEDENT`** tokens, using a stack, as follows.

Before the first line of the file is read, a single zero is pushed on the stack; this will never be popped off again. The numbers pushed on the stack will always be strictly increasing from bottom to top. At the beginning of each logical line, the line's indentation level is compared to the top of the stack. If it is equal, nothing happens. If it is larger, it is pushed on the stack, and one **`INDENT`** token is generated. If it is smaller, it *must* be one of the numbers occurring on the stack; all numbers on the stack that are larger are popped off, and for each number popped off a **`DEDENT`** token is generated. At the end of the file, a **`DEDENT`** token is generated for each number remaining on the stack that is larger than zero.

Aquí hay un ejemplo de un código de Python con una correcta (aunque no tan clara) sangría:

```
def perm(l):
        # Calcular la lista de todas las permutaciones c
    if len(l) <= 1:
                  return [l]
    r = []
    for i in range(len(l)):
             s = l[:i] + l[i+1:]
             p = perm(s)
             for x in p:
              r.append(l[i:i+1] + x)
    return r
```

El siguiente ejemplo muestra varios errores de sangría:

```
 def perm(l):                       # error: first line
for i in range(len(l)):             # error: not indente
    s = l[:i] + l[i+1:]
        p = perm(l[:i] + l[i+1:])   # error: unexpected
        for x in p:
                r.append(l[i:i+1] + x)
            return r                # error: inconsister
```

(En realidad, los tres primeros errores son detectados por el analizador; sólo el último error es encontrado por el analizador léxico — la sangría de `return r` no coincide con un nivel sacado de la pila.)

2.1.9. Espacios en blanco entre tokens

Except at the beginning of a logical line or in string literals, the whitespace characters space, tab and formfeed can be used interchangeably to separate tokens:

```
whitespace:  ' ' | tab | formfeed
```

Whitespace is needed between two tokens only if their concatenation could otherwise be interpreted as a different token. For example, `ab` is one token, but `a b` is two tokens. However, `+a` and `+ a` both produce two tokens, `+` and `a`, as `+a` is not a valid token.

2.1.10. End marker

At the end of non-interactive input, the lexical analyzer generates an **`ENDMARKER`** token.

2.2. Otros tokens

Besides **`NEWLINE`**, **`INDENT`** and **`DEDENT`**, the following categories

of tokens exist: *identifiers* and *keywords* (`NAME`), *literals* (such as `NUMBER` and `STRING`), and other symbols (*operators* and *delimiters*, `OP`). Whitespace characters (other than logical line terminators, discussed earlier) are not tokens, but serve to delimit tokens. Where ambiguity exists, a token comprises the longest possible string that forms a legal token, when read from left to right.

2.3. Names (identifiers and keywords)

`NAME` tokens represent *identifiers*, *keywords*, and *soft keywords*.

Names are composed of the following characters:

- uppercase and lowercase letters (`A-Z` and `a-z`),
- the underscore (`_`),
- digits (`0` through `9`), which cannot appear as the first character, and
- non-ASCII characters. Valid names may only contain «letter-like» and «digit-like» characters; see Non-ASCII characters in names for details.

Names must contain at least one character, but have no upper length limit. Case is significant.

Formally, names are described by the following lexical definitions:

```
NAME:          name_start name_continue*
name_start:    "a"..."z" | "A"..."Z" | "_" | <non-ASCII
name_continue: name_start | "0"..."9"
identifier:    <NAME, except keywords>
```

Note that not all names matched by this grammar are valid; see Non-ASCII characters in names for details.

2.3.1. Palabras clave

The following names are used as reserved words, or *keywords* of the language, and cannot be used as ordinary identifiers. They must be spelled exactly as written here:

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

2.3.2. Palabras clave suaves

Added in version 3.10.

Some names are only reserved under specific contexts. These are known as *soft keywords*:

- `match`, `case`, and _, when used in the **`match`** statement.
- `type`, when used in the **`type`** statement.

These syntactically act as keywords in their specific contexts, but this distinction is done at the parser level, not when tokenizing.

Como palabras clave suaves, su uso en la gramática es posible sin dejar de preservar la compatibilidad con el código existente que usa esos nombres como nombres de identificadores.

Distinto en la versión 3.12: `type` ahora es un palabra clave suave.

2.3.3. Clases reservadas de identificadores

Ciertas clases de identificadores (además de las palabras clave) tienen significados especiales. Estas clases se identifican por los patrones de los caracteres de guión bajo que van delante y detrás:

_*

No importado por `from module import *`.

_

En un patrón `case` dentro de una declaración **`match`**, _ es una palabra clave suave que denota un comodín wildcard.

Por separado, el intérprete interactivo pone a disposición el resultado de la última evaluación en la variable `_`. (Se almacena en el módulo `builtins`, junto con funciones incorporadas como `print`).

En otros lugares, `_` es un identificador regular. A menudo se usa para nombrar elementos «especiales», pero no es especial para Python en sí.

Nota

El nombre `_` se usa a menudo en conjunción con la internacionalización; consultar la documentación del módulo **`gettext`**` para más información sobre esta convención.

También se usa comúnmente para variables no utilizadas.

`__*__`

Nombres definidos por el sistema, conocidos informalmente como nombres «*dunder*». Estos nombres son definidos por el intérprete y su aplicación (incluida la biblioteca estándar). Los nombres actuales del sistema se discuten en la sección Nombres especiales de método y en otros lugares. Es probable que se definan más en futuras versiones de Python. *Cualquier* uso de nombres `__*__`, en cualquier contexto, que no siga un uso explícitamente documentado, está sujeto a que se rompa sin previo aviso.

`__*`

Nombres de clase privada. Los nombres de esta categoría, cuando se utilizan en el contexto de una definición de clase, se reescriben para utilizar una forma desfigurada que ayude a evitar conflictos de nombres entre los atributos «privados» de las clases base y derivadas. Ver la sección Identificadores (Nombres).

2.3.4. Non-ASCII characters in names

Names that contain non-ASCII characters need additional normalization and validation beyond the rules and grammar explained above. For example, `ř_1`, `蛇`, or `साँप` are valid names, but `r〰2`, `€`, or `🐍` are not.

This section explains the exact rules.

All names are converted into the normalization form [https://www.unicode.org/reports/tr15/#Norm_Forms] NFKC while parsing. This means that, for example, some typographic variants of characters are converted to their «basic» form. For example, `ﬁⁿₐˡᵢᶻₐᵗᵢᵒₙ` normalizes to `finalization`, so Python treats them as the same name:

```
>>> ﬁⁿₐˡᵢᶻₐᵗᵢᵒₙ = 3
>>> finalization
3
```

Nota

Normalization is done at the lexical level only. Run-time functions that take names as *strings* generally do not normalize their arguments. For example, the variable defined above is accessible at run time in the **`globals()`** dictionary as `globals()["finalization"]` but not `globals()["ﬁⁿₐˡᵢᶻₐᵗᵢᵒₙ"]`.

Similarly to how ASCII-only names must contain only letters, digits and the underscore, and cannot start with a digit, a valid name must start with a character in the «letter-like» set `xid_start`, and the remaining characters must be in the «letter- and digit-like» set `xid_continue`.

These sets are based on the *XID_Start* and *XID_Continue* sets as defined by the Unicode standard annex UAX-31 [https://www.unicode.org/reports/tr31/]. Python's `xid_start` additionally includes the underscore (_). Note that Python does not necessarily conform to UAX-31 [https://www.unicode.org/reports/tr31/].

A non-normative listing of characters in the *XID_Start* and

XID_Continue sets as defined by Unicode is available in the DerivedCoreProperties.txt [https://www.unicode.org/Public/16.0.0/ucd/DerivedCoreProperties.txt] file in the Unicode Character Database. For reference, the construction rules for the `xid_*` sets are given below.

The set `id_start` is defined as the union of:

- Unicode category `<Lu>` - uppercase letters (includes `A` to `Z`)
- Unicode category `<Ll>` - lowercase letters (includes `a` to `z`)
- Unicode category `<Lt>` - titlecase letters
- Unicode category `<Lm>` - modifier letters
- Unicode category `<Lo>` - other letters
- Unicode category `<Nl>` - letter numbers
- `{"_"}` - the underscore
- `<Other_ID_Start>` - an explicit set of characters in PropList.txt [https://www.unicode.org/Public/16.0.0/ucd/PropList.txt] to support backwards compatibility

The set `xid_start` then closes this set under NFKC normalization, by removing all characters whose normalization is not of the form `id_start id_continue*`.

The set `id_continue` is defined as the union of:

- `id_start` (see above)
- Unicode category `<Nd>` - decimal numbers (includes `0` to `9`)
- Unicode category `<Pc>` - connector punctuations
- Unicode category `<Mn>` - nonspacing marks
- Unicode category `<Mc>` - spacing combining marks
- `<Other_ID_Continue>` - another explicit set of characters in PropList.txt [https://www.unicode.org/Public/16.0.0/ucd/PropList.txt] to support backwards compatibility

Again, `xid_continue` closes this set under NFKC normalization.

Unicode categories use the version of the Unicode Character Database as included in the **`unicodedata`** module.

Ver también

- **PEP 3131** [https://peps.python.org/pep-3131/] – Supporting Non-ASCII Identifiers
- **PEP 672** [https://peps.python.org/pep-0672/] – Unicode-related Security Considerations for Python

2.4. Literales

Los literales son notaciones para los valores constantes de algunos tipos incorporados.

In terms of lexical analysis, Python has string, bytes and numeric literals.

Other «literals» are lexically denoted using keywords (`None`, `True`, `False`) and the special ellipsis token (`...`).

2.5. Literales de cadenas y bytes

String literals are text enclosed in single quotes (`'`) or double quotes (`"`). For example:

```
"spam"
'eggs'
```

The quote used to start the literal also terminates it, so a string literal can only contain the other quote (except with escape sequences, see below). For example:

```
'Say "Hello", please.'
"Don't do that!"
```

Except for this limitation, the choice of quote character (`'` or `"`) does not affect how the literal is parsed.

Inside a string literal, the backslash (`\`) character introduces an *escape sequence*, which has special meaning depending on the character after the backslash. For example, `\"` denotes the double quote character, and does *not* end the string:

```
>>> print("Say \"Hello\" to everyone!")
Say "Hello" to everyone!
```

See escape sequences below for a full list of such sequences, and more details.

2.5.1. Triple-quoted strings

Strings can also be enclosed in matching groups of three single or double quotes. These are generally referred to as *triple-quoted strings*:

```
"""This is a triple-quoted string."""
```

In triple-quoted literals, unescaped quotes are allowed (and are retained), except that three unescaped quotes in a row terminate the literal, if they are of the same kind (`'` or `"`) used at the start:

```
"""This string has "quotes" inside."""
```

Unescaped newlines are also allowed and retained:

```
'''This triple-quoted string
continues on the next line.'''
```

2.5.2. String prefixes

String literals can have an optional *prefix* that influences how the content of the literal is parsed, for example:

```
b"data"
f'{result=}'
```

The allowed prefixes are:

- `b`: Bytes literal
- `r`: Raw string
- `f`: Formatted string literal («f-string»)
- `t`: Template string literal («t-string»)
- `u`: No effect (allowed for backwards compatibility)

See the linked sections for details on each type.

Prefixes are case-insensitive (for example, “`B`” works the same as “`b`”). The “`r`” prefix can be combined with “`f`”, “`t`” or “`b`”, so “`fr`”, “`rf`”, “`tr`”, “`rt`”, “`br`”, and “`rb`” are also valid prefixes.

Added in version 3.3: El prefijo `'rb'` de literales de bytes raw se ha añadido como sinónimo de `'br'`.

Se reintrodujo el soporte para el legado unicode literal (`u'value'`) para simplificar el mantenimiento de las bases de código dual Python 2.x y 3.x. Ver **PEP 414** [https://peps.python.org/pep-0414/] para más información.

2.5.3. Formal grammar

String literals, except «f-strings» and «t-strings», are described by the following lexical definitions.

These definitions use negative lookaheads (`!`) to indicate that an ending quote ends the literal.

```
STRING:          [stringprefix] (stringcontent)
stringprefix:    <("r" | "u" | "b" | "br" | "rb"), case-
stringcontent:
    | "'''" ( !"'''" longstringitem)* "'''"
    | '"""' ( !'"""' longstringitem)* '"""'
    | "'" ( !"'" stringitem)* "'"
    | '"' ( !'"' stringitem)* '"'
stringitem:      stringchar | stringescapeseq
stringchar:      <any source_character, except backslash
longstringitem:  stringitem | newline
stringescapeseq: "\" <any source_character>
```

Note that as in all lexical definitions, whitespace is significant. In particular, the prefix (if any) must be immediately followed by the starting quote.

2.5.4. Secuencias de escape

Unless an “`r`” or “`R`” prefix is present, escape sequences in string and bytes literals are interpreted according to rules similar to those used by Standard C. The recognized escape sequences are:

Secuencia de escape	Significado
`\<newline>`	Ignored end of line
`\\`	Backslash
`\'`	Single quote
`\"`	Double quote
`\a`	ASCII Bell (BEL)
`\b`	ASCII Retroceso (BS)
`\f`	ASCII Formfeed (FF)
`\n`	ASCII Linefeed (LF)
`\r`	ASCII Retorno de carro (CR)
`\t`	ASCII Sangría horizontal (TAB)
`\v`	ASCII Sangría vertical (VT)
`\ooo`	Octal character
`\xhh`	Hexadecimal character
`\N{name}`	Named Unicode character
`\uxxxx`	Hexadecimal Unicode character
`\Uxxxxxxxx`	Hexadecimal Unicode character

2.5.4.1. Ignored end of line

Se puede agregar una barra invertida al final de una línea para ignorar la nueva línea:

```
>>> 'Esta cadena no incluirá \
... caracteres de barra invertida o nueva linea.'
'Esta cadena no incluirá caracteres de barra invertida
```

Se puede lograr el mismo resultado usando triple-quoted strings, o paréntesis y string literal concatenation.

2.5.4.2. Escaped characters

To include a backslash in a non-raw Python string literal, it must be doubled. The `\\` escape sequence denotes a single backslash character:

```
>>> print('C:\\Program Files')
C:\Program Files
```

Similarly, the `\'` and `\"` sequences denote the single and double quote character, respectively:

```
>>> print('\' and \"')
' and "
```

2.5.4.3. Octal character

The sequence `\ooo` denotes a *character* with the octal (base 8) value *ooo*:

```
>>> '\120'
'P'
```

Up to three octal digits (0 through 7) are accepted.

In a bytes literal, *character* means a *byte* with the given value. In a string literal, it means a Unicode character with the given value.

Distinto en la versión 3.11: Octal escapes with value larger than `0o377` (255) produce a **`DeprecationWarning`**.

Distinto en la versión 3.12: Octal escapes with value larger than `0o377` (255) produce a **`SyntaxWarning`**. In a future Python version they will raise a **`SyntaxError`**.

2.5.4.4. Hexadecimal character

The sequence `\xhh` denotes a *character* with the hex (base 16) value *hh*:

```
>>> '\x50'
'P'
```

A diferencia de C estándar, se requieren exactamente dos dígitos hexadecimales.

In a bytes literal, *character* means a *byte* with the given value. In a string literal, it means a Unicode character with the given value.

2.5.4.5. Named Unicode character

The sequence `\N{name}` denotes a Unicode character with the given *name*:

```
>>> '\N{LATIN CAPITAL LETTER P}'
'P'
>>> '\N{SNAKE}'
'🐍'
```

This sequence cannot appear in bytes literals.

Distinto en la versión 3.3: Support for name aliases [https://www.unicode.org/Public/16.0.0/ucd/NameAliases.txt] has been added.

2.5.4.6. Hexadecimal Unicode characters

These sequences `\uxxxx` and `\Uxxxxxxxx` denote the Unicode character with the given hex (base 16) value. Exactly four digits are required for `\u`; exactly eight digits are required for `\U`. The latter can encode any Unicode character.

```
>>> '\u1234'
'ሴ'
>>> '\U0001f40d'
'🐍'
```

These sequences cannot appear in bytes literals.

2.5.4.7. Unrecognized escape sequences

Unlike in Standard C, all unrecognized escape sequences are left in the string unchanged, that is, *the backslash is left in the result*:

```
>>> print('\q')
\q
>>> list('\q')
['\\', 'q']
```

Note that for bytes literals, the escape sequences only recognized in string literals (`\N...`, `\u...`, `\U...`) fall into the category of unrecognized escapes.

Distinto en la versión 3.6: Secuencias de escape no conocidas producen un **`DeprecationWarning`**.

Distinto en la versión 3.12: Unrecognized escape sequences produce a **`SyntaxWarning`**. In a future Python version they will raise a **`SyntaxError`**.

2.5.5. Bytes literals

Bytes literals are always prefixed with "`b`" or "`B`"; they produce an instance of the **`bytes`** type instead of the **`str`** type. They may only contain ASCII characters; bytes with a numeric value of 128 or greater must be expressed with escape sequences (typically Hexadecimal character or Octal character):

```
>>> b'\x89PNG\r\n\x1a\n'
b'\x89PNG\r\n\x1a\n'
>>> list(b'\x89PNG\r\n\x1a\n')
[137, 80, 78, 71, 13, 10, 26, 10]
```

Similarly, a zero byte must be expressed using an escape sequence (typically `\0` or `\x00`).

2.5.6. Raw string literals

Both string and bytes literals may optionally be prefixed with a letter "`r`" or "`R`"; such constructs are called *raw string literals* and *raw bytes literals* respectively and treat backslashes as literal characters. As a result, in raw string literals, escape sequences are not treated specially:

```
>>> r'\d{4}-\d{2}-\d{2}'
'\\d{4}-\\d{2}-\\d{2}'
```

Incluso en un literal *raw*, las comillas se pueden escapar con una barra inversa, pero la barra inversa permanece en el resultado; por ejemplo, `r"\""` es un literal de cadena válido que consiste en dos caracteres: una barra inversa y una comilla doble; `r"\"` no es un literal de cadena válido (incluso una cadena en bruto no puede terminar en un número impar de barras inversas). Específicamente, *un literal raw no puede terminar en una sola barra inversa* (ya que la barra inversa se escaparía del siguiente carácter de comillas). Nótese también que una sola barra inversa seguida de una nueva línea se interpreta como esos dos caracteres como parte del literal, *no* como una continuación de línea.

2.5.7. f-strings

Added in version 3.6.

Distinto en la versión 3.7: The **`await`** and **`async for`** can be used in expressions within f-strings.

Distinto en la versión 3.8: Added the debug specifier (=)

Distinto en la versión 3.12: Many restrictions on expressions within f-strings have been removed. Notably, nested strings, comments, and backslashes are now permitted.

A *formatted string literal* or *f-string* is a string literal that is prefixed with “`f`” or “`F`”. Unlike other string literals, f-strings do not have a constant value. They may contain *replacement fields* delimited by curly braces `{}`. Replacement fields contain expressions which are evaluated at run time. For example:

```
>>> who = 'nobody'
>>> nationality = 'Spanish'
>>> f'{who.title()} expects the {nationality} Inquisitio
'Nobody expects the Spanish Inquisition!'
```

Any doubled curly braces (`{{` or `}}`) outside replacement fields are replaced with the corresponding single curly brace:

```
>>> print(f'{{...}}')
{...}
```

Other characters outside replacement fields are treated like in ordinary string literals. This means that escape sequences are decoded (except when a literal is also marked as a raw string), and newlines are possible in triple-quoted f-strings:

```
>>> name = 'Galahad'
>>> favorite_color = 'blue'
>>> print(f'{name}:\t{favorite_color}')
Galahad:       blue
>>> print(rf"C:\Users\{name}")
C:\Users\Galahad
>>> print(f'''Three shall be the number of the counting
... and the number of the counting shall be three.''')
Three shall be the number of the counting
and the number of the counting shall be three.
```

Expressions in formatted string literals are treated like regular Python expressions. Each expression is evaluated in the context where the formatted string literal appears, in order from left to right. An empty expression is not allowed, and both **`lambda`** and assignment expressions `:=` must be surrounded by explicit parentheses:

```
>>> f'{(half := 1/2)}, {half * 42}'
'0.5, 21.0'
```

Se puede reutilizar el tipo de comilla del f-string exterior dentro de un campo de reemplazo:

```
>>> a = dict(x=2)
>>> f"abc {a["x"]} def"
'abc 2 def'
```

También se permiten barras invertidas en campos de reemplazo y se evalúan de la misma manera que cualquier otro contexto:

```
>>> a = ["a", "b", "c"]
>>> print(f"Lista a contiene:\n{"\n".join(a)}")
Lista a contiene:
a
b
```

```
c
```

It is possible to nest f-strings:

```
>>> name = 'world'
>>> f'Repeated:{f' hello {name}' * 3}'
'Repeated: hello world hello world hello world'
```

Portable Python programs should not use more than 5 levels of nesting.

Detalles de implementación de CPython: CPython does not limit nesting of f-strings.

Replacement expressions can contain newlines in both single-quoted and triple-quoted f-strings and they can contain comments. Everything that comes after a `#` inside a replacement field is a comment (even closing braces and quotes). This means that replacement fields with comments must be closed in a different line:

```
>>> a = 2
>>> f"abc{a  # This comment  }"  continues until the enc
...       + 3}"
'abc5'
```

After the expression, replacement fields may optionally contain:

- a *debug specifier* – an equal sign (=), optionally surrounded by whitespace on one or both sides;
- a *conversion specifier* – `!s`, `!r` or `!a`; and/or
- a *format specifier* prefixed with a colon (`:`).

See the Standard Library section on f-strings for details on how these fields are evaluated.

As that section explains, *format specifiers* are passed as the second argument to the **`format()`** function to format a replacement field value. For example, they can be used to specify a field width and padding characters using the Format Specification Mini-Language:

```
>>> number = 14.3
>>> f'{number:20.7f}'
'          14.3000000'
```

Top-level format specifiers may include nested replacement fields:

```
>>> field_size = 20
>>> precision = 7
>>> f'{number:{field_size}.{precision}f}'
'          14.3000000'
```

These nested fields may include their own conversion fields and format specifiers:

```
>>> number = 3
>>> f'{number:{field_size}}'
'                   3'
>>> f'{number:{field_size:05}}'
'00000000000000000003'
```

However, these nested fields may not include more deeply nested replacement fields.

Formatted string literals cannot be used as docstrings, even if they do not include expressions:

```
>>> def foo():
...     f"Not a docstring"
...
>>> print(foo.__doc__)
None
```

Ver también

- **PEP 498** [https://peps.python.org/pep-0498/] – Literal String Interpolation
- **PEP 701** [https://peps.python.org/pep-0701/] – Syntactic formalization of f-strings
- **`str.format()`**, which uses a related format string mechanism.

2.5.8. t-strings

Added in version 3.14.

A *template string literal* or *t-string* is a string literal that is prefixed with “`t`” or “`T`”. These strings follow the same syntax rules as formatted string literals. For differences in evaluation rules, see the Standard Library section on t-strings

2.5.9. Formal grammar for f-strings

F-strings are handled partly by the lexical analyzer, which produces the tokens **`FSTRING_START`**, **`FSTRING_MIDDLE`** and **`FSTRING_END`**, and partly by the parser, which handles expressions in the replacement field. The exact way the work is split is a CPython implementation detail.

Correspondingly, the f-string grammar is a mix of lexical and syntactic definitions.

Whitespace is significant in these situations:

- There may be no whitespace in **`FSTRING_START`** (between the prefix and quote).
- Whitespace in **`FSTRING_MIDDLE`** is part of the literal string contents.
- In `fstring_replacement_field`, if `f_debug_specifier` is present, all whitespace after the opening brace until the `f_debug_specifier`, as well as whitespace immediately following `f_debug_specifier`, is retained as part of the expression.

  Detalles de implementación de CPython: The expression is not handled in the tokenization phase; it is retrieved from the source code using locations of the `{` token and the token after `=`.

The `FSTRING_MIDDLE` definition uses negative lookaheads (`!`) to indicate special characters (backslash, newline, `{`, `}`) and sequences (`f_quote`).

```
fstring:    FSTRING_START fstring_middle* FSTRING_END

FSTRING_START:      fstringprefix ("'" | '"' | "'''" | '
FSTRING_END:        f_quote
fstringprefix:      <("f" | "fr" | "rf"), case-insensiti
f_debug_specifier:  '='
f_quote:            <the quote character(s) used in FSTR

fstring_middle:
    | fstring_replacement_field
    | FSTRING_MIDDLE
FSTRING_MIDDLE:
    | (!"\" !newline !'{' !'}' !f_quote) source_characte
    | stringescapeseq
    | "{{"
    | "}}"
    | <newline, in triple-quoted f-strings only>
fstring_replacement_field:
    | '{' f_expression [f_debug_specifier] [fstring_conve
          [fstring_full_format_spec] '}'
fstring_conversion:
    | "!" ("s" | "r" | "a")
fstring_full_format_spec:
    | ':' fstring_format_spec*
fstring_format_spec:
    | FSTRING_MIDDLE
    | fstring_replacement_field
f_expression:
    | ','.(conditional_expression | "*" or_expr)+ [","]
    | yield_expression
```

Nota

In the above grammar snippet, the `f_quote` and `FSTRING_MIDDLE` rules are context-sensitive – they depend on the contents of `FSTRING_START` of the nearest enclosing `fstring`.

Constructing a more traditional formal grammar from this template is left as an exercise for the reader.

The grammar for t-strings is identical to the one for f-strings, with *t* instead of *f* at the beginning of rule and token names and in the prefix.

```
tstring:    TSTRING_START tstring_middle* TSTRING_END

<rest of the t-string grammar is omitted; see above>
```

2.6. Literales numéricos

`NUMBER` tokens represent numeric literals, of which there are three types: integers, floating-point numbers, and imaginary numbers.

```
NUMBER: integer | floatnumber | imagnumber
```

The numeric value of a numeric literal is the same as if it were passed as a string to the **`int`**, **`float`** or **`complex`** class constructor, respectively. Note that not all valid inputs for those constructors are also valid literals.

Numeric literals do not include a sign; a phrase like `-1` is actually an expression composed of the unary operator “`-`” and the literal `1`.

2.6.1. Literales enteros

Integer literals denote whole numbers. For example:

```
7
3
2147483647
```

There is no limit for the length of integer literals apart from what can be stored in available memory:

```
7922816251426433759354395033679228162514264337593543950
```

Underscores can be used to group digits for enhanced readability, and are ignored for determining the numeric value of the literal. For example, the following literals are equivalent:

```
100_000_000_000
100000000000
1_00_00_00_00_000
```

Underscores can only occur between digits. For example, `_123`, `321_`, and `123__321` are *not* valid literals.

Integers can be specified in binary (base 2), octal (base 8), or hexadecimal (base 16) using the prefixes `0b`, `0o` and `0x`, respectively. Hexadecimal digits 10 through 15 are represented by letters `A-F`, case-insensitive. For example:

```
0b100110111
0b_1110_0101
0o177
0o377
0xdeadbeef
0xDead_Beef
```

An underscore can follow the base specifier. For example, `0x_1f` is a valid literal, but `0_x1f` and `0x__1f` are not.

Leading zeros in a non-zero decimal number are not allowed. For example, `0123` is not a valid literal. This is for disambiguation with C-style octal literals, which Python used before version 3.0.

Formally, integer literals are described by the following lexical definitions:

```
integer:      decinteger | bininteger | octinteger | hex
decinteger:   nonzerodigit (["_"] digit)*
bininteger:   "0" ("b" | "B") (["_"] bindigit)+
octinteger:   "0" ("o" | "O") (["_"] octdigit)+
hexinteger:   "0" ("x" | "X") (["_"] hexdigit)+
zerointeger:   "0"+ (["_"] "0")*
nonzerodigit:  "1"..."9"
digit:         "0"..."9"
bindigit:      "0" | "1"
octdigit:      "0"..."7"
hexdigit:      digit | "a"..."f" | "A"..."F"
```

Distinto en la versión 3.6: Los guiones bajos están ahora permitidos para agrupar en literales.

2.6.2. Literales de punto flotante

Floating-point (float) literals, such as `3.14` or `1.5`, denote approximations of real numbers.

They consist of *integer* and *fraction* parts, each composed of decimal digits. The parts are separated by a decimal point, `.`:

```
2.71828
4.0
```

Unlike in integer literals, leading zeros are allowed. For example, `077.010` is legal, and denotes the same number as `77.01`.

As in integer literals, single underscores may occur between digits to help readability:

```
96_485.332_123
3.14_15_93
```

Either of these parts, but not both, can be empty. For example:

```
10.   # (equivalent to 10.0)
.001  # (equivalent to 0.001)
```

Optionally, the integer and fraction may be followed by an *exponent*: the letter `e` or `E`, followed by an optional sign, `+` or `-`, and a number in the same format as the integer and fraction parts. The `e` or `E` represents «times ten raised to the power of»:

```
1.0e3  # (represents 1.0×10³, or 1000.0)
```

```
1.166e-5  # (represents 1.166×10⁻⁵, or 0.00001166)
6.02214076e+23  # (represents 6.02214076×10²³, or 602214
```

In floats with only integer and exponent parts, the decimal point may be omitted:

```
1e3  # (equivalent to 1.e3 and 1.0e3)
0e0  # (equivalent to 0.)
```

Formally, floating-point literals are described by the following lexical definitions:

```
floatnumber:
    | digitpart "." [digitpart] [exponent]
    | "." digitpart [exponent]
    | digitpart exponent
digitpart: digit (["_"] digit)*
exponent:  ("e" | "E") ["+" | "-"] digitpart
```

Distinto en la versión 3.6: Los guiones bajos están ahora permitidos para agrupar en literales.

2.6.3. Literales imaginarios

Python has complex number objects, but no complex literals. Instead, *imaginary literals* denote complex numbers with a zero real part.

For example, in math, the complex number $3+4.2i$ is written as the real number 3 added to the imaginary number $4.2i$. Python uses a similar syntax, except the imaginary unit is written as `j` rather than i:

```
3+4.2j
```

This is an expression composed of the integer literal `3`, the operator "+", and the imaginary literal `4.2j`. Since these are three separate tokens, whitespace is allowed between them:

```
3 + 4.2j
```

No whitespace is allowed *within* each token. In particular, the `j` suffix, may not be separated from the number before it.

The number before the `j` has the same syntax as a floating-point literal. Thus, the following are valid imaginary literals:

```
4.2j
3.14j
10.j
.001j
1e100j
3.14e-10j
3.14_15_93j
```

Unlike in a floating-point literal the decimal point can be omitted if the imaginary number only has an integer part. The number is still evaluated as a floating-point number, not an integer:

```
10j
0j
1000000000000000000000000j   # equivalent to 1e+24j
```

The `j` suffix is case-insensitive. That means you can use `J` instead:

```
3.14J   # equivalent to 3.14j
```

Formally, imaginary literals are described by the following lexical definition:

```
imagnumber: (floatnumber | digitpart) ("j" | "J")
```

2.7. Operators and delimiters

The following grammar defines *operator* and *delimiter* tokens, that is, the generic `OP` token type. A list of these tokens and their names is also available in the `token` module documentation.

```
OP:
    | assignment_operator
    | bitwise_operator
    | comparison_operator
    | enclosing_delimiter
    | other_delimiter
    | arithmetic_operator
    | "..."
    | other_op
```

```
assignment_operator:   "+="  | "-=" | "*="  | "**=" | "/="
                       "&="  | "|=" | "^="  | "<<=" | ">>=
bitwise_operator:      "&"   | "|"  | "^"   | "~"   | "<<"
comparison_operator:   "<="  | ">=" | "<"   | ">"   | "=="
enclosing_delimiter:   "("   | ")"  | "["   | "]"   | "{"
other_delimiter:       ","   | ":"  | "!"   | ";"   | "="
arithmetic_operator:   "+"   | "-"  | "**"  | "*"   | "//"
other_op:              "."   | "@"
```

Nota

Generally, *operators* are used to combine expressions, while *delimiters* serve other purposes. However, there is no clear, formal distinction between the two categories.

Some tokens can serve as either operators or delimiters, depending on usage. For example, `*` is both the multiplication operator and a delimiter used for sequence unpacking, and `@` is both the matrix multiplication and a delimiter that introduces decorators.

For some tokens, the distinction is unclear. For example, some people consider `.`, `(`, and `)` to be delimiters, while others see the **`getattr()`** operator and the function call operator(s).

Some of Python’s operators, like `and`, `or`, and `not in`, use keyword tokens rather than «symbols» (operator tokens).

A sequence of three consecutive periods (`...`) has a special meaning as an **`Ellipsis`** literal.

[1] [2] [3] [4] [5]

3. Modelo de datos

3.1. Objetos, valores y tipos

Objects are Python’s abstraction for data. All data in a Python program is represented by objects or by relations between objects. Even code is represented by objects.

Cada objeto tiene una identidad, un tipo y un valor. La *identidad* de un objeto nunca cambia una vez que ha sido creado; puede pensar en ello como la dirección del objeto en la memoria. El operador `is` compara la identidad de dos objetos; la función `id()` retorna un número entero que representa su identidad.

Para CPython, `id(x)` es la dirección de memoria donde se almacena `x`.

El tipo de un objeto determina las operaciones que admite el objeto (por ejemplo, «¿tiene una longitud?») y también define los posibles valores para los objetos de ese tipo. La función `type()` retorna el tipo de un objeto (que es un objeto en sí mismo). Al igual que su identidad, también el *type* de un objeto es inmutable. [1]

El *valor* de algunos objetos puede cambiar. Se dice que los objetos cuyo valor puede cambiar son *mutables*; Los objetos cuyo valor no se puede modificar una vez que se crean se denominan *inmutables*. (El valor de un objeto contenedor inmutable que contiene una referencia a un objeto mutable puede cambiar cuando se cambia el valor de este último; sin embargo, el contenedor todavía se considera inmutable, porque la colección de objetos que contiene no se puede cambiar. Por lo tanto, la inmutabilidad no es estrictamente lo mismo que tener un valor inmutable, es más sutil). La mutabilidad de un objeto está determinada por su tipo; por ejemplo, los números, las cadenas de caracteres y las tuplas son inmutables, mientras que los diccionarios y las listas son mutables.

Los objetos nunca se destruyen explícitamente; sin embargo, cuando se vuelven inalcanzables, se pueden recolectar basura. Se permite a una implementación posponer la recolección de basura u omitirla por completo; es una cuestión de calidad de la implementación cómo se implementa la recolección de basura, siempre que no se recolecten objetos que todavía sean accesibles.

CPython actualmente utiliza un esquema de conteo de referencias con detección retardada (opcional) de basura enlazada cíclicamente, que recolecta la mayoría de los objetos tan pronto como se vuelven inalcanzables, pero no se garantiza que recolecte basura que contenga referencias circulares. Vea la documentación del módulo `gc` para información sobre el control de la recolección de basura cíclica. Otras implementaciones actúan de manera diferente y CPython puede cambiar. No dependa de la finalización inmediata de los objetos cuando se vuelvan inalcanzables (por lo que siempre debe cerrar los archivos explícitamente).

Tenga en cuenta que el uso de las funciones de rastreo o depuración de la implementación puede mantener activos los objetos que normalmente serían coleccionables. También tenga en cuenta que la captura de una excepción con una sentencia `try`…`except` puede mantener objetos activos.

Algunos objetos contienen referencias a recursos «externos» como archivos abiertos o ventanas. Se entiende que estos recursos se liberan cuando el objeto es eliminado por el recolector de basura, pero como no se garantiza que la recolección de basura suceda, dichos objetos también proporcionan una forma explícita de liberar el recurso externo, generalmente un método `close()`. Se recomienda encarecidamente a los programas cerrar explícitamente dichos objetos. La declaración `try` … `finally` y la declaración `with` proporcionan formas convenientes de hacer esto.

Algunos objetos contienen referencias a otros objetos; estos se llaman *contenedores*. Ejemplos de contenedores son tuplas, listas y diccionarios. Las referencias son parte del valor de un contenedor. En la mayoría de los casos, cuando hablamos del valor de un contenedor, implicamos los valores, no las identidades de los objetos contenidos; sin embargo, cuando hablamos de la mutabilidad de un contenedor, solo se implican las identidades de los objetos contenidos inmediatamente. Entonces, si un contenedor inmutable (como una tupla) contiene una referencia a un objeto mutable, su valor cambia si se cambia ese objeto mutable.

Los tipos afectan a casi todos los aspectos del comportamiento del objeto. Incluso la importancia de la identidad del objeto se ve afectada en cierto sentido: para los tipos inmutables, las operaciones que calculan nuevos valores en realidad pueden retornar una referencia a cualquier objeto existente con el mismo tipo y valor, mientras que para los objetos mutables esto no está permitido. Por ejemplo, al hacer `a = 1; b = 1`, *a* y *b* puede o no referirse al mismo objeto con el valor 1, dependiendo de la implementación. Esto se debe a que `int` es un tipo inmutable, así que la referencia a `1` se puede reusar. Este comportamiento depende de la implementación utilizada, así que no se debe confiar en ello, pero es algo que hay que tener en cuenta al utilizar pruebas de identidad de objetos. Sin embargo, al hacer `c = []; d = []`, *c* y *d* se garantiza que se refieren a dos listas vacías diferentes, únicas y recién creadas. (Tenga en cuenta que `e = f = []` asigna el *mismo* objeto a ambos *e* y *f*.)

3.2. Jerarquía de tipos estándar

A continuación se muestra una lista de los tipos integrados en Python. Los módulos de extensión (escritos en C, Java u otros lenguajes, dependiendo de la implementación) pueden definir tipos adicionales. Las versiones futuras de Python pueden agregar tipos a la jerarquía de tipos (por ejemplo, números racionales, matrices de enteros almacenados de manera eficiente, etc.), aunque tales adiciones a menudo se proporcionarán a través de la biblioteca estándar.

Algunas de las descripciones de tipos a continuación contienen un párrafo que enumera "atributos especiales". Estos son atributos que proporcionan acceso a la implementación y no están destinados para uso general. Su definición puede cambiar en el futuro.

3.2.1. None

Este tipo tiene un solo valor. Hay un solo objeto con este valor. Se accede a este objeto a través del nombre incorporado `None`. Se utiliza para indicar la ausencia de un valor en muchas situaciones, por ejemplo, se retorna desde funciones que no retornan nada explícitamente. Su valor de verdad es falso.

3.2.2. NotImplemented

Este tipo tiene un solo valor. Hay un solo objeto con este valor. Se accede a este objeto a través del nombre integrado **`NotImplemented`**. Los métodos numéricos y los métodos de comparación enriquecidos deben retornar este valor si no implementan la operación para los operandos proporcionados. (El intérprete intentará entonces la operación reflejada, o alguna otra alternativa, dependiendo del operador). No debe evaluarse en un contexto booleano.

Vea Implementar operaciones aritméticas para más detalles.

Distinto en la versión 3.9: Evaluating **`NotImplemented`** in a boolean context was deprecated.

Distinto en la versión 3.14: Evaluating **`NotImplemented`** in a boolean context now raises a **`TypeError`**. It previously evaluated to **`True`** and emitted a **`DeprecationWarning`** since Python 3.9.

3.2.3. Elipsis

Este tipo tiene un solo valor. Hay un solo objeto con este valor. Se accede a este objeto a través del literal `...` o el nombre incorporado `Ellipsis`. Su valor de verdad es verdadero.

3.2.4. `numbers.Number`

Estos son creados por literales numéricos y retornados como resultados por operadores aritméticos y funciones aritméticas integradas. Los objetos numéricos son inmutables; una vez creado su valor nunca cambia. Los números de Python están, por supuesto, fuertemente relacionados con los números matemáticos, pero están sujetos a las limitaciones de la representación numérica en las computadoras.

Las representaciones de cadena de caracteres de las clases numéricas, calculadas por `__repr__()` y `__str__()`, tienen las siguientes propiedades:

- Son literales numéricos válidos que, cuando se pasan a su constructor de clase, producen un objeto que tiene el valor del numérico original.
- La representación está en base 10, cuando sea posible.
- Los ceros iniciales, posiblemente excepto un solo cero antes de un punto decimal, no se muestran.
- Los ceros finales, posiblemente excepto un solo cero después de un punto decimal, no se muestran.
- Solo se muestra un signo cuando el número es negativo.

Python distingue entre números enteros, números de coma flotante y números complejos:

3.2.4.1. `numbers.Integral`

Estos representan elementos del conjunto matemático de números enteros (positivo y negativo).

Nota

Las reglas para la representación de enteros están destinadas a dar la interpretación más significativa de las operaciones de cambio y máscara que involucran enteros negativos.

Hay dos tipos de números enteros:

Enteros (`int`)

Estos representan números en un rango ilimitado, sujetos solo a la memoria (virtual) disponible. Para las operaciones de desplazamiento y máscara, se asume una representación binaria, y los números negativos se representan en una variante del complemento de 2 que da la ilusión de una cadena de caracteres infinita de bits con signo que se extiende hacia la izquierda.

Booleanos (`bool`)

Estos representan los valores de verdad Falso y Verdadero. Los dos objetos que representan los valores `False` y `True` son los únicos objetos booleanos. El tipo booleano es un subtipo del tipo entero y los valores booleanos se comportan como los valores 0 y 1 respectivamente, en casi todos los contextos, con la excepción de que cuando se convierten en una cadena de caracteres, las cadenas de caracteres `"False"` o `"True"` son retornadas respectivamente.

3.2.4.2. `numbers.Real` (`float`)

Estos representan números de punto flotante de precisión doble a nivel de máquina. Está a merced de la arquitectura de la máquina subyacente (y la implementación de C o Java) para el rango aceptado y el manejo del desbordamiento. Python no admite números de coma flotante de precisión simple; el ahorro en el uso del procesador y la memoria, que generalmente son la razón para usarlos, se ven reducidos por la sobrecarga del uso de objetos en Python, por lo que no hay razón para complicar el lenguaje con dos tipos de números de coma flotante.

3.2.4.3. `numbers.Complex` (`complex`)

Estos representan números complejos como un par de números de coma flotante de precisión doble a nivel de máquina. Se aplican las mismas advertencias que para los números de coma flotante. Las partes reales e imaginarias de un número complejo `z` se pueden obtener a través de los atributos de solo lectura `z.real` y `z.imag`.

3.2.5. Secuencias

Estos representan conjuntos ordenados finitos indexados por números no negativos. La función incorporada `len()` retorna el número de elementos de una secuencia. Cuando la longitud de una secuencia es *n*, el conjunto de índices contiene los números 0, 1, …, *n*-1. El elemento *i* de la secuencia *a* se selecciona mediante `a[i]`. Algunas secuencias, incluidas las secuencias integradas, interpretan los subíndices negativos sumando la longitud de la secuencia. Por ejemplo, `a[-2]` es igual a `a[n-2]`, el penúltimo elemento de la secuencia a con longitud `n`.

The resulting value must be a nonnegative integer less than the number of items in the sequence. If it is not, an **`IndexError`** is raised.

Sequences also support slicing: `a[start:stop]` selects all items with index *k* such that *start* <= *k* < *stop*. When used as an expression, a slice is a sequence of the same type. The comment above about negative subscripts also applies to negative slice positions. Note that no error is raised if a slice position is less than zero or larger than the length of the sequence.

If *start* is missing or **`None`**, slicing behaves as if *start* was zero. If *stop* is missing or `None`, slicing behaves as if *stop* was equal to the length of the sequence.

Algunas secuencias también admiten «segmentación extendida» con un tercer parámetro «paso» : `a[i:j:k]` selecciona todos los elementos de *a* con índice *x* donde `x = i + n*k`, *n* `>= 0` y *i* `<=` *x* < *j*.

Las secuencias se distinguen según su mutabilidad:

3.2.5.1. Secuencias inmutables

Un objeto de un tipo de secuencia inmutable no puede cambiar una vez que se crea. (Si el objeto contiene referencias a otros objetos, estos otros objetos pueden ser mutables y pueden cambiarse; sin embargo, la colección de objetos a los que hace referencia directamente un objeto inmutable no puede cambiar).

Los siguientes tipos son secuencias inmutables:

Cadenas de caracteres

> A string (**`str`**) is a sequence of values that represent *characters*, or more formally, *Unicode code points*. All the code points in the range `0` to `0x10FFFF` can be represented in a string.

Python doesn't have a dedicated *character* type. Instead, every code point in the string is represented as a string object with length `1`.

The built-in function `ord()` converts a code point from its string form to an integer in the range `0` to `0x10FFFF`; `chr()` converts an integer in the range `0` to `0x10FFFF` to the corresponding length `1` string object. `str.encode()` can be used to convert a `str` to `bytes` using the given text encoding, and `bytes.decode()` can be used to achieve the opposite.

Tuplas

The items of a `tuple` are arbitrary Python objects. Tuples of two or more items are formed by comma-separated lists of expressions. A tuple of one item (a "singleton") can be formed by affixing a comma to an expression (an expression by itself does not create a tuple, since parentheses must be usable for grouping of expressions). An empty tuple can be formed by an empty pair of parentheses.

Bytes

A `bytes` object is an immutable array. The items are 8-bit bytes, represented by integers in the range $0 <= x < 256$. Bytes literals (like `b'abc'`) and the built-in `bytes()` constructor can be used to create bytes objects. Also, bytes objects can be decoded to strings via the `decode()` method.

3.2.5.2. Secuencias mutables

Las secuencias mutables se pueden cambiar después de su creación. Las anotaciones de suscripción y segmentación se pueden utilizar como el objetivo de asignaciones y declaraciones `del` (eliminar).

Nota

Los módulos `collections` y `array` proporcionan ejemplos adicionales de tipos de secuencias mutables.

Actualmente hay dos tipos intrínsecos de secuencias mutable:

Listas

Los elementos de una lista son objetos de Python arbitrarios. Las listas se forman colocando una lista de expresiones separadas por comas entre corchetes. (Tome en cuenta que no hay casos especiales necesarios para formar listas de longitud 0 o 1.)

Colecciones de bytes

Un objeto bytearray es una colección mutable. Son creados por el constructor incorporado `bytearray()`. Además de ser mutables (y, por lo tanto, inquebrantable), las colecciones de bytes proporcionan la misma interfaz y funcionalidad que los objetos inmutables `bytes`.

3.2.6. Tipos de conjuntos

Estos representan conjuntos finitos no ordenados de objetos únicos e inmutables. Como tal, no pueden ser indexados por ningún *subscript*. Sin embargo, pueden repetirse y la función incorporada `len()` retorna el número de elementos en un conjunto. Los usos comunes de los conjuntos son pruebas rápidas de membresía, eliminación de duplicados de una secuencia y cálculo de operaciones matemáticas como intersección, unión, diferencia y diferencia simétrica.

Para elementos del conjunto, se aplican las mismas reglas de inmutabilidad que para las claves de diccionario. Tenga en cuenta que los tipos numéricos obedecen las reglas normales para la comparación numérica: si dos números se comparan igual (por ejemplo, `1` y `1.0`), solo uno de ellos puede estar contenido en un conjunto.

Actualmente hay dos tipos de conjuntos intrínsecos:

Conjuntos

These represent a mutable set. They are created by the built-in `set()` constructor and can be modified afterwards by several methods, such as `add()`.

Conjuntos congelados

Estos representan un conjunto inmutable. Son creados por el constructor incorporado **`frozenset()`**. Como un conjunto congelado es inmutable y hashable, se puede usar nuevamente como un elemento de otro conjunto o como una clave de un diccionario.

3.2.7. Mapeos

Estos representan conjuntos finitos de objetos indexados por conjuntos de índices arbitrarios. La notación de subíndice `a[k]` selecciona el elemento indexado por `k` del mapeo `a`; esto se puede usar en expresiones y como el objetivo de asignaciones o declaraciones **`del`**. La función incorporada **`len()`** retorna el número de elementos en un mapeo.

Actualmente hay un único tipo de mapeo intrínseco:

3.2.7.1. Diccionarios

Estos representan conjuntos finitos de objetos indexados por valores casi arbitrarios. Los únicos tipos de valores no aceptables como claves son valores que contienen listas o diccionarios u otros tipos mutables que se comparan por valor en lugar de por identidad de objeto, la razón es que la implementación eficiente de los diccionarios requiere que el valor *hash* de una clave permanezca constante. Los tipos numéricos utilizados para las claves obedecen las reglas normales para la comparación numérica: si dos números se comparan igual (por ejemplo, `1` y `1.0`) entonces se pueden usar indistintamente para indexar la misma entrada del diccionario.

Los diccionarios conservan el orden de inserción, lo que significa que las claves se mantendrán en el mismo orden en que se agregaron secuencialmente sobre el diccionario. Reemplazar una clave existente no cambia el orden, sin embargo, eliminar una clave y volver a insertarla la agregará al final en lugar de mantener su lugar anterior.

Los diccionarios son mutables; pueden ser creados por la notación `{}` (vea la sección Despliegues de diccionario).

Los módulos de extensión `dbm.ndbm` y `dbm.gnu` proporcionan ejemplos adicionales de tipos de mapeo, al igual que el módulo `collections`.

Distinto en la versión 3.7: Los diccionarios no conservaban el orden de inserción en las versiones de Python anteriores a 3.6. En CPython 3.6, el orden de inserción se conserva, pero se consideró un detalle de implementación en ese momento en lugar de una garantía de idioma.

3.2.8. Tipos invocables

Estos son los tipos a los que la operación de llamada de función (vea la sección Invocaciones) puede ser aplicado:

3.2.8.1. Funciones definidas por el usuario

Un objeto función definido por el usuario, es creado por un definición de función (vea la sección Definiciones de funciones). Debe llamarse con una lista de argumentos que contenga el mismo número de elementos que la lista de parámetros formales de la función.

3.2.8.1.1. Atributos especiales de solo lectura

Atributo	**Significado**
function.__builtins__	A reference to the `dictionary` that holds the function's builtins namespace. *Added in version 3.10.*
function.__globals__	Una referencia al `dictionary` que contiene las global variables de la función – el espacio de nombres global del módulo en el que se definió la función.
function.__closure__	None or a `tuple` of cells that contain bindings for the names specified in the `co_freevars` attribute of the function's `code object`. Un objeto de celda tiene el atributo `cell_contents`. Esto se puede usar para obtener el valor de la celda, así como para establecer el valor.

3.2.8.1.2. Atributos especiales de escritura

La mayoría de estos atributos verifican el tipo del valor asignado:

Atributo	Significado
function.__doc__	El texto de documentación de la función, o `None` si no está disponible.
function.__name__	El nombre de la función. Vea también: **`atributos __name__`**.
function.__qualname__	El qualified name de la función. Vea también **`atributos __qualname__`**. *Added in version 3.3.*
function.__module__	El nombre del módulo en el que se definió la función, o `None` si no está disponible.
function.__defaults__	Una **`tuple`** que contiene valores de parameter predeterminados para aquellos argumentos que tienen valores predeterminados, o `None` si ningún parámetro tiene un valor predeterminado.
function.__code__	El objeto de código que representa el cuerpo de la función compilada.
function.__dict__	El espacio de nombres que admite atributos de funciones arbitrarias. Vea también: **`atributos __dict__`**.
function.__annotations__	A **`dictionary`** containing annotations of parameters. The keys of the dictionary are the parameter names, and `'return'` for the return annotation, if provided. See also: **`object.__annotations__`**. *Distinto en la versión 3.14:* Annotations are now lazily evaluated. See **PEP 649**.
function.__annotate__	The annotate function for this function, or `None` if the function has no annotations. See **`object.__annotate__`**. *Added in version 3.14.*
function.__kwdefaults__	Un **`diccionario`** que contiene valores predeterminados para parámetros de solo palabras clave.
function.__type_params__	Un objeto **`tuple`** que contiene los parámetros de tipo de una función genérica. *Added in version 3.12.*

Los objetos de función también admiten la obtención y configuración de atributos arbitrarios, que se pueden usar, por ejemplo, para adjuntar metadatos a funciones. La notación de puntos de atributo regular se utiliza para obtener y establecer dichos atributos.

La implementación actual de CPython solo admite atributos de función en funciones definidas por el usuario. Los atributos de función en funciones incorporadas se pueden soportar en el futuro.

Se puede obtener información adicional sobre la definición de una función a partir de su código de objeto (accesible a través del atributo `__code__`).

3.2.8.2. Métodos de instancia

Un objeto de método de instancia combina una clase, una instancia de clase y cualquier objeto invocable (normalmente una función definida por el usuario).

Atributos especiales de solo lectura:

method.__self__	Hace referencia al objeto de instancia de la clase al que está vinculado el método
method.__func__	Hace referencia al objeto de función original
method.__doc__	La documentación del método (igual que `method.__func__.__doc__`). Una `cadena de caracteres` si la función original tenía un docstring, de lo contrario `None`.
method.__name__	El nombre del método (igual que `method.__func__.__name__`)
method.__module__	El nombre del módulo en el que se definió el método, o `None` si no está disponible.

Los métodos también admiten obtener (más no establecer) los atributos arbitrarios de la función en el objeto de función subyacente.

Los objetos de métodos definidos por usuarios pueden ser creados al obtener el atributo de una clase (probablemente a través de la instancia de dicha clase), si tal atributo es un objeto de función definido por el usuario o un objeto `classmethod`.

Cuando un objeto de instancia de método es creado al obtener un objeto de función definida por el usuario desde una clase a través de una de sus instancias, su atributo `__self__` es la instancia, y el objeto de método se dice que está *enlazado*. El nuevo atributo de método `__func__` es el objeto de función original.

Cuando un objeto de instancia de método es creado al obtener un objeto `classmethod` a partir de una clase o instancia, su atributo `__self__` es la clase misma, y su atributo `__func__` es el objeto de función subyacente al método de la clase.

Cuando el objeto de la instancia de método es invocado, la función subyacente (`__func__`) es llamada, insertando la instancia de clase (`__self__`) delante de la lista de argumentos. Por ejemplo, cuando `C` es una clase que contiene la definición de una función `f()`, y `x` es una instancia de `C`, invocar `x.f(1)` es equivalente a invocar `C.f(x, 1)`.

Cuando el objeto de instancia de método es derivado del objeto `classmethod`, la «instancia de clase» almacenada en `__self__` en realidad será la clase misma, de manera que invocar ya sea `x.f(1)` o `C.f(1)` es equivalente a invocar `f(C,1)` donde `f` es la función subyacente.

Es importante señalar que las funciones definidas por el usuario que son atributos de una instancia de una clase no se convierten en métodos enlazados; esto *solo* ocurre cuando la función es un atributo de la clase.

3.2.8.3. Funciones generadoras

Una función o método que utiliza la instrucción `yield` (consulte la sección La declaración yield) se denomina *función generadora*. Una función de este tipo, cuando se llama, siempre retorna un objeto iterator que se puede usar para ejecutar el cuerpo de la función: llamar al método `iterator.__next__()` del iterador hará que la función se ejecute hasta que proporcione un valor usando la instrucción `yield`. Cuando la función ejecuta una instrucción `return` o se sale del final, se genera una excepción `StopIteration` y el iterador habrá llegado al final del conjunto de valores que se retornarán.

3.2.8.4. Funciones de corrutina

Una función o método que es definido utilizando `async def` se llama *coroutine function*. Dicha función, cuando es invocada, retorna un objeto coroutine. Éste puede contener expresiones `await`, así como declaraciones `async with` y `async for`. Ver también la sección Objetos de corrutina.

3.2.8.5. Funciones generadoras asincrónicas

Una función o método que se define usando `async def` y que usa la declaración `yield` se llama *función generadora asíncrona*. Una función de este tipo, cuando se llama, retorna un objeto asynchronous iterator que se puede utilizar en una instrucción `async for` para ejecutar el cuerpo de la función.

Llamar al método `aiterator.__anext__` del iterador asíncrono retornará un awaitable que, cuando se espere, se ejecutará hasta que proporcione un valor utilizando la expresión `yield`. Cuando la función ejecuta una instrucción `return` vacía o se sale del final, se genera una excepción `StopAsyncIteration` y el iterador asincrónico habrá llegado al final del conjunto de valores que se generarán.

3.2.8.6. Funciones incorporadas

Un objeto de función incorporada es un envoltorio (wrapper) alrededor de una función C. Ejemplos de funciones incorporadas son `len()` y `math.sin()` (`math` es un módulo estándar incorporado). El número y tipo de argumentos son determinados por la función C. Atributos especiales de solo lectura:

- **`__doc__`** es el texto de documentación de la función, o `None` si no está disponible. Vea `function.__doc__`.
- **`__name__`** es el nombre de la función. Vea `function.__name__`.
- **`__self__`** se establece en `None` (pero vea el siguiente punto).
- **`__module__`** es el nombre del módulo en el que se definió la función o `None` si no está disponible. Vea `function.__module__`.

3.2.8.7. Métodos incorporados

Éste es realmente un disfraz distinto de una función incorporada, esta vez teniendo un objeto que se pasa a la función C como un argumento extra implícito. Un ejemplo de un método incorporado es `alist.append()`, asumiendo que *alist* es un objeto de lista. En este caso, el atributo especial de solo lectura **`__self__`** es establecido al objeto indicado por *alist*. (El atributo tiene la misma semántica que con **`otros métodos de instancia`**.)

3.2.8.8. Clases

Las clases son invocables. Estos objetos normalmente actúan como fábricas para nuevas instancias de sí mismos, pero son posibles variaciones para los tipos de clases que anulan **`__new__()`**. Los argumentos de la llamada se pasan a **`__new__()`** y, en el caso típico, a **`__init__()`** para inicializar la nueva instancia.

3.2.8.9. Instancias de clases

Las instancias de clases arbitrarias se pueden hacer invocables definiendo un método **`__call__()`** en su clase.

3.2.9. Módulos

Los módulos son una unidad básica organizacional en código Python, y son creados por el sistema de importación al ser invocados ya sea por la declaración **`import`**, o invocando funciones como **`importlib.import_module()`** y la incorporada **`__import__()`**. Un objeto de módulo tiene un espacio de nombres implementado por un objeto **`diccionario`** (éste es el diccionario al que hace referencia el atributo de funciones **`__globals__`** definido en el módulo). Las referencias de atributos son traducidas a búsquedas en este diccionario, p. ej., `m.x` es equivalente a `m.__dict__["x"]`. Un objeto de módulo no contiene el objeto de código utilizado para iniciar el módulo (ya que no es necesario una vez que la inicialización es realizada).

La asignación de atributos actualiza el diccionario de espacio de nombres del módulo, p. ej., `m.x = 1` es equivalente a `m.__dict__["x"] = 1`.

3.2.9.1. Import-related attributes on module objects

Module objects have the following attributes that relate to the import system. When a module is created using the machinery associated with the import system, these attributes are filled in based on the module's spec, before the loader executes and loads the module.

To create a module dynamically rather than using the import system, it's recommended to use `importlib.util.module_from_spec()`, which will set the various import-controlled attributes to appropriate values. It's also possible to use the `types.ModuleType` constructor to create modules directly, but this technique is more error-prone, as most attributes must be manually set on the module object after it has been created when using this approach.

Prudencia

With the exception of `__name__`, it is **strongly** recommended that you rely on `__spec__` and its attributes instead of any of the other individual attributes listed in this subsection. Note that updating an attribute on `__spec__` will not update the corresponding attribute on the module itself:

```
>>> import typing
>>> typing.__name__, typing.__spec__.name
('typing', 'typing')
>>> typing.__spec__.name = 'spelling'
>>> typing.__name__, typing.__spec__.name
('typing', 'spelling')
>>> typing.__name__ = 'keyboard_smashing'
>>> typing.__name__, typing.__spec__.name
('keyboard_smashing', 'spelling')
```

module.__name__

The name used to uniquely identify the module in the import system. For a directly executed module, this will be set to `"__main__"`.

This attribute must be set to the fully qualified name of the module. It is expected to match the value of `module.__spec__.name`.

module.__spec__

A record of the module's import-system-related state.

Set to the `module spec` that was used when importing the module. See Module specs for more details.

Added in version 3.4.

module.__package__

The package a module belongs to.

If the module is top-level (that is, not a part of any specific package) then the attribute should be set to `''` (the empty string). Otherwise, it should be set to the name of the module's package (which can be equal to `module.__name__` if the module itself is a package). See **PEP 366** [https://peps.python.org/pep-0366/] for further details.

This attribute is used instead of `__name__` to calculate explicit relative imports for main modules. It defaults to `None` for modules created dynamically using the `types.ModuleType` constructor; use `importlib.util.module_from_spec()` instead to ensure the attribute is set to a `str`.

It is **strongly** recommended that you use `module.__spec__.parent` instead of `module.__package__`. `__package__` is now only used as a fallback if `__spec__.parent` is not set, and this fallback path is deprecated.

Distinto en la versión 3.4: This attribute now defaults to `None` for modules created dynamically using the `types.ModuleType` constructor. Previously the attribute was optional.

Distinto en la versión 3.6: The value of `__package__` is expected to be the same as `__spec__.parent`. `__package__` is now only used as a fallback during import resolution if `__spec__.parent` is not defined.

Distinto en la versión 3.10: `ImportWarning` is raised if an import resolution falls back to `__package__` instead of `__spec__.parent`.

Distinto en la versión 3.12: Raise `DeprecationWarning` instead of `ImportWarning` when falling back to `__package__` during import resolution.

Deprecated since version 3.13, will be removed in version 3.15: `__package__` will cease to be set or taken into consideration by the import system or standard library.

module.__loader__

The loader object that the import machinery used to load the module.

This attribute is mostly useful for introspection, but can be used for additional loader-specific functionality, for example getting data associated with a loader.

`__loader__` defaults to `None` for modules created dynamically using the `types.ModuleType` constructor; use `importlib.util.module_from_spec()` instead to ensure the attribute is set to a loader object.

It is **strongly** recommended that you use `module.__spec__.loader` instead of `module.__loader__`.

Distinto en la versión 3.4: This attribute now defaults to `None` for modules created dynamically using the `types.ModuleType` constructor. Previously the attribute was optional.

Deprecated since version 3.12, will be removed in version 3.16: Setting `__loader__` on a module while failing to set `__spec__.loader` is deprecated. In Python 3.16, `__loader__` will cease to be set or taken into consideration by the import system or the standard library.

module.__path__

A (possibly empty) sequence of strings enumerating the locations where the package's submodules will be found. Non-package modules should not have a `__path__` attribute. See __path__ attributes on modules for more details.

It is **strongly** recommended that you use `module.__spec__.submodule_search_locations` instead of `module.__path__`.

module.__file__

module.__cached__

`__file__` and `__cached__` are both optional attributes that may or may not be set. Both attributes should be a `str` when they are available.

`__file__` indicates the pathname of the file from which the module was loaded (if loaded from a file), or the pathname of the shared library file for extension modules loaded dynamically from a shared library. It might be missing for certain types of modules, such as C modules that are statically linked into the interpreter, and the import system may opt to leave it unset if it has no semantic meaning (for example, a module loaded from a database).

If `__file__` is set then the `__cached__` attribute might also be set, which is the path to any compiled version of the code (for example, a byte-compiled file). The file does not need to exist to set this attribute; the path can simply point to where the compiled file *would* exist (see **PEP 3147** [https://peps.python.org/pep-3147/]).

Note that `__cached__` may be set even if `__file__` is not set. However, that scenario is quite atypical. Ultimately, the loader is what makes use of the module spec provided by the finder (from which `__file__` and `__cached__` are derived). So if a loader can load from a cached module but otherwise does not load from a file, that atypical scenario may be appropriate.

It is **strongly** recommended that you use `module.__spec__.cached` instead of `module.__cached__`.

Deprecated since version 3.13, will be removed in version 3.15: Setting `__cached__` on a module while failing to set `__spec__.cached` is deprecated. In Python 3.15, `__cached__` will cease to be set or taken into consideration by the import system or standard library.

3.2.9.2. Other writable attributes on module objects

As well as the import-related attributes listed above, module objects also have the following writable attributes:

module.__doc__

The module’s documentation string, or `None` if unavailable. See also: `__doc__ attributes`.

module.__annotations__

A dictionary containing variable annotations collected during module body execution. For best practices on working with `__annotations__`, see `annotationlib`.

Distinto en la versión 3.14: Annotations are now lazily evaluated. See **PEP 649** [https://peps.python.org/pep-0649/].

module.__annotate__

The annotate function for this module, or `None` if the module has no annotations. See also: `__annotate__` attributes.

Added in version 3.14.

3.2.9.3. Module dictionaries

Module objects also have the following special read-only attribute:

module._dict_

The module’s namespace as a dictionary object. Uniquely among the attributes listed here, `__dict__` cannot be accessed as a global variable from within a module; it can only be accessed as an attribute on module objects.

Debido a la manera en la que CPython limpia los diccionarios de módulo, el diccionario de módulo será limpiado cuando el módulo se encuentra fuera de alcance, incluso si el diccionario aún tiene referencias existentes. Para evitar esto, copie el diccionario o mantenga el módulo cerca mientras usa el diccionario directamente.

3.2.10. Clases personalizadas

Los tipos de clases personalizadas son normalmente creadas por definiciones de clases (ver sección Definiciones de clase). Una clase tiene implementado un espacio de nombres por un objeto de diccionario. Las referencias de atributos de clase son traducidas a búsquedas en este diccionario, p. ej., `C.x` es traducido a `C.__dict__["x"]` (aunque hay una serie de enlaces que permiten la ubicación de atributos por otros medios). Cuando el nombre de atributo no es encontrado ahí, la búsqueda de atributo continúa en las clases base. Esta búsqueda de las clases base utiliza la orden de resolución de métodos C3 que se comporta correctamente aún en la presencia de estructuras de herencia “diamante” donde existen múltiples rutas de herencia que llevan a un ancestro común. Detalles adicionales en el MRO C3 utilizados por Python pueden ser encontrados en The Python 2.3 Method Resolution Order.

Cuando la referencia de un atributo de clase (digamos, para la clase `C`) produce un objeto de método de clase, éste es transformado a un objeto de método de instancia cuyo atributo `__self__` es `C`. Cuando produce un objeto `staticmethod`, éste es transformado al objeto envuelto por el objeto de método estático. Ver sección Implementando descriptores para otra manera en la que los atributos obtenidos de una clase pueden diferir de los que en realidad están contenidos en su `__dict__`.

Las asignaciones de atributos de clase actualizan el diccionario de la clase, nunca el diccionario de la clase base.

Un objeto de clase puede ser invocado (ver arriba) para producir una instancia de clase (ver a continuación).

3.2.10.1. Atributos especiales

Atributo	**Significado**
type.__name__	El nombre de la clase. Vea también: **atributos `__name__`**.
type.__qualname__	El qualified name de la clase. Vea también: **atributos `__qualname__`**.
type.__module__	El nombre del módulo en el que se definió la clase.
type.__dict__	Un **proxy de mapeo** que proporciona una vista de solo lectura del espacio de nombres de la clase. Vea también: **atributos `__dict__`**.
type.__bases__	Un objeto **`tuple`** que contiene las bases de la clase. En la mayoría de los casos, para una clase definida como `class X(A, B, C)`, `X.__bases__` será exactamente igual a `(A, B, C)`.
type.__base__	**Detalles de implementación de CPython:** The single base class in the inheritance chain that is responsible for the memory layout of instances. This attribute corresponds to **`tp_base`** at the C level.
type.__doc__	El texto de documentación de la clase, o `None` si no está disponible. No heredado por subclases.
type.__annotations__	A dictionary containing variable annotations collected during class body execution. See also: **`__annotations__` attributes**. For best practices on working with **`__annotations__`**, please see **`annotationlib`**. Use **`annotationlib.get_annotations()`** instead of accessing this attribute directly.

Advertencia

Accessing the **`__annotations__`** attribute directly on a class object may return annotations for the wrong class, specifically in certain cases where the class, its base class, or a metaclass is defined under `from __future__ import annotations`. See 749 for details.

This attribute does not exist on certain builtin classes. On user-defined classes without `__annotations__`, it is an empty dictionary.

Distinto en la versión 3.14: Annotations are now lazily evaluated. See **PEP 649**.

type.__annotate__()

The annotate function for this class, or `None` if the class has no annotations. See also: **`__annotate__ attributes`**.

Added in version 3.14.

type.__type_params__

Un objeto **`tuple`** que contiene parámetros de tipo de una clase genérica.

Added in version 3.12.

type.__static_attributes__

Un objeto **`tuple`** que contiene nombres de atributos de esta clase que se asignan a través de `self.X` desde cualquier función en su cuerpo.

Added in version 3.13.

type.__firstlineno__

The line number of the first line of the class definition, including decorators. Setting the **`__module__`** attribute removes the **`__firstlineno__`** item from the type's dictionary.

Added in version 3.13.

type.__mro__

El objeto **`tuple`** de clases que se consideran al buscar clases base durante la resolución del métodos.

3.2.10.2. Métodos especiales

Además de los atributos especiales descritos anteriormente, todas las clases de Python también tienen los siguientes dos métodos disponibles:

type.mro()

Este método se puede anular por una metaclase para personalizar el orden de resolución de métodos para sus instancias. Se llama en la instanciación de la clase, y su resultado se almacena en **`__mro__`**.

type.__subclasses__()

Cada clase mantiene una lista de referencias débiles a sus subclases inmediatas. Este método retorna una lista de todas esas referencias aún vivas. La lista está en orden de definición. Ejemplo:

```
>>> class A: pass
>>> class B(A): pass
>>> A.__subclasses__()
[<class 'B'>]
```

3.2.11. Instancias de clase

Una instancia de clase se crea llamando a un objeto de clase (ver arriba). Una instancia de clase tiene un espacio de nombres implementado como un diccionario, que es el primer lugar en el que se buscan las referencias de atributos. Cuando no se encuentra un atributo allí y la clase de la instancia tiene un atributo con ese nombre, la búsqueda continúa con los atributos de la clase. Si se encuentra un atributo de clase que es un objeto de función definido por el usuario, se transforma en un objeto de método de instancia cuyo atributo `__self__` es la instancia. Los objetos de métodos estáticos y de clases también se transforman; consulte más arriba en «Clases». Consulte la sección Implementando descriptores para conocer otra forma en la que los atributos de una clase recuperados a través de sus instancias pueden diferir de los objetos realmente almacenados en el `__dict__` de la clase. Si no se encuentra ningún atributo de clase y la clase del objeto tiene un método `__getattr__()`, se llama para satisfacer la búsqueda.

Las asignaciones y eliminaciones de atributos actualizan el diccionario de la instancia, nunca el diccionario de una clase. Si la clase tiene un método `__setattr__()` o `__delattr__()`, se llama a este en lugar de actualizar el diccionario de instancia directamente.

Instancias de clases pueden pretender ser números, secuencias o mapeos si tienen métodos con ciertos nombres especiales. Ver sección Nombres especiales de método.

3.2.11.1. Atributos especiales

object.__class__

La clase a la que pertenece una instancia de clase.

object.__dict__

Un diccionario u otro objeto de mapeo utilizado para almacenar los atributos (escribibles) de un objeto. No todas las instancias tienen un atributo **`__dict__`**; vea la sección sobre __slots__ para más detalles.

3.2.12. Objetos E/S (también conocidos como objetos de archivo)

Un file object representa un archivo abierto. Diversos accesos directos se encuentran disponibles para crear objetos de archivo: la función incorporada **`open()`**, así como **`os.popen()`**, **`os.fdopen()`**, y el método de objetos socket **`makefile()`** (y quizás por otras funciones y métodos proporcionados por módulos de extensión).

File objects implement common methods, listed below, to simplify usage in generic code. They are expected to be Gestores de Contexto en la Declaración with.

Los objetos `sys.stdin`, `sys.stdout` y `sys.stderr` son iniciados a objetos de archivos correspondientes a la entrada y salida estándar del intérprete, así como flujos de error; todos ellos están abiertos en el modo de texto y por lo tanto siguen la interface definida por la clase abstracta **`io.TextIOBase`**.

file.read(*size=-1*, */*)

Retrieve up to *size* data from the file. As a convenience if *size* is unspecified or -1 retrieve all data available.

file.write(*data*, */*)

Store *data* to the file.

file.close()

Flush any buffers and close the underlying file.

3.2.13. Tipos internos

Algunos tipos utilizados internamente por el intérprete son expuestos al usuario. Sus definiciones pueden cambiar en futuras versiones del intérprete, pero son mencionadas aquí para complementar.

3.2.13.1. Objetos de código

Los objetos de código representan código de Python ejecutable *compilado por bytes*, o bytecode. La diferencia entre un objeto de código y un objeto de función es que el objeto de función contiene una referencia explícita a los globales de la función (el módulo en el que fue definido), mientras el objeto de código no contiene contexto; de igual manera los valores por defecto de los argumentos son almacenados en el objeto de función, no en el objeto de código (porque representan valores calculados en tiempo de ejecución). A diferencia de objetos de función, los objetos de código son inmutables y no contienen referencias (directas o indirectas) a objetos mutables.

3.2.13.1.1. Atributos especiales de solo lectura

codeobject.co_name	El nombre de la función
codeobject.co_qualname	El nombre de la función completamente calificado *Added in version 3.11.*
codeobject.co_argcount	El número total de parámetros posicionales (incluye los parámetros solo posicionales y los parámetros con valores predeterminados) que tiene la función
codeobject.co_posonlyargcount	El número de parámetros solo posicionales (incluye argumentos con valores predeterminados) que tiene la función
codeobject.co_kwonlyargcount	El número de parámetros de solo palabras clave (incluye argumentos con valores predeterminados) que tiene la función
codeobject.co_nlocals	El número de variables locales utilizadas por la función (incluye los parámetros)
codeobject.co_varnames	Un **`tuple`** que contiene los nombres de las variables locales en la función (empezando por los nombres de los parámetros)
codeobject.co_cellvars	A **`tuple`** containing the names of local variables that are referenced from at least one nested scope inside the function
codeobject.co_freevars	A **`tuple`** containing the names of free (closure) variables that a nested scope references in an outer scope. See also

`function.__closure__`.
Note: references to global and builtin names are *not* included.

codeobject.co_code
Una cadena de caracteres que representa la secuencia de instrucciones bytecode en la función

codeobject.co_consts
Un objeto `tuple` que contiene los literales utilizados por bytecode en la función

codeobject.co_names
Un objeto `tuple` que contiene los nombres utilizados por bytecode en la función

codeobject.co_filename
El nombre del fichero en el que se compiló el código

codeobject.co_firstlineno
El número de la primer línea de la función

codeobject.co_lnotab
Una cadena de caracteres que codifica la asignación de los desplazamientos de bytecode a los números de línea. Para obtener más detalles, vea el código fuente del intérprete.
Obsoleto desde la versión 3.12: This attribute of code objects is deprecated, and may be removed in Python 3.15.

codeobject.co_stacksize
El tamaño de pila requerido del objeto de código

codeobject.co_flags
Un `entero` que codifica un número de banderas para el intérprete.

Los siguientes bits de bandera son definidos por `co_flags`: el bit `0x04` es establecido si la función utiliza la sintaxis `*arguments` para aceptar un número arbitrario de argumentos posicionales; el bit `0x08` es establecido si la función utiliza la sintaxis `**keywords` para aceptar argumentos de palabras clave arbitrarios; el bit `0x20` es establecido si la función es un generador. Vea Objetos de código Bit Flags para más detalles sobre la semántica de cada bandera que pueda estar presente.

Future feature declarations (for example, `from __future__ import division`) also use bits in `co_flags` to indicate whether a code object was compiled with a particular feature enabled. See `compiler_flag`.

Otros bits en `co_flags` son reservados para uso interno.

If a code object represents a function and has a docstring, the `CO_HAS_DOCSTRING` bit is set in `co_flags` and the first item in `co_consts` is the docstring of the function.

3.2.13.1.2. Métodos en objetos de código

codeobject.co_positions()

Retorna un iterable sobre las posiciones del código fuente de cada instrucción de bytecode en el objeto de código.

El iterador retorna objetos `tuple` que contienen `(start_line, end_line, start_column, end_column)`. La tupla *i-ésima* corresponde a la posición del código fuente que se compiló en la unidad de código *i-ésima*. La información de la columna son desplazamientos de bytes utf-8 indexados en 0 en la línea de origen dada.

Esta información posicional puede faltar. Una lista no exhaustiva de casos en los que esto puede suceder:

- Ejecutando el intérprete con `-X no_debug_ranges`.
- Cargando un archivo pyc compilado usando `-X no_debug_ranges`.
- Tuplas de posición correspondientes a instrucciones artificiales.
- Números de línea y columna que no se pueden representar debido a limitaciones específicas de la implementación.

Cuando esto ocurre, algunos o todos los elementos de la tupla pueden ser `None`.

Added in version 3.11.

Nota

Esta función requiere el almacenamiento de posiciones de columna en objetos de código, lo que puede resultar en un pequeño aumento del uso del disco de archivos de Python compilados o del uso de la memoria del intérprete. Para evitar almacenar la información extra y/o desactivar la impresión de la información extra de seguimiento, se puede usar el indicador de línea de comando `-X no_debug_ranges` o la variable de entorno `PYTHONNODEBUGRANGES`.

codeobject.co_lines()

Retorna un iterador que produce información sobre rangos sucesivos de objetos bytecode. Cada elemento que se produjo es un objeto **`tuple`** `(start, end, lineno)`:

- `start` (un objeto **`int`**) representa el desplazamiento (inclusivo) del inicio del rango de bytecode
- `end` (un objeto **`int`**) representa el desplazamiento (exclusivo) del final del rango de bytecode
- `lineno` es un objeto **`int`** que representa el número de línea del rango de bytecode, o `None` si los códigos de bytes en el rango dado no tienen número de línea

Los elementos producidos tendrán las siguientes propiedades:

- El primer rango producido tendrá un elemento `start` de 0.
- Los rangos `(start, end)` serán no decrecientes y consecutivos. Esto es, para cualquier par de objetos **`tuple`**, el elemento `start` del segundo será igual al elemento `end` del primero.
- Ningún rango será inverso: `end >= start` para todas las tripletas.
- El último objeto **`tuple`** producido tendrá el elemento `end` igual al tamaño de bytecode.

Se permiten rangos de ancho cero, donde `start == end`. Los rangos de ancho cero se utilizan para las líneas que están presentes en el código fuente, pero que han sido eliminadas por el compilador bytecode.

Added in version 3.10.

Ver también

PEP 626 [https://peps.python.org/pep-0626/] - Números de línea precisos para depuración y otras herramientas.
El PEP que introdujo el método **`co_lines()`**.

codeobject.replace(***kwargs*)

Retorna una copia del objeto de código con nuevos valores para los campos especificados.

Los objetos de código también son compatibles con la función genérica `copy.replace()`.

Added in version 3.8.

3.2.13.2. Objetos de marco

Los objetos de marco representan marcos de ejecución. Pueden ocurrir en objetos de rastreo, y son también pasados hacia funciones de rastreo registradas.

3.2.13.2.1. Atributos especiales de solo lectura

frame.f_back	Apunta al marco de pila anterior (hacia el que hace la llamada), o `None` si este es el marco de pila inferior
frame.f_code	El objeto de código que se está ejecutando en este marco. Al acceder a este atributo se lanza un evento de auditoría `object.__getattr__` con los argumentos `obj` y `"f_code"`.
frame.f_locals	La asignación que utiliza el marco para buscar variables locales. Si el marco hace referencia a un ámbito optimizado, esto puede retornar un objeto proxy de escritura directa. *Distinto en la versión 3.13:* Retorna un proxy para ámbitos optimizados.
frame.f_globals	El diccionario utilizado por el marco para buscar variables globales
frame.f_builtins	El diccionario utilizado por el marco para buscar nombres incorporados (intrínsecos)
frame.f_lasti	La «instrucción precisa» del objeto de marco (es un índice en la cadena bytecode del objeto de código)
frame.f_generator	The generator or coroutine object that owns this frame, or `None` if the frame is a normal function. *Added in version 3.14.*

3.2.13.2.2. Atributos especiales de escritura

frame.f_trace

Si no es `None`, esta es una función llamada por distintos eventos durante la ejecución del código (esto es utilizado por depuradores). Normalmente un evento es desencadenado por cada nueva línea fuente (vea **`f_trace_lines`**).

frame.f_trace_lines

Establecer este atributo a **`False`** para deshabilitar la activación de un evento de seguimiento para cada línea de origen.

frame.f_trace_opcodes

Establecer este atributo a **`True`** para permitir que se soliciten eventos por código de operación. Tenga en cuenta que esto puede llevar a un comportamiento indefinido del intérprete si se levantan excepciones por la función de rastreo escape hacia la función que está siendo rastreada.

frame.f_lineno

El número de línea actual del marco – escribiendo a esta forma dentro de una función de rastreo salta a la línea dada (solo para el último marco). Un depurador puede implementar un comando de salto (*Jump*) (también conocido como *Set Next Statement*) al escribir en este atributo.

3.2.13.2.3. Métodos de objeto de marco

Objetos de marco soportan un método:

frame.clear()

Este método limpia todas las referencias a variables locales mantenidas por el marco. También, si el marco pertenecía a un generator, éste es finalizado. Esto ayuda a interrumpir los ciclos de referencia que involucran objetos de marco (por ejemplo al detectar una excepción y almacenando su rastro para uso posterior).

`RuntimeError` es lanzado si el marco se encuentra actualmente en ejecución o suspendido.

Added in version 3.4.

Distinto en la versión 3.13: Al intentar borrar un marco suspendido lanza **`RuntimeError`** (como siempre ha sido el caso con los marcos en ejecución).

3.2.13.3. Objetos de seguimiento de pila (traceback)

Los objetos de seguimiento de pila representan el trazo de pila (*stack trace*) de una excepción. Un objeto de rastreo es creado de manera implícita cuando se da una excepción, y puede ser creada de manera explícita al llamar **`types.TracebackType`**.

Distinto en la versión 3.7: Los objetos de seguimiento de pila ya pueden ser instanciados de manera explícita desde código de Python.

Para seguimientos de pila (tracebacks) creados de manera implícita, cuando la búsqueda por un manejo de excepciones desenvuelve la pila de ejecución, en cada nivel de desenvolvimiento se inserta un objeto de rastreo al frente del rastreo actual. Cuando se entra a un manejo de excepción, la pila de rastreo se vuelve disponible para el programa. (Ver sección La sentencia try.) Es accesible como el tercer elemento de la tupla retornada por **`sys.exc_info()`**, y como el atributo **`__traceback__`** de la excepción capturada.

Cuando el programa no contiene un gestor apropiado, el trazo de pila es escrito (muy bien formateado) en el flujo de error estándar; si el intérprete es interactivo, también se vuelve disponible al usuario como **`sys.last_traceback`**.

Para seguimientos de pila creados de forma explícita, depende de su creador determinar cómo los atributos **`tb_next`** deberían ser ligados para formar un trazo de pila completo (*full stack trace*).

Atributos especiales de solo lectura:

traceback.tb_frame	Apunta al marco de ejecución del nivel actual. Acceder a este atributo lanza un objeto evento de auditoría `object.__getattr__` con argumentos `obj` y `"tb_frame"`.
traceback.tb_lineno	Indica el número de línea en la que se ha producido la excepción
traceback.tb_lasti	Indica la «instrucción precisa».

El número de línea y la última instrucción en el seguimiento de pila puede diferir del número de línea de su objeto de marco si la excepción ocurrió en una declaración **`try`** sin una cláusula de excepción (except) correspondiente o con una cláusula **`finally`**.

traceback.tb_next

El atributo especial escribible **`tb_next`** es el siguiente nivel en el trazo de pila (hacia el marco en donde ocurrió la excepción), o `None` si no existe un siguiente nivel.

Distinto en la versión 3.7: Este atributo es ahora escribible

3.2.13.4. Objetos de segmento (Slice objects)

Los objetos de sector se utilizan para representar sectores para métodos **`__getitem__()`**. También son creados por la función integrada **`slice()`**.

Atributos especiales de solo lectura: **`start`** es el límite inferior; **`stop`** es el límite superior; **`step`** es el valor de paso; cada uno es `None` si es omitido. Estos atributos pueden ser de cualquier tipo.

Los objetos de segmento soportan un método:

slice.indices(*self*, *length*)

Este método toma un argumento *length* de entero simple y calcula información relacionada con el segmento que el mismo describiría si fuera aplicado a una secuencia de elementos *length*. Retorna una tupla de tres enteros; respectivamente estos son los índices *start* y *stop* y el *step* o longitud del paso del segmento. Índices faltantes o fuera de los límites son manipulados de manera consistente con segmentos regulares.

3.2.13.5. Objetos de método estático

Los objetos de método estático proveen una forma de anular la transformación de objetos de función a objetos de método descritos anteriormente. Un objeto de método estático es una envoltura (*wrapper*) alrededor de cualquier otro objeto, usualmente un objeto de método definido por usuario. Cuando un objeto de método estático es obtenido desde una clase o una instancia de clase, usualmente el objeto retornado es el objeto envuelto, el cual no está objeto a ninguna transformación adicional. Los objetos de método estático también pueden ser llamados. Los objetos de método estático son creados por el constructor incorporado `staticmethod()`.

3.2.13.6. Objetos de método de clase

Un objeto de método de clase, igual que un objeto de método estático, es un envoltorio (wrapper) alrededor de otro objeto que altera la forma en la que el objeto es obtenido desde las clases y las instancias de clase. El comportamiento de los objetos de método de clase sobre tal obtención es descrita más arriba, debajo de «métodos de instancia». Objetos de clase de método son creados por el constructor incorporado `classmethod()`.

3.3. Nombres especiales de método

Una clase puede implementar ciertas operaciones que se invocan mediante una sintaxis especial (como operaciones aritméticas o subíndices y divisiones) definiendo métodos con nombres especiales. Este es el enfoque de Python para *sobrecarga de operadores*, permitiendo a las clases definir su propio comportamiento con respecto a los operadores del lenguaje. Por ejemplo, si una clase define un método denominado `__getitem__()` y `x` es una instancia de esta clase, entonces `x[i]` es aproximadamente equivalente a `type(x).__getitem__(x, i)`. Excepto donde se mencione, los intentos de ejecutar una operación generan una excepción cuando no se define ningún método apropiado (normalmente `AttributeError` o `TypeError`).

Establecer un método especial en `None` indica que la operación correspondiente no está disponible. Por ejemplo, si una clase establece `__iter__()` en `None`, la clase no es iterable, por lo que llamar a `iter()` en sus instancias generará un `TypeError` (sin recurrir a `__getitem__()`). [2]

When implementing a class that emulates any built-in type, it is important that the emulation only be implemented to the degree that it makes sense for the object being modelled. For example, some sequences may work well with retrieval of individual elements, but extracting a slice may not make sense. (One example of this is the NodeList interface in the W3C’s Document Object Model.)

3.3.1. Personalización básica

object.__new__(*cls*[, ...])

Es llamado para crear una nueva instancia de clase *cls*. `__new__()` es un método estático (como un caso especial, así que no se necesita declarar como tal) que toma la clase de donde fue solicitada una instancia como su primer argumento. Los argumentos restantes son aquellos que se pasan a la expresión del constructor de objetos (para llamar a la clase). El valor retornado de `__new__()` deberá ser la nueva instancia de objeto (normalmente una instancia de *cls*).

Las implementaciones típicas crean una nueva instancia de la clase invocando el método `__new__()` de la superclase usando `super().__new__(cls[, ...])` con los argumentos apropiados y luego modificando la instancia recién creada según sea necesario antes de retornarla.

Si `__new__()` es invocado durante la construcción del objeto y éste retorna una instancia de *cls*, entonces el nuevo método `__init__()` de la instancia será invocado como `__init__(self[, …])`, donde *self* es la nueva instancia y los argumentos restantes son iguales a como fueron pasados hacia el constructor de objetos.

Si `__new__()` no retorna una instancia de *cls*, entonces el nuevo método `__init__()` de la instancia no será invocado.

`__new__()` es destinado principalmente para permitir a subclases de tipos inmutables (como int, str, o tuple) personalizar la creación de instancias. También es comúnmente anulado en metaclases personalizadas con el fin de personalizar la creación de clase.

object.__init__(*self*[, ...])

Llamado después de que la instancia ha sido creada (por `__new__()`), pero antes es retornada a quien produce la llamada. Los argumentos son aquellos pasados a la expresión del constructor de la clase. Si una clase base tiene un método `__init__()`, el método `__init__()` de clase derivada, de existir, debe llamarlo explícitamente para asegurar la inicialización apropiada de la clase base que es parte de la instancia; por ejemplo: `super().__init__([args...])`.

Debido a que `__new__()` y `__init__()` trabajan juntos construyendo objetos (`__new__()` para crearlo y `__init__()` para personalizarlo), ningún valor distinto a `None` puede ser retornado por `__init__()`; hacer esto puede causar que se lance una excepción `TypeError` en tiempo de ejecución.

object.__del__(*self*)

Llamado cuando la instancia es a punto de ser destruida. Esto también es llamado finalizador o (indebidamente) destructor. Si una clase base tiene un método `__del__()` el método `__del__()` de la clase derivada, de existir, debe llamarlo explícitamente para asegurar la eliminación adecuada de la parte de la clase base de la instancia.

Es posible (¡aunque no recomendable!) para el método `__del__()` posponer la destrucción de la instancia al crear una nueva referencia hacia ésta. Esto es llamado *resurrección* de objeto. Es dependiente de la implementación si `__del__()` es llamado una segunda vez cuando un objeto resucitado está por ser destruido; la implementación CPython actual únicamente lo llama una vez.

No está garantizado que los métodos `__del__()` sean llamados para objetos que aún existen cuando el intérprete se cierra. `weakref.finalize` proporciona una forma directa de registrar una función de limpieza para ser llamada cuando un objeto es recogido de la basura.

Nota

`del x` no llama directamente `x.__del__()` — el primero disminuye el conteo de referencia para `x` uno por uno, y el segundo es llamado únicamente cuando el conteo de referencias de `x` llega a cero.

Es posible que un ciclo de referencia evite que el recuento de referencia de un objeto llegue a cero. En este caso, el ciclo será posteriormente detectado y eliminado por el cyclic garbage collector. Una causa común de los ciclos de referencia es cuando se detecta una excepción en una variable local. Luego, los locales del marco hacen referencia a la excepción, que hace referencia a su propio rastreo, que hace referencia a los locales de todos los marcos capturados en el rastreo.

Ver también

Documentación para el módulo **`gc`**.

Advertencia

Debido a las circunstancias inciertas bajo las que los métodos **`__del__()`** son invocados, las excepciones que ocurren durante su ejecución son ignoradas, y una advertencia es mostrada hacia `sys.stderr`. En particular:

- **`__del__()`** puede ser invocado cuando código arbitrario es ejecutado, incluyendo el de cualquier hilo arbitrario. Si **`__del__()`** necesita realizar un cierre de exclusión mutua (*lock*) o invocar cualquier otro recurso que lo esté bloqueando, podría provocar un bloqueo muto (*deadlock*) ya que el recurso podría estar siendo utilizado por el código que se interrumpe al ejecutar **`__del__()`**.
- **`__del__()`** puede ser ejecutado durante el cierre del intérprete. Como consecuencia, las variables globales que necesita para acceder (incluyendo otros módulos) podrían haber sido borradas o establecidas

a `None`. Python garantiza que los globales cuyo nombre comienza con un guión bajo simple sean borrados de su módulo antes que los globales sean borrados; si no existen otras referencias a dichas globales, esto puede ayudar asegurando que los módulos importados aún se encuentren disponibles al momento de llamar al método **`__del__()`**.

object._repr_(*self*)

Llamado por la función incorporada `repr()` para calcular la cadena “oficial” de representación de un objeto. Si es posible, esto debería verse como una expresión de Python válida que puede ser utilizada para recrear un objeto con el mismo valor (bajo el ambiente adecuado). Si no es posible, una cadena con la forma `<...some useful description...>` debe ser retornada. El valor de retorno debe ser un objeto de cadena (*string*). Si una clase define `__repr__()` pero no `__str__()`, entonces **`__repr__()`** también es utilizado cuando una cadena “informal” de representación de instancias de esas clases son requeridas.

This is typically used for debugging, so it is important that the representation is information-rich and unambiguous. A default implementation is provided by the `object` class itself.

object._str_(*self*)

Called by `str(object)`, the default `__format__()` implementation, and the built-in function `print()`, to compute the «informal» or nicely printable string representation of an object. The return value must be a str object.

Este método difiere de `object.__repr__()` en que no hay expectativas de que `__str__()` retorne una expresión de

Python válida: una representación más conveniente o concisa pueda ser utilizada.

La implementación por defecto definida por el tipo incorporado `object` llama a `object.__repr__()`.

object.__bytes__(*self*)

Called by bytes to compute a byte-string representation of an object. This should return a `bytes` object. The `object` class itself does not provide this method.

object.__format__(*self*, *format_spec*)

Llamado por la función incorporada `format()`, y por extensión, la evaluación de formatted string literals y el método `str.format()`, para producir la representación "formateada" de un objeto. El argumento *format_spec* es una cadena que contiene una descripción de las opciones de formato deseadas. La interpretación del argumento *format_spec* depende del tipo que implementa `__format__()`, sin embargo, ya sea que la mayoría de las clases deleguen el formato a uno de los tipos incorporados, o utilicen una sintaxis de opción de formato similar.

Ver Format specification mini-language para una descripción de la sintaxis de formato estándar.

El valor de retorno debe ser un objeto de cadena.

The default implementation by the `object` class should be given an empty *format_spec* string. It delegates to `__str__()`.

Distinto en la versión 3.4: El método __format__ del mismo `object` lanza un `TypeError` si se la pasa una cadena no vacía.

Distinto en la versión 3.7: `object.__format__(x, '')` es ahora equivalente a `str(x)` en lugar de `format(str(self), '')`.

object.__lt__(*self*, *other*)
object.__le__(*self*, *other*)
object.__eq__(*self*, *other*)
object.__ne__(*self*, *other*)
object.__gt__(*self*, *other*)
object.__ge__(*self*, *other*)

Estos son los llamados métodos de comparación *rich*. La correspondencia entre símbolos de operador y los nombres de método es de la siguiente manera: `x<y` llama `x.__lt__(y)`, `x<=y` llama `x.__le__(y)`, `x==y` llama `x.__eq__(y)`, `x!=y` llama `x.__ne__(y)`, `x>y` llama `x.__gt__(y)`, y `x>=y` llama `x.__ge__(y)`.

Un método de comparación enriquecido puede retornar el único **`NotImplemented`** si no implementa la operación para un par de argumentos dados. Por convención, `False` y `True` son retornados para una comparación exitosa. Sin embargo, estos métodos pueden retornar cualquier valor, así que si el operador de comparación es utilizado en un contexto Booleano (p. ej. en la condición de una sentencia `if`), Python llamará **`bool()`** en dicho valor para determinar si el resultado es verdadero (*true*) o falso (*false*).

Por defecto, `object` implementa **`__eq__()`** usando `is`, retornando **`NotImplemented`** en el caso de una comparación falsa: `True if x is y else NotImplemented`. Para **`__ne__()`**, por defecto delega a **`__eq__()`** e invierte el resultado a menos que sea **`NotImplemented`**. No hay otras relaciones implícitas entre los operadores de comparación o implementaciones predeterminadas; por ejemplo, la verdad de `(x<y o x==y)` no implica `x<=y`. Para generar automáticamente operaciones de ordenación a partir de una sola operación raíz, consulte **`functools.total_ordering()`**.

By default, the **`object`** class provides implementations consistent with Comparaciones de valor: equality compares according to object identity, and order comparisons raise `TypeError`. Each default method may generate these results directly, but may also return `NotImplemented`.

Ver el párrafo sobre `__hash__()` para más notas importantes sobre la creación de objetos hashable que soportan operaciones de comparación personalizadas y son utilizables como llaves de diccionario.

No existen versiones con argumento intercambiado de estos métodos (a ser utilizados cuando el argumento de la izquierda no soporta la operación pero el de la derecha sí); más bien, `__lt__()` y `__gt__()` son el reflejo del otro, `__le__()` y `__ge__()` son un reflejo del otro, y `__eq__()` y `__ne__()` son su propio reflejo. Si los operandos son de tipos distintos, y el tipo de operando de la derecha es una subclase directa o indirecta del tipo de operando de la izquierda, el método reflejado del operando de la derecha tiene prioridad, de otro modo el método del operando de la izquierda tiene prioridad. Subclases virtuales no son consideradas.

Cuando ningún método retorna un valor distinto de `NotImplemented`, los operadores `==` y `!=` vuelven a `is` y `is not`, respectivamente.

object.__hash__(*self*)

Lo llama la función integrada `hash()` y para operaciones en miembros de colecciones hash, incluidas `set`, `frozenset` y `dict`. El método `__hash__()` debería retornar un número entero. La única propiedad requerida es que los objetos que se comparan iguales tengan el mismo valor hash; Se recomienda mezclar los valores hash de los componentes del objeto que también desempeñan un papel en la comparación de objetos empaquetándolos en una tupla y aplicando hash a la tupla. Ejemplo:

```
def __hash__(self):
    return hash((self.name, self.nick, self.color))
```

Nota

`hash()` trunca el valor retornado del método personalizado `__hash__()` del objeto al tamaño de `Py_ssize_t`. Esto normalmente son 8 bytes en estructuras de 64-bits y 4 bytes en estructuras de 32 bits. Si el `__hash__()` de un objeto debe interoperar en estructuras de tamaños de bits diferentes, asegúrese de revisar la amplitud en todas las estructuras soportadas. Una forma fácil de hacer esto es con `python -c "import sys; print(sys.hash_info.width)"`.

Si una clase no define un método `__eq__()` tampoco debería definir una operación `__hash__()`; si define `__eq__()` pero no `__hash__()`, sus instancias no se podrán utilizar como elementos en colecciones hash. Si una clase define objetos mutables e implementa un método `__eq__()`, no debería implementar `__hash__()`, ya que la implementación de colecciones hashable requiere que el valor hash de una clave sea inmutable (si el valor hash del objeto cambia, estará en el depósito hash incorrecto).

User-defined classes have `__eq__()` and `__hash__()` methods by default (inherited from the `object` class); with them, all objects compare unequal (except with themselves) and `x.__hash__()` returns an appropriate value such that `x == y` implies both that `x is y` and `hash(x) == hash(y)`.

Una clase que anula `__eq__()` y no define `__hash__()` tendrá implícito su `__hash__()` establecido a `None`. Cuando el método `__hash__()` de una clase es `None`, instancias de la clase lanzarán un `TypeError` cuando el programa intente obtener el valor del hash, y también será correctamente identificado como de hash no calculable cuando se verifique `isinstance(obj, collections.abc.Hashable)`.

Si una clase que anula `__eq__()` necesita conservar la implementación de `__hash__()` de una clase padre, al intérprete se le debe informar explícitamente estableciendo `__hash__ = <ParentClass>.__hash__`.

Si una clase que no anula **`__eq__()`** desea eliminar el soporte de *hash*, debe incluir `__hash__ = None` en la definición de clase. Una clase que define su propio **`__hash__()`** y que explícitamente lanza un **`TypeError`** será identificado de manera incorrecta como de hash calculable por una llamada `isinstance(obj, collections.abc.Hashable)`.

Nota

Por defecto los valores de objetos str y bytes de **`__hash__()`** son "salados" con un valor aleatorio impredecible. Aunque se mantienen constantes dentro de un proceso Python particular, no son predecibles entre invocaciones repetidas de Python.

This is intended to provide protection against a denial-of-service caused by carefully chosen inputs that exploit the worst case performance of a dict insertion, $O(n^2)$ complexity. See https://ocert.org/advisories/ocert-2011-003.html for details.

Cambiar los valores hash afectan el orden de la iteración de los sets. Python nunca ha dado garantías en relación a este orden (y típicamente varía entre estructuras de 32-bits y 64-bits).

Ver también **`PYTHONHASHSEED`**.

Distinto en la versión 3.3: La aleatorización de hash es habilitada por defecto.

object._bool_(*self*)

Called to implement truth value testing and the built-in operation `bool()`; should return `False` or `True`. When this method is not defined, **`__len__()`** is called, if it is defined, and the object is considered true if its result is nonzero. If a class defines neither **`__len__()`** nor **`__bool__()`** (which is true of the **`object`** class itself), all its instances are considered true.

3.3.2. Personalizando acceso a atributos

Los siguientes métodos pueden ser definidos para personalizar el significado de acceso a atributos (uso de, asignación a, o borrado de `x.name`) para instancias de clase.

object.__getattr__(*self*, *name*)

> Called when the default attribute access fails with an `AttributeError` (either `__getattribute__()` raises an `AttributeError` because *name* is not an instance attribute or an attribute in the class tree for `self`; or `__get__()` of a *name* property raises `AttributeError`). This method should either return the (computed) attribute value or raise an `AttributeError` exception. The `object` class itself does not provide this method.
>
> Tome en cuenta que si el atributo es encontrado a través del mecanismo normal, `__getattr__()` no es llamado. (Esto es una desigualdad intencional entre `__getattr__()` y `__setattr__()`.) Esto es realizado tanto por motivos de eficiencia y porque de otra manera `__getattr__()` no tendría manera de acceder a otros atributos de la instancia. Tome en cuenta que al menos para variables de instancia, se puede fingir control total al no insertar ningún valor en el diccionario de atributo de instancia (sino insertándolos en otro objeto). Ver el método `__getattribute__()` a continuación para una forma de tener control total sobre el acceso de atributo.

object.__getattribute__(*self*, *name*)

> Es llamado incondicionalmente para implementar acceso de atributo por instancias de clase. Si la clase también define `__getattr__()`, éste no será llamado a menos que `__getattribute__()` lo llame de manera explícita o lance una excepción `AttributeError`. Este método deberá retornar el valor de atributo (calculado) o lanzar una excepción `AttributeError`. Para evitar la recursividad infinita en este método, su implementación deberá siempre llamar al método de la clase base con el mismo nombre para

acceder cualquier atributo que necesite, por ejemplo, `object.__getattribute__(self, name)`.

Nota

Este método aún puede ser sobrepasado cuando se buscan métodos especiales como resultado de una invocación implícita a través de sintaxis de lenguaje o funciones incorporadas. Ver Búsqueda de método especial.

Para ciertos accesos a atributos sensibles, lanza un evento de auditoría `object.__getattr__` con los argumentos `obj` y `name`.

object.__setattr__(*self*, *name*, *value*)

Es llamado cuando se intenta la asignación de atributos. Éste es llamado en lugar del mecanismo normal (p. ej. guardar el valor en el diccionario de instancias). *name* es el nombre de atributo, *value* es el valor que se le asigna.

Si **`__setattr__()`** quiere asignar a un atributo de instancia, debe llamar al método de la clase base con el mismo nombre, por ejemplo, `object.__setattr__(self, name, value)`.

Para ciertas asignaciones de atributos sensibles, lanza un evento de auditoría `object.__setattr__` con argumentos `obj`, `name`, `value`.

object.__delattr__(*self*, *name*)

Al igual que **`__setattr__()`** pero para borrado de atributos en lugar de establecerlos. Esto solo de ser implementado si `del obj.name` es significativo para el objeto.

Para ciertas eliminaciones de atributos sensibles, lanza un evento de auditoría `object.__delattr__` con argumentos `obj` y `name`.

object.__dir__(*self*)

Es llamado cuando `dir()` es llamado en el objeto. Un iterable debe ser retornado. `dir()` convierte el iterable retornado a una lista y la ordena.

3.3.2.1. Personalizando acceso a atributos de módulo

module.__getattr__()
module.__dir__()

Nombres especiales `__getattr__` y `__dir__` también pueden ser utilizados para personalizar acceso a atributos de módulo. La función `__getattr__` a nivel del módulo debe aceptar un argumento que es el nombre del atributo y retornar el valor calculado o lanzar una excepción `AttributeError`. Si un atributo no es encontrado en el objeto de módulo a través de una búsqueda normal, p. ej. `object.__getattribute__()`, entonces `__getattr__` es buscado en el módulo `__dict__` antes de lanzar una excepción `AttributeError`. Si es encontrado, es llamado con el nombre de atributo y el resultado es retornado.

La función `__dir__` no debe aceptar argumentos y retorna un iterable de cadena de caracteres que representan los nombres accesibles en el módulo. De existir, esta función anula la búsqueda estándar `dir()` en un módulo.

module.__class__

Para una personalización más precisa sobre el comportamiento del módulo (estableciendo atributos, propiedades, etc.), se puede establecer el atributo `__class__` de un objeto de módulo a una subclase de `types.ModuleType`. Por ejemplo:

```
import sys
from types import ModuleType

class VerboseModule(ModuleType):
    def __repr__(self):
        return f'Verbose {self.__name__}'
    def __setattr__(self, attr, value):
        print(f'Setting {attr}...')
        super().__setattr__(attr, value)

sys.modules[__name__].__class__ = VerboseModule
```

Nota

Definiendo un módulo `__getattr__` y estableciendo un módulo `__class__` solo afecta búsquedas que utilizan la sintaxis de acceso a atributo – acceder directamente a las globales del módulo (ya sea por código dentro del módulo, o a través de una referencia al diccionario de globales del módulo) no se ve afectado.

Distinto en la versión 3.5: El atributo de módulo `__class__` es ahora escribible.

Added in version 3.7: Atributos de módulo `__getattr__` y `__dir__`.

Ver también

PEP 562 [https://peps.python.org/pep-0562/] - Módulos _getattr_ y _dir_

Describe las funciones `__getattr__` y `__dir__` en módulos.

3.3.2.2. Implementando descriptores

The following methods only apply when an instance of the class containing the method (a so-called *descriptor* class) appears in an *owner* class (the descriptor must be in either the owner's class dictionary or in the class dictionary for one of its parents). In the examples below, «the attribute» refers to the attribute whose name is the key of the property in the owner class" `__dict__`. The `object` class itself does not implement any of these protocols.

object.__get__(*self*, *instance*, *owner=None*)

Es llamado para obtener el atributo de la clase propietaria (acceso a atributos de clase) o de una instancia de dicha clase (acceso a atributos de instancia). El argumento opcional *owner* es la clase propietaria, mientras que *instance* es la instancia a través de la cual el atributo fue accedido, o `None` cuando el atributo es accedido a través de *owner*.

Este método debe retornar el valor de atributo calculado o lanzar una excepción `AttributeError`.

PEP 252 [https://peps.python.org/pep-0252/] especifica que `__get__()` puede ser llamado con uno o dos argumentos. Los propios descriptores incorporados de Python soportan esta especificación; sin embargo, es probable que algunas herramientas de terceros tengan descriptores que requieran ambos argumentos. La propia implementación de `__getattribute__()` en Python siempre pasa ambos argumentos si son requeridos o no.

object.__set__(*self*, *instance*, *value*)

Es llamado para establecer el atributo en una instancia *instance* de la clase propietaria a un nuevo valor *value*.

Nota, agregar `__set__()` o `__delete__()` cambia el tipo de descriptor a un "descriptor de datos". Ver Invocando descriptores para más detalles.

object.__delete__(*self*, *instance*)

Es llamado para borrar el atributo en una instancia *instance* de la clase propietaria.

Las instancias de descriptores también pueden tener presente el atributo `__objclass__`:

object.__objclass__

El atributo `__objclass__` es interpretado por el módulo `inspect` como la especificación de la clase donde el objeto fue definido (establecer esto adecuadamente puede ayudar en introspección de atributos dinámicos de clases en tiempo de ejecución). Para invocables, puede indicar que una instancia de un tipo (o subclase) dado es esperado o requerido como el primer argumento posicional (por ejemplo, CPython establece este atributo para métodos independientes que son implementados en C).

3.3.2.3. Invocando descriptores

En general, un descriptor es un atributo de objeto con «comportamiento vinculante», uno cuyo acceso al atributo ha sido anulado por métodos en el protocolo del descriptor: **`__get__()`**, **`__set__()`** y **`__delete__()`**. Si alguno de esos métodos está definido para un objeto, se dice que es un descriptor.

El comportamiento por defecto para atributos de acceso es obtener (*get*), establecer (*set*) o borrar (*delete*) el atributo del diccionario del objeto. Por ejemplo, `a.x` tiene una cadena de búsqueda que comienza con `a.__dict__['x']`, luego `type(a).__dict__['x']`, y continúa por las clases base de `type(a)` excluyendo metaclases.

Sin embargo, si el valor buscado es un objeto definiendo uno de los métodos del descriptor, entonces Python puede anular el comportamiento por defecto e invocar al método del descriptor en su lugar. Dónde ocurre esto en la cadena de precedencia depende de qué métodos de descriptor fueron definidos y cómo son llamados.

El punto de inicio por invocación de descriptor es un enlace `a.x`. Cómo los argumentos son ensamblados dependen de `a`:

Llamado directto

El llamado más simple y menos común es cuando el código de usuario invoca directamente un método descriptor: `x.__get__(a)`.

Enlace de instancia

Al enlazar a una instancia de objeto, `a` es transformado en un llamado: `type(a).__dict__['x'].__get__(a, type(a))`.

Enlace de clase

Al enlazar a una clase, `A.x` es transformado en un llamado: `A.__dict__['x'].__get__(None, A)`.

Súper enlace

Una búsqueda punteada como `super(A, a).x` busca en `a.__class__.__mro__` una clase base `B` después de `A` y luego retorna `B.__dict__['x'].__get__(a, A)`. Si no es un descriptor, `x` se retorna sin cambios.

En el caso de los enlaces de instancias, la precedencia de la invocación de descriptores depende de qué métodos de descriptores están definidos. Un descriptor puede definir cualquier combinación de **`__get__()`**, **`__set__()`** y **`__delete__()`**. Si no define **`__get__()`**, acceder al atributo retornará el objeto descriptor en sí, a menos que haya un valor en el diccionario de instancia del objeto. Si el descriptor define **`__set__()`** y/o **`__delete__()`**, es un descriptor de datos; si no define ninguno de los dos, es un descriptor que no es de datos. Normalmente, los descriptores de datos definen tanto **`__get__()`** como **`__set__()`**, mientras que los descriptores que no son de datos tienen solo el método **`__get__()`**. Los descriptores de datos con **`__get__()`** y **`__set__()`** (y/o **`__delete__()`**) definidos siempre anulan una redefinición en un diccionario de instancia. Por el contrario, las instancias pueden anular los descriptores que no son datos.

Python methods (including those decorated with **`@staticmethod`** and **`@classmethod`**) are implemented as non-data descriptors. Accordingly, instances can redefine and override methods. This allows individual instances to acquire behaviors that differ from other instances of the same class.

La función **`property()`** es implementada como un descriptor de datos. Por lo tanto, las instancias no pueden anular el comportamiento de una propiedad.

3.3.2.4. _slots_

*_slots_* nos permite declarar explícitamente miembros de datos (como propiedades) y denegar la creación de **`__dict__`** y *_weakref_* (a menos que se declare explícitamente en *_slots_* o esté disponible en un padre).

El espacio ahorrado al usar `__dict__` puede ser significativo. La velocidad de búsqueda de atributos también se puede mejorar significativamente.

object._slots_

> A esta variable de clase se le puede asignar una cadena, un iterable o una secuencia de cadenas con nombres de variables utilizados por las instancias. *_slots_* reserva espacio para las variables declaradas y evita la creación automática de `__dict__` y *_weakref_* para cada instancia.

Notas sobre el uso de *_slots_*:

- Al heredar de una clase sin *_slots_*, los atributos `__dict__` y *_weakref_* de las instancias siempre estarán accesibles.
- Sin una variable `__dict__`, a las instancias no se les pueden asignar nuevas variables que no figuran en la definición *_slots_*. Los intentos de asignar un nombre de variable no listado generan `AttributeError`. Si desea una asignación dinámica de nuevas variables, agregue `'__dict__'` a la secuencia de cadenas en la declaración *_slots_*.
- Sin una variable *_weakref_* para cada instancia, las clases que definen *_slots_* no admiten `weak references` en sus instancias. Si se necesita soporte de referencia débil, agregue `'__weakref__'` a la secuencia de cadenas en la declaración *_slots_*.
- *_slots_* se implementa a nivel de clase creando descriptors para cada nombre de variable. Como resultado, los atributos de clase no se pueden utilizar para establecer valores predeterminados, por ejemplo, variables definidas por *_slots_*; de lo contrario, el atributo de clase sobrescribiría la asignación del descriptor.
- La acción de una declaración *_slots_* no se limita a la clase donde se define. *_slots_* declarado en clases principales está disponible en clases secundarias. Sin embargo, las subclases secundarias obtendrán `__dict__` y *_weakref_* a menos que las subclases también definan *_slots_* (que solo debe contener

nombres de cualquier ranura *additional*).

- Si una clase define un espacio (*slot*) también definido en una clase base, la variable de instancia definida por el espacio de la clase base es inaccesible (excepto al obtener su descriptor directamente de la clase base). Esto hace que el significado del programa sea indefinido. En el futuro se podría agregar una verificación para prevenir esto.
- **`TypeError`** se generará si se definen *_slots_* no vacíos para una clase derivada de un **`"variable-length" built-in type`** como **`int`**, **`bytes`** y **`tuple`**.
- Cualquier iterable que no sea una cadena se puede asignar a *_slots_*.
- Si se utiliza un **`dictionary`** para asignar *_slots_*, las claves del diccionario se utilizarán como nombres de ranura. Los valores del diccionario se pueden usar para proporcionar cadenas de documentos por atributo que **`inspect.getdoc()`** reconocerá y mostrará en la salida de **`help()`**.
- La asignación de **`__class__`** solo funciona si ambas clases tienen el mismo *_slots_*.
- Se puede usar Multiple inheritance con varias clases principales con ranuras, pero solo a una de las clases principales se le permite tener atributos creados por ranuras (las otras bases deben tener diseños de ranuras vacías); las infracciones generan **`TypeError`**.
- Si se utiliza un iterator para *_slots_*, entonces se crea un descriptor para cada uno de los valores del iterador. Sin embargo, el atributo *_slots_* será un iterador vacío.

3.3.3. Personalización de creación de clases

Siempre que una clase hereda de otra clase, se llama a **`__init_subclass__()`** en la clase principal. De esta forma, es posible escribir clases que cambien el comportamiento de las subclases. Esto está estrechamente relacionado con los decoradores de clases, pero mientras los decoradores de clases solo afectan la clase específica a la que se aplican, `__init_subclass__` solo se aplica a futuras subclases de la clase que define el método.

classmethod object._init_subclass_(*cls*)

Este método es llamado siempre que la clase que lo contiene sea heredada. *cls* es entonces, la nueva subclase. Si se define como un método de instancia normal, éste es convertido de manera implícita a un método de clase.

Los argumentos de palabra clave que fueron dados a una nueva clase, son pasados a la clase `__init_subclass__` del padre. Por temas de compatibilidad con otras clases que usan `__init_subclass__`, uno debería quitar los argumentos de palabra clave y pasar los otros a la clase base, como en:

```
class Philosopher:
    def __init_subclass__(cls, /, default_name, **k
        super().__init_subclass__(**kwargs)
        cls.default_name = default_name

class AustralianPhilosopher(Philosopher, default_na
    pass
```

La implementación por defecto `object.__init_subclass__` no hace nada, pero lanza un error si es llamado con cualquier argumento.

Nota

La sugerencia de metaclase `metaclass` es consumido por el resto de la maquinaria de tipos, y nunca se pasa a las implementaciones `__init_subclass__`. La clase meta actual (más que la sugerencia explícita) puede ser accedida como `type(cls)`.

Added in version 3.6.

When a class is created, **`type.__new__()`** scans the class variables and makes callbacks to those with a **`__set_name__()`** hook.

object._set_name_(*self*, *owner*, *name*)

Llamado automáticamente al momento en el que se crea la clase propietaria *owner*. El objeto es asignado a *name* en esa clase:

```
class A:
    x = C()   # Llama automáticamente: x.__set_name_
```

Si la variable de clase se asigna después de crear la clase, **`__set_name__()`** no se llamará automáticamente. Si es necesario, **`__set_name__()`** se puede llamar directamente:

```
class A:
   pass

c = C()
A.x = c                    # No se llama el enlace
c.__set_name__(A, 'x')    # Invoca manualmente el en
```

Ver Creando el objeto de clase para más detalles.

Added in version 3.6.

3.3.3.1. Metaclases

Por defecto, las clases son construidas usando **`type()`**. El cuerpo de la clase es ejecutado en un nuevo espacio de nombres y el nombre de la clase es ligado de forma local al resultado de `type(name, bases, namespace)`.

El proceso de creación de clase puede ser personalizado pasando el argumento de palabra clave `metaclass` en la línea de definición de la clase, o al heredar de una clase existente que incluya dicho argumento. En el siguiente ejemplo, ambos `MyClass` y `MySubclass` son instancias de `Meta`:

```
class Meta(type):
    pass

class MyClass(metaclass=Meta):
    pass
```

```
class MySubclass(MyClass):
    pass
```

Cualquier otro argumento de palabra clave que sea especificado en la definición de clase es pasado mediante todas las operaciones de metaclase descritas a continuación.

Cuando una definición de clase es ejecutada, los siguientes pasos ocurren:

- Entradas de la orden de resolución de método (MRU) son resueltas;
- se determina la metaclase adecuada;
- se prepara el espacio de nombres de clase;
- se ejecuta el cuerpo de la clase;
- se crea el objeto de clase.

3.3.3.2. Resolviendo entradas de la Orden de Resolución de Métodos (MRU)

object.__mro_entries__(*self*, *bases*)

Si una base que aparece en una definición de clase no es una instancia de `type`, entonces se busca un método **`__mro_entries__()`** en la base. Si se encuentra un método **`__mro_entries__()`**, la base se sustituye por el resultado de una llamada a **`__mro_entries__()`** al crear la clase. El método se llama con la tupla de bases original pasada al parámetro *bases* y debe retornar una tupla de clases que se utilizará en lugar de la base. La tupla retornada puede estar vacía: en estos casos, se ignora la base original.

Ver también

`types.resolve_bases()`

Resuelva dinámicamente bases que no sean instancias de `type`.

`types.get_original_bases()`

Recupera las «bases originales» de una clase antes de las modificaciones realizadas por `__mro_entries__()`.

PEP 560 **[https://peps.python.org/pep-0560/]**
Soporte principal para módulos de escritura y tipos genéricos.

3.3.3.3. Determinando la metaclase adecuada

La metaclase adecuada para la definición de una clase es determinada de la siguiente manera:

- si no se dan bases ni metaclases explícitas, entonces se utiliza `type()`;
- si se da una metaclase explícita y *no* es una instancia de `type()`, entonces se utiliza directamente como la metaclase;
- si se da una instancia de `type()` como la metaclase explícita, o se definen bases, entonces se utiliza la metaclase más derivada.

La metaclase más derivada es elegida de la metaclase especificada explícitamente (si existe) y de la metaclase (p. ej. `type(cls)`) de todas las clases base especificadas.

3.3.3.4. Preparando el espacio de nombres de la clase

Una vez que se ha identificado la metaclase adecuada, se prepara el espacio de nombres de la clase. Si la metaclase tiene un atributo `__prepare__`, se llama `namespace = metaclass.__prepare__(name, bases, **kwds)` (donde los argumentos de palabras clave adicionales, si los hay, provienen de la definición de clase). El método `__prepare__` debe implementarse como `classmethod`. El espacio de nombres retornado por `__prepare__` se pasa a `__new__`, pero cuando se crea el objeto de clase final, el espacio de nombres se copia en un nuevo `dict`.

Si la metaclase no tiene atributo `__prepare__`, entonces el espacio de nombres de clase es iniciado como un mapeo vacío ordenado.

Ver también

PEP 3115 [https://peps.python.org/pep-3115/] - Metaclases en Python 3000

Introduce el enlace de espacio de nombres `__prepare__`

3.3.3.5. Ejecutando el cuerpo de la clase

El cuerpo de la clase es ejecutado como `exec(body, globals(), namespace)` (aproximadamente). La diferencia clave con un llamado normal a `exec()` es que el alcance léxico permite que el cuerpo de la clase (incluyendo cualquier método) haga referencia a nombres de los alcances actuales y externos cuando la definición de clase sucede dentro de la función.

Sin embargo, aún cuando la definición de clase sucede dentro de la función, los métodos definidos dentro de la clase aún no pueden ver nombres definidos dentro del alcance de la clase. Variables de clase deben ser accedidas a través del primer parámetro de instancia o métodos de clase, o a través de la referencia al léxico implícito `__class__` descrita en la siguiente sección.

3.3.3.6. Creando el objeto de clase

Una vez que el espacio de nombres de la clase ha sido poblado al ejecutar el cuerpo de la clase, el objeto de clase es creado al llamar `metaclass(name, bases, namespace, **kwds)` (las palabras clave adicionales que se pasan aquí, son las mismas que aquellas pasadas en `__prepare__`).

Este objeto de clase es el que será referenciado por la forma sin argumentos de `super()`. `__class__` es una referencia de cierre implícita creada por el compilador si cualquier método en el cuerpo de una clase se refiere tanto a `__class__` o `super`. Esto permite que la forma sin argumentos de `super()` identifique correctamente la clase definida en base al alcance léxico, mientras la clase o instancia que fue utilizada para hacer el llamado actual es identificado en base al primer argumento que se pasa al método.

En CPython 3.6 y posterior, la celda `__class__` se pasa a la metaclase como una entrada `__classcell__` en el espacio de nombres de la clase. En caso de existir, esto debe ser propagado hacia el llamado `type.__new__` para que la clase se inicie correctamente. No hacerlo resultará en un error **`RuntimeError`** en Python 3.8.

Cuando se utiliza la metaclase por defecto **`type`**, o cualquier metaclase que finalmente llama a `type.__new__`, los siguientes pasos de personalización adicional son invocados después de crear el objeto de clase:

1. El método `type.__new__` recolecta todos los atributos en el espacio de nombres de la clase que definen un método **`__set_name__()`**;
2. Esos métodos `__set_name__` son llamados con la clase siendo definida y el nombre de ese atributo particular asignado;
3. El gancho **`__init_subclass__()`** llama al padre inmediato de la nueva clase en su orden de resolución del método.

Después de que el objeto de clase es creado, se pasa al decorador de clase incluido en su definición (si existe) y el objeto resultante es enlazado en el espacio de nombres local como la clase definida.

Cuando una nueva clase es creada por `type.__new__`, el objeto proporcionado como el parámetro de espacio de nombres es copiado a un nuevo trazado ordenado y el objeto original es descartado. La nueva copia es *envuelta* en un proxy de solo lectura, que se convierte en el atributo **`__dict__`** del objeto de clase.

Ver también

PEP 3135 [https://peps.python.org/pep-3135/] - Nuevo súper
Describe la referencia de cierre implícita `__class__`

3.3.3.7. Usos para metaclases

Los usos potenciales para metaclases son ilimitados. Algunas ideas que ya han sido exploradas incluyen enumeración, registros, revisión de interface, delegación automática, creación de propiedades automática, proxy, infraestructuras, y bloqueo/sincronización automática de recursos.

3.3.4. Personalizando revisiones de instancia y subclase

Los siguientes métodos son utilizados para anular el comportamiento por defecto de las funciones incorporadas `isinstance()` y `issubclass()`.

En particular, la metaclase `abc.ABCMeta` implementa estos métodos para permitir la adición de Clases Base Abstractas (ABCs, por su nombre en inglés *Abstract Base Classes*) como "clases base virtuales" a cualquier clase o tipo (incluyendo tipos incorporados), incluyendo otros ABCs.

type._instancecheck_(*self*, *instance*)

> Retorna *true* si la instancia *instance* debe ser considerada una instancia (directa o indirecta) de clase *class*. De ser definida, es llamado para implementar `isinstance(instance, class)`.

type._subclasscheck_(*self*, *subclass*)

> Retorna *true* si la subclase *subclass* debe ser considerada una subclase (directa o indirecta) de clase *class*. De ser definida, es llamado para implementar `issubclass(subclass, class)`.

Tome en cuenta que estos métodos son buscados en el tipo (metaclase) de una clase. No pueden ser definidos como métodos de clase en la clase actual. Esto es consistente con la búsqueda de métodos especiales que son llamados en instancias, solo en este caso la instancia es por sí misma una clase.

Ver también

PEP 3119 [https://peps.python.org/pep-3119/] - Introducción a Clases Base Abstractas (*Abstract Base Classes*)

Incluye la especificación para personalizar el comportamiento de **`isinstance()`** y **`issubclass()`** a través de **`__instancecheck__()`** y **`__subclasscheck__()`**, con motivación para esta funcionalidad en el contexto de agregar Clases Base Abstractas (ver el módulo **`abc`**) al lenguaje.

3.3.5. Emulando tipos genéricos

Cuando se usa type annotations, a menudo es útil *parameterize* a generic type usando la notación de corchetes de Python. Por ejemplo, la anotación `list[int]` podría usarse para indicar un **`list`** en el que todos los elementos son del tipo **`int`**.

Ver también

PEP 484 [https://peps.python.org/pep-0484/] - Sugerencias de tipo
Presentamos el marco de trabajo de Python para las anotaciones de tipo

Generic Alias Types
Documentación para objetos que representan clases genéricas parametrizadas

Genéricos, user-defined generics y `typing.Generic`
Documentación sobre cómo implementar clases genéricas que se pueden parametrizar en tiempo de ejecución y que los verificadores de tipos estáticos pueden entender.

Una clase *generally* solo se puede parametrizar si define el método de clase especial `__class_getitem__()`.

classmethod object.__class_getitem__(*cls*, *key*)

Retornar un objeto representando la especialización de una clase genérica por argumentos de tipo encontrados en *key*.

When defined on a class, `__class_getitem__()` is automatically a class method. As such, there is no need for it to be decorated with **`@classmethod`** when it is defined.

3.3.5.1. El propósito de _class_getitem_

El propósito de `__class_getitem__()` es permitir la parametrización en tiempo de ejecución de clases genéricas de biblioteca estándar para aplicar type hints a estas clases con mayor facilidad.

Para implementar clases genéricas personalizadas que se puedan parametrizar en tiempo de ejecución y que los verificadores de tipos estáticos las entiendan, los usuarios deben heredar de una clase de biblioteca estándar que ya implementa `__class_getitem__()`, o heredar de `typing.Generic`, que tiene su propia implementación de `__class_getitem__()`.

Es posible que los verificadores de tipos de terceros, como mypy, no entiendan las implementaciones personalizadas de `__class_getitem__()` en clases definidas fuera de la biblioteca estándar. Se desaconseja el uso de `__class_getitem__()` en cualquier clase para fines distintos a la sugerencia de tipo.

3.3.5.2. _class_getitem_ frente a _getitem_

Por lo general, el subscription de un objeto que usa corchetes llamará al método de instancia `__getitem__()` definido en la clase del objeto. Sin embargo, si el objeto que se suscribe es en sí mismo una clase, se puede llamar al método de clase `__class_getitem__()` en su lugar. `__class_getitem__()` debería retornar un objeto GenericAlias si está definido correctamente.

Presentado con el expression `obj[x]`, el intérprete de Python sigue un proceso similar al siguiente para decidir si se debe llamar a `__getitem__()` o `__class_getitem__()`:

```
from inspect import isclass

def subscribe(obj, x):
    """Retorna el resultado de la expresión 'obj[x]'"""

    class_of_obj = type(obj)
    # Si la clase de obj define __getitem__,
    # llama class_of_obj.__getitem__(obj, x)
    if hasattr(class_of_obj, '__getitem__'):
        return class_of_obj.__getitem__(obj, x)

    # De lo contrario, si obj es una clase y define __cl
    # llama obj.__class_getitem__(x)
    elif isclass(obj) and hasattr(obj, '__class_getitem_
        return obj.__class_getitem__(x)

    # De lo contrario, lanza una excepción
    else:
        raise TypeError(
            f"'{class_of_obj.__name__}' object is not su
        )
```

En Python, todas las clases son en sí mismas instancias de otras clases. La clase de una clase se conoce como metaclass de esa clase, y la mayoría de las clases tienen la clase **`type`** como su metaclase. **`type`** no define **`__getitem__()`**, lo que significa que expresiones como `list[int]`, `dict[str, float]` y `tuple[str, bytes]` dan como resultado que se llame a **`__class_getitem__()`**:

```
>>> # list tiene la clase "type" como su metaclase, como
>>> type(list)
<class 'type'>
>>> type(dict) == type(list) == type(tuple) == type(str)
True
>>> # "list[int]" llama "list.__class_getitem__(int)"
>>> list[int]
list[int]
>>> # list.__class_getitem__ retorna un objeto GenericAl
>>> type(list[int])
<class 'types.GenericAlias'>
```

Sin embargo, si una clase tiene una metaclase personalizada que define **`__getitem__()`**, la suscripción de la clase puede generar un comportamiento diferente. Un ejemplo de esto se puede encontrar en el módulo `enum`:

```
>>> from enum import Enum
>>> class Menu(Enum):
...     """Un menú de desayuno"""
...     SPAM = 'spam'
...     BACON = 'bacon'
...
>>> # Las clases de enumeración tienen una metaclase per
>>> type(Menu)
<class 'enum.EnumMeta'>
>>> # EnumMeta define __getitem__,
>>> # por lo que no se llama __class_getitem__,
>>> # y el resultado no es un objeto GenericAlias:
>>> Menu['SPAM']
<Menu.SPAM: 'spam'>
>>> type(Menu['SPAM'])
<enum 'Menu'>
```

Ver también

PEP 560 [https://peps.python.org/pep-0560/]: soporte principal para módulo de escritura y tipos genéricos

Presentamos `__class_getitem__()` y describimos cuándo un subscription da como resultado que se llame a `__class_getitem__()` en lugar de `__getitem__()`

3.3.6. Emulando objetos que se pueden llamar

object.__call__(*self*[, *args...*])

Called when the instance is «called» as a function; if this method is defined, `x(arg1, arg2, ...)` roughly translates to `type(x).__call__(x, arg1, ...)`. The `object` class itself does not provide this method.

3.3.7. Emulando tipos de contenedores

The following methods can be defined to implement container objects. None of them are provided by the `object` class itself. Containers usually are sequences (such as `lists` or `tuples`) or mappings (like dictionaries), but can represent other containers as well. The first set of methods is used either to emulate a sequence or to emulate a mapping; the difference is that for a sequence, the allowable keys should be the integers *k* for which `0 <= k < N` where *N* is the length of the sequence, or `slice` objects, which define a range of items. It is also recommended that mappings provide the methods `keys()`, `values()`, `items()`, `get()`, `clear()`, `setdefault()`, `pop()`, `popitem()`, `copy()`, and `update()` behaving similar to those for Python's standard `dictionary` objects. The `collections.abc` module provides a `MutableMapping` abstract base class to help create those methods from a base set of `__getitem__()`, `__setitem__()`, `__delitem__()`, and `keys()`.

Mutable sequences should provide methods `append()`, `clear()`, `count()`, `extend()`, `index()`, `insert()`, `pop()`, `remove()`, and `reverse()`, like Python standard `list` objects. Finally, sequence types should implement addition (meaning concatenation) and multiplication (meaning repetition) by defining the methods `__add__()`, `__radd__()`, `__iadd__()`, `__mul__()`, `__rmul__()` and `__imul__()` described below; they should not define other numerical operators.

It is recommended that both mappings and sequences implement the `__contains__()` method to allow efficient use of the `in` operator; for mappings, `in` should search the mapping's keys; for sequences, it should search through the values. It is further recommended that both mappings and sequences implement the `__iter__()` method to allow efficient iteration through the container; for mappings, `__iter__()` should iterate through the object's keys; for sequences, it should iterate through the values.

object._len_(*self*)

> Llamado para implementar la función incorporada `len()`. Debería retornar la longitud del objeto, un número entero `>=` 0. Además, un objeto que no define un método `__bool__()` y cuyo método `__len__()` retorna cero se considera falso

en un contexto booleano.

En CPython, se requiere que la longitud sea como máximo `sys.maxsize`. Si la longitud es mayor que `sys.maxsize`, algunas funciones (como `len()`) pueden generar `OverflowError`. Para evitar que se genere `OverflowError` mediante pruebas de valor de verdad, un objeto debe definir un método `__bool__()`.

object.__length_hint__(*self*)

Es llamado para implementar `operator.length_hint()`. Debe retornar una longitud estimada para el objeto (que puede ser mayor o menor que la longitud actual). La longitud debe ser un entero `>=` 0. El valor de retorno también debe ser `NotImplemented`, el cual es tratado de igual forma a que si el método `__length_hint__` no existiera en absoluto. Este método es puramente una optimización y nunca es requerido para precisión.

Added in version 3.4.

object.__getitem__(*self*, *subscript*)

Called to implement *subscription*, that is, `self[subscript]`. See Subscriptions and slicings for details on the syntax.

There are two types of built-in objects that support subscription via `__getitem__()`:

- **sequences**, where *subscript* (also called index) should be an integer or a `slice` object. See the sequence documentation for the expected behavior, including handling `slice` objects and negative indices.
- **mappings**, where *subscript* is also called the key. See mapping documentation for the expected behavior.

If *subscript* is of an inappropriate type, `__getitem__()` should raise `TypeError`. If *subscript* has an inappropriate value, `__getitem__()` should raise an `LookupError` or one of its subclasses (`IndexError` for sequences; `KeyError` for mappings).

Nota

Slicing is handled by **`__getitem__()`**, **`__setitem__()`**, and **`__delitem__()`**. A call like

```
a[1:2] = b
```

es traducido a

```
a[slice(1, 2, None)] = b
```

and so forth. Missing slice items are always filled in with `None`.

Nota

The sequence iteration protocol (used, for example, in **`for`** loops), expects that an **`IndexError`** will be raised for illegal indexes to allow proper detection of the end of a sequence.

Nota

When subscripting a *class*, the special class method **`__class_getitem__()`** may be called instead of **`__getitem__()`**. See _class_getitem_ frente a _getitem_ for more details.

object.__setitem__(*self*, *key*, *value*)

Es llamado para implementar la asignación a `self[key]`. Lo mismo con respecto a **`__getitem__()`**. Esto solo debe ser implementado para mapeos si los objetos permiten cambios a los valores de las llaves, o si nuevas llaves pueden ser añadidas, o para secuencias si los elementos pueden ser reemplazados. Las mismas excepciones deben ser lanzadas para valores de *key* inadecuados con respecto al método

`__getitem__()`.

object._delitem_(*self*, *key*)

Es llamado para implementar el borrado de `self[key]`. Lo mismo con respecto a **`__getitem__()`**. Esto solo debe ser implementado para mapeos si los objetos permiten el borrado de llaves, o para secuencias si los elementos pueden ser eliminados de la secuencia. Las mismas excepciones deben ser lanzadas por valores de *key* inapropiados con respecto al método **`__getitem__()`**.

object._missing_(*self*, *key*)

Es llamado por **`dict.__getitem__()`** para implementar `self[key]` para subclases de diccionarios cuando la llave no se encuentra en el diccionario.

object._iter_(*self*)

Se llama a este método cuando se requiere un iterator para un contenedor. Este método debería retornar un nuevo objeto iterador que pueda iterar sobre todos los objetos del contenedor. Para las asignaciones, debe iterar sobre las claves del contenedor.

object._reversed_(*self*)

Es llamado (si existe) por la función incorporada **`reversed()`** para implementar una interacción invertida. Debe retornar un nuevo objeto iterador que itere sobre todos los objetos en el contenedor en orden inverso.

Si el método **`__reversed__()`** no es proporcionado, la función incorporada **`reversed()`** recurrirá a utilizar el protocolo de secuencia (**`__len__()`** y **`__getitem__()`**). Objetos que permiten el protocolo de secuencia deben únicamente proporcionar **`__reversed__()`** si no pueden proporcionar una implementación que sea más eficiente que la proporcionada por **`reversed()`**.

Los operadores de prueba de pertenencia (**`in`** and **`not in`**) son normalmente implementados como una iteración sobre un contenedor. Sin embargo, los objetos de contenedor pueden proveer el siguiente método especial con una implementación más eficiente, que tampoco requiere que el objeto sea iterable.

object.__contains__(*self*, *item*)

> Es llamado para implementar operadores de prueba de pertenencia. Deben retornar *true* si *item* se encuentra en *self*, de lo contrario *false*. Para objetos de mapeo, estos debe considerar las llaves del mapeo en lugar de los valores o los pares de llave-valor.
>
> Para objetos que no definen `__contains__()`, la prueba de pertenencia primero intenta la iteración a través de `__iter__()`, y luego el antiguo protocolo de iteración de secuencia a través de `__getitem__()`, ver esta sección en la referencia del lenguaje.

3.3.8. Emulando tipos numéricos

Los siguientes métodos pueden ser definidos para emular objetos numéricos. Métodos que corresponden a operaciones que no son permitidas por el número particular implementado (por ejemplo, operaciones bit a bit para números no enteros) se deben dejar sin definir.

object.__add__(*self*, *other*)
object.__sub__(*self*, *other*)
object.__mul__(*self*, *other*)
object.__matmul__(*self*, *other*)
object.__truediv__(*self*, *other*)
object.__floordiv__(*self*, *other*)
object.__mod__(*self*, *other*)
object.__divmod__(*self*, *other*)
object.__pow__(*self*, *other*[, *modulo*])
object.__lshift__(*self*, *other*)
object.__rshift__(*self*, *other*)

object.__and__(*self*, *other*)
object.__xor__(*self*, *other*)
object.__or__(*self*, *other*)

These methods are called to implement the binary arithmetic operations (`+`, `-`, `*`, `@`, `/`, `//`, `%`, `divmod()`, `pow()`, `**`, `<<`, `>>`, `&`, `^`, `|`). For instance, to evaluate the expression `x + y`, where *x* is an instance of a class that has an `__add__()` method, `type(x).__add__(x, y)` is called. The `__divmod__()` method should be the equivalent to using `__floordiv__()` and `__mod__()`; it should not be related to `__truediv__()`. Note that `__pow__()` should be defined to accept an optional third argument if the three-argument version of the built-in `pow()` function is to be supported.

Si alguno de esos métodos no permiten la operación con los argumentos suministrados, debe retornar `NotImplemented`.

object.__radd__(*self*, *other*)
object.__rsub__(*self*, *other*)
object.__rmul__(*self*, *other*)
object.__rmatmul__(*self*, *other*)
object.__rtruediv__(*self*, *other*)
object.__rfloordiv__(*self*, *other*)
object.__rmod__(*self*, *other*)
object.__rdivmod__(*self*, *other*)
object.__rpow__(*self*, *other*[, *modulo*])
object.__rlshift__(*self*, *other*)
object.__rrshift__(*self*, *other*)
object.__rand__(*self*, *other*)
object.__rxor__(*self*, *other*)
object.__ror__(*self*, *other*)

These methods are called to implement the binary arithmetic operations (`+`, `-`, `*`, `@`, `/`, `//`, `%`, `divmod()`, `pow()`, `**`, `<<`, `>>`, `&`, `^`, `|`) with reflected (swapped) operands. These functions are only called if the operands are of different types, when the left operand does not support the corresponding operation [3], or the right operand's class is derived from the left operand's class. [4] For instance, to evaluate the expression `x - y`, where *y* is an instance of a class that has an **`__rsub__()`** method, `type(y).__rsub__(y, x)` is called if `type(x).__sub__(x, y)` returns **`NotImplemented`** or `type(y)` is a subclass of `type(x)`. [5]

Note that **`__rpow__()`** should be defined to accept an optional third argument if the three-argument version of the built-in **`pow()`** function is to be supported.

Distinto en la versión 3.14: Three-argument **`pow()`** now try calling **`__rpow__()`** if necessary. Previously it was only called in two-argument **`pow()`** and the binary power operator.

> **Nota**
>
> Si el tipo del operando de la derecha es una subclase del tipo del operando de la izquierda y esa subclase proporciona el método reflejado para la operación, este método será llamado antes del método no reflejado del operando izquierdo. Este comportamiento permite que las subclases anulen las operaciones de sus predecesores.

object.__iadd__(*self*, *other*)
object.__isub__(*self*, *other*)
object.__imul__(*self*, *other*)
object.__imatmul__(*self*, *other*)
object.__itruediv__(*self*, *other*)
object.__ifloordiv__(*self*, *other*)
object.__imod__(*self*, *other*)
object.__ipow__(*self*, *other*[, *modulo*])
object.__ilshift__(*self*, *other*)
object.__irshift__(*self*, *other*)

object.__iand__(*self*, *other*)
object.__ixor__(*self*, *other*)
object.__ior__(*self*, *other*)

Estos métodos son llamados para implementar las asignaciones aritméticas aumentadas (`+=`, `-=`, `*=`, `@=`, `/=`, `//=`, `%=`, `**=`, `<<=`, `>>=`, `&=`, `^=`, `|=`). Estos métodos deben intentar realizar la operación *in-place* (modificando *self*) y retornar el resultado (que puede, pero no tiene que ser *self*). Si un método específico no es definido o si ese método retorna **`NotImplemented`**, la asignación aumentada regresa a los métodos normales. Por ejemplo, si *x* es la instancia de una clase con el método **`__iadd__()`**, `x += y` es equivalente a `x = x.__iadd__(y)`. Si **`__iadd__()`** no existe o si `x.__iadd__(y)` retorna **`NotImplemented`**, `x.__add__(y)` y `y.__radd__(x)` se consideran al igual que con la evaluación de `x + y`. En ciertas situaciones, asignaciones aumentadas pueden resultar en errores no esperados (ver ¿Por qué hacer lo siguiente, a_tuple[i] += ['item'], lanza una excepción cuando la suma funciona?), pero este comportamiento es en realidad parte del modelo de datos.

object.__neg__(*self*)
object.__pos__(*self*)
object.__abs__(*self*)
object.__invert__(*self*)

Es llamado para implementar las operaciones aritméticas unarias (`-`, `+`, **`abs()`** and `~`).

object.__complex__(*self*)
object.__int__(*self*)
object.__float__(*self*)

Es llamado para implementar las funciones incorporadas **`complex()`**, **`int()`** y **`float()`**. Debe retornar un valor del tipo apropiado.

object.__index__(*self*)

Es llamado para implementar `operator.index()`, y cuando sea que Python necesite convertir sin pérdidas el objeto numérico a un objeto entero (tal como en la segmentación o *slicing*, o las funciones incorporadas `bin()`, `hex()` y `oct()`). La presencia de este método indica que el objeto numérico es un tipo entero. Debe retornar un entero.

Si `__int__()`, `__float__()` y `__complex__()` no son definidos, entonces todas las funciones incorporadas correspondientes `int()`, `float()` y `complex()` vuelven a `__index__()`.

object.__round__(*self*[, *ndigits*])
object.__trunc__(*self*)
object.__floor__(*self*)
object.__ceil__(*self*)

Es llamado para implementar la función incorporada `round()` y las funciones `math trunc()`, `floor()` y `ceil()`. A menos que *ndigits* sea pasado a `__round__()` todos estos métodos deben retornar el valor del objeto truncado a `Integral` (normalmente `int`).

Distinto en la versión 3.14: `int()` no longer delegates to the `__trunc__()` method.

3.3.9. Gestores de Contexto en la Declaración *with*

Un *context manager* es un objeto que define el contexto en tiempo de ejecución a ser establecido cuando se ejecuta una declaración `with`. El gestor de contexto maneja la entrada y la salida del contexto en tiempo de ejecución deseado para la ejecución del bloque de código. Los gestores de contexto son normalmente invocados utilizando la declaración `with` (descritos en la sección La sentencia with), pero también pueden ser utilizados al invocar directamente sus métodos.

Usos típicos de los gestores de contexto incluyen guardar y restablecer diversos tipos de declaraciones globales, bloquear y desbloquear recursos, cerrar archivos abiertos, etc.

For more information on context managers, see Tipos gestores de contexto. The `object` class itself does not provide the context manager methods.

object.__enter__(*self*)

Ingresa al contexto en tiempo de ejecución relacionado con este objeto. La declaración `with` ligará el valor de retorno de este método al objetivo especificado en cláusula `as` de la declaración, en caso de existir.

object.__exit__(*self*, *exc_type*, *exc_value*, *traceback*)

Sale del contexto en tiempo de ejecución relacionado a este objeto. Los parámetros describen la excepción que causa la salida del contexto. Si éste se termina sin excepción, los tres argumentos serán `None`.

Si se proporciona una excepción, y el método desea eliminarla (por ejemplo, prevenir que sea propagada), debe retornar un valor verdadero. De lo contrario, la excepción será procesada de forma normal al salir de este método.

Se debe tomar en cuenta que los métodos `__exit__()` no deben lanzar de nuevo la excepción que se pasa; esto es la responsabilidad de quien hace el llamado.

Ver también

PEP 343 [https://peps.python.org/pep-0343/] - La declaración "with"

La especificación, el antecedente, y los ejemplos para la declaración de Python `with`.

3.3.10. Personalización de argumentos posicionales en la coincidencia de patrones de clase

Cuando se utiliza un nombre de clase en un patrón, los argumentos posicionales en el patrón no están permitidos de forma predeterminada, es decir, `case MyClass(x, y)` normalmente no es válido sin un soporte especial en `MyClass`. Para poder utilizar ese tipo de patrón, la clase necesita definir un atributto *__match_args__*.

object.__match_args__

> A esta variable de clase se le puede asignar una tupla de cadenas. Cuando esta clase se utiliza en un patrón de clase con argumentos posicionales, cada argumento posicional se convertirá en un argumento de palabra clave, utilizando el valor correspondiente en *__match_args__* como palabra clave. La ausencia de este atributo es equivalente a establecerlo en `()`.

Por ejemplo, si `MyClass.__match_args__` es `("left", "center", "right")` eso significa que `case MyClass(x, y)` es equivalente a `case MyClass(left=x, center=y)`. Ten en cuenta que el número de argumentos en el patrón debe ser menor o igual que el número de elementos en *__match_args__*; si es más grande, el intento de coincidencia de patrón producirá un **`TypeError`**.

Added in version 3.10.

Ver también

PEP 634 [https://peps.python.org/pep-0634/] - Coincidencia de patrones estructurales

La especificación para la declaración `match` de Python.

3.3.11. Emulando tipos de búfer

buffer protocol proporciona una forma para que los objetos Python expongan un acceso eficiente a una matriz de memoria de bajo nivel. Este protocolo se implementa mediante tipos integrados como **`bytes`** y **`memoryview`**, y bibliotecas de terceros pueden definir tipos de búfer adicionales.

Si bien los tipos de búfer generalmente se implementan en C, también es posible implementar el protocolo en Python.

object.__buffer__(*self*, *flags*)

Se llama cuando se solicita un búfer desde *self* (por ejemplo, por el constructor `memoryview`). El argumento *flags* es un número entero que representa el tipo de búfer solicitado y afecta, por ejemplo, si el búfer retornado es de solo lectura o de escritura. `inspect.BufferFlags` proporciona una manera conveniente de interpretar las banderas. El método debe retornar un objeto `memoryview`.

Thread safety: In free-threaded Python, implementations must manage any internal export counter using atomic operations. The method must be safe to call concurrently from multiple threads, and the returned buffer's underlying data must remain valid until the corresponding `__release_buffer__()` call completes. See Thread safety for memoryview objects for details.

object.__release_buffer__(*self*, *buffer*)

Called when a buffer is no longer needed. The *buffer* argument is a `memoryview` object that was previously returned by `__buffer__()`. The method must release any resources associated with the buffer. This method should return `None`.

Thread safety: In free-threaded Python, any export counter decrement must use atomic operations. Resource cleanup must be thread-safe, as the final release may race with concurrent releases from other threads.

Buffer objects that do not need to perform any cleanup are not required to implement this method.

Added in version 3.12.

Ver también

PEP 688 [https://peps.python.org/pep-0688/]: hacer accesible el protocolo de búfer en Python

Presenta los métodos Python `__buffer__` y

`__release_buffer__`.

`collections.abc.Buffer`

ABC para tipos de buffer.

3.3.12. Annotations

Functions, classes, and modules may contain annotations, which are a way to associate information (usually type hints) with a symbol.

object.__annotations__

This attribute contains the annotations for an object. It is lazily evaluated, so accessing the attribute may execute arbitrary code and raise exceptions. If evaluation is successful, the attribute is set to a dictionary mapping from variable names to annotations.

Distinto en la versión 3.14: Annotations are now lazily evaluated.

object.__annotate__(*format*)

An annotate function. Returns a new dictionary object mapping attribute/parameter names to their annotation values.

Takes a format parameter specifying the format in which annotations values should be provided. It must be a member of the `annotationlib.Format` enum, or an integer with a value corresponding to a member of the enum.

If an annotate function doesn't support the requested format, it must raise `NotImplementedError`. Annotate functions must always support `VALUE` format; they must not raise `NotImplementedError()` when called with this format.

When called with `VALUE` format, an annotate function may raise `NameError`; it must not raise `NameError` when called requesting any other format.

If an object does not have any annotations, `__annotate__`

should preferably be set to `None` (it can't be deleted), rather than set to a function that returns an empty dict.

Added in version 3.14.

Ver también

PEP 649 [https://peps.python.org/pep-0649/] — Deferred evaluation of annotation using descriptors

Introduces lazy evaluation of annotations and the `__annotate__` function.

3.3.13. Búsqueda de método especial

Para clases personalizadas, invocaciones implícitas de métodos especiales solo están garantizados para trabajar correctamente si son definidos en un tipo de objeto, no en el diccionario de instancia del objeto. Ese comportamiento es la razón por la que el siguiente código lanza una excepción:

```
>>> class C:
...     pass
...
>>> c = C()
>>> c.__len__ = lambda: 5
>>> len(c)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'C' has no len()
```

La razón detrás de este comportamiento radica en una serie de métodos especiales, como **`__hash__()`** y **`__repr__()`**, que implementan todos los objetos, incluidos los objetos de tipo. Si la búsqueda implícita de estos métodos utilizara el proceso de búsqueda convencional, fallarían cuando se invocaran en el objeto de tipo mismo:

```
>>> 1 .__hash__() == hash(1)
True
```

```
>>> int.__hash__() == hash(int)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: descriptor '__hash__' of 'int' object needs a
```

Intentar invocar de manera incorrecta el método no ligado de una clase de esta forma a veces es denominado como 'confusión de metaclase', y se evita sobrepasando la instancia al buscar métodos especiales:

```
>>> type(1).__hash__(1) == hash(1)
True
>>> type(int).__hash__(int) == hash(int)
True
```

Además de omitir cualquier atributo de instancia en aras de la corrección, la búsqueda implícita de métodos especiales generalmente también omite el método `__getattribute__()` incluso de la metaclase del objeto:

```
>>> class Meta(type):
...     def __getattribute__(*args):
...         print("Metaclass getattribute invoked")
...         return type.__getattribute__(*args)
...
>>> class C(object, metaclass=Meta):
...     def __len__(self):
...         return 10
...     def __getattribute__(*args):
...         print("Class getattribute invoked")
...         return object.__getattribute__(*args)
...
>>> c = C()
>>> c.__len__()                 # Búsqueda explícita a t
Class getattribute invoked
10
>>> type(c).__len__(c)          # Búsqueda explícita a t
Metaclass getattribute invoked
10
```

```
>>> len(c)                    # Búsqueda implícita
10
```

Omitir la maquinaria `__getattribute__()` de esta manera proporciona un margen significativo para optimizar la velocidad dentro del intérprete, a costa de cierta flexibilidad en el manejo de métodos especiales (el método especial *must* debe configurarse en el propio objeto de clase para que el intérprete lo invoque consistentemente).).

3.4. Corrutinas

3.4.1. Objetos esperables

Un objeto awaitable generalmente implementa un método `__await__()`. Coroutine objects retornado por las funciones `async def` están a la espera.

Nota

Los objetos generator iterator retornados por generadores decorados con `types.coroutine()` también están a la espera, pero no implementan `__await__()`.

object.__await__(*self*)

Must return an iterator. Should be used to implement awaitable objects. For instance, `asyncio.Future` implements this method to be compatible with the `await` expression. The `object` class itself is not awaitable and does not provide this method.

Nota

El lenguaje no impone ninguna restricción sobre el tipo o valor de los objetos generados por el iterador retornado por `__await__`, ya que esto es específico de la implementación del marco de ejecución asincrónica (por ejemplo, `asyncio`) que administrará el objeto awaitable.

Added in version 3.5.

Ver también

PEP 492 [https://peps.python.org/pep-0492/] para información adicional sobre objetos esperables.

3.4.2. Objetos de corrutina

Coroutine objects son objetos awaitable. La ejecución de una corrutina se puede controlar llamando a `__await__()` e iterando sobre el resultado. Cuando la rutina termina de ejecutarse y regresa, el iterador genera `StopIteration` y el atributo `value` de la excepción contiene el valor de retorno. Si la rutina genera una excepción, el iterador la propaga. Las corrutinas no deberían generar directamente excepciones `StopIteration` no controladas.

Las corrutinas también tienen los métodos mencionados a continuación, los cuales son análogos a los de los generadores. (ver Métodos generador-iterador). Sin embargo, a diferencia de los generadores, las corrutinas no soportan directamente iteración.

Distinto en la versión 3.5.2: Es un error `RuntimeError` esperar a una corrutina más de una vez.

coroutine.send(*value*)

> Inicia o reanuda la ejecución de la corrutina. Si *value* es `None`, esto equivale a avanzar el iterador retornado por `__await__()`. Si *value* no es `None`, este método delega en el método `send()` del iterador que provocó la suspensión de la rutina. El resultado (valor de retorno, `StopIteration` u otra excepción) es el mismo que cuando se itera sobre el valor de retorno `__await__()`, descrito anteriormente.

coroutine.throw(*value*)

coroutine.throw(*type*[, *value*[, *traceback*]])

Genera la excepción especificada en la corrutina. Este método delega al método `throw()` del iterador que provocó la suspensión de la rutina, si tiene dicho método. En caso contrario, la excepción se plantea en el punto de suspensión. El resultado (valor de retorno, `StopIteration` u otra excepción) es el mismo que cuando se itera sobre el valor de retorno `__await__()`, descrito anteriormente. Si la excepción no queda atrapada en la rutina, se propaga de nuevo a la persona que llama.

Distinto en la versión 3.12: La segunda firma (type[, value[, traceback]]) está obsoleta y puede eliminarse en una versión futura de Python.

coroutine.close()

Causa que la corrutina misma se borre a sí misma y termine su ejecución. Si la corrutina es suspendida, este método primero delega a `close()`, si existe, del iterador que causó la suspensión de la corrutina. Luego lanza una excepción `GeneratorExit` en el punto de suspensión, causando que la corrutina se borre a sí misma. Finalmente, la corrutina es marcada como completada, aún si nunca inició.

Objetos de corrutina son cerrados automáticamente utilizando el proceso anterior cuando están a punto de ser destruidos.

3.4.3. Iteradores asíncronos

Un *iterador asíncrono* puede llamar código asíncrono en su método `__anext__`.

Iteradores asíncronos pueden ser utilizados en la declaración `async for`.

The `object` class itself does not provide these methods.

object.__aiter__(*self*)

Debe retornar un objeto de *iterador asíncrono*.

object.__anext__(*self*)

Debe retornar un *esperable* (awaitable) resultante en el siguiente valor del iterador. Debe levantar una excepción **`StopAsyncIteration`** cuando la iteración termina.

Un ejemplo de objeto iterable asíncrono:

```
class Reader:
    async def readline(self):
        ...

    def __aiter__(self):
        return self

    async def __anext__(self):
        val = await self.readline()
        if val == b'':
            raise StopAsyncIteration
        return val
```

Added in version 3.5.

Distinto en la versión 3.7: Antes de Python 3.7, **`__aiter__()`** podía retornar un *awaitable* que se resolvería en un asynchronous iterator.

A partir de Python 3.7, **`__aiter__()`** debe retornar un objeto iterador asincrónico. Retornar cualquier otra cosa resultará en un error **`TypeError`**.

3.4.4. Gestores de contexto asíncronos

Un *gestor de contexto asíncrono* es un *gestor de contexto* que puede suspender la ejecución en sus métodos `__aenter__` y `__aexit__`.

Los gestores de contexto asíncronos pueden ser utilizados en una declaración **`async with`**.

The **`object`** class itself does not provide these methods.

object.__aenter__(*self*)

Semánticamente similar a `__enter__()`, siendo la única diferencia que debe retornar un *esperable*.

object.__aexit__(*self*, *exc_type*, *exc_value*, *traceback*)

Semánticamente similar a `__exit__()`, siendo la única diferencia que debe retornar un *esperable*.

Un ejemplo de una clase de gestor de contexto asíncrono:

```
class AsyncContextManager:
    async def __aenter__(self):
        await log('entering context')

    async def __aexit__(self, exc_type, exc, tb):
        await log('exiting context')
```

Added in version 3.5.

Notas a pie de página

Es posible cambiar en algunos casos un tipo de objeto bajo ciertas circunstancias controladas. Generalmente no es buena idea, ya que esto puede llevar a un comportamiento bastante extraño de no ser tratado correctamente.

Los métodos `__hash__()`, `__iter__()`, `__reversed__()`, `__contains__()`, `__class_getitem__()` y `__fspath__()` tienen un manejo especial para esto. Otros seguirán lanzando un `TypeError`, pero pueden hacerlo confiando en el comportamiento de que `None` no es invocable.

«No soporta» aquí significa que la clase no tiene tal método, o el método retorna `NotImplemented`. No establecer el método a `None` si se quiere forzar el retroceso al método reflejado del operando correcto—eso, por el contrario, tendrá un efecto opuesto de *bloquear* explícitamente dicho retroceso.

For operands of the same type, it is assumed that if the non-reflected method (such as `__add__()`) fails then the operation is not supported, which is why the reflected method is not called.

If the right operand's type is a subclass of the left operand's type, the reflected method having precedence allows subclasses to override their ancestors" operations.

[1]

4. Modelo de ejecución

4.1. Estructura de un programa

Un programa de Python se construye a partir de bloques de código. Un *block* es una parte del texto del programa Python que se ejecuta como una unidad. Los siguientes son bloques: un módulo, un cuerpo de función y una definición de clase. Cada comando escrito de forma interactiva es un bloque. Un archivo de secuencia de comandos (un archivo proporcionado como entrada estándar al intérprete o especificado como un argumento de línea de comando para el intérprete) es un bloque de código. Un comando de secuencia de comandos (un comando especificado en la línea de comandos del intérprete con la opción: `-c`) es un bloque de código. Un módulo que se ejecuta como un script de nivel superior (como módulo `__main__`) desde la línea de comando usando un argumento `-m` también es un bloque de código. El argumento de cadena pasado a las funciones integradas `eval()` y `exec()` es un bloque de código.

Un bloque de código se ejecuta en un *execution frame*. Un marco contiene alguna información administrativa (que se usa para depuración) y determina dónde y cómo continuará la ejecución una vez que el bloque de código se haya completado.

4.2. Nombres y vínculos

4.2.1. Vinculación de nombres

Los *Names* refieren a objetos. Los nombres se introducen por las operaciones de vinculación de nombre (*name binding operations*).

Las siguientes construcciones enlazan nombres:

- parámetros formales de las funciones,

- definiciones de clase,
- definiciones de funciones,
- expresiones de asignación,
- targets que son identificadores si aparecen en una asignación:
    - cabecera del bucle **`for`**,
    - después de **`as`** en una declaración **`with`**, cláusula **`except`**, cláusula **`except*`**, o en el patrón as en la concordancia de patrones estructurales,
    - en un patrón de captura en la concordancia de patrones estructurales
- declaraciones **`import`**.
- declaraciones **`type`**.
- listas tipo de parámetros.

La declaración **`import`** de la forma `from ... import *` vincula todos los nombres definidos en el módulo importado, excepto los que empiezan por guión bajo. Esta forma sólo puede utilizarse a nivel de módulo.

Un objetivo que aparece en una sentencia **`del`** también se considera vinculado para este propósito (aunque la semántica real es desvincular el nombre).

Cada declaración de asignación o importación ocurre dentro de un bloque determinado por una definición de clase o de función, o a nivel de módulo (el bloque de código de máximo nivel).

If a name is bound in a block, it is a local variable of that block, unless declared as **`nonlocal`** or **`global`**. If a name is bound at the module level, it is a global variable. (The variables of the module code block are local and global.) If a variable is used in a code block but not defined there, it is a free variable.

Cada ocurrencia de un nombre en el texto del programa se refiere al *binding* de ese nombre, establecido por las siguientes reglas de resolución de nombres.

4.2.2. Resolución de nombres

Un *scope* define la visibilidad de un nombre en un bloque. Si una variable local se define en un bloque, su ámbito incluye ese bloque.

Si la definición ocurre en un bloque de función, el ámbito se extiende a cualquier bloque contenido en el bloque en donde está la definición, a menos que uno de los bloques contenidos introduzca un vínculo diferente para el nombre.

Cuando un nombre es utilizado en un bloque de código, se resuelve utilizando el ámbito de cierre más cercano. El conjunto de todos esos ámbitos visibles para un bloque de código se llama el *environment* del bloque.

Cuando un nombre no se encuentra, se lanza una excepción `NameError`. Si el ámbito actual es una función, y el nombre se refiere a una variable local que todavía no ha sido vinculada a un valor en el punto en el que nombre es utilizado, se lanza una excepción `UnboundLocalError`. `UnboundLocalError` es una subclase de `NameError`.

Si una operación de vinculación de nombre ocurre en cualquier parte dentro de un bloque de código, todos los usos del nombre dentro de ese bloque son tratados como referencias al bloque actual. Esto puede llevar a errores cuando el nombre es utilizado dentro del bloque antes de su vinculación. Esta regla es sutil. Python carece de declaraciones y permite que las operaciones de vinculación de nombres ocurran en cualquier lugar dentro del bloque de código. Las variables locales de un bloque de código pueden determinarse buscando operaciones de vinculación de nombres en el texto completo del bloque. Ver la entrada del FAQ sobre UnboundLocalError para ejemplos.

Si la declaración `global` ocurre dentro de un bloque, todos los usos del nombre especificado en la declaración se refieren a la vinculación que ese nombre tiene en el espacio de nombres (*namespace*) de nivel superior. Los nombres se resuelven en el espacio de nombres de nivel superior buscando en el espacio de nombres global, es decir, el espacio de nombres del módulo que contiene el bloque de código, y en el espacio de nombres incorporado, el *namespace* del módulo `builtins`. La búsqueda se realiza primero en el espacio de nombres global. Si el nombre no se encuentra ahí, se busca en el espacio de nombres incorporado (*builtins namespace*) próximamente. Si tampoco se encuentran los nombres en el espacio de nombres incorporado, se crean variables nuevas en el espacio de nombres global. La declaración global debe preceder a todos los usos de los nombres listados.

La declaración `global` tiene el mismo ámbito que una operación de vinculación de nombre en el mismo bloque. Si el ámbito de cierre más cercano para una variable libre contiene una declaración global, se trata a la variable libre como global.

La declaración `nonlocal` causa que los nombre correspondientes se refieran a variables previamente vinculadas en el ámbito de la función de cierre más cercano. Se lanza un `SyntaxError` en tiempo de compilación si el nombre dado no existe en ningún ámbito de las funciones dentro de las cuales está. Parámetros de tipo no puede recuperarse con la sentencia `nonlocal`.

El espacio de nombres (*namespace*) para un módulo se crea automáticamente la primera vez que se importa el módulo. El módulo principal de un *script* siempre se llama `__main__`.

Los bloques de definición de clase y los argumentos para `exec()` y `eval()` son especiales en el contexto de la resolución de nombres. Una definición de clase es una declaración ejecutable que puede usar y definir nombres. Estas referencias siguen las reglas normales para la resolución de nombres con la excepción de que se buscan las variables locales no vinculadas en el espacio de nombres global. El espacio de nombres de la definición de clase se vuelve el diccionario de atributos de la clase. El ámbito de nombres definido en un bloque de clase está limitado a dicho bloque; no se extiende a los bloques de código de los métodos. Esto incluye las comprensiones y las expresiones generadoras pero no incluye annotation scopes, que tienen acceso a sus ámbitos de clase adjuntos. Esto significa que lo siguiente fallará:

```
class A:
    a = 42
    b = list(a + i for i in range(10))
```

Sin embargo. lo siguiente tendrá éxito:

```
class A:
    type Alias = Nested
    class Nested: pass

print(A.Alias.__value__)  # <type 'A.Nested'>
```

4.2.3. Ámbitos de anotación

Annotations, type parameter lists and `type` statements introduce *annotation scopes*, which behave mostly like function scopes, but with some exceptions discussed below.

Los ámbitos de anotación se utilizan en los siguientes contextos:

- Function annotations.
- Variable annotations.
- Listas de tipo de parámetros para generic type aliases.
- Escriba listas de parámetros para generic functions. Las anotaciones de una función genérica se ejecutan dentro del alcance de la anotación, pero sus valores predeterminados y decoradores no.
- Tipo de parámetros de listas para generic classes. Las clases base y los argumentos de palabra clave de una clase genérica se ejecutan dentro del ámbito de la anotación, pero sus decoradores no.
- Los límites, restricciones y valores predeterminados para los parámetros de tipo (evaluados de forma diferida).
- El valor de los alias de tipo (lazy evaluated).

Los ámbitos de anotación difieren de los ámbitos de función en lo siguiente:

- Los ámbitos de anotación tienen acceso al espacio de nombres de la clase que los rodea. Si un ámbito de anotación está inmediatamente dentro de un ámbito de clase, o dentro de otro ámbito de anotación que está inmediatamente dentro de un ámbito de clase, el código en el ámbito de anotación puede utilizar nombres definidos en el ámbito de clase como si se ejecutara directamente dentro del cuerpo de la clase. Esto contrasta con las funciones normales definidas dentro de las clases, que no pueden acceder a los nombres definidos en el ámbito de la clase.

- Las expresiones en ámbitos de anotación no pueden contener expresiones **`yield`**, `yield from`, **`await`**, o **`:=`**. (Estas expresiones están permitidas en otros ámbitos contenidos dentro del ámbito de la anotación).
- Los nombres definidos en ámbitos de anotación no pueden recuperarse con sentencias **`nonlocal`** en ámbitos internos. Esto incluye sólo parámetros de tipo, ya que ningún otro elemento sintáctico que pueda aparecer dentro de ámbitos de anotación puede introducir nuevos nombres.
- Aunque los ámbitos de anotación tienen un nombre interno, ese nombre no se refleja en el qualified name de los objetos definidos dentro del ámbito. En su lugar, el **`__qualname__`** de dichos objetos es como si el objeto estuviera definido en el ámbito que lo encierra.

Added in version 3.12: Los ámbitos de anotación se introdujeron en Python 3.12 como parte de **PEP 695** [https://peps.python.org/pep-0695/].

Distinto en la versión 3.13: También se usan ámbitos de anotación para parámetros de tipo predeterminados, introducido por **PEP 696** [https://peps.python.org/pep-0696/].

Distinto en la versión 3.14: Annotation scopes are now also used for annotations, as specified in **PEP 649** [https://peps.python.org/pep-0649/] and **PEP 749** [https://peps.python.org/pep-0749/].

4.2.4. Evaluación perezosa

Most annotation scopes are *lazily evaluated*. This includes annotations, the values of type aliases created through the **`type`** statement, and the bounds, constraints, and default values of type variables created through the type parameter syntax. This means that they are not evaluated when the type alias or type variable is created, or when the object carrying annotations is created. Instead, they are only evaluated when necessary, for example when the `__value__` attribute on a type alias is accessed.

Ejemplo:

```
>>> type Alias = 1/0
>>> Alias.__value__
```

```
Traceback (most recent call last):
  ...
ZeroDivisionError: division by zero
>>> def func[T: 1/0](): pass
>>> T = func.__type_params__[0]
>>> T.__bound__
Traceback (most recent call last):
  ...
ZeroDivisionError: division by zero
```

Aquí la excepción se lanza sólo cuando se accede al atributo `__value__` del alias de tipo o al atributo `__bound__` de la variable de tipo.

Este comportamiento es útil principalmente para referencias a tipos que aún no se han definido cuando se crea el alias de tipo o la variable de tipo. Por ejemplo, la evaluación perezosa permite la creación de alias de tipo mutuamente recursivos:

```
from typing import Literal

type SimpleExpr = int | Parenthesized
type Parenthesized = tuple[Literal["("], Expr, Literal["
type Expr = SimpleExpr | tuple[SimpleExpr, Literal["+",
```

Los valores evaluados perezosamente se evalúan en annotation scope, lo que significa que los nombres que aparecen dentro del valor evaluado se buscan como si se utilizaran en el ámbito inmediatamente adyacente.

Added in version 3.12.

4.2.5. Integraciones y ejecución restringida

Los usuarios no deberían tocar `__builtins__`; es un detalle de la implementación en sentido estricto. Los usuarios que quieran sobreescribir valores en los espacios de nombres incorporados deberían usar **`import`** con el módulo **`builtins`** y modificar sus atributos de un modo adecuado.

El espacio de nombres incorporado (*builtin namespace*) asociado a la ejecución de un bloque de código es encontrado buscando el nombre `__builtins__` en su espacio de nombres global; debería ser un diccionario o un módulo (en este último caso, se usa el diccionario del módulo). Por defecto, en el módulo **`__main__`**, `__builtins__` es el módulo **`builtins`**. En cualquier otro módulo, `__builtins__` es un alias para el diccionario del propio módulo **`builtins`**.

4.2.6. Interacción con funcionalidades dinámicas

La resolución de variables libres sucede en tiempo de ejecución, no en tiempo de compilación. Esto significa que el siguiente código va a mostrar 42:

```
i = 10
def f():
    print(i)
i = 42
f()
```

Las funciones **`eval()`** y **`exec()`** no tienen acceso al entorno completo para resolver nombres. Los nombres pueden resolverse en los espacios de nombres locales y globales de la persona que llama. Las variables libres no se resuelven en el espacio de nombres adjunto más cercano, sino en el espacio de nombres global. [1] Las funciones **`exec()`** y **`eval()`** tienen argumentos opcionales para anular el espacio de nombres global y local. Si solo se especifica un espacio de nombres, se usa para ambos.

4.3. Excepciones

Las excepciones son un medio para salir del flujo de control normal de un bloque de código, para gestionar errores u otras condiciones excepcionales. Una excepción es *lanzada* (*raised*) en el momento en que se detecta el error; puede ser *gestionada* (*handled*) por el bloque de código que la rodea o por cualquier bloque de código que directa o indirectamente ha invocado al bloque de código en el que ocurrió el error.

El intérprete Python lanza una excepción cuando detecta un error en tiempo de ejecución (como una división por cero). Un programa Python también puede lanzar una excepción explícitamente, con la declaración `raise`. Los gestores de excepciones se especifican con la declaración `try` … `except`. La cláusula `finally` de tales declaraciones puede utilizarse para especificar código de limpieza que no es el que gestiona la excepción, sino que se ejecutará en cualquier caso, tanto cuando la excepción ha ocurrido en el código que la precede, como cuando esto no ha sucedido.

Python usa el modelo de gestión de errores de «terminación» (»*termination*»): un gestor de excepción puede descubrir qué sucedió y continuar la ejecución en un nivel exterior, pero no puede reparar la causa del error y reintentar la operación que ha fallado (excepto que se reingrese al trozo de código fallido desde su inicio).

Cuando una excepción no está gestionada en absoluto, el intérprete termina la ejecución del programa, o retorna a su bucle principal interactivo. En cualquier caso, imprime un seguimiento de pila, excepto cuando la excepción es `SystemExit`.

Las excepciones se identifican mediante instancias de clase. La cláusula `except` se selecciona en función de la clase de la instancia: debe hacer referencia a la clase de la instancia o a un non-virtual base class de la misma. La instancia puede ser recibida por el manejador y puede llevar información adicional sobre la condición excepcional.

Nota

Los mensajes de excepción no forman parte de la API Python. Su contenido puede cambiar entre una versión de Python y la siguiente sin ningún tipo de advertencia; el código que corre bajo múltiples versiones del intérprete no debería basarse en estos mensajes.

Mira también la descripción de la declaración `try` en la sección La sentencia try, y la declaración `raise` en la sección La declaración raise.

4.4. Runtime Components

4.4.1. General Computing Model

Python’s execution model does not operate in a vacuum. It runs on a host machine and through that host’s runtime environment, including its operating system (OS), if there is one. When a program runs, the conceptual layers of how it runs on the host look something like this:

- **host machine**
  - **process** (global resources)
    - **thread** (runs machine code)

Each process represents a program running on the host. Think of each process itself as the data part of its program. Think of the process” threads as the execution part of the program. This distinction will be important to understand the conceptual Python runtime.

The process, as the data part, is the execution context in which the program runs. It mostly consists of the set of resources assigned to the program by the host, including memory, signals, file handles, sockets, and environment variables.

Processes are isolated and independent from one another. (The same is true for hosts.) The host manages the process” access to its assigned resources, in addition to coordinating between processes.

Each thread represents the actual execution of the program’s machine code, running relative to the resources assigned to the program’s process. It’s strictly up to the host how and when that execution takes place.

From the point of view of Python, a program always starts with exactly one thread. However, the program may grow to run in multiple simultaneous threads. Not all hosts support multiple threads per process, but most do. Unlike processes, threads in a process are not isolated and independent from one another. Specifically, all threads in a process share all of the process” resources.

The fundamental point of threads is that each one does *run* independently, at the same time as the others. That may be only conceptually at the same time («concurrently») or physically («in parallel»). Either way, the threads effectively run at a non-synchronized rate.

> **Nota**
>
> That non-synchronized rate means none of the process” memory is guaranteed to stay consistent for the code running in any given thread. Thus multi-threaded programs must take care to coordinate access to intentionally shared resources. Likewise, they must take care to be absolutely diligent about not accessing any *other* resources in multiple threads; otherwise two threads running at the same time might accidentally interfere with each other’s use of some shared data. All this is true for both Python programs and the Python runtime.
>
> The cost of this broad, unstructured requirement is the tradeoff for the kind of raw concurrency that threads provide. The alternative to the required discipline generally means dealing with non-deterministic bugs and data corruption.

4.4.2. Python Runtime Model

The same conceptual layers apply to each Python program, with some extra data layers specific to Python:

- **host machine**
  - **process** (global resources)
    - Python global runtime (*state*)
      - Python interpreter (*state*)
        - **thread** (runs Python bytecode and «C-API»)
          - Python thread *state*

At the conceptual level: when a Python program starts, it looks exactly like that diagram, with one of each. The runtime may grow to include multiple interpreters, and each interpreter may grow to include multiple thread states.

Nota

A Python implementation won't necessarily implement the runtime layers distinctly or even concretely. The only exception is places where distinct layers are directly specified or exposed to users, like through the `threading` module.

Nota

The initial interpreter is typically called the «main» interpreter. Some Python implementations, like CPython, assign special roles to the main interpreter.

Likewise, the host thread where the runtime was initialized is known as the «main» thread. It may be different from the process" initial thread, though they are often the same. In some cases «main thread» may be even more specific and refer to the initial thread state. A Python runtime might assign specific responsibilities to the main thread, such as handling signals.

As a whole, the Python runtime consists of the global runtime state, interpreters, and thread states. The runtime ensures all that state stays consistent over its lifetime, particularly when used with multiple host threads.

The global runtime, at the conceptual level, is just a set of interpreters. While those interpreters are otherwise isolated and independent from one another, they may share some data or other resources. The runtime is responsible for managing these global resources safely. The actual nature and management of these resources is implementation-specific. Ultimately, the external utility of the global runtime is limited to managing interpreters.

In contrast, an «interpreter» is conceptually what we would normally think of as the (full-featured) «Python runtime». When machine code executing in a host thread interacts with the Python runtime, it calls into Python in the context of a specific interpreter.

Nota

The term «interpreter» here is not the same as the «bytecode interpreter», which is what regularly runs in threads, executing compiled Python code.

In an ideal world, «Python runtime» would refer to what we currently call «interpreter». However, it's been called «interpreter» at least since introduced in 1997 (CPython:a027efa5b [https://github.com/python/cpython/commit/a027efa5b]).

Each interpreter completely encapsulates all of the non-process-global, non-thread-specific state needed for the Python runtime to work. Notably, the interpreter's state persists between uses. It includes fundamental data like `sys.modules`. The runtime ensures multiple threads using the same interpreter will safely share it between them.

A Python implementation may support using multiple interpreters at the same time in the same process. They are independent and isolated from one another. For example, each interpreter has its own `sys.modules`.

For thread-specific runtime state, each interpreter has a set of thread states, which it manages, in the same way the global runtime contains a set of interpreters. It can have thread states for as many host threads as it needs. It may even have multiple thread states for the same host thread, though that isn't as common.

Each thread state, conceptually, has all the thread-specific runtime data an interpreter needs to operate in one host thread. The thread state includes the current raised exception and the thread's Python call stack. It may include other thread-specific resources.

Nota

The term «Python thread» can sometimes refer to a thread state, but normally it means a thread created using the `threading` module.

Each thread state, over its lifetime, is always tied to exactly one interpreter and exactly one host thread. It will only ever be used in that thread and with that interpreter.

Multiple thread states may be tied to the same host thread, whether for different interpreters or even the same interpreter. However, for any given host thread, only one of the thread states tied to it can be used by the thread at a time.

Thread states are isolated and independent from one another and don’t share any data, except for possibly sharing an interpreter and objects or other resources belonging to that interpreter.

Once a program is running, new Python threads can be created using the `threading` module (on platforms and Python implementations that support threads). Additional processes can be created using the `os`, `subprocess`, and `multiprocessing` modules. Interpreters can be created and used with the `interpreters` module. Coroutines (async) can be run using `asyncio` in each interpreter, typically only in a single thread (often the main thread).

Notas al pie

Esta limitación se da porque el código ejecutado por estas operaciones no está disponible en el momento en que se compila el módulo.

[1] [2]

5. El sistema de importación

El código Python en un módulo obtiene acceso al código en otro módulo por el proceso de importarlo. La instrucción `import` es la forma más común de invocar la maquinaria de importación, pero no es la única manera. Funciones como `importlib.import_module()` y built-in `__import__()` también se pueden utilizar para invocar la maquinaria de importación.

La instrucción `import` combina dos operaciones; busca el módulo con nombre y, a continuación, enlaza los resultados de esa búsqueda a un nombre en el ámbito local. La operación de búsqueda de la instrucción `import` se define como una llamada a la función `__import__()`, con los argumentos adecuados. El valor retornado de `__import__()` se utiliza para realizar la operación de enlace de nombre de la instrucción `import`. Consulte la instrucción `import` para obtener los detalles exactos de esa operación de enlace de nombres.

Una llamada directa a `__import__()` realiza solo la búsqueda del módulo y, si se encuentra, la operación de creación del módulo. Aunque pueden producirse ciertos efectos secundarios, como la importación de paquetes primarios y la actualización de varias memorias caché (incluidas `sys.modules`), solo la instrucción `import` realiza una operación de enlace de nombres.

Cuando se ejecuta una instrucción `import`, se llama a la función estándar incorporada `__import__()`. Otros mecanismos para invocar el sistema de importación (como `importlib.import_module()`) pueden optar por omitir `__import__()` y utilizar sus propias soluciones para implementar la semántica de importación.

Cuando se importa un módulo por primera vez, Python busca el módulo y, si se encuentra, crea un objeto de módulo [1], inicializándolo. Si no se encuentra el módulo con nombre, se genera un `ModuleNotFoundError`. Python implementa varias estrategias para buscar el módulo con nombre cuando se invoca la maquinaria de importación. Estas estrategias se pueden modificar y ampliar mediante el uso de varios ganchos descritos en las secciones siguientes.

Distinto en la versión 3.3: El sistema de importación se ha actualizado para aplicar plenamente la segunda fase de PEP 302 [https://peps.python.org/pep-0302/]. Ya no hay ninguna maquinaria de importación implícita: todo el sistema de importación se expone a través de `sys.meta_path`. Además, se ha implementado la compatibilidad con paquetes de espacio de nombres nativos (consulte PEP 420 [https://peps.python.org/pep-0420/]).

5.1. `importlib`

El módulo `importlib` proporciona una API enriquecida para interactuar con el sistema de importación. Por ejemplo `importlib.import_module()` proporciona una API recomendada y más sencilla que la integrada `__import__()` para invocar la maquinaria de importación. Consulte la documentación de la biblioteca `importlib` para obtener más detalles.

5.2. Paquetes

Python sólo tiene un tipo de objeto módulo, y todos los módulos son de este tipo, independientemente de si el módulo está implementado en Python, C, o en cualquier otro lenguaje. Para ayudar a organizar los módulos y proporcionar una jerarquía de nombres, Python tiene un concepto de paquete.

Puedes pensar en los paquetes como los directorios de un sistema de archivos y en los módulos como archivos dentro de los directorios, pero no te tomes esta analogía demasiado literalmente, ya que los paquetes y los módulos no tienen por qué originarse en el sistema de archivos. Para los propósitos de esta documentación, usaremos esta conveniente analogía de directorios y archivos. Al igual que los directorios del sistema de archivos, los paquetes están organizados de forma jerárquica, y los paquetes pueden contener subpaquetes, así como módulos regulares.

Es importante tener en cuenta que todos los paquetes son módulos, pero no todos los módulos son paquetes. O dicho de otro modo, los paquetes son sólo un tipo especial de módulo. Específicamente, cualquier módulo que contenga un atributo `__path__` se considera un paquete.

Todos los módulos tienen un nombre. Los nombres de los subpaquetes están separados de su nombre de paquete principal por un punto, similar a la sintaxis de acceso de atributo estándar de Python. Por lo tanto, podría tener un paquete llamado **`email`**, que a su vez tiene un subpaquete llamado **`email.mime`** y un módulo dentro de ese subpaquete llamado **`email.mime.text`**.

5.2.1. Paquetes regulares

Python define dos tipos de paquetes, paquetes regulares y paquetes de espacio de nombres. Los paquetes regulares son los paquetes tradicionales tal y como existían en Python 3.2 y anteriores. Un paquete regular se implementa típicamente como un directorio que contiene un archivo `init__.py`. Cuando se importa un paquete regular, este archivo `__init__.py` se ejecuta implícitamente, y los objetos que define están vinculados a nombres en el espacio de nombres del paquete. El archivo `__init__.py` puede contener el mismo código Python que puede contener cualquier otro módulo, y Python añadirá algunos atributos adicionales al módulo cuando se importe.

Por ejemplo, la siguiente disposición del sistema de archivos define un paquete `parent` de nivel superior con tres subpaquetes:

```
parent/
    __init__.py
    one/
        __init__.py
    two/
        __init__.py
    three/
        __init__.py
```

Importando `parent.one` se ejecutará implícitamente `parent/__init__.py` y `parent/one/__init__.py`. La importación posterior de `parent.two` o `parent.three` ejecutará `parent/two/__init__.py` y `parent/three/__init__.py` respectivamente.

5.2.2. Paquetes de espacio de nombres

Un paquete de espacio de nombres es un compuesto de varias porciones, donde cada porción contribuye con un subpaquete al paquete padre. Las porciones pueden residir en diferentes lugares del sistema de archivos. Las porciones también pueden encontrarse en archivos zip, en la red, o en cualquier otro lugar que Python busque durante la importación. Los paquetes de espacios de nombres pueden corresponder o no directamente a objetos del sistema de archivos; pueden ser módulos virtuales que no tienen una representación concreta.

Los paquetes de espacios de nombres no usan una lista ordinaria para su atributo `__path__`. En su lugar utilizan un tipo iterable personalizado que realizará automáticamente una nueva búsqueda de porciones de paquete en el siguiente intento de importación dentro de ese paquete si la ruta de su paquete padre (o **`sys.path``** para un paquete de nivel superior) cambia.

Con los paquetes de espacio de nombres, no hay ningún archivo `parent/__init__.py`. De hecho, puede haber varios directorios `padre` encontrados durante la búsqueda de importación, donde cada uno de ellos es proporcionado por una parte diferente. Por lo tanto, `padre/one` no puede estar físicamente situado junto a `padre/two`. En este caso, Python creará un paquete de espacio de nombres para el paquete `parent` de nivel superior siempre que se importe él o uno de sus subpaquetes.

Consulte también **PEP 420** [https://peps.python.org/pep-0420/] para conocer la especificación del paquete de espacio de nombres.

5.3. Buscando

Para comenzar la búsqueda, Python necesita el nombre totalmente calificado del módulo (o paquete, pero para los fines de esta discusión, la diferencia es irrelevante) que se está importando. Este nombre puede provenir de varios argumentos a la instrucción **`import`**, o de los parámetros de las funciones **`importlib.import_module()`** o **`__import__()`**.

Este nombre se utilizará en varias fases de la búsqueda de importación, y puede ser la ruta de acceso punteada a un submódulo, por ejemplo, `foo.bar.baz`. En este caso, Python primero intenta importar `foo`, luego `foo.bar`, y finalmente `foo.bar.baz`. Si se produce un error en cualquiera de las importaciones intermedias, se genera un **`ModuleNotFoundError`**.

5.3.1. La caché del módulo

El primer lugar comprobado durante la búsqueda de importación es **`sys.modules`**. Esta asignación sirve como caché de todos los módulos que se han importado previamente, incluidas las rutas intermedias. Por lo tanto, si `foo.bar.baz` se importó previamente, **`sys.modules`** contendrá entradas para `foo`, `foo.bar`, y `foo.bar.baz`. Cada clave tendrá como valor el objeto de módulo correspondiente.

Durante la importación, el nombre del módulo se busca en **`sys.modules`** y si está presente, el valor asociado es el módulo que satisface la importación y el proceso se completa. Sin embargo, si el valor es `None`, se genera un **`ModuleNotFoundError`**. Si falta el nombre del módulo, Python continuará buscando el módulo.

`sys.modules` se puede escribir. La eliminación de una clave no puede destruir el módulo asociado (ya que otros módulos pueden contener referencias a él), pero invalidará la entrada de caché para el módulo con nombre, lo que hará que Python busque de nuevo el módulo con nombre en su próxima importación. La clave también se puede asignar a `None`, lo que obliga a la siguiente importación del módulo a dar como resultado un **`ModuleNotFoundError`**.

Tenga cuidado, sin embargo, como si mantiene una referencia al objeto module, invalide su entrada de caché en **`sys.modules`** y, a continuación, vuelva a importar el módulo con nombre, los dos objetos de módulo *no* serán los mismos. Por el contrario, `importlib.reload()` reutilizará el objeto de módulo *same* y simplemente reinicializará el contenido del módulo volviendo a ejecutar el código del módulo.

5.3.2. Buscadores y cargadores

Si el módulo con nombre no se encuentra en `sys.modules`, se invoca el protocolo de importación de Python para buscar y cargar el módulo. Este protocolo consta de dos objetos conceptuales, buscadores y cargadores. El trabajo de un buscador es determinar si puede encontrar el módulo con nombre utilizando cualquier estrategia que conozca. Los objetos que implementan ambas interfaces se conocen como importadores se retornan a sí mismos cuando descubren que pueden cargar el módulo solicitado.

Python incluye una serie de buscadores e importadores predeterminados. El primero sabe cómo localizar módulos integrados, y el segundo sabe cómo localizar módulos congelados. Un tercer buscador predeterminado busca módulos en import path. El import path es una lista de ubicaciones que pueden nombrar rutas del sistema de archivos o archivos zip. También se puede ampliar para buscar cualquier recurso localizable, como los identificados por las direcciones URL.

La maquinaria de importación es extensible, por lo que se pueden añadir nuevos buscadores para ampliar el alcance y el alcance de la búsqueda de módulos.

En realidad, los buscadores no cargan módulos. Si pueden encontrar el módulo con nombre, retornan un *module spec*, una encapsulación de la información relacionada con la importación del módulo, que la maquinaria de importación utiliza al cargar el módulo.

En las secciones siguientes se describe el protocolo para buscadores y cargadores con más detalle, incluido cómo puede crear y registrar otros nuevos para ampliar la maquinaria de importación.

Distinto en la versión 3.4: En versiones anteriores de Python, los buscadores retornaban cargadores directamente, mientras que ahora retornen especificaciones de módulo que *contienen* cargadores. Los cargadores todavía se utilizan durante la importación, pero tienen menos responsabilidades.

5.3.3. Ganchos de importación

La maquinaria de importación está diseñada para ser extensible; el mecanismo principal para esto son los *ganchos de importación* (import hooks). Hay dos tipos de ganchos de importación: *meta hooks* (meta ganchos) y *import path hooks* (ganchos de ruta de acceso de importación).

Los meta ganchos se llaman al inicio del procesamiento de importación, antes de que se haya producido cualquier otro procesamiento de importación, que no sea búsqueda de caché de `sys.modules`. Esto permite que los metaganchos reemplacen el procesamiento de `sys.path`, módulos congelados o incluso módulos integrados. Los meta ganchos se registran agregando nuevos objetos de buscador a `sys.meta_path`, como se describe a continuación.

Los ganchos de ruta de acceso de importación se invocan como parte del procesamiento `sys.path` (o `package.__path__`), en el punto donde se encuentra su elemento de ruta de acceso asociado. Los ganchos de ruta de acceso de importación se registran agregando nuevos invocables a `sys.path_hooks` como se describe a continuación.

5.3.4. La meta ruta (*path*)

When the named module is not found in `sys.modules`, Python next searches `sys.meta_path`, which contains a list of meta path finder objects. These finders are queried in order to see if they know how to handle the named module. Meta path finders must implement a method called `find_spec()` which takes three arguments: a name, an import path, and (optionally) a target module. The meta path finder can use any strategy it wants to determine whether it can handle the named module or not.

Si el buscador de metarutas sabe cómo controlar el módulo con nombre, retorna un objeto de especificación. Si no puede controlar el módulo con nombre, retorna `None`. Si el procesamiento de

`sys.meta_path` llega al final de su lista sin retornar una especificación, se genera un **`ModuleNotFoundError`**. Cualquier otra excepción provocada simplemente se propaga hacia arriba, anulando el proceso de importación.

The **`find_spec()`** method of meta path finders is called with two or three arguments. The first is the fully qualified name of the module being imported, for example `foo.bar.baz`. The second argument is the path entries to use for the module search. For top-level modules, the second argument is `None`, but for submodules or subpackages, the second argument is the value of the parent package's `__path__` attribute. If the appropriate `__path__` attribute cannot be accessed, a **`ModuleNotFoundError`** is raised. The third argument is an existing module object that will be the target of loading later. The import system passes in a target module only during reload.

La metaruta se puede recorrer varias veces para una sola solicitud de importación. Por ejemplo, suponiendo que ninguno de los módulos implicados ya se haya almacenado en caché, la importación de `foo.bar.baz` realizará primero una importación de nivel superior, llamando a `mpf.find_spec("foo", None, None)` en cada buscador de metarutas (`mpf`). Después de importar `foo` , `foo.bar` se importará atravesando la meta ruta por segunda vez, llamando a `mpf.find_spec("foo.bar", foo.__path__, None)`. Una vez importado `foo.bar`, el recorrido final llamará a `mpf.find_spec("foo.bar.baz", foo.bar.__path__, None)`.

Algunos buscadores de metarutas solo admiten importaciones de nivel superior. Estos importadores siempre retornarán `None` cuando se pase algo distinto de `None` como segundo argumento.

El valor predeterminado de Python **`sys.meta_path`** tiene tres buscadores de metarutas, uno que sabe cómo importar módulos integrados, uno que sabe cómo importar módulos congelados y otro que sabe cómo importar módulos desde un import path (es decir, el path based finder).

Distinto en la versión 3.4: The **`find_spec()`** method of meta path finders replaced **`find_module()`**, which is now deprecated. While it will continue to work without change, the import machinery will try it only if the finder does not implement **`find_spec()`**.

Distinto en la versión 3.10: Use of **`find_module()`** by the import system now raises **`ImportWarning`**.

Distinto en la versión 3.12: **`find_module()`** has been removed. Use **`find_spec()`** instead.

5.4. Cargando

Si se encuentra una especificación de módulo, la maquinaria de importación la utilizará (y el cargador que contiene) al cargar el módulo. Aquí está una aproximación de lo que sucede durante la porción de carga de la importación:

```
module = None
if spec.loader is not None and hasattr(spec.loader, 'cre
    # It is assumed 'exec_module' will also be defined o
    module = spec.loader.create_module(spec)
if module is None:
    module = ModuleType(spec.name)
# The import-related module attributes get set here:
_init_module_attrs(spec, module)

if spec.loader is None:
    # unsupported
    raise ImportError
if spec.origin is None and spec.submodule_search_locatio
    # namespace package
    sys.modules[spec.name] = module
elif not hasattr(spec.loader, 'exec_module'):
    module = spec.loader.load_module(spec.name)
else:
    sys.modules[spec.name] = module
    try:
        spec.loader.exec_module(module)
    except BaseException:
        try:
            del sys.modules[spec.name]
        except KeyError:
            pass
        raise
return sys.modules[spec.name]
```

Tenga en cuenta los siguientes detalles:

- Si hay un objeto de módulo existente con el nombre dado en `sys.modules`, la importación ya lo habrá retornado.
- El módulo existirá en `sys.modules` antes de que el cargador ejecute el código del módulo. Esto es crucial porque el código del módulo puede (directa o indirectamente) importarse a sí mismo; agregándolo a `sys.modules` de antemano evita la recursividad sin límites en el peor de los casos y la carga múltiple en el mejor.
- Si se produce un error en la carga, el módulo con errores – y solo el módulo con errores – se elimina de `sys.modules`. Cualquier módulo que ya esté en la caché de `sys.modules` y cualquier módulo que se haya cargado correctamente como efecto secundario, debe permanecer en la memoria caché. Esto contrasta con la recarga donde incluso el módulo que falla se deja en `sys.modules`.
- Después de crear el módulo pero antes de la ejecución, la maquinaria de importación establece los atributos del módulo relacionados con la importación («_init_module_attrs» en el ejemplo de pseudocódigo anterior), como se resume en una sección posterior.
- La ejecución del módulo es el momento clave de la carga en el que se rellena el espacio de nombres del módulo. La ejecución se delega por completo en el cargador, lo que llega a decidir qué se rellena y cómo.
- El módulo creado durante la carga y pasado a exec_module() puede no ser el que se retorna al final de la importación [2].

Distinto en la versión 3.4: El sistema de importación se ha hecho cargo de las responsabilidades reutilizables de los cargadores. Estos fueron realizados previamente por el método `importlib.abc.Loader.load_module()`.

5.4.1. Cargadores

Los cargadores de módulos proporcionan la función crítica de carga: ejecución del módulo. La maquinaria de importación llama al método `importlib.abc.Loader.exec_module()` con un único argumento, el objeto module que se va a ejecutar. Se omite cualquier valor retornado de `exec_module()`.

Los cargadores deben cumplir los siguientes requisitos:

- Si el módulo es un módulo Python (a diferencia de un módulo integrado o una extensión cargada dinámicamente), el cargador debe ejecutar el código del módulo en el espacio de nombres global del módulo (`module.__dict__`).
- Si el cargador no puede ejecutar el módulo, debe generar un `ImportError`, aunque se propagará cualquier otra excepción provocada durante `exec_module()`.

En muchos casos, el buscador y el cargador pueden ser el mismo objeto; en tales casos, el método `find_spec()` simplemente retornaría una especificación con el cargador establecido en `self`.

Los cargadores de módulos pueden optar por crear el objeto de módulo durante la carga mediante la implementación de un método `create_module()`. Toma un argumento, la especificación del módulo, y retorna el nuevo objeto de módulo que se usará durante la carga. `create_module()` no necesita establecer ningún atributo en el objeto module. Si el método retorna `None`, la maquinaria de importación creará el nuevo módulo en sí.

Added in version 3.4: El método de cargadores `create_module()`.

Distinto en la versión 3.4: El método `load_module()` fue reemplazado por `exec_module()` y la maquinaria de importación asumió todas las responsabilidades reutilizables de la carga.

Para la compatibilidad con los cargadores existentes, la maquinaria de importación utilizará el método de cargadores `load_module()` si existe y el cargador no implementa también `exec_module()`. Sin embargo, `load_module()` ha quedado obsoleto y los cargadores deben implementar `exec_module()` en su lugar.

El método `load_module()` debe implementar toda la funcionalidad de carga reutilizable descrita anteriormente, además de ejecutar el módulo. Se aplican todas las mismas restricciones, con algunas aclaraciones adicionales:

- Si hay un objeto de módulo existente con el nombre dado en `sys.modules`, el cargador debe utilizar ese módulo existente. (De lo contrario, `importlib.reload()` no funcionará correctamente.) Si el módulo con nombre no existe en `sys.modules`, el cargador debe crear un nuevo objeto de módulo y agregarlo a `sys.modules`.
- El módulo *debe* existir en `sys.modules` antes de que el cargador ejecute el código del módulo, para evitar la recursividad sin límites o la carga múltiple.
- Si se produce un error en la carga, el cargador debe quitar los módulos que ha insertado en `sys.modules`, pero debe quitar **solo** los módulos con errores, y solo si el propio cargador ha cargado los módulos explícitamente.

Distinto en la versión 3.5: A `DeprecationWarning` se genera cuando se define `exec_module()` pero `create_module()` no lo es.

Distinto en la versión 3.6: Un `ImportError` se genera cuando `exec_module()` está definido, pero `create_module()` no lo es.

Distinto en la versión 3.10: El uso de `load_module()` lanzará `ImportWarning`.

5.4.2. Submódulos

Cuando se carga un submódulo mediante cualquier mecanismo (por ejemplo, API `importlib`, las instrucciones `import` o `import-from`, o `__import__()`) integradas, se coloca un enlace en el espacio de nombres del módulo primario al objeto submodule. Por ejemplo, si el paquete `spam` tiene un submódulo `foo`, después de importar `spam.foo`, `spam` tendrá un atributo `foo` que está enlazado al submódulo. Supongamos que tiene la siguiente estructura de directorios:

```
spam/
```

```
    __init__.py
    foo.py
```

y `spam/__init__.py` tiene la siguiente línea:

```
from .foo import Foo
```

a continuación, la ejecución de lo siguiente pone un nombre vinculante para `foo` y `Foo` en el módulo `spam`:

```
>>> import spam
>>> spam.foo
<module 'spam.foo' from '/tmp/imports/spam/foo.py'>
>>> spam.Foo
<class 'spam.foo.Foo'>
```

Dadas las reglas de enlace de nombres familiares de Python, esto puede parecer sorprendente, pero en realidad es una característica fundamental del sistema de importación. La retención invariable es que si tiene `sys.modules[`spam`]` y `sys.modules[`spam.foo`]` (como lo haría después de la importación anterior), este último debe aparecer como el atributo `foo` de la primera.

5.4.3. Module specs

La maquinaria de importación utiliza una variedad de información sobre cada módulo durante la importación, especialmente antes de la carga. La mayor parte de la información es común a todos los módulos. El propósito de las especificaciones de un módulo es encapsular esta información relacionada con la importación por módulo.

El uso de una especificación durante la importación permite transferir el estado entre los componentes del sistema de importación, por ejemplo, entre el buscador que crea la especificación del módulo y el cargador que la ejecuta. Lo más importante es que permite a la maquinaria de importación realizar las operaciones de caldera de carga, mientras que sin una especificación de módulo el cargador tenía esa responsabilidad.

The module’s spec is exposed as `module.__spec__`. Setting `__spec__` appropriately applies equally to modules initialized during interpreter startup. The one exception is `__main__`, where `__spec__` is set to None in some cases.

See `ModuleSpec` for details on the contents of the module spec.

Added in version 3.4.

5.4.4. _path_ attributes on modules

The `__path__` attribute should be a (possibly empty) sequence of strings enumerating the locations where the package’s submodules will be found. By definition, if a module has a `__path__` attribute, it is a package.

A package’s `__path__` attribute is used during imports of its subpackages. Within the import machinery, it functions much the same as `sys.path`, i.e. providing a list of locations to search for modules during import. However, `__path__` is typically much more constrained than `sys.path`.

The same rules used for `sys.path` also apply to a package’s `__path__`. `sys.path_hooks` (described below) are consulted when traversing a package’s `__path__`.

A package’s `__init__.py` file may set or alter the package’s `__path__` attribute, and this was typically the way namespace packages were implemented prior to **PEP 420** [https://peps.python.org/pep-0420/]. With the adoption of **PEP 420** [https://peps.python.org/pep-0420/], namespace packages no longer need to supply `__init__.py` files containing only `__path__` manipulation code; the import machinery automatically sets `__path__` correctly for the namespace package.

5.4.5. Representación (*Reprs*) de módulos

De forma predeterminada, todos los módulos tienen un repr utilizable, sin embargo, dependiendo de los atributos establecidos anteriormente, y en las especificaciones del módulo, puede controlar más explícitamente el repr de los objetos de módulo.

Si el módulo tiene una especificación (`__spec__`), la maquinaria de importación intentará generar un repr a partir de él. Si eso falla o no hay ninguna especificación, el sistema de importación creará un repr predeterminado usando cualquier información disponible en el módulo. Intentará utilizar el `module.__name__`, `module.__file__` y `module.__loader__` como entrada en el repr, con valores predeterminados para cualquier información que falte.

Aquí están las reglas exactas utilizadas:

- Si el módulo tiene un atributo `__spec__`, la información de la especificación se utiliza para generar el repr. Se consultan los atributos «name», «loader», «origin» y «has_location».
- Si el módulo tiene un atributo `__file__`, se utiliza como parte del repr del módulo.
- Si el módulo no tiene `__file__` pero tiene un `__loader__` que no es `None`, entonces el repr del cargador se utiliza como parte del repr del módulo.
- De lo contrario, sólo tiene que utilizar el `__name__` del módulo en el repr.

Distinto en la versión 3.12: El uso de **`module_repr()`**, al ser obsoleto desde Python 3.4, fue eliminado en Python 3.12 y no es llamado durante la resolución de la representación (repr) de un módulo.

5.4.6. Invalidación del código de bytes en caché

Antes de que Python cargue el código de bytes en caché de un archivo `.pyc`, verifica si el caché está actualizado con el archivo `.py` de origen. De forma predeterminada, Python hace esto almacenando la marca de tiempo y el tamaño de la última modificación de la fuente en el archivo de caché al escribirlo. En tiempo de ejecución, el sistema de importación valida el archivo de caché comprobando los metadatos almacenados en el archivo de caché con los metadatos de la fuente.

Python también admite archivos de caché «basados en hash», que almacenan un hash del contenido del archivo de origen en lugar de sus metadatos. Hay dos variantes de archivos `.pyc` basados en hash: marcados y desmarcados. Para los archivos `.pyc` marcados basados en hash, Python valida el archivo de caché mediante el hash del archivo de origen y la comparación del hash resultante con el hash en el archivo de caché. Si se encuentra que un archivo de caché basado en hash comprobado no es válido, Python lo regenera y escribe un nuevo archivo de caché basado en hash comprobado. Para los archivos `.pyc` sin marcar en hash, Python simplemente asume que el archivo de caché es válido si existe. El comportamiento de validación de archivos basado en hash `.pyc` se puede invalidar con el indicador **`--check-hash-based-pycs`**.

Distinto en la versión 3.7: Se han añadido archivos `.pyc` basados en hash. Anteriormente, Python solo admitía la invalidación basada en la marca de tiempo de la caché del código de bytes.

5.5. El buscador basado en rutas

Como se mencionó anteriormente, Python viene con varios buscadores de meta rutas predeterminados. Uno de ellos, llamado el buscador path based finder (**`PathFinder`**), busca una import path, que contiene una lista de entradas de ruta. Cada entrada de ruta de acceso nombra una ubicación para buscar módulos.

El buscador basado en rutas en sí no sabe cómo importar nada. En su lugar, atraviesa las entradas de ruta individuales, asociando cada una de ellas con un buscador de entrada de ruta que sabe cómo manejar ese tipo particular de ruta de acceso.

El conjunto predeterminado de buscadores de entradas de ruta implementa toda la semántica para encontrar módulos en el sistema de archivos, controlando tipos de archivos especiales como el código fuente de Python (archivos ` `.py`), el código de bytes de Python (archivos `.pyc`) y las bibliotecas compartidas (por ejemplo, archivos `.so``). Cuando es compatible con el módulo **`zipimport`** en la biblioteca estándar, los buscadores de entradas de ruta de acceso predeterminados también controlan la carga de todos estos tipos de archivo (excepto las bibliotecas compartidas) desde zipfiles.

Las entradas de ruta de acceso no deben limitarse a las ubicaciones del sistema de archivos. Pueden hacer referencia a direcciones URL, consultas de base de datos o cualquier otra ubicación que se pueda especificar como una cadena.

El buscador basado en rutas proporciona enlaces y protocolos adicionales para que pueda ampliar y personalizar los tipos de entradas de ruta de acceso que se pueden buscar. Por ejemplo, si desea admitir entradas de ruta de acceso como direcciones URL de red, podría escribir un enlace que implemente la semántica HTTP para buscar módulos en la web. Este gancho (un al que se puede llamar) retornaría un path entry finder compatible con el protocolo descrito a continuación, que luego se utilizó para obtener un cargador para el módulo de la web.

Una palabra de advertencia: esta sección y la anterior utilizan el término *finder*, distinguiendo entre ellos utilizando los términos meta path finder y path entry finder. Estos dos tipos de buscadores son muy similares, admiten protocolos similares y funcionan de maneras similares durante el proceso de importación, pero es importante tener en cuenta que son sutilmente diferentes. En particular, los buscadores de meta path operan al principio del proceso de importación, como se indica en el recorrido `sys.meta_path`.

Por el contrario, los buscadores de entradas de ruta son en cierto sentido un detalle de implementación del buscador basado en rutas y, de hecho, si el buscador basado en rutas se eliminara de `sys.meta_path`, no se invocaría ninguna semántica del buscador de entradas de ruta.

5.5.1. Buscadores de entradas de ruta

El path based finder es responsable de encontrar y cargar módulos y paquetes de Python cuya ubicación se especifica con una cadena path entry. La mayoría de las ubicaciones de nombres de entradas de ruta de acceso en el sistema de archivos, pero no es necesario limitarlas a esto.

Como buscador de meta rutas, el buscador path based finder implementa el protocolo `find_spec()` descrito anteriormente, sin embargo, expone enlaces adicionales que se pueden usar para personalizar cómo se encuentran y cargan los módulos desde la ruta import path.

Tres variables son usadas por path based finder, `sys.path`, `sys.path_hooks` y `sys.path_importer_cache`. También se utilizan los atributos `__path__` en los objetos de paquete. Estos proporcionan formas adicionales de personalizar la maquinaria de importación.

`sys.path` contains a list of strings providing search locations for modules and packages. It is initialized from the `PYTHONPATH` environment variable and various other installation- and implementation-specific defaults. Entries in `sys.path` can name directories on the file system, zip files, and potentially other «locations» (see the `site` module) that should be searched for modules, such as URLs, or database queries. Only strings should be present on `sys.path`; all other data types are ignored.

El buscador path based finder es un meta path finder, por lo que la maquinaria de importación comienza la búsqueda import path llamando al método `find_spec()` basado en la ruta de acceso, tal como se describió anteriormente. Cuando se proporciona el argumento `path` a `find_spec()`, será una lista de rutas de acceso de cadena para recorrer - normalmente el atributo `__path__` de un paquete para una importación dentro de ese paquete. Si el argumento `path` es `None`, esto indica una importación de nivel superior y se utiliza `sys.path`.

The path based finder iterates over every entry in the search path, and for each of these, looks for an appropriate path entry finder (`PathEntryFinder`) for the path entry. Because this can be an expensive operation (e.g. there may be `stat()` call overheads for this search), the path based finder maintains a cache mapping path entries to path entry finders. This cache is maintained in `sys.path_importer_cache` (despite the name, this cache actually stores finder objects rather than being limited to importer objects). In this way, the expensive search for a particular path entry location's path entry finder need only be done once. User code is free to remove cache entries from `sys.path_importer_cache` forcing the path based finder to perform the path entry search again.

Si la entrada de ruta de acceso no está presente en la memoria caché, el buscador basado en rutas de acceso recorre en iteración cada llamada que se puede llamar en `sys.path_hooks`. Cada uno de los enlaces de ganchos de rutas de entrada en esta lista se llama con un solo argumento, la entrada de ruta de acceso que se va a buscar. Esta invocable puede retornar un path entry finder que puede controlar la entrada de ruta de acceso, o puede generar `ImportError`. Un `ImportError` es utilizado por el buscador basado en ruta para indicar que el gancho no puede encontrar un path entry finder para eso entrada de ruta. Se omite la excepción y la iteración import path continúa. El enlace debe esperar un objeto de rutas o bytes; la codificación de objetos bytes está hasta el enlace (por ejemplo, puede ser una codificación del sistema de archivos, UTF-8, o algo más), y si el gancho no puede decodificar el argumento, debe generar `ImportError`.

Si la iteración `sys.path_hooks` termina sin que se retorne ningún valor path entry finder, a continuación, el método de búsqueda basado en la ruta de acceso `find_spec()` almacenará `None` en `sys.path_importer_cache` (para indicar que no hay ningún buscador para esta entrada de ruta) y retornará `None`, lo que indica que este meta path finder no pudo encontrar el módulo.

Si un path entry finder *is* retornado por uno de los path entry hook invocables en `sys.path_hooks`, entonces el siguiente protocolo se utiliza para pedir al buscador una especificación de módulo, que luego se utiliza al cargar el módulo.

The current working directory – denoted by an empty string – is handled slightly differently from other entries on `sys.path`. First, if the current working directory cannot be determined or is found not to exist, no value is stored in `sys.path_importer_cache`. Second, the value for the current working directory is looked up fresh for each module lookup. Third, the path used for `sys.path_importer_cache` and returned by `importlib.machinery.PathFinder.find_spec()` will be the actual current working directory and not the empty string.

5.5.2. Buscadores de entradas de ruta

Para admitir las importaciones de módulos y paquetes inicializados y también para contribuir con partes a paquetes de espacio de nombres, los buscadores de entradas de ruta de acceso deben implementar el método `importlib.abc.PathEntryFinder.find_spec()`.

**`importlib.abc.PathEntryFinder.find_spec` ()`** toma dos argumentos: el nombre completo del módulo que se va a importar y el módulo de destino (opcional). `find_spec()` retorna una especificación completamente poblada para el módulo. Esta especificación siempre tendrá «cargador» establecido (con una excepción).

To indicate to the import machinery that the spec represents a namespace portion, the path entry finder sets `submodule_search_locations` to a list containing the portion.

Distinto en la versión 3.4: `find_spec()` replaced **`find_loader()`** and **`find_module()`**, both of which are now deprecated, but will be used if `find_spec()` is not defined.

Los buscadores de entradas de ruta más antiguos pueden implementar uno de estos dos métodos en desuso en lugar de `find_spec()`. Los métodos todavía se respetan en aras de la compatibilidad con versiones anteriores. Sin embargo, si `find_spec()` se implementa en el buscador de entrada de ruta, se omiten los métodos heredados.

`find_loader()` takes one argument, the fully qualified name of the module being imported. `find_loader()` returns a 2-tuple where the first item is the loader and the second item is a namespace portion.

Para la compatibilidad con versiones anteriores con otras implementaciones del protocolo de importación, muchos buscadores de entradas de ruta de acceso también admiten el mismo método tradicional `find_module()` que admiten los buscadores de rutas de acceso meta. Sin embargo, nunca se llama a los métodos del buscador de entradas de ruta `find_module()` con un argumento `path` (se espera que registren la información de ruta adecuada desde la llamada inicial al enlace de ruta).

El método `find_module()` en los buscadores de entrada de ruta está en desuso, ya que no permite que el buscador de entradas de ruta de acceso aporte partes a paquetes de espacio de nombres. Si existen tanto `find_loader()` como `find_module()` en un buscador de entrada de ruta, el sistema de importación siempre llamará a `find_loader()` en lugar de `find_module()`.

Distinto en la versión 3.10: Calls to **`find_module()`** and **`find_loader()`** by the import system will raise **`ImportWarning`**.

Distinto en la versión 3.12: Los métodos `find_module()` y `find_loader()` fueron eliminados.

5.6. Reemplazando el sistema de importación estándar

El mecanismo más confiable para reemplazar todo el sistema de importación es eliminar el contenido predeterminado de **`sys.meta_path`**, sustituyéndolos por completo por un enlace de meta path personalizado.

If it is acceptable to only alter the behaviour of import statements without affecting other APIs that access the import system, then replacing the builtin **`__import__()`** function may be sufficient.

Para evitar selectivamente la importación de algunos módulos de un enlace al principio de la meta path (en lugar de deshabilitar completamente el sistema de importación estándar), es suficiente elevar **`ModuleNotFoundError`** directamente desde **`find_spec()`** en lugar de retornar `None`. Este último indica que la búsqueda de meta path debe continuar, mientras que la generación de una excepción termina inmediatamente.

5.7. Paquete Importaciones relativas

Las importaciones relativas utilizan puntos iniciales. Un único punto inicial indica una importación relativa, empezando por el paquete actual. Dos o más puntos iniciales indican una importación relativa a los elementos primarios del paquete actual, un nivel por punto después del primero. Por ejemplo, dado el siguiente diseño de paquete:

```
package/
    __init__.py
    subpackage1/
        __init__.py
        moduleX.py
        moduleY.py
    subpackage2/
        __init__.py
        moduleZ.py
    moduleA.py
```

En `subpackage1/moduleX.py` o `subpackage1/__init__.py`, las siguientes son importaciones relativas válidas:

```
from .moduleY import spam
from .moduleY import spam as ham
from . import moduleY
from ..subpackage1 import moduleY
from ..subpackage2.moduleZ import eggs
from ..moduleA import foo
```

Las importaciones absolutas pueden utilizar la sintaxis `import <>` o `from <> import <>`, pero las importaciones relativas solo pueden usar el segundo formulario; la razón de esto es que:

```
import XXX.YYY.ZZZ
```

debe exponer `XXX. Yyy. ZZZ` como una expresión utilizable, pero .moduleY no es una expresión válida.

5.8. Consideraciones especiales para _main_

El módulo **`__main__`** es un caso especial relativo al sistema de importación de Python. Como se señaló elsewhere, el módulo `__main__` se inicializa directamente al inicio del intérprete, al igual que **`sys`** y **`builtins`**. Sin embargo, a diferencia de esos dos, no califica estrictamente como un módulo integrado. Esto se debe a que la forma en que se inicializa `__main__` depende de las marcas y otras opciones con las que se invoca el intérprete.

5.8.1. `__main__.__spec__`

Dependiendo de cómo se inicializa **`__main__`**, `__main__.__spec__` se establece correctamente o en `None`.

Cuando Python se inicia con la opción **`-m`**, `__spec__` se establece en la especificación de módulo del módulo o paquete correspondiente. `__spec__` también se rellena cuando el módulo `__main__` se carga como parte de la ejecución de un directorio, zipfile u otro **`sys.path`** entrada.

En los casos restantes `__main__.__spec__` se establece en `None`, ya que el código utilizado para rellenar el **`__main__`** no se corresponde directamente con un módulo importable:

- mensaje interactivo
- opción **`-c`**
- ejecutando desde stdin
- que se ejecuta directamente desde un archivo de código fuente o de código de bytes

Tenga en cuenta que `__main__.__spec__` siempre es `None` en el último caso, *incluso si* el archivo técnicamente podría importarse directamente como un módulo en su lugar. Utilice el modificador **`-m`** si se desean metadatos de módulo válidos en **`__main__`**.

Tenga en cuenta también que incluso cuando `__main__` corresponde a un módulo importable y `__main__.__spec__` se establece en consecuencia, todavía se consideran módulos *distinct*. Esto se debe al hecho de que los bloques protegidos por las comprobaciones `if __name__ == "__main__":` solo se ejecutan cuando el módulo se utiliza para rellenar el espacio de nombres `__main__`, y no durante la importación normal.

5.9. Referencias

La maquinaria de importación ha evolucionado considerablemente desde los primeros días de Python. La especificación original para paquetes [https://www.python.org/doc/essays/packages/] todavía está disponible para leer, aunque algunos detalles han cambiado desde la escritura de ese documento.

La especificación original de `sys.meta_path` era **PEP 302** [https://peps.python.org/pep-0302/], con posterior extensión en **PEP 420** [https://peps.python.org/pep-0420/].

PEP 420 [https://peps.python.org/pep-0420/] introduced namespace packages for Python 3.3. **PEP 420** [https://peps.python.org/pep-0420/] also introduced the `find_loader()` protocol as an alternative to `find_module()`.

PEP 366 [https://peps.python.org/pep-0366/] describe la adición del atributo `__package__` para las importaciones relativas explícitas en los módulos principales.

PEP 328 [https://peps.python.org/pep-0328/] introdujo importaciones relativas absolutas y explícitas e inicialmente propuestas `__name__` para la semántica **PEP 366** [https://peps.python.org/pep-0366/] eventualmente especificaría para `__package__`.

PEP 338 [https://peps.python.org/pep-0338/] define la ejecución de módulos como scripts.

PEP 451 [https://peps.python.org/pep-0451/] agrega la encapsulación del estado de importación por módulo en los objetos de especificación. También descargara la mayoría de las responsabilidades de los cargadores en la maquinaria de importación. Estos cambios permiten el desuso de varias API en el sistema de importación y también la adición de nuevos métodos a los buscadores y cargadores.

Notas al Pie de Pagina

Véase `types. ModuleType`.

La implementación de importlib evita usar el valor retornado directamente. En su lugar, obtiene el objeto module buscando el nombre del módulo en `sys.modules`. El efecto indirecto de esto es que un módulo importado puede sustituirse a sí mismo en `sys.modules`. Este es un comportamiento específico de la implementación que no se garantiza que funcione en otras implementaciones de Python.

[1] [2] [3] [4] [5] [6]

6. Expresiones

Este capítulo explica el significado de los elementos de expresiones en Python.

Syntax Notes: In this and the following chapters, grammar notation will be used to describe syntax, not lexical analysis.

When (one alternative of) a syntax rule has the form:

```
name: othername
```

y no han sido dadas semánticas, las semánticas de esta forma de `name` son las mismas que para `othername`.

6.1. Conversiones aritméticas

When a description of an arithmetic operator below uses the phrase «the numeric arguments are converted to a common real type», this means that the operator implementation for built-in numeric types works as described in the Numeric Types section of the standard library documentation.

Some additional rules apply for certain operators and non-numeric operands (for example, a string as a left argument to the `%` operator). Extensions must define their own conversion behavior.

6.2. Átomos

Atoms are the most basic elements of expressions. The simplest atoms are names or literals. Forms enclosed in parentheses, brackets or braces are also categorized syntactically as atoms.

Formally, the syntax for atoms is:

```
atom:
    | 'True'
    | 'False'
    | 'None'
    | '...'
    | identifier
    | literal
    | enclosure
enclosure:
    | parenth_form
    | list_display
    | dict_display
    | set_display
    | generator_expression
    | yield_atom
```

6.2.1. Built-in constants

The keywords `True`, `False`, and `None` name built-in constants. The token `...` names the **`Ellipsis`** constant.

Evaluation of these atoms yields the corresponding value.

Nota

Several more built-in constants are available as global variables, but only the ones mentioned here are keywords. In particular, these names cannot be reassigned or used as attributes:

```
>>> False = 123
  File "<input>", line 1
   False = 123
   ^^^^^
SyntaxError: cannot assign to False
```

6.2.2. Identificadores (Nombres)

Un identificador encontrándose como un átomo es un nombre. Vea la sección Names (identifiers and keywords) para la definición léxica y la sección Nombres y vínculos para documentación de nombrar y vincular.

Cuando el nombre es vinculado a un objeto, la evaluación del átomo genera ese objeto. Cuando un nombre no es vinculado, un intento de evaluarlo genera una excepción **`NameError`**.

6.2.2.1. Alteración de nombre privado

Cuando un identificador que ocurre textualmente en una definición de clase comienza con dos o más caracteres de guión bajo y no termina en dos o más guiones bajos, es considerado un *private name* de esa clase.

Ver también

La class specifications.

Más concretamente, los nombres privados son transformados a una forma más larga antes de que se genere código para ellos. Si el nombre transformado tiene más de 255 caracteres, puede producirse un truncamiento definido por la implementación.

La transformación es independiente del contexto sintáctico en el que se utilice el identificador, pero sólo se alteran los siguientes identificadores privados:

- Cualquier nombre utilizado como nombre de una variable que se asigna o se lee o cualquier nombre de un atributo al que se accede.

  Sin embargo, el atributo **`__name__`** de funciones anidadas, clases y alias de tipo no son alterados.

- El nombre de módulos importados, por ejemplo `__spam` en `import __spam`. Si el módulo es parte de un paquete (por ejemplo, si su nombre contiene un punto), el nombre *no* es alterado. Por ejemplo, `__foo` en `import __foo.bar` no es alterado.

- El nombre de un miembro importado, por ejemplo `__f` en `from spam import __f`.

La norma de transformación se define de la siguiente forma:

- El nombre de la clase, con los caracteres de guión bajo iniciales eliminados y un único caracter de guión bajo inicial insertado, se inserta delante del identificador. Por ejemplo, el identificador `__spam` que aparece en la clase llamada `Foo`, `_Foo` o `__Foo` se transforma en `_Foo__spam`.
- Si el nombre de la clase consiste únicamente en caracteres de guión bajo, la transformación es la identidad. Por ejemplo, el identificador `__spam` que aparece en una clase llamada `_` o `__` permanece igual.

6.2.3. Literales

A *literal* is a textual representation of a value. Python supports numeric, string and bytes literals. Format strings and template strings are treated as string literals.

Numeric literals consist of a single **`NUMBER`** token, which names an integer, floating-point number, or an imaginary number. See the Literales numéricos section in Lexical analysis documentation for details.

String and bytes literals may consist of several tokens. See section String literal concatenation for details.

Note that negative and complex numbers, like `-3` or `3+4.2j`, are syntactically not literals, but unary or binary arithmetic operations involving the `-` or `+` operator.

Evaluation of a literal yields an object of the given type (**`int`**, **`float`**, **`complex`**, **`str`**, **`bytes`**, or **`Template`**) with the given value. The value may be approximated in the case of floating-point and imaginary literals.

The formal grammar for literals is:

```
literal: strings | NUMBER
```

6.2.3.1. Literals and object identity

Todos los literales corresponden a tipos de datos inmutables y, por lo tanto, la identidad del objeto es menos importante que su valor. Múltiples evaluaciones de literales con el mismo valor (ya sea la misma ocurrencia en el texto del programa o una ocurrencia diferente) pueden obtener el mismo objeto o un objeto diferente con el mismo valor.

CPython implementation detail

For example, in CPython, *small* integers with the same value evaluate to the same object:

```
>>> x = 7
>>> y = 7
>>> x is y
True
```

However, large integers evaluate to different objects:

```
>>> x = 123456789
>>> y = 123456789
>>> x is y
False
```

This behavior may change in future versions of CPython. In particular, the boundary between «small» and «large» integers has already changed in the past.

CPython will emit a `SyntaxWarning` when you compare literals using `is`:

```
>>> x = 7
>>> x is 7
<input>:1: SyntaxWarning: "is" with 'int' literal. Did y
True
```

See ¿Cuándo puedo fiarme de pruebas de identidad con el operador is? for more information.

Template strings are immutable but may reference mutable objects as **`Interpolation`** values. For the purposes of this section, two t-strings have the «same value» if both their structure and the *identity* of the values match.

Detalles de implementación de CPython: Currently, each evaluation of a template string results in a different object.

6.2.3.2. String literal concatenation

Multiple adjacent string or bytes literals, possibly using different quoting conventions, are allowed, and their meaning is the same as their concatenation:

```
>>> "hello" 'world'
"helloworld"
```

This feature is defined at the syntactical level, so it only works with literals. To concatenate string expressions at run time, the “+” operator may be used:

```
>>> greeting = "Hello"
>>> space = " "
>>> name = "Blaise"
>>> print(greeting + space + name)   # not: print(greeti
Hello Blaise
```

Literal concatenation can freely mix raw strings, triple-quoted strings, and formatted string literals. For example:

```
>>> "Hello" r', ' f"{name}!"
"Hello, Blaise!"
```

This feature can be used to reduce the number of backslashes needed, to split long strings conveniently across long lines, or even to add comments to parts of strings. For example:

```
re.compile("[A-Za-z_]"       # letter or underscore
           "[A-Za-z0-9_]*"   # letter, digit or undersco
         )
```

However, bytes literals may only be combined with other byte literals; not with string literals of any kind. Also, template string literals may only be combined with other template string literals:

```
>>> t"Hello" t"{name}!"
Template(strings=('Hello', '!'), interpolations=(...))
```

Formally:

```
strings: (STRING | fstring)+ | tstring+
```

6.2.4. Formas entre paréntesis

Una forma entre paréntesis es una lista de expresiones opcionales encerradas entre paréntesis:

```
parenth_form: "(" [starred_expression] ")"
```

Una expresión entre paréntesis produce lo que la lista de expresión *yield*: si la lista contiene al menos una coma, produce una tupla; en caso contrario, produce la única expresión que que forma la lista de expresiones.

Un par de paréntesis vacío produce un objeto de tupla vacío. Debido a que las tuplas son inmutables, se aplican las mismas reglas que aplican para literales (ej., dos ocurrencias de una tupla vacía puede o no produce el mismo objeto).

Tenga en cuenta que las tuplas no se forman con paréntesis, sino mediante el uso de coma. La excepción es la tupla vacía, para la cual *se* requieren paréntesis. Permitir «nada» sin paréntesis en las expresiones causaría ambigüedades y permitiría que errores tipográficos comunes pasaran sin detectarse.

6.2.5. Despliegues para listas, conjuntos y diccionarios

Para construir una lista, un conjunto o un diccionario, Python provee sintaxis especial denominada «despliegue», cada una de ellas en dos sabores:

- los contenidos del contenedor son listados explícitamente o
- son calculados mediante un conjunto de instrucciones de bucle y filtrado, denominadas una *comprehension*.

Los elementos comunes de sintaxis para las comprensiones son:

```
comprehension: assignment_expression comp_for
comp_for:      ["async"] "for" target_list "in" or_test
comp_iter:     comp_for | comp_if
comp_if:       "if" or_test [comp_iter]
```

La comprensión consiste en una única expresión seguida por al menos una cláusula `for` y cero o más cláusulas `for` o `if`. En este caso, los elementos del nuevo contenedor son aquellos que serían producidos mediante considerar cada una de las cláusulas `for` o `if` un bloque, anidado de izquierda a derecha y evaluando la expresión para producir un elemento cada vez que se alcanza el bloque más interno.

Sin embargo, aparte de la expresión iterable en la cláusula `for` más a la izquierda, la comprensión es ejecutada en un alcance separado implícitamente anidado. Esto asegura que los nombres asignados a en la lista objetiva no se «filtren» en el alcance adjunto.

La expresión iterable en la cláusula más a la izquierda `for` es evaluada directamente en el alcance anidado y luego pasada como un argumento al alcance implícitamente anidado. Subsecuentes cláusulas `for` y cualquier condición de filtro en la cláusula `for` más a la izquierda no pueden ser evaluadas en el alcance adjunto ya que pueden depender de los valores obtenidos del iterable de más a la izquierda. Por ejemplo, `[x*y for x in range(10) for y in range(x, x+10)]`.

Para asegurar que la comprensión siempre resulta en un contenedor del tipo apropiado, las expresiones `yield` y `yield from` están prohibidas en el alcance implícitamente anidado.

Desde Python 3.6, en una función `async def`, se puede usar una cláusula `async for` para iterar sobre un asynchronous iterator. Una comprensión en una función `async def` puede consistir en una cláusula `for` o `async for` después de la expresión inicial, puede contener cláusulas adicionales `for` o `async for` y también puede usar expresiones `await`.

Si una comprensión contiene cláusulas `async for` or si contiene expresiones `await` u otra comprensión asíncrona en cualquier sitio excepto la expresión iterable en la cláusula `for` más a la izquierda, se le llama una *asynchronous comprehension*. Una comprensión asíncrona podría suspender la ejecución de la función de corutina en la que aparece. Consulte también **PEP 530** [https://peps.python.org/pep-0530/].

Added in version 3.6: Fueron introducidas las comprensiones asincrónicas.

Distinto en la versión 3.8: Prohibidas `yield` y `yield from` en el alcance implícitamente anidado.

Distinto en la versión 3.11: Las comprensiones asincrónicas ahora están permitidas dentro de las comprensiones en funciones asincrónicas. Las comprensiones externas implícitamente se vuelven asincrónicas.

6.2.6. Despliegues de lista

Un despliegue de lista es una serie de expresiones posiblemente vacía encerrada entre corchetes:

```
list_display: "[" [flexible_expression_list | comprehens
```

Un despliegue de lista produce un nuevo objeto lista, el contenido se especifica por una lista de expresiones o una comprensión. Cuando se proporciona una lista de expresiones, sus elementos son evaluados desde la izquierda a la derecha y colocados en el objeto lista en ese orden. Cuando se proporciona una comprensión, la lista es construida desde los elementos resultantes de la comprensión.

6.2.7. Despliegues de conjuntos

Un despliegue de conjunto se denota mediante llaves y se distinguen de los despliegues de diccionarios por la ausencia de caracteres de doble punto separando claves y valores:

```
set_display: "{" (flexible_expression_list | comprehensi
```

Un despliegue de conjunto produce un nuevo objeto conjunto mutable, el contenido se especifica mediante una secuencia de expresiones o una comprensión. Cuando se proporciona una lista de expresiones separadas por comas, sus elementos son evaluados desde la izquierda a la derecha y añadidos al objeto de conjunto. Cuando se proporciona una comprensión, el conjunto es construido de los elementos resultantes de la comprensión.

Un conjunto vacío no puede ser construido con `{}`; este literal construye un diccionario vacío.

6.2.8. Despliegues de diccionario

La visualización de un diccionario es una serie posiblemente vacía de elementos de dictado (pares clave/valor) encerrados entre llaves:

```
dict_display:       "{" [dict_item_list | dict_compreher
dict_item_list:     dict_item ("," dict_item)* [","]
dict_item:          expression ":" expression | "**" or_
dict_comprehension: expression ":" expression comp_for
```

Un despliegue de diccionario produce un nuevo objeto diccionario.

Si se proporciona una secuencia de elementos dict separados por comas, se evalúan de izquierda a derecha para definir las entradas del diccionario: cada objeto clave se utiliza como clave en el diccionario para almacenar el valor correspondiente. Esto significa que puede especificar la misma clave varias veces en la lista de elementos de dictado, y el valor final del diccionario para esa clave será el último que se proporcione.

Un asterisco doble `**` indica *descomprimiendo el diccionario*. Su operando debe ser un mapping. Cada elemento de mapeo se agrega al nuevo diccionario. Los valores posteriores reemplazan los valores ya establecidos por elementos de dictado anteriores y desempaquetados de diccionarios anteriores.

Added in version 3.5: Desempaquetar en despliegues de diccionarios, originalmente propuesto por **PEP 448** [https://peps.python.org/pep-0448/].

Una comprensión de diccionario, en contraste a las compresiones de lista y conjunto, necesita dos expresiones separadas con un caracter de doble punto seguido por las cláusulas usuales «for» e «if». Cuando la comprensión se ejecuta, los elementos resultantes clave y valor son insertados en el nuevo diccionario en el orden que son producidos.

Las restricciones sobre los tipos de valores clave se enumeran anteriormente en la sección Jerarquía de tipos estándar. (En resumen, el tipo de clave debe ser hashable, que excluye todos los objetos mutables). No se detectan conflictos entre claves duplicadas; prevalece el último valor (textualmente más a la derecha en la pantalla) almacenado para un valor clave determinado.

Distinto en la versión 3.8: Antes de Python 3.8, en las comprensiones de diccionarios, el orden de evaluación de clave y valor no fue bien definido. En CPython, el valor fue evaluado antes de la clave. A partir de 3.8, la clave es evaluada antes que el valor, como fue propuesto por **PEP 572** [https://peps.python.org/pep-0572/].

6.2.9. Expresiones de generador

Una expresión de generador es una notación compacta de generador en paréntesis:

```
generator_expression: "(" expression comp_for ")"
```

Una expresión de generador produce un nuevo objeto generador. Su sintaxis es la misma que para las comprensiones, excepto que es encerrado en paréntesis en lugar de corchetes o llaves.

Variables used in the generator expression are evaluated lazily when the `__next__()` method is called for the generator object (in the same fashion as normal generators). However, the iterable expression in the leftmost **`for`** clause is immediately evaluated, and the iterator is immediately created for that iterable, so that an error produced while creating the iterator will be emitted at the point where the generator expression is defined, rather than at the point where the first value is retrieved. Subsequent **`for`** clauses and any filter condition in the leftmost **`for`** clause cannot be evaluated in the enclosing scope as they may depend on the values obtained from the leftmost iterable. For example: `(x*y for x in range(10) for y in range(x, x+10))`.

Los paréntesis pueden ser omitidos en ejecuciones con un solo argumento. Vea la sección Invocaciones para más detalles.

Para evitar interferir con la operación esperada de la expresión misma del generador, las expresiones `yield` y `yield from` están prohibidas en el generador definido implícitamente.

Si una expresión de generador contiene cláusulas **`async for`** o expresiones **`await`**, se ejecuta una *asynchronous generator expression*. Una expresión de generador asincrónica retorna un nuevo objeto de generador asincrónico, el cual es un iterador asincrónico (ver Iteradores asíncronos).

Added in version 3.6: Las expresiones de generador asincrónico fueron introducidas.

Distinto en la versión 3.7: Antes de Python 3.7, las expresiones de generador asincrónico podrían aparecer sólo en corrutinas **`async def`**. Desde 3.7, cualquier función puede usar expresiones de generador asincrónico.

Distinto en la versión 3.8: Prohibidas `yield` y `yield from` en el alcance implícitamente anidado.

6.2.10. Expresiones *yield*

```
yield_atom:        "(" yield_expression ")"
```

```
yield_from:       "yield" "from" expression
yield_expression: "yield" yield_list | yield_from
```

La expresión *yield* se usa al definir una función generator o una función asynchronous generator y, por lo tanto, solo se puede usar en el cuerpo de una definición de función. El uso de una expresión *yield* en el cuerpo de una función hace que esa función sea una función generadora, y su uso en el cuerpo de una función **`async def`** hace que la función corrutina sea una función generadora asíncrona. Por ejemplo:

```
def gen():  # define una función generadora
    yield 123

async def agen(): # define una función generadora asíncr
```

Debido a sus efectos secundarios sobre el alcance contenedor, las expresiones `yield` no están permitidas como parte de los alcances implícitamente definidos usados para implementar comprensiones y expresiones de generador.

Distinto en la versión 3.8: Expresiones *yield* prohibidas en los ámbitos anidados implícitamente utilizados para implementar comprensiones y expresiones de generador.

Las funciones generadoras son descritas a continuación, mientras que las funciones generadoras asincrónicas son descritas separadamente en la sección Funciones generadoras asincrónicas.

Cuando se llama a una función generadora, retorna un iterador conocido como generador. Ese generador luego controla la ejecución de la función del generador. La ejecución comienza cuando se llama a uno de los métodos del generador. En ese momento, la ejecución procede con la primera expresión *yield*, donde se suspende nuevamente, retornando el valor de **`yield_list`** a quien llame al generador, o `None` si **`yield_list`** se omite. Por suspendido entiéndase que se retiene todo el estado local, incluyendo los enlaces actuales de las variables locales, el puntero de instrucción, la pila de evaluación interna y el estado de cualquier manejo de excepciones. Cuando se reanuda la ejecución llamando a uno de los métodos del generador, la función puede continuar exactamente como si la expresión *yield* fuera simplemente otra llamada externa. El valor de la expresión *yield* después de la reanudación depende del método que reanudó la ejecución. Si se utiliza `__next__()` (normalmente a través de un `for` o la función incorporada `next()`), el resultado es `None`. De lo contrario, si se utiliza `send()`, el resultado será el valor pasado a ese método.

Todo este hace a las funciones generadores similar a las corrutinas; producen múltiples veces, tienen más de un punto de entrada y su ejecución puede ser suspendida. La única diferencia es que una función generadora no puede controlar si la ejecución debe continuar después de que ceda; el control siempre es transferido al invocador del generador.

Las expresiones *yield* están permitidas en cualquier lugar en un constructo `try`. Si el generador no es reanudado antes de finalizar (alcanzando un recuento de referencia cero o colectando basura), el método generador-iterador `close()` será invocado, permitiendo la ejecución de cualquier cláusula `finally` pendiente.

Cuando se usa `yield from <expr>`, la expresión proporcionada debe ser iterable. Los valores producidos al iterar ese iterable se pasan directamente al llamador de los métodos del generador actual. Cualquier valor pasado con `send()` y cualquier excepción pasada con `throw()` se pasan al iterador subyacente si tiene los métodos apropiados. Si este no es el caso, entonces `send()` lanzará `AttributeError` o `TypeError`, mientras que `throw()` solo lanzará la excepción pasada inmediatamente.

Cuando el iterador subyacente está completo, el atributo `value` de la instancia `StopIteration` generada se convierte en el valor de la expresión *yield*. Puede ser establecido explícitamente al generar `StopIteration` o automáticamente cuando el subiterador es un generador (retornando un valor del subgenerador).

Distinto en la versión 3.3: Añadido `yield from <expr>` para delegar el control de flujo a un subiterador.

Los paréntesis pueden ser omitidos cuando la expresión *yield* es la única expresión en el lado derecho de una sentencia de asignación.

Ver también

PEP 255 [https://peps.python.org/pep-0255/] - Generadores Simples

La propuesta para añadir generadores y la sentencia `yield` a Python.

PEP 342 [https://peps.python.org/pep-0342/] - Corrutinas mediante Generadores Mejorados

La propuesta para mejorar la API y la sintaxis de generadores, haciéndolos utilizables como corrutinas simples.

PEP 380 [https://peps.python.org/pep-0380/] - Sintaxis para Delegar a un Subgenerador

La propuesta para introducir la sintaxis `yield_from`, facilitando la delegación a subgeneradores.

PEP 525 [https://peps.python.org/pep-0525/] - Generadores Asincrónicos

La propuesta que expandió PEP 492 [https://peps.python.org/pep-0492/] añadiendo capacidades de generador a las funciones corrutina.

6.2.10.1. Métodos generador-iterador

Esta subsección describe los métodos de un generador iterador. Estos pueden ser usados para controlar la ejecución de una función generadora.

Tenga en cuenta que invocar cualquiera de los métodos de generador siguientes cuando el generador está todavía en ejecución genera una excepción `ValueError`.

generator.__next__()

Inicia la ejecución de una función generadora o la reanuda en la última expresión de rendimiento ejecutada. Cuando se reanuda una función de generador con un método `send()`, la expresión de rendimiento actual siempre se evalúa como

`None`. Luego, la ejecución continúa con la siguiente expresión de rendimiento, donde el generador se suspende nuevamente y el valor de `yield_list` se retorna a la aquello que llame `__next__()`. Si el generador sale sin generar otro valor, se lanza una excepción `StopIteration`.

Este método es normalmente invocado implícitamente, por ejemplo, por un bucle `for` o por la función incorporada `next()`.

generator.send(*value*)

Reanuda la ejecución y «envía» un valor dentro de la función generadora. El argumento *value* se convierte en el resultado de la expresión *yield* actual. El método `send()` retorna el siguiente valor producido por el generador o genera `StopIteration` si el generador termina sin producir otro valor. Cuando se ejecuta `send()` para comenzar el generador, debe ser invocado con `None` como el argumento, debido a que no hay expresión yield que pueda recibir el valor.

generator.throw(*value*)
generator.throw(*type*[, *value*[, *traceback*]])

Genera una excepción en el punto donde se pausó el generador y retorna el siguiente valor generado por la función del generador. Si el generador sale sin generar otro valor, se genera una excepción `StopIteration`. Si la función generadora no detecta la excepción pasada o genera una excepción diferente, esa excepción se propaga a la persona que llama.

En el uso típico, esto se llama con una sola instancia de excepción similar a la forma en que se usa la palabra clave `raise`.

Sin embargo, para la compatibilidad con versiones anteriores, se admite la segunda firma, siguiendo una convención de versiones anteriores de Python. El argumento *type* debe ser una clase de excepción y *value* debe ser una instancia de excepción. Si no se proporciona *value*, se llama al constructor *type* para obtener una instancia. Si se proporciona *traceback*, se establece en la excepción; de lo contrario, se puede borrar cualquier atributo `__traceback__` existente almacenado en *value*.

Distinto en la versión 3.12: La segunda firma (type[, value[, traceback]]) está obsoleta y puede eliminarse en una versión futura de Python.

generator.close()

Raises a `GeneratorExit` exception at the point where the generator function was paused (equivalent to calling `throw(GeneratorExit)`). The exception is raised by the yield expression where the generator was paused. If the generator function catches the exception and returns a value, this value is returned from `close()`. If the generator function is already closed, or raises **`GeneratorExit`** (by not catching the exception), **`close()`** returns `None`. If the generator yields a value, a `RuntimeError` is raised. If the generator raises any other exception, it is propagated to the caller. If the generator has already exited due to an exception or normal exit, **`close()`** returns **`None`** and has no other effect.

Distinto en la versión 3.13: Si un generador retorna un valor al cerrarse, el valor es retornado por `close()`.

6.2.10.2. Ejemplos

Aquí hay un ejemplo simple que demuestra el comportamiento de generadores y funciones generadoras:

```
>>> def echo(value=None):
...     print("Execution starts when 'next()' is called
...     try:
...         while True:
...             try:
...                 value = (yield value)
...             except Exception as e:
...                 value = e
...     finally:
...         print("Don't forget to clean up when 'close
...
>>> generator = echo(1)
>>> print(next(generator))
Execution starts when 'next()' is called for the first t
1
>>> print(next(generator))
None
>>> print(generator.send(2))
2
>>> generator.throw(TypeError, "spam")
TypeError('spam',)
>>> generator.close()
Don't forget to clean up when 'close()' is called.
```

Para ejemplos usando `yield from`, ver PEP 380: Sintaxis para delegar en un subgenerador en «Qué es nuevo en Python.»

6.2.10.3. Funciones generadoras asincrónicas

La presencia de una expresión *yield* en una función o método definido usando **`async def`** adicionalmente define la función como una función asynchronous generator.

Cuando se invoca una función generadora asincrónica, retorna un iterador asincrónico conocido como un objeto generador asincrónico. Este objeto entonces controla la ejecución de la función generadora. Un objeto generador asincrónico se usa típicamente en una sentencia **`async for`** en una función corrutina análogamente a como sería usado un objeto generador en una sentencia **`for`**.

Llamar a uno de los métodos del generador asíncrono retorna un objeto awaitable y la ejecución comienza cuando se espera este objeto. En ese momento, la ejecución procede a la primera expresión *yield*, donde se suspende nuevamente, retornando el valor de **`yield_list`** a la corutina en espera. Al igual que con un generador, la suspensión significa que se retiene todo el estado local, incluidos los enlaces actuales de las variables locales, el puntero de instrucción, la pila de evaluación interna y el estado de cualquier manejo de excepción. Cuando se reanuda la ejecución esperando el siguiente objeto retornado por los métodos del generador asíncrono, la función puede proceder exactamente como si la expresión *yield* fuera simplemente otra llamada externa. El valor de la expresión *yield* después de reanudar depende del método que reanudó la ejecución. Si se utiliza `__anext__()`, el resultado es `None`. De lo contrario, si se usa `asend()`, el resultado será el valor pasado a ese método.

Si un generador asincrónico sale temprano por `break`, la tarea de la persona que llama se cancela u otras excepciones, el código de limpieza asíncrono del generador se ejecutará y posiblemente lanzará excepciones o accederá a variables de contexto en un contexto inesperado, tal vez después de la vida útil de las tareas de las que depende, o durante el cierre del ciclo de eventos cuando se llama al gancho de recolección de basura del generador asíncrono. Para evitar esto, la persona que llama debe cerrar explícitamente el generador asíncrono llamando al método `aclose()` para finalizar el generador y finalmente desconectarlo del bucle de eventos.

En una función generadora asincrónica, las expresiones *yield* están permitidas en cualquier lugar de un constructo `try`. Sin embargo, si un generador asincrónico no es reanudado antes de finalizar (alcanzando un contador de referencia cero o recogiendo basura), entonces una expresión *yield* dentro de un constructo **`try`** podría fallar al ejecutar cláusulas `finally` pendientes. En este caso, es responsabilidad del bucle de eventos o del planificador ejecutando el generador asincrónico invocar el método `aclose()` del generador-iterador asincrónico y ejecutar el objeto corrutina resultante, permitiendo así la ejecución de cualquier cláusula **`finally`** pendiente.

Para encargarse de la finalización tras la finalización del ciclo de eventos, un ciclo de eventos debe definir una función *finalizer* que tome un generador-iterador asíncrono y presumiblemente llame a `aclose()` y ejecute la rutina. Este *finalizer* se puede registrar llamando a `sys.set_asyncgen_hooks()`. Cuando se itera por primera vez, un generador-iterador asíncrono almacenará el *finalizer* registrado para ser llamado al finalizar. Para obtener un ejemplo de referencia de un método *finalizer*, consulte la implementación de `asyncio.Loop.shutdown_asyncgens` en Lib/asyncio/base_events.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/base_events.py].

La expresión `yield from <expr>` es un error de sintaxis cuando es usada en una función generadora asincrónica.

6.2.10.4. Métodos asincrónicos de generador-iterador

Esta subsección describe los métodos de un generador iterador asincrónico, los cuales son usados para controlar la ejecución de una función generadora.

async agen.__anext__()

Retorna un aguardable que, cuando se ejecuta, comienza a ejecutar el generador asíncrono o lo reanuda en la última expresión *yield* ejecutada. Cuando se reanuda una función de generador asíncrono con un método `__anext__()`, la expresión *yield* actual siempre se evalúa como `None` en el aguardable retornado, que cuando se ejecute continuará con la siguiente expresión *yield*. El valor de `yield_list` de la expresión *yield* es el valor de la excepción `StopIteration` generada por la corutina de finalización. Si el generador asincrónico finaliza sin generar otro valor, el aguardable genera una excepción `StopAsyncIteration`, lo que indica que la iteración asincrónica se ha completado.

Este método es invocado normalmente de forma implícita por un bucle `async for`.

async agen.asend(*value*)

Retorna un aguardable el cual ejecuta cuando se reanude la ejecución del generador asincrónico. Como el método `send()` para un generador, este «envía» un valor a la función generadora asincrónica y el argumento *value* se convierte en el resultado de la expresión *yield* actual. El aguardable retornado por el método `asend()` retornará el siguiente valor producido por el generador como el valor de la `StopIteration` generada o genera `StopAsyncIteration` si el generador asincrónico termina sin generar otro valor. Cuando se invoca **`asend()`** para empezar el generador asincrónico, debe ser invocado con `None` como argumento, porque no hay expresión *yield* que pueda recibir el valor.

async agen.athrow(*value*)

async agen.athrow(*type*[, *value*[, *traceback*]])

Retorna un aguardable que genera una excepción de tipo `type` en el punto donde el generador asincrónico fue pausado y retorna el siguiente valor producido por la función generadora como el valor de la excepción `StopIteration` generada. Si el generador asincrónico termina sin producir otro valor, el aguardable genera una excepción `StopAsyncIteration`. Si la función generadora no caza la excepción pasada o genera una excepción diferente, entonces cuando se ejecuta el aguardable esa excepción se propaga al invocador del aguardable.

Distinto en la versión 3.12: La segunda firma (type[, value[, traceback]]) está obsoleta y puede eliminarse en una versión futura de Python.

async agen.aclose()

Retorna un aguardable que cuando corre lanza un `GeneratorExit` a la función generadora asincrónica en el punto donde fue pausada. Si la función generadora asincrónica termina exitosamente, ya está cerrada o genera **`GeneratorExit`** (sin cazar la excepción), el aguardable retornado lanzará una excepción `StopIteration`. Otros aguardables retornados por subsecuentes invocaciones al generador asincrónico lanzarán una excepción `StopAsyncIteration`. Si el generador asincrónico produce un valor, el aguardable genera un `RuntimeError`. Si el generador asincrónico genera cualquier otra excepción, esta es propagada al invocador del aguardable. Si el generador asincrónico ha terminado debido a una excepción o una terminación normal, entonces futuras invocaciones a `aclose()` retornarán un aguardable que no hace nada.

6.3. Primarios

Los primarios representan las operaciones más fuertemente ligadas al lenguaje. Su sintaxis es:

```
primary: atom | attributeref | subscription | call
```

6.3.1. Referencias de atributos

Una referencia de atributo es un primario seguido de un punto y un nombre:

```
attributeref: primary "." identifier
```

El primario debe evaluar a un objeto de un tipo que soporte referencias de atributos, lo cual la mayoría de los objetos soportan. Luego se le pide a este objeto que produzca el atributo cuyo nombre es el identificador. Este tipo y valor producidos son determinados por el objeto. Múltiples evaluaciones sobre la misma referencia de atributo pueden producir objetos diferentes.

Esta producción puede ser personalizada al sobreescribir el método `__getattribute__()` o `__getattr__()`. El método `__getattribute__()` es llamado primero y retorna un valor o se genera la excepción `AttributeError` si el atributo no está disponible.

Si se genera una excepción `AttributeError` y el objeto tiene un método `__getattr__()`, dicho método es llamado como respaldo.

6.3.2. Subscriptions and slicings

The *subscription* syntax is usually used for selecting an element from a container – for example, to get a value from a `dict`:

```
>>> digits_by_name = {'one': 1, 'two': 2}
>>> digits_by_name['two']  # Subscripting a dictionary u
2
```

In the subscription syntax, the object being subscribed – a primary – is followed by a *subscript* in square brackets. In the simplest case, the subscript is a single expression.

Depending on the type of the object being subscribed, the subscript is sometimes called a key (for mappings), index (for sequences), or *type argument* (for generic types). Syntactically, these are all equivalent:

```
>>> colors = ['red', 'blue', 'green', 'black']
>>> colors[3]  # Subscripting a list using the index 3
'black'

>>> list[str]  # Parameterizing the list type using the
list[str]
```

At runtime, the interpreter will evaluate the primary and the subscript, and call the primary's **`__getitem__()`** or **`__class_getitem__()`** special method with the subscript as argument. For more details on which of these methods is called, see _class_getitem_ frente a _getitem_.

To show how subscription works, we can define a custom object that implements **`__getitem__()`** and prints out the value of the subscript:

```
>>> class SubscriptionDemo:
...     def __getitem__(self, key):
...         print(f'subscripted with: {key!r}')
...
>>> demo = SubscriptionDemo()
>>> demo[1]
subscripted with: 1
>>> demo['a' * 3]
subscripted with: 'aaa'
```

See `__getitem__()` documentation for how built-in types handle subscription.

Subscriptions may also be used as targets in assignment or deletion statements. In these cases, the interpreter will call the subscripted object's `__setitem__()` or `__delitem__()` special method, respectively, instead of `__getitem__()`.

```
>>> colors = ['red', 'blue', 'green', 'black']
>>> colors[3] = 'white'  # Setting item at index
>>> colors
['red', 'blue', 'green', 'white']
>>> del colors[3]  # Deleting item at index 3
>>> colors
['red', 'blue', 'green']
```

All advanced forms of *subscript* documented in the following sections are also usable for assignment and deletion.

6.3.2.1. Segmentos

A more advanced form of subscription, *slicing*, is commonly used to extract a portion of a sequence. In this form, the subscript is a slice: up to three expressions separated by colons. Any of the expressions may be omitted, but a slice must contain at least one colon:

```
>>> number_names = ['zero', 'one', 'two', 'three', 'four
>>> number_names[1:3]
['one', 'two']
>>> number_names[1:]
['one', 'two', 'three', 'four', 'five']
>>> number_names[:3]
['zero', 'one', 'two']
>>> number_names[:]
['zero', 'one', 'two', 'three', 'four', 'five']
>>> number_names[::2]
['zero', 'two', 'four']
>>> number_names[:-3]
['zero', 'one', 'two']
>>> del number_names[4:]
>>> number_names
['zero', 'one', 'two', 'three']
```

When a slice is evaluated, the interpreter constructs a `slice` object whose `start`, `stop` and `step` attributes, respectively, are the results of the expressions between the colons. Any missing expression evaluates to `None`. This `slice` object is then passed to the `__getitem__()` or `__class_getitem__()` special method, as above.

```
# continuing with the SubscriptionDemo instance defined
>>> demo[2:3]
subscripted with: slice(2, 3, None)
>>> demo[::'spam']
subscripted with: slice(None, None, 'spam')
```

6.3.2.2. Comma-separated subscripts

The subscript can also be given as two or more comma-separated expressions or slices:

```
# continuing with the SubscriptionDemo instance defined
>>> demo[1, 2, 3]
subscripted with: (1, 2, 3)
>>> demo[1:2, 3]
subscripted with: (slice(1, 2, None), 3)
```

This form is commonly used with numerical libraries for slicing multi-dimensional data. In this case, the interpreter constructs a `tuple` of the results of the expressions or slices, and passes this tuple to the `__getitem__()` or `__class_getitem__()` special method, as above.

The subscript may also be given as a single expression or slice followed by a comma, to specify a one-element tuple:

```
>>> demo['spam',]
subscripted with: ('spam',)
```

6.3.2.3. «Starred» subscriptions

Added in version 3.11: Expressions in *tuple_slices* may be starred. See **PEP 646** [https://peps.python.org/pep-0646/].

The subscript can also contain a starred expression. In this case, the interpreter unpacks the result into a tuple, and passes this tuple to `__getitem__()` or `__class_getitem__()`:

```
# continuing with the SubscriptionDemo instance defined
>>> demo[*range(10)]
subscripted with: (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

Starred expressions may be combined with comma-separated expressions and slices:

```
>>> demo['a', 'b', *range(3), 'c']
subscripted with: ('a', 'b', 0, 1, 2, 'c')
```

6.3.2.4. Formal subscription grammar

```
subscription:     primary '[' subscript ']'
subscript:        single_subscript | tuple_subscript
single_subscript: proper_slice | assignment_expression
proper_slice:     [expression] ":" [expression] [ ":" [e
tuple_subscript:  ','.(single_subscript | starred_expres
```

Recall that the | operator denotes ordered choice. Specifically, in **subscript**, if both alternatives would match, the first (**single_subscript**) has priority.

6.3.3. Invocaciones

Una invocación invoca un objeto invocable (ej., una function) con una serie posiblemente vacía de argumentos:

```
call:                 primary "(" [argument_list [","] |
argument_list:        positional_arguments ["," starred_
                        ["," keywords_arguments]
                      | starred_and_keywords ["," keywor
```

```
                        | keywords_arguments
positional_arguments:  positional_item ("," positional_it
positional_item:       assignment_expression | "*" expres
starred_and_keywords:  ("*" expression | keyword_item)
                       ("," "*" expression | "," keyword_
keywords_arguments:    (keyword_item | "**" expression)
                       ("," keyword_item | "," "**" expre
keyword_item:          identifier "=" expression
```

Una coma final opcional puede estar presente después de los argumentos posicionales y de palabra clave pero no afecta a las semánticas.

La clave primaria debe evaluar a un objeto invocable (funciones definidas por el usuario, funciones incorporadas, métodos de objetos incorporados, métodos de instancias de clases y todos los objetos que tienen un método `__call__()` son invocables). Todas las expresiones de argumento son evaluadas antes de que la invocación sea intentada. Por favor, consulte la sección Definiciones de funciones para saber más sobre la sintaxis de listas parameter formales.

Si hay argumentos de palabras clave, primero se convierten en argumentos posicionales, de la siguiente manera. Primero, se crea una lista de espacios vacantes para los parámetros formales. Si hay N argumentos posicionales, se colocan en los primeros N espacios. A continuación, para cada argumento de palabra clave, se utiliza el identificador para determinar la ranura correspondiente (si el identificador es el mismo que el nombre del primer parámetro formal, se utiliza la primera ranura, y así sucesivamente). Si el espacio ya está ocupado, se genera una excepción `TypeError`. De lo contrario, el argumento se coloca en el espacio, llenándolo (incluso si la expresión es `None`, llena el espacio). Cuando se han procesado todos los argumentos, los espacios que aún están vacíos se llenan con el valor predeterminado correspondiente de la definición de función. (Los valores predeterminados se calculan, una vez, cuando se define la función; por lo tanto, un objeto mutable como una lista o diccionario usado como valor predeterminado será compartido por todas las llamadas que no especifican un valor de argumento para la ranura correspondiente;

esto debería normalmente se evita.) Si hay espacios vacíos para los cuales no se especifica ningún valor predeterminado, se genera una excepción **`TypeError`**. De lo contrario, la lista de espacios ocupados se utiliza como lista de argumentos para la llamada.

Una implementación puede proveer funciones incorporadas cuyos argumentos posicionales no tienen nombres, incluso si son «nombrados» a efectos de documentación y los cuales por consiguiente no pueden ser suplidos por palabras clave. En CPython, este es el caso para funciones implementadas en C que usan **`PyArg_ParseTuple()`** para analizar sus argumentos.

Si hay más argumentos posicionales que ranuras formales de parámetros, se genera una excepción **`TypeError`**, a no ser que un parámetro formal usando la sintaxis `*identifier` se encuentre presente; en este caso, ese parámetro formal recibe una tupla conteniendo los argumentos posicionales sobrantes (o una tupla vacía su no hay argumentos posicionales sobrantes).

Si un argumento de palabra clave no corresponde a un nombre de parámetro formal, se genera una excepción **`TypeError`**, a no ser que un parámetro formal usando la sintaxis `**identifier` esté presente; en este caso, ese parámetro formal recibe un diccionario que contiene los argumentos de palabra clave sobrantes (usando las palabras clave como claves y los valores de argumento como sus valores correspondientes), o un (nuevo) diccionario vacío si no hay argumentos de palabra clave sobrantes.

Si la sintaxis `*expression` aparece en la invocación de función, `expression` debe evaluar a un iterable. Elementos de esos iterables son tratados como si fueran argumentos posicionales adicionales. Para la invocación `f(x1, x2, *y, x3, x4)`, si *y* evalúa a una secuencia *y1*, …, *yM*, equivale a una invocación con M + 4 argumentos posicionales *x1*, *x2*, *y1*, …, *yM*, *x3*, *x4*.

Una consecuencia de esto es que aunque la sintaxis `*expression` puede aparecer *después* de argumentos de palabra clave explícitos, es procesada *antes* de los argumentos de palabra clave (y cualquiera de los argumentos `*expression` – ver abajo). Así que:

```
>>> def f(a, b):
```

```
...     print(a, b)
...
>>> f(b=1, *(2,))
```

Es inusual que se utilicen argumentos de palabras clave y la sintaxis `*expression` en la misma llamada, por lo que en la práctica esta confusión no suele surgir.

Si la sintaxis `**expression` aparece en la llamada de función, `expression` debe evaluarse como mapping, cuyo contenido se trata como argumentos de palabras clave adicionales. Si a un parámetro que coincide con una clave ya se le ha asignado un valor (mediante un argumento de palabra clave explícito o de otro desempaquetado), se genera una excepción **`TypeError`**.

Cuando se usa `**expression`, cada clave en esta asignación debe ser una cadena. Cada valor del mapeo se asigna al primer parámetro formal elegible para la asignación de palabras clave cuyo nombre es igual a la clave. No es necesario que una clave sea un identificador de Python (por ejemplo, `"max-temp °F"` es aceptable, aunque no coincidirá con ningún parámetro formal que pueda declararse). Si no hay ninguna coincidencia con un parámetro formal, el par clave-valor se recopila mediante el parámetro `**`; si lo hay, o si no lo hay, se genera una excepción **`TypeError`**.

No pueden ser usados parámetros formales usando la sintaxis `*identifier` o `**identifier` como ranuras de argumentos posicionales o como nombres de argumentos de palabra clave.

Distinto en la versión 3.5: Las invocaciones de función aceptan cualquier número de desempaquetados `*` y `**`, los argumentos posicionales pueden seguir a desempaquetados de iterable (`*`) y los argumentos de palabra clave pueden seguir a desempaquetados de diccionario (`*`). Originalmente propuesto por **PEP 448** [https://peps.python.org/pep-0448/].

Una invocación siempre retorna algún valor, posiblemente `None`, a no ser que genere una excepción. Cómo se calcula este valor depende del tipo del objeto invocable.

Si es—

una función definida por el usuario:

> The code block for the function is executed, passing it the argument list. The first thing the code block will do is bind the formal parameters to the arguments; this is described in section Definiciones de funciones. When the code block executes a **`return`** statement, this specifies the return value of the function call. If execution reaches the end of the code block without executing a **`return`** statement, the return value is `None`.

una función o método incorporado:

> El resultado depende del intérprete; ver Funciones incorporadas para las descripciones de funciones y métodos incorporados.

un objeto de clase:

> Se retorna una nueva instancia de esa clase.

un método de una instancia de clase:

> Se invoca la función definida por el usuario correspondiente, con una lista de argumentos con un largo uno mayor que la lista de argumentos de la invocación: la instancia se convierte en el primer argumento.

una instancia de clase:

> La clase debe definir un método **`__call__()`** ; el efecto es entonces el mismo que si ese método fuera invocado.

6.4. Expresión await

Suspende la ejecución de coroutine o un objeto awaitable. Puede ser usado sólo dentro de una coroutine function.

```
await_expr: "await" primary
```

Added in version 3.5.

6.5. El operador de potencia

El operador de potencia se vincula más estrechamente que los operadores unarios a su izquierda; se vincula con menos fuerza que los operadores unarios a su derecha. La sintaxis es:

```
power: (await_expr | primary) ["**" u_expr]
```

Por lo tanto, en una secuencia sin paréntesis de operadores unarios y de potencia, los operadores son evaluados desde la derecha a la izquierda (este no se constriñe al orden de evaluación para los operandos): `-1**2` resulta en `-1`.

The power operator has the same semantics as the built-in `pow()` function, when called with two arguments: it yields its left argument raised to the power of its right argument. Numeric arguments are first converted to a common type, and the result is of that type.

Para operandos int, el resultado tiene el mismo tipo que los operandos a no ser que el segundo argumento sea negativo; en ese caso, todos los argumentos son convertidos a float y se entrega un resultado float. Por ejemplo, `10**2` retorna `100`, pero `10**-2` retorna `0.01`.

Elevar `0.0` a una potencia negativa resulta en un `ZeroDivisionError`. Elevar un número negativo a una potencia fraccional resulta en un número `complex`. (En versiones anteriores se genera un `ValueError`.)

Esta operación se puede personalizar utilizando los métodos especiales `__pow__()` y `__rpow__()`.

6.6. Aritmética unaria y operaciones bit a bit

Toda la aritmética unaria y las operaciones bit a bit tienen la misma prioridad:

```
u_expr: power | "-" u_expr | "+" u_expr | "~" u_expr
```

El operador unario `-` (menos) genera la negación de su argumento numérico; la operación se puede sobreescribir con el método especial `__neg__()`.

El operador unario `+` (más) genera su argumento numérico sin cambios; la operación se puede sobreescribir con el método especial `__pos__()`.

El operador unario `~` (invertir) genera la inversión bit a bit de su argumento entero. La inversión bit a bit de `x` se define como `-(x+1)`. Solo se aplica a números enteros o a objetos personalizados que anulan el método especial `__invert__()`.

En todos los tres casos, si el argumento no tiene el tipo apropiado, se genera una excepción `TypeError`.

6.7. Operaciones aritméticas binarias

Las operaciones aritméticas binarias tienen los niveles convencionales de prioridad. Tenga en cuenta que algunas de esas operaciones también aplican a ciertos tipos no numéricos. Aparte del operador de potencia, hay sólo dos niveles, uno para operadores multiplicativos y uno para aditivos:

```
m_expr: u_expr | m_expr "*" u_expr | m_expr "@" m_expr |
        m_expr "//" u_expr | m_expr "/" u_expr |
        m_expr "%" u_expr
a_expr: m_expr | a_expr "+" m_expr | a_expr "-" m_expr
```

The `*` (multiplication) operator yields the product of its arguments. The arguments must either both be numbers, or one argument must be an integer and the other must be a sequence. In the former case, the numbers are converted to a common real type and then multiplied together. In the latter case, sequence repetition is performed; a negative repetition factor yields an empty sequence.

Esta operación se puede personalizar utilizando los métodos especiales **`__mul__()`** y **`__rmul__()`**.

Distinto en la versión 3.14: If only one operand is a complex number, the other operand is converted to a floating-point number.

El operador `@` (en) está destinado a ser usado para multiplicación de matrices. Ningún tipo incorporado en Python implementa este operador.

Esta operación se puede personalizar utilizando los métodos especiales **`__matmul__()`** y **`__rmatmul__()`**.

Added in version 3.5.

The `/` (division) and `//` (floor division) operators yield the quotient of their arguments. The numeric arguments are first converted to a common type. Division of integers yields a float, while floor division of integers results in an integer; the result is that of mathematical division with the “floor” function applied to the result. Division by zero raises the **`ZeroDivisionError`** exception.

La operación de división se puede personalizar usando los métodos especiales **`__truediv__()`** y **`__rtruediv__()`**. La operación de división entera se puede personalizar usando los métodos especiales **`__floordiv__()`** y **`__rfloordiv__()`**.

The `%` (modulo) operator yields the remainder from the division of the first argument by the second. The numeric arguments are first converted to a common type. A zero right argument raises the **`ZeroDivisionError`** exception. The arguments may be floating-point numbers, e.g., `3.14%0.7` equals `0.34` (since `3.14` equals `4*0.7 + 0.34`.) The modulo operator always yields a result with the same sign as its second operand (or zero); the absolute value of the result is strictly smaller than the absolute value of the second operand [1].

El operador de división entera a la baja y el de módulo están conectados por la siguiente identidad: `x == (x//y)*y + (x%y)`. La división entera a la baja y el módulo también están conectadas por la función incorporada **`divmod()`**: `divmod(x, y) == (x//y, x%y)`. [2].

Adicionalmente a realizar la operación módulo en números, el operador `%` también está sobrecargado por objetos cadena de caracteres para realizar formateo de cadenas al estilo antiguo (también conocido como interpolación). La sintaxis para el formateo de cadenas está descrita en la Referencia de la Biblioteca de Python, sección Formateo de cadenas al estilo *printf*.

La operación _*modulo*_ se puede personalizar utilizando los métodos especiales **`__mod__()`** y **`__rmod__()`**.

El operador de división entera a la baja, el operador de módulo y la función **`divmod()`** no están definidas para números complejos. En su lugar, convierta a un número de punto flotante usando la función **`abs()`** si es necesario.

The `+` (addition) operator yields the sum of its arguments. The arguments must either both be numbers or both be sequences of the same type. In the former case, the numbers are converted to a common real type and then added together. In the latter case, the sequences are concatenated.

Esta operación se puede personalizar utilizando los métodos especiales **`__add__()`** y **`__radd__()`**.

Distinto en la versión 3.14: If only one operand is a complex number, the other operand is converted to a floating-point number.

The `-` (subtraction) operator yields the difference of its arguments. The numeric arguments are first converted to a common real type.

Esta operación se puede personalizar utilizando los métodos especiales **`__sub__()`** y **`__rsub__()`**.

Distinto en la versión 3.14: If only one operand is a complex number, the other operand is converted to a floating-point number.

6.8. Operaciones de desplazamiento

Las operaciones de desplazamiento tienen menos prioridad que las operaciones aritméticas:

```
shift_expr: a_expr | shift_expr ("<<" | ">>") a_expr
```

Estos operadores aceptan enteros como argumentos. Ellos desplazan el primer argumento a la izquierda o derecha el número de dígitos dados por el segundo argumento.

Las operaciones de desplazamiento a la izquierda se pueden personalizar usando los métodos especiales `__lshift__()` y `__rlshift__()`. Las operaciones de desplazamiento a la derecha se pueden personalizar usando los métodos especiales `__rshift__()` y `__rrshift__()`.

Un desplazamiento de *n* bits hacia la derecha se define como una división entera a la baja entre `pow(2,n)`. Un desplazamiento de *n* bits hacia la izquierda se define como una multiplicación por `pow(2,n)`.

6.9. Operaciones bit a bit binarias

Cada una de las tres operaciones de bits binarias tienen diferente nivel de prioridad:

```
and_expr: shift_expr | and_expr "&" shift_expr
xor_expr: and_expr | xor_expr "^" and_expr
or_expr:  xor_expr | or_expr "|" xor_expr
```

El operador `&` genera el AND bit a bit de sus argumentos, que deben ser números enteros o uno de ellos debe ser un objeto personalizado que sobreescriba los métodos especiales `__and__()` o `__rand__()`.

El operador `^` genera el XOR bit a bit (OR exclusivo) de sus argumentos, que deben ser números enteros o uno de ellos debe ser un objeto personalizado que sobreescriba los métodos especiales `__xor__()` o `__rxor__()`.

El operador `|` genera el OR bit a bit (inclusive) de sus argumentos, que deben ser números enteros o uno de ellos debe ser un objeto personalizado que sobreescriba los métodos especiales `__or__()` o `__ror__()`.

6.10. Comparaciones

A diferencia de C, todas las operaciones de comparación en Python tienen la misma prioridad, la cual es menor que la de cualquier operación aritmética, de desplazamiento o bit a bit. También, a diferencia de C, expresiones como `a < b < c` tienen la interpretación convencional en matemáticas:

```
comparison:    or_expr (comp_operator or_expr)*
comp_operator: "<" | ">" | "==" | ">=" | "<=" | "!="
               | "is" ["not"] | ["not"] "in"
```

Las comparaciones generan valores booleanos: `True` o `False`.: dfn: los *Métodos de comparación enriquecidos* personalizados pueden retornar valores no booleanos. En este caso, Python llamará a `bool()` en dicho valor en contextos booleanos.

Las comparaciones pueden ser encadenadas arbitrariamente, ej., `x < y <= z` es equivalente a `x < y and y <= z`, excepto que `y` es evaluado sólo una vez (pero en ambos casos `z` no es evaluado para nada cuando `x < y` se encuentra que es falso).

Formalmente, si *a*, *b*, *c*, …, *y*, *z* son expresiones y *op1*, *op2*, …, *opN* son operadores de comparación, entonces `a op1 b op2 c ... y opN z` es equivalente a `a op1 b and b op2 c and ... y opN z`, excepto que cada expresión es evaluada como mucho una vez.

Tenga en cuenta que `a op1 b op2 c` no implica ningún tipo de comparación entre *a* y *c*, por lo que, por ejemplo, `x < y > z` es perfectamente legal (aunque quizás no es bonito).

6.10.1. Comparaciones de valor

Los operadores `<`, `>`, `==`, `>=`, `<=`, y `!=` comparan los valores de dos objetos. Los objetos no necesitan ser del mismo tipo.

El capítulo Objetos, valores y tipos afirma que los objetos tienen un valor (en adición al tipo e identidad). El valor de un objeto es una noción bastante abstracta en Python: Por ejemplo, no existe un método de acceso canónico para el valor de un objeto. Además, no se requiere que el valor de un objeto deba ser construido de una forma particular, ej. compuesto de todos sus atributos de datos. Los operadores de comparación implementan una noción particular de lo que es el valor de un objeto. Uno puede pensar en ellos definiendo el valor de un objeto indirectamente, mediante su implementación de comparación.

Debido a que todos los tipos son subtipos (directos o indirectos) de **`object`**, ellos heredan el comportamiento de comparación predeterminado desde **`object`**. Los tipos pueden personalizar su comportamiento de comparación implementando *rich comparison methods* como **`__lt__()`**, descritos en Personalización básica.

El comportamiento predeterminado para comparación de igualdad (`==` y `!=`) se basa en la identidad de los objetos. Por lo tanto, la comparación de instancias con la misma identidad resulta en igualdad, y la comparación de igualdad de instancias con diferentes entidades resulta en desigualdad. Una motivación para este comportamiento predeterminado es el deseo de que todos los objetos sean reflexivos (ej. `x is y` implica `x == y`).

No se provee un orden de comparación por defecto (`<`, `>`, `<=`, and `>=`); un intento genera **`TypeError`**. Una motivación para este comportamiento predeterminado es la falta de una invariante similar como para la igualdad.

El comportamiento de la comparación de igualdad predeterminado, que instancias con diferentes identidades siempre son desiguales, puede estar en contraste a que los tipos que necesitarán que tengan una definición sensata de valor de objeto e igualdad basada en el valor. Tales tipos necesitarán personalizar su comportamiento de comparación y, de hecho, un número de tipos incorporados lo han realizado.

La siguiente lista describe el comportamiento de comparación de los tipos incorporados más importantes.

- Números de tipos numéricos incorporadas (Tipos numéricos — int, float, complex) y tipos de la biblioteca estándar **`fractions.Fraction`** y **`decimal.Decimal`** pueden ser comparados consigo mismos y entre sus tipos, con la restricción de que números complejos no soportan orden de comparación. Dentro de los límites de los tipos involucrados, se comparan matemáticamente (algorítmicamente) correctos sin pérdida de precisión.

  Los valores no-un-número `float('NaN')` y `decimal.Decimal('NaN')` son especiales. Cualquier comparación ordenada de un número a un no-un-número es falsa. Una implicación contraintuitiva es que los valores no-un-número son son iguales a sí mismos. Por ejemplo, si `x = float('NaN')`, `3 < x`, `x < 3` y `x == x` son todos falso, mientras `x != x` es verdadero. Este comportamiento cumple con IEEE 754.

- `None` y **`NotImplemented`** son singletons. **PEP 8** [https://peps.python.org/pep-0008/] recomienda que las comparaciones para singletons deben ser realizadas siempre con `is` o `is not`, nunca los operadores de igualdad.

- Las secuencias binarias (instancias de **`bytes`** o **`bytearray`**) pueden ser comparadas entre sí y con otros tipos. Ellas comparan lexicográficamente utilizando los valores numéricos de sus elementos.

- Las cadenas de caracteres (instancias de **`str`**) comparan lexicográficamente usando los puntos de códigos numéricos Unicode (el resultado de la función incorporada **`ord()`**) o sus caracteres. [3]

  Las cadenas de caracteres y las secuencias binarias no pueden ser comparadas directamente.

- Las secuencias (instancias de **`tuple`**, **`list`**, o **`range`**) pueden ser comparadas sólo entre cada uno de sus tipos, con la restricción de que los rangos no soportan comparación de orden. Comparación de igualdad entre esos tipos resulta en desigualdad y la comparación de orden entre esos tipos genera **`TypeError`**.

Las secuencias comparan lexicográficamente usando comparación de sus correspondientes elementos. Los contenedores incorporados asumen que los objetos idénticos son iguales a sí mismos. Eso les permite omitir las pruebas de igualdad para objetos idénticos para mejorar el rendimiento y mantener sus invariantes internos.

La comparación lexicográfica entre colecciones incorporadas funciona de la siguiente forma:

  - Para que dos colecciones sean comparadas iguales, ellas deben ser del mismo tipo, tener el mismo largo, y cada para de elementos correspondientes deben comparar iguales (por ejemplo, `[1,2] == (1,2)` es falso debido a que el tipo no es el mismo).
  - Las colecciones que soportan comparación de orden son ordenadas igual que sus primeros elementos desiguales (por ejemplo, `[1,2,x] <= [1,2,y]` tiene el mismo valor que `x <= y`). Si un elemento correspondiente no existe, la colección más corta es ordenada primero (por ejemplo, `[1,2] < [1,2,3]` es verdadero).

- Las asignaciones (instancias de **`dict`**) se comparan iguales si y solo si tienen pares `(key, value)` iguales. La comparación equitativa de las claves y los valores impone la reflexividad.

  Comparaciones de orden (<, >, <=, and >=) generan **`TypeError`**.

- Conjuntos (instancias de **`set`** o **`frozenset`**) pueden ser comparadas entre sí y entre sus tipos.

  Ellas definen operadores de comparación de orden con la intención de comprobar subconjuntos y superconjuntos. Tales relaciones no definen ordenaciones completas (por ejemplo, los dos conjuntos `{1,2}` y `{2,3}` no son iguales, ni subconjuntos ni superconjuntos uno de otro). Acordemente, los conjuntos no son argumentos apropiados para funciones que dependen de ordenación completa (por ejemplo, **`min()`**, **`max()`** y **`sorted()`** producen resultados indefinidos dados una lista de conjuntos como entradas).

  La comparación de conjuntos refuerza la reflexibilidad de sus elementos.

- La mayoría de los otros tipos incorporados no tienen métodos de comparación implementados, por lo que ellos heredan el comportamiento de comparación predeterminado.

Las clases definidas por el usuario que personalizan su comportamiento de comparación deben seguir algunas reglas de consistencia, si es posible:

- La comparación de igualdad debe ser reflexiva. En otras palabras, los objetos idénticos deben comparar iguales:

  > `x is y` implica `x == y`

- La comparación debe ser simétrica. En otras palabras, las siguientes expresiones deben tener el mismo resultado:

  > `x == y` y `y == x`
  >
  > `x != y` y `y != x`
  >
  > `x < y` y `y > x`
  >
  > `x <= y` y `y >= x`

- La comparación debe ser transitiva. Los siguientes ejemplos (no exhaustivos) ilustran esto:

  > `x > y and y > z` implica `x > z`
  >
  > `x < y and y <= z` implica `x < z`

- La comparación inversa debe resultar en la negación booleana. En otras palabras, las siguientes expresiones deben tener el mismo resultado:

`x == y` y `not x != y`

`x < y` y `not x >= y` (para ordenación completa)

`x > y` y `not x <= y` (para ordenación completa)

Las últimas dos expresiones aplican a colecciones completamente ordenadas (ej. a secuencias, pero no a conjuntos o mapeos). Vea también el decorador **`total_ordering()`**.

- La función **`hash()`** debe ser consistente con la igualdad. Los objetos que son iguales deben tener el mismo valor de hash o ser marcados como inhashables.

Python no fuerza a cumplir esas reglas de coherencia. De hecho, los valores no-un-número son u ejemplo para no seguir esas reglas.

6.10.2. Operaciones de prueba de membresía

Los operadores **`in`** y **`not in`** comprueban membresía. `x in s` evalúa a `True` si *x* es un miembro de *s* y `False` en caso contrario. `x not in s` retorna la negación de `x in s`. Todas las secuencias incorporadas y tipos conjuntos soportan esto, así como diccionarios, para los cuales **`in`** comprueba si un diccionario tiene una clave dada. Para tipos contenedores como list, tuple, set, frozenset, dict o collections.deque, la expresión `x in y` es equivalente a `any(x is e or x == e for e in y)`.

Para los tipos cadenas de caracteres y bytes, `x in y` es `True` si y sólo si *x* es una subcadena de *y*. Una comprobación equivalente es `y.find(x) != -1`. Las cadenas de caracteres vacías siempre son consideradas como subcadenas de cualquier otra cadena de caracteres, por lo que `"" in "abc"` retornará `True`.

Para clases definidas por el usuario las cuales definen el método `__contains__()`, `x in y` retorna `True` si `y.__contains__(x)` retorna un valor verdadero y `False` en caso contrario.

Para clases definidas por el usuario las cuales no definen `__contains__()` pero sí definen `__iter__()`, `x in y` es `True` si algún valor `z`, para el cual la expresión `x is z or x == z` es verdadera, es producido iterando sobre `y`. Si una excepción es generada durante la iteración, es como si `in` hubiera generado esa excepción.

Por último, se intenta el protocolo de iteración al estilo antiguo: si una clase define `__getitem__()`, `x in y` es `True` si y sólo si hay un índice entero no negativo *i* tal que `x is y[i] or x == y[i]` y ningún índice entero menor genera la excepción `IndexError`. (Si cualquier otra excepción es generada, es como si `in` hubiera generado esa excepción).

El operador `not in` es definido para tener el valor de veracidad inverso de `in`.

6.10.3. Comparaciones de identidad

Los operadores `is` y `is not` comprueban la identidad de un objeto. `x is y` es verdadero si y sólo si *x* e *y* son el mismo objeto. La identidad de un Objeto se determina usando la función `id()`. `x is not y` produce el valor de veracidad inverso. [4]

6.11. Operaciones booleanas

```
or_test:  and_test | or_test "or" and_test
and_test: not_test | and_test "and" not_test
not_test: comparison | "not" not_test
```

En el contexto de operaciones booleanas, y también cuando las declaraciones de flujo de control utilizan expresiones, los siguientes valores se interpretan como falsos: `False`, `None`, cero numérico de todos los tipos y cadenas y contenedores vacíos (incluidas cadenas, tuplas, listas, diccionarios). , conjuntos y conjuntos congelados). Todos los demás valores se interpretan como verdaderos. Los objetos definidos por el usuario pueden personalizar su valor de verdad proporcionando un método `__bool__()`.

El operador `not` produce `True` si su argumento es falso, `False` si no.

La expresión `x and y` primero evalúa *x*; si *x* es falso, se retorna su valor; de otra forma, *y* es evaluado y se retorna el valor resultante.

La expresión `x or y` primero evalúa *x*; si *x* es verdadero, se retorna su valor; de otra forma, *y* es evaluado y se retorna el valor resultante.

Tenga en cuenta que ni `and` ni `or` restringen el valor y el tipo que retornan a `False` y `True`, sino retornan el último argumento evaluado. Esto es útil a veces, ej., si `s` es una cadena de caracteres que debe ser remplazada por un valor predeterminado si está vacía, la expresión `s or 'foo'` produce el valor deseado. Debido a que `not` tiene que crear un nuevo valor, retorna un valor booleano indiferentemente del tipo de su argumento (por ejemplo, `not 'foo'` produce `False` en lugar de `''`.)

6.12. Expresiones de asignación

```
assignment_expression: [identifier ":="] expression
```

Una expresión de asignación (a veces también llamada «expresión con nombre» o «morsa») asigna un `expression` a un `identifier`, al mismo tiempo que retorna el valor de el **`expresión`**.

Un caso de uso común es cuando se manejan expresiones regulares coincidentes:

```
if matching := pattern.search(data):
    do_something(matching)
```

O, al procesar un flujo de archivos en fragmentos:

```
while chunk := file.read(9000):
    process(chunk)
```

Las expresiones de asignación deben estar entre paréntesis cuando se usan como expresiones de sentencia y cuando se usan como subexpresiones en expresiones de segmentación, condicional, lambda, argumento de palabra clave y expresiones de comprensión-condicional y en sentencias `assert`, `with` y `assignment`. En todos los demás lugares donde se pueden usar, no se requieren paréntesis, incluidas las sentencias `if` y `while`.

Added in version 3.8: Vea **PEP 572** [https://peps.python.org/pep-0572/] para más detalles sobre las expresiones de asignación.

6.13. Expresiones condicionales

```
conditional_expression: or_test ["if" or_test "else" exp
expression:             conditional_expression | lambda_
```

A conditional expression (sometimes called a «ternary operator») is an alternative to the if-else statement. As it is an expression, it returns a value and can appear as a sub-expression.

La expresión `x if C else y` primero evalúa la condición, *C* en lugar de *x*. Si *C* es verdadero, *x* es evaluado y se retorna su valor; en caso contrario, *y* es evaluado y se retorna su valor.

Vea **PEP 308** [https://peps.python.org/pep-0308/] para más detalles sobre expresiones condicionales.

6.14. Lambdas

```
lambda_expr: "lambda" [parameter_list] ":" expression
```

Las expresiones lambda (a veces denominadas formas lambda) son usadas para crear funciones anónimas. La expresión `lambda parameters: expression` produce un objeto de función. El objeto sin nombre se comporta como un objeto función con:

```
def <lambda>(parameters):
    return expression
```

Vea la sección Definiciones de funciones para la sintaxis de listas de parámetros. Tenga en cuenta que las funciones creadas con expresiones lambda no pueden contener sentencias ni anotaciones.

6.15. Listas de expresiones

```
starred_expression:       "*" or_expr | expression
flexible_expression:      assignment_expression | starre
flexible_expression_list: flexible_expression ("," flexi
starred_expression_list:  starred_expression ("," starre
expression_list:          expression ("," expression)* [
yield_list:               expression_list | starred_expr
```

Excepto cuando son parte de un despliegue de lista o conjunto, una lista de expresión conteniendo al menos una coma produce una tupla. El largo de la tupla es el número de expresiones en la lista. Las expresiones son evaluadas de izquierda a derecha.

Un asterisco `*` denota *iterable unpacking*. Su operando deben ser un iterable. El iterable es expandido en una secuencia de elementos, los cuales son incluidos en la nueva tupla, lista o conjunto en el lugar del desempaquetado.

Added in version 3.5: Desempaquetado iterable en listas de expresiones, originalmente propuesto por **PEP 488** [https://peps.python.org/pep-0488/].

Added in version 3.11: Cualquier elemento en una lista de expresiones puede ser destacado. Vea **PEP 646** [https://peps.python.org/pep-0646/].

La coma final sólo es requerida para crear una tupla de un único elemento, como `1,`; es opcional en otros casos. Una única expresión sin una coma final no crea una tupla, si no genera el valor de esa expresión. (Para crear una tupla vacía, usa un par de paréntesis vacío: `()`.)

6.16. Orden de evaluación

Python evalúa las expresiones de izquierda a derecha. Note que mientras se evalúa una asignación, la parte derecha es evaluada antes que la parte izquierda.

En las siguientes líneas, las expresiones serán evaluadas en el orden aritmético de sus sufijos:

```
expr1, expr2, expr3, expr4
(expr1, expr2, expr3, expr4)
{expr1: expr2, expr3: expr4}
expr1 + expr2 * (expr3 - expr4)
expr1(expr2, expr3, *expr4, **expr5)
expr3, expr4 = expr1, expr2
```

6.17. Prioridad de operador

La siguiente tabla resume la precedencia de operadores en Python, desde la precedencia más alta (más vinculante) hasta la precedencia más baja (menos vinculante). Los operadores en el mismo cuadro tienen la misma prioridad. A menos que la sintaxis se proporcione explícitamente, los operadores son binarios. Los operadores en el mismo cuadro se agrupan de izquierda a derecha (excepto la exponenciación y las expresiones condicionales, que se agrupan de derecha a izquierda).

Tenga en cuenta que las comparaciones, comprobaciones de membresía y las comprobaciones de identidad tienen la misma prioridad y una característica de encadenado de izquierda a derecha como son descritas en la sección Comparaciones.

Operador	Descripción
`(expressions...)`, `[expressions...]`, `{key: value...}`, `{expressions...}`	Expresión de enlace o entre paréntesis, despliegues de lista, diccionario y conjunto
`x[index]`, `x[index:index]`, `x(arguments...)`, `x.attribute`	Subscription (including slicing), call, attribute reference
`await x`	Expresión await

`**`	Exponenciación [5]
`+x`, `-x`, `~x`	NOT positivo, negativo, bit a bit
`*`, `@`, `/`, `//`, `%`	Multiplicación, multiplicación de matrices, división, división entera a la baja, resto (módulo) [6]
`+`, `-`	Adición y sustracción
`<<`, `>>`	Desplazamientos
`&`	AND bit a bit
`^`	XOR bit a bit
`|`	OR bit a bit
`in`, `not in`, `is`, `is not`, `<`, `<=`, `>`, `>=`, `!=`, `==`	Comparaciones, incluyendo comprobaciones de membresía y de identidad
`not x`	Booleano NOT
`and`	Booleano AND
`or`	Booleano OR
`if` – `else`	Expresión condicional
`lambda`	Expresión lambda
`:=`	Expresión de asignación

Notas al pie

Mientras `abs(x%y) < abs(y)` es matemáticamente verdadero, para números de punto flotante puede no ser verdadero numéricamente debido al redondeo. Por ejemplo, y asumiendo una plataforma en la cual un número de punto flotante de Python es un número de doble precisión IEEE 754, a fin de que `-1e-100 % 1e100` tenga el mismo signo que `1e100`, el resultado calculado es `-1e-100 + 1e100`, el cual es numéricamente exactamente igual a `1e100`. La función `math.fmod()` retorna un resultado cuyo signo concuerda con el signo del primer argumento en su lugar, y por ello retorna `-1e-100` en este caso. La aproximación más apropiada depende de su aplicación.

Si x está muy cerca de un entero exacto múltiple de y, es posible para `x//y` que sea uno mayor que `(x-x%y)//y` debido al redondeo. En tales casos, Python retorna el último resultado, a fin de preservar que `divmod(x,y)[0] * y + x % y` sea muy cercano a `x`.

El estándar Unicode distingue entre *code points* (ej. U+0041) y *abstract characters* (ej. «LETRA MAYÚSCULA LATINA A»). Mientras la mayoría de caracteres abstractos en Unicode sólo son representados usando un punto de código, hay un número de caracteres abstractos que pueden adicionalmente ser representados usado una secuencia de más de un punto de código. Por ejemplo, el caracter abstracto «LETRA MAYÚSCULA C LATINA CON CEDILLA» puede ser representado como un único *precomposed character* en la posición de código U+00C7, o como una secuencia de un *base character* en la posición de código U+0043 (LETRA MAYÚSCULA C LATINA), seguida de un *combining character* en la posición de código U+0327 (CEDILLA COMBINADA).

Los operadores de comparación comparan en cadenas de caracteres al nivel de puntos de código Unicode. Esto puede ser contraintuitivo para humanos. Por ejemplo, `"\u00C7" == "\u0043\u0327"` es `False`, incluso aunque ambas cadenas presenten el mismo caracter abstracto «LETRA MAYÚSCULA C LATINA CON CEDILLA».

Para comparar cadenas al nivel de caracteres abstractos (esto es, de una forma intuitiva para humanos), usa **`unicodedata.normalize()`**.

Debido a la recolección automática de basura, listas libres y a la naturaleza dinámica de los descriptores, puede notar un comportamiento aparentemente inusual en ciertos usos del operador **`is`**, como aquellos involucrando comparaciones entre métodos de instancia, o constantes. Compruebe su documentación para más información.

El operador de potencia `**` vincula con menos fuerza que un operador unario aritmético uno bit a bit en su derecha, esto significa que `2**-1` is `0.5`.

El operador `%` también es usado para formateo de cadenas; aplica la misma prioridad.

7. Declaraciones simples

Una declaración simple se compone dentro de una sola línea lógica. Pueden producirse varias declaraciones simples en una sola línea separada por punto y coma. La sintaxis de las declaraciones simples es:

```
simple_stmt: expression_stmt
             | assert_stmt
             | assignment_stmt
             | augmented_assignment_stmt
             | annotated_assignment_stmt
             | pass_stmt
             | del_stmt
             | return_stmt
             | yield_stmt
             | raise_stmt
             | break_stmt
             | continue_stmt
             | import_stmt
             | future_stmt
             | global_stmt
             | nonlocal_stmt
             | type_stmt
```

7.1. Declaraciones de tipo expresión

Las declaraciones de tipo expresión son usadas (en su mayoría interactivamente) para computar y escribir un valor, o (usualmente) para llamar a un método (una función que no retorna un resultado significativo; en Python, los métodos retornan el valor `None`). Otros usos de las declaraciones de tipo expresión son permitidas y ocasionalmente útiles. La sintaxis para una declaración de tipo expresión es:

```
expression_stmt: starred_expression
```

Una declaración de tipo expresión evalúa la lista de expresiones (que puede ser una única expresión).

En modo interactivo, si el valor no es `None`, es convertido a cadena de caracteres usando la función built-in `repr()` y la cadena resultante es escrita en la salida estándar en una línea por si sola (excepto si el resultado es `None`, entonces el procedimiento llamado no produce ninguna salida.)

7.2. Declaraciones de asignación

Las declaraciones de asignación son usadas para (volver a) unir nombres a valores y para modificar atributos o elementos de objetos mutables:

```
assignment_stmt: (target_list "=")+ (starred_expression
target_list:     target ("," target)* [","]
target:          identifier
                 | "(" [target_list] ")"
                 | "[" [target_list] "]"
                 | attributeref
                 | subscription
                 | "*" target
```

(See section Primarios for the syntax definitions for *attributeref* and *subscription*.)

Una declaración de asignación evalúa la lista de expresiones (recuerda que ésta puede ser una única expresión o una lista separada por comas, la última produce una tupla) y asigna el único objeto resultante a cada una de las listas de objetivos, de izquierda a derecha.

Assignment is defined recursively depending on the form of the target (list). When a target is part of a mutable object (an attribute reference or subscription), the mutable object must ultimately perform the assignment and decide about its validity, and may raise an exception if the assignment is unacceptable. The rules observed by various types and the exceptions raised are given with the definition of the object types (see section Jerarquía de tipos estándar).

La asignación de un objeto a una lista de destino, opcionalmente entre paréntesis o corchetes, se define de forma recursiva de la siguiente manera.

- Si la lista de destino es un único objeto sin terminar en coma, opcionalmente entre paréntesis, el objeto es asignado a ese objetivo.
- Sino:
  - Si la lista de objetivos contiene un objetivo prefijado con un asterisco, llamado objetivo “destacado”: El objeto debe ser iterable con al menos tantos elementos como objetivos en la lista de objetivos, menos uno. Los primeros elementos del iterable se asignan, de izquierda a derecha, a los objetivos antes del objetivo destacado. Los elementos finales del iterable se asignan a los objetivos después del objetivo destacado. Luego se asigna una lista de los elementos restantes en el iterable al objetivo destacado (la lista puede estar vacía).
  - Sino: el objeto debe ser iterable con el mismo número de elementos que los elementos en la lista de objetivos, y los elementos son asignados a sus respectivos objetivos de izquierda a derecha.

La asignación de un objeto a un único objetivo se define a continuación de manera recursiva.

- Si el objetivo es un identificador (nombre):

  - Si el nombre no ocurre en una declaración `global` o `nonlocal` en el actual bloque de código: el nombre es unido al objeto del actual espacio de nombres local.
  - Por otra parte: el nombre es unido al objeto en el nombre de espacio global o el nombre de espacio exterior determinado por `nonlocal`, respectivamente.

El nombre se vuelve a unir si ya ha estado unido. Esto puede hacer que el recuento de referencia para el objeto previamente vinculado al nombre llegue a cero, provocando que el objeto se desasigne y se llame a su destructor (si tiene uno).

- Si el destino es una referencia de atributo: se evalúa la expresión primaria en la referencia. Debe producir un objeto con atributos asignables; si este no es el caso, una excepción **`TypeError`** es lanzada. Luego se le pide a ese objeto que asigne el objeto asignado al atributo dado; si no puede realizar la tarea, lanza una excepción (generalmente pero no necesariamente **`AttributeError`**).

  Nota: Si el objeto es una instancia de clase y la referencia de atributo ocurre en ambos lados del operador de asignación, la expresión del lado derecho, `a.x` puede acceder a un atributo de instancia o (si no existe un atributo de instancia) a una clase atributo. El objetivo del lado izquierdo `a.x` siempre se establece como un atributo de instancia, creándolo si es necesario. Por lo tanto, las dos ocurrencias de `a.x` no necesariamente se refieren al mismo atributo: si la expresión del lado derecho se refiere a un atributo de clase, el lado izquierdo crea un nuevo atributo de instancia como el objetivo de la asignación:

  ```
  class Cls:
      x = 3             # class variable
  inst = Cls()
  inst.x = inst.x + 1   # writes inst.x as 4 leaving
  ```

  Esta descripción no se aplica necesariamente a los atributos del descriptor, como las propiedades creadas con **`property()`**.

- If the target is a subscription: The primary expression in the reference is evaluated. Next, the subscript expression is evaluated. Then, the primary’s **`__setitem__()`** method is called with two arguments: the subscript and the assigned object.

> Typically, `__setitem__()` is defined on mutable sequence objects (such as lists) and mapping objects (such as dictionaries), and behaves as follows.
>
> Si el primario es un objeto de secuencia mutable (como una lista), el subíndice debe producir un número entero. Si es negativo, se le suma la longitud de la secuencia. El valor resultante debe ser un número entero no negativo menor que la longitud de la secuencia, y se le pide a la secuencia que asigne el objeto asignado a su elemento con ese índice. Si el índice está fuera de rango, `IndexError` se lanza (la asignación a una secuencia suscrita no puede agregar nuevos elementos a una lista).
>
> If the primary is a mapping object (such as a dictionary), the subscript must have a type compatible with the mapping's key type, and the mapping is then asked to create a key/value pair which maps the subscript to the assigned object. This can either replace an existing key/value pair with the same key value, or insert a new key/value pair (if no key with the same value existed).
>
> If the target is a slicing: The primary expression should evaluate to a mutable sequence object (such as a list). The assigned object should be iterable. The slicing's lower and upper bounds should be integers; if they are `None` (or not present), the defaults are zero and the sequence's length. If either bound is negative, the sequence's length is added to it. The resulting bounds are clipped to lie between zero and the sequence's length, inclusive. Finally, the sequence object is asked to replace the slice with the items of the assigned sequence. The length of the slice may be different from the length of the assigned sequence, thus changing the length of the target sequence, if the target sequence allows it.

Aunque la definición de asignación implica que las superposiciones entre el lado izquierdo y el lado derecho son 'simultáneas' (por ejemplo, `a, b = b, a` intercambia dos variables), las superposiciones *dentro* de la colección de las variables asignadas ocurren de izquierda a derecha, lo que a veces genera confusión. Por ejemplo, el siguiente programa imprime `[0, 2]`:

```
x = [0, 1]
i = 0
i, x[i] = 1, 2         # i is updated, then x[i] is upda
print(x)
```

Ver también

PEP 3132 [https://peps.python.org/pep-3132/] - Desembalaje Iterable Extendido
La especificación para la función `*target`.

7.2.1. Declaraciones de asignación aumentada

La asignación aumentada es la combinación, en una sola declaración, de una operación binaria y una declaración de asignación:

```
augmented_assignment_stmt: augtarget augop (expression_l
augtarget:                 identifier | attributeref | s
augop:                     "+=" | "-=" | "*=" | "@=" | "
                           | ">>=" | "<<=" | "&=" | "^="
```

(Ver sección Primarios para la definición de la sintaxis de los tres últimos símbolos.)

Una asignación aumentada evalúa el destino (que, a diferencia de las declaraciones de asignación normales, no puede ser un desempaquetado) y la lista de expresiones, realiza la operación binaria específica del tipo de asignación en los dos operandos y asigna el resultado al destino original. El objetivo sólo se evalúa una vez.

An augmented assignment statement like `x += 1` can be rewritten as `x = x + 1` to achieve a similar, but not exactly equal effect. In the augmented version, `x` is only evaluated once. Also, when possible, the actual operation is performed *in-place*, meaning that rather than creating a new object and assigning that to the target, the old object is modified instead.

A diferencia de las asignaciones normales, las asignaciones aumentadas evalúan el lado izquierdo *antes* de evaluar el lado derecho. Por ejemplo, `a[i] += f(x)` primero busca `a[i]`, entonces evalúa `f(x)` y realiza la suma, y finalmente, vuelve a escribir el resultado en `a[i]`.

Con la excepción de la asignación a tuplas y múltiples objetivos en una sola instrucción, la asignación realizada por las instrucciones de asignación aumentada se maneja de la misma manera que las asignaciones normales. De manera similar, con la excepción del posible comportamiento *in situ*, la operación binaria realizada por la asignación aumentada es la misma que las operaciones binarias normales.

Para destinos que son referencias a atributos, lo mismo que caveat about class and instance attributes se aplica para asignaciones regulares.

7.2.2. Declaraciones de asignación anotadas

Asignación Annotation es la combinación, en una única declaración, de una variable o anotación de atributo y una declaración de asignación opcional:

```
annotated_assignment_stmt: augtarget ":" expression
                           ["=" (starred_expression | yi
```

The difference from normal Declaraciones de asignación is that only a single target is allowed.

The assignment target is considered «simple» if it consists of a single name that is not enclosed in parentheses. For simple assignment targets, if in class or module scope, the annotations are gathered in a lazily evaluated annotation scope. The annotations can be evaluated using the **`__annotations__`** attribute of a class or module, or using the facilities in the **`annotationlib`** module.

If the assignment target is not simple (an attribute, subscript node, or parenthesized name), the annotation is never evaluated.

Si se anota un nombre en el ámbito de una función, este nombre es local para ese ámbito. Las anotaciones nunca se evalúan y almacenan en ámbitos de función.

If the right hand side is present, an annotated assignment performs the actual assignment as if there was no annotation present. If the right hand side is not present for an expression target, then the interpreter evaluates the target except for the last `__setitem__()` or `__setattr__()` call.

Ver también

PEP 526 [https://peps.python.org/pep-0526/] - Sintaxis para anotaciones de variable

La propuesta que agregó sintaxis para anotar los tipos de variables (incluidas variables de clase y variables de instancia), en lugar de expresarlas a través de comentarios.

PEP 484 [https://peps.python.org/pep-0484/] - Indicadores de tipo

La propuesta que agregó el módulo `typing` para proporcionar una sintaxis estándar para las anotaciones de tipo para ser utilizadas en herramientas de análisis estático e IDEs.

Distinto en la versión 3.8: Now annotated assignments allow the same expressions in the right hand side as regular assignments. Previously, some expressions (like un-parenthesized tuple expressions) caused a syntax error.

Distinto en la versión 3.14: Annotations are now lazily evaluated in a separate annotation scope. If the assignment target is not simple, annotations are never evaluated.

7.3. La declaración `assert`

Las declaraciones de afirmación son una forma conveniente de insertar afirmaciones de depuración en un programa:

```
assert_stmt: "assert" expression ["," expression]
```

La forma simple, `assert expressionn`, es equivalente a

```
if __debug__:
    if not expression: raise AssertionError
```

La forma extendida, `assert expression1, expression2`, es equivalente a

```
if __debug__:
    if not expression1: raise AssertionError(expression2
```

These equivalences assume that **`__debug__`** and **`AssertionError`** refer to the built-in variables with those names. In the current implementation, the built-in variable `__debug__` is `True` under normal circumstances, `False` when optimization is requested (command line option **`-O`**). The current code generator emits no code for an **`assert`** statement when optimization is requested at compile time. Note that it is unnecessary to include the source code for the expression that failed in the error message; it will be displayed as part of the stack trace.

Asignaciones a **`__debug__`** son ilegales. El valor para la variable se determina cuando arranca el intérprete.

7.4. La declaración `pass`

```
pass_stmt: "pass"
```

`pass` es una operación nula — cuando se ejecuta, no sucede nada. Es útil como marcador de posición cuando se requiere una declaración sintácticamente, pero no es necesario ejecutar código, por ejemplo:

```
def f(arg): pass    # a function that does nothing (yet)

class C: pass       # a class with no methods (yet)
```

7.5. La declaración `del`

```
del_stmt: "del" target_list
```

La eliminación se define de forma recursiva de manera muy similar a la forma en que se define la asignación. En lugar de explicarlo con todos los detalles, aquí hay algunas sugerencias.

La eliminación de una lista de objetivos elimina cada objetivo de forma recursiva, de izquierda a derecha.

Deletion of a name removes the binding of that name from the local or global namespace, depending on whether the name occurs in a `global` statement in the same code block. Trying to delete an unbound name raises a `NameError` exception.

Deletion of attribute references and subscriptions is passed to the primary object involved; deletion of a slicing is in general equivalent to assignment of an empty slice of the right type (but even this is determined by the sliced object).

Distinto en la versión 3.2: Anteriormente era ilegal eliminar un nombre del espacio de nombres local si aparece como una variable libre en un bloque anidado.

7.6. La declaración `return`

```
return_stmt: "return" [expression_list]
```

El `return` sólo puede ocurrir sintácticamente anidado en una definición de función, no dentro de una definición de clase anidada.

Si una lista de expresiones es presente, ésta es evaluada, sino es substituida por `None` .

`return` deja la llamada a la función actual con la lista de expresiones (o `None`) como valor de retorno.

Cuando **`return`** pasa el control de una sentencia **`try`** con una cláusula **`finally`**, esa cláusula **`finally`** se ejecuta antes de dejar realmente la función.

En una función generadora, la declaración **`return`** indica que el generador ha acabado y hará que **`StopIteration`** se lance. El valor retornado (si lo hay) se utiliza como argumento para construir **`StopIteration`** y se convierte en el atributo **`StopIteration.value`**.

En una función de generador asíncrono, una declaración **`return`** vacía indica que el generador asíncrono ha acabado y va a lanzar un **`StopAsyncIteration`**. Una declaración **`return`** no vacía es un error de sintaxis en una función generadora asíncrona.

7.7. La declaración `yield`

`yield_stmt`: **`yield_expression`**

A **`yield`** statement is semantically equivalent to a yield expression. The `yield` statement can be used to omit the parentheses that would otherwise be required in the equivalent yield expression statement. For example, the yield statements

```
yield <expr>
yield from <expr>
```

son equivalentes a las funciones que producen declaraciones

```
(yield <expr>)
(yield from <expr>)
```

Yield expressions and statements are only used when defining a generator function, and are only used in the body of the generator function. Using **`yield`** in a function definition is sufficient to cause that definition to create a generator function instead of a normal function.

Para todos los detalles de la semántica de **`yield`**, referirse a la sección Expresiones yield.

7.8. La declaración `raise`

```
raise_stmt: "raise" [expression ["from" expression]]
```

Si no hay expresiones presentes, `raise` vuelve a lanzar la excepción que se está manejando actualmente, que también se conoce como *excepción activa*. Si actualmente no hay una excepción activa, se lanza una excepción `RuntimeError` que indica que se trata de un error.

De lo contrario, `raise` evalúa la primera expresión como el objeto de excepción. Debe ser una subclase o una instancia de `BaseException`. Si es una clase, la instancia de excepción se obtendrá cuando sea necesario creando una instancia de la clase sin argumentos.

El *type* de la excepción es la instancia de la clase excepción, el *value* es la propia instancia.

A traceback object is normally created automatically when an exception is raised and attached to it as the `__traceback__` attribute. You can create an exception and set your own traceback in one step using the `with_traceback()` exception method (which returns the same exception instance, with its traceback set to its argument), like so:

```
raise Exception("foo occurred").with_traceback(traceback
```

The `from` clause is used for exception chaining: if given, the second *expression* must be another exception class or instance. If the second expression is an exception instance, it will be attached to the raised exception as the `__cause__` attribute (which is writable). If the expression is an exception class, the class will be instantiated and the resulting exception instance will be attached to the raised exception as the `__cause__` attribute. If the raised exception is not handled, both exceptions will be printed:

```
>>> try:
...     print(1 / 0)
... except Exception as exc:
...     raise RuntimeError("Something bad happened") fro
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    print(1 / 0)
          ~~^~~
ZeroDivisionError: division by zero

The above exception was the direct cause of the followi

Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
    raise RuntimeError("Something bad happened") from ex
RuntimeError: Something bad happened
```

A similar mechanism works implicitly if a new exception is raised when an exception is already being handled. An exception may be handled when an `except` or `finally` clause, or a `with` statement, is used. The previous exception is then attached as the new exception's `__context__` attribute:

```
>>> try:
...     print(1 / 0)
... except:
...     raise RuntimeError("Something bad happened")
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    print(1 / 0)
          ~~^~~
ZeroDivisionError: division by zero

During handling of the above exception, another exceptio

Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
```

```
    raise RuntimeError("Something bad happened")
RuntimeError: Something bad happened
```

Exception chaining can be explicitly suppressed by specifying **`None`** in the `from` clause:

```
>>> try:
...     print(1 / 0)
... except:
...     raise RuntimeError("Something bad happened") fro
...
Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
RuntimeError: Something bad happened
```

Se puede encontrar información adicional sobre excepciones en la sección Excepciones, e información sobre manejo de excepciones en la sección La sentencia try.

Distinto en la versión 3.3: **`None`** ahora es permitido como `Y` en `raise X from Y`.

Added the **`__suppress_context__`** attribute to suppress automatic display of the exception context.

Distinto en la versión 3.11: Si el rastreo de la excepción activa se modifica en una cláusula **`except`**, una instrucción `raise` posterior vuelve a generar la excepción con el rastreo modificado. Anteriormente, la excepción se volvía a generar con el rastreo que tenía cuando se detectó.

7.9. La declaración `break`

```
break_stmt: "break"
```

`break` solo puede ocurrir sintácticamente anidado en un bucle **`for`** o **`while`**, pero no anidado en una función o definición de clase dentro de ese bucle.

Termina el bucle adjunto más cercano, omitiendo la cláusula `else` opcional si el bucle tiene una.

Si un bucle `for` es terminado por `break`, el objetivo de control de bucle mantiene su valor actual.

Cuando `break` pasa el control de una sentencia `try` con una cláusula `finally`, esa cláusula `finally` se ejecuta antes de dejar realmente el bucle.

7.10. La declaración `continue`

```
continue_stmt:  "continue"
```

`continue` sólo puede ocurrir sintácticamente anidado en el ciclo `for` o `while`, pero no anidado en una función o definición de clase dentro de ese ciclo. Continúa con la siguiente iteración del bucle envolvente más cercano.

Cuando `continue` pasa el control de una sentencia `try` con una cláusula `finally`, esa cláusula `finally` se ejecuta antes de empezar realmente el siguiente ciclo del bucle.

7.11. La declaración `import`

```
import_stmt:     "import" module ["as" identifier] (","
                 | "from" relative_module "import" ident
                 ("," identifier ["as" identifier])*
                 | "from" relative_module "import" "(" i
                 ("," identifier ["as" identifier])* [",
                 | "from" relative_module "import" "*"
module:          (identifier ".")* identifier
relative_module: "."* module | "."+
```

La declaración básica de importación (sin la cláusula `from`) es ejecutada en 2 pasos:

1. encontrar un módulo, cargarlo e inicializarlo en caso de ser necesario
2. define un nombre o nombres en el espacio de nombres local para el alcance donde ocurre la instrucción `import`.

Cuando la declaración contiene varias cláusulas (separadas por comas), los dos pasos se llevan a cabo por separado para cada cláusula, como si las cláusulas se hubieran separado en declaraciones de importación individuales.

The details of the first step, finding and loading modules, are described in greater detail in the section on the import system, which also describes the various types of packages and modules that can be imported, as well as all the hooks that can be used to customize the import system. Note that failures in this step may indicate either that the module could not be located, *or* that an error occurred while initializing the module, which includes execution of the module's code.

Si el módulo solicitado se recupera correctamente, estará disponible en el espacio de nombres local de una de estas tres formas:

- Si el nombre del módulo va seguido de `as``, entonces el nombre siguiente `as` está vinculado directamente al módulo importado.
- Si no se especifica ningún otro nombre y el módulo que se está importando es un módulo de nivel superior, el nombre del módulo se enlaza en el espacio de nombres local como una referencia al módulo importado
- Si el módulo que se está importando *no* es un módulo de nivel superior, entonces el nombre del paquete de nivel superior que contiene el módulo se enlaza en el espacio de nombres local como una referencia al paquete de nivel superior. Se debe acceder al módulo importado utilizando su nombre calificado completo en lugar de directamente

La forma `from` usa un complejo un poco más complicado:

1. encuentra el módulo especificado en la cláusula `from`, cargando e inicializándolo si es necesario;
2. para cada uno de los identificadores especificados en la

cláusula **`import`**:

1. compruebe si el módulo importado tiene un atributo con ese nombre
2. de lo contrario, intente importar un submódulo con ese nombre y luego verifique el módulo importado nuevamente para ese atributo
3. si el atributo no se encuentra, **`ImportError`** es lanzada.
4. de lo contrario, una referencia a ese valor se almacena en el espacio de nombres local, usando el nombre en la cláusula **`as`** si ésta está presente, de lo contrario usando el nombre del atributo

Ejemplos:

```
import foo                    # foo imported and bound loca
import foo.bar.baz            # foo, foo.bar, and foo.bar.b
import foo.bar.baz as fbb     # foo, foo.bar, and foo.bar.b
from foo.bar import baz       # foo, foo.bar, and foo.bar.b
from foo import attr          # foo imported and foo.attr b
```

Si la lista de identificadores se reemplaza por una estrella (`'*'`), todos los nombres públicos definidos en el módulo se enlazan en el espacio de nombres local para el ámbito donde ocurre la declaración **`import`**.

The *public names* defined by a module are determined by checking the module's namespace for a variable named `__all__`; if defined, it must be a sequence of strings which are names defined or imported by that module. Names containing non-ASCII characters must be in the normalization form [https://www.unicode.org/reports/tr15/#Norm_Forms] NFKC; see Non-ASCII characters in names for details. The names given in `__all__` are all considered public and are required to exist. If `__all__` is not defined, the set of public names includes all names found in the module's namespace which do not begin with an underscore character (`'_'`). `__all__` should contain the entire public API. It is intended to avoid accidentally exporting items that are not part of the API (such as library modules which were imported and used within the module).

La forma de importación comodín — `from module import *` —

sólo se permite a nivel módulo. Intentar usarlo en una definición de clase o función lanza una `SyntaxError`.

Al especificar qué módulo importar, no es necesario especificar el nombre absoluto del módulo. Cuando un módulo o paquete está contenido dentro de otro paquete, es posible realizar una importación relativa dentro del mismo paquete superior sin tener que mencionar el nombre del paquete. Al usar puntos iniciales en el módulo o paquete especificado después de `from`, puede especificar qué tan alto recorrer la jerarquía actual del paquete sin especificar nombres exactos. Un punto inicial significa el paquete actual donde existe el módulo que realiza la importación. Dos puntos significan un nivel de paquete. Tres puntos son dos niveles, etc. Entonces, si ejecuta `from . import mod` de un módulo en el paquete `pkg` terminará importando `pkg.mod`. Si ejecuta `from ..subpkg2 import mod` desde dentro de `pkg.subpkg1`, importará `pkg.subpkg2.mod`. La especificación para las importaciones relativas está contenida en la sección Paquete Importaciones relativas.

`importlib.import_module()` se proporciona para soportar aplicaciones que determinan dinámicamente los módulos a cargar.

Lanza un auditing event `import` con argumentos `module`, `filename`, `sys.path`, `sys.meta_path`, `sys.path_hooks`.

7.11.1. Declaraciones Futuras

Una *future statement* es una directiva para el compilador para indicar que un módulo en particular debe compilarse usando la sintaxis o semántica que estará disponible en una versión futura específica de Python donde la característica se convierte en estándar.

La declaración futura está destinada a facilitar la migración a futuras versiones de Python que introducen cambios incompatibles en el lenguaje. Permite el uso de las nuevas funciones por módulo antes del lanzamiento en el que la función se convierte en estándar.

```
future_stmt: "from" "__future__" "import" feature ["as"
```

```
                  ("," feature ["as" identifier])*
                  | "from" "__future__" "import" "(" feature
                  ("," feature ["as" identifier])* [","] ")"
feature:          identifier
```

Una declaración futura debe aparecer cerca de la parte superior del módulo. Las únicas líneas que pueden aparecer antes de una declaración futura son:

- el docstring del módulo (si hay),
- comentarios,
- lineas en blanco, y
- otras declaraciones futuras.

La única característica en Python 3.7 que requiere el uso la declaración futuro es `annotations`.

108/5000 Python 3 aún reconoce todas las características históricas habilitadas por la declaración futura. La lista incluye `absolute_import`, `division`, `generators`, `generator_stop`, `unicode_literals`, `print_function`, `nested_scopes` y `with_statement`. Todos son redundantes porque siempre están habilitados y solo se conservan para compatibilidad con versiones anteriores.

Una declaración futura se reconoce y se trata especialmente en el momento de la compilación: los cambios en la semántica de las construcciones centrales a menudo se implementan generando código diferente. Incluso puede darse el caso de que una nueva característica introduzca una nueva sintaxis incompatible (como una nueva palabra reservada), en cuyo caso el compilador puede necesitar analizar el módulo de manera diferente. Tales decisiones no pueden postergarse hasta el tiempo de ejecución.

Para cualquier versión dada, el compilador sabe qué nombres de características se han definido y lanza un error en tiempo de compilación si una declaración futura contiene una característica que no conoce.

La semántica del tiempo de ejecución directo es la misma que para cualquier declaración de importación: hay un módulo estándar `__future__`, que se describe más adelante, y se importará de la forma habitual en el momento en que se ejecute la declaración futura.

La interesante semántica del tiempo de ejecución depende de la característica específica habilitada por la declaración futura.

Notar que no hay nada especial a cerca de la declaración:

```
import __future__ [as name]
```

Esa no es una declaración futura; es una declaración de importación ordinaria sin restricciones especiales de semántica o sintaxis.

Code compiled by calls to the built-in functions `exec()` and `compile()` that occur in a module `M` containing a future statement will, by default, use the new syntax or semantics associated with the future statement. This can be controlled by optional arguments to `compile()` — see the documentation of that function for details.

Una declaración futura escrita en una prompt interactiva del intérprete entrará en vigencia durante el resto de la sesión de dicho intérprete. Si un intérprete se inicia con la opción `-i`, se le pasa un nombre de script para ejecutar, y el script incluye una declaración futura, ésta estará en efecto en la sesión interactiva iniciada después de que se ejecute el script.

Ver también

PEP 236 [https://peps.python.org/pep-0236/] - Vuelta al _future_
La propuesta original para el mecanismo _future_.

7.12. La declaración `global`

```
global_stmt: "global" identifier ("," identifier)*
```

The `global` statement causes the listed identifiers to be interpreted as globals. It would be impossible to assign to a global variable without `global`, although free variables may refer to globals without being declared global.

The `global` statement applies to the entire current scope (module, function body or class definition). A `SyntaxError` is raised if a variable is used or assigned to prior to its global declaration in the scope.

At the module level, all variables are global, so a `global` statement has no effect. However, variables must still not be used or assigned to prior to their `global` declaration. This requirement is relaxed in the interactive prompt (REPL).

** Nota del programador: ** `global` es una directiva para el analizador. Se aplica solo al código analizado al mismo tiempo que la declaración `global`. En particular, una declaración `global` contenida en una cadena u objeto de código suministrado a la función incorporada `exec()` no afecta el bloque de código *que contiene* la llamada a la función, y el código contenido en dicha función una cadena no se ve afectada por la declaración `global` en el código que contiene la llamada a la función. Lo mismo se aplica a las funciones `eval()` y `compile()`.

7.13. La declaración `nonlocal`

```
nonlocal_stmt: "nonlocal" identifier ("," identifier)*
```

When the definition of a function or class is nested (enclosed) within the definitions of other functions, its nonlocal scopes are the local scopes of the enclosing functions. The `nonlocal` statement causes the listed identifiers to refer to names previously bound in nonlocal scopes. It allows encapsulated code to rebind such nonlocal identifiers. If a name is bound in more than one nonlocal scope, the nearest binding is used. If a name is not bound in any nonlocal scope, or if there is no nonlocal scope, a `SyntaxError` is raised.

The `nonlocal` statement applies to the entire scope of a function or class body. A `SyntaxError` is raised if a variable is used or

assigned to prior to its nonlocal declaration in the scope.

Ver también

PEP 3104 [https://peps.python.org/pep-3104/] - Acceso a Nombres de Ámbitos externos
La especificación para la declaración `nonlocal`.

Programmer's note: `nonlocal` is a directive to the parser and applies only to code parsed along with it. See the note for the `global` statement.

7.14. The `type` statement

```
type_stmt: 'type' identifier [type_params] "=" expressio
```

The `type` statement declares a type alias, which is an instance of `typing.TypeAliasType`.

For example, the following statement creates a type alias:

```
type Point = tuple[float, float]
```

This code is roughly equivalent to:

```
annotation-def VALUE_OF_Point():
    return tuple[float, float]
Point = typing.TypeAliasType("Point", VALUE_OF_Point())
```

`annotation-def` indicates an annotation scope, which behaves mostly like a function, but with several small differences.

The value of the type alias is evaluated in the annotation scope. It is not evaluated when the type alias is created, but only when the value is accessed through the type alias's `__value__` attribute (see Evaluación perezosa). This allows the type alias to refer to names that are not yet defined.

Type aliases may be made generic by adding a type parameter list after the name. See Alias de tipo genérico for more.

`type` is a soft keyword.

Added in version 3.12.

Ver también

PEP 695 [https://peps.python.org/pep-0695/] - Type Parameter Syntax

Introduced the `type` statement and syntax for generic classes and functions.

[1] [2] [3] [4] [5]

8. Sentencias compuestas

Las sentencias compuestas contienen (grupos de) otras sentencias; estas afectan o controlan la ejecución de esas otras sentencias de alguna manera. En general, las sentencias compuestas abarcan varias líneas, aunque en representaciones simples una sentencia compuesta completa puede estar contenida en una línea.

Las sentencias `if`, `while` y `for` implementan construcciones de control de flujo tradicionales. `try` especifica gestores de excepción o código de limpieza para un grupo de sentencias, mientras que las sentencias `with` permite la ejecución del código de inicialización y finalización alrededor de un bloque de código. Las definiciones de función y clase también son sentencias sintácticamente compuestas.

Una sentencia compuesta consta de una o más “cláusulas”. Una cláusula consta de un encabezado y una “suite”. Los encabezados de cláusula de una declaración compuesta particular están todos en el mismo nivel de indentación. Cada encabezado de cláusula comienza con una palabra clave de identificación única y termina con dos puntos. Una suite es un grupo de sentencias controladas por una cláusula. Una suite puede ser una o más sentencias simples separadas por punto y coma en la misma línea como el encabezado, siguiendo los dos puntos del encabezado, o puede ser una o puede ser una o más declaraciones indentadas en líneas posteriores. Solo la última forma de una suite puede contener sentencias compuestas anidadas; lo siguiente es ilegal, principalmente porque no estaría claro a qué cláusula `if` seguido de la cláusula `else` hace referencia:

```
if test1: if test2: print(x)
```

También tenga en cuenta que el punto y coma se une más apretado que los dos puntos en este contexto, de modo que en el siguiente ejemplo, todas o ninguna de las llamadas `print()` se ejecutan:

```
if x < y < z: print(x); print(y); print(z)
```

Resumiendo:

```
compound_stmt: if_stmt
               | while_stmt
               | for_stmt
               | try_stmt
               | with_stmt
               | match_stmt
               | funcdef
               | classdef
               | async_with_stmt
               | async_for_stmt
               | async_funcdef
suite:         stmt_list NEWLINE | NEWLINE INDENT statem
statement:     stmt_list NEWLINE | compound_stmt
stmt_list:     simple_stmt (";" simple_stmt)* [";"]
```

Tenga en cuenta que las sentencias siempre terminan en un `NEWLINE` posiblemente seguida de `DEDENT`. También tenga en cuenta que las cláusulas de continuación opcionales siempre comienzan con una palabra clave que no puede iniciar una sentencia, por lo tanto, no hay ambigüedades (el problema de "colgado `if`" se resuelve en Python al requerir que las sentencias anidadas `if` deben estar indentadas).

El formato de las reglas gramaticales en las siguientes secciones coloca cada cláusula en una línea separada para mayor claridad.

8.1. La sentencia `if`

La sentencia `if` se usa para la ejecución condicional:

```
if_stmt: "if" assignment_expression ":" suite
         ("elif" assignment_expression ":" suite)*
         ["else" ":" suite]
```

Selecciona exactamente una de las suites evaluando las expresiones una por una hasta que se encuentre una verdadera (vea la sección Operaciones booleanas para la definición de verdadero y falso); entonces esa suite se ejecuta (y ninguna otra parte de la sentencia `if` se ejecuta o evalúa). Si todas las expresiones son falsas, se ejecuta la suite de cláusulas `else`, si está presente.

8.2. La sentencia `while`

La sentencia `while` se usa para la ejecución repetida siempre que una expresión sea verdadera:

```
while_stmt: "while" assignment_expression ":" suite
            ["else" ":" suite]
```

Esto prueba repetidamente la expresión y, si es verdadera, ejecuta la primera suite; si la expresión es falsa (que puede ser la primera vez que se prueba), se ejecuta el conjunto de cláusulas `else`, si está presente, y el bucle termina.

La sentencia `break` ejecutada en la primer suite termina el bucle sin ejecutar la suite de cláusulas `else`. La sentencia `continue` ejecutada en la primera suite omite el resto de la suite y vuelve a probar la expresión.

8.3. La sentencia `for`

La sentencia `for` se usa para iterar sobre los elementos de una secuencia (como una cadena de caracteres, tupla o lista) u otro objeto iterable:

```
for_stmt: "for" target_list "in" starred_expression_list
          ["else" ":" suite]
```

The `starred_expression_list` expression is evaluated once; it should yield an iterable object. An iterator is created for that iterable. The first item provided by the iterator is then assigned to the target list using the standard rules for assignments (see

Declaraciones de asignación), and the suite is executed. This repeats for each item provided by the iterator. When the iterator is exhausted, the suite in the **`else`** clause, if present, is executed, and the loop terminates.

La sentencia **`break`** ejecutada en la primera suite termina el bucle sin ejecutar el conjunto de cláusulas **`else`**. La sentencia **`continue`** ejecutada en la primera suite omite el resto de las cláusulas y continúa con el siguiente elemento, o con la cláusula **`else`** si no hay un elemento siguiente.

El bucle `for` realiza asignaciones a las variables en la lista. Esto sobrescribe todas las asignaciones anteriores a esas variables, incluidas las realizadas en la suite del bucle `for`:

```
for i in range(10):
    print(i)
    i = 5             # this will not affect the for-loo
                      # because i will be overwritten wi
                      # index in the range
```

Los nombres en la lista no se eliminan cuando finaliza el bucle, pero si la secuencia está vacía, el bucle no les habrá asignado nada. Sugerencia: el tipo incorporado **`range()`** representa secuencias aritméticas inmutables de números enteros; por ejemplo, iterar sobre `range(3)` genera sucesivamente 0, 1, y luego 2.

Distinto en la versión 3.11: Los elementos destacados ahora están permitidos en la lista de expresiones.

8.4. La sentencia `try`

La sentencia **`try`** especifica controladores de excepciones y/o código de limpieza para un grupo de sentencias:

```
try_stmt:  try1_stmt | try2_stmt | try3_stmt
try1_stmt: "try" ":" suite
           ("except" [expression ["as" identifier]] ":"
           ["else" ":" suite]
           ["finally" ":" suite]
try2_stmt: "try" ":" suite
           ("except" "*" expression ["as" identifier] ":
           ["else" ":" suite]
           ["finally" ":" suite]
try3_stmt: "try" ":" suite
           "finally" ":" suite
```

Se puede encontrar información adicional sobre las excepciones en la sección Excepciones, e información sobre el uso de la sentencia **`raise`**, para lanzar excepciones se puede encontrar en la sección La declaración raise.

Distinto en la versión 3.14: Support for optionally dropping grouping parentheses when using multiple exception types. See **PEP 758** [https://peps.python.org/pep-0758/].

8.4.1. Cláusula `except`

Las cláusulas **`except`** especifican uno o más controladores de excepciones. Cuando no se produce ninguna excepción en la cláusula **`try`**, no se ejecuta ningún controlador de excepciones. Cuando ocurre una excepción en la suite **`try`**, se inicia una búsqueda de un controlador de excepciones. Esta búsqueda inspecciona las cláusulas **`except`** a su vez hasta que se encuentra una que coincida con la excepción. Una cláusula **`except`** sin expresión, si está presente, debe ser la última; coincide con cualquier excepción.

For an **`except`** clause with an expression, the expression must evaluate to an exception type or a tuple of exception types. Parentheses can be dropped if multiple exception types are provided and the `as` clause is not used. The raised exception matches an **`except`** clause whose expression evaluates to the class or a non-virtual base class of the exception object, or to a tuple that contains such a class.

Si ninguna cláusula **`except`** coincide con la excepción, la búsqueda de un controlador de excepciones continúa en el código circundante y en la pila de invocaciones. [1]

Si la evaluación de una expresión en el encabezado de una cláusula **`except`** genera una excepción, la búsqueda original de un controlador se cancela y comienza una búsqueda de la nueva excepción en el código circundante y en la pila de llamadas (se trata como si toda la sentencia **`try`** generara la excepción).

Cuando se encuentra una cláusula **`except`** coincidente, la excepción se asigna al destino especificado después de la palabra clave **`as`** en esa cláusula **`except`**, si está presente, y se ejecuta el conjunto de la cláusula **`except`**. Todas las cláusulas **`except`** deben tener un bloque ejecutable. Cuando se alcanza el final de este bloque, la ejecución continúa normalmente después de la instrucción **`try`** completa. (Esto significa que si existen dos controladores anidados para la misma excepción y la excepción se produce en la cláusula **`try`** del controlador interno, el controlador externo no controlará la excepción).

Cuando se ha asignado una excepción mediante `as target`, se borra al final de la cláusula **`except`**. Esto es como si

```
except E as N:
    foo
```

fue traducido a

```
except E as N:
    try:
        foo
    finally:
        del N
```

Esto significa que la excepción debe asignarse a un nombre diferente para poder hacer referencia a ella después de la cláusula **`except`**. Las excepciones se borran porque con el rastreo adjunto, forman un ciclo de referencia con el marco de la pila, manteniendo todos los locales en ese marco vivos hasta que ocurra la próxima recolección de elementos no utilizados.

Antes de que se ejecute el conjunto de una cláusula **`except`** , la excepción se almacena en el módulo **`sys`**, donde puede ser accesible desde dentro del cuerpo de la cláusula **`except`** llamando a la **`sys.excepcion()`**. Cuando se sale de un *handler* de excepciones, la excepción almacenada en el módulo **`sys`** se restablece a su valor anterior:

```
>>> print(sys.exception())
None
>>> try:
...     raise TypeError
... except:
...     print(repr(sys.exception()))
...     try:
...          raise ValueError
...     except:
...         print(repr(sys.exception()))
...     print(repr(sys.exception()))
...
TypeError()
ValueError()
TypeError()
>>> print(sys.exception())
None
```

8.4.2. Cláusula `except*`

The **`except*`** clause(s) specify one or more handlers for groups of exceptions (**`BaseExceptionGroup`** instances). A **`try`** statement can have either **`except`** or **`except*`** clauses, but not both. The exception type for matching is mandatory in the case of **`except*`**, so `except*:` is a syntax error. The type is interpreted as in the case of **`except`**, but matching is performed on the exceptions contained in the group that is being handled. An **`TypeError`** is raised if a matching type is a subclass of **`BaseExceptionGroup`**, because that would have ambiguous semantics.

When an exception group is raised in the try block, each **`except*`** clause splits (see **`split()`**) it into the subgroups of matching and non-matching exceptions. If the matching subgroup is not empty, it becomes the handled exception (the value returned from **`sys.exception()`**) and assigned to the target of the **`except*`**

clause (if there is one). Then, the body of the **`except*`** clause executes. If the non-matching subgroup is not empty, it is processed by the next **`except*`** in the same manner. This continues until all exceptions in the group have been matched, or the last **`except*`** clause has run.

After all **`except*`** clauses execute, the group of unhandled exceptions is merged with any exceptions that were raised or re-raised from within **`except*`** clauses. This merged exception group propagates on.:

```
>>> try:
...     raise ExceptionGroup("eg",
...         [ValueError(1), TypeError(2), OSError(3), OS
... except* TypeError as e:
...     print(f'caught {type(e)} with nested {e.exceptio
... except* OSError as e:
...     print(f'caught {type(e)} with nested {e.exceptio
...
caught <class 'ExceptionGroup'> with nested (TypeError(2
caught <class 'ExceptionGroup'> with nested (OSError(3),
  + Exception Group Traceback (most recent call last):
  |   File "<doctest default[0]>", line 2, in <module>
  |     raise ExceptionGroup("eg",
  |         [ValueError(1), TypeError(2), OSError(3), OS
  | ExceptionGroup: eg (1 sub-exception)
  +-+---------------- 1 ----------------
    | ValueError: 1
    +------------------------------------
```

If the exception raised from the **`try`** block is not an exception group and its type matches one of the **`except*`** clauses, it is caught and wrapped by an exception group with an empty message string. This ensures that the type of the target `e` is consistently **`BaseExceptionGroup`**:

```
>>> try:
...     raise BlockingIOError
... except* BlockingIOError as e:
...     print(repr(e))
```

```
...
ExceptionGroup('', (BlockingIOError(),))
```

`break`, `continue` and `return` cannot appear in an `except*` clause.

8.4.3. La sentencia `else`

La cláusula opcional `else` se ejecuta si el flujo de control sale de la suite `try`, no se produjo ninguna excepción, y no se ejecutó la sentencia `return`, `continue` o `break`. Las excepciones en la cláusula `else` no se gestionaron con las cláusulas precedentes `except`.

8.4.4. Cláusula `finally`

If `finally` is present, it specifies a “cleanup” handler. The `try` clause is executed, including any `except` and `else` clauses. If an exception occurs in any of the clauses and is not handled, the exception is temporarily saved. The `finally` clause is executed. If there is a saved exception it is re-raised at the end of the `finally` clause. If the `finally` clause raises another exception, the saved exception is set as the context of the new exception. If the `finally` clause executes a `return`, `break` or `continue` statement, the saved exception is discarded. For example, this function returns 42.

```
def f():
    try:
        1/0
    finally:
        return 42
```

La información de excepción no está disponible para el programa durante la ejecución de la cláusula `finally`.

Cuando se ejecuta una sentencia `return`, `break` o `continue` en el conjunto `try` de una sentencia `try`…`finally`, la cláusula `finally` también se ejecuta “al salir”.

The return value of a function is determined by the last `return` statement executed. Since the `finally` clause always executes, a `return` statement executed in the `finally` clause will always be the last one executed. The following function returns "finally".

```
def foo():
    try:
        return 'try'
    finally:
        return 'finally'
```

Distinto en la versión 3.8: Antes de Python 3.8, una declaración `continue` era ilegal en la cláusula `finally` debido a un problema con la implementación.

Distinto en la versión 3.14: The compiler emits a `SyntaxWarning` when a `return`, `break` or `continue` appears in a `finally` block (see PEP 765 [https://peps.python.org/pep-0765/]).

8.5. La sentencia `with`

La sentencia `with` se usa para ajustar la ejecución de un bloque con métodos definidos por un administrador de contexto (ver sección Gestores de Contexto en la Declaración with). Esto permite que los patrones de uso comunes `try`…`except`…`finally` se encapsulen para una reutilización conveniente.

```
with_stmt:          "with" ( "(" with_stmt_contents ","?
with_stmt_contents: with_item ("," with_item)*
with_item:          expression ["as" target]
```

La ejecución de la sentencia `with` con un «item» se realiza de la siguiente manera:

1. La expresión de contexto (la expresión dada en `with_item`) se evalúa para obtener un administrador de contexto.

2. El administrador de contexto `__enter__()` se carga para su uso posterior.

3. El administrador de contexto `__exit__()` se carga para su uso posterior.

4. Se invoca el método `__enter__()` del administrador de contexto.

5. Si se incluyó el destino en la declaración `with`, se le asigna el valor de retorno de `__enter__()`.

   Nota

   La declaración `with` garantiza que si el método `__enter__()` regresa sin error, entonces siempre se llamará a `__exit__()`. Por lo tanto, si se produce un error durante la asignación a la lista de destino, se trataría de la misma manera que si el error ocurriera dentro del suite. Consulte el paso 7 a continuación.

6. La suite se ejecuta.

7. Se invoca el método `__exit__()` del administrador de contexto. Si una excepción causó la salida de la suite, su tipo, valor y rastreo se pasan como argumentos a `__exit__()`. De lo contrario, se proporcionan tres argumentos `None`.

   Si se salió de la suite debido a una excepción, y el valor de retorno del método `__exit__()` fue falso, la excepción se vuelve a plantear. Si el valor de retorno era verdadero, la excepción se suprime y la ejecución continúa con la declaración que sigue a la declaración `with`.

   Si se salió de la suite por cualquier motivo que no sea una excepción, el valor de retorno de `__exit__()` se ignora y la ejecución continúa en la ubicación normal para el tipo de salida que se tomó.

El siguiente código:

```
with EXPRESSION as TARGET:
    SUITE
```

es semánticamente equivalente a:

```
manager = (EXPRESSION)
enter = manager.__enter__
exit = manager.__exit__
value = enter()
hit_except = False

try:
    TARGET = value
    SUITE
except:
    hit_except = True
    if not exit(*sys.exc_info()):
        raise
finally:
    if not hit_except:
        exit(None, None, None)
```

except that implicit special method lookup is used for **`__enter__()`** and **`__exit__()`**.

Con más de un elemento, los administradores de contexto se procesan como si varias sentencias **`with`** estuvieran anidadas:

```
with A() as a, B() as b:
    SUITE
```

es semánticamente equivalente a:

```
with A() as a:
    with B() as b:
        SUITE
```

También puedes escribir administradores de contexto de múltiples ítems en múltiples lineas si los ítems están entre paréntesis. Por ejemplo:

```
with (
    A() as a,
    B() as b,
):
    SUITE
```

Distinto en la versión 3.1: Soporte para múltiples expresiones de contexto.

Distinto en la versión 3.10: Soporte para el uso de paréntesis de agrupación para separar la declaración en múltiples líneas.

Ver también

PEP 343 [https://peps.python.org/pep-0343/] - La sentencia «with»
La especificación, antecedentes y ejemplos de la sentencia de Python `with`.

8.6. La sentencia `match`

Added in version 3.10.

La declaración match es usada para coincidencia de patrones. Sintaxis:

```
match_stmt:   'match' subject_expr ":" NEWLINE INDENT ca
subject_expr: `!star_named_expression` "," `!star_named_
              | `!named_expression`
case_block:   'case' patterns [guard] ":" `!block`
```

Nota

Esta sección utiliza comillas simples para denotar las palabras clave suaves.

La coincidencia de patrones toma un patrón como entrada (delante de `case`) y un valor de búsqueda (delante de `match`). El patrón (que puede contener subpatrones) es comparado con el valor de búsqueda. Los resultados son:

- Una coincidencia exitosa o fallida (también llamada éxito o fracaso de un patrón).
- Una posible vinculación de los valores coincidentes con un nombre. Los requisitos previos para esto se discuten abajo.

Las palabras clave `match` y `case` son palabras clave suaves.

Ver también

- **PEP 634** [https://peps.python.org/pep-0634/] – Coincidencia de patrones estructurales: Especificación
- **PEP 636** [https://peps.python.org/pep-0636/] – Coincidencia de patrones estructurales: Tutorial

8.6.1. Resumen

A continuación, un resumen del flujo lógico de una declaración de coincidencia:

1. Se evalúa la expresión `subject_expr` y se obtiene un valor sujeto resultante. Si la expresión contiene una coma, se construye una tupla utilizando las reglas estándar.

2. Se intenta coincidir cada patrón en un `case_block` con el valor sujeto. Las reglas específicas para el éxito o el fracaso se describen abajo. El intento de coincidencia también puede enlazar algunos o todos los nombres independientes dentro del patrón. Las reglas precisas de enlace de patrones varían según el tipo de patrón y se especifican a continuación. **Los enlaces de nombre realizados durante una coincidencia de patrones exitosa sobreviven al bloque ejecutado y se pueden usar después de la declaración de coincidencia**.

   Nota

   Durante las coincidencias de patrones fallidas, algunos subpatrones pueden tener éxito. No confíe en que los enlaces se realicen para una coincidencia fallida. Por el contrario, no confíe en que las variables permanezcan sin cambios después de una coincidencia fallida. El comportamiento exacto depende de la implementación y puede variar. Esta es una decisión intencional para permitir que diferentes implementaciones añadan optimizaciones.

3. Si el patrón es exitoso, se evalúa la protección correspondiente (si está presente). En este caso se garantiza que todos los enlaces de nombres han ocurrido.

    - Si la protección se evalúa como verdadera o no existe, se ejecuta el `block` dentro de `case_block`.
    - En caso contrario, se intenta con el siguiente `case_block` como se ha descrito anteriormente.
    - Si no hay más bloques de casos, la declaración de coincidencia se completa.

Nota

Por lo general, los usuarios no deben confiar en que se evalúe un patrón. Dependiendo de la implementación, el intérprete puede almacenar en caché los valores o utilizar otras optimizaciones que omitan las evaluaciones repetidas.

Un ejemplo de declaración de coincidencia:

```
>>> flag = False
>>> match (100, 200):
...     case (100, 300):  # No coinciden: 200 != 300
...         print('Case 1')
...     case (100, 200) if flag:  # Coinciden, pero la gu
...         print('Case 2')
...     case (100, y):  # Coinciden, y vincula `y` a 200
...         print(f'Case 3, y: {y}')
...     case _:  # Patrón no se intentó
...         print('Case 4, I match anything!')
...
Case 3, y: 200
```

En este caso, `if flag` es una protección. Lea más sobre eso en la siguiente sección.

8.6.2. Protecciones

```
guard: "if" `!named_expression`
```

Una `guard` (que es parte del `case`) debe ser exitosa para que el código dentro de `case` sea ejecutado. Toma la forma: `if` seguida de una expresión.

El flujo lógico de un bloque `case` con una `guard` es el siguiente:

1. Se comprueba que el patrón del bloque `case` fue exitoso. Si el patrón falló, el `guard` no se evalúa y se comprueba el siguiente bloque `case`.
2. Si el patrón tuvo éxito, se evalúa el `guard`.
    - Si la condición del `guard` es verdadera, se selecciona el bloque de ese caso.
    - Si la condición del `guard` es falsa, el bloque de ese caso no es seleccionado.
    - Si el `guard` lanza una excepción durante la evaluación, se levanta la excepción.

Se permite que las protecciones tengan efectos secundarios, ya que son expresiones. La evaluación de la protección debe ir desde el primer al último bloque de casos, uno a la vez, saltando los bloques de casos cuyo(s) patrón(es) no tenga(n) éxito. (Es decir, la evaluación de las protecciones debe realizarse en orden.) La evaluación de las protecciones debe detenerse una vez que se selecciona un bloque de casos.

8.6.3. Bloques de Casos Irrefutables

Un bloque de casos irrefutable es un bloque de casos que coincide con todo. Una declaración de coincidencia puede tener como máximo un bloque de casos irrefutable, y debe ser el último.

Un bloque de casos se considera irrefutable si no tiene protección y su patrón es irrefutable. Un patrón se considera irrefutable si podemos demostrar, solo por su sintaxis, que siempre tendrá éxito. Solo los siguientes patrones son irrefutables:

- patrones AS cuyo lado izquierdo es irrefutable
- Patrones OR que contienen al menos un patrón irrefutable
- Patrones de captura
- Patrones comodín
- patrones irrefutables entre paréntesis

8.6.4. Patrones

Nota

Esta sección utiliza notaciones gramaticales más allá del estándar EBNF:

- la notación `SEP.RULE+` es la abreviación de `RULE (SEP RULE)*`
- la notación `!RULE` es la abreviación de una aserción de anticipación negativa

La sintaxis de nivel superior para `patrones` es:

```
patterns:       open_sequence_pattern | pattern
pattern:        as_pattern | or_pattern
closed_pattern: | literal_pattern
                | capture_pattern
                | wildcard_pattern
                | value_pattern
                | group_pattern
                | sequence_pattern
                | mapping_pattern
                | class_pattern
```

Las descripciones a continuación incluirán una descripción «en términos simples» de lo que hace un patrón con fines ilustrativos (créditos a Raymond Hettinger por un documento que inspiró la mayoría de las descripciones). Tenga en cuenta que estas descripciones tienen únicamente fines ilustrativos y que **may not** refleja la implementación subyacente. Además, no cubren todos los formularios válidos.

8.6.4.1. Patrones OR

Un patrón OR son dos o más patrones separados por barras verticales `|`. Sintaxis:

```
or_pattern: "|".closed_pattern+
```

Solo el subpatrón final puede ser irrefutable, y cada subpatrón debe vincular el mismo conjunto de nombres para evitar ambigüedades.

Un patrón OR hace coincidir cada uno de sus subpatrones a su vez con el valor del sujeto, hasta que uno tiene éxito. Entonces, el patrón OR se considera exitoso. De lo contrario, si ninguno de los subpatrones tiene éxito, el patrón OR falla.

En términos simples, `P1 | P2 | ...` intentará igualar `P1`, si falla, intentará igualar `P2`, teniendo éxito inmediatamente si alguno tiene éxito, fallando en caso contrario.

8.6.4.2. patrones AS

Un patrón AS coincide con un patrón OR a la izquierda de la palabra clave `as` con un sujeto. Sintaxis:

```
as_pattern: or_pattern "as" capture_pattern
```

Si el patrón OR falla, el patrón AS falla. De lo contrario, el patrón AS vincula al sujeto con el nombre a la derecha de la palabra clave as y tiene éxito. `capture_pattern` no puede ser un `_`.

En términos simples, `P as NAME` coincidirá con `P` y, en caso de éxito, establecerá `NAME = <subject>`.

8.6.4.3. Patrones literales

Un patrón literal corresponde a la mayoría de literals en Python. Sintaxis:

```
literal_pattern: signed_number
                 | signed_number "+" NUMBER
                 | signed_number "-" NUMBER
                 | strings
                 | "None"
                 | "True"
                 | "False"
signed_number:   ["-"] NUMBER
```

The rule `strings` and the token `NUMBER` are defined in the standard Python grammar. Triple-quoted strings are supported. Raw strings and byte strings are supported. f-strings and t-strings are not supported.

Los formularios `signed_number '+' NUMBER` y `signed_number '-' NUMBER` son para expresar complex numbers; requieren un número real a la izquierda y un número imaginario a la derecha. P.ej. `3 + 4j`.

En términos simples, `LITERAL` solo tendrá éxito si `<subject> == LITERAL`. Para los singleton `None`, `True` y `False`, se utiliza el operador **`is`**.

8.6.4.4. Patrones de captura

Un patrón de captura vincula el valor del sujeto a un nombre. Sintaxis:

```
capture_pattern: !'_' NAME
```

Un solo guión bajo `_` no es un patrón de captura (esto es lo que expresa `!'_'`). En su lugar, se trata como un **`wildcard_pattern`**.

En un patrón dado, un nombre dado solo se puede vincular una vez. P.ej. `case x, x: ...` no es válido mientras `case [x] | x: ...` está permitido.

Los patrones de captura siempre tienen éxito. El enlace sigue las reglas de alcance establecidas por el operador de expresión de asignación en **PEP 572** [https://peps.python.org/pep-0572/]; el nombre se convierte en una variable local en el alcance de la función contenedora más cercana, a menos que haya una declaración **`global`** o **`nonlocal`** aplicable.

En términos simples, `NAME` siempre tendrá éxito y establecerá `NAME = <subject>`.

8.6.4.5. Patrones comodín

Un patrón comodín siempre tiene éxito (coincide con cualquier cosa) y no vincula ningún nombre. Sintaxis:

```
wildcard_pattern: '_'
```

_ es un soft keyword dentro de cualquier patrón, pero solo dentro de patrones. Es un identificador, como de costumbre, incluso dentro de las expresiones de sujeto `match`, `guard` sy bloques `case`.

En términos simples, _ siempre tendrá éxito.

8.6.4.6. Patrones de valor

Un patrón de valor representa un valor con nombre en Python. Sintaxis:

```
value_pattern: attr
attr:          name_or_attr "." NAME
name_or_attr:  attr | NAME
```

El nombre con puntos en el patrón se busca usando el estándar Python name resolution rules. El patrón tiene éxito si el valor encontrado se compara con el valor del sujeto (usando el operador de igualdad `==`).

En términos simples, `NAME1.NAME2` solo tendrá éxito si `<subject> == NAME1.NAME2`

Nota

Si el mismo valor ocurre varias veces en la misma declaración de coincidencia, el intérprete puede almacenar en caché el primer valor encontrado y reutilizarlo en lugar de repetir la misma búsqueda. Este caché está estrictamente vinculado a una ejecución determinada de una declaración de coincidencia determinada.

8.6.4.7. Patrones de grupo

Un patrón de grupo permite a los usuarios agregar paréntesis alrededor de los patrones para enfatizar la agrupación deseada. De lo contrario, no tiene sintaxis adicional. Sintaxis:

```
group_pattern: "(" pattern ")"
```

En términos simples, `(P)` tiene el mismo efecto que `P`.

8.6.4.8. Patrones de secuencia

Un patrón de secuencia contiene varios subpatrones para hacer coincidir con elementos de secuencia. La sintaxis es similar al desempaquetado de una lista o tupla.

```
sequence_pattern:       "[" [maybe_sequence_pattern] "]"
                        | "(" [open_sequence_pattern] ")
open_sequence_pattern:  maybe_star_pattern "," [maybe_se
maybe_sequence_pattern: ",".maybe_star_pattern+ ","?
maybe_star_pattern:     star_pattern | pattern
star_pattern:           "*" (capture_pattern | wildcard_
```

No hay diferencia si se utilizan paréntesis o corchetes para los patrones de secuencia (es decir, `(...)` vs `[...]`).

Nota

Un solo patrón encerrado entre paréntesis sin una coma final (por ejemplo, `(3 | 4)`) es un group pattern. Mientras que un solo patrón encerrado entre corchetes (por ejemplo, `[3 | 4]`) sigue siendo un patrón de secuencia.

A lo sumo, un subpatrón de estrella puede estar en un patrón de secuencia. El subpatrón de estrella puede ocurrir en cualquier posición. Si no hay ningún subpatrón de estrella, el patrón de secuencia es un patrón de secuencia de longitud fija; de lo contrario, es un patrón de secuencia de longitud variable.

El siguiente es el flujo lógico para hacer coincidir un patrón de secuencia con un valor de sujeto:

1. Si el valor del sujeto no es una secuencia [2], el patrón de secuencia falla.

2. Si el valor del sujeto es una instancia de `str`, `bytes` o `bytearray`, el patrón de secuencia falla.

3. Los pasos subsiguientes dependen de si el patrón de secuencia es de longitud fija o variable.

   Si el patrón de secuencia es de longitud fija:

   1. Si la longitud de la secuencia del sujeto no es igual al número de subpatrones, el patrón de secuencia falla
   2. Los subpatrones del patrón de secuencia se hacen coincidir con sus elementos correspondientes en la secuencia del sujeto de izquierda a derecha. El emparejamiento se detiene tan pronto como falla un subpatrón. Si todos los subpatrones tienen éxito en hacer coincidir su elemento correspondiente, el patrón de secuencia tiene éxito.

   De lo contrario, si el patrón de secuencia es de longitud variable:

1. Si la longitud de la secuencia del sujeto es menor que el número de subpatrones sin estrella, el patrón de secuencia falla.
2. Los subpatrones principales no en estrella se emparejan con sus elementos correspondientes como para las secuencias de longitud fija.
3. Si el paso anterior tiene éxito, el subpatrón en estrella coincide con una lista formada por los elementos restantes del sujeto, excluyendo los elementos restantes correspondientes a los subpatrones que no son en estrella que siguen el subpatrón en estrella.
4. Los subpatrones restantes que no son estrellas se emparejan con sus elementos temáticos correspondientes, como para una secuencia de longitud fija.

Nota

The length of the subject sequence is obtained via `len()` (i.e. via the `__len__()` protocol). This length may be cached by the interpreter in a similar manner as value patterns.

En términos simples, `[P1, P2, P3, … , P<N>]` solo coincide si ocurre todo lo siguiente:

- comprobar que `<subject>` es una secuencia
- `len(subject) == <N>`
- `P1` coincide con `<subject>[0]` (tenga en cuenta que esta coincidencia también puede vincular nombres)
- `P2` coincide con `<subject>[1]` (tenga en cuenta que esta coincidencia también puede vincular nombres)
- … y así sucesivamente para el patrón/elemento correspondiente.

8.6.4.9. Patrones de mapeo

Un patrón de asignación contiene uno o más patrones clave-valor. La sintaxis es similar a la construcción de un diccionario. Sintaxis:

```
mapping_pattern:     "{" [items_pattern] "}"
items_pattern:       ",".key_value_pattern+ ","?
key_value_pattern:   (literal_pattern | value_pattern) "
                     | double_star_pattern
double_star_pattern: "**" capture_pattern
```

Como máximo, un patrón de estrella doble puede estar en un patrón de mapeo. El patrón de estrella doble debe ser el último subpatrón del patrón de mapeo.

No se permiten claves duplicadas en patrones de mapeo. Las claves literales duplicadas lanzarán un `SyntaxError`. Dos claves que de otro modo tendrían el mismo valor lanzarán un `ValueError` en tiempo de ejecución.

El siguiente es el flujo lógico para hacer coincidir un patrón de mapeo con un valor de sujeto:

1. Si el valor del sujeto no es una asignación [3], el patrón de asignación falla.
2. Si cada clave dada en el patrón de mapeo está presente en el mapeo del sujeto, y el patrón para cada clave coincide con el elemento correspondiente del mapeo del sujeto, el patrón de mapeo tiene éxito.
3. Si se detectan claves duplicadas en el patrón de mapeo, el patrón se considera inválido. Se lanza un `SyntaxError` para valores literales duplicados; o un `ValueError` para claves con nombre del mismo valor.

Nota

Key-value pairs are matched using the two-argument form of the mapping subject's `get()` method. Matched key-value pairs must already be present in the mapping, and not created on-the-fly via `__missing__()` or `__getitem__()`.

En términos simples, `{KEY1: P1, KEY2: P2, ... }` solo coincide si ocurre todo lo siguiente:

- comprobar `<subject>` es un mapeo
- `KEY1 in <subject>`
- `P1` coincide con `<subject>[KEY1]`
- … y así sucesivamente para el par correspondiente de KEY / patrón.

8.6.4.10. Patrones de clase

Un patrón de clase representa una clase y sus argumentos posicionales y de palabras clave (si los hay). Sintaxis:

```
class_pattern:       name_or_attr "(" [pattern_arguments
pattern_arguments:   positional_patterns ["," keyword_pa
                     | keyword_patterns
positional_patterns: ",".pattern+
keyword_patterns:    ",".keyword_pattern+
keyword_pattern:     NAME "=" pattern
```

La misma palabra clave no debe repetirse en los patrones de clase.

El siguiente es el flujo lógico para hacer coincidir un patrón de clase con un valor de materia:

1. Si `name_or_attr` no es una instancia del `type` incorporado, genere `TypeError`.

2. Si el valor del sujeto no es una instancia de `name_or_attr` (probado a través de `isinstance()`), el patrón de clase falla.

3. Si no hay argumentos de patrón, el patrón tiene éxito. De lo contrario, los pasos siguientes dependen de si están presentes patrones de argumentos de posición o de palabras clave.

   Para varios tipos integrados (especificados a continuación), se acepta un único subpatrón posicional que coincidirá con todo el tema; para estos tipos, los patrones de palabras clave también funcionan como para otros tipos.

Si solo hay patrones de palabras clave, se procesan de la siguiente manera, uno por uno:

I. The keyword is looked up as an attribute on the subject.
    - Si esto lanza una excepción distinta de **`AttributeError`**, la excepción aparece.
    - Si esto lanza **`AttributeError`**, el patrón de clase ha fallado.
    - De lo contrario, el subpatrón asociado con el patrón de palabra clave se compara con el valor del atributo del sujeto. Si esto falla, el patrón de clase falla; si esto tiene éxito, la coincidencia continúa con la siguiente palabra clave.

V. If all keyword patterns succeed, the class pattern succeeds.

Si hay algún patrón posicional presente, se convierte en patrones de palabras clave utilizando el atributo **`__match_args__`** en la clase `name_or_attr` antes de hacer coincidir:

I. The equivalent of `getattr(cls, "__match_args__", ())` is called.

    - Si esto lanza una excepción, la excepción surge.
    - Si el valor retornado no es una tupla, la conversión falla y se lanza **`TypeError`**.
    - Si hay más patrones posicionales que `len(cls.__match_args__)`, se lanza **`TypeError`**.
    - De lo contrario, el patrón posicional `i` se convierte en un patrón de palabra clave utilizando `__match_args__[i]` como palabra clave. `__match_args__[i]` debe ser una cadena; si no, se lanza **`TypeError`**.
    - Si hay palabras clave duplicadas, se lanza **`TypeError`**.

Ver también

Personalización de argumentos posicionales en la coincidencia de patrones de clase

VII. Once all positional patterns have been converted to keyword patterns, the match proceeds as if there were only keyword patterns.

Para los siguientes tipos integrados, el manejo de subpatrones posicionales es diferente:

- `bool`
- `bytearray`
- `bytes`
- `dict`
- `float`
- `frozenset`
- `int`
- `list`
- `set`
- `str`
- `tuple`

Estas clases aceptan un solo argumento posicional, y el patrón allí se compara con el objeto completo en lugar de con un atributo. Por ejemplo, `int(0|1)` coincide con el valor `0`, pero no con el valor `0.0`.

En términos simples, `CLS(P1, attr=P2)` solo coincide si ocurre lo siguiente:

- `isinstance(<subject>, CLS)`
- convierta `P1` en un patrón de palabra clave usando `CLS.__match_args__`
- Para cada argumento de palabra clave `attr=P2`:
  - ○ `hasattr(<subject>, "attr")`
  - ○ `P2` coincide con `<subject>.attr`
- … y así sucesivamente para el par de patrón / argumento de palabra clave correspondiente.

Ver también

- **PEP 634** [https://peps.python.org/pep-0634/] – Coincidencia de patrones estructurales: Especificación
- **PEP 636** [https://peps.python.org/pep-0636/] – Coincidencia de patrones estructurales: Tutorial

8.7. Definiciones de funciones

Una definición de función define una función objeto determinada por el usuario (consulte la sección Jerarquía de tipos estándar):

```
funcdef:                    [decorators] "def" funcname [
                            ["->" expression] ":" suite
decorators:                 decorator+
decorator:                  "@" assignment_expression NEW
parameter_list:             defparameter ("," defparamete
                              | parameter_list_no_posonly
parameter_list_no_posonly:  defparameter ("," defparamete
                            | parameter_list_starargs
parameter_list_starargs:    "*" [star_parameter] ("," def
                            | "*" ("," defparameter)+ [",
                            | parameter_star_kwargs
parameter_star_kwargs:      "**" parameter [","]
parameter:                  identifier [":" expression]
star_parameter:             identifier [":" ["*"] express
defparameter:               parameter ["=" expression]
funcname:                   identifier
```

Una definición de función es una sentencia ejecutable. Su ejecución vincula el nombre de la función en el espacio de nombres local actual a un objeto de función (un contenedor alrededor del código ejecutable para la función). Este objeto de función contiene una referencia al espacio de nombres global actual como el espacio de nombres global que se utilizará cuando se llama a la función.

La definición de la función no ejecuta el cuerpo de la función; esto se ejecuta solo cuando se llama a la función. [4]

Una definición de función puede estar envuelta por una o más expresiones decorator. Las expresiones de decorador se evalúan cuando se define la función, en el ámbito que contiene la definición de la función. El resultado debe ser invocable, la cual se invoca con el objeto de función como único argumento. El valor retornado está vinculado al nombre de la función en lugar del objeto de la función. Se aplican múltiples decoradores de forma anidada. Por ejemplo, el siguiente código

```
@f1(arg)
@f2
def func(): pass
```

es más o menos equivalente a

```
def func(): pass
func = f1(arg)(f2(func))
```

excepto que la función original no está vinculada temporalmente al nombre `func`.

Distinto en la versión 3.9: Las funciones se pueden decorar con cualquier **`assignment_expression`** válido. Anteriormente, la gramática era mucho más restrictiva; ver **PEP 614** [https://peps.python.org/pep-0614/] para más detalles.

Una lista de parámetros del tipo se puede dar entre corchetes entre el nombre de la función y el paréntesis de apertura para su lista de parámetros. Esto indica a los verificadores de tipo estático que la función es genérica. En ejecución, los parámetros de tipo pueden recuperarse del atributo **`__type_params__`**. Consulte Funciones genéricas para más información.

Distinto en la versión 3.12: Los parámetros de tipo lista son nuevos en Python 3.12.

Cuando uno o más parameters tienen la forma *parameter* = *expression*, se dice que la función tiene «valores de parámetros predeterminados». Para un parámetro con un valor predeterminado, el correspondiente argument puede omitirse desde una llamada, en cuyo caso se sustituye el valor predeterminado del parámetro. Si un parámetro tiene un valor predeterminado, todos los parámetros siguientes hasta el «`*`» también deben tener un valor predeterminado — esta es una restricción sintáctica que la gramática no expresa.

Los valores de los parámetros predeterminados se evalúan de izquierda a derecha cuando se ejecuta la definición de la función. Esto significa que la expresión se evalúa una vez, cuando se define la función, y que se utiliza el mismo valor «precalculado» para cada llamada . Esto es especialmente importante para entender cuando un parámetro predeterminado es un objeto mutable, como una lista o un diccionario: si la función modifica el objeto (por ejemplo, al agregar un elemento a una lista), el valor predeterminado está en efecto modificado. Esto generalmente no es lo que se pretendía. Una forma de evitar esto es usar `None` como valor predeterminado y probarlo explícitamente en el cuerpo de la función, por ejemplo:

```
def whats_on_the_telly(penguin=None):
    if penguin is None:
        penguin = []
    penguin.append("property of the zoo")
    return penguin
```

La semántica de llamadas de función se describe con más detalle en la sección Invocaciones. Una llamada a la función siempre asigna valores a todos los parámetros mencionados en la lista de parámetros, ya sea desde argumentos de posición, desde argumentos por palabra clave o desde valores predeterminados. Si está presente la forma «`*identifier`», se inicializa en una tupla que recibe cualquier parámetro posicional excedente, por defecto en la tupla vacía. Si el formulario «`**identifier`» está presente, se inicializa a una nueva asignación ordenada que recibe cualquier exceso de argumentos por palabra clave, por defecto a una nueva asignación vacía del mismo tipo. Los parámetros después de «`*`» o «`*identifier`» son parámetros solo por palabra clave y solo pueden pasarse con argumentos de palabras claves usadas.

Distinto en la versión 3.8: La sintaxis del parámetro de función `/` se puede utilizar para indicar parámetros de posición únicamente. Consulte **PEP 570** [https://peps.python.org/pep-0570/] para obtener más detalles.

Parameters may have an annotation of the form «`: expression`» following the parameter name. Any parameter may have an annotation, even those of the form `*identifier` or `**identifier`. (As a special case, parameters of the form `*identifier` may have an annotation «`: *expression`».) Functions may have «return» annotation of the form «`-> expression`» after the parameter list. These annotations can be any valid Python expression. The presence of annotations does not change the semantics of a function. See Annotations for more information on annotations.

Distinto en la versión 3.11: Parámetros de la forma «`*identifier`» puede tener una anotación «`: *expression`». Consulte **PEP 646** [https://peps.python.org/pep-0646/].

También es posible crear funciones anónimas (funciones no vinculadas a un nombre), para uso inmediato en expresiones. Utiliza expresiones lambda, descritas en la sección Lambdas. Tenga en cuenta que la expresión lambda es simplemente una abreviatura para una definición de función simplificada; una función definida en una sentencia «**`def`**» puede pasarse o asignarse a otro nombre al igual que una función definida por una expresión lambda. La forma «**`def`**» es en realidad más poderosa ya que permite la ejecución de múltiples sentencias y anotaciones.

Nota del programador: Las funciones son objetos de la primera-clase. Una sentencia «`def`» ejecutada dentro de una definición de función define una función local que se puede retornar o pasar. Las variables libres utilizadas en la función anidada pueden acceder a las variables locales de la función que contiene el def. Vea la sección Nombres y vínculos para más detalles.

Ver también

PEP 3107 [https://peps.python.org/pep-3107/] - Anotaciones de

funciones

La especificación original para anotaciones de funciones.

PEP 484 [https://peps.python.org/pep-0484/] - Sugerencias de tipo

Definición de un significado estándar para anotaciones: sugerencias de tipo.

PEP 526 [https://peps.python.org/pep-0526/] - Sintaxis para anotaciones variables

Capacidad para escribir anotaciones de tipo para declaraciones de variables, incluidas variables de clase y variables de instancia.

PEP 563 [https://peps.python.org/pep-0563/] - Evaluación pospuesta de anotaciones

Admite referencias directas dentro de las anotaciones conservando las anotaciones en forma de cadena de caracteres en tiempo de ejecución en lugar de una evaluación apresurada.

PEP 318 [https://peps.python.org/pep-0318/] - Decoradores para Funciones y Métodos

Decoradores de función y método se introdujeron en **PEP 3129** [https://peps.python.org/pep-3129/].

8.8. Definiciones de clase

Una definición de clase define un objeto de clase (ver sección Jerarquía de tipos estándar):

```
classdef:    [decorators] "class" classname [type_params
inheritance: "(" [argument_list] ")"
classname:   identifier
```

Una definición de clase es una sentencia ejecutable. La lista de herencia generalmente proporciona una lista de clases base (consulte Metaclases para usos más avanzados), por lo que cada elemento de la lista debe evaluar a un objeto de clase que permita la subclasificación. Las clases sin una lista de herencia heredan, por defecto, de la clase base `object`; por lo tanto,

```
class Foo:
    pass
```

es equivalente a

```
class Foo(object):
    pass
```

La suite de la clase se ejecuta en un nuevo marco de ejecución (ver Nombres y vínculos), usando un espacio de nombres local recién creado y el espacio de nombres global original. (Por lo general, el bloque contiene principalmente definiciones de funciones). Cuando la suite de la clase finaliza la ejecución, su marco de ejecución se descarta pero se guarda su espacio de nombres local. [5] Luego se crea un objeto de clase utilizando la lista de herencia para las clases base y el espacio de nombres local guardado para el diccionario de atributos. El nombre de la clase está vinculado a este objeto de clase en el espacio de nombres local original.

El orden en que se definen los atributos en el cuerpo de la clase se conserva en el `__dict__` de la nueva clase. Tenga en cuenta que esto es confiable solo justo después de crear la clase y solo para las clases que se definieron utilizando la sintaxis de definición.

La creación de clases se puede personalizar en gran medida usando metaclasses.

Las clases también se pueden decorar: al igual que cuando se decoran funciones,

```
@f1(arg)
@f2
class Foo: pass
```

es más o menos equivalente a

```
class Foo: pass
Foo = f1(arg)(f2(Foo))
```

Las reglas de evaluación para las expresiones de decorador son las mismas que para los decoradores de funciones. El resultado se vincula al nombre de la clase.

Distinto en la versión 3.9: Las clases se pueden decorar con cualquier **`assignment_expression`** válido. Anteriormente, la gramática era mucho más restrictiva; ver **PEP 614** [https://peps.python.org/pep-0614/] para más detalles.

Una lista de parámetros del tipo definida inmediatamente después de un nombre de clase debe ir entre corchetes. Esto indica a los verificadores de tipo estático que la clase es genérica. En ejecución, el tipo de parámetros puede retirarse del atributo de la clase **`__type_params__`**. Para más información consulte Clases genéricas.

Distinto en la versión 3.12: Los parámetros de tipo lista son nuevos en Python 3.12.

** Nota del programador: ** Las variables definidas en la definición de la clase son atributos de clase; son compartidos por instancias. Los atributos de instancia se pueden establecer en un método con `self.name = value`. Se puede acceder a los atributos de clase e instancia a través de la notación «`self.name`», y un atributo de instancia oculta un atributo de clase con el mismo nombre cuando se accede de esta manera. Los atributos de clase se pueden usar como valores predeterminados para los atributos de instancia, pero el uso de valores mutables puede generar resultados inesperados. Descriptors se puede usar para crear variables de instancia con diferentes detalles de implementación.

Ver también

PEP 3115 [https://peps.python.org/pep-3115/] - Metaclases en Python 3000

La propuesta que cambió la declaración de metaclases a la sintaxis actual y la semántica de cómo se construyen las clases con metaclases.

PEP 3129 [https://peps.python.org/pep-3129/] - Decoradores de clase

La propuesta que agregó decoradores de clase. Los decoradores de funciones y métodos se introdujeron en **PEP 318** [https://peps.python.org/pep-0318/].

8.9. Corrutinas

Added in version 3.5.

8.9.1. Definición de la función corrutina

```
async_funcdef: [decorators] "async" "def" funcname "(" [
               ["->" expression] ":" suite
```

La ejecución de corrutinas de Python se puede suspender y reanudar en muchos puntos (consulte coroutine). Las expresiones **`await`**, **`async for`** y **`async with`** solo se pueden utilizar en el cuerpo de una función de corrutina.

Las funciones definidas con la sintaxis `async def` siempre son funciones de corrutina, incluso si no contienen palabras claves `await` o `async`.

Es un error del tipo **`SyntaxError`** usar una expresión `yield from` dentro del cuerpo de una función de corrutina.

Un ejemplo de una función corrutina:

```
async def func(param1, param2):
    do_stuff()
    await some_coroutine()
```

Distinto en la versión 3.7: `await` y `async` ahora son palabras clave; anteriormente solo se los trataba como tales dentro del cuerpo de una función de rutina.

8.9.2. La sentencia `async for`

```
async_for_stmt: "async" for_stmt
```

Un asynchronous iterable proporciona un método `__aiter__` que retorna directamente un asynchronous iterator, que puede llamar a código asincrónico en su método `__anext__`.

La sentencia `async for` permite una iteración apropiada sobre iteradores asincrónicos.

El siguiente código:

```
async for TARGET in ITER:
    SUITE
else:
    SUITE2
```

Es semánticamente equivalente a:

```
iter = (ITER).__aiter__()
running = True

while running:
    try:
        TARGET = await iter.__anext__()
    except StopAsyncIteration:
        running = False
    else:
        SUITE
else:
    SUITE2
```

except that implicit special method lookup is used for `__aiter__()` and `__anext__()`.

Es un error del tipo `SyntaxError` usar una sentencia `async for` fuera del cuerpo de una función de corrutina.

8.9.3. La sentencia `async with`

```
async_with_stmt: "async" with_stmt
```

Un asynchronous context manager es un context manager que puede suspender la ejecución en sus métodos *enter* y *exit*.

El siguiente código:

```
async with EXPRESSION as TARGET:
    SUITE
```

es semánticamente equivalente a:

```
manager = (EXPRESSION)
aenter = manager.__aenter__
aexit = manager.__aexit__
value = await aenter()
hit_except = False

try:
    TARGET = value
    SUITE
except:
    hit_except = True
    if not await aexit(*sys.exc_info()):
        raise
finally:
    if not hit_except:
        await aexit(None, None, None)
```

except that implicit special method lookup is used for **`__aenter__()`** and **`__aexit__()`**.

Es un error del tipo **`SyntaxError`** usar una sentencia `async with` fuera del cuerpo de una función de corrutina.

Ver también

PEP 492 [https://peps.python.org/pep-0492/] - Corrutinas con sintaxis `async` y `await`

La propuesta que convirtió a las corrutinas en un concepto independiente adecuado en Python, y agregó una sintaxis de soporte.

8.10. Listas de tipo parámetro

Added in version 3.12.

Distinto en la versión 3.13: Compatibilidad para valores predeterminados se añadió (Consulte **PEP 696** [https://peps.python.org/pep-0696/]).

```
type_params:   "[" type_param ("," type_param)* "]"
type_param:    typevar | typevartuple | paramspec
typevar:       identifier (":" expression)? ("=" expressi
typevartuple:  "*" identifier ("=" expression)?
paramspec:     "**" identifier ("=" expression)?
```

Functions (incluyendo coroutines), classes y type aliases debe contener un parámetro de tipo lista:

```
def max[T](args: list[T]) -> T:
    ...

async def amax[T](args: list[T]) -> T:
    ...

class Bag[T]:
    def __iter__(self) -> Iterator[T]:
        ...

    def add(self, arg: T) -> None:
        ...

type ListOrSet[T] = list[T] | set[T]
```

Semánticamente, esto indica que la función, clase, o alias de tipo es genérico sobre una variable. Esta información en principalmente usada por verificadores de tipo estático, y en ejecución, los objetos genéricos se comportan de forma muy similar a sus homólogos no genéricos.

Los tipos de parámetros son declarados entre corchetes (`[]`) inmediatamente después del nombre de la función, clase o alias. El tipo de parámetro es accesible en el ámbito del objeto genérico, pero no en otro lugar. Así, después de una declaración `def func[T](): pass`, el nombre `T` no está disponible en el ámbito del módulo. A continuación, se describe con más precisión la semántica de los objetos genéricos. El ámbito de los parámetros de tipo se modela con una función especial (técnicamente, una ámbito de anotación) que envuelve la creación del objeto genérico.

Funciones genéricas, clases y aliases de tipo tienen un atributo :**`__type_params__`** que lista sus parámetros de tipo.

Los tipos de parámetros son de tres tipos:

- **`typing.TypeVar`**, introducido por un nombre plano (p.j., `T`). Sistemáticamente, esto representa un único tipo para un verificador de tipos.
- **`typing.TypeVarTuple`**, introducido por un nombre precedido de un asterisco (p.j, `*Ts`). Semánticamente, representa una tupla de cualquier número de tipos.
- **`typing.ParamSpec`**, introducido por un nombre precedido de dos asterisco (p.j, `*P`). Semánticamente, representa los parámetros de una llamada.

Las declaraciones **`typing.TypeVar`** pueden definir *bounds* y *constraints* con dos puntos (`:`) seguidos de una expresión. Una sola expresión después de los dos puntos indica un límite (por ejemplo, `T: int`). Semánticamente, esto significa que **`typing.TypeVar`** solo puede representar tipos que sean un subtipo de este límite. Una tupla de expresiones entre paréntesis después de los dos puntos indica un conjunto de restricciones (por ejemplo, `T: (str, bytes)`). Cada miembro de la tupla debe ser un tipo (de nuevo, esto no se aplica en tiempo de ejecución). Las variables de tipo restringido solo pueden tomar uno de los tipos de la lista de restricciones.

Para **`typing.TypeVar`**s declarados utilizando la sintaxis de lista de parámetros de tipo, los límites y restricciones no se evalúan cuando se crea el objeto genérico, sino solo cuando se accede explícitamente al valor a través de los atributos `__bound__` y `__constraints__`. Para ello, los límites o restricciones se evalúan en un ámbito de anotación separado.

`typing.TypeVarTuple` s y **`typing.ParamSpec`** s no pueden tener límites ni restricciones.

Los tres sabores de parámetros de tipo también pueden tener un *valor predeterminado*, lo que se usa cuando el parámetro de tipo no se proporciona explícitamente. Esto se añade adjuntando un solo signo igual (=) seguido por una expresión. Como los límites y restricciones del tipo variables, el valor predeterminado no se evalúa cuando el objeto se crea, solamente cuando se accede el atributo `__default__` del parámetro del tipo. Con este fin, el valor predeterminado se evalúa en un ámbito de anotación separado. Si no se especifica un valor predeterminado para un parámetro de tipo, el atributo `__default__` se establece en el objeto de centinela especial **`typing.NoDefault`**.

El siguiente ejemplo indica el conjunto completo de declaraciones de parámetros de tipo permitidas:

```
def overly_generic[
   SimpleTypeVar,
   TypeVarWithDefault = int,
   TypeVarWithBound: int,
   TypeVarWithConstraints: (str, bytes),
   *SimpleTypeVarTuple = (int, float),
   **SimpleParamSpec = (str, bytearray),
](
   a: SimpleTypeVar,
   b: TypeVarWithDefault,
   c: TypeVarWithBound,
   d: Callable[SimpleParamSpec, TypeVarWithConstraints],
   *e: SimpleTypeVarTuple,
): ...
```

8.10.1. Funciones genéricas

Las funciones genéricas son declaradas de la siguiente forma:

```
def func[T](arg: T): ...
```

Esta sintaxis es equivalente a:

```
annotation-def TYPE_PARAMS_OF_func():
    T = typing.TypeVar("T")
    def func(arg: T): ...
    func.__type_params__ = (T,)
    return func
func = TYPE_PARAMS_OF_func()
```

Aquí `annotation-def` indica un ámbito de anotación, que en realidad no está vinculado a ningún nombre en tiempo de ejecución. (Se ha tomado otra libertad en la traducción: la sintaxis no pasa por el acceso a atributos en el módulo **`typing`**, sino que crea una instancia de **`typing.TypeVar`** directamente).

Las anotaciones de las funciones genéricas se evalúan dentro del ámbito de anotación utilizado para declarar los parámetros de tipo, pero no así los valores por defecto y los decoradores de la función.

El siguiente ejemplo ilustra las reglas de alcance para estos casos, así como para otros tipos de parámetros de tipo:

```
@decorator
def func[T: int, *Ts, **P](*args: *Ts, arg: Callable[P,
    ...
```

Excepto para la lazy-evaluation del **`TypeVar`** vinculada, esto es equivalente a:

```
DEFAULT_OF_arg = some_default

annotation-def TYPE_PARAMS_OF_func():

    annotation-def BOUND_OF_T():
        return int
    # In reality, BOUND_OF_T() is evaluated only on dema
    T = typing.TypeVar("T", bound=BOUND_OF_T())
    Ts = typing.TypeVarTuple("Ts")
    P = typing.ParamSpec("P")

    def func(*args: *Ts, arg: Callable[P, T] = DEFAULT_
        ...

    func.__type_params__ = (T, Ts, P)
    return func
func = decorator(TYPE_PARAMS_OF_func())
```

Los nombres en mayúsculas como `DEFAULT_OF_arg` no se vinculan en tiempo de ejecución.

8.10.2. Clases genéricas

Las clases genéricas son declaradas de la siguiente forma:

```
class Bag[T]: ...
```

Esta sintaxis es equivalente a:

```
annotation-def TYPE_PARAMS_OF_Bag():
    T = typing.TypeVar("T")
    class Bag(typing.Generic[T]):
        __type_params__ = (T,)
        ...
    return Bag
Bag = TYPE_PARAMS_OF_Bag()
```

Aquí de nuevo `annotation-def` (no es una palabra clave real) indica un ámbito de anotación, y el nombre `TYPE_PARAMS_OF_Bag` no está vinculado en tiempo de ejecución.

Las clases genéricas heredan implícitamente de **`typing.Generic`**. Las clases base y los argumentos de palabra clave de las clases genéricas se evalúan dentro del ámbito del tipo para los parámetros de tipo, y los decoradores se evalúan fuera de ese ámbito. Esto se ilustra con este ejemplo:

```
@decorator
class Bag(Base[T], arg=T): ...
```

Esto es equivalente a:

```
annotation-def TYPE_PARAMS_OF_Bag():
    T = typing.TypeVar("T")
    class Bag(Base[T], typing.Generic[T], arg=T):
        __type_params__ = (T,)
        ...
    return Bag
Bag = decorator(TYPE_PARAMS_OF_Bag())
```

8.10.3. Alias de tipo genérico

La declaración `type` también se puede usar para crear un alias de tipo genérico:

```
type ListOrSet[T] = list[T] | set[T]
```

Excepto para la evaluación perezosa del valor, esto es equivalente a:

```
annotation-def TYPE_PARAMS_OF_ListOrSet():
    T = typing.TypeVar("T")

    annotation-def VALUE_OF_ListOrSet():
        return list[T] | set[T]
    # In reality, the value is lazily evaluated
    return typing.TypeAliasType("ListOrSet", VALUE_OF_Li
ListOrSet = TYPE_PARAMS_OF_ListOrSet()
```

Aquí, `annotation-def` (no es una palabra clave real) indica un ámbito de anotación. Los nombres en mayúsculas como `TYPE_PARAMS_OF_ListOrSet` no están vinculados en tiempo de ejecución.

8.11. Annotations

Distinto en la versión 3.14: Annotations are now lazily evaluated by default.

Variables and function parameters may carry annotations, created by adding a colon after the name, followed by an expression:

```
x: annotation = 1
def f(param: annotation): ...
```

Functions may also carry a return annotation following an arrow:

```
def f() -> annotation: ...
```

Annotations are conventionally used for type hints, but this is not enforced by the language, and in general annotations may contain arbitrary expressions. The presence of annotations does not change the runtime semantics of the code, except if some mechanism is used that introspects and uses the annotations (such as **`dataclasses`** or **`functools.singledispatch()`**).

By default, annotations are lazily evaluated in an annotation scope. This means that they are not evaluated when the code containing the annotation is evaluated. Instead, the interpreter saves information that can be used to evaluate the annotation later if requested. The **`annotationlib`** module provides tools for evaluating annotations.

If the future statement `from __future__ import annotations` is present, all annotations are instead stored as strings:

```
>>> from __future__ import annotations
>>> def f(param: annotation): ...
>>> f.__annotations__
{'param': 'annotation'}
```

This future statement will be deprecated and removed in a future version of Python, but not before Python 3.13 reaches its end of life (see **PEP 749** [https://peps.python.org/pep-0749/]). When it is used, introspection tools like **`annotationlib.get_annotations()`** and **`typing.get_type_hints()`** are less likely to be able to resolve annotations at runtime.

Notas al pie

La excepción se propaga a la pila de invocación a menos que haya una cláusula `finally` que provoque otra excepción. Esa nueva excepción hace que se pierda la anterior.

En la coincidencia de patrones, una secuencia se define como una de las siguientes:

- una clase que hereda de `collections.abc.Sequence`
- una clase de Python que se ha registrado como `collections.abc.Sequence`
- una clase incorporada que tiene su conjunto de bits (CPython) **`Py_TPFLAGS_SEQUENCE`**
- una clase que hereda de cualquiera de los anteriores

Las siguientes clases de biblioteca estándar son secuencias:

- `array.array`
- `collections.deque`
- `list`
- `memoryview`
- `range`
- `tuple`

Nota

Los valores de sujeto de tipo `str`, `bytes` y `bytearray` no coinciden con los patrones de secuencia.

En la coincidencia de patrones, un mapeo se define como uno de los siguientes:

- una clase que hereda de `collections.abc.Mapping`
- una clase de Python que se ha registrado como `collections.abc.Mapping`
- una clase incorporada que tiene su conjunto de bits (CPython) **`Py_TPFLAGS_MAPPING`**
- una clase que hereda de cualquiera de los anteriores

Las clases de biblioteca estándar `dict` y `types.MappingProxyType` son asignaciones.

Una cadena de caracteres literal que aparece como la primera declaración en el cuerpo de la función se transforma en el atributo `__doc__` y por lo tanto en docstring de la función.

Una cadena de caracteres literal que aparece como la primera declaración en el cuerpo de la clase se transforma en el elemento del espacio de nombre `__doc__` y, por lo tanto, de la clase docstring.

9. Componentes de nivel superior

El intérprete de Python puede obtener su entrada de varias fuentes: de un script que se le pasa como entrada estándar o como argumento del programa, escrito interactivamente, de un archivo fuente de módulo, etc. Este capítulo proporciona la sintaxis utilizada en estos casos.

9.1. Programas completos de Python

Si bien una especificación de lenguaje no necesita prescribir cómo se invoca al intérprete de lenguaje, es útil tener una noción de un programa completo de Python. Un programa completo de Python se ejecuta en un entorno mínimamente inicializado: todos los módulos estándar e integrados están disponibles, pero ninguno ha sido inicializado, excepto `sys` (varios servicios del sistema), `builtins` (funciones integradas, excepciones y `Ninguno`) y `__main__`. Este último se utiliza para proporcionar el espacio de nombres local y global para la ejecución del programa completo.

La sintaxis de un programa completo de Python es la entrada de archivos, que se describe en la siguiente sección.

El intérprete también puede invocarse en modo interactivo; en este caso, no lee ni ejecuta un programa completo, sino que lee y ejecuta una instrucción (posiblemente compuesta) a la vez. El entorno inicial es idéntico al de un programa completo; cada instrucción se ejecuta en el espacio de nombres de `__main__`.

Se puede pasar un programa completo al intérprete en tres formas: con la opción `-c` *string* de línea de comando, como un archivo pasado como primer argumento de línea de comando o como entrada estándar. Si el archivo o la entrada estándar es un dispositivo tty, el intérprete ingresa al modo interactivo; de lo contrario, ejecuta el archivo como un programa completo.

9.2. Entrada de archivo

Todas las entradas leídas de archivos no interactivos tienen la misma forma:

```
file_input: (NEWLINE | statement)* ENDMARKER
```

Esta sintaxis se utiliza en las siguientes situaciones:

- al analizar un programa completo de Python (desde un archivo o desde una cadena);
- al analizar un módulo;
- al analizar una cadena pasada a la función: `exec()`;

9.3. Entrada interactiva

La entrada en modo interactivo se analiza utilizando la siguiente gramática:

```
interactive_input: [stmt_list] NEWLINE | compound_stmt N
```

Tenga en cuenta que una declaración compuesta (de nivel superior) debe ir seguida de una línea en blanco en modo interactivo; esto es necesario para ayudar al analizador sintáctico a detectar el final de la entrada.

9.4. Entrada de expresión

`eval()` se utiliza para la entrada de expresiones. Ignora los espacios en blanco iniciales. El argumento de cadena para `eval()` debe tener la siguiente forma:

```
eval_input: expression_list NEWLINE* ENDMARKER
```

10. Especificación completa de la gramática

Esta es la gramática completa de Python, derivada directamente de la gramática utilizada para generar el analizador CPython (ver Grammar/python.gram [https://github.com/python/cpython/tree/3.14/Grammar/python.gram]). La versión aquí omite detalles relacionados con la generación de código y la recuperación de errores.

The notation used here is the same as in the preceding docs, and is described in the notation section, except for an extra complication:

- ~ («cut»): commit to the current alternative; fail the rule if the alternative fails to parse

  Python mainly uses cuts for optimizations or improved error messages. They often appear to be useless in the listing below.

  Cuts currently don’t appear inside parentheses, brackets, lookaheads and similar. Their behavior in these contexts is deliberately left unspecified.

```
# PEG grammar for Python



# ========================= START OF THE GRAMMAR =======

# General grammatical elements and rules:
#
# * Strings with double quotes (") denote SOFT KEYWORDS
# * Strings with single quotes (') denote KEYWORDS
# * Upper case names (NAME) denote tokens in the Grammar
# * Rule names starting with "invalid_" are used for spe
#     - These rules are NOT used in the first pass of th
#     - Only if the first pass fails to parse, a second
#       rules will be executed.
#     - If the parser fails in the second phase with a g
#       location of the generic failure of the first pas
#       reporting incorrect locations due to the invalic
#     - The order of the alternatives involving invalid
#       (like any rule in PEG).
#
# Grammar Syntax (see PEP 617 for more information):
#
# rule_name: expression
#   Optionally, a type can be included right after the r
#   specifies the return type of the C or Python functic
#   rule:
# rule_name[return_type]: expression
#   If the return type is omitted, then a void * is retu
#   Python.
# e1 e2
#   Match e1, then match e2.
# e1 | e2
#   Match e1 or e2.
#   The first alternative can also appear on the line af
#   formatting purposes. In that case, a | must be used
#   alternative, like so:
#       rule_name[return_type]:
#            | first_alt
#            | second_alt
# ( e )
#   Match e (allows also to use other operators in the g
# [ e ] or e?
#   Optionally match e.
# e*
#   Match zero or more occurrences of e.
# e+
#   Match one or more occurrences of e.
# s.e+
#   Match one or more occurrences of e, separated by s.
#   does not include the separator. This is otherwise i
# &e
#   Succeed if e can be parsed, without consuming any i
# !e
#   Fail if e can be parsed, without consuming any input
# ~
#   Commit to the current alternative, even if it fails
# &&e
#   Eager parse e. The parser will not backtrack and wil
#   fail with SyntaxError if e cannot be parsed.
#

# STARTING RULES
# ==============

file: [statements] ENDMARKER
interactive: statement_newline
eval: expressions NEWLINE* ENDMARKER
func_type: '(' [type_expressions] ')' '->' expression NE

# GENERAL STATEMENTS
# ==================

statements: statement+

statement:
    | compound_stmt
    | simple_stmts

single_compound_stmt:
    | compound_stmt

statement_newline:
    | single_compound_stmt NEWLINE
    | simple_stmts
    | NEWLINE
    | ENDMARKER
```

```
simple_stmts:
    | simple_stmt !';' NEWLINE  # Not needed, there for
    | ';'.simple_stmt+ [';'] NEWLINE

# NOTE: assignment MUST precede expression, else parsing
# will throw a SyntaxError.
simple_stmt:
    | assignment
    | type_alias
    | star_expressions
    | return_stmt
    | import_stmt
    | raise_stmt
    | pass_stmt
    | del_stmt
    | yield_stmt
    | assert_stmt
    | break_stmt
    | continue_stmt
    | global_stmt
    | nonlocal_stmt

compound_stmt:
    | function_def
    | if_stmt
    | class_def
    | with_stmt
    | for_stmt
    | try_stmt
    | while_stmt
    | match_stmt

# SIMPLE STATEMENTS
# =================

# NOTE: annotated_rhs may start with 'yield'; yield_expr
assignment:
    | NAME ':' expression ['=' annotated_rhs ]
    | ('(' single_target ')'
         | single_subscript_attribute_target) ':' expres
    | (star_targets '=' )+ annotated_rhs !'=' [TYPE_COMM
    | single_target augassign ~ annotated_rhs

annotated_rhs: yield_expr | star_expressions

augassign:
    | '+='
    | '-='
    | '*='
    | '@='
    | '/='
    | '%='
    | '&='
    | '|='
    | '^='
    | '<<='
    | '>>='
    | '**='
    | '//='

return_stmt:
    | 'return' [star_expressions]

raise_stmt:
    | 'raise' expression ['from' expression ]
    | 'raise'

pass_stmt:
    | 'pass'

break_stmt:
    | 'break'

continue_stmt:
    | 'continue'
```

```
global_stmt: 'global' ','.NAME+

nonlocal_stmt: 'nonlocal' ','.NAME+

del_stmt:
    | 'del' del_targets &(';' | NEWLINE)

yield_stmt: yield_expr

assert_stmt: 'assert' expression [',' expression ]

import_stmt:
    | import_name
    | import_from

# Import statements
# -----------------

import_name: 'import' dotted_as_names
# note below: the ('.' | '...') is necessary because '..
import_from:
    | 'from' ('.' | '...')* dotted_name 'import' import_
    | 'from' ('.' | '...')+ 'import' import_from_targets
import_from_targets:
    | '(' import_from_as_names [','] ')'
    | import_from_as_names !','
    | '*'
import_from_as_names:
    | ','.import_from_as_name+
import_from_as_name:
    | NAME ['as' NAME ]

dotted_as_names:
    | ','.dotted_as_name+
dotted_as_name:
    | dotted_name ['as' NAME ]

dotted_name:
    | dotted_name '.' NAME
    | NAME

# COMPOUND STATEMENTS
# ===================

# Common elements
# ---------------

block:
    | NEWLINE INDENT statements DEDENT
    | simple_stmts

decorators: ('@' named_expression NEWLINE )+

# Class definitions
# -----------------

class_def:
    | decorators class_def_raw
    | class_def_raw

class_def_raw:
    | 'class' NAME [type_params] ['(' [arguments] ')' ]

# Function definitions
# --------------------

function_def:
    | decorators function_def_raw
    | function_def_raw

function_def_raw:
    | 'def' NAME [type_params] '(' [params] ')' ['->' ex
    | 'async' 'def' NAME [type_params] '(' [params] ')'

# Function parameters
# -------------------
```

```
params:
    | parameters

parameters:
    | slash_no_default param_no_default* param_with_defa
    | slash_with_default param_with_default* [star_etc]
    | param_no_default+ param_with_default* [star_etc]
    | param_with_default+ [star_etc]
    | star_etc

# Some duplication here because we can't write (',' | &'
# which is because we don't support empty alternatives (

slash_no_default:
    | param_no_default+ '/' ','
    | param_no_default+ '/' &')'
slash_with_default:
    | param_no_default* param_with_default+ '/' ','
    | param_no_default* param_with_default+ '/' &')'

star_etc:
    | '*' param_no_default param_maybe_default* [kwds]
    | '*' param_no_default_star_annotation param_maybe_d
    | '*' ',' param_maybe_default+ [kwds]
    | kwds

kwds:
    | '**' param_no_default

# One parameter.  This *includes* a following comma and
#
# There are three styles:
# - No default
# - With default
# - Maybe with default
#
# There are two alternative forms of each, to deal with
# - Ends in a comma followed by an optional type comment
# - No comma, optional type comment, must be followed by
# The latter form is for a final parameter without trail
#

param_no_default:
    | param ',' TYPE_COMMENT?
    | param TYPE_COMMENT? &')'
param_no_default_star_annotation:
    | param_star_annotation ',' TYPE_COMMENT?
    | param_star_annotation TYPE_COMMENT? &')'
param_with_default:
    | param default ',' TYPE_COMMENT?
    | param default TYPE_COMMENT? &')'
param_maybe_default:
    | param default? ',' TYPE_COMMENT?
    | param default? TYPE_COMMENT? &')'
param: NAME annotation?
param_star_annotation: NAME star_annotation
annotation: ':' expression
star_annotation: ':' star_expression
default: '=' expression  | invalid_default

# If statement
# ------------

if_stmt:
    | 'if' named_expression ':' block elif_stmt
    | 'if' named_expression ':' block [else_block]
elif_stmt:
    | 'elif' named_expression ':' block elif_stmt
    | 'elif' named_expression ':' block [else_block]
else_block:
    | 'else' ':' block

# While statement
# ---------------
```

```
while_stmt:
    | 'while' named_expression ':' block [else_block]

# For statement
# -------------

for_stmt:
    | 'for' star_targets 'in' ~ star_expressions ':' [TY
    | 'async' 'for' star_targets 'in' ~ star_expressions

# With statement
# --------------

with_stmt:
    | 'with' '(' ','.with_item+ ','? ')' ':' [TYPE_COMME
    | 'with' ','.with_item+ ':' [TYPE_COMMENT] block
    | 'async' 'with' '(' ','.with_item+ ','? ')' ':' blo
    | 'async' 'with' ','.with_item+ ':' [TYPE_COMMENT] b

with_item:
    | expression 'as' star_target &(',' | ')' | ':')
    | expression

# Try statement
# -------------

try_stmt:
    | 'try' ':' block finally_block
    | 'try' ':' block except_block+ [else_block] [finall
    | 'try' ':' block except_star_block+ [else_block] [f


# Except statement
# ----------------

except_block:
    | 'except' expression ':' block
    | 'except' expression 'as' NAME ':' block
    | 'except' expressions ':' block
    | 'except' ':' block
except_star_block:
    | 'except' '*' expression ':' block
    | 'except' '*' expression 'as' NAME ':' block
    | 'except' '*' expressions ':' block
finally_block:
    | 'finally' ':' block

# Match statement
# ---------------

match_stmt:
    | "match" subject_expr ':' NEWLINE INDENT case_block

subject_expr:
    | star_named_expression ',' star_named_expressions?
    | named_expression

case_block:
    | "case" patterns guard? ':' block

guard: 'if' named_expression

patterns:
    | open_sequence_pattern
    | pattern

pattern:
    | as_pattern
    | or_pattern

as_pattern:
    | or_pattern 'as' pattern_capture_target

or_pattern:
    | '|'.closed_pattern+
closed_pattern:
    | literal_pattern
    | capture_pattern
    | wildcard_pattern
    | value_pattern
    | group_pattern
    | sequence_pattern
    | mapping_pattern
    | class_pattern

# Literal patterns are used for equality and identity co
literal_pattern:
    | signed_number !('+' | '-')
    | complex_number
    | strings
    | 'None'
    | 'True'
    | 'False'

# Literal expressions are used to restrict permitted map
literal_expr:
    | signed_number !('+' | '-')
    | complex_number
    | strings
    | 'None'
    | 'True'
    | 'False'

complex_number:
    | signed_real_number '+' imaginary_number
    | signed_real_number '-' imaginary_number

signed_number:
    | NUMBER
    | '-' NUMBER

signed_real_number:
    | real_number
    | '-' real_number

real_number:
    | NUMBER

imaginary_number:
    | NUMBER

capture_pattern:
    | pattern_capture_target

pattern_capture_target:
    | !"_" NAME !('.' | '(' | '=')

wildcard_pattern:
    | "_"

value_pattern:
    | attr !('.' | '(' | '=')

attr:
    | name_or_attr '.' NAME

name_or_attr:
    | attr
    | NAME

group_pattern:
    | '(' pattern ')'

sequence_pattern:
    | '[' maybe_sequence_pattern? ']'
    | '(' open_sequence_pattern? ')'

open_sequence_pattern:
    | maybe_star_pattern ',' maybe_sequence_pattern?

maybe_sequence_pattern:
    | ','.maybe_star_pattern+ ','?

maybe_star_pattern:
    | star_pattern
    | pattern

star_pattern:
    | '*' pattern_capture_target
    | '*' wildcard_pattern

mapping_pattern:
    | '{' '}'
    | '{' double_star_pattern ','? '}'
    | '{' items_pattern ',' double_star_pattern ','? '}'
    | '{' items_pattern ','? '}'

items_pattern:
    | ','.key_value_pattern+

key_value_pattern:
    | (literal_expr | attr) ':' pattern

double_star_pattern:
    | '**' pattern_capture_target

class_pattern:
    | name_or_attr '(' ')'
    | name_or_attr '(' positional_patterns ','? ')'
    | name_or_attr '(' keyword_patterns ','? ')'
    | name_or_attr '(' positional_patterns ',' keyword_p

positional_patterns:
    | ','.pattern+

keyword_patterns:
    | ','.keyword_pattern+

keyword_pattern:
    | NAME '=' pattern

# Type statement
# ---------------

type_alias:
    | "type" NAME [type_params] '=' expression

# Type parameter declaration
# --------------------------

type_params:
    | '[' type_param_seq ']'

type_param_seq: ','.type_param+ [',']

type_param:
    | NAME [type_param_bound] [type_param_default]
    | '*' NAME [type_param_starred_default]
    | '**' NAME [type_param_default]

type_param_bound: ':' expression
type_param_default: '=' expression
type_param_starred_default: '=' star_expression

# EXPRESSIONS
# -----------

expressions:
    | expression (',' expression )+ [',']
    | expression ','
    | expression

expression:
    | disjunction 'if' disjunction 'else' expression
    | disjunction
    | lambdef
```

```
yield_expr:
    | 'yield' 'from' expression
    | 'yield' [star_expressions]

star_expressions:
    | star_expression (',' star_expression )+ [',']
    | star_expression ','
    | star_expression

star_expression:
    | '*' bitwise_or
    | expression

star_named_expressions: ','.star_named_expression+ [',']

star_named_expression:
    | '*' bitwise_or
    | named_expression

assignment_expression:
    | NAME ':=' ~ expression

named_expression:
    | assignment_expression
    | expression !':='

disjunction:
    | conjunction ('or' conjunction )+
    | conjunction

conjunction:
    | inversion ('and' inversion )+
    | inversion

inversion:
    | 'not' inversion
    | comparison
```

```
# Comparison operators
# --------------------

comparison:
    | bitwise_or compare_op_bitwise_or_pair+
    | bitwise_or

compare_op_bitwise_or_pair:
    | eq_bitwise_or
    | noteq_bitwise_or
    | lte_bitwise_or
    | lt_bitwise_or
    | gte_bitwise_or
    | gt_bitwise_or
    | notin_bitwise_or
    | in_bitwise_or
    | isnot_bitwise_or
    | is_bitwise_or

eq_bitwise_or: '==' bitwise_or
noteq_bitwise_or:
    | ('!=' ) bitwise_or
lte_bitwise_or: '<=' bitwise_or
lt_bitwise_or: '<' bitwise_or
gte_bitwise_or: '>=' bitwise_or
gt_bitwise_or: '>' bitwise_or
notin_bitwise_or: 'not' 'in' bitwise_or
in_bitwise_or: 'in' bitwise_or
isnot_bitwise_or: 'is' 'not' bitwise_or
is_bitwise_or: 'is' bitwise_or

# Bitwise operators
# -----------------

bitwise_or:
    | bitwise_or '|' bitwise_xor
    | bitwise_xor
```

```
bitwise_xor:
    | bitwise_xor '^' bitwise_and
    | bitwise_and

bitwise_and:
    | bitwise_and '&' shift_expr
    | shift_expr

shift_expr:
    | shift_expr '<<' sum
    | shift_expr '>>' sum
    | sum

# Arithmetic operators
# --------------------

sum:
    | sum '+' term
    | sum '-' term
    | term

term:
    | term '*' factor
    | term '/' factor
    | term '//' factor
    | term '%' factor
    | term '@' factor
    | factor

factor:
    | '+' factor
    | '-' factor
    | '~' factor
    | power

power:
    | await_primary '**' factor
    | await_primary
```

```
# Primary elements
# ----------------

# Primary elements are things like "obj.someth

await_primary:
    | 'await' primary
    | primary

primary:
    | primary '.' NAME
    | primary genexp
    | primary '(' [arguments] ')'
    | primary '[' slices ']'
    | atom

slices:
    | slice !','
    | ','.(slice | starred_expression)+ [',']

slice:
    | [expression] ':' [expression] [':' [expression] ]
    | named_expression

atom:
    | NAME
    | 'True'
    | 'False'
    | 'None'
    | strings
    | NUMBER
    | (tuple | group | genexp)
    | (list | listcomp)
    | (dict | set | dictcomp | setcomp)
    | '...'

group:
    | '(' (yield_expr | named_expression) ')'

# Lambda functions
# ----------------

lambdef:
    | 'lambda' [lambda_params] ':' expression

lambda_params:
    | lambda_parameters

# lambda_parameters etc. duplicates parameters but witho
# or type comments, and if there's no comma after a para
# a colon, not a close parenthesis.  (For more, see para
#
lambda_parameters:
    | lambda_slash_no_default lambda_param_no_default* l
    | lambda_slash_with_default lambda_param_with_defaul
    | lambda_param_no_default+ lambda_param_with_default
    | lambda_param_with_default+ [lambda_star_etc]
    | lambda_star_etc

lambda_slash_no_default:
    | lambda_param_no_default+ '/' ','
    | lambda_param_no_default+ '/' &':'

lambda_slash_with_default:
    | lambda_param_no_default* lambda_param_with_default
    | lambda_param_no_default* lambda_param_with_default

lambda_star_etc:
    | '*' lambda_param_no_default lambda_param_maybe_def
    | '*' ',' lambda_param_maybe_default+ [lambda_kwds]
    | lambda_kwds

lambda_kwds:
    | '**' lambda_param_no_default
```

```
lambda_param_no_default:
    | lambda_param ','
    | lambda_param &':'
lambda_param_with_default:
    | lambda_param default ','
    | lambda_param default &':'
lambda_param_maybe_default:
    | lambda_param default? ','
    | lambda_param default? &':'
lambda_param: NAME

# LITERALS
# ========

fstring_middle:
    | fstring_replacement_field
    | FSTRING_MIDDLE
fstring_replacement_field:
    | '{' annotated_rhs '='? [fstring_conversion] [fstri
fstring_conversion:
    | "!" NAME
fstring_full_format_spec:
    | ':' fstring_format_spec*
fstring_format_spec:
    | FSTRING_MIDDLE
    | fstring_replacement_field
fstring:
    | FSTRING_START fstring_middle* FSTRING_END

tstring_format_spec_replacement_field:
    | '{' annotated_rhs '='? [fstring_conversion] [tstri
tstring_format_spec:
    | TSTRING_MIDDLE
    | tstring_format_spec_replacement_field
tstring_full_format_spec:
    | ':' tstring_format_spec*
tstring_replacement_field:
    | '{' annotated_rhs '='? [fstring_conversion] [tstri
```

```
tstring_middle:
    | tstring_replacement_field
    | TSTRING_MIDDLE
tstring:
    | TSTRING_START tstring_middle* TSTRING_END

string: STRING
strings:
    | (fstring|string)+
    | tstring+

list:
    | '[' [star_named_expressions] ']'

tuple:
    | '(' [star_named_expression ',' [star_named_express

set: '{' star_named_expressions '}'

# Dicts
# -----

dict:
    | '{' [double_starred_kvpairs] '}'

double_starred_kvpairs: ','.double_starred_kvpair+ [',']

double_starred_kvpair:
    | '**' bitwise_or
    | kvpair

kvpair: expression ':' expression

# Comprehensions & Generators
# ---------------------------

for_if_clauses:
    | for_if_clause+
for_if_clause:
    | 'async' 'for' star_targets 'in' ~ disjunction ('if
    | 'for' star_targets 'in' ~ disjunction ('if' disjun

listcomp:
    | '[' named_expression for_if_clauses ']'

setcomp:
    | '{' named_expression for_if_clauses '}'

genexp:
    | '(' ( assignment_expression | expression !':=') fo

dictcomp:
    | '{' kvpair for_if_clauses '}'

# FUNCTION CALL ARGUMENTS
# =======================

arguments:
    | args [','] &')'

args:
    | ','.(starred_expression | ( assignment_expression
    | kwargs

kwargs:
    | ','.kwarg_or_starred+ ',' ','.kwarg_or_double_star
    | ','.kwarg_or_starred+
    | ','.kwarg_or_double_starred+

starred_expression:
    | '*' expression

kwarg_or_starred:
    | NAME '=' expression
    | starred_expression
```

```
kwarg_or_double_starred:
    | NAME '=' expression
    | '**' expression

# ASSIGNMENT TARGETS
# ==================

# Generic targets
# ---------------

# NOTE: star_targets may contain *bitwise_or, targets ma
star_targets:
    | star_target !','
    | star_target (',' star_target )* [',']

star_targets_list_seq: ','.star_target+ [',']

star_targets_tuple_seq:
    | star_target (',' star_target )+ [',']
    | star_target ','

star_target:
    | '*' (!'*' star_target)
    | target_with_star_atom

target_with_star_atom:
    | t_primary '.' NAME !t_lookahead
    | t_primary '[' slices ']' !t_lookahead
    | star_atom

star_atom:
    | NAME
    | '(' target_with_star_atom ')'
    | '(' [star_targets_tuple_seq] ')'
    | '[' [star_targets_list_seq] ']'

single_target:
    | single_subscript_attribute_target
    | NAME
    | '(' single_target ')'

single_subscript_attribute_target:
    | t_primary '.' NAME !t_lookahead
    | t_primary '[' slices ']' !t_lookahead

t_primary:
    | t_primary '.' NAME &t_lookahead
    | t_primary '[' slices ']' &t_lookahead
    | t_primary genexp &t_lookahead
    | t_primary '(' [arguments] ')' &t_lookahead
    | atom &t_lookahead

t_lookahead: '(' | '[' | '.'

# Targets for del statements
# --------------------------

del_targets: ','.del_target+ [',']

del_target:
    | t_primary '.' NAME !t_lookahead
    | t_primary '[' slices ']' !t_lookahead
    | del_t_atom

del_t_atom:
    | NAME
    | '(' del_target ')'
    | '(' [del_targets] ')'
    | '[' [del_targets] ']'

# TYPING ELEMENTS
# ---------------

# type_expressions allow */** but ignore them
type_expressions:
    | ','.expression+ ',' '*' expression ',' '**' expres
    | ','.expression+ ',' '*' expression
    | ','.expression+ ',' '**' expression
    | '*' expression ',' '**' expression
    | '*' expression
    | '**' expression
    | ','.expression+

func_type_comment:
    | NEWLINE TYPE_COMMENT &(NEWLINE INDENT)   # Must be
    | TYPE_COMMENT

# ========================= END OF THE GRAMMAR =========



# ========================= START OF INVALID RULES =====
```

La biblioteca estándar de Python

Aunque Referencia del Lenguaje Python describe la sintaxis y semántica precisa del lenguaje Python, este manual de referencia de la biblioteca describe la biblioteca estándar que se distribuye con Python. También describe algunos componentes opcionales que son usualmente incluidos en las distribuciones de Python.

La biblioteca estándar de Python es muy amplia, y ofrece una gran cantidad de producciones como puede verse en la larga lista de contenidos. La biblioteca contiene módulos incorporados (escritos en C) que brindan acceso a las funcionalidades del sistema como entrada y salida de archivos que serían de otra forma inaccesibles para los programadores en Python, así como módulos escritos en Python que proveen soluciones estandarizadas para los diversos problemas que pueden ocurrir en el día a día en la programación. Algunos de éstos módulos están diseñados explícitamente para alentar y reforzar la portabilidad de los programas en Python abstrayendo especificidades de las plataformas para lograr APIs neutrales a la plataforma.

Los instaladores de Python para la plataforma Windows frecuentemente incluyen la biblioteca estándar completa y suelen también incluir muchos componentes adicionales. Para los sistemas operativos tipo Unix Python suele ser provisto como una colección de paquetes, así que puede requerirse usar las herramientas de empaquetado disponibles en los sistemas operativos para obtener algunos o todos los componentes opcionales.

Además de la biblioteca estándar, existe un colección creciente de varios miles de componentes (abarcando módulos o programas individuales, paquetes o *frameworks* completos de desarrollo de aplicaciones), disponibles en el Python Package Index [https://pypi.org].

- Introducción

    - Notas sobre la disponibilidad

- Funciones incorporadas
- Constantes incorporadas

    - Constantes agregadas por el módulo `site`

- Tipos integrados

    - Evaluar como valor verdadero/falso
    - Operaciones booleanas — `and`, `or`, `not`
    - Comparaciones
    - Tipos numéricos — `int`, `float`, `complex`
    - Tipo Booleano — `bool`
    - Tipos de iteradores
    - Tipos secuencia — `list`, `tuple`, `range`
    - Text and Binary Sequence Type Methods Summary
    - Cadenas de caracteres — `str`
    - Tipos de secuencias binarias — `bytes`, `bytearray` y `memoryview`
    - Conjuntos — `set`, `frozenset`
    - Tipos mapa — `dict`
    - Tipos gestores de contexto
    - Tipos de anotaciones de type — alias genérico, Union
    - Otros tipos predefinidos
    - Atributos especiales
    - Limitación de longitud de conversión de cadena de tipo entero

- Excepciones incorporadas

    - Contexto de una excepción
    - Heredando de excepciones incorporadas
    - Clases base
    - Excepciones específicas
    - Advertencias
    - Grupos de excepciones
    - Jerarquía de excepción

- Thread Safety Guarantees
    - Thread safety levels
    - Thread safety for list objects
    - Thread safety for dict objects
    - Thread safety for set objects
    - Thread safety for bytearray objects
    - Thread safety for memoryview objects
- Servicios de procesamiento de texto
    - **`string`** — Common string operations
    - **`string.templatelib`** — Support for template string literals
    - **`re`** — Regular expression operations
    - **`difflib`** — Helpers for computing deltas
    - **`textwrap`** — Text wrapping and filling
    - **`unicodedata`** — Unicode Database
    - **`stringprep`** — Internet String Preparation
    - **`readline`** — GNU readline interface
    - **`rlcompleter`** — Completion function for GNU readline
- Servicios de datos binarios
    - **`struct`** — Interpret bytes as packed binary data
    - **`codecs`** — Codec registry and base classes
- Tipos de datos
    - **`datetime`** — Basic date and time types
    - **`zoneinfo`** — IANA time zone support
    - **`calendar`** — General calendar-related functions
    - **`collections`** — Container datatypes
    - **`collections.abc`** — Abstract Base Classes for Containers
    - **`heapq`** — Heap queue algorithm
    - **`bisect`** — Array bisection algorithm
    - **`array`** — Efficient arrays of numeric values
    - **`weakref`** — Weak references
    - **`types`** — Dynamic type creation and names for built-in

types
  - **`copy`** — Shallow and deep copy operations
  - **`pprint`** — Data pretty printer
  - **`reprlib`** — Alternate **`repr()`** implementation
  - **`enum`** — Support for enumerations
  - **`graphlib`** — Functionality to operate with graph-like structures

- Módulos numéricos y matemáticos

  - **`numbers`** — Numeric abstract base classes
  - **`math`** — Mathematical functions
  - **`cmath`** — Mathematical functions for complex numbers
  - **`decimal`** — Decimal fixed-point and floating-point arithmetic
  - **`fractions`** — Rational numbers
  - **`random`** — Generate pseudo-random numbers
  - **`statistics`** — Mathematical statistics functions

- Módulos de programación funcional

  - **`itertools`** — Functions creating iterators for efficient looping
  - **`functools`** — Higher-order functions and operations on callable objects
  - **`operator`** — Standard operators as functions

- Acceso a archivos y directorios

  - **`pathlib`** — Rutas de sistemas orientada a objetos
  - **`os.path`** — Common pathname manipulations
  - **`stat`** — Interpreting **`stat()`** results
  - **`filecmp`** — File and Directory Comparisons
  - **`tempfile`** — Generate temporary files and directories
  - **`glob`** — Unix style pathname pattern expansion
  - **`fnmatch`** — Unix filename pattern matching
  - **`linecache`** — Random access to text lines
  - **`shutil`** — High-level file operations

- Persistencia de datos

  - `pickle` — Python object serialization
  - `copyreg` — Register `pickle` support functions
  - `shelve` — Python object persistence
  - `marshal` — Internal Python object serialization
  - `dbm` — Interfaces to Unix «databases»
  - `sqlite3` — DB-API 2.0 interfaz para bases de datos SQLite

- Compresión de datos y archivado

  - The `compression` package
  - `compression.zstd` — Compression compatible with the Zstandard format
  - `zlib` — Compression compatible with **gzip**
  - `gzip` — Support for **gzip** files
  - `bz2` — Support for **bzip2** compression
  - `lzma` — Compression using the LZMA algorithm
  - `zipfile` — Work with ZIP archives
  - `tarfile` — Read and write tar archive files

- Formatos de archivo

  - `csv` — CSV File Reading and Writing
  - `configparser` — Configuration file parser
  - `tomllib` — Parse TOML files
  - `netrc` — netrc file processing
  - `plistlib` — Generate and parse Apple `.plist` files

- Servicios criptográficos

  - `hashlib` — Secure hashes and message digests
  - `hmac` — Keyed-Hashing for Message Authentication
  - `secrets` — Generate secure random numbers for managing secrets

- Servicios genéricos del sistema operativo

  - `os` — Interfaces misceláneas del sistema operativo
  - `io` — Core tools for working with streams
  - `time` — Time access and conversions
  - `logging` — Logging facility for Python

  - `logging.config` — Logging configuration
  - `logging.handlers` — Logging handlers
  - `platform` — Access to underlying platform’s identifying data
  - `errno` — Standard errno system symbols
  - `ctypes` — A foreign function library for Python
- Command-line interface libraries
  - `argparse` — Parser for command-line options, arguments and subcommands
  - `optparse` — Parser for command line options
  - `getpass` — Portable password input
  - `fileinput` — Iterate over lines from multiple input streams
  - `curses` — Terminal handling for character-cell displays
  - `curses.textpad` — Text input widget for curses programs
  - `curses.ascii` — Utilities for ASCII characters
  - `curses.panel` — A panel stack extension for curses
  - `cmd` — Support for line-oriented command interpreters
- Ejecución concurrente
  - `threading` — Thread-based parallelism
  - `multiprocessing` — Process-based parallelism
  - `multiprocessing.shared_memory` — Shared memory for direct access across processes
  - El paquete `concurrent`
  - `concurrent.futures` — Launching parallel tasks
  - `concurrent.interpreters` — Multiple interpreters in the same process
  - `subprocess` — Subprocess management
  - `sched` — Event scheduler
  - `queue` — A synchronized queue class
  - `contextvars` — Context Variables
  - `_thread` — Low-level threading API
- Comunicación en redes y entre procesos

  - **`asyncio`** — Asynchronous I/O
  - **`socket`** — Low-level networking interface
  - **`ssl`** — TLS/SSL wrapper for socket objects
  - **`select`** — Waiting for I/O completion
  - **`selectors`** — High-level I/O multiplexing
  - **`signal`** — Set handlers for asynchronous events
  - **`mmap`** — Memory-mapped file support

- Manejo de datos de internet

  - **`email`** — An email and MIME handling package
  - **`json`** — JSON encoder and decoder
  - **`mailbox`** — Manipulate mailboxes in various formats
  - **`mimetypes`** — Map filenames to MIME types
  - **`base64`** — Base16, Base32, Base64, Base85 Data Encodings
  - **`binascii`** — Convert between binary and ASCII
  - **`quopri`** — Encode and decode MIME quoted-printable data

- Herramientas Para Procesar Formatos de Marcado Estructurado

  - **`html`** — HyperText Markup Language support
  - **`html.parser`** — Simple HTML and XHTML parser
  - **`html.entities`** — Definitions of HTML general entities
  - Módulos de procesamiento XML
  - **`xml.etree.ElementTree`** — The ElementTree XML API
  - **`xml.dom`** — The Document Object Model API
  - **`xml.dom.minidom`** — Minimal DOM implementation
  - **`xml.dom.pulldom`** — Support for building partial DOM trees
  - **`xml.sax`** — Support for SAX2 parsers
  - **`xml.sax.handler`** — Base classes for SAX handlers
  - **`xml.sax.saxutils`** — SAX Utilities
  - **`xml.sax.xmlreader`** — Interface for XML parsers
  - **`xml.parsers.expat`** — Fast XML parsing using Expat

- Protocolos y soporte de Internet

    - **`webbrowser`** — Convenient web-browser controller
    - **`wsgiref`** — WSGI Utilities and Reference Implementation
    - **`urllib`** — URL handling modules
    - **`urllib.request`** — Extensible library for opening URLs
    - **`urllib.response`** — Response classes used by urllib
    - **`urllib.parse`** — Parse URLs into components
    - **`urllib.error`** — Exception classes raised by urllib.request
    - **`urllib.robotparser`** — Parser for robots.txt
    - **`http`** — HTTP modules
    - **`http.client`** — HTTP protocol client
    - **`ftplib`** — FTP protocol client
    - **`poplib`** — POP3 protocol client
    - **`imaplib`** — IMAP4 protocol client
    - **`smtplib`** — SMTP protocol client
    - **`uuid`** — UUID objects according to **RFC 9562**
    - **`socketserver`** — A framework for network servers
    - **`http.server`** — HTTP servers
    - **`http.cookies`** — HTTP state management
    - **`http.cookiejar`** — Cookie handling for HTTP clients
    - **`xmlrpc`** — Módulos XMLRPC para cliente y servidor
    - **`xmlrpc.client`** — XML-RPC client access
    - **`xmlrpc.server`** — Basic XML-RPC servers
    - **`ipaddress`** — IPv4/IPv6 manipulation library
- Servicios Multimedia
    - **`wave`** — Read and write WAV files
    - **`colorsys`** — Conversions between color systems
- Internacionalización
    - **`gettext`** — Multilingual internationalization services
    - **`locale`** — Internationalization services
- Graphical user interfaces with Tk
    - **`tkinter`** — Python interface to Tcl/Tk
    - **`tkinter.colorchooser`** — Color choosing dialog

    - **`tkinter.font`** — Tkinter font wrapper
    - Diálogos tkinter
    - **`tkinter.messagebox`** — Tkinter message prompts
    - **`tkinter.scrolledtext`** — Scrolled Text Widget
    - **`tkinter.dnd`** — Drag and drop support
    - **`tkinter.ttk`** — Tk themed widgets
    - IDLE — Python editor and shell
    - **`turtle`** — Turtle graphics

- Herramientas de desarrollo

    - **`typing`** — Support for type hints
    - **`pydoc`** — Documentation generator and online help system
    - Modo de desarrollo de Python
    - **`doctest`** — Test interactive Python examples
    - **`unittest`** — Unit testing framework
    - **`unittest.mock`** — mock object library
    - **`unittest.mock`** — getting started
    - **`test`** — Regression tests package for Python
    - **`test.support`** — Utilities for the Python test suite
    - **`test.support.socket_helper`** — Utilities for socket tests
    - **`test.support.script_helper`** — Utilities for the Python execution tests
    - **`test.support.bytecode_helper`** — Support tools for testing correct bytecode generation
    - **`test.support.threading_helper`** — Utilities for threading tests
    - **`test.support.os_helper`** — Utilities for os tests
    - **`test.support.import_helper`** — Utilities for import tests
    - **`test.support.warnings_helper`** — Utilities for warnings tests

- Depuración y perfilado

    - Tabla de auditoría de eventos
    - **`bdb`** — Debugger framework
    - **`faulthandler`** — Dump the Python traceback
    - **`pdb`** — The Python Debugger

  - Los perfiladores de Python
  - **`timeit`** — Measure execution time of small code snippets
  - **`trace`** — Trace or track Python statement execution
  - **`tracemalloc`** — Trace memory allocations

- Empaquetado y distribución de software

  - **`ensurepip`** — Bootstrapping the **`pip`** installer
  - **`venv`** — Creation of virtual environments
  - **`zipapp`** — Manage executable Python zip archives

- Servicios en tiempo de ejecución de Python

  - **`sys`** — System-specific parameters and functions
  - **`sys.monitoring`** — Execution event monitoring
  - **`sysconfig`** — Provide access to Python's configuration information
  - **`builtins`** — Built-in objects
  - **`__main__`** — Top-level code environment
  - **`warnings`** — Warning control
  - **`dataclasses`** — Data Classes
  - **`contextlib`** — Utilidades para declaraciones de contexto **`with`**
  - **`abc`** — Abstract Base Classes
  - **`atexit`** — Exit handlers
  - **`traceback`** — Print or retrieve a stack traceback
  - **`__future__`** — Future statement definitions
  - **`gc`** — Garbage Collector interface
  - **`inspect`** — Inspect live objects
  - **`annotationlib`** — Functionality for introspecting annotations
  - **`site`** — Site-specific configuration hook

- Intérpretes de Python personalizados

  - **`code`** — Interpreter base classes
  - **`codeop`** — Compile Python code

- Importando módulos

  - **`zipimport`** — Import modules from Zip archives
  - **`pkgutil`** — Package extension utility
  - **`modulefinder`** — Find modules used by a script
  - **`runpy`** — Localización y ejecución de módulos Python
  - **`importlib`** — La implementación de **`import`**
  - **`importlib.resources`** – Package resource reading, opening and access
  - **`importlib.resources.abc`** – Abstract base classes for resources
  - **`importlib.metadata`** – Acceso a los metadatos de los paquetes
  - La inicialización de la ruta de búsqueda de módulo de **`sys.path`**

- Servicios del lenguaje Python

  - **`ast`** — Abstract syntax trees
  - **`symtable`** — Access to the compiler's symbol tables
  - **`token`** — Constants used with Python parse trees
  - **`keyword`** — Testing for Python keywords
  - **`tokenize`** — Tokenizer for Python source
  - **`tabnanny`** — Detection of ambiguous indentation
  - **`pyclbr`** — Python module browser support
  - **`py_compile`** — Compile Python source files
  - **`compileall`** — Byte-compile Python libraries
  - **`dis`** — Desensamblador para bytecode de Python
  - **`pickletools`** — Tools for pickle developers

- Servicios Específicos para MS Windows

  - **`msvcrt`** — Useful routines from the MS VC+ + runtime
  - **`winreg`** — Windows registry access
  - **`winsound`** — Sound-playing interface for Windows

- Unix-specific services

  - **`shlex`** — Simple lexical analysis
  - **`posix`** — The most common POSIX system calls
  - **`pwd`** — The password database
  - **`grp`** — The group database

  - `termios` — POSIX style tty control
  - `tty` — Terminal control functions
  - `pty` — Pseudo-terminal utilities
  - `fcntl` — The `fcntl` and `ioctl` system calls
  - `resource` — Resource usage information
  - `syslog` — Unix syslog library routines
- Modules command-line interface (CLI)
- Módulos reemplazados
  - `getopt` — C-style parser for command line options
- Removed Modules
- Consideraciones de seguridad

Introducción

La «biblioteca Python» contiene varios tipos de componentes diferentes.

Contiene tipos de datos que normalmente se considerarían parte del «núcleo» de un lenguaje, como números y listas. Para estos tipos, el núcleo del lenguaje Python define la forma de los literales y coloca algunas restricciones a su semántica, pero no define completamente la semántica. (Por otro lado, el núcleo del lenguaje sí define propiedades sintácticas como la ortografía y las prioridades de los operadores.)

La biblioteca también contiene funciones y excepciones incorporadas — objetos que pueden ser utilizados por todo código Python sin la necesidad de una declaración `import`. Algunas de ellos están definidas por el núcleo del lenguaje, pero muchos no son esenciales para la semántica del núcleo y sólo se describen aquí.

La mayor parte de la biblioteca, sin embargo, consiste en una colección de módulos. Hay muchas maneras de diseccionar esta colección. Algunos módulos están escritos en C y fueron incorporados en el intérprete de Python; otros están escritos en Python y se importan en código fuente. Algunos módulos proporcionan interfaces muy específicas de Python, como la impresión de un *stack trace*; otros proporcionan interfaces que son específicas para determinados sistemas operativos, como el acceso a hardware específico; otros proveen interfaces específicas para un dominio de aplicación concreto, como la World Wide Web. Algunos módulos están disponibles en todas las versiones y plataformas de Python; otros sólo están disponibles cuando el sistema subyacente los soporta o los requiere; otros solo están disponibles cuando se ha elegido una opción de configuración particular en el momento en que compiló e instaló Python.

Este manual está organizado «desde adentro hacia afuera:» primero describe las funciones integradas, los tipos de datos y las excepciones, y finalmente describe los módulos, agrupados en capítulos de módulos relacionados.

Esto significa que si comienza a leer este manual desde el principio, y salta al siguiente capítulo cuando se aburra, obtendrá una visión general razonable de los módulos y áreas de aplicación disponibles que son soportados por la biblioteca de Python. Por supuesto, no *tienes* que leerlo necesariamente como una novela — sino que también puedes navegar por la tabla de contenidos (al principio del manual), o buscar una función, módulo o término específico en el glosario (en la parte final del manual). Y por último, si disfruta aprender sobre diferentes temas de manera aleatoria, puede escoger un número de página al azar (ver módulo **`random`**) y leer una o dos secciones. Independientemente del orden en que lea las secciones de este manual, es útil comenzar con el capítulo Funciones incorporadas, ya que el resto del manual asume la familiaridad con este material.

¡Que comience el espectáculo!

Notas sobre la disponibilidad

- Una nota de «Disponibilidad: Unix» significa que esta función se encuentra comúnmente en los sistemas Unix. Pero no hace ninguna afirmación sobre su existencia en un sistema operativo específico.
- If not separately noted, all functions that claim «Availability: Unix» are supported on macOS, iOS and Android, all of which build on a Unix core.
- Si una nota de disponibilidad contiene tanto una versión mínima del Kernel como una versión mínima de libc, entonces ambas condiciones deben cumplirse. Por ejemplo, una característica con la nota *Availability: Linux >= 3.17 with glibc >= 2.27* requiere tanto Linux 3.17 o posterior como glibc 2.27 o posterior.

Plataformas WebAssembly

Las plataformas WebAssembly [https://webassembly.org/] `wasm32-emscripten` (Emscripten [https://emscripten.org/]) y `wasm32-wasi`

(WASI [https://wasi.dev/]) proporcionan un subconjunto de APIs POSIX. Los tiempos de ejecución de WebAssembly y los navegadores están aislados y tienen acceso limitado al host y a los recursos externos. Cualquier módulo de la biblioteca estándar de Python que utilice procesos, hilos, redes, señales u otras formas de comunicación entre procesos (IPC), no está disponible o puede no funcionar como en otros sistemas tipo Unix. La E/S de archivos, el sistema de archivos y las funciones relacionadas con permisos Unix también están restringidas. Emscripten no permite el bloqueo de E/S. Otras operaciones bloqueantes como **`sleep()`** bloquean el bucle de eventos del navegador.

Las propiedades y el comportamiento de Python en plataformas WebAssembly dependen de la versión del SDK Emscripten [https://emscripten.org/] o del SDK WASI [https://wasi.dev/], de los tiempos de ejecución WASM (navegador, NodeJS, wasmtime [https://wasmtime.dev/]) y de los indicadores de tiempo de compilación de Python. WebAssembly, Emscripten, y WASI son estándares en evolución; algunas características como las redes pueden ser soportadas en el futuro.

Para Python en el navegador, los usuarios deberían considerar Pyodide [https://pyodide.org/] o PyScript [https://pyscript.net/]. PyScript está construido sobre Pyodide, que a su vez está construido sobre CPython y Emscripten. Pyodide proporciona acceso a las APIs JavaScript y DOM de los navegadores, así como capacidades de red limitadas con las APIs `XMLHttpRequest` y `Fetch` de JavaScript.

- Las APIs relacionadas con procesos no están disponibles o siempre fallan con un error. Esto incluye APIs que generan nuevos procesos (**`fork()`**, **`execve()`**), esperan procesos (**`waitpid()`**), envían señales (**`kill()`**), o interactúan con procesos. El **`subprocess`** se puede importar pero no funciona.
- El módulo **`socket`** está disponible, pero es limitado y se comporta de forma diferente a otras plataformas. En Emscripten, los sockets son siempre no bloqueantes y requieren código JavaScript adicional y auxiliares en el servidor para proxy TCP a través de WebSockets; ver Emscripten Networking [https://emscripten.org/docs/porting/

networking.html] para más información. Vista previa de instantánea WASI 1 permite sólo sockets desde un descriptor de fichero existente.
- Algunas funciones son stubs que no hacen nada y siempre devuelven valores codificados.
- Las funciones relacionadas con descriptores de archivos, permisos de archivos, propiedad de archivos y enlaces son limitadas y no admiten algunas operaciones. Por ejemplo, WASI no permite enlaces simbólicos con nombres de archivo absolutos.

Mobile platforms

Android and iOS are, in most respects, POSIX operating systems. File I/O, socket handling, and threading all behave as they would on any POSIX operating system. However, there are several major differences:

- Mobile platforms can only use Python in «embedded» mode. There is no Python REPL, and no ability to use separate executables such as **python** or **pip**. To add Python code to your mobile app, you must use the Python embedding API. For more details, see Using Python on Android and Using Python on iOS.

- Subprocesses:

    - On Android, creating subprocesses is possible but officially unsupported [https://issuetracker.google.com/issues/128554619#comment4]. In particular, Android does not support any part of the System V IPC API, so **`multiprocessing`** is not available.
    - An iOS app cannot use any form of subprocessing, multiprocessing, or inter-process communication. If an iOS app attempts to create a subprocess, the process creating the subprocess will either lock up, or crash. An iOS app has no visibility of other applications that are running, nor any ability to communicate with other running applications, outside of the iOS-specific APIs that exist for this purpose.

- Mobile apps have limited access to modify system resources (such as the system clock). These resources will often be *readable*, but attempts to modify those resources will usually fail.

- Console input and output:

    - On Android, the native `stdout` and `stderr` are not connected to anything, so Python installs its own streams which redirect messages to the system log. These can be seen under the tags `python.stdout` and `python.stderr` respectively.

    - iOS apps have a limited concept of console output. `stdout` and `stderr` *exist*, and content written to `stdout` and `stderr` will be visible in logs when running in Xcode, but this content *won't* be recorded in the system log. If a user who has installed your app provides their app logs as a diagnostic aid, they will not include any detail written to `stdout` or `stderr`.

    - Mobile apps have no usable `stdin` at all. While apps can display an on-screen keyboard, this is a software feature, not something that is attached to `stdin`.

      As a result, Python modules that involve console manipulation (such as **`curses`** and **`readline`**) are not available on mobile platforms.

[1]

Funciones incorporadas

El intérprete de Python tiene una serie de funciones y tipos incluidos en él que están siempre disponibles. Están listados aquí en orden alfabético.

Funciones incorporadas

A
- `abs()`
- `aiter()`
- `all()`
- `anext()`
- `any()`
- `ascii()`

B
- `bin()`
- `bool()`
- `breakpoint()`
- `bytearray()`
- `bytes()`

C
- `callable()`
- `chr()`
- `classmethod()`
- `compile()`
- `complex()`

D
- `delattr()`
- `dict()`
- `dir()`
- `divmod()`

E
- `enumerate()`
- `eval()`
- `exec()`

F
- `filter()`
- `float()`
- `format()`
- `frozenset()`

G
- `getattr()`
- `globals()`

H
- `hasattr()`
- `hash()`
- `help()`
- `hex()`

I
- `id()`
- `input()`
- `int()`
- `isinstance()`
- `issubclass()`
- `iter()`

L
- `len()`
- `list()`
- `locals()`

M
- `map()`
- `max()`
- `memoryview()`
- `min()`

N
- `next()`

O
- `object()`
- `oct()`
- `open()`
- `ord()`

P
- `pow()`
- `print()`
- `property()`

R
- `range()`
- `repr()`
- `reversed()`
- `round()`

S
- `set()`
- `setattr()`
- `slice()`
- `sorted()`
- `staticmethod()`
- `str()`
- `sum()`
- `super()`

T
- `tuple()`
- `type()`

V
- `vars()`

Z
- `zip()`

_
- `__import__()`

abs(*number*, */*)

Return the absolute value of a number. The argument may be an integer, a floating-point number, or an object implementing **`__abs__()`**. If the argument is a complex number, its magnitude is returned.

aiter(*async_iterable*, */*)

Retorna un asynchronous iterator para un asynchronous iterable. Equivalente a llamar `x.__aiter__()`.

Nota: A diferencia de **`iter()`**, **`aiter()`** no tiene una variante con 2 argumentos.

Added in version 3.10.

all(*iterable*, */*)

Retorna `True` si todos los elementos del *iterable* son verdaderos (o si el iterable está vacío). Equivalente a:

```
def all(iterable):
    for element in iterable:
        if not element:
            return False
    return True
```

awaitable anext(*async_iterator*, */*)
awaitable anext(*async_iterator*, *default*, */*)

Cuando está esperado, retorna el siguiente elemento del asynchronous iterator otorgado, o *default* (por defecto) si se da y el iterador está agotado.

Esta es la variante asincrónica del **`next()`** incorporado, y se comporta de manera similar.

Esto llama al método **`__anext__()`** de *async_iterator*, retornando un awaitable. A la espera de esto se retorna el siguiente valor del iterador. Si se otorga *default* (por defecto), se retorna si el iterador está agotado, de lo contrario se lanza

`StopAsyncIteration`.

Added in version 3.10.

any(*iterable*, */*)

Retorna `True` si un elemento cualquiera del *iterable* es verdadero. Si el iterable está vacío, retorna `False`. Equivalente a:

```
def any(iterable):
    for element in iterable:
        if element:
            return True
    return False
```

ascii(*object*, */*)

Como **`repr()`**, retorna una cadena que contiene una representación imprimible de un objeto, pero que escapa los caracteres no ASCII que **`repr()`** retorna usando `\x`, `\u` or `\U`. Esto genera una cadena similar a la retornada por **`repr()`** en Python 2.

bin(*integer*, */*)

Convert an integer number to a binary string prefixed with «0b». The result is a valid Python expression. If *integer* is not a Python **`int`** object, it has to define an **`__index__()`** method that returns an integer. Some examples:

```
>>> bin(3)
'0b11'
>>> bin(-10)
'-0b1010'
```

Según se desee o no el prefijo «0b», se puede usar uno u otro de las siguientes maneras.

```
>>> format(14, '#b'), format(14, 'b')
('0b1110', '1110')
>>> f'{14:#b}', f'{14:b}'
```

```
('0b1110', '1110')
```

See also **`enum.bin()`** to represent negative values as twos-complement.

Véase también **`format()`** para más información.

class bool(*object=False*, */*)

Return a Boolean value, i.e. one of `True` or `False`. The argument is converted using the standard truth testing procedure. If the argument is false or omitted, this returns `False`; otherwise, it returns `True`. The **`bool`** class is a subclass of **`int`** (see Tipos numéricos — int, float, complex). It cannot be subclassed further. Its only instances are `False` and `True` (see Tipo Booleano — bool).

Distinto en la versión 3.7: The parameter is now positional-only.

breakpoint(**args*, ***kws*)

This function drops you into the debugger at the call site. Specifically, it calls **`sys.breakpointhook()`**, passing `args` and `kws` straight through. By default, `sys.breakpointhook()` calls **`pdb.set_trace()`** expecting no arguments. In this case, it is purely a convenience function so you don't have to explicitly import **`pdb`** or type as much code to enter the debugger. However, **`sys.breakpointhook()`** can be set to some other function and **`breakpoint()`** will automatically call that, allowing you to drop into the debugger of choice. If **`sys.breakpointhook()`** is not accessible, this function will raise **`RuntimeError`**.

Por defecto, el comportamiento de **`breakpoint()`** se puede cambiar con la variable de entorno **`PYTHONBREAKPOINT`**. Véase **`sys.breakpointhook()`** para obtener detalles de uso.

Tenga en cuenta que esto no está garantizado si se ha reemplazado **`sys.breakpointhook()`**.

Lanza un evento de auditoría `builtins.breakpoint` con `breakpointhook` como argumento.

Added in version 3.7.

class bytearray(*source=b''*)
class bytearray(*source, encoding, errors='strict'*)

Retorna un nuevo array de bytes. La clase **`bytearray`** es una secuencia mutable de enteros en el rango 0 < = x < 256. Tiene la mayoría de los métodos comunes en las secuencias mutables, descritos en Tipos de secuencia mutables, así como la mayoría de los métodos que la clase **`bytes`** tiene, véase Operaciones de bytes y bytearray.

El parámetro opcional *source* puede ser empleado para inicializar el vector (*array*) de varias maneras distintas:

- Si es una *string*, debes proporcionar también el parámetro *encoding* (y opcionalmente, *errors*; entonces **`bytearray()`** convierte la cadena a bytes empleando **`str.encode()`**.
- Si es un *integer*, el array tendrá ese tamaño y será inicializado con bytes nulos.
- Si es un objeto conforme a interfaz de búfer, se utilizará un búfer de solo lectura del objeto para inicializar el arreglo de bytes.
- Si es un *iterable*, debe ser un iterable de enteros en el rango `0 <= x < 256`, que son usados como los contenidos iniciales del array.

Sin argumento, se crea un array de tamaño 0.

Ver también: Tipos de secuencias binarias — bytes, bytearray y memoryview y Objetos de tipo Bytearray.

class bytes(*source=b''*)
class bytes(*source, encoding, errors='strict'*)

Retorna un nuevo objeto «bytes», que es una secuencia inmutable de enteros en el rango `0 <= x < 256`. **`bytes`** es una versión inmutable de **`bytearray`** – tiene los mismos métodos no mutables y el mismo comportamiento en términos de indexación y segmentación.

En consecuencia, los argumentos del constructor son interpretados como los de **`bytearray()`**.

Los objetos de bytes también pueden ser creados con literales, ver Literales de cadenas y bytes.

Ver también Tipos de secuencias binarias — bytes, bytearray y memoryview, Objetos de tipo Bytes, y Operaciones de bytes y bytearray.

callable(*object*, /)

Return **`True`** if the *object* argument appears callable, **`False`** if not. If this returns `True`, it is still possible that a call fails, but if it is `False`, calling *object* will never succeed. Note that classes are callable (calling a class returns a new instance); instances are callable if their class has a **`__call__()`** method.

Added in version 3.2: Está función fue eliminada en Python 3.0 pero traída de vuelta en Python 3.2.

chr(*codepoint*, /)

Return the string representing a character with the specified Unicode code point. For example, `chr(97)` returns the string `'a'`, while `chr(8364)` returns the string `'€'`. This is the inverse of **`ord()`**.

The valid range for the argument is from 0 through 1,114,111 (0x10FFFF in base 16). **`ValueError`** will be raised if it is outside that range.

@classmethod

Transforma un método en un método de clase.

Un método de clase recibe la clase como primer argumento implícito, de la misma forma que un método de instancia recibe la instancia. Para declarar un método de clase, emplea la siguiente expresión:

```
class C:
    @classmethod
    def f(cls, arg1, arg2): ...
```

La forma `@classmethod` es una función decorator — ver Definiciones de funciones para más detalles.

Un método de clase puede ser llamado en la clase (como `C.f()`) o en la instancia (como `C().f()`). La instancia es ignorada salvo por su clase. Si un método de clase es llamado desde una clase derivada, entonces el objeto de la clase derivada se pasa como primer argumento implícito.

Los métodos de clase son diferentes a los métodos estáticos de C++ o Java. Si los desea, consulte **`staticmethod()`** en esta sección. Para obtener más información sobre los métodos de clase, consulte Jerarquía de tipos estándar.

Distinto en la versión 3.9: Los métodos de clase ahora pueden envolver otros descriptores como **`property()`**.

Distinto en la versión 3.10: Class methods now inherit the method attributes (**`__module__`**, **`__name__`**, **`__qualname__`**, **`__doc__`** and **`__annotations__`**) and have a new `__wrapped__` attribute.

Deprecated since version 3.11, removed in version 3.13: Los métodos de clase ya no pueden envolver otros descriptores como **`property()`**.

compile(*source*, *filename*, *mode*, *flags=0*, *dont_inherit=False*, *optimize=-1*)

Compila el *source* en código o en un objeto AST (*Abstract Syntax Tree*, árbol de sintaxis abstracta). Los objetos código pueden ser ejecutados por las funciones **`exec()`** or **`eval()`**. *source* puede ser una cadena normal, una cadena de bytes o un objeto AST. Para más información sobre cómo trabajar con objetos AST, revisa la documentación del módulo **`ast`**.

El argumento *filename* debería dar el fichero desde el que el código fue leído; pasará un valor reconocible en caso este no fuera leído de un fichero (`'<string>'` es usado comúnmente para esto).

El argumento *mode* especifica que tipo de código debe ser compilado; puede ser `'exec'` si *source* consiste en una secuencia de declaraciones, `'eval'` si consiste de una expresión sencilla, o `'single'` si consiste en una declaración única interactiva (en este último caso, las declaraciones de expresiones que evalúen a algo distinto de `None` serán impresas).

Los argumentos opcionales *flags* y *dont_inherit* controlan qué opciones del compilador deben activarse y cuáles características futuras deben permitirse. Si ninguno está presente (o ambos son cero), el código se compila con los mismos flags que afectan al código que llama **`compile()`**. Si se proporciona el argumento *flags* y *dont_inherit* no es (o es cero), las opciones del compilador y las declaraciones futuras especificadas por el argumento *flags* se utilizan además de las que se utilizarían de todos modos. Si *dont_inherit* es un número entero distinto de cero, entonces el argumento *flags* lo es: los indicadores (características futuras y opciones del compilador) en el código circundante se ignoran.

Las opciones del compilador y las declaraciones futuras se especifican mediante bits que pueden combinarse mediante OR bit a bit para especificar varias opciones. El campo de bits requerido para especificar una característica futura dada se puede encontrar como el atributo **`compiler_flag`** en la instancia **`_Feature`** en el módulo **`__future__`**. Las flags del compilador se pueden encontrar en el módulo **`ast`**, con el prefijo `PyCF_`.

El argumento *optimize* especifica el nivel de optimización del compilador; el valor por defecto de `-1` selecciona el nivel de optimización del intérprete de la misma forma que las opciones `-O`. Los niveles explícitos son `0` (sin optimización; `__debug__` es true), `1` (se eliminan los asserts, `__debug__` es false) or `2` (las *docstrings* también son eliminadas).

This function raises `SyntaxError` or `ValueError` if the compiled source is invalid.

Si quieres transformar código Python a su representación AST, revisa `ast.parse()`.

Lanza un evento de auditoría `compile` con argumentos `source` y `filename`. Este evento también puede ser lanzado por la compilación implícita.

Nota

Cuando se compile una cadena multilínea de código en los modos `'single'` o `'eval'`, la entrada debe terminar con un carácter de nueva línea como mínimo. Esto facilita la detección de declaraciones completas e incompletas en el módulo `code`.

Advertencia

Con una cadena lo suficientemente larga o compleja, al compilar a un objeto AST es posible que el intérprete de Python pare inesperadamente debido a las limitaciones de la profundidad de la pila en el compilador del AST de Python.

Distinto en la versión 3.2: Permite el uso de caracteres de nueva línea de Windows y Mac. También, la entrada en el modo `'exec'` ya no tiene que terminar necesariamente en una nueva línea. Se añade el parámetro *optimize*.

Distinto en la versión 3.5: Anteriormente, un `TypeError` era lanzado cuando se encontraban bytes nulos en *source*.

Added in version 3.8: `ast.PyCF_ALLOW_TOP_LEVEL_AWAIT` puede ahora ser pasado en *flags* para permitir el soporte de

`await`, `async for`, y `async with` en niveles superiores.

class complex(*number=0*, */*)
class complex(*string*, */*)
class complex(*real=0*, *imag=0*)

Convert a single string or number to a complex number, or create a complex number from real and imaginary parts.

Examples:

```
>>> complex('+1.23')
(1.23+0j)
>>> complex('-4.5j')
-4.5j
>>> complex('-1.23+4.5j')
(-1.23+4.5j)
>>> complex('\t( -1.23+4.5J )\n')
(-1.23+4.5j)
>>> complex('-Infinity+NaNj')
(-inf+nanj)
>>> complex(1.23)
(1.23+0j)
>>> complex(imag=-4.5)
-4.5j
>>> complex(-1.23, 4.5)
(-1.23+4.5j)
```

If the argument is a string, it must contain either a real part (in the same format as for **`float()`**) or an imaginary part (in the same format but with a `'j'` or `'J'` suffix), or both real and imaginary parts (the sign of the imaginary part is mandatory in this case). The string can optionally be surrounded by whitespaces and the round parentheses `'('` and `')'`, which are ignored. The string must not contain whitespace between `'+'`, `'-'`, the `'j'` or `'J'` suffix, and the decimal number. For example, `complex('1+2j')` is fine, but `complex('1 + 2j')` raises **`ValueError`**. More precisely, the input must conform to the **`complexvalue`** production rule in the following grammar, after parentheses

and leading and trailing whitespace characters are removed:

```
complexvalue: floatvalue |
              floatvalue ("j" | "J") |
              floatvalue sign absfloatvalue ("j" |
```

If the argument is a number, the constructor serves as a numeric conversion like `int` and `float`. For a general Python object `x`, `complex(x)` delegates to `x.__complex__()`. If `__complex__()` is not defined then it falls back to `__float__()`. If `__float__()` is not defined then it falls back to `__index__()`.

If two arguments are provided or keyword arguments are used, each argument may be any numeric type (including complex). If both arguments are real numbers, return a complex number with the real component *real* and the imaginary component *imag*. If both arguments are complex numbers, return a complex number with the real component `real.real-imag.imag` and the imaginary component `real.imag+imag.real`. If one of arguments is a real number, only its real component is used in the above expressions.

See also `complex.from_number()` which only accepts a single numeric argument.

If all arguments are omitted, returns `0j`.

El tipo complejo está descrito en Tipos numéricos — int, float, complex.

Distinto en la versión 3.6: Agrupar dígitos con guiones bajos como en los literales de código está permitido.

Distinto en la versión 3.8: Recurre a `__index__()` si `__complex__()` y `__float__()` no están definidos.

Obsoleto desde la versión 3.14: Passing a complex number as the *real* or *imag* argument is now deprecated; it should only be passed as a single positional argument.

delattr(*object*, *name*, */*)

Esta función es relativa a `setattr()`. Los argumentos son un objeto y una cadena. La cadena debe ser el mismo nombre que alguno de los atributos del objeto. La función elimina el atributo nombrado, si es que el objeto lo permite. Por ejemplo `delattr(x, 'foobar')` es equivalente a `del x.foobar`. *name* necesita no ser un identificador Python (ver **`setattr()`**).

class dict(***kwargs*)
class dict(*mapping*, */*, ***kwargs*)
class dict(*iterable*, */*, ***kwargs*)

Crea un nuevo diccionario. El objeto `dict` es la clase diccionario. Véase **`dict`** y Tipos mapa — dict para más información sobre esta clase.

Para otros contenedores véase las clases incorporadas (*built-in*) `list`, `set`, y `tuple`, así como el módulo `collections`.

dir()
dir(*object*, */*)

Sin argumentos, retorna la lista de nombres en el ámbito local. Con un argumento, intenta retornar una lista de atributos válidos para ese objeto.

If the object has a method named `__dir__()`, this method will be called and must return the list of attributes. This allows objects that implement a custom `__getattr__()` or `__getattribute__()` function to customize the way `dir()` reports their attributes.

If the object does not provide `__dir__()`, the function tries its best to gather information from the object's `__dict__` attribute, if defined, and from its type object. The resulting list is not necessarily complete and may be inaccurate when the object has a custom `__getattr__()`.

El mecanismo por defecto de `dir()` se comporta de forma distinta con diferentes tipos de objeto, ya que intenta producir la información más relevante en vez de la más completa:

- Si el objeto es un módulo, la lista contiene los nombres de los atributos del módulo.
- Si el objeto es un tipo o una clase, la lista contiene los nombres de sus atributos, y recursivamente la de los atributos de sus clases base.
- En cualquier otro caso, la lista contiene los nombres de los atributos del objeto, los nombres de los atributos de su clase, y recursivamente los atributos de sus clases base.

La lista resultante está ordenada alfabéticamente. Por ejemplo:

```
>>> import struct
>>> dir()   # show the names in the module namespac
['__builtins__', '__name__', 'struct']
>>> dir(struct)   # show the names in the struct mo
['Struct', '__all__', '__builtins__', '__cached__',
 '__initializing__', '__loader__', '__name__', '__p
 '_clearcache', 'calcsize', 'error', 'pack', 'pack_
 'unpack', 'unpack_from']
>>> class Shape:
...     def __dir__(self):
...         return ['area', 'perimeter', 'location'
...
>>> s = Shape()
>>> dir(s)
['area', 'location', 'perimeter']
```

Nota

Ya que `dir()` se ofrece como una herramienta para uso en una terminal de forma interactiva, intenta ofrecer un grupo interesante de nombres más que un uno rigurosamente definido, y su comportamiento detallado puede cambiar entre versiones. Por ejemplo, los atributos de metaclase no están en la lista resultante cuando el argumento es una clase.

divmod(*a*, *b*, */*)

Take two (non-complex) numbers as arguments and return a pair of numbers consisting of their quotient and remainder when using integer division. With mixed operand types, the rules for binary arithmetic operators apply. For integers, the result is the same as `(a // b, a % b)`. For floating-point numbers the result is `(q, a % b)`, where *q* is usually `math.floor(a / b)` but may be 1 less than that. In any case `q * b + a % b` is very close to *a*, if `a % b` is non-zero it has the same sign as *b*, and `0 <= abs(a % b) < abs(b)`.

enumerate(*iterable*, *start=0*)

Retorna un objeto enumerador. *iterable* tiene que ser una secuencia, un iterator, o algún otro objeto que soporte la iteración. El método **`__next__()`** del iterador retornado por la función **`enumerate()`** retorna una tupla que contiene un contador (desde *start*, cuyo valor por defecto es 0) y los valores obtenidos al iterar sobre *iterable*.

```
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter'
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Wi
>>> list(enumerate(seasons, start=1))
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Wi
```

Equivalente a:

```
def enumerate(iterable, start=0):
    n = start
    for elem in iterable:
        yield n, elem
        n += 1
```

eval(*source*, */*, *globals=None*, *locals=None*)

Parámetros:

- **source** (`str` | code object) – A Python expression.
- **globals** (`dict` | `None`) – The global namespace (default: `None`).
- **locals** (mapping | `None`) – The local namespace (default: `None`).

Devuelve:

The result of the evaluated expression.

Raises:

Syntax errors are reported as exceptions.

Advertencia

This function executes arbitrary code. Calling it with untrusted user-supplied input will lead to security vulnerabilities.

The *source* argument is parsed and evaluated as a Python expression (technically speaking, a condition list) using the *globals* and *locals* mappings as global and local namespace. If the *globals* dictionary is present and does not contain a value for the key `__builtins__`, a reference to the dictionary of the built-in module `builtins` is inserted under that key before *source* is parsed. Overriding `__builtins__` can be used to restrict or change the available names, but this is **not** a security mechanism: the executed code can still access all builtins. If the *locals* mapping is omitted it defaults to the *globals* dictionary. If both mappings are omitted, the source is executed with the *globals* and *locals* in the environment where `eval()` is called. Note, *eval()* will only have access to the nested scopes (non-locals) in the enclosing environment if they are already referenced in the scope that is calling `eval()` (e.g. via a `nonlocal` statement).

Example:

```
>>> x = 1
>>> eval('x+1')
2
```

Esta función también puede ser utilizada para ejecutar objetos de código arbitrario (como los que crea la función `compile()`). En este caso, se pasa un objeto de código en vez de una cadena de caracteres. Si el objeto de código ha sido compilado usando `'exec'` como el argumento *mode*, el valor que retornará `eval()` será `None`.

Consejos: la ejecución dinámica de declaraciones está soportada por la función `exec()`. Las funciones `globals()` y `locals()` retornan los diccionarios global y local en ese momento, respectivamente, lo cual puede ser útil para su uso en `eval()` o `exec()`.

Si la fuente dada es una cadena de caracteres, se eliminan los espacios y tabulaciones principales y finales.

See `ast.literal_eval()` for a function to evaluate strings with expressions containing only literals.

Lanza un evento de auditoría `exec` con el objeto código como el argumento. También podrían ser lanzados eventos de compilación de código.

Distinto en la versión 3.13: The *globals* and *locals* arguments can now be passed as keywords.

Distinto en la versión 3.13: The semantics of the default *locals* namespace have been adjusted as described for the `locals()` builtin.

exec(*source*, */*, *globals=None*, *locals=None*, ***, *closure=None*)

Advertencia

This function executes arbitrary code. Calling it with untrusted user-supplied input will lead to security vulnerabilities.

This function supports dynamic execution of Python code. *source* must be either a string or a code object. If it is a string, the string is parsed as a suite of Python statements which is then executed (unless a syntax error occurs). [1] If it is a code object, it is simply executed. In all cases, the code that's executed is expected to be valid as file input (see the section Entrada de archivo in the Reference Manual). Be aware that the **`nonlocal`**, **`yield`**, and **`return`** statements may not be used outside of function definitions even within the context of code passed to the **`exec()`** function. The return value is `None`.

In all cases, if the optional parts are omitted, the code is executed in the current scope. If only *globals* is provided, it must be a dictionary (and not a subclass of dictionary), which will be used for both the global and the local variables. If *globals* and *locals* are given, they are used for the global and local variables, respectively. If provided, *locals* can be any mapping object. Remember that at the module level, globals and locals are the same dictionary.

Nota

When `exec` gets two separate objects as *globals* and *locals*, the code will be executed as if it were embedded in a class definition. This means functions and classes defined in the executed code will not be able to access variables assigned at the top level (as the «top level» variables are treated as class variables in a class definition).

If the *globals* dictionary does not contain a value for the key `__builtins__`, a reference to the dictionary of the built-in module **`builtins`** is inserted under that key. Overriding `__builtins__` can be used to restrict or change the available names, but this is **not** a security mechanism: the executed code can still access all builtins.

The *closure* argument specifies a closure–a tuple of cellvars. It's only valid when the *object* is a code object containing free (closure) variables. The length of the tuple must exactly

match the length of the code object's `co_freevars` attribute.

Lanza un evento de auditoría `exec` con el objeto código como el argumento. También podrían ser lanzados eventos de compilación de código.

Nota

The built-in functions `globals()` and `locals()` return the current global and local namespace, respectively, which may be useful to pass around for use as the second and third argument to `exec()`.

Nota

The default *locals* act as described for function `locals()` below. Pass an explicit *locals* dictionary if you need to see effects of the code on *locals* after function `exec()` returns.

Distinto en la versión 3.11: Añadido el parámetro *closure*.

Distinto en la versión 3.13: The *globals* and *locals* arguments can now be passed as keywords.

Distinto en la versión 3.13: The semantics of the default *locals* namespace have been adjusted as described for the `locals()` builtin.

filter(*function*, *iterable*, */*)

Construye un iterador a partir de aquellos elementos de *iterable* para los cuales *function* retorna true. *iterable* puede ser una secuencia, un contenedor que soporta iteración, o un iterador. Si *function* es `None`, se asume la función identidad, es decir, todos los elementos de *iterable* que son false son eliminados.

Ten en cuenta que `filter(function, iterable)` es equivalente a la expresión de un generador `(item for item in iterable if function(item))` si *function* no es `None` y a `(item for item in iterable if item)` si *function* es `None`.

Ver **`itertools.filterfalse()`** para la función complementaria que retorna los elementos de *iterable* para los cuales *function* retorna false.

class float(*number=0.0*, */*)

class float(*string*, */*)

Return a floating-point number constructed from a number or a string.

Examples:

```
>>> float('+1.23')
1.23
>>> float('   -12345\n')
-12345.0
>>> float('1e-003')
0.001
>>> float('+1E6')
1000000.0
>>> float('-Infinity')
-inf
```

If the argument is a string, it should contain a decimal number, optionally preceded by a sign, and optionally embedded in whitespace. The optional sign may be `'+'` or `'-'`; a `'+'` sign has no effect on the value produced. The argument may also be a string representing a NaN (not-a-number), or positive or negative infinity. More precisely, the input must conform to the **`floatvalue`** production rule in the following grammar, after leading and trailing whitespace characters are removed:

```
sign:          "+" | "-"
infinity:      "Infinity" | "inf"
nan:           "nan"
```

```
digit:          <a Unicode decimal digit, i.e. chara
digitpart:      digit (["_"] digit)*
number:         [digitpart] "." digitpart | digitpar
exponent:       ("e" | "E") [sign] digitpart
floatnumber:    number [exponent]
absfloatvalue:  floatnumber | infinity | nan
floatvalue:     [sign] absfloatvalue
```

Case is not significant, so, for example, «inf», «Inf», «INFINITY», and «iNfINity» are all acceptable spellings for positive infinity.

Otherwise, if the argument is an integer or a floating-point number, a floating-point number with the same value (within Python's floating-point precision) is returned. If the argument is outside the range of a Python float, an `OverflowError` will be raised.

Para el objeto general de Python `x`, `float(x)` delega a `x.__float__()`. Si `__float__()` no está definido entonces recurre a `__index__()`.

See also `float.from_number()` which only accepts a numeric argument.

Si no se le da un argumento, retorna `0.0`.

El tipo float está descrito en Tipos numéricos — int, float, complex.

Distinto en la versión 3.6: Agrupar dígitos con guiones bajos como en los literales de código está permitido.

Distinto en la versión 3.7: The parameter is now positional-only.

Distinto en la versión 3.8: Recurre a `__index__()` si `__float__()` no está definido.

format(*value*, *format_spec*='', /)

Convierte *value* a su representación «con formato», de forma controlada por *format_spec*. La interpretación de *format_spec* dependerá del tipo del argumento *value*. Sin embargo, hay una sintaxis estándar de formato que emplean la mayoría de los tipos incorporados: Format specification mini-language.

El *format_spec* por defecto es una cadena vacía que normalmente produce el mismo efecto que llamar a **`str(value)`**.

A call to `format(value, format_spec)` is translated to `type(value).__format__(value, format_spec)` which bypasses the instance dictionary when searching for the value's **`__format__()`** method. A **`TypeError`** exception is raised if the method search reaches **`object`** and the *format_spec* is non-empty, or if either the *format_spec* or the return value are not strings.

Distinto en la versión 3.4: `object().__format__(format_spec)` lanza **`TypeError`** si *format_spec* no es una cadena vacía.

class frozenset(*iterable=()*, */*)

Retorna un nuevo objeto **`frozenset`**, opcionalmente con elementos tomados de *iterable*. `frozenset` es una clase built-in. Ver **`frozenset`** y Conjuntos — set, frozenset para documentación sobre esta clase.

Para otro tipo de contenedores, ver las clases built-in **`set`**, **`list`**, **`tuple`**, y **`dict`**, así como el módulo **`collections`**.

getattr(*object*, *name*, */*)
getattr(*object*, *name*, *default*, */*)

Retorna el valor del atributo nombrado de *object*. *name* debe ser una cadena. Si la cadena es el nombre de uno de los atributos del objeto, el resultado es el valor de ese atributo. Por ejemplo, `getattr(x, 'foobar')` es equivalente a `x.foobar`. Si el atributo nombrado no existe, se retorna *default* si ha sido proporcionado como argumento, y sino se lanza una excepción **`AttributeError`**. *name* no necesita ser un identificador de Python (ver `setattr()`).

Nota

Dado que private name mangling ocurre en el momento de la compilación, uno debe enredar manualmente el nombre de un atributo privado (atributos con dos guiones bajos principales) para recuperarlo con `getattr()`.

globals()

Retorna un diccionario que implementa el espacio de nombres del módulo actual. Para el código dentro de las funciones, esto se establece cuando se define la función y permanece igual independientemente de dónde se llame a la función.

hasattr(*object*, *name*, */*)

Los argumentos son un objeto y una cadena. El resultado es `True` si la cadena es el nombre de uno de los atributos del objeto, y `False` en caso contrario. (Está implementado mediante una llamada a `getattr(object, name)` que comprueba si se lanza una excepción `AttributeError` o no).

hash(*object*, */*)

Retorna el valor hash del objeto (si tiene uno). Los valores hash son enteros. Se usan para comparar de forma rápida claves de diccionarios durante las operaciones de búsqueda. Valores numéricos que son iguales tienen el mismo valor hash (incluso si son de tipos diferentes, como es el caso para 1 y 1.0).

Nota

For objects with custom `__hash__()` methods, note that `hash()` truncates the return value based on the bit width of the host machine.

help()
help(*request*)

Invoca el sistema de ayuda integrado (built-in). (Esta función está indicada para su uso interactivo). Si no se le da argumento, el sistema interactivo de ayuda se inicia en la consola del intérprete. Si el argumento es una cadena, entonces es buscada como nombre de un módulo, función, clase, método, palabra clave o tema de documentación, y una página de ayuda es impresa en la consola. Si el argumento es cualquier otro tipo de objeto, una página de ayuda sobre el objeto es generada.

Ten en cuenta que si aparece una barra(/) en la lista de parámetros de una función al invocar `help()`, significa que los parámetros anteriores a la barra son solo posicionales. Para más información, puedes ver the FAQ entry on positional-only parameters.

Esta función se añade al espacio de nombres built-in a través del módulo `site`.

Distinto en la versión 3.4: Cambios a los módulos `pydoc` y `inspect` implican que las signaturas reportadas para objetos invocables son más completas y consistentes.

hex(*integer*, */*)

Convert an integer number to a lowercase hexadecimal string prefixed with «0x». If *integer* is not a Python `int` object, it has to define an `__index__()` method that returns an integer. Some examples:

```
>>> hex(255)
'0xff'
>>> hex(-42)
'-0x2a'
```

Si quieres convertir un número entero a una cadena hexadecimal de mayúsculas o minúsculas con prefijo o sin el, puedes usar cualquiera de las siguientes formas:

```
>>> '%#x' % 255, '%x' % 255, '%X' % 255
('0xff', 'ff', 'FF')
>>> format(255, '#x'), format(255, 'x'), format(255
('0xff', 'ff', 'FF')
>>> f'{255:#x}', f'{255:x}', f'{255:X}'
('0xff', 'ff', 'FF')
```

Véase también `format()` para más información.

Ver también `int()` para convertir una cadena hexadecimal a un entero usando una base de 16.

Nota

Para obtener una cadena hexadecimal que represente un punto flotante, utiliza el método `float.hex()`.

id(*object*, */*)

Retorna la «identidad» de un objeto. Esto es un entero que está garantizado que es único y constante para este objeto durante toda su existencia. Dos objetos con existencias en el tiempo que no coincidan pueden tener el mismo valor de `id()`.

Esta es la dirección del objeto en memoria.

Lanza un evento de auditoría `builtins.id` con el argumento `id`.

input()
input(*prompt*, */*)

Si el argumento *prompt* está presente, se escribe a la salida estándar sin una nueva línea a continuación. La función lee entonces una línea de la entrada, la convierte en una cadena (eliminando la nueva línea), y retorna eso. Cuando se lee EOF, se lanza una excepción `EOFError`. Ejemplo:

```
>>> s = input('--> ')
--> Monty Python's Flying Circus
>>> s
"Monty Python's Flying Circus"
```

Si el módulo **`readline`** estaba cargado, entonces **`input()`** lo usará para proporcionar características más elaboradas de edición de líneas e historiales.

Lanza un evento de auditoría `builtins.input` con el argumento `prompt` antes de leer entrada

Lanza un evento de auditoría `builtins.input/result` con el resultado justo después de haber leído con éxito la entrada.

class int(*number=0*, */*)
class int(*string*, */*, *base=10*)

Return an integer object constructed from a number or a string, or return `0` if no arguments are given.

Examples:

```
>>> int(123.45)
123
>>> int('123')
123
>>> int('   -12_345\n')
-12345
>>> int('FACE', 16)
64206
>>> int('0xface', 0)
64206
>>> int('01110011', base=2)
115
```

If the argument defines **`__int__()`**, `int(x)` returns `x.__int__()`. If the argument defines **`__index__()`**, it returns `x.__index__()`. For floating-point numbers, this truncates towards zero.

If the argument is not a number or if *base* is given, then it must be a string, **`bytes`**, or **`bytearray`** instance representing an integer in radix *base*. Optionally, the string can be preceded by `+` or `-` (with no space in between), have leading zeros, be surrounded by whitespace, and have single underscores interspersed between digits.

Una cadena de enteros de base n contiene dígitos, cada uno de los cuales representa un valor de 0 a n-1. Los valores 0–9 se pueden representar por cualquier dígito decimal Unicode. Los valores 10–35 se pueden representar de `a` a `z` (o de `A` a `Z`). La *base* predeterminada es 10. Las bases permitidas son 0 y 2–36. Las cadenas de caracteres de base 2, 8 y 16 pueden tener opcionalmente el prefijo `0b`/`0B`, `0o`/`0O` o `0x`/`0X`, como ocurre con los literales enteros en el código. Para base 0, la cadena de caracteres se interpreta de manera similar a un literal entero en el código, en el que la base de verdad es 2, 8, 10 o 16 según lo determine el prefijo. La base 0 tampoco permite los ceros a la izquierda: `int('010', 0)` no es legal, mientras que `int('010')` e `int('010', 8)` sí lo son.

El tipo entero se describe en Tipos numéricos — int, float, complex.

Distinto en la versión 3.4: Si *base* no es una instancia de **`int`** y el objeto *base* tiene un método **`base.__index__`**, ese método es llamado para obtener un entero para esa base. En versiones anteriores se empleaba **`base.__int__`** en vez de **`base.__index__`**.

Distinto en la versión 3.6: Agrupar dígitos con guiones bajos como en los literales de código está permitido.

Distinto en la versión 3.7: The first parameter is now positional-only.

Distinto en la versión 3.8: Recurre a **`__index__()`** si no está definido **`__int__()`**.

Distinto en la versión 3.11: **`int`** string inputs and string representations can be limited to help avoid denial of service attacks. A **`ValueError`** is raised when the limit is exceeded while converting a string to an **`int`** or when converting an **`int`** into a string would exceed the limit. See the integer string conversion length limitation documentation.

Distinto en la versión 3.14: **`int()`** no longer delegates to the **`__trunc__()`** method.

isinstance(*object*, *classinfo*, */*)

Retorna `True` si el argumento *object* es una instancia del argumento *classinfo*, o de una subclase (directa, indirecta o virtual) del mismo. Si *object* no es un objeto del tipo indicado, esta función siempre retorna `False`. Si *classinfo* es una tupla de objetos de tipo (o recursivamente, otras tuplas lo son) o Tipo de conversión de múltiples tipos , retorna `True` si *object* es una instancia de alguno de los tipos. Si *classinfo* no es un tipo, una tupla de tipos, o una tupla de tuplas de tipos, una excepción **`TypeError`** es lanzada. Es posible que **`TypeError`** no se genere para un tipo no válido si una verificación anterior tiene éxito.

Distinto en la versión 3.10: classinfo puede ser una Tipo de conversión.

issubclass(*class*, *classinfo*, */*)

Retorna `True` si *class* es una subclase (directa, indirecta o virtual) de *classinfo*. Una clase es considerada una subclase de sí misma. *classinfo* puede ser una tupla de objetos de clase o una Tipo de conversión, en cuyo caso retorna `True` si *class* es una subclase de cualquier entrada en *classinfo*. En cualquier otro caso, se lanzará una excepción **`TypeError`**.

Distinto en la versión 3.10: classinfo puede ser una Tipo de conversión.

iter(*iterable*, */*)
iter(*callable*, *sentinel*, */*)

Return an iterator object. The first argument is interpreted very differently depending on the presence of the second argument. Without a second argument, the single argument must be a collection object which supports the iterable protocol (the `__iter__()` method), or it must support the sequence protocol (the `__getitem__()` method with integer arguments starting at `0`). If it does not support either of those protocols, `TypeError` is raised. If the second argument, *sentinel*, is given, then the first argument must be a callable object. The iterator created in this case will call *callable* with no arguments for each call to its `__next__()` method; if the value returned is equal to *sentinel*, `StopIteration` will be raised, otherwise the value will be returned.

Ver también Tipos de iteradores.

Una aplicación muy útil de la segunda forma de `iter()` es la construcción de un lector de bloques. Por ejemplo, leer bloques de ancho fijo de una base de datos binaria hasta que el fin del fichero sea alcanzado:

```
from functools import partial
with open('mydata.db', 'rb') as f:
    for block in iter(partial(f.read, 64), b''):
        process_block(block)
```

len(*object*, */*)

Retorna el tamaño (el número de elementos) de un objeto. El argumento puede ser una secuencia (como una cadena, un objeto byte, una tupla, lista o rango) o una colección (como un diccionario, un set o un *frozen set*).

`len` aumenta `OverflowError` en longitudes mayores que `sys.maxsize`, como `range(2 ** 100)`.

class list(*iterable*=*()*, */*)

Más que una función, `list` es realmente un tipo de secuencia mutable, como está documentado en Listas y Tipos secuencia — list, tuple, range.

locals()

Return a mapping object representing the current local symbol table, with variable names as the keys, and their currently bound references as the values.

At module scope, as well as when using **`exec()`** or **`eval()`** with a single namespace, this function returns the same namespace as **`globals()`**.

At class scope, it returns the namespace that will be passed to the metaclass constructor.

When using `exec()` or `eval()` with separate local and global arguments, it returns the local namespace passed in to the function call.

In all of the above cases, each call to `locals()` in a given frame of execution will return the *same* mapping object. Changes made through the mapping object returned from `locals()` will be visible as assigned, reassigned, or deleted local variables, and assigning, reassigning, or deleting local variables will immediately affect the contents of the returned mapping object.

In an optimized scope (including functions, generators, and coroutines), each call to `locals()` instead returns a fresh dictionary containing the current bindings of the function's local variables and any nonlocal cell references. In this case, name binding changes made via the returned dict are *not* written back to the corresponding local variables or nonlocal cell references, and assigning, reassigning, or deleting local variables and nonlocal cell references does *not* affect the contents of previously returned dictionaries.

Calling `locals()` as part of a comprehension in a function, generator, or coroutine is equivalent to calling it in the containing scope, except that the comprehension's initialised iteration variables will be included. In other scopes, it behaves as if the comprehension were running as a nested function.

Calling `locals()` as part of a generator expression is equivalent to calling it in a nested generator function.

Distinto en la versión 3.12: The behaviour of `locals()` in a comprehension has been updated as described in **PEP 709** [https://peps.python.org/pep-0709/].

Distinto en la versión 3.13: As part of **PEP 667** [https://peps.python.org/pep-0667/], the semantics of mutating the mapping objects returned from this function are now defined. The behavior in optimized scopes is now as described above. Aside from being defined, the behaviour in other scopes remains unchanged from previous versions.

map(*function*, *iterable*, */*, **iterables*, *strict=False*)

Return an iterator that applies *function* to every item of *iterable*, yielding the results. If additional *iterables* arguments are passed, *function* must take that many arguments and is applied to the items from all iterables in parallel. With multiple iterables, the iterator stops when the shortest iterable is exhausted. If *strict* is `True` and one of the iterables is exhausted before the others, a **`ValueError`** is raised. For cases where the function inputs are already arranged into argument tuples, see **`itertools.starmap()`**.

Distinto en la versión 3.14: Added the *strict* parameter.

max(*iterable*, */*, ***, *key=None*)
max(*iterable*, */*, ***, *default*, *key=None*)
max(*arg1*, *arg2*, */*, **args*, *key=None*)

Retorna el elemento mayor en un iterable o el mayor de dos o más argumentos.

Si un argumento posicional es dado, debe ser un iterable. El elemento mayor en el iterable es retornado. Si dos o más argumentos posicionales son indicados, el mayor de los argumentos posicionales será retornado.

Hay dos argumentos de solo palabra clave que son opcionales.

El argumento *key* especifica una función de ordenación de un sólo argumento, como la usada para **`list.sort()`**. El argumento *default* especifica un objeto a retornar si el iterable proporcionado está vacío. Si el iterable está vacío y *default* no ha sido indicado, se lanza un **`ValueError`**.

Si hay múltiples elementos con el valor máximo, la función retorna el primero que ha encontrado. Esto es consistente con otras herramientas para preservar la estabilidad de la ordenación como `sorted(iterable, key=keyfunc, reverse=True)[0]` y `heapq.nlargest(1, iterable, key=keyfunc)`.

Distinto en la versión 3.4: Added the *default* keyword-only parameter.

Distinto en la versión 3.8: *key* puede ser `None`.

class memoryview(*object*)

Retorna un objeto *«memory view»* creado a partir del argumento indicado. Para más información ver Vistas de memoria.

min(*iterable*, */*, ***, *key=None*)
min(*iterable*, */*, ***, *default*, *key=None*)
min(*arg1*, *arg2*, */*, **args*, *key=None*)

Retorna el menor elemento en un iterable o el menor de dos o más argumentos.

Si se le indica un argumento posicional, debe ser un iterable. El menor elemento del iterable es retornado. Si dos o más argumentos posicionales son indicados, el menor de los argumentos posicionales es retornado.

Hay dos argumentos de solo palabra clave que son opcionales. El argumento *key* especifica una función de ordenación de un sólo argumento, como la usada para **`list.sort()`**. El argumento *default* especifica un objeto a retornar si el iterable proporcionado está vacío. Si el iterable está vacío y *default* no ha sido indicado, se lanza un `ValueError`.

Si hay múltiples elementos con el valor mínimo, la función retorna el primero que encuentra. Esto es consistente con otras herramientas que preservan la estabilidad de la ordenación como `sorted(iterable, key=keyfunc)[0]` y `heapq.nsmallest(1, iterable, key=keyfunc)`.

Distinto en la versión 3.4: Added the *default* keyword-only parameter.

Distinto en la versión 3.8: *key* puede ser `None`.

next(*iterator*, */*)
next(*iterator*, *default*, */*)

Extrae el siguiente elemento de iterator llamando a su método `__next__()`. Si se le indica *default*, éste será retornado si se agota el iterador, de lo contrario, se lanza un `StopIteration`.

class object

This is the ultimate base class of all other classes. It has methods that are common to all instances of Python classes. When the constructor is called, it returns a new featureless object. The constructor does not accept any arguments.

Nota

`object` instances do *not* have `__dict__` attributes, so you can't assign arbitrary attributes to an instance of **`object`**.

oct(*integer*, */*)

Convert an integer number to an octal string prefixed with «0o». The result is a valid Python expression. If *integer* is not a Python `int` object, it has to define an `__index__()` method that returns an integer. For example:

```
>>> oct(8)
'0o10'
>>> oct(-56)
'-0o70'
```

Si quieres convertir un número entero a una cadena octal, tanto con prefijo «0o» como sin el, puedes usar cualquiera de las siguientes formas.

```
>>> '%#o' % 10, '%o' % 10
('0o12', '12')
>>> format(10, '#o'), format(10, 'o')
('0o12', '12')
>>> f'{10:#o}', f'{10:o}'
('0o12', '12')
```

Véase también **`format()`** para más información.

open(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*, *opener=None*)

Abre *file* y retorna un file object correspondiente. Si el archivo no se puede abrir, se lanza un **`OSError`**. Consulte Leyendo y escribiendo archivos para obtener más ejemplos de cómo utilizar esta función.

file es un path-like object que da la ruta (absoluta o relativa al directorio de trabajo actual) del fichero a ser abierto o un descriptor de fichero entero del fichero a ser envuelto. (Si un descriptor de fichero es dado, será cerrado cuando el objeto de entrada-salida sea cerrado, a menos que *closefd* esté puesto a `False`.)

mode is an optional string that specifies the mode in which the file is opened. It defaults to `'r'` which means open for reading in text mode. Other common values are `'w'` for writing (truncating the file if it already exists), `'x'` for exclusive creation, and `'a'` for appending (which on *some* Unix systems, means that *all* writes append to the end of the file regardless of the current seek position). In text mode, if *encoding* is not specified the encoding used is platform-dependent: `locale.getencoding()` is called to get the current locale encoding. (For reading and writing raw bytes use binary mode and leave *encoding* unspecified.) The available modes are:

Carácter	**Significado**
`'r'`	abierto para lectura (por defecto)
`'w'`	abierto para escritura, truncando primero el fichero
`'x'`	abierto para creación en exclusiva, falla si el fichero ya existe
`'a'`	abierto para escritura, añadiendo al final del fichero si este existe
`'b'`	modo binario
`'t'`	modo texto (por defecto)
`'+'`	abierto para actualizar (lectura y escritura)

El modo por defecto es `'r'` (abierto para lectura de texto, sinónimo de `'rt'`. Los modos `'w+'` y `'w+b'` abren y truncan el fichero. Los modos `'r+'` y `'r+b'` abren el fichero sin truncarlo.

Como se menciona en Resumen, Python distingue entre I/O binario y de texto. Los ficheros abiertos en modo binario (incluyendo `'b'` en el argumento *mode*) retornan su contenido como objetos de `bytes` sin ninguna descodificación. En modo de texto (por defecto, o cuando se incluye `'t'` en el argumento *mode*), los contenidos del fichero se retornan como `str`, tras descodificar los *bytes* usando una codificación dependiente de plataforma o usando el *encoding* especificado como argumento.

Nota

Python no depende de la noción de ficheros de texto del sistema operativo subyacente; todo el procesado lo hace Python, y es por tanto independiente de plataforma.

buffering is an optional integer used to set the buffering policy. Pass 0 to switch buffering off (only allowed in binary mode), 1 to select line buffering (only usable when writing in text mode), and an integer > 1 to indicate the size in bytes of a fixed-size chunk buffer. Note that specifying a buffer size this way applies for binary buffered I/O, but `TextIOWrapper` (i.e., files opened with `mode='r+'`) would have another buffering. To disable buffering in `TextIOWrapper`, consider using the `write_through` flag for **`io.TextIOWrapper.reconfigure()`**. When no *buffering* argument is given, the default buffering policy works as follows:

- Binary files are buffered in fixed-size chunks; the size of the buffer is `max(min(blocksize, 8 MiB), DEFAULT_BUFFER_SIZE)` when the device block size is available. On most systems, the buffer will typically be 128 kilobytes long.
- Los ficheros de texto «interactivos» (ficheros para los cuales **`isatty()`** retorna `True`) usan buffering por líneas. Otros ficheros de texto emplean la norma descrita anteriormente para ficheros binarios.

encoding es el nombre de la codificación empleada con el fichero. Esto solo debe ser usado en el modo texto. La codificación por defecto es dependiente de plataforma (aquello que retorna **`locale.getencoding()`**), pero puede emplearse cualquier text encoding soportado por Python. Ver el módulo **`codecs`** para la lista de codificaciones soportadas.

errors es una cadena opcional que especifica como deben manejarse los errores de codificación y descodificación – esto no puede ser usado en modo binario. Están disponibles varios gestores de error (listados en Manejadores de errores), pero también es válido cualquier nombre de gestión de error registrado con **`codecs.register_error()`**. Los nombres estándar incluyen:

- `'strict'` para lanzar una excepción **`ValueError`** si hay un error de codificación. El valor por defecto, `None`, produce el mismo efecto.
- `'ignore'` ignora los errores. Nótese que ignorar errores de codificación puede conllevar la pérdida de

datos.

- `'replace'` provoca que se inserte un marcador de reemplazo (como `'?'`) en aquellos sitios donde hay datos malformados.
- `'surrogateescape'` representará cualquier bytes incorrectos como unidades de código sustituto bajo que van desde U+DC80 a U+DCFF. Estas unidades de código sustituto volverán a convertirse en los mismos bytes cuando el gestor de errores `surrogateescape` sea usado al escribir datos. Esto es útil para el procesado de ficheros con una codificación desconocida.
- `'xmlcharrefreplace'` está soportado solamente cuando se escribe a un fichero. Los caracteres que no estén soportados por la codificación son reemplazados por la referencia al carácter XML apropiado `&#nnn;`.
- `'backslashreplace'` reemplaza datos malformados con las secuencias de escapes de barra invertida de Python.
- `'namereplace'` reemplaza caracteres no soportados con secuencias de escape `\N{...}` (y también está sólo soportado en escritura).

newline determina cómo analizar los caracteres de nueva línea de la secuencia. Puede ser `None`, `''`, `'\n'`, `'\r'`, y `'\r\n'`. Funciona de la siguiente manera:

- Cuando se está leyendo entrada desde el flujo, si *newline* es `None`, el modo *universal newlines* es activado. Las líneas en la entrada pueden terminar en `'\n'`, `'\r'`, o `'\r\n'`, y serán traducidas a `'\n'` antes de ser retornadas a la entidad que llama. Si es `''`, el modo *universal newlines* estará activado pero los finales de línea se retornan sin traducir a la entidad que llama. Si tiene cualquiera de los otros valoras válidos, las líneas de entrada estarán terminadas sólo por la cadena dada, y los finales de línea serán retornados sin traducir a la entidad que llama.
- Cuando se escribe salida al flujo, si *newline* es `None`, cualquier carácter `'\n'` escrito es traducido al

separador de línea por defecto en el sistema, **`os.linesep`**. Si *newline* es `''` o `'\n'`, entonces no se produce ninguna traducción. Si *newline* toma cualquiera de los otros valores válidos, entonces cualquier carácter `'\n'` escrito es traducido a la cadena indicada.

Si *closefd* es `False` y se indica un descriptor de fichero en vez de un nombre de fichero, el descriptor del fichero se mantendrá abierto cuando se cierre el fichero. Si se indica un nombre de fichero, *closefd* debe ser `True` (lo es por defecto), ya que de otra forma se lanzará un error.

Un abridor personalizado puede emplearse pasando un invocable como *opener*. El descriptor de fichero del objeto se obtiene llamando a *opener* con (*file*, *flags*). *opener* debe retornar un descriptor de fichero abierto (pasando **`os.open`** como *opener* resulta en una funcionalidad similar a `None`).

El nuevo fichero creado es no-heredable.

El siguiente ejemplo emplea el parámetro dir_fd de la función **`os.open()`** para abrir un fichero relativo a un directorio dado:

```
>>> import os
>>> dir_fd = os.open('somedir', os.O_RDONLY)
>>> def opener(path, flags):
...     return os.open(path, flags, dir_fd=dir_fd)
...
>>> with open('spamspam.txt', 'w', opener=opener) a
...     print('This will be written to somedir/spam
...
>>> os.close(dir_fd)  # don't leak a file descripto
```

El tipo de file object retornado por la función **`open()`** depende del modo. Cuando se emplea **`open()`** para abrir un fichero en modo texto (`'w'`, `'r'`, `'wt'`, `'rt'`, etc.), retorna una subclase de **`io.TextIOBase`** (específicamente **`io.TextIOWrapper`**). Cuando se emplea para abrir un fichero en modo binario con buffering, la clase retornada es una subclase de **`io.BufferedIOBase`**. La clase exacta varía: en modo binario de lectura, retorna **`io.BufferedReader`**; en modo de escritura y adición en binario, retorna **`io.BufferedWriter`**, y en modo de escritura/lectura, retorna **`io.BufferedRandom`**. Si el buffering está desactivado, el flujo en crudo, una subclase de **`io.RawIOBase`**, **`io.FileIO`**, es retornada.

Véase también los módulos para manejo de ficheros, como **`fileinput`**, **`io`** (donde es declarada **`open()`**), **`os`**, **`os.path`**, **`tempfile`**, y **`shutil`**.

Raises an auditing event `open` with arguments `path`, `mode`, `flags`.

Los argumentos `mode` y `flags` pueden haber sido modificados o inferidos de la llamada original.

Distinto en la versión 3.3:

- El parámetro *opener* fue añadido.
- El modo `'x'` fue añadido.
- **`IOError`** era la excepción lanzada anteriormente, ahora es un alias de **`OSError`**.
- Se lanza **`FileExistsError`** si ya existe el fichero abierto en modo de creación exclusiva (`'x'`).

Distinto en la versión 3.4:

- El fichero ahora es no-heredable.

Distinto en la versión 3.5:

- Si la llamada al sistema es interrumpida y el gestor de señales no lanza una excepción, ahora la función reintenta la llamada de sistema en vez de lanzar una excepción **`InterruptedError`** (véase **PEP 475** [https://peps.python.org/pep-0475/] para la justificación).
- El gestor de errores `'namereplace'` fue añadido.

Distinto en la versión 3.6:

- Añadido el soporte para aceptar objetos que

implementan **`os.PathLike`**.
- En Windows, abrir un búfer en la consola puede retornar una subclase de **`io.RawIOBase`** en vez de **`io.FileIO`**.

Distinto en la versión 3.11: El modo `'X'` has ido eliminado.

ord(*character*, */*)

Return the ordinal value of a character.

If the argument is a one-character string, return the Unicode code point of that character. For example, `ord('a')` returns the integer `97` and `ord('€')` (Euro sign) returns `8364`. This is the inverse of **`chr()`**.

If the argument is a **`bytes`** or **`bytearray`** object of length 1, return its single byte value. For example, `ord(b'a')` returns the integer `97`.

pow(*base*, *exp*, *mod=None*)

Retorna *base* elevado a *exp*; si *mod* está presente, retorna *base* elevado a *exp*, módulo *mod* (calculado de manera más eficiente que `pow(base, exp) % mod`). La forma con dos argumentos `pow(base, exp)` es equivalente a usar el operador potencia: `base**exp`.

When arguments are builtin numeric types with mixed operand types, the coercion rules for binary arithmetic operators apply. For **`int`** operands, the result has the same type as the operands (after coercion) unless the second argument is negative; in that case, all arguments are converted to float and a float result is delivered. For example, `pow(10, 2)` returns `100`, but `pow(10, -2)` returns `0.01`. For a negative base of type **`int`** or **`float`** and a non-integral exponent, a complex result is delivered. For example, `pow(-9, 0.5)` returns a value close to `3j`. Whereas, for a negative base of type **`int`** or **`float`** with an integral exponent, a float result is delivered. For example, `pow(-9, 2.0)` returns `81.0`.

Para operandos **`int`** como *base* y *exp*, si *mod* está presente, *mod* debe ser también de tipo entero y distinto de cero. Si *mod* está presente y *exp* es negativo, *base* debe ser un número primo relativo a *mod*. En ese caso, se retorna `pow(inv_base, -exp, mod)`, dónde *inv_base* es la inversa al módulo *mod* de *base*.

Aquí tienes un ejemplo de cómo calcular la inversa de `38` módulo `97`:

```
>>> pow(38, -1, mod=97)
23
>>> 23 * 38 % 97 == 1
True
```

Distinto en la versión 3.8: Para operandos **`int`**, la forma de `pow` con tres argumentos acepta ahora que el segundo argumento sea negativo, lo que permite el cálculo de inversos modulares.

Distinto en la versión 3.8: Permite argumentos de palabra clave. Anteriormente, solo se soportaba el uso de argumentos posicionales.

print(**objects, sep=' ', end='\n', file=None, flush=False*)

Imprime *objects* al flujo de texto *file*, separándolos por *sep* y seguidos por *end*. *sep*, *end*, *file* y *flush*, si están presentes, deben ser dados como argumentos por palabra clave.

Todos los argumentos que no son por palabra clave se convierten a cadenas tal y como **`str()`** hace y se escriben al flujo, separados por *sep* y seguidos por *end*. Tanto *sep* como *end* deben ser cadenas; también pueden ser `None`, lo cual significa que se empleen los valores por defecto. Si no se indica *objects*, **`print()`** escribirá *end*.

El argumento *file* debe ser un objeto que implemente un método `write(string)`; si éste no está presente o es `None`, se usará **`sys.stdout`**. Dado que los argumentos mostrados son convertidos a cadenas de texto, **`print()`** no puede ser utilizada con objetos fichero en modo binario. Para esos, utiliza en cambio `file.write(…)`.

El almacenamiento en búfer de salida generalmente está determinado por *file*. Sin embargo, si *flush* es verdadero, el flujo se vacía forzosamente.

Distinto en la versión 3.3: Añadido el argumento por palabra clave *flush*.

class property(*fget=None*, *fset=None*, *fdel=None*, *doc=None*)

Retorna un atributo propiedad.

fget es una función para obtener el valor de un atributo. *fset* es una función para asignar el valor de un atributo. *fdel* es una función para eliminar el valor de un atributo. Y *doc* crea un *docstring* para el atributo.

Un caso de uso típico es la definición de un atributo gestionado `x`:

```
class C:
    def __init__(self):
        self._x = None

    def getx(self):
        return self._x

    def setx(self, value):
        self._x = value

    def delx(self):
        del self._x

    x = property(getx, setx, delx, "I'm the 'x' pro
```

Si *c* es una instancia de *C*, `c.x` invocará el obtenedor (*getter*), `c.x = value` invocará el asignador (*setter*) y `del c.x` el suprimidor (*deleter*).

Si está indicada, *doc* será la docstring del atributo propiedad. En caso contrario, la propiedad copiará la *dosctring* de *fget* si ésta existe. Esto permite crear propiedades de sólo lectura de forma fácil empleando **`property()`** como decorator:

```
class Parrot:
    def __init__(self):
        self._voltage = 100000

    @property
    def voltage(self):
        """Get the current voltage."""
        return self._voltage
```

The `@property` decorator turns the **`voltage()`** method into a «getter» for a read-only attribute with the same name, and it sets the docstring for *voltage* to «Get the current voltage.»

@getter

@setter

@deleter

A property object has `getter`, `setter`, and `deleter` methods usable as decorators that create a copy of the property with the corresponding accessor function set to the decorated function. This is best explained with an example:

```
class C:
    def __init__(self):
        self._x = None

    @property
    def x(self):
        """I'm the 'x' property."""
        return self._x
    @x.setter
    def x(self, value):
        self._x = value

    @x.deleter
    def x(self):
        del self._x
```

Este código equivale exactamente al primer ejemplo. Asegúrese de otorgarle a las funciones adicionales el mismo nombre que la propiedad original (`x` en este caso.)

El objeto propiedad retornado tiene también los atributos `fget`, `fset`, y `fdel` correspondientes a los argumentos del constructor.

Distinto en la versión 3.5: Las *docstrings* de los objetos propiedad son escribibles.

__name__

Attribute holding the name of the property. The name of the property can be changed at runtime.

Added in version 3.13.

class range(*stop*, */*)
class range(*start*, *stop*, *step*=*1*, */*)

`range`, más que una función, es en realidad un tipo de secuencia inmutable, tal y como está documentado en Rangos y Tipos secuencia — list, tuple, range.

repr(*object*, */*)

Return a string containing a printable representation of an object. For many types, this function makes an attempt to return a string that would yield an object with the same value when passed to **`eval()`**; otherwise, the representation is a string enclosed in angle brackets that contains the name of the type of the object together with additional information often including the name and address of the object. A class can control what this function returns for its instances by defining a **`__repr__()`** method. If **`sys.displayhook()`** is not accessible, this function will raise **`RuntimeError`**.

This class has a custom representation that can be evaluated:

```
class Person:
   def __init__(self, name, age):
      self.name = name
      self.age = age

   def __repr__(self):
      return f"Person('{self.name}', {self.age})"
```

reversed(*object*, */*)

Return a reverse iterator. The argument must be an object which has a **`__reversed__()`** method or supports the sequence protocol (the **`__len__()`** method and the **`__getitem__()`** method with integer arguments starting at `0`).

round(*number*, *ndigits=None*)

Retorna *number* redondeado a *ndigits* de precisión después del punto decimal. Si *ndigits* es omitido o es `None`, retorna el entero más cercano a su entrada.

Para los tipos integrados que admiten **`round()`**, los valores se redondean al múltiplo más cercano de 10 a la potencia menos *ndigits*; si dos múltiplos están igualmente cerca, el redondeo se realiza hacia la opción par (así, por ejemplo, tanto `round(0.5)` como `round(-0.5)` son `0`, y `round(1.5)` es `2`). Cualquier valor entero es válido para *ndigits* (positivo, cero o negativo). El valor retornado es un entero si se omite *ndigits* o `None`. De lo contrario, el valor retornado tiene el mismo tipo que *number*.

Para un objeto `number` general de Python, `round` delega a `number.__round__`.

Nota

El comportamiento de `round()` para flotantes puede ser sorprendente: por ejemplo, `round(2.675, 2)` da `2.67` en vez de los `2.68` esperados. Esto no es un error: es el resultado del hecho de que la mayoría de fracciones decimales no se puede representar de forma exacta como flotantes. Véase Floating-Point Arithmetic: Issues and Limitations para más información.

class set(*iterable*=*()*, */*)

Retorna un nuevo objeto `set`, opcionalmente con elementos tomados de *iterable*. `set` es una clase integrada (*built-in*). Véase `set` y Conjuntos — set, frozenset para documentación sobre esta clase.

Para otros contenedores ver las clases integradas (*built-in*) `frozenset`, `list`, `tuple`, y `dict`, así como el módulo `collections`.

setattr(*object*, *name*, *value*, */*)

Es la función complementaria a `getattr()`. Los argumentos son un objeto, una cadena de caracteres, y un valor arbitrario. La cadena de caracteres puede nombrar un atributo existente o uno nuevo. La función asigna el valor al atributo si el objeto lo permite. Por ejemplo, `setattr(x, 'foobar', 123)` es equivalente a `x.foobar = 123`.

name no necesita ser un identificador de Python como se define en Names (identifiers and keywords) a menos que el objeto decida imponerlo, por ejemplo, en un `__getattribute__()` personalizado o a través de `__slots__`. Un atributo cuyo nombre no es un identificador no será accesible usando la notación de puntos, pero sí a través de `getattr()` etc..

Nota

Dado que private name mangling ocurre en el momento de la compilación, uno debe enredar manualmente el nombre de un atributo privado (atributos con dos guiones bajos principales) para recuperarlo con **`getattr()`**.

class slice(*stop*, */*)
class slice(*start*, *stop*, *step*=*None*, */*)

Return a slice object representing the set of indices specified by `range(start, stop, step)`. The *start* and *step* arguments default to `None`.

Slice objects are also generated when slicing syntax is used. For example: `a[start:stop:step]` or `a[start:stop, i]`.

See **`itertools.islice()`** for an alternate version that returns an iterator.

start
stop
step

These read-only attributes are set to the argument values (or their default). They have no other explicit functionality; however, they are used by NumPy and other third-party packages.

Distinto en la versión 3.12: Ahora los objetos slice son hashable (siempre que **`start`**, **`stop`** y **`step`** sean *hashable*).

sorted(*iterable*, */*, ***, *key*=*None*, *reverse*=*False*)

Retorna una nueva lista ordenada a partir de los elementos en *iterable*.

Tiene dos argumentos opcionales que deben ser especificados como argumentos de palabra clave.

key especifica una función de un argumento que es empleada para extraer una clave de comparación de cada elemento en *iterable* (por ejemplo, `key=str.lower`). El valor por defecto es `None` (compara los elementos directamente).

reverse es un valor boleado. Si está puesto a `True`, entonces la lista de elementos se ordena como si cada comparación fuera reversa.

Puedes usar **`functools.cmp_to_key()`** para convertir las funciones *cmp* a la antigua usanza en funciones *key*.

La función built-in **`sorted()`** está garantizada en cuanto a su estabilidad. Un ordenamiento es estable si garantiza que no cambia el orden relativo de elementos que resultan igual en la comparación — esto es de gran ayuda para ordenar en múltiples pases (por ejemplo, ordenar por departamento, después por el escalafón de salario).

The sort algorithm uses only `<` comparisons between items. While defining an **`__lt__()`** method will suffice for sorting, **PEP 8** [https://peps.python.org/pep-0008/] recommends that all six rich comparisons be implemented. This will help avoid bugs when using the same data with other ordering tools such as **`max()`** that rely on a different underlying method. Implementing all six comparisons also helps avoid confusion for mixed type comparisons which can call the reflected **`__gt__()`** method.

Para ejemplos de ordenamiento y para un breve tutorial sobre ello, ver Sorting Techniques.

@staticmethod

Transforma un método en un método estático.

Un método estático no recibe un primer argumento implícito. Para declarar un método estático, utiliza esta expresión:

```
class C:
    @staticmethod
    def f(arg1, arg2, argN): ...
```

La forma `@staticmethod` es una función decorator — ver

Definiciones de funciones para más detalles.

A static method can be called either on the class (such as `C.f()`) or on an instance (such as `C().f()`). Moreover, the static method descriptor is also callable, so it can be used in the class definition (such as `f()`).

Los métodos estáticos en Python son similares a los que se encuentran en Java o C++. Ver también **`classmethod()`** para una variante que es útil para crear constructores de clase alternativos.

Como todos los decoradores, es también posible llamar a `staticmethod` como una función normal y hacer algo con su resultado. Esto es necesario a veces cuando necesitas una referencia a una función desde el cuerpo de una clase y quieres evitar la transformación automática a un método de la instancia. Para dichos casos, emplea esta expresión:

```
def regular_function():
    ...

class C:
    method = staticmethod(regular_function)
```

Para más información sobre métodos estáticos, ver Jerarquía de tipos estándar.

Distinto en la versión 3.10: Static methods now inherit the method attributes (**`__module__`**, **`__name__`**, **`__qualname__`**, **`__doc__`** and **`__annotations__`**), have a new `__wrapped__` attribute, and are now callable as regular functions.

class str(***, *encoding='utf-8'*, *errors='strict'*)
class str(*object*)
class str(*object*, *encoding*, *errors='strict'*)
class str(*object*, ***, *errors*)

Retorna una versión **`str`** del *object*. Ver **`str()`** para más detalles.

`str` es la class cadena built-in . Para información general sobre strings, ver Cadenas de caracteres — str.

sum(*iterable*, /, *start=0*)

Suma *start* y los elementos de un *iterable* de izquierda a derecha y Retorna el total. Los elementos del *iterable* son normalmente números, y el valor *start* no puede ser una cadena.

For some use cases, there are good alternatives to **`sum()`**. The preferred, fast way to concatenate a sequence of strings is by calling `''.join(sequence)`. To add floating-point values with extended precision, see **`math.fsum()`**. To concatenate a series of iterables, consider using **`itertools.chain()`**.

Distinto en la versión 3.8: El parámetro *start* puede ser especificado como un argumento de palabra clave.

Distinto en la versión 3.12: Summation of floats switched to an algorithm that gives higher accuracy and better commutativity on most builds.

Distinto en la versión 3.14: Added specialization for summation of complexes, using same algorithm as for summation of floats.

class super
class super(*type*, *object_or_type=None*, /)

Retorna un objeto proxy que delega las llamadas de métodos a clases padre o hermanas de *type*. Esto es útil para acceder métodos heredados que han sido invalidados en una clase.

object-or-type determina el method resolution order a ser buscado. La búsqueda empieza desde la clase justo después de *type*.

For example, if **`__mro__`** of *object_or_type* is `D -> B -> C`

`-> A -> object` and the value of *type* is `B`, then **`super()`** searches `C -> A -> object`.

The **`__mro__`** attribute of the class corresponding to *object_or_type* lists the method resolution search order used by both **`getattr()`** and **`super()`**. The attribute is dynamic and can change whenever the inheritance hierarchy is updated.

Si se omite el segundo argumento, el objeto super retornado no está vinculado. Si el segundo argumento es un objeto, `isinstance(obj, type)` debe ser verdadero. Si el segundo argumento es un tipo, `issubclass(type2, type)` debe ser verdadero (esto es útil para classmethods).

When called directly within an ordinary method of a class, both arguments may be omitted («zero-argument **`super()`**»). In this case, *type* will be the enclosing class, and *obj* will be the first argument of the immediately enclosing function (typically `self`). (This means that zero-argument **`super()`** will not work as expected within nested functions, including generator expressions, which implicitly create nested functions.)

Hay dos casos de uso típicos para *super*. En una jerarquía de clases con herencia única, *super* puede ser utilizado para referirse a las clases padre sin llamarlas explícitamente, haciendo el código más sencillo de mantener. Este uso es muy similar al de *super* en otros lenguajes de programación.

El segundo caso de uso es el soporte de herencia múltiple cooperativa en un entorno de ejecución dinámica. Este caso de uso es único en Python y no se encuentra en lenguajes estáticos compilados o que solo soportan herencia única. Esto hace posible implementar los «diagramas de diamantes», donde múltiples clases base implementan el mismo método. Las buenas prácticas de diseño dictan que estas implementaciones tengan la misma signatura de llamada en cada caso (porque el orden de llamadas se determina en tiempo de ejecución, porque ese orden se adapta a los cambios en la jerarquía de clase y porque ese orden puede incluir clases hermanas que son desconocidas antes de la ejecución).

Para ambos casos, la llamada típica de una superclase se parece a:

```
class C(B):
    def method(self, arg):
        super().method(arg)    # This does the same
                               # super(C, self).met
```

Además de para los métodos de búsqueda, **`super()`** también funciona para búsquedas de atributos. Un caso de uso posible para esto es llamar a descriptores en una clase padre o hermana.

Note that **`super()`** is implemented as part of the binding process for explicit dotted attribute lookups such as `super().__getitem__(name)`. It does so by implementing its own **`__getattribute__()`** method for searching classes in a predictable order that supports cooperative multiple inheritance. Accordingly, **`super()`** is undefined for implicit lookups using statements or operators such as `super()[name]`.

Nótese también, que aparte de en su forma sin argumentos, **`super()`** no está limitada a su uso dentro de métodos. La forma con dos argumentos especifica de forma exacta los argumentos y hace las referencias apropiadas. La forma sin argumentos solo funciona dentro de una definición de clase, ya que el compilador completa los detalles necesarios para obtener correctamente la clase que está siendo definida, así como accediendo a la instancia actual para métodos ordinarios.

Para sugerencias prácticas sobre como diseñas clases cooperativas usando **`super()`**, véase guía de uso de super() [https://rhettinger.wordpress.com/2011/05/26/super-considered-super/].

Distinto en la versión 3.14: **`super`** objects are now **`pickleable`** and **`copyable`**.

class tuple(*iterable*=(), /)

Más que una función, `tuple` es en realidad un tipo de secuencia inmutable, tal y como está documentado en Tuplas y Tipos secuencia — list, tuple, range.

class type(*object*, /)

class type(*name*, *bases*, *dict*, /, ***kwargs*)

With one argument, return the type of an *object*. The return value is a type object and generally the same object as returned by `object.__class__`.

La función integrada `isinstance()` es la recomendada para testear el tipo de un objeto, ya que tiene en cuenta las subclases.

With three arguments, return a new type object. This is essentially a dynamic form of the `class` statement. The *name* string is the class name and becomes the `__name__` attribute. The *bases* tuple contains the base classes and becomes the `__bases__` attribute; if empty, `object`, the ultimate base of all classes, is added. The *dict* dictionary contains attribute and method definitions for the class body; it may be copied or wrapped before becoming the `__dict__` attribute. The following two statements create identical **`type`** objects:

```
>>> class X:
...     a = 1
...
>>> X = type('X', (), dict(a=1))
```

See also:

- Documentation on attributes and methods on classes.
- Objetos Tipo

Los argumentos de palabras clave proporcionados al formulario de tres argumentos se pasan a la maquinaria de metaclase apropiada (generalmente `__init_subclass__()`) de la misma manera que lo harían las palabras clave en una definición de clase (además de *metaclase*).

Ver también Personalización de creación de clases.

Distinto en la versión 3.6: Subclasses of **`type`** which don't override `type.__new__` may no longer use the one-argument form to get the type of an object.

vars()
vars(*object*, */*)

Return the **`__dict__`** attribute for a module, class, instance, or any other object with a **`__dict__`** attribute.

Objects such as modules and instances have an updateable **`__dict__`** attribute; however, other objects may have write restrictions on their **`__dict__`** attributes (for example, classes use a **`types.MappingProxyType`** to prevent direct dictionary updates).

Without an argument, **`vars()`** acts like **`locals()`**.

Una excepción **`TypeError`** se genera si se especifica un objeto pero no tiene un atributo **`__dict__`** (por ejemplo, si su clase define el atributo **`__slots__`**).

Distinto en la versión 3.13: The result of calling this function without an argument has been updated as described for the **`locals()`** builtin.

zip(**iterables*, *strict=False*)

Iterar sobre varios iterables en paralelo, generando tuplas con un item para cada una.

Ejemplo:

```
>>> for item in zip([1, 2, 3], ['sugar', 'spice', '
...     print(item)
...
(1, 'sugar')
```

```
(2, 'spice')
(3, 'everything nice')
```

Más formalmente: `zip()` retorna un iterador de tuplas, donde la tupla *i*-ésima contiene el elemento *i*-ésimo de cada uno de los argumentos iterables.

Otra forma de pensar en `zip()` es que convierte las filas en columnas y las columnas en filas. Esto es similar a transponer una matriz [https://en.wikipedia.org/wiki/Transpose].

`zip()` es perezoso: los elementos no se procesarán hasta que el iterable sea iterado, por ejemplo, mediante un bucle `for` o envolviendo en un `list`.

Una cosa a considerar es que los iterables pasados a `zip()` podrían tener diferentes longitudes; a veces por diseño, y a veces por un error en el código que preparó estos iterables. Python ofrece tres enfoques diferentes para tratar este problema:

- De forma predeterminada, `zip()` se detiene cuando se agota el iterable más corto. Ignorará los elementos restantes en las iteraciones más largas, cortando el resultado a la longitud del iterable más corto:

  ```
  >>> list(zip(range(3), ['fee', 'fi', 'fo', 'fum
  [(0, 'fee'), (1, 'fi'), (2, 'fo')]
  ```

- `zip()` se usa a menudo en los casos en que se supone que los iterables son de igual longitud. En tales casos, se recomienda usar la opción `strict=True`. Su salida es la misma que la normal `zip()`:

  ```
  >>> list(zip(('a', 'b', 'c'), (1, 2, 3), strict
  [('a', 1), ('b', 2), ('c', 3)]
  ```

  A diferencia del comportamiento predeterminado, lanza una excepción `ValueError` si un iterable se agota antes que los demás:

```
>>> for item in zip(range(3), ['fee', 'fi', 'fo
...     print(item)
...
(0, 'fee')
(1, 'fi')
(2, 'fo')
Traceback (most recent call last):
  ...
ValueError: zip() argument 2 is longer than arg
```

Sin el argumento `strict=True`, cualquier error que resulte en iterables de diferentes longitudes será silenciado, posiblemente manifestándose como un error difícil de encontrar en otra parte del programa.

- Las iteraciones más cortas se pueden acolchar con un valor constante para que todas las iteraciones tengan la misma longitud. Esto lo hace **`itertools.zip_longest()`**.

Casos extremos: con un único argumento iterable, **`zip()`** retorna un iterador de 1-tuplas. Sin argumentos, retorna un iterador vacío.

Consejos y trucos:

- La evaluación de izquierda a derecha de los iterables está garantizada. Esto permite emplear una expresión idiomática para agregar una serie de datos en grupos de tamaño *n* usando `zip(*[iter(s)]*n, strict=True)`. Esto repite el *mismo* iterador `n` veces de forma que cada tupla de salida tiene el resultado de `n` llamadas al iterador. Esto tiene el efecto de dividir la entrada en trozos de longitud *n*.

- **`zip()`** en conjunción con el operador `*` puede usar para descomprimir (*unzip*) una lista:

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> list(zip(x, y))
```

```
[(1, 4), (2, 5), (3, 6)]
>>> x2, y2 = zip(*zip(x, y))
>>> x == list(x2) and y == list(y2)
True
```

Distinto en la versión 3.10: Añadido el argumento `strict`.

__import__(*name*, *globals=None*, *locals=None*, *fromlist=()*, *level=0*)

Nota

Ésta es una función avanzada que no se necesita en el uso cotidiano de programación en Python, a diferencia de **`importlib.import_module()`**.

Esta función es invocada por la declaración **`import`**. Puede ser reemplazada para cambiar la semántica de la declaración **`import`** (mediante la importación del módulo **`builtins`** y su asignación a `builtins.__import__`), pero esto está **fuertemente** no recomendado ya que es normalmente mucho más simple usar ganchos (*hooks*) de importación (ver **PEP 302** [https://peps.python.org/pep-0302/]) para obtener los mismos objetivos y sin causar problemas en código que asume que la implementación por defecto de importaciones está siendo utilizada. El uso directo de **`__import__()`** tampoco está recomendado y se prefiere **`importlib.import_module()`**.

La función importa el módulo *name*, usando potencialmente las *globals* y *locals* indicadas para determinar como interpretar el nombre en el contexto de un paquete. *fromlist* indica los nombres de objetos o submódulos que deberían ser importados del módulo indicado por *name*. La implementación estándar no utiliza su argumento *locals* para nada, y usa *globals* solo para determinar el contexto en un paquete de la declaración **`import`**.

level especifica si usar importaciones absolutas o relativas. `0` (por defecto) significa sólo realizar importaciones absolutas. Valores positivos de *level* indican el número de directorios progenitores relativos al del módulo para buscar llamando a `__import__()` (ver **PEP 328** [https://peps.python.org/pep-0328/] para los detalles).

Cuando la variable *name* tiene la forma `package.module`, normalmente el paquete del nivel más alto (el nombre hasta el primer punto) se retorna, *no* el modulo llamado por *name*. Sin embargo, cuando un argumento *fromlist* no vacío es indicado, el módulo llamado por *name* es retornado.

Por ejemplo, la declaración `import spam` resulta en un bytecode similar al siguiente código:

```
spam = __import__('spam', globals(), locals(), [],
```

La declaración `import spam.ham` resulta en esta llamada:

```
spam = __import__('spam.ham', globals(), locals(),
```

Nótese cómo `__import__()` retorna el módulo del nivel superior en este caso porque este es el objeto que está enlazado a un nombre por la declaración `import`.

Por otra parte, la declaración `from spam.ham import eggs, sausage as saus` resulta en

```
_temp = __import__('spam.ham', globals(), locals(),
eggs = _temp.eggs
saus = _temp.sausage
```

Aquí, el módulo `spam.ham` es retornado desde `__import__()`. Desde este objeto, los nombres a importar son obtenidos y asignados a sus nombres respectivos.

Si simplemente quieres importar un módulo (potencialmente dentro de un paquete) por nombre, usa `importlib.import_module()`.

Distinto en la versión 3.3: Valores negativos para *level* ya no están soportados (lo que también cambia el valor por defecto a 0).

Distinto en la versión 3.9: Cuando se utilizan las opciones de línea de comando `-E` o `-I`, la variable de entorno `PYTHONCASEOK` ahora se ignora.

Notas al pie

Ten en cuenta que el *parser* sólo acepta la convención de final de línea de tipo Unix. Si estás leyendo el código de un fichero, asegúrate de usar la el modo de conversión de nueva línea para convertir las líneas de tipo Windows o Mac.

Constantes incorporadas

Un pequeño número de constantes viven en el espacio de nombres incorporado. Ellas son:

False

  El valor falso del tipo **`bool`**. Las asignaciones a `False` son ilegales y generan un **`SyntaxError`**.

True

  El valor verdadero del tipo **`bool`**. Las asignaciones a `True` son ilegales y generan un **`SyntaxError`**.

None

  An object frequently used to represent the absence of a value, as when default arguments are not passed to a function. Assignments to `None` are illegal and raise a **`SyntaxError`**. `None` is the sole instance of the **`NoneType`** type.

NotImplemented

  A special value which should be returned by the binary special methods (e.g. **`__eq__()`**, **`__lt__()`**, **`__add__()`**, **`__rsub__()`**, etc.) to indicate that the operation is not implemented with respect to the other type; may be returned by the in-place binary special methods (e.g. **`__imul__()`**, **`__iand__()`**, etc.) for the same purpose. It should not be evaluated in a boolean context. **`NotImplemented`** is the sole instance of the **`types.NotImplementedType`** type.

  Nota

  When a binary (or in-place) method returns **`NotImplemented`** the interpreter will try the reflected operation on the other type (or some other fallback, depending on the operator). If all attempts return

`NotImplemented`, the interpreter will raise an appropriate exception. Incorrectly returning **`NotImplemented`** will result in a misleading error message or the **`NotImplemented`** value being returned to Python code.

Consulte Implementar operaciones aritméticas para ver ejemplos.

Prudencia

`NotImplemented` and **`NotImplementedError`** are not interchangeable. This constant should only be used as described above; see **`NotImplementedError`** for details on correct usage of the exception.

Distinto en la versión 3.9: Evaluating **`NotImplemented`** in a boolean context was deprecated.

Distinto en la versión 3.14: Evaluating **`NotImplemented`** in a boolean context now raises a **`TypeError`**. It previously evaluated to **`True`** and emitted a **`DeprecationWarning`** since Python 3.9.

Ellipsis

The same as the ellipsis literal «`...`», an object frequently used to indicate that something is omitted. Assignment to `Ellipsis` is possible, but assignment to `...` raises a **`SyntaxError`**. `Ellipsis` is the sole instance of the **`types.EllipsisType`** type.

__debug__

Esta constante es verdadera si Python no se inició con una opción **`-O`**. Vea también la instrucción **`assert`**.

Nota

Los nombres: **`None`**, **`False`**, **`True`** y **`__debug__`** no se pueden reasignar (asignaciones a ellos, incluso como un nombre de atributo, lanza `SyntaxError`), por lo que pueden considerarse constantes «verdaderas».

Constantes agregadas por el módulo `site`

El módulo `site` (que se importa automáticamente durante el inicio, excepto si se proporciona la opción `-S` en la línea de comandos) agrega varias constantes al espacio de nombres integrado. Son útiles para el intérprete interactivo y no deben usarse en programas.

quit(*code*=*None*)
exit(*code*=*None*)

Objects that when printed, print a message like «Use quit() or Ctrl-D (i.e. EOF) to exit», and when accessed directly in the interactive interpreter or called as functions, raise `SystemExit` with the specified exit code.

help

Object that when printed, prints the message «Type help() for interactive help, or help(object) for help about object.», and when accessed directly in the interactive interpreter, invokes the built-in help system (see `help()`).

copyright
credits

Objetos que al ser impresos o llamados imprimen el texto de derechos de autor o créditos, respectivamente.

license

Objeto que cuando se imprime, muestra el mensaje «Escriba licencia () para ver el texto completo de la licencia», y cuando se le llama, muestra el texto completo de la licencia en forma de buscapersonas (una pantalla a la vez).

[1] [2] [3] [4] (1,2,3,4) [5] (1,2)

Tipos integrados

Esta sección describe los tipos de datos estándar que vienen incluidos en el intérprete.

Los principales tipos de datos son: numéricos, secuencias, mapas, clases, instancias y excepciones.

Algunas clases de tipo colección son mutables. Los métodos que añaden, retiran u ordenan sus miembros en su lugar, y a no ser que retornen un elemento concreto, nunca retornan la propia instancia contenedora, sino `None`.

Algunas operaciones son soportadas por varios tipos de objetos diferentes; por ejemplo, prácticamente todos los objetos pueden ser comparados por igualdad, evaluados para ser considerados como valores booleanos, o representarse en forma de cadena de caracteres (ya sea con la función `repr()` o con la función ligeramente diferente `str()`). Esta última es la usada implícitamente cuando un objeto se escribe con la función `print()`.

Evaluar como valor verdadero/falso

Cualquier objeto puede ser evaluado como si fuera un valor verdadero o falso, para ser usado directamente en sentencias `if` o `while`, o como un operador en una operación booleana como las que veremos más adelante.

By default, an object is considered true unless its class defines either a `__bool__()` method that returns `False` or a `__len__()` method that returns zero, when called with the object. [1] If one of the methods raises an exception when called, the exception is propagated and the object does not have a truth value (for example, `NotImplemented`). Here are most of the built-in objects considered false:

- constantes definidas para tener valor falso: `None` y `False`.
- cero en cualquiera de los diferentes tipos numéricos: `0`, `0.0`, `0j`, `Decimal(0)`, `Fraction(0, 1)`
- cualquier colección o secuencia vacía: `''`, `()`, `[]`, `{}`, `set()`, `range(0)`

Las operaciones y funciones integradas que tienen como resultado un booleano siempre retornan `0` o `False` para un valor falso, y `1` o `True` para un valor verdadero, a no ser que se indique otra cosa. (Hay una excepción importante: Las operaciones booleanas `or` y `and` siempre retornan uno de los dos operadores.)

Operaciones booleanas — `and`, `or`, `not`

Estas son las operaciones booleanas, ordenadas de menor a mayor prioridad:

Operación	Resultado	Notas
`x or y`	si *x* es verdad, entonces *x*, si no, *y*	(1)
`x and y`	si *x* es falso, entonces *x*, si no, *y*	(2)
`not x`	si *x* es falso, entonces `True`, si no, `False`	(3)

Notas:

1. Este operador usa lógica cortocircuitada, por lo que solo evalúa el segundo argumento si el primero es falso.
2. Este operador usa lógica cortocircuitada, por lo que solo evalúa el segundo argumento si el primero es verdadero.
3. El operador `not` tiene menos prioridad que los operadores no booleanos, así que `not a == b` se interpreta como `not (a == b)`, y `a == not b` es un error sintáctico.

Comparaciones

Existen ocho operadores de comparación en Python. Todos comparten el mismo nivel de prioridad (que es mayor que el nivel de las operaciones booleanas). Las comparaciones pueden encadenarse de cualquier manera; por ejemplo, `x < y <= z` equivale a `x < y and y <= z`, excepto porque *y* solo se evalúa una vez (no obstante, en ambos casos *z* no se evalúa si no es verdad que `x < y`).

Esta tabla resume las operaciones de comparación:

Operación	**Significado**
`<`	estrictamente menor que
`<=`	menor o igual que
`>`	estrictamente mayor que
`>=`	mayor o igual que
`==`	igual que
`!=`	diferente que
`is`	igualdad a nivel de identidad (Son el mismo objeto)
`is not`	desigualdad a nivel de identidad (no son el mismo objeto)

Unless stated otherwise, objects of different types never compare equal. The `==` operator is always defined but for some object types (for example, class objects) is equivalent to **`is`**. The `<`, `<=`, `>` and `>=` operators are only defined where they make sense; for example, they raise a **`TypeError`** exception when one of the arguments is a complex number.

Las instancias de una clase que no son idénticas normalmente se comparan como diferentes, a no ser que la clase defina el método **`__eq__()`**.

Las instancias de una clase no pueden ordenarse con respecto a otras instancias de la misma clase, ni con otro tipo de objetos, a no ser que la clase defina suficientes métodos **`__lt__()`**, **`__le__()`**, **`__gt__()`** y **`__ge__()`** (en general, **`__lt__()`** y **`__eq__()`** son suficientes, si solo necesitas los significados convencionales de los operadores de comparación).

El comportamiento de los operadores **`is`** e **`is not`** no se puede personalizar; además, se pueden aplicar a dos objetos cualquiera y nunca lanzar una excepción.

Dos operaciones más con la misma prioridad sintáctica, **`in`** y **`not in`**, son compatibles con tipos que son iterable o implementan el método **`__contains__()`**.

Tipos numéricos — `int`, `float`, `complex`

Existen tres tipos numéricos distintos: *números enteros*, *números de punto flotante* y *números complejos*. Además, los booleanos son un subtipo de los números enteros. Los números enteros tienen una precisión ilimitada. Los números de punto flotante se implementan normalmente utilizando double en C; la información sobre la precisión y la representación interna de los números de punto flotante para la máquina en la que se ejecuta el programa está disponible en `sys.float_info`. Los números complejos tienen una parte real y una imaginaria, que son cada una un número de punto flotante. Para extraer estas partes de un número complejo *z*, utilice `z.real` y `z.imag`. (La biblioteca estándar incluye los tipos numéricos adicionales `fractions.Fraction`, para números racionales, y `decimal.Decimal`, para números de punto flotante con precisión definible por el usuario).

Los números se crean mediante literales numéricos o como resultado de funciones y operadores integrados. Los literales enteros sin adornos (incluidos los números hexadecimales, octales y binarios) generan números enteros. Los literales numéricos que contienen un punto decimal o un signo de exponente generan números de punto flotante. Si se añade `'j'` o `'J'` a un literal numérico, se obtiene un número imaginario (un número complejo con una parte real cero) que se puede sumar a un entero o un flotante para obtener un número complejo con partes reales e imaginarias.

Las funciones constructoras `int()`, `float()` y `complex()` se pueden usar para generar números de cada tipo determinado.

Python fully supports mixed arithmetic: when a binary arithmetic operator has operands of different built-in numeric types, the operand with the «narrower» type is widened to that of the other:

- If both arguments are complex numbers, no conversion is performed;
- if either argument is a complex or a floating-point number, the other is converted to a floating-point number;

- otherwise, both must be integers and no conversion is necessary.

Arithmetic with complex and real operands is defined by the usual mathematical formula, for example:

```
x + complex(u, v) = complex(x + u, v)
x * complex(u, v) = complex(x * u, x * v)
```

A comparison between numbers of different types behaves as though the exact values of those numbers were being compared. [2]

Todos los tipos numéricos (menos los complejos) soportan las siguientes operaciones (para las prioridades de las operaciones, véase Prioridad de operador):

Operación	**Resultado**	**Notas**	**Documentación completa**
`x + y`	suma de *x* e *y*		
`x - y`	resta de *x* e *y*		
`x * y`	multiplicación de *x* por *y*		
`x / y`	división de *x* entre *y*		
`x // y`	división entera a la baja de *x* entre *y*	(1)(2)	
`x % y`	resto o residuo de `x / y`	(2)	
`-x`	valor de *x*, negado		
`+x`	valor de *x*, sin cambiar		
`abs(x)`	valor absoluto de la magnitud de *x*		**`abs()`**
`int(x)`	valor de *x* convertido a entero	(3)(6)	**`int()`**
`float(x)`	valor de *x* convertido a número de coma flotante	(4)(6)	**`float()`**
`complex(re, im)`	un número complejo, con parte real *re* y parte imaginaria *im*. El valor de *im* por defecto vale cero.	(6)	**`complex()`**
`c.conjugate()`	conjugado del número complejo *c*		
`divmod(x, y)`	el par de valores `(x // y, x % y)`	(2)	**`divmod()`**
`pow(x, y)`	*x* elevado a *y*	(5)	**`pow()`**
`x ** y`	*x* elevado a *y*	(5)	

Notas:

1. También conocida como división entera a la baja. Para operandos de tipo **`int`**, el resultado será de tipo **`int`**. Para operandos de tipo **`float`**, el resultado será de tipo **`float`**. En general, el resultado es un número entero en el sentido

matemático, pero no necesariamente de tipo entero **`int`**. El resultado se redondea de forma automática hacia menos infinito: `1//2` es `0`, `(-1)//2` es `-1`, `1//(-2)` es `-1` y `(-1)//(-2)` es `0`.

2. No es apropiada para números complejos. Es preferible convertir a valores en coma flotante usando la función **`abs()`** si fuera apropiado.

3. La conversión de **`float`** a **`int`** trunca y descarta la parte fraccionaria. Consulte las funciones **`math.floor()`** y **`math.ceil()`** para conocer conversiones alternativas.

4. float también acepta las cadenas de caracteres «*nan*» e «*inf*», con un prefijo opcional «+» o «-», para los valores *Not a Number* (*NaN*) e infinito positivo o negativo.

5. Python define `pow(0, 0)` y `0 ** 0` para que valgan `1`, como es práctica habitual en los lenguajes de programación.

6. Los literales numéricos aceptables incluyen los dígitos desde el `0` hasta el `9`, así como cualquier carácter Unicode equivalente (puntos de código con la propiedad `Nd`).

   Consulte the Unicode Standard [https://unicode.org/Public/UNIDATA/extracted/DerivedNumericType.txt] para obtener una lista completa de puntos de código con la propiedad `Nd`.

Todas las clases derivadas de **`numbers.Real`** (**`int`** y **`float`**) también soportan las siguientes operaciones:

Operación	**Resultado**
`math.trunc(x)`	x truncado a **`Integral`**
`round(x[, n])`	El valor de x redondeado a n dígitos, redondeando la mitad al número par más cercano (redondeo del banquero). Si no se especifica valor para n, se asume 0.
`math.floor(x)`	el mayor número **`Integral`** que sea $<= x$
`math.ceil(x)`	el menor número **`Integral`** que sea $>= x$

Para más operaciones numéricas consulta los módulos **`math`** y **`cmath`**.

Operaciones de bits en números enteros

Las operaciones a nivel de bit solo tienen sentido con números enteros. El resultado de una de estas operaciones se calcula como si se hubiera realizado en una representación en complemento a dos que tuviera un número infinito de bits de signo.

La prioridad de todas las operaciones de bits son menores que las operaciones numéricas, pero mayores que las comparaciones; la operación unaria `~` tiene la misma prioridad que las otras operaciones unarias numéricas (`+` y `-`).

Esta tabla lista las operaciones de bits, ordenadas de menor a mayor prioridad:

Operación	**Resultado**	**Notas**
`x | y`	la operación *or* entre *x* e *y*	(4)
`x ^ y`	la operación *exclusive or* entre *x* e *y*	(4)
`x & y`	la operación *and* entre *x* e *y*	(4)
`x << n`	valor de *x* desplazado a la izquierda *n* bits	(1)(2)
`x >> n`	valor de *x* desplazado a la derecha *n* bits	(1)(3)
`~x`	invierte los bits de *x*	

Notas:

1. Los desplazamientos negativos son ilegales y lanzarán una excepción de tipo **`ValueError`**.
2. Un desplazamiento de *n* bits a la izquierda es equivalente a multiplicar por `pow(2, n)`.
3. Un desplazamiento de *n* bits a la derecha es equivalente a efectuar la división entera a la baja entre `pow(2, n)`.
4. Realizar estos cálculos con al menos un bit extra de signo en una representación finita de un número en complemento a dos (un ancho de bits de trabajo de `1 + max(x.bit_length(), y.bit_length())` o más) es suficiente para obtener el mismo resultado que si se hubiera realizado con un número infinito de bits de signo.

Métodos adicionales de los enteros

El tipo int implementa la clase base abstracta `numbers.Integral`. Además, proporciona los siguientes métodos:

int.bit_length()

Retorna el número de bits necesarios para representar un número entero, excluyendo el bit de signo y los ceros a la izquierda:

```
>>> n = -37
>>> bin(n)
'-0b100101'
>>> n.bit_length()
6
```

De forma más precisa, si `x` es distinto de cero, entonces `x.bit_length()` es el único número entero positivo `k` tal que `2**(k-1) <= abs(x) < 2**k`. De igual manera, cuando `abs(x)` es lo suficientemente pequeño para tener un logaritmo redondeado correctamente, entonces `k = 1 + int(log(abs(x), 2))`. Si `x` es cero, entonces `x.bit_length()` retorna `0`.

Equivale a:

```
def bit_length(self):
    s = bin(self)       # binary representation:  b
    s = s.lstrip('-0b') # remove leading zeros and 
    return len(s)       # len('100101') --> 6
```

Added in version 3.1.

int.bit_count()

Retorna el número de unos en la representación binaria del valor absoluto del entero. Esto también se conoce como el recuento de la población. Ejemplo:

```
>>> n = 19
>>> bin(n)
'0b10011'
>>> n.bit_count()
3
```

```
>>> (-n).bit_count()
3
```

Equivale a:

```
def bit_count(self):
    return bin(self).count("1")
```

Added in version 3.10.

int.to_bytes(*length=1*, *byteorder='big'*, ***, *signed=False*)

Retorna un arreglo de bytes que representan el número entero.

```
>>> (1024).to_bytes(2, byteorder='big')
b'\x04\x00'
>>> (1024).to_bytes(10, byteorder='big')
b'\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00'
>>> (-1024).to_bytes(10, byteorder='big', signed=Tr
b'\xff\xff\xff\xff\xff\xff\xff\xff\xfc\x00'
>>> x = 1000
>>> x.to_bytes((x.bit_length() + 7) // 8, byteorder
b'\xe8\x03'
```

El número entero se representa usando el número de bits indicados con *length* y el valor predeterminado es 1. Se lanzará la excepción `OverflowError` si no se puede representar el entero con ese número de bits.

El argumento *byteorder* determina el orden de representación del número entero y el valor predeterminado es `"big"`. Si *byteorder* es `"big"`, el byte más significativo ocupa la primera posición del arreglo del byte. Si *byteorder* es `"little"`, el byte más significativo estará en la última posición.

El parámetro *signed* determina si se usa el complemento a dos para representar los números enteros. Si *signed* es `False`, y se usa un valor entero negativo, se lanzará la excepción `OverflowError`. El valor por defecto para *signed* es `False`.

Los valores predeterminados se pueden utilizar para convertir cómodamente un entero en un objeto de un solo byte:

```
>>> (65).to_bytes()
b'A'
```

Sin embargo, al utilizar los argumentos predeterminados, no intente convertir un valor mayor que 255 o obtendrá un **`OverflowError`**.

Equivale a:

```
def to_bytes(n, length=1, byteorder='big', signed=F
    if byteorder == 'little':
        order = range(length)
    elif byteorder == 'big':
        order = reversed(range(length))
    else:
        raise ValueError("byteorder must be either 

    return bytes((n >> i*8) & 0xff for i in order)
```

Added in version 3.2.

Distinto en la versión 3.11: Se agregaron valores de argumentos predeterminados para `length` y `byteorder`.

classmethod int.from_bytes(*bytes*, *byteorder='big'*, ***, *signed=False*)

Retorna el número entero que se representa por el arreglo de bytes.

```
>>> int.from_bytes(b'\x00\x10', byteorder='big')
16
>>> int.from_bytes(b'\x00\x10', byteorder='little')
4096
>>> int.from_bytes(b'\xfc\x00', byteorder='big', si
-1024
>>> int.from_bytes(b'\xfc\x00', byteorder='big', si
64512
>>> int.from_bytes([255, 0, 0], byteorder='big')
```

```
16711680
```

El argumento *bytes* debe ser o bien un objeto tipo binario o un iterable que produzca bytes.

El argumento *byteorder* determina el orden de representación del número entero y el valor predeterminado es `"big"`. Si *byteorder* es `"big"`, el byte más significativo ocupa la primera posición en el arreglo del byte. Si *byteorder* es `"little"`, el byte más significativo estará en la última posición. Para solicitar el orden de bytes nativo del sistema host, usa **`sys.byteorder`** como valor de orden de bytes.

El argumento *signed* indica si se representará el número entero usando complemento a dos.

Equivale a:

```
def from_bytes(bytes, byteorder='big', signed=False
    if byteorder == 'little':
        little_ordered = list(bytes)
    elif byteorder == 'big':
        little_ordered = list(reversed(bytes))
    else:
        raise ValueError("byteorder must be either

    n = sum(b << i*8 for i, b in enumerate(little_o
    if signed and little_ordered and (little_ordere
        n -= 1 << 8*len(little_ordered)

    return n
```

Added in version 3.2.

Distinto en la versión 3.11: Se agregó valor de argumento predeterminado para `byteorder`.

int.as_integer_ratio()

Devuelve un par de números enteros cuyo cociente es igual al entero original y tiene un denominador positivo. El cociente entero de los números enteros (números naturales) es siempre el entero como numerador y `1` como denominador.

Added in version 3.8.

int.is_integer()

Devuelve `True`. Existe para compatibilidad de tipo pato con **`float.is_integer()`**.

Added in version 3.12.

Métodos adicionales de float

El tipo float implementa la clase base abstracta **`numbers.Real`**. Los números float tienen además los siguientes métodos.

classmethod float.from_number(*x*)

Class method to return a floating-point number constructed from a number *x*.

If the argument is an integer or a floating-point number, a floating-point number with the same value (within Python's floating-point precision) is returned. If the argument is outside the range of a Python float, an **`OverflowError`** will be raised.

For a general Python object `x`, `float.from_number(x)` delegates to `x.__float__()`. If **`__float__()`** is not defined then it falls back to **`__index__()`**.

Added in version 3.14.

float.as_integer_ratio()

Devuelve un par de números enteros cuyo cociente es exactamente igual al número flotante original. El cociente está en su mínima expresión y tiene un denominador positivo. Genera **`OverflowError`** en los números infinitos y **`ValueError`** en los números NaN.

float.is_integer()

Retorna `True` si el valor en coma flotante es finita con valor integral, y `False` en caso contrario:

```
>>> (-2.0).is_integer()
True
>>> (3.2).is_integer()
False
```

Hay dos métodos que convierten desde y hacia cadenas de caracteres en hexadecimal. Como los valores en coma flotante en Python se almacenan internamente en binario, las conversiones desde o hacia cadenas *decimales* pueden implicar un pequeño error de redondeo. Pero con cadenas de caracteres en hexadecimal, las cadenas se corresponden y permiten representar de forma exacta los números en coma flotante. Esto puede ser útil, ya sea a la hora de depurar errores, o en procesos numéricos.

float.hex()

Retorna la representación de un valor en coma flotante en forma de cadena de caracteres en hexadecimal. Para números finitos, la representación siempre empieza con el prefijo `0x`, y con una `p` justo antes del exponente.

classmethod float.fromhex(*s*)

Método de clase que retorna el valor en coma flotante que se representa por la cadena de caracteres en hexadecimal *s*. La cadena de caracteres *s* puede tener espacios en blanco al principio o al final.

Nótese que **`float.hex()`** es un método de instancia, mientras que **`float.fromhex()`** es un método de clase.

Una cadena de caracteres en hexadecimal sigue este formato:

```
[sign] ['0x'] integer ['.' fraction] ['p' exponent]
```

donde el componente opcional `sign` puede ser o bien `+` o `-`, las componentes `integer` y `fraction` son cadenas de caracteres que solo usan dígitos hexadecimales, y `exponent` es un número decimal, precedido con un signo opcional. No se distingue entre mayúsculas y minúsculas, y debe haber al menos un dígito hexadecimal tanto en la parte entera como en la fracción. Esta sintaxis es similar a la sintaxis especificada en la sección 6.4.4.2 del estándar C99, y es también la sintaxis usada en Java desde la versión 1.5. En particular, la salida de **`float.hex()`** se puede usar como una cadena de caracteres en hexadecimal en código C o Java, y las cadenas de caracteres hexadecimal producidas por el carácter de formato `%a` en C, o por el método Java, `Double.toHexString`, son aceptadas por **`float.fromhex()`**.

Nótese que el valor del exponente se expresa en decimal, no en hexadecimal, e indica la potencia de 2 por la que debemos multiplicar el coeficiente. Por ejemplo, la cadena de caracteres hexadecimal `0x3.a7p10` representa el número en coma flotante `(3 + 10./16 + 7./16**2) * 2.0**10`, o `3740.0`:

```
>>> float.fromhex('0x3.a7p10')
3740.0
```

Si aplicamos la operación inversa a `3740.0` retorna una cadena de caracteres hexadecimal diferente que, aun así, representa el mismo número:

```
>>> float.hex(3740.0)
'0x1.d380000000000p+11'
```

Additional Methods on Complex

The **`complex`** type implements the **`numbers.Complex`** abstract base class. **`complex`** also has the following additional methods.

classmethod complex.from_number(*x*)

Class method to convert a number to a complex number.

For a general Python object `x`, `complex.from_number(x)` delegates to `x.__complex__()`. If **`__complex__()`** is not defined then it falls back to **`__float__()`**. If **`__float__()`** is not defined then it falls back to `__index__()`.

Added in version 3.14.

Calculo del *hash* de tipos numéricos

Para los números `x` y `y`, posiblemente de tipos diferentes, es un requisito que `hash(x) == hash(y)` siempre que `x == y` (consulte la documentación del método `__hash__()` para obtener más detalles). Para facilitar la implementación y la eficiencia en una variedad de tipos numéricos (incluidos `int`, `float`, `decimal.Decimal` y `fractions.Fraction`), el hash de Python para tipos numéricos se basa en una única función matemática que se define para cualquier número racional y, por lo tanto, se aplica a todas las instancias de **`int`** y **`fractions.Fraction`**, y a todas las instancias finitas de **`float`** y **`decimal.Decimal`**. Esencialmente, esta función se da por reducción módulo `P` para un primo fijo `P`. El valor de `P` se pone a disposición de Python como el atributo `modulus` de `sys.hash_info`.

Actualmente, el número primo usado es `P = 2**31 - 1` para máquinas de 32 bits, y `P = 2**61 - 1` en máquinas de 64 bits.

Aquí están las reglas en detalle:

- Si `x = m / n` es un número racional no negativo y `n` no es divisible por `P`, se define `hash(x)` como `m * invmod(n, P) % P`, donde `invmod(n, P)` retorna la inversa de `n` módulo `P`.
- Si `x = m / n` es un número racional no negativo y `n` es divisible por `P` (pero no así `m`), entonces `n` no tiene módulo inverso de `P` y no se puede aplicar la regla anterior; en este caso, `hash(x)` retorna el valor constante definido en `sys.hash_info.inf`.
- Si `x = m / n` es un número racional negativo se define `hash(x)` como `-hash(-x)`. Si el resultado fuera `-1`, lo cambia por `-2`.
- Los valores concretos `sys.hash_info.inf` y `-sys.hash_info.inf` se usan como valores *hash* para infinito positivo o infinito negativo (respectivamente).

- Para un número complejo `z` (una instancia de la clase **`complex`**), el valor de *hash* se calcula combinando los valores de *hash* de la parte real e imaginaria, usando la fórmula `hash(z.real) + sys.hash_info.imag * hash(z.imag)`, módulo reducido `2**sys.hash_info.width`, de forma que el valor obtenido esté en rango `range(-2**(sys.hash_info.width - 1), 2**(sys.hash_info.width - 1))`. De nuevo, si el resultado fuera `-1`, se reemplaza por `-2`.

Para clarificar las reglas previas, aquí mostramos un ejemplo de código Python, equivalente al cálculo realizado en la función *hash*, para calcular el *hash* de un número racional de tipo **`float`** o **`complex`**:

```
import sys, math

def hash_fraction(m, n):
    """Compute the hash of a rational number m / n.

    Assumes m and n are integers, with n positive.
    Equivalent to hash(fractions.Fraction(m, n)).

    """
    P = sys.hash_info.modulus
    # Remove common factors of P.  (Unnecessary if m and
    while m % P == n % P == 0:
        m, n = m // P, n // P

    if n % P == 0:
        hash_value = sys.hash_info.inf
    else:
        # Fermat's Little Theorem: pow(n, P-1, P) is 1,
        # pow(n, P-2, P) gives the inverse of n modulo P
        hash_value = (abs(m) % P) * pow(n, P - 2, P) % P
    if m < 0:
        hash_value = -hash_value
    if hash_value == -1:
        hash_value = -2
    return hash_value

def hash_float(x):
    """Compute the hash of a float x."""

    if math.isnan(x):
        return object.__hash__(x)
    elif math.isinf(x):
        return sys.hash_info.inf if x > 0 else -sys.hash
    else:
        return hash_fraction(*x.as_integer_ratio())

def hash_complex(z):
    """Compute the hash of a complex number z."""

    hash_value = hash_float(z.real) + sys.hash_info.imag
    # do a signed reduction modulo 2**sys.hash_info.widt
    M = 2**(sys.hash_info.width - 1)
    hash_value = (hash_value & (M - 1)) - (hash_value &
    if hash_value == -1:
        hash_value = -2
    return hash_value
```

Tipo Booleano — `bool`

Los valores booleanos representan valores de verdad. El tipo **`bool`** tiene exactamente dos instancias constantes: `True` y `False`.

La función incorporada **`bool()`** convierte cualquier valor en un booleano, si el valor puede interpretarse como un valor de verdad (consulte la sección Evaluar como valor verdadero/falso más arriba).

Para operaciones lógicas, utilice boolean operators, `and`, `or` y `not`. Al aplicar los operadores bit a bit `&`, `|`, `^` a dos booleanos, devuelven un booleano equivalente a las operaciones lógicas «y», «o», «xor». Sin embargo, los operadores lógicos `and`, `or` y `!=` deberían preferirse a `&`, `|` y `^`.

Obsoleto desde la versión 3.12: El uso del operador de inversión bit a bit `~` está obsoleto y generará un error en Python 3.16.

`bool` es una subclase de **`int`** (consulte Tipos numéricos — int, float, complex). En muchos contextos numéricos, `False` y `True` se comportan como los números enteros 0 y 1, respectivamente. Sin embargo, no se recomienda confiar en esto; realice la conversión explícitamente utilizando **`int()`**.

Tipos de iteradores

Python soporta el concepto de iteradores sobre contenedores. Esto se implementa usando dos métodos diferentes: Estos son usados por las clases definidas por el usuario para soportar iteración. Las secuencias, que se describirán con mayor detalle, siempre soportan la iteración.

Es necesario definir un método para que los objetos contenedores proporcionen compatibilidad iterable:

container.__iter__()

> Retorna un objeto iterator. Este objeto es requerido para soportar el protocolo de iteración que se describe a continuación. Si un contenedor soporta diferentes tipos de iteración, se pueden implementar métodos adicionales para estos iteradores. (Por ejemplo, un tipo de contenedor que puede soportar distintas formas de iteración podría ser una estructura de tipo árbol que proporcione a la vez un recorrido en profundidad o en anchura). Este método se corresponde al *slot* **`tp_iter`** de la estructura usada para los objetos Python en la API Python/C.

Los objetos iteradores en si necesitan definir los siguientes dos métodos, que forma juntos el *iterator protocol*:

iterator.__iter__()

> Retorna el propio objeto iterator. Este método es necesario para permitir tanto a los contenedores como a los iteradores usar la palabras clave **`for`** o **`in`**. Este método se corresponde

con el *slot* `tp_iter` de la estructura usada para los objetos Python en la API Python/C.

iterator.__next__()

Retorna el siguiente elemento del iterator. Si no hubiera más elementos, lanza la excepción `StopIteration`. Este método se corresponde con el *slot* `tp_iternext` de la estructura usada para los objetos Python en la API Python/C.

Python define varios objetos iteradores que permiten iterar sobre las secuencias, ya sean generales o específicas, diccionarios y otras estructuras de datos especializadas. Los tipos específicos no son tan importantes como la implementación del protocolo iterador.

Una vez que la ejecución del método `__next__()` lanza la excepción `StopIteration`, debe continuar haciéndolo en subsiguientes llamadas al método. Si una implementación no cumple esto, se considera rota.

Tipos generador

Python’s generators provide a convenient way to implement the iterator protocol. If a container object’s `__iter__()` method is implemented as a generator, it will automatically return an iterator object (technically, a generator object) supplying the `__iter__()` and `__next__()` methods. More information about generators can be found in the documentation for the yield expression.

Tipos secuencia — `list`, `tuple`, `range`

Hay tres tipos básicos de secuencia: listas, tuplas y objetos de tipo rango. Existen tipos de secuencia especiales usados para el procesado de datos binarios y cadenas de caracteres que se describirán en secciones específicas.

Operaciones comunes de las secuencias

Las operaciones de la siguiente tabla están soportadas por la mayoría de los tipos secuencia, tanto mutables como inmutables. La clase ABC **`collections.abc.Sequence`** se incluye para facilitar la implementación correcta de estas operaciones en nuestros propios tipos de secuencias.

La tabla lista las operaciones ordenadas de menor a mayor prioridad. En la tabla, *s* y *t* representan secuencias del mismo tipo, *n*, *i*, *j* y *k* son números enteros y *x* es un objeto arbitrario que cumple con cualquier restricción de tipo o valor impuesta por *s*.

Las operaciones `in` y `not in` tienen la misma prioridad que los operadores de comparación. Las operaciones `+` (concatenación) y `*` (repetición) tienen la misma prioridad que sus equivalentes numéricos. [3]

Operación	**Resultado**	**Notas**
`x in s`	`True` si un elemento de *s* es igual a *x*, `False` en caso contrario	(1)
`x not in s`	`False` si un elemento de *s* es igual a *x*, `True` en caso contrario	(1)
`s + t`	la concatenación de *s* y *t*	(6)(7)
`s * n` o `n * s`	equivale a concatenar *s* consigo mismo *n* veces	(2)(7)
`s[i]`	El elemento *i-esimo* de *s*, empezando a contar en 0	(3)(8)
`s[i:j]`	el segmento de *s* desde *i* hasta *j*	(3)(4)
`s[i:j:k]`	el segmento de *s* desde *i* hasta *j*, con paso *k*	(3)(5)
`len(s)`	longitud de *s*	
`min(s)`	el elemento más pequeño de *s*	
`max(s)`	el elemento más grande de *s*	

También se pueden comparar secuencias del mismo tipo. En particular, las tuplas y las listas se comparan por orden lexicográfico, comparando los elementos en la misma posición. Esto significa que, para que se consideren iguales, todos los elementos correspondientes deben ser iguales entre si, y las dos secuencias deben ser del mismo tipo y de la misma longitud. (Para más detalles, véase Comparaciones en la referencia del lenguaje).

Los iteradores directos e inversos sobre secuencias mutables acceden a valores al usar un índice. Este índice continuará avanzando (o retrocediendo) incluso si la secuencia subyacente está mutada. El iterador termina solo cuando se encuentra un **`IndexError`** o un **`StopIteration`** (o cuando el índice cae por debajo de cero).

Notas:

1. Aunque las operaciones `in` y `not in` se usan generalmente para comprobar si un elemento está dentro de un contenedor, en algunas secuencias especializadas (como **`str`**, **`bytes`** y **`bytearray`**) también se pueden usar para comprobar si está incluida una secuencia:

   ```
   >>> "gg" in "eggs"
   True
   ```

2. Valores de *n* menores que `0` se consideran como `0` (que produce una secuencia vacía del mismo tipo que *s*). Nótese que los elementos de la secuencia *s* no se copian, sino que se referencian múltiples veces. Esto a menudo confunde a programadores noveles de Python; considérese:

   ```
   >>> lists = [[]] * 3
   >>> lists
   [[], [], []]
   >>> lists[0].append(3)
   >>> lists
   [[3], [3], [3]]
   ```

   Lo que ha pasado es que `[[]]` es una lista de un elemento, siendo este elemento una lista vacía, así que los tres elementos de `[[]] * 3` son referencias a la misma lista vacía. Modificar cualquiera de los elementos de `lists` modifica la lista inicial. Para crear una lista de listas independientes entre si, se puede hacer:

   ```
   >>> lists = [[] for i in range(3)]
   >>> lists[0].append(3)
   >>> lists[1].append(5)
   >>> lists[2].append(7)
   >>> lists
   [[3], [5], [7]]
   ```

   Se puede consultar una explicación más completa en esta entrada de la lista de preguntas más frecuentes ¿Cómo puedo crear una lista multidimensional?.

3. Si *i* o *j* es negativo, el índice es relativo al final de la secuencia *s*: Se realiza la sustitución `len(s) + i` o `len(s) + j`. Nótese que `-0` sigue siendo `0`.

4. The slice of *s* from *i* to *j* is defined as the sequence of items with index *k* such that `i <= k < j`.

   - If *i* is omitted or `None`, use `0`.
   - If *j* is omitted or `None`, use `len(s)`.
   - If *i* or *j* is less than `-len(s)`, use `0`.
   - If *i* or *j* is greater than `len(s)`, use `len(s)`.
   - If *i* is greater than or equal to *j*, the slice is empty.

10. El segmento de *s*, desde *i* hasta *j* con paso *k*, se define como la secuencia de elementos con índice `x = i + n*k` tal que `0 <= n < (j-i)/k`. En otras palabras, los índices son `i`, `i+k`, `i+2*k`, `i+3*k` y así consecutivamente, hasta que se alcance el valor de *j* (pero sin incluir nunca *j*). Cuando *k* es positivo, *i* y *j* se limitan al valor de `len(s)`, si fueran mayores. Si *k* es negativo, *i* y *j* se reducen de `len(s) - 1`. Si *i* o *j* se omiten o su valor es `None`, se convierten es valores «finales» (donde el sentido de final depende del signo de *k*). Nótese que *k* no puede valer `0`. Si *k* vale `None`, se considera como `1`.

11. La concatenación de secuencias inmutables siempre produce un nuevo objeto. Esto significa que construir una secuencia usando la concatenación tiene un coste en ejecución cuadrático respecto al tamaño de la secuencia final. Para obtener un rendimiento lineal, se puede optar por una de las alternativas siguientes:

    - en vez de concatenar objetos de tipo **`str`**, se puede construir una lista y usar finalmente el método **`str.join()`**, o bien utilizar una instancia de la clase **`io.StringIO`** y recuperar el valor final completo
    - de forma similar, en vez de concatenar objetos de tipo **`bytes`** se puede usar el método **`bytes.join()`**, la clase **`io.BytesIO`**, o se puede realizar una modificación interna usando un objeto de la clase **`bytearray`**. Los objetos de tipo **`bytearray`** son

mutables y tienen un mecanismo interno de gestión muy eficiente
- en vez de concatenar tuplas (instancias de `tuple`), usar una lista (`list`) y expandirla
- para otros tipos, investiga la documentación relevante de la clase

16. Algunos tipos de secuencia (como la clase `range`) solo soportan elementos que siguen un patrón específico, y por tanto no soportan la concatenación ni la repetición.

17. An `IndexError` is raised if *i* is outside the sequence range.

Sequence Methods

Sequence types also support the following methods:

sequence.count(*value*, */*)

Return the total number of occurrences of *value* in *sequence*.

sequence.index(*value*[, *start*[, *stop*]])

Return the index of the first occurrence of *value* in *sequence*.

Raises `ValueError` if *value* is not found in *sequence*.

The *start* or *stop* arguments allow for efficient searching of subsections of the sequence, beginning at *start* and ending at *stop*. This is roughly equivalent to `start + sequence[start:stop].index(value)`, only without copying any data.

Prudencia

Not all sequence types support passing the *start* and *stop* arguments.

Tipos de secuencia inmutables

La única operación que las secuencias inmutables implementan generalmente, y que no esta definida también en las secuencias mutables, es el soporte para el cálculo de la función predefinida `hash()`.

Este soporte permite usar secuencias inmutables, como por ejemplo las instancias de la clase `tuple`, como claves para diccionarios (`dict`), así como ser almacenadas en conjuntos (`set`) o conjuntos congelados (`frozenset`).

Intentar calcular el *hash* de una secuencia inmutable que contenga objetos mutables producirá una excepción de tipo `TypeError`.

Tipos de secuencia mutables

Las operaciones de la siguiente tabla están definidas para todas los tipos de secuencia mutables. La clase ABC `collections.abc.MutableSequence` se incluye para facilitar la implementación correcta de un tipo de secuencia propio.

En la tabla, *s* es una instancia de una secuencia de tipo mutable, *t* es cualquier objeto iterable y *x* es un objeto arbitrario que cumple las restricciones de tipo y valor que vengan impuestas por *s* (como ejemplo, la clase `bytearray` solo acepta enteros que cumplan la condición `0 <= x <= 255`).

Operación	**Resultado**	**Notas**
`s[i] = x`	el elemento *i* de *s* es reemplazado por *x*	
`del s[i]`	removes item *i* of *s*	
`s[i:j] = t`	el segmento de valores de *s* que van de *i* a *j* es reemplazada por el contenido del iterador *t*	
`del s[i:j]`	removes the elements of `s[i:j]` from the list (same as `s[i:j] = []`)	
`s[i:j:k] = t`	los elementos de `s[i:j:k]` son reemplazados por los elementos de *t*	(1)
`del s[i:j:k]`	borra los elementos de `s[i:j:k]` de la lista	
`s += t`	extiende *s* con los contenidos de *t* (en la mayoría de los casos equivale a `s[len(s):len(s)] = t`)	

actualiza *s* con su contenido repetido *n* veces (2)

Notas:

1. Si *k* no es igual a `1`, *t* debe tener la misma longitud que el segmento a la que reemplaza.
2. El valor de *n* es un entero, o un objeto que implemente el método **`__index__()`**. Los valores negativos, junto con el cero, producen una lista vacía. Los elementos de la secuencia no son copiados, sino que se referencian múltiples veces, tal y como se explicó para `s * n` en Operaciones comunes de las secuencias.

Mutable Sequence Methods

Mutable sequence types also support the following methods:

sequence.append(*value*, */*)

Append *value* to the end of the sequence. This is equivalent to writing `seq[len(seq):len(seq)] = [value]`.

sequence.clear()

Added in version 3.3.

Remove all items from *sequence*. This is equivalent to writing `del sequence[:]`.

sequence.copy()

Added in version 3.3.

Create a shallow copy of *sequence*. This is equivalent to writing `sequence[:]`.

Consejo

The **`copy()`** method is not part of the **`MutableSequence`** **`ABC`**, but most concrete mutable sequence types provide it.

sequence.extend(*iterable*, */*)

Extend *sequence* with the contents of *iterable*. For the most part, this is the same as writing `seq[len(seq):len(seq)] = iterable`.

sequence.insert(*index*, *value*, */*)

Insert *value* into *sequence* at the given *index*. This is equivalent to writing `sequence[index:index] = [value]`.

sequence.pop(*index=-1*, */*)

Retrieve the item at *index* and also removes it from *sequence*. By default, the last item in *sequence* is removed and returned.

sequence.remove(*value*, */*)

Remove the first item from *sequence* where `sequence[i] == value`.

Raises **`ValueError`** if *value* is not found in *sequence*.

sequence.reverse()

Reverse the items of *sequence* in place. This method maintains economy of space when reversing a large sequence. To remind users that it operates by side-effect, it returns `None`.

Listas

Las listas son secuencia mutables, usadas normalmente para

almacenar colecciones de elementos homogéneos (donde el grado de similitud de los mismos depende de la aplicación).

class list(*iterable=()*, */*)

Las listas se pueden construir de diferentes formas:

- Usando un par de corchetes para definir una lista vacía: `[]`
- Usando corchetes, separando los elementos incluidos con comas: `[a]`, `[a, b, c]`
- Usando una lista intensiva o por comprensión: `[x for x in iterable]`
- Usando el constructor de tipo: `list()` o `list(iterable)`

La lista se construye con los mismos elementos y en el mismo orden que *iterable*, donde *iterable* puede ser una secuencia, un contenedor que soporta iteración, o un objeto iterador. Si *iterable* es de por si una lista, se construye y retorna una copia, como si se hubiera llamado a `iterable[:]`. Por ejemplo, `list('abc')` retorna `['a', 'b', 'c']` y `list( (1, 2, 3) )` retorna `[1, 2, 3]`. Si no se pasan parámetros, se construye una nueva lista vacía, `[]`.

Muchas otras operaciones también producen listas, incluyendo la función incorporada **`sorted()`**.

Las listas implementan todas las operaciones comunes y mutables propias de las secuencias. Además, las listas incorporan los siguientes métodos:

sort(***, *key=None*, *reverse=False*)

Este método ordena la lista *in situ* (se modifica internamente), usando únicamente comparaciones de tipo <. Las excepciones no son capturadas internamente: si alguna comparación falla, la operación entera de ordenación falla (y la lista probablemente haya quedado modificada parcialmente).

El método **`sort()`** acepta dos parámetros, que solo pueden pasarse por nombre (keyword-only arguments):

El parámetro *key* especifica una función de un argumento que se usa para obtener, para cada elemento de la lista, un valor concreto o clave (*key*) a usar en las operaciones de comparación (por ejemplo, `key=str.lower`). El elemento clave para cada elemento se calcula una única vez y se reutiliza para todo el proceso de ordenamiento. El valor por defecto, `None`, hace que la lista se ordene comparando directamente los elementos, sin obtener valores clave.

La utilidad **`functools.cmp_to_key()`** se puede usar para convertir una función *cmp* del estilo de la versión 2.x a una función *key*.

El valor de *reverse* es un valor booleano. Si se define como `True`, entonces los elementos de la lista se ordenan como si las operaciones de comparación se hubiesen invertido.

Este método modifica la lista *in situ*, para ahorrar espacio cuando se ordena una secuencia muy grande. Para recordar a los usuarios que funciona de esta manera, no se retorna la secuencia ordenada (puedes usar **`sorted()`** para obtener de forma explicita una nueva secuencia ordenada).

El método **`sort()`** es estable. Un algoritmo de ordenación es estable si garantiza que no se cambia el orden relativo que mantienen inicialmente los elementos que se consideran iguales — esto es útil para realizar ordenaciones en múltiples fases (por ejemplo, ordenar por departamento y después por salario).

Para ver ejemplos de ordenación y un breve tutorial sobre el tema, véase Sorting Techniques.

Mientras una lista está siendo ordenada, los efectos de intentar modificarla, o incluso examinarla, no están definidos. La implementación en C de Python hace que la lista parezca vacía durante la ordenación, y lanza una excepción del tipo **`ValueError`** si detecta un cambio en la lista durante el proceso de ordenación.

Ver también

For detailed information on thread-safety guarantees for **`list`** objects, see Thread safety for list objects.

Tuplas

Las tuplas son secuencias inmutables, usadas normalmente para almacenar colecciones de datos heterogéneos (como las duplas o tuplas de dos elementos producidas por la función incorporada **`enumerate()`**). También son usadas en aquellos casos donde se necesite una secuencia inmutable de datos homogéneos (como por ejemplo permitir el almacenamiento en un objeto de tipo **`set`** o **`dict`**).

class tuple(*iterable*=*()*, */*)

Las tuplas se pueden construir de diferentes maneras:

- Usando un par de símbolos de paréntesis, para indicar una tupla vacía: `()`
- Usando una coma al final, para crear una tupla de un único elemento: `a,` o `(a,)`
- Separando los elementos por comas: `a, b, c` o `(a, b, c)`
- Usando la función incorporada **`tuple()`**: `tuple()` o `tuple(iterable)`

El constructor genera una tupla cuyos elementos son los mismos y están en el mismo orden que los elementos del *iterable*, donde *iterable* puede ser una secuencia, un contenedor que soporta iteración, o un objeto de tipo *iterator*. Si *iterable* es ya de por si una tupla, se retorna sin cambiar. Por ejemplo, `tuple('abc')` retorna `('a', 'b', 'c')` y `tuple( [1, 2, 3] )` retorna `(1, 2, 3)`. Si no se indica ningún parámetro, el constructor creará una nueva tupla vacía. `()`.

Nótese que es la coma la que realmente construye la tupla, no los paréntesis. Los paréntesis son opcionales, excepto en el caso de la tupla vacía, o cuando se necesitan para evitar una ambigüedad sintáctica. Por ejemplo, `f(a, b, c)` es una llamada a una función con tres parámetros, pero `f((a, b, c))` es una llamada a una función con un único parámetro, en este caso una tupla de tres elementos.

Las tuplas implementan todas las operaciones de secuencia comunes.

Para colecciones de datos heterogéneos donde el acceso por nombre resulta más claro que por índice, quizá crear una tupla con nombres (`collections.namedtuple()`) pueden ser más apropiado.

Rangos

Los objetos de tipo `range` representan una secuencia inmutable de números y se usan habitualmente para ejecutar un bucle `for` un número determinado de veces.

class range(*stop*, */*)

class range(*start*, *stop*, *step*=*1*, */*)

Los parámetros usados por el constructor del rango deben ser números enteros (o bien objetos de tipo `int` o instancias de una clase que implemente el método especial `__index__()`). Si el parámetro *step* se omite, se asume el valor `1`. Si se omite el parámetro `start`, se toma como `0`. Si *step* es cero, se lanza una excepción de tipo `ValueError`.

Para un valor positivo de *step*, el contenido del rango `r` viene determinado por la fórmula `r[i] = start + step*i` donde `i >= 0` y `r[i] < stop`.

Para un valor negativo de *step*, el contenido del rango sigue estando determinado por la fórmula `r[i] = start + step*i`, pero las restricciones ahora son `i >= 0` y `r[i] > stop`.

Un objeto de tipo rango se considera vacío si `r[0]` no cumple con las restricciones de valor. Los rangos soportan índices negativos, pero estos son interpretados como índices considerados desde el final de la secuencia determinada por los índices positivos.

Los rangos que contengan valores mayores que **`sys.maxsize`** se permiten, pero algunas capacidades (como la función **`len()`**) pueden lanzar una excepción de tipo **`OverflowError`**.

Ejemplos de rangos:

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]
>>> list(range(0, 10, 3))
[0, 3, 6, 9]
>>> list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> list(range(0))
[]
>>> list(range(1, 0))
[]
```

Los rangos implementan todas las operaciones comunes de las secuencias, excepto la concatenación y la repetición (esto es porque los objetos de tipo rango solamente pueden representar secuencias que siguen un patrón estricto, y tanto la repetición como la concatenación pueden romperlo).

start

> El valor del parámetro *start* (`0` si no se utiliza el parámetro)

stop

> El valor del parámetro *stop*

step

El valor del parámetro *step* (`1` si no se utiliza el parámetro)

La ventaja de usar un objeto de tipo **`range`** en vez de uno de tipo **`list`** o **`tuple`** es que con **`range`** siempre se usa una cantidad fija (y pequeña) de memoria, independientemente del rango que represente (ya que solamente necesita almacenar los valores para `start`, `stop` y `step`, y calcula los valores intermedios a medida que los va necesitando).

Los objetos rango implementan la clase ABC **`collections.abc.Sequence`**, y proporcionan capacidades como comprobación de inclusión, búsqueda de elementos por índice, operaciones de segmentación y soporte de índices negativos (véase Tipos secuencia — list, tuple, range):

```
>>> r = range(0, 20, 2)
>>> r
range(0, 20, 2)
>>> 11 in r
False
>>> 10 in r
True
>>> r.index(10)
5
>>> r[5]
10
>>> r[:5]
range(0, 10, 2)
>>> r[-1]
18
```

La comparación entre rangos usando los operadores `==` y `!=` se realiza como con las secuencias. Esto es, dos rangos se consideran iguales si representan exactamente la misma secuencia de elementos. (Fíjate que, según esta definición, dos rangos pueden considerarse iguales aunque tengan diferentes valores para **`start`**, **`stop`** y **`step`**, por ejemplo `range(0) == range(2, 1, 3)` y `range(0, 3, 2) == range(0, 4, 2)`).

Distinto en la versión 3.2: Implementa la secuencia ABC. Soporta operaciones de segmentación e índices negativos. Comprueba si un entero de tipo `int` está incluido en un rango que se realiza en un tiempo constante, en lugar de iterar a través de todos los elementos.

Distinto en la versión 3.3: Define los operadores "= =" y "! =" para comparar rangos en base a la secuencia de valores que definen (en vez de compararse en base a la identidad).

Se agregan los atributos `start`, `stop` y `step`.

Ver también

- El linspace recipe [https://code.activestate.com/recipes/579000-equally-spaced-numbers-linspace/] muestra cómo implementar una versión perezosa de rango adecuada para aplicaciones de punto flotante.

Text and Binary Sequence Type Methods Summary

The following table summarizes the text and binary sequence types methods by category.

Category	`str` methods	`bytes` and `bytearray` methods
Formatting	`str.format()`	
	`str.format_map()`	
	f-strings	
	Formateo de cadenas al estilo *printf*	Usando el formateo tipo printf con bytes
Searching and Replacing	`str.find()` `str.rfind()`	`bytes.find()` `bytes.rfind()`
	`str.index()` `str.rindex()`	`bytes.index()` `bytes.rindex()`
	`str.startswith()`	`bytes.startswith()`
	`str.endswith()`	`bytes.endswith()`
	`str.count()`	`bytes.count()`
	`str.replace()`	`bytes.replace()`
Splitting and Joining	`str.split()` `str.rsplit()`	`bytes.split()` `bytes.rsplit()`
	`str.splitlines()`	`bytes.splitlines()`

	`str.partition()`	`bytes.partition()`
	`str.rpartition()`	`bytes.rpartition()`
	`str.join()`	`bytes.join()`
String Classification	`str.isalpha()`	`bytes.isalpha()`
	`str.isdecimal()`	
	`str.isdigit()`	`bytes.isdigit()`
	`str.isnumeric()`	
	`str.isalnum()`	`bytes.isalnum()`
	`str.isidentifier()`	
	`str.islower()`	`bytes.islower()`
	`str.isupper()`	`bytes.isupper()`
	`str.istitle()`	`bytes.istitle()`
	`str.isspace()`	`bytes.isspace()`
	`str.isprintable()`	
Case Manipulation	`str.lower()`	`bytes.lower()`
	`str.upper()`	`bytes.upper()`
	`str.casefold()`	
	`str.capitalize()`	`bytes.capitalize()`
	`str.title()`	`bytes.title()`
	`str.swapcase()`	`bytes.swapcase()`
Padding and Stripping	`str.ljust()`	`bytes.ljust()`
	`str.center()`	`bytes.center()`
	`str.expandtabs()`	`bytes.expandtabs()`
	`str.strip()`	`bytes.strip()`
	`str.lstrip()`	`bytes.lstrip()`
Translation and Encoding	`str.translate()`	`bytes.translate()`
	`str.maketrans()`	`bytes.maketrans()`
	`str.encode()`	
		`bytes.decode()`

Cadenas de caracteres — `str`

La información textual se representa en Python con objetos de tipo `str`, normalmente llamados cadenas de caracteres o simplemente *strings*. Las cadenas de caracteres son secuencias inmutables de puntos de código Unicode. Las cadenas se pueden definir de diferentes maneras:

- Comillas simples: `'permite incluir comillas "dobles"'`

- Comillas dobles: `"permite incluir comillas 'simples'"`
- Triples comillas: ya sea con comillas simples `'''Triples comillas simples'''` o dobles `"""Triples comillas dobles"""`

Las cadenas definidas con comillas tripes pueden incluir varias líneas - todos los espacios en blancos incluidos se incorporan a la cadena de forma literal.

Cadenas literales que forman parte de una expresión y que solo estén separados por espacios en blanco, se convertirán implícitamente a una única cadena. Esto es, `("spam " "eggs") == "spam eggs"`.

Consulte Literales de cadenas y bytes para obtener más información sobre las distintas formas de literal de cadena, incluido escape sequences compatible y el prefijo `r` («sin procesar») que deshabilita la mayoría del procesamiento de secuencias de escape.

Las cadenas de caracteres también se pueden crear usando el constructor **`str`**.

Como no hay un tipo separado para los «caracteres», indexar una cadena produce una cadena de longitud 1. Esto es, para una cadena de caracteres no vacía *s*, `s[0] == s[0:1]`.

Tampoco hay una versión mutable de las cadenas de caracteres, pero el método **`str.join()`** o la clase **`io.StringIO`** pueden usarse para construir de forma eficiente una cadena de caracteres a partir de fragmentos.

Distinto en la versión 3.3: Para facilitar la compatibilidad hacia atrás con la versión 2, el prefijo `u` se permite en las cadenas de caracteres. No tiene ningún efecto en la interpretación del literal y no se puede combinar con el prefijo `r`.

class str(**, encoding='utf-8', errors='strict'*)
class str(*object*)
class str(*object, encoding, errors='strict'*)

class str(*object*, *, *errors*)

Retorna una representación en forma de cadena de caracteres de *object*. Si no se proporciona ningún valor, retorna una cadena vacía. Si se proporciona, el comportamiento de `str()` depende de los valores pasados en los parámetros *encoding* y *errors*, como veremos.

Si no se proporciona ni *encoding* ni *errors*, `str(object)` devuelve **`type(object).__str__(object)`**, que es la representación de cadena «informal» o fácilmente imprimible de *object*. Para los objetos de cadena, se trata de la cadena en sí. Si *object* no tiene un método **`__str__()`**, entonces **`str()`** vuelve a devolver **`repr(object)`**.

Si se indica alguno de los dos parámetros *encoding* o *errors*, entonces *object* debe ser un objeto binario o similar (bytes-like object, es decir, una instancia de **`bytes`** o **`bytearray`**). En este caso, si *object* es de tipo **`bytes`** o **`bytearray`**, la llamada a `str(bytes, encoding, errors)` es equivalente a **`bytes.decode(encoding, errors)`**. Si no, el objeto de tipo *bytes* que esta subyacente en el objeto *buffer* se obtiene mediante una llamada a **`bytes.decode()`**. Véase Tipos de secuencias binarias — bytes, bytearray y memoryview y Protocolo búfer para más información sobre los objetos *buffer*.

Si se pasa un objeto de tipo **`bytes`** a la función **`str()`** sin especificar o bien el parámetro *encoding* o bien el *errors*, se vuelve al caso normal donde se retorna la representación informal de la cadena de caracteres (véase también la **`-b`** de las opciones de línea de órdenes de Python). Por ejemplo:

```
>>> str(b'Zoot!')
"b'Zoot!'"
```

Para más información sobre la clase `str` y sus métodos, consulta Cadenas de caracteres — str y la sección Métodos de las cadenas de caracteres a continuación. Para las opciones de formateo de cadenas, lee las secciones f-strings y Format string syntax. También puedes consultar la sección Servicios de procesamiento de texto.

Métodos de las cadenas de caracteres

Todas las cadenas de caracteres implementan las operaciones comunes de las secuencias, junto con los métodos descritos a continuación.

Las cadenas soportan dos estilos de formateo, uno proporciona un grado muy completo de flexibilidad y personalización (véase **`str.format()`**, Format string syntax y Custom string formatting) mientras que el otro se basa en la función C `printf`, que soporta un menor número de tipos y es ligeramente más complicada de usar, pero es a menudo más rápida para los casos que puede manejar (Formateo de cadenas al estilo *printf*).

La sección Servicios de procesamiento de texto de la librería estándar cubre una serie de módulos que proporcionan varias utilidades para trabajar con textos (incluyendo las expresiones regulares en el módulo **`re`**).

str.capitalize()

Retorna una copia de la cadena con el primer carácter en mayúsculas y el resto en minúsculas.

Distinto en la versión 3.8: El primer carácter se pasa ahora a título, más que a mayúsculas. Esto significa que caracteres como dígrafos solo tendrán la primera letra en mayúsculas, en vez de todo el carácter.

str.casefold()

Retorna el texto de la cadena, normalizado a minúsculas. Los textos así normalizados pueden usarse para realizar búsquedas textuales independientes de mayúsculas y minúsculas.

Casefolding is similar to lowercasing but more aggressive because it is intended to remove all case distinctions in a string. For example, the German lowercase letter `'ß'` is equivalent to `"ss"`. Since it is already lowercase, **`lower()`**

would do nothing to `'ß'`; **`casefold()`** converts it to `"ss"`. For example:

```
>>> 'straße'.lower()
'straße'
>>> 'straße'.casefold()
'strasse'
```

The casefolding algorithm is described in section 3.13 “Default Case Folding” of the Unicode Standard [https://www.unicode.org/versions/Unicode16.0.0/core-spec/chapter-3/#G33992].

Added in version 3.3.

str.center(*width*, *fillchar*=' ', /)

Return centered in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to `len(s)`. For example:

```
>>> 'Python'.center(10)
'  Python  '
>>> 'Python'.center(10, '-')
'--Python--'
>>> 'Python'.center(4)
'Python'
```

str.count(*sub*[, *start*[, *end*]])

Retorna el número de ocurrencias no solapadas de la cadena *sub* en el rango [*start*, *end*]. Los parámetros opcionales *start* y *end* se interpretan como en una expresión de segmento.

If *sub* is empty, returns the number of empty strings between characters which is the length of the string plus one. For example:

```
>>> 'spam, spam, spam'.count('spam')
3
>>> 'spam, spam, spam'.count('spam', 5)
```

```
2
>>> 'spam, spam, spam'.count('spam', 5, 10)
1
>>> 'spam, spam, spam'.count('eggs')
0
>>> 'spam, spam, spam'.count('')
17
```

str.encode(*encoding='utf-8'*, *errors='strict'*)

Devuelve la cadena codificada a **`bytes`**.

El valor predeterminado de *encoding* es `'utf-8'`; consulte Codificaciones estándar para conocer los valores posibles.

errors controla cómo se manejan los errores de codificación. Si se usa `'strict'` (el valor predeterminado), se genera una excepción **`UnicodeError`**. Otros valores posibles son `'ignore'`, `'replace'`, `'xmlcharrefreplace'`, `'backslashreplace'` y cualquier otro nombre registrado mediante **`codecs.register_error()`**. Consulte Manejadores de errores para obtener más detalles.

For performance reasons, the value of *errors* is not checked for validity unless an encoding error actually occurs, Modo de desarrollo de Python is enabled or a debug build is used. For example:

```
>>> encoded_str_to_bytes = 'Python'.encode()
>>> type(encoded_str_to_bytes)
<class 'bytes'>
>>> encoded_str_to_bytes
b'Python'
```

Distinto en la versión 3.1: Añadido soporte para poder usar parámetros por nombre.

Distinto en la versión 3.9: El valor del argumento *errors* ahora se comprueba en Modo de desarrollo de Python y en debug mode.

str.endswith(*suffix*[, *start*[, *end*]])

Return `True` if the string ends with the specified *suffix*, otherwise return `False`. *suffix* can also be a tuple of suffixes to look for. With optional *start*, test beginning at that position. With optional *end*, stop comparing at that position. Using *start* and *end* is equivalent to `str[start:end].endswith(suffix)`. For example:

```
>>> 'Python'.endswith('on')
True
>>> 'a tuple of suffixes'.endswith(('at', 'in'))
False
>>> 'a tuple of suffixes'.endswith(('at', 'es'))
True
>>> 'Python is amazing'.endswith('is', 0, 9)
True
```

See also **`startswith()`** and **`removesuffix()`**.

str.expandtabs(*tabsize*=*8*)

Return a copy of the string where all tab characters are replaced by one or more spaces, depending on the current column and the given tab size. Tab positions occur every *tabsize* characters (default is 8, giving tab positions at columns 0, 8, 16 and so on). To expand the string, the current column is set to zero and the string is examined character by character. If the character is a tab (`\t`), one or more space characters are inserted in the result until the current column is equal to the next tab position. (The tab character itself is not copied.) If the character is a newline (`\n`) or return (`\r`), it is copied and the current column is reset to zero. Any other character is copied unchanged and the current column is incremented by one regardless of how the character is represented when printed. For example:

```
>>> '01\t012\t0123\t01234'.expandtabs()
'01      012     0123    01234'
>>> '01\t012\t0123\t01234'.expandtabs(4)
'01  012 0123    01234'
```

```
>>> print('01\t012\n0123\t01234'.expandtabs(4))
01  012
0123    01234
```

str.find(*sub*[, *start*[, *end*]])

Return the lowest index in the string where substring *sub* is found within the slice `s[start:end]`. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` if *sub* is not found. For example:

```
>>> 'spam, spam, spam'.find('sp')
0
>>> 'spam, spam, spam'.find('sp', 5)
6
```

See also **`rfind()`** and **`index()`**.

Nota

El método **`find()`** se debe usar solo si se necesita saber la posición de la cadena *sub*. Si solo se necesita comprobar si *sub* es una parte de *s*, es mejor usar el operador **`in`**:

```
>>> 'Py' in 'Python'
True
```

str.format(**args*, ***kwargs*)

Perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces `{}`. Each replacement field contains either the numeric index of a positional argument, or the name of a keyword argument. Returns a copy of the string where each replacement field is replaced with the string value of the corresponding argument. For example:

```
>>> "The sum of 1 + 2 is {0}".format(1+2)
'The sum of 1 + 2 is 3'
>>> "The sum of {a} + {b} is {answer}".format(answe
'The sum of 1 + 2 is 3'
>>> "{1} expects the {0} Inquisition!".format("Span
'Nobody expects the Spanish Inquisition!'
```

Véase Format string syntax para una descripción de las distintas opciones de formateo que se pueden usar.

Nota

Cuando se formatea un número (**`int`**, **`float`**, **`complex`**, **`decimal.Decimal`** y clases derivadas) usando la `n` (por ejemplo, `'{:n}'.format(1234)`), la función ajusta temporalmente el valor de la variable de entorno local `LC_TYPE` a `LC_NUMERIC` para decodificar los campos `decimal_point` y `thousands_sep` de la función **`localeconv()`**, si usan caracteres que no son ASCII o si ocupan más de un byte, y el valor definido en `LC_NUMERIC` es diferente del definido en `LC_CTYPE`. Estos cambios temporales pueden afectar a otros *threads*.

Distinto en la versión 3.7: Cuando se formatea un número usando la `n`, la función puede asignar de forma temporal la variable `LC_CTYPE` a `LC_NUMERIC` en algunos casos.

str.format_map(*mapping*, */*)

Similar a `str.format(**mapping)`, pero se usa `mapping` de forma directa y no se copia a un diccionario. Esto es útil si `mapping` es, por ejemplo, una instancia de una subclase de **`dict`**:

```
>>> class Default(dict):
...     def __missing__(self, key):
...         return key
...
>>> '{name} was born in {country}'.format_map(Defau
'Guido was born in country'
```

Added in version 3.2.

str.index(*sub*[, *start*[, *end*]])

Like `find()`, but raise `ValueError` when the substring is not found. For example:

```
>>> 'spam, spam, spam'.index('spam')
0
>>> 'spam, spam, spam'.index('eggs')
Traceback (most recent call last):
  File "<python-input-0>", line 1, in <module>
    'spam, spam, spam'.index('eggs')
    ~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^
ValueError: substring not found
```

See also `rindex()`.

str.isalnum()

Return `True` if all characters in the string are alphanumeric and there is at least one character, `False` otherwise. A character `c` is alphanumeric if one of the following returns `True`: `c.isalpha()`, `c.isdecimal()`, `c.isdigit()`, or `c.isnumeric()`. For example:

```
>>> 'abc123'.isalnum()
True
>>> 'abc123!@#'.isalnum()
False
>>> ''.isalnum()
False
>>> ' '.isalnum()
False
```

str.isalpha()

Return `True` if all characters in the string are alphabetic and there is at least one character, `False` otherwise. Alphabetic characters are those characters defined in the Unicode character database as «Letter», i.e., those with general category property being one of «Lm», «Lt», «Lu», «Ll», or «Lo». Note that this is different from the Alphabetic property defined in the section 4.10 “Letters, Alphabetic, and Ideographic” of the Unicode Standard [https://www.unicode.org/versions/Unicode16.0.0/core-spec/chapter-4/#G91002]. For example:

```
>>> 'Letters and spaces'.isalpha()
False
>>> 'LettersOnly'.isalpha()
True
>>> 'µ'.isalpha()  # non-ASCII characters can be co
True
```

See Propiedades Unicode.

str.isascii()

Return `True` if the string is empty or all characters in the string are ASCII, `False` otherwise. ASCII characters have code points in the range U+0000-U+007F. For example:

```
>>> 'ASCII characters'.isascii()
True
>>> 'µ'.isascii()
False
```

Added in version 3.7.

str.isdecimal()

Return `True` if all characters in the string are decimal characters and there is at least one character, `False` otherwise. Decimal characters are those that can be used to form numbers in base 10, such as U+0660, ARABIC-INDIC DIGIT ZERO. Formally a decimal character is a character in the Unicode General Category «Nd». For example:

```
>>> '0123456789'.isdecimal()
True
>>> '٠١٢٣٤٥٦٧٨٩'.isdecimal()  # Arabic-Indic digits ze
True
>>> 'alphabetic'.isdecimal()
False
```

str.isdigit()

Retorna `True` si todos los caracteres de la cadena son dígitos y hay, al menos, un carácter, en caso contrario, retorna `False`. Los dígitos incluyen los caracteres decimales y dígitos que requieren un tratamiento especial, como por ejemplo los usados para superíndices. Esto incluye dígitos que no pueden ser usados para formar números en base 10, como los números Kharosthi. Formalmente, un dígito es un carácter que tiene la propiedad `Numeric_Type` definida como `Digit` o `Decimal`.

str.isidentifier()

Retorna `True` si la cadena de caracteres es un identificador válido de acuerdo a la especificación del lenguaje, véase Names (identifiers and keywords).

`keyword.iskeyword()` se puede utilizar para probar si la cadena `s` es un identificador reservado, como **`def`** y **`class`**.

Ejemplo:

```
>>> from keyword import iskeyword

>>> 'hello'.isidentifier(), iskeyword('hello')
(True, False)
>>> 'def'.isidentifier(), iskeyword('def')
(True, True)
```

str.islower()

Retorna `True` si todos los caracteres de la cadena que tengan formas en mayúsculas y minúsculas [4] están en minúsculas y hay, al menos, un carácter de ese tipo, en caso contrario, retorna `False`.

str.isnumeric()

Return `True` if all characters in the string are numeric characters, and there is at least one character, `False` otherwise. Numeric characters include digit characters, and

all characters that have the Unicode numeric value property, e.g. U+2155, VULGAR FRACTION ONE FIFTH. Formally, numeric characters are those with the property value Numeric_Type=Digit, Numeric_Type=Decimal or Numeric_Type=Numeric. For example:

```
>>> '0123456789'.isnumeric()
True
>>> '٠١٢٣٤٥٦٧٨٩'.isnumeric()  # Arabic-indic digit zer
True
>>> '⅕'.isnumeric()  # Vulgar fraction one fifth
True
>>> '²'.isdecimal(), '²'.isdigit(),  '²'.isnumeric(
(False, True, True)
```

See also `isdecimal()` and `isdigit()`. Numeric characters are a superset of decimal numbers.

str.isprintable()

Return `True` if all characters in the string are printable, `False` if it contains at least one non-printable character.

Here «printable» means the character is suitable for `repr()` to use in its output; «non-printable» means that **`repr()`** on built-in types will hex-escape the character. It has no bearing on the handling of strings written to `sys.stdout` or `sys.stderr`.

The printable characters are those which in the Unicode character database (see `unicodedata`) have a general category in group Letter, Mark, Number, Punctuation, or Symbol (L, M, N, P, or S); plus the ASCII space 0x20. Nonprintable characters are those in group Separator or Other (Z or C), except the ASCII space.

For example:

```
>>> ''.isprintable(), ' '.isprintable()
(True, True)
>>> '\t'.isprintable(), '\n'.isprintable()
```

```
(False, False)
```

See also **`isspace()`**.

str.isspace()

Retorna `True` si todos los caracteres de la cadena son espacios en blanco y hay, al menos, un carácter, en caso contrario, retorna `False`.

For example:

```
>>> ''.isspace()
False
>>> ' '.isspace()
True
>>> '\t\n'.isspace() # TAB and BREAK LINE
True
>>> '\u3000'.isspace() # IDEOGRAPHIC SPACE
True
```

Un carácter se considera espacio en blanco si, en la base de datos de Unicode (véase **`unicodedata`**), está clasificado en la categoría general `Zs` («Separador, espacio») o la clase bidireccional es `WS`, `B` o `S`.

See also **`isprintable()`**.

str.istitle()

Retorna `True` si las palabras en la cadena tiene forma de título y hay, al menos, un carácter, por ejemplo una mayúscula solo puede aparecer al principio o después de un carácter que no tenga formas alternativas mayúsculas-minúsculas, y las minúsculas solo después de carácter que si tiene formas alternativas mayúsculas-minúsculas. En caso contrario, retorna `False`.

For example:

```
>>> 'Spam, Spam, Spam'.istitle()
True
```

```
>>> 'spam, spam, spam'.istitle()
False
>>> 'SPAM, SPAM, SPAM'.istitle()
False
```

See also **`title()`**.

str.isupper()

Retorna `True` si todos los caracteres de la cadena que tengan formas en mayúsculas y minúsculas [4] están en mayúsculas y hay, al menos, un carácter de ese tipo, en caso contrario, retorna `False`.

```
>>> 'BANANA'.isupper()
True
>>> 'banana'.isupper()
False
>>> 'baNana'.isupper()
False
>>> ' '.isupper()
False
```

str.join(*iterable*, /)

Return a string which is the concatenation of the strings in *iterable*. A **`TypeError`** will be raised if there are any non-string values in *iterable*, including **`bytes`** objects. The separator between elements is the string providing this method. For example:

```
>>> ', '.join(['spam', 'spam', 'spam'])
'spam, spam, spam'
>>> '-'.join('Python')
'P-y-t-h-o-n'
```

See also **`split()`**.

str.ljust(*width*, *fillchar*=' ', /)

Retorna el texto de la cadena, justificado a la izquierda en una cadena de longitud *width*. El carácter de relleno a usar viene definido por el parámetro *fillchar* (por defecto se usa el carácter espacio ASCII). Si la cadena original tiene una longitud `len(s)` igual o superior a *width*, se retorna el texto sin modificar.

For example:

```
>>> 'Python'.ljust(10)
'Python    '
>>> 'Python'.ljust(10, '.')
'Python....'
>>> 'Monty Python'.ljust(10, '.')
'Monty Python'
```

See also **`rjust()`**.

str.lower()

Return a copy of the string with all the cased characters [4] converted to lowercase. For example:

```
>>> 'Lower Method Example'.lower()
'lower method example'
```

The lowercasing algorithm used is described in section 3.13 “Default Case Folding” of the Unicode Standard [https://www.unicode.org/versions/Unicode16.0.0/core-spec/chapter-3/#G33992].

str.lstrip(*chars=None*, /)

Retorna una copia de la cadena, eliminado determinados caracteres si se encuentren al principio. El parámetro *chars* especifica el conjunto de caracteres a eliminar. Si se omite o si se especifica `None`, se eliminan todos los espacios en blanco. No debe entenderse el valor de *chars* como un prefijo, sino que se elimina cualquier combinación de sus caracteres:

```
>>> '   spacious   '.lstrip()
'spacious   '
>>> 'www.example.com'.lstrip('cmowz.')
```

```
'example.com'
```

Véase `str.removeprefix()` para un método que removerá una única cadena de prefijo en lugar de todas las ocurrencias dentro de un set de caracteres. Por ejemplo:

```
>>> 'Arthur: three!'.lstrip('Arthur: ')
'ee!'
>>> 'Arthur: three!'.removeprefix('Arthur: ')
'three!'
```

static str.maketrans(*dict*, /)
static str.maketrans(*from*, *to*, *remove*='', /)

Este método estático retorna una tabla de traducción apta para ser usada por el método `str.translate()`.

Si solo se usa un parámetro, este debe ser un diccionario que mapea valores de punto Unicode (enteros) o caracteres (cadenas de longitud 1) a valores Unicode, cadenas (de cualquier longitud) o `None`. Las claves se convertirán a ordinales.

If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in *from* will be mapped to the character at the same position in *to*. If there is a third argument, it must be a string, whose characters will be mapped to `None` in the result.

str.partition(*sep*, /)

Divide la cadena en la primera ocurrencia de *sep*, y retorna una tupla de tres elementos, que contiene la parte anterior al separador, el separador en sí y la parte posterior al separador. Si no se encuentra el separador, retorna una tupla de tres elementos, el primero contiene la cadena original y los dos siguientes son cadenas vacías.

For example:

```
>>> 'Monty Python'.partition(' ')
```

```
('Monty', ' ', 'Python')
>>> "Monty Python's Flying Circus".partition(' ')
('Monty', ' ', "Python's Flying Circus")
>>> 'Monty Python'.partition('-')
('Monty Python', '', '')
```

See also **`rpartition()`**.

str.removeprefix(*prefix*, */*)

If the string starts with the *prefix* string, return `string[len(prefix):]`. Otherwise, return a copy of the original string:

```
>>> 'TestHook'.removeprefix('Test')
'Hook'
>>> 'BaseTestCase'.removeprefix('Test')
'BaseTestCase'
```

Added in version 3.9.

See also **`removesuffix()`** and **`startswith()`**.

str.removesuffix(*suffix*, */*)

If the string ends with the *suffix* string and that *suffix* is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string:

```
>>> 'MiscTests'.removesuffix('Tests')
'Misc'
>>> 'TmpDirMixin'.removesuffix('Tests')
'TmpDirMixin'
```

Added in version 3.9.

See also **`removeprefix()`** and **`endswith()`**.

str.replace(*old*, *new*, */*, *count=-1*)

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If *count* is given, only the first *count* occurrences are replaced. If *count* is not specified or `-1`, then all occurrences are replaced. For example:

```
>>> 'spam, spam, spam'.replace('spam', 'eggs')
'eggs, eggs, eggs'
>>> 'spam, spam, spam'.replace('spam', 'eggs', 1)
'eggs, spam, spam'
```

Distinto en la versión 3.13: count es ahora soportado como argumento de palabra clave.

str.rfind(*sub*[, *start*[, *end*]])

Return the highest index in the string where substring *sub* is found, such that *sub* is contained within `s[start:end]`. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure. For example:

```
>>> 'spam, spam, spam'.rfind('sp')
12
>>> 'spam, spam, spam'.rfind('sp', 0, 10)
6
```

See also **`find()`** and **`rindex()`**.

str.rindex(*sub*[, *start*[, *end*]])

Like **`rfind()`** but raises **`ValueError`** when the substring *sub* is not found. For example:

```
>>> 'spam, spam, spam'.rindex('spam')
12
>>> 'spam, spam, spam'.rindex('eggs')
Traceback (most recent call last):
  File "<stdin-0>", line 1, in <module>
    'spam, spam, spam'.rindex('eggs')
    ~~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^
ValueError: substring not found
```

See also **`index()`** and **`find()`**.

str.rjust(*width*, *fillchar=' '*, */*)

Retorna el texto de la cadena, justificado a la derecha en una cadena de longitud *width*. El carácter de relleno a usar viene definido por el parámetro *fillchar* (por defecto se usa el carácter espacio ASCII). Si *width* es menor o igual que `len(s)`, se retorna el texto sin modificar.

For example:

```
>>> 'Python'.rjust(10)
'    Python'
>>> 'Python'.rjust(10, '.')
'....Python'
>>> 'Monty Python'.rjust(10, '.')
'Monty Python'
```

See also **`ljust()`** and **`zfill()`**.

str.rpartition(*sep*, */*)

Divide la cadena en la última ocurrencia de *sep*, y retorna una tupla de tres elementos, conteniendo la parte anterior al separador, el separador en sí y la parte posterior al separador. Si no se encuentra el separador, retorna una tupla de tres elementos, los dos primeras son posiciones con cadenas vacías y en la tercera la cadena original.

For example:

```
>>> 'Monty Python'.rpartition(' ')
('Monty', ' ', 'Python')
>>> "Monty Python's Flying Circus".rpartition(' ')
("Monty Python's Flying", ' ', 'Circus')
>>> 'Monty Python'.rpartition('-')
('', '', 'Monty Python')
```

See also **`partition()`**.

str.rsplit(*sep=None*, *maxsplit=-1*)

Retorna una lista con las palabras que componen la cadena de caracteres original, usando como separador el valor de *sep*. Si se utiliza el parámetro *maxsplit*, se realizan como máximo *maxsplit* divisiones, retornando los que están más a la derecha. Si no se especifica *sep* o se pasa con valor `None`, se usa como separador cualquier carácter de espacio en blanco. Si no contamos la diferencia de empezar las divisiones desde la derecha, el comportamiento de este método **`rsplit()`** es equivalente al de **`split()`**, que se describe con detalle más adelante.

str.rstrip(*chars=None*, */*)

Return a copy of the string with trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a suffix; rather, all combinations of its values are stripped. For example:

```
>>> '   spacious   '.rstrip()
'   spacious'
>>> 'mississippi'.rstrip('ipz')
'mississ'
```

See **`removesuffix()`** for a method that will remove a single suffix string rather than all of a set of characters. For example:

```
>>> 'Monty Python'.rstrip(' Python')
'M'
>>> 'Monty Python'.removesuffix(' Python')
'Monty'
```

See also **`strip()`**.

str.split(*sep=None*, *maxsplit=-1*)

Retorna una lista con las palabras que componen la cadena de caracteres original, usando como separador el valor de *sep*. Si se utiliza el parámetro *maxsplit*, se realizan como máximo *maxsplit* divisiones (por tanto, la lista resultante tendrá `maxsplit+1` elementos). Si no se especifica *maxsplit* o se pasa con valor `-1`, entonces no hay límite al número de divisiones a realizar (se harán todas las que se puedan).

Si se proporciona *sep*, los delimitadores consecutivos no se agrupan y se consideran que delimitan cadenas vacías (por ejemplo, `'1,,2'.split(',')` devuelve `['1', '', '2']`). El argumento *sep* puede constar de varios caracteres como un único delimitador (para dividir con varios delimitadores, utilice **`re.split()`**). Dividir una cadena vacía con un separador especificado devuelve `['']`.

Por ejemplo:

```
>>> '1,2,3'.split(',')
['1', '2', '3']
>>> '1,2,3'.split(',', maxsplit=1)
['1', '2,3']
>>> '1,2,,3,'.split(',')
['1', '2', '', '3', '']
>>> '1<>2<>3<4'.split('<>')
['1', '2', '3<4']
```

Si no se especifica *sep* o es `None`, se usa un algoritmo de división diferente: secuencias consecutivas de caracteres de espacio en blanco se consideran como un único separador, y el resultado no contendrá cadenas vacías ni al principio ni al final de la lista, aunque la cadena original tuviera espacios en blanco al principio o al final. En consecuencia, dividir una cadena vacía o una cadena que solo contenga espacios en blanco usando `None` como separador siempre retornará una lista vacía `[]`.

Por ejemplo:

```
>>> '1 2 3'.split()
['1', '2', '3']
>>> '1 2 3'.split(maxsplit=1)
['1', '2 3']
>>> '   1   2   3   '.split()
```

```
['1', '2', '3']
```

If *sep* is not specified or is `None` and *maxsplit* is `0`, only leading runs of consecutive whitespace are considered.

Por ejemplo:

```
>>> "".split(None, 0)
[]
>>> "   ".split(None, 0)
[]
>>> "   foo   ".split(maxsplit=0)
['foo   ']
```

See also **`join()`**.

str.splitlines(*keepends*=*False*)

Retorna una lista con las líneas en la cadena, dividiendo por los saltos de línea. Los caracteres de salto de línea en sí no se incluyen a no ser que se especifique lo contrario pasando el valor `True` en al parámetro *keepends*.

Este método considera como saltos de línea los siguientes caracteres. En concreto, estos son un superconjunto de los saltos de líneas universales.

Representación	**Descripción**
`\n`	Salto de línea
`\r`	Retorno de carro
`\r\n`	Retorno de carro + salto de línea
`\v` o `\x0b`	Tabulación de línea
`\f` o `\x0c`	Avance de página
`\x1c`	Separador de archivo
`\x1d`	Separador de grupo
`\x1e`	Separador de registro
`\x85`	Siguiente línea (Código de control C1)
`\u2028`	Separador de línea
`\u2029`	Separador de párrafo

Distinto en la versión 3.2: Se añadieron `\v` y `\f` a la lista de separadores.

Por ejemplo:

```
>>> 'ab c\n\nde fg\rkl\r\n'.splitlines()
['ab c', '', 'de fg', 'kl']
>>> 'ab c\n\nde fg\rkl\r\n'.splitlines(keepends=Tru
['ab c\n', '\n', 'de fg\r', 'kl\r\n']
```

Al contrario que con **`split()`**, cuando se especifica una cadena con *sep*, el método retorna una lista vacía para la cadena vacía, y un salto de línea al final del texto no produce una línea extra:

```
>>> "".splitlines()
[]
>>> "One line\n".splitlines()
['One line']
```

Por comparación, `split('\n')` entrega:

```
>>> ''.split('\n')
['']
>>> 'Two lines\n'.split('\n')
['Two lines', '']
```

str.startswith(*prefix*[, *start*[, *end*]])

Retorna `True` si la cadena empieza por *prefix*, en caso contrario retorna `False`. El valor de *prefix* puede ser también una tupla de prefijos por los que buscar. Con el parámetro opcional *start*, la comprobación empieza en esa posición de la cadena. Con el parámetro opcional *end*, la comprobación se detiene en esa posición de la cadena.

For example:

```
>>> 'Python'.startswith('Py')
True
>>> 'a tuple of prefixes'.startswith(('at', 'a'))
True
```

```
>>> 'Python is amazing'.startswith('is', 7)
True
```

See also `endswith()` and `removeprefix()`.

str.strip(*chars=None*, /)

Return a copy of the string with the leading and trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a prefix or suffix; rather, all combinations of its values are stripped.

For example:

```
>>> '   spacious   '.strip()
'spacious'
>>> 'www.example.com'.strip('cmowz.')
'example'
```

The outermost leading and trailing *chars* argument values are stripped from the string. Characters are removed from the leading end until reaching a string character that is not contained in the set of characters in *chars*. A similar action takes place on the trailing end.

For example:

```
>>> comment_string = '#....... Section 3.2.1 Issue
>>> comment_string.strip('.#! ')
'Section 3.2.1 Issue #32'
```

See also `rstrip()`.

str.swapcase()

Return a copy of the string with uppercase characters converted to lowercase and vice versa. For example:

```
>>> 'Hello World'.swapcase()
'hELLO wORLD'
```

Note that it is not necessarily true that `s.swapcase().swapcase() == s`. For example:

```
>>> 'straße'.swapcase().swapcase()
'strasse'
```

See also `str.lower()` and `str.upper()`.

str.title()

Retorna una versión en forma de título de la cadena, con la primera letra de cada palabra en mayúsculas y el resto en minúsculas.

Por ejemplo:

```
>>> 'Hello world'.title()
'Hello World'
```

El algoritmo usa una definición sencilla e independiente del lenguaje, consistente en considerar una palabra como un grupo de letras consecutivas. Esta definición funciona en varios entornos, pero implica que, por ejemplo en inglés, los apóstrofos en las contracciones y en los posesivos constituyen una separación entre palabras, que puede que no sea el efecto deseado:

```
>>> "they're bill's friends from the UK".title()
"They'Re Bill'S Friends From The Uk"
```

La función `string.capwords()` no tiene este problema, ya que solo divide palabras en espacios.

Alternativamente, se puede solucionar parcialmente el problema de los apóstrofos usando expresiones regulares:

```
>>> import re
>>> def titlecase(s):
...     return re.sub(r"[A-Za-z]+('[A-Za-z]+)?",
...                   lambda mo: mo.group(0).capita
...                   s)
```

```
...
>>> titlecase("they're bill's friends.")
"They're Bill's Friends."
```

See also `istitle()`.

str.translate(*table*, */*)

Devuelve una copia de la cadena en la que se ha asignado cada carácter a través de la tabla de traducción dada. La tabla debe ser un objeto que implemente la indexación mediante `__getitem__()`, normalmente mapping o sequence. Cuando se indexa mediante un ordinal Unicode (un entero), el objeto de tabla puede realizar cualquiera de las siguientes acciones: devolver un ordinal Unicode o una cadena, para asignar el carácter a uno o más caracteres; devolver `None`, para eliminar el carácter de la cadena de retorno; o generar una excepción `LookupError`, para asignar el carácter a sí mismo.

Se puede usar `str.maketrans()` para crear un mapa de traducción carácter a carácter de diferentes formas.

Véase también el módulo `codecs` para una aproximación más flexible al mapeo de caracteres.

str.upper()

Retorna una copia de la cadena, con todos los caracteres con formas mayúsculas/minúsculas [4] pasados a mayúsculas. Nótese que `s.upper().isupper()` puede ser `False` si `s` contiene caracteres que no tengan las dos formas, o si la categoría Unicode del carácter o caracteres resultantes no es «Lu» (Letra, mayúsculas), sino, por ejemplo, «Lt» (Letra, Título).

The uppercasing algorithm used is described in section 3.13 “Default Case Folding” of the Unicode Standard [https://www.unicode.org/versions/Unicode16.0.0/core-spec/chapter-3/#G33992].

str.zfill(*width*, */*)

Retorna una copia de la cadena, rellena por la izquierda con dígitos `'0'` de ASCII necesarios para conseguir una cadena de longitud *width*. El carácter prefijo de signo (`'+'`/`'-'`) se gestiona insertando el relleno *después* del carácter de signo en vez de antes. Si *width* es menor o igual que `len(s)`, se retorna la cadena original.

For example:

```
>>> "42".zfill(5)
'00042'
>>> "-42".zfill(5)
'-0042'
```

See also **`rjust()`**.

Formatted String Literals (f-strings)

Added in version 3.6.

Distinto en la versión 3.7: The **`await`** and **`async for`** can be used in expressions within f-strings.

Distinto en la versión 3.8: Added the debug specifier (=)

Distinto en la versión 3.12: Many restrictions on expressions within f-strings have been removed. Notably, nested strings, comments, and backslashes are now permitted.

An *f-string* (formally a *formatted string literal*) is a string literal that is prefixed with `f` or `F`. This type of string literal allows embedding the results of arbitrary Python expressions within *replacement fields*, which are delimited by curly brackets (`{}`). Each replacement field must contain an expression, optionally followed by:

- a *debug specifier* – an equal sign (=);
- a *conversion specifier* – `!s`, `!r` or `!a`; and/or
- a *format specifier* prefixed with a colon (`:`).

See the Lexical Analysis section on f-strings for details on the syntax of these fields.

Debug specifier

Added in version 3.8.

If a debug specifier – an equal sign (=) – appears after the replacement field expression, the resulting f-string will contain the expression's source, the equal sign, and the value of the expression. This is often useful for debugging:

```
>>> number = 14.3
>>> f'{number=}'
'number=14.3'
```

Whitespace before, inside and after the expression, as well as whitespace after the equal sign, is significant — it is retained in the result:

```
>>> f'{ number  -  4  = }'
' number  -  4  = 10.3'
```

Conversion specifier

By default, the value of a replacement field expression is converted to a string using **`str()`**:

```
>>> from fractions import Fraction
>>> one_third = Fraction(1, 3)
>>> f'{one_third}'
'1/3'
```

When a debug specifier but no format specifier is used, the default conversion instead uses **`repr()`**:

```
>>> f'{one_third = }'
'one_third = Fraction(1, 3)'
```

The conversion can be specified explicitly using one of these specifiers:

- `!s` for **`str()`**
- `!r` for **`repr()`**

- `!a` for **`ascii()`**

Por ejemplo:

```
>>> str(one_third)
'1/3'
>>> repr(one_third)
'Fraction(1, 3)'

>>> f'{one_third!s} is {one_third!r}'
'1/3 is Fraction(1, 3)'

>>> string = "¡kočka 😸!"
>>> ascii(string)
"'\\xa1ko\\u010dka \\U0001f638!'"

>>> f'{string = !a}'
"string = '\\xa1ko\\u010dka \\U0001f638!'"
```

Format specifier

After the expression has been evaluated, and possibly converted using an explicit conversion specifier, it is formatted using the **`format()`** function. If the replacement field includes a *format specifier* introduced by a colon (`:`), the specifier is passed to **`format()`** as the second argument. The result of **`format()`** is then used as the final value for the replacement field. For example:

```
>>> from fractions import Fraction
>>> one_third = Fraction(1, 3)
>>> f'{one_third:.6f}'
'0.333333'
>>> f'{one_third:_^+10}'
'___+1/3___'
>>> >>> f'{one_third!r:_^20}'
'___Fraction(1, 3)___'
>>> f'{one_third = :~>10}~'
'one_third = ~~~~~~~1/3~'
```

Template String Literals (t-strings)

An *t-string* (formally a *template string literal*) is a string literal that is prefixed with `t` or `T`.

These strings follow the same syntax and evaluation rules as formatted string literals, with for the following differences:

- Rather than evaluating to a `str` object, template string literals evaluate to a **`string.templatelib.Template`** object.
- The **`format()`** protocol is not used. Instead, the format specifier and conversions (if any) are passed to a new **`Interpolation`** object that is created for each evaluated expression. It is up to code that processes the resulting **`Template`** object to decide how to handle format specifiers and conversions.
- Format specifiers containing nested replacement fields are evaluated eagerly, prior to being passed to the **`Interpolation`** object. For instance, an interpolation of the form `{amount:.{precision}f}` will evaluate the inner expression `{precision}` to determine the value of the `format_spec` attribute. If `precision` were to be `2`, the resulting format specifier would be `'.2f'`.
- When the equals sign `'='` is provided in an interpolation expression, the text of the expression is appended to the literal string that precedes the relevant interpolation. This includes the equals sign and any surrounding whitespace. The **`Interpolation`** instance for the expression will be created as normal, except that **`conversion`** will be set to "`r`" (**`repr()`**) by default. If an explicit conversion or format specifier are provided, this will override the default behaviour.

Formateo de cadenas al estilo **`*printf*`**

Nota

The formatting operations described here exhibit a variety of quirks that lead to a number of common errors (such as failing to

display tuples and dictionaries correctly).

> Using formatted string literals, the `str.format()` interface, or `string.Template` may help avoid these errors. Each of these alternatives provides their own trade-offs and benefits of simplicity, flexibility, and/or extensibility.

Los objetos de cadena tienen una única operación incorporada: el operador `%` (módulo). También se lo conoce como operador de cadena *formatting* o *interpolation*. Dado `format % values` (donde *format* es una cadena), las especificaciones de conversión `%` en *format* se reemplazan con cero o más elementos de *values*. El efecto es similar al uso de la función **`sprintf()`** en el lenguaje C. Por ejemplo:

```
>>> print('%s has %d quote types.' % ('Python', 2))
Python has 2 quote types.
```

Si *formato* tiene un único marcador, *valores* puede ser un objeto sencillo, no una tupla. [5] En caso contrario, *valores* debe ser una tupla con exactamente el mismo número de elementos que marcadores usados en la cadena de formato, o un único objeto de tipo mapa (por ejemplo, un diccionario).

Un especificador de conversión consiste en dos o más caracteres y tiene los siguientes componentes, que deben aparecer en el siguiente orden:

1. El carácter `'%'`, que identifica el inicio del marcador.
2. Una clave de mapeo (opcional), consistente en una secuencia de caracteres entre paréntesis, como por ejemplo, `(somename)`.
3. Indicador de conversión (opcional), que afecta el resultado de ciertas conversiones de tipos.
4. Valor de ancho mínimo (opcional). Si se especifica un `'*'` (asterisco), el ancho real se lee del siguiente elemento de la tupla *valores*, y el objeto a convertir viene después del ancho mínimo, con un indicador de precisión opcional.
5. Precisión (opcional), en la forma `'.'` (punto) seguido de la precisión. Si se especifica un `'*'` (asterisco), el valor de

   precisión real se lee del siguiente elemento de la tupla *valores*, y el valor a convertir viene después de la precisión.
6. Modificador de longitud (opcional).
7. Tipo de conversión.

Cuando el operador derecho es un diccionario (o cualquier otro objeto de tipo mapa), los marcadores en la cadena *deben* incluir un valor de clave entre paréntesis, inmediatamente después del carácter `'%'`. El valor de la clave se usa para seleccionar el valor a formatear desde el mapa. Por ejemplo:

```
>>> print('%(language)s has %(number)03d quote types.' %
...       {'language': "Python", "number": 2})
Python has 002 quote types.
```

En este caso, no se puede usar el especificador `*` en la cadena de formato (dado que requiere una lista secuencial de parámetros).

Los indicadores de conversión son:

Flag	Significado
`'#'`	El valor a convertir usara la «forma alternativa» (que se definirá más adelante)
`'0'`	La conversión rellena con ceros por la izquierda para valores numéricos.
`'-'`	El valor convertido se ajusta a la izquierda (sobreescribe la conversión `'0'` si se especifican los dos)
`' '`	(Un espacio) Se debe añadir un espacio en blanco antes de un número positivo (o una cadena vacía) si se usa una conversión con signo.
`'+'`	Un carácter signo (`'+'` o `'-'`) precede a la conversión (sobreescribe el indicador de «espacio»)

Puede estar presente un modificador de longitud (`h`, `l` o `L`), pero se ignora y no es necesario para Python – por lo que, por ejemplo, la salida de `%ld` es idéntica a `%d`.

Los tipos de conversión son:

Conversión	Significado
`'d'`	Entero decimal con signo.

`'i'`	Entero decimal con signo.	
`'o'`	Valor octal con signo.	(1)
`'u'`	Obsoleto – es idéntico a `'d'`.	(6)
`'x'`	Hexadecimal con signo (en minúsculas).	(2)
`'X'`	Hexadecimal con signo (en mayúsculas).	(2)
`'e'`	Formato en coma flotante exponencial (en minúsculas).	(3)
`'E'`	Formato en coma flotante exponencial (en mayúsculas).	(3)
`'f'`	Formato en coma flotante decimal.	(3)
`'F'`	Formato en coma flotante decimal.	(3)
`'g'`	Formato de punto flotante. Utiliza el formato exponencial en minúsculas si el exponente es menor que -4 o igual que la precisión; en caso contrario, utiliza el formato decimal.	(4)
`'G'`	Formato de punto flotante. Utiliza el formato exponencial en mayúsculas si el exponente es menor que -4 o igual que la precisión; en caso contrario, utiliza el formato decimal.	(4)
`'c'`	Un único carácter (acepta números enteros o cadenas de caracteres de longitud 1).	
`'r'`	Cadena de caracteres (representará cualquier objeto usando la función **`repr()`**).	(5)
`'s'`	Cadena de caracteres (representará cualquier objeto usando la función **`str()`**).	(5)
`'a'`	Cadena de caracteres (representará cualquier objeto usando la función **`ascii()`**).	(5)
`'%'`	No se representa ningún argumento, obteniéndose en el resultado la cadena `'%'`.	

For floating-point formats, the result should be correctly rounded to a given precision `p` of digits after the decimal point. The rounding mode matches that of the **`round()`** builtin.

Notas:

1. La forma alternativa hace que se inserte antes del primer dígito un prefijo indicativo del formato octal (`'0o'`)

2. La forma alternativa hace que se inserte antes del primer dígito uno de los dos prefijos indicativos del formato hexadecimal `'0x'` o `'0X'` (que se use uno u otro depende de que indicador de formato se haya usado, `'x'` o `'X'`).

3. La forma alternativa hace que se incluya siempre el símbolo

del punto o coma decimal, incluso si no hubiera dígitos después.

La precisión determina el número de dígitos que vienen después del punto decimal, y por defecto es 6.

4. La forma alternativa hace que se incluya siempre el símbolo del punto o coma decimal, y los ceros a su derecha no se eliminan, como seria el caso en la forma normal.

   La precisión determina el número de dígitos significativos que vienen antes y después del punto decimal, y por defecto es 6.

5. Si la precisión es `N`, la salida se trunca a `N` caracteres.

6. Véase **PEP 237** [https://peps.python.org/pep-0237/].

Como en Python las cadenas de caracteres tiene una longitud explícita, la conversión de `%s` no requiere que la cadena termine con `'\0'`.

Distinto en la versión 3.1: Las conversiones `%f` para números con valores absolutos mayores que 1e50 ya no son reemplazadas por conversiones `%g`.

Tipos de secuencias binarias — `bytes`, `bytearray` y `memoryview`

Los tipos básicos para trabajar con datos binarios son las clases **`bytes`** y **`bytearray`**. Ambas pueden ser usadas por la clase **`memoryview`**, que usa el protocolo buffer para acceder a la memoria de otros objetos binarios sin necesidad de hacer una copia.

El módulo **`array`** soporta un almacenamiento eficiente de tipos de datos básicos como enteros de 32 bits o números en formato de doble precisión en coma flotante IEEE754.

Objetos de tipo Bytes

Los objetos *bytes* son secuencias inmutables de bytes. Como muchos de los protocolos binarios más usados se basan en la codificación ASCII para texto, los objetos *bytes* ofrecen varios métodos que solo son válidos cuando se trabaja con datos compatibles ASCII y son, en varios aspectos, muy cercanos a los cadenas de caracteres.

class bytes(*source=b''*)
class bytes(*source*, *encoding*, *errors='strict'*)

Para empezar, la sintaxis de los valores literales de *bytes* son prácticamente iguales que para las cadenas de caracteres, con la diferencia de que se añade el carácter `b` como prefijo:

- Comillas sencillas: `b'Se siguen aceptando comillas "dobles" embebidas'`
- Comillas dobles: `b"Se siguen aceptando comillas 'simples' embebidas"`
- Comillas triples: `b'''3 comillas simples'''`, `b"""3 comillas dobles"""`

Solo se admiten caracteres ASCII en representaciones literales de *bytes* (con independencia del tipo de codificación declarado). Cualquier valor por encima de 127 debe ser definido usando su secuencia de escape.

Al igual que con las cadenas, los literales de *bytes* pueden usar el prefijo `r` para deshabilitar el procesado de las secuencias de escape. Véase Literales de cadenas y bytes para más información acerca de los diferentes formas de expresar *bytes* de forma literal, incluyendo el soporte de secuencias de escape.

Aunque las secuencias de bytes y sus representaciones se basen en texto ASCII, los objetos *bytes* se comportan más como secuencias inmutables de números enteros, donde cada elemento de la secuencia está restringido a los valores de *x* tal que `0 <= x < 256` (si se intenta violar esta restricción se lanzará una excepción de tipo **`ValueError`**). Esto se ha hecho de forma intencionada para enfatizar que, aunque muchos formatos binarios incluyen elementos basados en caracteres ASCII y pueden ser manipulados mediante algunas técnicas de procesado de textos, este no es el caso general para los datos binarios (aplicar algoritmos pensados para proceso de textos a datos binarios que no se compatibles con ASCII normalmente corromperán dichos datos).

Además de con literales, se pueden crear objetos de tipo *byte* de las siguientes maneras:

- Un secuencia de una longitud especificada rellena con ceros: `bytes(10)`
- A partir de un iterable de números enteros: `bytes(range(20))`
- Copiando datos binarios ya existentes mediante el protocolo *buffer*: `bytes(obj)`

Véase además la función incorporada bytes.

Como dos dígitos hexadecimales se corresponden exactamente con un byte, suelen usase números hexadecimales para describir datos binarios. Por ello, los objetos de tipo *byte* disponen de un método adicional para leer datos en ese formato:

classmethod fromhex(*string*, */*)

Este método de clase de **`bytes`** retorna un objeto binario, decodificado a partir de la cadena suministrada como parámetro. La cadena de caracteres debe consistir en dos dígitos hexadecimales por cada byte, ignorándose los caracteres ASCII de espacio en blanco, si los hubiera.

```
>>> bytes.fromhex('2Ef0 F1f2  ')
b'.\xf0\xf1\xf2'
```

Distinto en la versión 3.7: El método **`bytes.fromhex()`** ignora ahora todos los caracteres ASCII de espacio en blanco, no solo el carácter espacio.

Distinto en la versión 3.14: **`bytes.fromhex()`** now accepts ASCII **`bytes`** and bytes-like objects as input.

Existe una función que realiza la operación inversa, es decir, transforma un objeto binario en una representación textual usando hexadecimal.

hex(*, *bytes_per_sep=1*)
hex(*sep*, *bytes_per_sep=1*)

Retorna una cadena de caracteres que contiene dos dígitos hexadecimales por cada byte de la instancia.

```
>>> b'\xf0\xf1\xf2'.hex()
'f0f1f2'
```

Si quieres que la cadena en hexadecimal sea más fácil de leer, se puede especificar un único carácter separador con el parámetro *sep* para que se añada a la salida. Por defecto, este separador se incluirá entre cada byte. Un segundo parámetro opcional, *bytes_per_sep*, controla los espacios. Valores positivos calculan la posición del separador desde la derecha, los negativos lo hacen desde la izquierda.

```
>>> value = b'\xf0\xf1\xf2'
>>> value.hex('-')
'f0-f1-f2'
>>> value.hex('_', 2)
'f0_f1f2'
>>> b'UUDDLRLRAB'.hex(' ', -4)
'55554444 4c524c52 4142'
```

Added in version 3.5.

Distinto en la versión 3.8: El método **`bytes.hex()`** ahora soporta los parámetros opcionales *sep* y *bytes_per_sep*, que permiten insertar separadores entre los bytes de la cadena de salida.

Como los objetos de tipo *bytes* son secuencias de números enteros (similares a tuplas), para un objeto binario *b*, `b[0]` retorna un entero, mientras que `b[0:1]` retorna un objeto de tipo *bytes* de longitud 1. (Mientras que las cadenas de caracteres siempre retornan una cadena de longitud 1, ya sea accediendo por índice o mediante una operación de segmentado).

La representación de los objetos tipo *bytes* usa el formato literal (`b'...'`) ya que es, por lo general, más útil que, digamos, `bytes([46, 46, 46])`. Siempre se puede convertir un objeto binario en una lista de enteros usando `list(b)`.

Objetos de tipo *Bytearray*

Los objetos de tipo **`bytearray`** son versiones mutables de los objetos de tipo **`bytes`**.

class bytearray(*source=b''*)

class bytearray(*source, encoding, errors='strict'*)

No existe una sintaxis específica para crear objetos de tipo *bytearray*, hay que crearlos siempre llamando a su constructor:

- Creando una secuencia vacía: `bytearray()`
- Creando una instancia de una longitud determinada, rellena con ceros: `bytearray(10)`
- A partir de un iterable de números enteros: `bytearray(range(20))`
- Copiando datos binarios ya existentes mediante el protocolo *buffer*: `bytearray(b'Hi!')`

Como los objetos *bytearray* son mutables, soportan todas las operaciones aplicables a tipos mutables, además de las operaciones propias de los *bytearrays* descritas en Operaciones de bytes y bytearray.

Véase también la función incorporada bytearray.

Como dos dígitos hexadecimales se corresponden exactamente con un byte, suelen usase números hexadecimales para describir datos binarios. Por ello, los objetos de tipo *bytearray* disponen de un método de clase adicional para leer datos en ese formato:

classmethod fromhex(*string*, */*)

Este método de clase de `bytearray` retorna un objeto *bytearray*, decodificado a partir de la cadena suministrada como parámetro. La cadena de caracteres debe consistir en dos dígitos hexadecimales por cada byte, ignorándose los caracteres ASCII de espacio en blanco, si los hubiera.

```
>>> bytearray.fromhex('2Ef0 F1f2  ')
bytearray(b'.\xf0\xf1\xf2')
```

Distinto en la versión 3.7: El método `bytearray.fromhex()` ignora ahora todos los caracteres ASCII de espacio en blanco, no solo el carácter espacio.

Distinto en la versión 3.14: **`bytearray.fromhex()`** now accepts ASCII `bytes` and bytes-like objects as input.

Existe una función que realiza la operación inversa, es decir, transforma un objeto *bytearray* en una representación textual usando hexadecimal.

hex(***, *bytes_per_sep=1*)
hex(*sep*, *bytes_per_sep=1*)

Retorna una cadena de caracteres que contiene dos dígitos hexadecimales por cada byte de la instancia.

```
>>> bytearray(b'\xf0\xf1\xf2').hex()
'f0f1f2'
```

Added in version 3.5.

Distinto en la versión 3.8: De forma similar a `bytes.hex()`, `bytearray.hex()` soporta ahora los parámetros opcionales *sep* y *bytes_per_sep* para insertar separadores entre los bytes en la cadena hexadecimal de salida.

resize(*size*, */*)

Resize the **`bytearray`** to contain *size* bytes. *size* must be greater than or equal to 0.

If the **`bytearray`** needs to shrink, bytes beyond *size* are truncated.

If the **`bytearray`** needs to grow, all new bytes, those beyond *size*, will be set to null bytes.

This is equivalent to:

```
>>> def resize(ba, size):
...     if len(ba) > size:
...         del ba[size:]
...     else:
...         ba += b'\0' * (size - len(ba))
```

Examples:

```
>>> shrink = bytearray(b'abc')
>>> shrink.resize(1)
>>> (shrink, len(shrink))
(bytearray(b'a'), 1)
>>> grow = bytearray(b'abc')
>>> grow.resize(5)
>>> (grow, len(grow))
(bytearray(b'abc\x00\x00'), 5)
```

Added in version 3.14.

Como los objetos de tipo *bytearray* son secuencias de números enteros (similares a listas), para un objeto *bytearray b*, `b[0]` retorna un entero, mientras que `b[0:1]` retorna un objeto de tipo *bytearray* de longitud 1. (Mientras que las cadenas de caracteres siempre retornan una cadena de longitud 1, ya sea accediendo por índice o mediante una operación de segmentado).

La representación de los objetos tipo *bytearray* usa el formato literal (`bytearray(b'...')`) ya que es, por lo general, más útil que, digamos, `bytearray([46, 46, 46])`. Siempre se puede convertir un objeto *bytearray* en una lista de enteros usando `list(b)`.

Ver también

For detailed information on thread-safety guarantees for **`bytearray`** objects, see Thread safety for bytearray objects.

Operaciones de *bytes* y *bytearray*

Ambos tipos, *bytes* y *bytearray* soportan las operaciones comunes de las secuencias. Los operadores no funcionan solo con operandos del mismo tipo, sino con cualquier objeto tipo binario. Gracias a esta flexibilidad, estos tipos pueden combinarse libremente en expresiones sin que se produzcan errores. Sin embargo, el tipo del valor resultante puede depender del orden de los operandos.

Nota

Los métodos de objetos de tipo *bytes* y *bytesarray* no aceptan cadenas de caracteres como parámetros, de la misma manera que los métodos de las cadenas tampoco aceptan *bytes* como parámetros. Por ejemplo, debes escribir:

```
a = "abc"
b = a.replace("a", "f")
```

y:

```
a = b"abc"
b = a.replace(b"a", b"f")
```

Algunas operaciones de *bytes* y *bytearrays* asumen el uso de formatos binarios compatibles ASCII, y por tanto deben ser evitadas cuando trabajamos con datos binarios arbitrarios. Estas restricciones se explican a continuación.

Nota

Usar estas operaciones basadas en ASCII para manipular datos binarios que no se almacenan en un formato basado en ASCII pueden producir corrupción de datos.

Los siguientes métodos de *bytes* y *bytearrays* pueden ser usados con datos en formatos binarios arbitrarios.

bytes.count(*sub*[, *start*[, *end*]])
bytearray.count(*sub*[, *start*[, *end*]])

Retorna el número de secuencias no solapadas de la subsecuencia *sub* en el rango [*start*, *end*]. Los parámetros opcionales *start* y *end* se interpretan como en las operaciones de segmentado.

La subsecuencia a buscar puede ser cualquier objeto tipo binario o un número entero entre 0 y 255.

Si *sub* está vacío, devuelve el número de porciones vacías entre caracteres, que es la longitud del objeto de bytes más uno.

Distinto en la versión 3.3: También acepta como subsecuencia un número entero entre 0 y 255.

bytes.removeprefix(*prefix*, /)
bytearray.removeprefix(*prefix*, /)

Si los datos binarios comienzan con la cadena *prefix*, retorna `bytes[len(prefix):]`. De otra manera, retorna una copia de los datos binarios originales:

```
>>> b'TestHook'.removeprefix(b'Test')
b'Hook'
>>> b'BaseTestCase'.removeprefix(b'Test')
b'BaseTestCase'
```

El argumento *prefix* puede ser cualquier objeto tipo binario.

Nota

La versión *bytearray* de este método *no* opera in situ -

siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

Added in version 3.9.

bytes.removesuffix(*suffix*, */*)
bytearray.removesuffix(*suffix*, */*)

Si los datos binarios terminan con la cadena expresada en *suffix* y el argumento *suffix* no está vacío, retorna `bytes[:-len(suffix)]`. De otra manera, retorna una copia de los datos binarios originales:

```
>>> b'MiscTests'.removesuffix(b'Tests')
b'Misc'
>>> b'TmpDirMixin'.removesuffix(b'Tests')
b'TmpDirMixin'
```

El argumento *suffix* puede ser cualquier objeto tipo binario.

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

Added in version 3.9.

bytes.decode(*encoding='utf-8'*, *errors='strict'*)
bytearray.decode(*encoding='utf-8'*, *errors='strict'*)

Devuelve los bytes decodificados a un **`str`**.

El valor predeterminado de *encoding* es `'utf-8'`; consulte Codificaciones estándar para conocer los valores posibles.

errors controla cómo se manejan los errores de decodificación. Si se usa `'strict'` (el valor predeterminado), se genera una excepción **`UnicodeError`**. Otros valores posibles son `'ignore'`, `'replace'` y cualquier otro nombre registrado mediante `codecs.register_error()`. Consulte Manejadores de errores para obtener más detalles.

Por razones de rendimiento, no se verifica la validez del valor de *errors* a menos que realmente se produzca un error de decodificación, se habilite Modo de desarrollo de Python o se utilice debug build.

Nota

Pasar el argumento *encoding* a `str` permite decodificar cualquier bytes-like object directamente, sin necesidad de crear un objeto temporal `bytes` o `bytearray`.

Distinto en la versión 3.1: Añadido soporte para poder usar parámetros por nombre.

Distinto en la versión 3.9: El valor del argumento *errors* ahora se comprueba en Modo de desarrollo de Python y en debug mode.

bytes.endswith(*suffix*[, *start*[, *end*]])

bytearray.endswith(*suffix*[, *start*[, *end*]])

Retorna `True` si los datos binarios acaban con el valor indicado por *suffix*, en caso contrario retorna `False`. El valor de *suffix* puede ser también una tupla de sufijos para buscar. Con el parámetro opcional *start*, la comparación empieza a partir de esa posición. Si se especifica el parámetro opcional *end*, la comparación termina en esa posición.

El sufijo (o sufijos) a buscar puede ser cualquier objeto tipo binario.

bytes.find(*sub*[, *start*[, *end*]])

bytearray.find(*sub*[, *start*[, *end*]])

Retorna el mínimo índice dentro de los datos donde se ha encontrado la subsecuencia *sub*, de forma que *sub* está contenida en el segmento `s[start:end]`. Los parámetros opcionales *start* y *end* se interpretan como en las operaciones de segmentado. Retorna `-1` si no se puede encontrar *sub*.

La subsecuencia a buscar puede ser cualquier objeto tipo binario o un número entero entre 0 y 255.

Nota

El método **`find()`** se debe usar solo si se necesita saber la posición de *sub*. Si solo se necesita comprobar si *sub* es una parte de *s*, es mejor usar el operador **`in`**:

```
>>> b'Py' in b'Python'
True
```

Distinto en la versión 3.3: También acepta como subsecuencia un número entero entre 0 y 255.

bytes.index(*sub*[, *start*[, *end*]])

bytearray.index(*sub*[, *start*[, *end*]])

Como **`find()`**, pero lanza una excepción de tipo **`ValueError`** si no se encuentra la subsecuencia a buscar.

La subsecuencia a buscar puede ser cualquier objeto tipo binario o un número entero entre 0 y 255.

Distinto en la versión 3.3: También acepta como subsecuencia un número entero entre 0 y 255.

bytes.join(*iterable*, */*)

bytearray.join(*iterable*, */*)

Retorna un objeto de tipo *bytes* o *bytearray* que es la concatenación de las secuencias binarias en *iterable*. Si alguno de los objetos de la secuencia no es un objeto tipo binario se lanza la excepción **`TypeError`**, incluso si son cadenas de caracteres (objetos **`str`**). El separador entre los distintos elementos es el contenido del objeto *bytes* o *bytearray* usando para invocar el método.

static bytes.maketrans(*from*, *to*, */*)

static bytearray.maketrans(*from*, *to*, */*)

Este método estático retorna una tabla de traducción apta para ser usada por el método `bytes.translate()`, que mapea cada carácter en *from* en la misma posición en *to*; tanto *from* como *to* deben ser objetos tipo binario y deben tener la misma longitud.

Added in version 3.1.

bytes.partition(*sep*, */*)

bytearray.partition(*sep*, */*)

Divide la secuencia en la primera ocurrencia de *sep*, y retorna una tupla de tres elementos que contiene la parte antes del separador, el separador en sí o una copia de tipo *bytearray* y la parte después del separador. Si no se encuentra el separador, retorna una tupla de tres elementos, con la primera posición ocupada por la secuencia original, y las dos posiciones siguientes rellenas con objetos *bytes* o *bytearray* vacíos.

El separador a buscar puede ser cualquier objeto tipo binario.

bytes.replace(*old*, *new*, *count=-1*, */*)

bytearray.replace(*old*, *new*, *count=-1*, */*)

Retorna una copia de la secuencia con todas las ocurrencias de *old* sustituidas por *new*. Si se utiliza el parámetro *count*, solo se cambian las primeras *count* ocurrencias.

La subsecuencia a buscar y su reemplazo puede ser cualquier objeto tipo binario.

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

bytes.rfind(*sub*[, *start*[, *end*]])

bytearray.rfind(*sub*[, *start*[, *end*]])

Retorna el mayor índice dentro de la secuencia *s* donde se puede encontrar *sub*, estando *sub* incluida en `s[start:end]`. Los parámetros opcionales *start* y *end* se interpretan igual que en las operaciones de segmentado. Retorna `-1` si no se encuentra *sub*.

La subsecuencia a buscar puede ser cualquier objeto tipo binario o un número entero entre 0 y 255.

Distinto en la versión 3.3: También acepta como subsecuencia un número entero entre 0 y 255.

bytes.rindex(*sub*[, *start*[, *end*]])

bytearray.rindex(*sub*[, *start*[, *end*]])

Como el método **`rfind()`**, pero lanza la excepción **`ValueError`** si no se encuentra *sub*.

La subsecuencia a buscar puede ser cualquier objeto tipo binario o un número entero entre 0 y 255.

Distinto en la versión 3.3: También acepta como subsecuencia un número entero entre 0 y 255.

bytes.rpartition(*sep*, /)

bytearray.rpartition(*sep*, /)

Divide la secuencia en la primera ocurrencia de *sep*, y retorna una tupla de tres elementos que contiene la parte antes del separador, el separador en sí o una copia de tipo *bytearray* y la parte después del separador. Si no se encuentra el separador, retorna una tupla de tres elementos, con las dos primeras posiciones rellenas con objetos *bytes* o *bytearray* vacíos, y la tercera posición ocupada por la secuencia original.

El separador a buscar puede ser cualquier objeto tipo binario.

bytes.startswith(*prefix*[, *start*[, *end*]])
bytearray.startswith(*prefix*[, *start*[, *end*]])

Retorna `True` si los datos binarios empiezan con el valor indicado por *prefix*, en caso contrario retorna `False`. El valor de *prefix* puede ser también una tupla de prefijos para buscar. Con el parámetro opcional *start*, la comparación empieza a partir de esa posición. Si se especifica el parámetro opcional *end*, la comparación termina en esa posición.

El prefijo (o prefijos) a buscar puede ser cualquier objeto tipo binario.

bytes.translate(*table*, /, *delete=b''*)
bytearray.translate(*table*, /, *delete=b''*)

Retorna una copia del objeto *bytes* o *bytearray* donde todas las ocurrencias de bytes especificados en el parámetro *delete* han sido borrados, y el resto han sido mapeados a través de la tabla de traducción indicada, que debe ser un objeto de tipo *bytes* con una longitud de 256 elementos.

Puedes usar el método **`bytes.maketrans()`** para crear la tabla de traducción.

Se puede ajustar el parámetro *table* a `None` para conseguir una traducción que solo borra caracteres:

```
>>> b'read this short text'.translate(None, b'aeiou
b'rd ths shrt txt'
```

Distinto en la versión 3.6: El parámetro *delete* se puede ahora especificar por nombre.

Los siguientes métodos de los objetos *bytes* y *bytearray* presentan un comportamiento por defecto que asume el uso de formatos binarios compatibles con ASCII, pero aun así pueden ser usados con datos binarios arbitrarios usando los parámetros apropiados. Nótese que todos los métodos de *bytearray* en esta sección nunca operan in situ, sino que siempre retornan objetos nuevos.

bytes.center(*width*, *fillbyte=b' '*, */*)

bytearray.center(*width*, *fillbyte=b' '*, */*)

Retorna una copia del objeto centrado en una secuencia de longitud *width*. El relleno se realiza usando el valor definido en el parámetro *fillbyte* (por defecto, el carácter espacio en ASCII). Para los objetos de tipo `bytes`, se retorna la secuencia original intacta si *width* es menor o igual que `len(s)`.

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

bytes.ljust(*width*, *fillbyte=b' '*, */*)

bytearray.ljust(*width*, *fillbyte=b' '*, */*)

Retorna una copia del objeto justificado por la izquierda en una secuencia de longitud *width*. El relleno se realiza usando el valor definido en el parámetro *fillbyte* (por defecto, el carácter espacio en ASCII). Para los objetos de tipo `bytes`, se retorna la secuencia original intacta si *width* es menor o igual que `len(s)`.

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

bytes.lstrip(*bytes=None*, */*)

bytearray.lstrip(*bytes=None*, */*)

Return a copy of the sequence with specified leading bytes removed. The *bytes* argument is a binary sequence specifying the set of byte values to be removed. If omitted or `None`, the *bytes* argument defaults to removing ASCII whitespace. The *bytes* argument is not a prefix; rather, all combinations of its values are stripped:

```
>>> b'   spacious   '.lstrip()
b'spacious   '
>>> b'www.example.com'.lstrip(b'cmowz.')
b'example.com'
```

La secuencia binaria que especifica el conjunto bytes a ser eliminados puede ser cualquier objeto de tipo binario. Véase **`removeprefix()`** para un método que removerá una única cadena de prefijo en lugar de todas las ocurrencias dentro de un set de caracteres. Por ejemplo:

```
>>> b'Arthur: three!'.lstrip(b'Arthur: ')
b'ee!'
>>> b'Arthur: three!'.removeprefix(b'Arthur: ')
b'three!'
```

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

bytes.rjust(*width*, *fillbyte=b' '*, */*)
bytearray.rjust(*width*, *fillbyte=b' '*, */*)

Retorna una copia del objeto justificado por la derecha en una secuencia de longitud *width*. El relleno se realiza usando el valor definido en el parámetro *fillbyte* (por defecto, el carácter espacio en ASCII). Para los objetos de tipo **`bytes`**, se retorna la secuencia original intacta si *width* es menor o igual que `len(s)`.

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

bytes.rsplit(*sep=None*, *maxsplit=-1*)

bytearray.rsplit(*sep=None*, *maxsplit=-1*)

Divide una secuencia binaria en subsecuencias del mismo tipo, usando como separador el valor de *sep*. Si se utiliza el parámetro *maxsplit*, se realizan como máximo *maxsplit* divisiones, retornando los que están más a la derecha. Si no se especifica *sep* o se pasa con valor `None`, se usa como separador el carácter espacio en ASCII. Si no contamos la diferencia de empezar las divisiones desde la derecha, el comportamiento de este método **`rsplit()`** es equivalente al de **`split()`**, que se describe con detalle más adelante.

bytes.rstrip(*bytes=None*, */*)

bytearray.rstrip(*bytes=None*, */*)

Return a copy of the sequence with specified trailing bytes removed. The *bytes* argument is a binary sequence specifying the set of byte values to be removed. If omitted or `None`, the *bytes* argument defaults to removing ASCII whitespace. The *bytes* argument is not a suffix; rather, all combinations of its values are stripped:

```
>>> b'   spacious   '.rstrip()
b'   spacious'
>>> b'mississippi'.rstrip(b'ipz')
b'mississ'
```

La secuencia binaria que especifica el conjunto bytes a ser eliminados puede ser cualquier objeto de tipo binario. Véase **`removesuffix()`** para un método que removerá una única cadena de sufijo en lugar de todas las ocurrencias dentro de un set de caracteres. Por ejemplo:

```
>>> b'Monty Python'.rstrip(b' Python')
b'M'
>>> b'Monty Python'.removesuffix(b' Python')
b'Monty'
```

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

bytes.split(*sep=None*, *maxsplit=-1*)
bytearray.split(*sep=None*, *maxsplit=-1*)

Divide una secuencia binaria en subsecuencias del mismo tipo, usando como separador el valor de *sep*. Si se utiliza el parámetro *maxsplit* y es un número positivo, se realizan como máximo *maxsplit* divisiones (resultando en una secuencia de como mucho `maxsplit+1` elementos). Si no se especifica *maxsplit* o se pasa `'-1`, no hay límite al número de divisiones (se hacen todas las posibles divisiones).

Si se proporciona *sep*, los delimitadores consecutivos no se agrupan y se consideran que delimitan subsecuencias vacías (por ejemplo, `b'1,,2'.split(b',')` devuelve `[b'1', b'', b'2']`). El argumento *sep* puede consistir en una secuencia multibyte como un único delimitador. Dividir una secuencia vacía con un separador especificado devuelve `[b'']` o `[bytearray(b'')]` según el tipo de objeto que se esté dividiendo. El argumento *sep* puede ser cualquier bytes-like object.

Por ejemplo:

```
>>> b'1,2,3'.split(b',')
[b'1', b'2', b'3']
>>> b'1,2,3'.split(b',', maxsplit=1)
[b'1', b'2,3']
>>> b'1,2,,3,'.split(b',')
[b'1', b'2', b'', b'3', b'']
>>> b'1<>2<>3<4'.split(b'<>')
[b'1', b'2', b'3<4']
```

Si no se especifica *sep* o es `None`, se usa un algoritmo de división diferente: secuencias consecutivas de caracteres de espacio en ASCII se consideran como un único separador, y el resultado no contendrá cadenas vacías ni al principio ni al final de la lista, aunque la cadena original tuviera espacios en blanco al principio o al final. En consecuencia, dividir una secuencia vacía o que solo contenga espacios en blanco usando `None` como separador siempre retornará una lista vacía `[]`.

Por ejemplo:

```
>>> b'1 2 3'.split()
[b'1', b'2', b'3']
>>> b'1 2 3'.split(maxsplit=1)
[b'1', b'2 3']
>>> b'   1   2   3   '.split()
[b'1', b'2', b'3']
```

bytes.strip(*bytes=None*, */*)
bytearray.strip(*bytes=None*, */*)

Return a copy of the sequence with specified leading and trailing bytes removed. The *bytes* argument is a binary sequence specifying the set of byte values to be removed. If omitted or `None`, the *bytes* argument defaults to removing ASCII whitespace. The *bytes* argument is not a prefix or suffix; rather, all combinations of its values are stripped:

```
>>> b'   spacious   '.strip()
b'spacious'
>>> b'www.example.com'.strip(b'cmowz.')
b'example'
```

La secuencia binaria de bytes a eliminar debe ser un objeto tipo binario.

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

Los siguientes métodos de los objetos *bytes* y *bytearray* asumen el uso de formatos binarios compatibles con ASCII, y no deben ser usados con datos binarios arbitrarios. Nótese que todos los métodos de *bytearray* en esta sección nunca operan in situ, sino que siempre retornan objetos nuevos.

bytes.capitalize()

bytearray.capitalize()

Retorna una copia de la secuencia con cada byte interpretado como un carácter ASCII, y el primer byte en mayúsculas y el resto en minúsculas. Los valores que no sean ASCII no se ven modificados.

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

bytes.expandtabs(*tabsize=8*)

bytearray.expandtabs(*tabsize=8*)

Retorna una copia de la secuencia, con todos los caracteres ASCII *tab* reemplazados por uno o más espacios ASCII, dependiendo de la columna actual y del tamaño definido para el tabulador. Las posiciones de tabulación ocurren cada *tabsize* caracteres (siendo el valor por defecto de *tabsize* 8, lo que produce las posiciones de tabulación 0, 8, 16,…). Para expandir la secuencia, la columna actual se pone a cero y se va examinando byte a byte. Si se encuentra un tabulador, (`b'\t'`), se insertan uno o más espacios hasta que sea igual a la siguiente posición de tabulación. (El carácter tabulador en sí es descartado). Si el byte es un indicador de salto de línea (`b'\n'`) o de retorno (`b'\r'`), se copia y el valor de columna actual se vuelve a poner a cero. Cualquier otro carácter es copiado sin cambios y hace que el contador de columna se incremente en 1, sin tener en cuenta como se representa impreso el byte:

```
>>> b'01\t012\t0123\t01234'.expandtabs()
b'01      012     0123    01234'
>>> b'01\t012\t0123\t01234'.expandtabs(4)
b'01  012 0123    01234'
```

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

bytes.isalnum()
bytearray.isalnum()

Retorna `True` si todos los bytes de la secuencia son caracteres alfabéticos ASCII o caracteres decimales ASCII y la secuencia no está vacía, en cualquier otro caso retorna `False`. Los caracteres alfabéticos ASCII son los bytes incluidos en la secuencia `b'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVW` Los caracteres decimales ASCII son los bytes incluidos en la secuencia `b'0123456789'`.

Por ejemplo:

```
>>> b'ABCabc1'.isalnum()
True
>>> b'ABC abc1'.isalnum()
False
```

bytes.isalpha()
bytearray.isalpha()

Retorna `True` si todos los bytes de la secuencia son caracteres alfabéticos ASCII y la secuencia no está vacía, en cualquier otro caso retorna `False`. Los caracteres alfabéticos ASCII son los bytes incluidos en la secuencia `b'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVW`

Por ejemplo:

```
>>> b'ABCabc'.isalpha()
True
>>> b'ABCabc1'.isalpha()
False
```

bytes.isascii()
bytearray.isascii()

Retorna `True` si la secuencia está vacía o si todos los bytes de la secuencia son caracteres ASCII, en cualquier otro caso retorna `False`. Los caracteres ASCII son los bytes incluidos en el rango 0-0x7F.

Added in version 3.7.

bytes.isdigit()
bytearray.isdigit()

Retorna `True` si todos los bytes de la secuencia son caracteres decimales ASCII y la secuencia no está vacía, en cualquier otro caso retorna `False`. Los caracteres decimales ASCII son los bytes incluidos en la secuencia `b'0123456789'`.

Por ejemplo:

```
>>> b'1234'.isdigit()
True
>>> b'1.23'.isdigit()
False
```

bytes.islower()
bytearray.islower()

Retorna `True` si hay al menos un carácter ASCII en minúsculas, y no hay ningún carácter ASCII en mayúsculas, en cualquier otro caso retorna `False`.

Por ejemplo:

```
>>> b'hello world'.islower()
True
>>> b'Hello world'.islower()
False
```

Los caracteres ASCII en minúsculas son los bytes incluidos en la secuencia `b'abcdefghijklmnopqrstuvwxyz'`. Los caracteres ASCII en mayúsculas son los bytes en la secuencia `b'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`.

bytes.isspace()
bytearray.isspace()

Retorna `True` si todos los bytes de la secuencia son caracteres ASCII de espacio en blanco y la secuencia no está vacía, en cualquier otro caso retorna `False`. Los caracteres de espacio en blanco ASCII son los bytes incluidos en la secuencia `b' \t\n\r\x0b\f'` (espacio, tabulador, nueva línea, retorno de carro, tabulador vertical y avance de página).

bytes.istitle()
bytearray.istitle()

Retorna `True` si la secuencia ASCII está en forma de título, y la secuencia no está vacía, en cualquier otro caso retorna `False`. Véase el método **`bytes.title()`** para más detalles en la definición de «En forma de título».

Por ejemplo:

```
>>> b'Hello World'.istitle()
True
>>> b'Hello world'.istitle()
False
```

bytes.isupper()
bytearray.isupper()

Retorna `True` si hay al menos un carácter ASCII en mayúsculas, y no hay ningún carácter ASCII en minúsculas, en cualquier otro caso retorna `False`.

Por ejemplo:

```
>>> b'HELLO WORLD'.isupper()
True
>>> b'Hello world'.isupper()
False
```

Los caracteres ASCII en minúsculas son los bytes incluidos en la secuencia `b'abcdefghijklmnopqrstuvwxyz'`. Los caracteres ASCII en mayúsculas son los bytes en la secuencia `b'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`.

bytes.lower()
bytearray.lower()

Retorna una copia de la secuencia con todos los caracteres ASCII en mayúsculas sustituidos por su versión correspondiente en minúsculas.

Por ejemplo:

```
>>> b'Hello World'.lower()
b'hello world'
```

Los caracteres ASCII en minúsculas son los bytes incluidos en la secuencia `b'abcdefghijklmnopqrstuvwxyz'`. Los caracteres ASCII en mayúsculas son los bytes en la secuencia `b'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`.

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

bytes.splitlines(*keepends=False*)
bytearray.splitlines(*keepends=False*)

Retorna una lista de las líneas en la secuencia binaria, usando como separadores los saltos de líneas universales. Los caracteres usados como separadores no se incluyen en la lista de resultados a no ser que se pase el parámetro *keepends* a `True`.

Por ejemplo:

```
>>> b'ab c\n\nde fg\rkl\r\n'.splitlines()
[b'ab c', b'', b'de fg', b'kl']
>>> b'ab c\n\nde fg\rkl\r\n'.splitlines(keepends=Tr
[b'ab c\n', b'\n', b'de fg\r', b'kl\r\n']
```

Al contrario que el método **`split()`**, cuando se especifica una cadena delimitadora con el parámetro *sep*, este método retorna una lista vacía para la cadena vacía, y un carácter de salto de línea al final de la secuencia no resulta en una línea extra:

```
>>> b"".split(b'\n'), b"Two lines\n".split(b'\n')
([b''], [b'Two lines', b''])
>>> b"".splitlines(), b"One line\n".splitlines()
([], [b'One line'])
```

bytes.swapcase()
bytearray.swapcase()

Retorna una copia de la secuencia con todos los caracteres ASCII en minúsculas sustituidos por su versión correspondiente en mayúsculas, y viceversa.

Por ejemplo:

```
>>> b'Hello World'.swapcase()
b'hELLO wORLD'
```

Los caracteres ASCII en minúsculas son los bytes incluidos en la secuencia `b'abcdefghijklmnopqrstuvwxyz'`. Los caracteres ASCII en mayúsculas son los bytes en la secuencia `b'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`.

A diferencia de `str.swapcase()`, siempre ocurre lo mismo con `bin.swapcase().swapcase() == bin` para las versiones binarias. Las conversiones de mayúsculas y minúsculas son simétricas en ASCII, aunque esto no suele ser así para puntos de código Unicode arbitrarios.

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

bytes.title()

bytearray.title()

Retorna una versión en forma de título de la secuencia binaria, con la primera letra de cada palabra en mayúsculas y el resto en minúsculas. Los valores de bytes sin mayúsculas y minúsculas se dejan sin modificar.

Por ejemplo:

```
>>> b'Hello world'.title()
b'Hello World'
```

Los caracteres ASCII en minúsculas son los bytes incluidos en la secuencia `b'abcdefghijklmnopqrstuvwxyz'`. Los caracteres ASCII en mayúsculas son los bytes en la secuencia `b'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`. El resto de los caracteres no presentan diferencias entre mayúsculas y minúsculas.

El algoritmo usa una definición sencilla e independiente del lenguaje, consistente en considerar una palabra como un grupo de letras consecutivas. Esta definición funciona en varios entornos, pero implica que, por ejemplo en inglés, los apóstrofos en las contracciones y en los posesivos constituyen una separación entre palabras, que puede que no sea el efecto deseado:

```
>>> b"they're bill's friends from the UK".title()
b"They'Re Bill'S Friends From The Uk"
```

Se puede solucionar parcialmente el problema de los apóstrofos usando expresiones regulares:

```
>>> import re
>>> def titlecase(s):
...     return re.sub(rb"[A-Za-z]+('[A-Za-z]+)?",
...                   lambda mo: mo.group(0)[0:1].u
...                              mo.group(0)[1:].lo
...                   s)
...
>>> titlecase(b"they're bill's friends.")
b"They're Bill's Friends."
```

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

bytes.upper()

bytearray.upper()

Retorna una copia de la secuencia con todos los caracteres ASCII en minúsculas sustituidos por su versión correspondiente en mayúsculas.

Por ejemplo:

```
>>> b'Hello World'.upper()
b'HELLO WORLD'
```

Los caracteres ASCII en minúsculas son los bytes incluidos en la secuencia `b'abcdefghijklmnopqrstuvwxyz'`. Los caracteres ASCII en mayúsculas son los bytes en la secuencia `b'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`.

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

bytes.zfill(*width*, */*)
bytearray.zfill(*width*, */*)

Retorna una copia de la secuencia rellenada por la izquierda con los caracteres ASCII `b'0'` necesarios para conseguir una cadena de longitud *width*. El carácter prefijo de signo (`b'+'`/ `b'-'`) se gestiona insertando el relleno *después* del carácter de signo en vez de antes. Para objetos **`bytes`**, se retorna la secuencia original si *width* es menor o igual que `len(s)`.

Por ejemplo:

```
>>> b"42".zfill(5)
b'00042'
>>> b"-42".zfill(5)
b'-0042'
```

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

Usando el formateo tipo `printf` con bytes

Nota

Las operaciones de formateo explicadas aquí tienen una serie de peculiaridades que conducen a ciertos errores comunes (como fallar al representar tuplas y diccionarios correctamente). Si el valor a representar es una tupla o un diccionario, hay que envolverlos en una tupla.

Los objetos binarios (`bytes/bytearray`) tienen una operación incorporada: el operador `%` (módulo). Esta operación se conoce también como operador de *formateo* o de *interpolación*. Dada la expresión `formato % valores` (donde *formato* es un objeto binario), las especificaciones de conversión indicadas en la cadena con el símbolo `%` son reemplazadas por cero o más elementos de *valores*. El efecto es similar a usar la función del lenguaje C **`sprintf()`**.

Si *formato* tiene un único marcador, *valores* puede ser un objeto sencillo, no una tupla. [5] En caso contrario, *valores* debe ser una tupla con exactamente el mismo número de elementos que marcadores usados en el objeto binario, o un único objeto de tipo mapa (por ejemplo, un diccionario).

Un especificador de conversión consiste en dos o más caracteres y tiene los siguientes componentes, que deben aparecer en el siguiente orden:

1. El carácter `'%'`, que identifica el inicio del marcador.
2. Una clave de mapeo (opcional), consistente en una secuencia de caracteres entre paréntesis, como por ejemplo, `(somename)`.
3. Indicador de conversión (opcional), que afecta el resultado de ciertas conversiones de tipos.
4. Valor de ancho mínimo (opcional). Si se especifica un `'*'` (asterisco), el ancho real se lee del siguiente elemento de la tupla *valores*, y el objeto a convertir viene después del ancho mínimo, con un indicador de precisión opcional.
5. Precisión (opcional), en la forma `'.'` (punto) seguido de la precisión. Si se especifica un `'*'` (asterisco), el valor de precisión real se lee del siguiente elemento de la tupla *valores*, y el valor a convertir viene después de la precisión.
6. Modificador de longitud (opcional).
7. Tipo de conversión.

Cuando el argumento derecho es un diccionario (o cualquier otro objeto de tipo mapa), los marcadores en el objeto binario *deben* incluir un valor de clave entre paréntesis, inmediatamente después del carácter `'%'`. El valor de la clave se usa para seleccionar el valor a formatear desde el mapa. Por ejemplo:

```
>>> print(b'%(language)s has %(number)03d quote types.'
...       {b'language': b"Python", b"number": 2})
b'Python has 002 quote types.'
```

En este caso, no se puede usar el especificador `*` en la cadena de formato (dado que requiere una lista secuencial de parámetros).

Los indicadores de conversión son:

Flag	**Significado**
`'#'`	El valor a convertir usara la «forma alternativa» (que se definirá más adelante)
`'0'`	La conversión rellena con ceros por la izquierda para valores numéricos.
`'-'`	El valor convertido se ajusta a la izquierda (sobreescribe la conversión `'0'` si se especifican los dos)
`' '`	(Un espacio) Se debe añadir un espacio en blanco antes de un número positivo (o una cadena vacía) si se usa una conversión con signo.
`'+'`	Un carácter signo (`'+'` o `'-'`) precede a la conversión (sobreescribe el indicador de «espacio»)

Puede estar presente un modificador de longitud (`h`, `l` o `L`), pero se ignora y no es necesario para Python – por lo que, por ejemplo, la salida de `%ld` es idéntica a `%d`.

Los tipos de conversión son:

Conversión	**Significado**	**Notas**
`'d'`	Entero decimal con signo.	
`'i'`	Entero decimal con signo.	
`'o'`	Valor octal con signo.	(1)
`'u'`	Obsoleto – es idéntico a `'d'`.	(8)
`'x'`	Hexadecimal con signo (en minúsculas).	(2)
`'X'`	Hexadecimal con signo (en mayúsculas).	(2)
`'e'`	Formato en coma flotante exponencial (en minúsculas).	(3)

`'E'`	Formato en coma flotante exponencial (en mayúsculas).	(3)
`'f'`	Formato en coma flotante decimal.	(3)
`'F'`	Formato en coma flotante decimal.	(3)
`'g'`	Formato de punto flotante. Utiliza el formato exponencial en minúsculas si el exponente es menor que -4 o igual que la precisión; en caso contrario, utiliza el formato decimal.	(4)
`'G'`	Formato de punto flotante. Utiliza el formato exponencial en mayúsculas si el exponente es menor que -4 o igual que la precisión; en caso contrario, utiliza el formato decimal.	(4)
`'c'`	Byte único (acepta números enteros o binarios de un único byte).	
`'b'`	Bytes (cualquier objeto que siga al buffer protocol o tenga **`__bytes__()`**).	(5)
`'s'`	`'s'` es un alias de `'b'` y solo debe ser usado para bases de código Python2/3.	(6)
`'a'`	Bytes (convierte cualquier objeto Python usando `repr(obj).encode('ascii','backslashreplace')`).	(5)
`'r'`	`'r'` es un alias de `'a'` y solo debe ser usado para bases de código Python2/3.	(7)
`'%'`	No se representa ningún argumento, obteniéndose en el resultado la cadena `'%'`.	

Notas:

1. La forma alternativa hace que se inserte antes del primer dígito un prefijo indicativo del formato octal (`'0o'`)

2. La forma alternativa hace que se inserte antes del primer dígito uno de los dos prefijos indicativos del formato hexadecimal `'0x'` o `'0X'` (que se use uno u otro depende de que indicador de formato se haya usado, `'x'` o `'X'`).

3. La forma alternativa hace que se incluya siempre el símbolo del punto o coma decimal, incluso si no hubiera dígitos después.

   La precisión determina el número de dígitos que vienen después del punto decimal, y por defecto es 6.

4. La forma alternativa hace que se incluya siempre el símbolo del punto o coma decimal, y los ceros a su derecha no se eliminan, como seria el caso en la forma normal.

La precisión determina el número de dígitos significativos que vienen antes y después del punto decimal, y por defecto es 6.

5. Si la precisión es `N`, la salida se trunca a `N` caracteres.
6. `b'%s'` está obsoleto, pero no se retirará durante la serie 3.x.
7. `b'%r'` está obsoleto, pero no se retirará durante la serie 3.x.
8. Véase **PEP 237** [https://peps.python.org/pep-0237/].

Nota

La versión *bytearray* de este método *no* opera in situ - siempre produce un nuevo objeto, aún si no se hubiera realizado ningún cambio.

Ver también

PEP 461 [https://peps.python.org/pep-0461/] - Añadiendo % formatea a bytes y bytearray

Added in version 3.5.

Vistas de memoria

Los objetos de tipo **`memoryview`** permiten al código Python acceder a los datos internos de objetos que soporten el protocolo buffer sin necesidad de hacer copias.

class memoryview(*object*)

Crea un **`memoryview`** que referencia *object*. La variable *object* debe soportar el protocolo buffer. Los objetos incorporados que soportan el protocolo buffer incluyen los **`bytes`** y **`bytearray`**.

La clase **`memoryview`** usa el concepto de *elemento*, que es la unidad de memoria atómica gestionada por el objeto original *object*. Para muchos tipos de datos simples como **`bytes`** y **`bytearray`**, un elemento es un único byte, pero otros tipos, como la clase **`array.array`** pueden tener elementos más grandes.

`len(view)` es igual a la longitud de **`tolist`**, que es la representación de lista anidada de la vista. Si es `view.ndim = 1`, es igual a la cantidad de elementos de la vista.

Distinto en la versión 3.12: Si es `view.ndim == 0`, `len(view)` ahora genera **`TypeError`** en lugar de devolver 1.

El atributo **`itemsize`** le dará la cantidad de bytes en un solo elemento.

Un objeto de tipo **`memoryview`** soporta operaciones de segmentado y acceso por índices a sus datos. Un segmentado unidimensional producirá una sub-vista:

```
>>> v = memoryview(b'abcefg')
>>> v[1]
98
>>> v[-1]
103
>>> v[1:4]
<memory at 0x7f3ddc9f4350>
>>> bytes(v[1:4])
b'bce'
```

Si **`format`** es uno de los especificadores de formato nativos del módulo **`struct`**, el indexado con un número entero o una tupla de números enteros también es posible, y retorna un único *elemento* con el tipo adecuado. Objetos *memoryview* unidimensionales pueden ser indexados con un entero o con una tupla de enteros. Los *memoryview* con múltiples dimensiones pueden ser indexados con tuplas de exactamente *ndim* enteros, donde *ndim* es el número de dimensiones. Vistas *memoryviews* con cero dimensiones pueden ser indexados con una tupla vacía.

Aquí hay un ejemplo con un formato que no es un byte:

```
>>> import array
>>> a = array.array('l', [-11111111, 22222222
>>> m = memoryview(a)
>>> m[0]
-11111111
>>> m[-1]
44444444
>>> m[::2].tolist()
[-11111111, -33333333]
```

Si el objeto usado para crear la vista es modificable, la vista *memoryview* soporta asignación unidimensional mediante segmentos. Sin embargo, no se permite el cambio de tamaño:

```
>>> data = bytearray(b'abcefg')
>>> v = memoryview(data)
>>> v.readonly
False
>>> v[0] = ord(b'z')
>>> data
bytearray(b'zbcefg')
>>> v[1:4] = b'123'
>>> data
bytearray(b'z123fg')
>>> v[2:3] = b'spam'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: memoryview assignment: lvalue and
>>> v[2:6] = b'spam'
>>> data
bytearray(b'z1spam')
```

Las vistas de memoria unidimensionales de tipos hashable (solo lectura) con formatos “B”, “b” o “c” también se pueden codificar en hash. El hash se define como `hash(m) == hash(m.tobytes())`:

```
>>> v = memoryview(b'abcefg')
>>> hash(v) == hash(b'abcefg')
True
>>> hash(v[2:4]) == hash(b'ce')
True
>>> hash(v[::-2]) == hash(b'abcefg'[::-2])
True
```

Distinto en la versión 3.3: Ahora es posible dividir las vistas de memoria unidimensionales. Las vistas de memoria unidimensionales con formatos “B”, “b” o “c” ahora son hashable.

Distinto en la versión 3.4: los objetos *memoryview* son registrados automáticamente con la clase **`collections.abc.Sequence`**

Distinto en la versión 3.5: los objetos *memoryviews* se pueden ahora acceder usando como índices una tupla de números enteros.

Distinto en la versión 3.14: memoryview is now a generic type.

La clase **`memoryview`** tiene varios métodos:

__eq__(*exporter*)

Un objeto *memoryview* y un exportador **PEP 3118** [https://peps.python.org/pep-3118/] son iguales si sus formas son equivalentes y todos los valores correspondientes son iguales cuando los formatos respectivos de los operandos son interpretados usando la sintaxis de **`struct`**.

Para el subconjunto de formatos de **`struct`** soportados actualmente por **`tolist()`**, `v` y `w` son iguales si `v.tolist() == w.tolist()`:

```
>>> import array
>>> a = array.array('I', [1, 2, 3, 4, 5])
>>> b = array.array('d', [1.0, 2.0, 3.0,
>>> c = array.array('b', [5, 3, 1])
>>> x = memoryview(a)
>>> y = memoryview(b)
>>> x == a == y == b
True
>>> x.tolist() == a.tolist() == y.tolist
True
>>> z = y[::-2]
>>> z == c
True
>>> z.tolist() == c.tolist()
True
```

Si cualquiera de las cadenas de formato no es soportada por el módulo **`struct`**, entonces la comparación de los objetos siempre los considerará diferentes (incluso si las cadenas de formato y el contenido del *buffer* son idénticos):

```
>>> from ctypes import BigEndianStructure
>>> class BEPoint(BigEndianStructure):
```

```
...         _fields_ = [("x", c_long), ("y",
...
>>> point = BEPoint(100, 200)
>>> a = memoryview(point)
>>> b = memoryview(point)
>>> a == point
False
>>> a == b
False
```

Tenga en cuenta que, al igual que con los números de punto flotante, `v is w` no implica *not* o `v == w` para los objetos de vista de memoria.

Distinto en la versión 3.3: Versiones previas comparaban la memoria directamente, sin considerar ni el formato de los elementos ni la estructura lógica del arreglo.

tobytes(*order='C'*)

Retorna los datos en el *buffer* en forma de cadena de bytes. Equivale a llamar al constructor de la clase **`bytes`** en el objeto *memoryview*:

```
>>> m = memoryview(b"abc")
>>> m.tobytes()
b'abc'
>>> bytes(m)
b'abc'
```

Para arreglos no contiguos el resultado es igual a la representación en forma de lista aplanada, con todos los elementos convertidos a bytes. El método **`tobytes()`** soporta todos los formatos de cadenas de caracteres, incluidos aquellos que no se encuentran en la sintaxis del módulo **`struct`**.

Added in version 3.8: El valor de *order* puede ser {“C”, “F”, “A”}. Cuando *order* es “C” o “F”, los datos en el arreglo original se convierten al orden de C o Fortran. Para vistas contiguas, “A” retorna una copia exacta de la memoria física. En particular, el orden en memoria de Fortran se mantiene inalterado. Para vistas no contiguas, los datos se convierten primero a C. Definir *order=None* es lo mismo que *order=”C”*.

hex(**, bytes_per_sep=1*)
hex(*sep, bytes_per_sep=1*)

Retorna una cadena de caracteres que contiene dos dígitos hexadecimales por cada byte en el *buffer*:

```
>>> m = memoryview(b"abc")
>>> m.hex()
'616263'
```

Added in version 3.5.

Distinto en la versión 3.8: De forma similar a **`bytes.hex()`**, **`memoryview.hex()`** soporta ahora los parámetros opcionales *sep* y *bytes_per_sep* para insertar separadores entre los bytes en la cadena hexadecimal de salida.

tolist()

Retorna los datos en el *buffer* como una lista de elementos.

```
>>> memoryview(b'abc').tolist()
[97, 98, 99]
>>> import array
>>> a = array.array('d', [1.1, 2.2, 3.3])
>>> m = memoryview(a)
>>> m.tolist()
[1.1, 2.2, 3.3]
```

Distinto en la versión 3.3: El método **`tolist()`** soporta ahora todos los formatos nativos de un solo carácter definidos en el módulo **`struct`**, así como las representaciones de múltiples dimensiones.

toreadonly()

Retorna una versión de solo lectura del objeto *memoryview*. El objeto original permanece inalterado:

```
>>> m = memoryview(bytearray(b'abc'))
>>> mm = m.toreadonly()
>>> mm.tolist()
[97, 98, 99]
>>> mm[0] = 42
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot modify read-only memory
>>> m[0] = 43
>>> mm.tolist()
[43, 98, 99]
```

Added in version 3.8.

release()

Libera el buffer subyacente expuesto por el objeto *memoryview*. Muchos objetos realizan operaciones especiales cuando una vista los está conteniendo (por ejemplo, un objeto **`bytearray`** temporalmente prohíbe el cambio de tamaño); la llamada a *release()* sirve para eliminar estas restricciones (así como para tratar con los recursos pendientes) lo más pronto posible.

Después de llamar a este método, cualquier operación posterior en la vista genera un **`ValueError`** (excepto el propio **`release()`**, que se puede llamar varias veces):

```
>>> m = memoryview(b'abc')
>>> m.release()
>>> m[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: operation forbidden on releas
```

El protocolo de gestión de contexto puede ser usado para obtener un efecto similar, usando la sentencia `with`:

```
>>> with memoryview(b'abc') as m:
...     m[0]
...
97
>>> m[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: operation forbidden on releas
```

Added in version 3.2.

cast(*format*, */*)
cast(*format*, *shape*, */*)

Transforma el formato o el tamaño de un objeto *memoryview*. El parámetro *shape* por defecto vale `[byte_length//new_itemsize]`, lo que significa que el resultado será unidimensional. El valor de retorno es un nuevo objeto de tipo *memoryview*, pero el buffer en sí no se copia. Las transformaciones pueden ser 1D -> C-contiguo y C-contiguo -> 1D.

El formato de destino está restringido a un formato nativo de un solo elemento en la sintaxis **`struct`**. Uno de los formatos debe ser un formato de bytes (“B”, “b” o “c”). La longitud en bytes del resultado debe ser la misma que la longitud original. Tenga en cuenta que todas las longitudes en bytes pueden depender del sistema operativo.

Transforma de `1D/long` a bytes `1D/unsigned`:

```
>>> import array
>>> a = array.array('l', [1,2,3])
>>> x = memoryview(a)
>>> x.format
'l'
>>> x.itemsize
8
>>> len(x)
3
>>> x.nbytes
24
>>> y = x.cast('B')
>>> y.format
'B'
>>> y.itemsize
1
```

```
>>> len(y)
24
>>> y.nbytes
24
```

Transforma de `1D/unsigned` a bytes `1D/char`:

```
>>> b = bytearray(b'zyz')
>>> x = memoryview(b)
>>> x[0] = b'a'
Traceback (most recent call last):
  ...
TypeError: memoryview: invalid type for f
>>> y = x.cast('c')
>>> y[0] = b'a'
>>> b
bytearray(b'ayz')
```

Transforma de `1D/bytes` a `3D/ints` a caracteres `1D/signed`:

```
>>> import struct
>>> buf = struct.pack("i"*12, *list(range
>>> x = memoryview(buf)
>>> y = x.cast('i', shape=[2,2,3])
>>> y.tolist()
[[[0, 1, 2], [3, 4, 5]], [[6, 7, 8], [9,
>>> y.format
'i'
>>> y.itemsize
4
>>> len(y)
2
>>> y.nbytes
48
>>> z = y.cast('b')
>>> z.format
'b'
```

```
>>> z.itemsize
1
>>> len(z)
48
>>> z.nbytes
48
```

Transforma de *long* `1D/unsigned` a *long* `2D/unsigned`:

```
>>> buf = struct.pack("L"*6, *list(range
>>> x = memoryview(buf)
>>> y = x.cast('L', shape=[2,3])
>>> len(y)
2
>>> y.nbytes
48
>>> y.tolist()
[[0, 1, 2], [3, 4, 5]]
```

Added in version 3.3.

Distinto en la versión 3.5: El formato de origen ya no está restringido cuando se transforma a una vista de bytes.

count(*value*, */*)

Count the number of occurrences of *value*.

Added in version 3.14.

index(*value*, *start=0*, *stop=sys.maxsize*, */*)

Return the index of the first occurrence of *value* (at or after index *start* and before index *stop*).

Raises a **`ValueError`** if *value* cannot be found.

Added in version 3.14.

Hay disponibles varios atributos de solo lectura:

obj

El objeto subyacente del *memoryview*:

```
>>> b  = bytearray(b'xyz')
>>> m = memoryview(b)
>>> m.obj is b
True
```

Added in version 3.3.

nbytes

`nbytes == product(shape) * itemsize == len(m.tobytes())`. Este es el espacio, medido en bytes, que usará el arreglo en una representación continua. No tiene que ser necesariamente igual a `len(m)`:

```
>>> import array
>>> a = array.array('i', [1,2,3,4,5])
>>> m = memoryview(a)
>>> len(m)
5
>>> m.nbytes
20
>>> y = m[::2]
>>> len(y)
3
>>> y.nbytes
12
>>> len(y.tobytes())
12
```

Arreglos de múltiples dimensiones:

```
>>> import struct
>>> buf = struct.pack("d"*12, *[1.5*x for
>>> x = memoryview(buf)
>>> y = x.cast('d', shape=[3,4])
>>> y.tolist()
[[0.0, 1.5, 3.0, 4.5], [6.0, 7.5, 9.0, 10.
>>> len(y)
3
>>> y.nbytes
96
```

Added in version 3.3.

readonly

Un booleano que indica si la memoria es de solo lectura.

format

Una cadena de caracteres que contiene el formato (en el estilo del módulo **`struct`**) para cada elemento de la vista. Un objeto *memoryview* se puede crear a partir de un exportador con textos de formato arbitrarios, pero algunos métodos (como, por ejemplo, **`tolist()`**) están restringidos a usar formatos de elementos nativos sencillos.

Distinto en la versión 3.3: el formato `'B'` se gestiona ahora de acuerdo a la sintaxis descrita en el módulo `struct`. Esto significa que `memoryview(b'abc')[0] == b'abc'[0] == 97`.

itemsize

El tamaño en bytes de cada elemento del objeto *memoryview*:

```
>>> import array, struct
>>> m = memoryview(array.array('H', [32000
>>> m.itemsize
2
```

```
>>> m[0]
32000
>>> struct.calcsize('H') == m.itemsize
True
```

ndim

Un número entero que indica cuantas dimensiones de un arreglo multi-dimensional representa la memoria.

shape

Una tupla de números enteros, de longitud **`ndim`**, que indica la forma de la memoria en un arreglo de *N* dimensiones.

Distinto en la versión 3.3: Una tupla vacía, en vez de `None`, cuando `ndim = 0`.

strides

Una tupla de números enteros, de longitud **`ndim`**, que indica el tamaño en bytes para acceder a cada dimensión del arreglo.

Distinto en la versión 3.3: Una tupla vacía, en vez de `None`, cuando `ndim = 0`.

suboffsets

De uso interno para los arreglos estilo *PIL*. El valor es solo informativo.

c_contiguous

Un booleano que indica si la memoria es contiguous al estilo C.

Added in version 3.3.

f_contiguous

Un booleano que indica si la memoria es

contiguous al estilo Fortran.

Added in version 3.3.

contiguous

Un booleano que indica si la memoria es contiguous.

Added in version 3.3.

For information on the thread safety of `memoryview` objects in the free-threaded build, see Thread safety for memoryview objects.

Conjuntos — `set`, `frozenset`

Un objeto de tipo *set* es una colección no ordenada de distintos objetos hashable. Los casos de uso habituales incluyen comprobar la pertenencia al conjunto de un elemento, eliminar duplicados de una secuencia y realizar operaciones matemáticas como la intersección, la unión, la diferencia o la diferencia simétrica. (Para otros tipos de contenedores véanse las clases incorporadas `dict`, `list`, y `tuple`, así como el módulo `collections`).

Como otras colecciones, los conjuntos soportan `x in set`, `len(set)` y `for x in set`. Como es una colección sin orden, los conjuntos no registran ni la posición ni el orden de inserción de los elementos. Por lo mismo, los conjuntos no soportan indexado, ni operaciones de segmentado, ni otras capacidades propias de las secuencias.

There are currently two built-in set types, `set` and `frozenset`. The `set` type is mutable — the contents can be changed using methods like `add()` and `remove()`. Since it is mutable, it has no hash value and cannot be used as either a dictionary key or as an element of another set. The `frozenset` type is immutable and hashable — its contents cannot be altered after it is created; it can therefore be used as a dictionary key or as an element of another set.

Se pueden crear conjuntos no vacíos (*sets*, no *frozensets*) escribiendo una lista de elementos separados por comas, entre llaves, por ejemplo `{'jack', 'sjoerd'}`, además de con el constructor de la clase **`set`**.

El constructor para ambas clases se usa de la misma forma:

class set(*iterable=()*, */*)
class frozenset(*iterable=()*, */*)

Retorna un nuevo *set* o *frozenset*, tomando los elementos a partir de *iterable*. Los elementos de un conjunto tienen que tener la propiedad de ser hashable. Para representar conjuntos anidados, o conjuntos de conjuntos, los conjuntos interiores tienen que ser instancias de **`frozenset`**. Si no se especifica el parámetro *iterable*, se retorna un conjunto vacío.

Los conjuntos (*sets*) se pueden construir de diferentes formas:

- Usando una lista de elementos separados por coma entre corchetes: `{'jack', 'sjoerd'}`
- Usando un *set comprehention*: `{c for c in 'abracadabra' if c not in 'abc'}`
- Usando un constructor de tipo: `set()`, `set('foobar')`, `set(['a', 'b', 'foo'])`

Las instancias de **`set`** y **`frozenset`** proporcionan las siguientes operaciones:

len(s)

Retorna el número de elementos en el conjunto *s* (cardinalidad de *s*).

x in s

Comprueba que el elemento *x* está incluido en *s*.

x not in s

Comprueba que el elemento *x* no está incluido en *s*.

frozenset.isdisjoint(*other*, */*)
set.isdisjoint(*other*, */*)

Retorna `True` si el conjunto no tienen ningún elemento en común con *other*. Dos conjuntos son disjuntos si y solo si su intersección es el conjunto vacío.

frozenset.issubset(*other*, /)
set.issubset(*other*, /)

set < = other

Comprueba si cada elemento del conjunto también se encuentra en *other*.

set < other

Comprueba si el conjunto es un subconjunto propio de *other*, es decir, `set <= other and set != other`.

frozenset.issuperset(*other*, /)
set.issuperset(*other*, /)

set > = other

Comprueba que cada elemento de *other* está incluido en el conjunto.

set > other

Comprueba si el conjunto es un superconjunto propio de *other*, es decir, `set >= other and set != other`.

frozenset.union(**others*)
set.union(**others*)

set | other | ...

Retorna un conjunto nuevo que contiene todos los elementos del conjunto y de *others*.

frozenset.intersection(**others*)
set.intersection(**others*)

set & other & ...

Retorna un conjunto nuevo que contiene todos los elementos que están a la vez en conjunto y en *others*.

frozenset.difference(**others*)
set.difference(**others*)

set - other - ...

Retorna un conjunto nuevo que contiene todos los elementos del conjunto y que no están incluidos en *others*.

frozenset.symmetric_difference(*other*, */*)
set.symmetric_difference(*other*, */*)

set ^ other

Retorna un conjunto nuevo que contiene elementos que están incluidos en el conjunto o en *others*, pero no en los dos a la vez.

frozenset.copy()
set.copy()

Retorna una copia superficial del conjunto.

Note, the non-operator versions of **`union()`**, **`intersection()`**, **`difference()`**, **`symmetric_difference()`**, **`issubset()`**, and **`issuperset()`** methods will accept any iterable as an argument. In contrast, their operator based counterparts require their arguments to be sets. This precludes error-prone constructions like `set('abc') & 'cbs'` in favor of the more readable `set('abc').intersection('cbs')`.

Ambas clases **`set`** y **`frozenset`** soportan comparaciones entre sí. Dos conjuntos son iguales si y solo si cada elemento de cada conjunto está incluido en el otro (cada uno de ellos es subconjunto del otro). Un conjunto es menor que otro si y solo si el primero es un subconjunto propio del segundo (es un subconjunto, pero no son iguales). Un conjunto es mayor que otro si y solo si el primero es un superconjunto propio del segundo (es un superconjunto, pero no son iguales).

Las instancias de **`set`** se comparan con las instancias de **`frozenset`** en base a sus elementos. Por ejemplo `set('abc') == frozenset('abc')` retorna `True` y lo mismo hace `set('abc') in set([frozenset('abc')])`.

Las comparaciones de subconjunto e igualdad no son tan generales que permitan una función de ordenación total. Por ejemplo, dos conjuntos cualesquiera que no estén vacíos y que sean disjuntos no son iguales y tampoco son subconjuntos uno del otro, así que todas estas operaciones retornan `False`: `a<b`, `a==b` o `a>b`.

Como los conjuntos solo definen un orden parcial (relaciones de conjuntos), la salida del método **`list.sort()`** no está definida para listas de conjuntos.

Los elementos de un conjunto, al igual que las claves de un diccionario, deben ser hashable.

Las operaciones binarias que mezclan instancias de **`set`** y **`frozenset`** retornan el tipo del primer operando. Por ejemplo: `frozenset('ab') | set('bc')` retornará una instancia de **`frozenset`**.

La siguiente tabla muestra las operaciones disponibles para la clase **`set`** que no son aplicables a los conjuntos inmutables **`frozenset`**:

set.update(**others*)

set |= other | ...

> Actualiza el conjunto, añadiendo los elementos que se encuentren en *others*.

set.intersection_update(**others*)

set &= other & ...

> Actualiza el conjunto, manteniendo solo los elementos que se encuentren en si mismo y en *others*.

set.difference_update(**others*)

set -= other | ...

Actualiza el conjunto, eliminado los elementos que se encuentren en *others*.

set.symmetric_difference_update(*other*, */*)

set ^= other

Actualiza el conjunto, manteniendo solo los elementos que se encuentren en el conjunto o en *others*, pero no en los dos a la vez.

set.add(*elem*, */*)

Añade al conjunto el elemento *elem*.

set.remove(*elem*, */*)

Elimina del conjunto el elemento *elem*. Lanza la excepción `KeyError` si *elem* no estaba incluido en el conjunto.

set.discard(*elem*, */*)

Elimina del conjunto el elemento *elem*, si estuviera incluido.

set.pop()

Elimina y retorna un elemento cualquiera del conjunto. Lanza la excepción `KeyError` si el conjunto está vacío.

set.clear()

Elimina todos los elementos del conjunto.

Note, the non-operator versions of the `update()`, `intersection_update()`, `difference_update()`, and `symmetric_difference_update()` methods will accept any iterable as an argument.

Note, the *elem* argument to the `__contains__()`, `remove()`, and `discard()` methods may be a set. To support searching for an equivalent frozenset, a temporary one is created from *elem*.

Ver también

For detailed information on thread-safety guarantees for `set` objects, see Thread safety for set objects.

Tipos mapa — `dict`

Un objeto de tipo mapping relaciona valores (que deben ser hashable) con objetos de cualquier tipo. Los mapas son objetos mutables. En este momento solo hay un tipo estándar de mapa, los *dictionary*. (Para otros tipos contenedores, véanse las clases incorporadas `list`, `set`, y `tuple`, así como el módulo `collections`).

Las claves de un diccionario son valores arbitrarios *almost*. Los valores que no sean hashable, es decir, los valores que contienen listas, diccionarios u otros tipos mutables (que se comparan por valor en lugar de por identidad de objeto) no se pueden utilizar como claves. Los valores que se comparan como iguales (como `1`, `1.0` y `True`) se pueden utilizar indistintamente para indexar la misma entrada de diccionario.

class dict(***kwargs*)
class dict(*mapping*, */*, ***kwargs*)
class dict(*iterable*, */*, ***kwargs*)

Retorna un diccionario creado a partir de un parámetro opcional por posición, y por una serie de parámetros por nombre, también opcionales.

Los diccionarios se pueden construir de diferentes formas:

- Usando una lista separada por comas de pares `key: value` entre llaves: `{'jack': 4098, 'sjoerd': 4127}` o `{4098: 'jack', 4127: 'sjoerd'}`
- Usando una comprensión de diccionario: `{}`, `{x: x`

  `** 2 for x in range(10)}`
- Usando un constructor de tipo: `dict()`, `dict([('foo', 100), ('bar', 200)])`, `dict(foo=100, bar=200)`

If no positional argument is given, an empty dictionary is created. If a positional argument is given and it defines a `keys()` method, a dictionary is created by calling **`__getitem__()`** on the argument with each returned key from the method. Otherwise, the positional argument must be an iterable object. Each item in the iterable must itself be an iterable with exactly two elements. The first element of each item becomes a key in the new dictionary, and the second element the corresponding value. If a key occurs more than once, the last value for that key becomes the corresponding value in the new dictionary.

Si se usan parámetros por nombre, los nombres de los parámetros y los valores asociados se añaden al diccionario creado a partir del parámetro por posición. Si un valor de clave ya estaba presente, el valor pasado con el parámetro por nombre reemplazará el valor del parámetro por posición.

Dictionaries compare equal if and only if they have the same `(key, value)` pairs (regardless of ordering). Order comparisons (“<”, “<=”, “>=”, “>”) raise **`TypeError`**. To illustrate dictionary creation and equality, the following examples all return a dictionary equal to `{"one": 1, "two": 2, "three": 3}`:

```
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}
>>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]
>>> d = dict([('two', 2), ('one', 1), ('three', 3)]
>>> e = dict({'three': 3, 'one': 1, 'two': 2})
>>> f = dict({'one': 1, 'three': 3}, two=2)
>>> a == b == c == d == e == f
True
```

Si queremos definir claves con parámetros por nombre, como en el primer ejemplo, entonces los valores de clave solo puede ser cadenas de texto conteniendo identificadores de Python válidos. En los otros casos, se puede usar cualquier valor como clave.

Los diccionarios mantienen de forma interna el orden de inserción. Actualizar una clave no modifica ese orden. Las claves que vuelven a ser insertadas después de haber sido borradas se añaden al final.:

```
>>> d = {"one": 1, "two": 2, "three": 3, "four": 4}
>>> d
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
>>> list(d)
['one', 'two', 'three', 'four']
>>> list(d.values())
[1, 2, 3, 4]
>>> d["one"] = 42
>>> d
{'one': 42, 'two': 2, 'three': 3, 'four': 4}
>>> del d["two"]
>>> d["two"] = None
>>> d
{'one': 42, 'three': 3, 'four': 4, 'two': None}
```

Distinto en la versión 3.7: Se garantiza que el orden del diccionario es el de inserción. Este comportamiento era un detalle de implementación en CPython desde la versión 3.6.

Estas son las operaciones soportados por los diccionarios (y que, por tanto, deberían ser soportados por los tipos de mapa personalizados):

list(d)

Retorna una lista de todas las claves usadas en el diccionario *d*.

len(d)

Retorna el número de elementos almacenados en el diccionario *d*.

d[key]

Retorna el elemento dentro de *d* almacenado bajo la clave *key*. Lanza una excepción de tipo **`KeyError`** si la clave *key* no se encuentra en el diccionario *d*.

If a subclass of dict defines a method **`__missing__()`** and *key* is not present, the `d[key]` operation calls that method with the key *key* as argument. The `d[key]` operation then returns or raises whatever is returned or raised by the `__missing__(key)` call. No other operations or methods invoke **`__missing__()`**. If **`__missing__()`** is not defined, **`KeyError`** is raised. **`__missing__()`** must be a method; it cannot be an instance variable:

```
>>> class Counter(dict):
...     def __missing__(self, key):
...         return 0
...
>>> c = Counter()
>>> c['red']
0
>>> c['red'] += 1
>>> c['red']
1
```

The example above shows part of the implementation of **`collections.Counter`**. A different **`__missing__()`** method is used by **`collections.defaultdict`**.

d[key] = value

Asigna el valor *value* a `d[key]`.

del d[key]

Elimina `d[key]` de *d*. Lanza una excepción **`KeyError`** si *key* no está en el mapa.

key in d

Retorna `True` si *d* tiene una entrada en la clave *key*, `False` en caso contrario.

key not in d

Equivale a `not key in d`.

iter(d)

Retorna un iterador que recorre todas las claves de un diccionario. Es una forma abreviada de `iter(d.keys())`.

clear()

Elimina todos los elementos del diccionario.

copy()

Retorna una copia superficial del diccionario.

classmethod fromkeys(*iterable*, *value=None*, */*)

Crea un nuevo diccionario con las claves obtenidos a partir del *iterable* y con valor *value*.

El método **`fromkeys()`** es un método de clase que retorna un diccionario nuevo. El valor de *value* por defecto es `None`. Todos los valores harán referencia a una única instancia, por lo que en general no tiene sentido que *value* sea un objeto mutable, como una lista vacía. Para poder obtener valores diferentes, se puede usar mejor un diccionario por comprensión.

get(*key*, *default=None*, */*)

Retorna el elemento dentro de *d* almacenado bajo la clave *key*, si *key* está en el diccionario; si no, retorna *default*. El valor de *default* por defecto es `None`, por lo que esta función nunca lanza la excepción **`KeyError`**.

items()

Retorna una nueva vista de los elementos del diccionario (pares `(key, value)`). Véase la documentación de los objetos vistas.

keys()

Retorna una nueva vista de las claves del diccionario. Véase la documentación de las vistas.

pop(*key*, /)
pop(*key*, *default*, /)

Si *key* está en el diccionario, lo elimina del diccionario y retorna su valor; si no está, retorna *default*. Si no se especifica valor para *default* y la *key* no se encuentra en el diccionario, se lanza la excepción **`KeyError`**.

popitem()

Elimina y retorna una pareja `(key, value)` del diccionario. Las parejas se retornan en el orden LIFO.

El método **`popitem()`** es útil para recorrer y a la vez vaciar un diccionario, un proceso usado a menudo en algoritmos de conjuntos. Si el diccionario está vacío, llamar a **`popitem()`** lanza la excepción **`KeyError`**.

Distinto en la versión 3.7: El orden *LIFO* ahora está garantizado. En versiones anteriores, el método **`popitem()`** retorna una pareja clave/valor arbitraria.

reversed(d)

Retorna un iterador que recorre las claves en orden inverso. Es una forma abreviada de `reversed(d.keys())`.

Added in version 3.8.

setdefault(*key*, *default=None*, /)

Si *key* está incluida en el diccionario, retorna el valor almacenado. Si no, inserta con la clave *key* el valor definido en *default* y retorna *default*. El valor por defecto de *default* es `None`.

update(***kwargs*)
update(*mapping*, /, ***kwargs*)
update(*iterable*, /, ***kwargs*)

Update the dictionary with the key/value pairs from *mapping* or *iterable* and *kwargs*, overwriting existing keys. Return `None`.

`update()` accepts either another object with a `keys()` method (in which case **`__getitem__()`** is called with every key returned from the method) or an iterable of key/value pairs (as tuples or other iterables of length two). If keyword arguments are specified, the dictionary is then updated with those key/value pairs: `d.update(red=1, blue=2)`.

values()

Retorna una nueva vista de los valores del diccionario. Véase la documentación sobre objetos vistas.

Una comparación de igualdad entre una vista `dict.values()` y otra siempre retornará `False`. Esto también pasa cuando se compara `dict.values()` consigo mismo:

```
>>> d = {'a': 1}
>>> d.values() == d.values()
False
```

d | other

Crea un nuevo diccionario con las claves y valores fusionados de *d* y *other*, por lo cual ambos deben ser diccionarios. Los valores de *other* tienen prioridad cuando *d* y *other* tienen claves compartidas.

Added in version 3.9.

d |= other

Actualiza el diccionario *d* con las claves y valores de *other*, el cual podría ser ya sea un a mapping o un iterable de pares clave/valor. Los valores de *other* tienen prioridad cuando *d* y *other* tienen claves compartidas.

Added in version 3.9.

Tanto los diccionarios como las vistas basadas en diccionarios son reversibles:

```
>>> d = {"one": 1, "two": 2, "three": 3, "four": 4}
>>> d
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
>>> list(reversed(d))
['four', 'three', 'two', 'one']
>>> list(reversed(d.values()))
[4, 3, 2, 1]
>>> list(reversed(d.items()))
[('four', 4), ('three', 3), ('two', 2), ('one', 1)]
```

Distinto en la versión 3.8: Los diccionarios son ahora reversibles.

Ver también

Se puede usar un objeto de tipo `types.MappingProxyType` para crear una vista de solo lectura de un objeto `dict`.

Ver también

For detailed information on thread-safety guarantees for `dict` objects, see Thread safety for dict objects.

Objetos tipos vista de diccionario

Los objetos retornados por los métodos `dict.keys()`,

`dict.values()` y **`dict.items()`** son objetos tipo vista o *view*. Estos objetos proporcionan una vista dinámica del contenido del diccionario, lo que significa que si el diccionario cambia, las vistas reflejan estos cambios.

Las vistas de un diccionario pueden ser iteradas para retornar sus datos respectivos, y soportan operaciones de comprobación de pertenencia:

len(dictview)

> Retorna el número de entradas en un diccionario.

iter(dictview)

> Retorna un iterador sobre las claves, valores o elementos (representados en forma de tuplas `(key, value)`) de un diccionario.
>
> Las claves y los valores se iteran en orden de inserción. Esto permite la creación de parejas `(value, key)` usando la función **`zip()`**: `pairs = zip(d.values(), d.keys())`. Otra forma de crear la misma lista es `pairs = [(v, k) for (k, v) in d.items()]`.
>
> Iterar sobre un diccionario a la vez que se borran o añaden entradas puede lanzar una excepción de tipo **`RuntimeError`**, o puede provocar que no se iteren sobre todas las entradas.
>
> *Distinto en la versión 3.7:* Se garantiza que el orden de los diccionarios es el de inserción.

x in dictview

> Retorna `True` si *x* está incluido en las claves, los valores o los elementos del diccionario (en el último caso, *x* debe ser una tupla `(key, value)`).

reversed(dictview)

> Retorna un iterador inverso sobre las claves, los valores o los elementos del diccionario. El orden de la vista será el inverso

del orden de inserción.

Distinto en la versión 3.8: Las vistas de un diccionario no son reversibles.

dictview.mapping

Retorna un `types.MappingProxyType` que envuelve el diccionario original al que se refiere la vista.

Added in version 3.10.

Las vistas de claves son similares a conjuntos, ya que sus entradas son únicas y hashable. Las vistas de elementos también tienen operaciones similares a conjuntos, ya que los pares (clave, valor) son únicos y las claves se pueden codificar en hash. Si todos los valores de una vista de elementos también se pueden codificar en hash, la vista de elementos puede interoperar con otros conjuntos. (Las vistas de valores no se tratan como similares a conjuntos, ya que las entradas generalmente no son únicas). Para las vistas similares a conjuntos, están disponibles todas las operaciones definidas para la clase base abstracta `collections.abc.Set` (por ejemplo, `==`, `<` o `^`). Al utilizar operadores de conjunto, las vistas similares a conjuntos aceptan cualquier iterable como el otro operando, a diferencia de los conjuntos que solo aceptan conjuntos como entrada.

Un ejemplo de uso de una vista de diccionario:

```
>>> dishes = {'eggs': 2, 'sausage': 1, 'bacon': 1, 'spam
>>> keys = dishes.keys()
>>> values = dishes.values()

>>> # iteration
>>> n = 0
>>> for val in values:
...     n += val
...
>>> print(n)
504
```

```
>>> # Las llaves y los valores se iteran en el mismo orc
>>> list(keys)
['eggs', 'sausage', 'bacon', 'spam']
>>> list(values)
[2, 1, 1, 500]

>>> # Los objetos vista son dinámicos y reflejan cambios
>>> del dishes['eggs']
>>> del dishes['sausage']
>>> list(keys)
['bacon', 'spam']

>>> # operaciones de conjunto
>>> keys & {'eggs', 'bacon', 'salad'}
{'bacon'}
>>> keys ^ {'sausage', 'juice'} == {'juice', 'sausage',
True
>>> keys | ['juice', 'juice', 'juice'] == {'bacon', 'spa
True

>>> # recuperar un proxy de solo lectura para el diccior
>>> values.mapping
mappingproxy({'bacon': 1, 'spam': 500})
>>> values.mapping['spam']
500
```

Tipos gestores de contexto

La expresión **`with`** de Python soporta el concepto de un contexto en tiempo de ejecución definido mediante un gestor de contexto. Para implementar esto, se permite al usuario crear clases para definir estos contextos definiendo dos métodos, uno a ejecutar antes de entrar del bloque de código y otro a ejecutar justo después de salir del mismo:

contextmanager.__enter__()

> Entra en el contexto en tiempo de ejecución, y retorna o bien este objeto u otro relacionado con el contexto. El valor

retornado por este método se vincula al identificador que viene tras la palabra clave `as` usada en la sentencia `with` que define el contexto.

Un ejemplo de gestor de contexto que se retorna a si mismo es un objeto de tipo file object. Los objetos de tipo `File` se retornan a si mismo en la llamada a `__enter__()`, lo que permite que `open()` sea usado como gestores de contexto en una sentencia `with`.

Un ejemplo de gestor de contexto que retorna otro objeto relacionado en el que define la función `decimal.localcontext()`. Este gestor define el contexto de uso en operaciones decimales a partir de una copia del contexto original, y retorna esa copia. De esta manera se puede cambiar el contexto decimal dentro del cuerpo del `with` sin afectar al código fuera de `with`.

contextmanager.__exit__(*exc_type*, *exc_val*, *exc_tb*)

Sale del contexto y retorna un indicador booleano que indica si se debe ignorar cualquier posible excepción que hubiera ocurrido dentro del mismo. Si se produce una excepción mientras se ejecutan las sentencias definidas en el cuerpo de la sentencia `with`, los parámetros de esta función contienen el tipo y valor de la excepción, así como la información relativa a la traza de ejecución. Si no se produce ninguna excepción, los tres parámetros valen `None`.

Returning a true value from this method will cause the `with` statement to suppress the exception and continue execution with the statement immediately following the `with` statement. Otherwise the exception continues propagating after this method has finished executing.

If this method raises an exception while handling an earlier exception from the `with` block, the new exception is raised, and the original exception is stored in its `__context__` attribute.

La excepción que se pasa nunca debe volver a generarse

> explícitamente; en su lugar, este método debe devolver un valor falso para indicar que el método se completó correctamente y no desea suprimir la excepción generada. Esto permite que el código de administración de contexto detecte fácilmente si un método `__exit__()` ha fallado o no.

Python define varios gestores de contexto para facilitar la sincronía entre hilos, asegurarse del cierre de ficheros y otros objetos similares y para modificar de forma simple el contexto para las expresiones aritméticas con decimales. Los tipos específicos no se tratan especialmente fuera de su implementación del protocolo de administración de contexto. Véase el módulo `contextlib` para algunos ejemplos.

Los generator de Python y el decorador definidos en la clase `contextlib.contextmanager` permiten implementar de forma sencilla estos protocolos. Si una función generadora se decora con la clase `contextlib.contextmanager`, retornará un gestor de contexto que incluye los métodos necesarios `__enter__()` y `__exit__()`, en vez del iterador que se produciría si no se decora la función generadora.

Nótese que no hay una ranura específica para ninguno de estos métodos en la estructura usada para los objetos Python en el nivel de la API de Python/C. Objetos que quieran definir estos métodos deben implementarlos como métodos normales de Python. Comparado con la complejidad de definir un contexto en tiempo de ejecución, lo complejidad de una búsqueda simple en un diccionario es mínima.

Tipos de anotaciones de type — alias genérico, Union

Los tipos principales integrados para anotaciones de tipo son alias genérico y Union.

Tipo Alias Genérico

Los objetos `GenericAlias` generalmente se crean para suscribir a una clase. Se utilizan con mayor frecuencia con clases contenedoras, como **`list`** o **`dict`**. Por ejemplo, `list[int]` es un objeto `GenericAlias` que se creó para suscribir la clase `list` con el argumento **`int`**. Los objetos `GenericAlias` están pensados principalmente para usar con anotaciones de tipo.

Nota

Generalmente solo es posible suscribir a una clase si ésta implementa el método especial **`__class_getitem__()`**.

El objeto `GenericAlias` actúa como *proxy* para tipo genérico, implementando *parameterized generics*.

Para una clase contenedora, el argumento (o los argumentos) proporcionado(s) a una suscripción de la clase puede indicar el tipo (o tipos) de los elementos que contiene un objeto. Por ejemplo, se puede usar `set[bytes]` en anotaciones de tipo para significar un **`set`** en el que todos los elementos son del tipo **`bytes`**.

Para una clase que define el método **`__class_getitem__()`** pero no es un contenedor, el argumento (o los argumentos) proporcionado(s) a una suscripción de la clase a menudo indicarán el tipo (o los tipos) de retorno de uno o más métodos definidos en un objeto. Por ejemplo, las **`expresiones regulares`** se pueden usar tanto para el tipo de datos **`str`** y el tipo de datos **`bytes`**:

- Si `x = re.search('foo', 'foo')`, `x` será un objeto re.Match donde los valores de retorno de `x.group(0)` y `x[0]` serán de tipo **`str`**. Podemos representar este tipo de objeto en anotaciones de type con el `GenericAlias` de `re.Match[str]`.
- Si `y = re.search(b'bar', b'bar')`, (nótese que `b` es para **`bytes`**), `y` también será una instancia de `re.Match`, pero los valores de retorno de `y.group(0)` y `y[0]` serán de tipo **`bytes`**. En anotaciones de type, representaríamos esta variedad de objetos re.Match con `re.Match[bytes]`.

Los objetos `GenericAlias` son instancias de la clase

`types.GenericAlias`, que también se puede usar para crear directamente objetos `GenericAlias`.

T[X, Y, ...]

Crea un `GenericAlias` que representa un tipo `T` parametrizado por tipos *X*, *Y* y más dependiendo de la `T` usada. Por ejemplo, una función espera un **`list`** que contenga elementos **`float`**:

```
def average(values: list[float]) -> float:
    return sum(values) / len(values)
```

Otro ejemplo para el mapping de objetos, usando un **`dict`**, el cual es un tipo genérico que espera dos parámetros de tipo que representan el tipo de la clave y el tipo del valor. En este ejemplo, la función espera un `dict` con claves de tipo **`str`** y valores de tipo **`int`**:

```
def send_post_request(url: str, body: dict[str, int
    ...
```

Las funciones integradas **`isinstance()`** y **`issubclass()`** no aceptan tipos `GenericAlias` como segundo argumento:

```
>>> isinstance([1, 2], list[str])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: isinstance() argument 2 cannot be a parameter
```

Python en tiempo de ejecución no hace cumplir las anotaciones de tipo. Esto se extiende para tipos genéricos y sus parámetros. Cuando se crea un objeto contenedor desde un `GenericAlias`, los elementos del contenedor no se verifican con su tipo. Por ejemplo, el siguiente código no es recomendado en lo absoluto, pero correrá sin errores:

```
>>> t = list[str]
>>> t([1, 2, 3])
[1, 2, 3]
```

Además, los genéricos parametrizados borran parámetros de type durante la creación de objetos:

```
>>> t = list[str]
>>> type(t)
<class 'types.GenericAlias'>

>>> l = t()
>>> type(l)
<class 'list'>
```

Llamando a `repr()` o `str()` en un genérico se muestra el tipo parametrizado:

```
>>> repr(list[int])
'list[int]'

>>> str(list[int])
'list[int]'
```

El método `__getitem__()` de contenedores genéricos lanzarán una excepción para rechazar los errores como `dict[str][str]`:

```
>>> dict[str][str]
Traceback (most recent call last):
  ...
TypeError: dict[str] is not a generic class
```

Sin embargo, estas expresiones son válidas cuando se usan variables de type. El índice debe tener tantos elementos como los elementos de variable de type en los `__args__` del objeto `GenericAlias`:

```
>>> from typing import TypeVar
>>> Y = TypeVar('Y')
>>> dict[str, Y][int]
dict[str, int]
```

Clases genéricas estándar

Las siguientes clases de la biblioteca estándar soportan genéricos parametrizados. Esta lista no es exhaustiva.

- `tuple`
- `list`
- `dict`
- `set`
- `frozenset`
- `type`
- `asyncio.Future`
- `asyncio.Task`
- `collections.deque`
- `collections.defaultdict`
- `collections.OrderedDict`
- `collections.Counter`
- `collections.ChainMap`
- `collections.abc.Awaitable`
- `collections.abc.Coroutine`
- `collections.abc.AsyncIterable`
- `collections.abc.AsyncIterator`
- `collections.abc.AsyncGenerator`
- `collections.abc.Iterable`
- `collections.abc.Iterator`
- `collections.abc.Generator`
- `collections.abc.Reversible`
- `collections.abc.Container`
- `collections.abc.Collection`
- `collections.abc.Callable`
- `collections.abc.Set`
- `collections.abc.MutableSet`
- `collections.abc.Mapping`
- `collections.abc.MutableMapping`
- `collections.abc.Sequence`
- `collections.abc.MutableSequence`
- `collections.abc.ByteString`
- `collections.abc.MappingView`
- `collections.abc.KeysView`
- `collections.abc.ItemsView`
- `collections.abc.ValuesView`
- `contextlib.AbstractContextManager`
- `contextlib.AbstractAsyncContextManager`

- `dataclasses.Field`
- `functools.cached_property`
- `functools.partialmethod`
- `os.PathLike`
- `queue.LifoQueue`
- `queue.Queue`
- `queue.PriorityQueue`
- `queue.SimpleQueue`
- re.Pattern
- re.Match
- `shelve.BsdDbShelf`
- `shelve.DbfilenameShelf`
- `shelve.Shelf`
- `types.MappingProxyType`
- `weakref.WeakKeyDictionary`
- `weakref.WeakMethod`
- `weakref.WeakSet`
- `weakref.WeakValueDictionary`

Atributos especiales de los objetos `GenericAlias`

Todos los genéricos parametrizados implementan atributos especiales de solo lectura.

genericalias.__origin__

Este atributo apunta a la clase genérica no parametrizada:

```
>>> list[int].__origin__
<class 'list'>
```

genericalias.__args__

Este atributo es una clase `tuple` (posiblemente de longitud 1) de tipos genéricos pasados al método original `__class_getitem__()` de la clase genérica:

```
>>> dict[str, list[int]].__args__
(<class 'str'>, list[int])
```

genericalias.__parameters__

Este atributo es una tupla (posiblemente vacía) computada perezosamente con las variables de tipo únicas encontradas en `__args__`:

```
>>> from typing import TypeVar

>>> T = TypeVar('T')
>>> list[T].__parameters__
(~T,)
```

Nota

Un objeto `GenericAlias` con **`typing.ParamSpec`** parámetros puede no tener los `__parameters__` correctos después de la sustitución porque **`typing.ParamSpec`** está destinado principalmente a la verificación de tipos estáticos.

genericalias.__unpacked__

Un booleano que es verdadero si el alias ha sido desempaquetado usando el operador `*` (véase **`TypeVarTuple`**).

Added in version 3.11.

Ver también

PEP 484 [https://peps.python.org/pep-0484/] - Type Hints
Presentación del marco de trabajo de Python para anotaciones de tipo.

PEP 585 [https://peps.python.org/pep-0585/] - Sugerencias de tipo genéricas en colecciones estándar
Introducción a la capacidad de parametrizar de forma nativa clases de la biblioteca estándar, siempre que implementen el método de clase especial **`__class_getitem__()`**.

Genéricos, tipos genéricos definidos por el usuario y `typing.Generic`

Documentación sobre cómo implementar clases genéricas que se pueden parametrizar en tiempo de ejecución y que los validadores estático de tipos pueden entender.

Added in version 3.9.

Tipo de conversión

A union object holds the value of the `|` (bitwise or) operation on multiple type objects. These types are intended primarily for type annotations. The union type expression enables cleaner type hinting syntax compared to subscripting `typing.Union`.

X | Y | ...

Define un objeto de conversión que contiene tipos *X*, *Y*, etc. `X | Y` significa X o Y. Es equivalente a `typing.Union[X, Y]`. Por ejemplo, la siguiente función espera un argumento de tipo `int` or `float`:

```
def square(number: int | float) -> int | float:
    return number ** 2
```

Nota

El operando `|` no se puede utilizar en tiempo de ejecución para definir uniones en las que uno o más miembros sean una referencia hacia delante. Por ejemplo, `int | "Foo"`, donde `"Foo"` es una referencia a una clase que aún no se ha definido, fallará en tiempo de ejecución. Para las uniones que incluyen referencias hacia delante, presente la expresión completa como una cadena, por ejemplo, `"int | Foo"`.

union_object == other

Los objetos de conversión se pueden probar para determinar su igualdad con otros objetos de conversión. Detalles:

- Las conversiones de conversión se aplanan:

  ```
  (int | str) | float == int | str | float
  ```

- Se eliminan los tipos redundantes:

  ```
  int | str | int == int | str
  ```

- Al comparar conversiones, se ignora el orden:

  ```
  int | str == str | int
  ```

- It creates instances of **`typing.Union`**:

  ```
  int | str == typing.Union[int, str]
  type(int | str) is typing.Union
  ```

- Los tipos opcionales se pueden escribir como una unión con `None`:

  ```
  str | None == typing.Optional[str]
  ```

isinstance(obj, union_object)

issubclass(obj, union_object)

Las llamadas a **`isinstance()`** y **`issubclass()`** también son compatibles con un objeto de conversión:

```
>>> isinstance("", int | str)
True
```

Sin embargo, no se puede comprobar parameterized generics en objetos de unión:

```
>>> isinstance(1, int | list[int])  # short-circuit
True
>>> isinstance([1], int | list[int])
Traceback (most recent call last):
  ...
TypeError: isinstance() argument 2 cannot be a para
```

The user-exposed type for the union object can be accessed from `typing.Union` and used for `isinstance()` checks:

```
>>> import typing
>>> isinstance(int | str, typing.Union)
True
>>> typing.Union()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot create 'typing.Union' instances
```

Nota

Se agregó el método `__or__()` para objetos de tipo para admitir la sintaxis `X | Y`. Si una metaclase implementa `__or__()`, la Unión puede anularlo:

```
>>> class M(type):
...     def __or__(self, other):
...         return "Hello"
...
>>> class C(metaclass=M):
...     pass
...
>>> C | int
'Hello'
>>> int | C
int | C
```

Ver también

PEP 604 [https://peps.python.org/pep-0604/] — PEP propone la sintaxis `X | Y` y tipo Conversión.

Added in version 3.10.

Distinto en la versión 3.14: Union objects are now instances of `typing.Union`. Previously, they were instances of `types.UnionType`, which remains an alias for `typing.Union`.

Otros tipos predefinidos

El intérprete soporta otros tipos de objetos variados. La mayoría de ellos solo implementan una o dos operaciones.

Módulos

La única operación especial que implementan los módulos es el acceso como atributos: `m.name`, donde *m* es un módulo y *name* accede a un nombre definido en la tabla de símbolos del módulo *m*. También se puede asignar valores a los atributos de un módulo (nótese que la sentencia `import` no es, estrictamente hablando, una operación del objeto de tipo módulo; la sentencia `import foo` no requiere la existencia de un módulo llamado *foo*, sino una *definición* (externa) de un módulo *foo* en alguna parte).

Un atributo especial de cada módulo es `__dict__`. Es un diccionario que contiene la tabla de símbolos del módulo. Cambiar el diccionario cambiará por tanto el contenido de la tabla de símbolos, pero no es posible realizar una asignación directa al atributo `__dict__` (se puede realizar una asignación como `m.__dict__['a'] = 1`, que define el valor de `m.a` como `1`, pero no se puede hacer `m.__dict__ = {}`). No se recomienda manipular los contenidos del atributo `__dict__` directamente.

Los módulos incluidos en el intérprete se escriben así: `<module 'sys' (built-in)>`. Si se cargan desde un archivo, se escriben como `<module 'os' from '/usr/local/lib/pythonX.Y/os.pyc'>`.

Clases e instancias de clase

Véase Objetos, valores y tipos y Definiciones de clase para más información.

Funciones

Los objetos de tipo función se crean mediante definiciones de función. La única operación posible con un objeto de tipo función es llamarla: `func(argument-list)`.

Hay dos tipos de funciones: Las funciones básicas o predefinidas y las funciones definidas por el usuario. Las dos soportan la misma operación (ser llamadas), pero la implementación es diferente, de ahí que se consideren de distintos tipo.

Véase Definiciones de funciones para más información.

Métodos

Methods are functions that are called using the attribute notation. There are two flavors: built-in methods (such as **`append()`** on lists) and class instance method. Built-in methods are described with the types that support them.

Si accede a un método (una función definida en un espacio de nombres de clase) a través de una instancia, obtendrá un objeto especial: un objeto *método enlazado* (también llamado instance method). Cuando se lo llama, agregará el argumento `self` a la lista de argumentos. Los métodos enlazados tienen dos atributos especiales de solo lectura: **`m.__self__`** es el objeto en el que opera el método y **`m.__func__`** es la función que implementa el método. Llamar a `m(arg-1, arg-2, ..., arg-n)` es completamente equivalente a llamar a `m.__func__(m.__self__, arg-1, arg-2, ..., arg-n)`.

Al igual que function objects, los objetos de método enlazados admiten la obtención de atributos arbitrarios. Sin embargo, dado que los atributos de método se almacenan en realidad en el objeto de función subyacente (**`method.__func__`**), no se permite configurar atributos de método en métodos enlazados. Intentar configurar un atributo en un método da como resultado que se genere un **`AttributeError`**. Para configurar un atributo de método, debe configurarlo explícitamente en el objeto de función subyacente:

```
>>> class C:
...     def method(self):
```

```
...         pass
...
>>> c = C()
>>> c.method.whoami = 'my name is method'  # can't set o
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'method' object has no attribute 'whoami
>>> c.method.__func__.whoami = 'my name is method'
>>> c.method.whoami
'my name is method'
```

Véase Métodos de instancia para más información.

Objetos código

La implementación utiliza objetos de código para representar código Python ejecutable «pseudocompilado», como el cuerpo de una función. Se diferencian de los objetos de función porque no contienen una referencia a su entorno de ejecución global. Los objetos de código son devueltos por la función **`compile()`** incorporada y se pueden extraer de los objetos de función a través de su atributo **`__code__`**. Consulte también el módulo **`code`**.

Al acceder a **`__code__`** se genera un auditing event `object.__getattr__` con argumentos `obj` y `"__code__"`.

Un objeto de tipo código puede ser evaluado o ejecutando pasándolo como parámetros a las funciones incorporadas **`exec()`** o **`eval()`** (que también aceptan código Python en forma de cadena de caracteres).

Véase Jerarquía de tipos estándar para más información.

Objetos Tipo

Los objetos de tipo Tipo (*Type*) representan a los distintos tipos de datos. El tipo de un objeto particular puede ser consultado usando la función incorporada **`type()`**. Los objetos Tipo no tienen ninguna operación especial. El módulo **`types`** define nombres para todos los tipos básicos definidos en la biblioteca estándar.

Los tipos se escriben de la siguiente forma: `<class 'int'>`.

El objeto nulo (*Null*)

Todas las funciones que no definen de forma explícita un valor de retorno retornan este objeto. Los objetos nulos no soportan ninguna operación especial. Solo existe un único objeto nulo, llamado `None` (un nombre predefinido o básico). La expresión `type(None)()` produce el mismo objeto `None`, esto se conoce como *Singleton*.

Se escribe `None`.

El objeto puntos suspensivos (*Ellipsis*)

This object is commonly used to indicate that something is omitted. It supports no special operations. There is exactly one ellipsis object, named **`Ellipsis`** (a built-in name). `type(Ellipsis)()` produces the **`Ellipsis`** singleton.

Se puede escribir como `Ellipsis` o `...`.

In typical use, `...` as the `Ellipsis` object appears in a few different places, for instance:

- In type annotations, such as callable arguments or tuple elements.
- As the body of a function instead of a pass statement.
- In third-party libraries, such as Numpy's slicing and striding [https://numpy.org/doc/stable/user/basics.indexing.html#slicing-and-striding].

Python also uses three dots in ways that are not `Ellipsis` objects, for instance:

- Doctest's **`ELLIPSIS`**, as a pattern for missing content.
- The default Python prompt of the interactive shell when partial input is incomplete.

Lastly, the Python documentation often uses three dots in conventional English usage to mean omitted content, even in code examples that also use them as the `Ellipsis`.

El objeto *NotImplemented*

Este objeto se devuelve a partir de comparaciones y operaciones binarias cuando se les pide que operen en tipos que no admiten. Consulte Comparaciones para obtener más información. Hay exactamente un objeto **`NotImplemented`**. `type(NotImplemented)()` produce la instancia singleton.

Se escribe como `NotImplemented`.

Objetos internos

Consulte Jerarquía de tipos estándar para obtener esta información. Describe stack frame objects, traceback objects y los objetos de sector.

Atributos especiales

La implementación añade unos cuantos atributos de solo lectura a varios tipos de objetos, cuando resulta relevante. Algunos de estos atributos son reportados por la función incorporada **`dir()`**.

definition.__name__

El nombre de la clase, función, método, descriptor o instancia generadora.

definition.__qualname__

El nombre calificado (qualified name) de la clase, función, método, descriptor o instancia generadora.

Added in version 3.3.

definition.__module__

El nombre del módulo en el que se definió una clase o función.

definition.__doc__

La cadena de documentación de una clase o función, o `None` si no está definida.

definition.__type_params__

type parameters de clases y funciones genéricas y type aliases. Para clases y funciones que no sean genéricas, será una tupla vacía.

Added in version 3.12.

Limitación de longitud de conversión de cadena de tipo entero

CPython tiene un límite global para conversiones entre **`int`** y **`str`** para mitigar los ataques de denegación de servicio. Este límite *solo* se aplica a decimales u otras bases numéricas que no sean potencias de dos. Las conversiones hexadecimales, octales y binarias son ilimitadas. Se puede configurar el límite.

El tipo **`int`** en CPython es un número de longitud arbitraria almacenado en formato binario (comúnmente conocido como «bignum»). No existe ningún algoritmo que pueda convertir una cadena en un entero binario o un entero binario en una cadena en tiempo lineal; *unless* tiene como base una potencia de 2. Incluso los algoritmos más conocidos para la base 10 tienen una complejidad subcuadrática. Convertir un valor grande como `int('1' * 500_000)` puede llevar más de un segundo en una CPU rápida.

Limitar el tamaño de la conversión ofrece una forma práctica de evitar **CVE 2020-10735** [https://www.cve.org/CVERecord?id=CVE-2020-10735].

El límite se aplica al número de caracteres de dígitos en la cadena de entrada o salida cuando estaría involucrado un algoritmo de conversión no lineal. Los guiones bajos y el signo no se cuentan para el límite.

Cuando una operación excede el límite, se lanza una excepción **`ValueError`**:

```
>>> import sys
>>> sys.set_int_max_str_digits(4300)  # Illustrative, th
>>> _ = int('2' * 5432)
Traceback (most recent call last):
...
ValueError: Exceeds the limit (4300 digits) for integer
>>> i = int('2' * 4300)
>>> len(str(i))
4300
>>> i_squared = i*i
>>> len(str(i_squared))
Traceback (most recent call last):
...
ValueError: Exceeds the limit (4300 digits) for integer
>>> len(hex(i_squared))
7144
>>> assert int(hex(i_squared), base=16) == i*i  # Hexade
```

El límite predeterminado es de 4300 dígitos como se indica en **`sys.int_info.default_max_str_digits`**. El límite más bajo que se puede configurar es de 640 dígitos como se indica en **`sys.int_info.str_digits_check_threshold`**.

Verificación:

```
>>> import sys
>>> assert sys.int_info.default_max_str_digits == 4300,
>>> assert sys.int_info.str_digits_check_threshold == 64
>>> msg = int('578966293710682886880994035146873798396722
...           '92529255143839154833338127435805497794361
...           '571186405732').to_bytes(53, 'big')
...
```

Added in version 3.11.

APIs afectadas

La limitación solo se aplica a conversiones potencialmente lentas entre **`int`** y **`str`** o **`bytes`**:

- `int(string)` con base predeterminada a 10.
- `int(string, base)` para todas las bases que no sean una potencia de 2.
- `str(integer)`.
- `repr(integer)`.
- cualquier otra conversión de cadena a base 10, por ejemplo, `f"{integer}"`, `"{}".format(integer)` o `b"%d" % integer`.

Las limitaciones no se aplican a funciones con un algoritmo lineal:

- `int(string, base)` con base 2, 4, 8, 16 o 32.
- **`int.from_bytes()`** y **`int.to_bytes()`**.
- **`hex()`**, **`oct()`**, **`bin()`**.
- Format specification mini-language para números hexadecimales, octales y binarios.
- **`str`** a **`float`**.
- **`str`** a **`decimal.Decimal`**.

Configuración del límite

Antes de que se inicie Python, puedes usar una variable de entorno o un indicador de línea de comandos del intérprete para configurar el límite:

- **`PYTHONINTMAXSTRDIGITS`**, por ejemplo, `PYTHONINTMAXSTRDIGITS=640 python3` para configurar el límite a 640 o `PYTHONINTMAXSTRDIGITS=0 python3` para desactivar la limitación.
- **`-X int_max_str_digits`**, por ejemplo, `python3 -X int_max_str_digits=640`
- **`sys.flags.int_max_str_digits`** contiene el valor de **`PYTHONINTMAXSTRDIGITS`** o **`-X int_max_str_digits`**. Si tanto la variable de entorno como la opción `-X` están configuradas, la opción `-X` tiene prioridad. Un valor de *-1* indica que ambas opciones no estaban configuradas, por lo que se utilizó un valor de **`sys.int_info.default_max_str_digits`** durante la inicialización.

Desde el código, puedes inspeccionar el límite actual y configurar uno nuevo al usar estas APIs de `sys`:

- `sys.get_int_max_str_digits()` y `sys.set_int_max_str_digits()` son un getter y un setter para el límite de todo el intérprete. Los subintérpretes tienen su propio límite.

La información sobre los valores predeterminados y mínimos se puede encontrar en `sys.int_info`:

- `sys.int_info.default_max_str_digits` es el límite predeterminado compilado.
- `sys.int_info.str_digits_check_threshold` es el valor más bajo aceptado para el límite (aparte de 0, que lo desactiva).

Added in version 3.11.

Prudencia

Configurar un límite bajo *puede* generar problemas. Si bien es raro, existe un código que contiene constantes enteras en decimal en su origen que excede el umbral mínimo. Una consecuencia de configurar el límite es que el código fuente de Python que contiene literales enteros decimales más grandes que el límite encontrará un error durante el análisis, generalmente en el momento de inicio o en el momento de importación o incluso en el momento de instalación - en cualquier momento y actualizado, `.pyc` no existe ya para el código. Una solución para la fuente que contiene constantes tan grandes es convertirlas a la forma hexadecimal `0x` ya que no tiene límite.

Prueba tu aplicación minuciosamente si utilizas un límite bajo. Asegúrate de que tus pruebas se ejecuten con el límite configurado temprano a través del entorno o del indicador para que se aplique durante el inicio e incluso durante cualquier paso de instalación que pueda invocar a Python para precompilar las fuentes `.py` a los archivos `.pyc`.

Configuración recomendada

Se espera que el valor predeterminado **`sys.int_info.default_max_str_digits`** sea razonable para la mayoría de las aplicaciones. Si su aplicación requiere un límite diferente, configúrelo desde el punto de entrada principal mediante un código independiente de la versión de Python, ya que estas API se agregaron en versiones de parches de seguridad anteriores a la 3.12.

Por ejemplo:

```
>>> import sys
>>> if hasattr(sys, "set_int_max_str_digits"):
...     upper_bound = 68000
...     lower_bound = 4004
...     current_limit = sys.get_int_max_str_digits()
...     if current_limit == 0 or current_limit > upper_b
...         sys.set_int_max_str_digits(upper_bound)
...     elif current_limit < lower_bound:
...         sys.set_int_max_str_digits(lower_bound)
```

Si necesitas deshabilitarlo por completo, configúralo en `0`.

Notas al pie

Se puede consultar información adicional sobre estos métodos especiales en el manual de referencia de Python (Personalización básica).

En consecuencia, la lista `[1, 2]` se considera igual que `[1.0, 2.0]`, y de forma similar para las tuplas.

Deben haberlo hecho, ya que el analizador no puede decir el tipo de operandos.

Los caracteres con versiones mayúsculas/minúsculas son aquellos cuya categoría general corresponde con «Lu» (Letra, Mayúscula), «Ll» (Letra, minúscula) o «Lt» (Letra, *titlecase*).

Para formatear solo una tupla se debe, por tanto, usar una tupla conteniendo un único elemento, que sería la tupla a ser formateada.

Excepciones incorporadas

En Python, todas las excepciones deben ser instancias de una clase que se derive de **`BaseException`**. En una instrucción **`try`** con una cláusula **`except`** que menciona una clase determinada, esa cláusula también controla las clases de excepción derivadas de esa clase (excepto las clases de excepción de las que esta se deriva). Dos clases de excepción que no están relacionadas mediante subclases nunca son equivalentes, incluso si tienen el mismo nombre.

Las excepciones incorporadas enumeradas en este apartado pueden ser lanzadas por el intérprete o funciones integradas. Excepto donde se mencione lo contrario, tienen un «valor asociado» que indica la causa detallada del error. Esto podría una cadena de caracteres o una tupla de varios elementos de información (por ejemplo, un código de error y una cadena que explica el código). El valor asociado generalmente se pasa como argumentos al constructor de la clase de excepción.

El código de usuario puede lanzar excepciones predefinidas. Esto puede ser usado para probar un gestor de excepciones o para notificar una condición de error «igual» a la situación en la cual el intérprete lance la misma excepción; pero tenga en cuenta que no hay nada que impida que el código del usuario produzca un error inapropiado.

Las clases de excepción predefinidas pueden ser usadas como subclases para definir otras nuevas; se recomienda a los programadores que deriven nuevas excepciones a partir de la clase **`Exception`** o de una de sus subclases, y no de **`BaseException`**. Hay disponible más información sobre la definición de excepciones en el Tutorial de Python en Excepciones definidas por el usuario.

Contexto de una excepción

Tres atributos en los objetos de excepción ofrecen información sobre el contexto en el cual la excepción fue lanzada:

BaseException._context_
BaseException._cause_
BaseException._suppress_context_

Al lanzar una nueva excepción mientras otra excepción está siendo manejada, el atributo `__context__` de la nueva excepción es automáticamente asignado a la excepción manejada. Una excepción puede ser manejada cuando se utiliza la cláusula `except` o `finally`, o con la sentencia `with`.

Este contexto de excepción implícita se puede complementar con una causa explícita usando `from` con `raise`:

```
raise new_exc from original_exc
```

La expresión que sigue a `from` debe ser una excepción o `None`. Se establecerá como `__cause__` en la excepción lanzada. La configuración de `__cause__` también establece implícitamente el atributo `__suppress_context__` a `True`, de modo que el uso de `raise new_exc from None` reemplaza efectivamente la excepción anterior con la nueva para fines de visualización (por ejemplo, conversión de `KeyError` a `AttributeError`), dejando la excepción anterior disponible en `__context__` para introspección durante la depuración.

La visualización por defecto de la traza de seguimiento muestra estas excepciones encadenadas además de la traza de la propia excepción. Siempre se muestra una excepción encadenada explícitamente en `__cause__` cuando está presente. Una excepción implícitamente encadenada en `__context__` solo se muestra si `__cause__` es `None` y `__suppress_context__` es falso.

En cualquier caso, la excepción en sí siempre se muestra después de cualquier excepción encadenada para que la línea final del seguimiento siempre muestre la última excepción que se generó.

Heredando de excepciones incorporadas

El código del usuario puede crear subclases que heredan de una excepción. Se recomienda heredar solo de una clase de tipo excepción a la vez para evitar cualquier conflicto posible en el manejo del atributo `args` por parte de las clases base, como también por posibles incompatibilidades en de la memoria.

La mayoría de las excepciones integradas están implementadas en C por eficiencia, vea: Objects/exceptions.c [https://github.com/python/cpython/tree/3.14/Objects/exceptions.c]. Algunas tienen distribuciones de memoria personalizadas, lo que hace imposible crear una subclase que herede de múltiples tipos de excepciones. La distribución de memoria de un tipo es un detalle de implementación y podría cambiar entre versiones de Python, llevando a nuevos conflictos en el futuro. Por lo tanto, se recomienda evitar la herencia de múltiples tipos de excepción.

Clases base

Las siguientes excepciones se utilizan principalmente como clases base para otras excepciones.

exception BaseException

La clase base para todas las excepciones predefinidas. No está pensada para ser heredada directamente por las clases definidas por el usuario (para eso, use **`Exception`**). Si se llama a **`str()`** en una instancia de esta clase, se retorna la representación de los argumentos de la instancia o la cadena vacía cuando no había argumentos.

args

La tupla de argumentos dada al constructor de excepción. Algunas excepciones predefinidas (como **`OSError`**) esperan un cierto número de argumentos y asignan un significado especial a los elementos de esta tupla, mientras que otras normalmente se llaman solo con una sola cadena que da un mensaje de error.

with_traceback(*tb*)

Este método establece *tb* como el nuevo rastreo para la excepción y retorna el objeto de excepción. Se usaba con más frecuencia antes de que las características de encadenamiento de excepciones de **PEP 3134** [https://peps.python.org/pep-3134/] estuvieran disponibles. El siguiente ejemplo muestra cómo podemos convertir una instancia de `SomeException` en una instancia de `OtherException` mientras se conserva el rastreo. Una vez lanzada, el marco actual se inserta en el rastreo del `OtherException`, como habría sucedido con el rastreo del `SomeException` original si hubiéramos permitido que se propagara al llamador.

```
try:
    ...
except SomeException:
    tb = sys.exception().__traceback__
    raise OtherException(...).with_traceback(t
```

traceback

Un campo escribible que mantenga los objetos de seguimiento de pila (traceback) asociados con esta excepción. Consulte La declaración raise para más información.

add_note(*note*)

Agrega la cadena `note` a las notas de la excepción, las cuales aparecen en el rastreo después de la cadena de la excepción. Se lanza un **`TypeError`** si `note` no es una cadena de caracteres.

Added in version 3.11.

notes

Una lista de las notas de esta excepción, las cuales fueron añadidas con el método **`add_note()`**. Este atributo es creado cuando se llama a **`add_note()`**.

Added in version 3.11.

exception Exception

Todas las excepciones predefinidas *non-system-exiting*, que no salen del sistema se derivan de esta clase. Todas las excepciones definidas por el usuario también deben derivarse de esta clase.

exception ArithmeticError

La clase base para las excepciones predefinidas que se lanzan para varios errores aritméticos: `OverflowError`, `ZeroDivisionError`, `FloatingPointError`.

exception BufferError

Se lanza cuando buffer no se puede realizar una operación relacionada.

exception LookupError

La clase base para las excepciones que se lanzan cuando una clave o índice utilizado en un mapa o secuencia que no es válido: `IndexError`, `KeyError`. Esto se puede lanzar directamente por `codecs.lookup()`.

Excepciones específicas

Las siguientes excepciones son las excepciones que normalmente se lanzan.

exception AssertionError

Se lanza cuando se produce un error en una instrucción `assert`.

exception AttributeError

Se lanza cuando se produce un error en una referencia de atributo (ver Referencias de atributos) o la asignación falla. (Cuando un objeto no admite referencias de atributos o asignaciones de atributos en absoluto, se lanza `TypeError`.)

The optional *name* and *obj* keyword-only arguments set the corresponding attributes:

name

The name of the attribute that was attempted to be accessed.

obj

The object that was accessed for the named attribute.

Distinto en la versión 3.10: Se agregaron los atributos **`name`** y **`obj`**.

exception EOFError

Raised when the **`input()`** function hits an end-of-file condition (EOF) without reading any data. (Note: the **`io.TextIOBase.read()`** and **`io.IOBase.readline()`** methods return an empty string when they hit EOF.)

exception FloatingPointError

No se usa actualmente.

exception GeneratorExit

Se lanza cuando un generator o coroutine está cerrado; ver **`generator.close()`** y **`coroutine.close()`**. Hereda directamente de **`BaseException`** en lugar de **`Exception`** ya que técnicamente no es un error.

exception ImportError

Se lanza cuando la instrucción **`import`** tiene problemas al intentar cargar un módulo. También se produce cuando la *from list* en `from ... import` tiene un nombre que no se puede encontrar.

Los argumentos opcionales *name* y *path* de solo palabras clave establecen los atributos correspondientes:

name

El nombre del módulo que se intentó importar.

path

La ruta a cualquier archivo que provocó la excepción.

Distinto en la versión 3.3: Se han añadido los atributos **`name`** y **`path`**.

exception ModuleNotFoundError

Una subclase de **`ImportError`** que se lanza mediante **`import`** cuando no se pudo encontrar un módulo. También se lanza cuando `None` se encuentra en **`sys.modules`**.

Added in version 3.6.

exception IndexError

Se lanza cuando un subíndice de secuencia está fuera del rango. (Los índices de la rebanada son truncados silenciosamente para caer en el intervalo permitido; si un índice no es un entero, se lanza **`TypeError`**.)

exception KeyError

Se lanza cuando no se encuentra una clave de asignación (diccionario) en el conjunto de claves existentes (mapa).

exception KeyboardInterrupt

Se lanza cuando el usuario pulsa la tecla de interrupción (normalmente `Control-C` o `Delete`). Durante la ejecución, se realiza una comprobación de interrupciones con regularidad. La excepción hereda de **`BaseException`** para no ser detectada de forma accidental por **`Exception`** y, por lo tanto, prevenir que el intérprete salga.

Nota

Capturar una **`KeyboardInterrupt`** requiere especial consideración. Debido a que puede ser lanzada en puntos impredecibles, puede, en algunas circunstancias, dejar al programa en ejecución en un estado inconsistente. Por lo general, es mejor permitir que **`KeyboardInterrupt`** termine el programa tan pronto como sea o evitar lanzarla por completo. (Vea Nota sobre Manejadores de Señales y Excepciones.)

exception MemoryError

Se lanza cuando una operación se queda sin memoria pero la situación aún puede ser recuperada (eliminando algunos objetos). El valor asociado es una cadena que indica que tipo de operación (interna) se quedó sin memoria. Tenga en cuenta que debido a la arquitectura de administración de memoria (función **`malloc()`** de C), el intérprete no siempre puede recuperarse completamente de esta situación; sin embargo, plantea una excepción para que se pueda imprimir un seguimiento de la pila, en caso de que un programa fuera de servicio fuera lo causa.

exception NameError

Se lanza cuando no se encuentra un nombre local o global. Esto se aplica solo a nombres no calificados. El valor asociado es un mensaje de error que incluye el nombre que no se pudo encontrar.

The optional *name* keyword-only argument sets the attribute:

name

The name of the variable that was attempted to be accessed.

Distinto en la versión 3.10: Se agregó el atributo **`name`**.

exception NotImplementedError

Esta excepción se deriva de **`RuntimeError`**. En las clases base definidas por el usuario, los métodos abstractos deberían lanzar esta excepción cuando requieren clases derivadas para anular el método, o mientras se desarrolla la clase para indicar que la implementación real aún necesita ser agregada.

Nota

No debe usarse para indicar que un operador o método no debe ser admitido en absoluto – en ese caso, deje el operador / método sin definir o, si es una subclase, se establece en `None`.

Prudencia

`NotImplementedError` and `NotImplemented` are not interchangeable. This exception should only be used as described above; see `NotImplemented` for details on correct usage of the built-in constant.

exception OSError([*arg*])

exception OSError(*errno*, *strerror*[, *filename*[, *winerror*[, *filename2*]]])

Esta excepción se produce cuando una función del sistema retorna un error relacionado con el sistema, que incluye fallas de E/S como `file not found` o `disk full` (no para tipos de argumentos ilegales u otros errores incidentales).

La segunda forma del constructor establece los atributos correspondientes, que se describen a continuación. Los atributos predeterminados son `None` si no se especifican. Para compatibilidad con versiones anteriores, si se pasan tres argumentos, el atributo `args` contiene solo una tupla de 2 de los dos primeros argumentos del constructor.

El constructor a menudo retorna una subclase de `OSError`, como se describe en OS exceptions a continuación. La subclase particular depende del valor final `errno`. Este comportamiento solo ocurre cuando se construye `OSError` directamente o mediante un alias, y no se hereda al derivar.

errno

Un código de error numérico de la variable C, `errno`.

winerror

En Windows, esto le proporciona el código de error nativo de Windows. El atributo `errno` es entonces una traducción aproximada, en términos POSIX, de ese código de error nativo.

En Windows, si el argumento del constructor `winerror` es un entero, el atributo `errno` se determina a partir del código de error de Windows y el argumento `errno` se ignora. En otras plataformas, el argumento `winerror` se ignora y el atributo `winerror` no existe.

strerror

The corresponding error message, as provided by the operating system. It is formatted by the C functions `perror()` under POSIX, and `FormatMessage()` under Windows.

filename
filename2

Para las excepciones que involucran una ruta del sistema de archivos (como `open()` o `os.unlink()`), `filename` es el nombre del archivo pasado a la función. Para las funciones que involucran dos rutas del sistema de archivos (como `os.rename()`), `filename2` corresponde al segundo nombre de archivo pasado a la función.

Distinto en la versión 3.3: `EnvironmentError`, `IOError`, `WindowsError`, `socket.error`, `select.error` and `mmap.error` have been merged into `OSError`, and the constructor may return a subclass.

Distinto en la versión 3.4: El atributo `filename` es ahora el nombre de archivo original pasado a la función, en lugar del nombre codificado o decodificado desde filesystem encoding and error handler. Además, se agregaron el argumento y atributo del constructor *filename2*.

exception OverflowError

Se lanza cuando el resultado de una operación aritmética es demasiado grande para ser representado. Esto no puede ocurrir para los enteros (para lo cual lanza un **`MemoryError`** en vez de darse por vencido). Sin embargo, por razones históricas, `OverflowError` a veces se lanza para enteros que están fuera del rango requerido. Debido a la falta de estandarización del manejo de excepciones de coma flotante en C, la mayoría de las operaciones de coma flotante no se verifican.

exception PythonFinalizationError

Esta excepción es derivada de **`RuntimeError`**. Se lanza cuando una operación es bloqueada durante el apagado del intérprete, también conocido como finalización de Python.

Ejemplos de operaciones que pueden ser bloqueadas con un **`PythonFinalizationError`** durante la finalización de Python:

- Creación de nuevos hilos de Python.
- **`Joining`** a running daemon thread.
- **`os.fork()`**.

Revise también la función **`sys.is_finalizing()`**.

Added in version 3.13: Anteriormente, se planteó un simple **`RuntimeError`**.

Distinto en la versión 3.14: **`threading.Thread.join()`** can now raise this exception.

exception RecursionError

Esta excepción se deriva de **`RuntimeError`**. Se lanza cuando el intérprete detecta que se excede la profundidad máxima de recursión (ver **`sys.getrecursionlimit()`**).

Added in version 3.5: Anteriormente, se planteó un simple `RuntimeError`.

exception ReferenceError

Esta excepción se produce cuando un proxy de referencia débil, creado por la función `weakref.proxy()`, se utiliza para acceder a un atributo del referente después de que se ha recolectado basura. Para obtener más información sobre referencias débiles, consulte el módulo `weakref` .

exception RuntimeError

Se lanza cuando se detecta un error que no corresponde a ninguna de las otras categorías. El valor asociado es una cadena que indica exactamente qué salió mal.

exception StopIteration

Lanzado por la función incorporada `next()` y un iterator's `__next__()` para indicar que no hay más elementos producidos por el iterador.

value

El objeto de excepción tiene un solo atributo `value`, que se proporciona como argumento al construir la excepción, y por defecto es `None`.

Cuando se retorna una función generator o coroutine, se lanza una nueva instancia `StopIteration`, y el valor retornado por la función se utiliza como parámetro `value` para constructor de la excepción.

Si un generador lanza directa o indirectamente `StopIteration`, se convierte en `RuntimeError` (conservando `StopIteration` como la causa de la nueva excepción).

Distinto en la versión 3.3: Se agregó el atributo **value** y la capacidad de las funciones del generador de usarlo para retornar un valor.

Distinto en la versión 3.5: Introdujo la transformación *RuntimeError* a través de `from __future__ import generator_stop`, ver **PEP 479** [https://peps.python.org/pep-0479/].

Distinto en la versión 3.7: Habilitar **PEP 479** [https://peps.python.org/pep-0479/] para todo el código por defecto: a **`StopIteration`** lanzado en un generador se transforma en **`RuntimeError`**.

exception StopAsyncIteration

Se debe lanzar mediante el método **`__anext__()`** de un objeto asynchronous iterator para detener la iteración.

Added in version 3.5.

exception SyntaxError(*message*, *details*)

Se lanza cuando el analizador encuentra un error de sintaxis. Esto puede ocurrir en una instrucción **`import`**, en una llamada a las funciones integradas **`compile()`**, **`exec()`** o **`eval()`**, o al leer el script inicial o la entrada estándar (también de forma interactiva).

El **`str()`** de la instancia de excepción retorna solo el mensaje de error. Detalles es una tupla cuyos miembros también están disponibles como atributos separados.

filename

El nombre del archivo en el que se produjo el error de sintaxis.

lineno

En qué número de línea del archivo se produjo el error. Tiene un índice 1: la primera línea del archivo tiene un `lineno` de 1.

offset

La columna de la línea donde ocurrió el error. Está indexado a 1: el primer carácter de la línea tiene un `offset` de 1.

text

El texto del código fuente involucrado en el error.

end_lineno

En qué número de línea del archivo termina el error. Está indexado a 1: la primera línea del archivo tiene un `lineno` de 1.

end_offset

Finaliza la columna de la línea final donde ocurrió el error. Está indexado a 1: el primer carácter de la línea tiene un `offset` de 1.

Para errores en los campos de cadena f, el mensaje tiene el prefijo «cadena f:» y las compensaciones son compensaciones en un texto construido a partir de la expresión de reemplazo. Por ejemplo, compilar f'Bad {ab} field "da como resultado este atributo args: (" f-string: … ", (" ", 1, 2," (ab) n ", 1, 5)) .

Distinto en la versión 3.10: Se agregaron los atributos **`end_lineno`** y **`end_offset`**.

exception IndentationError

Clase base para errores de sintaxis relacionados con sangría incorrecta. Esta es una subclase de **`SyntaxError`**.

exception TabError

Se lanza cuando la sangría contiene un uso inconsistente de pestañas y espacios. Esta es una subclase de **`IndentationError`**.

exception SystemError

Raised when the interpreter finds an internal error, but the situation does not look so serious to cause it to abandon all hope. The associated value is a string indicating what went wrong (in low-level terms). In CPython, this could be raised by incorrectly using Python's C API, such as returning a `NULL` value without an exception set.

If you're confident that this exception wasn't your fault, or the fault of a package you're using, you should report this to the author or maintainer of your Python interpreter. Be sure to report the version of the Python interpreter (`sys.version`; it is also printed at the start of an interactive Python session), the exact error message (the exception's associated value) and if possible the source of the program that triggered the error.

exception SystemExit

This exception is raised by the **`sys.exit()`** function. It inherits from **`BaseException`** instead of **`Exception`** so that it is not accidentally caught by code that catches **`Exception`**. This allows the exception to properly propagate up and cause the interpreter to exit. When it is not handled, the Python interpreter exits; no stack traceback is printed. The constructor accepts the same optional argument passed to **`sys.exit()`**. If the value is an integer, it specifies the system exit status (passed to C's **`exit()`** function); if it is `None`, the exit status is zero; if it has another type (such as a string), the object's value is printed and the exit status is one.

Una llamada **`sys.exit()`** se traduce en una excepción para que los gestores de limpieza (**`finally`** cláusulas de **`try`**) puedan ejecutarse, y para que un depurador pueda ejecutar un *script* sin correr el riesgo de perder el control. La función **`os._exit()`** se puede usar si es absolutamente necesario salir (por ejemplo, en el proceso secundario después de una llamada a **`os.fork()`**).

code

El estado de salida o mensaje de error que se pasa al constructor. (El valor predeterminado es `None`.)

exception TypeError

Se lanza cuando una operación o función se aplica a un objeto de tipo inapropiado. El valor asociado es una cadena que proporciona detalles sobre la falta de coincidencia de tipos.

El código de usuario puede lanzar esta excepción para indicar que un intento de operación en un objeto no es compatible y no debe serlo. Si un objeto está destinado a soportar una operación dada pero aún no ha proporcionado una implementación, **`NotImplementedError`** es la excepción adecuada para lanzar.

Pasar argumentos del tipo incorrecto (por ejemplo, pasar a **`list`** cuando se espera un **`int`**) debería dar como resultado **`TypeError`**, pero pasar argumentos con el valor incorrecto (por ejemplo, un número fuera límites esperados) debería dar como resultado **`ValueError`**.

exception UnboundLocalError

Se lanza cuando se hace referencia a una variable local en una función o método, pero no se ha vinculado ningún valor a esa variable. Esta es una subclase de **`NameError`**.

exception UnicodeError

Se lanza cuando se produce un error de codificación o decodificación relacionado con Unicode. Es una subclase de **`ValueError`**.

`UnicodeError` tiene atributos que describen el error de codificación o decodificación. Por ejemplo, `err.object[err.start:err.end]` proporciona la entrada inválida particular en la que falló el códec.

encoding

El nombre de la codificación que provocó el error.

reason

Una cadena que describe el error de códec específico.

object

El objeto que el códec intentaba codificar o decodificar.

start

El primer índice de datos no válidos en `object`.

This value should not be negative as it is interpreted as an absolute offset but this constraint is not enforced at runtime.

end

El índice después de los últimos datos no válidos en `object`.

This value should not be negative as it is interpreted as an absolute offset but this constraint is not enforced at runtime.

exception UnicodeEncodeError

Se lanza cuando se produce un error relacionado con Unicode durante la codificación. Es una subclase de `UnicodeError`.

exception UnicodeDecodeError

Se lanza cuando se produce un error relacionado con Unicode durante la codificación. Es una subclase de `UnicodeError`.

exception UnicodeTranslateError

Se lanza cuando se produce un error relacionado con Unicode durante la codificación. Es una subclase de `UnicodeError`.

exception ValueError

Se lanza cuando una operación o función recibe un argumento que tiene el tipo correcto pero un valor inapropiado, y la situación no se describe con una excepción más precisa como `IndexError`.

exception ZeroDivisionError

Se lanza cuando el segundo argumento de una operación de división o módulo es cero. El valor asociado es una cadena que indica el tipo de operandos y la operación.

Las siguientes excepciones se mantienen por compatibilidad con versiones anteriores; a partir de Python 3.3, son alias de `OSError`.

exception EnvironmentError

exception IOError

exception WindowsError

Solo disponible en Windows.

Excepciones del sistema operativo

Las siguientes excepciones son subclases de `OSError`, se lanzan según el código de error del sistema.

exception BlockingIOError

Se lanza cuando una operación se bloquearía en un objeto (ejemplo: *socket*) configurado para una operación no bloqueante. Corresponde a `errno` `EAGAIN`, `EALREADY`, `EWOULDBLOCK` y `EINPROGRESS`.

Además de los de `OSError`, `BlockingIOError` puede tener un atributo más:

characters_written

An integer containing the number of **bytes** written to the stream before it blocked. This attribute is available when using the buffered I/O classes from the `io` module.

exception ChildProcessError

Se lanza cuando falla una operación en un proceso secundario. Corresponde a `errno` `ECHILD`.

exception ConnectionError

Una clase base para problemas relacionados con la conexión.

Las subclases son `BrokenPipeError`, `ConnectionAbortedError`, `ConnectionRefusedError` y `ConnectionResetError`.

exception BrokenPipeError

Una subclase de `ConnectionError`, que se lanza cuando se intenta escribir en una tubería mientras el otro extremo se ha cerrado, o cuando se intenta escribir en un *socket* que se ha cerrado para escritura. Corresponde a `errno` `EPIPE` y `ESHUTDOWN`.

exception ConnectionAbortedError

Una subclase de `ConnectionError`, que se lanza cuando el par aborta un intento de conexión. Corresponde a `errno` `ECONNABORTED`.

exception ConnectionRefusedError

Una subclase de `ConnectionError`, que se lanza cuando el par rechaza un intento de conexión. Corresponde a `errno` `ECONNREFUSED`.

exception ConnectionResetError

Una subclase de `ConnectionError`, que se lanza cuando el par restablece una conexión. Corresponde a `errno` `ECONNRESET`.

exception FileExistsError

Se lanza al intentar crear un archivo o directorio que ya existe. Corresponde a `errno` `EEXIST`.

exception FileNotFoundError

Se lanza cuando se solicita un archivo o directorio pero no existe. Corresponde a `errno` `ENOENT`.

exception InterruptedError

Se lanza cuando una señal entrante interrumpe una llamada del sistema. Corresponde a `errno` `EINTR`.

Distinto en la versión 3.5: Python ahora vuelve a intentar las llamadas del sistema cuando una señal interrumpe un *syscall*, excepto si el gestor de señales lanza una excepción (ver **PEP 475** [https://peps.python.org/pep-0475/] para la justificación), en lugar de lanzar `InterruptedError`.

exception IsADirectoryError

Se lanza cuando se solicita una operación de archivo (como `os.remove()`) en un directorio. Corresponde a `errno` `EISDIR`.

exception NotADirectoryError

Se lanza cuando se solicita una operación de directorio (como `os.listdir()`) en algo que no es un directorio. En la mayoría de las plataformas POSIX, también se puede lanzar si una operación intenta abrir o recorrer un archivo que no es de directorio como si fuera un directorio. Corresponde a `errno` `ENOTDIR`.

exception PermissionError

Se lanza cuando se intenta ejecutar una operación sin los permisos de acceso adecuados, por ejemplo permisos del sistema de archivos. Corresponde a `errno` `EACCES`, `EPERM` y `ENOTCAPABLE`.

Distinto en la versión 3.11.1: El error `ENOTCAPABLE` de WASI ahora se mapea a `PermissionError`.

exception ProcessLookupError

Se lanza cuando un proceso dado no existe. Corresponde a `errno` `ESRCH`.

exception TimeoutError

Se lanza cuando se agota el tiempo de espera de una función del sistema a nivel del sistema. Corresponde a `errno` `ETIMEDOUT`.

Added in version 3.3: Por lo anterior se agregaron las subclases

`OSError`.

Ver también

PEP 3151 [https://peps.python.org/pep-3151/] - Reelaborando el sistema operativo y la jerarquía de excepciones E/S

Advertencias

Las siguientes excepciones se utilizan como categorías de advertencia; consulte la documentación de Categorías de advertencia para obtener más detalles.

exception Warning

Clase base para categorías de advertencia.

exception UserWarning

Clase base para advertencias generadas por código de usuario.

exception DeprecationWarning

Clase base para advertencias sobre características obsoletas cuando esas advertencias están destinadas a otros desarrolladores de Python.

Ignorado por los filtros de advertencia predeterminados, excepto en el módulo `__main__` (**PEP 565** [https://peps.python.org/pep-0565/]). Habilitando Modo Desarrollo de Python muestra esta advertencia.

La política de obsolescencia se describe en **PEP 387** [https://peps.python.org/pep-0387/].

exception PendingDeprecationWarning

Clase base para advertencias sobre características que están obsoletas y que se espera que sean obsoletas en el futuro, pero que no lo son en este momento.

Esta clase rara vez se usa para emitir una advertencia sobre una posible desaprobación próxima es inusual, y `DeprecationWarning` se prefiere para las desaprobaciones ya activas.

Ignorado por los filtros de advertencia predeterminados. Habilitando Modo Desarrollo de Python muestra esta advertencia.

La política de obsolescencia se describe en **PEP 387** [https://peps.python.org/pep-0387/].

exception SyntaxWarning

Clase base para advertencias sobre sintaxis dudosa.

This warning is typically emitted when compiling Python source code, and usually won't be reported when running already compiled code.

exception RuntimeWarning

Clase base para advertencias sobre comportamiento dudoso en tiempo de ejecución.

exception FutureWarning

Clase base para advertencias sobre características en desuso cuando esas advertencias están destinadas a usuarios finales de aplicaciones escritas en Python.

exception ImportWarning

Clase base para advertencias sobre posibles errores en la importación de módulos.

Ignorado por los filtros de advertencia predeterminados. Habilitando Modo Desarrollo de Python muestra esta advertencia.

exception UnicodeWarning

Clase base para advertencias relacionadas con Unicode.

exception EncodingWarning

Clase base para advertencias relacionadas con codificaciones.

Consulte EncodingWarning opcional para obtener más detalles.

Added in version 3.10.

exception BytesWarning

Clase base para advertencias relacionadas con **`bytes`** y **`bytearray`**.

exception ResourceWarning

Clase base para advertencias relacionadas con el uso de recursos.

Ignorado por los filtros de advertencia predeterminados. Habilitando Modo Desarrollo de Python muestra esta advertencia.

Added in version 3.2.

Grupos de excepciones

Las siguientes se utilizan cuando es necesario lanzar múltiples excepciones no relacionadas. Forman parte de la jerarquía de excepciones, por lo que pueden ser manejadas con **`except`** como todas las demás excepciones. Además, son reconocidas por **`except*`**, lo que hace coincidir sus subgrupos basándose en los tipos de las excepciones contenidas.

exception ExceptionGroup(*msg*, *excs*)

exception BaseExceptionGroup(*msg*, *excs*)

Ambos tipos de excepción envuelven las excepciones en la secuencia `excs`. El parámetro `msg` debe ser una cadena de caracteres. La diferencia entre las dos clases es que **`BaseExceptionGroup`** extiende a **`BaseException`** y puede envolver cualquier excepción, mientras que **`ExceptionGroup`** extiende a **`Exception`** y solo puede envolver subclases de **`Exception`**. Este diseño está pensado para que `except Exception` capture un **`ExceptionGroup`** pero no **`BaseExceptionGroup`**.

El constructor **`BaseExceptionGroup`** retorna una **`ExceptionGroup`** en lugar de una **`BaseExceptionGroup`** si todas las excepciones contenidas son instancias de **`Exception`** , por lo que puede utilizarse para hacer la selección automática. El constructor **`ExceptionGroup`**, por su parte, lanza un **`TypeError`** si alguna de las excepciones contenidas no es una subclase de **`Exception`**.

Detalles de implementación de CPython: The `excs` parameter may be any sequence, but lists and tuples are specifically processed more efficiently here. For optimal performance, pass a tuple as `excs`.

message

> El argumento `msg` para el constructor. Este atributo es de sólo lectura.

exceptions

> Una tupla de la excepciones en la secuencia `excs` pasada al constructor. Este atributo es de sólo lectura.

subgroup(*condition*)

> Retorna un grupo de excepciones que contiene sólo las excepciones del grupo actual que cumplen con *condition*, o `None` si el resultado está vacío.
>
> La condición puede ser un tipo de excepción o una tupla de tipos de excepción, en cuyo caso cada excepción se verifica para comprobar si coincide utilizando la misma verificación que es usada en una cláusula *except*. La condición también puede ser un invocable (distinto de un tipo objeto) que acepte una excepción como su único argumento, y retorne *true*

para las excepciones que deban estar en el subgrupo.

La estructura de anidamiento de la excepción actual se conserva en el resultado, así como también los valores de sus campos **`message`**, **`__traceback__`**, **`__cause__`**, **`__context__`** y **`__notes__`** . Los grupos anidados vacíos son omitidos del resultado.

La condición se comprueba para todas las excepciones del grupo de excepción anidado, incluyendo el nivel superior y cualquier grupo de excepción anidado. Si la condición es verdadera para dicho grupo de excepción, se incluye en el resultado en su totalidad.

Added in version 3.13: `condition` puede ser cualquier invocable que no sea un objeto tipo.

split(*condition*)

Al igual que **`subgroup()`**, pero retorna el par `(match, rest)` donde `match` es `subgroup(condition)` y `rest` es la parte restante que no coincide.

derive(*excs*)

Retorna un grupo de excepción con los mismos **`message`**, pero que envuelve las excepciones en `excs`.

Este método es usado por **`subgroup()`** y **`split()`**, los cuales se usan en varios contextos para romper un grupo de excepción. Se necesita una subclase que lo sobrescriba para que **`subgroup()`** y **`split()`** retornen instancias de la subclase en lugar de **`ExceptionGroup`**.

`subgroup()` y **`split()`** copian los campos **`__traceback__`**, **`__cause__`**, **`__context__`** y **`__notes__`** del grupo de excepción original al retornado por **`derive()`**, por lo que estos campos no necesitan ser actualizados por **`derive()`**.

```
>>> class MyGroup(ExceptionGroup):
...     def derive(self, excs):
...         return MyGroup(self.message, excs)
...
>>> e = MyGroup("eg", [ValueError(1), TypeError
>>> e.add_note("a note")
>>> e.__context__ = Exception("context")
>>> e.__cause__ = Exception("cause")
>>> try:
...    raise e
... except Exception as e:
...    exc = e
...
>>> match, rest = exc.split(ValueError)
>>> exc, exc.__context__, exc.__cause__, exc.__
(MyGroup('eg', [ValueError(1), TypeError(2)]),
>>> match, match.__context__, match.__cause__,
(MyGroup('eg', [ValueError(1)]), Exception('con
>>> rest, rest.__context__, rest.__cause__, res
(MyGroup('eg', [TypeError(2)]), Exception('cont
>>> exc.__traceback__ is match.__traceback__ is
True
```

Note that **`BaseExceptionGroup`** defines `__new__()`, so subclasses that need a different constructor signature need to override that rather than `__init__()`. For example, the following defines an exception group subclass which accepts an exit_code and constructs the group's message from it.

```
class Errors(ExceptionGroup):
   def __new__(cls, errors, exit_code):
      self = super().__new__(Errors, f"exit code: {
      self.exit_code = exit_code
      return self

   def derive(self, excs):
      return Errors(excs, self.exit_code)
```

Al igual que **`ExceptionGroup`**, cualquier subclase de

`BaseExceptionGroup` que también es una subclase de **`Exception`** sólo puede envolver instancias de **`Exception`**.

Added in version 3.11.

Jerarquía de excepción

La jerarquía de clases para las excepciones incorporadas es:

```
BaseException
 ├── BaseExceptionGroup
 ├── GeneratorExit
 ├── KeyboardInterrupt
 ├── SystemExit
 └── Exception
      ├── ArithmeticError
      │    ├── FloatingPointError
      │    ├── OverflowError
      │    └── ZeroDivisionError
      ├── AssertionError
      ├── AttributeError
      ├── BufferError
      ├── EOFError
      ├── ExceptionGroup [BaseExceptionGroup]
      ├── ImportError
      │    └── ModuleNotFoundError
      ├── LookupError
      │    ├── IndexError
      │    └── KeyError
      ├── MemoryError
      ├── NameError
      │    └── UnboundLocalError
      ├── OSError
      │    ├── BlockingIOError
      │    ├── ChildProcessError
      │    ├── ConnectionError
      │    │    ├── BrokenPipeError
      │    │    ├── ConnectionAbortedError
```

```
                    ConnectionRefusedError
                    ConnectionResetError
               FileExistsError
               FileNotFoundError
               InterruptedError
               IsADirectoryError
               NotADirectoryError
               PermissionError
               ProcessLookupError
               TimeoutError
          ReferenceError
          RuntimeError
               NotImplementedError
               PythonFinalizationError
               RecursionError
          StopAsyncIteration
          StopIteration
          SyntaxError
               IndentationError
                    TabError
          SystemError
          TypeError
          ValueError
               UnicodeError
                    UnicodeDecodeError
                    UnicodeEncodeError
                    UnicodeTranslateError
          Warning
               BytesWarning
               DeprecationWarning
               EncodingWarning
               FutureWarning
               ImportWarning
               PendingDeprecationWarning
               ResourceWarning
               RuntimeWarning
               SyntaxWarning
               UnicodeWarning
```

UserWarning

Thread Safety Guarantees

This page documents thread-safety guarantees for built-in types in Python's free-threaded build. The guarantees described here apply when using Python with the GIL disabled (free-threaded mode). When the GIL is enabled, most operations are implicitly serialized.

For general guidance on writing thread-safe code in free-threaded Python, see Python support for free threading.

Thread safety levels

The C API documentation uses the following levels to describe the thread safety guarantees of each function. The levels are listed from least to most safe.

Incompatible

A function or operation that cannot be made safe for concurrent use even with external synchronization. Incompatible code typically accesses global state in an unsynchronized way and must only be called from a single thread throughout the program's lifetime.

Example: a function that modifies process-wide state such as signal handlers or environment variables, where concurrent calls from any threads, even with external locking, can conflict with the runtime or other libraries.

Compatible

A function or operation that is safe to call from multiple threads *provided* the caller supplies appropriate external synchronization, for example by holding a lock for the duration of each call. Without such synchronization, concurrent calls may produce race conditions or data races.

Example: a function that reads from or writes to an object whose internal state is not protected by a lock. Callers must ensure that no two threads access the same object at the same time.

Safe on distinct objects

A function or operation that is safe to call from multiple threads without external synchronization, as long as each thread operates on a **different** object. Two threads may call the function at the same time, but they must not pass the same object (or objects that share underlying state) as arguments.

Example: a function that modifies fields of a struct using non-atomic writes. Two threads can each call the function on their own struct instance safely, but concurrent calls on the *same* instance require external synchronization.

Safe on shared objects

A function or operation that is safe for concurrent use on the **same** object. The implementation uses internal synchronization (such as per-object locks or critical sections) to protect shared mutable state, so callers do not need to supply their own locking.

Example: `PyList_GetItemRef()` can be called from multiple threads on the same `PyListObject` - it uses internal synchronization to serialize access.

Atomic

A function or operation that appears atomic with respect to other threads - it executes instantaneously from the perspective of other threads. This is the strongest form of thread safety.

Example: `PyMutex_IsLocked()` performs an atomic read of the mutex state and can be called from any thread at any time.

Thread safety for list objects

Reading a single element from a `list` is atomic:

```
lst[i]   # list.__getitem__
```

The following methods traverse the list and use atomic reads of each item to perform their function. That means that they may return results affected by concurrent modifications:

```
item in lst
lst.index(item)
lst.count(item)
```

All of the above operations avoid acquiring per-object locks. They do not block concurrent modifications. Other operations that hold a lock will not block these from observing intermediate states.

All other operations from here on block using the per-object lock.

Writing a single item via `lst[i] = x` is safe to call from multiple threads and will not corrupt the list.

The following operations return new objects and appear atomic to other threads:

```
lst1 + lst2    # concatenates two lists into a new list
x * lst        # repeats lst x times into a new list
lst.copy()     # returns a shallow copy of the list
```

The following methods that only operate on a single element with no shifting required are atomic:

```
lst.append(x)  # append to the end of the list, no shift
lst.pop()      # pop element from the end of the list, r
```

The **`clear()`** method is also atomic. Other threads cannot observe elements being removed.

The **`sort()`** method is not atomic. Other threads cannot observe intermediate states during sorting, but the list appears empty for the duration of the sort.

The following operations may allow lock-free operations to observe intermediate states since they modify multiple elements in place:

```
lst.insert(idx, item)  # shifts elements
lst.pop(idx)           # idx not at the end of the list,
lst *= x               # copies elements in place
```

The `remove()` method may allow concurrent modifications since element comparison may execute arbitrary Python code (via `__eq__()`).

`extend()` is safe to call from multiple threads. However, its guarantees depend on the iterable passed to it. If it is a `list`, a `tuple`, a `set`, a `frozenset`, a `dict` or a dictionary view object (but not their subclasses), the `extend` operation is safe from concurrent modifications to the iterable. Otherwise, an iterator is created which can be concurrently modified by another thread. The same applies to inplace concatenation of a list with other iterables when using `lst += iterable`.

Similarly, assigning to a list slice with `lst[i:j] = iterable` is safe to call from multiple threads, but `iterable` is only locked when it is also a `list` (but not its subclasses).

Operations that involve multiple accesses, as well as iteration, are never atomic. For example:

```
# NOT atomic: read-modify-write
lst[i] = lst[i] + 1

# NOT atomic: check-then-act
if lst:
    item = lst.pop()

# NOT thread-safe: iteration while modifying
for item in lst:
    process(item)  # another thread may modify lst
```

Consider external synchronization when sharing `list` instances across threads.

Thread safety for dict objects

Creating a dictionary with the `dict` constructor is atomic when the argument to it is a `dict` or a `tuple`. When using the `dict.fromkeys()` method, dictionary creation is atomic when the argument is a `dict`, `tuple`, `set` or `frozenset`.

The following operations and functions are lock-free and atomic.

```
d[key]         # dict.__getitem__
d.get(key)     # dict.get
key in d       # dict.__contains__
len(d)         # dict.__len__
```

All other operations from here on hold the per-object lock.

Writing or removing a single item is safe to call from multiple threads and will not corrupt the dictionary:

```
d[key] = value           # write
del d[key]               # delete
d.pop(key)               # remove and return
d.popitem()              # remove and return last item
d.setdefault(key, v)     # insert if missing
```

These operations may compare keys using `__eq__()`, which can execute arbitrary Python code. During such comparisons, the dictionary may be modified by another thread. For built-in types like `str`, `int`, and `float`, that implement `__eq__()` in C, the underlying lock is not released during comparisons and this is not a concern.

The following operations return new objects and hold the per-object lock for the duration of the operation:

```
d.copy()      # returns a shallow copy of the dictionary
d | other     # merges two dicts into a new dict
d.keys()      # returns a new dict_keys view object
d.values()    # returns a new dict_values view object
d.items()     # returns a new dict_items view object
```

The `clear()` method holds the lock for its duration. Other threads cannot observe elements being removed.

The following operations lock both dictionaries. For `update()` and `|=`, this applies only when the other operand is a `dict` that uses the standard dict iterator (but not subclasses that override iteration). For equality comparison, this applies to **`dict`** and its subclasses:

```
d.update(other_dict)   # both locked when other_dict is a
d |= other_dict        # both locked when other_dict is a
d == other_dict        # both locked for dict and subclas
```

All comparison operations also compare values using `__eq__()`, so for non-built-in types the lock may be released during comparison.

`fromkeys()` locks both the new dictionary and the iterable when the iterable is exactly a `dict`, `set`, or `frozenset` (not subclasses):

```
dict.fromkeys(a_dict)      # locks both
dict.fromkeys(a_set)       # locks both
dict.fromkeys(a_frozenset) # locks both
```

When updating from a non-dict iterable, only the target dictionary is locked. The iterable may be concurrently modified by another thread:

```
d.update(iterable)         # iterable is not a dict: only
d |= iterable              # iterable is not a dict: only
dict.fromkeys(iterable)    # iterable is not a dict/set/f
```

Operations that involve multiple accesses, as well as iteration, are never atomic:

```
# NOT atomic: read-modify-write
d[key] = d[key] + 1

# NOT atomic: check-then-act (TOCTOU)
if key in d:
    del d[key]

# NOT thread-safe: iteration while modifying
```

```
for key, value in d.items():
    process(key)  # another thread may modify d
```

To avoid time-of-check to time-of-use (TOCTOU) issues, use atomic operations or handle exceptions:

```
# Use pop() with default instead of check-then-delete
d.pop(key, None)

# Or handle the exception
try:
    del d[key]
except KeyError:
    pass
```

To safely iterate over a dictionary that may be modified by another thread, iterate over a copy:

```
# Make a copy to iterate safely
for key, value in d.copy().items():
    process(key)
```

Consider external synchronization when sharing `dict` instances across threads.

Thread safety for set objects

The `len()` function is lock-free and atomic.

The following read operation is lock-free. It does not block concurrent modifications and may observe intermediate states from operations that hold the per-object lock:

```
elem in s    # set.__contains__
```

This operation may compare elements using `__eq__()`, which can execute arbitrary Python code. During such comparisons, the set may be modified by another thread. For built-in types like `str`, `int`, and `float`, `__eq__()` does not release the underlying lock during comparisons and this is not a concern.

All other operations from here on hold the per-object lock.

Adding or removing a single element is safe to call from multiple threads and will not corrupt the set:

```
s.add(elem)      # add element
s.remove(elem)   # remove element, raise if missing
s.discard(elem)  # remove element if present
s.pop()          # remove and return arbitrary element
```

These operations also compare elements, so the same `__eq__()` considerations as above apply.

The `copy()` method returns a new object and holds the per-object lock for the duration so that it is always atomic.

The `clear()` method holds the lock for its duration. Other threads cannot observe elements being removed.

The following operations only accept `set` or `frozenset` as operands and always lock both objects:

```
s |= other                   # other must be set/frozens
s &= other                   # other must be set/frozens
s -= other                   # other must be set/frozens
s ^= other                   # other must be set/frozens
s & other                    # other must be set/frozens
s | other                    # other must be set/frozens
s - other                    # other must be set/frozens
s ^ other                    # other must be set/frozens
```

`set.update()`, `set.union()`, `set.intersection()` and `set.difference()` can take multiple iterables as arguments. They all iterate through all the passed iterables and do the following:

- `set.update()` and `set.union()` lock both objects only when
  - the other operand is a `set`, `frozenset`, or `dict`.

- `set.intersection()` and `set.difference()` always try to lock all objects.

`set.symmetric_difference()` tries to lock both objects.

The update variants of the above methods also have some differences between them:

- `set.difference_update()` and `set.intersection_update()` try to lock all objects one-by-one.
- `set.symmetric_difference_update()` only locks the arguments if it is of type `set`, `frozenset`, or `dict`.

The following methods always try to lock both objects:

```
s.isdisjoint(other)          # both locked
s.issubset(other)            # both locked
s.issuperset(other)          # both locked
```

Operations that involve multiple accesses, as well as iteration, are never atomic:

```
# NOT atomic: check-then-act
if elem in s:
      s.remove(elem)

# NOT thread-safe: iteration while modifying
for elem in s:
      process(elem)  # another thread may modify s
```

Consider external synchronization when sharing `set` instances across threads. See Python support for free threading for more information.

Thread safety for bytearray objects

The `len()` function is lock-free and atomic.

Concatenation and comparisons use the buffer protocol, which prevents resizing but does not hold the per-object lock. These operations may observe intermediate states from concurrent modifications:

```
ba + other     # may observe concurrent writes
ba == other    # may observe concurrent writes
ba < other     # may observe concurrent writes
```

All other operations from here on hold the per-object lock.

Reading a single element or slice is safe to call from multiple threads:

```
ba[i]         # bytearray.__getitem__
ba[i:j]       # slice
```

The following operations are safe to call from multiple threads and will not corrupt the bytearray:

```
ba[i] = x           # write single byte
ba[i:j] = values    # write slice
ba.append(x)        # append single byte
ba.extend(other)    # extend with iterable
ba.insert(i, x)     # insert single byte
ba.pop()            # remove and return last byte
ba.pop(i)           # remove and return byte at inde
ba.remove(x)        # remove first occurrence
ba.reverse()        # reverse in place
ba.clear()          # remove all bytes
```

Slice assignment locks both objects when *values* is a `bytearray`:

```
ba[i:j] = other_bytearray  # both locked
```

The following operations return new objects and hold the per-object lock for the duration:

```
ba.copy()      # returns a shallow copy
ba * n         # repeat into new bytearray
```

The membership test holds the lock for its duration:

```
x in ba        # bytearray.__contains__
```

All other bytearray methods (such as **`find()`**, **`replace()`**, **`split()`**, **`decode()`**, etc.) hold the per-object lock for their duration.

Operations that involve multiple accesses, as well as iteration, are never atomic:

```
# NOT atomic: check-then-act
if x in ba:
    ba.remove(x)

# NOT thread-safe: iteration while modifying
for byte in ba:
    process(byte)  # another thread may modify ba
```

To safely iterate over a bytearray that may be modified by another thread, iterate over a copy:

```
# Make a copy to iterate safely
for byte in ba.copy():
    process(byte)
```

Consider external synchronization when sharing **`bytearray`** instances across threads. See Python support for free threading for more information.

Thread safety for memoryview objects

`memoryview` objects provide access to the internal data of an underlying object without copying. Thread safety depends on both the memoryview itself and the underlying buffer exporter.

The memoryview implementation uses atomic operations to track its own exports in the free-threaded build. Creating and releasing a memoryview are thread-safe. Attribute access (e.g., **`shape`**, **`format`**) reads fields that are immutable for the lifetime of the memoryview, so concurrent reads are safe as long as the memoryview has not been released.

However, the actual data accessed through the memoryview is owned by the underlying object. Concurrent access to this data is only safe if the underlying object supports it:

- For immutable objects like **`bytes`**, concurrent reads through multiple memoryviews are safe.
- For mutable objects like **`bytearray`**, reading and writing the same memory region from multiple threads without external synchronization is not safe and may result in data corruption. Note that even read-only memoryviews of mutable objects do not prevent data races if the underlying object is modified from another thread.

```
# NOT safe: concurrent writes to the same buffer
data = bytearray(1000)
view = memoryview(data)
# Thread 1: view[0:500] = b'x' * 500
# Thread 2: view[0:500] = b'y' * 500

# Safe: use a lock for concurrent access
import threading
lock = threading.Lock()
data = bytearray(1000)
view = memoryview(data)

with lock:
    view[0:500] = b'x' * 500
```

Resizing or reallocating the underlying object (such as calling **`bytearray.resize()`**) while a memoryview is exported raises **`BufferError`**. This is enforced regardless of threading.

Servicios de procesamiento de texto

Los módulos descritos en este capítulo proporcionan una amplia gama de operaciones de manipulación de cadenas de texto y otros servicios de procesamiento de texto.

El módulo **`codecs`** descrito en Servicios de datos binarios también es muy relevante para el procesamiento de texto. Además, consulta la documentación para el tipo *string* de Python en Cadenas de caracteres — str.

- **`string`** — Common string operations
    - Constantes de cadenas
    - Custom string formatting
    - Format string syntax
        - Format specification mini-language
        - Ejemplos de formateo
    - Template strings ($-strings)
    - Funciones de ayuda
- **`string.templatelib`** — Support for template string literals
    - Template strings
    - Types
    - Helper functions
- **`re`** — Regular expression operations
    - Sintaxis de expresiones regulares
    - Contenidos del módulo
        - Indicadores

        - Funciones
        - Excepciones
    - Objetos expresión regular
    - Objetos de coincidencia
    - Ejemplos de expresiones regulares
        - Buscando un par
        - Simular scanf()
        - search() vs. match()
        - Haciendo una guía telefónica
        - Mungear texto
        - Encontrar todos los adverbios
        - Encontrar todos los adverbios y sus posiciones
        - Notación de cadena *raw*
        - Escribir un Tokenizador
- **`difflib`** — Helpers for computing deltas
    - SequenceMatcher objects
    - SequenceMatcher examples
    - Differ objects
    - Differ example
    - Una interfaz de línea de comandos para **`difflib`**
    - ndiff example
- **`textwrap`** — Text wrapping and filling
- **`unicodedata`** — Unicode Database
- **`stringprep`** — Internet String Preparation
- **`readline`** — GNU readline interface
    - Archivo de inicio
    - Búfer de línea
    - Archivo de historial
    - Lista del historial
    - Ganchos (*hooks*) de inicialización
    - Terminación
    - Ejemplo
- **`rlcompleter`** — Completion function for GNU readline

`string` — Common string operations

Source code: Lib/string/__init__.py [https://github.com/python/cpython/tree/3.14/Lib/string/__init__.py]

Ver también

Cadenas de caracteres — str

Métodos de las cadenas de caracteres

Constantes de cadenas

Las constantes definidas en este módulo son:

string.ascii_letters

La concatenación de las constantes abajo descriptas **`ascii_lowercase`** y **`ascii_uppercase`**. Este valor es independiente de la configuración regional.

string.ascii_lowercase

Las letras minúsculas `'abcdefghijklmnopqrstuvwxyz'`. Este valor es independiente de la configuración regional y no cambiará.

string.ascii_uppercase

Las letras mayúsculas `'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`. Este valor es independiente de la configuración regional y no cambiará.

string.digits

La cadena `'0123456789'`.

string.hexdigits

La cadena `'0123456789abcdefABCDEF'`.

string.octdigits

La cadena de caracteres `'01234567'`.

string.punctuation

Cadena de caracteres ASCII que se consideran caracteres de puntuación en la configuración regional `C`: ``!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~``.

string.printable

String of ASCII characters which are considered printable by Python. This is a combination of **`digits`**, **`ascii_letters`**, **`punctuation`**, and **`whitespace`**.

Nota

By design, **`string.printable.isprintable()`** returns **`False`**. In particular, `string.printable` is not printable in the POSIX sense (see *LC_CTYPE* *[https://manpages.debian.org/locale(5)]*).

string.whitespace

Una cadena cuyos caracteres ASCII se consideran todos espacios en blanco. Esto incluye los caracteres espacio, tabulador, salto de línea, retorno, salto de página y tabulador vertical.

Custom string formatting

The built-in string class provides the ability to do complex variable substitutions and value formatting via the **`format()`** method described in **PEP 3101** [https://peps.python.org/pep-3101/]. The

`Formatter` class in the `string` module allows you to create and customize your own string formatting behaviors using the same implementation as the built-in `format()` method.

class string.Formatter

La clase `Formatter` tiene los siguientes métodos públicos:

format(*format_string*, /, **args*, ***kwargs*)

Método principal de la API. Recibe una cadena de formato y argumentos posicionales y de palabra clave arbitrarios. Es sólo un envoltorio que llama a `vformat()`.

Distinto en la versión 3.7: Un argumento de cadena de formato ahora es solo posicional.

vformat(*format_string*, *args*, *kwargs*)

Esta función realiza es la que realmente hace el trabajo de formateo. Se expone como una función independiente para los casos en los que desea pasar un diccionario predefinido de argumentos, en lugar de desempaquetar y volver a empaquetar el diccionario como argumentos individuales mediante la sintaxis `*args` y `**kwargs`. `vformat()` hace el trabajo de dividir la cadena de formato en datos de caracteres y campos de reemplazo. Llama a los diversos métodos descritos a continuación.

Además de eso, la clase `Formatter` define varios métodos que se espera sean reemplazados por las subclases:

parse(*format_string*)

Itera sobre *format_string* y retorna un iterable de tuplas (*literal_text*, *field_name*, *format_spec*, *conversion*). Es usado por `vformat()` para dividir la cadena de caracteres en texto literal o en campos de reemplazo.

The values in the tuple conceptually represent a span of literal text followed by a single replacement field. If there is no literal text (which can happen if two replacement fields occur consecutively), then *literal_text* will be a zero-length string. If there is no replacement field, then the values of *field_name*, *format_spec* and *conversion* will be `None`. The value of *field_name* is unmodified and auto-numbering of non-numbered positional fields is done by **`vformat()`**.

get_field(*field_name*, *args*, *kwargs*)

Given *field_name*, convert it to an object to be formatted. Auto-numbering of *field_name* returned from **`parse()`** is done by **`vformat()`** before calling this method. Returns a tuple (obj, used_key). The default version takes strings of the form defined in **PEP 3101** [https://peps.python.org/pep-3101/], such as «0[name]» or «label.title». *args* and *kwargs* are as passed in to **`vformat()`**. The return value *used_key* has the same meaning as the *key* parameter to **`get_value()`**.

get_value(*key*, *args*, *kwargs*)

Recuperar un valor de campo determinado. El argumento *key* será un entero o una cadena de caracteres. Si es un entero, representa el índice del argumento posicional en *args*; si es una cadena, representa un argumento definido en *kwargs*.

El parámetro *args* se establece como lista de argumentos posicionales en **`vformat()`**, y el parámetro *kwargs* se establece como diccionario de argumentos de palabra clave.

Para nombres de campo compuesto, estas funciones son únicamente llamadas para el primer componente del campo. Los componentes que le siguen son tratados a través de operaciones normales de atributo e indexación.

Por ejemplo, la expresión de campo “0.name” haría que **`get_value()`** se llame con un argumento *key* igual a

0. El atributo “”name”” se buscará después del retorno de `get_value()` llamando a la función incorporada `getattr()`.

Si el índice o la palabra clave hace referencia a un elemento que no existe, se debe generar un `IndexError` o un `KeyError`.

check_unused_args(*used_args*, *args*, *kwargs*)

Implementa el chequeo de argumentos no utilizados si así se desea. Los argumentos de esta función son el conjunto de todas las claves de argumento a las que se hizo referencia en la cadena de formato (enteros para argumentos posicionales y cadenas de caracteres para argumentos con nombre) y una referencia a los *args* y *kwargs* que se pasaron a *vformat*. El conjunto de *args* no utilizados se puede calcular a partir de estos parámetros. se asume que `check_unused_args()` genera una excepción si se produce un error en el chequeo.

format_field(*value*, *format_spec*)

`format_field()` simplemente llama a la función incorporada `format()`. El método se proporciona para que las subclases puedan sobrescribirlo.

convert_field(*value*, *conversion*)

Convierte el valor (retornado por `get_field()`) dado un tipo de conversión (tal como la tupla retornada por el método `parse()`). La versión por defecto entiende los tipos de conversión “s” (str), “r” (repr) y “a” (ascii).

Format string syntax

The `str.format()` method and the `Formatter` class share the same syntax for format strings (although in the case of **`Formatter`**, subclasses can define their own format string syntax). The syntax is related to that of formatted string literals and template string literals, but it is less sophisticated and, in particular, does not support arbitrary expressions in interpolations.

Las cadenas de caracteres de formato contienen «campos de reemplazo» rodeados de llaves `{}`. Todo lo que no está contenido entre llaves se considera texto literal, que se copia sin cambios en la salida. Si se necesita incluir un carácter de llave en el texto literal, se puede escapar duplicando: `{{` *and* `}}`.

La gramática para un campo de reemplazo es la siguiente:

```
replacement_field: "{" [field_name] ["!" conversion] [":
field_name:        arg_name ("." attribute_name | "[" el
arg_name:          [identifier | digit+]
attribute_name:    identifier
element_index:     digit+ | index_string
index_string:      <any source character except "]"> +
conversion:        "r" | "s" | "a"
format_spec:       format-spec:format_spec
```

En términos menos formales, el campo de reemplazo puede comenzar con un *field_name* (nombre de campo) que especifica el objeto cuyo valor se va a formatear e insertar en la salida en lugar del campo de reemplazo. El nombre de campo (*field_name*) va seguido opcionalmente de un campo *conversion* (conversión), que va precedido de un signo de exclamación `'!'`, y un *format_spec*, que va precedido de dos puntos `':'`. Estos especifican un formato no predeterminado para el valor de reemplazo.

Véase también la sección Format specification mini-language.

El *field_name* (nombre de campo) comienza con un *arg_name* que es un número o una palabra clave. Si es un número, hace referencia a un argumento posicional y, si es una palabra clave, hace referencia a un argumento de palabra clave. Un *arg_name* se trata como un número si una llamada a `str.isdecimal()` sobre la cadena retorna verdadero. Si los *arg_names* numéricos en una cadena de caracteres de formato son una secuencia como 0, 1, 2, …, todos pueden ser omitidos (no sólo algunos) y los números 0, 1, 2, … se insertarán automáticamente en ese orden. Dado que *arg_name* no está delimitado por comillas, no es posible especificar claves de diccionario arbitrarias (por ejemplo, las cadenas `'10'` or `':-]'`) dentro de una cadena de caracteres de formato. El *arg_name* puede ir seguido de cualquier número de expresiones de índice o atributo. Una expresión con forma `'.name'` selecciona el atributo con nombre mediante **`getattr()`**, mientras que una expresión con forma `'[index]'` realiza una búsqueda de índice mediante **`__getitem__()`**.

Distinto en la versión 3.1: Los especificadores de argumentos posicionales pueden ser omitidos en **`str.format()`**, así `'{} {}'.format(a, b)` es equivalente a `'{0} {1}'.format(a, b)`.

Distinto en la versión 3.4: Para la clase **`Formatter`**, los especificadores de argumento posicional pueden ser omitidos.

Algunos ejemplos simples de cadena de formato:

```
"First, thou shalt count to {0}"  # References first pos
"Bring me a {}"                   # Implicitly reference
"From {} to {}"                   # Same as "From {0} to
"My quest is {name}"              # References keyword a
"Weight in tons {0.weight}"       # 'weight' attribute o
"Units destroyed: {players[0]}"   # First element of key
```

El campo *conversion* causa una coerción de tipo antes del formateo. Normalmente, el formateo es hecho el método **`__format__()`** del valor mismo. Sin embargo, en algunos es deseable forzar el tipo a ser formateado como una cadena de caracteres, sobrescribiendo su propia definición de formateo. Cuando se convierte el valor a una cadena de caracteres antes de llamar al método **`__format__()`**, la lógica normal de formateo es evitada.

Tres banderas de conversión son admitidas actualmente: `'!s'`, que llama a **`str()`** con el valor; `'!r'`, que llama a **`repr()`**; y `'!a'` que llama a **`ascii()`**.

Algunos ejemplos:

```
"Harold's a clever {0!s}"        # Calls str() on the ar
```

```
"Bring out the holy {name!r}"    # Calls repr() on the a
"More {!a}"                      # Calls ascii() on the
```

El campo *format_spec* contiene la especificación de cómo presentar el valor, incluyendo detalles como ancho del campo, alineación, relleno, precisión decimal, etc. Cada tipo de valor puede definir su propio «mini lenguaje de formateo» o interpretación de *format_spec*.

La mayoría de los tipos integrados admiten un formateo común de mini-idioma descrito en la siguiente sección.

Un campo *format_spec* también puede incluir campos de reemplazo anidados dentro de él. Estos campos de reemplazo anidados pueden contener un nombre de campo, una bandera de conversión y una especificación de formato, pero no se permite anidamiento más profundo. Los campos de reemplazo dentro de *format_spec* se sustituyen antes de que la cadena *format_spec* se interprete. Esto permite especificar dinámicamente el formato de un valor.

Para más ejemplos, véase la sección Ejemplos de formateo.

Format specification mini-language

«Format specifications» are used within replacement fields contained within a format string to define how individual values are presented (see Format string syntax, f-strings, and t-strings). They can also be passed directly to the built-in `format()` function. Each formattable type may define how the format specification is to be interpreted.

La mayoría de los tipos integrados implementan las siguientes opciones para especificaciones de formato, aunque algunas de las opciones de formateo sólo son posibles con los tipos numéricos.

Una convención general es que una especificación de formato vacía produce el mismo resultado que llamar a la función `str()` con el valor. Una especificación no vacía típicamente modifica el resultado.

La forma general de un *especificador estándar de formato* es:

```
format_spec:                [options][width_and_precision][
options:                    [[fill]align][sign]["z"]["#"]["
fill:                       <any character>
align:                      "<" | ">" | "=" | "^"
sign:                       "+" | "-" | " "
width_and_precision:        [width_with_grouping][precisio
width_with_grouping:        [width][grouping]
precision_with_grouping:    "." [precision][grouping] | "."
width:                      digit+
precision:                  digit+
grouping:                   "," | "_"
type:                       "b" | "c" | "d" | "e" | "E" | "
                            | "G" | "n" | "o" | "s" | "x" |
```

Si se especifica un valor *align* válido, puede ir precedido por un carácter *fill*, que puede ser cualquier carácter y cuyo valor predeterminado es un espacio si se omite. No es posible utilizar una llave literal (»`{`» or «`}`») como el carácter *fill* en un formato literal de cadena o cuando se utiliza el método `str.format()`. Sin embargo, es posible insertar una llave con un campo de reemplazo anidado. Esta limitación no afecta a la función `format()`.

El significado de las distintas opciones de alineación es el siguiente:

Opción	Significado
'<'	Fuerza el campo a ser alineado a la izquierda dentro del espacio disponible (éste es el comportamiento por defecto para la mayoría de los objetos).
'>'	Fuerza el campo a ser alineado a la derecha dentro del espacio disponible (éste es el comportamiento por defecto para números).
'='	Forces the padding to be placed after the sign (if any) but before the digits. This is used for printing fields in the form "+000000120". This alignment option is only valid for numeric types, excluding `complex`. It becomes the default for numbers when "0" immediately precedes the field width.
'^'	Fuerza el centrado del campo dentro del espacio disponible.

Notar que, a menos que se defina un ancho de campo mínimo, el ancho del campo siempre tendrá el mismo tamaño que los datos para rellenarlo, de modo que la opción de alineación no tiene ningún significado en este caso.

La opción *sign* (signo) sólo es válida para los tipos numéricos y puede ser una de las siguientes:

Opción	**Significado**
`'+'`	Indicates that a sign should be used for both positive as well as negative numbers.
`'-'`	Indicates that a sign should be used only for negative numbers (this is the default behavior).
espacio	Indicates that a leading space should be used on positive numbers, and a minus sign on negative numbers.

La opción `'z'` convierte los valores de coma flotante de cero negativos en cero positivos después de redondear al formato de precisión. Esta opción solo es válida para los tipos de presentación de coma flotante.

Distinto en la versión 3.11: Se agregó la opción `'z'` (véase también **PEP 682** [https://peps.python.org/pep-0682/]).

La opción `'#'` hace que la «forma alternativa» se utilice para la conversión. La forma alternativa se define de forma diferente para diferentes tipos. Esta opción solo es válida para los tipos entero, flotante y complejo. Para los enteros, cuando se utiliza la salida binaria, octal o hexadecimal, esta opción agrega el respectivo prefijo `'0b'`, `'0o'` o `'0x'` al valor de salida. Para los flotantes y complejos, la forma alternativa hace que el resultado de la conversión siempre contenga un carácter de punto decimal, incluso si no hay dígitos que lo sigan. Normalmente, un carácter de punto decimal aparece en el resultado de estas conversiones sólo si un dígito lo sigue. Además, para las conversiones `'g'` y `'G'`, los ceros finales no se eliminan del resultado.

The *width* is a decimal integer defining the minimum total field width, including any prefixes, separators, and other formatting characters. If not specified, then the field width will be determined by the content.

When no explicit alignment is given, preceding the *width* field by a zero (`'0'`) character enables sign-aware zero-padding for numeric types, excluding **`complex`**. This is equivalent to a *fill* character of `'0'` with an *alignment* type of `'='`.

Distinto en la versión 3.10: Anteponer el campo *width* por `'0'` ya no afecta el alineamiento por defecto para cadenas de caracteres.

La *precisión* es un entero decimal que indica cuántos dígitos deben mostrarse después del punto decimal para los tipos de presentación `'f'` y `'F'`, o antes y después del punto decimal para los tipos de presentación `'g'` o `'G'`. Para los tipos de presentación de cadena, el campo indica el tamaño máximo del campo; en otras palabras, cuántos caracteres se utilizarán del contenido del campo. La *precisión* no está permitida para los tipos de presentación de enteros.

The *grouping* option after *width* and *precision* fields specifies a digit group separator for the integral and fractional parts of a number respectively. It can be one of the following:

Opción	**Significado**
`','`	Inserts a comma every 3 digits for integer presentation type `'d'` and floating-point presentation types, excluding `'n'`. For other presentation types, this option is not supported.
`'_'`	Inserts an underscore every 3 digits for integer presentation type `'d'` and floating-point presentation types, excluding `'n'`. For integer presentation types `'b'`, `'o'`, `'x'`, and `'X'`, underscores are inserted every 4 digits. For other presentation types, this option is not supported.

For a locale aware separator, use the `'n'` presentation type instead.

Distinto en la versión 3.1: Se agregó la opción `','` (véase también **PEP 378** [https://peps.python.org/pep-0378/]).

Distinto en la versión 3.6: Se agregó la opción `'_'` (véase también **PEP 515** [https://peps.python.org/pep-0515/]).

Distinto en la versión 3.14: Support the *grouping* option for the fractional part.

Finalmente, *type* (el tipo) determina como presentar los datos.

Los tipos de presentación cadena disponibles son:

Tipo	Significado
`'s'`	Formato de cadena de caracteres. Éste es el tipo por defecto y puede ser omitido.
None	Lo mismo que `'s'`.

Los tipos disponibles para la presentación de enteros son:

Tipo	Significado
`'b'`	Formato binario. retorna el número en base 2.
`'c'`	Carácter. Convierte el entero en el carácter unicode correspondiente antes de imprimirlo.
`'d'`	Decimal entero. retorna el número en base 10.
`'o'`	Formato octal. retorna el número en base 8.
`'x'`	Formato hexadecimal. retorna el número en base 16, utilizando letras minúsculas para los dígitos superiores a 9.
`'X'`	Formato hexadecimal. retorna el número en base 16, utilizando letras mayúsculas para los dígitos superiores a 9. En caso de que `'#'` se especifique, el prefijo `'0x'` convertirá en `'0X'` también.
`'n'`	Number. This is the same as `'d'`, except that it uses the current locale setting to insert the appropriate digit group separators.
None	Lo mismo que `'d'`.

In addition to the above presentation types, integers can be formatted with the floating-point presentation types listed below (except `'n'` and `None`). When doing so, `float()` is used to convert the integer to a floating-point number before formatting.

Los tipos de presentación disponibles para valores `float` y `Decimal` son:

Tipo	Significado
`'e'`	Scientific notation. For a given precision `p`, formats the number in scientific notation with

	the letter “e” separating the coefficient from the exponent. The coefficient has one digit before and `p` digits after the decimal point, for a total of `p + 1` significant digits. With no precision given, uses a precision of `6` digits after the decimal point for **`float`**, and shows all coefficient digits for **`Decimal`**. If `p=0`, the decimal point is omitted unless the `#` option is used. For **`float`**, the exponent always contains at least two digits, and is zero if the value is zero.
`'E'`	Notación científica. Igual que `'e'` excepto que utiliza una “E” mayúscula como carácter separador.
`'f'`	Fixed-point notation. For a given precision `p`, formats the number as a decimal number with exactly `p` digits following the decimal point. With no precision given, uses a precision of `6` digits after the decimal point for **`float`**, and uses a precision large enough to show all coefficient digits for **`Decimal`**. If `p=0`, the decimal point is omitted unless the `#` option is used.
`'F'`	Notación de punto fijo. Igual que `'f'`, pero convierte (nulos) `nan` a `NAN` e `inf` a `INF`.
`'g'`	Formato general. Para una precisión dada `p >= 1`, redondea el número a `p` dígitos significativos y luego formatea el resultado como formato de punto fijo o en notación científica, dependiendo de su magnitud. Una precisión de `0` es tratada como equivalente a una precisión de `1`. Las reglas precisas son las siguientes: supongamos que el resultado formateado con el tipo de presentación `'e'` y la precisión `p-1` tiene exponente `exp`. Entonces, si `m <= exp < p`, donde `m` es -4 para *floats* y -6 para **`Decimals`**, el número se formatea con el tipo de presentación `'f'` y la precisión `p-1-exp`. De lo contrario, el número se formatea con el tipo de presentación `'e'` y precisión `p-1`. En ambos

	casos, los ceros finales insignificantes se eliminan del significado, y el punto decimal también se elimina si no hay dígitos restantes que lo sigan, a menos que se utilice la opción `'#'`. Si no se da una precisión, usa una precisión de `6` dígitos significativos para **`float`**. Para **`Decimal`** el coeficiente del resultado se construye usando los dígitos del coeficiente del valor; se usa notación científica para valores menores a `1e-6` en valor absoluto y valores donde el valor posicional del dígito menos significativo es mayor a 1, de otra forma se usa notación de punto fijo. Infinito positivo y negativo, cero positivo y negativo, y nulos (*nans*) son respectivamente formateados como `inf`, `-inf`, `0`, `-0` y `nan`, independientemente de la precisión.
`'G'`	Formato general. Igual que `'g'` excepto que cambia a `'E'` si el número se vuelve muy grande. Las representaciones de infinito y NaN también se convierten a mayúsculas.
`'n'`	Number. This is the same as `'g'`, except that it uses the current locale setting to insert the appropriate digit group separators for the integral part of a number.
`'%'`	Porcentaje. Multiplica el número por 100 y lo muestra en formato fijo (`'f'`) seguido del signo porcentaje.
None	For **`float`** this is like the `'g'` type, except that when fixed-point notation is used to format the result, it always includes at least one digit past the decimal point, and switches to the scientific notation when `exp >= p - 1`. When the precision is not specified, the latter will be as large as needed to represent the given value faithfully. Para **`Decimal`**, esto es lo mismo que `'g'` o `'G'` dependiendo del valor de `context.capitals` para el contexto decimal actual.

El efecto general es el de igualar la salida de `str()` al ser alterada por los otros modificadores de formato.

The result should be correctly rounded to a given precision `p` of digits after the decimal point. The rounding mode for `float` matches that of the `round()` builtin. For `Decimal`, the rounding mode of the current context will be used.

The available presentation types for `complex` are the same as those for `float` (`'%'` is not allowed). Both the real and imaginary components of a complex number are formatted as floating-point numbers, according to the specified presentation type. They are separated by the mandatory sign of the imaginary part, the latter being terminated by a `j` suffix. If the presentation type is missing, the result will match the output of `str()` (complex numbers with a non-zero real part are also surrounded by parentheses), possibly altered by other format modifiers.

Ejemplos de formateo

Esta sección contiene ejemplos de la sintaxis `str.format()` y comparaciones con el antiguo método de formateo usando `%`.

En la mayoría de los casos, la sintaxis es similar al antiguo formato `%`, con la adición de `{}` y con `:` utilizado en lugar de `%`. Por ejemplo, `'%03.2f'` puede ser traducido como `'{:03.2f}'`.

La nueva sintaxis de formato también soporta opciones diferentes y nuevas que se muestran en los ejemplos siguientes.

Accediendo argumentos por posición:

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{}, {}, {}'.format('a', 'b', 'c')  # 3.1+ only
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
'c, b, a'
>>> '{2}, {1}, {0}'.format(*'abc')      # unpacking argu
'c, b, a'
```

```
>>> '{0}{1}{0}'.format('abra', 'cad')   # arguments' inc
'abracadabra'
```

Accediendo argumentos por nombre:

```
>>> 'Coordinates: {latitude}, {longitude}'.format(latitu
'Coordinates: 37.24N, -115.81W'
>>> coord = {'latitude': '37.24N', 'longitude': '-115.81
>>> 'Coordinates: {latitude}, {longitude}'.format(**coor
'Coordinates: 37.24N, -115.81W'
```

Accediendo los atributos de los argumentos:

```
>>> c = 3-5j
>>> ('The complex number {0} is formed from the real par
...  'and the imaginary part {0.imag}.').format(c)
'The complex number (3-5j) is formed from the real part 
>>> class Point:
...     def __init__(self, x, y):
...         self.x, self.y = x, y
...     def __str__(self):
...         return 'Point({self.x}, {self.y})'.format(se
...
>>> str(Point(4, 2))
'Point(4, 2)'
```

Accediendo ítems de los argumentos:

```
>>> coord = (3, 5)
>>> 'X: {0[0]};  Y: {0[1]}'.format(coord)
'X: 3;  Y: 5'
```

Reemplazar `%s` y `%r`:

```
>>> "repr() shows quotes: {!r}; str() doesn't: {!s}".for
"repr() shows quotes: 'test1'; str() doesn't: test2"
```

Alinear el texto y especificar el ancho:

```
>>> '{:<30}'.format('left aligned')
'left aligned                  '
```

```
>>> '{:>30}'.format('right aligned')
'                 right aligned'
>>> '{:^30}'.format('centered')
'           centered           '
>>> '{:*^30}'.format('centered')  # use '*' as a fill ch
'***********centered***********'
```

Reemplazar `%+f`, `%-f`, y `% f` y especificar el signo:

```
>>> '{:+f}; {:+f}'.format(3.14, -3.14)  # show it always
'+3.140000; -3.140000'
>>> '{: f}; {: f}'.format(3.14, -3.14)  # show a space f
' 3.140000; -3.140000'
>>> '{:-f}; {:-f}'.format(3.14, -3.14)  # show only the
'3.140000; -3.140000'
```

Reemplazando `%x` y `%o` y convirtiendo el valor a diferentes bases:

```
>>> # format also supports binary numbers
>>> "int: {0:d};  hex: {0:x};  oct: {0:o};  bin: {0:b}".
'int: 42;  hex: 2a;  oct: 52;  bin: 101010'
>>> # with 0x, 0o, or 0b as prefix:
>>> "int: {0:d};  hex: {0:#x};  oct: {0:#o};  bin: {0:#b
'int: 42;  hex: 0x2a;  oct: 0o52;  bin: 0b101010'
```

Using the comma or the underscore as a digit group separator:

```
>>> '{:,}'.format(1234567890)
'1,234,567,890'
>>> '{:_}'.format(1234567890)
'1_234_567_890'
>>> '{:_b}'.format(1234567890)
'100_1001_1001_0110_0000_0010_1101_0010'
>>> '{:_x}'.format(1234567890)
'4996_02d2'
>>> '{:_}'.format(123456789.123456789)
'123_456_789.12345679'
>>> '{:.,}'.format(123456789.123456789)
'123456789.123,456,79'
>>> '{:,._}'.format(123456789.123456789)
```

```
'123,456,789.123_456_79'
```

Expresar un porcentaje:

```
>>> points = 19
>>> total = 22
>>> 'Correct answers: {:.2%}'.format(points/total)
'Correct answers: 86.36%'
```

Uso del formateo específico de tipo:

```
>>> import datetime as dt
>>> d = dt.datetime(2010, 7, 4, 12, 15, 58)
>>> '{:%Y-%m-%d %H:%M:%S}'.format(d)
'2010-07-04 12:15:58'
```

Anidando argumentos y ejemplos más complejos:

```
>>> for align, text in zip('<^>', ['left', 'center', 'ri
...     '{0:{fill}{align}16}'.format(text, fill=align, a
...
'left<<<<<<<<<<<<'
'^^^^^center^^^^^'
'>>>>>>>>>>>right'
>>>
>>> octets = [192, 168, 0, 1]
>>> '{:02X}{:02X}{:02X}{:02X}'.format(*octets)
'C0A80001'
>>> int(_, 16)
3232235521
>>>
>>> width = 5
>>> for num in range(5,12):
...     for base in 'dXob':
...         print('{0:{width}{base}}'.format(num, base=b
...     print()
...
    5     5     5   101
    6     6     6   110
    7     7     7   111
    8     8    10  1000
    9     9    11  1001
   10     A    12  1010
   11     B    13  1011
```

Template strings ($-strings)

Nota

The feature described here was introduced in Python 2.4; a simple templating method based upon regular expressions. It predates `str.format()`, formatted string literals, and template string literals.

It is unrelated to template string literals (t-strings), which were introduced in Python 3.14. These evaluate to `string.templatelib.Template` objects, found in the `string.templatelib` module.

Las cadenas de plantilla proporcionan sustituciones de cadenas de caracteres más sencillas como se describe en **PEP 292** [https://peps.python.org/pep-0292/]. Un caso de uso principal para las cadenas de plantilla es la internacionalización (i18n) ya que en ese contexto, la sintaxis y la funcionalidad más sencillas hacen la traducción más fácil que otras instalaciones de formato de cadena de caracteres integradas en Python. Como ejemplo de una biblioteca creada sobre cadenas de plantilla para i18n, véase el paquete flufl.i18n [http://flufli18n.readthedocs.io/en/latest/].

Las cadenas de caracteres de plantilla admiten sustituciones basadas en `$` de acuerdo a las siguientes reglas:

- `$$` es un escape. Es reemplazado con un único `$`.
- `$identifier` (identificador) nombra un comodín de sustitución que coincide con una clave de asignación de `"identifier"` (identificador). De forma predeterminada, `"identifier"` está restringido a cualquier cadena alfanumérica ASCII (insensible a mayúsculas/minúsculas e incluyendo los guiones bajos) que comience con un guión

bajo o una letra ASCII. El primer carácter no identificador después del carácter `$` termina esta especificación de comodín.

- `${identifier}` es equivalente a `$identifier`. Es requerido cuando caracteres identificadores válidos siguen al comodín pero no son parte de él, por ejemplo `"${noun}ification"`.

Cualquier otra aparición de `$` en la cadena de caracteres resultará en una excepción `ValueError`.

The **`string`** module provides a `Template` class that implements these rules. The methods of **`Template`** are:

class string.Template(*template*)

El constructor sólo lleva un argumento, la cadena plantilla.

substitute(*mapping={}*, */*, ***kwds*)

Realiza la sustitución de plantilla y retorna una nueva cadena de caracteres. *mapping* (mapeo) es un objeto tipo diccionario con claves (*keys*) que coinciden con los *placeholders* (comodines) de la plantilla. Como alternativa, es posible pasar argumentos de palabra clave cuyas palabras clave son los *placeholders* (comodines). Cuando *mapping* y *kwds* son dados y hay elementos duplicados, los *placeholders* (comodines) de *kwds* tienen prioridad.

safe_substitute(*mapping={}*, */*, ***kwds*)

Igual que `substitute()`, excepto que si faltan comodines de *mapping* y *kwds*, en lugar de generar una excepción `KeyError`, el comodín original aparecerá en la cadena de caracteres resultante intacta. Además, a diferencia de **`substitute()`**, cualquier otra aparición de `$` simplemente retornará `$` en lugar de generar `ValueError`.

Mientras que otras excepciones aún pueden ocurrir, este método es llamado «seguro» (*safe*) porque siempre intenta retornar una cadena de caracteres que pueda ser usada en lugar de levantar una excepción. Desde otro punto de vista, el método **`safe_substitute()`** es en realidad para nada seguro, dado que ignorará plantillas defectuosas con delimitadores colgados, llaves sin cerrar, o comodines que no son identificadores válidos en Python.

is_valid()

Returns `False` if the template has invalid placeholders that will cause **`substitute()`** to raise **`ValueError`**.

Added in version 3.11.

get_identifiers()

Retorna una lista de los identificadores válidos en la plantilla, en el orden en que aparecen por primera vez, ignorando cualquier identificador no válido.

Added in version 3.11.

Las instancias de **`Template`** también proporcionan un atributo de datos públicos:

template

Éste es el objeto que se le pasa como argumento *template* al constructor. En general, no debería ser modificado, pero el acceso de sólo lectura (*read-only*) no es impuesto.

Aquí un ejemplo de cómo usar una plantilla (*Template*):

```
>>> from string import Template
>>> s = Template('$who likes $what')
>>> s.substitute(who='tim', what='kung pao')
'tim likes kung pao'
>>> d = dict(who='tim')
>>> Template('Give $who $100').substitute(d)
Traceback (most recent call last):
```

```
...
ValueError: Invalid placeholder in string: line 1, col 1
>>> Template('$who likes $what').substitute(d)
Traceback (most recent call last):
...
KeyError: 'what'
>>> Template('$who likes $what').safe_substitute(d)
'tim likes $what'
```

Uso avanzado: es posible derivar subclases de **`Template`** para personalizar la sintaxis de *placeholder*, caracteres de delimitación, o bien la expresión regular entera usada para procesar cadenas de plantillas. Para ello, es posible sobrescribir los siguientes atributos de clase:

- *delimiter*: es la cadena de caracteres literal que describe al delimitador que introduce un comodín. El valor predeterminado es `$`. Notar que esto *no debe* ser una expresión regular, ya que la implementación llamará a **`re.escape()`** en esta cadena de caracteres según sea necesario. Tener en cuenta además, que se no puede cambiar el delimitador después haber creado la clase (es decir, se debe establecer un delimitador diferente en el espacio de nombres de la subclase).

- *idpattern* – Esta es la expresión regular que describe el patrón para comodines fuera de las llaves. El valor predeterminado es la expresión regular `(?a:[_a-z][_a-z0-9]*)`. Si se proporciona este argumento y *braceidpattern* es `None` este patrón también se aplicará a los comodines entre llaves.

  Nota

  Dado que el valor predeterminado de *flags* es `re.IGNORECASE`, el patrón `[a-z]` puede coincidir con algunos caracteres que no son ASCII. Por ello se utiliza aquí la bandera local `a`.

  Distinto en la versión 3.7: braceidpattern puede ser usado para definir patrones separados, usados dentro y fuera de los corchetes.

- *braceidpattern* – Es como *idpattern* pero describe el patrón para comodines entre llaves. El valor por defecto es `None`, lo que significa volver a *idpattern* (es decir, el mismo patrón se utiliza tanto dentro como fuera de las llaves). Si se proporciona, esto le permite definir diferentes patrones para comodines dentro y fuera de las llaves.

  Added in version 3.7.

- *flags* – Banderas de expresión regular que se aplicarán al compilar la expresión regular utilizada para reconocer sustituciones. El valor por defecto es `re.IGNORECASE`. Téngase en cuenta que `re.VERBOSE` siempre se agregará a las banderas, por lo que *idpattern* (s) personalizado(s) debe(n) seguir las convenciones de expresiones regulares detalladas.

  Added in version 3.2.

Como alternativa, se puede proporcionar el patrón de expresión regular completo, reemplazando así el atributo de clase *pattern*. Si eso ocurre, el valor debe ser un objeto de expresión regular con cuatro grupos de captura con nombre. Los grupos de captura corresponden a las reglas indicadas anteriormente, junto con la regla de marcador de posición no válida:

- *escaped* – Este grupo coincide con la secuencia de escape en el patrón predeterminado, por ejemplo, `$$`.
- *named* – Este grupo coincide con el nombre comodín fuera de las llaves. No debe incluir el delimitador del grupo de captura.
- *braced* – Este grupo coincide con el nombre del comodín adjunto; no debe incluir ni el delimitador ni las llaves en el grupo de captura.
- *invalid* – Este grupo se empareja con cualquier otro patrón de delimitación (usualmente un único carácter) y debe ser lo último en aparecer en la expresión regular.

Los métodos de esta clase generarán **`ValueError`** si el patrón coincide con la plantilla sin que coincida uno de estos grupos nombrados.

Funciones de ayuda

string.capwords(*s*, *sep=None*)

Separa el argumento en dos palabras utilizando el método `str.split()`, convierte a mayúsculas vía `str.capitalize()` y une las palabras en mayúscula con el método `str.join()`. Si el segundo argumento opcional *sep* está ausente o es `None`, los espacios en blanco se reemplazarán con un único espacio y los espacios en blanco iniciales y finales se eliminarán; caso contrario, *sep* se usa para separar y unir las palabras.

`string.templatelib` — Support for template string literals

Source code: Lib/string/templatelib.py [https://github.com/python/cpython/tree/3.14/Lib/string/templatelib.py]

Ver también

- Format strings
- Template string literal (t-string) syntax
- **PEP 750** [https://peps.python.org/pep-0750/]

Template strings

Added in version 3.14.

Template strings are a mechanism for custom string processing. They have the full flexibility of Python's f-strings, but return a **`Template`** instance that gives access to the static and interpolated (in curly brackets) parts of a string *before* they are combined.

To write a t-string, use a `'t'` prefix instead of an `'f'`, like so:

```
>>> pi = 3.14
>>> t't-strings are new in Python {pi!s}!'
Template(
   strings=('t-strings are new in Python ', '!'),
   interpolations=(Interpolation(3.14, 'pi', 's', ''),)
)
```

Types

class string.templatelib.Template

The **`Template`** class describes the contents of a template string. It is immutable, meaning that attributes of a template cannot be reassigned.

The most common way to create a **`Template`** instance is to use the template string literal syntax. This syntax is identical to that of f-strings, except that it uses a `t` prefix in place of an `f`:

```
>>> cheese = 'Red Leicester'
>>> template = t"We're fresh out of {cheese}, sir."
>>> type(template)
<class 'string.templatelib.Template'>
```

Templates are stored as sequences of literal **`strings`** and dynamic **`interpolations`**. A **`values`** attribute holds the values of the interpolations:

```
>>> cheese = 'Camembert'
>>> template = t'Ah! We do have {cheese}.'
>>> template.strings
('Ah! We do have ', '.')
>>> template.interpolations
(Interpolation('Camembert', ...),)
>>> template.values
('Camembert',)
```

The **`strings`** tuple has one more element than **`interpolations`** and **`values`**; the interpolations “belong” between the strings. This may be easier to understand when tuples are aligned

```
template.strings:  ('Ah! We do have ',
template.values:   (                   'Camembert',
```

Attributes

strings: *tuple[str, ...]*

A **`tuple`** of the static strings in the template.

```
>>> cheese = 'Camembert'
>>> template = t'Ah! We do have {cheese}.'
>>> template.strings
('Ah! We do have ', '.')
```

Empty strings *are* included in the tuple:

```
>>> response = 'We do have '
>>> cheese = 'Camembert'
>>> template = t'Ah! {response}{cheese}.'
>>> template.strings
('Ah! ', '', '.')
```

The `strings` tuple is never empty, and always contains one more string than the `interpolations` and `values` tuples:

```
>>> t''.strings
('',)
>>> t''.values
()
>>> t'{'cheese'}'.strings
('', '')
>>> t'{'cheese'}'.values
('cheese',)
```

interpolations*: tuple[Interpolation, ...]*

A **`tuple`** of the interpolations in the template.

```
>>> cheese = 'Camembert'
>>> template = t'Ah! We do have {cheese}.'
>>> template.interpolations
(Interpolation('Camembert', 'cheese', None, '')
```

The `interpolations` tuple may be empty and always contains one fewer values than the `strings` tuple:

```
>>> t'Red Leicester'.interpolations
()
```

values: *tuple[object, ...]*

A tuple of all interpolated values in the template.

```
>>> cheese = 'Camembert'
>>> template = t'Ah! We do have {cheese}.'
>>> template.values
('Camembert',)
```

The `values` tuple always has the same length as the `interpolations` tuple. It is always equivalent to `tuple(i.value for i in template.interpolations)`.

Methods

new(**args: str | Interpolation*)

While literal syntax is the most common way to create a **`Template`**, it is also possible to create them directly using the constructor:

```
>>> from string.templatelib import Interpolatio
>>> cheese = 'Camembert'
>>> template = Template(
...     'Ah! We do have ', Interpolation(cheese
... )
>>> list(template)
['Ah! We do have ', Interpolation('Camembert',
```

If multiple strings are passed consecutively, they will be concatenated into a single value in the **`strings`** attribute. For example, the following code creates a **`Template`** with a single final string:

```
>>> from string.templatelib import Template
>>> template = Template('Ah! We do have ', 'Car
>>> template.strings
('Ah! We do have Camembert.',)
```

If multiple interpolations are passed consecutively, they will be treated as separate interpolations and an empty string will be inserted between them. For example, the following code creates a template with empty placeholders in the `strings` attribute:

```
>>> from string.templatelib import Interpolatio
>>> template = Template(
...     Interpolation('Camembert', 'cheese'),
...     Interpolation('.', 'punctuation'),
... )
>>> template.strings
('', '', '')
```

iter(template)

Iterate over the template, yielding each non-empty string and `Interpolation` in the correct order:

```
>>> cheese = 'Camembert'
>>> list(t'Ah! We do have {cheese}.')
['Ah! We do have ', Interpolation('Camembert',
```

Prudencia

Empty strings are **not** included in the iteration:

```
>>> response = 'We do have '
>>> cheese = 'Camembert'
>>> list(t'Ah! {response}{cheese}.')
['Ah! ',
 Interpolation('We do have ', 'response', None
 Interpolation('Camembert', 'cheese', None, '
 '.']
```

template + other

template + = other

Concatenate this template with another, returning a new **`Template`** instance:

```
>>> cheese = 'Camembert'
>>> list(t'Ah! ' + t'We do have {cheese}.')
['Ah! We do have ', Interpolation('Camembert',
```

Concatenating a **`Template`** and a `str` is **not** supported. This is because it is unclear whether the string should be treated as a static string or an interpolation. If you want to concatenate a **`Template`** with a string, you should either wrap the string directly in a **`Template`** (to treat it as a static string) or use an **`Interpolation`** (to treat it as dynamic):

```
>>> from string.templatelib import Interpolatio
>>> template = t'Ah! '
>>> # Treat 'We do have ' as a static string
>>> template += Template('We do have ')
>>> # Treat cheese as an interpolation
>>> cheese = 'Camembert'
>>> template += Template(Interpolation(cheese,
>>> list(template)
['Ah! We do have ', Interpolation('Camembert',
```

class string.templatelib.Interpolation

The **`Interpolation`** type represents an expression inside a template string. It is immutable, meaning that attributes of an interpolation cannot be reassigned.

Interpolations support pattern matching, allowing you to match against their attributes with the match statement:

```
>>> from string.templatelib import Interpolation
>>> interpolation = t'{1. + 2.:.2f}'.interpolations
>>> interpolation
Interpolation(3.0, '1. + 2.', None, '.2f')
>>> match interpolation:
```

```
...     case Interpolation(value, expression, conve
...         print(value, expression, conversion, fo
...
3.0 | 1. + 2. | None | .2f
```

Attributes

value: *object*

The evaluated value of the interpolation.

```
>>> t'{1 + 2}'.interpolations[0].value
3
```

expression: *str*

For interpolations created by t-string literals, **`expression`** is the expression text found inside the curly brackets (`{` & `}`), including any whitespace, excluding the curly brackets themselves, and ending before the first `!`, `:`, or `=` if any is present. For manually created interpolations, **`expression`** is the arbitrary string provided when constructing the interpolation instance.

We recommend using valid Python expressions or the empty string for the `expression` field of manually created **`Interpolation`** instances, although this is not enforced at runtime.

```
>>> t'{1 + 2}'.interpolations[0].expression
'1 + 2'
```

conversion: *Literal['a', 'r', 's'] | None*

The conversion to apply to the value, or `None`.

The **`conversion`** is the optional conversion to apply to the value:

```
>>> t'{1 + 2!a}'.interpolations[0].conversion
```

```
'a'
```

Nota

Unlike f-strings, where conversions are applied automatically, the expected behavior with t-strings is that code that *processes* the `Template` will decide how to interpret and whether to apply the `conversion`. For convenience, the `convert()` function can be used to mimic f-string conversion semantics.

format_spec: *str*

The format specification to apply to the value.

The `format_spec` is an optional, arbitrary string used as the format specification to present the value:

```
>>> t'{1 + 2:.2f}'.interpolations[0].format_spe
'.2f'
```

Nota

Unlike f-strings, where format specifications are applied automatically via the `format()` protocol, the expected behavior with t-strings is that code that *processes* the interpolation will decide how to interpret and whether to apply the format specification. As a result, `format_spec` values in interpolations can be arbitrary strings, including those that do not conform to the `format()` protocol.

Methods

__new__(*value: object*, *expression: str*, *conversion: Literal['a', 'r',* *'s'] | None = None, format_spec: str = ''*)

Create a new `Interpolation` object from component parts.

Parámetros:

- **value** – The evaluated, in-scope result of the interpolation.
- **expression** – The text of a valid Python expression, or an empty string.
- **conversion** – The conversion to be used, one of `None`, `'a'`, `'r'`, or `'s'`.
- **format_spec** – An optional, arbitrary string used as the format specification to present the value.

Helper functions

string.templatelib.convert(*obj*, /, *conversion*)

Applies formatted string literal conversion semantics to the given object *obj*. This is frequently useful for custom template string processing logic.

Three conversion flags are currently supported:

- `'s'` which calls **`str()`** on the value (like `!s`),
- `'r'` which calls **`repr()`** (like `!r`), and
- `'a'` which calls **`ascii()`** (like `!a`).

If the conversion flag is `None`, *obj* is returned unchanged.

`re` — Regular expression operations

Código fuente: Lib/re/ [https://github.com/python/cpython/tree/3.14/Lib/re/]

Este módulo proporciona operaciones de coincidencia de expresiones regulares similares a las encontradas en Perl.

Both patterns and strings to be searched can be Unicode strings (**`str`**) as well as 8-bit strings (**`bytes`**). However, Unicode strings and 8-bit strings cannot be mixed: that is, you cannot match a Unicode string with a bytes pattern or vice-versa; similarly, when asking for a substitution, the replacement string must be of the same type as both the pattern and the search string.

Las expresiones regulares usan el carácter de barra inversa (`'\'`) para indicar formas especiales o para permitir el uso de caracteres especiales sin invocar su significado especial. Esto choca con el uso de Python de este carácter para el mismo propósito con los literales de cadena; por ejemplo, para hacer coincidir una barra inversa literal, se podría escribir `'\\\\'` como patrón, porque la expresión regular debe ser `\\`, y cada barra inversa debe ser expresada como `\\` dentro de un literal de cadena regular de Python. También, notar que cualquier secuencia de escape inválida mientras se use la barra inversa de Python en los literales de cadena ahora genera un **`SyntaxWarning`** y en el futuro esto se convertirá en un **`SyntaxError`**. Este comportamiento ocurrirá incluso si es una secuencia de escape válida para una expresión regular.

La solución es usar la notación de cadena *raw* de Python para los patrones de expresiones regulares; las barras inversas no se manejan de ninguna manera especial en un literal de cadena prefijado con `'r'`. Así que `r"\n"` es una cadena de dos caracteres que contiene `'\'` y `'n'`, mientras que `"\n"` es una cadena de un carácter que contiene una nueva línea. Normalmente los patrones se expresan en código Python usando esta notación de cadena *raw*.

Es importante señalar que la mayoría de las operaciones de expresiones regulares están disponibles como funciones y métodos a nivel de módulo en expresiones regulares compiladas (expresiones regulares compiladas). Las funciones son atajos que no requieren de compilar un objeto regex primero, aunque pasan por alto algunos parámetros de ajuste.

Ver también

The third-party regex [https://pypi.org/project/regex/] module, which has an API compatible with the standard library `re` module, but offers additional functionality and a more thorough Unicode support.

Sintaxis de expresiones regulares

Una expresión regular (o RE, por sus siglas en inglés) especifica un conjunto de cadenas que coinciden con ella; las funciones de este módulo permiten comprobar si una determinada cadena coincide con una expresión regular dada (o si una expresión regular dada coincide con una determinada cadena, que se reduce a lo mismo).

Las expresiones regulares pueden ser concatenadas para formar nuevas expresiones regulares; si *A* y *B* son ambas expresiones regulares, entonces *AB* es también una expresión regular. En general, si una cadena *p* coincide con *A* y otra cadena *q* coincide con *B*, la cadena *porque* coincidirá con AB. Esto se mantiene a menos que *A* o *B* contengan operaciones de baja precedencia; condiciones límite entre *A* y *B*; o tengan referencias de grupo numeradas. Así, las expresiones complejas pueden construirse fácilmente a partir de expresiones primitivas más simples como las que se describen aquí. Para detalles de la teoría e implementación de las expresiones regulares, consulte el libro de Friedl [Frie09], o casi cualquier libro de texto sobre la construcción de compiladores.

A continuación se explica brevemente el formato de las expresiones regulares. Para más información y una presentación más amena, consultar la Regular expression HOWTO.

Las expresiones regulares pueden contener tanto caracteres especiales como ordinarios. La mayoría de los caracteres ordinarios, como `'A'`, `'a'`, o `'0'` son las expresiones regulares más sencillas; simplemente se ajustan a sí mismas. Se pueden concatenar caracteres ordinarios, así que `last` coincide con la cadena `'last'`. (En el resto de esta sección, se escribirán los RE en `este estilo especial`, normalmente sin comillas, y las cadenas que deban coincidir `'entre comillas simples'`.)

Algunos caracteres, como `'|'` o `'('`, son especiales. Los caracteres especiales representan clases de caracteres ordinarios, o afectan a la forma en que se interpretan las expresiones regulares que los rodean.

Los operadores de repetición o cuantificadores (`*`, `+`, `?`, `{m,n}`, etc.) no pueden ser anidados directamente. Esto evita la ambigüedad con el sufijo modificador no codicioso `?`, y con otros modificadores en otras implementaciones. Para aplicar una segunda repetición a una repetición interna, se pueden usar paréntesis. Por ejemplo, la expresión `(?:a{6})*` coincide con cualquier múltiplo de seis caracteres `'a'`.

Los caracteres especiales son:

`.`

> (Dot.) In the default mode, this matches any character except a newline. If the **`DOTALL`** flag has been specified, this matches any character including a newline. `(?s:.)` matches any character regardless of flags.

`^`

> (Circunflejo.) Coincide con el comienzo de la cadena, y en modo **`MULTILINE`** también coincide inmediatamente después de cada nueva línea.

`$`

> Coincide con el final de la cadena o justo antes de la nueva línea al final de la cadena, y en modo **`MULTILINE`** también

coincide antes de una nueva línea. `foo` coincide con “foo” y “foobar”, mientras que la expresión regular `foo$` sólo coincide con “foo”. Más interesante aún, al buscar `foo.$` en `'foo1\nfoo2\n'` coincide con “foo2” normalmente, pero solo “foo1” en **`MULTILINE`**`; si busca un solo `$` en `'foo\n'` encontrará dos coincidencias (vacías): una justo antes de una nueva línea, y otra al final de la cadena.

`*`

Hace que el RE resultante coincida con 0 o más repeticiones del RE precedente, tantas repeticiones como sean posibles. `ab*` coincidirá con “a”, “ab” o “a” seguido de cualquier número de “b”.

`+`

Hace que la RE resultante coincida con 1 o más repeticiones de la RE precedente. `ab+` coincidirá con “a” seguido de cualquier número distinto de cero de “b”; no coincidirá solo con “a”.

`?`

Hace que la RE resultante coincida con 0 o 1 repeticiones de la RE precedente. `ab?` coincidirá con “a” o “ab”.

`*?`, `+?`, `??`

Los delimitadores `'*'`, `'+'`, y `'?'` son todos *greedy*; coinciden con la mayor cantidad de texto posible. A veces este comportamiento no es deseado; si el RE `<.*>` se utiliza para coincidir con `'<a> b <c>'`, coincidirá con toda la cadena, y no sólo con `'<a>'`. Añadiendo `?` después del delimitador hace que se realice la coincidencia de manera *non-greedy* o *minimal*; coincidirá la *mínima* cantidad de caracteres como sea posible. Usando el RE `<.*?>` sólo coincidirá con `'<a>'`.

`*+`, `++`, `?+`

Como los cuantificadores `'*'`, `'+'`, y `'?'` , aquellos en los que se agrega `'+'` también coinciden tantas veces como sea posible. Sin embargo, a diferencia de los verdaderos cuantificadores codiciosos, estos no permiten retroceder cuando la expresión que le sigue no coincide. Estos se conocen como cuantificadores *possessive*. Por ejemplo, `a*a` coincidirá con `'aaaa'` porque la `a*` coincidirá con los 4 `'a'`s, pero, cuando se encuentra la `'a'` final, la expresión retrocede de modo que al final la `a*` termina coincidiendo con 3 `'a'`s total, y la cuarta `'a'` coincide con la final `'a'`. Sin embargo, cuando `a*+a` se usa para que coincida con `'aaaa'`, el `a*+` coincidirá con los 4 `'a'`, pero cuando el `'a'` final no encuentra más caracteres para coincidir, la expresión no puede retroceder y, por lo tanto, no coincidirá. `x*+`, `x++` and `x?+` son equivalentes a `(?>x*)`, `(?>x+)` and `(?>x?)` correspondientemente.

Added in version 3.11.

`{m}`

Especifica que exactamente *m* copias de la RE anterior deben coincidir; menos coincidencias hacen que la RE entera no coincida. Por ejemplo, `a{6}` coincidirá exactamente con seis caracteres `'a'`, pero no con cinco.

`{m,n}`

Hace que el RE resultante coincida de *m* a *n* repeticiones del RE precedente, tratando de coincidir con el mayor número de repeticiones posible. Por ejemplo, `a{3,5}` coincidirá de 3 a 5 caracteres `'a'`. Omitiendo *m* se especifica un límite inferior de cero, y omitiendo *n* se especifica un límite superior infinito. Por ejemplo, `a{4,}b` coincidirá con `'aaaab'` o mil caracteres `'a'` seguidos de una `'b'`, pero no `'aaab'`. La coma no puede ser omitida o el modificador se confundiría con la forma descrita anteriormente.

`{m,n}?`

Hace que el RE resultante coincida de *m* a *n* repeticiones del RE precedente, tratando de coincidir con el *mínimo de* repeticiones posible. Esta es la versión *non-greedy* (no codiciosa) del delimitador anterior. Por ejemplo, en la cadena de 6 caracteres `'aaaaaa'`, `a{3,5}` coincidirá con 5 caracteres `'a'`, mientras que `a{3,5}?` solo coincidirá con 3 caracteres.

`{m,n}+`

Hace que el RE resultante coincida de *m* a *n* repeticiones del RE anterior, intentando hacer coincidir tantas repeticiones como sea posible *sin* establecer ningún punto de retroceso. Esta es la versión posesiva del cuantificador anterior. Por ejemplo, en la cadena de 6 caracteres `'aaaaaa'`, `a{3,5}+aa` intenta hacer coincidir 5 caracteres `'a'`, entonces, al requerir 2 más `'a'`s, necesitará más caracteres de los disponibles y, por lo tanto, fallará, mientras que `a{3,5}aa` coincidirá con `a{3,5}` capturando 5, luego 4 `'a'`s por retroceso y luego los últimos 2 `'a'`s son emparejados por el `aa` final en el patrón. `x{m,n}+` es equivalente a `(?>x{m,n})`.

Added in version 3.11.

`\`

O bien se escapan a los caracteres especiales (lo que le permite hacer coincidir caracteres como `'*'`, `'?'`, y así sucesivamente), o se señala una secuencia especial; las secuencias especiales se explican más adelante.

Si no se utiliza una cadena *raw* para expresar el patrón, recuerde que Python también utiliza la barra inversa como secuencia de escape en los literales de la cadena; si el analizador sintáctico de Python no reconoce la secuencia de escape, la barra inversa y el carácter subsiguiente se incluyen en la cadena resultante. Sin embargo, si Python quisiera reconocer la secuencia resultante, la barra inversa debería repetirse dos veces. Esto es complicado y difícil de entender, por lo que se recomienda encarecidamente utilizar cadenas *raw* para todas las expresiones salvo las más simples.

`[]`

Se utiliza para indicar un conjunto de caracteres. En un conjunto:

- Los caracteres pueden ser listados individualmente, ej.

`[amk]` coincidirá con `'a'`, `'m'`, o `'k'`.

- Los rangos de caracteres se pueden indicar mediante dos caracteres y separándolos con un `'-'`. Por ejemplo, `[a-z]` coincidirá con cualquier letra ASCII en minúscula, `[0-5][0-9]` coincidirá con todos los números de dos dígitos desde el `00` hasta el `59`, y `[0-9A-Fa-f]` coincidirá con cualquier dígito hexadecimal. Si se escapa `-` (por ejemplo, `[a\-z]`) o si se coloca como el primer o el último carácter (por ejemplo, `[-a]` o `[a-]`), coincidirá con un literal `'-'`.
- Special characters except backslash lose their special meaning inside sets. For example, `[(+*)]` will match any of the literal characters `'('`, `'+'`, `'*'`, or `')'`.

- Backslash either escapes characters which have special meaning in a set such as `'-'`, `']'`, `'^'` and `'\\'` itself or signals a special sequence which represents a single character such as `\xa0` or `\n` or a character class such as `\w` or `\S` (defined below). Note that `\b` represents a single «backspace» character, not a word boundary as outside a set, and numeric escapes such as `\1` are always octal escapes, not group references. Special sequences which do not match a single character such as `\A` and `\z` are not allowed.

- Los caracteres que no están dentro de un rango pueden ser coincidentes con *complementing* el conjunto. Si el primer carácter del conjunto es `'^'`, todos los caracteres que *no* están en el conjunto coincidirán. Por ejemplo, `[^5]` coincidirá con cualquier carácter excepto con `'5'`, y `[^^]` coincidirá con cualquier carácter excepto con `'^'`. `^` no tiene un significado especial si no es el primer carácter del conjunto.
- Para coincidir con un `']'` literal dentro de un set, se debe preceder con una barra inversa, o colocarlo al principio del set. Por ejemplo, tanto `[()[\]{}]` como `[]()[{}]` coincidirá con los paréntesis, corchetes y llaves.

- El soporte de conjuntos anidados y operaciones de conjuntos como en Unicode Technical Standard #18 [https://unicode.org/reports/tr18/] podría ser añadido en el futuro. Esto cambiaría la sintaxis, así que por el momento se planteará un **`FutureWarning`** en casos ambiguos para facilitar este cambio. Ello incluye conjuntos que empiecen con un literal `'['` o que contengan secuencias de caracteres literales `'--'`, `'&&'`, `'~~'` y `'||'`. Para evitar una advertencia, utilizar el código de escape con una barra inversa.

Distinto en la versión 3.7: **`FutureWarning`** se genera si un conjunto de caracteres contiene construcciones que cambiarán semánticamente en el futuro.

`|`

`A|B`, donde *A* y *B* pueden ser RE arbitrarias, crea una expresión regular que coincidirá con *A* or *B*. Un número arbitrario de RE puede ser separado por `'|'` de esta manera. Esto puede también ser usado dentro de grupos (ver más adelante). Cuando la cadena de destino es procesada, los RE separados por `'|'` son probados de izquierda a derecha. Cuando un patrón coincide completamente, esa rama es aceptada. Esto significa que una vez que *A* coincida, *B* no se comprobará más, incluso si se produce una coincidencia general más larga. En otras palabras, el operador de `'|'` nunca es codicioso. Para emparejar un literal `'|'`, se usa `\|`, o se envuelve dentro de una clase de caracteres, como en `[|]`.

`(...)`

Coincide con cualquier expresión regular que esté dentro de los paréntesis, e indica el comienzo y el final de un grupo; el contenido de un grupo puede ser recuperado después de que se haya realizado una coincidencia, y puede coincidir más adelante en la cadena con la secuencia especial `\number`, que se describe más adelante. Para hacer coincidir los literales ``'('`` o `')'`, se usa `\(` o `\)`, o se envuelve dentro de una clase de caracteres: `[(]`, `[)]`.

`(?...)`

Esta es una notación de extensión (un `'?'` después de un `'('` no tiene ningún otro significado). El primer carácter después de `'?'` determina el significado y la sintaxis de la construcción. Las extensiones normalmente no crean un nuevo grupo; `(?P<name>...)` es la única excepción a esta regla. A continuación se muestran las extensiones actualmente soportadas.

`(?aiLmsux)`

(One or more letters from the set `'a'`, `'i'`, `'L'`, `'m'`, `'s'`, `'u'`, `'x'`.) The group matches the empty string; the letters set the corresponding flags for the entire regular expression:

- **`re.A`** (ASCII-only matching)
- **`re.I`** (ignore case)
- **`re.L`** (locale dependent)
- **`re.M`** (multi-line)
- **`re.S`** (dot matches all)
- **`re.U`** (Unicode matching)
- **`re.X`** (verbose)

(The flags are described in Contenidos del módulo.) This is useful if you wish to include the flags as part of the regular expression, instead of passing a *flag* argument to the **`re.compile()`** function. Flags should be used first in the expression string.

Distinto en la versión 3.11: Esta construcción solo se puede usar al comienzo de la expresión.

`(?:...)`

Una versión no capturable de los paréntesis regulares. Hace coincidir cualquier expresión regular que esté dentro de los paréntesis, pero la subcadena coincidente con el grupo *no puede* ser recuperada después de realizar una coincidencia o referenciada más adelante en el patrón.

`(?aiLmsux-imsx:...)`

(Zero or more letters from the set `'a'`, `'i'`, `'L'`, `'m'`, `'s'`, `'u'`, `'x'`, optionally followed by `'-'` followed by one or more letters from the `'i'`, `'m'`, `'s'`, `'x'`.) The letters set or remove the corresponding flags for the part of the expression:

- **`re.A`** (ASCII-only matching)
- **`re.I`** (ignore case)
- **`re.L`** (locale dependent)
- **`re.M`** (multi-line)
- **`re.S`** (dot matches all)
- **`re.U`** (Unicode matching)
- **`re.X`** (verbose)

(The flags are described in Contenidos del módulo.)

The letters `'a'`, `'L'` and `'u'` are mutually exclusive when used as inline flags, so they can't be combined or follow `'-'`. Instead, when one of them appears in an inline group, it overrides the matching mode in the enclosing group. In Unicode patterns `(?a:...)` switches to ASCII-only matching, and `(?u:...)` switches to Unicode matching (default). In bytes patterns `(?L:...)` switches to locale dependent matching, and `(?a:...)` switches to ASCII-only matching (default). This override is only in effect for the narrow inline group, and the original matching mode is restored outside of the group.

Added in version 3.6.

Distinto en la versión 3.7: Las letras `'a'`, `'L'` y `'u'` también pueden ser usadas en un grupo.

`(?>...)`

Intenta hacer coincidir `...` como si fuera una expresión regular separada, y si tiene éxito, continúa coincidiendo con el resto del patrón que la sigue. Si el patrón posterior no coincide, la pila solo se puede desenrollar a un punto *antes* del `(?>...)` Porque una vez que salió, la expresión, conocida como *grupo atomico* <*atomic group*>, ha desechado todos los puntos de pila dentro de sí misma. Por lo tanto, `(?>.*).` nunca coincidiría con nada porque primero el `.*`

coincidiría con todos los caracteres posibles, luego, al no tener nada que igualar, el `.` final no coincidiría. Dado que no hay puntos de pila guardados en el Grupo Atómico, y no hay ningún punto de pila antes de él, toda la expresión no coincidiría.

Added in version 3.11.

`(?P<name>...)`

Similar a los paréntesis regulares, pero la subcadena coincidente con el grupo es accesible a través del nombre simbólico del grupo, *name*. Los nombres de grupo deben ser identificadores válidos de Python, y en los patrones de **`bytes`** solo pueden contener bytes en el rango ASCII. Cada nombre de grupo debe ser definido sólo una vez dentro de una expresión regular. Un grupo simbólico es también un grupo numerado, del mismo modo que si el grupo no tuviera nombre.

Los grupos con nombre pueden ser referenciados en tres contextos. Si el patrón es `(?P<quote>['"]).*?(?P=quote)` (es decir, hacer coincidir una cadena citada con comillas simples o dobles):

Contexto de la referencia al grupo *quote* (cita)	**Formas de hacer referencia**
en el mismo patrón en sí mismo	`(?P=quote)` (como se muestra) `\1`
cuando se procesa el objeto de la coincidencia *m*	`m.group('quote')` `m.end('quote')` (etc.)

en una cadena pasada al argumento *repl* de `re.sub()`	`\g<quote>`
	`\g<1>`
	`\1`

Distinto en la versión 3.12: En patrones de tipo **`bytes`**, el nombre del grupo *name* solo puede contener bytes en el rango ASCII (`b'\x00'`-`b'\x7f'`).

`(?P=name)`

Una referencia inversa a un grupo nombrado; coincide con cualquier texto correspondido por el grupo anterior llamado *name*.

`(?#...)`
Un comentario; el contenido de los paréntesis es simplemente ignorado.

`(?=...)`
Coincide si `...` coincide con el siguiente patrón, pero no procesa nada de la cadena. Esto se llama una *lookahead assertion* (aserción de búsqueda anticipada). Por ejemplo, `Isaac (?=Asimov)` coincidirá con `'Isaac '` sólo si va seguido de `'Asimov'`.

`(?!...)`
Coincide si `...` no coincide con el siguiente. Esta es una *negative lookahead assertion* (aserción negativa de búsqueda anticipada). Por ejemplo, `Isaac (?!Asimov)` coincidirá con `'Isaac '` sólo si *no* es seguido por `'Asimov'`.

`(?<=...)`
Coincide si la posición actual en la cadena es precedida por una coincidencia para `...` que termina en la posición actual. Esto se llama una *positive lookbehind assertion* (aserciones positivas de búsqueda tardía). `(?<=abc)def` encontrará una coincidencia en `'abcdef'`, ya que la búsqueda tardía hará una copia de seguridad de 3 caracteres y comprobará si el patrón contenido coincide. El patrón contenido sólo debe coincidir con cadenas de alguna longitud fija, lo que significa que `abc` o `a|b` están permitidas, pero `a*` y `a{3,4}` no lo están. Hay que tener en cuenta que los patrones que empiezan con aserciones positivas de búsqueda tardía no coincidirán con el principio de la cadena que se está buscando; lo más probable es que se quiera usar la función **`search()`** en lugar de la función **`match()`**:

```
>>> import re
>>> m = re.search('(?<=abc)def', 'abcdef')
>>> m.group(0)
'def'
```

Este ejemplo busca una palabra seguida de un guión:

```
>>> m = re.search(r'(?<=-)\w+', 'spam-egg')
>>> m.group(0)
'egg'
```

Distinto en la versión 3.5: Se añadió soporte a las referencias de grupo de longitud fija.

`(?<!...)`

Coincide si la posición actual en la cadena no está precedida por una coincidencia de «…». Esto se llama una *negative lookbehind assertion* (Aserciones negativas de búsqueda tardía). Similar a las aserciones positivas de búsqueda tardía, el patrón contenido sólo debe coincidir con cadenas de alguna longitud fija. Los patrones que empiezan con aserciones negativas pueden coincidir al principio de la cadena que se busca.

`(?(id/name)yes-pattern|no-pattern)`

Tratará de coincidir con el `yes-pattern` (con patrón) si el grupo con un *id* o *nombre* existe, y con el `no-pattern` (sin patrón) si no existe. El `no-pattern` es opcional y puede ser omitido. Por ejemplo, `(<)?(\w+@\w+(?:\.\w+)+)(?(1)>|$)` es un patrón de coincidencia de correo electrónico deficiente, ya que coincidirá con `'<user@host.com>'` así como con `'user@host.com'`, pero no con `'<user@host.com'` ni con `'user@host.com>'`.

Distinto en la versión 3.12: El *id* del grupo solo puede contener dígitos ASCII. En patrones de tipo **`bytes`**, el *name* del grupo solo puede contener bytes en el rango ASCII (`b'\x00'`-`b'\x7f'`).

Las secuencias especiales consisten en `'\'` y un carácter de la lista que aparece más adelante. Si el carácter ordinario no es un dígito ASCII o una letra ASCII, entonces el RE resultante coincidirá con el segundo carácter. Por ejemplo, `\$` coincide con el carácter `'$'`.

`\number`

Coincide con el contenido del grupo del mismo número. Los grupos se numeran empezando por el 1. Por ejemplo, `(.+) \1` coincide con `'el el'` o `'55 55'`, pero no con `'elel'` (notar el espacio después del grupo). Esta secuencia especial sólo puede ser usada para hacer coincidir uno de los primeros 99 grupos. Si el primer dígito del *número* es 0, o el *número* tiene 3 dígitos octales, no se interpretará como una coincidencia de grupo, sino como el carácter con valor octal *número*. Dentro de los `'['` y `']'` de una clase de caracteres, todos los escapes numéricos son tratados como caracteres.

`\A`

Coincide sólo al principio de la cadena.

`\b`

Matches the empty string, but only at the beginning or end of a word. A word is defined as a sequence of word characters. Note that formally, `\b` is defined as the boundary between a `\w` and a `\W` character (or vice versa), or between `\w` and the beginning or end of the string. This means that `r'\bat\b'` matches `'at'`, `'at.'`, `'(at)'`, and `'as at ay'` but not `'attempt'` or `'atlas'`.

The default word characters in Unicode (str) patterns are Unicode alphanumerics and the underscore, but this can be changed by using the **`ASCII`** flag. Word boundaries are determined by the current locale if the **`LOCALE`** flag is used.

Nota

Inside a character range, `\b` represents the backspace character, for compatibility with Python's string literals.

`\B`

Matches the empty string, but only when it is *not* at the beginning or end of a word. This means that `r'at\B'` matches `'athens'`, `'atom'`, `'attorney'`, but not `'at'`, `'at.'`, or `'at!'`. `\B` is the opposite of `\b`, so word characters in Unicode (str) patterns are Unicode alphanumerics or the underscore, although this can be changed by using the **`ASCII`** flag. Word boundaries are determined by the current locale if the **`LOCALE`** flag is used.

Distinto en la versión 3.14: `\B` now matches empty input string.

`\d`

Para los patrones de Unicode (str):

> Matches any Unicode decimal digit (that is, any character in Unicode character category [Nd] [https://www.unicode.org/versions/Unicode15.0.0/ch04.pdf#G134153]). This includes `[0-9]`, and also many other digit characters.
>
> Matches `[0-9]` if the **`ASCII`** flag is used.

Para patrones de 8 bits (bytes):

> Matches any decimal digit in the ASCII character set; this is equivalent to `[0-9]`.

`\D`

Matches any character which is not a decimal digit. This is the opposite of `\d`.

Matches `[^0-9]` if the **`ASCII`** flag is used.

`\s`

Para los patrones de Unicode (str):

> Matches Unicode whitespace characters (as defined by **`str.isspace()`**). This includes `[ \t\n\r\f\v]`, and also many other characters, for example the non-breaking spaces mandated by typography rules in many languages.
>
> Matches `[ \t\n\r\f\v]` if the **`ASCII`** flag is used.

Para patrones de 8 bits (bytes):

> Coincide con los caracteres considerados como espacios en blanco en el conjunto de caracteres ASCII, lo que equivale a `[ \t\n\r\f\v]`.

`\S`

Matches any character which is not a whitespace character. This is the opposite of `\s`.

Matches `[^ \t\n\r\f\v]` if the **`ASCII`** flag is used.

`\w`

Para los patrones de Unicode (str):

> Matches Unicode word characters; this includes all Unicode alphanumeric characters (as defined by **`str.isalnum()`**), as well as the underscore (_).
>
> Matches `[a-zA-Z0-9_]` if the **`ASCII`** flag is used.

Para patrones de 8 bits (bytes):

> Matches characters considered alphanumeric in the ASCII character set; this is equivalent to `[a-zA-Z0-9_]`. If the **`LOCALE`** flag is used, matches characters considered alphanumeric in the current locale and the underscore.

`\W`

Matches any character which is not a word character. This is the opposite of `\w`. By default, matches non-underscore (_) characters for which **`str.isalnum()`** returns `False`.

Matches `[^a-zA-Z0-9_]` if the **`ASCII`** flag is used.

If the **`LOCALE`** flag is used, matches characters which are neither alphanumeric in the current locale nor the underscore.

`\z`

Coincide sólo el final de la cadena.

Added in version 3.14.

`\Z`

The same as `\z`. For compatibility with old Python versions.

La mayoría de las secuencias de escape soportadas por los literales de cadena de Python también son aceptadas por el analizador de expresiones regulares:

```
\a      \b      \f      \n
\N      \r      \t      \u
\U      \v      \x      \\
```

(Notar que `\b` se usa para representar los límites de las palabras, y significa «retroceso» (*backspace*) sólo dentro de las clases de caracteres.)

`'\u'`, `'\U'`, and `'\N'` escape sequences are only recognized in Unicode (str) patterns. In bytes patterns they are errors. Unknown escapes of ASCII letters are reserved for future use and treated as errors.

Los escapes octales se incluyen en una forma limitada. Si el primer dígito es un 0, o si hay tres dígitos octales, se considera un escape octal. De lo contrario, es una referencia de grupo. En cuanto a los literales de cadena, los escapes octales siempre tienen como máximo tres dígitos de longitud.

Distinto en la versión 3.3: Se han añadido las secuencias de escape `'\u'` y `'\U'`.

Distinto en la versión 3.6: Los escapes desconocidos que consisten en `'\'` y una letra ASCII ahora son errores.

Distinto en la versión 3.8: Se añadió la secuencia de escape `'\N{name}'`. Como en los literales de cadena, se expande al carácter Unicode nombrado (p. ej. `'\N{EM DASH}'`).

Contenidos del módulo

El módulo define varias funciones, constantes y una excepción. Algunas de las funciones son versiones simplificadas de los métodos completos de las expresiones regulares compiladas. La mayoría de las aplicaciones no triviales utilizan siempre la forma compilada.

Indicadores

Distinto en la versión 3.6: Ahora las constantes de indicadores son instancias de `RegexFlag`, que es una subclase de `enum.IntFlag`.

class re.RegexFlag

Una clase `enum.IntFlag` contiene las opciones regex que se enumeran a continuación.

Added in version 3.11: - added to `__all__`

re.A
re.ASCII

Make `\w`, `\W`, `\b`, `\B`, `\d`, `\D`, `\s` and `\S` perform ASCII-only matching instead of full Unicode matching. This is only meaningful for Unicode (str) patterns, and is ignored for bytes patterns.

Corresponds to the inline flag `(?a)`.

Nota

The `U` flag still exists for backward compatibility, but is redundant in Python 3 since matches are Unicode by default for `str` patterns, and Unicode matching isn't allowed for bytes patterns. `UNICODE` and the inline flag `(?u)` are similarly redundant.

re.DEBUG

Display debug information about compiled expression.

No corresponding inline flag.

re.I
re.IGNORECASE

Perform case-insensitive matching; expressions like `[A-Z]` will also match lowercase letters. Full Unicode matching (such as `Ü` matching `ü`) also works unless the **`ASCII`** flag is used to disable non-ASCII matches. The current locale does not change the effect of this flag unless the **`LOCALE`** flag is also used.

Corresponds to the inline flag `(?i)`.

Note that when the Unicode patterns `[a-z]` or `[A-Z]` are used in combination with the **`IGNORECASE`** flag, they will match the 52 ASCII letters and 4 additional non-ASCII letters: "İ" (U+0130, Latin capital letter I with dot above), "ı" (U+0131, Latin small letter dotless i), "ſ" (U+017F, Latin small letter long s) and "K" (U+212A, Kelvin sign). If the **`ASCII`** flag is used, only letters "a" to "z" and "A" to "Z" are matched.

re.L
re.LOCALE

Make `\w`, `\W`, `\b`, `\B` and case-insensitive matching dependent on the current locale. This flag can be used only with bytes patterns.

Corresponds to the inline flag `(?L)`.

Advertencia

This flag is discouraged; consider Unicode matching instead. The locale mechanism is very unreliable as it only handles one «culture» at a time and only works with 8-bit locales. Unicode matching is enabled by default for Unicode (str) patterns and it is able to handle different locales and languages.

Distinto en la versión 3.6: **`LOCALE`** can be used only with bytes

patterns and is not compatible with **`ASCII`**.

Distinto en la versión 3.7: Compiled regular expression objects with the **`LOCALE`** flag no longer depend on the locale at compile time. Only the locale at matching time affects the result of matching.

re.M
re.MULTILINE

When specified, the pattern character `'^'` matches at the beginning of the string and at the beginning of each line (immediately following each newline); and the pattern character `'$'` matches at the end of the string and at the end of each line (immediately preceding each newline). By default, `'^'` matches only at the beginning of the string, and `'$'` only at the end of the string and immediately before the newline (if any) at the end of the string.

Corresponds to the inline flag `(?m)`.

re.NOFLAG

Indica que no se aplica ninguna bandera, el valor es `0`. Esta bandera puede ser utilizada como valor predeterminado para un argumento de palabra clave de función o como un valor que será condicionalmente combinado con otras banderas usando el operador binario OR. Ejemplo de uso como valor predeterminado:

```
def myfunc(text, flag=re.NOFLAG):
    return re.match(text, flag)
```

Added in version 3.11.

re.S
re.DOTALL

Make the `'.'` special character match any character at all, including a newline; without this flag, `'.'` will match anything *except* a newline.

Corresponds to the inline flag `(?s)`.

re.U
re.UNICODE

In Python 3, Unicode characters are matched by default for `str` patterns. This flag is therefore redundant with **no effect** and is only kept for backward compatibility.

See **`ASCII`** to restrict matching to ASCII characters instead.

re.X
re.VERBOSE

Esta bandera permite escribir expresiones regulares que se ven mejor y son más legibles, facilitando la separación visual de las secciones lógicas del patrón y la adición de comentarios. Los espacios en blanco dentro del patrón se ignoran, excepto cuando están en una clase de caracteres, o cuando están precedidos por una barra inversa sin escapar, o dentro de tokens como `*?`, `(?:` o `(?P<...>`. Por ejemplo, `(? :` y `* ?` no están permitidos. Cuando una línea contiene un `#` que no está en una clase de caracteres y no está precedida por una barra inversa sin escapar, todos los caracteres desde el `#` más a la izquierda hasta el final de la línea son ignorados.

Esto significa que los dos siguientes objetos expresión regular que coinciden con un número decimal son funcionalmente iguales:

```
a = re.compile(r"""\d +  # the integral part
                   \.    # the decimal point
                   \d *  # some fractional digits""
b = re.compile(r"\d+\.\d*")
```

Corresponde al indicador en línea `(?x)`.

Funciones

re.compile(*pattern*, *flags=0*)

Compila un patrón de expresión regular en un objeto de expresión regular, que puede ser usado para las coincidencias usando `match()`, `search()` y otros métodos, descritos más adelante.

The expression's behaviour can be modified by specifying a *flags* value. Values can be any of the flags variables, combined using bitwise OR (the `|` operator).

La secuencia

```
prog = re.compile(pattern)
result = prog.match(string)
```

es equivalente a

```
result = re.match(pattern, string)
```

pero usando `re.compile()` y guardando el objeto resultante de la expresión regular para su reutilización es más eficiente cuando la expresión será usada varias veces en un solo programa.

Nota

Las versiones compiladas de los patrones más recientes pasaron a `re.compile()` y las funciones de coincidencia a nivel de módulo están en caché, así que los programas que usan sólo unas pocas expresiones regulares a la vez no tienen que preocuparse de compilar expresiones regulares.

re.search(*pattern*, *string*, *flags*=*0*)

Explora la cadena de caracteres *string* en busca de la primera ubicación donde el patrón *pattern* de la expresión regular produce una coincidencia, y retorna un `Match` correspondiente. Retorna `None` si ninguna posición en la cadena coincide con el patrón; nota que esto es diferente a encontrar una coincidencia de longitud cero en algún punto de la cadena.

The expression's behaviour can be modified by specifying a *flags* value. Values can be any of the flags variables, combined using bitwise OR (the | operator).

re.match(*pattern*, *string*, *flags=0*)

Si cero o más caracteres al principio de la cadena *string* coinciden con el patrón *pattern* de la expresión regular, retorna un **`Match`** correspondiente. Retorna `None` si la cadena no coincide con el patrón; notar que esto es diferente de una coincidencia de longitud cero.

Notar que incluso en el modo **`MULTILINE`**, **`re.match()`** sólo coincidirá al principio de la cadena y no al principio de cada línea.

Si se quiere localizar una coincidencia en cualquier lugar de la *string* («cadena»), se utiliza **`search()`** en su lugar (ver también search() vs. match()).

The expression's behaviour can be modified by specifying a *flags* value. Values can be any of the flags variables, combined using bitwise OR (the | operator).

re.fullmatch(*pattern*, *string*, *flags=0*)

Si toda la cadena *string* coincide con el patrón *pattern* de la expresión regular, retorna un **`Match`** correspondiente. Retorna `None` si la cadena no coincide con el patrón; notar que esto es diferente de una coincidencia de longitud cero.

The expression's behaviour can be modified by specifying a *flags* value. Values can be any of the flags variables, combined using bitwise OR (the | operator).

Added in version 3.4.

re.split(*pattern*, *string*, *maxsplit=0*, *flags=0*)

Divide la *string* («cadena») por el número de ocurrencias del *pattern* («patrón»). Si se utilizan paréntesis de captura en *pattern*, entonces el texto de todos los grupos en el patrón también se retornan como parte de la lista resultante. Si *maxsplit* (máxima divisibilidad) es distinta de cero, como mucho se producen *maxsplit* divisiones, y el resto de la cadena se retorna como elemento final de la lista.

```
>>> re.split(r'\W+', 'Words, words, words.')
['Words', 'words', 'words', '']
>>> re.split(r'(\W+)', 'Words, words, words.')
['Words', ', ', 'words', ', ', 'words', '.', '']
>>> re.split(r'\W+', 'Words, words, words.', maxspl
['Words', 'words, words.']
>>> re.split('[a-f]+', '0a3B9', flags=re.IGNORECASE
['0', '3', '9']
```

Si hay grupos de captura en el separador y coincide al principio de la cadena, el resultado comenzará con una cadena vacía. Lo mismo ocurre con el final de la cadena:

```
>>> re.split(r'(\W+)', '...words, words...')
['', '...', 'words', ', ', 'words', '...', '']
```

De esa manera, los componentes de los separadores se encuentran siempre en los mismos índices relativos dentro de la lista de resultados.

Adjacent empty matches are not possible, but an empty match can occur immediately after a non-empty match.

```
>>> re.split(r'\b', 'Words, words, words.')
['', 'Words', ', ', 'words', ', ', 'words', '.']
>>> re.split(r'\W*', '...words...')
['', '', 'w', 'o', 'r', 'd', 's', '', '']
>>> re.split(r'(\W*)', '...words...')
['', '...', '', '', 'w', '', 'o', '', 'r', '', 'd',
```

The expression's behaviour can be modified by specifying a *flags* value. Values can be any of the flags variables, combined using bitwise OR (the `|` operator).

Distinto en la versión 3.1: Se añadió el argumento de los indicadores opcionales.

Distinto en la versión 3.7: Se añadió el soporte de la división en un patrón que podría coincidir con una cadena vacía.

Obsoleto desde la versión 3.13: Passing *maxsplit* and *flags* as positional arguments is deprecated. In future Python versions they will be keyword-only parameters.

re.findall(*pattern*, *string*, *flags=0*)

Retorna todas las coincidencias no superpuestas de *pattern* en *string*, como una lista de strings o tuplas. El *string* se escanea de izquierda a derecha y las coincidencias se retornan en el orden en que se encuentran. Las coincidencias vacías se incluyen en el resultado.

El resultado depende del número de grupos detectados en el patrón. Si no hay grupos, retorna una lista de strings que coincidan con el patrón completo. Si existe exactamente un grupo, retorna una lista de strings que coincidan con ese grupo. Si hay varios grupos presentes, retorna una lista de tuplas de strings que coinciden con los grupos. Los grupos que no son detectados no afectan la forma del resultado.

```
>>> re.findall(r'\bf[a-z]*', 'which foot or hand fe
['foot', 'fell', 'fastest']
>>> re.findall(r'(\w+)=(\d+)', 'set width=20 and he
[('width', '20'), ('height', '10')]
```

The expression’s behaviour can be modified by specifying a *flags* value. Values can be any of the flags variables, combined using bitwise OR (the `|` operator).

Distinto en la versión 3.7: Las coincidencias no vacías ahora pueden empezar justo después de una coincidencia vacía anterior.

re.finditer(*pattern*, *string*, *flags=0*)

Retorna un iterador que produce objetos **`Match`** sobre todas las coincidencias no superpuestas para el patrón de RE *pattern* en la *string*. La *string* es examinada de izquierda a derecha, y las coincidencias son retornadas en el orden en que se encuentran. Las coincidencias vacías se incluyen en el resultado.

The expression’s behaviour can be modified by specifying a *flags* value. Values can be any of the flags variables, combined using bitwise OR (the `|` operator).

Distinto en la versión 3.7: Las coincidencias no vacías ahora pueden empezar justo después de una coincidencia vacía anterior.

re.sub(*pattern*, *repl*, *string*, *count=0*, *flags=0*)

Retorna la cadena obtenida reemplazando las ocurrencias no superpuestas del *pattern* («patrón») en la *string* («cadena») por el reemplazo de *repl*. Si el patrón no se encuentra, se retorna *string* sin cambios. *repl* puede ser una cadena o una función; si es una cadena, cualquier barra inversa escapada en ella es procesada. Es decir, `\n` se convierte en un carácter de una sola línea nueva, `\r` se convierte en un retorno de carro, y así sucesivamente. Los escapes desconocidos de las letras ASCII se reservan para un uso futuro y se tratan como errores. Otros escapes desconocidos como `\&` no se utilizan. Las referencias inversas, como `\6`, se reemplazan por la subcadena que corresponde al grupo 6 del patrón. Por ejemplo:

```
>>> re.sub(r'def\s+([a-zA-Z_][a-zA-Z_0-9]*)\s*\(\s*
...        r'static PyObject*\npy_\1(void)\n{',
...        'def myfunc():')
'static PyObject*\npy_myfunc(void)\n{'
```

Si *repl* es una función, se llama para cada ocurrencia no superpuesta de *pattern*. La función toma un solo argumento **`Match`**, y retorna la cadena de sustitución. Por ejemplo:

```
>>> def dashrepl(matchobj):
...     if matchobj.group(0) == '-': return ' '
...     else: return '-'
...
```

```
>>> re.sub('-{1,2}', dashrepl, 'pro----gram-files')
'pro--gram files'
>>> re.sub(r'\sAND\s', ' & ', 'Baked Beans And Spam
'Baked Beans & Spam'
```

El patrón puede ser una cadena o un **`Pattern`**.

The optional argument *count* is the maximum number of pattern occurrences to be replaced; *count* must be a non-negative integer. If omitted or zero, all occurrences will be replaced.

Adjacent empty matches are not possible, but an empty match can occur immediately after a non-empty match. As a result, `sub('x*', '-', 'abxd')` returns `'-a-b--d-'` instead of `'-a-b-d-'`.

En los argumentos *repl* de tipo cadena, además de los escapes de caracteres y las referencias inversas descritas anteriormente, `\g<name>` usará la subcadena coincidente con el grupo llamado `name`, como se define en la sintaxis `(?P<name>...)`. `\g<number>` utiliza el número de grupo correspondiente; `\g<2>` es por lo tanto equivalente a `\2`, pero no es ambiguo en un reemplazo como sucede con `\g<2>0`. `\20` se interpretaría como una referencia al grupo 20, no como una referencia al grupo 2 seguido del carácter literal `'0'`. La referencia inversa `\g<0>` sustituye en toda la subcadena coincidente con la RE.

The expression's behaviour can be modified by specifying a *flags* value. Values can be any of the flags variables, combined using bitwise OR (the `|` operator).

Distinto en la versión 3.1: Se añadió el argumento de los indicadores opcionales.

Distinto en la versión 3.5: Los grupos no coincidentes son reemplazados por una cadena vacía.

Distinto en la versión 3.6: Los escapes desconocidos en el *pattern* que consisten en `'\'` y una letra ASCII ahora son errores.

Distinto en la versión 3.7: Unknown escapes in *repl* consisting of `'\'` and an ASCII letter now are errors. An empty match can occur immediately after a non-empty match.

Distinto en la versión 3.12: El *id* del grupo solo puede contener dígitos ASCII. En las cadenas de reemplazo **`bytes`**, el nombre del grupo *name* solo puede contener bytes en el rango ASCII (`b'\x00'`-`b'\x7f'`).

Obsoleto desde la versión 3.13: Passing *count* and *flags* as positional arguments is deprecated. In future Python versions they will be keyword-only parameters.

re.subn(*pattern*, *repl*, *string*, *count=0*, *flags=0*)

Realiza la misma operación que **`sub()`**, pero retorna una tupla `(new_string, number_of_subs_made)`.

The expression's behaviour can be modified by specifying a *flags* value. Values can be any of the flags variables, combined using bitwise OR (the `|` operator).

re.escape(*pattern*)

Caracteres de escape especiales en *pattern* (» patrón»). Esto es útil si quieres hacer coincidir una cadena literal arbitraria que puede tener metacaracteres de expresión regular en ella. Por ejemplo:

```
>>> print(re.escape('https://www.python.org'))
https://www\.python\.org

>>> legal_chars = string.ascii_lowercase + string.d
>>> print('[%s]+' % re.escape(legal_chars))
[abcdefghijklmnopqrstuvwxyz0123456789!\#\$%\&'\*\+\

>>> operators = ['+', '-', '*', '/', '**']
>>> print('|'.join(map(re.escape, sorted(operators,
/|\-|\+|\*\*|\*
```

Esta función no debe usarse para la cadena de reemplazo en **`sub()`** y **`subn()`**, sólo deben escaparse las barras inversas. Por ejemplo:

```
>>> digits_re = r'\d+'
>>> sample = '/usr/sbin/sendmail - 0 errors, 12 war
>>> print(re.sub(digits_re, digits_re.replace('\\',
/usr/sbin/sendmail - \d+ errors, \d+ warnings
```

Distinto en la versión 3.3: El carácter de `'_'` ya no se escapa.

Distinto en la versión 3.7: Sólo se escapan los caracteres que pueden tener un significado especial en una expresión regular. Como resultado, `'!'`, `'"'`, `'%'`, `"'"`, `','`, `'/'`, `':'`, `';'`, `'<'`, `'='`, `'>'`, `'@'` y ``"`"`` ya no se escapan.

re.purge()

Despeja la caché de expresión regular.

Excepciones

exception re.PatternError(*msg*, *pattern=None*, *pos=None*)

Exception raised when a string passed to one of the functions here is not a valid regular expression (for example, it might contain unmatched parentheses) or when some other error occurs during compilation or matching. It is never an error if a string contains no match for a pattern. The `PatternError` instance has the following additional attributes:

msg

El mensaje de error sin formato.

pattern

El patrón de expresión regular.

pos

El índice en *pattern* («patrón») donde la compilación falló (puede ser `None`).

lineno

La línea correspondiente a *pos* (puede ser `None`).

colno

La columna correspondiente a *pos* (puede ser `None`).

Distinto en la versión 3.5: Se añadieron atributos adicionales.

Distinto en la versión 3.13: `PatternError` was originally named `error`; the latter is kept as an alias for backward compatibility.

Objetos expresión regular

class re.Pattern

Objeto de expresión regular compilada devuelto por **`re.compile()`**.

Distinto en la versión 3.9: **`re.Pattern`** soporta `[]` para indicar un patrón Unicode (str) o de bytes. Ver Tipo Alias Genérico.

Pattern.search(*string*[, *pos*[, *endpos*]])

Escanea a través de la cadena *string* buscando la primera ubicación donde esta expresión regular produce una coincidencia, y retorna un **`Match`** correspondiente. Retorna `None` si ninguna posición en la cadena coincide con el patrón; notar que esto es diferente a encontrar una coincidencia de longitud cero en algún punto de la cadena.

El segundo parámetro opcional *pos* proporciona un índice en la cadena donde la búsqueda debe comenzar; por defecto es `0`. Esto no es completamente equivalente a dividir la cadena; el patrón de carácter `'^'` coincide en el inicio real de la cadena y en las posiciones justo después de una nueva línea, pero no necesariamente en el índice donde la búsqueda va a comenzar.

El parámetro opcional *endpos* limita hasta dónde se buscará la cadena; será como si la cadena fuera de *endpos* caracteres de largo, por lo que sólo se buscará una coincidencia entre los caracteres de *pos* a `endpos - 1`. Si *endpos* es menor que *pos*, no se encontrará ninguna coincidencia; de lo contrario, si *rx* es un objeto de expresión regular compilado, `rx.search(string, 0, 50)` es equivalente a `rx.search(string[:50], 0)`.

```
>>> pattern = re.compile("d")
>>> pattern.search("dog")     # Match at index 0
<re.Match object; span=(0, 1), match='d'>
>>> pattern.search("dog", 1)  # No match; search do
```

Pattern.match(*string*[, *pos*[, *endpos*]])

Si cero o más caracteres en el comienzo *beginning* de la cadena *string* coinciden con esta expresión regular, retorna un **`Match`** correspondiente. Retorna `None` si la cadena no coincide con el patrón; notar que esto es diferente de una coincidencia de longitud cero.

Los parámetros opcionales *pos* y *endpos* tienen el mismo significado que para el método **`search()`**.

```
>>> pattern = re.compile("o")
>>> pattern.match("dog")      # No match as "o" is
>>> pattern.match("dog", 1)   # Match as "o" is the
<re.Match object; span=(1, 2), match='o'>
```

Si se quiere encontrar una coincidencia en cualquier lugar de *string*, utilizar **`search()`** en su lugar (ver también search() vs. match()).

Pattern.fullmatch(*string*[, *pos*[, *endpos*]])

Si toda la cadena *string* coincide con esta expresión regular, retorna un **`Match`** correspondiente. Retorna `None` si la cadena no coincide con el patrón; notar que esto es diferente de una coincidencia de longitud cero.

Los parámetros opcionales *pos* y *endpos* tienen el mismo significado que para el método **`search()`**.

```
>>> pattern = re.compile("o[gh]")
>>> pattern.fullmatch("dog")      # No match as "o"
>>> pattern.fullmatch("ogre")     # No match as not
>>> pattern.fullmatch("doggie", 1, 3)   # Matches w
<re.Match object; span=(1, 3), match='og'>
```

Added in version 3.4.

Pattern.split(*string*, *maxsplit=0*)

Idéntico a la función **`split()`**, usando el patrón compilado.

Pattern.findall(*string*[, *pos*[, *endpos*]])

Similar a la función **`findall()`**, usando el patrón compilado, pero también acepta parámetros opcionales *pos* y *endpos* que limitan la región de búsqueda como para **`search()`**.

Pattern.finditer(*string*[, *pos*[, *endpos*]])

Similar a la función **`finditer()`**, usando el patrón compilado, pero también acepta parámetros opcionales *pos* y *endpos* que limitan la región de búsqueda como para **`search()`**.

Pattern.sub(*repl*, *string*, *count=0*)

Idéntico a la función **`sub()`**, usando el patrón compilado.

Pattern.subn(*repl*, *string*, *count=0*)

Idéntico a la función **`subn()`**, usando el patrón compilado.

Pattern.flags

The regex matching flags. This is a combination of the flags given to **`compile()`**, any `(?...)` inline flags in the pattern, and implicit flags such as **`UNICODE`** if the pattern is a Unicode string.

Pattern.groups

El número de grupos de captura en el patrón.

Pattern.groupindex

Un diccionario que mapea cualquier nombre de grupo simbólico definido por `(?P<id>)` para agrupar números. El diccionario está vacío si no se utilizaron grupos simbólicos en el patrón.

Pattern.pattern

La cadena de patrones a partir de la cual el objeto de patrón fue compilado.

Distinto en la versión 3.7: Se añadió el soporte de **`copy.copy()`** y **`copy.deepcopy()`**. Los objetos expresión regular compilados se consideran atómicos.

Objetos de coincidencia

Los objetos de coincidencia siempre tienen un valor booleano de `True` («Verdadero»). Ya que **`match()`** y **`search()`** retornan `None` cuando no hay coincidencia. Se puede probar si hubo una coincidencia con una simple declaración `if`:

```
match = re.search(pattern, string)
if match:
    process(match)
```

class re.Match

Objeto Match devuelto por llamadas exitosas a `match` y `search`.

Distinto en la versión 3.9: **`re.Match`** soporta `[]` para indicar una coincidencia Unicode (str) o de bytes. Ver Tipo Alias Genérico.

Match.expand(*template*)

Return the string obtained by doing backslash substitution on the template string *template*, as done by the **`sub()`** method. Escapes such as `\n` are converted to the appropriate characters, and numeric backreferences (`\1`, `\2`) and named backreferences (`\g<1>`, `\g<name>`) are replaced by the contents of the corresponding group. The backreference `\g<0>` will be replaced by the entire match.

Distinto en la versión 3.5: Los grupos no coincidentes son reemplazados por una cadena vacía.

Match.group([*group1*, ...])

Returns one or more subgroups of the match. If there is a single argument, the result is a single string; if there are multiple arguments, the result is a tuple with one item per argument. Without arguments, *group1* defaults to zero (the whole match is returned). If a *groupN* argument is zero, the corresponding return value is the entire matching string; if it is a positive integer, it is the string matching the corresponding parenthesized group. If a group number is negative or larger than the number of groups defined in the pattern, an **`IndexError`** exception is raised. If a group is contained in a part of the pattern that did not match, the corresponding result is `None`. If a group is contained in a part of the pattern that matched multiple times, the last match is returned.

```
>>> m = re.match(r"(\w+) (\w+)", "Isaac Newton, phy
>>> m.group(0)       # The entire match
'Isaac Newton'
>>> m.group(1)       # The first parenthesized subg
'Isaac'
>>> m.group(2)       # The second parenthesized sub
'Newton'
>>> m.group(1, 2)    # Multiple arguments give us a
('Isaac', 'Newton')
```

Si la expresión regular usa la sintaxis `(?P<name>...)`, los argumentos *groupN* también pueden ser cadenas que identifican a los grupos por su nombre de grupo. Si un argumento de cadena no se usa como nombre de grupo en el patrón, se produce una excepción **`IndexError`**.

Un ejemplo moderadamente complicado:

```
>>> m = re.match(r"(?P<first_name>\w+) (?P<last_nam
>>> m.group('first_name')
'Malcolm'
>>> m.group('last_name')
'Reynolds'
```

Los grupos nombrados también pueden ser referidos por su índice:

```
>>> m.group(1)
'Malcolm'
>>> m.group(2)
'Reynolds'
```

Si un grupo coincide varias veces, sólo se puede acceder a la última coincidencia:

```
>>> m = re.match(r"(..)+", "a1b2c3")  # Matches 3 t
>>> m.group(1)                        # Returns onl
'c3'
```

Match.__getitem__(*g*)

Esto es idéntico a `m.group(g)`. Esto permite un acceso más fácil a un grupo individual de una coincidencia:

```
>>> m = re.match(r"(\w+) (\w+)", "Isaac Newton, phy
>>> m[0]       # The entire match
'Isaac Newton'
>>> m[1]       # The first parenthesized subgroup.
'Isaac'
>>> m[2]       # The second parenthesized subgroup.
'Newton'
```

Los grupos con nombre también son compatibles:

```
>>> m = re.match(r"(?P<first_name>\w+) (?P<last_nam
>>> m['first_name']
'Isaac'
>>> m['last_name']
'Newton'
```

Added in version 3.6.

Match.groups(*default=None*)

Retorna una tupla que contenga todos los subgrupos de la coincidencia, desde 1 hasta tantos grupos como haya en el patrón. El argumento *default* («por defecto») se utiliza para los grupos que no participaron en la coincidencia; por defecto es `None`.

Por ejemplo:

```
>>> m = re.match(r"(\d+)\.(\d+)", "24.1632")
>>> m.groups()
('24', '1632')
```

Si hacemos que el decimal y todo lo que sigue sea opcional, no todos los grupos podrían participar en la coincidencia. Estos grupos serán por defecto `None` a menos que se utilice el argumento *default*:

```
>>> m = re.match(r"(\d+)\.?(\d+)?", "24")
>>> m.groups()      # Second group defaults to None
('24', None)
>>> m.groups('0')   # Now, the second group default
('24', '0')
```

Match.groupdict(*default=None*)

Retorna un diccionario que contiene todos los subgrupos *nombrados* de la coincidencia, tecleado por el nombre del subgrupo. El argumento *por defecto* se usa para los grupos que no participaron en la coincidencia; por defecto es `None`. Por ejemplo:

```
>>> m = re.match(r"(?P<first_name>\w+) (?P<last_nam
>>> m.groupdict()
{'first_name': 'Malcolm', 'last_name': 'Reynolds'}
```

Match.start([*group*])

Match.end([*group*])

Retorna los índices del comienzo y el final de la subcadena coincidiendo con el *group*; el *group* por defecto es cero (es decir, toda la subcadena coincidente). Retorna `-1` si *grupo* existe pero no ha contribuido a la coincidencia. Para un objeto coincidente *m*, y un grupo *g* que sí contribuyó a la coincidencia, la subcadena coincidente con el grupo *g* (equivalente a `m.group(g)`) es

```
m.string[m.start(g):m.end(g)]
```

Notar que `m.start(group)` será igual a `m.end(group)` si *group* coincidió con una cadena nula. Por ejemplo, después de `m = re.search('b(c?)', 'cba')`, `m.start(0)` es 1, `m.end(0)` es 2, `m.start(1)` y `m.end(1)` son ambos 2, y `m.start(2)` produce una excepción **`IndexError`**.

Un ejemplo que eliminará *remove_this* («quita esto») de las direcciones de correo electrónico:

```
>>> email = "tony@tiremove_thisger.net"
>>> m = re.search("remove_this", email)
>>> email[:m.start()] + email[m.end():]
'tony@tiger.net'
```

Match.span([*group*])

Para una coincidencia *m*, retorna la tupla-2 `(m.inicio(grupo), m.fin(grupo))`. Notar que si *group* no contribuyó a la coincidencia, esto es `(-1, -1)`. *group* por se convierte a cero para toda la coincidencia.

Match.pos

El valor de *pos* que fue pasado al método **`search()`** o

`match()` de un objeto regex. Este es el índice de la cadena en la que el motor RE comenzó a buscar una coincidencia.

Match.endpos

El valor de *endpos* que se pasó al método **`search()`** o **`match()`** de un objeto regex. Este es el índice de la cadena más allá de la cual el motor RE no irá.

Match.lastindex

El índice entero del último grupo de captura coincidente, o``None`` si no hay ningún grupo coincidente. Por ejemplo, las expresiones `(a)b`, `((a)(b))` y `((ab))` tendrán `lastindex == 1` si se aplican a la cadena `'ab'`, mientras que la expresión `(a)(b)` tendrá `lastindex == 2`, si se aplica a la misma cadena.

Match.lastgroup

El nombre del último grupo capturador coincidente, o``None`` si el grupo no tenía nombre, o si no había ningún grupo coincidente.

Match.re

El objeto de expresión regular cuyo método **`match()`** o **`search()`** produce esta instancia de coincidencia.

Match.string

La cadena pasada a **`match()`** o **`search()`**.

Distinto en la versión 3.7: Se añadió el soporte de **`copy.copy()`** y **`copy.deepcopy()`**. Los objetos de coincidencia se consideran atómicos.

Ejemplos de expresiones regulares

Buscando un par

En este ejemplo, se utilizará la siguiente función de ayuda para mostrar los objetos de coincidencia con un poco más de elegancia:

```
def displaymatch(match):
    if match is None:
        return None
    return '<Match: %r, groups=%r>' % (match.group(), ma
```

Supongamos que se está escribiendo un programa de póquer en el que la mano de un jugador se representa como una cadena de 5 caracteres en la que cada carácter representa una carta, «a» para el as, «k» para el rey, «q» para la reina, «j» para la jota, «t» para el 10, y del « 2» al «9» representando la carta con ese valor.

Para ver si una cadena dada es una mano válida, se podría hacer lo siguiente:

```
>>> valid = re.compile(r"^[a2-9tjqk]{5}$")
>>> displaymatch(valid.match("akt5q"))  # Valid.
"<Match: 'akt5q', groups=()>"
>>> displaymatch(valid.match("akt5e"))  # Invalid.
>>> displaymatch(valid.match("akt"))    # Invalid.
>>> displaymatch(valid.match("727ak"))  # Valid.
"<Match: '727ak', groups=()>"
```

Esa última mano, `"727ak"`, contenía un par, o dos de las mismas cartas de valor. Para igualar esto con una expresión regular, se podrían usar referencias inversas como tales:

```
>>> pair = re.compile(r".*(.).*\1")
>>> displaymatch(pair.match("717ak"))     # Pair of 7s.
"<Match: '717', groups=('7',)>"
>>> displaymatch(pair.match("718ak"))     # No pairs.
>>> displaymatch(pair.match("354aa"))     # Pair of aces
"<Match: '354aa', groups=('a',)>"
```

Para averiguar en qué carta consiste el par, se podría utilizar el método **`group()`** del objeto de coincidencia de la siguiente manera:

```
>>> pair = re.compile(r".*(.).*\1")
>>> pair.match("717ak").group(1)
'7'

# Error because re.match() returns None, which doesn't h
>>> pair.match("718ak").group(1)
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    re.match(r".*(.).*\1", "718ak").group(1)
AttributeError: 'NoneType' object has no attribute 'grou

>>> pair.match("354aa").group(1)
'a'
```

Simular scanf()

Actualmente, Python no tiene un equivalente a **`scanf()`**. Las expresiones regulares son generalmente más poderosas, aunque también más detalladas, que las cadenas de formato de **`scanf()`**. La tabla siguiente ofrece algunas correspondencias más o menos equivalentes entre los tokens de formato de **`scanf()`** y expresiones regulares.

Token `scanf()`	Expresión regular
`%c`	`.`
`%5c`	`.{5}`
`%d`	`[-+]?\d+`
`%e`, `%E`, `%f`, `%g`	`[-+]?(\d+(\.\d*)?|\.\d+)([eE][-+]?\d+)?`
`%i`	`[-+]?(0[xX][\dA-Fa-f]+|0[0-7]*|\d+)`
`%o`	`[-+]?[0-7]+`
`%s`	`\S+`
`%u`	`\d+`
`%x`, `%X`	`[-+]?(0[xX])?[\dA-Fa-f]+`

Para extraer el nombre de archivo y los números de una cadena como

```
/usr/sbin/sendmail - 0 errors, 4 warnings
```

utilizaría un formato **`scanf()`** como

```
%s - %d errors, %d warnings
```

La expresión regular equivalente sería

```
(\S+) - (\d+) errors, (\d+) warnings
```

search() vs. match()

Python ofrece diferentes operaciones primitivas basadas en expresiones regulares:

- **`re.match()`** verifica una coincidencia solo al principio de la cadena
- **`re.search()`** busca una coincidencia en cualquier parte de la cadena (esto es lo que Perl hace por defecto)
- **`re.fullmatch()`** verifica si la cadena completa es una coincidencia

Por ejemplo:

```
>>> re.match("c", "abcdef")     # No match
>>> re.search("c", "abcdef")    # Match
<re.Match object; span=(2, 3), match='c'>
>>> re.fullmatch("p.*n", "python") # Match
<re.Match object; span=(0, 6), match='python'>
>>> re.fullmatch("r.*n", "python") # No match
```

Las expresiones regulares que comienzan con `'^'` pueden ser usadas con **`search()`** para restringir la coincidencia al principio de la cadena:

```
>>> re.match("c", "abcdef")     # No match
>>> re.search("^c", "abcdef")   # No match
>>> re.search("^a", "abcdef")   # Match
<re.Match object; span=(0, 1), match='a'>
```

Notar, sin embargo, que en el modo **`MULTILINE`** **`match()`** sólo coincide al principio de la cadena, mientras que usando **`search()`** con una expresión regular que comienza con `'^'` coincidirá al principio de cada línea.

```
>>> re.match("X", "A\nB\nX", re.MULTILINE)  # No match
>>> re.search("^X", "A\nB\nX", re.MULTILINE)  # Match
<re.Match object; span=(4, 5), match='X'>
```

Haciendo una guía telefónica

`split()` divide una cadena en una lista delimitada por el patrón recibido. El método es muy útil para convertir datos textuales en estructuras de datos que pueden ser fácilmente leídas y modificadas por Python, como se demuestra en el siguiente ejemplo en el que se crea una guía telefónica.

Primero, aquí está la información. Normalmente puede venir de un archivo, aquí se usa la sintaxis de cadena de triple comilla

```
>>> text = """Ross McFluff: 834.345.1254 155 Elm Street
...
... Ronald Heathmore: 892.345.3428 436 Finley Avenue
... Frank Burger: 925.541.7625 662 South Dogwood Way
...
...
... Heather Albrecht: 548.326.4584 919 Park Place"""
```

Las entradas (*entries*) están separadas por una o más líneas nuevas. Ahora se convierte la cadena en una lista en la que cada línea no vacía tiene su propia entrada:

```
>>> entries = re.split("\n+", text)
>>> entries
['Ross McFluff: 834.345.1254 155 Elm Street',
'Ronald Heathmore: 892.345.3428 436 Finley Avenue',
'Frank Burger: 925.541.7625 662 South Dogwood Way',
'Heather Albrecht: 548.326.4584 919 Park Place']
```

Finalmente, se divide cada entrada en una lista con nombre, apellido, número de teléfono y dirección. Se utiliza el parámetro `maxsplit` (división máxima) de **`split()`** porque la dirección tiene espacios dentro del patrón de división:

```
>>> [re.split(":? ", entry, maxsplit=3) for entry in ent
```

```
[['Ross', 'McFluff', '834.345.1254', '155 Elm Street'],
['Ronald', 'Heathmore', '892.345.3428', '436 Finley Aver
['Frank', 'Burger', '925.541.7625', '662 South Dogwood W
['Heather', 'Albrecht', '548.326.4584', '919 Park Place'
```

El patrón `:?` coincide con los dos puntos después del apellido, de manera que no aparezca en la lista de resultados. Con `maxsplit` de `4`, se podría separar el número de casa del nombre de la calle:

```
>>> [re.split(":? ", entry, maxsplit=4) for entry in ent
[['Ross', 'McFluff', '834.345.1254', '155', 'Elm Street'
['Ronald', 'Heathmore', '892.345.3428', '436', 'Finley A
['Frank', 'Burger', '925.541.7625', '662', 'South Dogwoo
['Heather', 'Albrecht', '548.326.4584', '919', 'Park Pla
```

Mungear texto

`sub()` reemplaza cada ocurrencia de un patrón con una cadena o el resultado de una función. Este ejemplo demuestra el uso de **`sub()`** con una función para «mungear» (*munge*) el texto, o aleatorizar el orden de todos los caracteres en cada palabra de una frase excepto el primer y último carácter:

```
>>> def repl(m):
...     inner_word = list(m.group(2))
...     random.shuffle(inner_word)
...     return m.group(1) + "".join(inner_word) + m.grou
...
>>> text = "Professor Abdolmalek, please report your abs
>>> re.sub(r"(\w)(\w+)(\w)", repl, text)
'Poefsrosr Aealmlobdk, pslaee reorpt your abnseces plmrp
>>> re.sub(r"(\w)(\w+)(\w)", repl, text)
'Pofsroser Aodlambelk, plasee reoprt yuor asnebces potlm
```

Encontrar todos los adverbios

`findall()` coincide con *todas* las ocurrencias de un patrón, no sólo con la primera, como lo hace `search()`. Por ejemplo, si un escritor quisiera encontrar todos los adverbios en algún texto, podría usar **`findall()`** de la siguiente manera:

```
>>> text = "He was carefully disguised but captured quic
>>> re.findall(r"\w+ly\b", text)
['carefully', 'quickly']
```

Encontrar todos los adverbios y sus posiciones

Si se desea obtener más información sobre todas las coincidencias de un patrón en lugar del texto coincidente, **`finditer()`** es útil ya que proporciona **`Match`** objetos en lugar de cadenas. Continuando con el ejemplo anterior, si un escritor quisiera encontrar todos los adverbios *y sus posiciones* en algún texto, usaría **`finditer()`** de la siguiente manera:

```
>>> text = "He was carefully disguised but captured quic
>>> for m in re.finditer(r"\w+ly\b", text):
...     print('%02d-%02d: %s' % (m.start(), m.end(), m.g
07-16: carefully
40-47: quickly
```

Notación de cadena *raw*

La notación de cadena *raw* (`r "text"`) permite escribir expresiones regulares razonables. Sin ella, para «escapar» cada barra inversa (`'\'`) en una expresión regular tendría que ser precedida por otra. Por ejemplo, las dos siguientes líneas de código son funcionalmente idénticas:

```
>>> re.match(r"\W(.)\1\W", " ff ")
<re.Match object; span=(0, 4), match=' ff '>
>>> re.match("\\W(.)\\1\\W", " ff ")
<re.Match object; span=(0, 4), match=' ff '>
```

Cuando uno quiere igualar una barra inversa literal, debe escaparse en la expresión regular. Con la notación de cadena *raw*, esto significa `r"\\"`. Sin la notación de cadena, uno debe usar `"\\\\"`, haciendo que las siguientes líneas de código sean funcionalmente idénticas:

```
>>> re.match(r"\\", r"\\")
<re.Match object; span=(0, 1), match='\\'>
>>> re.match("\\\\", r"\\")
<re.Match object; span=(0, 1), match='\\'>
```

Escribir un Tokenizador

Un tokenizador o analizador léxico [https://es.wikipedia.org/wiki/Analizador_léxico] analiza una cadena para categorizar grupos de caracteres. Este es un primer paso útil para escribir un compilador o intérprete.

Las categorías de texto se especifican con expresiones regulares. La técnica consiste en combinarlas en una única expresión regular maestra y en hacer un bucle sobre las sucesivas coincidencias:

```
from typing import NamedTuple
import re

class Token(NamedTuple):
    type: str
    value: int | float | str
    line: int
    column: int

def tokenize(code):
    keywords = {'IF', 'THEN', 'ENDIF', 'FOR', 'NEXT', 'G
    token_specification = [
        ('NUMBER',   r'\d+(\.\d*)?'),  # Integer or deci
        ('ASSIGN',   r':='),           # Assignment oper
        ('END',      r';'),            # Statement termi
        ('ID',       r'[A-Za-z]+'),    # Identifiers
        ('OP',       r'[+\-*/]'),      # Arithmetic oper
        ('NEWLINE',  r'\n'),           # Line endings
        ('SKIP',     r'[ \t]+'),       # Skip over spac
        ('MISMATCH', r'.'),            # Any other chara
    ]
    tok_regex = '|'.join('(?P<%s>%s)' % pair for pair i
    line_num = 1
    line_start = 0
    for mo in re.finditer(tok_regex, code):
        kind = mo.lastgroup
        value = mo.group()
        column = mo.start() - line_start
        if kind == 'NUMBER':
            value = float(value) if '.' in value else in
        elif kind == 'ID' and value in keywords:
            kind = value
        elif kind == 'NEWLINE':
            line_start = mo.end()
            line_num += 1
            continue
        elif kind == 'SKIP':
            continue
        elif kind == 'MISMATCH':
            raise RuntimeError(f'{value!r} unexpected on
        yield Token(kind, value, line_num, column)

statements = '''
    IF quantity THEN
        total := total + price * quantity;
        tax := price * 0.05;
    ENDIF;
'''

for token in tokenize(statements):
    print(token)
```

El tokenizador produce el siguiente resultado:

```
Token(type='IF', value='IF', line=2, column=4)
Token(type='ID', value='quantity', line=2, column=7)
Token(type='THEN', value='THEN', line=2, column=16)
Token(type='ID', value='total', line=3, column=8)
Token(type='ASSIGN', value=':=', line=3, column=14)
Token(type='ID', value='total', line=3, column=17)
Token(type='OP', value='+', line=3, column=23)
Token(type='ID', value='price', line=3, column=25)
Token(type='OP', value='*', line=3, column=31)
Token(type='ID', value='quantity', line=3, column=33)
Token(type='END', value=';', line=3, column=41)
Token(type='ID', value='tax', line=4, column=8)
Token(type='ASSIGN', value=':=', line=4, column=12)
Token(type='ID', value='price', line=4, column=15)
Token(type='OP', value='*', line=4, column=21)
Token(type='NUMBER', value=0.05, line=4, column=23)
Token(type='END', value=';', line=4, column=27)
Token(type='ENDIF', value='ENDIF', line=5, column=4)
Token(type='END', value=';', line=5, column=9)
```

[Frie09]

Friedl, Jeffrey. *Mastering Regular Expressions*. 3a ed., O'Reilly Media, 2009. La tercera edición del libro ya no abarca a Python en absoluto, pero la primera edición cubría la escritura de buenos patrones de expresiones regulares con gran detalle.

`difflib` — Helpers for computing deltas

Código fuente: Lib/difflib.py [https://github.com/python/cpython/tree/3.14/Lib/difflib.py]

Este módulo proporciona clases y funciones para comparar secuencias. Se puede utilizar, por ejemplo, para comparar archivos y puede producir información sobre diferencias de archivo en varios formatos, incluidos HTML y contexto y diferencias unificadas. Para comparar directorios y archivos, consulte también el módulo **`filecmp`**.

class difflib.SequenceMatcher

Esta es una clase flexible para comparar pares de secuencias de cualquier tipo, siempre que los elementos de la secuencia sean hashable. El algoritmo básico es anterior, y un poco mas agradable, que el publicado a fines de los 80" por Ratcliff y Obershelp, bajo el nombre hiperbólico de «*gestalt pattern matching*». La idea es encontrar la subsecuencia coincidente contigua mas larga que no contenga elementos «no deseados»; estos elementos «no deseados» son aquellos que no son de interés por algún motivo, como ser lineas en blanco o espacios. (El tratamiento de elementos no deseados es una extensión al algoritmo de Ratcliff y Obershelp). La misma idea se aplica recursivamente a las partes de la secuencia a la derecha e izquierda de cada subsecuencia correspondiente. Esto no proporciona secuencias de edición mínimas, pero tiende a producir coincidencias que «parecen correctas» a las personas.

Complejidad temporal: En el peor de los casos el algoritmo Ratcliff-Obershelp básico es de complejidad cúbica y de complejidad temporal cuadrática en el caso esperado.

`SequenceMatcher` es de complejidad temporal cuadrática en el peor de los casos y el comportamiento del caso esperado depende de manera complicada de cuántos elementos tienen en común las secuencias; en el mejor de los casos la complejidad temporal es lineal.

Heurística automática de elementos no deseados: `SequenceMatcher` implementa un método heurístico que identifica automáticamente a ciertos elementos como no deseados. El método heurístico consiste en contar cuantas veces aparece cada elemento en la secuencia. Si las apariciones del duplicado de un elemento (después del primero) contabilizan mas del 1% de la secuencia, y a su vez la secuencia contiene mas de 200 elementos, este es identificado como «popular» y es considerado no deseado. Este método heurístico puede desactivarse estableciendo el argumento `autojunk` como `False` al crear la clase **`SequenceMatcher`**.

Distinto en la versión 3.2: Added the *autojunk* parameter.

class difflib.Differ

Esta clase se utiliza para comparar secuencias de lineas de texto y producir diferencias o deltas en una forma legible por humanos. Differ usa `SequenceMatcher` tanto para comparar secuencias de lineas, como para comparar secuencias de caracteres entre lineas similares.

Cada linea de un delta de `Differ` comienza con un código de dos letras:

Código	**Significado**
`'- '`	línea única para la secuencia 1
`'+ '`	línea única para la secuencia 2
`'  '`	línea común a ambas secuencias
`'? '`	línea ausente en todas las secuencias de entrada

Lines beginning with "`?`" attempt to guide the eye to intraline differences, and were not present in either input sequence. These lines can be confusing if the sequences contain whitespace characters, such as spaces, tabs or line breaks.

class difflib.HtmlDiff

Esta clase puede ser usada para crear una tabla HTML (o un archivo HTML completo que contenga la tabla) mostrando comparaciones lado a lado y linea por linea del texto, con cambios interlineales e intralineales. La tabla se puede generar en modo de diferencia completa o contextual.

El constructor de esta clase es:

__init__(*tabsize=8*, *wrapcolumn=None*, *linejunk=None*, *charjunk=IS_CHARACTER_JUNK*)

Inicializa una instancia de `HtmlDiff`.

tabsize es un argumento por palabra clave opcional para especificar el espaciado de tabulación. Su valor predeterminado es `8`.

wrapcolumn es un argumento por palabra clave opcional para especificar el número de columnas donde las lineas serán divididas y ajustadas al ancho de columna, su valor por defecto es `None`, donde las lineas no son ajustadas.

linejunk y *charjunk* son argumentos por palabra clave opcionales que serán pasados a `ndiff()` (que es utilizado por `HtmlDiff` para generar las diferencias lado a lado en HTML). Refiérase a la documentación de **`ndiff()`** para conocer los detalles y valores por defecto de sus argumentos.

Los siguientes métodos son públicos:

make_file(*fromlines*, *tolines*, *fromdesc=''*, *todesc=''*, *context=False*, *numlines=5*, ***, *charset='utf-8'*)

Compara *fromlines* y *tolines* (listas de cadenas de texto) y retorna una cadena de caracteres que representa un archivo HTML completo que contiene una tabla mostrando diferencias línea por línea del texto con cambios interlineales e intralineales resaltados.

fromdesc y *todesc* son argumentos por palabra clave opcionales para especificar los encabezados de las columnas desde *fromdesc* hasta *todesc* en el archivo (ambas cadenas están vacías por defecto).

context y *numlines* son parámetros opcionales. Establezca *context* como `True` para mostrar diferencias contextuales, de lo contrario su valor por defecto es `False` que muestra los archivos completos. El valor por defecto de *numlines* es `5`. Cuando *context* es `True`, *numlines* controla el número de lineas de contexto que rodean las diferencias resaltadas. Cuando *context* es `False`, *numlines* controla el número de líneas que se muestran antes de una diferencia resaltada cuando se usan los hipervínculos «next» (un valor de cero produce que los hipervínculos «next» ubiquen el siguiente resaltado en la parte superior del navegador, sin ningún contexto principal).

Nota

fromdesc y *todesc* se interpretan como HTML no escapado y se deben escapar correctamente si los datos son recibidos de fuentes no confiables.

Distinto en la versión 3.5: Se agregó el argumento sólo de palabra clave *charset*. La codificación de caracteres por defecto para documentos HTML se cambió de `'ISO-8859-1'` a `'utf-8'`.

make_table(*fromlines, tolines, fromdesc=", todesc=", context=False, numlines=5*)

Compara *fromlines* y *tolines* (listas de cadenas de texto) y retorna una cadena de caracteres que representa una tabla HTML mostrando comparaciones lado a lado y línea por línea del texto con cambios interlineales e intralineales.

Los argumentos para este método son los mismos que los del método **`make_file()`**.

difflib.context_diff(*a*, *b*, *fromfile=''*, *tofile=''*, *fromfiledate=''*, *tofiledate=''*, *n=3*, *lineterm='\n'*)

Compara *a* y *b* (listas de cadenas de texto); retorna un delta (un generator que genera las lineas delta) en formato de diferencias de contexto.

El formato de diferencias de contexto es una forma compacta de mostrar solamente las líneas que fueron modificadas y algunas líneas adicionales de contexto. Los cambios son mostrados utilizando el estilo antes/después. El número de líneas de contexto es determinado por *n*, cuyo valor por defecto es `3`.

Por defecto, las líneas de control (aquellas que comienzan con `***` o `---`) son creadas con una línea nueva. Esto es de ayuda para que las entradas creadas por **`io.IOBase.readlines()`** generen diferencias que puedan ser utilizadas con **`io.IOBase.writelines()`** ya que ambas, la entrada y la salida, tienen líneas nuevas al final.

Para entradas que no tienen nuevas líneas finales, establezca el argumento *lineterm* como `""` de forma que la salida sea uniforme y libre de nuevas líneas.

El formato de diferencias de contexto normalmente tiene un encabezado para nombres de archivos y tiempos de modificaciones. Alguno o todos estos debe ser especificado utilizando las cadenas de texto para *fromfile*, *tofile*, *fromfiledate* y *tofiledate*. Los tiempos de modificación son normalmente expresados en formato ISO 8601. Si no es especificado las cadenas por defecto son espacios en blanco.

```
>>> import sys
>>> from difflib import *
>>> s1 = ['bacon\n', 'eggs\n', 'ham\n', 'guido\n']
>>> s2 = ['python\n', 'eggy\n', 'hamster\n', 'guido
>>> sys.stdout.writelines(context_diff(s1, s2, from
...                        tofile='after.py'))
*** before.py
--- after.py
***************
*** 1,4 ****
! bacon
! eggs
! ham
  guido
--- 1,4 ----
! python
! eggy
! hamster
  guido
```

Vea Una interfaz de línea de comandos para difflib para un ejemplo mas detallado.

difflib.get_close_matches(*word*, *possibilities*, *n=3*, *cutoff=0.6*)

Retorna una lista de las mejores coincidencias «lo suficientemente buenas». *word* es una secuencia para la cual coincidencias cercanas son deseadas (usualmente una cadena de texto), y *possibilities* es una lista de secuencias contra la cual se compara *word* (comúnmente una lista de cadenas de caracteres).

Argumento opcional *n* (por defecto `3`) es el máximo número de coincidencias cercanas a retornar; *n* debe ser mayor que `0`.

Argumento opcional *cutoff* (por defecto `0.6`) es un flotante en el rango [0, 1]. Las posibilidades que no alcanzan un puntaje al menos similar al de *word* son ignoradas.

Las mejores (no mas de *n*) coincidencias entre las posibilidades son retornadas en una lista, ordenadas por similitud de puntaje, las mas similares primero.

```
>>> get_close_matches('appel', ['ape', 'apple', 'pe
['apple', 'ape']
>>> import keyword
>>> get_close_matches('wheel', keyword.kwlist)
['while']
>>> get_close_matches('pineapple', keyword.kwlist)
[]
>>> get_close_matches('accept', keyword.kwlist)
['except']
```

difflib.ndiff(*a*, *b*, *linejunk=None*, *charjunk=IS_CHARACTER_JUNK*)

Compara *a* y *b* (listas de cadenas de texto); retorna un delta del estilo **`Differ`** (un generator que genera los deltas de las líneas).

Parámetros de palabra clave opcional *linejunk* y *charjunk* son funciones de filtrado (o `None`):

linejunk: A function that accepts a single string argument, and returns true if the string is junk, or false if not. The default is `None`. There is also a module-level function **`IS_LINE_JUNK()`**, which filters out lines without visible characters, except for at most one hash character (`'#'`) – however the underlying **`SequenceMatcher`** class does a dynamic analysis of which lines are so frequent as to constitute noise, and this usually works better than using this function.

charjunk: Una función que acepta un carácter (una cadena de caracteres de longitud 1) como argumento, y retorna *True* si el carácter es un elemento no deseado, o *False* si no lo es. El valor por defecto es una función a nivel del módulo **`IS_CHARACTER_JUNK()`**, que filtra caracteres de espacios en blanco (un espacio en blanco o tabulación; es una mala idea incluir saltos de lineas en esto!)

```
>>> diff = ndiff('one\ntwo\nthree\n'.splitlines(kee
...              'ore\ntree\nemu\n'.splitlines(keep
>>> print(''.join(diff), end="")
- one
```

```
?  ^
+ ore
?  ^
- two
- three
?  -
+ tree
+ emu
```

difflib.restore(*sequence*, *which*)

Retorna uno de las dos secuencias que generaron un delta.

Dada una *sequence* producida por `Differ.compare()` o `ndiff()`, extrae las líneas originadas por el archivo 1 o 2 (parámetro *which*), quitando los prefijos de la línea.

Ejemplo:

```
>>> diff = ndiff('one\ntwo\nthree\n'.splitlines(kee
...              'ore\ntree\nemu\n'.splitlines(keep
>>> diff = list(diff) # materialize the generated d
>>> print(''.join(restore(diff, 1)), end="")
one
two
three
>>> print(''.join(restore(diff, 2)), end="")
ore
tree
emu
```

difflib.unified_diff(*a*, *b*, *fromfile=''*, *tofile=''*, *fromfiledate=''*, *tofiledate=''*, *n=3*, *lineterm='\n'*)

Compara *a* y *b* (listas de cadenas de caracteres); retorna un delta (un generator que genera los delta de líneas) en formato de diferencias unificado.

Las diferencias unificadas son una forma compacta de mostrar sólo las líneas que presentan cambios y algunas líneas de contexto adicionales. Los cambios son mostrados en una sola línea (en lugar de bloques separados antes y después). El número de líneas de contexto se establece mediante *n*, cuyo valor por defecto es tres.

Por defecto, las líneas de control de diferencias (aquellas con `---`, `+++`, o `@@`) son creadas con un salto de línea nuevo. Esto es de ayuda para que las entradas creadas por **`io.IOBase.readlines()`** generen diferencias que puedan ser utilizadas con **`io.IOBase.writelines()`** ya que ambas, la entrada y la salida, tienen líneas nuevas al final.

Para entradas que no tienen nuevas líneas finales, establezca el argumento *lineterm* como `""` de forma que la salida sea uniforme y libre de nuevas líneas.

The unified diff format normally has a header for filenames and modification times. Any or all of these may be specified using strings for *fromfile*, *tofile*, *fromfiledate*, and *tofiledate*. The modification times are normally expressed in the ISO 8601 format. If not specified, the strings default to blanks.

```
>>> s1 = ['bacon\n', 'eggs\n', 'ham\n', 'guido\n']
>>> s2 = ['python\n', 'eggy\n', 'hamster\n', 'guido
>>> sys.stdout.writelines(unified_diff(s1, s2, from
--- before.py
+++ after.py
@@ -1,4 +1,4 @@
-bacon
-eggs
-ham
+python
+eggy
+hamster
 guido
```

Vea Una interfaz de línea de comandos para difflib para un ejemplo mas detallado.

difflib.diff_bytes(*dfunc*, *a*, *b*, *fromfile=b''*, *tofile=b''*, *fromfiledate=b''*, *tofiledate=b''*, *n=3*, *lineterm=b'\n'*)

Compara *a* y *b* (listas de objetos de bytes) usando *dfunc*; produce una secuencia de líneas delta (también bytes) en el formato retornado por *dfunc*. *dfunc* debe ser invocable, comúnmente cualquiera de `unified_diff()` o `context_diff()`.

Permite comparar datos con codificación desconocida o inconsistente. Todas las entradas, excepto *n*, deben ser objetos de bytes, no cadenas de texto. Funciona convirtiendo sin pérdidas todas las entradas (excepto *n*) a cadenas de texto, e invoca `dfunc(a, b, fromfile, tofile, fromfiledate, tofiledate, n, lineterm)`. La salida de *dfunc* es entonces convertida nuevamente a bytes, de forma que las líneas delta que son recibidas tienen la misma codificación desconocida/inconsistente que *a* y *b*.

Added in version 3.5.

difflib.IS_LINE_JUNK(*line*)

Retorna `True` para líneas que deben ser ignoradas. La línea *line* es ignorada si *line* es un espacio vacío o contiene un solo `'#'`, en cualquier otro caso no es ignorado. Es usado como valor por defecto para el parámetro *linejunk* por `ndiff()` en versiones anteriores.

difflib.IS_CHARACTER_JUNK(*ch*)

Retorna `True` para los caracteres que deben ser ignorados. El carácter *ch* es ignorado si *ch* es un espacio en blanco o tabulación, en cualquier otro caso no es ignorado. Es utilizado como valor por defecto para el parámetro *charjunk* en `ndiff()`.

Ver también

Pattern Matching: The Gestalt Approach [https://jacobfilipp.com/DrDobbs/articles/DDJ/1988/8807/8807c/8807c.htm]

Discussion of a similar algorithm by John W. Ratcliff and D. E. Metzener. This was published in Dr. Dobb’s Journal in July, 1988.

SequenceMatcher objects

La clase **`SequenceMatcher`** tiene este constructor:

class difflib.SequenceMatcher(*isjunk=None*, *a=''*, *b=''*, *autojunk=True*)

El argumento opcional *isjunk* debe ser `None` (que es su valor por defecto) o una función de un solo argumento que reciba un elemento de la secuencia y retorne verdadero si y solo si el elemento es no deseado y deba ser ignorado. Pasar el argumento *isjunk* como `None` es equivalente a pasar `lambda x: False`; en otras palabras, ningún elemento es ignorado. Por ejemplo, pasar:

```
lambda x: x in " \t"
```

si se están comparando líneas como secuencias de caracteres, y no se quiere sincronizar en espacios blancos o tabulaciones.

Los argumentos opcionales *a* y *b* son las secuencias a comparar; ambos tienen como valor por defecto una cadena de texto vacía. Los elementos de ambas secuencias deben ser hashable.

El argumento opcional *autojunk* puede ser usado para deshabilitar la heurística automática de elementos no deseados.

Distinto en la versión 3.2: Added the *autojunk* parameter.

Los objetos *SequenceMatcher* reciben tres atributos: *bjunk* es el conjunto de elementos de *b* para los cuales *isjunk* es `True`; *bpopular* es el set de elementos que no son basura considerados populares por la heurística (si no es que fue deshabilitada); *b2j* es un diccionario que mapea elementos de *b* a una lista de posiciones donde estos ocurren. Los tres atributos son reseteados cuando *b* es reseteado mediante **`set_seqs()`** o **`set_seq2()`**.

Added in version 3.2: Los atributos *bjunk* y *bpopular*.

Los objetos **`SequenceMatcher`** tienen los siguientes métodos:

set_seqs(*a*, *b*)

> Establece las dos secuencias a ser comparadas.

`SequenceMatcher` calcula y almacena información detallada sobre la segunda secuencia, con lo cual si quieres comparar una secuencia contra muchas otras, usa **`set_seq2()`** para establecer la secuencia común una sola vez y llamar **`set_seq1()`** repetidamente, una vez por cada una de las otras secuencias.

set_seq1(*a*)

> Establece la primer secuencia a ser comparada. La segunda secuencia a ser comparada no es modificada.

set_seq2(*b*)

> Establece la segunda secuencia a ser comparada. La primera secuencia a ser comparada no es modificada.

find_longest_match(*alo=0*, *ahi=None*, *blo=0*, *bhi=None*)

> Encuentra el bloque de coincidencia mas largo en `a[alo:ahi]` y `b[blo:bhi]`.
>
> Si *isjunk* fue omitido o es `None`, **`find_longest_match()`** retorna `(i, j, k)` tal que `a[i:i+k]` es igual a `b[j:j+k]`, donde `alo <= i <= i+k <= ahi` y `blo <= j <= j+k <= bhi`. Para todo `(i', j', k')` cumpliendo esas condiciones, las condiciones adicionales `k >= k'`, `i <= i'`, y si `i == i'`, `j <= j'` también se cumplen. En otras palabras, de todos los bloques coincidentes máximos, retorna aquel que comienza antes en *a*, y de todos esos bloques coincidentes máximos que comienzan antes en *a*, retorna aquel que comienza antes en *b*.

```
>>> s = SequenceMatcher(None, " abcd", "abcd ab
>>> s.find_longest_match(0, 5, 0, 9)
Match(a=0, b=4, size=5)
```

Si se porporcionó *isjunk*, primero se determina el bloque coincidente mas largo como fue indicado anteriormente, pero con la restricción adicional de que no aparezca ningún elemento no deseado en el bloque. Entonces, ese bloque se extiende tan lejos como sea posible haciendo coincidir (solo) elementos no deseados de ambos lados. Por lo tanto, el bloque resultante nunca hará coincidir ningún elemento no deseado, excepto que un elemento no deseado idéntico pase a ser adyacente a una coincidencia interesante.

Este es el mismo ejemplo que el mostrado anteriormente, pero considerando elementos en blanco como no deseados. Esto previene que `' abcd'` sea coincidente con `'abcd'` en el final de la segunda secuencia directamente. En cambio, sólo el `'abcd'` puede coincidir, y coincide con el `'abcd'` que se encuentre mas a la izquierda en la segunda secuencia:

```
>>> s = SequenceMatcher(lambda x: x==" ", " abc
>>> s.find_longest_match(0, 5, 0, 9)
Match(a=1, b=0, size=4)
```

Si no coincide ningún bloque, esto retorna `(alo, blo, 0)`.

Este método retorna un named tuple `Match(a, b, size)`.

Distinto en la versión 3.9: Se agregaron argumentos predeterminados.

get_matching_blocks()

Retorna una lista de triplas (tuplas de tres elementos) describiendo subsecuencias coincidentes no superpuestas. Cada tripla sigue el formato `(i, j, n)`, y significa que `a[i:i+n] == b[j:j+n]`. Las triplas son monótonamente crecientes en *i* y *j*.

La última tripla es un objeto ficticio (dummy), y tiene el valor `(len(a), len(b), 0)`. Es la única tripla con `n == 0`. Si `(i, j, n)` y `(i', j', n')` son triplas adyacentes en la lista, y la segunda no es el último elemento de la lista, entonces `i+n < i'` o `j+n < j'`; en otras palabras, las triplas adyacentes describen bloques iguales no adyacentes.

```
>>> s = SequenceMatcher(None, "abxcd", "abcd")
>>> s.get_matching_blocks()
[Match(a=0, b=0, size=2), Match(a=3, b=2, size=
```

get_opcodes()

Retorna una lista de quíntuplas (tuplas de cinco elementos) describiendo como convertir *a* en *b*. Cada tupla tiene la forma `(tag, i1, i2, j1, j2)`. En la primer tupla se cumple que `i1 == j1 == 0`, y las tuplas restantes tienen *i1* igual al *i2* de la tupla precedente, y de igual manera, *j1* igual al *j2* de la tupla anterior.

Los valores de *tag* son cadenas de caracteres, con el siguiente significado:

Valor	**Significado**
`'replace'`	`a[i1:i2]` debe ser reemplazado por `b[j1:j2]`.
`'delete'`	`a[i1:i2]` debe ser eliminado. Nótese que en este caso `j1 == j2`.
`'insert'`	`b[j1:j2]` debe ser insertado en `a[i1:i1]`. Nótese que en este caso `i1 == i2`.
`'equal'`	`a[i1:i2] == b[j1:j2]` (las subsecuencias son iguales).

Por ejemplo:

```
>>> a = "qabxcd"
>>> b = "abycdf"
>>> s = SequenceMatcher(None, a, b)
>>> for tag, i1, i2, j1, j2 in s.get_opcodes():
...     print('{:7}   a[{}:{}] --> b[{}:{}] {!
...         tag, i1, i2, j1, j2, a[i1:i2], b[j1
delete    a[0:1] --> b[0:0]      'q' --> ''
equal     a[1:3] --> b[0:2]     'ab' --> 'ab'
replace   a[3:4] --> b[2:3]      'x' --> 'y'
equal     a[4:6] --> b[3:5]     'cd' --> 'cd'
insert    a[6:6] --> b[5:6]       '' --> 'f'
```

get_grouped_opcodes(*n*=*3*)

Retorna un generator de grupos de hasta *n* líneas de contexto.

Empezando con los grupos retornados por **`get_opcodes()`**, este método separa grupos con cambios menores y elimina los rangos intermedios que no tienen cambios.

Los grupos son retornados en el mismo formato que **`get_opcodes()`**.

ratio()

Retorna una medida de la similitud de las secuencias como un flotante en el rango [0, 1].

Donde T es el número total de elementos en ambas secuencias y M es el número de coincidencias, esto es 2.0*M / T. Nótese que esto es `1.0` si las secuencias son idénticas y `0.0` si no tienen nada en común.

Esto es computacionalmente costoso si **`get_matching_blocks()`** o **`get_opcodes()`** no fueron ejecutados, in tal caso deberías considerar primero **`quick_ratio()`** o **`real_quick_ratio()`** para obtener un límite superior.

Nota

Precaución: El resultado de una llamada a **`ratio()`** puede depender del orden de los argumentos. Por ejemplo:

```
>>> SequenceMatcher(None, 'tide', 'diet').rati
0.25
>>> SequenceMatcher(None, 'diet', 'tide').rati
0.5
```

quick_ratio()

Retorna un límite superior en **`ratio()`** relativamente rápido.

real_quick_ratio()

Retorna un límite superior en **`ratio()`** muy rápido.

The three methods that return the ratio of matching to total characters can give different results due to differing levels of approximation, although **`quick_ratio()`** and **`real_quick_ratio()`** are always at least as large as **`ratio()`**:

```
>>> s = SequenceMatcher(None, "abcd", "bcde")
>>> s.ratio()
0.75
>>> s.quick_ratio()
0.75
>>> s.real_quick_ratio()
1.0
```

SequenceMatcher examples

Este ejemplo compara dos cadenas de texto, considerando los espacios en blanco como caracteres no deseados:

```
>>> s = SequenceMatcher(lambda x: x == " ",
...                     "private Thread currentThread;",
...                     "private volatile Thread current
```

`ratio()` returns a float in [0, 1], measuring the similarity of the sequences. As a rule of thumb, a **`ratio()`** value over 0.6 means the sequences are close matches:

```
>>> print(round(s.ratio(), 3))
0.866
```

If you're only interested in where the sequences match, `get_matching_blocks()` is handy:

```
>>> for block in s.get_matching_blocks():
...     print("a[%d] and b[%d] match for %d elements" %
a[0] and b[0] match for 8 elements
a[8] and b[17] match for 21 elements
a[29] and b[38] match for 0 elements
```

Note that the last tuple returned by `get_matching_blocks()` is always a dummy, `(len(a), len(b), 0)`, and this is the only case in which the last tuple element (number of elements matched) is `0`.

If you want to know how to change the first sequence into the second, use `get_opcodes()`:

```
>>> for opcode in s.get_opcodes():
...     print("%6s a[%d:%d] b[%d:%d]" % opcode)
 equal a[0:8] b[0:8]
insert a[8:8] b[8:17]
 equal a[8:29] b[17:38]
```

Ver también

- La función `get_close_matches()` en este módulo que muestra lo simple que es el código que construye `SequenceMatcher` puede ser utilizada para hacer un trabajo útil.
- Simple version control recipe [https://code.activestate.com/recipes/576729-simple-version-control/] for a small application built with `SequenceMatcher`.

Differ objects

Nótese que los deltas generados por `Differ` no dicen ser diferencias **mínimas**. Todo lo contrario, las diferencias mínimas suelen ser contra-intuitivas, ya que se sincronizan en cualquier lugar posible, a veces coinciden accidentalmente con 100 páginas de diferencia. Restringiendo los puntos de sincronización a coincidencias contiguas se preserva cierta noción de cercanía, con el costo adicional de producir diferencias mas largas.

La clase `Differ` tiene el siguiente constructor:

class difflib.Differ(*linejunk=None*, *charjunk=None*)

> Parámetros de palabra clave opcionales *linejunk* y *charjunk* son para funciones de filtrado (o `None`):
>
> *linejunk*: Una función que acepta una sola cadena de texto como argumento y retorna verdadero si la cadena de texto es un elemento no deseado. Su valor por defecto es `None`, lo que significa que ninguna línea es considerada no deseada.
>
> *charjunk*: Una función que acepta un solo carácter como argumento (una cadena de caracteres de longitud 1) y retorna verdadero si el carácter es un elemento no deseado. Su valor por defecto es `None`, lo que significa que ningún carácter es considerado no deseado.
>
> Estas funciones de elementos no deseados aceleran la coincidencia para encontrar diferencies y no hacen que se ignoren líneas o caracteres diferentes. Lea la descripción del parámetro *isjunk* en el método `find_longest_match()` para una explicación mas detallada.
>
> Los objetos `Differ` son usados (una vez generados los deltas) mediante un solo método:
>
> compare(*a*, *b*)
>
> > Compara dos secuencias de líneas y genera el delta correspondiente (una secuencia de líneas).

Cada secuencia debe contener cadenas de texto individuales de una sola linea terminadas con una línea nueva. Este tipo de secuencias pueden ser obtenidas mediante el método **`readlines()`** de objetos de tipo archivo. Los delta generados consisten también en cadenas de texto terminadas en nuevas lineas, listas para imprimirse tal cual a través del método **`writelines()`** de un objeto de tipo archivo.

Differ example

This example compares two texts. First we set up the texts, sequences of individual single-line strings ending with newlines (such sequences can also be obtained from the **`readlines()`** method of file-like objects):

```
>>> text1 = '''  1. Beautiful is better than ugly.
...   2. Explicit is better than implicit.
...   3. Simple is better than complex.
...   4. Complex is better than complicated.
... '''.splitlines(keepends=True)
>>> len(text1)
4
>>> text1[0][-1]
'\n'
>>> text2 = '''  1. Beautiful is better than ugly.
...   3.   Simple is better than complex.
...   4. Complicated is better than complex.
...   5. Flat is better than nested.
... '''.splitlines(keepends=True)
```

Luego instanciamos el objeto *Differ*:

```
>>> d = Differ()
```

Nótese que cuando instanciamos un objeto **`Differ`** deberíamos pasar funciones para filtrar lineas y caracteres no deseados. Consulte el constructor de **`Differ()`** para mas detalles.

Finalmente, comparamos las dos:

```
>>> result = list(d.compare(text1, text2))
```

`result` es una lista de cadenas de caracteres, entonces vamos a mostrarlo de una forma elegante:

```
>>> from pprint import pprint
>>> pprint(result)
['    1. Beautiful is better than ugly.\n',
 '-   2. Explicit is better than implicit.\n',
 '-   3. Simple is better than complex.\n',
 '+   3.   Simple is better than complex.\n',
 '?     ++\n',
 '-   4. Complex is better than complicated.\n',
 '?            ^                     ---- ^\n',
 '+   4. Complicated is better than complex.\n',
 '?           ++++ ^                      ^\n',
 '+   5. Flat is better than nested.\n']
```

Representado como una sola cadena de caracteres de múltiples líneas se ve así:

```
>>> import sys
>>> sys.stdout.writelines(result)
    1. Beautiful is better than ugly.
-   2. Explicit is better than implicit.
-   3. Simple is better than complex.
+   3.   Simple is better than complex.
?     ++
-   4. Complex is better than complicated.
?            ^                     ---- ^
+   4. Complicated is better than complex.
?           ++++ ^                      ^
+   5. Flat is better than nested.
```

Una interfaz de línea de comandos para `difflib`

This example shows how to use difflib to create a `diff`-like utility.

```
""" Command-line interface to difflib.py providing diffs

* ndiff:    lists every line and highlights interline ch
* context:  highlights clusters of changes in a before/a
* unified:  highlights clusters of changes in an inline 
* html:     generates side by side comparison with chang

"""

import sys, os, difflib, argparse
import datetime as dt

def file_mtime(path):
    t = dt.datetime.fromtimestamp(os.stat(path).st_mtime
                                  dt.timezone.utc)
    return t.astimezone().isoformat()

def main():

    parser = argparse.ArgumentParser()
    parser.add_argument('-c', action='store_true', defau
                        help='Produce a context format d
    parser.add_argument('-u', action='store_true', defau
                        help='Produce a unified format d
    parser.add_argument('-m', action='store_true', defau
                        help='Produce HTML side by side 
                             '(can use -c and -l in conj
    parser.add_argument('-n', action='store_true', defau
                        help='Produce a ndiff format dif
    parser.add_argument('-l', '--lines', type=int, defau
                        help='Set number of context line
    parser.add_argument('fromfile')
    parser.add_argument('tofile')
    options = parser.parse_args()

    n = options.lines
    fromfile = options.fromfile
    tofile = options.tofile

    fromdate = file_mtime(fromfile)
    todate = file_mtime(tofile)
    with open(fromfile) as ff:
        fromlines = ff.readlines()
    with open(tofile) as tf:
        tolines = tf.readlines()

    if options.u:
        diff = difflib.unified_diff(fromlines, tolines,
    elif options.n:
        diff = difflib.ndiff(fromlines, tolines)
    elif options.m:
        diff = difflib.HtmlDiff().make_file(fromlines,to
    else:
        diff = difflib.context_diff(fromlines, tolines,

    sys.stdout.writelines(diff)

if __name__ == '__main__':
    main()
```

ndiff example

This example shows how to use **`difflib.ndiff()`**.

```
"""ndiff [-q] file1 file2
    or
ndiff (-r1 | -r2) < ndiff_output > file1_or_file2

Print a human-friendly file difference report to stdout.
and intra-line differences are noted.  In the second for
(-r1) or file2 (-r2) on stdout, from an ndiff report on

In the first form, if -q ("quiet") is not specified, the
of output are
```

```
-: file1
+: file2

Each remaining line begins with a two-letter code:

    "- "    line unique to file1
    "+ "    line unique to file2
    "  "    line common to both files
    "? "    line not present in either input file

Lines beginning with "? " attempt to guide the eye to i
differences, and were not present in either input file.
confusing if the source files contain tab characters.

The first file can be recovered by retaining only lines
"  " or "- ", and deleting those 2-character prefixes; u

The second file can be recovered similarly, but by retai
"+ " lines; use ndiff with -r2; or, on Unix, the second
recovered by piping the output through

    sed -n '/^[+ ] /s/^..//p'
"""

__version__ = 1, 7, 0

import difflib, sys

def fail(msg):
    out = sys.stderr.write
    out(msg + "\n\n")
    out(__doc__)
    return 0

# open a file & return the file object; gripe and retur
# couldn't be opened
def fopen(fname):
    try:
        return open(fname)
    except IOError as detail:
        return fail("couldn't open " + fname + ": " + st

# open two files & spray the diff to stdout; return fals
def fcompare(f1name, f2name):
    f1 = fopen(f1name)
    f2 = fopen(f2name)
    if not f1 or not f2:
        return 0

    a = f1.readlines(); f1.close()
    b = f2.readlines(); f2.close()
    for line in difflib.ndiff(a, b):
        print(line, end=' ')

    return 1

# crack args (sys.argv[1:] is normal) & compare;
# return false iff a problem

def main(args):
    import getopt
    try:
        opts, args = getopt.getopt(args, "qr:")
    except getopt.error as detail:
        return fail(str(detail))
    noisy = 1
    qseen = rseen = 0
    for opt, val in opts:
        if opt == "-q":
            qseen = 1
            noisy = 0
        elif opt == "-r":
            rseen = 1
            whichfile = val
    if qseen and rseen:
        return fail("can't specify both -q and -r")
    if rseen:
        if args:
            return fail("no args allowed with -r option"
        if whichfile in ("1", "2"):
            restore(whichfile)
            return 1
        return fail("-r value must be 1 or 2")
    if len(args) != 2:
        return fail("need 2 filename args")
    f1name, f2name = args
    if noisy:
        print('-:', f1name)
        print('+:', f2name)
    return fcompare(f1name, f2name)

# read ndiff output from stdin, and print file1 (which==
# file2 (which=='2') to stdout

def restore(which):
    restored = difflib.restore(sys.stdin.readlines(), wh
    sys.stdout.writelines(restored)

if __name__ == '__main__':
    main(sys.argv[1:])
```

`textwrap` — Text wrapping and filling

Source code: Lib/textwrap.py [https://github.com/python/cpython/tree/3.14/Lib/textwrap.py]

The `textwrap` module provides some convenience functions, as well as `TextWrapper`, the class that does all the work. If you're just wrapping or filling one or two text strings, the convenience functions should be good enough; otherwise, you should use an instance of `TextWrapper` for efficiency.

textwrap.wrap(*text*, *width=70*, ***, *initial_indent=''*, *subsequent_indent=''*, *expand_tabs=True*, *replace_whitespace=True*, *fix_sentence_endings=False*, *break_long_words=True*, *drop_whitespace=True*, *break_on_hyphens=True*, *tabsize=8*, *max_lines=None*, *placeholder=' [...]'*)

> Envuelve el párrafo individual en *text* (una cadena) para que cada línea tenga como máximo *width* de caracteres de largo. Retorna una lista de líneas de salida, sin las nuevas líneas finales.
>
> Los argumentos de palabras clave opcionales corresponden a los atributos de instancia de `TextWrapper`, que se documentan a continuación.
>
> Ver el método `TextWrapper.wrap()` para más detalles sobre el comportamiento de `wrap()`.

textwrap.fill(*text*, *width=70*, ***, *initial_indent=''*, *subsequent_indent=''*, *expand_tabs=True*, *replace_whitespace=True*, *fix_sentence_endings=False*, *break_long_words=True*, *drop_whitespace=True*, *break_on_hyphens=True*, *tabsize=8*, *max_lines=None*, *placeholder=' [...]'*)

Envuelve el único párrafo en *text*, y retorna una sola cadena que contiene el párrafo envuelto. `fill()` es la abreviatura de

```
"\n".join(wrap(text, ...))
```

En particular, `fill()` acepta exactamente los mismos argumentos de palabras clave que `wrap()`.

textwrap.shorten(*text*, *width*, ***, *fix_sentence_endings=False*, *break_long_words=True*, *break_on_hyphens=True*, *placeholder=' [...]'*)

Colapsa y trunca el *text* dado para que encaje en el *width* dado.

Primero el espacio blanco en *text* se colapsa (todos los espacios blancos son reemplazados por espacios sencillos). Si el resultado cabe en el *width*, se retorna. En caso contrario, se eliminan suficientes palabras del final para que las palabras restantes más el *placeholder* encajen dentro de *width*:

```
>>> textwrap.shorten("Hello  world!", width=12)
'Hello world!'
>>> textwrap.shorten("Hello  world!", width=11)
'Hello [...]'
>>> textwrap.shorten("Hello world", width=10, place
'Hello...'
```

Los argumentos opcionales de las palabras clave corresponden a los atributos de la instancia de `TextWrapper`, documentados a continuación. Observe que el espacio en blanco se colapsa antes de pasar el texto a la función **`TextWrapper`** `fill()`, por lo que cambiar el valor de `tabsize`, `expand_tabs`, `drop_whitespace`, y `replace_whitespace` no tendrá ningún efecto.

Added in version 3.4.

textwrap.dedent(*text*)

Elimina cualquier espacio en blanco común de cada línea de *text*.

Esto puede utilizarse para hacer que las cadenas con comillas triples se alineen con el borde izquierdo de la pantalla, mientras que se siguen presentando en el código fuente en forma indentada.

Nótese que los tabuladores y los espacios se tratan como espacios en blanco, pero no son iguales: las líneas `"  hello"` y `"\thello"` se consideran que no tienen un espacio en blanco común.

Las líneas que sólo contienen espacios en blanco se ignoran en la entrada y se normalizan a un solo carácter de nueva línea en la salida.

Por ejemplo:

```
def test():
    # end first line with \ to avoid the empty line
    s = '''\
    hello
      world
    '''
    print(repr(s))          # prints '    hello\n
    print(repr(dedent(s)))  # prints 'hello\n  worl
```

Distinto en la versión 3.14: The **`dedent()`** function now correctly normalizes blank lines containing only whitespace characters. Previously, the implementation only normalized blank lines containing tabs and spaces.

textwrap.indent(*text*, *prefix*, *predicate=None*)

Añade *prefix* al principio de las líneas seleccionadas en *text*.

Las líneas se separan llamando a `text.splitlines(True)`.

Por defecto, se añade *prefix* a todas las líneas que no consisten únicamente en espacios en blanco (incluyendo cualquier terminación de línea).

Por ejemplo:

```
>>> s = 'hello\n\n \nworld'
>>> indent(s, '  ')
'  hello\n\n \n  world'
```

El argumento opcional *predicate* puede ser usado para controlar qué líneas están indentadas. Por ejemplo, es fácil añadir *prefix* incluso a las líneas vacías y de espacio en blanco:

```
>>> print(indent(s, '+ ', lambda line: True))
+ hello
+
+
+ world
```

Added in version 3.3.

`wrap()`, `fill()` y `shorten()` funcionan creando una instancia `TextWrapper` y llamando a un solo método en ella. Esa instancia no se reutiliza, por lo que para las aplicaciones que procesan muchas cadenas de texto usando **`wrap()`** y/o **`fill()`**, puede ser más eficiente crear su propio objeto **`TextWrapper`**.

El texto se envuelve preferentemente en espacios en blanco y justo después de los guiones en palabras con guión; sólo entonces se romperán las palabras largas si es necesario, a menos que `TextWrapper.break_long_words` sea falso.

class textwrap.TextWrapper(***kwargs*)

El constructor `TextWrapper` acepta un número de argumentos de palabras clave opcionales. Cada argumento de palabra clave corresponde a un atributo de la instancia, por ejemplo

```
wrapper = TextWrapper(initial_indent="* ")
```

es lo mismo que

```
wrapper = TextWrapper()
wrapper.initial_indent = "* "
```

You can reuse the same **`TextWrapper`** object many times, and you can change any of its options through direct assignment to instance attributes between uses.

Los atributos de la instancia **`TextWrapper`** (y los argumentos de las palabras clave para el constructor) son los siguientes:

width

> (default: `70`) La longitud máxima de las líneas envueltas. Mientras no haya palabras individuales en el texto de entrada más largas que **`width`**, **`TextWrapper`** garantiza que ninguna línea de salida será más larga que los caracteres **`width`**.

expand_tabs

> (default: `True`) Si es verdadero, entonces todos los caracteres de tabulación en *text* serán expandidos a espacios usando el método **`expandtabs()`** de *text*.

tabsize

> (default: `8`) Si **`expand_tabs`** es verdadero, entonces todos los caracteres de tabulación en *text* se expandirán a cero o más espacios, dependiendo de la columna actual y el tamaño de tabulación dado.
>
> *Added in version 3.3.*

replace_whitespace

> (default: `True`) Si es verdadero, después de la expansión por tabulador pero antes de envolver, el método **`wrap()`** reemplazará cada carácter de espacio en blanco con un espacio sencillo. Los caracteres de los espacios en blanco reemplazados son los siguientes: tab, nueva línea, tab vertical, *formfeed* y *carriage return* (`'\t\n\v\f\r'`).

Nota

Si `expand_tabs` es falso y `replace_whitespace` es verdadero, cada carácter del tabulador será reemplazado por un solo espacio, que *no* es lo mismo que la expansión del tabulador.

Nota

Si `replace_whitespace` es falso, las nuevas líneas pueden aparecer en medio de una línea y causar una salida extraña. Por esta razón, el texto debe ser dividido en párrafos (usando `str.splitlines()` o similar) que se envuelven por separado.

drop_whitespace

(default: `True`) Si es verdadero, se eliminan los espacios en blanco al principio y al final de cada línea (después de la envoltura pero antes del indentado). Sin embargo, el espacio en blanco al principio del párrafo no se elimina si lo sigue un espacio en blanco. Si el espacio blanco que se deja caer ocupa una línea entera, se deja caer toda la línea.

initial_indent

(default: `''`) Cadena que será preparada para la primera línea de salida envuelta. Cuenta hacia la longitud de la primera línea. La cadena vacía no está indentada.

subsequent_indent

(default: `''`) Cadena que se preparará para todas las líneas de salida envueltas excepto la primera. Cuenta hacia la longitud de cada línea excepto la primera.

fix_sentence_endings

(default: `False`) If true, **`TextWrapper`** attempts to detect sentence endings and ensure that sentences are always separated by exactly two spaces. This is generally desired for text in a monospaced font. However, the sentence detection algorithm is imperfect: it assumes that a sentence ending consists of a lowercase letter followed by one of `'.'`, `'!'`, or `'?'`, possibly followed by one of `'"'` or `"'"`, followed by a space. One problem with this algorithm is that it is unable to detect the difference between «Dr.» in

```
[...] Dr. Frankenstein's monster [...]
```

y «Spot.» en:

```
[...] See Spot. See Spot run [...]
```

`fix_sentence_endings` es falso por defecto.

Dado que el algoritmo de detección de oraciones se basa en `string.lowercase` para la definición de «letra en minúscula», y en la convención de utilizar dos espacios después de un punto para separar las oraciones en la misma línea, es específico para los textos en inglés.

break_long_words

(default: `True`) Si es verdadero, entonces las palabras más largas que **`width`** se romperán para asegurar que ninguna línea sea más larga que **`width`**. Si es falso, las palabras largas no se romperán, y algunas líneas pueden ser más largas que **`width`**. (Las palabras largas se pondrán en una línea por sí mismas, para minimizar la cantidad en que se excede **`width`**).

break_on_hyphens

(predeterminado: `True`) Si es verdadero, el ajuste se producirá preferiblemente en espacios en blanco y justo después de los guiones en palabras compuestas, como es habitual en inglés. Si es falso, solo los espacios en blanco se considerarán lugares potencialmente buenos para los saltos de línea, pero debe establecer **`break_long_words`** en falso si desea palabras verdaderamente insecables. El comportamiento predeterminado en versiones anteriores era permitir siempre romper palabras con guiones.

max_lines

(default: `None`) Si es `None`, entonces la salida contendrá como máximo *max_lines*, con un *placeholder* que aparecerá al final de la salida.

Added in version 3.4.

placeholder

(default: `' [...]'`) Cadena que aparecerá al final del texto de salida si ha sido truncado.

Added in version 3.4.

`TextWrapper` también proporciona algunos métodos públicos, análogos a las funciones de conveniencia a nivel de módulo:

wrap(*text*)

Envuelve el párrafo individual en *text* (una cadena) para que cada línea tenga como máximo **`width`** caracteres de largo. Todas las opciones de envoltura se toman de los atributos de la instancia **`TextWrapper`**. Retorna una lista de líneas de salida, sin las nuevas líneas finales. Si la salida envuelta no tiene contenido, la lista retornada estará vacía.

fill(*text*)

Envuelve el único párrafo en *text*, y retorna una única cadena que contiene el párrafo envuelto.

[1] [2]

`unicodedata` — Unicode Database

This module provides access to the Unicode Character Database (UCD) which defines character properties for all Unicode characters. The data contained in this database is compiled from the UCD version 16.0.0 [https://www.unicode.org/Public/16.0.0/ucd].

El módulo utiliza los mismos nombres y símbolos definidos por el Anexo #44 del estándar Unicode de la «Base de datos de caracteres Unicode» [https://www.unicode.org/reports/tr44/]. Define las siguientes funciones:

Ver también

The CÓMO (HOWTO) Unicode for more information about Unicode and how to use this module.

unicodedata.lookup(*name*)

Look up character by name. If a character with the given name is found, return the corresponding character. If not found, **`KeyError`** is raised. For example:

```
>>> unicodedata.lookup('LEFT CURLY BRACKET')
'{'
```

The characters returned by this function are the same as those produced by `\N` escape sequence in string literals. For example:

```
>>> unicodedata.lookup('MIDDLE DOT') == '\N{MIDDLE
True
```

Distinto en la versión 3.3: Se ha agregado soporte para alias de

nombre [1] y secuencias con nombre [2].

unicodedata.name(*chr*, *default=None*, */*)

Returns the name assigned to the character *chr* as a string. If no name is defined, *default* is returned, or, if not given, `ValueError` is raised. For example:

```
>>> unicodedata.name('½')
'VULGAR FRACTION ONE HALF'
>>> unicodedata.name('\uFFFF', 'fallback')
'fallback'
```

unicodedata.decimal(*chr*, *default=None*, */*)

Returns the decimal value assigned to the character *chr* as integer. If no such value is defined, *default* is returned, or, if not given, `ValueError` is raised. For example:

```
>>> unicodedata.decimal('\N{ARABIC-INDIC DIGIT NINE
9
>>> unicodedata.decimal('\N{SUPERSCRIPT NINE}', -1)
-1
```

unicodedata.digit(*chr*, *default=None*, */*)

Returns the digit value assigned to the character *chr* as integer. If no such value is defined, *default* is returned, or, if not given, `ValueError` is raised:

```
>>> unicodedata.digit('\N{SUPERSCRIPT NINE}')
9
```

unicodedata.numeric(*chr*, *default=None*, */*)

Returns the numeric value assigned to the character *chr* as float. If no such value is defined, *default* is returned, or, if not given, `ValueError` is raised:

```
>>> unicodedata.numeric('½')
0.5
```

unicodedata.category(*chr*)

Returns the general category assigned to the character *chr* as string. General category names consist of two letters. See the General Category Values section of the Unicode Character Database documentation [https://www.unicode.org/reports/tr44/tr44-34.html#General_Category_Values] for a list of category codes. For example:

```
>>> unicodedata.category('A')  # 'L'etter, 'u'pperc
'Lu'
```

unicodedata.bidirectional(*chr*)

Returns the bidirectional class assigned to the character *chr* as string. If no such value is defined, an empty string is returned. See the Bidirectional Class Values section of the Unicode Character Database [https://www.unicode.org/reports/tr44/tr44-34.html#Bidi_Class_Values] documentation for a list of bidirectional codes. For example:

```
>>> unicodedata.bidirectional('\N{ARABIC-INDIC DIGI
'AN'
```

unicodedata.combining(*chr*)

Returns the canonical combining class assigned to the character *chr* as integer. Returns `0` if no combining class is defined. See the Canonical Combining Class Values section of the Unicode Character Database [https://www.unicode.org/reports/tr44/tr44-34.html#Canonical_Combining_Class_Values] for more information.

unicodedata.east_asian_width(*chr*)

Returns the east asian width assigned to the character *chr* as string. For a list of widths and or more information, see the Unicode Standard Annex #11 [https://www.unicode.org/reports/tr11/tr11-43.html].

unicodedata.mirrored(*chr*)

Returns the mirrored property assigned to the character *chr* as integer. Returns `1` if the character has been identified as a «mirrored» character in bidirectional text, `0` otherwise. For example:

```
>>> unicodedata.mirrored('>')
1
```

unicodedata.decomposition(*chr*)

Returns the character decomposition mapping assigned to the character *chr* as string. An empty string is returned in case no such mapping is defined. For example:

```
>>> unicodedata.decomposition('Ã')
'0041 0303'
```

unicodedata.normalize(*form*, *unistr*)

Retorna la forma normalizada *form* para la cadena Unicode *unistr*. Los valores válidos para *form* son “NFC”, “NFKC”, “NFD” y “NFKD”.

El estándar Unicode define varias formas de normalización de una cadena Unicode, basándose en la definición de equivalencia canónica y equivalencia de compatibilidad. En Unicode, varios caracteres se pueden expresar de diversas formas. Por ejemplo, el carácter U+00C7 (LETRA C LATINA MAYÚSCULA CON CEDILLA) también se puede expresar con la secuencia U+0043 (LETRA C LATINA MAYÚSCULA) U+0327 (CEDILLA COMBINABLE).

Para cada carácter, hay dos formas normalizadas: la forma normal C y la forma normal D. La forma normal D (NFD) también se conoce como descomposición canónica y traduce cada carácter a su forma descompuesta. La forma normal C (NFC) primero aplica una descomposición canónica y luego vuelve a componer los caracteres combinados previamente.

In addition to these two forms, there are two additional normal forms based on compatibility equivalence. In Unicode, certain characters are supported which normally would be unified with other characters. For example, U+2160 (ROMAN NUMERAL ONE) is really the same thing as U+0049 (LATIN CAPITAL LETTER I). However, it is supported in Unicode for compatibility with existing character sets (for example, gb2312).

The normal form KD (NFKD) will apply the compatibility decomposition, that is, replace all compatibility characters with their equivalents. The normal form KC (NFKC) first applies the compatibility decomposition, followed by the canonical composition.

Incluso si dos cadenas Unicode están normalizadas y parecen iguales para un lector humano, si una tiene caracteres combinados y la otra no, es posible que no se comparen como iguales.

unicodedata.is_normalized(*form*, *unistr*)

Retorna si la cadena Unicode *unistr* está en la forma normalizada *form*. Los valores válidos para *form* son "NFC", "NFKC", "NFD" y "NFKD".

Added in version 3.8.

Además, el módulo expone las siguientes constantes:

unicodedata.unidata_version

La versión de la base de datos Unicode usada en este módulo.

unicodedata.ucd_3_2_0

Este es un objeto que tiene los mismos métodos que el módulo completo, pero usa la versión 3.2 de la base de datos Unicode en su lugar. Es útil para aplicaciones que requieren esta versión específica de la base de datos Unicode (como IDNA).

Notas al pie

https://www.unicode.org/Public/16.0.0/ucd/NameAliases.txt

https://www.unicode.org/Public/16.0.0/ucd/NamedSequences.txt

`stringprep` — Internet String Preparation

Código fuente: Lib/stringprep.py [https://github.com/python/cpython/tree/3.14/Lib/stringprep.py]

Cuando se quiere identificar cosas (como nombres de host) en internet, generalmente es necesario comparar tales identificaciones para «igualdad». La manera en la que esta comparación se ejecuta dependerá del dominio de la aplicación, ej. si tiene o no tiene que distinguir entre mayúsculas y minúsculas. Además, en algunos casos será necesario restringir las posibles identificaciones, de tal manera que solo se permitan identificadores de caracteres que se puedan «imprimir».

RFC 3454 [https://datatracker.ietf.org/doc/html/rfc3454.html] define el proceso para la «preparación» de cadenas Unicode para protocolos de internet. Antes de pasar cadenas a un cable, se procesan con el proceso de preparación, después del cual tienen una forma normalizada. El RFC define un conjunto de tablas, que pueden ser combinadas en perfiles. Cada perfil debe definir qué tablas usa, y que partes opcionales del proceso `stringprep` son parte del perfil. Un ejemplo de perfil de `stringprep` es `nameprep`, que se usa para nombres de dominios internacionalizados.

The module **`stringprep`** only exposes the tables from **RFC 3454** [https://datatracker.ietf.org/doc/html/rfc3454.html]. As these tables would be very large to represent as dictionaries or lists, the module uses the Unicode character database internally. The module source code itself was generated using the `mkstringprep.py` utility.

As a result, these tables are exposed as functions, not as data structures. There are two kinds of tables in the RFC: sets and mappings. For a set, **`stringprep`** provides the «characteristic function», i.e. a function that returns `True` if the parameter is part of the set. For mappings, it provides the mapping function: given the key, it returns the associated value. Below is a list of all functions available in the module.

stringprep.in_table_a1(*code*)

Determina si *code* está en la tablaA.1 (Puntos de Código sin asignar en Unicode 3.2).

stringprep.in_table_b1(*code*)

Determina si *code* está en la tablaB.1 (Generalmente mapeado a nada).

stringprep.map_table_b2(*code*)

Retorna el valor mapeado para *code* de acuerdo a la tablaB.2 (Mapeo para *case-folding* usado con NFKC).

stringprep.map_table_b3(*code*)

Retorna el valor mapeado para *code* de acuerdo a tablaB.3 (Mapeo para *case-folding* usado sin normalización).

stringprep.in_table_c11(*code*)

Determina si *code* está en la tablaC.1.1 (Caracteres de espacio ASCII).

stringprep.in_table_c12(*code*)

Determina si *code* está en la tablaC.1.2 (Caracteres de espacio no-ASCII).

stringprep.in_table_c11_c12(*code*)

Determina si *code* está en la tablaC.1 (Caracteres de espacio, unión de C.1.1 y C.1.2).

stringprep.in_table_c21(*code*)

Determina si *code* está en la tablaC.2.1 (Caracteres de control ASCII).

stringprep.in_table_c22(*code*)

Determina si *code* está en la tablaC.2.2 (Caracteres de control no ASCII).

stringprep.in_table_c21_c22(*code*)

Determina si *code* está en la tablaC.2 (Caracteres de control, unión de C.2.1 y C.2.2).

stringprep.in_table_c3(*code*)

Determina si *code* está en la tablaC.3 (Uso privado).

stringprep.in_table_c4(*code*)

Determina si *code* está en la tablaC.4 (Puntos de código sin caracteres)

stringprep.in_table_c5(*code*)

Determina si *code* está en la tablaC.5 (Códigos surrogados).

stringprep.in_table_c6(*code*)

Determina si *code* está en la tablaC.6 (Inadecuado para texto plano).

stringprep.in_table_c7(*code*)

Determina si *code* está en la tablaC.7 (Inadecuado para la representación canónica).

stringprep.in_table_c8(*code*)

Determina si *code* está en la tablaC.8 (Cambia las propiedades de apariencia o están obsoletas).

stringprep.in_table_c9(*code*)

Determina si *code* está en la tablaC.9 (Caracteres de etiquetado).

stringprep.in_table_d1(*code*)

Determina si *code* está en la tablaD.1 (Caracteres con propiedad bidireccional «R» o «AL»).

stringprep.in_table_d2(*code*)

Determina si *code* está en la tablaD.2 (Caracteres con propiedad bidireccional «L»).

`readline` — GNU readline interface

The **`readline`** module defines a number of functions to facilitate completion and reading/writing of history files from the Python interpreter. This module can be used directly, or via the **`rlcompleter`** module, which supports completion of Python identifiers at the interactive prompt. Settings made using this module affect the behaviour of both the interpreter's interactive prompt and the prompts offered by the built-in **`input()`** function.

Las combinaciones de teclas de Readline se pueden configurar mediante un archivo de inicialización, generalmente `.inputrc` en su directorio de inicio. Consulte Readline Init File [https://tiswww.cwru.edu/php/chet/readline/rluserman.html#Readline-Init-File] en el manual de GNU Readline para obtener información sobre el formato y las construcciones permitidas de ese archivo, y las capacidades de la biblioteca Readline en general.

Availability: not Android, not iOS, not WASI.

This module is not supported on mobile platforms or WebAssembly platforms.

This is an optional module. If it is missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

Availability: Unix.

Nota

The underlying Readline library API may be implemented by the `editline` (`libedit`) library instead of GNU readline. On macOS the **`readline`** module detects which library is being used at run time.

The configuration file for `editline` is different from that of GNU readline. If you programmatically load configuration strings you can use **`backend`** to determine which library is being used.

If you use `editline`/`libedit` readline emulation on macOS, the initialization file located in your home directory is named `.editrc`. For example, the following content in `~/.editrc` will turn ON *vi* keybindings and TAB completion:

```
python:bind -v
python:bind ^I rl_complete
```

Also note that different libraries may use different history file formats. When switching the underlying library, existing history files may become unusable.

readline.backend

The name of the underlying Readline library being used, either `"readline"` or `"editline"`.

Added in version 3.13.

Archivo de inicio

Las siguientes funciones se relacionan con el archivo de inicio y la configuración del usuario:

readline.parse_and_bind(*string*)

Execute the init line provided in the *string* argument. This calls **`rl_parse_and_bind()`** in the underlying library.

readline.read_init_file([*filename*])

Execute a readline initialization file. The default filename is the last filename used. This calls **`rl_read_init_file()`** in the underlying library. It raises an auditing event `open` with the file name if given, and `"<readline_init_file>"`

otherwise, regardless of which file the library resolves.

Distinto en la versión 3.14: The auditing event was added.

Búfer de línea

Las siguientes funciones operan en el búfer de línea:

readline.get_line_buffer()

Return the current contents of the line buffer (**`rl_line_buffer`** in the underlying library).

readline.insert_text(*string*)

Insert text into the line buffer at the cursor position. This calls **`rl_insert_text()`** in the underlying library, but ignores the return value.

readline.redisplay()

Change what's displayed on the screen to reflect the current contents of the line buffer. This calls **`rl_redisplay()`** in the underlying library.

Archivo de historial

Las siguientes funciones operan en un archivo de historial:

readline.read_history_file([*filename*])

Load a readline history file, and append it to the history list. The default filename is `~/.history`. This calls **`read_history()`** in the underlying library and raises an auditing event `open` with the file name if given and `"~/.history"` otherwise.

Distinto en la versión 3.14: The auditing event was added.

readline.write_history_file([*filename*])

Save the history list to a readline history file, overwriting any existing file. The default filename is `~/.history`. This calls **`write_history()`** in the underlying library and raises an auditing event `open` with the file name if given and `"~/.history"` otherwise.

Distinto en la versión 3.14: The auditing event was added.

readline.append_history_file(*nelements*[, *filename*])

Append the last *nelements* items of history to a file. The default filename is `~/.history`. The file must already exist. This calls **`append_history()`** in the underlying library. This function only exists if Python was compiled for a version of the library that supports it. It raises an auditing event `open` with the file name if given and `"~/.history"` otherwise.

Added in version 3.5.

Distinto en la versión 3.14: The auditing event was added.

readline.get_history_length()

readline.set_history_length(*length*)

Set or return the desired number of lines to save in the history file. The **`write_history_file()`** function uses this value to truncate the history file, by calling **`history_truncate_file()`** in the underlying library. Negative values imply unlimited history file size.

Lista del historial

Las siguientes funciones operan en una lista de historial global:

readline.clear_history()

Clear the current history. This calls **`clear_history()`** in the underlying library. The Python function only exists if Python was compiled for a version of the library that supports it.

readline.get_current_history_length()

Retorna el número de elementos actuales en el historial. (Esto es diferente de la función `get_history_length()`, que retorna el número máximo de líneas que se escribirán en un archivo de historial).

readline.get_history_item(*index*)

Return the current contents of history item at *index*. The item index is one-based. This calls **`history_get()`** in the underlying library.

readline.remove_history_item(*pos*)

Remove history item specified by its position from the history. The position is zero-based. This calls **`remove_history()`** in the underlying library.

readline.replace_history_item(*pos*, *line*)

Replace history item specified by its position with *line*. The position is zero-based. This calls **`replace_history_entry()`** in the underlying library.

readline.add_history(*line*)

Append *line* to the history buffer, as if it was the last line typed. This calls **`add_history()`** in the underlying library.

readline.set_auto_history(*enabled*)

Enable or disable automatic calls to **`add_history()`** when reading input via readline. The *enabled* argument should be a Boolean value that when true, enables auto history, and that when false, disables auto history.

Added in version 3.6.

Auto history está habilitado de forma predeterminada, y cambiarlo no hará que persista en múltiples sesiones.

Ganchos (*hooks*) de inicialización

readline.set_startup_hook([*function*])

Set or remove the function invoked by the **`rl_startup_hook`** callback of the underlying library. If *function* is specified, it will be used as the new hook function; if omitted or `None`, any function already installed is removed. The hook is called with no arguments just before readline prints the first prompt.

readline.set_pre_input_hook([*function*])

Set or remove the function invoked by the **`rl_pre_input_hook`** callback of the underlying library. If *function* is specified, it will be used as the new hook function; if omitted or `None`, any function already installed is removed. The hook is called with no arguments after the first prompt has been printed and just before readline starts reading input characters. This function only exists if Python was compiled for a version of the library that supports it.

Terminación

The following functions relate to implementing a custom word completion function. This is typically operated by the Tab key, and can suggest and automatically complete a word being typed. By default, Readline is set up to be used by **`rlcompleter`** to complete Python identifiers for the interactive interpreter. If the **`readline`** module is to be used with a custom completer, a different set of word delimiters should be set.

readline.set_completer([*function*])

Establece o elimina la función de finalización. Si se especifica *function*, se usará como la nueva función de finalización; Si se omite o es `None`, se elimina cualquier función de finalización ya instalada. La función completa se llama como `function(text, state)`, para *state* en `0`, `1`, `2`, …, hasta que retorna un valor que no es una cadena de caracteres.

Debería retornar las siguientes terminaciones posibles comenzando con *text*.

The installed completer function is invoked by the *entry_func* callback passed to **`rl_completion_matches()`** in the underlying library. The *text* string comes from the first parameter to the **`rl_attempted_completion_function`** callback of the underlying library.

readline.get_completer()

Obtiene la función de finalización o `None` si no se ha definido ninguna función de finalización.

readline.get_completion_type()

Get the type of completion being attempted. This returns the **`rl_completion_type`** variable in the underlying library as an integer.

readline.get_begidx()
readline.get_endidx()

Get the beginning or ending index of the completion scope. These indexes are the *start* and *end* arguments passed to the **`rl_attempted_completion_function`** callback of the underlying library. The values may be different in the same input editing scenario based on the underlying C readline implementation. Ex: libedit is known to behave differently than libreadline.

readline.set_completer_delims(*string*)
readline.get_completer_delims()

Set or get the word delimiters for completion. These determine the start of the word to be considered for completion (the completion scope). These functions access the **`rl_completer_word_break_characters`** variable in the underlying library.

readline.set_completion_display_matches_hook([*function*])

Set or remove the completion display function. If *function* is specified, it will be used as the new completion display function; if omitted or `None`, any completion display function already installed is removed. This sets or clears the **`rl_completion_display_matches_hook`** callback in the underlying library. The completion display function is called as `function(substitution, [matches], longest_match_length)` once each time matches need to be displayed.

Ejemplo

The following example demonstrates how to use the **`readline`** module's history reading and writing functions to automatically load and save a history file named `.python_history` from the user's home directory. The code below would normally be executed automatically during interactive sessions from the user's **`PYTHONSTARTUP`** file.

```
import atexit
import os
import readline

histfile = os.path.join(os.path.expanduser("~"), ".pytho
try:
    readline.read_history_file(histfile)
    # default history len is -1 (infinite), which may gr
    readline.set_history_length(1000)
except FileNotFoundError:
    pass

atexit.register(readline.write_history_file, histfile)
```

Este código se ejecuta automáticamente cuando Python se ejecuta en modo interactivo (ver Configuración de Readline).

El siguiente ejemplo logra el mismo objetivo pero administra sesiones interactivas concurrentes, agregando solo el nuevo historial.

```
import atexit
import os
import readline
histfile = os.path.join(os.path.expanduser("~"), ".pytho

try:
    readline.read_history_file(histfile)
    h_len = readline.get_current_history_length()
except FileNotFoundError:
    open(histfile, 'wb').close()
    h_len = 0

def save(prev_h_len, histfile):
    new_h_len = readline.get_current_history_length()
    readline.set_history_length(1000)
    readline.append_history_file(new_h_len - prev_h_len,
atexit.register(save, h_len, histfile)
```

El siguiente ejemplo amplía la clase `code.InteractiveConsole` para administrar el guardado/restauración del historial.

```
import atexit
import code
import os
import readline

class HistoryConsole(code.InteractiveConsole):
    def __init__(self, locals=None, filename="<console>"
                 histfile=os.path.expanduser("~/.console
        code.InteractiveConsole.__init__(self, locals, f
        self.init_history(histfile)

    def init_history(self, histfile):
        readline.parse_and_bind("tab: complete")
        if hasattr(readline, "read_history_file"):
            try:
                readline.read_history_file(histfile)
            except FileNotFoundError:
                pass
            atexit.register(self.save_history, histfile)

    def save_history(self, histfile):
        readline.set_history_length(1000)
        readline.write_history_file(histfile)
```

Nota

The new REPL introduced in version 3.13 doesn’t support readline. However, readline can still be used by setting the **`PYTHON_BASIC_REPL`** environment variable.

`rlcompleter` — Completion function for GNU readline

Código fuente: Lib/rlcompleter.py [https://github.com/python/cpython/tree/3.14/Lib/rlcompleter.py]

The **`rlcompleter`** module defines a completion function suitable to be passed to `set_completer()` in the `readline` module.

When this module is imported on a Unix platform with the `readline` module available, an instance of the `Completer` class is automatically created and its `complete()` method is set as the readline completer. The method provides completion of valid Python identifiers and keywords.

Ejemplo:

```
>>> import rlcompleter
>>> import readline
>>> readline.parse_and_bind("tab: complete")
>>> readline. <TAB PRESSED>
readline.__doc__          readline.get_line_buffer(  rea
readline.__file__         readline.insert_text(      rea
readline.__name__         readline.parse_and_bind(
>>> readline.
```

The **`rlcompleter`** module is designed for use with Python's interactive mode. Unless Python is run with the `-S` option, the module is automatically imported and configured (see Configuración de Readline).

En plataformas sin `readline`, la clase `Completer` definida por este módulo puede ser usada igualmente para fines personalizados.

class rlcompleter.Completer

Los objetos de completado tienen el siguiente método:

complete(*text*, *state*)

Return the next possible completion for *text*.

When called by the **`readline`** module, this method is called successively with `state == 0, 1, 2, ...` until the method returns `None`.

Si es invocado para *text* que no incluye un caracter de punto (`'.'`), este completará con nombres actualmente definidos en **`__main__`**, **`builtins`** y las palabras clave (tal y como están definidas en el módulo **`keyword`**).

If called for a dotted name, it will try to evaluate anything without obvious side-effects (functions will not be evaluated, but it can generate calls to **`__getattr__()`**) up to the last part, and find matches for the rest via the **`dir()`** function. Any exception raised during the evaluation of the expression is caught, silenced and **`None`** is returned.

Servicios de datos binarios

Los módulos descritos en este capítulo proporcionan algunas operaciones básicas de servicios para la manipulación de datos binarios. Otras operaciones sobre datos binarios específicamente relacionadas con formatos de archivo y protocolos de red están descritas en las secciones relevantes.

Algunas bibliotecas descritas bajo Servicios de procesamiento de texto también funcionan o bien sobre formatos binarios compatibles con ASCII (por ejemplo **`re`**), o bien sobre todos los datos binarios (por ejemplo **`difflib`**).

Adicionalmente, véase la documentación para los tipos de datos binarios incorporados en Python en Tipos de secuencias binarias — bytes, bytearray y memoryview.

- **`struct`** — Interpret bytes as packed binary data
  - Funciones y excepciones
  - Cadenas de formato
    - Orden de bytes, tamaño y alineación
    - Caracteres de formato
    - Ejemplos
  - Aplicaciones
    - Formatos nativos
    - Formatos estándar
  - Clases
- **`codecs`** — Codec registry and base classes
  - Clases Base de Códec
    - Manejadores de errores

  - Codificación y decodificación sin estado
  - Codificación y decodificación incrementales
    - Objetos IncrementalEncoder
    - Objetos IncrementalDecoder
  - Codificación y decodificación de flujos
    - Objetos StreamWriter
    - Objetos StreamReader
    - Objetos StreamReaderWriter
    - Objetos StreamRecoder
- Codificaciones y Unicode
- Codificaciones estándar
- Codificaciones específicas de Python
  - Codificaciones de texto
  - Transformaciones Binarias
  - Standalone Codec Functions
  - Transformaciones de texto
- **`encodings`** — Encodings package
- **`encodings.idna`** — Internationalized Domain Names in Applications
- **`encodings.mbcs`** — Windows ANSI codepage
- **`encodings.utf_8_sig`** — UTF-8 codec with BOM signature

`struct` — Interpret bytes as packed binary data

Código fuente: Lib/struct.py [https://github.com/python/cpython/tree/3.14/Lib/struct.py]

Este módulo convierte entre valores de Python y estructuras de C representadas como objetos de Python `bytes`. Las cadenas de formato compacto describen las conversiones previstas hacia/desde valores de Python. Las funciones y objetos del módulo se pueden utilizar para dos aplicaciones en gran medida distintas, el intercambio de datos con fuentes externas (archivos o conexiones de red) o la transferencia de datos entre la aplicación de Python y la capa de C.

Nota

Cuando no se proporciona ningún carácter de prefijo, el modo nativo es el predeterminado. Empaqueta o desempaqueta datos según la plataforma y el compilador en el que se creó el intérprete de Python. El resultado de empaquetar una estructura C determinada incluye bytes de relleno que mantienen la alineación adecuada para los tipos C involucrados; asimismo, se tiene en cuenta la alineación al desempaquetar. Por el contrario, cuando se comunican datos entre fuentes externas, el programador es responsable de definir el orden de bytes y el relleno entre elementos. Ver Orden de bytes, tamaño y alineación para más detalles.

Several `struct` functions (and methods of `Struct`) take a *buffer* argument. This refers to objects that implement the Protocolo búfer and provide either a readable or read-writable buffer. The most common types used for that purpose are `bytes` and `bytearray`, but many other types that can be viewed as an array of bytes implement the buffer protocol, so that they can be read/filled without additional copying from a `bytes` object.

Funciones y excepciones

El módulo define la siguiente excepción y funciones:

exception struct.error

> Excepción lanzada en varias ocasiones; el argumento es una *string* que describe lo que está mal.

struct.pack(*format, v1, v2, ...*)

> Retorna un objeto bytes que contiene los valores *v1, v2, …* empaquetado de acuerdo con la cadena de formato *format*. Los argumentos deben coincidir exactamente con los valores requeridos por el formato.

struct.pack_into(*format, buffer, offset, v1, v2, ...*)

> Empaqueta los valores *v1, v2, …* de acuerdo con la cadena de formato *format* y escribe los bytes empaquetados en el búfer de escritura *buffer* comenzando en la posición *offset*. Nota: *offset* es un argumento obligatorio.

struct.unpack(*format, buffer*)

> Desempaqueta del búfer *buffer* (normalmente empaquetado por `pack(format, ...)`) según la cadena de formato *format*. El resultado es una tupla incluso si contiene un solo elemento. El tamaño del búfer en bytes debe coincidir con el tamaño requerido por el formato, como se refleja en `calcsize()`.

struct.unpack_from(*format, /, buffer, offset=0*)

> Desempaqueta del *buffer* comenzando en la posición *offset*, según la cadena de formato *format*. El resultado es una tupla incluso si contiene un solo elemento. El tamaño del búfer en bytes, comenzando en la posición *offset*, debe tener al menos el tamaño requerido por el formato, como se refleja en

`calcsize()`.

struct.iter_unpack(*format, buffer*)

Desempaqueta de manera iterativa desde el búfer *buffer* según la cadena de formato *format*. Esta función retorna un iterador que leerá fragmentos de igual tamaño desde el búfer hasta que se haya consumido todo su contenido. El tamaño del búfer en bytes debe ser un múltiplo del tamaño requerido por el formato, como se refleja en `calcsize()`.

Cada iteración produce una tupla según lo especificado por la cadena de formato.

Added in version 3.4.

struct.calcsize(*format*)

Retorna el tamaño de la estructura (y, por lo tanto, del objeto bytes generado por `pack(format, ...)`) correspondiente a la cadena de formato *format*.

Cadenas de formato

Las cadenas de formato describen el diseño de los datos cuando empaquetan y desempaquetan datos. Se construyen a partir caracteres de formato, que especifican el tipo de datos que se empaquetan/desempaquetan. Además, los caracteres especiales controlan el orden de bytes, tamaño y alineación. Cada cadena de formato consta de un carácter de prefijo opcional que describe las propiedades generales de los datos y uno o más caracteres de formato que describen los valores de datos actuales y el relleno.

Orden de bytes, tamaño y alineación

Por defecto, los tipos C se representan en el formato nativo y el orden de bytes de la máquina, y se alinean correctamente omitiendo *bytes* de relleno si es necesario (según las reglas utilizadas por el compilador de C). Se elige este comportamiento para que los bytes de una estructura empaquetada correspondan correctamente al diseño de memoria de la estructura C correspondiente. El uso de ordenamiento y relleno de bytes nativos o formatos estándar depende de la aplicación.

Como alternativa, el primer carácter de la cadena de formato se puede utilizar para indicar el orden de bytes, el tamaño y la alineación de los datos empaquetados, según la tabla siguiente:

Carácter	Orden de bytes	Tamaño	Alineación
@	nativo	nativo	nativo
=	nativo	standard	ninguno
<	little-endian	standard	ninguno
>	big-endian	standard	ninguno
!	red (= big-endian)	standard	ninguno

Si el primer carácter no es uno de estos, se asume `'@'`.

Nota

The number 1023 (`0x3ff` in hexadecimal) has the following byte representations:

- `03 ff` in big-endian (>)
- `ff 03` in little-endian (<)

Python example:

```
>>> import struct
>>> struct.pack('>h', 1023)
b'\x03\xff'
>>> struct.pack('<h', 1023)
b'\xff\x03'
```

El orden de bytes nativo es big-endian o little-endian, dependiendo del sistema host. Por ejemplo, Intel x86, AMD64 (x86-64) y Apple M1 son little-endian; IBM z y muchas arquitecturas heredadas son big-endian; ARM e Intel *Itanium* tienen la propiedad de trabajar con ambos formatos (middle-endian). Utiliza `sys.byteorder` para comprobar la endianness («extremidad») de tu sistema.

El tamaño y la alineación nativos se determinan mediante la expresión `sizeof` del compilador de C. Esto siempre se combina con el orden de bytes nativo.

El tamaño estándar depende únicamente del carácter de formato; ver la tabla en la sección Caracteres de formato.

Nótese la diferencia entre `'@'` y `'='`: ambos utilizan el orden de bytes nativo, pero el tamaño y la alineación de este último está estandarizado.

La forma `'!'` representa el orden de bytes en la red, el cuál siempre es big-endian tal como se define en IETF RFC 1700 [https://datatracker.ietf.org/doc/html/rfc1700].

No hay manera de indicar el orden de bytes no nativo (forzar el intercambio de bytes); utiliza la elección adecuada de `'<'` o `'>'`.

Notas:

1. El relleno solo se agrega automáticamente entre los miembros sucesivos de la estructura. No se agrega ningún relleno al principio o al final de la estructura codificada.
2. No se añade ningún relleno cuando se utiliza el tamaño y la alineación no nativos, por ejemplo, con "<", ">", "=" y "!".
3. Para alinear el final de una estructura con el requisito de alineación de un tipo determinado, termina el formato con el código de ese tipo con un conteo repetido de ceros. Véase Ejemplos.

Caracteres de formato

Los caracteres de formato tienen el siguiente significado; la conversión entre los valores de C y Python debe ser obvia dados sus tipos. La columna "Tamaño estándar" hace referencia al tamaño del valor empaquetado en bytes cuando se utiliza el tamaño estándar; es decir, cuando la cadena de formato comienza con uno de `'<'`, `'>'`, `'!'` o `'='`. Cuando se utiliza el tamaño nativo, el tamaño del valor empaquetado depende de la plataforma.

Formato	Tipo C	Tipo Python	Tamaño estándar	Notas

byte de relleno	sin valor		(7)
char	bytes de longitud 1	1	
signed char	int	1	(1), (2)
unsigned char	int	1	(2)
_Bool	bool	1	(1)
short	int	2	(2)
unsigned short	int	2	(2)
int	int	4	(2)
unsigned int	int	4	(2)
long	int	4	(2)
unsigned long	int	4	(2)
long long	int	8	(2)
unsigned long long	int	8	(2)
`ssize_t`	int		(2), (3)
`size_t`	int		(2), (3)
(6)	float	2	(4), (6)
float	float	4	(4)
double	float	8	(4)
float complex	complex	8	(10)
double complex	complex	16	(10)
char[]	bytes		(9)
char[]	bytes		(8)
void*	int		(2), (5)

Distinto en la versión 3.3: Soporte añadido para los formatos `'n'` y `'N'`.

Distinto en la versión 3.6: Soporte añadido para el formato `'e'`.

Distinto en la versión 3.14: Added support for the `'F'` and `'D'` formats.

Ver también

The **`array`** and ctypes modules, as well as third-party modules like numpy [https://numpy.org/doc/stable/reference/arrays.interface.html#object.__array_interface__], use similar – but slightly different – type codes.

Notas:

1. The `'?'` conversion code corresponds to the _Bool type defined by C standards since C99. In standard mode, it is represented by one byte.

2. Al intentar empaquetar un no entero mediante cualquiera de los códigos de conversión de enteros, si el no entero tiene un método **`__index__()`**, se llama a ese método para convertir el argumento en un entero antes de empaquetar.

   Distinto en la versión 3.2: Agregado el uso del método **`__index__()`** para los no enteros.

3. Los códigos de conversión `'n'` y `'N'` solo están disponibles para el tamaño nativo (seleccionado como predeterminado o con el carácter de orden de bytes `'@'`). Para el tamaño estándar, puedes usar cualquiera de los otros formatos enteros que se ajusten a tu aplicación.

4. Para los códigos de conversión `'f'`, `'d'` y `'e'`, la representación empaquetada utiliza el formato IEEE 754 binary32, binary64 o binary16 (para `'f'`, `'d'` o `'e'` respectivamente), independientemente del formato de punto flotante utilizado por la plataforma.

5. El carácter de formato `'P'` solo está disponible para el orden nativo de bytes (seleccionado como predeterminado o con el carácter de orden de bytes `'@'`). El carácter de orden de bytes `'='` elige utilizar el orden little- o big-endian basado en el sistema host. El módulo *struct* no interpreta esto como un orden nativo, por lo que el formato `'P'` no está disponible.

6. The IEEE 754 binary16 «half precision» type was introduced in the 2008 revision of the IEEE 754 standard [https://en.wikipedia.org/wiki/IEEE_754-2008_revision]. It has a sign bit, a 5-bit exponent and 11-bit precision (with 10 bits explicitly stored), and can represent numbers between approximately `6.1e-05` and `6.5e+04` at full precision. This type is not widely supported by C compilers: it's available as _Float16 type, if the compiler supports the Annex H of the C23 standard. On a typical machine, an unsigned short can be

used for storage, but not for math operations. See the Wikipedia page on the half-precision floating-point format [https://en.wikipedia.org/wiki/Half-precision_floating-point_format] for more information.

7. Al empaquetar, `'x'` inserta un byte NUL.

8. The `'p'` format character encodes a «Pascal string», meaning a short variable-length string stored in a *fixed number of bytes*, given by the count. The first byte stored is the length of the string, or 255, whichever is smaller. The bytes of the string follow. If the byte string passed in to **`pack()`** is too long (longer than the count minus 1), only the leading `count-1` bytes of the string are stored. If the byte string is shorter than `count-1`, it is padded with null bytes so that exactly count bytes in all are used. Note that for **`unpack()`**, the `'p'` format character consumes `count` bytes, but that the **`bytes`** object returned can never contain more than 255 bytes. When packing, arguments of types **`bytes`** and **`bytearray`** are accepted.

9. For the `'s'` format character, the count is interpreted as the length of the byte string, not a repeat count like for the other format characters; for example, `'10s'` means a single 10-byte string mapping to or from a single Python byte string, while `'10c'` means 10 separate one byte character elements (e.g., `cccccccccc`) mapping to or from ten different Python byte objects. (See Ejemplos for a concrete demonstration of the difference.) If a count is not given, it defaults to 1. For packing, the byte string is truncated or padded with null bytes as appropriate to make it fit. For unpacking, the resulting **`bytes`** object always has exactly the specified number of bytes. As a special case, `'0s'` means a single, empty byte string (while `'0c'` means 0 characters). When packing, arguments of types **`bytes`** and **`bytearray`** are accepted.

10. For the `'F'` and `'D'` format characters, the packed representation uses the IEEE 754 binary32 and binary64 format for components of the complex number, regardless of the floating-point format used by the platform. Note that

complex types (`F` and `D`) are available unconditionally, despite complex types being an optional feature in C. As specified in the C11 standard, each complex type is represented by a two-element C array containing, respectively, the real and imaginary parts.

Un carácter de formato puede ir precedido de un número de recuento que repite tantas veces el carácter. Por ejemplo, la cadena de formato `'4h'` significa exactamente lo mismo que `'hhhh'`.

Se omiten los caracteres de espacio entre formatos; sin embargo, un recuento y su formato no deben contener espacios en blanco.

Al empaquetar un valor `x` utilizando uno de los formatos enteros (`'b'`, `'B'`, `'h'`, `'H'`, `'i'`, `'I'`, `'l'`, `'L'`, `'q'`, `'Q'`), si `x` está fuera de un rango válido para ese formato, entonces se lanza la excepción **`struct.error`**.

Distinto en la versión 3.1: Anteriormente, algunos de los formatos enteros ajustaban los valores fuera de rango y lanzaban **`DeprecationWarning`** en vez de **`struct.error`**.

Para el carácter de formato `'?'`, el valor retornado es **`True`** o **`False`**. Al empaquetar, se utiliza el valor verdadero del objeto del argumento. Se empaquetará 0 o 1 en la representación *bool* nativa o estándar, y cualquier valor distinto de cero será `True` al desempaquetar.

Ejemplos

Nota

Los ejemplos de orden de bytes nativos (designados por el prefijo de formato `'@'` o la falta de cualquier carácter de prefijo) pueden no coincidir con lo que produce la máquina del lector, ya que eso depende de la plataforma y el compilador.

Empaqueta y desempaqueta enteros de tres tamaños diferentes, utilizando el orden *big endian*:

```
>>> from struct import *
>>> pack(">bhl", 1, 2, 3)
b'\x01\x00\x02\x00\x00\x00\x03'
>>> unpack('>bhl', b'\x01\x00\x02\x00\x00\x00\x03')
(1, 2, 3)
>>> calcsize('>bhl')
7
```

Intenta empaquetar un entero que es demasiado grande para el campo definido:

```
>>> pack(">h", 99999)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
struct.error: 'h' format requires -32768 <= number <= 32
```

Demuestra la diferencia entre los caracteres de formato `'s'` y `'c'`:

```
>>> pack("@ccc", b'1', b'2', b'3')
b'123'
>>> pack("@3s", b'123')
b'123'
```

Los campos desempaquetados se pueden nombrar asignándolos a variables o ajustando el resultado en una tupla con nombre:

```
>>> record = b'raymond   \x32\x12\x08\x01\x08'
>>> name, serialnum, school, gradelevel = unpack('<10sHH

>>> from collections import namedtuple
>>> Student = namedtuple('Student', 'name serialnum scho
>>> Student._make(unpack('<10sHHb', record))
Student(name=b'raymond   ', serialnum=4658, school=264,
```

El orden de los caracteres de formato puede afectar el tamaño en el modo nativo, ya que el relleno está implícito. En el modo estándar, el usuario es responsable de insertar el relleno que desee. Toma en cuenta que en la primera llamada `pack` a continuación se agregaron tres bytes NUL después del `'#'` empaquetado para alinear el siguiente entero en un límite de cuatro bytes. En este ejemplo, el resultado se produjo en una máquina little endian:

```
>>> pack('@ci', b'#', 0x12131415)
b'#\x00\x00\x00\x15\x14\x13\x12'
>>> pack('@ic', 0x12131415, b'#')
b'\x15\x14\x13\x12#'
>>> calcsize('@ci')
8
>>> calcsize('@ic')
5
```

El siguiente formato `'llh0l'` especifica dos bytes de relleno al final, suponiendo que los tipos *longs* están alineados en los límites de 4 bytes:

```
>>> pack('@llh0l', 1, 2, 3)
b'\x00\x00\x00\x01\x00\x00\x00\x02\x00\x03\x00\x00'
```

Ver también

Módulo `array`
Almacenamiento binario empaquetado de datos homogéneos.

Módulo `json`
Codificador y decodificador JSON.

Módulo `pickle`
Serialización de objetos Python.

Aplicaciones

Two main applications for the **`struct`** module exist, data interchange between Python and C code within an application or another application compiled using the same compiler (native formats), and data interchange between applications using agreed upon data layout (standard formats). Generally speaking, the format strings constructed for these two domains are distinct.

Formatos nativos

Al construir cadenas de formato que imitan diseños nativos, el compilador y la arquitectura de la máquina determinan el orden y el relleno de los bytes. En tales casos, se debe utilizar el carácter de formato `@` para especificar el orden de bytes nativo y el tamaño de los datos. Los bytes de relleno internos normalmente se insertan automáticamente. Es posible que se necesite un código de formato de repetición cero al final de una cadena de formato para redondear hasta el límite de bytes correcto para una alineación adecuada de fragmentos de datos consecutivos.

Considera estos dos ejemplos sencillos (en una máquina little-endian de 64 bits):

```
>>> calcsize('@lhl')
24
>>> calcsize('@llh')
18
```

Los datos no se rellenan hasta un límite de 8 bytes al final de la segunda cadena de formato sin el uso de relleno adicional. Un código de formato de repetición cero resuelve ese problema:

```
>>> calcsize('@llh0l')
24
```

El código de formato `'x'` se puede utilizar para especificar la repetición, pero para formatos nativos es mejor utilizar un formato de repetición cero como `'0l'`.

De forma predeterminada, se utiliza el orden y la alineación de bytes nativos, pero es mejor ser explícito y utilizar el carácter de prefijo `'@'`.

Formatos estándar

Cuando intercambia datos más allá de su proceso, como redes o almacenamiento, sé preciso. Especifica el orden de bytes, el tamaño y la alineación exactos. No asumas que coinciden con el orden nativo de una máquina en particular. Por ejemplo, el orden de los bytes de la red es big-endian, mientras que muchas CPU populares son little-endian. Al definir esto explícitamente, el usuario no necesita preocuparse por las especificaciones de la plataforma en la que se ejecuta su código. El primer carácter normalmente debería ser `<` o `>` (o `!`). El relleno es responsabilidad del programador. El carácter de formato de repetición cero no funcionará. En su lugar, el usuario debe agregar explícitamente `'x'` bytes de relleno donde sea necesario. Revisando los ejemplos de la sección anterior, tenemos:

```
>>> calcsize('<qh6xq')
24
>>> pack('<qh6xq', 1, 2, 3) == pack('@lhl', 1, 2, 3)
True
>>> calcsize('@llh')
18
>>> pack('@llh', 1, 2, 3) == pack('<qqh', 1, 2, 3)
True
>>> calcsize('<qqh6x')
24
>>> calcsize('@llh0l')
24
>>> pack('@llh0l', 1, 2, 3) == pack('<qqh6x', 1, 2, 3)
True
```

No se garantiza que los resultados anteriores (ejecutados en una máquina de 64 bits) coincidan cuando se ejecutan en diferentes máquinas. Por ejemplo, los siguientes ejemplos se ejecutaron en una máquina de 32 bits:

```
>>> calcsize('<qqh6x')
24
>>> calcsize('@llh0l')
12
>>> pack('@llh0l', 1, 2, 3) == pack('<qqh6x', 1, 2, 3)
False
```

Clases

The **`struct`** module also defines the following type:

class struct.Struct(*format*)

Retorna un nuevo objeto Struct que escribe y lee datos binarios según la cadena de formato *format*. Crear un objeto `Struct` una vez y llamar a sus métodos es más eficaz que llamar a las funciones a nivel de módulo con el mismo formato, ya que la cadena de formato solo se compila una vez.

> **Nota**
>
> Las versiones compiladas de las cadenas de formato más recientes pasadas a las funciones de nivel de módulo se almacenan en caché, por lo que los programas que utilizan solo unas pocas cadenas de formato no necesitan preocuparse por volver a usar una sola instancia **`Struct`**.

Los objetos Struct compilados admiten los siguientes métodos y atributos:

pack(*v1*, *v2*, *...*)

Idéntico a la función **`pack()`**, utilizando el formato compilado. (`len(result)` será igual a **`size`**.)

pack_into(*buffer*, *offset*, *v1*, *v2*, *...*)

Idéntico a la función **`pack_into()`**, utilizando el formato compilado.

unpack(*buffer*)

Idéntico a la función **`unpack()`**, utilizando el formato compilado. El tamaño del búfer en bytes debe ser igual a **`size`**.

unpack_from(*buffer*, *offset=0*)

Idéntico a la función **`unpack_from()`**, utilizando el formato compilado. El tamaño del búfer en bytes, comenzando en la posición *offset*, debe ser al menos `size`.

iter_unpack(*buffer*)

Idéntico a la función `iter_unpack()`, utilizando el formato compilado. El tamaño del búfer en bytes debe ser un múltiplo de `size`.

Added in version 3.4.

format

Cadena de formato utilizada para construir este objeto Struct.

Distinto en la versión 3.7: El tipo de cadena de formato es ahora `str` en lugar de `bytes`.

size

El tamaño calculado de la estructura (y, por lo tanto, del objeto bytes generado por el método `pack()`) correspondiente a `format`.

Distinto en la versión 3.13: The *repr()* of structs has changed. It is now:

```
>>> Struct('i')
Struct('i')
```

[1]

codecs — Codec registry and base classes

Código fuente: Lib/codecs.py [https://github.com/python/cpython/tree/3.14/Lib/codecs.py]

Este módulo define las clases base para los códecs estándar de Python (codificadores y decodificadores) y proporciona acceso al registro interno de códecs de Python, que administra el códec y el proceso de búsqueda del manejo de errores. La mayoría de los códecs estándar son text encodings, que codifican texto a bytes (y decodifican bytes a texto), pero también se proporcionan códecs que codifican texto a texto y bytes a bytes. Los códecs personalizados pueden codificar y decodificar entre tipos arbitrarios, pero algunas características del módulo están restringidas para usarse específicamente con text encodings o con códecs que codifican a **`bytes`**.

El módulo define las siguientes funciones para codificar y decodificar con cualquier códec:

codecs.encode(*obj*, *encoding='utf-8'*, *errors='strict'*)

> Codifica *obj* utilizando el códec registrado para *encoding*.
>
> Se pueden dar *errors* para establecer el esquema de manejo de errores deseado. El manejador de errores predeterminado es `'estricto'`, lo que significa que los errores de codificación provocan **`ValueError`** (o una subclase más específica del códec, como **`UnicodeEncodeError`**). Consulte Clases Base de Códec para obtener más información sobre el manejo de errores de códec.

codecs.decode(*obj*, *encoding='utf-8'*, *errors='strict'*)

> Decodifica *obj* utilizando el códec registrado para *encoding*.

Se pueden dar *errors* para establecer el esquema de manejo de errores deseado. El manejador de errores predeterminado es `'estricto'`, lo que significa que los errores de decodificación generan **`ValueError`** (o una subclase más específica de códec, como **`UnicodeDecodeError`**). Consulte Clases Base de Códec para obtener más información sobre el manejo de errores de códec.

codecs.charmap_build(*string*)

Return a mapping suitable for encoding with a custom single-byte encoding. Given a **`str`** *string* of up to 256 characters representing a decoding table, returns either a compact internal mapping object `EncodingMap` or a **`dictionary`** mapping character ordinals to byte values. Raises a **`TypeError`** on invalid input.

Los detalles completos de cada códec también se pueden consultar directamente:

codecs.lookup(*encoding*, */*)

Busca la información de códec en el registro de códec de Python y retorna un objeto **`CodecInfo`** como se define a continuación.

Las codificaciones se buscan primero en la memoria caché del registro. Si no se encuentran, se explora la lista de funciones de búsqueda registradas. Si no se encuentran objetos **`CodecInfo`**, se lanza un **`LookupError`**. De lo contrario, el objeto **`CodecInfo`** se almacena en la memoria caché y se retorna a quien llama.

class codecs.CodecInfo(*encode*, *decode*, *streamreader=None*, *streamwriter=None*, *incrementalencoder=None*, *incrementaldecoder=None*, *name=None*)

Detalles de códec al buscar el registro de códec. Los argumentos del constructor se almacenan en atributos del mismo nombre:

name

El nombre de la codificación.

encode
decode

Las funciones de codificación y decodificación sin estado. Deben ser funciones o métodos que tengan la misma interfaz que los métodos `encode()` y `decode()` de instancias de *Codec* (ver Codec Interface). Se espera que las funciones o métodos funcionen en modo sin estado.

incrementalencoder
incrementaldecoder

Clases de codificación y decodificación incremental o funciones de fábrica. Deben proporcionar la interfaz definida por las clases base `IncrementalEncoder` y `IncrementalDecoder`, respectivamente. Los códecs incrementales pueden mantener el estado.

streamwriter
streamreader

Las clases *stream*, tanto *writer* como *reader* o funciones de fábrica. Estos tienen que proporcionar la interfaz definida por las clases base `StreamWriter` y `StreamReader`, respectivamente. Los códecs de flujo pueden mantener el estado.

Para simplificar el acceso a los diversos componentes de códec, el módulo proporciona estas funciones adicionales que utilizan `lookup()` para la búsqueda de códec:

codecs.getencoder(*encoding*)

Busca el códec para la codificación dada y retorna su función de codificador.

Lanza un `LookupError` en caso de que no se encuentre la codificación.

codecs.getdecoder(*encoding*)

Busca el códec para la codificación dada y retorna su función de decodificador.

Lanza un `LookupError` en caso de que no se encuentre la codificación.

codecs.getincrementalencoder(*encoding*)

Busca el códec para la codificación dada y retorna su clase de codificador incremental o función de fábrica.

Lanza un `LookupError` en caso de que no se encuentre la codificación o el códec no admita un codificador incremental.

codecs.getincrementaldecoder(*encoding*)

Busca el códec para la codificación dada y retorna su clase de decodificador incremental o función de fábrica.

Lanza un `LookupError` en caso de que no se encuentre la codificación o el códec no admita un decodificador incremental.

codecs.getreader(*encoding*)

Busca el códec para la codificación dada y retorna su clase `StreamReader` o función de fábrica.

Lanza un `LookupError` en caso de que no se encuentre la codificación.

codecs.getwriter(*encoding*)

Busca el códec para la codificación dada y retorna su clase `StreamWriter` o función de fábrica.

Lanza un `LookupError` en caso de que no se encuentre la codificación.

Los códecs personalizados se ponen a disposición registrando una función de búsqueda de códecs adecuada:

codecs.register(*search_function*, */*)

Registra una función de búsqueda de códec. Se espera que las funciones de búsqueda tomen un argumento, que sea el nombre de codificación en minúsculas con guiones y espacios convertidos a guiones bajos, y que retorne un objeto `CodecInfo`. En caso de que una función de búsqueda no pueda encontrar una codificación dada, debería retornar `None`.

Distinto en la versión 3.9: Guiones y espacios se convierten a guiones bajos.

codecs.unregister(*search_function*, */*)

Anula el registro de una función de búsqueda de códecs y elimina el caché del registro. Si la función de búsqueda no está registrada, no hace nada.

Added in version 3.10.

Mientras que la función incorporada `open()` y el módulo asociado `io` son el enfoque recomendado para trabajar con archivos de texto codificados, este módulo proporciona funciones y clases de utilidad adicionales que permiten el uso de una gama más amplia de códecs cuando se trabaja con archivos binarios:

codecs.open(*filename*, *mode='r'*, *encoding=None*, *errors='strict'*, *buffering=-1*)

Abre un archivo codificado utilizando el modo dado y retorna una instancia de `StreamReaderWriter`, proporcionando codificación/decodificación transparente. El modo de archivo predeterminado es `'r'`, que significa abrir el archivo en modo de lectura.

Nota

Si el valor de *encoding* no es `None`, entonces, los archivos codificados subyacentes siempre se abren en modo binario. No se realiza ninguna conversión automática de `'\n'` al leer y escribir. El argumento *mode* puede ser cualquier modo binario aceptable para la función integrada `open()`; la `'b'` se añade automáticamente.

encoding especifica la codificación que se utilizará para el archivo. Se permite cualquier codificación que codifique y decodifique desde bytes, y los tipos de datos admitidos por los métodos de archivo dependen del códec utilizado.

pueden proporcionarse *errors* para definir el manejo de errores. El valor predeterminado es `'estricto'`, lo que hace que se genere un `ValueError` en caso de que ocurra un error de codificación.

buffering tiene el mismo significado que para la función incorporada `open()`. Su valor predeterminado es -1, lo que significa que se utilizará el tamaño predeterminado del búfer.

Distinto en la versión 3.11: El modo `'U'` ha sido eliminado.

Obsoleto desde la versión 3.14: `codecs.open()` has been superseded by `open()`.

codecs.EncodedFile(*file*, *data_encoding*, *file_encoding=None*, *errors='strict'*)

Retorna una instancia de `StreamRecoder`, una versión envuelta de *file* que proporciona transcodificación transparente. El archivo original se cierra cuando se cierra la versión empaquetada.

Los datos escritos en el archivo empaquetado se decodifican de acuerdo con la *data_encoding* dada y luego se escriben en el archivo original como bytes usando *file_encoding*. Los bytes leídos del archivo original se decodifican según *file_encoding*, y el resultado se codifica utilizando *data_encoding*.

Si no se proporciona *file_encoding*, el valor predeterminado es *data_encoding*.

Pueden proporcionarse *errors* para definir el manejo de errores. Su valor predeterminado es `'estricto'`, lo que hace que se genere **`ValueError`** en caso de que ocurra un error de codificación.

codecs.iterencode(*iterator*, *encoding*, *errors='strict'*, ***kwargs*)

Uses an incremental encoder to iteratively encode the input provided by *iterator*. *iterator* must yield **`str`** objects. This function is a generator. The *errors* argument (as well as any other keyword argument) is passed through to the incremental encoder.

Esta función requiere que el códec acepte texto en objetos **`str`** para codificar. Por lo tanto, no admite codificadores de bytes a bytes, como `base64_codec`.

codecs.iterdecode(*iterator*, *encoding*, *errors='strict'*, ***kwargs*)

Uses an incremental decoder to iteratively decode the input provided by *iterator*. *iterator* must yield **`bytes`** objects. This function is a generator. The *errors* argument (as well as any other keyword argument) is passed through to the incremental decoder.

Esta función requiere que el códec acepte objetos **`bytes`** para decodificar. Por lo tanto, no admite codificadores de texto a texto como `rot_13`, aunque `rot_13` puede usarse de manera equivalente con **`iterencode()`**.

codecs.readbuffer_encode(*buffer*, *errors=None*, */*)

Return a **`tuple`** containing the raw bytes of *buffer*, a buffer-compatible object or **`str`** (encoded to UTF-8 before processing), and their length in bytes.

The *errors* argument is ignored.

```
>>> codecs.readbuffer_encode(b"Zito")
(b'Zito', 4)
```

El módulo también proporciona las siguientes constantes que son útiles para leer y escribir en archivos dependientes de la plataforma:

codecs.BOM
codecs.BOM_BE
codecs.BOM_LE
codecs.BOM_UTF8
codecs.BOM_UTF16
codecs.BOM_UTF16_BE
codecs.BOM_UTF16_LE
codecs.BOM_UTF32
codecs.BOM_UTF32_BE
codecs.BOM_UTF32_LE

> Estas constantes definen varias secuencias de bytes, que son marcas de orden de bytes Unicode (BOM) para varias codificaciones. Se utilizan en flujos de datos UTF-16 y UTF-32 para indicar el orden de bytes utilizado, y en UTF-8 como firma Unicode. `BOM_UTF16` es `BOM_UTF16_BE` o `BOM_UTF16_LE` dependiendo del orden de bytes nativo de la plataforma, `BOM` es un alias para **`BOM_UTF16`**, `BOM_LE` para **`BOM_UTF16_LE`** y `BOM_BE` para **`BOM_UTF16_BE`**. Los otros representan la lista de materiales en las codificaciones UTF-8 y UTF-32.

Clases Base de Códec

The `codecs` module defines a set of base classes which define the interfaces for working with codec objects, and can also be used as the basis for custom codec implementations.

Cada códec tiene que definir cuatro interfaces para que pueda usarse como códec en Python: codificador sin estado, decodificador sin estado, lector de flujo y escritor de flujo. El lector de flujo y los escritores suelen reutilizar el codificador/decodificador sin estado para implementar los protocolos de archivo. Los autores de códecs también necesitan definir cómo manejará los errores de codificación y decodificación.

Manejadores de errores

Para simplificar y estandarizar el manejo de errores, los códecs pueden implementar diferentes esquemas de manejo de errores aceptando el argumento de cadena *errors*:

```
>>> 'German ß, ♬'.encode(encoding='ascii', errors='backs
b'German \\xdf, \\u266c'
>>> 'German ß, ♬'.encode(encoding='ascii', errors='xmlch
b'German &#223;, &#9836;'
```

Los siguientes manejadores de errores se pueden emplear con todos los códecs Python Codificaciones estándar:

Valor	Significado
`'strict'`	Lanza un `UnicodeError` (o una subclase), este es el valor predeterminado. Implementado en **`strictly_errors()`**.
`'ignore'`	Ignore los datos mal formados y continúe sin previo aviso. Implementado en `ignore_errors()`.
`'replace'`	Sustituir con un marcador de reemplazo. Al codificar, emplear `?` (carácter ASCII). Al decodificar, usar � (U+FFFD, el CARÁCTER DE REEMPLAZO oficial). Implementado en `replace_errors()`.
`'backslashreplace'`	Reemplazar con secuencias de escape mediante barra invertida. Al codificar, emplear la forma hexadecimal del punto de código Unicode con los formatos `\xhh` `\uxxxx` `\Uxxxxxxxx`. Al decodificar, usa la forma hexadecimal del valor del byte con el formato `\xhh`. Implementado en `backslashreplace_errors()`.
`'surrogateescape'`	En la decodificación, reemplace el byte con código sustituto individual que va desde `U+DC80` a `U+DCFF`. Este código se volverá a convertir en el mismo byte cuando se use el manejador de errores `'sustituto de paisaje'` al codificar los datos. (Ver **PEP 383** para más información).

Los siguientes manejadores de errores solo son aplicables a codificaciones de texto:

Valor	Significado
`'xmlcharrefreplace'`	Reemplazar con una referencia de carácter numérico XML/HTML, que es una forma decimal del punto de código Unicode con formato `&#num;`. Implementado en `xmlcharrefreplace_errors()`.
`'namereplace'`	Reemplazar con secuencias de escape `\N{...}`, lo que aparece entre llaves, es la propiedad Nombre de la Base de datos de

Caracteres Unicode. Implementado en **`namereplace_errors()`**.

Además, el siguiente manejador de errores es específico de los códecs dados:

Valor	Códecs	Significado
`'surrogatepass'`	utf-8, utf-16, utf-32, utf-16-be, utf-16-le, utf-32-be, utf-32-le	Permite la codificación y decodificación del punto de código sustituto (`U+D800 - U+DFFF`) como punto de código normal. De lo contrario, estos códecs tratan la presencia de un punto de código sustituto en **`str`** como un error.

Added in version 3.1: Los manejadores de errores `'surrogateescape'` y `'surrogatepass'`.

Distinto en la versión 3.4: Los manejadores de errores `'surrogatepass'` ahora funcionan con los códecs utf-16* y utf-32*.

Added in version 3.5: El manejador de errores `'namereplace'`.

Distinto en la versión 3.5: El manejador de errores `'backslashreplace'` ahora funciona con decodificación y traducción.

El conjunto de valores permitidos puede ampliarse registrando un nuevo manejador de errores con nombre:

codecs.register_error(*name*, *error_handler*, */*)

Registre la función de manejo de errores *error_handler* bajo el nombre *name*. Se invocará el argumento *error_handler* durante la codificación y decodificación en caso de error, cuando *name* se especifica como el parámetro de errores.

Para la codificación, se llamará a *error_handler* con una instancia **`UnicodeEncodeError`**, que contiene información sobre la ubicación del error. El manejador de errores debe generar esta o una excepción diferente, o retornar una tupla con un reemplazo para la parte no codificable de la entrada y una posición donde la codificación debe continuar. El reemplazo puede ser **`str`** o **`bytes`**. Si el reemplazo son bytes, el codificador simplemente los copiará en el búfer de salida. Si el reemplazo es una cadena de caracteres, el codificador codificará el reemplazo. La codificación continúa en la entrada original en la posición especificada. Los valores de posición negativos se tratarán como relativos al final de la cadena de entrada. Si la posición resultante está fuera del límite, se lanzará **`IndexError`**.

La decodificación y la traducción funcionan de manera similar, excepto que **`UnicodeDecodeError`** o **`UnicodeTranslateError`** se pasarán al manejador y el sustituto del manejador de errores se colocará directamente en la salida.

Los manejadores de errores registrados previamente (incluidos los manejadores de error estándar) se pueden buscar por nombre:

codecs.lookup_error(*name*, /)

Retorna el manejador de errores previamente registrado con el nombre *name*.

Lanza un **`LookupError`** en caso de que no se pueda encontrar el controlador.

Los siguientes manejadores de errores estándar también están disponibles como funciones de nivel de módulo:

codecs.strict_errors(*exception*)

Implementa el manejo de errores `'strict'`.

Cada error de codificación o decodificación genera un **`UnicodeError`**.

codecs.ignore_errors(*exception*)

Implementa el manejo de errores `'ignore'`.

Los datos con formato incorrecto se ignoran; la codificación o decodificación continúa sin previo aviso.

codecs.replace_errors(*exception*)

Implementa el manejo de errores `'replace'`.

Sustituye `?` (carácter ASCII) por errores de codificación o � (U+FFFD, el CARÁCTER DE REEMPLAZO oficial) por errores de decodificación.

codecs.backslashreplace_errors(*exception*)

Implementa el manejador de errores `'backslashreplace'`.

Los datos con formato incorrecto se reemplazan por una secuencia de escape con barra invertida. Al codificar, emplea la forma hexadecimal del punto de código Unicode con los formatos `\xhh` `\uxxxx` `\Uxxxxxxxx`. Al decodificar, usa la forma hexadecimal del valor del byte con el formato `\xhh`.

Distinto en la versión 3.5: Funciona con la decodificación y traducción.

codecs.xmlcharrefreplace_errors(*exception*)

Implementa el manejador de errores `'xmlcharrefreplace'` (solo para codificar dentro de text encoding).

El carácter no codificable se reemplaza por una referencia de carácter numérico XML/HTML adecuada, que es una forma decimal del punto de código Unicode con formato `&#num;`.

codecs.namereplace_errors(*exception*)

Implementa el manejo de errores `'namereplace` (solo para codificar dentro de text encoding).

El carácter no codificable se reemplaza por una secuencia de escape `\N{...}`. El conjunto de caracteres que aparecen entre llaves es la propiedad Nombre de la Base de datos de Caracteres Unicode. Por ejemplo, la letra minúscula alemana `'ß'` se convertirá en la secuencia de bytes `\N{LATIN SMALL LETTER SHARP S}`.

Added in version 3.5.

Codificación y decodificación sin estado

La clase base `Codec` define estos métodos que también definen las interfaces de función del codificador y decodificador sin estado:

class codecs.Codec

encode(*input*, *errors='strict'*)

Codifica el objeto *input* y retorna una tupla (objeto de salida, longitud consumida). Por ejemplo text encoding convierte un objeto de cadena de caracteres en un objeto de bytes utilizando una codificación de juego de caracteres particular (por ejemplo,``cp1252`` o `iso-8859-1`).

El argumento *errors* define el manejo de errores a aplicar. El valor predeterminado es el manejo `estricto`.

Es posible que el método no almacene estado en la instancia `Codec`. Use `StreamWriter` para códecs que deben mantener el estado para que la codificación sea eficiente.

El codificador debe poder manejar la entrada de longitud cero y retornar un objeto vacío del tipo de objeto de salida en esta situación.

decode(*input*, *errors='strict'*)

Decodifica el objeto *input* y retorna una tupla (objeto de salida, longitud consumida). Por ejemplo, para un codificación de texto, la decodificación convierte un objeto de bytes codificado usando una codificación de juego de caracteres particular en un objeto de cadena de caracteres.

Para codificaciones de texto y códecs de bytes a bytes, *input* debe ser un objeto de bytes o uno que proporcione la interfaz de búfer de solo lectura, por ejemplo, objetos de búfer y archivos mapeados en memoria.

El argumento *errors* define el manejo de errores a aplicar. El valor predeterminado es el manejo `estricto`.

Es posible que el método no almacene estado en la instancia de `Codec`. Use `StreamReader` para códecs que deben mantener el estado para que la decodificación sea eficiente.

El decodificador debe poder manejar la entrada de longitud cero y retornar un objeto vacío del tipo de objeto de salida en esta situación.

Codificación y decodificación incrementales

Las clases `IncrementalEncoder` y `IncrementalDecoder` proporcionan la interfaz básica para la codificación y decodificación incrementales. La codificación/decodificación de la entrada no se realiza con una llamada a la función de codificador/decodificador sin estado, sino con varias llamadas al método `encode()`/ `decode()` del codificador incremental /decodificador. El codificador/decodificador incremental realiza un seguimiento del proceso de codificación/decodificación durante las llamadas a métodos.

La salida combinada de las llamadas al método `encode()`/ `decode()` es el mismo que si todas las entradas individuales se unieran en una, y esta entrada se codificara/decodificara con codificador/decodificador sin estado.

Objetos IncrementalEncoder

La clase `IncrementalEncoder` se usa para codificar una entrada en varios pasos. Define los siguientes métodos que cada codificador incremental debe definir para ser compatible con el registro de códec Python.

class codecs.IncrementalEncoder(*errors='strict'*)

Constructor para una clase instancia de `IncrementalEncoder`.

Todos los codificadores incrementales deben proporcionar esta interfaz de constructor. Son libres de agregar argumentos de palabras clave adicionales, pero el registro de códecs de Python solo utiliza los definidos aquí.

La clase `IncrementalEncoder` puede implementar diferentes esquemas de manejo de errores al proporcionar el argumento de palabra clave *errors*. Ver Manejadores de errores para posibles valores.

El argumento *errors* se asignará a un atributo del mismo nombre. La asignación a este atributo hace posible cambiar entre diferentes estrategias de manejo de errores durante la vida útil del objeto `IncrementalEncoder`.

encode(*object*, *final=False*)

Codifica *object* (teniendo en cuenta el estado actual del codificador) y retorna el objeto codificado resultante. Si esta es la última llamada a `encode()` *final* debe ser verdadero (el valor predeterminado es falso).

reset()

Restablece el codificador al estado inicial. La salida se descarta: llama a `.encode(object, final=True)`, pasando un byte vacío o una cadena de texto si es necesario, para restablecer el codificador y obtener la salida.

getstate()

Retorna el estado actual del codificador que debe ser un número entero. La implementación debe asegurarse de que `0` sea el estado más común. (Los estados que son más complicados que los enteros se pueden convertir en un entero al empaquetar/serializar el estado y codificar los bytes de la cadena resultante en un entero).

setstate(*state*)

Establece el estado del codificador en *state*. *state* debe ser un estado de codificador retornado por `getstate()`.

Objetos IncrementalDecoder

La clase `IncrementalDecoder` se usa para decodificar una entrada en varios pasos. Define los siguientes métodos que cada decodificador incremental debe definir para ser compatible con el registro de códec Python.

class codecs.IncrementalDecoder(*errors='strict'*)

Constructor para una instancia de `IncrementalDecoder`.

Todos los decodificadores incrementales deben proporcionar esta interfaz de constructor. Son libres de agregar argumentos de palabras clave adicionales, pero el registro de códecs de Python solo utiliza los definidos aquí.

La clase `IncrementalDecoder` puede implementar diferentes esquemas de manejo de errores al proporcionar el argumento de palabra clave *errors*. Ver Manejadores de errores para posibles valores.

El argumento *errors* se asignará a un atributo del mismo nombre. La asignación a este atributo hace posible cambiar entre diferentes estrategias de manejo de errores durante la vida útil del objeto `IncrementalDecoder`.

decode(*object*, *final=False*)

Decodifica *object* (teniendo en cuenta el estado actual del decodificador) y retorna el objeto decodificado resultante. Si esta es la última llamada a `decode()` *final* debe ser verdadero (el valor predeterminado es falso). Si *final* es verdadero, el decodificador debe decodificar la entrada por completo y debe vaciar todos los búferes. Si esto no es posible (por ejemplo, debido a secuencias de bytes incompletas al final de la entrada), debe iniciar el manejo de errores al igual que en el caso sin estado (lo que podría generar una excepción).

reset()

Restablece el decodificador al estado inicial.

getstate()

Retorna el estado actual del decodificador. Debe ser una tupla con dos elementos, el primero debe ser el búfer que contiene la entrada aún sin codificar. El segundo debe ser un número entero y puede ser información de estado adicional. (La implementación debe asegurarse de que `0` sea la información de estado adicional más común). Si esta información de estado adicional es `0`, debe ser posible establecer el decodificador en el estado que no tiene entrada almacenada y `0` como información de estado adicional, de modo que alimentar la entrada previamente almacenada en el búfer al decodificador la retorna al estado anterior sin producir ninguna salida. (La información de estado adicional que es más complicada que los enteros se puede convertir en un entero al empaquetar/serializar la información y codificar los bytes de la cadena resultante en un entero).

setstate(*state*)

Establezca el estado del decodificador en *state*. *state* debe ser un estado de decodificador retornado por `getstate()`.

Codificación y decodificación de flujos

The `StreamWriter` and `StreamReader` classes provide generic working interfaces which can be used to implement new encoding submodules very easily. See **`encodings.utf_8`** for an example of how this is done.

Objetos StreamWriter

La clase `StreamWriter` es una subclase de `Codec` y define los siguientes métodos que cada escritor del flujo debe definir para ser compatible con el registro de códecs Python.

class codecs.StreamWriter(*stream*, *errors='strict'*)

Constructor para una instancia de `StreamWriter`.

Todos los escritores de flujos deben proporcionar esta interfaz de constructor. Son libres de agregar argumentos de palabras clave adicionales, pero el registro de códecs de Python solo utiliza los definidos aquí.

El argumento *stream* debe ser un objeto tipo archivo abierto para escribir texto o datos binarios, según corresponda para el códec específico.

La clase `StreamWriter` puede implementar diferentes esquemas de manejo de errores al proporcionar el argumento de palabra clave *errors*. Consulte Manejadores de errores para ver los manejadores de errores estándar que puede admitir el códec de flujo subyacente.

El argumento *errors* se asignará a un atributo del mismo nombre. La asignación a este atributo hace posible cambiar entre diferentes estrategias de manejo de errores durante la vida útil del objeto `StreamWriter`.

write(*object*)

Escribe el contenido del objeto codificado en el flujo.

writelines(*list*)

Escribe una lista concatenada de cadenas en el flujo (posiblemente reutilizando el método `write()`). No se admiten iterables infinitos o muy grandes. Los códecs estándar de bytes a bytes no admiten este método.

reset()

Restablece los búfers de códec utilizados para mantener el estado interno.

Llamar a este método debería garantizar que los datos en la salida se pongan en un estado limpio que permita agregar datos nuevos sin tener que volver a escanear todo el flujo para recuperar el estado.

Además de los métodos anteriores, la clase `StreamWriter` también debe heredar todos los demás métodos y atributos del flujo subyacente.

Objetos StreamReader

La clase `StreamReader` es una subclase de `Codec` y define los siguientes métodos que cada lector de flujo debe definir para ser compatible con el registro de códecs de Python.

class codecs.StreamReader(*stream*, *errors='strict'*)

Constructor para una instancia de `StreamReader`.

Todos los lectores de flujo deben proporcionar esta interfaz de constructor. Son libres de agregar argumentos de palabras clave adicionales, pero el registro de códecs de Python solo utiliza los definidos aquí.

El argumento *stream* debe ser un objeto tipo archivo abierto para leer texto o datos binarios, según corresponda para el códec específico.

La clase `StreamReader` puede implementar diferentes esquemas de manejo de errores al proporcionar el argumento de palabra clave *errors*. Consulte Manejadores de errores para ver los manejadores de errores estándar que puede admitir el códec de flujo subyacente.

El argumento *errors* se asignará a un atributo del mismo nombre. La asignación a este atributo hace posible cambiar entre diferentes estrategias de manejo de errores durante la vida útil del objeto `StreamReader`.

El conjunto de valores permitidos para el argumento *errors* se puede ampliar con `register_error()`.

read(*size=-1*, *chars=-1*, *firstline=False*)

Decodifica datos del flujo y retorna el objeto resultante.

El argumento *chars* indica el número de puntos de código decodificados o bytes a retornar. El método `read()` nunca retornará más datos de los solicitados, pero podría retornar menos, si no hay suficientes disponibles.

El argumento *size* indica el número máximo aproximado de bytes codificados o puntos de código para leer para la decodificación. El decodificador puede modificar esta configuración según corresponda. El valor predeterminado -1 indica leer y decodificar tanto como sea posible. Este parámetro está diseñado para evitar tener que decodificar archivos grandes en un solo paso.

La bandera *firstline* indica que sería suficiente retornar solo la primera línea, si hay errores de decodificación en las líneas posteriores.

El método debe usar una estrategia de lectura codiciosa, lo que significa que debe leer la mayor cantidad de datos permitidos dentro de la definición de la codificación y el tamaño dado, por ejemplo si las terminaciones de codificación opcionales o los marcadores de estado están disponibles en la transmisión, también deben leerse.

readline(*size=None*, *keepends=True*)

Lee una línea del flujo de entrada y retorna los datos decodificados.

size, si se da, se pasa como argumento de tamaño al método `read()` del *stream*.

Si *keepends* es falso, las terminaciones de línea se eliminarán de las líneas retornadas.

readlines(*sizehint=None*, *keepends=True*)

Lee todas las líneas disponibles en el flujo de entrada y las retorna como una lista de líneas.

Los finales de línea se implementan utilizando el método `decode()` del códec y se incluyen en las entradas de la lista si *keepends* es verdadero.

sizehint, si se proporciona, se pasa como argumento *size* al método `read()` del *stream*.

reset()

Restablece los búfers de códec utilizados para mantener el estado interno.

Tenga en cuenta que ningún reposicionamiento de flujo debe suceder. Este método está destinado principalmente a poder recuperarse de errores de decodificación.

Además de los métodos anteriores, la clase `StreamReader` también debe heredar todos los demás métodos y atributos del flujo subyacente.

Objetos StreamReaderWriter

La clase `StreamReaderWriter` es una clase de conveniencia que permite envolver flujos que funcionan tanto en modo de lectura como de escritura.

El diseño es tal que uno puede usar las funciones de fábrica retornadas por la función `lookup()` para construir la instancia.

class codecs.StreamReaderWriter(*stream*, *Reader*, *Writer*, *errors='strict'*)

Crea una instancia de `StreamReaderWriter`. *stream* debe ser un objeto similar a un archivo. *Reader* y *Writer* deben ser funciones o clases de fábrica que proporcionen la interfaz `StreamReader` y `StreamWriter` respectivamente. El manejo de errores se realiza de la misma manera que se define para los lectores y escritores de flujos.

Las instancias `StreamReaderWriter` definen las interfaces combinadas de `StreamReader` y clases `StreamWriter`. Heredan todos los demás métodos y atributos del flujo subyacente.

Objetos StreamRecoder

La clase `StreamRecoder` traduce datos de una codificación a otra, lo que a veces es útil cuando se trata de diferentes entornos de codificación.

El diseño es tal que uno puede usar las funciones de fábrica retornadas por la función `lookup()` para construir la instancia.

class codecs.StreamRecoder(*stream*, *encode*, *decode*, *Reader*, *Writer*, *errors='strict'*)

Creates a **`StreamRecoder`** instance which implements a two-way conversion: *encode* and *decode* work on the frontend — the data visible to code calling `read()` and `write()`, while *Reader* and *Writer* work on the backend — the data in *stream*.

Puede usar estos objetos para realizar transcodificaciones transparentes, por ejemplo, de Latin-1 a UTF-8 y viceversa.

El argumento *stream* debe ser un objeto similar a un archivo.

Los argumentos *encode* y *decode* deben cumplir con la interfaz de `Codec`. *Reader* y *Writer* deben ser funciones o clases de fábrica que proporcionen objetos de la interfaz `StreamReader` y `StreamWriter` respectivamente.

El manejo de errores se realiza de la misma manera que se define para los lectores y escritores de flujos.

las instancias `StreamRecoder` definen las interfaces combinadas de las clases `StreamReader` y `StreamWriter`. Heredan todos los demás métodos y atributos del flujo subyacente.

Codificaciones y Unicode

Las cadenas de caracteres se almacenan internamente como secuencias de puntos de código en el rango `U+0000`–`U+10FFFF`. (Consultar **PEP 393** [https://peps.python.org/pep-0393/] para obtener más detalles sobre la implementación). Una vez utilizado un objeto de cadena de caracteres fuera de la CPU y de la memoria, la *endianness* y cómo se almacenan estas matrices como bytes se convierte en un problema. Al igual que con otros códecs, la serialización de una cadena en una secuencia de bytes se conoce como codificación, y la recreación de la cadena a partir de los bytes se conoce como decodificación. Hay múltiples códecs para la serialización de texto, a los que se puede consultar colectivamente mediante text encodings.

La codificación de texto más simple (llamada `'latin-1'` o `'iso-8859-1'`) asigna los puntos de código 0–255 a los bytes `0x0` – `0xff`, lo que significa que un objeto de cadena de caracteres que contiene puntos de código encima de `U+00FF` no se puede codificar con este códec. Al hacerlo, lanzará un **`UnicodeEncodeError`** que se parece a lo siguiente (aunque los detalles del mensaje de error pueden diferir): `UnicodeEncodeError: 'latin-1' codec can't encode character '\u1234' 'in position 3: ordinal not in range(256).`

Hay otro grupo de codificaciones (las llamadas codificaciones de mapa de caracteres) que eligen un subconjunto diferente de todos los puntos de código Unicode y cómo estos puntos de código se asignan a los bytes `0x0` – `0xff`. Para ver cómo se hace esto, simplemente abra, por ejemplo `encodings/cp1252.py` (que es una codificación que se usa principalmente en Windows). Hay una cadena constante con 256 caracteres que le muestra qué carácter está asignado a qué valor de byte.

All of these encodings can only encode 256 of the 1114112 code points defined in Unicode. A simple and straightforward way that can store each Unicode code point, is to store each code point as four consecutive bytes. There are two possibilities: store the bytes in big endian or in little endian order. These two encodings are called

`UTF-32-BE` and `UTF-32-LE` respectively. Their disadvantage is that if, for example, you use `UTF-32-BE` on a little endian machine you will always have to swap bytes on encoding and decoding. Python's `UTF-16` and `UTF-32` codecs avoid this problem by using the platform's native byte order when no BOM is present. Python follows prevailing platform practice, so native-endian data round-trips without redundant byte swapping, even though the Unicode Standard defaults to big-endian when the byte order is unspecified. When these bytes are read by a CPU with a different endianness, the bytes have to be swapped. To be able to detect the endianness of a `UTF-16` or `UTF-32` byte sequence, a BOM («Byte Order Mark») is used. This is the Unicode character `U+FEFF`. This character can be prepended to every `UTF-16` or `UTF-32` byte sequence. The byte swapped version of this character (`0xFFFE`) is an illegal character that may not appear in a Unicode text. When the first character of a `UTF-16` or `UTF-32` byte sequence is `U+FFFE`, the bytes have to be swapped on decoding.

Unfortunately the character `U+FEFF` had a second purpose as a `ZERO WIDTH NO-BREAK SPACE`: a character that has no width and doesn't allow a word to be split. It can e.g. be used to give hints to a ligature algorithm. With Unicode 4.0 using `U+FEFF` as a `ZERO WIDTH NO-BREAK SPACE` has been deprecated (with `U+2060` (`WORD JOINER`) assuming this role). Nevertheless Unicode software still must be able to handle `U+FEFF` in both roles: as a BOM it's a device to determine the storage layout of the encoded bytes, and vanishes once the byte sequence has been decoded into a string; as a `ZERO WIDTH NO-BREAK SPACE` it's a normal character that will be decoded like any other.

Hay otra codificación que puede codificar el rango completo de caracteres Unicode: UTF-8. UTF-8 es una codificación de 8 bits, lo que significa que no hay problemas con el orden de bytes en UTF-8. Cada byte en una secuencia de bytes UTF-8 consta de dos partes: bits marcadores (los bits más significativos) y bits de carga útil. Los bits marcadores son una secuencia de cero a cuatro bits `1` seguidos de un bit `0`. Los caracteres Unicode se codifican de esta manera (con x siendo bits de carga útil, que cuando se concatenan dan el carácter Unicode):

Rango	Codificación
`U-00000000 ... U-0000007F`	0xxxxxxx
`U-00000080 ... U-000007FF`	110xxxxx 10xxxxxx
`U-00000800 ... U-0000FFFF`	1110xxxx 10xxxxxx 10xxxxxx
`U-00010000 ... U-0010FFFF`	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

El bit menos significativo del carácter Unicode es el bit x más a la derecha.

Como UTF-8 es una codificación de 8 bits, no se requiere una lista de materiales y cualquier carácter `U+FEFF` en la cadena decodificada (incluso si es el primer carácter) se trata como un `ESPACIO SIN QUIEBRE DE ANCHO CERO` (``*ZERO WIDTH NO-BREAK SPACE*``).

Sin información externa, es imposible determinar de manera fidedigna qué codificación se utilizó para codificar una cadena de caracteres. Cada codificación de mapa de caracteres puede decodificar cualquier secuencia de bytes aleatoria. Sin embargo, eso no es posible con UTF-8, ya que las secuencias de bytes UTF-8 tienen una estructura que no permite secuencias de bytes arbitrarias. Para aumentar la confiabilidad con la que se puede detectar una codificación UTF-8, Microsoft inventó una variante de UTF-8 (que Python 2.5 llama `"utf-8-sig"`) para su programa Bloc de notas: Antes de que cualquier carácter Unicode sea escrito en un archivo, se escribe un BOM codificado en UTF-8 (que se muestra como una secuencia de bytes: `0xef`, `0xbb`, `0xbf`). Como es bastante improbable que cualquier archivo codificado del mapa de caracteres comience con estos valores de bytes (que, por ejemplo, se asignarían a

> LETRA LATINA PEQUEÑA I CON DIAERESIS
> SEÑALADO A LA DERECHA DE DOBLE ÁNGULO
> MARCA DE CITA
> SIGNO DE PREGUNTA INVERTIDO

en iso-8859-1), esto aumenta la probabilidad de que una codificación `utf-8-sig` pueda adivinarse correctamente a partir de la secuencia de bytes. Por lo tanto, aquí la lista de materiales no se utiliza para poder determinar el orden de bytes utilizado para generar la secuencia de bytes, sino como una firma que ayuda a adivinar la codificación. Al codificar, el códec utf-8-sig escribirá `0xef`, `0xbb`, `0xbf` como los primeros tres bytes del archivo. Al decodificar, `utf-8-sig` omitirá esos tres bytes si aparecen como los primeros tres bytes en el archivo. En UTF-8, se desaconseja el uso de la lista de materiales y, en general, debe evitarse.

Codificaciones estándar

Python comes with a number of codecs built-in, either implemented as C functions or with dictionaries as mapping tables. The following table lists the codecs by name, together with a few common aliases, and the languages for which the encoding is likely used. Neither the list of aliases nor the list of languages is meant to be exhaustive. Notice that spelling alternatives that only differ in case or use a hyphen instead of an underscore are also valid aliases because they are equivalent when normalized by `normalize_encoding()`. For example, `'utf-8'` is a valid alias for the `'utf_8'` codec.

Nota

The below table lists the most common aliases, for a complete list refer to the source aliases.py [https://github.com/python/cpython/tree/3.14/Lib/encodings/aliases.py] file.

On Windows, `cpXXX` codecs are available for all code pages. But only codecs listed in the following table are guarantead to exist on other platforms.

Algunas codificaciones comunes pueden omitir la maquinaria de búsqueda de códecs para mejorar el rendimiento. CPython solo reconoce estas oportunidades de optimización para un conjunto limitado de alias (sin distinción entre mayúsculas y minúsculas): utf-8, utf8, latin-1, latin1, iso-8859-1, iso8859-1, mbcs (solo Windows), ascii, us-ascii, utf-16, utf16, utf-32, utf32, y lo mismo usando guiones bajos en lugar de guiones. El uso de alias alternativos para estas codificaciones puede resultar en una ejecución más lenta.

Distinto en la versión 3.6: Oportunidad de optimización reconocida para us-ascii.

Muchos de los juegos de caracteres admiten los mismos idiomas. Varían en caracteres individuales (por ejemplo, si el SIGNO EURO es compatible o no), y en la asignación de caracteres para codificar posiciones. Para los idiomas europeos en particular, generalmente existen las siguientes variantes:

- un conjunto de códigos ISO 8859
- una página de códigos de Microsoft Windows, que generalmente se deriva de un conjunto de códigos 8859, pero reemplaza los caracteres de control con caracteres gráficos adicionales
- una página de códigos EBCDIC de IBM
- una página de códigos de IBM PC, que es compatible con ASCII

Códec	**Aliases**	**Lenguajes**
ascii	646, us-ascii	Inglés
big5	big5-tw, csbig5	Chino Tradicional
big5hkscs	big5-hkscs, hkscs	Chino Tradicional
cp037	IBM037, IBM039	Inglés
cp273	273, IBM273, csIBM273	Alemán *Added in version 3.4.*
cp424	EBCDIC-CP-HE, IBM424	Hebreo
cp437	437, IBM437	Inglés
cp500	EBCDIC-CP-BE, EBCDIC-CP-CH, IBM500	Europa Occidental
cp720		Árabe
cp737		Griego
cp775	IBM775	Lenguajes bálticos
cp850	850, IBM850	Europa Occidental
cp852	852, IBM852	Europa central y del este
cp855	855, IBM855	Bulgarian, Macedonian, Russian, Serbian
cp856		Hebreo
cp857	857, IBM857	Turco
cp858	858, IBM858	Europa Occidental
cp860	860, IBM860	Portugués
cp861	861, CP-IS, IBM861	Islandés
cp862	862, IBM862	Hebreo
cp863	863, IBM863	Canadiense

cp864	IBM864	Árabe
cp865	865, IBM865	Danés, Noruego
cp866	866, IBM866	Ruso
cp869	869, CP-GR, IBM869	Griego
cp874		Tailandés
cp875		Griego
cp932	932, ms932, mskanji, ms-kanji, windows-31j	Japonés
cp949	949, ms949, uhc	Coreano
cp950	950, ms950	Chino Tradicional
cp1006		Urdu
cp1026	ibm1026	Turco
cp1125	1125, ibm1125, cp866u, ruscii	Ucraniano *Added in version 3.4.*
cp1140	ibm1140	Europa Occidental
cp1250	windows-1250	Europa central y del este
cp1251	windows-1251	Belarusian, Bulgarian, Macedonian, Russian, Serbian
cp1252	windows-1252	Europa Occidental
cp1253	windows-1253	Griego
cp1254	windows-1254	Turco
cp1255	windows-1255	Hebreo
cp1256	windows-1256	Árabe
cp1257	windows-1257	Lenguajes bálticos
cp1258	windows-1258	Vietnamita
euc_jp	eucjp, ujis, u-jis	Japonés
euc_jis_2004	jisx0213, eucjis2004	japonés
euc_jisx0213	eucjisx0213	Japonés
euc_kr	euckr, korean, ksc5601, ks_c-5601, ks_c-5601-1987, ksx1001, ks_x-1001	Coreano
gb2312	chinese, csiso58gb231280, euc-cn, euccn, eucgb2312-cn, gb2312-1980, gb2312-80, iso-ir-58	Chino simplificado
gbk	936, cp936, ms936	Chino unificado
gb18030	gb18030-2000	Chino unificado
hz	hzgb, hz-gb, hz-gb-2312	Chino simplificado
iso2022_jp	csiso2022jp, iso2022jp, iso-2022-jp	Japonés
iso2022_jp_1	iso2022jp-1, iso-2022-jp-1	Japonés
iso2022_jp_2	iso2022jp-2, iso-2022-jp-2	Japonés, coreano, Chino simplificado, Europa occidental, Griego
iso2022_jp_2004	iso2022jp-2004, iso-2022-jp-2004	Japonés
iso2022_jp_3	iso2022jp-3, iso-2022-jp-3	Japonés
iso2022_jp_ext	iso2022jp-ext, iso-2022-jp-ext	Japonés
iso2022_kr	csiso2022kr, iso2022kr, iso-2022-kr	Coreano

latin_1	iso-8859-1, iso8859-1, 8859, cp819, latin, latin1, L1	Europa Occidental
iso8859_2	iso-8859-2, latin2, L2	Europa central y del este
iso8859_3	iso-8859-3, latin3, L3	Esperanto, Maltés
iso8859_4	iso-8859-4, latin4, L4	Northern Europe
iso8859_5	iso-8859-5, cyrillic	Belarusian, Bulgarian, Macedonian, Russian, Serbian
iso8859_6	iso-8859-6, arabic	Árabe
iso8859_7	iso-8859-7, greek, greek8	Griego
iso8859_8	iso-8859-8, hebrew	Hebreo
iso8859_9	iso-8859-9, latin5, L5	Turco
iso8859_10	iso-8859-10, latin6, L6	Lenguajes nórdicos
iso8859_11	iso-8859-11, thai	Lenguajes tailandeses
iso8859_13	iso-8859-13, latin7, L7	Lenguajes bálticos
iso8859_14	iso-8859-14, latin8, L8	Lenguajes Celtas
iso8859_15	iso-8859-15, latin9, L9	Europa Occidental
iso8859_16	iso-8859-16, latin10, L10	Europa suroriental
johab	cp1361, ms1361	coreano
koi8_r		Ruso
koi8_t		Tayiko *Added in version 3.5.*
koi8_u		Ucraniano
kz1048	kz_1048, strk1048_2002, rk1048	Kazajo *Added in version 3.5.*
mac_cyrillic	maccyrillic	Belarusian, Bulgarian, Macedonian, Russian, Serbian
mac_greek	macgreek	Griego
mac_iceland	maciceland	Islandés
mac_latin2	maclatin2, maccentraleurope, mac_centeuro	Europa central y del este
mac_roman	macroman, macintosh	Europa Occidental
mac_turkish	macturkish	Turco
ptcp154	csptcp154, pt154, cp154, cyrillic-asian	Kazajo
shift_jis	csshiftjis, shiftjis, sjis, s_jis	Japonés
shift_jis_2004	shiftjis2004, sjis_2004, sjis2004	Japonés
shift_jisx0213	shiftjisx0213, sjisx0213, s_jisx0213	Japonés
utf_32	U32, utf32	todos los lenguajes
utf_32_be	UTF-32BE	todos los lenguajes
utf_32_le	UTF-32LE	todos los lenguajes
utf_16	U16, utf16	todos los lenguajes
utf_16_be	UTF-16BE	todos los lenguajes
utf_16_le	UTF-16LE	todos los lenguajes
utf_7	U7, unicode-1-1-utf-7	todos los lenguajes
utf_8	U8, UTF, utf8, cp65001	todos los lenguajes

utf_8_sig		todos los lenguajes

Distinto en la versión 3.4: Los codificadores utf-16* y utf-32* ya no permiten codificar puntos de código sustitutos (`U+D800` – `U+DFFF`). Los decodificadores utf-32* ya no decodifican secuencias de bytes que corresponden a puntos de código sustituto.

Distinto en la versión 3.8: `cp65001` ahora es un alias de `utf_8`.

Distinto en la versión 3.14: On Windows, `cpXXX` codecs are now available for all code pages.

Codificaciones específicas de Python

Varios códecs predefinidos son específicos de Python, por lo que sus nombres de códec no tienen significado fuera de Python. Estos se enumeran en las tablas a continuación según los tipos de entrada y salida esperados (tenga en cuenta que si bien las codificaciones de texto son el caso de uso más común para los códecs, la infraestructura de códecs subyacente admite transformaciones de datos arbitrarias en lugar de solo codificaciones de texto). Para los códecs asimétricos, el significado indicado describe la dirección de codificación.

Codificaciones de texto

Los siguientes códecs proporcionan codificación de **`str`** a **`bytes`** y decodificación de bytes-like object a **`str`**, similar a las codificaciones de texto Unicode.

Códec	Alias	Significado
idna		Implementar **RFC 3490**, ver también **`encodings.idna`**. Solo se admite `errors='strict'`.
mbcs	ansi, dbcs	Solo Windows: codifique el operando de acuerdo con la página de códigos ANSI (CP_ACP).
oem		Solo Windows: codifique el operando de acuerdo con la página de códigos OEM (CP_OEMCP). *Added in version 3.6.*
palmos		Codificación de PalmOS 3.5.
punycode		Implementar **RFC 3492**. Los códecs con estado no son

	compatibles.
raw_unicode_escape	Codificación Latin-1 con `\uXXXX` y `\UXXXXXXXX` para otros puntos de código. Las barras invertidas existentes no se escapan de ninguna manera. Se usa en el protocolo Python *pickle*.
undefined	This Codec should only be used for testing purposes. Lanza una excepción para todas las conversiones, incluso cadenas vacías. El manejador de errores se ignora.
unicode_escape	Codificación adecuada como contenido de un literal Unicode en código fuente Python codificado en ASCII, excepto que no se escapan las comillas. Decodificar desde el código fuente Latin-1. Tenga en cuenta que el código fuente de Python realmente usa UTF-8 por defecto.

Distinto en la versión 3.8: Se elimina el códec «unicode_internal».

Transformaciones Binarias

Los siguientes códecs proporcionan transformaciones binarias: mapeos de bytes-like object a **`bytes`**. No son compatibles con **`bytes.decode()`** (que solo produce **`str`** de salida).

Códec	Alias	Significado	Codificador / decodificador
base64_codec	base64, base_64	Convierte el operando a MIME base64 multilínea (el resultado siempre incluye un `'\n'` final). *Distinto en la versión 3.4:* acepta cualquier bytes-like object como entrada para codificar y decodificar	**`base64.encodebytes()`** / **`base64.decodebytes()`**
bz2_codec	bz2	Comprime el operando usando bz2.	**`bz2.compress()`** / **`bz2.decompress()`**
hex_codec	hex	Convierte el operando en representación hexadecimal, con dos dígitos por byte.	**`binascii.b2a_hex()`** / **`binascii.a2b_hex()`**
quopri_codec	quopri, quotedprintable, quoted_printable	Convierte el operando a MIME quoted printable.	**`quopri.encode()`** / **`quopri.decode()`**
uu_codec	uu	Convierte el operando usando uuencode.	
zlib_codec	zip, zlib	Comprime el operando usando gzip.	**`zlib.compress()`** / **`zlib.decompress()`**

Además de objetos similares a bytes, `'base64_codec'` también acepta instancias solo ASCII de **`str`** para decodificación

Added in version 3.2: Restauración de las transformaciones binarias.

Distinto en la versión 3.4: Restauración de los alias para las transformaciones binarias.

Standalone Codec Functions

The following functions provide encoding and decoding functionality similar to codecs, but are not available as named codecs through `codecs.encode()` or `codecs.decode()`. They are used internally (for example, by `pickle`) and behave similarly to the `string_escape` codec that was removed in Python 3.

codecs.escape_encode(*input*, *errors=None*)

Encode *input* using escape sequences. Similar to how `repr()` on bytes produces escaped byte values.

input must be a `bytes` object.

Returns a tuple `(output, length)` where *output* is a `bytes` object and *length* is the number of bytes consumed.

codecs.escape_decode(*input*, *errors=None*)

Decode *input* from escape sequences back to the original bytes.

input must be a bytes-like object.

Returns a tuple `(output, length)` where *output* is a `bytes` object and *length* is the number of bytes consumed.

Transformaciones de texto

El siguiente códec proporciona una transformación de texto: un mapeo de `str` a `str`. No es compatible con `str.encode()` (que solo produce `bytes` de salida).

Códec	**Alias**	**Significado**
rot_13	rot13	Retorna el cifrado César (*Caesar-cypher*) del operando.

Added in version 3.2: Restauración de la transformación de texto `rot_13`.

Distinto en la versión 3.4: Restauración del alias `rot13`.

`encodings` — Encodings package

This module implements the following functions:

encodings.normalize_encoding(*encoding*)

Normalize encoding name *encoding*.

Normalization works as follows: all non-alphanumeric characters except the dot used for Python package names are collapsed and replaced with a single underscore, leading and trailing underscores are removed. For example, `'  -;#'` becomes `'_'`.

Note that *encoding* should be ASCII only.

Nota

The following functions should not be used directly, except for testing purposes; `codecs.lookup()` should be used instead.

encodings.search_function(*encoding*)

Search for the codec module corresponding to the given encoding name *encoding*.

This function first normalizes the *encoding* using `normalize_encoding()`, then looks for a corresponding alias. It attempts to import a codec module from the encodings package using either the alias or the normalized name. If the module is found and defines a valid `getregentry()` function that returns a `codecs.CodecInfo` object, the codec is cached and returned.

If the codec module defines a `getaliases()` function any returned aliases are registered for future use.

encodings.win32_code_page_search_function(*encoding*)

Search for a Windows code page encoding *encoding* of the

form `cpXXXX`.

If the code page is valid and supported, return a **`codecs.CodecInfo`** object for it.

Availability: Windows.

Added in version 3.14.

This module implements the following exception:

exception encodings.CodecRegistryError

Raised when a codec is invalid or incompatible.

`encodings.idna` — Internationalized Domain Names in Applications

Este módulo implementa **RFC 3490** [https://datatracker.ietf.org/doc/html/rfc3490.html] (nombres de dominio internacionalizados en aplicaciones) y **RFC 3492** [https://datatracker.ietf.org/doc/html/rfc3492.html] (*Nameprep*: un perfil de *Stringprep* para nombres de dominio internacionalizados (IDN)). Se basa en la codificación `punycode` y **`stringprep`**.

If you need the IDNA 2008 standard from **RFC 5891** [https://datatracker.ietf.org/doc/html/rfc5891.html] and **RFC 5895** [https://datatracker.ietf.org/doc/html/rfc5895.html], use the third-party idna [https://pypi.org/project/idna/] module.

Estas RFC juntas definen un protocolo para admitir caracteres no ASCII en los nombres de dominio. Un nombre de dominio que contiene caracteres no ASCII (como `www.Alliancefrançaise.nu`) se convierte en una codificación compatible con ASCII (ACE, como `www.xn--alliancefranaise-npb.nu`). La forma ACE del nombre de dominio se utiliza en todos los lugares donde el protocolo no permite caracteres arbitrarios, como consultas DNS, campos HTTP *Host*, etc. Esta conversión se lleva a cabo en la aplicación; si es posible invisible para el usuario: la aplicación debe convertir de forma transparente las etiquetas de dominio Unicode a IDNA en el cable, y volver a convertir las etiquetas ACE a Unicode antes de presentarlas al usuario.

Python admite esta conversión de varias maneras: el códec `idna` realiza la conversión entre Unicode y ACE, separando una cadena de entrada en etiquetas basadas en los caracteres separadores definidos en la sección 3.1 de RFC 3490 **RFC 3490 Section 3.1** [https://datatracker.ietf.org/doc/html/rfc3490.html#section-3.1] y convertir cada etiqueta a ACE según sea necesario, y por el contrario, separar una cadena de bytes de entrada en etiquetas basadas en el separador `.` y convertir cualquier etiqueta ACE encontrada en unicode. Además, el módulo **`socket`** convierte de forma transparente los nombres de host Unicode a ACE, por lo que las aplicaciones no necesitan preocuparse por convertir los nombres de host ellos mismos cuando los pasan al módulo de socket. Además de eso, los módulos que tienen nombres de host como parámetros de función, como **`http.client`** y **`ftplib`**, aceptan nombres de host Unicode (**`http.client`** y luego también envían un mensaje transparente IDNA *hostname* en el campo *Host* si envía ese campo).

Al recibir nombres de host desde el cable (como en la búsqueda inversa de nombres), no se realiza una conversión automática a Unicode: las aplicaciones que deseen presentar dichos nombres de host al usuario deben decodificarlos en Unicode.

The module **`encodings.idna`** also implements the nameprep procedure, which performs certain normalizations on host names, to achieve case-insensitivity of international domain names, and to unify similar characters. The nameprep functions can be used directly if desired.

encodings.idna.nameprep(*label*)

> Retorna la versión pasada por *nameprep* (o versión *nameprepped*) de *label*. La implementación actualmente asume cadenas de caracteres de consulta, por lo que `AllowUnassigned` es verdadero.

encodings.idna.ToASCII(*label*)

> Convierte una etiqueta a ASCII, como se especifica en **RFC**

3490 [https://datatracker.ietf.org/doc/html/rfc3490.html]. Se supone que `UseSTD3ASCIIRules` es falso.

encodings.idna.ToUnicode(*label*)

Convierte una etiqueta a Unicode, como se especifica en **RFC 3490** [https://datatracker.ietf.org/doc/html/rfc3490.html].

`encodings.mbcs` — Windows ANSI codepage

Este módulo implementa la página de códigos ANSI (CP_ACP).

Availability: Windows.

Distinto en la versión 3.2: Antes de 3.2, se ignoraba el argumento *errors*; `'replace'` siempre se usó para codificar e `'ignore'` para decodificar.

Distinto en la versión 3.3: Admite cualquier manejador de errores.

`encodings.utf_8_sig` — UTF-8 codec with BOM signature

Este módulo implementa una variante del códec UTF-8. Al codificar, una lista de materiales codificada en UTF-8 se antepondrá a los bytes codificados en UTF-8. Para el codificador con estado esto solo se hace una vez (en la primera escritura en el flujo de bytes). En la decodificación, se omitirá una lista de materiales opcional codificada en UTF-8 al comienzo de los datos.

Tipos de datos

Los módulos descritos en este capítulo proporcionan una variedad de tipos de datos especializados, como fechas y horas, matrices de tipo fijo (*fixed-type arrays*), colas de montículos (*heap queues*), colas de doble extremo (*double-ended queues*) y enumeraciones.

Python también proporciona algunos tipos de datos integrados, concretamente `dict`, `list`, `set` y `frozenset`, y `tuple`. La clase `str` se utiliza para contener cadenas de caracteres Unicode, y las clases `bytes` y `bytearray` se utilizan para contener datos binarios.

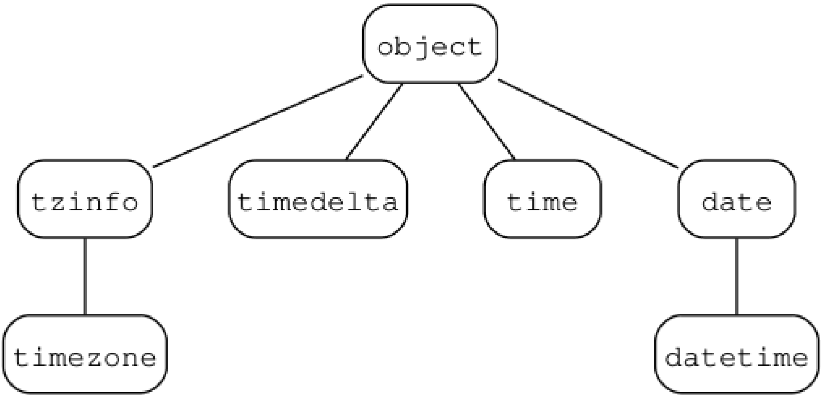
En este capítulo se documentan los siguientes módulos:

- `datetime` — Basic date and time types
    - Aware and naive objects
    - Constantes
    - Available types
        - Common properties
        - Determining if an object is aware or naive
    - `timedelta` objects
        - Examples of usage: `timedelta`
    - `date` objects
        - Examples of usage: `date`
    - `datetime` objects
        - Examples of usage: `datetime`
    - `time` objects

        - Examples of usage: `time`
    - `tzinfo` objects
    - `timezone` objects
    - `strftime()` and `strptime()` behavior
        - `strftime()` and `strptime()` format codes
        - Technical detail
- `zoneinfo` — IANA time zone support
    - Usando `ZoneInfo`
    - Fuentes de datos
        - Configurando los orígenes de datos
            - Configuración en tiempo de compilación
            - Configuración del entorno
            - Configuración de tiempo de ejecución
    - La clase `ZoneInfo`
        - Representaciones de cadenas
        - Serialización de Pickle
    - Funciones
    - Globales
    - Excepciones y advertencias
- `calendar` — General calendar-related functions
    - Command-line usage
- `collections` — Container datatypes
    - Objetos `ChainMap`
        - Ejemplos y recetas `ChainMap`
    - Objetos `Counter`
    - Objetos `deque`
        - Recetas `deque`

    - Objetos `defaultdict`
        - Ejemplos `defaultdict`
    - `namedtuple()` Funciones *Factory* para Tuplas y Campos con Nombres
    - Objetos `OrderedDict`
        - Ejemplos y recetas `OrderedDict`
    - Objetos `UserDict`
    - Objetos `UserList`
    - Objetos `UserString`
- `collections.abc` — Abstract Base Classes for Containers
    - Colecciones clases base abstractas
    - Colecciones Clases base abstractas - Descripciones detalladas
    - Ejemplos y Recetas
- `heapq` — Heap queue algorithm
    - Ejemplos Básicos
    - Other Applications
    - Notas de Aplicación de la Cola de Prioridades
    - Teoría
- `bisect` — Array bisection algorithm
    - Notas de rendimiento
    - Búsqueda en listas ordenadas
    - Ejemplos
- `array` — Efficient arrays of numeric values
- `weakref` — Weak references
    - Objetos de referencias débiles
    - Ejemplo
    - Objetos finalizadores
    - Comparando finalizadores con los métodos `__del__()`

- `types` — Dynamic type creation and names for built-in types
    - Creación dinámica de tipos
    - Tipos de Intérpretes Estándar
    - Clases y funciones de utilidad adicionales
    - Funciones de utilidad de corutina
- `copy` — Shallow and deep copy operations
- `pprint` — Data pretty printer
    - Functions
    - Objetos *PrettyPrinter*
    - Ejemplo
- `reprlib` — Alternate `repr()` implementation
    - Objetos Repr
    - Subclasificando Objetos Repr
- `enum` — Support for enumerations
    - Module contents
    - Data types
        - Nombres soportados `__dunder__`
        - Nombres `_sunder_` compatibles
    - Utilities and decorators
    - Notas
- `graphlib` — Functionality to operate with graph-like structures
    - Excepciones



[1]

[2] [3]

`datetime` — Basic date and time types

Código fuente: Lib/datetime.py [https://github.com/python/cpython/tree/3.14/Lib/datetime.py]

The `datetime` module supplies classes for manipulating dates and times.

Si bien la implementación permite operaciones aritméticas con fechas y horas, su principal objetivo es poder extraer campos de forma eficiente para su posterior manipulación o formateo.

Truco

Skip to the format codes.

Ver también

Módulo `calendar`
Funciones generales relacionadas a *calendar*.

Módulo `time`
Acceso a tiempo y conversiones.

Módulo `zoneinfo`
Zonas horarias concretas que representan la base de datos de zonas horarias de la IANA.

Paquete dateutil [https://dateutil.readthedocs.io/en/stable/]
Biblioteca de terceros con zona horaria ampliada y soporte de análisis.

Package DateType [https://pypi.org/project/DateType/]
Third-party library that introduces distinct static types to for example, allow static type checkers to differentiate between naive and aware datetimes.

Aware and naive objects

Date and time objects may be categorized as «aware» or «naive» depending on whether or not they include time zone information.

Con suficiente conocimiento de los ajustes de tiempo políticos y algorítmicos aplicables, como la zona horaria y la información del horario de verano, un objeto **consciente** puede ubicarse en relación con otros objetos conscientes. Un objeto consciente representa un momento específico en el tiempo que no está abierto a interpretación. [1]

A **naive** object does not contain enough information to unambiguously locate itself relative to other date/time objects. Whether a naive object represents Coordinated Universal Time (UTC), local time, or time in some other time zone is purely up to the program, just like it is up to the program whether a particular number represents metres, miles, or mass. Naive objects are easy to understand and to work with, at the cost of ignoring some aspects of reality.

For applications requiring aware objects, `datetime` and `time` objects have an optional time zone information attribute, `tzinfo`, that can be set to an instance of a subclass of the abstract `tzinfo`

class. These `tzinfo` objects capture information about the offset from UTC time, the time zone name, and whether daylight saving time is in effect.

Only one concrete `tzinfo` class, the `timezone` class, is supplied by the **`datetime`** module. The **`timezone`** class can represent simple time zones with fixed offsets from UTC, such as UTC itself or North American EST and EDT time zones. Supporting time zones at deeper levels of detail is up to the application. The rules for time adjustment across the world are more political than rational, change frequently, and there is no standard suitable for every application aside from UTC.

Constantes

The **`datetime`** module exports the following constants:

datetime.MINYEAR

> The smallest year number allowed in a `date` or `datetime` object. `MINYEAR` is 1.

datetime.MAXYEAR

> The largest year number allowed in a `date` or `datetime` object. `MAXYEAR` is 9999.

datetime.UTC

> Alias for the UTC time zone singleton `datetime.timezone.utc`.
>
> *Added in version 3.11.*

Available types

class datetime.date

> Una fecha naíf (*naive*) idealizada, suponiendo que el calendario gregoriano actual siempre estuvo, y siempre estará, vigente. Atributos: `year`, `month`, y `day`.

class datetime.time

Un tiempo idealizado, independiente de cualquier día en particular, suponiendo que cada día tenga exactamente 24* 60* 60 segundos. (Aquí no hay noción de «segundos intercalares».) Atributos: `hour`, `minute`, `second`, `microsecond`, y `tzinfo`.

class datetime.datetime

Una combinación de una fecha y una hora. Atributos: `year`, `month`, `day`, `hour`, `minute`, `second`, `microsecond` , y `tzinfo`.

class datetime.timedelta

A duration expressing the difference between two `datetime` or `date` instances to microsecond resolution.

class datetime.tzinfo

Una clase base abstracta para objetos de información de zona horaria. Estos son utilizados por las clases `datetime` y `time` para proporcionar una noción personalizable de ajuste de hora (por ejemplo, para tener en cuenta la zona horaria y / o el horario de verano).

class datetime.timezone

Una clase que implementa la clase de base abstracta `tzinfo` como un desplazamiento fijo desde el UTC.

Added in version 3.2.

Los objetos de este tipo son inmutables.

Subclass relationships:

Common properties

Las clases `date`, `datetime`, `time`, y `timezone` comparten estas características comunes:

- Los objetos de este tipo son inmutables.

- Objects of these types are hashable, meaning that they can be used as dictionary keys.
- Los objetos de este tipo admiten el *pickling* eficiente a través del módulo **`pickle`**.

Determining if an object is aware or naive

Los objetos del tipo **`date`** son siempre naíf (*naive*).

Un objeto de tipo **`time`** o **`datetime`** puede ser consciente (*aware*) o naíf (*naive*).

A **`datetime`** object `d` is aware if both of the following hold:

1. `d.tzinfo` no es `None`
2. `d.tzinfo.utcoffset(d)` no retorna `None`

Otherwise, `d` is naive.

A **`time`** object `t` is aware if both of the following hold:

1. `t.tzinfo` no es `None`
2. `t.tzinfo.utcoffset(None)` no retorna `None`.

Otherwise, `t` is naive.

La distinción entre los objetos consciente (*aware*) y naíf (*naive*) no se aplica a **`timedelta`**.

`timedelta` objects

A **`timedelta`** object represents a duration, the difference between two **`datetime`** or **`date`** instances.

class datetime.timedelta(*days=0*, *seconds=0*, *microseconds=0*, *milliseconds=0*, *minutes=0*, *hours=0*, *weeks=0*)

All arguments are optional and default to 0. Arguments may be integers or floats, and may be positive or negative.

Solo *days*, *seconds* y *microseconds* se almacenan internamente.

Los argumentos se convierten a esas unidades:

- Un milisegundo se convierte a 1000 microsegundos.
- Un minuto se convierte a 60 segundos.
- Una hora se convierte a 3600 segundos.
- Una semana se convierte a 7 días.

y los días, segundos y microsegundos se normalizan para que la representación sea única, con

- `0 <= microsegundos < 1000000`
- `0 <= segundos< 3600*24` (el número de segundos en un día)
- `-999999999 <= days <= 999999999`

El siguiente ejemplo ilustra cómo cualquier argumento además de *days*, *seconds* y *microseconds* se «fusionan» y normalizan en esos tres atributos resultantes:

```
>>> import datetime as dt
>>> delta = dt.timedelta(
...     days=50,
...     seconds=27,
...     microseconds=10,
...     milliseconds=29000,
...     minutes=5,
...     hours=8,
...     weeks=2
... )
>>> # Only days, seconds, and microseconds remain
>>> delta
datetime.timedelta(days=64, seconds=29156, microsec
```

Truco

`import datetime as dt` instead of `import datetime` or `from datetime import datetime` to avoid confusion between the module and the class. See How I Import Python's datetime Module [https://adamj.eu/tech/2019/09/12/how-i-import-pythons-datetime-module/].

Si algún argumento es flotante y hay microsegundos fraccionarios, los microsegundos fraccionarios que quedan de todos los argumentos se combinan y su suma se redondea al microsegundo más cercano utilizando el desempate de medio redondeo a par. Si ningún argumento es flotante, los procesos de conversión y normalización son exactos (no se pierde información).

Si el valor normalizado de los días se encuentra fuera del rango indicado, se lanza **`OverflowError`**.

Tenga en cuenta que la normalización de los valores negativos puede ser sorprendente al principio. Por ejemplo:

```
>>> import datetime as dt
>>> d = dt.timedelta(microseconds=-1)
>>> (d.days, d.seconds, d.microseconds)
(-1, 86399, 999999)
```

Since the string representation of **`timedelta`** objects can be confusing, use the following recipe to produce a more readable format:

```
>>> def pretty_timedelta(td):
...     if td.days >= 0:
...         return str(td)
...     return f'-({-td!s})'
...
>>> d = timedelta(hours=-1)
>>> str(d)  # not human-friendly
'-1 day, 23:00:00'
>>> pretty_timedelta(d)
'-(1:00:00)'
```

Atributos de clase:

timedelta.min

El objeto más negativo en **`timedelta`**, `timedelta(-999999999)`.

timedelta.max

El objeto más positivo de la **`timedelta`**, `timedelta(days=999999999, hours=23, minutes=59, seconds=59, microseconds=999999)`.

timedelta.resolution

La diferencia más pequeña posible entre los objetos no iguales **`timedelta`** `timedelta(microseconds=1)`.

Note that, because of normalization, `timedelta.max` is greater than `-timedelta.min`. `-timedelta.max` is not representable as a **`timedelta`** object.

Atributos de instancia (solo lectura):

timedelta.days

Between -999,999,999 and 999,999,999 inclusive.

timedelta.seconds

Between 0 and 86,399 inclusive.

Prudencia

It is a somewhat common bug for code to unintentionally use this attribute when it is actually intended to get a **`total_seconds()`** value instead:

```
>>> import datetime as dt
>>> duration = dt.timedelta(seconds=11235813)
>>> duration.days, duration.seconds
(130, 3813)
>>> duration.total_seconds()
11235813.0
```

timedelta.microseconds

Between 0 and 999,999 inclusive.

Operaciones soportadas:

Operación	Resultado
`t1 = t2 + t3`	Sum of `t2` and `t3`. Afterwards `t1 - t2 == t3` and `t1 - t3 == t2` are true. (1)
`t1 = t2 - t3`	Difference of `t2` and `t3`. Afterwards `t1 == t2 - t3` and `t2 == t1 + t3` are true. (1)(6)
`t1 = t2 * i o t1 = i * t2`	Delta multiplied by an integer. Afterwards `t1 // i == t2` is true, provided `i != 0`.
	In general, `t1  * i == t1 * (i-1) + t1` is true. (1)
`t1 = t2 * f o t1 = f * t2`	Delta multiplicado por un número decimal. El resultado se redondea al múltiplo mas cercano de *timedelta.resolution* usando redondeo de medio a par.
`f = t2 / t3`	Division (3) of overall duration `t2` by interval unit `t3`. Returns a **`float`** object.
`t1 = t2 / f o t1 = t2 / i`	Delta dividido por un número decimal o un entero. El resultado se redondea al múltiplo más cercano de *timedelta.resolution* usando redondeo de medio a par.
`t1 = t2 // i` o `t1 = t2 // t3`	El piso (*floor*) se calcula y el resto (si lo hay) se descarta. En el segundo caso, se retorna un entero. (3)
`t1 = t2 % t3`	El resto se calcula como un objeto **`timedelta`**. (3)
`q, r = divmod(t1, t2)`	Computes the quotient and the remainder: `q = t1 // t2` (3) and `r = t1 % t2`. `q` is an integer and `r` is a **`timedelta`** object.
`+t1`	Retorna un objeto **`timedelta`** con el mismo valor. (2)
`-t1`	Equivalent to `timedelta(-t1.days, -t1.seconds, -t1.microseconds)`, and to `t1 * -1`. (1)(4)
`abs(t)`	Equivalent to `+t` when `t.days >= 0`, and to `-t` when `t.days < 0`. (2)
`str(t)`	Retorna una cadena de caracteres en la forma `[D day[s], ][H]H:MM:SS[.UUUUUU]`, donde D es negativo para negativo `t`. (5)
`repr(t)`	Retorna una representación de cadena del objeto **`timedelta`** como una llamada de constructor con valores de atributos canónicos.

Notas:

1. Esto es exacto pero puede desbordarse.

2. Esto es exacto pero no puede desbordarse.

3. Division by zero raises **`ZeroDivisionError`**.

4. `-timedelta.max` is not representable as a **`timedelta`** object.

5. Las representaciones de cadena de caracteres de los objetos **`timedelta`** se normalizan de manera similar a su representación interna. Esto conduce a resultados algo inusuales para *timedeltas* negativos. Por ejemplo:

   ```
   >>> timedelta(hours=-5)
   datetime.timedelta(days=-1, seconds=68400)
   >>> print(_)
   -1 day, 19:00:00
   ```

6. La expresión `t2 - t3` siempre será igual a la expresión `t2 + (-t3)` excepto cuando *t3* es igual a `timedelta.max`; en ese caso, el primero producirá un resultado mientras que el segundo se desbordará.

Además de las operaciones enumeradas anteriormente, los objetos **`timedelta`** admiten ciertas sumas y restas con objetos **`date`** y **`datetime`** (ver más abajo).

Distinto en la versión 3.2: Floor division and true division of a **`timedelta`** object by another **`timedelta`** object are now supported, as are remainder operations and the **`divmod()`** function. True division and multiplication of a **`timedelta`** object by a **`float`** object are now supported.

`timedelta` objects support equality and order comparisons.

En contextos booleanos, un objeto **`timedelta`** se considera verdadero si y solo si no es igual a `timedelta (0)`.

Métodos de instancia:

timedelta.total_seconds()

Return the total number of seconds contained in the duration. Equivalent to `td / timedelta(seconds=1)`. For interval units other than seconds, use the division form directly (for example, `td / timedelta(microseconds=1)`).

Tenga en cuenta que para intervalos de tiempo muy largos (más de 270 años en la mayoría de las plataformas) este método perderá precisión de microsegundos.

Added in version 3.2.

Examples of usage: `timedelta`

Ejemplos adicionales de normalización:

```
>>> # Components of another_year add up to exactly 365
>>> import datetime as dt
>>> year = dt.timedelta(days=365)
>>> another_year = dt.timedelta(weeks=40, days=84, hours
...                             minutes=50, seconds=600)
>>> year == another_year
True
>>> year.total_seconds()
31536000.0
```

Ejemplos de `timedelta` aritmética:

```
>>> import datetime as dt
>>> year = dt.timedelta(days=365)
>>> ten_years = 10 * year
>>> ten_years
datetime.timedelta(days=3650)
>>> ten_years.days // 365
10
>>> nine_years = ten_years - year
>>> nine_years
datetime.timedelta(days=3285)
>>> three_years = nine_years // 3
>>> three_years, three_years.days // 365
(datetime.timedelta(days=1095), 3)
```

`date` objects

El objeto `date` representa una fecha (año, mes y día) en un calendario idealizado, el calendario gregoriano actual se extiende indefinidamente en ambas direcciones.

El 1 de enero del año 1 se llama día número 1, el 2 de enero del año 1 se llama día número 2, y así sucesivamente. [2]

class datetime.date(*year*, *month*, *day*)

Todos los argumentos son obligatorios. Los argumentos deben ser enteros, en los siguientes rangos:

- `MINYEAR <= year <= MAXYEAR`
- `1 <= month <= 12`
- `1 <= day <= number of days in the given month and year`

Si se proporciona un argumento fuera de esos rangos, **`ValueError`** se genera.

Otros constructores, todos los métodos de clase:

classmethod date.today()

Retorna la fecha local actual.

Esto es equivalente a `date.fromtimestamp(time.time())`.

classmethod date.fromtimestamp(*timestamp*)

Return the local date corresponding to the POSIX *timestamp*, such as is returned by **`time.time()`**.

Esto puede generar **`OverflowError`**, si la marca de tiempo está fuera del rango de valores admitidos por la plataforma *C* **`localtime()`**, y **`OSError`** en **`localtime()`** falla . Es común que esto se restrinja a años desde 1970 hasta 2038. Tenga en cuenta que en los sistemas que no son POSIX que incluyen segundos bisiestos en su noción de marca de tiempo, los segundos bisiestos son ignorados por **`fromtimestamp()`**.

Distinto en la versión 3.3: Se genera `OverflowError` en lugar de `ValueError` si la marca de tiempo está fuera del rango de valores admitidos por la plataforma *C* **`localtime()`**. Se genera `OSError` en lugar de **`ValueError`** cuando **`localtime()`**, falla.

classmethod date.fromordinal(*ordinal*)

Return the date corresponding to the proleptic Gregorian *ordinal*, where January 1 of year 1 has ordinal 1.

`ValueError` is raised unless `1 <= ordinal <= date.max.toordinal()`. For any date `d`, `date.fromordinal(d.toordinal()) == d`.

classmethod date.fromisoformat(*date_string*)

Return a `date` corresponding to a *date_string* given in any valid ISO 8601 format, with the following exceptions:

1. Reduced precision dates are not currently supported (`YYYY-MM`, `YYYY`).
2. Extended date representations are not currently supported (`±YYYYYY-MM-DD`).
3. Ordinal dates are not currently supported (`YYYY-OOO`).

Ejemplos:

```
>>> import datetime as dt
>>> dt.date.fromisoformat('2019-12-04')
datetime.date(2019, 12, 4)
>>> dt.date.fromisoformat('20191204')
datetime.date(2019, 12, 4)
>>> dt.date.fromisoformat('2021-W01-1')
datetime.date(2021, 1, 4)
```

Added in version 3.7.

Distinto en la versión 3.11: Anteriormente, este método solo admitía el formato `YYYY-MM-DD`.

classmethod date.fromisocalendar(*year*, *week*, *day*)

Return a `date` corresponding to the ISO calendar date specified by *year*, *week* and *day*. This is the inverse of the function `date.isocalendar()`.

Added in version 3.8.

classmethod date.strptime(*date_string*, *format*)

Return a `date` corresponding to *date_string*, parsed according to *format*. This is equivalent to:

```
date(*(time.strptime(date_string, format)[0:3]))
```

`ValueError` is raised if the date_string and format can't be parsed by `time.strptime()` or if it returns a value which isn't a time tuple. See also strftime() and strptime() behavior and `date.fromisoformat()`.

Nota

If *format* specifies a day of month without a year a `DeprecationWarning` is emitted. This is to avoid a quadrennial leap year bug in code seeking to parse only a month and day as the default year used in absence of one in the format is not a leap year. Such *format* values may raise an error as of Python 3.15. The workaround is to always include a year in your *format*. If parsing *date_string* values that do not have a year, explicitly add a year that is a leap year before parsing:

```
>>> import datetime as dt
>>> date_string = "02/29"
>>> when = dt.date.strptime(f"{date_string};1984",
>>> when.strftime("%B %d")
'February 29'
```

Added in version 3.14.

Atributos de clase:

date.min

La fecha representable más antigua, `date(MINYEAR, 1, 1)`.

date.max

La última fecha representable, `date(MAXYEAR, 12, 31)`.

date.resolution

La menor diferencia entre objetos de fecha no iguales, `timedelta(days=1)`.

Atributos de instancia (solo lectura):

date.year

Entre **`MINYEAR`** y **`MAXYEAR`** inclusive.

date.month

Entre 1 y 12 inclusive.

date.day

Entre 1 y el número de días en el mes dado del año dado.

Operaciones soportadas:

Operación	Resultado
`date2 = date1 + timedelta`	`date2` will be `timedelta.days` days after `date1`. (1)
`date2 = date1 - timedelta`	Computes `date2` such that `date2 + timedelta == date1`. (2)
`timedelta = date1 - date2`	(3)
`date1 == date2` `date1 != date2`	Equality comparison. (4)

`date1 < date2`
`date1 > date2`
`date1 <= date2`
`date1 >= date2`
	Order comparison. (5)

Notas:

1. *date2* se mueve hacia adelante en el tiempo si `timedelta.days > 0`, o hacia atrás si `timedelta.days < 0`. Después `date2 - date1 == timedelta. days`. `timedelta.seconds` y `timedelta.microseconds` se ignoran. **`OverflowError`** se lanza si `date2.year` sería menor que **`MINYEAR`** o mayor que **`MAXYEAR`**.

2. `timedelta.seconds` y `timedelta.microseconds` son ignorados.

3. This is exact, and cannot overflow. `timedelta.seconds` and `timedelta.microseconds` are 0, and `date2 + timedelta == date1` after.

4. **`date`** objects are equal if they represent the same date.

   `date` objects that are not also **`datetime`** instances are never equal to **`datetime`** objects, even if they represent the same date.

5. *date1* is considered less than *date2* when *date1* precedes *date2* in time. In other words, `date1 < date2` if and only if `date1.toordinal() < date2.toordinal()`.

   Order comparison between a **`date`** object that is not also a **`datetime`** instance and a **`datetime`** object raises **`TypeError`**.

Distinto en la versión 3.13: Comparison between **`datetime`** object and an instance of the **`date`** subclass that is not a **`datetime`** subclass no longer converts the latter to **`date`**, ignoring the time part and the time zone. The default behavior can be changed by overriding the special comparison methods in subclasses.

En contextos booleanos, todos los objetos **`date`** se consideran verdaderos.

Métodos de instancia:

date.replace(*year=self.year*, *month=self.month*, *day=self.day*)

Return a new **`date`** object with the same values, but with specified parameters updated.

Ejemplo:

```
>>> import datetime as dt
>>> d = dt.date(2002, 12, 31)
>>> d.replace(day=26)
datetime.date(2002, 12, 26)
```

The generic function `copy.replace()` also supports `date` objects.

date.timetuple()

Retorna una `time.struct_time` como la que retorna `time.localtime()`.

Las horas, minutos y segundos son 0, y el indicador DST es -1.

`d.timetuple()` es equivalente a:

```
time.struct_time((d.year, d.month, d.day, 0, 0, 0,
```

where `yday = d.toordinal() - date(d.year, 1, 1).toordinal() + 1` is the day number within the current year starting with 1 for January 1st.

date.toordinal()

Return the proleptic Gregorian ordinal of the date, where January 1 of year 1 has ordinal 1. For any `date` object `d`, `date.fromordinal(d.toordinal()) == d`.

date.weekday()

Retorna el día de la semana como un número entero, donde el lunes es 0 y el domingo es 6. Por ejemplo, `date(2002, 12, 4).weekday() == 2`, un miércoles. Ver también `isoweekday()`.

date.isoweekday()

Retorna el día de la semana como un número entero, donde el lunes es 1 y el domingo es 7. Por ejemplo, `date(2002, 12, 4).isoweekday() == 3`, un miércoles. Ver también **`weekday()`**, **`isocalendar()`**.

date.isocalendar()

Retorna un objeto named tuple con tres componentes: `year`, `week` y `weekday`.

El calendario ISO es una variante amplia utilizada del calendario gregoriano. [3]

El año ISO consta de 52 o 53 semanas completas, y donde una semana comienza un lunes y termina un domingo. La primera semana de un año ISO es la primera semana calendario (gregoriana) de un año que contiene un jueves. Esto se llama semana número 1, y el año ISO de ese jueves es el mismo que el año gregoriano.

Por ejemplo, 2004 comienza en jueves, por lo que la primera semana del año ISO 2004 comienza el lunes 29 de diciembre de 2003 y termina el domingo 4 de enero de 2004

```
>>> import datetime as dt
>>> dt.date(2003, 12, 29).isocalendar()
datetime.IsoCalendarDate(year=2004, week=1, weekday
>>> dt.date(2004, 1, 4).isocalendar()
datetime.IsoCalendarDate(year=2004, week=1, weekday
```

Distinto en la versión 3.9: El resultado cambió de una tupla a un named tuple.

date.isoformat()

Retorna una cadena de caracteres que representa la fecha en formato ISO 8601, `AAAA-MM-DD`:

```
>>> import datetime as dt
>>> dt.date(2002, 12, 4).isoformat()
'2002-12-04'
```

date.__str__()

For a date `d`, `str(d)` is equivalent to `d.isoformat()`.

date.ctime()

Retorna una cadena de caracteres que representa la fecha:

```
>>> import datetime as dt
>>> dt.date(2002, 12, 4).ctime()
'Wed Dec  4 00:00:00 2002'
```

`d.ctime()` es equivalente a:

```
time.ctime(time.mktime(d.timetuple()))
```

en plataformas donde la función nativa C **`ctime()`** (donde **`time.ctime()`** llama, pero que **`date.ctime()`** no se llama) se ajusta al estándar C.

date.strftime(*format*)

Return a string representing the date, controlled by an explicit format string. Format codes referring to hours, minutes or seconds will see 0 values. See also strftime() and strptime() behavior and **`date.isoformat()`**.

date.__format__(*format*)

Same as **`date.strftime()`**. This makes it possible to specify a format string for a **`date`** object in formatted string literals and when using **`str.format()`**. See also strftime() and strptime() behavior and **`date.isoformat()`**.

Examples of usage: `date`

Ejemplo de contar días para un evento:

```
>>> import time
>>> import datetime as dt
>>> today = dt.date.today()
>>> today
```

```
datetime.date(2007, 12, 5)
>>> today == dt.date.fromtimestamp(time.time())
True
>>> my_birthday = dt.date(today.year, 6, 24)
>>> if my_birthday < today:
...     my_birthday = my_birthday.replace(year=today.yea
...
>>> my_birthday
datetime.date(2008, 6, 24)
>>> time_to_birthday = abs(my_birthday - today)
>>> time_to_birthday.days
202
```

Más ejemplos de trabajo con `date`:

```
>>> import datetime as dt
>>> d = dt.date.fromordinal(730920) # 730920th day after
>>> d
datetime.date(2002, 3, 11)

>>> # Methods related to formatting string output
>>> d.isoformat()
'2002-03-11'
>>> d.strftime("%d/%m/%y")
'11/03/02'
>>> d.strftime("%A %d. %B %Y")
'Monday 11. March 2002'
>>> d.ctime()
'Mon Mar 11 00:00:00 2002'
>>> 'The {1} is {0:%d}, the {2} is {0:%B}.'.format(d, "c
'The day is 11, the month is March.'

>>> # Methods for extracting 'components' under differen
>>> t = d.timetuple()
>>> for i in t:
...     print(i)
2002                # year
3                   # month
11                  # day
```

```
0
0
0
0                   # weekday (0 = Monday)
70                  # 70th day in the year
-1
>>> ic = d.isocalendar()
>>> for i in ic:
...     print(i)
2002                # ISO year
11                  # ISO week number
1                   # ISO day number ( 1 = Monday )

>>> # A date object is immutable; all operations produce
>>> d.replace(year=2005)
datetime.date(2005, 3, 11)
```

`datetime` objects

El objeto `datetime` es un único objeto que contiene toda la información de un objeto `date` y un objeto `time`.

Like a `date` object, `datetime` assumes the current Gregorian calendar extended in both directions; like a `time` object, **`datetime`** assumes there are exactly 3600*24 seconds in every day.

Constructor:

class datetime.datetime(*year*, *month*, *day*, *hour=0*, *minute=0*, *second=0*, *microsecond=0*, *tzinfo=None*, ***, *fold=0*)

Se requieren los argumentos *year*, *month* y *day*. *tzinfo* puede ser `None`, o una instancia de una subclase `tzinfo`. Los argumentos restantes deben ser enteros en los siguientes rangos:

- `MINYEAR <= year <= MAXYEAR`,
- `1 <= month <= 12`,
- `1 <= day <= number of days in the given month and year`,

- `0 <= hour < 24`,
- `0 <= minute < 60`,
- `0 <= second < 60`,
- `0 <= microsecond < 1000000`,
- `fold in [0, 1]`.

Si se proporciona un argumento fuera de esos rangos, **`ValueError`** se genera.

Distinto en la versión 3.6: Added the *fold* parameter.

Otros constructores, todos los métodos de clase:

classmethod datetime.today()

Return the current local date and time, with **`tzinfo`** `None`.

Equivalente a:

```
datetime.fromtimestamp(time.time())
```

Ver también **`now()`**, **`fromtimestamp()`**.

Este método es funciona como **`now()`**, pero sin un parámetro `tz`.

classmethod datetime.now(*tz=None*)

Retorna la fecha y hora local actual.

Si el argumento opcional *tz* es `None` o no se especifica, es como **`today()`**, pero, si es posible, proporciona más precisión de la que se puede obtener al pasar por **`time.time()`** marca de tiempo (por ejemplo, esto puede ser posible en plataformas que suministran la función C **`gettimeofday()`**).

Si *tz* no es `None`, debe ser una instancia de una subclase **`tzinfo`**, y la fecha y hora actuales se convierten en la zona horaria de *tz*.

Esta función es preferible a **`today()`** y **`utcnow()`**.

Nota

Subsequent calls to **`datetime.now()`** may return the same instant depending on the precision of the underlying clock.

classmethod datetime.utcnow()

Retorna la fecha y hora UTC actual, con **`tzinfo`** `None`.

Esto es como **`now()`**, pero retorna la fecha y hora UTC actual, como un objeto naíf (*naive*): **`datetime`**. Se puede obtener una fecha y hora UTC actual consciente (*aware*) llamando a `datetime.now (timezone.utc)`. Ver también **`now()`**.

Advertencia

Debido a que los objetos naífs (*naive*) de `datetime` son tratados por muchos métodos de `datetime` como horas locales, se prefiere usar fechas y horas conscientes(*aware*) para representar las horas en UTC. Como tal, la forma recomendada de crear un objeto que represente la hora actual en UTC es llamando a `datetime.now(timezone.utc)`.

Obsoleto desde la versión 3.12: Use **`datetime.now()`** with **`UTC`** instead.

classmethod datetime.fromtimestamp(*timestamp*, *tz=None*)

Retorna la fecha y hora local correspondiente a la marca de tiempo POSIX, tal como la retorna **`time.time()`**. Si el argumento opcional *tz* es `None` o no se especifica, la marca de tiempo se convierte a la fecha y hora local de la plataforma, y el objeto retornado **`datetime`** es naíf (*naive*).

Si *tz* no es `None`, debe ser una instancia de una subclase **`tzinfo`**, y la fecha y hora actuales se convierten en la zona horaria de *tz*.

`fromtimestamp()` puede aumentar `OverflowError`, si la marca de tiempo está fuera del rango de valores admitidos por la plataforma *C* `localtime()` o `gmtime()`, y `OSError` en `localtime()` o `gmtime()` falla. Es común que esto se restrinja a los años 1970 a 2038. Tenga en cuenta que en los sistemas que no son POSIX que incluyen segundos bisiestos en su noción de marca de tiempo, los segundos bisiestos son ignorados por `fromtimestamp()`, y luego es posible tener dos marcas de tiempo que difieren en un segundo que producen objetos idénticos `datetime`. Se prefiere este método sobre `utcfromtimestamp()`.

Distinto en la versión 3.3: Se genera `OverflowError` en lugar de `ValueError` si la marca de tiempo está fuera del rango de valores admitidos por la plataforma *C* `localtime()` o `gmtime()`. genera `OSError` en lugar de la función `ValueError` en `localtime()` o error `gmtime()`.

Distinto en la versión 3.6: `fromtimestamp()` puede retornar instancias con `fold` establecido en 1.

classmethod datetime.utcfromtimestamp(*timestamp*)

Retorna el UTC `datetime` correspondiente a la marca de tiempo POSIX, con `tzinfo` `None`. (El objeto resultante es naíf (*naive*).)

Esto puede generar `OverflowError`, si la marca de tiempo está fuera del rango de valores admitidos por la plataforma C `gmtime()`, y error en `OSError` en `gmtime()`. Es común que esto se restrinja a los años entre1970 a 2038.

Para conocer un objeto `datetime`, llama a `fromtimestamp()`:

```
datetime.fromtimestamp(timestamp, timezone.utc)
```

En las plataformas compatibles con POSIX, es equivalente a la siguiente expresión:

```
datetime(1970, 1, 1, tzinfo=timezone.utc) + timedel
```

excepto que la última fórmula siempre admite el rango de años completo: entre `MINYEAR` y `MAXYEAR` inclusive.

Advertencia

Debido a que los objetos naíf (*naive*) de `datetime` son tratados por muchos métodos de `datetime` como horas locales, se prefiere usar fechas y horas conscientes para representar las horas en UTC. Como tal, la forma recomendada de crear un objeto que represente una marca de tiempo específica en UTC es llamando a `datetime.fromtimestamp(timestamp, tz=timezone.utc)`.

Distinto en la versión 3.3: Se genera `OverflowError` en lugar de `ValueError` si la marca de tiempo está fuera del rango de valores admitidos por la plataforma *C* **`gmtime()`**. genera `OSError` en lugar de **`ValueError`** en el error de **`gmtime()`**.

Distinto en la versión 3.15: Accepts any real number as *timestamp*, not only integer or float.

Obsoleto desde la versión 3.12: Use `datetime.fromtimestamp()` with `UTC` instead.

classmethod datetime.fromordinal(*ordinal*)

Se genera `datetime` correspondiente al ordinal del proléptico gregoriano, donde el 1 de enero del año 1 tiene ordinal 1. `ValueError` se genera a menos que `1 <= ordinal <= datetime.max.toordinal()`. La hora, minuto, segundo y microsegundo del resultado son todos 0, y `tzinfo` es `None`.

classmethod datetime.combine(*date, time, tzinfo=time.tzinfo*)

Return a new `datetime` object whose date components are equal to the given `date` object's, and whose time components are equal to the given `time` object's. If the *tzinfo* argument is provided, its value is used to set the `tzinfo` attribute of the result, otherwise the `tzinfo` attribute of the *time* argument is used. If the *date* argument is a `datetime` object, its time components and `tzinfo` attributes are ignored.

For any `datetime` object `d`, `d == datetime.combine(d.date(), d.time(), d.tzinfo)`.

Distinto en la versión 3.6: Se agregó el argumento *tzinfo*.

classmethod datetime.fromisoformat(*date_string*)

Devuelve un `datetime` correspondiente a un *date_string* en cualquier formato ISO 8601 válido, con las siguientes excepciones:

1. Las compensaciones de zona horaria pueden tener fracciones de segundo.
2. El separador `T` se puede reemplazar por cualquier carácter Unicode único.
3. No se admiten fracciones de horas y minutos.
4. Reduced precision dates are not currently supported (`YYYY-MM`, `YYYY`).
5. Extended date representations are not currently supported (`±YYYYYY-MM-DD`).
6. Ordinal dates are not currently supported (`YYYY-OOO`).

Ejemplos:

```
>>> import datetime as dt
>>> dt.datetime.fromisoformat('2011-11-04')
datetime.datetime(2011, 11, 4, 0, 0)
>>> dt.datetime.fromisoformat('20111104')
datetime.datetime(2011, 11, 4, 0, 0)
>>> dt.datetime.fromisoformat('2011-11-04T00:05:23'
datetime.datetime(2011, 11, 4, 0, 5, 23)
>>> dt.datetime.fromisoformat('2011-11-04T00:05:23Z
datetime.datetime(2011, 11, 4, 0, 5, 23, tzinfo=dat
>>> dt.datetime.fromisoformat('20111104T000523')
datetime.datetime(2011, 11, 4, 0, 5, 23)
```

```
>>> dt.datetime.fromisoformat('2011-W01-2T00:05:23.
datetime.datetime(2011, 1, 4, 0, 5, 23, 283000)
>>> dt.datetime.fromisoformat('2011-11-04 00:05:23.
datetime.datetime(2011, 11, 4, 0, 5, 23, 283000)
>>> dt.datetime.fromisoformat('2011-11-04 00:05:23.
datetime.datetime(2011, 11, 4, 0, 5, 23, 283000, tz
>>> dt.datetime.fromisoformat('2011-11-04T00:05:23+
datetime.datetime(2011, 11, 4, 0, 5, 23,
    tzinfo=datetime.timezone(datetime.timedelta(sec
```

Added in version 3.7.

Distinto en la versión 3.11: Previously, this method only supported formats that could be emitted by **`date.isoformat()`** or **`datetime.isoformat()`**.

classmethod datetime.fromisocalendar(*year*, *week*, *day*)

Return a **`datetime`** corresponding to the ISO calendar date specified by *year*, *week* and *day*. The non-date components of the datetime are populated with their normal default values. This is the inverse of the function **`datetime.isocalendar()`**.

Added in version 3.8.

classmethod datetime.strptime(*date_string*, *format*)

Retorna **`datetime`** correspondiente a *date_string*, analizado según *format*.

If *format* does not contain microseconds or time zone information, this is equivalent to:

```
datetime(*(time.strptime(date_string, format)[0:6]))
```

`ValueError` is raised if the date_string and format can't be parsed by **`time.strptime()`** or if it returns a value which isn't a time tuple. See also strftime() and strptime() behavior and **`datetime.fromisoformat()`**.

Distinto en la versión 3.13: If *format* specifies a day of month

without a year a `DeprecationWarning` is now emitted. This is to avoid a quadrennial leap year bug in code seeking to parse only a month and day as the default year used in absence of one in the format is not a leap year. Such *format* values may raise an error as of Python 3.15. The workaround is to always include a year in your *format*. If parsing *date_string* values that do not have a year, explicitly add a year that is a leap year before parsing:

```
>>> import datetime as dt
>>> date_string = "02/29"
>>> when = dt.datetime.strptime(f"{date_string};198
>>> when.strftime("%B %d")
'February 29'
```

Atributos de clase:

datetime.min

La primera fecha representable `datetime`, `datetime(MINYEAR, 1, 1, tzinfo=None)`.

datetime.max

La última fecha representable `datetime`, `datetime(MAXYEAR, 12, 31, 23, 59, 59, 999999, tzinfo=None)`.

datetime.resolution

La diferencia más pequeña posible entre objetos no iguales `datetime`, `timedelta(microseconds=1)`.

Atributos de instancia (solo lectura):

datetime.year

Entre `MINYEAR` y `MAXYEAR` inclusive.

datetime.month

Entre 1 y 12 inclusive.

datetime.day

Entre 1 y el número de días en el mes dado del año dado.

datetime.hour

En `range(24)`.

datetime.minute

En `range(60)`.

datetime.second

En `range(60)`.

datetime.microsecond

En `range(1000000)`.

datetime.tzinfo

El objeto pasó como argumento *tzinfo* al constructor **`datetime`**, o `None` si no se pasó ninguno.

datetime.fold

In `[0, 1]`. Used to disambiguate wall times during a repeated interval. (A repeated interval occurs when clocks are rolled back at the end of daylight saving time or when the UTC offset for the current zone is decreased for political reasons.) The values 0 and 1 represent, respectively, the earlier and later of the two moments with the same wall time representation.

Added in version 3.6.

Operaciones soportadas:

Operación	Resultado
`datetime2 = datetime1 + timedelta`	(1)
`datetime2 = datetime1 - timedelta`	(2)
`timedelta = datetime1 - datetime2`	(3)
`datetime1 == datetime2` `datetime1 != datetime2`	Equality comparison (4)

Order comparison. (5)

`datetime1 < datetime2`
`datetime1 > datetime2`
`datetime1 <= datetime2`
`datetime1 >= datetime2`

1. `datetime2` is a duration of `timedelta` removed from `datetime1`, moving forward in time if `timedelta.days > 0`, or backward if `timedelta.days < 0`. The result has the same **tzinfo** attribute as the input datetime, and `datetime2 - datetime1 == timedelta` after. **OverflowError** is raised if `datetime2.year` would be smaller than **MINYEAR** or larger than **MAXYEAR**. Note that no time zone adjustments are done even if the input is an aware object.

2. Computes the `datetime2` such that `datetime2 + timedelta == datetime1`. As for addition, the result has the same **tzinfo** attribute as the input datetime, and no time zone adjustments are done even if the input is aware.

3. Subtraction of a **datetime** from a **datetime** is defined only if both operands are naive, or if both are aware. If one is aware and the other is naive, **TypeError** is raised.

   If both are naive, or both are aware and have the same **tzinfo** attribute, the **tzinfo** attributes are ignored, and the result is a **timedelta** object `t` such that `datetime2 + t == datetime1`. No time zone adjustments are done in this case.

   If both are aware and have different **tzinfo** attributes, `a-b` acts as if `a` and `b` were first converted to naive UTC datetimes. The result is `(a.replace(tzinfo=None) - a.utcoffset()) - (b.replace(tzinfo=None) - b.utcoffset())` except that the implementation never overflows.

4. **datetime** objects are equal if they represent the same date and time, taking into account the time zone.

   Naive and aware **datetime** objects are never equal.

If both comparands are aware, and have the same **`tzinfo`** attribute, the **`tzinfo`** and **`fold`** attributes are ignored and the base datetimes are compared. If both comparands are aware and have different **`tzinfo`** attributes, the comparison acts as comparands were first converted to UTC datetimes except that the implementation never overflows. **`datetime`** instances in a repeated interval are never equal to **`datetime`** instances in other time zone.

5. *datetime1* is considered less than *datetime2* when *datetime1* precedes *datetime2* in time, taking into account the time zone.

   Order comparison between naive and aware **`datetime`** objects raises **`TypeError`**.

   If both comparands are aware, and have the same **`tzinfo`** attribute, the **`tzinfo`** and **`fold`** attributes are ignored and the base datetimes are compared. If both comparands are aware and have different **`tzinfo`** attributes, the comparison acts as comparands were first converted to UTC datetimes except that the implementation never overflows.

Distinto en la versión 3.3: Las comparaciones de igualdad entre las instancias conscientes (*aware*) y naíf (*naive*) **`datetime`** no generan **`TypeError`**.

Distinto en la versión 3.13: Comparison between **`datetime`** object and an instance of the **`date`** subclass that is not a **`datetime`** subclass no longer converts the latter to **`date`**, ignoring the time part and the time zone. The default behavior can be changed by overriding the special comparison methods in subclasses.

Métodos de instancia:

datetime.date()

Retorna el objeto **`date`** con el mismo año, mes y día.

datetime.time()

Retorna el objeto **`time`** con la misma hora, minuto, segundo, microsegundo y doblado(*fold*). **`tzinfo`** es `None`. Ver también método **`timetz()`**.

Distinto en la versión 3.6: El valor de plegado (*fold value*) se copia en el objeto **`time`** retornado.

datetime.timetz()

Retorna el objeto **`time`** con los mismos atributos de hora, minuto, segundo, microsegundo, pliegue y *tzinfo*. Ver también método **`time()`**.

Distinto en la versión 3.6: El valor de plegado (*fold value*) se copia en el objeto **`time`** retornado.

datetime.replace(*year=self.year*, *month=self.month*, *day=self.day*, *hour=self.hour*, *minute=self.minute*, *second=self.second*, *microsecond=self.microsecond*, *tzinfo=self.tzinfo*, ***, *fold=0*)

Return a new **`datetime`** object with the same attributes, but with specified parameters updated. Note that `tzinfo=None` can be specified to create a naive datetime from an aware datetime with no conversion of date and time data.

`datetime` objects are also supported by generic function **`copy.replace()`**.

Distinto en la versión 3.6: Added the *fold* parameter.

datetime.astimezone(*tz=None*)

Retorna un objeto **`datetime`** con el atributo nuevo **`tzinfo`** *tz*, ajustando los datos de fecha y hora para que el resultado sea la misma hora UTC que *self*, pero en hora local *tz*.

If provided, *tz* must be an instance of a **`tzinfo`** subclass, and its **`utcoffset()`** and **`dst()`** methods must not return `None`. If *self* is naive, it is presumed to represent time in the system time zone.

If called without arguments (or with `tz=None`) the system local time zone is assumed for the target time zone. The `.tzinfo` attribute of the converted datetime instance will be set to an instance of `timezone` with the zone name and offset obtained from the OS.

If `self.tzinfo` is *tz*, `self.astimezone(tz)` is equal to *self*: no adjustment of date or time data is performed. Else the result is local time in the time zone *tz*, representing the same UTC time as *self*: after `astz = dt.astimezone(tz)`, `astz - astz.utcoffset()` will have the same date and time data as `dt - dt.utcoffset()`.

If you merely want to attach a `timezone` object *tz* to a datetime *dt* without adjustment of date and time data, use `dt.replace(tzinfo=tz)`. If you merely want to remove the `timezone` object from an aware datetime *dt* without conversion of date and time data, use `dt.replace(tzinfo=None)`.

Tenga en cuenta que el método predeterminado `tzinfo.fromutc()` se puede reemplazar en una subclase `tzinfo` para afectar el resultado retornado por `astimezone()`. `astimezone()` ignora los casos de error, actúa como:

```
def astimezone(self, tz):
    if self.tzinfo is tz:
        return self
    # Convert self to UTC, and attach the new timez
    utc = (self - self.utcoffset()).replace(tzinfo=
    # Convert from UTC to tz's local time.
    return tz.fromutc(utc)
```

Distinto en la versión 3.3: tz ahora puede ser omitido.

Distinto en la versión 3.6: El método `astimezone()` ahora se puede invocar en instancias naíf (*naive*) que se supone representan la hora local del sistema.

datetime.utcoffset()

Si `tzinfo` es `None`, retorna `None`, de lo contrario retorna `self.tzinfo.utcoffset (self)`, y genera una excepción si este último no retorna `None` o un objeto **`timedelta`** con magnitud inferior a un día.

Distinto en la versión 3.7: El desfase UTC no está restringido a un número entero de minutos.

datetime.dst()

Si **`tzinfo`** es `None`, retorna `None`, de lo contrario retorna `self.tzinfo.utcoffset (self)`, y genera una excepción si este último no retorna `None` o un objeto **`timedelta`** con magnitud inferior a un día.

Distinto en la versión 3.7: El desfase DST no está restringido a un número entero de minutos.

datetime.tzname()

Si **`tzinfo`** es `None`, retorna `None`, de lo contrario retorna `self.tzinfo.tzname(self)`, genera una excepción si este último no retorna `None` o un objeto de cadena de caracteres,

datetime.timetuple()

Retorna una **`time.struct_time`** como la que retorna **`time.localtime()`**.

`d.timetuple()` es equivalente a:

```
time.struct_time((d.year, d.month, d.day,
                  d.hour, d.minute, d.second,
                  d.weekday(), yday, dst))
```

where `yday = d.toordinal() - date(d.year, 1, 1).toordinal() + 1` is the day number within the current year starting with 1 for January 1st. The **`tm_isdst`** flag of the result is set according to the **`dst()`** method: **`tzinfo`** is `None` or **`dst()`** returns `None`, **`tm_isdst`** is set to `-1`; else if **`dst()`** returns a non-zero value, **`tm_isdst`** is set to 1; else **`tm_isdst`** is set to 0.

datetime.utctimetuple()

If **`datetime`** instance `d` is naive, this is the same as `d.timetuple()` except that **`tm_isdst`** is forced to 0 regardless of what `d.dst()` returns. DST is never in effect for a UTC time.

If `d` is aware, `d` is normalized to UTC time, by subtracting `d.utcoffset()`, and a **`time.struct_time`** for the normalized time is returned. **`tm_isdst`** is forced to 0. Note that an **`OverflowError`** may be raised if `d.year` was `MINYEAR` or `MAXYEAR` and UTC adjustment spills over a year boundary.

> **Advertencia**
>
> Because naive `datetime` objects are treated by many `datetime` methods as local times, it is preferred to use aware datetimes to represent times in UTC; as a result, using **`datetime.utctimetuple()`** may give misleading results. If you have a naive `datetime` representing UTC, use `datetime.replace(tzinfo=timezone.utc)` to make it aware, at which point you can use **`datetime.timetuple()`**.

datetime.toordinal()

Retorna el ordinal gregoriano proléptico de la fecha. Lo mismo que `self.date().toordinal()`.

datetime.timestamp()

Retorna la marca de tiempo (*timestamp*) POSIX correspondiente a la instancia **`datetime`**. El valor de retorno es **`float`** similar al retornado por **`time.time()`**.

Naive **`datetime`** instances are assumed to represent local time and this method relies on the platform C **`mktime()`** function to perform the conversion. Since **`datetime`** supports wider range of values than **`mktime()`** on many platforms, this method may raise **`OverflowError`** or **`OSError`** for times far in the past or far in the future.

Para las instancias de **`datetime`**, el valor de retorno se calcula como:

```
(dt - datetime(1970, 1, 1, tzinfo=timezone.utc)).to
```

Nota

There is no method to obtain the POSIX timestamp directly from a naive **`datetime`** instance representing UTC time. If your application uses this convention and your system time zone is not set to UTC, you can obtain the POSIX timestamp by supplying `tzinfo=timezone.utc`:

```
timestamp = dt.replace(tzinfo=timezone.utc).timest
```

o calculando la marca de tiempo (*timestamp*) directamente:

```
timestamp = (dt - datetime(1970, 1, 1)) / timedelt
```

Added in version 3.3.

Distinto en la versión 3.6: El método **`timestamp()`** utiliza el atributo **`fold`** para desambiguar los tiempos durante un intervalo repetido.

datetime.weekday()

Retorna el día de la semana como un entero, donde el lunes es 0 y el domingo es 6. Lo mismo que `self.date().weekday()`. Ver también **`isoweekday()`**.

datetime.isoweekday()

Retorna el día de la semana como un número entero, donde el lunes es 1 y el domingo es 7. Lo mismo que `self.date().isoweekday()`. Ver también **`weekday()`**, **`isocalendar()`**.

datetime.isocalendar()

Retorna una named tuple con tres componentes: `year`, `week`, y `weekday`. Lo mismo que `self.date().isocalendar()`.

datetime.isoformat(*sep='T'*, *timespec='auto'*)

Retorna una cadena de caracteres representando la fecha y la hora en formato ISO 8601:

- `YYYY-MM-DDTHH:MM:SS.ffffff`, si **`microsecond`** no es 0
- `YYYY-MM-DDTHH:MM:SS`, si **`microsecond`** es 0

Si **`utcoffset()`** no retorna `None`, se agrega una cadena de caracteres dando el desplazamiento UTC:

- `YYYY-MM-DDTHH:MM:SS.ffffff+HH:MM[:SS[.ffffff]]`, si **`microsecond`** no es 0
- `YYYY-MM-DDTHH:MM:SS+HH:MM[:SS[.ffffff]]`, si **`microsecond`** es 0

Ejemplos:

```
>>> import datetime as dt
>>> dt.datetime(2019, 5, 18, 15, 17, 8, 132263).iso
'2019-05-18T15:17:08.132263'
>>> dt.datetime(2019, 5, 18, 15, 17, tzinfo=dt.time
'2019-05-18T15:17:00+00:00'
```

El argumento opcional *sep* (default `'T'`) es un separador de un carácter, ubicado entre las porciones de fecha y hora del resultado. Por ejemplo:

```
>>> import datetime as dt
>>> class TZ(dt.tzinfo):
...     """A time zone with an arbitrary, constant
...     def utcoffset(self, when):
...         return dt.timedelta(hours=-6, minutes=-
...
>>> dt.datetime(2002, 12, 25, tzinfo=TZ()).isoforma
'2002-12-25 00:00:00-06:39'
>>> dt.datetime(2009, 11, 27, microsecond=100, tzin
'2009-11-27T00:00:00.000100-06:39'
```

El argumento opcional *timespec* especifica el número de componentes adicionales del tiempo a incluir (el valor predeterminado es `'auto'`). Puede ser uno de los siguientes:

- `'auto'`: Igual que `'seconds'` si **`microsecond`** es 0, igual que `'microseconds'` de lo contrario.
- `'hours'`: incluye el **`hour`** en el formato de dos dígitos `HH`.
- `'minutes'`: Incluye **`hour`** y **`minute`** en formato `HH:MM`.
- `'seconds'`: Incluye **`hour`**, **`minute`**, y **`second`** en formato `HH:MM:SS`.
- `'milliseconds'`: Incluye tiempo completo, pero trunca la segunda parte fraccionaria a milisegundos. Formato `HH:MM:SS.sss`.
- `'microseconds'`: Incluye tiempo completo en formato `HH:MM:SS.ffffff`.

Nota

Los componentes de tiempo excluidos están truncados, no redondeados.

`ValueError` lanzará un argumento inválido *timespec*:

```
>>> import datetime as dt
>>> dt.datetime.now().isoformat(timespec='minutes')
'2002-12-25T00:00'
>>> my_datetime = dt.datetime(2015, 1, 1, 12, 30, 5
>>> my_datetime.isoformat(timespec='microseconds')
'2015-01-01T12:30:59.000000'
```

Distinto en la versión 3.6: Added the *timespec* parameter.

datetime.__str__()

For a **`datetime`** instance `d`, `str(d)` is equivalent to `d.isoformat(' ')`.

datetime.ctime()

Retorna una cadena de caracteres que representa la fecha y la hora:

```
>>> import datetime as dt
>>> dt.datetime(2002, 12, 4, 20, 30, 40).ctime()
'Wed Dec  4 20:30:40 2002'
```

La cadena de salida *no* incluirá información de zona horaria, independientemente de si la entrada es consciente (*aware*) o naíf (*naive*).

`d.ctime()` es equivalente a:

```
time.ctime(time.mktime(d.timetuple()))
```

en plataformas donde la función nativa C **`ctime()`** (que **`time.ctime()`** invoca, pero que **`datetime.ctime()`** no invoca) se ajusta al estándar *C*.

datetime.strftime(*format*)

Return a string representing the date and time, controlled by an explicit format string. See also strftime() and strptime() behavior and **`datetime.isoformat()`**.

datetime.__format__(*format*)

Same as **`datetime.strftime()`**. This makes it possible to specify a format string for a **`datetime`** object in formatted string literals and when using **`str.format()`**. See also strftime() and strptime() behavior and **`datetime.isoformat()`**.

Examples of usage: `datetime`

Examples of working with **`datetime`** objects:

```
>>> import datetime as dt

>>> # Using datetime.combine()
>>> d = dt.date(2005, 7, 14)
>>> t = dt.time(12, 30)
>>> dt.datetime.combine(d, t)
datetime.datetime(2005, 7, 14, 12, 30)

>>> # Using datetime.now()
>>> dt.datetime.now()
datetime.datetime(2007, 12, 6, 16, 29, 43, 79043)   # G
>>> dt.datetime.now(dt.timezone.utc)
datetime.datetime(2007, 12, 6, 15, 29, 43, 79060, tzinf

>>> # Using datetime.strptime()
>>> my_datetime = dt.datetime.strptime("21/11/06 16:30",
>>> my_datetime
datetime.datetime(2006, 11, 21, 16, 30)

>>> # Using datetime.timetuple() to get tuple of all att
>>> tt = my_datetime.timetuple()
>>> for it in tt:
...     print(it)
...
2006    # year
11      # month
21      # day
16      # hour
30      # minute
0       # second
1       # weekday (0 = Monday)
325     # number of days since 1st January
-1      # dst - method tzinfo.dst() returned None

>>> # Date in ISO format
>>> ic = my_datetime.isocalendar()
>>> for it in ic:
...     print(it)
...
2006    # ISO year
47      # ISO week
2       # ISO weekday
```

```
>>> # Formatting a datetime
>>> my_datetime.strftime("%A, %d. %B %Y %I:%M%p")
'Tuesday, 21. November 2006 04:30PM'
>>> 'The {1} is {0:%d}, the {2} is {0:%B}, the {3} is {
'The day is 21, the month is November, the time is 04:3
```

El siguiente ejemplo define una subclase **`tzinfo`** que captura la información de zona horaria de *Kabul*, Afganistán, que utilizó +4 UTC hasta 1945 y +4:30 UTC a partir de entonces:

```
import datetime as dt

class KabulTz(dt.tzinfo):
    # Kabul used +4 until 1945, when they moved to +4:3
    UTC_MOVE_DATE = dt.datetime(1944, 12, 31, 20, tzinf

    def utcoffset(self, when):
        if when.year < 1945:
            return dt.timedelta(hours=4)
        elif (1945, 1, 1, 0, 0) <= when.timetuple()[:5]
            # An ambiguous ("imaginary") half-hour rang
            # a 'fold' in time due to the shift from +4
            # If when falls in the imaginary range, use
            # to resolve. See PEP 495.
            return dt.timedelta(hours=4, minutes=(30 if
        else:
            return dt.timedelta(hours=4, minutes=30)

    def fromutc(self, when):
        # Follow same validations as in datetime.tzinfo
        if not isinstance(when, dt.datetime):
            raise TypeError("fromutc() requires a dateti
        if when.tzinfo is not self:
            raise ValueError("when.tzinfo is not self")

        # A custom implementation is required for fromut
        # the input to this function is a datetime with
        # but with a tzinfo set to self.
        # See datetime.astimezone or fromtimestamp.
        if when.replace(tzinfo=dt.timezone.utc) >= self.
            return when + dt.timedelta(hours=4, minutes=
        else:
            return when + dt.timedelta(hours=4)

    def dst(self, when):
        # Kabul does not observe daylight saving time.
        return dt.timedelta(0)

    def tzname(self, when):
        if when >= self.UTC_MOVE_DATE:
            return "+04:30"
        return "+04"
```

Uso de `KabulTz` desde arriba

```
>>> tz1 = KabulTz()

>>> # Datetime before the change
>>> dt1 = dt.datetime(1900, 11, 21, 16, 30, tzinfo=tz1)
>>> print(dt1.utcoffset())
4:00:00

>>> # Datetime after the change
>>> dt2 = dt.datetime(2006, 6, 14, 13, 0, tzinfo=tz1)
>>> print(dt2.utcoffset())
4:30:00

>>> # Convert datetime to another time zone
>>> dt3 = dt2.astimezone(dt.timezone.utc)
>>> dt3
datetime.datetime(2006, 6, 14, 8, 30, tzinfo=datetime.ti
>>> dt2
datetime.datetime(2006, 6, 14, 13, 0, tzinfo=KabulTz())
>>> dt2 == dt3
True
```

`time` objects

A **`time`** object represents a (local) time of day, independent of any particular day, and subject to adjustment via a **`tzinfo`** object.

class datetime.time(*hour=0*, *minute=0*, *second=0*, *microsecond=0*, *tzinfo=None*, ***, *fold=0*)

Todos los argumentos son opcionales. *tzinfo* puede ser `None`, o una instancia de una subclase **`tzinfo`**. Los argumentos restantes deben ser enteros en los siguientes rangos:

- `0 <= hour < 24`,
- `0 <= minute < 60`,
- `0 <= second < 60`,
- `0 <= microsecond < 1000000`,
- `fold in [0, 1]`.

If an argument outside those ranges is given, **`ValueError`** is raised. All default to 0 except *tzinfo*, which defaults to `None`.

Atributos de clase:

time.min

El primero representable **`time`**,``time (0, 0, 0, 0)``.

time.max

El último representable **`time`**, `time(23, 59, 59, 999999)`.

time.resolution

La diferencia más pequeña posible entre los objetos no iguales **`time`**, `timedelta(microseconds=1)`, aunque tenga en cuenta que la aritmética en objetos **`time`** no es compatible.

Atributos de instancia (solo lectura):

time.hour

En `range(24)`.

time.minute

En `range(60)`.

time.second

En `range(60)`.

time.microsecond

En `range(1000000)`.

time.tzinfo

El objeto pasado como argumento *tzinfo* al constructor de la clase **`time`** , o `None` si no se pasó ninguno.

time.fold

In `[0, 1]`. Used to disambiguate wall times during a repeated interval. (A repeated interval occurs when clocks are rolled back at the end of daylight saving time or when the UTC offset for the current zone is decreased for political reasons.) The values 0 and 1 represent, respectively, the earlier and later of the two moments with the same wall time representation.

Added in version 3.6.

`time` objects support equality and order comparisons, where `a` is considered less than `b` when `a` precedes `b` in time.

Naive and aware **`time`** objects are never equal. Order comparison between naive and aware **`time`** objects raises **`TypeError`**.

If both comparands are aware, and have the same **`tzinfo`** attribute, the **`tzinfo`** and **`fold`** attributes are ignored and the base times are compared. If both comparands are aware and have different **`tzinfo`** attributes, the comparands are first adjusted by subtracting their UTC offsets (obtained from `self.utcoffset()`).

Distinto en la versión 3.3: Equality comparisons between aware and naive **`time`** instances don't raise **`TypeError`**.

En contextos booleanos, un objeto `time` siempre se considera verdadero.

Distinto en la versión 3.5: Before Python 3.5, a `time` object was considered to be false if it represented midnight in UTC. This behavior was considered obscure and error-prone and has been removed in Python 3.5. See bpo-13936 [https://bugs.python.org/issue?@action=redirect&bpo=13936] for more information.

Other constructors:

classmethod time.fromisoformat(*time_string*)

Devuelve un `time` correspondiente a un *time_string* en cualquier formato ISO 8601 válido, con las siguientes excepciones:

1. Las compensaciones de zona horaria pueden tener fracciones de segundo.
2. No se requiere el `T` inicial, que normalmente se requiere en los casos en que puede haber ambigüedad entre una fecha y una hora.
3. Las fracciones de segundo pueden tener cualquier número de dígitos (cualquier número más allá de 6 será truncado).
4. No se admiten fracciones de horas y minutos.

Examples:

```
>>> import datetime as dt
>>> dt.time.fromisoformat('04:23:01')
datetime.time(4, 23, 1)
>>> dt.time.fromisoformat('T04:23:01')
datetime.time(4, 23, 1)
>>> dt.time.fromisoformat('T042301')
datetime.time(4, 23, 1)
>>> dt.time.fromisoformat('04:23:01.000384')
datetime.time(4, 23, 1, 384)
>>> dt.time.fromisoformat('04:23:01,000384')
datetime.time(4, 23, 1, 384)
>>> dt.time.fromisoformat('04:23:01+04:00')
```

```
datetime.time(4, 23, 1, tzinfo=datetime.timezone(da
>>> dt.time.fromisoformat('04:23:01Z')
datetime.time(4, 23, 1, tzinfo=datetime.timezone.ut
>>> dt.time.fromisoformat('04:23:01+00:00')
datetime.time(4, 23, 1, tzinfo=datetime.timezone.ut
```

Added in version 3.7.

Distinto en la versión 3.11: Previously, this method only supported formats that could be emitted by `time.isoformat()`.

classmethod time.strptime(*date_string*, *format*)

Return a `time` corresponding to *date_string*, parsed according to *format*.

If *format* does not contain microseconds or timezone information, this is equivalent to:

```
time(*(time.strptime(date_string, format)[3:6]))
```

`ValueError` is raised if the *date_string* and *format* cannot be parsed by `time.strptime()` or if it returns a value which is not a time tuple. See also strftime() and strptime() behavior and `time.fromisoformat()`.

Added in version 3.14.

Métodos de instancia:

time.replace(*hour=self.hour*, *minute=self.minute*, *second=self.second*, *microsecond=self.microsecond*, *tzinfo=self.tzinfo*, ***, *fold=0*)

Return a new `time` with the same values, but with specified parameters updated. Note that `tzinfo=None` can be specified to create a naive `time` from an aware `time`, without conversion of the time data.

`time` objects are also supported by generic function `copy.replace()`.

Distinto en la versión 3.6: Added the *fold* parameter.

time.isoformat(*timespec='auto'*)

Retorna una cadena que representa la hora en formato ISO 8601, una de:

- `HH:MM:SS.ffffff`, si **`microsecond`** no es 0
- `HH:MM:SS`, si **`microsecond`** es 0
- `HH:MM:SS.ffffff+HH:MM[:SS[.ffffff]]`, si **`utcoffset()`** no retorna `None`
- `HH:MM:SS+HH:MM[:SS[.ffffff]]`, si **`microsecond`** es 0 y **`utcoffset()`** no retorna `None`

El argumento opcional *timespec* especifica el número de componentes adicionales del tiempo a incluir (el valor predeterminado es `'auto'`). Puede ser uno de los siguientes:

- `'auto'`: Igual que `'seconds'` si **`microsecond`** es 0, igual que `'microseconds'` de lo contrario.
- `'hours'`: incluye el **`hour`** en el formato de dos dígitos `HH`.
- `'minutes'`: Incluye **`hour`** y **`minute`** en formato `HH:MM`.
- `'seconds'`: Incluye **`hour`**, **`minute`**, y **`second`** en formato `HH:MM:SS`.
- `'milliseconds'`: Incluye tiempo completo, pero trunca la segunda parte fraccionaria a milisegundos. Formato `HH:MM:SS.sss`.
- `'microseconds'`: Incluye tiempo completo en formato `HH:MM:SS.ffffff`.

Nota

Los componentes de tiempo excluidos están truncados, no redondeados.

`ValueError` lanzará un argumento inválido *timespec*.

Ejemplo:

```
>>> import datetime as dt
>>> dt.time(hour=12, minute=34, second=56, microsec
'12:34'
>>> my_time = dt.time(hour=12, minute=34, second=56
>>> my_time.isoformat(timespec='microseconds')
'12:34:56.000000'
>>> my_time.isoformat(timespec='auto')
'12:34:56'
```

Distinto en la versión 3.6: Added the *timespec* parameter.

time.__str__()

For a time `t`, `str(t)` is equivalent to `t.isoformat()`.

time.strftime(*format*)

Return a string representing the time, controlled by an explicit format string. See also strftime() and strptime() behavior and **`time.isoformat()`**.

time.__format__(*format*)

Same as **`time.strftime()`**. This makes it possible to specify a format string for a **`time`** object in formatted string literals and when using **`str.format()`**. See also strftime() and strptime() behavior and **`time.isoformat()`**.

time.utcoffset()

Si **`tzinfo`** es `None`, retorna `None`, sino retorna `self.tzinfo.utcoffset(None)`, y genera una excepción si este último no retorna `None` o un objeto de **`timedelta`** con magnitud inferior a un día.

Distinto en la versión 3.7: El desfase UTC no está restringido a un número entero de minutos.

time.dst()

Si **`tzinfo`** es `None`, retorna `None`, sino retorna `self.tzinfo.utcoffset(None)`, y genera una excepción si este último no retorna `None`, o un objeto de **`timedelta`** con magnitud inferior a un día.

Distinto en la versión 3.7: El desfase DST no está restringido a un número entero de minutos.

time.tzname()

Si **`tzinfo`** es `None`, retorna `None`, sino retorna `self.tzinfo.tzname(None)`, o genera una excepción si este último no retorna `None` o un objeto de cadena.

Examples of usage: `time`

Ejemplos de trabajo con el objeto **`time`**:

```
>>> import datetime as dt
>>> class TZ1(dt.tzinfo):
...     def utcoffset(self, when):
...         return dt.timedelta(hours=1)
...     def dst(self, when):
...         return dt.timedelta(0)
...     def tzname(self, when):
...         return "+01:00"
...     def  __repr__(self):
...         return f"{self.__class__.__name__}()"
...
>>> t = dt.time(12, 10, 30, tzinfo=TZ1())
>>> t
datetime.time(12, 10, 30, tzinfo=TZ1())
>>> t.isoformat()
'12:10:30+01:00'
>>> t.dst()
datetime.timedelta(0)
>>> t.tzname()
'+01:00'
>>> t.strftime("%H:%M:%S %Z")
'12:10:30 +01:00'
>>> 'The {} is {:%H:%M}.'.format("time", t)
```

```
'The time is 12:10.'
```

`tzinfo` objects

class datetime.tzinfo

This is an abstract base class, meaning that this class should not be instantiated directly. Define a subclass of `tzinfo` to capture information about a particular time zone.

An instance of (a concrete subclass of) `tzinfo` can be passed to the constructors for `datetime` and `time` objects. The latter objects view their attributes as being in local time, and the `tzinfo` object supports methods revealing offset of local time from UTC, the name of the time zone, and DST offset, all relative to a date or time object passed to them.

You need to derive a concrete subclass, and (at least) supply implementations of the standard `tzinfo` methods needed by the `datetime` methods you use. The `datetime` module provides `timezone`, a simple concrete subclass of `tzinfo` which can represent time zones with fixed offset from UTC such as UTC itself or North American EST and EDT.

Special requirement for pickling: A `tzinfo` subclass must have an `__init__()` method that can be called with no arguments, otherwise it can be pickled but possibly not unpickled again. This is a technical requirement that may be relaxed in the future.

A concrete subclass of `tzinfo` may need to implement the following methods. Exactly which methods are needed depends on the uses made of aware `datetime` objects. If in doubt, simply implement all of them.

tzinfo.utcoffset(*dt*)

Retorna el desplazamiento de la hora local desde UTC, como un objeto `timedelta` que es positivo al este de UTC. Si la hora local es al oeste de UTC, esto debería ser negativo.

Esto representa el desplazamiento *total* de UTC; por ejemplo, si un objeto `tzinfo` representa ajustes de zona horaria y DST, `utcoffset()` debería retornar su suma. Si no se conoce el desplazamiento UTC, retorna `None`. De lo contrario, el valor detonado debe ser un objeto de `timedelta` estrictamente entre `-timedelta(hours = 24)` y `timedelta (hours = 24)` (la magnitud del desplazamiento debe ser inferior a un día) La mayoría de las implementaciones de **`utcoffset()`** probablemente se parecerán a una de estas dos:

```
return CONSTANT                 # fixed-offset clas
return CONSTANT + self.dst(dt)  # daylight-aware cl
```

Si `utcoffset()` no retorna `None`, `dst()` no debería retornar `None` tampoco.

La implementación por defecto de `utcoffset()` genera **`NotImplementedError`**.

Distinto en la versión 3.7: El desfase UTC no está restringido a un número entero de minutos.

tzinfo.dst(*dt*)

Retorna el ajuste del horario de verano (DST), como un objeto de `timedelta` o `None` si no se conoce la información de DST.

Return `timedelta(0)` if DST is not in effect. If DST is in effect, return the offset as a `timedelta` object (see `utcoffset()` for details). Note that DST offset, if applicable, has already been added to the UTC offset returned by **`utcoffset()`**, so there's no need to consult **`dst()`** unless you're interested in obtaining DST info separately. For example, `datetime.timetuple()` calls its `tzinfo` attribute's **`dst()`** method to determine how the `tm_isdst` flag should be set, and `tzinfo.fromutc()` calls **`dst()`** to account for DST changes when crossing time zones.

Una instancia *tz* de una subclase `tzinfo` que modela los horarios estándar y diurnos debe ser coherente en este sentido:

```
tz.utcoffset(dt) - tz.dst(dt)
```

must return the same result for every **`datetime`** *dt* with `dt.tzinfo == tz`. For sane **`tzinfo`** subclasses, this expression yields the time zone's «standard offset», which should not depend on the date or the time, but only on geographic location. The implementation of **`datetime.astimezone()`** relies on this, but cannot detect violations; it's the programmer's responsibility to ensure it. If a **`tzinfo`** subclass cannot guarantee this, it may be able to override the default implementation of **`tzinfo.fromutc()`** to work correctly with **`astimezone()`** regardless.

La mayoría de las implementaciones de **`dst()`** probablemente se parecerán a una de estas dos:

```
import datetime as dt

def dst(self, when):
    # a fixed-offset class:  doesn't account for DS
    return dt.timedelta(0)
```

o:

```
import datetime as dt

def dst(self, when):
    # Code to set dston and dstoff to the time zone
    # transition times based on the input when.year
    # in standard local time.

    if dston <= when.replace(tzinfo=None) < dstoff:
        return dt.timedelta(hours=1)
    else:
        return dt.timedelta(0)
```

La implementación predeterminada de **`dst()`** genera

`NotImplementedError`.

Distinto en la versión 3.7: El desfase DST no está restringido a un número entero de minutos.

tzinfo.tzname(*dt*)

Return the time zone name corresponding to the `datetime` object *dt*, as a string. Nothing about string names is defined by the **`datetime`** module, and there's no requirement that it mean anything in particular. For example, `"GMT"`, `"UTC"`, `"-500"`, `"-5:00"`, `"EDT"`, `"US/Eastern"`, `"America/New York"` are all valid replies. Return `None` if a string name isn't known. Note that this is a method rather than a fixed string primarily because some `tzinfo` subclasses will wish to return different names depending on the specific value of *dt* passed, especially if the **`tzinfo`** class is accounting for daylight time.

La implementación predeterminada de `tzname()` genera `NotImplementedError`.

These methods are called by a `datetime` or `time` object, in response to their methods of the same names. A **`datetime`** object passes itself as the argument, and a **`time`** object passes `None` as the argument. A `tzinfo` subclass's methods should therefore be prepared to accept a *dt* argument of `None`, or of class **`datetime`**.

Cuando se pasa `None`, corresponde al diseñador de la clase decidir la mejor respuesta. Por ejemplo, retornar `None` es apropiado si la clase desea decir que los objetos de tiempo no participan en los protocolos `tzinfo`. Puede ser más útil que `utcoffset(None)` retorne el desplazamiento UTC estándar, ya que no existe otra convención para descubrir el desplazamiento estándar.

When a `datetime` object is passed in response to a **`datetime`** method, `dt.tzinfo` is the same object as *self*. `tzinfo` methods can rely on this, unless user code calls **`tzinfo`** methods directly. The intent is that the **`tzinfo`** methods interpret *dt* as being in local time, and not need worry about objects in other time zones.

Hay un método más `tzinfo` que una subclase puede desear anular:

tzinfo.fromutc(*dt*)

This is called from the default `datetime.astimezone()` implementation. When called from that, `dt.tzinfo` is *self*, and *dt*'s date and time data are to be viewed as expressing a UTC time. The purpose of `fromutc()` is to adjust the date and time data, returning an equivalent datetime in *self*'s local time.

Most `tzinfo` subclasses should be able to inherit the default `fromutc()` implementation without problems. It's strong enough to handle fixed-offset time zones, and time zones accounting for both standard and daylight time, and the latter even if the DST transition times differ in different years. An example of a time zone the default `fromutc()` implementation may not handle correctly in all cases is one where the standard offset (from UTC) depends on the specific date and time passed, which can happen for political reasons. The default implementations of `astimezone()` and `fromutc()` may not produce the result you want if the result is one of the hours straddling the moment the standard offset changes.

Código de omisión para casos de error, el valor predeterminado `fromutc()` la implementación actúa como

```
import datetime as dt

def fromutc(self, when):
    # raise ValueError error if when.tzinfo is not
    dtoff = when.utcoffset()
    dtdst = when.dst()
    # raise ValueError if dtoff is None or dtdst is
    delta = dtoff - dtdst  # this is self's standar
    if delta:
        when += delta   # convert to standard local
        dtdst = when.dst()
        # raise ValueError if dtdst is None
        if dtdst:
            return when + dtdst
        else:
            return when
```

En el siguiente archivo **`tzinfo_examples.py`** hay algunos ejemplos de clases **`tzinfo`**:

```
import datetime as dt

# A class capturing the platform's idea of local time.
# (May result in wrong values on historical times in
#  timezones where UTC offset and/or the DST rules had
#  changed in the past.)
import time

ZERO = dt.timedelta(0)
HOUR = dt.timedelta(hours=1)
SECOND = dt.timedelta(seconds=1)

STDOFFSET = dt.timedelta(seconds=-time.timezone)
if time.daylight:
    DSTOFFSET = dt.timedelta(seconds=-time.altzone)
else:
    DSTOFFSET = STDOFFSET

DSTDIFF = DSTOFFSET - STDOFFSET


class LocalTimezone(dt.tzinfo):

    def fromutc(self, when):
        assert when.tzinfo is self
        stamp = (when - dt.datetime(1970, 1, 1, tzinfo=s
        args = time.localtime(stamp)[:6]
        dst_diff = DSTDIFF // SECOND
        # Detect fold
        fold = (args == time.localtime(stamp - dst_diff)
        return dt.datetime(*args, microsecond=when.micro
                           tzinfo=self, fold=fold)

    def utcoffset(self, when):
        if self._isdst(when):
            return DSTOFFSET
        else:
            return STDOFFSET

    def dst(self, when):
        if self._isdst(when):
            return DSTDIFF
        else:
            return ZERO

    def tzname(self, when):
        return time.tzname[self._isdst(when)]

    def _isdst(self, when):
        tt = (when.year, when.month, when.day,
              when.hour, when.minute, when.second,
              when.weekday(), 0, 0)
        stamp = time.mktime(tt)
        tt = time.localtime(stamp)
        return tt.tm_isdst > 0


Local = LocalTimezone()


# A complete implementation of current DST rules for maj

def first_sunday_on_or_after(when):
    days_to_go = 6 - when.weekday()
    if days_to_go:
        when += dt.timedelta(days_to_go)
    return when
```

```
# US DST Rules
#
# This is a simplified (i.e., wrong for a few cases) set
# DST start and end times. For a complete and up-to-date
# and timezone definitions, visit the Olson Database (or
# http://www.twinsun.com/tz/tz-link.htm
# https://sourceforge.net/projects/pytz/ (might not be u
#
# In the US, since 2007, DST starts at 2am (standard tim
# Sunday in March, which is the first Sunday on or after
DSTSTART_2007 = dt.datetime(1, 3, 8, 2)
# and ends at 2am (DST time) on the first Sunday of Nov.
DSTEND_2007 = dt.datetime(1, 11, 1, 2)
# From 1987 to 2006, DST used to start at 2am (standard
# Sunday in April and to end at 2am (DST time) on the la
# Sunday of October, which is the first Sunday on or aft
DSTSTART_1987_2006 = dt.datetime(1, 4, 1, 2)
DSTEND_1987_2006 = dt.datetime(1, 10, 25, 2)
# From 1967 to 1986, DST used to start at 2am (standard
# Sunday in April (the one on or after April 24) and to
# on the last Sunday of October, which is the first Sund
# on or after Oct 25.
DSTSTART_1967_1986 = dt.datetime(1, 4, 24, 2)
DSTEND_1967_1986 = DSTEND_1987_2006


def us_dst_range(year):
    # Find start and end times for US DST. For years bef
    # start = end for no DST.
    if 2006 < year:
        dststart, dstend = DSTSTART_2007, DSTEND_2007
    elif 1986 < year < 2007:
        dststart, dstend = DSTSTART_1987_2006, DSTEND_19
    elif 1966 < year < 1987:
        dststart, dstend = DSTSTART_1967_1986, DSTEND_19
    else:
        return (dt.datetime(year, 1, 1), ) * 2
    start = first_sunday_on_or_after(dststart.replace(ye
    end = first_sunday_on_or_after(dstend.replace(year=y
    return start, end


class USTimeZone(dt.tzinfo):

    def __init__(self, hours, reprname, stdname, dstname
        self.stdoffset = dt.timedelta(hours=hours)
        self.reprname = reprname
        self.stdname = stdname
        self.dstname = dstname

    def __repr__(self):
        return self.reprname

    def tzname(self, when):
        if self.dst(when):
            return self.dstname
        else:
            return self.stdname

    def utcoffset(self, when):
        return self.stdoffset + self.dst(when)

    def dst(self, when):
        if when is None or when.tzinfo is None:
            # An exception may be sensible here, in one
            # It depends on how you want to treat them.
            # fromutc() implementation (called by the de
            # implementation) passes a datetime with whe
            return ZERO
        assert when.tzinfo is self
        start, end = us_dst_range(when.year)
        # Can't compare naive to aware objects, so strip
        # when first.
        when = when.replace(tzinfo=None)
        if start + HOUR <= when < end - HOUR:
            # DST is in effect.
            return HOUR
        if end - HOUR <= when < end:
            # Fold (an ambiguous hour): use when.fold to
            return ZERO if when.fold else HOUR
        if start <= when < start + HOUR:
            # Gap (a non-existent hour): reverse the fol
            return HOUR if when.fold else ZERO
        # DST is off.
        return ZERO

    def fromutc(self, when):
        assert when.tzinfo is self
        start, end = us_dst_range(when.year)
        start = start.replace(tzinfo=self)
        end = end.replace(tzinfo=self)
        std_time = when + self.stdoffset
        dst_time = std_time + HOUR
        if end <= dst_time < end + HOUR:
            # Repeated hour
            return std_time.replace(fold=1)
        if std_time < start or dst_time >= end:
            # Standard time
            return std_time
        if start <= std_time < end - HOUR:
            # Daylight saving time
            return dst_time


Eastern  = USTimeZone(-5, "Eastern",  "EST", "EDT")
Central  = USTimeZone(-6, "Central",  "CST", "CDT")
Mountain = USTimeZone(-7, "Mountain", "MST", "MDT")
Pacific  = USTimeZone(-8, "Pacific",  "PST", "PDT")
```

Tenga en cuenta que hay sutilezas inevitables dos veces al año en una subclase **`tzinfo`** que representa tanto el horario estándar como el horario de verano, en los puntos de transición DST. Para mayor concreción, considere *US Eastern* (UTC -0500), donde EDT comienza el minuto después de 1:59 (EST) el segundo domingo de marzo y termina el minuto después de 1:59 (EDT) el primer domingo de noviembre

```
  UTC   3:MM  4:MM  5:MM  6:MM  7:MM  8:MM
  EST  22:MM 23:MM  0:MM  1:MM  2:MM  3:MM
  EDT  23:MM  0:MM  1:MM  2:MM  3:MM  4:MM

start  22:MM 23:MM  0:MM  1:MM  3:MM  4:MM

  end  23:MM  0:MM  1:MM  1:MM  2:MM  3:MM
```

Cuando comienza el horario de verano (la línea de «inicio»),tiempo real transcurrido (*wall time*) salta de 1:59 a 3:00. Un tiempo de pared de la forma 2: MM realmente no tiene sentido ese día, por lo que `astimezone (Eastern)` no entregará un resultado con `hour == 2` el día en que comienza el horario de verano. Por ejemplo, en la transición de primavera de 2016, obtenemos

```
>>> import datetime as dt
>>> from tzinfo_examples import HOUR, Eastern
>>> u0 = dt.datetime(2016, 3, 13, 5, tzinfo=dt.timezone.
>>> for i in range(4):
...     u = u0 + i*HOUR
...     t = u.astimezone(Eastern)
...     print(u.time(), 'UTC =', t.time(), t.tzname())
...
05:00:00 UTC = 00:00:00 EST
06:00:00 UTC = 01:00:00 EST
07:00:00 UTC = 03:00:00 EDT
08:00:00 UTC = 04:00:00 EDT
```

When DST ends (the «end» line), there’s a potentially worse problem: there’s an hour that can’t be spelled unambiguously in local wall time: the last hour of daylight time. In Eastern, that’s times of the form 5:MM UTC on the day daylight time ends. The local wall clock leaps from 1:59 (daylight time) back to 1:00 (standard time) again. Local times of the form 1:MM are ambiguous. **`astimezone()`** mimics the local clock’s behavior by mapping two adjacent UTC hours into the same local hour then. In the Eastern example, UTC times of the form 5:MM and 6:MM both

map to 1:MM when converted to Eastern, but earlier times have the **`fold`** attribute set to 0 and the later times have it set to 1. For example, at the Fall back transition of 2016, we get:

```
>>> import datetime as dt
>>> from tzinfo_examples import HOUR, Eastern
>>> u0 = dt.datetime(2016, 11, 6, 4, tzinfo=dt.timezone.
>>> for i in range(4):
...     u = u0 + i*HOUR
...     t = u.astimezone(Eastern)
...     print(u.time(), 'UTC =', t.time(), t.tzname(), t
...
04:00:00 UTC = 00:00:00 EDT 0
05:00:00 UTC = 01:00:00 EDT 0
06:00:00 UTC = 01:00:00 EST 1
07:00:00 UTC = 02:00:00 EST 0
```

Note that the **`datetime`** instances that differ only by the value of the **`fold`** attribute are considered equal in comparisons.

Applications that can't bear wall-time ambiguities should explicitly check the value of the **`fold`** attribute or avoid using hybrid **`tzinfo`** subclasses; there are no ambiguities when using **`timezone`**, or any other fixed-offset **`tzinfo`** subclass (such as a class representing only EST (fixed offset -5 hours), or only EDT (fixed offset -4 hours)).

Ver también

`zoneinfo`

The **`datetime`** module has a basic **`timezone`** class (for handling arbitrary fixed offsets from UTC) and its **`timezone.utc`** attribute (a UTC **`timezone`** instance).

`zoneinfo` brings the *IANA time zone database* (also known as the Olson database) to Python, and its usage is recommended.

IANA time zone database [https://www.iana.org/time-zones]

La base de datos de zonas horarias (a menudo llamada *tz, tzdata o zoneinfo*) contiene código y datos que representan el historial de la hora local de muchos lugares representativos de todo el mundo. Se actualiza periódicamente para reflejar los cambios realizados por los cuerpos políticos en los límites de la zona horaria, las compensaciones UTC y las reglas de horario de verano.

`timezone` objects

The `timezone` class is a subclass of `tzinfo`, each instance of which represents a time zone defined by a fixed offset from UTC.

Objects of this class cannot be used to represent time zone information in the locations where different offsets are used in different days of the year or where historical changes have been made to civil time.

class datetime.timezone(*offset*, *name=None*)

El argumento *offset* debe especificarse como un objeto de `timedelta` que representa la diferencia entre la hora local y UTC. Debe estar estrictamente entre `-timedelta(horas = 24)` y `timedelta(horas = 24)`, de lo contrario `ValueError` se genera.

El argumento *name* es opcional. Si se especifica, debe ser una cadena de caracteres que se utilizará como el valor retornado por el método `datetime.tzname()`.

Added in version 3.2.

Distinto en la versión 3.7: El desfase UTC no está restringido a un número entero de minutos.

timezone.utcoffset(*dt*)

Retorna el valor fijo especificado cuando se construye la instancia `timezone`.

El argumento *dt* se ignora. El valor de retorno es una instancia de **`timedelta`** igual a la diferencia entre la hora local y UTC.

Distinto en la versión 3.7: El desfase UTC no está restringido a un número entero de minutos.

timezone.tzname(*dt*)

Retorna el valor fijo especificado cuando se construye la instancia **`timezone`**.

Si no se proporciona *name* en el constructor, el nombre retornado por `tzname(dt)` se genera a partir del valor del `offset` de la siguiente manera. Si *offset* es `timedelta (0)`, el nombre es «UTC», de lo contrario es una cadena en el formato `UTC±HH:MM`, donde ± es el signo de `offset`, HH y MM son dos dígitos de `offset.hours` y `offset.minutes` respectivamente.

Distinto en la versión 3.6: El nombre generado a partir de `offset=timedelta(0)` ahora es simplemente `'UTC'`, no `'UTC+00:00'`.

timezone.dst(*dt*)

Siempre retorna `None`.

timezone.fromutc(*dt*)

Retorna `dt + offset`. El argumento *dt* debe ser una instancia consciente (*aware*) **`datetime`**, con `tzinfo` establecido en `self`.

Atributos de clase:

timezone.utc

The UTC time zone, `timezone(timedelta(0))`.

`strftime()` and `strptime()` behavior

`date`, `datetime`, y `time` los objetos admiten un método `strftime(format)`, para crear una cadena que represente el tiempo bajo el control de una cadena de caracteres de formato explícito.

Conversely, the `date.strptime()`, `datetime.strptime()` and `time.strptime()` class methods create an object from a string representing the time and a corresponding format string.

The table below provides a high-level comparison of `strftime()` versus `strptime()`:

	`strftime`	`strptime`
Uso	Convierte objetos a una cadena de acuerdo con un formato dado	Parse a string into an object given a corresponding format
Tipo de método	Método de instancia	Método de clase
Firma	`strftime(format)`	`strptime(date_string, format)`

`strftime()` and `strptime()` format codes

These methods accept format codes that can be used to parse and format dates:

```
>>> import datetime as dt
>>> dt.datetime.strptime('31/01/22 23:59:59.999999',
...                      '%d/%m/%y %H:%M:%S.%f')
datetime.datetime(2022, 1, 31, 23, 59, 59, 999999)
>>> _.strftime('%a %d %b %Y, %I:%M%p')
'Mon 31 Jan 2022, 11:59PM'
```

La siguiente es una lista de todos los códigos de formato que requiere el estándar 1989 C, y estos funcionan en todas las plataformas con una implementación estándar C.

Significado	Ejemplo	Notas
Día de la semana como nombre abreviado según la configuración regional.	*Sun, Mon, …, Sat (en_US);* *So, Mo, …, Sa (de_DE)*	(1)
Día de la semana como nombre completo de la localidad.	*Sunday, Monday, …, Saturday (en_US);* *Sonntag, Montag, …, Samstag (de_DE)*	(1)
Día de la semana como un número decimal, donde 0 es domingo y 6 es sábado.	0, 1, …, 6	

Día del mes como un número decimal rellenado con ceros.	01, 02, …, 31
Mes como nombre abreviado según la configuración regional.	*Jan, Feb, …, Dec (en_US);* *Jan, Feb, …, Dez (de_DE)*
Mes como nombre completo según la configuración regional.	*January, February, …, December (en_US);* *Januar, Februar, …, Dezember (de_DE)*
Mes como un número decimal rellenado con ceros.	01, 02, …, 12
Año sin siglo como un número decimal rellenado con ceros.	00, 01, …, 99
Año con siglo como un número decimal.	0001, 0002, …, 2013, 2014, …, 9998, 9999
Hora (reloj de 24 horas) como un número decimal rellenado con ceros.	00, 01, …, 23
Hora (reloj de 12 horas) como un número decimal rellenado con ceros.	01, 02, …, 12
Equivalente de la configuración regional de AM o PM.	AM, PM (en_US); am, pm (de_DE)
Minuto como un número decimal rellenado con ceros.	00, 01, …, 59
Segundo como un número decimal rellenado con ceros.	00, 01, …, 59
Microsegundo como un número decimal, con ceros hasta 6 dígitos.	000000, 000001, …, 999999
Desplazamiento (*offset*) UTC en la forma `±HHMM[SS[.ffffff]]` (cadena de caracteres vacía si el objeto es naíf (*naive*)).	(vacío), +0000, -0400, +1030, +063415, -030712.345216
Nombre de zona horaria (cadena de caracteres vacía si el objeto es naíf (*naive*)).	(vacío), UTC, GMT
Día del año como un número decimal rellenado con ceros.	001, 002, …, 366
Número de semana del año (domingo como primer día de la semana) como un número decimal con ceros. Todos los días de un nuevo año que preceden al primer domingo se consideran en la semana 0.	00, 01, …, 53
Número de semana del año (lunes como primer día de la semana) como un número decimal con ceros. Todos los días de un nuevo año que preceden al primer lunes se consideran en la semana 0.	00, 01, …, 53
Representación apropiada de fecha y hora de la configuración regional.	*Tue Aug 16 21:30:00 1988 (en_US);* *Di 16 Aug 21:30:00 1988 (de_DE)*
Representación de fecha apropiada de la configuración regional.	08/16/88 (*None*); 08/16/1988 (en_US); 16.08.1988 (de_DE)
Representación de la hora apropiada de la configuración regional.	21:30:00 (en_US); 21:30:00 (de_DE)
Un carácter literal `'%'`.	%

Se incluyen varias directivas adicionales no requeridas por el estándar C89 por conveniencia. Todos estos parámetros corresponden a valores de fecha ISO 8601.

Significado	Ejemplo	Notas
ISO 8601 año con siglo que representa el año que contiene la mayor parte de la semana ISO (`%V`).	0001, 0002, …, 2013, 2014, …, 9998, 9999	(8)
ISO 8601 día de la semana como un número decimal donde 1 es lunes.	1, 2, …, 7	
ISO 8601 semana como un número decimal con lunes como primer día de la semana. La semana 01 es la semana que contiene el 4 de enero.	01, 02, …, 53	(8), (9)
UTC offset in the form `±HH:MM[:SS[.ffffff]]` (empty string if the object is naive).	(empty), +00:00, -04:00, +10:30, +06:34:15, -03:07:12.345216	(6)

These may not be available on all platforms when used with the **`strftime()`** method. The ISO 8601 year and ISO 8601 week directives are not interchangeable with the year and week number directives above. Calling **`strptime()`** with incomplete or ambiguous ISO 8601 directives will raise a **`ValueError`**.

The full set of format codes supported varies across platforms, because Python calls the platform C library's **`strftime()`** function, and platform variations are common. To see the full set of format codes supported on your platform, consult the *strftime(3)* *[https://manpages.debian.org/strftime(3)]* documentation. There are also differences between platforms in handling of unsupported format specifiers.

Added in version 3.6: `%G`, `%u` y `%V` fueron añadidos.

Added in version 3.12: `%:z` was added.

Technical detail

Broadly speaking, `d.strftime(fmt)` acts like the **`time`** module's `time.strftime(fmt, d.timetuple())` although not all objects support a **`timetuple()`** method.

For the **`datetime.strptime()`** and **`date.strptime()`** class methods, the default value is `1900-01-01T00:00:00.000`: any components not specified in the format string will be pulled from

the default value.

Nota

Format strings without separators can be ambiguous for parsing. For example, with `%Y%m%d`, the string `2026111` may be parsed either as `2026-11-01` or as `2026-01-11`. Use separators to ensure the input is parsed as intended.

Nota

When used to parse partial dates lacking a year, **`datetime.strptime()`** and **`date.strptime()`** will raise when encountering February 29 because the default year of 1900 is *not* a leap year. Always add a default leap year to partial date strings before parsing.

```
>>> import datetime as dt
>>> value = "2/29"
>>> dt.datetime.strptime(value, "%m/%d")
Traceback (most recent call last):
...
ValueError: day 29 must be in range 1..28 for month 2 i
>>> dt.datetime.strptime(f"1904 {value}", "%Y %m/%d")
datetime.datetime(1904, 2, 29, 0, 0)
```

Usar `datetime.strptime(date_string, format)` es equivalente a:

```
datetime(*(time.strptime(date_string, format)[0:6]))
```

except when the format includes sub-second components or time zone offset information, which are supported in `datetime.strptime` but are discarded by `time.strptime`.

For **`time`** objects, the format codes for year, month, and day should not be used, as **`time`** objects have no such values. If they're used anyway, 1900 is substituted for the year, and 1 for the month and day.

For `date` objects, the format codes for hours, minutes, seconds, and microseconds should not be used, as `date` objects have no such values. If they're used anyway, 0 is substituted for them.

Por la misma razón, el manejo de cadenas de formato que contienen puntos de código Unicode que no se pueden representar en el conjunto de caracteres del entorno local actual también depende de la plataforma. En algunas plataformas, estos puntos de código se conservan intactos en la salida, mientras que en otros `strftime` puede generar `UnicodeError` o retornar una cadena vacía.

Notas:

1. Because the format depends on the current locale, care should be taken when making assumptions about the output value. Field orderings will vary (for example, «month/day/year» versus «day/month/year»), and the output may contain non-ASCII characters.

2. The `strptime()` method can parse years in the full [1, 9999] range, but years < 1000 must be zero-filled to 4-digit width.

   Distinto en la versión 3.2: In previous versions, `strftime()` method was restricted to years $>= 1900$.

   Distinto en la versión 3.3: In version 3.2, `strftime()` method was restricted to years $>= 1000$.

3. When used with the `strptime()` method, the `%p` directive only affects the output hour field if the `%I` directive is used to parse the hour.

4. Unlike the `time` module, the `datetime` module does not support leap seconds.

5. When used with the `strptime()` method, the `%f` directive accepts from one to six digits and zero pads on the right. `%f` is an extension to the set of format characters in the C standard (but implemented separately in datetime objects, and therefore always available).

6. For a naive object, the `%z`, `%:z` and `%Z` format codes are replaced by empty strings.

   Para un objeto consciente (*aware*)

   `%z`

      `utcoffset()` is transformed into a string of the form `±HHMM[SS[.ffffff]]`, where `HH` is a 2-digit string giving the number of UTC offset hours, `MM` is a 2-digit string giving the number of UTC offset minutes, `SS` is a 2-digit string giving the number of UTC offset seconds and `ffffff` is a 6-digit string giving the number of UTC offset microseconds. The `ffffff` part is omitted when the offset is a whole number of seconds and both the `ffffff` and the `SS` part is omitted when the offset is a whole number of minutes. For example, if **`utcoffset()`** returns `timedelta(hours=-3, minutes=-30)`, `%z` is replaced with the string `'-0330'`.

   Distinto en la versión 3.7: El desfase UTC no está restringido a un número entero de minutos.

   Distinto en la versión 3.7: When the `%z` directive is provided to the **`strptime()`** method, the UTC offsets can have a colon as a separator between hours, minutes and seconds. For example, `'+01:00:00'` will be parsed as an offset of one hour. In addition, providing `'Z'` is identical to `'+00:00'`.

   `%:z`

      Behaves exactly as `%z`, but has a colon separator added between hours, minutes and seconds.

   `%Z`

      In **`strftime()`**, `%Z` is replaced by an empty string if **`tzname()`** returns `None`; otherwise `%Z` is replaced by the returned value, which must be a string.

      `strptime()` only accepts certain values for `%Z`:

      1. cualquier valor en `time.tzname` para la

   configuración regional de su máquina
2. los valores codificados de forma rígida `UTC` y `GMT`

Entonces, alguien que viva en Japón puede tener `JST`, `UTC` y `GMT` como valores válidos, pero probablemente no `EST`. Lanzará `ValueError` para valores no válidos.

Distinto en la versión 3.2: When the `%z` directive is provided to the **`strptime()`** method, an aware **`datetime`** object will be produced. The `tzinfo` of the result will be set to a **`timezone`** instance.

7. When used with the **`strptime()`** method, `%U` and `%W` are only used in calculations when the day of the week and the calendar year (`%Y`) are specified.

8. Similar to `%U` and `%W`, `%V` is only used in calculations when the day of the week and the ISO year (`%G`) are specified in a **`strptime()`** format string. Also note that `%G` and `%Y` are not interchangeable.

9. When used with the **`strptime()`** method, the leading zero is optional for formats `%d`, `%m`, `%H`, `%I`, `%M`, `%S`, `%j`, `%U`, `%W`, and `%V`. Format `%y` does require a leading zero.

10. When parsing a month and day using **`strptime()`**, always include a year in the format. If the value you need to parse lacks a year, append an explicit dummy leap year. Otherwise your code will raise an exception when it encounters leap day because the default year used by the parser (1900) is not a leap year. Users run into that bug every leap year.

```
>>> month_day = "02/29"
>>> dt.datetime.strptime(f"{month_day};1984", "%m/%
datetime.datetime(1984, 2, 29, 0, 0)
```

Deprecated since version 3.13, will be removed in version 3.15: **`strptime()`** calls using a format string containing a day of month without a year now emit a **`DeprecationWarning`**.

In 3.15 or later we may change this into an error or change the default year to a leap year. See gh-70647 [https://github.com/python/cpython/issues/70647].

Pie de notas

If, that is, we ignore the effects of relativity.

Esto coincide con la definición del calendario «proléptico gregoriano» en el libro de *Dershowitz y Reingold Cálculos calendáricos*, donde es el calendario base para todos los cálculos. Consulte el libro sobre algoritmos para convertir entre ordinales gregorianos prolépticos y muchos otros sistemas de calendario.

Consulte guide to the mathematics of the ISO 8601 calendar [https://web.archive.org/web/20220531051136/https://webspace.science.uu.nl/~gent0113/calendar/isocalendar.htm] de R. H. van Gent para obtener una buena explicación.

`zoneinfo` — IANA time zone support

Added in version 3.9.

Código fuente: Lib/zoneinfo [https://github.com/python/cpython/tree/3.14/Lib/zoneinfo]

The `zoneinfo` module provides a concrete time zone implementation to support the IANA time zone database as originally specified in **PEP 615** [https://peps.python.org/pep-0615/]. By default, `zoneinfo` uses the system's time zone data if available; if no system time zone data is available, the library will fall back to using the first-party tzdata [https://pypi.org/project/tzdata/] package available on PyPI.

Ver también

Modulo: `datetime`
Proporciona los tipos `time` y `datetime` con los que está diseñada la clase `ZoneInfo`.

Package tzdata [https://pypi.org/project/tzdata/]
Paquete de origen mantenido por los desarrolladores del núcleo de CPython para suministrar datos de zonas horarias a través de PyPI.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

Usando `ZoneInfo`

`ZoneInfo` es una implementación concreta de la clase base abstracta `datetime.tzinfo`, y está pensada para ser adjuntada a `tzinfo`, bien a través del constructor, del método `datetime.replace` o de `datetime.astimezone`:

```
>>> from zoneinfo import ZoneInfo
>>> import datetime as dt

>>> when = dt.datetime(2020, 10, 31, 12, tzinfo=ZoneInfo
>>> print(when)
2020-10-31 12:00:00-07:00

>>> when.tzname()
'PDT'
```

Las fechas construidas de esta manera son compatibles con la aritmética de fechas y manejan las transiciones del horario de verano sin ninguna otra intervención:

```
>>> when_add = when + dt.timedelta(days=1)

>>> print(when_add)
2020-11-01 12:00:00-08:00

>>> when_add.tzname()
'PST'
```

Estas zonas horarias también soportan el atributo `fold` introducido en **PEP 495** [https://peps.python.org/pep-0495/]. Durante las transiciones de desfase que inducen tiempos ambiguos (como una transición de horario de verano a horario estándar), se utiliza el desfase de *antes* de la transición cuando `fold=0`, y el desfase *después* de la transición cuando `fold=1`, por ejemplo:

```
>>> when = dt.datetime(2020, 11, 1, 1, tzinfo=ZoneInfo("
>>> print(when)
2020-11-01 01:00:00-07:00

>>> print(when.replace(fold=1))
2020-11-01 01:00:00-08:00
```

Al convertir desde otra zona horaria, el pliegue se ajustará al valor correcto:

```
>>> LOS_ANGELES = ZoneInfo("America/Los_Angeles")
>>> when_utc = dt.datetime(2020, 11, 1, 8, tzinfo=dt.tim

>>> # Before the PDT -> PST transition
>>> print(when_utc.astimezone(LOS_ANGELES))
2020-11-01 01:00:00-07:00

>>> # After the PDT -> PST transition
>>> print((when_utc + dt.timedelta(hours=1)).astimezone(
2020-11-01 01:00:00-08:00
```

Fuentes de datos

The `zoneinfo` module does not directly provide time zone data, and instead pulls time zone information from the system time zone database or the first-party PyPI package tzdata [https://pypi.org/project/tzdata/], if available. Some systems, including notably Windows systems, do not have an IANA database available, and so for projects targeting cross-platform compatibility that require time zone data, it is recommended to declare a dependency on tzdata. If neither system data nor tzdata are available, all calls to **`ZoneInfo`** will raise **`ZoneInfoNotFoundError`**.

Configurando los orígenes de datos

Cuando se llama a `ZoneInfo(key)`, el constructor busca primero en los directorios especificados en **`TZPATH`** un archivo que coincida con `key`, y en caso de fallo busca una coincidencia en el paquete tzdata. Este comportamiento puede configurarse de tres maneras:

1. El **`TZPATH`** por defecto cuando no se especifica otra cosa puede configurarse en tiempo de compilación.
2. **`TZPATH`** puede configurarse utilizando una variable de entorno.
3. En runtime, la ruta de búsqueda puede ser manipulada usando la función **`reset_tzpath()`**.

Configuración en tiempo de compilación

El **`TZPATH`** predeterminado incluye varias ubicaciones de implementación comunes para la base de datos de la zona horaria (excepto en Windows, donde no hay ubicaciones «conocidas» para los datos de la zona horaria). En los sistemas POSIX, los distribuidores posteriores y aquellos que compilan Python desde la fuente que saben dónde se implementan los datos de la zona horaria del sistema pueden cambiar la ruta de la zona horaria predeterminada especificando la opción de tiempo de compilación `TZPATH` (o, más probablemente, el **`configure flag --with-tzpath`**), que debería ser una cadena delimitada por **`os.pathsep`**.

En todas las plataformas, el valor configurado está disponible como la clave `TZPATH` en **`sysconfig.get_config_var()`**.

Configuración del entorno

Cuando se inicializa **`TZPATH`** (ya sea en el momento de la importación o cuando se llama a **`reset_tzpath()`** sin argumentos), el módulo `zoneinfo` utilizará la variable de entorno `PYTHONTZPATH`, si existe, para establecer la ruta de búsqueda.

PYTHONTZPATH

> Se trata de una cadena separada por **`os.pathsep`** que contiene la ruta de búsqueda de la zona horaria a utilizar. Debe consistir sólo en rutas absolutas y no relativas. Los componentes relativos especificados en `PYTHONTZPATH` no se utilizarán, pero por lo demás el comportamiento cuando se especifica una ruta relativa es definido por la implementación; CPython lanzará **`InvalidTZPathWarning`**, pero otras implementaciones son libres de ignorar silenciosamente el componente erróneo o lanzar una excepción.

Para que el sistema ignore los datos del sistema y utilice el paquete tzdata en su lugar, establezca `PYTHONTZPATH=""`.

Configuración de tiempo de ejecución

La ruta de búsqueda de TZ también puede configurarse en tiempo de ejecución mediante la función **`reset_tzpath()`**. Por lo general, esta operación no es aconsejable, aunque es razonable utilizarla en funciones de prueba que requieran el uso de una ruta de zona horaria específica (o que requieran deshabilitar el acceso a las zonas horarias del sistema).

La clase `ZoneInfo`

class zoneinfo.ZoneInfo(*key*)

Una subclase concreta de **`datetime.tzinfo`** que representa una zona horaria IANA especificada por la cadena `key`. Las llamadas al constructor primario siempre devolverán objetos que se comparan de forma idéntica; dicho de otro modo, salvo la invalidación de la caché mediante **`ZoneInfo.clear_cache()`**, para todos los valores de `key`, la siguiente afirmación siempre será verdadera:

```
a = ZoneInfo(key)
b = ZoneInfo(key)
assert a is b
```

La `key` debe tener la forma de una ruta POSIX relativa y normalizada, sin referencias de nivel superior. El constructor lanzará **`ValueError`** si se pasa una clave no conforme.

Si no se encuentra ningún archivo que coincida con la `clave`, el constructor lanzará **`ZoneInfoNotFoundError`**.

La clase `ZoneInfo` tiene dos constructores alternativos:

classmethod ZoneInfo.from_file(*file_obj*, /, *key=None*)

Construye un objeto `ZoneInfo` a partir de un objeto tipo archivo que retorna bytes (por ejemplo, un archivo abierto en modo binario o un objeto **`io.BytesIO`**). A diferencia del constructor primario, éste siempre construye un nuevo objeto.

El parámetro `key` establece el nombre de la zona a efectos de **`__str__()`** y **`__repr__()`**.

Los objetos creados a través de este constructor no pueden ser serializados (ver pickling).

`ValueError` is raised if the data read from *file_obj* is not a valid TZif file.

classmethod ZoneInfo.no_cache(*key*)

Un constructor alternativo que omite la caché del constructor. Es idéntico al constructor principal, pero retorna un nuevo objeto en cada llamada. Es muy probable que esto sea útil para propósitos de prueba o demostración, pero también se puede utilizar para crear un sistema con una estrategia de invalidación de caché diferente.

Los objetos creados a través de este constructor también pasarán por encima de la caché de un proceso de deserialización cuando sean deserializados.

Prudencia

El uso de este constructor puede cambiar la semántica de tus datetimes de manera sorprendente, sólo úsalo si sabes que lo necesitas.

También están disponibles los siguientes métodos de clase:

classmethod ZoneInfo.clear_cache(**, only_keys=None*)

Un método para invalidar la caché de la clase `ZoneInfo`. Si no se pasan argumentos, se invalidan todas las cachés y la siguiente llamada al constructor primario de cada clave devolverá una nueva instancia.

Si se pasa un iterable de nombres de claves al parámetro `only_keys`, sólo se eliminarán de la caché las claves especificadas. Las claves pasadas a `only_keys` pero que no se encuentran en la caché se ignoran.

Advertencia

Invocar esta función puede cambiar la semántica de las fechas y horas que utilizan `ZoneInfo` de formas sorprendentes; esto modifica el estado del módulo y, por lo tanto, puede tener efectos de gran alcance. Utilícela solo si sabe que la necesita.

La clase tiene un atributo:

ZoneInfo.key

Se trata de un attribute de sólo lectura que retorna el valor de `key` pasado al constructor, que debe ser una clave de búsqueda en la base de datos de zonas horarias de la IANA (por ejemplo, `America/New_York`, `Europe/Paris` o `Asia/Tokyo`.

Para las zonas construidas a partir de un archivo sin especificar un parámetro `key`, se establecerá como `None`.

Nota

Aunque es una práctica algo común exponerlos a los usuarios finales, estos valores están diseñados para ser claves primarias para representar las zonas relevantes y no necesariamente elementos orientados al usuario. Se pueden utilizar proyectos como CLDR (Unicode Common Locale Data Repository) para obtener cadenas más fáciles de usar a partir de estas claves.

Representaciones de cadenas

La representación de cadena que se retorna al llamar a **`str`** sobre un objeto **`ZoneInfo`** utiliza por defecto el atributo **`ZoneInfo.key`** (ver la nota de uso en la documentación del atributo):

```
>>> zone = ZoneInfo("Pacific/Kwajalein")
>>> str(zone)
```

```
'Pacific/Kwajalein'

>>> when = dt.datetime(2020, 4, 1, 3, 15, tzinfo=zone)
>>> f"{when.isoformat()} [{when.tzinfo}]"
'2020-04-01T03:15:00+12:00 [Pacific/Kwajalein]'
```

Para los objetos construidos a partir de un fichero sin especificar un parámetro `key`, `str` vuelve a llamar a **repr` ()**. El parámetro `repr` de `ZoneInfo` está definido por la implementación y no es necesariamente estable entre versiones, pero se garantiza que no es una clave válida de `ZoneInfo`.Pickled.

Serialización de Pickle

En lugar de serializar todos los datos de transición, los objetos `ZoneInfo` se serializan por clave, y los objetos `ZoneInfo` construidos a partir de archivos (incluso los que tienen un valor por `key` especifico) no pueden ser serializados.

El comportamiento de un archivo `ZoneInfo` depende de cómo se haya construido:

1. `ZoneInfo(key)`: When constructed with the primary constructor, a `ZoneInfo` object is serialized by key, and when deserialized, the deserializing process uses the primary and thus it is expected that these are the same object as other references to the same time zone. For example, if `europe_berlin_pkl` is a string containing a pickle constructed from `ZoneInfo("Europe/Berlin")`, one would expect the following behavior:

   ```
   >>> a = ZoneInfo("Europe/Berlin")
   >>> b = pickle.loads(europe_berlin_pkl)
   >>> a is b
   True
   ```

2. `ZoneInfo.no_cache(key)`: Cuando se construye a partir del constructor que evita la caché, el objeto `ZoneInfo` también se serializa por clave, pero cuando se deserializa, el proceso de deserialización utiliza el constructor que evita la

caché. Si `europe_berlin_pkl_nc` es una cadena que contiene un pickle construido a partir de `ZoneInfo.no_cache("Europe/Berlin")`, cabría esperar el siguiente comportamiento:

```
>>> a = ZoneInfo("Europe/Berlin")
>>> b = pickle.loads(europe_berlin_pkl_nc)
>>> a is b
False
```

3. `ZoneInfo.from_file(file_obj, /, key=None)`: When constructed from a file, the `ZoneInfo` object raises an exception on pickling. If an end user wants to pickle a `ZoneInfo` constructed from a file, it is recommended that they use a wrapper type or a custom serialization function: either serializing by key or storing the contents of the file object and serializing that.

Este método de serialización requiere que los datos de la zona horaria para la clave requerida estén disponibles tanto en el lado de serialización como en el de deserialización, de forma similar a como se espera que las referencias a las clases y funciones existan tanto en el entorno de serialización como en el de deserialización. También significa que no se garantiza la consistencia de los resultados cuando se retira un `ZoneInfo` recogido en un entorno con una versión diferente de los datos de la zona horaria.

Funciones

zoneinfo.available_timezones()

Obtiene un conjunto que contiene todas las claves válidas para las zonas horarias de la IANA disponibles en cualquier lugar de la ruta de zonas horarias. Se recalcula en cada llamada a la función.

Esta función sólo incluye los nombres de zona canónicos y no incluye las zonas «especiales» como las que se encuentran bajo los directorios `posix/` y `right/`, o la zona `posixrules`.

Prudencia

Esta función puede abrir un gran número de archivos, ya que la mejor manera de determinar si un archivo en la ruta de la zona horaria es una zona horaria válida es leer la «magic string» (cadena mágica) al principio.

Nota

Estos valores no están diseñados para ser expuestos a los usuarios finales; para los elementos de cara al usuario, las aplicaciones deberían utilizar algo como CLDR (el Unicode Common Locale Data Repository) para obtener cadenas más fáciles de usar. Véase también la nota de advertencia sobre **`ZoneInfo.key`**.

zoneinfo.reset_tzpath(*to=None*)

Establece o restablece la ruta de búsqueda de la zona horaria (**`TZPATH`**) para el módulo. Cuando se llama sin argumentos, **`TZPATH`** se establece en el valor por defecto.

La llamada a `reset_tzpath` no invalidará la caché de **`ZoneInfo`**, por lo que las llamadas al constructor primario de `ZoneInfo` sólo utilizarán el nuevo `TZPATH` en caso de que se pierda la caché.

El parámetro `to` debe ser un sequence de cadenas o **`os.PathLike`** y no una cadena, todos los cuales deben ser rutas absolutas. **`ValueError`** se lanzará si se pasa algo que no sea una ruta absoluta.

Globales

zoneinfo.TZPATH

Una secuencia de sólo lectura que representa la ruta de búsqueda de zonas horarias – cuando se construye un `ZoneInfo` a partir de una clave, la clave se une a cada entrada del `TZPATH`, y se utiliza el primer archivo encontrado.

`TZPATH` sólo puede contener rutas absolutas, nunca relativas, independientemente de cómo esté configurado.

El objeto al que apunta `zoneinfo.TZPATH` puede cambiar en respuesta a una llamada a **`reset_tzpath()`**, por lo que se recomienda utilizar `zoneinfo.TZPATH` en lugar de importar `TZPATH` desde `zoneinfo` o asignar una variable de larga duración a `zoneinfo.TZPATH`.

Para más información sobre la configuración de la ruta de búsqueda de zonas horarias, consulte Configurando los orígenes de datos.

Excepciones y advertencias

exception zoneinfo.ZoneInfoNotFoundError

Se lanza cuando la construcción de un objeto **`ZoneInfo`** falla porque la clave especificada no puede encontrarse en el sistema. Es una subclase de **`KeyError`**.

exception zoneinfo.InvalidTZPathWarning

Se lanza cuando **`PYTHONTZPATH`** contiene un componente no válido que será filtrado, como una ruta relativa.

`calendar` — General calendar-related functions

Código fuente: Lib/calendar.py [https://github.com/python/cpython/tree/3.14/Lib/calendar.py]

Este módulo te permite generar calendarios como el programa Unix **cal**, y proporciona funciones útiles adicionales relacionadas con el calendario. Por defecto, estos calendarios tienen el lunes como el primer día de la semana, y el domingo como el último (la convención europea). Use `setfirstweekday()` para establecer el primer día de la semana en domingo (6) o en cualquier otro día de la semana. Los parámetros que especifican fechas se indican como enteros. Para la funcionalidad relacionada, consulta también los módulos `datetime` y `time`.

Las funciones y clases definidas en este módulo utilizan un calendario idealizado, el calendario gregoriano actual extendido indefinidamente en ambas direcciones. Esto coincide con la definición del calendario «Gregoriano proléptico» en el libro «Calendrical Calculations» de Dershowitz y Reingold, donde es el calendario base para todos los cálculos. Los años cero y negativos se interpretan según lo prescrito por la norma ISO 8601. El año 0 es 1 A. C., el año -1 es 2 a. C., y así sucesivamente.

class calendar.Calendar(*firstweekday=0*)

> Crea un objeto `Calendar`. *firstweekday* es un entero que especifica el primer día de la semana. `MONDAY` es `0` (por defecto), `SUNDAY` es `6`.
>
> Un objeto `Calendar` proporciona varios métodos que se pueden utilizar para preparar los datos del calendario para dar formato. Esta clase no hace ningún formato en sí. Este es el trabajo de las subclases.

`Calendar` instances have the following methods and attributes:

firstweekday

> The first weekday as an integer (0–6).
>
> This property can also be set and read using `setfirstweekday()` and `getfirstweekday()` respectively.

getfirstweekday()

> Return an `int` for the current first weekday (0–6).
>
> Identical to reading the `firstweekday` property.

setfirstweekday(*firstweekday*)

> Set the first weekday to *firstweekday*, passed as an `int` (0–6).
>
> Identical to setting the `firstweekday` property.

iterweekdays()

> Return an iterator for the weekday numbers that will be used for one week. The first value from the iterator will be the same as the value of the `firstweekday` property.

itermonthdates(*year*, *month*)

> Retorna un iterador para el mes *month* (1–12) en el año *year*. Este iterador retornará todos los días (como objetos `datetime.date`) para el mes y todos los días antes del inicio del mes o después del final del mes que se requieren para obtener una semana completa.

itermonthdays(*year*, *month*)

> Retorna un iterador para el mes *month* en el año *year* similar a `itermonthdates()`, pero no restringido por el intervalo `datetime.date`. Los días retornados

serán simplemente el día de los números del mes. Para los días fuera del mes especificado, el número de día es `0`.

itermonthdays2(*year*, *month*)

Return an iterator for the month *month* in the year *year* similar to `itermonthdates()`, but not restricted by the `datetime.date` range. Days returned will be tuples consisting of a day of the month number and a weekday number.

itermonthdays3(*year*, *month*)

Retorna un iterador para el mes *month* del año *year* similar a `itermonthdates()`, pero no restringido por el rango `datetime.date`. Los días retornados serán tuplas que consisten en un año, un mes y un día del mes.

Added in version 3.7.

itermonthdays4(*year*, *month*)

Retorna un iterador para el mes *month* del año *year* similar a `itermonthdates()`, pero no restringido por el rango `datetime.date`. Los días retornados serán tuplas que consisten en un año, un mes, un día del mes y un día de la semana.

Added in version 3.7.

monthdatescalendar(*year*, *month*)

Retorna una lista de las semanas del mes *month* del año *year* como semanas completas. Las semanas son listas de siete objetos `datetime.date`.

monthdays2calendar(*year*, *month*)

Retorna una lista de las semanas del mes *month* del año *year* como semanas completas. Las semanas son listas de siete tuplas de números de días y números de días de la semana.

monthdayscalendar(*year*, *month*)

Retorna una lista de las semanas del mes *month* del año *year* como semanas completas. Las semanas son listas de números de siete días.

yeardatescalendar(*year*, *width=3*)

Retorna los datos del año especificado listos para ser formateados. El valor de retorno es una lista de filas de mes. Cada fila de mes contiene hasta *width* meses (por defecto hasta 3). Cada mes contiene entre 4 y 6 semanas y cada semana contiene 1–7 días. Los días son objetos `datetime.date`.

yeardays2calendar(*year*, *width=3*)

Retorna los datos del año especificado listos para ser formateados (similar a `yeardatescalendar()`). Las entradas en las listas de la semana son tuplas de números de días y números de días de la semana. Los números de los días fuera de este mes son cero.

yeardayscalendar(*year*, *width=3*)

Retorna los datos del año especificado listos para ser formateados (similar a `yeardatescalendar()`). Las entradas en las listas de la semana son números de día. Los números de día fuera de este mes son cero.

class calendar.TextCalendar(*firstweekday=0*)

Esta clase puede ser usada para generar calendarios de texto simple.

Las instancias de `TextCalendar` tienen los siguientes métodos:

formatday(*theday*, *weekday*, *width*)

Return a string representing a single day formatted with the given *width*. If *theday* is `0`, return a string of spaces of the specified width, representing an empty day. The *weekday* parameter is unused.

formatweek(*theweek*, *w=0*)

Return a single week in a string with no newline. If *w* is provided, it specifies the width of the date columns, which are centered. Depends on the first weekday as specified in the constructor or set by the **`setfirstweekday()`** method.

formatweekday(*weekday*, *width*)

Return a string representing the name of a single weekday formatted to the specified *width*. The *weekday* parameter is an integer representing the day of the week, where `0` is Monday and `6` is Sunday.

formatweekheader(*width*)

Return a string containing the header row of weekday names, formatted with the given *width* for each column. The names depend on the locale settings and are padded to the specified width.

formatmonth(*theyear*, *themonth*, *w=0*, *l=0*)

Retorna el calendario de un mes en una cadena de varias líneas. Si se proporciona *w*, especifica el ancho de las columnas de fecha, que están centradas. Si se proporciona *l*, especifica el número de líneas que se utilizarán cada semana. Depende del primer día de la semana como se especifica en el constructor o se establece por el método **`setfirstweekday()`**.

formatmonthname(*theyear*, *themonth*, *width=0*, *withyear=True*)

Return a string representing the month’s name centered within the specified *width*. If *withyear* is `True`, include

the year in the output. The *theyear* and *themonth* parameters specify the year and month for the name to be formatted respectively.

prmonth(*theyear*, *themonth*, *w=0*, *l=0*)

Imprime el calendario de un mes como lo retorna **`formatmonth()`**.

formatyear(*theyear*, *w=2*, *l=1*, *c=6*, *m=3*)

Retorna un calendario de *m* columnas para todo un año como una cadena de varias líneas. Los parámetros opcionales *w*, *l* y *c* son para el ancho de la columna de la fecha, las líneas por semana y el número de espacios entre las columnas del mes, respectivamente. Depende del primer día de la semana como se especifica en el constructor o se establece por el método **`setfirstweekday()`**. El primer año para el que se puede generar un calendario depende de la plataforma.

pryear(*theyear*, *w=2*, *l=1*, *c=6*, *m=3*)

Imprime el calendario de un año entero como lo retorna **`formatyear()`**.

class calendar.HTMLCalendar(*firstweekday=0*)

Esta clase puede utilizarse para generar calendarios HTML.

Las instancias de **`HTMLCalendar`** tienen los siguientes métodos:

formatmonth(*theyear*, *themonth*, *withyear=True*)

Retorna el calendario de un mes como una tabla HTML. Si *withyear* es verdadero, el año será incluido en el encabezado, de lo contrario sólo se usará el nombre del mes.

formatyear(*theyear*, *width=3*)

Retorna el calendario de un año como una tabla HTML.

width (por defecto a 3) especifica el número de meses por fila.

formatyearpage(*theyear*, *width=3*, *css='calendar.css'*, *encoding=None*)

Retorna el calendario de un año como una página HTML completa. *width* (por defecto a 3) especifica el número de meses por fila. *css* es el nombre de la hoja de estilo en cascada que se debe usar. `None` puede ser pasada si no se debe usar una hoja de estilo. *encoding* especifica la codificación a ser usada para la salida (por defecto a la codificación por defecto del sistema).

formatmonthname(*theyear*, *themonth*, *withyear=True*)

Return a month name as an HTML table row. If *withyear* is true the year will be included in the row, otherwise just the month name will be used.

`HTMLCalendar` tiene los siguientes atributos que puedes sobrescribir para personalizar las clases CSS utilizadas por el calendario:

cssclasses

Una lista de clases CSS utilizadas para cada día de la semana. La lista de clases predeterminada es:

```
cssclasses = ["mon", "tue", "wed", "thu", "fri'
```

se pueden añadir más estilos para cada día:

```
cssclasses = ["mon text-bold", "tue", "wed", "t
```

Ten en cuenta que la longitud de esta lista debe ser de siete elementos.

cssclass_noday

La clase CSS para un día de la semana que ocurre en el mes anterior o siguiente.

Added in version 3.7.

cssclasses_weekday_head

Una lista de clases CSS utilizadas para los nombres de los días de la semana en la fila del encabezado. El valor por defecto es el mismo que **`cssclasses`**.

Added in version 3.7.

cssclass_month_head

La clase de CSS del mes (usada por **`formatmonthname()`**). El valor por defecto es `"month"`.

Added in version 3.7.

cssclass_month

La clase de CSS para la tabla de todo el mes (usada por **`formatmonth()`**). El valor por defecto es `"month"`.

Added in version 3.7.

cssclass_year

La clase de CSS para la tabla de tablas de todo el año (usada por **`formatyear()`**). El valor por defecto es `"year"`.

Added in version 3.7.

cssclass_year_head

La clase de CSS para el encabezado de la tabla para todo el año (usado por **`formatyear()`**). El valor por defecto es `"year"`.

Added in version 3.7.

Nótese que aunque la denominación de los atributos de clase descritos anteriormente es singular (por ejemplo, `cssclass_month` `cssclass_noday`), uno puede reemplazar la clase CSS única con una lista de clases CSS separadas por espacios, por ejemplo:

```
"text-bold text-red"
```

Aquí hay un ejemplo de cómo **`HTMLCalendar`** puede ser personalizado:

```
class CustomHTMLCal(calendar.HTMLCalendar):
    cssclasses = [style + " text-nowrap" for style
                  calendar.HTMLCalendar.cssclasses]
    cssclass_month_head = "text-center month-head"
    cssclass_month = "text-center month"
    cssclass_year = "text-italic lead"
```

class calendar.LocaleTextCalendar(*firstweekday=0*, *locale=None*)

Esta subclase de **`TextCalendar`** se le puede pasar un nombre de configuración regional en el constructor y retornará los nombres de los meses y días de la semana en la configuración regional especificada.

class calendar.LocaleHTMLCalendar(*firstweekday=0*, *locale=None*)

Esta subclase de **`HTMLCalendar`** se le puede pasar un nombre de configuración regional en el constructor y retornará los nombres de los meses y días de la semana en la configuración regional especificada.

Nota

The constructor, **`formatweekday()`** and **`formatmonthname()`** methods of these two classes temporarily change the `LC_TIME` locale to the given *locale*. Because the current locale is a process-wide setting, they are not thread-safe.

Para los calendarios de texto simples este módulo proporciona las siguientes funciones.

calendar.setfirstweekday(*weekday*)

Establece el día de la semana (el `0` es el lunes, el `6` es el domingo) para empezar cada semana. Los valores **`MONDAY`**, **`TUESDAY`**, **`WEDNESDAY`**, **`THURSDAY`**, **`FRIDAY`**, **`SATURDAY`**, y **`SUNDAY`** se proporcionan por conveniencia. Por ejemplo, para fijar el primer día de la semana en domingo:

```
import calendar
calendar.setfirstweekday(calendar.SUNDAY)
```

calendar.firstweekday()

Retorna la configuración actual para el día de la semana para empezar cada semana.

calendar.isleap(*year*)

Retorna **`True`** si *year* es un año bisiesto, si no **`False`**.

calendar.leapdays(*y1*, *y2*)

Retorna el número de años bisiestos en el rango de *y1* a *y2* (exclusivo), donde *y1* y *y2* son años.

Esta función opera para rangos que abarcan un cambio de siglo.

calendar.weekday(*year*, *month*, *day*)

Retorna el día de la semana (`0` es lunes) para *year* (`1970`–...), *month* (`1`–`12`), *day* (`1`–`31`).

calendar.weekheader(*n*)

Retorna un encabezado con nombres abreviados de los días de la semana. *n* especifica el ancho en caracteres de un día de la semana.

calendar.monthrange(*year*, *month*)

Returns weekday of first day of the month and number of days in month, for the specified *year* and *month*.

calendar.monthcalendar(*year*, *month*)

Retorna una matriz que representa el calendario de un mes. Cada fila representa una semana; los días fuera del mes se representan con ceros. Cada semana comienza con el lunes a menos que lo establezca `setfirstweekday()`.

calendar.prmonth(*theyear*, *themonth*, *w=0*, *l=0*)

Imprime el calendario de un mes según lo retorna `month()`.

calendar.month(*theyear*, *themonth*, *w=0*, *l=0*)

Returns a month's calendar in a multi-line string using the `formatmonth()` of the `TextCalendar` class.

calendar.prcal(*year*, *w=0*, *l=0*, *c=6*, *m=3*)

Imprime el calendario de todo un año como lo retorna `calendar()`.

calendar.calendar(*year*, *w=2*, *l=1*, *c=6*, *m=3*)

Returns a 3-column calendar for an entire year as a multi-line string using the `formatyear()` of the `TextCalendar` class.

calendar.timegm(*tuple*)

An unrelated but handy function that takes a time tuple such as returned by the `gmtime()` function in the `time` module, and returns the corresponding Unix timestamp value, assuming an epoch of 1970, and the POSIX encoding. In fact, `time.gmtime()` and `timegm()` are each other's inverse.

The `calendar` module exports the following data attributes:

calendar.day_name

A sequence that represents the days of the week in the current locale, where Monday is day number 0.

```
>>> import calendar
>>> list(calendar.day_name)
```

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Fri
```

calendar.day_abbr

A sequence that represents the abbreviated days of the week in the current locale, where Mon is day number 0.

```
>>> import calendar
>>> list(calendar.day_abbr)
['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
```

calendar.MONDAY
calendar.TUESDAY
calendar.WEDNESDAY
calendar.THURSDAY
calendar.FRIDAY
calendar.SATURDAY
calendar.SUNDAY

Alias para los días de la semana, donde `MONDAY` es `0` y `SUNDAY` es `6`.

Added in version 3.12.

class calendar.Day

Enumeración que define los días de la semana como constantes enteras. Los miembros de esta enumeración se exportan al alcance del módulo como **`MONDAY`** hasta **`SUNDAY`**.

Added in version 3.12.

calendar.month_name

A sequence that represents the months of the year in the current locale. This follows normal convention of January being month number 1, so it has a length of 13 and `month_name[0]` is the empty string.

```
>>> import calendar
>>> list(calendar.month_name)
['', 'January', 'February', 'March', 'April', 'May'
```

calendar.month_abbr

A sequence that represents the abbreviated months of the year in the current locale. This follows normal convention of January being month number 1, so it has a length of 13 and `month_abbr[0]` is the empty string.

```
>>> import calendar
>>> list(calendar.month_abbr)
['', 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul
```

calendar.JANUARY
calendar.FEBRUARY
calendar.MARCH
calendar.APRIL
calendar.MAY
calendar.JUNE
calendar.JULY
calendar.AUGUST
calendar.SEPTEMBER
calendar.OCTOBER
calendar.NOVEMBER
calendar.DECEMBER

Alias para los meses del año, donde `JANUARY` es `1` y `DECEMBER` es `12`.

Added in version 3.12.

class calendar.Month

Enumeración que define los meses del año como constantes enteras. Los miembros de esta enumeración se exportan al alcance del módulo como **`JANUARY`** hasta **`DECEMBER`**.

Added in version 3.12.

The **`calendar`** module defines the following exceptions:

exception calendar.IllegalMonthError(*month*)

A subclass of **`ValueError`** and **`IndexError`**, raised when the given month number is outside of the range 1-12 (inclusive).

Distinto en la versión 3.12: `IllegalMonthError` is now also a subclass of `ValueError`. New code should avoid catching `IndexError`.

month

El número del mes no válido.

exception calendar.IllegalWeekdayError(*weekday*)

Una subclase de `ValueError`, que se lanza cuando el número del día de la semana proporcionado está fuera del rango 0-6 (inclusive).

weekday

El número de día de la semana laborable no válido.

Ver también

Módulo `datetime`

Interfaz orientada a objetos para fechas y horas con una funcionalidad similar a la del módulo `time`.

Módulo `time`

Funciones de bajo nivel relacionadas con el tiempo.

Command-line usage

Added in version 2.5.

The `calendar` module can be executed as a script from the command line to interactively print a calendar.

```
python -m calendar [-h] [-L LOCALE] [-e ENCODING] [-t {t
                   [-w WIDTH] [-l LINES] [-s SPACING] [-
                   [-f FIRST_WEEKDAY] [year] [month]
```

Por ejemplo, para imprimir un calendario para el año 2000:

```
$ python -m calendar 2000
                                  2000

      January                   February
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo T
                1  2          1  2  3  4  5  6
 3  4  5  6  7  8  9       7  8  9 10 11 12 13       6
10 11 12 13 14 15 16      14 15 16 17 18 19 20      13 1
17 18 19 20 21 22 23      21 22 23 24 25 26 27      20 2
24 25 26 27 28 29 30      28 29                     27 2
31

       April                      May
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo T
                1  2       1  2  3  4  5  6  7
 3  4  5  6  7  8  9       8  9 10 11 12 13 14       5
10 11 12 13 14 15 16      15 16 17 18 19 20 21      12 1
17 18 19 20 21 22 23      22 23 24 25 26 27 28      19 2
24 25 26 27 28 29 30      29 30 31                  26 2

        July                     August
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo T
                1  2          1  2  3  4  5  6
 3  4  5  6  7  8  9       7  8  9 10 11 12 13       4
10 11 12 13 14 15 16      14 15 16 17 18 19 20      11 1
17 18 19 20 21 22 23      21 22 23 24 25 26 27      18 1
24 25 26 27 28 29 30      28 29 30 31               25 2
31

      October                   November
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo T
                   1             1  2  3  4  5
 2  3  4  5  6  7  8       6  7  8  9 10 11 12       4
 9 10 11 12 13 14 15      13 14 15 16 17 18 19      11 1
16 17 18 19 20 21 22      20 21 22 23 24 25 26      18 1
23 24 25 26 27 28 29      27 28 29 30               25 2
30 31
```

Se aceptan las siguientes opciones:

--help, -h

Muestra el mensaje de ayuda y salida.

--locale LOCALE, -L LOCALE

La configuración regional que se utiliza para los nombres de meses y días de la semana. El valor predeterminado es inglés.

--encoding ENCODING, -e ENCODING

La codificación que se utiliza para la salida. Se necesita **`--encoding`** si **`--locale`** está configurada.

--type {text,html}, -t {text,html}

Imprime el calendario en la terminal como texto o como un documento HTML.

--first-weekday FIRST_WEEKDAY, -f FIRST_WEEKDAY

The weekday to start each week. Must be a number between 0 (Monday) and 6 (Sunday). Defaults to 0.

Added in version 3.13.

year

The year to print the calendar for. Defaults to the current year.

month

El mes de **`year`** especificado para imprimir el calendario. Debe ser un número entre 1 y 12 y sólo se puede usar en modo texto. El valor predeterminado es imprimir un calendario para el año completo.

Opciones de modo texto:

--width WIDTH, -w WIDTH

El ancho de la columna de fecha en las columnas terminales. La fecha se imprime centrada en la columna. Cualquier valor inferior a 2 se ignora. El valor predeterminado es 2.

--lines LINES, -l LINES

El número de líneas para cada semana en filas terminales. La fecha se imprime alineada en la parte superior. Cualquier valor inferior a 1 se ignora. El valor predeterminado es 1.

--spacing SPACING, -s SPACING

El espacio entre meses en columnas. Cualquier valor inferior a 2 se ignora. El valor predeterminado es 6.

--months MONTHS, -m MONTHS

El número de meses se imprimen por fila. El valor predeterminado es 3.

Distinto en la versión 3.14: By default, today's date is highlighted in color and can be controlled using environment variables.

Opciones de modo HTML:

--css CSS, -c CSS

La ruta de una hoja de estilos CSS que se utiliza para el calendario. Esta debe ser relativa al HTML generado, o un HTTP absoluto o una URL `file:///`.

`collections` — Container datatypes

Source code: Lib/collections/__init__.py [https://github.com/python/cpython/tree/3.14/Lib/collections/__init__.py]

Este módulo implementa tipos de datos de contenedores especializados que proporcionan alternativas a los contenedores integrados de uso general de Python, `dict`, `list`, `set`, and `tuple`.

`namedtuple()`	función *factory* para crear subclases de *tuplas* con campos con nombre
`deque`	contenedor similar a una lista con *appends* y *pops* rápidos en ambos extremos
`ChainMap`	clase similar a *dict* para crear una vista única de múltiples *mapeados*
`Counter`	subclase de *dict* para contar objetos hashable
`OrderedDict`	subclase de *dict* que recuerda las entradas de la orden que se agregaron
`defaultdict`	subclase de *dict* que llama a una función de *factory* para suministrar valores faltantes
`UserDict`	envoltura alrededor de los objetos de diccionario para facilitar subclasificaciones *dict*
`UserList`	envoltura alrededor de los objetos de lista para facilitar la subclasificación de un *list*
`UserString`	envoltura alrededor de objetos de cadena para facilitar la subclasificación de *string*

Objetos `ChainMap`

Added in version 3.3.

Una clase `ChainMap` se proporciona para vincular rápidamente una serie de *mappings* de modo que puedan tratarse como una sola unidad. Suele ser mucho más rápido que crear un diccionario nuevo y ejecutar varias llamadas a `update()`.

La clase se puede utilizar para simular ámbitos anidados y es útil para crear plantillas.

class collections.ChainMap(**maps*)

Un `ChainMap` agrupa varios diccionarios u otros *mappings* para crear una vista única y actualizable. Si no se especifican *maps*, se proporciona un solo diccionario vacío para que una nueva cadena siempre tenga al menos un *mapeo*.

Las asignaciones subyacentes se almacenan en una lista. Esa lista es pública y se puede acceder a ella o actualizarla usando el atributo *maps*. No hay otro estado.

Las búsquedas buscan los mapeos subyacentes sucesivamente hasta que se encuentra una clave. Por el contrario, las escrituras, actualizaciones y eliminaciones solo operan en el primer *mapeo*.

Un `ChainMap` incorpora los mapeos subyacentes por referencia. Entonces, si una de los mapeos subyacentes se actualiza, esos cambios se reflejarán en `ChainMap`.

Se admiten todos los métodos habituales de un diccionario. Además, hay un atributo *maps*, un método para crear nuevos sub contextos y una propiedad para acceder a todos menos al primer mapeo:

maps

Una lista de mapeos actualizable por el usuario. La lista está ordenada desde la primera búsqueda hasta la última búsqueda. Es el único estado almacenado y se puede modificar para cambiar los mapeos que se buscan. La lista siempre debe contener al menos un mapeo.

new_child(*m=None*, ***kwargs*)

Retorna un nuevo **`ChainMap`** conteniendo un nuevo mapa seguido de todos los mapas de la instancia actual. Si se especifica `m`, se convierte en el nuevo mapa al principio de la lista de asignaciones; si no se especifica, se usa un diccionario vacío, de modo que una llamada a `d.new_child()` es equivalente a: `ChainMap({}, *d.maps)`. Si se especifica algún argumento de palabra clave, actualiza el mapa pasado o el nuevo diccionario vacío. Este método se utiliza para crear sub contextos que se pueden actualizar sin alterar los valores en ninguna de los mapeos padre.

Distinto en la versión 3.4: Se agregó el parámetro opcional `m`.

Distinto en la versión 3.10: Se añadió soporte para argumentos por palabras clave.

parents

Propiedad que retorna un nuevo **`ChainMap`** conteniendo todos los mapas de la instancia actual excepto el primero. Esto es útil para omitir el primer mapa en la búsqueda. Los casos de uso son similares a los de **`nonlocal`** la palabra clave usada en alcances anidados. Los casos de uso también son paralelos a los de la función incorporada **`super()`**. Una referencia a `d.parents` es equivalente a: `ChainMap(*d.maps[1:])`.

Note, the iteration order of a **`ChainMap`** is determined by scanning the mappings last to first:

```
>>> baseline = {'music': 'bach', 'art': 'rembrandt'
>>> adjustments = {'art': 'van gogh', 'opera': 'car
>>> list(ChainMap(adjustments, baseline))
['music', 'art', 'opera']
```

Esto da el mismo orden que una serie de llamadas a **`dict.update()`** comenzando con el último mapeo:

```
>>> combined = baseline.copy()
>>> combined.update(adjustments)
>>> list(combined)
['music', 'art', 'opera']
```

Distinto en la versión 3.9: Se agregó soporte para los operadores `|` y `|=`, especificados en **PEP 584** [https://peps.python.org/pep-0584/].

Ver también

- La clase MultiContext [https://github.com/enthought/codetools/blob/4.0.0/codetools/contexts/multi_context.py] en el paquete de Enthought llamado CodeTools [https://github.com/enthought/codetools] tiene opciones para admitir la escritura en cualquier mapeo de la cadena.
- La clase de Django Context [https://github.com/django/django/blob/master/django/template/context.py] para crear plantillas es una cadena de mapeo de solo lectura. También presenta características de pushing y popping de contextos similar al método `new_child()` y a la propiedad `parents`.
- The Nested Contexts recipe [https://code.activestate.com/recipes/577434-nested-contexts-a-chain-of-mapping-objects/] has options to control whether writes and other mutations apply only to the first mapping or to any mapping in the chain.
- Una versión de solo lectura muy simplificada de Chainmap [https://code.activestate.com/recipes/305268/].

Ejemplos y recetas `ChainMap`

Esta sección muestra varios enfoques para trabajar con mapas encadenados.

Ejemplo de simulación de la cadena de búsqueda interna de Python:

```
import builtins
pylookup = ChainMap(locals(), globals(), vars(builtins))
```

Ejemplo de dejar que los argumentos de la línea de comandos especificados por el usuario tengan prioridad sobre las variables de entorno que, a su vez, tienen prioridad sobre los valores predeterminados:

```
import os, argparse

defaults = {'color': 'red', 'user': 'guest'}

parser = argparse.ArgumentParser()
parser.add_argument('-u', '--user')
parser.add_argument('-c', '--color')
namespace = parser.parse_args()
command_line_args = {k: v for k, v in vars(namespace).it

combined = ChainMap(command_line_args, os.environ, defau
print(combined['color'])
print(combined['user'])
```

Patrones de ejemplo para usar la clase **`ChainMap`** para simular contextos anidados:

```
c = ChainMap()          # Create root context
d = c.new_child()       # Create nested child context
e = c.new_child()       # Child of c, independent from d
e.maps[0]               # Current context dictionary -- li
e.maps[-1]              # Root context -- like Python's gl
e.parents               # Enclosing context chain -- like

d['x'] = 1              # Set value in current context
d['x']                  # Get first key in the chain of co
del d['x']              # Delete from current context
list(d)                 # All nested values
k in d                  # Check all nested values
len(d)                  # Number of nested values
d.items()               # All nested items
dict(d)                 # Flatten into a regular dictionar
```

La clase **`ChainMap`** solo realiza actualizaciones (escrituras y eliminaciones) en el primer mapeo de la cadena, mientras que las búsquedas buscarán en la cadena completa. Sin embargo, si se desean escrituras y eliminaciones profundas, es fácil crear una subclase que actualice las claves que se encuentran más profundas en la cadena:

```
class DeepChainMap(ChainMap):
    'Variant of ChainMap that allows direct updates to i

    def __setitem__(self, key, value):
        for mapping in self.maps:
            if key in mapping:
                mapping[key] = value
                return
        self.maps[0][key] = value

    def __delitem__(self, key):
        for mapping in self.maps:
            if key in mapping:
                del mapping[key]
                return
        raise KeyError(key)

>>> d = DeepChainMap({'zebra': 'black'}, {'elephant': 'b
>>> d['lion'] = 'orange'         # update an existing ke
>>> d['snake'] = 'red'           # new keys get added to
>>> del d['elephant']            # remove an existing ke
>>> d                            # display result
DeepChainMap({'zebra': 'black', 'snake': 'red'}, {}, {'l
```

Objetos `Counter`

Se proporciona una herramienta de contador para respaldar recuentos rápidos y convenientes. Por ejemplo:

```
>>> # Tally occurrences of words in a list
>>> cnt = Counter()
>>> for word in ['red', 'blue', 'red', 'green', 'blue',
...     cnt[word] += 1
```

```
...
>>> cnt
Counter({'blue': 3, 'red': 2, 'green': 1})

>>> # Find the ten most common words in Hamlet
>>> import re
>>> words = re.findall(r'\w+', open('hamlet.txt').read()
>>> Counter(words).most_common(10)
[('the', 1143), ('and', 966), ('to', 762), ('of', 669),
 ('you', 554),  ('a', 546), ('my', 514), ('hamlet', 471)
```

class collections.Counter(***kwargs*)
class collections.Counter(*iterable*, /, ***kwargs*)
class collections.Counter(*mapping*, /, ***kwargs*)

Una clase `Counter` es una subclase `dict` para contar objetos hashable. Es una colección donde los elementos se almacenan como claves de diccionario y sus conteos se almacenan como valores de diccionario. Se permite que los conteos sean cualquier valor entero, incluidos los conteos de cero o negativos. La clase `Counter` es similar a los *bags* o multiconjuntos en otros idiomas.

Los elementos se cuentan desde un *iterable* o se inicializan desde otro *mapeo* (o contador):

```
>>> c = Counter()                           # a new
>>> c = Counter('gallahad')                 # a new
>>> c = Counter({'red': 4, 'blue': 2})      # a new
>>> c = Counter(cats=4, dogs=8)             # a new
```

Los objetos Counter tienen una interfaz de diccionario, excepto que retornan un conteo de cero para los elementos faltantes en lugar de levantar una `KeyError`:

```
>>> c = Counter(['eggs', 'ham'])
>>> c['bacon']                              # count
0
```

Establecer un conteo en cero no elimina un elemento de un contador. Utilice `del` para eliminarlo por completo:

```
>>> c['sausage'] = 0                        # count
>>> del c['sausage']                        # del a
```

Added in version 3.1.

Distinto en la versión 3.7: Como subclase de **`dict`**, **`Counter`** heredó la capacidad de recordar el orden de inserción. Las operaciones matemáticas en objetos *Counter* también preservan el orden. Los resultados se ordenan en función de cuándo se encuentra un elemento por primera vez en el operando izquierdo y, a continuación, por el orden en que se encuentra en el operando derecho.

Los objetos Counter admiten métodos adicionales a los disponibles para todos los diccionarios:

elements()

Retorna un iterador sobre los elementos que se repiten tantas veces como su conteo. Los elementos se retornan en el orden en que se encontraron por primera vez. Si el conteo de un elemento es menor que uno, **`elements()`** lo ignorará.

```
>>> c = Counter(a=4, b=2, c=0, d=-2)
>>> sorted(c.elements())
['a', 'a', 'a', 'a', 'b', 'b']
```

most_common(*n*=*None*)

Retorna una lista de los *n* elementos mas comunes y sus conteos, del mas común al menos común. Si se omite *n* o `None`, **`most_common()`** retorna *todos* los elementos del contador. Los elementos con conteos iguales se ordenan en el orden en que se encontraron por primera vez:

```
>>> Counter('abracadabra').most_common(3)
[('a', 5), ('b', 2), ('r', 2)]
```

subtract(***kwargs*)

subtract(*iterable*, /, ***kwargs*)

subtract(*mapping*, /, ***kwargs*)

Los elementos se restan de un *iterable* o de otro *mapeo* (o contador). Como `dict.update()` pero resta los conteos en lugar de reemplazarlos. Tanto las entradas como las salidas pueden ser cero o negativas.

```
>>> c = Counter(a=4, b=2, c=0, d=-2)
>>> d = Counter(a=1, b=2, c=3, d=4)
>>> c.subtract(d)
>>> c
Counter({'a': 3, 'b': 0, 'c': -3, 'd': -6})
```

Added in version 3.2.

total()

Computa la suma de las cuentas.

```
>>> c = Counter(a=10, b=5, c=0)
>>> c.total()
15
```

Added in version 3.10.

Los métodos de diccionario habituales están disponibles para objetos `Counter` excepto dos que funcionan de manera diferente para los contadores.

fromkeys(*iterable*)

Este método de clase no está implementado para objetos `Counter` .

update(***kwargs*)

update(*iterable*, /, ***kwargs*)

update(*mapping*, /, ***kwargs*)

Los elementos se cuentan desde un *iterable* o agregados desde otro *mapeo* (o contador). Como `dict.update()` pero agrega conteos en lugar de reemplazarlos. Además, se espera que el *iterable* sea una secuencia de elementos, no una secuencia de parejas `(clave, valor)`.

Los contadores admiten operadores de comparación ricos para las relaciones de igualdad, subconjunto, y superconjunto: `==`, `!=`, `<`, `<=`, `>`, `>=`. Todas esas pruebas tratan los elementos faltantes como si tuvieran cero recuentos, por lo que `Counter(a=1) == Counter(a=1, b=0)` retorne verdadero.

Distinto en la versión 3.10: Se han añadido operaciones de comparación enriquecidas.

Distinto en la versión 3.10: En pruebas de igualdad, los elementos faltantes son tratados como si tuvieran cero recuentos. Anteriormente, `Counter(a=3)` y `Counter(a=3, b=0)` se consideraban distintos.

Patrones comunes para trabajar con objetos `Counter`:

```
c.total()                       # total of all counts
c.clear()                       # reset all counts
list(c)                         # list unique elements
set(c)                          # convert to a set
dict(c)                         # convert to a regular c
c.items()                       # access the (elem, cnt)
Counter(dict(list_of_pairs))    # convert from a list of
c.most_common()[:-n-1:-1]       # n least common element
+c                              # remove zero and negati
```

Existen varias operaciones matemáticas que permiten combinar objetos `Counter` para producir conjuntos múltiples (contadores con recuentos superiores a cero). La suma y la resta combinan contadores sumando o restando los recuentos de los elementos correspondientes. La intersección y la unión retornan el mínimo y el máximo de los recuentos correspondientes. La igualdad y la inclusión comparan los recuentos correspondientes. Cada operación puede aceptar entradas con recuentos con signo, pero la salida excluirá los resultados con recuentos iguales o inferiores a cero.

```
>>> c = Counter(a=3, b=1)
>>> d = Counter(a=1, b=2)
>>> c + d                       # add two counters toget
Counter({'a': 4, 'b': 3})
>>> c - d                       # subtract (keeping only
Counter({'a': 2})
>>> c & d                       # intersection:  min(c[x
Counter({'a': 1, 'b': 1})
>>> c | d                       # union:  max(c[x], d[x]
Counter({'a': 3, 'b': 2})
>>> c == d                      # equality:  c[x] == d[x
False
>>> c <= d                      # inclusion:  c[x] <= d[
False
```

La suma y resta unaria son atajos para agregar un contador vacío o restar de un contador vacío.

```
>>> c = Counter(a=2, b=-4)
>>> +c
Counter({'a': 2})
>>> -c
Counter({'b': 4})
```

Added in version 3.3: Se agregó soporte para operaciones unarias de adición, resta y multiconjunto en su lugar (*in-place*).

Nota

Los Counters se diseñaron principalmente para trabajar con números enteros positivos para representar conteos continuos; sin embargo, se tuvo cuidado de no excluir innecesariamente los casos de uso que necesitan otros tipos o valores negativos. Para ayudar con esos casos de uso, esta sección documenta el rango mínimo y las restricciones de tipo.

- La clase `Counter` en sí misma es una subclase de diccionario sin restricciones en sus claves y valores. Los valores están pensados para ser números que representan conteos, pero *podría* almacenar cualquier cosa en el campo de valor.

- El método `most_common()` solo requiere que los valores se puedan ordenar.
- Para operaciones en su lugar (*in-place*) como `c[key] += 1`, el tipo de valor solo necesita admitir la suma y la resta. Por lo tanto, las fracciones, flotantes y decimales funcionarían y se admiten valores negativos. Lo mismo ocurre con `update()` y `subtract()` que permiten valores negativos y cero para las entradas y salidas.
- Los métodos de multiconjuntos están diseñados solo para casos de uso con valores positivos. Las entradas pueden ser negativas o cero, pero solo se crean salidas con valores positivos. No hay restricciones de tipo, pero el tipo de valor debe admitir la suma, la resta y la comparación.
- El método `elements()` requiere conteos enteros. Ignora los conteos de cero y negativos.

Ver también

- Clase Bag [https://www.gnu.org/software/smalltalk/manual-base/html_node/Bag.html] en Smalltalk.
- Entrada de Wikipedia para Multiconjuntos [https://es.wikipedia.org/wiki/Multiconjunto].
- Tutorial de multiconjuntos de C++ [http://www.java2s.com/Tutorial/Cpp/0380_set-multiset/Catalog0380_set-multiset.htm] con ejemplos.
- Para operaciones matemáticas en multiconjuntos y sus casos de uso, consulte *Knuth, Donald. The Art of Computer Programming Volume II, Sección 4.6.3, Ejercicio 19*.
- Para enumerar todos los distintos multiconjuntos de un tamaño dado sobre un conjunto dado de elementos, consulte `itertools.combinations_with_replacement()`:

```
map(Counter, combinations_with_replacement('ABC',
```

Objetos `deque`

class collections.deque([*iterable*[, *maxlen*]])

Retorna un nuevo objeto deque inicializado de izquierda a derecha (usando `append()`) con datos de *iterable*. Si no se especifica *iterable*, el nuevo deque estará vacío.

Deques are a generalization of stacks and queues (the name is pronounced «deck» and is short for «double-ended queue»). Deques support thread-safe, memory efficient appends and pops from either side of the deque with approximately the same $O(1)$ performance in either direction.

Though `list` objects support similar operations, they are optimized for fast fixed-length operations and incur $O(n)$ memory movement costs for `pop(0)` and `insert(0, v)` operations which change both the size and position of the underlying data representation.

Si no se especifica *maxlen* o es `None`, los deques pueden crecer hasta una longitud arbitraria. De lo contrario, el deque está limitado a la longitud máxima especificada. Una vez que un deque de longitud limitada esta lleno, cuando se agregan nuevos elementos, se descarta el número correspondiente de elementos del extremo opuesto. Los deques de longitud limitada proporcionan una funcionalidad similar al filtro `tail` en Unix. También son útiles para rastrear transacciones y otros grupos de datos donde solo la actividad más reciente es de interés.

Los objetos deque admiten los siguientes métodos:

append(*item*, */*)

Add *item* to the right side of the deque.

appendleft(*item*, */*)

Add *item* to the left side of the deque.

clear()

Retire todos los elementos del deque dejándolo con longitud 0.

copy()

Crea una copia superficial del deque.

Added in version 3.5.

count(*value*, */*)

Count the number of deque elements equal to *value*.

Added in version 3.2.

extend(*iterable*, */*)

Extienda el lado derecho del deque agregando elementos del argumento iterable.

extendleft(*iterable*, */*)

Extienda el lado izquierdo del deque agregando elementos de *iterable*. Tenga en cuenta que la serie de appends a la izquierda da como resultado la inversión del orden de los elementos en el argumento iterable.

index(*value*[, *start*[, *stop*]])

Return the position of *value* in the deque (at or after index *start* and before index *stop*). Returns the first match or raises `ValueError` if not found.

Added in version 3.5.

insert(*index*, *value*, */*)

Insert *value* into the deque at position *index*.

Si la inserción causara que un deque limitado crezca más allá de *maxlen*, se lanza un `IndexError`.

Added in version 3.5.

pop()

Elimina y retorna un elemento del lado derecho del deque. Si no hay elementos presentes, lanza un **`IndexError`**.

popleft()

Elimina y retorna un elemento del lado izquierdo del deque. Si no hay elementos presentes, lanza un **`IndexError`**.

remove(*value*, */*)

Elimina la primera aparición de *value*. Si no se encuentra, lanza un **`ValueError`**.

reverse()

Invierte los elementos del deque en su lugar (*in-place*) y luego retorna `None`.

Added in version 3.2.

rotate(*n*=*1*, */*)

Gira el deque *n* pasos a la derecha. Si *n* es negativo, lo gira hacia la izquierda.

Cuando el deque no está vacío, girar un paso hacia la derecha equivale a `d.appendleft(d.pop())`, y girar un paso hacia la izquierda equivale a `d.append(d.popleft())`.

Los objetos deque también proporcionan un atributo de solo lectura:

maxlen

Tamaño máximo de un deque o `None` si no está limitado.

Added in version 3.1.

In addition to the above, deques support iteration, pickling, `len(d)`, `reversed(d)`, `copy.copy(d)`, `copy.deepcopy(d)`, membership testing with the **`in`** operator, and subscript references such as `d[0]` to access the first element. Indexed access is $O(1)$ at both ends but slows to $O(n)$ in the middle. For fast random access, use lists instead.

A partir de la versión 3.5, los deques admiten `__add__()`, `__mul__()`, y `__imul__()`.

Ejemplo:

```
>>> from collections import deque
>>> d = deque('ghi')                 # make a new deque
>>> for elem in d:                   # iterate over the
...     print(elem.upper())
G
H
I

>>> d.append('j')                    # add a new entry t
>>> d.appendleft('f')                # add a new entry t
>>> d                                # show the represen
deque(['f', 'g', 'h', 'i', 'j'])

>>> d.pop()                          # return and remove
'j'
>>> d.popleft()                      # return and remove
'f'
>>> list(d)                          # list the contents
['g', 'h', 'i']
>>> d[0]                             # peek at leftmost
'g'
>>> d[-1]                            # peek at rightmost
'i'

>>> list(reversed(d))                # list the contents
```

```
['i', 'h', 'g']
>>> 'h' in d                         # search the deque
True
>>> d.extend('jkl')                  # add multiple elem
>>> d
deque(['g', 'h', 'i', 'j', 'k', 'l'])
>>> d.rotate(1)                      # right rotation
>>> d
deque(['l', 'g', 'h', 'i', 'j', 'k'])
>>> d.rotate(-1)                     # left rotation
>>> d
deque(['g', 'h', 'i', 'j', 'k', 'l'])

>>> deque(reversed(d))               # make a new deque
deque(['l', 'k', 'j', 'i', 'h', 'g'])
>>> d.clear()                        # empty the deque
>>> d.pop()                          # cannot pop from a
Traceback (most recent call last):
    File "<pyshell#6>", line 1, in -toplevel-
        d.pop()
IndexError: pop from an empty deque

>>> d.extendleft('abc')              # extendleft() reve
>>> d
deque(['c', 'b', 'a'])
```

Recetas `deque`

Esta sección muestra varios enfoques para trabajar con deques.

Los deques de longitud limitada proporcionan una funcionalidad similar al filtro `tail` en Unix:

```
def tail(filename, n=10):
    'Return the last n lines of a file'
    with open(filename) as f:
        return deque(f, n)
```

Otro enfoque para usar deques es mantener una secuencia de elementos agregados recientemente haciendo appending a la derecha y popping a la izquierda:

```
def moving_average(iterable, n=3):
    # moving_average([40, 30, 50, 46, 39, 44]) --> 40.0
    # https://en.wikipedia.org/wiki/Moving_average
    it = iter(iterable)
    d = deque(itertools.islice(it, n-1))
    d.appendleft(0)
    s = sum(d)
    for elem in it:
        s += elem - d.popleft()
        d.append(elem)
        yield s / n
```

Un scheduler round-robin [https://en.wikipedia.org/wiki/Round-robin_scheduling] se puede implementar con iteradores de entrada almacenados en **`deque`**. Los valores son producidos del iterador activo en la posición cero. Si ese iterador está agotado, se puede eliminar con **`popleft()`**; de lo contrario, se puede volver en ciclos al final con el método **`rotate()`**

```
def roundrobin(*iterables):
    "roundrobin('ABC', 'D', 'EF') --> A D E B F C"
    iterators = deque(map(iter, iterables))
    while iterators:
        try:
            while True:
                yield next(iterators[0])
                iterators.rotate(-1)
        except StopIteration:
            # Remove an exhausted iterator.
            iterators.popleft()
```

El método **`rotate()`** proporciona una forma de implementar eliminación y rebanado de **`deque`**. Por ejemplo, una implementación pura de Python de `del d[n]` se basa en el método `rotate()` para colocar los elementos que se van a extraer:

```
def delete_nth(d, n):
```

```
d.rotate(-n)
d.popleft()
d.rotate(n)
```

Para implementar el rebanado de un **`deque`**, use un enfoque similar aplicando **`rotate()`** para traer un elemento objetivo al lado izquierdo del deque. Elimine las entradas antiguas con **`popleft()`**, agregue nuevas entradas con **`extend()`**, y luego invierta la rotación. Con variaciones menores en ese enfoque, es fácil implementar manipulaciones de pila de estilo hacia adelante como `dup`, `drop`, `swap`, `over`, `pick`, `rot`, y `roll`.

Objetos `defaultdict`

class collections.defaultdict(*default_factory=None*, */*, ***kwargs*)
class collections.defaultdict(*default_factory*, *mapping*, */*, ***kwargs*)
class collections.defaultdict(*default_factory*, *iterable*, */*, ***kwargs*)

Retorna un nuevo objeto similar a un diccionario. **`defaultdict`** es una subclase de la clase incorporada **`dict`**. Anula un método y agrega una variable de instancia de escritura. La funcionalidad restante es la misma que para la clase **`dict`** y no está documentada aquí.

El primer argumento proporciona el valor inicial para el atributo **`default_factory`**; por defecto es `None`. Todos los argumentos restantes se tratan de la misma forma que si se pasaran al constructor **`dict`**, incluidos los argumentos de palabras clave.

Los objetos **`defaultdict`** admiten el siguiente método además de las operaciones estándar de **`dict`**:

__missing__(*key*, */*)

Si el atributo **`default_factory`** es `None`, lanza una excepción **`KeyError`** con la *key* como argumento.

Si **`default_factory`** no es `None`, se llama sin argumentos para proporcionar un valor predeterminado para la *key* dada, este valor se inserta en el diccionario para la *key* y se retorna.

Si llamar a `default_factory` lanza una excepción, esta excepción se propaga sin cambios.

This method is called by the `__getitem__()` method of the `dict` class when the requested key is not found; whatever it returns or raises is then returned or raised by `__getitem__()`.

Note that `__missing__()` is *not* called for any operations besides `__getitem__()`. This means that `get()` will, like normal dictionaries, return `None` as a default rather than using `default_factory`.

los objetos `defaultdict` admiten la siguiente variable de instancia:

default_factory

This attribute is used by the `__missing__()` method; it is initialized from the first argument to the constructor, if present, or to `None`, if absent.

Distinto en la versión 3.9: Se agregaron los operadores unir (|) y actualizar (|=), especificados en PEP 584 [https://peps.python.org/pep-0584/].

Ejemplos `defaultdict`

Usando `list` como `default_factory`, es fácil agrupar una secuencia de pares clave-valor en un diccionario de listas:

```
>>> s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('bl
>>> d = defaultdict(list)
>>> for k, v in s:
...     d[k].append(v)
...
>>> sorted(d.items())
[('blue', [2, 4]), ('red', [1]), ('yellow', [1, 3])]
```

Cuando se encuentra cada clave por primera vez, no está ya en el mapping; por lo que una entrada se crea automáticamente usando la función `default_factory` que retorna una `list` vacía. La operación `list.append()` luego adjunta el valor a la nueva lista. Cuando se vuelven a encontrar claves, la búsqueda procede normalmente (retornando la lista para esa clave) y la operación **`list.append()`** agrega otro valor a la lista. Esta técnica es más simple y rápida que una técnica equivalente usando `dict.setdefault()`:

```
>>> d = {}
>>> for k, v in s:
...     d.setdefault(k, []).append(v)
...
>>> sorted(d.items())
[('blue', [2, 4]), ('red', [1]), ('yellow', [1, 3])]
```

Establecer `default_factory` en `int` hace que `defaultdict` sea útil para contar (como un bag o multiconjunto en otros idiomas):

```
>>> s = 'mississippi'
>>> d = defaultdict(int)
>>> for k in s:
...     d[k] += 1
...
>>> sorted(d.items())
[('i', 4), ('m', 1), ('p', 2), ('s', 4)]
```

Cuando se encuentra una letra por primera vez, falta en el mapping, por lo que la función `default_factory` llama a `int()` para proporcionar una cuenta predeterminada de cero. La operación de incremento luego acumula el conteo de cada letra.

La función `int()` que siempre retorna cero es solo un caso especial de funciones constantes. Una forma más rápida y flexible de crear funciones constantes es utilizar una función lambda que pueda proporcionar cualquier valor constante (no solo cero):

```
>>> def constant_factory(value):
```

```
...     return lambda: value
...
>>> d = defaultdict(constant_factory('<missing>'))
>>> d.update(name='John', action='ran')
>>> '%(name)s %(action)s to %(object)s' % d
'John ran to <missing>'
```

Establecer **`default_factory`** en **`set`** hace que **`defaultdict`** sea útil para construir un diccionario de conjuntos:

```
>>> s = [('red', 1), ('blue', 2), ('red', 3), ('blue', 4
>>> d = defaultdict(set)
>>> for k, v in s:
...     d[k].add(v)
...
>>> sorted(d.items())
[('blue', {2, 4}), ('red', {1, 3})]
```

`namedtuple()` Funciones *Factory* para Tuplas y Campos con Nombres

Las tuplas con nombre asignan significado a cada posición en una tupla y permiten un código más legible y autodocumentado. Se pueden usar donde se usen tuplas regulares y agregan la capacidad de acceder a los campos por nombre en lugar del índice de posición.

collections.namedtuple(*typename*, *field_names*, *, *rename=False*, *defaults=None*, *module=None*)

> Returns a new tuple subclass named *typename*. The new subclass is used to create tuple-like objects that have fields accessible by attribute lookup as well as being indexable and iterable. Instances of the subclass also have a helpful docstring (with *typename* and *field_names*) and a helpful **`__repr__()`** method which lists the tuple contents in a `name=value` format.
>
> Los *nombres de campo* son una secuencia de cadenas como `[‘x’, ‘y’]`. Alternativamente, *nombres de campo* puede ser

una sola cadena con cada nombre de campo separado por espacios en blanco y/o comas, por ejemplo `'x y'` or `'x, y'`.

Se puede usar cualquier identificador de Python válido para un *fieldname*, excepto para los nombres que comienzan con un guión bajo. Los identificadores válidos constan de letras, dígitos y guiones bajos, pero no comienzan con un dígito o guion bajo y no pueden ser **`keyword`** como *class*, *for*, *return*, *global*, *pass*, o *raise*.

Si *rename* es verdadero, los nombres de campo no válidos se reemplazan automáticamente con nombres posicionales. Por ejemplo, `['abc', 'def', 'ghi', 'abc']` se convierte en `['abc', '_1', 'ghi', '_3']`, eliminando la palabra clave `def` y el nombre de campo duplicado `abc`.

defaults pueden ser `None` o un iterable de los valores predeterminados. Dado que los campos con un valor predeterminado deben venir después de cualquier campo sin un valor predeterminado, los *defaults* se aplican a los parámetros situados más a la derecha. Por ejemplo, si los nombres de campo son `['x', 'y', 'z']` y los valores predeterminados son `(1, 2)`, entonces `x` será un argumento obligatorio, `y` tendrá el valor predeterminado de `1`, y `z` el valor predeterminado de `2`.

If *module* is defined, the **`__module__`** attribute of the named tuple is set to that value.

Las instancias de tuplas con nombre no tienen diccionarios por instancia, por lo que son livianas y no requieren más memoria que las tuplas normales.

Para admitir el serializado (*pickling*), la clase tupla con nombre debe asignarse a una variable que coincida con *typename*.

Distinto en la versión 3.1: Se agregó soporte para *rename*.

Distinto en la versión 3.6: Los parámetros *verbose* y *rename* se convirtieron en argumentos de solo palabra clave.

Distinto en la versión 3.6: Se agregó el parámetro *module*.

Distinto en la versión 3.7: Removed the *verbose* parameter and the **`_source`** attribute.

Distinto en la versión 3.7: Added the *defaults* parameter and the **`_field_defaults`** attribute.

```
>>> # Basic example
>>> Point = namedtuple('Point', ['x', 'y'])
>>> p = Point(11, y=22)      # instantiate with positiona
>>> p[0] + p[1]              # indexable like the plain t
33
>>> x, y = p                 # unpack like a regular tupl
>>> x, y
(11, 22)
>>> p.x + p.y                # fields also accessible by
33
>>> p                        # readable __repr__ with a n
Point(x=11, y=22)
```

Las tuplas con nombre son especialmente útiles para asignar nombres de campo a las tuplas de resultado retornadas por los módulos **`csv`** o **`sqlite3`**:

```
EmployeeRecord = namedtuple('EmployeeRecord', 'name, age

import csv
for emp in map(EmployeeRecord._make, csv.reader(open("em
    print(emp.name, emp.title)

import sqlite3
conn = sqlite3.connect('/companydata')
cursor = conn.cursor()
cursor.execute('SELECT name, age, title, department, pay
for emp in map(EmployeeRecord._make, cursor.fetchall()):
    print(emp.name, emp.title)
```

Además de los métodos heredados de las tuplas, las tuplas con nombre admiten tres métodos adicionales y dos atributos. Para evitar conflictos con los nombres de campo, los nombres de método y atributo comienzan con un guión bajo.

classmethod somenamedtuple._make(*iterable*, */*)

Método de clase que crea una nueva instancia a partir de una secuencia existente o iterable.

```
>>> t = [11, 22]
>>> Point._make(t)
Point(x=11, y=22)
```

somenamedtuple._asdict()

Retorna un nuevo `dict` que asigna los nombres de los campos a sus valores correspondientes:

```
>>> p = Point(x=11, y=22)
>>> p._asdict()
{'x': 11, 'y': 22}
```

Distinto en la versión 3.1: Retorna un `OrderedDict` en lugar de un `dict` regular.

Distinto en la versión 3.8: Retorna un `dict` normal en lugar de un `OrderedDict`. A partir de Python 3.7, se garantiza el orden de los diccionarios normales. Si se requieren las características adicionales de **`OrderedDict`** , la corrección sugerida es emitir el resultado al tipo deseado: `OrderedDict(nt._asdict())`.

somenamedtuple._replace(***kwargs*)

Retorna una nueva instancia de la tupla nombrada reemplazando los campos especificados con nuevos valores:

```
>>> p = Point(x=11, y=22)
>>> p._replace(x=33)
Point(x=33, y=22)
```

```
>>> for partnum, record in inventory.items():
...     inventory[partnum] = record._replace(price=
```

Named tuples are also supported by generic function **`copy.replace()`**.

Distinto en la versión 3.13: Raise **`TypeError`** instead of **`ValueError`** for invalid keyword arguments.

somenamedtuple._fields

Tupla de cadenas que lista los nombres de los campos. Útil para la introspección y para crear nuevos tipos de tuplas con nombre a partir de tuplas con nombre existentes.

```
>>> p._fields            # view the field names
('x', 'y')

>>> Color = namedtuple('Color', 'red green blue')
>>> Pixel = namedtuple('Pixel', Point._fields + Col
>>> Pixel(11, 22, 128, 255, 0)
Pixel(x=11, y=22, red=128, green=255, blue=0)
```

somenamedtuple._field_defaults

Diccionario de nombres de campos mapeados a valores predeterminados.

```
>>> Account = namedtuple('Account', ['type', 'balan
>>> Account._field_defaults
{'balance': 0}
>>> Account('premium')
Account(type='premium', balance=0)
```

Para recuperar un campo cuyo nombre está almacenado en una cadena, use la función **`getattr()`**:

```
>>> getattr(p, 'x')
11
```

Para convertir un diccionario en una tupla con nombre, use el operador de doble estrella (como se describe en Desempaquetando una lista de argumentos):

```
>>> d = {'x': 11, 'y': 22}
>>> Point(**d)
Point(x=11, y=22)
```

Dado que una tupla con nombre es una clase Python normal, es fácil agregar o cambiar la funcionalidad con una subclase. A continuación, se explica cómo agregar un campo calculado y un formato de impresión de ancho fijo:

```
>>> class Point(namedtuple('Point', ['x', 'y'])):
...     __slots__ = ()
...     @property
...     def hypot(self):
...         return (self.x ** 2 + self.y ** 2) ** 0.5
...     def __str__(self):
...         return 'Point: x=%6.3f  y=%6.3f  hypot=%6.3f

>>> for p in Point(3, 4), Point(14, 5/7):
...     print(p)
Point: x= 3.000  y= 4.000  hypot= 5.000
Point: x=14.000  y= 0.714  hypot=14.018
```

La subclase que se muestra arriba establece `__slots__` a una tupla vacía. Esto ayuda a mantener bajos los requisitos de memoria al evitar la creación de diccionarios de instancia.

La subclasificación no es útil para agregar campos nuevos almacenados. En su lugar, simplemente cree un nuevo tipo de tupla con nombre a partir del atributo **`_fields`**:

```
>>> Point3D = namedtuple('Point3D', Point._fields + ('z'
```

Los docstrings se pueden personalizar realizando asignaciones directas a los campos `__doc__`:

```
>>> Book = namedtuple('Book', ['id', 'title', 'authors']
>>> Book.__doc__ += ': Hardcover book in active collecti
>>> Book.id.__doc__ = '13-digit ISBN'
>>> Book.title.__doc__ = 'Title of first printing'
>>> Book.authors.__doc__ = 'List of authors sorted by la
```

Distinto en la versión 3.5: Los docstrings de propiedad se pueden escribir.

Ver también

- Consulte `typing.NamedTuple` para ver una forma de agregar sugerencias de tipo para tuplas con nombre. También proporciona una notación elegante usando la palabra clave `class`:

  ```
  class Component(NamedTuple):
      part_number: int
      weight: float
      description: Optional[str] = None
  ```

- Vea `types.SimpleNamespace()` para un espacio de nombres mutable basado en un diccionario subyacente en lugar de una tupla.
- El módulo `dataclasses` proporciona un decorador y funciones para agregar automáticamente métodos especiales generados a clases definidas por el usuario.

Objetos `OrderedDict`

Los diccionarios ordenados son como los diccionarios normales, pero tienen algunas capacidades adicionales relacionadas con las operaciones de ordenado. Se han vuelto menos importantes ahora que la clase incorporada `dict` ganó la capacidad de recordar el orden de inserción (este nuevo comportamiento quedó garantizado en Python 3.7).

Aún quedan algunas diferencias con `dict` :

- El `dict` normal fue diseñado para ser muy bueno en operaciones de mapeo. El seguimiento del pedido de inserción era secundario.

- La `OrderedDict` fue diseñada para ser buena para reordenar operaciones. La eficiencia del espacio, la velocidad de iteración y el rendimiento de las operaciones de actualización fueron secundarios.

- El algoritmo `OrderedDict` puede manejar operaciones frecuentes de re-ordenamiento mejor que `dict`. Como se muestra en las recetas siguientes, esto lo vuelve adecuado para implementar varios tipos de caches LRU.

- La operación de igualdad para `OrderedDict` comprueba el orden coincidente.

  Un `dict` regular puede emular la prueba de igualdad sensible al orden con `p == q and all(k1 == k2 for k1, k2 in zip(p, q))`.

- The `popitem()` method of `OrderedDict` has a different signature. It accepts an optional argument to specify which item is popped.

  Un `dict` regular puede emular `od.popitem(last=True)` de OrderedDict con `d.popitem()` que se garantiza que salte el elemento situado más a la derecha (el último).

  Un `dict` regular puede emular `od.popitem(last=False)` de OrderedDict con `(k := next(iter(d)), d.pop(k))` que retornará y eliminará el elemento situado más a la izquierda (el primero) si existe.

- `OrderedDict` has a `move_to_end()` method to efficiently reposition an element to an endpoint.

  Un `dict` regular puede emular `od.move_to_end(k, last=True)` de OrderedDict con `d[k] = d.pop(k)` que desplazará la clave y su valor asociado a la posición más a la derecha (última).

  Un `dict` regular no tiene un equivalente eficiente para `od.move_to_end(k, last=False)` de OrderedDict que desplaza la clave y su valor asociado a la posición más a la izquierda (primera).

- Until Python 3.8, **`dict`** lacked a **`__reversed__()`** method.

class collections.OrderedDict(***kwargs*)
class collections.OrderedDict(*mapping*, /, ***kwargs*)
class collections.OrderedDict(*iterable*, /, ***kwargs*)

Retorna una instancia de una subclase **`dict`** que tiene métodos especializados para reorganizar el orden del diccionario.

Added in version 3.1.

popitem(*last=True*)

El método **`popitem()`** para diccionarios ordenados retorna y elimina un par (clave, valor). Los pares se retornan en orden LIFO si el *último* es verdadero o en orden FIFO si es falso.

move_to_end(*key*, *last=True*)

Mueve una *clave* existente a cualquiera de los extremos de un diccionario ordenado. El elemento se mueve al extremo derecho si *last* es verdadero (por defecto) o al principio si *last* es falso. Lanza **`KeyError`** si la *clave* no existe:

```
>>> d = OrderedDict.fromkeys('abcde')
>>> d.move_to_end('b')
>>> ''.join(d)
'acdeb'
>>> d.move_to_end('b', last=False)
>>> ''.join(d)
'bacde'
```

Added in version 3.2.

Además de los métodos de mapeo habituales, los diccionarios ordenados también admiten la iteración inversa usando `reversed()`.

Equality tests between `OrderedDict` objects are order-sensitive and are roughly equivalent to `list(od1.items())==list(od2.items())`.

Equality tests between `OrderedDict` objects and other `Mapping` objects are order-insensitive like regular dictionaries. This allows `OrderedDict` objects to be substituted anywhere a regular dictionary is used.

Distinto en la versión 3.5: Los elementos, claves y valores vistas de `OrderedDict` ahora admiten la iteración inversa usando `reversed()`.

Distinto en la versión 3.6: With the acceptance of **PEP 468** [https://peps.python.org/pep-0468/], order is retained for keyword arguments passed to the `OrderedDict` constructor and its `update()` method.

Distinto en la versión 3.9: Se agregaron los operadores unir (|) y actualizar (|=), especificados en **PEP 584** [https://peps.python.org/pep-0584/].

Ejemplos y recetas `OrderedDict`

Es sencillo crear una variante de diccionario ordenado que recuerde el orden en que las claves se insertaron por *última vez*. Si una nueva entrada sobrescribe una entrada existente, la posición de inserción original se cambia y se mueve al final:

```
class LastUpdatedOrderedDict(OrderedDict):
    'Store items in the order the keys were last added'

    def __setitem__(self, key, value):
        super().__setitem__(key, value)
        self.move_to_end(key)
```

Un `OrderedDict` también sería útil para implementar variantes de `functools.lru_cache()`:

```
from collections import OrderedDict
from time import time

class TimeBoundedLRU:
    "LRU Cache that invalidates and refreshes old entrie

    def __init__(self, func, maxsize=128, maxage=30):
        self.cache = OrderedDict()      # { args : (time
        self.func = func
        self.maxsize = maxsize
        self.maxage = maxage

    def __call__(self, *args):
        if args in self.cache:
            self.cache.move_to_end(args)
            timestamp, result = self.cache[args]
            if time() - timestamp <= self.maxage:
                return result
        result = self.func(*args)
        self.cache[args] = time(), result
        if len(self.cache) > self.maxsize:
            self.cache.popitem(last=False)
        return result

class MultiHitLRUCache:
    """ LRU cache that defers caching a result until
        it has been requested multiple times.

        To avoid flushing the LRU cache with one-time re
        we don't cache until a request has been made mor

    """

    def __init__(self, func, maxsize=128, maxrequests=40
        self.requests = OrderedDict()   # { uncached_key
        self.cache = OrderedDict()      # { cached_key :
        self.func = func
        self.maxrequests = maxrequests  # max number of
        self.maxsize = maxsize          # max number of
        self.cache_after = cache_after

    def __call__(self, *args):
        if args in self.cache:
            self.cache.move_to_end(args)
            return self.cache[args]
        result = self.func(*args)
        self.requests[args] = self.requests.get(args, 0)
        if self.requests[args] <= self.cache_after:
            self.requests.move_to_end(args)
            if len(self.requests) > self.maxrequests:
                self.requests.popitem(last=False)
        else:
            self.requests.pop(args, None)
            self.cache[args] = result
            if len(self.cache) > self.maxsize:
                self.cache.popitem(last=False)
        return result
```

Objetos `UserDict`

La clase, `UserDict` actúa como un contenedor alrededor de los objetos del diccionario. La necesidad de esta clase ha sido parcialmente suplantada por la capacidad de crear subclases directamente desde `dict`; sin embargo, es más fácil trabajar con esta clase porque se puede acceder al diccionario subyacente como un atributo.

class collections.UserDict(***kwargs*)
class collections.UserDict(*mapping*, /, ***kwargs*)
class collections.UserDict(*iterable*, /, ***kwargs*)

> Class that simulates a dictionary. The instance's contents are kept in a regular dictionary, which is accessible via the `data` attribute of `UserDict` instances. If arguments are provided, they are used to initialize `data`, like a regular dictionary.
>
> In addition to supporting the methods and operations of mappings, `UserDict` instances provide the following

attribute:

data

Un diccionario real utilizado para almacenar el contenido de la clase `UserDict` .

Objetos `UserList`

Esta clase actúa como un contenedor alrededor de los objetos de lista. Es una clase base útil para tus propias clases tipo lista que pueden heredar de ellas y anular métodos existentes o agregar nuevos. De esta forma, se pueden agregar nuevos comportamientos a las listas.

La necesidad de esta clase ha sido parcialmente suplantada por la capacidad de crear subclases directamente desde `list`; sin embargo, es más fácil trabajar con esta clase porque se puede acceder a la lista subyacente como atributo.

class collections.UserList([*list*])

Clase que simula una lista. El contenido de la instancia se mantiene en una lista normal, a la que se puede acceder mediante el atributo `data` de las instancias `UserList`. El contenido de la instancia se establece inicialmente en una copia de *list*, por defecto a la lista vacía `[]`. *list* puede ser cualquier iterable, por ejemplo, una lista de Python real o un objeto **`UserList`**.

Además de admitir los métodos y operaciones de secuencias mutables, las instancias `UserList` proporcionan el siguiente atributo:

data

Un objeto real `list` usado para almacenar el contenido de la clase `UserList` .

Requisitos de subclases: Se espera que las subclases de `UserList` ofrezcan un constructor al que se pueda llamar sin argumentos o con un solo argumento. Las operaciones de lista que retornan una nueva secuencia intentan crear una instancia de la clase de implementación real. Para hacerlo, asume que se puede llamar al constructor con un solo parámetro, que es un objeto de secuencia utilizado como fuente de datos.

Si una clase derivada no desea cumplir con este requisito, todos los métodos especiales admitidos por esta clase deberán de anularse; consulte las fuentes para obtener información sobre los métodos que deben proporcionarse en ese caso.

Objetos `UserString`

La clase, `UserString` actúa como un envoltorio alrededor de los objetos de cadena. La necesidad de esta clase ha sido parcialmente suplantada por la capacidad de crear subclases directamente de `str`; sin embargo, es más fácil trabajar con esta clase porque se puede acceder a la cadena subyacente como atributo.

class collections.UserString(*seq*)

Clase que simula un objeto de cadena. El contenido de la instancia se mantiene en un objeto de cadena normal, al que se puede acceder a través del atributo `data` de las instancias `UserString`. El contenido de la instancia se establece inicialmente en una copia de *seq*. El argumento *seq* puede ser cualquier objeto que se pueda convertir en una cadena usando la función incorporada `str()`.

Además de admitir los métodos y operaciones de cadenas, las instancias `UserString` proporcionan el siguiente atributo:

data

Un objeto real `str` usado para almacenar el contenido de la clase `UserString`.

Distinto en la versión 3.5: Nuevos métodos `__getnewargs__`, `__rmod__`, `casefold`, `format_map`, `isprintable`, y `maketrans`.

[1] (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15) [2]

`collections.abc` — Abstract Base Classes for Containers

Added in version 3.3: Anteriormente, este módulo formaba parte del módulo **`collections`**.

Código fuente: Lib/_collections_abc.py [https://github.com/python/cpython/tree/3.14/Lib/_collections_abc.py]

This module provides abstract base classes that can be used to test whether a class provides a particular interface; for example, whether it is hashable or whether it is a mapping.

Una prueba **`issubclass()`** o **`isinstance()`** para una interfaz funciona de tres formas.

1. A newly written class can inherit directly from one of the abstract base classes. The class must supply the required abstract methods. The remaining mixin methods come from inheritance and can be overridden if desired. Other methods may be added as needed:

   ```
   class C(Sequence):                      # Direct in
       def __init__(self): ...             # Extra met
       def __getitem__(self, index):  ...  # Required
       def __len__(self):  ...             # Required
       def count(self, value): ...         # Optionall
   ```

   ```
   >>> issubclass(C, Sequence)
   True
   >>> isinstance(C(), Sequence)
   True
   ```

2. Existing classes and built-in classes can be registered as «virtual subclasses» of the ABCs. Those classes should define

the full API including all of the abstract methods and all of the mixin methods. This lets users rely on `issubclass()` or `isinstance()` tests to determine whether the full interface is supported. The exception to this rule is for methods that are automatically inferred from the rest of the API:

```
class D:                                           # No inher
    def __init__(self): ...                        # Extra me
    def __getitem__(self, index):  ...             # Abstract
    def __len__(self):  ...                        # Abstract
    def count(self, value): ...                    # Mixin me
    def index(self, value): ...                    # Mixin me

Sequence.register(D)                               # Register

>>> issubclass(D, Sequence)
True
>>> isinstance(D(), Sequence)
True
```

In this example, class **D** does not need to define `__contains__`, `__iter__`, and `__reversed__` because the in-operator, the iteration logic, and the `reversed()` function automatically fall back to using `__getitem__` and `__len__`.

3. Some simple interfaces are directly recognizable by the presence of the required methods (unless those methods have been set to `None`):

```
class E:
    def __iter__(self): ...
    def __next__(self): ...

>>> issubclass(E, Iterable)
True
>>> isinstance(E(), Iterable)
True
```

Las interfaces complejas no admiten esta última técnica

porque una interfaz es más que la simple presencia de nombres de métodos. Las interfaces especifican la semántica y las relaciones entre métodos que no se pueden inferir únicamente de la presencia de nombres de métodos específicos. Por ejemplo, saber que una clase proporciona `__getitem__`, `__len__` y `__iter__` no es suficiente para distinguir un **`Sequence`** de un **`Mapping`**.

Added in version 3.9: Estas clases abstractas ahora soportan `[]`. Vea Tipo Alias Genérico y **PEP 585** [https://peps.python.org/pep-0585/].

Colecciones clases base abstractas

El módulo de colecciones ofrece lo siguiente ABCs:

ABC	Hereda de	Métodos abstractos	Métodos mixin
`Container` [1]		`__contains__`	
`Hashable` [1]		`__hash__`	
`Iterable` [1] [2]		`__iter__`	
`Iterator` [1]	**`Iterable`**	`__next__`	`__iter__`
`Reversible` [1]	**`Iterable`**	`__reversed__`	
`Generator` [1]	**`Iterator`**	`send`, `throw`	`close`, `__iter__`, `__next__`
`Sized` [1]		`__len__`	
`Callable` [1]		`__call__`	
`Collection` [1]	**`Sized`**, **`Iterable`**, **`Container`**	`__contains__`, `__iter__`, `__len__`	
`Sequence`	**`Reversible`**, **`Collection`**	`__getitem__`, `__len__`	`__contains__`, `__iter__`, `__reversed__`, `index`, and `count`
`MutableSequence`	**`Sequence`**	`__getitem__`, `__setitem__`, `__delitem__`, `__len__`, `insert`	Inherited **`Sequence`** methods and `append`, `clear`, `reverse`, `extend`, `pop`, `remove`, and `__iadd__`
`ByteString`	**`Sequence`**	`__getitem__`, `__len__`	Métodos heredados **`Sequence`**
`Set`	**`Collection`**	`__contains__`, `__iter__`, `__len__`	`__le__`, `__lt__`, `__eq__`, `__ne__`, `__gt__`, `__ge__`, `__and__`, `__or__`, `__sub__`, `__rsub__`, `__xor__`, `__rxor__` and `isdisjoint`
`MutableSet`	**`Set`**	`__contains__`, `__iter__`, `__len__`, `add`, `discard`	Métodos heredados **`Set`** y `clear`, `pop`, `remove`, `__ior__`, `__iand__`, `__ixor__`, and `__isub__`
`Mapping`	**`Collection`**	`__getitem__`, `__iter__`, `__len__`	`__contains__`, `keys`, `items`, `values`, `get`, `__eq__`, and `__ne__`
`MutableMapping`	**`Mapping`**	`__getitem__`, `__setitem__`, `__delitem__`, `__iter__`, `__len__`	Métodos heredados **`Mapping`** y `pop`, `popitem`, `clear`, `update`, and `setdefault`
`MappingView`	**`Sized`**		`__init__`, `__len__` and `__repr__`

	MappingView, **Set**		`__contains__`, `__iter__`
	MappingView, **Set**		`__contains__`, `__iter__`
	MappingView, **Collection**		`__contains__`, `__iter__`
Awaitable [1]		`__await__`	
Coroutine [1]	**Awaitable**	`send`, `throw`	`close`
AsyncIterable [1]		`__aiter__`	
AsyncIterator [1]	**AsyncIterable**	`__anext__`	`__aiter__`
AsyncGenerator [1]	**AsyncIterator**	`asend`, `athrow`	`aclose`, `__aiter__`, `__anext__`
Buffer [1]		`__buffer__`	

Notas al pie

[1] These ABCs override **`__subclasshook__()`** to support testing an interface by verifying the required methods are present and have not been set to **`None`**. This only works for simple interfaces. More complex interfaces require registration or direct subclassing.

[2] Checking `isinstance(obj, Iterable)` detects classes that are registered as **`Iterable`** or that have an **`__iter__()`** method, but it does not detect classes that iterate with the **`__getitem__()`** method. The only reliable way to determine whether an object is iterable is to call `iter(obj)`.

Colecciones Clases base abstractas - Descripciones detalladas

class collections.abc.Container

ABC for classes that provide the **`__contains__()`** method.

class collections.abc.Hashable

ABC for classes that provide the **`__hash__()`** method.

class collections.abc.Sized

ABC for classes that provide the **`__len__()`** method.

class collections.abc.Callable

ABC for classes that provide the **`__call__()`** method.

See Anotaciones en objetos invocables for details on how to use `Callable` in type annotations.

class collections.abc.Iterable

ABC for classes that provide the `__iter__()` method.

Checking `isinstance(obj, Iterable)` detects classes that are registered as `Iterable` or that have an `__iter__()` method, but it does not detect classes that iterate with the `__getitem__()` method. The only reliable way to determine whether an object is iterable is to call `iter(obj)`.

class collections.abc.Collection

ABC para clases de contenedor iterables de tamaño.

Added in version 3.6.

class collections.abc.Iterator

ABC para clases que proporcionan el método `__iter__()` y `__next__()`. Ver también la definición de iterator.

class collections.abc.Reversible

ABC for iterable classes that also provide the `__reversed__()` method.

Added in version 3.6.

class collections.abc.Generator

ABC for generator classes that implement the protocol defined in **PEP 342** [https://peps.python.org/pep-0342/] that extends iterators with the `send()`, `throw()` and `close()` methods.

See Anotación de generadores y corrutinas for details on using `Generator` in type annotations.

Added in version 3.5.

class collections.abc.Sequence
class collections.abc.MutableSequence
class collections.abc.ByteString

ABC para solo lectura y mutable secuencias.

Implementation note: Some of the mixin methods, such as **`__iter__()`**, **`__reversed__()`**, and **`index()`** make repeated calls to the underlying **`__getitem__()`** method. Consequently, if **`__getitem__()`** is implemented with constant access speed, the mixin methods will have linear performance; however, if the underlying method is linear (as it would be with a linked list), the mixins will have quadratic performance and will likely need to be overridden.

index(*value*, *start=0*, *stop=None*)

Return first index of *value*.

Raises **`ValueError`** if the value is not present.

Supporting the *start* and *stop* arguments is optional, but recommended.

Distinto en la versión 3.5: The **`index()`** method gained support for the *stop* and *start* arguments.

Deprecated since version 3.12, will be removed in version 3.17: The **`ByteString`** ABC has been deprecated.

Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use **`Buffer`** or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).

`ByteString` was originally intended to be an abstract class that would serve as a supertype of both **`bytes`** and **`bytearray`**. However, since the ABC never had any methods, knowing that an object was an instance of **`ByteString`** never actually told you anything useful about the object. Other common buffer types such as **`memoryview`** were also

never understood as subtypes of **`ByteString`** (either at runtime or by static type checkers).

See **PEP 688** [https://peps.python.org/pep-0688/#current-options] for more details.

class collections.abc.Set
class collections.abc.MutableSet

ABCs for read-only and mutable sets.

class collections.abc.Mapping
class collections.abc.MutableMapping

ABC para solo lectura y mutable mapeos.

class collections.abc.MappingView
class collections.abc.ItemsView
class collections.abc.KeysView
class collections.abc.ValuesView

ABC para mapeo, elementos, claves y valores vistas.

class collections.abc.Awaitable

ABC for awaitable objects, which can be used in **`await`** expressions. Custom implementations must provide the **`__await__()`** method.

Coroutine objetos e instancias de la clase **`Coroutine`** ABC son todas las instancias de este ABC.

Nota

In CPython, generator-based coroutines (generators decorated with **`@types.coroutine`**) are *awaitables*, even though they do not have an **`__await__()`** method. Using `isinstance(gencoro, Awaitable)` for them will return `False`. Use **`inspect.isawaitable()`** to detect them.

Added in version 3.5.

class collections.abc.Coroutine

ABC for coroutine compatible classes. These implement the following methods, defined in Objetos de corrutina: **`send()`**, **`throw()`**, and **`close()`**. Custom implementations must also implement **`__await__()`**. All **`Coroutine`** instances are also instances of **`Awaitable`**.

Nota

In CPython, generator-based coroutines (generators decorated with **`@types.coroutine`**) are *awaitables*, even though they do not have an **`__await__()`** method. Using `isinstance(gencoro, Coroutine)` for them will return `False`. Use **`inspect.isawaitable()`** to detect them.

See Anotación de generadores y corrutinas for details on using **`Coroutine`** in type annotations. The variance and order of type parameters correspond to those of **`Generator`**.

Added in version 3.5.

class collections.abc.AsyncIterable

ABC for classes that provide an `__aiter__` method. See also the definition of asynchronous iterable.

Added in version 3.5.

class collections.abc.AsyncIterator

ABC para clases que proveen métodos `__aiter__` and `__anext__`. Ver también la definición de asynchronous iterator.

Added in version 3.5.

class collections.abc.AsyncGenerator

ABC for asynchronous generator classes that implement the protocol defined in **PEP 525** [https://peps.python.org/pep-0525/] and **PEP 492** [https://peps.python.org/pep-0492/].

See Anotación de generadores y corrutinas for details on using **`AsyncGenerator`** in type annotations.

Added in version 3.6.

class collections.abc.Buffer

ABC para clases que proveen el método **`__buffer__()`**, implementando el protocolo búfer. Ver **PEP 688** [https://peps.python.org/pep-0688/].

Added in version 3.12.

Ejemplos y Recetas

Los ABC nos permiten preguntar a las clases o instancias si brindan una funcionalidad particular, por ejemplo:

```
size = None
if isinstance(myvar, collections.abc.Sized):
    size = len(myvar)
```

Several of the ABCs are also useful as mixins that make it easier to develop classes supporting container APIs. For example, to write a class supporting the full **`Set`** API, it is only necessary to supply the three underlying abstract methods: **`__contains__()`**, **`__iter__()`**, and **`__len__()`**. The ABC supplies the remaining methods such as **`__and__()`** and **`isdisjoint()`**:

```
class ListBasedSet(collections.abc.Set):
    ''' Alternate set implementation favoring space over
        and not requiring the set elements to be hashabl
    def __init__(self, iterable):
        self.elements = lst = []
        for value in iterable:
            if value not in lst:
                lst.append(value)

    def __iter__(self):
        return iter(self.elements)
    def __contains__(self, value):
        return value in self.elements

    def __len__(self):
        return len(self.elements)

s1 = ListBasedSet('abcdef')
s2 = ListBasedSet('defghi')
overlap = s1 & s2            # The __and__() method is s
```

Notas sobre el uso de `Set` y `MutableSet` como un mixin:

1. Since some set operations create new sets, the default mixin methods need a way to create new instances from an iterable. The class constructor is assumed to have a signature in the form `ClassName(iterable)`. That assumption is factored-out to an internal `classmethod` called **`_from_iterable()`** which calls `cls(iterable)` to produce a new set. If the `Set` mixin is being used in a class with a different constructor signature, you will need to override **`_from_iterable()`** with a classmethod or regular method that can construct new instances from an iterable argument.
2. To override the comparisons (presumably for speed, as the semantics are fixed), redefine `__le__()` and `__ge__()`, then the other operations will automatically follow suit.
3. The `Set` mixin provides a **`_hash()`** method to compute a hash value for the set; however, `__hash__()` is not defined because not all sets are hashable or immutable. To add set hashability using mixins, inherit from both `Set()` and `Hashable()`, then define `__hash__ = Set._hash`.

Ver también

- OrderedSet receta [https://code.activestate.com/recipes/576694/] para un ejemplo basado en `MutableSet`.
- Para obtener más información sobre ABCs, ver el módulo `abc` y **PEP 3119** [https://peps.python.org/pep-3119/].

[1]

`heapq` — Heap queue algorithm

Código fuente: Lib/heapq.py [https://github.com/python/cpython/tree/3.14/Lib/heapq.py]

Este módulo proporciona una implementación del algoritmo de montículos, también conocido como algoritmo de cola con prioridad.

Min-heaps are binary trees for which every parent node has a value less than or equal to any of its children. We refer to this condition as the heap invariant.

For min-heaps, this implementation uses lists for which `heap[k] <= heap[2*k+1]` and `heap[k] <= heap[2*k+2]` for all *k* for which the compared elements exist. Elements are counted from zero. The interesting property of a min-heap is that its smallest element is always the root, `heap[0]`.

Max-heaps satisfy the reverse invariant: every parent node has a value *greater* than any of its children. These are implemented as lists for which `maxheap[2*k+1] <= maxheap[k]` and `maxheap[2*k+2] <= maxheap[k]` for all *k* for which the compared elements exist. The root, `maxheap[0]`, contains the *largest* element; `heap.sort(reverse=True)` maintains the max-heap invariant.

The **`heapq`** API differs from textbook heap algorithms in two aspects: (a) We use zero-based indexing. This makes the relationship between the index for a node and the indexes for its children slightly less obvious, but is more suitable since Python uses zero-based indexing. (b) Textbooks often focus on max-heaps, due to their suitability for in-place sorting. Our implementation favors min-heaps as they better correspond to Python **`lists`**.

These two aspects make it possible to view the heap as a regular Python list without surprises: `heap[0]` is the smallest item, and `heap.sort()` maintains the heap invariant!

Like **`list.sort()`**, this implementation uses only the `<` operator for comparisons, for both min-heaps and max-heaps.

In the API below, and in this documentation, the unqualified term *heap* generally refers to a min-heap. The API for max-heaps is named using a `_max` suffix.

To create a heap, use a list initialized as `[]`, or transform an existing list into a min-heap or max-heap using the **`heapify()`** or **`heapify_max()`** functions, respectively.

The following functions are provided for min-heaps:

heapq.heapify(*x*)

Transform list *x* into a min-heap, in-place, in linear time.

heapq.heappush(*heap*, *item*)

Push the value *item* onto the *heap*, maintaining the min-heap invariant.

heapq.heappop(*heap*)

Pop and return the smallest item from the *heap*, maintaining the min-heap invariant. If the heap is empty, **`IndexError`** is raised. To access the smallest item without popping it, use `heap[0]`.

heapq.heappushpop(*heap*, *item*)

Apila el elemento o *iem* en el montículo, y luego desapila y retorna el elemento más pequeño del montículo. La acción combinada se ejecuta más eficientemente que **`heappush()`** seguido de una llamada separada a **`heappop()`**.

heapq.heapreplace(*heap*, *item*)

Desapila y retorna el elemento más pequeño del *heap*, y

también apile el nuevo *item*. El tamaño del montículo no cambia. Si el montículo está vacío, **`IndexError`** se lanza.

Esta operación de un solo paso es más eficiente que un **`heappop()`** seguido por **`heappush()`** y puede ser más apropiada cuando se utiliza un montículo de tamaño fijo. La combinación pop/push siempre retorna un elemento del montículo y lo reemplaza con *item*.

El valor retornado puede ser mayor que el *item* añadido. Si no se desea eso, considere usar **`heappushpop()`** en su lugar. Su combinación push/pop retorna el menor de los dos valores, dejando el mayor valor en el montículo.

For max-heaps, the following functions are provided:

heapq.heapify_max(*x*)

Transform list *x* into a max-heap, in-place, in linear time.

Added in version 3.14.

heapq.heappush_max(*heap*, *item*)

Push the value *item* onto the max-heap *heap*, maintaining the max-heap invariant.

Added in version 3.14.

heapq.heappop_max(*heap*)

Pop and return the largest item from the max-heap *heap*, maintaining the max-heap invariant. If the max-heap is empty, **`IndexError`** is raised. To access the largest item without popping it, use `maxheap[0]`.

Added in version 3.14.

heapq.heappushpop_max(*heap*, *item*)

Push *item* on the max-heap *heap*, then pop and return the largest item from *heap*. The combined action runs more efficiently than **`heappush_max()`** followed by a separate call to **`heappop_max()`**.

Added in version 3.14.

heapq.heapreplace_max(*heap*, *item*)

Pop and return the largest item from the max-heap *heap* and also push the new *item*. The max-heap size doesn't change. If the max-heap is empty, **`IndexError`** is raised.

The value returned may be smaller than the *item* added. Refer to the analogous function **`heapreplace()`** for detailed usage notes.

Added in version 3.14.

El módulo también ofrece tres funciones de propósito general basadas en los montículos.

heapq.merge(**iterables*, *key=None*, *reverse=False*)

Fusionar varias entradas ordenadas en una sola salida ordenada (por ejemplo, fusionar entradas con marca de tiempo de varios archivos de registro). Retorna un iterator sobre los valores ordenados.

Similar a `sorted(itertools.chain(*iterables))` pero retorna un iterable, no hala los datos a la memoria de una sola vez, y asume que cada uno de los flujos de entrada ya están ordenado (de menor a mayor).

Tiene dos argumentos opcionales que deben ser especificados como argumentos de palabras clave.

key especifica una key function de un argumento que se utiliza para extraer una clave de comparación de cada elemento de entrada. El valor por defecto es `None` (compara los elementos directamente).

reverse es un valor booleano. Si se establece en `True`, entonces los elementos de entrada se fusionan como si cada comparación se invirtiera. Para lograr un comportamiento similar a `sorted(itertools.chain(*iterables), reverse=True)`, todos los iterables deben ser ordenados de mayor a menor.

Distinto en la versión 3.5: Añadió los parámetros opcionales de *key* y *reverse*.

heapq.nlargest(*n*, *iterable*, *key=None*)

Retorna una lista con los *n* elementos más grandes del conjunto de datos definidos por *iterable*. *key*, si se proporciona, especifica una función de un argumento que se utiliza para extraer una clave de comparación de cada elemento en *iterable* (por ejemplo, `key=str.lower`). Equivalente a: `sorted(iterable, key=clave, reverse=True)[:n]`.

heapq.nsmallest(*n*, *iterable*, *key=None*)

Retorna una lista con los *n* elementos más pequeños del conjunto de datos definidos por *iterable*. *key*, si se proporciona, especifica una función de un argumento que se utiliza para extraer una clave de comparación de cada elemento en *iterable* (por ejemplo, `key=str.lower`). Equivalente a: `sorted(iterable, key=clave)[:n]`.

Las dos últimas funciones funcionan mejor para valores más pequeños de *n*. Para valores más grandes, es más eficiente usar la función **`sorted()`**. Además, cuando `n==1`, es más eficiente usar las funciones incorporadas **`min()`** y **`` max`() ``**. Si se requiere el uso repetido de estas funciones, considere convertir lo iterable en un verdadero montículo.

Ejemplos Básicos

Un heapsort» [https://en.wikipedia.org/wiki/Heapsort] puede ser implementado empujando todos los valores en un montículo y luego desapilando los valores más pequeños uno a la vez:

```
>>> def heapsort(iterable):
```

```
...     h = []
...     for value in iterable:
...         heappush(h, value)
...     return [heappop(h) for i in range(len(h))]
...
>>> heapsort([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Esto es similar a `sorted(iterable)`, pero a diferencia de **`sorted()`**, esta implementación no es estable.

Los elementos del montículo pueden ser tuplas. Esto es útil para asignar valores de comparación (como las prioridades de las tareas) junto con el registro principal que se está rastreando:

```
>>> h = []
>>> heappush(h, (5, 'write code'))
>>> heappush(h, (7, 'release product'))
>>> heappush(h, (1, 'write spec'))
>>> heappush(h, (3, 'create tests'))
>>> heappop(h)
(1, 'write spec')
```

Other Applications

Medians [https://en.wikipedia.org/wiki/Median] are a measure of central tendency for a set of numbers. In distributions skewed by outliers, the median provides a more stable estimate than an average (arithmetic mean). A running median is an online algorithm [https://en.wikipedia.org/wiki/Online_algorithm] that updates continuously as new data arrives.

A running median can be efficiently implemented by balancing two heaps, a max-heap for values at or below the midpoint and a min-heap for values above the midpoint. When the two heaps have the same size, the new median is the average of the tops of the two heaps; otherwise, the median is at the top of the larger heap:

```
def running_median(iterable):
    "Yields the cumulative median of values seen so far.
    lo = []  # max-heap
    hi = []  # min-heap (same size as or one smaller tha

    for x in iterable:
        if len(lo) == len(hi):
            heappush_max(lo, heappushpop(hi, x))
            yield lo[0]
        else:
            heappush(hi, heappushpop_max(lo, x))
            yield (lo[0] + hi[0]) / 2
```

For example:

```
>>> list(running_median([5.0, 9.0, 4.0, 12.0, 8.0, 9.0])
[5.0, 7.0, 5.0, 7.0, 8.0, 8.5]
```

Notas de Aplicación de la Cola de Prioridades

Una cola de prioridad [https://en.wikipedia.org/wiki/Priority_queue] es de uso común para un montículo, y presenta varios desafíos de implementación:

- Estabilidad de la clasificación: ¿cómo se consigue que dos tareas con iguales prioridades sean retornadas en el orden en que fueron añadidas originalmente?
- Interrupciones de comparación en tupla para pares (prioridad, tarea) si las prioridades son iguales y las tareas no tienen un orden de comparación por defecto.
- ¿Si la prioridad de una tarea cambia, cómo la mueves a una nueva posición en el montículo?
- ¿O si una tarea pendiente necesita ser borrada, cómo la encuentras y la eliminas de la cola?

Una solución a los dos primeros desafíos es almacenar las entradas como una lista de 3 elementos que incluya la prioridad, un recuento de entradas y la tarea. El recuento de entradas sirve como un desempate para que dos tareas con la misma prioridad sean retornadas en el orden en que fueron añadidas. Y como no hay dos recuentos de entradas iguales, la comparación tupla nunca intentará comparar directamente dos tareas.

Otra solución al problema de las tareas no comparables es crear una clase envolvente que ignore el elemento de la tarea y sólo compare el campo de prioridad:

```
from dataclasses import dataclass, field
from typing import Any

@dataclass(order=True)
class PrioritizedItem:
    priority: int
    item: Any=field(compare=False)
```

Los desafíos restantes giran en torno a encontrar una tarea pendiente y hacer cambios en su prioridad o eliminarla por completo. Encontrar una tarea se puede hacer con un diccionario que apunta a una entrada en la cola.

Eliminar la entrada o cambiar su prioridad es más difícil porque rompería las invariantes de la estructura del montículo. Por lo tanto, una posible solución es marcar la entrada como eliminada y añadir una nueva entrada con la prioridad revisada:

```
pq = []                         # list of entries arrang
entry_finder = {}               # mapping of tasks to e
REMOVED = '<removed-task>'      # placeholder for a remo
counter = itertools.count()     # unique sequence count

def add_task(task, priority=0):
    'Add a new task or update the priority of an existi
    if task in entry_finder:
        remove_task(task)
    count = next(counter)
    entry = [priority, count, task]
    entry_finder[task] = entry
    heappush(pq, entry)
```

```
def remove_task(task):
    'Mark an existing task as REMOVED.  Raise KeyError i
    entry = entry_finder.pop(task)
    entry[-1] = REMOVED

def pop_task():
    'Remove and return the lowest priority task. Raise K
    while pq:
        priority, count, task = heappop(pq)
        if task is not REMOVED:
            del entry_finder[task]
            return task
    raise KeyError('pop from an empty priority queue')
```

Teoría

Los montículos son conjuntos para los cuales `a[k] <= a[2*k+1]` y `a[k] <= a[2*k+2]` para todos los *k*, contando los elementos desde 0. Para comparar, los elementos no existentes se consideran infinitos. La interesante propiedad de un montículo es que `a[0]` es siempre su elemento más pequeño.

La extraña invariante de arriba intenta ser una representación eficiente de la memoria para un torneo. Los números de abajo son *k*, no `a[k]`:

```
                               0

              1                                 2

      3               4                5               6

  7       8       9       10      11      12      13

15 16   17 18   19 20   21 22   23 24   25 26   27 28
```

En el árbol de arriba, cada celda *k* está coronada por `2*k+1` y `2*k+2`. En un torneo binario habitual que vemos en los deportes, cada celda es el ganador sobre las dos celdas que supera, y podemos rastrear al ganador hasta el árbol para ver todos los oponentes que tuvo. Sin embargo, en muchas aplicaciones informáticas de tales torneos, no necesitamos rastrear la historia de un ganador. Para ser más eficientes en la memoria, cuando un ganador es ascendido, tratamos de reemplazarlo por algo más en un nivel inferior, y la regla se convierte en que una celda y las dos celdas que supera contienen tres elementos diferentes, pero la celda superior «gana» sobre las dos celdas superiores.

If this heap invariant is protected at all time, index 0 is clearly the overall winner. The simplest algorithmic way to remove it and find the «next» winner is to move some loser (let's say cell 30 in the diagram above) into the 0 position, and then percolate this new 0 down the tree, exchanging values, until the invariant is re-established. This is clearly logarithmic on the total number of items in the tree. By iterating over all items, you get an $O(n \log n)$ sort.

Una buena característica de este tipo es que puedes insertar nuevos elementos de manera eficiente mientras se realiza la clasificación, siempre y cuando los elementos insertados no sean «mejores» que el último 0'th elemento que has extraído. Esto es especialmente útil en contextos de simulación, donde el árbol contiene todos los eventos entrantes, y la condición de «ganar» significa el menor tiempo programado. Cuando un evento programa otros eventos para su ejecución, se programan en el futuro, para que puedan ir fácilmente al montículo. Por lo tanto, un montículo es una buena estructura para implementar planificadores o *schedulers* (esto es lo que usé para mi secuenciador MIDI :-).

Se han estudiado extensamente varias estructuras para implementar los planificadores, y los montículos son buenos para ello, ya que son razonablemente rápidos, la velocidad es casi constante, y el peor de los casos no es muy diferente del caso promedio. Sin embargo, hay otras representaciones que son más eficientes en general, aunque los peores casos podrían ser terribles.

Los montículos también son muy útiles en las grandes ordenaciones de elementos en discos de memoria. Lo más probable es que todos sepan que un tipo grande implica la producción de «ejecuciones» (que son secuencias preclasificadas, cuyo tamaño suele estar relacionado con la cantidad de memoria de la CPU), seguidas de una fusión de pases para estas ejecuciones, cuya fusión suele estar muy inteligentemente organizada [1]. Es muy importante que la clasificación inicial produzca las ejecuciones posibles más largas. Los torneos son una buena manera de lograrlo. Si, utilizando toda la memoria disponible para celebrar un torneo, sustituyes y filtras los elementos que encajan en la carrera actual, producirás carreras que tienen el doble del tamaño de la memoria para la entrada aleatoria, y mucho mejor para la entrada ordenada de forma difusa.

Además, si se da salida al *0’th item* en el disco y se obtiene una entrada que no puede caber en el torneo actual (porque el valor «gana» sobre el último valor de salida), no puede caber en el montículo, por lo que el tamaño del montículo disminuye. La memoria liberada podría ser ingeniosamente reutilizada inmediatamente para construir progresivamente un segundo montículo, que crece exactamente al mismo ritmo que el primer montículo se está fundiendo. Cuando el primer montículo se desvanece completamente, se cambia de montículo y se inicia una nueva carrera. ¡Ingenioso y muy efectivo!

En una palabra, los montículos son estructuras de memoria útiles a conocer. Las uso en algunas aplicaciones, y creo que es bueno tener un módulo “heap” alrededor. :-)

Notas al pie de página

Los algoritmos de balanceo de discos que están vigentes hoy en día, son más molestos que inteligentes, y esto es una consecuencia de las capacidades de búsqueda de los discos. En los dispositivos que no pueden buscar, como las grandes unidades de cinta, la historia era muy diferente, y había que ser muy inteligente para asegurarse (con mucha antelación) de que cada movimiento de la cinta fuera el más efectivo (es decir, que participara mejor en el «progreso» de la fusión). Algunas cintas eran incluso capaces de leer al revés, y esto también se utilizó para evitar el tiempo rebobinado. Créanme, ¡la ordenación de elementos en cinta realmente buenos fueron espectaculares de ver! ¡Desde todos los tiempos, la ordenación de elementos siempre ha sido un Gran Arte! :-)

`bisect` — Array bisection algorithm

Código fuente: Lib/bisect.py [https://github.com/python/cpython/tree/3.14/Lib/bisect.py]

Este módulo brinda soporte para mantener una lista ordenada sin tener que reordenar la lista tras cada nueva inserción. Para listas largas de elementos que tienen operaciones de comparación costosas, esto puede suponer una mejora con respecto a las búsquedas lineales o el reordenado frecuente.

The module is called `bisect` because it uses a basic bisection algorithm to do its work. Unlike other bisection tools that search for a specific value, the functions in this module are designed to locate an insertion point. Accordingly, the functions never call an `__eq__()` method to determine whether a value has been found. Instead, the functions only call the `__lt__()` method and will return an insertion point between values in an array.

Nota

The functions in this module are not thread-safe. If multiple threads concurrently use `bisect` functions on the same sequence, this may result in undefined behaviour. Likewise, if the provided sequence is mutated by a different thread while a `bisect` function is operating on it, the result is undefined. For example, using `insort_left()` on the same list from multiple threads may result in the list becoming unsorted.

Las siguientes funciones están disponibles:

bisect.bisect_left(*a*, *x*, *lo=0*, *hi=len(a)*, ***, *key=None*)

Ubicar el punto de inserción para *x* en *a* para mantener el ordenamiento. Los parámetros *lo* (inferior) y *hi* (superior) pueden utilizarse para especificar un subconjunto de la lista que debería considerarse; por defecto se utiliza la lista completa. Si *x* ya está presente en *a*, el punto de inserción será antes (a la izquierda) de cualquier elemento existente. El valor de retorno es adecuado para que se utilice como primer parámetro para `list.insert()`, suponiendo que *a* ya está ordenada.

El punto de inserción retornado *ip* divide el arreglo *a* en dos partes, de modo que `all(elem < x for elem in a[lo : ip])` es verdadero para el lado izquierdo y `all(elem >= x for elem in a[ip : hi])` es verdadero para el lado derecho.

key especifica una función clave de un argumento que es usada para extraer una clave de comparación de cada elemento en el arreglo. La función clave no se aplica a *x* para facilitar la búsqueda de registros complejos.

Si *key* es `None`, los elementos se comparan directamente y ninguna función clave es llamada.

Distinto en la versión 3.10: Se agregó el parámetro *key*.

bisect.bisect_right(*a*, *x*, *lo=0*, *hi=len(a)*, ***, *key=None*)
bisect.bisect(*a*, *x*, *lo=0*, *hi=len(a)*, ***, *key=None*)

Similar a **`bisect_left()`**, pero retorna un punto de inserción que viene después (a la derecha) de cualquier entrada existente de *x* en *a*.

El punto de inserción retornado *ip* divide el arreglo *a* en dos partes, de modo que `all(elem <= x for elem in a[lo : ip])` es verdadero para el lado izquierdo y `all(elem > x for elem in a[ip : hi])` es verdadero para el lado derecho.

Distinto en la versión 3.10: Se agregó el parámetro *key*.

bisect.insort_left(*a*, *x*, *lo=0*, *hi=len(a)*, ***, *key=None*)

Inserte *x* en *a* en orden ordenado.

This function first runs `bisect_left()` to locate an insertion point. Next, it runs the `insert()` method on *a* to insert *x* at the appropriate position to maintain sort order.

Para mantener la inserción de registros en una tabla, la función *key* (en caso de existir) se aplica a *x* en el paso de búsqueda pero no en el paso de inserción.

Keep in mind that the $O(\log n)$ search is dominated by the slow $O(n)$ insertion step.

Distinto en la versión 3.10: Se agregó el parámetro *key*.

bisect.insort_right(*a, x, lo=0, hi=len(a), *, key=None*)
bisect.insort(*a, x, lo=0, hi=len(a), *, key=None*)

Similar a `insort_left()`, pero inserta *x* en *a* después de cualquier entrada de *x* existente.

This function first runs `bisect_right()` to locate an insertion point. Next, it runs the `insert()` method on *a* to insert *x* at the appropriate position to maintain sort order.

Para mantener la inserción de registros en una tabla, la función *key* (en caso de existir) se aplica a *x* en el paso de búsqueda pero no en el paso de inserción.

Keep in mind that the $O(\log n)$ search is dominated by the slow $O(n)$ insertion step.

Distinto en la versión 3.10: Se agregó el parámetro *key*.

Notas de rendimiento

Al escribir código sensible al tiempo usando *bisect()* y *insort()*, tenga en cuenta estos pensamientos:

- La bisección es eficaz para buscar rangos de valores. Para localizar valores específicos, los diccionarios son más eficaces.

- The *insort()* functions are *O*(*n*) because the logarithmic search step is dominated by the linear time insertion step.
- Las funciones de búsqueda no tienen estado y descartan los resultados de las funciones clave después de ser usadas. En consecuencia, si las funciones de búsqueda se utilizan en un bucle, la función clave se puede llamar una y otra vez en los mismos elementos del arreglo. Si la función clave no es rápida, considere envolverla con `functools.cache()` para evitar cálculos duplicados. Alternativamente, considere buscar un arreglo de claves precalculadas para ubicar el punto de inserción (como se muestra en la sección de ejemplos a continuación).

Ver también

- Sorted Collections [https://grantjenks.com/docs/sortedcollections/] es un módulo de alto rendimiento que utiliza *bisect* para gestionar colecciones de datos ordenadas.
- El SortedCollection recipe [https://code.activestate.com/recipes/577197-sortedcollection/] usa bisect para construir una clase de colección con todas las funciones con métodos de búsqueda sencillos y soporte para una función clave. Las claves se calculan previamente para ahorrar llamadas innecesarias a la función clave durante las búsquedas.

Búsqueda en listas ordenadas

Las bisect functions anteriores son útiles para encontrar puntos de inserción, pero pueden resultar difíciles o engorrosas para tareas de búsqueda habituales. Las siguientes cinco funciones muestran cómo convertirlas en búsquedas estándar para listas ordenadas:

```
def index(a, x):
    'Locate the leftmost value exactly equal to x'
    i = bisect_left(a, x)
    if i != len(a) and a[i] == x:
        return i
    raise ValueError
```

```
def find_lt(a, x):
    'Find rightmost value less than x'
    i = bisect_left(a, x)
    if i:
        return a[i-1]
    raise ValueError

def find_le(a, x):
    'Find rightmost value less than or equal to x'
    i = bisect_right(a, x)
    if i:
        return a[i-1]
    raise ValueError

def find_gt(a, x):
    'Find leftmost value greater than x'
    i = bisect_right(a, x)
    if i != len(a):
        return a[i]
    raise ValueError

def find_ge(a, x):
    'Find leftmost item greater than or equal to x'
    i = bisect_left(a, x)
    if i != len(a):
        return a[i]
    raise ValueError
```

Ejemplos

La función **`bisect()`** puede ser útil para búsquedas en tablas numéricas. Este ejemplo utiliza **`bisect()`** para buscar una calificación de un examen (digamos) dada por una letra, basándose en un conjunto de punto de corte numéricos ordenados: 90 o más es una “A”, de 80 a 89 es una “B”, y así sucesivamente:

```
>>> def grade(score)
...     i = bisect([60, 70, 80, 90], score)
...     return "FDCBA"[i]
...
>>> [grade(score) for score in [33, 99, 77, 70, 89, 90,
['F', 'A', 'C', 'C', 'B', 'A', 'A']
```

Las funciones **`bisect()`** y **`insort()`** también funcionan con listas de tuplas. El argumento *key* puede usarse para extraer el campo usado para ordenar registros en una tabla:

```
>>> from collections import namedtuple
>>> from operator import attrgetter
>>> from bisect import bisect, insort
>>> from pprint import pprint

>>> Movie = namedtuple('Movie', ('name', 'released', 'di

>>> movies = [
...     Movie('Jaws', 1975, 'Spielberg'),
...     Movie('Titanic', 1997, 'Cameron'),
...     Movie('The Birds', 1963, 'Hitchcock'),
...     Movie('Aliens', 1986, 'Cameron')
... ]

>>> # Find the first movie released after 1960
>>> by_year = attrgetter('released')
>>> movies.sort(key=by_year)
>>> movies[bisect(movies, 1960, key=by_year)]
Movie(name='The Birds', released=1963, director='Hitchco

>>> # Insert a movie while maintaining sort order
>>> romance = Movie('Love Story', 1970, 'Hiller')
>>> insort(movies, romance, key=by_year)
>>> pprint(movies)
[Movie(name='The Birds', released=1963, director='Hitchc
 Movie(name='Love Story', released=1970, director='Hille
 Movie(name='Jaws', released=1975, director='Spielberg')
 Movie(name='Aliens', released=1986, director='Cameron')
 Movie(name='Titanic', released=1997, director='Cameron'
```

Para evitar llamadas repetidas a la función clave, cuando ésta usa muchos recursos, se puede buscar en una lista de claves previamente calculadas para encontrar el índice de un registro:

```
>>> data = [('red', 5), ('blue', 1), ('yellow', 8), ('bl
>>> data.sort(key=lambda r: r[1])       # Or use operato
>>> keys = [r[1] for r in data]         # Precompute a l
>>> data[bisect_left(keys, 0)]
('black', 0)
>>> data[bisect_left(keys, 1)]
('blue', 1)
>>> data[bisect_left(keys, 5)]
('red', 5)
>>> data[bisect_left(keys, 8)]
('yellow', 8)
```

`array` — Efficient arrays of numeric values

This module defines an object type which can compactly represent an array of basic values: characters, integers, floating-point numbers. Arrays are mutable sequence types and behave very much like lists, except that the type of objects stored in them is constrained. The type is specified at object creation time by using a *type code*, which is a single character. The following type codes are defined:

Código de tipo	Tipo C	Tipo Python	Tamaño mínimo en bytes	Notas
`'b'`	signed char	int	1	
`'B'`	unsigned char	int	1	
`'u'`	wchar_t	Carácter unicode	2	(1)
`'w'`	Py_UCS4	Carácter unicode	4	(2)
`'h'`	signed short	int	2	
`'H'`	unsigned short	int	2	
`'i'`	signed int	int	2	
`'I'`	unsigned int	int	2	
`'l'`	signed long	int	4	
`'L'`	unsigned long	int	4	
`'q'`	signed long long	int	8	
`'Q'`	unsigned long long	int	8	
`'f'`	float	float	4	
`'d'`	double	float	8	

Notas:

1. Puede ser de 16 bits o 32 bits según la plataforma.

   Distinto en la versión 3.9: `array('u')` now uses **`wchar_t`** as C type instead of deprecated `Py_UNICODE`. This change doesn't affect its behavior because `Py_UNICODE` is alias of **`wchar_t`** since Python 3.3.

Deprecated since version 3.3, will be removed in version 3.16: Please migrate to `'w'` typecode.

2. *Added in version 3.13.*

Ver también

The ctypes and struct modules, as well as third-party modules like numpy [https://numpy.org/doc/stable/reference/arrays.interface.html#object.__array_interface__], use similar – but slightly different – type codes.

The actual representation of values is determined by the machine architecture (strictly speaking, by the C implementation). The actual size can be accessed through the `array.itemsize` attribute.

The module defines the following item:

array.typecodes

Una cadena de caracteres con todos los códigos de tipos disponible.

El módulo define los siguientes tipos:

class array.array(*typecode*[, *initializer*])

A new array whose items are restricted by *typecode*, and initialized from the optional *initializer* value, which must be a `bytes` or `bytearray` object, a Unicode string, or iterable over elements of the appropriate type.

If given a `bytes` or `bytearray` object, the initializer is passed to the new array's `frombytes()` method; if given a Unicode string, the initializer is passed to the `fromunicode()` method; otherwise, the initializer's iterator is passed to the `extend()` method to add initial items to the array.

Array objects support the ordinary mutable sequence operations of indexing, slicing, concatenation, and multiplication. When using slice assignment, the assigned value must be an array object with the same type code; in all other cases, **`TypeError`** is raised. Array objects also implement the buffer interface, and may be used wherever bytes-like objects are supported.

Lanza un evento de auditoría `array.__new__` con argumentos `typecode`, `initializer`.

typecode

El carácter typecode utilizado para crear el arreglo.

itemsize

La longitud en bytes de un elemento del arreglo en su representación interna.

append(*value*, */*)

Append a new item with the specified value to the end of the array.

buffer_info()

Return a tuple `(address, length)` giving the current memory address and the length in elements of the buffer used to hold array's contents. The size of the memory buffer in bytes can be computed as `array.buffer_info()[1] * array.itemsize`. This is occasionally useful when working with low-level (and inherently unsafe) I/O interfaces that require memory addresses, such as certain **`ioctl()`** operations. The returned numbers are valid as long as the array exists and no length-changing operations are applied to it.

Nota

Cuando utilizamos objetos tipo arreglo escritos en C o C++ (la única manera de utilizar esta información de forma más efectiva), tiene más sentido utilizar interfaces buffer que soporten objetos del tipo arreglo. Este método es mantenido con retro compatibilidad y tiene que ser evitado en el nuevo código. Las interfaces de buffer son documentadas en Protocolo búfer.

byteswap()

«Byteswap» todos los elementos del arreglo. Solo es soportado para valores de tamaño 1,2,3,4 o 8 bytes; para otros valores se lanza `RuntimeError`. Es útil cuando leemos información de un fichero en una máquina con diferente orden de bytes.

count(*value*, /)

Return the number of occurrences of *value* in the array.

extend(*iterable*, /)

Añade los elementos del *iterable* al final del arreglo. Si el *iterable* es de otro arreglo, este debe ser *exactamente* del mismo tipo; si no, se lanza `TypeError`. Si el *iterable* no es un arreglo, este debe de ser un iterable y sus elementos deben ser del tipo correcto para ser añadidos al arreglo.

frombytes(*buffer*, /)

Appends items from the bytes-like object, interpreting its content as an array of machine values (as if it had been read from a file using the `fromfile()` method).

Added in version 3.2: `fromstring()` is renamed to `frombytes()` for clarity.

fromfile(*f*, *n*, /)

Lee *n* elementos (como valores de máquina) del file object *f* y los añade al final del arreglo. Si hay menos de *n* elementos disponibles, se lanza `EOFError`, pero los elementos que estaban disponibles todavía se insertan en el arreglo.

fromlist(*list*, */*)

Añade los elementos de la lista. Es equivalente a `for x in list: a.append(x)` excepto que si hay un error de tipo, el arreglo no se modifica.

fromunicode(*ustr*, */*)

Extends this array with data from the given Unicode string. The array must have type code `'u'` or `'w'`; otherwise a **`ValueError`** is raised. Use `array.frombytes(unicodestring.encode(enc))` to append Unicode data to an array of some other type.

index(*value*[, *start*[, *stop*]])

Return the smallest *i* such that *i* is the index of the first occurrence of *value* in the array. The optional arguments *start* and *stop* can be specified to search for *value* within a subsection of the array. Raise **`ValueError`** if *value* is not found.

Distinto en la versión 3.10: Se agregaron parámetros opcionales *start* y *stop*.

insert(*index*, *value*, */*)

Insert a new item *value* in the array before position *index*. Negative values are treated as being relative to the end of the array.

pop(*index=-1*, */*)

Elimina el elemento con índice *i* del arreglo y lo retorna. El argumento opcional por defecto es `-1`, en caso de utilizar el argumento por defecto el ultimo elemento es eliminado y retornado.

remove(*value*, */*)

Remove the first occurrence of *value* from the array.

clear()

Remove all elements from the array.

Added in version 3.13.

reverse()

Invierte el orden de los elementos en el arreglo.

tobytes()

Convierte el arreglo en un arreglo de valores máquina y retorna una representación en formato de bytes (la misma secuencia de bytes que se deben escribir en un fichero por el método **`tofile()`**.)

Added in version 3.2: **`tostring()`** is renamed to **`tobytes()`** for clarity.

tofile(*f*, */*)

Escribe todos los elementos (incluido elementos máquina) a el file object *f*.

tolist()

Convierte el arreglo a una lista ordinaria con los mismos elementos.

tounicode()

Convert the array to a Unicode string. The array must have a type `'u'` or `'w'`; otherwise a **`ValueError`** is raised. Use `array.tobytes().decode(enc)` to obtain a Unicode string from an array of some other type.

The string representation of array objects has the form `array(typecode, initializer)`. The *initializer* is omitted if the array is empty, otherwise it is a Unicode string if the *typecode* is `'u'` or `'w'`, otherwise it is a list of numbers. The string representation is guaranteed to be able to be converted back to an

array with the same type and value using **`eval()`**, so long as the **`array`** class has been imported using `from array import array`. Variables `inf` and `nan` must also be defined if it contains corresponding floating-point values. Examples:

```
array('l')
array('w', 'hello \u2641')
array('l', [1, 2, 3, 4, 5])
array('d', [1.0, 2.0, 3.14, -inf, nan])
```

Ver también

Módulo `struct`
Empaquetado y desempaquetado de datos binarios heterogéneos.

NumPy [https://numpy.org/]
El paquete NumPy define otro tipo de arreglo.

`weakref` — Weak references

Código Fuente: Lib/weakref.py [https://github.com/python/cpython/tree/3.14/Lib/weakref.py]

The **`weakref`** module allows the Python programmer to create *weak references* to objects.

En lo consecutivo, el término *referente* aludirá al objeto que es referenciado por una referencia débil.

Una referencia débil a un objeto no es suficiente para mantener al objeto con vida: cuando las únicas referencias que le queden a un referente son referencias débiles, la (recolección de basura) es libre de destruir al referente y reusar su memoria para algo más. Sin embargo, hasta que el objeto no sea realmente destruido, la referencia débil puede retornar el objeto incluso si no tiene referencias fuertes.

Un uso principal para las referencias débiles es para implementar caches o mapeados que mantienen objetos grandes, cuando no se desea que un objeto grande no sea mantenido con vida sólo porque aparece en un cache o mapeado.

For example, if you have a number of large binary image objects, you may wish to associate a name with each. If you used a Python dictionary to map names to images, or images to names, the image objects would remain alive just because they appeared as values or keys in the dictionaries. The **`WeakKeyDictionary`** and **`WeakValueDictionary`** classes supplied by the **`weakref`** module are an alternative, using weak references to construct mappings that don't keep objects alive solely because they appear in the mapping objects. If, for example, an image object is a value in a **`WeakValueDictionary`**, then when the last remaining references to that image object are the weak references held by weak mappings, garbage collection can reclaim the object, and its corresponding entries in weak mappings are simply deleted.

`WeakKeyDictionary` y `WeakValueDictionary` usan referencias débiles en sus implementaciones, estableciendo retrollamadas (*callback*) en las referencias débiles que notifiquen a los diccionarios débiles cuando una llave o valor ha sido reclamado por la recolección de basura. `WeakSet` implementa la interfaz `set`, pero mantiene referencias débiles de sus elementos, justo como lo hace `WeakKeyDictionary`.

`finalize` provee una forma directa de registrar una función de limpieza que se llame cuando un objeto es recogido por la recolección de basura. Esto es más simple que configurar una retrollamada en una referencia débil pura, ya que el módulo automáticamente se asegura que el finalizador se mantenga con vida hasta que el objeto sea recolectado.

Most programs should find that using one of these weak container types or `finalize` is all they need – it's not usually necessary to create your own weak references directly. The low-level machinery is exposed by the `weakref` module for the benefit of advanced uses.

No todos los objetos pueden ser débilmente referenciados. Objetos que soportan referencias débiles pueden incluir instancias de clases, funciones escritas en Python (pero no en C), métodos de instancia, conjuntos, frozensets, algunos objetos archivo, generadores, objetos de tipo, sockets, arreglos, deques, objetos de patrones de expresiones regulares, y objetos código.

Distinto en la versión 3.2: Se añadió el soporte para thread.lock, threading.Lock, y objetos código.

Varios tipos incorporados como `list` y `dict` no soportan directamente referencias débiles pero pueden añadir soporte al crear una subclase:

```
class Dict(dict):
    pass

obj = Dict(red=1, green=2, blue=3)   # this object is we
```

Otros tipos incorporados como `tuple` y `int` no soportan referencias débiles incluso cuando son usadas como clase base.

Los tipos de extensiones se pueden hacer para soportar referencias débiles; véase Soporte de referencia débil.

Cuando `__slots__` es definido para un tipo específico, el soporte para referencia débil es deshabilitado a menos que una cadena `'__weakref__'` también esté presente en la secuencia de cadenas en la declaración `__slots__`. Véase _slots_ documentation para más detalles.

class weakref.ref(*object*[, *callback*])

Retorna una referencia débil de *object*. El objeto original puede ser recuperado al llamar la referencia del objeto si el referente sigue con vida; si el referente ya no está con vida, llamar a la referencia del objeto causará que se retorne un `None`. Si se proporciona *callback* y no `None`, y el objeto weakref retornado aún sigue con vida, la retrollamada (*callback*) será llamado cuando el objeto esté a punto de ser finalizado; el objeto de la referencia débil será pasado como el único parámetro a la retrollamada, el referente ya no estará disponible.

Se permite que muchas referencias débiles sean construidas por el mismo objeto. Las retrollamadas registradas por cada referencia débil serán llamados desde la retrollamada registrada más recientemente hasta la retrollamada registrada más antigua.

Las excepciones lanzadas por la retrollamada serán visibles en la salida de error estándar pero no pueden ser propagadas; son manejadas de la misma forma que las excepciones lanzadas por el método `__del__()` de un objeto.

Las referencias débiles son hashable si el *object* es mapeable. Ellos mantendrán su valor del hash incluso cuando el *objet* haya sido eliminado. Si `hash()` es llamado por primera vez sólo después de que *object* sea eliminado, la llamada lanzará un `TypeError`.

Las referencias débiles soportan pruebas para igualdad, pero no para ordenación. Si los referentes están todavía con vida, dos referencias tiene la misma relación de igualdad como sus referentes (sin importar el *callback*). Si un referente ha sido eliminado, las referencias son iguales sólo si el objetos de referencia son el mismo objeto.

Es un tipo del que se puede crear una subclase en vez de una función de fábrica.

__callback__

Este atributo de sólo lectura retorna la llamada que está asociada actualmente con el weakref. Si no hay retrollamadas o si el referente del weakref no está con vida entonces este atributo tendrá de valor `None`.

Distinto en la versión 3.4: Se añadió el atributo **`__callback__`**.

weakref.proxy(*object*[, *callback*])

Retorna un proxy a *object* que usa una referencia débil. Esto soporta el uso del proxy en la mayoría de los contextos en vez de requerir la dereferencia explícita usada con los objetos de referencia débil. El objeto retornado tendrá un tipo `ProxyType` o `CallableProxyType`, dependiendo si *object* es invocable. Objetos Proxy no son hashable independiente de la referencia; esto evita un número de problemas relacionados a su naturaleza mutable fundamental, y evita su uso como clave de diccionario. *callback* corresponde al parámetro del mismo nombre de la función **`ref()`**.

Acceder al atributo de un objeto proxy después de que el objeto referenciado haya sido recolectado por el recolector de basura lanza **`ReferenceError`**.

Distinto en la versión 3.8: Se extendió el soporto de operadores en objetos proxy para incluir los operadores de multiplicación de matrices `@` and `@=`.

weakref.getweakrefcount(*object*)

Retorna el número de referencias débiles y proxies que refieren a *object*.

weakref.getweakrefs(*object*)

Retorna una lista de todas las referencias débiles y objetos proxy que refieren a *object*.

class weakref.WeakKeyDictionary([*dict*])

Clase de mapeado que referencia llaves débilmente. Las entradas en el diccionario serán descartadas cuando no haya una referencia fuerte a la llave. Esto puede ser usado para asociar datos con un objeto apropiado por otras partes de una aplicación sin añadir atributos a esos objetos. Esto puede ser especialmente útil con objetos que sobre escriben atributos de acceso.

Nótese que cuando una clave cuyo valor es igual a una clave ya existente (pero no tienen igual identidad) es insertado en el diccionario, el valor es reemplazado pero no se reemplaza la clave existente. Debido a esto, cuando la referencia a la clave original es eliminada, se elimina la entrada en el diccionario:

```
>>> class T(str): pass
...
>>> k1, k2 = T(), T()
>>> d = weakref.WeakKeyDictionary()
>>> d[k1] = 1   # d = {k1: 1}
>>> d[k2] = 2   # d = {k1: 2}
>>> del k1      # d = {}
```

Una solución alternativa sería remover la clave antes de la reasignación:

```
>>> class T(str): pass
...
>>> k1, k2 = T(), T()
>>> d = weakref.WeakKeyDictionary()
>>> d[k1] = 1    # d = {k1: 1}
>>> del d[k1]
>>> d[k2] = 2    # d = {k2: 2}
>>> del k1       # d = {k2: 2}
```

Distinto en la versión 3.9: Se agregó soporte para los operadores | y |=, como se especifica en **PEP 584** [https://peps.python.org/pep-0584/].

Los objetos **`WeakKeyDictionary`** tiene un método adicional que expone las referencias internas directamente. Las referencias no tienen garantía de estar con «vida» en el momento en que son usadas, por lo que el resultado de llamar las referencias necesita ser revisado antes de ser usado. Esto puede ser usado para evitar crear referencias que causen que recolector de basura mantenga las llaves en existencia más tiempo del que necesitan.

WeakKeyDictionary.keyrefs()

Retorna un iterable de las referencias débiles a las llaves.

class weakref.WeakValueDictionary([*dict*])

Clase de mapeado que referencia valores débilmente. Las entradas en el diccionario serán descartadas cuando ya no existan las referencias fuertes a los valores.

Distinto en la versión 3.9: Se agregó soporte para los operadores | y |=, como se especifica en **PEP 584** [https://peps.python.org/pep-0584/].

Los objetos **`WeakValueDictionary`** tienen un método adicional que posee los mismos problemas que el método **`WeakKeyDictionary.keyrefs()`**.

WeakValueDictionary.valuerefs()

Retorna un iterable de las referencias débiles a los valores.

class weakref.WeakSet([*elements*])

Clase Conjunto que mantiene referencias débiles a sus elementos. Un elemento será descartado cuando ya no existan referencias fuertes.

class weakref.WeakMethod(*method*[, *callback*])

Una subclase `ref` personalizada que simula una referencia débil a un método vinculado (i.e., un método definido en una clase y visto en una instancia). Ya que un método atado es efímero, una referencia débil estándar no puede mantenerlo. El `WeakMethod` tiene un código especial para recrear el método atado hasta que o el objeto o la función original muera:

```
>>> class C:
...     def method(self):
...         print("method called!")
...
>>> c = C()
>>> r = weakref.ref(c.method)
>>> r()
>>> r = weakref.WeakMethod(c.method)
>>> r()
<bound method C.method of <__main__.C object at 0x7
>>> r()()
method called!
>>> del c
>>> gc.collect()
0
>>> r()
>>>
```

callback corresponde al parámetro del mismo nombre de la función `ref()`.

Added in version 3.4.

class weakref.finalize(*obj*, *func*, /, **args*, ***kwargs*)

Retorna un objeto finalizador invocable que será llamado cuando *obj* sea recolectado por el recolector de basura. A diferencia de referencias débiles ordinarias, un finalizador siempre sobrevivirá hasta que el objeto de referencia sea recolectado, simplificando enormemente la gestión del ciclo de vida.

Se considera a un finalizador como *vivo* hasta que sea llamado (o explícitamente o en la recolección de basura), y después que esté *muerto*. Llamar a un finalizador vivo retorna el resultado de evaluar `func(*arg, **kwargs)`, mientras que llamar a un finalizador muerto retorna **`None`**.

Las excepciones lanzadas por retrollamadas de finalizadores durante la recolección de basura serán mostradas en la salida de error estándar, pero no pueden ser propagadas. Son gestionados de la misma forma que las excepciones lanzadas por el método **`__del__()`** de un objeto o la retrollamada de una referencia débil.

Cuando el programa sale, cada finalizador vivo que quede es llamado a menos que su atributo **`atexit`** sea falso. Son llamados en el orden reverso de creación.

Un finalizador nunca invocará su retrollamada durante la última parte del apagado del intérprete cuando las variables globales del módulo (globals) están sujetos a ser reemplazados por **`None`**.

call()

> Si *self* está vivo entonces lo marca como muerto y retorna el resultado de llamar a `func(*args, **kwargs)`. Si *self* está muerto entonces retorna **`None`**.

detach()

> Si *self* está vivo entonces lo marca como muerto y retorna la tupla `(obj, func, args, kwargs)`. Si *self* está muerto entonces retorna **`None`**.

peek()

> Si *self* está vivo entonces retorna la tupla `(obj,`

`func, args, kwargs)`. Si *self* está muerto entonces retorna **`None`**.

alive

Propiedad que es verdadera si el finalizador está vivo, caso contrario, falso.

atexit

Una propiedad booleana con permisos de escritura que por defecto es verdadero. Cuando el programa sale, llama a todos los finalizadores vivos que queden para los cuales **`atexit`** es verdadero. Ellos son llamados en el orden reverso de creación.

Nota

Es importante asegurar que *func*, *args* y *kwargs* no sean dueños de ninguna referencia a *obj*, o directamente o indirectamente, ya que de otra manera *obj* nunca será recolectado por el recolector de basura. En particular, *func* no debe ser un método vinculado de *obj*.

Added in version 3.4.

weakref.ReferenceType

El objeto de tipo para objetos de referencias débiles.

weakref.ProxyType

El objeto de tipo para proxies de objetos que no son invocables.

weakref.CallableProxyType

El objeto de tipo para proxies de objetos invocables.

weakref.ProxyTypes

Una secuencia que contiene todos los objetos de tipo para los proxies. Esto puede hacerlo más simple para pruebas si un objeto es un proxy sin ser dependiente en nombrar a ambos tipos proxy.

Ver también

PEP 205 [https://peps.python.org/pep-0205/] - Referencias Débiles
La propuesta y lógica de esta característica, incluyendo los enlaces a implementaciones tempranas e información acerca de características similares en otros lenguajes.

Objetos de referencias débiles

Los objetos de referencias débiles no tiene métodos y atributos aparte de **`ref.__calback__`**. Un objeto de referencia débil permite que el referente sea obtenido, si todavía existe, al llamarlo:

```
>>> import weakref
>>> class Object:
...     pass
...
>>> o = Object()
>>> r = weakref.ref(o)
>>> o2 = r()
>>> o is o2
True
```

Si el referente no existe, llamar al objeto de referencia retorna **`None`**:

```
>>> del o, o2
>>> print(r())
None
```

Probar que un objeto de referencia débil está todavía con vida debe ser hecho usando la expresión `ref() is not None`. Normalmente, el código de aplicación que necesite usar un objeto de referencia debe seguir este patrón:

```
# r is a weak reference object
```

```
o = r()
if o is None:
    # referent has been garbage collected
    print("Object has been deallocated; can't frobnicate
else:
    print("Object is still live!")
    o.do_something_useful()
```

Usar una prueba separada para «vivacidad» crea una condición de carrera en aplicaciones con hilos; otro hilo puede hacer que una referencia débil sea invalidada antes de que la referencia débil sea llamada; El modismo mostrado arriba es seguro en aplicaciones con hilos también como aplicaciones de un sólo hilo.

Versiones especializadas de objetos `ref` pueden ser creadas a través de creación por subclase. Esto es usado en la implementación de `WeakValueDictionary` para reducir la memoria elevada por cada entrada en el mapeado. Esto puede ser lo más útil para asociar información adicional con un referencia, pero también puede ser usado para insertar procesamiento adicional en llamadas para recuperar el referente.

Este ejemplo muestra como una subclase de `ref` puede ser usado para guardar información adicional sobre un objeto y afectar el valor que se retorna cuando el referente es accedido:

```
import weakref

class ExtendedRef(weakref.ref):
    def __init__(self, ob, callback=None, /, **annotatio
        super().__init__(ob, callback)
        self.__counter = 0
        for k, v in annotations.items():
            setattr(self, k, v)

    def __call__(self):
        """Return a pair containing the referent and the
        times the reference has been called.
        """
        ob = super().__call__()
        if ob is not None:
            self.__counter += 1
            ob = (ob, self.__counter)
        return ob
```

Ejemplo

Este simple ejemplo muestra como una aplicación puede usar objetos ID para recuperar objetos que han sido visto antes. Los ID de los objetos pueden ser usados en otras estructuras de datos sin forzar que los objetos permanezcan con vida, pero los objetos pueden aún pueden ser recuperados por el ID si lo hacen.

```
import weakref

_id2obj_dict = weakref.WeakValueDictionary()

def remember(obj):
    oid = id(obj)
    _id2obj_dict[oid] = obj
    return oid

def id2obj(oid):
    return _id2obj_dict[oid]
```

Objetos finalizadores

El principal beneficio de usar **`finalize`** es que hace simple registrar una retrollamada sin necesitar preservar el objeto finalizador retornado. Por ejemplo

```
>>> import weakref
>>> class Object:
...     pass
...
>>> kenny = Object()
>>> weakref.finalize(kenny, print, "You killed Kenny!")
<finalize object at ...; for 'Object' at ...>
```

```
>>> del kenny
You killed Kenny!
```

El finalizador puede ser llamado directamente también. Sin embargo, el finalizador invocará la retrollamada como máximo una vez.

```
>>> def callback(x, y, z):
...     print("CALLBACK")
...     return x + y + z
...
>>> obj = Object()
>>> f = weakref.finalize(obj, callback, 1, 2, z=3)
>>> assert f.alive
>>> assert f() == 6
CALLBACK
>>> assert not f.alive
>>> f()                         # callback not called becaus
>>> del obj                     # callback not called becaus
```

Puedes de-registrar un finalizador usando su método `detach()`. Esto mata el finalizador y retorna los argumentos pasados al constructor cuando fue creado.

```
>>> obj = Object()
>>> f = weakref.finalize(obj, callback, 1, 2, z=3)
>>> f.detach()
(<...Object object ...>, <function callback ...>, (1, 2)
>>> newobj, func, args, kwargs = _
>>> assert not f.alive
>>> assert newobj is obj
>>> assert func(*args, **kwargs) == 6
CALLBACK
```

A menos que pongas el atributo `atexit` a `False`, un finalizador será llamado cuando el programa salga si todavía está con vida. Por ejemplo

```
>>> obj = Object()
>>> weakref.finalize(obj, print, "obj dead or exiting")
```

```
<finalize object at ...; for 'Object' at ...>
>>> exit()
obj dead or exiting
```

Comparando finalizadores con los métodos `__del__()`

Suponga que queremos crear una clase cuyas instancias representan directorios temporales. Los directorios deben ser eliminados con sus contenidos cuando el primero de los siguiente eventos ocurre:

- el objeto es recolectado por el recolector de basura,
- el método **`remove()`** del objeto es llamado, o
- el programa sale.

Podemos intentar implementar la clase usando un método `__del__()` como se muestra a continuación:

```
class TempDir:
    def __init__(self):
        self.name = tempfile.mkdtemp()

    def remove(self):
        if self.name is not None:
            shutil.rmtree(self.name)
            self.name = None

    @property
    def removed(self):
        return self.name is None

    def __del__(self):
        self.remove()
```

A partir de Python 3.4, los métodos `__del__()` ya no impiden que los ciclos de referencia sean recolectados como basura y las variables globales del módulo (globals) ya no son forzados a `None` durante el apagado del intérprete. Por lo tanto, este código debería funcionar sin ningún problema en CPython.

Sin embargo, la gestión de métodos `__del__()` es notoriamente específica a la implementación, ya que depende de detalles internos de la implementación del recolector de basura del intérprete.

Una alternativa más robusta puede ser para definir un finalizador que sólo hace referencia a funciones específicas y objetos que necesite, en vez de tener acceso al estado completo del objeto:

```
class TempDir:
    def __init__(self):
        self.name = tempfile.mkdtemp()
        self._finalizer = weakref.finalize(self, shutil.

    def remove(self):
        self._finalizer()

    @property
    def removed(self):
        return not self._finalizer.alive
```

Definido así, nuestro finalizador sólo recibe una referencia a los detalles que necesita limpiar los directorios apropiadamente. Si el objeto nueva llega a ser recolectado como basura el finalizador aún será llamado al salir.

La otra ventaja de weakref basados en finalizadores es que ellos pueden ser usados para registrar finalizadores para clases donde la definición es controlado por terceros, como un código que corre cuando un módulo es descargado:

```
import weakref, sys
def unloading_module():
    # implicit reference to the module globals from the
weakref.finalize(sys.modules[__name__], unloading_module
```

Nota

Si creas un objeto finalizador en un hilo daemon sólo como el programa sale entonces hay la posibilidad de que el finalizador no llegue a ser llamado. Sin embargo, en un hilo daemonic

`atexit.register()`, `try: ... finally: ...` y `with: ...` no garantizan que la limpieza ocurra tampoco.

`types` — Dynamic type creation and names for built-in types

Código fuente: Lib/types.py [https://github.com/python/cpython/tree/3.14/Lib/types.py]

Este módulo define funciones de utilidad para ayudar en la creación dinámica de tipos nuevos.

Este también define nombres para algunos tipos de objetos que son utilizados por el intérprete estándar de Python, pero no expuestos como integrados como lo son **`int`** o **`str`**.

Por último, este proporciona algunas clases de utilidad y funciones adicionales relacionadas con tipos que no son lo suficientemente fundamentales como para ser integradas.

Creación dinámica de tipos

types.new_class(*name*, *bases=()*, *kwds=None*, *exec_body=None*)

Crea un objeto de clase dinámicamente utilizando la metaclase adecuada.

Los tres primeros argumentos son los componentes que componen un encabezado de definición de clase: el nombre de la clase, las clases base (en orden), los argumentos por palabra clave (tal como `metaclass`).

El argumento *exec_body* es una retrollamada que se usa para rellenar el espacio de nombres de clase recién creado. Debe aceptar el espacio de nombre de clase como su único argumento y actualizar el espacio de nombres directamente con el contenido de la clase. Si no se proporciona ninguna retrollamada, tiene el mismo efecto que pasar `lambda ns: None`.

Added in version 3.3.

types.prepare_class(*name*, *bases=()*, *kwds=None*)

Calcula la metaclase adecuada y crea el espacio de nombre de clase.

Los argumentos son los componentes que constituyen un encabezado de definición de clase: el nombre de la clase, las clases base (en orden) y los argumentos de palabra clave (como `metaclass`).

El valor retornado es una tupla de 3: `metaclass, namespace, kwds`

metaclass es la metaclase adecuada, *namespace* es el espacio de nombre de clase preparado y *kwds* es una copia actualizada del pasado en el argumento *kwds* con cualquier entrada `'metaclass'` eliminada. Si no se pasa ningún argumento *kwds*, será un diccionario vacío.

Added in version 3.3.

Distinto en la versión 3.6: El valor predeterminado para el elemento `namespace` de la tupla retornada ha cambiado. Ahora una asignación de inserción-orden-conservación es utilizada cuando la metaclase no tiene un método `__prepare__`.

Ver también

Metaclases

Detalles completos del proceso de creación de clases soportado por estas funciones

PEP 3115 [https://peps.python.org/pep-3115/] - Metaclases en Python 3000

Se presenta el *hook* de espacio de nombres `__prepare__`

types.resolve_bases(*bases*)

Resuelve las entradas MRO dinámicamente según lo especificado por **PEP 560** [https://peps.python.org/pep-0560/].

Esta función busca elementos en *bases* que no sean instancias de **`type`**, y retorna una tupla donde cada objeto que tenga un método **`__mro_entries__()`** es reemplazado por el resultado desempaquetado de llamar a dicho método. Si un elemento de *bases* es una instancia de **`type`**, o no tiene un método **`__mro_entries__()`**, entonces es incluido sin cambios en la tupla de retorno.

Added in version 3.7.

types.get_original_bases(*cls*, */*)

Retorna la tupla de objetos originalmente dados como bases de *cls* antes de que el método **`__mro_entries__()`** sea llamado en cualquier base (siguiendo los mecanismos establecidos en **PEP 560** [https://peps.python.org/pep-0560/]). Esto es útil para introspección Genéricos.

For classes that have an `__orig_bases__` attribute, this function returns the value of `cls.__orig_bases__`. For classes without the `__orig_bases__` attribute, **`cls.__bases__`** is returned.

Ejemplos:

```
from typing import TypeVar, Generic, NamedTuple, Ty

T = TypeVar("T")
class Foo(Generic[T]): ...
class Bar(Foo[int], float): ...
class Baz(list[str]): ...
Eggs = NamedTuple("Eggs", [("a", int), ("b", str)])
Spam = TypedDict("Spam", {"a": int, "b": str})
```

```
assert Bar.__bases__ == (Foo, float)
assert get_original_bases(Bar) == (Foo[int], float)

assert Baz.__bases__ == (list,)
assert get_original_bases(Baz) == (list[str],)

assert Eggs.__bases__ == (tuple,)
assert get_original_bases(Eggs) == (NamedTuple,)

assert Spam.__bases__ == (dict,)
assert get_original_bases(Spam) == (TypedDict,)

assert int.__bases__ == (object,)
assert get_original_bases(int) == (object,)
```

Added in version 3.12.

Ver también

PEP 560 [https://peps.python.org/pep-0560/] - Soporte principal para módulos de tipo y tipos genéricos

Tipos de Intérpretes Estándar

Este módulo proporciona nombres para muchos de los tipos necesarios para implementar un intérprete de Python. Esto evita deliberadamente incluir algunos de los tipos que surgen sólo accidentalmente durante el procesamiento, tal como el tipo `listiterator`.

El uso típico de estos nombres es para verificar **`isinstance()`** o **`issubclass()`**.

Si se crea una instancia de cualquiera de estos tipos, tenga en cuenta que las firmas pueden variar entre las versiones de Python.

Los nombres estándar son definidos para los siguientes tipos:

types.NoneType

El tipo de **`None`**.

Added in version 3.10.

types.FunctionType
types.LambdaType

El tipo de funciones definidas por el usuario y funciones creadas por expresiones **`lambda`**.

Lanza un auditing event `function.__new__` con el argumento `code`.

El evento auditor solo ocurre para la instanciación directa de objetos de código y no se genera para la compilación normal.

types.GeneratorType

El tipo de iterador generator de objetos, creados por funciones generadoras.

types.CoroutineType

El tipo de objetos coroutine, creados por funciones **`async def`**.

Added in version 3.5.

types.AsyncGeneratorType

El tipo de iterador asynchronous generator de objetos, creados por funciones generadoras asíncronas.

Added in version 3.6.

class types.CodeType(***kwargs*)

The type of code objects such as returned by **`compile()`**.

Lanza un evento auditor `code.__new__` con los argumentos `code`, `filename`, `name`, `argcount`, `posonlyargcount`, `kwonlyargcount`, `nlocals`, `stacksize`, `flags`.

Tenga en cuenta que los argumentos auditados pueden no coincidir con los nombres o posiciones requeridos por el inicializador. El evento auditor solo ocurre para la instanciación directa de objetos de código y no se genera para la compilación normal.

types.CellType

The type for cell objects: such objects are used as containers for a function's closure variables.

Added in version 3.8.

types.MethodType

El tipo de métodos de instancias de clase definidas por el usuario.

types.BuiltinFunctionType
types.BuiltinMethodType

El tipo de funciones integradas como **`len()`** o **`sys.exit()`** y métodos de clases integradas. (Aquí, el término «incorporado» significa «escrito en C».)

types.WrapperDescriptorType

El tipo de métodos de algunos tipos de datos integrados y clases base como **`object.__init__()`** o **`object.__lt__()`**.

Added in version 3.7.

types.MethodWrapperType

El tipo de métodos *bound* de algunos tipos de datos integrados y clases base. Por ejemplo, es el tipo de `object().__str__`.

Added in version 3.7.

types.NotImplementedType

El tipo de **`NotImplemented`**.

Added in version 3.10.

types.MethodDescriptorType

El tipo de métodos de algunos tipos de datos integrados como **`str.join()`**.

Added in version 3.7.

types.ClassMethodDescriptorType

El tipo de métodos de clase *unbound* de algunos tipos de datos integrados como `dict.__dict__['fromkeys']`.

Added in version 3.7.

class types.ModuleType(*name*, *doc=None*)

El tipo de módulos. El constructor toma el nombre del módulo que se va a crear y de forma opcional su docstring.

Ver también

Documentation on module objects

Provides details on the special attributes that can be found on instances of **`ModuleType`**.

`importlib.util.module_from_spec()`

Modules created using the **`ModuleType`** constructor are created with many of their special attributes unset or set to default values. **`module_from_spec()`** provides a more robust way of creating **`ModuleType`** instances which ensures the various attributes are set appropriately.

types.EllipsisType

El tipo de **`Ellipsis`**.

Added in version 3.10.

class types.GenericAlias(*t_origin*, *t_args*)

El tipo de parameterized generics como `list[int]`.

`t_origin` debería ser una clase genérica no parametrizada, como `list`, `tuple` o `dict`. `t_args` debe ser una **`tuple`** (posiblemente de longitud 1) de tipos que parametriza `t_origin`:

```
>>> from types import GenericAlias

>>> list[int] == GenericAlias(list, (int,))
True
>>> dict[str, int] == GenericAlias(dict, (str, int)
True
```

Added in version 3.9.

Distinto en la versión 3.9.2: Este tipo ahora puede heredarse.

Ver también

Tipos de alias genéricos
Documentación detallada sobre instancias de **`types.GenericAlias`**

PEP 585 [https://peps.python.org/pep-0585/] - Sugerencias de tipo genéricas en colecciones estándar
Presentando la clase **`types.GenericAlias`**

class types.UnionType

El tipo de union type expressions.

Added in version 3.10.

Distinto en la versión 3.14: This is now an alias for **`typing.Union`**.

class types.TracebackType(*tb_next*, *tb_frame*, *tb_lasti*, *tb_lineno*)

El tipo de objetos de rastreo como los que se encuentran en `sys.exception().__traceback__`.

Consulte la referencia de lenguaje para obtener detalles de los atributos y operaciones disponibles, y orientación sobre cómo crear *tracebacks* dinámicamente.

types.FrameType

The type of frame objects such as found in **`tb.tb_frame`** if `tb` is a traceback object.

types.GetSetDescriptorType

The type of objects defined in extension modules with `PyGetSetDef`, such as **`FrameType.f_locals`** or `array.array.typecode`. This type is used as descriptor for object attributes; it has the same purpose as the **`property`** type, but for classes defined in extension modules.

types.MemberDescriptorType

El tipo de objetos definidos en módulos de extensión con `PyMemberDef`, como `datetime.timedelta.days`. Este tipo se utiliza como descriptor para miembros de datos C simples que utilizan funciones de conversión estándar; tiene el mismo propósito que el tipo **`property`**, pero para las clases definidas en los módulos de extensión.

In addition, when a class is defined with a **`__slots__`** attribute, then for each slot, an instance of **`MemberDescriptorType`** will be added as an attribute on the class. This allows the slot to appear in the class's **`__dict__`**.

En otras implementaciones de Python, este tipo puede ser idéntico a `GetSetDescriptorType`.

class types.MappingProxyType(*mapping*)

Proxy de solo lectura de un mapeo. Proporciona una vista dinámica en las entradas de la asignación, lo que significa que cuando cambia la asignación, la vista refleja estos cambios.

Added in version 3.3.

Distinto en la versión 3.9: Actualizado para soportar el nuevo operador de unión (`|`) de **PEP 584** [https://peps.python.org/pep-0584/], que simplemente delega al mapeo subyacente.

key in proxy

Retorna `True` si la asignación subyacente tiene una clave *key*, de lo contrario `False`.

proxy[key]

Retorna el elemento de la asignación subyacente con la clave *key*. Lanza un **`KeyError`** si *key* no está en la asignación subyacente.

iter(proxy)

Retorna un iterador sobre las claves de la asignación subyacente. Este es un método abreviado para `iter(proxy.keys())`.

len(proxy)

Retorna el número de elementos de la asignación subyacente.

copy()

Retorna una copia superficial de la asignación subyacente.

get(*key*[, *default*])

Retorna el valor de *key* si *key* está en la asignación subyacente, de lo contrario *default*. Si no se proporciona *default*, el valor predeterminado es `None`, por lo que este método nunca lanza un **`KeyError`**.

items()

Retorna una nueva vista de los elementos de la asignación subyacente (en pares `(key, value)`).

keys()

Retorna una nueva vista de las claves de la asignación subyacente.

values()

Retorna una nueva vista de los valores de la asignación subyacente.

reversed(proxy)

Retorna un iterador inverso sobre las claves de la asignación subyacente.

Added in version 3.9.

hash(proxy)

Retorna un hash del mapeo subyacente.

Added in version 3.12.

class types.CapsuleType

The type of capsule objects.

Added in version 3.13.

Clases y funciones de utilidad adicionales

class types.SimpleNamespace

Una subclase simple `object` que proporciona acceso de atributo a su espacio de nombre, así como una representación significativa.

Unlike `object`, with **`SimpleNamespace`** you can add and remove attributes.

`SimpleNamespace` objects may be initialized in the same way as `dict`: either with keyword arguments, with a single positional argument, or with both. When initialized with keyword arguments, those are directly added to the underlying namespace. Alternatively, when initialized with a positional argument, the underlying namespace will be updated with key-value pairs from that argument (either a mapping object or an iterable object producing key-value pairs). All such keys must be strings.

El tipo es aproximadamente equivalente al código siguiente:

```
class SimpleNamespace:
    def __init__(self, mapping_or_iterable=(), /, *
        self.__dict__.update(mapping_or_iterable)
        self.__dict__.update(kwargs)

    def __repr__(self):
        items = (f"{k}={v!r}" for k, v in self.__di
        return "{}({})".format(type(self).__name__,

    def __eq__(self, other):
        if isinstance(self, SimpleNamespace) and is
           return self.__dict__ == other.__dict__
        return NotImplemented
```

`SimpleNamespace` puede ser útil como reemplazo para `class NS: pass`. Sin embargo, para un tipo de registro estructurado, utilice **`namedtuple()`** en su lugar.

`SimpleNamespace` objects are supported by **`copy.replace()`**.

Added in version 3.3.

Distinto en la versión 3.9: El orden de los atributos en el *repr* cambió de alfabético a orden de inserción (como `dict`).

Distinto en la versión 3.13: Added support for an optional positional argument.

types.DynamicClassAttribute(*fget=None*, *fset=None*, *fdel=None*, *doc=None*)

Acceso de atributo de ruta en una clase para _getattr_.

Se trata de un descriptor, que se utiliza para definir atributos que actúan de forma diferente cuando se accede a través de una instancia y a través de una clase. El acceso a la instancia sigue siendo normal, pero el acceso a un atributo a través de una clase se enrutará al método _getattr_ de la clase; esto se hace lanzando AttributeError.

Esto permite tener propiedades activas en una instancia y tener atributos virtuales en la clase con el mismo nombre (véase `enum.Enum` para obtener un ejemplo).

Added in version 3.4.

Funciones de utilidad de corutina

types.coroutine(*gen_func*)

Esta función transforma una función generator en una coroutine function que retorna una corrutina basada en generador. La corrutina basada en generador sigue siendo un generator iterator, pero también se considera un objeto coroutine y es awaitable.Sin embargo, es posible que no implemente necesariamente el método `__await__()`.

Si *gen_func* es una función generadora, se modificará en el lugar.

Si *gen_func* no es una función generadora, se envolverá. Si retorna una instancia de `collections.abc.Generator`, la instancia se ajustará en un objeto proxy *awaitable*. Todos los demás tipos de objetos se retornarán tal cual.

Added in version 3.5.

`copy` — Shallow and deep copy operations

Código fuente: Lib/copy.py [https://github.com/python/cpython/tree/3.14/Lib/copy.py]

Las declaraciones de asignación en Python no copian objetos, crean enlaces entre un objetivo y un objeto. Para colecciones que son mutables o que contienen elementos mutables, a veces se necesita una copia para que uno pueda cambiar una copia sin cambiar la otra. Este módulo proporciona operaciones genéricas de copia superficial y profunda (explicadas a continuación).

Resumen de la interfaz:

copy.copy(*obj*)

Return a shallow copy of *obj*.

copy.deepcopy(*obj*[, *memo*])

Return a deep copy of *obj*.

copy.replace(*obj*, /, ***changes*)

Creates a new object of the same type as *obj*, replacing fields with values from *changes*.

Added in version 3.13.

exception copy.Error

Levantado para errores específicos del módulo.

La diferencia entre copia superficial y profunda solo es relevante para objetos compuestos (objetos que contienen otros objetos, como listas o instancias de clase):

- Una copia superficial (*shallow copy*) construye un nuevo objeto compuesto y luego (en la medida de lo posible) inserta *references* en él a los objetos encontrados en el original.
- Una copia profunda (*deep copy*) construye un nuevo objeto compuesto y luego, recursivamente, inserta *copias* en él de los objetos encontrados en el original.

A menudo existen dos problemas con las operaciones de copia profunda que no existen con las operaciones de copia superficial:

- Los objetos recursivos (objetos compuestos que, directa o indirectamente, contienen una referencia a sí mismos) pueden causar un bucle recursivo.
- Debido a que la copia profunda copia todo, puede copiar demasiado, como los datos que deben compartirse entre copias.

La función **`deepcopy()`** evita estos problemas al:

- mantener un diccionario `memo` de objetos ya copiados durante la aprobación de copia actual; y
- permitiendo que las clases definidas por el usuario anulen la operación de copia o el conjunto de componentes copiados.

Este módulo no copia tipos como módulo, método, seguimiento de pila, marco de pila, archivo, socket, ventana, o cualquier tipo similar. Sí «copia» funciones y clases (superficiales y profundas), retornando el objeto original sin cambios; Esto es compatible con la forma en que son tratados por el módulo **`pickle`**.

Se pueden hacer copias superficiales de los diccionarios usando **`dict.copy()`**, y de las listas mediante la asignación de una porción de la lista completa, por ejemplo, `lista_copiada = lista_original[:]`.

Classes can use the same interfaces to control copying that they use to control pickling. See the description of module **`pickle`** for information on these methods. In fact, the **`copy`** module uses the registered pickle functions from the **`copyreg`** module.

In order for a class to define its own copy implementation, it can

define special methods `__copy__()` and `__deepcopy__()`.

object.__copy__(*self*)

Called to implement the shallow copy operation; no additional arguments are passed.

object.__deepcopy__(*self*, *memo*)

Called to implement the deep copy operation; it is passed one argument, the *memo* dictionary. If the `__deepcopy__` implementation needs to make a deep copy of a component, it should call the `deepcopy()` function with the component as first argument and the *memo* dictionary as second argument. The *memo* dictionary should be treated as an opaque object.

Function `copy.replace()` is more limited than `copy()` and `deepcopy()`, and only supports named tuples created by `namedtuple()`, `dataclasses`, and other classes which define method `__replace__()`.

object.__replace__(*self*, /, ***changes*)

This method should create a new object of the same type, replacing fields with values from *changes*.

Added in version 3.13.

Ver también

Módulo `pickle`

Discusión de los métodos especiales utilizados para apoyar la recuperación y restauración del estado del objeto.

`pprint` — Data pretty printer

Código fuente: Lib/pprint.py [https://github.com/python/cpython/tree/3.14/Lib/pprint.py]

The **`pprint`** module provides a capability to «pretty-print» arbitrary Python data structures in a form which can be used as input to the interpreter. If the formatted structures include objects which are not fundamental Python types, the representation may not be loadable. This may be the case if objects such as files, sockets or classes are included, as well as many other objects which are not representable as Python literals.

The formatted representation keeps objects on a single line if it can, and breaks them onto multiple lines if they don't fit within the allowed width, adjustable by the *width* parameter defaulting to 80 characters.

Distinto en la versión 3.9: Soporte añadido para imprimir de forma bonita **`types.SimpleNamespace`**.

Distinto en la versión 3.10: Soporte añadido para imprimir de forma bonita **`dataclasses.dataclass`**.

Functions

pprint.pp(*object*, *stream=None*, *indent=1*, *width=80*, *depth=None*, ***, *compact=False*, *sort_dicts=False*, *underscore_numbers=False*)

Prints the formatted representation of *object*, followed by a newline. This function may be used in the interactive interpreter instead of the **`print()`** function for inspecting values. Tip: you can reassign `print = pprint.pp` for use within a scope.

Parámetros:

- **object** – The object to be printed.
- **stream** (file-like object | None) – A file-like object to which the output will be written by calling its `write()` method. If `None` (the default), `sys.stdout` is used.
- **indent** (*int*) – The amount of indentation added for each nesting level.
- **width** (*int*) – The desired maximum number of characters per line in the output. If a structure cannot be formatted within the width constraint, a best effort will be made.
- **depth** (*int* | *None*) – The number of nesting levels which may be printed. If the data structure being printed is too deep, the next contained level is replaced by `...`. If `None` (the default), there is no constraint on the depth of the objects being formatted.
- **compact** (*bool*) – Control the way long sequences are formatted. If `False` (the default), each item of a sequence will be formatted on a separate line, otherwise as many items as will fit within the *width* will be formatted on each output line.
- **sort_dicts** (*bool*) – If `True`, dictionaries will be formatted with their keys sorted, otherwise they will be displayed in insertion order (the default).
- **underscore_numbers** (*bool*) – If `True`, integers will be formatted with the `_` character for a thousands separator, otherwise underscores are not displayed (the default).

```
>>> import pprint
>>> stuff = ['spam', 'eggs', 'lumberjack', 'knights
>>> stuff.insert(0, stuff)
>>> pprint.pp(stuff)
[<Recursion on list with id=...>,
 'spam',
 'eggs',
 'lumberjack',
 'knights',
 'ni']
```

Added in version 3.8.

pprint.pprint(*object*, *stream=None*, *indent=1*, *width=80*, *depth=None*, ***, *compact=False*, *sort_dicts=True*, *underscore_numbers=False*)

Alias for **`pp()`** with *sort_dicts* set to `True` by default, which would automatically sort the dictionaries" keys, you might want to use **`pp()`** instead where it is `False` by default.

pprint.pformat(*object*, *indent=1*, *width=80*, *depth=None*, ***, *compact=False*, *sort_dicts=True*, *underscore_numbers=False*)

Return the formatted representation of *object* as a string. *indent*, *width*, *depth*, *compact*, *sort_dicts* and *underscore_numbers* are passed to the **`PrettyPrinter`** constructor as formatting parameters and their meanings are as described in the documentation above.

pprint.isreadable(*object*)

Determina si la representación formateada de *object* es «legible» o si puede usarse para reconstruir el objeto usando **`eval()`**. Siempre retorna `False` para objetos recursivos.

```
>>> pprint.isreadable(stuff)
False
```

pprint.isrecursive(*object*)

Determine if *object* requires a recursive representation. This function is subject to the same limitations as noted in `saferepr()` below and may raise an `RecursionError` if it fails to detect a recursive object.

pprint.saferepr(*object*)

Return a string representation of *object*, protected against recursion in some common data structures, namely instances of `dict`, `list` and `tuple` or subclasses whose `__repr__` has not been overridden. If the representation of object exposes a recursive entry, the recursive reference will be represented as `<Recursion on typename with id=number>`. The representation is not otherwise formatted.

```
>>> pprint.saferepr(stuff)
"[<Recursion on list with id=...>, 'spam', 'eggs',
```

Objetos *PrettyPrinter*

class pprint.PrettyPrinter(*indent=1, width=80, depth=None, stream=None, *, compact=False, sort_dicts=True, underscore_numbers=False*)

Construct a **`PrettyPrinter`** instance.

Arguments have the same meaning as for `pp()`. Note that they are in a different order, and that *sort_dicts* defaults to `True`.

```
>>> import pprint
>>> stuff = ['spam', 'eggs', 'lumberjack', 'knights
>>> stuff.insert(0, stuff[:])
>>> pp = pprint.PrettyPrinter(indent=4)
>>> pp.pprint(stuff)
[   ['spam', 'eggs', 'lumberjack', 'knights', 'ni']
    'spam',
    'eggs',
    'lumberjack',
    'knights',
    'ni']
```

```
>>> pp = pprint.PrettyPrinter(width=41, compact=Tru
>>> pp.pprint(stuff)
[['spam', 'eggs', 'lumberjack',
  'knights', 'ni'],
 'spam', 'eggs', 'lumberjack', 'knights',
 'ni']
>>> tup = ('spam', ('eggs', ('lumberjack', ('knight
... ('parrot', ('fresh fruit',))))))))
>>> pp = pprint.PrettyPrinter(depth=6)
>>> pp.pprint(tup)
('spam', ('eggs', ('lumberjack', ('knights', ('ni',
```

Distinto en la versión 3.4: Añadido el argumento *compact*.

Distinto en la versión 3.8: Añadido el argumento *sort_dicts*.

Distinto en la versión 3.10: Se agregó el parámetro *underscore_numbers*.

Distinto en la versión 3.11: No longer attempts to write to **`sys.stdout`** if it is `None`.

Las instancias de **`PrettyPrinter`** tienen los siguientes métodos:

PrettyPrinter.pformat(*object*)

Retorna la representación formateada de *object*. Tiene en cuenta las opciones pasadas al constructor de la clase **`PrettyPrinter`**.

PrettyPrinter.pprint(*object*)

Imprime la representación formateada de *object* en la secuencia configurada, seguida de una nueva línea.

Los siguientes métodos proporcionan las implementaciones para las funciones correspondientes con los mismos nombres. Usar estos métodos en una instancia es algo más eficiente, ya que no es necesario crear nuevos objetos **`PrettyPrinter`**.

PrettyPrinter.isreadable(*object*)

Determina si la representación formateada de *object* es «legible» o si se puede usar para reconstruir su valor usando **`eval()`**. Se debe tener en cuenta que se retorna `False` para objetos recursivos. Si el parámetro *depth* de **`PrettyPrinter`** es proporcionado y el objeto es más profundo de lo permitido, también se retorna `False`.

PrettyPrinter.isrecursive(*object*)

Determina si *object* requiere una representación recursiva.

Este método se proporciona como un punto de entrada o método de enlace automático (*hook* en inglés) para permitir que las subclases modifiquen la forma en que los objetos se convierten en cadenas de caracteres. La implementación por defecto utiliza la implementación interna de **`saferepr()`**.

PrettyPrinter.format(*object*, *context*, *maxlevels*, *level*)

Retorna tres valores: la versión formateada de *object* como una cadena de caracteres, una bandera que indica si el resultado es legible y una bandera que indica si se detectó recursividad. El primer argumento es el objeto a representar. El segundo es un diccionario que contiene la **`id()`** de los objetos que son parte del contexto de representación actual (contenedores directos e indirectos para *object* que están afectando a la representación), como las claves; si es necesario representar un objeto que ya está representado en *context*, el tercer valor de retorno será `True`. Las llamadas recursivas al método **`format()`** deben agregar entradas adicionales a los contenedores de este diccionario. El tercer argumento, *maxlevels*, proporciona el límite máximo de recursividad; su valor por defecto es `0`. Este argumento debe pasarse sin modificaciones a las llamadas recursivas. El cuarto argumento, *level*, da el nivel actual; las llamadas recursivas sucesivas deben pasar un valor menor que el de la llamada actual.

Ejemplo

To demonstrate several uses of the `pp()` function and its parameters, let’s fetch information about a project from PyPI [https://pypi.org]:

```
>>> import json
>>> import pprint
>>> from urllib.request import urlopen
>>> with urlopen('https://pypi.org/pypi/sampleproject/1.
...     project_info = json.load(resp)['info']
```

In its basic form, `pp()` shows the whole object:

```
>>> pprint.pp(project_info)
{'author': 'The Python Packaging Authority',
 'author_email': 'pypa-dev@googlegroups.com',
 'bugtrack_url': None,
 'classifiers': ['Development Status :: 3 - Alpha',
                 'Intended Audience :: Developers',
                 'License :: OSI Approved :: MIT License
                 'Programming Language :: Python :: 2',
                 'Programming Language :: Python :: 2.6'
                 'Programming Language :: Python :: 2.7'
                 'Programming Language :: Python :: 3',
                 'Programming Language :: Python :: 3.2'
                 'Programming Language :: Python :: 3.3'
                 'Programming Language :: Python :: 3.4'
                 'Topic :: Software Development :: Buil
 'description': 'A sample Python project\n'
                '=======================\n'
                '\n'
                'This is the description file for the p
                '\n'
                'The file should use UTF-8 encoding and
                'ReStructured Text. It\n'
                'will be used to generate the project w
                'should be written for\n'
                'that purpose.\n'
                '\n'
                'Typical contents for this file would i
```

```
                 'the project, basic\n'
                 'usage examples, etc. Generally, includi
                 'changelog in here is not\n'
                 'a good idea, although a simple "What\'s
                 'most recent version\n'
                 'may be appropriate.',
 'description_content_type': None,
 'docs_url': None,
 'download_url': 'UNKNOWN',
 'downloads': {'last_day': -1, 'last_month': -1, 'last_w
 'home_page': 'https://github.com/pypa/sampleproject',
 'keywords': 'sample setuptools development',
 'license': 'MIT',
 'maintainer': None,
 'maintainer_email': None,
 'name': 'sampleproject',
 'package_url': 'https://pypi.org/project/sampleproject/
 'platform': 'UNKNOWN',
 'project_url': 'https://pypi.org/project/sampleproject/
 'project_urls': {'Download': 'UNKNOWN',
                  'Homepage': 'https://github.com/pypa/s
 'release_url': 'https://pypi.org/project/sampleproject/
 'requires_dist': None,
 'requires_python': None,
 'summary': 'A sample Python project',
 'version': '1.2.0'}
```

El resultado puede limitarse a una cierta profundidad asignando un valor al argumento *depth* (`...` se utiliza para contenidos más «profundos»):

```
>>> pprint.pp(project_info, depth=1)
{'author': 'The Python Packaging Authority',
 'author_email': 'pypa-dev@googlegroups.com',
 'bugtrack_url': None,
 'classifiers': [...],
 'description': 'A sample Python project\n'
                 '=======================\n'
                 '\n'
                 'This is the description file for the pr
                 '\n'
                 'The file should use UTF-8 encoding and 
                 'ReStructured Text. It\n'
                 'will be used to generate the project we
                 'should be written for\n'
                 'that purpose.\n'
                 '\n'
                 'Typical contents for this file would in
                 'the project, basic\n'
                 'usage examples, etc. Generally, includi
                 'changelog in here is not\n'
                 'a good idea, although a simple "What\'s
                 'most recent version\n'
                 'may be appropriate.',
 'description_content_type': None,
 'docs_url': None,
 'download_url': 'UNKNOWN',
 'downloads': {...},
 'home_page': 'https://github.com/pypa/sampleproject',
 'keywords': 'sample setuptools development',
 'license': 'MIT',
 'maintainer': None,
 'maintainer_email': None,
 'name': 'sampleproject',
 'package_url': 'https://pypi.org/project/sampleproject/
 'platform': 'UNKNOWN',
 'project_url': 'https://pypi.org/project/sampleproject/
 'project_urls': {...},
 'release_url': 'https://pypi.org/project/sampleproject/
 'requires_dist': None,
 'requires_python': None,
 'summary': 'A sample Python project',
 'version': '1.2.0'}
```

Además, se puede establecer un valor máximo de caracteres por línea asignando un valor al parámetro *width*. Si un objeto largo no se puede dividir, el valor dado al ancho se anulará y será excedido:

```
>>> pprint.pp(project_info, depth=1, width=60)
{'author': 'The Python Packaging Authority',
 'author_email': 'pypa-dev@googlegroups.com',
 'bugtrack_url': None,
 'classifiers': [...],
 'description': 'A sample Python project\n'
                '=======================\n'
                '\n'
                'This is the description file for the '
                'project.\n'
                '\n'
                'The file should use UTF-8 encoding and
                'written using ReStructured Text. It\n'
                'will be used to generate the project '
                'webpage on PyPI, and should be written
                'for\n'
                'that purpose.\n'
                '\n'
                'Typical contents for this file would '
                'include an overview of the project, '
                'basic\n'
                'usage examples, etc. Generally, includi
                'the project changelog in here is not\n'
                'a good idea, although a simple "What\'s
                'New" section for the most recent versio
                'may be appropriate.',
 'description_content_type': None,
 'docs_url': None,
 'download_url': 'UNKNOWN',
 'downloads': {...},
 'home_page': 'https://github.com/pypa/sampleproject',
 'keywords': 'sample setuptools development',
 'license': 'MIT',
 'maintainer': None,
 'maintainer_email': None,
 'name': 'sampleproject',
 'package_url': 'https://pypi.org/project/sampleproject/
 'platform': 'UNKNOWN',
'project_url': 'https://pypi.org/project/sampleproject/
'project_urls': {...},
'release_url': 'https://pypi.org/project/sampleproject/
'requires_dist': None,
'requires_python': None,
'summary': 'A sample Python project',
'version': '1.2.0'}
```

`reprlib` — Alternate `repr()` implementation

Código fuente: Lib/reprlib.py [https://github.com/python/cpython/tree/3.14/Lib/reprlib.py]

The **`reprlib`** module provides a means for producing object representations with limits on the size of the resulting strings. This is used in the Python debugger and may be useful in other contexts as well.

Este módulo provee una clase, una instancia y una función:

class reprlib.Repr(**, maxlevel=6, maxtuple=6, maxlist=6, maxarray=5, maxdict=4, maxset=6, maxfrozenset=6, maxdeque=6, maxstring=30, maxlong=40, maxother=30, fillvalue='...', indent=None*)

Clase que provee de servicios de formateo útiles en la implementación de funciones similar a la integrada **`repr()`**; los límites de tamaño para diferentes tipos de objetos son añadidos para evitar la generación de representaciones que son excesivamente largas.

The keyword arguments of the constructor can be used as a shortcut to set the attributes of the **`Repr`** instance. Which means that the following initialization:

```
aRepr = reprlib.Repr(maxlevel=3)
```

Is equivalent to:

```
aRepr = reprlib.Repr()
aRepr.maxlevel = 3
```

See section Repr Objects for more information about **`Repr`** attributes.

Distinto en la versión 3.12: Allow attributes to be set via keyword arguments.

reprlib.aRepr

Esta es una instancia de **`Repr`** que es usada para proveer la función **`repr()`** descrita debajo. Cambiar los atributos de este objeto afectará los límites de tamaño usados por **`repr()`** y el depurador de Python.

reprlib.repr(*obj*)

Este es el método **`repr()`** de `aRepr`. Retorna una cadena similar a la retornada por la función integrada del mismo nombre, pero con límites en la mayoría de tamaños.

In addition to size-limiting tools, the module also provides a decorator for detecting recursive calls to **`__repr__()`** and substituting a placeholder string instead.

@reprlib.recursive_repr(*fillvalue='...'*)

Decorator for **`__repr__()`** methods to detect recursive calls within the same thread. If a recursive call is made, the *fillvalue* is returned, otherwise, the usual **`__repr__()`** call is made. For example:

```
>>> from reprlib import recursive_repr
>>> class MyList(list):
...     @recursive_repr()
...     def __repr__(self):
...         return '<' + '|'.join(map(repr, self))
...
>>> m = MyList('abc')
>>> m.append(m)
>>> m.append('x')
>>> print(m)
<'a'|'b'|'c'|...|'x'>
```

Added in version 3.2.

Objetos Repr

Las instancias **`Repr`** proveen varios atributos que pueden ser usados para proporcionar límites de tamaño para las representaciones de diferentes tipos de objetos, y métodos que formatean tipos de objetos específicos.

Repr.fillvalue

> Esta cadena se muestra para referencias recursivas. El valor predeterminado es `...`.
>
> *Added in version 3.11.*

Repr.maxlevel

> Límite de profundidad en la creación de representaciones recursivas. El valor por defecto es `6`.

Repr.maxdict
Repr.maxlist
Repr.maxtuple
Repr.maxset
Repr.maxfrozenset
Repr.maxdeque
Repr.maxarray

> Límites en el número de entradas representadas por el tipo de objeto nombrado. El valor por defecto es `4` para **`maxdict`**, `5` para **`maxarray`**, y `6` para los otros.

Repr.maxlong

> Máximo número de caracteres en la representación para un entero. Los dígitos son eliminados desde el medio. El valor por defecto es `40`.

Repr.maxstring

> Límite en el número de caracteres en la representación de la cadena. Fíjese que la representación «normal» de la cadena es la usada como la fuente de caracteres: si se necesitan secuencias de escape en la representación, estas pueden ser

desordenadas cuando la representación se ha acortado. El valor por defecto es `30`.

Repr.maxother

Este límite es usado para controlar el tamaño de los tipos de objetos para los cuales no hay ningún método de formateo específico en el objeto **`Repr`**. Se aplica de una manera similar a **`maxstring`**. El valor por defecto es `20`.

Repr.indent

If this attribute is set to `None` (the default), the output is formatted with no line breaks or indentation, like the standard **`repr()`**. For example:

```
>>> example = [
...     1, 'spam', {'a': 2, 'b': 'spam eggs', 'c':
>>> import reprlib
>>> aRepr = reprlib.Repr()
>>> print(aRepr.repr(example))
[1, 'spam', {'a': 2, 'b': 'spam eggs', 'c': {3: 4.5
```

If **`indent`** is set to a string, each recursion level is placed on its own line, indented by that string:

```
>>> aRepr.indent = '-->'
>>> print(aRepr.repr(example))
[
-->1,
-->'spam',
-->{
-->-->'a': 2,
-->-->'b': 'spam eggs',
-->-->'c': {
-->-->-->3: 4.5,
-->-->-->6: [],
-->-->},
-->},
-->'ham',
]
```

Setting **`indent`** to a positive integer value behaves as if it was set to a string with that number of spaces:

```
>>> aRepr.indent = 4
>>> print(aRepr.repr(example))
[
    1,
    'spam',
    {
        'a': 2,
        'b': 'spam eggs',
        'c': {
            3: 4.5,
            6: [],
        },
    },
    'ham',
]
```

Added in version 3.12.

Repr.repr(*obj*)

El equivalente a la función integrada **`repr()`** que usa el formateo impuesto por la instancia.

Repr.repr1(*obj*, *level*)

Implementación recursiva usada por **`repr()`**. Este usa el tipo de *obj* para determinar qué método invocar, pasándole *obj* y *level*. Los métodos de tipo específico deben invocar **`repr1()`** para realizar formateo recursivo, con `level - 1` para el valor de *level* en la invocación recursiva.

Repr.repr_TYPE(*obj*, *level*)

Métodos de formateo para tipos específicos son implementados como métodos con un nombre basado en el nombre del tipo. En el nombre del método, **TYPE** es reemplazado por `'_'.join(type(obj).__name__.split())`. El envío a estos métodos es gestionado por **`repr1()`**. Los métodos de tipo específico que necesitan formatear recursivamente un valor deben invocar `self.repr1(subobj, level - 1)`.

Subclasificando Objetos Repr

The use of dynamic dispatching by **`Repr.repr1()`** allows subclasses of **`Repr`** to add support for additional built-in object types or to modify the handling of types already supported. This example shows how special support for file objects could be added:

```
import reprlib
import sys

class MyRepr(reprlib.Repr):

    def repr_TextIOWrapper(self, obj, level):
        if obj.name in {'<stdin>', '<stdout>', '<stderr>
            return obj.name
        return repr(obj)

aRepr = MyRepr()
print(aRepr.repr(sys.stdin))          # prints '<stdin>'
```

```
<stdin>
```

`enum` — Support for enumerations

Added in version 3.4.

Código fuente: Lib/enum.py [https://github.com/python/cpython/tree/3.14/Lib/enum.py]

Important

Esta página contiene la información de referencia de la API. Para obtener información sobre tutoriales y debates sobre temas más avanzados, consulte

- Tutorial básico
- Tutorial avanzado
- Libro de recetas Enum

Una enumeración:

- es un conjunto de nombres simbólicos (miembros) vinculados a valores únicos
- can be iterated over to return its canonical (i.e. non-alias) members in definition order
- usa la sintaxis *call* para retornar miembros por valor
- usa la sintaxis *index* para retornar miembros por nombre

Las enumeraciones se crean mediante la sintaxis `class` o mediante la sintaxis de llamadas a funciones:

```
>>> from enum import Enum

>>> # class syntax
>>> class Color(Enum):
...     RED = 1
...     GREEN = 2
...     BLUE = 3

>>> # functional syntax
>>> Color = Enum('Color', [('RED', 1), ('GREEN', 2), ('
```

Aunque podemos usar la sintaxis `class` para crear enumeraciones, las enumeraciones no son clases normales de Python. Ver ¿En qué se diferencian las enumeraciones? para más detalles.

Nota

Nomenclatura

- The class `Color` is an *enumeration* (or *enum*)
- The attributes `Color.RED`, `Color.GREEN`, etc., are *enumeration members* (or *members*) and are functionally constants.
- The enum members have *names* and *values* (the name of `Color.RED` is `RED`, the value of `Color.BLUE` is `3`, etc.)

Module contents

`EnumType`

El `type` para Enum y sus subclases.

`Enum`

Clase base para crear constantes enumeradas.

`IntEnum`

Clase base para crear constantes enumeradas que también son subclases de `int`. (Notes)

`StrEnum`

Clase base para crear constantes enumeradas que también son subclases de `str`. (Notes)

`Flag`

Clase base para crear constantes enumeradas que se pueden combinar utilizando las operaciones *bitwise* sin perder su membresía `Flag`.

`IntFlag`

Clase base para crear constantes enumeradas que se pueden combinar mediante los operadores bit a bit sin perder su pertenencia a `IntFlag`. Los miembros `IntFlag` también son subclases de `int`. (Notes)

`ReprEnum`

Usado por `IntEnum`, `StrEnum` y `IntFlag` para mantener el `str()` del tipo mixto.

`EnumCheck`

Una enumeración con los valores `CONTINUOUS`, `NAMED_FLAGS` y `UNIQUE`, para usar con `verify()` para garantizar que una enumeración determinada cumpla varias restricciones.

`FlagBoundary`

Una enumeración con los valores

`STRICT`, `CONFORM`, `EJECT` y `KEEP` que permite un control más detallado sobre cómo se tratan los valores no válidos en una enumeración.

`EnumDict`

A subclass of **`dict`** for use when subclassing **`EnumType`**.

`auto`

Las instancias se reemplazan con un valor apropiado para los miembros de Enum. **`StrEnum`** usa de manera predeterminada la versión en minúsculas del nombre del miembro, mientras que otras enumeraciones tienen el valor predeterminado de 1 y aumentan a partir de ahí.

`property()`

Allows **`Enum`** members to have attributes without conflicting with member names. The `value` and `name` attributes are implemented this way.

`unique()`

El decorador de clase Enum que garantiza que solo un nombre esté vinculado a cualquier valor.

`verify()`

Decorador de clase Enum que verifica las restricciones seleccionables por el usuario en una enumeración.

`member()`

Convierta a `obj` en miembro. Se puede utilizar como decorador.

`nonmember()`

No convierta a `obj` en miembro. Se puede utilizar como decorador.

`global_enum()`

Modify the **`str()`** and **`repr()`** of an enum to show its members as belonging to the module instead of its class, and export the enum members to the global namespace.

`show_flag_values()`

Retorna una lista de todos los enteros de potencia de dos contenidos en una bandera.

`enum.bin()`

Like built-in **`bin()`**, except negative values are represented in two's complement, and the leading bit always indicates sign (`0` implies positive, `1` implies negative).

Added in version 3.6: `Flag`, `IntFlag`, `auto`

Added in version 3.11: `StrEnum`, `EnumCheck`, `ReprEnum`, `FlagBoundary`, `property`, `member`, `nonmember`, `global_enum`, `show_flag_values`

Added in version 3.13: `EnumDict`

Data types

class enum.EnumType

EnumType es el metaclass para enumeraciones *enum*. Es posible subclasificar *EnumType*; consulte Subclassing EnumType para obtener más detalles.

`EnumType` is responsible for setting the correct **`__repr__()`**, **`__str__()`**, **`__format__()`**, and **`__reduce__()`** methods on the final *enum*, as well as creating the enum members, properly handling duplicates, providing iteration over the enum class, etc.

Added in version 3.11: Before 3.11 `EnumType` was called `EnumMeta`, which is still available as an alias.

call(*cls, value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Este método se llama de dos maneras diferentes:

- para buscar un miembro existente:

  cls: La clase de enumeración que se llama.

  value: El valor a buscar.

- to use the `cls` enum to create a new enum (only if the existing enum does not have any members):

  cls: La clase de enumeración que se llama.

  value: El nombre

	del nuevo Enum para crear.
names:	Los nombres/valores de los miembros para el nuevo Enum.
module:	El nombre del módulo en el que se crea el nuevo Enum.
qualname:	La ubicación real en el módulo donde se puede encontrar este Enum.
type:	Un tipo de mezcla para el nuevo Enum.
start:	The first integer value for the Enum (used by `auto`).
boundary:	How to handle out-of-range

values from
bit
operations
(**`Flag`** only).

__contains__(*cls*, *member*)

Retorna `True` si el miembro pertenece a `cls`:

```
>>> some_var = Color.RED
>>> some_var in Color
True
>>> Color.RED.value in Color
True
```

Distinto en la versión 3.12: Before Python 3.12, a `TypeError` is raised if a non-Enum-member is used in a containment check.

__dir__(*cls*)

Retorna `['__class__', '__doc__', '__members__', '__module__']` y los nombres de los miembros en *cls*:

```
>>> dir(Color)
['BLUE', 'GREEN', 'RED', '__class__', '__contai
```

__getitem__(*cls*, *name*)

Returns the Enum member in *cls* matching *name*, or raises a **`KeyError`**:

```
>>> Color['BLUE']
<Color.BLUE: 3>
```

__iter__(*cls*)

Retorna cada miembro en *cls* en orden de definición:

```
>>> list(Color)
[<Color.RED: 1>, <Color.GREEN: 2>, <Color.BLUE:
```

len(*cls*)

Returns the number of members in *cls*:

```
>>> len(Color)
3
```

members

Returns a mapping of every enum name to its member, including aliases

reversed(*cls*)

Retorna cada miembro en *cls* en orden de definición inverso:

```
>>> list(reversed(Color))
[<Color.BLUE: 3>, <Color.GREEN: 2>, <Color.RED:
```

class enum.Enum

Enum es la clase base para todas las enumeraciones *enum*.

name

El nombre utilizado para definir el miembro `Enum`:

```
>>> Color.BLUE.name
'BLUE'
```

value

El valor dado al miembro `Enum`:

```
>>> Color.RED.value
1
```

Value of the member, can be set in `__new__()`.

Nota

Valores de miembros de Enum

Member values can be anything: `int`, `str`, etc. If

the exact value is unimportant you may use `auto` instances and an appropriate value will be chosen for you. See `auto` for the details.

While mutable/unhashable values, such as `dict`, `list` or a mutable `dataclass`, can be used, they will have a quadratic performance impact during creation relative to the total number of mutable/unhashable values in the enum.

name

Name of the member.

value

Value of the member, can be set in `__new__()`.

order

No longer used, kept for backward compatibility. (class attribute, removed during class creation).

The `_order_` attribute can be provided to help keep Python 2 / Python 3 code in sync. It will be checked against the actual order of the enumeration and raise an error if the two do not match:

```
>>> class Color(Enum):
...     _order_ = 'RED GREEN BLUE'
...     RED = 1
...     BLUE = 3
...     GREEN = 2
...
Traceback (most recent call last):
...
TypeError: member order does not match _order_:
   ['RED', 'BLUE', 'GREEN']
   ['RED', 'GREEN', 'BLUE']
```

Nota

In Python 2 code the **`_order_`** attribute is necessary as definition order is lost before it can be recorded.

Added in version 3.6.

ignore

`_ignore_` solo se usa durante la creación y se elimina de la enumeración una vez que se completa la creación.

`_ignore_` es una lista de nombres que no se convertirán en miembros y cuyos nombres también se eliminarán de la enumeración completa. Consulte TimePeriod para ver un ejemplo.

Added in version 3.7.

dir(*self*)

Retorna `['__class__', '__doc__', '__module__', 'name', 'value']` y cualquier método público definido en *self._class_*:

```
>>> from enum import Enum
>>> import datetime as dt
>>> class Weekday(Enum):
...     MONDAY = 1
...     TUESDAY = 2
...     WEDNESDAY = 3
...     THURSDAY = 4
...     FRIDAY = 5
...     SATURDAY = 6
...     SUNDAY = 7
...     @classmethod
...     def today(cls):
...         print(f'today is {cls(dt.date.toda
...
>>> dir(Weekday.SATURDAY)
```

```
['__class__', '__doc__', '__eq__', '__hash__',
```

_generate_next_value_(*name*, *start*, *count*, *last_values*)

name:	El nombre del miembro que se está definiendo (por ejemplo, “RED”).
start:	El valor inicial de Enum; el valor predeterminado es 1.
count:	El número de miembros actualmente definidos, sin incluir este.
last_values:	Una lista de los valores anteriores.

A *staticmethod* that is used to determine the next value returned by `auto`.

Nota

For standard `Enum` classes the next value chosen is the highest value seen incremented by one.

For `Flag` classes the next value chosen will be the next highest power-of-two.

This method may be overridden, e.g.:

```
>>> from enum import auto, Enum
>>> class PowersOfThree(Enum):
```

```
...     @staticmethod
...     def _generate_next_value_(name, start,
...         return 3 ** (count + 1)
...     FIRST = auto()
...     SECOND = auto()
...
>>> PowersOfThree.SECOND.value
9
```

Added in version 3.6.

Distinto en la versión 3.13: Prior versions would use the last seen value instead of the highest value.

__init__(*self*, **args*, ***kwds*)

By default, does nothing. If multiple values are given in the member assignment, those values become separate arguments to `__init__`; e.g.

```
>>> from enum import Enum
>>> class Weekday(Enum):
...     MONDAY = 1, 'Mon'
```

`Weekday.__init__()` would be called as `Weekday.__init__(self, 1, 'Mon')`

__init_subclass__(*cls*, ***kwds*)

Un *classmethod* que se usa para configurar más subclases subsiguientes. Por defecto, no hace nada.

missing(*cls*, *value*)

Un *classmethod* para buscar valores que no se encuentran en *cls*. De forma predeterminada, no hace nada, pero se puede anular para implementar un comportamiento de búsqueda personalizado:

```
>>> from enum import auto, StrEnum
>>> class Build(StrEnum):
...     DEBUG = auto()
...     OPTIMIZED = auto()
...     @classmethod
...     def _missing_(cls, value):
...         value = value.lower()
...         for member in cls:
...             if member.value == value:
...                 return member
...         return None
...
>>> Build.DEBUG.value
'debug'
>>> Build('deBUG')
<Build.DEBUG: 'debug'>
```

Added in version 3.6.

`__new__`(*cls*, **args*, ***kwds*)

By default, doesn't exist. If specified, either in the enum class definition or in a mixin class (such as `int`), all values given in the member assignment will be passed; e.g.

```
>>> from enum import Enum
>>> class MyIntEnum(int, Enum):
...     TWENTYSIX = '1a', 16
```

results in the call `int('1a', 16)` and a value of `26` for the member.

Nota

When writing a custom `__new__`, do not use `super().__new__` – call the appropriate `__new__` instead.

`__repr__`(*self*)

Retorna la cadena utilizada para las llamadas *repr()*. De forma predeterminada, retorna el nombre *Enum*, el

nombre del miembro y el valor, pero se puede anular:

```
>>> from enum import auto, Enum
>>> class OtherStyle(Enum):
...     ALTERNATE = auto()
...     OTHER = auto()
...     SOMETHING_ELSE = auto()
...     def __repr__(self):
...         cls_name = self.__class__.__name__
...         return f'{cls_name}.{self.name}'
...
>>> OtherStyle.ALTERNATE, str(OtherStyle.ALTERN
(OtherStyle.ALTERNATE, 'OtherStyle.ALTERNATE',
```

__str__(*self*)

Retorna la cadena utilizada para las llamadas *str()*. De forma predeterminada, retorna el nombre *Enum* y el nombre del miembro, pero se puede anular:

```
>>> from enum import auto, Enum
>>> class OtherStyle(Enum):
...     ALTERNATE = auto()
...     OTHER = auto()
...     SOMETHING_ELSE = auto()
...     def __str__(self):
...         return f'{self.name}'
...
>>> OtherStyle.ALTERNATE, str(OtherStyle.ALTERN
(<OtherStyle.ALTERNATE: 1>, 'ALTERNATE', 'ALTER
```

__format__(*self*)

Returns the string used for *format()* and *f-string* calls. By default, returns **`__str__()`** return value, but can be overridden:

```
>>> from enum import auto, Enum
>>> class OtherStyle(Enum):
...     ALTERNATE = auto()
...     OTHER = auto()
...     SOMETHING_ELSE = auto()
...     def __format__(self, spec):
...         return f'{self.name}'
...
>>> OtherStyle.ALTERNATE, str(OtherStyle.ALTERN
(<OtherStyle.ALTERNATE: 1>, 'OtherStyle.ALTERNA
```

Nota

El uso de **`auto`** con **`Enum`** da como resultado números enteros de valor creciente, comenzando con `1`.

Distinto en la versión 3.12: Added Soporte de Dataclass

_add_alias_()

Adds a new name as an alias to an existing member:

```
>>> Color.RED._add_alias_("ERROR")
>>> Color.ERROR
<Color.RED: 1>
```

Raises a **`NameError`** if the name is already assigned to a different member.

Added in version 3.13.

_add_value_alias_()

Adds a new value as an alias to an existing member:

```
>>> Color.RED._add_value_alias_(42)
>>> Color(42)
<Color.RED: 1>
```

Raises a **`ValueError`** if the value is already linked with a different member.
See MultiValueEnum for an example.

Added in version 3.13.

class enum.IntEnum

IntEnum is the same as `Enum`, but its members are also integers and can be used anywhere that an integer can be used. If any integer operation is performed with an *IntEnum* member, the resulting value loses its enumeration status.

```
>>> from enum import IntEnum
>>> class Number(IntEnum):
...     ONE = 1
...     TWO = 2
...     THREE = 3
...
>>> Number.THREE
<Number.THREE: 3>
>>> Number.ONE + Number.TWO
3
>>> Number.THREE + 5
8
>>> Number.THREE == 3
True
```

Nota

El uso de `auto` con `IntEnum` da como resultado números enteros de valor creciente, comenzando con `1`.

Distinto en la versión 3.11: `__str__()` is now **`int.__str__()`** to better support the *replacement of existing constants* use-case. `__format__()` was already **`int.__format__()`** for that same reason.

class enum.StrEnum

StrEnum is the same as `Enum`, but its members are also strings and can be used in most of the same places that a string can be used. The result of any string operation performed on or with a *StrEnum* member is not part of the enumeration.

```
>>> from enum import StrEnum, auto
>>> class Color(StrEnum):
...     RED = 'r'
...     GREEN = 'g'
...     BLUE = 'b'
...     UNKNOWN = auto()
...
>>> Color.RED
<Color.RED: 'r'>
>>> Color.UNKNOWN
<Color.UNKNOWN: 'unknown'>
>>> str(Color.UNKNOWN)
'unknown'
```

Nota

There are places in the stdlib that check for an exact **`str`** instead of a **`str`** subclass (i.e. `type(unknown) == str` instead of `isinstance(unknown, str)`), and in those locations you will need to use `str(MyStrEnum.MY_MEMBER)`.

Nota

El uso de **`auto`** con **`StrEnum`** da como resultado el nombre de miembro en minúsculas como valor.

Nota

`__str__()` is **`str.__str__()`** to better support the *replacement of existing constants* use-case. **`__format__()`** is likewise **`str.__format__()`** for that same reason.

Added in version 3.11.

class enum.Flag

`Flag` is the same as **`Enum`**, but its members support the bitwise operators `&` (*AND*), `|` (*OR*), `^` (*XOR*), and `~` (*INVERT*); the results of those operations are (aliases of) members of the enumeration.

contains(*self*, *value*)

Retorna *True* si el valor está en sí mismo:

```
>>> from enum import Flag, auto
>>> class Color(Flag):
...     RED = auto()
...     GREEN = auto()
...     BLUE = auto()
...
>>> purple = Color.RED | Color.BLUE
>>> white = Color.RED | Color.GREEN | Color.BLU
>>> Color.GREEN in purple
False
>>> Color.GREEN in white
True
>>> purple in white
True
>>> white in purple
False
```

iter(self):

Returns all contained non-alias members:

```
>>> list(Color.RED)
[<Color.RED: 1>]
>>> list(purple)
[<Color.RED: 1>, <Color.BLUE: 4>]
```

Added in version 3.11.

len(self):

Retorna el número de miembros en la bandera:

```
>>> len(Color.GREEN)
1
>>> len(white)
3
```

Added in version 3.11.

bool(self):

Retorna *True* si hay algún miembro en la bandera, *False* de lo contrario:

```
>>> bool(Color.GREEN)
True
>>> bool(white)
True
>>> black = Color(0)
>>> bool(black)
False
```

_or_(*self*, *other*)

Retorna la bandera actual binaria o con otra:

```
>>> Color.RED | Color.GREEN
<Color.RED|GREEN: 3>
```

and(*self*, *other*)

Retorna el binario de la bandera actual y se combina con otro:

```
>>> purple & white
<Color.RED|BLUE: 5>
>>> purple & Color.GREEN
<Color: 0>
```

xor(*self*, *other*)

Retorna la bandera actual binaria xor'ed con otra:

```
>>> purple ^ white
<Color.GREEN: 2>
>>> purple ^ Color.GREEN
<Color.RED|GREEN|BLUE: 7>
```

invert(self):

Returns all the flags in *type(self)* that are not in *self*:

```
>>> ~white
<Color: 0>
>>> ~purple
<Color.GREEN: 2>
>>> ~Color.RED
<Color.GREEN|BLUE: 6>
```

_numeric_repr_()

Función utilizada para dar formato a los valores numéricos restantes sin nombre. El valor predeterminado es la repr del valor; las opciones comunes son **`hex()`** y **`oct()`**.

Nota

El uso de **`auto`** con **`Flag`** da como resultado números enteros que son potencias de dos, comenzando con `1`.

Distinto en la versión 3.11: The *repr()* of zero-valued flags has changed. It is now:

```
>>> Color(0)
<Color: 0>
```

class enum.IntFlag

`IntFlag` is the same as **`Flag`**, but its members are also integers and can be used anywhere that an integer can be used.

```
>>> from enum import IntFlag, auto
>>> class Color(IntFlag):
...     RED = auto()
...     GREEN = auto()
...     BLUE = auto()
...
>>> Color.RED & 2
```

```
<Color: 0>
>>> Color.RED | 2
<Color.RED|GREEN: 3>
```

Si se realiza alguna operación con enteros con un miembro *IntFlag*, el resultado no es un *IntFlag*:

```
>>> Color.RED + 2
3
```

If a `Flag` operation is performed with an *IntFlag* member and:

- el resultado es un *IntFlag* válido: se retorna un *IntFlag*
- the result is not a valid *IntFlag*: the result depends on the `FlagBoundary` setting

The `repr()` of unnamed zero-valued flags has changed. It is now:

```
>>> Color(0)
<Color: 0>
```

Nota

El uso de `auto` con `IntFlag` da como resultado números enteros que son potencias de dos, comenzando con `1`.

Distinto en la versión 3.11: `__str__()` is now `int.__str__()` to better support the *replacement of existing constants* use-case. `__format__()` was already `int.__format__()` for that same reason.

Inversion of an `IntFlag` now returns a positive value that is the union of all flags not in the given flag, rather than a negative value. This matches the existing `Flag` behavior.

class enum.ReprEnum

`ReprEnum` uses the `repr()` of `Enum`, but the `str()` of the mixed-in data type:

- **`int.__str__()`** para `IntEnum` y `IntFlag`
- **`str.__str__()`** para `StrEnum`

Inherit from **`ReprEnum`** to keep the `str()` / `format()` of the mixed-in data type instead of using the `Enum`-default `str()`.

Added in version 3.11.

class enum.EnumCheck

EnumCheck contiene las opciones utilizadas por el decorador `verify()` para garantizar diversas restricciones; las restricciones fallidas dan como resultado un `ValueError`.

UNIQUE

Asegúrese de que cada valor tenga un solo nombre:

```
>>> from enum import Enum, verify, UNIQUE
>>> @verify(UNIQUE)
... class Color(Enum):
...     RED = 1
...     GREEN = 2
...     BLUE = 3
...     CRIMSON = 1
Traceback (most recent call last):
...
ValueError: aliases found in <enum 'Color'>: C
```

CONTINUOUS

Asegúrese de que no falten valores entre el miembro de menor valor y el miembro de mayor valor:

```
>>> from enum import Enum, verify, CONTINUOUS
>>> @verify(CONTINUOUS)
... class Color(Enum):
...     RED = 1
...     GREEN = 2
...     BLUE = 5
Traceback (most recent call last):
```

```
...
ValueError: invalid enum 'Color': missing value
```

NAMED_FLAGS

Ensure that any flag groups/masks contain only named flags – useful when values are specified instead of being generated by **`auto()`**:

```
>>> from enum import Flag, verify, NAMED_FLAGS
>>> @verify(NAMED_FLAGS)
... class Color(Flag):
...     RED = 1
...     GREEN = 2
...     BLUE = 4
...     WHITE = 15
...     NEON = 31
Traceback (most recent call last):
...
ValueError: invalid Flag 'Color': aliases WHITE
```

Nota

CONTINUOUS y NAMED_FLAGS están diseñados para funcionar con miembros con valores enteros.

Added in version 3.11.

class enum.FlagBoundary

`FlagBoundary` controls how out-of-range values are handled in **`Flag`** and its subclasses.

STRICT

Out-of-range values cause a **`ValueError`** to be raised. This is the default for **`Flag`**:

```
>>> from enum import Flag, STRICT, auto
>>> class StrictFlag(Flag, boundary=STRICT):
...     RED = auto()
...     GREEN = auto()
...     BLUE = auto()
...
>>> StrictFlag(2**2 + 2**4)
Traceback (most recent call last):
...
ValueError: <flag 'StrictFlag'> invalid value
    given 0b0 10100
  allowed 0b0 00111
```

CONFORM

Out-of-range values have invalid values removed, leaving a valid `Flag` value:

```
>>> from enum import Flag, CONFORM, auto
>>> class ConformFlag(Flag, boundary=CONFORM):
...     RED = auto()
...     GREEN = auto()
...     BLUE = auto()
...
>>> ConformFlag(2**2 + 2**4)
<ConformFlag.BLUE: 4>
```

EJECT

Out-of-range values lose their `Flag` membership and revert to `int`.

```
>>> from enum import Flag, EJECT, auto
>>> class EjectFlag(Flag, boundary=EJECT):
...     RED = auto()
...     GREEN = auto()
...     BLUE = auto()
...
>>> EjectFlag(2**2 + 2**4)
20
```

KEEP

Out-of-range values are kept, and the `Flag` membership is kept. This is the default for **`IntFlag`**:

```
>>> from enum import Flag, KEEP, auto
>>> class KeepFlag(Flag, boundary=KEEP):
...     RED = auto()
...     GREEN = auto()
...     BLUE = auto()
...
>>> KeepFlag(2**2 + 2**4)
<KeepFlag.BLUE|16: 20>
```

Added in version 3.11.

class enum.EnumDict

EnumDict is a subclass of **`dict`** that is used as the namespace for defining enum classes (see Preparando el espacio de nombres de la clase). It is exposed to allow subclasses of **`EnumType`** with advanced behavior like having multiple values per member. It should be called with the name of the enum class being created, otherwise private names and internal classes will not be handled correctly.

Note that only the **`MutableMapping`** interface (**`__setitem__()`** and **`update()`**) is overridden. It may be possible to bypass the checks using other **`dict`** operations like **`|=`**.

member_names

A list of member names.

Added in version 3.13.

Nombres soportados `__dunder__`

`__members__` is a read-only ordered mapping of `member_name:member` items. It is only available on the class.

`__new__()`, if specified, must create and return the enum members; it is also a very good idea to set the member's **`_value_`** appropriately. Once all the members are created it is no longer used.

Nombres **`_sunder_`** compatibles

- **`_name_`** – name of the member
- **`_value_`** – value of the member; can be set in `__new__`
- **`_missing_()`** – a lookup function used when a value is not found; may be overridden
- **`_ignore_`** – a list of names, either as a **`list`** or a **`str`**, that will not be transformed into members, and will be removed from the final class
- **`_order_`** – no longer used, kept for backward compatibility (class attribute, removed during class creation)
- **`_generate_next_value_()`** – used to get an appropriate value for an enum member; may be overridden
- **`_add_alias_()`** – adds a new name as an alias to an existing member.
- **`_add_value_alias_()`** – adds a new value as an alias to an existing member.
- While `_sunder_` names are generally reserved for the further development of the **`Enum`** class and can not be used, some are explicitly allowed:
  - `_repr_*` (e.g. `_repr_html_`), as used in IPython's rich display [https://ipython.readthedocs.io/en/stable/config/integrating.html#rich-display]

Added in version 3.6: `_missing_`, `_order_`, `_generate_next_value_`

Added in version 3.7: `_ignore_`

Added in version 3.13: `_add_alias_`, `_add_value_alias_`, `_repr_*`

Utilities and decorators

class enum.auto

auto can be used in place of a value. If used, the *Enum* machinery will call an `Enum`'s `_generate_next_value_()` to get an appropriate value. For **`Enum`** and `IntEnum` that appropriate value will be the last value plus one; for `Flag` and `IntFlag` it will be the first power-of-two greater than the highest value; for `StrEnum` it will be the lower-cased version of the member's name. Care must be taken if mixing *auto()* with manually specified values.

auto instances are only resolved when at the top level of an assignment, either by itself or as part of a tuple:

- `FIRST = auto()` will work (auto() is replaced with `1`);
- `SECOND = auto(), -2` will work (auto is replaced with `2`, so `2, -2` is used to create the `SECOND` enum member;
- `THREE = [auto(), -3]` will *not* work (`[<auto instance>, -3]` is used to create the `THREE` enum member)

Distinto en la versión 3.11.1: In prior versions, `auto()` had to be the only thing on the assignment line to work properly.

`_generate_next_value_` se puede anular para personalizar los valores utilizados por *auto*.

Nota

in 3.13 the default `_generate_next_value_` will always return the highest member value incremented by 1, and will fail if any member is an incompatible type.

@enum.property

Un decorador similar al *property* integrado, pero específico para enumeraciones. Permite que los atributos de los miembros tengan los mismos nombres que los propios miembros.

Nota

el *property* y el miembro deben definirse en clases separadas; por ejemplo, los atributos *value* y *name* se definen en la clase *Enum* y las subclases *Enum* pueden definir miembros con los nombres `value` y `name`.

Added in version 3.11.

@enum.unique

A `class` decorator specifically for enumerations. It searches an enumeration's `__members__`, gathering any aliases it finds; if any are found `ValueError` is raised with the details:

```
>>> from enum import Enum, unique
>>> @unique
... class Mistake(Enum):
...     ONE = 1
...     TWO = 2
...     THREE = 3
...     FOUR = 3
...
Traceback (most recent call last):
...
ValueError: duplicate values found in <enum 'Mistak
```

@enum.verify

Un decorador `class` específicamente para enumeraciones. Los miembros de `EnumCheck` se utilizan para especificar qué restricciones deben verificarse en la enumeración decorada.

Added in version 3.11.

@enum.member

Un decorador para usar en enumeraciones: su objetivo se convertirá en miembro.

Added in version 3.11.

@enum.nonmember

Un decorador para usar en enumeraciones: su destino no se convertirá en miembro.

Added in version 3.11.

@enum.global_enum

Un decorador para cambiar el `str()` y `repr()` de una enumeración para mostrar sus miembros como pertenecientes al módulo en lugar de a su clase. Solo debe usarse cuando los miembros de la enumeración se exportan al espacio de nombres global del módulo (consulte `re.RegexFlag` para ver un ejemplo).

Added in version 3.11.

enum.show_flag_values(*value*)

Retorna una lista de todos los enteros de potencia de dos contenidos en un indicador *value*.

Added in version 3.11.

enum.bin(*num*, *max_bits=None*)

Like built-in `bin()`, except negative values are represented in two's complement, and the leading bit always indicates sign (`0` implies positive, `1` implies negative).

```
>>> import enum
>>> enum.bin(10)
'0b0 1010'
>>> enum.bin(~10)   # ~10 is -11
'0b1 0101'
```

Added in version 3.11.

Notas

`IntEnum`, **`StrEnum`** y **`IntFlag`**

Estos tres tipos de enumeración están diseñados para ser reemplazos directos de los valores existentes basados en cadenas y enteros; como tales, tienen limitaciones adicionales:

- `__str__` usa el valor y no el nombre del miembro de enumeración
- `__format__`, debido a que usa `__str__`, también usará el valor del miembro de enumeración en lugar de su nombre

Si no necesita/quiere esas limitaciones, puede crear su propia clase base mezclando el tipo `int` o `str` usted mismo:

```
>>> from enum import Enum
>>> class MyIntEnum(int, Enum):
...     pass
```

o puede reasignar el **`str()`** apropiado, etc., en su enumeración:

```
>>> from enum import Enum, IntEnum
>>> class MyIntEnum(IntEnum):
...     __str__ = Enum.__str__
```

`graphlib` — Functionality to operate with graph-like structures

Código fuente: Lib/graphlib.py [https://github.com/python/cpython/tree/3.14/Lib/graphlib.py]

---

class graphlib.TopologicalSorter(*graph=None*)

Provee una funcionalidad para ordenar topológicamente un grafo de nodos hashable.

Un ordenamiento topológico es un ordenamiento lineal de los vértices en un grafo de modo que para cada arista dirigida u -> v desde el vértice u al vértice v, el vértice u viene antes del vértice v en el ordenamiento. Por ejemplo, los vértices del grafo pueden representar tareas a realizar y las aristas pueden representar restricciones de que una tarea debe realizarse antes que otra; en este ejemplo, un ordenamiento topológico es solo una secuencia válida para las tareas. Es posible un ordenamiento topológico completo si y solo si el grafo no tiene ciclos dirigidos, es decir, si es un grafo acíclico dirigido.

Si se proporciona el argumento opcional *graph*, este debe ser un diccionario que represente un grafo acíclico dirigido donde las claves son nodos y los valores son iterables de todos los predecesores de ese nodo en el grafo (los nodos que tienen las aristas que apuntan al valor clave). Se pueden agregar nodos adicionales al grafo utilizando el `add()` method.

En el caso general, los pasos necesarios para realizar el ordenamiento de un grafo son los siguientes:

- Cree una instancia `TopologicalSorter` con un grafo inicial opcional.

- Agregue nodos adicionales al grafo.
- Llame a `prepare()` en el grafo.
- Mientras `is_active()` is `True`, itere sobre los nodos retornados por `get_ready()` y procéselos . Llame a `done()` en cada nodo a medida que finaliza el procesamiento.

En caso de que sólo se requiera una ordenación inmediata de los nodos del grafo y no haya paralelismo, se puede utilizar directamente el método de conveniencia `TopologicalSorter.static_order()`.

```
>>> graph = {"D": {"B", "C"}, "C": {"A"}, "B": {"A"
>>> ts = TopologicalSorter(graph)
>>> tuple(ts.static_order())
('A', 'C', 'B', 'D')
```

La clase está diseñada para soportar fácilmente el procesamiento en paralelo de los nodos a medida que estén listos. Para la instancia:

```
topological_sorter = TopologicalSorter()

# Add nodes to 'topological_sorter'...

topological_sorter.prepare()
while topological_sorter.is_active():
    for node in topological_sorter.get_ready():
        # Worker threads or processes take nodes to
        # 'task_queue' queue.
        task_queue.put(node)

    # When the work for a node is done, workers put
    # 'finalized_tasks_queue' so we can get more no
    # The definition of 'is_active()' guarantees th
    # least one node has been placed on 'task_queue
    # been passed to 'done()', so this blocking 'ge
    # succeed.  After calling 'done()', we loop bac
    # again, so put newly freed nodes on 'task_queu
    # logically possible.
```

```
node = finalized_tasks_queue.get()
topological_sorter.done(node)
```

add(*node*, **predecessors*)

Añade un nuevo nodo y sus predecesores al grafo. Tanto el *node* como todos los elementos de *predecessors* deben ser hashable.

Si se llama varias veces con el mismo argumento del nodo, el conjunto de dependencias será la unión de todas las dependencias pasadas.

Es posible añadir un nodo sin dependencias (no se proporciona *predecessors*) o proporcionar una dependencia dos veces. Si un nodo que no se ha proporcionado antes se incluye entre los *predecessors*, se añadirá automáticamente al grafo sin predecesores propios.

Provoca **`ValueError`** si se llama después de **`prepare()`**.

prepare()

Marca el grafo como terminado y comprueba si existen ciclos en el grafo. Si se detecta algún ciclo, se lanzará **`CycleError`**, pero se puede seguir utilizando **`get_ready()`** para obtener tantos nodos como sea posible hasta que los ciclos bloqueen más el progreso. Después de una llamada a esta función, el grafo no puede ser modificado, y por lo tanto no se pueden añadir más nodos utilizando **`add()`**.

A **`ValueError`** will be raised if the sort has been started by **`static_order()`** or **`get_ready()`**.

Distinto en la versión 3.14: `prepare()` can now be called more than once as long as the sort has not started. Previously this raised **`ValueError`**.

is_active()

Retorna `True` si se puede avanzar más y `False` en caso contrario. Se puede avanzar si los ciclos no bloquean la resolución y, o bien todavía existen nodos listos que aún no han sido retornados por **`TopologicalSorter.get_ready()`** o el número de nodos marcados con **`TopologicalSorter.done()`** es menor que el número que han sido retornados por **`TopologicalSorter.get_ready()`**.

El método **`__bool__()`** de esta clase defiere a esta función, por lo que en lugar de:

```
if ts.is_active():
    ...
```

es posible hacer simplemente:

```
if ts:
    ...
```

Lanzar **`ValueError`** si se llama sin llamar previamente a **`prepare()`**.

done(**nodes*)

Marca un conjunto de nodos retornados por **`TopologicalSorter.get_ready()`** como procesados, desbloqueando cualquier sucesor de cada nodo en *nodes* para ser retornado en el futuro por una llamada a **`TopologicalSorter.get_ready()`**.

Lanza **`ValueError`** si algún nodo de *nodes* ya ha sido marcado como procesado por una llamada anterior a este método o si un nodo no fue añadido al grafo usando **`TopologicalSorter.add()`**, si se llama sin llamar a **`prepare()`** o si el nodo aún no ha sido retornado por **`get_ready()`**.

get_ready()

Retorna una `tupla` con todos los nodos que están listos. Inicialmente retorna todos los nodos sin predecesores, y una vez que éstos se marcan como procesados llamando a **`TopologicalSorter.done()`**, las llamadas posteriores devolverán todos los nuevos nodos que tengan todos sus predecesores ya procesados. Una vez que no se puede avanzar más, se retornan tuplas vacías.

Lanzar **`ValueError`** si se llama sin llamar previamente a **`prepare()`**.

static_order()

Retorna un iterable de nodos en un ordenamiento topológico. El uso de este método no requiere llamar a **`TopologicalSorter.prepare()`** o **`TopologicalSorter.done()`**. Este método es equivalente a:

```
def static_order(self):
    self.prepare()
    while self.is_active():
        node_group = self.get_ready()
        yield from node_group
        self.done(*node_group)
```

El orden concreto que se retorna puede depender del orden específico en que se insertaron los elementos en el grafo. Por ejemplo:

```
>>> ts = TopologicalSorter()
>>> ts.add(3, 2, 1)
>>> ts.add(1, 0)
>>> print([*ts.static_order()])
[2, 0, 1, 3]

>>> ts2 = TopologicalSorter()
>>> ts2.add(1, 0)
>>> ts2.add(3, 2, 1)
>>> print([*ts2.static_order()])
[0, 2, 1, 3]
```

Esto se debe a que «0» y «2» están en el mismo nivel en el gráfico (habrían sido retornados en la misma llamada a `get_ready()`) y el orden entre ellos está determinado por el orden de inserción.

Si se detecta algún ciclo, se lanzará `CycleError`.

Added in version 3.9.

Excepciones

The **`graphlib`** module defines the following exception classes:

exception graphlib.CycleError

Subclase de `ValueError` planteada por `TopologicalSorter.prepare()` si existen ciclos en el grafo de trabajo. Si existen múltiples ciclos, sólo se informará de una elección indefinida entre ellos y se incluirá en la excepción.

Se puede acceder al ciclo detectado a través del segundo elemento del atributo `args` de la instancia de la excepción y consiste en una lista de nodos, tal que cada nodo este, en el grafo, un predecesor inmediato del siguiente nodo en la lista. En la lista reportada, el primer y el último nodo serán el mismo, para dejar claro que es cíclico.

Módulos numéricos y matemáticos

Los módulos descritos en este capítulo proporcionan funciones y tipos de datos numéricos y relacionados con las matemáticas. El módulo `numbers` define una jerarquía abstracta de tipos numéricos. Los módulos `math` y `cmath` contienen varias funciones matemáticas para números complejos y de punto flotante. El módulo `decimal` admite representaciones exactas de números decimales, utilizando aritmética de precisión arbitraria.

En este capítulo se documentan los siguientes módulos:

- `numbers` — Numeric abstract base classes
    - La torre numérica
    - Notes for type implementers
        - Agregar más *ABCs* numéricos
        - Implementar operaciones aritméticas
- `math` — Mathematical functions
    - Number-theoretic functions
    - Floating point arithmetic
    - Floating point manipulation functions
    - Power, exponential and logarithmic functions
    - Summation and product functions
    - Conversión angular
    - Funciones trigonométricas
    - Funciones hiperbólicas
    - Funciones especiales
    - Constantes
- `cmath` — Mathematical functions for complex numbers

    - Conversión a y desde coordenadas polares
    - Funciones logarítmicas y de potencias
    - Funciones trigonométricas
    - Funciones hiperbólicas
    - Funciones de clasificación
    - Constantes

- **`decimal`** — Decimal fixed-point and floating-point arithmetic

    - Quick-start tutorial
    - Objetos Decimal

        - Operandos lógicos

    - Objetos context
    - Constantes
    - Modos de redondeo
    - Señales
    - Floating-point notes

        - Mitigación del error de redondeo usando mayor precisión
        - Valores especiales

    - Trabajando con hilos
    - Casos prácticos
    - Preguntas frecuentes sobre decimal

- **`fractions`** — Rational numbers
- **`random`** — Generate pseudo-random numbers

    - Funciones de contabilidad
    - Funciones para los bytes
    - Funciones para enteros
    - Funciones para secuencias
    - Discrete distributions
    - Distribuciones para los nombres reales
    - Generador alternativo
    - Notas sobre la Reproducibilidad
    - Ejemplos

    - Recetas
    - Command-line usage
    - Command-line example
- `statistics` — Mathematical statistics functions
    - Promedios y medidas de tendencia central
    - Medidas de dispersión
    - Estadísticas para relaciones entre dos entradas
    - Detalles de las funciones
    - Excepciones
    - Objetos `NormalDist`
    - Examples and Recipes
        - Problemas de probabilidad clásicos
        - Entradas de Monte Carlo para simulaciones
        - Aproximación de la distribución binomial
        - Clasificador bayesiano ingenuo

`numbers` — Numeric abstract base classes

Código fuente: Lib/numbers.py [https://github.com/python/cpython/tree/3.14/Lib/numbers.py]

The **`numbers`** module (**PEP 3141** [https://peps.python.org/pep-3141/]) defines a hierarchy of numeric abstract base classes which progressively define more operations. None of the types defined in this module are intended to be instantiated.

class numbers.Number

La raíz de la jerarquía numérica. Si desea validar si un argumento *x* es un número, sin importar su tipo, use `isinstance(x, Number)`.

La torre numérica

class numbers.Complex

Las subclases de este tipo describen números complejos e incluyen las operaciones que funcionan en el tipo **`complex`** integrado. Estos son: conversiones a **`complex`** y **`bool`**, **`real`**, **`imag`**, `+`, `-`, `*`, `/`, `**`, **`abs()`**, **`conjugate()`**, `==` y `!=`. Todos excepto `-` y `!=` son abstractos.

real

Abstracto. Recupera el componente real de este número.

imag

Abstracto. Recupera el componente imaginario de este número.

abstractmethod conjugate()

Abstracto. Retorna el complejo conjugado. Por ejemplo, `(1+3j).conjugate() == (1-3j)`.

class numbers.Real

To `Complex`, **`Real`** adds the operations that work on real numbers.

En resumen, estos son: conversiones a `float`, `math.trunc()`, `round()`, `math.floor()`, `math.ceil()`, `divmod()`, `//`, `%`, `<`, `<=`, `>`, y `>=`.

Real también proporciona valores predeterminados para `complex()`, `real`, `imag`, y `conjugate()`.

class numbers.Rational

Subtipos `Real` y agrega propiedades de `numerator` y `denominator`. También proporciona un valor predeterminado para `float()`.

Los valores del `numerator` y del `denominator` deben ser instancias de `Integral` y deben estar en términos mínimos con **`denominator`** positivo.

numerator

Abstract. The numerator of this rational number.

denominator

Abstract. The denominator of this rational number.

class numbers.Integral

Subtipos `Rational` y agrega una conversión a `int`. Proporciona valores predeterminados para `float()`, `numerator` y `denominator`. Agrega métodos abstractos para `pow()` con operaciones de módulo y cadena de bits: `<<`, `>>`, `&`, `^`, `|`, `~`.

Notes for type implementers

Implementers should be careful to make equal numbers equal and hash them to the same values. This may be subtle if there are two different extensions of the real numbers. For example, `fractions.Fraction` implements `hash()` as follows:

```
def __hash__(self):
    if self.denominator == 1:
        # Get integers right.
        return hash(self.numerator)
    # Expensive check, but definitely correct.
    if self == float(self):
        return hash(float(self))
    else:
        # Use tuple's hash to avoid a high collision rat
        # simple fractions.
        return hash((self.numerator, self.denominator))
```

Agregar más *ABCs* numéricos

Por supuesto, hay más *ABCs* posibles para los números, y esto sería una jerarquía deficiente si se excluye la posibilidad de añadirlos. Puede usar `MyFoo` entre `Complex` y `Real` así:

```
class MyFoo(Complex): ...
MyFoo.register(Real)
```

Implementar operaciones aritméticas

We want to implement the arithmetic operations so that mixed-mode operations either call an implementation whose author knew about the types of both arguments, or convert both to the nearest built in type and do the operation there. For subtypes of `Integral`, this means that `__add__()` and `__radd__()` should be defined as:

```
class MyIntegral(Integral):

    def __add__(self, other):
        if isinstance(other, MyIntegral):
            return do_my_adding_stuff(self, other)
        elif isinstance(other, OtherTypeIKnowAbout):
            return do_my_other_adding_stuff(self, other)
        else:
            return NotImplemented

    def __radd__(self, other):
        if isinstance(other, MyIntegral):
            return do_my_adding_stuff(other, self)
        elif isinstance(other, OtherTypeIKnowAbout):
            return do_my_other_adding_stuff(other, self)
        elif isinstance(other, Integral):
            return int(other) + int(self)
        elif isinstance(other, Real):
            return float(other) + float(self)
        elif isinstance(other, Complex):
            return complex(other) + complex(self)
        else:
            return NotImplemented
```

Hay 5 casos diferentes para una operación de tipo mixto en subclases de **`Complex`**. Se explicará todo el código anterior que no se refiere a `MyIntegral` y `OtherTypeIKnowAbout` como «repetitivo». `a` será una instancia de `A`, que es un subtipo de **`Complex`** (`a: A <: Complex`), y `b : B <: Complex`. Consideraré `a + b`:

1. If `A` defines an **`__add__()`** which accepts `b`, all is well.
2. If `A` falls back to the boilerplate code, and it were to return a value from **`__add__()`**, we'd miss the possibility that `B` defines a more intelligent **`__radd__()`**, so the boilerplate should return **`NotImplemented`** from **`__add__()`**. (Or `A` may not implement **`__add__()`** at all.)
3. Then `B`'s **`__radd__()`** gets a chance. If it accepts `a`, all is well.
4. Si se vuelve a caer en el código repetitivo, no hay más posibles métodos para probar, por lo que acá debería vivir la implementación predeterminada.
5. Si `B <: A`, Python probara `B.__radd__` antes que

`A.__add__`. Esto está bien, porque se implementó con conocimiento de `A`, por lo que puede manejar instancias antes de delegar un `Complex`.

If `A <: Complex` and `B <: Real` without sharing any other knowledge, then the appropriate shared operation is the one involving the built in `complex`, and both `__radd__()` s land there, so `a+b == b+a`.

Dado que la mayoría de las operaciones en un tipo determinado serán muy similares, puede ser útil definir una función auxiliar que genere las instancias *forward* y *reverse* de cualquier operador dado. Por ejemplo, `fractions.Fraction` así:

```
def _operator_fallbacks(monomorphic_operator, fallback_
    def forward(a, b):
        if isinstance(b, (int, Fraction)):
            return monomorphic_operator(a, b)
        elif isinstance(b, float):
            return fallback_operator(float(a), b)
        elif isinstance(b, complex):
            return fallback_operator(complex(a), b)
        else:
            return NotImplemented
    forward.__name__ = '__' + fallback_operator.__name__
    forward.__doc__ = monomorphic_operator.__doc__

    def reverse(b, a):
        if isinstance(a, Rational):
            # Includes ints.
            return monomorphic_operator(a, b)
        elif isinstance(a, Real):
            return fallback_operator(float(a), float(b))
        elif isinstance(a, Complex):
            return fallback_operator(complex(a), complex
        else:
            return NotImplemented
    reverse.__name__ = '__r' + fallback_operator.__name_
    reverse.__doc__ = monomorphic_operator.__doc__
    return forward, reverse

def _add(a, b):
    """a + b"""
    return Fraction(a.numerator * b.denominator +
                    b.numerator * a.denominator,
                    a.denominator * b.denominator)

__add__, __radd__ = _operator_fallbacks(_add, operator.a

# ...
```

`math` — Mathematical functions

This module provides access to common mathematical functions and constants, including those defined by the C standard.

Estas funciones no pueden ser usadas con números complejos; usa las funciones con el mismo nombre del módulo `cmath` si requieres soporte para números complejos. La distinción entre las funciones que admiten números complejos y las que no se hace debido a que la mayoría de los usuarios no quieren aprender tantas matemáticas como se requiere para comprender los números complejos. Recibir una excepción en lugar de un resultado complejo permite la detección temprana del número complejo inesperado utilizado como parámetro, de modo que el programador pueda determinar cómo y porqué se generó en primer lugar.

Este módulo proporciona las funciones descritas a continuación. Excepto cuando se indique lo contrario explícitamente, todos los valores retornados son flotantes.

Number-theoretic functions	
`comb(n, k)`	Number of ways to choose *k* items from *n* items without repetition and without order
`factorial(n)`	*n* factorial
`gcd(*integers)`	Greatest common divisor of the integer arguments
`isqrt(n)`	Integer square root of a nonnegative integer *n*
`lcm(*integers)`	Least common multiple of the integer arguments
`perm(n, k)`	Number of ways to choose *k* items from *n* items without repetition and with order
Floating point arithmetic	
`ceil(x)`	Ceiling of *x*, the smallest integer greater than or equal to *x*
`fabs(x)`	Absolute value of *x*
`floor(x)`	Floor of *x*, the largest integer less than or equal to *x*

`fma(x, y, z)`	Fused multiply-add operation: `(x * y) + z`
`fmod(x, y)`	Remainder of division `x / y`
`modf(x)`	Fractional and integer parts of *x*
`remainder(x, y)`	Remainder of *x* with respect to *y*
`trunc(x)`	Integer part of *x*
Floating point manipulation functions	
`copysign(x, y)`	Magnitude (absolute value) of *x* with the sign of *y*
`frexp(x)`	Mantissa and exponent of *x*
`isclose(a, b, rel_tol, abs_tol)`	Check if the values *a* and *b* are close to each other
`isfinite(x)`	Check if *x* is neither an infinity nor a NaN
`isinf(x)`	Check if *x* is a positive or negative infinity
`isnan(x)`	Check if *x* is a NaN (not a number)
`ldexp(x, i)`	`x * (2**i)`, inverse of function `frexp()`
`nextafter(x, y, steps)`	Floating-point value *steps* steps after *x* towards *y*
`ulp(x)`	Value of the least significant bit of *x*
Power, exponential and logarithmic functions	
`cbrt(x)`	Cube root of *x*
`exp(x)`	*e* raised to the power *x*
`exp2(x)`	2 raised to the power *x*
`expm1(x)`	*e* raised to the power *x*, minus 1
`log(x, base)`	Logarithm of *x* to the given base (*e* by default)
`log1p(x)`	Natural logarithm of $1+x$ (base *e*)
`log2(x)`	Base-2 logarithm of *x*
`log10(x)`	Base-10 logarithm of *x*
`pow(x, y)`	*x* raised to the power *y*
`sqrt(x)`	Square root of *x*
Summation and product functions	
`dist(p, q)`	Euclidean distance between two points *p* and *q* given as an iterable of coordinates
`fsum(iterable)`	Sum of values in the input *iterable*
`hypot(*coordinates)`	Euclidean norm of an iterable of coordinates
`prod(iterable, start)`	Product of elements in the input *iterable* with a *start* value
`sumprod(p, q)`	Sum of products from two iterables *p* and *q*
Angular conversion	
`degrees(x)`	Convert angle *x* from radians to degrees
`radians(x)`	Convert angle *x* from degrees to radians
Trigonometric functions	
`acos(x)`	Arc cosine of *x*
`asin(x)`	Arc sine of *x*
`atan(x)`	Arc tangent of *x*

`atan2(y, x)`	(y / x)
`cos(x)`	Cosine of *x*
`sin(x)`	Sine of *x*
`tan(x)`	Tangent of *x*
Hyperbolic functions	
`acosh(x)`	Inverse hyperbolic cosine of *x*
`asinh(x)`	Inverse hyperbolic sine of *x*
`atanh(x)`	Inverse hyperbolic tangent of *x*
`cosh(x)`	Hyperbolic cosine of *x*
`sinh(x)`	Hyperbolic sine of *x*
`tanh(x)`	Hyperbolic tangent of *x*
Special functions	
`erf(x)`	Error function at *x*
`erfc(x)`	Complementary error function at *x*
`gamma(x)`	Gamma function at *x*
`lgamma(x)`	Natural logarithm of the absolute value of the Gamma function at *x*
Constants	
`pi`	$\pi = 3.141592\ldots$
`e`	$e = 2.718281\ldots$
`tau`	$\tau = 2\pi = 6.283185\ldots$
`inf`	Positive infinity
`nan`	«Not a number» (NaN)

Number-theoretic functions

math.comb(*n*, *k*)

Retorna el número de formas posibles de elegir *k* elementos de *n*, de forma ordenada y sin repetición.

Se evalúa como `n! / (k! * (n - k)!)` cuando `k <= n` y como cero cuando `k > n`.

También llamado coeficiente binomial porque es equivalente al coeficiente del k-ésimo término en la expansión polinomial de `(1 + x)`^{n}.

Lanza una excepción **`TypeError`** si alguno de los argumentos no es un entero. Lanza una excepción

`ValueError` si alguno de los argumentos es negativo.

Added in version 3.8.

math.factorial(*n*)

Return factorial of the nonnegative integer *n*.

Distinto en la versión 3.10: Floats with integral values (like `5.0`) are no longer accepted.

math.gcd(**integers*)

Retorna el máximo común divisor de los argumentos enteros. Si cualquiera de los argumentos no es cero, entonces el valor retornado es el entero positivo más grande que divide a todos los argumentos. Si todos los argumentos son cero, entonces el valor retornado es `0`. `gcd()` sin argumentos retorna `0`.

Added in version 3.5.

Distinto en la versión 3.9: Agregado soporte para un número arbitrario de argumentos. Anteriormente sólo se soportaba dos argumentos.

math.isqrt(*n*)

Retorna la raíz cuadrada del número entero no negativo *n*. Es el resultado de aplicar la función suelo al valor exacto de la raíz cuadrada de *n*, o de forma equivalente, el mayor entero *a* tal que $a^2 \le n$.

Para algunas aplicaciones, puede ser más conveniente tener el menor número entero *a* tal que $n \le a^2$, en otras palabras, el resultado de aplicar la función techo a la raíz cuadrada exacta de *n*. Para *n* positivo, esto se puede calcular usando `a = 1 + isqrt(n - 1)`.

Added in version 3.8.

math.lcm(**integers*)

Retorna el mínimo común múltiplo de los argumentos enteros. Si todos los argumentos no son cero, entonces el valor retornado es el entero positivo más pequeño que es un múltiplo de todos los argumentos. Si cualquiera de los argumentos es cero, entonces el valor retornado es `0`. `lcm()` sin argumentos retorna `1`.

Added in version 3.9.

math.perm(*n*, *k=None*)

Retorna el número de formas posibles de elegir *k* elementos de *n* elementos, sin repetición y en orden.

Se evalúa como `n! / (n - k)!` cuando `k <= n` y como cero cuando `k > n`.

If *k* is not specified or is `None`, then *k* defaults to *n* and the function returns `n!`.

Lanza una excepción **`TypeError`** si alguno de los argumentos no es un entero. Lanza una excepción **`ValueError`** si alguno de los argumentos es negativo.

Added in version 3.8.

Floating point arithmetic

math.ceil(*x*)

Retorna el «techo» de *x*, el número entero más pequeño que es mayor o igual que *x*. Si *x* no es un flotante, delega en **`x.__ceil__`**, que debería retornar un valor **`Integral`**.

math.fabs(*x*)

Retorna el valor absoluto de *x*.

math.floor(*x*)

Retorna el «suelo» de *x*, el primer número entero menor o igual que *x*. Si *x* no es un flotante, delega en **`x.__floor__`**, que debería retornar un valor **`Integral`**.

math.fma(*x*, *y*, *z*)

Fused multiply-add operation. Return `(x * y) + z`, computed as though with infinite precision and range followed by a single round to the `float` format. This operation often provides better accuracy than the direct expression `(x * y) + z`.

This function follows the specification of the fusedMultiplyAdd operation described in the IEEE 754 standard. The standard leaves one case implementation-defined, namely the result of `fma(0, inf, nan)` and `fma(inf, 0, nan)`. In these cases, `math.fma` returns a NaN, and does not raise any exception.

Added in version 3.13.

math.fmod(*x*, *y*)

Return the floating-point remainder of `x / y`, as defined by the platform C library function `fmod(x, y)`. Note that the Python expression `x % y` may not return the same result. The intent of the C standard is that `fmod(x, y)` be exactly (mathematically; to infinite precision) equal to `x - n*y` for some integer *n* such that the result has the same sign as *x* and magnitude less than `abs(y)`. Python's `x % y` returns a result with the sign of *y* instead, and may not be exactly computable for float arguments. For example, `fmod(-1e-100, 1e100)` is `-1e-100`, but the result of Python's `-1e-100 % 1e100` is `1e100-1e-100`, which cannot be represented exactly as a float, and rounds to the surprising `1e100`. For this reason, function **`fmod()`** is generally preferred when working with floats, while Python's `x % y` is preferred when working with integers.

math.modf(*x*)

Retorna la parte fraccionaria y entera de *x*. Ambos resultados son flotantes y tienen el mismo signo que *x* .

Note that **`modf()`** has a different call/return pattern than its C equivalents: it takes a single argument and return a pair of

values, rather than returning its second return value through an “output parameter” (there is no such thing in Python).

math.remainder(*x*, *y*)

Retorna el resto o residuo según la norma IEEE 754 de *x* con respecto a *y*. Para un valor *x* finito y un valor *y* finito distinto de cero, es la diferencia `x - n * y`, donde `n` es el número entero más cercano al valor exacto del cociente `x / y`. Si `x / y` está exactamente en mitad de dos enteros consecutivos, el entero *par* más cercano se utiliza para `n`. Por lo tanto, el residuo `r = remainder(x, y)` siempre satisface `abs(r) <= 0.5 * abs(y)`.

Los casos especiales siguen el estándar IEEE 754: en particular, `remainder(x, math.inf)` es *x* para todo *x* finito, y `remainder(x, 0)` junto a `remainder(math.inf, x)` lanzan una excepción **`ValueError`** para todo *x* que no sea NaN. Si el resultado de la operación residuo es cero, este cero tendrá el mismo signo que *x*.

On platforms using IEEE 754 binary floating point, the result of this operation is always exactly representable: no rounding error is introduced.

Added in version 3.7.

math.trunc(*x*)

Retorna *x* con la parte fraccionaria eliminada, dejando la parte entera.Esto redondea hacia 0: `trunc()` es equivalente a **`floor()`** para *x* positivo y equivalente a **`ceil()`** para *x* negativo. Si *x* no es un flotante, delega a **`x.__trunc__`**, que debería retornar un valor **`Integral`**.

Para las funciones **`ceil()`**, **`floor()`** y **`modf()`**, ten en cuenta que *todos* los números de coma flotante de magnitud suficientemente grande son enteros exactos. Los flotantes de Python normalmente no tienen más de 53 bits de precisión (lo mismo que el tipo double de C en la plataforma), en cuyo caso cualquier flotante *x* con `abs(x) >= 2**52` no necesariamente tiene bits fraccionarios.

Floating point manipulation functions

math.copysign(*x*, *y*)

Retorna un flotante con la magnitud (valor absoluto) de *x* pero el signo de *y*. En plataformas que admiten ceros con signo, `copysign(1.0, -0.0)` retorna *-1.0*.

math.frexp(*x*)

Retorna la mantisa y el exponente de *x* como el par `(m, e)`. *m* es un flotante y *e* es un número entero tal que `x == m * 2**e` exactamente. Si *x* es cero, retorna `(0.0, 0)`, y retorna `0.5 <= abs(m) < 1` en caso contrario. Se utiliza como una forma portable de «extraer» la representación interna de un flotante.

Note that **`frexp()`** has a different call/return pattern than its C equivalents: it takes a single argument and return a pair of values, rather than returning its second return value through an "output parameter" (there is no such thing in Python).

math.isclose(*a, b, *, rel_tol=1e-09, abs_tol=0.0*)

Retorna `True` si los valores *a* y *b* están cerca el uno del otro y `False` en caso contrario.

Whether or not two values are considered close is determined according to given absolute and relative tolerances. If no errors occur, the result will be: `abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)`.

rel_tol is the relative tolerance – it is the maximum allowed difference between *a* and *b*, relative to the larger absolute value of *a* or *b*. For example, to set a tolerance of 5%, pass `rel_tol=0.05`. The default tolerance is `1e-09`, which assures that the two values are the same within about 9 decimal digits. *rel_tol* must be nonnegative and less than `1.0`.

abs_tol is the absolute tolerance; it defaults to `0.0` and it must be nonnegative. When comparing `x` to `0.0`, `isclose(x, 0)` is computed as `abs(x) <= rel_tol * abs(x)`, which is `False` for any nonzero `x` and *rel_tol* less than `1.0`. So add an appropriate positive *abs_tol* argument to the call.

Los valores especiales de IEEE 754 `NaN`, `inf` e `-inf` se manejarán de acuerdo con las reglas del IEEE. Concretamente, `NaN` no se considera cercano a ningún otro valor, incluido `NaN`. Por su parte, `inf` e `-inf` solo se consideran cercanos a sí mismos.

Added in version 3.5.

Ver también

PEP 485 [https://peps.python.org/pep-0485/] – Una función para comprobar la igualdad aproximada

math.isfinite(*x*)

Retorna `True` si *x* no es infinito ni NaN, o `False` en caso contrario. (Ten en cuenta que `0.0` *es* considerado finito.)

Added in version 3.2.

math.isinf(*x*)

Retorna `True` si *x* es infinito positivo o negativo, o `False` en caso contrario.

math.isnan(*x*)

Retorna `True` si *x* es NaN (not a number, en español: no es un número), o `False` en caso contrario.

math.ldexp(*x*, *i*)

Retorna `x * (2**i)`. Esta es esencialmente la función inversa de **`frexp()`**.

math.nextafter(*x*, *y*, *steps=1*)

Retorna el valor en coma flotante *steps* pasos después *x* hacia *y*.

Si *x* es igual a *y*, retorna *y*, a menos que *steps* sea cero.

Ejemplos:

- `math.nextafter(x, math.inf)` va hacia el infinito positivo.
- `math.nextafter(x, -math.inf)` va hacia el infinito negativo.
- `math.nextafter(x, 0.0)` va hacia cero.
- `math.nextafter(x, math.copysign(math.inf, x))` se aleja de cero.

Ver también **`math.ulp()`**.

Added in version 3.9.

Distinto en la versión 3.12: Añadido el argumento *steps*.

math.ulp(*x*)

Retorna el valor del bit menos significativo del flotante *x*:

- Si *x* es un NaN (*not a number*), retorna *x*.
- Si *x* es negativo, retorna `ulp(-x)`.
- Si *x* es un infinito positivo, retorna *x*.
- Si *x* es igual a cero, retorna el flotante representable *desnormalizado* positivo más pequeño (menor que el flotante *normalizado* positivo mínimo, **`sys.float_info.min`**).
- Si *x* es igual al flotante representable positivo más pequeño, retorna el bit menos significativo de *x*, de tal manera que el primer flotante menor que *x* es `x - ulp(x)`.
- De lo contrario (*x* es un número finito positivo), retorna

el valor del bit menos significativo de *x* , de tal forma que el primer flotante mayor a *x* es `x + ulp(x)`.

ULP significa *Unit in the Last Place* (unidad en el último lugar).

Ver también **`math.nextafter()`** y **`sys.float_info.epsilon`**.

Added in version 3.9.

Power, exponential and logarithmic functions

math.cbrt(*x*)

Retorna la raíz cúbica de *x*.

Added in version 3.11.

math.exp(*x*)

Retorna *e* elevado a la *x* potencia, dónde *e* = 2.718281… es la base de los logaritmos naturales. Esto generalmente es más preciso que `math.e ** x` o `pow(math.e, x)`.

math.exp2(*x*)

Retorna *2* elevado a la potencia *x*.

Added in version 3.11.

math.expm1(*x*)

Retorna *e* elevado a la *x* potencia, menos 1. Aquí *e* es la base de los logaritmos naturales. Para flotantes *x* pequeños, la resta en `exp(x) - 1` puede resultar en una pérdida significativa de precisión [https://en.wikipedia.org/wiki/Loss_of_significance]; la función **`expm1()`** proporciona una forma de calcular este valor con una precisión total:

```
>>> from math import exp, expm1
>>> exp(1e-5) - 1  # gives result accurate to 11 pl
1.0000050000069649e-05
>>> expm1(1e-5)    # result accurate to full precis
1.0000050000166668e-05
```

Added in version 3.2.

math.log(*x*[, *base*])

Con un argumento, retorna el logaritmo natural de *x* (en base *e*).

Con dos argumentos, retorna el logaritmo de *x* en la *base* dada, calculado como `log(x)/log(base)`.

math.log1p(*x*)

Retorna el logaritmo natural de *1+x* (base *e*). El resultado se calcula de forma precisa para *x* cercano a cero.

math.log2(*x*)

Retorna el logaritmo en base 2 de *x*. Esto suele ser más preciso que `log(x, 2)`.

Added in version 3.3.

Ver también

`int.bit_length()` retorna el número de bits necesarios para representar un entero en binario, excluyendo el signo y los ceros iniciales.

math.log10(*x*)

Retorna el logaritmo en base 10 de *x*. Esto suele ser más preciso que `log(x, 10)`.

math.pow(*x*, *y*)

Return *x* raised to the power *y*. Exceptional cases follow the

IEEE 754 standard as far as possible. In particular, `pow(1.0, x)` and `pow(x, 0.0)` always return `1.0`, even when *x* is a zero or a NaN. If both *x* and *y* are finite, *x* is negative, and *y* is not an integer then `pow(x, y)` is undefined, and raises **`ValueError`**.

A diferencia del operador incorporado `**`, **`math.pow()`** convierte ambos argumentos al tipo **`float`**. Utiliza `**` o la función incorporada **`pow()`** para calcular potencias enteras exactas.

Distinto en la versión 3.11: Los casos especiales `pow(0.0, -inf)` y `pow(-0.0, -inf)` se cambiaron para devolver `inf` en lugar de generar **`ValueError`**, por consistencia con IEEE 754.

math.sqrt(*x*)

Retorna la raíz cuadrada de *x*.

Summation and product functions

math.dist(*p*, *q*)

Retorna la distancia euclidiana entre dos puntos *p* y *q*, cada uno de ellos dado como una secuencia (o iterable) de coordenadas. Los dos puntos deben tener la misma dimensión.

Aproximadamente equivalente a:

```
sqrt(sum((px - qx) ** 2.0 for px, qx in zip(p, q)))
```

Added in version 3.8.

math.fsum(*iterable*)

Return an accurate floating-point sum of values in the iterable. Avoids loss of precision by tracking multiple intermediate partial sums.

La precisión del algoritmo depende de las garantías aritméticas de IEEE-754 y del caso típico en el que se usa el «medio redondo a par» (half-even) como método de redondeo. En algunas compilaciones que no son de Windows, la biblioteca de C subyacente utiliza la adición de precisión extendida y, ocasionalmente, puede realizar un doble redondeo en una suma intermedia, haciendo que el bit menos significativo tome el valor incorrecto.

For further discussion and two alternative approaches, see the ASPN cookbook recipes for accurate floating-point summation [https://code.activestate.com/recipes/393090-binary-floating-point-summation-accurate-to-full-p/].

math.hypot(**coordinates*)

Retorna la norma euclidiana, `sqrt(sum(x**2 for x in coordinates))`. Esta es la longitud del vector que va desde el origen hasta el punto dado por las coordenadas.

Para un punto bidimensional `(x, y)`, esto equivale a calcular la hipotenusa de un triángulo rectángulo usando el teorema de Pitágoras, `sqrt(x*x + y*y)`.

Distinto en la versión 3.8: Agregado soporte para puntos n-dimensionales. Anteriormente, solo se admitía el caso bidimensional.

Distinto en la versión 3.10: Se mejoró la precisión del algoritmo para que el error máximo sea inferior a 1 ulp (unidad en último lugar). Más típicamente, el resultado casi siempre se redondea correctamente dentro de 1/2 ulp.

math.prod(*iterable*, ***, *start=1*)

Calcula el producto de todos los elementos en la entrada *iterable*. El valor *start* predeterminado para el producto es `1`.

Cuando el iterable está vacío, retorna el valor inicial. Esta función está diseñada específicamente para su uso con valores numéricos y puede rechazar tipos no numéricos.

Added in version 3.8.

math.sumprod(*p*, *q*)

Retorna la suma de productos de valores de dos iterables *p* y *q*.

Lanza **`ValueError`** si las entradas no tienen la misma longitud.

Aproximadamente equivalente a:

```
sum(map(operator.mul, p, q, strict=True))
```

Para entradas flotantes y mixtas int/float, los productos intermedios y las sumas se calculan con precisión ampliada.

Added in version 3.12.

Conversión angular

math.degrees(*x*)

Convierte el ángulo *x* de radianes a grados.

math.radians(*x*)

Convierte el ángulo *x* de grados a radianes.

Funciones trigonométricas

math.acos(*x*)

Retorna el arcocoseno de *x*, en radianes. El resultado está entre `0` y `pi`.

math.asin(*x*)

Retorna el arcoseno de *x*, en radianes. El resultado está entre `-pi/2` y `pi/2`.

math.atan(*x*)

Retorna la arcotangente de *x*, en radianes. El resultado está entre `-pi/2` y `pi/2`.

math.atan2(*y*, *x*)

Retorna `atan(y / x)`, en radianes. El resultado está entre `-pi` y `pi`. El vector del plano que va del origen al punto `(x, y)`, forma este ángulo con el eje X positivo. La ventaja de **`atan2()`** es que el signo de ambas entradas es conocido, por lo que se puede calcular el cuadrante correcto para el ángulo. Por ejemplo, `atan(1)` y `atan2(1, 1)` son ambas `pi/4`, pero `atan2(-1, -1)` es `-3*pi/4`.

math.cos(*x*)

Retorna el coseno de *x* radianes.

math.sin(*x*)

Retorna el seno de *x* radianes.

math.tan(*x*)

Retorna la tangente de *x* radianes.

Funciones hiperbólicas

Las funciones hiperbólicas [https://es.wikipedia.org/wiki/Funci%C3%B3n_hiperb%C3%B3lica] son análogas a las funciones trigonométricas que están basadas en hipérbolas en lugar de círculos.

math.acosh(*x*)

Retorna el coseno hiperbólico inverso de *x*.

math.asinh(*x*)

Retorna el seno hiperbólico inverso de *x*.

math.atanh(*x*)

Retorna la tangente hiperbólica inversa de *x*.

math.cosh(*x*)

Retorna el coseno hiperbólico de *x*.

math.sinh(*x*)

Retorna el seno hiperbólico de *x*.

math.tanh(*x*)

Retorna la tangente hiperbólica de *x*.

Funciones especiales

math.erf(*x*)

Retorna la función error [https://es.wikipedia.org/wiki/Funci%C3%B3n_error] en *x*.

The **`erf()`** function can be used to compute traditional statistical functions such as the cumulative standard normal distribution [https://en.wikipedia.org/wiki/Cumulative_distribution_function]:

```
def phi(x):
    'Cumulative distribution function for the stand
    return (1.0 + erf(x / sqrt(2.0))) / 2.0
```

Added in version 3.2.

math.erfc(*x*)

Retorna la función error complementaria en *x*. La función error complementaria [https://es.wikipedia.org/wiki/Funci%C3%B3n_error] se define como `1.0 - erf(x)`. Se usa para valores grandes de *x* donde una resta de 1 causaría una pérdida de presición [https://en.wikipedia.org/wiki/Loss_of_significance].

Added in version 3.2.

math.gamma(*x*)

Retorna la función gamma [https://es.wikipedia.org/wiki/Funci%C3%B3n_gamma] en *x*.

Added in version 3.2.

math.lgamma(*x*)

Retorna el logaritmo natural del valor absoluto de la función gamma en *x*.

Added in version 3.2.

Constantes

math.pi

La constante matemática $\pi = 3.141592\ldots$, hasta la precisión disponible.

math.e

La constante matemática $e = 2.718281\ldots$, hasta la precisión disponible.

math.tau

The mathematical constant $\tau = 6.283185\ldots$, to available precision. Tau is a circle constant equal to 2π, the ratio of a circle's circumference to its radius. To learn more about Tau, check out Vi Hart's video Pi is (still) Wrong [https://vimeo.com/147792667], and start celebrating Tau day [https://tauday.com/] by eating twice as much pie!

Added in version 3.6.

math.inf

Un valor infinito positivo en punto flotante. (Para un valor infinito negativo, usa `-math.inf`.) Equivalente a la salida de `float('inf')`.

Added in version 3.5.

math.nan

Un valor de coma flotante «no es un número» (NaN). Equivalente a la salida de `float('nan')`. Debido a los requisitos del estándar IEEE-754 [https://es.wikipedia.org/wiki/IEEE_754], `math.nan` y `float('nan')` no se consideran iguales a ningún otro valor numérico, incluidos ellos mismos. Para verificar si un número es NaN, use la función **`isnan()`** para probar NaN en lugar de `is` o `==`. Ejemplo:

```
>>> import math
>>> math.nan == math.nan
False
>>> float('nan') == float('nan')
False
>>> math.isnan(math.nan)
True
>>> math.isnan(float('nan'))
True
```

Added in version 3.5.

Distinto en la versión 3.11: Ahora está siempre disponible.

Detalles de implementación de CPython: The **`math`** module consists mostly of thin wrappers around the platform C math library functions. Behavior in exceptional cases follows Annex F of the C99 standard where appropriate. The current implementation will raise **`ValueError`** for invalid operations like `sqrt(-1.0)` or `log(0.0)` (where C99 Annex F recommends signaling invalid operation or divide-by-zero), and **`OverflowError`** for results that overflow (for example, `exp(1000.0)`). A NaN will not be returned from any of the functions above unless one or more of the input arguments was a NaN; in that case, most functions will return a NaN, but (again following C99 Annex F) there are some exceptions to this rule, for example `pow(float('nan'), 0.0)` or `hypot(float('nan'), float('inf'))`.

Ten en cuenta que Python no hace ningún esfuerzo por distinguir los NaN de señalización de los NaN silenciosos, y el comportamiento de señalización de los NaN permanece sin especificar. El comportamiento estándar es tratar a todos los NaN como silenciosos.

Ver también

Módulo `cmath`
    Versiones de muchas de estas funciones para números complejos.

`cmath` — Mathematical functions for complex numbers

This module provides access to mathematical functions for complex numbers. The functions in this module accept integers, floating-point numbers or complex numbers as arguments. They will also accept any Python object that has either a `__complex__()` or a `__float__()` method: these methods are used to convert the object to a complex or floating-point number, respectively, and the function is then applied to the result of the conversion.

Nota

For functions involving branch cuts, we have the problem of deciding how to define those functions on the cut itself. Following Kahan's «Branch cuts for complex elementary functions» paper, as well as Annex G of C99 and later C standards, we use the sign of zero to distinguish one side of the branch cut from the other: for a branch cut along (a portion of) the real axis we look at the sign of the imaginary part, while for a branch cut along the imaginary axis we look at the sign of the real part.

For example, the `cmath.sqrt()` function has a branch cut along the negative real axis. An argument of `-2-0j` is treated as though it lies *below* the branch cut, and so gives a result on the negative imaginary axis:

```
>>> cmath.sqrt(-2-0j)
-1.4142135623730951j
```

But an argument of `-2+0j` is treated as though it lies above the branch cut:

```
>>> cmath.sqrt(-2+0j)
1.4142135623730951j
```

Conversions to and from polar coordinates	
`phase(z)`	Return the phase of *z*
`polar(z)`	Return the representation of *z* in polar coordinates
`rect(r, phi)`	Return the complex number *z* with polar coordinates *r* and *phi*
Power and logarithmic functions	
`exp(z)`	Return *e* raised to the power *z*
`log(z[, base])`	Return the logarithm of *z* to the given *base* (*e* by default)
`log10(z)`	Return the base-10 logarithm of *z*
`sqrt(z)`	Return the square root of *z*
Trigonometric functions	
`acos(z)`	Return the arc cosine of *z*
`asin(z)`	Return the arc sine of *z*
`atan(z)`	Return the arc tangent of *z*
`cos(z)`	Return the cosine of *z*
`sin(z)`	Return the sine of *z*
`tan(z)`	Return the tangent of *z*
Hyperbolic functions	
`acosh(z)`	Return the inverse hyperbolic cosine of *z*
`asinh(z)`	Return the inverse hyperbolic sine of *z*
`atanh(z)`	Return the inverse hyperbolic tangent of *z*
`cosh(z)`	Return the hyperbolic cosine of *z*
`sinh(z)`	Return the hyperbolic sine of *z*
`tanh(z)`	Return the hyperbolic tangent of *z*
Classification functions	
`isfinite(z)`	Check if all components of *z* are finite
`isinf(z)`	Check if any component of *z* is infinite
`isnan(z)`	Check if any component of *z* is a NaN
`isclose(a, b, *, rel_tol, abs_tol)`	Check if the values *a* and *b* are close to each other
Constants	
`pi`	$\pi = 3.141592\ldots$
`e`	$e = 2.718281\ldots$
`tau`	$\tau = 2\pi = 6.283185\ldots$
`inf`	Positive infinity
`infj`	Pure imaginary infinity
`nan`	«Not a number» (NaN)
`nanj`	Pure imaginary NaN

Conversión a y desde coordenadas polares

A Python complex number `z` is stored internally using *rectangular* or *Cartesian* coordinates. It is completely determined by its *real part* `z.real` and its *imaginary part* `z.imag`.

Las coordenadas polares dan una alternativa a la representación de números complejos. En las coordenadas polares, un número complejo *z* se define por los módulos *r* y el ángulo de fase *phi*. El módulo *r* es la distancia desde *z* hasta el origen, mientras que la fase *phi* es el ángulo que va en contra de las agujas del reloj, medido en radianes, desde el eje positivo de las X hasta el segmento de linea que une el origen con *z*.

Las siguientes funciones pueden ser usadas para convertir desde coordenadas rectangulares nativas hasta coordenadas polares y viceversa.

cmath.phase(*z*)

> Return the phase of *z* (also known as the *argument* of *z*), as a float. `phase(z)` is equivalent to `math.atan2(z.imag, z.real)`. The result lies in the range [-π, π], and the branch cut for this operation lies along the negative real axis. The sign of the result is the same as the sign of `z.imag`, even when `z.imag` is zero:
>
> ```
> >>> phase(-1+0j)
> 3.141592653589793
> >>> phase(-1-0j)
> -3.141592653589793
> ```

Nota

The modulus (absolute value) of a complex number *z* can be computed using the built-in `abs()` function. There is no separate **`cmath`** module function for this operation.

cmath.polar(*z*)

Return the representation of *z* in polar coordinates. Returns a pair `(r, phi)` where *r* is the modulus of *z* and *phi* is the phase of *z*. `polar(z)` is equivalent to `(abs(z), phase(z))`.

cmath.rect(*r*, *phi*)

Return the complex number *z* with polar coordinates *r* and *phi*. Equivalent to `complex(r * math.cos(phi), r * math.sin(phi))`.

Funciones logarítmicas y de potencias

cmath.exp(*z*)

Return *e* raised to the power *z*, where *e* is the base of natural logarithms.

cmath.log(*z*[, *base*])

Return the logarithm of *z* to the given *base*. If the *base* is not specified, returns the natural logarithm of *z*. There is one branch cut, from 0 along the negative real axis to `-∞`.

cmath.log10(*z*)

Return the base-10 logarithm of *z*. This has the same branch cut as **`log()`**.

cmath.sqrt(*z*)

Return the square root of *z*. This has the same branch cut as **`log()`**.

Funciones trigonométricas

cmath.acos(*z*)

Return the arc cosine of *z*. There are two branch cuts: One extends right from 1 along the real axis to `∞`. The other extends left from -1 along the real axis to `-∞`.

cmath.asin(*z*)

Return the arc sine of *z*. This has the same branch cuts as **`acos()`**.

cmath.atan(*z*)

Return the arc tangent of *z*. There are two branch cuts: One extends from `1j` along the imaginary axis to `∞j`. The other extends from `-1j` along the imaginary axis to `-∞j`.

cmath.cos(*z*)

Return the cosine of *z*.

cmath.sin(*z*)

Return the sine of *z*.

cmath.tan(*z*)

Return the tangent of *z*.

Funciones hiperbólicas

cmath.acosh(*z*)

Return the inverse hyperbolic cosine of *z*. There is one branch cut, extending left from 1 along the real axis to `-∞`.

cmath.asinh(*z*)

Return the inverse hyperbolic sine of *z*. There are two branch cuts: One extends from `1j` along the imaginary axis to `∞j`. The other extends from `-1j` along the imaginary axis to `-∞j`.

cmath.atanh(*z*)

Return the inverse hyperbolic tangent of *z*. There are two branch cuts: One extends from `1` along the real axis to `∞`. The other extends from `-1` along the real axis to `-∞`.

cmath.cosh(*z*)

Return the hyperbolic cosine of *z*.

cmath.sinh(*z*)

Return the hyperbolic sine of *z*.

cmath.tanh(*z*)

Return the hyperbolic tangent of *z*.

Funciones de clasificación

cmath.isfinite(*z*)

Return `True` if both the real and imaginary parts of *z* are finite, and `False` otherwise.

Added in version 3.2.

cmath.isinf(*z*)

Return `True` if either the real or the imaginary part of *z* is an infinity, and `False` otherwise.

cmath.isnan(*z*)

Return `True` if either the real or the imaginary part of *z* is a NaN, and `False` otherwise.

cmath.isclose(*a, b, *, rel_tol=1e-09, abs_tol=0.0*)

Retorna `True` si los valores *a* y *b* son cercanos el uno al otro y `Falso` de otro modo.

Whether or not two values are considered close is determined according to given absolute and relative tolerances. If no errors occur, the result will be: `abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)`.

rel_tol is the relative tolerance – it is the maximum allowed difference between *a* and *b*, relative to the larger absolute value of *a* or *b*. For example, to set a tolerance of 5%, pass `rel_tol=0.05`. The default tolerance is `1e-09`, which assures that the two values are the same within about 9 decimal digits. *rel_tol* must be nonnegative and less than `1.0`.

abs_tol is the absolute tolerance; it defaults to `0.0` and it must be nonnegative. When comparing `x` to `0.0`, `isclose(x, 0)` is computed as `abs(x) <= rel_tol * abs(x)`, which is `False` for any `x` and rel_tol less than `1.0`. So add an appropriate positive abs_tol argument to the call.

Los valores especiales IEEE 754 de `NaN`, `inf` y `-inf` serán manejados de acuerdo al estándar de IEEE. Especialmente, `NaN` no se considera cercano a ningún otro valor, incluido `NaN`. `inf` y `-inf` solo son considerados cercanos a sí mismos.

Added in version 3.5.

Ver también

PEP 485 [https://peps.python.org/pep-0485/] – Una función para probar igualdad aproximada.

Constantes

cmath.pi

La constante matemática π, como número de coma flotante.

cmath.e

La constante matemática *e*, como número de coma flotante.

cmath.tau

La constante matemática τ, como número de coma flotante.

Added in version 3.6.

cmath.inf

Números de coma flotante de +infinito. Equivalente a `float('inf')`.

Added in version 3.6.

cmath.infj

Números complejos con la parte real cero y números positivos infinitos como la parte imaginaria. Equivalente a `complex(0.0, float('inf'))`.

Added in version 3.6.

cmath.nan

A floating-point «not a number» (NaN) value. Equivalent to `float('nan')`. See also **`math.nan`**.

Added in version 3.6.

cmath.nanj

Números complejos con parte real cero y como parte imaginaria NaN. Equivalente a `complex(0.0, float('nan'))`.

Added in version 3.6.

Note that the selection of functions is similar, but not identical, to that in module **`math`**. The reason for having two modules is that some users aren't interested in complex numbers, and perhaps don't even know what they are. They would rather have `math.sqrt(-1)` raise an exception than return a complex number. Also note that the functions defined in **`cmath`** always return a complex number, even if the answer can be expressed as a real number (in which case the complex number has an imaginary part of zero).

Un apunte en los tramos: Se tratan de curvas en las cuales las funciones fallan a ser continua. Son un complemento necesario de muchas funciones complejas. Se asume que si se necesitan cálculos con funciones complejas, usted entenderá sobre tramos. Consulte casi cualquier(no muy elemental) libro sobre variables complejas para saber más. Para más información en la correcta elección de los tramos para propósitos numéricos, se recomienda la siguiente bibliografía:

Ver también

Kahan, W: Branch cuts for complex elementary functions; o, Much ado about nothing's sign bit. En Iserles, A., and Powell, M. (eds.), The state of the art in numerical analysis. Clarendon Press (1987) pp165–211.

[1] [2]

`decimal` — Decimal fixed-point and floating-point arithmetic

Código fuente: Lib/decimal.py [https://github.com/python/cpython/tree/3.14/Lib/decimal.py]

The `decimal` module provides support for fast correctly rounded decimal floating-point arithmetic. It offers several advantages over the `float` datatype:

- Decimal «is based on a floating-point model [https://speleotrove.com/decimal/damodel.html#refnumber] which was designed with people in mind, and necessarily has a paramount guiding principle – computers must provide an arithmetic that works in the same way as the arithmetic that people learn at school.» – excerpt from the decimal arithmetic specification.

- Los números decimales se pueden representar de forma exacta en coma flotante decimal. En cambio, números como `1.1` y `2.2` no tienen representaciones exactas en coma flotante binaria. Los usuarios finales normalmente no esperaran que `1.1 + 2.2` se muestre como `3.3000000000000003`, como ocurre al usar la representación binaria en coma flotante.

- La exactitud se traslada a la aritmética. En coma flotante decimal, `0.1 + 0.1 + 0.1 - 0.3` es exactamente igual a cero. En coma flotante binaria, el resultado es `5.5511151231257827e-017`. Aunque cercanas a cero, las diferencias impiden pruebas de igualdad confiables y las diferencias pueden acumularse. Por estas razones, se recomienda el uso de decimal en aplicaciones de contabilidad con estrictas restricciones de confiabilidad.

- El módulo decimal incorpora una noción de dígitos significativos, de modo que `1.30 + 1.20` es `2.50`. El cero final se mantiene para indicar el número de dígitos significativos. Esta es la representación habitual en aplicaciones monetarias. Para la multiplicación, el método de «libro escolar» utilizado usa todas las cifras de los multiplicandos. Por ejemplo, `1.3 * 1.2` es igual a `1.56`, mientras que `1.30 * 1.20` es igual a `1.5600`.

- A diferencia del punto flotante binario basado en hardware, el módulo decimal tiene una precisión modificable por el usuario (por defecto es de 28 dígitos decimales) que puede ser tan grande como sea necesario para un problema dado:

```
>>> from decimal import *
>>> getcontext().prec = 6
>>> Decimal(1) / Decimal(7)
Decimal('0.142857')
>>> getcontext().prec = 28
>>> Decimal(1) / Decimal(7)
Decimal('0.1428571428571428571428571429')
```

- Tanto la representación en coma flotante binaria como la decimal se implementan de acuerdo a estándares publicados. Mientras que el tipo incorporado float expone solo una pequeña parte de sus capacidades, el módulo decimal expone todos los componentes requeridos del estándar. Cuando es necesario, el desarrollador tiene control total sobre el redondeo y la gestión de las señales. Esto incluye la capacidad de forzar la aritmética exacta, utilizando excepciones para bloquear cualquier operación inexacta.

- El módulo decimal fue diseñado para admitir «indiscriminadamente, tanto aritmética decimal exacta sin redondeo (a veces llamada aritmética de coma fija) como la aritmética de coma flotante con redondeo.» – extracto (traducido) de la especificación de la aritmética decimal.

El módulo está diseñado en torno a tres conceptos: el número decimal, el contexto aritmético y las señales.

Un número decimal es inmutable. Tiene un signo, un coeficiente y un exponente. Para conservar el número de dígitos significativos, los ceros iniciales no se truncan. Los números decimales también incluyen valores especiales como `Infinity`, `-Infinity` y `NaN`. El estándar también marca la diferencia entre `-0` y `+0`.

El contexto aritmético es un entorno que permite especificar una precisión, reglas de redondeo, límites en los exponentes, flags que indican el resultado de las operaciones y habilitadores de trampas que especifican si las señales (reportadas durante operaciones ilegales) son tratadas o no como excepciones de Python. Las opciones de redondeo incluyen **`ROUND_CEILING`**, **`ROUND_DOWN`**, **`ROUND_FLOOR`**, **`ROUND_HALF_DOWN`**, **`ROUND_HALF_EVEN`**, **`ROUND_HALF_UP`**, **`ROUND_UP`** y **`ROUND_05UP`**.

Las señales son grupos de condiciones excepcionales que ocurren durante el cálculo. Dependiendo de las necesidades de la aplicación, las señales pueden ignorarse, tratarse como información o tratarse como excepciones. Las señales existentes en el módulo decimal son **`Clamped`**, **`InvalidOperation`**, **`DivisionByZero`**, **`Inexact`**, **`Rounded`**, **`Subnormal`**, **`Overflow`**, **`Underflow`** y **`FloatOperation`**.

Por cada señal hay un flag y un habilitador de trampa. Cuando ocurre una operación ilegal, su flag se establece en uno, luego, si su habilitador de trampa está establecido en uno, se lanza una excepción. La configuración de los flags es persistente, por lo que el usuario debe restablecerlos antes de comenzar un cálculo que desee monitorear.

Ver también

- Especificación general de la aritmética decimal de IBM, The General Decimal Arithmetic Specification [https://speleotrove.com/decimal/decarith.html].

Quick-start tutorial

El punto de partida habitual para usar decimales es importar el módulo, ver el contexto actual con **`getcontext()`** y, si es necesario, establecer nuevos valores para la precisión, el redondeo o trampas de señales habilitadas:

```
>>> from decimal import *
>>> getcontext()
Context(prec=28, rounding=ROUND_HALF_EVEN, Emin=-999999,
        capitals=1, clamp=0, flags=[], traps=[Overflow,
        InvalidOperation])

>>> getcontext().prec = 7       # Set a new precision
```

Las instancias de la clase Decimal se pueden construir a partir de números enteros, cadenas de caracteres, flotantes o tuplas. La construcción a partir de un número entero o flotante realiza una conversión exacta del valor de ese número. Los números decimales incluyen valores especiales como `NaN`, que significa «No es un número», `Infinity` positivo y negativo o `-0`:

```
>>> getcontext().prec = 28
>>> Decimal(10)
Decimal('10')
>>> Decimal('3.14')
Decimal('3.14')
>>> Decimal(3.14)
Decimal('3.140000000000000124344978758017532527446746826
>>> Decimal((0, (3, 1, 4), -2))
Decimal('3.14')
>>> Decimal(str(2.0 ** 0.5))
Decimal('1.4142135623730951')
>>> Decimal(2) ** Decimal('0.5')
Decimal('1.414213562373095048801688724')
>>> Decimal('NaN')
Decimal('NaN')
>>> Decimal('-Infinity')
Decimal('-Infinity')
```

Si la señal **`FloatOperation`** es atrapada, la mezcla accidental de decimales y flotantes en constructores o comparaciones de orden lanzará una excepción:

```
>>> c = getcontext()
>>> c.traps[FloatOperation] = True
>>> Decimal(3.14)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
decimal.FloatOperation: [<class 'decimal.FloatOperation'
>>> Decimal('3.5') < 3.7
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
decimal.FloatOperation: [<class 'decimal.FloatOperation'
>>> Decimal('3.5') == 3.5
True
```

Added in version 3.3.

La significación de un nuevo objeto Decimal es determinada únicamente por el número de dígitos ingresados. La precisión y el redondeo establecidos en el contexto solo entran en juego durante las operaciones aritméticas.

```
>>> getcontext().prec = 6
>>> Decimal('3.0')
Decimal('3.0')
>>> Decimal('3.1415926535')
Decimal('3.1415926535')
>>> Decimal('3.1415926535') + Decimal('2.7182818285')
Decimal('5.85987')
>>> getcontext().rounding = ROUND_UP
>>> Decimal('3.1415926535') + Decimal('2.7182818285')
Decimal('5.85988')
```

Se lanza una excepción **`InvalidOperation`** si durante la construcción de un objeto Decimal se exceden los límites internos de la versión de C:

```
>>> Decimal("1e9999999999999999999")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
decimal.InvalidOperation: [<class 'decimal.InvalidOperat
```

Distinto en la versión 3.3.

Decimals interact well with much of the rest of Python. Here is a small decimal floating-point flying circus:

```
>>> data = list(map(Decimal, '1.34 1.87 3.45 2.35 1.00 0
>>> max(data)
Decimal('9.25')
>>> min(data)
Decimal('0.03')
>>> sorted(data)
[Decimal('0.03'), Decimal('1.00'), Decimal('1.34'), Deci
 Decimal('2.35'), Decimal('3.45'), Decimal('9.25')]
>>> sum(data)
Decimal('19.29')
>>> a,b,c = data[:3]
>>> str(a)
'1.34'
>>> float(a)
1.34
>>> round(a, 1)
Decimal('1.3')
>>> int(a)
1
>>> a * 5
Decimal('6.70')
>>> a * b
Decimal('2.5058')
>>> c % a
Decimal('0.77')
```

Decimals can be formatted (with **`format()`** built-in or f-strings) in fixed-point or scientific notation, using the same formatting syntax (see Format specification mini-language) as builtin **`float`** type:

```
>>> format(Decimal('2.675'), "f")
'2.675'
>>> format(Decimal('2.675'), ".2f")
'2.68'
```

```
>>> f"{Decimal('2.675'):.2f}"
'2.68'
>>> format(Decimal('2.675'), ".2e")
'2.68e+0'
>>> with localcontext() as ctx:
...     ctx.rounding = ROUND_DOWN
...     print(format(Decimal('2.675'), ".2f"))
...
2.67
```

Y algunas funciones matemáticas también están disponibles para Decimal:

```
>>> getcontext().prec = 28
>>> Decimal(2).sqrt()
Decimal('1.414213562373095048801688724')
>>> Decimal(1).exp()
Decimal('2.718281828459045235360287471')
>>> Decimal('10').ln()
Decimal('2.302585092994045684017991455')
>>> Decimal('10').log10()
Decimal('1')
```

El método **`quantize()`** redondea un número a un exponente fijo. Este método es útil para aplicaciones monetarias, que a menudo redondean los resultados a un número fijo de dígitos significativos:

```
>>> Decimal('7.325').quantize(Decimal('.01'), rounding=R
Decimal('7.32')
>>> Decimal('7.325').quantize(Decimal('1.'), rounding=RO
Decimal('8')
```

Como se muestra arriba, la función **`getcontext()`** accede al contexto actual y permite cambiar la configuración. Este enfoque satisface las necesidades de la mayoría de las aplicaciones.

For more advanced work, it may be useful to create alternate contexts using the **`Context()`** constructor. To make an alternate active, use the **`setcontext()`** function.

In accordance with the standard, the **`decimal`** module provides two ready to use standard contexts, **`BasicContext`** and **`ExtendedContext`**. The former is especially useful for debugging because many of the traps are enabled:

```
>>> myothercontext = Context(prec=60, rounding=ROUND_HAL
>>> setcontext(myothercontext)
>>> Decimal(1) / Decimal(7)
Decimal('0.142857142857142857142857142857142857142857142

>>> ExtendedContext
Context(prec=9, rounding=ROUND_HALF_EVEN, Emin=-999999,
        capitals=1, clamp=0, flags=[], traps=[])
>>> setcontext(ExtendedContext)
>>> Decimal(1) / Decimal(7)
Decimal('0.142857143')
>>> Decimal(42) / Decimal(0)
Decimal('Infinity')

>>> setcontext(BasicContext)
>>> Decimal(42) / Decimal(0)
Traceback (most recent call last):
  File "<pyshell#143>", line 1, in -toplevel-
    Decimal(42) / Decimal(0)
DivisionByZero: x / 0
```

Los objetos Context también tienen flags de señalización para detectar condiciones excepcionales detectadas durante los cálculos. Estos flags permanecen habilitados hasta que se restablecen explícitamente. Por esta razón, suele ser buena idea restablecerlos mediante el método **`clear_flags()`** antes de proceder con cada conjunto de cálculos monitorizados.

```
>>> setcontext(ExtendedContext)
>>> getcontext().clear_flags()
>>> Decimal(355) / Decimal(113)
Decimal('3.14159292')
>>> getcontext()
Context(prec=9, rounding=ROUND_HALF_EVEN, Emin=-999999,
        capitals=1, clamp=0, flags=[Inexact, Rounded], t
```

La entrada *flags* muestra que la aproximación racional de Pi fue redondeada (los dígitos más allá de la precisión especificada por el contexto se descartaron) y que el resultado es inexacto (algunos de los dígitos descartados no eran cero).

Las trampas de señales se habilitan a través del diccionario expuesto por el atributo `traps` del objeto Context:

```
>>> setcontext(ExtendedContext)
>>> Decimal(1) / Decimal(0)
Decimal('Infinity')
>>> getcontext().traps[DivisionByZero] = 1
>>> Decimal(1) / Decimal(0)
Traceback (most recent call last):
  File "<pyshell#112>", line 1, in -toplevel-
    Decimal(1) / Decimal(0)
DivisionByZero: x / 0
```

La mayoría de los programas ajustan el contexto actual una sola vez, al comienzo del programa. Y, en muchas aplicaciones, los datos se convierten a `Decimal` mediante una única conversión dentro de un bucle. Con el contexto establecido y los decimales creados, la mayor parte del programa manipula los datos de la misma forma que con otros tipos numéricos de Python.

Objetos Decimal

class decimal.Decimal(*value='0'*, *context=None*)

Construye un nuevo objeto `Decimal` basado en *value*.

value puede ser un entero, una cadena de caracteres, una tupla, un `float` u otro objeto `Decimal`. Si no se proporciona *value*, retorna `Decimal('0')`. Si *value* es una cadena, debe ajustarse a la sintaxis de cadena numérica decimal después de que los espacios en blanco iniciales y finales, así como los guiones bajos, sean eliminados:

```
sign           ::=  '+' | '-'
digit          ::=  '0' | '1' | '2' | '3' | '4' | '
indicator      ::=  'e' | 'E'
digits         ::=  digit [digit]...
decimal-part   ::=  digits '.' [digits] | ['.'] dig
exponent-part  ::=  indicator [sign] digits
infinity       ::=  'Infinity' | 'Inf'
nan            ::=  'NaN' [digits] | 'sNaN' [digits
numeric-value  ::=  decimal-part [exponent-part] |
numeric-string ::=  [sign] numeric-value | [sign] n
```

Other Unicode decimal digits are also permitted where `digit` appears above. These include decimal digits from various other alphabets (for example, Arabic-Indic and Devanāgarī digits) along with the fullwidth digits `'\uff10'` through `'\uff19'`. Case is not significant, so, for example, `inf`, `Inf`, `INFINITY`, and `iNfINity` are all acceptable spellings for positive infinity.

Si *value* es un objeto **`tuple`**, debe tener tres componentes, un signo (`0` para positivo o `1` para negativo), un objeto **`tuple`** con los dígitos y un exponente entero. Por ejemplo, `Decimal((0, (1, 4, 1, 4), -3))` retorna `Decimal('1.414')`.

If *value* is a **`float`**, the binary floating-point value is losslessly converted to its exact decimal equivalent. This conversion can often require 53 or more digits of precision. For example, `Decimal(float('1.1'))` converts to `Decimal('1.1000000000000000888178419700125232338905`

La precisión de *context* no afecta a la cantidad de dígitos almacenados. Eso está determinado exclusivamente por el número de dígitos en *value*. Por ejemplo, `Decimal('3.00000')` registra los cinco ceros incluso si la precisión del contexto es solo tres.

El propósito del argumento *context* es determinar qué hacer si *value* es una cadena de caracteres mal formada. Si el contexto atrapa la señal **`InvalidOperation`**, se genera una excepción; de lo contrario, el constructor retorna un nuevo decimal con el valor `NaN`.

Una vez construidos, los objetos **`Decimal`** son inmutables.

Distinto en la versión 3.2: Ahora se permite que el argumento del constructor sea una instancia **`float`**.

Distinto en la versión 3.3: Los argumentos **`float`** ahora generan una excepción si se establece la trampa **`FloatOperation`**. Por defecto, la trampa está desactivada.

Distinto en la versión 3.6: Se permiten guiones bajos para la agrupación, como ocurre en el código con los literales enteros y de punto flotante.

Decimal floating-point objects share many properties with the other built-in numeric types such as **`float`** and **`int`**. All of the usual math operations and special methods apply. Likewise, decimal objects can be copied, pickled, printed, used as dictionary keys, used as set elements, compared, sorted, and coerced to another type (such as **`float`** or **`int`**).

Hay algunas pequeñas diferencias entre la aritmética en objetos decimales y la aritmética en enteros y flotantes. Cuando el operador de resto `%` se aplica a objetos Decimal, el signo del resultado es el signo del *dividendo* en lugar del signo del divisor:

```
>>> (-7) % 4
1
>>> Decimal(-7) % Decimal(4)
Decimal('-3')
```

El operador de división entera `//` se comporta de manera análoga, retornando la parte entera del cociente verdadero (truncando hacia cero) en lugar del resultado de aplicarle la función suelo. Esto se hace con la finalidad de preservar la identidad habitual `x == (x // y) * y + x % y`:

```
>>> -7 // 4
```

```
-2
>>> Decimal(-7) // Decimal(4)
Decimal('-1')
```

Los operadores `%` y `//` implementan las operaciones `remainder` y `divide-integer` (respectivamente) como se describe en la especificación.

Los objetos de la clase Decimal generalmente no se pueden combinar con flotantes o instancias de **`fractions.Fraction`** en operaciones aritméticas: un intento de agregar un objeto **`Decimal`** a un **`float`**, por ejemplo, lanzará una excepción **`TypeError`**. Sin embargo, es posible usar los operadores de comparación de Python para comparar una instancia de **`Decimal`** `x` con otro número `y`. Esto evita resultados confusos al hacer comparaciones de igualdad entre números de diferentes tipos.

Distinto en la versión 3.2: Las comparaciones de tipo mixto entre instancias de **`Decimal`** y otros tipos numéricos ahora son totalmente compatibles.

In addition to the standard numeric properties, decimal floating-point objects also have a number of specialized methods:

adjusted()

Retorna el exponente ajustado después de desplazar los dígitos del extremo derecho del coeficiente hasta que solo quede el dígito principal: `Decimal('321e+5').adjusted()` retorna siete. Se utiliza para determinar la posición del dígito más significativo con respecto al punto decimal.

as_integer_ratio()

Retorna un par de enteros `(n, d)` que representan la instancia de **`Decimal`** proporcionada como una fracción irreducible y con un denominador positivo:

```
>>> Decimal('-3.14').as_integer_ratio()
```

```
(-157, 50)
```

La conversión es exacta. Lanza una excepción OverflowError si se proporcionan valores infinitos y ValueError con valores NaN.

Added in version 3.6.

as_tuple()

Retorna una representación en forma de named tuple del número: `DecimalTuple(sign, digits, exponent)`.

canonical()

Retorna la codificación canónica del argumento. Actualmente, la codificación de una instancia de **`Decimal`** es siempre canónica, por lo que esta operación retorna su argumento sin cambios.

compare(*other*, *context=None*)

Compara los valores de dos instancias de Decimal. El método **`compare()`** retorna una instancia de Decimal, y si alguno de los operandos es un NaN, el resultado es un NaN:

```
a or b is a NaN  ==> Decimal('NaN')
a < b            ==> Decimal('-1')
a == b           ==> Decimal('0')
a > b            ==> Decimal('1')
```

compare_signal(*other*, *context=None*)

Esta operación es idéntica al método **`compare()`**, excepto que todos los valores NaN generan una señal. Es decir, si ninguno de los operandos es un NaN señalizador, cualquier operando de NaN silencioso se trata como si fuera un NaN señalizador.

compare_total(*other*, *context=None*)

Compara dos operandos utilizando su representación abstracta en lugar de su valor numérico. Similar al método **`compare()`**, pero el resultado proporciona un ordenamiento total en las instancias de **`Decimal`**. Dos instancias de **`Decimal`** con el mismo valor numérico, pero diferentes representaciones, se comparan como desiguales usando este orden:

```
>>> Decimal('12.0').compare_total(Decimal('12')
Decimal('-1')
```

Los NaN silenciosos y señalizadores también se incluyen en el ordenamiento total. El resultado de esta función es `Decimal('0')` si ambos operandos tienen la misma representación, `Decimal('-1')` si el primer operando es menor en el orden total que el segundo y `Decimal('1')` si el primer operando es mayor en el orden total que el segundo operando. Consulta las especificaciones para obtener detalles sobre el ordenamiento total.

Esta operación no se ve afectada por el contexto y es silenciosa: no se cambian los flags y no se realiza ningún redondeo. Como excepción, la versión de C puede lanzar InvalidOperation si el segundo operando no se puede convertir exactamente.

compare_total_mag(*other*, *context*=*None*)

Compara dos operandos usando su representación abstracta en lugar de su valor, como en **`compare_total()`**, pero ignorando el signo de cada operando. `x.compare_total_mag(y)` es equivalente a `x.copy_abs().compare_total(y.copy_abs())`.

Esta operación no se ve afectada por el contexto y es silenciosa: no se cambian los flags y no se realiza ningún redondeo. Como excepción, la versión de C puede lanzar InvalidOperation si el segundo operando no se puede convertir exactamente.

conjugate()

Simplemente retorna self (el propio objeto al que pertenece el método invocado). Este método existe solo para cumplir con la Especificación decimal.

copy_abs()

Retorna el valor absoluto del argumento. Esta operación no se ve afectada por el contexto y es silenciosa: no se modifican los flags y no se realiza ningún redondeo.

copy_negate()

Retorna la negación del argumento. Esta operación no se ve afectada por el contexto y es silenciosa: no se cambian los flags y no se realiza ningún redondeo.

copy_sign(*other*, *context=None*)

Retorna una copia del primer operando pero con el signo establecido para que sea el mismo que el del segundo operando. Por ejemplo:

```
>>> Decimal('2.3').copy_sign(Decimal('-1.5'))
Decimal('-2.3')
```

Esta operación no se ve afectada por el contexto y es silenciosa: no se cambian los flags y no se realiza ningún redondeo. Como excepción, la versión de C puede lanzar InvalidOperation si el segundo operando no se puede convertir exactamente.

exp(*context=None*)

Retorna el valor de la función exponencial (natural) `e**x` en el número dado. El resultado es correctamente redondeado utilizando el modo de redondeo **`ROUND_HALF_EVEN`**.

```
>>> Decimal(1).exp()
Decimal('2.718281828459045235360287471')
```

```
>>> Decimal(321).exp()
Decimal('2.561702493119680037517373933E+139')
```

classmethod from_float(*f*, */*)

Constructor alternativo que acepta únicamente instancias de **`float`** o **`int`**.

Fíjate que `Decimal.from_float(0.1)` no es lo mismo que `Decimal('0.1')`. Dado que 0.1 no es exactamente representable en coma flotante binaria, el valor se almacena como el valor representable más cercano, que es `0x1.999999999999ap-4`. Ese valor equivalente en decimal es `0.10000000000000000555111512312578270211815834 0`

Nota

Desde Python 3.2 en adelante, una instancia de **`Decimal`** también se puede construir directamente desde una instancia de **`float`**.

```
>>> Decimal.from_float(0.1)
Decimal('0.10000000000000000555111512312578270 2
>>> Decimal.from_float(float('nan'))
Decimal('NaN')
>>> Decimal.from_float(float('inf'))
Decimal('Infinity')
>>> Decimal.from_float(float('-inf'))
Decimal('-Infinity')
```

Added in version 3.1.

classmethod from_number(*number*, */*)

Alternative constructor that only accepts instances of **`float`**, **`int`** or **`Decimal`**, but not strings or tuples.

```
>>> Decimal.from_number(314)
Decimal('314')
>>> Decimal.from_number(0.1)
```

```
Decimal('0.10000000000000000555111512312578270 2
>>> Decimal.from_number(Decimal('3.14'))
Decimal('3.14')
```

Added in version 3.14.

fma(*other*, *third*, *context=None*)

Fusión de la multiplicación y la suma. Retorna self*other+third sin redondeo del producto intermedio self*other.

```
>>> Decimal(2).fma(3, 5)
Decimal('11')
```

is_canonical()

Retorna **`True`** si el argumento es canónico y **`False`** en caso contrario. Actualmente, una instancia de **`Decimal`** es siempre canónica, por lo que esta operación siempre retorna **`True`**.

is_finite()

Retorna **`True`** si el argumento es un número finito y **`False`** si el argumento es un valor infinito o un NaN.

is_infinite()

Retorna **`True`** si el argumento es un valor infinito positivo o negativo y **`False`** en caso contrario.

is_nan()

Retorna **`True`** si el argumento es un NaN (silencioso o señalizador) y **`False`** en caso contrario.

is_normal(*context=None*)

Retorna **`True`** si el argumento es un número finito *normal*. Retorna **`False`** si el argumento es cero, subnormal, infinito o un NaN.

is_qnan()

Retorna `True` si el argumento es un NaN silencioso y `False` en caso contrario.

is_signed()

Retorna `True` si el argumento tiene signo negativo y `False` en caso contrario. Ten en cuenta que tanto los ceros como los NaN pueden tener signo.

is_snan()

Retorna `True` si el argumento es un NaN señalizador y `False` en caso contrario.

is_subnormal(*context=None*)

Retorna `True` si el argumento es subnormal y `False` en caso contrario.

is_zero()

Retorna `True` si el argumento es un cero (positivo o negativo) y `False` en caso contrario.

ln(*context=None*)

Retorna el logaritmo natural (base e) del operando. El resultado es correctamente redondeado utilizando el modo de redondeo `ROUND_HALF_EVEN`.

log10(*context=None*)

Retorna el logaritmo en base diez del operando. El resultado es correctamente redondeado utilizando el modo de redondeo `ROUND_HALF_EVEN`.

logb(*context=None*)

Para un número distinto de cero, retorna el exponente ajustado de su operando como una instancia de `Decimal`. Si el operando es cero, se retorna `Decimal('-Infinity')` y se activa el flag

`DivisionByZero`. Si el operando es infinito, se retorna `Decimal('Infinity')`.

logical_and(*other*, *context=None*)

`logic_and()` es una operación lógica que toma dos *operandos lógicos* (consultar Operandos lógicos). El resultado es el `and` dígito por dígito de los dos operandos.

logical_invert(*context=None*)

`logic_invert()` es una operación lógica. El resultado es la inversión dígito a dígito del operando.

logical_or(*other*, *context=None*)

`logical_or()` es una operación lógica que toma dos *operandos lógicos* (consultar Operandos lógicos). El resultado es un `or` dígito a dígito de los dos operandos.

logical_xor(*other*, *context=None*)

`logic_xor()` es una operación lógica que toma dos *operandos lógicos* (consultar Operandos lógicos). El resultado es la disyunción exclusiva («exclusive or») dígito a dígito de ambos operandos.

max(*other*, *context=None*)

Como `max(self, other)`, excepto que la regla de redondeo del contexto se aplica antes de retornar y que los valores `NaN` generan una señal o son ignorados (dependiendo del contexto y de si son señalizadores o silenciosos).

max_mag(*other*, *context=None*)

Similar al método **`max()`**, pero la comparación se realiza utilizando los valores absolutos de los operandos.

min(*other*, *context=None*)

Como `min(self, other)`, excepto que la regla de redondeo del contexto se aplica antes de retornar y que los valores `NaN` generan una señal o se ignoran (según el contexto y si son señalizadores o no).

min_mag(*other*, *context=None*)

Similar al método **`min()`**, pero la comparación se realiza utilizando los valores absolutos de los operandos.

next_minus(*context=None*)

Retorna el número más grande representable en el contexto proporcionado (o en el contexto del hilo actual si no se proporciona un contexto) que sea más pequeño que el operando proporcionado.

next_plus(*context=None*)

Retorna el número más pequeño representable en el contexto proporcionado (o en el contexto del hilo actual si no se proporciona ningún contexto) que sea más grande que el operando proporcionado.

next_toward(*other*, *context=None*)

Si los dos operandos no son iguales, retorna el número más cercano al primer operando en la dirección del segundo operando. Si ambos operandos son numéricamente iguales, retorna una copia del primer operando con el signo establecido para que sea el mismo que el signo del segundo operando.

normalize(*context=None*)

Usado para producir valores canónicos de una clase de equivalencia dentro del contexto actual o del contexto especificado.

Este tiene la misma semántica que el operador unario

+, excepto que si el resultado final es finito, entonces se reduce a su forma más simple, eliminando todos los ceros finales y preservando el signo. Es decir, que mientras que el coeficiente sea distinto de cero y múltiplo de 10, el coeficiente se divide por 10 y el exponente se incremente por 1. En otro caso (el coeficiente es cero), el exponente se establece en 0. En todos los casos el signo no se modifica.

Por ejemplo, `Decimal('32.100')` y `Decimal('0.321000e+2')` ambos se normalizan al valor equivalente `Decimal('32.1')`.

Fíjate que el redondeo se aplica *antes* de reducir a la forma más simple.

En las últimas versiones de la especificación, esta operación también era conocida como `reduce`.

number_class(*context*=*None*)

Retorna una cadena de caracteres que describe la *class* del operando. El valor retornado es una de las siguientes diez cadenas de caracteres.

- `"-Infinity"`, que indica que el operando es un infinito negativo.
- `"-Normal"`, que indica que el operando es un número normal negativo.
- `"-Subnormal"`, que indica que el operando es negativo y subnormal.
- `"-Zero"`, que indica que el operando es un cero negativo.
- `"+Zero"`, que indica que el operando es un cero positivo.
- `"+Subnormal"`,que indica que el operando es positivo y subnormal.
- `"+Normal"`, que indica que el operando es un número normal positivo.
- `"+Infinity"`, que indica que el operando es un infinito positivo.

- `"NaN"`, que indica que el operando es un NaN (no es un número) silencioso.
- `"sNaN"`, que indica que el operando es un NaN (no es un número) señalizador.

quantize(*exp*, *rounding=None*, *context=None*)

Retorna un valor igual al primer operando después de ser redondeado y de asignarle el exponente del segundo operando.

```
>>> Decimal('1.41421356').quantize(Decimal('1.
Decimal('1.414')
```

A diferencia de otras operaciones, se genera una señal **`InvalidOperation`** si la longitud del coeficiente después de la operación quantize es mayor que la precisión. Esto garantiza que, a menos que exista una condición de error, el exponente cuantificado sea siempre igual al del operando de la derecha.

Además, a diferencia de otras operaciones, quantize nunca genera una señal Underflow, incluso si el resultado es subnormal e inexacto.

Si el exponente del segundo operando es mayor que el del primero, puede ser necesario redondear. En este caso, el modo de redondeo está determinado por el argumento `rounding`, si se proporciona, o por el argumento `context` en caso contrario. Si no se proporciona ninguno de estos dos argumentos, se utiliza el modo de redondeo establecido en el contexto del hilo actual.

Se retorna un error siempre que el exponente resultante sea mayor que **`Emax`** o menor que **`Etiny()`**.

radix()

Retorna `Decimal(10)`, que es la raíz (base) en la que la clase **`Decimal`** hace toda su aritmética. Este método está incluido solo por compatibilidad con la especificación.

remainder_near(*other*, *context=None*)

Retorna el resto de dividir *self* entre *other*. Esto difiere de la operación `self % other`, en la que el signo del resto se elige para minimizar su valor absoluto. Más precisamente, el valor de retorno es `self - n * other`, donde `n` es el número entero más cercano al valor exacto de `self / other`. Si dos enteros están igualmente cerca, entonces el valor par es el elegido.

Si el resultado es cero, entonces su signo será el signo de *self*.

```
>>> Decimal(18).remainder_near(Decimal(10))
Decimal('-2')
>>> Decimal(25).remainder_near(Decimal(10))
Decimal('5')
>>> Decimal(35).remainder_near(Decimal(10))
Decimal('-5')
```

rotate(*other*, *context=None*)

Retorna el resultado de rotar los dígitos del primer operando en una cantidad especificada por el segundo operando. El segundo operando debe ser un número entero en el rango comprendido desde -precisión hasta precisión. El valor absoluto del segundo operando da el número de lugares a rotar. Si el segundo operando es positivo, la rotación es hacia la izquierda; de lo contrario, la rotación es hacia la derecha. El coeficiente del primer operando se rellena con ceros a la izquierda para satisfacer la precisión de longitud si es necesario. El signo y el exponente del primer operando no se modifican.

same_quantum(*other*, *context=None*)

Comprueba si self y other tienen el mismo exponente o si ambos son `NaN`.

Esta operación no se ve afectada por el contexto y es silenciosa: no se cambian los flags y no se realiza ningún redondeo. Como excepción, la versión de C puede lanzar InvalidOperation si el segundo operando no se puede convertir exactamente.

scaleb(*other*, *context=None*)

Retorna el primer operando con su exponente ajustado por el segundo. De manera equivalente, retorna el primer operando multiplicado por `10**other`. El segundo operando debe ser un número entero.

shift(*other*, *context=None*)

Retorna el resultado de cambiar los dígitos del primer operando en una cantidad especificada por el segundo operando. El segundo operando debe ser un número entero en el rango comprendido desde -precisión hasta precisión. El valor absoluto del segundo operando da el número de lugares a desplazar. Si el segundo operando es positivo, el desplazamiento es hacia la izquierda; de lo contrario, el desplazamiento es hacia la derecha. Los dígitos desplazados en el coeficiente son ceros. El signo y el exponente del primer operando no se modifican.

sqrt(*context=None*)

Retorna la raíz cuadrada del argumento con precisión total.

to_eng_string(*context=None*)

Convierte a una cadena de caracteres, usando notación de ingeniería si se necesita un exponente.

La notación de ingeniería tiene como exponente un múltiplo de 3. Esto puede dejar hasta 3 dígitos a la izquierda del punto decimal y puede requerir la adición de uno o dos ceros finales.

Por ejemplo, este método convierte `Decimal('123E`

`+1')` en `Decimal('1.23E+3')`.

to_integral(*rounding=None*, *context=None*)

Idéntico al método **`to_integral_value()`**. El nombre `to_integral` se ha mantenido por compatibilidad con versiones anteriores.

to_integral_exact(*rounding=None*, *context=None*)

Redondea al entero más cercano, generando la señal **`Inexact`** o **`Rounded`**, según corresponda, si se produce un redondeo. El modo de redondeo está determinado por el parámetro `rounding` si es proporcionado, o por el establecido en el `context` proporcionado en caso contrario. Si no se proporciona ninguno de estos dos parámetros, se utiliza el modo de redondeo establecido en el contexto actual.

to_integral_value(*rounding=None*, *context=None*)

Redondea al entero más cercano sin generar la señal **`Inexact`** o **`Rounded`**. Si se proporciona, se aplica el método de redondeo especificado por *rounding*; en caso contrario, se utiliza el método de redondeo del *context* proporcionado o el del contexto actual.

Decimal numbers can be rounded using the **`round()`** function:

round(number)

round(number, ndigits)

If *ndigits* is not given or `None`, returns the nearest **`int`** to *number*, rounding ties to even, and ignoring the rounding mode of the **`Decimal`** context. Raises **`OverflowError`** if *number* is an infinity or **`ValueError`** if it is a (quiet or signaling) NaN.

If *ndigits* is an **`int`**, the context's rounding mode is respected and a **`Decimal`** representing *number* rounded to the nearest multiple of `Decimal('1E-ndigits')` is returned; in this case, `round(number, ndigits)` is equivalent to `self.quantize(Decimal('1E-ndigits'))`. Returns `Decimal('NaN')` if *number* is a quiet NaN. Raises **`InvalidOperation`** if *number* is an infinity, a signaling NaN, or if the length of the coefficient after the quantize operation would be greater than the current context's precision. In other words, for the non-corner cases:

- if *ndigits* is positive, return *number* rounded to *ndigits* decimal places;
- if *ndigits* is zero, return *number* rounded to the nearest integer;
- if *ndigits* is negative, return *number* rounded to the nearest multiple of `10**abs(ndigits)`.

For example:

```
>>> from decimal import Decimal, getcontext, RO
>>> getcontext().rounding = ROUND_DOWN
>>> round(Decimal('3.75'))     # context roundi
4
>>> round(Decimal('3.5'))      # round-ties-to-
4
>>> round(Decimal('3.75'), 0)  # uses the conte
Decimal('3')
>>> round(Decimal('3.75'), 1)
Decimal('3.7')
>>> round(Decimal('3.75'), -1)
Decimal('0E+1')
```

Operandos lógicos

Los métodos **`logical_and()`**, **`logical_invert()`**, **`logical_or()`**, y **`logical_xor()`** esperan que sus argumentos sean *operandos lógicos*. Un *operando lógico* es una instancia de **`Decimal`** cuyo exponente y signo son ambos cero, y cuyos dígitos son todos `0` o `1`.

Objetos context

Los contextos son entornos para operaciones aritméticas. Gobiernan la precisión, establecen reglas para el redondeo, determinan qué señales se tratan como excepciones y limitan el rango para los exponentes.

Cada hilo tiene su propio contexto actual, al que se accede o se reemplaza usando las funciones `getcontext()` y `setcontext()` respectivamente:

decimal.getcontext()

Retorna el contexto actual del hilo activo.

decimal.setcontext(*c*, /)

Establece *c* como contexto actual para el hilo activo.

También puedes usar la declaración `with` y la función `localcontext()` para cambiar temporalmente el contexto activo.

decimal.localcontext(*ctx=None*, ***kwargs*)

Retorna un gestor de contexto que establecerá el contexto actual para el hilo activo en una copia de *ctx* al ingresar en la sentencia with y restaurará el contexto anterior al salir de la misma. Si no se especifica ningún contexto, se utiliza una copia del contexto actual. El argumento *kwargs* se usa para establecer los atributos del nuevo contexto.

Por ejemplo, el siguiente código establece la precisión decimal actual en 42 lugares, realiza un cálculo y luego restaura automáticamente el contexto anterior:

```
from decimal import localcontext

with localcontext() as ctx:
    ctx.prec = 42   # Perform a high precision calc
    s = calculate_something()
s = +s  # Round the final result back to the defaul
```

Usando argumentos de palabra clave, el código sería el siguiente:

```
from decimal import localcontext

with localcontext(prec=42) as ctx:
    s = calculate_something()
s = +s
```

Lanza `TypeError` si *kwargs* proporciona un atributo que `Context` no soporta. Lanza también `TypeError` o `ValueError` si *kwargs* proporciona un valor no válido para un atributo.

Distinto en la versión 3.11: `localcontext()` ahora admite la configuración de atributos de contexto mediante el uso de argumentos de palabra clave.

decimal.IEEEContext(*bits*)

Return a context object initialized to the proper values for one of the IEEE interchange formats. The argument must be a multiple of 32 and less than `IEEE_CONTEXT_MAX_BITS`.

Added in version 3.14.

También se pueden crear nuevos contextos utilizando el constructor de la clase `Context` que se describe a continuación. Además, el módulo proporciona tres contextos prediseñados:

decimal.BasicContext

Este es un contexto estándar definido por la Especificación general de la aritmética decimal. La precisión se establece en nueve. El redondeo se establece en `ROUND_HALF_UP`. Se restablecen todos los flags. Todas las trampas están habilitadas (las señales son tratadas como excepciones) excepto `Inexact`, `Rounded` y `Subnormal`.

Debido a que la mayoría de las trampas están habilitadas, este contexto es especialmente útil para la depuración.

decimal.ExtendedContext

Este es un contexto estándar definido por la Especificación general de la aritmética decimal. La precisión se establece en nueve. El redondeo se establece en `ROUND_HALF_EVEN`. Se restablecen todos los flags. No se habilitan trampas (para que no se generen excepciones durante los cálculos).

Debido a que las trampas están deshabilitadas, este contexto es útil para aplicaciones que prefieren tener un valor `NaN` o `Infinity` como resultado en lugar de lanzar excepciones. Esto permite que una aplicación complete una ejecución en presencia de condiciones que, de otra manera, detendrían el programa.

decimal.DefaultContext

Este contexto es utilizado por el constructor de la clase `Context` como un prototipo para nuevos contextos. Cambiar un campo (como la precisión) tiene el efecto de cambiar el valor predeterminado para los nuevos contextos creados por el constructor de `Context`.

Este contexto es más útil en entornos con múltiples hilos. Cambiar uno de los campos antes de que se inicien los hilos tiene el efecto de establecer valores predeterminados en todo el sistema. No se recomienda cambiar los campos después de que se hayan iniciado los hilos, ya que requeriría el uso de mecanismos de sincronización para evitar condiciones de carrera entre los hilos.

En entornos de un solo hilo, es preferible no utilizar este contexto en absoluto. En su lugar, simplemente crea contextos explícitamente como se describe a continuación.

Los valores predeterminados son `Context.prec`=`28`, `Context.rounding`=`ROUND_HALF_EVEN`, y trampas habilitadas para `Overflow`, `InvalidOperation`, y `DivisionByZero`.

Además de los tres contextos proporcionados, se pueden crear nuevos contextos mediante el constructor de la clase `Context`.

class decimal.Context(*prec=None*, *rounding=None*, *Emin=None*, *Emax=None*, *capitals=None*, *clamp=None*, *flags=None*, *traps=None*)

Crea un nuevo contexto. Si no se especifica un campo, o es **`None`**, los valores predeterminados se copian de **`DefaultContext`**. Si el campo *flags* no está especificado, o es **`None`**, se restablecen todas los flags.

prec

An integer in the range [`1`, **`MAX_PREC`**] that sets the precision for arithmetic operations in the context.

rounding

One of the constants listed in the section Rounding Modes.

traps
flags

Lists of any signals to be set. Generally, new contexts should only set traps and leave the flags clear.

Emin
Emax

Integers specifying the outer limits allowable for exponents. *Emin* must be in the range [**`MIN_EMIN`**, `0`], *Emax* in the range [`0`, **`MAX_EMAX`**].

capitals

Either `0` or `1` (the default). If set to `1`, exponents are printed with a capital `E`; otherwise, a lowercase `e` is used: `Decimal('6.02e+23')`.

clamp

Either `0` (the default) or `1`. If set to `1`, the exponent `e` of a **`Decimal`** instance representable in this context is strictly limited to the range `Emin - prec + 1 <= e <= Emax - prec + 1`. If *clamp* is `0` then a weaker condition holds: the adjusted exponent of the **`Decimal`** instance is at most **`Emax`**. When *clamp* is `1`, a large normal number will, where possible, have its exponent reduced and a corresponding number of zeros added to its coefficient, in order to fit the exponent constraints; this preserves the value of the number but loses information about significant trailing zeros. For example:

```
>>> Context(prec=6, Emax=999, clamp=1).create_c
Decimal('1.23000E+999')
```

Un valor *clamp* de `1` permite la compatibilidad con los formatos de intercambio decimal de ancho fijo especificados en IEEE 754.

La clase **`Context`** define varios métodos de propósito general, así como una gran cantidad de métodos para hacer aritmética directamente en un contexto dado. Además, para cada uno de los métodos de la clase **`Decimal`** descritos anteriormente (con la excepción de los métodos **`adjusted()`** y **`as_tuple()`**) hay un método correspondiente en la clase **`Context`**. Por ejemplo, para una instancia de **`Context`** `C` y una instancia de **`Decimal`** `x`, `C.exp(x)` es equivalente a `x.exp(context=C)`. Cada método **`Context`** acepta también un entero de Python (una instancia de **`int`**) en cualquier lugar donde se acepte una instancia de Decimal.

clear_flags()

Restablece todos los flags a `0`.

clear_traps()

Restablece todas las trampas a `0`.

Added in version 3.3.

copy()

Retorna un duplicado del contexto.

copy_decimal(*num*, */*)

Retorna una copia de la instancia de Decimal num.

create_decimal(*num='0'*, */*)

Crea una nueva instancia de Decimal a partir de *num* pero usando *self* como contexto. A diferencia del constructor de **`Decimal`**, la precisión del contexto, el método de redondeo, los flags y las trampas se aplican a la conversión.

Esto es útil porque las constantes a menudo se proporcionan con una precisión mayor que la que necesita la aplicación. Otro beneficio es que el redondeo elimina inmediatamente los efectos no deseados de los dígitos más allá de la precisión actual. En el siguiente ejemplo, usar entradas no redondeadas significa que agregar cero a una suma puede cambiar el resultado:

```
>>> getcontext().prec = 3
>>> Decimal('3.4445') + Decimal('1.0023')
Decimal('4.45')
>>> Decimal('3.4445') + Decimal(0) + Decimal('1
Decimal('4.44')
```

Este método implementa la operación to-number de la especificación de IBM. Si el argumento es una cadena de caracteres, no se permiten espacios en blanco ni guiones bajos, ni al principio ni al final.

create_decimal_from_float(*f*, */*)

Crea una nueva instancia de Decimal a partir de un flotante *f*, pero redondeando usando *self* como contexto. A diferencia del método de clase **`Decimal.from_float()`**, la precisión del contexto, el método de redondeo, los flags y las trampas se aplican a la conversión.

```
>>> context = Context(prec=5, rounding=ROUND_DO
>>> context.create_decimal_from_float(math.pi)
Decimal('3.1415')
>>> context = Context(prec=5, traps=[Inexact])
>>> context.create_decimal_from_float(math.pi)
Traceback (most recent call last):
    ...
decimal.Inexact: None
```

Added in version 3.1.

Etiny()

Retorna un valor igual a `Emin - prec + 1` que es el valor mínimo del exponente para resultados subnormales. Cuando ocurre un desbordamiento numérico negativo («underflow»), el exponente se establece en `Etiny`.

Etop()

Retorna un valor igual a `Emax - prec + 1`.

El enfoque habitual para trabajar con decimales es crear instancias de la clase `Decimal` y luego aplicar operaciones aritméticas que tienen lugar dentro del contexto actual para el hilo activo. Un enfoque alternativo es utilizar métodos de contexto para calcular dentro de un contexto específico. Los métodos son similares a los de la clase **`Decimal`** y aquí solo se relatan brevemente.

abs(*x*, */*)

Retorna el valor absoluto de *x*.

add(*x*, *y*, */*)

Retorna la suma de *x* e *y*.

canonical(*x*, */*)

Retorna el mismo objeto Decimal *x*.

compare(*x*, *y*, */*)

Compara *x* e *y* numéricamente.

compare_signal(*x*, *y*, */*)

Compara los valores de los dos operandos numéricamente.

compare_total(*x*, *y*, */*)

Compara los dos operandos utilizando su representación abstracta.

compare_total_mag(*x*, *y*, */*)

Compara los dos operandos utilizando su representación abstracta, ignorando el signo.

copy_abs(*x*, */*)

Retorna una copia de *x* con el signo establecido en 0.

copy_negate(*x*, */*)

Retorna una copia de *x* con el signo invertido.

copy_sign(*x*, *y*, */*)

Copia el signo de *y* en *x*.

divide(*x*, *y*, */*)

Retorna *x* dividido entre *y*.

divide_int(*x*, *y*, */*)

Retorna *x* dividido entre *y*, truncando el resultado a un número entero.

divmod(*x*, *y*, */*)

Divide dos números y retorna la parte entera del resultado.

exp(*x*, /)

Retorna `e ** x`.

fma(*x*, *y*, *z*, /)

Retorna *x* multiplicado por *y*, más *z*.

is_canonical(*x*, /)

Retorna `True` si *x* está en forma canónica, en caso contrario retorna `False`.

is_finite(*x*, /)

Retorna `True` si *x* es un valor finito, en caso contrario retorna `False`.

is_infinite(*x*, /)

Retorna `True` si *x* es un valor infinito, en caso contrario retorna `False`.

is_nan(*x*, /)

Retorna `True` si *x* es un valor qNaN o sNaN , en caso contrario retorna `False`.

is_normal(*x*, /)

Retorna `True` si *x* es un número normal, en caso contrario retorna `False`.

is_qnan(*x*, /)

Retorna `True` si *x* es un NaN silencioso, en caso contrario retorna `False`.

is_signed(*x*, /)

Retorna `True` si *x* es un valor negativo, en caso contrario retorna `False`.

is_snan(*x*, /)

Retorna `True` si *x* es un NaN señalizador, en caso contrario retorna `False`.

is_subnormal(*x*, /)

Retorna `True` si *x* es un número subnormal, en caso contrario retorna `False`.

is_zero(*x*, /)

Retorna `True` si *x* es un cero, en caso contrario retorna `False`.

ln(*x*, /)

Retorna el logaritmo natural (base e) de *x*.

log10(*x*, /)

Retorna el logaritmo en base 10 de *x*.

logb(*x*, /)

Retorna el exponente de la magnitud del MSD («dígito más significativo») del operando.

logical_and(*x*, *y*, /)

Aplica la operación lógica *and* entre los dígitos de cada operando.

logical_invert(*x*, /)

Invierte todos los dígitos en *x*.

logical_or(*x*, *y*, /)

Aplica la operación lógica *or* entre los dígitos de cada operando.

logical_xor(*x*, *y*, /)

Aplica la operación lógica *xor* entre los dígitos de cada operando.

max(*x*, *y*, /)

Compara dos valores numéricamente y retorna el mayor de ellos.

max_mag(*x*, *y*, /)

Compara los valores numéricamente ignorando sus signos.

min(*x*, *y*, /)

Compara dos valores numéricamente y retorna el menor de ellos.

min_mag(*x*, *y*, /)

Compara los valores numéricamente ignorando sus signos.

minus(*x*, /)

Se corresponde con el operador unario de resta (prefijo) de Python.

multiply(*x*, *y*, /)

Retorna el producto de *x* por *y*.

next_minus(*x*, /)

Retorna el número más grande representable menor que *x*.

next_plus(*x*, /)

Retorna el número más pequeño representable mayor que *x*.

next_toward(*x*, *y*, /)

Retorna el número más cercano a *x*, en la dirección de *y*.

normalize(*x*, /)

Reduce *x* a su forma más simple.

number_class(*x*, /)

Retorna una cadena de caracteres indicando la clase de *x*.

plus(*x*, /)

Se corresponde con el operador unario de suma (prefijo) de Python. Esta operación aplica la precisión y el redondeo establecidos en el contexto, por lo que *no* es una operación de identidad.

power(*x*, *y*, *modulo=None*)

Retorna `x` elevado a la potencia `y`, reconduciendo al módulo `modulo` si se proporciona.

Con dos argumentos, calcula `x**y`. Si `x` es negativo, entonces `y` debe ser un entero. El resultado será inexacto, a menos que `y` sea un entero y el resultado sea finito y pueda expresarse exactamente con los dígitos de la "precisión" establecida. Se utiliza el modo de redondeo establecido en el contexto. Los resultados siempre se redondean correctamente en la versión de Python.

`Decimal(0) ** Decimal(0)` da como resultado `InvalidOperation`, y si `InvalidOperation` no es atrapada, entonces da como resultado `Decimal('NaN')`.

Distinto en la versión 3.3: El módulo C calcula **`power()`** en términos de las funciones **`exp()`** y **`ln()`** redondeadas correctamente. El resultado está bien definido pero sólo «casi siempre correctamente redondeado».

Con tres argumentos, calcula `(x**y) % modulo`. Para la forma de tres argumentos, se mantienen las siguientes restricciones sobre los argumentos:

- los tres argumentos deben ser enteros
- `y` debe ser un valor no negativo
- al menos uno, `x` o `y`, no debe ser cero
- `modulo` no debe ser cero y tener como mínimo los dígitos de la "precisión"

El valor resultante de `Context.power(x, y, modulo)` es igual al valor que se obtendría calculando `(x**y) % modulo` con precisión ilimitada, la diferencia es que se calcula de manera más eficiente . El exponente del resultado es cero, independientemente de los exponentes de `x`, `y` y `modulo`. El resultado siempre es exacto.

quantize(*x*, *y*, */*)

Retorna un valor igual a *x* (redondeado), pero que tiene el exponente de *y*.

radix()

Simplemente retorna 10, ya que es Decimal, :)

remainder(*x*, *y*, */*)

Retorna el resto de la división entera.

El signo del resultado, si no es cero, es el mismo que el del dividendo original.

remainder_near(*x*, *y*, */*)

Retorna `x - y * n`, donde *n* es el número entero más cercano al valor exacto de `x / y` (si el resultado es 0, entonces su signo será el signo de *x*).

rotate(*x*, *y*, */*)

Retorna una copia de *x* rotada *y* veces.

same_quantum(*x*, *y*, */*)

Retorna `True` si los dos operandos tienen el mismo exponente.

scaleb(*x*, *y*, */*)

Retorna el primer operando después de agregar el segundo valor a su exponente.

shift(*x*, *y*, */*)

Retorna una copia de *x* desplazada *y* veces.

sqrt(*x*, */*)

Retorna la raíz cuadrada de un número no negativo para la precisión del contexto.

subtract(*x*, *y*, */*)

Retorna la diferencia entre *x* e *y*.

to_eng_string(*x*, */*)

Convierte a una cadena de caracteres, usando notación de ingeniería si se necesita un exponente.

La notación de ingeniería tiene como exponente un múltiplo de 3. Esto puede dejar hasta 3 dígitos a la izquierda del punto decimal y puede requerir la adición de uno o dos ceros finales.

to_integral_exact(*x*, */*)

Redondea a un entero.

to_sci_string(*x*, */*)

Convierte un número en una cadena de caracteres usando notación científica.

Constantes

Las constantes detalladas en esta sección solo son relevantes para el módulo de C. Se incluyen también en la versión pura de Python por compatibilidad.

	32-bit	**64-bit**
decimal.MAX_PREC	`425000000`	`999999999999999999`
decimal.MAX_EMAX	`425000000`	`999999999999999999`
decimal.MIN_EMIN	`-425000000`	`-999999999999999999`
decimal.MIN_ETINY	`-849999999`	`-1999999999999999997`
decimal.IEEE_CONTEXT_MAX_BITS	`256`	`512`

decimal.HAVE_THREADS

El valor es `True`. Está obsoleta, debido ha que Python ahora siempre tiene soporte para hilos.

Obsoleto desde la versión 3.9.

decimal.HAVE_CONTEXTVAR

El valor predeterminado es `True`. Si Python se **configura usando --without-decimal-contextvar**, la versión de C usa un contexto de hilos-locales en lugar de un contexto de corrutinas-locales y el valor de la constante es `False`. Esto es algo más rápido en algunos escenarios de contexto anidado.

Added in version 3.8.3.

Modos de redondeo

decimal.ROUND_CEILING

Redondear hacia `Infinity`.

decimal.ROUND_DOWN

Redondear hacia cero.

decimal.ROUND_FLOOR

Redondear hacia `-Infinity`.

decimal.ROUND_HALF_DOWN

Redondear al valor contiguo más cercano, con empates hacia cero.

decimal.ROUND_HALF_EVEN

Redondear al valor contiguo más cercano, con empates al entero par contiguo.

decimal.ROUND_HALF_UP

Redondear al valor contiguo más cercano, con empates alejándose de cero.

decimal.ROUND_UP

Redondear alejándose de cero.

decimal.ROUND_05UP

Si el último dígito después de redondear hacia cero es 0 ó 5, redondear alejándose de cero, en caso contrario, redondear hacia cero.

Señales

Las señales representan condiciones que surgen durante el cálculo. Cada una se corresponde con un solo flag de contexto y un habilitador de trampas de contexto.

El flag de contexto se establece siempre que se encuentra la condición. Después del cálculo, los flags pueden comprobarse con fines informativos (por ejemplo, para determinar si un cálculo fue exacto). Después de verificar los flags, asegúrate de borrarlos antes de comenzar con el siguiente cálculo.

Si el habilitador de trampas del contexto está configurado para la señal, entonces la condición hace que se lance una excepción de Python. Por ejemplo, si se establece la trampa `DivisionByZero`, se genera una excepción `DivisionByZero` al encontrar la condición.

class decimal.Clamped

Cambia un exponente para ajustar las restricciones de representación.

Normalmente, la restricción ocurre cuando un exponente cae fuera de los límites **`Emin`** y **`Emax`** del contexto. Si es posible, el exponente se reduce para ajustar agregando ceros al coeficiente.

class decimal.DecimalException

Clase base para otras señales. Es una subclase de **`ArithmeticError`**.

class decimal.DivisionByZero

Señala la división de un número no infinito entre cero.

Puede ocurrir en la división, en la división modular o al elevar un número a una potencia negativa. Si esta señal no es atrapada, se retorna `Infinity` o `-Infinity`, con el signo determinado por las entradas del cálculo.

class decimal.Inexact

Indica que se produjo un redondeo y el resultado no es exacto.

Señala que se descartaron dígitos distintos de cero durante el redondeo. Se retorna el resultado redondeado. El flag o la trampa de señal se utiliza para detectar cuando los resultados son inexactos.

class decimal.InvalidOperation

Señala que se realizó una operación no válida.

Indica que se solicitó una operación que no tiene lógica. Si esta señal no está atrapada, se retorna `NaN`. Las posibles causas incluyen:

```
Infinity - Infinity
0 * Infinity
Infinity / Infinity
```

```
x % 0
Infinity % x
sqrt(-x) and x > 0
0 ** 0
x ** (non-integer)
x ** Infinity
```

class decimal.Overflow

Desbordamiento numérico.

Indica que el exponente es mayor que **`Context.Emax`** después de que se haya producido el redondeo. Si no está atrapada, el resultado depende del modo de redondeo, ya sea tirando hacia adentro hasta el mayor número finito representable o redondeando hacia afuera a `Infinity`. En cualquier caso, también se activan las señales **`Inexact`** y **`Rounded`**.

class decimal.Rounded

Se produjo un redondeo, aunque posiblemente no hubo pérdida de información.

Señal lanzada cada vez que el redondeo descarta dígitos; incluso si esos dígitos son cero (como al redondear `5.00` a `5.0`). Si no está atrapada, se retorna el resultado sin cambios. Esta señal se utiliza para detectar la pérdida de dígitos significativos.

class decimal.Subnormal

El exponente antes del redondeo era menor que **`Emin`**.

Ocurre cuando el resultado de una operación es subnormal (el exponente es demasiado pequeño). Si no está atrapada, se retorna el resultado sin cambios.

class decimal.Underflow

Desbordamiento numérico negativo con resultado redondeado a cero.

Ocurre cuando un resultado subnormal se lleva a cero mediante redondeo. `Inexact` y `Subnormal` también se señalan.

class decimal.FloatOperation

Habilita una semántica más estricta para mezclar flotantes y objetos Decimal.

Si la señal no está atrapada (predeterminado), se permite mezclar flotantes y objetos Decimal en el constructor de `Decimal`, en el método `create_decimal()` y en todos los operadores de comparación. Tanto la conversión como las comparaciones son exactas. Cualquier ocurrencia de una operación mixta se registra silenciosamente estableciendo `FloatOperation` a los flags del contexto. Las conversiones explícitas usando `from_float()` o `create_decimal_from_float()` no establecen el flag.

En caso contrario (la señal está atrapada), solo las comparaciones de igualdad y las conversiones explícitas permanecen silenciadas. Todas las demás operaciones mixtas lanzan una excepción `FloatOperation`.

La siguiente tabla resume la jerarquía de señales:

```
exceptions.ArithmeticError(exceptions.Exception)
    DecimalException
        Clamped
        DivisionByZero(DecimalException, exceptions.Zero
        Inexact
            Overflow(Inexact, Rounded)
            Underflow(Inexact, Rounded, Subnormal)
        InvalidOperation
        Rounded
        Subnormal
        FloatOperation(DecimalException, exceptions.Type
```

Floating-point notes

Mitigación del error de redondeo usando mayor precisión

El uso de la coma flotante decimal elimina el error de representación decimal (haciendo posible representar `0.1` de forma exacta). Sin embargo, algunas operaciones aún pueden incurrir en errores de redondeo cuando los dígitos distintos de cero exceden la precisión fija.

The effects of round-off error can be amplified by the addition or subtraction of nearly offsetting quantities resulting in loss of significance. Knuth provides two instructive examples where rounded floating-point arithmetic with insufficient precision causes the breakdown of the associative and distributive properties of addition:

```
# Examples from Seminumerical Algorithms, Section 4.2.2.
>>> from decimal import Decimal, getcontext
>>> getcontext().prec = 8

>>> u, v, w = Decimal(11111113), Decimal(-11111111), Dec
>>> (u + v) + w
Decimal('9.5111111')
>>> u + (v + w)
Decimal('10')

>>> u, v, w = Decimal(20000), Decimal(-6), Decimal('6.00
>>> (u*v) + (u*w)
Decimal('0.01')
>>> u * (v+w)
Decimal('0.0060000')
```

The **`decimal`** module makes it possible to restore the identities by expanding the precision sufficiently to avoid loss of significance:

```
>>> getcontext().prec = 20
>>> u, v, w = Decimal(11111113), Decimal(-11111111), Dec
>>> (u + v) + w
Decimal('9.51111111')
```

```
>>> u + (v + w)
Decimal('9.51111111')
>>>
>>> u, v, w = Decimal(20000), Decimal(-6), Decimal('6.00
>>> (u*v) + (u*w)
Decimal('0.0060000')
>>> u * (v+w)
Decimal('0.0060000')
```

Valores especiales

The number system for the **`decimal`** module provides special values including `NaN`, `sNaN`, `-Infinity`, `Infinity`, and two zeros, `+0` and `-0`.

Los infinitos se pueden construir directamente con `Decimal('Infinity')`. Además, pueden surgir al dividir entre cero cuando la señal **`DivisionByZero`** no es interceptada. Asimismo, cuando la señal **`Overflow`** no es interceptada, un infinito puede resultar del redondeo más allá de los límites del mayor número representable.

Los infinitos tienen signo (afín) y se pueden usar en operaciones aritméticas donde se tratan como números muy grandes e indeterminados. Por ejemplo, adicionar una constante a infinito resulta en otro infinito.

Algunas operaciones son indeterminadas y retornan `NaN`, o lanzan una excepción si la señal **`InvalidOperation`** es atrapada. Por ejemplo, `0/0` retorna `NaN` que significa «no es un número». Esta variedad de `NaN` es silenciosa y, una vez creada, fluirá a través de otros cálculos dando siempre como resultado otro `NaN`. Este comportamiento puede ser útil para una serie de cálculos a los que ocasionalmente les faltan entradas, permitiendo que el cálculo continúe mientras se marcan resultados específicos como no válidos.

Una variante es `sNaN`, que emite una señal en lugar de permanecer en silencio después de cada operación. Este es un valor de retorno útil cuando un resultado no válido requiere interrumpir un cálculo para un manejo especial.

El comportamiento de los operadores de comparación de Python puede ser un poco sorprendente cuando está involucrado un valor `NaN`. Una prueba de igualdad donde uno de los operandos es un `NaN` silencioso o señalizador siempre retorna **`False`**` (incluso cuando se hace `Decimal('NaN')==Decimal('NaN')`), mientras que una prueba de desigualdad siempre retorna **`True`**. Un intento de comparar dos objetos Decimal usando cualquiera de los operadores `<`, `<=`, `>` o `>=` lanzará la señal **`InvalidOperation`** si alguno de los operandos es `NaN`, y retornará **`False`** si esta señal no es capturada. Ten en cuenta que la Especificación General de la Aritmética Decimal no especifica el comportamiento de las comparaciones directas. Estas reglas para las comparaciones que involucran a `NaN` se tomaron del estándar IEEE 854 (consulta la Tabla 3 en la sección 5.7). Utiliza en su lugar los métodos **`compare()`** y **`compare_signal()`** para garantizar el cumplimiento estricto de los estándares.

Los ceros con signo pueden resultar de cálculos que desbordan la precisión establecida. Mantienen el signo que habría resultado si el cálculo se hubiera realizado con mayor precisión. Dado que su magnitud es cero, los ceros positivos y negativos se tratan como iguales y su signo es solo informativo.

In addition to the two signed zeros which are distinct yet equal, there are various representations of zero with differing precisions yet equivalent in value. This takes a bit of getting used to. For an eye accustomed to normalized floating-point representations, it is not immediately obvious that the following calculation returns a value equal to zero:

```
>>> 1 / Decimal('Infinity')
Decimal('0E-1000026')
```

Trabajando con hilos

La función **`getcontext()`** accede a un objeto **`Context`** diferente para cada hilo. Tener contextos de hilo separados significa que los hilos pueden realizar cambios (como `getcontext().prec=10`) sin interferir con otros hilos.

Asimismo, la función `setcontext()` asigna automáticamente su objetivo al hilo actual.

If `setcontext()` has not been called before `getcontext()`, then `getcontext()` will automatically create a new context for use in the current thread. New context objects have default values set from the `decimal.DefaultContext` object.

The `sys.flags.thread_inherit_context` flag affects the context for new threads. If the flag is false, new threads will start with an empty context. In this case, `getcontext()` will create a new context object when called and use the default values from *DefaultContext*. If the flag is true, new threads will start with a copy of context from the caller of `threading.Thread.start()`.

To control the defaults so that each thread will use the same values throughout the application, directly modify the *DefaultContext* object. This should be done *before* any threads are started so that there won't be a race condition between threads calling `getcontext()`. For example:

```
# Set applicationwide defaults for all threads about to
DefaultContext.prec = 12
DefaultContext.rounding = ROUND_DOWN
DefaultContext.traps = ExtendedContext.traps.copy()
DefaultContext.traps[InvalidOperation] = 1
setcontext(DefaultContext)

# Afterwards, the threads can be started
t1.start()
t2.start()
t3.start()
 . . .
```

Casos prácticos

A continuación hay algunos casos prácticos que sirven como funciones de utilidad y que muestran formas de trabajar con la clase

Decimal:

```
def moneyfmt(value, places=2, curr='', sep=',', dp='.',
             pos='', neg='-', trailneg=''):
    """Convert Decimal to a money formatted string.

    places:  required number of places after the decimal
    curr:    optional currency symbol before the sign (m
    sep:     optional grouping separator (comma, period,
    dp:      decimal point indicator (comma or period)
             only specify as blank when places is zero
    pos:     optional sign for positive numbers: '+', sp
    neg:     optional sign for negative numbers: '-', '(
    trailneg:optional trailing minus indicator:  '-', ')

    >>> d = Decimal('-1234567.8901')
    >>> moneyfmt(d, curr='$')
    '-$1,234,567.89'
    >>> moneyfmt(d, places=0, sep='.', dp='', neg='', tr
    '1.234.568-'
    >>> moneyfmt(d, curr='$', neg='(', trailneg=')')
    '($1,234,567.89)'
    >>> moneyfmt(Decimal(123456789), sep=' ')
    '123 456 789.00'
    >>> moneyfmt(Decimal('-0.02'), neg='<', trailneg='>'
    '<0.02>'

    """
    q = Decimal(10) ** -places      # 2 places --> '0.01
    sign, digits, exp = value.quantize(q).as_tuple()
    result = []
    digits = list(map(str, digits))
    build, next = result.append, digits.pop
    if sign:
        build(trailneg)
    for i in range(places):
        build(next() if digits else '0')
    if places:
        build(dp)
    if not digits:
        build('0')
    i = 0
    while digits:
        build(next())
        i += 1
        if i == 3 and digits:
            i = 0
            build(sep)
    build(curr)
    build(neg if sign else pos)
    return ''.join(reversed(result))

def pi():
    """Compute Pi to the current precision.

    >>> print(pi())
    3.141592653589793238462643383

    """
    getcontext().prec += 2  # extra digits for intermedi
    three = Decimal(3)      # substitute "three=3.0" for
    lasts, t, s, n, na, d, da = 0, three, 3, 1, 0, 0, 24
    while s != lasts:
        lasts = s
        n, na = n+na, na+8
        d, da = d+da, da+32
        t = (t * n) / d
        s += t
    getcontext().prec -= 2
    return +s               # unary plus applies the new

def exp(x):
    """Return e raised to the power of x.  Result type m

    >>> print(exp(Decimal(1)))
    2.718281828459045235360287471
    >>> print(exp(Decimal(2)))
    7.38905609893065022723042746
    >>> print(exp(2.0))
    7.38905609893
    >>> print(exp(2+0j))
    (7.38905609893+0j)

    """
    getcontext().prec += 2
    i, lasts, s, fact, num = 0, 0, 1, 1, 1
    while s != lasts:
        lasts = s
        i += 1
        fact *= i
        num *= x
        s += num / fact
    getcontext().prec -= 2
    return +s

def cos(x):
    """Return the cosine of x as measured in radians.

    The Taylor series approximation works best for a sma
    For larger values, first compute x = x % (2 * pi).

    >>> print(cos(Decimal('0.5')))
    0.8775825618903727161162815826
    >>> print(cos(0.5))
    0.87758256189
    >>> print(cos(0.5+0j))
    (0.87758256189+0j)

    """
    getcontext().prec += 2
    i, lasts, s, fact, num, sign = 0, 0, 1, 1, 1, 1
    while s != lasts:
        lasts = s
        i += 2
        fact *= i * (i-1)
        num *= x * x
        sign *= -1
        s += num / fact * sign
    getcontext().prec -= 2
    return +s

def sin(x):
    """Return the sine of x as measured in radians.

    The Taylor series approximation works best for a sma
    For larger values, first compute x = x % (2 * pi).

    >>> print(sin(Decimal('0.5')))
    0.4794255386042030002732879352
    >>> print(sin(0.5))
    0.479425538604
    >>> print(sin(0.5+0j))
    (0.479425538604+0j)

    """
    getcontext().prec += 2
    i, lasts, s, fact, num, sign = 1, 0, x, 1, x, 1
    while s != lasts:
        lasts = s
        i += 2
        fact *= i * (i-1)
        num *= x * x
        sign *= -1
        s += num / fact * sign
    getcontext().prec -= 2
    return +s
```

Preguntas frecuentes sobre decimal

Q: It is cumbersome to type `decimal.Decimal('1234.5')`. Is there a way to minimize typing when using the interactive interpreter?

A: Some users abbreviate the constructor to just a single letter:

```
>>> D = decimal.Decimal
>>> D('1.23') + D('3.45')
Decimal('4.68')
```

Q: In a fixed-point application with two decimal places, some inputs have many places and need to be rounded. Others are not supposed to have excess digits and need to be validated. What methods should be used?

A: The `quantize()` method rounds to a fixed number of decimal places. If the `Inexact` trap is set, it is also useful for validation:

```
>>> TWOPLACES = Decimal(10) ** -2       # same as Decima

>>> # Round to two places
>>> Decimal('3.214').quantize(TWOPLACES)
Decimal('3.21')

>>> # Validate that a number does not exceed two places
>>> Decimal('3.21').quantize(TWOPLACES, context=Context(
Decimal('3.21')

>>> Decimal('3.214').quantize(TWOPLACES, context=Context
Traceback (most recent call last):
   ...
Inexact: None
```

Q: Once I have valid two place inputs, how do I maintain that invariant throughout an application?

A: Some operations like addition, subtraction, and multiplication by an integer will automatically preserve fixed point. Others operations, like division and non-integer multiplication, will change the number of decimal places and need to be followed-up with a `quantize()` step:

```
>>> a = Decimal('102.72')           # Initial fixed-poir
>>> b = Decimal('3.17')
>>> a + b                           # Addition preserves
```

```
Decimal('105.89')
>>> a - b
Decimal('99.55')
>>> a * 42                          # So does integer mu
Decimal('4314.24')
>>> (a * b).quantize(TWOPLACES)     # Must quantize non-
Decimal('325.62')
>>> (b / a).quantize(TWOPLACES)     # And quantize divis
Decimal('0.03')
```

Al desarrollar aplicaciones de coma fija, es conveniente definir funciones para gestionar el paso **`quantize()`**:

```
>>> def mul(x, y, fp=TWOPLACES):
...     return (x * y).quantize(fp)
...
>>> def div(x, y, fp=TWOPLACES):
...     return (x / y).quantize(fp)

>>> mul(a, b)                       # Automatically pres
Decimal('325.62')
>>> div(b, a)
Decimal('0.03')
```

Q: There are many ways to express the same value. The numbers `200`, `200.000`, `2E2`, and `.02E+4` all have the same value at various precisions. Is there a way to transform them to a single recognizable canonical value?

A: The **`normalize()`** method maps all equivalent values to a single representative:

```
>>> values = map(Decimal, '200 200.000 2E2 .02E+4'.split
>>> [v.normalize() for v in values]
[Decimal('2E+2'), Decimal('2E+2'), Decimal('2E+2'), Deci
```

Q: When does rounding occur in a computation?

A: It occurs *after* the computation. The philosophy of the decimal specification is that numbers are considered exact and are created independent of the current context. They can even have greater precision than current context. Computations process with those exact inputs and then rounding (or other context operations) is applied to the *result* of the computation:

```
>>> getcontext().prec = 5
>>> pi = Decimal('3.1415926535')   # More than 5 digits
>>> pi                             # All digits are reta
Decimal('3.1415926535')
>>> pi + 0                         # Rounded after an ac
Decimal('3.1416')
>>> pi - Decimal('0.00005')        # Subtract unrounded
Decimal('3.1415')
>>> pi + 0 - Decimal('0.00005').   # Intermediate values
Decimal('3.1416')
```

Q: Some decimal values always print with exponential notation. Is there a way to get a non-exponential representation?

A: For some values, exponential notation is the only way to express the number of significant places in the coefficient. For example, expressing `5.0E+3` as `5000` keeps the value constant but cannot show the original's two-place significance.

Si una aplicación no necesita preocuparse por el seguimiento de significación, es fácil eliminar el exponente y los ceros finales, perdiendo significación, pero manteniendo el valor sin cambios:

```
>>> def remove_exponent(d):
...     return d.quantize(Decimal(1)) if d == d.to_integ

>>> remove_exponent(Decimal('5E+3'))
Decimal('5000')
```

Q: Is there a way to convert a regular float to a **`Decimal`**?

A: Yes, any binary floating-point number can be exactly expressed as a Decimal though an exact conversion may take more precision than intuition would suggest:

```
>>> Decimal(math.pi)
```

```
Decimal('3.14159265358979311599796346854418516159057617
```

Q: Within a complex calculation, how can I make sure that I haven’t gotten a spurious result because of insufficient precision or rounding anomalies.

A: The decimal module makes it easy to test results. A best practice is to re-run calculations using greater precision and with various rounding modes. Widely differing results indicate insufficient precision, rounding mode issues, ill-conditioned inputs, or a numerically unstable algorithm.

Q: I noticed that context precision is applied to the results of operations but not to the inputs. Is there anything to watch out for when mixing values of different precisions?

A: Yes. The principle is that all values are considered to be exact and so is the arithmetic on those values. Only the results are rounded. The advantage for inputs is that «what you type is what you get». A disadvantage is that the results can look odd if you forget that the inputs haven’t been rounded:

```
>>> getcontext().prec = 3
>>> Decimal('3.104') + Decimal('2.104')
Decimal('5.21')
>>> Decimal('3.104') + Decimal('0.000') + Decimal('2.104
Decimal('5.20')
```

La solución es aumentar la precisión o forzar el redondeo de las entradas utilizando la operación unaria más:

```
>>> getcontext().prec = 3
>>> +Decimal('1.23456789')      # unary plus triggers ro
Decimal('1.23')
```

Alternativamente, las entradas se pueden redondear en el momento que se crean usando el método **`Context.create_decimal()`**:

```
>>> Context(prec=5, rounding=ROUND_DOWN).create_decimal(
Decimal('1.2345')
```

Q: Is the CPython implementation fast for large numbers?

A: Yes. In the CPython and PyPy3 implementations, the C/CFFI versions of the decimal module integrate the high speed libmpdec [https://www.bytereef.org/mpdecimal/doc/libmpdec/index.html] library for arbitrary precision correctly rounded decimal floating-point arithmetic [1]. `libmpdec` uses Karatsuba multiplication [https://en.wikipedia.org/wiki/Karatsuba_algorithm] for medium-sized numbers and the Number Theoretic Transform [https://en.wikipedia.org/wiki/Discrete_Fourier_transform_(general)#Number-theoretic_transform] for very large numbers.

El contexto debe adaptarse para una aritmética de precisión arbitraria exacta. `Emin` y `Emax` siempre deben establecerse en sus valores máximos, `clamp` siempre debe ser 0 (el valor predeterminado). Establecer `prec` requiere cierto cuidado.

El enfoque más fácil para probar la aritmética bignum es usar también el valor máximo para `prec` [2]:

```
>>> setcontext(Context(prec=MAX_PREC, Emax=MAX_EMAX, Emi
>>> x = Decimal(2) ** 256
>>> x / 128
Decimal('9046256971665327767466483203803742801036717552 0
```

For inexact results, `MAX_PREC` is far too large on 64-bit platforms and the available memory will be insufficient:

```
>>> Decimal(1) / 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
MemoryError
```

En sistemas con sobreasignación (por ejemplo, Linux), un enfoque más sofisticado es establecer `prec` a la cantidad de RAM disponible. Supongamos que tenemos 8GB de RAM y esperamos 10 operandos simultáneos usando un máximo de 500 MB cada uno:

```
>>> import sys
>>>
>>> # Maximum number of digits for a single operand usin
>>> # with 19 digits per word (4-byte and 9 digits for t
>>> maxdigits = 19 * ((500 * 1024**2) // 8)
>>>
>>> # Check that this works:
>>> c = Context(prec=maxdigits, Emax=MAX_EMAX, Emin=MIN_
>>> c.traps[Inexact] = True
>>> setcontext(c)
>>>
>>> # Fill the available precision with nines:
>>> x = Decimal(0).logical_invert() * 9
>>> sys.getsizeof(x)
524288112
>>> x + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  decimal.Inexact: [<class 'decimal.Inexact'>]
```

En general (y especialmente en sistemas sin sobreasignación), se recomienda estimar límites aún más estrictos y establecer la trampa **`Inexact`** si se espera que todos los cálculos sean exactos.

Added in version 3.3.

Distinto en la versión 3.9: Este enfoque ahora funciona para todos los resultados exactos excepto para las potencias no enteras. También retro-portado a 3.7 y 3.8.

`fractions` — Rational numbers

Source code: Lib/fractions.py [https://github.com/python/cpython/tree/3.14/Lib/fractions.py]

The **`fractions`** module provides support for rational number arithmetic.

A Fraction instance can be constructed from a pair of rational numbers, from a single number, or from a string.

class fractions.Fraction(*numerator=0*, *denominator=1*)
class fractions.Fraction(*number*)
class fractions.Fraction(*string*)

> The first version requires that *numerator* and *denominator* are instances of **`numbers.Rational`** and returns a new **`Fraction`** instance with a value equal to `numerator/denominator`. If *denominator* is zero, it raises a **`ZeroDivisionError`**.
>
> The second version requires that *number* is an instance of **`numbers.Rational`** or has the **`as_integer_ratio()`** method (this includes **`float`** and **`decimal.Decimal`**). It returns a **`Fraction`** instance with exactly the same value. Assumed, that the **`as_integer_ratio()`** method returns a pair of coprime integers and last one is positive. Note that due to the usual issues with binary point (see Floating-Point Arithmetic: Issues and Limitations), the argument to `Fraction(1.1)` is not exactly equal to 11/10, and so `Fraction(1.1)` does *not* return `Fraction(11, 10)` as one might expect. (But see the documentation for the **`limit_denominator()`** method below.)

The last version of the constructor expects a string. The usual form for this instance is:

```
[sign] numerator ['/' denominator]
```

donde el `sign` opcional puede ser "+" o "-" y `numerator` y `denominator` (si están presentes) son cadenas de caracteres de dígitos decimales (guiones bajos se pueden usar para delimitar dígitos como con las integrales literales en el código). Además, cualquier cadena de caracteres que represente un valor finito y sea aceptado por el constructor de **`float`** también es aceptado por el constructor de **`Fraction`**. En cualquier caso, la cadena de caracteres de entrada también puede tener espacios en blanco iniciales y / o finales. Aquí hay unos ejemplos:

```
>>> from fractions import Fraction
>>> Fraction(16, -10)
Fraction(-8, 5)
>>> Fraction(123)
Fraction(123, 1)
>>> Fraction()
Fraction(0, 1)
>>> Fraction('3/7')
Fraction(3, 7)
>>> Fraction(' -3/7 ')
Fraction(-3, 7)
>>> Fraction('1.414213 \t\n')
Fraction(1414213, 1000000)
>>> Fraction('-.125')
Fraction(-1, 8)
>>> Fraction('7e-6')
Fraction(7, 1000000)
>>> Fraction(2.25)
Fraction(9, 4)
>>> Fraction(1.1)
Fraction(2476979795053773, 2251799813685248)
>>> from decimal import Decimal
>>> Fraction(Decimal('1.1'))
```

```
Fraction(11, 10)
```

La clase `Fraction` hereda de la clase base abstracta `numbers.Rational`, e implementa todos los métodos y operaciones de esa clase. Las instancias **`Fraction`** son hashable, y deben ser tratadas como inmutables. Adicionalmente **`Fraction`** tiene los siguientes propiedades y métodos:

Distinto en la versión 3.2: El constructor de `Fraction` ahora acepta instancias de `float` y `decimal`.

Distinto en la versión 3.9: The `math.gcd()` function is now used to normalize the *numerator* and *denominator*. **`math.gcd()`** always returns an `int` type. Previously, the GCD type depended on *numerator* and *denominator*.

Distinto en la versión 3.11: Ahora se permiten guiones bajos al crear una instancia de `Fraction` a partir de una cadena de caracteres, siguiendo las reglas de **PEP 515** [https://peps.python.org/pep-0515/].

Distinto en la versión 3.11: `Fraction` ahora implementa `__int__` para satisfacer las comprobaciones de instancia de `typing.SupportsInt`.

Distinto en la versión 3.12: Se permite espacio alrededor de la barra para entrada de cadena de caracteres: `Fraction('2 / 3')`.

Distinto en la versión 3.12: `Fraction` instancias ahora apoya formato de estilo flotante, con tipos de presentación `"e"`, `"E"`, `"f"`, `"F"`, `"g"`, `"G"` and `"%""`.

Distinto en la versión 3.13: Formatting of **`Fraction`** instances without a presentation type now supports fill, alignment, sign handling, minimum width and grouping.

Distinto en la versión 3.14: The **`Fraction`** constructor now accepts any objects with the **`as_integer_ratio()`** method.

numerator

Numerador de la fracción irreducible.

denominator

Denominator of the Fraction in lowest terms. Guaranteed to be positive.

as_integer_ratio()

Retorna una tupla de dos números enteros, cuyo relación es igual a la fracción original.La relación está en términos más bajos y tiene un denominador positivo.

Added in version 3.8.

is_integer()

Retorna `True` si la fracción es un número entero.

Added in version 3.12.

classmethod from_float(*f*)

Constructor alternativo que solo acepta instancias de **`float`** o **`numbers.Integral`**. Ten cuidado que `Fraction.from_float(0.3)` no es lo mismo que `Fraction(3, 10)`.

Nota

Desde Python 3.2 en adelante, puedes construir una instancia **`Fraction`** directamente desde **`float`**.

classmethod from_decimal(*dec*)

Constructor alternativo que solo acepta instancias de **`decimal.Decimal`** o **`numbers.Integral`**.

Nota

Desde Python 3.2 en adelante, puedes construir una instancia **`Fraction`** directamente desde una instancia **`decimal.Decimal`**.

classmethod from_number(*number*)

Alternative constructor which only accepts instances of **`numbers.Integral`**, **`numbers.Rational`**, **`float`** or **`decimal.Decimal`**, and objects with the **`as_integer_ratio()`** method, but not strings.

Added in version 3.14.

limit_denominator(*max_denominator=1000000*)

Busca y retorna la instancia de **`Fraction`** mas cercana a `self` que tenga como denominador *max_denominator*. Este método es útil para encontrar aproximaciones racionales a un número en punto flotante determinado:

```
>>> from fractions import Fraction
>>> Fraction('3.1415926535897932').limit_denomi
Fraction(355, 113)
```

o para recuperar un numero racional que esta representado como flotante:

```
>>> from math import pi, cos
>>> Fraction(cos(pi/3))
Fraction(4503599627370497, 9007199254740992)
>>> Fraction(cos(pi/3)).limit_denominator()
Fraction(1, 2)
>>> Fraction(1.1).limit_denominator()
Fraction(11, 10)
```

__floor__()

Retorna el máximo **`int`** `<= self`. Este método puede accederse también a través de la función **`math.floor()`**:

```
>>> from math import floor
>>> floor(Fraction(355, 113))
3
```

ceil()

Retorna el mínimo **`int`** `>= self`. Este método puede accederse también a través de la función **`math.ceil()`**.

round()
round(*ndigits*)

La primera versión retorna el valor **`int`** mas cercano a `self` redondeando mitades al valor par. La segunda versión redondea `self` al múltiplo mas cercano de `Fraction(1, 10**ndigits)` (lógicamente, si `ndigits` es negativo), nuevamente redondeando mitades al valor par. Este método también puede accederse a través de la función **`round()`**.

format(*format_spec*, */*)

Provides support for formatting of **`Fraction`** instances via the **`str.format()`** method, the **`format()`** built-in function, or Formatted string literals.

If the `format_spec` format specification string does not end with one of the presentation types `'e'`, `'E'`, `'f'`, `'F'`, `'g'`, `'G'` or `'%'` then formatting follows the general rules for fill, alignment, sign handling, minimum width, and grouping as described in the format specification mini-language. The «alternate form» flag `'#'` is supported: if present, it forces the output string to always include an explicit denominator, even when the value being formatted is an exact integer. The zero-fill flag `'0'` is not supported.

If the `format_spec` format specification string ends with one of the presentation types `'e'`, `'E'`, `'f'`, `'F'`, `'g'`, `'G'` or `'%'` then formatting follows the rules outlined for the **`float`** type in the Format specification mini-language section.

Aquí hay unos ejemplos:

```
>>> from fractions import Fraction
>>> format(Fraction(103993, 33102), '_')
'103_993/33_102'
>>> format(Fraction(1, 7), '.^+10')
'...+1/7...'
>>> format(Fraction(3, 1), '')
'3'
>>> format(Fraction(3, 1), '#')
'3/1'
>>> format(Fraction(1, 7), '.40g')
'0.1428571428571428571428571428571428571429'
>>> format(Fraction('1234567.855'), '_.2f')
'1_234_567.86'
>>> f"{Fraction(355, 113):*>20.6e}"
'********3.141593e+00'
>>> old_price, new_price = 499, 672
>>> "{:.2%} price increase".format(Fraction(new
'34.67% price increase'
```

Ver también

Módulo `numbers`
Las clases base abstractas que representan la jerarquía de números.

`random` — Generate pseudo-random numbers

Código fuente: Lib/random.py [https://github.com/python/cpython/tree/3.14/Lib/random.py]

Este módulo implementa generadores de números pseudoaleatorios para varias distribuciones.

Para los enteros, existe una selección uniforme dentro de un rango. Para las secuencias, existe una selección uniforme de un elemento aleatorio, una función para generar una permutación aleatoria de una lista *in-situ* y una función para el muestreo aleatorio sin reemplazo.

Para números reales, existen funciones para calcular distribuciones uniformes, normales (Gaussianas), log-normales, exponenciales negativas, gamma y beta. Para generar distribuciones angulares está disponible la distribución de von Mises.

Almost all module functions depend on the basic function **`random()`**, which generates a random float uniformly in the half-open range `0.0 <= X < 1.0`. Python uses the Mersenne Twister as the core generator. It produces 53-bit precision floats and has a period of 2**19937-1. The underlying implementation in C is both fast and threadsafe. The Mersenne Twister is one of the most extensively tested random number generators in existence. However, being completely deterministic, it is not suitable for all purposes, and is completely unsuitable for cryptographic purposes.

Las funciones proporcionadas por este módulo en realidad son métodos enlazados a instancias ocultas a la clase **`random.Random`**. Puedes instanciar tus propias instancias de **`Random`** para obtener generadores que no compartan estado.

Class `Random` can also be subclassed if you want to use a different basic generator of your own devising: see the documentation on that class for more details.

The `random` module also provides the `SystemRandom` class which uses the system function `os.urandom()` to generate random numbers from sources provided by the operating system.

Advertencia

Los generadores pseudoaleatorios de este módulo no deben ser utilizados con fines de seguridad. Para usos de seguridad o criptográficos, consulta el módulo `secrets`.

Ver también

M. Matsumoto and T. Nishimura, «Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator», ACM Transactions on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3–30 1998.

Complementary-Multiply-with-Carry recipe [https://code.activestate.com/recipes/576707-long-period-random-number-generator/] for a compatible alternative random number generator with a long period and comparatively simple update operations.

Nota

The global random number generator and instances of `Random` are thread-safe. However, in the free-threaded build, concurrent calls to the global generator or to the same instance of `Random` may encounter contention and poor performance. Consider using separate instances of `Random` per thread instead.

Funciones de contabilidad

random.seed(*a=None*, *version=2*)

Inicializa el generador de números aleatorios.

Si *a* es omitida o `None`, se utilizará la hora actual del sistema. Si las fuentes de aleatoriedad vienen del sistema operativo, éstas se usarán en vez de la hora del sistema (ver la función **`os.urandom()`** para detalles sobre su disponibilidad).

If *a* is an int, its absolute value is used directly.

Con la versión 2 (la versión por defecto), un objeto **`str`**, **`bytes`**, o **`bytearray`** se convierte en **`int`** y se usarán todos sus bits.

Con la versión 1 (proporcionada para reproducir secuencias aleatorias desde versiones anteriores de Python), el algoritmo para **`str`** y **`bytes`** genera un rango de semillas más estrecho.

Distinto en la versión 3.2: El esquema que usa todos los bits en una semilla de tipo cadena, se ha movido a la versión 2.

Distinto en la versión 3.11: The *seed* must be one of the following types: `None`, **`int`**, **`float`**, **`str`**, **`bytes`**, or **`bytearray`**.

random.getstate()

Retorna un objeto capturando el estado interno del generador. Este objeto puede pasarse a **`setstate()`** para restaurar su estado.

random.setstate(*state*)

El *state* debería haberse obtenido de una llamada previa a **`getstate()`**, y **`setstate()`** reestablece el estado interno del generador al que tenia cuando se llamó a la función **`getstate()`**.

Funciones para los bytes

random.randbytes(*n*)

Genera *n* bytes aleatorios.

Este método no debe utilizarse para generar tokens de seguridad. Utilice **`secrets.token_bytes()`** en su lugar.

Added in version 3.9.

Funciones para enteros

random.randrange(*stop*)

random.randrange(*start*, *stop*[, *step*])

Return a randomly selected element from `range(start, stop, step)`.

This is roughly equivalent to `choice(range(start, stop, step))` but supports arbitrarily large ranges and is optimized for common cases.

The positional argument pattern matches the **`range()`** function.

Keyword arguments should not be used because they can be interpreted in unexpected ways. For example `randrange(start=100)` is interpreted as `randrange(0, 100, 1)`.

Distinto en la versión 3.2: **`randrange()`** es más sofisticado produciendo valores igualmente distribuidos. Anteriormente utilizaba un estilo como `int(random()*n)` el cual puede producir distribuciones ligeramente desiguales.

Distinto en la versión 3.12: Automatic conversion of non-integer types is no longer supported. Calls such as `randrange(10.0)` and `randrange(Fraction(10, 1))` now raise a **`TypeError`**.

random.randint(*a*, *b*)

Retorna un entero aleatorio *N* tal que `a <= N <= b`. Alias de `randrange(a, b+1)`.

random.getrandbits(*k*)

Returns a non-negative Python integer with *k* random bits. This method is supplied with the Mersenne Twister generator and some other generators may also provide it as an optional part of the API. When available, **`getrandbits()`** enables **`randrange()`** to handle arbitrarily large ranges.

Distinto en la versión 3.9: Este método ahora acepta cero para *k*.

Funciones para secuencias

random.choice(*seq*)

Retorna un elemento aleatorio de una secuencia *seq* no vacía. Si *seq* está vacía, lanza **`IndexError`**.

random.choices(*population*, *weights=None*, ***, *cum_weights=None*, *k=1*)

Retorna una lista de elementos de tamaño *k* elegidos de la *population* con reemplazo. Si la *population* está vacía, lanza **`IndexError`**.

Si se especifica una secuencia *weights*, las selecciones se realizan de acuerdo con las ponderaciones relativas. Alternativamente, si se da una secuencia *cum_weights*, las selecciones se harán según los pesos cumulativos (posiblemente se calculen usando **`itertools.accumulate()`**). Por ejemplo, los pesos relativos `[10, 5, 30, 5]` son equivalentes a los pesos cumulativos `[10, 15, 45, 50]`. Internamente, los pesos relativos se convierten en pesos cumulativos antes de hacer selecciones, por lo cual suplir los pesos cumulativos ahorra trabajo.

Si ni *weights* ni *cum_weights* están especificadas, las selecciones se realizan con la misma probabilidad. Si se proporciona una secuencia de ponderaciones, debe tener la misma longitud que la secuencia *population*. Es un `TypeError` especificar ambas *weights* y *cum_weights*.

Los *weights* o los *cum_weights* pueden utilizar cualquier tipo numérico que interactúe con los valores `float` retornados por `random()` (eso incluye enteros, flotantes y fracciones pero excluye decimales). Se asume que los pesos son no negativos y finitos. Se lanza un `ValueError` si todos los pesos son cero.

For a given seed, the `choices()` function with equal weighting typically produces a different sequence than repeated calls to `choice()`. The algorithm used by **`choices()`** uses floating-point arithmetic for internal consistency and speed. The algorithm used by **`choice()`** defaults to integer arithmetic with repeated selections to avoid small biases from round-off error.

Added in version 3.6.

Distinto en la versión 3.9: Genera un `ValueError` si todos los pesos son cero.

random.shuffle(*x*)

Mezcla la secuencia *x* in-situ.

Para mezclar una secuencia inmutable y retornar una nueva lista mezclada, utilice `muestra(x, k=len(x))` en su lugar.

Tenga en cuenta que incluso para pequeños `len(x)`, el número total de permutaciones de *x* puede crecer rápidamente más que el periodo de muchos generadores de números aleatorios. Esto implica que la mayoría de las permutaciones de una secuencia larga nunca se pueden generar. Por ejemplo, una secuencia de longitud 2080 es la más grande que cabe dentro del período del generador de números aleatorios de Mersenne Twister.

Distinto en la versión 3.11: Removed the optional parameter *random.*

random.sample(*population*, *k*, ***, *counts=None*)

Return a *k* length list of unique elements chosen from the population sequence. Used for random sampling without replacement.

Retorna una nueva lista que contiene elementos de la población sin modificar la población original. La lista resultante está en orden de selección de forma que todos los subsectores también son muestras aleatorias válidas. Esto permite que los ganadores de la rifa (la muestra) se dividan en primer premio y ganadores del segundo lugar (los subsectores).

Los miembros de la población no tienen porqué ser hashable o únicos. Si la población incluye repeticiones, entonces cada ocurrencia es una posible selección en la muestra.

Los elementos repetidos pueden especificarse de uno en uno o con el parámetro opcional *counts*, que es una palabra clave. Por ejemplo, `sample(['red', 'blue'], counts=[4, 2], k=5)` es equivalente a `sample(['red', 'red', 'red', 'red', 'blue', 'blue'], k=5)`.

Para escoger una muestra de un rango de enteros, use un objeto **`range()`** como argumento. Esto es especialmente rápido y eficiente en espacio para el muestreo de poblaciones grandes: `sample(range(10000000), k=60)`.

Si el tamaño de la muestra es mayor que el tamaño de la población, se lanzará un **`ValueError`**.

Distinto en la versión 3.9: Se añadió el parámetro *counts*.

Distinto en la versión 3.11: La población debe ser una secuencia. La conversión automática de sets a listas ya no es compatible.

Discrete distributions

The following function generates a discrete distribution.

random.binomialvariate(*n=1*, *p=0.5*)

Binomial distribution [https://mathworld.wolfram.com/BinomialDistribution.html]. Return the number of successes for *n* independent trials with the probability of success in each trial being *p*:

Mathematically equivalent to:

```
sum(random() < p for i in range(n))
```

The number of trials *n* should be a non-negative integer. The probability of success *p* should be between `0.0 <= p <= 1.0`. The result is an integer in the range `0 <= X <= n`.

Added in version 3.12.

Distribuciones para los nombres reales

Las siguientes funciones generan distribuciones específicas para números reales. Los parámetros de la función reciben el nombre de las variables correspondientes en la ecuación de distribución, tal y como se utilizan en la práctica matemática común.; la mayoría de estas ecuaciones se pueden encontrar en cualquier texto estadístico.

random.random()

Return the next random floating-point number in the range `0.0 <= X < 1.0`

random.uniform(*a*, *b*)

Return a random floating-point number *N* such that `a <= N <= b` for `a <= b` and `b <= N <= a` for `b < a`.

The end-point value `b` may or may not be included in the range depending on floating-point rounding in the expression `a + (b-a) * random()`.

random.triangular(*low*, *high*, *mode*)

Return a random floating-point number *N* such that `low <= N <= high` and with the specified *mode* between those bounds. The *low* and *high* bounds default to zero and one. The *mode* argument defaults to the midpoint between the bounds, giving a symmetric distribution.

random.betavariate(*alpha*, *beta*)

Distribución beta. Las condiciones de los parámetros son `alpha > 0` y `beta > 0`. Retorna valores dentro del rango entre 0 y 1.

random.expovariate(*lambd=1.0*)

Distribución exponencial. *lambd* es 1.0 dividido entre la media deseada. Debe ser distinto a cero (El parámetro debería llamarse `lambda` pero esa es una palabra reservada en Python). Retorna valores dentro del rango de 0 a infinito positivo si *lambd* es positivo, y de infinito negativo a 0 si *lambd* es negativo.

Distinto en la versión 3.12: Added the default value for `lambd`.

random.gammavariate(*alpha*, *beta*)

Gamma distribution. (*Not* the gamma function!) The shape and scale parameters, *alpha* and *beta*, must have positive values. (Calling conventions vary and some sources define "beta" as the inverse of the scale).

La función de distribución de la probabilidad es:

```
          x ** (alpha - 1) * math.exp(-x / beta)
pdf(x) =  --------------------------------------
            math.gamma(alpha) * beta ** alpha
```

random.gauss(*mu=0.0*, *sigma=1.0*)

Normal distribution, also called the Gaussian distribution. *mu* is the mean, and *sigma* is the standard deviation. This is slightly faster than the **`normalvariate()`** function defined below.

Nota sobre el multithreading: Cuando dos hilos llaman a esta función simultáneamente, es posible que reciban el mismo valor de retorno. Esto se puede evitar de tres maneras. 1) Hacer que cada hilo utilice una instancia diferente del generador de números aleatorios. 2) Poner bloqueos alrededor de todas las llamadas. 3) Utilizar la función `normalvariate()`, más lenta pero segura para los hilos, en su lugar.

Distinto en la versión 3.11: mu y *sigma* ahora tienen argumentos predeterminados.

random.lognormvariate(*mu*, *sigma*)

Logaritmo de la distribución normal. Si se usa un logaritmo natural de esta distribución, se obtendrá una distribución normal con media *mu* y desviación estándar *sigma*. *mu* puede tener cualquier valor, y *sigma* debe ser mayor que cero.

random.normalvariate(*mu=0.0*, *sigma=1.0*)

Distribución normal. *mu* es la media y *sigma* es la desviación estándar.

Distinto en la versión 3.11: mu y *sigma* ahora tienen argumentos predeterminados.

random.vonmisesvariate(*mu*, *kappa*)

mu es el ángulo medio, expresado en radiantes entre 0 y 2**pi*, y *kappa* es el parámetro de concentración, que debe ser mayor o igual a cero. Si *kappa* es igual a cero, esta distribución se reduce a un ángulo aleatorio uniforme sobre el rango de 0 a 2**pi*.

random.paretovariate(*alpha*)

Distribución de Pareto. *alpha* es el parámetro de forma.

random.weibullvariate(*alpha*, *beta*)

Distribución de Weibull. *alpha* es el parámetro de escala y *beta* es el parámetro de forma.

Generador alternativo

class random.Random([*seed*])

Class that implements the default pseudo-random number generator used by the **`random`** module.

Distinto en la versión 3.11: Formerly the *seed* could be any hashable object. Now it is limited to: `None`, **`int`**, **`float`**, **`str`**, **`bytes`**, or **`bytearray`**.

Subclasses of **`Random`** should override the following methods if they wish to make use of a different basic generator:

seed(*a=None*, *version=2*)

Override this method in subclasses to customise the **`seed()`** behaviour of **`Random`** instances.

getstate()

Override this method in subclasses to customise the **`getstate()`** behaviour of **`Random`** instances.

setstate(*state*)

Override this method in subclasses to customise the **`setstate()`** behaviour of **`Random`** instances.

random()

Override this method in subclasses to customise the **`random()`** behaviour of **`Random`** instances.

Optionally, a custom generator subclass can also supply the following method:

getrandbits(*k*)

Override this method in subclasses to customise the `getrandbits()` behaviour of `Random` instances.

randbytes(*n*)

Override this method in subclasses to customise the `randbytes()` behaviour of `Random` instances.

class random.SystemRandom([*seed*])

Clase que utiliza la función `os.urandom()` para generar números aleatorios a partir de fuentes proporcionadas por el sistema operativo. No está disponible en todos los sistemas. No se basa en el estado del software y las secuencias no son reproducibles. En consecuencia, el método `seed()` no tiene efecto y es ignorado. Los métodos `getstate()` y `setstate()` lanzan `NotImplementedError` si se les llama.

Notas sobre la Reproducibilidad

Sometimes it is useful to be able to reproduce the sequences given by a pseudo-random number generator. By reusing a seed value, the same sequence should be reproducible from run to run as long as multiple threads are not running.

Muchos de los algoritmos y de las funciones de generación de semillas del módulo aleatorio pueden cambiar entre versiones de Python, pero se garantiza que dos aspectos no cambien:

- Si se añade un nuevo método de generación de semilla, se ofrecerá un generador de semilla retrocompatible.
- El método generador `random()` continuará produciendo la misma secuencia cuando se le da la misma semilla al generador de semilla compatible.

Ejemplos

Ejemplos básicos:

```
>>> random()                             # Random float:
```

```
0.37444887175646646

>>> uniform(2.5, 10.0)                # Random float:  2
3.1800146073117523

>>> expovariate(1 / 5)                # Interval betweer
5.148957571865031

>>> randrange(10)                     # Integer from 0 t
7

>>> randrange(0, 101, 2)              # Even integer frc
26

>>> choice(['win', 'lose', 'draw'])   # Single random el
'draw'

>>> deck = 'ace two three four'.split()
>>> shuffle(deck)                     # Shuffle a list
>>> deck
['four', 'two', 'ace', 'three']

>>> sample([10, 20, 30, 40, 50], k=4) # Four samples wit
[40, 10, 50, 30]
```

Simulaciones:

```
>>> # Six roulette wheel spins (weighted sampling with r
>>> choices(['red', 'black', 'green'], [18, 18, 2], k=6)
['red', 'green', 'black', 'black', 'red', 'black']

>>> # Deal 20 cards without replacement from a deck
>>> # of 52 playing cards, and determine the proportion
>>> # with a ten-value:  ten, jack, queen, or king.
>>> deal = sample(['tens', 'low cards'], counts=[16, 36]
>>> deal.count('tens') / 20
0.15

>>> # Estimate the probability of getting 5 or more heac
>>> # of a biased coin that settles on heads 60% of the
>>> sum(binomialvariate(n=7, p=0.6) >= 5 for i in range(
0.4169

>>> # Probability of the median of 5 samples being in mi
>>> def trial():
...     return 2_500 <= sorted(choices(range(10_000), k=
...
>>> sum(trial() for i in range(10_000)) / 10_000
0.7958
```

Ejemplo de statistical bootstrapping [https://en.wikipedia.org/wiki/Bootstrapping_(statistics)] utilizando el remuestreo con reemplazo para estimar un intervalo de confianza para la media de una muestra:

```
# https://www.thoughtco.com/example-of-bootstrapping-312
from statistics import fmean as mean
from random import choices

data = [41, 50, 29, 37, 81, 30, 73, 63, 20, 35, 68, 22,
means = sorted(mean(choices(data, k=len(data))) for i in
print(f'The sample mean of {mean(data):.1f} has a 90% co
      f'interval from {means[5]:.1f} to {means[94]:.1f}'
```

Ejemplo de un test de permutación en remuestreo (en) [https://en.wikipedia.org/wiki/Resampling_(statistics)#Permutation_tests] para determinar la significación estadística o p-valor [https://es.wikipedia.org/wiki/Valor_p] de una diferencia observada entre los efectos de un fármaco y un placebo:

```
# Example from "Statistics is Easy" by Dennis Shasha and
from statistics import fmean as mean
from random import shuffle

drug = [54, 73, 53, 70, 73, 68, 52, 65, 65]
placebo = [54, 51, 58, 44, 55, 52, 42, 47, 58, 46]
observed_diff = mean(drug) - mean(placebo)

n = 10_000
```

```
count = 0
combined = drug + placebo
for i in range(n):
    shuffle(combined)
    new_diff = mean(combined[:len(drug)]) - mean(combine
    count += (new_diff >= observed_diff)

print(f'{n} label reshufflings produced only {count} ins
print(f'at least as extreme as the observed difference c
print(f'The one-sided p-value of {count / n:.4f} leads u
print(f'hypothesis that there is no difference between t
```

Simulación de tiempos de llegada y entrega de servicios para una cola de múltiples servidores:

```
from heapq import heapify, heapreplace
from random import expovariate, gauss
from statistics import mean, quantiles

average_arrival_interval = 5.6
average_service_time = 15.0
stdev_service_time = 3.5
num_servers = 3

waits = []
arrival_time = 0.0
servers = [0.0] * num_servers  # time when each server b
heapify(servers)
for i in range(1_000_000):
    arrival_time += expovariate(1.0 / average_arrival_in
    next_server_available = servers[0]
    wait = max(0.0, next_server_available - arrival_time
    waits.append(wait)
    service_duration = max(0.0, gauss(average_service_ti
    service_completed = arrival_time + wait + service_du
    heapreplace(servers, service_completed)

print(f'Mean wait: {mean(waits):.1f}   Max wait: {max(wa
print('Quartiles:', [round(q, 1) for q in quantiles(wait
```

Ver también

Statistics for Hackers [https://www.youtube.com/watch?v=Iq9DzN6mvYA] un video tutorial de Jake Vanderplas [https://us.pycon.org/2016/speaker/profile/295/] sobre análisis estadístico usando sólo algunos conceptos fundamentales incluyendo simulación, muestreo, baraja y validación cruzada.

Economics Simulation [https://nbviewer.org/url/norvig.com/ipython/Economics.ipynb] a simulation of a marketplace by Peter Norvig [https://norvig.com/bio.html] that shows effective use of many of the tools and distributions provided by this module (gauss, uniform, sample, betavariate, choice, triangular, and randrange).

A Concrete Introduction to Probability (using Python) [https://nbviewer.org/url/norvig.com/ipython/Probability.ipynb] a tutorial by Peter Norvig [https://norvig.com/bio.html] covering the basics of probability theory, how to write simulations, and how to perform data analysis using Python.

Recetas

Estas recetas muestran cómo hacer de manera eficiente selecciones aleatorias de los iteradores combinatorios en el módulo **`itertools`**:

```
import random

def random_product(*iterables, repeat=1):
    "Random selection from itertools.product(*iterables,
    pools = tuple(map(tuple, iterables)) * repeat
    return tuple(map(random.choice, pools))

def random_permutation(iterable, r=None):
    "Random selection from itertools.permutations(iterab
    pool = tuple(iterable)
    r = len(pool) if r is None else r
    return tuple(random.sample(pool, r))
```

```
def random_combination(iterable, r):
    "Random selection from itertools.combinations(iterab
    pool = tuple(iterable)
    n = len(pool)
    indices = sorted(random.sample(range(n), r))
    return tuple(pool[i] for i in indices)

def random_combination_with_replacement(iterable, r):
    "Choose r elements with replacement.  Order the resu
    # Result will be in set(itertools.combinations_with_
    pool = tuple(iterable)
    n = len(pool)
    indices = sorted(random.choices(range(n), k=r))
    return tuple(pool[i] for i in indices)

def random_derangement(iterable):
    "Choose a permutation where no element stays in its
    seq = tuple(iterable)
    if len(seq) < 2:
        if not seq:
            return ()
        raise IndexError('No derangments to choose from'
    perm = list(range(len(seq)))
    start = tuple(perm)
    while True:
        random.shuffle(perm)
        if all(p != q for p, q in zip(start, perm)):
            return tuple([seq[i] for i in perm])
```

La función **`random()`** por defecto devuelve múltiplos de 2^{-53} en el rango $0.0 \le x < 1.0$. Todos estos números están espaciados uniformemente y son representables exactamente como flotantes de Python. Sin embargo, muchos otros flotadores representables en ese intervalo no son selecciones posibles. Por ejemplo, `0.05954861408025609` no es un múltiplo entero de 2^{-53}.

La siguiente receta adopta un enfoque diferente. Todos los flotantes en el intervalo son selecciones posibles. La mantisa proviene de una distribución uniforme de enteros en el rango $2^{52} \leq mantisa < 2^{53}$. El exponente proviene de una distribución geométrica en la que los exponentes menores de -53 ocurren con la mitad de frecuencia que el siguiente exponente mayor.

```
from random import Random
from math import ldexp

class FullRandom(Random):

    def random(self):
        mantissa = 0x10_0000_0000_0000 | self.getrandbit
        exponent = -53
        x = 0
        while not x:
            x = self.getrandbits(32)
            exponent += x.bit_length() - 32
        return ldexp(mantissa, exponent)
```

Todas las distribuciones de valor real de la clase utilizarán el nuevo método:

```
>>> fr = FullRandom()
>>> fr.random()
0.05954861408025609
>>> fr.expovariate(0.25)
8.87925541791544
```

La receta es conceptualmente equivalente a un algoritmo que elige entre todos los múltiplos de 2^{-1074} en el rango $0{,}0 \leq x < 1{,}0$. Todos esos números son uniformes, pero la mayoría tienen que ser redondeados al flotante de Python representable más cercano. (El valor 2^{-1074} es el menor flotante positivo no normalizado y es igual a `math.ulp(0.0)`.)

Ver también

Generating Pseudo-random Floating-Point Values [https://allendowney.com/research/rand/downey07randfloat.pdf] un artículo de Allen B. Downey en el que se describen formas de generar

flotantes más refinados que los generados normalmente por `random()`.

Command-line usage

Added in version 3.13.

The **random** module can be executed from the command line.

```
python -m random [-h] [-c CHOICE [CHOICE ...] | -i N | -
```

The following options are accepted:

-h, --help

Show the help message and exit.

-c CHOICE [CHOICE ...]
--choice CHOICE [CHOICE ...]

Print a random choice, using `choice()`.

-i <N>
--integer <N>

Print a random integer between 1 and N inclusive, using `randint()`.

-f <N>
--float <N>

Print a random floating-point number between 0 and N inclusive, using `uniform()`.

If no options are given, the output depends on the input:

- String or multiple: same as `--choice`.
- Integer: same as `--integer`.
- Float: same as `--float`.

Command-line example

Here are some examples of the **`random`** command-line interface:

```
$ # Choose one at random
$ python -m random egg bacon sausage spam "Lobster Therm
Lobster Thermidor aux crevettes with a Mornay sauce

$ # Random integer
$ python -m random 6
6

$ # Random floating-point number
$ python -m random 1.8
1.7080016272295635

$ # With explicit arguments
$ python  -m random --choice egg bacon sausage spam "Lob
egg

$ python -m random --integer 6
3

$ python -m random --float 1.8
1.5666339105010318

$ python -m random --integer 6
5

$ python -m random --float 6
3.1942323316565915
```

`statistics` — Mathematical statistics functions

Added in version 3.4.

Código fuente: Lib/statistics.py [https://github.com/python/cpython/tree/3.14/Lib/statistics.py]

Este módulo proporciona funciones para calcular estadísticas matemáticas de datos numéricos (de tipo **`Real`**).

Este módulo no pretende ser competidor o sustituto de bibliotecas de terceros como NumPy [https://numpy.org] o SciPy [https://scipy.org/], ni de paquetes completos de software propietario para estadistas profesionales como Minitab, SAS o Matlab. Este módulo se ubica a nivel de calculadoras científicas y gráficas.

A menos que se indique explícitamente lo contrario, las funciones de este módulo manejan objetos **`int`**, **`float`**, **`Decimal`** y **`Fraction`**. No se garantiza un correcto funcionamiento con otros tipos (numéricos o no). El comportamiento de estas funciones con colecciones mixtas que contengan objetos de diferente tipo no está definido y depende de la implementación. Si tus datos de entrada consisten en una mezcla de varios tipos, puedes usar **`map()`** para asegurarte de que el resultado sea consistente, por ejemplo: `map(float, input_data)`.

Algunos conjuntos de datos utilizan valores `NaN` (no es un número) para representar los datos que faltan. Dado que los valores NaN tienen una semántica de comparación inusual, provocan comportamientos sorprendentes o indefinidos en las funciones estadísticas que ordenan los datos o que cuentan las ocurrencias. Las funciones afectadas son `median()`, `median_low()`, `median_high()`, `median_grouped()`, `mode()`, `multimode()`, y `quantiles()`. Los valores `NaN` deben eliminarse antes de llamar a estas funciones:

```
>>> from statistics import median
>>> from math import isnan
>>> from itertools import filterfalse

>>> data = [20.7, float('NaN'),19.2, 18.3, float('NaN'),
>>> sorted(data)  # This has surprising behavior
[20.7, nan, 14.4, 18.3, 19.2, nan]
>>> median(data)  # This result is unexpected
16.35

>>> sum(map(isnan, data))    # Number of missing values
2
>>> clean = list(filterfalse(isnan, data))  # Strip NaN
>>> clean
[20.7, 19.2, 18.3, 14.4]
>>> sorted(clean)  # Sorting now works as expected
[14.4, 18.3, 19.2, 20.7]
>>> median(clean)       # This result is now well define
18.75
```

Promedios y medidas de tendencia central

Estas funciones calculan el promedio o el valor típico de una población o muestra.

`mean()`	Media aritmética («promedio») de los datos.
`fmean()`	Fast, floating-point arithmetic mean, with optional weighting.
`geometric_mean()`	Media geométrica de los datos.
`harmonic_mean()`	Media armónica de los datos.
`kde()`	Estimate the probability density distribution of the data.
`kde_random()`	Random sampling from the PDF generated by kde().
`median()`	Mediana (valor central) de los datos.
`median_low()`	Mediana baja de los datos.
`median_high()`	Mediana alta de los datos.
`median_grouped()`	Median (50th percentile) of grouped data.
`mode()`	Moda única (valor más común) de datos discretos o nominales.
`multimode()`	Lista de modas (valores más comunes) de datos discretos o

	nominales.
`quantiles()`	Divide los datos en intervalos equiprobables.

Medidas de dispersión

Estas funciones calculan una medida de cuánto tiende a desviarse la población o muestra de los valores típicos o promedios.

`pstdev()`	Desviación típica poblacional de los datos.
`pvariance()`	Varianza poblacional de los datos.
`stdev()`	Desviación típica muestral de los datos.
`variance()`	Varianza muestral de los datos.

Estadísticas para relaciones entre dos entradas

Estas funciones calculan estadísticas sobre las relaciones entre dos entradas.

`covariance()`	Covarianza muestral de dos variables.
`correlation()`	Coeficiente de correlación de Pearson y Spearman.
`linear_regression()`	Pendiente e intersección para regresión lineal simple.

Detalles de las funciones

Nota: Las funciones no requieren que se ordenen los datos que se les proporcionan. Sin embargo, para facilitar la lectura, la mayoría de los ejemplos muestran secuencias ordenadas.

statistics.mean(*data*)

> Retorna la media aritmética muestral de *data*, que puede ser una secuencia o un iterable.
>
> La media aritmética es la suma de los valores dividida entre el número de observaciones. Es comúnmente denominada «promedio», aunque hay muchas formas de definir el promedio matemáticamente. Es una medida de tendencia central de los datos.

Se lanza una excepción **`StatisticsError`** si *data* está vacío.

Algunos ejemplos de uso:

```
>>> mean([1, 2, 3, 4, 4])
2.8
>>> mean([-1.0, 2.5, 3.25, 5.75])
2.625

>>> from fractions import Fraction as F
>>> mean([F(3, 7), F(1, 21), F(5, 3), F(1, 3)])
Fraction(13, 21)

>>> from decimal import Decimal as D
>>> mean([D("0.5"), D("0.75"), D("0.625"), D("0.375
Decimal('0.5625')
```

Nota

La media se ve muy afectada por los outliers [https://en.wikipedia.org/wiki/Outlier] y no es necesariamente un ejemplo típico de los puntos de datos. Para una medida más robusta, aunque menos eficiente, de la tendencia central [https://en.wikipedia.org/wiki/Central_tendency], véase **`median()`**.

La media muestral proporciona una estimación no sesgada de la media real de la población. Por lo tanto, al calcular el promedio de todas las muestras posibles, `mean(sample)` converge con el promedio real de toda la población. Si *data* representa a una población completa, en lugar de a una muestra, entonces `mean(data)` equivale a calcular la media poblacional verdadera μ.

statistics.fmean(*data*, *weights=None*)

Convierte los valores de *data* a flotantes y calcula la media aritmética.

Esta función se ejecuta más rápido que `mean()` y siempre retorna un `float`. *data* puede ser una secuencia o un iterable. Si el conjunto de datos de entrada está vacío, se lanza una excepción `StatisticsError`.

```
>>> fmean([3.5, 4.0, 5.25])
4.25
```

La ponderación es opcional. Por ejemplo, un profesor asigna una nota a un curso ponderando las pruebas en un 20%, los deberes en un 20%, un examen parcial en un 30% y un examen final en un 30%:

```
>>> grades = [85, 92, 83, 91]
>>> weights = [0.20, 0.20, 0.30, 0.30]
>>> fmean(grades, weights)
87.6
```

Si se proporciona *weights*, debe tener la misma longitud que los *data* o se producirá un `ValueError`.

Added in version 3.8.

Distinto en la versión 3.11: Soporte añadido a *weights*.

statistics.geometric_mean(*data*)

Convierte los valores de *data* a flotantes y calcula la media geométrica.

La media geométrica indica la tendencia central o valor típico de *data* utilizando el producto de los valores (en oposición a la media aritmética, que utiliza su suma).

Lanza una excepción `StatisticsError` si el conjunto de datos de entrada está vacío, o si contiene un cero o un valor negativo. *data* puede ser una secuencia o un iterable.

No se toman medidas especiales para garantizar que el resultado sea completamente preciso. (Sin embargo, esto puede cambiar en una versión futura.)

```
>>> round(geometric_mean([54, 24, 36]), 1)
36.0
```

Added in version 3.8.

statistics.harmonic_mean(*data*, *weights=None*)

Return the harmonic mean of *data*, a sequence or iterable of real-valued numbers. If *weights* is omitted or `None`, then equal weighting is assumed.

La media armónica es recíproco de la **`mean()`** aritmética de los recíprocos de los datos. Por ejemplo, la media armónica de tres valores *a*, *b* and *c* es equivalente a `3/(1/a + 1/b + 1/c)`. Si alguno de los valores es cero, el resultado va a ser cero.

La media armónica es un tipo de promedio, una medida de la tendencia central de los datos. Generalmente es adecuada para calcular promedios de tasas o fracciones, por ejemplo, velocidades.

Supongamos que un automóvil viaja 10 km a 40 km/h, luego otros 10 km a 60 km/h. ¿Cuál es su velocidad media?

```
>>> harmonic_mean([40, 60])
48.0
```

Supongamos que un un automóvil viaja 5 km a 40 km/h, y cuando el tráfico se despeja, acelera a 60 km/h durante los 30 km restantes del viaje. ¿Cuál es su velocidad media?

```
>>> harmonic_mean([40, 60], weights=[5, 30])
56.0
```

Una excepción **`StatisticsError`** es lanzada si *data* está vacío, algún elemento es menor que cero, o si la suma ponderada no es positiva.

El algoritmo actual tiene una salida anticipada cuando encuentra un cero en la entrada. Esto significa que no se comprueba la validez de las entradas posteriores al cero. (Este

comportamiento puede cambiar en el futuro.)

Added in version 3.6.

Distinto en la versión 3.10: Soporte añadido a *weights*.

statistics.kde(*data*, *h*, *kernel='normal'*, ***, *cumulative=False*)

Kernel Density Estimation (KDE) [https://www.itm-conferences.org/articles/itmconf/pdf/2018/08/itmconf_sam2018_00037.pdf]: Create a continuous probability density function or cumulative distribution function from discrete samples.

The basic idea is to smooth the data using a kernel function [https://en.wikipedia.org/wiki/Kernel_(statistics)]. to help draw inferences about a population from a sample.

The degree of smoothing is controlled by the scaling parameter *h* which is called the bandwidth. Smaller values emphasize local features while larger values give smoother results.

The *kernel* determines the relative weights of the sample data points. Generally, the choice of kernel shape does not matter as much as the more influential bandwidth smoothing parameter.

Kernels that give some weight to every sample point include *normal* (*gauss*), *logistic*, and *sigmoid*.

Kernels that only give weight to sample points within the bandwidth include *rectangular* (*uniform*), *triangular*, *parabolic* (*epanechnikov*), *quartic* (*biweight*), *triweight*, and *cosine*.

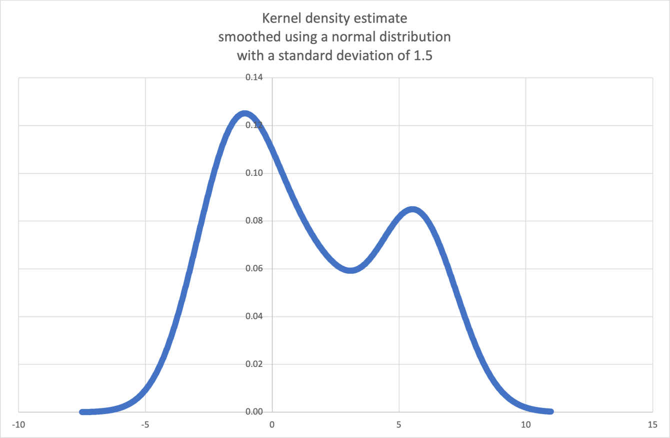
If *cumulative* is true, will return a cumulative distribution function.

A `StatisticsError` will be raised if the *data* sequence is empty.

Wikipedia has an example [https://en.wikipedia.org/wiki/Kernel_density_estimation#Example] where we can use `kde()` to generate and plot a probability density function estimated from a small sample:

```
>>> sample = [-2.1, -1.3, -0.4, 1.9, 5.1, 6.2]
>>> f_hat = kde(sample, h=1.5)
>>> xarr = [i/100 for i in range(-750, 1100)]
>>> yarr = [f_hat(x) for x in xarr]
```

Los puntos de `xarr` y `yarr` pueden utilizarse para hacer una gráfica de la función de densidad de probabilidad:



Added in version 3.13.

statistics.kde_random(_data_, _h_, _kernel='normal'_, *, _seed=None_)

Return a function that makes a random selection from the estimated probability density function produced by `kde(data, h, kernel)`.

Providing a *seed* allows reproducible selections. In the future, the values may change slightly as more accurate kernel inverse CDF estimates are implemented. The seed may be an integer, float, str, or bytes.

A **`StatisticsError`** will be raised if the *data* sequence is empty.

Continuing the example for **`kde()`**, we can use **`kde_random()`** to generate new random selections from an

estimated probability density function:

```
>>> data = [-2.1, -1.3, -0.4, 1.9, 5.1, 6.2]
>>> rand = kde_random(data, h=1.5, seed=8675309)
>>> new_selections = [rand() for i in range(10)]
>>> [round(x, 1) for x in new_selections]
[0.7, 6.2, 1.2, 6.9, 7.0, 1.8, 2.5, -0.5, -1.8, 5.6
```

Added in version 3.13.

statistics.median(*data*)

Retorna la mediana (valor central) de los datos numéricos, utilizando el método clásico de «media de los dos del medio». Si *data* está vacío, se lanza una excepción **`StatisticsError`**. *data* puede ser una secuencia o un iterable.

La mediana es una medida de tendencia central robusta y es menos sensible a la presencia de valores atípicos que la media. Cuando el número de casos es impar, se retorna el valor central:

```
>>> median([1, 3, 5])
3
```

Cuando el número de observaciones es par, la mediana se interpola calculando el promedio de los dos valores centrales:

```
>>> median([1, 3, 5, 7])
4.0
```

Este enfoque es adecuado para datos discretos, siempre que se acepte que la mediana no es necesariamente parte de las observaciones.

Si los datos son ordinales (se pueden ordenar) pero no numéricos (no se pueden sumar), considera usar **`median_low()`** o **`median_high()`** en su lugar.

statistics.median_low(*data*)

Retorna la mediana baja de los datos numéricos. Se lanza una excepción **`StatisticsError`** si *data* está vacío. *data* puede ser una secuencia o un iterable.

La mediana baja es siempre un valor presente en el conjunto de datos. Cuando el número de casos es impar, se retorna el valor central. Cuando el número de casos es par, se retorna el menor de los dos valores centrales.

```
>>> median_low([1, 3, 5])
3
>>> median_low([1, 3, 5, 7])
3
```

Utiliza la mediana baja cuando tus datos sean discretos y prefieras que la mediana sea un valor representado en tus observaciones, en lugar de ser el resultado de una interpolación.

statistics.median_high(*data*)

Retorna la mediana alta de los datos. Lanza una excepción **`StatisticsError`** si *data* está vacío. *data* puede ser una secuencia o un iterable.

La mediana alta es siempre un valor presente en el conjunto de datos. Cuando el número de casos es impar, se retorna el valor central. Cuando el número de casos es par, se retorna el mayor de los dos valores centrales.

```
>>> median_high([1, 3, 5])
3
>>> median_high([1, 3, 5, 7])
5
```

Utiliza la mediana alta cuando tus datos sean discretos y prefieras que la mediana sea un valor representado en tus observaciones, en lugar de ser el resultado de una interpolación.

statistics.median_grouped(*data*, *interval=1.0*)

Estimates the median for numeric data that has been grouped or binned [https://en.wikipedia.org/wiki/Data_binning] around the midpoints of consecutive, fixed-width intervals.

The *data* can be any iterable of numeric data with each value being exactly the midpoint of a bin. At least one value must be present.

The *interval* is the width of each bin.

For example, demographic information may have been summarized into consecutive ten-year age groups with each group being represented by the 5-year midpoints of the intervals:

```
>>> from collections import Counter
>>> demographics = Counter({
...    25: 172,   # 20 to 30 years old
...    35: 484,   # 30 to 40 years old
...    45: 387,   # 40 to 50 years old
...    55:  22,   # 50 to 60 years old
...    65:   6,   # 60 to 70 years old
... })
...
```

The 50th percentile (median) is the 536th person out of the 1071 member cohort. That person is in the 30 to 40 year old age group.

The regular `median()` function would assume that everyone in the tricenarian age group was exactly 35 years old. A more tenable assumption is that the 484 members of that age group are evenly distributed between 30 and 40. For that, we use **`median_grouped()`**:

```
>>> data = list(demographics.elements())
>>> median(data)
35
>>> round(median_grouped(data, interval=10), 1)
37.5
```

The caller is responsible for making sure the data points are separated by exact multiples of *interval*. This is essential for getting a correct result. The function does not check this precondition.

Inputs may be any numeric type that can be coerced to a float during the interpolation step.

statistics.mode(*data*)

Retorna el valor más común del conjunto de datos discretos o nominales *data*.La moda (cuando existe) es el valor más representativo y sirve como medida de tendencia central.

Si hay varias modas con la misma frecuencia, retorna la primera encontrada en *data*. Si deseas la menor o la mayor de ellas, usa `min(multimode(data))` o `max(multimode(data))`. Se lanza una excepción **`StatisticsError`** si la entrada *data* está vacía.

`mode` asume que los datos de entrada son discretos y retorna un solo valor. Esta es la definición habitual de la moda que se enseña en las escuelas:

```
>>> mode([1, 1, 2, 3, 3, 3, 3, 4])
3
```

La moda tiene la particularidad de ser la única estadística de este módulo que se puede calcular sobre datos nominales (no numéricos):

```
>>> mode(["red", "blue", "blue", "red", "green", "r
'red'
```

Only hashable inputs are supported. To handle type **`set`**, consider casting to **`frozenset`**. To handle type **`list`**, consider casting to **`tuple`**. For mixed or nested inputs, consider using this slower quadratic algorithm that only depends on equality tests: `max(data, key=data.count)`.

Distinto en la versión 3.8: Ahora maneja conjuntos de datos multimodales, retornando la primera moda encontrada.

Anteriormente, se lanzaba una excepción **`StatisticsError`** cuando se daba esta situación.

statistics.multimode(*data*)

Retorna una lista de los valores más frecuentes en el orden en que aparecen en *data*. Retornará varios resultados en el caso de que existan varias modas, o una lista vacía si *data* está vacío:

```
>>> multimode('aabbbbccddddeeffffgg')
['b', 'd', 'f']
>>> multimode('')
[]
```

Added in version 3.8.

statistics.pstdev(*data*, *mu=None*)

Retorna la desviación típica poblacional (la raíz cuadrada de la varianza poblacional). Consultar **`pvariance()`** para los argumentos y otros detalles.

```
>>> pstdev([1.5, 2.5, 2.5, 2.75, 3.25, 4.75])
0.986893273527251
```

statistics.pvariance(*data*, *mu=None*)

Retorna la varianza poblacional de *data*, que debe ser una secuencia no vacía o un iterable de números reales. La varianza, o momento de segundo orden respecto a la media, es una medida de la variabilidad (o dispersión) de los datos. Una alta varianza indica una amplia dispersión de valores; una varianza baja indica que los valores están agrupados alrededor de la media.

If the optional second argument *mu* is given, it should be the *population* mean of the *data*. It can also be used to compute the second moment around a point that is not the mean. If it is missing or `None` (the default), the arithmetic mean is automatically calculated.

Utiliza esta función para calcular la varianza de toda la población. Para estimar la varianza de una muestra, la función `variance()` suele ser una opción mejor.

Lanza una excepción `StatisticsError` si *data* está vacío.

Ejemplos:

```
>>> data = [0.0, 0.25, 0.25, 1.25, 1.5, 1.75, 2.75,
>>> pvariance(data)
1.25
```

Si ya has calculado la media de tus datos, puedes pasarla como segundo argumento opcional *mu* para evitar que se tenga que volver a calcular:

```
>>> mu = mean(data)
>>> pvariance(data, mu)
1.25
```

Se admiten decimales (Decimal) y fracciones (Fraction):

```
>>> from decimal import Decimal as D
>>> pvariance([D("27.5"), D("30.25"), D("30.25"), D
Decimal('24.815')

>>> from fractions import Fraction as F
>>> pvariance([F(1, 4), F(5, 4), F(1, 2)])
Fraction(13, 72)
```

Nota

Esta función retorna la varianza poblacional σ^2 cuando se aplica a toda la población. Si se aplica solo a una muestra, el resultado es la varianza muestral s^2, conocida también como varianza con N grados de libertad.

Si se conoce de antemano la verdadera media poblacional μ, se puede usar esta función para calcular la varianza muestral, pasando la media poblacional conocida como segundo argumento. Suponiendo que las observaciones provienen de una selección aleatoria uniforme de la población, el resultado será una estimación no sesgada de la varianza poblacional.

statistics.stdev(*data*, *xbar=None*)

Retorna la desviación típica muestral (la raíz cuadrada de la varianza muestral). Consultar `variance()` para los argumentos y otros detalles.

```
>>> stdev([1.5, 2.5, 2.5, 2.75, 3.25, 4.75])
1.0810874155219827
```

statistics.variance(*data*, *xbar=None*)

Retorna la varianza muestral de *data*, que debe ser un iterable de al menos dos números reales. La varianza, o momento de segundo orden respecto a la media, es una medida de la variabilidad (difusión o dispersión) de los datos. Una alta varianza indica que los datos están dispersos; una baja varianza indica que los datos están agrupados estrechamente alrededor de la media.

If the optional second argument *xbar* is given, it should be the *sample* mean of *data*. If it is missing or `None` (the default), the mean is automatically calculated.

Utiliza esta función cuando tus datos sean una muestra de una población. Para calcular la varianza de toda la población, consulta `pvariance()`.

Lanza una excepción `StatisticsError` si *data* tiene menos de dos valores.

Ejemplos:

```
>>> data = [2.75, 1.75, 1.25, 0.25, 0.5, 1.25, 3.5]
>>> variance(data)
1.3720238095238095
```

If you have already calculated the sample mean of your data, you can pass it as the optional second argument *xbar* to avoid recalculation:

```
>>> m = mean(data)
>>> variance(data, m)
1.3720238095238095
```

Esta función no comprueba si el valor pasado al argumento *xbar* corresponde al promedio. El uso de valores arbitrarios para *xbar* produce resultados imposibles o incorrectos.

La función maneja decimales (Decimal) y fracciones (Fraction):

```
>>> from decimal import Decimal as D
>>> variance([D("27.5"), D("30.25"), D("30.25"), D(
Decimal('31.01875')

>>> from fractions import Fraction as F
>>> variance([F(1, 6), F(1, 2), F(5, 3)])
Fraction(67, 108)
```

Nota

Esta es la varianza muestral s^2 con la corrección de Bessel, también conocida como varianza con N-1 grados de libertad. Suponiendo que las observaciones son representativas de la población (es decir, independientes y distribuidas de forma idéntica), el resultado es una estimación no sesgada de la varianza.

Si conoces de antemano la verdadera media poblacional µ, debes pasarla a **`pvariance()`** mediante el parámetro *mu* para obtener la varianza muestral.

statistics.quantiles(*data*, ***, *n=4*, *method='exclusive'*)

Divide *data* en *n* intervalos continuos equiprobables. Retorna una lista de `n - 1` límites que delimitan los intervalos (cuantiles).

Establece *n* en 4 para obtener los cuartiles (el valor predeterminado), en 10 para obtener los deciles y en 100 para obtener los percentiles (lo que produce 99 valores que separan *data* en 100 grupos del mismo tamaño). Si *n* es menor que 1, se lanza una excepción **`StatisticsError`**.

The *data* can be any iterable containing sample data. For meaningful results, the number of data points in *data* should be larger than *n*. Raises **`StatisticsError`** if there is not at least one data point.

Los límites de los intervalos se interpolan linealmente a partir de los dos valores más cercanos de la muestra. Por ejemplo, si un límite es un tercio de la distancia entre los valores 100 y 112 de la muestra, el límite será 104.

El argumento *method* indica el método que se utilizará para calcular los cuantiles y se puede modificar para especificar si se deben incluir o excluir valores de *data* extremos, altos y bajos, de la población.

El valor predeterminado para *method* es «exclusive» y es aplicable a los datos muestreados de una población que puede tener valores más extremos que los encontrados en las muestras. La proporción de la población que se encuentra por debajo del *i-ésimo* valor de *m* valores ordenados se calcula mediante la fórmula `i / (m + 1)`. Por ejemplo, asumiendo que hay 9 valores en la muestra, este método los ordena y los asocia con los siguientes percentiles: 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%.

Si se usa «inclusive» como valor para el parámetro *method*, se asume que los datos corresponden a una población completa o que los valores extremos de la población están representados en la muestra. El valor mínimo de *data* se considera entonces como percentil 0 y el máximo como percentil 100. La proporción de la población que se encuentra por debajo del *i-ésimo* valor de *m* valores ordenados se calcula mediante la fórmula `(i - 1) / (m - 1)`. Suponiendo que tenemos 11 valores en la muestra, este método los ordena y los asocia con los siguientes percentiles: 0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80 %, 90%, 100%.

```
# Decile cut points for empirically sampled data
>>> data = [105, 129, 87, 86, 111, 111, 89, 81, 108
...          100, 75, 105, 103, 109, 76, 119, 99, 91
...          106, 101, 84, 111, 74, 87, 86, 103, 103
...          111, 75, 87, 102, 121, 111, 88, 89, 101
...          103, 107, 101, 81, 109, 104]
>>> [round(q, 1) for q in quantiles(data, n=10)]
[81.0, 86.2, 89.0, 99.4, 102.5, 103.6, 106.0, 109.8
```

Added in version 3.8.

Distinto en la versión 3.13: No longer raises an exception for an input with only a single data point. This allows quantile estimates to be built up one sample point at a time becoming gradually more refined with each new data point.

statistics.covariance(*x*, *y*, /)

Retorna la covarianza de muestra de dos entradas *x* y *y*. La covarianza es una medida de la variabilidad conjunta de dos entradas.

Ambas entradas deben ser del mismo largo (no menor a dos), de lo contrario se lanza **`StatisticsError`**.

Ejemplos:

```
>>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> y = [1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> covariance(x, y)
0.75
>>> z = [9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> covariance(x, z)
-7.5
>>> covariance(z, x)
-7.5
```

Added in version 3.10.

statistics.correlation(*x*, *y*, /, *, *method='linear'*)

Retorna el coeficiente de correlación de Pearson [https://es.wikipedia.org/wiki/Coeficiente_de_correlaci%C3%B3n_de_Spearman] para dos entradas. El coeficiente de correlación de Pearson *r* toma valores entre -1 y +1. Mide la fuerza y dirección de la relación lineal.

Si *method* es «*ranked*», calcula El coeficiente de correlación de Spearman [https://es.wikipedia.org/wiki/Coeficiente_de_correlaci%C3%B3n_de_Spearman] para dos entradas. Los datos se sustituyen por rangos. Los empates se promedian para que valores iguales reciban el mismo rango. El coeficiente resultante mide la fuerza de una relación monótona.

El coeficiente de correlación de Spearman es apropiado para datos ordinales o para datos continuos que no cumplen el requisito de proporción lineal para el coeficiente de correlación de Pearson.

Ambas entradas deben ser del mismo largo (no menor a dos), y no necesitan ser constantes, de lo contrario se lanza **`StatisticsError`**.

Ejemplo con "Leyes de Kepler sobre el movimiento planetario <https://es.wikipedia.org/wiki/Leyes_de_Kepler'_ [https://es.wikipedia.org/wiki/Leyes_de_Kepler'_]:

```
>>> # Mercury, Venus, Earth, Mars, Jupiter, Saturn,
>>> orbital_period = [88, 225, 365, 687, 4331, 10_7
>>> dist_from_sun = [58, 108, 150, 228, 778, 1_400,

>>> # Show that a perfect monotonic relationship ex
>>> correlation(orbital_period, dist_from_sun, meth
1.0

>>> # Observe that a linear relationship is imperfe
>>> round(correlation(orbital_period, dist_from_sun
0.9882
```

```
>>> # Demonstrate Kepler's third law: There is a li
>>> # between the square of the orbital period and 
>>> # distance from the sun.
>>> period_squared = [p * p for p in orbital_period
>>> dist_cubed = [d * d * d for d in dist_from_sun]
>>> round(correlation(period_squared, dist_cubed), 
1.0
```

Added in version 3.10.

Distinto en la versión 3.12: Soporte añadido para el coeficiente de correlación de Spearman.

statistics.linear_regression(*x*, *y*, /, *, *proportional=False*)

Retorna la pendiente e intersección de los parámetros de una regresión lineal simple [https://en.wikipedia.org/wiki/Simple_linear_regression] estimados usando mínimos cuadrados ordinarios. La regresión lineal simple describe la relación entre la variable independiente *x* y la variable dependiente *y* en términos de esta función lineal:

$$y = slope * x + intercept + noise$$

donde `slope` e `intercept` son los parámetros de regresión que son estimados, y `noise` representa la variabilidad de los datos que no fue explicado por la regresión lineal (es igual a la diferencia entre los valores predichos y reales de la variable dependiente).

Ambas entradas deben ser del mismo largo (no menor a dos), y la variable independiente *x* no puede ser constante, de lo contrario se lanza **`StatisticsError`**.

Por ejemplo, podemos usar las fechas de lanzamiento de las películas de Monty Python [https://en.wikipedia.org/wiki/Monty_Python#Films] para predecir el número acumulativo de películas de Monty Python que se habría producido en 2019 asumiendo que hubiesen mantenido el ritmo:

```
>>> year = [1971, 1975, 1979, 1982, 1983]
>>> films_total = [1, 2, 3, 4, 5]
>>> slope, intercept = linear_regression(year, film
>>> round(slope * 2019 + intercept)
16
```

Si *proportional* es verdadero, se supone que la variable independiente *x* y la variable dependiente *y* son directamente proporcionales. Los datos se ajustan a una recta que pasa por el origen. Como la *intercept* siempre será 0,0, la función lineal subyacente se simplifica a:

$$y = slope * x + noise$$

Continuing the example from `correlation()`, we look to see how well a model based on major planets can predict the orbital distances for dwarf planets:

```
>>> model = linear_regression(period_squared, dist_
>>> slope = model.slope

>>> # Dwarf planets:   Pluto,  Eris,    Makemake, H
>>> orbital_periods = [90_560, 204_199, 111_845, 10
>>> predicted_dist = [math.cbrt(slope * (p * p)) fo
>>> list(map(round, predicted_dist))
[5912, 10166, 6806, 6459, 414]

>>> [5_906, 10_152, 6_796, 6_450, 414]  # actual di
[5906, 10152, 6796, 6450, 414]
```

Added in version 3.10.

Distinto en la versión 3.11: Soporte añadido a *proportional*.

Excepciones

Se define una sola excepción:

exception statistics.StatisticsError

Subclase de `ValueError` para excepciones relacionadas con la estadística.

Objetos `NormalDist`

`NormalDist` es una herramienta para crear y manipular distribuciones normales de una variable aleatoria [http://www.stat.yale.edu/Courses/1997-98/101/ranvar.htm]. Esta clase gestiona la desviación típica y la media de un conjunto de observaciones como una sola entidad.

Las distribuciones normales surgen del Teorema del límite central [https://es.wikipedia.org/wiki/Teorema_del_l%C3%ADmite_central] y tienen una amplia gama de aplicaciones en estadística.

class statistics.NormalDist(*mu=0.0, sigma=1.0*)

Retorna un nuevo objeto *NormalDist* donde *mu* representa la media aritmética [https://es.wikipedia.org/wiki/Media_aritm%C3%A9tica] y *sigma* representa la desviación típica [https://es.wikipedia.org/wiki/Desviaci%C3%B3n_t%C3%ADpica].

Se lanza una excepción `StatisticsError` si *sigma* es negativo.

mean

Una propiedad de solo lectura para la media aritmética [https://es.wikipedia.org/wiki/Media_aritm%C3%A9tica] de una distribución normal.

median

Una propiedad de solo lectura para la mediana [https://es.wikipedia.org/wiki/Mediana_(estad%C3%ADstica)] de una distribución normal.

mode

Una propiedad de solo lectura para la moda [https://es.wikipedia.org/wiki/Moda_(estad%C3%ADstica)] de una distribución normal.

stdev

Una propiedad de solo lectura para la desviación típica [https://es.wikipedia.org/wiki/Desviaci%C3%B3n_t%C3%ADpica] de una distribución normal.

variance

Una propiedad de solo lectura para la varianza [https://es.wikipedia.org/wiki/Varianza] de una distribución normal. Es igual al cuadrado de la desviación típica.

classmethod from_samples(*data*)

Crea una instancia de distribución normal con los parámetros *mu* y *sigma* estimados a partir de *data* usando `fmean()` y `stdev()`.

data puede ser cualquier iterable de valores que se puedan convertir al tipo `float`. Se lanza una excepción `StatisticsError` si *data* no contiene al menos dos elementos, esto se debe a que se necesita al menos un punto para estimar un valor central y al menos dos puntos para estimar la dispersión.

samples(*n*, ***, *seed=None*)

Genera *n* muestras aleatorias para una media y una desviación típica proporcionadas. Retorna un objeto `list` de valores `float`.

Si se proporciona *seed*, su valor se usa para inicializar una nueva instancia del generador de números aleatorios subyacente. Esto permite producir resultados reproducibles incluso en un contexto de paralelismo con múltiples hilos.

Distinto en la versión 3.13.

Switched to a faster algorithm. To reproduce samples from previous versions, use `random.seed()` and `random.gauss()`.

pdf(*x*)

Haciendo uso de una función de densidad de probabilidad (FPD o PDF en inglés) [https://es.wikipedia.org/wiki/Funci%C3%B3n_de_densidad_de_probabilidad], calcula la verosimilitud relativa de que una variable aleatoria *X* caiga en una región cercana al valor *x* proporcionado. Matemáticamente, esto corresponde al límite de la razón `P(x <= X < x+dx) / dx` cuando *dx* tiende a cero.

La verosimilitud relativa se calcula como la probabilidad de que una observación pertenezca a un intervalo estrecho dividida entre el ancho del intervalo (de ahí el término «densidad»). Como la verosimilitud es relativa a los otros puntos, su valor puede ser mayor que `1.0`.

cdf(*x*)

Usando una función de distribución acumulada (FDA, CDF en inglés) [https://es.wikipedia.org/wiki/Funci%C3%B3n_de_distribuci%C3%B3n], calcula la probabilidad de que una variable aleatoria *X* sea menor o igual que *x*. Matemáticamente, se escribe `P(X <= x)`.

inv_cdf(*p*)

Calcula la función de distribución acumulada inversa, también conocida como función cuantil [https://es.wikipedia.org/wiki/Funci%C3%B3n_cuantil] o función punto porcentual [https://web.archive.org/web/20190203145224/https://www.statisticshowto.datasciencecentral.com/inverse-distribution-function/]. Matemáticamente, se escribe `x : P(X <= x) = p`.

Calcula el valor *x* de la variable aleatoria *X* tal que la probabilidad de que la variable sea menor o igual a este valor es igual a la probabilidad *p* dada.

overlap(*other*)

Mide la concordancia entre dos distribuciones de probabilidad normales. Retorna un valor entre 0.0 y 1.0 que indica el área de superposición de dos funciones de densidad de probabilidad [https://www.rasch.org/rmt/rmt101r.htm].

quantiles(*n=4*)

Divide la distribución normal en *n* intervalos continuos equiprobables. Retorna una lista de (n - 1) cuantiles que separan los intervalos.

Establece *n* en 4 para obtener los cuartiles (el valor predeterminado), en 10 para obtener los deciles y en 100 para obtener los percentiles (lo que produce 99 límites que separan los datos en 100 grupos del mismo tamaño).

zscore(*x*)

Computa el Standard Score [https://www.statisticshowto.com/probability-and-statistics/z-score/] describiendo *x* en términos de los números de desviaciones estándar sobre o bajo la media de una distribución normal: `(x - mean) / stdev`.

Added in version 3.9.

Las instancias de la clase **`NormalDist`** soportan la suma, resta, multiplicación y división por una constante. Estas operaciones se pueden utilizar para traducir o escalar, por ejemplo:

```
>>> temperature_february = NormalDist(5, 2.5)
>>> temperature_february * (9/5) + 32
NormalDist(mu=41.0, sigma=4.5)
```

No se admite la división de una constante entre una instancia de **`NormalDist`** debido a que el resultado no sería una distribución normal.

Dado que las distribuciones normales se derivan de las propiedades aditivas de variables independientes, es posible sumar o restar dos variables independientes con distribución normal [https://en.wikipedia.org/wiki/Sum_of_normally_distributed_random_variables] representadas por instancias de **`NormalDist`**. Por ejemplo :

```
>>> birth_weights = NormalDist.from_samples([2.5, 3
>>> drug_effects = NormalDist(0.4, 0.15)
>>> combined = birth_weights + drug_effects
>>> round(combined.mean, 1)
3.1
>>> round(combined.stdev, 1)
0.5
```

Added in version 3.8.

Examples and Recipes

Problemas de probabilidad clásicos

`NormalDist` permite resolver fácilmente problemas probabilísticos clásicos.

Por ejemplo, sabiendo que los datos históricos de los exámenes SAT [https://nces.ed.gov/programs/digest/d17/tables/dt17_226.40.asp] siguen una distribución normal con una media de 1060 y una desviación típica de 195, determinar el porcentaje de estudiantes con puntuaciones entre 1100 y 1200, redondeado al número entero más cercano:

```
>>> sat = NormalDist(1060, 195)
>>> fraction = sat.cdf(1200 + 0.5) - sat.cdf(1100 - 0.5)
>>> round(fraction * 100.0, 1)
18.4
```

Determinar los cuartiles [https://es.wikipedia.org/wiki/Cuartil] y deciles [https://es.wikipedia.org/wiki/Decil_(estad%C3%ADstica)] de las puntuaciones del SAT:

```
>>> list(map(round, sat.quantiles()))
```

```
[928, 1060, 1192]
>>> list(map(round, sat.quantiles(n=10)))
[810, 896, 958, 1011, 1060, 1109, 1162, 1224, 1310]
```

Entradas de Monte Carlo para simulaciones

To estimate the distribution for a model that isn't easy to solve analytically, **`NormalDist`** can generate input samples for a Monte Carlo simulation [https://en.wikipedia.org/wiki/Monte_Carlo_method]:

```
>>> def model(x, y, z):
...     return (3*x + 7*x*y - 5*y) / (11 * z)
...
>>> n = 100_000
>>> X = NormalDist(10, 2.5).samples(n, seed=3652260728)
>>> Y = NormalDist(15, 1.75).samples(n, seed=4582495471)
>>> Z = NormalDist(50, 1.25).samples(n, seed=6582483453)
>>> quantiles(map(model, X, Y, Z))
[1.4591308524824727, 1.8035946855390597, 2.1750914472747
```

Aproximación de la distribución binomial

Las distribuciones normales se pueden utilizar para aproximar distribuciones binomiales [https://mathworld.wolfram.com/BinomialDistribution.html] cuando el tamaño de la muestra es grande y la probabilidad de un ensayo exitoso es cercana al 50%.

Por ejemplo, 750 personas asisten a una conferencia sobre código abierto y se dispone de dos salas con capacidad para 500 personas cada una. En la primera sala hay una charla sobre Python, en la otra una sobre Ruby. En conferencias pasadas, el 65% de las personas prefirieron escuchar las charlas sobre Python. Suponiendo que las preferencias de la población no hayan cambiado, ¿cuál es la probabilidad de que la sala de Python permanezca por debajo de su capacidad máxima?

```
>>> n = 750             # Sample size
>>> p = 0.65            # Preference for Python
>>> q = 1.0 - p         # Preference for Ruby
>>> k = 500             # Room capacity
>>> # Approximation using the cumulative normal distribu
>>> from math import sqrt
>>> round(NormalDist(mu=n*p, sigma=sqrt(n*p*q)).cdf(k +
0.8402

>>> # Exact solution using the cumulative binomial distr
>>> from math import comb, fsum
>>> round(fsum(comb(n, r) * p**r * q**(n-r) for r in ran
0.8402

>>> # Approximation using a simulation
>>> from random import seed, binomialvariate
>>> seed(8675309)
>>> mean(binomialvariate(n, p) <= k for i in range(10_00
0.8406
```

Clasificador bayesiano ingenuo

Las distribuciones normales a menudo están involucradas en el aprendizaje automático.

Wikipedia detalla un buen ejemplo de un clasificador bayesiano ingenuo [https://es.wikipedia.org/wiki/Clasificador_bayesiano_ingenuo]. El reto consiste en predecir el género de una persona a partir de características físicas que siguen una distribución normal, como la altura, el peso y el tamaño del pie.

Disponemos de un conjunto de datos de entrenamiento que contiene las medidas de ocho personas. Se supone que estas medidas siguen una distribución normal, por lo que podemos sintetizar los datos usando **`NormalDist`**:

```
>>> height_male = NormalDist.from_samples([6, 5.92, 5.58
>>> height_female = NormalDist.from_samples([5, 5.5, 5.4
>>> weight_male = NormalDist.from_samples([180, 190, 170
>>> weight_female = NormalDist.from_samples([100, 150, 1
>>> foot_size_male = NormalDist.from_samples([12, 11, 12
>>> foot_size_female = NormalDist.from_samples([6, 8, 7,
```

A continuación, nos encontramos con un nuevo individuo del que conocemos las medidas de sus características pero no su género:

```
>>> ht = 6.0        # height
>>> wt = 130        # weight
>>> fs = 8          # foot size
```

Partiendo de una probabilidad a priori [https://es.wikipedia.org/wiki/Probabilidad_a_priori] del 50% de ser hombre o mujer, calculamos la probabilidad a posteriori como el producto de la probabilidad a priori y la verosimilitud de las diferentes medidas dado el género:

```
>>> prior_male = 0.5
>>> prior_female = 0.5
>>> posterior_male = (prior_male * height_male.pdf(ht) *
...                    weight_male.pdf(wt) * foot_size_ma

>>> posterior_female = (prior_female * height_female.pdf
...                      weight_female.pdf(wt) * foot_siz
```

La predicción final es la que tiene mayor probabilidad a posteriori. Este enfoque se denomina máximo a posteriori [https://en.wikipedia.org/wiki/Maximum_a_posteriori_estimation] o MAP:

```
>>> 'male' if posterior_male > posterior_female else 'fe
'female'
```

Módulos de programación funcional

Los módulos descritos en este capítulo proporcionan funciones y clases que admiten un estilo de programación funcional y operaciones generales invocables (*callables*).

En este capítulo se documentan los siguientes módulos:

- **`itertools`** — Functions creating iterators for efficient looping
    - Itertool Functions
    - Fórmulas con itertools
- **`functools`** — Higher-order functions and operations on callable objects
    - **`partial`** Objetos
- **`operator`** — Standard operators as functions
    - Asignación de operadores a funciones
    - Operadores *In-place*

`itertools` — Functions creating iterators for efficient looping

Este módulo implementa un número de piezas básicas iterator inspiradas en *constructs* de APL, Haskell y SML. Cada pieza ha sido reconvertida a una forma apropiada para Python.

El módulo estandariza un conjunto base de herramientas rápidas y eficientes en memoria, útiles por sí mismas o en combinación con otras. Juntas, forman un «álgebra de iteradores», haciendo posible la construcción de herramientas especializadas, sucintas y eficientes, en Python puro.

Por ejemplo, SML provee una herramienta de tabulación `tabulate(f)`, que produce una secuencia `f(0), f(1), ...`. En Python, se puede lograr el mismo efecto al combinar **`map()`** y **`count()`** para formar `map(f, count())`.

General iterators:

Iterador	Argumentos	Resultados	Ejemplo
`accumulate()`	p [,func]	p0, p0+p1, p0+p1+p2, …	`accumulate([1,2,3,4,5]) → 1 3 6 10 15`
`batched()`	p, n	(p0, p1, …, p_n-1), …	`batched('ABCDEFG', n=3) → ABC DEF G`
`chain()`	p, q, …	p0, p1, … plast, q0, q1, …	`chain('ABC', 'DEF') → A B C D E F`
`chain.from_iterable()`	iterable	p0, p1, … plast, q0, q1, …	`chain.from_iterable(['ABC', 'DEF']) → A B C D E F`
`compress()`	data, selectors	(d[0] if s[0]), (d[1] if s[1]), …	`compress('ABCDEF', [1,0,1,0,1,1]) → A C E F`
`count()`	[start[, step]]	start, start+step, start+2*step, …	`count(10) → 10 11 12 13 14 ...`
`cycle()`	p	p0, p1, … plast, p0, p1, …	`cycle('ABCD') → A B C D A B C D ...`
`dropwhile()`	predicate, seq	seq[n], seq[n+1], starting when predicate fails	`dropwhile(lambda x: x<5, [1,4,6,3,8]) → 6 3 8`
`filterfalse()`	predicate, seq	elements of seq where predicate(elem) fails	`filterfalse(lambda x: x<5, [1,4,6,3,8]) → 6 8`
`groupby()`	iterable[, key]	sub-iteradores agrupados según el valor de key(v)	`groupby(['A','B','DEF'], len) → (1, A B) (3, DEF)`

`islice()`	seq, [start,] stop [, step]	elementos de seq[start:stop:step]	`islice('ABCDEFG', 2, None) → C D E F G`
`pairwise()`	iterable	(p[0], p[1]), (p[1], p[2])	`pairwise('ABCDEFG') → AB BC CD DE EF FG`
`repeat()`	elem [,n]	elem, elem, elem, … indefinidamente o hasta n veces	`repeat(10, 3) → 10 10 10`
`starmap()`	func, seq	func(*seq[0]), func(*seq[1]), …	`starmap(pow, [(2,5), (3,2), (10,3)]) → 32 9 1000`
`takewhile()`	pred, seq	seq[0], seq[1], until predicate fails	`takewhile(lambda x: x<5, [1,4,6,3,8]) → 1 4`
`tee()`	it, n	it1, it2, … itn divide un iterador en n	`tee('ABC', 2) → A B C A B C`
`zip_longest()`	p, q, …	(p[0], q[0]), (p[1], q[1]), …	`zip_longest('ABCD', 'xy', fillvalue='-') → Ax By C- D-`

Iteradores combinatorios:

Iterador	Argumentos	Resultados
`product()`	p, q, … [repeat=1]	producto cartesiano, equivalente a un bucle *for* anidado
`permutations()`	p[, r]	tuplas de longitud r, en todas los órdenes posibles, sin elementos repetidos
`combinations()`	p, r	tuplas de longitud r, ordenadas, sin elementos repetidos
`combinations_with_replacement()`	p, r	tuplas de longitud r, ordenadas, con elementos repetidos

Ejemplos	Resultados
`product('ABCD', repeat=2)`	`AA AB AC AD BA BB BC BD CA CB CC CD DA DB DC DD`
`permutations('ABCD', 2)`	`AB AC AD BA BC BD CA CB CD DA DB DC`
`combinations('ABCD', 2)`	`AB AC AD BC BD CD`
`combinations_with_replacement('ABCD', 2)`	`AA AB AC AD BB BC BD CC CD DD`

Itertool Functions

The following functions all construct and return iterators. Some provide streams of infinite length, so they should only be accessed by functions or loops that truncate the stream.

itertools.accumulate(*iterable*[, *function*, *, *initial=None*])

Make an iterator that returns accumulated sums or accumulated results from other binary functions.

The *function* defaults to addition. The *function* should accept two arguments, an accumulated total and a value from the *iterable*.

If an *initial* value is provided, the accumulation will start with that value and the output will have one more element than the input iterable.

Aproximadamente equivalente a:

```
def accumulate(iterable, function=operator.add, *,
    'Return running totals'
    # accumulate([1,2,3,4,5]) → 1 3 6 10 15
    # accumulate([1,2,3,4,5], initial=100) → 100 10
    # accumulate([1,2,3,4,5], operator.mul) → 1 2 6

    iterator = iter(iterable)
    total = initial
    if initial is None:
        try:
            total = next(iterator)
        except StopIteration:
            return

    yield total
    for element in iterator:
        total = function(total, element)
        yield total
```

To compute a running minimum, set *function* to **`min()`**. For a running maximum, set *function* to **`max()`**. Or for a running product, set *function* to **`operator.mul()`**. To build an amortization table [https://www.ramseysolutions.com/real-estate/amortization-schedule], accumulate the interest and apply payments:

```
>>> data = [3, 4, 6, 2, 1, 9, 0, 7, 5, 8]
>>> list(accumulate(data, max))              # runn
[3, 4, 6, 6, 6, 9, 9, 9, 9, 9]
>>> list(accumulate(data, operator.mul))     # runn
[3, 12, 72, 144, 144, 1296, 0, 0, 0, 0]

# Amortize a 5% loan of 1000 with 10 annual payment
>>> update = lambda balance, payment: round(balance
>>> list(accumulate(repeat(90, 10), update, initial
[1000, 960, 918, 874, 828, 779, 728, 674, 618, 559,
```

Para una función similar que retorne únicamente el valor final acumulado, revisa **`functools.reduce()`**.

Added in version 3.2.

Distinto en la versión 3.3: Added the optional *function* parameter.

Distinto en la versión 3.8: Adicionó el argumento opcional *initial*.

itertools.batched(*iterable*, *n*, ***, *strict=False*)

Batch data from the *iterable* into tuples of length *n*. The last batch may be shorter than *n*.

If *strict* is true, will raise a **`ValueError`** if the final batch is shorter than *n*.

Loops over the input iterable and accumulates data into tuples up to size *n*. The input is consumed lazily, just enough to fill a batch. The result is yielded as soon as the batch is full or when the input iterable is exhausted:

```
>>> flattened_data = ['roses', 'red', 'violets', 'b
>>> unflattened = list(batched(flattened_data, 2))
>>> unflattened
[('roses', 'red'), ('violets', 'blue'), ('sugar', '
```

Aproximadamente equivalente a:

```
def batched(iterable, n, *, strict=False):
    # batched('ABCDEFG', 3) → ABC DEF G
    if n < 1:
        raise ValueError('n must be at least one')
    iterator = iter(iterable)
    while batch := tuple(islice(iterator, n)):
        if strict and len(batch) != n:
            raise ValueError('batched(): incomplete
        yield batch
```

Added in version 3.12.

Distinto en la versión 3.13: Added the *strict* option.

itertools.chain(**iterables*)

Make an iterator that returns elements from the first iterable until it is exhausted, then proceeds to the next iterable, until all of the iterables are exhausted. This combines multiple data sources into a single iterator. Roughly equivalent to:

```
def chain(*iterables):
    # chain('ABC', 'DEF') → A B C D E F
    for iterable in iterables:
        yield from iterable
```

classmethod chain.from_iterable(*iterable*)

Constructor alternativo para **`chain()`**. Obtiene entradas enlazadas de un mismo argumento que se evalúa perezosamente. Aproximadamente equivalente a:

```
def from_iterable(iterables):
    # chain.from_iterable(['ABC', 'DEF']) → A B C D
    for iterable in iterables:
        yield from iterable
```

itertools.combinations(*iterable*, *r*)

Retorna subsecuencias de longitud *r* con elementos del *iterable* de entrada.

The output is a subsequence of **`product()`** keeping only entries that are subsequences of the *iterable*. The length of the output is given by **`math.comb()`** which computes `n! / r! / (n - r)!` when `0 ≤ r ≤ n` or zero when `r > n`.

The combination tuples are emitted in lexicographic order according to the order of the input *iterable*. If the input *iterable* is sorted, the output tuples will be produced in sorted order.

Elements are treated as unique based on their position, not on their value. If the input elements are unique, there will be no repeated values within each combination.

Aproximadamente equivalente a:

```
def combinations(iterable, r):
    # combinations('ABCD', 2) → AB AC AD BC BD CD
    # combinations(range(4), 3) → 012 013 023 123

    pool = tuple(iterable)
    n = len(pool)
    if r > n:
        return
    indices = list(range(r))

    yield tuple(pool[i] for i in indices)
    while True:
        for i in reversed(range(r)):
            if indices[i] != i + n - r:
                break
        else:
            return
        indices[i] += 1
        for j in range(i+1, r):
            indices[j] = indices[j-1] + 1
        yield tuple(pool[i] for i in indices)
```

itertools.combinations_with_replacement(*iterable*, *r*)

Retorna subsecuencias, de longitud *r*, con elementos del *iterable* de entrada, permitiendo que haya elementos individuales repetidos más de una vez.

The output is a subsequence of `product()` that keeps only entries that are subsequences (with possible repeated elements) of the *iterable*. The number of subsequence returned is `(n + r - 1)! / r! / (n - 1)!` when `n > 0`.

The combination tuples are emitted in lexicographic order

according to the order of the input *iterable*. if the input *iterable* is sorted, the output tuples will be produced in sorted order.

Elements are treated as unique based on their position, not on their value. If the input elements are unique, the generated combinations will also be unique.

Aproximadamente equivalente a:

```
def combinations_with_replacement(iterable, r):
    # combinations_with_replacement('ABC', 2) → AA

    pool = tuple(iterable)
    n = len(pool)
    if not n and r:
        return
    indices = [0] * r

    yield tuple(pool[i] for i in indices)
    while True:
        for i in reversed(range(r)):
            if indices[i] != n - 1:
                break
        else:
            return
        indices[i:] = [indices[i] + 1] * (r - i)
        yield tuple(pool[i] for i in indices)
```

Added in version 3.1.

itertools.compress(*data*, *selectors*)

Make an iterator that returns elements from *data* where the corresponding element in *selectors* is true. Stops when either the *data* or *selectors* iterables have been exhausted. Roughly equivalent to:

```
def compress(data, selectors):
    # compress('ABCDEF', [1,0,1,0,1,1]) → A C E F
    return (datum for datum, selector in zip(data,
```

Added in version 3.1.

itertools.count(*start=0*, *step=1*)

Make an iterator that returns evenly spaced values beginning with *start*. Can be used with `map()` to generate consecutive data points or with `zip()` to add sequence numbers. Roughly equivalent to:

```
def count(start=0, step=1):
    # count(10) → 10 11 12 13 14 ...
    # count(2.5, 0.5) → 2.5 3.0 3.5 ...
    n = start
    while True:
        yield n
        n += step
```

When counting with floating-point numbers, better accuracy can sometimes be achieved by substituting multiplicative code such as: `(start + step * i for i in count())`.

Distinto en la versión 3.1: Se adicionó el argumento *step* y se permitieron argumentos diferentes a enteros.

itertools.cycle(*iterable*)

Make an iterator returning elements from the *iterable* and saving a copy of each. When the iterable is exhausted, return elements from the saved copy. Repeats indefinitely. Roughly equivalent to:

```
def cycle(iterable):
    # cycle('ABCD') → A B C D A B C D A B C D ...

    saved = []
    for element in iterable:
        yield element
        saved.append(element)
    while saved:
        for element in saved:
            yield element
```

This itertool may require significant auxiliary storage (depending on the length of the iterable).

itertools.dropwhile(*predicate, iterable*)

Make an iterator that drops elements from the *iterable* while the *predicate* is true and afterwards returns every element. Roughly equivalent to:

```
def dropwhile(predicate, iterable):
    # dropwhile(lambda x: x<5, [1,4,6,3,8]) → 6 3 8

    iterator = iter(iterable)
    for x in iterator:
        if not predicate(x):
            yield x
            break

    for x in iterator:
        yield x
```

Note this does not produce *any* output until the predicate first becomes false, so this itertool may have a lengthy start-up time.

itertools.filterfalse(*predicate, iterable*)

Make an iterator that filters elements from the *iterable* returning only those for which the *predicate* returns a false value. If *predicate* is `None`, returns the items that are false. Roughly equivalent to:

```
def filterfalse(predicate, iterable):
    # filterfalse(lambda x: x<5, [1,4,6,3,8]) → 6 8

    if predicate is None:
        predicate = bool

    for x in iterable:
        if not predicate(x):
            yield x
```

itertools.groupby(*iterable*, *key=None*)

Crea un iterador que retorna claves consecutivas y grupos del *iterable*. *key* es una función que calcula un valor clave para cada elemento. Si no se especifica o es `None`, *key* es una función de identidad por defecto y retorna el elemento sin cambios. Generalmente, el iterable necesita estar ordenado con la misma función *key*.

El funcionamiento de **`groupby()`** es similar al del filtro `uniq` en Unix. Genera un salto o un nuevo grupo cada vez que el valor de la función clave cambia (por lo que usualmente es necesario ordenar los datos usando la misma función clave). Ese comportamiento difiere del de GROUP BY de SQL, el cual agrega elementos comunes sin importar el orden de entrada.

El grupo retornado es un iterador mismo que comparte el iterable subyacente con **`groupby()`**. Al compartir la fuente, cuando el objeto **`groupby()`** se avanza, el grupo previo deja de ser visible. En ese caso, si los datos se necesitan posteriormente, se deberían almacenar como lista:

```
groups = []
uniquekeys = []
data = sorted(data, key=keyfunc)
for k, g in groupby(data, keyfunc):
    groups.append(list(g))      # Store group itera
    uniquekeys.append(k)
```

`groupby()` es aproximadamente equivalente a:

```
def groupby(iterable, key=None):
    # [k for k, g in groupby('AAAABBBCCDAABBB')] →
    # [list(g) for k, g in groupby('AAAABBBCCD')] →
    keyfunc = (lambda x: x) if key is None else key
    iterator = iter(iterable)
    exhausted = False

    def _grouper(target_key):
        nonlocal curr_value, curr_key, exhausted
        yield curr_value
        for curr_value in iterator:
            curr_key = keyfunc(curr_value)
            if curr_key != target_key:
                return
            yield curr_value
        exhausted = True

    try:
        curr_value = next(iterator)
    except StopIteration:
        return
    curr_key = keyfunc(curr_value)

    while not exhausted:
        target_key = curr_key
        curr_group = _grouper(target_key)
        yield curr_key, curr_group
        if curr_key == target_key:
            for _ in curr_group:
                pass
```

itertools.islice(*iterable*, *stop*)

itertools.islice(*iterable*, *start*, *stop*[, *step*])

Make an iterator that returns selected elements from the iterable. Works like sequence slicing but does not support negative values for *start*, *stop*, or *step*.

If *start* is zero or `None`, iteration starts at zero. Otherwise, elements from the iterable are skipped until *start* is reached.

If *stop* is `None`, iteration continues until the input is exhausted, if at all. Otherwise, it stops at the specified position.

If *step* is `None`, the step defaults to one. Elements are returned consecutively unless *step* is set higher than one which results in items being skipped.

Aproximadamente equivalente a:

```
def islice(iterable, *args):
    # islice('ABCDEFG', 2) → A B
    # islice('ABCDEFG', 2, 4) → C D
    # islice('ABCDEFG', 2, None) → C D E F G
    # islice('ABCDEFG', 0, None, 2) → A C E G

    s = slice(*args)
    start = 0 if s.start is None else s.start
    stop = s.stop
    step = 1 if s.step is None else s.step
    if start < 0 or (stop is not None and stop < 0)
        raise ValueError

    indices = count() if stop is None else range(ma
    next_i = start
    for i, element in zip(indices, iterable):
        if i == next_i:
            yield element
            next_i += step
```

If the input is an iterator, then fully consuming the *islice* advances the input iterator by `max(start, stop)` steps regardless of the *step* value.

itertools.pairwise(*iterable*)

Retorna los sucesivos pares superpuestos tomados de la entrada *iterable*.

El número de tuplas de 2 elementos en el iterador de salida será uno menos que el número de entradas. Estará vacío si la entrada iterable tiene menos de dos valores.

Aproximadamente equivalente a:

```
def pairwise(iterable):
    # pairwise('ABCDEFG') → AB BC CD DE EF FG

    iterator = iter(iterable)
    a = next(iterator, None)

    for b in iterator:
        yield a, b
        a = b
```

Added in version 3.10.

itertools.permutations(*iterable*, *r=None*)

Return successive *r* length permutations of elements [https://www.britannica.com/science/permutation] from the *iterable*.

Si *r* no es especificado o si es `None`, entonces por defecto *r* será igual a la longitud de *iterable* y todas las permutaciones de máxima longitud serán generadas.

The output is a subsequence of **`product()`** where entries with repeated elements have been filtered out. The length of the output is given by **`math.perm()`** which computes `n! / (n - r)!` when `0 ≤ r ≤ n` or zero when `r > n`.

The permutation tuples are emitted in lexicographic order according to the order of the input *iterable*. If the input *iterable* is sorted, the output tuples will be produced in sorted order.

Elements are treated as unique based on their position, not on their value. If the input elements are unique, there will be no repeated values within a permutation.

Aproximadamente equivalente a:

```
def permutations(iterable, r=None):
    # permutations('ABCD', 2) → AB AC AD BA BC BD C
    # permutations(range(3)) → 012 021 102 120 201

    pool = tuple(iterable)
    n = len(pool)
    r = n if r is None else r
    if r > n:
        return

    indices = list(range(n))
    cycles = list(range(n, n-r, -1))
    yield tuple(pool[i] for i in indices[:r])

    while n:
        for i in reversed(range(r)):
            cycles[i] -= 1
            if cycles[i] == 0:
                indices[i:] = indices[i+1:] + indic
                cycles[i] = n - i
            else:
                j = cycles[i]
                indices[i], indices[-j] = indices[-
                yield tuple(pool[i] for i in indice
                break
        else:
            return
```

itertools.product(*iterables*, *repeat*=*1*)

Cartesian product [https://en.wikipedia.org/wiki/Cartesian_product] of the input iterables.

Aproximadamente equivalente a tener bucles *for* anidados en un generador. Por ejemplo, `product(A, B)` es equivalente a `((x,y) for x in A for y in B)`.

Los bucles anidados hacen ciclos como un cuentapasos o taxímetro, con el elemento más hacia la derecha avanzando en cada iteración. Este patrón crea un orden lexicográfico en el que, si los iterables de entrada están ordenados, las tuplas producidas son emitidas de manera ordenada.

Para calcular el producto de un iterable consigo mismo, especifica el número de repeticiones con el argumento opcional *repeat*. Por ejemplo, `product(A, repeat=4)` es equivalente a `product(A, A, A, A)`.

Esta función es aproximadamente equivalente al código siguiente, exceptuando que la implementación real no acumula resultados intermedios en memoria:

```
def product(*iterables, repeat=1):
    # product('ABCD', 'xy') → Ax Ay Bx By Cx Cy Dx
    # product(range(2), repeat=3) → 000 001 010 011

    if repeat < 0:
        raise ValueError('repeat argument cannot be
    pools = [tuple(pool) for pool in iterables] * r

    result = [[]]
    for pool in pools:
        result = [x+[y] for x in result for y in po

    for prod in result:
        yield tuple(prod)
```

Antes de que **`product()`** se ejecute, consume completamente los iterables de entrada, manteniendo grupos de valores en la memoria para generar los productos. En consecuencia, solo es útil con entradas finitas.

itertools.repeat(*object*[, *times*])

Crea un iterador que retorna *object* una y otra vez. Se ejecuta indefinidamente a menos que se especifique el argumento *times*.

Aproximadamente equivalente a:

```
def repeat(object, times=None):
    # repeat(10, 3) → 10 10 10
    if times is None:
        while True:
            yield object
    else:
        for i in range(times):
            yield object
```

Un uso común de *repeat* es proporcionar un flujo de valores constantes a *map* o *zip*:

```
>>> list(map(pow, range(10), repeat(2)))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

itertools.starmap(*function*, *iterable*)

Make an iterator that computes the *function* using arguments obtained from the *iterable*. Used instead of **`map()`** when argument parameters have already been «pre-zipped» into tuples.

La diferencia entre **`map()`** y **`starmap()`** es paralela a la distinción entre `function(a,b)` y `function(*c)`. Aproximadamente equivalente a:

```
def starmap(function, iterable):
    # starmap(pow, [(2,5), (3,2), (10,3)]) → 32 9 1
    for args in iterable:
        yield function(*args)
```

itertools.takewhile(*predicate*, *iterable*)

Make an iterator that returns elements from the *iterable* as long as the *predicate* is true. Roughly equivalent to:

```
def takewhile(predicate, iterable):
    # takewhile(lambda x: x<5, [1,4,6,3,8]) → 1 4
    for x in iterable:
        if not predicate(x):
            break
        yield x
```

Note, the element that first fails the predicate condition is consumed from the input iterator and there is no way to access it. This could be an issue if an application wants to further consume the input iterator after *takewhile* has been run to exhaustion. To work around this problem, consider using more-itertools before_and_after() [https://more-itertools.readthedocs.io/en/stable/api.html#more_itertools.before_and_after] instead.

itertools.tee(*iterable*, *n=2*)

Retorna *n* iteradores independientes de un mismo iterador.

Aproximadamente equivalente a:

```
def tee(iterable, n=2):
    if n < 0:
        raise ValueError
    if n == 0:
        return ()
    iterator = _tee(iterable)
    result = [iterator]
    for _ in range(n - 1):
        result.append(_tee(iterator))
    return tuple(result)

class _tee:

    def __init__(self, iterable):
        it = iter(iterable)
        if isinstance(it, _tee):
            self.iterator = it.iterator
            self.link = it.link
        else:
            self.iterator = it
            self.link = [None, None]

    def __iter__(self):
        return self
    def __next__(self):
        link = self.link
        if link[1] is None:
            link[0] = next(self.iterator)
            link[1] = [None, None]
        value, self.link = link
        return value
```

When the input *iterable* is already a tee iterator object, all members of the return tuple are constructed as if they had been produced by the upstream `tee()` call. This «flattening step» allows nested `tee()` calls to share the same underlying data chain and to have a single update step rather than a chain of calls.

The flattening property makes tee iterators efficiently peekable:

```
def lookahead(tee_iterator):
     "Return the next value without moving the inpu
     [forked_iterator] = tee(tee_iterator, 1)
     return next(forked_iterator)

>>> iterator = iter('abcdef')
>>> [iterator] = tee(iterator, 1)    # Make the inpu
>>> next(iterator)                   # Move the iter
'a'
>>> lookahead(iterator)              # Check next va
'b'
>>> next(iterator)                   # Continue movi
'b'
```

`tee` iterators are not threadsafe. A **`RuntimeError`** may be raised when simultaneously using iterators returned by the same **`tee()`** call, even if the original *iterable* is threadsafe.

Esta herramienta de iteración puede requerir almacenamiento auxiliar significativo (dependiendo de qué tantos datos necesitan ser almacenados). En general, si un iterador utiliza todos o la mayoría de los datos antes que otro iterador comience, es más rápido utilizar `list()` en vez de `tee()`.

itertools.zip_longest(**iterables*, *fillvalue=None*)

Make an iterator that aggregates elements from each of the *iterables*.

If the iterables are of uneven length, missing values are filled-in with *fillvalue*. If not specified, *fillvalue* defaults to `None`.

Iteration continues until the longest iterable is exhausted.

Aproximadamente equivalente a:

```
def zip_longest(*iterables, fillvalue=None):
    # zip_longest('ABCD', 'xy', fillvalue='-') → Ax

    iterators = list(map(iter, iterables))
    num_active = len(iterators)
    if not num_active:
        return

    while True:
        values = []
        for i, iterator in enumerate(iterators):
            try:
                value = next(iterator)
            except StopIteration:
                num_active -= 1
                if not num_active:
                    return
                iterators[i] = repeat(fillvalue)
                value = fillvalue
            values.append(value)
        yield tuple(values)
```

If one of the iterables is potentially infinite, then the **`zip_longest()`** function should be wrapped with something that limits the number of calls (for example `islice()` or `takewhile()`).

Fórmulas con itertools

Esta sección muestra fórmulas para crear un conjunto de herramientas extendido usando las herramientas de itertools como piezas básicas.

The primary purpose of the itertools recipes is educational. The recipes show various ways of thinking about individual tools — for example, that `chain.from_iterable` is related to the concept of flattening. The recipes also give ideas about ways that the tools can be combined — for example, how `starmap()` and `repeat()` can work together. The recipes also show patterns for using itertools with the **`operator`** and **`collections`** modules as well as with the built-in itertools such as `map()`, `filter()`, `reversed()`, and `enumerate()`.

A secondary purpose of the recipes is to serve as an incubator. The `accumulate()`, `compress()`, and `pairwise()` itertools started out as recipes. Currently, the `sliding_window()`, `derangements()`, and `sieve()` recipes are being tested to see whether they prove their worth.

Substantially all of these recipes and many, many others can be installed from the more-itertools [https://pypi.org/project/more-itertools/] project found on the Python Package Index:

```
python -m pip install more-itertools
```

Many of the recipes offer the same high performance as the underlying toolset. Superior memory performance is kept by processing elements one at a time rather than bringing the whole iterable into memory all at once. Code volume is kept small by linking the tools together in a functional style [https://www.cs.kent.ac.uk/people/staff/dat/miranda/whyfp90.pdf]. High speed is retained by preferring «vectorized» building blocks over the use of for-loops and generators which incur interpreter overhead.

```
from itertools import (accumulate, batched, chain, combi
     count, cycle, filterfalse, groupby, islice, permuta
     repeat, starmap, tee, zip_longest)
```

```
from collections import Counter, deque
from contextlib import suppress
from functools import reduce
from heapq import heappush, heappushpop, heappush_max, h
from math import comb, isqrt, prod, sumprod
from operator import getitem, is_not, itemgetter, mul, r


# ==== Basic one liners ====

def take(n, iterable):
    "Return first n items of the iterable as a list."
    return list(islice(iterable, n))

def prepend(value, iterable):
    "Prepend a single value in front of an iterable."
    # prepend(1, [2, 3, 4]) → 1 2 3 4
    return chain([value], iterable)

def repeatfunc(function, times=None, *args):
    "Repeat calls to a function with specified arguments
    if times is None:
        return starmap(function, repeat(args))
    return starmap(function, repeat(args, times))

def flatten(list_of_lists):
    "Flatten one level of nesting."
    return chain.from_iterable(list_of_lists)

def ncycles(iterable, n):
    "Returns the sequence elements n times."
    return chain.from_iterable(repeat(tuple(iterable), r

def loops(n):
    "Loop n times. Like range(n) but without creating i
    # for _ in loops(100): ...
    return repeat(None, n)
```

```
def tail(n, iterable):
    "Return an iterator over the last n items."
    # tail(3, 'ABCDEFG') → E F G
    return iter(deque(iterable, maxlen=n))

def consume(iterator, n=None):
    "Advance the iterator n-steps ahead. If n is None, c
    # Use functions that consume iterators at C speed.
    if n is None:
        deque(iterator, maxlen=0)
    else:
        next(islice(iterator, n, n), None)

def nth(iterable, n, default=None):
    "Returns the nth item or a default value."
    return next(islice(iterable, n, None), default)

def quantify(iterable, predicate=bool):
    "Given a predicate that returns True or False, count
    return sum(map(predicate, iterable))

def first_true(iterable, default=False, predicate=None):
    "Returns the first true value or the *default* if th
    # first_true([a, b, c], x) → a or b or c or x
    # first_true([a, b], x, f) → a if f(a) else b if f(b
    return next(filter(predicate, iterable), default)

def all_equal(iterable, key=None):
    "Returns True if all the elements are equal to each
    # all_equal('4٤௪౪໔', key=int) → True
    return len(take(2, groupby(iterable, key))) <= 1


# ==== Data pipelines ====

def unique_justseen(iterable, key=None):
    "Yield unique elements, preserving order. Remember o
    # unique_justseen('AAAABBBCCDAABBB') → A B C D A B
    # unique_justseen('ABBcCAD', str.casefold) → A B c A
    if key is None:
        return map(itemgetter(0), groupby(iterable))
    return map(next, map(itemgetter(1), groupby(iterable

def unique_everseen(iterable, key=None):
    "Yield unique elements, preserving order. Remember a
    # unique_everseen('AAAABBBCCDAABBB') → A B C D
    # unique_everseen('ABBcCAD', str.casefold) → A B c D
    seen = set()
    if key is None:
        for element in filterfalse(seen.__contains__, it
            seen.add(element)
            yield element
    else:
        for element in iterable:
            k = key(element)
            if k not in seen:
                seen.add(k)
                yield element

def unique(iterable, key=None, reverse=False):
    "Yield unique elements in sorted order. Supports unh
    # unique([[1, 2], [3, 4], [1, 2]]) → [1, 2] [3, 4]
    sequenced = sorted(iterable, key=key, reverse=revers
    return unique_justseen(sequenced, key=key)

def sliding_window(iterable, n):
    "Collect data into overlapping fixed-length chunks o
    # sliding_window('ABCDEFG', 3) → ABC BCD CDE DEF EFG
    iterator = iter(iterable)
    window = deque(islice(iterator, n - 1), maxlen=n)
    for x in iterator:
        window.append(x)
        yield tuple(window)

def grouper(iterable, n, *, incomplete='fill', fillvalue
    "Collect data into non-overlapping fixed-length chun
    # grouper('ABCDEFG', 3, fillvalue='x')       → ABC D
    # grouper('ABCDEFG', 3, incomplete='strict') → ABC D
    # grouper('ABCDEFG', 3, incomplete='ignore') → ABC D
    iterators = [iter(iterable)] * n
    match incomplete:
        case 'fill':
            return zip_longest(*iterators, fillvalue=fil
        case 'strict':
            return zip(*iterators, strict=True)
        case 'ignore':
            return zip(*iterators)
        case _:
            raise ValueError('Expected fill, strict, or

def roundrobin(*iterables):
    "Visit input iterables in a cycle until each is exha
    # roundrobin('ABC', 'D', 'EF') → A D E B F C
    # Algorithm credited to George Sakkis
    iterators = map(iter, iterables)
    for num_active in range(len(iterables), 0, -1):
        iterators = cycle(islice(iterators, num_active))
        yield from map(next, iterators)

def subslices(seq):
    "Return all contiguous non-empty subslices of a sequ
    # subslices('ABCD') → A AB ABC ABCD B BC BCD C CD D
    slices = starmap(slice, combinations(range(len(seq)
    return map(getitem, repeat(seq), slices)

def derangements(iterable, r=None):
    "Produce r length permutations without fixed points.
    # derangements('ABCD') → BADC BCDA BDAC CADB CDAB CD
    # Algorithm credited to Stefan Pochmann
    seq = tuple(iterable)
    pos = tuple(range(len(seq)))
    have_moved = map(map, repeat(is_not), repeat(pos), p
    valid_derangements = map(all, have_moved)
    return compress(permutations(seq, r=r), valid_derang
def iter_index(iterable, value, start=0, stop=None):
    "Return indices where a value occurs in a sequence
    # iter_index('AABCADEAF', 'A') → 0 1 4 7
    seq_index = getattr(iterable, 'index', None)
    if seq_index is None:
        iterator = islice(iterable, start, stop)
        for i, element in enumerate(iterator, start):
            if element is value or element == value:
                yield i
    else:
        stop = len(iterable) if stop is None else stop
        i = start
        with suppress(ValueError):
            while True:
                yield (i := seq_index(value, i, stop))
                i += 1

def iter_except(function, exception, first=None):
    "Convert a call-until-exception interface to an iter
    # iter_except(d.popitem, KeyError) → non-blocking di
    with suppress(exception):
        if first is not None:
            yield first()
        while True:
            yield function()


# ==== Mathematical operations ====

def multinomial(*counts):
    "Number of distinct arrangements of a multiset."
    # Counter('abracadabra').values() → 5 2 2 1 1
    # multinomial(5, 2, 2, 1, 1) → 83160
    return prod(map(comb, accumulate(counts), counts))

def powerset(iterable):
    "Subsequences of the iterable from shortest to longe
    # powerset([1,2,3]) → () (1,) (2,) (3,) (1,2) (1,3)
    s = list(iterable)
    return chain.from_iterable(combinations(s, r) for r

def sum_of_squares(iterable):
    "Add up the squares of the input values."
    # sum_of_squares([10, 20, 30]) → 1400
    return sumprod(*tee(iterable))


# ==== Matrix operations ====

def reshape(matrix, columns):
    "Reshape a 2-D matrix to have a given number of colu
    # reshape([(0, 1), (2, 3), (4, 5)], 3) →  (0, 1, 2)
    return batched(chain.from_iterable(matrix), columns,

def transpose(matrix):
    "Swap the rows and columns of a 2-D matrix."
    # transpose([(1, 2, 3), (11, 22, 33)]) → (1, 11) (2,
    return zip(*matrix, strict=True)

def matmul(m1, m2):
    "Multiply two matrices."
    # matmul([(7, 5), (3, 5)], [(2, 5), (7, 9)]) → (49,
    n = len(m2[0])
    return batched(starmap(sumprod, product(m1, transpos


# ==== Polynomial arithmetic ====

def convolve(signal, kernel):
    """Discrete linear convolution of two iterables.
    Equivalent to polynomial multiplication.

    Convolutions are mathematically commutative; however
    evaluated differently.  The signal is consumed lazil
    infinite. The kernel is fully consumed before the ca
    Article:  https://betterexplained.com/articles/intui
    Video:    https://www.youtube.com/watch?v=KuXjwB4LzS
    """
    # convolve([1, -1, -20], [1, -3]) → 1 -4 -17 60
    # convolve(data, [0.25, 0.25, 0.25, 0.25]) → Moving
    # convolve(data, [1/2, 0, -1/2]) → 1st derivative es
    # convolve(data, [1, -2, 1]) → 2nd derivative estima
    kernel = tuple(kernel)[::-1]
    n = len(kernel)
    padded_signal = chain(repeat(0, n-1), signal, repeat
    windowed_signal = sliding_window(padded_signal, n)
    return map(sumprod, repeat(kernel), windowed_signal)

def polynomial_from_roots(roots):
    """Compute a polynomial's coefficients from its root

       (x - 5) (x + 4) (x - 3)  expands to:   x³ -4x² -1
    """
    # polynomial_from_roots([5, -4, 3]) → [1, -4, -17, 6
    factors = zip(repeat(1), map(neg, roots))
    return list(reduce(convolve, factors, [1]))

def polynomial_eval(coefficients, x):
    """Evaluate a polynomial at a specific value.

    Computes with better numeric stability than Horner's
    """
    # Evaluate x³ -4x² -17x + 60 at x = 5
    # polynomial_eval([1, -4, -17, 60], x=5) → 0
    n = len(coefficients)
    if not n:
        return type(x)(0)
    powers = map(pow, repeat(x), reversed(range(n)))
    return sumprod(coefficients, powers)

def polynomial_derivative(coefficients):
    """Compute the first derivative of a polynomial.
```

```
       f(x)  =  x³ -4x² -17x + 60
       f'(x) = 3x² -8x  -17
    """
    # polynomial_derivative([1, -4, -17, 60]) → [3, -8,
    n = len(coefficients)
    powers = reversed(range(1, n))
    return list(map(mul, coefficients, powers))


# ==== Number theory ====

def sieve(n):
    "Primes less than n."
    # sieve(30) → 2 3 5 7 11 13 17 19 23 29
    if n > 2:
        yield 2
    data = bytearray((0, 1)) * (n // 2)
    for p in iter_index(data, 1, start=3, stop=isqrt(n)
        data[p*p : n : p+p] = bytes(len(range(p*p, n, p
    yield from iter_index(data, 1, start=3)

def factor(n):
    "Prime factors of n."
    # factor(99) → 3 3 11
    # factor(1_000_000_000_000_007) → 47 59 36062026685
    # factor(1_000_000_000_000_403) → 1000000000000403
    for prime in sieve(isqrt(n) + 1):
        while not n % prime:
            yield prime
            n //= prime
            if n == 1:
                return
    if n > 1:
        yield n

def is_prime(n):
    "Return True if n is prime."
    # is_prime(1_000_000_000_000_403) → True
    return n > 1 and next(factor(n)) == n

def totient(n):
    "Count of natural numbers up to n that are coprime t
    # https://mathworld.wolfram.com/TotientFunction.html
    # totient(12) → 4 because len([1, 5, 7, 11]) == 4
    for prime in set(factor(n)):
        n -= n // prime
    return n


# ==== Running statistics ====

def running_mean(iterable):
    "Average of values seen so far."
    # running_mean([37, 33, 38, 28]) → 37 35 36 34
    return map(truediv, accumulate(iterable), count(1))

def running_min(iterable):
    "Smallest of values seen so far."
    # running_min([37, 33, 38, 28]) → 37 33 33 28
    return accumulate(iterable, func=min)

def running_max(iterable):
    "Largest of values seen so far."
    # running_max([37, 33, 38, 28]) → 37 37 38 38
    return accumulate(iterable, func=max)

def running_median(iterable):
    "Median of values seen so far."
    # running_median([37, 33, 38, 28]) → 37 35 37 35
    read = iter(iterable).__next__
    lo = []  # max-heap
    hi = []  # min-heap the same size as or one smaller
    with suppress(StopIteration):
        while True:
            heappush_max(lo, heappushpop(hi, read()))
```

```
                yield lo[0]
                heappush(hi, heappushpop_max(lo, read()))
                yield (lo[0] + hi[0]) / 2

def running_statistics(iterable):
    "Aggregate statistics for values seen so far."
    # Generate tuples:  (size, minimum, median, maximum,
    t0, t1, t2, t3 = tee(iterable, 4)
    return zip(
        count(1),
        running_min(t0),
        running_median(t1),
        running_max(t2),
        running_mean(t3),
    )
```

`functools` — Higher-order functions and operations on callable objects

Código fuente: Lib/functools.py [https://github.com/python/cpython/tree/3.14/Lib/functools.py]

The **`functools`** module is for higher-order functions: functions that act on or return other functions. In general, any callable object can be treated as a function for the purposes of this module.

The **`functools`** module defines the following functions:

@functools.cache(*user_function*)

Caché de funciones ilimitado, ligero y simple. Aveces llamado «memoización» [https://es.wikipedia.org/wiki/Memoizaci%C3%B3n].

Returns the same as `lru_cache(maxsize=None)`, creating a thin wrapper around a dictionary lookup for the function arguments. Because it never needs to evict old values, this is smaller and faster than **`lru_cache()`** with a size limit.

Por ejemplo:

```
@cache
def factorial(n):
    return n * factorial(n-1) if n else 1

>>> factorial(10)    # no previously cached result, 
3628800
>>> factorial(5)     # no new calls, just returns th
120
>>> factorial(12)    # two new recursive calls, fact
```

```
479001600
```

The cache is threadsafe so that the wrapped function can be used in multiple threads. This means that the underlying data structure will remain coherent during concurrent updates.

It is possible for the wrapped function to be called more than once if another thread makes an additional call before the initial call has been completed and cached.

Added in version 3.9.

@functools.cached_property(*func*)

Transforma un método de una clase en una propiedad cuyo valor se computa una vez y luego se almacena como un atributo normal durante la vida de la instancia. Similar a **`property()`**, con la adición de caching. Útil para propiedades calculadas costosas de instancias que de otra manera son efectivamente inmutables.

Ejemplo:

```
class DataSet:

    def __init__(self, sequence_of_numbers):
        self._data = tuple(sequence_of_numbers)

    @cached_property
    def stdev(self):
        return statistics.stdev(self._data)
```

La mecánica de **`cached_property()`** es algo diferente de **`property()`**. Un atributo de bloques de propiedad normal escribe a menos que se defina un establecedor. Por el contrario, *cached_property* permite escrituras.

El decorador *cached_property* solo se ejecuta en búsquedas y solo cuando no existe un atributo con el mismo nombre. Cuando se ejecuta, *cached_property* escribe en el atributo con el mismo nombre. Las lecturas y escrituras de atributos posteriores tienen prioridad sobre el método *cached_property* y funciona como un atributo normal.

El valor en caché se puede borrar eliminando el atributo. Esto permite que el método *cached_property* se ejecute nuevamente.

The *cached_property* does not prevent a possible race condition in multi-threaded usage. The getter function could run more than once on the same instance, with the latest run setting the cached value. If the cached property is idempotent or otherwise not harmful to run more than once on an instance, this is fine. If synchronization is needed, implement the necessary locking inside the decorated getter function or around the cached property access.

Tenga en cuenta que este decorador interfiere con el funcionamiento de diccionarios de intercambio de claves **PEP 412** [https://peps.python.org/pep-0412/]. Esto significa que los diccionarios de instancias pueden ocupar más espacio de lo habitual.

Además, este decorador requiere que el atributo `__dict__` en cada instancia sea un mapeo mutable. Esto significa que no funcionará con algunos tipos, como las metaclases (ya que los atributos `__dict__` en las instancias de tipos son proxies de solo lectura para el espacio de nombres de la clase) y aquellos que especifican `__slots__` sin incluir `__dict__` como una de las ranuras definidas (ya que tales clases no proporcionan un atributo `__dict__` en absoluto).

If a mutable mapping is not available or if space-efficient key sharing is desired, an effect similar to **`cached_property()`** can also be achieved by stacking **`property()`** on top of **`lru_cache()`**. See ¿Cómo cacheo llamadas de método? for more details on how this differs from **`cached_property()`**.

Added in version 3.8.

Distinto en la versión 3.12: Prior to Python 3.12, `cached_property` included an undocumented lock to ensure that in multi-threaded usage the getter function was guaranteed to run only once per instance. However, the lock was per-property, not per-instance, which could result in unacceptably high lock contention. In Python 3.12+ this locking is removed.

functools.cmp_to_key(*func*)

Transformar una función de comparación de estilo antiguo en una key function. Se utiliza con herramientas que aceptan funciones clave (como **`sorted()`**, **`min()`**, **`max()`**, **`heapq.nlargest()`**, **`heapq.nsmallest()`**, **`itertools.groupby()`**). Esta función se utiliza principalmente como una herramienta de transición para los programas que se están convirtiendo a partir de Python 2, que soportaba el uso de funciones de comparación.

Una función de comparación es cualquier invocable que acepta dos argumentos, los compara y devuelve un número negativo para menor que, cero para igualdad o un número positivo para mayor que. Una función clave es una función invocable que acepta un argumento y devuelve otro valor para usar como clave de ordenación.

Ejemplo:

```
sorted(iterable, key=cmp_to_key(locale.strcoll))  #
```

Para ejemplos de clasificación y un breve tutorial de clasificación, ver Sorting Techniques.

Added in version 3.2.

@functools.lru_cache(*user_function*)

@functools.lru_cache(*maxsize=128*, *typed=False*)

Decorador para envolver una función con un memorizador invocable que guarda hasta el *maxsize* de las llamadas más recientes. Puede salvar el tiempo cuando una función costosa o de E/S es llamada periódicamente con los mismos argumentos.

The cache is threadsafe so that the wrapped function can be used in multiple threads. This means that the underlying data structure will remain coherent during concurrent updates.

It is possible for the wrapped function to be called more than once if another thread makes an additional call before the initial call has been completed and cached.

Since a dictionary is used to cache results, the positional and keyword arguments to the function must be hashable.

Los patrones de argumentos distintos pueden considerarse llamadas distintas con entradas de memoria caché separadas. Por ejemplo, `f(a=1, b=2)` y `f(b=2, a=1)` difieren en el orden de sus argumentos de palabras clave y pueden tener dos entradas de caché separadas.

Si se especifica *user_function*, debe ser una llamada. Esto permite que el decorador *lru_cache* se aplique directamente a una función de usuario, dejando el *maxsize* en su valor por defecto de 128:

```
@lru_cache
def count_vowels(sentence):
    return sum(sentence.count(vowel) for vowel in '
```

Si *maxsize* está configurado como `None`, la función LRU está desactivada y la caché puede crecer sin límites.

Si *typed* se establece en verdadero, los argumentos de función de diferentes tipos se almacenarán en caché por separado. Si *typed* es falso, la implementación generalmente las considerará como llamadas equivalentes y solo almacenará en caché un único resultado. (Algunos tipos como *str* y *int* pueden almacenarse en caché por separado incluso cuando *typed* es falso).

Note, type specificity applies only to the function's immediate arguments rather than their contents. The scalar arguments, `Decimal(42)` and `Fraction(42)` are treated as distinct calls with distinct results. In contrast, the tuple arguments `('answer', Decimal(42))` and `('answer',`

`Fraction(42))` are treated as equivalent.

The wrapped function is instrumented with a **`cache_parameters()`** function that returns a new **`dict`** showing the values for *maxsize* and *typed*. This is for information purposes only. Mutating the values has no effect.

To help measure the effectiveness of the cache and tune the *maxsize* parameter, the wrapped function is instrumented with a **`cache_info()`** function that returns a named tuple showing *hits*, *misses*, *maxsize* and *currsize*.

The decorator also provides a **`cache_clear()`** function for clearing or invalidating the cache.

La función subyacente original es accesible a través del atributo **`__wrapped__`**. Esto es útil para la introspección, para evitar el caché, o para volver a envolver la función con un caché diferente.

La caché mantiene referencias de los argumentos y retorna los valores hasta que se caduquen de la caché o hasta que se borre la caché.

Si un método se almacena en caché, el argumento de la instancia `self` se incluye en el caché. Ver ¿Cómo cacheo llamadas de método?

An LRU (least recently used) cache [https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_Recently_Used_(LRU)] works best when the most recent calls are the best predictors of upcoming calls (for example, the most popular articles on a news server tend to change each day). The cache's size limit assures that the cache does not grow without bound on long-running processes such as web servers.

In general, the LRU cache should only be used when you want to reuse previously computed values. Accordingly, it doesn't make sense to cache functions with side-effects, functions that need to create distinct mutable objects on each call (such as generators and async functions), or impure

functions such as time() or random().

Ejemplo de un caché de la LRU para contenido web estático:

```
@lru_cache(maxsize=32)
def get_pep(num):
    'Retrieve text of a Python Enhancement Proposal
    resource = f'https://peps.python.org/pep-{num:0
    try:
        with urllib.request.urlopen(resource) as s:
            return s.read()
    except urllib.error.HTTPError:
        return 'Not Found'

>>> for n in 8, 290, 308, 320, 8, 218, 320, 279, 28
...     pep = get_pep(n)
...     print(n, len(pep))

>>> get_pep.cache_info()
CacheInfo(hits=3, misses=8, maxsize=32, currsize=8)
```

Ejemplo de computar eficientemente los números de Fibonacci [https://en.wikipedia.org/wiki/Fibonacci_number] usando una memoria caché para implementar una técnica de programación dinámica [https://en.wikipedia.org/wiki/Dynamic_programming]:

```
@lru_cache(maxsize=None)
def fib(n):
    if n < 2:
        return n
    return fib(n-1) + fib(n-2)

>>> [fib(n) for n in range(16)]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233,

>>> fib.cache_info()
CacheInfo(hits=28, misses=16, maxsize=None, currsiz
```

Added in version 3.2.

Distinto en la versión 3.3: Añadida la opción *typed* option.

Distinto en la versión 3.8: Añadida la opción *user_function*.

Distinto en la versión 3.9: Added the function **`cache_parameters()`**

@functools.total_ordering

Dada una clase que define uno o más métodos de ordenamiento de comparación ricos, este decorador de clase suministra el resto. Esto simplifica el esfuerzo de especificar todas las posibles operaciones de comparación rica:

The class must define one of `__lt__()`, `__le__()`, `__gt__()`, or `__ge__()`. In addition, the class should supply an `__eq__()` method.

Por ejemplo:

```
@total_ordering
class Student:
    def _is_valid_operand(self, other):
        return (hasattr(other, "lastname") and
                hasattr(other, "firstname"))
    def __eq__(self, other):
        if not self._is_valid_operand(other):
            return NotImplemented
        return ((self.lastname.lower(), self.firstn
                (other.lastname.lower(), other.firs
    def __lt__(self, other):
        if not self._is_valid_operand(other):
            return NotImplemented
        return ((self.lastname.lower(), self.firstn
                (other.lastname.lower(), other.firs
```

Nota

Mientras que este decorador facilita la creación de tipos bien comportados y totalmente ordenados, *does* a costa de una ejecución más lenta y de trazos de pila (*stack traces*) más complejos para los métodos de comparación derivados. Si la evaluación comparativa del rendimiento indica que se trata de un cuello de botella para una aplicación determinada, la aplicación de los seis métodos de comparación ricos en su lugar es probable que proporcione un fácil aumento de la velocidad.

Nota

Este decorador no intenta anular métodos que han sido declarados en la clase *sus superclases*. Lo que significa que si una superclase define un operador de comparación, *total_ordering* no lo implementará de nuevo, incluso si el método original es abstracto.

Added in version 3.2.

Distinto en la versión 3.4: Returning `NotImplemented` from the underlying comparison function for unrecognised types is now supported.

functools.Placeholder

A singleton object used as a sentinel to reserve a place for positional arguments when calling **`partial()`** and **`partialmethod()`**.

Added in version 3.14.

functools.partial(*func*, /, **args*, ***keywords*)

Retorna un nuevo partial object que cuando sea llamado se comportará como *func* llamado con los argumentos posicionales *args* y los argumentos de palabras clave *keywords*. Si se suministran más argumentos a la llamada, se añaden a *args*. Si se suministran más argumentos de palabras clave, se extienden y anulan las *keywords*. Aproximadamente equivalente a:

```
def partial(func, /, *args, **keywords):
    def newfunc(*more_args, **more_keywords):
        return func(*args, *more_args, **(keywords
    newfunc.func = func
    newfunc.args = args
    newfunc.keywords = keywords
    return newfunc
```

The **`partial()`** function is used for partial function application which «freezes» some portion of a function's arguments and/or keywords resulting in a new object with a simplified signature. For example, **`partial()`** can be used to create a callable that behaves like the **`int()`** function where the *base* argument defaults to `2`:

```
>>> basetwo = partial(int, base=2)
>>> basetwo.__doc__ = 'Convert base 2 string to an
>>> basetwo('10010')
18
```

If **`Placeholder`** sentinels are present in *args*, they will be filled first when **`partial()`** is called. This makes it possible to pre-fill any positional argument with a call to **`partial()`**; without **`Placeholder`**, only the chosen number of leading positional arguments can be pre-filled.

If any **`Placeholder`** sentinels are present, all must be filled at call time:

```
>>> say_to_world = partial(print, Placeholder, Plac
>>> say_to_world('Hello', 'dear')
Hello dear world!
```

Calling `say_to_world('Hello')` raises a **`TypeError`**, because only one positional argument is provided, but there are two placeholders that must be filled in.

If **`partial()`** is applied to an existing **`partial()`** object, **`Placeholder`** sentinels of the input object are filled in with new positional arguments. A placeholder can be retained by inserting a new **`Placeholder`** sentinel to the place held by a previous **`Placeholder`**:

```
>>> from functools import partial, Placeholder as _
>>> remove = partial(str.replace, _, _, '')
>>> message = 'Hello, dear dear world!'
>>> remove(message, ' dear')
'Hello, world!'
>>> remove_dear = partial(remove, _, ' dear')
>>> remove_dear(message)
'Hello, world!'
>>> remove_first_dear = partial(remove_dear, _, 1)
>>> remove_first_dear(message)
'Hello, dear world!'
```

`Placeholder` cannot be passed to **`partial()`** as a keyword argument.

Distinto en la versión 3.14: Added support for **`Placeholder`** in positional arguments.

class functools.partialmethod(*func*, /, *args, **keywords)

Retorna un nuevo descriptor **`partialmethod`** que se comporta como **`partial`** excepto que está diseñado para ser usado como una definición de método en lugar de ser directamente invocable.

func debe ser un descriptor o un invocable (los objetos que son ambos, como las funciones normales, se manejan como descriptores).

When *func* is a descriptor (such as a normal Python function, **`classmethod()`**, **`staticmethod()`**, **`abstractmethod()`** or another instance of **`partialmethod`**), calls to `__get__` are delegated to the underlying descriptor, and an appropriate partial object returned as the result.

Cuando *func* es una llamada no descriptiva, se crea dinámicamente un método de unión apropiado. Esto se comporta como una función Python normal cuando se usa como método: el argumento *self* se insertará como el primer argumento posicional, incluso antes de las *args* y *keywords* suministradas al constructor **`partialmethod`**.

Ejemplo:

```
>>> class Cell:
...     def __init__(self):
...         self._alive = False
...     @property
...     def alive(self):
...         return self._alive
...     def set_state(self, state):
...         self._alive = bool(state)
...     set_alive = partialmethod(set_state, True)
...     set_dead = partialmethod(set_state, False)
...
>>> c = Cell()
>>> c.alive
False
>>> c.set_alive()
>>> c.alive
True
```

Added in version 3.4.

functools.reduce(*function*, *iterable*, /[, *initial*])

Apply *function* of two arguments cumulatively to the items of *iterable*, from left to right, so as to reduce the iterable to a single value. For example, `reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])` calculates `((((1+2)+3)+4)+5)`. The left argument, *x*, is the accumulated value and the right argument, *y*, is the update value from the *iterable*. If the optional *initial* is present, it is placed before the items of the iterable in the calculation, and serves as a default when the iterable is empty. If *initial* is not given and *iterable* contains only one item, the first item is returned.

Aproximadamente equivalente a:

```
initial_missing = object()

def reduce(function, iterable, /, initial=initial_m
    it = iter(iterable)
    if initial is initial_missing:
        value = next(it)
    else:
        value = initial
    for element in it:
        value = function(value, element)
    return value
```

Ver `itertools.accumulate()` para un iterador que produce todos los valores intermedios.

Distinto en la versión 3.14: initial is now supported as a keyword argument.

@functools.singledispatch

Transformar una función en una single-dispatch generic function.

Para definir una función genérica, decórala con el decorador `@singledispatch`. Al definir una función usando `@singledispatch`, tenga en cuenta que el envío ocurre en el tipo del primer argumento:

```
>>> from functools import singledispatch
>>> @singledispatch
... def fun(arg, verbose=False):
...     if verbose:
...         print("Let me just say,", end=" ")
...     print(arg)
```

To add overloaded implementations to the function, use the **`register()`** attribute of the generic function, which can be used as a decorator. For functions annotated with types, the decorator will infer the type of the first argument automatically:

```
>>> @fun.register
... def _(arg: int, verbose=False):
...     if verbose:
```

```
...         print("Strength in numbers, eh?", end="
...     print(arg)
...
>>> @fun.register
... def _(arg: list, verbose=False):
...     if verbose:
...         print("Enumerate this:")
...     for i, elem in enumerate(arg):
...         print(i, elem)
```

`typing.Union` can also be used:

```
>>> @fun.register
... def _(arg: int | float, verbose=False):
...     if verbose:
...         print("Strength in numbers, eh?", end="
...     print(arg)
...
>>> from typing import Union
>>> @fun.register
... def _(arg: Union[list, set], verbose=False):
...     if verbose:
...         print("Enumerate this:")
...     for i, elem in enumerate(arg):
...         print(i, elem)
...
```

Para el código que no utiliza anotaciones de tipo, el argumento de tipo apropiado puede ser pasado explícitamente al propio decorador:

```
>>> @fun.register(complex)
... def _(arg, verbose=False):
...     if verbose:
...         print("Better than complicated.", end="
...     print(arg.real, arg.imag)
...
```

For code that dispatches on a collections type (e.g., `list`), but wants to typehint the items of the collection (e.g.,

`list[int]`), the dispatch type should be passed explicitly to the decorator itself with the typehint going into the function definition:

```
>>> @fun.register(list)
... def _(arg: list[int], verbose=False):
...     if verbose:
...         print("Enumerate this:")
...     for i, elem in enumerate(arg):
...         print(i, elem)
```

Nota

At runtime the function will dispatch on an instance of a list regardless of the type contained within the list i.e. `[1,2,3]` will be dispatched the same as `["foo", "bar", "baz"]`. The annotation provided in this example is for static type checkers only and has no runtime impact.

To enable registering lambdas and pre-existing functions, the **`register()`** attribute can also be used in a functional form:

```
>>> def nothing(arg, verbose=False):
...     print("Nothing.")
...
>>> fun.register(type(None), nothing)
```

The **`register()`** attribute returns the undecorated function. This enables decorator stacking, **`pickling`**, and the creation of unit tests for each variant independently:

```
>>> @fun.register(float)
... @fun.register(Decimal)
... def fun_num(arg, verbose=False):
...     if verbose:
...         print("Half of your number:", end=" ")
...     print(arg / 2)
...
>>> fun_num is fun
```

```
False
```

Cuando se llama, la función genérica despacha sobre el tipo del primer argumento:

```
>>> fun("Hello, world.")
Hello, world.
>>> fun("test.", verbose=True)
Let me just say, test.
>>> fun(42, verbose=True)
Strength in numbers, eh? 42
>>> fun(['spam', 'spam', 'eggs', 'spam'], verbose=T
Enumerate this:
0 spam
1 spam
2 eggs
3 spam
>>> fun(None)
Nothing.
>>> fun(1.23)
0.615
```

Cuando no hay una implementación registrada para un tipo específico, se usa su orden de resolución de métodos para encontrar una implementación más genérica. La función original decorada con `@singledispatch` está registrada para el tipo base **`object`**, lo que significa que se usa si no se encuentra una implementación mejor.

Si una implementación está registrada en un abstract base class, las subclases virtuales de la clase base se enviarán a esa implementación:

```
>>> from collections.abc import Mapping
>>> @fun.register
... def _(arg: Mapping, verbose=False):
...     if verbose:
...         print("Keys & Values")
...     for key, value in arg.items():
...         print(key, "=>", value)
```

```
...
>>> fun({"a": "b"})
a => b
```

Para verificar qué implementación elegirá la función genérica para un tipo dado, use el atributo `dispatch()`:

```
>>> fun.dispatch(float)
<function fun_num at 0x1035a2840>
>>> fun.dispatch(dict)    # note: default implement
<function fun at 0x103fe0000>
```

Para acceder a todas las implementaciones registradas, utilice el atributo `registry` de sólo lectura:

```
>>> fun.registry.keys()
dict_keys([<class 'NoneType'>, <class 'int'>, <clas
          <class 'decimal.Decimal'>, <class 'list'>
          <class 'float'>])
>>> fun.registry[float]
<function fun_num at 0x1035a2840>
>>> fun.registry[object]
<function fun at 0x103fe0000>
```

Added in version 3.4.

Distinto en la versión 3.7: The **`register()`** attribute now supports using type annotations.

Distinto en la versión 3.11: The **`register()`** attribute now supports **`typing.Union`** as a type annotation.

class functools.singledispatchmethod(*func*)

Transformar un método en un single-dispatch generic function.

To define a generic method, decorate it with the `@singledispatchmethod` decorator. When defining a method using `@singledispatchmethod`, note that the dispatch happens on the type of the first non-*self* or non-*cls*

argument:

```
class Negator:
    @singledispatchmethod
    def neg(self, arg):
        raise NotImplementedError("Cannot negate a"

    @neg.register
    def _(self, arg: int):
        return -arg

    @neg.register
    def _(self, arg: bool):
        return not arg
```

`@singledispatchmethod` supports nesting with other decorators such as **`@classmethod`**. Note that to allow for `dispatcher.register`, `singledispatchmethod` must be the *outer most* decorator. Here is the `Negator` class with the `neg` methods bound to the class, rather than an instance of the class:

```
class Negator:
    @singledispatchmethod
    @classmethod
    def neg(cls, arg):
        raise NotImplementedError("Cannot negate a"

    @neg.register
    @classmethod
    def _(cls, arg: int):
        return -arg

    @neg.register
    @classmethod
    def _(cls, arg: bool):
        return not arg
```

The same pattern can be used for other similar decorators:

`@staticmethod`, **`@~abc.abstractmethod`**, and others.

Added in version 3.8.

functools.update_wrapper(*wrapper*, *wrapped*, *assigned=WRAPPER_ASSIGNMENTS*, *updated=WRAPPER_UPDATES*)

Update a *wrapper* function to look like the *wrapped* function. The optional arguments are tuples to specify which attributes of the original function are assigned directly to the matching attributes on the wrapper function and which attributes of the wrapper function are updated with the corresponding attributes from the original function. The default values for these arguments are the module level constants `WRAPPER_ASSIGNMENTS` (which assigns to the wrapper function's **`__module__`**, **`__name__`**, **`__qualname__`**, **`__annotations__`**, **`__type_params__`**, and **`__doc__`**, the documentation string) and `WRAPPER_UPDATES` (which updates the wrapper function's **`__dict__`**, i.e. the instance dictionary).

Para permitir el acceso a la función original para la introspección y otros propósitos (por ejemplo, evitando un decorador de caché como **`lru_cache()`**), esta función añade automáticamente un atributo `__wrapped__` al envoltorio que se refiere a la función que se está envolviendo.

El principal uso previsto para esta función es en decorator functions que envuelven la función decorada y retornan el envoltorio. Si la función de envoltura no se actualiza, los metadatos de la función retornada reflejarán la definición de la envoltura en lugar de la definición de la función original, lo que normalmente no es de gran ayuda.

`update_wrapper()` puede ser usado con otros invocables que no sean funciones. Cualquier atributo nombrado en *assigned* o *updated* que falte en el objeto que se está invoca se ignora (es decir, esta función no intentará establecerlos en la función de envoltura (*wrapper*)). **`AttributeError`** sigue apareciendo si la propia función de envoltura no tiene ningún atributo nombrado en *updated*.

Distinto en la versión 3.2: The `__wrapped__` attribute is now automatically added. The **`__annotations__`** attribute is now copied by default. Missing attributes no longer trigger an **`AttributeError`**.

Distinto en la versión 3.4: El atributo `__wrapped__` ahora siempre se refiere a la función envuelta, incluso si esa función definió un atributo `__wrapped__`. (see bpo-17482 [https://bugs.python.org/issue?@action=redirect&bpo=17482])

Distinto en la versión 3.12: The **`__type_params__`** attribute is now copied by default.

@functools.wraps(*wrapped*, *assigned=WRAPPER_ASSIGNMENTS*, *updated=WRAPPER_UPDATES*)

Esta es una función conveniente para invocar **`update_wrapper()`** como decorador de la función cuando se define una función de envoltura (*wrapper*). Es equivalente a `partial(update_wrapper, wrapped=wrapped, assigned=assigned, updated=updated)`. Por ejemplo:

```
>>> from functools import wraps
>>> def my_decorator(f):
...     @wraps(f)
...     def wrapper(*args, **kwds):
...         print('Calling decorated function')
...         return f(*args, **kwds)
...     return wrapper
...
>>> @my_decorator
... def example():
...     """Docstring"""
...     print('Called example function')
...
>>> example()
Calling decorated function
Called example function
```

```
>>> example.__name__
'example'
>>> example.__doc__
'Docstring'
```

Without the use of this decorator factory, the name of the example function would have been `'wrapper'`, and the docstring of the original **`example()`** would have been lost.

`partial` Objetos

Los objetos `partial` son objetos invocables creados por `partial()`. Tienen tres atributos de sólo lectura:

partial.func

Un objeto o función invocable. Las llamadas al objeto `partial` serán reenviadas a `func` con nuevos argumentos y palabras clave.

partial.args

Los argumentos posicionales de la izquierda que se prepararán para los argumentos posicionales proporcionados un llamado al objeto `partial`.

partial.keywords

Los argumentos de la palabra clave que se suministrarán cuando se llame al objeto `partial`.

`partial` objects are like function objects in that they are callable, weak referenceable, and can have attributes. There are some important differences. For instance, the `__name__` and `__doc__` attributes are not created automatically.

`operator` — Standard operators as functions

Código fuente: Lib/operator.py [https://github.com/python/cpython/tree/3.14/Lib/operator.py]

The **`operator`** module exports a set of efficient functions corresponding to the intrinsic operators of Python. For example, `operator.add(x, y)` is equivalent to the expression `x+y`. Many function names are those used for special methods, without the double underscores. For backward compatibility, many of these have a variant with the double underscores kept. The variants without the double underscores are preferred for clarity.

Las funciones se dividen en categorías que realizan comparaciones de objetos, operaciones lógicas, operaciones matemáticas y operaciones sobre secuencias.

Las funciones de comparación de objetos son útiles para todos los objetos, y llevan el nombre de los operadores de comparación enriquecida que soportan:

operator.lt(*a*, *b*)
operator.le(*a*, *b*)
operator.eq(*a*, *b*)
operator.ne(*a*, *b*)
operator.ge(*a*, *b*)
operator.gt(*a*, *b*)
operator.__lt__(*a*, *b*)
operator.__le__(*a*, *b*)
operator.__eq__(*a*, *b*)
operator.__ne__(*a*, *b*)
operator.__ge__(*a*, *b*)

operator.__gt__(*a*, *b*)

Realiza «comparaciones enriquecidas» entre *a* y *b*. Específicamente, `lt(a, b)` es equivalente a `a < b`, `le(a, b)` es equivalente a `a <= b`, `eq(a, b)` es equivalente a `a == b`, `ne(a, b)` es equivalente a `a != b`, `gt(a, b)` es equivalente a `a > b` y `ge(a, b)` es equivalente a `a >= b`. Tenga en cuenta que estas funciones pueden retornar cualquier valor, que puede o no ser interpretable como un valor booleano. Consulte Comparaciones para obtener más información sobre las comparaciones enriquecidas.

Las operaciones lógicas también son aplicables a todos los objetos, y admiten pruebas de verdad, pruebas de identidad y operaciones booleanas:

operator.not_(*obj*)
operator.__not__(*obj*)

Retorna el resultado de **`not`** *obj*. (Tenga en cuenta que no hay ningún método **`__not__()`** para las instancias de objeto; solo el núcleo del intérprete define esta operación. El resultado se ve afectado por los métodos **`__bool__()`** y **`__len__()`**.)

operator.truth(*obj*)

Retorna **`True`** si *obj* es verdadero, y **`False`** de lo contrario. Esto equivale a usar el constructor **`bool`**.

operator.is_(*a*, *b*)

Retorna `a is b`. Chequea la identidad del objeto.

operator.is_not(*a*, *b*)

Retorna `a is not b`. Chequea la identidad del objeto.

operator.is_none(*a*)

Return `a is None`. Tests object identity.

Added in version 3.14.

operator.is_not_none(*a*)

Return `a is not None`. Tests object identity.

Added in version 3.14.

Las operaciones matemáticas y a nivel de bits son las más numerosas:

operator.abs(*obj*)
operator.__abs__(*obj*)

Retorna el valor absoluto de *obj*.

operator.add(*a*, *b*)
operator.__add__(*a*, *b*)

Retorna `a + b`, para los números *a* y *b*.

operator.and_(*a*, *b*)
operator.__and__(*a*, *b*)

Retorna la «conjunción bit a bit» (*bitwise and*) de *a* y *b*.

operator.floordiv(*a*, *b*)
operator.__floordiv__(*a*, *b*)

Retorna `a // b`.

operator.index(*a*)
operator.__index__(*a*)

Retorna *a* convertido a un número entero. Equivalente a `a.__index__()`.

Distinto en la versión 3.10: El resultado siempre tiene el tipo exacto **`int`**. Anteriormente, el resultado podría haber sido una instancia de una subclase de `int`.

operator.inv(*obj*)

operator.invert(*obj*)
operator.__inv__(*obj*)
operator.__invert__(*obj*)

Retorna el «inverso bit a bit» (*bitwise inverse*) del número *obj*. Esto es equivalente a `~obj`.

operator.lshift(*a*, *b*)
operator.__lshift__(*a*, *b*)

Retorna *a* desplazado a izquierda (*shift left*) *b* bits.

operator.mod(*a*, *b*)
operator.__mod__(*a*, *b*)

Retorna `a % b`.

operator.mul(*a*, *b*)
operator.__mul__(*a*, *b*)

Retorna `a * b`, para los números *a* y *b*.

operator.matmul(*a*, *b*)
operator.__matmul__(*a*, *b*)

Retorna `a @ b`.

Added in version 3.5.

operator.neg(*obj*)
operator.__neg__(*obj*)

Retorna *obj* negado (`-obj`).

operator.or_(*a*, *b*)
operator.__or__(*a*, *b*)

Retorna la «disyunción bit a bit» (*bitwise or*) de *a* y *b*.

operator.pos(*obj*)
operator.__pos__(*obj*)

Retorna *obj* positivo (`+obj`).

operator.pow(*a*, *b*)
operator.__pow__(*a*, *b*)

Retorna `a ** b`, para los números *a* y *b*.

operator.rshift(*a*, *b*)
operator.__rshift__(*a*, *b*)

Retorna *a* desplazado a derecha (*shift right*) *b* bits.

operator.sub(*a*, *b*)
operator.__sub__(*a*, *b*)

Retorna `a - b`.

operator.truediv(*a*, *b*)
operator.__truediv__(*a*, *b*)

Retorna `a / b` donde 2/3 es .66 en lugar de 0. Esto también se conoce como división «real» (*true division*).

operator.xor(*a*, *b*)
operator.__xor__(*a*, *b*)

Retorna la disyunción exclusiva bit a bit (*bitwise exclusive or*) de *a* y *b*.

Las operaciones que funcionan con secuencias (y algunas de ellas también con mapeos) incluyen:

operator.concat(*a*, *b*)
operator.__concat__(*a*, *b*)

Retorna `a + b` para las secuencias *a* y *b*.

operator.contains(*a*, *b*)
operator.__contains__(*a*, *b*)

Retorna el resultado del chequeo `b in a`. Notar que los operandos se invirtieron.

operator.countOf(*a*, *b*)

Retorna el número de ocurrencias de *b* en *a*.

operator.delitem(*a*, *b*)
operator.__delitem__(*a*, *b*)

Remueve el valor de *a* en el índice *b*.

operator.getitem(*a*, *b*)
operator.__getitem__(*a*, *b*)

Retorna el valor de *a* en el índice *b*.

operator.indexOf(*a*, *b*)

Retorna el índice de la primera ocurrencia de *b* en *a*.

operator.setitem(*a*, *b*, *c*)
operator.__setitem__(*a*, *b*, *c*)

Establece el valor de *a* en el índice *b* a *c*.

operator.length_hint(*obj*, *default=0*)

Retorna un largo estimativo del objeto *obj*. Primero intenta retornar su largo real, luego un estimativo usando **`object.__length_hint__()`**, y finalmente retorna un valor predeterminado.

Added in version 3.4.

La siguiente operación funciona con invocables:

operator.call(*obj*, */*, **args*, ***kwargs*)
operator.__call__(*obj*, */*, **args*, ***kwargs*)

Retorna `obj(*args, **kwargs)`.

Added in version 3.11.

The **`operator`** module also defines tools for generalized attribute and item lookups. These are useful for making fast field extractors as arguments for `map()`, `sorted()`, `itertools.groupby()`, or other functions that expect a function argument.

operator.attrgetter(*attr*)
operator.attrgetter(**attrs*)

Retorna un objeto invocable que obtiene *attr* de su operando. Si se solicita más de un atributo, retorna una tupla de los atributos. Los nombres de los atributos también pueden contener puntos. Por ejemplo:

- Después de `f = attrgetter('name')`, la llamada `f(b)` retorna `b.name`.
- Después de `f = attrgetter('name', 'date')`, la llamada `f(b)` retorna `(b.name, b.date)`.
- Después de `f = attrgetter('name.first', 'name.last')`, la llamada `f(b)` retorna `(b.name.first, b.name.last)`.

Equivalente a:

```
def attrgetter(*items):
    if any(not isinstance(item, str) for item in it
        raise TypeError('attribute name must be a s
    if len(items) == 1:
        attr = items[0]
        def g(obj):
            return resolve_attr(obj, attr)
    else:
        def g(obj):
            return tuple(resolve_attr(obj, attr) fo
    return g

def resolve_attr(obj, attr):
    for name in attr.split("."):
        obj = getattr(obj, name)
    return obj
```

operator.itemgetter(*item*)
operator.itemgetter(**items*)

Return a callable object that fetches *item* from its operand using the operand's `__getitem__()` method. If multiple items are specified, returns a tuple of lookup values. For example:

- Después de `f = itemgetter(2)`, la llamada `f(r)` retorna `r[2]`.
- Después de `g = itemgetter(2, 5, 3)`, la llamada `g(r)` retorna `(r[2], r[5], r[3])`.

Equivalente a:

```
def itemgetter(*items):
    if len(items) == 1:
        item = items[0]
        def g(obj):
            return obj[item]
    else:
        def g(obj):
            return tuple(obj[item] for item in item
    return g
```

The items can be any type accepted by the operand's `__getitem__()` method. Dictionaries accept any hashable value. Lists, tuples, and strings accept an index or a slice:

```
>>> itemgetter(1)('ABCDEFG')
'B'
>>> itemgetter(1, 3, 5)('ABCDEFG')
('B', 'D', 'F')
>>> itemgetter(slice(2, None))('ABCDEFG')
'CDEFG'
>>> soldier = dict(rank='captain', name='dotterbart
>>> itemgetter('rank')(soldier)
'captain'
```

Ejemplos que utilizan `itemgetter()` para obtener campos específicos de un registro tupla:

```
>>> inventory = [('apple', 3), ('banana', 2), ('pea
>>> getcount = itemgetter(1)
>>> list(map(getcount, inventory))
[3, 2, 5, 1]
>>> sorted(inventory, key=getcount)
[('orange', 1), ('banana', 2), ('apple', 3), ('pear
```

operator.methodcaller(*name*, /, **args*, ***kwargs*)

Retorna un objeto invocable que a su vez invoca al método *name* sobre su operando. Si se pasan argumentos adicionales y/o argumentos por palabra clave, estos serán a su vez pasados al método. Por ejemplo:

- Después de `f = methodcaller('name')`, la llamada `f(b)` retorna `b.name()`.
- Después de `f = methodcaller('name', 'foo', bar=1)`, la llamada `f(b)` retorna `b.name('foo', bar=1)`.

Equivalente a:

```
def methodcaller(name, /, *args, **kwargs):
    def caller(obj):
        return getattr(obj, name)(*args, **kwargs)
    return caller
```

Asignación de operadores a funciones

This table shows how abstract operations correspond to operator symbols in the Python syntax and the functions in the **`operator`** module.

Operación	**Sintaxis**	**Función**
Adición	`a + b`	`add(a, b)`
Concatenación	`seq1 + seq2`	`concat(seq1, seq2)`
Chequeo de pertenencia	`obj in seq`	`contains(seq, obj)`
División	`a / b`	`truediv(a, b)`
División	`a // b`	`floordiv(a, b)`
Conjunción lógica bit a bit (*bitwise and*)	`a & b`	`and_(a, b)`
Disyunción lógica bit a bit (*bitwise exclusive or*)	`a ^ b`	`xor(a, b)`

Inversión bit a bit (*bitwise inversion*)	`invert(a)`
Disyunción lógica bit a bit (*bitwise or*)	`or_(a, b)`
Exponenciación	`pow(a, b)`
Identidad	`is_(a, b)`
Identidad	`is_not(a, b)`
Identidad	`is_none(a)`
Identidad	`is_not_none(a)`
Asignación indexada	`setitem(obj, k, v)`
Eliminación indexada	`delitem(obj, k)`
Indexado	`getitem(obj, k)`
Desplazamiento a izquierda (*left shift*)	`lshift(a, b)`
Módulo	`mod(a, b)`
Multiplicación	`mul(a, b)`
Multiplicación de matrices	`matmul(a, b)`
Negación (aritmética)	`neg(a)`
Negación (lógica)	`not_(a)`
Positivo	`pos(a)`
Desplazamiento a derecha (*right shift*)	`rshift(a, b)`
Asignación por segmento	`setitem(seq, slice(i, j), values)`
Eliminación por segmento	`delitem(seq, slice(i, j))`
Segmentación	`getitem(seq, slice(i, j))`
Formateo de cadenas	`mod(s, obj)`
Sustracción	`sub(a, b)`
Chequeo de verdad	`truth(obj)`
Ordenado	`lt(a, b)`
Ordenado	`le(a, b)`
Igualdad	`eq(a, b)`
Diferencia	`ne(a, b)`
Ordenado	`ge(a, b)`
Ordenado	`gt(a, b)`

Operadores *In-place*

Muchas operaciones tienen una versión *«in-place»*. Abajo se listan las funciones que proveen un acceso más primitivo a operadores *in-place* que la sintaxis usual; por ejemplo, el statement `x += y` es equivalente a `x = operator.iadd(x, y)`. Otra forma de decirlo es que `z = operator.iadd(x, y)` es equivalente a la sentencia (*statement*) compuesta `z = x; z += y`.

En esos ejemplo, notar que cuando se invoca un método *in-place*, el cómputo y la asignación se realizan en dos pasos separados. Las funciones *in-place* que se listan aquí debajo solo hacen el primer paso, llamar al método *in-place*. El segundo paso, la asignación, no se gestiona.

Para objetivos inmutables como cadenas de caracteres, números, y tuplas, el valor actualizado es computado, pero no es asignado de nuevo a la variable de entrada:

```
>>> a = 'hello'
>>> iadd(a, ' world')
'hello world'
>>> a
'hello'
```

Para objetivos mutables como listas y diccionarios, el método *in-place* realiza la actualización, así que no es necesaria una asignación subsiguiente:

```
>>> s = ['h', 'e', 'l', 'l', 'o']
>>> iadd(s, [' ', 'w', 'o', 'r', 'l', 'd'])
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
>>> s
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
```

operator.iadd(*a*, *b*)
operator._iadd_(*a*, *b*)

`a = iadd(a, b)` es equivalente a `a += b`.

operator.iand(*a*, *b*)
operator._iand_(*a*, *b*)

`a = iand(a, b)` es equivalente a `a &= b`.

operator.iconcat(*a*, *b*)
operator._iconcat_(*a*, *b*)

`a = iconcat(a, b)` es equivalente a `a += b` para dos *a* y *b* secuencias.

operator.ifloordiv(*a*, *b*)
operator.__ifloordiv__(*a*, *b*)

`a = ifloordiv(a, b)` es equivalente a `a //= b`.

operator.ilshift(*a*, *b*)
operator.__ilshift__(*a*, *b*)

`a = ilshift(a, b)` es equivalente a `a <<= b`.

operator.imod(*a*, *b*)
operator.__imod__(*a*, *b*)

`a = imod(a, b)` es equivalente a `a %= b`.

operator.imul(*a*, *b*)
operator.__imul__(*a*, *b*)

`a = imul(a, b)` es equivalente a `a *= b`.

operator.imatmul(*a*, *b*)
operator.__imatmul__(*a*, *b*)

`a = imul(a, b)` es equivalente a `a *= b`.

Added in version 3.5.

operator.ior(*a*, *b*)
operator.__ior__(*a*, *b*)

`a = ior(a, b)` es equivalente a `a |= b`.

operator.ipow(*a*, *b*)
operator.__ipow__(*a*, *b*)

`a = ipow(a, b)` es equivalente a `a **= b`.

operator.irshift(*a*, *b*)
operator.__irshift__(*a*, *b*)

`a = irshift(a, b)` es equivalente a `a >>= b`.

operator.isub(*a*, *b*)
operator.__isub__(*a*, *b*)

`a = isub(a, b)` es equivalente a `a -= b`.

operator.itruediv(*a*, *b*)
operator.__itruediv__(*a*, *b*)

`a = itruediv(a, b)` es equivalente a `a /= b`.

operator.ixor(*a*, *b*)
operator.__ixor__(*a*, *b*)

`a = ixor(a, b)` es equivalente a `a ^= b`.

Acceso a archivos y directorios

Los módulos descritos en este capítulo tratan de archivos de disco y directorios. Por ejemplo, hay módulos para leer las propiedades de los archivos, manipular rutas de acceso de forma portátil y crear archivos temporales. La lista completa de módulos en este capítulo es:

- **`pathlib`** — Rutas de sistemas orientada a objetos
  - Uso básico
  - Excepciones
  - Rutas puras
    - Propiedades generales
    - Operadores
    - Acceso a partes individuales
    - Métodos y propiedades
  - Rutas concretas
    - Análisis y generación de URI
    - Ampliando y resolviendo caminos
    - Consulta de tipo de archivo y estado
    - Lectura y escritura de archivos
    - Directorios de lectura
    - Creando archivos y directorios
    - Copying, moving and deleting
    - Permisos y propiedad
  - Lenguaje de patrones
  - Comparación con el módulo **`glob`**
  - Comparación con los módulos **`os`** y **`os.path`**
    - Herramientas correspondientes
  - Protocols

- `os.path` — Common pathname manipulations
- `stat` — Interpreting `stat()` results
- `filecmp` — File and Directory Comparisons
    - La clase `dircmp`
- `tempfile` — Generate temporary files and directories
    - Ejemplos
    - Funciones y variables deprecadas
- `glob` — Unix style pathname pattern expansion
    - Examples
- `fnmatch` — Unix filename pattern matching
- `linecache` — Random access to text lines
- `shutil` — High-level file operations
    - Operaciones de directorios y archivos
        - Operaciones de copia eficientes dependientes de la plataforma
        - ejemplo de *copytree*
        - ejemplo de rmtree
    - Operaciones de archivado
        - Ejemplo de archivado
        - Ejemplo de archivado con *base_dir*
    - Consulta el tamaño de la terminal de salida

Ver también

Módulo `os`

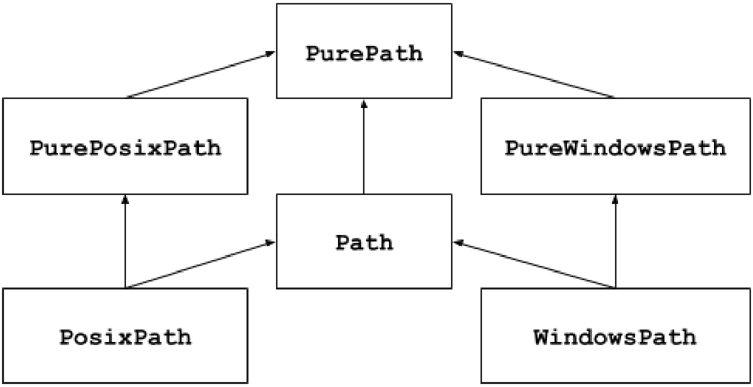
Interfaces del sistema operativo, incluidas las funciones para trabajar con archivos en un nivel inferior al de Python file objects.

Módulo `io`

La biblioteca de I/O integrada de Python, incluidas las clases abstractas y algunas clases concretas, como la I/O de archivos.

Función incorporada `open()`

La forma estándar de abrir archivos para leer y escribir con Python.



[1]

[2] [3] [4]

`pathlib` — Rutas de sistemas orientada a objetos

Added in version 3.4.

Código fuente: Lib/pathlib/ [https://github.com/python/cpython/tree/3.14/Lib/pathlib/]

Este módulo ofrece clases que representan rutas del sistema de archivos con semántica apropiada para diferentes sistemas operativos. Las clases ruta se dividen entre rutas puras, que proporcionan operaciones puramente computacionales sin E/S; y rutas concretas, que heredan de rutas puras pero también proporcionan operaciones de E/S.

Si nunca has usado este módulo o simplemente no estás seguro de qué clase es la adecuada para tu tarea, `Path` es probablemente lo que necesitas. Crea una instancia ruta concreta para la plataforma en la que se ejecuta el código.

Las rutas puras son útiles en algunos casos especiales, por ejemplo:

1. Si deseas manipular las rutas de Windows en una máquina Unix (o viceversa). No puedes crear una instancia de **`WindowsPath`** cuando se ejecuta en Unix, pero puedes crear una instancia de **`PureWindowsPath`**.
2. Desea asegurar que su código solo manipule rutas sin acceder realmente al sistema operativo. En este caso, crear instancias de una de las clases puras puede ser útil, ya que simplemente no tienen ninguna operación de acceso al sistema operativo.

Ver también

PEP 428 [https://peps.python.org/pep-0428/]: El módulo pathlib – rutas de sistema orientadas a objetos.

Ver también

Para la manipulación de rutas de bajo nivel en cadenas, también puede usar el módulo **`os.path`**.

Uso básico

Importar la clase principal:

```
>>> from pathlib import Path
```

Listado de subdirectorios:

```
>>> p = Path('.')
>>> [x for x in p.iterdir() if x.is_dir()]
[PosixPath('.hg'), PosixPath('docs'), PosixPath('dist'),
 PosixPath('__pycache__'), PosixPath('build')]
```

Listado de archivos fuente de Python en este árbol de directorios:

```
>>> list(p.glob('**/*.py'))
[PosixPath('test_pathlib.py'), PosixPath('setup.py'),
 PosixPath('pathlib.py'), PosixPath('docs/conf.py'),
 PosixPath('build/lib/pathlib.py')]
```

Navegar dentro de un árbol de directorios:

```
>>> p = Path('/etc')
>>> q = p / 'init.d' / 'reboot'
>>> q
PosixPath('/etc/init.d/reboot')
>>> q.resolve()
PosixPath('/etc/rc.d/init.d/halt')
```

Consultar propiedades de ruta:

```
>>> q.exists()
True
>>> q.is_dir()
False
```

Abrir un archivo:

```
>>> with q.open() as f: f.readline()
...
'#!/bin/bash\n'
```

Excepciones

exception pathlib.UnsupportedOperation

Una excepción que hereda `NotImplementedError` y que se genera cuando se llama a una operación no compatible en un objeto de ruta.

Added in version 3.13.

Rutas puras

Los objetos ruta pura proporcionan operaciones de manejo de rutas que en realidad no acceden al sistema de archivos. Hay tres formas de acceder a estas clases, que llamaremos *familias*:

class pathlib.PurePath(**pathsegments*)

Una clase genérica que representa la familia de rutas del sistema (al crear una instancia se crea **`PurePosixPath`** o **`PureWindowsPath`**):

```
>>> PurePath('setup.py')      # Ejecutándose en una
PurePosixPath('setup.py')
```

Cada elemento de *pathsegments* puede ser una cadena que representa un segmento de ruta o un objeto que implemente la interfaz **`os.PathLike`** donde el método **`__fspath__()`** retorna una cadena como otro objeto de ruta:

```
>>> PurePath('foo', 'some/path', 'bar')
PurePosixPath('foo/some/path/bar')
>>> PurePath(Path('foo'), Path('bar'))
PurePosixPath('foo/bar')
```

Cuando *pathsegments* está vacío, se asume el directorio actual:

```
>>> PurePath()
PurePosixPath('.')
```

Si un segmento es una ruta absoluta, se ignoran todos los segmentos anteriores (como **`os.path.join()`**):

```
>>> PurePath('/etc', '/usr', 'lib64')
PurePosixPath('/usr/lib64')
>>> PureWindowsPath('c:/Windows', 'd:bar')
PureWindowsPath('d:bar')
```

En Windows, la unidad no se restablece cuando se encuentra un segmento de ruta relativa en la raíz (por ejemplo, `r'\foo'`):

```
>>> PureWindowsPath('c:/Windows', '/Program Files')
PureWindowsPath('c:/Program Files')
```

Las barras espurias y los puntos simples se colapsan, pero los puntos dobles (`'..'`) y las barras dobles iniciales (`'//'`) no, ya que esto cambiaría el significado de una ruta por varias razones (por ejemplo, enlaces simbólicos, rutas UNC):

```
>>> PurePath('foo//bar')
PurePosixPath('foo/bar')
>>> PurePath('//foo/bar')
PurePosixPath('//foo/bar')
>>> PurePath('foo/./bar')
PurePosixPath('foo/bar')
>>> PurePath('foo/../bar')
PurePosixPath('foo/../bar')
```

(un enfoque naif haría pensar que `PurePosixPath('foo/../bar')` es equivalente a `PurePosixPath('bar')`, lo cual es incorrecto si `foo` es un enlace simbólico a otro directorio)

Los objetos ruta pura implementan la interfaz **`os.PathLike`**, lo que permite su uso en cualquier lugar donde se acepte la interfaz.

Distinto en la versión 3.6: Se agregó soporte para la interfaz **`os.PathLike`**.

class pathlib.PurePosixPath(**pathsegments*)

Una subclase de **`PurePath`**, esta *familia* representa rutas que no son del sistema de archivos de Windows:

```
>>> PurePosixPath('/etc/hosts')
PurePosixPath('/etc/hosts')
```

pathsegments se especifica de manera similar a **`PurePath`**.

class pathlib.PureWindowsPath(**pathsegments*)

Una subclase de **`PurePath`**, este tipo de ruta representa las rutas del sistema de archivos de Windows, incluido UNC paths [https://en.wikipedia.org/wiki/Path_(computing)#UNC]:

```
>>> PureWindowsPath('c:/', 'Users', 'Ximénez')
PureWindowsPath('c:/Users/Ximénez')
>>> PureWindowsPath('//server/share/file')
PureWindowsPath('//server/share/file')
```

pathsegments se especifica de manera similar a **`PurePath`**.

Independientemente del sistema en el que se encuentre, puede crear instancias de todas estas clases, ya que no proporcionan ninguna operación que llame al sistema operativo.

Propiedades generales

Las rutas son inmutables y hashable. Las rutas de una misma familia son comparables y ordenables. Estas propiedades respetan el orden lexicográfico definido por la familia:

```
>>> PurePosixPath('foo') == PurePosixPath('FOO')
False
>>> PureWindowsPath('foo') == PureWindowsPath('FOO')
True
>>> PureWindowsPath('FOO') in { PureWindowsPath('foo') }
True
>>> PureWindowsPath('C:') < PureWindowsPath('d:')
True
```

Rutas de diferentes familias no son iguales y no se pueden ordenar:

```
>>> PureWindowsPath('foo') == PurePosixPath('foo')
False
>>> PureWindowsPath('foo') < PurePosixPath('foo')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'PureW
```

Operadores

El operador de barra inclinada ayuda a crear rutas hijas, como **`os.path.join()`**. Si el argumento es una ruta absoluta, se ignora la ruta anterior. En Windows, la unidad no se restablece cuando el argumento es una ruta relativa en la raíz (por ejemplo, `r'\foo'`):

```
>>> p = PurePath('/etc')
>>> p
PurePosixPath('/etc')
```

```
>>> p / 'init.d' / 'apache2'
PurePosixPath('/etc/init.d/apache2')
>>> q = PurePath('bin')
>>> '/usr' / q
PurePosixPath('/usr/bin')
>>> p / '/an_absolute_path'
PurePosixPath('/an_absolute_path')
>>> PureWindowsPath('c:/Windows', '/Program Files')
PureWindowsPath('c:/Program Files')
```

Se puede usar un objeto ruta en cualquier lugar donde se acepte un objeto que implemente `os.PathLike`:

```
>>> import os
>>> p = PurePath('/etc')
>>> os.fspath(p)
'/etc'
```

La representación de una ruta en una cadena de caracteres es la ruta del sistema de archivos sin procesar en sí (en forma nativa, por ejemplo con barras invertidas en Windows), y que puede pasar a cualquier función que tome una ruta como una cadena de caracteres:

```
>>> p = PurePath('/etc')
>>> str(p)
'/etc'
>>> p = PureWindowsPath('c:/Program Files')
>>> str(p)
'c:\\Program Files'
```

Del mismo modo, llamar a `bytes` en una ruta proporciona la ruta cruda del sistema de archivos como un objeto bytes, codificado por `os.fsencode()`:

```
>>> bytes(p)
b'/etc'
```

Nota

Llamar a `bytes` solo se recomienda en Unix. En Windows, la forma unicode es la representación canónica de las rutas del sistema de archivos.

Acceso a partes individuales

Para acceder a las «partes» (componentes) individuales de una ruta, use la siguiente propiedad:

PurePath.parts

Una tupla que da acceso a los diversos componentes de la ruta:

```
>>> p = PurePath('/usr/bin/python3')
>>> p.parts
('/', 'usr', 'bin', 'python3')

>>> p = PureWindowsPath('c:/Program Files/PSF')
>>> p.parts
('c:\\', 'Program Files', 'PSF')
```

(obsérvese cómo la unidad y la raíz local se reagrupan en una sola parte)

Métodos y propiedades

Las rutas puras proporcionan los siguientes métodos y propiedades:

PurePath.parser

The implementation of the `os.path` module used for low-level path parsing and joining: either **`posixpath`** or **`ntpath`**.

Added in version 3.13.

PurePath.drive

Una cadena que representa la letra o el nombre de la unidad, si corresponde:

```
>>> PureWindowsPath('c:/Program Files/').drive
'c:'
>>> PureWindowsPath('/Program Files/').drive
''
>>> PurePosixPath('/etc').drive
''
```

Las localizaciones UNC también son consideradas como unidades:

```
>>> PureWindowsPath('//host/share/foo.txt').drive
'\\\\host\\share'
```

PurePath.root

Una cadena que representa la raíz (local o global), si la hay:

```
>>> PureWindowsPath('c:/Program Files/').root
'\\'
>>> PureWindowsPath('c:Program Files/').root
''
>>> PurePosixPath('/etc').root
'/'
```

Las localizaciones UNC siempre tienen una raíz:

```
>>> PureWindowsPath('//host/share').root
'\\'
```

Si la ruta comienza con más de dos barras diagonales sucesivas, `PurePosixPath` las contrae:

```
>>> PurePosixPath('//etc').root
'//'
>>> PurePosixPath('///etc').root
'/'
>>> PurePosixPath('////etc').root
'/'
```

Nota

Este comportamiento se ajusta a *The Open Group Base Specifications Issue 6*, párrafo 4.11 Pathname Resolution [https://pubs.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap04.html#tag_04_11]:

«Un nombre de ruta que comienza con dos barras oblicuas sucesivas se puede interpretar de una manera definida por la implementación, aunque más de dos barras oblicuas iniciales se tratarán como una sola barra oblicua.»

PurePath.anchor

La concatenación de la unidad y la raíz:

```
>>> PureWindowsPath('c:/Program Files/').anchor
'c:\\'
>>> PureWindowsPath('c:Program Files/').anchor
'c:'
>>> PurePosixPath('/etc').anchor
'/'
>>> PureWindowsPath('//host/share').anchor
'\\\\host\\share\\'
```

PurePath.parents

Una secuencia inmutable que proporciona acceso a los ancestros lógicos de la ruta:

```
>>> p = PureWindowsPath('c:/foo/bar/setup.py')
>>> p.parents[0]
PureWindowsPath('c:/foo/bar')
>>> p.parents[1]
PureWindowsPath('c:/foo')
>>> p.parents[2]
PureWindowsPath('c:/')
```

Distinto en la versión 3.10: La secuencia de padres ahora admite rebanada y valores de índice negativos.

PurePath.parent

El padre lógico de la ruta

```
>>> p = PurePosixPath('/a/b/c/d')
>>> p.parent
PurePosixPath('/a/b/c')
```

No puede ir más allá de un ancla o una ruta vacía:

```
>>> p = PurePosixPath('/')
>>> p.parent
PurePosixPath('/')
>>> p = PurePosixPath('.')
>>> p.parent
PurePosixPath('.')
```

Nota

Esta es una operación puramente léxica, de ahí el siguiente comportamiento:

```
>>> p = PurePosixPath('foo/..')
>>> p.parent
PurePosixPath('foo')
```

Si desea recorrer una ruta de sistema de archivos arbitraria hacia arriba, se recomienda llamar primero a `Path.resolve()` para resolver los enlaces simbólicos y eliminar los componentes `".."`.

PurePath.name

Una cadena que representa el componente final de la ruta, excluyendo la unidad y la raíz, si hay alguna:

```
>>> PurePosixPath('my/library/setup.py').name
'setup.py'
```

Los nombres de unidad UNC no se consideran:

```
>>> PureWindowsPath('//some/share/setup.py').name
```

```
'setup.py'
>>> PureWindowsPath('//some/share').name
''
```

PurePath.suffix

La última porción separada por puntos del componente final, si la hay:

```
>>> PurePosixPath('my/library/setup.py').suffix
'.py'
>>> PurePosixPath('my/library.tar.gz').suffix
'.gz'
>>> PurePosixPath('my/library').suffix
''
```

Esto comúnmente se llama extensión de archivo.

Distinto en la versión 3.14: A single dot (» `.` ») is considered a valid suffix.

PurePath.suffixes

Una lista de los sufijos de la ruta, a menudo llamados extensiones de archivo:

```
>>> PurePosixPath('my/library.tar.gar').suffixes
['.tar', '.gar']
>>> PurePosixPath('my/library.tar.gz').suffixes
['.tar', '.gz']
>>> PurePosixPath('my/library').suffixes
[]
```

Distinto en la versión 3.14: A single dot (» `.` ») is considered a valid suffix.

PurePath.stem

El componente final de la ruta, sin su sufijo:

```
>>> PurePosixPath('my/library.tar.gz').stem
'library.tar'
```

```
>>> PurePosixPath('my/library.tar').stem
'library'
>>> PurePosixPath('my/library').stem
'library'
```

Distinto en la versión 3.14: A single dot (» . «) is considered a valid suffix.

PurePath.as_posix()

Retorna una cadena que representa la ruta con barras invertidas (/):

```
>>> p = PureWindowsPath('c:\\windows')
>>> str(p)
'c:\\windows'
>>> p.as_posix()
'c:/windows'
```

PurePath.is_absolute()

Retorna si la ruta es absoluta o no. Una ruta se considera absoluta si tiene una raíz y (si la **familia** lo permite) una unidad:

```
>>> PurePosixPath('/a/b').is_absolute()
True
>>> PurePosixPath('a/b').is_absolute()
False

>>> PureWindowsPath('c:/a/b').is_absolute()
True
>>> PureWindowsPath('/a/b').is_absolute()
False
>>> PureWindowsPath('c:').is_absolute()
False
>>> PureWindowsPath('//some/share').is_absolute()
True
```

PurePath.is_relative_to(*other*)

Retorna si esta ruta es relativa o no a la *otra* ruta.

```
>>> p = PurePath('/etc/passwd')
>>> p.is_relative_to('/etc')
True
>>> p.is_relative_to('/usr')
False
```

Este método se basa en cadenas; no accede al sistema de archivos ni trata los segmentos «`..`» de manera especial. El siguiente código es equivalente:

```
>>> u = PurePath('/usr')
>>> u == p or u in p.parents
False
```

Added in version 3.9.

Deprecated since version 3.12, removed in version 3.14: Está obsoleto pasar argumentos adicionales; si se suministran, se juntan con *other*.

PurePath.is_reserved()

Con **`PureWindowsPath`**, retorna `True` si la ruta se considera reservada en Windows, `False` en caso contrario. Con **`PurePosixPath`**, siempre retorna `False`.

Distinto en la versión 3.13: Los nombres de ruta de Windows que contienen dos puntos o terminan con un punto o un espacio se consideran reservados. Las rutas UNC pueden estar reservadas.

Deprecated since version 3.13, will be removed in version 3.15: Este método está obsoleto; utilice **`os.path.isreserved()`** para detectar rutas reservadas en Windows.

PurePath.joinpath(**pathsegments*)

Llamar a este método es equivalente a combinar la ruta con cada uno de los *pathsegments* dados sucesivamente:

```
>>> PurePosixPath('/etc').joinpath('passwd')
PurePosixPath('/etc/passwd')
>>> PurePosixPath('/etc').joinpath(PurePosixPath('p
PurePosixPath('/etc/passwd')
>>> PurePosixPath('/etc').joinpath('init.d', 'apach
PurePosixPath('/etc/init.d/apache2')
>>> PureWindowsPath('c:').joinpath('/Program Files'
PureWindowsPath('c:/Program Files')
```

PurePath.full_match(*pattern*, *, *case_sensitive*=*None*)

Compare esta ruta con el patrón de estilo glob proporcionado. Devuelva `True` si la coincidencia es exitosa, `False` en caso contrario. Por ejemplo:

```
>>> PurePath('a/b.py').full_match('a/*.py')
True
>>> PurePath('a/b.py').full_match('*.py')
False
>>> PurePath('/a/b/c.py').full_match('/a/**')
True
>>> PurePath('/a/b/c.py').full_match('**/*.py')
True
```

Ver también

Documentación Lenguaje de patrones.

Al igual que con otros métodos, la distinción entre mayúsculas y minúsculas sigue los valores predeterminados de la plataforma:

```
>>> PurePosixPath('b.py').full_match('*.PY')
False
>>> PureWindowsPath('b.py').full_match('*.PY')
True
```

Establece *case_sensitive* en `True` o `False` para invalidar este comportamiento.

Added in version 3.13.

PurePath.match(*pattern*, *, *case_sensitive=None*)

Compare esta ruta con el patrón de estilo glob no recursivo proporcionado. Devuelva `True` si la coincidencia es exitosa, `False` en caso contrario.

Este método es similar a **`full_match()`**, pero no se permiten patrones vacíos (se genera **`ValueError`**), no se admite el comodín recursivo «`**`» (actúa como «`*`» no recursivo) y, si se proporciona un patrón relativo, la coincidencia se realiza desde la derecha:

```
>>> PurePath('a/b.py').full_match('a/*.py')
True
>>> PurePath('a/b.py').full_match('*.py')
False
>>> PurePath('/a/b/c.py').full_match('/a/**')
True
>>> PurePath('/a/b/c.py').full_match('**/*.py')
True
```

Distinto en la versión 3.12: El parámetro *pattern* acepta un path-like object.

Distinto en la versión 3.12: Se agregó el parámetro *case_sensitive*.

PurePath.relative_to(*other*, *walk_up=False*)

Computa una versión de esta ruta relativa a la ruta representada por *other*. Si es imposible, se lanza **`ValueError`**:

```
>>> p = PurePosixPath('/etc/passwd')
>>> p.relative_to('/')
PurePosixPath('etc/passwd')
>>> p.relative_to('/etc')
PurePosixPath('passwd')
>>> p.relative_to('/usr')
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "pathlib.py", line 941, in relative_to
    raise ValueError(error_message.format(str(self)
ValueError: '/etc/passwd' is not in the subpath of
```

Cuando *walk_up* es falso (el valor predeterminado), la ruta debe comenzar con *other*. Cuando el argumento es verdadero, se pueden agregar entradas `..` para formar la ruta relativa. En todos los demás casos, como las rutas que hacen referencia a diferentes unidades, se genera **`ValueError`**.:

```
>>> p.relative_to('/usr', walk_up=True)
PurePosixPath('../etc/passwd')
>>> p.relative_to('foo', walk_up=True)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "pathlib.py", line 941, in relative_to
    raise ValueError(error_message.format(str(self)
ValueError: '/etc/passwd' is not on the same drive
```

Advertencia

Esta función es parte de **`PurePath`** y trabaja con cadenas de caracteres. No revisa ni accede a la estructura de archivos subyacentes. Esto puede afectar la opción *walk_up* ya que supone que no hay enlaces simbólicos en la ruta; si es necesario llame primero **`resolve()`** para resolver los enlaces simbólicos.

Distinto en la versión 3.12: Se agregó el parámetro *walk_up* (el comportamiento anterior es el mismo que `walk_up=False`).

Deprecated since version 3.12, removed in version 3.14: Está obsoleto pasar argumentos posicionales adicionales; si se suministran, se juntan con *other*.

PurePath.with_name(*name*)

Retorna una nueva ruta con **`name`** cambiado. Si la ruta original no tiene nombre, se genera *ValueError*:

```
>>> p = PureWindowsPath('c:/Downloads/pathlib.tar.g
>>> p.with_name('setup.py')
PureWindowsPath('c:/Downloads/setup.py')
>>> p = PureWindowsPath('c:/')
>>> p.with_name('setup.py')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/antoine/cpython/default/Lib/pathlib.p
    raise ValueError("%r has an empty name" % (self
ValueError: PureWindowsPath('c:/') has an empty nam
```

PurePath.with_stem(*stem*)

Retorna una nueva ruta con `stem` cambiado. Si la ruta original no tiene nombre, se lanza *ValueError*:

```
>>> p = PureWindowsPath('c:/Downloads/draft.txt')
>>> p.with_stem('final')
PureWindowsPath('c:/Downloads/final.txt')
>>> p = PureWindowsPath('c:/Downloads/pathlib.tar.g
>>> p.with_stem('lib')
PureWindowsPath('c:/Downloads/lib.gz')
>>> p = PureWindowsPath('c:/')
>>> p.with_stem('')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/antoine/cpython/default/Lib/pathlib.p
    return self.with_name(stem + self.suffix)
  File "/home/antoine/cpython/default/Lib/pathlib.p
    raise ValueError("%r has an empty name" % (self
ValueError: PureWindowsPath('c:/') has an empty nam
```

Added in version 3.9.

PurePath.with_suffix(*suffix*)

Retorna una nueva ruta con `suffix` cambiado. Si la ruta original no tiene un sufijo, se agrega el nuevo *suffix*. Si *suffix* es una cadena vacía, el sufijo original se elimina:

```
>>> p = PureWindowsPath('c:/Downloads/pathlib.tar.g
>>> p.with_suffix('.bz2')
PureWindowsPath('c:/Downloads/pathlib.tar.bz2')
>>> p = PureWindowsPath('README')
>>> p.with_suffix('.txt')
PureWindowsPath('README.txt')
>>> p = PureWindowsPath('README.txt')
>>> p.with_suffix('')
PureWindowsPath('README')
```

Distinto en la versión 3.14: A single dot (» `.` ») is considered a valid suffix. In previous versions, **`ValueError`** is raised if a single dot is supplied.

PurePath.with_segments(*pathsegments)

Crea un nuevo objeto de ruta del mismo tipo combinando los *pathsegments* dados. Este método se llama siempre que se crea una ruta derivada, como **`parent`** y **`relative_to()`**. Las subclases pueden anular este método para pasar información a las rutas derivadas, por ejemplo:

```
from pathlib import PurePosixPath

class MyPath(PurePosixPath):
    def __init__(self, *pathsegments, session_id):
        super().__init__(*pathsegments)
        self.session_id = session_id

    def with_segments(self, *pathsegments):
        return type(self)(*pathsegments, session_id

etc = MyPath('/etc', session_id=42)
hosts = etc / 'hosts'
print(hosts.session_id)  # 42
```

Added in version 3.12.

Rutas concretas

Las rutas concretas son subclases de las rutas puras. Además de las operaciones proporcionadas por estas últimas, también proporcionan métodos realizar llamadas del sistema a objetos ruta. Hay tres formas de crear instancias de rutas concretas:

class pathlib.Path(**pathsegments*)

Una subclase de `PurePath`, esta clase representa rutas concretas de la *familia* ruta del sistema (al crear una instancia crea ya sea `PosixPath` o `WindowsPath`):

```
>>> Path('setup.py')
PosixPath('setup.py')
```

pathsegments se especifica de manera similar a `PurePath`.

class pathlib.PosixPath(**pathsegments*)

Una subclase de `Path` y `PurePosixPath`, esta clase representa rutas concretas de sistemas de archivos que no son de Windows:

```
>>> PosixPath('/etc/hosts')
PosixPath('/etc/hosts')
```

pathsegments se especifica de manera similar a `PurePath`.

Distinto en la versión 3.13: Genera el error `UnsupportedOperation` en Windows. En versiones anteriores, se generaba el error `NotImplementedError`.

class pathlib.WindowsPath(**pathsegments*)

Una subclase de `Path` y `PureWindowsPath`, esta clase representa rutas concretas del sistema de archivos de Windows:

```
>>> WindowsPath('c:/', 'Users', 'Ximénez')
WindowsPath('c:/Users/Ximénez')
```

pathsegments se especifica de manera similar a **`PurePath`**.

Distinto en la versión 3.13: Genera el error **`UnsupportedOperation`** en plataformas que no sean Windows. En versiones anteriores, se generaba el error **`NotImplementedError`**.

Solo puedes crear instancias de la *familia* de clase que corresponde a su sistema operativo (permitir llamadas del sistema no compatibles podría provocar errores o fallas en su aplicación):

```
>>> import os
>>> os.name
'posix'
>>> Path('setup.py')
PosixPath('setup.py')
>>> PosixPath('setup.py')
PosixPath('setup.py')
>>> WindowsPath('setup.py')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "pathlib.py", line 798, in __new__
    % (cls.__name__,))
UnsupportedOperation: cannot instantiate 'WindowsPath'
```

Algunos métodos de ruta concretos pueden generar un **`OSError`** si falla una llamada del sistema (por ejemplo, porque la ruta no existe).

Análisis y generación de URI

Los objetos de ruta concretos se pueden crear y representar como URI de «archivo» que cumplan con **RFC 8089** [https://datatracker.ietf.org/doc/html/rfc8089.html].

Nota

Los URI de archivos no son portables entre máquinas con diferentes filesystem encodings.

classmethod Path.from_uri(*uri*)

Devuelve un nuevo objeto de ruta a partir del análisis de una URL de «archivo». Por ejemplo:

```
>>> p = Path.from_uri('file:///etc/hosts')
PosixPath('/etc/hosts')
```

En Windows, los dispositivos DOS y las rutas UNC se pueden analizar desde las URI:

```
>>> p = Path.from_uri('file:///c:/windows')
WindowsPath('c:/windows')
>>> p = Path.from_uri('file://server/share')
WindowsPath('//server/share')
```

Se admiten varias formas variantes:

```
>>> p = Path.from_uri('file:////server/share')
WindowsPath('//server/share')
>>> p = Path.from_uri('file://///server/share')
WindowsPath('//server/share')
>>> p = Path.from_uri('file:c:/windows')
WindowsPath('c:/windows')
>>> p = Path.from_uri('file:/c|/windows')
WindowsPath('c:/windows')
```

Se genera **`ValueError`** si la URI no comienza con `file:` o la ruta analizada no es absoluta.

Added in version 3.13.

Distinto en la versión 3.14: The URL authority is discarded if it matches the local hostname. Otherwise, if the authority isn't empty or `localhost`, then on Windows a UNC path is returned (as before), and on other platforms a **`ValueError`** is raised.

Path.as_uri()

Representa la ruta como una URL de «archivo». Se genera **`ValueError`** si la ruta no es absoluta.

```
>>> p = PosixPath('/etc/passwd')
>>> p.as_uri()
'file:///etc/passwd'
>>> p = WindowsPath('c:/Windows')
>>> p.as_uri()
'file:///c:/Windows'
```

Deprecated since version 3.14, will be removed in version 3.19: Calling this method from **`PurePath`** rather than **`Path`** is possible but deprecated. The method's use of **`os.fsencode()`** makes it strictly impure.

Ampliando y resolviendo caminos

classmethod Path.home()

Retorna un nuevo objeto ruta que representa el directorio de inicio del usuario (como lo retorna **`os.path.expanduser()`** con el agregado `~`). Si el directorio de inicio no se puede resolver, se lanza **`RuntimeError`**.

```
>>> Path.home()
PosixPath('/home/antoine')
```

Added in version 3.5.

Path.expanduser()

Retorna una nueva ruta con las construcciones `~` y `~user` expandidas, como lo retorna **`os.path.expanduser()`**. Si el directorio de inicio no se puede resolver, se lanza **`RuntimeError`**.

```
>>> p = PosixPath('~/films/Monty Python')
>>> p.expanduser()
PosixPath('/home/eric/films/Monty Python')
```

Added in version 3.5.

classmethod Path.cwd()

Retorna un nuevo objeto ruta que representa el directorio actual (como lo retorna **`os.getcwd()`**):

```
>>> Path.cwd()
PosixPath('/home/antoine/pathlib')
```

Path.absolute()

Hace que la ruta sea absoluta, sin normalización ni resolución de enlaces simbólicos. Retorna un nuevo objeto de ruta:

```
>>> p = Path('tests')
>>> p
PosixPath('tests')
>>> p.absolute()
PosixPath('/home/antoine/pathlib/tests')
```

Path.resolve(*strict=False*)

Hace que la ruta sea absoluta, resolviendo los enlaces simbólicos. Se retorna un nuevo objeto ruta:

```
>>> p = Path()
>>> p
PosixPath('.')
>>> p.resolve()
PosixPath('/home/antoine/pathlib')
```

Los componentes «`..`» también se eliminan (este es el único método para hacerlo):

```
>>> p = Path('docs/../setup.py')
>>> p.resolve()
PosixPath('/home/antoine/pathlib/setup.py')
```

Si no existe una ruta o se encuentra un bucle de enlace simbólico y *strict* es `True`, se genera **`OSError`**. Si *strict* es `False`, la ruta se resuelve en la medida de lo posible y cualquier resto se agrega sin verificar si existe.

Distinto en la versión 3.6: Se agregó el parámetro *strict* (el comportamiento previo a 3.6 es *strict*).

Distinto en la versión 3.13: Los bucles de enlaces simbólicos se tratan como otros errores: **`OSError`** se genera en modo estricto y no se genera ninguna excepción en modo no estricto. En versiones anteriores, **`RuntimeError`** se genera sin importar el valor de *strict*.

Path.readlink()

Retorna la ruta a la que apunta el vínculo simbólico (como lo retorna **`os.readlink()`**):

```
>>> p = Path('mylink')
>>> p.symlink_to('setup.py')
>>> p.readlink()
PosixPath('setup.py')
```

Added in version 3.9.

Distinto en la versión 3.13: Genera **`UnsupportedOperation`** si **`os.readlink()`** no está disponible. En versiones anteriores, se generaba **`NotImplementedError`**.

Consulta de tipo de archivo y estado

Distinto en la versión 3.8: **`exists()`**, **`is_dir()`**, **`is_file()`**, **`is_mount()`**, **`is_symlink()`**, **`is_block_device()`**, **`is_char_device()`**, **`is_fifo()`**, **`is_socket()`** ahora devuelven `False` en lugar de generar una excepción para las rutas que contienen caracteres no representables en el nivel del sistema operativo.

Distinto en la versión 3.14: The methods given above now return `False` instead of raising any **`OSError`** exception from the operating system. In previous versions, some kinds of **`OSError`** exception are raised, and others suppressed. The new behaviour is consistent with **`os.path.exists()`**, **`os.path.isdir()`**, etc. Use **`stat()`** to retrieve the file status without suppressing exceptions.

Path.stat(*, *follow_symlinks=True*)

Devuelve un objeto **`os.stat_result`** que contiene información sobre esta ruta, como **`os.stat()`**. El resultado se consulta en cada llamada a este método.

Este método normalmente sigue enlaces simbólicos; para evitar enlaces simbólicos, agregue el argumento `follow_symlinks = False`, o use **`lstat()`**.

```
>>> p = Path('setup.py')
>>> p.stat().st_size
956
>>> p.stat().st_mtime
1327883547.852554
```

Distinto en la versión 3.10: Se agregó el parámetro *follow_symlinks*.

Path.lstat()

Del mismo modo que **`Path.stat()`** pero si la ruta apunta a un enlace simbólico, retorna la información del enlace simbólico en lugar de la de su objetivo.

Path.exists(*, *follow_symlinks=True*)

Return `True` if the path points to an existing file or directory. `False` will be returned if the path is invalid, inaccessible or missing. Use **`Path.stat()`** to distinguish between these cases.

Este método normalmente sigue enlaces simbólicos; para comprobar si existe un enlace simbólico, agregue el argumento `follow_symlinks=False`.

```
>>> Path('.').exists()
True
>>> Path('setup.py').exists()
True
>>> Path('/etc').exists()
True
```

```
>>> Path('nonexistentfile').exists()
False
```

Distinto en la versión 3.12: Se agregó el parámetro *follow_symlinks*.

Path.is_file(***, *follow_symlinks=True*)

Return `True` if the path points to a regular file. `False` will be returned if the path is invalid, inaccessible or missing, or if it points to something other than a regular file. Use **`Path.stat()`** to distinguish between these cases.

Este método normalmente sigue los enlaces simbólicos; para excluirlos, agregue el argumento `follow_symlinks=False`.

Distinto en la versión 3.13: Se agregó el parámetro *follow_symlinks*.

Path.is_dir(***, *follow_symlinks=True*)

Return `True` if the path points to a directory. `False` will be returned if the path is invalid, inaccessible or missing, or if it points to something other than a directory. Use **`Path.stat()`** to distinguish between these cases.

Este método normalmente sigue los enlaces simbólicos; para excluir los enlaces simbólicos a directorios, agregue el argumento `follow_symlinks=False`.

Distinto en la versión 3.13: Se agregó el parámetro *follow_symlinks*.

Path.is_symlink()

Return `True` if the path points to a symbolic link, even if that symlink is broken. `False` will be returned if the path is invalid, inaccessible or missing, or if it points to something other than a symbolic link. Use **`Path.stat()`** to distinguish between these cases.

Path.is_junction()

Retorna `True` si la ruta apunta a una unión y `False` para cualquier otro tipo de archivo. Actualmente, solo Windows admite uniones.

Added in version 3.12.

Path.is_mount()

Retorna `True` si la ruta es un *mount point*: un punto en un sistema de archivos donde otro sistema de archivo se encuentra montado. En POSIX, la función comprueba si el padre de *path* , `path/..`, se encuentra en un dispositivo diferente que *path*, o si `path/..` y *path* apuntan al mismo i-nodo en el mismo dispositivo — esto debería detectar puntos de montajes para todas las variantes Unix y POSIX. En Windows, se considera que un punto de montaje es una letra de unidad de raíz (por ejemplo, `c:\`), un recurso compartido UNC (por ejemplo, `\\server\share`) o un directorio de sistema de archivos montado.

Added in version 3.7.

Distinto en la versión 3.12: Se agregó soporte para Windows.

Path.is_socket()

Return `True` if the path points to a Unix socket. `False` will be returned if the path is invalid, inaccessible or missing, or if it points to something other than a Unix socket. Use **`Path.stat()`** to distinguish between these cases.

Path.is_fifo()

Return `True` if the path points to a FIFO. `False` will be returned if the path is invalid, inaccessible or missing, or if it points to something other than a FIFO. Use **`Path.stat()`** to distinguish between these cases.

Path.is_block_device()

Return `True` if the path points to a block device. `False` will be returned if the path is invalid, inaccessible or missing, or if it points to something other than a block device. Use **`Path.stat()`** to distinguish between these cases.

Path.is_char_device()

Return `True` if the path points to a character device. `False` will be returned if the path is invalid, inaccessible or missing, or if it points to something other than a character device. Use **`Path.stat()`** to distinguish between these cases.

Path.samefile(*other_path*)

Retorna si la ruta apunta al mismo archivo que *other_path*, que puede ser un objeto *Path* o una cadena. La semántica es similar a **`os.path.samefile()`** y **`os.path.samestat()`**.

Se puede generar **`OSError`** si no se accede a alguno de los archivos por algún motivo.

```
>>> p = Path('spam')
>>> q = Path('eggs')
>>> p.samefile(q)
False
>>> p.samefile('spam')
True
```

Added in version 3.5.

Path.info

A **`PathInfo`** object that supports querying file type information. The object exposes methods that cache their results, which can help reduce the number of system calls needed when switching on file type. For example:

```
>>> p = Path('src')
>>> if p.info.is_symlink():
...     print('symlink')
... elif p.info.is_dir():
...     print('directory')
... elif p.info.exists():
...     print('something else')
... else:
...     print('not found')
...
directory
```

If the path was generated from `Path.iterdir()` then this attribute is initialized with some information about the file type gleaned from scanning the parent directory. Merely accessing **`Path.info`** does not perform any filesystem queries.

To fetch up-to-date information, it's best to call `Path.is_dir()`, `is_file()` and `is_symlink()` rather than methods of this attribute. There is no way to reset the cache; instead you can create a new path object with an empty info cache via `p = Path(p)`.

Added in version 3.14.

Lectura y escritura de archivos

Path.open(*mode='r', buffering=-1, encoding=None, errors=None, newline=None*)

Abre el archivo señalado por la ruta, como lo hace la función incorporada `open()`:

```
>>> p = Path('setup.py')
>>> with p.open() as f:
...     f.readline()
...
'#!/usr/bin/env python3\n'
```

Path.read_text(*encoding=None, errors=None, newline=None*)

Retorna el contenido decodificado del archivo apuntado como una cadena:

```
>>> p = Path('my_text_file')
>>> p.write_text('Text file contents')
18
>>> p.read_text()
'Text file contents'
```

El archivo se abre y luego se cierra. Los parámetros opcionales funcionan de la misma manera que en `open()`.

Added in version 3.5.

Distinto en la versión 3.13: Se agregó el parámetro *newline*.

Path.read_bytes()

Retorna el contenido binario del archivo apuntado como un objeto bytes:

```
>>> p = Path('my_binary_file')
>>> p.write_bytes(b'Binary file contents')
20
>>> p.read_bytes()
b'Binary file contents'
```

Added in version 3.5.

Path.write_text(*data*, *encoding=None*, *errors=None*, *newline=None*)

Abre el archivo apuntado en modo texto, escribe *data* y cierra el archivo:

```
>>> p = Path('my_text_file')
>>> p.write_text('Text file contents')
18
>>> p.read_text()
'Text file contents'
```

Se sobrescribe un archivo existente con el mismo nombre. Los parámetros opcionales tienen el mismo significado que en `open()`.

Added in version 3.5.

Distinto en la versión 3.10: Se agregó el parámetro *newline*.

Path.write_bytes(*data*)

Abre el archivo apuntado en modo bytes, escribe *data* y cierra el archivo:

```
>>> p = Path('my_binary_file')
>>> p.write_bytes(b'Binary file contents')
20
>>> p.read_bytes()
b'Binary file contents'
```

Se sobrescribe un archivo existente con el mismo nombre.

Added in version 3.5.

Directorios de lectura

Path.iterdir()

Cuando la ruta apunta a un directorio, produce objetos de ruta del contenido del directorio:

```
>>> p = Path('docs')
>>> for child in p.iterdir(): child
...
PosixPath('docs/conf.py')
PosixPath('docs/_templates')
PosixPath('docs/make.bat')
PosixPath('docs/index.rst')
PosixPath('docs/_build')
PosixPath('docs/_static')
PosixPath('docs/Makefile')
```

Los elementos secundarios se obtienen en un orden arbitrario y no se incluyen las entradas especiales `'.'` y `'..'`. Si se elimina o se agrega un archivo al directorio después de crear el iterador, no se especifica si se incluye un objeto de ruta para ese archivo.

Si la ruta no es un directorio o es inaccesible por algún motivo, se genera `OSError`.

Path.glob(*pattern*, ***, *case_sensitive=None*, *recurse_symlinks=False*)

Analiza el patrón comodín relativo a esta ruta para producir todos los archivos coincidentes (de cualquier tipo):

```
>>> sorted(Path('.').glob('*.py'))
[PosixPath('pathlib.py'), PosixPath('setup.py'), Po
>>> sorted(Path('.').glob('*/*.py'))
[PosixPath('docs/conf.py')]
>>> sorted(Path('.').glob('**/*.py'))
[PosixPath('build/lib/pathlib.py'),
 PosixPath('docs/conf.py'),
 PosixPath('pathlib.py'),
 PosixPath('setup.py'),
 PosixPath('test_pathlib.py')]
```

Nota

The paths are returned in no particular order. If you need a specific order, sort the results.

Ver también

Documentación Lenguaje de patrones.

De forma predeterminada, o cuando el argumento de sólo palabra clave *case_sensitive* está configurado en `None`, este método coincide con las rutas utilizando reglas de mayúsculas y minúsculas específicas de la plataforma: normalmente, distingue entre mayúsculas y minúsculas en POSIX y no distingue entre mayúsculas y minúsculas en Windows. Se configura *case_sensitive* en `True` o `False` para anular este comportamiento.

De forma predeterminada, o cuando el argumento de palabra clave *recurse_symlinks* se establece en `False`, este método sigue los enlaces simbólicos excepto cuando se expanden los comodines «`**`». Establezca *recurse_symlinks* en `True` para seguir siempre los enlaces simbólicos.

Nota

Any **`OSError`** exceptions raised from scanning the filesystem are suppressed. This includes **`PermissionError`** when accessing directories without read permission.

Levanta un auditing event `pathlib.Path.glob` con argumentos `self`, `pattern`.

Distinto en la versión 3.12: Se agregó el parámetro *case_sensitive*.

Distinto en la versión 3.13: Se agregó el parámetro *recurse_symlinks*.

Distinto en la versión 3.13: El parámetro *pattern* acepta un path-like object.

Distinto en la versión 3.13: Se suprimen todas las excepciones **`OSError`** generadas al escanear el sistema de archivos. En versiones anteriores, dichas excepciones se suprimían en muchos casos, pero no en todos.

Path.rglob(*pattern*, ***, *case_sensitive=None*, *recurse_symlinks=False*)

Incorpore de forma recursiva el *pattern* relativo dado. Es como llamar a **`Path.glob()`** con «`**/`» agregado delante de *pattern*.

Nota

The paths are returned in no particular order. If you need a specific order, sort the results.

Nota

Any **`OSError`** exceptions raised from scanning the filesystem are suppressed. This includes **`PermissionError`** when accessing directories without read permission.

Ver también

Documentación Lenguaje de patrones y **`Path.glob()`**.

Levanta un auditing event `pathlib.Path.rglob` con argumentos `self`, `pattern`.

Distinto en la versión 3.12: Se agregó el parámetro *case_sensitive*.

Distinto en la versión 3.13: Se agregó el parámetro *recurse_symlinks*.

Distinto en la versión 3.13: El parámetro *pattern* acepta un path-like object.

Path.walk(*top_down=True*, *on_error=None*, *follow_symlinks=False*)

Genera los nombres de archivos en un árbol de directorios recorriendo el árbol de arriba hacia abajo o de abajo hacia arriba.

Para cada directorio en el árbol de directorios con raíz en *self* (incluyendo *self* pero excluyendo "." y ".."), el método produce una tupla de 3 de elementos (o tripleta) `(dirpath, dirnames, filenames)`.

dirpath es un objeto **`Path`** al directorio que se está recorriendo actualmente, *dirnames* es una lista de cadenas de caracteres para los nombres de los subdirectorios en *dirpath* (excluyendo `'.'` y `'..'`), y *filenames* es una lista de cadenas de caracteres para los nombres de los archivos que no son directorios en *dirpath*. Para obtener una ruta completa (que comienza con *self*) a un archivo o directorio en *dirpath*, se ejecuta `dirpath / name`. El orden de las listas depende del sistema de archivos.

Si el argumento opcional *top_down* es verdadero (que es el valor predeterminado), la tripleta de un directorio se genera antes que las tripletas de cualquiera de sus subdirectorios (los directorios se recorren de arriba hacia abajo). Si *top_down* es falso, la tripleta de un directorio se genera después de las tripletas de todos sus subdirectorios (los directorios se recorren de abajo hacia arriba). Sin importar el valor de *top_down*, la lista de subdirectorios se recupera antes de que se recorran las tripletas del directorio y sus subdirectorios.

Cuando *top_down* es verdadero, el llamador puede modificar la lista *dirnames* en el lugar (por ejemplo, utilizando `del` o la asignación de sectores), y `Path.walk()` solo recurrirá a los subdirectorios cuyos nombres permanezcan en *dirnames*. Esto se puede utilizar para podar la búsqueda, o para imponer un orden específico de visita, o incluso para informar a `Path.walk()` sobre los directorios que el llamador crea o renombra antes de reanudar `Path.walk()` nuevamente. Modificar *dirnames* cuando *top_down* es falso no tiene ningún efecto sobre el comportamiento de `Path.walk()` ya que los directorios en *dirnames* ya se han generado en el momento en que *dirnames* se cede al llamador.

De forma predeterminada, los errores de `os.scandir()` se ignoran. Si se especifica el argumento opcional *on_error*, debe ser un objeto invocable; se llamará con un argumento, una instancia de `OSError`. El objeto invocable puede manejar el error para continuar la ejecución o volver a generarlo para detenerla. Tenga en cuenta que el nombre del archivo está disponible como el atributo `filename` del objeto de excepción.

De forma predeterminada, `Path.walk()` no sigue los enlaces simbólicos, sino que los agrega a la lista *filenames*. Establezca *follow_symlinks* como verdadero para resolver los enlaces simbólicos y colocarlos en *dirnames* y *filenames* según corresponda para sus destinos y, en consecuencia, visitar los directorios a los que apuntan los enlaces simbólicos (donde sea compatible).

Nota

Tener en cuenta que establecer *follow_symlinks* como verdadero puede generar una recursión infinita si un enlace apunta a un directorio principal de sí mismo. `Path.walk()` no realiza un seguimiento de los directorios que ya ha visitado.

Nota

`Path.walk()` asume que los directorios que recorre no se modifican durante la ejecución. Por ejemplo, si un directorio de *dirnames* ha sido reemplazado por un enlace simbólico y *follow_symlinks* es falso, `Path.walk()` intentará descender a él. Para evitar este comportamiento, elimine los directorios de *dirnames* según corresponda.

Nota

A diferencia de `os.walk()`, `Path.walk()` enumera enlaces simbólicos a directorios en *filenames* si *follow_symlinks* es falso.

Este ejemplo muestra la cantidad de bytes utilizados por todos los archivos en cada directorio, mientras ignora los directorios `__pycache__`:

```
from pathlib import Path
for root, dirs, files in Path("cpython/Lib/concurre
  print(
      root,
      "consumes",
      sum((root / file).stat().st_size for file in
      "bytes in",
      len(files),
      "non-directory files"
  )
  if '__pycache__' in dirs:
        dirs.remove('__pycache__')
```

El siguiente ejemplo es una implementación simple de `shutil.rmtree()`. Recorrer el árbol de abajo a arriba es esencial ya que `rmdir()` no permite eliminar un directorio antes de que esté vacío:

```
# Delete everything reachable from the directory "t
# CAUTION:  This is dangerous! For example, if top
# it could delete all of your files.
for root, dirs, files in top.walk(top_down=False):
    for name in files:
        (root / name).unlink()
    for name in dirs:
        (root / name).rmdir()
```

Added in version 3.12.

Creando archivos y directorios

Path.touch(*mode=0o666*, *exist_ok=True*)

Crea un archivo en esta ruta dada. Si se proporciona *mode*, se combina con el valor `umask` del proceso para determinar el modo de archivo y los indicadores de acceso. Si el archivo ya existe, la función se ejecuta correctamente cuando *exist_ok* es verdadero (y su hora de modificación se actualiza a la hora actual); de lo contrario, se genera `FileExistsError`.

Ver también

Los métodos `open()`, `write_text()` y `write_bytes()` se utilizan a menudo para crear archivos.

Path.mkdir(*mode=0o777*, *parents=False*, *exist_ok=False*)

Crea un nuevo directorio en esta ruta indicada. Si se proporciona *mode*, se combina con el valor `umask` del proceso para determinar el modo de archivo y los indicadores de acceso. Si la ruta ya existe, se genera `FileExistsError`.

Si *parents* es verdadero, los padres que faltan de esta ruta se crean según sea necesario; se crean con los permisos predeterminados sin tener en cuenta *mode* (imitando el comando POSIX `mkdir -p`).

Si *parents* es falso (el valor predeterminado), se genera un padre que falta **`FileNotFoundError`**.

Si *exist_ok* es falso (el valor predeterminado), se genera **`FileExistsError`** si el directorio de destino ya existe.

Si *exist_ok* es verdadero, no se generará **`FileExistsError`** a menos que la ruta dada ya exista en el sistema de archivos y no sea un directorio (el mismo comportamiento que el comando POSIX `mkdir -p`).

Distinto en la versión 3.5: Se agregó el parámetro *exist_ok*.

Path.symlink_to(*target*, *target_is_directory=False*)

Convierte esta ruta en un enlace simbólico que apunte a *target*.

En Windows, un enlace simbólico representa un archivo o un directorio y no se transforma en el destino de forma dinámica. Si el destino está presente, se creará el tipo de enlace simbólico para que coincida. De lo contrario, el enlace simbólico se creará como un directorio si *target_is_directory* es verdadero o como un enlace simbólico de archivo (el valor predeterminado) en caso contrario. En plataformas que no sean Windows, se ignora *target_is_directory*.

```
>>> p = Path('mylink')
>>> p.symlink_to('setup.py')
>>> p.resolve()
PosixPath('/home/antoine/pathlib/setup.py')
>>> p.stat().st_size
956
>>> p.lstat().st_size
8
```

Nota

El orden de los argumentos (*link*, *target*) es el reverso de `os.symlink()`"s.

Distinto en la versión 3.13: Genera `UnsupportedOperation` si `os.symlink()` no está disponible. En versiones anteriores, se generaba `NotImplementedError`.

Path.hardlink_to(*target*)

Hace de esta ruta un enlace fijo que apunta al mismo archivo que *target*.

Nota

El orden de los argumentos (link, target) es el reverso de `os.link()`.

Added in version 3.10.

Distinto en la versión 3.13: Genera `UnsupportedOperation` si `os.link()` no está disponible. En versiones anteriores, se generaba `NotImplementedError`.

Copying, moving and deleting

Path.copy(*target*, ***, *follow_symlinks=True*, *preserve_metadata=False*)

Copy this file or directory tree to the given *target*, and return a new **`Path`** instance pointing to *target*.

If the source is a file, the target will be replaced if it is an existing file. If the source is a symlink and *follow_symlinks* is true (the default), the symlink's target is copied. Otherwise, the symlink is recreated at the destination.

If *preserve_metadata* is false (the default), only directory structures and file data are guaranteed to be copied. Set *preserve_metadata* to true to ensure that file and directory permissions, flags, last access and modification times, and extended attributes are copied where supported. This argument has no effect when copying files on Windows (where metadata is always preserved).

Nota

Where supported by the operating system and file system, this method performs a lightweight copy, where data blocks are only copied when modified. This is known as copy-on-write.

Added in version 3.14.

Path.copy_into(*target_dir*, **, *follow_symlinks=True*, *preserve_metadata=False*)

Copy this file or directory tree into the given *target_dir*, which should be an existing directory. Other arguments are handled identically to **`Path.copy()`**. Returns a new **`Path`** instance pointing to the copy.

Added in version 3.14.

Path.rename(*target*)

Cambie el nombre de este archivo o directorio al *target* indicado y devuelva una nueva instancia de **`Path`** que apunte a *target*. En Unix, si *target* existe y es un archivo, se reemplazará de forma silenciosa si el usuario tiene permiso. En Windows, si *target* existe, se generará **`FileExistsError`**. *target* puede ser una cadena u otro objeto de ruta:

```
>>> p = Path('foo')
>>> p.open('w').write('some text')
9
>>> target = Path('bar')
>>> p.rename(target)
PosixPath('bar')
>>> target.open().read()
'some text'
```

La ruta de destino puede ser absoluta o relativa. Las rutas relativas se interpretan en relación con el directorio de trabajo actual, *not* el directorio del objeto `Path`.

Se implementa en términos de `os.rename()` y ofrece las mismas garantías.

Distinto en la versión 3.8: Se agregó valor de retorno, devuelve la nueva instancia `Path`.

Path.replace(*target*)

Cambie el nombre de este archivo o directorio por el *target* indicado y devuelva una nueva instancia de `Path` que apunte a *target*. Si *target* apunta a un archivo existente o a un directorio vacío, se reemplazará sin condiciones.

La ruta de destino puede ser absoluta o relativa. Las rutas relativas se interpretan en relación con el directorio de trabajo actual, *not* el directorio del objeto `Path`.

Distinto en la versión 3.8: Se agregó valor de retorno, devuelve la nueva instancia `Path`.

Path.move(*target*)

Move this file or directory tree to the given *target*, and return a new `Path` instance pointing to *target*.

If the *target* doesn't exist it will be created. If both this path and the *target* are existing files, then the target is overwritten. If both paths point to the same file or directory, or the *target* is a non-empty directory, then `OSError` is raised.

If both paths are on the same filesystem, the move is performed with `os.replace()`. Otherwise, this path is copied (preserving metadata and symlinks) and then deleted.

Added in version 3.14.

Path.move_into(*target_dir*)

Move this file or directory tree into the given *target_dir*, which should be an existing directory. Returns a new **`Path`** instance pointing to the moved path.

Added in version 3.14.

Path.unlink(*missing_ok=False*)

Elimine el archivo o enlace simbólico. Si la ruta apunta a un directorio, use **`Path.rmdir()`** en su lugar.

Si *missing_ok* es falso (el valor predeterminado), se genera **`FileNotFoundError`** si la ruta no existe.

Si *missing_ok* es verdadero, las excepciones **`FileNotFoundError`** serán ignoradas (el mismo comportamiento que el comando POSIX `rm -f`).

Distinto en la versión 3.8: Se agregó el parámetro *missing_ok*.

Path.rmdir()

Elimina el directorio. El directorio debe estar vacío.

Permisos y propiedad

Path.owner(*, *follow_symlinks=True*)

Devuelve el nombre del usuario propietario del archivo. Se genera el error **`KeyError`** si el identificador de usuario (UID) del archivo no se encuentra en la base de datos del sistema.

Este método normalmente sigue enlaces simbólicos; para obtener el propietario del enlace simbólico, agregue el argumento `follow_symlinks=False`.

Distinto en la versión 3.13: Genera **`UnsupportedOperation`** si el módulo **`pwd`** no está disponible. En versiones anteriores, se generaba **`NotImplementedError`**.

Distinto en la versión 3.13: Se agregó el parámetro *follow_symlinks*.

Path.group(**, follow_symlinks=True*)

Devuelve el nombre del grupo propietario del archivo. Se genera **KeyError** si no se encuentra el identificador de grupo (GID) del archivo en la base de datos del sistema.

Este método normalmente sigue enlaces simbólicos; para obtener el grupo del enlace simbólico, agregue el argumento `follow_symlinks=False`.

Distinto en la versión 3.13: Genera **UnsupportedOperation** si el módulo **grp** no está disponible. En versiones anteriores, se generaba **NotImplementedError**.

Distinto en la versión 3.13: Se agregó el parámetro *follow_symlinks*.

Path.chmod(*mode, *, follow_symlinks=True*)

Cambia el modo y los permisos de archivo, como **os.chmod()**.

Este método normalmente sigue enlaces simbólicos. Algunas versiones de Unix admiten el cambio de permisos en el enlace simbólico en sí; en estas plataformas puede agregar el argumento `follow_symlinks=False`, o usar **lchmod()**.

```
>>> p = Path('setup.py')
>>> p.stat().st_mode
33277
>>> p.chmod(0o444)
>>> p.stat().st_mode
33060
```

Distinto en la versión 3.10: Se agregó el parámetro *follow_symlinks*.

Path.lchmod(*mode*)

Del mismo modo que **Path.chmod()** pero si la ruta apunta a un enlace simbólico, el modo del enlace simbólico cambia en lugar del de su objetivo.

Lenguaje de patrones

Los siguientes comodines son compatibles con los patrones para **`full_match()`**, **`glob()`** y **`rglob()`**:

`**` (segmento completo)
: Coincide con cualquier número de segmentos de archivo o directorio, incluido cero.

`*` (segmento completo)
: Coincide con un segmento de archivo o directorio.

`*` (parte de un segmento)
: Coincide con cualquier número de caracteres no separadores, incluido el cero.

`?`
: Coincide con un carácter no separador.

`[seq]`
: Matches one character in *seq*, where *seq* is a sequence of characters. Range expressions are supported; for example, `[a-z]` matches any lowercase ASCII letter. Multiple ranges can be combined: `[a-zA-Z0-9_]` matches any ASCII letter, digit, or underscore.

`[!seq]`
: Matches one character not in *seq*, where *seq* follows the same rules as above.

Para una coincidencia literal, encierre los metacaracteres entre corchetes. Por ejemplo, `"[?]"` coincide con el carácter `"?"`.

El comodín «`**`» permite la codificación recursiva. Algunos ejemplos:

Patrón	Significado

«`**/*`»	Cualquier camino con al menos un segmento.
«`**/*.py`»	Cualquier ruta con un segmento final que termine «`.py`».
«`assets/**`»	Cualquier ruta que comience con «`assets/`».
«`assets/**/*`»	Cualquier ruta que comience con «`assets/`», excluyendo «`assets/`» en sí.

Nota

Al utilizar el comodín «`**`», se visitan todos los directorios del árbol. Las búsquedas en árboles de directorios grandes pueden llevar mucho tiempo.

Distinto en la versión 3.13: La codificación con un patrón que termina en «`**`» devuelve tanto archivos como directorios. En versiones anteriores, solo se devolvían directorios.

En `Path.glob()` y `rglob()`, se puede agregar una barra diagonal final al patrón para que coincida solo con directorios.

Distinto en la versión 3.11: El uso de un patrón que termina con un separador de componentes de ruta de acceso (`sep` o `altsep`) solo devuelve directorios.

Comparación con el módulo `glob`

Los patrones aceptados y los resultados generados por `Path.glob()` y `Path.rglob()` difieren ligeramente de los del módulo `glob`:

1. Los archivos que comienzan con un punto no son especiales en pathlib. Es como pasar `include_hidden=True` a `glob.glob()`.
2. Los componentes del patrón «`**`» siempre son recursivos en pathlib. Esto es como pasar `recursive=True` a `glob.glob()`.
3. Los componentes del patrón «`**`» no siguen los enlaces simbólicos de forma predeterminada en pathlib. Este comportamiento no tiene equivalente en `glob.glob()`, pero puede pasar `recurse_symlinks=True` a

`Path.glob()` para lograr un comportamiento compatible.
4. Al igual que todos los objetos `PurePath` y `Path`, los valores devueltos por `Path.glob()` y `Path.rglob()` no incluyen barras diagonales finales.
5. Los valores devueltos de `path.glob()` y `path.rglob()` de pathlib incluyen *path* como prefijo, a diferencia de los resultados de `glob.glob(root_dir=path)`.
6. Los valores devueltos de `path.glob()` y `path.rglob()` de pathlib pueden incluir el propio *path*, por ejemplo, al incluir «`**`», mientras que los resultados de `glob.glob(root_dir=path)` nunca incluyen una cadena vacía que corresponda a *path*.

Comparación con los módulos `os` y `os.path`

pathlib implementa operaciones de ruta utilizando objetos `PurePath` y `Path`, por lo que se dice que es *object-oriented*. Por otro lado, los módulos `os` y `os.path` proporcionan funciones que funcionan con objetos `str` y `bytes` de bajo nivel, lo que es un enfoque más propio de *procedural*. Algunos usuarios consideran que el estilo orientado a objetos es más legible.

Muchas funciones de `os` y `os.path` admiten rutas `bytes` y paths relative to directory descriptors. Estas funciones no están disponibles en pathlib.

Los tipos `str` y `bytes` de Python, y partes de los módulos `os` y `os.path`, están escritos en C y son muy rápidos. pathlib está escrito en Python puro y a menudo es más lento, pero rara vez lo suficientemente lento como para ser importante.

la normalización de rutas de pathlib es ligeramente más estricta y consistente que la de `os.path`. Por ejemplo, mientras que `os.path.abspath()` elimina los segmentos «`..`» de una ruta, que pueden cambiar su significado si hay enlaces simbólicos involucrados, `Path.absolute()` conserva estos segmentos para mayor seguridad.

la normalización de rutas de pathlib puede hacer que no sea adecuado para algunas aplicaciones:

1. pathlib normaliza `Path("my_folder/")` a `Path("my_folder")`, lo que cambia el significado de una ruta cuando se proporciona a varias API del sistema operativo y utilidades de línea de comandos. En concreto, la ausencia de un separador final puede permitir que la ruta se resuelva como un archivo o directorio, en lugar de solo como un directorio.
2. pathlib normaliza `Path("./my_program")` a `Path("my_program")`, lo que cambia el significado de una ruta cuando se utiliza como ruta de búsqueda ejecutable, como en un shell o al generar un proceso secundario. En concreto, la ausencia de un separador en la ruta puede obligar a que se busque en **`PATH`** en lugar de en el directorio actual.

Como consecuencia de estas diferencias, pathlib no es un reemplazo directo para `os.path`.

Herramientas correspondientes

A continuación se muestra una tabla que asigna varias funciones `os` a sus equivalentes en `PurePath`/`Path`.

`os.path`	**pathlib**
`os.path.dirname()`	`PurePath.parent`
`os.path.basename()`	`PurePath.name`
`os.path.splitext()`	`PurePath.stem`, `PurePath.suffix`
`os.path.join()`	`PurePath.joinpath()`
`os.path.isabs()`	`PurePath.is_absolute()`
`os.path.relpath()`	`PurePath.relative_to()` [1]
`os.path.expanduser()`	`Path.expanduser()` [2]
`os.path.realpath()`	`Path.resolve()`
`os.path.abspath()`	`Path.absolute()` [3]
`os.path.exists()`	`Path.exists()`
`os.path.isfile()`	`Path.is_file()`
`os.path.isdir()`	`Path.is_dir()`
`os.path.islink()`	`Path.is_symlink()`
`os.path.isjunction()`	`Path.is_junction()`

`os.path.ismount()`	`Path.is_mount()`
`os.path.samefile()`	`Path.samefile()`
`os.getcwd()`	`Path.cwd()`
`os.stat()`	`Path.stat()`
`os.lstat()`	`Path.lstat()`
`os.listdir()`	`Path.iterdir()`
`os.walk()`	`Path.walk()` [4]
`os.mkdir()`, `os.makedirs()`	`Path.mkdir()`
`os.link()`	`Path.hardlink_to()`
`os.symlink()`	`Path.symlink_to()`
`os.readlink()`	`Path.readlink()`
`os.rename()`	`Path.rename()`
`os.replace()`	`Path.replace()`
`os.remove()`, `os.unlink()`	`Path.unlink()`
`os.rmdir()`	`Path.rmdir()`
`os.chmod()`	`Path.chmod()`
`os.lchmod()`	`Path.lchmod()`

Notas al pie

`os.path.relpath()` llama a `abspath()` para hacer que las rutas sean absolutas y eliminar partes «`..`», mientras que `PurePath.relative_to()` es una operación léxica que genera `ValueError` cuando los anclajes de sus entradas difieren (por ejemplo, si una ruta es absoluta y la otra relativa).

`os.path.expanduser()` devuelve la ruta sin cambios si no se puede resolver el directorio de inicio, mientras que `Path.expanduser()` genera `RuntimeError`.

`os.path.abspath()` elimina los componentes «`..`» sin resolver los enlaces simbólicos, lo que puede cambiar el significado de la ruta, mientras que `Path.absolute()` deja cualquier componente «`..`» en la ruta.

`os.walk()` siempre sigue los enlaces simbólicos al categorizar rutas en *dirnames* y *filenames*, mientras que `Path.walk()` categoriza todos los enlaces simbólicos en *filenames* cuando *follow_symlinks* es falso (el valor predeterminado).

Protocols

The **`pathlib.types`** module provides types for static type checking.

Added in version 3.14.

class pathlib.types.PathInfo

A **`typing.Protocol`** describing the **`Path.info`** attribute. Implementations may return cached results from their methods.

exists(**, follow_symlinks=True*)

Return `True` if the path is an existing file or directory, or any other kind of file; return `False` if the path doesn't exist.

If *follow_symlinks* is `False`, return `True` for symlinks without checking if their targets exist.

is_dir(**, follow_symlinks=True*)

Return `True` if the path is a directory, or a symbolic link pointing to a directory; return `False` if the path is (or points to) any other kind of file, or if it doesn't exist.

If *follow_symlinks* is `False`, return `True` only if the path is a directory (without following symlinks); return `False` if the path is any other kind of file, or if it doesn't exist.

is_file(**, follow_symlinks=True*)

Return `True` if the path is a file, or a symbolic link pointing to a file; return `False` if the path is (or points to) a directory or other non-file, or if it doesn't exist.

If *follow_symlinks* is `False`, return `True` only if the path is a file (without following symlinks); return `False` if the path is a directory or other non-file, or if it doesn't exist.

is_symlink()

Return `True` if the path is a symbolic link (even if broken); return `False` if the path is a directory or any kind of file, or if it doesn't exist.

`os.path` — Common pathname manipulations

Source code: Lib/genericpath.py [https://github.com/python/cpython/tree/3.14/Lib/genericpath.py], Lib/posixpath.py [https://github.com/python/cpython/tree/3.14/Lib/posixpath.py] (for POSIX) and Lib/ntpath.py [https://github.com/python/cpython/tree/3.14/Lib/ntpath.py] (for Windows).

Este módulo implementa algunas funciones útiles sobre los nombres de rutas. Para leer o escribir archivos consulte `open()`, y para acceder a archivos del sistema vea el `os`. Los parámetros para las rutas pueden ser pasados como caracteres, o bytes ,o cualquier objeto implementado en el protocolo `os.PathLike`.

A diferencia de un shell Unix, Python no realiza ninguna expansión de ruta *automática*. Las funciones como `expanduser()` y `expandvars()` puede ser invocadas de manera explicita cuando una aplicación desea realizar una expansión de ruta *shell-like*. (Consulte el módulo `glob`.)

Ver también

El módulo `pathlib` ofrece objetos de ruta de alto nivel.

Nota

Todas estas funciones aceptan solo bytes o solo objetos de cadena como sus parámetros. El resultado es un objeto del mismo tipo, si se retorna una ruta o un nombre de archivo.

Nota

Since different operating systems have different path name conventions, there are several versions of this module in the standard library. The **`os.path`** module is always the path module suitable for the operating system Python is running on, and therefore usable for local paths. However, you can also import and use the individual modules if you want to manipulate a path that is *always* in one of the different formats. They all have the same interface:

- **`posixpath`** for UNIX-style paths
- **`ntpath`** for Windows paths

Distinto en la versión 3.8: **`exists()`**, **`lexists()`**, **`isdir()`**, **`isfile()`**, **`islink()`**, y **`ismount()`** ahora retornan `False` en lugar de lanzar una excepción para rutas que contienen caracteres o bytes que no se puedan representar a nivel de sistema operativo.

os.path.abspath(*path*)

Return a normalized absolutized version of the pathname *path*. On most platforms, this is equivalent to calling `normpath(join(os.getcwd(), path))`.

Ver también

`os.path.join()` and **`os.path.normpath()`**.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.basename(*path*, */*)

Retorna un nombre base de nombre de ruta *path*. Este es el segundo elemento del par retornado al pasar *path* a la función **`split()`**. Toma en cuenta que el resultado de esta función es diferente a la del programa de Unix **basename**; donde **basename** para `'/foo/bar/'` retorna `'bar'`, la función **`basename()`** retorna una cadena vacía (`''`).

Distinto en la versión 3.6: Acepta un path-like object.

os.path.commonpath(*paths*)

Return the longest common sub-path of each pathname in the iterable *paths*. Raise `ValueError` if *paths* contain both absolute and relative pathnames, if *paths* are on different drives, or if *paths* is empty. Unlike `commonprefix()`, this returns a valid path.

Added in version 3.5.

Distinto en la versión 3.6: Acepta una secuencia de path-like objects.

Distinto en la versión 3.13: Any iterable can now be passed, rather than just sequences.

os.path.commonprefix(*list*, */*)

Return the longest string prefix (taken character-by-character) that is a prefix of all strings in *list*. If *list* is empty, return the empty string (`''`).

Advertencia

This function may return invalid paths because it works a character at a time. If you need a **common path prefix**, then the algorithm implemented in this function is not secure. Use `commonpath()` for finding a common path prefix.

```
>>> os.path.commonprefix(['/usr/lib', '/usr/local/
'/usr/l'

>>> os.path.commonpath(['/usr/lib', '/usr/local/li
'/usr'
```

Distinto en la versión 3.6: Acepta un path-like object.

os.path.dirname(*path*, */*)

Retorna el nombre del directorio de la ruta *path*. Es el primer elemento del par retornado al pasar *path* a la función `split()`.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.exists(*path*)

Retorna `True` si *path* se refiere a una ruta existente o un descriptor de archivo abierto. Retorna `False` para enlaces simbólicos rotos. En algunas plataformas, esta función puede retornar `False` si no se concede permiso para ejecutar **`os.stat()`** en el archivo solicitado, incluso la ruta *path* existe físicamente.

Distinto en la versión 3.3: path ahora puede ser un valor entero: retorna `True` si es un descriptor de archivo abierto, de otro modo retorna `False`.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.lexists(*path*)

Return `True` if *path* refers to an existing path, including broken symbolic links. Equivalent to **`exists()`** on platforms lacking **`os.lstat()`**.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.expanduser(*path*)

En Unix y Windows, retorna el argumento con un componente inicial de `~` o `~user` reemplazado por el directorio *home* de *user*.

En Unix, el `~` inicial es reemplazado por la variable de entorno **`HOME`** si está activada; si no, el directorio principal del usuario actual se busca en el directorio de contraseñas a través del módulo incorporado **`pwd`**. Un `~user` inicial es buscado directamente en el directorio de contraseñas.

En Windows, se usará **`USERPROFILE`** si se establece, de lo contrario se usará una combinación de **`HOMEPATH`** y **`HOMEDRIVE`**. Un `~user` inicial se maneja comprobando que el último componente de directorio del directorio de inicio del usuario actual coincide con **`USERNAME`**, y reemplazándolo si es así.

Si la expansión falla o si la ruta no comienza con una tilde, la ruta se retorna sin cambios.

Distinto en la versión 3.6: Acepta un path-like object.

Distinto en la versión 3.8: Ya no utiliza **`HOME`** en Windows.

os.path.expandvars(*path*)

Retorna el argumento con variables de entorno expandidas. Las sub-cadenas de la forma `$name` or `${name}` se reemplazan por el valor de la variable de entorno *name*. Los nombres de variables mal formadas y las referencias a variables no existentes se dejan sin cambios.

En Windows, las expansiones `%name%` están soportadas además de `$name` y `${name}`.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.getatime(*path*, /)

Return the time of last access of *path*. The return value is a floating-point number giving the number of seconds since the epoch (see the **`time`** module). Raise **`OSError`** if the file does not exist or is inaccessible.

os.path.getmtime(*path*, /)

Return the time of last modification of *path*. The return value is a floating-point number giving the number of seconds since the epoch (see the **`time`** module). Raise **`OSError`** if the file does not exist or is inaccessible.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.getctime(*path*, /)

Retorna el *ctime* del sistema que, en algunos sistemas (como Unix) es la hora del último cambio de metadatos y, en otros (como Windows), es el tiempo de creación de *path*. El valor retornado es un número que da el número de segundos desde la época (consulta el módulo `time`). Lanza una excepción `OSError` si el archivo no existe o es inaccesible.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.getsize(*path*, */*)

Retorna el tamaño en bytes de *path*, Lanza una excepción `OSError` si el archivo no existe o es inaccesible.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.isabs(*path*, */*)

Return `True` if *path* is an absolute pathname. On Unix, that means it begins with a slash, on Windows that it begins with two (back)slashes, or a drive letter, colon, and (back)slash together.

Ver también

`abspath()`

Distinto en la versión 3.6: Acepta un path-like object.

Distinto en la versión 3.13: On Windows, returns `False` if the given path starts with exactly one (back)slash.

os.path.isfile(*path*)

Retorna `True` si *path* es un archivo `existing`. Esto sigue los enlaces simbólicos, por lo que tanto `islink()` como `isfile()` pueden ser verdaderos para la misma ruta.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.isdir(*path*, */*)

Retorna `True` si *path* es un directorio `existing`. Esto sigue los enlaces simbólicos, por lo que tanto `islink()` como `isdir()` pueden ser verdaderos para la misma ruta.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.isjunction(*path*)

Return `True` if *path* refers to an **`existing`** directory entry that is a junction. Always return `False` if junctions are not supported on the current platform.

Added in version 3.12.

os.path.islink(*path*)

Retorna `True` si *path* hace referencia a una entrada de directorio **`existing`** que es un enlace simbólico. Siempre `False` si el entorno de ejecución de Python no admite vínculos simbólicos..

Distinto en la versión 3.6: Acepta un path-like object.

os.path.ismount(*path*)

Return `True` if pathname *path* is a *mount point*: a point in a file system where a different file system has been mounted. On POSIX, the function checks whether *path*'s parent, `path/..`, is on a different device than *path*, or whether `path/..` and *path* point to the same i-node on the same device — this should detect mount points for all Unix and POSIX variants. It is not able to reliably detect bind mounts on the same filesystem. On Linux systems, it will always return `True` for btrfs subvolumes, even if they aren't mount points. On Windows, a drive letter root and a share UNC are always mount points, and for any other path `GetVolumePathName` is called to see if it is different from the input path.

Distinto en la versión 3.4: Added support for detecting non-root mount points on Windows.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.isdevdrive(*path*)

Return `True` if pathname *path* is located on a Windows Dev Drive. A Dev Drive is optimized for developer scenarios, and offers faster performance for reading and writing files. It is recommended for use for source code, temporary build directories, package caches, and other IO-intensive operations.

May raise an error for an invalid path, for example, one without a recognizable drive, but returns `False` on platforms that do not support Dev Drives. See the Windows documentation [https://learn.microsoft.com/windows/dev-drive/] for information on enabling and creating Dev Drives.

Added in version 3.12.

Distinto en la versión 3.13: The function is now available on all platforms, and will always return `False` on those that have no support for Dev Drives

os.path.isreserved(*path*)

Return `True` if *path* is a reserved pathname on the current system.

On Windows, reserved filenames include those that end with a space or dot; those that contain colons (i.e. file streams such as «name:stream»), wildcard characters (i.e. `'*?"<>'`), pipe, or ASCII control characters; as well as DOS device names such as «NUL», «CON», «CONIN$», «CONOUT$», «AUX», «PRN», «COM1», and «LPT1».

Nota

This function approximates rules for reserved paths on most Windows systems. These rules change over time in various Windows releases. This function may be updated in future Python releases as changes to the rules become broadly available.

Availability: Windows.

Added in version 3.13.

os.path.join(*path*, /, **paths*)

Join one or more path segments intelligently. The return value is the concatenation of *path* and all members of **paths*, with exactly one directory separator following each non-empty part, except the last. That is, the result will only end in a separator if the last part is either empty or ends in a separator.

If a segment is an absolute path (which on Windows requires both a drive and a root), then all previous segments are ignored and joining continues from the absolute path segment. On Linux, for example:

```
>>> os.path.join('/home/foo', 'bar')
'/home/foo/bar'
>>> os.path.join('/home/foo', '/home/bar')
'/home/bar'
```

On Windows, the drive is not reset when a rooted path segment (e.g., `r'\foo'`) is encountered. If a segment is on a different drive or is an absolute path, all previous segments are ignored and the drive is reset. For example:

```
>>> os.path.join('c:\\', 'foo')
'c:\\foo'
>>> os.path.join('c:\\foo', 'd:\\bar')
'd:\\bar'
```

Note that since there is a current directory for each drive, `os.path.join("c:", "foo")` represents a path relative to the current directory on drive `C:` (`c:foo`), not `c:\foo`.

Distinto en la versión 3.6: Acepta un objeto path-like object para *path* y *paths*.

os.path.normcase(*path*, /)

Normaliza las mayúsculas y minúsculas de un nombre de

ruta. En Windows convierte todos los caracteres en el nombre de ruta a minúsculas y también convierte las barras inclinadas hacia atrás en barras inclinadas hacia atrás. En otros sistemas operativos, retorna la ruta sin cambios.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.normpath(*path*)

Normaliza un nombre de ruta colapsando separadores redundantes y referencias de nivel superior para que `A//B`, `A/B/`, `A/./B` y `A/foo/../B` se transformen en `A/B`. Esta modificación de cadena puede que modifique el significado de la ruta que contenga enlaces simbólicos. En Windows, convierte las barras inclinadas hacia adelante en barras hacia atrás. Para normalizar mayúsculas y minúsculas, utiliza **`normcase()`**.

Nota

En sistemas POSIX, de acuerdo con IEEE Std 1003.1 Edición 2013; 4.13 Resolución de Nombres de Rutas [https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_13], si el nombre de ruta comienza con exactamente dos barras diagonales, el primer componente que sigue a los caracteres principales puede interpretarse de una manera definida por la implementación, aunque más de dos caracteres principales se tratarán como un solo carácter.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.realpath(*path*, /, *, *strict=False*)

Return the canonical path of the specified filename, eliminating any symbolic links encountered in the path (if they are supported by the operating system). On Windows, this function will also resolve MS-DOS (also called 8.3) style names such as `C:\\PROGRA~1` to `C:\\Program Files`.

By default, the path is evaluated up to the first component that does not exist, is a symlink loop, or whose evaluation raises `OSError`. All such components are appended unchanged to the existing part of the path.

Some errors that are handled this way include «access denied», «not a directory», or «bad argument to internal function». Thus, the resulting path may be missing or inaccessible, may still contain links or loops, and may traverse non-directories.

This behavior can be modified by keyword arguments:

If *strict* is `True`, the first error encountered when evaluating the path is re-raised. In particular, `FileNotFoundError` is raised if *path* does not exist, or another `OSError` if it is otherwise inaccessible.

If *strict* is `os.path.ALLOW_MISSING`, errors other than `FileNotFoundError` are re-raised (as with `strict=True`). Thus, the returned path will not contain any symbolic links, but the named file and some of its parent directories may be missing.

Nota

Esta función emula el procedimiento del sistema operativo para hacer que una ruta sea canónica, que difiere ligeramente entre Windows y UNIX con respecto a cómo interactúan los enlaces y los componentes de la ruta posterior.

Las API del sistema operativo hacen que las rutas sean canónicas según sea necesario, por lo que normalmente no es necesario llamar a esta función.

Distinto en la versión 3.6: Acepta un path-like object.

Distinto en la versión 3.8: Los enlaces y uniones simbólicos ahora se resuelven en Windows.

Distinto en la versión 3.10: Se agregó el parámetro *strict*.

Distinto en la versión 3.14: The `ALLOW_MISSING` value for the *strict* parameter was added.

os.path.ALLOW_MISSING

Special value used for the *strict* argument in `realpath()`.

Added in version 3.14.

os.path.relpath(*path*, *start=os.curdir*)

Retorna un nombre de ruta relativo a *path* desde el directorio actual o de un directorio *start* opcional. Este es un cálculo de ruta: No se accede al sistema de archivos para confirmar la existencia o la naturaleza de *path* o *start*. En Windows, se lanza `ValueError` cuando *path* y *start* están en discos diferentes.

start defaults to `os.curdir`.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.samefile(*path1*, *path2*, */*)

Retorna `True` si ambos argumentos de nombre de ruta refieren al mismo archivo o directorio. Esto se determina por el número de dispositivo y el número de *i-node* y lanza una excepción si una llamada de `os.stat()` en alguno de los nombres de ruta falla.

Distinto en la versión 3.2: Añadido soporte para Windows.

Distinto en la versión 3.4: Windows ahora utiliza la misma implementación que en el resto de las plataformas.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.sameopenfile(*fp1*, *fp2*)

Retorna `True` si los descriptores de archivo *fp1* y *fp2* se refieren al mismo archivo.

Distinto en la versión 3.2: Añadido soporte para Windows.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.samestat(*stat1*, *stat2*, */*)

Retorna `True` si las tuplas de *stat* (*stat1* y *stat2*) refieren al mismo archivo. Estas estructuras pueden haber sido retornadas por **`os.fstat()`**, **`os.lstat()`**, o **`os.stat()`**. Esta función implementa la comparación subyacente utilizada por: **`samefile()`** y **`sameopenfile()`**.

Distinto en la versión 3.4: Añadido soporte para Windows.

os.path.split(*path*, */*)

Split the pathname *path* into a pair, `(head, tail)` where *tail* is the last pathname component and *head* is everything leading up to that. The *tail* part will never contain a slash; if *path* ends in a slash, *tail* will be empty. If there is no slash in *path*, *head* will be empty. If *path* is empty, both *head* and *tail* are empty. Trailing slashes are stripped from *head* unless it is the root (one or more slashes only). In all cases, `join(head, tail)` returns a path to the same location as *path* (but the strings may differ). Also see the functions **`join()`**, **`dirname()`** and **`basename()`**.

Distinto en la versión 3.6: Acepta un path-like object.

os.path.splitdrive(*path*, */*)

Divide el nombre de ruta *path* en un par `(drive, tail)` donde *drive* es un punto de montaje o una cadena vacía. En sistemas que no utilizan especificaciones de unidad, *drive* siempre será una cadena vacía. En todos los casos, `drive + tail` será lo mismo que *path*.

En Windows, divide un nombre de ruta en unidad / punto compartido UNC y ruta relativa.

Si la ruta contiene una letra de unidad, la unidad contendrá todo hasta los dos puntos inclusive:

```
>>> splitdrive("c:/dir")
```

```
("c:", "/dir")
```

If the path contains a UNC path, drive will contain the host name and share:

```
>>> splitdrive("//host/computer/dir")
("//host/computer", "/dir")
```

Distinto en la versión 3.6: Acepta un path-like object.

os.path.splitroot(*path*, */*)

Split the pathname *path* into a 3-item tuple `(drive, root, tail)` where *drive* is a device name or mount point, *root* is a string of separators after the drive, and *tail* is everything after the root. Any of these items may be the empty string. In all cases, `drive + root + tail` will be the same as *path*.

On POSIX systems, *drive* is always empty. The *root* may be empty (if *path* is relative), a single forward slash (if *path* is absolute), or two forward slashes (implementation-defined per IEEE Std 1003.1-2017; 4.13 Pathname Resolution [https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_13].) For example:

```
>>> splitroot('/home/sam')
('', '/', 'home/sam')
>>> splitroot('//home/sam')
('', '//', 'home/sam')
>>> splitroot('///home/sam')
('', '/', '//home/sam')
```

On Windows, *drive* may be empty, a drive-letter name, a UNC share, or a device name. The *root* may be empty, a forward slash, or a backward slash. For example:

```
>>> splitroot('C:/Users/Sam')
('C:', '/', 'Users/Sam')
>>> splitroot('//Server/Share/Users/Sam')
('//Server/Share', '/', 'Users/Sam')
```

Added in version 3.12.

os.path.splitext(*path*, /)

Divide el nombre de la ruta *path* en un par `(root, ext)` de tal forma que `root + ext == path`, y *ext* queda vacío o inicia con un punto y contiene a lo mucho un punto.

Si la ruta no contiene ninguna extensión, *ext* será `''`:

```
>>> splitext('bar')
('bar', '')
```

Si la ruta contiene una extensión, *ext* se establecerá como esta extensión, incluido el punto por delante. Tenga en cuenta que los puntos anteriores se ignorarán:

```
>>> splitext('foo.bar.exe')
('foo.bar', '.exe')
>>> splitext('/foo/bar.exe')
('/foo/bar', '.exe')
```

Los puntos que están por delante del último componente de la ruta son considerados parte de la raíz:

```
>>> splitext('.cshrc')
('.cshrc', '')
>>> splitext('/foo/....jpg')
('/foo/....jpg', '')
```

Distinto en la versión 3.6: Acepta un path-like object.

os.path.supports_unicode_filenames

`True` si se pueden utilizar cadenas Unicode arbitrarias como nombres de archivo (dentro de las limitaciones impuestas por el sistema de archivos).

`stat` — Interpreting `stat()` results

Código fuente: Lib/stat.py [https://github.com/python/cpython/tree/3.14/Lib/stat.py]

The `stat` module defines constants and functions for interpreting the results of `os.stat()`, `os.fstat()` and `os.lstat()` (if they exist). For complete details about the `stat()`, `fstat()` and `lstat()` calls, consult the documentation for your system.

Distinto en la versión 3.4: El módulo *stat* se apoya en una implementación en C.

The `stat` module defines the following functions to test for specific file types:

stat.S_ISDIR(*mode*)

Retorna un valor no nulo si el modo es de un directorio.

stat.S_ISCHR(*mode*)

Retorna un valor no nulo si el modo es de un archivo de un dispositivo especial de caracteres.

stat.S_ISBLK(*mode*)

Retorna un valor no nulo si el modo es de un archivo de un dispositivo especial de bloques.

stat.S_ISREG(*mode*)

Retorna un valor no nulo si el modo es de un archivo normal.

stat.S_ISFIFO(*mode*)

Retorna un valor no nulo si el modo es de un *FIFO* (tubería con nombre).

stat.S_ISLNK(*mode*)

Retorna un valor no nulo si el modo es de un enlace simbólico.

stat.S_ISSOCK(*mode*)

Retorna un valor no nulo si el modo es de un socket.

stat.S_ISDOOR(*mode*)

Retorna un valor no nulo si el modo es de un *door*.

Added in version 3.4.

stat.S_ISPORT(*mode*)

Retorna un valor no nulo si el modo es de un *event port*.

Added in version 3.4.

stat.S_ISWHT(*mode*)

Retorna un valor no nulo si el modo es de un *whiteout*.

Added in version 3.4.

Se definen dos funciones adicionales para una manipulación más general del modo del archivo:

stat.S_IMODE(*mode*)

Retorna la porción del modo del archivo que puede ser establecida por `os.chmod()`— esto es, los bits de los permisos del archivo más los bits *sticky bit*, *set-group-id* y *set-user-id* (en los sistemas que lo soporten).

stat.S_IFMT(*mode*)

Retorna la porción del modo del archivo que describe el tipo de archivo (usado por las funciones `S_IS*()` de más arriba).

Normalmente se usarían las funciones `os.path.is*()` para comprobar el tipo de un archivo; estas funciones de aquí son útiles cuando se hacen múltiples comprobaciones sobre el mismo archivo y se desea evitar la sobrecarga causada por la llamada al sistema `stat()` en cada comprobación. También son útiles cuando se comprueba información de un archivo que no es gestionada por `os.path`, como buscar dispositivos de bloques o caracteres.

Ejemplo:

```
import os, sys
from stat import *

def walktree(top, callback):
    '''recursively descend the directory tree rooted at
       calling the callback function for each regular fi

    for f in os.listdir(top):
        pathname = os.path.join(top, f)
        mode = os.lstat(pathname).st_mode
        if S_ISDIR(mode):
            # It's a directory, recurse into it
            walktree(pathname, callback)
        elif S_ISREG(mode):
            # It's a file, call the callback function
            callback(pathname)
        else:
            # Unknown file type, print a message
            print('Skipping %s' % pathname)

def visitfile(file):
    print('visiting', file)

if __name__ == '__main__':
    walktree(sys.argv[1], visitfile)
```

Se proporciona una función de utilidad adicional para convertir el modo del archivo en una cadena de caracteres legible por humanos:

stat.filemode(*mode*)

Convierte el modo del archivo a una cadena de caracteres de la forma "-rwxrwxrwx".

Added in version 3.3.

Distinto en la versión 3.4: La función soporta **`S_IFDOOR`**, **`S_IFPORT`** y **`S_IFWHT`**.

Todas las variables de debajo son simplemente índices simbólicos sobre la tupla de 10 elementos retornada por **`os.stat()`**, **`os.fstat()`** o **`os.lstat()`**.

stat.ST_MODE

Modo de protección del *inode*.

stat.ST_INO

Número del *inode*.

stat.ST_DEV

Dispositivo en el que reside el *inode*.

stat.ST_NLINK

Número de enlaces al *inode*.

stat.ST_UID

Id de usuario del propietario.

stat.ST_GID

Id del grupo del propietario.

stat.ST_SIZE

Tamaño en bytes de un archivo normal; cantidad de datos esperando en algunos archivos especiales.

stat.ST_ATIME

Momento del último acceso.

stat.ST_MTIME

Momento de la última modificación.

stat.ST_CTIME

El «ctime» reportado por el sistema operativo. En algunos sistemas (como Unix) es el momento del último cambio en los metadatos, y en otros (como Windows), es el momento de creación (véase la documentación de la plataforma para más detalles).

La interpretación de «tamaño de archivo» cambia dependiendo del tipo de archivo. Para archivos normales es el tamaño del archivo en bytes. En la mayoría de sistemas Unix (incluyendo Linux en particular), para *FIFOs* y sockets es el número de bytes que esperan ser leídos en el momento de la llamada a `os.stat()`, `os.fstat()`, o `os.lstat()`; en ocasiones, esto puede ser útil, especialmente para sondear uno de estos archivos especiales después de una apertura no bloqueante. Para otros dispositivos de caracteres y bloques el significado del campo *size* es más variado, dependiendo de la implementación de la llamada al sistema subyacente.

Las variables de debajo definen los flags usados en el campo `ST_MODE`.

El uso de las funciones de arriba es más portable que el uso del primer juego de flags:

stat.S_IFSOCK

Socket.

stat.S_IFLNK

Enlace simbólico.

stat.S_IFREG

Archivo normal.

stat.S_IFBLK

Dispositivo de bloques.

stat.S_IFDIR

Directorio.

stat.S_IFCHR

Dispositivo de caracteres.

stat.S_IFIFO

FIFO.

stat.S_IFDOOR

Door.

Added in version 3.4.

stat.S_IFPORT

Event port.

Added in version 3.4.

stat.S_IFWHT

Whiteout.

Added in version 3.4.

Nota

`S_IFDOOR`, `S_IFPORT` o `S_IFWHT` se definen como 0 cuando la plataforma no soporta los tipos de archivo.

Los siguientes flags también pueden usarse en el argumento *mode* de `os.chmod()`:

stat.S_ISUID

Establecer el bit *UID*.

stat.S_ISGID

Bit *Set-group-ID*. Este bit tiene varios usos especiales. Para un directorio indica que la semántica BSD debe usarse para ese directorio: los archivos creados ahí heredan el *ID* de grupo del directorio, no del *ID* de grupo efectivo del proceso que los crea, y los directorios creados ahí también tendrán activado el bit **`S_ISGID`**. Para un archivo que no tiene activado el bit de ejecución de grupo (**`S_IXGRP`**), el bit *Set-group-ID* indica el bloqueo obligatorio del archivo/registro (véase también **`S_ENFMT`**).

stat.S_ISVTX

Sticky bit. Cuando este bit está activado en un directorio, significa que un archivo dentro de ese directorio puede ser renombrado o borrado sólo por el propietario del archivo, por el propietario del directorio, o por un proceso con privilegios.

stat.S_IRWXU

Máscara para los permisos del propietario del archivo.

stat.S_IRUSR

El propietario tiene permiso de lectura.

stat.S_IWUSR

El propietario tiene permiso de escritura.

stat.S_IXUSR

El propietario tiene permiso de ejecución.

stat.S_IRWXG

Máscara para los permisos del grupo.

stat.S_IRGRP

El grupo tiene permiso de lectura.

stat.S_IWGRP

El grupo tiene permiso de escritura.

stat.S_IXGRP

El grupo tiene permiso de ejecución.

stat.S_IRWXO

Máscara para permisos de los otros (no en el grupo).

stat.S_IROTH

Los otros tienen permiso de lectura.

stat.S_IWOTH

Los otros tienen permiso de escritura.

stat.S_IXOTH

Los otros tienen permiso de ejecución.

stat.S_ENFMT

Imposición del bloqueo de archivos de System V. Este flag se comparte con `S_ISGID`: se impone el bloqueo de archivos/registros en archivos que no tengan activado el bit de ejecución por el grupo (`S_IXGRP`).

stat.S_IREAD

Sinónimo de `S_IRUSR` en Unix V7.

stat.S_IWRITE

Sinónimo de `S_IWUSR` en Unix V7.

stat.S_IEXEC

Sinónimo de `S_IXUSR` en Unix V7.

Los siguientes flags pueden usarse como argumento *flags* de `os.chflags()`:

stat.UF_SETTABLE

All user settable flags.

Added in version 3.13.

stat.UF_NODUMP

No volcar el archivo.

stat.UF_IMMUTABLE

El archivo no puede ser modificado.

stat.UF_APPEND

Sólo se puede añadir al archivo.

stat.UF_OPAQUE

El directorio es opaco cuando se mira a través de un *union stack*.

stat.UF_NOUNLINK

El archivo no puede ser renombrado o borrado.

stat.UF_COMPRESSED

El archivo se almacena comprimido (macOS 10.6+).

stat.UF_TRACKED

Used for handling document IDs (macOS)

Added in version 3.13.

stat.UF_DATAVAULT

The file needs an entitlement for reading or writing (macOS 10.13+)

Added in version 3.13.

stat.UF_HIDDEN

El archivo no debe ser mostrado en una GUI (macOS 10.5+).

stat.SF_SETTABLE

All super-user changeable flags

Added in version 3.13.

stat.SF_SUPPORTED

All super-user supported flags

Availability: macOS

Added in version 3.13.

stat.SF_SYNTHETIC

All super-user read-only synthetic flags

Availability: macOS

Added in version 3.13.

stat.SF_ARCHIVED

El archivo puede ser archivado.

stat.SF_IMMUTABLE

El archivo no puede ser modificado.

stat.SF_APPEND

Sólo se puede añadir al archivo.

stat.SF_RESTRICTED

The file needs an entitlement to write to (macOS 10.13+)

Added in version 3.13.

stat.SF_NOUNLINK

El archivo no puede ser renombrado o borrado.

stat.SF_SNAPSHOT

El archivo es una instantánea.

stat.SF_FIRMLINK

The file is a firmlink (macOS 10.15+)

Added in version 3.13.

stat.SF_DATALESS

The file is a dataless object (macOS 10.15+)

Added in version 3.13.

Consulte la página de manual de sistemas * BSD o macOS *chflags(2)* *[https://manpages.debian.org/chflags(2)]* para obtener más información.

En Windows, las siguientes constantes de atributos de fichero están disponibles para ser usadas al comprobar los bits del miembro `st_file_attributes` retornado por **`os.stat()`**. Véase Windows API documentation [https://msdn.microsoft.com/en-us/library/windows/desktop/gg258117.aspx] para más detalles sobre el significado de estas constantes.

stat.FILE_ATTRIBUTE_ARCHIVE
stat.FILE_ATTRIBUTE_COMPRESSED
stat.FILE_ATTRIBUTE_DEVICE
stat.FILE_ATTRIBUTE_DIRECTORY
stat.FILE_ATTRIBUTE_ENCRYPTED
stat.FILE_ATTRIBUTE_HIDDEN
stat.FILE_ATTRIBUTE_INTEGRITY_STREAM
stat.FILE_ATTRIBUTE_NORMAL
stat.FILE_ATTRIBUTE_NOT_CONTENT_INDEXED
stat.FILE_ATTRIBUTE_NO_SCRUB_DATA
stat.FILE_ATTRIBUTE_OFFLINE
stat.FILE_ATTRIBUTE_READONLY
stat.FILE_ATTRIBUTE_REPARSE_POINT
stat.FILE_ATTRIBUTE_SPARSE_FILE
stat.FILE_ATTRIBUTE_SYSTEM
stat.FILE_ATTRIBUTE_TEMPORARY
stat.FILE_ATTRIBUTE_VIRTUAL

Added in version 3.5.

En Windows, las siguientes constantes están disponibles para la comparación con el miembro `st_reparse_tag` retornado por **`os.lstat()`**. Estas constantes son muy conocidas, pero no se trata de una lista exhaustiva.

stat.IO_REPARSE_TAG_SYMLINK

stat.IO_REPARSE_TAG_MOUNT_POINT
stat.IO_REPARSE_TAG_APPEXECLINK

Added in version 3.8.

`filecmp` — File and Directory Comparisons

Código fuente: Lib/filecmp.py [https://github.com/python/cpython/tree/3.14/Lib/filecmp.py]

The **`filecmp`** module defines functions to compare files and directories, with various optional time/correctness trade-offs. For comparing files, see also the **`difflib`** module.

The **`filecmp`** module defines the following functions:

filecmp.cmp(*f1*, *f2*, *shallow=True*)

> Compara los ficheros llamados *f1* y *f2*, retornando `True` si son iguales, `False` en caso contrario.
>
> Si *shallow* es verdadero y las firmas **`os.stat()`** (tipo de fichero, tamaño, y tiempo de modificación) de los dos ficheros son idénticas, los ficheros se consideran iguales.
>
> De lo contrario, los ficheros son tratados como diferentes si sus tamaños o contenidos difieren.
>
> Note que ningún programa externo es llamado desde esta función, dándole portabilidad y eficiencia.
>
> La función utiliza un caché para comparaciones pasadas y los resultados, con entradas de caché invalidadas si la información de **`os.stat()`** para el fichero cambia. El caché entero puede ser limpiado utilizando **`clear_cache()`**.

filecmp.cmpfiles(*dir1*, *dir2*, *common*, *shallow=True*)

> Compara los ficheros en los dos directorios *dir1* y *dir2* cuyos nombres son dados por *common*.

Retorna tres listas de nombres de fichero: *match*, *mismatch*, *errors*. *match* contiene la lista de ficheros que coinciden, *mismatch* contiene los nombres de aquellos que no, y *errors* lista los nombres de los ficheros que no deberían ser comparados. Los ficheros son listados en *errors* si no existen en uno de los directorios, el usuario carece de permisos para leerlos o si la comparación no puede hacerse por alguna razón.

El parámetro *shallow* tiene el mismo significado y valor por defecto en cuanto a **`filecmp.cmp()`**.

Por ejemplo, `cmpfiles('a', 'b', ['c', 'd/e'])` comparará `a/c` con `b/c` y `a/d/e` con `b/d/e`. `'c'` y `'d/e'` estará cada uno en una de las tres listas retornadas.

filecmp.clear_cache()

Limpia el caché de filecmp. Esto podría ser útil si un fichero es comparado muy rápido después de que es modificado que está dentro de la resolución mtime del sistema de archivos subyacente.

Added in version 3.4.

La clase **`dircmp`**

class filecmp.dircmp(*a*, *b*, *ignore=None*, *hide=None*, ***, *shallow=True*)

Construye un nuevo objeto de comparación de directorio, para comparar los directorios *a* y *b*. *ignore* es una lista de nombres a ignorar, y predetermina a **`filecmp.DEFAULT_IGNORES`**. *hide* es una lista de nombres a ocultar, y predetermina a `[os.curdir, os.pardir]`.

The **`dircmp`** class compares files by doing *shallow* comparisons as described for **`filecmp.cmp()`** by default using the *shallow* parameter.

Distinto en la versión 3.13: Added the *shallow* parameter.

La clase `dircmp` provee los siguientes métodos:

report()

Imprime (a `sys.stdout`) una comparación entre *a* y *b*.

report_partial_closure()

Imprime una comparación entre *a* y *b* y subdirectorios inmediatos comunes.

report_full_closure()

Imprime una comparación entre *a* y *b* y subdirectorios comunes (recursivamente).

La clase `dircmp` ofrece un número de atributos interesantes que pueden ser utilizados para obtener varios bits de información sobre los árboles de directorio que están siendo comparados.

Note que vía los hooks `__getattr__()`, todos los atributos son perezosamente computados, así que no hay penalización de velocidad si sólo esos atributos que son ligeros de computar son utilizados.

left

El directorio *a*.

right

El directorio *b*.

left_list

Ficheros y subdirectorios en *a*, filtrados por *hide* e *ignore*.

right_list

Ficheros y subdirectorios en *b*, filtrados por *hide* e *ignore*.

common

Ficheros y subdirectorios en *a* y *b*.

left_only

Ficheros y subdirectorios sólo en *a*.

right_only

Ficheros y subdirectorios sólo en *b*.

common_dirs

Subdirectorios en *a* y *b*.

common_files

Ficheros en *a* y *b*.

common_funny

Nombres en *a* y *b*, de forma que el tipo difiera entre los directorios, o los nombres por los que `os.stat()` reporta un error.

same_files

Ficheros que son idénticos en *a* y *b*, utilizando el operador de comparación de fichero de la clase.

diff_files

Ficheros que están en *a* y *b*, cuyos contenidos difieren acorde al operador de comparación del fichero de clase.

funny_files

Ficheros que están en *a* y *b*, pero no deberían ser comparados.

subdirs

Un diccionario que asigna nombres en `common_dirs` a instancias `dircmp` (o instancias MyDirCmp si esa instancias es del tipo MyDirCmp, una subclase de

dircmp).

Distinto en la versión 3.10: Anteriormente las entradas siempre eran instancias **dircmp**. Ahora son del mismo tipo que *self*, si *self* es una subclase de **dircmp**.

filecmp.DEFAULT_IGNORES

Added in version 3.4.

Lista de directorios ignorados por **dircmp** por defecto.

Aquí hay un ejemplo simplificado de uso del atributo `subdirs` para buscar recursivamente a través de dos directorios para mostrar diferentes ficheros comunes:

```
>>> from filecmp import dircmp
>>> def print_diff_files(dcmp):
...     for name in dcmp.diff_files:
...         print("diff_file %s found in %s and %s" % (r
...               dcmp.right))
...     for sub_dcmp in dcmp.subdirs.values():
...         print_diff_files(sub_dcmp)
...
>>> dcmp = dircmp('dir1', 'dir2')
>>> print_diff_files(dcmp)
```

`tempfile` — Generate temporary files and directories

Código fuente: Lib/tempfile.py [https://github.com/python/cpython/tree/3.14/Lib/tempfile.py]

This module creates temporary files and directories. It works on all supported platforms. **`TemporaryFile`**, **`NamedTemporaryFile`**, **`TemporaryDirectory`**, and **`SpooledTemporaryFile`** are high-level interfaces which provide automatic cleanup and can be used as context managers. **`mkstemp()`** and **`mkdtemp()`** are lower-level functions which require manual cleanup.

Todas las funciones y constructores invocables por el usuario toman argumentos adicionales que permiten el control directo sobre la ubicación y el nombre de los archivos y directorios temporales. Los nombres de archivos utilizados por este módulo incluyen una cadena de caracteres aleatorios que permite que esos archivos se creen de forma segura en directorios temporales compartidos. Para mantener la compatibilidad con versiones anteriores, el orden de argumentos es algo extraño; se recomienda utilizar argumentos nombrados para mayor claridad.

El módulo define los siguientes elementos invocables por el usuario:

tempfile.TemporaryFile(*mode='w+b'*, *buffering=-1*, *encoding=None*, *newline=None*, *suffix=None*, *prefix=None*, *dir=None*, ***, *errors=None*)

> Retorna un file-like object que se puede usar como área de almacenamiento temporal. El archivo se crea de forma segura, usando las mismas reglas que **`mkstemp()`**. Este se destruirá tan pronto como se cierre (incluido un cierre implícito cuando el objeto es recolectado como basura). Bajo Unix, la entrada de directorio para el archivo no se crea en lo

absoluto o se elimina inmediatamente después de crear el archivo. Otras plataformas no soportan esto; tu código no debería depender en un archivo temporal creado con esta función teniendo o no un nombre visible en el sistema de archivos.

The resulting object can be used as a context manager (see Ejemplos). On completion of the context or destruction of the file object the temporary file will be removed from the filesystem.

El parámetro *mode* por defecto es `'w+b'` para que el archivo creado pueda leerse y escribirse sin cerrarse. El modo binario se utiliza para que se comporte consistentemente en todas las plataformas sin tener en cuenta los datos que se almacenan. *buffering*, *encoding*, *errors* y *newline* se interpretan como en **`open()`**.

Los parámetros *dir*, *prefix* y *suffix* tienen el mismo significado y valores predeterminados de **`mkstemp()`**.

El objeto retornado es un objeto de archivo verdadero en las plataformas POSIX. En otras plataformas, es un objeto similar a un archivo cuyo atributo **`file`** es el objeto de archivo verdadero subyacente.

El indicador **`os.O_TMPFILE`** se usa si está disponible y funciona (específico de Linux, requiere el kernel de Linux 3.11 o posterior).

En plataformas que no son ni Posix ni Cygwin, TemporaryFile es un alias de NamedTemporaryFile.

Lanza un evento de auditoria `tempfile.mkstemp` con argumento `fullpath`.

Distinto en la versión 3.5: El indicador **`os.O_TMPFILE`** ahora se usa si está disponible.

Distinto en la versión 3.8: Se agregó el parámetro *errors*.

tempfile.NamedTemporaryFile(*mode='w+b'*, *buffering=-1*, *encoding=None*, *newline=None*, *suffix=None*, *prefix=None*, *dir=None*, *delete=True*, ***, *errors=None*, *delete_on_close=True*)

Esta función opera exactamente como `TemporaryFile()`, excepto por las siguientes diferencias:

- Esta función retorna un archivo que tiene garantizado un nombre visible en el sistema de archivos.
- Para administrar el archivo nombrado, amplía los parámetros de `TemporaryFile()` con los parámetros *delete* y *delete_on_close* que determinan si el archivo nombrado debe eliminarse automáticamente y cómo.

The returned object is always a file-like object whose `file` attribute is the underlying true file object. This file-like object can be used in a `with` statement, just like a normal file. The name of the temporary file can be retrieved from the `name` attribute of the returned file-like object. On Unix, unlike with the `TemporaryFile()`, the directory entry does not get unlinked immediately after the file creation.

Si *delete* es true (el valor predeterminado) y *delete_on_close* es true (el valor predeterminado), el archivo se elimina tan pronto como se cierra. Si *delete* es true y *delete_on_close* es false, el archivo se elimina solo al salir del administrador de contexto, o bien cuando se finaliza el file-like object. La eliminación no siempre está garantizada en este caso (ver `object.__del__()`). Si *delete* es false, se omite el valor de *delete_on_close*.

Por lo tanto, para usar el nombre del archivo temporal para volver a abrir el archivo después de cerrarlo, asegúrese de no eliminar el archivo al cerrarlo (establezca el parámetro *delete* en false) o, en caso de que el archivo temporal se cree en una declaración `with`, establezca el parámetro *delete_on_close* en false. Se recomienda este último enfoque, ya que proporciona asistencia en la limpieza automática del archivo temporal al salir del administrador de contexto.

Abrir el archivo temporal de nuevo por su nombre mientras todavía está abierto funciona de la siguiente forma:

- En POSIX el archivo siempre se puede volver a abrir.
- En Windows, asegúrese de que se cumple al menos una de las siguientes condiciones:
  - *delete* es falso
  - la función de apertura adicional comparte el acceso de eliminación (por ejemplo, llamando **`os.open()`** con la opción `O_TEMPORARY`)
  - *delete* es verdadero, pero *delete_on_close* es falso. Tenga en cuenta que, en este caso, las funciones de apertura adicional que no comparten el acceso de eliminación (por ejemplo, creado mediante el tipo integrado **`open()`**) deben cerrarse antes de salir del gestor de contexto, sino la llamada **`os.unlink()`** al salir del gestor de contexto fallará con un **`PermissionError`**.

En Windows, si *delete_on_close* es falso y el archivo se crea en un directorio para el que el usuario carece de acceso de eliminación, la llamada **`os.unlink()`** al salir del gestor de contexto fallará con un **`PermissionError`**. Esto no puede suceder cuando *delete_on_close* es true, porque el acceso de eliminación es solicitado por la función de apertura, que falla inmediatamente si no se concede el acceso solicitado.

En POSIX (solo), un proceso que finaliza abruptamente con SIGKILL no puede eliminar automáticamente ningún NamedTemporaryFiles que se haya creado.

Lanza un evento de auditoria `tempfile.mkstemp` con argumento `fullpath`.

Distinto en la versión 3.8: Se agregó el parámetro *errors*.

Distinto en la versión 3.12: Se agregó el parámetro *delete_on_close*.

class tempfile.SpooledTemporaryFile(*max_size=0*, *mode='w+b'*, *buffering=-1*, *encoding=None*, *newline=None*, *suffix=None*,

prefix=None, *dir=None*, ***, *errors=None*)

This class operates exactly as `TemporaryFile()` does, except that data is spooled in memory until the file size exceeds *max_size*, or until the file's `fileno()` method is called, at which point the contents are written to disk and operation proceeds as with `TemporaryFile()`.

rollover()

The resulting file has one additional method, `rollover()`, which causes the file to roll over to an on-disk file regardless of its size.

The returned object is a file-like object whose `_file` attribute is either an `io.BytesIO` or `io.TextIOWrapper` object (depending on whether binary or text *mode* was specified) or a true file object, depending on whether `rollover()` has been called. This file-like object can be used in a `with` statement, just like a normal file.

Distinto en la versión 3.3: the truncate method now accepts a *size* argument.

Distinto en la versión 3.8: Se agregó el parámetro *errors*.

Distinto en la versión 3.11: Implementa completamente las clases base abstractas `io.BufferedIOBase` y `io.TextIOBase` (dependiendo de si se especificó *modo* binario o de texto).

class tempfile.TemporaryDirectory(*suffix=None*, *prefix=None*, *dir=None*, *ignore_cleanup_errors=False*, ***, *delete=True*)

This class securely creates a temporary directory using the same rules as `mkdtemp()`. The resulting object can be used as a context manager (see Ejemplos). On completion of the context or destruction of the temporary directory object, the newly created temporary directory and all its contents are removed from the filesystem.

name

The directory name can be retrieved from the **`name`** attribute of the returned object. When the returned object is used as a context manager, the **`name`** will be assigned to the target of the **`as`** clause in the **`with`** statement, if there is one.

cleanup()

The directory can be explicitly cleaned up by calling the **`cleanup()`** method. If *ignore_cleanup_errors* is true, any unhandled exceptions during explicit or implicit cleanup (such as a **`PermissionError`** removing open files on Windows) will be ignored, and the remaining removable items deleted on a «best-effort» basis. Otherwise, errors will be raised in whatever context cleanup occurs (the **`cleanup()`** call, exiting the context manager, when the object is garbage-collected or during interpreter shutdown).

El parámetro *delete* se puede utilizar para deshabilitar la limpieza del árbol de directorios al salir del contexto. Aunque puede parecer inusual que un administrador de contexto deshabilite la acción realizada al salir del contexto, puede ser útil durante la depuración o cuando necesita que el comportamiento de limpieza sea condicional en función de otra lógica.

Lanza un evento de auditoria `tempfile.mkdtemp` con argumento `fullpath`.

Added in version 3.2.

Distinto en la versión 3.10: Se agregó el parámetro *ignore_cleanup_errors*.

Distinto en la versión 3.12: Se agregó el parámetro *delete*.

tempfile.mkstemp(*suffix=None*, *prefix=None*, *dir=None*, *text=False*)

Creates a temporary file in the most secure manner possible.

There are no race conditions in the file's creation, assuming that the platform properly implements the **`os.O_EXCL`** flag for **`os.open()`**. The file is readable and writable only by the creating user ID. If the platform uses permission bits to indicate whether a file is executable, the file is executable by no one.

The file descriptor is not inherited by child processes.

A diferencia de **`TemporaryFile()`**, el usuario de **`mkstemp()`** es responsable de eliminar el archivo temporal cuando haya terminado con él.

Si *suffix* no es `None`, el nombre del archivo terminará con ese sufijo, de lo contrario no habrá sufijo. **`mkstemp()`** no pone un punto entre el nombre del archivo y el sufijo; si necesita uno, póngalo al comienzo del *suffix*.

Si *prefix* no es `None`, el nombre del archivo comenzará con ese prefijo; de lo contrario, se usa un prefijo predeterminado. El valor predeterminado es el valor de retorno de **`gettempprefix()`** o **`gettempprefixb()`**, según corresponda.

Si *dir* no es `None`, el archivo se creará en ese directorio; de lo contrario, se usa un directorio predeterminado. El directorio predeterminado se elige de una lista dependiente de la plataforma, pero el usuario de la aplicación puede controlar la ubicación del directorio configurando las variables de entorno *TMPDIR*, *TEMP* o *TMP* . Por lo tanto, no hay garantía de que el nombre de archivo generado tenga buenas propiedades, como no requerir comillas cuando se pasa a comandos externos a través de `os.popen()`.

Si alguno de los argumentos *suffix*, *prefix*, y *dir* no son `None`, estos deben ser del mismo tipo. Si son bytes, el nombre retornado será bytes en lugar de str. Si desea forzar un valor de retorno de bytes con un comportamiento predeterminado, pase `suffix=b''`.

Si se especifica *text* y es verdadero, el archivo se abre en modo texto. De lo contrario, (por defecto) el archivo se abre en modo binario.

`mkstemp()` retorna una tupla que contiene un controlador de nivel de sistema operativo a un archivo abierto (como sería retornado por **`os.open()`**) y la ruta absoluta de ese archivo, en ese orden.

Lanza un evento de auditoria `tempfile.mkstemp` con argumento `fullpath`.

Distinto en la versión 3.5: suffix, *prefix*, y *dir* ahora se pueden suministrar en bytes para obtener el valor de retorno en bytes. Antes de esto, solo str se permitía. *suffix* y *prefix* ahora aceptan y por defecto `None` para hacer que se use un valor predeterminado apropiado.

Distinto en la versión 3.6: El parámetro *dir* ahora acepta un path-like object.

tempfile.mkdtemp(*suffix=None*, *prefix=None*, *dir=None*)

Crea un directorio temporal de la manera más segura posible. No hay condiciones de carrera en la creación del directorio. El directorio se puede leer, escribir y buscar solo por el ID del usuario creador.

El usuario de **`mkdtemp()`** es responsable de eliminar el directorio temporal y su contenido cuando haya terminado con él.

Los argumentos *prefix*, *suffix*, y *dir* son los mismos que para **`mkstemp()`**.

`mkdtemp()` retorna la ruta absoluta del nuevo directorio.

Lanza un evento de auditoria `tempfile.mkdtemp` con argumento `fullpath`.

Distinto en la versión 3.5: suffix, *prefix*, y *dir* ahora se pueden suministrar en bytes para obtener el valor de retorno en bytes. Antes de esto, solo str se permitía. *suffix* y *prefix* ahora aceptan y por defecto `None` para hacer que se use un valor predeterminado apropiado.

Distinto en la versión 3.6: El parámetro *dir* ahora acepta un path-like object.

Distinto en la versión 3.12: **`mkdtemp()`** no siempre retorna la ruta absoluta del nuevo directorio, incluso si *dir* es relativo.

tempfile.gettempdir()

Retorna el nombre del directorio utilizado para archivos temporales. Esto define el valor predeterminado para el argumento *dir* para todas las funciones en este módulo.

Python busca en una lista estándar de directorios para encontrar uno dentro del cual el usuario pueda crear archivos. La lista es:

1. El directorio nombrado por la variable de entorno **`TMPDIR`**.
2. El directorio nombrado por la variable de entorno **`TEMP`**.
3. El directorio nombrado por la variable de entorno **`TMP`**.
4. Una ubicación especifica de la plataforma:
    - En Windows, los directorios `C:\TEMP`, `C:\TMP`, `\TEMP`, y `\TMP`, en ese orden.
    - En todas las otras plataformas, los directorios `/tmp`, `/var/tmp`, y `/usr/tmp`, en ese orden.
7. Y como última alternativa, el directorio de trabajo actual.

El resultado de la búsqueda es cacheada, vea la descripción de **`tempdir`** abajo.

Distinto en la versión 3.10: Siempre retorna str. Anteriormente retornaría cualquier valor **`tempdir`** independientemente del tipo siempre que no fuera `None`.

tempfile.gettempdirb()

Igual a **`gettempdir()`** pero el valor retornado es en bytes.

Added in version 3.5.

tempfile.gettempprefix()

Retorna el prefijo del nombre de archivo utilizado para crear archivos temporales. Este no contiene el componente de directorio.

tempfile.gettempprefixb()

Igual que **`gettempprefix()`** pero el valor retornado es en bytes.

Added in version 3.5.

El módulo utiliza una variable global para almacenar el nombre del directorio utilizado para los archivos temporales retornados por **`gettempdir()`**. Se puede configurar directamente para sobrescribir el proceso de selección, pero esto no se recomienda. Todas las funciones en este módulo toman un argumento *dir* que puede usarse para especificar el directorio. Este es el enfoque recomendado que no toma por sorpresa otro código inesperado al cambiar el comportamiento global de la API.

tempfile.tempdir

Cuando se establece en un valor distinto de `None`, esta variable define el valor predeterminado para el argumento *dir* para las funciones definidas en este módulo, incluyendo su tipo, bytes o str. No puede ser un path-like object.

Si `tempdir` es `None` (por defecto) en cualquier llamada a cualquiera de las funciones anteriores excepto **`gettempprefix()`** se inicializa siguiendo el algoritmo descrito en **`gettempdir()`**.

Nota

Tenga en cuenta que si establece `tempdir` en un valor de bytes, hay un efecto secundario desagradable: el tipo de retorno global predeterminado de **`mkstemp()`** y **`mkdtemp()`** cambia a bytes cuando no se proporcionan argumentos explícitos `prefix`, `suffix` o `dir` de tipo str. Por favor, no escriba código esperando o dependiendo de esto. Este comportamiento incómodo se mantiene por compatibilidad con la implementación histórica.

Ejemplos

Here are some examples of typical usage of the **`tempfile`** module:

```
>>> import tempfile

# create a temporary file and write some data to it
>>> fp = tempfile.TemporaryFile()
>>> fp.write(b'Hello world!')
# read data from file
>>> fp.seek(0)
>>> fp.read()
b'Hello world!'
# close the file, it will be removed
>>> fp.close()

# create a temporary file using a context manager
>>> with tempfile.TemporaryFile() as fp:
...     fp.write(b'Hello world!')
...     fp.seek(0)
...     fp.read()
b'Hello world!'
>>>
# file is now closed and removed

# create a temporary file using a context manager
# close the file, use the name to open the file again
>>> with tempfile.NamedTemporaryFile(delete_on_close=Fal
...     fp.write(b'Hello world!')
...     fp.close()
... # the file is closed, but not removed
... # open the file again by using its name
...     with open(fp.name, mode='rb') as f:
...         f.read()
b'Hello world!'
>>>
# file is now removed

# create a temporary directory using the context manager
>>> with tempfile.TemporaryDirectory() as tmpdirname:
...     print('created temporary directory', tmpdirname)
>>>
# directory and contents have been removed
```

Funciones y variables deprecadas

Una forma histórica de crear archivos temporales era generar primero un nombre de archivo con la función `mktemp()` y luego crear un archivo con este nombre. Desafortunadamente, esto no es seguro, porque un proceso diferente puede crear un archivo con este nombre en el tiempo entre la llamada a `mktemp()` y el intento posterior de crear el archivo mediante el primer proceso. La solución es combinar los dos pasos y crear el archivo de inmediato. Este enfoque es utilizado por `mkstemp()` y las otras funciones descritas anteriormente.

tempfile.mktemp(*suffix=''*, *prefix='tmp'*, *dir=None*)

Obsoleto desde la versión 2.3: Utilice `mkstemp()` en su lugar.

Retorna el nombre de la ruta absoluta de un archivo que no existía en el momento en que se realiza la llamada. Los argumentos *prefix*, *suffix* y *dir* son similares a los de `mkstemp()`, excepto los nombres de archivo de bytes, `suffix=None` y `prefix=None` no son soportados.

Advertencia

El uso de esta función puede introducir un agujero de seguridad en su programa. Para cuando llegues a hacer algo con el nombre de archivo que retorna, alguien más pudo haberse adelantado. El uso de `mktemp()` se puede reemplazar fácilmente con **`NamedTemporaryFile()`**, pasándole el parámetro `delete=False`:

```
>>> f = NamedTemporaryFile(delete=False)
>>> f.name
'/tmp/tmptjujjt'
>>> f.write(b"Hello World!\n")
13
>>> f.close()
>>> os.unlink(f.name)
>>> os.path.exists(f.name)
False
```

`glob` — Unix style pathname pattern expansion

Código fuente: Lib/glob.py [https://github.com/python/cpython/tree/3.14/Lib/glob.py]

The **`glob`** module finds pathnames using pattern matching rules similar to the Unix shell. No tilde expansion is done, but `*`, `?`, and character ranges expressed with `[]` will be correctly matched. This is done by using the **`os.scandir()`** and **`fnmatch.fnmatch()`** functions in concert, and not by actually invoking a subshell.

Nota

The pathnames are returned in no particular order. If you need a specific order, sort the results.

Files beginning with a dot (`.`) can only be matched by patterns that also start with a dot, unlike **`fnmatch.fnmatch()`** or **`pathlib.Path.glob()`**. For tilde and shell variable expansion, use **`os.path.expanduser()`** and **`os.path.expandvars()`**.

Para una coincidencia literal, envuelve los meta-caracteres en corchetes. Por ejemplo, `'[?]'` empareja el caracter `'?'`.

The **`glob`** module defines the following functions:

glob.glob(*pathname*, ***, *root_dir=None*, *dir_fd=None*, *recursive=False*, *include_hidden=False*)

Return a possibly empty list of path names that match *pathname*, which must be a string containing a path specification. *pathname* can be either absolute (like `/usr/src/Python-1.5/Makefile`) or relative (like `../../Tools/*/*.gif`), and can contain shell-style wildcards.

Broken symlinks are included in the results (as in the shell). Whether or not the results are sorted depends on the file system. If a file that satisfies conditions is removed or added during the call of this function, whether a path name for that file will be included is unspecified.

If *root_dir* is not `None`, it should be a path-like object specifying the root directory for searching. It has the same effect on **`glob()`** as changing the current directory before calling it. If *pathname* is relative, the result will contain paths relative to *root_dir*.

Esta función puede admitir rutas relativas a descriptores de directorio con el parámetro *dir_fd*.

Si *recursive* es verdadero, el patrón «`**`» coincidirá con cualquier fichero y cero o más directorios, subdirectorios y enlaces simbólicos a directorios. Si el patrón va seguido de un **`os.sep`** o **`os.altsep`** los ficheros no coincidirán.

Si *include_hidden* es verdadero, el patrón «`**`» coincidirá con los directorios ocultos.

Lanza un evento de auditoría `glob.glob` con los argumentos `pathname`, `recursive`.

Lanza un auditing event `glob.glob/2` con argumentos `pathname`, `recursive`, `root_dir`, `dir_fd`.

Nota

El uso del patrón «`**`» en árboles de directorios grandes podría consumir una cantidad de tiempo excesiva.

Nota

This function may return duplicate path names if *pathname* contains multiple «`**`» patterns and *recursive* is true.

Nota

Any **`OSError`** exceptions raised from scanning the filesystem are suppressed. This includes **`PermissionError`** when accessing directories without read permission.

Distinto en la versión 3.5: Soporte para globs recursivos usando «`**`».

Distinto en la versión 3.10: Se agregaron los parámetros *root_dir* y *dir_fd*.

Distinto en la versión 3.11: Agregado el parámetro *include_hidden*.

glob.iglob(*pathname*, ***, *root_dir=None*, *dir_fd=None*, *recursive=False*, *include_hidden=False*)

Retorna un iterador el cual produce los mismos valores que **`glob()`** sin necesidad de almacenarlos todos de forma simultánea.

Lanza un evento de auditoría `glob.glob` con los argumentos `pathname`, `recursive`.

Lanza un auditing event `glob.glob/2` con argumentos `pathname`, `recursive`, `root_dir`, `dir_fd`.

Nota

This function may return duplicate path names if *pathname* contains multiple «`**`» patterns and *recursive* is true.

Nota

Any **`OSError`** exceptions raised from scanning the filesystem are suppressed. This includes **`PermissionError`** when accessing directories without read permission.

Distinto en la versión 3.5: Soporte para globs recursivos usando

«**».

Distinto en la versión 3.10: Se agregaron los parámetros *root_dir* y *dir_fd*.

Distinto en la versión 3.11: Agregado el parámetro *include_hidden*.

glob.escape(*pathname*)

Escapa todos los caracteres especiales (`'?'`, `'*'` y `'['`). Esto es útil si deseas coincidir una cadena de caracteres literal arbitraria que podría contener caracteres especiales. Los caracteres especiales en unidades compartidas/UNC no se eluden, e.g. en Windows `escape('//?/c:/Quo vadis?.txt')` retorna `'//?/c:/Quo vadis[?].txt'`.

Added in version 3.4.

glob.translate(*pathname*, *, *recursive=False*, *include_hidden=False*, *seps=None*)

Convert the given path specification to a regular expression for use with `re.match()`. The path specification can contain shell-style wildcards.

For example:

```
>>> import glob, re
>>>
>>> regex = glob.translate('**/*.txt', recursive=Tr
>>> regex
'(?s:(?:.+/)?[^/]*\\.txt)\\z'
>>> reobj = re.compile(regex)
>>> reobj.match('foo/bar/baz.txt')
<re.Match object; span=(0, 15), match='foo/bar/baz.
```

Path separators and segments are meaningful to this function, unlike `fnmatch.translate()`. By default wildcards do not match path separators, and `*` pattern segments match precisely one path segment.

If *recursive* is true, the pattern segment «`**`» will match any number of path segments.

If *include_hidden* is true, wildcards can match path segments that start with a dot (`.`).

A sequence of path separators may be supplied to the *seps* argument. If not given, `os.sep` and `altsep` (if available) are used.

Ver también

`pathlib.PurePath.full_match()` and `pathlib.Path.glob()` methods, which call this function to implement pattern matching and globbing.

Added in version 3.13.

Examples

Consider a directory containing the following files: `1.gif`, `2.txt`, `card.gif` and a subdirectory `sub` which contains only the file `3.txt`. `glob()` will produce the following results. Notice how any leading components of the path are preserved.

```
>>> import glob
>>> glob.glob('./[0-9].*')
['./1.gif', './2.txt']
>>> glob.glob('*.gif')
['1.gif', 'card.gif']
>>> glob.glob('?.gif')
['1.gif']
>>> glob.glob('**/*.txt', recursive=True)
['2.txt', 'sub/3.txt']
>>> glob.glob('./**/', recursive=True)
['./', './sub/']
```

Si un directorio contiene ficheros que comienzan con `.` no coincidirá por defecto. Por ejemplo, considera un directorio que

contiene `card.gif` y `.card.gif`:

```
>>> import glob
>>> glob.glob('*.gif')
['card.gif']
>>> glob.glob('.c*')
['.card.gif']
```

Ver también

The **`fnmatch`** module offers shell-style filename (not path) expansion.

Ver también

El módulo **`pathlib`** ofrece objetos ruta de alto nivel.

`fnmatch` — Unix filename pattern matching

Código fuente: Lib/fnmatch.py [https://github.com/python/cpython/tree/3.14/Lib/fnmatch.py]

Este módulo proporciona soporte para comodines de estilo shell de Unix, que *no* son lo mismo que las expresiones regulares (que se documentan en el módulo `re`). Los caracteres especiales utilizados en los comodines de estilo shell son:

Patrón	Significado
`*`	coincide con todo
`?`	coincide con un solo carácter
`[seq]`	coincide con cualquier carácter presente en *seq*
`[!seq]`	coincide con cualquier carácter ausente en *seq*

Para una coincidencia literal, envuelva los meta-caracteres entre paréntesis. Por ejemplo, `'[?]'` coincide con el carácter `'?'`.

Ten en cuenta que el separador de nombre de archivo (`'/'` en Unix) *no* es tratado de forma especial en este módulo. Consulta el módulo `glob` para realizar expansiones de nombre de ruta (`glob` usa `filter()` para hacer coincidir con los componentes del nombre de ruta). Del mismo modo, los nombres de archivo que comienzan con un punto tampoco son tratados de forma especial por este módulo y se corresponden con los patrones `*` y `?`.

Unless stated otherwise, «filename string» and «pattern string» either refer to `str` or `ISO-8859-1` encoded `bytes` objects. Note that the functions documented below do not allow to mix a `bytes` pattern with a `str` filename, and vice-versa.

Finally, note that `functools.lru_cache()` with a *maxsize* of 32768 is used to cache the (typed) compiled regex patterns in the

following functions: **`fnmatch()`**, **`fnmatchcase()`**, **`filter()`**, **`filterfalse()`**.

fnmatch.fnmatch(*name*, *pat*)

Test whether the filename string *name* matches the pattern string *pat*, returning `True` or `False`. Both parameters are case-normalized using **`os.path.normcase()`**. **`fnmatchcase()`** can be used to perform a case-sensitive comparison, regardless of whether that's standard for the operating system.

Este ejemplo imprimirá todos los nombres de archivo en el directorio actual con la extensión `.txt`:

```
import fnmatch
import os

for file in os.listdir('.'):
    if fnmatch.fnmatch(file, '*.txt'):
        print(file)
```

fnmatch.fnmatchcase(*name*, *pat*)

Test whether the filename string *name* matches the pattern string *pat*, returning `True` or `False`; the comparison is case-sensitive and does not apply **`os.path.normcase()`**.

fnmatch.filter(*names*, *pat*)

Construct a list from those elements of the iterable of filename strings *names* that match the pattern string *pat*. It is the same as `[n for n in names if fnmatch(n, pat)]`, but implemented more efficiently.

fnmatch.filterfalse(*names*, *pat*)

Construct a list from those elements of the iterable of filename strings *names* that do not match the pattern string *pat*. It is the same as `[n for n in names if not fnmatch(n, pat)]`, but implemented more efficiently.

Added in version 3.14.

fnmatch.translate(*pat*)

Return the shell-style pattern *pat* converted to a regular expression for using with `re.match()`. The pattern is expected to be a `str`.

Ejemplo:

```
>>> import fnmatch, re
>>>
>>> regex = fnmatch.translate('*.txt')
>>> regex
'(?s:.*\\.txt)\\z'
>>> reobj = re.compile(regex)
>>> reobj.match('foobar.txt')
<re.Match object; span=(0, 10), match='foobar.txt'>
```

Ver también

Módulo `glob`

Expansión de ruta estilo shell en Unix.

`linecache` — Random access to text lines

Código fuente: Lib/linecache.py [https://github.com/python/cpython/tree/3.14/Lib/linecache.py]

The **`linecache`** module allows one to get any line from a Python source file, while attempting to optimize internally, using a cache, the common case where many lines are read from a single file. This is used by the **`traceback`** module to retrieve source lines for inclusion in the formatted traceback.

La función **`tokenize.open()`** se utiliza para abrir archivos. Esta función usa **`tokenize.detect_encoding()`** para obtener la codificación del archivo; en la ausencia de un token de codificación, la codificación del archivo es por defecto UTF-8.

The **`linecache`** module defines the following functions:

linecache.getline(*filename*, *lineno*, *module_globals=None*)

Obtiene la línea *lineno* del archivo llamado *filename*. Esta función nunca lanzará una excepción — retornará `''` en los errores (el carácter de la nueva línea de terminación se incluirá para las líneas que se encuentren).

If *filename* indicates a frozen module (starting with `'<frozen '`), the function will attempt to get the real file name from `module_globals['__file__']` if *module_globals* is not `None`.

Si un archivo llamado *filename* no se encuentra, la función primero verifica un `__loader__` en *module_globals* **PEP 302** [https://peps.python.org/pep-0302/]. Si existe tal cargador y define un método `get_source`, entonces eso determina las líneas de origen (si `get_source()` retorna `None`, entonces se retorna `''`). Finalmente, si *filename* es un nombre de fichero relativo, se busca en relación a las entradas en la ruta de búsqueda del módulo, `sys.path`.

Distinto en la versión 3.14: Support *filename* of frozen modules.

linecache.clearcache()

Borra el caché. Use esta función si ya no necesita las líneas de archivos leídos previamente usando **`getline()`**.

linecache.checkcache(*filename=None*)

Comprueba la validez de la caché. Use esta función si los archivos de la caché pueden haber cambiado en disco y necesita la versión actualizada. Si se omite *filename*, comprobará todas las entradas en la caché.

linecache.lazycache(*filename*, *module_globals*)

Captura suficientes detalles sobre un módulo no basado en archivos para permitir obtener sus líneas más tarde a través de **`getline()`** incluso si *module_globals* es `None` en la llamada posterior. Esto evita hacer E/S hasta que una línea es realmente necesaria, sin tener que llevar los módulo globales indefinidamente.

Added in version 3.5.

Ejemplo:

```
>>> import linecache
>>> linecache.getline(linecache.__file__, 8)
'import sys\n'
```

`shutil` — High-level file operations

Código fuente: Lib/shutil.py [https://github.com/python/cpython/tree/3.14/Lib/shutil.py]

The `shutil` module offers a number of high-level operations on files and collections of files. In particular, functions are provided which support file copying and removal. For operations on individual files, see also the `os` module.

Advertencia

Incluso las funciones de copia de archivos de nivel superior (`shutil.copy()`, `shutil.copy2()`) no pueden copiar todos los metadatos del archivo.

En las plataformas POSIX, esto significa que tanto el propietario como el grupo del archivo se pierden, al igual que las ACLs (*access control lists* o lista de control de acceso). En Mac OS, la bifurcación (*fork*) de recursos y de otros metadatos no se utilizan. Esto quiere decir que los recursos se perderán y que el tipo de archivo y los códigos de creador no serán correctos. En Windows, los propietarios de archivos, las ACLs y secuencias de datos alternativas no se copian.

Operaciones de directorios y archivos

shutil.copyfileobj(*fsrc*, *fdst*[, *length*])

Copy the contents of the file-like object *fsrc* to the file-like object *fdst*. The integer *length*, if given, is the buffer size. In particular, a negative *length* value means to copy the data without looping over the source data in chunks; by default the data is read in chunks to avoid uncontrolled memory consumption. Note that if the current file position of the *fsrc* object is not 0, only the contents from the current file position to the end of the file will be copied.

`copyfileobj()` will *not* guarantee that the destination stream has been flushed on completion of the copy. If you want to read from the destination at the completion of the copy operation (for example, reading the contents of a temporary file that has been copied from a HTTP stream), you must ensure that you have called **`flush()`** or **`close()`** on the file-like object before attempting to read the destination file.

shutil.copyfile(*src*, *dst*, ***, *follow_symlinks=True*)

Copy the contents (no metadata) of the file named *src* to a file named *dst* and return *dst* in the most efficient way possible. *src* and *dst* are path-like objects or path names given as strings.

dst debe ser el nombre completo del archivo destino; véase **`shutil.copy()`** para una copia que acepta una ruta de directorio destino. Si *src* y *dst* refieren al mismo archivo, se lanza **`SameFileError`**.

El local de destino debe tener permisos de escritura; de lo contrario, se genera una excepción **`OSError`**. Si *dst* ya existe, se reemplazará. Los archivos especiales, como los dispositivos de caracteres o de bloques y *pipes*, no se pueden copiar con esta función.

Si *follow_symlinks* es falso y *src* es un enlace simbólico, un enlace simbólico nuevo se creará en lugar de copiar el archivo al que *src* apunta.

Se genera un evento de auditoría `shutil.copyfile` con argumentos `src`, `dst`.

Distinto en la versión 3.3: Solía generarse **`IOError`** en lugar de **`OSError`**. Se añadió el argumento *follow_symlinks*. Ahora retorna *dst*.

Distinto en la versión 3.4: Genera **`SameFileError`** en lugar de **`Error`**. Dado que la primera es una subclase de la segunda, el cambio es compatible con versiones anteriores.

Distinto en la versión 3.8: Las llamadas a sistema de copia rápida específicas de la plataforma se pueden usar internamente para copiar el archivo de manera más eficiente. Véase la sección Operaciones de copia eficientes dependientes de la plataforma.

exception shutil.SpecialFileError

This exception is raised when **`copyfile()`** or **`copytree()`** attempt to copy a named pipe.

Added in version 2.7.

exception shutil.SameFileError

Esta excepción se genera si el origen y destino en **`copyfile()`** son el mismo archivo.

Added in version 3.4.

shutil.copymode(*src*, *dst*, *, *follow_symlinks=True*)

Copy the permission bits from *src* to *dst*. The file contents, owner, and group are unaffected. *src* and *dst* are path-like objects or path names given as strings. If *follow_symlinks* is false, and both *src* and *dst* are symbolic links, **`copymode()`** will attempt to modify the mode of *dst* itself (rather than the file it points to). This functionality is not available on every platform; please see **`copystat()`** for more information. If **`copymode()`** cannot modify symbolic links on the local platform, and it is asked to do so, it will do nothing and return.

Genera un evento de auditoría `shutil.copymode` con argumentos `src`, `dst`.

Distinto en la versión 3.3: Se añadió el argumento *follow_symlinks*.

shutil.copystat(*src*, *dst*, *, *follow_symlinks=True*)

Copy the permission bits, last access time, last modification time, and flags from *src* to *dst*. On Linux, **`copystat()`** also copies the «extended attributes» where possible. The file contents, owner, and group are unaffected. *src* and *dst* are path-like objects or path names given as strings.

Si *follow_symlinks* es falso, y *src* y *dst* hacen referencia los enlaces simbólicos, **`copystat()`** funcionará con los enlaces simbólicos en lugar de en los archivos a los que estos se refieren: leer la información del enlace simbólico *src* y escribirla en el enlace simbólico *dst*.

Nota

No todas las plataformas proporcionan la capacidad de examinar y modificar enlaces simbólicos. Python puede indicarte qué funcionalidad está disponible localmente.

- Si `os.chmod in os.supports_follow_symlinks` es `True`, **`copystat()`** puede modificar los bits de permiso de un enlace simbólico.
- Si `os.utime in os.supports_follow_symlinks` es `True`, **`copystat()`** puede modificar el último acceso y las veces que un enlace simbólico fue modificado.
- Si `os.chflags in os.supports_follow_symlinks` es `True`, **`copystat()`** puede modificar las flags de un enlace simbólico. (`os.chflags` no está disponible en todas las plataformas.)

En plataformas donde parte o toda esta funcionalidad no está disponible, cuando se le pida modificar un enlace simbólico, **`copystat()`** copiará todo lo que pueda. **`copystat()`** nunca retorna un error.

Véase **`os.supports_follow_symlinks`** para más información.

Genera un evento de auditoría `shutil.copystat` con argumentos `src`, `dst`.

Distinto en la versión 3.3: Se ha añadido el argumento *follow_symlinks* y el soporte para atributos extendidos de Linux.

shutil.copy(*src*, *dst*, ***, *follow_symlinks=True*)

Copia el archivo *src* al archivo o directorio *dst*. *src* y *dst* deberían ser objetos tipo ruta o cadenas de caracteres (*strings*). Si *dst* especifica un directorio, el archivo será copiado en *dst* usando el nombre de archivo base de *src*. Si *dst* especifica un archivo que existe, será reemplazado. Retorna la ruta del archivo recién creado.

Si *follow_symlinks* es falso, y *src* es un enlace simbólico, *dst* se creará como enlace simbólico. Si *follow_symlinks* es verdadero y *src* es un enlace simbólico, *dst* será una copia del archivo al que *src* hace referencia.

`copy()` copia los datos del archivo y el modo de permiso del archivo (véase **`os.chmod()`**). Otros metadatos, como los tiempos de creación y modificación de archivos, no se preservan. Para preservar todos los metadatos del archivo, usa **`copy2()`** en su lugar.

Se genera un evento de auditoría `shutil.copyfile` con argumentos `src`, `dst`.

Genera un evento de auditoría `shutil.copymode` con argumentos `src`, `dst`.

Distinto en la versión 3.3: Se ha añadido el argumento *follow_symlinks*. Ahora retorna la ruta de acceso al archivo recién creado.

Distinto en la versión 3.8: Las llamadas a sistema de copia rápida específicas de la plataforma se pueden usar internamente para copiar el archivo de manera más eficiente. Véase la sección Operaciones de copia eficientes dependientes de la plataforma.

shutil.copy2(*src*, *dst*, ***, *follow_symlinks=True*)

Idéntico a **`copy()`**, excepto que **`copy2()`** también intenta conservar los metadatos del archivo.

Cuando *follow_symlinks* es falso, y *src* es un enlace simbólico, **`copy2()`** intenta copiar todos los metadatos del enlace simbólico *src* en el enlace simbólico *dst* recién creado. Sin embargo, esta funcionalidad no está disponible en todas las plataformas. En las plataformas donde parte o toda esta funcionalidad no está disponible, **`copy2()`** conservará todos los metadatos que pueda; **`copy2()`** nunca genera una excepción porque no puede conservar los metadatos del archivo.

`copy2()` usa **`copystat()`** para copiar todos los metadatos del archivo. Véase **`copystat()`** para más información sobre la compatibilidad con la plataforma para modificar los metadatos del enlace simbólico.

Se genera un evento de auditoría `shutil.copyfile` con argumentos `src`, `dst`.

Genera un evento de auditoría `shutil.copystat` con argumentos `src`, `dst`.

Distinto en la versión 3.3: Añadido el argumento *follow_symlinks*, intenta copiar también los atributos del sistema de archivos extendidos (actualmente solo en Linux). Ahora retorna la ruta de acceso al archivo recién creado.

Distinto en la versión 3.8: Las llamadas a sistema de copia rápida específicas de la plataforma se pueden usar internamente para copiar el archivo de manera más eficiente. Véase la sección Operaciones de copia eficientes dependientes de la plataforma.

shutil.ignore_patterns(**patterns*)

Esta función de fábrica crea una función que puede ser usada como un invocable para el argumento *ignore* de **`copytree()`**, ignorando los archivos y directorios que coinciden con uno de los patrones *patterns* de estilo glob provistos. Véase el ejemplo siguiente.

shutil.copytree(*src, dst, symlinks=False, ignore=None, copy_function=copy2, ignore_dangling_symlinks=False, dirs_exist_ok=False*)

Copia recursivamente un directorio completo que inicia en *src* hasta un directorio llamado *dst* y devuelve el directorio destino. Todos los directorios intermedios necesarios para contener *dst* serán creados por default.

Los permisos y los tiempos de directorios son copiados con **`copystat()`**; los archivos individuales son copiados usando **`copy2()`**.

Si *symlinks* es verdadero, los enlaces simbólicos en el árbol de origen son representados como enlaces simbólicos en el árbol nuevo y los metadatos de los enlaces originales serán copiados mientras la plataforma lo permita; si es falso o se omite, los contenidos y los metadatos de los archivos vinculados se copiarán en el árbol nuevo.

When *symlinks* is false, if the file pointed to by the symlink doesn't exist, an exception will be added in the list of errors raised in an **`Error`** exception at the end of the copy process. You can set the optional *ignore_dangling_symlinks* flag to true if you want to silence this exception. Notice that this option has no effect on platforms that don't support **`os.symlink()`**.

Si se proporciona *ignore*, debe ser un invocable que recibirá como argumentos el directorio visitado por **`copytree()`** y una lista de sus contenidos, tal como los retorna **`os.listdir()`**. Ya que **`copytree()`** se invoca recursivamente, el invocable *ignore* se llamará una vez por cada directorio que se copia. El invocable debe retornar una secuencia de directorio y de nombres de archivo en relación con el directorio actual (es decir, un subconjunto de los elementos en su segundo argumento); estos nombres serán ignorados en el proceso de copia. **`ignore_patterns()`** se pueden usar para crear un invocable que ignora los nombres basados en patrones de estilo gob.

Si ocurren excepciones, se genera, un **`Error`** con una lista de razones.

Si *copy_function* está dado, debe ser un invocable que se usará para copiar cada archivo. Se le invocará con la ruta de origen y la ruta de destino como argumentos. Por defecto, se usa **`copy2()`**, pero se puede usar cualquier función que admita la misma (como **`copy()`**).

Si *dirs_exist_ok* es falso (el default) y *dst* existe, la excepción **`FileExistsError`** es lanzada. Si *dirs_exist_ok* es verdadero, la operación de copiado continuara si encuentra directorios existentes, los archivos dentro del árbol de *dst* serán sobreescritos por los archivos correspondientes del árbol *src*.

Se genera un evento de auditoría `shutil.copytree` con argumentos `src`, `dst`.

Distinto en la versión 3.2: Se añadió el argumento *copy_function* para poder proporcionar una función de copia personalizada. Se añadió el argumento *ignore_dangling_symlinks* para silenciar los errores de enlaces colgantes cuando *symlinks* es falso.

Distinto en la versión 3.3: Copia los metadatos cuando *symlinks* es falso. Ahora retorna *dst*.

Distinto en la versión 3.8: Las llamadas a sistema de copia rápida específicas de la plataforma se pueden usar internamente para copiar el archivo de manera más eficiente. Véase la sección Operaciones de copia eficientes dependientes de la plataforma.

Distinto en la versión 3.8: Added the *dirs_exist_ok* parameter.

shutil.rmtree(*path*, *ignore_errors=False*, *onerror=None*, ***, *onexc=None*, *dir_fd=None*)

Delete an entire directory tree; *path* must point to a directory (but not a symbolic link to a directory). If *ignore_errors* is true, errors resulting from failed removals will be ignored; if false or omitted, such errors are handled by calling a handler specified by *onexc* or *onerror* or, if both are omitted, exceptions are propagated to the caller.

Esta función puede soportar rutas relativas a descriptores de directorio.

Nota

En plataformas que admiten las funciones basadas en descriptores de archivo necesarias, una versión resistente de `rmtree()` al ataque de enlace simbólico se usa por defecto. En otras plataformas, la implementación `rmtree()` es susceptible a un ataque de enlace simbólico; dados el tiempo y las circunstancias adecuados, los atacantes pueden manipular los enlaces simbólicos en el sistema de archivos para eliminar archivos a los que no podrían acceder de otra manera. Las aplicaciones pueden usar el atributo de función `rmtree.avoids_symlink_attacks` para determinar qué caso aplica.

If *onexc* is provided, it must be a callable that accepts three parameters: *function*, *path*, and *excinfo*.

The first parameter, *function*, is the function which raised the exception; it depends on the platform and implementation. The second parameter, *path*, will be the path name passed to *function*. The third parameter, *excinfo*, is the exception that was raised. Exceptions raised by *onexc* will not be caught.

The deprecated *onerror* is similar to *onexc*, except that the third parameter it receives is the tuple returned from `sys.exc_info()`.

Ver también

ejemplo de rmtree for an example of handling the removal of a directory tree that contains read-only files.

Genera un evento de auditoría `shutil.rmtree` con argumentos `path`, `dir_fd`.

Distinto en la versión 3.3: Se añadió una versión resistente a los ataques de enlaces simbólicos que se usan automáticamente si la plataforma admite funciones pasadas en descriptores de archivo.

Distinto en la versión 3.8: En Windows, ya no eliminará los contenidos de un cruce de directorios antes de quitar el cruce.

Distinto en la versión 3.11: Added the *dir_fd* parameter.

Distinto en la versión 3.12: Added the *onexc* parameter, deprecated *onerror*.

Distinto en la versión 3.13: **`rmtree()`** now ignores **`FileNotFoundError`** exceptions for all but the top-level path. Exceptions other than **`OSError`** and subclasses of **`OSError`** are now always propagated to the caller.

rmtree.avoids_symlink_attacks

Indica si la plataforma actual y la implementación proporciona una versión de **`rmtree()`** resistente al ataque de enlace simbólico. Actualmente, esto solo sucede para plataformas que admitan funciones de acceso a directorios basadas en descriptores de archivo.

Added in version 3.3.

shutil.move(*src*, *dst*, *copy_function=copy2*)

Recursively move a file or directory (*src*) to another location and return the destination.

If *dst* is an existing directory or a symlink to a directory, then *src* is moved inside that directory. The destination path in that directory must not already exist.

If *dst* already exists but is not a directory, it may be overwritten depending on `os.rename()` semantics.

If the destination is on the current filesystem, then `os.rename()` is used. Otherwise, *src* is copied to the destination using *copy_function* and then removed. In case of symlinks, a new symlink pointing to the target of *src* will be created as the destination and *src* will be removed.

If *copy_function* is given, it must be a callable that takes two arguments, *src* and the destination, and will be used to copy *src* to the destination if `os.rename()` cannot be used. If the source is a directory, `copytree()` is called, passing it the *copy_function*. The default *copy_function* is `copy2()`. Using `copy()` as the *copy_function* allows the move to succeed when it is not possible to also copy the metadata, at the expense of not copying any of the metadata.

Genera un evento de auditoría `shutil.move` con argumentos `src`, `dst`.

Distinto en la versión 3.3: Se añadió el manejo explícito de enlaces simbólicos para sistemas de archivos extranjeros, de manera que se adapta al comportamiento del **mv** del GNU. Ahora retorna *dst*.

Distinto en la versión 3.5: Se agregó el argumento de keyword *copy_function*.

Distinto en la versión 3.8: Las llamadas a sistema de copia rápida específicas de la plataforma se pueden usar internamente para copiar el archivo de manera más eficiente. Véase la sección Operaciones de copia eficientes dependientes de la plataforma.

Distinto en la versión 3.9: Acepta un objeto tipo ruta como *src* y *dst*.

shutil.disk_usage(*path*)

Retorna las estadísticas de uso de disco sobre la ruta de acceso dada como un named tuple con los atributos *total*, *used* y *free*, que son las cantidades del espacio total, el usado y el libre, en bytes. *path* puede ser un archivo o un directorio.

Nota

On Unix filesystems, *path* must point to a path within a **mounted** filesystem partition. On those platforms, CPython doesn't attempt to retrieve disk usage information from non-mounted filesystems.

Added in version 3.3.

Distinto en la versión 3.8: En Windows, *path* ahora puede ser un archivo o un directorio.

Availability: Unix, Windows.

shutil.chown(*path*, *user=None*, *group=None*, ***, *dir_fd=None*, *follow_symlinks=True*)

Cambia el propietario *user* y/o *group* de la *ruta* dada.

user puede ser un nombre usuario de sistema o un UID (identificador de usuario); lo mismo aplica a *group*. Se requiere al menos un argumento.

Véase también **`os.chown()`**, la función subyacente.

Genera un evento de auditoría `shutil.chown` con argumentos `path`, `user`, `group`.

Availability: Unix.

Added in version 3.3.

Distinto en la versión 3.13: Added *dir_fd* and *follow_symlinks* parameters.

shutil.which(*cmd*, *mode=os.F_OK | os.X_OK*, *path=None*)

Retorna la ruta de acceso a un ejecutable que se ejecutaría si el *cmd* dado se invoca. Si no se invoca a *cmd*, retorna `None`.

mode is a permission mask passed to `os.access()`, by default determining if the file exists and is executable.

path is a «`PATH` string» specifying the directories to look in, delimited by `os.pathsep`. When no *path* is specified, the **`PATH`** environment variable is read from `os.environ`, falling back to `os.defpath` if it is not set.

If *cmd* contains a directory component, **`which()`** only checks the specified path directly and does not search the directories listed in *path* or in the system's **`PATH`** environment variable.

On Windows, the current directory is prepended to the *path* if *mode* does not include `os.X_OK`. When the *mode* does include `os.X_OK`, the Windows API `NeedCurrentDirectoryForExePathW` will be consulted to determine if the current directory should be prepended to *path*. To avoid consulting the current working directory for executables: set the environment variable `NoDefaultCurrentDirectoryInExePath`.

Also on Windows, the **`PATHEXT`** environment variable is used to resolve commands that may not already include an extension. For example, if you call `shutil.which("python")`, **`which()`** will search `PATHEXT` to know that it should look for `python.exe` within the *path* directories. For example, on Windows:

```
>>> shutil.which("python")
'C:\\Python33\\python.EXE'
```

This is also applied when *cmd* is a path that contains a directory component:

```
>>> shutil.which("C:\\Python33\\python")
'C:\\Python33\\python.EXE'
```

Added in version 3.3.

Distinto en la versión 3.8: Ahora se acepta el tipo **`bytes`**. Si el tipo *cmd* es **`bytes`**, el tipo resultante también es **`bytes`**.

Distinto en la versión 3.12: On Windows, the current directory is no longer prepended to the search path if *mode* includes `os.X_OK` and WinAPI `NeedCurrentDirectoryForExePathW(cmd)` is false, else the current directory is prepended even if it is already in the search path; `PATHEXT` is used now even when *cmd* includes a directory component or ends with an extension that is in `PATHEXT`; and filenames that have no extension can now be found.

exception shutil.Error

Esta excepción recopila las excepciones que se generan durante una operación de varios archivos. Para **`copytree()`**, el argumento de excepción es una lista de 3 tuplas (*scrname*, *dstname*, *exception*).

Operaciones de copia eficientes dependientes de la plataforma

A partir de Python 3.8, todas las funciones que incluyen una copia de archivo (**`copyfile()`**, **`copy()`**, **`copy2()`**, **`copytree()`**, y **`move()`**) puede usar llamadas a sistema de «copia rápida» específicas a cada plataforma para poder copiar el archivo de forma más eficiente (véase bpo-33671 [https://bugs.python.org/issue?@action=redirect&bpo=33671]). «copia rápida» significa que la operación de copia ocurre dentro del núcleo, evitando el uso de búferes en espacio de usuario en Python como en «`outfd.write(infd.read())`».

En macOS, se usa fcopyfile [http://www.manpagez.com/man/3/copyfile/] para copiar el contenido del archivo (pero no los metadatos).

On Linux **`os.copy_file_range()`** or **`os.sendfile()`** is used.

On Solaris **`os.sendfile()`** is used.

En Windows, **`shutil.copyfile()`** usa un tamaño de búfer predeterminado más grande (1 MiB en lugar de 64 KiB) y se usa una variante de **`shutil.copyfileobj()`** basada en **`memoryview()`**.

If the fast-copy operation fails and no data was written in the destination file then shutil will silently fall back to less efficient **`copyfileobj()`** function internally.

Distinto en la versión 3.8.

Distinto en la versión 3.14: Solaris now uses **`os.sendfile()`**.

Distinto en la versión 3.14: Copy-on-write or server-side copy may be used internally via **`os.copy_file_range()`** on supported Linux filesystems.

ejemplo de *copytree*

Un ejemplo que usa el auxiliar **`ignore_patterns()`**:

```
from shutil import copytree, ignore_patterns

copytree(source, destination, ignore=ignore_patterns('*.
```

Esto copiará todo excepto archivos `.pyc` y archivos o directorios cuyo nombre comience con `tmp`.

Otro ejemplo que usa el argumento *ignore* para agregar una llamada a iniciar sesión:

```
from shutil import copytree
import logging

def _logpath(path, names):
    logging.info('Working in %s', path)
    return []   # nothing will be ignored

copytree(source, destination, ignore=_logpath)
```

ejemplo de rmtree

This example shows how to remove a directory tree on Windows where some of the files have their read-only bit set. It uses the onexc callback to clear the readonly bit and reattempt the remove. Any subsequent failure will propagate.

```
import os, stat
import shutil

def remove_readonly(func, path, _):
    "Clear the readonly bit and reattempt the removal"
    os.chmod(path, stat.S_IWRITE)
    func(path)

shutil.rmtree(directory, onexc=remove_readonly)
```

Operaciones de archivado

Added in version 3.2.

Distinto en la versión 3.5: Se agregó soporte para el formato *xztar*.

Se proveen también las utilidades de alto nivel para crear y leer archivos comprimidos y archivados. Utilizan para esto los módulos **`zipfile`** y **`tarfile`**.

shutil.make_archive(*base_name*, *format*[, *root_dir*[, *base_dir*[, *verbose*[, *dry_run*[, *owner*[, *group*[, *logger*]]]]]]])

Crea un documento de archivado (como zip o tar) y retorna su nombre.

base_name is the name of the file to create, including the path, minus any format-specific extension.

format is the archive format: one of «zip» (if the **`zlib`** module is available), «tar», «gztar» (if the **`zlib`** module is available), «bztar» (if the **`bz2`** module is available), «xztar» (if the **`lzma`** module is available), or «zstdtar» (if the **`compression.zstd`** module is available).

root_dir es un directorio que será el directorio raíz del archivo, todas las rutas en el archivo serán relativas a él; por ejemplo, típicamente nos cambiaremos de directorio (*chdir*) a *root_dir* antes de crear el archivo.

base_dir es el directorio desde donde iniciamos el archivado; i.e, *base_dir* será el prefijo común para todos los archivos y directorios en el archivo. *base_dir* debe ser dado de forma relativa a *root_dir*. En Ejemplo de archivado con base_dir se ve un ejemplo de cómo usar *base_dir* y *root_dir* juntos.

root_dir y *base_dir* irán por defecto al directorio actual.

If *dry_run* es verdadero, no se crea un archivo, pero las operaciones que se ejecutarán están registradas en *logger*.

owner y *group* se usan al crear un archivador tar. Por defecto, usa el propietario y el grupo actual.

logger debe ser un objeto compatible con **PEP 282** [https://peps.python.org/pep-0282/], usualmente una instancia de **`logging.Logger`**.

El argumento *verbose* está fuera de uso y es obsoleto.

Genera un evento de auditoría `shutil.make_archive` con argumentos `base_name`, `format`, `root_dir`, `base_dir`.

Nota

This function is not thread-safe when custom archivers registered with **`register_archive_format()`** do not support the *root_dir* argument. In this case it temporarily changes the current working directory of the process to *root_dir* to perform archiving.

Distinto en la versión 3.8: El formato de *pax* moderno (POSIX.1-2001) se usa ahora en lugar del formato de legado GNU para archivadores creados con `format="tar"`.

Distinto en la versión 3.10.6: Esta función ahora es segura para hilos durante la creación de archivos estándar `.zip` y tar.

shutil.get_archive_formats()

Retorna una lista de formatos admitidos para archivado. Cada elemento de una secuencia retornada es una tupla `(name, description)`.

By default **`shutil`** provides these formats:

- *zip*: archivo ZIP (si el módulo **`zlib`** está disponible).
- *tar*: archivo tar sin comprimir. Utiliza el formato pax POSIX.1-2001 para archivos nuevos.
- *gztar*: archivo tar comprimido con gzip (si el módulo **`zlib`** está disponible).
- *bztar*: archivo tar comprimido con bzip2 (si el módulo **`bz2`** está disponible).
- *xztar*: archivo tar comprimido con xz (si el módulo **`lzma`** está disponible).
- *zstdtar*: Zstandard compressed tar-file (if the **`compression.zstd`** module is available).

Puedes registrar formatos nuevos o proporcionar tu propio archivador para cualquier formato existente, usando **`register_archive_format()`**.

shutil.register_archive_format(*name*, *function*[, *extra_args*[, *description*]])

Registra un archivador para el formato *name*.

function es el invocable que se usará para desempacar archivos. El invocable recibirá el *base_name* del archivo para crearlo, seguido por *base_dir* (que cae por defecto en **`os.curdir`**) para comenzar a archivar desde allí. Otros argumentos se pasan como argumentos de palabras clave: *owner*, *group*, *dry_run* y *logger* (como se pasa en **`make_archive()`**).

If *function* has the custom attribute `function.supports_root_dir` set to `True`, the *root_dir* argument is passed as a keyword argument. Otherwise the current working directory of the process is temporarily changed to *root_dir* before calling *function*. In this case `make_archive()` is not thread-safe.

Si está dado, *extra_args* es una secuencia de pares `(name, value)` que se usarán como argumentos de palabras clave extra cuando se usa el archivador invocable.

description se usa por `get_archive_formats()` que retorna la lista de archivadores. Cae por defecto en una cadena de caracteres vacía.

Distinto en la versión 3.12: Added support for functions supporting the *root_dir* argument.

shutil.unregister_archive_format(*name*)

Elimina el formato de archivo *name* de la lista de formatos admitidos.

shutil.unpack_archive(*filename*[, *extract_dir*[, *format*[, *filter*]]])

Desempaqueta un archivo. *filename* es la ruta completa del archivo.

extract_dir es el nombre del directorio donde se desempaca el archivo. Si no está provisto, se usa el nombre del directorio actual.

format is the archive format: one of «zip», «tar», «gztar», «bztar», «xztar», or «zstdtar». Or any other format registered with `register_unpack_format()`. If not provided, **`unpack_archive()`** will use the archive file name extension and see if an unpacker was registered for that extension. In case none is found, a `ValueError` is raised.

The keyword-only *filter* argument is passed to the underlying unpacking function. For zip files, *filter* is not accepted. For tar files, it is recommended to use `'data'` (default since Python 3.14), unless using features specific to tar and UNIX-like filesystems. (See Extraction filters for details.)

Se genera un evento de auditoría `shutil.unpack_archive` con argumentos `filename`, `extract_dir`, `format`.

Advertencia

Nunca extraiga archivos de fuentes no confiables sin una inspección previa. Es posible que los archivos sean creados fuera de la ruta especificada en el argumento *extract_dir*, p.e. miembros que tienen nombres de archivo absolutos comenzando con "/" or nombres de archivos con dos puntos "..".

Since Python 3.14, the defaults for both built-in formats (zip and tar files) will prevent the most dangerous of such security issues, but will not prevent *all* unintended behavior. Read the Hints for further verification section for tar-specific details.

Distinto en la versión 3.7: Acepta un path-like object como *filename* y *extract_dir*.

Distinto en la versión 3.12: Added the *filter* argument.

shutil.register_unpack_format(*name*, *extensions*, *function*[, *extra_args*[, *description*]])

Registra un formato de desempaque. *name* es el nombre del formato y *extensions* es una lista de extensiones que corresponden al formato, como `.zip` para archivos zip.

function is the callable that will be used to unpack archives. The callable will receive:

- the path of the archive, as a positional argument;
- the directory the archive must be extracted to, as a positional argument;
- possibly a *filter* keyword argument, if it was given to **`unpack_archive()`**;
- additional keyword arguments, specified by *extra_args* as a sequence of `(name, value)` tuples.

description se puede proporcionar para describir el formato y será retornado por la función `get_unpack_formats()`.

shutil.unregister_unpack_format(*name*)

Anula el registro del formato de desempaque. *name* es el nombre del formato.

shutil.get_unpack_formats()

Retorna una lista de todos los formatos registrados para desempaquetar. Cada elemento de la secuencia es una tupla `(name, extensions, description)`.

By default **`shutil`** provides these formats:

- *zip*: archivo zip (desempaquetar archivos comprimidos funciona solo si el módulo correspondiente está disponible).
- *tar*: archivo tar sin comprimir.
- *gztar*: archivo tar comprimido con gzip (si el módulo `zlib` está disponible).
- *bztar*: archivo tar comprimido con bzip2 (si el módulo `bz2` está disponible).
- *xztar*: archivo tar comprimido con xz (si el módulo `lzma` está disponible).
- *zstdtar*: Zstandard compressed tar-file (if the `compression.zstd` module is available).

Puedes registrar formatos nuevos o proporcionar tu propio desempaquetado para cualquier formato existente, usando `register_unpack_format()`.

Ejemplo de archivado

En este ejemplo, creamos un archivo tar de tipo gzip que contiene todos los archivos que se encuentran en el directorio `.ssh` del usuario:

```
>>> from shutil import make_archive
>>> import os
>>> archive_name = os.path.expanduser(os.path.join('~',
>>> root_dir = os.path.expanduser(os.path.join('~', '.ss
>>> make_archive(archive_name, 'gztar', root_dir)
'/Users/tarek/myarchive.tar.gz'
```

El archivo resultante contiene:

```
$ tar -tzvf /Users/tarek/myarchive.tar.gz
drwx------ tarek/staff       0 2010-02-01 16:23:40 ./
-rw-r--r-- tarek/staff     609 2008-06-09 13:26:54 ./aut
-rwxr-xr-x tarek/staff      65 2008-06-09 13:26:54 ./cor
-rwx------ tarek/staff     668 2008-06-09 13:26:54 ./id_
-rwxr-xr-x tarek/staff     609 2008-06-09 13:26:54 ./id_
-rw------- tarek/staff    1675 2008-06-09 13:26:54 ./id_
-rw-r--r-- tarek/staff     397 2008-06-09 13:26:54 ./id_
-rw-r--r-- tarek/staff   37192 2010-02-06 18:23:10 ./kno
```

Ejemplo de archivado con *base_dir*

En este ejemplo, similar al de arriba, mostramos cómo usar **`make_archive()`**, pero esta vez utilizando *base_dir*. Ahora tenemos la siguiente estructura de directorios:

```
$ tree tmp
tmp
    root
        structure
            content
                please_add.txt
            do_not_add.txt
```

En el archivo final, `please_add.txt` debería estar incluido, pero `do_not_add.txt` no. Por lo tanto usamos lo siguiente:

```
>>> from shutil import make_archive
>>> import os
>>> archive_name = os.path.expanduser(os.path.join('~',
>>> make_archive(
...     archive_name,
...     'tar',
...     root_dir='tmp/root',
...     base_dir='structure/content',
... )
'/Users/tarek/myarchive.tar'
```

Listando los archivos en el archivo resultante nos da:

```
$ python -m tarfile -l /Users/tarek/myarchive.tar
structure/content/
structure/content/please_add.txt
```

Consulta el tamaño de la terminal de salida

shutil.get_terminal_size(*fallback=(columns, lines)*)

Obtén el tamaño de la ventana de la terminal.

Para cada una de las dos dimensiones, se comprueba la variable de entorno `COLUMNS` y `LINES` respectivamente. Si la variable está definida y el valor es un entero positivo, se utiliza.

Cuando `COLUMNS` o `LINES` no está definido, como suele ocurrir, la terminal conectada a `sys.__stdout__` se consulta invocando `os.get_terminal_size()`.

Si el tamaño de la terminal no se puede consultar correctamente, ya sea porque el sistema no admite consultas o porque no estamos conectados a una terminal, se usa el valor dado en el parámetro `fallback`. `fallback` tiene por defecto `(80, 24)`, que es el tamaño por defecto que se usa en muchos emuladores de terminal.

El valor retornado es una tupla con su respectivo nombre, de tipo `os.terminal_size`.

Ver también: *The Single UNIX Specification*, Versión 2, Other Environment Variables [https://pubs.opengroup.org/onlinepubs/7908799/xbd/envvar.html#tag_002_003].

Added in version 3.3.

Distinto en la versión 3.11: Los valores `fallback` son usados también si `os.get_terminal_size()` devuelve zeros.

Persistencia de datos

Los módulos descritos en este capítulo soportan el almacenamiento de datos de Python de forma persistente en el disco. Los módulos `pickle` y `marshal` pueden convertir muchos tipos de datos de Python en un flujo de bytes y luego recrear los objetos a partir de los bytes. Los diversos módulos relacionados con DBM admiten una familia de formatos de archivo basados en hash que almacenan un mapeo de cadenas a otras cadenas.

La lista de módulos descritos en este capítulo es:

- `pickle` — Python object serialization
  - Relación con otros módulos de Python
    - Comparación con `marshal`
    - Comparación con `json`
  - Formato de flujo de datos
  - Interfaz del módulo
  - ¿Qué se puede serializar (pickled) y deserializar (unpickled) con *pickle*?
  - *Pickling* de Instancias de clases
    - Persistencia de objetos externos
    - Tablas de despacho
    - Manejo de objetos con estado
  - Reducción personalizada para tipos, funciones y otros objetos
  - Búferes fuera de banda
    - API de proveedor
    - API de consumidor
    - Ejemplo

  - Restricción de globals
  - Performance
  - Ejemplos
  - Command-line interface

- **`copyreg`** — Register **`pickle`** support functions

  - Ejemplo

- **`shelve`** — Python object persistence

  - Restricciones
  - Ejemplo

- **`marshal`** — Internal Python object serialization
- **`dbm`** — Interfaces to Unix «databases»

  - **`dbm.sqlite3`** — SQLite backend for dbm
  - **`dbm.gnu`** — GNU database manager
  - **`dbm.ndbm`** — New Database Manager
  - **`dbm.dumb`** — Portable DBM implementation

- **`sqlite3`** — DB-API 2.0 interfaz para bases de datos SQLite

  - Tutorial
  - Referencia

    - Funciones del módulo
    - Constantes del módulo
    - Objetos de conexión
    - Objetos cursor
    - Objetos fila (*Row*)
    - Objetos fila (*Row*)
    - Objetos PrepareProtocol
    - Excepciones
    - SQLite y tipos de Python
    - Adaptadores y convertidores predeterminados (en desuso)
    - Interfaz de línea de comandos

  - Guías prácticas

    - Cómo usar marcadores de posición para vincular valores en consultas SQL
    - Cómo adaptar tipos de Python personalizados a valores de SQLite
        - Cómo escribir objetos adaptables
        - Como registrar un adaptador invocable
    - Como convertir valores SQLite a tipos de Python personalizados
    - Ejemplos para adaptadores y convertidores
    - Cómo utilizar los métodos de acceso directo de conexión
    - Como usar la conexión con un administrador de contexto
    - Como trabajar con URIs SQLite
    - Cómo crear y utilizar fábricas de filas
    - Cómo manejar codificaciones de texto que no sean UTF-8
- Explicación
    - Control transaccional
        - Control de transacciones mediante el atributo **`autocommit`**
        - Control de transacciones mediante el atributo **`isolation_level`**

[1] [2] [3] [4] [5]

`pickle` — Python object serialization

Código fuente: Lib/pickle.py [https://github.com/python/cpython/tree/3.14/Lib/pickle.py]

The **`pickle`** module implements binary protocols for serializing and de-serializing a Python object structure. *«Pickling»* is the process whereby a Python object hierarchy is converted into a byte stream, and *«unpickling»* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as «serialization», «marshalling,» [1] or «flattening»; however, to avoid confusion, the terms used here are «pickling» and «unpickling».

Advertencia

El modulo `pickle` **no es seguro**. Solo deserialize con *pickle* los datos en los que confía.

Es posible construir datos maliciosos con *pickle* que **ejecuten código arbitrario durante el proceso de `unpickling`**. Nunca deserialize datos con *pickle* que podrían haber venido de una fuente no confiable, o que podrían haber sido manipulados.

Considere firmar los datos con **`hmac`** si necesita asegurarse de que no hayan sido alterados.

Los formatos de serialización más seguros como **`json`** pueden ser más apropiados si está procesando datos no confiables. Ver Comparación con json.

Relación con otros módulos de Python

Comparación con `marshal`

Python has a more primitive serialization module called `marshal`, but in general `pickle` should always be the preferred way to serialize Python objects. `marshal` exists primarily to support Python's `.pyc` files.

The `pickle` module differs from `marshal` in several significant ways:

- `marshal` cannot be used to serialize user-defined classes and their instances. `pickle` can save and restore class instances transparently, however the class definition must be importable and live in the same module as when the object was stored.
- The `marshal` serialization format is not guaranteed to be portable across Python versions. Because its primary job in life is to support `.pyc` files, the Python implementers reserve the right to change the serialization format in non-backwards compatible ways should the need arise. The `pickle` serialization format is guaranteed to be backwards compatible across Python releases provided a compatible pickle protocol is chosen and pickling and unpickling code deals with Python 2 to Python 3 type differences if your data is crossing that unique breaking change language boundary.

Comparación con `json`

Existen diferencias fundamentales entre los protocolos pickle y JSON (JavaScript Object Notation) [https://json.org]:

- JSON es un formato de serialización de texto (genera texto unicode, aunque la mayoría de las veces se codifica a `utf-8`), mientras que *pickle* es un formato de serialización binario;
- JSON es legible por humanos, mientras que *pickle* no lo es;
- JSON es interoperable y ampliamente utilizado fuera del ecosistema de Python, mientras que *pickle* es específico de Python;
- JSON, por defecto, solo puede representar un subconjunto de

los tipos integrados de Python, y no clases personalizadas; *pickle* puede representar un número extremadamente grande de tipos de Python (muchos de ellos automáticamente, mediante el uso inteligente de la introspección de objetos en Python; los casos complejos se pueden abordar implementando API de objetos específicos, specific object APIs);

- A diferencia de *pickle*, deserializar JSON no confiable no crea en sí mismo una vulnerabilidad de ejecución de código arbitraria.

Ver también

El modulo `json`: un módulo de la biblioteca estándar que permite la serialización y deserialización de JSON.

Formato de flujo de datos

The data format used by `pickle` is Python-specific. This has the advantage that there are no restrictions imposed by external standards such as JSON (which can't represent pointer sharing); however it means that non-Python programs may not be able to reconstruct pickled Python objects.

By default, the `pickle` data format uses a relatively compact binary representation. If you need optimal size characteristics, you can efficiently compress pickled data.

The module `pickletools` contains tools for analyzing data streams generated by `pickle`. `pickletools` source code has extensive comments about opcodes used by pickle protocols.

Actualmente hay 6 protocolos diferentes que se pueden utilizar para serializar con *pickle*. Cuanto mayor sea el protocolo utilizado, más reciente será la versión de Python necesaria para leer el *pickle* producido.

- La versión 0 del protocolo es el protocolo original «legible para humanos» y es compatible con versiones anteriores de

Python.

- La versión 1 del protocolo es un formato binario antiguo que también es compatible con versiones anteriores de Python.
- La versión 2 del protocolo se introdujo en Python 2.3. Proporciona un serializado con pickle mucho más eficiente de new-style classes. Consulte **PEP 307** [https://peps.python.org/pep-0307/] para obtener información sobre las mejoras que trae el protocolo 2.
- Se agregó la versión 3 del protocolo en Python 3.0. Tiene soporte explícito para objetos `bytes` y no puede ser deserializado con *pickle* por Python 2.x. Este era el protocolo predeterminado en Python 3.0–3.7.
- Protocol version 4 was added in Python 3.4. It adds support for very large objects, pickling more kinds of objects, and some data format optimizations. This was the default protocol in Python 3.8–3.13. Refer to **PEP 3154** [https://peps.python.org/pep-3154/] for information about improvements brought by protocol 4.
- Protocol version 5 was added in Python 3.8. It adds support for out-of-band data and speedup for in-band data. It is the default protocol starting with Python 3.14. Refer to **PEP 574** [https://peps.python.org/pep-0574/] for information about improvements brought by protocol 5.

Nota

Serialization is a more primitive notion than persistence; although `pickle` reads and writes file objects, it does not handle the issue of naming persistent objects, nor the (even more complicated) issue of concurrent access to persistent objects. The `pickle` module can transform a complex object into a byte stream and it can transform the byte stream into an object with the same internal structure. Perhaps the most obvious thing to do with these byte streams is to write them onto a file, but it is also conceivable to send them across a network or store them in a database. The `shelve` module provides a simple interface to pickle and unpickle objects on DBM-style database files.

Interfaz del módulo

Para serializar una jerarquía de objetos, simplemente llame a la función `dumps()`. De manera similar, para deserializar un flujo de datos, llama a la función `loads()`. Sin embargo, si desea tener más control sobre la serialización y la deserialización, puede crear un objeto `Pickler` o `Unpickler`, respectivamente.

The `pickle` module provides the following constants:

pickle.HIGHEST_PROTOCOL

> Un entero, la versión de protocolo (protocol version) más alta disponible. Este valor se puede pasar como un valor de *protocolo* a las funciones `dump()` y `dumps()` así como al constructor `Pickler`.

pickle.DEFAULT_PROTOCOL

> An integer, the default protocol version used for pickling. May be less than `HIGHEST_PROTOCOL`. Currently the default protocol is 5, introduced in Python 3.8 and incompatible with previous versions. This version introduces support for out-of-band buffers, where **PEP 3118** [https://peps.python.org/pep-3118/]-compatible data can be transmitted separately from the main pickle stream.
>
> *Distinto en la versión 3.0:* El protocolo predeterminado es 3.
>
> *Distinto en la versión 3.8:* El protocolo predeterminado es 4.
>
> *Distinto en la versión 3.14:* The default protocol is 5.

The `pickle` module provides the following functions to make the pickling process more convenient:

pickle.dump(*obj*, *file*, *protocol=None*, ***, *fix_imports=True*, *buffer_callback=None*)

> Escribe la representación *pickle* del objeto *obj* en el *archivo* abierto file object. Esto es equivalente a `Pickler(file, protocol).dump(obj)`.
>
> Los argumentos *file*, *protocol*, *fix_imports* y *buffer_callback* tienen el mismo significado que en el constructor `Pickler`.

Distinto en la versión 3.8: Se agregó el argumento *buffer_callback*.

pickle.dumps(*obj*, *protocol=None*, ***, *fix_imports=True*, *buffer_callback=None*)

Retorna la representación *pickle* del objeto *obj* como un objeto **`bytes`**, en lugar de escribirlo en un archivo.

Los argumentos *protocol*, *fix_imports* y *buffer_callback* tienen el mismo significado que en el constructor **`Pickler`**.

Distinto en la versión 3.8: Se agregó el argumento *buffer_callback*.

pickle.load(*file*, ***, *fix_imports=True*, *encoding='ASCII'*, *errors='strict'*, *buffers=None*)

Lee la representación *pickle* de un objeto desde un *archivo* abierto file object y retorna la jerarquía de objetos reconstituidos especificada en el mismo. Esto es equivalente a `Unpickler(file).load()`.

La versión de protocolo del *pickle* se detecta automáticamente, por lo que no se necesita ningún argumento de protocolo. Los bytes más allá de la representación empaquetada son ignorados.

Los argumentos *file*, *fix_imports*, *encoding*, *errors*, *strict* y *buffers* tienen el mismo significado que en el constructor **`Unpickler`**.

Distinto en la versión 3.8: Se agregó el argumento *buffers*.

pickle.loads(*data*, */*, ***, *fix_imports=True*, *encoding='ASCII'*, *errors='strict'*, *buffers=None*)

Retorna la jerarquía de objetos reconstruida de la representación *pickle data* de un objeto. *data* debe ser un objeto tipo binario (bytes-like object).

La versión de protocolo del *pickle* se detecta automáticamente, por lo que no se necesita ningún argumento de protocolo. Los bytes más allá de la representación empaquetada son ignorados.

Los argumentos *fix_imports*, *encoding*, *errors*, *strict* y *buffers* tienen el mismo significado que en el constructor `Unpickler`.

Distinto en la versión 3.8: Se agregó el argumento *buffers*.

The **`pickle`** module defines three exceptions:

exception pickle.PickleError

Common base class for the other pickling exceptions. It inherits from `Exception`.

exception pickle.PicklingError

Error raised when an unpicklable object is encountered by `Pickler`. It inherits from `PickleError`.

Consulte ¿Qué se puede serializar (pickled) y deserializar (unpickled) con pickle? para aprender qué tipos de objetos se pueden serializar con *pickle*.

exception pickle.UnpicklingError

Error raised when there is a problem unpickling an object, such as a data corruption or a security violation. It inherits from `PickleError`.

Tenga en cuenta que también se pueden generar otras excepciones durante la deserializacion con *pickle*, incluyendo (pero no necesariamente limitado a) *AttributeError*, *EOFError*, *ImportError*, e *IndexError*.

The **`pickle`** module exports three classes, `Pickler`, `Unpickler` and `PickleBuffer`:

class pickle.Pickler(*file*, *protocol=None*, ***, *fix_imports=True*, *buffer_callback=None*)

Esto toma un archivo binario para escribir un flujo de datos de *pickle*.

El argumento opcional *protocol* , un entero, le dice al *pickler* que use el protocolo dado; los protocolos admitidos son 0 para `HIGHEST_PROTOCOL`. Si no se especifica, el valor predeterminado es `DEFAULT_PROTOCOL`. Si se especifica un número negativo, `HIGHEST_PROTOCOL` es seleccionado.

El argumento *file* debe tener un método *write()* que acepte un argumento de bytes individuales. Por lo tanto, puede ser un archivo en disco abierto para escritura binaria, una instancia `io.BytesIO` , o cualquier otro objeto personalizado que cumpla con esta interfaz.

Si *fix_imports* es verdadero y *protocol* es menor que 3, *pickle* intentará asignar los nuevos nombres de Python 3 a los nombres de módulos antiguos utilizados en Python 2, de modo que la secuencia de datos de *pickle* sea legible con Python 2.

If *buffer_callback* is `None` (the default), buffer views are serialized into *file* as part of the pickle stream.

If *buffer_callback* is not `None`, then it can be called any number of times with a buffer view. If the callback returns a false value (such as `None`), the given buffer is out-of-band; otherwise the buffer is serialized in-band, i.e. inside the pickle stream.

It is an error if *buffer_callback* is not `None` and *protocol* is `None` or smaller than 5.

Distinto en la versión 3.8: Se agregó el argumento *buffer_callback*.

dump(*obj*)

Escribe la representación serializada con *pickle* del objeto *obj* en el objeto archivo abierto dado en el constructor.

persistent_id(*obj*)

No hacer nada por defecto. Esto existe para que una subclase pueda sobreescribirlo.

Si `persistent_id()` retorna `None`, *obj* es serializado con *pickle* como siempre. Cualquier otro valor hace que `Pickler` emita el valor retornado como un ID persistente para *obj*. El significado de este ID persistente debe definirse por `Unpickler.persistent_load()`. Tenga en cuenta que el valor retornado por **`persistent_id()`** no puede tener una ID persistente.

Ver Persistencia de objetos externos para detalles y ejemplos de uso.

Distinto en la versión 3.13: Add the default implementation of this method in the C implementation of **`Pickler`**.

dispatch_table

A pickler object's dispatch table is a registry of *reduction functions* of the kind which can be declared using `copyreg.pickle()`. It is a mapping whose keys are classes and whose values are reduction functions. A reduction function takes a single argument of the associated class and should conform to the same interface as a `__reduce__()` method.

Por defecto, un objeto de *pickle* no tendrá un atributo `dispatch_table`, y en su lugar utilizará la tabla de despacho global administrada por el módulo `copyreg`. Sin embargo, para personalizar el *pickling* para un objeto de *pickle* específico, se puede establecer el atributo **`dispatch_table`** en un objeto tipo dict. Alternativamente, si una subclase de `Pickler` tiene un atributo **`dispatch_table`** esto se usará como la tabla de despacho predeterminada para instancias de esa clase.

Ver Tablas de despacho para ejemplos de uso.

Added in version 3.3.

reducer_override(*obj*)

Special reducer that can be defined in **`Pickler`** subclasses. This method has priority over any reducer in the **`dispatch_table`**. It should conform to the same interface as a **`__reduce__()`** method, and can optionally return **`NotImplemented`** to fallback on **`dispatch_table`**-registered reducers to pickle `obj`.

Para un ejemplo detallado, ver Reducción personalizada para tipos, funciones y otros objetos.

Added in version 3.8.

fast

Obsoleto. Habilite el modo rápido si se establece en un valor verdadero. El modo rápido deshabilita el uso de memo, por lo tanto, acelera el proceso de *pickling* al no generar códigos de operación PUT superfluos. No debe usarse con objetos autorreferenciales; de lo contrario, la clase **`Pickler`** se repetirá infinitamente.

Use **`pickletools.optimize()`** si necesita *pickles* más compactos.

clear_memo()

Clears the pickler's «memo».

The memo is the data structure that remembers which objects the pickler has already seen, so that shared or recursive objects are pickled by reference and not by value. This method is useful when re-using picklers.

class pickle.Unpickler(*file*, ***, *fix_imports = True*, *encoding = 'ASCII'*, *errors = 'strict'*, *buffers = None*)

Esto toma un archivo binario para leer un flujo de datos de *pickle*.

La versión de protocolo de *pickle* se detecta automáticamente, por lo que no se necesita ningún argumento de protocolo.

El argumento *file* debe tener tres métodos, un método read() que toma un argumento entero, un método *readinto()* que toma un argumento búfer y un método *readline()* que no requiere argumentos, como en la interfaz **`io.BufferedIOBase`**. Por lo tanto *file* puede ser un archivo en disco abierto para lectura binaria, un objeto **`io.BytesIO`**, o cualquier otro objeto personalizado que cumpla con esta interfaz.

Los argumentos opcionales *fix_imports*, *encoding* and *errors* se utilizan para controlar el soporte de compatibilidad para el flujo de *pickle* generado por Python 2. Si *fix_imports* es verdadero, *pickle* intentará asignar los nombres antiguos de Python 2 a los nuevos nombres utilizados en Python 3. Tanto *encoding* como *errors* le indican a *pickle* cómo decodificar instancias de cadenas de 8 bits seleccionadas por Python 2; estos son predeterminados a “ASCII” y “strict”, respectivamente. *encoding* puede ser “bytes” para leer estas instancias de cadena de 8 bits como objetos de bytes. Se requiere el uso de `encoding='latin1'` para realizar el *unpickling* de arreglos de NumPy e instancias de **`datetime`**, **`date`** y **`time`** serializados con *pickle* por Python 2.

If *buffers* is `None` (the default), then all data necessary for deserialization must be contained in the pickle stream. This means that the *buffer_callback* argument was `None` when a **`Pickler`** was instantiated (or when **`dump()`** or **`dumps()`** was called).

If *buffers* is not `None`, it should be an iterable of buffer-enabled objects that is consumed each time the pickle stream references an out-of-band buffer view. Such buffers have been given in order to the *buffer_callback* of a Pickler object.

Distinto en la versión 3.8: Se agregó el argumento *buffers*.

load()

Lee la representación serializada con *pickle* de un objeto desde el objeto de archivo abierto dado en el constructor, y retorne la jerarquía de objetos reconstituidos especificada allí. Los Bytes más allá de la representación serializada con *pickle* del objeto se ignoran.

persistent_load(*pid*)

Lanza un **`UnpicklingError`** de forma predeterminada.

Si se define, **`persistent_load()`** debería retornar el objeto especificado por el ID persistente *pid*. Si se encuentra un ID persistente no válido, se debe lanzar un **`UnpicklingError`**.

Ver Persistencia de objetos externos para detalles y ejemplos de uso.

Distinto en la versión 3.13: Add the default implementation of this method in the C implementation of **`Unpickler`**.

find_class(*module*, *name*)

Importa *module* si es necesario y retorna el objeto llamado *name* desde el, donde los argumentos *module* y *name* son objetos de **`str`**. Tenga en cuenta que, a diferencia de lo que sugiere su nombre, **`find_class()`** también se usa para buscar funciones.

Las subclases pueden sobreescribir esto para obtener control sobre qué tipo de objetos y cómo se pueden cargar, reduciendo potencialmente los riesgos de seguridad. Consulte Restricción de globals para obtener más detalles.

Lanza un auditing event `pickle.find_class` con argumentos `module`, `name`.

class pickle.PickleBuffer(*buffer*)

Un envoltorio (*wrapper*) para un búfer que representa datos serializables con *pickle* (*picklable data*). *buffer* debe ser un objeto que proporciona un búfer (buffer-providing), como objeto tipo binario (bytes-like object) o un arreglo N-dimensional.

`PickleBuffer` es en sí mismo un proveedor de búfer, por lo que es posible pasarlo a otras API que esperan un objeto que provea un búfer, como **`memoryview`**.

Los objetos **`PickleBuffer`** solo se pueden serializar usando el protocolo *pickle* 5 o superior. Son elegibles para serialización fuera de banda (out-of-band serialization).

Added in version 3.8.

raw()

Retorna un **`memoryview`** del área de memoria subyacente a este búfer. El objeto retornado es una vista de memoria unidimensional, C-contigua con formato `B` (bytes sin firmar). **`BufferError`** es lanzado si el búfer no es contiguo a C ni a Fortran.

release()

Libera el búfer subyacente expuesto por el objeto PickleBuffer.

¿Qué se puede serializar (pickled) y deserializar (unpickled) con *pickle*?

Los siguientes tipos se pueden serializar con *pickle* (pickled):

- built-in constants (`None`, `True`, `False`, `Ellipsis`, and **`NotImplemented`**);
- enteros, números de coma flotante, números complejos;
- cadenas de caracteres, bytes, bytearrays;
- tuplas, listas, conjuntos y diccionarios que contienen solo

objetos serializables con pickle;
- funciones (incorporadas y definidas por el usuario) accesibles desde el nivel superior de un módulo (usando **`def`**, no **`lambda`**)
- clases accesibles desde el nivel superior de un módulo;
- instances of such classes for which the result of calling **`__getstate__()`** is picklable (see section Pickling de Instancias de clases for details).

Los intentos de serializar objetos no serializables con *pickle* lanzaran la excepción **`PicklingError`**; cuando esto sucede, es posible que ya se haya escrito una cantidad no especificada de bytes en el archivo subyacente. Intentar serializar con *pickle* una estructura de datos altamente recursiva puede exceder la profundidad máxima de recursividad, en este caso se lanzará un **`RecursionError`**. Puede aumentar cuidadosamente este límite con **`sys.setrecursionlimit()`**.

Tenga en cuenta que las funciones (integradas y definidas por el usuario) están completamente serializadas con *pickle* por qualified name, no por valor. [2] Esto significa que solo se serializa el nombre de la función, junto con el nombre del módulo y las clases que lo contienen. No se serializa ni el código de la función ni ninguno de sus atributos de función. Por lo tanto, el módulo de definición debe poder importarse en el entorno de deserialización y el módulo debe contener el objeto nombrado; de lo contrario, se generará una excepción. [3]

De manera similar, las clases se serializan por nombre completo, por lo que se aplican las mismas restricciones en el entorno de deserialización. Tenga en cuenta que ninguno de los códigos o datos de la clase se serializa, por lo que en el siguiente ejemplo, el atributo de clase `attr` no se restaura en el entorno de deserializado:

```
class Foo:
    attr = 'A class attribute'

picklestring = pickle.dumps(Foo)
```

Estas restricciones son la razón por la que las funciones y clases serializables con pickle deben definirse en el nivel superior de un módulo.

Similarly, when class instances are pickled, their class’s code and data are not pickled along with them. Only the instance data are pickled. This is done on purpose, so you can fix bugs in a class or add methods to the class and still load objects that were created with an earlier version of the class. If you plan to have long-lived objects that will see many versions of a class, it may be worthwhile to put a version number in the objects so that suitable conversions can be made by the class’s `__setstate__()` method.

Pickling de Instancias de clases

En esta sección, describimos los mecanismos generales disponibles para que usted defina, personalice y controle cómo se serializan y deserializan con *Pickle* las instancias de clase.

In most cases, no additional code is needed to make instances picklable. By default, pickle will retrieve the class and the attributes of an instance via introspection. When a class instance is unpickled, its `__init__()` method is usually *not* invoked. The default behaviour first creates an uninitialized instance and then restores the saved attributes. The following code shows an implementation of this behaviour:

```
def save(obj):
    return (obj.__class__, obj.__dict__)

def restore(cls, attributes):
    obj = cls.__new__(cls)
    obj.__dict__.update(attributes)
    return obj
```

Las clases pueden alterar el comportamiento predeterminado proporcionando uno o varios métodos especiales:

object.__getnewargs_ex__()

In protocols 2 and newer, classes that implement the

`__getnewargs_ex__()` method can dictate the values passed to the **`__new__()`** method upon unpickling. The method must return a pair `(args, kwargs)` where *args* is a tuple of positional arguments and *kwargs* a dictionary of named arguments for constructing the object. Those will be passed to the **`__new__()`** method upon unpickling.

Debes implementar este método si el método **`__new__()`** de tu clase requiere argumentos de solo palabras clave. De lo contrario, se recomienda para la compatibilidad implementar **`__getnewargs__()`**.

Distinto en la versión 3.6: **`__getnewargs_ex__()`** ahora se usa en los protocolos 2 y 3.

object.__getnewargs__()

Este método tiene un propósito similar a **`__getnewargs_ex__()`**, pero solo admite argumentos posicionales. Debe retornar una tupla de argumentos `args` que se pasarán al método **`__new__()`** al hacer *unpickling*.

`__getnewargs__()` no se llamará si **`__getnewargs_ex__()`** está definido.

Distinto en la versión 3.6: Antes de Python 3.6, se llamaba a, **`__getnewargs__()`** en lugar de **`__getnewargs_ex__()`** en los protocolos 2 y 3.

object.__getstate__()

Las clases pueden influir aún más en cómo se serializan con *pickle* sus instancias sobrescribiendo el método **`__getstate__()`**. Se llama y el objeto devuelto se conserva como el contenido de la instancia, en lugar de un estado predeterminado. Hay varios casos:

- Para una clase que no tiene instancias **`__dict__`** ni **`__slots__`**, el estado predeterminado es `None`.
- Para una clase que tiene una instancia **`__dict__`** y no tiene **`__slots__`**, el estado predeterminado es `self.__dict__`.

- Para una clase que tiene una instancia **`__dict__`** y **`__slots__`**, el estado predeterminado es una tupla que consta de dos diccionarios: `self.__dict__` y un diccionario que asigna nombres de ranura a valores de ranura. Solo las ranuras que tienen un valor se incluyen en este último.
- Para una clase que tiene **`__slots__`** y ninguna instancia **`__dict__`**, el estado predeterminado es una tupla cuyo primer elemento es `None` y cuyo segundo elemento es un diccionario que asigna nombres de ranura a valores de ranura descritos en la viñeta anterior.

Distinto en la versión 3.11: Se agregó la implementación predeterminada del método `__getstate__()` en la clase **`object`**.

object.__setstate__(*state*)

Al hacer *unpickling*, si la clase define **`__setstate__()`**, este es llamado con el estado *unpickled* (no serializado con *pickle*). En ese caso, no es necesario que el objeto de estado sea un diccionario. De lo contrario, el estado *pickled* (*pickled state*) debe ser un diccionario y sus elementos se asignan al diccionario de la nueva instancia.

Nota

If **`__reduce__()`** returns a state with value `None` at pickling, the **`__setstate__()`** method will not be called upon unpickling.

Refer to the section Manejo de objetos con estado for more information about how to use the methods **`__getstate__()`** and **`__setstate__()`**.

Nota

At unpickling time, some methods like **`__getattr__()`**, `__getattribute__()`, or `__setattr__()` may be called upon the instance. In case those methods rely on some internal invariant being true, the type should implement `__new__()` to establish such an invariant, as `__init__()` is not called when unpickling an instance.

As we shall see, pickle does not use directly the methods described above. In fact, these methods are part of the copy protocol which implements the `__reduce__()` special method. The copy protocol provides a unified interface for retrieving the data necessary for pickling and copying objects. [4]

Although powerful, implementing `__reduce__()` directly in your classes is error prone. For this reason, class designers should use the high-level interface (i.e., `__getnewargs_ex__()`, `__getstate__()` and `__setstate__()`) whenever possible. We will show, however, cases where using `__reduce__()` is the only option or leads to more efficient pickling or both.

object.__reduce__()

La interfaz se define actualmente de la siguiente manera. El método `__reduce__()` no toma ningún argumento y retornará una cadena o preferiblemente una tupla (el objeto retornado a menudo se denomina «valor reducido»).

Si se retorna una cadena, la cadena debe interpretarse como el nombre de una variable global. Debe ser el nombre local del objeto relativo a su módulo; el módulo *pickle* busca en el espacio de nombres del módulo para determinar el módulo del objeto. Este comportamiento suele ser útil para singletons.

Cuando se retorna una tupla, debe tener entre dos y seis elementos. Los elementos opcionales se pueden omitir o se puede proporcionar `None` como su valor. La semántica de cada elemento está en orden:

- Un objeto invocable que se llamará para crear la versión inicial del objeto.
- Una tupla de argumentos para el objeto invocable. Se

debe proporcionar una tupla vacía si el invocable no acepta ningún argumento.

- Opcionalmente, el estado del objeto, que se pasará al método **`__setstate__()`** del objeto como se describió anteriormente. Si el objeto no tiene dicho método, el valor debe ser un diccionario y se agregará al atributo **`__dict__`** del objeto.

- Optionally, an iterator (and not a sequence) yielding successive items. These items will be appended to the object either using `obj.append(item)` or, in batch, using `obj.extend(list_of_items)`. This is primarily used for list subclasses, but may be used by other classes as long as they have **`append()`** and **`extend()`** methods with the appropriate signature. (Whether **`append()`** or **`extend()`** is used depends on which pickle protocol version is used as well as the number of items to append, so both must be supported.)

- Opcionalmente, un iterador (no una secuencia) que produce pares clave-valor sucesivos. Estos elementos se almacenarán en el objeto usando `obj[key] = value`. Esto se usa principalmente para subclases de diccionario, pero otras clases pueden usarlo siempre que implementen **`__setitem__()`**.

- Opcionalmente, un invocable con una firma `(obj, state)`. Este invocable permite al usuario controlar programáticamente el comportamiento de actualización de estado de un objeto específico, en lugar de usar el método estático de `obj` **`__setstate__()`**. Si no es `None`, este invocable tendrá prioridad sobre `obj`’s **`__setstate__()`**.

  Added in version 3.8: Se agregó el sexto elemento opcional de tupla `(obj, state)`.

object.__reduce_ex__(*protocol*)

Alternativamente, se puede definir un método `__reduce_ex__()`. La única diferencia es que este método debe tomar un único argumento entero, la versión del protocolo. Cuando esté definido, *pickle* lo preferirá en lugar del método `__reduce__()`. Además, `__reduce__()` se convierte automáticamente en sinónimo de la versión extendida. El uso principal de este método es proporcionar valores reducidos compatibles con versiones anteriores para versiones anteriores de Python.

Persistencia de objetos externos

For the benefit of object persistence, the `pickle` module supports the notion of a reference to an object outside the pickled data stream. Such objects are referenced by a persistent ID, which should be either a string of alphanumeric characters (for protocol 0) [5] or just an arbitrary object (for any newer protocol).

The resolution of such persistent IDs is not defined by the `pickle` module; it will delegate this resolution to the user-defined methods on the pickler and unpickler, `persistent_id()` and `persistent_load()` respectively.

Para seleccionar objetos que tienen una ID persistente externo, el *pickler* debe tener un método personalizado `persistent_id()` que toma un objeto como argumento y retorna `None` o el ID persistente para ese objeto. Cuando se retorna `None`, el *pickler* simplemente serializará el objeto de forma normal. Cuando se retorna una cadena de identificación persistente, el *pickler* serializará ese objeto, junto con un marcador para que el *unpickler* lo reconozca como una identificación persistente.

Para hacer el *unpickling* objetos externos, el *unpickler* debe tener un método personalizado `persistent_load()` que toma un objeto de identificación persistente y retorna el objeto referenciado.

Aquí hay un ejemplo completo que presenta cómo se puede usar la identificación persistente para hacer el *pickling* objetos externos por referencia.

```
# Simple example presenting how persistent ID can be use
# external objects by reference.

import pickle
import sqlite3
from collections import namedtuple

# Simple class representing a record in our database.
MemoRecord = namedtuple("MemoRecord", "key, task")

class DBPickler(pickle.Pickler):

    def persistent_id(self, obj):
        # Instead of pickling MemoRecord as a regular cl
        # persistent ID.
        if isinstance(obj, MemoRecord):
            # Here, our persistent ID is simply a tuple,
            # key, which refers to a specific record in
            return ("MemoRecord", obj.key)
        else:
            # If obj does not have a persistent ID, retu
            # needs to be pickled as usual.
            return None


class DBUnpickler(pickle.Unpickler):

    def __init__(self, file, connection):
        super().__init__(file)
        self.connection = connection

    def persistent_load(self, pid):
        # This method is invoked whenever a persistent I
        # Here, pid is the tuple returned by DBPickler.
        cursor = self.connection.cursor()
        type_tag, key_id = pid
        if type_tag == "MemoRecord":
            # Fetch the referenced record from the datab
            cursor.execute("SELECT * FROM memos WHERE ke
            key, task = cursor.fetchone()
            return MemoRecord(key, task)
        else:
            # Always raises an error if you cannot retur
            # Otherwise, the unpickler will think None i
            # by the persistent ID.
            raise pickle.UnpicklingError("unsupported pe


def main():
    import io
    import pprint

    # Initialize and populate our database.
    conn = sqlite3.connect(":memory:")
    cursor = conn.cursor()
    cursor.execute("CREATE TABLE memos(key INTEGER PRIMA
    tasks = (
        'give food to fish',
        'prepare group meeting',
        'fight with a zebra',
        )
    for task in tasks:
        cursor.execute("INSERT INTO memos VALUES(NULL, ?

    # Fetch the records to be pickled.
    cursor.execute("SELECT * FROM memos")
    memos = [MemoRecord(key, task) for key, task in curs
    # Save the records using our custom DBPickler.
    file = io.BytesIO()
    DBPickler(file).dump(memos)

    print("Pickled records:")
    pprint.pprint(memos)

    # Update a record, just for good measure.
    cursor.execute("UPDATE memos SET task='learn italian
    # Load the records from the pickle data stream.
    file.seek(0)
    memos = DBUnpickler(file, conn).load()

    print("Unpickled records:")
    pprint.pprint(memos)


if __name__ == '__main__':
    main()
```

Tablas de despacho

Si se desea personalizar el *pickling* de algunas clases sin alterar ningún otro código que dependa del *pickling*, se puede crear un *pickler* con una tabla de despacho privada.

The global dispatch table managed by the **`copyreg`** module is available as **`copyreg.dispatch_table`**. Therefore, one may choose to use a modified copy of **`copyreg.dispatch_table`** as a private dispatch table.

Por ejemplo

```
f = io.BytesIO()
p = pickle.Pickler(f)
p.dispatch_table = copyreg.dispatch_table.copy()
p.dispatch_table[SomeClass] = reduce_SomeClass
```

crea una instancia de **`pickle.Pickler`** con una tabla de despacho privada que maneja la clase `AlgunaClase` especialmente. Alternativamente, el código

```
class MyPickler(pickle.Pickler):
    dispatch_table = copyreg.dispatch_table.copy()
    dispatch_table[SomeClass] = reduce_SomeClass
f = io.BytesIO()
p = MyPickler(f)
```

hace lo mismo, pero todas las instancias de `MyPickler` compartirán de forma predeterminada la tabla de despacho privada. Por otro lado, el código

```
copyreg.pickle(SomeClass, reduce_SomeClass)
f = io.BytesIO()
p = pickle.Pickler(f)
```

modifica la tabla de despacho global compartida por todos los usuarios del módulo **`copyreg`**.

Manejo de objetos con estado

Here’s an example that shows how to modify pickling behavior for a class. The **`TextReader`** class below opens a text file, and returns the line number and line contents each time its **`readline()`** method is called. If a **`TextReader`** instance is pickled, all attributes *except* the file object member are saved. When the instance is unpickled, the file is reopened, and reading resumes from the last location. The **`__setstate__()`** and **`__getstate__()`** methods are used to implement this behavior.

```
class TextReader:
    """Print and number lines in a text file."""

    def __init__(self, filename):
        self.filename = filename
        self.file = open(filename)
        self.lineno = 0

    def readline(self):
        self.lineno += 1
        line = self.file.readline()
        if not line:
            return None
        if line.endswith('\n'):
            line = line[:-1]
        return "%i: %s" % (self.lineno, line)
    def __getstate__(self):
        # Copy the object's state from self.__dict__ whi
        # all our instance attributes. Always use the di
        # method to avoid modifying the original state.
        state = self.__dict__.copy()
        # Remove the unpicklable entries.
        del state['file']
        return state

    def __setstate__(self, state):
        # Restore instance attributes (i.e., filename ar
        self.__dict__.update(state)
        # Restore the previously opened file's state. To
        # reopen it and read from it until the line cour
        file = open(self.filename)
        for _ in range(self.lineno):
            file.readline()
        # Finally, save the file.
        self.file = file
```

Un ejemplo de uso podría ser algo como esto:

```
>>> reader = TextReader("hello.txt")
>>> reader.readline()
'1: Hello world!'
>>> reader.readline()
'2: I am line number two.'
>>> new_reader = pickle.loads(pickle.dumps(reader))
>>> new_reader.readline()
'3: Goodbye!'
```

Reducción personalizada para tipos, funciones y otros objetos

Added in version 3.8.

A veces, `dispatch_table` puede no ser lo suficientemente flexible. En particular, es posible que deseemos personalizar el *pickling* en función de otro criterio que no sea el tipo de objeto, o es posible que deseemos personalizar el *pickling* de funciones y clases.

For those cases, it is possible to subclass from the `Pickler` class and implement a `reducer_override()` method. This method can return an arbitrary reduction tuple (see `__reduce__()`). It can alternatively return `NotImplemented` to fallback to the traditional behavior.

Si se definen tanto `dispatch_table` como `reducer_override()`, entonces **`reducer_override()`** tiene prioridad.

Nota

Por motivos de rendimiento, no se puede llamar a `reducer_override()` para los siguientes objetos: `None`, `True`, `False`, e instancias exactas de `int`, `float`, `bytes`, `str`, `dict`, `set`, `frozenset`, `list` y `tuple`.

Aquí hay un ejemplo simple donde permitimos el *pickling* y reconstruir una clase dada class:

```
import io
import pickle

class MyClass:
    my_attribute = 1

class MyPickler(pickle.Pickler):
    def reducer_override(self, obj):
        """Custom reducer for MyClass."""
        if getattr(obj, "__name__", None) == "MyClass":
            return type, (obj.__name__, obj.__bases__,
                          {'my_attribute': obj.my_attrib
        else:
            # For any other object, fallback to usual re
            return NotImplemented
```

```
f = io.BytesIO()
p = MyPickler(f)
p.dump(MyClass)

del MyClass

unpickled_class = pickle.loads(f.getvalue())

assert isinstance(unpickled_class, type)
assert unpickled_class.__name__ == "MyClass"
assert unpickled_class.my_attribute == 1
```

Búferes fuera de banda

Added in version 3.8.

In some contexts, the `pickle` module is used to transfer massive amounts of data. Therefore, it can be important to minimize the number of memory copies, to preserve performance and resource consumption. However, normal operation of the `pickle` module, as it transforms a graph-like structure of objects into a sequential stream of bytes, intrinsically involves copying data to and from the pickle stream.

Esta restricción puede evitarse si tanto el *proveedor* (la implementación de los tipos de objeto a transferir) como el *consumidor* (a implementación del sistema de comunicaciones) admiten las facilidades de transferencia fuera de banda proporcionadas por el protocolo *pickle* 5 y mayor.

API de proveedor

The large data objects to be pickled must implement a `__reduce_ex__()` method specialized for protocol 5 and higher, which returns a `PickleBuffer` instance (instead of e.g. a `bytes` object) for any large data.

A `PickleBuffer` object *signals* that the underlying buffer is eligible for out-of-band data transfer. Those objects remain compatible with normal usage of the `pickle` module. However, consumers can also opt-in to tell `pickle` that they will handle those buffers by themselves.

API de consumidor

Un sistema de comunicaciones puede permitir el manejo personalizado de los objetos `PickleBuffer` generados al serializar un gráfico de objetos.

En el lado del envío, necesita pasar un argumento *buffer_callback* a `Pickler` (o a las funciones `dump()` o `dumps()`), que se llamará con cada `PickleBuffer` generado al hacer *pickling* del gráfico del objeto. Los búferes acumulados por *buffer_callback* no verán sus datos copiados en el flujo de *pickle*, solo se insertará un marcador barato.

En el lado receptor, necesita pasar un argumento *buffers* a `Unpickler` (o a las funciones `load()` o `loads()`), que es un iterable de los búferes que fueron pasado a *buffer_callback*. Ese iterable debería producir búferes en el mismo orden en que se pasaron a *buffer_callback*. Esos búferes proporcionarán los datos esperados por los reconstructores de los objetos cuyo *pickling* produjo los objetos originales `PickleBuffer`.

Entre el lado de envío y el lado de recepción, el sistema de comunicaciones es libre de implementar su propio mecanismo de transferencia para memorias intermedias fuera de banda. Las posibles optimizaciones incluyen el uso de memoria compartida o compresión dependiente del tipo de datos.

Ejemplo

Aquí hay un ejemplo trivial donde implementamos una subclase `bytearray` capaz de participar en el *pickling* de un búfer fuera de banda:

```
class ZeroCopyByteArray(bytearray):

    def __reduce_ex__(self, protocol):
        if protocol >= 5:
            return type(self)._reconstruct, (PickleBuffe
        else:
            # PickleBuffer is forbidden with pickle prot
            return type(self)._reconstruct, (bytearray(s

    @classmethod
    def _reconstruct(cls, obj):
        with memoryview(obj) as m:
            # Get a handle over the original buffer obje
            obj = m.obj
            if type(obj) is cls:
                # Original buffer object is a ZeroCopyBy
                # as-is.
                return obj
            else:
                return cls(obj)
```

El reconstructor (el método de clase `_reconstruct`) retorna el objeto que proporciona el búfer si tiene el tipo correcto. Esta es una manera fácil de simular el comportamiento de copia cero en este ejemplo de juguete.

En el lado del consumidor, podemos serializar con *pickle* esos objetos de la forma habitual, que cuando no se serializan nos dará una copia del objeto original:

```
b = ZeroCopyByteArray(b"abc")
data = pickle.dumps(b, protocol=5)
new_b = pickle.loads(data)
print(b == new_b)  # True
print(b is new_b)  # False: a copy was made
```

Pero si pasamos un *buffer_callback* y luego retornamos los búferes acumulados al anular la serialización, podemos recuperar el objeto original:

```
b = ZeroCopyByteArray(b"abc")
buffers = []
data = pickle.dumps(b, protocol=5, buffer_callback=buffe
new_b = pickle.loads(data, buffers=buffers)
print(b == new_b)   # True
print(b is new_b)   # True: no copy was made
```

Este ejemplo está limitado por el hecho de que `bytearray` asigna su propia memoria: no puedes crear una instancia de `bytearray` que esté respaldada por la memoria de otro objeto. Sin embargo, los tipos de datos de terceros, como las matrices NumPy no tienen esta limitación y permiten el uso de *pickling* de copia cero (o realizar la menor cantidad de copias posible) cuando se transfieren entre procesos o sistemas distintos.

Ver también

PEP 574 [https://peps.python.org/pep-0574/] – Protocolo Pickle 5 con datos fuera de banda

Restricción de globals

De forma predeterminada, el *unpickling* importará cualquier clase o función que encuentre en los datos de *pickle*. Para muchas aplicaciones, este comportamiento es inaceptable, ya que permite al *unpickler* importar e invocar código arbitrario. Solo considere lo que hace este flujo de datos de *pickle* hechos a mano cuando se carga:

```
>>> import pickle
>>> pickle.loads(b"cos\nsystem\n(S'echo hello world'\ntR
hello world
0
```

En este ejemplo, el *unpickler* importa la función `os.system()` y luego aplica el argumento de cadena *«echo hello world»*. Aunque este ejemplo es inofensivo, no es difícil imaginar uno que pueda dañar su sistema.

Por esta razón, es posible que desee controlar lo que se deserializa con *pickle* personalizando `Unpickler.find_class()`. A diferencia de lo que sugiere su nombre, `Unpickler.find_class()` se llama siempre que se solicita un global (es decir, una clase o una función). Por lo tanto, es posible prohibir completamente los globales o restringirlos a un subconjunto seguro.

Aquí hay un ejemplo de un *unpickler* que permite cargar solo unas pocas clases seguras del módulo **`builtins`**:

```
import builtins
import io
import pickle

safe_builtins = {
    'range',
    'complex',
    'set',
    'frozenset',
    'slice',
}

class RestrictedUnpickler(pickle.Unpickler):

    def find_class(self, module, name):
        # Only allow safe classes from builtins.
        if module == "builtins" and name in safe_builtin
            return getattr(builtins, name)
        # Forbid everything else.
        raise pickle.UnpicklingError("global '%s.%s' is
                                     (module, name))

def restricted_loads(s):
    """Helper function analogous to pickle.loads()."""
    return RestrictedUnpickler(io.BytesIO(s)).load()
```

Un ejemplo de uso de nuestro deserializador que funciona según lo previsto:

```
>>> restricted_loads(pickle.dumps([1, 2, range(15)]))
[1, 2, range(0, 15)]
>>> restricted_loads(b"cos\nsystem\n(S'echo hello world'
Traceback (most recent call last):
  ...
pickle.UnpicklingError: global 'os.system' is forbidden
>>> restricted_loads(b'cbuiltins\neval\n'
...                  b'(S\'getattr(__import__("os"), "sy
...                  b'("echo hello world")\'\ntR.')
Traceback (most recent call last):
  ...
pickle.UnpicklingError: global 'builtins.eval' is forbic
```

Como muestran nuestros ejemplos, debes tener cuidado con lo que permites que se deserialize con *pickle*. Por lo tanto, si la seguridad es un problema, puede considerar alternativas como la API de *marshalling* en **`xmlrpc.client`** o soluciones de terceros.

Performance

Recent versions of the pickle protocol (from protocol 2 and upwards) feature efficient binary encodings for several common features and built-in types. Also, the **`pickle`** module has a transparent optimizer written in C.

Ejemplos

Para obtener el código más simple, use las funciones **`dump()`** y **`load()`**.

```
import pickle

# An arbitrary collection of objects supported by pickle
data = {
    'a': [1, 2.0, 3+4j],
    'b': ("character string", b"byte string"),
    'c': {None, True, False}
}

with open('data.pickle', 'wb') as f:
    # Pickle the 'data' dictionary using the highest pro
    pickle.dump(data, f, pickle.HIGHEST_PROTOCOL)
```

El siguiente ejemplo lee los datos serializados con *pickle* resultantes.

```
import pickle

with open('data.pickle', 'rb') as f:
    # The protocol version used is detected automaticall
    # have to specify it.
    data = pickle.load(f)
```

Command-line interface

The **`pickle`** module can be invoked as a script from the command line, it will display contents of the pickle files. However, when the pickle file that you want to examine comes from an untrusted source, `-m pickletools` is a safer option because it does not execute pickle bytecode, see pickletools CLI usage.

```
python -m pickle pickle_file [pickle_file ...]
```

The following option is accepted:

pickle_file
: A pickle file to read, or `-` to indicate reading from standard input.

Ver también

Módulo `copyreg`
Registro de constructor de interfaz *Pickle* para tipos de extensión.

Módulo `pickletools`
Herramientas para trabajar y analizar datos serializados con *pickle*.

Módulo `shelve`
Indexed databases of objects; uses **`pickle`**.

Módulo `copy`
Copia de objetos superficial y profunda.

Módulo `marshal`
Serialización de alto rendimiento de tipos integrados.

Notas al pie

No confunda esto con el módulo `marshal`

Esta es la razón por la que las funciones `lambda` no se pueden serializar con *pickle*: todas las funciones **`lambda`** comparten el mismo nombre: `<lambda>`.

La excepción generada probablemente será un `ImportError` o un `AttributeError` pero podría ser otra cosa.

El módulo `copy` utiliza este protocolo para operaciones de copia superficial y profunda.

La limitación de caracteres alfanuméricos se debe a que los ID persistentes en el protocolo 0 están delimitados por el carácter de nueva línea. Por lo tanto, si se produce algún tipo de carácter de nueva línea en los ID persistentes, los datos serializados resultantes se volverán ilegibles.

`copyreg` — Register `pickle` support functions

Código fuente: Lib/copyreg.py [https://github.com/python/cpython/tree/3.14/Lib/copyreg.py]

The `copyreg` module offers a way to define functions used while pickling specific objects. The `pickle` and `copy` modules use those functions when pickling/copying those objects. The module provides configuration information about object constructors which are not classes. Such constructors may be factory functions or class instances.

copyreg.constructor(*object*)

Declara que el *object* es un constructor válido. Si el *object* no es invocable (y por lo tanto, no es válido como constructor), lanza una excepción `TypeError`.

copyreg.pickle(*type*, *function*, *constructor_ob*=*None*)

Declares that *function* should be used as a «reduction» function for objects of type *type*. *function* must return either a string or a tuple containing between two and six elements. See the `dispatch_table` for more details on the interface of *function*.

El parámetro *constructor_ob* es una característica heredada y ahora se ignora, pero si se pasa debe ser invocable.

Note que el atributo `dispatch_table` de un objeto pickler o subclase de `pickle.Pickler` también puede ser utilizado para declarar funciones de reducción.

Ejemplo

El siguiente ejemplo pretende mostrar cómo registrar una función pickle y cómo se utilizará:

```
>>> import copyreg, copy, pickle
>>> class C:
...     def __init__(self, a):
...         self.a = a
...
>>> def pickle_c(c):
...     print("pickling a C instance...")
...     return C, (c.a,)
...
>>> copyreg.pickle(C, pickle_c)
>>> c = C(1)
>>> d = copy.copy(c)
pickling a C instance...
>>> p = pickle.dumps(c)
pickling a C instance...
```

`shelve` — Python object persistence

Source code: Lib/shelve.py [https://github.com/python/cpython/tree/3.14/Lib/shelve.py]

Un «estante» o *shelve*, es un objeto persistente similar a un diccionario. La diferencia con las bases de datos «dbm» es que los valores (¡no las claves!) en un estante pueden ser esencialmente objetos Python arbitrarios — cualquier cosa que el módulo `pickle` pueda manejar. Esto incluye la mayoría de las instancias de clase, tipos de datos recursivos y objetos que contienen muchos subobjetos compartidos. Las claves son cadenas ordinarias.

shelve.open(*filename*, *flag='c'*, *protocol=None*, *writeback=False*)

Abre un diccionario persistente. El nombre de archivo especificado es el nombre de archivo base para la base de datos subyacente. Como efecto secundario, se puede agregar una extensión al nombre de archivo y se puede crear más de un archivo. De forma predeterminada, el archivo de base de datos subyacente se abre para leer y escribir. El parámetro opcional *flag* tiene la misma interpretación que el parámetro *flag* de `dbm.open()`.

By default, pickles created with `pickle.DEFAULT_PROTOCOL` are used to serialize values. The version of the pickle protocol can be specified with the *protocol* parameter.

Debido a la semántica de Python, un estante no puede saber cuándo se modifica una entrada de diccionario persistente mutable. De forma predeterminada, los objetos modificados se escriben *sólo* cuando se asignan al estante (consulte Ejemplo). Si el parámetro opcional *writeback* se establece en

`True`, todas las entradas a las que se accede también se almacenan en caché en la memoria y se vuelven a escribir en `sync()` y `close()`; esto puede hacer que sea más práctico mutar entradas mutables en el diccionario persistente, pero, si se accede a muchas entradas, puede consumir grandes cantidades de memoria para la memoria caché, y puede hacer que la operación de cierre sea muy lenta ya que todas las entradas a las que se accede se vuelven a escribir (no hay manera de determinar qué entradas a las que se accede son mutables, ni cuáles se mutaron realmente).

Distinto en la versión 3.10: `pickle.DEFAULT_PROTOCOL` is now used as the default pickle protocol.

Distinto en la versión 3.11: Acepta path-like object por nombre de archivo.

Nota

No confíe en que el estante se cerrará automáticamente; siempre llame a `close()` explícitamente cuando ya no lo necesite, o use `shelve.open()` como administrador de contexto:

```
with shelve.open('spam') as db:
    db['eggs'] = 'eggs'
```

Advertencia

Because the **`shelve`** module is backed by `pickle`, it is insecure to load a shelf from an untrusted source. Like with pickle, loading a shelf can execute arbitrary code.

Shelf objects support most of the methods and operations supported by dictionaries (except copying, constructors and operators `|` and `|=`). This eases the transition from dictionary based scripts to those requiring persistent storage.

Se admiten dos métodos adicionales:

Shelf.sync()

Escriba todas las entradas en la caché si el estante se abrió con *writeback* establecido en `True`. También vacíe la caché y sincronice el diccionario persistente en el disco, si es posible. Esto se llama automáticamente cuando el estante se cierra con `close()`.

Shelf.close()

Sincronice y cierre el objeto persistente *dict*. Las operaciones en un estante cerrado fallarán con un `ValueError`.

Ver también

Persistent dictionary recipe [https://code.activestate.com/recipes/576642-persistent-dict-with-multiple-standard-file-format/] with widely supported storage formats and having the speed of native dictionaries.

Restricciones

- La elección de qué paquete de base de datos se utilizará (como `dbm.ndbm` o `dbm.gnu`) depende de la interfaz disponible. Por lo tanto, no es seguro abrir la base de datos directamente usando `dbm`. La base de datos también está (desafortunadamente) sujeta a las limitaciones de `dbm`, si se usa — esto significa que (la representación *pickle* de) los objetos almacenados en la base de datos debe ser bastante pequeña, y en casos raros las colisiones de claves pueden hacer que la base de datos rechace las actualizaciones.
- The `shelve` module does not support *concurrent* read/write access to shelved objects. (Multiple simultaneous read accesses are safe.) When a program has a shelf open for writing, no other program should have it open for reading or writing. Unix file locking can be used to solve this, but this differs across Unix versions and requires knowledge about the database implementation used.
- On macOS `dbm.ndbm` can silently corrupt the database file on updates, which can cause hard crashes when trying to read

from the database.

class shelve.Shelf(*dict*, *protocol=None*, *writeback=False*, *keyencoding='utf-8'*)

Una subclase de `collections.abc.MutableMapping` que almacena valores *pickle* en el objeto *dict*.

By default, pickles created with `pickle.DEFAULT_PROTOCOL` are used to serialize values. The version of the pickle protocol can be specified with the *protocol* parameter. See the `pickle` documentation for a discussion of the pickle protocols.

Si el parámetro *writeback* es `True`, el objeto mantendrá un caché de todas las entradas a las que se accedió y las volverá a escribir en el *dict* en las horas de sincronización y cierre. Esto permite operaciones naturales en entradas mutables, pero puede consumir mucha más memoria y hacer que la sincronización y el cierre tomen mucho tiempo.

El parámetro *keyencoding* es la codificación utilizada para codificar las claves antes de que se utilicen con el *dict* subyacente.

El objeto `Shelf` también se puede utilizar como administrador de contexto, en cuyo caso se cerrará automáticamente cuando finalice el bloque `with`.

Distinto en la versión 3.2: Se agregó el parámetro *keyencoding*; anteriormente, las claves siempre estaban codificadas en UTF-8.

Distinto en la versión 3.4: Agregado soporte para administrador de contexto.

Distinto en la versión 3.10: `pickle.DEFAULT_PROTOCOL` is now used as the default pickle protocol.

class shelve.BsdDbShelf(*dict*, *protocol=None*, *writeback=False*, *keyencoding='utf-8'*)

A subclass of **`Shelf`** which exposes **`first()`**, **`next()`**, **`previous()`**, **`last()`** and **`set_location()`** methods. These are available in the third-party **`bsddb`** module from pybsddb [https://www.jcea.es/programacion/pybsddb.htm] but not in other database modules. The *dict* object passed to the constructor must support those methods. This is generally accomplished by calling one of **`bsddb.hashopen()`**, **`bsddb.btopen()`** or **`bsddb.rnopen()`**. The optional *protocol*, *writeback*, and *keyencoding* parameters have the same interpretation as for the **`Shelf`** class.

class shelve.DbfilenameShelf(*filename*, *flag*='c', *protocol*=*None*, *writeback*=*False*)

Una subclase de **`Shelf`** que acepta un *filename* en lugar de un objeto tipo diccionario (*dict*). El archivo subyacente se abrirá usando **`dbm.open()`**. De forma predeterminada, el archivo se creará y se abrirá tanto para lectura como para escritura. El parámetro opcional *flag* tiene la misma interpretación que para la función **`open()`**. Los parámetros opcionales *protocol* y *writeback* tienen la misma interpretación que para la clase **`Shelf`**.

Ejemplo

Para resumir la interfaz (`key` es una cadena de caracteres, `data` es un objeto arbitrario):

```
import shelve

d = shelve.open(filename)  # open -- file may get suffi
                           # library

d[key] = data              # store data at key (overwrit
                           # using an existing key)
data = d[key]              # retrieve a COPY of data at
                           # if no such key)
del d[key]                 # delete data stored at key (
                           # if no such key)
```

```
flag = key in d              # true if the key exists
klist = list(d.keys())       # a list of all existing keys

# as d was opened WITHOUT writeback=True, beware:
d['xx'] = [0, 1, 2]          # this works as expected, but
d['xx'].append(3)            # *this doesn't!* -- d['xx']

# having opened d without writeback=True, you need to co
temp = d['xx']               # extracts the copy
temp.append(5)               # mutates the copy
d['xx'] = temp               # stores the copy right back,

# or, d=shelve.open(filename,writeback=True) would let y
# d['xx'].append(5) and have it work as expected, BUT it
# consume more memory and make the d.close() operation s

d.close()                    # close it
```

Ver también

Módulo `dbm`
Interfaz genérica para bases de datos estilo `dbm`.

Módulo `pickle`
Object serialization used by **`shelve`**.

[1]

`marshal` — Internal Python object serialization

This module contains functions that can read and write Python values in a binary format. The format is specific to Python, but independent of machine architecture issues (e.g., you can write a Python value to a file on a PC, transport the file to a Mac, and read it back there). Details of the format are undocumented on purpose; it may change between Python versions (although it rarely does). [1]

This is not a general «persistence» module. For general persistence and transfer of Python objects through RPC calls, see the modules **`pickle`** and **`shelve`**. The **`marshal`** module exists mainly to support reading and writing the «pseudo-compiled» code for Python modules of `.pyc` files. Therefore, the Python maintainers reserve the right to modify the marshal format in backward incompatible ways should the need arise. The format of code objects is not compatible between Python versions, even if the version of the format is the same. De-serializing a code object in the incorrect Python version has undefined behavior. If you're serializing and de-serializing Python objects, use the **`pickle`** module instead – the performance is comparable, version independence is guaranteed, and pickle supports a substantially wider range of objects than marshal.

Advertencia

The **`marshal`** module is not intended to be secure against erroneous or maliciously constructed data. Never unmarshal data received from an untrusted or unauthenticated source.

Hay funciones que leen/escriben archivos, así como funciones que operan en objetos similares a bytes.

Not all Python object types are supported; in general, only objects whose value is independent from a particular invocation of Python can be written and read by this module. The following types are supported:

- Numeric types: `int`, `bool`, `float`, `complex`.
- Strings (`str`) and `bytes`. Bytes-like objects like `bytearray` are marshalled as `bytes`.
- Containers: `tuple`, `list`, `set`, `frozenset`, and (since `version` 5), `slice`. It should be understood that these are supported only if the values contained therein are themselves supported. Recursive containers are supported since `version` 3.
- The singletons `None`, `Ellipsis` and `StopIteration`.
- `code` objects, if *allow_code* is true. See note above about version dependence.

Distinto en la versión 3.4:

- Added format version 3, which supports marshalling recursive lists, sets and dictionaries.
- Added format version 4, which supports efficient representations of short strings.

Distinto en la versión 3.14: Added format version 5, which allows marshalling slices.

El módulo define estas funciones:

marshal.dump(*value*, *file*, *version*=*version*, /, *, *allow_code*=*True*)

Escribe el valor en el archivo abierto. El valor debe ser un tipo admitido. El archivo debe ser un archivo binary file en el que se pueda escribir.

If the value has (or contains an object that has) an unsupported type, a `ValueError` exception is raised — but garbage data will also be written to the file. The object will not be properly read back by `load()`. Code objects are only supported if *allow_code* is true.

El argumento *version* indica el formato de datos que `dump` debe usar (véase más adelante).

Lanza un evento de auditoría `marshal.dumps` con argumentos `value`, `version`.

Distinto en la versión 3.13: Added the *allow_code* parameter.

marshal.load(*file*, */*, ***, *allow_code=True*)

Read one value from the open file and return it. If no valid value is read (e.g. because the data has a different Python version's incompatible marshal format), raise **`EOFError`**, **`ValueError`** or **`TypeError`**. Code objects are only supported if *allow_code* is true. The file must be a readable binary file.

Lanza un evento de auditoría `marshal.load` sin argumentos.

Nota

Si un objeto que contiene un tipo no admitido se calcula con **`dump()`**, **`load()`** sustituirá `None` por el tipo «unmarshallable».

Distinto en la versión 3.10: Esta llamada solía lanzar un evento de auditoría `code.__new__` para cada objeto código. Ahora lanza un único evento `marshal.load` para toda la operación de carga.

Distinto en la versión 3.13: Added the *allow_code* parameter.

marshal.dumps(*value*, *version=version*, */*, ***, *allow_code=True*)

Return the bytes object that would be written to a file by `dump(value, file)`. The value must be a supported type. Raise a **`ValueError`** exception if value has (or contains an object that has) an unsupported type. Code objects are only supported if *allow_code* is true.

El argumento *version* indica el formato de datos que `dumps` debe usar (véase más adelante).

Lanza un evento de auditoría `marshal.dumps` con argumentos `value`, `version`.

Distinto en la versión 3.13: Added the *allow_code* parameter.

marshal.loads(*bytes*, /, *, *allow_code=True*)

Convert the bytes-like object to a value. If no valid value is found, raise **`EOFError`**, **`ValueError`** or **`TypeError`**. Code objects are only supported if *allow_code* is true. Extra bytes in the input are ignored.

Lanza un evento de auditoría `marshal.loads` con argumento `bytes`.

Distinto en la versión 3.10: Esta llamada solía lanzar un evento de auditoría `code.__new__` para cada objeto código. Ahora lanza un único evento `marshal.loads` para toda la operación de carga.

Distinto en la versión 3.13: Added the *allow_code* parameter.

Además, se definen las siguientes constantes:

marshal.version

Indicates the format that the module uses. Version 0 is the historical first version; subsequent versions add new features. Generally, a new version becomes the default when it is introduced.

Version	**Available since**	**New features**
1	Python 2.4	Sharing interned strings
2	Python 2.5	Binary representation of floats
3	Python 3.4	Support for object instancing and recursion
4	Python 3.4	Efficient representation of short strings
5	Python 3.14	Support for **`slice`** objects

Notas al pie

El nombre de este módulo proviene de algunos términos utilizados por los diseñadores de Modula-3 (entre otros), que utilizan el término «marshalling» para el envío de datos de forma autocontenida. Estrictamente hablando «marshalling», significa convertir algunos datos internos en un formato externo (por ejemplo, en un búfer RPC) y «unmarshalling» es el proceso inverso.

`dbm` — Interfaces to Unix «databases»

Código fuente: Lib/dbm/__init__.py [https://github.com/python/cpython/tree/3.14/Lib/dbm/__init__.py]

`dbm` is a generic interface to variants of the DBM database:

- `dbm.sqlite3`
- `dbm.gnu`
- `dbm.ndbm`

If none of these modules are installed, the slow-but-simple implementation in module `dbm.dumb` will be used. There is a third party interface [https://www.jcea.es/programacion/pybsddb.htm] to the Oracle Berkeley DB.

exception dbm.error

> Una tupla que contiene las excepciones que pueden ser lanzadas por cada uno de los módulos soportados, con una excepción única también denominada `dbm.error` como el primer elemento — el último se usa cuando se genera `dbm.error`.

dbm.whichdb(*filename*)

> This function attempts to guess which of the several simple database modules available — `dbm.sqlite3`, `dbm.gnu`, `dbm.ndbm`, or `dbm.dumb` — should be used to open a given file.
>
> Return one of the following values:
>
> - `None` if the file can't be opened because it's unreadable or doesn't exist

- the empty string (`''`) if the file's format can't be guessed
- a string containing the required module name, such as `'dbm.ndbm'` or `'dbm.gnu'`

Distinto en la versión 3.11: filename accepts a path-like object.

dbm.open(*file*, *flag='r'*, *mode=0o666*)

Open a database and return the corresponding database object.

Parámetros:

- **file** (path-like object) –
  The database file to open.
  If the database file already exists, the **`whichdb()`** function is used to determine its type and the appropriate module is used; if it does not exist, the first submodule listed above that can be imported is used.
- **flag** (*str*) –
  - ○ `'r'` (default): Open existing database for reading only.
  - ○ `'w'`: Open existing database for reading and writing.
  - ○ `'c'`: Open database for reading and writing, creating it if it doesn't exist.
  - ○ `'n'`: Always create a new, empty database, open for reading and writing.
- **mode** (*int*) – The Unix file access mode of the file (default: octal `0o666`), used only when the database has to be created.

Distinto en la versión 3.11: file accepts a path-like object.

The object returned by **`open()`** supports the basic functionality of mutable mappings; keys and their corresponding values can be stored, retrieved, and deleted, and iteration, the **`in`** operator and

methods `keys()`, `get()`, `setdefault()` and `clear()` are available. The `keys()` method returns a list instead of a view object. The `setdefault()` method requires two arguments.

Key and values are always stored as `bytes`. This means that when strings are used they are implicitly converted to the default encoding before being stored.

Estos objetos también admiten el uso en una instrucción `with`, que los cerrará automáticamente cuando termine.

Distinto en la versión 3.2: `get()` and `setdefault()` methods are now available for all `dbm` backends.

Distinto en la versión 3.4: Added native support for the context management protocol to the objects returned by `open()`.

Distinto en la versión 3.8: Deleting a key from a read-only database raises a database module specific exception instead of `KeyError`.

Distinto en la versión 3.13: `clear()` methods are now available for all `dbm` backends.

El siguiente ejemplo registra algunos nombres de host y un título correspondiente, y luego imprime el contenido de la base de datos:

```
import dbm

# Open database, creating it if necessary.
with dbm.open('cache', 'c') as db:

    # Record some values
    db[b'hello'] = b'there'
    db['www.python.org'] = 'Python Website'
    db['www.cnn.com'] = 'Cable News Network'

    # Note that the keys are considered bytes now.
    assert db[b'www.python.org'] == b'Python Website'
    # Notice how the value is now in bytes.
    assert db['www.cnn.com'] == b'Cable News Network'
    # Often-used methods of the dict interface work too.
    print(db.get('python.org', b'not present'))

    # Storing a non-string key or value will raise an ex
    # likely a TypeError).
    db['www.yahoo.com'] = 4

# db is automatically closed when leaving the with state
```

Ver también

Módulo `shelve`
: Módulo de persistencia que almacena datos que no son cadenas de caracteres.

Los submódulos individuales se describen en las siguientes secciones.

`dbm.sqlite3` — SQLite backend for dbm

Added in version 3.13.

Source code: Lib/dbm/sqlite3.py [https://github.com/python/cpython/tree/3.14/Lib/dbm/sqlite3.py]

This module uses the standard library `sqlite3` module to provide an SQLite backend for the `dbm` module. The files created by `dbm.sqlite3` can thus be opened by `sqlite3`, or any other SQLite browser, including the SQLite CLI.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

dbm.sqlite3.open(*filename*, /, *flag='r'*, *mode=0o666*)

Open an SQLite database.

Parámetros:

- **filename** (path-like object) – The path to the database to be opened.
- **flag** (*str*) –
  - `'r'` (default): Open existing database for reading only.
  - `'w'`: Open existing database for reading and writing.
  - `'c'`: Open database for reading and writing, creating it if it doesn't exist.
  - `'n'`: Always create a new, empty database, open for reading and writing.
- **mode** – The Unix file access mode of the file (default: octal `0o666`), used only when the database has to be created.

The returned database object behaves similar to a mutable mapping, but the **`keys()`** method returns a list, and the **`setdefault()`** method requires two arguments. It also supports a «closing» context manager via the **`with`** keyword.

The following method is also provided:

sqlite3.close()

Close the SQLite database.

`dbm.gnu` — GNU database manager

Código fuente: Lib/dbm/gnu.py [https://github.com/python/cpython/tree/3.14/Lib/dbm/gnu.py]

The **`dbm.gnu`** module provides an interface to the GDBM library, similar to the **`dbm.ndbm`** module, but with additional functionality like crash tolerance.

Nota

The file formats created by `dbm.gnu` and `dbm.ndbm` are incompatible and can not be used interchangeably.

Availability: not Android, not iOS, not WASI.

This module is not supported on mobile platforms or WebAssembly platforms.

Availability: Unix.

exception dbm.gnu.error

Raised on `dbm.gnu`-specific errors, such as I/O errors. `KeyError` is raised for general mapping errors like specifying an incorrect key.

dbm.gnu.open_flags

A string of characters the *flag* parameter of `open()` supports.

dbm.gnu.open(*filename*, *flag='r'*, *mode=0o666*, */*)

Open a GDBM database and return a `gdbm` object.

Parámetros:

- **filename** (path-like object) – The database file to open.
- **flag** (*str*) –
  - ○ `'r'` (default): Open existing database for reading only.
  - ○ `'w'`: Open existing database for reading and writing.
  - ○ `'c'`: Open database for reading and writing, creating it if it doesn't exist.
  - ○ `'n'`: Always create a new, empty database, open for reading and writing.

  The following additional characters may be appended to control how the database is opened:
  - ○ `'f'`: Open the database in fast mode. Writes to the database will not be

    synchronized.
    - ○ `'s'`: Synchronized mode. Changes to the database will be written immediately to the file.
    - ○ `'u'`: Do not lock database.

  Not all flags are valid for all versions of GDBM. See the `open_flags` member for a list of supported flag characters.
- **mode** (*int*) – The Unix file access mode of the file (default: octal `0o666`), used only when the database has to be created.

Muestra:

error – If an invalid *flag* argument is passed.

Distinto en la versión 3.11: filename accepts a path-like object.

`gdbm` objects behave similar to mutable mappings, but methods **`items()`**, **`values()`**, **`pop()`**, **`popitem()`**, and **`update()`** are not supported, the **`keys()`** method returns a list, and the **`setdefault()`** method requires two arguments. It also supports a «closing» context manager via the **`with`** keyword.

Distinto en la versión 3.2: Added the **`get()`** and **`setdefault()`** methods.

Distinto en la versión 3.13: Added the **`clear()`** method.

The following methods are also provided:

gdbm.close()

Close the GDBM database.

gdbm.firstkey()

It’s possible to loop over every key in the database using this method and the **`nextkey()`** method. The traversal is ordered by GDBM’s internal hash values, and won’t be sorted by the key values. This method

returns the starting key.

gdbm.nextkey(*key*)

Retorna la clave que sigue a *key* en el recorrido. El siguiente código imprime todas las claves en la base de datos `db`, sin tener que crear una lista en la memoria que las contenga todas:

```
k = db.firstkey()
while k is not None:
    print(k)
    k = db.nextkey(k)
```

gdbm.reorganize()

If you have carried out a lot of deletions and would like to shrink the space used by the GDBM file, this routine will reorganize the database. **`gdbm`** objects will not shorten the length of a database file except by using this reorganization; otherwise, deleted file space will be kept and reused as new (key, value) pairs are added.

gdbm.sync()

Cuando la base de datos se ha abierto en modo rápido, este método obliga a que los datos no escritos se escriban en el disco.

`dbm.ndbm` — New Database Manager

Código fuente: Lib/dbm/ndbm.py [https://github.com/python/cpython/tree/3.14/Lib/dbm/ndbm.py]

The **`dbm.ndbm`** module provides an interface to the NDBM library. This module can be used with the «classic» NDBM interface or the GDBM compatibility interface.

Nota

The file formats created by `dbm.gnu` and `dbm.ndbm` are incompatible and can not be used interchangeably.

Advertencia

The NDBM library shipped as part of macOS has an undocumented limitation on the size of values, which can result in corrupted database files when storing values larger than this limit. Reading such corrupted files can result in a hard crash (segmentation fault).

Availability: not Android, not iOS, not WASI.

This module is not supported on mobile platforms or WebAssembly platforms.

Availability: Unix.

exception dbm.ndbm.error

Raised on `dbm.ndbm`-specific errors, such as I/O errors. `KeyError` is raised for general mapping errors like specifying an incorrect key.

dbm.ndbm.library

Name of the NDBM implementation library used.

dbm.ndbm.open(*filename*, *flag='r'*, *mode=0o666*, */*)

Open an NDBM database and return an `ndbm` object.

Parámetros:

- **filename** (path-like object) – The basename of the database file (without the `.dir` or `.pag` extensions).
- **flag** (*str*) –
  - ○ `'r'` (default): Open existing database for reading only.
  - ○ `'w'`: Open existing database for reading and writing.

    - ○ `'c'`: Open database for reading and writing, creating it if it doesn't exist.
    - ○ `'n'`: Always create a new, empty database, open for reading and writing.
  - **mode** (*int*) – The Unix file access mode of the file (default: octal `0o666`), used only when the database has to be created.

Distinto en la versión 3.11: Acepta path-like object como nombre de archivo.

`ndbm` objects behave similar to mutable mappings, but methods **`items()`**, **`values()`**, **`pop()`**, **`popitem()`**, and **`update()`** are not supported, the **`keys()`** method returns a list, and the **`setdefault()`** method requires two arguments. It also supports a «closing» context manager via the **`with`** keyword.

Distinto en la versión 3.2: Added the **`get()`** and **`setdefault()`** methods.

Distinto en la versión 3.13: Added the **`clear()`** method.

The following method is also provided:

ndbm.close()

Close the NDBM database.

`dbm.dumb` — Portable DBM implementation

Código fuente: Lib/dbm/dumb.py [https://github.com/python/cpython/tree/3.14/Lib/dbm/dumb.py]

Nota

The **`dbm.dumb`** module is intended as a last resort fallback for the **`dbm`** module when a more robust module is not available. The **`dbm.dumb`** module is not written for speed and is not nearly as heavily used as the other database modules.

The **`dbm.dumb`** module provides a persistent `dict`-like interface which is written entirely in Python. Unlike other **`dbm`** backends, such as `dbm.gnu`, no external library is required.

The **`dbm.dumb`** module defines the following:

exception dbm.dumb.error

Raised on **`dbm.dumb`**-specific errors, such as I/O errors. `KeyError` is raised for general mapping errors like specifying an incorrect key.

dbm.dumb.open(*filename*, *flag='c'*, *mode=0o666*)

Open a **`dbm.dumb`** database.

Parámetros:

- **filename** – The basename of the database file (without extensions). A new database creates the following files:
  - `filename.dat`
  - `filename.dir`
- **flag** (*str*) –
  - `'r'`: Open existing database for reading only.
  - `'w'`: Open existing database for reading and writing.
  - `'c'` (default): Open database for reading and writing, creating it if it doesn’t exist.
  - `'n'`: Always create a new, empty database, open for reading and writing.
- **mode** (*int*) – The Unix file access mode of the file (default: octal `0o666`), used

only when the database has to be created.

Advertencia

Es posible bloquear el intérprete de Python cuando se carga una base de datos con una entrada suficientemente grande/compleja debido a las limitaciones de profundidad de la pila en el compilador AST de Python.

Distinto en la versión 3.5: `open()` always creates a new database when *flag* is `'n'`.

Distinto en la versión 3.8: A database opened read-only if *flag* is `'r'`. A database is not created if it does not exist if *flag* is `'r'` or `'w'`.

Distinto en la versión 3.11: filename accepts a path-like object.

The returned database object behaves similar to a mutable mapping, but the `keys()` and `items()` methods return lists, and the `setdefault()` method requires two arguments. It also supports a «closing» context manager via the `with` keyword.

The following methods are also provided:

dumbdbm.close()

Close the database.

dumbdbm.sync()

Synchronize the on-disk directory and data files. This method is called by the `shelve.Shelf.sync()` method.

`sqlite3` — DB-API 2.0 interfaz para bases de datos SQLite

Código fuente: Lib/sqlite3/ [https://github.com/python/cpython/tree/3.14/Lib/sqlite3/]

SQLite es una biblioteca de C que provee una base de datos ligera basada en disco que no requiere un proceso de servidor separado y permite acceder a la base de datos usando una variación no estándar del lenguaje de consulta SQL. Algunas aplicaciones pueden usar SQLite para almacenamiento interno. También es posible prototipar una aplicación usando SQLite y luego transferir el código a una base de datos más grande como PostgreSQL u Oracle.

The `sqlite3` module was written by Gerhard Häring. It provides an SQL interface compliant with the DB-API 2.0 specification described by **PEP 249** [https://peps.python.org/pep-0249/], and requires the third-party SQLite [https://sqlite.org/] library.

This is an optional module. If it is missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

Esta documentación contiene cuatro secciones principales:

- Tutorial enseña como usar el módulo `sqlite3` module.
- Referencia describe las clases y funciones que se definen en este módulo.
- Guías prácticas detalla como manipular tareas específicas.
- Explicación proporciona en profundidad información sobre el control transaccional.

Ver también

https://www.sqlite.org

La página web SQLite; la documentación describe la sintaxis y los tipos de datos disponibles para el lenguaje SQL soportado.

https://www.w3schools.com/sql/
Tutorial, referencia y ejemplos para aprender sintaxis SQL.

PEP 249 [https://peps.python.org/pep-0249/] - Especificación de la API 2.0 de base de datos
PEP escrito por Marc-André Lemburg.

Tutorial

En este tutorial, será creada una base de datos de películas Monty Python usando funcionalidades básicas de `sqlite3`. Se asume un entendimiento fundamental de conceptos de bases de dados, como cursors [https://en.wikipedia.org/wiki/Cursor_(databases)] y transactions [https://en.wikipedia.org/wiki/Database_transaction].

Primero, necesitamos crear una nueva base de datos y abrir una conexión de base de datos para permitir que `sqlite3` trabaje con ella. Llamamos a `sqlite3.connect()` para crear una conexión a la base de datos `tutorial.db` en el directorio de trabajo actual, creándola implícitamente si no existe:

```
import sqlite3
con = sqlite3.connect("tutorial.db")
```

El retorno será un objecto de la clase `Connection` representado en `con` como una base de datos local.

Con el fin de ejecutar sentencias SQL y obtener resultados de consultas SQL, necesitaremos usar un cursor para la base de datos. Llamando `con.cursor()` se creará el `Cursor`:

```
cur = con.cursor()
```

Ahora que hemos configurado una conexión y un cursor, podemos crear una tabla `movie` con las columnas *title*, *release year*, y *review score*. Para simplificar, podemos solamente usar nombres de columnas en la declaración de la tabla – gracias al recurso de flexible typing [https://www.sqlite.org/flextypegood.html] SQLite, especificar los tipos de datos es opcional. Ejecuta la sentencia `CREATE TABLE` llamando **`cur.execute(...)`**:

```
cur.execute("CREATE TABLE movie(title, year, score)")
```

Podemos verificar que una nueva tabla ha sido creada consultando la tabla `sqlite_master` incorporada en SQLite, la cual ahora debería contener una entrada para la tabla `movie` (consulte The Schema Table [https://www.sqlite.org/schematab.html] para más información). Ejecute esa consulta llamando a **`cur.execute(...)`**, asigne el resultado a `res` y llame a **`res.fetchone()`** para obtener la fila resultante:

```
>>> res = cur.execute("SELECT name FROM sqlite_master")
>>> res.fetchone()
('movie',)
```

Podemos ver que la tabla ha sido creada, ya que la consulta devuelve un **`tuple`** que contiene el nombre de la tabla. Si consultamos `sqlite_master` para una tabla inexistente `spam`, **`res.fetchone()`** devolverá `None`:

```
>>> res = cur.execute("SELECT name FROM sqlite_master WH
>>> res.fetchone() is None
True
```

Ahora, agrega dos filas de datos proporcionados como SQL literales, ejecutando la sentencia `INSERT`, una vez más llamando a **`cur.execute(...)`**:

```
cur.execute("""
    INSERT INTO movie VALUES
        ('Monty Python and the Holy Grail', 1975, 8.2),
        ('And Now for Something Completely Different', 1
""")
```

La sentencia `INSERT` implícitamente abre una transacción, la cual necesita ser confirmada antes que los cambios sean guardados en la base de datos (consulte Control transaccional para más información). Llamando a **`con.commit()`** sobre el objeto de la conexión, se confirmará la transacción:

```
con.commit()
```

Podemos verificar que la información fue introducida correctamente ejecutando la consulta `SELECT`. Ejecute el ahora conocido **`cur.execute(...)`** para asignar el resultado a `res`, y luego llame a **`res.fetchall()`** para obtener todas las filas como resultado:

```
>>> res = cur.execute("SELECT score FROM movie")
>>> res.fetchall()
[(8.2,), (7.5,)]
```

El resultado es una **`list`** de dos **`tuple`**s, una por fila, cada una conteniendo el valor del `score` de esa fila.

Ahora, introduzca tres filas más llamando **`cur.executemany(...)`**:

```
data = [
    ("Monty Python Live at the Hollywood Bowl", 1982, 7.
    ("Monty Python's The Meaning of Life", 1983, 7.5),
    ("Monty Python's Life of Brian", 1979, 8.0),
]
cur.executemany("INSERT INTO movie VALUES(?, ?, ?)", dat
con.commit()  # Remember to commit the transaction afte
```

Note que los marcadores de posición `?` son usados para enlazar `data` a la consulta. SIempre use marcadores de posición en lugar de string formatting para unir valores Python a sentencias SQL, y así evitará SQL injection attacks [https://en.wikipedia.org/wiki/SQL_injection] (consulte Cómo usar marcadores de posición para vincular valores en consultas SQL para más información).

Podemos verificar que las nuevas filas fueron introducidas ejecutando una consulta `SELECT`, esta vez iterando sobre los resultados de la consulta:

```
>>> for row in cur.execute("SELECT year, title FROM movi
...     print(row)
(1971, 'And Now for Something Completely Different')
(1975, 'Monty Python and the Holy Grail')
(1979, "Monty Python's Life of Brian")
(1982, 'Monty Python Live at the Hollywood Bowl')
(1983, "Monty Python's The Meaning of Life")
```

Cada fila es una `tuple` de dos items con `(year, title)`, donde las columnas seleccionadas coinciden con la consulta.

Finalmente, se puede verificar que la base de datos ha sido escrita en disco, llamando `con.close()` para cerrar la conexión existente, abriendo una nueva, creando un nuevo cursor y luego consultando la base de datos:

```
>>> con.close()
>>> new_con = sqlite3.connect("tutorial.db")
>>> new_cur = new_con.cursor()
>>> res = new_cur.execute("SELECT title, year FROM movie
>>> title, year = res.fetchone()
>>> print(f'The highest scoring Monty Python movie is {t
The highest scoring Monty Python movie is 'Monty Python 
>>> new_con.close()
```

Ahora ha creado una base de datos SQLite usando el módulo **`sqlite3`**, insertado datos y recuperado valores de varias maneras.

Ver también

- Guías prácticas para lecturas de interés:
  - Cómo usar marcadores de posición para vincular valores en consultas SQL
  - Cómo adaptar tipos de Python personalizados a valores de SQLite
  - Como convertir valores SQLite a tipos de Python personalizados
  - Como usar la conexión con un administrador de contexto
  - Cómo crear y utilizar fábricas de filas
- Explicación para obtener información detallada sobre el

control de transacciones..

Referencia

Funciones del módulo

sqlite3.connect(*database*, *timeout=5.0*, *detect_types=0*, *isolation_level='DEFERRED'*, *check_same_thread=True*, *factory=sqlite3.Connection*, *cached_statements=128*, *uri=False*, ***, *autocommit=sqlite3.LEGACY_TRANSACTION_CONTROL*)

Abre una conexión con una base de datos SQLite.

Parámetros:

- **database** (path-like object) – La ruta al archivo de base de datos que se va a abrir. Puede pasar `":memory:"` para crear un SQLite database existing only in memory [https://sqlite.org/inmemorydb.html] y abrir una conexión con él.
- **timeout** (*float*) – Cuántos segundos debe esperar la conexión antes de generar un **`OperationalError`** cuando una tabla está bloqueada. Si otra conexión abre una transacción para modificar una tabla, esa tabla se bloqueará hasta que se confirme la transacción. El valor predeterminado es cinco segundos.
- **detect_types** (*int*) – Control whether and how data types not natively supported by SQLite are looked up to be converted to Python types, using the converters registered with **`register_converter()`**. Set it to any combination (using `|`, bitwise or) of **`PARSE_DECLTYPES`** and **`PARSE_COLNAMES`** to enable this.

Column names take precedence over declared types if both flags are set. By default (`0`), type detection is disabled.

- **isolation_level** (*str* | *None*) – Controla el comportamiento de manejo de transacciones heredadas. Consulta **`Connection.isolation_level`** y Control de transacciones mediante el atributo isolation_level para obtener más información. Puede ser `"DEFERRED"` (predeterminado), `"EXCLUSIVE"` o `"IMMEDIATE"`; o `None` para deshabilitar la apertura de transacciones de manera implícita. No tiene efecto a menos que **`Connection.autocommit`** se configure en **`LEGACY_TRANSACTION_CONTROL`** (predeterminado).
- **check_same_thread** (*bool*) – Si es `True` (predeterminado), se generará **`ProgrammingError`** si la conexión de la base de datos la utiliza un subproceso distinto del que la creó. Si es `False`, se puede acceder a la conexión en varios subprocesos; es posible que el usuario deba serializar las operaciones de escritura para evitar la corrupción de datos. Consulte **`threadsafety`** para obtener más información.
- **factory** (*Connection*) – Una subclase personalizada de **`Connection`** con la que crear la conexión, sino la **`Connection`** será la predeterminada.
- **cached_statements** (*int*) – El número de instrucciones que **`sqlite3`** debe almacenar internamente en caché para esta conexión, para evitar la sobrecarga de análisis. El valor predeterminada,

son 128 instrucciones.
- **uri** (*bool*) – Si se establece en `True`, *database* es interpretada como una URI con la ruta de un archivo y una consulta de cadena de caracteres de modo opcional. La parte del esquema *debe* ser `"file:"`, y la ruta puede ser relativa o absoluta. La consulta permite pasar parámetros a SQLite, habilitando varias Como trabajar con URIs SQLite.
- **autocommit** (*bool*) – Controla el comportamiento de manejo de transacciones de **PEP 249** [https://peps.python.org/pep-0249/]. Consulta **`Connection.autocommit`** y Control de transacciones mediante el atributo autocommit para obtener más información. *autocommit* tiene como valor predeterminado **`LEGACY_TRANSACTION_CONTROL`**. El valor predeterminado cambiará a `False` en una futura versión de Python.

Tipo del valor devuelto:
Connection

Lanza un auditing event `sqlite3.connect` con el argumento `database`.

Lanza un auditing event `sqlite3.connect/handle` con el argumento `connection_handle`.

Distinto en la versión 3.4: Se agregó el parámetro *uri*.

Distinto en la versión 3.7: *database* ahora también puede ser un path-like object, no solo una cadena de caracteres.

Distinto en la versión 3.10: Se agregó el evento de auditoría `sqlite3.connect/handle`.

Distinto en la versión 3.12: Se agregó el parámetro *autocommit*.

Distinto en la versión 3.13: El uso posicional de los parámetros *timeout*, *detect_types*, *isolation_level*, *check_same_thread*, *factory*, *cached_statements* y *uri* está obsoleto. Se convertirán en parámetros de solo palabras clave en Python 3.15.

sqlite3.complete_statement(*statement*)

Retorna `True` si la cadena de caracteres *statement* pareciera contener uno o más sentencias SQL completas. No se realiza ninguna verificación o análisis sintáctico de ningún tipo, aparte de comprobar que no hay cadena de caracteres literales sin cerrar y que la sentencia se termine con un punto y coma.

Por ejemplo:

```
>>> sqlite3.complete_statement("SELECT foo FROM bar
True
>>> sqlite3.complete_statement("SELECT foo")
False
```

Esta función puede ser útil con entradas de línea de comando para determinar si los textos introducidos parecen formar una sentencia completa SQL, o si una entrada adicional se necesita antes de llamar a **`execute()`**.

Consulte **`runsource()`** en Lib/sqlite3/__main__.py [https://github.com/python/cpython/tree/3.14/Lib/sqlite3/__main__.py] para uso en el mundo real.

sqlite3.enable_callback_tracebacks(*flag*, /)

Habilita o deshabilita seguimiento de retrollamadas. Por defecto no se obtendrá ningún *tracebacks* en funciones *aggregates*, *converters*, autorizador de *callbacks* etc, definidas por el usuario. Si quieres depurarlas, se puede llamar esta función con *flag* establecida como `True`. Después se obtendrán *tracebacks* de los *callbacks* en `sys.stderr`. Usar **`False`** para deshabilitar la característica de nuevo.

Nota

Los errores en las devoluciones de llamadas de funciones definidas por el usuario se registran como excepciones que no se pueden generar. Utilice un **`unraisable hook handler`** para la introspección de la devolución de llamada fallida.

sqlite3.register_adapter(*type*, *adapter*, */*)

Registre un *adapter* callable para adaptar el tipo Python *type* a un tipo SQLite. El adaptador se llama con un objeto Python de tipo *type* como único argumento y debe devolver un valor type that SQLite natively understands.

sqlite3.register_converter(*typename*, *converter*, */*)

Registre *converter* callable para convertir objetos SQLite de tipo *typename* en un objeto Python de un tipo específico. El convertidor se invoca para todos los valores SQLite de tipo *typename*; se le pasa un objeto **`bytes`** y debe devolver un objeto del tipo Python deseado. Consulte el parámetro *detect_types* de **`connect()`** para obtener información sobre cómo funciona la detección de tipos.

Nota: *typename* y el nombre del tipo en tu consulta coinciden sin distinción entre mayúsculas y minúsculas.

Constantes del módulo

sqlite3.LEGACY_TRANSACTION_CONTROL

Establezca **`autocommit`** en esta constante para seleccionar el comportamiento de control de transacciones de estilo antiguo (anterior a Python 3.12). Consulte Control de transacciones mediante el atributo isolation_level para obtener más información.

sqlite3.PARSE_DECLTYPES

Pasa este valor de flag como parámetro *detect_types* de la **`connect()`** para buscar una función *converter* usando los tipos declarados para cada columna. Los tipos son declarados cuando la tabla de la base de datos se creó. **`sqlite3`** buscará una función *converter* usando la primera palabra del tipo declarado como la llave del diccionario conversor. Por ejemplo:

```
CREATE TABLE test(
   i integer primary key,  ! will look up a convert
   p point,                ! will look up a convert
   n number(10)            ! will look up a convert
 )
```

Esta puede ser combinada con **`PARSE_COLNAMES`** usando el operador `|` (*bitwise or*).

Nota

Generated fields (for example `MAX(p)`) are returned as **`str`**. Use **`PARSE_COLNAMES`** to enforce types for such queries.

sqlite3.PARSE_COLNAMES

Pass this flag value to the *detect_types* parameter of **`connect()`** to look up a converter function by using the type name, parsed from the query column name, as the converter dictionary key. The query column name must be wrapped in double quotes (`"`) and the type name must be wrapped in square brackets (`[]`).

```
SELECT MAX(p) as "p [point]" FROM test;  ! will loo
```

Esta flag puede ser combinada con **`PARSE_DECLTYPES`** usando el operador `|` (*bitwise or*) .

sqlite3.SQLITE_OK
sqlite3.SQLITE_DENY
sqlite3.SQLITE_IGNORE

Banderas que debe devolver el *authorizer_callback* que callable pasa a **`Connection.set_authorizer()`**, para indicar si:

- Acceso es permitido (**`SQLITE_OK`**),
- La sentencia SQL debería ser abortada con un error (**`SQLITE_DENY`**)
- La columna debería ser tratada como un valor `NULL` (**`SQLITE_IGNORE`**)

sqlite3.apilevel

Cadena de caracteres constante que indica el nivel DB-API soportado. Requerido por DB-API. Codificado de forma rígida en `"2.0"`.

sqlite3.paramstyle

Cadena de caracteres que indica el tipo de formato del marcador de parámetros esperado por el módulo **`sqlite3`**. Requerido por DB-API. Codificado de forma rígida en `"qmark"`.

Nota

También se admite el estilo de parámetro `named` DB-API.

sqlite3.sqlite_version

El número de versión de la librería SQLite en tiempo de ejecución, como una **`cadena de caracteres`**.

sqlite3.sqlite_version_info

El número de versión de la librería SQLite en tiempo de ejecución, como una **`tupla`** de **`enteros`**.

sqlite3.threadsafety

Constante de tipo entero requerida por la DB-API 2.0, la cual indica el nivel de seguridad de subproceso que **`sqlite3`** soporta. Este atributo se establece basándose en el modo de subprocesamiento [https://sqlite.org/threadsafe.html] por defecto con el que la biblioteca SQLite se compila. Los modos de subprocesamiento de SQLite son:

1. **Single-thread**: En este modo, todos los *mutexes* son deshabilitados y no es seguro usar SQLite en más de un solo hilo a la vez.
2. **Multi-thread**: En este modo, es seguro usar SQLite desde varios hilos dado que que ninguna conexión de base de datos sea usada simultáneamente en dos o más hilos.
3. **Serialized**: En el modo serializado, es seguro usar SQLite por varios hilos sin restricción.

Las asignaciones de los modos de subprocesos SQLite a los niveles de seguridad de subprocesos de DB-API 2.0 son las siguientes:

Modo de subprocesamiento SQLite	threadsafety	SQLITE_THREADSAFE	Significado de DB-API 2.0
single-thread	0	0	Hilos no pueden compartir el módulo
multi-thread	1	2	Hilos pueden compartir el módulo, pero no conexiones
serialized	3	1	Hilos pueden compartir el módulo, conexiones y cursores

Distinto en la versión 3.11: Establece *threadsafety* dinámicamente en vez de codificarlo rígidamente a `1`.

sqlite3.SQLITE_DBCONFIG_DEFENSIVE
sqlite3.SQLITE_DBCONFIG_DQS_DDL
sqlite3.SQLITE_DBCONFIG_DQS_DML
sqlite3.SQLITE_DBCONFIG_ENABLE_FKEY
sqlite3.SQLITE_DBCONFIG_ENABLE_FTS3_TOKENIZER
sqlite3.SQLITE_DBCONFIG_ENABLE_LOAD_EXTENSION
sqlite3.SQLITE_DBCONFIG_ENABLE_QPSG
sqlite3.SQLITE_DBCONFIG_ENABLE_TRIGGER
sqlite3.SQLITE_DBCONFIG_ENABLE_VIEW
sqlite3.SQLITE_DBCONFIG_LEGACY_ALTER_TABLE
sqlite3.SQLITE_DBCONFIG_LEGACY_FILE_FORMAT
sqlite3.SQLITE_DBCONFIG_NO_CKPT_ON_CLOSE
sqlite3.SQLITE_DBCONFIG_RESET_DATABASE
sqlite3.SQLITE_DBCONFIG_TRIGGER_EQP
sqlite3.SQLITE_DBCONFIG_TRUSTED_SCHEMA
sqlite3.SQLITE_DBCONFIG_WRITABLE_SCHEMA

Estas constantes se utilizan para los métodos `Connection.setconfig()` y `getconfig()`.

La disponibilidad de estas constantes varía según la versión de SQLite con la que se compiló Python.

Added in version 3.12.

Ver también

https://www.sqlite.org/c3ref/c_dbconfig_defensive.html
Documentación de SQLite: Opciones de configuración de la conexión a la base de datos

Deprecated since version 3.12, removed in version 3.14: The **`version`** and **`version_info`** constants.

Objetos de conexión

class sqlite3.Connection

Cada base de datos SQLite abierta es representada por un objeto `Connection`, el cual se crea usando **`sqlite3.connect()`**. Su objetivo principal es crear objetos **`Cursor`**, y Control transaccional.

Ver también

- Cómo utilizar los métodos de acceso directo de conexión
- Como usar la conexión con un administrador de contexto

Distinto en la versión 3.13: Se emite un **`ResourceWarning`** si no se llama a **`close()`** antes de eliminar un objeto **`Connection`**.

Una conexión a una base de datos SQLite tiene los siguientes atributos y métodos:

cursor(*factory=Cursor*)

Crea y devuelve un objeto `Cursor`. El método del cursor acepta un único parámetro opcional, *factory*. Si se proporciona, debe ser un callable que devuelva una instancia de `Cursor` o sus subclases.

blobopen(*table*, *column*, *rowid*, */*, ***, *readonly=False*, *name='main'*)

Abre una `Blob` para manejar un BLOB existente.

Parámetros:

- **table** (*str*) – El nombre de la tabla donde el *blob* está ubicado.
- **column** (*str*) – El nombre de la columna donde el *blob* está ubicado.
- **rowid** (*int*) – The row id where the blob is located.
- **readonly** (*bool*) – Se define como `True` si el *blob* deberá ser abierto sin permisos de escritura. El valor por defecto es `False`.
- **name** (*str*) – El nombre de la base de datos donde el *blob* está ubicado. El valor por defecto es `"main"`.

Muestra:

OperationalError – Cuando se intenta abrir un *blob* en una tabla `WITHOUT ROWID`.

Tipo del valor devuelto:

Blob

Nota

El tamaño de un *blob* no puede ser cambiado usando la clase `Blob`. Usa la función de SQL `zeroblob` para crear un *blob* con un tamaño fijo.

Added in version 3.11.

commit()

Confirma cualquier transacción pendiente en la base de datos. Si `autocommit` es `True` o no hay ninguna transacción abierta, este método no hace nada. Si **`autocommit`** es `False`, se abre implícitamente una nueva transacción si se confirmó una transacción pendiente con este método.

rollback()

Revertir al inicio de cualquier transacción pendiente. Si `autocommit` es `True` o no hay ninguna transacción abierta, este método no hace nada. Si **`autocommit`** es `False`, se abre implícitamente una nueva transacción si se revirtió una transacción pendiente con este método.

close()

Cierre la conexión a la base de datos. Si `autocommit` es `False`, cualquier transacción pendiente se revierte implícitamente. Si **`autocommit`** es `True` o `LEGACY_TRANSACTION_CONTROL`, no se ejecuta ningún control de transacción implícito. Asegúrese de ejecutar `commit()` antes de cerrar para evitar perder los cambios pendientes.

execute(*sql*, *parameters=()*, */*)

Crea un nuevo objeto `Cursor` y llama `execute()` con los parámetros y el SQL dados. Retorna el nuevo objeto cursor.

executemany(*sql*, *parameters*, */*)

Crea una nueva `Cursor` object and call `executemany()` con los parámetros y el SQL dados. Retorna el nuevo objeto cursor.

executescript(*sql_script*, */*)

Crea una nueva `Cursor` object and call `executescript()` con el *sql_script* dado. Retorna el nuevo objeto cursor.

create_function(*name*, *narg*, *func*, ***, *deterministic=False*)

Crea o elimina una función SQL definida por el usuario.

Parámetros:

- **name** (*str*) – El nombre de la función SQL.
- **narg** (*int*) – El número de argumentos que la función SQL puede aceptar. Si es `-1`, podrá entonces recibir cualquier cantidad de argumentos.
- **func** (callback | None) – Un callable que se llama cuando se invoca la función SQL. El invocable debe devolver a type natively supported by SQLite. Establézcalo en `None` para eliminar una función SQL existente.
- **deterministic** (*bool*) – Si se establece `True`, la función SQL creada se marcará como determinista [https://sqlite.org/deterministic.html], lo cual permite a SQLite realizar optimizaciones adicionales.

Distinto en la versión 3.8: Se agregó el parámetro *deterministic*.

Ejemplo:

```
>>> import hashlib
>>> def md5sum(t):
...     return hashlib.md5(t).hexdigest()
>>> con = sqlite3.connect(":memory:")
>>> con.create_function("md5", 1, md5sum)
>>> for row in con.execute("SELECT md5(?)", (b'
...     print(row)
('acbd18db4cc2f85cedef654fccc4a4d8',)
>>> con.close()
```

Distinto en la versión 3.13: El uso de *name*, *narg* y *func* como argumentos de palabras clave ya no es recomendable. Estos parámetros pasarán a ser solo posicionales en Python 3.15.

create_aggregate(*name*, *n_arg*, *aggregate_class*)

Crea o elimina una función agregada SQL definida por el usuario.

Parámetros:

- **name** (*str*) – El nombre de la función agregada SQL.
- **n_arg** (*int*) – El número de argumentos que la función agregada SQL puede aceptar. Si es `-1`, podrá entonces recibir cualquier cantidad de argumentos.
- **aggregate_class** (class | None) – Una clase debe implementar los siguientes métodos: * `step()`: Adiciona una fila al *aggregate*. * `finalize()`: Retorna el resultado final del *aggregate* como un tipo soportado nativamente por SQLite. La cantidad de argumentos que el método `step()` puede aceptar, es controlado por *n_arg*. Establece `None` para eliminar una función agregada SQL existente.

Ejemplo:

```
class MySum:
    def __init__(self):
        self.count = 0

    def step(self, value):
        self.count += value

    def finalize(self):
        return self.count

con = sqlite3.connect(":memory:")
con.create_aggregate("mysum", 1, MySum)
cur = con.execute("CREATE TABLE test(i)")
cur.execute("INSERT INTO test(i) VALUES(1)")
cur.execute("INSERT INTO test(i) VALUES(2)")
cur.execute("SELECT mysum(i) FROM test")
print(cur.fetchone()[0])

con.close()
```

Distinto en la versión 3.13: El uso de *name*, *n_arg* y *aggregate_class* como argumentos de palabras clave ya no es recomendable. Estos parámetros pasarán a ser solo posicionales en Python 3.15.

create_window_function(*name*, *num_params*, *aggregate_class*, */*)

Crea o elimina una función agregada de ventana definida por el usuario.

Parámetros:

- **name** (*str*) – El nombre de la función agregada de ventana a ser creada o eliminada.
- **num_params** (*int*) – La cantidad de argumentos que la función agregada de ventana SQL acepta. Si es `-1`, podrá entonces recibir cualquier cantidad de argumentos.
- **aggregate_class** (class | None) – Una clase debe implementar los

siguientes métodos: * `step()`: Agrega una fila a la ventana actual. * `value()`: Retorna el valor actual del *aggregate* . * `inverse()`: Elimina una fila de la ventana actual. * `finalize()`: Retorna el resultado final del *aggregate* como una un tipo soportado nativamente por SQLite. La cantidad de argumentos que los métodos `step()` y `value()` pueden aceptar son controlados por *num_params*. Establece `None` para eliminar una función agregada de ventana SQL.

Muestra:

NotSupportedError – Si es usado con una versión de SQLite inferior a 3.25.0, la cual no soporte funciones agregadas de ventana.

Added in version 3.11.

Ejemplo:

```
# Example taken from https://www.sqlite.org/wir
class WindowSumInt:
    def __init__(self):
        self.count = 0

    def step(self, value):
        """Add a row to the current window."""
        self.count += value

    def value(self):
        """Return the current value of the aggr
        return self.count

    def inverse(self, value):
```

```
    """Remove a row from the current windo
    self.count -= value

  def finalize(self):
    """Return the final value of the aggre

    Any clean-up actions should be placed
    """
    return self.count


con = sqlite3.connect(":memory:")
cur = con.execute("CREATE TABLE test(x, y)")
values = [
  ("a", 4),
  ("b", 5),
  ("c", 3),
  ("d", 8),
  ("e", 1),
]
cur.executemany("INSERT INTO test VALUES(?, ?)"
con.create_window_function("sumint", 1, Window
cur.execute("""
  SELECT x, sumint(y) OVER (
    ORDER BY x ROWS BETWEEN 1 PRECEDING AN
  ) AS sum_y
  FROM test ORDER BY x
""")
print(cur.fetchall())
con.close()
```

create_collation(*name*, *callable*, */*)

Crea una colación nombrada *name* usando la función *collating callable*. A *callable* se le pasan dos argumentos **`string`**, y este debería retornar un **`entero`**:

- `1` si el primero es ordenado más alto que el segundo

- `-1` si el primero es ordenado más pequeño que el segundo
- `0` si están ordenados de manera igual

El siguiente ejemplo muestra una ordenación de colación inversa:

```
def collate_reverse(string1, string2):
    if string1 == string2:
        return 0
    elif string1 < string2:
        return 1
    else:
        return -1

con = sqlite3.connect(":memory:")
con.create_collation("reverse", collate_reverse

cur = con.execute("CREATE TABLE test(x)")
cur.executemany("INSERT INTO test(x) VALUES(?)'
cur.execute("SELECT x FROM test ORDER BY x COLI
for row in cur:
    print(row)
con.close()
```

Elimina una función de colación al establecer el *callable* como `None`.

Distinto en la versión 3.11: El nombre de la colación puede contener cualquier caracter Unicode. Anteriormente, solamente caracteres ASCII eran permitidos.

interrupt()

Llame a este método desde un subproceso diferente para cancelar cualquier consulta que pueda estar ejecutándose en la conexión. Las consultas canceladas generarán un error **`OperationalError`**.

set_authorizer(*authorizer_callback*)

Registre callable *authorizer_callback* para que se invoque cada vez que se intente acceder a una columna de una tabla en la base de datos. La devolución de llamada debe devolver uno de los siguientes: **`SQLITE_OK`**, **`SQLITE_DENY`** o **`SQLITE_IGNORE`** para indicar cómo debe gestionar la biblioteca SQLite subyacente el acceso a la columna.

El primer argumento del callback significa que tipo de operación será autorizada. El segundo y tercer argumento serán argumentos o `None` dependiendo del primer argumento. El cuarto argumento es el nombre de la base de datos («main», «temp», etc.) si aplica. El quinto argumento es el nombre del disparador más interno o vista que es responsable por los intentos de acceso o `None` si este intento de acceso es directo desde el código SQL de entrada.

Por favor consulte la documentación de SQLite sobre los posibles valores para el primer argumento y el significado del segundo y tercer argumento dependiendo del primero. Todas las constantes necesarias están disponibles en el módulo **`sqlite3`**.

Pasando `None` como *authorizer_callback* deshabilitará el autorizador.

Distinto en la versión 3.11: Agregado soporte para deshabilitar el autorizador usando `None`.

Distinto en la versión 3.13: El uso de *authorizer_callback* como argumento de palabra clave ya no es recomendable. El parámetro pasará a ser solo posicional en Python 3.15.

set_progress_handler(*progress_handler*, *n*)

Registre callable *progress_handler* para que se invoque para cada instrucción *n* de la máquina virtual SQLite. Esto es útil si desea recibir llamadas de SQLite durante operaciones de larga duración, por ejemplo, para actualizar una GUI.

Si se desea limpiar cualquier *progress_handler* instalado anteriormente, llame el método con `None` para *progress_handler*.

Si se devuelve un valor distinto de cero desde la función del controlador, se finalizará la consulta que se está ejecutando actualmente y se generará una excepción **`DatabaseError`**.

Distinto en la versión 3.13: El uso de *progress_handler* como argumento de palabra clave ya no es recomendable. El parámetro pasará a ser solo posicional en Python 3.15.

set_trace_callback(*trace_callback*)

Registre callable *trace_callback* para que se invoque para cada declaración SQL que realmente ejecute el backend de SQLite.

El único argumento pasado a la retrollamada es la declaración (como **`str`**) que se está ejecutando. El valor de retorno de la retrollamada es ignorado. Tenga en cuenta que el backend no solo ejecuta declaraciones pasadas a los métodos **`Cursor.execute()`**. Otras fuentes incluyen el transaction management del módulo **`sqlite3`** y la ejecución de disparadores definidos en la base de datos actual.

Pasando `None` como *trace_callback* deshabilitará el *trace callback*.

Nota

Las excepciones que se producen en la llamada de retorno no se propagan. Como ayuda para el desarrollo y la depuración, utilice **`enable_callback_tracebacks()`** para habilitar la impresión de *tracebacks* de las excepciones que se producen en la llamada de retorno.

Added in version 3.3.

Distinto en la versión 3.13: El uso de *trace_callback* como argumento de palabra clave ya no es recomendable. El parámetro pasará a ser solo posicional en Python 3.15.

enable_load_extension(*enabled*, */*)

Habilita el motor de SQLite para cargar extensiones SQLite de bibliotecas compartidas si *enabled* se establece como `True`; sino deshabilitará la carga de extensiones SQLite. Las extensiones SQLite pueden definir implementaciones de nuevas funciones, agregaciones, o tablas virtuales enteras. Una extensión bien conocida es *fulltext-search* distribuida con SQLite.

Nota

El módulo **`sqlite3`** no está construido con soporte de extensión cargable de forma predeterminada, porque algunas plataformas (especialmente macOS) tienen bibliotecas SQLite que se compilan sin esta función. Para obtener soporte para extensiones cargables, debe pasar la opción **`--enable-loadable-sqlite-extensions`** a **configure**.

Lanza un auditing event `sqlite3.enable_load_extension` con los argumentos `connection`, `enabled`.

Added in version 3.2.

Distinto en la versión 3.10: Agregado el evento de auditoría `sqlite3.enable_load_extension`.

```
con.enable_load_extension(True)
```

```
# Load the fulltext search extension
con.execute("select load_extension('./fts3.so')

# alternatively you can load the extension usir
# con.load_extension("./fts3.so")

# disable extension loading again
con.enable_load_extension(False)

# example from SQLite wiki
con.execute("CREATE VIRTUAL TABLE recipe USING
con.executescript("""
    INSERT INTO recipe (name, ingredients) VALU
    INSERT INTO recipe (name, ingredients) VALU
    INSERT INTO recipe (name, ingredients) VALU
    INSERT INTO recipe (name, ingredients) VALU
    """)
for row in con.execute("SELECT rowid, name, ing
    print(row)
```

load_extension(*path*, /, *, *entrypoint=None*)

Cargue una extensión SQLite desde una biblioteca compartida. Habilite la carga de extensiones con **`enable_load_extension()`** antes de llamar a este método.

Parámetros:

- **path** (*str*) – La ruta a la extensión SQLite.
- **entrypoint** (*str* | *None*) – Nombre del punto de entrada. Si es `None` (el valor predeterminado), SQLite creará un nombre de punto de entrada propio; consulte la documentación de SQLite Loading an Extension [https://www.sqlite.org/loadext.html#loading_an_extension] para obtener más detalles.

Lanza un auditing event `sqlite3.load_extension` con argumentos `connection`, `path`.

Added in version 3.2.

Distinto en la versión 3.10: Agregado el evento de auditoría `sqlite3.load_extension`.

Distinto en la versión 3.12: Se agregó el parámetro *entrypoint*.

iterdump(**, filter=None*)

Retorna un iterator para volcar la base de datos en un texto de formato SQL. Es útil cuando guardamos una base de datos en memoria para una posterior restauración. Esta función provee las mismas capacidades que el comando `.dump` en el *shell* **sqlite3**.

Parámetros:

- **filter** (*str* | *None*) – Un patrón `LIKE` opcional para volcar objetos de la base de datos, p. ej., `prefix_%`. Si se usa `None` (el valor predeterminado), se incluirán todos los objetos de la base de datos.

Ejemplo:

```
# Convert file example.db to SQL dump file dump
con = sqlite3.connect('example.db')
with open('dump.sql', 'w') as f:
    for line in con.iterdump():
        f.write('%s\n' % line)
con.close()
```

Ver también

Cómo manejar codificaciones de texto que no sean UTF-8

Distinto en la versión 3.13: Se agregó el parámetro *filter*.

backup(*target*, ***, *pages=-1*, *progress=None*, *name='main'*, *sleep=0.250*)

Crea una copia de seguridad de la base de datos SQLite.

Funciona incluso si la base de datos está siendo accedida por otros clientes al mismo tiempo sobre la misma conexión.

Parámetros:

- **target** (*Connection*) – La conexión de la base de datos para guardar la copia de seguridad.
- **pages** (*int*) – Cantidad de páginas que serán copiadas cada vez. Si es menor o igual a `0`, toda la base de datos será copiada en un solo paso. El valor por defecto es `-1`.
- **progress** (callback | None) – Si se configura en callable, se invoca con tres argumentos enteros para cada iteración de copia de seguridad: el *status* de la última iteración, el *remaining* número de páginas que aún se deben copiar y el *total* número de páginas. El valor predeterminado es `None`.
- **name** (*str*) – El nombre de la base de datos a ser guardada. O bien sea `"main"` (valor por defecto) para la base de datos *main*, `"temp"` para la base de datos temporal, o el nombre de una base de datos personalizada adjuntada usando la sentencia `ATTACH DATABASE`.

- **sleep** (*float*) – Número de segundos a dormir entre sucesivos intentos para respaldar páginas restantes.

Ejemplo 1, copiar una base de datos existente en otra:

```
def progress(status, remaining, total):
    print(f'Copied {total-remaining} of {total

src = sqlite3.connect('example.db')
dst = sqlite3.connect('backup.db')
with dst:
    src.backup(dst, pages=1, progress=progress)
dst.close()
src.close()
```

Ejemplo 2: copiar una base de datos existente en una copia transitoria:

```
src = sqlite3.connect('example.db')
dst = sqlite3.connect(':memory:')
src.backup(dst)
dst.close()
src.close()
```

Added in version 3.7.

Ver también

Cómo manejar codificaciones de texto que no sean UTF-8

getlimit(*category*, */*)

Obtiene el límite de tiempo de ejecución de una conexión.

Parámetros:

category (*int*) – La SQLite limit category [https://www.sqlite.org/c3ref/c_limit_attached.html] a ser consultada.

Tipo del valor devuelto:

int

Muestra:

ProgrammingError – Si *category* no se reconoce por las capas inferiores de la biblioteca SQLite.

Ejemplo, consulta el tamaño máximo de una sentencia SQL para la **`Connection`** `con` (el valor por defecto es 1000000000):

```
>>> con.getlimit(sqlite3.SQLITE_LIMIT_SQL_LENGT
1000000000
```

Added in version 3.11.

setlimit(*category*, *limit*, */*)

Establezca un límite de tiempo de ejecución de la conexión. Los intentos de aumentar un límite por encima de su límite superior estricto se truncan de forma silenciosa hasta el límite superior estricto. Independientemente de si se modificó o no el límite, se devuelve el valor anterior del límite.

Parámetros:

- **category** (*int*) – La SQLite limit category [https://www.sqlite.org/c3ref/c_limit_attached.html] a ser establecido.
- **limit** (*int*) – El valor del nuevo límite. Si es negativo, el límite actual no cambia.

Tipo del valor devuelto:

int

Muestra:

ProgrammingError – Si *category* no se reconoce por las capas inferiores de la biblioteca SQLite.

Por ejemplo, limite el número de bases de datos adjuntas a 1 para la **`Connection`** `con` (el valor por defecto es 10):

```
>>> con.setlimit(sqlite3.SQLITE_LIMIT_ATTACHED,
10
>>> con.getlimit(sqlite3.SQLITE_LIMIT_ATTACHED)
1
```

Added in version 3.11.

getconfig(*op*, */*)

Consultar una opción de configuración de conexión booleana.

Parámetros:
op (*int*) – Un SQLITE_DBCONFIG code.

Tipo del valor devuelto:
bool

Added in version 3.12.

setconfig(*op*, *enable=True*, */*)

Establezca una opción de configuración de conexión booleana.

Parámetros:

- **op** (*int*) – Un SQLITE_DBCONFIG code.
- **enable** (*bool*) – `True` si la opción de configuración debe estar habilitada (predeterminado); `False` si debe estar deshabilitada.

Added in version 3.12.

serialize(*, *name='main'*)

Serializa la base de datos en un objeto `bytes`. Para un archivo ordinario de base de datos en disco, la serialización es solo una copia del archivo de disco. Para el caso de una base de datos en memoria o una base de datos «temp», la serialización es la misma secuencia de bytes que se escribiría en el disco si se hiciera una copia de seguridad de esa base de datos en el disco.

Parámetros:

name (*str*) – El nombre de la base de datos a ser serializada. El valor por defecto es `"main"`.

Tipo del valor devuelto:

bytes

Nota

Este método solo está disponible si las capas internas de la biblioteca SQLite tienen la API *serialise*.

Added in version 3.11.

deserialize(*data*, */*, *, *name='main'*)

Deserializa una base de datos `serializsda` en una clase `Connection`. Este método hace que la conexión de base de datos se desconecte de la base de datos *name*, y la abre nuevamente como una base de datos en memoria, basada en la serialización contenida en *data*.

Parámetros:

- **data** (*bytes*) – Una base de datos serializada.
- **name** (*str*) – El nombre de la base de datos a ser deserializada. El valor por defecto es `"main"`.

Muestra:

- **OperationalError** – Si la

conexión con la base de datos está actualmente involucrada en una transacción de lectura o una operación de copia de seguridad.
- **DatabaseError** – Si *data* no contiene una base de datos SQLite válida.
- **OverflowError** – Si `len(data)` es más grande que `2**63 - 1`.

Nota

Este método solo está disponible si las capas internas de la biblioteca SQLite tienen la API *deserialize*.

Added in version 3.11.

autocommit

Este atributo controla el comportamiento de las transacciones conforme a **PEP 249** [https://peps.python.org/pep-0249/]. **`autocommit`** tiene tres valores permitidos:

- `False`: Seleccione un comportamiento de transacción compatible con **PEP 249** [https://peps.python.org/pep-0249/], lo que implica que **`sqlite3`** garantiza que una transacción esté siempre abierta. Utilice `commit()` y `rollback()` para cerrar transacciones.

  Este es el valor recomendado de **`autocommit`**.

- `True`: Utilice autocommit mode [https://www.sqlite.org/lang_transaction.html#implicit_versus_explicit_transactions] de SQLite. `commit()` y `rollback()` no tienen efecto en este modo.

- `LEGACY_TRANSACTION_CONTROL`: Control de transacciones anterior a Python 3.12 (no compatible con **PEP 249** [https://peps.python.org/

pep-0249/]). Consulte **`isolation_level`** para obtener más detalles.

Este es actualmente el valor predeterminado de **`autocommit`**.

Cambiar **`autocommit`** a `False` abrirá una nueva transacción y cambiarlo a `True` confirmará cualquier transacción pendiente.

Consulte Control de transacciones mediante el atributo autocommit para obtener más detalles.

Nota

El atributo **`isolation_level`** no tiene efecto a menos que **`autocommit`** sea **`LEGACY_TRANSACTION_CONTROL`**.

Added in version 3.12.

in_transaction

Este atributo de solo lectura corresponde al autocommit mode [https://www.sqlite.org/lang_transaction.html#implicit_versus_explicit_transactions] de SQLite de bajo nivel.

`True` si una transacción está activa (existen cambios sin confirmar),``False`` en caso contrario.

Added in version 3.2.

isolation_level

Controla el legacy transaction handling mode de **`sqlite3`**. Si se configura en `None`, las transacciones nunca se abren de forma implícita. Si se configura en uno de los valores `"DEFERRED"`, `"IMMEDIATE"` o `"EXCLUSIVE"`, correspondiente al SQLite transaction behaviour [https://www.sqlite.org/lang_transaction.html#deferred_immediate_and_exclusive_transactions] subyacente, se ejecuta implicit transaction management.

Si no se sobreescribe por el parámetro *isolation_level* de **`connect()`**, el valor predeterminado es `""`, que es un alias para `"DEFERRED"`.

Nota

Se recomienda utilizar **`autocommit`** para controlar el manejo de transacciones en lugar de **`isolation_level`**. **`isolation_level`** no tiene efecto a menos que **`autocommit`** se configure en **`LEGACY_TRANSACTION_CONTROL`** (el valor predeterminado).

row_factory

El **`row_factory`** inicial para los objetos **`Cursor`** creados a partir de esta conexión. La asignación a este atributo no afecta al **`row_factory`** de los cursores existentes que pertenecen a esta conexión, solo a los nuevos. Es `None` de forma predeterminada, lo que significa que cada fila se devuelve como **`tuple`**.

Consulte Cómo crear y utilizar fábricas de filas para obtener más detalles.

text_factory

Un callable que acepta un parámetro **`bytes`** y devuelve una representación de texto del mismo. El objeto invocable se invoca para valores SQLite con el tipo de datos `TEXT`. De manera predeterminada, este atributo está configurado en **`str`**.

Consulte Cómo manejar codificaciones de texto que no sean UTF-8 para obtener más detalles.

total_changes

Retorna el número total de filas de la base de datos que han sido modificadas, insertadas o borradas desde que la conexión a la base de datos fue abierta.

Objetos cursor

Un objeto `Cursor` representa un database cursor [https://en.wikipedia.org/wiki/Cursor_(databases)] que se utiliza para ejecutar sentencias SQL y administrar el contexto de una operación de búsqueda. Los cursores son creados usando **`Connection.cursor()`**, o por usar alguno de los connection shortcut methods.

Los objetos cursores son iterators, lo que significa que si **`execute()`** una consulta `SELECT`, simplemente podrás iterar sobre el cursor para obtener las filas resultantes:

```
for row in cur.execute("SELECT t FROM data"):
    print(row)
```

class sqlite3.Cursor

Una instancia **`Cursor`** tiene los siguientes atributos y métodos.

execute(*sql*, *parameters=()*, */*)

Ejecuta una única declaración SQL, vinculando opcionalmente valores de Python mediante placeholders.

Parámetros:

- **sql** (*str*) – Una sola declaración SQL.
- **parameters** (**`dict`** | sequence) – Valores de Python para vincular a marcadores de posición en *sql*. Un **`dict`** si se utilizan marcadores de posición con

nombre. Un sequence si se utilizan marcadores de posición sin nombre. Consulte Cómo usar marcadores de posición para vincular valores en consultas SQL.

Muestra:

ProgrammingError – When *sql* contains more than one SQL statement. When named placeholders are used and *parameters* is a sequence instead of a `dict`.

Si `autocommit` es `LEGACY_TRANSACTION_CONTROL`, `isolation_level` no es `None`, *sql* es una declaración `INSERT`, `UPDATE`, `DELETE` o `REPLACE` y no hay ninguna transacción abierta, se abre implícitamente una transacción antes de ejecutar *sql*.

Distinto en la versión 3.14: `ProgrammingError` is emitted if named placeholders are used and *parameters* is a sequence instead of a `dict`.

Utilice `executescript()` para ejecutar múltiples declaraciones SQL.

executemany(*sql*, *parameters*, */*)

Para cada elemento en *parameters*, ejecute repetidamente la instrucción SQL parameterized DML *sql*.

Utiliza el mismo manejo de transacciones implícitas que `execute()`.

Parámetros:

- **sql** (*str*) – Una sola declaración DML de SQL.
- **parameters** (iterable) – Un

iterable de parámetros para vincular con los marcadores de posición en *sql*. Consulte Cómo usar marcadores de posición para vincular valores en consultas SQL.

Muestra:

ProgrammingError – When *sql* contains more than one SQL statement or is not a DML statement, When named placeholders are used and the items in *parameters* are sequences instead of `dict`s.

Ejemplo:

```
rows = [
    ("row1",),
    ("row2",),
]
# cur is an sqlite3.Cursor object
cur.executemany("INSERT INTO data VALUES(?)", r
```

Nota

Se descartan todas las filas resultantes, incluidas las declaraciones DML con RETURNING clauses [https://www.sqlite.org/lang_returning.html].

Distinto en la versión 3.14: `ProgrammingError` is emitted if named placeholders are used and the items in *parameters* are sequences instead of `dict`s.

executescript(*sql_script*, */*)

Ejecute las sentencias SQL en *sql_script*. Si `autocommit` es `LEGACY_TRANSACTION_CONTROL` y hay una transacción pendiente, primero se ejecuta una sentencia `COMMIT` implícita. No se realiza ningún otro control de transacción implícito; cualquier control de transacción debe agregarse a *sql_script*.

sql_script debe ser una instancia de **`string`**.

Ejemplo:

```
# cur is an sqlite3.Cursor object
cur.executescript("""
    BEGIN;
    CREATE TABLE person(firstname, lastname, ag
    CREATE TABLE book(title, author, published)
    CREATE TABLE publisher(name, address);
    COMMIT;
""")
```

fetchone()

Si **`row_factory`** es `None`, devuelve el siguiente conjunto de resultados de consulta de fila como **`tuple`**. De lo contrario, páselo a la fábrica de filas y devuelve su resultado. Devuelve `None` si no hay más datos disponibles.

fetchmany(*size=cursor.arraysize*)

Retorna el siguiente conjunto de filas del resultado de una consulta como una **`list`**. Una lista vacía será retornada cuando no hay más filas disponibles.

El número de filas que se van a obtener por llamada se especifica mediante el parámetro *size*. Si *size* no es informado, entonces **`arraysize`** determinará el número de filas que se van a recuperar. Si hay menos filas *size* disponibles, se retornan tantas filas como estén disponibles.

Nótese que hay consideraciones de desempeño involucradas con el parámetro *size*. Para un optimo desempeño, es usualmente mejor usar el atributo *arraysize*. Si el parámetro *size* es usado, entonces es mejor retener el mismo valor de una llamada `fetchmany()` a la siguiente.

Distinto en la versión 3.14.1: Negative *size* values are rejected by raising `ValueError`.

fetchall()

Retorna todas las filas (restantes) de un resultado de consulta como `list`. Retorna una lista vacía si no hay filas disponibles. Tenga en cuenta que el atributo `arraysize` puede afectar al rendimiento de esta operación.

close()

Cierra el cursor ahora (en lugar que cuando `__del__` es llamado)

El cursor no será usable de este punto en adelante; una excepción `ProgrammingError` será lanzada si se intenta cualquier operación con el cursor.

setinputsizes(*sizes*, */*)

Requerido por la DB-API. No hace nada en **`sqlite3`**.

setoutputsize(*size*, *column=None*, */*)

Requerido por la DB-API. No hace nada en **`sqlite3`**.

arraysize

Atributo de lectura/escritura que controla el número de filas retornadas por `fetchmany()`. El valor por defecto es 1, lo cual significa que una única fila será obtenida por llamada.

Distinto en la versión 3.14.1: Negative values are rejected by raising `ValueError`.

connection

Este atributo de solo lectura que provee la **`Connection`** de la base de datos SQLite pertenece al cursor. Un objeto **`Cursor`** creado llamando a **`con.cursor()`** tendrá un atributo **`connection`** que hace referencia a *con*:

```
>>> con = sqlite3.connect(":memory:")
>>> cur = con.cursor()
>>> cur.connection == con
True
>>> con.close()
```

description

Este atributo de solo lectura provee el nombre de las columnas de la última consulta. Para seguir siendo compatible con la API de base de datos de Python, retornará una tupla de 7 elementos para cada columna donde los últimos seis elementos de cada tupla son `Ninguno`.

También es configurado para sentencias `SELECT` sin ninguna fila coincidente.

lastrowid

Atributo de solo lectura que proporciona el identificador de fila de la última insertada. Solo se actualiza después de que las sentencias `INSERT` o `REPLACE` hayan sido exitosas usando el método **`execute()`**. Para otras instrucciones, después de **`executemany()`** o **`executescript()`**, o si se produjo un error en la inserción, el valor de `lastrowid` se deja sin cambios. El valor inicial de `lastrowid` es `None`.

Nota

Las inserciones en tablas `WITHOUT ROWID` no se registran.

Distinto en la versión 3.6: Se agregó soporte para sentencias `REPLACE`.

rowcount

Atributo de solo lectura que proporciona la cantidad de filas modificadas para las instrucciones `INSERT`, `UPDATE`, `DELETE` y `REPLACE`; es `-1` para otras instrucciones, incluidas las consultas CTE (Common Table Expression) . Solo se actualiza mediante los métodos **`execute()`** y **`executemany()`**, después de que la instrucción se haya ejecutado hasta su finalización. Esto significa que se deben obtener todas las filas resultantes para que se actualice **`rowcount`**.

row_factory

Controla cómo se representa una fila obtenida de este **`Cursor`**. Si es `None`, una fila se representa como **`tuple`**. Se puede configurar como **`sqlite3.Row`** incluido; o como callable que acepta dos argumentos, un objeto **`Cursor`** y el **`tuple`** de valores de fila, y devuelve un objeto personalizado que representa una fila de SQLite.

El valor predeterminado es el valor que se estableció en **`Connection.row_factory`** cuando se creó **`Cursor`**. La asignación a este atributo no afecta a **`Connection.row_factory`** de la conexión principal.

Consulte Cómo crear y utilizar fábricas de filas para obtener más detalles.

Objetos fila (*Row*)

class sqlite3.Row

Una instancia de **`Row`** sirve como una instancia **`row_factory`** altamente optimizada para objetos **`Connection`**. Admite iteración, pruebas de igualdad, acceso a **`len()`** y mapping por nombre de columna e índice.

Dos objetos `Row` se consideran iguales si tienen nombres de columna y valores idénticos.

Consulte Cómo crear y utilizar fábricas de filas para obtener más detalles.

keys()

Este método retorna una `list` con los nombre de columnas como `strings`. Inmediatamente después de una consulta, es el primer miembro de cada tupla en `Cursor.description`.

Distinto en la versión 3.5: Agrega soporte para segmentación.

Objetos fila (*Row*)

class sqlite3.Blob

Added in version 3.11.

Una instancia `Blob` es un file-like object que puede leer y escribir datos en un SQLite BLOB. Llame a `len(blob)` para obtener el tamaño (número de bytes) del blob. Use índices y slices para obtener acceso directo a los datos del blob.

Use `Blob` como context manager para asegurarse de que el identificador de blob se cierra después de su uso.

```
con = sqlite3.connect(":memory:")
con.execute("CREATE TABLE test(blob_col blob)")
con.execute("INSERT INTO test(blob_col) VALUES(zero

# Write to our blob, using two write operations:
with con.blobopen("test", "blob_col", 1) as blob:
    blob.write(b"hello, ")
    blob.write(b"world.")
    # Modify the first and last bytes of our blob
    blob[0] = ord("H")
    blob[-1] = ord("!")
```

```
# Read the contents of our blob
with con.blobopen("test", "blob_col", 1) as blob:
    greeting = blob.read()

print(greeting)  # outputs "b'Hello, world!'"
con.close()
```

close()

Cierra el *blob*.

El cursor no será usable de este punto en adelante; una excepción **`Error`** (o subclase) será lanzada si se intenta cualquier operación con el cursor.

read(*length=-1, /*)

Leer bytes *length* de datos del blob en la posición de desplazamiento actual. Si se alcanza el final del blob, se devolverán los datos hasta EOF. Cuando *length* no se especifica, o es negativo, **`read()`** se leerá hasta el final del blob.

write(*data, /*)

Escriba *data* en el blob en el desplazamiento actual. Esta función no puede cambiar la longitud del blob. Escribir más allá del final del blob generará un **`ValueError`**.

tell()

Devolver la posición de acceso actual del blob.

seek(*offset, origin=os.SEEK_SET, /*)

Establezca la posición de acceso actual del blob en *offset*. El argumento *origin* tiene como valor predeterminado **`os.SEEK_SET`** (posición absoluta del blob). Otros valores para *origin* son **`os.SEEK_CUR`** (búsqueda relativa a la posición actual) y **`os.SEEK_END`** (búsqueda relativa al final del blob).

Objetos PrepareProtocol

class sqlite3.PrepareProtocol

El único propósito del tipo *PrepareProtocol* es actuar como un protocolo de adaptación de estilo **PEP 246** [https://peps.python.org/pep-0246/] para objetos que pueden adapt themselves a native SQLite types.

Excepciones

La jerarquía de excepciones está definida por DB-API 2.0 (**PEP 249** [https://peps.python.org/pep-0249/]).

exception sqlite3.Warning

Esta excepción no se genera actualmente en el módulo **`sqlite3`**, pero puede ser provocada por aplicaciones que usan :mod:!sqlite3`, por ejemplo, si una función definida por el usuario trunca datos durante la inserción. `Warning` es una subclase de **`Exception`**.

exception sqlite3.Error

La clase base de las otras excepciones de este módulo. Use esto para detectar todos los errores con una sola instrucción **`except`**.``Error`` is una subclase de **`Exception`**.

Si la excepción se originó dentro de la biblioteca SQLite, se agregan los siguientes dos atributos a la excepción:

sqlite_errorcode

El código de error numérico de SQLite API [https://sqlite.org/rescode.html]

Added in version 3.11.

sqlite_errorname

El nombre simbólico del código de error numérico de SQLite API [https://sqlite.org/rescode.html]

Added in version 3.11.

exception sqlite3.InterfaceError

Excepción lanzada por el uso indebido de la API de SQLite C de bajo nivel. En otras palabras, si se lanza esta excepción, probablemente indica un error en el módulo **`sqlite3`**. `InterfaceError` es una subclase de **`Error`**.

exception sqlite3.DatabaseError

Excepción lanzada por errores relacionados con la base de datos. Esto sirve como excepción base para varios tipos de errores de base de datos. Solo se genera implícitamente a través de las subclases especializadas. `DatabaseError` es una subclase de **`Error`**.

exception sqlite3.DataError

Excepción lanzada por errores causados por problemas con los datos procesados, como valores numéricos fuera de rango y cadenas de caracteres demasiado largas. `DataError` es una subclase de **`DatabaseError`**.

exception sqlite3.OperationalError

Excepción lanzada por errores que están relacionados con el funcionamiento de la base de datos y no necesariamente bajo el control del programador. Por ejemplo, no se encuentra la ruta de la base de datos o no se pudo procesar una transacción. `OperationalError` es una subclase de **`DatabaseError`**.

exception sqlite3.IntegrityError

Excepción lanzada cuando la integridad de la base de datos es afectada, por ejemplo la comprobación de una llave foránea falla. Es una subclase de **`DatabaseError`**.

exception sqlite3.InternalError

Se genera una excepción cuando SQLite encuentra un error interno. Si se genera esto, puede indicar que hay un problema con la biblioteca SQLite en tiempo de ejecución. `InternalError` es una subclase de **`DatabaseError`**.

exception sqlite3.ProgrammingError

Excepción lanzada por errores de programación de la API de **`sqlite3`**, por ejemplo, proporcionar el número incorrecto de enlaces a una consulta, o intentar operar en una **`Connection`** cerrada. `ProgrammingError` es una subclase de **`DatabaseError`**.

exception sqlite3.NotSupportedError

Excepción lanzada en caso que la biblioteca SQLite subyacente no admita un método o una API de base de datos. Por ejemplo, establecer *determinista* como `Verdadero` en **`create_function()`**, si la biblioteca SQLite subyacente no admite funciones *deterministic*. `NotSupportedError` es una subclase de **`DatabaseError`**.

SQLite y tipos de Python

SQLite soporta de forma nativa los siguientes tipos: `NULL`, `INTEGER`, `REAL`, `TEXT`, `BLOB`.

Los siguientes tipos de Python se pueden enviar a SQLite sin problema alguno:

Tipo de Python	Tipo de SQLite
`None`	`NULL`
`int`	`INTEGER`
`float`	`REAL`
`str`	`TEXT`
`bytes`	`BLOB`

De esta forma es como los tipos de SQLite son convertidos a tipos de Python por defecto:

Tipo de SQLite	Tipo de Python
`NULL`	`None`
`INTEGER`	**`int`**
`REAL`	**`float`**
`TEXT`	depende de **`text_factory`**, **`str`** por defecto
`BLOB`	**`bytes`**

El sistema de tipos del módulo `sqlite3` es extensible en dos formas: se puede almacenar tipos de Python adicionales en una base de datos SQLite vía a objetos adaptadores, y se puede permitir que `sqlite3` convierta tipos SQLite a diferentes tipos de Python vía converters.

Adaptadores y convertidores predeterminados (en desuso)

Nota

Los adaptadores y convertidores predeterminados están obsoletos a partir de Python 3.12. En su lugar, utilice Ejemplos para adaptadores y convertidores y adáptelos a sus necesidades.

Los adaptadores y convertidores predeterminados obsoletos consisten en:

- Un adaptador para objetos `datetime.date` a `strings` en formato ISO 8601 [https://en.wikipedia.org/wiki/ISO_8601].
- Un adaptador para objetos `datetime.datetime` a cadenas en formato ISO 8601.
- Un convertidor de tipos «fecha» declared a objetos `datetime.date`.
- Un convertidor de tipos de «marca de tiempo» declarados a objetos `datetime.datetime`. Las partes fraccionarias se truncarán a 6 dígitos (precisión de microsegundos).

Nota

El convertidor por defecto «timestamp» ignora las compensaciones UTC en la base de datos y siempre devuelve un objeto `datetime.datetime` *naive*. Para conservar las compensaciones UTC en las marcas de tiempo, deje los convertidores deshabilitados o registre un convertidor que reconozca la compensación con `register_converter()`.

Obsoleto desde la versión 3.12.

Interfaz de línea de comandos

El módulo **`sqlite3`** se puede invocar como un script, utilizando el modificador **`-m`** del intérprete, para proporcionar un shell SQLite simple. La firma del argumento es la siguiente:

```
python -m sqlite3 [-h] [-v] [filename] [sql]
```

Escriba `.quit` o CTRL-D para salir del shell.

-h, --help

Ayuda de la CLI de impresión.

-v, --version

Imprima la versión de la biblioteca SQLite subyacente.

Added in version 3.12.

Guías prácticas

Cómo usar marcadores de posición para vincular valores en consultas SQL

Las operaciones SQL suelen necesitar utilizar valores de variables de Python. Sin embargo, tenga cuidado al utilizar las operaciones de cadena de Python para ensamblar consultas, ya que son vulnerables a SQL injection attacks [https://en.wikipedia.org/wiki/SQL_injection]. Por ejemplo, un atacante puede simplemente cerrar la comilla simple e inyectar `OR TRUE` para seleccionar todas las filas:

```
>>> # Never do this -- insecure!
>>> symbol = input()
' OR TRUE; --
>>> sql = "SELECT * FROM stocks WHERE symbol = '%s'" % s
>>> print(sql)
SELECT * FROM stocks WHERE symbol = '' OR TRUE; --'
>>> cur.execute(sql)
```

En su lugar, utilice la sustitución de parámetros de DB-API. Para insertar una variable en una cadena de consulta, utilice un marcador de posición en la cadena y sustituya los valores reales en la consulta proporcionándolos como un `tuple` de valores al segundo argumento del método `execute()` del cursor.

Una sentencia SQL puede utilizar uno de dos tipos de marcadores de posición: signos de interrogación (estilo qmark) o marcadores de posición con nombre (estilo con nombre). Para el estilo qmark, *parameters* debe ser un sequence cuya longitud debe coincidir con el número de marcadores de posición, o se genera un `ProgrammingError`. Para el estilo con nombre, *parameters* debe ser una instancia de un `dict` (o una subclase), que debe contener claves para todos los parámetros con nombre; se ignoran todos los elementos adicionales. A continuación, se muestra un ejemplo de ambos estilos:

```
con = sqlite3.connect(":memory:")
cur = con.execute("CREATE TABLE lang(name, first_appeare

# This is the named style used with executemany():
data = (
    {"name": "C", "year": 1972},
    {"name": "Fortran", "year": 1957},
    {"name": "Python", "year": 1991},
    {"name": "Go", "year": 2009},
)
cur.executemany("INSERT INTO lang VALUES(:name, :year)",

# This is the qmark style used in a SELECT query:
params = (1972,)
cur.execute("SELECT * FROM lang WHERE first_appeared = ?
print(cur.fetchall())
con.close()
```

Nota

Los marcadores de posición numéricos **PEP 249** [https://peps.python.org/pep-0249/] son compatibles con *not*. Si se utilizan, se interpretarán como marcadores de posición con nombre.

Cómo adaptar tipos de Python personalizados a valores de SQLite

SQLite solo admite un conjunto limitado de tipos de datos de forma nativa. Para almacenar tipos personalizados de Python en bases de datos SQLite, adáptelos a uno de los tipos Python que SQLite entiende de forma nativa.

Hay dos formas de adaptar los objetos de Python a los tipos de SQLite: dejar que su objeto se adapte a sí mismo o usar un *adapter callable*. Este último prevalecerá sobre el primero. Para una biblioteca que exporta un tipo personalizado, puede tener sentido permitir que ese tipo se adapte. Como desarrollador de aplicaciones, puede tener más sentido tomar el control directo registrando funciones de adaptador personalizadas.

Cómo escribir objetos adaptables

Supongamos que tenemos una clase **`Point`** que representa un par de coordenadas, `x` e `y`, en un sistema de coordenadas cartesianas. El par de coordenadas se almacenará como una cadena de texto en la base de datos, utilizando un punto y coma para separar las coordenadas. Esto se puede implementar agregando un método `__conform__(self, protocol)` que retorna el valor adaptado. El objeto pasado a *protocolo* será de tipo **`PrepareProtocol`**.

```
class Point:
    def __init__(self, x, y):
        self.x, self.y = x, y

    def __conform__(self, protocol):
        if protocol is sqlite3.PrepareProtocol:
            return f"{self.x};{self.y}"

con = sqlite3.connect(":memory:")
cur = con.cursor()

cur.execute("SELECT ?", (Point(4.0, -3.2),))
print(cur.fetchone()[0])
```

```
con.close()
```

Como registrar un adaptador invocable

La otra posibilidad es crear una función que convierta el objeto Python a un tipo compatible de SQLite. Esta función puede de esta forma ser registrada usando un `register_adapter()`.

```
class Point:
    def __init__(self, x, y):
        self.x, self.y = x, y

def adapt_point(point):
    return f"{point.x};{point.y}"

sqlite3.register_adapter(Point, adapt_point)

con = sqlite3.connect(":memory:")
cur = con.cursor()

cur.execute("SELECT ?", (Point(1.0, 2.5),))
print(cur.fetchone()[0])
con.close()
```

Como convertir valores SQLite a tipos de Python personalizados

Escribir un adaptador le permite convertir *de* tipos personalizados de Python *a* valores SQLite. Para poder convertir *de* valores SQLite *a* tipos personalizados de Python, usamos *convertidores*.

Regresemos a la clase `Point`. Almacenamos las coordenadas x y y separadas por punto y coma como una cadena de caracteres en SQLite.

Primero, se define una función de conversión que acepta la cadena de texto como un parámetro y construya un objeto `Point` de ahí.

Nota

Las funciones de conversión **siempre** son llamadas con un objeto `bytes`, no importa bajo qué tipo de dato se envió el valor a SQLite.

```
def convert_point(s):
    x, y = map(float, s.split(b";"))
    return Point(x, y)
```

Ahora necesitamos decirle a `sqlite3` cuando debería convertir un valor dado SQLite. Esto se hace cuando se conecta a una base de datos, utilizando el parámetro *detect_types* de `connect()`. Hay tres opciones:

- Implícito: establece *detect_types* para que `PARSE_DECLTYPES`
- Explícito: establece *detect_types* para que `PARSE_COLNAMES`
- Ambos: establece *detect_types* para `sqlite3.PARSE_DECLTYPES | sqlite3.PARSE_COLNAMES`. Los nombres de columna tienen prioridad sobre los tipos declarados.

El siguiente ejemplo ilustra ambos enfoques:

```
class Point:
    def __init__(self, x, y):
        self.x, self.y = x, y

    def __repr__(self):
        return f"Point({self.x}, {self.y})"

def adapt_point(point):
    return f"{point.x};{point.y}"

def convert_point(s):
    x, y = list(map(float, s.split(b";")))
    return Point(x, y)

# Register the adapter and converter
sqlite3.register_adapter(Point, adapt_point)
sqlite3.register_converter("point", convert_point)
```

```
# 1) Parse using declared types
p = Point(4.0, -3.2)
con = sqlite3.connect(":memory:", detect_types=sqlite3.P
cur = con.execute("CREATE TABLE test(p point)")

cur.execute("INSERT INTO test(p) VALUES(?)", (p,))
cur.execute("SELECT p FROM test")
print("with declared types:", cur.fetchone()[0])
cur.close()
con.close()

# 2) Parse using column names
con = sqlite3.connect(":memory:", detect_types=sqlite3.P
cur = con.execute("CREATE TABLE test(p)")

cur.execute("INSERT INTO test(p) VALUES(?)", (p,))
cur.execute('SELECT p AS "p [point]" FROM test')
print("with column names:", cur.fetchone()[0])
cur.close()
con.close()
```

Ejemplos para adaptadores y convertidores

En esta sección se muestran ejemplos para adaptadores y convertidores comunes.

```
import datetime as dt
import sqlite3

def adapt_date_iso(val):
    """Adapt datetime.date to ISO 8601 date."""
    return val.isoformat()

def adapt_datetime_iso(val):
    """Adapt datetime.datetime to timezone-naive ISO 860
    return val.replace(tzinfo=None).isoformat()

def adapt_datetime_epoch(val):
    """Adapt datetime.datetime to Unix timestamp."""
    return int(val.timestamp())

sqlite3.register_adapter(dt.date, adapt_date_iso)
sqlite3.register_adapter(dt.datetime, adapt_datetime_isc
sqlite3.register_adapter(dt.datetime, adapt_datetime_epc

def convert_date(val):
    """Convert ISO 8601 date to datetime.date object."""
    return dt.date.fromisoformat(val.decode())

def convert_datetime(val):
    """Convert ISO 8601 datetime to datetime.datetime ob
    return dt.datetime.fromisoformat(val.decode())

def convert_timestamp(val):
    """Convert Unix epoch timestamp to datetime.datetime
    return dt.datetime.fromtimestamp(int(val))

sqlite3.register_converter("date", convert_date)
sqlite3.register_converter("datetime", convert_datetime)
sqlite3.register_converter("timestamp", convert_timestam
```

Cómo utilizar los métodos de acceso directo de conexión

Usando los métodos **`execute()`**, **`executemany()`**, y **`executescript()`** de la clase **`Connection`**, su código se puede escribir de manera más concisa porque no tiene que crear los (a menudo superfluo) objetos **`Cursor`** explícitamente. Por el contrario, los objetos **`Cursor`** son creados implícitamente y esos métodos de acceso directo retornarán objetos cursores. De esta forma, se puede ejecutar una sentencia `SELECT` e iterar sobre ella directamente usando un simple llamado sobre el objeto de clase **`Connection`**.

```
# Create and fill the table.
con = sqlite3.connect(":memory:")
con.execute("CREATE TABLE lang(name, first_appeared)")
```

```
data = [
    ("C++", 1985),
    ("Objective-C", 1984),
]
con.executemany("INSERT INTO lang(name, first_appeared)

# Print the table contents
for row in con.execute("SELECT name, first_appeared FRO
    print(row)

print("I just deleted", con.execute("DELETE FROM lang").

# close() is not a shortcut method and it's not called a
# the connection object should be closed manually
con.close()
```

Como usar la conexión con un administrador de contexto

Un objeto **`Connection`** se puede utilizar como un administrador de contexto que confirma o revierte automáticamente las transacciones abiertas al salir del cuerpo del administrador de contexto. Si el cuerpo de la instrucción **`with`** finaliza sin excepciones, la transacción se confirma. Si esta confirmación falla, o si el cuerpo de la instrucción `with` genera una excepción no detectada, la transacción se revierte. Si **`autocommit`** es `False`, se abre implícitamente una nueva transacción después de confirmar o revertir.

Si no hay ninguna transacción abierta al salir del cuerpo de la declaración `with`, o si **`autocommit`** es `True`, el administrador de contexto no hace nada.

Nota

El administrador de contexto no abre implícitamente una nueva transacción ni cierra la conexión. Si necesita un administrador de contexto de cierre, considere usar **`contextlib.closing()`**.

```
con = sqlite3.connect(":memory:")
con.execute("CREATE TABLE lang(id INTEGER PRIMARY KEY, r

# Successful, con.commit() is called automatically after
with con:
    con.execute("INSERT INTO lang(name) VALUES(?)", ("Py

# con.rollback() is called after the with block finishes
# the exception is still raised and must be caught
try:
    with con:
        con.execute("INSERT INTO lang(name) VALUES(?)",
except sqlite3.IntegrityError:
    print("couldn't add Python twice")

# Connection object used as context manager only commits
# so the connection object should be closed manually
con.close()
```

Como trabajar con URIs SQLite

Algunos trucos útiles de URI incluyen:

- Abra una base de datos en modo de solo lectura:

```
>>> con = sqlite3.connect("file:tutorial.db?mode=ro", ur
>>> con.execute("CREATE TABLE readonly(data)")
Traceback (most recent call last):
OperationalError: attempt to write a readonly database
>>> con.close()
```

- No cree implícitamente un nuevo archivo de base de datos si aún no existe; esto lanzará un `OperationalError` si no puede crear un nuevo archivo:

```
>>> con = sqlite3.connect("file:nosuchdb.db?mode=rw", ur
Traceback (most recent call last):
OperationalError: unable to open database file
```

- Crea un nombre compartido sobre una base de datos en memoria:

```
db = "file:mem1?mode=memory&cache=shared"
con1 = sqlite3.connect(db, uri=True)
con2 = sqlite3.connect(db, uri=True)
with con1:
    con1.execute("CREATE TABLE shared(data)")
    con1.execute("INSERT INTO shared VALUES(28)")
res = con2.execute("SELECT data FROM shared")
assert res.fetchone() == (28,)

con1.close()
con2.close()
```

Más información sobre esta característica, incluyendo una lista de opciones reconocidas, pueden encontrarse en SQLite URI documentation [https://www.sqlite.org/uri.html].

Cómo crear y utilizar fábricas de filas

De forma predeterminada, `sqlite3` representa cada fila como `tuple`. Si `tuple` no se adapta a sus necesidades, puede utilizar la clase `sqlite3.Row` o una `row_factory` personalizada.

Si bien `row_factory` existe como atributo tanto en `Cursor` como en `Connection`, se recomienda configurar `Connection.row_factory`, de modo que todos los cursores creados a partir de la conexión utilicen la misma fábrica de filas.

`Row` proporciona acceso indexado y sin distinción entre mayúsculas y minúsculas a las columnas, con una sobrecarga de memoria y un impacto en el rendimiento mínimos en comparación con `tuple`. Para utilizar `Row` como una fábrica de filas, asígnelo al atributo `row_factory`:

```
>>> con = sqlite3.connect(":memory:")
>>> con.row_factory = sqlite3.Row
```

Las consultas ahora devuelven objetos `Row`:

```
>>> res = con.execute("SELECT 'Earth' AS name, 6378 AS r
>>> row = res.fetchone()
>>> row.keys()
['name', 'radius']
>>> row[0]         # Access by index.
'Earth'
>>> row["name"]    # Access by name.
'Earth'
>>> row["RADIUS"]  # Column names are case-insensitive.
6378
>>> con.close()
```

Nota

La cláusula `FROM` se puede omitir en la declaración `SELECT`, como en el ejemplo anterior. En tales casos, SQLite devuelve una sola fila con columnas definidas por expresiones, por ejemplo, literales, con los alias indicados `expr AS alias`.

Puede crear un **`row_factory`** personalizado que devuelva cada fila como un **`dict`**, con nombres de columnas asignados a valores:

```
def dict_factory(cursor, row):
    fields = [column[0] for column in cursor.descriptio
    return {key: value for key, value in zip(fields, ro
```

Al usarlo, las consultas ahora devuelven un **`dict`** en lugar de un **`tuple`**:

```
>>> con = sqlite3.connect(":memory:")
>>> con.row_factory = dict_factory
>>> for row in con.execute("SELECT 1 AS a, 2 AS b"):
...     print(row)
{'a': 1, 'b': 2}
>>> con.close()
```

La siguiente fábrica de filas devuelve un named tuple:

```
from collections import namedtuple
```

```
def namedtuple_factory(cursor, row):
    fields = [column[0] for column in cursor.descriptio
    cls = namedtuple("Row", fields)
    return cls._make(row)
```

`namedtuple_factory()` se puede utilizar de la siguiente manera:

```
>>> con = sqlite3.connect(":memory:")
>>> con.row_factory = namedtuple_factory
>>> cur = con.execute("SELECT 1 AS a, 2 AS b")
>>> row = cur.fetchone()
>>> row
Row(a=1, b=2)
>>> row[0]  # Indexed access.
1
>>> row.b   # Attribute access.
2
>>> con.close()
```

Con algunos ajustes, la receta anterior se puede adaptar para utilizar un **`dataclass`**, o cualquier otra clase personalizada, en lugar de un **`namedtuple`**.

Cómo manejar codificaciones de texto que no sean UTF-8

De forma predeterminada, **`sqlite3`** utiliza **`str`** para adaptar los valores de SQLite con el tipo de datos `TEXT`. Esto funciona bien para texto codificado en UTF-8, pero puede fallar para otras codificaciones y UTF-8 no válido. Puede utilizar un **`text_factory`** personalizado para manejar estos casos.

Debido a flexible typing [https://www.sqlite.org/flextypegood.html] de SQLite, no es raro encontrar columnas de tabla con el tipo de datos `TEXT` que contienen codificaciones que no son UTF-8, o incluso datos arbitrarios. Para demostrarlo, supongamos que tenemos una base de datos con texto codificado en ISO-8859-2 (Latin-2), por ejemplo, una tabla de entradas de diccionario checo-inglés. Suponiendo que ahora tenemos una instancia **`Connection`** **`con`** conectada a esta base de datos, podemos decodificar el texto codificado en Latin-2 utilizando este `text_factory`:

```
con.text_factory = lambda data: str(data, encoding="lati
```

Para datos UTF-8 no válidos o arbitrarios almacenados en las columnas de la tabla `TEXT`, puede utilizar la siguiente técnica, tomada prestada de CÓMO (HOWTO) Unicode:

```
con.text_factory = lambda data: str(data, errors="surrog
```

Nota

La API del módulo `sqlite3` no admite cadenas que contengan sustitutos.

Ver también

CÓMO (HOWTO) Unicode

Explicación

Control transaccional

`sqlite3` ofrece múltiples métodos para controlar si se abren y cierran las transacciones de la base de datos, cuándo y cómo. Se recomienda Control de transacciones mediante el atributo autocommit, mientras que Control de transacciones mediante el atributo isolation_level conserva el comportamiento anterior a Python 3.12.

Control de transacciones mediante el atributo `autocommit`

La forma recomendada de controlar el comportamiento de las transacciones es a través del atributo `Connection.autocommit`, que preferiblemente debe configurarse utilizando el parámetro *autocommit* de `connect()`.

Se recomienda configurar *autocommit* en `False`, lo que implica un control de transacciones compatible con **PEP 249** [https://peps.python.org/pep-0249/]. Esto significa:

- **`sqlite3`** garantiza que una transacción esté siempre abierta, por lo que `connect()`, `Connection.commit()` y `Connection.rollback()` abrirán implícitamente una nueva transacción (inmediatamente después de cerrar la pendiente, en el caso de las dos últimas). **`sqlite3`** utiliza instrucciones `BEGIN DEFERRED` al abrir transacciones.
- Las transacciones deben confirmarse explícitamente mediante `commit()`.
- Las transacciones deben revertirse explícitamente utilizando `rollback()`.
- Se realiza una reversión implícita si la base de datos está modificada como `close()` con cambios pendientes.

Establezca *autocommit* en `True` para habilitar autocommit mode [https://www.sqlite.org/lang_transaction.html#implicit_versus_explicit_transactions] de SQLite. En este modo, `Connection.commit()` y `Connection.rollback()` no tienen efecto. Tenga en cuenta que el modo de confirmación automática de SQLite es distinto del atributo `Connection.autocommit` compatible con **PEP 249** [https://peps.python.org/pep-0249/]; utilice `Connection.in_transaction` para consultar el modo de confirmación automática de bajo nivel de SQLite.

Establezca *autocommit* en `LEGACY_TRANSACTION_CONTROL` para dejar el comportamiento de control de transacciones al atributo `Connection.isolation_level`. Consulte Control de transacciones mediante el atributo isolation_level para obtener más información.

Control de transacciones mediante el atributo `isolation_level`

Nota

La forma recomendada de controlar las transacciones es mediante el atributo `autocommit`. Consulte Control de transacciones mediante el atributo autocommit.

Si `Connection.autocommit` se configura en `LEGACY_TRANSACTION_CONTROL` (el valor predeterminado), el comportamiento de la transacción se controla mediante el atributo `Connection.isolation_level`. De lo contrario, **`isolation_level`** no tiene ningún efecto.

Si el atributo de conexión `isolation_level` no es `None`, las nuevas transacciones se abrirán implícitamente antes de `execute()` y `executemany()` ejecutará sentencias `INSERT`, `UPDATE`, `DELETE` o `REPLACE`; para otras sentencias, no se realiza ningún manejo de transacción implícito. Utilice los métodos `commit()` y `rollback()` para confirmar y deshacer respectivamente las transacciones pendientes. Puede elegir el SQLite transaction behaviour [https://www.sqlite.org/lang_transaction.html#deferred_immediate_and_exclusive_transactions] subyacente, es decir, si y qué tipo de declaraciones `BEGIN` **`sqlite3`** se ejecutarán implícitamente, a través del atributo **`isolation_level`**.

Si `isolation_level` se establece como `None`, no se abre ninguna transacción implícitamente. Esto deja la biblioteca SQLite subyacente en autocommit mode [https://www.sqlite.org/lang_transaction.html#implicit_versus_explicit_transactions], pero también permite que el usuario realice su propio manejo de transacciones usando declaraciones SQL explícitas. El modo de confirmación automática de la biblioteca SQLite subyacente se puede consultar mediante el atributo `in_transaction`.

El método `executescript()` guarda implícitamente cualquier transacción pendiente antes de la ejecución del script SQL dado, independientemente del valor de `isolation_level`.

Distinto en la versión 3.6: **`sqlite3`** solía realizar commit en transacciones implícitamente antes de sentencias DDL. Este ya no es el caso.

Distinto en la versión 3.12: La forma recomendada de controlar las transacciones ahora es a través del atributo `autocommit`.

Compresión de datos y archivado

The modules described in this chapter support data compression with the zlib, gzip, bzip2, lzma, and zstd algorithms, and the creation of ZIP- and tar-format archives. See also Operaciones de archivado provided by the `shutil` module.

- The `compression` package
- `compression.zstd` — Compression compatible with the Zstandard format
  - Exceptions
  - Reading and writing compressed files
  - Compressing and decompressing data in memory
  - Zstandard dictionaries
  - Advanced parameter control
  - Miscellaneous
  - Examples
- `zlib` — Compression compatible with **gzip**
- `gzip` — Support for **gzip** files
  - Ejemplos de uso
  - Command-line interface
    - Command-line options
- `bz2` — Support for **bzip2** compression
  - (De)compresión de archivos
  - (De)compresión incremental
  - (Des)comprimir en un solo paso
  - Ejemplos de uso
- `lzma` — Compression using the LZMA algorithm

    - Leyendo y escribiendo ficheros comprimidos
    - Comprimiendo y descomprimiendo datos en memoria
    - Misceláneas
    - Especificando cadenas de filtro personalizadas
    - Ejemplos
- `zipfile` — Work with ZIP archives
    - ZipFile objects
    - Path objects
    - PyZipFile objects
    - ZipInfo objects
    - Command-line interface
        - Opciones de línea de comando
    - Problemas de descompresión
        - Del archivo mismo
        - File system limitations
        - Limitaciones de recursos
        - Interrupción
        - Comportamientos predeterminados de extracción
- `tarfile` — Read and write tar archive files
    - Objetos *TarFile*
    - Objetos TarInfo
    - Extraction filters
        - Default named filters
        - Filter errors
        - Hints for further verification
        - Supporting older Python versions
        - Stateful extraction filter example
    - Interfaz de línea de comandos
        - Opciones de línea de comandos
    - Ejemplos

    - Reading examples
    - Writing examples
- Formatos tar con soporte
- Problemas unicode

The `compression` package

Added in version 3.14.

The **`compression`** package contains the canonical compression modules containing interfaces to several different compression algorithms. Some of these modules have historically been available as separate modules; those will continue to be available under their original names for compatibility reasons, and will not be removed without a deprecation cycle. The use of modules in **`compression`** is encouraged where practical.

- **`compression.bz2`** – Re-exports **`bz2`**
- **`compression.gzip`** – Re-exports **`gzip`**
- **`compression.lzma`** – Re-exports **`lzma`**
- **`compression.zlib`** – Re-exports **`zlib`**
- **`compression.zstd`** – Wrapper for the Zstandard compression library

`compression.zstd` — Compression compatible with the Zstandard format

Added in version 3.14.

Source code: Lib/compression/zstd/_init_.py [https://github.com/python/cpython/tree/3.14/Lib/compression/zstd/_init_.py]

This module provides classes and functions for compressing and decompressing data using the Zstandard (or *zstd*) compression algorithm. The zstd manual [https://facebook.github.io/zstd/doc/api_manual_latest.html] describes Zstandard as «a fast lossless compression algorithm, targeting real-time compression scenarios at zlib-level and better compression ratios.» Also included is a file interface that supports reading and writing the contents of `.zst` files created by the **zstd** utility, as well as raw zstd compressed streams.

The **`compression.zstd`** module contains:

- The **`open()`** function and **`ZstdFile`** class for reading and writing compressed files.
- The **`ZstdCompressor`** and **`ZstdDecompressor`** classes for incremental (de)compression.
- The **`compress()`** and **`decompress()`** functions for one-shot (de)compression.
- The **`train_dict()`** and **`finalize_dict()`** functions and the **`ZstdDict`** class to train and manage Zstandard dictionaries.
- The **`CompressionParameter`**, **`DecompressionParameter`**, and **`Strategy`** classes for setting advanced (de)compression parameters.

This is an optional module. If it is missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

Exceptions

exception compression.zstd.ZstdError

This exception is raised when an error occurs during compression or decompression, or while initializing the (de)compressor state.

Reading and writing compressed files

compression.zstd.open(*file*, */*, *mode='rb'*, ***, *level=None*, *options=None*, *zstd_dict=None*, *encoding=None*, *errors=None*, *newline=None*)

Open a Zstandard-compressed file in binary or text mode, returning a file object.

The *file* argument can be either a file name (given as a **`str`**, **`bytes`** or path-like object), in which case the named file is opened, or it can be an existing file object to read from or write to.

The mode argument can be either `'rb'` for reading (default), `'wb'` for overwriting, `'ab'` for appending, or `'xb'` for exclusive creation. These can equivalently be given as `'r'`, `'w'`, `'a'`, and `'x'` respectively. You may also open in text mode with `'rt'`, `'wt'`, `'at'`, and `'xt'` respectively.

When reading, the *options* argument can be a dictionary providing advanced decompression parameters; see **`DecompressionParameter`** for detailed information about supported parameters. The *zstd_dict* argument is a **`ZstdDict`** instance to be used during decompression. When reading, if the *level* argument is not None, a **`TypeError`** will be raised.

When writing, the *options* argument can be a dictionary providing advanced compression parameters; see **`CompressionParameter`** for detailed information about supported parameters. The *level* argument is the compression level to use when writing compressed data. Only one of *level* or *options* may be non-None. The *zstd_dict* argument is a **`ZstdDict`** instance to be used during compression.

In binary mode, this function is equivalent to the **`ZstdFile`** constructor: `ZstdFile(file, mode, ...)`. In this case, the *encoding*, *errors*, and *newline* parameters must not be provided.

In text mode, a **`ZstdFile`** object is created, and wrapped in an **`io.TextIOWrapper`** instance with the specified encoding, error handling behavior, and line endings.

class compression.zstd.ZstdFile(*file*, /, *mode='rb'*, *, *level=None*, *options=None*, *zstd_dict=None*)

Open a Zstandard-compressed file in binary mode.

A **`ZstdFile`** can wrap an already-open file object, or operate directly on a named file. The *file* argument specifies either the file object to wrap, or the name of the file to open (as a **`str`**, **`bytes`** or path-like object). If wrapping an existing file object, the wrapped file will not be closed when the **`ZstdFile`** is closed.

The *mode* argument can be either `'rb'` for reading (default), `'wb'` for overwriting, `'xb'` for exclusive creation, or `'ab'` for appending. These can equivalently be given as `'r'`, `'w'`, `'x'` and `'a'` respectively.

If *file* is a file object (rather than an actual file name), a mode of `'w'` does not truncate the file, and is instead equivalent to `'a'`.

When reading, the *options* argument can be a dictionary providing advanced decompression parameters; see **`DecompressionParameter`** for detailed information about supported parameters. The *zstd_dict* argument is a **`ZstdDict`** instance to be used during decompression. When reading, if the *level* argument is not None, a `TypeError` will be raised.

When writing, the *options* argument can be a dictionary providing advanced compression parameters; see `CompressionParameter` for detailed information about supported parameters. The *level* argument is the compression level to use when writing compressed data. Only one of *level* or *options* may be passed. The *zstd_dict* argument is a `ZstdDict` instance to be used during compression.

`ZstdFile` supports all the members specified by `io.BufferedIOBase`, except for `detach()` and `truncate()`. Iteration and the `with` statement are supported.

The following method and attributes are also provided:

peek(*size=-1*)

Return buffered data without advancing the file position. At least one byte of data will be returned, unless EOF has been reached. The exact number of bytes returned is unspecified (the *size* argument is ignored).

Nota

While calling **`peek()`** does not change the file position of the **`ZstdFile`**, it may change the position of the underlying file object (for example, if the **`ZstdFile`** was constructed by passing a file object for *file*).

mode

`'rb'` for reading and `'wb'` for writing.

name

The name of the Zstandard file. Equivalent to the `name` attribute of the underlying file object.

Compressing and decompressing data in memory

compression.zstd.compress(*data*, *level=None*, *options=None*, *zstd_dict=None*)

Compress *data* (a bytes-like object), returning the compressed data as a `bytes` object.

The *level* argument is an integer controlling the level of compression. *level* is an alternative to setting `CompressionParameter.compression_level` in *options*. Use `bounds()` on `compression_level` to get the values that can be passed for *level*. If advanced compression options are needed, the *level* argument must be omitted and in the *options* dictionary the `CompressionParameter.compression_level` parameter should be set.

The *options* argument is a Python dictionary containing advanced compression parameters. The valid keys and values for compression parameters are documented as part of the `CompressionParameter` documentation.

The *zstd_dict* argument is an instance of `ZstdDict` containing trained data to improve compression efficiency. The function `train_dict()` can be used to generate a Zstandard dictionary.

compression.zstd.decompress(*data*, *zstd_dict=None*, *options=None*)

Decompress *data* (a bytes-like object), returning the uncompressed data as a `bytes` object.

The *options* argument is a Python dictionary containing advanced decompression parameters. The valid keys and values for compression parameters are documented as part of the `DecompressionParameter` documentation.

The *zstd_dict* argument is an instance of `ZstdDict` containing trained data used during compression. This must be the same Zstandard dictionary used during compression.

If *data* is the concatenation of multiple distinct compressed frames, decompress all of these frames, and return the concatenation of the results.

class compression.zstd.ZstdCompressor(*level=None*, *options=None*, *zstd_dict=None*)

Create a compressor object, which can be used to compress data incrementally.

For a more convenient way of compressing a single chunk of data, see the module-level function `compress()`.

The *level* argument is an integer controlling the level of compression. *level* is an alternative to setting `CompressionParameter.compression_level` in *options*. Use `bounds()` on `compression_level` to get the values that can be passed for *level*. If advanced compression options are needed, the *level* argument must be omitted and in the *options* dictionary the `CompressionParameter.compression_level` parameter should be set.

The *options* argument is a Python dictionary containing advanced compression parameters. The valid keys and values for compression parameters are documented as part of the `CompressionParameter` documentation.

The *zstd_dict* argument is an optional instance of `ZstdDict` containing trained data to improve compression efficiency. The function `train_dict()` can be used to generate a Zstandard dictionary.

compress(*data*, *mode=ZstdCompressor.CONTINUE*)

Compress *data* (a bytes-like object), returning a `bytes` object with compressed data if possible, or otherwise an empty `bytes` object. Some of *data* may be buffered internally, for use in later calls to **`compress()`** and `flush()`. The returned data should be concatenated with the output of any previous calls to **`compress()`**.

The *mode* argument is a **`ZstdCompressor`** attribute, either `CONTINUE`, `FLUSH_BLOCK`, or `FLUSH_FRAME`.

When all data has been provided to the compressor, call the `flush()` method to finish the compression process. If **`compress()`** is called with *mode* set to `FLUSH_FRAME`, **`flush()`** should not be called, as it would write out a new empty frame.

flush(*mode=ZstdCompressor.FLUSH_FRAME*)

Finish the compression process, returning a `bytes` object containing any data stored in the compressor's internal buffers.

The *mode* argument is a **`ZstdCompressor`** attribute, either `FLUSH_BLOCK`, or `FLUSH_FRAME`.

set_pledged_input_size(*size*)

Specify the amount of uncompressed data *size* that will be provided for the next frame. *size* will be written into the frame header of the next frame unless `CompressionParameter.content_size_flag` is `False` or `0`. A size of `0` means that the frame is empty. If *size* is `None`, the frame header will omit the frame size. Frames that include the uncompressed data size require less memory to decompress, especially at higher compression levels.

If `last_mode` is not `FLUSH_FRAME`, a `ValueError` is raised as the compressor is not at the start of a frame. If the pledged size does not match the actual size of data provided to `compress()`, future calls to **`compress()`** or `flush()` may raise `ZstdError` and the last chunk of data may be lost.

After `flush()` or `compress()` are called with mode

`FLUSH_FRAME`, the next frame will not include the frame size into the header unless **`set_pledged_input_size()`** is called again.

CONTINUE

Collect more data for compression, which may or may not generate output immediately. This mode optimizes the compression ratio by maximizing the amount of data per block and frame.

FLUSH_BLOCK

Complete and write a block to the data stream. The data returned so far can be immediately decompressed. Past data can still be referenced in future blocks generated by calls to `compress()`, improving compression.

FLUSH_FRAME

Complete and write out a frame. Future data provided to `compress()` will be written into a new frame and *cannot* reference past data.

last_mode

The last mode passed to either `compress()` or `flush()`. The value can be one of `CONTINUE`, `FLUSH_BLOCK`, or `FLUSH_FRAME`. The initial value is **`FLUSH_FRAME`**, signifying that the compressor is at the start of a new frame.

class compression.zstd.ZstdDecompressor(*zstd_dict=None, options=None*)

Create a decompressor object, which can be used to decompress data incrementally.

For a more convenient way of decompressing an entire compressed stream at once, see the module-level function `decompress()`.

The *options* argument is a Python dictionary containing advanced decompression parameters. The valid keys and values for compression parameters are documented as part of the `DecompressionParameter` documentation.

The *zstd_dict* argument is an instance of `ZstdDict` containing trained data used during compression. This must be the same Zstandard dictionary used during compression.

Nota

This class does not transparently handle inputs containing multiple compressed frames, unlike the `decompress()` function and `ZstdFile` class. To decompress a multi-frame input, you should use **`decompress()`**, **`ZstdFile`** if working with a file object, or multiple **`ZstdDecompressor`** instances.

decompress(*data*, *max_length*=*-1*)

Decompress *data* (a bytes-like object), returning uncompressed data as bytes. Some of *data* may be buffered internally, for use in later calls to **`decompress()`**. The returned data should be concatenated with the output of any previous calls to **`decompress()`**.

If *max_length* is non-negative, the method returns at most *max_length* bytes of decompressed data. If this limit is reached and further output can be produced, the `needs_input` attribute will be set to `False`. In this case, the next call to **`decompress()`** may provide *data* as `b''` to obtain more of the output.

If all of the input data was decompressed and returned (either because this was less than *max_length* bytes, or because *max_length* was negative), the `needs_input` attribute will be set to `True`.

Attempting to decompress data after the end of a frame will raise a `ZstdError`. Any data found after the end of the frame is ignored and saved in the `unused_data` attribute.

eof

`True` if the end-of-stream marker has been reached.

unused_data

Data found after the end of the compressed stream.

Before the end of the stream is reached, this will be `b''`.

needs_input

`False` if the `decompress()` method can provide more decompressed data before requiring new compressed input.

Zstandard dictionaries

compression.zstd.train_dict(*samples*, *dict_size*)

Train a Zstandard dictionary, returning a `ZstdDict` instance. Zstandard dictionaries enable more efficient compression of smaller sizes of data, which is traditionally difficult to compress due to less repetition. If you are compressing multiple similar groups of data (such as similar files), Zstandard dictionaries can improve compression ratios and speed significantly.

The *samples* argument (an iterable of `bytes` objects), is the population of samples used to train the Zstandard dictionary.

The *dict_size* argument, an integer, is the maximum size (in bytes) the Zstandard dictionary should be. The Zstandard documentation suggests an absolute maximum of no more than 100 KB, but the maximum can often be smaller depending on the data. Larger dictionaries generally slow down compression, but improve compression ratios. Smaller dictionaries lead to faster compression, but reduce the compression ratio.

compression.zstd.finalize_dict(*zstd_dict*, */*, *samples*, *dict_size*, *level*)

An advanced function for converting a «raw content» Zstandard dictionary into a regular Zstandard dictionary. «Raw content» dictionaries are a sequence of bytes that do not need to follow the structure of a normal Zstandard dictionary.

The *zstd_dict* argument is a **`ZstdDict`** instance with the **`dict_content`** containing the raw dictionary contents.

The *samples* argument (an iterable of **`bytes`** objects), contains sample data for generating the Zstandard dictionary.

The *dict_size* argument, an integer, is the maximum size (in bytes) the Zstandard dictionary should be. See **`train_dict()`** for suggestions on the maximum dictionary size.

The *level* argument (an integer) is the compression level expected to be passed to the compressors using this dictionary. The dictionary information varies for each compression level, so tuning for the proper compression level can make compression more efficient.

class compression.zstd.ZstdDict(*dict_content*, */*, ***, *is_raw=False*)

A wrapper around Zstandard dictionaries. Dictionaries can be used to improve the compression of many small chunks of data. Use **`train_dict()`** if you need to train a new dictionary from sample data.

The *dict_content* argument (a bytes-like object), is the already trained dictionary information.

The *is_raw* argument, a boolean, is an advanced parameter controlling the meaning of *dict_content*. `True` means *dict_content* is a «raw content» dictionary, without any format restrictions. `False` means *dict_content* is an ordinary Zstandard dictionary, created from Zstandard functions, for example, `train_dict()` or the external **zstd** CLI.

When passing a **`ZstdDict`** to a function, the **`as_digested_dict`** and **`as_undigested_dict`** attributes can control how the dictionary is loaded by passing them as the `zstd_dict` argument, for example, `compress(data, zstd_dict=zd.as_digested_dict)`. Digesting a dictionary is a costly operation that occurs when loading a Zstandard dictionary. When making multiple calls to compression or decompression, passing a digested dictionary will reduce the overhead of loading the dictionary.

	Digested dictionary	**Undigested dictionary**
Advanced parameters of the compressor, which may be overridden by the dictionary's parameters	`window_log`, `hash_log`, `chain_log`, `search_log`, `min_match`, `target_length`, `strategy`, `enable_long_distance_matching`, `ldm_hash_log`, `ldm_min_match`, `ldm_bucket_size_log`, `ldm_hash_rate_log`, and some non-public parameters.	None
`ZstdDict` internally caches the dictionary	Yes. It's faster when loading a digested dictionary again with the same compression level.	No. If you wish to load an undigested dictionary multiple times, consider reusing a compressor object.

Difference for compression

If passing a **`ZstdDict`** without any attribute, an undigested dictionary is passed by default when compressing and a digested dictionary is generated if necessary and passed by default when decompressing.

dict_content

The content of the Zstandard dictionary, a `bytes` object. It's the same as the *dict_content* argument in the `__init__` method. It can be used with other programs, such as the `zstd` CLI program.

dict_id

Identifier of the Zstandard dictionary, a non-negative int value.

Non-zero means the dictionary is ordinary, created by Zstandard functions and following the Zstandard format.

`0` means a «raw content» dictionary, free of any format restriction, used for advanced users.

Nota

The meaning of `0` for **`ZstdDict.dict_id`** is different from the `dictionary_id` attribute to the **`get_frame_info()`** function.

as_digested_dict

Load as a digested dictionary.

as_undigested_dict

Load as an undigested dictionary.

Advanced parameter control

class compression.zstd.CompressionParameter

An **`IntEnum`** containing the advanced compression parameter keys that can be used when compressing data.

The **`bounds()`** method can be used on any attribute to get the valid values for that parameter.

Parameters are optional; any omitted parameter will have it’s value selected automatically.

Example getting the lower and upper bound of **`compression_level`**:

```
lower, upper = CompressionParameter.compression_lev
```

Example setting the **`window_log`** to the maximum size:

```
_lower, upper = CompressionParameter.window_log.bou
options = {CompressionParameter.window_log: upper}
compress(b'venezuelan beaver cheese', options=optio
```

bounds()

Return the tuple of int bounds, `(lower, upper)`, of a compression parameter. This method should be called on the attribute you wish to retrieve the bounds of. For example, to get the valid values for **`compression_level`**, one may check the result of `CompressionParameter.compression_level.bounds()`

Both the lower and upper bounds are inclusive.

compression_level

A high-level means of setting other compression parameters that affect the speed and ratio of compressing data.

Regular compression levels are greater than `0`. Values greater than `20` are considered «ultra» compression and require more memory than other levels. Negative values can be used to trade off faster compression for worse compression ratios.

Setting the level to zero uses **`COMPRESSION_LEVEL_DEFAULT`**.

window_log

Maximum allowed back-reference distance the compressor can use when compressing data, expressed as power of two, `1 << window_log` bytes. This parameter greatly influences the memory usage of compression. Higher values require more memory but gain better compression values.

A value of zero causes the value to be selected automatically.

hash_log

Size of the initial probe table, as a power of two. The resulting memory usage is `1 << (hash_log+2)` bytes. Larger tables improve compression ratio of strategies <= **`dfast`**, and improve compression speed of strategies > **`dfast`**.

A value of zero causes the value to be selected automatically.

chain_log

Size of the multi-probe search table, as a power of two. The resulting memory usage is `1 << (chain_log +2)` bytes. Larger tables result in better and slower compression. This parameter has no effect for the **`fast`** strategy. It's still useful when using **`dfast`** strategy, in which case it defines a secondary probe table.

A value of zero causes the value to be selected automatically.

search_log

Number of search attempts, as a power of two. More attempts result in better and slower compression. This parameter is useless for **`fast`** and **`dfast`** strategies.

A value of zero causes the value to be selected automatically.

min_match

Minimum size of searched matches. Larger values increase compression and decompression speed, but decrease ratio. Note that Zstandard can still find matches of smaller size, it just tweaks its search algorithm to look for this size and larger. For all strategies < `btopt`, the effective minimum is `4`; for all strategies > `fast`, the effective maximum is `6`.

A value of zero causes the value to be selected automatically.

target_length

The impact of this field depends on the selected `Strategy`.

For strategies `btopt`, `btultra` and `btultra2`, the value is the length of a match considered «good enough» to stop searching. Larger values make compression ratios better, but compresses slower.

For strategy `fast`, it is the distance between match sampling. Larger values make compression faster, but with a worse compression ratio.

A value of zero causes the value to be selected automatically.

strategy

The higher the value of selected strategy, the more complex the compression technique used by zstd, resulting in higher compression ratios but slower compression.

Ver también

`Strategy`

enable_long_distance_matching

Long distance matching can be used to improve compression for large inputs by finding large matches at greater distances. It increases memory usage and window size.

`True` or `1` enable long distance matching while `False` or `0` disable it.

Enabling this parameter increases default **`window_log`** to 128 MiB except when expressly set to a different value. This setting is enabled by default if **`window_log`** >= 128 MiB and the compression strategy >= **`btopt`** (compression level 16+).

ldm_hash_log

Size of the table for long distance matching, as a power of two. Larger values increase memory usage and compression ratio, but decrease compression speed.

A value of zero causes the value to be selected automatically.

ldm_min_match

Minimum match size for long distance matcher. Larger or too small values can often decrease the compression ratio.

A value of zero causes the value to be selected automatically.

ldm_bucket_size_log

Log size of each bucket in the long distance matcher hash table for collision resolution. Larger values improve collision resolution but decrease compression speed.

A value of zero causes the value to be selected automatically.

ldm_hash_rate_log

Frequency of inserting/looking up entries into the long distance matcher hash table. Larger values improve compression speed. Deviating far from the default value will likely result in a compression ratio decrease.

A value of zero causes the value to be selected automatically.

content_size_flag

Write the size of the data to be compressed into the Zstandard frame header when known prior to compressing.

This flag only takes effect under the following scenarios:

- Calling `compress()` for one-shot compression
- Providing all of the data to be compressed in the frame in a single `ZstdCompressor.compress()` call, with the `ZstdCompressor.FLUSH_FRAME` mode.
- Calling `ZstdCompressor.set_pledged_input_size()` with the exact amount of data that will be provided to the compressor prior to any calls to `ZstdCompressor.compress()` for the current frame. **`ZstdCompressor.set_pledged_input_size()`** must be called for each new frame.

All other compression calls may not write the size information into the frame header.

`True` or `1` enable the content size flag while `False` or `0` disable it.

checksum_flag

A four-byte checksum using XXHash64 of the uncompressed content is written at the end of each frame. Zstandard’s decompression code verifies the checksum. If there is a mismatch a `ZstdError` exception is raised.

`True` or `1` enable checksum generation while `False` or `0` disable it.

dict_id_flag

When compressing with a **`ZstdDict`**, the dictionary's ID is written into the frame header.

`True` or `1` enable storing the dictionary ID while `False` or `0` disable it.

nb_workers

Select how many threads will be spawned to compress in parallel. When **`nb_workers`** > 0, enables multi-threaded compression, a value of `1` means «one-thread multi-threaded mode». More workers improve speed, but also increase memory usage and slightly reduce compression ratio.

A value of zero disables multi-threading.

job_size

Size of a compression job, in bytes. This value is enforced only when **`nb_workers`** > = 1. Each compression job is completed in parallel, so this value can indirectly impact the number of active threads.

A value of zero causes the value to be selected automatically.

overlap_log

Sets how much data is reloaded from previous jobs (threads) for new jobs to be used by the look behind window during compression. This value is only used when **`nb_workers`** > = 1. Acceptable values vary from 0 to 9.

- 0 means dynamically set the

overlap amount
- 1 means no overlap
- 9 means use a full window size from the previous job

Each increment halves/doubles the overlap size. «8» means an overlap of `window_size/2`, «7» means an overlap of `window_size/4`, etc.

class compression.zstd.DecompressionParameter

An **`IntEnum`** containing the advanced decompression parameter keys that can be used when decompressing data. Parameters are optional; any omitted parameter will have it's value selected automatically.

The **`bounds()`** method can be used on any attribute to get the valid values for that parameter.

Example setting the **`window_log_max`** to the maximum size:

```
data = compress(b'Some very long buffer of bytes...

_lower, upper = DecompressionParameter.window_log_m

options = {DecompressionParameter.window_log_max: u
decompress(data, options=options)
```

bounds()

Return the tuple of int bounds, `(lower, upper)`, of a decompression parameter. This method should be called on the attribute you wish to retrieve the bounds of.

Both the lower and upper bounds are inclusive.

window_log_max

The base-two logarithm of the maximum size of the window used during decompression. This can be useful to limit the amount of memory used when decompressing data. A larger maximum window size leads to faster decompression.

A value of zero causes the value to be selected automatically.

class compression.zstd.Strategy

An `IntEnum` containing strategies for compression. Higher-numbered strategies correspond to more complex and slower compression.

Nota

The values of attributes of **`Strategy`** are not necessarily stable across zstd versions. Only the ordering of the attributes may be relied upon. The attributes are listed below in order.

The following strategies are available:

fast

dfast

greedy

lazy

lazy2

btlazy2

btopt

btultra

btultra2

Miscellaneous

compression.zstd.get_frame_info(*frame_buffer*)

Retrieve a **`FrameInfo`** object containing metadata about a Zstandard frame. Frames contain metadata related to the compressed data they hold.

class compression.zstd.FrameInfo

Metadata related to a Zstandard frame.

decompressed_size

The size of the decompressed contents of the frame.

dictionary_id

An integer representing the Zstandard dictionary ID needed for decompressing the frame. `0` means the dictionary ID was not recorded in the frame header. This may mean that a Zstandard dictionary is not needed, or that the ID of a required dictionary was not recorded.

compression.zstd.COMPRESSION_LEVEL_DEFAULT

The default compression level for Zstandard: `3`.

compression.zstd.zstd_version_info

Version number of the runtime zstd library as a tuple of integers (major, minor, release).

Examples

Reading in a compressed file:

```
from compression import zstd

with zstd.open("file.zst") as f:
    file_content = f.read()
```

Creating a compressed file:

```
from compression import zstd
```

```
data = b"Insert Data Here"
with zstd.open("file.zst", "w") as f:
    f.write(data)
```

Compressing data in memory:

```
from compression import zstd

data_in = b"Insert Data Here"
data_out = zstd.compress(data_in)
```

Incremental compression:

```
from compression import zstd

comp = zstd.ZstdCompressor()
out1 = comp.compress(b"Some data\n")
out2 = comp.compress(b"Another piece of data\n")
out3 = comp.compress(b"Even more data\n")
out4 = comp.flush()
# Concatenate all the partial results:
result = b"".join([out1, out2, out3, out4])
```

Writing compressed data to an already-open file:

```
from compression import zstd

with open("myfile", "wb") as f:
    f.write(b"This data will not be compressed\n")
    with zstd.open(f, "w") as zstf:
        zstf.write(b"This *will* be compressed\n")
    f.write(b"Not compressed\n")
```

Creating a compressed file using compression parameters:

```
from compression import zstd

options = {
   zstd.CompressionParameter.checksum_flag: 1
}
with zstd.open("file.zst", "w", options=options) as f:
    f.write(b"Mind if I squeeze in?")
```

`zlib` — Compression compatible with gzip

For applications that require data compression, the functions in this module allow compression and decompression, using the zlib library [https://www.zlib.net].

This is an optional module. If it is missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

zlib's functions have many options and often need to be used in a particular order. This documentation doesn't attempt to cover all of the permutations; consult the zlib manual [https://www.zlib.net/manual.html] for authoritative information.

Para leer y escribir archivos `.gz` consultar el módulo **`gzip`**.

La excepción y las funciones disponibles en este módulo son:

exception zlib.error

Excepción provocada en errores de compresión y descompresión.

zlib.adler32(*data*[, *value*])

Calcula una suma de comprobación Adler-32 de *data*. (Una suma de comprobación Adler-32 es casi tan confiable como un CRC32, pero se puede calcular mucho más rápidamente). El resultado es un entero de 32 bits sin signo. Si *value* está presente, se utiliza como el valor inicial de la suma de comprobación; de lo contrario, se utiliza un valor predeterminado de 1. Pasar *value* permite calcular una suma de comprobación en ejecución durante la concatenación de varias entradas. El algoritmo no es criptográficamente fuerte y no se debe utilizar para la autenticación o las firmas digitales. Puesto que está diseñado como un algoritmo de suma de comprobación, no es adecuado su uso como un algoritmo *hash* general.

Distinto en la versión 3.0: El resultado es siempre sin signo.

zlib.compress(*data, /, level=Z_DEFAULT_COMPRESSION, wbits=MAX_WBITS*)

Compresses the bytes in *data*, returning a bytes object containing compressed data. *level* is an integer from `0` to `9` or `-1` controlling the level of compression; See **`Z_BEST_SPEED`** (`1`), **`Z_BEST_COMPRESSION`** (`9`), **`Z_NO_COMPRESSION`** (`0`), and the default, **`Z_DEFAULT_COMPRESSION`** (`-1`) for more information about these values.

The *wbits* argument controls the size of the history buffer (or the «window size») used when compressing data, and whether a header and trailer is included in the output. It can take several ranges of values, defaulting to `15` (**`MAX_WBITS`**):

- +9 a +15: el logaritmo en base dos del tamaño de la ventana, que por lo tanto oscila entre 512 y 32768. Los valores más grandes producen una mejor compresión a expensas de un mayor uso de memoria. Como resultado se incluirá un encabezado y un avance específicos de zlib.
- −9 a −15: utiliza el valor absoluto de *wbits* como el logaritmo del tamaño de la ventana, al tiempo que produce una secuencia de salida sin encabezado ni suma de verificación.
- +25 a +31 = 16 + (9 a 15): utiliza los 4 bits bajos del valor como el logaritmo del tamaño de la ventana, mientras que incluye un encabezado básico **gzip** y suma de verificación en la salida.

Lanza la excepción **`error`** si ocurre cualquier error.

Distinto en la versión 3.6: level ahora se puede utilizar como parámetro de palabra clave.

Distinto en la versión 3.11: El parámetro *wbits* está ahora disponible para establecer bits de la ventana y tipo de compresión.

zlib.compressobj(*level=Z_DEFAULT_COMPRESSION*, *method=DEFLATED*, *wbits=MAX_WBITS*, *memLevel=DEF_MEM_LEVEL*, *strategy=Z_DEFAULT_STRATEGY*[, *zdict*])

Retorna un objeto de compresión, para ser usado para comprimir flujos de datos que no caben en la memoria de una vez.

level is the compression level – an integer from `0` to `9` or `-1`. See **`Z_BEST_SPEED`** (`1`), **`Z_BEST_COMPRESSION`** (`9`), **`Z_NO_COMPRESSION`** (`0`), and the default, **`Z_DEFAULT_COMPRESSION`** (`-1`) for more information about these values.

method es el algoritmo de compresión. Actualmente, el único valor admitido es **`DEFLATED`**.

El parámetro *wbits* controla el tamaño del búfer histórico (o el «tamaño de ventana») y qué formato de encabezado y cola será usado. Tiene el mismo significado que el descrito para decompress().

El argumento *memLevel* controla la cantidad de memoria utilizada para el estado de compresión interna. Los valores posibles varían de `1` a `9`. Los valores más altos usan más memoria, pero son más rápidos y producen una salida más pequeña.

strategy is used to tune the compression algorithm. Possible values are **`Z_DEFAULT_STRATEGY`**, **`Z_FILTERED`**, **`Z_HUFFMAN_ONLY`**, **`Z_RLE`** and **`Z_FIXED`**.

zdict es un diccionario de compresión predefinido. Este es una secuencia de bytes (como un objeto `bytes`) que contiene subsecuencias que se espera que ocurran con frecuencia en los datos que se van a comprimir. Las subsecuencias que se espera que sean más comunes deben aparecer al final del diccionario.

Distinto en la versión 3.3: Se agregó el parámetro *zdict* y el soporte de argumentos de palabras clave.

zlib.crc32(*data*[, *value*])

Calcula un suma de comprobación CRC (comprobación de redundancia cíclica, por sus siglas en inglés *Cyclic Redundancy Check*) de *datos*. El resultado es un entero de 32 bits sin signo. Si *value* está presente, se usa como el valor inicial de la suma de verificación; de lo contrario, se usa el valor predeterminado 0. Pasar *value* permite calcular una suma de verificación en ejecución sobre la concatenación de varias entradas. El algoritmo no es criptográficamente fuerte y no debe usarse para autenticación o firmas digitales. Dado que está diseñado para usarse como un algoritmo de suma de verificación, no es adecuado su uso como algoritmo *hash* general.

Distinto en la versión 3.0: El resultado es siempre sin signo.

zlib.decompress(*data*, */*, *wbits=MAX_WBITS*, *bufsize=DEF_BUF_SIZE*)

Descomprime los bytes en *data*, retornando un objeto de bytes que contiene los datos sin comprimir. El parámetro *wbits* depende del formato de *data*, y se trata más adelante. Si se da *bufsize*, se usa como el tamaño inicial del búfer de salida. Provoca la excepción `error` si se produce algún error.

El parámetro *wbits* controla el tamaño del búfer histórico (o el «tamaño de ventana») y qué formato de encabezado y cola se espera. Es similar al parámetro para `compressobj()`, pero acepta más rangos de valores:

- +8 a +15: el logaritmo en base dos del tamaño del

  búfer. La entrada debe incluir encabezado y cola zlib.
- 0: determina automáticamente el tamaño del búfer desde el encabezado zlib. Solo se admite desde zlib 1.2.3.5.
- -8 to -15: utiliza el valor absoluto de *wbits* como el logaritmo del tamaño del búfer. La entrada debe ser una transmisión sin formato, sin encabezado ni cola.
- +24 a +31 = 16 + (8 a 15): utiliza los 4 bits de bajo orden del valor como el logaritmo del tamaño del búfer. La entrada debe incluir encabezado y cola gzip.
- +40 a +47 = 32 + (8 a 15): utiliza los 4 bits de bajo orden del valor como el logaritmo del tamaño del búfer y acepta automáticamente el formato zlib o gzip.

Al descomprimir una secuencia, el tamaño del búfer no debe ser menor al tamaño utilizado originalmente para comprimir la secuencia; el uso de un valor demasiado pequeño puede generar una excepción `error`. El valor predeterminado *wbits* corresponde al tamaño más grande de búfer y requiere que se incluya encabezado y cola zlib.

bufsize es el tamaño inicial del búfer utilizado para contener datos descomprimidos. Si se requiere más espacio, el tamaño del búfer aumentará según sea necesario, por lo que no hay que tomar este valor como exactamente correcto; al ajustarlo solo se guardarán algunas llamadas en **`malloc()`**.

Distinto en la versión 3.6: wbits y *bufsize* se pueden usar como argumentos de palabra clave.

zlib.decompressobj(*wbits=MAX_WBITS*[, *zdict*])

Retorna un objeto de descompresión, que se utilizará para descomprimir flujos de datos que no caben en la memoria de una vez.

El parámetro *wbits* controla el tamaño del búfer histórico (o el «tamaño de ventana») y qué formato de encabezado y cola se espera. Tiene el mismo significado que described for decompress().

El parámetro *zdict* especifica un diccionario de compresión predefinido. Si se proporciona, este debe ser el mismo diccionario utilizado por el compresor que produjo los datos que se van a descomprimir.

Nota

Si *zdict* es un objeto mutable (como **`bytearray`**), no se debe modificar su contenido entre la llamada **`decompressobj()`** y la primera llamada al método descompresor `decompress()`.

Distinto en la versión 3.3: Se agregó el parámetro *zdict*.

Los objetos de compresión admiten los siguientes métodos:

Compress.compress(*data*)

Comprime *data*, y retorna al menos algunos de los datos comprimidos como un objeto de bytes. Estos datos deben concatenarse a la salida producida por cualquier llamada anterior al método **`compress()`**. Algunas entradas pueden mantenerse en un búfer interno para su posterior procesamiento.

Compress.flush([*mode*])

All pending input is processed, and a bytes object containing the remaining compressed output is returned. *mode* can be selected from the constants **`Z_NO_FLUSH`**, **`Z_PARTIAL_FLUSH`**, **`Z_SYNC_FLUSH`**, **`Z_FULL_FLUSH`**, **`Z_BLOCK`**, or **`Z_FINISH`**, defaulting to **`Z_FINISH`**. Except **`Z_FINISH`**, all constants allow compressing further bytestrings of data, while **`Z_FINISH`** finishes the compressed stream and prevents compressing any more data. After calling **`flush()`** with *mode* set to **`Z_FINISH`**, the **`compress()`** method cannot be called again; the only realistic action is to delete the object.

Compress.copy()

Retorna una copia del objeto compresor. Esto se puede utilizar para comprimir eficientemente un conjunto de datos que comparten un prefijo inicial común.

Distinto en la versión 3.8: Añadido `copy.copy()` y `copy.deepcopy()` para el soporte de objetos de compresión.

Los objetos de descompresión admiten los siguientes métodos y atributos:

Decompress.unused_data

Un objeto de bytes que contiene todos los bytes restantes después de los datos comprimidos. Es decir, esto permanece `b ""` hasta que esté disponible el último byte que contiene datos de compresión. Si la cadena de bytes completa resultó contener datos comprimidos, es `b""`, un objeto de bytes vacío.

Decompress.unconsumed_tail

Un objeto de bytes que contiene datos no procesados por la última llamada al método **`descompress()`** , debido a un desbordamiento del límite del búfer de datos descomprimidos. Estos datos aún no han sido procesados por la biblioteca zlib, por lo que debe enviarlos (potencialmente concatenando los datos nuevamente) mediante una llamada al método **`descompress()`** para obtener la salida correcta.

Decompress.eof

Un valor booleano que indica si se ha alcanzado el final del flujo de datos comprimido.

Esto hace posible distinguir entre un flujo comprimido correctamente y uno incompleto o truncado.

Added in version 3.3.

Decompress.decompress(*data*, *max_length=0*)

Descomprime *data*, retornando un objeto de bytes que contiene al menos parte de los datos descomprimidos en *string*. Estos datos deben concatenarse con la salida producida por cualquier llamada anterior al método `decompress()`. Algunos de los datos de entrada pueden conservarse en búfer internos para su posterior procesamiento.

Si el parámetro opcional *max_length* no es cero, el valor de retorno no será más largo que *max_length*. Esto puede significar que no toda la entrada comprimida puede procesarse; y los datos no consumidos se almacenarán en el atributo `unconsumed_tail`. Esta cadena de bytes debe pasarse a una llamada posterior a `decompress()` si la descompresión ha de continuar. Si *max_length* es cero, toda la entrada se descomprime y `unconsumed_tail` queda vacío.

Distinto en la versión 3.6: max_length puede ser utilizado como argumento de palabras clave.

Decompress.flush([*length*])

Se procesan todas las entradas pendientes y se retorna un objeto de bytes que contiene el resto de los datos que se descomprimirán. Después de llamar a `flush()`, no se puede volver a llamar al método `decompress()`. La única acción posible es eliminar el objeto.

El parámetro opcional *length* establece el tamaño inicial del búfer de salida.

Decompress.copy()

Retorna una copia del objeto de descompresión. Puede usarlo para guardar el estado de la descompresión actual, de modo que pueda regresar rápidamente a esta ubicación más tarde.

Distinto en la versión 3.8: Añadido `copy.copy()` y `copy.deepcopy()` para el soporte de objetos de compresión.

The following constants are available to configure compression and decompression behavior:

zlib.DEFLATED

The deflate compression method.

zlib.MAX_WBITS

The maximum window size, expressed as a power of 2. For example, if **`MAX_WBITS`** is `15` it results in a window size of `32 KiB`.

zlib.DEF_MEM_LEVEL

The default memory level for compression objects.

zlib.DEF_BUF_SIZE

The default buffer size for decompression operations.

zlib.Z_NO_COMPRESSION

Compression level `0`; no compression.

Added in version 3.6.

zlib.Z_BEST_SPEED

Compression level `1`; fastest and produces the least compression.

zlib.Z_BEST_COMPRESSION

Compression level `9`; slowest and produces the most compression.

zlib.Z_DEFAULT_COMPRESSION

Default compression level (`-1`); a compromise between speed and compression. Currently equivalent to compression level `6`.

zlib.Z_DEFAULT_STRATEGY

Default compression strategy, for normal data.

zlib.Z_FILTERED

Compression strategy for data produced by a filter (or predictor).

zlib.Z_HUFFMAN_ONLY

Compression strategy that forces Huffman coding only.

zlib.Z_RLE

Compression strategy that limits match distances to one (run-length encoding).

This constant is only available if Python was compiled with zlib 1.2.0.1 or greater.

Added in version 3.6.

zlib.Z_FIXED

Compression strategy that prevents the use of dynamic Huffman codes.

This constant is only available if Python was compiled with zlib 1.2.2.2 or greater.

Added in version 3.6.

zlib.Z_NO_FLUSH

Flush mode `0`. No special flushing behavior.

Added in version 3.6.

zlib.Z_PARTIAL_FLUSH

Flush mode `1`. Flush as much output as possible.

zlib.Z_SYNC_FLUSH

Flush mode `2`. All output is flushed and the output is aligned to a byte boundary.

zlib.Z_FULL_FLUSH

Flush mode `3`. All output is flushed and the compression state is reset.

zlib.Z_FINISH

Flush mode `4`. All pending input is processed, no more input is expected.

zlib.Z_BLOCK

Flush mode `5`. A deflate block is completed and emitted.

This constant is only available if Python was compiled with zlib 1.2.2.2 or greater.

Added in version 3.6.

zlib.Z_TREES

Flush mode `6`, for inflate operations. Instructs inflate to return when it gets to the next deflate block boundary.

This constant is only available if Python was compiled with zlib 1.2.3.4 or greater.

Added in version 3.6.

La información sobre la versión de la biblioteca zlib en uso está disponible a través de las siguientes constantes:

zlib.ZLIB_VERSION

Versión de la biblioteca zlib utilizada al compilar el módulo. Puede ser diferente de la biblioteca zlib utilizada actualmente por el sistema, que puede ver en **`ZLIB_RUNTIME_VERSION`**.

zlib.ZLIB_RUNTIME_VERSION

Cadena que contiene la versión de la biblioteca zlib utilizada actualmente por el intérprete.

Added in version 3.3.

zlib.ZLIBNG_VERSION

The version string of the zlib-ng library that was used for building the module if zlib-ng was used. When present, the **`ZLIB_VERSION`** and **`ZLIB_RUNTIME_VERSION`** constants reflect the version of the zlib API provided by zlib-ng.

If zlib-ng was not used to build the module, this constant will be absent.

Added in version 3.14.

Ver también

Módulo **`gzip`**
Lectura y escritura de los archivos en formato **gzip**.

https://www.zlib.net
Página oficial de la biblioteca zlib.

https://www.zlib.net/manual.html
El manual de zlib explica la semántica y el uso de las numerosas funciones de la biblioteca.

In case gzip (de)compression is a bottleneck, the python-isal [https://github.com/pycompression/python-isal] package speeds up (de)compression with a mostly compatible API.

`gzip` — Support for gzip files

Código fuente: Lib/gzip.py [https://github.com/python/cpython/tree/3.14/Lib/gzip.py]

Este módulo proporciona una interfaz simple para comprimir y descomprimir archivos como lo harían los programas GNU **gzip** y **gunzip**.

This is an optional module. If it is missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

La compresión de datos la proporciona el módulo `zlib`.

The `gzip` module provides the `GzipFile` class, as well as the `open()`, `compress()` and `decompress()` convenience functions. The `GzipFile` class reads and writes **gzip**-format files, automatically compressing or decompressing the data so that it looks like an ordinary file object.

Tenga en cuenta que los formatos de archivo adicionales que pueden ser descomprimidos por los programas **gzip** y **gunzip**, como los producidos por **compress** y **pack**, no son compatibles con este módulo.

El módulo define los siguientes elementos:

gzip.open(*filename*, *mode='rb'*, *compresslevel=9*, *encoding=None*, *errors=None*, *newline=None*)

Abre un archivo comprimido gzip en modo binario o de texto, retornando un file object.

El argumento *filename* puede ser un nombre de archivo real (un objeto `str` o `bytes`), o un objeto archivo existente para leer o escribir.

El argumento *mode* puede ser cualquiera de `'r'`, `'rb'`, `'a'`, `'ab'`, `'w'`, `'wb'`, `'x'` or `'xb'` para el modo binario, o `'rt'`, `'at'`, `'wt'`, or `'xt'` para el modo texto. El valor predeterminado es `'rb'`.

El argumento *compresslevel* es un número entero de 0 a 9, como para el constructor **`GzipFile`**.

Para el modo binario, esta función es equivalente al constructor **`GzipFile`**: `GzipFile (filename, mode, compresslevel)`. En este caso, no se deben proporcionar los argumentos *encoding*, *errors* y *newline*.

Para el modo texto, se crea un objeto **`GzipFile`** y se envuelve en una instancia **`io.TextIOWrapper`** con la codificación especificada, comportamiento de manejo de errores y terminación(es) de línea.

Distinto en la versión 3.3: Se agregó soporte para que *filename* sea un objeto de archivo, soporte para el modo de texto y los argumentos *encoding*, *errors* y *newline*.

Distinto en la versión 3.4: Se agregó soporte para los modos `'x'`, `'xb'` y `'xt'`.

Distinto en la versión 3.6: Acepta un path-like object.

exception gzip.BadGzipFile

An exception raised for invalid gzip files. It inherits from **`OSError`**. **`EOFError`** and **`zlib.error`** can also be raised for invalid gzip files.

Added in version 3.8.

class gzip.GzipFile(*filename=None*, *mode=None*, *compresslevel=9*, *fileobj=None*, *mtime=None*)

Constructor para la clase **`GzipFile`**, que simula la mayoría de los métodos de file object, con la excepción del método `truncate()`. Al menos uno de *fileobj* y *filename* debe recibir un valor no trivial.

La nueva instancia de clase se basa en *fileobj*, que puede ser un archivo normal, un objeto `io.BytesIO`, o cualquier otro objeto que simule un archivo. El valor predeterminado es `None`, en cuyo caso se abre *filename* para proporcionar un objeto de archivo.

Cuando *fileobj* no es `None`, el argumento *filename* solo se usa para ser incluido en el encabezado del archivo **gzip**, que puede incluir el nombre de archivo original del archivo sin comprimir. De forma predeterminada, el nombre de archivo es de *fileobj*, si es discernible; de lo contrario, el valor predeterminado es la cadena vacía, y en este caso el nombre del archivo original no se incluye en el encabezado.

El argumento *mode* puede ser cualquiera de `'r'`, `'rb'`, `'a'`, `'ab'`, `'w'`, `'wb'`, `'x'`, or `'xb'`, dependiendo de si el archivo se leerá o escribirá. El valor predeterminado es el modo de *fileobj* si es discernible; de lo contrario, el valor predeterminado es `'rb'`. En futuras versiones de Python, no se utilizará el modo *fileobj*. Es mejor especificar siempre *mode* para escribir.

Tenga en cuenta que el archivo siempre se abre en modo binario. Para abrir un archivo comprimido en modo de texto, utilice `open()` (o ajuste su `GzipFile` con un `io.TextIOWrapper`).

El argumento *compresslevel* es un entero de `0` a `9` que controla el nivel de compresión; `1` es más rápido y produce la menor compresión, y `9` es más lento y produce la mayor compresión. `0` no es una compresión. El valor predeterminado es `9`.

The optional *mtime* argument is the timestamp requested by gzip. The time is in Unix format, i.e., seconds since 00:00:00 UTC, January 1, 1970. If *mtime* is omitted or `None`, the current time is used. Use *mtime* = 0 to generate a compressed stream that does not depend on creation time.

See below for the `mtime` attribute that is set when decompressing.

Calling a `GzipFile` object's `close()` method does not close *fileobj*, since you might wish to append more material after the compressed data. This also allows you to pass an `io.BytesIO` object opened for writing as *fileobj*, and retrieve the resulting memory buffer using the `io.BytesIO` object's `getvalue()` method.

`GzipFile` admite la interfaz `io.BufferedIOBase`, incluida la iteración y la declaración `with`. Solo el método `truncate()` no está implementado.

`GzipFile` también proporciona el siguiente método y atributo:

peek(*n*)

Read *n* uncompressed bytes without advancing the file position. The number of bytes returned may be more or less than requested.

Nota

Al llamar a `peek()` no cambia la posición del archivo `GzipFile`, puede cambiar la posición del objeto de archivo subyacente (por ejemplo, si el `GzipFile` se construyó con el parámetro *fileobj*).

Added in version 3.2.

mode

`'rb'` for reading and `'wb'` for writing.

Distinto en la versión 3.13: In previous versions it was an integer `1` or `2`.

mtime

When decompressing, this attribute is set to the last timestamp in the most recently read header. It is an integer, holding the number of seconds since the Unix epoch (00:00:00 UTC, January 1, 1970). The initial value before reading any headers is `None`.

name

La ruta al archivo gzip en disco, como **`str`** o **`bytes`**. Equivale a la salida de **`os.fspath()`** en la ruta de entrada original, sin ninguna otra normalización, resolución o expansión.

Distinto en la versión 3.1: Se ha agregado soporte para la declaración **`with`**, junto con el argumento del constructor *mtime* y el atributo **`mtime`**.

Distinto en la versión 3.2: Se agregó soporte para archivos con relleno de ceros y que no se pueden buscar.

Distinto en la versión 3.3: El método **`io.BufferedIOBase.read1()`** ahora está implementado.

Distinto en la versión 3.4: Se agregó soporte para los modos `'x'` y `'xb'`.

Distinto en la versión 3.5: Se ha añadido soporte para escribir objetos arbitrarios bytes-like objects. El método **`read()`** ahora acepta un argumento de `None`.

Distinto en la versión 3.6: Acepta un path-like object.

Obsoleto desde la versión 3.9: Abriendo **`GzipFile`** para escribir sin especificar el argumento *mode* está obsoleto.

Distinto en la versión 3.12: Elimine el atributo `filename`, utilice en su lugar el atributo **`name`**`.

gzip.compress(*data*, *compresslevel=9*, ***, *mtime=0*)

Compress the *data*, returning a **`bytes`** object containing the compressed data. *compresslevel* and *mtime* have the same meaning as in the **`GzipFile`** constructor above, but *mtime* defaults to 0 for reproducible output.

Added in version 3.2.

Distinto en la versión 3.8: Se agregó el parámetro *mtime* para una salida reproducible.

Distinto en la versión 3.11: Speed is improved by compressing all data at once instead of in a streamed fashion. Calls with *mtime* set to `0` are delegated to **`zlib.compress()`** for better speed. In this situation the output may contain a gzip header «OS» byte value other than 255 «unknown» as supplied by the underlying zlib implementation.

Distinto en la versión 3.13: The gzip header OS byte is guaranteed to be set to 255 when this function is used as was the case in 3.10 and earlier.

Distinto en la versión 3.14: The *mtime* parameter now defaults to 0 for reproducible output. For the previous behaviour of using the current time, pass `None` to *mtime*.

gzip.decompress(*data*)

Descomprime *data*, retornando un objeto **`bytes`** que contiene los datos sin comprimir. Esta función es capaz de descomprimir datos gzip de varios miembros (múltiples bloques gzip concatenados entre sí). Cuando se está seguro de que los datos contienen un solo miembro, la función **`zlib.decompress()`** es más rápida si se establece *wbits* a 31.

Added in version 3.2.

Distinto en la versión 3.11: La velocidad se mejora al descomprimir los miembros simultáneamente en memoria en lugar de hacerlo de manera continua.

Ejemplos de uso

Ejemplos de como leer un archivo comprimido:

```
import gzip
```

```
with gzip.open('/home/joe/file.txt.gz', 'rb') as f:
    file_content = f.read()
```

Ejemplos de como crear un archivo comprimido GZIP:

```
import gzip
content = b"Lots of content here"
with gzip.open('/home/joe/file.txt.gz', 'wb') as f:
    f.write(content)
```

Ejemplos de como GZIP comprime un archivo existente:

```
import gzip
import shutil
with open('/home/joe/file.txt', 'rb') as f_in:
    with gzip.open('/home/joe/file.txt.gz', 'wb') as f_
        shutil.copyfileobj(f_in, f_out)
```

Ejemplo de como GZIP comprime una cadena binaria:

```
import gzip
s_in = b"Lots of content here"
s_out = gzip.compress(s_in)
```

Ver también

Módulo `zlib`
El módulo básico de compresión de datos necesario para admitir el formato de archivo **gzip**.

In case gzip (de)compression is a bottleneck, the python-isal [https://github.com/pycompression/python-isal] package speeds up (de)compression with a mostly compatible API.

Command-line interface

The **`gzip`** module provides a simple command line interface to compress or decompress files.

Once executed the `gzip` module keeps the input file(s).

Distinto en la versión 3.8: Agrega una nueva interfaz de línea de comandos con un uso. De forma predeterminada, cuando ejecutará la CLI, el nivel de compresión predeterminado es 6.

Command-line options

file

Si no se especifica *file*, lee de `sys.stdin`.

--fast

Indica el método de compresión más rápido (menos compresión).

--best

Indica el método de compresión más lento (mejor compresión).

-d, --decompress

Descomprime el archivo dado.

-h, --help

Muestra el mensaje de ayuda.

`bz2` — Support for bzip2 compression

Código fuente: Lib/bz2.py [https://github.com/python/cpython/tree/3.14/Lib/bz2.py]

Este módulo proporciona una interfaz completa para comprimir y descomprimir datos utilizando el algoritmo de compresión bzip2.

The `bz2` module contains:

- La función `open()` y la clase `BZ2File` para leer y escribir archivos comprimidos.
- Las clases `BZ2Compressor` y `BZ2Decompressor` para la (de)compresión incremental.
- Las funciones `compress()` y `decompress()` para una (de)compresión en un solo paso.

This is an optional module. If it is missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

(De)compresión de archivos

bz2.open(*filename*, *mode='rb'*, *compresslevel=9*, *encoding=None*, *errors=None*, *newline=None*)

Abre un archivo comprimido con bzip2 en modo binario o texto, retorna un objeto archivo.

Al igual que con el constructor para `BZ2File`, el argumento *filename* puede ser un nombre de archivo real (un objeto `str` o `bytes`), o un objeto de archivo existente para leer o escribirle.

El argumento *mode* puede ser cualquiera de `'r'`, `'rb'`, `'w'`, `'wb'`, `'x'`, `'xb'`, `'a'` o `'ab'` para modo binario, o `'rt'`, `'wt'`, `'xt'` , o `'a'` para el modo de texto. El valor predeterminado es `'rb'`.

El argumento *compresslevel* es un entero del 1 al 9, como para el constructor de **`BZ2File`**.

Para el modo binario, esta función es equivalente a **`BZ2File`** constructor: `BZ2File(filename, mode, compresslevel=compresslevel)`. En este caso, no se deben proporcionar los argumentos *encoding*, *errors* y *newline* (nueva linea).

Para el modo de texto, se crea un objeto **`BZ2File`**, y se envuelve en una instancia **`io.TextIOWrapper`** con la codificación especificada, el comportamiento de manejo de errores y los final(es) de línea.

Added in version 3.3.

Distinto en la versión 3.4: El modo `'x'` (creación exclusiva) ha sido agregado.

Distinto en la versión 3.6: Acepta un objeto similar a una ruta (path-like object).

class bz2.BZ2File(*filename*, *mode*=*'r'*, ***, *compresslevel*=*9*)

Abre un archivo comprimido con bzip2 en modo binario.

Si *filename* es un objeto tipo **`str`** o **`bytes`**, abre el archivo nombrado directamente. De lo contrario, *filename* debería ser un file object, que se utilizará para leer o escribir los datos comprimidos.

El argumento *mode* puede ser `'r'` para lectura (predeterminado), `'w'` para sobrescribir, `'x'` para creación exclusiva o `'a'` para anexar. Estos se pueden dar de manera equivalente como `'rb'`, `'wb'`, `'xb'` y `'ab'` respectivamente.

Si *filename* es un objeto de archivo (en lugar de un nombre de archivo real), el modo `'w'` no trunca el archivo, y es equivalente a `'a'`.

Si *mode* es `'w'` o `'a'`, *compresslevel* puede ser un número entero entre `1` y `9` especificando el nivel de compresión: `1` produce la menor compresión y `9` (predeterminado) produce la mayor compresión.

Si *mode* es `'r'`, el archivo de entrada puede ser la concatenación de múltiples flujos (streams) comprimidos.

`BZ2File` proporciona todos los miembros especificados por `io.BufferedIOBase`, excepto `detach()` y `truncate()`. La iteración y la instrucción `with` están soportadas.

`BZ2File` also provides the following methods and attributes:

peek([*n*])

Retorna datos almacenados en el búfer sin avanzar la posición del archivo. Se retornará al menos un byte de datos (a menos que sea EOF). El número exacto de bytes retornados no está especificado.

Nota

Al invocar `peek()` no cambia la posición del archivo de `BZ2File`, puede cambiar la posición del objeto de archivo subyacente (por ejemplo, si **`BZ2File`** se construyó pasando un objeto de archivo a *filename*).

Added in version 3.3.

fileno()

Return the file descriptor for the underlying file.

Added in version 3.3.

readable()

Return whether the file was opened for reading.

Added in version 3.3.

seekable()

Return whether the file supports seeking.

Added in version 3.3.

writable()

Return whether the file was opened for writing.

Added in version 3.3.

read1(*size=-1*)

Read up to *size* uncompressed bytes, while trying to avoid making multiple reads from the underlying stream. Reads up to a buffer's worth of data if size is negative.

Returns `b''` if the file is at EOF.

Added in version 3.3.

readinto(*b*)

Read bytes into *b*.

Returns the number of bytes read (0 for EOF).

Added in version 3.3.

mode

`'rb'` for reading and `'wb'` for writing.

Added in version 3.13.

name

The bzip2 file name. Equivalent to the **`name`** attribute of the underlying file object.

Added in version 3.13.

Distinto en la versión 3.1: Se agregó soporte para la declaración **`with`**.

Distinto en la versión 3.3: Se agregó soporte para *filename* siendo un objeto de archivo (file object) en lugar de un nombre de archivo real.

Se agregó el modo `'a'` (agregar), junto con el soporte para leer archivos de flujo múltiple (multi-stream).

Distinto en la versión 3.4: El modo `'x'` (creación exclusiva) ha sido agregado.

Distinto en la versión 3.5: El método **`read()`** ahora acepta el argumento `None`.

Distinto en la versión 3.6: Acepta un objeto similar a una ruta (path-like object).

Distinto en la versión 3.9: El parámetro *buffering* ha sido retirado. Desde Python 3.0 era ignorado y estaba obsoleto. Pasa un objeto archivo abierto para controlar cómo el archivo es abierto.

El parámetro *compresslevel* se convirtió en un argumento sólo por palabra clave.

Distinto en la versión 3.10: Esta clase es insegura en hilos frente a múltiples lectores o escritores, al igual que sus clases equivalentes en **`gzip`** y **`lzma`** siempre lo han sido.

(De)compresión incremental

class bz2.BZ2Compressor(*compresslevel=9*)

Crea un nuevo objeto compresor. Este objeto puede usarse para comprimir datos de forma incremental. Para comprimir en un solo paso, use la función `compress()` en su lugar.

compresslevel, si se proporciona, debe ser un número entero entre `1` y `9`. El valor predeterminado es `9`.

compress(*data*)

Provee datos al objeto del compresor. Retorna un fragmento de datos comprimidos si es posible, o una cadena de bytes vacía de lo contrario.

Cuando haya terminado de proporcionar datos al compresor, llame al método `flush()` para finalizar el proceso de compresión.

flush()

Termina el proceso de compresión. Retorna los datos comprimidos que quedan en los búferes internos.

El objeto compresor no puede usarse después de que se haya llamado a este método.

class bz2.BZ2Decompressor

Crea un nuevo objeto descompresor. Este objeto puede usarse para descomprimir datos de forma incremental. Para descomprimir en un solo paso, use la función `decompress()` en su lugar.

Nota

Esta clase no maneja de forma transparente las entradas que contienen múltiples flujos comprimidos, a diferencia de `decompress()` y `BZ2File`. Si necesitas descomprimir una entrada de flujo múltiple con `BZ2Decompressor`, debe usar un nuevo descompresor para cada flujo (stream).

decompress(*data*, *max_length=-1*)

Descomprime *datos* (un bytes-like object), retornando datos sin comprimir como bytes. Algunos de los *datos* pueden almacenarse internamente para su uso en llamadas posteriores a `decompress()`. Los datos retornados deben concatenarse con la salida de cualquier llamada anterior a `decompress()`.

Si *max_length* no es negativo, retorna como máximo *max_length* bytes de datos descomprimidos. Si se alcanza este límite y se pueden producir más resultados, el atributo `needs_input` se establecerá en `False`. En este caso, la siguiente llamada a `decompress()` puede proporcionar *datos* como `b''` para obtener más de la salida.

Si todos los datos de entrada se descomprimieron y retornaron (ya sea porque esto era menor que *max_length* bytes, o porque *max_length* era negativo), el atributo `needs_input` se establecerá en `True`.

Intentar descomprimir datos una vez que se alcanza el final del flujo genera un `EOFError`. Cualquier información encontrada después del final del flujo se ignora y se guarda en el atributo `unused_data`.

Distinto en la versión 3.5: Añadido el parámetro *max_length*.

eof

`True` si se ha alcanzado el marcador de fin de flujo.

Added in version 3.3.

unused_data

Datos encontrados después del final del flujo comprimido.

Si se accede a este atributo antes de que se haya alcanzado el final del flujo, su valor será `b''`.

needs_input

`False` si el método **`decompress()`** puede proporcionar más datos descomprimidos antes de requerir una nueva entrada sin comprimir.

Added in version 3.5.

(Des)comprimir en un solo paso

bz2.compress(*data*, *compresslevel=9*)

Comprime *datos*, un objetos tipo binarios.

compresslevel, si se proporciona, debe ser un número entero entre `1` y `9`. El valor predeterminado es `9`.

Para compresión incremental, use **`BZ2Compressor`** en su lugar.

bz2.decompress(*data*)

Descomprime *datos*, un objetos tipo binarios.

Si *data* es la concatenación de múltiples flujos comprimidos, descomprime todos los flujos.

Para la descompresión incremental, use **`BZ2Decompressor`** en su lugar.

Distinto en la versión 3.3: Se agregó soporte para entradas de flujo múltiple.

Ejemplos de uso

Below are some examples of typical usage of the **`bz2`** module.

Usando **`compress()`** y **`decompress()`** para demostrar una compresión de ida y vuelta (*round-trip*):

```
>>> import bz2
>>> data = b"""\
... Donec rhoncus quis sapien sit amet molestie. Fusce s
... nec ullamcorper. Nam rutrum pretium placerat. Aliqua
... sit amet cursus ante. In interdum laoreet mi, sit am
... pulvinar a. Nam gravida euismod magna, non varius ju
... Aliquam pharetra lacus non risus vehicula rutrum. Ma
... felis. Pellentesque semper nunc sit amet nibh ullamc
... dolor luctus. Curabitur lacinia mi ornare consectetu
>>> c = bz2.compress(data)
>>> len(data) / len(c)  # Data compression ratio
1.513595166163142
>>> d = bz2.decompress(c)
>>> data == d  # Check equality to original object after
True
```

Usando `BZ2Compressor` para compresión incremental:

```
>>> import bz2
>>> def gen_data(chunks=10, chunksize=1000):
...     """Yield incremental blocks of chunksize bytes."
...     for _ in range(chunks):
...         yield b"z" * chunksize
...
>>> comp = bz2.BZ2Compressor()
>>> out = b""
>>> for chunk in gen_data():
...     # Provide data to the compressor object
...     out = out + comp.compress(chunk)
...
>>> # Finish the compression process.  Call this once yo
>>> # finished providing data to the compressor.
>>> out = out + comp.flush()
```

El ejemplo anterior utiliza un flujo de datos bastante «no aleatorio» (un flujo de fragmentos `b"z"`). Los datos aleatorios tienden a comprimirse mal, mientras que los datos ordenados y repetitivos generalmente producen un alto grado de compresión.

Escribiendo y leyendo un archivo comprimido con bzip2 en modo binario:

```
>>> import bz2
>>> data = b"""\
... Donec rhoncus quis sapien sit amet molestie. Fusce s
... nec ullamcorper. Nam rutrum pretium placerat. Aliqua
... sit amet cursus ante. In interdum laoreet mi, sit am
... pulvinar a. Nam gravida euismod magna, non varius ju
... Aliquam pharetra lacus non risus vehicula rutrum. Ma
... felis. Pellentesque semper nunc sit amet nibh ullamc
... dolor luctus. Curabitur lacinia mi ornare consectetu
>>> with bz2.open("myfile.bz2", "wb") as f:
...     # Write compressed data to file
...     unused = f.write(data)
...
>>> with bz2.open("myfile.bz2", "rb") as f:
...     # Decompress data from file
...     content = f.read()
...
>>> content == data  # Check equality to original object
True
```

`lzma` — Compression using the LZMA algorithm

Added in version 3.3.

Código fuente: Lib/lzma.py [https://github.com/python/cpython/tree/3.14/Lib/lzma.py]

Este módulo provee clases y funciones de conveniencia para comprimir y descomprimir datos utilizando el algoritmo de compresión LZMA. También se incluye una interfaz de fichero soportando el `.xz` y formatos de fichero `.lzma` heredados utilizados por la utilidad **xz**, así como flujos comprimidos sin procesar.

La interfaz proporcionada por este módulo es muy similar a la del módulo **`bz2`**. Tenga en cuenta que **`LZMAFile`** y **`bz2.BZ2File`** son seguros para subprocesos *not*, por lo que si necesita utilizar una única instancia **`LZMAFile`** de varios subprocesos, es necesario protegerla con un candado.

This is an optional module. If it is missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

exception lzma.LZMAError

Esta excepción es generada cuando un error ocurre durante la compresión o descompresión, o mientras se inicializa el estado de compresor/descompresor.

Leyendo y escribiendo ficheros comprimidos

lzma.open(*filename*, *mode='rb'*, ***, *format=None*, *check=-1*, *preset=None*, *filters=None*, *encoding=None*, *errors=None*, *newline=None*)

Abre un fichero comprimido LZMA in binario o modo texto, retornando un file object.

El argumento *filename* puede ser un nombre de fichero real (dado como un objeto **`str`**, **`bytes`** o path-like), en cuyo caso el fichero nombrado es abierto, o puede ser un objeto de fichero existente para leer o escribir.

El argumento *mode* puede ser cualquiera de `"r"`, `"rb"`, `"w"`, `"wb"`, `"x"`, `"xb"`, `"a"` o `"a"` para modo binario, o `"rt"`, `"wt"`, `"xt"`, o `"at"` para modo texto. Por defecto es `"rb"`.

Al abrir un fichero para lectura, los argumentos *format* y *filters* tienen el mismo significado que para **`LZMADecompressor`**. En este caso, los argumentos *check* y *preset* no deberían ser utilizados.

Al abrir un fichero para lectura, los argumentos *format, check, preset* y *filters* tienen el mismo significado que para **`LZMACompressor`**.

Para modo binario, esta función es equivalente al constructor **`LZMAFile`**: `LZMAFile(filename, mode, ...)`. En este caso, los argumentos *encoding, errors* y *newline* no deben ser proveídos.

Para modo texto, un objeto **`LZMAFile`** es creado, y envuelto en una instancia **`io.TextIOWrapper`** con codificación específica, comportamiento de manejo de errores, y codificación(es) de línea.

Distinto en la versión 3.4: Agregado soporte para los modos `"x"`, `"xb"` y `"xt"`.

Distinto en la versión 3.6: Acepta un path-like object.

class lzma.LZMAFile(*filename=None*, *mode='r'*, ***, *format=None*, *check=-1*, *preset=None*, *filters=None*)

Abre un fichero comprimido LZMA en modo binario.

Un **`LZMAFIle`** puede envolver un ya abierto file object, u operar directamente en un fichero nombrado. El argumento *filename* especifica el objeto de fichero a envolver, o el nombre del fichero para abrir (como un **`str`**, **`bytes`** o un objeto path-like). Al envolver un objeto de fichero existente, el fichero envuelto no será cerrado cuando el **`LZMAFile`** es cerrado.

El argumento *mode* puede ser `"r"` para lectura (por defecto), `"w"` para sobreescritura, `"x"` para creación exclusiva, o `"a"` para agregado, Estos pueden ser equivalentemente dados como `"rb"`, `"wb"`, `"xb"` y `"ab"` respectivamente.

Si *filename* es un objeto de fichero (en lugar de un nombre de fichero actual), un modo de `"w"` no trunca el fichero, y en cambio es equivalente a `"a"`.

Al abrir un fichero para escritura, el fichero de entrada puede ser la concatenación para múltiples flujos comprimidos separados. Estos son transparentemente decodificados como un único flujo lógico.

Al abrir un fichero para lectura, los argumentos *format* y *filters* tienen el mismo significado que para **`LZMADecompressor`**. En este caso, los argumentos *check* y *preset* no deberían ser utilizados.

Al abrir un fichero para lectura, los argumentos *format*, *check*, *preset* y *filters* tienen el mismo significado que para **`LZMACompressor`**.

`LZMAFile` soporta todos los miembros especificados por **`io.BufferedIOBase`**, excepto por **`detach()`** y **`truncate()`**. La iteración y la instrucción **`with`** son soportadas.

The following method and attributes are also provided:

peek(*size=-1*)

Retorna datos almacenados en búfer sin avanzar la posición del fichero. Al menos un byte de datos será retornado, a menos que EOF es alcanzado. El número exacto de bytes retornado no está especificado (el argumento *size* es ignorado).

Nota

Mientras que llamar **`peek()`** no cambia la posición de fichero del **`LZMAFile`**, puede cambiar la posición del objeto de fichero subyacente (por ejemplo si el **`LZMAFile`** fue construido pasando un objeto de fichero por *filename*).

mode

`'rb'` for reading and `'wb'` for writing.

Added in version 3.13.

name

The lzma file name. Equivalent to the **`name`** attribute of the underlying file object.

Added in version 3.13.

Distinto en la versión 3.4: Agregado soporte para los modos `"x"` y `"xb"`.

Distinto en la versión 3.5: El método **`read()`** acepta ahora un argumento de `None`.

Distinto en la versión 3.6: Acepta un path-like object.

Comprimiendo y descomprimiendo datos en memoria

class lzma.LZMACompressor(*format=FORMAT_XZ*, *check=-1*, *preset=None*, *filters=None*)

Crea un objeto compresor, el cual puede ser utilizado para comprimir datos incrementalmente.

Para una forma más conveniente de comprimir un único fragmento de datos, vea `compress()`.

El argumento *format* especifica qué formato de contenedor debería ser utilizado. Posibles valores son:

- **`FORMAT_XZ`**: El formato de contenedor `.xz`.
  Este es el formato por defecto.
- **`FORMAT_ALONE`**: El formato de contenedor `.lzma` heredado.
  Este formato es más limitado que `.xz` – no soporta chequeos de integridad o múltiples filtros.
- **`FORMAT_RAW`**: Un flujo de datos sin procesar, sin utilizar ningún formato de contenedor.
  Este especificador de formato no soporta chequeos de integridad, y requiere que siempre especifiques una cadena de filtro personalizada (para compresión y descompresión). Adicionalmente, los datos comprimidos de esta manera no pueden ser descomprimidos utilizando **`FORMAT_AUTO`** (vea `LZMADecompressor`).

El argumento *check* especifica el tipo de chequeo de integridad a incluir en los datos descomprimidos. Este chequeo es utilizado al descomprimir, para asegurarse que los datos no han sido corrompidos. Los posibles valores son:

- **`CHECK_NONE`**: Sin verificación de integridad. Este es el valor por defecto (y el único valor aceptable) para **`FORMAT_ALONE`** y **`FORMAT_RAW`**.
- **`CHECK_CRC32`**: Chequeo de Redundancia Cíclica de 32 bits.

- **`CHECK_CRC64`**: Chequeo de Redundancia Cíclica de 64 bits. Este es el valor por defecto para **`FORMAT_XZ`**.
- **`CHECK_SHA256`**: Algoritmo Hash Seguro de 256 bits.

Si el chequeo especificado no es soportado, un `LZMAError` es generado.

Las configuraciones de compresión pueden ser especificadas como un nivel de compresión predefinido (con el argumento *preset*), o en detalle como una cadena de filtro personalizada (con el argumento *filters*).

El argumento *preset* (si es proveído) debería ser un entero entre `0` y `9` (inclusive), opcionalmente OR con la constante **`PRESET_EXTREME`**. Si no se proporciona *preset* ni *filters*, el comportamiento por defecto es utilizar **`PRESET_DEFAULT`** (nivel preestablecido `6`). Altos preestablecidos producen salidas más pequeñas, pero vuelve el proceso de compresión más lento.

Nota

En adición a ser más CPU-intensivo, la compresión con preestablecidos más altos también requiere mucha más memoria (y produce salida que necesita más memoria a descomprimir). Con un preestablecido de `9` por ejemplo, la sobrecarga para un objeto `LZMACompressor` puede ser tan alto como 800MiB. Por esta razón, es generalmente mejor quedarse con el preestablecido por defecto.

El argumento *filters* (si es proveído) debería ser un especificador de cadena de filtro. Vea Especificando cadenas de filtro personalizadas para detalles.

compress(*data*)

Comprime *data* (un objeto `bytes`), retornando un objeto **`bytes`** conteniendo información comprimida por al menos parte de la entrada. Algo de *data* puede ser almacenado internamente, para uso en llamadas posteriores a `compress()` y `flush()`. La información retornada debería ser concatenada con la salida en cualquier llamada previa a `compress()`.

flush()

Finiquita el proceso de compresión, retornando un objeto `bytes` conteniendo cualquier información almacenada en los búferes internos del compresor.

El compresor no puede ser utilizado después de que este método es llamado.

class lzma.LZMADecompressor(*format=FORMAT_AUTO*, *memlimit=None*, *filters=None*)

Crea un objeto descompresor, el cual puede ser utilizado para descomprimir datos incrementalmente.

Para una forma más conveniente de descomprimir un flujo completo de compresión a la vez, vea `decompress()`.

El argumento *format* especifica el formato de contenedor que debería ser utilizado. El valor por defecto es `FORMAT_AUTO`, el cual puede descomprimir los ficheros `.xz` y `.lzma`. Otros posibles valores son `FORMAT_XZ`, `FORMAT_ALONE`, y `FORMAT_RAW`.

El argumento *memlimit* especifica un límite (en bytes) en la cantidad de memoria que el descompresor puede utilizar. Cuando este argumento es utilizado, la descompresión fallará con un `LZMAError` si no es posible descomprimir la entrada dentro del límite de memoria dado.

El argumento *filters* especifica la cadena de filtro que fue utilizado para crear el flujo que se descomprime. El argumento es requerido si *format* es `FORMAT_RAW`, pero no debería ser utilizado para otros formatos. Vea Especificando cadenas de filtro personalizadas para más información sobre cadenas de filtro.

Nota

Esta clase no maneja transparentemente entradas que contienen múltiples flujos comprimidos, a diferencia de **`decompress()`** y **`LZMAFile`**. Para descomprimir una entrada multi-flujo con **`LZMADecompressor`**, debería crear un nuevo descompresor para cada flujo.

decompress(*data*, *max_length=-1*)

Descomprime *data* (un bytes-like object), retornando información sin comprimir como bytes. Alguna *data* puede ser almacenada internamente, para uso en llamadas posteriores a **`decompress()`**. La información retornada debería ser concatenada con la salida de cualquier llamada anterior a **`decompress()`**.

Si *max_length* no es negativo, retorna al menos *max_length* bytes para descomprimir información, Si este límite es alcanzado y salidas adicionales pueden ser producidas, el atributo **`needs_input`** será establecido a `False`. En este caso, la siguiente llamada a **`decompress()`** podría proveer *data* como `"b''` para obtener más de la salida.

Si toda la información ingresada fue descomprimida y retornada (ya sea porque esto fue menos que *max_length* bytes, o porque *max_length* fue negativo), el atributo **`needs_input`** será establecido a `True`.

Intentar descomprimir la información después de que el fin del flujo es alcanzado genera un **`EOFError`**. Cualquier información encontrada después del fin del flujo es ignorada y guardada en el atributo **`unused_data`**.

Distinto en la versión 3.5: Agregado el parámetro *max_length*.

check

El ID del chequeo de integridad utilizado por el flujo de entrada. Esto puede ser **`CHECK_UNKNOWN`** hasta que suficiente de la entrada ha sido decodificada para determinar qué chequeo de integridad utiliza.

eof

`True` si el marcador de fin-de-flujo ha sido alcanzado.

unused_data

Información encontrada después del fin del flujo comprimido.

Antes de que el fin del flujo es alcanzado, este será `"b"`.

needs_input

`False` si el método **`decompress()`** puede proveer más información descomprimida antes de requerir nueva entrada descomprimida.

Added in version 3.5.

lzma.compress(*data*, *format=FORMAT_XZ*, *check=-1*, *preset=None*, *filters=None*)

Comprime *data* (un objeto **`bytes`**), retornando la información comprimida como un objeto **`bytes`**.

Vea **`LZMACompressor`** arriba para una descripción de los argumentos *format*, *check*, *preset* y *filters*.

lzma.decompress(*data*, *format=FORMAT_AUTO*, *memlimit=None*, *filters=None*)

Descomprime *data* (un objeto **`bytes`**), retornando la información descomprimida como un objeto **`bytes`**.

Si *data* es la concatenación de múltiples flujos comprimidos distintos, descomprime todos esos flujos, y retorna la concatenación de los resultados.

Vea **`LZMADecompressor`** arriba para una descripción de los argumentos *format*, *memlimit* y *filters*.

Misceláneas

lzma.is_check_supported(*check*)

Retorna `True` si el chequeo de integridad dado es soportado en este sistema.

`CHECK_NONE` y **`CHECK_CRC32`** son soportados siempre. **`CHECK_CRC64`** y **`CHECK_SHA256`** pueden no estar disponibles si está utilizando una versión de **liblzma** que fue compilada con un conjunto de funciones limitado.

Especificando cadenas de filtro personalizadas

Un especificador de cadena de filtro es una secuencia de diccionarios, donde cada diccionario contiene el ID y opciones para un único filtro. Cada diccionario debe contener la llave `"id"`, y puede contener llaves adicionales para especificar opciones filtro-dependientes. Los ID de filtro válidos son como sigue:

- Filtro de compresión:
  - **`FILTER_LZMA1`** (para uso con **`FORMAT_ALONE`**)
  - **`FILTER_LZMA2`** (para uso con **`FORMAT_XZ`** y **`bytesFORMAT_RAW`**)
- Filtro delta:
  - **`FILTER_DELTA`**
- Filtros *Branch-Call-Jump (BCJ)*:
  - **`FILTER_X86`**
  - **`FILTER_IA64`**
  - **`FILTER_ARM`**
  - **`FILTER_ARMTHUMB`**
  - **`FILTER_POWERPC`**
  - **`FILTER_SPARC`**

Una cadena de filtro puede consistir de hasta 4 filtros, y no puede estar vacía. El último filtro en la cadena debe ser un filtro de compresión, y cualquier otro filtro debe ser un filtro delta o BCJ.

Los filtros de compresión soportan las siguientes opciones (especificadas como entradas adicionales:

- `preset`: Un ajuste de compresión a utilizar como una fuente de valores por defecto para opciones que no están especificadas explícitamente.
- `dict_size`: Tamaño del diccionario en bytes. Esto debería estar entre 4 kiB y 1.5 GiB (inclusive).
- `lc` Número de bits de contexto literal.
- `lp`: Número de bits de posición literal. La suma `lc + lp` debe ser al menos 4.
- `pb`: Número de bits de posición; debe ser al menos 4.
- `mode`: **`MODE_FAST`** o **`MODE_NORMAL`**.
- `nice_len`: Lo que debería ser considerado una «buena longitud» para una coincidencia. Esto debería ser 273 o menos.
- `mf`: Qué buscador de coincidencias utilizar – **`MF_HC3`**, **`MF_HC4`**. **`MF_BT2`**, **`MF_BT3`**, o **`MF_BT4`**.
- `depth`: Profundidad de búsqueda máxima utilizada por el buscador de coincidencias. 0 (por defecto) significa seleccionar automáticamente basado en otras opciones de filtro.

El filtro delta almacena las diferencias entre bytes, produciendo más entrada repetitiva para el compresor en ciertas circunstancias. Soporta una opción, `dist`. Esto indica la distancia entre bytes a ser sustraída. Por defecto es 1, por ejemplo toma las diferencias entre bytes adyacentes.

Los filtros BCJ están destinados a ser aplicados a código máquina. Convierten ramas, llamadas y saltos relativos en el código para utilizar el direccionamiento absoluto, con el objetivo de incrementar la redundancia que puede ser explotada por el compresor. Estos filtros soportan una opción, `start_offset`. Esto especifica la dirección que debería ser mapeada al comienzo de la entrada de información. Por defecto es 0.

Ejemplos

Leyendo un fichero comprimido:

```
import lzma
with lzma.open("file.xz") as f:
    file_content = f.read()
```

Creando un fichero comprimido:

```
import lzma
data = b"Insert Data Here"
with lzma.open("file.xz", "w") as f:
    f.write(data)
```

Comprimiendo información en memoria:

```
import lzma
data_in = b"Insert Data Here"
data_out = lzma.compress(data_in)
```

Compresión incremental:

```
import lzma
lzc = lzma.LZMACompressor()
out1 = lzc.compress(b"Some data\n")
out2 = lzc.compress(b"Another piece of data\n")
out3 = lzc.compress(b"Even more data\n")
out4 = lzc.flush()
# Concatenate all the partial results:
result = b"".join([out1, out2, out3, out4])
```

Escribiendo información comprimida en fichero ya abierto:

```
import lzma
with open("file.xz", "wb") as f:
    f.write(b"This data will not be compressed\n")
    with lzma.open(f, "w") as lzf:
        lzf.write(b"This *will* be compressed\n")
    f.write(b"Not compressed\n")
```

Creando un fichero comprimido utilizando una cadena de filtro personalizada:

```
import lzma
my_filters = [
    {"id": lzma.FILTER_DELTA, "dist": 5},
    {"id": lzma.FILTER_LZMA2, "preset": 7 | lzma.PRESET_
]
with lzma.open("file.xz", "w", filters=my_filters) as f:
    f.write(b"blah blah blah")
```

`zipfile` — Work with ZIP archives

Código fuente: Lib/zipfile/ [https://github.com/python/cpython/tree/3.14/Lib/zipfile/]

El formato de archivo ZIP es un estándar común de archivo y compresión. Este módulo proporciona herramientas para crear, leer, escribir, agregar y listar un archivo ZIP. Cualquier uso avanzado de este módulo requerirá una comprensión del formato, tal como se define en la PKZIP Application Note [https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT].

This module does not handle multipart ZIP files. It can handle ZIP files that use the ZIP64 extensions (that is ZIP files that are more than 4 GiB in size). It supports decryption of encrypted files in ZIP archives, but it cannot create an encrypted file. Decryption is extremely slow as it is implemented in native Python rather than C.

Handling compressed archives requires optional modules such as **`zlib`**, **`bz2`**, **`lzma`**, and **`compression.zstd`**. If any of them are missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

El módulo define los siguientes elementos:

exception zipfile.BadZipFile

El error lanzado para archivos ZIP incorrectos.

Added in version 3.2.

exception zipfile.BadZipfile

Alias de **`BadZipFile`**, para compatibilidad con versiones anteriores de Python.

Obsoleto desde la versión 3.2.

exception zipfile.LargeZipFile

El error lanzado cuando un archivo ZIP requiera la funcionalidad ZIP64 pero no ha sido habilitado.

class zipfile.ZipFile

La clase para leer y escribir archivos ZIP. Vea la sección ZipFile objects para detalles del constructor.

class zipfile.Path

Clase que implementa un subconjunto de la interfaz proporcionada por **`pathlib.Path`**, incluyendo la interfaz completa **`importlib.resources.abc.Traversable`**.

Added in version 3.8.

class zipfile.PyZipFile

Clase para crear archivos ZIP que contienen bibliotecas de Python.

class zipfile.ZipInfo(*filename='NoName', date_time=(1980, 1, 1, 0, 0, 0)*)

Class used to represent information about a member of an archive. Instances of this class are returned by the **`getinfo()`** and **`infolist()`** methods of **`ZipFile`** objects. Most users of the **`zipfile`** module will not need to create these, but only use those created by this module. *filename* should be the full name of the archive member, and *date_time* should be a tuple containing six fields which describe the time of the last modification to the file; the fields are described in section ZipInfo objects.

Distinto en la versión 3.13: A public **`compress_level`** attribute has been added to expose the formerly protected **`_compresslevel`**. The older protected name continues to work as a property for backwards compatibility.

_for_archive(*archive*)

Resolve the date_time, compression attributes, and external attributes to suitable defaults as used by **`ZipFile.writestr()`**.

Returns self for chaining.

Added in version 3.14.

zipfile.is_zipfile(*filename*)

Retorna `True` si *filename* es un archivo ZIP válido basado en su número mágico; de lo contrario, retorna `False`. *filename* también puede ser un archivo o un objeto similar a un archivo.

Distinto en la versión 3.1: Soporte para archivos y objetos similares a archivos.

zipfile.ZIP_STORED

La constante numérica para un miembro de archivo sin comprimir.

zipfile.ZIP_DEFLATED

La constante numérica para el método de compresión ZIP habitual. Esto requiere el módulo **`zlib`**.

zipfile.ZIP_BZIP2

La constante numérica para el método de compresión BZIP2. Esto requiere el módulo **`bz2`**.

Added in version 3.3.

zipfile.ZIP_LZMA

La constante numérica para el método de compresión LZMA. Esto requiere el módulo **`lzma`**.

Added in version 3.3.

zipfile.ZIP_ZSTANDARD

The numeric constant for Zstandard compression. This requires the `compression.zstd` module.

Nota

In APPNOTE 6.3.7, the method ID `20` was assigned to Zstandard compression. This was changed in APPNOTE 6.3.8 to method ID `93` to avoid conflicts, with method ID `20` being deprecated. For compatibility, the **`zipfile`** module reads both method IDs but will only write data with method ID `93`.

Added in version 3.14.

Nota

The ZIP file format specification has included support for bzip2 compression since 2001, for LZMA compression since 2006, and Zstandard compression since 2020. However, some tools (including older Python releases) do not support these compression methods, and may either refuse to process the ZIP file altogether, or fail to extract individual files.

Ver también

PKZIP Application Note [https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT]

Documentación sobre el formato de archivo ZIP por Phil Katz, el creador del formato y los algoritmos utilizados.

Página principal de Info-ZIP [https://infozip.sourceforge.net/]

Información sobre los programas de archivo ZIP del proyecto Info-ZIP y las bibliotecas de desarrollo.

ZipFile objects

class zipfile.ZipFile(*file*, *mode='r'*, *compression=ZIP_STORED*, *allowZip64=True*, *compresslevel=None*, ***, *strict_timestamps=True*, *metadata_encoding=None*)

Abra un archivo ZIP, donde *file* puede ser una ruta a un archivo (una cadena), un objeto similar a un archivo o un path-like object.

El parámetro *mode* debe ser `'r'` para leer un archivo existente, `'w'` para truncar y escribir un nuevo archivo, `'a'` para agregarlo a un archivo existente, o `'x'` para crear y escribir exclusivamente un nuevo archivo. Si *mode* es `'x'` y *file* se refiere a un archivo existente, se lanzará un **`FileExistsError`**. Si *mode* es `'a'` y *file* se refiere a un archivo ZIP existente, entonces se le agregan archivos adicionales. Si *file* no se refiere a un archivo ZIP, se agrega un nuevo archivo ZIP al archivo. Esto está destinado a agregar un archivo ZIP a otro archivo (como `python.exe`). Si *mode* es `'a'` y el archivo no existe en absoluto, se crea. Si *mode* es `'r'` o `'a'`, el archivo debe poder buscarse.

compression is the ZIP compression method to use when writing the archive, and should be **`ZIP_STORED`**, **`ZIP_DEFLATED`**, **`ZIP_BZIP2`**, **`ZIP_LZMA`**, or **`ZIP_ZSTANDARD`**; unrecognized values will cause **`NotImplementedError`** to be raised. If **`ZIP_DEFLATED`**, **`ZIP_BZIP2`**, **`ZIP_LZMA`**, or **`ZIP_ZSTANDARD`** is specified but the corresponding module (**`zlib`**, **`bz2`**, **`lzma`**, or **`compression.zstd`**) is not available, **`RuntimeError`** is raised. The default is **`ZIP_STORED`**.

If *allowZip64* is `True` (the default) zipfile will create ZIP files that use the ZIP64 extensions when the zipfile is larger than 4 GiB. If it is `false` **`zipfile`** will raise an exception when the ZIP file would require ZIP64 extensions.

The *compresslevel* parameter controls the compression level to use when writing files to the archive. When using **`ZIP_STORED`** or **`ZIP_LZMA`** it has no effect. When using **`ZIP_DEFLATED`** integers `0` through `9` are accepted (see **`zlib`** for more information). When using **`ZIP_BZIP2`** integers `1` through `9` are accepted (see **`bz2`** for more information). When using **`ZIP_ZSTANDARD`** integers `-131072` through `22` are commonly accepted (see **`CompressionParameter.compression_level`** for more on retrieving valid values and their meaning).

El argumento *strictly_timestamps*, cuando se establece en `False`, permite comprimir archivos anteriores a 1980-01-01 a costa de establecer la marca de tiempo en 1980-01-01. Un comportamiento similar ocurre con archivos más nuevos que 2107-12-31, la marca de tiempo también se establece en el límite.

Cuando el parámetro *mode* es `'r'`, el *metadata_encoding* podría ser asignado al nombre del codec, el cual será usado para decodificar metadata como los nombres de los miembros y comentarios ZIP.

Si el archivo se crea con el modo `'w'`, `'x'` o `'a'` y luego **`closed`** sin agregar ningún archivo al archivo, Las estructuras ZIP apropiadas para un archivo vacío se escribirán en el archivo.

ZipFile también es un manejador de contexto y por lo tanto, admite la declaración **`with`**. En el ejemplo, *myzip* se cierra después que el conjunto de instrucciones **`with`** se termine—incluso si se produce una excepción:

```
with ZipFile('spam.zip', 'w') as myzip:
    myzip.write('eggs.txt')
```

Nota

metadata_encoding is an instance-wide setting for the ZipFile. It is not possible to set this on a per-member basis.

Este atributo es una solución alterna para implementaciones heredadas que producen archivos con nombres basados en la codificación regional o la página de códigos actual (con mayor frecuencia en Windows). De acuerdo con el estándar .ZIP, la codificación de la metadata podría ser especificada por la página de códigos de IBM (por defecto) or por UTF-8 mediante un flag en la cabecera. Este flag toma precedencia sobre *metadata_encoding*, el cual es una extensión específica a Python.

Distinto en la versión 3.2: Se agregó la capacidad de usar **`ZipFile`** como administrador de contexto.

Distinto en la versión 3.3: Soporte agregado para **`bzip2`** y compresión **`lzma`**.

Distinto en la versión 3.4: Las extensiones ZIP64 están habilitadas por defecto.

Distinto en la versión 3.5: Se agregó soporte para escribir en secuencias que no se pueden buscar. Se agregó soporte para el modo `'x'`.

Distinto en la versión 3.6: Anteriormente, se generó un simple **`RuntimeError`** para valores de compresión no reconocidos.

Distinto en la versión 3.6.2: El parámetro *file* acepta un path-like object.

Distinto en la versión 3.7: Agregue el parámetro *compresslevel*.

Distinto en la versión 3.8: The *strict_timestamps* keyword-only parameter.

Distinto en la versión 3.11: Se ha añadido soporte para especificar la codificación del nombre del miembro para leer metadatos en las cabeceras de directorio y archivo del archivo zip.

ZipFile.close()

Cierra el archivo. Debe llamar a **`close()`** antes de salir de su programa o no se escribirán registros esenciales.

ZipFile.getinfo(*name*)

Retorna un objeto **`ZipInfo`** con información sobre el miembro del archivo *name*. Llamando a **`getinfo()`** para obtener un nombre que no figura actualmente en el archivo lanzará un **`KeyError`**.

ZipFile.infolist()

Retorna una lista que contiene un objeto **`ZipInfo`** para cada miembro del archivo. Los objetos están en el mismo orden que sus entradas en el archivo ZIP real en el disco si se abrió un archivo existente.

ZipFile.namelist()

Retorna una lista de miembros del archivo por nombre.

ZipFile.open(*name*, *mode='r'*, *pwd=None*, ***, *force_zip64=False*)

Acceda a un miembro del archivo comprimido como un objeto binario similar a un archivo. *name* puede ser el nombre de un archivo dentro del archivo comprimido o un objeto **`ZipInfo`**. El parámetro *mode*, si está incluido, debe ser `'r'` (el valor predeterminado) o `'w'`. *pwd* es la contraseña utilizada para descifrar archivos ZIP encriptados como un objeto **`bytes`**.

`open()` también es un administrador de contexto y por lo tanto, soporta **`with`** `statement`:

```
with ZipFile('spam.zip') as myzip:
    with myzip.open('eggs.txt') as myfile:
        print(myfile.read())
```

Con *mode* `'r'`, el objeto tipo archivo (`ZipExtFile`) es de solo lectura y proporciona los siguientes métodos: **`read()`**, **`readline()`**, **`readlines()`**, **`seek()`**, **`tell()`**, **`__iter__()`**, **`__ next__()`**. Estos objetos pueden funcionar independientemente del archivo Zip.

Con `mode = 'w'`, se retorna un controlador de archivo escribible, que admite el método **`write()`**. Mientras está abierto un identificador de archivo escribible, intentar leer o escribir otros archivos en el archivo ZIP lanzará un `ValueError`.

In both cases the file-like object has also attributes **`name`**, which is equivalent to the name of a file within the archive, and **`mode`**, which is `'rb'` or `'wb'` depending on the input mode.

Al escribir un archivo, si el tamaño del archivo no se conoce de antemano pero puede exceder los 2 GB, pase `force_zip64 = True` para asegurarse de que el formato del encabezado sea capaz de admitir archivos grandes. Si el tamaño del archivo se conoce de antemano, construya un objeto `ZipInfo` con `file_size` establecido, y úselo como parámetro *name*.

Nota

Los métodos `open()`, `read()` y `extract()` pueden tomar un nombre de archivo o un objeto `ZipInfo`. Apreciará esto cuando intente leer un archivo ZIP que contiene miembros con nombres duplicados.

Distinto en la versión 3.6: Se eliminó el soporte de `mode='U'`. Use `io.TextIOWrapper` para leer archivos de texto comprimido en modo universal newlines.

Distinto en la versión 3.6: `ZipFile.open()` ahora se puede usar para escribir archivos en el archivo comprimido con la opción `mode='w'`.

Distinto en la versión 3.6: Llamar a `open()` en un ZipFile cerrado lanzará un `ValueError`. Anteriormente, se planteó a `RuntimeError`.

Distinto en la versión 3.13: Added attributes **`name`** and **`mode`** for the writeable file-like object. The value of the **`mode`** attribute for the readable file-like object was changed from `'r'` to `'rb'`.

ZipFile.extract(*member*, *path=None*, *pwd=None*)

Extraer un miembro del archivo comprimido al directorio de trabajo actual; *member* debe ser su nombre completo o un objeto `ZipInfo`. La información de su archivo se extrae con la mayor precisión posible. *path* especifica un directorio diferente para extraer. *member* puede ser un nombre de archivo o un objeto `ZipInfo`. *pwd* es la contraseña utilizada para archivos encriptados como un objeto `bytes`.

Retorna la ruta normalizada creada (un directorio o archivo nuevo).

Nota

Si el nombre de archivo de un miembro es una ruta absoluta, se eliminarán un punto compartido de `drive/UNC` y las barras diagonales (hacia atrás), ej: `///foo/bar` se convierte en `foo/bar` en Unix y `C:\foo\bar` se convierte en `foo\bar` en Windows. Y todos los componentes `".."` en un nombre de archivo miembro se eliminarán, ej: `../../foo../../ba..r` se convierte en `foo../ba..r`. En Windows, los caracteres ilegales (`:`, `<`, `>`, `|`, `"`, `?` Y `*`) se reemplazan por guion bajo (`_`).

Distinto en la versión 3.6: Llamando `extract()` en un ZipFile cerrado lanzará un `ValueError`. Anteriormente, se planteó a `RuntimeError`.

Distinto en la versión 3.6.2: El parámetro *path* acepta un path-like object.

ZipFile.extractall(*path=None*, *members=None*, *pwd=None*)

Extrae todos los miembros del archivo comprimido al directorio de trabajo actual. *path* especifica un directorio diferente para extraer. *members* es opcional y debe ser un subconjunto de la lista devuelta por `namelist()`. *pwd* es la contraseña utilizada para archivos encriptados como un objeto `bytes`.

Advertencia

Nunca extraiga archivos de fuentes no confiables sin inspección previa. Es posible que los archivos se creen fuera de *path*, ej. miembros que tienen nombres de archivo absolutos que comienzan con `"/"` o nombres de archivo con dos puntos `".."`. Este módulo intenta evitar eso. Ver **`extract()`** nota.

Distinto en la versión 3.6: Llamar a **`extractall()`** en un ZipFile cerrado lanzará un **`ValueError`**. Anteriormente, se planteó a **`RuntimeError`**.

Distinto en la versión 3.6.2: El parámetro *path* acepta un path-like object.

ZipFile.printdir()

Imprime una tabla de contenido para el archivo en `sys.stdout`.

ZipFile.setpassword(*pwd*)

Establece *pwd* (un objeto **`bytes`**) como contraseña predeterminada para extraer archivos encriptados.

ZipFile.read(*name*, *pwd=None*)

Return the bytes of the file *name* in the archive. *name* is the name of the file in the archive, or a **`ZipInfo`** object. The archive must be open for read or append. *pwd* is the password used for encrypted files as a **`bytes`** object and, if specified, overrides the default password set with **`setpassword()`**. Calling **`read()`** on a ZipFile that uses a compression method other than **`ZIP_STORED`**, **`ZIP_DEFLATED`**, **`ZIP_BZIP2`**, **`ZIP_LZMA`**, or **`ZIP_ZSTANDARD`** will raise a **`NotImplementedError`**. An error will also be raised if the corresponding compression module is not available.

Distinto en la versión 3.6: Llamando **`read()`** en un ZipFile cerrado lanzará un **`ValueError`**. Anteriormente, se planteó a **`RuntimeError`**.

ZipFile.testzip()

Lee todos los archivos en el archivo y verifica sus CRC y encabezados de archivo. Retorna el nombre del primer archivo incorrecto o retorna `None`.

Distinto en la versión 3.6: Llamar a **`testzip()`** en un ZipFile cerrado lanzará un **`ValueError`**. Anteriormente, se planteó a **`RuntimeError`**.

ZipFile.write(*filename*, *arcname=None*, *compress_type=None*, *compresslevel=None*)

Escribe el archivo llamado *filename* en el archivo, dándole el nombre de archivo *arcname* (por defecto, será el mismo que *filename*, pero sin una letra de unidad y con los separadores de ruta principales eliminados). Si se proporciona, *compress_type* anula el valor dado para el parámetro *compression* al constructor para la nueva entrada. Del mismo modo, *compresslevel* anulará el constructor si se proporciona. El archivo debe estar abierto con el modo `'w'`, `'x'` o `'a'`.

Nota

Históricamente, el estándar del archivo ZIP no especificaba una codificación para la metadata, pero si era fuertemente recomendado usar CP437(la codificación original de IBM PC) con fines de interoperabilidad. Versiones recientes permiten el uso de UTF-8 (exclusivamente). En éste modulo, UTF-8 será usada automáticamente para escribir los nombres de los miembros si es que estos contienen algún caracter no-ASCII. No es posible escribir nombres de miembros en otra codificación que no sea ASCII o UTF-8.

Nota

Los nombres de archivo deben ser relativos a la raíz del archivo, es decir, no deben comenzar con un separador de ruta.

Nota

Si `arcname` (o `filename`, si `arcname` no se proporciona) contiene un byte nulo, el nombre del archivo en el archivo se truncará en el byte nulo.

Nota

Un barra al frente en el nombre del archivo puede hacer que el archivo sea imposible de abrir en algunos programas zip en sistemas Windows.

Distinto en la versión 3.6: Llamando **`write()`** en un ZipFile creado con el modo `'r'` o un ZipFile cerrado lanzará un **`ValueError`**. Anteriormente, se planteó a **`RuntimeError`**.

ZipFile.writestr(*zinfo_or_arcname*, *data*, *compress_type=None*, *compresslevel=None*)

Escribe un registro en el archivo. El contenido es *data*, que puede ser una instancia de **`str`** o a **`bytes`**; si es una **`str`**, primero se codifica como UTF-8. *zinfo_or_arcname* es el nombre del archivo que se le dará en el archivo o una instancia de **`ZipInfo`**. Si se trata de una instancia, se debe proporcionar al menos el nombre de archivo, la fecha y la hora. Si es un nombre, la fecha y la hora se configuran en la fecha y hora actuales. El archivo debe abrirse con el modo `'w'`, `'x'` o `'a'`.

Si se proporciona, *compress_type* anula el valor dado para el parámetro *compression* al constructor para la nueva entrada, o en *zinfo_or_arcname* (si es una instancia de **`ZipInfo`**). Del mismo modo, *compresslevel* anulará el constructor si se proporciona.

Nota

Al pasar una instancia de **`ZipInfo`** como el parámetro *zinfo_or_arcname*, el método de compresión utilizado será el especificado en el miembro *compress_type* de la instancia dada **`ZipInfo`**. Por defecto, el constructor **`ZipInfo`** establece este miembro en `ZIP_STORED`.

Distinto en la versión 3.2: El argumento *compress_type*.

Distinto en la versión 3.6: Llamando `writestr()` en un ZipFile creado con el modo `'r'` o un ZipFile cerrado lanzará un `ValueError`. Anteriormente, se planteó a `RuntimeError`.

Distinto en la versión 3.14: Now respects the **SOURCE_DATE_EPOCH** environment variable. If set, it uses this value as the modification timestamp for the file written into the ZIP archive, instead of using the current time.

ZipFile.mkdir(*zinfo_or_directory*, *mode=511*)

Crea un directorio dentro del archivo. Si *zinfo_or_directory* es un string, el directorio es creado dentro del archivo con el modo especificado en el argumento *mode*. Si del contrario, *zinfo_or_directory* es una instancia de `ZipInfo` entonces el argumento *mode* es ignorado.

El archivo debe estar abierto en modo `'w'`, `'x'` o `'a'`.

Added in version 3.11.

Los siguientes atributos de datos también están disponibles:

ZipFile.filename

Nombre del archivo ZIP.

ZipFile.debug

El nivel de salida de depuración a usar. Esto se puede configurar de `0` (el valor predeterminado, sin salida) a `3` (la mayor cantidad de salida). La información de depuración se escribe en `sys.stdout`.

ZipFile.comment

El comentario asociado con el archivo ZIP como un objeto `bytes`. Si se asigna un comentario a una instancia de

`ZipFile` creada con el modo `'w'`, `'x'` o `'a'`, no debe tener más de 65535 bytes. Los comentarios más largos que esto se truncarán.

Path objects

class zipfile.Path(*root*, *at=''*)

Construye un objeto Path a partir de un archivo zip `root` (que puede ser una instancia **`ZipFile`** o `file` adecuado para pasar al constructor **`ZipFile`**).

`at` especifica la ubicación de esta ruta dentro del archivo zip, ej. "dir/file.txt", "dir/" o "".El valor predeterminado es la cadena vacía, que indica la raíz.

Nota

The **`Path`** class does not sanitize filenames within the ZIP archive. Unlike the **`ZipFile.extract()`** and **`ZipFile.extractall()`** methods, it is the caller's responsibility to validate or sanitize filenames to prevent path traversal vulnerabilities (e.g., filenames containing «..» or absolute paths). When handling untrusted archives, consider resolving filenames using **`os.path.abspath()`** and checking against the target directory with **`os.path.commonpath()`**.

Los objetos de ruta exponen las siguientes características de objetos **`pathlib.Path`**:

Los objetos de ruta son transitables utilizando el operador `/` o utilizando `joinpath`.

Path.name

El componente final de la ruta.

Path.open(*mode='r'*, ***, *pwd*, ****)

Invoca **`ZipFile.open()`** en la ruta actual. Permite la apertura para lectura o escritura, texto o binario a través de los modos admitidos: "r", "w", "rb", "wb". Los argumentos posicionales y de palabras clave se pasan a través de **`io.TextIOWrapper`** cuando se abren como texto y se ignoran en caso contrario. `pwd` es el parámetro `pwd` para **`ZipFile.open()`**.

Distinto en la versión 3.9: Se agregó soporte para modos de texto y binarios para abrir. El modo predeterminado ahora es texto.

Distinto en la versión 3.11.2: El parámetro `encoding` puede proporcionarse como un argumento posicional sin causar un **`TypeError`**. Tal y como podía ocurrir en la versión 3.9. El código que necesite ser compatible con versiones no parcheadas 3.10 y 3.11 debe pasar como palabras clave todos los argumentos de **`io.TextIOWrapper`**, incluido `encoding`.

Path.iterdir()

Enumera los hijos del directorio actual.

Path.is_dir()

Retorna `True` si el contexto actual hace referencia a un directorio.

Path.is_file()

Retorna `True` si el contexto actual hace referencia a un archivo.

Path.is_symlink()

Return `True` if the current context references a symbolic link.

Added in version 3.12.

Distinto en la versión 3.13: Previously, `is_symlink` would unconditionally return `False`.

Path.exists()

Retorna `True` si el contexto actual hace referencia a un archivo o directorio en el archivo zip.

Path.suffix

The last dot-separated portion of the final component, if any. This is commonly called the file extension.

Added in version 3.11: Propiedad agregada **`Path.suffix`**.

Path.stem

El componente final de la ruta, sin su sufijo.

Added in version 3.11: Propiedad agregada **`Path.stem`**.

Path.suffixes

A list of the path's suffixes, commonly called file extensions.

Added in version 3.11: Propiedad agregada **`Path.suffixes`**.

Path.read_text(*, **)

Lee el archivo actual como texto unicode. Los argumentos posicionales y de palabras clave se pasan a **`io.TextIOWrapper`** (excepto `buffer`, que está implícito en el contexto).

Distinto en la versión 3.11.2: El parámetro `encoding` puede proporcionarse como un argumento posicional sin causar un **`TypeError`**. Tal y como podía ocurrir en la versión 3.9. El código que necesite ser compatible con versiones no parcheadas 3.10 y 3.11 debe pasar como palabras clave todos los argumentos de **`io.TextIOWrapper`**, incluido `encoding`.

Path.read_bytes()

Lee el archivo actual como bytes.

Path.joinpath(**other*)

Retorna un nuevo objeto de ruta con cada argumentos *other* unidos. Los siguientes son equivalentes:

```
>>> Path(...).joinpath('child').joinpath('grandchil
>>> Path(...).joinpath('child', 'grandchild')
>>> Path(...) / 'child' / 'grandchild'
```

Distinto en la versión 3.10: Antes de 3.10, `joinpath` no estaba documentado y aceptaba exactamente un parámetro.

The zipp [https://pypi.org/project/zipp/] project provides backports of the latest path object functionality to older Pythons. Use `zipp.Path` in place of `zipfile.Path` for early access to changes.

PyZipFile objects

El constructor **`PyZipFile`** toma los mismos parámetros que el constructor **`ZipFile`**, y un parámetro adicional, *optimize*.

class zipfile.PyZipFile(*file*, *mode='r'*, *compression=ZIP_STORED*, *allowZip64=True*, *optimize=-1*)

Distinto en la versión 3.2: Added the *optimize* parameter.

Distinto en la versión 3.4: Las extensiones ZIP64 están habilitadas por defecto.

Las instancias tienen un método ademas de los objetos **`ZipFile`**:

writepy(*pathname*, *basename=''*, *filterfunc=None*)

Busca archivos `*.py` y agrega el archivo correspondiente al archivo.

Si no se proporcionó el parámetro *optimize* a **`PyZipFile`** o `-1`, el archivo correspondiente es un archivo `*.pyc`, compilando si es necesario.

Si el parámetro *optimize* a **`PyZipFile`** era `0`, `1` or `2`, solo se agregarán a ese archivo los archivos con ese nivel de optimización (ver **`compile()`**) el archivo, compilando si es necesario.

Si *pathname* es un archivo, el nombre del archivo debe terminar con `.py`, y solo el archivo (correspondiente `*.pyc`) se agrega en el nivel superior (sin información de ruta). Si *pathname* es un archivo que no termina con `.py`, se lanzará **`RuntimeError`**. Si es un directorio, y el directorio no es un directorio de paquetes, entonces todos los archivos `*.pyc` se agregan en el nivel superior. Si el directorio es un directorio de paquetes, todos `*.pyc` se agregan bajo el nombre del paquete como una ruta de archivo, y si alguno de los subdirectorios son directorios de paquetes, todos estos se agregan recursivamente en orden ordenado.

basename está destinado solo para uso interno.

filterfunc, si se proporciona, debe ser una función que tome un único argumento de cadena. Se le pasará cada ruta (incluida cada ruta de archivo completa individual) antes de que se agregue al archivo. Si *filterfunc* retorna un valor falso, la ruta no se agregará y si se trata de un directorio se ignorará su contenido. Por ejemplo, si nuestros archivos de prueba están todos en directorios de `test` o comienzan con la cadena `test_`, podemos usar un *filterfunc* para excluirlos

```
>>> zf = PyZipFile('myprog.zip')
>>> def notests(s):
...     fn = os.path.basename(s)
...     return (not (fn == 'test' or fn.startsw
...
>>> zf.writepy('myprog', filterfunc=notests)
```

El método **`writepy()`** crea archivos con nombres de archivo como este

```
string.pyc                   # Top level name
```

```
test/__init__.pyc           # Package director
test/testall.pyc            # Module test.test
test/bogus/__init__.pyc     # Subpackage direc
test/bogus/myfile.pyc       # Submodule test.b
```

Distinto en la versión 3.4: Added the *filterfunc* parameter.

Distinto en la versión 3.6.2: El parámetro *pathname* acepta un path-like object.

Distinto en la versión 3.7: La recursividad ordena las entradas del directorio.

ZipInfo objects

Las instancias de la clase **`ZipInfo`** son retornadas por los métodos **`getinfo()`** y **`infolist()`** de **`ZipFile`**. Cada objeto almacena información sobre un solo miembro del archivo ZIP.

Hay un método de clase para hacer una instancia de **`ZipInfo`** para un archivo de sistema de archivos:

classmethod ZipInfo.from_file(*filename*, *arcname=None*, ***, *strict_timestamps=True*)

Construye una instancia de **`ZipInfo`** para un archivo en el sistema de archivos, en preparación para agregarlo a un archivo zip.

filename debe ser la ruta a un archivo o directorio en el sistema de archivos.

Si se especifica *arcname*, este es usado como el nombre dentro del archivo. Si no se especifica *arcname*, el nombre será el mismo que *filename*, pero con cualquier letra de unidad y separadores de ruta principales eliminados.

El argumento *strictly_timestamps*, cuando se establece en `False`, permite comprimir archivos anteriores a 1980-01-01 a costa de establecer la marca de tiempo en 1980-01-01. Un comportamiento similar ocurre con archivos más nuevos que

2107-12-31, la marca de tiempo también se establece en el límite.

Added in version 3.6.

Distinto en la versión 3.6.2: El parámetro *filename* acepta un path-like object.

Distinto en la versión 3.8: Added the *strict_timestamps* keyword-only parameter.

Las instancias tienen los siguientes métodos y atributos:

ZipInfo.is_dir()

Retorna `True` si este miembro del archivo es un directorio.

Utiliza el nombre de la entrada: los directorios siempre deben terminar con `/`.

Added in version 3.6.

ZipInfo.filename

Nombre del archivo en el archivo.

ZipInfo.date_time

The time and date of the last modification to the archive member. This is a tuple of six values representing the «last [modified] file time» and «last [modified] file date» fields from the ZIP file's central directory.

The tuple contains:

Índice	**Valor**
0	Año (> = 1980)
1	Mes (basado en uno)
2	Día del mes (basado en uno)
3	Horas (basados en cero)
4	Minutos (basados en cero)
5	Segundos (basado en cero)

Nota

The ZIP format supports multiple timestamp fields in different locations (central directory, extra fields for NTFS/UNIX systems, etc.). This attribute specifically returns the timestamp from the central directory. The central directory timestamp format in ZIP files does not support timestamps before 1980. While some extra field formats (such as UNIX timestamps) can represent earlier dates, this attribute only returns the central directory timestamp.

The central directory timestamp is interpreted as representing local time, rather than UTC time, to match the behavior of other zip tools.

ZipInfo.compress_type

Tipo de compresión para la miembro del archivo.

ZipInfo.comment

Comenta para el miembro de archivo individual como un objeto `bytes`.

ZipInfo.extra

Datos de campo de expansión. La PKZIP Application Note [https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT] contiene algunos comentarios sobre la estructura interna de los datos contenidos en este objeto `bytes`.

ZipInfo.create_system

Sistema que creó el archivo ZIP.

ZipInfo.create_version

Versión PKZIP que creó el archivo ZIP.

ZipInfo.extract_version

Se necesita la versión PKZIP para extraer el archivo.

ZipInfo.reserved

Debe ser cero.

ZipInfo.flag_bits

Bits de bandera ZIP.

ZipInfo.volume

Número de volumen del encabezado del archivo.

ZipInfo.internal_attr

Atributos internos.

ZipInfo.external_attr

Atributos de archivo externo.

ZipInfo.header_offset

Byte desplazado al encabezado del archivo.

ZipInfo.CRC

CRC-32 del archivo sin comprimir.

ZipInfo.compress_size

Tamaño de los datos comprimidos.

ZipInfo.file_size

Tamaño del archivo sin comprimir.

Command-line interface

The `zipfile` module provides a simple command-line interface to interact with ZIP archives.

Si desea crear un nuevo archivo ZIP, especifique su nombre después de la opción `-c` y luego enumere los nombres de archivo que deben incluirse:

```
$ python -m zipfile -c monty.zip spam.txt eggs.txt
```

Pasar un directorio también es aceptable:

```
$ python -m zipfile -c monty.zip life-of-brian_1979/
```

Si desea extraer un archivo ZIP en el directorio especificado, use la opción `-e`:

```
$ python -m zipfile -e monty.zip target-dir/
```

Para obtener una lista de los archivos en un archivo ZIP, use la opción `-l`:

```
$ python -m zipfile -l monty.zip
```

Opciones de línea de comando

-l <zipfile>
--list <zipfile>

Lista de archivos en un archivo zip.

-c <zipfile> <source1> ... <sourceN>
--create <zipfile> <source1> ... <sourceN>

Crea el archivo zip a partir de archivos fuente.

-e <zipfile> <output_dir>
--extract <zipfile> <output_dir>

Extrae el archivo zip en el directorio de destino.

-t <zipfile>
--test <zipfile>

Prueba si el archivo zip es válido o no.

--metadata-encoding <encoding>

Especifica la codificación de nombre de miembros para `-l`, `-e` y `-t`.

Added in version 3.11.

Problemas de descompresión

La extracción en el módulo zipfile puede fallar debido a algunos problemas que se enumeran a continuación.

Del archivo mismo

La descompresión puede fallar debido a una contraseña incorrecta / suma de verificación CRC / formato ZIP o método / descifrado de compresión no compatible.

File system limitations

Exceder las limitaciones en diferentes sistemas de archivos puede causar que la descompresión falle. Como los caracteres permitidos en las entradas del directorio, la longitud del nombre del archivo, la longitud de la ruta, el tamaño de un solo archivo y la cantidad de archivos, etc.

Limitaciones de recursos

La falta de memoria o volumen de disco conduciría a la descompresión fallida. Por ejemplo, las bombas de descompresión (también conocido como ZIP bomb [https://en.wikipedia.org/wiki/Zip_bomb]) se aplican a la biblioteca de archivos zip que pueden causar el agotamiento del volumen del disco.

Interrupción

La interrupción durante la descompresión, como presionar control-C o matar el proceso de descompresión, puede dar como resultado una descompresión incompleta del archivo.

Comportamientos predeterminados de extracción

No conocer los comportamientos de extracción predeterminados puede causar resultados de descompresión inesperados. Por ejemplo, al extraer el mismo archivo dos veces, sobrescribe los archivos sin preguntar.

`tarfile` — Read and write tar archive files

Código fuente: Lib/tarfile.py [https://github.com/python/cpython/tree/3.14/Lib/tarfile.py]

The **`tarfile`** module makes it possible to read and write tar archives, including those using gzip, bz2 and lzma compression. Use the **`zipfile`** module to read or write `.zip` files, or the higher-level functions in shutil.

Algunos hechos y cifras:

- reads and writes **`gzip`**, **`bz2`**, **`compression.zstd`**, and **`lzma`** compressed archives if the respective modules are available.

  If any of these optional modules are missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

- soporte de lectura/escritura para el formato POSIX.1-1988 (ustar).
- soporte de lectura/escritura para el formato GNU tar incluyendo extensiones *longname* y *longlink*, soporte de solo escritura para todas las variantes de extensiones de *sparse* (archivo disperso) incluyendo restablecimiento de archivos dispersos.
- soporte de lectura/escritura para el formato POSIX.1-2001 (pax).
- gestiona directorios, archivos regulares, enlaces duros,

enlaces simbólicos, fifos, dispositivos de carácter, dispositivos de bloque y puede adquirir y restaurar información de archivo como marca de tiempo, permisos de acceso y dueño.

Distinto en la versión 3.3: Añadido soporte para compresión **`lzma`**.

Distinto en la versión 3.12: Archives are extracted using a filter, which makes it possible to either limit surprising/dangerous features, or to acknowledge that they are expected and the archive is fully trusted.

Distinto en la versión 3.14: Set the default extraction filter to **`data`**, which disallows some dangerous features such as links to absolute paths or paths outside of the destination. Previously, the filter strategy was equivalent to **`fully_trusted`**.

Distinto en la versión 3.14: Added support for Zstandard compression using **`compression.zstd`**.

tarfile.open(*name=None*, *mode='r'*, *fileobj=None*, *bufsize=10240*, ***kwargs*)

Retorna un objeto **`TarFile`** para el nombre de ruta *name*. Para información detallada en los objetos de la clase **`TarFile`** y los argumentos por palabra clave que están permitidos, dirígete a Objetos TarFile.

mode tiene que ser una cadena de texto en el formato `'filemode[:compression]'`, adquiere el valor de `'r'` de forma predeterminada. Aquí hay una lista completa de combinaciones de modo:

modo	**acción**
`'r'` o `'r:*'`	Abrir para leer con compresión transparente (recomendado).
`'r:'`	Abrir para leer exclusivamente sin compresión.
`'r:gz'`	Abrir para leer con compresión gzip.
`'r:bz2'`	Abrir para leer con compresión bzip2.
`'r:xz'`	Abrir para leer con compresión lzma.
`'r:zst'`	Open for reading with Zstandard compression.
`'x'` o `'x:'`	Create a tarfile exclusively without compression. Raise a

	`FileExistsError` exception if it already exists.
`'x:gz'`	Create a tarfile with gzip compression. Raise a **`FileExistsError`** exception if it already exists.
`'x:bz2'`	Create a tarfile with bzip2 compression. Raise a **`FileExistsError`** exception if it already exists.
`'x:xz'`	Create a tarfile with lzma compression. Raise a **`FileExistsError`** exception if it already exists.
`'x:zst'`	Create a tarfile with Zstandard compression. Raise a **`FileExistsError`** exception if it already exists.
`'a'` o `'a:'`	Abrir para anexar sin compresión. El archivo se crea si no existe.
`'w'` o `'w:'`	Abrir para escritura con compresión lzma.
`'w:gz'`	Abrir para escritura con compresión gzip.
`'w:bz2'`	Abrir para escritura con compresión bzip2.
`'w:xz'`	Abrir para escritura con compresión lzma.
`'w:zst'`	Open for Zstandard compressed writing.

Ten en cuenta que `'a:gz'`, `'a:bz2'` or `'a:xz'` no son posibles. Si *mode* no es apropiado para abrir ciertos archivos (comprimidos) para lectura, se lanza una excepción de tipo **`ReadError`**. Usa el *modo* `'r'` para evitar esto. Si el método de compresión no es admitido, se lanza una excepción **`CompressionError`**.

Si *fileobj* es especificado, se utiliza como alternativa a file object abierto en modo binario para *name*. Debe estar en la posición 0.

For modes `'w:gz'`, `'x:gz'`, `'w|gz'`, `'w:bz2'`, `'x:bz2'`, `'w|bz2'`, **`tarfile.open()`** accepts the keyword argument *compresslevel* (default `9`) to specify the compression level of the file.

For modes `'w:xz'`, `'x:xz'` and `'w|xz'`, **`tarfile.open()`** accepts the keyword argument *preset* to specify the compression level of the file.

For modes `'w:zst'`, `'x:zst'` and `'w|zst'`, **`tarfile.open()`** accepts the keyword argument *level* to specify the compression level of the file. The keyword argument *options* may also be passed, providing advanced

Zstandard compression parameters described by **`CompressionParameter`**. The keyword argument *zstd_dict* can be passed to provide a **`ZstdDict`**, a Zstandard dictionary used to improve compression of smaller amounts of data.

For special purposes, there is a second format for *mode*: `'filemode|[compression]'`. **`tarfile.open()`** will return a **`TarFile`** object that processes its data as a stream of blocks. No random seeking will be done on the file. If given, *fileobj* may be any object that has a **`read()`** or **`write()`** method (depending on the *mode*) that works with bytes. *bufsize* specifies the blocksize and defaults to `20 * 512` bytes. Use this variant in combination with e.g. `sys.stdin.buffer`, a socket file object or a tape device. However, such a **`TarFile`** object is limited in that it does not allow random access, see Ejemplos. The currently possible modes:

Modo	**Acción**
`'r|*'`	Abre un *stream* de bloques tar para leer con compresión transparente.
`'r|'`	Abre un *stream* de bloques tar sin comprimir para lectura.
`'r|gz'`	Abre un *stream* comprimido con gzip para lectura.
`'r|bz2'`	Abre un *stream* bzip2 comprimido para lectura.
`'r|xz'`	Abre un *stream* lzma comprimido para lectura.
`'r|zst'`	Open a Zstandard compressed *stream* for reading.
`'w|'`	Abre un *stream* sin comprimir para escritura.
`'w|gz'`	Abre un *stream* gzip comprimido para escritura.
`'w|bz2'`	Abre un *stream* bzip2 comprimido para escritura.
`'w|xz'`	Abre un *stream* lzma comprimido para escritura.
`'w|zst'`	Open a Zstandard compressed *stream* for writing.

Distinto en la versión 3.5: El modo `'x'` (creación exclusiva) fue añadido.

Distinto en la versión 3.6: El parámetro *name* acepta un objeto path-like object.

Distinto en la versión 3.12: The *compresslevel* keyword argument also works for streams.

Distinto en la versión 3.14: The *preset* keyword argument also works for streams.

class tarfile.TarFile

Clase para leer y escribir archivos tar. No utilices esta clase directamente; usa `tarfile.open()` en su lugar. Consulta Objetos TarFile.

tarfile.is_tarfile(*name*)

Return `True` if *name* is a tar archive file, that the **`tarfile`** module can read. *name* may be a `str`, file, or file-like object.

Distinto en la versión 3.9: Soporte para archivos y objetos similares a archivos.

The **`tarfile`** module defines the following exceptions:

exception tarfile.TarError

Base class for all **`tarfile`** exceptions.

exception tarfile.ReadError

Is raised when a tar archive is opened, that either cannot be handled by the **`tarfile`** module or is somehow invalid.

exception tarfile.CompressionError

Se lanza cuando un método de compresión no tiene soporte o cuando la información no puede ser decodificada de manera apropiada.

exception tarfile.StreamError

Se lanza para limitaciones que son comunes para objetos *stream-like* `TarFile`.

exception tarfile.ExtractError

Se lanza para errores no fatales cuando se utiliza `TarFile.extract()`, pero solo si `TarFile.errorlevel`` == 2`.

exception tarfile.HeaderError

Se lanza por `TarInfo.frombuf()` si el buffer que obtiene es invalido.

exception tarfile.FilterError

Base class for members refused by filters.

tarinfo

Information about the member that the filter refused to extract, as TarInfo.

exception tarfile.AbsolutePathError

Raised to refuse extracting a member with an absolute path.

exception tarfile.OutsideDestinationError

Raised to refuse extracting a member outside the destination directory.

exception tarfile.SpecialFileError

Raised to refuse extracting a special file (e.g. a device or pipe).

exception tarfile.AbsoluteLinkError

Raised to refuse extracting a symbolic link with an absolute path.

exception tarfile.LinkOutsideDestinationError

Raised to refuse extracting a symbolic link pointing outside the destination directory.

exception tarfile.LinkFallbackError

Raised to refuse emulating a link (hard or symbolic) by extracting another archive member, when that member would be rejected by the filter location. The exception that was raised to reject the replacement member is available as `BaseException.__context__`.

Added in version 3.14.

Las siguientes constantes están disponibles a nivel de módulo:

tarfile.ENCODING

La codificación para caracteres predeterminada: `'utf-8'` en Windows, de otra manera, el valor retornado por `sys.getfilesystemencoding()`.

tarfile.REGTYPE
tarfile.AREGTYPE

A regular file `type`.

tarfile.LNKTYPE

A link (inside tarfile) `type`.

tarfile.SYMTYPE

A symbolic link `type`.

tarfile.CHRTYPE

A character special device `type`.

tarfile.BLKTYPE

A block special device `type`.

tarfile.DIRTYPE

A directory `type`.

tarfile.FIFOTYPE

A FIFO special device `type`.

tarfile.CONTTYPE

A contiguous file `type`.

tarfile.GNUTYPE_LONGNAME

A GNU tar longname `type`.

tarfile.GNUTYPE_LONGLINK

A GNU tar longlink `type`.

tarfile.GNUTYPE_SPARSE

A GNU tar sparse file `type`.

Each of the following constants defines a tar archive format that the **`tarfile`** module is able to create. See section Formatos tar con soporte for details.

tarfile.USTAR_FORMAT

Formato POSIX.1-1988 (ustar).

tarfile.GNU_FORMAT

Formato GNU tar.

tarfile.PAX_FORMAT

Formato POSIX.1-2001 (pax).

tarfile.DEFAULT_FORMAT

El formato predeterminado para crear archivos. Es actualmente `PAX_FORMAT`.

Distinto en la versión 3.8: El formato predeterminado para nuevos archivos fue cambiado de `GNU_FORMAT` a `PAX_FORMAT`.

Ver también

Módulo `zipfile`

Documentación del módulo estándar `zipfile`.

Operaciones de archivado

Documentación para las facilidades de archivos de más alto nivel proporcionadas por el módulo estándar `shutil`.

Manual GNU tar, formato básico tar [https://www.gnu.org/

software/tar/manual/html_node/Standard.html]

Documentación para archivos de tipo tar, incluyendo extensiones GNU tar.

Objetos *TarFile*

El objeto `TarFile` provee una interfaz a un archivo tar. Un archivo tar es una secuencia de bloques. Un miembro de archivos (un archivo almacenado) está hecho de un bloque de encabezado seguido de bloques de datos. Es posible almacenar un archivo dentro de un tar múltiples veces. Cada archivo miembro está representado por un objeto `TarInfo`, consulta Objetos TarInfo para más detalles.

Un objeto `TarFile` puede ser usado como un administrador de contexto en una declaración `with`. Será automáticamente cerrado cuando el bloque sea completado. Ten en cuenta que en el evento de una excepción un archivo abierto para escritura no será terminado; solo el objeto de archivo interno será cerrado. Consulta la sección Ejemplos para un caso de uso.

Added in version 3.2: Añadido soporte para el protocolo de administración de contexto.

class tarfile.TarFile(*name=None, mode='r', fileobj=None, format=DEFAULT_FORMAT, tarinfo=TarInfo, dereference=False, ignore_zeros=False, encoding=ENCODING, errors='surrogateescape', pax_headers=None, debug=0, errorlevel=1, stream=False*)

Los siguientes argumentos son opcionales y también se pueden acceder como atributos de instancia.

name is the pathname of the archive. *name* may be a path-like object. It can be omitted if *fileobj* is given. In this case, the file object's `name` attribute is used if it exists.

mode puede ser `'r'` para leer desde un archivo existente, `'a'` para adjuntar datos a un archivo existente,``"w"`` para crear un nuevo archivo sobrescribiendo uno existente, o `'x'` para crear un nuevo archivo únicamente si este no existe.

Si *fileobj* es proporcionado, se utiliza para escritura o lectura de datos. Si puede ser determinado, *mode* puede ser anulado por el modo de *fileobj*. *fileobj* será usado desde la posición 0.

Nota

fileobj no se cierra cuando `TarFile` se cierra.

format controla el formato de archivo para la escritura. Debe ser una de las constantes `USTAR_FORMAT`, `GNU_FORMAT` o `PAX_FORMAT` que se definen a nivel de módulo. Al leer, el formato se detectará automáticamente, incluso si hay diferentes formatos en un solo archivo.

El argumento *tarinfo* puede ser utilizado para reemplazar la clase predeterminada `TarInfo` con una diferente.

Si *dereference* tiene el valor de `False`, añade enlaces simbólicos y duros al archivo. Si tiene el valor de `True`, añade el contenido de archivos objetivo al archivo. Esto no tiene ningún efecto en los sistemas que no admiten enlaces simbólicos.

Si *ignore_zeros* tiene el valor de `False`, trata un bloque vacío como el final del archivo. Si es `True`, omite los bloques vacíos (y no válidos) e intenta obtener tantos miembros como sea posible. Esto solo es útil para leer archivos concatenados o dañados.

debug puede ser establecido con valores desde `0` (sin mensajes de depuración) hasta `3` (todos los mensajes de depuración). Los mensajes son escritos en `sys.stderr`.

errorlevel controls how extraction errors are handled, see `the corresponding attribute`.

Los argumentos *encoding* y *errors* definen la codificación de caracteres que se utilizará para lectura o escritura del archivo y como los errores de conversión van a ser manejados. La configuración predeterminada funcionará para la mayoría de los usuarios. Mira la sección Problemas unicode para más información a detalle.

El argumento *pax_headers* es un diccionario opcional de cadenas las cuales serán añadidas como un encabezado pax global si el valor de *format* es `PAX_FORMAT`.

If *stream* is set to `True` then while reading the archive info about files in the archive are not cached, saving memory.

Distinto en la versión 3.2: Utiliza `'surrogateescape'` como valor predeterminado del argumento *errors*.

Distinto en la versión 3.5: El modo `'x'` (creación exclusiva) fue añadido.

Distinto en la versión 3.6: El parámetro *name* acepta un objeto path-like object.

Distinto en la versión 3.13: Add the *stream* parameter.

classmethod TarFile.open(...)

Constructor alternativo. La función `tarfile.open()` es un acceso directo a este método de la clase.

TarFile.getmember(*name*)

Retorna un objeto `TarInfo` para el miembro *name*. Si *name* no puede ser encontrado, entonces una excepción de tipo `KeyError` será lanzada.

Nota

Si un miembro aparece más de una vez en el archivo, se asume que su última aparición es la versión más actualizada.

TarFile.getmembers()

Retorna los miembros del archivo como una lista de objetos `TarInfo`. La lista tiene el mismo orden que los miembros del archivo.

TarFile.getnames()

Retorna los miembros como una lista de sus nombres. Tiene el mismo orden que la lista retornada por `getmembers()`.

TarFile.list(*verbose=True*, ***, *members=None*)

Imprime en `sys.stdout` una tabla de contenidos. Si *verbose* es `False`, solo los nombres de los miembros son impresos. si es `True`, se produce una salida de impresión similar a la de **ls -l**. Si el parámetro *members* es proporcionado, debe ser un subconjunto de la lista retornada por `getmembers()`.

Distinto en la versión 3.5: Se agregó el parámetro *members*.

TarFile.next()

Retorna el siguiente miembro del archivo como un objeto `TarInfo` cuando `TarFile` se abre para lectura. Retorna `None` si no hay más miembros disponibles.

TarFile.extractall(*path='.'*, *members=None*, ***, *numeric_owner=False*, *filter=None*)

Extrae todos los miembros de un archivo al directorio de trabajo actual o al directorio de *path*. Si se proporciona el parámetro opcional *members*, debe ser un subconjunto de la lista retornada por el método `getmembers()`. Información de directorio como dueño, fecha de modificación y permisos son establecidos una vez que todos los miembros han sido extraídos. Esto se realiza para trabajar con dos problemáticas: La fecha de modificación de un directorio es modificada cada vez que un archivo es creado en el. Y si los permisos de un directorio no permiten escritura, extraer archivos en el no será posible.

Si *numeric_owner* tiene el valor de `True`, los números uid y gid del archivo tar se utilizan para establecer el dueño/grupo de los archivos extraídos. De lo contrario, se utilizan los valores nombrados del archivo tar.

The *filter* argument specifies how `members` are modified or rejected before extraction. See Extraction filters for details. It is recommended to set this explicitly only if specific *tar* features are required, or as `filter='data'` to support Python versions with a less secure default (3.13 and lower).

Advertencia

Never extract archives from untrusted sources without prior inspection.

Since Python 3.14, the default (**`data`**) will prevent the most dangerous security issues. However, it will not prevent *all* unintended or insecure behavior. Read the Extraction filters section for details.

Distinto en la versión 3.5: Se agregó el parámetro *numeric_owner*.

Distinto en la versión 3.6: El parámetro *path* acepta a path-like object.

Distinto en la versión 3.12: Se agregó el parámetro *filter*.

Distinto en la versión 3.14: The *filter* parameter now defaults to `'data'`.

TarFile.extract(*member*, *path=''*, *set_attrs=True*, ***, *numeric_owner=False*, *filter=None*)

Extrae un miembro del archivo al directorio de trabajo actual utilizando su nombre completo. La información de su archivo se extrae con la mayor precisión posible. *member* puede ser un nombre de archivo o un objeto **`TarInfo`**. Puedes especificar un directorio diferente utilizando *path*. *path* puede ser un path-like object. Los atributos de archivo (dueño, fecha de modificación, modo) son establecidos a no ser que *set_attrs* sea falso.

The *numeric_owner* and *filter* arguments are the same as for **`extractall()`**.

Nota

El método `extract()` no cuida de varios problemas de extracción. En la mayoría de los casos deberías considerar utilizar el método `extractall()`.

Advertencia

Never extract archives from untrusted sources without prior inspection. See the warning for `extractall()` for details.

Distinto en la versión 3.2: Se agregó el parámetro *set_attrs*.

Distinto en la versión 3.5: Se agregó el parámetro *numeric_owner*.

Distinto en la versión 3.6: El parámetro *path* acepta a path-like object.

Distinto en la versión 3.12: Se agregó el parámetro *filter*.

TarFile.extractfile(*member*)

Extrae un miembro del archivo como un objeto de archivo. *member* puede ser un nombre de archivo o un objeto `TarInfo`. Si *member* es un archivo normal o un enlace, se retorna un objeto `io.BufferedReader`. Para todos los demás miembros existentes, se retorna `None`. Si *member* no aparece en el archivo, se lanza `KeyError`.

Distinto en la versión 3.3: Retorna un objeto `io.BufferedReader`.

Distinto en la versión 3.13: The returned `io.BufferedReader` object has the `mode` attribute which is always equal to `'rb'`.

TarFile.errorlevel*: int*

If *errorlevel* is `0`, errors are ignored when using

`TarFile.extract()` and **`TarFile.extractall()`**. Nevertheless, they appear as error messages in the debug output when *debug* is greater than 0. If `1` (the default), all *fatal* errors are raised as **`OSError`** or **`FilterError`** exceptions. If `2`, all *non-fatal* errors are raised as **`TarError`** exceptions as well.

Some exceptions, e.g. ones caused by wrong argument types or data corruption, are always raised.

Custom extraction filters should raise **`FilterError`** for *fatal* errors and **`ExtractError`** for *non-fatal* ones.

Note that when an exception is raised, the archive may be partially extracted. It is the user's responsibility to clean up.

TarFile.extraction_filter

Added in version 3.12.

The extraction filter used as a default for the *filter* argument of **`extract()`** and **`extractall()`**.

The attribute may be `None` or a callable. String names are not allowed for this attribute, unlike the *filter* argument to **`extract()`**.

If `extraction_filter` is `None` (the default), extraction methods will use the **`data`** filter by default.

The attribute may be set on instances or overridden in subclasses. It also is possible to set it on the `TarFile` class itself to set a global default, although, since it affects all uses of *tarfile*, it is best practice to only do so in top-level applications or **`site configuration`**. To set a global default this way, a filter function needs to be wrapped in **`staticmethod()`** to prevent injection of a `self` argument.

Distinto en la versión 3.14: The default filter is set to **`data`**, which disallows some dangerous features such as links to absolute paths or paths outside of the destination. Previously, the default was equivalent to **`fully_trusted`**.

TarFile.add(*name*, *arcname=None*, *recursive=True*, ***, *filter=None*)

Añade el archivo *name* al archivo. *name* puede ser cualquier tipo de archivo (directorio, fifo, enlace simbólico, etc.). Si se proporciona, *arcname* especifica un nombre alternativo para el archivo en el archivo. Los directorios se agregan de forma recursiva de forma predeterminada. Esto se puede evitar estableciendo *recursive* con el valor de `False`. La recursividad agrega entradas en orden ordenado. Si se proporciona *filter*, debería ser una función que tome un argumento de objeto `TarInfo` y retorna el objeto `TarInfo` modificado. Si en cambio retorna `None`, el objeto `TarInfo` será excluido del archivo. Consulta Ejemplos para ver un ejemplo.

Distinto en la versión 3.2: Se agregó el parámetro *filter*.

Distinto en la versión 3.7: La recursividad agrega entradas en orden ordenado.

TarFile.addfile(*tarinfo*, *fileobj=None*)

Add the `TarInfo` object *tarinfo* to the archive. If *tarinfo* represents a non zero-size regular file, the *fileobj* argument should be a binary file, and `tarinfo.size` bytes are read from it and added to the archive. You can create `TarInfo` objects directly, or by using `gettarinfo()`.

Distinto en la versión 3.13: *fileobj* must be given for non-zero-sized regular files.

TarFile.gettarinfo(*name=None*, *arcname=None*, *fileobj=None*)

Crea un objeto `TarInfo` a partir del resultado de `os.stat()` o equivalente en un archivo existente. El archivo es nombrado por *name* o especificado como un objeto file object *fileobj* con un descriptor de archivo. *name* puede ser un objeto path-like object. Si es proporcionado, *arcname* especifica un nombre alternativo para el archivo en el archivo, de otra forma, el nombre es tomado del atributo *fileobj*'s `name`, o el argumento *name*. El nombre puede ser una cadena de texto.

Puedes modificar algunos de los atributos de la clase `TarInfo`’ antes de que lo añadas utilizado el método `addfile()`. Si el archivo objeto no es un archivo objeto ordinario posicionado al principio del archivo, atributos como `size` puede que necesiten modificaciones. Este es el caso para objetos como `GzipFile`. El atributo `name` también puede ser modificado, en cuyo caso *arcname* podría ser una cadena ficticia.

Distinto en la versión 3.6: El parámetro *name* acepta un objeto path-like object.

TarFile.close()

Cierra `TarFile`. En el modo de escritura, se añaden dos bloques de cero de finalización al archivo.

TarFile.pax_headers: *dict*

Un diccionario que contiene pares claves-valor de los encabezados globales pax.

Objetos TarInfo

Un objeto `TarInfo` representa un miembro en `TarFile`. Además de almacenar todos los atributos requeridos de un archivo (como tipo de archivo, tamaño, fecha, permisos, dueño, etc.), provee de algunos métodos útiles para determinar su tipo. No contiene los datos del archivo por si mismo.

`TarInfo` objects are returned by `TarFile`’s methods `getmember()`, `getmembers()` and `gettarinfo()`.

Modifying the objects returned by `getmember()` or `getmembers()` will affect all subsequent operations on the archive. For cases where this is unwanted, you can use `copy.copy()` or call the `replace()` method to create a modified copy in one step.

Several attributes can be set to `None` to indicate that a piece of metadata is unused or unknown. Different `TarInfo` methods handle `None` differently:

- The **`extract()`** or **`extractall()`** methods will ignore the corresponding metadata, leaving it set to a default.
- **`addfile()`** will fail.
- **`list()`** will print a placeholder string.

class tarfile.TarInfo(*name=''*)

Crea un objeto **`TarInfo`**.

classmethod TarInfo.frombuf(*buf*, *encoding*, *errors*)

Crea y retorna un objeto **`TarInfo`**.

Lanza una excepción **`HeaderError`** si el buffer es invalido.

classmethod TarInfo.fromtarfile(*tarfile*)

Lee el siguiente miembro del objeto de la clase **`TarFile`** *tarfile* y lo retorna como un objeto **`TarInfo`**.

TarInfo.tobuf(*format=DEFAULT_FORMAT*, *encoding=ENCODING*, *errors='surrogateescape'*)

Crear un *buffer* de cadena a partir de un objeto **`TarInfo`**. Para información sobre los argumentos consulta el constructor de la clase **`TarFile`**.

Distinto en la versión 3.2: Utiliza `'surrogateescape'` como valor predeterminado del argumento *errors*.

Un objeto `TarInfo` tiene los siguientes atributos de dato públicos:

TarInfo.name: *str*

Nombre del archivo miembro.

TarInfo.size: *int*

Tamaño en bytes.

TarInfo.mtime: *int* | *float*

Time of last modification in seconds since the epoch, as in **`os.stat_result.st_mtime`**.

Distinto en la versión 3.12: Can be set to `None` for **`extract()`** and **`extractall()`**, causing extraction to skip applying this attribute.

TarInfo.mode*: int*

Permission bits, as for **`os.chmod()`**.

Distinto en la versión 3.12: Can be set to `None` for **`extract()`** and **`extractall()`**, causing extraction to skip applying this attribute.

TarInfo.type

Tipo de archivo. *type* es por lo general una de las siguientes constantes: **`REGTYPE`**, **`AREGTYPE`**, **`LNKTYPE`**, **`SYMTYPE`**, **`DIRTYPE`**, **`FIFOTYPE`**, **`CONTTYPE`**, **`CHRTYPE`**, **`BLKTYPE`**, **`GNUTYPE_SPARSE`**. Para determinar el tipo de un objeto **`TarInfo`** de forma más conveniente, utiliza los métodos `is*()` de abajo.

TarInfo.linkname*: str*

Nombre del archivo objetivo, el cual solo está presente en los objetos **`TarInfo`** de tipo **`LNKTYPE`** y del tipo **`SYMTYPE`**.

For symbolic links (`SYMTYPE`), the *linkname* is relative to the directory that contains the link. For hard links (`LNKTYPE`), the *linkname* is relative to the root of the archive.

TarInfo.uid*: int*

ID de usuario que originalmente almacenó este miembro.

Distinto en la versión 3.12: Can be set to `None` for **`extract()`** and **`extractall()`**, causing extraction to skip applying this attribute.

TarInfo.gid*: int*

ID de grupo del usuario que originalmente almacenó este miembro.

Distinto en la versión 3.12: Can be set to `None` for **`extract()`** and **`extractall()`**, causing extraction to skip applying this attribute.

TarInfo.uname*: str*

Nombre de usuario.

Distinto en la versión 3.12: Can be set to `None` for **`extract()`** and **`extractall()`**, causing extraction to skip applying this attribute.

TarInfo.gname*: str*

Nombre del grupo.

Distinto en la versión 3.12: Can be set to `None` for **`extract()`** and **`extractall()`**, causing extraction to skip applying this attribute.

TarInfo.chksum*: int*

Header checksum.

TarInfo.devmajor*: int*

Device major number.

TarInfo.devminor*: int*

Device minor number.

TarInfo.offset*: int*

The tar header starts here.

TarInfo.offset_data*: int*

The file's data starts here.

TarInfo.sparse

Sparse member information.

TarInfo.pax_headers: *dict*

Un diccionario que contiene pares *key-value* de un encabezado *pax extended*.

TarInfo.replace(*name=...*, *mtime=...*, *mode=...*, *linkname=...*, *uid=...*, *gid=...*, *uname=...*, *gname=...*, *deep=True*)

Added in version 3.12.

Return a *new* copy of the **`TarInfo`** object with the given attributes changed. For example, to return a `TarInfo` with the group name set to `'staff'`, use:

```
new_tarinfo = old_tarinfo.replace(gname='staff')
```

By default, a deep copy is made. If *deep* is false, the copy is shallow, i.e. `pax_headers` and any custom attributes are shared with the original `TarInfo` object.

Un objeto **`TarInfo`** también provee de algunos métodos de consulta convenientes:

TarInfo.isfile()

Return **`True`** if the **`TarInfo`** object is a regular file.

TarInfo.isreg()

Igual que **`isfile()`**.

TarInfo.isdir()

Retorna **`True`** si es un directorio.

TarInfo.issym()

Retorna **`True`** si es un enlace simbólico.

TarInfo.islnk()

Retorna **`True`** si es un enlace duro.

TarInfo.ischr()

Retorna `True` si es un dispositivo de caracter.

TarInfo.isblk()

Retorna `True` si es un dispositivo de bloque.

TarInfo.isfifo()

Retorna `True` si es un FIFO.

TarInfo.isdev()

Retorna `True` si es uno de los caracteres de dispositivo, dispositivo de bloque o FIFO.

Extraction filters

Added in version 3.12.

The *tar* format is designed to capture all details of a UNIX-like filesystem, which makes it very powerful. Unfortunately, the features make it easy to create tar files that have unintended – and possibly malicious – effects when extracted. For example, extracting a tar file can overwrite arbitrary files in various ways (e.g. by using absolute paths, `..` path components, or symlinks that affect later members).

In most cases, the full functionality is not needed. Therefore, *tarfile* supports extraction filters: a mechanism to limit functionality, and thus mitigate some of the security issues.

Advertencia

None of the available filters blocks *all* dangerous archive features. Never extract archives from untrusted sources without prior inspection. See also Hints for further verification.

Ver también

PEP 706 [https://peps.python.org/pep-0706/]

Contains further motivation and rationale behind the design.

The *filter* argument to **`TarFile.extract()`** or **`extractall()`** can be:

- the string `'fully_trusted'`: Honor all metadata as specified in the archive. Should be used if the user trusts the archive completely, or implements their own complex verification.

- the string `'tar'`: Honor most *tar*-specific features (i.e. features of UNIX-like filesystems), but block features that are very likely to be surprising or malicious. See **`tar_filter()`** for details.

- the string `'data'`: Ignore or block most features specific to UNIX-like filesystems. Intended for extracting cross-platform data archives. See **`data_filter()`** for details.

- `None` (default): Use **`TarFile.extraction_filter`**.

  If that is also `None` (the default), the `'data'` filter will be used.

  > *Distinto en la versión 3.14:* The default filter is set to **`data`**. Previously, the default was equivalent to **`fully_trusted`**.

- A callable which will be called for each extracted member with a TarInfo describing the member and the destination path to where the archive is extracted (i.e. the same path is used for all members):

  ```
  filter(member: TarInfo, path: str, /) -> TarInfo |
  ```

  The callable is called just before each member is extracted, so it can take the current state of the disk into account. It can:

  - return a **`TarInfo`** object which will be used instead of the metadata in the archive, or

- ○ return `None`, in which case the member will be skipped, or
- ○ raise an exception to abort the operation or skip the member, depending on **`errorlevel`**. Note that when extraction is aborted, **`extractall()`** may leave the archive partially extracted. It does not attempt to clean up.

Default named filters

The pre-defined, named filters are available as functions, so they can be reused in custom filters:

tarfile.fully_trusted_filter(*member*, *path*)

Return *member* unchanged.

This implements the `'fully_trusted'` filter.

tarfile.tar_filter(*member*, *path*)

Implements the `'tar'` filter.

- Strip leading slashes (`/` and **`os.sep`**) from filenames.
- Refuse to extract files with absolute paths (in case the name is absolute even after stripping slashes, e.g. `C:/foo` on Windows). This raises **`AbsolutePathError`**.
- Refuse to extract files whose absolute path (after following symlinks) would end up outside the destination. This raises **`OutsideDestinationError`**.
- Clear high mode bits (setuid, setgid, sticky) and group/other write bits (**`S_IWGRP`** | **`S_IWOTH`**).

Return the modified `TarInfo` member.

tarfile.data_filter(*member*, *path*)

Implements the `'data'` filter. In addition to what `tar_filter` does:

- Normalize link targets (**`TarInfo.linkname`**) using `os.path.normpath()`. Note that this removes internal `..` components, which may change the meaning of the link if the path in **`TarInfo.linkname`** traverses symbolic links.

- Refuse to extract links (hard or soft) that link to absolute paths, or ones that link outside the destination.

  This raises `AbsoluteLinkError` or `LinkOutsideDestinationError`.

  Note that such files are refused even on platforms that do not support symbolic links.

- Refuse to extract device files (including pipes). This raises `SpecialFileError`.

- For regular files, including hard links:

  - ○ Set the owner read and write permissions (`S_IRUSR` | `S_IWUSR`).
  - ○ Remove the group & other executable permission (`S_IXGRP` | `S_IXOTH`) if the owner doesn't have it (`S_IXUSR`).

- For other files (directories), set `mode` to `None`, so that extraction methods skip applying permission bits.

- Set user and group info (`uid`, `gid`, `uname`, `gname`) to `None`, so that extraction methods skip setting it.

Return the modified `TarInfo` member.

Note that this filter does not block *all* dangerous archive features. See Hints for further verification for details.

Distinto en la versión 3.14: Link targets are now normalized.

Filter errors

When a filter refuses to extract a file, it will raise an appropriate exception, a subclass of `FilterError`. This will abort the extraction if `TarFile.errorlevel` is 1 or more. With `errorlevel=0` the error will be logged and the member will be skipped, but extraction will continue.

Hints for further verification

Even with `filter='data'`, *tarfile* is not suited for extracting untrusted files without prior inspection. Among other issues, the pre-defined filters do not prevent denial-of-service attacks. Users should do additional checks.

Here is an incomplete list of things to consider:

- Extract to a `new temporary directory` to prevent e.g. exploiting pre-existing links, and to make it easier to clean up after a failed extraction.
- Disallow symbolic links if you do not need the functionality.
- When working with untrusted data, use external (e.g. OS-level) limits on disk, memory and CPU usage.
- Check filenames against an allow-list of characters (to filter out control characters, confusables, foreign path separators, and so on).
- Check that filenames have expected extensions (discouraging files that execute when you “click on them”, or extension-less files like Windows special device names).
- Limit the number of extracted files, total size of extracted data, filename length (including symlink length), and size of individual files.
- Check for files that would be shadowed on case-insensitive filesystems.

Also note that:

- Tar files may contain multiple versions of the same file. Later ones are expected to overwrite any earlier ones. This feature is crucial to allow updating tape archives, but can be abused maliciously.
- *tarfile* does not protect against issues with “live” data, e.g. an attacker tinkering with the destination (or source) directory while extraction (or archiving) is in progress.

Supporting older Python versions

Extraction filters were added to Python 3.12, but may be backported to older versions as security updates. To check whether the feature is available, use e.g. `hasattr(tarfile, 'data_filter')` rather than checking the Python version.

The following examples show how to support Python versions with and without the feature. Note that setting `extraction_filter` will affect any subsequent operations.

- Fully trusted archive:

  ```
  my_tarfile.extraction_filter = (lambda member, path
  my_tarfile.extractall()
  ```

- Use the `'data'` filter if available, but revert to Python 3.11 behavior (`'fully_trusted'`) if this feature is not available:

  ```
  my_tarfile.extraction_filter = getattr(tarfile, 'da
                                         (lambda memb
  my_tarfile.extractall()
  ```

- Use the `'data'` filter; *fail* if it is not available:

  ```
  my_tarfile.extractall(filter=tarfile.data_filter)
  ```

  or:

  ```
  my_tarfile.extraction_filter = tarfile.data_filter
  my_tarfile.extractall()
  ```

- Use the `'data'` filter; *warn* if it is not available:

  ```
  if hasattr(tarfile, 'data_filter'):
      my_tarfile.extractall(filter='data')
  else:
      # remove this when no longer needed
      warn_the_user('Extracting may be unsafe; consid
      my_tarfile.extractall()
  ```

Stateful extraction filter example

While *tarfile*'s extraction methods take a simple *filter* callable, custom filters may be more complex objects with an internal state. It may be useful to write these as context managers, to be used like this:

```
with StatefulFilter() as filter_func:
    tar.extractall(path, filter=filter_func)
```

Such a filter can be written as, for example:

```
class StatefulFilter:
    def __init__(self):
        self.file_count = 0

    def __enter__(self):
        return self

    def __call__(self, member, path):
        self.file_count += 1
        return member

    def __exit__(self, *exc_info):
        print(f'{self.file_count} files extracted')
```

Interfaz de línea de comandos

Added in version 3.4.

The **`tarfile`** module provides a simple command-line interface to interact with tar archives.

Si quieres crear un nuevo archivo tar, especifica su nombre después de la opción `-c` y después lista el nombre de archivo(s) que deberían ser incluidos:

```
$ python -m tarfile -c monty.tar  spam.txt eggs.txt
```

Proporcionar un directorio también es aceptable:

```
$ python -m tarfile -c monty.tar life-of-brian_1979/
```

Si deseas extraer un archivo tar en el directorio actual, utiliza la opción **`-e`**:

```
$ python -m tarfile -e monty.tar
```

También puedes extraer un archivo tar en un directorio diferente pasando el nombre del directorio como parámetro:

```
$ python -m tarfile -e monty.tar  other-dir/
```

Para obtener una lista de archivos dentro de un archivo tar, utiliza la opción **`-l`**:

```
$ python -m tarfile -l monty.tar
```

Opciones de línea de comandos

-l <tarfile>
--list <tarfile>

Listar archivos en un tar.

-c <tarfile> <source1> ... <sourceN>
--create <tarfile> <source1> ... <sourceN>

Crear archivo tar desde archivos fuente.

-e <tarfile> [<output_dir>]
--extract <tarfile> [<output_dir>]

Extrae el archivo tar en el directorio actual si *output_dir* no es especificado.

-t <tarfile>
--test <tarfile>

Probar si el archivo tar es valido o no.

-v, --verbose

Output *verbose*.

--filter <filtername>

Specifies the *filter* for `--extract`. See Extraction filters for details. Only string names are accepted (that is, `fully_trusted`, `tar`, and `data`).

Ejemplos

Reading examples

Cómo extraer un archivo tar entero al directorio de trabajo actual:

```
import tarfile
tar = tarfile.open("sample.tar.gz")
tar.extractall(filter='data')
tar.close()
```

Cómo extraer un subconjunto de un archivo tar con **`TarFile.extractall()`** utilizando una función generadora en lugar de una lista:

```
import os
import tarfile

def py_files(members):
    for tarinfo in members:
        if os.path.splitext(tarinfo.name)[1] == ".py":
            yield tarinfo

tar = tarfile.open("sample.tar.gz")
tar.extractall(members=py_files(tar))
tar.close()
```

Cómo leer un archivo tar comprimido con gzip y desplegar un poco de información del miembro:

```
import tarfile
tar = tarfile.open("sample.tar.gz", "r:gz")
for tarinfo in tar:
    print(tarinfo.name, "is", tarinfo.size, "bytes in si
    if tarinfo.isreg():
        print("a regular file.")
    elif tarinfo.isdir():
        print("a directory.")
    else:
        print("something else.")
tar.close()
```

Writing examples

Cómo crear un archivo tar sin comprimir desde una lista de nombres de archivo:

```
import tarfile
tar = tarfile.open("sample.tar", "w")
for name in ["foo", "bar", "quux"]:
    tar.add(name)
tar.close()
```

El mismo ejemplo utilizando la declaración **`with`**:

```
import tarfile
with tarfile.open("sample.tar", "w") as tar:
    for name in ["foo", "bar", "quux"]:
        tar.add(name)
```

How to create and write an archive to stdout using **`sys.stdout.buffer`** in the *fileobj* parameter in **`TarFile.add()`**:

```
import sys
import tarfile
with tarfile.open("sample.tar.gz", "w|gz", fileobj=sys.s
    for name in ["foo", "bar", "quux"]:
        tar.add(name)
```

Cómo crear un archivo y reiniciar la información del usuario usando el parámetro *filter* en **`TarFile.add()`**:

```
import tarfile
```

```
def reset(tarinfo):
    tarinfo.uid = tarinfo.gid = 0
    tarinfo.uname = tarinfo.gname = "root"
    return tarinfo
tar = tarfile.open("sample.tar.gz", "w:gz")
tar.add("foo", filter=reset)
tar.close()
```

Formatos tar con soporte

There are three tar formats that can be created with the **`tarfile`** module:

- El formato (**`USTAR_FORMAT`**) POSIX.1-1988 ustar. Admite nombres de archivos de hasta una longitud de 256 caracteres y nombres de enlace de hasta 100 caracteres. El tamaño máximo de archivo es de 8 GiB. Este es un formato viejo y limitado pero con amplio soporte.

- The GNU tar format (**`GNU_FORMAT`**). It supports long filenames and linknames, files bigger than 8 GiB and sparse files. It is the de facto standard on GNU/Linux systems. **`tarfile`** fully supports the GNU tar extensions for long names, sparse file support is read-only.

- The POSIX.1-2001 pax format (**`PAX_FORMAT`**). It is the most flexible format with virtually no limits. It supports long filenames and linknames, large files and stores pathnames in a portable way. Modern tar implementations, including GNU tar, bsdtar/libarchive and star, fully support extended *pax* features; some old or unmaintained libraries may not, but should treat *pax* archives as if they were in the universally supported *ustar* format. It is the current default format for new archives.

  Extiende el formato existente *ustar* con cabeceras adicionales para información que no puede ser almacenada de otra manera. Hay dos variaciones de encabezados pax: Los encabezados extendidos solo afectan al encabezado del archivo posterior, las encabezados globales son validos para el archivo entero y afectan todos los siguientes archivos. Todos los datos en un encabezado pax son codificados en *UTF-8* por razones de portabilidad.

Existen más variantes del formato tar que puede ser leídas, pero no creadas:

- El antiguo formato V7. Este es el primer formato tar de *Unix Seventh Edition*, almacena solo archivos y directorios normales. Los nombres no deben tener más de 100 caracteres, no hay información de nombre de usuario/grupo disponible. Algunos archivos tienen sumas de comprobación de encabezado mal calculadas en el caso de campos con caracteres que no son ASCII.
- El formato extendido tar de SunOS. Este formato es una variante del formato POSIX.1-2001 pax, pero no es compatible.

Problemas unicode

El formato tar fue originalmente concebido para realizar respaldos en unidades de cinta con el objetivo principal de preservar la información del sistema de archivos. Hoy en día los archivos tar son comúnmente utilizados para distribución de archivos e intercambio de archivos sobre redes. Un problema del formato original (el cual es la base de todos los demás formatos) es que no hay concepto de soporte para diferentes codificaciones de caracteres. Por ejemplo, un archivo tar ordinario creado en un sistema *UTF-8* no puede ser leído correctamente en un sistema *Latin-1* si este contiene caracteres de tipo no ASCII. Los meta-datos textuales (como nombres de archivos, nombres de enlace, nombres de usuario/grupo) aparecerán dañados. Desafortunadamente, no hay manera de detectar de forma automática la codificación de un archivo. El formato *pax* fue diseñado para resolver este problema. Almacena los metadatos no ASCII usando una codificación de caracteres universal *UTF-8*.

The details of character conversion in `tarfile` are controlled by the *encoding* and *errors* keyword arguments of the `TarFile` class.

encoding define la codificación de caracteres a utilizar para los metadatos del archivo. El valor predeterminado es **`sys.getfilesystemencoding()`** o `'ascii'` como una alternativa. Dependiendo de si el archivo se lee o escribe, los metadatos deben decodificarse o codificarse. Si el *encoding* no se configura correctamente, esta conversión puede fallar.

El argumento *errors* define como van a ser tratados los caracteres que no pueden ser transformados. Los valores admitidos están listados en la sección Manejadores de errores. El esquema predeterminado es `'surrogateescape'` el cual Python también utiliza para sus llamadas al sistema de archivos. Consulta Nombres de archivos, argumentos de la línea de comandos y variables de entorno.

Para archivos **`PAX_FORMAT`** (el formato default), el *encoding* generalmente no es necesario porque todos los meta-datos son almacenados utilizando *UTF-8*, el *encoding* solo es utilizado en aquellos casos cuando las cabeceras binarias PAX son decodificadas o cuando se almacenan cadenas de texto con caracteres sustitutos.

Formatos de archivo

Los módulos descritos en este capítulo analizan varios formatos de archivo que no son lenguajes de marcado y no están relacionados con el correo electrónico.

- **`csv`** — CSV File Reading and Writing
    - Contenidos del módulo
    - Dialectos y parámetros de formato
    - Objetos *Reader*
    - Objetos *Writer*
    - Ejemplos
- **`configparser`** — Configuration file parser
    - Inicio Rápido
    - Tipos de Datos Soportados
    - Valores de contingencia
    - Estructura soportada para el archivo ini
    - Unnamed Sections
    - Interpolación de valores
    - Acceso por protocolo de mapeo
    - Personalizando el comportamiento del parser
    - Ejemplos de la API heredada
    - Objetos ConfigParser
    - Objetos RawConfigParser
    - Excepciones
- **`tomllib`** — Parse TOML files
    - Ejemplos
    - Tabla de conversión
- **`netrc`** — netrc file processing
    - Objetos netrc

- `plistlib` — Generate and parse Apple `.plist` files
    - Ejemplos

[1] (1,2)

`csv` — CSV File Reading and Writing

Código fuente: Lib/csv.py [https://github.com/python/cpython/tree/3.14/Lib/csv.py]

El tan llamado CSV (Valores Separados por Comas) es el formato más común de importación y exportación de hojas de cálculo y bases de datos. El formato CSV se utilizó durante muchos años antes de intentar describir el formato de manera estandarizada en **RFC 4180** [https://datatracker.ietf.org/doc/html/rfc4180.html]. La falta de un estándar bien definido significa que a veces existen pequeñas diferencias en la información producida y consumida por diferentes aplicaciones. Estas diferencias pueden ser molestas al momento de procesar archivos CSV desde múltiples fuentes. Aún así, aunque los delimitadores y separadores varíen, el formato general es lo suficientemente similar como para que sea posible un sólo módulo que puede manipular tal información eficientemente, escondiendo los detalles de lectura y escritura de datos del programador.

The `csv` module implements classes to read and write tabular data in CSV format. It allows programmers to say, «write this data in the format preferred by Excel,» or «read data from this file which was generated by Excel,» without knowing the precise details of the CSV format used by Excel. Programmers can also describe the CSV formats understood by other applications or define their own special-purpose CSV formats.

The `csv` module’s `reader` and `writer` objects read and write sequences. Programmers can also read and write data in dictionary form using the `DictReader` and `DictWriter` classes.

Ver también

PEP 305 [https://peps.python.org/pep-0305/] - API de archivo CSV
La Propuesta de Mejora de Python (PEP, por sus siglas en inglés) que propone esta adición a Python.

Contenidos del módulo

The **`csv`** module defines the following functions:

csv.reader(*csvfile*, /, *dialect='excel'*, **fmtparams)

Return a reader object that will process lines from the given *csvfile*. A csvfile must be an iterable of strings, each in the reader's defined csv format. A csvfile is most commonly a file-like object or list. If *csvfile* is a file object, it should be opened with `newline=''`. [1] An optional *dialect* parameter can be given which is used to define a set of parameters specific to a particular CSV dialect. It may be an instance of a subclass of the **`Dialect`** class or one of the strings returned by the **`list_dialects()`** function. The other optional *fmtparams* keyword arguments can be given to override individual formatting parameters in the current dialect. For full details about the dialect and formatting parameters, see section Dialectos y parámetros de formato.

Each row read from the csv file is returned as a list of strings. No automatic data type conversion is performed unless the **`QUOTE_NONNUMERIC`** format option is specified (in which case unquoted fields are transformed into floats).

Un pequeño ejemplo de uso:

```
>>> import csv
>>> with open('eggs.csv', newline='') as csvfile:
...     spamreader = csv.reader(csvfile, delimiter=
...     for row in spamreader:
...         print(', '.join(row))
Spam, Spam, Spam, Spam, Spam, Baked Beans
Spam, Lovely Spam, Wonderful Spam
```

csv.writer(*csvfile*, /, *dialect='excel'*, ***fmtparams*)

Return a writer object responsible for converting the user's data into delimited strings on the given file-like object. *csvfile* can be any object with a `write()` method. If *csvfile* is a file object, it should be opened with `newline=''` [1]. An optional *dialect* parameter can be given which is used to define a set of parameters specific to a particular CSV dialect. It may be an instance of a subclass of the `Dialect` class or one of the strings returned by the `list_dialects()` function. The other optional *fmtparams* keyword arguments can be given to override individual formatting parameters in the current dialect. For full details about dialects and formatting parameters, see the Dialectos y parámetros de formato section. To make it as easy as possible to interface with modules which implement the DB API, the value `None` is written as the empty string. While this isn't a reversible transformation, it makes it easier to dump SQL NULL data values to CSV files without preprocessing the data returned from a `cursor.fetch*` call. All other non-string data are stringified with `str()` before being written.

Un pequeño ejemplo de uso:

```
import csv
with open('eggs.csv', 'w', newline='') as csvfile:
    spamwriter = csv.writer(csvfile, delimiter=' ',
                            quotechar='|', quoting=
    spamwriter.writerow(['Spam'] * 5 + ['Baked Bean
    spamwriter.writerow(['Spam', 'Lovely Spam', 'Wo
```

csv.register_dialect(*name*, /, *dialect='excel'*, ***fmtparams*)

Asocie *dialect* con *name*. *name* debe ser una cadena. El dialecto se puede especificar pasando una subclase de `Dialect`, o mediante argumentos de palabra clave *fmtparams*, o ambos, con argumentos de palabra clave que anulan los parámetros del dialecto. Para obtener detalles completos sobre dialectos y parámetros de formato, consulte la sección Dialectos y parámetros de formato.

csv.unregister_dialect(*name*)

Borra el dialecto asociado a *name* del registro de dialectos. Un `Error` es lanzado si *name* no está registrado como el nombre de un dialecto.

csv.get_dialect(*name*)

Retorna el dialecto asociado a *name*. Un `Error` es lanzado si *name* no está registrado como el nombre de un dialecto. Esta función retorna un objeto `Dialect` inmutable.

csv.list_dialects()

Retorna los nombres de todos los dialectos registrados.

csv.field_size_limit()
csv.field_size_limit(*new_limit*)

Retorna el tamaño máximo de campo permitido actualmente por el intérprete. Si *new_limit* es dado, este se convierte en el nuevo límite.

The **`csv`** module defines the following classes:

class csv.DictReader(*f*, *fieldnames=None*, *restkey=None*, *restval=None*, *dialect='excel'*, **args*, ***kwds*)

Crea un objeto que opera como un *reader* común, pero mapea la información en cada fila a un `dict` cuyas claves son provistas en el parámetro opcional *fieldnames*.

The *fieldnames* parameter is a sequence. If *fieldnames* is omitted, the values in the first row of file *f* will be used as the fieldnames and will be omitted from the results. If *fieldnames* is provided, they will be used and the first row will be included in the results. Regardless of how the fieldnames are determined, the dictionary preserves their original ordering.

Si una fila tiene más campos que nombres de campo, los datos restantes son colocados en una lista y guardados con el nombre de campo especificado por *restkey* (que por defecto es

`None`). Si una fila que no esta en blanco tiene menos campos que nombres de campo, los valores faltantes son rellenados con el valor de *restval* (que por defecto es `None`).

Todos los demás argumentos de palabra clave u opcionales son pasados a la instancia subyacente de **`reader`**.

Si el argumento pasado a *fieldnames* es un iterador, sera forzado a una **`list`**.

Distinto en la versión 3.6: Las filas retornadas son ahora de tipo **`OrderedDict`**.

Distinto en la versión 3.8: Las filas retornadas son ahora de tipo **`dict`**.

Un pequeño ejemplo de uso:

```
>>> import csv
>>> with open('names.csv', newline='') as csvfile:
...     reader = csv.DictReader(csvfile)
...     for row in reader:
...         print(row['first_name'], row['last_name
...
Eric Idle
John Cleese

>>> print(row)
{'first_name': 'John', 'last_name': 'Cleese'}
```

class csv.DictWriter(*f*, *fieldnames*, *restval=''*, *extrasaction='raise'*, *dialect='excel'*, **args*, ***kwds*)

Create an object which operates like a regular writer but maps dictionaries onto output rows. The *fieldnames* parameter is a **`sequence`** of keys that identify the order in which values in the dictionary passed to the **`writerow()`** method are written to file *f*. The optional *restval* parameter specifies the value to be written if the dictionary is missing a key in *fieldnames*. If the dictionary passed to the **`writerow()`** method contains a key not found in *fieldnames*, the optional *extrasaction* parameter indicates what action to take. If it is set to `'raise'`, the default value, a **`ValueError`** is raised. If it is set to `'ignore'`, extra values in the dictionary are ignored. Any other optional or keyword arguments are passed to the underlying **`writer`** instance.

Nótese que a diferencia de la clase **`DictReader`**, el parámetro *fieldnames* de la clase **`DictWriter`** no es opcional.

Si el argumento pasado a *fieldnames* es un iterador, sera forzado a una **`list`**.

Un pequeño ejemplo de uso:

```
import csv

with open('names.csv', 'w', newline='') as csvfile:
    fieldnames = ['first_name', 'last_name']
    writer = csv.DictWriter(csvfile, fieldnames=fie

    writer.writeheader()
    writer.writerow({'first_name': 'Baked', 'last_n
    writer.writerow({'first_name': 'Lovely', 'last_
    writer.writerow({'first_name': 'Wonderful', 'la
```

class csv.Dialect

La clase **`Dialect`** es una clase contenedora cuyos atributos contienen información sobre cómo manejar las comillas dobles, los espacios en blanco, los delimitadores, etc. Debido a la falta de una especificación estricta de CSV, diferentes aplicaciones producen datos CSV sutilmente diferentes. Las instancias de **`Dialect`** definen cómo se comportan las instancias de **`reader`** y **`writer`**.

Todos los nombres disponibles de **`Dialect`** son retornados por **`list_dialects()`**, y pueden ser registrados con clases específicas de **`reader`** y **`writer`** a través de sus funciones que inician (`__init__`) así:

```
import csv

with open('students.csv', 'w', newline='') as csvfi
    writer = csv.writer(csvfile, dialect='unix')
```

class csv.excel

La clase **`excel`** define las propiedades usuales de un archivo CSV generado por Excel. Esta registrada con el nombre de dialecto `'excel'`.

class csv.excel_tab

La clase **`excel_tab`** define las propiedades usuales de un archivo delimitado por tabulaciones generado por Excel. Esta registrada con el nombre de dialecto `'excel-tab'`.

class csv.unix_dialect

La clase **`unix_dialect`** define las propiedades usuales de un archivo CSV generado en sistemas UNIX, es decir, usando `'\n'` como terminador de línea y citando todos los campos. Esta registrada con el nombre de dialecto `'unix'`.

Added in version 3.2.

class csv.Sniffer

La clase **`Sniffer`** es usada para deducir el formato de un archivo CSV.

La clase **`Sniffer`** provee 2 métodos:

sniff(*sample*, *delimiters=None*)

Analiza la muestra dada y retorna una subclase **`Dialect`** reflejando los parámetros encontrados. Si el parámetro opcional *delimiters* es dado, este será interpretado como una cadena que contiene posibles caracteres delimitadores válidos.

has_header(*sample*)

Analiza el texto de muestra (se supone que está en formato CSV) y retorna `True` si la primera fila parece ser una serie de encabezados de columna. Al inspeccionar cada columna, se considerará uno de dos criterios clave para estimar si la muestra contiene un encabezado:

- las filas segunda n-th contienen valores numéricos
- las segundas filas n-th contienen cadenas en las que la longitud de al menos un valor difiere de la de la supuesta cabecera de esa columna.

Twenty-one rows after the header are sampled; if more than half of the columns + rows meet the criteria, `True` is returned.

Nota

Este método es una heurística aproximada y puede producir tanto falsos positivos como negativos.

Un ejemplo para el uso de `Sniffer`:

```
with open('example.csv', newline='') as csvfile:
    dialect = csv.Sniffer().sniff(csvfile.read(1024))
    csvfile.seek(0)
    reader = csv.reader(csvfile, dialect)
    # ... process CSV file contents here ...
```

The **`csv`** module defines the following constants:

csv.QUOTE_ALL

Ordena a los objetos `writer` citar todos los campos.

csv.QUOTE_MINIMAL

Instructs `writer` objects to only quote those fields which contain special characters such as *delimiter*, *quotechar*, `'\r'`, `'\n'` or any of the characters in *lineterminator*.

csv.QUOTE_NONNUMERIC

Ordena a los objetos `writer` citar todos los campos no numéricos.

Instructs `reader` objects to convert all non-quoted fields to type `float`.

Nota

Some numeric types, such as `bool`, `Fraction`, or `IntEnum`, have a string representation that cannot be converted to `float`. They cannot be read in the **`QUOTE_NONNUMERIC`** and `QUOTE_STRINGS` modes.

csv.QUOTE_NONE

Instructs `writer` objects to never quote fields. When the current *delimiter*, *quotechar*, *escapechar*, `'\r'`, `'\n'` or any of the characters in *lineterminator* occurs in output data it is preceded by the current *escapechar* character. If *escapechar* is not set, the writer will raise `Error` if any characters that require escaping are encountered. Set *quotechar* to `None` to prevent its escaping.

Ordena a los objetos `reader` no ejecutar un procesamiento especial de caracteres comillas.

csv.QUOTE_NOTNULL

Ordena a los objetos `writer` poner entre comillas todos los campos que no sean `None`. Esto es similar a `QUOTE_ALL`, con la excepción que si un campo es `None` se escribe una cadena de caracteres vacía (sin comillas).

Instructs `reader` objects to interpret an empty (unquoted) field as `None` and to otherwise behave as `QUOTE_ALL`.

Added in version 3.12.

csv.QUOTE_STRINGS

Ordena a los objetos **`writer`** siempre poner comillas alrededor de campos que sean cadenas de caracteres. Esto es similar a **`QUOTE_NONNUMERIC`**, con la excepción que si el valor de un campo es `None` se escribe una cadena de caracteres vacía (sin comillas).

Ordena a los objetos **`reader`** interpretar una cadena de caracteres vacía (sin comillas) como `None` y en caso contrario comportarse como **`QUOTE_NONNUMERIC`**.

Added in version 3.12.

The **`csv`** module defines the following exception:

exception csv.Error

Lanzada por cualquiera de las funciones cuando se detecta un error.

Dialectos y parámetros de formato

To make it easier to specify the format of input and output records, specific formatting parameters are grouped together into dialects. A dialect is a subclass of the **`Dialect`** class containing various attributes describing the format of the CSV file. When creating **`reader`** or **`writer`** objects, the programmer can specify a string or a subclass of the **`Dialect`** class as the dialect parameter. In addition to, or instead of, the *dialect* parameter, the programmer can also specify individual formatting parameters, which have the same names as the attributes defined below for the **`Dialect`** class.

Los dialectos soportan los siguientes atributos:

Dialect.delimiter

Una cadena de un solo carácter usada para separar campos. Por defecto es `','`.

Dialect.doublequote

Controla como las instancias de *quotechar* que aparecen dentro de un campo deben estar citadas. Cuando es **`True`**, el carácter es duplicado. Cuando es `False`, el *escapechar* es usado como un prefijo el *quotechar*. Por defecto es **`True`**.

En la salida, si el *doublequote* es `False` y el *escapechar* no está configurado, un `Error` es lanzado si se encuentra un *quotechar* en algún campo.

Dialect.escapechar

A one-character string used by the writer to escape characters that require escaping:

- the *delimiter*, the *quotechar*, `'\r'`, `'\n'` and any of the characters in *lineterminator* are escaped if *quoting* is set to `QUOTE_NONE`;
- the *quotechar* is escaped if *doublequote* is `False`;
- the *escapechar* itself.

On reading, the *escapechar* removes any special meaning from the following character. It defaults to `None`, which disables escaping.

Distinto en la versión 3.11: No se permite un *quotechar* vacío.

Dialect.lineterminator

La cadena de caracteres usada para terminar las líneas producidas por `writer`. Por defecto es `'\r\n'`.

Nota

El `reader` esta codificado para reconocer tanto `'\r'` o `'\n'` como final de línea, e ignora *lineterminator*. Este comportamiento puede cambiar en el futuro.

Dialect.quotechar

A one-character string used to quote fields containing special characters, such as the *delimiter* or the *quotechar*, or which contain new-line characters (`'\r'`, `'\n'` or any of the characters in *lineterminator*). It defaults to `'"'`. Can be set to `None` to prevent escaping `'"'` if *quoting* is set to **`QUOTE_NONE`**.

Distinto en la versión 3.11: No se permite un *quotechar* vacío.

Dialect.quoting

Controls when quotes should be generated by the writer and recognised by the reader. It can take on any of the QUOTE_* constants and defaults to **`QUOTE_MINIMAL`** if *quotechar* is not `None`, and **`QUOTE_NONE`** otherwise.

Dialect.skipinitialspace

When **`True`**, spaces immediately following the *delimiter* are ignored. The default is **`False`**. When combining `delimiter=' '` with `skipinitialspace=True`, unquoted empty fields are not allowed.

Dialect.strict

Cuando es `True`, lanza una excepción **`Error`** sobre una mala entrada CSV. Por defecto es `False`.

Objetos *Reader*

Los objetos *reader* (instancias de **`DictReader`** y objetos retornados por la función **`reader()`**) contienen los siguientes métodos públicos:

csvreader.__next__()

Retorna la siguiente fila del objeto iterable del lector como una lista (si el objeto retornó desde **`reader()`**) o un dict (si es una instancia de **`DictReader`**), analizado de acuerdo con el **`Dialect`** actual. Por lo general, debería llamar a esto como `next(reader)`.

Los objetos *reader* contienen los siguientes atributos públicos:

csvreader.dialect

Una descripción de sólo lectura del dialecto en uso por el intérprete.

csvreader.line_num

El número de líneas leídas del iterador fuente. Esto no es lo mismo que el número de registros retornado, ya que los registros pueden abarcar múltiples líneas.

Los objetos *DictReader* tienen los siguientes atributos públicos:

DictReader.fieldnames

Si no son pasados como parámetros cuando se crea el objeto, este atributo es inicializado en el primer acceso o cuando es leído el primer registro del archivo.

Objetos *Writer*

`writer` objects (`DictWriter` instances and objects returned by the `writer()` function) have the following public methods. A *row* must be an iterable of strings or numbers for `writer` objects and a dictionary mapping fieldnames to strings or numbers (by passing them through `str()` first) for `DictWriter` objects. Note that complex numbers are written out surrounded by parens. This may cause some problems for other programs which read CSV files (assuming they support complex numbers at all).

csvwriter.writerow(*row*, */*)

Escriba el parámetro *row* en el objeto de archivo del escritor, formateado de acuerdo con el `Dialect` actual. Retorna el valor de retorno de la llamada al método *write* del objeto de archivo subyacente.

Distinto en la versión 3.5: Agregado soporte para iterables.

csvwriter.writerows(*rows*, */*)

Escribe todos los elementos en *rows* (un iterable de objetos *row* como se describe anteriormente) al objeto de archivo del *writer*, formateados según el dialecto actual.

Los objetos *writer* contienen los siguientes atributos públicos:

csvwriter.dialect

Una descripción de solo lectura del dialecto en uso por el *writer*.

Los objetos *DictWriter* contienen los siguientes métodos públicos:

DictWriter.writeheader()

Escribe una fila con los nombres de los campos (como se especifica en el constructor) al objeto de archivo del *writer*, formateada según el dialecto actual. Retorna el valor de retorno de la llamada a `csvwriter.writerow()` usada internamente.

Added in version 3.2.

Distinto en la versión 3.8: `writeheader()` ahora también retorna el valor retornado por el método `csvwriter.writerow()` que usa internamente.

Ejemplos

El ejemplo más simple de lectura de un archivo CSV:

```
import csv
with open('some.csv', newline='') as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

Lectura de un archivo con un formato alternativo:

```
import csv
with open('passwd', newline='') as f:
    reader = csv.reader(f, delimiter=':', quoting=csv.QU
    for row in reader:
        print(row)
```

El correspondiente ejemplo de escritura más simple es:

```
import csv
with open('some.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerows(someiterable)
```

Cuando se usa `open()` para abrir un archivo CSV en modo lectura, el archivo será decodificado por defecto en unicode usando la codificación por defecto del sistema (ver `locale.getpreferredencoding()`). Para decodificar un archivo usando una codificación diferente, usa el argumento `encoding` de open:

```
import csv
with open('some.csv', newline='', encoding='utf-8') as f
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

Lo mismo aplica a escribir en algo diferente a la codificación por defecto del sistema: especifique el argumento de codificación cuando abra el archivo de salida.

Registrando un nuevo dialecto:

```
import csv
csv.register_dialect('unixpwd', delimiter=':', quoting=c
with open('passwd', newline='') as f:
    reader = csv.reader(f, 'unixpwd')
```

Un uso ligeramente más avanzado del *reader* — captura y reporte de errores:

```
import csv, sys
filename = 'some.csv'
with open(filename, newline='') as f:
    reader = csv.reader(f)
    try:
        for row in reader:
            print(row)
    except csv.Error as e:
        sys.exit(f'file {filename}, line {reader.line_nu
```

Y a pesar de que el módulo no soporta el análisis de cadenas de caracteres directamente, puede ser realizado fácilmente:

```
import csv
for row in csv.reader(['one,two,three']):
    print(row)
```

Notas al pie

If `newline=''` is not specified, newlines embedded inside quoted fields will not be interpreted correctly, and on platforms that use `\r\n` line endings on write an extra `\r` will be added. It should always be safe to specify `newline=''`, since the csv module does its own (universal) newline handling.

[1] (1,2,3,4,5,6,7,8,9,10,11)

`configparser` — Configuration file parser

Código fuente: Lib/configparser.py [https://github.com/python/cpython/tree/3.14/Lib/configparser.py]

Este módulo provee la clase `ConfigParser`, la cual implementa un lenguaje básico de configuración que proporciona una estructura similar a la encontrada en los archivos INI de Microsoft Windows. Puedes utilizarla para escribir programas Python que los usuarios finales puedan personalizar con facilidad.

Nota

Esta biblioteca *no* interpreta o escribe los prefijos valor-tipo usados en la versión extendida de la sintaxis INI, utilizada en el registro de Windows.

Ver también

Módulo `tomllib`
TOML es un formato bien-especificado para archivos de configuración de aplicaciones. Está específicamente diseñado para ser una versión mejorada de INI.

Módulo `shlex`
Soporta la creación de un mini-lenguaje parecido a shell de Unix, que puede utilizarse para archivos de configuración de aplicaciones.

Módulo `json`
El módulo `json` implementa un subconjunto de la sintaxis de JavaScript, que también puede utilizarse para configuración, pero no soporta comentarios.

Inicio Rápido

Tomemos un archivo de configuración muy básico, el cual luce así:

```
[DEFAULT]
ServerAliveInterval = 45
Compression = yes
CompressionLevel = 9
ForwardX11 = yes

[forge.example]
User = hg

[topsecret.server.example]
Port = 50022
ForwardX11 = no
```

The structure of INI files is described in the following section. Essentially, the file consists of sections, each of which contains keys with values. **`configparser`** classes can read and write such files. Let's start by creating the above configuration file programmatically.

```
>>> import configparser
>>> config = configparser.ConfigParser()
>>> config['DEFAULT'] = {'ServerAliveInterval': '45',
...                      'Compression': 'yes',
...                      'CompressionLevel': '9'}
>>> config['forge.example'] = {}
>>> config['forge.example']['User'] = 'hg'
>>> config['topsecret.server.example'] = {}
>>> topsecret = config['topsecret.server.example']
>>> topsecret['Port'] = '50022'     # mutates the parser
>>> topsecret['ForwardX11'] = 'no'  # same here
>>> config['DEFAULT']['ForwardX11'] = 'yes'
>>> with open('example.ini', 'w') as configfile:
...   config.write(configfile)
...
```

Como puedes ver, podemos tratar al *config parser* como a un diccionario. Existen diferencias, descritas posteriormente, pero su comportamiento es muy parecido al que esperarías de un diccionario.

Ahora que hemos creado y guardado el archivo de configuración, vamos a releerlo y analizar los datos que contiene.

```
>>> config = configparser.ConfigParser()
>>> config.sections()
[]
>>> config.read('example.ini')
['example.ini']
>>> config.sections()
['forge.example', 'topsecret.server.example']
>>> 'forge.example' in config
True
>>> 'python.org' in config
False
>>> config['forge.example']['User']
'hg'
>>> config['DEFAULT']['Compression']
'yes'
>>> topsecret = config['topsecret.server.example']
>>> topsecret['ForwardX11']
'no'
>>> topsecret['Port']
'50022'
>>> for key in config['forge.example']:
...     print(key)
user
compressionlevel
serveraliveinterval
compression
forwardx11
>>> config['forge.example']['ForwardX11']
'yes'
```

Como podemos apreciar, la API es muy clara. La única “porción de magia” está en la sección `DEFAULT`, la cual proporciona los valores por defecto para todas las demás secciones [1]. Observe también que las claves de las secciones son insensibles a mayúsculas y minúsculas, pero se almacenan en minúscula [1].

It is possible to read several configurations into a single **`ConfigParser`**, where the most recently added configuration has the highest priority. Any conflicting keys are taken from the more recent configuration while the previously existing keys are retained. The example below reads in an `override.ini` file, which will override any conflicting keys from the `example.ini` file.

```
[DEFAULT]
ServerAliveInterval = -1
```

```
>>> config_override = configparser.ConfigParser()
>>> config_override['DEFAULT'] = {'ServerAliveInterval':
>>> with open('override.ini', 'w') as configfile:
...     config_override.write(configfile)
...
>>> config_override = configparser.ConfigParser()
>>> config_override.read(['example.ini', 'override.ini']
['example.ini', 'override.ini']
>>> print(config_override.get('DEFAULT', 'ServerAliveInt
-1
```

Este comportamiento es equivalente a una llamada a **`ConfigParser.read()`** pasando varios ficheros en el parámetro *filenames*.

Tipos de Datos Soportados

Los *config parsers* no especulan respecto a los tipos de datos de los valores en los archivos de configuración, siempre los almacenan internamente como cadenas de caracteres. Esto significa que si necesitas otros tipos de datos, deberás hacer la conversión por ti mismo:

```
>>> int(topsecret['Port'])
50022
```

```
>>> float(topsecret['CompressionLevel'])
9.0
```

Dado que esta tarea es muy común, los *config parsers* proporcionan una amplia variedad de útiles métodos de lectura (*getter*) que manejan enteros, números de punto flotante y booleanos. Este último es el más interesante, porque puede no ser correcto pasar simplemente el valor a `bool()`, ya que `bool('False')` resultará en `True`. Por esta razón los *config parsers* proporcionan también el método **`getboolean()`**. Este método es insensible a mayúsculas y minúsculas, y reconoce como valores booleanos a los valores `'yes'`/`'no'`, `'on'`/`'off'`, `'true'`/`'false'` and `'1'`/`'0'` [1]. Por ejemplo:

```
>>> topsecret.getboolean('ForwardX11')
False
>>> config['forge.example'].getboolean('ForwardX11')
True
>>> config.getboolean('forge.example', 'Compression')
True
```

Además de **`getboolean()`**, los *config parsers* también proporcionan los métodos equivalentes **`getint()`** y **`getfloat()`**. Puedes registrar tus propios conversores, y personalizar los que se proporcionan. [1]

Valores de contingencia

As with a dictionary, you can use a section's **`get()`** method to provide fallback values:

```
>>> topsecret.get('Port')
'50022'
>>> topsecret.get('CompressionLevel')
'9'
>>> topsecret.get('Cipher')
>>> topsecret.get('Cipher', '3des-cbc')
'3des-cbc'
```

Por favor, fíjate que los valores por defecto tienen prioridad sobre los valores de contingencia (*fallback*). Así, en nuestro ejemplo, la clave `'CompressionLevel'` sólo fue especificada en la sección `'DEFAULT'`. Si tratamos de obtener su valor de la sección `'topsecret.server.example'`, obtendremos siempre el valor por defecto, incluso si especificamos un valor de contingencia:

```
>>> topsecret.get('CompressionLevel', '3')
'9'
```

One more thing to be aware of is that the parser-level **`get()`** method provides a custom, more complex interface, maintained for backwards compatibility. When using this method, a fallback value can be provided via the `fallback` keyword-only argument:

```
>>> config.get('forge.example', 'monster',
...            fallback='No such things as monsters')
'No such things as monsters'
```

El mismo argumento `fallback` puede utilizarse con los métodos **`getint()`**, **`getfloat()`** y **`getboolean()`**, por ejemplo:

```
>>> 'BatchMode' in topsecret
False
>>> topsecret.getboolean('BatchMode', fallback=True)
True
>>> config['DEFAULT']['BatchMode'] = 'no'
>>> topsecret.getboolean('BatchMode', fallback=True)
False
```

Estructura soportada para el archivo ini

Un archivo de configuración consta de secciones, cada una iniciada por una cabecera `[section]`, seguida por registros clave-valor separados por una cadena de caracteres específica (= ó `:` por defecto [1]). De forma predeterminada, los nombres de sección son sensibles a mayúsculas y minúsculas pero las claves no [1]. Los espacios al inicio y final de las claves y valores son eliminados. Los valores pueden ser omitidos si el *parser* está configurado para permitirlo [1], en cuyo caso el delimitador del dato clave-valor puede ser omitido también. Los valores pueden ocupar varias líneas, siempre y cuando tengan una indentación mayor que la primera línea del valor. Dependiendo del modo del *parser*, las líneas en blanco pueden tratarse como parte de un valor multilínea o ser ignoradas.

By default, a valid section name can be any string that does not contain "\n". To change this, see **`ConfigParser.SECTCRE`**.

The first section name may be omitted if the parser is configured to allow an unnamed top level section with `allow_unnamed_section=True`. In this case, the keys/values may be retrieved by **`UNNAMED_SECTION`** as in `config[UNNAMED_SECTION]`.

Los archivos de configuración pueden incluir comentarios, con caracteres específicos como prefijos (`#` y `;` por defecto [1]). Los comentarios pueden ocupar su propia línea, posiblemente indentada. [1]

Por ejemplo:

```
[Simple Values]
key=value
spaces in keys=allowed
spaces in values=allowed as well
spaces around the delimiter = obviously
you can also use : to delimit keys from values

[All Values Are Strings]
values like this: 1000000
or this: 3.14159265359
are they treated as numbers? : no
integers, floats and booleans are held as: strings
can use the API to get converted values directly: true

[Multiline Values]
chorus: I'm a lumberjack, and I'm okay
    I sleep all night and I work all day

[No Values]
```

```
key_without_value
empty string value here =

[You can use comments]
# like this
; or this

# By default only in an empty line.
# Inline comments can be harmful because they prevent us
# from using the delimiting characters as parts of value
# That being said, this can be customized.

    [Sections Can Be Indented]
        can_values_be_as_well = True
        does_that_mean_anything_special = False
        purpose = formatting for readability
        multiline_values = are
            handled just fine as
            long as they are indented
            deeper than the first line
            of a value
        # Did I mention we can indent comments, too?
```

Unnamed Sections

The name of the first section (or unique) may be omitted and values retrieved by the `UNNAMED_SECTION` attribute.

```
>>> config = """
... option = value
...
... [  Section 2  ]
... another = val
... """
>>> unnamed = configparser.ConfigParser(allow_unnamed_se
>>> unnamed.read_string(config)
>>> unnamed.get(configparser.UNNAMED_SECTION, 'option')
'value'
```

Interpolación de valores

En el nivel superior de su funcionalidad central, **`ConfigParser`** soporta la interpolación. Esto significa que los valores pueden ser preprocesados, antes de ser retornados por los llamados a `get()`.

class configparser.BasicInterpolation

Es la implementación por defecto que utiliza **`ConfigParser`**. Permite valores que contengan cadenas de formato, que hacen referencia a otros valores en la misma sección o a valores presentes en la sección especial *default* [1]. Valores por defecto adicionales pueden ser proporcionados en la inicialización.

Por ejemplo:

```
[Paths]
home_dir: /Users
my_dir: %(home_dir)s/lumberjack
my_pictures: %(my_dir)s/Pictures

[Escape]
# use a %% to escape the % sign (% is the only char
gain: 80%%
```

En el ejemplo anterior, **`ConfigParser`** con la *interpolación* establecida a `BasicInterpolation()` puede resolver `%(home_dir)s` al valor `home_dir` (`/Users` en este caso). En efecto, `%(my_dir)s` podría resolverse como `/Users/lumberjack`. Todas las interpolaciones son realizadas bajo demanda, de modo que las claves utilizadas en la cadena de referencias no requieren un orden específico en el archivo de configuración.

Con `interpolation` establecida al valor `None`, el *parser* retornará simplemente `%(my_dir)s/Pictures` como el valor de `my_pictures` y `%(home_dir)s/lumberjack` como el valor de `my_dir`.

class configparser.ExtendedInterpolation

Es un gestor alternativo para la interpolación, que implementa una sintaxis más avanzada; es utilizado, por ejemplo, en `zc.buildout`. La interpolación es extendida al utilizar `${section:option}`, a fin de especificar un valor que proviene de una sección externa. La interpolación puede cubrir múltiples niveles. Por conveniencia, si la parte `section:` es omitida, la interpolación utilizará la sección actual (y posiblemente los valores por defecto establecidos en la sección especial).

Por ejemplo, la configuración indicada anteriormente, con interpolación básica, luciría de la siguiente manera utilizando interpolación extendida:

```
[Paths]
home_dir: /Users
my_dir: ${home_dir}/lumberjack
my_pictures: ${my_dir}/Pictures

[Escape]
# use a $$ to escape the $ sign ($ is the only char
cost: $$80
```

También pueden buscarse valores en otras secciones:

```
[Common]
home_dir: /Users
library_dir: /Library
system_dir: /System
macports_dir: /opt/local

[Frameworks]
Python: 3.2
path: ${Common:system_dir}/Library/Frameworks/

[Arthur]
nickname: Two Sheds
last_name: Jackson
my_dir: ${Common:home_dir}/twosheds
my_pictures: ${my_dir}/Pictures
```

```
python_dir: ${Frameworks:path}/Python/Versions/${Fr
```

Acceso por protocolo de mapeo

Added in version 3.2.

Mapping protocol access is a generic name for functionality that enables using custom objects as if they were dictionaries. In case of **`configparser`**, the mapping interface implementation is using the `parser['section']['option']` notation.

En particular, `parser['section']` retorna un proxy para los datos de la sección en el *parser*. Esto significa que los valores no son copiados, sino que son tomados del *parser* original sobre la marcha.

`configparser` objects behave as close to actual dictionaries as possible. The mapping interface is complete and adheres to the **`MutableMapping`** ABC. However, there are a few differences that should be taken into account:

- Por defecto, todas las claves de las secciones se pueden acceder de una forma insensible a mayúsculas y minúsculas [1]. Ej. `for option in parser["section"]` entrega solamente nombres de claves de opción que han pasado por `optionxform`. Esto significa, claves en minúscula por defecto. A la vez, para una sección que contiene la clave `'a'`, ambas expresiones retornan `True`:

  ```
  "a" in parser["section"]
  "A" in parser["section"]
  ```

- Todas las secciones también incluyen valores `DEFAULTSECT`, lo que significa que `.clear()` en una sección puede no significar que esta quede vacía de forma visible. Ello debido a que los valores por defecto no pueden ser borrados de la sección (ya que técnicamente no están allí). Si fueron redefinidas en la sección, su borrado ocasiona que los valores por defecto sean visibles de nuevo. Cualquier intento de borrar un valor por defecto ocasiona una excepción **`KeyError`**.

- `DEFAULTSECT` no puede ser eliminado del *parser*:
    - el intento de borrarlo lanza una excepción **`ValueError`**,
    - `parser.clear()` lo deja intacto,
    - `parser.popitem()` nunca lo retorna.

- `parser.get(section, option, **kwargs)` - el segundo argumento **no** es un valor de contingencia. Observe, sin embargo, que los métodos `get()` a nivel de sección son compatibles tanto con el protocolo de mapeo como con la API clásica de *configparser*.

- `parser.items()` es compatible con el protocolo de mapeo (retorna una lista de pares *section_name*, *section_proxy* que estén incluidos en DEFAULTSECT). Sin embargo, este método también puede ser invocado con los argumentos: `parser.items(section, raw, vars)`. Esta última llamada retorna una lista de pares *option*, *value* para una `section` específica, con todas las interpolaciones expandidas (a menos que se especifique `raw=True`).

El protocolo de mapeo se implementa en el nivel superior de la actual API heredada, de modo que esas subclases que sobre-escriben la interfaz original aún deberían hacer funcionar el mapeo como se esperaría.

Personalizando el comportamiento del parser

There are nearly as many INI format variants as there are applications using it. **`configparser`** goes a long way to provide support for the largest sensible set of INI styles available. The default functionality is mainly dictated by historical background and it's very likely that you will want to customize some of the features.

The most common way to change the way a specific config parser works is to use the **`__init__()`** options:

- *defaults*, valor por defecto: `None`

  Esta opción acepta un diccionario de pares clave-valor, que debe ser colocado inicialmente en la sección `DEFAULT`. De este modo se obtiene una manera elegante de apoyar los archivos de configuración concisos, que no especifican valores que sean los mismos documentados por defecto.

  Hint: if you want to specify default values for a specific section, use **`read_dict()`** before you read the actual file.

- *dict_type*, valor por defecto: **`dict`**

  Esta opción tiene un gran impacto en cómo funcionará el protocolo de mapeo, y cómo lucirán los archivos de configuración a escribir. Con el diccionario estándar, cada sección se almacena en el orden en que se añadieron al *parser*. Lo mismo ocurre con las opciones dentro de las secciones.

  Un tipo alternativo de diccionario puede ser utilizado, por ejemplo, para ordenar las secciones y opciones en un modo a posteriori (*write-back*).

  Por favor, tome en cuenta que: existen formas de agregar un par clave-valor en una sola operación. Cuando se utiliza un diccionario común en tales operaciones, el orden de las claves será el empleado en la creación. Por ejemplo:

  ```
  >>> parser = configparser.ConfigParser()
  >>> parser.read_dict({'section1': {'key1': 'value1'
  ...                                'key2': 'value2'
  ...                                'key3': 'value3'
  ...                   'section2': {'keyA': 'valueA'
  ...                                'keyB': 'valueB'
  ...                                'keyC': 'valueC'
  ...                   'section3': {'foo': 'x',
  ...                                'bar': 'y',
  ...                                'baz': 'z'}
  ... })
  >>> parser.sections()
  ['section1', 'section2', 'section3']
  ```

```
>>> [option for option in parser['section3']]
['foo', 'bar', 'baz']
```

- *allow_no_value*, valor por defecto: `False`

  Some configuration files are known to include settings without values, but which otherwise conform to the syntax supported by **`configparser`**. The *allow_no_value* parameter to the constructor can be used to indicate that such values should be accepted:

  ```
  >>> import configparser

  >>> sample_config = """
  ... [mysqld]
  ...   user = mysql
  ...   pid-file = /var/run/mysqld/mysqld.pid
  ...   skip-external-locking
  ...   old_passwords = 1
  ...   skip-bdb
  ...   # we don't need ACID today
  ...   skip-innodb
  ... """
  >>> config = configparser.ConfigParser(allow_no_val
  >>> config.read_string(sample_config)

  >>> # Settings with values are treated as before:
  >>> config["mysqld"]["user"]
  'mysql'

  >>> # Settings without values provide None:
  >>> config["mysqld"]["skip-bdb"]

  >>> # Settings which aren't specified still raise a
  >>> config["mysqld"]["does-not-exist"]
  Traceback (most recent call last):
    ...
  KeyError: 'does-not-exist'
  ```

- *delimiters*, valor por defecto: `('=', ':')`

  Los *delimiters* son cadenas de caracteres que separan las claves de los valores dentro de una sección. La primera ocurrencia de una cadena de separación en una línea se considera como un separador. Esto significa que los valores pueden contener separadores, no así las claves.

  Vea también el argumento *space_around_delimiters* de **`ConfigParser.write()`**.

- *comment_prefixes*, valor por defecto: `('#', ';')`

- *inline_comment_prefixes*, valor por defecto: `None`

  Los prefijos de comentario son cadenas de caracteres que indican el inicio de un comentario válido dentro de un archivo de configuración. Los *comment_prefixes* son utilizados solamente en lo que serían líneas en blanco (opcionalmente indentadas), mientras que *inline_comment_prefixes* pueden ser utilizados después de cada valor válido (ej. nombres de sección, opciones y también líneas en blanco). De forma predeterminada, los comentarios en la misma línea están deshabilitados, y tanto `'#'` como `';'` se utilizan como prefijos para comentarios que ocupan toda la línea.

  Distinto en la versión 3.2: In previous versions of **`configparser`** behaviour matched `comment_prefixes=('#',';')` and `inline_comment_prefixes=(';',)`.

  Por favor, observe que los *config parsers* no soportan el escapado de los prefijos de comentarios, de modo que la utilización de los *inline_comment_prefixes* puede impedir que los usuarios especifiquen valores de opciones que incluyan caracteres empleados como prefijos de comentarios. Ante la duda, evite especificar los *inline_comment_prefixes*. En cualquier circunstancia, la única forma de almacenar caracteres prefijos de comentario al inicio de una línea, en valores multilínea, es mediante la interpolación del prefijo, por ejemplo:

```
>>> from configparser import ConfigParser, Extended
>>> parser = ConfigParser(interpolation=ExtendedInt
>>> # the default BasicInterpolation could be used
>>> parser.read_string("""
... [DEFAULT]
... hash = #
...
... [hashes]
... shebang =
...   ${hash}!/usr/bin/env python
...   ${hash} -*- coding: utf-8 -*-
...
... extensions =
...   enabled_extension
...   another_extension
...   #disabled_by_comment
...   yet_another_extension
...
... interpolation not necessary = if # is not at li
... even in multiline values = line #1
...   line #2
...   line #3
... """)
>>> print(parser['hashes']['shebang'])

#!/usr/bin/env python
# -*- coding: utf-8 -*-
>>> print(parser['hashes']['extensions'])

enabled_extension
another_extension
yet_another_extension
>>> print(parser['hashes']['interpolation not neces
if # is not at line start
>>> print(parser['hashes']['even in multiline value
line #1
line #2
line #3
```

- *strict*, valor por defecto: `True`

  When set to `True`, the parser will not allow for any section or option duplicates while reading from a single source (using `read_file()`, `read_string()` or `read_dict()`). It is recommended to use strict parsers in new applications.

  Distinto en la versión 3.2: In previous versions of **`configparser`** behaviour matched `strict=False`.

- *empty_lines_in_values*, valor por defecto: `True`

  En los *config parsers*, los valores pueden abarcar varias líneas, siempre y cuando estén indentadas a un nivel superior al de la clave que las contiene. De forma predeterminada, los *parsers* permiten que las líneas en blanco formen parte de los valores. Igualmente, las claves pueden estar indentadas a sí mismas de forma arbitraria, a fin de mejorar la legibilidad. Por lo tanto, cuando los archivos de configuración se vuelven grandes y complejos, es fácil para el usuario el perder la pista de la estructura del archivo. Tomemos como ejemplo:

  ```
  [Section]
  key = multiline
    value with a gotcha

   this = is still a part of the multiline value of '
  ```

  Esto puede ser especialmente problemático de observar por parte del usuario, en caso que se esté utilizando un tipo de fuente proporcional para editar el archivo. Y es por ello que, cuando tu aplicación no necesite valores con líneas en blanco, debes considerar invalidarlas. Esto ocasionaría que las líneas en blanco sirvan para dividir a las claves, siempre. En el ejemplo anterior, produciría dos claves: `key` y `this`.

- *default_section*, valor por defecto: `configparser.DEFAULTSECT` (es decir: `"DEFAULT"`)

  El convenio de permitir una sección especial con valores por defecto para otras secciones, o con propósito de interpolación, es un concepto poderoso de esta librería, el cual permite a los usuarios crear configuraciones declarativas complejas. Esta sección se suele denominar `"DEFAULT"`, pero puede personalizarse para corresponder a cualquier otro nombre de sección. Algunas valores comunes son: `"general"` ó `"common"`. El nombre proporcionado es utilizado para reconocer las secciones por defecto al momento de realizar la lectura desde cualquier fuente, y es utilizado ante cualquier escritura al archivo de configuración. Su valor actual puede obtenerse utilizando el atributo `parser_instance.default_section` y puede ser modificado en tiempo de ejecución (es decir, para convertir archivos de un formato a otro).

- *interpolation*, valor por defecto: `configparser.BasicInterpolation`

  El comportamiento de la interpolación puede ser personalizado al proporcionar un gestor personalizado mediante el argumento *interpolation*. El valor `None` se utiliza para desactivar la interpolación por completo, mientras que `ExtendedInterpolation()` proporciona una variante más avanzada, inspirada por `zc.buildout`. Más información al respecto en la sección dedicada de la documentación. El **`RawConfigParser`** tiene un valor por defecto de `None`.

- *converters*, valor por defecto: no definido

  Config parsers provide option value getters that perform type conversion. By default **`getint()`**, **`getfloat()`**, and **`getboolean()`** are implemented. Should other getters be desirable, users may define them in a subclass or pass a dictionary where each key is a name of the converter and each value is a callable implementing said conversion. For instance, passing `{'decimal': decimal.Decimal}` would add **`getdecimal()`** on both the parser object and all section proxies. In other words, it will be possible to write both `parser_instance.getdecimal('section', 'key', fallback=0)` and `parser_instance['section'].getdecimal('key', 0)`.

Si el conversor necesita acceder al estado del *parser*, este puede ser implementado como un método en una subclase del *config parser*. Si el nombre de este método comienza con `get`, estará disponible en todas las secciones proxy, en su forma compatible con diccionarios (vea el ejemplo anterior de `getdecimal()`).

Una personalización más avanzada puede conseguirse al sustituir los valores por defecto de estos atributos del *parser*. Los valores por defecto son definidos en las clases, de modo que pueden ser sustituidos por las subclases o mediante la asignación de atributos.

ConfigParser.BOOLEAN_STATES

Por defecto, al utilizar el método **`getboolean()`**, los *config parsers* consideran a los siguientes valores como `True`: `'1'`, `'yes'`, `'true'`, `'on'` y a los siguientes valores como `False`: `'0'`, `'no'`, `'false'`, `'off'`. Puedes cambiar ello proporcionando un diccionario personalizado de cadenas de caracteres, con sus correspondientes valores booleanos. Por ejemplo:

```
>>> custom = configparser.ConfigParser()
>>> custom['section1'] = {'funky': 'nope'}
>>> custom['section1'].getboolean('funky')
Traceback (most recent call last):
...
ValueError: Not a boolean: nope
>>> custom.BOOLEAN_STATES = {'sure': True, 'nope':
>>> custom['section1'].getboolean('funky')
False
```

Otros pares booleanos comunes incluyen `accept`/`reject` ó `enabled`/`disabled`.

ConfigParser.optionxform(*option*)

Este método realiza la transformación de los nombres de opciones para cada operación de lectura, obtención o asignación. Por defecto convierte los nombres a minúsculas. Esto significa que, cuando un archivo de configuración es escrito, todas las claves son convertidas a minúsculas. Sobreescribe este método si tal comportamiento no es adecuado. Por ejemplo:

```
>>> config = """
... [Section1]
... Key = Value
...
... [Section2]
... AnotherKey = Value
... """
>>> typical = configparser.ConfigParser()
>>> typical.read_string(config)
>>> list(typical['Section1'].keys())
['key']
>>> list(typical['Section2'].keys())
['anotherkey']
>>> custom = configparser.RawConfigParser()
>>> custom.optionxform = lambda option: option
>>> custom.read_string(config)
>>> list(custom['Section1'].keys())
['Key']
>>> list(custom['Section2'].keys())
['AnotherKey']
```

Nota

La función *optionxform* transforma los nombres de opción a una forma canónica. Esta debería ser una función idempotente: si el nombre ya está en su forma canónica, debería retornarse sin cambios.

ConfigParser.SECTCRE

Es una expresión regular compilada que se utiliza para parsear cabeceras de sección. Por defecto hace corresponder `[section]` con el nombre `"section"`. Los espacios en blanco son considerados parte del nombre de sección, por lo que `[  larch  ]` será leído como la sección de nombre `"` `larch   `". Sobre-escribe este atributo si tal comportamiento no es adecuado. Por ejemplo:

```
>>> import re
>>> config = """
... [Section 1]
... option = value
...
... [  Section 2  ]
... another = val
... """
>>> typical = configparser.ConfigParser()
>>> typical.read_string(config)
>>> typical.sections()
['Section 1', '  Section 2  ']
>>> custom = configparser.ConfigParser()
>>> custom.SECTCRE = re.compile(r"\[ *(?P<header>[^
>>> custom.read_string(config)
>>> custom.sections()
['Section 1', 'Section 2']
```

Nota

Mientras que los objetos *ConfigParser* también utilizan un atributo `OPTCRE` para reconocer las líneas de opciones, no se recomienda su sobre-escritura porque puede interferir con las opciones *allow_no_value* y *delimiters* del constructor.

Ejemplos de la API heredada

Mainly because of backwards compatibility concerns, **`configparser`** provides also a legacy API with explicit `get`/`set` methods. While there are valid use cases for the methods outlined below, mapping protocol access is preferred for new projects. The legacy API is at times more advanced, low-level and downright counterintuitive.

Un ejemplo de escritura a un archivo de configuración:

```
import configparser

config = configparser.RawConfigParser()

# Please note that using RawConfigParser's set functions
# non-string values to keys internally, but will receive
# attempting to write to a file or when you get it in no
# values using the mapping protocol or ConfigParser's se
# such assignments to take place.
config.add_section('Section1')
config.set('Section1', 'an_int', '15')
config.set('Section1', 'a_bool', 'true')
config.set('Section1', 'a_float', '3.1415')
config.set('Section1', 'baz', 'fun')
config.set('Section1', 'bar', 'Python')
config.set('Section1', 'foo', '%(bar)s is %(baz)s!')

# Writing our configuration file to 'example.cfg'
with open('example.cfg', 'w') as configfile:
    config.write(configfile)
```

Un ejemplo de lectura de un archivo de configuración, nuevamente:

```
import configparser

config = configparser.RawConfigParser()
config.read('example.cfg')

# getfloat() raises an exception if the value is not a f
# getint() and getboolean() also do this for their respe
a_float = config.getfloat('Section1', 'a_float')
an_int = config.getint('Section1', 'an_int')
print(a_float + an_int)

# Notice that the next output does not interpolate '%(ba
# This is because we are using a RawConfigParser().
if config.getboolean('Section1', 'a_bool'):
    print(config.get('Section1', 'foo'))
```

Para obtener la interpolación, utilice `ConfigParser`:

```
import configparser

cfg = configparser.ConfigParser()
cfg.read('example.cfg')

# Set the optional *raw* argument of get() to True if yo
# interpolation in a single get operation.
print(cfg.get('Section1', 'foo', raw=False))  # -> "Pyth
print(cfg.get('Section1', 'foo', raw=True))   # -> "%(ba

# The optional *vars* argument is a dict with members th
# precedence in interpolation.
print(cfg.get('Section1', 'foo', vars={'bar': 'Documenta
                                       'baz': 'evil'}))

# The optional *fallback* argument can be used to provic
print(cfg.get('Section1', 'foo'))
      # -> "Python is fun!"

print(cfg.get('Section1', 'foo', fallback='Monty is not.
      # -> "Python is fun!"

print(cfg.get('Section1', 'monster', fallback='No such t
      # -> "No such things as monsters."

# A bare print(cfg.get('Section1', 'monster')) would rai
# but we can also use:

print(cfg.get('Section1', 'monster', fallback=None))
      # -> None
```

Los valores por defecto están disponibles en ambos tipos de ConfigParsers. Ellos son utilizados en la interpolación cuando una opción utilizada no está definida en otro lugar.

```
import configparser
```

```
# New instance with 'bar' and 'baz' defaulting to 'Life'
config = configparser.ConfigParser({'bar': 'Life', 'baz'
config.read('example.cfg')

print(config.get('Section1', 'foo'))     # -> "Python is
config.remove_option('Section1', 'bar')
config.remove_option('Section1', 'baz')
print(config.get('Section1', 'foo'))     # -> "Life is h
```

Objetos ConfigParser

class configparser.ConfigParser(*defaults=None*, *dict_type=dict*, *allow_no_value=False*, ***, *delimiters=('=', ':')*, *comment_prefixes=('#', ';')*, *inline_comment_prefixes=None*, *strict=True*, *empty_lines_in_values=True*, *default_section=configparser.DEFAULTSECT*, *interpolation=BasicInterpolation()*, *converters={}*, *allow_unnamed_section=False*)

La *config parser* principal. Si se proporciona *defaults*, su valor es inicializado en el diccionario de valores por defecto intrínsecos. Si se proporciona *dict_type*, se utiliza para crear el diccionario de objetos para la lista de secciones, las opciones dentro de una sección, y los valores por defecto.

Si se proporciona *delimiters*, se utiliza como un conjunto de cadenas de caracteres que dividen las claves de los valores. Si se proporciona *comment_prefixes*, se utiliza como un conjunto de cadenas de caracteres que preceden a los comentarios, en líneas que estarían, de otro modo, vacías. Los comentarios pueden estar indentados. Si se proporciona *inline_comment_prefixes*, se utiliza como un conjunto de cadenas de caracteres que preceden a los comentarios en líneas que no están vacías.

Cuando *strict* es `True` (por defecto), el *parser* no permitirá duplicados en ninguna sección u opción, al realizar la lectura de una sola fuente (archivo, cadena de caracteres o diccionario), generando una excepción **`DuplicateSectionError`** ó **`DuplicateOptionError`**.

Cuando *empty_lines_in_values* es `False` (valor por defecto: `True`), cada línea en blanco indica el fin de una opción. De otro modo, las líneas en blanco de una opción multilínea son tratadas como parte del valor de esta. Cuando *allow_no_value* es `True` (valor por defecto: `False`), se aceptan opciones sin valores; el valor que toman esas opciones es `None` y serán serializadas sin el delimitador final.

Cuando se proporciona *default_section*, se define el nombre de la sección especial que contiene valores por defecto para otras secciones y con propósito de interpolación (habitualmente denominada `"DEFAULT"`). Este valor puede obtenerse y modificarse en tiempo de ejecución utilizando el atributo de instancia `default_section`. Esto no volverá a evaluar un fichero de configuración previamente analizado sintácticamente (*parsed*), pero se usará al escribir configuraciones de analizado sintáctico (*parsed*) en un nuevo fichero de configuración.

La interpolación puede personalizarse al proporcionar un gestor personalizado, mediante el argumento *interpolation*. El valor `None` se utiliza para desactivar la interpolación completamente, `ExtendedInterpolation()` proporciona una variante avanzada, inspirada en `zc.buildout`. Más al respecto puede encontrarse en la correspondiente sección de la documentación.

Todos los nombres de opción que se utilizan en la interpolación pasarán por el método **`optionxform()`**, igual que cualquier otra referencia a un nombre de opción. Por lo tanto, si se utiliza la implementación por defecto de **`optionxform()`** (la cual convierte los nombres de opción a minúsculas), los valores `foo %(bar)s` y `foo %(BAR)s` son equivalentes.

When *converters* is given, it should be a dictionary where each key represents the name of a type converter and each value is a callable implementing the conversion from string to the desired datatype. Every converter gets its own corresponding **`get*()`** method on the parser object and section proxies.

When *allow_unnamed_section* is `True` (default: `False`), the first section name can be omitted. See the «Unnamed Sections» section.

It is possible to read several configurations into a single **`ConfigParser`**, where the most recently added configuration has the highest priority. Any conflicting keys are taken from the more recent configuration while the previously existing keys are retained. The example below reads in an `override.ini` file, which will override any conflicting keys from the `example.ini` file.

```
[DEFAULT]
ServerAliveInterval = -1
```

```
>>> config_override = configparser.ConfigParser()
>>> config_override['DEFAULT'] = {'ServerAliveInter
>>> with open('override.ini', 'w') as configfile:
...     config_override.write(configfile)
...
>>> config_override = configparser.ConfigParser()
>>> config_override.read(['example.ini', 'override.
['example.ini', 'override.ini']
>>> print(config_override.get('DEFAULT', 'ServerAli
-1
```

Distinto en la versión 3.1: El valor por defecto de *dict_type* es **`collections.OrderedDict`**.

Distinto en la versión 3.2: se agregaron los argumentos *allow_no_value*, *delimiters*, *comment_prefixes*, *strict*, *empty_lines_in_values*, *default_section* y *interpolation*.

Distinto en la versión 3.5: Se agregó el argumento *converters*.

Distinto en la versión 3.7: The *defaults* argument is read with **`read_dict()`**, providing consistent behavior across the parser: non-string keys and values are implicitly converted to strings.

Distinto en la versión 3.8: El valor por defecto para *dict_type* es `dict`, dado que este ahora preserva el orden de inserción.

Distinto en la versión 3.13: Raise a `MultilineContinuationError` when *allow_no_value* is `True`, and a key without a value is continued with an indented line.

Distinto en la versión 3.13: The *allow_unnamed_section* argument was added.

defaults()

Retorna un diccionario que contiene los valores por defecto para toda la instancia.

sections()

Retorna una lista de las secciones disponibles; *default section* no se incluye en la lista.

add_section(*section*)

Agrega una sección llamada *section* a la instancia. Si ya existe una sección con el nombre proporcionado, se genera la excepción `DuplicateSectionError`. Si se suministra el nombre *default section*, se genera la excepción `ValueError`. El nombre de la sección debe ser una cadena de caracteres, de lo contrario, se genera la excepción `TypeError`.

Distinto en la versión 3.2: Nombres de sección que no sean del tipo cadena de caracteres generan la excepción `TypeError`.

has_section(*section*)

Indica si la sección de nombre *section* existe en la configuración. No se permite *default section*.

options(*section*)

Retorna una lista de opciones disponibles en la sección especificada.

has_option(*section*, *option*)

Si la sección indicada existe, y esta contiene las opción proporcionada, retorna **`True`**; de lo contrario, retorna **`False`**. Si la sección especificada es **`None`** o una cadena de caracteres vacía, se asume DEFAULT.

read(*filenames*, *encoding=None*)

Intenta leer y analizar sintácticamente (*parse*) un iterable de nombres de archivos, retornando una lista de nombres de archivos que han sido analizados (*parsed*) con éxito.

Si *filenames* es una cadena de caracteres, un objeto **`bytes`** o un path-like object, es tratado como un simple nombre de archivo. Si un archivo mencionado en *filenames* no puede ser abierto, será ignorado. Está diseñado de forma que puedas especificar un iterable de potenciales ubicaciones para archivos de configuración (por ejemplo, el directorio actual, el directorio *home* del usuario, o algún directorio del sistema), y todos los archivos de configuración existentes serán leídos.

Si no existe ninguno de los archivos mencionados, la instancia de **`ConfigParser`** contendrá un conjunto de datos vacío. Una aplicación que requiera valores iniciales, que sean cargados desde un archivo, deberá cargar el(los) archivo(s) requerido(s) utilizando **`read_file()`** antes de llamar a **`read()`** para cualquier otro archivo opcional:

```
import configparser, os

config = configparser.ConfigParser()
config.read_file(open('defaults.cfg'))
config.read(['site.cfg', os.path.expanduser('~/
            encoding='cp1250')
```

Distinto en la versión 3.2: Added the *encoding* parameter.

Previously, all files were read using the default encoding for `open()`.

Distinto en la versión 3.6.1: El parámetro *filenames* acepta un path-like object.

Distinto en la versión 3.7: El parámetro *filenames* acepta un objeto `bytes`.

read_file(*f*, *source=None*)

Leer y analizar (*parse*) los datos de configuración de *f*, el cual debe ser un iterable que retorne cadenas de caracteres Unicode (por ejemplo, archivos abiertos en modo texto).

Optional argument *source* specifies the name of the file being read. If not given and *f* has a `name` attribute, that is used for *source*; the default is `'<???>'`.

Added in version 3.2: Replaces `readfp()`.

read_string(*string*, *source='<string>'*)

Analiza (*parse*) los datos de configuración desde una cadena de caracteres.

El argumento opcional *source* especifica un nombre para la cadena de caracteres proporcionada, relativo al contexto. Si no se proporciona, se utiliza `'<string>'`. Esto, por lo general, debería ser una ruta de archivo o una URL.

Added in version 3.2.

read_dict(*dictionary*, *source='<dict>'*)

Carga la configuración a partir de cualquier objeto que proporcione un método `items()`, similar a un diccionario. Las claves son nombres de secciones, los valores son diccionarios con claves y valores que deben estar presentes en la sección. Si el tipo de diccionario utilizado preserva el orden, las secciones y sus claves serán agregados en orden. Los valores son convertidos a cadenas de caracteres de forma automática.

El argumento opcional *source* especifica un nombre para el diccionario proporcionado, relativo al contexto. Si no se proporciona, se utiliza `<dict>`.

Este método puede utilizarse para copiar el estado entre *parsers*.

Added in version 3.2.

get(*section*, *option*, ***, *raw=False*, *vars=None*[, *fallback*])

Obtiene el valor de una *option* para la sección indicada. Si se proporciona *vars*, tiene que ser un diccionario. El valor de *option* será buscado en *vars* (si se proporciona), en *section* y en *DEFAULTSECT*, en ese orden. Si la clave no se encuentra, y se ha proporcionado *fallback*, este es utilizado como un valor de contingencia. Se puede utilizar `None` como valor de contingencia (*fallback*).

Todas las interpolaciones `'%'` son expandidas en el valor retornado, a menos que el argumento *raw* sea *true*. Los valores para la interpolación de las claves son buscados de la misma forma que para la opción.

Distinto en la versión 3.2: Los argumentos *raw*, *vars* y *fallback* son argumentos nombrados solamente, a fin de proteger a los usuarios de los intentos de emplear el tercer argumento como el valor de contingencia de *fallback* (especialmente cuando se utiliza el protocolo de mapeo).

getint(*section*, *option*, ***, *raw=False*, *vars=None*[, *fallback*])

Un cómodo método para forzar a entero el valor de la opción de la sección indicada. Revise **`get()`** para una explicación acerca de *raw*, *vars* y *fallback*.

getfloat(*section*, *option*, ***, *raw=False*, *vars=None*[, *fallback*])

A convenience method which coerces the *option* in the specified *section* to a floating-point number. See `get()` for explanation of *raw*, *vars* and *fallback*.

getboolean(*section*, *option*, ***, *raw=False*, *vars=None*[, *fallback*])

Un cómodo método para forzar a booleano el valor de la opción de la sección indicada. Tome nota que los valores aceptados para la opción son `'1'`, `'yes'`, `'true'`, and `'on'`, para que el método retorne `True`, y `'0'`, `'no'`, `'false'`, y `'off'` para que el método retorne `False`. Esos valores de cadenas de caracteres son revisados sin diferenciar mayúsculas de minúsculas. Cualquier otro valor ocasionará que se genere la excepción **`ValueError`**. Revise `get()` para una explicación acerca de *raw*, *vars* y *fallback*.

items(*raw=False*, *vars=None*)
items(*section*, *raw=False*, *vars=None*)

Cuando no se proporciona el argumento *section*, retorna una lista de pares *section_name*, *section_proxy*, incluyendo DEFAULTSECT.

De lo contrario, retorna una lista de pares *name*, *value* para las opciones de la sección especificada. Los argumentos opcionales tienen el mismo significado que en el método `get()`.

Distinto en la versión 3.8: Los elementos que estén en *vars* no aparecen en el resultado. El comportamiento previo mezcla las opciones actuales del *parser* con las variables proporcionadas para la interpolación.

set(*section*, *option*, *value*)

Si la sección indicada existe, asigna el valor especificado a la opción indicada; en caso contrario, genera la excepción `NoSectionError`. *option* y *value* deben ser cadenas de caracteres; de lo contrario, se genera la excepción `TypeError`.

write(*fileobject*, *space_around_delimiters=True*)

Escribe una representación de la configuración al file object especificado, el cual debe ser abierto en modo texto (aceptando cadenas de caracteres). Esta representación puede ser analizada (*parsed*) por una posterior llamada a `read()`. Si *space_around_delimiters* es *true*, los delimitadores entre claves y valores son rodeados por espacios.

Distinto en la versión 3.14: Raises InvalidWriteError if this would write a representation which cannot be accurately parsed by a future `read()` call from this parser.

Nota

Los comentarios presentes en el fichero original no se mantienen cuando se vuelve a escribir la configuración. Qué es considerado un comentario depende de los valores asignados a *comment_prefix* e *inline_comment_prefix*.

remove_option(*section*, *option*)

Elimina la opción especificada de la sección indicada. Si la sección no existe, se genera una excepción `NoSectionError`. Si la opción existía antes de la eliminación, retorna `True`; de lo contrario retorna `False`.

remove_section(*section*)

Elimina la sección especificada de la configuración. Si la sección existía, retorna `True`. De lo contrario, retorna `False`.

optionxform(*option*)

Transforma el nombre de opción *option* de la forma en que se encontraba en el archivo de entrada o como fue pasada por el código del cliente, a la forma en que debe ser utilizada en las estructuras internas. La implementación por defecto retorna una versión en minúsculas de *option*; las subclases pueden sobreescribirla o el código del cliente puede asignar un atributo de su nombre en las instancias, para afectar su comportamiento.

No necesitas crear una subclase del *parser* para utilizar este método, ya que también puedes asignarlo en una instancia a una función, la cual reciba una cadena de caracteres como argumento y retorne otra. Por ejemplo, estableciéndola a `str`, se hará que los nombres de opciones sean sensibles a mayúsculas y minúsculas:

```
cfgparser = ConfigParser()
cfgparser.optionxform = str
```

Tome en cuenta que cuando se leen archivos de configuración, los espacios en blanco alrededor de los nombres de opción son eliminados antes de llamar a **`optionxform()`**.

configparser.UNNAMED_SECTION

A special object representing a section name used to reference the unnamed section (see Unnamed Sections).

configparser.MAX_INTERPOLATION_DEPTH

The maximum depth for recursive interpolation for **`get()`** when the *raw* parameter is false. This is relevant only when the default *interpolation* is used.

Objetos RawConfigParser

class configparser.RawConfigParser(*defaults=None, dict_type=dict, allow_no_value=False, *, delimiters=('=', ':'), comment_prefixes=('#', ';'), inline_comment_prefixes=None, strict=True,* *empty_lines_in_values = True, default_section = configparser.DEFAULTSECT, interpolation = BasicInterpolation(), converters = {}, allow_unnamed_section = False*)

Variante heredada de `ConfigParser`. Tiene la interpolación deshabilitada por defecto y permite nombres de sección que no sean cadenas de caracteres, nombres de opciones, y valores a través de sus métodos inseguros `add_section` y `set`, así como el manejo heredado del argumento nombrado `defaults=`.

Distinto en la versión 3.2: se agregaron los argumentos *allow_no_value*, *delimiters*, *comment_prefixes*, *strict*, *empty_lines_in_values*, *default_section* y *interpolation*.

Distinto en la versión 3.5: Se agregó el argumento *converters*.

Distinto en la versión 3.8: El valor por defecto para *dict_type* es `dict`, dado que este ahora preserva el orden de inserción.

Distinto en la versión 3.13: The *allow_unnamed_section* argument was added.

Nota

Considere el uso de `ConfigParser` en su lugar, el cual verifica los tipos de datos de los valores que se almacenarán internamente. Si no quieres la interpolación, puedes utilizar `ConfigParser(interpolation=None)`.

add_section(*section*)

Add a section named *section* or `UNNAMED_SECTION` to the instance.

If the given section already exists, `DuplicateSectionError` is raised. If the *default section* name is passed, `ValueError` is raised. If `UNNAMED_SECTION` is passed and support is disabled, `UnnamedSectionDisabledError` is raised.

No se comprueba el tipo de datos de *section*, con lo cual se permite que los usuarios creen secciones con nombres que no sean cadenas de caracteres. Este comportamiento no está soportado y puede ocasionar errores internos.

Distinto en la versión 3.14: Added support for `UNNAMED_SECTION`.

set(*section*, *option*, *value*)

Si existe la sección indicada, asigna el valor especificado a la sección indicada; de lo contrario, genera la excepción `NoSectionError`. Aunque es posible utilizar `RawConfigParser` (ó `ConfigParser` con parámetros *raw* con valor *true*) como almacenamiento interno para valores que no sean cadenas de caracteres, el funcionamiento completo (incluyendo la interpolación y escritura en archivos) sólo puede lograrse utilizando valores del tipo cadena de caracteres.

Este método permite que los usuarios asignen valores que no sean cadenas de caracteres a las claves, internamente. Este comportamiento no está soportado y causará errores cuando se intente escribir en un archivo o cuando se intente obtenerlo en un modo no *raw*. **Utilice la API del protocolo de mapeo**, la cual no permite ese tipo de asignaciones.

Excepciones

exception configparser.Error

Base class for all other `configparser` exceptions.

exception configparser.NoSectionError

Excepción generada cuando no se encuentra una sección especificada.

exception configparser.DuplicateSectionError

Exception raised if `add_section()` is called with the name of a section that is already present or in strict parsers when a section if found more than once in a single input file, string or dictionary.

Distinto en la versión 3.2: Added the optional *source* and *lineno* attributes and parameters to `__init__()`.

exception configparser.DuplicateOptionError

Excepción generada por *parsers* estrictos si una sola opción aparece dos veces durante la lectura de un solo archivo, cadena de caracteres o diccionario. Captura errores de escritura o relacionados con el uso de mayúsculas y minúsculas, ej. un diccionario podría tener dos claves que representan la misma clave de configuración bajo un esquema insensible a mayúsculas y minúsculas.

exception configparser.NoOptionError

Excepción generada cuando una opción especificada no se encuentra en una sección indicada.

exception configparser.InterpolationError

Clase base para excepciones generadas por problemas que ocurren al realizar la interpolación de cadenas de caracteres.

exception configparser.InterpolationDepthError

Excepción generada cuando la interpolación de cadenas de caracteres no puede completarse, debido a que el número de iteraciones excede a `MAX_INTERPOLATION_DEPTH`. Subclase de `InterpolationError`.

exception configparser.InterpolationMissingOptionError

Excepción generada cuando no existe una opción referida por un valor. Subclase de `InterpolationError`.

exception configparser.InterpolationSyntaxError

Excepción generada cuando el texto fuente, donde se realizan las sustituciones, no se ajusta a la sintaxis requerida. Subclase de **`InterpolationError`**.

exception configparser.MissingSectionHeaderError

Excepción generada cuando se intenta analizar (*parse*) un archivo que no tiene encabezados de sección.

exception configparser.ParsingError

Excepción generada cuando ocurren errores intentando analizar (*parse*) un archivo.

Distinto en la versión 3.12: The `filename` attribute and **`__init__()`** constructor argument were removed. They have been available using the name `source` since 3.2.

exception configparser.MultilineContinuationError

Exception raised when a key without a corresponding value is continued with an indented line.

Added in version 3.13.

exception configparser.UnnamedSectionDisabledError

Exception raised when attempting to use the **`UNNAMED_SECTION`** without enabling it.

Added in version 3.14.

exception configparser.InvalidWriteError

Exception raised when an attempted **`ConfigParser.write()`** would not be parsed accurately with a future **`ConfigParser.read()`** call.

Ex: Writing a key beginning with the **`ConfigParser.SECTCRE`** pattern would parse as a section header when read. Attempting to write this will raise this exception.

Added in version 3.14.

Notas al pie

Los *config parsers* permiten una gran personalización. Si estás interesado en modificar el comportamiento descrito por la referencia de la nota al pie, consulta la sección Customizing Parser Behaviour.

`tomllib` — Parse TOML files

Added in version 3.11.

Código fuente: Lib/tomllib [https://github.com/python/cpython/tree/3.14/Lib/tomllib]

This module provides an interface for parsing TOML 1.0.0 (Tom's Obvious Minimal Language, https://toml.io [https://toml.io/en/]). This module does not support writing TOML.

Advertencia

Be cautious when parsing data from untrusted sources. A malicious TOML string may cause the decoder to consume considerable CPU and memory resources. Limiting the size of data to be parsed is recommended.

Ver también

The Tomli-W package [https://pypi.org/project/tomli-w/] is a TOML writer that can be used in conjunction with this module, providing a write API familiar to users of the standard library **`marshal`** and **`pickle`** modules.

Ver también

The TOML Kit package [https://pypi.org/project/tomlkit/] is a style-preserving TOML library with both read and write capability. It is a recommended replacement for this module for editing already existing TOML files.

Este módulo define las siguientes funciones:

tomllib.load(*fp*, /, *, *parse_float=float*)

Lee un archivo TOML. El primer argumento debe ser un objeto de archivo legible y binario. Retorna un **`dict`**. Convierte los tipos TOML a Python utilizando esta tabla de conversión.

parse_float se llamará con la cadena de cada TOML flotante que se decodificará. Por defecto, esto es equivalente a `float(num_str)`. Esto se puede utilizar para usar otro tipo de datos o analizador para flotantes TOML (por ejemplo, **`decimal.Decimal`**). La llamada no debe devolver un **`dict`** o un **`list`**, de lo contrario se lanza un **`ValueError`**.

Un **`TOMLDecodeError`** se levantará en un documento TOML inválido.

tomllib.loads(*s*, /, *, *parse_float=float*)

Carga TOML de un objeto **`str`**. Retorna un **`dict`**. Convierte los tipos TOML a Python utilizando esta tabla de conversión. El argumento *parse_float* tiene el mismo significado que en **`load()`**.

Un **`TOMLDecodeError`** se levantará en un documento TOML inválido.

Las siguientes excepciones están disponibles:

exception tomllib.TOMLDecodeError(*msg*, *doc*, *pos*)

Subclass of **`ValueError`** with the following additional attributes:

msg

The unformatted error message.

doc

The TOML document being parsed.

pos

The index of *doc* where parsing failed.

lineno

The line corresponding to *pos*.

colno

The column corresponding to *pos*.

Distinto en la versión 3.14: Added the *msg*, *doc* and *pos* parameters. Added the `msg`, `doc`, `pos`, `lineno` and `colno` attributes.

Obsoleto desde la versión 3.14: Passing free-form positional arguments is deprecated.

Ejemplos

Analiza un TOML file:

```
import tomllib

with open("pyproject.toml", "rb") as f:
    data = tomllib.load(f)
```

Analiza un TOML string:

```
import tomllib

toml_str = """
python-version = "3.11.0"
python-implementation = "CPython"
"""

data = tomllib.loads(toml_str)
```

Tabla de conversión

TOML	Python

TOML document	dict
cadena	str
integer	int
float	flotante (configurable con *parse_float*)
boolean	bool
offset date-time	datetime.datetime (atributo `tzinfo` establecido en una instancia de `datetime.timezone`)
local date-time	datetime.datetime (atributo `tzinfo` establecido en `None`)
local date	datetime.date
local time	datetime.time
array	list
tabla	dict
inline table	dict
array of tables	list of dicts

`netrc` — netrc file processing

Código fuente: Lib/netrc.py [https://github.com/python/cpython/tree/3.14/Lib/netrc.py]

La clase `netrc` analiza y encapsula el formato del fichero netrc, usado por el programa Unix **ftp** y otros clientes FTP.

class netrc.netrc([*file*])

A `netrc` instance or subclass instance encapsulates data from a netrc file. The initialization argument, if present, specifies the file to parse. If no argument is given, the file `.netrc` in the user's home directory – as determined by `os.path.expanduser()` – will be read. Otherwise, a `FileNotFoundError` exception will be raised. Parse errors will raise `NetrcParseError` with diagnostic information including the file name, line number, and terminating token.

If no argument is specified on a POSIX system, the presence of passwords in the `.netrc` file will raise a `NetrcParseError` if the file ownership or permissions are insecure (owned by a user other than the user running the process, or accessible for read or write by any other user). This implements security behavior equivalent to that of ftp and other programs that use `.netrc`. Such security checks are not available on platforms that do not support `os.getuid()`.

Distinto en la versión 3.4: Añadida la comprobación de permisos POSIX.

Distinto en la versión 3.7: `os.path.expanduser()` se usa para encontrar la localización del fichero `.netrc` cuando *file* no se pasa como argumento.

Distinto en la versión 3.10: `netrc` prueba la codificación

UTF-8 antes de usar la codificación específica en la configuración regional. Ya no es necesario que la entrada en el archivo netrc contenga todos los tokens. El valor predeterminado para los tokens faltantes es una cadena de caracteres vacía. Todos los tokens y sus valores ahora pueden contener caracteres arbitrarios, como espacios en blanco y caracteres no ASCII. Si el nombre de login es anónimo, no se disparará el chequeo de seguridad.

exception netrc.NetrcParseError

Excepción lanzada por la clase **`netrc`** cuando se encuentran errores sintácticos en el texto origen. Las instancias de esta excepción ofrecen tres atributos interesantes:

msg

Explicación textual del error.

filename

El nombre del archivo fuente.

lineno

El número de línea donde se encontró el error.

Objetos netrc

Una instancia **`netrc`** tiene los siguientes métodos:

netrc.authenticators(*host*)

Retorna una 3-tupla `(login, account, password)` para autenticarse contra *host*. Si el fichero netrc no contiene una entrada para el host dado, retorna una tupla asociada con la entrada por defecto. Si no están disponibles ni el host correspondiente ni la entrada por defecto, retorna `None`.

netrc.__repr__()

Vuelca los datos de la clase como una cadena de caracteres en el formato de un fichero netrc. (Esto descarta comentarios y puede reordenar las entradas.)

Las instancias de `netrc` tienen variables de instancia públicas:

netrc.hosts

Diccionario que asocia nombres de hosts a tuplas `(login, account, password)`. La entrada por defecto, si existe, está representada como un pseudo-host por ese nombre.

netrc.macros

Diccionario que asocia nombres de macros a listas de cadenas de caracteres.

`plistlib` — Generate and parse Apple `.plist` files

Código fuente: Lib/plistlib.py [https://github.com/python/cpython/tree/3.14/Lib/plistlib.py]

Este módulo proporciona una interfaz para leer y escribir los archivos de «lista de propiedades» utilizados por Apple, principalmente en macOS e iOS. Este módulo admite archivos plist binarios y XML.

El formato de lista de propiedades (`.plist`) es una serialización simple que soporta tipos de objetos básicos, como diccionarios, listas, números y cadenas. Generalmente el objeto de nivel superior es un diccionario.

Para escribir y analizar un archivo plist usa las funciones **`dump()`** y **`load()`**.

To work with plist data in bytes or string objects, use **`dumps()`** and **`loads()`**.

Los valores pueden ser cadenas, enteros, flotantes, booleanos, tuplas, listas, diccionarios (pero solo con claves de cadena), **`bytes`**, **`bytearray`** u objetos **`datetime.datetime`**.

Distinto en la versión 3.4: Nueva API, API antigua obsoleta. Añadido soporte para plists en formato binario.

Distinto en la versión 3.8: Añadido soporte para lectura y escritura de tokens **`UID`** en plists binarios como lo usan NSKeyedArchiver y NSKeyedUnarchiver.

Distinto en la versión 3.9: API antigua eliminada.

Ver también

PList manual page [https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/PropertyLists/]
Documentación de Apple del formato de archivo.

Este módulo define las siguientes funciones:

plistlib.load(*fp*, ***, *fmt=None*, *dict_type=dict*, *aware_datetime=False*)

Lee un archivo plist. *fp* debe ser un objeto de archivo binario y legible. Retorna el objeto raíz desempaquetado (el cual generalmente es un diccionario).

El *fmt* es el formato del archivo y los siguientes valores son válidos:

- `None`: Autodetecta el formato de archivo
- `FMT_XML`: Formato de archivo XML
- `FMT_BINARY`: Formato binario plist

EL *dict_type* es el tipo usado por los diccionarios que son leídos del archivo plist.

When *aware_datetime* is true, fields with type `datetime.datetime` will be created as aware object, with `tzinfo` as `datetime.UTC`.

Los datos XML para el formato `FMT_XML` son analizados usando el analizador Expat desde `xml.parsers.expat` – consulte su documentación para conocer las posibles excepciones en XML mal formado. Elementos desconocidos serán simplemente ignorados por el analizador plist.

The parser raises `InvalidFileException` when the file cannot be parsed.

Added in version 3.4.

Distinto en la versión 3.13: The keyword-only parameter *aware_datetime* has been added.

plistlib.loads(*data*, *, *fmt=None*, *dict_type=dict*, *aware_datetime=False*)

Load a plist from a bytes or string object. See **`load()`** for an explanation of the keyword arguments.

Added in version 3.4.

Distinto en la versión 3.13: data can be a string when *fmt* equals **`FMT_XML`**.

plistlib.dump(*value*, *fp*, *, *fmt=FMT_XML*, *sort_keys=True*, *skipkeys=False*, *aware_datetime=False*)

Write *value* to a plist file. *fp* should be a writable, binary file object.

El argumento *fmt* especifica el formato del archivo plist y puede ser uno de los siguientes valores:

- **`FMT_XML`**: Archivo plist con formato XML
- **`FMT_BINARY`**: Archivo plist con formato binario

Cuando *sort_keys* es verdadero (el valor por defecto) las claves para los diccionarios serán escritas al plist ordenadamente, si no serán escritas en el orden de iteración del diccionario.

Cuando *skipkeys* es falso (el valor por defecto) la función lanza **`TypeError`** cuando una clave del diccionario no es una cadena de caracteres, si no tales claves serán omitidas.

When *aware_datetime* is true and any field with type `datetime.datetime` is set as an aware object, it will convert to UTC timezone before writing it.

Una excepción **`TypeError`** será levantada si el objeto es un tipo no admitido o el contenedor que contiene tipos no admitidos.

Una excepción **`OverflowError`** será levantada para valores enteros que no pueden ser representados en archivos plist (binarios).

Added in version 3.4.

Distinto en la versión 3.13: The keyword-only parameter *aware_datetime* has been added.

plistlib.dumps(*value*, ***, *fmt=FMT_XML*, *sort_keys=True*, *skipkeys=False*, *aware_datetime=False*)

Retorna *value* como un objeto de bytes con formato plist. Consulte la documentación de **`dump()`** para una explicación de los argumentos de palabra clave de esta función.

Added in version 3.4.

Las siguientes clases están disponibles:

class plistlib.UID(*data*)

Envuelve un **`int`**. Este es usado leyendo o escribiendo datos codificados NSKeyedArchiver, los cuales contienen UID (ver manual PList).

data

Int value of the UID. It must be in the range `0 <= data < 2**64`.

Added in version 3.8.

Las siguientes constantes están disponibles:

plistlib.FMT_XML

El formato XML para archivos plist.

Added in version 3.4.

plistlib.FMT_BINARY

El formato binario para archivos plist

Added in version 3.4.

The module defines the following exceptions:

exception plistlib.InvalidFileException

Raised when a file cannot be parsed.

Added in version 3.4.

Ejemplos

Generar un plist:

```
import datetime as dt
import plistlib

pl = dict(
    aString = "Doodah",
    aList = ["A", "B", 12, 32.1, [1, 2, 3]],
    aFloat = 0.1,
    anInt = 728,
    aDict = dict(
        anotherString = "<hello & hi there!>",
        aThirdString = "M\xe4ssig, Ma\xdf",
        aTrueValue = True,
        aFalseValue = False,
    ),
    someData = b"<binary gunk>",
    someMoreData = b"<lots of binary gunk>" * 10,
    aDate = dt.datetime.now()
)
print(plistlib.dumps(pl).decode())
```

Analizar un plist:

```
import plistlib

plist = b"""<plist version="1.0">
<dict>
    <key>foo</key>
    <string>bar</string>
</dict>
</plist>"""
```

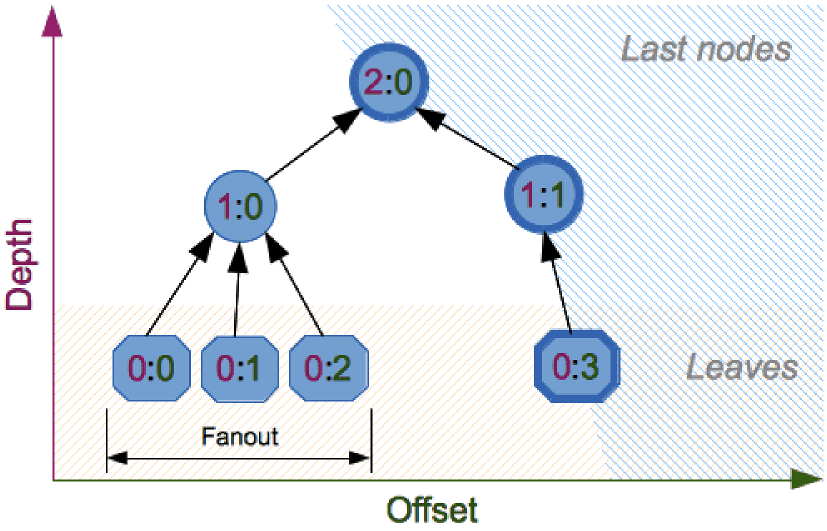
```
pl = plistlib.loads(plist)
print(pl["foo"])
```

Servicios criptográficos

The modules described in this chapter implement various algorithms of a cryptographic nature. They are available at the discretion of the installation. Here's an overview:

- **`hashlib`** — Secure hashes and message digests
    - Algoritmos de hash
    - Usage
    - Constructors
    - Attributes
    - Hash Objects
    - Resúmenes SHAKE de largo variable
    - Cifrado de archivos
    - Derivación de clave
    - BLAKE2
        - Creando objetos hash
        - Constantes
        - Ejemplos
            - Cifrado simple
            - Usar diferentes tamaños de resumen
            - Cifrado de clave
            - Cifrado aleatorio
            - Personalización
            - Modo de árbol
        - Créditos
- **`hmac`** — Keyed-Hashing for Message Authentication
- **`secrets`** — Generate secure random numbers for managing secrets
    - Números aleatorios
    - Generando tokens

    - ¿Cuántos bytes deben tener los tokens?
- Otras funciones
- Recetas y mejores prácticas



`hashlib` — Secure hashes and message digests

Código fuente: Lib/hashlib.py [https://github.com/python/cpython/tree/3.14/Lib/hashlib.py]

This module implements a common interface to many different hash algorithms. Included are the FIPS secure hash algorithms SHA224, SHA256, SHA384, SHA512, (defined in the FIPS 180-4 standard [https://csrc.nist.gov/pubs/fips/180-4/upd1/final]), the SHA-3 series (defined in the FIPS 202 standard [https://csrc.nist.gov/pubs/fips/202/final]) as well as the legacy algorithms SHA1 (formerly part of FIPS [https://csrc.nist.gov/news/2023/decision-to-revise-fips-180-4]) and the MD5 algorithm (defined in internet **RFC 1321** [https://datatracker.ietf.org/doc/html/rfc1321.html]).

Nota

Si quieres las funciones de hash adler32 o crc32, están disponibles en el módulo **`zlib`**.

Algoritmos de hash

There is one constructor method named for each type of *hash*. All return a hash object with the same simple interface. For example: use `sha256()` to create a SHA-256 hash object. You can now feed this object with bytes-like objects (normally `bytes`) using the `update` method. At any point you can ask it for the *digest* of the concatenation of the data fed to it so far using the `digest()` or `hexdigest()` methods.

To allow multithreading, the Python GIL is released while computing a hash supplied more than 2047 bytes of data at once in its constructor or `.update` method.

Constructors for hash algorithms that are always present in this module are `sha1()`, `sha224()`, `sha256()`, `sha384()`, `sha512()`, `sha3_224()`, `sha3_256()`, `sha3_384()`, `sha3_512()`, `shake_128()`, `shake_256()`, `blake2b()`, and `blake2s()`. `md5()` is normally available as well, though it may be missing or blocked if you are using a rare «FIPS compliant» build of Python. These correspond to `algorithms_guaranteed`.

Additional algorithms may also be available if your Python distribution's **`hashlib`** was linked against a build of OpenSSL that provides others. Others *are not guaranteed available* on all installations and will only be accessible by name via `new()`. See `algorithms_available`.

Advertencia

Some algorithms have known hash collision weaknesses (including MD5 and SHA1). Refer to Attacks on cryptographic hash algorithms [https://en.wikipedia.org/wiki/Cryptographic_hash_function#Attacks_on_cryptographic_hash_algorithms] and the hashlib-seealso section at the end of this document.

Added in version 3.6: SHA3 (Keccak) and SHAKE constructors `sha3_224()`, `sha3_256()`, `sha3_384()`, `sha3_512()`, `shake_128()`, `shake_256()` were added. `blake2b()` and

`blake2s()` were added.

Distinto en la versión 3.9: Todos los constructores hashlib toman un argumento de solo palabra clave *usedforsecurity* con el valor predeterminado `True`. Un valor falso permite el uso de algoritmos hash inseguros y bloqueados en entornos restringidos. `False` indica que el algoritmo hash no se utiliza en un contexto de seguridad, por ejemplo, como una función de compresión unidireccional no criptográfica.

Distinto en la versión 3.9: Hashlib now uses SHA3 and SHAKE from OpenSSL if it provides it.

Distinto en la versión 3.12: For any of the MD5, SHA1, SHA2, or SHA3 algorithms that the linked OpenSSL does not provide we fall back to a verified implementation from the HACL* project [https://github.com/hacl-star/hacl-star].

Usage

To obtain the digest of the byte string `b"Nobody inspects the spammish repetition"`:

```
>>> import hashlib
>>> m = hashlib.sha256()
>>> m.update(b"Nobody inspects")
>>> m.update(b" the spammish repetition")
>>> m.digest()
b'\x03\x1e\xdd}Ae\x15\x93\xc5\xfe\\\x00o\xa5u+7\xfd\xdf\
>>> m.hexdigest()
'031edd7d41651593c5fe5c006fa5752b37fddff7bc4e843aa6af0c9
```

Más resumido:

```
>>> hashlib.sha256(b"Nobody inspects the spammish repeti
'031edd7d41651593c5fe5c006fa5752b37fddff7bc4e843aa6af0c9
```

Constructors

hashlib.new(*name*, [*data*,]*, *usedforsecurity=True*)

Is a generic constructor that takes the string *name* of the desired algorithm as its first parameter. It also exists to allow access to the above listed hashes as well as any other algorithms that your OpenSSL library may offer.

Using `new()` with an algorithm name:

```
>>> h = hashlib.new('sha256')
>>> h.update(b"Nobody inspects the spammish repetition")
>>> h.hexdigest()
'031edd7d41651593c5fe5c006fa5752b37fddff7bc4e843aa6af0c9
```

hashlib.md5([*data*,]*, *usedforsecurity=True*)

hashlib.sha1([*data*,]*, *usedforsecurity=True*)

hashlib.sha224([*data*,]*, *usedforsecurity=True*)

hashlib.sha256([*data*,]*, *usedforsecurity=True*)

hashlib.sha384([*data*,]*, *usedforsecurity=True*)

hashlib.sha512([*data*,]*, *usedforsecurity=True*)

hashlib.sha3_224([*data*,]*, *usedforsecurity=True*)

hashlib.sha3_256([*data*,]*, *usedforsecurity=True*)

hashlib.sha3_384([*data*,]*, *usedforsecurity=True*)

hashlib.sha3_512([*data*,]*, *usedforsecurity=True*)

Named constructors such as these are faster than passing an algorithm name to `new()`.

Attributes

Hashlib provides the following constant module attributes:

hashlib.algorithms_guaranteed

Un conjunto que contiene los nombres de los algoritmos garantizados a ser soportados por este módulo en todas las plataformas. Ten en cuenta que "md5" se encuentra en esta lista a pesar de que algunos proveedores ofrecen una extraña construcción Python «compatible con FIPS» que la excluye.

Added in version 3.2.

hashlib.algorithms_available

Un conjunto que contiene los nombres de los algoritmos de hash que están disponibles en el intérprete de Python en ejecución. Estos nombres serán reconocidos cuando sean pasados a `new()`. `algorithms_guaranteed` siempre será un subconjunto. El mismo algoritmo puede aparecer múltiples veces en este conjunto bajo diferentes nombres (gracias a OpenSSL).

Added in version 3.2.

Hash Objects

Los siguientes valores son provistos como atributos constantes de los objetos hash retornados por los constructores:

hash.digest_size

El tamaño del hash resultante en bytes.

hash.block_size

El tamaño del bloque interno del algoritmo de hash en bytes.

Un objeto hash tiene los siguientes atributos:

hash.name

El nombre canónico de este hash, siempre en minúsculas y siempre adecuado como un parámetro a **`new()`** para crear otro hash de este tipo.

Distinto en la versión 3.4: El atributo *name* ha estado presente en CPython desde su inicio, pero desde Python 3.4 no fue especificado formalmente, por lo que puede no existir en algunas plataformas.

Un objeto hash tiene los siguientes métodos:

hash.update(*data*)

Actualiza el objeto de hash con el bytes-like object. Invocaciones repetidas son equivalentes a una única invocación con la concatenación de todos los argumentos: `m.update(a); m.update(b)` es equivalente a `m.update(a+b)`.

hash.digest()

Retorna el resumen de los datos pasados al método **`update()`** hasta el momento. Este es un objeto de bytes de tamaño **`digest_size`** el cual puede contener bytes en el rango completo desde 0 a 255.

hash.hexdigest()

Como **`digest()`** excepto que el resumen es retornado como un objeto de cadena del doble de largo, conteniendo sólo dígitos hexadecimales. Este puede ser usado para intercambiar el valor de forma segura en correos electrónicos u otros entornos no binarios.

hash.copy()

Retorna una copia («clon») del objeto hash. Este puede ser usado para calcular eficientemente los resúmenes de datos compartiendo una subcadena inicial común.

Resúmenes SHAKE de largo variable

hashlib.shake_128([*data*,]*, *usedforsecurity=True*)

hashlib.shake_256([*data*,]*, *usedforsecurity=True*)

Los algoritmos `shake_128()` y `shake_256()` proveen resúmenes de largo variable con largo_en_bits//2 hasta 128 ó 256 bits de seguridad. Como tales, sus métodos de resumen requieren un largo. El largo máximo no está limitado por el algoritmo SHAKE.

shake.digest(*length*)

> Return the digest of the data passed to the `update()` method so far. This is a bytes object of size *length* which may contain bytes in the whole range from 0 to 255.

shake.hexdigest(*length*)

> Like `digest()` except the digest is returned as a string object of double length, containing only hexadecimal digits. This may be used to exchange the value in email or other non-binary environments.

Example use:

```
>>> h = hashlib.shake_256(b'Nobody inspects the spammish
>>> h.hexdigest(20)
'44709d6fcb83d92a76dcb0b668c98e1b1d3dafe7'
```

Cifrado de archivos

El módulo hashlib proporciona una función de ayuda para el cifrado eficiente de un archivo o un objeto similar a un archivo.

hashlib.file_digest(*fileobj*, *digest*, */*)

> Retorna un objeto digest que ha sido actualizado con el contenido del objeto de tipo archivo.
>
> *fileobj* must be a file-like object opened for reading in binary mode. It accepts file objects from builtin `open()`, `BytesIO`

instances, SocketIO objects from **`socket.socket.makefile()`**, and similar. *fileobj* must be opened in blocking mode, otherwise a **`BlockingIOError`** may be raised.

The function may bypass Python's I/O and use the file descriptor from **`fileno()`** directly. *fileobj* must be assumed to be in an unknown state after this function returns or raises. It is up to the caller to close *fileobj*.

digest debe ser un nombre de algoritmo de cifrado como *str*, un constructor de cifrado o un invocable que devuelva un objeto cifrado.

Ejemplo:

```
>>> import io, hashlib, hmac
>>> with open("library/hashlib.rst", "rb") as f:
...     digest = hashlib.file_digest(f, "sha256")
...
>>> digest.hexdigest()
'...'

>>> buf = io.BytesIO(b"somedata")
>>> mac1 = hmac.HMAC(b"key", digestmod=hashlib.sha5
>>> digest = hashlib.file_digest(buf, lambda: mac1)

>>> digest is mac1
True
>>> mac2 = hmac.HMAC(b"key", b"somedata", digestmod
>>> mac1.digest() == mac2.digest()
True
```

Added in version 3.11.

Distinto en la versión 3.14: Now raises a **`BlockingIOError`** if the file is opened in non-blocking mode. Previously, spurious null bytes were added to the digest.

Derivación de clave

Los algoritmos de derivación de clave y estiramiento de clave están diseñados para el cifrado seguro de contraseña. Algoritmos ingenuos como `sha1(password)` no son resistentes contra ataques de fuerza bruta. Una buena función hash de contraseña debe ser afinable, lenta e incluir una sal [https://en.wikipedia.org/wiki/Salt_%28cryptography%29].

hashlib.pbkdf2_hmac(*hash_name*, *password*, *salt*, *iterations*, *dklen=None*)

La función provee contraseñas PKCS#5 basadas en función de derivación de clave 2. Usa HMAC como función de pseudoaleatoriedad.

La cadena *hash_name* es el nombre deseado del algoritmo de resumen de hash para HMAC, ej. “sha1” o “sha256”. *password* y *salt* son interpretados como búferes de bytes. Aplicaciones y bibliotecas deberían limitar *password* a un largo razonable (ej. 1024). *salt* debería ser sobre 16 o más bytes desde una fuente adecuada, ej. **`os.urandom()`**.

El número de *iterations* debe elegirse en función del algoritmo de cifrado y la potencia de cálculo. A partir de 2022, se sugieren cientos de miles de iteraciones de SHA-256. Para saber por qué y cómo elegir lo mejor para su aplicación, lea el *Apéndice A.2.2* de NIST-SP-800-132 [https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf]. Las respuestas sobre stackexchange pbkdf2 iterations question [https://security.stackexchange.com/questions/3959/recommended-of-iterations-when-using-pbkdf2-sha256/] lo explican en detalle.

dklen is the length of the derived key in bytes. If *dklen* is `None` then the digest size of the hash algorithm *hash_name* is used, e.g. 64 for SHA-512.

```
>>> from hashlib import pbkdf2_hmac
>>> our_app_iters = 500_000  # Application specific
>>> dk = pbkdf2_hmac('sha256', b'password', b'bad s
>>> dk.hex()
'15530bba69924174860db778f2c6f8104d3aaf9d26241840c8
```

Function only available when Python is compiled with OpenSSL.

Added in version 3.4.

Distinto en la versión 3.12: Function now only available when Python is built with OpenSSL. The slow pure Python implementation has been removed.

hashlib.scrypt(*password*, ***, *salt*, *n*, *r*, *p*, *maxmem=0*, *dklen=64*)

La función provee una contraseña scrypt basada en una función derivación de clave como es definida en **RFC 7914** [https://datatracker.ietf.org/doc/html/rfc7914.html].

password y *salt* deben ser objetos de bytes. Aplicaciones y bibliotecas deberían limitar *password* a un largo razonable (ej. 1024). *salt* debería ser aproximadamente 16 o más bytes de una fuente adecuada, ej. `os.unrandom()`.

n is the CPU/Memory cost factor, *r* the block size, *p* parallelization factor and *maxmem* limits memory (OpenSSL 1.1.0 defaults to 32 MiB). *dklen* is the length of the derived key in bytes.

Added in version 3.6.

BLAKE2

BLAKE2 [https://www.blake2.net] es una función de hash criptográfico definida en **RFC 7693** [https://datatracker.ietf.org/doc/html/rfc7693.html] que viene en dos sabores:

- **BLAKE2b**, optimizada para plataformas de 64 bits y produce resúmenes de cualquier tamaño entre 1 y 64 bytes,
- **BLAKE2s**, optimizada para plataformas de 8 a 32 bits y produce resúmenes de cualquier tamaño entre 1 y 32 bytes.

BLAKE2 proporciona el **modo keyed** (un remplazamiento más simple rápido para HMAC [https://en.wikipedia.org/wiki/Hash-based_message_authentication_code]), **cifrado salado** (*salted hashing*),

personalización y **cifrado de árbol**.

Hash objects from this module follow the API of standard library's `hashlib` objects.

Creando objetos hash

Se crean nuevos objetos hash invocando a las funciones de constructor:

hashlib.blake2b(*data=b'', *, digest_size=64, key=b'', salt=b'', person=b'', fanout=1, depth=1, leaf_size=0, node_offset=0, node_depth=0, inner_size=0, last_node=False, usedforsecurity=True*)

hashlib.blake2s(*data=b'', *, digest_size=32, key=b'', salt=b'', person=b'', fanout=1, depth=1, leaf_size=0, node_offset=0, node_depth=0, inner_size=0, last_node=False, usedforsecurity=True*)

Estas funciones retornan los objetos hash correspondientes para calcular BLAKE2b o BLAKE2s. Ellas toman opcionalmente estos parámetros generales:

- *data*: trozo inicial de datos a cifrar, el cual debe ser un bytes-like object. Puede ser pasado sólo como argumento posicional.
- *digest_size*: tamaño del resumen de salida en bytes.
- *key*: clave para el cifrado de clave (*keyed hashing*) (hasta 64 bytes para BLAKE2b, hasta 32 bytes para BLAKE2s).
- *salt*: sal para el cifrado aleatorio (hasta 16 bytes para BLAKE2b, hasta 8 bytes para BLAKE2s).
- *person*: cadena de personalización (hasta 16 bytes para BLAKE2b, hasta 8 bytes para BLAKE2s).

La siguiente tabla muestra los límites para parámetros generales (en bytes):

Hash	**digest_size**	**len(key)**	**len(salt)**	**len(person)**
BLAKE2b	64	64	16	16
BLAKE2s	32	32	8	8

Nota

La especificación BLAKE2 define largos constantes para los parámetros de sal y personalización, sin embargo, por conveniencia, esta implementación acepta cadenas de bytes de cualquier tamaño hasta el largo especificado. Si el largo del parámetro es menor que el especificado, es acolchado con ceros, por lo tanto, por ejemplo, `b'salt'` y `b'salt\x00'` es el mismo valor. (Este no es el caso para *key*.)

Estos tamaños están disponibles como constantes del módulo descritas abajo.

Las funciones constructoras también aceptan los siguientes parámetros de cifrado de árbol:

- *fanout*: despliegue en abanico (0 a 255, 0 si ilimitado, 1 en modo secuencial).
- *depth*: profundidad máxima del árbol (1 a 255, 255 si ilimitado, 1 en modo secuencial).
- *leaf_size*: tamaño máximo en bytes de hoja (0 a `2**32-1`, 0 si ilimitado o en modo secuencial).
- *node_offset*: desplazamiento del nodo (0 a `2**64-1` para BLAKE2b, 0 a `2**48-1` para BLAKE2s, 0 para la primera hoja más a la izquierda, o en modo secuencial).
- *node_depth*: profundidad de nodo (0 a 255, 0 para hojas o en modo secuencial).
- *inner_size*: tamaño interno del resumen (0 a 64 para BLAKE2b, 0 a 32 para BLAKE2s, 0 en modo secuencial).
- *last_node*: booleano indicando si el nodo procesado es el último (`False` para modo secuencial).

See section 2.10 in BLAKE2 specification [https://www.blake2.net/blake2_20130129.pdf] for comprehensive review of tree hashing.

Constantes

blake2b.SALT_SIZE

blake2s.SALT_SIZE

Largo de sal (largo máximo aceptado por los constructores).

blake2b.PERSON_SIZE

blake2s.PERSON_SIZE

Largo de cadena de personalización (largo máximo aceptado por los constructores).

blake2b.MAX_KEY_SIZE

blake2s.MAX_KEY_SIZE

Tamaño máximo de clave.

blake2b.MAX_DIGEST_SIZE

blake2s.MAX_DIGEST_SIZE

Tamaño máximo de resumen que puede producir la función hash.

Ejemplos

Cifrado simple

To calculate hash of some data, you should first construct a hash object by calling the appropriate constructor function (**`blake2b()`** or **`blake2s()`**), then update it with the data by calling **`update()`** on the object, and, finally, get the digest out of the object by calling **`digest()`** (or **`hexdigest()`** for hex-encoded string).

```
>>> from hashlib import blake2b
>>> h = blake2b()
>>> h.update(b'Hello world')
>>> h.hexdigest()
'6ff843ba685842aa82031d3f53c48b66326df7639a63d128974c5c1
```

Como atajo, puedes pasar el primer trozo de datos para actualizar directamente el constructor como el argumento posicional:

```
>>> from hashlib import blake2b
>>> blake2b(b'Hello world').hexdigest()
'6ff843ba685842aa82031d3f53c48b66326df7639a63d128974c5c1
```

Puedes invocar `hash.update()` tantas veces como necesites para actualizar el hash iterativamente:

```
>>> from hashlib import blake2b
>>> items = [b'Hello', b' ', b'world']
>>> h = blake2b()
>>> for item in items:
...     h.update(item)
...
>>> h.hexdigest()
'6ff843ba685842aa82031d3f53c48b66326df7639a63d128974c5c1
```

Usar diferentes tamaños de resumen

BLAKE2 tiene tamaño de resúmenes configurables de hasta 64 bytes para BLAKE2b y 32 bytes para BLAKE2s. Por ejemplo, para remplazar SHA-1 con BLAKE2b sin cambiar el tamaño de la salida, puedes decirle a BLAKE2b que produzca resúmenes de 20 bytes:

```
>>> from hashlib import blake2b
>>> h = blake2b(digest_size=20)
>>> h.update(b'Replacing SHA1 with the more secure funct
>>> h.hexdigest()
'd24f26cf8de66472d58d4e1b1774b4c9158b1f4c'
>>> h.digest_size
20
>>> len(h.digest())
20
```

Objetos hash con diferentes tamaños de resumen tienen salidas completamente diferentes (hashes más cortos *no* son prefijos de hashes más largos); BLAKE2b y BLAKE2s producen salidas diferentes incluso si el largo de salida es el mismo:

```
>>> from hashlib import blake2b, blake2s
>>> blake2b(digest_size=10).hexdigest()
```

```
'6fa1d8fcfd719046d762'
>>> blake2b(digest_size=11).hexdigest()
'eb6ec15daf9546254f0809'
>>> blake2s(digest_size=10).hexdigest()
'1bf21a98c78a1c376ae9'
>>> blake2s(digest_size=11).hexdigest()
'567004bf96e4a25773ebf4'
```

Cifrado de clave

El cifrado de clave puede ser usado para autentificación como un remplazo más rápido y simple para Código de autentificación de mensajes en clave-hash [https://es.wikipedia.org/wiki/HMAC] (HMAC). BLAKE2 puede ser usado de forma segura en modo de prefijo MAC gracias a la propiedad de indiferenciabilidad heredada de BLAKE.

Este ejemplo muestra como obtener un código de autentificación (codificado como hexadecimal) de 128 bits para el mensaje `b'message data'` con la clave `b'pseudorandom key'`:

```
>>> from hashlib import blake2b
>>> h = blake2b(key=b'pseudorandom key', digest_size=16)
>>> h.update(b'message data')
>>> h.hexdigest()
'3d363ff7401e02026f4a4687d4863ced'
```

Como ejemplo práctico, una aplicación web puede firmar simétricamente cookies enviadas a los usuarios y verificarlas más tarde para asegurar que no fueron manipuladas con:

```
>>> from hashlib import blake2b
>>> from hmac import compare_digest
>>>
>>> SECRET_KEY = b'pseudorandomly generated server secre
>>> AUTH_SIZE = 16
>>>
>>> def sign(cookie):
...     h = blake2b(digest_size=AUTH_SIZE, key=SECRET_KE
...     h.update(cookie)
...     return h.hexdigest().encode('utf-8')
>>>
>>> def verify(cookie, sig):
...     good_sig = sign(cookie)
...     return compare_digest(good_sig, sig)
>>>
>>> cookie = b'user-alice'
>>> sig = sign(cookie)
>>> print("{0},{1}".format(cookie.decode('utf-8'), sig))
user-alice,b'43b3c982cf697e0c5ab22172d1ca7421'
>>> verify(cookie, sig)
True
>>> verify(b'user-bob', sig)
False
>>> verify(cookie, b'0102030405060708090a0b0c0d0e0f00')
False
```

Incluso aunque hay un modo de cifrado de claves nativo, BLAKE2 puede, por supuesto, ser usado en construcción de HMAC con el módulo **`hmac`**:

```
>>> import hmac, hashlib
>>> m = hmac.new(b'secret key', digestmod=hashlib.blake2
>>> m.update(b'message')
>>> m.hexdigest()
'e3c8102868d28b5ff85fc35dda07329970d1a01e273c37481326fe0
```

Cifrado aleatorio

Definiendo el parámetro *salt* los usuarios pueden introducir aleatoriedad a la función hash. El cifrado aleatorio es útil para proteger contra ataques de colisión en la función hash usada en firmas digitales.

> El cifrado aleatorio está diseñado para situaciones en las que una parte, el preparador del mensaje, genera todo o parte de un mensaje para ser firmado por una segunda parte, el firmante del mensaje. Si el preparador del mensaje es capaz de encontrar

> colisiones de funciones hash criptográficas (ej., dos mensajes produciendo el mismo valor de hash), entonces ellos pueden preparar versiones significativas del mensaje que producirían el mismo valor de hash y firma digital, pero con diferentes resultados (ej., transfiriendo 1,000,000 $ a una cuenta, en lugar de 10 $), Las funciones de hash criptográfico han sido diseñadas con resistencia de colisión como objetivo principal, pero la concentración actual en el ataque a las funciones hash criptográficas puede resultar en una función hash criptográfica dada que provea menor resistencia de colisión de la esperada. El cifrado aleatorio ofrece al firmante protección adicional reduciendo la probabilidad de que un preparador puede generar dos o más mensajes que en última instancia producen el mismo valor hash durante el proceso de generación de la firma digital, — incluso si es práctico encontrar colisiones para la función hash. Sin embargo, el uso de cifrado aleatorio puede reducir la cantidad de seguridad provista por una firma digital cuando todas las porciones del mensaje son preparadas por el firmante.
>
> (NIST SP-800-106 «Randomized Hashing for Digital Signatures» [https://csrc.nist.gov/pubs/sp/800/106/final])

En BLAKE2 la sal es procesada como una entrada de una vez a la función hash durante la inicialización, en lugar de como una entrada para cada función de compresión.

Advertencia

Salted hashing (or just hashing) with BLAKE2 or any other general-purpose cryptographic hash function, such as SHA-256, is not suitable for hashing passwords. See BLAKE2 FAQ [https://www.blake2.net/#qa] for more information.

```
>>> import os
>>> from hashlib import blake2b
>>> msg = b'some message'
>>> # Calculate the first hash with a random salt.
>>> salt1 = os.urandom(blake2b.SALT_SIZE)
>>> h1 = blake2b(salt=salt1)
>>> h1.update(msg)
>>> # Calculate the second hash with a different random
>>> salt2 = os.urandom(blake2b.SALT_SIZE)
>>> h2 = blake2b(salt=salt2)
>>> h2.update(msg)
>>> # The digests are different.
>>> h1.digest() != h2.digest()
True
```

Personalización

A veces es útil forzar a la función hash para producir diferentes resúmenes para la misma entrada para diferentes propósitos. Citando a los autores de la función hash Skein:

> Recomendamos que todos los diseñadores de aplicaciones consideren seriamente hacer esto; hemos visto muchos protocolos donde un hash que es calculado en una parte del protocolo puede ser usado en una parte completamente diferente porque dos cálculos hash fueron realizados en datos similares o relacionados, y el atacante puede forzar a la aplicación a hacer las entradas hash iguales. Personalizar cada función hash usada en el protocolo resumidamente detiene este tipo de ataque.
>
> (The Skein Hash Function Family [https://www.schneier.com/wp-content/uploads/2016/02/skein.pdf], p. 21)

BLAKE2 puede ser personalizado pasando bytes al argumento *person*:

```
>>> from hashlib import blake2b
>>> FILES_HASH_PERSON = b'MyApp Files Hash'
>>> BLOCK_HASH_PERSON = b'MyApp Block Hash'
>>> h = blake2b(digest_size=32, person=FILES_HASH_PERSON
>>> h.update(b'the same content')
>>> h.hexdigest()
'20d9cd024d4fb086aae819a1432dd2466de12947831b75c5a30cf2
>>> h = blake2b(digest_size=32, person=BLOCK_HASH_PERSON
>>> h.update(b'the same content')
>>> h.hexdigest()
'cf68fb5761b9c44e7878bfb2c4c9aea52264a80b75005e65619778
```

Se puede usar también personalización en conjunto con el modo de clave para derivar diferentes claves desde una sola.

```
>>> from hashlib import blake2s
>>> from base64 import b64decode, b64encode
>>> orig_key = b64decode(b'Rm5EPJai72qcK3RGBpW3vPNfZy5OZ
>>> enc_key = blake2s(key=orig_key, person=b'kEncrypt').
>>> mac_key = blake2s(key=orig_key, person=b'kMAC').dige
>>> print(b64encode(enc_key).decode('utf-8'))
rbPb15S/Z9t+agffno5wuhB77VbRi6F9Iv2qIxU7WHw=
>>> print(b64encode(mac_key).decode('utf-8'))
G9GtHFE1YluXY1zWPlYk1e/nWfu0WSEb0KRcjhDeP/o=
```

Modo de árbol

Aquí hay un ejemplo de cifrar un árbol mínimo con dos nodos de hoja:

```
  10
 /  \
00  01
```

Este ejemplo usa resúmenes internos de 64 bytes, y retorna el resumen final de 32 bytes:

```
>>> from hashlib import blake2b
>>>
>>> FANOUT = 2
>>> DEPTH = 2
>>> LEAF_SIZE = 4096
>>> INNER_SIZE = 64
>>>
>>> buf = bytearray(6000)
>>>
>>> # Left leaf
... h00 = blake2b(buf[0:LEAF_SIZE], fanout=FANOUT, depth
...                 leaf_size=LEAF_SIZE, inner_size=INNER_
...                 node_offset=0, node_depth=0, last_node
>>> # Right leaf
... h01 = blake2b(buf[LEAF_SIZE:], fanout=FANOUT, depth=
...                 leaf_size=LEAF_SIZE, inner_size=INNER_
...                 node_offset=1, node_depth=0, last_node
>>> # Root node
... h10 = blake2b(digest_size=32, fanout=FANOUT, depth=D
...                 leaf_size=LEAF_SIZE, inner_size=INNER_
...                 node_offset=0, node_depth=1, last_node
>>> h10.update(h00.digest())
>>> h10.update(h01.digest())
>>> h10.hexdigest()
'3ad2a9b37c6070e374c7a8c508fe20ca86b6ed54e286e93a0318e95
```

Créditos

BLAKE2 [https://www.blake2.net] fue diseñado por *Jean-Philippe Aumasson*, *Samuel Neves*, *Zooko Wilcox-O'Hearn* y *Christian Winnerlein* basado en el SHA-3 [https://en.wikipedia.org/wiki/Secure_Hash_Algorithms] finalista BLAKE [https://web.archive.org/web/20200918190133/https://131002.net/blake/] creado por *Jean-Philippe Aumasson*, *Luca Henzen*, *Willi Meier* y *Raphael C.-W. Phan*.

Usa el algoritmo núcleo del cifrado ChaCha [https://cr.yp.to/chacha.html] diseñado por *Daniel J. Bernstein*.

La implementación stdlib está basada en el módulo pyblake2 [https://pythonhosted.org/pyblake2/]. Fue escrita por *Dmitry Chestnykh* basada en la implementación C escrita por *Samuel Neves*. La documentación fue copiada desde pyblake2 [https://pythonhosted.org/pyblake2/] y escrita por *Dmitry Chestnykh*.

El código C fue parcialmente reescrito para Python por *Christian Heimes*.

La siguiente dedicación de dominio público aplica tanto para la implementación de la función hash C, el código de extensión y su documentación:

> En la medida en que la ley lo permite, el/los autor/es han dedicado todos los derechos de autor y los derechos relacionados y vecinos de este software al dominio público mundial. Este software se distribuye sin ninguna garantía.
>
> Deberías haber recibido una copia de la Dedicación CC0 de Dominio Público junto a este software. Si no, consulta https://creativecommons.org/publicdomain/zero/1.0/.

Las siguientes personas han ayudado con el desarrollo o contribuyeron con sus cambios al proyecto y el dominio público de acuerdo a Creative Commons Public Domain Dedication 1.0 Universal:

- *Alexandr Sokolovskiy*

Ver también

Módulo `hmac`
Un módulo para generar mensajes de códigos de autentificación usando hashes.

Módulo `base64`
Otra forma de codificar hashes binarios para entornos no binarios.

https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.180-4.pdf
The FIPS 180-4 publication on Secure Hash Algorithms.

https://csrc.nist.gov/pubs/fips/202/final

The FIPS 202 publication on the SHA-3 Standard.

https://www.blake2.net/
Sitio web oficial de BLAKE2.

https://en.wikipedia.org/wiki/Cryptographic_hash_function
Artículo de Wikipedia con información sobre cuáles algoritmos tienen errores conocidos y lo que eso significa con respecto a su uso.

https://www.ietf.org/rfc/rfc8018.txt
PKCS #5: Password-Based Cryptography Specification Version 2.1

https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf
Recomendación de NIST para la derivación de claves basadas en contraseña.

`hmac` — Keyed-Hashing for Message Authentication

Código fuente: Lib/hmac.py [https://github.com/python/cpython/tree/3.14/Lib/hmac.py]

This module implements the HMAC algorithm as described by **RFC 2104** [https://datatracker.ietf.org/doc/html/rfc2104.html]. The interface allows to use any hash function with a *fixed* digest size. In particular, extendable output functions such as SHAKE-128 or SHAKE-256 cannot be used with HMAC.

hmac.new(*key*, *msg=None*, *digestmod*)

Retorna un nuevo objeto hmac. *key* es un objeto *bytes* o *bytearray* que proporciona la clave secreta. Si *msg* está presente, se realiza la llamada al método `update(msg)`. *digestmod* es el nombre del resumen, constructor o módulo del resumen para el objeto HMAC que se va a usar. Puede ser cualquier nombre adecuado para **`hashlib.new()`**. Se requiere este argumento a pesar de su posición.

Distinto en la versión 3.4: El parámetro *key* puede ser un objeto *bytes* o *bytearray*. El parámetro *msg* puede ser de cualquier tipo soportado por **`hashlib`**. El parámetro *digestmod* puede ser el nombre del algoritmo de *hash*.

Distinto en la versión 3.8: The *digestmod* argument is now required. Pass it as a keyword argument to avoid awkwardness when you do not have an initial *msg*.

hmac.digest(*key*, *msg*, *digest*)

Retorna el resumen de *msg* para una clave *key* secreta y un resumen *digest* dados. La función es equivalente a `HMAC(key, msg, digest).digest()`, pero utiliza una implementación optimizada en C o *inline*, que es más rápida para mensajes que caben en memoria. Los parámetros *key*, *msg* y *digest* tienen el mismo significado que en **`new()`**.

Un detalle de la implementación de CPython: la implementación optimizada en C solo se usa cuando *digest* es una cadena de caracteres y el nombre de un algoritmo de resumen, que está soportado por OpenSSL.

Added in version 3.7.

class hmac.HMAC

Un objeto HMAC tiene los siguientes métodos:

HMAC.update(*msg*)

Actualiza el objeto hmac con *msg*. Las llamadas repetidas equivalen a una sola llamada con la concatenación de todos los argumentos: `m.update(a); m.update(b)` es equivalente a `m.update(a + b)`.

Distinto en la versión 3.4: El parámetro *msg* puede ser de cualquier tipo soportado por **`hashlib`**.

HMAC.digest()

Retorna el resumen de los *bytes* que se pasaron al método **`update()`** hasta el momento. Este objeto *bytes* será de la misma longitud que el *digest_size* del resumen que se pasa al constructor. Puede contener *bytes* no ASCII, incluyendo *bytes* NUL.

Advertencia

Cuando se compara la salida de **`digest()`** a un resumen provisto externamente durante una rutina de verificación, se recomienda utilizar la función **`compare_digest()`** en lugar del operador `==` para reducir la vulnerabilidad a ataques de temporización.

HMAC.hexdigest()

Como `digest()` excepto que el resumen se retorna como una cadena de caracteres de dos veces la longitud conteniendo solo dígitos hexadecimales. Esto se puede utilizar para intercambiar el valor de forma segura en email u otros entornos no binarios.

Advertencia

Cuando se compara la salida de `hexdigest()` a un resumen provisto externamente durante una rutina de verificación, se recomienda utilizar la función `compare_digest()` en lugar del operador == para reducir la vulnerabilidad a ataques de temporización.

HMAC.copy()

Retorna una copia («clon») del objeto hmac. Esto se puede utilizar para calcular de forma eficiente los resúmenes de las cadenas de caracteres que comparten una subcadena de caracteres inicial común.

Un objeto *hash* tiene los siguientes atributos:

HMAC.digest_size

El tamaño del resumen HMAC resultante en *bytes*.

HMAC.block_size

El tamaño de bloque interno del algoritmo de *hash* en *bytes*.

Added in version 3.4.

HMAC.name

El nombre canónico de este HMAC, siempre en minúsculas, por ejemplo `hmac-md5`.

Added in version 3.4.

Distinto en la versión 3.10: Removed the undocumented attributes `HMAC.digest_cons`, `HMAC.inner`, and `HMAC.outer`.

Este módulo también provee las siguiente funciones auxiliares:

hmac.compare_digest(*a*, *b*)

Retorna `a == b`. Esta función utiliza un enfoque diseñado para prevenir el análisis de temporización evitando el comportamiento de cortocircuito basado en contenido, haciéndolo adecuado para criptografía. *a* y *b* deben ser del mismo tipo: ya sea **`str`** (solo ASCII, como por ejemplo retornado por **`HMAC.hexdigest()`**), o un objeto tipo binario.

Nota

Si *a* y *b* son de diferente longitud, o si ocurre un error, un ataque de temporización teóricamente podría revelar información sobre los tipos y longitudes de *a* y *b*—pero no sus valores.

Added in version 3.3.

Distinto en la versión 3.10: La función utiliza `CRYPTO_memcmp()` de OpenSSL internamente cuando está disponible.

Ver también

Módulo **`hashlib`**

El módulo de Python que provee funciones de *hash* seguras.

`secrets` — Generate secure random numbers for managing secrets

Added in version 3.6.

Código fuente: Lib/secrets.py [https://github.com/python/cpython/tree/3.14/Lib/secrets.py]

The `secrets` module is used for generating cryptographically strong random numbers suitable for managing data such as passwords, account authentication, security tokens, and related secrets.

In particular, `secrets` should be used in preference to the default pseudo-random number generator in the `random` module, which is designed for modelling and simulation, not security or cryptography.

Ver también

PEP 506 [https://peps.python.org/pep-0506/]

Números aleatorios

The `secrets` module provides access to the most secure source of randomness that your operating system provides.

class secrets.SystemRandom

Una clase para generar números aleatorios utilizando las fuentes de mayor calidad que proporciona el sistema operativo. Ver `random.SystemRandom` para más detalles.

secrets.choice(*seq*)

Retorna un elemento elegido aleatoriamente a partir de una secuencia no vacía.

secrets.randbelow(*exclusive_upper_bound*)

Return a random int in the range [0, *exclusive_upper_bound*).

secrets.randbits(*k*)

Return a non-negative int with *k* random bits.

Generando tokens

The `secrets` module provides functions for generating secure tokens, suitable for applications such as password resets, hard-to-guess URLs, and similar.

secrets.token_bytes(*nbytes=None*)

Return a random byte string containing *nbytes* number of bytes.

If *nbytes* is not specified or `None`, **`DEFAULT_ENTROPY`** is used instead.

```
>>> token_bytes(16)
b'\xebr\x17D*t\xae\xd4\xe3S\xb6\xe2\xebP1\x8b'
```

secrets.token_hex(*nbytes=None*)

Return a random text string, in hexadecimal. The string has *nbytes* random bytes, each byte converted to two hex digits.

If *nbytes* is not specified or `None`, **`DEFAULT_ENTROPY`** is used instead.

```
>>> token_hex(16)
'f9bf78b9a18ce6d46a0cd2b0b86df9da'
```

secrets.token_urlsafe(*nbytes=None*)

Return a random URL-safe text string, containing *nbytes* random bytes. The text is Base64 encoded, so on average each byte results in approximately 1.3 characters.

If *nbytes* is not specified or `None`, **`DEFAULT_ENTROPY`** is used instead.

```
>>> token_urlsafe(16)
'Drmhze6EPcv0fN_81Bj-nA'
```

¿Cuántos bytes deben tener los tokens?

To be secure against brute-force attacks [https://en.wikipedia.org/wiki/Brute-force_attack], tokens need to have sufficient randomness. Unfortunately, what is considered sufficient will necessarily increase as computers get more powerful and able to make more guesses in a shorter period. As of 2015, it is believed that 32 bytes (256 bits) of randomness is sufficient for the typical use-case expected for the **`secrets`** module.

Para quienes quieran gestionar la longitud de sus propios tokens, pueden especificar explícitamente cuánta aleatoriedad se utiliza para los tokens dando un argumento **`int`** a las funciones `token_*`. Ese argumento se toma como el número de bytes de aleatoriedad a utilizar.

Otherwise, if no argument is provided, or if the argument is `None`, the `token_*` functions use **`DEFAULT_ENTROPY`** instead.

secrets.DEFAULT_ENTROPY

Default number of bytes of randomness used by the `token_*` functions.

The exact value is subject to change at any time, including during maintenance releases.

Otras funciones

secrets.compare_digest(*a*, *b*)

Retorna `True` si las cadenas de caracteres o objetos tipo-bytes *a* y *b* son iguales, de lo contrario `False`, usando una «comparación de tiempo constante» para reducir el riesgo de ataques de análisis temporal [https://codahale.com/a-lesson-in-timing-attacks/]. Ver **`hmac.compare_digest`()** para detalles adicionales.

Recetas y mejores prácticas

This section shows recipes and best practices for using **`secrets`** to manage a basic level of security.

Generar una contraseña alfanumérica de ocho caracteres:

```
import string
import secrets
alphabet = string.ascii_letters + string.digits
password = ''.join(secrets.choice(alphabet) for i in rar
```

Nota

Applications should not **store passwords in a recoverable format** [https://cwe.mitre.org/data/definitions/257.html], whether plain text or encrypted. They should be salted and hashed using a cryptographically strong one-way (irreversible) hash function.

Generar una contraseña alfanumérica de diez caracteres con al menos un carácter en minúscula, al menos un carácter en mayúscula y al menos tres dígitos:

```
import string
import secrets
alphabet = string.ascii_letters + string.digits
while True:
    password = ''.join(secrets.choice(alphabet) for i ir
    if (any(c.islower() for c in password)
            and any(c.isupper() for c in password)
            and sum(c.isdigit() for c in password) >= 3)
        break
```

Generar una contraseña al estilo XKCD [https://xkcd.com/936/]:

```
import secrets
# On standard Linux systems, use a convenient dictionary
# Other platforms may need to provide their own word-lis
with open('/usr/share/dict/words') as f:
    words = [word.strip() for word in f]
    password = ' '.join(secrets.choice(words) for i in r
```

Generar una URL temporal difícil de adivinar que contenga un token de seguridad adecuado para la recuperación de contraseñas:

```
import secrets
url = 'https://example.com/reset=' + secrets.token_urlsa
```

Servicios genéricos del sistema operativo

Los módulos descritos en este capítulo proporcionan interfaces a las características del sistema operativo que están disponibles en (casi) todos los sistemas operativos, como archivos y un reloj. Las interfaces se modelan, por norma general, según las interfaces Unix o C, pero también están disponibles en la mayoría de los otros sistemas. Esta es una visión general:

- `os` — Interfaces misceláneas del sistema operativo
  - Nombres de archivos, argumentos de la línea de comandos y variables de entorno
  - Modo Python UTF-8
  - Parámetros de proceso
  - Creación de objetos de tipo archivo
  - Operaciones de descriptores de archivos
    - Consultando las dimensiones de una terminal
    - Herencia de los descriptores de archivos
  - Archivos y directorios
    - Descriptores de archivos del temporizador
    - Atributos extendidos de Linux
  - Gestión de proceso
  - Interfaz al planificador
  - Información miscelánea del sistema
  - Números al azar
- `io` — Core tools for working with streams
  - Resumen

        - E/S Texto
        - E/S Binaria
        - E/S sin formato
    - Codificación de texto
        - EncodingWarning opcional
    - Interfaz de módulo de alto nivel
    - Jerarquía de clases
        - Clases base E/S
        - Archivo sin formato E/S
        - *Streams* almacenados (búfer)
        - E/S Texto
    - Static Typing
    - Rendimiento
        - E/S Binaria
        - E/S Texto
        - Multihilo
        - Reentrada
- **`time`** — Time access and conversions
    - Las Funciones
    - Constantes de ID de reloj
    - Constantes de zona horaria
- **`logging`** — Logging facility for Python
    - Objetos logger
    - Niveles de logging
    - Gestor de objetos
    - Objetos formateadores
    - Filtro de Objetos
    - Objetos LogRecord
    - Atributos LogRecord
    - Objetos LoggerAdapter
    - Seguridad del hilo
    - Funciones a nivel de módulo

    - Atributos a nivel de módulo
    - Integración con el módulo de advertencias
- **`logging.config`** — Logging configuration
    - Funciones de configuración
    - Consideraciones de seguridad
    - Esquema del diccionario de configuración
        - Detalles del esquema del diccionario
        - Configuración incremental
        - Conexiones de objeto
        - Objetos definidos por el usuario
        - Handler configuration order
        - Acceso a objetos externos
        - Acceso a objetos internos
        - Resolución de importación e importadores personalizados
        - Configuring QueueHandler and QueueListener
    - Formato de archivo de configuración
- **`logging.handlers`** — Logging handlers
    - StreamHandler
    - FileHandler
    - NullHandler
    - WatchedFileHandler
    - BaseRotatingHandler
    - RotatingFileHandler
    - TimedRotatingFileHandler
    - SocketHandler
    - DatagramHandler
    - Gestor *SysLog* (*SysLogHandler*)
    - Gestor de eventos *NTELog* (NTEventLogHandler)
    - SMTPHandler
    - MemoryHandler
    - HTTPHandler
    - QueueHandler
    - QueueListener

- **`platform`** — Access to underlying platform’s identifying data
    - Cross platform
    - Java platform
    - Windows platform
    - macOS platform
    - iOS platform
    - Unix platforms
    - Linux platforms
    - Android platform
    - Command-line usage
- **`errno`** — Standard errno system symbols
- **`ctypes`** — A foreign function library for Python
    - tutorial de ctypes
        - Carga de bibliotecas de enlaces dinámicos
        - Acceder a las funciones de los dll cargados
        - Funciones de llamada
        - Tipos de datos fundamentales
        - Funciones de llamada, continuación
        - Calling variadic functions
        - Funciones de llamada con sus propios tipos de datos personalizados
        - Especificar los tipos de argumentos requeridos (prototipos de funciones)
        - Tipos de retorno
        - Pasar los punteros (o: pasar los parámetros por referencia)
        - Estructuras y uniones
        - Structure/union layout, alignment and byte order
        - Campos de bits en estructuras y uniones
        - Arreglos
        - Punteros
        - Thread safety without the GIL
        - Conversiones de tipos
        - Tipos incompletos
        - Funciones de retrollamadas (*callback*)
        - Acceder a los valores exportados de los dlls
        - Sorpresas

    - Tipos de datos de tamaño variable
  - referencia ctypes
    - Encontrar bibliotecas compartidas
    - Listing loaded shared libraries
    - Cargando bibliotecas compartidas
    - Funciones foráneas
    - Prototipos de funciones
    - Funciones de utilidad
    - Tipos de datos
    - Tipos de datos fundamentales
    - Tipos de datos estructurados
    - Arreglos y punteros
    - Exceptions

`os` — Interfaces misceláneas del sistema operativo

Código fuente: Lib/os.py [https://github.com/python/cpython/tree/3.14/Lib/os.py]

Este módulo provee una manera versátil de usar funcionalidades dependientes del sistema operativo. Si quieres leer o escribir un archivo mira `open()`, si quieres manipular rutas, mira el módulo `os.path`, y si quieres leer todas las líneas en todos los archivos en la línea de comandos mira el módulo `fileinput`. Para crear archivos temporales y directorios mira el módulo `tempfile`, y para el manejo de alto nivel de archivos y directorios puedes ver el módulo `shutil`.

Notas sobre la disponibilidad de estas funciones:

- El diseño de todos los módulos incorporados de Python dependientes del sistema operativo es tal que, mientras funcionalidad esté disponible, usará la misma interfaz; por ejemplo, la función `os.stat(path)` retorna estadísticas sobre la ruta (*path*) en el mismo formato (lo que sucede originalmente con la interfaz POSIX).
- Extensions peculiar to a particular operating system are also available through the `os` module, but using them is of course a threat to portability.
- Todas las funciones que aceptan rutas o nombres de archivos aceptan *bytes* o cadenas de texto, y el resultado es un objeto del mismo tipo, siempre que se retorna una ruta o un archivo.
- En VxWorks, os.popen, os.fork, os.execv y os.spawn*p* no son compatibles.
- On WebAssembly platforms, Android and iOS, large parts of the `os` module are not available or behave differently. APIs related to processes (e.g. `fork()`, `execve()`) and resources (e.g. `nice()`) are not available. Others like `getuid()` and

`getpid()` are emulated or stubs. WebAssembly platforms also lack support for signals (e.g. `kill()`, `wait()`).

Nota

Todas las funciones en este módulo lanzan `OSError` (o subclases), en el caso de archivos o rutas inaccesibles o inválidas, u otros argumentos que tienen el tipo correcto, pero que no son aceptados por el sistema operativo.

exception os.error

Un alias de la excepción incorporada `OSError`.

os.name

El nombre del módulo dependiente del sistema operativo importado. Los siguientes nombres están registrados: `'posix'`, `'nt'`, `'java'`.

Ver también

`sys.platform` tiene una granularidad más fina. `os.uname()` proporciona información de versión dependiente del sistema.

El módulo `platform` proporciona verificaciones detalladas de la identidad del sistema.

Nombres de archivos, argumentos de la línea de comandos y variables de entorno

En Python, los nombres de archivo, los argumentos de la línea de comandos y las variables de entorno se representan mediante el tipo de cadena de caracteres. En algunos sistemas, es necesario decodificar estas cadenas desde y hacia bytes antes de pasarlas al sistema operativo. Python usa la codificación del sistema de archivos y manejador de errores para realizar esta conversión (consulte `sys.getfilesystemencoding()`).

La codificación del sistema de archivos y manejador de errores se configuran al inicio de Python mediante la función **`PyConfig_Read()`**: consulte los miembros **`filesystem_encoding`** y **`filesystem_errors`** de **`PyConfig`**.

Distinto en la versión 3.1: En algunos sistemas, la conversión mediante la codificación del sistema de archivos puede fallar. En este caso, Python utiliza surrogateescape encoding error handler, lo que significa que los bytes no decodificables se reemplazan por un carácter Unicode U+DC*xx* durante la decodificación y estos se traducen nuevamente al byte original durante la codificación.

La codificación del sistema de archivos y manejador de errores debe garantizar la decodificación exitosa de todos los bytes por debajo de 128. Si la codificación del sistema de archivos no proporciona esta garantía, las funciones de API pueden generar **`UnicodeError`**.

Consulte también la codificación de la configuración regional.

Modo Python UTF-8

Added in version 3.7: Consulte **PEP 540** [https://peps.python.org/pep-0540/] para obtener más detalles.

El modo Python UTF-8 ignora la codificación de la configuración regional y fuerza el uso de la codificación UTF-8:

- Utilice UTF-8 como codificación del sistema de archivos y manejador de errores.
- **`sys.getfilesystemencoding()`** retorna `'utf-8'`.
- **`locale.getpreferredencoding()`** retorna `'utf-8'` (el argumento *do_setlocale* no tiene ningún efecto).
- **`sys.stdin`**, **`sys.stdout`** y **`sys.stderr`** usan UTF-8 como su codificación de texto, con `surrogateescape` error handler habilitado para **`sys.stdin`** y **`sys.stdout`** (**`sys.stderr`** continúa usando `backslashreplace` como lo hace en el modo predeterminado de reconocimiento de configuración regional)
- En Unix, **`os.device_encoding()`** retorna `'utf-8'`. en lugar de la codificación del dispositivo.

Tenga en cuenta que `PYTHONIOENCODING` puede anular la configuración de transmisión estándar en el modo UTF-8 (al igual que en el modo predeterminado con reconocimiento de configuración regional).

Como consecuencia de los cambios en esas API de nivel inferior, otras API de nivel superior también exhiben diferentes comportamientos predeterminados:

- Los argumentos de la línea de comandos, las variables de entorno y los nombres de archivo se decodifican en texto utilizando la codificación UTF-8.
- `os.fsdecode()` y `os.fsencode()` utilizan la codificación UTF-8.
- `open()`, `io.open()` y `codecs.open()` utilizan la codificación UTF-8 de forma predeterminada. Sin embargo, todavía usan el controlador de errores estricto de forma predeterminada, por lo que es probable que intentar abrir un archivo binario en modo texto genere una excepción en lugar de producir datos sin sentido.

El Modo Python UTF-8 se habilita si la configuración regional LC_CTYPE es `C` o `POSIX` al iniciar Python (consulte la función `PyConfig_Read()`).

Se puede habilitar o deshabilitar usando la opción de línea de comando `-X utf8` y la variable de entorno `PYTHONUTF8`.

Si la variable de entorno `PYTHONUTF8` no está configurada en absoluto, entonces el intérprete utiliza de forma predeterminada la configuración regional actual, *a no ser que* la configuración regional actual se identifica como una configuración regional heredada basada en ASCII (como se describe para `PYTHONCOERCECLOCALE`) y la coerción de la configuración regional está deshabilitada o falla . En tales configuraciones regionales heredadas, el intérprete habilitará de forma predeterminada el modo UTF-8 a menos que se le indique explícitamente que no lo haga.

El modo Python UTF-8 solo se puede habilitar al inicio de Python. Su valor se puede leer en `sys.flags.utf8_mode`.

Consulte también modo UTF-8 en Windows y codificación del sistema de archivos y manejador de errores.

Ver también

PEP 686 [https://peps.python.org/pep-0686/]
Python 3.15 hará que Modo Python UTF-8 sea el predeterminado.

Parámetros de proceso

Estas funciones y elementos de datos proporcionan información y operan en el proceso y con el usuario actuales.

os.ctermid()

Retorna el nombre del archivo correspondiente al terminal que controla el proceso.

Availability: Unix, not WASI.

os.environ

Un objeto mapping donde las claves y los valores son cadenas que representan el entorno del proceso. Por ejemplo, `environ['HOME']` es el nombre de ruta de su directorio de inicio (en algunas plataformas) y es equivalente a `getenv("HOME")` en C.

This mapping is captured the first time the **`os`** module is imported, typically during Python startup as part of processing `site.py`. Changes to the environment made after this time are not reflected in **`os.environ`**, except for changes made by modifying **`os.environ`** directly.

Este mapeo puede ser usado para modificar el entorno y también para consultarlo. **`putenv()`** será llamado automáticamente cuando el mapeo sea modificado.

En Unix, claves y valores usan la función **`sys.getfilesystemencoding()`** y el controlador de errores `'surrogateescape'`. Hay que utilizar `environb` si se quiere usar una codificación diferente.

En Windows, las claves se convierten a mayúsculas. Esto también se aplica al obtener, configurar o eliminar un elemento. Por ejemplo, `environ['monty'] = 'python'` asigna a la clave `'MONTY'` el valor `'python'`.

Nota

Llamar a `putenv()` directamente no cambia `os.environ`, por lo que es mejor modificar `os.environ`.

Nota

En algunas plataformas, incluidas FreeBSD y macOS, configurar `environ` puede provocar pérdidas de memoria. Consulte la documentación del sistema para `putenv()`.

Puede eliminar elementos en esta asignación para desactivar las variables de entorno. `unsetenv()` se llamará automáticamente cuando se elimine un elemento de `os.environ` y cuando se llame a uno de los métodos `pop()` o `clear()`.

Ver también

The `os.reload_environ()` function.

Distinto en la versión 3.9: Actualizado para soportar los operadores merge (`|`) y update (`|=`) de PEP 584 [https://peps.python.org/pep-0584/]'s.

os.environb

Versión de bytes de `environ`: un objeto mapping donde tanto las claves como los valores son objetos `bytes` que representan el entorno del proceso. `environ` y `environb` están sincronizados (modificando **`environb`** actualiza **`environ`**, y viceversa).

`environb` está disponible sólo si **`supports_bytes_environ`** está establecido en `True`.

Added in version 3.2.

Distinto en la versión 3.9: Actualizado para soportar los operadores merge (|) y update (|=) de **PEP 584** [https://peps.python.org/pep-0584/]'s.

os.reload_environ()

The **`os.environ`** and **`os.environb`** mappings are a cache of environment variables at the time that Python started. As such, changes to the current process environment are not reflected if made outside Python, or by **`os.putenv()`** or **`os.unsetenv()`**. Use **`os.reload_environ()`** to update **`os.environ`** and **`os.environb`** with any such changes to the current process environment.

Advertencia

This function is not thread-safe. Calling it while the environment is being modified in another thread is an undefined behavior. Reading from **`os.environ`** or **`os.environb`**, or calling **`os.getenv()`** while reloading, may return an empty result.

Added in version 3.14.

os.chdir(*path*)
os.fchdir(*fd*)
os.getcwd()

Estas funciones están detalladas en Archivos y directorios.

os.fsencode(*filename*)

Codifique path-like *filename* en filesystem encoding and error handler; retorna `bytes` sin cambios.

`fsdecode()` es la función inversa.

Added in version 3.2.

Distinto en la versión 3.6: Soporte agregado para aceptar objetos que implementan una interfaz `os.PathLike`.

os.fsdecode(*filename*)

Decodifica el path-like *filename* del filesystem encoding and error handler; retorna `str` sin cambios.

`fsencode()` es la función inversa.

Added in version 3.2.

Distinto en la versión 3.6: Soporte agregado para aceptar objetos que implementan una interfaz `os.PathLike`.

os.fspath(*path*)

Retorna la representación en el sistema de archivos de la ruta.

Si se le pasa `str` o `bytes`, retorna sin alterar. De lo contrario se llama a `__fspath__()` y se retorna su valor siempre que sea un objeto `str` o `bytes`. En los demás casos se lanza una excepción del tipo `TypeError`.

Added in version 3.6.

class os.PathLike

Una clase base abstracta para objetos que representan una ruta del sistema de archivos, i.e. `pathlib.PurePath`.

Added in version 3.6.

abstractmethod __fspath__()

Retorna la representación de la ruta del sistema de archivos del objeto.

Este método sólo retornará objetos `str` or `bytes`, preferentemente **`str`**.

os.getenv(*key*, *default=None*)

Devuelve el valor de la variable de entorno *key* como una cadena si existe, o *default* si no existe. *key* es una cadena. Tenga en cuenta que dado que `getenv()` usa `os.environ`, la asignación de **`getenv()`** también se captura de manera similar en la importación, y es posible que la función no refleje futuros cambios en el entorno.

En Unix, claves y valores se decodifican con la función `sys.getfilesystemencoding()` y con el controlador de errores `'surrogateescape'`. Usar `os.getenvb()` si se quiere usar una codificación diferente.

Availability: Unix, Windows.

os.getenvb(*key*, *default=None*)

Devuelve el valor de la variable de entorno *key* como bytes si existe, o *default* si no existe. *key* debe ser bytes. Tenga en cuenta que dado que `getenvb()` usa `os.environb`, la asignación de **`getenvb()`** también se captura de manera similar en la importación, y es posible que la función no refleje futuros cambios en el entorno.

`getenvb()` está disponible sólo si `supports_bytes_environ` está establecido en `True`.

Availability: Unix.

Added in version 3.2.

os.get_exec_path(*env=None*)

Retorna una lista de directorios en la que se buscará un ejecutable, similar a una *shell*, cuando se ejecuta un proceso. *env*, cuando se especifica, tienen que ser un diccionario de variables de entorno donde buscar el *PATH*. Por defecto cuando *env* es `None`, se usa `environ`.

Added in version 3.2.

os.getegid()

Retorna el *id* del grupo (*gid*) efectivo correspondiente al proceso que se está ejecutando. Esto corresponde al bit de *«set id»* en el archivo que se está ejecutando en el proceso actual.

Availability: Unix, not WASI.

os.geteuid()

Retorna el id el usuario correspondiente al proceso que se está ejecutando actualmente.

Availability: Unix, not WASI.

os.getgid()

Retorna el id del grupo real correspondiente al proceso que se está ejecutando actualmente.

Availability: Unix.

La función es un «stub» de WASI, consulte Plataformas WebAssembly para obtener más información.

os.getgrouplist(*user*, *group*, */*)

Devuelve la lista de ID de grupo a los que pertenece *user*. Si *group* no está en la lista, se incluye; normalmente, *group* se especifica como el campo de ID de grupo del registro de contraseña para *user*, porque de lo contrario, esa ID de grupo podría omitirse.

Availability: Unix, not WASI.

Added in version 3.3.

os.getgroups()

Retorna la lista de *ids* de grupos secundarios asociados con el proceso actual.

Availability: Unix, not WASI.

Nota

En macOS, el comportamiento de `getgroups()` difiere un poco del de otras plataformas Unix. Si el intérprete de Python se creó con un objetivo de implementación `10.5` o anterior, `getgroups()` devuelve la lista de identificadores de grupo efectivos asociados con el proceso de usuario actual; esta lista está limitada a una cantidad de entradas definida por el sistema, normalmente 16, y puede modificarse mediante llamadas a `setgroups()` si se cuenta con los privilegios adecuados. Si se creó con un objetivo de implementación mayor que `10.5`, `getgroups()` devuelve la lista de acceso de grupo actual para el usuario asociado con el identificador de usuario efectivo del proceso; la lista de acceso de grupo puede cambiar durante la vida útil del proceso, no se ve afectada por las llamadas a `setgroups()` y su longitud no está limitada a 16. El valor del objetivo de implementación, **MACOSX_DEPLOYMENT_TARGET**, se puede obtener con `sysconfig.get_config_var()`.

os.getlogin()

Retorna el nombre del usuario que inició sesión en el terminal que controla el proceso. Para la mayoría de los casos, es más útil usar `getpass.getuser()` ya que este último verifica las variables de entorno **LOGNAME** o **USERNAME** para averiguar quién es el usuario y recurre a `pwd.getpwuid(os.getuid())[0]` para obtener el nombre de inicio de sesión del ID de usuario real actual.

Availability: Unix, Windows, not WASI.

os.getpgid(*pid*)

Retorna el id del grupo de procesos del proceso con la identificación del proceso *pid*. Si *pid* es 0, se retorna la identificación del grupo de proceso del proceso actual.

Availability: Unix, not WASI.

os.getpgrp()

Retorna el *id* del grupo de proceso actual.

Availability: Unix, not WASI.

os.getpid()

Retorna el *id* del proceso actual.

La función es un «stub» de WASI, consulte Plataformas WebAssembly para obtener más información.

os.getppid()

Retorna el *id* del proceso del padre. Cuando el proceso padre ha terminado, en Unix la identificación que retorna es la del proceso *init* (1), en Windows sigue siendo la misma identificación, que ya puede ser reutilizada por otro proceso.

Availability: Unix, Windows, not WASI.

Distinto en la versión 3.2: Se agregó soporte para Windows.

os.getpriority(*which*, *who*)

Obtenga la prioridad del programa. El valor *which* es uno de **`PRIO_PROCESS`**, **`PRIO_PGRP`**, o **`PRIO_USER`**, y *who* se interpreta en relación a *which* (un identificador de proceso para **`PRIO_PROCESS`**, un identificador de grupo de proceso para **`PRIO_PGRP`**, y un ID de usuario para **`PRIO_USER`**). Un valor cero para *who* denota (respectivamente) el proceso llamado, el grupo de proceso del proceso llamado o el ID de usuario real del proceso llamado.

Availability: Unix, not WASI.

Added in version 3.3.

os.PRIO_PROCESS
os.PRIO_PGRP

os.PRIO_USER

Parámetros para las funciones `getpriority()` y `setpriority()`.

Availability: Unix, not WASI.

Added in version 3.3.

os.PRIO_DARWIN_THREAD
os.PRIO_DARWIN_PROCESS
os.PRIO_DARWIN_BG
os.PRIO_DARWIN_NONUI

Parámetros para las funciones `getpriority()` y `setpriority()`.

Availability: macOS

Added in version 3.12.

os.getresuid()

Retorna una tupla (ruid, euid, suid) que denota los ID de usuario reales, efectivos y guardados del proceso actual.

Availability: Unix, not WASI.

Added in version 3.2.

os.getresgid()

Retorna una tupla (rgid, egid, sgid) que denota los ID de grupo reales, efectivos y guardados del proceso actual.

Availability: Unix, not WASI.

Added in version 3.2.

os.getuid()

Retorna el *id* del usuario real del proceso actual.

Availability: Unix.

La función es un «stub» de WASI, consulte Plataformas WebAssembly para obtener más información.

os.initgroups(*username*, *gid*, */*)

Call the system `initgroups()` to initialize the group access list with all of the groups of which the specified username is a member, plus the specified group id.

Availability: Unix, not WASI, not Android.

Added in version 3.2.

os.putenv(*key*, *value*, */*)

Establece la variable de entorno llamada *key* con el valor de la cadena *value*. Dichos cambios en el entorno impactan a los subprocesos iniciados con **`os.system()`**, **`popen()`** o **`fork()`** y **`execv()`**.

Las asignaciones a elementos en **`os.environ`** se traducen automáticamente en llamadas correspondientes a **`putenv()`**; sin embargo, las llamadas a **`putenv()`** no actualizan **`os.environ`**, por lo que en realidad es preferible asignar elementos de **`os.environ`**. Esto también se aplica a **`getenv()`** y **`getenvb()`**, que utilizan respectivamente **`os.environ`** y **`os.environb`** en sus implementaciones.

See also the **`os.reload_environ()`** function.

Nota

En algunas plataformas, incluidas FreeBSD y macOS, la configuración de `environ` puede provocar pérdidas de memoria. Consulta la documentación del sistema para obtener información sobre **`putenv()`**.

Lanza un evento de auditoría `os.putenv` con argumentos `key`, `value`.

Distinto en la versión 3.9: Esta función actualmente siempre esta disponible.

os.setegid(*egid*, */*)

Establece el *id* de grupo efectivo del proceso actual.

Availability: Unix, not WASI, not Android.

os.seteuid(*euid*, */*)

Establece el *id* de usuario efectivo del proceso actual.

Availability: Unix, not WASI, not Android.

os.setgid(*gid*, */*)

Establece el *id* de grupo del proceso actual.

Availability: Unix, not WASI, not Android.

os.setgroups(*groups*, */*)

Establece la lista de *ids* de grupos secundarios asociados con el proceso actual en *groups*. *groups* debe ser una secuencia y cada elemento debe ser un número entero que identifique un grupo. Esta operación generalmente está disponible sólo para el superusuario.

Availability: Unix, not WASI.

Nota

En macOS, la longitud de *groups* no puede exceder el número máximo definido por el sistema de ID de grupo efectivos, por lo general 16. Consulte la documentación de **`getgroups()`** para ver los casos en los que puede no retornar la misma lista de grupos establecida llamando a setgroups().

os.setns(*fd*, *nstype=0*)

Reasociar el hilo actual con un espacio de nombres de Linux. Consulta las páginas del manual *setns(2)* *[https://manpages.debian.org/setns(2)]* y *namespaces(7)* *[https://*

manpages.debian.org/namespaces(7)] para obtener más detalles.

Si *fd* hace referencia a un enlace `/proc/pid/ns/`, `setns()` reasocia el hilo que llama con el espacio de nombres asociado con ese enlace, y *nstype* puede configurarse en uno de los CLONE_NEW* constants para imponer restricciones a la operación (`0` significa sin restricciones).

Desde Linux 5.8, *fd* puede hacer referencia a un descriptor de archivo PID obtenido de **`pidfd_open()`**. En este caso, `setns()` reasocia el subproceso que realiza la llamada en uno o más de los mismos espacios de nombres que el subproceso al que hace referencia *fd*. Esto está sujeto a las restricciones impuestas por *nstype*, que es una máscara de bits que combina uno o más de los CLONE_NEW* constants, por ejemplo, `setns(fd, os.CLONE_NEWUTS | os.CLONE_NEWPID)`. Las pertenencias del autor de la llamada a espacios de nombres no especificados no se modifican.

fd puede ser cualquier objeto con un método **`fileno()`** o un descriptor de archivo sin formato.

Este ejemplo reasocia el hilo con el espacio de nombres de red del proceso `init`:

```
fd = os.open("/proc/1/ns/net", os.O_RDONLY)
os.setns(fd, os.CLONE_NEWNET)
os.close(fd)
```

Availability: Linux >= 3.0 with glibc >= 2.14.

Added in version 3.12.

Ver también

La función **`unshare()`**.

os.setpgrp()

Llama a la llamada del sistema **`setpgrp()`** o `setpgrp(0, 0)` según la versión implementada (si hay alguna). Consulte el manual de Unix para conocer la semántica.

Availability: Unix, not WASI.

os.setpgid(*pid*, *pgrp*, */*)

Llama a la llamada del sistema **`setpgid()`** para establecer el ID del grupo de procesos del proceso con el ID *pid* en el grupo de procesos con el ID *pgrp*. Consulte el manual de Unix para conocer la semántica.

Availability: Unix, not WASI.

os.setpriority(*which*, *who*, *priority*)

Establecer la prioridad del programa. El valor *which* es uno de **`PRIO_PROCESS`**, **`PRIO_PGRP`**, o **`PRIO_USER`**, y *who* se interpreta en relación con *which* (un identificador de proceso para **`PRIO_PROCESS`**, un identificador de grupo de proceso para **`PRIO_PGRP`**, y un ID de usuario para **`PRIO_USER`**). Un valor cero para *who* denota (respectivamente) el proceso llamado, el grupo de procesos del proceso llamado o el ID del usuario real del proceso llamado. *priority* es un valor en el rango de -20 a 19. La prioridad predeterminada es 0; las prioridades más bajas causan una programación más favorable.

Availability: Unix, not WASI.

Added in version 3.3.

os.setregid(*rgid*, *egid*, */*)

Establece los *ids* de grupos reales y efectivos del proceso actual.

Availability: Unix, not WASI, not Android.

os.setresgid(*rgid*, *egid*, *sgid*, */*)

Establece los *ids* de grupo reales, efectivos y guardados del proceso actual.

Availability: Unix, not WASI, not Android.

Added in version 3.2.

os.setresuid(*ruid, euid, suid, /*)

Establece los *ids* de usuario reales, efectivos y guardados del proceso actual.

Availability: Unix, not WASI, not Android.

Added in version 3.2.

os.setreuid(*ruid, euid, /*)

Establece los *ids* de usuario reales y efectivos del proceso actual.

Availability: Unix, not WASI, not Android.

os.getsid(*pid, /*)

Llamar a la llamada del sistema **`getsid()`**. Consultar el manual de Unix para conocer la semántica.

Availability: Unix, not WASI.

os.setsid()

Llamar a la llamada del sistema **`setsid()`**. Consultar el manual de Unix para conocer la semántica.

Availability: Unix, not WASI.

os.setuid(*uid, /*)

Establece *id* del usuario del proceso actual.

Availability: Unix, not WASI, not Android.

os.strerror(*code, /*)

Devuelve el mensaje de error correspondiente al código de error en *code*. En las plataformas donde **`strerror()`** devuelve `NULL` cuando se le asigna un número de error desconocido, se genera **`ValueError`**.

os.supports_bytes_environ

`True` si el tipo de entorno nativo del sistema operativo es bytes (por ejemplo, `False` en Windows).

Added in version 3.2.

os.umask(*mask*, */*)

Establece la *umask* numérica actual y retorna la *umask* anterior.

La función es un «stub» de WASI, consulte Plataformas WebAssembly para obtener más información.

os.uname()

Retorna información que identifica el sistema operativo actual. El valor retornado es un objeto con cinco atributos:

- **`sysname`** - nombre del sistema operativo
- **`nodename`** - nombre de la máquina en la red (definida por la implementación)
- **`release`** - *release* del sistema operativo
- **`version`** - versión del sistema operativo
- **`machine`** - identificador de hardware

Por compatibilidad con versiones anteriores, este objeto también es iterable, se comporta como una tupla que contiene **`sysname`**, **`nodename`**, **`release`**, **`version`**, y **`machine`** en ese orden.

Algunos sistemas se truncan **`nodename`** a 8 caracteres o al componente principal; una mejor manera de obtener el nombre de host es usar **`socket.gethostname()`** o incluso `socket.gethostbyaddr(socket.gethostname())`.

En macOS, iOS y Android, esto devuelve el nombre y la versión de *kernel* (es decir, `'Darwin'` en macOS e iOS; `'Linux'` en Android). **`platform.uname()`** se puede utilizar para obtener el nombre y la versión del sistema operativo que ven los usuarios en iOS y Android.

Availability: Unix.

Distinto en la versión 3.3: El tipo de objeto retornado cambió de una tupla a un objeto tipo tupla con atributos con nombre.

os.unsetenv(*key*, */*)

Desestablece (elimine) la variable de entorno llamada *key*. Dichos cambios en el entorno afectan a los subprocesos iniciados con **`os.system()`**, **`popen()`** o **`fork()`** y **`execv()`**.

La eliminación de elementos en **`os.environ`** se traduce automáticamente en una llamada correspondiente a **`unsetenv()`**; sin embargo, las llamadas a **`unsetenv()`** no actualizan **`os.environ`**, por lo que en realidad es preferible eliminar elementos de **`os.environ`**.

See also the **`os.reload_environ()`** function.

Lanza un evento de auditoría `os.unsetenv` con argumento `key`.

Distinto en la versión 3.9: Estas funciones se encuentra ahora siempre disponible y también esta disponible en Windows.

os.unshare(*flags*)

Desasociar partes del contexto de ejecución del proceso y moverlas a un nuevo espacio de nombres creado. Consulte la página del manual *unshare(2)* *[https://manpages.debian.org/unshare(2)]* para obtener más detalles. El argumento *flags* es una máscara de bits que combina cero o más de los CLONE_* constants y que especifica qué partes del contexto de ejecución deben dejar de compartirse de sus asociaciones existentes y moverse a un nuevo espacio de nombres. Si el argumento *flags* es `0`, no se realizan cambios en el contexto de ejecución del proceso que realiza la llamada.

Availability: Linux >= 2.6.16.

Added in version 3.12.

Ver también

La función `setns()`.

Indicadores de la función `unshare()`, si la implementación los admite. Consulte *unshare(2)* *[https://manpages.debian.org/unshare(2)]* en el manual de Linux para conocer su efecto exacto y su disponibilidad.

os.CLONE_FILES
os.CLONE_FS
os.CLONE_NEWCGROUP
os.CLONE_NEWIPC
os.CLONE_NEWNET
os.CLONE_NEWNS
os.CLONE_NEWPID
os.CLONE_NEWTIME
os.CLONE_NEWUSER
os.CLONE_NEWUTS
os.CLONE_SIGHAND
os.CLONE_SYSVSEM
os.CLONE_THREAD
os.CLONE_VM

Creación de objetos de tipo archivo

Estas funciones crean nuevos objetos de archivo. (Consulte también `open()` para abrir los descriptores de archivos).

os.fdopen(*fd*, **args*, ***kwargs*)

Retorna un objeto de archivo abierto conectado al descriptor de archivo *fd*. Este es un alias de la función incorporada `open()` y acepta los mismos argumentos. La única diferencia es que el primer argumento de `fdopen()` siempre debe ser un número entero.

Operaciones de descriptores de archivos

Estas funciones operan en flujos de E/S a los que se hace referencia mediante descriptores de archivo.

Los descriptores de archivo son enteros pequeños que corresponden a un archivo que ha sido abierto por el proceso actual. Por ejemplo, la entrada estándar suele ser el descriptor de archivo 0, la salida estándar es el 1 y el error estándar es el 2. A los archivos abiertos por un proceso se les asignará 3, 4, 5, y así sucesivamente. El nombre «descriptor de archivo» es ligeramente engañoso; en las plataformas Unix, los descriptores de archivo también hacen referencia a *sockets* y tuberías.

El método `fileno()` se puede utilizar para obtener el descriptor de archivo asociado con un file object cuando sea necesario. Tenga en cuenta que el uso del descriptor de archivo directamente omitirá los métodos de objeto de archivo, ignorando aspectos como el almacenamiento interno interno de datos.

os.close(*fd*)

Cierra el descriptor de archivo *fd*.

Nota

Esta función está diseñada para E/S de bajo nivel y debe aplicarse a un descriptor de archivo tal como lo retorna `os.open()` o `pipe()`. Para cerrar un «objeto de archivo» retornado por la función incorporada `open()` o por `popen()` o `fdopen()`, use el método `close()`.

os.closerange(*fd_low*, *fd_high*, */*)

Cierra todos los descriptores de archivo desde *fd_low* (inclusive) hasta *fd_high* (exclusivo), ignorando los errores. Equivalente a (pero mucho más rápido que):

```
for fd in range(fd_low, fd_high):
    try:
        os.close(fd)
    except OSError:
        pass
```

os.copy_file_range(*src*, *dst*, *count*, *offset_src=None*, *offset_dst=None*)

Copiar bytes *count* desde el descriptor de archivo *src*, comenzando desde el desplazamiento *offset_src*, al descriptor de archivo *dst*, comenzando desde el desplazamiento *offset_dst*. Si *offset_src* es `None`, entonces *src* se lee desde la posición actual; respectivamente para *offset_dst*.

En un kernel de Linux anterior a 5.3, los archivos a los que apuntan *src* y *dst* deben residir en el mismo sistema de archivos; de lo contrario, se genera un `OSError` con `errno` establecido en `errno.EXDEV`.

Esta copia se realiza sin el costo adicional de transferir datos desde el núcleo al espacio de usuario y luego de vuelta al núcleo. Además, algunos sistemas de archivos podrían implementar optimizaciones adicionales, como el uso de enlaces de referencia (es decir, dos o más inodos que comparten punteros a los mismos bloques de disco de copia en escritura; los sistemas de archivos compatibles incluyen btrfs y XFS) y copia del lado del servidor (en el caso de NFS).

La función copia bytes entre dos descriptores de archivo. Las opciones de texto, como la codificación y el final de línea, se ignoran.

El valor de retorno es la cantidad de bytes copiados. Esto podría ser menor que la cantidad solicitada.

Nota

En Linux, no se debe utilizar `os.copy_file_range()` para copiar un rango de un pseudoarchivo desde un sistema de archivos especial como procfs y sysfs. Siempre no copiará ningún byte y devolverá 0 como si el archivo estuviera vacío debido a un problema conocido del kernel de Linux.

Availability: Linux >= 4.5 with glibc >= 2.27.

Added in version 3.8.

os.device_encoding(*fd*)

Retorna una cadena que describe la codificación del dispositivo asociado con *fd* si está conectado a una terminal; sino retorna **`None`**.

En Unix, si el Modo Python UTF-8 está habilitado, retorna `'UTF-8'` en lugar de la codificación del dispositivo.

Distinto en la versión 3.10: En Unix, la función ahora implementa el modo Python UTF-8.

os.dup(*fd*, */*)

Retorna un duplicado del descriptor de archivo *fd*. El nuevo descriptor de archivo es no heredable.

En Windows, al duplicar un flujo estándar (0: stdin, 1: stdout, 2: stderr), el nuevo descriptor de archivo es heredable.

Availability: not WASI.

Distinto en la versión 3.4: El nuevo descriptor de archivo ahora es no heredable.

os.dup2(*fd*, *fd2*, *inheritable=True*)

Duplicar el descriptor de archivo *fd* a *fd2*, cerrando el anterior si es necesario. Retorna *fd2*. El nuevo descriptor de archivo es heredable por defecto o no heredable si *inheritable* es `False`.

Availability: not WASI.

Distinto en la versión 3.4: Agrega el parámetro opcional *inheritable*.

Distinto en la versión 3.7: Retorna *fd2* en caso de éxito. Anteriormente se retornaba siempre `None`.

os.fchmod(*fd*, *mode*)

Cambia el modo del archivo dado por *fd* al modo numérico *mode*. Consulte los documentos para **`chmod()`** para conocer los posibles valores de *mode*. A partir de Python 3.3, esto es equivalente a `os.chmod(fd, mode)`.

Lanza un evento de auditoría `os.chmod` con argumentos `path`, `mode`, `dir_fd`.

Availability: Unix, Windows.

La función está limitada en WASI, consulte Plataformas WebAssembly para obtener más información.

Distinto en la versión 3.13: Se agregó soporte en Windows.

os.fchown(*fd*, *uid*, *gid*)

Cambia el propietario y el *id* del grupo del archivo proporcionado por *fd* a los numéricos dados por *uid* y *gid*. Para dejar uno de los identificadores sin cambios, configúrelo en -1. Ver **`chown()`**. A partir de Python 3.3, esto es equivalente a `os.chown(fd, uid, gid)`.

Lanza un evento de auditoría `os.chown` con argumentos `path`, `uid`, `gid`, `dir_fd`.

Availability: Unix.

La función está limitada en WASI, consulte Plataformas WebAssembly para obtener más información.

os.fdatasync(*fd*)

Fuerza la escritura del archivo con el descriptor de archivo *fd* en el disco. No fuerza la actualización de metadatos.

Availability: Unix.

Nota

Esta función no está disponible en MacOS.

os.fpathconf(*fd*, *name*, */*)

Retorna la información de configuración del sistema relevante para un archivo abierto. *name* especifica el valor de configuración para recuperar; puede ser una cadena que es el nombre de un valor de sistema definido; estos nombres se especifican en varios estándares (POSIX.1, Unix 95, Unix 98 y otros). Algunas plataformas también definen nombres adicionales. Los nombres conocidos por el sistema operativo anfitrión se dan en el diccionario `pathconf_names`. Para las variables de configuración no incluidas en esa asignación, también se acepta pasar un número entero para *name*.

Si *name* es una cadena y no se conoce, se lanza un **`ValueError`**. Si el sistema anfitrión no admite un valor específico para *name*, incluso si está incluido en `pathconf_names`, se genera un **`OSError`** con **`errno.EINVAL`** para el número de error.

A partir de Python 3.3, esto es equivalente a `os.pathconf(fd, name)`.

Availability: Unix.

os.fstat(*fd*)

Obtiene el estado del descriptor de archivo *fd*. Retorna un objeto **`stat_result`**.

A partir de Python 3.3, esto es equivalente a `os.stat(fd)`.

Ver también

La función **`stat()`**.

os.fstatvfs(*fd*, */*)

Retorna información sobre el sistema de archivos que contiene el archivo asociado con el descriptor de archivo *fd*, como **`statvfs()`**. A partir de Python 3.3, esto es equivalente a `os.statvfs(fd)`.

Availability: Unix.

os.fsync(*fd*)

Fuerza la escritura de un archivo con el descriptor de archivo *fd* en el disco. En Unix, esto llama a la función **`fsync()`** nativa; en Windows, a la función **`_commit()`** de MS.

Si está comenzando con un Python almacenado en búfer file object *f*, primero haga `f.flush()`, y luego haga `os.fsync(f.fileno())`, para garantizar que todas las memorias intermedias internas asociadas con *f* se escriban en disco.

Availability: Unix, Windows.

os.ftruncate(*fd*, *length*, */*)

Trunca el archivo correspondiente al descriptor de archivo *fd*, para que tenga como máximo *length* bytes de tamaño. A partir de Python 3.3, esto es equivalente a `os.truncate(fd, length)`.

Lanza un evento de auditoría `os.truncate` con argumentos `fd`, `length`.

Availability: Unix, Windows.

Distinto en la versión 3.5: Se agregó soporte para Windows

os.get_blocking(*fd*, */*)

Obtiene el modo de bloqueo del descriptor de archivo: `False` si se establece el indicador **`O_NONBLOCK`**, `True` si el indicador se borra.

Consulte también **`set_blocking()`** y **`socket.socket.setblocking()`**.

Availability: Unix, Windows.

La función está limitada en WASI, consulte Plataformas WebAssembly para obtener más información.

En Windows, esta función está limitada a las tuberías.

Added in version 3.5.

Distinto en la versión 3.12: Se agregó soporte para pipes en Windows.

os.grantpt(*fd*, */*)

Otorgar acceso al dispositivo pseudoterminal esclavo asociado con el dispositivo pseudoterminal maestro al que hace referencia el descriptor de archivo *fd*. El descriptor de archivo *fd* no se cierra en caso de falla.

Llama a la función de la biblioteca estándar C **`grantpt()`**.

Availability: Unix, not WASI.

Added in version 3.13.

os.isatty(*fd*, */*)

Retorna `True` si el descriptor de archivo *fd* está abierto y conectado a un dispositivo tipo tty, de lo contrario, `False`.

os.lockf(*fd*, *cmd*, *len*, */*)

Aplique, pruebe o elimine un bloqueo POSIX en un descriptor de archivo abierto. *fd* es un descriptor de archivo abierto. *cmd* especifica el comando a usar - uno de **`F_LOCK`**, **`F_TLOCK`**, **`F_ULOCK`** o **`F_TEST`**. *len* especifica la sección del archivo a bloquear.

Lanza un evento de auditoría `os.lockf` con argumentos `fd`, `cmd`, `len`.

Availability: Unix.

Added in version 3.3.

os.F_LOCK
os.F_TLOCK
os.F_ULOCK
os.F_TEST

Indicadores que especifican qué acción tomará **`lockf()`**.

Availability: Unix.

Added in version 3.3.

os.login_tty(*fd*, */*)

Prepare el tty del cual fd es un descriptor de archivo para una nueva sesión de inicio de sesión. Convierta el proceso de convocatoria en un líder de sesión; hacer que el tty sea el tty controlador, el stdin, el stdout y el stderr del proceso de llamada; cerrar fd.

Availability: Unix, not WASI.

Added in version 3.11.

os.lseek(*fd*, *pos*, *whence*, */*)

Establece la posición actual del descriptor de archivo *fd* en la posición *pos*, modificada por *whence*, y devuelve la nueva posición en bytes con respecto al inicio del archivo. Los valores válidos para *whence* son:

- **`SEEK_SET`** o `0`: establece *pos* en relación con el comienzo del archivo
- **`SEEK_CUR`** o `1`: establece *pos* en relación con la posición actual del archivo
- **`SEEK_END`** o `2`: establece *pos* en relación con el final del archivo
- **`SEEK_HOLE`**: establece *pos* en la siguiente ubicación de datos, relativa a *pos*

- `SEEK_DATA`: establece *pos* en el siguiente agujero de datos, relativo a *pos*

Distinto en la versión 3.3: Añadir soporte para `SEEK_HOLE` y `SEEK_DATA`.

os.SEEK_SET
os.SEEK_CUR
os.SEEK_END

Parámetros de la función `lseek()` y del método `seek()` en file-like objects, para desde donde ajustar el indicador de posición del archivo.

`SEEK_SET`
: Ajuste la posición del archivo en relación con el comienzo del archivo.

`SEEK_CUR`
: Ajuste la posición del archivo en relación con la posición del archivo actual.

`SEEK_END`
: Ajuste la posición del archivo en relación con el final del archivo.

Sus valores son 0, 1 y 2, respectivamente.

os.SEEK_HOLE
os.SEEK_DATA

Parámetros de la función `lseek()` y del método `seek()` en file-like objects, para buscar datos de archivos y agujeros en archivos escasamente asignados.

`SEEK_DATA`
: Ajuste el desplazamiento del archivo a la siguiente ubicación que contenga datos, en relación con la posición de búsqueda.

`SEEK_HOLE`
: Ajuste el desplazamiento del archivo a la siguiente

ubicación que contenga un agujero, en relación con la posición de búsqueda. Un agujero se define como una secuencia de ceros.

Nota

Estas operaciones sólo tienen sentido para los sistemas de archivos que las admiten.

Availability: Linux >= 3.1, macOS, Unix

Added in version 3.3.

os.open(*path*, *flags*, *mode=0o777*, ***, *dir_fd=None*)

Abre el archivo *path* y configura varios indicadores según *flags* y su modo según *mode*. Al calcular el modo el valor actual de *umask* se enmascara primero. Retorna el descriptor de archivo para el archivo recién abierto. El nuevo descriptor de archivo es no heredable.

For a description of the flag and mode values, see the C run-time documentation; flag constants (like `O_RDONLY` and `O_WRONLY`) are defined in the `os` module. In particular, on Windows adding `O_BINARY` is needed to open files in binary mode.

Esta función puede admitir rutas relativas a descriptores de directorio con el parámetro *dir_fd*.

Lanza un evento de auditoría `open` con argumentos `path`, `mode`, `flags`.

Distinto en la versión 3.4: El nuevo descriptor de archivo ahora es no heredable.

Nota

This function is intended for low-level I/O. For normal usage, use the built-in function `open()`, which returns a file object with `read()` and `write()` methods. To wrap a file descriptor in a file object, use `fdopen()`.

Distinto en la versión 3.3: Se agregó el parámetro *dir_fd*.

Distinto en la versión 3.5: Si la llamada al sistema se interrumpe y el controlador de señal no genera una excepción, la función vuelve a intentar la llamada del sistema en lugar de generar una excepción `InterruptedError` (ver **PEP 475** [https://peps.python.org/pep-0475/] para la justificación).

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

Las siguientes constantes son opciones para el parámetro *flags* de la función `open()`. Se pueden combinar con el operador OR a nivel de bit `|`. Algunos de ellas no están disponibles en todas las plataformas. Para obtener descripciones de su disponibilidad y uso, consulte la página de manual *open(2)* *[https://manpages.debian.org/open(2)]* en Unix o la MSDN [https://msdn.microsoft.com/en-us/library/z0kc8e3z.aspx] en Windows.

os.O_RDONLY
os.O_WRONLY
os.O_RDWR
os.O_APPEND
os.O_CREAT
os.O_EXCL
os.O_TRUNC

Las constantes anteriores están disponibles en Unix y Windows.

os.O_DSYNC
os.O_RSYNC
os.O_SYNC
os.O_NDELAY
os.O_NONBLOCK
os.O_NOCTTY
os.O_CLOEXEC

Las constantes anteriores sólo están disponibles en Unix.

Distinto en la versión 3.3: Se agregó la constante `O_CLOEXEC`.

os.O_BINARY
os.O_NOINHERIT
os.O_SHORT_LIVED
os.O_TEMPORARY
os.O_RANDOM
os.O_SEQUENTIAL
os.O_TEXT

Las constantes anteriores sólo están disponibles en Windows.

os.O_EVTONLY
os.O_FSYNC
os.O_SYMLINK
os.O_NOFOLLOW_ANY

Las constantes anteriores solo están disponibles en macOS.

Distinto en la versión 3.10: Agregue las constantes **`O_EVTONLY`**, **`O_FSYNC`**, **`O_SYMLINK`** y **`O_NOFOLLOW_ANY`**.

os.O_ASYNC
os.O_DIRECT
os.O_DIRECTORY
os.O_NOFOLLOW
os.O_NOATIME
os.O_PATH
os.O_TMPFILE
os.O_SHLOCK
os.O_EXLOCK

Las constantes anteriores son extensiones y no están presentes si no están definidas por la biblioteca de C.

Distinto en la versión 3.4: Se agrega la constante **`O_PATH`** en los sistemas que lo admiten. Se agrega **`O_TMPFILE`**, sólo disponible en Linux para el Kernel 3.11 o posterior.

os.openpty()

Abre un nuevo par de pseudo-terminal. Retorna un par de descriptores de archivo `(master, slave)`; para pty y tty, respectivamente. Los nuevos descriptores de archivo son no heredable. Para un enfoque (ligeramente) más portátil, use el módulo `pty`.

Availability: Unix, not WASI.

Distinto en la versión 3.4: Los nuevos descriptores de archivo ahora son no heredables.

os.pipe()

Crea una tubería. Retorna un par de descriptores de archivo `(r, w)` que se pueden usar para leer y escribir, respectivamente. El nuevo descriptor de archivo es no heredable.

Availability: Unix, Windows.

Distinto en la versión 3.4: Los nuevos descriptores de archivo ahora son no heredables.

os.pipe2(*flags*, */*)

Crea una tubería con *flags* establecidas atómicamente. *flags* pueden construirse juntando uno o más de estos valores: **`O_NONBLOCK`**, **`O_CLOEXEC`** con el operador OR. Retorna un par de descriptores de archivo `(r, w)` que se pueden usar para leer y escribir, respectivamente.

Availability: Unix, not WASI.

Added in version 3.3.

os.posix_fallocate(*fd*, *offset*, *len*, */*)

Asegura que se asigne suficiente espacio en disco para el archivo especificado por *fd* a partir de *offset* y se extiende por *len* bytes.

Availability: Unix.

Added in version 3.3.

os.posix_fadvise(*fd*, *offset*, *len*, *advice*, */*)

Avisa una intención de acceder a los datos en un patrón específico, permitiendo así que el núcleo haga optimizaciones. El consejo se aplica a la región del archivo especificada por *fd* que comienza en *offset* y se extiende para *len* bytes. *advice* es uno de **`POSIX_FADV_NORMAL`**, **`POSIX_FADV_SEQUENTIAL`**, **`POSIX_FADV_RANDOM`**, **`POSIX_FADV_NOREUSE`**, **`POSIX_FADV_WILLNEED`** o **`POSIX_FADV_DONTNEED`**.

Availability: Unix.

Added in version 3.3.

os.POSIX_FADV_NORMAL
os.POSIX_FADV_SEQUENTIAL
os.POSIX_FADV_RANDOM
os.POSIX_FADV_NOREUSE
os.POSIX_FADV_WILLNEED
os.POSIX_FADV_DONTNEED

Indicadores que se pueden usar en *advice* en **`posix_fadvise()`** que especifican el patrón de acceso que es probable que se use.

Availability: Unix.

Added in version 3.3.

os.pread(*fd*, *n*, *offset*, */*)

Lee como máximo *n* bytes del descriptor de archivo *fd* en una posición de *offset*, sin modificar el desplazamiento (*offset*) del archivo.

Retorna una cadena de bytes que contiene los bytes leídos. Si se alcanza el final del archivo al que hace referencia *fd*, se retorna un objeto de bytes vacío.

Availability: Unix.

Added in version 3.3.

os.posix_openpt(*oflag*, */*)

Abrir y devolver un descriptor de archivo para un dispositivo pseudo-terminal maestro.

Llama a la función de la biblioteca estándar de C **`posix_openpt()`**. El argumento *oflag* se utiliza para establecer indicadores de estado de archivo y modos de acceso a archivos, tal como se especifica en la página del manual de **`posix_openpt()`** de su sistema.

El descriptor de archivo devuelto es non-inheritable. Si el valor **`O_CLOEXEC`** está disponible en el sistema, se agrega a *oflag*.

Availability: Unix, not WASI.

Added in version 3.13.

os.preadv(*fd*, *buffers*, *offset*, *flags=0*, */*)

Lee de un descriptor de archivo *fd* en una posición de *offset* en mutable objetos de tipo bytes *buffers*, dejando el desplazamiento del archivo sin cambios. Transfiere datos a cada búfer hasta que esté lleno y luego pase al siguiente búfer en la secuencia para contener el resto de los datos.

El argumento flags contiene un operador de bit a bit OR de cero o más de las siguientes flags:

- **`RWF_HIPRI`**
- **`RWF_NOWAIT`**

Retorna el número total de bytes realmente leídos que puede ser menor que la capacidad total de todos los objetos.

El sistema operativo puede establecer un límite (**`sysconf()`** valor `'SC_IOV_MAX'`) en el número de búferes que se pueden usar.

Combina la funcionalidad de **`os.readv()`** y **`os.pread()`**.

Availability: Linux >= 2.6.30, FreeBSD >= 6.0, OpenBSD

> = 2.7, AIX > = 7.1.

El uso de banderas requiere Linux > = 4.6.

Added in version 3.7.

os.RWF_NOWAIT

No espere datos que no estén disponibles de inmediato. Si se especifica este indicador, la llamada al sistema regresará instantáneamente si tuviera que leer datos del almacenamiento de respaldo o esperar por un bloqueo.

Si se leyeron correctamente algunos datos, devolverá la cantidad de bytes leídos. Si no se leyeron bytes, devolverá `-1` y establecerá errno en **`errno.EAGAIN`**.

Availability: Linux > = 4.14.

Added in version 3.7.

os.RWF_HIPRI

Alta prioridad de lectura/escritura. Permite que los sistemas de archivos basados en bloques utilicen el sondeo del dispositivo, lo que proporciona una latencia más baja, pero puede usar recursos adicionales.

Actualmente, en Linux, esta función sólo se puede usar en un descriptor de archivo abierto con el indicador **`O_DIRECT`**.

Availability: Linux > = 4.6.

Added in version 3.7.

os.ptsname(*fd*, /)

Devuelve el nombre del dispositivo pseudoterminal esclavo asociado con el dispositivo pseudoterminal maestro al que hace referencia el descriptor de archivo *fd*. El descriptor de archivo *fd* no se cierra en caso de error.

Llama a la función de biblioteca estándar reentrante C `ptsname_r()` si está disponible; de lo contrario, se llama a la función de biblioteca estándar C `ptsname()`, que no se garantiza que sea segura para subprocesos.

Availability: Unix, not WASI.

Added in version 3.13.

os.pwrite(*fd*, *str*, *offset*, */*)

Escribe la cadena de bytes en *str* en el descriptor de archivo *fd* en la posición *offset*, sin modificar el desplazamiento del archivo.

Retorna el número de bytes realmente escritos.

Availability: Unix.

Added in version 3.3.

os.pwritev(*fd*, *buffers*, *offset*, *flags=0*, */*)

Escribe el contenido de *buffers* en el descriptor de archivo *fd* en un desplazamiento *offset*, sin modificar el desplazamiento del archivo. *buffers* debe ser una secuencia de bytes-like objects. Los búferes se procesan en orden de matriz. Se escribe todo el contenido del primer búfer antes de pasar al segundo, y así sucesivamente.

El argumento flags contiene un operador de bit a bit OR de cero o más de las siguientes flags:

- **`RWF_DSYNC`**
- **`RWF_SYNC`**
- **`RWF_APPEND`**

Retorna el número total de bytes realmente escritos.

El sistema operativo puede establecer un límite (**`sysconf()`** valor `'SC_IOV_MAX'`) en el número de búferes que se pueden usar.

Combina la funcionalidad de **`os.writev()`** y `os.pwrite()`.

Availability: Linux >= 2.6.30, FreeBSD >= 6.0, OpenBSD >= 2.7, AIX >= 7.1.

El uso de banderas requiere Linux >= 4.6.

Added in version 3.7.

os.RWF_DSYNC

Proporcione un equivalente por escritura del indicador `O_DSYNC` `os.open()`. Este efecto de bandera se aplica solo al rango de datos escrito por la llamada al sistema.

Availability: Linux >= 4.7.

Added in version 3.7.

os.RWF_SYNC

Proporcione un equivalente por escritura del indicador `O_SYNC` `os.open()`. Este efecto de bandera se aplica solo al rango de datos escrito por la llamada al sistema.

Availability: Linux >= 4.7.

Added in version 3.7.

os.RWF_APPEND

Proporcione un equivalente por escritura del indicador `O_APPEND` `os.open()`. Esta bandera es significativa solo para `os.pwritev()` y su efecto se aplica solo al rango de datos escrito por la llamada al sistema. El argumento *offset* no afecta la operación de escritura; los datos siempre se añaden al final del archivo. Sin embargo, si el argumento *offset* es `-1`, se actualiza el *offset* actual del archivo.

Availability: Linux >= 4.16.

Added in version 3.10.

os.read(*fd*, *n*, */*)

Lee como máximo *n* bytes del descriptor de archivo *fd*.

Retorna una cadena de bytes que contiene los bytes leídos. Si se alcanza el final del archivo al que hace referencia *fd*, se retorna un objeto de bytes vacío.

Nota

This function is intended for low-level I/O and must be applied to a file descriptor as returned by `os.open()` or `pipe()`. To read a «file object» returned by the built-in function `open()` or by `popen()` or `fdopen()`, or `sys.stdin`, use its `read()` or `readline()` methods.

Distinto en la versión 3.5: Si la llamada al sistema se interrumpe y el controlador de señal no genera una excepción, la función vuelve a intentar la llamada del sistema en lugar de generar una excepción `InterruptedError` (ver **PEP 475** [https://peps.python.org/pep-0475/] para la justificación).

os.readinto(*fd*, *buffer*, */*)

Read from a file descriptor *fd* into a mutable buffer object *buffer*.

The *buffer* should be mutable and bytes-like. On success, returns the number of bytes read. Less bytes may be read than the size of the buffer. The underlying system call will be retried when interrupted by a signal, unless the signal handler raises an exception. Other errors will not be retried and an error will be raised.

Returns 0 if *fd* is at end of file or if the provided *buffer* has length 0 (which can be used to check for errors without reading data). Never returns negative.

Nota

This function is intended for low-level I/O and must be applied to a file descriptor as returned by `os.open()` or

`os.pipe()`. To read a «file object» returned by the built-in function **`open()`**, or **`sys.stdin`**, use its member functions, for example **`io.BufferedIOBase.readinto()`**, **`io.BufferedIOBase.read()`**, or **`io.TextIOBase.read()`**

Added in version 3.14.

os.sendfile(*out_fd*, *in_fd*, *offset*, *count*)
os.sendfile(*out_fd*, *in_fd*, *offset*, *count*, *headers=()*, *trailers=()*, *flags=0*)

Copia *count* bytes del descriptor de archivo *in_fd* al descriptor de archivo *out_fd* comenzando en *offset*. Retorna el número de bytes enviados. Cuando se alcanza EOF, retorna `0`.

La primera notación de la función es compatible con todas las plataformas que definen **`sendfile()`**.

En Linux, si *offset* se entrega como `None`, los bytes se leen desde la posición actual de *in_fd* y la posición de *in_fd* se actualiza.

El segundo caso se puede usar en macOS y FreeBSD donde *headers* y *trailers* son secuencias arbitrarias de búferes que se escriben antes y después de que se escriban los datos de *in_fd*. Retorna lo mismo que el primer caso.

En macOS y FreeBSD, un valor de `0` para *count* especifica enviar hasta que se alcanza el final de *in_fd*.

Todas las plataformas admiten sockets como descriptor de archivo *out_fd*, y algunas plataformas también permiten otros tipos (por ejemplo, archivo regular, tuberías).

Las aplicaciones multiplataforma no deben usar los argumentos *headers*, *trailers* y *flags*.

Availability: Unix, not WASI.

Nota

Para un contenedor de alto nivel de `sendfile()`, vea `socket.socket.sendfile()`.

Added in version 3.3.

Distinto en la versión 3.9: Parámetros *out* e *in* han sido renombrados a *out_fd* e *in_fd*.

os.SF_NODISKIO
os.SF_MNOWAIT
os.SF_SYNC

Parámetros para la función `sendfile()`, si la implementación los admite.

Availability: Unix, not WASI.

Added in version 3.3.

os.SF_NOCACHE

Parámetro a la función `sendfile()`, si la implementación lo admite. Los datos no se almacenarán en caché en la memoria virtual y se liberarán después.

Availability: Unix, not WASI.

Added in version 3.11.

os.set_blocking(*fd*, *blocking*, */*)

Establece el modo de bloqueo del descriptor de archivo especificado. Establece la flag `O_NONBLOCK` si se quiere que el bloqueo sea `False`, borre la flag de lo contrario.

Consulte también `get_blocking()` y `socket.socket.setblocking()`.

Availability: Unix, Windows.

La función está limitada en WASI, consulte Plataformas

WebAssembly para obtener más información.

En Windows, esta función está limitada a las tuberías.

Added in version 3.5.

Distinto en la versión 3.12: Se agregó soporte para pipes en Windows.

os.splice(*src*, *dst*, *count*, *offset_src=None*, *offset_dst=None*, *flags=0*)

Transfer *count* bytes from file descriptor *src*, starting from offset *offset_src*, to file descriptor *dst*, starting from offset *offset_dst*.

The splicing behaviour can be modified by specifying a *flags* value. Any of the following variables may used, combined using bitwise OR (the `|` operator):

- If **`SPLICE_F_MOVE`** is specified, the kernel is asked to move pages instead of copying, but pages may still be copied if the kernel cannot move the pages from the pipe.
- If **`SPLICE_F_NONBLOCK`** is specified, the kernel is asked to not block on I/O. This makes the splice pipe operations nonblocking, but splice may nevertheless block because the spliced file descriptors may block.
- If **`SPLICE_F_MORE`** is specified, it hints to the kernel that more data will be coming in a subsequent splice.

At least one of the file descriptors must refer to a pipe. If *offset_src* is `None`, then *src* is read from the current position; respectively for *offset_dst*. The offset associated to the file descriptor that refers to a pipe must be `None`. The files pointed to by *src* and *dst* must reside in the same filesystem, otherwise an **`OSError`** is raised with **`errno`** set to **`errno.EXDEV`**.

Esta copia se realiza sin el costo adicional de transferir datos desde el kernel al espacio del usuario y luego nuevamente al kernel. También, algunos sistemas de archivos podrían implementar optimizaciones adicionales. La copia se realiza como si ambos archivos se abrieran como binarios.

Una vez completado con éxito, retorna el número de bytes empalmados hacia o desde la tubería. Un valor de retorno de 0 significa el final de la entrada. Si *src* se refiere a una tubería, esto significa que no hay datos para transferir y no tendría sentido bloquear porque no hay escritores conectados al extremo de escritura de la tubería.

Ver también

The *splice(2)* *[https://manpages.debian.org/splice(2)]* man page.

Availability: Linux $>=$ 2.6.17 with glibc $>=$ 2.5

Added in version 3.10.

os.SPLICE_F_MOVE
os.SPLICE_F_NONBLOCK
os.SPLICE_F_MORE

Added in version 3.10.

os.readv(*fd*, *buffers*, */*)

Leer desde un descriptor de archivo *fd* en una cantidad de mutable objetos tipo bytes *buffers*. Transfiere datos a cada búfer hasta que esté lleno y luego pase al siguiente búfer en la secuencia para contener el resto de los datos.

Retorna el número total de bytes realmente leídos que puede ser menor que la capacidad total de todos los objetos.

El sistema operativo puede establecer un límite (**`sysconf()`** valor `'SC_IOV_MAX'`) en el número de búferes que se pueden usar.

Availability: Unix.

Added in version 3.3.

os.tcgetpgrp(*fd*, */*)

Retorna el grupo del proceso asociado con la terminal proporcionada por *fd* (un descriptor de archivo abierto como lo retorna `os.open()`).

Availability: Unix, not WASI.

os.tcsetpgrp(*fd*, *pg*, */*)

Establece el grupo del proceso asociado con la terminal dada por *fd* (un descriptor de archivo abierto como lo retorna `os.open()`) a *pg*.

Availability: Unix, not WASI.

os.ttyname(*fd*, */*)

Retorna una cadena que especifica el dispositivo de terminal asociado con el descriptor de archivo *fd*. Si *fd* no está asociado con un dispositivo de terminal, se genera una excepción.

Availability: Unix.

os.unlockpt(*fd*, */*)

Desbloquee el dispositivo pseudoterminal esclavo asociado con el dispositivo pseudoterminal maestro al que hace referencia el descriptor de archivo *fd*. El descriptor de archivo *fd* no se cierra en caso de falla.

Llama a la función de la biblioteca estándar C `unlockpt()`.

Availability: Unix, not WASI.

Added in version 3.13.

os.write(*fd*, *str*, */*)

Escribe la cadena de bytes en *str* en el descriptor de archivo *fd*.

Retorna el número de bytes realmente escritos.

Nota

This function is intended for low-level I/O and must be applied to a file descriptor as returned by `os.open()` or `pipe()`. To write a «file object» returned by the built-in function `open()` or by `popen()` or `fdopen()`, or `sys.stdout` or `sys.stderr`, use its `write()` method.

Distinto en la versión 3.5: Si la llamada al sistema se interrumpe y el controlador de señal no genera una excepción, la función vuelve a intentar la llamada del sistema en lugar de generar una excepción `InterruptedError` (ver **PEP 475** [https://peps.python.org/pep-0475/] para la justificación).

os.writev(*fd*, *buffers*, */*)

Escribe el contenido de *buffers* en el descriptor de archivo *fd*. *buffers* debe ser una secuencia de objetos tipo bytes. Los búferes se procesan en orden secuencial. Se escribe todo el contenido del primer búfer antes de pasar al segundo, y así sucesivamente.

Retorna el número total de bytes realmente escritos.

El sistema operativo puede establecer un límite (`sysconf()` valor `'SC_IOV_MAX'`) en el número de búferes que se pueden usar.

Availability: Unix.

Added in version 3.3.

Consultando las dimensiones de una terminal

Added in version 3.3.

os.get_terminal_size(*fd=STDOUT_FILENO*, */*)

Retorna el tamaño de la ventana de la terminal como `(columns, lines)`, una tupla del tipo `terminal_size`.

El argumento opcional `fd` (por defecto es `STDOUT_FILENO`, o la salida estándar) especifica qué descriptor de archivo debe consultarse.

Si el descriptor de archivo no está conectado a una terminal, se genera un `OSError`.

`shutil.get_terminal_size()` es la función de alto nivel que normalmente debería usarse, `os.get_terminal_size` es la implementación de bajo nivel.

Availability: Unix, Windows.

class os.terminal_size

Una subclase de tupla, que contiene `(columns, lines)` representando el tamaño de la ventana de la terminal.

columns

Ancho de la ventana de la terminal en caracteres.

lines

Alto de la ventana de la terminal en caracteres.

Herencia de los descriptores de archivos

Added in version 3.4.

Un descriptor de archivo tiene un indicador heredable (*inheritable*) que indica si el descriptor de archivo puede ser heredado por procesos secundarios. Desde Python 3.4, los descriptores de archivo creados por Python son no heredables por defecto.

On UNIX, non-inheritable file descriptors are closed in child processes at the execution of a new program, other file descriptors are inherited. Note that non-inheritable file descriptors are still *inherited* by child processes on `os.fork()`.

En Windows, los descriptores de archivo y los identificadores no heredables se cierran en los procesos hijos, a excepción de los flujos estándar (descriptores de archivo 0, 1 y 2: stdin, stdout y stderr), que siempre se heredan. Usando las funciones `spawn*`, todos los identificadores heredables y todos los descriptores de archivos heredables se heredan. Usando el módulo `subprocess`, todos los descriptores de archivo, excepto los flujos estándar, están cerrados, y los identificadores heredables sólo se heredan si el parámetro *close_fds* es `False`.

En las plataformas WebAssembly, el descriptor de archivo no se puede modificar.

os.get_inheritable(*fd*, */*)

Obtiene el indicador heredable (*inheritable*) del descriptor de archivo especificado (un valor booleano).

os.set_inheritable(*fd*, *inheritable*, */*)

Establece el indicador heredable (*inheritable*) del descriptor de archivo especificado.

os.get_handle_inheritable(*handle*, */*)

Obtiene el indicador heredable (*inheritable*) del identificador especificado (un valor booleano).

Availability: Windows.

os.set_handle_inheritable(*handle*, *inheritable*, */*)

Establece el indicador heredable (*inheritable*) del identificador especificado.

Availability: Windows.

Archivos y directorios

En algunas plataformas Unix, muchas de estas funciones admiten una o más de estas características:

- **specifying a file descriptor:** Normally the *path* argument provided to functions in the `os` module must be a string specifying a file path. However, some functions now alternatively accept an open file descriptor for their *path* argument. The function will then operate on the file referred to by the descriptor. For POSIX systems, Python will call the variant of the function prefixed with `f` (e.g. call `fchdir` instead of `chdir`).

Puede verificar si *path* se puede especificar o no como un descriptor de archivo para una función particular en su plataforma usando **`os.supports_fd`**. Si esta funcionalidad no está disponible, su uso lanzará un **`NotImplementedError`**.

Si la función también admite argumentos *dir_fd* o *follow_symlinks*, es un error especificar uno de esos al suministrar *path* como descriptor de archivo.

- **paths relative to directory descriptors:** If *dir_fd* is not `None`, it should be a file descriptor referring to a directory, and the path to operate on should be relative; path will then be relative to that directory. If the path is absolute, *dir_fd* is ignored. For POSIX systems, Python will call the variant of the function with an `at` suffix and possibly prefixed with `f` (e.g. call `faccessat` instead of `access`).

  Puede verificar si *dir_fd* es compatible o no para una función particular en su plataforma usando **`os.supports_dir_fd`**. Si no está disponible, usarlo lanzará un **`NotImplementedError`**.

- **not following symlinks:** If *follow_symlinks* is `False`, and the last element of the path to operate on is a symbolic link, the function will operate on the symbolic link itself rather than the file pointed to by the link. For POSIX systems, Python will call the `l...` variant of the function.

  Puede verificar si *follow_symlinks* es compatible o no para una función particular en su plataforma usando **`os.supports_follow_symlinks`**. Si no está disponible, usarlo lanzará un **`NotImplementedError`**.

os.access(*path*, *mode*, ***, *dir_fd=None*, *effective_ids=False*, *follow_symlinks=True*)

Use el uid/gid real para probar el acceso a *path*. Tenga en cuenta que la mayoría de las operaciones utilizarán el uid/gid efectivo, por lo tanto, esta rutina se puede usar en un entorno suid/sgid para probar si el usuario que invoca tiene el acceso especificado a *path*. *mode* debería ser `F_OK` para probar la existencia de *path*, o puede ser el OR inclusivo de uno o más de `R_OK`, `W_OK`, y `X_OK` para probar los permisos. Retorna `True` si el acceso está permitido, `False` si no. Consulte la página de manual de Unix *access (2) [https://manpages.debian.org/access (2)]* para obtener más información.

Esta función puede admitir la especificación rutas relativas a descriptores de directorio y no seguir los enlaces simbólicos.

Si *effective_ids* es `True`, `access()` realizará sus comprobaciones de acceso utilizando el uid/gid efectivo en lugar del uid/gid real. *effective_ids* puede no ser compatible con su plataforma; puede verificar si está disponible o no usando `os.supports_effective_ids`. Si no está disponible, usarlo lanzará un `NotImplementedError`.

Nota

Usando `access()` para verificar si un usuario está autorizado para, por ejemplo, abrir un archivo antes de hacerlo usando `open()` crea un agujero de seguridad, porque el usuario podría explotar el breve intervalo de tiempo entre verificar y abrir el archivo para manipularlo es preferible utilizar técnicas EAFP. Por ejemplo:

```
if os.access("myfile", os.R_OK):
    with open("myfile") as fp:
        return fp.read()
return "some default data"
```

está mejor escrito como:

```
try:
    fp = open("myfile")
except PermissionError:
    return "some default data"
```

```
else:
    with fp:
        return fp.read()
```

Nota

Las operaciones de E/S pueden fallar incluso cuando `access()` indica que tendrán éxito, particularmente para operaciones en sistemas de archivos de red que pueden tener una semántica de permisos más allá del modelo habitual de bits de permiso POSIX.

Distinto en la versión 3.3: Se agregaron los parámetros *dir_fd*, *effective_ids* y *follow_symlinks*.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.F_OK
os.R_OK
os.W_OK
os.X_OK

Valores para pasar como parámetro *mode* de `access()` para probar la existencia, legibilidad, escritura y ejecutabilidad de *path*, respectivamente.

os.chdir(*path*)

Cambia el directorio de trabajo actual a *path*.

Esta función puede soportar especificando un descriptor de archivo. El descriptor debe hacer referencia a un directorio abierto, no a un archivo abierto.

Esta función puede generar `OSError` y subclases como `FileNotFoundError`, `PermissionError`, y `NotADirectoryError`.

Lanza un evento de auditoría `os.chdir` con argumento `path`.

Distinto en la versión 3.3: Se agregó soporte para especificar *path* como descriptor de archivo en algunas plataformas.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.chflags(*path*, *flags*, ***, *follow_symlinks=True*)

Establece las flags del *path* a las *flags* numéricas. *flags* puede tomar una combinación (OR bit a bit) de los siguientes valores (como se define en el módulo **`stat`**):

- **`stat.UF_NODUMP`**
- **`stat.UF_IMMUTABLE`**
- **`stat.UF_APPEND`**
- **`stat.UF_OPAQUE`**
- **`stat.UF_NOUNLINK`**
- **`stat.UF_COMPRESSED`**
- **`stat.UF_HIDDEN`**
- **`stat.SF_ARCHIVED`**
- **`stat.SF_IMMUTABLE`**
- **`stat.SF_APPEND`**
- **`stat.SF_NOUNLINK`**
- **`stat.SF_SNAPSHOT`**

Esta función puede soportar no seguir enlaces simbólicos.

Lanza un evento de auditoría `os.chflags` con argumentos `path`, `flags`.

Availability: Unix, not WASI.

Distinto en la versión 3.3: Se agregó el argumento *follow_symlinks*.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.chmod(*path*, *mode*, ***, *dir_fd=None*, *follow_symlinks=True*)

Cambia el modo de *path* al modo numérico *mode*. *mode* puede tomar uno de los siguientes valores (como se define en el módulo **`stat`**) o combinaciones OR de bit a bit de ellos:

- **`stat.S_ISUID`**
- **`stat.S_ISGID`**
- **`stat.S_ENFMT`**
- **`stat.S_ISVTX`**
- **`stat.S_IREAD`**
- **`stat.S_IWRITE`**
- **`stat.S_IEXEC`**
- **`stat.S_IRWXU`**
- **`stat.S_IRUSR`**
- **`stat.S_IWUSR`**
- **`stat.S_IXUSR`**
- **`stat.S_IRWXG`**
- **`stat.S_IRGRP`**
- **`stat.S_IWGRP`**
- **`stat.S_IXGRP`**
- **`stat.S_IRWXO`**
- **`stat.S_IROTH`**
- **`stat.S_IWOTH`**
- **`stat.S_IXOTH`**

Esta función puede soportar especificando un descriptor de archivo, rutas relativas a los descriptores de directorio y no seguir enlaces simbólicos.

Nota

Aunque Windows admite **`chmod()`**, solo se puede configurar el indicador de solo lectura del archivo con él (a través de las constantes `stat.S_IWRITE` y `stat.S_IREAD` o un valor entero correspondiente). Se ignoran todos los demás bits. El valor predeterminado de *follow_symlinks* es `False` en Windows.

La función está limitada en WASI, consulte Plataformas WebAssembly para obtener más información.

Lanza un evento de auditoría `os.chmod` con argumentos `path`, `mode`, `dir_fd`.

Distinto en la versión 3.3: Se agregó soporte para especificar *path* como un descriptor de archivo abierto, y los argumentos *dir_fd* y *follow_symlinks*.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

Distinto en la versión 3.13: Se agregó soporte para un descriptor de archivo y el argumento *follow_symlinks* en Windows.

os.chown(*path*, *uid*, *gid*, ***, *dir_fd=None*, *follow_symlinks=True*)

Cambia el propietario y el *id* del grupo de *path* a los numéricos *uid* y *gid*. Para dejar uno de los identificadores sin cambios, configúrelo en -1.

Esta función puede soportar especificando un descriptor de archivo, rutas relativas a los descriptores de directorio y no seguir enlaces simbólicos.

Ver **`shutil.chown()`** para una función de nivel superior que acepta nombres además de identificadores numéricos.

Lanza un evento de auditoría `os.chown` con argumentos `path`, `uid`, `gid`, `dir_fd`.

Availability: Unix.

La función está limitada en WASI, consulte Plataformas WebAssembly para obtener más información.

Distinto en la versión 3.3: Se agregó soporte para especificar *path* como un descriptor de archivo abierto, y los argumentos *dir_fd* y *follow_symlinks*.

Distinto en la versión 3.6: Admite un objeto tipo ruta.

os.chroot(*path*)

Cambia el directorio raíz del proceso actual a *path*.

Availability: Unix, not WASI, not Android.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.fchdir(*fd*)

Cambia el directorio de trabajo actual al directorio representado por el descriptor de archivo *fd*. El descriptor debe hacer referencia a un directorio abierto, no a un archivo abierto. A partir de Python 3.3, esto es equivalente a `os.chdir(fd)`.

Lanza un evento de auditoría `os.chdir` con argumento `path`.

Availability: Unix.

os.getcwd()

Retorna una cadena que representa el directorio de trabajo actual.

os.getcwdb()

Retorna una cadena de bytes que representa el directorio de trabajo actual.

Distinto en la versión 3.8: La función ahora usa la codificación UTF-8 en Windows, en lugar de los códigos ANSI: consulte **PEP 529** [https://peps.python.org/pep-0529/] para ver la justificación. La función ya no está en desuso en Windows.

os.lchflags(*path*, *flags*)

Establece las flags de *path* a las *flags* numéricas, como **`chflags()`**, pero no siga los enlaces simbólicos. A partir de Python 3.3, esto es equivalente a `os.chflags(path, flags, follow_symlinks=False)`.

Lanza un evento de auditoría `os.chflags` con argumentos `path`, `flags`.

Availability: Unix, not WASI.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.lchmod(*path*, *mode*)

Cambia el modo de *path* al *mode* numérico. Si la ruta es un enlace simbólico, esto afecta al enlace simbólico en lugar del objetivo. Consulte los documentos para **`chmod()`** para conocer los posibles valores de *mode*. A partir de Python 3.3, esto es equivalente a `os.chmod(path, mode, follow_symlinks=False)`.

`lchmod()` no es parte de POSIX, pero las implementaciones de Unix pueden tenerlo si se admite el cambio del modo de enlaces simbólicos.

Lanza un evento de auditoría `os.chmod` con argumentos `path`, `mode`, `dir_fd`.

Availability: Unix, Windows, not Linux, FreeBSD $>=$ 1.3, NetBSD $>=$ 1.3, not OpenBSD

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

Distinto en la versión 3.13: Se agregó soporte en Windows.

os.lchown(*path*, *uid*, *gid*)

Cambia el propietario y la identificación del grupo de *path* a los numéricos *uid* y *gid*. Esta función no seguirá enlaces simbólicos. A partir de Python 3.3, esto es equivalente a `os.chown(path, uid, gid, follow_symlinks=False)`.

Lanza un evento de auditoría `os.chown` con argumentos `path`, `uid`, `gid`, `dir_fd`.

Availability: Unix.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.link(*src*, *dst*, ***, *src_dir_fd=None*, *dst_dir_fd=None*, *follow_symlinks=True*)

Cree un enlace rígido que apunte a *src* llamado *dst*.

This function can support specifying *src_dir_fd* and/or *dst_dir_fd* to supply paths relative to directory descriptors, and not following symlinks. The default value of *follow_symlinks* is `False` on Windows.

Lanza un evento de auditoría `os.link` con argumentos `src`, `dst`, `src_dir_fd`, `dst_dir_fd`.

Availability: Unix, Windows.

Distinto en la versión 3.2: Se agregó soporte para Windows.

Distinto en la versión 3.3: Se agregaron los parámetros *src_dir_fd*, *dst_dir_fd* y *follow_symlinks*.

Distinto en la versión 3.6: Acepta un path-like object para *src* y *dst*.

os.listdir(*path='.'*)

Retorna una lista que contiene los nombres de las entradas en el directorio dado por *path*. La lista está en un orden arbitrario y no incluye las entradas especiales `'.'` Y `'..'` incluso si están presentes en el directorio. Si un archivo es removido de o añadido al directorio mientras se llama a esta función, no se especifica si el nombre para el archivo será incluido.

path puede ser un path-like object. Si *path* es de tipo `bytes` (directa o indirectamente a través de la interfaz **`PathLike`**), los nombres de archivo retornados también serán de tipo `bytes`; en todas las demás circunstancias, serán del tipo `str`.

Esta función también puede admitir especificando un descriptor de archivo; el descriptor de archivo debe hacer referencia a un directorio.

Lanza un evento de auditoría `os.listdir` con el argumento `ruta`.

Nota

Para codificar los nombres de archivo `str` en `bytes`, use **`fsencode()`**.

Ver también

La función **`scandir()`** retorna entradas de directorio junto con información de atributos de archivo, lo que proporciona un mejor rendimiento para muchos casos de uso comunes.

Distinto en la versión 3.2: El parámetro *path* se convirtió en opcional.

Distinto en la versión 3.3: Se agregó soporte para especificar *path* como un descriptor de archivo abierto.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.listdrives()

Devuelve una lista que contiene los nombres de las unidades en un sistema Windows.

El nombre de una unidad suele ser `'C:\\'`. No todos los nombres de unidades se asociarán a un volumen y es posible que algunas sean inaccesibles por diversos motivos, incluidos permisos, conectividad de red o medios faltantes. Esta función no prueba el acceso.

Puede generar **`OSError`** si ocurre un error al recopilar los nombres de las unidades.

Genera un auditing event `os.listdrives` sin argumentos.

Availability: Windows

Added in version 3.12.

os.listmounts(*volume*)

Devuelve una lista que contiene los puntos de montaje de un volumen en un sistema Windows.

volume debe representarse como una ruta GUID, como las que devuelve **`os.listvolumes()`**. Los volúmenes pueden montarse en varias ubicaciones o no montarse en ninguna. En este último caso, la lista estará vacía. Esta función no devolverá los puntos de montaje que no estén asociados a un volumen.

Los puntos de montaje devueltos por esta función serán rutas absolutas y pueden ser más largos que el nombre de la unidad.

Genera **`OSError`** si no se reconoce el volumen o si se produce un error al recopilar las rutas.

Genera un auditing event `os.listmounts` con el argumento `volume`.

Availability: Windows

Added in version 3.12.

os.listvolumes()

Devuelve una lista que contiene los volúmenes del sistema.

Los volúmenes se representan normalmente como una ruta GUID similar a `\\?\Volume{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}\`. Normalmente, se puede acceder a los archivos a través de una ruta GUID, si los permisos lo permiten. Sin embargo, los usuarios no suelen estar familiarizados con ellos, por lo que el uso recomendado de esta función es recuperar los puntos de montaje mediante **`os.listmounts()`**.

Puede generar **`OSError`** si ocurre un error al recopilar los volúmenes.

Genera un auditing event `os.listvolumes` sin argumentos.

Availability: Windows

Added in version 3.12.

os.lstat(*path*, ***, *dir_fd=None*)

Realiza el equivalente de una llamada al sistema `lstat()` en la ruta indicada. Similar a `stat()`, pero no sigue enlaces simbólicos. Devuelve un objeto `stat_result`.

En plataformas que no admiten enlaces simbólicos, este es un alias para `stat()`.

A partir de Python 3.3, esto es equivalente a `os.stat(path, dir_fd=dir_fd, follow_symlinks=False)`.

Esta función también puede admitir rutas relativas a descriptores de directorio.

Ver también

La función `stat()`.

Distinto en la versión 3.2: Se agregó soporte para enlaces simbólicos de Windows 6.0 (Vista).

Distinto en la versión 3.3: Se agregó el parámetro *dir_fd*.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

Distinto en la versión 3.8: En Windows, ahora abre puntos de análisis que representan otra ruta (nombres sustitutos), incluidos enlaces simbólicos y uniones de directorio. El sistema operativo resuelve otros tipos de puntos de análisis como `stat()`.

os.mkdir(*path*, *mode=0o777*, ***, *dir_fd=None*)

Cree un directorio llamado *path* con modo numérico *mode*.

Si el directorio ya existe, se genera `FileExistsError`. Si no existe un directorio principal en la ruta, se genera `FileNotFoundError`.

En algunos sistemas, *mode* se ignora. Donde se usa, el valor actual de umask se enmascara primero. Si se establecen bits distintos de los últimos 9 (es decir, los últimos 3 dígitos de la representación octal del *mode*), su significado depende de la plataforma. En algunas plataformas, se ignoran y debe llamar a **`chmod()`** explícitamente para configurarlos.

En Windows, se maneja específicamente un *mode* o `0o700` para aplicar el control de acceso al nuevo directorio de modo que solo el usuario actual y los administradores tengan acceso. Se ignoran otros valores de *mode*.

Esta función también puede admitir rutas relativas a descriptores de directorio.

También es posible crear directorios temporales; vea la función **`tempfile`** del módulo **`tempfile.mkdtemp()`**.

Lanza un evento de auditoría `os.mkdir` con argumentos `ruta`, `modo`, `dir_fd`.

Distinto en la versión 3.3: Se agregó el parámetro *dir_fd*.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

Distinto en la versión 3.13: Windows ahora maneja un *mode* o `0o700`.

os.makedirs(*name*, *mode=0o777*, *exist_ok=False*)

Función de creación de directorio recursiva. Como **`mkdir()`**, pero hace que todos los directorios de nivel intermedio sean necesarios para contener el directorio hoja.

El parámetro *mode* se pasa a **`mkdir()`** para crear el directorio hoja; consulte the mkdir() description para saber cómo se interpreta. Para configurar los bits de permiso de archivo de cualquier directorio principal recién creado, puede configurar umask antes de invocar **`makedirs()`**. Los bits de permiso de archivo de los directorios principales existentes no se modifican.

Si *exist_ok* es `False` (el valor predeterminado), se genera un **`FileExistsError`** si el directorio de destino ya existe.

Nota

`makedirs()` se confundirá si los elementos de ruta a crear incluyen **`pardir`** (por ejemplo, «..» en sistemas UNIX).

Esta función maneja las rutas UNC correctamente.

Lanza un evento de auditoría `os.mkdir` con argumentos `ruta`, `modo`, `dir_fd`.

Distinto en la versión 3.2: Se agregó el parámetro *exist_ok*.

Distinto en la versión 3.4.1: Antes de Python 3.4.1, si *exist_ok* era `True` y el directorio existía, **`makedirs()`** aún generaría un error si *mode* no coincidía con el modo del directorio existente. Como este comportamiento era imposible de implementar de forma segura, se eliminó en Python 3.4.1. Ver bpo-21082 [https://bugs.python.org/issue?@action=redirect&bpo=21082].

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

Distinto en la versión 3.7: El argumento *mode* ya no afecta los bits de permiso de archivo de los directorios de nivel intermedio recién creados.

os.mkfifo(*path*, *mode=0o666*, ***, *dir_fd=None*)

Cree una FIFO (una tubería con nombre) llamada *path* con modo numérico *modo*. El valor actual de umask se enmascara primero del modo.

Esta función también puede admitir rutas relativas a descriptores de directorio.

Los FIFO son tuberías a las que se puede acceder como archivos normales. Los FIFO existen hasta que se eliminan (por ejemplo con **`os.unlink()`**). En general, los FIFO se utilizan como punto de encuentro entre los procesos de tipo «cliente» y «servidor»: el servidor abre el FIFO para leer y el cliente lo abre para escribir. Tenga en cuenta que **`mkfifo()`** no abre el FIFO — solo crea el punto de encuentro.

Availability: Unix, not WASI.

Distinto en la versión 3.3: Se agregó el parámetro *dir_fd*.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.mknod(*path*, *mode=0o600*, *device=0*, ***, *dir_fd=None*)

Cree un nodo del sistema de archivos (archivo, archivo especial del dispositivo o canalización con nombre) llamado *path*. *mode* especifica tanto los permisos para usar como el tipo de nodo que se creará, combinándose (OR bit a bit) con uno de `stat.S_IFREG`, `stat.S_IFCHR`, `stat.S_IFBLK` , y `stat.S_IFIFO` (esas constantes están disponibles en **`stat`**). Para `stat.S_IFCHR` y `stat.S_IFBLK`, *device* define el archivo especial del dispositivo recién creado (probablemente usando **`os.makedev()`**), de lo contrario se ignora.

Esta función también puede admitir rutas relativas a descriptores de directorio.

Availability: Unix, not WASI.

Distinto en la versión 3.3: Se agregó el parámetro *dir_fd*.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.major(*device*, */*)

Extrae el número principal del dispositivo de un número de dispositivo sin procesar (generalmente el campo **`st_dev`** o **`st_rdev`** de **`stat`**).

os.minor(*device*, */*)

Extrae el número menor del dispositivo de un número de dispositivo sin procesar (generalmente el campo **`st_dev`** o **`st_rdev`** de **`stat`**).

os.makedev(*major*, *minor*, */*)

Compone un número de dispositivo sin procesar a partir de los números de dispositivo mayor y menor.

os.pathconf(*path*, *name*)

Retorna información de configuración del sistema relevante para un archivo con nombre. *name* especifica el valor de configuración para recuperar; puede ser una cadena que es el nombre de un valor de sistema definido; Estos nombres se especifican en varios estándares (POSIX.1, Unix 95, Unix 98 y otros). Algunas plataformas también definen nombres adicionales. Los nombres conocidos por el sistema operativo host se dan en el diccionario `pathconf_names`. Para las variables de configuración no incluidas en esa asignación, también se acepta pasar un número entero para *name*.

Si *name* es una cadena y no se conoce, se lanza un **`ValueError`**. Si el sistema anfitrión no admite un valor específico para *name*, incluso si está incluido en `pathconf_names`, se genera un **`OSError`** con **`errno.EINVAL`** para el número de error.

Esta función puede soportar especificando un descriptor de archivo.

Availability: Unix.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.pathconf_names

Nombres de mapeo de diccionario aceptados por **`pathconf()`** y **`fpathconf()`** a los valores enteros definidos para esos nombres por el sistema operativo host. Esto se puede usar para determinar el conjunto de nombres conocidos por el sistema.

Availability: Unix.

os.readlink(*path*, ***, *dir_fd=None*)

Retorna una cadena que representa la ruta a la que apunta el enlace simbólico. El resultado puede ser un nombre de ruta absoluto o relativo; si es relativo, se puede convertir a un nombre de ruta absoluto usando `os.path.join(os.path.dirname(path), result)`.

Si la *path* es un objeto de cadena (directa o indirectamente a través de una interfaz **`PathLike`**), el resultado también será un objeto de cadena y la llamada puede generar un UnicodeDecodeError. Si la *path* es un objeto de bytes (directa o indirectamente), el resultado será un objeto de bytes.

Esta función también puede admitir rutas relativas a descriptores de directorio.

Cuando intente resolver una ruta que puede contener enlaces, use **`realpath()`** para manejar adecuadamente la recurrencia y las diferencias de plataforma.

Availability: Unix, Windows.

Distinto en la versión 3.2: Se agregó soporte para enlaces simbólicos de Windows 6.0 (Vista).

Distinto en la versión 3.3: Se agregó el parámetro *dir_fd*.

Distinto en la versión 3.6: Acepta un path-like object en Unix.

Distinto en la versión 3.8: Acepta un path-like object y un objeto de bytes en Windows.

Se agregó soporte para uniones de directorio y se modificó para retornar la ruta de sustitución (que generalmente incluye el prefijo `\\?\`) En lugar del campo opcional «nombre de impresión» que se retornó anteriormente.

os.remove(*path*, ***, *dir_fd=None*)

Eliminar (borrar) el archivo *path*. Si *path* es un directorio, se genera un error **`OSError`**. Utilice **`rmdir()`** para eliminar directorios. Si el archivo no existe, se genera un error **`FileNotFoundError`**.

Esta función puede admitir rutas relativas a descriptores de directorio.

En Windows, intentar eliminar un archivo que está en uso provoca una excepción; en Unix, la entrada del directorio se elimina pero el almacenamiento asignado al archivo no está disponible hasta que el archivo original ya no esté en uso.

Esta función es semánticamente idéntica a **`unlink()`**.

Lanza un evento de auditoría `os.remove` con argumentos `ruta`, `dir_fd`.

Distinto en la versión 3.3: Se agregó el parámetro *dir_fd*.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.removedirs(*name*)

Eliminar directorios de forma recursiva. Funciona como **`rmdir()`** excepto que, si el directorio hoja se elimina con éxito, **`removeirs()`** intenta eliminar sucesivamente cada directorio principal mencionado en *path* hasta que se genere un error (que se ignora, porque generalmente significa que un directorio padre no está vacío). Por ejemplo, `os.removedirs('foo/bar/baz')` primero eliminará el directorio `'foo/bar/baz'`, y luego eliminará `'foo/bar'` y `'foo'` si están vacíos. Genera **`OSError`** si el directorio hoja no se pudo eliminar con éxito.

Lanza un evento de auditoría `os.remove` con argumentos `ruta`, `dir_fd`.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.rename(*src*, *dst*, ***, *src_dir_fd=None*, *dst_dir_fd=None*)

Cambia el nombre del archivo o directorio *src* a *dst*. Si *dst* existe, la operación fallará con una subclase **`OSError`** en varios casos:

En Windows, si existe *dst*, siempre se genera un **`FileExistsError`**. La operación puede fallar si *src* y *dst* están en sistemas de archivos diferentes. Utilice **`shutil.move()`** para admitir movimientos a un sistema de archivos diferente.

En Unix, si *src* es un archivo y *dst* es un directorio o viceversa, se generará un **`IsADirectoryError`** o un **`NotADirectoryError`** respectivamente. Si ambos son directorios y *dst* está vacío, *dst* se reemplazará silenciosamente. Si *dst* no es un directorio vacío, se genera un **`OSError`**. Si ambos son archivos, *dst* se reemplazará silenciosamente si el usuario tiene permiso. La operación puede fallar en algunos tipos de Unix si *src* y *dst* están en sistemas de archivos diferentes. Si tiene éxito, el cambio de nombre será una operación atómica (este es un requisito POSIX).

Esta función puede admitir la especificación de *src_dir_fd* o *dst_dir_fd* para proporcionar rutas relativas a los descriptores de directorio.

Si desea sobrescribir multiplataforma del destino, use **`replace()`**.

Lanza un evento de auditoría `os.rename` con argumentos `src`, `dst`, `src_dir_fd`, `dst_dir_fd`.

Distinto en la versión 3.3: Se agregaron los parámetros *src_dir_fd* y *dst_dir_fd*.

Distinto en la versión 3.6: Acepta un path-like object para *src* y *dst*.

os.renames(*old*, *new*)

Directorio recursivo o función de cambio de nombre de archivo. Funciona como **`rename()`**, excepto que primero se intenta crear cualquier directorio intermedio necesario para que el nuevo nombre de ruta sea bueno. Después del cambio de nombre, los directorios correspondientes a los segmentos de ruta más a la derecha del nombre anterior se eliminarán usando **`removeirs()`**.

Nota

Esta función puede fallar con la nueva estructura de directorios realizada si carece de los permisos necesarios para eliminar el directorio o archivo hoja.

Lanza un evento de auditoría `os.rename` con argumentos `src`, `dst`, `src_dir_fd`, `dst_dir_fd`.

Distinto en la versión 3.6: Acepta un path-like object para *old* y *new*.

os.replace(*src*, *dst*, ***, *src_dir_fd=None*, *dst_dir_fd=None*)

Cambie el nombre del archivo o directorio *src* a *dst*. Si *dst* no es un directorio vacío, se generará **`OSError`**. Si *dst* existe y es un archivo, se reemplazará de forma silenciosa si el usuario tiene permiso. La operación puede fallar si *src* y *dst* están en sistemas de archivos diferentes. Si tiene éxito, el cambio de nombre será una operación atómica (este es un requisito POSIX).

Esta función puede admitir la especificación de *src_dir_fd* o *dst_dir_fd* para proporcionar rutas relativas a los descriptores de directorio.

Lanza un evento de auditoría `os.rename` con argumentos `src`, `dst`, `src_dir_fd`, `dst_dir_fd`.

Added in version 3.3.

Distinto en la versión 3.6: Acepta un path-like object para *src* y *dst*.

os.rmdir(*path*, ***, *dir_fd=None*)

Suprime (elimina) el directorio *path*. Si el directorio no existe o no está vacío, se genera un **`FileNotFoundError`** o un **`OSError`** respectivamente. Para eliminar árboles de directorios completos, se puede utilizar **`shutil.rmtree()`**.

Esta función puede admitir rutas relativas a descriptores de directorio.

Lanza un evento de auditoría `os.rmdir` con argumentos `ruta`, `dir_fd`.

Distinto en la versión 3.3: Se agregó el parámetro *dir_fd*.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.scandir(*path*='.')

Retorna un iterador de objetos **`os.DirEntry`** correspondientes a las entradas en el directorio dado por *path*. Las entradas se entregan en orden arbitrario, y las entradas especiales **`'.'`** Y **`'..'`** no están incluidas. Si un archivo es removido de o añadido al directorio después de crear el iterador, no se especifica si una entrada para el archivo será incluido.

El uso de **`scandir()`** en lugar de **`listdir()`** puede aumentar significativamente el rendimiento del código que también necesita información de tipo de archivo o atributo de archivo, porque: los objetos **`os.DirEntry`** exponen esta información si el sistema operativo proporciona cuando escanea un directorio. Todos los métodos **`os.DirEntry`** pueden realizar una llamada al sistema, pero **`is_dir()`** y **`is_file()`** generalmente solo requieren una llamada al sistema para enlaces simbólicos; **`os.DirEntry.stat()`** siempre requiere una llamada al sistema en Unix, pero solo requiere una para enlaces simbólicos en Windows.

path puede ser un path-like object. Si *path* es de tipo `bytes` (directa o indirectamente a través de la interfaz **`PathLike`**), el tipo de **`name`** y los atributos **`path`** de cada **`os.DirEntry`** serán `bytes`; en todas las demás circunstancias, serán del tipo `str`.

Esta función también puede admitir especificando un descriptor de archivo; el descriptor de archivo debe hacer referencia a un directorio.

Lanza un evento de auditoría `os.scandir` con argumento `ruta`.

El iterador **`scandir()`** admite el protocolo context manager y tiene el siguiente método:

scandir.close()

Cierre el iterador y libere los recursos adquiridos.

Esto se llama automáticamente cuando el iterador se agota o se recolecta basura, o cuando ocurre un error durante la iteración. Sin embargo, es aconsejable llamarlo explícitamente o utilizar la palabra clave **`with`**.

Added in version 3.6.

El siguiente ejemplo muestra un uso simple de **`scandir()`** para mostrar todos los archivos (excepto los directorios) en la *path* dada que no comienzan con `'.'`. La llamada `entry.is_file()` generalmente no realizará una llamada adicional al sistema:

```
with os.scandir(path) as it:
    for entry in it:
        if not entry.name.startswith('.') and entry
            print(entry.name)
```

Nota

En los sistemas basados en Unix, **`scandir()`** usa las funciones opendir() [https://pubs.opengroup.org/onlinepubs/009695399/functions/opendir.html] y readdir() [https://pubs.opengroup.org/onlinepubs/009695399/functions/readdir_r.html] del sistema. En Windows, utiliza las funciones Win32

FindFirstFileW [https://msdn.microsoft.com/en-us/library/windows/desktop/aa364418(v=vs.85).aspx] y FindNextFileW [https://msdn.microsoft.com/en-us/library/windows/desktop/aa364428(v=vs.85).aspx].

Added in version 3.5.

Distinto en la versión 3.6: Se agregó compatibilidad con el protocolo context manager y el método **`close()`**. Si un iterador **`scandir()`** no se agota ni se cierra explícitamente, se emitirá un **`ResourceWarning`** en su destructor.

La función acepta un path-like object.

Distinto en la versión 3.7: Soporte agregado para descriptores de archivo en Unix.

class os.DirEntry

Objeto generado por **`scandir()`** para exponer la ruta del archivo y otros atributos de archivo de una entrada de directorio.

`scandir()` proporcionará tanta información como sea posible sin hacer llamadas adicionales al sistema. Cuando se realiza una llamada al sistema `stat()` o `lstat()`, el objeto `os.DirEntry` almacenará en caché el resultado.

Las instancias `os.DirEntry` no están destinadas a ser almacenadas en estructuras de datos de larga duración; si sabe que los metadatos del archivo han cambiado o si ha transcurrido mucho tiempo desde la llamada **`scandir()`**, llame a `os.stat(entry.path)` para obtener información actualizada.

Debido a que los métodos `os.DirEntry` pueden hacer llamadas al sistema operativo, también pueden generar **`OSError`**. Si necesita un control muy preciso sobre los errores, puede detectar **`OSError`** cuando llame a uno de los métodos `os.DirEntry` y maneje según corresponda.

Para ser directamente utilizable como path-like object, `os.DirEntry` implementa la interfaz **`PathLike`**.

Los atributos y métodos en una instancia de `os.DirEntry` son los siguientes:

name

El nombre de archivo base de la entrada, relativo al argumento **`scandir()`** *path*.

El atributo **`name`** será `bytes` si el argumento **`scandir()`** *path* es de tipo `bytes` y `str` de lo contrario. Utilice **`fsdecode()`** para decodificar los nombres de archivo de bytes.

path

El nombre completo de la ruta de entrada: equivalente a `os.path.join(scandir_path, entry.name)` donde *scandir_path* es el argumento **`scandir()`** *path*. La ruta solo es absoluta si el argumento **`scandir()`** *path* fue absoluto. Si el argumento **`scandir()`** *path* era un descriptor de archivo, el atributo **`path`** es el mismo que el atributo **`name`**.

El atributo **`path`** será `bytes` si el argumento **`scandir()`** *path* es de tipo `bytes` y `str` de lo contrario. Utilice **`fsdecode()`** para decodificar los nombres de archivo de bytes.

inode()

Retorna el número de inodo de la entrada.

El resultado se almacena en caché en el objeto `os.DirEntry`. Use `os.stat(entry.path, follow_symlinks=False).st_ino` para obtener información actualizada.

En la primera llamada no almacenada en caché, se requiere una llamada del sistema en Windows pero no en Unix.

is_dir(*, *follow_symlinks=True*)

Retorna `True` si esta entrada es un directorio o un enlace simbólico que apunta a un directorio; retorna `False` si la entrada es o apunta a cualquier otro tipo de archivo, o si ya no existe.

Si *follow_symlinks* es `False`, retorna `True` solo si esta entrada es un directorio (sin seguir los enlaces simbólicos); retorna `False` si la entrada es cualquier otro tipo de archivo o si ya no existe.

El resultado se almacena en caché en el objeto `os.DirEntry`, con un caché separado para *follow_symlinks* `True` y `False`. Llame a **`os.stat()`** junto con **`stat.S_ISDIR()`** para obtener información actualizada.

En la primera llamada no almacenada en caché, no se requiere ninguna llamada al sistema en la mayoría de los casos. Específicamente, para los enlaces no simbólicos, ni Windows ni Unix requieren una llamada al sistema, excepto en ciertos sistemas de archivos Unix, como los sistemas de archivos de red, que retornan `dirent.d_type == DT_UNKNOWN`. Si la entrada es un enlace simbólico, se requerirá una llamada al sistema para seguir el enlace simbólico a menos que *follow_symlinks* sea `False`.

Este método puede generar **`OSError`**, como **`PermissionError`**, pero **`FileNotFoundError`** se captura y no se genera.

is_file(*, *follow_symlinks=True*)

Retorna `True` si esta entrada es un archivo o un enlace simbólico que apunta a un archivo; retorna `False` si la entrada es o apunta a un directorio u otra entrada que no sea de archivo, o si ya no existe.

Si *follow_symlinks* es `False`, retorna `True` solo si esta entrada es un archivo (sin los siguientes enlaces simbólicos); retorna `False` si la entrada es un directorio u otra entrada que no sea de archivo, o si ya no existe.

El resultado se almacena en caché en el objeto `os.DirEntry`. El almacenamiento en caché, las llamadas realizadas al sistema y las excepciones generadas son las siguientes **`is_dir()`**.

is_symlink()

Retorna `True` si esta entrada es un enlace simbólico (incluso si está roto); retorna `False` si la entrada apunta a un directorio o cualquier tipo de archivo, o si ya no existe.

El resultado se almacena en caché en el objeto `os.DirEntry`. Llame a **`os.path.islink()`** para obtener información actualizada.

En la primera llamada no almacenada en caché, no se requiere ninguna llamada al sistema en la mayoría de los casos. Específicamente, ni Windows ni Unix requieren una llamada al sistema, excepto en ciertos sistemas de archivos Unix, como los sistemas de archivos de red, que retornan `dirent.d_type == DT_UNKNOWN`.

Este método puede generar **`OSError`**, como **`PermissionError`**, pero **`FileNotFoundError`** se captura y no se genera.

is_junction()

Devuelve `True` si esta entrada es una unión (incluso si está rota); devuelve `False` si la entrada apunta a un directorio normal, cualquier tipo de archivo, un enlace simbólico o si ya no existe.

El resultado se almacena en caché en el objeto `os.DirEntry`. Llame a **`os.path.isjunction()`** para obtener información actualizada.

Added in version 3.12.

stat(**, follow_symlinks=True*)

Retorna un objeto a **`stat_result`** para esta entrada. Este método sigue enlaces simbólicos por defecto; para crear un enlace simbólico agregue el argumento `follow_symlinks=False`.

En Unix, este método siempre requiere una llamada al sistema. En Windows, solo requiere una llamada al sistema si *follow_symlinks* es `True` y la entrada es un punto de análisis (por ejemplo, un enlace simbólico o una unión de directorio).

En Windows, los atributos `st_ino`, `st_dev` y `st_nlink` de **`stat_result`** siempre se establecen en cero. Llame a **`os.stat()`** para obtener estos atributos.

El resultado se almacena en caché en el objeto `os.DirEntry`, con un caché separado para *follow_symlinks* `True` y `False`. Llame a **`os.stat()`** para obtener información actualizada.

Tenga en cuenta que existe una buena correspondencia entre varios atributos y métodos de `os.DirEntry` y **`pathlib.Path`**. En particular, el atributo `name` tiene el mismo significado, al igual que los métodos `is_dir()`, `is_file()`, `is_symlink()`, `is_junction()` y `stat()`.

Added in version 3.5.

Distinto en la versión 3.6: Se agregó soporte para la interfaz **`PathLike`**. Se agregó soporte para rutas de **`bytes`** en Windows.

Distinto en la versión 3.12: El atributo `st_ctime` de un resultado estadístico está obsoleto en Windows. La hora de creación del archivo está disponible correctamente como `st_birthtime` y, en el futuro, `st_ctime` puede cambiarse para que devuelva cero o la hora de cambio de metadatos, si está disponible.

os.stat(*path*, ***, *dir_fd=None*, *follow_symlinks=True*)

Obtener el estado de un archivo o un descriptor de archivo. Realice el equivalente de a llamada del sistema **`stat()`** en la ruta dada. *path* puede especificarse como una cadena o bytes, directa o indirectamente a través de la interfaz **`PathLike`**, o como un descriptor de archivo abierto. Retorna un objeto **`stat_result`**.

Esta función normalmente sigue enlaces simbólicos; para crear un enlace simbólico agregue el argumento `follow_symlinks=False`, o use **`lstat()`**.

Esta función puede soportar especificando un descriptor de archivo y no siguen enlaces simbólicos.

En Windows, pasar `follow_symlinks=False` deshabilitará el seguimiento de todos los puntos de análisis sustitutos de nombre, que incluyen enlaces simbólicos y uniones de directorio. Se abrirán directamente otros tipos de puntos de análisis que no se parecen a los enlaces o que el sistema operativo no puede seguir. Al seguir una cadena de enlaces múltiples, esto puede dar como resultado que se retorna el enlace original en lugar del no enlace que impidió el recorrido completo. Para obtener resultados estadísticos para la ruta final en este caso, use la función **`os.path.realpath()`** para resolver el nombre de la ruta lo más posible y llame a **`lstat()`** en el resultado. Esto no se aplica a enlaces simbólicos o puntos de unión colgantes, lo que lanzará las excepciones habituales.

Ejemplo:

```
>>> import os
>>> statinfo = os.stat('somefile.txt')
>>> statinfo
os.stat_result(st_mode=33188, st_ino=7876932, st_de
st_nlink=1, st_uid=501, st_gid=501, st_size=264, st_
st_mtime=1297230027, st_ctime=1297230027)
```

```
>>> statinfo.st_size
264
```

Ver también

`fstat()` y funciones `lstat()`.

Distinto en la versión 3.3: Se agregaron los parámetros *dir_fd* y *follow_symlinks*, especificando un descriptor de archivo en lugar de una ruta.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

Distinto en la versión 3.8: En Windows, ahora se siguen todos los puntos de análisis que el sistema operativo puede resolver, y pasar `follow_symlinks=False` desactiva los siguientes puntos de análisis sustitutos de nombre. Si el sistema operativo alcanza un punto de análisis que no puede seguir, *stat* ahora retorna la información de la ruta original como si se hubiera especificado `follow_symlinks=False` en lugar de generar un error.

class os.stat_result

Objeto cuyos atributos corresponden aproximadamente a los miembros de la estructura **`stat`**. Se utiliza para el resultado de `os.stat()`, `os.fstat()` y `os.lstat()`.

Atributos:

st_mode

Modo de archivo: tipo de archivo y bits de modo de archivo (permisos).

st_ino

Dependiendo de la plataforma, pero si no es cero, identifica de forma exclusiva el archivo para un valor dado de `st_dev`. Típicamente:

- el número de inodo en Unix,
- el índice del archivo [https://msdn.microsoft.com/en-

us/library/aa363788] en Windows

st_dev

Identificador del dispositivo en el que reside este archivo.

st_nlink

Número de enlaces duros.

st_uid

Identificador de usuario del propietario del archivo.

st_gid

Identificador de grupo del propietario del archivo.

st_size

Tamaño del archivo en bytes, si es un archivo normal o un enlace simbólico. El tamaño de un enlace simbólico es la longitud del nombre de ruta que contiene, sin un byte nulo de terminación.

Marcas de tiempo:

st_atime

Tiempo de acceso más reciente expresado en segundos.

st_mtime

Tiempo de modificación de contenido más reciente expresado en segundos.

st_ctime

Hora del cambio de metadatos más reciente expresado en segundos.

Distinto en la versión 3.12: `st_ctime` está obsoleto en Windows. Utilice `st_birthtime` para indicar la hora de creación del archivo. En el futuro, `st_ctime` contendrá la hora del cambio de metadatos más reciente, como en otras plataformas.

st_atime_ns

Tiempo de acceso más reciente expresado en nanosegundos como un entero.

Added in version 3.3.

st_mtime_ns

Hora de la modificación de contenido más reciente expresada en nanosegundos como un entero.

Added in version 3.3.

st_ctime_ns

Hora del cambio de metadatos más reciente expresado en nanosegundos como un número entero.

Added in version 3.3.

Distinto en la versión 3.12: `st_ctime_ns` está obsoleto en Windows. Utilice `st_birthtime_ns` para indicar la hora de creación del archivo. En el futuro, `st_ctime` contendrá la hora del cambio de metadatos más reciente, como en otras plataformas.

st_birthtime

Hora de creación del archivo expresada en segundos. Este atributo no siempre está disponible y puede generar el error **`AttributeError`**.

Distinto en la versión 3.12: `st_birthtime` ahora está disponible en Windows.

st_birthtime_ns

Hora de creación del archivo expresada en nanosegundos como un número entero. Este atributo no siempre está disponible y puede generar el error

`AttributeError`.

Added in version 3.12.

Nota

El significado y la resolución exactos de los atributos `st_atime`, `st_mtime`, `st_ctime` y `st_birthtime` dependen del sistema operativo y del sistema de archivos. Por ejemplo, en los sistemas Windows que utilizan sistemas de archivos FAT32, `st_mtime` tiene una resolución de 2 segundos y `st_atime` tiene una resolución de solo 1 día. Consulte la documentación de su sistema operativo para obtener más detalles.

De manera similar, aunque `st_atime_ns`, `st_mtime_ns`, `st_ctime_ns` y `st_birthtime_ns` siempre se expresan en nanosegundos, muchos sistemas no proporcionan precisión en nanosegundos. En los sistemas que sí proporcionan precisión en nanosegundos, el objeto de punto flotante utilizado para almacenar `st_atime`, `st_mtime`, `st_ctime` y `st_birthtime` no puede conservarlo todo y, por lo tanto, será ligeramente inexacto. Si necesita las marcas de tiempo exactas, siempre debe utilizar `st_atime_ns`, `st_mtime_ns`, `st_ctime_ns` y `st_birthtime_ns`.

En algunos sistemas Unix (como Linux), los siguientes atributos también pueden estar disponibles:

st_blocks

Número de bloques de 512 bytes asignados para el archivo. Esto puede ser más pequeño que `st_size` / 512 cuando el archivo tiene agujeros.

st_blksize

Tamaño de bloque «preferido» para una eficiente E / S del sistema de archivos. Escribir en un archivo en fragmentos más pequeños puede causar una lecturamodificación-reescritura ineficiente.

st_rdev

Tipo de dispositivo si es un dispositivo inodo.

st_flags

Indicadores definidos por el usuario para el archivo.

En otros sistemas Unix (como FreeBSD), los siguientes atributos pueden estar disponibles (pero solo se pueden completar si la raíz intenta usarlos):

st_gen

Número de generación de archivos.

En Solaris y derivados, los siguientes atributos también pueden estar disponibles:

st_fstype

Cadena que identifica de forma exclusiva el tipo de sistema de archivos que contiene el archivo.

En los sistemas macOS, los siguientes atributos también pueden estar disponibles:

st_rsize

Tamaño real del archivo.

st_creator

Creador del archivo.

st_type

Tipo de archivo.

En los sistemas Windows, los siguientes atributos también están disponibles:

st_file_attributes

Atributos de archivo de Windows: `dwFileAttributes` miembro de la estructura `BY_HANDLE_FILE_INFORMATION` devuelta por **`GetFileInformationByHandle()`**. Consulte las constantes **`FILE_ATTRIBUTE_* <stat.FILE_ATTRIBUTE_ARCHIVE>`** en el módulo **`stat`**.

Added in version 3.5.

st_reparse_tag

Cuando **`st_file_attributes`** tiene **`FILE_ATTRIBUTE_REPARSE_POINT`** configurado, este campo contiene la etiqueta que identifica el tipo de punto de análisis. Consulte las constantes **`IO_REPARSE_TAG_*`** en el módulo **`stat`**.

El módulo estándar **`stat`** define funciones y constantes que son útiles para extraer información de una estructura **`stat`**. (En Windows, algunos elementos se rellenan con valores ficticios).

Para compatibilidad con versiones anteriores, también se puede acceder a una instancia **`stat_result`** como una tupla de al menos 10 números enteros que den los miembros más importantes (y portables) de la estructura **`stat`**, en el orden **`st_mode`**, **`st_ino`**, **`st_dev`**, **`st_nlink`**, **`st_uid`**, **`st_gid`**, **`st_size`**, **`st_atime`**, **`st_mtime`**, **`st_ctime`**. Algunas implementaciones pueden agregar más elementos al final. Para compatibilidad con versiones anteriores de Python, acceder a **`stat_result`** como una tupla siempre devuelve números enteros.

Distinto en la versión 3.5: Windows ahora retorna el índice del archivo como **`st_ino`** cuando está disponible.

Distinto en la versión 3.7: Se agregó el miembro **`st_fstype`** a Solaris/derivados.

Distinto en la versión 3.8: Se agregó el miembro `st_reparse_tag` en Windows.

Distinto en la versión 3.8: En Windows, el miembro `st_mode` ahora identifica archivos especiales como **`S_IFCHR`**, **`S_IFIFO`** o **`S_IFBLK`** según corresponda.

Distinto en la versión 3.12: En Windows, `st_ctime` está obsoleto. Con el tiempo, contendrá la última hora de cambio de metadatos, para mantener la coherencia con otras plataformas, pero por ahora aún contiene la hora de creación. Utilice `st_birthtime` para la hora de creación.

En Windows, `st_ino` puede tener hasta 128 bits, según el sistema de archivos. Antes no superaba los 64 bits y los identificadores de archivos más grandes se empaquetaban de forma arbitraria.

En Windows, `st_rdev` ya no devuelve un valor. Antes contenía el mismo valor que `st_dev`, lo cual era incorrecto.

Se agregó el miembro `st_birthtime` en Windows.

os.statvfs(*path*)

Realizar una llamada al sistema **`statvfs()`** en la ruta indicada. El valor de retorno es un objeto cuyos atributos describen el sistema de archivos en la ruta indicada y corresponden a los miembros de la estructura **`statvfs`**, a saber: **`f_bsize`**, **`f_frsize`**, **`f_blocks`**, **`f_bfree`**, **`f_bavail`**, **`f_files`**, **`f_ffree`**, **`f_favail`**, **`f_flag`**, **`f_namemax`**, **`f_fsid`**.

Se definen dos constantes de nivel de módulo para: indicadores de bit del atributo **`f_flag`**: si **`ST_RDONLY`** está configurado, el sistema de archivos está montado de solo lectura, y si **`ST_NOSUID`** está configurado, el la semántica de los bits setuid/setgid está deshabilitada o no es compatible.

Se definen constantes de nivel de módulo adicionales para sistemas basados en GNU / glibc. Estos son **`ST_NODEV`** (no permitir el acceso a archivos especiales del dispositivo), **`ST_NOEXEC`** (no permitir la ejecución del programa),

`ST_SYNCHRONOUS` (las escrituras se sincronizan a la vez), **`ST_MANDLOCK`** (permitir bloqueos obligatorios en un FS), **`ST_WRITE`** (escribir en el archivo/directorio/enlace simbólico), **`ST_APPEND`** (archivo de solo agregado), **`ST_IMMUTABLE`** (archivo inmutable), **`ST_NOATIME`** (no actualiza los tiempos de acceso), **`ST_NODIRATIME`** (no actualiza los tiempos de acceso al directorio), **`ST_RELATIME`** (tiempo de actualización relativo a mtime/ctime).

Esta función puede soportar especificando un descriptor de archivo.

Availability: Unix.

Distinto en la versión 3.2: Se agregaron las constantes **`ST_RDONLY`** y **`ST_NOSUID`**.

Distinto en la versión 3.3: Se agregó soporte para especificar *path* como un descriptor de archivo abierto.

Distinto en la versión 3.4: El **`ST_NODEV`**, **`ST_NOEXEC`**, **`ST_SYNCHRONOUS`**, **`ST_MANDLOCK`**, **`ST_WRITE`**, **`ST_APPEND`**, **`ST_IMMUTABLE`**, **`ST_NOATIME`**, **`ST_NODIRATIME`**, y **`ST_RELATIME`** se agregaron constantes.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

Distinto en la versión 3.7: Se agrega el atributo **`f_fsid`**.

os.supports_dir_fd

A **`set`** object indicating which functions in the **`os`** module accept an open file descriptor for their *dir_fd* parameter. Different platforms provide different features, and the underlying functionality Python uses to implement the *dir_fd* parameter is not available on all platforms Python supports. For consistency's sake, functions that may support *dir_fd* always allow specifying the parameter, but will throw an exception if the functionality is used when it's not locally available. (Specifying `None` for *dir_fd* is always supported on all platforms.)

Para verificar si una función particular acepta un descriptor de archivo abierto para su parámetro *dir_fd*, use el operador `in` en `supports_dir_fd`. Como ejemplo, esta expresión se evalúa como `True` si **`os.stat()`** acepta descriptores de archivos abiertos para *dir_fd* en la plataforma local:

```
os.stat en os.supports_dir_fd
```

Actualmente, los parámetros *dir_fd* solo funcionan en plataformas Unix; ninguno de ellos funciona en Windows.

Added in version 3.3.

os.supports_effective_ids

Un objeto **`set`** que indica si **`os.access()`** permite especificar `True` para su parámetro *fective_ids* en la plataforma local. (Especificar `False` para *effective_id* siempre es compatible con todas las plataformas). Si la plataforma local lo admite, la colección contendrá **`os.access()`**; de lo contrario estará vacío.

Esta expresión se evalúa como `True` si **`os.access()`** admite `effective_id=True` en la plataforma local:

```
os.access en os.supports_effective_ids
```

Actualmente, *effective_ids* solo es compatible con plataformas Unix; No funciona en Windows.

Added in version 3.3.

os.supports_fd

A **`set`** object indicating which functions in the **`os`** module permit specifying their *path* parameter as an open file descriptor on the local platform. Different platforms provide different features, and the underlying functionality Python uses to accept open file descriptors as *path* arguments is not available on all platforms Python supports.

Para determinar si una función en particular permite especificar un descriptor de archivo abierto para su parámetro *path*, use el operador `in` en `supports_fd`. Como ejemplo, esta expresión se evalúa como `True` si **`os.chdir()`** acepta descriptores de archivo abiertos para *path* en su plataforma local:

```
os.chdir en os.supports_fd
```

Added in version 3.3.

os.supports_follow_symlinks

A **`set`** object indicating which functions in the **`os`** module accept `False` for their *follow_symlinks* parameter on the local platform. Different platforms provide different features, and the underlying functionality Python uses to implement *follow_symlinks* is not available on all platforms Python supports. For consistency's sake, functions that may support *follow_symlinks* always allow specifying the parameter, but will throw an exception if the functionality is used when it's not locally available. (Specifying `True` for *follow_symlinks* is always supported on all platforms.)

Para verificar si una función particular acepta `False` para su parámetro *follow_symlinks*, use el operador `in` en `supports_follow_symlinks`. Como ejemplo, esta expresión se evalúa como `True` si puede especificar `follow_symlinks=False` al llamar a **`os.stat()`** en la plataforma local:

```
os.stat en os.supports_follow_symlinks
```

Added in version 3.3.

os.symlink(*src*, *dst*, *target_is_directory=False*, ***, *dir_fd=None*)

Cree un enlace simbólico que apunte a *src* llamado *dst*.

The *src* parameter refers to the target of the link (the file or directory being linked to), and *dst* is the name of the link being created.

En Windows, un enlace simbólico representa un archivo o un directorio, y no se transforma dinámicamente en el destino. Si el objetivo está presente, el tipo de enlace simbólico se creará para que coincida. De lo contrario, el enlace simbólico se creará como un directorio si *target_is_directory* es `True` o un enlace simbólico de archivo (el valor predeterminado) de lo contrario. En plataformas que no son de Windows, *target_is_directory* se ignora.

Esta función puede admitir rutas relativas a descriptores de directorio.

Nota

En las versiones más recientes de Windows 10, las cuentas sin privilegios pueden crear enlaces simbólicos si el Modo desarrollador está habilitado. Cuando el Modo desarrollador no está disponible / habilitado, se requiere el privilegio *SeCreateSymbolicLinkPrivilege*, o el proceso debe ejecutarse como administrador.

`OSError` se lanza cuando un usuario sin privilegios llama a la función.

Lanza un evento de auditoría `os.symlink` con argumentos `src`, `dst`, `dir_fd`.

Availability: Unix, Windows.

La función está limitada en WASI, consulte Plataformas WebAssembly para obtener más información.

Distinto en la versión 3.2: Se agregó soporte para enlaces simbólicos de Windows 6.0 (Vista).

Distinto en la versión 3.3: Se agregó el parámetro *dir_fd* y ahora se permite *target_is_directory* en plataformas que no sean Windows.

Distinto en la versión 3.6: Acepta un path-like object para *src* y *dst*.

Distinto en la versión 3.8: Se agregó soporte para enlaces simbólicos sin elevar en Windows con el modo de desarrollador.

os.sync()

Forzar la escritura de todo en el disco.

Availability: Unix.

Added in version 3.3.

os.truncate(*path*, *length*)

Trunca el archivo correspondiente a *path*, para que tenga como máximo *length* bytes de tamaño.

Esta función puede soportar especificando un descriptor de archivo.

Lanza un evento de auditoría `os.truncate` con argumentos `path`, `length`.

Availability: Unix, Windows.

Added in version 3.3.

Distinto en la versión 3.5: Se agregó soporte para Windows

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.unlink(*path*, ***, *dir_fd=None*)

Elimina (elimine) el archivo *path*. Esta función es semánticamente idéntica a **`remove()`**; El nombre `unlink` es su nombre tradicional de Unix. Consulte la documentación de **`remove()`** para obtener más información.

Lanza un evento de auditoría `os.remove` con argumentos `ruta`, `dir_fd`.

Distinto en la versión 3.3: Se agregó el parámetro *dir_fd*.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.utime(*path*, *times=None*, ***, [*ns*,]*dir_fd=None*, *follow_symlinks=True*)

Establece el acceso y los tiempos modificados del archivo especificado por *path*.

`utime()` toma dos parámetros opcionales, *times* y *ns*. Estos especifican los tiempos establecidos en *path* y se utilizan de la siguiente manera:

- Si se especifica *ns*, debe ser una tupla de 2 de la forma `(atime_ns, mtime_ns)` donde cada miembro es un int que expresa nanosegundos.
- Si *times* no es `None`, debe ser una 2-tupla de la forma `(atime, mtime)` donde cada miembro es un int o flotante que expresa segundos.
- Si *times* es `None` y *ns* no está especificado, esto es equivalente a especificar `ns=(atime_ns, mtime_ns)` donde ambas horas son la hora actual.

Es un error especificar tuplas para *times* y *ns*.

Tenga en cuenta que es posible que las horas exactas que establezca aquí no se devuelvan en una llamada **`stat()`** posterior, dependiendo de la resolución con la que su sistema operativo registre las horas de acceso y modificación; ver **`stat()`**. La mejor manera de conservar las horas exactas es usar los campos *st_atime_ns* y *st_mtime_ns* del objeto de resultado **`os.stat()`** con el parámetro *ns* a **`utime()`**.

Esta función puede soportar especificando un descriptor de archivo, rutas relativas a los descriptores de directorio y no seguir enlaces simbólicos.

Lanza un evento de auditoría `os.utime` con argumentos `path`, `times`, `ns`, `dir_fd`.

Distinto en la versión 3.3: Se agregó soporte para especificar *path* como un descriptor de archivo abierto, y los parámetros *dir_fd*, *follow_symlinks* y *ns*.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.walk(*top*, *topdown=True*, *onerror=None*, *followlinks=False*)

Genere los nombres de archivo en un árbol de directorios recorriendo el árbol de arriba hacia abajo o de abajo hacia arriba. Para cada directorio en el árbol enraizado en el directorio *top* (incluido *top*), produce una tupla de 3 tuplas `(dirpath, dirnames, filenames)`.

dirpath es una cadena, la ruta al directorio. *dirnames* es una lista de los nombres de los subdirectorios en *dirpath* (incluidos los enlaces simbólicos a los directorios y excluyendo `'.'` y `'..'`). *filenames* es una lista de los nombres de los archivos que no son de directorio en *dirpath*. Tenga en cuenta que los nombres de las listas no contienen componentes de ruta. Para obtener una ruta completa (que comienza con *top*) a un archivo o directorio en *dirpath*, ejecute `os.path.join(dirpath, name)`. Que las listas estén ordenadas o no depende del sistema de archivos. Si un archivo se elimina o se agrega al directorio *dirpath* durante la generación de las listas, no se especifica si se incluirá un nombre para ese archivo.

Si el argumento opcional *topdown* es `True` o no se especifica, el triple para un directorio se genera antes de triplicarse para cualquiera de sus subdirectorios (los directorios se generan de arriba hacia abajo). Si *topdown* es `False`, el triple para un directorio se genera después de los triples para todos sus subdirectorios (los directorios se generan de abajo hacia arriba). No importa el valor de *topdown*, la lista de subdirectorios se recupera antes de que se generen las tuplas para el directorio y sus subdirectorios.

Cuando *topdown* es `True`, la persona que llama puede modificar la lista *dirnames* en su lugar (quizás usando **`del`** o asignación de corte) y **`walk()`** solo se repetirá en los subdirectorios cuyos nombres permanecen en *dirnames*; Esto se puede utilizar para podar la búsqueda, imponer un orden específico de visitas o incluso para informar `walk()` sobre los directorios que la persona que llama crea o renombra antes de que se reanude `walk()` nuevamente. La modificación de *dirnames* cuando *topdown* es `False` no tiene ningún efecto en el comportamiento de la caminata, porque en el modo ascendente los directorios en *dirnames* se generan antes de que se genere *dirpath*.

Por defecto, los errores de la llamada `scandir()` se ignoran. Si se especifica el argumento opcional *onerror*, debería ser una función; se llamará con un argumento, una instancia `OSError`. Puede informar el error para continuar con la caminata, o generar la excepción para abortar la caminata. Tenga en cuenta que el nombre de archivo está disponible como el atributo `filename` del objeto de excepción.

Por defecto, `walk()` no entrará en enlaces simbólicos que se resuelven en directorios. Establece *followlinks* en `True` para visitar los directorios señalados por los enlaces simbólicos, en los sistemas que los admiten.

Nota

Tenga en cuenta que establecer *followlinks* en `True` puede conducir a una recursión infinita si un enlace apunta a un directorio padre de sí mismo. `walk()` no realiza un seguimiento de los directorios que ya visitó.

Nota

Si pasa un nombre de ruta relativo, no cambie el directorio de trabajo actual entre las reanudaciones de `walk()`. `walk()` nunca cambia el directorio actual, y supone que la persona que llama tampoco.

This example displays the number of bytes taken by non-directory files in each directory under the starting directory, except that it doesn't look under any `__pycache__` subdirectory:

```
import os
from os.path import join, getsize
for root, dirs, files in os.walk('python/Lib/xml'):
    print(root, "consumes", end=" ")
    print(sum(getsize(join(root, name)) for name in
    print("bytes in", len(files), "non-directory fi
    if '__pycache__' in dirs:
        dirs.remove('__pycache__')  # don't visit _
```

En el siguiente ejemplo (implementación simple de **`shutil.rmtree()`**), recorrer el árbol de abajo hacia arriba es esencial, **`rmdir()`** no permite eliminar un directorio antes de que el directorio esté vacío:

```
# Delete everything reachable from the directory na
# assuming there are no symbolic links.
# CAUTION:  This is dangerous!  For example, if top
# could delete all your disk files.
import os
for root, dirs, files in os.walk(top, topdown=False
    for name in files:
        os.remove(os.path.join(root, name))
    for name in dirs:
        os.rmdir(os.path.join(root, name))
os.rmdir(top)
```

Lanza un evento de auditoría `os.spawn` con argumentos `top`, `topdown`, `onerror`, `followlinks`.

Distinto en la versión 3.5: Esta función ahora llama **`os.scandir()`** en lugar de **`os.listdir()`**, lo que lo hace más rápido al reducir el número de llamadas a **`os.stat()`**.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.fwalk(*top='.'*, *topdown=True*, *onerror=None*, ***, *follow_symlinks=False*, *dir_fd=None*)

Esto se comporta exactamente como **`walk()`**, excepto que produce 4 tuplas `(dirpath, dirnames, filenames,` `dirfd)`, y admite `dir_fd`.

dirpath, *dirnames* y *filenames* son idénticos a **`walk()`** output, y *dirfd* es un descriptor de archivo que se refiere al directorio *dirpath*.

Esta función siempre admite rutas relativas a descriptores de directorio y no siguen enlaces simbólicos. Sin embargo, tenga en cuenta que, a diferencia de otras funciones, el valor predeterminado **`fwalk()`** para *follow_symlinks* es `False`.

Nota

Dado que **`fwalk()`** produce descriptores de archivo, estos solo son válidos hasta el siguiente paso de iteración, por lo que debe duplicarlos (por ejemplo, con **`dup()`**) si desea mantenerlos más tiempo.

This example displays the number of bytes taken by non-directory files in each directory under the starting directory, except that it doesn't look under any `__pycache__` subdirectory:

```
import os
for root, dirs, files, rootfd in os.fwalk('python/L
    print(root, "consumes", end=" ")
    print(sum([os.stat(name, dir_fd=rootfd).st_size
          end=" ")
    print("bytes in", len(files), "non-directory fi
    if '__pycache__' in dirs:
        dirs.remove('__pycache__')  # don't visit _
```

En el siguiente ejemplo, recorrer el árbol de abajo hacia arriba es esencial: **`rmdir()`** no permite eliminar un directorio antes de que el directorio esté vacío:

```
# Borre todo lo que sea accesible desde el director
# asumiendo que no hay enlaces simbólicos.
# PRECAUCIÓN: ¡Esto es peligroso! Por ejemplo, si t
# podría borrar todos los archivos de su disco.
```

```
import os
for root, dirs, files, rootfd in os.fwalk(top, topd
for name in files:
os.unlink(name, dir_fd=rootfd)
for name in dirs:
os.rmdir(name, dir_fd=rootfd)
```

Lanza un evento de auditoría `os.chown` con argumentos `top`, `topdown`, `onerror`, `follow_symlinks`, `dir_fd`.

Availability: Unix.

Added in version 3.3.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

Distinto en la versión 3.7: Se agregó soporte para rutas de acceso **`bytes`**.

os.memfd_create(*name*[, *flags=os.MFD_CLOEXEC*])

Cree un archivo anónimo y retorna un descriptor de archivo que se refiera a él. *flags* debe ser una de las constantes `os.MFD_*` disponibles en el sistema (o una combinación ORed bit a bit de ellas). Por defecto, el nuevo descriptor de archivo es no heredable.

El nombre proporcionado en *name* se utiliza como nombre de archivo y se mostrará como el destino del enlace simbólico correspondiente en el directorio `/proc/self/fd/`. El nombre que se muestra siempre tiene el prefijo `memfd:` y solo sirve para fines de depuración. Los nombres no afectan el comportamiento del descriptor de archivo y, como tal, varios archivos pueden tener el mismo nombre sin efectos secundarios.

Availability: Linux $>=$ 3.17 with glibc $>=$ 2.27.

Added in version 3.8.

os.MFD_CLOEXEC

os.MFD_ALLOW_SEALING
os.MFD_HUGETLB
os.MFD_HUGE_SHIFT
os.MFD_HUGE_MASK
os.MFD_HUGE_64KB
os.MFD_HUGE_512KB
os.MFD_HUGE_1MB
os.MFD_HUGE_2MB
os.MFD_HUGE_8MB
os.MFD_HUGE_16MB
os.MFD_HUGE_32MB
os.MFD_HUGE_256MB
os.MFD_HUGE_512MB
os.MFD_HUGE_1GB
os.MFD_HUGE_2GB
os.MFD_HUGE_16GB

Estas flags se pueden pasar a **`memfd_create()`**.

Availability: Linux >= 3.17 with glibc >= 2.27

Las banderas `MFD_HUGE*` solo están disponibles desde Linux 4.14.

Added in version 3.8.

os.eventfd(*initval*[, *flags=os.EFD_CLOEXEC*])

Crea y retorna un descriptor de archivo de eventos. Los descriptores de archivo admiten **`read()`** y **`write()`** sin procesar con un tamaño de búfer de 8, **`select()`**, **`poll()`** y similares. Consulte la página de manual *eventfd(2)* *[https://manpages.debian.org/eventfd(2)]* para obtener más información. De forma predeterminada, el nuevo descriptor de archivo no es heredable.

initval es el valor inicial del contador de eventos. El valor inicial debe ser un entero sin signo de 32 bits. Tenga en cuenta que el valor inicial está limitado a un entero sin signo de 32 bits, aunque el contador de eventos es un entero sin signo de 64 bits con un valor máximo de 2^{64}-2.

flags se puede construir a partir de `EFD_CLOEXEC`, `EFD_NONBLOCK` y `EFD_SEMAPHORE`.

Si se especifica `EFD_SEMAPHORE` y el contador de eventos no es cero, `eventfd_read()` retorna 1 y reduce el contador en uno.

Si no se especifica `EFD_SEMAPHORE` y el contador de eventos no es cero, `eventfd_read()` retorna el valor actual del contador de eventos y restablece el contador a cero.

Si el contador de eventos es cero y no se especifica `EFD_NONBLOCK`, `eventfd_read()` se bloquea.

`eventfd_write()` incrementa el contador de eventos. Escribe bloques si la operación de escritura incrementaría el contador a un valor mayor que ^{64}-2.

Ejemplo:

```
import os

# semaphore with start value '1'
fd = os.eventfd(1, os.EFD_SEMAPHORE | os.EFD_CLOEXE
try:
    # acquire semaphore
    v = os.eventfd_read(fd)
    try:
        do_work()
    finally:
        # release semaphore
        os.eventfd_write(fd, v)
finally:
    os.close(fd)
```

Availability: Linux >= 2.6.27 with glibc >= 2.8

Added in version 3.10.

os.eventfd_read(*fd*)

Lee el valor de un descriptor de archivo `eventfd()` y retorna un int sin signo de 64 bits. La función no verifica que *fd* sea un **`eventfd()`**.

Availability: Linux >= 2.6.27

Added in version 3.10.

os.eventfd_write(*fd*, *value*)

Agregue valor a un descriptor de archivo `eventfd()`. *value* debe ser un int sin signo de 64 bits. La función no verifica que *fd* sea un **`eventfd()`**.

Availability: Linux >= 2.6.27

Added in version 3.10.

os.EFD_CLOEXEC

Establezca el indicador close-on-exec para el nuevo descriptor de archivo `eventfd()`.

Availability: Linux >= 2.6.27

Added in version 3.10.

os.EFD_NONBLOCK

Establezca el indicador de estado `O_NONBLOCK` para el nuevo descriptor de archivo `eventfd()`.

Availability: Linux >= 2.6.27

Added in version 3.10.

os.EFD_SEMAPHORE

Proporcionar semántica similar a la de un semáforo para las lecturas desde un descriptor de archivo `eventfd()`. Al leer, el contador interno se reduce en uno.

Availability: Linux >= 2.6.30

Added in version 3.10.

Descriptores de archivos del temporizador

Added in version 3.13.

Estas funciones brindan soporte para la API *timer file descriptor* de Linux. Naturalmente, todas ellas solo están disponibles en Linux.

os.timerfd_create(*clockid*, */*, ***, *flags=0*)

Crea y devuelve un descriptor de archivo de temporizador (*timerfd*).

El descriptor de archivo devuelto por **`timerfd_create()`** admite:

- **`read()`**
- **`select()`**
- **`poll()`**

El método **`read()`** del descriptor de archivo se puede llamar con un tamaño de búfer de 8. Si el temporizador ya ha expirado una o más veces, **`read()`** devuelve la cantidad de expiraciones con el orden de bytes del host, que se puede convertir a **`int`** mediante `int.from_bytes(x, byteorder=sys.byteorder)`.

Se pueden usar **`select()`** y **`poll()`** para esperar hasta que el temporizador expire y el descriptor de archivo sea legible.

clockid debe ser un clock ID válido, como se define en el módulo **`time`**:

- **`time.CLOCK_REALTIME`**
- **`time.CLOCK_MONOTONIC`**
- **`time.CLOCK_BOOTTIME`** (desde Linux 3.15 para timerfd_create)

Si *clockid* es **`time.CLOCK_REALTIME`**, se utiliza un reloj de tiempo real configurable para todo el sistema. Si se modifica el reloj del sistema, es necesario actualizar la configuración del temporizador. Para cancelar el temporizador cuando se modifica el reloj del sistema, consulte `TFD_TIMER_CANCEL_ON_SET`.

Si *clockid* es `time.CLOCK_MONOTONIC`, se utiliza un reloj que aumenta de forma monótona y no se puede configurar. Incluso si se cambia el reloj del sistema, la configuración del temporizador no se verá afectada.

Si *clockid* es `time.CLOCK_BOOTTIME`, igual que `time.CLOCK_MONOTONIC` excepto que incluye cualquier momento en que el sistema esté suspendido.

The file descriptor's behaviour can be modified by specifying a *flags* value. Any of the following variables may be used, combined using bitwise OR (the `|` operator):

- `TFD_NONBLOCK`
- `TFD_CLOEXEC`

Si `TFD_NONBLOCK` no está configurado como indicador, `read()` se bloquea hasta que el temporizador expire. Si está configurado como indicador, `read()` no se bloquea, pero si no ha habido una expiración desde la última llamada para leer, `read()` genera `OSError` y `errno` se configura como `errno.EAGAIN`.

Python siempre establece `TFD_CLOEXEC` automáticamente.

El descriptor de archivo debe cerrarse con `os.close()` cuando ya no sea necesario, de lo contrario se filtrará.

Ver también

La página del manual *timerfd_create(2)* *[https://manpages.debian.org/timerfd_create(2)]*.

Availability: Linux >= 2.6.27 with glibc >= 2.8

Added in version 3.13.

os.timerfd_settime(*fd*, /, *, *flags*=*flags*, *initial*=*0.0*, *interval*=*0.0*)

Modificar el temporizador interno de un descriptor de archivo de temporizador. Esta función opera con el mismo temporizador de intervalos que **`timerfd_settime_ns()`**.

fd debe ser un descriptor de archivo de temporizador válido.

The timer's behaviour can be modified by specifying a *flags* value. Any of the following variables may be used, combined using bitwise OR (the `|` operator):

- **`TFD_TIMER_ABSTIME`**
- **`TFD_TIMER_CANCEL_ON_SET`**

El temporizador se desactiva al configurar *initial* en cero (`0`). Si *initial* es igual o mayor que cero, el temporizador se activa. Si *initial* es menor que cero, genera una excepción **`OSError`** con `errno` configurado en **`errno.EINVAL`**

De forma predeterminada, el temporizador se activará cuando transcurran *initial* segundos. (Si *initial* es cero, el temporizador se activará inmediatamente).

Sin embargo, si se establece el indicador **`TFD_TIMER_ABSTIME`**, el temporizador se activará cuando el reloj del temporizador (establecido por *clockid* en **`timerfd_create()`**) alcance los *initial* segundos.

El intervalo del temporizador se establece mediante *interval* **`float`**. Si *interval* es cero, el temporizador solo se activa una vez, en la expiración inicial. Si *interval* es mayor que cero, el temporizador se activa cada vez que transcurren *interval* segundos desde la expiración anterior. Si *interval* es menor que cero, genera **`OSError`** con `errno` establecido en **`errno.EINVAL`**

Si el indicador **`TFD_TIMER_CANCEL_ON_SET`** se configura junto con **`TFD_TIMER_ABSTIME`** y el reloj de este temporizador es **`time.CLOCK_REALTIME`**, el temporizador se marca como cancelable si el reloj de tiempo real se modifica de manera discontinua. La lectura del descriptor se cancela con el error ECANCELED.

Linux administra el reloj del sistema como UTC. La transición al horario de verano se realiza modificando únicamente la diferencia horaria y no provoca cambios discontinuos en el reloj del sistema.

El cambio discontinuo del reloj del sistema será causado por los siguientes eventos:

- `settimeofday`
- `clock_settime`
- Establezca la fecha y la hora del sistema mediante el comando `date`

Devuelve una tupla de dos elementos de (`next_expiration, interval`) del estado del temporizador anterior, antes de que se ejecutara esta función.

Ver también

timerfd_create(2) [https://manpages.debian.org/timerfd_create(2)], *timerfd_settime(2) [https://manpages.debian.org/timerfd_settime(2)]*, *settimeofday(2) [https://manpages.debian.org/settimeofday(2)]*, *clock_settime(2) [https://manpages.debian.org/clock_settime(2)]*, y *date(1) [https://manpages.debian.org/date(1)]*.

Availability: Linux >= 2.6.27 with glibc >= 2.8

Added in version 3.13.

os.timerfd_settime_ns(*fd*, */*, ***, *flags=0*, *initial=0*, *interval=0*)

Similar a **`timerfd_settime()`**, pero utiliza el tiempo en nanosegundos. Esta función opera con el mismo temporizador de intervalos que **`timerfd_settime()`**.

Availability: Linux >= 2.6.27 with glibc >= 2.8

Added in version 3.13.

os.timerfd_gettime(*fd*, */*)

Devuelve una tupla de dos elementos de flotantes (`next_expiration, interval`).

`next_expiration` denotes the relative time until the timer next fires, regardless of if the **`TFD_TIMER_ABSTIME`** flag is set.

`interval` indica el intervalo del temporizador. Si es cero, el temporizador se activará solo una vez, después de que transcurran `next_expiration` segundos.

Ver también

timerfd_gettime(2) [https://manpages.debian.org/timerfd_gettime(2)]

Availability: Linux > = 2.6.27 with glibc > = 2.8

Added in version 3.13.

os.timerfd_gettime_ns(*fd*, */*)

Similar a **`timerfd_gettime()`**, pero devuelve el tiempo en nanosegundos.

Availability: Linux > = 2.6.27 with glibc > = 2.8

Added in version 3.13.

os.TFD_NONBLOCK

Un indicador para la función **`timerfd_create()`**, que establece el indicador de estado **`O_NONBLOCK`** para el nuevo descriptor de archivo del temporizador. Si **`TFD_NONBLOCK`** no está establecido como indicador, **`read()`** se bloquea.

Availability: Linux > = 2.6.27 with glibc > = 2.8

Added in version 3.13.

os.TFD_CLOEXEC

Una bandera para la función `timerfd_create()`. Si `TFD_CLOEXEC` está configurado como bandera, establece la bandera de cierre al ejecutar para el nuevo descriptor de archivo.

Availability: Linux >= 2.6.27 with glibc >= 2.8

Added in version 3.13.

os.TFD_TIMER_ABSTIME

Indicador para las funciones `timerfd_settime()` y `timerfd_settime_ns()`. Si se establece este indicador, *initial* se interpreta como un valor absoluto en el reloj del temporizador (en segundos UTC o nanosegundos desde la época Unix).

Availability: Linux >= 2.6.27 with glibc >= 2.8

Added in version 3.13.

os.TFD_TIMER_CANCEL_ON_SET

Una bandera para las funciones `timerfd_settime()` y `timerfd_settime_ns()` junto con `TFD_TIMER_ABSTIME`. El temporizador se cancela cuando la hora del reloj subyacente cambia de manera discontinua.

Availability: Linux >= 2.6.27 with glibc >= 2.8

Added in version 3.13.

Atributos extendidos de Linux

Added in version 3.3.

Estas funciones están disponibles solo en Linux.

os.getxattr(*path*, *attribute*, *, *follow_symlinks=True*)

Retorna el valor del atributo del sistema de archivos extendido *atrribute* para *path*. *attribute* puede ser bytes o str (directa o indirectamente a través de la interfaz `PathLike`).

Si es str, se codifica con la codificación del sistema de archivos.

Esta función puede soportar especificando un descriptor de archivo y no siguen enlaces simbólicos.

Lanza un evento de auditoría `os.getxattr` con argumentos `path`, `atributo`.

Distinto en la versión 3.6: Acepta un path-like object para *path*y *attribute*.

os.listxattr(*path=None*, ***, *follow_symlinks=True*)

Retorna una lista de los atributos del sistema de archivos extendido en *path*. Los atributos en la lista se representan como cadenas decodificadas con la codificación del sistema de archivos. Si *path* es `None`, **`listxattr()`** examinará el directorio actual.

Esta función puede soportar especificando un descriptor de archivo y no siguen enlaces simbólicos.

Lanza un evento de auditoría `os.listxattr` con el argumento `path`.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os.removexattr(*path*, *attribute*, ***, *follow_symlinks=True*)

Elimina el atributo extendido del sistema de archivos *attribute* de *path*. *attribute* debe ser bytes o str (directa o indirectamente a través de la interfaz **`PathLike`**). Si es una cadena, está codificada con filesystem encoding and error handler.

Esta función puede soportar especificando un descriptor de archivo y no siguen enlaces simbólicos.

Lanza un evento de auditoría `os.removexattr` con argumentos `path`, `attribute`.

Distinto en la versión 3.6: Acepta un path-like object para *path*y *attribute*.

os.setxattr(*path*, *attribute*, *value*, *flags=0*, ***, *follow_symlinks=True*)

Establece el atributo del sistema de archivos extendido *attribute* en *path* en *value*. *attribute* debe ser un bytes o una cadena sin NUL incrustados (directa o indirectamente a través de la interfaz `PathLike`). Si es una cadena, está codificado con filesystem encoding and error handler. *flags* puede ser `XATTR_REPLACE` o `XATTR_CREATE`. Si se proporciona **`XATTR_REPLACE`** y el atributo no existe, se lanzará `ENODATA`. Si se proporciona **`XATTR_CREATE`** y el atributo ya existe, no se creará el atributo y se lanza `EEXISTS`.

Esta función puede soportar especificando un descriptor de archivo y no siguen enlaces simbólicos.

Nota

Un error en las versiones de kernel de Linux anteriores a 2.6.39 hizo que el argumento de las flags se ignorara en algunos sistemas de archivos.

Lanza un evento de auditoría `os.setxattr` con argumentos `path`, `attribute`, `value`, `flags`.

Distinto en la versión 3.6: Acepta un path-like object para *path*y *attribute*.

os.XATTR_SIZE_MAX

El tamaño máximo que puede tener el valor de un atributo extendido. Actualmente, esto es 64 KiB en Linux.

os.XATTR_CREATE

Este es un valor posible para el argumento flags en `setxattr()`. Indica que la operación debe crear un atributo.

os.XATTR_REPLACE

Este es un valor posible para el argumento flags en `setxattr()`. Indica que la operación debe reemplazar un atributo existente.

Gestión de proceso

Estas funciones pueden usarse para crear y administrar procesos.

Las varias funciones `exec *` toman una lista de argumentos para el nuevo programa cargado en el proceso. En cada caso, el primero de estos argumentos se pasa al nuevo programa como su propio nombre en lugar de como un argumento que un usuario puede haber escrito en una línea de comando. Para el programador C, este es el `argv[0]` pasado a un programa `main()`. Por ejemplo, `os.execv('/bin/echo', ['foo', 'bar'])` solo imprimirá `bar` en la salida estándar; `foo` parecerá ignorado.

os.abort()

Genere una señal **`SIGABRT`** para el proceso actual. En Unix, el comportamiento predeterminado es producir un volcado de núcleo; en Windows, el proceso retorna inmediatamente un código de salida de `3`. Tenga en cuenta que llamar a esta función no llamará al controlador de señal Python registrado para **`SIGABRT`** con `signal.signal()`.

os.add_dll_directory(*path*)

Agregue una ruta a la ruta de búsqueda de DLL.

Esta ruta de búsqueda se utiliza al resolver dependencias para módulos de extensión importados (el propio módulo se resuelve mediante `sys.path`) y también mediante `ctypes`.

Elimina el directorio llamando a **close()** en el objeto retornado o utilizándolo en una `with` instrucción.

Consulte la documentación de Microsoft [https://msdn.microsoft.com/44228cf2-6306-466c-8f16-f513cd3ba8b5] para obtener más información sobre cómo se cargan las DLL.

Lanza un evento de auditoría `os.add_dll_directory` con el argumento `path`.

Availability: Windows.

Added in version 3.8: Las versiones anteriores de CPython resolverían las DLL utilizando el comportamiento predeterminado para el proceso actual. Esto condujo a inconsistencias, como solo a veces buscar **`PATH`** o el directorio de trabajo actual, y las funciones del sistema operativo como `AddDllDirectory` no tienen ningún efecto.

En 3.8, las dos formas principales en que se cargan las DLL ahora anulan explícitamente el comportamiento de todo el proceso para garantizar la coherencia. Ver el notas de portabilidad para obtener información sobre la actualización de bibliotecas.

os.execl(*path*, *arg0*, *arg1*, *...*)
os.execle(*path*, *arg0*, *arg1*, *...*, *env*)
os.execlp(*file*, *arg0*, *arg1*, *...*)
os.execlpe(*file*, *arg0*, *arg1*, *...*, *env*)
os.execv(*path*, *args*)
os.execve(*path*, *args*, *env*)
os.execvp(*file*, *args*)
os.execvpe(*file*, *args*, *env*)

Todas estas funciones ejecutan un nuevo programa, reemplazando el proceso actual; No vuelven. En Unix, el nuevo ejecutable se carga en el proceso actual y tendrá la misma identificación de proceso que la persona que llama. Los errores se informarán como excepciones **`OSError`**.

The current process is replaced immediately. Open file objects and descriptors are not flushed, so if there may be data buffered on these open files, you should flush them using **`flush()`** or **`os.fsync()`** before calling an **`exec*`** function.

Las variantes «l» y «v» de las funciones **`exec*`** difieren en la forma en que se pasan los argumentos de la línea de comandos. Las variantes «l» son quizás las más fáciles de usar si el número de parámetros es fijo cuando se escribe el código; los parámetros individuales simplemente se convierten en parámetros adicionales para las funciones `execl*()`. Las variantes «v» son buenas cuando el número de parámetros es variable, y los argumentos se pasan en una lista o tupla como el parámetro *args*. En cualquier caso, los argumentos del proceso secundario deben comenzar con el nombre del comando que se está ejecutando, pero esto no se aplica.

Las variantes que incluyen una «p» cerca del final (`execlp()`, `execlpe()`, `execvp()` y `execvpe()`) utilizarán la variable de entorno `PATH` para localizar el programa *file*. Cuando se reemplaza el entorno (utilizando una de las variantes `exec*e`, que se analizan en el párrafo siguiente), el nuevo entorno se utiliza como fuente de la variable `PATH`. Las otras variantes, `execl()`, `execle()`, `execv()` y `execve()`, no utilizarán la variable `PATH` para localizar el ejecutable; *path* debe contener una ruta absoluta o relativa adecuada. Las rutas relativas deben incluir al menos una barra, incluso en Windows, ya que los nombres simples no se resolverán.

Para `execle()`, `execlpe()`, `execve()` y `execvpe()` (tenga en cuenta que todo esto termina en «e»), el parámetro *env* debe ser un mapeo que se utiliza para definir las variables de entorno para el nuevo proceso (se utilizan en lugar del entorno del proceso actual); las funciones `execl()`, `execlp()`, `execv()` y `execvp()` hacen que el nuevo proceso herede el entorno del proceso actual.

Para `execve()` en algunas plataformas, *path* también puede especificarse como un descriptor de archivo abierto. Es posible que esta funcionalidad no sea compatible con su plataforma; puede verificar si está disponible o no usando `os.supports_fd`. Si no está disponible, su uso lanzará un `NotImplementedError`.

Lanza un evento de auditoría `os.exec` con argumentos `ruta`, `args`, `env`.

Availability: Unix, Windows, not WASI, not Android, not iOS.

Distinto en la versión 3.3: Se agregó soporte para especificar *path* como un descriptor de archivo abierto para `execve()`.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

os._exit(*n*)

Salga del proceso con el estado *n*, sin llamar a los controladores de limpieza, vaciar los buffers stdio, etc.

Nota

La forma estándar de salir es `sys.exit(n)`. Normalmente, `_exit()` solo se debe utilizar en el proceso secundario después de `fork()`.

Los siguientes códigos de salida están definidos y se pueden usar con `_exit()`, aunque no son obligatorios. Por lo general, se usan para programas del sistema escritos en Python, como el programa de entrega de comandos externos de un servidor de correo.

Nota

Es posible que algunos de estos no estén disponibles en todas las plataformas Unix, ya que hay alguna variación. Estas constantes se definen donde están definidas por la plataforma subyacente.

os.EX_OK

Código de salida que significa que no ocurrió ningún error. Puede tomarse del valor definido de `EXIT_SUCCESS` en algunas plataformas. Generalmente tiene un valor de cero.

Availability: Unix, Windows.

os.EX_USAGE

Código de salida que significa que el comando se usó incorrectamente, como cuando se da un número incorrecto de argumentos.

Availability: Unix, not WASI.

os.EX_DATAERR

Código de salida que significa que los datos de entrada eran incorrectos.

Availability: Unix, not WASI.

os.EX_NOINPUT

Código de salida que significa que no existía un archivo de entrada o que no era legible.

Availability: Unix, not WASI.

os.EX_NOUSER

Código de salida que significa que un usuario especificado no existía.

Availability: Unix, not WASI.

os.EX_NOHOST

Código de salida que significa que no existía un host especificado.

Availability: Unix, not WASI.

os.EX_UNAVAILABLE

Código de salida que significa que un servicio requerido no está disponible.

Availability: Unix, not WASI.

os.EX_SOFTWARE

Código de salida que significa que se detectó un error interno de software.

Availability: Unix, not WASI.

os.EX_OSERR

Código de salida que significa que se detectó un error del sistema operativo, como la imposibilidad de bifurcar o crear una tubería.

Availability: Unix, not WASI.

os.EX_OSFILE

Código de salida que significa que algunos archivos del sistema no existían, no podían abrirse o tenían algún otro tipo de error.

Availability: Unix, not WASI.

os.EX_CANTCREAT

Código de salida que significa que no se pudo crear un archivo de salida especificado por el usuario.

Availability: Unix, not WASI.

os.EX_IOERR

Código de salida que significa que se produjo un error al realizar E / S en algún archivo.

Availability: Unix, not WASI.

os.EX_TEMPFAIL

Código de salida que significa que ocurrió una falla temporal. Esto indica algo que puede no ser realmente un error, como una conexión de red que no se pudo realizar durante una operación recuperable.

Availability: Unix, not WASI.

os.EX_PROTOCOL

Código de salida que significa que un intercambio de protocolo fue ilegal, inválido o no se entendió.

Availability: Unix, not WASI.

os.EX_NOPERM

Código de salida que significa que no había permisos suficientes para realizar la operación (pero no para problemas del sistema de archivos).

Availability: Unix, not WASI.

os.EX_CONFIG

Código de salida que significa que se produjo algún tipo de error de configuración.

Availability: Unix, not WASI.

os.EX_NOTFOUND

Código de salida que significa algo así como «no se encontró una entrada».

Availability: Unix, not WASI.

os.fork()

Bifurcar un proceso hijo. Retorna `0` en el niño y la identificación del proceso del niño en el padre. Si se produce un error se genera **`OSError`**.

Tenga en cuenta que algunas plataformas que incluyen FreeBSD &lt;= 6.3 y Cygwin tienen problemas conocidos al usar `fork()` desde un hilo.

Lanza un evento de auditoría `os.fork` sin argumentos.

Advertencia

Si utiliza sockets TLS en una aplicación que llama a `fork()`, consulte la advertencia en la documentación de **`ssl`**.

Advertencia

En macOS, el uso de esta función no es seguro cuando se combina con el uso de API de sistema de nivel superior, incluido el uso de **`urllib.request`**.

Distinto en la versión 3.8: Llamar a `fork()` en un subinterpretador ya no es compatible (**`RuntimeError`** está activado).

Distinto en la versión 3.12: Si Python puede detectar que su proceso tiene varios subprocesos, **`os.fork()`** ahora genera un **`DeprecationWarning`**.

Elegimos mostrar esto como una advertencia, cuando sea detectable, para informar mejor a los desarrolladores sobre un problema de diseño que la plataforma POSIX señala específicamente como no compatible. Incluso en el código en el que *appears* funciona, nunca ha sido seguro mezclar subprocesos con **`os.fork()`** en plataformas POSIX. El entorno de ejecución de CPython en sí siempre ha realizado llamadas a la API que no son seguras para su uso en el proceso secundario cuando existían subprocesos en el primario (como `malloc` y `free`).

Los usuarios de macOS o los usuarios de implementaciones de libc o malloc distintas de las que se encuentran habitualmente en glibc hasta la fecha se encuentran entre los que tienen más probabilidades de experimentar bloqueos al ejecutar dicho código.

Consulte this discussion on fork being incompatible with threads [https://discuss.python.org/t/33555] para obtener detalles técnicos sobre el motivo por el cual estamos informando a los desarrolladores de este problema de compatibilidad de plataformas de larga data.

Availability: POSIX, not WASI, not Android, not iOS.

os.forkpty()

Bifurca un proceso hijo, usando un nuevo pseudo-terminal como terminal de control del niño. Retorna un par de `(pid, fd)`, donde *pid* es `0` en el elemento secundario, la identificación del proceso del elemento secundario nuevo en el elemento primario y *fd* es el descriptor de archivo del final maestro de El pseudo-terminal. Para un enfoque más portátil, use el módulo **`pty`**. Si se produce un error se genera **`OSError`**.

Lanza un evento de auditoría `os.forkpty` sin argumentos.

Advertencia

En macOS, el uso de esta función no es seguro cuando se combina con el uso de API de sistema de nivel superior, incluido el uso de **`urllib.request`**.

Distinto en la versión 3.8: Llamar a `forkpty()` en un subinterpretador ya no es compatible (**`RuntimeError`** está activado).

Distinto en la versión 3.12: Si Python puede detectar que su proceso tiene varios subprocesos, esto ahora genera un error **`DeprecationWarning`**. Consulte la explicación más extensa sobre **`os.fork()`**.

Availability: Unix, not WASI, not Android, not iOS.

os.kill(*pid*, *sig*, */*)

Enviar señal *sig* al proceso *pid*. Las constantes para las señales específicas disponibles en la plataforma host se definen en el módulo **`signal`**.

Windows: The **`signal.CTRL_C_EVENT`** and **`signal.CTRL_BREAK_EVENT`** signals are special signals which can only be sent to console processes which share a common console window, e.g., some subprocesses. Any other value for *sig* will cause the process to be unconditionally killed by the TerminateProcess API, and the exit code will be set to *sig*.

Ver también **`signal.pthread_kill()`**.

Lanza un evento de auditoría `os.kill` con argumentos `pid`, `sig`.

Availability: Unix, Windows, not WASI, not iOS.

Distinto en la versión 3.2: Se agregó soporte para Windows.

os.killpg(*pgid*, *sig*, */*)

Envíe la señal *sig* al grupo de procesos *pgid*.

Lanza un evento de auditoría `os.killpg` con argumentos `pgid`, `sig`.

Availability: Unix, not WASI, not iOS.

os.nice(*increment*, */*)

Agregue *increment* a la «simpatía» del proceso. Retorna la nueva amabilidad.

Availability: Unix, not WASI.

os.pidfd_open(*pid*, *flags=0*)

Devuelve un descriptor de archivo que hace referencia al proceso *pid* con *flags* configurado. Este descriptor se puede utilizar para realizar la gestión de procesos sin carreras ni señales.

Ver la página manual de *pidfd_open(2)* *[https://manpages.debian.org/pidfd_open(2)]* para mas detalles.

Availability: Linux > = 5.3, Android > = **`build-time`** API level 31

Added in version 3.9.

os.PIDFD_NONBLOCK

Esta bandera indica que el descriptor de archivo no será bloqueante. Si el proceso al que hace referencia el descriptor de archivo aún no ha finalizado, entonces un intento de esperar en el descriptor de archivo utilizando *waitid(2)* *[https://manpages.debian.org/waitid(2)]* devolverá inmediatamente el error **`EAGAIN`** en lugar de bloquearse.

Availability: Linux >= 5.10

Added in version 3.12.

os.plock(*op*, */*)

Bloquee segmentos del programa en la memoria. El valor de *op* (definido en `<sys/lock.h>`) determina qué segmentos están bloqueados.

Availability: Unix, not WASI, not iOS.

os.popen(*cmd*, *mode='r'*, *buffering=-1*)

Abre una tubería hacia o desde el comando *cmd*. El valor retornado es un objeto de archivo abierto conectado a la canalización, que se puede leer o escribir dependiendo de si *mode* es `'r'` (predeterminado) o `'w'`. El argumento *buffering* tiene el mismo significado que el argumento correspondiente a la función integrada **`open()`**. El objeto de archivo retornado lee o escribe cadenas de texto en lugar de bytes.

El método `close` retorna **`None`** si el subproceso salió correctamente, o el código de retorno del subproceso si hubo un error. En los sistemas POSIX, si el código de retorno es positivo, representa el valor de retorno del proceso desplazado a la izquierda en un byte. Si el código de retorno es negativo, el proceso fue terminado por la señal dada por el valor negado del código de retorno. (Por ejemplo, el valor de retorno podría ser `- signal.SIGKILL` si se eliminó el subproceso). En los sistemas Windows, el valor de retorno contiene el código de retorno entero firmado del proceso secundario.

En Unix, **`waitstatus_to_exitcode()`** puede ser usado para convertir el resultado del método `close` (estado de salida) en un código de salida si es que no es `None`. En Windows, el resultado del método `close` es directamente el código de salida (o `None`).

Esto se implementa usando **`subprocess.Popen`**; consulte la documentación de esa clase para obtener formas más potentes de administrar y comunicarse con subprocesos.

Availability: not WASI, not Android, not iOS.

Nota

El Python UTF-8 Mode afecta las codificaciones utilizadas para *cmd* y el contenido de la canalización.

`popen()` es un contenedor simple alrededor de **`subprocess.Popen`**. Use **`subprocess.Popen`** o **`subprocess.run()`** para controlar opciones como codificaciones.

Soft deprecated since version 3.14: The **`subprocess`** module is recommended instead.

os.posix_spawn(*path*, *argv*, *env*, ***, *file_actions=None*, *setpgroup=None*, *resetids=False*, *setsid=False*, *setsigmask=()*, *setsigdef=()*, *scheduler=None*)

Envuelve la API de la biblioteca C **`posix_spawn()`** para su uso desde Python.

La mayoría de los usuarios deberían usar **`subprocess.run()`** en lugar de **`posix_spawn()`**.

Los argumentos posicionales *path*, *args* y *env* son similares a **`execve()`**. Se permite que *env* sea `None`, en cuyo caso se utiliza el entorno del proceso actual.

El parámetro *path* es la ruta al archivo ejecutable. La *path* debe contener un directorio. Utilice **`posix_spawnp()`** para pasar un archivo ejecutable sin directorio.

El argumento *file_actions* puede ser una secuencia de tuplas que describen acciones para tomar descriptores de archivo específicos en el proceso secundario entre los pasos de implementación de la biblioteca C **`fork()`** y **`exec()`**. El primer elemento de cada tupla debe ser uno de los tres indicadores de tipo que se enumeran a continuación y que describen los elementos de tupla restantes:

os.POSIX_SPAWN_OPEN

> (`os.POSIX_SPAWN_OPEN`, *fd*, *path*, *flags*, *mode*)
>
> Realiza `os.dup2(os.open(path, flags, mode), fd)`.

os.POSIX_SPAWN_CLOSE

> (`os.POSIX_SPAWN_CLOSE`, *fd*)
>
> Realiza `os.close(fd)`.

os.POSIX_SPAWN_DUP2

> (`os.POSIX_SPAWN_DUP2`, *fd*, *new_fd*)
>
> Realiza `os.dup2(fd, new_fd)`.

os.POSIX_SPAWN_CLOSEFROM

> (`os.POSIX_SPAWN_CLOSEFROM`, *fd*)
>
> Ejecuta `os.closerange(fd, INF)`.

Estas tuplas corresponden a las llamadas API de la biblioteca C **`posix_spawn_file_actions_addopen()`**, **`posix_spawn_file_actions_addclose()`**, **`posix_spawn_file_actions_adddup2()`** y **`posix_spawn_file_actions_addclosefrom_np()`** utilizadas para preparar la propia llamada **`posix_spawn()`**.

El argumento *setpgroup* establecerá el grupo de procesos del hijo en el valor especificado. Si el valor especificado es 0, el ID del grupo de procesos del hijo será el mismo que el ID de su proceso. Si el valor de *setpgroup* no está establecido, el hijo heredará el ID del grupo de procesos del padre. Este argumento corresponde al indicador **`POSIX_SPAWN_SETPGROUP`** de la biblioteca C.

Si el argumento *resetids* es `True`, restablecerá el UID y GID efectivos del hijo al UID y GID reales del proceso padre. Si el argumento es `False`, el hijo conserva el UID y GID efectivos del padre. En cualquier caso, si los bits de permiso set-user-ID y set-group-ID están habilitados en el archivo ejecutable, su efecto anulará la configuración del UID y GID efectivos. Este argumento corresponde al indicador **`POSIX_SPAWN_RESETIDS`** de la biblioteca C.

Si el argumento *setsid* es `True`, se creará un nuevo ID de sesión para `posix_spawn`. *setsid* requiere el indicador **`POSIX_SPAWN_SETSID`** o **`POSIX_SPAWN_SETSID_NP`**. De lo contrario, se activa **`NotImplementedError`**.

El argumento *setsigmask* establecerá la máscara de señal en el conjunto de señales especificado. Si no se utiliza el parámetro, el elemento secundario hereda la máscara de señal del elemento principal. Este argumento corresponde al indicador **`POSIX_SPAWN_SETSIGMASK`** de la biblioteca C.

El argumento *sigdef* restablecerá la disposición de todas las señales en el conjunto especificado. Este argumento corresponde al indicador **`POSIX_SPAWN_SETSIGDEF`** de la biblioteca C.

El argumento *scheduler* debe ser una tupla que contenga la política del programador (opcional) y una instancia de **`sched_param`** con los parámetros del programador. Un valor de `None` en lugar de la política del programador indica que no se proporciona. Este argumento es una combinación de los indicadores **`POSIX_SPAWN_SETSCHEDPARAM`** y **`POSIX_SPAWN_SETSCHEDULER`** de la biblioteca C.

Lanza un evento de auditoría `os.posix_spawn` con argumentos `ruta`, `argv`, `env`.

Added in version 3.8.

Distinto en la versión 3.13: El parámetro *env* acepta `None`. `os.POSIX_SPAWN_CLOSEFROM` está disponible en plataformas donde existe **`posix_spawn_file_actions_addclosefrom_np()`**.

Availability: Unix, not WASI, not Android, not iOS.

os.posix_spawnp(*path*, *argv*, *env*, ***, *file_actions=None*, *setpgroup=None*, *resetids=False*, *setsid=False*, *setsigmask=()*, *setsigdef=()*, *scheduler=None*)

Envuelve la API de la biblioteca C **`posix_spawnp()`** para su uso desde Python.

Similar a **`posix_spawn()`** excepto que el sistema busca el archivo *executable* en la lista de directorios especificada por la variable de entorno **`PATH`** (de la misma manera que para `execvp(3)`)

Lanza un evento de auditoría `os.posix_spawn` con argumentos `ruta`, `argv`, `env`.

Added in version 3.8.

Availability: POSIX, not WASI, not Android, not iOS.

Consulte la documentación de **`posix_spawn()`**.

os.register_at_fork(***, *before=None*, *after_in_parent=None*, *after_in_child=None*)

Registra los invocables que se ejecutarán cuando se bifurca un nuevo proceso secundario utilizando **`os.fork()`** o API de clonación de procesos similares. Los parámetros son opcionales y solo de palabras clave. Cada uno especifica un punto de llamada diferente.

- *before* es una función llamada antes de bifurcar un proceso hijo.
- *after_in_parent* es una función llamada desde el proceso padre después de bifurcar un proceso hijo.
- *after_in_child* es una función llamada desde el proceso

hijo.

Estas llamadas solo se realizan si se espera que el control regrese al intérprete de Python. Un lanzamiento típico `subprocess` no los activará ya que el niño no va a volver a ingresar al intérprete.

Las funciones registradas para su ejecución antes de la bifurcación se invocan en orden de registro inverso. Las funciones registradas para la ejecución después de la bifurcación (ya sea en el padre o en el hijo) se invocan en orden de registro.

Tenga en cuenta que las llamadas `fork()` realizadas por código C de terceros no pueden llamar a esas funciones, a menos que explícitamente llame a `PyOS_BeforeFork()`, `PyOS_AfterFork_Parent()` y `PyOS_AfterFork_Child()`.

No hay forma de cancelar el registro de una función.

Availability: Unix, not WASI, not Android, not iOS.

Added in version 3.7.

os.spawnl(*mode, path, ...*)
os.spawnle(*mode, path, ..., env*)
os.spawnlp(*mode, file, ...*)
os.spawnlpe(*mode, file, ..., env*)
os.spawnv(*mode, path, args*)
os.spawnve(*mode, path, args, env*)
os.spawnvp(*mode, file, args*)
os.spawnvpe(*mode, file, args, env*)

Ejecute el programa *path* en un nuevo proceso.

(Tenga en cuenta que el módulo `subprocess` proporciona funciones más potentes para generar nuevos procesos y recuperar sus resultados; es preferible usar ese módulo que usar estas funciones. Compruebe especialmente la sección

Replacing Older Functions with the subprocess Module.)

Si *mode* es `P_NOWAIT`, esta función retorna la identificación del proceso del nuevo proceso; if *mode* is `P_WAIT`, retorna el código de salida del proceso si sale normalmente, o `-signal`, donde *signal* es la señal que mató el proceso. En Windows, la identificación del proceso en realidad será el identificador del proceso, por lo que se puede usar con la función `waitpid()`.

Nota sobre VxWorks, esta función no retorna `-signal` cuando se cierra el nuevo proceso. En su lugar, genera una excepción OSError.

Las variantes «l» y «v» de las funciones `spawn*` difieren en la forma en que se pasan los argumentos de la línea de comandos. Las variantes «l» son quizás las más fáciles de usar si el número de parámetros es fijo cuando se escribe el código; los parámetros individuales simplemente se convierten en parámetros adicionales para las funciones `spawnl*()`. Las variantes «v» son buenas cuando el número de parámetros es variable, y los argumentos se pasan en una lista o tupla como el parámetro *args*. En cualquier caso, los argumentos del proceso hijo deben comenzar con el nombre del comando que se está ejecutando.

Las variantes que incluyen una segunda «p» cerca del final (`spawnlp()`, `spawnlpe()`, `spawnvp()`, y `spawnvpe()`) usarán `PATH` variable de entorno para ubicar el programa *file*. Cuando se reemplaza el entorno (usando uno de los siguientes `spawn*e` variantes, discutidas en el siguiente párrafo), el nuevo entorno se utiliza como fuente de la variable `PATH`. Las otras variantes, `spawnl()`, `spawnle()`, `spawnv()`, y `spawnve()`, no utilizarán la variable `PATH` para localizar el ejecutable; *path* debe contener una ruta absoluta o relativa apropiada.

Para `spawnle()`, `spawnlpe()`, `spawnve()`, y `spawnvpe()` (tenga en cuenta que todo esto termina en «e»), el parámetro *env* debe ser un mapeo que se utiliza para definir las variables de entorno para el nuevo proceso (se utilizan en lugar del entorno del proceso actual); las funciones `spawnl()`, `spawnlp()`, `spawnv()` y `spawnvp()` hacen que el nuevo proceso herede el entorno del proceso actual. Tenga en cuenta que las claves y los valores en el diccionario *env* deben ser cadenas; Las teclas o valores no válidos harán que la función falle, con un valor de retorno de `127`.

Como ejemplo, las siguientes llamadas a `spawnlp()` y `spawnvpe()` son equivalentes:

```
import os
os.spawnlp(os.P_WAIT, 'cp', 'cp', 'index.html', '/d

L = ['cp', 'index.html', '/dev/null']
os.spawnvpe(os.P_WAIT, 'cp', L, os.environ)
```

Lanza un evento de auditoría `os.spawn` con argumentos `mode`, `path`, `args`, `env`.

Availability: Unix, Windows, not WASI, not Android, not iOS.

`spawnlp()`, `spawnlpe()`, `spawnvp()` y `spawnvpe()` no están disponibles en Windows. `spawnle()` y `spawnve()` no son seguros para subprocesos en Windows; le recomendamos que utilice el módulo `subprocess` en su lugar.

Distinto en la versión 3.6: Acepta un objeto tipo ruta.

Soft deprecated since version 3.14: The `subprocess` module is recommended instead.

os.P_NOWAIT
os.P_NOWAITO

Valores posibles para el parámetro *mode* de la familia de funciones `spawn*`. Si se proporciona cualquiera de estos valores, las funciones **`spawn*`** retornarán tan pronto como se haya creado el nuevo proceso, con el identificador del proceso como valor de retorno.

Availability: Unix, Windows.

os.P_WAIT

Valor posible para el parámetro *mode* de la familia de funciones `spawn*`. Si se proporciona como *mode*, las funciones **`spawn*`** no regresarán hasta que el nuevo proceso se haya ejecutado por completo y devolverán el código de salida del proceso cuya ejecución se realizó correctamente, o `-signal` si una señal finaliza el proceso.

Availability: Unix, Windows.

os.P_DETACH
os.P_OVERLAY

Valores posibles para el parámetro *mode* para `spawn*` familia de funciones. Estos son menos portátiles que los enumerados anteriormente. `P_DETACH` es similar a `P_NOWAIT`, pero el nuevo proceso se desconecta de la consola del proceso de llamada. Si se usa `P_OVERLAY`, el proceso actual será reemplazado; la función **`spawn*`** no volverá.

Availability: Windows.

os.startfile(*path*[, *operation*][, *arguments*][, *cwd*][, *show_cmd*])

Inicie un archivo con su aplicación asociada.

Cuando no se especifica *operation*, esto actúa como hacer doble clic en el archivo en el Explorador de Windows o dar el nombre del archivo como argumento al comando **start** desde el shell de comandos interactivo: el archivo se abre con cualquier aplicación (si hay alguna) con la que esté asociada su extensión.

Cuando se proporciona otro *operation*, debe ser un «verbo de comando» que especifique qué se debe hacer con el archivo. Los verbos comunes documentados por Microsoft son `'open'`, `'print'` y `'edit'` (para usar en archivos), así como `'explore'` y `'find'` (para usar en directorios).

Al iniciar una aplicación, especifique *arguments* para que se pase como una sola cadena. Es posible que este argumento no tenga ningún efecto cuando se utiliza esta función para iniciar un documento.

El directorio de trabajo predeterminado se hereda, pero el argumento *cwd* puede anularlo. Este debería ser un camino absoluto. Un *path* relativo se resolverá contra este argumento.

Utilice *show_cmd* para anular el estilo de ventana predeterminado. El efecto que esto tenga dependerá de la aplicación que se esté ejecutando. Los valores son números enteros, tal como lo admite la función **`ShellExecute()`** de Win32.

`startfile()` retorna tan pronto como se inicia la aplicación asociada. No hay opción para esperar a que se cierre la aplicación y no hay forma de recuperar el estado de salida de la aplicación. El parámetro *path* es relativo al directorio actual o *cwd*. Si desea utilizar una ruta absoluta, asegúrese de que el primer carácter no sea una barra (`'/'`) Utilice **`pathlib`** o la función **`os.path.normpath()`** para asegurarse de que las rutas estén codificadas correctamente para Win32.

Para reducir la sobrecarga de inicio del intérprete, la función Win32 **`ShellExecute()`** no se resuelve hasta que se la llama por primera vez. Si no se puede resolver la función, se generará **`NotImplementedError`**.

Lanza un evento de auditoría `os.startfile` con argumentos `path`, `operation`.

Lanza un evento de auditoria `os.startfile/2` con argumentos `path`, `operation`, `arguments`, `cwd`, `show_cmd`.

Availability: Windows.

Distinto en la versión 3.10: Se agregaron los argumentos *arguments*, *cwd* y *show_cmd*, y el evento de auditoría `os.startfile/2`.

os.system(*command*)

Ejecute el comando (una cadena) en una subcapa. Esto se implementa llamando a la función C estándar `system()` y tiene las mismas limitaciones. Los cambios en `sys.stdin`, etc. no se reflejan en el entorno del comando ejecutado. Si *command* genera alguna salida, se enviará al flujo de salida estándar del intérprete. El estándar C no especifica el significado del valor de retorno de la función C, por lo que el valor de retorno de la función de Python depende del sistema.

En Unix, el valor de retorno es el estado de salida del proceso codificado en el formato especificado para `wait()`.

En Windows, el valor de retorno es el que retorna el shell del sistema después de ejecutar *command*. El shell viene dado por la variable de entorno de Windows `COMSPEC`: generalmente es **cmd.exe**, que retorna el estado de salida de la ejecución del comando; En sistemas que utilizan un shell no nativo, consulte la documentación del shell.

The `subprocess` module provides more powerful facilities for spawning new processes and retrieving their results; using that module is recommended to using this function. See the Replacing Older Functions with the subprocess Module section in the `subprocess` documentation for some helpful recipes.

En Unix, `waitstatus_to_exitcode()` puede ser usado para convertir el resultado (estado de salida) en un código de salida. En Windows, el resultado es directamente el código de salida.

Lanza un evento de auditoría `os.system` con argumento `command`.

Availability: Unix, Windows, not WASI, not Android, not iOS.

os.times()

Retorna los tiempos de proceso globales actuales. El valor de retorno es un objeto con cinco atributos:

- **`user`** - tiempo de usuario
- **`system`** - tiempo de sistema
- **`children_user`** - tiempo de usuario de todos los procesos secundarios
- **`children_system`** - hora del sistema de todos los procesos secundarios
- **`elapsed`**: tiempo real transcurrido desde un punto fijo en el pasado

Por compatibilidad con versiones anteriores, este objeto también se comporta como una tupla de cinco que contiene **`user`**, **`system`**, **`children_user`**, **`children_system`** y **`elapsed`** en ese orden.

Consulte las páginas del manual de Unix *times(2) [https://manpages.debian.org/times(2)]* y times(3) [https://man.freebsd.org/cgi/man.cgi?time(3)] en Unix o the GetProcessTimes MSDN [https://docs.microsoft.com/windows/win32/api/processthreadsapi/nf-processthreadsapi-getprocesstimes] en Windows. En Windows, solo se conocen **`user`** y **`system`**; los demás atributos son cero.

Availability: Unix, Windows.

Distinto en la versión 3.3: El tipo de objeto retornado cambió de una tupla a un objeto tipo tupla con atributos con nombre.

os.wait()

Espere a que se complete un proceso secundario y retorna una tupla que contenga su indicación de estado pid y de salida: un número de 16 bits, cuyo byte bajo es el número de señal que mató el proceso y cuyo byte alto es el estado de salida (si la señal el número es cero); el bit alto del byte bajo se establece si se produjo un archivo central.

Si no hay niños a los que esperar, se genera **`ChildProcessError`**.

`waitstatus_to_exitcode()` puede ser usado para convertir el estado de salida en el código de salida.

Availability: Unix, not WASI, not Android, not iOS.

Ver también

Las demás funciones `wait*()` documentadas a continuación se pueden utilizar para esperar a que se complete un proceso secundario específico y tener más opciones. `waitpid()` es la única que también está disponible en Windows.

os.waitid(*idtype*, *id*, *options*, */*)

Esperar a que se complete un proceso secundario.

idtype puede ser `P_PID`, `P_PGID`, `P_ALL` o (en Linux) `P_PIDFD`. La interpretación de *id* depende de ello; consulte sus descripciones individuales.

options es una combinación OR de indicadores. Se requiere al menos uno de `WEXITED`, `WSTOPPED` o `WCONTINUED`; `WNOHANG` y `WNOWAIT` son indicadores opcionales adicionales.

El valor de retorno es un objeto que representa los datos contenidos en la estructura `siginfo_t` con los siguientes atributos:

- `si_pid` (identificación del proceso)
- `si_uid` (ID de usuario real del niño)
- `si_signo` (siempre `SIGCHLD`)
- `si_status` (el estado de salida o el número de señal, según `si_code`)
- `si_code` (consulte `CLD_EXITED` para conocer los posibles valores)

Si se especifica `WNOHANG` y no hay elementos secundarios coincidentes en el estado solicitado, se devuelve `None`. De lo contrario, si no hay elementos secundarios coincidentes que se puedan esperar, se genera `ChildProcessError`.

Availability: Unix, not WASI, not Android, not iOS.

Added in version 3.3.

Distinto en la versión 3.13: Esta función ahora también está disponible en macOS.

os.waitpid(*pid*, *options*, /)

Los detalles de esta función difieren en Unix y Windows.

En Unix: espere a que se complete un proceso secundario dado por la identificación del proceso *pid*, y retorna una tupla que contenga su identificación del proceso y la indicación del estado de salida (codificado como para **`wait()`**). La semántica de la llamada se ve afectada por el valor del número entero *options*, que debe ser `0` para el funcionamiento normal.

Si *pid* es mayor que `0`, **`waitpid()`** solicita información de estado para ese proceso específico. Si *pid* es `0`, la solicitud es para el estado de cualquier hijo en el grupo de procesos del proceso actual. Si *pid* es `-1`, la solicitud corresponde a cualquier elemento secundario del proceso actual. Si *pid* es menor que `-1`, se solicita el estado de cualquier proceso en el grupo de procesos `-pid` (el valor absoluto de *pid*).

options es una combinación OR de indicadores. Si contiene **`WNOHANG`** y no hay hijos coincidentes en el estado solicitado, se devuelve `(0, 0)`. De lo contrario, si no hay hijos coincidentes que se puedan esperar, se genera **`ChildProcessError`**. Otras opciones que se pueden utilizar son **`WUNTRACED`** y **`WCONTINUED`**.

En Windows: espere a que se complete un proceso dado por el identificador de proceso *pid*, y retorna una tupla que contiene *pid*, y su estado de salida se desplazó a la izquierda en 8 bits (el desplazamiento facilita el uso de la función en la plataforma). A *pid* menor o igual que `0` no tiene un significado especial en Windows y genera una excepción. El valor de entero *options* no tiene ningún efecto. *pid* puede referirse a cualquier proceso cuya identificación sea conocida, no necesariamente un proceso hijo. Las funciones `spawn*` llamadas con `P_NOWAIT` retornan manejadores de proceso adecuados.

`waitstatus_to_exitcode()` puede ser usado para convertir el estado de salida en el código de salida.

Availability: Unix, Windows, not WASI, not Android, not iOS.

Distinto en la versión 3.5: Si la llamada al sistema se interrumpe y el controlador de señal no genera una excepción, la función vuelve a intentar la llamada del sistema en lugar de generar una excepción `InterruptedError` (ver **PEP 475** [https://peps.python.org/pep-0475/] para la justificación).

os.wait3(*options*)

Similar a `waitpid()`, excepto que no se proporciona ningún argumento de identificación de proceso y se devuelve una tupla de 3 elementos que contiene la identificación del proceso del hijo, la indicación del estado de salida y la información de uso de recursos. Consulte `resource.getrusage()` para obtener detalles sobre la información de uso de recursos. El argumento *options* es el mismo que se proporciona a `waitpid()` y `wait4()`.

`waitstatus_to_exitcode()` puede ser usado para convertir el estado de salida al código de salida.

Availability: Unix, not WASI, not Android, not iOS.

os.wait4(*pid*, *options*)

Similar a `waitpid()`, excepto que se devuelve una tupla de 3 elementos que contiene el identificador del proceso secundario, la indicación del estado de salida y la información de uso de recursos. Consulte `resource.getrusage()` para obtener detalles sobre la información de uso de recursos. Los argumentos de `wait4()` son los mismos que los proporcionados a `waitpid()`.

`waitstatus_to_exitcode()` puede ser usado para convertir el estado de salida al código de salida.

Availability: Unix, not WASI, not Android, not iOS.

os.P_PID
os.P_PGID
os.P_ALL
os.P_PIDFD

Estos son los valores posibles para *idtype* en `waitid()`. Afectan a la interpretación de *id*:

- **`P_PID`**: espera al niño cuyo PID es *id*.
- **`P_PGID`**: espera a cualquier niño cuyo ID de grupo de progreso sea *id*.
- **`P_ALL`** - espera a cualquier hijo; *id* se ignora.
- **`P_PIDFD`**: espera al niño identificado por el descriptor de archivo *id* (un descriptor de archivo de proceso creado con `pidfd_open()`).

Availability: Unix, not WASI, not Android, not iOS.

Nota

`P_PIDFD` solo está disponible en Linux > = 5.4.

Added in version 3.3.

Added in version 3.9: La constante **`P_PIDFD`**.

os.WCONTINUED

Esta bandera *options* para `waitpid()`, `wait3()`, `wait4()` y `waitid()` hace que se informe sobre los procesos secundarios si han continuado desde una detención del control de trabajo desde que se informaron por última vez.

Availability: Unix, not WASI, not Android, not iOS.

os.WEXITED

Esta bandera *options* para `waitid()` hace que se informen los procesos secundarios que han finalizado.

Las demás funciones `wait*` siempre informan sobre los hijos que han finalizado, por lo que esta opción no está disponible para ellos.

Availability: Unix, not WASI, not Android, not iOS.

Added in version 3.3.

os.WSTOPPED

Esta bandera *options* para **`waitid()`** hace que se informen los procesos secundarios que se han detenido por la entrega de una señal.

Esta opción no está disponible para las demás funciones `wait*`.

Availability: Unix, not WASI, not Android, not iOS.

Added in version 3.3.

os.WUNTRACED

Esta bandera *options* para **`waitpid()`**, **`wait3()`** y **`wait4()`** hace que también se informe sobre los procesos secundarios si se han detenido pero no se ha informado sobre su estado actual desde que se detuvieron.

Esta opción no está disponible para **`waitid()`**.

Availability: Unix, not WASI, not Android, not iOS.

os.WNOHANG

Esta bandera *options* hace que **`waitpid()`**, **`wait3()`**, **`wait4()`** y **`waitid()`** regresen inmediatamente si no hay ningún estado de proceso secundario disponible inmediatamente.

Availability: Unix, not WASI, not Android, not iOS.

os.WNOWAIT

Esta bandera *options* hace que **`waitid()`** deje al niño en un estado de espera, de modo que se pueda usar una llamada **`wait*()`** posterior para recuperar nuevamente la información del estado del niño.

Esta opción no está disponible para las demás funciones `wait*`.

Availability: Unix, not WASI, not Android, not iOS.

os.CLD_EXITED
os.CLD_KILLED
os.CLD_DUMPED
os.CLD_TRAPPED
os.CLD_STOPPED
os.CLD_CONTINUED

Estos son los valores posibles para **`si_code`** en el resultado devuelto por **`waitid()`**.

Availability: Unix, not WASI, not Android, not iOS.

Added in version 3.3.

Distinto en la versión 3.9: Agregado los valores **`CLD_KILLED`** y **`CLD_STOPPED`**.

os.waitstatus_to_exitcode(*status*)

Convertir un estado de espera a un código de salida.

En Unix:

- Si el proceso termino con normalidad (if `WIFEXITED(status)` is true, se retornan el estado de salida del proceso (return `WEXITSTATUS(status)`): resultado mayor o igual a 0.
- Si el proceso fue terminado por una señal (if `WIFSIGNALED(status)` is true), retorna `-signum` donde *signum* es el numero de la señal que cause que el proceso termine (return `-WTERMSIG(status)`):

resultado menor que 0.
- En el caso contrario, se lanza un `ValueError`.

En Windows, retorna *status* desplazado a la derecha en 8 bits.

En Unix, si el proceso esta siendo rastreado o si `waitpid()` fue llamado con la opción `WUNTRACED`, el que llama debe revisar primero si `WIFSTOPPED(status)` es verdadero. La función no debe de ser llamada si `WIFSTOPPED(status)` es verdadero.

Ver también

Funciones `WIFEXITED()`, `WEXITSTATUS()`, `WIFSIGNALED()`, `WTERMSIG()`, `WIFSTOPPED()`, `WSTOPSIG()`.

Availability: Unix, Windows, not WASI, not Android, not iOS.

Added in version 3.9.

Las siguientes funciones toman un código de estado del proceso retornado por `system()`, `wait()`, o `waitpid()` como parámetro. Pueden usarse para determinar la disposición de un proceso.

os.WCOREDUMP(*status*, */*)

Retorna `True` si se generó un volcado de núcleo para el proceso; de lo contrario, retorna `False`.

Esta función debe emplearse solo si `WIFSIGNALED()` es verdadero.

Availability: Unix, not WASI, not Android, not iOS.

os.WIFCONTINUED(*status*)

Devuelve `True` si un hijo detenido se ha reanudado mediante la entrega de `SIGCONT` (si el proceso ha continuado desde una detención del control de trabajo); de lo contrario, devuelve `False`.

Ver opción **`WCONTINUED`**.

Availability: Unix, not WASI, not Android, not iOS.

os.WIFSTOPPED(*status*)

Retorna `True` si el proceso se detuvo mediante la entrega de una señal; de lo contrario, retorna `False`.

`WIFSTOPPED()` solo retorna `True` si la llamada **`waitpid()`** se realizó utilizando la opción **`WUNTRACED`** o cuando se rastrea el proceso (consulte *ptrace(2)* *[https://manpages.debian.org/ptrace(2)]*)

Availability: Unix, not WASI, not Android, not iOS.

os.WIFSIGNALED(*status*)

Retorna `True` si el proceso finalizó con una señal; de lo contrario, retorna `False`.

Availability: Unix, not WASI, not Android, not iOS.

os.WIFEXITED(*status*)

Retorna `True` si el proceso finalizó normalmente, es decir, llamando a `exit()` o `_exit()`, o volviendo de `main()`; de lo contrario, retorna `False`.

Availability: Unix, not WASI, not Android, not iOS.

os.WEXITSTATUS(*status*)

Retorna el estado de salida del proceso.

Esta función debe emplearse solo si **`WIFEXITED()`** es verdadero.

Availability: Unix, not WASI, not Android, not iOS.

os.WSTOPSIG(*status*)

Retorna la señal que hizo que el proceso se detuviera.

Esta función debe emplearse solo si `WIFSTOPPED()` es verdadero.

Availability: Unix, not WASI, not Android, not iOS.

os.WTERMSIG(*status*)

Retorna el número de la señal que provocó la finalización del proceso.

Esta función debe emplearse solo si `WIFSIGNALED()` es verdadero.

Availability: Unix, not WASI, not Android, not iOS.

Interfaz al planificador

Estas funciones controlan cómo el sistema operativo asigna el tiempo de CPU a un proceso. Solo están disponibles en algunas plataformas Unix. Para obtener información más detallada, consulte las páginas de manual de Unix.

Added in version 3.3.

Las siguientes políticas de programación están expuestas si son compatibles con el sistema operativo.

os.SCHED_OTHER

La política de programación predeterminada.

os.SCHED_BATCH

Política de programación para procesos intensivos en CPU que intenta preservar la interactividad en el resto de la computadora.

os.SCHED_DEADLINE

Scheduling policy for tasks with deadline constraints.

Added in version 3.14.

os.SCHED_IDLE

Política de programación para tareas en segundo plano de prioridad extremadamente baja.

os.SCHED_NORMAL

Alias for **`SCHED_OTHER`**.

Added in version 3.14.

os.SCHED_SPORADIC

Política de programación para programas de servidor esporádicos.

os.SCHED_FIFO

Una política de programación *First In First Out*.

os.SCHED_RR

Una política de programación round-robin.

os.SCHED_RESET_ON_FORK

Esta flag se puede OR con cualquier otra política de programación. Cuando un proceso con este indicador establece bifurcaciones, la política de programación y la prioridad de su hijo se restablecen a los valores predeterminados.

class os.sched_param(*sched_priority*)

Esta clase representa los parámetros de programación ajustables utilizados en **`sched_setparam()`**, **`sched_setscheduler()`** y **`sched_getparam()`**. Es inmutable.

Por el momento, solo hay un parámetro posible:

sched_priority

La prioridad de programación para una política de programación.

os.sched_get_priority_min(*policy*)

Obtiene el valor de prioridad mínimo para *policy*. *policy* es una de las constantes de política de programación anteriores.

os.sched_get_priority_max(*policy*)

Obtiene el valor de prioridad máxima para *policy*. *policy* es una de las constantes de política de programación anteriores.

os.sched_setscheduler(*pid*, *policy*, *param*, */*)

Establece la política de programación para el proceso con PID *pid*. Un *pid* de 0 significa el proceso de llamada. *policy* es una de las constantes de política de programación anteriores. *param* es una instancia de `sched_param`.

os.sched_getscheduler(*pid*, */*)

Retorna la política de programación para el proceso con PID *pid*. Un *pid* de 0 significa el proceso de llamada. El resultado es una de las constantes de política de programación anteriores.

os.sched_setparam(*pid*, *param*, */*)

Establece los parámetros de programación para el proceso con PID *pid*. Un *pid* de 0 significa el proceso de llamada. *param* es una instancia de `sched_param`.

os.sched_getparam(*pid*, */*)

Retorna los parámetros de programación como una instancia de `sched_param` para el proceso con PID *pid*. Un *pid* de 0 significa el proceso de llamada.

os.sched_rr_get_interval(*pid*, */*)

Retorna el round-robin quantum en segundos para el proceso con PID *pid*. Un *pid* de 0 significa el proceso de llamada.

os.sched_yield()

Voluntarily relinquish the CPU. See *sched_yield(2)* *[https://*

manpages.debian.org/sched_yield(2)] for details.

os.sched_setaffinity(*pid*, *mask*, */*)

Restringe el proceso con PID *pid* (o el proceso actual si es cero) a un conjunto de CPU. *mask* es un entero iterable que representa el conjunto de CPU a las que se debe restringir el proceso.

os.sched_getaffinity(*pid*, */*)

Devuelve el conjunto de CPU a las que está restringido el proceso con PID *pid*.

Si *pid* es cero, devuelve el conjunto de CPU a las que está restringido el hilo de llamada del proceso actual.

Véase también la función **`process_cpu_count()`**.

Información miscelánea del sistema

os.confstr(*name*, */*)

Retorna valores de configuración del sistema con valores de cadena. *name* especifica el valor de configuración para recuperar; puede ser una cadena que es el nombre de un valor de sistema definido; estos nombres se especifican en varios estándares (POSIX, Unix 95, Unix 98 y otros). Algunas plataformas también definen nombres adicionales. Los nombres conocidos por el sistema operativo host se dan como las claves del diccionario `confstr_names`. Para las variables de configuración no incluidas en esa asignación, también se acepta pasar un número entero para *name*.

Si el valor de configuración especificado por *name* no está definido, se retorna `None`.

Si *name* es una cadena y no se conoce, se excita **`ValueError`**. Si el sistema host no admite un valor específico para *name*, incluso si está incluido en `confstr_names`, se lanza un **`OSError`** con **`errno.EINVAL`** para el número de error .

Availability: Unix.

os.confstr_names

Nombres de mapeo de diccionario aceptados por `confstr()` a los valores enteros definidos para esos nombres por el sistema operativo host. Esto se puede usar para determinar el conjunto de nombres conocidos por el sistema.

Availability: Unix.

os.cpu_count()

Devuelve la cantidad de CPU lógicas en **system**. Devuelve `None` si no se determina.

La función `process_cpu_count()` se puede utilizar para obtener la cantidad de CPU lógicas que puede utilizar el hilo que llama a **current process**.

Added in version 3.4.

Distinto en la versión 3.13: If `-X cpu_count` is given or `PYTHON_CPU_COUNT` is set, `cpu_count()` returns the override value *n*.

os.getloadavg()

Retorna el número de procesos en la cola de ejecución del sistema promediada durante los últimos 1, 5 y 15 minutos o aumentos `OSError` si el promedio de carga no se pudo obtener.

Availability: Unix.

os.process_cpu_count()

Obtiene la cantidad de CPU lógicas que puede utilizar el subproceso que realiza la llamada a **current process**. Devuelve `None` si no se determina. Puede ser menor que `cpu_count()` según la afinidad de la CPU.

La función `cpu_count()` se puede utilizar para obtener la cantidad de CPU lógicas en **system**.

If `-X cpu_count` is given or `PYTHON_CPU_COUNT` is set, **`process_cpu_count()`** returns the override value *n*.

See also the `sched_getaffinity()` function.

Added in version 3.13.

os.sysconf(*name*, /)

Retorna valores de configuración del sistema con valores enteros. Si el valor de configuración especificado por *name* no está definido, se retorna `-1`. Los comentarios sobre el parámetro *name* para `confstr()` se aplican aquí también; El diccionario que proporciona información sobre los nombres conocidos viene dado por `sysconf_names`.

Availability: Unix.

os.sysconf_names

Nombres de asignación de diccionario aceptados por `sysconf()` a los valores enteros definidos para esos nombres por el sistema operativo host. Esto se puede usar para determinar el conjunto de nombres conocidos por el sistema.

Availability: Unix.

Distinto en la versión 3.11: Agregue el nombre `'SC_MINSIGSTKSZ'`.

Los siguientes valores de datos se utilizan para admitir operaciones de manipulación de rutas. Estos están definidos para todas las plataformas.

Las operaciones de nivel superior en los nombres de ruta se definen en el módulo `os.path`.

os.curdir

La cadena constante utilizada por el sistema operativo para referirse al directorio actual. Esto es `'.'` Para Windows y POSIX. También disponible a través de **`os.path`**.

os.pardir

La cadena constante utilizada por el sistema operativo para hacer referencia al directorio principal. Esto es `'...'` para Windows y POSIX. También disponible a través de **`os.path`**.

os.sep

El carácter utilizado por el sistema operativo para separar los componentes del nombre de ruta. Esto es `'/'` para POSIX y `'\\'` para Windows. Tenga en cuenta que saber esto no es suficiente para poder analizar o concatenar nombres de ruta — use **`os.path.split()`** y **`os.path.join()`** — pero en ocasiones es útil. También disponible a través de **`os.path`**.

os.altsep

Un carácter alternativo utilizado por el sistema operativo para separar los componentes del nombre de ruta, o `None` si solo existe un carácter separador. Esto se establece en `'/'` en los sistemas Windows donde `sep` es una barra invertida. También disponible a través de **`os.path`**.

os.extsep

El carácter que separa el nombre de archivo base de la extensión; por ejemplo, el `'.'` en `os.py`. También disponible a través de **`os.path`**.

os.pathsep

El carácter utilizado convencionalmente por el sistema operativo para separar los componentes de la ruta de búsqueda (como en **`PATH`**), como `':'` para POSIX o `';'` para Windows. También disponible a través de **`os.path`**.

os.defpath

La ruta de búsqueda predeterminada utilizada por **`exec*p*`** y **`spawn*p*`** si el entorno no tiene una tecla `'RUTA'`.

También disponible a través de **`os.path`**.

os.linesep

La cadena utilizada para separar (o, más bien, terminar) líneas en la plataforma actual. Este puede ser un solo carácter, como `'\n'` para POSIX, o varios caracteres, por ejemplo, `'\r\n'` para Windows. No utilice *os.linesep* como terminador de línea cuando escriba archivos abiertos en modo texto (el valor predeterminado); use un solo `'\n'` en su lugar, en todas las plataformas.

os.devnull

La ruta del archivo del dispositivo nulo. Por ejemplo: `'/dev/null'` para POSIX, `'nul'` para Windows. También disponible a través de **`os.path`**.

os.RTLD_LAZY
os.RTLD_NOW
os.RTLD_GLOBAL
os.RTLD_LOCAL
os.RTLD_NODELETE
os.RTLD_NOLOAD
os.RTLD_DEEPBIND

Flags para usar con las funciones **`setdlopenflags()`** y **`getdlopenflags()`**. Consulte la página del manual de Unix *dlopen(3)* *[https://manpages.debian.org/dlopen(3)]* para saber qué significan las diferentes flags.

Added in version 3.3.

Números al azar

os.getrandom(*size*, *flags*=*0*)

Obtiene hasta *size* bytes aleatorios. La función puede retornar menos bytes que los solicitados.

Estos bytes se pueden usar para generar generadores de números aleatorios en el espacio del usuario o para fines criptográficos.

`getrandom()` se basa en la entropía obtenida de los controladores de dispositivos y otras fuentes de ruido ambiental. La lectura innecesaria de grandes cantidades de datos tendrá un impacto negativo en otros usuarios de los dispositivos `/dev/random` y `/dev/urandom`.

El argumento de banderas es una máscara de bits que puede contener cero o más de los siguientes valores unidos mediante OR: **`os.GRND_RANDOM`** y **`GRND_NONBLOCK`**.

Véase también el Linux getrandom() manual page [https://man7.org/linux/man-pages/man2/getrandom.2.html].

Availability: Linux >= 3.17.

Added in version 3.6.

os.urandom(*size*, /)

Retorna una cadena de bytes de bytes aleatorios *size* adecuados para uso criptográfico.

Esta función retorna bytes aleatorios de una fuente de aleatoriedad específica del sistema operativo. Los datos retornados deben ser lo suficientemente impredecibles para las aplicaciones criptográficas, aunque su calidad exacta depende de la implementación del sistema operativo.

En Linux, si la llamada al sistema `getrandom()` está disponible, se usa en modo de bloqueo: bloquee hasta que el grupo de entropía urandom del sistema se inicialice (el núcleo recopila 128 bits de entropía). Ver el **PEP 524** [https://peps.python.org/pep-0524/] para la justificación. En Linux, la función **`getrandom()`** puede usarse para obtener bytes aleatorios en modo sin bloqueo (usando el indicador **`GRND_NONBLOCK`**) o para sondear hasta que el grupo de entropía urandom del sistema se inicialice.

En un sistema tipo Unix, los bytes aleatorios se leen desde el dispositivo `/dev/urandom`. Si el dispositivo `/dev/`

`urandom` no está disponible o no es legible, se genera la excepción **`NotImplementedError`**.

En Windows, usará `BCryptGenRandom()`.

Ver también

El módulo **`secrets`** proporciona funciones de nivel superior. Para obtener una interfaz fácil de usar con el generador de números aleatorios proporcionado por su plataforma, consulte **`random.SystemRandom`**.

Distinto en la versión 3.5: En Linux 3.17 y versiones posteriores, la llamada al sistema `getrandom()` ahora se usa cuando está disponible. En OpenBSD 5.6 y posterior, ahora se usa la función C `getentropy()`. Estas funciones evitan el uso de un descriptor de archivo interno.

Distinto en la versión 3.5.2: En Linux, si el syscall `getrandom()` bloquea (el grupo de entropía urandom aún no está inicializado), recurra a la lectura `/dev/urandom`.

Distinto en la versión 3.6: En Linux, `getrandom()` ahora se usa en modo de bloqueo para aumentar la seguridad.

Distinto en la versión 3.11: En Windows, se utiliza `BCryptGenRandom()` en lugar de `CryptGenRandom()`, que está en desuso.

os.GRND_NONBLOCK

Por defecto, cuando lee desde `/dev/random`, **`getrandom()`** bloquea si no hay bytes aleatorios disponibles, y cuando lee desde `/dev/urandom`, bloquea si el grupo de entropía no tiene, sin embargo, se ha inicializado.

Si se establece el indicador **`GRND_NONBLOCK`**, entonces **`getrandom()`** no se bloquea en estos casos, sino que inmediatamente genera **`BlockingIOError`**.

Added in version 3.6.

os.GRND_RANDOM

Si se establece este bit, los bytes aleatorios se extraen del grupo `/dev/random` en lugar del grupo `/dev/urandom`.

Added in version 3.6.

`io` — Core tools for working with streams

Código fuente: Lib/io.py [https://github.com/python/cpython/tree/3.14/Lib/io.py]

Resumen

The `io` module provides Python's main facilities for dealing with various types of I/O. There are three main types of I/O: *text I/O*, *binary I/O* and *raw I/O*. These are generic categories, and various backing stores can be used for each of them. A concrete object belonging to any of these categories is called a file object. Other common terms are *stream* and *file-like object*.

Independiente de su categoría, cada objeto *stream* también tendrá varias capacidades: puede ser solamente para lectura, solo escritura, or lectura y escritura. También permite arbitrariamente acceso aleatorio (buscando adelante o hacia atrás en cualquier lugar) o solamente acceso secuencial (por ejemplo en el caso de un *socket* o *pipe*).

All streams are careful about the type of data you give to them. For example giving a `str` object to the `write()` method of a binary stream will raise a `TypeError`. So will giving a `bytes` object to the `write()` method of a text stream.

Distinto en la versión 3.3: Operaciones que lanzarán un `IOError` ahora lanzan `OSError`, ya que `IOError` es un alias de `OSError`.

E/S Texto

E/S de tipo texto espera y produce objetos de clase `str`. Esto significa que cuando el respaldo de almacenamiento está compuesto de forma nativa de *bytes* (como en el caso de un archivo), la codificación y descodificación de datos está hecho de forma transparente tanto como traducción opcional de caracteres de nueva línea específicos de la plataforma.

The easiest way to create a text stream is with **`open()`**, optionally specifying an encoding:

```
f = open("myfile.txt", "r", encoding="utf-8")
```

Streams de texto en memoria también están disponibles como objetos de tipo **`StringIO`**:

```
f = io.StringIO("some initial text data")
```

Nota

When working with a non-blocking stream, be aware that read operations on text I/O objects might raise a **`BlockingIOError`** if the stream cannot perform the operation immediately.

El *API* (interfaz de programación de aplicaciones) de *streams* tipo texto está descrito con detalle en la documentación de **`TextIOBase`**.

E/S Binaria

E/S binaria (también conocido como *buffered E/S*) espera objetos tipo bytes y produce objetos tipo **`bytes`**. No se hace codificación, descodificación, o traducciones de nueva línea. Esta categoría de *streams* puede ser usada para todos tipos de datos sin texto, y también cuando se desea control manual sobre el manejo de dato textual.

The easiest way to create a binary stream is with **`open()`** with `'b'` in the mode string:

```
f = open("myfile.jpg", "rb")
```

Los *streams* binarios en memoria también están disponibles como objetos tipo **`BytesIO`**:

```
f = io.BytesIO(b"some initial binary data: \x00\x01")
```

El *API* de *stream* binario está descrito con detalle en la documentación de **`BufferedIOBase`**.

Otros módulos bibliotecarios pueden proveer maneras alternativas para crear *streams* de tipo texto o binario. Ver **`socket.socket.makefile()`** como ejemplo.

E/S sin formato

E/S sin formato (también conocido como *unbuffered E/S*) es generalmente usado como un fundamento de nivel bajo para *streams* binario y tipo texto; es raramente útil para manipular directamente *streams* sin formatos del código de usuario. Sin embargo puedes crear un *stream* sin formato abriendo un archivo en modo binario con el búfer apagado:

```
f = open("myfile.jpg", "rb", buffering=0)
```

El *API* de *streams* sin formato está descrito con detalle en la documentación de **`RawIOBase`**.

Codificación de texto

The default encoding of **`TextIOWrapper`** and **`open()`** is locale-specific (**`locale.getencoding()`**).

Sin embargo, muchos desarrolladores olvidan especificar la codificación al abrir archivos de texto codificados en UTF-8 (por ejemplo, JSON, TOML, Markdown, etc.) ya que la mayoría de las plataformas Unix usan la configuración regional UTF-8 de forma predeterminada. Esto causa errores porque la codificación local no es UTF-8 para la mayoría de los usuarios de Windows. Por ejemplo:

```
# May not work on Windows when non-ASCII characters in t
with open("README.md") as f:
    long_description = f.read()
```

Accordingly, it is highly recommended that you specify the encoding explicitly when opening text files. If you want to use UTF-8, pass `encoding="utf-8"`. To use the current locale encoding, `encoding="locale"` is supported since Python 3.10.

Ver también

Modo Python UTF-8
El modo UTF-8 de Python se puede utilizar para cambiar la codificación predeterminada a UTF-8 desde la codificación específica de la configuración regional.

PEP 686 **[https://peps.python.org/pep-0686/]**
Python 3.15 hará que Modo Python UTF-8 sea el valor predeterminado.

EncodingWarning opcional

Added in version 3.10: Consulte **PEP 597** [https://peps.python.org/pep-0597/] para más detalles.

To find where the default locale encoding is used, you can enable the `-X warn_default_encoding` command line option or set the `PYTHONWARNDEFAULTENCODING` environment variable, which will emit an `EncodingWarning` when the default encoding is used.

Si está proporcionando una API que usa `open()` o `TextIOWrapper` y pasa `encoding=None` como parámetro, puede usar `text_encoding()` para que las personas que llaman a la API emitan un `EncodingWarning` si no pasan un `encoding`. Sin embargo, considere usar UTF-8 de forma predeterminada (es decir, `encoding="utf-8"`) para las nuevas API.

Interfaz de módulo de alto nivel

io.DEFAULT_BUFFER_SIZE

Un *int* que contiene el búfer de tamaño predeterminado usado por las clases de tipo E/S. `open()` utiliza el *blksize* del archivo (obtenido por `os.stat()`) si es posible.

io.open(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*, *opener=None*)

Esto es un alias para la función incorporada `open()`.

This function raises an auditing event `open` with arguments *path*, *mode* and *flags*. The *mode* and *flags* arguments may have been modified or inferred from the original call.

io.open_code(*path*)

Abre el archivo dado con el modo `'rb'`. Esta función debe ser usado cuando la intención es tratar el contenido como código ejecutable.

path should be a `str` and an absolute path.

The behavior of this function may be overridden by an earlier call to the `PyFile_SetOpenCodeHook()`. However, assuming that *path* is a `str` and an absolute path, `open_code(path)` should always behave the same as `open(path, 'rb')`. Overriding the behavior is intended for additional validation or preprocessing of the file.

Added in version 3.8.

io.text_encoding(*encoding*, *stacklevel=2*, */*)

Esta es una función auxiliar para personas que llaman que usan `open()` o `TextIOWrapper` y tienen un parámetro `encoding=None`.

This function returns *encoding* if it is not `None`. Otherwise, it returns `"locale"` or `"utf-8"` depending on UTF-8 Mode.

This function emits an `EncodingWarning` if `sys.flags.warn_default_encoding` is true and *encoding* is `None`. *stacklevel* specifies where the warning is emitted. For example:

```
def read_text(path, encoding=None):
    encoding = io.text_encoding(encoding)  # stackl
    with open(path, encoding) as f:
        return f.read()
```

En este ejemplo, se emite un **`EncodingWarning`** para el llamador de `read_text()`.

Consulte Codificación de texto para obtener más información.

Added in version 3.10.

Distinto en la versión 3.11: **`text_encoding()`** retornará «utf-8» cuando esté habilitado el modo UTF-8 y el *encoding* es `None`.

exception io.BlockingIOError

Esto es un alias de compatibilidad para la incorporada excepción **`BlockingIOError`**.

exception io.UnsupportedOperation

Una excepción heredando **`OSError`** y **`ValueError`** que es generado cuando se llama a una operación no admitida en un *stream*.

Ver también

`sys`

contiene los *streams* estándar de IO **`sys.stdin`**, **`sys.stdout`**, y **`sys.stderr`**.

Jerarquía de clases

La implementación de *streams* E/S está organizada como una jerarquía de clases. Primero abstract base classes (ABC), que son usados para especificar las varias categorías de *streams*, luego las clases concretas proveen un *stream* estándar de implementaciones.

Nota

The abstract base classes also provide default implementations of some methods in order to help implementation of concrete stream classes. For example, `BufferedIOBase` provides unoptimized implementations of `readinto()` and `readline()`.

En la parte superior de la jerarquía E/S está la clase abstracta base `IOBase`. Define la interfaz básica del *stream*. Tenga en cuenta que no hay separación entre *streams* de lectura y escritura; implementaciones están permitidos lanzar `UnsupportedOperation` si no apoyan la operación.

La clase `RawIOBase` extiende `IOBase`. Maneja la lectura y escritura de bytes a un *stream*. `FileIO` subclasifica `RawIOBase` para proveer una interfaz a los archivos en el sistema de archivos de la máquina.

The `BufferedIOBase` ABC extends `IOBase`. It deals with buffering on a raw binary stream (`RawIOBase`). Its subclasses, `BufferedWriter`, `BufferedReader`, and `BufferedRWPair` buffer raw binary streams that are writable, readable, and both readable and writable, respectively. `BufferedRandom` provides a buffered interface to seekable streams. Another `BufferedIOBase` subclass, `BytesIO`, is a stream of in-memory bytes.

El `TextIOBase` ABC, otra subclasificación de `IOBase`, trata con los streams cuyos bytes representan texto, y maneja la codificación y decodificación para cadenas de caracteres y de estos mismos. `TextIOWrapper`, que extiende `TextIOBase`, es un interfaz textual almacenado un *stream* sin formato amortiguado (`BufferedIOBase`). Finalmente, `StringIO` es una stream en memoria para texto.

Los nombres de los argumentos no son parte de la especificación, y solo los argumentos de `open()` están destinados a ser utilizados como argumentos de palabras clave.

The following table summarizes the ABCs provided by the `io` module:

		Métodos de resguardo (*Stub*)	Métodos Mixin y propiedades
`IOBase`		`fileno`, `seek`, and `truncate`	`close`, `closed`, `__enter__`, `__exit__`, `flush`, `isatty`, `__iter__`, `__next__`, `readable`, `readline`, `readlines`, `seekable`, `tell`, `writable`, and `writelines`
`RawIOBase`	`IOBase`	`readinto` and `write`	Métodos `IOBase` heredados, `read`, and `readall`
`BufferedIOBase`	`IOBase`	`detach`, `read`, `read1`, and `write`	Métodos `IOBase` heredados, `readinto`, and `readinto1`
`TextIOBase`	`IOBase`	`detach`, `read`, `readline`, and `write`	Métodos `IOBase` heredados, `encoding`, `errors`, and `newlines`

Clases base E/S

class io.IOBase

La clase base abstracta para todas las clases de E/S.

Esta clase provee implementaciones abstractas vacías para muchos métodos que clases que derivadas pueden anular selectivamente; la implementación predeterminada representa un archivo que no se puede leer, grabar o ser buscado.

Even though `IOBase` does not declare **`read()`** or **`write()`** because their signatures will vary, implementations and clients should consider those methods part of the interface. Also, implementations may raise a `ValueError` (or `UnsupportedOperation`) when operations they do not support are called.

El tipo básico usado para leer datos binarios o grabar un archivo es `bytes`. Otros bytes-like objects son aceptados como argumentos para métodos también. Clases de tipo E/S funcionan usando datos de tipo `str`.

Tenga en cuenta que llamando cualquier método (incluso indagaciones) en un *stream* cerrada es indefinido. En este caso implementaciones podrían lanzar un error `ValueError`.

`IOBase` (y sus subclasificaciones) apoyan el protocolo iterador, significando que un objeto de clase **`IOBase`** puede ser iterado sobre el rendimiento de las líneas en un *stream* de datos. Líneas son definidas un poco diferente dependiendo si el *stream* es de tipo binario (produciendo *bytes*), o un *stream* de texto (produciendo cadenas de caracteres). Ver `readline()` abajo.

`IOBase` es también un gestor de contexto y por ende apoya la declaración `with`. En este ejemplo, *file* es cerrado después de que la declaración **`with`** termina–incluso si alguna excepción ocurre:

```
with open('spam.txt', 'w') as file:
    file.write('Spam and eggs!')
```

`IOBase` provee los siguientes atributos y métodos:

close()

Cierra el *stream*. Este método no tiene efecto si el archivo ya está cerrado. Cuándo está cerrado, cualquier operación que se le haga al archivo (ej. leer or grabar) lanzará el error `ValueError`.

Como conveniencia, se permite llamar este método más que una vez. Sin embargo, solamente el primer llamado tenderá efecto.

closed

`True` si está cerrada el *stream*.

fileno()

Retorna el descriptor de archivo subyacente (un número de tipo entero) de el *stream* si existe. Un `OSError` se lanza si el objeto IO no tiene un archivo descriptor.

flush()

Vacía los buffers de grabación del *stream* si corresponde. Esto no hace nada para *streams* que son solamente de lectura o *streams* sin bloqueo.

isatty()

Retorna `True` si el *stream* es interactiva (ej., si está conectado a un terminal o dispositivo tty).

readable()

Return `True` if the stream can be read from. If `False`, **`read()`** will raise **`OSError`**.

readline(*size=-1*, */*)

Lee y retorna una línea del *stream*. Si *size* (tamaño) es especificado, se capturará un máximo de ése mismo tamaño especificado en *bytes*.

El terminador de la línea siempre es `b'\n'` para archivos de tipo binario; para archivos de tipo texto el argumento *newline* para la función **`open()`** pueden ser usados para seleccionar las líneas terminadoras reconocidas.

readlines(*hint=-1*, */*)

Lee y retorna una lista de líneas del *stream*. *hint* puede ser especificado para controlar el número de líneas que se lee: no se leerán más líneas si el tamaño total (en *bytes* / caracteres) de todas las líneas excede *hint*.

Los valores *hint* de `0` o menos, así como `None`, se tratan como sin pista.

Note that it's already possible to iterate on file objects using `for line in file: ...` without calling **`file.readlines()`**.

seek(*offset*, *whence=os.SEEK_SET*, */*)

Change the stream position to the given byte *offset*, interpreted relative to the position indicated by *whence*, and return the new absolute position. Values for *whence* are:

- **`os.SEEK_SET`** or `0` – start of the stream (the default); *offset* should be zero or positive

- `os.SEEK_CUR` or `1` – current stream position; *offset* may be negative
- `os.SEEK_END` or `2` – end of the stream; *offset* is usually negative

Added in version 3.1: The **`SEEK_*`** constants.

Added in version 3.3: Some operating systems could support additional values, like `os.SEEK_HOLE` or `os.SEEK_DATA`. The valid values for a file could depend on it being open in text or binary mode.

seekable()

Retorna `True` si el *stream* apoya acceso aleatorio. Si retorna `False`, `seek()`, `tell()` y `truncate()` lanzarán `OSError`.

tell()

Retorna la posición actual del *stream*.

truncate(*size*=*None*, */*)

Cambia el tamaño del *stream* al *size* dado en *bytes* (o la posición actual si no se especifica *size*). La posición actual del *stream* no se cambia. Este cambio de tamaño puede incrementar o reducir el tamaño actual del archivo. En caso de extensión, los contenidos del área del nuevo archivo depende de la plataforma (en la mayoría de los sistemas *bytes* adicionales son llenos de cero). Se retorna el nuevo tamaño del archivo.

Distinto en la versión 3.5: Windows llenará los archivos con cero cuando extienda.

writable()

Return `True` if the stream supports writing. If `False`, **`write()`** and `truncate()` will raise `OSError`.

writelines(*lines*, */*)

Escribir una lista de líneas al *stream*. No se agrega separadores de líneas, así que es usual que las líneas tengan separador al final.

del()

Prepara para la destrucción de un objeto. **`IOBase`** proporciona una implementación dada de este método que ejecuta las instancias del método **`close()`**.

class io.RawIOBase

Base class for raw binary streams. It inherits from **`IOBase`**.

Los flujos binarios sin procesar generalmente brindan acceso de bajo nivel a un dispositivo OS subyacente o API, y no intentan encapsularlos en primitivas de alto nivel (esta funcionalidad se realiza en un nivel superior en flujos binarios almacenados en búfer y flujos de texto, que se describen más adelante en esta página).

`RawIOBase` proporciona estos métodos además de los de **`IOBase`**:

read(*size=-1*, */*)

Lee hasta el *size* de los *bytes* del objeto y los retorna. Como conveniencia si no se especifica *size* o es -1, se retornan todos los *bytes* hasta que se retorne el fin del archivo. Sino, se hace solo un llamado al sistema. Se pueden retornar menos de *size bytes* si la llamada del sistema operativo retorna menos de *size bytes*.

Si se retorna 0 *bytes* y el *size* no era 0, esto indica que es el fin del archivo. Si el objeto está en modo sin bloqueo y no hay *bytes* disponibles, se retorna `None`.

La implementación dada difiera al método **`readall()`** y **`readinto()`**.

readall()

Lee y retorna todos los *bytes* del *stream* hasta llegar al fin del archivo, usando, si es necesario, varias llamadas al *stream*.

readinto(*b*, */*)

Lee *bytes* en objeto pre-asignado y grabable bytes-like object *b*, y retorna el número de *bytes* leído. Por ejemplo, *b* puede ser una clase de tipo **`bytearray`**. Si el objeto está en modo sin bloquear y no hay *bytes* disponibles, se retorna `None`.

write(*b*, */*)

Escribe bytes-like object dado, *b*, al *stream* subyacente y retorna la cantidad de *bytes* grabadas. Esto puede ser menos que la longitud de *b* en *bytes*, dependiendo de la especificaciones del *stream* subyacente, especialmente si no está en modo no-bloqueo. `None` se retorna si el *stream* sin formato está configurado para no bloquear y ningún *byte* puede ser rápidamente grabada. El llamador puede deshacer o mutar *b* después que retorne este método, así que la implementación solo debería acceder *b* durante la ejecución al método.

class io.BufferedIOBase

Base class for binary streams that support some kind of buffering. It inherits from **`IOBase`**.

The main difference with **`RawIOBase`** is that methods **`read()`**, **`readinto()`** and **`write()`** will try (respectively) to read as much input as requested or to emit all provided data.

In addition, if the underlying raw stream is in non-blocking mode, when the system returns would block **`write()`** will raise **`BlockingIOError`** with **`BlockingIOError.characters_written`** and **`read()`** will return data read so far or `None` if no data is available.

Además, el método **`read()`** no tiene una implementación dada que difiere al método `readinto()`.

Una implementación típica de `BufferedIOBase` no debería heredar una implementación de `RawIOBase`, es más, debería envolver como uno, así como hacen las clases `BufferedWriter` y `BufferedReader`.

`BufferedIOBase` proporciona o anula estos atributos y métodos de datos además de los de `IOBase`:

raw

> El *stream* sin formato subyacente (una instancia `RawIOBase`) que `BufferedIOBase` maneja. Esto no es parte de la API **`BufferedIOBase`** y posiblemente no exista en algunas implementaciones.

detach()

> Separa el *stream* subyacente del búfer y lo retorna.
>
> Luego que el *stream* sin formato ha sido separado, el búfer está en un estado inutilizable.
>
> Algunos búfer, como `BytesIO`, no tienen el concepto de un *stream* sin formato singular para retornar de este método. Lanza un `UnsupportedOperation`.
>
> *Added in version 3.1.*

read(*size=-1*, */*)

> Read and return up to *size* bytes. If the argument is omitted, `None`, or negative read as much as possible.
>
> Fewer bytes may be returned than requested. An empty `bytes` object is returned if the stream is already at EOF. More than one read may be made and calls may be retried if specific errors are encountered, see `os.read()` and PEP 475 [https://peps.python.org/pep-0475/] for more details. Less than size bytes being returned does not imply that EOF is imminent.

When reading as much as possible the default implementation will use `raw.readall` if available (which should implement **`RawIOBase.readall()`**), otherwise will read in a loop until read returns `None`, an empty **`bytes`**, or a non-retryable error. For most streams this is to EOF, but for non-blocking streams more data may become available.

Nota

When the underlying raw stream is non-blocking, implementations may either raise **`BlockingIOError`** or return `None` if no data is available. **`io`** implementations return `None`.

read1(*size=-1*, */*)

Read and return up to *size* bytes, calling **`readinto()`** which may retry if **`EINTR`** is encountered per PEP 475 [https://peps.python.org/pep-0475/]. If *size* is `-1` or not provided, the implementation will choose an arbitrary value for *size*.

Nota

When the underlying raw stream is non-blocking, implementations may either raise **`BlockingIOError`** or return `None` if no data is available. **`io`** implementations return `None`.

readinto(*b*, */*)

Lee *bytes* a un objeto predeterminado y grabable bytes-like object *b* y retorna el número de *bytes* leídos. Por ejemplo, *b* puede ser un **`bytearray`**.

Como **`read()`**, varias lecturas pueden ser otorgadas al *stream* sin formato subyacente al menos que esto último sea interactivo.

Un `BlockingIOError` se lanza si el *stream* subyacente está en modo no bloqueo y no tiene datos al momento.

readinto1(*b*, */*)

Leer *bytes* a un objeto predeterminado y grabable bytes-like object *b* usando por lo menos una llamada al método `read()` (o `readinto()`) del *stream* subyacente. Retorna la cantidad de *bytes* leídas.

Un `BlockingIOError` se lanza si el *stream* subyacente está en modo no bloqueo y no tiene datos al momento.

Added in version 3.5.

write(*b*, */*)

Escribe el bytes-like object dado, *b*, y retorna el número de bytes grabados (siempre el equivalente en longitud de *b* en bytes, ya que si falla la grabación se lanza un `OSError`). Dependiendo en la implementación actual estos bytes pueden ser grabados rápidamente al *stream* subyacente o mantenido en un búfer por razones de rendimiento y latencia.

Cuando estás en modo no bloqueo, se lanza un `BlockingIOError` si los datos tenían que ser grabadas al *stream* sin formato pero no pudo aceptar todos los datos sin bloquear.

El llamador puede otorgar o mutar *b* después que este método retorne algo, entonces la implementación debería acceder solamente a *b* durante la llamada al método.

Archivo sin formato E/S

class io.FileIO(*name*, *mode='r'*, *closefd=True*, *opener=None*)

A raw binary stream representing an OS-level file containing bytes data. It inherits from **`RawIOBase`**.

El *name* puede ser una de dos cosas:

- una cadena de caracteres u objeto de tipo **`bytes`** representando la ruta del archivo en la que fue abierto. En este caso *closefd* es `True` (el valor dado) de otra manera un error será dada.
- un *integer* representando el número de descriptores de archivos de nivel OS que resultan dando acceso a través del objeto **`FileIO`**. Cuando el objeto *FileIO* está cerrado este fd cerrará también a no ser que *closefd* esté configurado a `False`.

El *mode* puede ser `'r'`, `'w'`, `'x'` o `'a'` para lectura (el valor dado), grabación, creación exclusiva o anexando. Si no existe el archivo se creará cuando se abra para grabar o anexar; se truncará cuando se abra para grabar. Se lanzará un error **`FileExistsError`** si ya existe cuando se abra para crear. Abriendo un archivo para crear implica grabar entonces este modo se comporta similarmente a `'w'`. Agrega un `'+'` al modo para permitir lectura y grabación simultáneas.

The **`read()`** (when called with a positive argument), **`readinto()`** and **`write()`** methods on this class will only make one system call.

Un abridor personalizado puede ser usado pasando un llamador como *opener*. El descriptor de archivo subyacente es obtenido llamando *opener* con (*name*, *flags*). *opener* debe retornar un descriptor de archivo abierto (pasando **`os.open`** como *opener* resulta con funcionamiento similar a pasando `None`).

El archivo recién creado es non-inheritable.

Ver la función incorporada **`open()`** para ejemplos usando el parámetro *opener*.

Distinto en la versión 3.3: El parámetro *opener* fue agregado. El modo `'x'` fue agregado.

Distinto en la versión 3.4: El archivo ahora no es heredable.

`FileIO` proporciona estos atributos de datos además de los de **`RawIOBase`** y **`IOBase`**:

mode

El modo dado en el constructor.

name

El nombre del archivo. Este es el descriptor del archivo cuando no se proporciona ningún nombre en el constructor.

Streams almacenados (búfer)

Streams E/S almacenadas (búfer) proveen una interfaz de más alto nivel a un dispositivo E/S que a un E/S sin formato.

class io.BytesIO(*initial_bytes=b''*)

A binary stream using an in-memory bytes buffer. It inherits from **`BufferedIOBase`**. The buffer is discarded when the **`close()`** method is called.

El argumento opcional *initial_bytes* es un bytes-like object que contiene datos iniciales.

`BytesIO` provee o anula estos métodos además de los de **`BufferedIOBase`** y **`IOBase`**:

getbuffer()

Retorna una vista legible y grabable acerca de los contenidos del búfer sin copiarlos. Además mutando la vista actualizará de forma transparente los contenidos del búfer:

```
>>> b = io.BytesIO(b"abcdef")
>>> view = b.getbuffer()
>>> view[2:4] = b"56"
>>> b.getvalue()
b'ab56ef'
```

Nota

Mientras exista la vista el objeto `BytesIO` no se le puede cambiar el tamaño o cerrado.

Added in version 3.2.

getvalue()

Retorna `bytes` que contiene los contenidos enteros del búfer.

read1(*size=-1*, */*)

En la clase `BytesIO` esto es lo mismo que `read()`.

Distinto en la versión 3.7: Ahora es opcional el argumento *size*.

readinto1(*b*, */*)

En la clase `BytesIO` esto es lo mismo que `readinto()`.

Added in version 3.5.

class io.BufferedReader(*raw*, *buffer_size=DEFAULT_BUFFER_SIZE*)

A buffered binary stream providing higher-level access to a readable, non seekable `RawIOBase` raw binary stream. It inherits from `BufferedIOBase`.

Al leer datos de este objeto, es posible que se solicite una mayor cantidad de datos del flujo sin procesar subyacente y se mantenga en un búfer interno. Los datos almacenados en búfer se pueden retornar directamente en lecturas posteriores.

El constructor crea un `BufferedReader` para el *stream* legible sin formato *raw* y *buffer_size*. Si se omite *buffer_size* se usa `DEFAULT_BUFFER_SIZE`.

`BufferedReader` provee o anula los métodos en adición a los de `BufferedIOBase` y `IOBase`:

peek(*size=0, /*)

Return bytes from the stream without advancing the position. The number of bytes returned may be less or more than requested. If the underlying raw stream is non-blocking and the operation would block, returns empty bytes.

read(*size=-1, /*)

In **`BufferedReader`** this is the same as `io.BufferedIOBase.read()`

read1(*size=-1, /*)

In **`BufferedReader`** this is the same as `io.BufferedIOBase.read1()`

Distinto en la versión 3.7: Ahora es opcional el argumento *size*.

class io.BufferedWriter(*raw, buffer_size=DEFAULT_BUFFER_SIZE*)

A buffered binary stream providing higher-level access to a writeable, non seekable `RawIOBase` raw binary stream. It inherits from `BufferedIOBase`.

Al escribir en este objeto, los datos normalmente se colocan en un búfer interno. El búfer se escribirá en el objeto `RawIOBase` subyacente en varias condiciones, que incluyen:

- cuando el búfer se vuelve demasiado pequeño para todos los datos pendientes;
- when `flush()` is called;
- when a `seek()` is requested (for `BufferedRandom` objects);
- cuando el objeto `BufferedWriter` is cerrado o

anulado.

El constructor crea un `BufferedWriter` para el *stream* grabable *raw*. Si no es dado el *buffer_size*, recurre el valor `DEFAULT_BUFFER_SIZE`.

`BufferedWriter` provee o anula estos métodos además de los de `BufferedIOBase` y `IOBase`:

flush()

Forzar bytes retenidos en el búfer al *stream* sin formato. Un `BlockingIOError` debería ser lanzado si el *stream* sin formato bloquea.

write(*b*, */*)

Write the bytes-like object, *b*, and return the number of bytes written. When in non-blocking mode, a `BlockingIOError` with `BlockingIOError.characters_written` set is raised if the buffer needs to be written out but the raw stream blocks.

class io.BufferedRandom(*raw*, *buffer_size=DEFAULT_BUFFER_SIZE*)

A buffered binary stream providing higher-level access to a seekable `RawIOBase` raw binary stream. It inherits from `BufferedReader` and `BufferedWriter`.

El constructor crea un lector y una grabación para un *stream* sin formato buscable, dado en el primer argumento. Si se omite el *buffer_size* este recae sobre el valor predeterminado `DEFAULT_BUFFER_SIZE`.

`BufferedRandom` is capable of anything `BufferedReader` or `BufferedWriter` can do. In addition, `seek()` and `tell()` are guaranteed to be implemented.

class io.BufferedRWPair(*reader*, *writer*, *buffer_size=DEFAULT_BUFFER_SIZE*, */*)

A buffered binary stream providing higher-level access to two non seekable `RawIOBase` raw binary streams—one readable, the other writeable. It inherits from `BufferedIOBase`.

reader y *writer* son objetos `RawIOBase` que son respectivamente legibles y escribibles. Si se omite *buffer_size* este se recae sobre el valor predeterminado `DEFAULT_BUFFER_SIZE`.

`BufferedRWPair` implementa todos los métodos de `BufferedIOBase` excepto por `detach()`, que lanza un `UnsupportedOperation`.

Advertencia

`BufferedRWPair` no intenta sincronizar accesos al *stream* sin formato subyacente. No debes pasar el mismo objeto como legible y escribible; usa `BufferedRandom` en su lugar.

E/S Texto

class io.TextIOBase

Base class for text streams. This class provides a character and line based interface to stream I/O. It inherits from `IOBase`.

`TextIOBase` provee o anula estos atributos y métodos de datos además de los de `IOBase`:

encoding

El nombre de la codificación utilizada para decodificar los *bytes* del *stream* a cadenas de caracteres y para codificar cadenas de caracteres en bytes.

errors

La configuración de error del decodificador o codificador.

newlines

Una cadena de caracteres, una tupla de cadena de caracteres, o `None`, indicando las nuevas líneas traducidas hasta ese momento. Dependiendo de la implementación y los indicadores iniciales del constructor, esto puede no estar disponible.

buffer

The underlying binary buffer (a **`BufferedIOBase`** or **`RawIOBase`** instance) that **`TextIOBase`** deals with. This is not part of the **`TextIOBase`** API and may not exist in some implementations.

detach()

Separa el búfer binario subyacente de **`TextIOBase`** y lo retorna.

Una vez que se ha separado el búfer subyacente, la **`TextIOBase`** está en un estado inutilizable.

Algunas implementaciones de **`TextIOBase`**, como **`StringIO`**, puede no tener el concepto de un búfer subyacente y llamar a este método se lanzará **`UnsupportedOperation`**.

Added in version 3.1.

read(*size*=*-1*, */*)

Lee y retorna como máximo *size* caracteres del *stream* como un **`str`** singular. Si *size* es negativo o `None`, lee hasta llegar al fin del archivo.

readline(*size*=*-1*, */*)

Read until newline or EOF and return a single **`str`**. If the stream is already at EOF, an empty string is returned.

Si se especifica *size* como máximo *size* de caracteres será leído.

seek(*offset*, *whence=SEEK_SET*, */*)

Change the stream position to the given *offset*. Behaviour depends on the *whence* parameter. The default value for *whence* is **`SEEK_SET`**.

- **`SEEK_SET`** or `0`: seek from the start of the stream (the default); *offset* must either be a number returned by **`TextIOBase.tell()`**, or zero. Any other *offset* value produces undefined behaviour.
- **`SEEK_CUR`** or `1`: «seek» to the current position; *offset* must be zero, which is a no-operation (all other values are unsupported).
- **`SEEK_END`** or `2`: seek to the end of the stream; *offset* must be zero (all other values are unsupported).

Retorna la nueva posición absoluta como un número opaco.

Added in version 3.1: The **`SEEK_*`** constants.

tell()

Retorna la posición actual de la secuencia como un número opaco. El número no suele representar una cantidad de bytes en el almacenamiento binario subyacente.

write(*s*, */*)

Escribe la cadena de caracteres *s* al *stream* y retorna el número de caracteres grabadas.

class io.TextIOWrapper(*buffer*, *encoding=None*, *errors=None*, *newline=None*, *line_buffering=False*, *write_through=False*)

A buffered text stream providing higher-level access to a **`BufferedIOBase`** buffered binary stream. It inherits from **`TextIOBase`**.

encoding gives the name of the encoding that the stream will be decoded or encoded with. In UTF-8 Mode, this defaults to UTF-8. Otherwise, it defaults to **`locale.getencoding()`**. `encoding="locale"` can be used to specify the current locale's encoding explicitly. See Codificación de texto for more information.

errors es una cadena de caracteres opcional que especifica cómo se manejan los errores codificados y descodificados. Pasa `'strict'` para lanzar una excepción **`ValueError`** si hay un error codificado (el valor predeterminado `None` tiene el mismo efecto), o pasa `'ignore'` para ignorar errores. (Tenga en cuenta que ignorar errores puede llevar a perder datos). `'replace'` causa un marcador de reemplazo (como `'?'`) para ser insertado donde haya datos mal formados. `'backslashreplace'` reemplaza datos mal formados con una secuencia de escape de tipo barra invertida. Cuando se escribe, `'xmlcharrefreplace'` (reemplazar con la referencia apropiada de XML) o `'namereplace'` (reemplazar con secuencias de caracteres `\N{…}`) pueden ser usadas. Cualquier otro nombre de manejador de errores que hayan sido registrados con **`codecs.register_error()`** también son validas.

newline controla cómo finalizar las terminaciones de líneas. Pueden ser `None`, `''`, `'\n'`, `'\r'`, and `'\r\n'`. Funciona de la siguiente manera:

- Al leer la entrada de la secuencia, si *newline* es `None`, el modo universal newlines está habilitado. Las líneas de la entrada pueden terminar en `'\n'`, `'\r'` o `'\r\n'`, y se traducen a `'\n'` antes de retornarse al llamador. Si *newline* es `''`, el modo de salto de línea universal está habilitado, pero los finales de línea se retornan a la persona que llama sin traducir. Si *newline* tiene alguno de los otros valores legales, las líneas de entrada solo terminan con la cadena dada y la línea final se retorna al llamador sin traducir.
- Al escribir la salida al *stream*, si *newline* es `None`, cualquier carácter `'\n'` escrito son traducidos a la

línea separador *default* del sistema, **`os.linesep`**. Si *newline* es `''` o `'\n'`, la traducción no ocurre. Si *newline* es de cualquier de los otros valores legales, cualquier carácter `'\n'` escrito es traducido a la cadena de caracteres dada.

If *line_buffering* is `True`, **`flush()`** is implied when a call to write contains a newline character or a carriage return.

If *write_through* is `True`, calls to **`write()`** are guaranteed not to be buffered: any data written on the **`TextIOWrapper`** object is immediately handled to its underlying binary *buffer*.

Distinto en la versión 3.3: Se ha agregado el argumento *write_through*.

Distinto en la versión 3.3: La codificación (*encoding*) por defecto es ahora `locale.getpreferredencoding(False)` en vez de `locale.getpreferredencoding()`. No cambie temporalmente la codificación local usando **`locale.setlocale()`**, use la codificación local actual en vez del preferido del usuario.

Distinto en la versión 3.10: El argumento *encoding* ahora admite el nombre de codificación ficticia `"locale"`.

Nota

When the underlying raw stream is non-blocking, a **`BlockingIOError`** may be raised if a read operation cannot be completed immediately.

`TextIOWrapper` proporciona estos atributos y métodos de datos además de los de **`TextIOBase`** y **`IOBase`**:

line_buffering

Si el almacenamiento en línea está habilitado.

write_through

Si grabaciones son pasadas inmediatamente al búfer binario subyacente.

Added in version 3.7.

reconfigure(***, *encoding=None*, *errors=None*, *newline=None*, *line_buffering=None*, *write_through=None*)

Reconfigura este *stream* textual usando las nuevas configuraciones de *encoding*, *errors*, *newline*, *line_buffering* y *write_through*.

Los parámetros que no son especificados mantienen las configuraciones actuales, excepto por `errors='strict'` cuando *encoding* se especifica pero *errors* no está especificado.

No es posible cambiar la codificación o nueva línea si algunos datos han sido captados por el *stream*. Sin embargo, cambiando la codificación después de grabar es posible.

Este método hace una nivelación implícita del *stream* antes de configurar los nuevos parámetros.

Added in version 3.7.

Distinto en la versión 3.11: El método admite la opción `encoding="locale"`.

seek(*cookie*, *whence=os.SEEK_SET*, */*)

Set the stream position. Return the new stream position as an `int`.

Four operations are supported, given by the following argument combinations:

- `seek(0, SEEK_SET)`: Rewind to the start of the stream.
- `seek(cookie, SEEK_SET)`: Restore a previous position; *cookie* **must be** a number returned by

`tell()`.

- `seek(0, SEEK_END)`: Fast-forward to the end of the stream.
- `seek(0, SEEK_CUR)`: Leave the current stream position unchanged.

Any other argument combinations are invalid, and may raise exceptions.

Ver también

`os.SEEK_SET`, `os.SEEK_CUR`, and `os.SEEK_END`.

tell()

Return the stream position as an opaque number. The return value of **`tell()`** can be given as input to `seek()`, to restore a previous stream position.

class io.StringIO(*initial_value* = '', *newline* = '\n')

A text stream using an in-memory text buffer. It inherits from `TextIOBase`.

El búfer de texto se descarta cuando se llama al método `close()`.

El valor inicial del búfer se puede establecer proporcionando *initial_value*. Si la traducción de nuevas líneas está habilitada, las nuevas líneas se codificarán como si fueran `write()`. La secuencia se coloca al comienzo del búfer que emula la apertura de un archivo existente en un modo `w+`, preparándolo para una escritura inmediata desde el principio o para una escritura que sobrescribiría el valor inicial. Para emular la apertura de un archivo en un modo `a+` listo para anexar, use `f.seek(0, io.SEEK_END)` para reponer la secuencia al final del búfer.

El argumento *newline* funciona como el de la clase

`TextIOWrapper`, excepto que cuando escribe al flujo de salida, si *newline* es `None`, se escriben líneas nuevas como `\n` en todas las plataformas

`StringIO` proporciona este método además de los de **`TextIOBase`** y **`IOBase`**:

getvalue()

Return a **`str`** containing the entire contents of the buffer. Newlines are decoded as if by **`read()`**, although the stream position is not changed.

Ejemplos de uso:

```
import io

output = io.StringIO()
output.write('First line.\n')
print('Second line.', file=output)

# Retrieve file contents -- this will be
# 'First line.\nSecond line.\n'
contents = output.getvalue()

# Close object and discard memory buffer --
# .getvalue() will now raise an exception.
output.close()
```

class io.IncrementalNewlineDecoder

A helper codec that decodes newlines for universal newlines mode. It inherits from **`codecs.IncrementalDecoder`**.

Static Typing

The following protocols can be used for annotating function and method arguments for simple stream reading or writing operations. They are decorated with **`@typing.runtime_checkable`**.

class io.Reader[*T*]

Generic protocol for reading from a file or other input stream. `T` will usually be **`str`** or **`bytes`**, but can be any type that is read from the stream.

Added in version 3.14.

read()

read(*size*, */*)

Read data from the input stream and return it. If *size* is specified, it should be an integer, and at most *size* items (bytes/characters) will be read.

For example:

```
def read_it(reader: Reader[str]):
    data = reader.read(11)
    assert isinstance(data, str)
```

class io.Writer[*T*]

Generic protocol for writing to a file or other output stream. `T` will usually be **`str`** or **`bytes`**, but can be any type that can be written to the stream.

Added in version 3.14.

write(*data*, */*)

Write *data* to the output stream and return the number of items (bytes/characters) written.

For example:

```
def write_binary(writer: Writer[bytes]):
    writer.write(b"Hello world!\n")
```

See ABCs and Protocols for working with I/O for other I/O related protocols and classes that can be used for static type checking.

Rendimiento

Esta sección discute el rendimiento de las implementaciones concretas de E/S proporcionadas.

E/S Binaria

Leyendo y grabando solamente grandes porciones de datos incluso cuando el usuario pide para solo un byte, E/S de tipo búfer esconde toda ineficiencia llamando y ejecutando las rutinas E/S del sistema operativo que no ha pasado por el proceso de búfer. La ganancia depende en el OS y el tipo de E/S que se ejecuta. Por ejemplo, en algunos sistemas operativos modernos como Linux, un disco E/S sin búfer puede ser rápido como un E/S búfer. Al final, sin embargo, es que el E/S búfer ofrece rendimiento predecible independientemente de la plataforma y el dispositivo de respaldo. Entonces es siempre preferible user E/S búfer que E/S sin búfer para datos binarios.

E/S Texto

Text I/O over a binary storage (such as a file) is significantly slower than binary I/O over the same storage, because it requires conversions between unicode and binary data using a character codec. This can become noticeable handling huge amounts of text data like large log files. Also, `tell()` and `seek()` are both quite slow due to the reconstruction algorithm used.

`StringIO`, sin embargo, es un contenedor unicode nativo en memoria y exhibirá una velocidad similar a `BytesIO`.

Multihilo

`FileIO` objects are thread-safe to the extent that the operating system calls (such as *read(2)* *[https://manpages.debian.org/read(2)]* under Unix) they wrap are thread-safe too.

Objetos binarios búfer (instancias de `BufferedReader`, `BufferedWriter`, `BufferedRandom` y `BufferedRWPair`) protegen sus estructuras internas usando un bloqueo; es seguro, entonces, llamarlos de varios hilos a la vez.

objetos `TextIOWrapper` no son seguros para subprocesos.

Reentrada

Objetos binarios búfer (instancias de `BufferedReader`, `BufferedWriter`, `BufferedRandom` y `BufferedRWPair`) no son reentrante. Mientras llamadas reentrantes no ocurren en situaciones normales pueden surgir haciendo E/S en un manejador `signal`. Si un hilo trata de entrar de nuevo a un objeto búfer que se está accediendo actualmente, se lanza un `RuntimeError`. Tenga en cuenta que esto no prohíbe un hilo diferente entrando un objeto búfer.

The above implicitly extends to text files, since the `open()` function will wrap a buffered object inside a `TextIOWrapper`. This includes standard streams and therefore affects the built-in `print()` function as well.

[1] (1,2,3)

`time` — Time access and conversions

Este módulo proporciona varias funciones relacionadas con el tiempo. Para la funcionalidad relacionada, consulte también los módulos **`datetime`** y **`calendar`**.

Aunque este módulo siempre está disponible, no todas las funciones están disponibles en todas las plataformas. La mayoría de las funciones definidas en este módulo llaman a funciones C de la biblioteca de la plataforma con el mismo nombre. A veces puede ser útil consultar la documentación de la plataforma, ya que la semántica de estas funciones varía entre plataformas.

Una explicación de algunas terminologías y convenciones está en orden.

- Una *epoch* es el punto en el cual el tiempo empieza, el valor retorno de la función `time.gmtime(0)`. El cual es Enero, 1, 1970, 00:00:00(UTC) en todas las plataformas.
- El término *seconds since the epoch* se refiere al número total de segundos transcurridos desde la época, excluyendo típicamente leap seconds [https://en.wikipedia.org/wiki/Leap_second] . Los segundos intercalares se excluyen de este total en todas las plataformas compatibles con POSIX.
- The functions in this module may not handle dates and times before the epoch or far in the future. The cut-off point in the future is determined by the C library; for 32-bit systems, it is typically in 2038.
- La función **`strptime()`** puede analizar años de 2 dígitos cuando se le da el código de formato `%y`. Cuando se analizan los años de 2 dígitos, se convierten de acuerdo con los

estándares POSIX e ISO C: los valores 69–99 se asignan a 1969–1999, y los valores 0–68 se asignan a 2000–2068.

- UTC is Coordinated Universal Time [https://en.wikipedia.org/wiki/Coordinated_Universal_Time] and superseded Greenwich Mean Time [https://en.wikipedia.org/wiki/Greenwich_Mean_Time] or GMT as the basis of international timekeeping. The acronym UTC is not a mistake but conforms to an earlier, language-agnostic naming scheme for time standards such as UT0, UT1, and UT2.

- El horario de verano es el horario de verano, un ajuste de la zona horaria por (generalmente) una hora durante parte del año. Las reglas DST son mágicas (determinadas por la ley local) y pueden cambiar de un año a otro. La biblioteca C tiene una tabla que contiene las reglas locales (a menudo se lee desde un archivo del sistema para flexibilidad) y es la única fuente de verdadera sabiduría en este sentido.

- La precisión de las diversas funciones en tiempo real puede ser inferior a la sugerida por las unidades en las que se expresa su valor o argumento. P.ej. En la mayoría de los sistemas Unix, el reloj «funciona» solo 50 o 100 veces por segundo.

- On the other hand, the precision of `time()` and `sleep()` is better than their Unix equivalents: times are expressed as floating-point numbers, `time()` returns the most accurate time available (using Unix `gettimeofday()` where available), and `sleep()` will accept a time with a nonzero fraction (Unix `select()` is used to implement this, where available).

- El valor de tiempo retornado por `gmtime()`, `localtime()`, y `strptime()`, y aceptado por `asctime()`, `mktime()` y `strftime()`, es una secuencia de 9 enteros. Los valores de retorno de `gmtime()`, `localtime()`, y `strptime()` también ofrecen nombres de atributos para campos individuales.

  Ver `struct_time` para una descripción de estos objetos.

Distinto en la versión 3.3: The `struct_time` type was extended to provide the `tm_gmtoff` and `tm_zone` attributes when platform supports corresponding `struct tm` members.

Distinto en la versión 3.6: The `struct_time` attributes `tm_gmtoff` and `tm_zone` are now available on all platforms.

- Use las siguientes funciones para convertir entre representaciones de tiempo:

Desde	A	Usar
segundos desde la época	`struct_time` en UTC	`gmtime()`
segundos desde la época	`struct_time` en hora local	`localtime()`
`struct_time` en UTC	segundos desde la época	`calendar.timegm()`
`struct_time` en hora local	segundos desde la época	`mktime()`

Las Funciones

time.asctime([*t*])

Convierta una tupla o `struct_time` que represente una hora retornada por `gmtime()` o `localtime()` en una cadena de la siguiente forma: `'Dom 20 de junio 23:21:05 1993'`. El campo del día tiene dos caracteres de largo y se rellena con espacio si el día es un solo dígito, por ejemplo: `'Miercoles  9 de Julio 04:26:40 1993'`.

Si no se proporciona *t*, se utiliza la hora actual retornada por `localtime()`. La información regional no es utilizada por `asctime()`.

Nota

A diferencia de la función C del mismo nombre, `asctime()` no agrega una nueva línea final.

time.pthread_getcpuclockid(*thread_id*)

Retorna el *clk_id* del reloj de tiempo de CPU específico del subproceso para el *thread_id* especificado.

Utilice `threading.get_ident()` o el atributo `ident` de objetos `threading.Thread` para obtener un valor adecuado para * thread_id*.

Advertencia

Pasar un * thread_id * no válido o caducado puede provocar un comportamiento indefinido, como un error de segmentación.

Availability: Unix

See the man page for *pthread_getcpuclockid(3)* *[https://manpages.debian.org/pthread_getcpuclockid(3)]* for further information.

Added in version 3.7.

time.clock_getres(*clk_id*)

Retorna la resolución (precisión) del reloj especificado *clk_id*. Consulte Constantes de ID de reloj para obtener una lista de los valores aceptados para *clk_id*.

Availability: Unix.

Added in version 3.3.

time.clock_gettime(*clk_id*) → float

Retorna la hora del reloj especificado *clk_id*. Consulte Constantes de ID de reloj para obtener una lista de los valores aceptados para *clk_id*.

Utilice `clock_gettime_ns()` para evitar la pérdida de precisión que causa el tipo `float`.

Availability: Unix.

Added in version 3.3.

time.clock_gettime_ns(*clk_id*) → int

Similar a **`clock_gettime()`** pero retorna el tiempo en nanosegundos.

Availability: Unix.

Added in version 3.7.

time.clock_settime(*clk_id*, *time: float*)

Establece la hora del reloj especificado *clk_id*. Actualmente, **`CLOCK_REALTIME`** es el único valor aceptado para *clk_id*.

Utilice **`clock_settime_ns()`** para evitar la pérdida de precisión que causa el tipo **`float`**.

Availability: Unix, not Android, not iOS.

Added in version 3.3.

time.clock_settime_ns(*clk_id*, *time: int*)

Similar a **`clock_settime()`** pero establece el tiempo con nanosegundos.

Availability: Unix, not Android, not iOS.

Added in version 3.7.

time.ctime([*secs*])

Convert a time expressed in seconds since the epoch to a string of a form: `'Sun Jun 20 23:21:05 1993'` representing local time. The day field is two characters long and is space padded if the day is a single digit, e.g.: `'Wed Jun  9 04:26:40 1993'`.

Si no se proporciona *secs* o **`None`**, se utiliza la hora actual retornada por **`time()`**. `ctime(secs)` es equivalente a `asctime(localtime(secs))`. La información de configuración regional no la utiliza **`ctime()`**.

time.get_clock_info(*name*)

Obtenga información sobre el reloj especificado como un objeto de espacio de nombres. Los nombres de reloj admitidos y las funciones correspondientes para leer su valor son:

- `'monotonic'`: **`time.monotonic()`**
- `'perf_counter'`: **`time.perf_counter()`**
- `'process_time'`: **`time.process_time()`**
- `'thread_time'`: **`time.thread_time()`**
- `'time'`: **`time.time()`**

El resultado tiene los siguientes atributos:

- *adjustable*: `True` if the clock can be set to jump forward or backward in time, `False` otherwise. Does not refer to gradual NTP rate adjustments.
- *implementation*: el nombre de la función C subyacente utilizada para obtener el valor del reloj. Consulte Constantes de ID de reloj para conocer los posibles valores.
- *monotonic*: `True` si el reloj no puede retroceder, `False` de lo contrario
- *resolution*: La resolución del reloj en segundos (**`float`**)

Added in version 3.3.

time.gmtime([*secs*])

Convert a time expressed in seconds since the epoch to a **`struct_time`** in UTC in which the dst flag is always zero. If *secs* is not provided or **`None`**, the current time as returned by **`time()`** is used. Fractions of a second are ignored. See above for a description of the **`struct_time`** object. See **`calendar.timegm()`** for the inverse of this function.

time.localtime([*secs*])

Como **`gmtime()`** pero se convierte a la hora local. Si no se proporciona *secs* o **`None`**, se utiliza la hora actual retornada por **`time()`**. El indicador *dst* se establece en `1` cuando DST se aplica al tiempo dado.

`localtime()` podría lanzar un `OverflowError`, si los valores de la marca de tiempo están fuera del rango soportado por las funciones **`localtime()`** o **`gmtime()`** de la plataforma C, y el `OSError` en las funciones **`localtime()`** o **`gmtime()`**. Es común que este esté restringido para los años entre 1970 y 2038.

time.mktime(*t*)

This is the inverse function of `localtime()`. Its argument is the `struct_time` or full 9-tuple (since the dst flag is needed; use `-1` as the dst flag if it is unknown) which expresses the time in *local* time, not UTC. It returns a floating-point number, for compatibility with `time()`. If the input value cannot be represented as a valid time, either `OverflowError` or `ValueError` will be raised (which depends on whether the invalid value is caught by Python or the underlying C libraries). The earliest date for which it can generate a time is platform-dependent.

time.monotonic() → float

Retorna el valor (en segundos fraccionarios) de un reloj monótono, es decir, un reloj que no puede retroceder. El reloj no se ve afectado por las actualizaciones del reloj del sistema. El punto de referencia del valor retornado no está definido, de modo que sólo la diferencia entre los resultados de dos llamadas es válida.

Clock:

- On Windows, call `QueryPerformanceCounter()` and `QueryPerformanceFrequency()`.
- On macOS, call `mach_absolute_time()` and `mach_timebase_info()`.
- On HP-UX, call `gethrtime()`.
- Call `clock_gettime(CLOCK_HIGHRES)` if available.
- Otherwise, call `clock_gettime(CLOCK_MONOTONIC)`.

Utilice `monotonic_ns()` para evitar la pérdida de precisión que causa el tipo `float`.

Added in version 3.3.

Distinto en la versión 3.5: The function is now always available and the clock is now the same for all processes.

Distinto en la versión 3.10: On macOS, the clock is now the same for all processes.

time.monotonic_ns() → int

Similar a `monotonic()`, pero el tiempo de retorno es en nanosegundos.

Added in version 3.7.

time.perf_counter() → float

Return the value (in fractional seconds) of a performance counter, i.e. a clock with the highest available resolution to measure a short duration. It does include time elapsed during sleep. The clock is the same for all processes. The reference point of the returned value is undefined, so that only the difference between the results of two calls is valid.

Detalles de implementación de CPython: On CPython, use the same clock as `time.monotonic()` and is a monotonic clock, i.e. a clock that cannot go backwards.

Utilice la función `perf_counter_ns()` para evitar la pérdida de precisión que causa el tipo `float`.

Added in version 3.3.

Distinto en la versión 3.10: On Windows, the clock is now the same for all processes.

Distinto en la versión 3.13: Use the same clock as `time.monotonic()`.

time.perf_counter_ns() → int

Similar a `perf_counter()`, pero el tiempo de retorno es en nanosegundos.

Added in version 3.7.

time.process_time() → float

Retorna el valor (en fracciones de segundo) de la suma del sistema y el tiempo de CPU del usuario del proceso actual. No incluye el tiempo transcurrido durante el sleep. Es todo el proceso por definición. El punto de referencia del valor retornado no está definido, de modo que sólo la diferencia entre los resultados de dos llamadas es válida.

Utilice `process_time_ns()` para evitar la pérdida de precisión que causa el tipo `float`.

Added in version 3.3.

time.process_time_ns() → int

Similar a `process_time()` pero retorna el tiempo en nanosegundos.

Added in version 3.7.

time.sleep(*secs*)

Suspend execution of the calling thread for the given number of seconds. The argument may be a floating-point number to indicate a more precise sleep time.

Si el sleep es interrumpido por una señal y ninguna excepción fue lanzada por el gestor de señales, este sleep es reiniciado con un tiempo límite recalculado.

El tiempo de suspensión podría ser mayor al requerido por un monto arbitrario, a raíz de la programación de otra actividad del sistema.

Windows implementation

On Windows, if *secs* is zero, the thread relinquishes the remainder of its time slice to any other thread that is ready to run. If there are no other threads ready to run, the function returns immediately, and the thread continues execution. On Windows 10 and newer the implementation uses a high-resolution timer [https://learn.microsoft.com/windows/win32/api/synchapi/nf-synchapi-createwaitabletimerexw] which provides resolution of 100 nanoseconds. If *secs* is zero, `Sleep(0)` is used.

Unix implementation

- Si está disponible, use `clock_nanosleep()` (resolución: 1 nanosegundo);
- O use `nanosleep()` si está disponible (resolución: 1 nanosegundo);
- O use `select()` (resolución: 1 microsegundo).

Nota

To emulate a «no-op», use **`pass`** instead of `time.sleep(0)`.

To voluntarily relinquish the CPU, specify a real-time scheduling policy and use **`os.sched_yield()`** instead.

Raises an auditing event `time.sleep` with argument `secs`.

Distinto en la versión 3.5: La función ahora duerme al menos * segundos * incluso si el sleep es interrumpido por una señal, excepto si el manejador de la señal genera una excepción (ver **PEP 475** [https://peps.python.org/pep-0475/] para la justificación).

Distinto en la versión 3.11: On Unix, the `clock_nanosleep()` and `nanosleep()` functions are now used if available. On Windows, a waitable timer is now used.

Distinto en la versión 3.13: Raises an auditing event.

time.strftime(*format*[, *t*])

Convierta una tupla o `struct_time` que represente un tiempo retornado por `gmtime()` o `localtime()` en una cadena como se especifica mediante el argumento *format*. Si no se proporciona *t*, se utiliza la hora actual retornada por `localtime()`. *format* debe ser una cadena. Se lanza un `ValueError` si algún campo en *t* está fuera del rango permitido.

0 es un argumento legal para cualquier posición en la tupla de tiempo; si normalmente es ilegal, el valor se fuerza a uno correcto.

Las siguientes directivas se pueden incrustar en la cadena *format*. Se muestran sin el ancho de campo opcional y la especificación de precisión, y se reemplazan por los caracteres indicados en el resultado `strftime()`:

Directiva	Significado	Notas
%a	Nombre abreviado del día local de la localidad.	
%A	Nombre completo del día laborable de Localidad.	
%b	Nombre abreviado del mes de la localidad.	
%B	Nombre completo del mes de la localidad.	
%c	Representación apropiada de fecha y hora de la localidad.	
%d	Día del mes como número decimal [01,31].	
%f	Microseconds as a decimal number [000000,999999].	(1)
%H	Hora (reloj de 24 horas) como un número decimal [00,23].	
%I	Hora (reloj de 12 horas) como un número decimal [01,12].	
%j	Día del año como número decimal [001,366].	
%m	Mes como un número decimal [01,12].	
%M	Minuto como un número decimal [00,59].	
%p	Equivalente de la configuración regional de AM o PM.	(2)
%S	Segunda como un número decimal [00,61].	(3)
%U	Número de semana del año (domingo como primer día de la semana) como número decimal [00,53]. Todos los días en un nuevo año anterior al primer domingo se consideran en la semana 0.	(4)
%u	Day of the week (Monday is 1; Sunday is 7) as a decimal number [1, 7].	

`%w`	Día de la semana como un número decimal [0 (domingo), 6].	
`%W`	Número de semana del año (lunes como primer día de la semana) como número decimal [00,53]. Todos los días en un nuevo año anterior al primer lunes se consideran en la semana 0.	(4)
`%x`	Representación de fecha apropiada de la localidad.	
`%X`	Representación del tiempo apropiado de la localidad.	
`%y`	Año sin siglo como número decimal [00,99].	
`%Y`	Año con siglo como número decimal.	
`%z`	Time zone offset indicating a positive or negative time difference from UTC/GMT of the form +HHMM or -HHMM, where H represents decimal hour digits and M represents decimal minute digits [-23:59, +23:59]. [1]	
`%Z`	Time zone name (no characters if no time zone exists). Deprecated. [1]	
`%G`	ISO 8601 year (similar to `%Y` but follows the rules for the ISO 8601 calendar year). The year starts with the week that contains the first Thursday of the calendar year.	
`%V`	ISO 8601 week number (as a decimal number [01,53]). The first week of the year is the one that contains the first Thursday of the year. Weeks start on Monday.	
`%%`	Un carácter literal `'%'`.	

Notas:

1. The `%f` format directive only applies to **`strptime()`**, not to **`strftime()`**. However, see also **`datetime.datetime.strptime()`** and **`datetime.datetime.strftime()`** where the `%f` format directive applies to microseconds.
2. Cuando se usa con la función **`strptime()`**, la directiva `%p` solo afecta el campo de hora de salida si se usa la directiva `%I` para analizar la hora.

1. El rango realmente es `0` a `61`; el valor `60` es válido en las marcas de tiempo que representan segundos intercalares [https://en.wikipedia.org/wiki/Leap_second] y el valor `61` es compatible por razones históricas.
2. Cuando se usa con la función **`strptime()`**, `%U` y `%W` solo se usan en los cálculos cuando se especifica el día

de la semana y el año.

Here is an example, a format for dates compatible with that specified in the **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html] Internet email standard. [1]

```
>>> from time import gmtime, strftime
>>> strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime(
'Thu, 28 Jun 2001 14:17:15 +0000'
```

Es posible que se admitan directivas adicionales en ciertas plataformas, pero solo las que se enumeran aquí tienen un significado estandarizado por ANSI C. Para ver el conjunto completo de códigos de formato admitidos en su plataforma, consulte la documentación de *strftime(3) [https://manpages.debian.org/strftime(3)]*.

En algunas plataformas, una especificación de precisión y ancho de campo opcional puede seguir inmediatamente el `'%'` inicial de una directiva en el siguiente orden; Esto tampoco es portátil. El ancho del campo es normalmente 2 excepto `%j` donde es 3.

time.strptime(*string*[, *format*])

Analiza una cadena que representa un tiempo de acuerdo con un formato. El valor de retorno es a `struct_time` como lo retorna `gmtime()` o `localtime()`.

El parámetro *format* utiliza las mismas directivas que las utilizadas por `strftime()`; el valor predeterminado es `"%a %b %d %H:%M:%S %Y"` que coincide con el formato retornado por `ctime()`. Si *string* no se puede analizar de acuerdo con *format*, o si tiene un exceso de datos después del análisis, se lanza un `ValueError`. Los valores predeterminados utilizados para completar los datos faltantes cuando no se pueden inferir valores más precisos son `(1900, 1, 1, 0, 0, 0, 0, 1, -1)`. Tanto *string* como *format* deben ser strings.

Por ejemplo:

```
>>> import time
>>> time.strptime("30 Nov 00", "%d %b %y")
time.struct_time(tm_year=2000, tm_mon=11, tm_mday=3
                 tm_sec=0, tm_wday=3, tm_yday=335,
```

La compatibilidad con la directiva `%Z` se basa en los valores contenidos en `tzname` y en si `daylight` es verdadero. Debido a esto, es específico de la plataforma, excepto para reconocer UTC y GMT que siempre se conocen (y se consideran zonas horarias que no son de horario de verano).

Solo se admiten las directivas especificadas en la documentación. Debido a que `strftime()` se implementa por plataforma, a veces puede ofrecer más directivas que las enumeradas. Pero `strptime()` es independiente de cualquier plataforma y, por lo tanto, no necesariamente admite todas las directivas disponibles que no están documentadas como compatibles.

class time.struct_time

El tipo de secuencia de valores de tiempo retornado por **`gmtime()`**, **`localtime()`**, y **`strptime()`**. Es un objeto con una interfaz named tuple: se puede acceder a los valores por índice y por nombre de atributo. Los siguientes valores están presentes:

Índice	Atributo	Valores
0	tm_year	(por ejemplo, 1993)
1	tm_mon	rango [1, 12]
2	tm_mday	rango [1, 31]
3	tm_hour	rango [0, 23]
4	tm_min	rango [0, 59]
5	tm_sec	rango [0, 61]; see Note (2) in **`strftime()`**
6	tm_wday	rango [0, 6]; Monday is 0
7	tm_yday	rango [1, 366]
8	tm_isdst	0, 1 ó -1; ver abajo
N/A	tm_zone	abreviatura del nombre de la zona horaria
N/A	tm_gmtoff	desplazamiento al este de UTC en segundos

Tenga en cuenta que, a diferencia de la estructura C, el valor del mes es un rango de [1, 12], no [0, 11].

En llamadas a **`mktime()`**, **`tm_isdst`** puede establecerse en 1 cuando el horario de verano está vigente y 0 cuando no lo está. Un valor de -1 indica que esto no se conoce y, por lo general, se completará el estado correcto.

Cuando una tupla con una longitud incorrecta se pasa a una función que espera a **`struct_time`**, o que tiene elementos del tipo incorrecto, se lanza un **`TypeError`**.

time.time() → float

Return the time in seconds since the epoch as a floating-point number. The handling of leap seconds [https://en.wikipedia.org/wiki/Leap_second] is platform dependent. On Windows and most Unix systems, the leap seconds are not counted towards the time in seconds since the epoch. This is commonly referred to as Unix time [https://en.wikipedia.org/wiki/Unix_time].

Note that even though the time is always returned as a floating-point number, not all systems provide time with a better precision than 1 second. While this function normally returns non-decreasing values, it can return a lower value than a previous call if the system clock has been set back between the two calls.

El número retornado por **`time()`** se puede convertir a un formato de hora más común (es decir, año, mes, día, hora, etc.) en UTC pasándolo a función **`gmtime()`** o en hora local pasándola a la función **`localtime()`**. En ambos casos se retorna un objeto **`struct_time`**, desde el cual se puede acceder a los componentes de la fecha del calendario como atributos.

Clock:

- On Windows, call `GetSystemTimePreciseAsFileTime()`.
- Call `clock_gettime(CLOCK_REALTIME)` if available.

- Otherwise, call `gettimeofday()`.

Utilice **`time_ns()`** para evitar la pérdida de precisión que causa el tipo **`float`**.

Distinto en la versión 3.13: On Windows, calls `GetSystemTimePreciseAsFileTime()` instead of `GetSystemTimeAsFileTime()`.

time.time_ns() → int

Similar a **`time()`** pero retorna el tiempo como un número entero de nanosegundos desde la época.

Added in version 3.7.

time.thread_time() → float

Retorna el valor (en fracciones de segundo) de la suma del sistema y el tiempo de CPU del usuario del subproceso actual. No incluye el tiempo transcurrido durante el sleep. Es específico del hilo por definición. El punto de referencia del valor retornado no está definido, de modo que sólo la diferencia entre los resultados de dos llamadas en el mismo hilo es válida.

Utilice **`thread_time_ns()`** para evitar la pérdida de precisión que causa el tipo **`float`**.

Availability: Linux, Unix, Windows.

Sistemas Unix que soporten `CLOCK_THREAD_CPUTIME_ID`.

Added in version 3.7.

time.thread_time_ns() → int

Similar a **`thread_time()`** pero retorna el tiempo en nanosegundos.

Added in version 3.7.

time.tzset()

Restablezca las reglas de conversión de tiempo utilizadas por las rutinas de la biblioteca. La variable de entorno **`TZ`** especifica cómo se hace esto. También establecerá las variables `tzname` (de variable de entorno **`TZ`**), `timezone` (segundos que no son DST al oeste de UTC), `altzone` (segundos DST al oeste de UTC) y `daylight` (a 0 si esta zona horaria no tiene ninguna regla de horario de verano, o a cero si hay un horario pasado, presente o futuro cuando se aplica el horario de verano).

Availability: Unix.

Nota

Aunque en muchos casos, cambiar la variable de entorno **`TZ`** puede afectar la salida de funciones como **`localtime()`** sin llamar a **`tzset()`**, no se debe confiar en este comportamiento.

La variable de entorno **`TZ`** no debe contener espacios en blanco.

El formato estándar de la variable de entorno **`TZ`** es (espacio en blanco agregado para mayor claridad):

```
std offset [dst [offset [,start[/time], end[/time]]
```

Donde están los componentes:

`std` y `dst`

Tres o más caracteres alfanuméricos que dan las abreviaturas de zona horaria. Estos se propagarán en time.tzname

`offset`

El desplazamiento tiene la forma: `± hh[:mm[:ss]]`. Esto indica el valor agregado de la hora local para llegar a UTC. Si está precedido por un “-”, la zona horaria está al este del primer meridiano; de lo contrario, es oeste. Si no hay desplazamiento después de dst, se supone que el horario de verano es una hora antes del horario estándar.

`start[/time], end[/time]`

Indica cuándo cambiar hacia y desde DST. El formato de las fechas de inicio y finalización es uno de los siguientes:

`J`*`n`*

El día juliano * n * (1 < = * n * < = 365). Los días bisiestos no se cuentan, por lo que en todos los años el 28 de febrero es el día 59 y el 1 de marzo es el día 60.

`n`

El día juliano basado en cero (0 < = * n * < = 365). Los días bisiestos se cuentan y es posible referirse al 29 de febrero.

`M`*`m`*`.`*`n`*`.`*`d`*

El día d (0 < = *d* < = 6) de la semana *n* del mes *m* del año (1 < = *n* < = 5, 1 < = *m* < = 12, donde la semana 5 significa «el último *d* día del mes *m*», que puede ocurrir en la cuarta o quinta semana). La semana 1 es la primera semana en la que ocurre el día *d*. El día cero es un domingo.

`time` tiene el mismo formato que `offset`, excepto que no se permite ningún signo inicial ("-" o "+"). El valor predeterminado, si no se da el tiempo, es 02:00:00.

```
>>> os.environ['TZ'] = 'EST+05EDT,M4.1.0,M10.5.0'
>>> time.tzset()
>>> time.strftime('%X %x %Z')
'02:07:36 05/08/03 EDT'
>>> os.environ['TZ'] = 'AEST-10AEDT-11,M10.5.0,M3.5
>>> time.tzset()
>>> time.strftime('%X %x %Z')
```

```
'16:08:12 05/08/03 AEST'
```

En muchos sistemas Unix (incluidos *BSD, Linux, Solaris y Darwin), es más conveniente utilizar la base de datos *zoneinfo* (*tzfile(5)* *[https://manpages.debian.org/tzfile(5)]*) del sistema para especificar las reglas de zona horaria. Para hacer esto, configure la variable de entorno **`TZ`** en la ruta del archivo de datos de zona horaria requerida, en relación con la raíz de la base de datos de zona horaria "zoneinfo" de los sistemas, generalmente ubicada en `/usr/share/zoneinfo`. Por ejemplo, `'US/Eastern'`, `'Australia/Melbourne'`, `'Egypt'` o `'Europe/Amsterdam'`.

```
>>> os.environ['TZ'] = 'US/Eastern'
>>> time.tzset()
>>> time.tzname
('EST', 'EDT')
>>> os.environ['TZ'] = 'Egypt'
>>> time.tzset()
>>> time.tzname
('EET', 'EEST')
```

Constantes de ID de reloj

Estas constantes se utilizan como parámetros para **`clock_getres()`** y **`clock_gettime()`**.

time.CLOCK_BOOTTIME

Idéntico a **`CLOCK_MONOTONIC`**, excepto que también incluye cualquier momento en que el sistema esté suspendido.

Esto permite que las aplicaciones obtengan un reloj monótono con suspensión sin tener que lidiar con las complicaciones de **`CLOCK_REALTIME`**, que puede tener discontinuidades si se cambia la hora usando `settimeofday()` o similar.

Availability: Linux > = 2.6.39.

Added in version 3.7.

time.CLOCK_HIGHRES

El sistema operativo Solaris tiene un temporizador `CLOCK_HIGHRES` que intenta utilizar una fuente de hardware óptima y puede brindar una resolución cercana a los nanosegundos. `CLOCK_HIGHRES` es el reloj de alta resolución no ajustable.

Availability: Solaris.

Added in version 3.3.

time.CLOCK_MONOTONIC

Reloj que no se puede configurar y representa el tiempo monótono desde algún punto de partida no especificado.

Availability: Unix.

Added in version 3.3.

time.CLOCK_MONOTONIC_RAW

Similar a **`CLOCK_MONOTONIC`**, pero proporciona acceso a un tiempo sin procesar basado en hardware que no está sujeto a ajustes NTP.

Availability: Linux > = 2.6.28, macOS > = 10.12.

Added in version 3.3.

time.CLOCK_MONOTONIC_RAW_APPROX

Similar to **`CLOCK_MONOTONIC_RAW`**, but reads a value cached by the system at context switch and hence has less accuracy.

Availability: macOS > = 10.12.

Added in version 3.13.

time.CLOCK_PROCESS_CPUTIME_ID

Temporizador por proceso de alta resolución desde la CPU.

Availability: Unix.

Added in version 3.3.

time.CLOCK_PROF

Temporizador por proceso de alta resolución desde la CPU.

Availability: FreeBSD, NetBSD > = 7, OpenBSD.

Added in version 3.7.

time.CLOCK_TAI

International Atomic Time [https://www.nist.gov/pml/time-and-frequency-division/how-utcnist-related-coordinated-universal-time-utc-international]

El sistema debe contar con una tabla de segundos intercalares para que éste dé la respuesta correcta. Software PTP o NTP puede mantener una tabla de segundos intercalares.

Availability: Linux.

Added in version 3.9.

time.CLOCK_THREAD_CPUTIME_ID

Reloj de tiempo de CPU específico de subproceso.

Availability: Unix.

Added in version 3.3.

time.CLOCK_UPTIME

Tiempo cuyo valor absoluto es el tiempo que el sistema ha estado funcionando y no suspendido, proporcionando una medición precisa del tiempo de actividad, tanto absoluta como de intervalo.

Availability: FreeBSD, OpenBSD > = 5.5.

Added in version 3.7.

time.CLOCK_UPTIME_RAW

Reloj que se incrementa monótonamente, rastreando el tiempo desde un punto arbitrario, no afectado por los ajustes de frecuencia o tiempo y no incrementado mientras el sistema está dormido.

Availability: macOS > = 10.12.

Added in version 3.8.

time.CLOCK_UPTIME_RAW_APPROX

Like **`CLOCK_UPTIME_RAW`**, but the value is cached by the system at context switches and therefore has less accuracy.

Availability: macOS > = 10.12.

Added in version 3.13.

La siguiente constante es el único parámetro que se puede enviar a **`clock_settime()`**.

time.CLOCK_REALTIME

Real-time clock. Setting this clock requires appropriate privileges. The clock is the same for all processes.

Availability: Unix.

Added in version 3.3.

Constantes de zona horaria

time.altzone

El desplazamiento de la zona horaria de horario de verano local, en segundos al oeste de UTC, si se define uno. Esto es negativo si la zona horaria local del horario de verano está al este de UTC (como en Europa occidental, incluido el Reino Unido). Solo use esto si la `daylight` no es cero. Vea la nota abajo.

time.daylight

No es cero si se define una zona horaria DST. Vea la nota abajo.

time.timezone

El desplazamiento de la zona horaria local (no DST), en segundos al oeste de UTC (negativo en la mayoría de Europa occidental, positivo en los EE. UU., Cero en el Reino Unido). Vea la nota abajo.

time.tzname

Una tupla de dos cadenas: la primera es el nombre de la zona horaria local no DST, la segunda es el nombre de la zona horaria local DST. Si no se define la zona horaria DST, la segunda cadena no debe usarse. Vea la nota abajo.

Nota

For the above Timezone constants (`altzone`, `daylight`, `timezone`, and `tzname`), the value is determined by the timezone rules in effect at module load time or the last time `tzset()` is called and may be incorrect for times in the past. It is recommended to use the `tm_gmtoff` and `tm_zone` results from `localtime()` to obtain timezone information.

Ver también

Modulo `datetime`

Más interfaz orientada a objetos para fechas y horas.

Modulo `locale`

Servicios de internacionalización. La configuración regional afecta la interpretación de muchos especificadores de formato en `strftime()` y `strptime()`.

Modulo `calendar`

Funciones generales relacionadas con el calendario. `timegm()` es el inverso de `gmtime()` de este módulo.

Notas al pie

The use of `%Z` is now deprecated, but the `%z` escape that expands to the preferred hour/minute offset is not supported by all ANSI C libraries. Also, a strict reading of the original 1982 **RFC 822** [https://datatracker.ietf.org/doc/html/rfc822.html] standard calls for a two-digit year (`%y` rather than `%Y`), but practice moved to 4-digit years long before the year 2000. After that, **RFC 822** [https://datatracker.ietf.org/doc/html/rfc822.html] became obsolete and the 4-digit year has been first recommended by **RFC 1123** [https://datatracker.ietf.org/doc/html/rfc1123.html] and then mandated by **RFC 2822** [https://datatracker.ietf.org/doc/html/rfc2822.html], with **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html] continuing this requirement.

`logging` — Logging facility for Python

Source code: Lib/logging/__init__.py [https://github.com/python/cpython/tree/3.14/Lib/logging/__init__.py]

Important

Esta página contiene la información de referencia de la API. Para información sobre tutorial y discusión de temas más avanzados, ver

- Tutorial básico
- Tutorial avanzado
- Libro de recetas de Logging

Este módulo define funciones y clases que implementan un sistema flexible de logging de eventos para aplicaciones y bibliotecas.

El beneficio clave de tener la API de logging proporcionada por un módulo de la biblioteca estándar es que todos los módulos de Python pueden participar en el logging, por lo que el registro de su aplicación puede incluir sus propios mensajes integrados con mensajes de módulos de terceros.

Here’s a simple example of idiomatic usage:

```
# myapp.py
import logging
import mylib
logger = logging.getLogger(__name__)

def main():
    logging.basicConfig(filename='myapp.log', level=logg
    logger.info('Started')
    mylib.do_something()
    logger.info('Finished')

if __name__ == '__main__':
    main()

# mylib.py
import logging
logger = logging.getLogger(__name__)

def do_something():
    logger.info('Doing something')
```

If you run *myapp.py*, you should see this in *myapp.log*:

```
INFO:__main__:Started
INFO:mylib:Doing something
INFO:__main__:Finished
```

The key feature of this idiomatic usage is that the majority of code is simply creating a module level logger with `getLogger(__name__)`, and using that logger to do any needed logging. This is concise, while allowing downstream code fine-grained control if needed. Logged messages to the module-level logger get forwarded to handlers of loggers in higher-level modules, all the way up to the highest-level logger known as the root logger; this approach is known as hierarchical logging.

For logging to be useful, it needs to be configured: setting the levels and destinations for each logger, potentially changing how specific modules log, often based on command-line arguments or application configuration. In most cases, like the one above, only the root logger needs to be so configured, since all the lower level loggers at module level eventually forward their messages to its handlers. **`basicConfig()`** provides a quick way to configure the root logger that handles many use cases.

El módulo proporciona mucha funcionalidad y flexibilidad. Si no está familiarizado con logging, la mejor manera de familiarizarse con él es ver los tutoriales (**vea los enlaces arriba a la derecha**).

The basic classes defined by the module, together with their

attributes and methods, are listed in the sections below.

- Los loggers exponen la interfaz que el código de la aplicación usa directamente.
- Los gestores envían los registros (creados por los loggers) al destino apropiado.
- Los filtros proporcionan una facilidad de ajuste preciso para determinar que registros generar.
- Los formateadores especifican el diseño de los registros en el resultado final.

Objetos logger

Los loggers tienen los siguientes atributos y métodos. Tenga en cuenta que los loggers *NUNCA* deben ser instanciados directamente, siempre a través de la función de nivel de módulo `logging.getLogger(name)`. Múltiples llamadas a **`getLogger()`** con el mismo nombre siempre retornarán una referencia al mismo objeto Logger.

The `name` is potentially a period-separated hierarchical value, like `foo.bar.baz` (though it could also be just plain `foo`, for example). Loggers that are further down in the hierarchical list are children of loggers higher up in the list. For example, given a logger with a name of `foo`, loggers with names of `foo.bar`, `foo.bar.baz`, and `foo.bam` are all descendants of `foo`. In addition, all loggers are descendants of the root logger. The logger name hierarchy is analogous to the Python package hierarchy, and identical to it if you organise your loggers on a per-module basis using the recommended construction `logging.getLogger(__name__)`. That's because in a module, `__name__` is the module's name in the Python package namespace.

class logging.Logger

name

This is the logger's name, and is the value that was passed to **`getLogger()`** to obtain the logger.

Nota

This attribute should be treated as read-only.

level

The threshold of this logger, as set by the `setLevel()` method.

Nota

Do not set this attribute directly - always use `setLevel()`, which has checks for the level passed to it.

parent

The parent logger of this logger. It may change based on later instantiation of loggers which are higher up in the namespace hierarchy.

Nota

This value should be treated as read-only.

propagate

Si este atributo se evalúa como verdadero, los eventos registrados en este logger se pasarán a los gestores de los loggers de nivel superior (ancestro), además de los gestores asociados a este logger. Los mensajes se pasan directamente a los gestores de los loggers ancestrales; no se consideran ni el nivel ni los filtros de los loggers ancestrales en cuestión.

Si esto se evalúa como falso, los mensajes de registro no se pasan a los gestores de los logger ancestrales.

Explicándolo con un ejemplo: Si el atributo propagado del logger llamado `A.B.C` se evalúa a true, cualquier evento registrado en `A.B.C` a través de una llamada a un método como `logging.getLogger('A.B.C').error(...)` será [sujeto a pasar el nivel de ese logger y la configuración del filtro] pasado a su vez a cualquier manejador adjunto a los loggers llamados `A.B`, `A` y al logger raíz, después de ser pasado primero a cualquier manejador adjunto a `A.B.C`. Si cualquier logger en la cadena `A.B.C`, `A.B`, `A` tiene su atributo `propagate` a false, entonces ese es el último logger a cuyos manejadores se les ofrece el evento a manejar, y la propagación se detiene en ese punto.

El constructor establece este atributo en `True`.

Nota

Si adjunta un gestor a un logger *y* uno o más de sus ancestros, puede emitir el mismo registro varias veces. En general, no debería necesitar adjuntar un gestor a más de un logger; si solo lo adjunta al logger apropiado que está más arriba en la jerarquía del logger, verá todos los eventos registrados por todos los logger descendientes, siempre que la configuración de propagación sea `True`. Un escenario común es adjuntar gestores solo al logger raíz y dejar que la propagación se encargue del resto.

handlers

The list of handlers directly attached to this logger instance.

Nota

This attribute should be treated as read-only; it is normally changed via the **`addHandler()`** and **`removeHandler()`** methods, which use locks to ensure thread-safe operation.

disabled

This attribute disables handling of any events. It is set to `False` in the initializer, and only changed by logging configuration code.

Nota

This attribute should be treated as read-only.

setLevel(*level*)

Establece el umbral para este logger en *level*. Los mensajes de logging que son menos severos que *level* serán ignorados; los mensajes de logging que tengan un nivel de severidad *level* o superior serán emitidos por cualquier gestor o gestores que atiendan este logger, a menos que el nivel de un gestor haya sido configurado en un nivel de severidad más alto que *level*.

Cuando se crea un logger, el nivel se establece en **`NOTSET`** (que hace que todos los mensajes se procesen cuando el logger es el logger raíz, o la delegación al padre cuando el logger no es un logger raíz). Tenga en cuenta que el logger raíz se crea con el nivel **`WARNING`**.

El término "delegación al padre" significa que si un logger tiene un nivel de NOTSET, su cadena de logger ancestrales se atraviesa hasta que se encuentra un ancestro con un nivel diferente a NOTSET o se alcanza la raíz.

Si se encuentra un antepasado con un nivel distinto de NOTSET, entonces el nivel de ese antepasado se trata como el nivel efectivo del logger donde comenzó la búsqueda de antepasados, y se utiliza para determinar cómo se maneja un evento de registro.

Si se alcanza la raíz y tiene un nivel de NOTSET, se procesarán todos los mensajes. De lo contrario, el nivel de la raíz se utilizará como el nivel efectivo.

Ver Niveles de logging para obtener una lista de niveles.

Distinto en la versión 3.2: El parámetro *level* ahora acepta una representación de cadena del nivel como "INFO" como alternativa a las constantes de enteros como **`INFO`**. Sin embargo, tenga en cuenta que los niveles se almacenan internamente como e.j. **`getEffectiveLevel()`** y **`isEnabledFor()`** retornará/esperará que se pasen enteros.

isEnabledFor(*level*)

Indica si este logger procesará un mensaje de gravedad *level*. Este método verifica primero el nivel de nivel de módulo establecido por `logging.disable(level)` y luego el nivel efectivo del logger según lo determinado por **`getEffectiveLevel()`**.

getEffectiveLevel()

Indica el nivel efectivo para este logger. Si se ha establecido un valor distinto de **`NOTSET`** utilizando **`setLevel()`**, se retorna. De lo contrario, la jerarquía se atraviesa hacia la raíz hasta que se encuentre un valor que no sea **`NOTSET`** y se retorna ese valor. El valor retornado es un entero, típicamente uno de **`logging.DEBUG`**, **`logging.INFO`** etc.

getChild(*suffix*)

Retorna un logger que es descendiente de este logger, según lo determinado por el sufijo. Por lo tanto, `logging.getLogger('abc').getChild('def.ghi')` retornaría el mismo logger que retornaría `logging.getLogger('abc.def.ghi')`. Este es un método convenientemente útil cuando el logger principal se nombra usando e.j. `__name__` en lugar de una cadena literal.

Added in version 3.2.

getChildren()

Retorna un conjunto de loggers que son hijos inmediatos de este logger. Así, por ejemplo, `logging.getLogger().getChildren()` podría retornar un conjunto que contenga loggers llamados `foo` y `bar`, pero un logger llamado `foo.bar` no estaría incluido en el conjunto. Del mismo modo, `logging.getLogger('foo').getChildren()` podría retornar un conjunto que incluyera un logger llamado `foo.bar`, pero no incluiría uno llamado `foo.bar.baz`.

Added in version 3.12.

debug(*msg*, **args*, ***kwargs*)

Registra un mensaje con el nivel **`DEBUG`** en este logger. El *msg* es la cadena de formato del mensaje, y los *args* son los argumentos que se fusionan en *msg* utilizando el operador de formato de cadena. (Tenga en cuenta que esto significa que puede usar palabras clave en la cadena de formato, junto con un solo argumento de diccionario). No se realiza ninguna operación de formateo mediante % en *msg* cuando no se proporcionan *args*.

Hay cuatro argumentos de palabras clave *kwargs* que se inspeccionan: *exc_info*, *stack_info*, *stacklevel* y *extra*.

Si *exc_info* no se evalúa como falso, hace que se agregue información de excepción al mensaje de registro. Si se proporciona una tupla de excepción (en el formato retornado por **`sys.exc_info()`**) o se proporciona una instancia de excepción, se utiliza; de lo contrario, se llama a **`sys.exc_info()`** para obtener la información de excepción.

El segundo argumento opcional con la palabra clave *stack_info*, que por defecto es `False`. Si es verdadero, la información de la pila agregara el mensaje de registro, incluida la actual llamada del registro. Tenga en cuenta que esta no es la misma información de la pila que se muestra al especificar *exc_info*: la primera son los cuadros de la pila desde la parte inferior de la pila hasta la llamada de registro en el hilo actual, mientras que la segunda es la información sobre los cuadros de la pila que se han desenrollado, siguiendo una excepción, mientras busca gestores de excepción.

Puede especificar *stack_info* independientemente de *exc_info*, por ejemplo solo para mostrar cómo llegaste a cierto punto en tu código, incluso cuando no se lanzaron excepciones. Los marcos de la pila se imprimen siguiendo una línea de encabezado que dice:

```
Stack (most recent call last):
```

Esto imita el `Traceback (most recent call last):` que se usa cuando se muestran marcos de excepción.

El tercer argumento opcional con la palabra clave *stacklevel*, que por defecto es `1`. Si es mayor que 1, se omite el número correspondiente de cuadros de pila al calcular el número de línea y el nombre de la función establecidos en **`LogRecord`** creado para el evento de registro. Esto se puede utilizar en el registro de ayudantes para que el nombre de la función, el nombre de archivo y el número de línea registrados no sean la información para la función/método auxiliar, sino más bien su llamador. El nombre de este parámetro refleja el equivalente en el modulo **`warnings`**.

The fourth keyword argument is *extra* which can be used to pass a dictionary which is used to populate the **`__dict__`** of the **`LogRecord`** created for the logging event with user-defined attributes. These custom attributes can then be used as you like. For example, they could be incorporated into logged messages. For example:

```
FORMAT = '%(asctime)s %(clientip)-15s %(user)-8
logging.basicConfig(format=FORMAT)
d = {'clientip': '192.168.0.1', 'user': 'fblogg
logger = logging.getLogger('tcpserver')
logger.warning('Protocol problem: %s', 'connect
```

imprimiría algo como

```
2006-02-08 22:20:02,165 192.168.0.1 fbloggs  Pr
```

Las claves en el diccionario pasado *extra* no deben entrar en conflicto con las claves utilizadas por el sistema de registro. (Ver la documentación de Atributos LogRecord para obtener más información sobre qué claves utiliza el sistema de registro).

Si elige usar estos atributos en los mensajes registrados, debe tener cuidado. En el ejemplo anterior, se ha configurado **`Formatter`** con una cadena de formato que espera *clientip* y "usuario" en el diccionario de atributos de **`LogRecord`**. Si faltan, el mensaje no se registrará porque se producirá una excepción de formato de cadena. En este caso, siempre debe pasar el diccionario *extra* con estas teclas.

Si bien esto puede ser molesto, esta función está diseñada para su uso en circunstancias especializadas, como servidores de subprocesos múltiples donde el mismo código se ejecuta en muchos contextos, y las condiciones interesantes que surgen dependen de este contexto (como la dirección IP del cliente remoto y autenticado nombre de usuario, en el ejemplo anterior). En tales circunstancias, es probable que se especialice **`Formatter`**s con particular **`Handler`**s.

If no handler is attached to this logger (or any of its ancestors, taking into account the relevant **`Logger.propagate`** attributes), the message will be sent to the handler set on `lastResort`.

Distinto en la versión 3.2: Se agregó el parámetro *stack_info*.

Distinto en la versión 3.5: El parámetro *exc_info* ahora puede aceptar instancias de excepción.

Distinto en la versión 3.8: Se agregó el parámetro *stacklevel*.

info(*msg*, **args*, ***kwargs*)

Registra un mensaje con el nivel `INFO` en este logger. Los argumentos se interpretan como `debug()`.

warning(*msg*, **args*, ***kwargs*)

Registra un mensaje con el nivel `WARNING` en este logger. Los argumentos se interpretan como `debug()`.

Nota

Hay un método obsoleto `warn` que es funcionalmente idéntico a `warning`. Como `warn` está en desuso, no lo use, use `warning` en su lugar.

error(*msg*, **args*, ***kwargs*)

Registra un mensaje con nivel `ERROR` en este logger. Los argumentos se interpretan como `debug()`.

critical(*msg*, **args*, ***kwargs*)

Registra un mensaje con el nivel `CRITICAL` en este logger. Los argumentos se interpretan como `debug()`.

log(*level*, *msg*, **args*, ***kwargs*)

Registra un mensaje con nivel entero *level* en este logger. Los otros argumentos se interpretan como `debug()`.

exception(*msg*, **args*, ***kwargs*)

Registra un mensaje con nivel **`ERROR`** en este logger. Los argumentos se interpretan como **`debug()`**. La información de excepción se agrega al mensaje de registro. Este método solo debe llamarse desde un gestor de excepciones.

addFilter(*filter*)

Agrega el filtro *filter* especificado a este logger.

removeFilter(*filter*)

Elimina el filtro *filter* especificado de este logger.

filter(*record*)

Aplique los filtros de este logger al registro y retorna `True` si se va a procesar el registro. Los filtros se consultan a su vez, hasta que uno de ellos retorna un valor falso. Si ninguno de ellos retorna un valor falso, el registro será procesado (pasado a los gestores). Si se retorna un valor falso, no se produce más procesamiento del registro.

addHandler(*hdlr*)

Agrega el gestor especificado *hdlr* a este logger.

removeHandler(*hdlr*)

Elimina el gestor especificado *hdlr* de este logger.

findCaller(*stack_info=False*, *stacklevel=1*)

Encuentra el nombre de archivo de origen de la invoca y el número de línea. Retorna el nombre del archivo, el número de línea, el nombre de la función y la información de la pila como una tupla de 4 elementos. La información de la pila se retorna como `None` a menos que *stack_info* sea `True`.

El parámetro *stacklevel* se pasa del código que llama a **`debug()`** y otras API. Si es mayor que 1, el exceso se utiliza para omitir los cuadros de la pila antes de determinar los valores que se retornarán. Esto generalmente será útil al llamar a las API de registro desde un *helper/wrapper*, de modo que la información en el registro de eventos no se refiera al *helper/wrapper*, sino al código que lo llama.

handle(*record*)

Gestiona un registro pasándolo a todos los gestores asociados con este logger y sus antepasados (hasta que se encuentre un valor falso de *propagar*). Este método se utiliza para registros no empaquetados recibidos de un socket, así como para aquellos creados localmente. El filtrado a nivel de logger se aplica usando **`filter()`**.

makeRecord(*name*, *level*, *fn*, *lno*, *msg*, *args*, *exc_info*, *func=None*, *extra=None*, *sinfo=None*)

Este es un método *factory* que se puede sobreescribir en subclases para crear instancias especializadas **`LogRecord`**.

hasHandlers()

Comprueba si este logger tiene algún gestor configurado. Esto se hace buscando gestores en este logger y sus padres en la jerarquía del logger. Retorna `True` si se encontró un gestor, de lo contrario, `False`. El método deja de buscar en la jerarquía cada vez que se encuentra un logger con el atributo *propagate* establecido en falso: ese será el último logger que verificará la existencia de gestores.

Added in version 3.2.

Distinto en la versión 3.7: Los logger ahora se pueden serializar y deserializar (*pickled and unpickled*).

Niveles de logging

Los valores numéricos de los niveles de logging se dan en la siguiente tabla. Estos son principalmente de interés si desea definir sus propios niveles y necesita que tengan valores específicos en relación con los niveles predefinidos. Si define un nivel con el mismo valor numérico, sobrescribe el valor predefinido; se pierde el nombre predefinido.

Nivel	Valor numérico	Qué significa / Cuándo utilizarlo
logging.NOTSET	0	Cuando se establece en un logger, indica que los loggers antecesores deben ser consultados para determinar el nivel efectivo. Si el resultado sigue siendo **`NOTSET`**, se registrarán todos los eventos. Cuando se establece en un gestor, todos los eventos son gestionados.
logging.DEBUG	10	Información detallada, normalmente sólo de interés para un desarrollador que intenta diagnosticar un problema.
logging.INFO	20	Confirmación de que todo funciona según lo previsto.
logging.WARNING	30	Una indicación de que ha ocurrido algo inesperado o de que podría producirse un problema en un futuro próximo (por ejemplo, «espacio en disco bajo»). El software sigue funcionando como se esperaba.
logging.ERROR	40	Debido a un problema más grave, el software no ha podido realizar alguna función.
logging.CRITICAL	50	Un error grave, que indica que es posible que el propio programa no pueda seguir ejecutándose.

Gestor de objetos

Los gestores tienen los siguientes atributos y métodos. Tenga en cuenta que `Handler` nunca se instancia directamente; esta clase actúa como base para subclases más útiles. Sin embargo, el método **`__init__()`** en las subclases debe llamar a `Handler.__init__()`.

class logging.Handler

init(*level=NOTSET*)

Inicializa la instancia `Handler` estableciendo su nivel, configurando la lista de filtros en la lista vacía y creando un bloqueo (usando `createLock()`) para serializar el acceso a un mecanismo de E/S.

createLock()

Inicializa un bloqueo de subprocesos que se puede utilizar para serializar el acceso a la funcionalidad de E/S subyacente que puede no ser segura para subprocesos.

acquire()

Adquiere el bloqueo de hilo creado con `createLock()`.

release()

Libera el bloqueo de hilo adquirido con `acquire()`.

setLevel(*level*)

Establece el umbral para este gestor en *level*. Los mensajes de registro que son menos severos que *level* serán ignorados. Cuando se crea un gestor, el nivel se establece en `NOTSET` (lo que hace que se procesen todos los mensajes).

Ver Niveles de logging para obtener una lista de niveles.

Distinto en la versión 3.2: El parámetro *level* ahora acepta una representación de cadena del nivel como “INFO” como alternativa a las constantes de enteros como `INFO`.

setFormatter(*fmt*)

Sets the formatter for this handler to *fmt*. The *fmt* argument must be a `Formatter` instance or `None`.

addFilter(*filter*)

Agrega el filtro *filter* especificado a este gestor.

removeFilter(*filter*)

Elimina el filtro especificado *filter* de este gestor.

filter(*record*)

Aplique los filtros de este gestor al registro y retorna `True` si se va a procesar el registro. Los filtros se consultan a su vez, hasta que uno de ellos retorna un valor falso. Si ninguno de ellos retorna un valor falso, se emitirá el registro. Si uno retorna un valor falso, el gestor no emitirá el registro.

flush()

Asegúrese de que toda la salida de logging se haya vaciado. Esta versión no hace nada y está destinada a ser implementada por subclases.

close()

Tidy up any resources used by the handler. This version does no output but removes the handler from an internal map of handlers, which is used for handler lookup by name.

Subclasses should ensure that this gets called from overridden **`close()`** methods.

handle(*record*)

Emite condicionalmente el registro especifico, según los filtros que se hayan agregado al gestor. Envuelve la actual emisión del registro con *acquisition/release* del hilo de bloqueo E/S.

handleError(*record*)

This method should be called from handlers when an exception is encountered during an **`emit()`** call. If the module-level attribute **`raiseExceptions`** is `False`, exceptions get silently ignored. This is what is mostly wanted for a logging system - most users will not care about errors in the logging system, they are more interested in application errors. You could, however, replace this with a custom handler if you wish. The specified record is the one which was being processed when the exception occurred. (The default value of **`raiseExceptions`** is `True`, as that is more useful during development).

format(*record*)

Formato para un registro - si se configura un formateador, úselo. De lo contrario, use el formateador predeterminado para el módulo.

emit(*record*)

Haga lo que sea necesario para registrar de forma especifica el registro. Esta versión está destinada a ser implementada por subclases y, por lo tanto, lanza un `NotImplementedError`.

Advertencia

Este método es llamado después de que un bloqueo a nivel de gestor es adquirido, el cual es liberado después de que este método retorna. Cuando sobrescriba este método, tenga en cuenta que debe tener cuidado al llamar a cualquier cosa que invoque a otras partes de la API de registro que puedan realizar bloqueos, ya que esto podría provocar un bloqueo. Específicamente:

- Las API de configuración del registro adquieren el bloqueo a nivel de módulo y, a continuación, los bloqueos a nivel de gestor individual a medida que se configuran dichos gestores.
- Muchas APIs de registro bloquean el bloqueo a nivel de módulo. Si se llama a una API de este

tipo desde este método, podría causar un bloqueo si se realiza una llamada de configuración en otro hilo, porque ese hilo intentará adquirir el bloqueo a nivel de módulo *antes* del bloqueo a nivel de gestor, mientras que este hilo intenta adquirir el bloqueo a nivel de módulo *después* del bloqueo a nivel de gestor (porque en este método, el bloqueo a nivel de gestor ya se ha adquirido).

Para obtener una lista de gestores incluidos como estándar, consulte **`logging.handlers`**.

Objetos formateadores

class logging.Formatter(*fmt=None*, *datefmt=None*, *style='%'*, *validate=True*, ***, *defaults=None*)

Responsable de convertir un **`LogRecord`** en una cadena de salida para ser interpretada por un humano o un sistema externo.

Parámetros:

- **fmt** (*str*) – Una cadena de formato en el *estilo* dado para la salida registrada en su conjunto. Las posibles claves de mapeo se extraen de Atributos LogRecord del objeto **`LogRecord`**. Si no se especifica, se utiliza `'%(message)s'`, que es sólo el mensaje registrado.
- **datefmt** (*str*) – A format string for the date/time portion of the logged output. If not specified, the default described in **`formatTime()`** is used.
- **style** (*str*) – Can be one of `'%'`, `'{'` or `'$'` and determines how the format string will be merged with its data:

using one of Formateo de cadenas al estilo *printf* (`%`), **`str.format()`** (`{}`) or **`string.Template`** (`$`). This only applies to *fmt* (e.g. `'%(message)s'` versus `'{message}'`), not to the actual log messages passed to the logging methods. However, there are other ways to use `{`- and `$`-formatting for log messages.

- **validate** (*bool*) – Si el parámetro *validate* es `True` (lo es por defecto), es incorrecto o no coincidente, *fmt* y *style* lanzarán un **`ValueError`**; Por ejemplo, `logging.Formatter('%(asctime)s - %(message)s', style='{')`.
- **defaults** (*dict[str, Any]*) – El parámetro *defaults* puede ser un diccionario con valores por defecto para usar en campos personalizados. Por ejemplo: `logging.Formatter('%(ip)s %(message)s', defaults={"ip": None})`

Distinto en la versión 3.2: Added the *style* parameter.

Distinto en la versión 3.8: Added the *validate* parameter.

Distinto en la versión 3.10: Added the *defaults* parameter.

format(*record*)

El diccionario de atributos del registro se usa como el operando de una operación para formateo de cadenas. Retorna la cadena resultante. Antes de formatear el diccionario, se lleva a cabo un par de pasos preparatorios. El atributo *message* del registro se calcula usando *msg* % *args*. Si el formato de la cadena contiene `'(asctime)'`, **`formatTime()`** se llama para dar formato al tiempo del evento. Si hay información sobre la excepción, se formatea usando

`formatException()` y se adjunta al mensaje. Tenga en cuenta que la información de excepción formateada se almacena en caché en el atributo *exc_text*. Esto es útil porque la información de excepción se puede serializar con pickle (*pickled*) y propagarse por el cable, pero debe tener cuidado si tiene más de una subclase de **`Formatter`** que personaliza el formato de la información de la excepción. En este caso, tendrá que borrar el valor almacenado en caché (estableciendo el atributo *exc_text* a `None`) después de que un formateador haya terminado su formateo, para que el siguiente formateador que maneje el evento no use el valor almacenado en caché, sino que lo recalcule de nuevo.

Si la información de la pila está disponible, se agrega después de la información de la excepción, usando **`formatStack()`** para transformarla si es necesario.

formatTime(*record*, *datefmt*=*None*)

Este método debe ser llamado desde **`format()`** por un formateador que espera un tiempo formateado . Este método se puede reemplazar en formateadores para proporcionar cualquier requisito específico, pero el comportamiento básico es el siguiente: if *datefmt* (una cadena), se usa con **`time.strftime()`** para formatear el tiempo de creación del registro De lo contrario, se utiliza el formato “%Y-%m-%d %H:%M:%S,uuu”, donde la parte *uuu* es un valor de milisegundos y las otras letras son **`time.strftime()`** . Un ejemplo de tiempo en este formato es `2003-01-23 00:29:50,411`. Se retorna la cadena resultante.

Esta función utiliza una función configurable por el usuario para convertir el tiempo de creación en una tupla. Por defecto, se utiliza **`time.localtime()`**; Para cambiar esto para una instancia de formateador particular, se agrega el atributo `converter` en una función con igual firma como **`time.localtime()`** o `time.gmtime()`. Para cambiarlo en todos los formateadores, por ejemplo, si desea que todos los tiempos de registro se muestren en GMT, agregue el atributo `converter` en la clase `Formatter`.

Distinto en la versión 3.3: Anteriormente, el formato predeterminado estaba codificado como en este ejemplo: `2010-09-06 22:38:15,292` donde la parte anterior a la coma es manejada por una cadena de formato strptime (`'%Y-%m-%d %H:%M:%S'`), y la parte después de la coma es un valor de milisegundos. Debido a que strptime no tiene una posición de formato para milisegundos, el valor de milisegundos se agrega usando otra cadena de formato, `'%s,%03d'`— ambas cadenas de formato se han codificado en este método. Con el cambio, estas cadenas se definen como atributos de nivel de clase que pueden *overridden* a nivel de instancia cuando se desee. Los nombres de los atributos son `default_time_format` (para una cadena de formato strptime) y `default_msec_format` (para agregar el valor de milisegundos).

Distinto en la versión 3.9: El formato `default_msec_format` puede ser `None`.

formatException(*exc_info*)

Formatea la información de una excepción especificada (una excepción como una tupla estándar es retornada por `sys.exc_info()`) como una cadena. Esta implementación predeterminada solo usa `traceback.print_exception()`. La cadena resultantes retornada.

formatStack(*stack_info*)

Formatea la información de una pila especificada (una cadena es retornada por `traceback.print_stack()`, pero con la ultima línea removida) como una cadena. Esta implementación predeterminada solo retorna el valor de entrada.

class logging.BufferingFormatter(*linefmt=None*)

Una clase base de formateador adecuada para subclasificar cuando se desea formatear un número de registros. Puede pasar una instancia `Formatter` que desee utilizar para formatear cada línea (que corresponde a un único registro). Si no se especifica, el formateador por defecto (que sólo muestra el mensaje del evento) se utiliza como formateador de línea.

formatHeader(*records*)

Retorna una cabecera para una lista de *registros*. La implementación base sólo retorna la cadena vacía. Tendrá que anular este método si desea un comportamiento específico, por ejemplo, mostrar el recuento de registros, un título o una línea separadora.

formatFooter(*records*)

Retorna un pie de página para una lista de *registros*. La implementación base sólo retorna la cadena vacía. Tendrá que anular este método si desea un comportamiento específico, por ejemplo, para mostrar el recuento de registros o una línea separadora.

format(*records*)

Retorna el texto formateado de una lista de *registros*. La implementación base sólo retorna la cadena vacía si no hay registros; en caso contrario, retorna la concatenación de la cabecera, cada registro formateado con el formateador de líneas y el pie de página.

Filtro de Objetos

Los `Manejadores` y los `Registradores` pueden usar los `Filtros` para un filtrado más sofisticado que el proporcionado por los niveles. La clase de filtro base solo permite eventos que están por debajo de cierto punto en la jerarquía del logger. Por ejemplo, un filtro inicializado con "A.B" permitirá los eventos registrados por los logger "A.B", "A.B.C", "A.B.C.D", "A.B.D" etc., pero no "A.BB", “B.A.B”, etc. Si se inicializa con una cadena vacía, se pasan todos los eventos.

class logging.Filter(*name=''*)

Retorna una instancia de la clase **`Filter`**. Si se especifica *name*, nombra un logger que, junto con sus hijos, tendrá sus eventos permitidos a través del filtro. Si *name* es una cadena vacía, permite todos los eventos.

filter(*record*)

¿Debe registrarse el registro especificado? Retorna “false” para no, “true” para sí. Los filtros pueden modificar los registros en el lugar o retornar una instancia de registro completamente diferente que sustituirá al registro original en cualquier procesamiento futuro del evento.

Tenga en cuenta que los filtros adjuntos a los gestores se consultan antes de que el gestor emita un evento, mientras que los filtros adjuntos a los loggers se consultan cada vez que se registra un evento (usando **`debug()`**, **`info()`**, etc.), antes de enviar un evento a los gestores. Esto significa que los eventos que han sido generados por loggers descendientes no serán filtrados por la configuración del filtro del logger, a menos que el filtro también se haya aplicado a esos loggers descendientes.

En realidad, no se necesita la subclase `Filtro`: se puede pasar cualquier instancia que tenga un método de `filter` con la misma semántica.

Distinto en la versión 3.2: No es necesario crear clases especializadas de `Filter` ni usar otras clases con un método `filter`: puede usar una función (u otra invocable) como filtro. La lógica de filtrado verificará si el objeto de filtro tiene un atributo `filter`: si lo tiene, se asume que es un `Filter` y se llama a su método **`filter()`**. De lo contrario, se supone que es invocable y se llama con el registro como único parámetro. El valor retornado debe ajustarse al retornado por **`filter()`**.

Distinto en la versión 3.12: Ahora puedes retornar una instancia de

`LogRecord` desde los filtros para reemplazar el registro en lugar de modificarlo. Esto permite a los filtros adjuntos a un `Handler` modificar el registro de log antes de que se emita, sin tener efectos secundarios en otros gestores.

Aunque los filtros se utilizan principalmente para filtrar registros basados en criterios más sofisticados que los niveles, son capaces de ver cada registro que es procesado por el gestor o logger al que están adjuntos: esto puede ser útil si desea hacer cosas como contar cuántos registros fueron procesados por un logger o gestor en particular, o agregando, cambiando o quitando atributos en `LogRecord` que se está procesando. Obviamente, el cambio de LogRecord debe hacerse con cierto cuidado, pero permite la inyección de información contextual en los registros (ver Usar filtros para impartir información contextual).

Objetos LogRecord

Las instancias `LogRecord` son creadas automáticamente por `Logger` cada vez que se registra algo, y se pueden crear manualmente a través de `makeLogRecord()` (por ejemplo, a partir de un evento serializado (*pickled*) recibido en la transmisión).

class logging.LogRecord(*name*, *level*, *pathname*, *lineno*, *msg*, *args*, *exc_info*, *func=None*, *sinfo=None*)

Contiene toda la información pertinente al evento que se registra.

La información principal se pasa en *msg* y *args*, que se combinan usando `msg % args` para crear el atributo **`message`** del registro.

Parámetros:

- **name** (*str*) – El nombre del logger utilizado para registrar el evento representado por este **`LogRecord`**. Tenga en cuenta que el nombre del logger en el **`LogRecord`** siempre tendrá este valor, aunque puede ser emitido por un gestor adjunto a un logger diferente (ancestro).

- **level** (*int*) – El numeric level del evento de registro (como `10` para `DEBUG`, `20` para `INFO`, etc). Tenga en cuenta que esto se convierte en *dos* atributos del LogRecord: **`levelno`** para el valor numérico y **`levelname`** para el nombre del nivel correspondiente.
- **pathname** (*str*) – El nombre de ruta completo del archivo de origen donde se realizó la llamada logging.
- **lineno** (*int*) – El número de línea en el archivo de origen donde se realizó la llamada logging.
- **msg** (*Any*) – El mensaje de descripción del evento, que puede ser una cadena de formato %, con marcadores de posición para datos variables o un objeto arbitrario (para más información vea Usando objetos arbitrarios como mensajes).
- **args** (*tuple* | *dict[str, Any]*) – Datos variables para fusionar en el argumento *msg* para obtener la descripción del evento.
- **exc_info** (*tuple[type[BaseException], BaseException, types.TracebackType]* | *None*) – Una tupla de excepción con la información de excepción actual, tal como se retorna por **`sys.exc_info()`** o `None` si no hay información de excepción disponible.
- **func** (*str* | *None*) – El nombre de la función o método desde el que se invocó la llamada de logging.
- **sinfo** (*str* | *None*) – Una cadena de texto que representa la información de la pila desde la base de la pila en el hilo actual, hasta la llamada de

logging.

getMessage()

Retorna el mensaje para la instancia `LogRecord` después de fusionar cualquier argumento proporcionado por el usuario con el mensaje. Si el argumento del mensaje proporcionado por el usuario para la llamada de logging no es una cadena de caracteres, se invoca `str()` para convertirlo en una cadena. Esto permite el uso de clases definidas por el usuario como mensajes, cuyo método `__str__` puede retornar la cadena de formato real que se utilizará.

Distinto en la versión 3.2: La creación de `LogRecord` se ha hecho más configurable al proporcionar una fábrica que se utiliza para crear el registro. La fábrica se puede configurar usando `getLogRecordFactory()` y `setLogRecordFactory()` (ver esto para la firma de la fábrica).

Esta funcionalidad se puede utilizar para inyectar sus propios valores en `LogRecord` en el momento de la creación. Puede utilizar el siguiente patrón:

```
old_factory = logging.getLogRecordFactory()

def record_factory(*args, **kwargs):
    record = old_factory(*args, **kwargs)
    record.custom_attribute = 0xdecafbad
    return record

logging.setLogRecordFactory(record_factory)
```

Con este patrón, se podrían encadenar varias fábricas y, siempre que no sobrescriban los atributos de las demás o se sobrescriban involuntariamente los atributos estándar enumerados anteriormente, no debería haber sorpresas.

Atributos LogRecord

El LogRecord tiene una serie de atributos, la mayoría de los cuales se derivan de los parámetros del constructor. (Tenga en cuenta que los nombres no siempre se corresponden exactamente entre los parámetros del constructor de LogRecord y los atributos de LogRecord). Estos atributos pueden utilizarse para combinar los datos del registro en la cadena de formato. La siguiente tabla enumera (en orden alfabético) los nombres de los atributos, sus significados y el correspondiente marcador de posición en una cadena de formato %-style.

Si utilizas formato-{} (**`str.format()`**), puedes usar `{attrname}` como marcador de posición en la cadena de caracteres de formato. Si está utilizando formato-$ (**`string.Template`**), use la forma `${attrname}`. En ambos casos, por supuesto, reemplace `attrname` con el nombre de atributo real que desea utilizar.

En el caso del formato con {}, puede especificar *flags* de formato colocándolos después del nombre del atributo, separados con dos puntos. Por ejemplo: un marcador de posición de `{msecs:03.0f}` formateará un valor de milisegundos de `4` como `004`. Consulte la documentación **`str.format()`** para obtener detalles completos sobre las opciones disponibles.

Nombre del atributo	**Formato**	**Descripción**
args	No debería necesitar formatear esto usted mismo.	La tupla de argumentos se fusionó en `msg` para producir un `messsage`, o un dict cuyos valores se utilizan para la fusión (cuando solo hay un argumento y es un diccionario).
asctime	`%(asctime)s`	Fecha y Hora en formato legible por humanos cuando se creó **`LogRecord`**. De forma predeterminada, tiene el formato “2003-07-08 16: 49: 45,896” (los números después de la coma son milisegundos).
created	`%(created)f`	Time when the **`LogRecord`** was created (as returned by **`time.time_ns()`** / 1e9).
exc_info	No debería necesitar formatear esto usted mismo.	Tupla de excepción (al modo `sys.exc_info`) o, si no se ha producido ninguna excepción, `None`.
exc_text	No debería necesitar formatear esto usted mismo.	Exception information formatted as a string. This is set when **`Formatter.format()`** is invoked, or `None` if no exception has occurred.
filename	`%(filename)s`	Parte del nombre de archivo de `pathname`.
funcName	`%(funcName)s`	Nombre de la función que contiene la llamada de logging.

levelname	`%(levelname)s`	Texto de nivel de logging para el mensaje (`'DEBUG'`, `'INFO'`, `'WARNING'`, `'ERROR'`, `'CRITICAL'`).
levelno	`%(levelno)s`	Número de nivel de logging para el mensaje (**`DEBUG`**, **`INFO`**, **`WARNING`**, **`ERROR`**, **`CRITICAL`**).
lineno	`%(lineno)d`	Número de línea original donde se emitió la llamada de logging (si está disponible).
message	`%(message)s`	El mensaje registrado, computado como `msg % args`. Esto se establece cuando se invoca **`Formatter.format()`**.
module	`%(module)s`	Módulo (parte del nombre de `filename`).
msecs	`%(msecs)d`	Porción de milisegundos del tiempo en que se creó **`LogRecord`**.
msg	No debería necesitar formatearlo usted mismo.	La cadena de caracteres de formato pasada en la llamada logging original. Se fusionó con `args` para producir un `message`, o un objeto arbitrario (ver Usando objetos arbitrarios como mensajes).
name	`%(name)s`	Nombre del logger usado para registrar la llamada.
pathname	`%(pathname)s`	Nombre de ruta completo del archivo de origen donde se emitió la llamada de logging (si está disponible).
process	`%(process)d`	ID de proceso (si está disponible).
processName	`%(processName)s`	Nombre del proceso (si está disponible).
relativeCreated	`%(relativeCreated)d`	Tiempo en milisegundos cuando se creó el LogRecord, en relación con el tiempo en que se cargó el módulo logging.
stack_info	No debería necesitar formatearlo usted mismo.	Apila la información del marco (si está disponible) desde la parte inferior de la pila en el hilo actual hasta la llamada de registro que dio como resultado la generación de este registro.
thread	`%(thread)d`	ID de hilo (si está disponible).
threadName	`%(threadName)s`	Nombre del hilo (si está disponible).
taskName	`%(taskName)s`	nombre de **`asyncio.Task`** (si está disponible).

Distinto en la versión 3.1: processName fue agregado.

Distinto en la versión 3.12: taskName fue agregado.

Objetos LoggerAdapter

Las instancias **`LoggerAdapter`** se utilizan para pasar convenientemente información contextual en las llamadas de logging. Para ver un ejemplo de uso, consulte la sección sobre agregar información contextual a su salida de logging.

class logging.LoggerAdapter(*logger*, *extra=None*, *merge_extra=False*)

Returns an instance of **`LoggerAdapter`** initialized with an underlying `Logger` instance, an optional dict-like object (*extra*), and an optional boolean (*merge_extra*) indicating whether or not the *extra* argument of individual log calls should be merged with the **`LoggerAdapter`** extra. The default behavior is to ignore the *extra* argument of individual log calls and only use the one of the **`LoggerAdapter`** instance

process(*msg*, *kwargs*)

> Modifica el mensaje y/o los argumentos de palabra clave pasados a una llamada de logging para insertar información contextual. Esta implementación toma el objeto pasado como *extra* al constructor y lo agrega a *kwargs* usando la clave "extra". El valor de retorno es una tupla (*msg*, *kwargs*) que tiene las versiones (posiblemente modificadas) de los argumentos pasados.

manager

> Delegates to the underlying **`manager`** on *logger*.

_log

> Delegates to the underlying **`_log()`** method on *logger*.

Además de lo anterior, `LoggerAdapter` admite los siguientes métodos de `Logger`: `debug()`, `info()`, `warning()`, `error()`, `exception()`, `critical()`, `log()`, `isEnabledFor()`, `getEffectiveLevel()`, `setLevel()` y `hasHandlers()`. Estos métodos tienen las mismas firmas que sus contrapartes en **`Logger`**, por lo que puede usar los dos tipos de instancias indistintamente.

Distinto en la versión 3.2: Los métodos `isEnabledFor()`, `getEffectiveLevel()`, `setLevel()` y `hasHandlers()` se agregaron a `LoggerAdapter` . Estos métodos se delegan al logger subyacente.

Distinto en la versión 3.6: Se añadió el atributo **`manager`** y el método **`_log()`**, que delegan al logger subyacente y permiten que los adaptadores se aniden.

Distinto en la versión 3.10: The *extra* argument is now optional.

Distinto en la versión 3.13: The *merge_extra* parameter was added.

Seguridad del hilo

The logging module is intended to be thread-safe without any special work needing to be done by its clients. It achieves this through using threading locks; there is one lock to serialize access to the module's shared data, and each handler also creates a lock to serialize access to its underlying I/O.

Si está implementando gestores de señales asíncronos usando el módulo **`signal`**, es posible que no pueda usar logging desde dichos gestores. Esto se debe a que las implementaciones de bloqueo en el módulo **`threading`** no siempre son reentrantes y, por lo tanto, no se pueden invocar desde dichos gestores de señales.

Funciones a nivel de módulo

Además de las clases descritas anteriormente, hay una serie de funciones a nivel de módulo.

logging.getLogger(*name=None*)

Return a logger with the specified name or, if name is `None`, return the root logger of the hierarchy. If specified, the name is typically a dot-separated hierarchical name like *"a"*, *"a.b"* or *"a.b.c.d"*. Choice of these names is entirely up to the developer who is using logging, though it is recommended that `__name__` be used unless you have a specific reason for not doing that, as mentioned in Objetos logger.

Todas las llamadas a esta función con un nombre dado retornan la misma instancia de logger. Esto significa que las instancias del logger nunca necesitan pasar entre diferentes partes de una aplicación.

logging.getLoggerClass()

Retorna ya sea la clase estándar **`Logger`**, o la última clase pasada a **`setLoggerClass()`**. Esta función se puede llamar desde una nueva definición de clase, para garantizar que la instalación de una clase personalizada **`Logger`** no deshaga las *customizaciones* ya aplicadas por otro código. Por ejemplo:

```
class MyLogger(logging.getLoggerClass()):
    # ... override behaviour here
```

logging.getLogRecordFactory()

Retorna un invocable que se usa para crear una **`LogRecord`**.

Added in version 3.2: Esta función se ha proporcionado, junto con **`setLogRecordFactory()`**, para permitir a los desarrolladores un mayor control sobre cómo **`LogRecord`** representa un evento logging construido.

Consulte **`setLogRecordFactory()`** para obtener más información sobre cómo se llama a la fábrica.

logging.debug(*msg*, **args*, ***kwargs*)

This is a convenience function that calls **`Logger.debug()`**, on the root logger. The handling of the arguments is in every way identical to what is described in that method.

The only difference is that if the root logger has no handlers, then **`basicConfig()`** is called, prior to calling `debug` on the root logger.

For very short scripts or quick demonstrations of `logging` facilities, `debug` and the other module-level functions may be convenient. However, most programs will want to carefully and explicitly control the logging configuration, and should therefore prefer creating a module-level logger and calling **`Logger.debug()`** (or other level-specific methods) on it, as described at the beginning of this documentation.

logging.info(*msg*, **args*, ***kwargs*)

Logs a message with level `INFO` on the root logger. The arguments and behavior are otherwise the same as for `debug()`.

logging.warning(*msg*, **args*, ***kwargs*)

Logs a message with level `WARNING` on the root logger. The arguments and behavior are otherwise the same as for `debug()`.

Nota

Hay una función obsoleta `warn` que es funcionalmente idéntica a `warning`. Como `warn` está deprecado, por favor no lo use, use `warning` en su lugar.

logging.error(*msg*, **args*, ***kwargs*)

Logs a message with level `ERROR` on the root logger. The arguments and behavior are otherwise the same as for `debug()`.

logging.critical(*msg*, **args*, ***kwargs*)

Logs a message with level `CRITICAL` on the root logger. The arguments and behavior are otherwise the same as for `debug()`.

logging.exception(*msg*, **args*, ***kwargs*)

Logs a message with level `ERROR` on the root logger. The arguments and behavior are otherwise the same as for `debug()`. Exception info is added to the logging message. This function should only be called from an exception handler.

logging.log(*level*, *msg*, **args*, ***kwargs*)

Logs a message with level *level* on the root logger. The arguments and behavior are otherwise the same as for

`debug()`.

logging.disable(*level=CRITICAL*)

Proporciona un nivel superior de *level* para todos los loggers que tienen prioridad sobre el propio nivel del logger. Cuando surge la necesidad de reducir temporalmente la salida de logging en toda la aplicación, esta función puede resultar útil. Su efecto es deshabilitar todas las llamadas de gravedad *level* e inferior, de modo que si lo llaman con un valor de INFO, todos los eventos INFO y DEBUG se descartarán, mientras que los de gravedad WARNING y superiores se procesarán de acuerdo con el nivel efectivo del logger. Si se llama a `logging.disable(logging.NOTSET)` , elimina efectivamente este nivel primordial, de modo que la salida del registro depende nuevamente de los niveles efectivos de los loggers individuales.

Tenga en cuenta que si ha definido un nivel de logging personalizado superior a `CRITICAL` (esto no es recomendado), no podrá confiar en el valor predeterminado para el parámetro *level*, pero tendrá que proporcionar explícitamente un valor adecuado.

Distinto en la versión 3.7: El parámetro *level* se estableció por defecto en el nivel `CRITICAL`. Consulte el Issue #28524 para obtener más información sobre este cambio.

logging.addLevelName(*level*, *levelName*)

Asocia nivel *level* con el texto *levelName* en un diccionario interno, que se utiliza para asignar niveles numéricos a una representación textual, por ejemplo, cuando `Formatter` formatea un mensaje. Esta función también se puede utilizar para definir sus propios niveles. Las únicas restricciones son que todos los niveles utilizados deben registrarse utilizando esta función, los niveles deben ser números enteros positivos y deben aumentar en orden creciente de severidad.

Nota

Si está pensando en definir sus propios niveles, consulte la sección sobre Niveles personalizados.

logging.getLevelNamesMapping()

Retorna una correspondencia entre los nombres de nivel y sus correspondientes niveles de registro. Por ejemplo, la cadena «CRITICAL» corresponde a `CRITICAL`. La correspondencia retornada se copia de una correspondencia interna en cada llamada a esta función.

Added in version 3.11.

logging.getLevelName(*level*)

Retorna la representación textual o numérica del nivel de registro *level*.

Si *level* es uno de los niveles predefinidos `CRITICAL`, `ERROR`, `WARNING`, `INFO` o `DEBUG` entonces se obtiene la cadena correspondiente. Si has asociado niveles con nombres usando `addLevelName()` entonces se retorna el nombre que has asociado con *level*. Si se pasa un valor numérico correspondiente a uno de los niveles definidos, se retorna la representación de cadena correspondiente.

El parámetro *level* también acepta una representación de cadena del nivel como, por ejemplo, «INFO». En estos casos, esta función retorna el correspondiente valor numérico del nivel.

Si no se pasa un valor numérico o de cadena que coincida, se retorna el valor de nivel de la cadena "Level %s".

Nota

Los niveles internamente son números enteros (ya que deben compararse en la lógica de logging). Esta función se utiliza para convertir entre un nivel entero y el nombre del nivel que se muestra en la salida de logging formateado mediante el especificador de formato `%(levelname)s` (ver Atributos LogRecord).

Distinto en la versión 3.4: En las versiones de Python anteriores a la 3.4, esta función también podría pasar un nivel de texto y retornaría el valor numérico correspondiente del nivel. Este comportamiento indocumentado se consideró un error y se eliminó en Python 3.4, pero se restableció en 3.4.2 debido a que conserva la compatibilidad con versiones anteriores.

logging.getHandlerByName(*name*)

Retorna un gestor con el *name* especificado o `None` si no hay un gestor con ese nombre.

Added in version 3.12.

logging.getHandlerNames()

Retorna un conjunto inmutable del nombre de todos los gestor conocidos.

Added in version 3.12.

logging.makeLogRecord(*attrdict*)

Crea y retorna una nueva instancia **`LogRecord`** cuyos atributos están definidos por *attrdict*. Esta función es útil para tomar un diccionario de atributos serializado (*pickled*) **`LogRecord`**, enviado a través de un socket, y reconstituido como una instancia **`LogRecord`** en el extremo receptor.

logging.basicConfig(***kwargs*)

Realiza una configuración básica para el sistema de logging creando una **`StreamHandler`** con un **`Formatter`** predeterminado y agregándolo al logger raíz. Las funciones **`debug()`**, **`info()`**, **`warning()`**, **`error()`** y **`critical()`** llamarán **`basicConfig()`** automáticamente si no se definen gestores para el logger raíz.

Esta función no hace nada si el logger raíz ya tiene gestores configurados, a menos que el argumento de palabra clave *force* esté establecido como `True`.

Nota

Esta función debe llamarse desde el hilo principal antes de que se inicien otros hilos. En las versiones de Python anteriores a 2.7.1 y 3.2, si se llama a esta función desde varios subprocesos, es posible (en raras circunstancias) que se agregue un gestor al logger raíz más de una vez, lo que genera resultados inesperados como mensajes duplicados en el registro.

Se admiten los siguientes argumentos de palabras clave.

Formato	**Descripción**
filename	Especifica que se cree un `FileHandler`, utilizando el nombre de archivo especificado, en lugar de `StreamHandler`.
filemode	Si se especifica *filename*, abre el archivo en mode. Por defecto es `'a'`.
format	Utiliza la cadena de caracteres de formato especificada para el gestor.Los atributos por defecto son `levelname`, `name` y `message` separado por dos puntos.
datefmt	Utiliza el formato de fecha/hora especificado, aceptado por `time.strftime()`.
style	Si *format* es especificado, utilice este estilo para la cadena de caracteres de formato. Uno de `'%'`, `'{'` o `'$'` para printf-style, `str.format()` o `string.Template` respectivamente. El valor predeterminado es `'%'`.
level	Establece el nivel del logger raíz en el level especificado.
stream	Utiliza la secuencia especificada para inicializar `StreamHandler`. Tenga en cuenta que este argumento es incompatible con *filename* - si ambos están presentes, se lanza un `ValueError`.
handlers	Si se especifica, debe ser una iteración de los gestores ya creados para agregar al logger raíz. A cualquier gestor que aún no tenga un formateador configurado se le asignará el formateador predeterminado creado en esta función. Tenga en cuenta que este argumento es incompatible con *filename*

o *stream*; si ambos están presentes, se lanza un `ValueError`.

force Si este argumento de palabra clave se especifica como verdadero, los gestores existentes adjuntos al logger raíz se eliminan y cierran antes de llevar a cabo la configuración tal como se especifica en los otros argumentos.

encoding Si este argumento de palabra clave se especifica junto con *filename*, su valor se utiliza cuando se crea el **`FileHandler`**, y por lo tanto se utiliza al abrir el archivo de salida.

errors Si este argumento de palabra clave se especifica junto con *filename*, su valor se utiliza cuando se crea el **`FileHandler`**, y por lo tanto cuando se abre el archivo de salida. Si no se especifica, se utiliza el valor "backslashreplace". Tenga en cuenta que si se especifica `None`, se pasará como tal a **`open()`**, lo que significa que se tratará igual que pasar "errors".

Distinto en la versión 3.2: Se agregó el argumento *style*.

Distinto en la versión 3.3: Se agregó el argumento *handlers*. Se agregaron verificaciones adicionales para detectar situaciones en las que se especifican argumentos incompatibles (por ejemplo, *handlers* junto con *stream* o *filename*, o *stream* junto con *filename*).

Distinto en la versión 3.8: Se agregó el argumento *force*.

Distinto en la versión 3.9: Se han añadido los argumentos *encoding* y *errors*.

logging.shutdown()

Informa al sistema de logging para realizar un apagado ordenado descargando y cerrando todos los gestores. Esto se debe llamar al salir de la aplicación y no se debe hacer ningún uso posterior del sistema de logging después de esta llamada.

Cuando se importa el módulo de logging, registra esta función como un gestor de salida (ver **`atexit`**), por lo que normalmente no es necesario hacerlo manualmente.

logging.setLoggerClass(*klass*)

Le dice al sistema de logging que use la clase *klass* al crear una instancia de un logger. La clase debe definir `__init__()` tal que solo se requiera un argumento de nombre, y `__init__()` debe llamar `Logger.__init__()`. Por lo general, esta función se llama antes de que cualquier logger sea instanciado por las aplicaciones que necesitan utilizar un comportamiento de logger personalizado. Después de esta llamada, como en cualquier otro momento, no cree instancias de loggers directamente usando la subclase: continúe usando la API `logging.getLogger()` para obtener sus loggers.

logging.setLogRecordFactory(*factory*)

Establece un invocable que se utiliza para crear `LogRecord`.

Parámetros:

factory – La fábrica invocable que se utilizará para crear una instancia de un registro.

Added in version 3.2: Esta función se ha proporcionado, junto con `getLogRecordFactory()`, para permitir a los desarrolladores un mayor control sobre cómo se construye `LogRecord` que representa un evento de logging.

La fábrica tiene la siguiente firma:

`factory(name, level, fn, lno, msg, args, exc_info, func=None, sinfo=None, **kwargs)`

name:	El nombre del logger.
level:	El nivel de logging (numérico).
fn:	

El nombre de ruta completo del archivo donde se realizó la llamada de logging.

lno:

El número de línea en el archivo donde se realizó la llamada de logging.

msg:

El mensaje de logging.

args:

Los argumentos para el mensaje de logging.

exc_info:

Una tupla de excepción o `None`.

func:

El nombre de la función o método que invocó la llamada de logging.

sinfo:

Un seguimiento de pila como el que proporciona `traceback.print_stack()`, que muestra la jerarquía de llamadas.

kwargs:

Argumentos de palabras clave adicionales.

Atributos a nivel de módulo

logging.lastResort

Un «gestor de último recurso» está disponible a través de este atributo. Esta es una `StreamHandler` que escribe en `sys.stderr` con un nivel `WARNING`, y se usa para gestionar eventos de logging en ausencia de cualquier configuración de logging. El resultado final es simplemente imprimir el mensaje en `sys.stderr`. Esto reemplaza el mensaje de error anterior que decía que «no se pudieron encontrar gestores para el logger XYZ». Si necesita el comportamiento anterior por alguna razón, `lastResort` se puede configurar en `None`.

Added in version 3.2.

logging.raiseExceptions

Used to see if exceptions during handling should be propagated.

Default: `True`.

If **`raiseExceptions`** is `False`, exceptions get silently ignored. This is what is mostly wanted for a logging system - most users will not care about errors in the logging system, they are more interested in application errors.

Integración con el módulo de advertencias

The **`captureWarnings()`** function can be used to integrate **`logging`** with the **`warnings`** module.

logging.captureWarnings(*capture*)

Esta función se utiliza para activar y desactivar la captura de advertencias (*warnings*).

Si *capture* es `True`, las advertencias emitidas por el módulo **`warnings`** serán redirigidas al sistema de logging. Específicamente, una advertencia se formateará usando **`warnings.formatwarning()`** y la cadena de caracteres resultante se registrará en un logger llamado `'py.warnings'` con severidad **`WARNING`**.

Si *capture* es `False`, la redirección de advertencias al sistema de logging se detendrá y las advertencias serán redirigidas a sus destinos originales (es decir, aquellos en vigor antes de que se llamara a `captureWarnings(True)`).

Ver también

Módulo `logging.config`
API de configuración para el módulo logging.

Módulo `logging.handlers`
Gestores útiles incluidos con el módulo logging.

PEP 282 [https://peps.python.org/pep-0282/] - A Logging System
La propuesta que describió esta característica para su inclusión en la biblioteca estándar de Python.

Paquete logging original de Python [https://old.red-dove.com/python_logging.html]
This is the original source for the `logging` package. The version of the package available from this site is suitable for use with Python 1.5.2, 2.1.x and 2.2.x, which do not include the `logging` package in the standard library.

`logging.config` — Logging configuration

Código fuente: Lib/logging/config.py [https://github.com/python/cpython/tree/3.14/Lib/logging/config.py]

Important

Esta página solo contiene información de referencia. Para tutoriales, por favor consulte

- Tutorial Básico
- Tutorial Avanzado
- Guía de registro *Logging*

Esta sección describe la API para configurar el módulo de registro.

Funciones de configuración

The following functions configure the logging module. They are located in the `logging.config` module. Their use is optional — you can configure the logging module using these functions or by making calls to the main API (defined in `logging` itself) and defining handlers which are declared either in `logging` or `logging.handlers`.

logging.config.dictConfig(*config*)

Toma la configuración de registro de un diccionario. Los contenidos de este diccionario se describen en Esquema del diccionario de configuración a continuación.

Si se encuentra un error durante la configuración, esta función lanzará un `ValueError`, `TypeError`, `AttributeError` o `ImportError` con un mensaje descriptivo adecuado. La siguiente es una lista (posiblemente incompleta) de condiciones que lanzarán un error:

- Un `level` que no es una cadena o que es una cadena que no corresponde a un nivel de registro real.
- Un valor de `propagate` que no es booleano.
- Una identificación que no tiene un destino correspondiente.
- Una identificación de gestor inexistente encontrada durante una llamada incremental.
- Un nombre de registrador no válido.
- Incapacidad para resolver un objeto interno o externo.

Parsing is performed by the **`DictConfigurator`** class, whose constructor is passed the dictionary used for configuration, and has a **`configure()`** method. The **`logging.config`** module has a callable attribute **`dictConfigClass`** which is initially set to **`DictConfigurator`**. You can replace the value of **`dictConfigClass`** with a suitable implementation of your own.

`dictConfig()` llama **`dictConfigClass`** pasando el diccionario especificado, y luego llama al método **`configure()`** en el objeto retornado para que la configuración surta efecto:

```
def dictConfig(config):
    dictConfigClass(config).configure()
```

For example, a subclass of **`DictConfigurator`** could call `DictConfigurator.__init__()` in its own **`__init__()`**, then set up custom prefixes which would be usable in the subsequent **`configure()`** call. **`dictConfigClass`** would be bound to this new subclass, and then **`dictConfig()`** could be called exactly as in the default, uncustomized state.

Added in version 3.2.

logging.config.fileConfig(*fname*, *defaults=None*, *disable_existing_loggers=True, encoding=None*)

Lee la configuración de registro desde un archivo de formato de `configparser`. El formato del archivo debe ser como se describe en Formato de archivo de configuración. Esta función se puede invocar varias veces desde una aplicación, lo que permite al usuario final seleccionar entre varias configuraciones predeterminadas (si el desarrollador proporciona un mecanismo para presentar las opciones y cargar la configuración elegida).

It will raise `FileNotFoundError` if the file doesn't exist and `RuntimeError` if the file is invalid or empty.

Parámetros:

- **fname** – A filename, or a file-like object, or an instance derived from `RawConfigParser`. If a `RawConfigParser`-derived instance is passed, it is used as is. Otherwise, a `ConfigParser` is instantiated, and the configuration read by it from the object passed in `fname`. If that has a `readline()` method, it is assumed to be a file-like object and read using `read_file()`; otherwise, it is assumed to be a filename and passed to `read()`.
- **defaults** – Defaults to be passed to the `ConfigParser` can be specified in this argument.
- **disable_existing_loggers** – If specified as `False`, loggers which exist when this call is made are left enabled. The default is `True` because this enables old behaviour in a backward-compatible way. This behaviour is to disable any existing non-root loggers unless they or their ancestors are explicitly named in the logging configuration.

- **encoding** – La codificación que se usa para abrir archivos cuando *fname* es nombre del archivo.

Distinto en la versión 3.4: Una instancia de una subclase de **`RawConfigParser`** ahora se acepta como un valor para `fname`. Esto facilita:

- Uso de un archivo de configuración donde la configuración de registro es solo parte de la configuración general de la aplicación.
- Uso de una configuración leída de un archivo, y luego modificada por la aplicación que lo usa (por ejemplo, basada en parámetros de línea de comandos u otros aspectos del entorno de ejecución) antes de pasar a `fileConfig`.

Distinto en la versión 3.10: Added the *encoding* parameter.

Distinto en la versión 3.12: An exception will be thrown if the provided file doesn't exist or is invalid or empty.

logging.config.listen(*port=DEFAULT_LOGGING_CONFIG_PORT*, *verify=None*)

Inicia un servidor de socket en el puerto especificado y escucha nuevas configuraciones. Si no se especifica ningún puerto, se usa el valor predeterminado del módulo **`DEFAULT_LOGGING_CONFIG_PORT`**. Las configuraciones de registro se enviarán como un archivo adecuado para su procesamiento por **`dictConfig()`** o **`fileConfig()`**. Retorna una instancia de **`Thread`** en la que puede llamar **`start()`** para iniciar el servidor, y que puede **`join()`** cuando corresponda . Para detener el servidor, llame a **`stopListening()`**.

El argumento `verify`, si se especifica, debe ser invocable, lo que debería verificar si los bytes recibidos en el socket son válidos y deben procesarse. Esto podría hacerse encriptando y / o firmando lo que se envía a través del socket, de modo que el `verify` invocable pueda realizar la verificación o descifrado de la firma. El llamado invocable `verify` se llama con un solo argumento (los bytes recibidos a través del socket) y debe retornar los bytes a procesar, o `None` para indicar que los bytes deben descartarse. Los bytes retornados podrían ser los mismos que los pasados en bytes (por ejemplo, cuando solo se realiza la verificación), o podrían ser completamente diferentes (tal vez si se realizó el descifrado).

Para enviar una configuración al socket, lea el archivo de configuración y envíelo al socket como una secuencia de bytes precedida por una cadena de longitud de cuatro bytes empaquetada en binario usando `struct.pack('>L', n)`.

Nota

Debido a que partes de la configuración se pasan a través de `eval()`, el uso de esta función puede abrir a sus usuarios a un riesgo de seguridad. Si bien la función solo se une a un socket en `localhost` y, por lo tanto, no acepta conexiones de máquinas remotas, hay escenarios en los que se puede ejecutar código no confiable bajo la cuenta del proceso que llama `listen()`. Específicamente, si el proceso que llama **`listen()`** se ejecuta en una máquina multiusuario donde los usuarios no pueden confiar el uno en el otro, entonces un usuario malintencionado podría hacer arreglos para ejecutar código esencialmente arbitrario en el proceso del usuario víctima, simplemente conectándose al socket **`listen()`** de la víctima y enviando una configuración que ejecuta cualquier código que el atacante quiera ejecutar en el proceso de la víctima. Esto es especialmente fácil de hacer si se usa el puerto predeterminado, pero no es difícil incluso si se usa un puerto diferente. Para evitar el riesgo de que esto suceda, use el argumento `verify` para **`listen()`** para escuchar y evitar que se apliquen configuraciones no reconocidas.

Distinto en la versión 3.4: Se agregó el argumento `verify`.

Nota

Si desea enviar configuraciones al oyente que no deshabiliten los registradores existentes, deberá usar un formato JSON para la configuración, que utilizará **`dictConfig()`** para la configuración. Este método le permite especificar `disable_existing_loggers` como `False` en la configuración que envía.

logging.config.stopListening()

Detiene el servidor de escucha que se creó con una llamada a **`listen()`**. Esto normalmente se llama antes de llamar **`join()`** en el valor de retorno de **`listen()`**.

Consideraciones de seguridad

The logging configuration functionality tries to offer convenience, and in part this is done by offering the ability to convert text in configuration files into Python objects used in logging configuration - for example, as described in Objetos definidos por el usuario. However, these same mechanisms (importing callables from user-defined modules and calling them with parameters from the configuration) could be used to invoke any code you like, and for this reason you should treat configuration files from untrusted sources with *extreme caution* and satisfy yourself that nothing bad can happen if you load them, before actually loading them.

Esquema del diccionario de configuración

La descripción de una configuración de registro requiere una lista de los diversos objetos para crear y las conexiones entre ellos; por ejemplo, puede crear un gestor llamado "consola" y luego decir que el registrador llamado "inicio" enviará sus mensajes al gestor "consola". Estos objetos no se limitan a los proporcionados por el módulo **`logging`** porque podría escribir su propia clase de formateador o gestor. Los parámetros de estas clases también pueden necesitar incluir objetos externos como `sys.stderr`. La sintaxis para describir estos objetos y conexiones se define en Conexiones de objeto a continuación.

Detalles del esquema del diccionario

El diccionario pasado a **`dictConfig()`** debe contener las siguientes claves:

- *version* - se establece en un valor entero que representa la versión del esquema. El único valor válido en este momento es 1, pero tener esta clave permite que el esquema evolucione sin perder la compatibilidad con versiones anteriores.

Todas las demás claves son opcionales, pero si están presentes se interpretarán como se describe a continuación. En todos los casos a continuación, donde se menciona un "dict de configuración", se verificará la clave especial `'()'` para ver si se requiere una instanciación personalizada. Si es así, el mecanismo descrito en Objetos definidos por el usuario a continuación se usa para crear una instancia; de lo contrario, el contexto se usa para determinar qué instanciar.

- *formatters* - el valor correspondiente será un diccionario en el que cada clave es una identificación de formateador y cada valor es un diccionario que describe cómo configurar la instancia correspondiente **`Formatter`**.

  Se busca el diccionario de configuración por las siguientes claves opcionales que corresponden a los argumentos pasados para crear un objeto **`Formatter`**:

  - ○ `format`
  - ○ `datefmt`
  - ○ `style`
  - ○ `validate` (desde la versión $>=$3.8)
  - ○ `defaults` (since version $>=$3.12)

  Una clave opcional `class` indica el nombre de la de clase del formateador (como un módulo punteado y nombre de clase). Por su parte los argumentos de instanciación son **`Formatter`**, de este modo esta clave es más útil para instanciar una subclase personalizada de **`Formatter`**. Por ejemplo, la clase alternativa presentaría excepciones rastreadas en un formato amplio o resumido. Si tu formateador necesita claves de configuración diferentes o extra debes usar Objetos definidos por el usuario.

- *filters* - el valor correspondiente será un diccionario en el que cada clave es una identificación de filtro y cada valor es un diccionario que describe cómo configurar la instancia de filtro correspondiente.

  El diccionario de configuración busca la clave `name` (por defecto en la cadena vacía) y esto se utiliza para construir una instancia de **`logging.Filter`**.

- *handlers* - el valor correspondiente será un diccionario en el que cada clave es una identificación de gestor y cada valor es un diccionario que describe cómo configurar la instancia del gestor correspondiente.

  El diccionario de configuración busca las siguientes claves:

  - `clase` (obligatorio). Este es el nombre completo de la clase de gestor.
  - `level` (opcional). El nivel del gestor.
  - `formatter` (opcional). La identificación del formateador para este gestor.
  - `filters` (opcional). Una lista de identificadores de los filtros para este gestor.

    Distinto en la versión 3.11: `filters` can take filter instances in addition to ids.

  Todas las claves *other* se pasan como argumentos de palabras clave al constructor del gestor. Por ejemplo, dado el fragmento:

```
handlers:
  console:
    class : logging.StreamHandler
    formatter: brief
    level   : INFO
    filters: [allow_foo]
    stream  : ext://sys.stdout
  file:
    class : logging.handlers.RotatingFileHandler
    formatter: precise
    filename: logconfig.log
    maxBytes: 1024
    backupCount: 3
```

el gestor con id `console` se instancia como **`logging.StreamHandler`**, usando `sys.stdout` como la secuencia subyacente. El gestor con la identificación `file` se instancia como **`logging.handlers.RotatingFileHandler`** con los argumentos de la palabra clave `filename='logconfig.log', maxBytes=1024, backupCount=3`.

- *loggers* - el valor correspondiente será un diccionario en el que cada clave es un nombre de *logger* y cada valor es un diccionario que describe cómo configurar la instancia de *Logger* correspondiente.

  El diccionario de configuración busca las siguientes claves:

  - `level` (opcional). El nivel del registrador.
  - `propagate` (opcional). La configuración de propagación del registrador.
  - `filters` (opcional). Una lista de identificadores de los filtros para este registrador.

    Distinto en la versión 3.11: `filters` can take filter instances in addition to ids.

    - `handlers` (opcional). Una lista de identificadores de los gestores para este registrador.

  Los registradores especificados se configurarán de acuerdo con el nivel, la propagación, los filtros y los gestores especificados.

- *root* - Esta será la configuración para el registrador raíz. El procesamiento de la configuración será como para cualquier registrador, excepto que la configuración de `propagate` no será aplicable.

- *incremental* - si la configuración debe interpretarse como incremental a la configuración existente. Este valor predeterminado es `False`, lo que significa que la configuración especificada reemplaza la configuración existente con la misma semántica que la utilizada por la API existente **`fileConfig()`**.

  Si el valor especificado es `True`, la configuración se procesa como se describe en la sección sobre Configuración incremental.

- *disable_existing_loggers* - si se debe deshabilitar cualquier registrador que no sea raíz existente. Esta configuración refleja el parámetro del mismo nombre en **`fileConfig()`**. Si está ausente, este parámetro tiene el valor predeterminado `True`. Este valor se ignora si *incremental* es `True`.

Configuración incremental

Es difícil proporcionar flexibilidad completa para la configuración incremental. Por ejemplo, debido a que los objetos como los filtros y formateadores son anónimos, una vez que se configura una configuración, no es posible hacer referencia a dichos objetos anónimos al aumentar una configuración.

Además, no hay un caso convincente para alterar arbitrariamente el gráfico de objetos de registradores, gestores, filtros, formateadores en tiempo de ejecución, una vez que se configura una configuración; la verbosidad de los registradores y gestores se puede controlar simplemente estableciendo niveles (y, en el caso de los registradores, flags de propagación). Cambiar el gráfico de objetos de manera arbitraria y segura es problemático en un entorno de subprocesos múltiples; Si bien no es imposible, los beneficios no valen la complejidad que agrega a la implementación.

Por lo tanto, cuando la tecla `incremental` de un diccionario de configuración está presente y es `True`, el sistema ignorará por completo cualquier entrada de `formatters` y `filters`, y procesará solo el `level` configuraciones en las entradas de `handlers`, y las configuraciones de `level` y `propagate` en las entradas de `loggers` y `root`.

El uso de un valor en la configuración diccionario permite que las configuraciones se envíen a través del cable como dictados en vinagre a un escucha de socket. Por lo tanto, la verbosidad de registro de una aplicación de larga ejecución puede modificarse con el tiempo sin necesidad de detener y reiniciar la aplicación.

Conexiones de objeto

El esquema describe un conjunto de objetos de registro (registradores, gestores, formateadores, filtros) que están conectados entre sí en un gráfico de objetos. Por lo tanto, el esquema necesita representar conexiones entre los objetos. Por ejemplo, supongamos que, una vez configurado, un registrador particular le ha adjuntado un gestor particular. A los fines de esta discusión, podemos decir que el registrador representa la fuente y el gestor el destino de una conexión entre los dos. Por supuesto, en los objetos configurados esto está representado por el registrador que tiene una referencia al gestor. En la configuración dict, esto se hace dando a cada objeto de destino una identificación que lo identifica inequívocamente, y luego utilizando la identificación en la configuración del objeto de origen para indicar que existe una conexión entre el origen y el objeto de destino con esa identificación.

Entonces, por ejemplo, considere el siguiente fragmento de YAML:

```
formatters:
  brief:
    # configuration for formatter with id 'brief' goes h
  precise:
    # configuration for formatter with id 'precise' goes
handlers:
  h1: #This is an id
   # configuration of handler with id 'h1' goes here
   formatter: brief
  h2: #This is another id
   # configuration of handler with id 'h2' goes here
   formatter: precise
loggers:
  foo.bar.baz:
    # other configuration for logger 'foo.bar.baz'
    handlers: [h1, h2]
```

(Nota: YAML se usa aquí porque es un poco más legible que el formulario fuente Python equivalente para el diccionario.)

Los identificadores para los registradores son los nombres de los registradores que se utilizarían mediante programación para obtener una referencia a esos registradores, por ejemplo `foo.bar.baz`. Los identificadores para Formateadores y Filtros pueden ser cualquier valor de cadena (como `breve`, `preciso` arriba) y son transitorios, ya que solo son significativos para procesar el diccionario de configuración y se utilizan para determinar conexiones entre objetos , y no persisten en ninguna parte cuando se completa la llamada de configuración.

El fragmento anterior indica que el registrador llamado `foo.bar.baz` debe tener dos gestores adjuntos, que se describen mediante los identificadores de gestor `h1` y `h2`. El formateador para `h1` es el descrito por identificador `brief`, y el formateador para `h2` es el descrito por identificador `precise`.

Objetos definidos por el usuario

El esquema admite objetos definidos por el usuario para gestores, filtros y formateadores. (Los registradores no necesitan tener diferentes tipos para diferentes instancias, por lo que no hay soporte en este esquema de configuración para las clases de registrador definidas por el usuario).

Los objetos a configurar son descritos por diccionarios que detallan su configuración. En algunos lugares, el sistema de registro podrá inferir del contexto cómo se va a instanciar un objeto, pero cuando se va a instanciar un objeto definido por el usuario, el sistema no sabrá cómo hacerlo. Con el fin de proporcionar una flexibilidad completa para la creación de instancias de objetos definidos por el usuario, el usuario debe proporcionar una “fábrica”, una llamada que se llama con un diccionario de configuración y que retorna el objeto instanciado. Esto se indica mediante una ruta de importación absoluta a la fábrica disponible bajo la tecla especial `'()'`. Aquí hay un ejemplo concreto:

```
formatters:
  brief:
    format: '%(message)s'
  default:
    format: '%(asctime)s %(levelname)-8s %(name)-15s %(m
    datefmt: '%Y-%m-%d %H:%M:%S'
  custom:
      (): my.package.customFormatterFactory
      bar: baz
      spam: 99.9
      answer: 42
```

El fragmento de YAML anterior define tres formateadores. El primero, con identificador `breve`, es una instancia estándar **`logging.Formatter`** con la cadena de formato especificada. El segundo, con identificador `predeterminado`, tiene un formato más largo y también define el formato de hora explícitamente, y dará como resultado **`logging.Formatter`** inicializado con esas dos cadenas de formato. En forma de fuente Python, los formateadores `breve` y `predeterminado` tienen sub-diccionarios de configuración:

```
{
  'format' : '%(message)s'
}
```

y:

```
{
  'format' : '%(asctime)s %(levelname)-8s %(name)-15s %(
  'datefmt' : '%Y-%m-%d %H:%M:%S'
}
```

respectivamente, y como estos diccionarios no contienen la clave especial `'()'`, la instanciación se infiere del contexto: como resultado, se crean las instancias estándar **`logging.Formatter`**. El sub-diccionario de configuración para el tercer formateador, con identificador `custom`, es:

```
{
  '()' : 'my.package.customFormatterFactory',
  'bar' : 'baz',
  'spam' : 99.9,
  'answer' : 42
}
```

y esto contiene la clave especial `'()'`, lo que significa que se desea la creación de instancias definida por el usuario. En este caso, se utilizará la llamada especificada de fábrica especificada. Si es una llamada real, se usará directamente; de lo contrario, si especifica una cadena (como en el ejemplo), la llamada real se ubicará utilizando mecanismos de importación normales. Se llamará al invocable con los elementos **restantes** en el sub-diccionario de configuración como argumentos de palabras clave. En el ejemplo anterior, se supondrá que el formateador con identificador `custom` será retornado por la llamada:

```
my.package.customFormatterFactory(bar='baz', spam=99.9,
```

Advertencia

The values for keys such as `bar`, `spam` and `answer` in the above example should not be configuration dictionaries or references such as `cfg://foo` or `ext://bar`, because they will not be processed by the configuration machinery, but passed to the callable as-is.

La clave `'()'` se ha utilizado como clave especial porque no es un nombre de parámetro de palabra clave válido, por lo que no entrará en conflicto con los nombres de los argumentos de palabras clave utilizados en la llamada. El `'()'` también sirve como mnemónico de que el valor correspondiente es invocable.

Distinto en la versión 3.11: Los miembros `filter` de `handlers` y `loggers` pueden tomar instancias de filtro en adición a identificadores.

You can also specify a special key `'.'` whose value is a mapping of attribute names to values. If found, the specified attributes will be set on the user-defined object before it is returned. Thus, with the following configuration:

```
{
  '()' : 'my.package.customFormatterFactory',
  'bar' : 'baz',
  'spam' : 99.9,
  'answer' : 42,
  '.' {
    'foo': 'bar',
    'baz': 'bozz'
  }
}
```

el formateador retornado tendrá el atributo `foo` establecido en `'bar'` y el atributo `baz` establecido en `'bozz'`.

Advertencia

The values for attributes such as `foo` and `baz` in the above example should not be configuration dictionaries or references such as `cfg://foo` or `ext://bar`, because they will not be processed by the configuration machinery, but set as attribute values as-is.

Handler configuration order

Handlers are configured in alphabetical order of their keys, and a configured handler replaces the configuration dictionary in (a working copy of) the `handlers` dictionary in the schema. If you use a construct such as `cfg://handlers.foo`, then initially `handlers['foo']` points to the configuration dictionary for the handler named `foo`, and later (once that handler has been configured) it points to the configured handler instance. Thus, `cfg://handlers.foo` could resolve to either a dictionary or a handler instance. In general, it is wise to name handlers in a way such that dependent handlers are configured *after* any handlers they depend on; that allows something like `cfg://handlers.foo` to be used in configuring a handler that depends on handler `foo`. If that dependent handler were named `bar`, problems would result, because the configuration of `bar` would be attempted before that of `foo`, and `foo` would not yet have been configured. However, if the dependent handler were named `foobar`, it would be configured after `foo`, with the result that `cfg://handlers.foo` would resolve to configured handler `foo`, and not its configuration dictionary.

Acceso a objetos externos

Hay momentos en que una configuración debe referirse a objetos externos a la configuración, por ejemplo, `sys.stderr`. Si el diccionario de configuración se construye utilizando el código Python, esto es sencillo, pero surge un problema cuando la configuración se proporciona a través de un archivo de texto (por ejemplo, JSON, YAML). En un archivo de texto, no hay una forma estándar de distinguir `sys.stderr` de la cadena literal `'sys.stderr'`. Para facilitar esta distinción, el sistema de configuración busca ciertos prefijos especiales en valores de cadena y los trata especialmente. Por ejemplo, si la cadena literal `'ext://sys.stderr'` se proporciona como un valor en la configuración, entonces la `ext://` se eliminará y se procesará el resto del valor utilizando mecanismos normales de importación.

El manejo de dichos prefijos se realiza de manera análoga al manejo del protocolo: existe un mecanismo genérico para buscar prefijos que coincidan con la expresión regular `^(?P<prefix>[a-z]+)://(?P<suffix>.*)$` por el cual, si se reconoce el `prefix`, el `suffix` se procesa de manera dependiente del prefijo y el resultado del procesamiento reemplaza el valor de la cadena. Si no se reconoce el prefijo, el valor de la cadena se dejará tal cual.

Acceso a objetos internos

Además de los objetos externos, a veces también es necesario hacer referencia a los objetos en la configuración. El sistema de configuración hará esto implícitamente para las cosas que conoce. Por ejemplo, el valor de cadena `'DEBUG'` para un `level` en un registrador o gestor se convertirá automáticamente al valor `logging.DEBUG`, y las entradas `handlers`, `filters` y `formatter` tomarán una identificación de objeto y se resuelven en el objeto de destino apropiado.

Sin embargo, se necesita un mecanismo más genérico para los objetos definidos por el usuario que no conoce el módulo **`logging`**. Por ejemplo, considere **`logging.handlers.MemoryHandler`**, que toma un argumento `target` que es otro gestor para delegar. Dado que el sistema ya conoce esta clase, entonces en la configuración, el `target` dado solo necesita ser la identificación del objeto del gestor de destino relevante, y el sistema resolverá el gestor desde la identificación. Sin embargo, si un usuario define un `my.package.MyHandler` que tiene un gestor `alternate`, el sistema de configuración no sabría que el `alternate` se refería a un gestor. Para atender esto, un sistema de resolución genérico permite al usuario especificar:

```
handlers:
  file:
    # configuration of file handler goes here

  custom:
    (): my.package.MyHandler
    alternate: cfg://handlers.file
```

La cadena literal `'cfg://handlers.file'` se resolverá de manera análoga a las cadenas con el prefijo `ext://`, pero buscando en la configuración misma en lugar del espacio de nombres de importación. El mecanismo permite el acceso por punto o por índice, de manera similar a la proporcionada por `str.format`. Por lo tanto, dado el siguiente fragmento:

```
handlers:
  email:
    class: logging.handlers.SMTPHandler
    mailhost: localhost
    fromaddr: my_app@domain.tld
    toaddrs:
      - support_team@domain.tld
      - dev_team@domain.tld
    subject: Houston, we have a problem.
```

in the configuration, the string `'cfg://handlers'` would resolve to the dict with key `handlers`, the string `'cfg://handlers.email` would resolve to the dict with key `email` in the `handlers` dict, and so on. The string `'cfg://handlers.email.toaddrs[1]` would resolve to `'dev_team@domain.tld'` and the string `'cfg://handlers.email.toaddrs[0]'` would resolve to the value `'support_team@domain.tld'`. The `subject` value could be accessed using either `'cfg://handlers.email.subject'` or, equivalently, `'cfg://handlers.email[subject]'`. The latter form only needs to be used if the key contains spaces or non-alphanumeric characters. Please note that the characters `[` and `]` are not allowed in the keys. If an index value consists only of decimal digits, access will be attempted using the corresponding integer value, falling back to the string value if needed.

Dada una cadena `cfg://handlers.myhandler.mykey.123`, esto se resolverá en `config_dict['handlers']['myhandler']['mykey']['123']`. Si la cadena se especifica como `cfg: //handlers.myhandler.mykey[123]`, el sistema intentará recuperar el valor de `config_dict['handlers']['myhandler']['mykey'][123]`, y vuelva a `config_dict['handlers']['myhandler']['mykey']['123']` si eso falla.

Resolución de importación e importadores personalizados

La resolución de importación, por defecto, utiliza la función incorporada `__import__()` para importar. Es posible que desee reemplazar esto con su propio mecanismo de importación: si es así, puede reemplazar el atributo **`importer`** de **`DictConfigurator`** o su superclase, la clase **`BaseConfigurator`**. Sin embargo, debe tener cuidado debido a la forma en que se accede a las funciones desde las clases a través de descriptores. Si está utilizando un Python invocable para realizar sus importaciones, y lo desea definir a nivel de clase en lugar de a nivel de instancia, debe envolverlo con `staticmethod()`. Por ejemplo:

```
from importlib import import_module
from logging.config import BaseConfigurator

BaseConfigurator.importer = staticmethod(import_module)
```

No necesita envolver con `staticmethod()` si está configurando la importación invocable en un configurador *instance*.

Configuring QueueHandler and QueueListener

If you want to configure a `QueueHandler`, noting that this is normally used in conjunction with a `QueueListener`, you can configure both together. After the configuration, the `QueueListener` instance will be available as the `listener` attribute of the created handler, and that in turn will be available to you using `getHandlerByName()` and passing the name you have used for the `QueueHandler` in your configuration. The dictionary schema for configuring the pair is shown in the example YAML snippet below.

```
handlers:
  qhand:
    class: logging.handlers.QueueHandler
    queue: my.module.queue_factory
    listener: my.package.CustomListener
    handlers:
      - hand_name_1
      - hand_name_2
      ...
```

The `queue` and `listener` keys are optional.

If the `queue` key is present, the corresponding value can be one of the following:

- An object implementing the **`Queue.put_nowait`** and **`Queue.get`** public API. For instance, this may be an actual instance of **`queue.Queue`** or a subclass thereof, or a proxy obtained by **`multiprocessing.managers.SyncManager.Queue()`**.

  This is of course only possible if you are constructing or modifying the configuration dictionary in code.

- A string that resolves to a callable which, when called with no arguments, returns the queue instance to use. That callable could be a **`queue.Queue`** subclass or a function which returns a suitable queue instance, such as `my.module.queue_factory()`.

- A dict with a `'()'` key which is constructed in the usual way as discussed in Objetos definidos por el usuario. The result of this construction should be a **`queue.Queue`** instance.

If the `queue` key is absent, a standard unbounded **`queue.Queue`** instance is created and used.

If the `listener` key is present, the corresponding value can be one of the following:

- A subclass of **`logging.handlers.QueueListener`**. This is of course only possible if you are constructing or modifying the configuration dictionary in code.
- A string which resolves to a class which is a subclass of `QueueListener`, such as `'my.package.CustomListener'`.
- A dict with a `'()'` key which is constructed in the usual way as discussed in Objetos definidos por el usuario. The result of this construction should be a callable with the same signature as the `QueueListener` initializer.

If the `listener` key is absent, **`logging.handlers.QueueListener`** is used.

The values under the `handlers` key are the names of other handlers in the configuration (not shown in the above snippet) which will be passed to the queue listener.

Any custom queue handler and listener classes will need to be defined with the same initialization signatures as **`QueueHandler`** and **`QueueListener`**.

Added in version 3.12.

Formato de archivo de configuración

El formato del archivo de configuración que entiende **`fileConfig()`** se basa en la funcionalidad **`configparser`**. El archivo debe contener secciones llamadas `[loggers]`, `[handlers]` y `[formatters]` que identifican por nombre las entidades de cada tipo que se definen en el archivo. Para cada una de esas entidades, hay una sección separada que identifica cómo se configura esa entidad. Por lo tanto, para un registrador llamado `log01` en la sección `[loggers]`, los detalles de configuración relevantes se encuentran en una sección `[logger_log01]`. Del mismo modo, un gestor llamado `hand01` en la sección `[handlers]` tendrá su configuración en una sección llamada `[handler_hand01]`, mientras que un formateador llamado `form01` en el `[formatters]` sección tendrá su configuración especificada en una sección llamada `[formatter_form01]`. La configuración del registrador raíz debe especificarse en una sección llamada `[logger_root]`.

Nota

La API **`fileConfig()`** es más antigua que la API **`dictConfig()`** y no proporciona funcionalidad para cubrir ciertos aspectos del registro. Por ejemplo, no puede configurar objetos **`Filter`**, que permiten el filtrado de mensajes más allá de niveles enteros simples, usando **`fileConfig()`**. Si necesita tener instancias de **`Filter`** en su configuración de registro, deberá usar **`dictConfig()`**. Tenga en cuenta que las mejoras futuras a la funcionalidad de configuración se agregarán a **`dictConfig()`**, por lo que vale la pena considerar la transición a esta API más nueva cuando sea conveniente hacerlo.

A continuación se dan ejemplos de estas secciones en el archivo.

```
[loggers]
keys=root,log02,log03,log04,log05,log06,log07

[handlers]
keys=hand01,hand02,hand03,hand04,hand05,hand06,hand07,ha

[formatters]
keys=form01,form02,form03,form04,form05,form06,form07,fo
```

El registrador raíz debe especificar un nivel y una lista de gestores. A continuación se muestra un ejemplo de una sección de registrador raíz.

```
[logger_root]
level=NOTSET
handlers=hand01
```

La entrada `level` puede ser una de `DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL` o `NOTSET`. Solo para el registrador raíz, `NOTSET` significa que todos los mensajes se registrarán. Los valores de nivel son **`evaluados`** en el contexto del espacio de nombres del paquete `logging`.

La entrada `handlers` es una lista separada por comas de nombres de gestores, que debe aparecer en la sección `[handlers]`. Estos nombres deben aparecer en la sección `[handlers]` y tener las secciones correspondientes en el archivo de configuración.

Para los registradores que no sean el registrador raíz, se requiere información adicional. Esto se ilustra en el siguiente ejemplo.

```
[logger_parser]
level=DEBUG
```

```
handlers=hand01
propagate=1
qualname=compiler.parser
```

Las entradas `level` y `handlers` se interpretan como para el registrador raíz, excepto que si el nivel de un registrador que no sea raíz se especifica como `NOTSET`, el sistema consulta a los registradores más arriba en la jerarquía para determinar el nivel efectivo del registrador. La entrada `propagate` se establece en 1 para indicar que los mensajes deben propagarse a los gestores que están más arriba en la jerarquía del registrador, o 0 para indicar que los mensajes **no** se propagan a los gestores en la jerarquía superior. La entrada `qualname` es el nombre jerárquico del canal del registrador, es decir, el nombre utilizado por la aplicación para obtener el registrador.

Las secciones que especifican la configuración del gestor se ejemplifican a continuación.

```
[handler_hand01]
class=StreamHandler
level=NOTSET
formatter=form01
args=(sys.stdout,)
```

La entrada `class` indica la clase del gestor (según lo determinado por **`eval()`** en el espacio de nombres del paquete `logging`). El `level` se interpreta como para los registradores, y `NOTSET` se entiende como “registrar todo”.

La entrada `formatter` indica el nombre clave del formateador para este gestor. Si está en blanco, se utiliza un formateador predeterminado (`logging._defaultFormatter`). Si se especifica un nombre, debe aparecer en la sección `[formatters]` y tener una sección correspondiente en el archivo de configuración.

La entrada `args`, cuando es **`evaluada`** en el contexto del espacio de nombres del paquete `logging`, es la lista de argumentos para el constructor de la clase gestor. Consulte los constructores de los gestores relevantes, o los ejemplos a continuación, para ver cómo se construyen las entradas típicas. Si no se proporciona, el valor predeterminado es `()`.

La entrada opcional `kwargs`, cuando es **`evaluada`** en el contexto del espacio de nombres del paquete `logging`, es el diccionario generado a partir de los argumentos de palabra clave para el constructor de la clase gestor. Si no se proporciona, el valor predeterminado es `{}`.

```
[handler_hand02]
class=FileHandler
level=DEBUG
formatter=form02
args=('python.log', 'w')

[handler_hand03]
class=handlers.SocketHandler
level=INFO
formatter=form03
args=('localhost', handlers.DEFAULT_TCP_LOGGING_PORT)

[handler_hand04]
class=handlers.DatagramHandler
level=WARN
formatter=form04
args=('localhost', handlers.DEFAULT_UDP_LOGGING_PORT)

[handler_hand05]
class=handlers.SysLogHandler
level=ERROR
formatter=form05
args=(('localhost', handlers.SYSLOG_UDP_PORT), handlers.

[handler_hand06]
class=handlers.NTEventLogHandler
level=CRITICAL
formatter=form06
args=('Python Application', '', 'Application')

[handler_hand07]
```

```
class=handlers.SMTPHandler
level=WARN
formatter=form07
args=('localhost', 'from@abc', ['user1@abc', 'user2@xyz'
kwargs={'timeout': 10.0}

[handler_hand08]
class=handlers.MemoryHandler
level=NOTSET
formatter=form08
target=
args=(10, ERROR)

[handler_hand09]
class=handlers.HTTPHandler
level=NOTSET
formatter=form09
args=('localhost:9022', '/log', 'GET')
kwargs={'secure': True}
```

Las secciones que especifican la configuración del formateador se caracterizan por lo siguiente.

```
[formatter_form01]
format=F1 %(asctime)s %(levelname)s %(message)s %(custom
datefmt=
style=%
validate=True
defaults={'customfield': 'defaultvalue'}
class=logging.Formatter
```

Los argumentos para la configuración del formateador son los mismos que las claves en el esquema del diccionario formatters section.

The `defaults` entry, when evaluated in the context of the `logging` package’s namespace, is a dictionary of default values for custom formatting fields. If not provided, it defaults to `None`.

Nota

Debido al uso de `eval()` como se describió anteriormente, existen riesgos potenciales de seguridad que resultan del uso de `listen()` para enviar y recibir configuraciones a través de sockets. Los riesgos se limitan a donde múltiples usuarios sin confianza mutua ejecutan código en la misma máquina; consulte la documentación de `listen()` para obtener más información.

Ver también

Módulo `logging`
Referencia de API para el módulo de registro.

Módulo `logging.handlers`
Gestores útiles incluidos con el módulo de registro.

`logging.handlers` — Logging handlers

Código fuente Lib/logging/handlers.py [https://github.com/python/cpython/tree/3.14/Lib/logging/handlers.py]

Important

Esta página contiene solo información de referencia. Para tutoriales por favor véase

- Tutorial Básico
- Tutorial avanzado
- Libro de cocina de *Logging*

Estos gestores son muy útiles y están provistos en este paquete. Nota que tres de los gestores de las clases (`StreamHandler`, `FileHandler` and `NullHandler`) están definidos en propio módulo `logging` pero fueron documentados aquí junto con los otros gestores.

StreamHandler

La clase `StreamHandler` ubicada en el paquete núcleo `logging` envía la salida del *logging* a un *stream* como *sys.stdout*, *sys.stderr* o cualquier objeto tipo archivo (o mas precisamente cualquier objeto que soporte los métodos `write()` y `flush()`).

class logging.StreamHandler(*stream=None*)

Retorna una nueva instancia de la clase `StreamHandler`. Si *stream* esta especificado, la instancia lo usará para la salida del registro, sino se usará *sys.stderr*.

emit(*record*)

Si se especifica un formateador, se utiliza para formatear el registro. Luego, el registro se escribe en el flujo seguido de `terminator`. Si hay información de excepción, se formatea con `traceback.print_exception()` y se agrega al flujo.

flush()

Descarga el *stream* llamando a su método `flush()`. Nota que el método `close()` es heredado de la clase `Handler` y por lo tanto no produce ninguna salida. Por eso muchas veces será necesario invocar al método explícito `flush()`.

setStream(*stream*)

Establece el *stream* de la instancia a un valor especifico, si este es diferente. El anterior *stream* es vaciado antes de que el nuevo *stream* sea establecido.

Parámetros:	**stream** – El *stream* que el gestor debe usar.
Devuelve:	the old stream, if the stream was changed, or `None` if it wasn't.

Added in version 3.7.

terminator

Cadena utilizada como terminador al escribir un registro formateado en un flujo. El valor por defecto es `'\n'`.

Si no quieres una terminación de nueva línea, puedes establecer el atributo `terminator` de la instancia del manejador a la cadena vacía.

En versiones anteriores, el terminador estaba codificado como `'\n'`.

Added in version 3.2.

FileHandler

La clase **`FileHandler`** está localizada en el paquete núcleo **`logging`**, envía la salida del *logging* a un archivo de disco. Hereda la funcionalidad de salida de la clase **`StreamHandler`**.

class logging.FileHandler(*filename*, *mode='a'*, *encoding=None*, *delay=False*, *errors=None*)

Retorna una nueva instancia de la clase **`FileHandler`**. El archivo especificado se abre y se utiliza como flujo para el registro. Si no se especifica *mode*, se utiliza `'a'`. Si *encoding* no es `None`, se utiliza para abrir el archivo con esa codificación. Si *delay* es verdadero, la apertura del archivo se pospone hasta la primera llamada a **`emit()`**. Por defecto, el archivo crece de manera indefinida. Si se especifica *errors*, se usa para determinar cómo se manejan los errores de codificación.

Distinto en la versión 3.6: Así como valores de cadena de caracteres, también se aceptan objetos de la clase **`Path`** para el argumento «*filename*».

Distinto en la versión 3.9: Se agregó el parámetro *errors*.

close()

Cierra el archivo.

emit(*record*)

Da la salida del registro al archivo.

Ten en cuenta que si el archivo se cerró debido al apagado del registro al salir y el modo de archivo es “w”, el registro no se emitirá (consulta bpo-42378 [https://bugs.python.org/issue?@action=redirect&bpo=42378]).

NullHandler

Added in version 3.1.

La clase `NullHandler` está ubicada en el núcleo biblioteca `logging` . No realiza ningún formateo o salida. Es en esencia un gestor "no-op" para uso de desarrolladores de bibliotecas.

class logging.NullHandler

Retorna una nueva instancia de la clase `NullHandler`.

emit(*record*)

Este método no realiza ninguna acción.

handle(*record*)

Este método no realiza ninguna acción.

createLock()

Este método retorna `None` para el bloqueo , dado que no hay una E/S subyacente cuyo acceso se necesite serializar.

Véase Configurando Logging para una biblioteca para mas información en como usar la clase `NullHandler`.

WatchedFileHandler

The `WatchedFileHandler` class, located in the **`logging.handlers`** module, is a **`FileHandler`** which watches the file it is logging to. If the file changes, it is closed and reopened using the file name.

Puede suceder que haya un cambio de archivo por uso de programas como *newsyslog* y *logrotate* que realizan una rotación del archivo log. Este gestor destinado para uso bajo Unix/Linux controla el archivo para ver si hubo cambios desde la última emisión. (Un archivo se considera que cambió si su dispositivo o nodo índice cambió). Si el archivo cambió entonces el anterior *stream* de archivo se cerrará, y se abrirá el nuevo para obtener un nuevo *stream*.

Este gestor no es apropiado para uso bajo Windows porque bajo Windows los archivos log abiertos no se pueden mover o renombrar. *Logging* abre los archivos con bloqueos exclusivos y entonces no hay necesidad de usar el gestor. Por otra parte *ST_INO* no es soportado bajo Windows. La función `stat()` siempre retorna cero para este valor.

class logging.handlers.WatchedFileHandler(*filename*, *mode='a'*, *encoding=None*, *delay=False*, *errors=None*)

Retorna una nueva instancia de la clase `WatchedFileHandler`. El archivo especificado se abre y se utiliza como flujo para el registro. Si no se especifica *mode*, se utiliza `'a'`. Si *encoding* no es `None`, se utiliza para abrir el archivo con esa codificación. Si *delay* es verdadero, la apertura del archivo se pospone hasta la primera llamada a `emit()`. Por defecto, el archivo crece de manera indefinida. Si se proporciona *errors*, determina cómo se manejan los errores de codificación.

Distinto en la versión 3.6: Así como valores de cadena de caracteres, también se aceptan objetos de la clase `Path` para el argumento «*filename*».

Distinto en la versión 3.9: Se agregó el parámetro *errors*.

reopenIfNeeded()

Revisa si el archivo cambió. Si hubo cambio, el *stream* existente se vacía y cierra y el archivo se abre nuevamente. Típicamente es un precursor para dar salida del registro a un archivo.

Added in version 3.6.

emit(*record*)

Da salida al registro a un archivo, pero primero invoca al método `reopenIfNeeded()` para reabrir el archivo si es que cambió.

BaseRotatingHandler

The `BaseRotatingHandler` class, located in the **`logging.handlers`** module, is the base class for the rotating file handlers, `RotatingFileHandler` and `TimedRotatingFileHandler`. You should not need to instantiate this class, but it has attributes and methods you may need to override.

class logging.handlers.BaseRotatingHandler(*filename*, *mode*, *encoding=None*, *delay=False*, *errors=None*)

Los parámetros son como los de la clase **`FileHandler`**. Los atributos son:

namer

Si este atributo se establece como invocable, el método `rotation_filename()` delega a este invocable. Los parámetros pasados al invocable son aquellos pasados al método **`rotation_filename()`**.

Nota

La función de nombrado es invocada unas cuantas veces durante el volcado (*rollover*) , entonces debe ser tan simple y rápida como sea posible. Debe también retornar siempre la misma salida para una misma entrada, de otra manera el volcado puede no funcionar como se espera.

También vale la pena señalar que se debe tener cuidado al usar un nombrador para conservar ciertos atributos en el nombre de archivo que se usan durante la rotación. Por ejemplo, `RotatingFileHandler` espera tener un conjunto de archivos de registro cuyos nombres contengan números enteros sucesivos, para que la rotación funcione como se espera, y `TimedRotatingFileHandler` elimina los archivos de registro antiguos (según el parámetro `backupCount` pasado al inicializador del controlador), determinando los archivos más antiguos para eliminar. Para que esto suceda, los nombres de los archivos deben poder ordenarse utilizando la parte de fecha/hora del nombre del archivo, y un nombrador debe respetar esto. (Si se desea un nombrador que no respete este esquema, deberá usarse en una subclase de **`TimedRotatingFileHandler`** que anula el método `getFilesToDelete()` para encajar con el esquema de denominación personalizada.)

Added in version 3.3.

rotator

Si este atributo se estableció como invocable, el método `rotate()` delega a este invocable. Los parámetros pasados al invocable son aquellos pasados al método **`rotate()`**.

Added in version 3.3.

rotation_filename(*default_name*)

Modifica el nombre de un archivo log cuando esta rotando.

Esto esta previsto para que pueda usarse un nombre de archivo personalizado.

La implementación por defecto llama al atributo "namer" del gestor, si este es invocable, pasando el nombre por defecto a él. Si el atributo no es invocable (por defecto es `None`) el nombre se retorna sin cambios.

Parámetros:

default_name – El nombre por defecto para el archivo de log.

Added in version 3.3.

rotate(*source, dest*)

Cuando está rotando, rotar el actual log.

La implementación por defecto llama al atributo “rotator” del gestor, si es invocable, pasando los argumentos de origen y destino a él. Si no se puede invocar (porque el atributo por defecto es “None”) el origen es simplemente renombrado al destino.

Parámetros:

- **source** – El nombre de archivo origen . Normalmente el nombre de archivo base, por ejemplo “test.log”.
- **dest** – El nombre de archivo de destino. Normalmente es el nombre al que se rota el archivo origen por ejemplo “test.log.1”.

Added in version 3.3.

La razón de que existen los atributos es para evitar tener que usar una subclase - puedes usar los mismos invocadores para instancias de clase `RotatingFileHandler` y `TimedRotatingFileHandler`. Si el rotador invocable o la función de nombrado plantean una excepción esta se manejará de la misma manera que cualquier otra excepción durante una llamada al método `emit()` por ejemplo a través del método `handleError()` del gestor.

Si necesitas hacer cambios mas significativos al proceso de rotación puedes obviar los métodos.

Para un ejemplo véase Usar un rotador y un nombre para personalizar el procesamiento de rotación de log.

RotatingFileHandler

The `RotatingFileHandler` class, located in the `logging.handlers` module, supports rotation of disk log files.

class logging.handlers.RotatingFileHandler(*filename*, *mode='a'*, *maxBytes=0*, *backupCount=0*, *encoding=None*, *delay=False*, *errors=None*)

Retorna una nueva instancia de la clase **`RotatingFileHandler`**. El archivo especificado se abre y se utiliza como flujo para el registro. Si no se especifica *mode*, se utiliza `'a'`. Si *encoding* no es `None`, se utiliza para abrir el archivo con esa codificación. Si *delay* es verdadero, la apertura del archivo se aplaza hasta la primera llamada a **`emit()`**. De forma predeterminada, el archivo crece indefinidamente. Si se proporciona *errors*, determina cómo se manejan los errores de codificación.

Se pueden usar los valores *maxBytes* y *backupCount* para permitir que el archivo *rollover* tenga un tamaño predeterminado. Cuando el tamaño del archivo está a punto de excederse, se cerrará y un nuevo archivo se abrirá silenciosamente para salida. El volcado (*rollover*) ocurre cada vez que el actual archivo log esta cerca de *maxBytes* en tamaño , pero si cualquiera *maxBytes* o *backupCount* es cero, el volcado (*rollover*) no ocurre. Por eso generalmente necesitas establecer *backupCount* por lo menos en 1 y no tener cero en *maxBytes*. Cuando *backupCount* no es cero, el sistema guardará los anteriores archivos log agregando las extensiones “.1”, “.2” etc. al nombre del archivo. Por ejemplo con un *backupCount* de 5 y un nombre de archivo base de `app.log`, tendrás como nombre de archivo t `app.log`, `app.log.1`, `app.log.2`, hasta `app.log.5`. El archivo que esta siendo escrito es siempre `app.log`. Cuando este se completa , se cierra y se renombra a `app.log.1` y si ya existen `app.log.1`, `app.log.2`, etc. Entonces se renombrará como `app.log.2`, `app.log.3` etc. respectivamente.

Distinto en la versión 3.6: Así como valores de cadena de caracteres, también se aceptan objetos de la clase **`Path`** para el argumento «*filename*».

Distinto en la versión 3.9: Se agregó el parámetro *errors*.

doRollover()

Realiza un volcado (*rollover*) como se describe arriba.

emit(*record*)

Da la salida del registro al archivo , dando suministro para el volcado (*rollover*) como está descripto anteriormente.

shouldRollover(*record*)

See if the supplied record would cause the file to exceed the configured size limit.

TimedRotatingFileHandler

The **`TimedRotatingFileHandler`** class, located in the **`logging.handlers`** module, supports rotation of disk log files at certain timed intervals.

class logging.handlers.TimedRotatingFileHandler(*filename*, *when='h'*, *interval=1*, *backupCount=0*, *encoding=None*, *delay=False*, *utc=False*, *atTime=None*, *errors=None*)

Retorna una nueva instancia de la clase **`TimedRotatingFileHandler`** . El archivo especificado es abierto y usado como *stream* para el historial de log. En la rotación también establece el sufijo del nombre de archivo. La rotación ocurre basada en el producto de *when* y *interval*.

Puedes usar el *when* para especificar el tipo de intervalo *interval*. La lista de posibles valores esta debajo. Nota que no distingue mayúsculas y minúsculas.

Valor	**Tipo de intervalo**	**Si/como *atTime* es usado**
'S'	Segundos	Ignorado
'M'	Minutos	Ignorado
'H'	Horas	Ignorado
'D'	Días	Ignorado
'W0'-'W6'	Día de la semana (0 = Lunes)	Usado para calcular la hora inicial del volcado *rollover*
'midnight'	Volcado (*rollover*) a medianoche, si *atTime* no está	Usado para calcular la hora inicial del volcado *rollover*

especificado, sino el volcado se hará *atTime*

Cuando se usa rotación basada en día de la semana, especifica “W0” para Lunes, “W1” para Martes y así, hasta “W6” para Domingo. en este caso el valor pasado para *Interval* no se usa.

El sistema guardará los archivos de log anteriores agregándoles extensiones al nombre de archivo. Las extensiones están basadas en día-hora usando el formato `%Y-%m-%d_%H-%M-%S` o un prefijo respecto el intervalo del volcado (*rollover*).

Cuando se calcula la hora del siguiente volcado (*rollover*) por primera vez (cuando el gestor es creado), la última hora de modificación de un archivo log existente o sino la hora actual, se usa para calcular cuando será la próxima rotación.

Si el argumento *utc* es *true* se usará la hora en UTC, sino se usará la hora local.

Si *backupCount* no es cero, se conservará como máximo una cantidad de archivos especificada en *backupCount*,y si son creados más, cuando ocurre el volcado (*rollover*) se borrará el último. La lógica de borrado usa el intervalo para determinar que archivos borrar, pues entonces cambiando el intervalo puede dejar viejos archivos abandonados.

Si *delay* es *true* entonces la apertura del archivo se demorará hasta la primer llamada a **`emit()`**.

Si *atTime* no es «None», debe haber una instancia `datetime.time` que especifica la hora que ocurre el volcado (*rollover*) , para los casos en que el volcado esta establecido para ocurrir «a medianoche» o «un día en particular». Nótese que en estos casos el valor *atTime* se usa para calcular el valor *initial* del volcado (*rollover*) y los subsecuentes volcados serán calculados a través del calculo normal de intervalos.

Si se especifica *errors*, se utiliza para determinar cómo se manejan los errores de codificación.

Nota

El cálculo de la hora en que se realizara el volcado (*rollover*) inicial cuando se inicializa el gestor. El cálculo de la hora de los siguientes volcados (*rollovers*) se realiza solo cuando este ocurre, y el volcado ocurre solo cuando se emite una salida. Si esto no se tiene en cuenta puede generar cierta confusión. Por ejemplo si se establece un intervalo de «cada minuto» eso no significa que siempre se verán archivos log con hora (en el nombre del archivo) separados por un minuto. Si durante la ejecución de la aplicación el *logging* se genera con mayor frecuencia que un minuto entonces se pueden esperar archivos log separados por un minuto. Si por otro lado los mensajes *logging* son establecidos cada digamos cinco minutos, entonces habrá brechas de tiempo en los archivos correspondientes a los minutos que no hubo salida (y no ocurrió por tanto volcado alguno).

Distinto en la versión 3.4: Se agregó el parámetro *atTime*.

Distinto en la versión 3.6: Así como valores de cadena de caracteres, también se aceptan objetos de la clase `Path` para el argumento «*filename*».

Distinto en la versión 3.9: Se agregó el parámetro *errors*.

doRollover()

Realiza un volcado (*rollover*) como se describe arriba.

emit(*record*)

Da la salida del registro a un archivo , proveyendo la información para el volcado (*rollover*) como esta descripto anteriormente.

getFilesToDelete()

Returns a list of filenames which should be deleted as part of rollover. These are the absolute paths of the oldest backup log files written by the handler.

shouldRollover(*record*)

See if enough time has passed for a rollover to occur and if it has, compute the next rollover time.

SocketHandler

The `SocketHandler` class, located in the **`logging.handlers`** module, sends logging output to a network socket. The base class uses a TCP socket.

class logging.handlers.SocketHandler(*host*, *port*)

Retorna una nueva instancia de la clase `SocketHandler` destinada para comunicarse con una terminal remota cuya dirección esta dada por *host* y *port*.

Distinto en la versión 3.4: Si «port» se especifica como `None` se crea un socket de dominio Unix, usando el valor en `host` - de otra manera se creará un socket TCP.

close()

Cierra el socket.

emit()

Serializa (*Pickles*) el registro del diccionario de atributos y lo escribe en el socket en formato binario. Si hay un error con el socket, silenciosamente descarta el paquete. Si la conexión se perdió previamente, la restablece. Para deserializar (*unpickle*) un registro en el extremo receptor a una clase `LogRecord`, usa la función `makeLogRecord()`.

handleError()

Maneja un error que ocurrió durante el método `emit()`. La causa mas común es una perdida de conexión. Cierra el socket para que podamos reintentar en el próximo evento.

makeSocket()

Este es un método patrón que permite subclases para definir el tipo preciso de socket que se necesita. La implementación por defecto crea un socket TCP(**`socket.SOCK_STREAM`**).

makePickle(*record*)

Serializa (*pickles*) el registro del diccionario de atributos en formato binario con un prefijo de tamaño, y lo retorna listo para transmitir a través del socket. Los detalles de esta operación son equivalentes a:

```
data = pickle.dumps(record_attr_dict, 1)
datalen = struct.pack('>L', len(data))
return datalen + data
```

Nota que los serializados (*pickles*) no son totalmente seguros. Si te preocupa la seguridad desearás evitar este método para implementar un mecanismo mas seguro. Por ejemplo puedes firmar *pickles* usando HMAC y verificarlos después en el extremo receptor. O alternativamente puedes deshabilitar la deserialización (*unpickling*) de objetos globales en el extremo receptor.

send(*packet*)

Envía un paquete serializado (*pickled*) de cadena de caracteres al socket. El formato de la cadena de bytes enviada es tal como se describe en la documentación de **`makePickle()`**.

Esta función permite envíos parciales, que pueden ocurrir cuando la red esta ocupada.

createSocket()

Intenta crear un socket, si hay una falla usa un algoritmo de marcha atrás exponencial. En el fallo inicial el gestor desechará el mensaje que intentaba enviar. Cuando los siguientes mensajes sean gestionados por la misma instancia no intentará conectarse hasta que haya transcurrido cierto tiempo. Los parámetros por defecto son tales que el retardo inicial es un segundo y si después del retardo la conexión todavía no se puede realizar, el gestor doblará el retardo cada vez hasta un máximo de 30 segundos.

Este comportamiento es controlado por los siguientes atributos del gestor:

- `retryStart` (retardo inicial por defecto 1.0 segundos)
- `retryFactor` (multiplicador por defecto 2.0)
- `retryMax` (máximo retardo por defecto 30.0 segundos)

Esto significa que si el oyente remoto (*listener*) comienza después de que se usó el gestor , pueden perderse mensajes (dado que el gestor no puede siquiera intentar una conexión hasta que se haya cumplido el retardo, y silenciosamente desechará los mensajes mientras se cumpla el retardo).

DatagramHandler

The `DatagramHandler` class, located in the **`logging.handlers`** module, inherits from `SocketHandler` to support sending logging messages over UDP sockets.

class logging.handlers.DatagramHandler(*host*, *port*)

Retorna una nueva instancia de la clase `DatagramHandler` destinada para comunicarse con la terminal remota cuya dirección es dada por *host* y *port*.

Nota

Como UDP no es un protocolo de transmisión, no existe una conexión persistente entre una instancia de este controlador y *host*. Por esta razón, cuando se usa un socket de red, es posible que se deba realizar una búsqueda de

DNS cada vez que se registra un evento, lo que puede generar cierta latencia en el sistema. Si esto te afecta, puedes realizar una búsqueda tu mismo e inicializar este controlador utilizando la dirección IP buscada en lugar del nombre de host.

Distinto en la versión 3.4: Si "port" se especifica como «None», se crea un socket de dominio Unix, usando el valor en «host» - de otra manera se crea un socket UDP.

emit()

Serializa (*pickles*) el registro del diccionario de atributos y lo escribe en el socket en formato binario. Si hay un error con el socket, silenciosamente desecha el paquete. Para deserializar (*unpickle*) el registro en el extremo de recepción a una clase `LogRecord`, usa la función `makeLogRecord()`.

makeSocket()

El método original de la clase `SocketHandler` se omite para crear un socket UDP (`socket.SOCK_DGRAM`).

send(*s*)

Enviar una cadena de caracteres serializada (*pickled*) a un socket de red. El formato de la cadena de *bytes* enviado es tal como se describe en la documentación para `SocketHandler.makePickle()`.

Gestor *SysLog* (*SysLogHandler*)

The `SysLogHandler` class, located in the `logging.handlers` module, supports sending logging messages to a remote or local Unix syslog.

class logging.handlers.SysLogHandler(*address=('localhost', SYSLOG_UDP_PORT)*, *facility=LOG_USER*, *socktype=socket.SOCK_DGRAM, timeout=None*)

Returns a new instance of the **`SysLogHandler`** class intended to communicate with a remote Unix machine whose address is given by *address* in the form of a `(host, port)` tuple. If *address* is not specified, `('localhost', 514)` is used. The address is used to open a socket. An alternative to providing a `(host, port)` tuple is providing an address as a string, for example "/dev/log". In this case, a Unix domain socket is used to send the message to the syslog. If *facility* is not specified, **`LOG_USER`** is used. The type of socket opened depends on the *socktype* argument, which defaults to **`socket.SOCK_DGRAM`** and thus opens a UDP socket. To open a TCP socket (for use with the newer syslog daemons such as rsyslog), specify a value of **`socket.SOCK_STREAM`**. If *timeout* is specified, it sets a timeout (in seconds) for the socket operations. This can help prevent the program from hanging indefinitely if the syslog server is unreachable. By default, *timeout* is `None`, meaning no timeout is applied.

Nótese que si el servidor no esta escuchando el puerto UDP 514, la clase **`SysLogHandler`** puede parecer no funcionar. En ese caso chequea que dirección deberías usar para un socket de dominio . Es sistema-dependiente. Por ejemplo en Linux generalmente es "/dev/log" pero en OS/X es "/var/run/syslog". Será necesario chequear tu plataforma y usar la dirección adecuada (quizá sea necesario realizar este chequeo mientras corre la aplicación si necesita correr en diferentes plataformas). En Windows seguramente tengas que usar la opción UDP.

Nota

En macOS 12.x (Monterey), Apple cambió el comportamiento de su syslog daemon: ya no escucha en un socket de dominio. Por lo tanto, no puedes esperar que **`SysLogHandler`** funcione en este sistema.

Ver gh-91070 [https://github.com/python/cpython/issues/91070] para más información.

Distinto en la versión 3.2: Se agregó *socktype*.

Distinto en la versión 3.14: *timeout* was added.

close()

Cierra el socket del host remoto.

createSocket()

Tries to create a socket and, if it's not a datagram socket, connect it to the other end. This method is called during handler initialization, but it's not regarded as an error if the other end isn't listening at this point - the method will be called again when emitting an event, if there is no socket at that point.

Added in version 3.11.

emit(*record*)

El registro es formateado, y luego enviado al servidor *syslog*. Si hay información de excepción presente entonces no se enviará al servidor.

Distinto en la versión 3.2.1: (Véase el bpo-12168 [https://bugs.python.org/issue?@action=redirect&bpo=12168].) En versiones anteriores , los mensajes enviados a los *daemons syslog* siempre terminaban con un byte NUL ya que versiones anteriores de estos *daemons* esperaban un mensaje NUL de terminación. Incluso a pesar que no es relevante la especificación (**RFC 5424** [https://datatracker.ietf.org/doc/html/rfc5424.html]). Versiones mas recientes de estos *daemons* no esperan el byte NUL pero lo quitan si esta ahí. Versiones aún mas recientes que están mas cercanas a la especificación RFC 5424 pasan el byte NUL como parte del mensaje.

Para habilitar una gestión mas sencilla de los mensajes *syslog* respecto de todos esos *daemons* de diferentes comportamientos el agregado del byte NUL es configurable a través del uso del atributo de nivel de clase `append_nul`. Este es por defecto “`True` (preservando el comportamiento ya existente) pero se puede establecer a “False” en una instancia `SysLogHandler` como para que esa instancia no añada el terminador NUL.

Distinto en la versión 3.3: (Véase: issue “12419”) en versiones anteriores, no había posibilidades para un prefijo “ident” o “tag” para identificar el origen del mensaje. Esto ahora se puede especificar usando un atributo de nivel de clase, que por defecto será «””» para preservar el comportamiento existente , pero puede ser sorteado en una instancia `SysLogHandler` para que esta instancia anteponga el *ident* a todos los mensajes gestionados. Nótese que el *ident* provisto debe ser texto, no bytes y se antepone al mensaje tal como es.

encodePriority(*facility*, *priority*)

Codifica la funcionalidad y prioridad en un entero. Puedes pasar cadenas de caracteres o enteros, si pasas cadenas de caracteres se usarán los diccionarios de mapeo interno para convertirlos en enteros.

Los valores simbólicos `LOG_` están definidos en **`SysLogHandler`** e invierten los valores definidos en el archivo de encabezado `sys/syslog.h`.

Prioridades

Nombre (cadena de caracteres)	Valor simbólico
`alert`	LOG_ALERT
`crit` or `critical`	LOG_CRIT
`debug`	LOG_DEBUG
`emerg` or `panic`	LOG_EMERG
`err` or `error`	LOG_ERR
`info`	LOG_INFO
`notice`	LOG_NOTICE
`warn` or `warning`	LOG_WARNING

Facilities

Nombre (cadena de caracteres)	Valor simbólico
auth	LOG_AUTH
authpriv	LOG_AUTHPRIV
cron	LOG_CRON
daemon	LOG_DAEMON
ftp	LOG_FTP
kern	LOG_KERN
lpr	LOG_LPR
mail	LOG_MAIL
news	LOG_NEWS
syslog	LOG_SYSLOG
user	LOG_USER
uucp	LOG_UUCP
local0	LOG_LOCAL0
local1	LOG_LOCAL1
local2	LOG_LOCAL2
local3	LOG_LOCAL3
local4	LOG_LOCAL4
local5	LOG_LOCAL5
local6	LOG_LOCAL6
local7	LOG_LOCAL7

mapPriority(*levelname*)

Mapea un nombre de nivel *logging* a un nombre de prioridad *syslog*. Puedes necesitar omitir esto si estas usando niveles personalizados, o si el algoritmo por defecto no es aplicable a tus necesidades. El algoritmo por defecto mapea `DEBUG`, `INFO`, `WARNING`, `ERROR` y `CRITICAL` a sus nombres equivalentes *syslog*, y todos los demás nombres de nivel a "warning".

Gestor de eventos *NTELog* (NTEventLogHandler)

The **`NTEventLogHandler`** class, located in the **`logging.handlers`** module, supports sending logging messages to a local Windows NT, Windows 2000 or Windows XP event log. Before you can use it, you need Mark Hammond’s Win32 extensions for Python installed.

class logging.handlers.NTEventLogHandler(*appname, dllname=None, logtype='Application'*)

Retorna una nueva instancia de la clase **`NTEventLogHandler`** la *appname* se usa para definir el nombre de la aplicación tal como aparece en el log de eventos. Se crea una entrada de registro apropiada usando este nombre. El *dllname* debe dar la ruta completa calificada de un .dll o .exe que contiene definiciones de mensaje para conservar en el log. (si no esta especificada, se usara `'win32service.pyd'` esto es instalado con las extensiones de Win32 y contiene algunas definiciones básicas de mensajes de conservación de lugar. Nótese que el uso de estos conservadores de lugar harán tu log de eventos extenso, dado que el origen completo del mensaje es guardado en el log. Si quieres *logs* menos extensos deberás pasar el nombre de tu propio .dll o .exe que contiene la definición de mensajes que quieres usar en el log. El *logtype* puede ser de `'Application'`, `'System'` or `'Security'` y sino por defecto será de `'Application'`.

close()

Llegado a este punto puedes remover el nombre de aplicación del registro como origen de entrada de log de eventos. Sin embargo si haces esto no te será posible ver los eventos tal como has propuesto en el visor del log de eventos - necesita ser capaz de acceder al registro para tomar el nombre .dll. Esto no lo hace la versión actual.

emit(*record*)

Determina el ID del mensaje, categoría y tipo de evento y luego registra el mensaje en el log de eventos NT.

getEventCategory(*record*)

Retorna la categoría de evento del registro. Evita esto si quieres especificar tus propias categorías. Esta versión retorna 0.

getEventType(*record*)

Retorna el tipo de evento del registro. Haz caso omiso de esto si quieres especificar tus propios tipos. Esta versión realiza un mapeo usando el atributo *typemap* del gestor, que se establece en `__init__()` a un diccionario que contiene mapeo para `DEBUG`, `INFO`, `WARNING`, `ERROR` y `CRITICAL`. Si estas usando tus propios niveles, necesitarás omitir este método o colocar un diccionario a medida en el atributo *typemap* del gestor.

getMessageID(*record*)

Retorna el ID de mensaje para el registro. Si estas usando tus propios mensajes, podrás hacerlo pasando el *msg* al *logger* siendo un ID mas que un formato de cadena de caracteres. Luego aquí puedes usar una búsqueda de diccionario para obtener el ID de mensaje. Esta versión retorna 1, que es el ID de mensaje base en `win32service.pyd`.

SMTPHandler

The `SMTPHandler` class, located in the `logging.handlers` module, supports sending logging messages to an email address via SMTP.

class logging.handlers.SMTPHandler(*mailhost*, *fromaddr*, *toaddrs*, *subject*, *credentials=None*, *secure=None*, *timeout=1.0*)

Retorna una nueva instancia de la clase `SMTPHandler`. Esta instancia se inicializa con la dirección de: y para: y asunto: del correo electrónico. El *toaddrs* debe ser una lista de cadena de caracteres. Para especificar un puerto SMTP no estandarizado usa el formato de tupla (host, puerto) para el argumento *mailhost*. Si usas una cadena de caracteres, se utiliza el puerto estándar SMTP. Si tu servidor SMTP necesita autenticación, puedes especificar una tupla (usuario, contraseña) para el argumento de *credentials*.

Para especificar el uso de un protocolo de seguridad (TLS), pasa una tupla al argumento *secure*. Esto solo se utilizará cuando sean provistas las credenciales de autenticación. La tupla deberá ser una tupla vacía o una tupla con único valor con el nombre de un archivo-clave *keyfile*, o una tupla de 2 valores con el nombre del archivo-clave *keyfile* y archivo certificado. (Esta tupla se pasa al método `smtplib.SMTP.starttls()` method.).

Se puede especificar un tiempo de espera para comunicación con el servidor SMTP usando el argumento *timeout*.

Distinto en la versión 3.3: Added the *timeout* parameter.

emit(*record*)

Formatea el registro y lo envía a las direcciones especificadas.

getSubject(*record*)

Si quieres especificar una línea de argumento que es registro-dependiente, sobrescribe (*override*) este método.

MemoryHandler

The `MemoryHandler` class, located in the **`logging.handlers`** module, supports buffering of logging records in memory, periodically flushing them to a *target* handler. Flushing occurs whenever the buffer is full, or when an event of a certain severity or greater is seen.

`MemoryHandler` es una subclase de la clase mas general `BufferingHandler`, que es una clase abstracta. Este almacena temporalmente registros *logging* en la memoria. Cada vez que un registro es agregado al búfer, se realiza una comprobación llamando al método **`shouldFlush()`** para ver si dicho búfer debe ser descargado. Si debiera, entonces el método **`flush()`** se espera que realice la descarga.

class logging.handlers.BufferingHandler(*capacity*)

Inicializa el gestor con un búfer de una capacidad especifica. Aquí capacidad significa el número de registros *logging* en el almacenamiento temporal.

emit(*record*)

Añade un registro al búfer. Si el método **`shouldFlush()`** retorna *true* , entonces llama al método **`flush()`** para procesar el búfer.

flush()

Para una instancia de **`BufferingHandler`**, vaciar significa que establece el búfer a una lista vacía. Este método puede ser sobrescrito para implementar un comportamiento de vaciado más útil.

shouldFlush(*record*)

Retorna `True` si el búfer tiene aún capacidad. Este método puede ser omitido para implementar estrategias a medida de vaciado.

class logging.handlers.MemoryHandler(*capacity*, *flushLevel=ERROR*, *target=None*, *flushOnClose=True*)

Retorna una nueva instancia de la clase **`MemoryHandler`** . La instancia se inicializa con un búfer del tamaño *capacity*. Si el *flushLevel* no se especifica, se usará **`ERROR`** . Si no se especifica *target* el objetivo deberá especificarse usando el método **`setTarget()`** -Antes de esto el gestor no realizará nada útil. Si se especifica *flushOnClose* como `False`, entonces el búfer no se vaciará cuando el gestor se cierra. Si no se especifica o se especifica como `True`, el comportamiento previo de vaciado del búfer sucederá cuando se cierre el gestor.

Distinto en la versión 3.6: Se añadió el parámetro *flushOnClose*.

close()

Invoca al método **`flush()`** y establece el objetivo a “None” y vacía el búfer.

flush()

Para una instancia de **`MemoryHandler`**, el vaciado implica simplemente enviar los registros almacenados en el búfer al objetivo, si es que existe uno. El búfer también se limpia cuando se envían los registros almacenados al objetivo. Sobrescribe si deseas un comportamiento diferente.

setTarget(*target*)

Establece el gestor de objetivo para este gestor.

shouldFlush(*record*)

Comprueba si el búfer esta lleno o un registro igual o mas alto que *flushLevel*.

HTTPHandler

The **`HTTPHandler`** class, located in the **`logging.handlers`** module, supports sending logging messages to a web server, using either `GET` or `POST` semantics.

class logging.handlers.HTTPHandler(*host*, *url*, *method='GET'*, *secure=False*, *credentials=None*, *context=None*)

Retorna una nueva instancia de la clase **`HTTPHandler`**. el *host* puede ser de la forma «host:puerto», y necesitarás usar un numero de puerto especifico. Si no se especifica *method* se usará `GET` . Si *secure* es true se usará una conexión HTTPS. El parámetro *context* puede establecerse a una instancia **`ssl.SSLContext`** para establecer la configuración de SSL usado en la conexión HTTPS. Si se especifica *credentials* debe ser una tupla doble, consistente en usuario y contraseña, que se colocará en un encabezado de autorización HTTP usando autenticación básica. Si especificas credenciales también deberás especificar *secure=True* así tu usuario y contraseña no son pasados como texto en blanco por la conexión.

Distinto en la versión 3.5: Se agregó el parámetro *context*.

mapLogRecord(*record*)

Provee un diccionario, basado en `record` para ser codificado en forma URL y enviado al servidor web. La implementación por defecto retorna `record.__dict__`. Este método puede omitirse si por ejemplo solo se enviará al servidor web un subconjunto de la clase `LogRecord` o si se requiere enviar al servidor algo mas específico.

emit(*record*)

Envía el registro al servidor web como un diccionario codificado en URL. El método `mapLogRecord()` se utiliza para convertir el registro al diccionario que se enviará.

Nota

Dado que preparar un registro para enviarlo a un servidor web no es lo mismo que una operación de formateo genérico, el uso de `setFormatter()` para especificar un `Formatter` para un `HTTPHandler` no tiene ningún efecto. En lugar de llamar a `format()`, este controlador llama a `mapLogRecord()` y luego a `urllib.parse.urlencode()` para codificar el diccionario en una forma adecuada para enviar a un servidor web.

QueueHandler

Added in version 3.2.

The `QueueHandler` class, located in the `logging.handlers` module, supports sending logging messages to a queue, such as those implemented in the `queue` or `multiprocessing` modules.

Junto con la clase `QueueListener`, `QueueHandler` se puede usar para permitir que los controladores hagan su trabajo en un hilo separado del que realiza el registro. Esto es importante en aplicaciones web y también en otras aplicaciones de servicio donde los subprocesos que atienden a los clientes deben responder lo más rápido posible, mientras que cualquier operación potencialmente lenta (como enviar un correo electrónico a través de `SMTPHandler`) se realiza en un subproceso separado.

class logging.handlers.QueueHandler(*queue*)

Retorna una nueva instancia de la clase `QueueHandler`. La instancia se inicializa con la cola a la que se enviarán los mensajes. La cola puede ser cualquier objeto tipo-cola; es usado tal como por el método `enqueue()` que necesita saber como enviar los mensajes a ella. La cola no es *requerida* para tener una API de rastreo de tareas, lo que significa que puedes usar instancias de `SimpleQueue` como *queue*.

Nota

Si estás usando `multiprocessing`, debes evitar usar `SimpleQueue` y en su lugar usar `multiprocessing.Queue`.

Advertencia

The `multiprocessing` module uses an internal logger created and accessed via `get_logger()`. `multiprocessing.Queue` will log `DEBUG` level messages upon items being queued. If those log messages are processed by a `QueueHandler` using the same `multiprocessing.Queue` instance, it will cause a deadlock or infinite recursion.

emit(*record*)

Pone en cola el resultado de preparar el LogRecord. Si ocurre una excepción (por ejemplo, porque una cola limitada se ha llenado), se invoca el método **`handleError()`** para manejar el error. Esto puede resultar en que el registro se descarte de manera silenciosa (si **`logging.raiseExceptions`** es `False`) o en que se imprima un mensaje en `sys.stderr` (si **`logging.raiseExceptions`** es `True`).

prepare(*record*)

Prepara un registro para poner en la cola. El objeto que retorna este método se colocará en cola.

La implementación base le da formato al registro para fusionar el mensaje, los argumentos, la excepción y la información de la pila, si está presente. También elimina elementos que no se pueden serializar (*pickle*) del registro en el lugar. Específicamente, sobrescribe los atributos **`msg`** y **`message`** del registro con el mensaje fusionado (obtenido llamando al método **`format()`** del controlador) y establece los atributos **`args`**, **`exc_info`** y **`exc_text`** a `None`.

Puedes querer hacer caso omiso de este método si quieres convertir el registro en un diccionario o cadena de caracteres JSON, o enviar una copia modificada del registro mientras dejas el original intacto.

Nota

La implementación base formatea el mensaje con argumentos, establece los atributos `message` y `msg` en el mensaje formateado y establece los atributos `args` y `exc_text` en `None` para permitir serializado (*pickle*) y para evitar nuevos intentos de formateo. Esto significa que un controlador en el lado **`QueueListener`** no tendrá la información para hacer un formato personalizado, por ej. de excepciones. Es posible que desees crear una subclase de `QueueHandler` y anular este método para, por ej. evitar establecer `exc_text` en `None`. Ten en cuenta que los cambios de `message` / `msg` / `args` están relacionados con garantizar que el registro se pueda seleccionar, y es posible que puedas o no evitar hacerlo dependiendo de si sus `args` son serializables. (Ten en cuenta que es posible que debas considerar no solo tu propio código, sino también el código en cualquier biblioteca que uses).

enqueue(*record*)

Coloca en la cola al registro usando `put_nowait()`; puede que quieras sobrescribe (*override*) esto si quieres usar una acción de bloqueo, o un tiempo de espera, o una implementación de cola a medida.

listener

Cuando se crea a través de la configuración usando **`dictConfig()`**, este atributo contendrá una instancia de **`QueueListener`** para usar con este manejador. De lo contrario, será `None`.

Added in version 3.12.

QueueListener

Added in version 3.2.

The **`QueueListener`** class, located in the **`logging.handlers`** module, supports receiving logging messages from a queue, such as those implemented in the **`queue`** or **`multiprocessing`** modules. The messages are received from a queue in an internal thread and passed, on the same thread, to one or more handlers for processing. While **`QueueListener`** is not itself a handler, it is documented here because it works hand-in-hand with **`QueueHandler`**.

Junto con la clase **`QueueHandler`**, **`QueueListener`** se puede usar para permitir que los controladores hagan su trabajo en un hilo separado del que realiza el registro. Esto es importante en aplicaciones web y también en otras aplicaciones de servicio donde los subprocesos que atienden a los clientes deben responder lo más rápido posible, mientras que cualquier operación potencialmente lenta (como enviar un correo electrónico a través de `SMTPHandler`) se realiza en un subproceso separado.

class logging.handlers.QueueListener(*queue*, **handlers*, *respect_handler_level=False*)

Retorna una nueva instancia de la clase `QueueListener`. La instancia se inicializa con la cola para enviar mensajes a una lista de gestores que manejarán entradas colocadas en la cola. La cola puede ser cualquier objeto de tipo-cola, es usado tal como está por el método `dequeue()` que necesita saber como tomar los mensajes de esta. La cola no es *obligatoria* para tener la API de seguimiento de tareas (aunque se usa si está disponible), lo que significa que puede usar instancias `SimpleQueue` para *queue*.

Nota

Si estás usando `multiprocessing`, debes evitar usar `SimpleQueue` y en su lugar usar `multiprocessing.Queue`.

Si `respect_handler_level` es `True`, se respeta un nivel de gestor (comparado con el nivel del mensaje) cuando decide si pasar el mensajes al gestor; de otra manera, el comportamiento es como en versiones anteriores de Python - de pasar cada mensaje a cada gestor.

Distinto en la versión 3.5: Se agregó el argumento `respect_handler_levels`.

Distinto en la versión 3.14: `QueueListener` can now be used as a context manager via `with`. When entering the context, the listener is started. When exiting the context, the listener is stopped. `__enter__()` returns the `QueueListener` object.

dequeue(*block*)

Extrae de la cola un registro y lo retorna, con opción a bloquearlo.

La implementación base usa `get()`. Puedes sobrescribir (*override*) este método si quieres usar tiempos de espera o trabajar con colas implementadas a medida.

prepare(*record*)

Prepara un registro para ser gestionado.

Esta implementación solo retorna el registro que fue pasado. Puedes sobrescribir (*override*) este método para hacer una serialización a medida o una manipulación del registro antes de pasarlo a los gestores.

handle(*record*)

Manejar un registro.

Esto solo realiza un bucle a través de los gestores ofreciéndoles el registro para ser gestionado. El objeto actual pasado a los gestores es aquel que es retornado por el método **`prepare()`**.

start()

Da comienzo al oyente (*listener*).

Esto da comienzo a un hilo en segundo plano para supervisar la cola de registros log a procesar.

Distinto en la versión 3.14: Raises **`RuntimeError`** if called and the listener is already running.

stop()

Detiene el oyente (*listener*).

Esto solicita terminar al hilo, y luego espera hasta que termine. Nota que si no llamas a esto antes de que tu aplicación salga, puede haber algunos registros que aun están en la cola, que no serán procesados.

enqueue_sentinel()

Escribe un centinela (*sentinel*) en la cola para decir al oyente (*listener*) de salir. Esta implementación usa `put_nowait()`. Puedes sobrescribir (*override*) este método si quieres usar tiempos de espera o trabajar con implementaciones de cola a tu medida.

Added in version 3.3.

Ver también

Módulo `logging`
Referencia API para el módulo de *logging*.

Módulo `logging.config`
Configuración API para el módulo de *logging*.

`platform` — Access to underlying platform's identifying data

código fuente: Lib/platform.py [https://github.com/python/cpython/tree/3.14/Lib/platform.py]

Nota

Plataformas específicas listadas alfabéticamente, con Linux incluido en la sección de Unix.

Cross platform

platform.architecture(*executable=sys.executable*, *bits=''*, *linkage=''*)

Consulta el ejecutable provisto (por defecto el archivo binario del intérprete de Python) para obtener información de diversas arquitecturas.

Retorna una tupla `(bits, linkage)`, siendo *bits* la información sobre la arquitectura del procesador y *linkage* el formato de conexión usado por el ejecutable. Ambos valores se retornan como cadenas.

Los valores que no se pueden determinar se retornan según lo indicado por los ajustes por defecto de los parámetros. Si bits se da como `''`, el `sizeof(pointer)` (o `sizeof(long)` en la versión de Python < 1.5.2) se utiliza como indicador para el tamaño del puntero admitido.

La función se basa en el comando `file` del sistema para realizar la tarea. Está disponible en la mayoría de las plataformas Unix, si no en todas, y en algunas plataformas que no son de Unix y solo si el ejecutable apunta al intérprete de Python. Unos valores por defecto se utilizan cuando no se satisfacen las necesidades anteriores.

Nota

En macOS (y quizás en otras plataformas), los archivos ejecutables pueden ser archivos universales que contienen múltiples arquitecturas.

Para saber si el intérprete actual es de 64-bits, es más fiable consultar el atributo `sys.maxsize`:

```
is_64bits = sys.maxsize > 2**32
```

platform.machine()

Retorna el tipo de máquina, por ejemplo `'i386'`. Si no se puede determinar el valor, retorna una cadena vacía.

The output is platform-dependent and may differ in casing and naming conventions.

platform.node()

Retorna el nombre de la red del ordenador (¡tal vez no sea el nombre completo!). Si no se puede determinar el valor, se retorna una cadena vacía.

platform.platform(*aliased=False*, *terse=False*)

Retorna una sola cadena identificando la plataforma subyacente con la mayor información útil posible.

La salida se intenta que sea *humanamente legible* más que tratable por una máquina. Tal vez la salida sea diferente en diversas plataformas y eso mismo es lo que se pretende.

Si *aliased* es verdadero, la función usará aliases para varias plataformas que informen de nombres de sistema que sean diferentes a sus nombres comunes. Por ejemplo, SunOS se reportará como Solaris. La función **`system_alias()`** ha sido usada para implementar esto.

Estableciendo *terse* a verdadero provoca que la función retorne el mínimo de información necesaria para identificar la plataforma.

Distinto en la versión 3.8: En macOS, la función ahora usa **`mac_ver()`**, si retorna una cadena no vacía para obtener la versión de macOS más que la versión de darwin.

platform.processor()

Retorna el nombre (real) del procesador. E.j. `'amdk6'`.

Una cadena vacía se retorna si el valor no se puede determinar. Destacar que muchas plataformas no proveen esta información o simplemente retorna los mismos valores que para **`machine()`**, como hace NetBSD.

platform.python_build()

Retorna una tupla `(buildno, builddate)` con *buildno* indicando el número de la build de Python y *builddate* su fecha de publicación como cadenas.

platform.python_compiler()

Retorna la string con la identificación del compilador usado para compilar Python.

platform.python_branch()

Retorna la string identificando la implementación de la rama SCM de Python.

platform.python_implementation()

Retorna la string identificando la implementación de Python. Algunos valores posibles son: “CPython”, “IronPython”, “Jython”, “PyPy”.

platform.python_revision()

Retorna la string identificando la implementación de la revisión SCM de Python.

platform.python_version()

Retorna la versión de Python en formato de cadena de caracteres con la forma `'major.minor.patchlevel'` siendo *major* la versión principal, *minor* la versión menor y *patchlevel* el último parche aplicado.

Destacar que a diferencia del `sys.version` de Python, el valor retornado siempre incluirá el último parche aplicado (siendo 0 por defecto).

platform.python_version_tuple()

Retorna la versión de Python como una tupla `(major, minor, patchlevel)` de cadena, siendo *major* la versión principal, *minor* la versión menor y *patchlevel* último parche aplicado.

Destacar que a diferencia del `sys.version` de Python, el valor retornado siempre incluirá el último parche aplicado (siendo `'0'` por defecto).

platform.release()

Retorna la versión de publicación del sistema. Por ejemplo `'2.2.0'` o `'NT'`. Si no se puede determinar el valor, retorna una cadena vacía.

platform.system()

Retorna el nombre del sistema/SO, como `'Linux'`, `'Darwin'`, `'Java'`, `'Windows'`. Si no se puede determinar el valor, retorna una cadena vacía.

On iOS and Android, this returns the user-facing OS name (i.e, `'iOS`, `'iPadOS'` or `'Android'`). To obtain the kernel name (`'Darwin'` or `'Linux'`), use **`os.uname()`**.

platform.system_alias(*system*, *release*, *version*)

Retorna la tupla `(system, release, version)` con los alias de los nombres comerciales usados por algunos sistemas siendo *system* el nombre comercial del sistema, *release* como la versión principal de publicación y *version* como el número de la versión del sistema. También hace cierta reordenación de la información en algunos casos donde se produjera algún tipo de confusión.

platform.version()

Retorna la versión de la publicación del sistema. Por ejemplo: `'#3 on degas'`. Una cadena vacía se retorna en el caso de que el valor no pueda ser determinado.

On iOS and Android, this is the user-facing OS version. To obtain the Darwin or Linux kernel version, use **`os.uname()`**.

platform.uname()

Interfaz uname relativamente portable. Retorna una **`namedtuple()`** con seis atributos: **`system`**, **`node`**, **`release`**, **`version`**, **`machine`**, and **`processor`**.

`processor` es resuelto de forma tardía, bajo demanda.

Note: the first two attribute names differ from the names presented by **`os.uname()`**, where they are named **`sysname`** and **`nodename`**.

Cualquier entrada que no pueda ser determinada se establece como `''`.

Distinto en la versión 3.3: El resultado ha cambiado de tupla **`namedtuple()`**..

Distinto en la versión 3.9: **`processor`** es resuelto de forma tardía en lugar de inmediatamente.

platform.invalidate_caches()

Clear out the internal cache of information, such as the

`uname()`. This is typically useful when the platform's **`node()`** is changed by an external process and one needs to retrieve the updated value.

Added in version 3.14.

Java platform

platform.java_ver(*release=", vendor=", vminfo=(", ", "), osinfo=(", ", ")*)

Versión de la interfaz de Jython.

Retorna una tupla `(release, vendor, vminfo, osinfo)` con *vminfo* siendo una tupla `(vm_name, vm_release, vm_vendor)` y *osinfo* siendo una tupla `(os_name, os_version, os_arch)`. Los valores que no se pueden determinar son establecidos por defecto por los parámetros (todos los valores predeterminados son `''`).

Deprecated since version 3.13, will be removed in version 3.15: It was largely untested, had a confusing API, and was only useful for Jython support.

Windows platform

platform.win32_ver(*release=", version=", csd=", ptype="*)

Obtiene información adicional de la versión de registro de windows y retorna una tupla `(release, version, csd, ptype)` la cual se refiere a la versión del sistema operativo, número de su versión , nivel CSD (service pack) y el tipo de sistema operativo (multi/único procesador). Los valores que no se pueden determinar son establecidos por defecto por los parámetros (todos los valores predeterminados son una cadena vacía).

As a hint: *ptype* is `'Uniprocessor Free'` on single processor NT machines and `'Multiprocessor Free'` on multi processor machines. The `'Free'` refers to the OS version being free of debugging code. It could also state `'Checked'` which means the OS version uses debugging code, i.e. code that checks arguments, ranges, etc.

platform.win32_edition()

Retorna una cadena que representa la edición actual de Windows o `None` si el valor no puede ser determinado. Los valores posibles incluyen, entre otros, `'Enterprise'`, `'IoTUAP'`, `'ServerStandard'` y `'nanoserver'`.

Added in version 3.8.

platform.win32_is_iot()

Retorna `True` si la edición de Windows retornada por **`win32_edition()`** se reconoce como una edición IoT.

Added in version 3.8.

macOS platform

platform.mac_ver(*release=''*, *versioninfo=('', '', '')*, *machine=''*)

Obtenga información de la versión de macOS y devuélvala como tupla `(release, versioninfo, machine)` con *versioninfo* como tupla `(version, dev_stage, non_release_version)`.

Cualquier registro que no puede ser determinado se establece como `''`. Todas los registros de la tupla son cadenas.

iOS platform

platform.ios_ver(*system=''*, *release=''*, *model=''*, *is_simulator=False*)

Get iOS version information and return it as a **`namedtuple()`** with the following attributes:

- `system` is the OS name; either `'iOS'` or `'iPadOS'`.
- `release` is the iOS version number as a string (e.g.,

`'17.2'`).

- `model` is the device model identifier; this will be a string like `'iPhone13,2'` for a physical device, or `'iPhone'` on a simulator.
- `is_simulator` is a boolean describing if the app is running on a simulator or a physical device.

Entries which cannot be determined are set to the defaults given as parameters.

Unix platforms

platform.libc_ver(*executable=sys.executable, lib='', version='', chunksize=16384*)

Intenta determinar la versión libc al que está enlazado el fichero ejecutable (por defecto el intérprete de Python). Retorna una tupla de cadenas `(lib, version)` que tiene por defecto los parámetros que han sido introducidos en caso de que la búsqueda fallase.

Destacar que esta función tiene un conocimiento íntimo de cómo las diferentes versiones de libc agregan símbolos al ejecutable. Probablemente sólo se puede utilizar para los ejecutables compilados mediante **gcc**.

El archivo se lee y se analiza en fragmentos de bytes *chunksize*.

Linux platforms

platform.freedesktop_os_release()

Obtiene la identificación del sistema operativo del archivo `os-release` y la retorna como dict. El archivo `os-release` es un freedesktop.org standard [https://www.freedesktop.org/software/systemd/man/os-release.html] y está disponible en la mayoría de las distribuciones de Linux. Una excepción notable son las distribuciones de Android y basadas en Android.

Lanza **`OSError`** o subclase cuando no se pueden leer ni `/etc/os-release` ni `/usr/lib/os-release`.

En caso de éxito, la función retorna un diccionario donde las claves y los valores son cadenas de caracteres. Los valores tienen sus caracteres especiales como `"` y `$` sin comillas. Los campos `NAME`, `ID` y `PRETTY_NAME` siempre se definen de acuerdo con el estándar. Todos los demás campos son opcionales. Los proveedores pueden incluir campos adicionales.

Tenga en cuenta que campos como `NAME`, `VERSION` y `VARIANT` son cadenas de caracteres adecuadas para la presentación a los usuarios. Los programas deben usar campos como `ID`, `ID_LIKE`, `VERSION_ID` o `VARIANT_ID` para identificar distribuciones de Linux.

Ejemplo:

```
def get_like_distro():
    info = platform.freedesktop_os_release()
    ids = [info["ID"]]
    if "ID_LIKE" in info:
        # ids are space separated and ordered by pr
        ids.extend(info["ID_LIKE"].split())
    return ids
```

Added in version 3.10.

Android platform

platform.android_ver(*release=''*, *api_level=0*, *manufacturer=''*, *model=''*, *device=''*, *is_emulator=False*)

Get Android device information. Returns a **`namedtuple()`** with the following attributes. Values which cannot be determined are set to the defaults given as parameters.

- `release` - Android version, as a string (e.g. `"14"`).
- `api_level` - API level of the running device, as an

integer (e.g. `34` for Android 14). To get the API level which Python was built against, see **`sys.getandroidapilevel()`**.

- `manufacturer` - Manufacturer name [https://developer.android.com/reference/android/os/Build#MANUFACTURER].
- `model` - Model name [https://developer.android.com/reference/android/os/Build#MODEL] – typically the marketing name or model number.
- `device` - Device name [https://developer.android.com/reference/android/os/Build#DEVICE] – typically the model number or a codename.
- `is_emulator` - `True` if the device is an emulator; `False` if it's a physical device.

Google maintains a list of known model and device names [https://storage.googleapis.com/play_public/supported_devices.html].

Added in version 3.13.

Command-line usage

`platform` can also be invoked directly using the **`-m`** switch of the interpreter:

```
python -m platform [--terse] [--nonaliased] [{nonaliasec
```

The following options are accepted:

--terse

Print terse information about the platform. This is equivalent to calling **`platform.platform()`** with the *terse* argument set to `True`.

--nonaliased

Print platform information without system/OS name aliasing. This is equivalent to calling **`platform.platform()`** with the *aliased* argument set to `True`.

You can also pass one or more positional arguments (`terse`, `nonaliased`) to explicitly control the output format. These behave similarly to their corresponding options.

`errno` — Standard errno system symbols

Este módulo pone a disposición los símbolos del sistema `errno` estándar. El valor de cada símbolo es el valor entero correspondiente. Los nombres y las descripciones se toman prestados de `linux/include/errno.h`, que debería incluir todo.

errno.errorcode

> Diccionario que proporciona un mapeo del valor de errno al nombre de la cadena en el sistema subyacente. Por ejemplo, `errno.errorcode[errno.EPERM]` se asigna a `'EPERM'`.

Para traducir un código de error numérico en un mensaje de error, use **`os.strerror()`**.

De la siguiente lista, los símbolos que no se utilizan en la plataforma actual no están definidos por el módulo. La lista específica de símbolos definidos está disponible como `errno.errorcode.keys()`. Los símbolos disponibles pueden incluir:

errno.EPERM

> Operación no permitida. Este error se asigna a la excepción **`PermissionError`**.

errno.ENOENT

> El archivo o directorio no existe. Este error se asigna a la excepción **`FileNotFoundError`**.

errno.ESRCH

> No hay tal proceso. Este error se asigna a la excepción **`ProcessLookupError`**.

errno.EINTR

Llamada al sistema interrumpida. Este error se asigna a la excepción `InterruptedError`.

errno.EIO

Error de E/S

errno.ENXIO

No existe tal dispositivo o dirección

errno.E2BIG

Lista de argumentos demasiado larga

errno.ENOEXEC

Error de formato de ejecución

errno.EBADF

Número de archivo incorrecto

errno.ECHILD

No hay procesos secundarios. Este error se asigna a la excepción `ChildProcessError`.

errno.EAGAIN

Intentar otra vez. Este error se asigna a la excepción `BlockingIOError`.

errno.ENOMEM

Sin memoria

errno.EACCES

Permiso denegado. Este error se asigna a la excepción `PermissionError`.

errno.EFAULT

Dirección incorrecta

errno.ENOTBLK

Bloquear dispositivo requerido

errno.EBUSY

Dispositivo o recurso ocupado

errno.EEXIST

El archivo existe. Este error se asigna a la excepción **`FileExistsError`**.

errno.EXDEV

Enlace entre dispositivos

errno.ENODEV

Hay tal dispositivo

errno.ENOTDIR

No es un directorio. Este error se asigna a la excepción **`NotADirectoryError`**.

errno.EISDIR

Es un directorio. Este error se asigna a la excepción **`IsADirectoryError`**.

errno.EINVAL

Argumento inválido

errno.ENFILE

Desbordamiento de la tabla de archivos

errno.EMFILE

Demasiados archivos abiertos

errno.ENOTTY

No es un typewriter

errno.ETXTBSY

Archivo de texto ocupado

errno.EFBIG

Archivo demasiado grande

errno.ENOSPC

No queda espacio en el dispositivo

errno.ESPIPE

Búsqueda ilegal

errno.EROFS

Sistema de archivos de sólo lectura

errno.EMLINK

Demasiados enlaces

errno.EPIPE

Tubería rota. Este error se asigna a la excepción `BrokenPipeError`.

errno.EDOM

Argumento matemático fuera del dominio de función

errno.ERANGE

Resultado matemático no representable

errno.EDEADLK

Podría ocurrir un bloqueo de recursos

errno.ENAMETOOLONG

Nombre de archivo demasiado largo

errno.ENOLCK

No hay bloqueos de registro disponibles

errno.ENOSYS

Función no implementada

errno.ENOTEMPTY

Directorio no vacío

errno.ELOOP

Se han encontrado demasiados enlaces simbólicos

errno.EWOULDBLOCK

La operación se bloquearía. Este error se asigna a la excepción `BlockingIOError`.

errno.ENOMSG

Ningún mensaje del tipo deseado

errno.EIDRM

Identificador eliminado

errno.ECHRNG

Número de canal fuera de rango

errno.EL2NSYNC

Nivel 2 no sincronizado

errno.EL3HLT

Nivel 3 detenido

errno.EL3RST

Nivel 3 restablecido

errno.ELNRNG

Número de enlace fuera de rango

errno.EUNATCH

Controlador de protocolo no adjunto

errno.ENOCSI
No hay estructura CSI disponible

errno.EL2HLT
Nivel 2 detenido

errno.EBADE
Intercambio inválido

errno.EBADR
Descriptor de solicitud inválido

errno.EXFULL
Intercambio completo

errno.ENOANO
Sin ánodo

errno.EBADRQC
Código de solicitud inválido

errno.EBADSLT
Ranura inválida

errno.EDEADLOCK
Error de interbloqueo de bloqueo de archivos

errno.EBFONT
Formato de archivo de fuente incorrecto

errno.ENOSTR
El dispositivo no es una secuencia

errno.ENODATA
Datos no disponibles

errno.ETIME

Temporizador expirado

errno.ENOSR

Recursos fuera de flujos

errno.ENONET

La computadora no está en la red

errno.ENOPKG

Paquete no instalado

errno.EREMOTE

El objeto es remoto

errno.ENOLINK

El enlace ha sido cortado

errno.EADV

Error de publicidad

errno.ESRMNT

Error de Srmount

errno.ECOMM

Error de comunicación al enviar

errno.EPROTO

Error de protocolo

errno.EMULTIHOP

Intento de salto múltiple

errno.EDOTDOT

Error específico de RFS (por su significado en inglés *Remote File System*) errno.EBADMSG

No es un mensaje de datos

errno.EOVERFLOW

Valor demasiado grande para el tipo de datos definido

errno.ENOTUNIQ

Nombre no único en la red

errno.EBADFD

Descriptor de archivo en mal estado

errno.EREMCHG

La dirección remota cambió

errno.ELIBACC

No se puede acceder a una biblioteca compartida necesaria

errno.ELIBBAD

Accediendo a una biblioteca compartida dañada

errno.ELIBSCN

Sección .lib en a.out corrupta

errno.ELIBMAX

Intentando vincular demasiadas bibliotecas compartidas

errno.ELIBEXEC

No se puede ejecutar una biblioteca compartida directamente

errno.EILSEQ

Secuencia de byte ilegal

errno.ERESTART

Llamada al sistema interrumpida debe reiniciarse errno.ESTRPIPE

Error de tubería de flujos

errno.EUSERS

Demasiados usuarios

errno.ENOTSOCK

Operación de socket en no-socket

errno.EDESTADDRREQ

Dirección de destino requerida

errno.EMSGSIZE

Mensaje demasiado largo

errno.EPROTOTYPE

Protocolo de tipo incorrecto para socket

errno.ENOPROTOOPT

Protocolo no disponible

errno.EPROTONOSUPPORT

Protocolo no soportado

errno.ESOCKTNOSUPPORT

Tipo de socket no soportado

errno.EOPNOTSUPP

Operación no soportada en el endpoint de transporte

errno.ENOTSUP

Operación no soportada

Added in version 3.2.

errno.EPFNOSUPPORT

Familia de protocolo no soportada

errno.EAFNOSUPPORT

Familia de direcciones no soportada por protocolo

errno.EADDRINUSE

Dirección ya en uso

errno.EADDRNOTAVAIL

No se puede asignar la dirección solicitada

errno.ENETDOWN

Red caída

errno.ENETUNREACH

Red es inalcanzable

errno.ENETRESET

Conexión de red interrumpida debido al reinicio

errno.ECONNABORTED

El software causó falla de conexión. Este error se asigna a la excepción `ConnectionAbortedError`.

errno.ECONNRESET

Restablecimiento de la conexión por par. Este error se asigna a la excepción `ConnectionResetError`.

errno.ENOBUFS

No hay espacio de búfer disponible

errno.EISCONN

El endpoint de transporte ya está conectado

errno.ENOTCONN

El endpoint final de transporte no está conectado errno.ESHUTDOWN

No se puede enviar después del apagado del endpoint de transporte. Este error se asigna a la excepción **`BrokenPipeError`**.

errno.ETOOMANYREFS

Demasiadas referencias: no se puede empalmar

errno.ETIMEDOUT

Tiempo de conexión agotado. Este error se asigna a la excepción **`TimeoutError`**.

errno.ECONNREFUSED

Conexión denegada. Este error se asigna a la excepción **`ConnectionRefusedError`**.

errno.EHOSTDOWN

Anfitrión caído

errno.EHOSTUNREACH

Sin ruta al anfitrión

errno.EHWPOISON

Memory page has hardware error.

Added in version 3.14.

errno.EALREADY

Operación ya en curso. Este error se asigna a la excepción **`BlockingIOError`**.

errno.EINPROGRESS

Operación ahora en curso. Este error se asigna a la excepción **`BlockingIOError`**.

errno.ESTALE

Manejador de archivos NFS (por su significado en inglés

Network File System) obsoleto

errno.EUCLEAN

La estructura necesita limpieza

errno.ENOTNAM

No es un archivo de tipo con nombre XENIX

errno.ENAVAIL

No hay semáforos XENIX disponibles

errno.EISNAM

Es un archivo de tipo con nombre

errno.EREMOTEIO

Error de E/S remota

errno.EDQUOT

Cuota excedida

errno.EQFULL

La cola de salida de la interfaz está llena

Added in version 3.11.

errno.ENOMEDIUM

No medium found

errno.EMEDIUMTYPE

Wrong medium type

errno.ENOKEY

Required key not available

errno.EKEYEXPIRED

Key has expired

errno.EKEYREVOKED

Key has been revoked

errno.EKEYREJECTED

Key was rejected by service

errno.ERFKILL

Operation not possible due to RF-kill

errno.ELOCKUNMAPPED

Locked lock was unmapped

errno.ENOTACTIVE

Facility is not active

errno.EAUTH

Authentication error

Added in version 3.2.

errno.EBADARCH

Bad CPU type in executable

Added in version 3.2.

errno.EBADEXEC

Bad executable (or shared library)

Added in version 3.2.

errno.EBADMACHO

Malformed Mach-o file

Added in version 3.2.

errno.EDEVERR

Device error

Added in version 3.2.

errno.EFTYPE

Inappropriate file type or format

Added in version 3.2.

errno.ENEEDAUTH

Need authenticator

Added in version 3.2.

errno.ENOATTR

Attribute not found

Added in version 3.2.

errno.ENOPOLICY

Policy not found

Added in version 3.2.

errno.EPROCLIM

Too many processes

Added in version 3.2.

errno.EPROCUNAVAIL

Bad procedure for program

Added in version 3.2.

errno.EPROGMISMATCH

Program version wrong

Added in version 3.2.

errno.EPROGUNAVAIL

RPC prog. not avail

Added in version 3.2.

errno.EPWROFF

Device power is off

Added in version 3.2.

errno.EBADRPC

RPC struct is bad

Added in version 3.2.

errno.ERPCMISMATCH

RPC version wrong

Added in version 3.2.

errno.ESHLIBVERS

Shared library version mismatch

Added in version 3.2.

errno.ENOTCAPABLE

Capacidades insuficientes. Este error se asigna a la excepción **`PermissionError`**.

Availability: WASI, FreeBSD

Added in version 3.11.1.

errno.ECANCELED

Operación cancelada

Added in version 3.2.

errno.EOWNERDEAD

El propietario murió

Added in version 3.2.

errno.ENOTRECOVERABLE

Estado no recuperable

Added in version 3.2.

`ctypes` — A foreign function library for Python

Source code: Lib/ctypes [https://github.com/python/cpython/tree/3.14/Lib/ctypes]

`ctypes` is a foreign function library for Python. It provides C compatible data types, and allows calling functions in DLLs or shared libraries. It can be used to wrap these libraries in pure Python.

This is an optional module. If it is missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

tutorial de ctypes

Nota: Algunos ejemplos de código hacen referencia al tipo ctypes **`c_int`**. En las plataformas donde `sizeof(long) == sizeof(int)` es un alias de **`c_long`**. Por lo tanto, no debe confundirse si **`c_long`** se imprime si espera **`c_int`** — son en realidad del mismo tipo.

Carga de bibliotecas de enlaces dinámicos

`ctypes` exports the **`cdll`**, and on Windows **`windll`** and **`oledll`** objects, for loading dynamic link libraries.

You load libraries by accessing them as attributes of these objects. **`cdll`** loads libraries which export functions using the standard `cdecl` calling convention, while **`windll`** libraries call functions using the `stdcall` calling convention. **`oledll`** also uses the `stdcall` calling convention, and assumes the functions return a

Windows **`HRESULT`** error code. The error code is used to automatically raise an **`OSError`** exception when the function call fails.

Distinto en la versión 3.3: Los errores de Windows solían generar **`WindowsError`**, que ahora es un alias de **`OSError`**.

Here are some examples for Windows. Note that `msvcrt` is the MS standard C library containing most standard C functions, and uses the `cdecl` calling convention:

```
>>> from ctypes import *
>>> print(windll.kernel32)
<WinDLL 'kernel32', handle ... at ...>
>>> print(cdll.msvcrt)
<CDLL 'msvcrt', handle ... at ...>
>>> libc = cdll.msvcrt
>>>
```

Windows agrega automáticamente la extensión común `.dll`.

Nota

Acceder a la biblioteca estándar de C a través de `cdll.msvcrt` utilizará una versión obsoleta de la biblioteca que puede ser incompatible con la utilizada por Python. Cuando sea posible, use la funcionalidad nativa de Python, o bien importe y use el módulo `msvcrt`.

Other systems require the filename *including* the extension to load a library, so attribute access can not be used to load libraries. Either the **`LoadLibrary()`** method of the dll loaders should be used, or you should load the library by creating an instance of **`CDLL`** by calling the constructor.

For example, on Linux:

```
>>> cdll.LoadLibrary("libc.so.6")
<CDLL 'libc.so.6', handle ... at ...>
>>> libc = CDLL("libc.so.6")
>>> libc
<CDLL 'libc.so.6', handle ... at ...>
>>>
```

On macOS:

```
>>> cdll.LoadLibrary("libc.dylib")
<CDLL 'libc.dylib', handle ... at ...>
>>> libc = CDLL("libc.dylib")
>>> libc
<CDLL 'libc.dylib', handle ... at ...>
```

Acceder a las funciones de los dll cargados

Las funciones se acceden como atributos de los objetos dll:

```
>>> libc.printf
<_FuncPtr object at 0x...>
>>> print(windll.kernel32.GetModuleHandleA)
<_FuncPtr object at 0x...>
>>> print(windll.kernel32.MyOwnFunction)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "ctypes.py", line 239, in __getattr__
    func = _StdcallFuncPtr(name, self)
AttributeError: function 'MyOwnFunction' not found
>>>
```

Note that win32 system dlls like `kernel32` and `user32` often export ANSI as well as UNICODE versions of a function. The UNICODE version is exported with a `W` appended to the name, while the ANSI version is exported with an `A` appended to the name. The win32 `GetModuleHandle` function, which returns a *module handle* for a given module name, has the following C prototype, and a macro is used to expose one of them as `GetModuleHandle` depending on whether UNICODE is defined or not:

```
/* ANSI version */
HMODULE GetModuleHandleA(LPCSTR lpModuleName);
```

```
/* UNICODE version */
HMODULE GetModuleHandleW(LPCWSTR lpModuleName);
```

windll no intenta seleccionar una de ellas por arte de magia, se debe acceder a la versión que se necesita especificando `GetModuleHandleA` o `GetModuleHandleW` explícitamente, y luego llamarlo con bytes u objetos de cadena respectivamente.

A veces, las dlls exportan funciones con nombres que no son identificadores válidos de Python, como `"??2@YAPAXI@Z"`. En este caso tienes que usar **`getattr()`** para recuperar la función:

```
>>> getattr(cdll.msvcrt, "??2@YAPAXI@Z")
<_FuncPtr object at 0x...>
>>>
```

En Windows, algunas dlls exportan funciones no por nombre sino por ordinal. Se pueden acceder a estas funciones indexando el objeto dll con el número ordinal:

```
>>> cdll.kernel32[1]
<_FuncPtr object at 0x...>
>>> cdll.kernel32[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "ctypes.py", line 310, in __getitem__
    func = _StdcallFuncPtr(name, self)
AttributeError: function ordinal 0 not found
>>>
```

Funciones de llamada

You can call these functions like any other Python callable. This example uses the `rand()` function, which takes no arguments and returns a pseudo-random integer:

```
>>> print(libc.rand())
1804289383
```

On Windows, you can call the `GetModuleHandleA()` function, which returns a win32 module handle (passing `None` as single argument to call it with a `NULL` pointer):

```
>>> print(hex(windll.kernel32.GetModuleHandleA(None)))
0x1d000000
>>>
```

`ValueError` es lanzado cuando se llama a una función `stdcall` con la convención de llamada `cdecl`, o viceversa:

```
>>> cdll.kernel32.GetModuleHandleA(None)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: Procedure probably called with not enough ar
>>>

>>> windll.msvcrt.printf(b"spam")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: Procedure probably called with too many argu
>>>
```

Para saber la convención de llamada correcta, hay que mirar en el archivo de encabezado C o en la documentación de la función que se quiere llamar.

On Windows, **`ctypes`** uses win32 structured exception handling to prevent crashes from general protection faults when functions are called with invalid argument values:

```
>>> windll.kernel32.GetModuleHandleA(32)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OSError: exception: access violation reading 0x00000020
>>>
```

There are, however, enough ways to crash Python with **`ctypes`**, so you should be careful anyway. The **`faulthandler`** module can be helpful in debugging crashes (e.g. from segmentation faults produced by erroneous C library calls).

`None`, integers, bytes objects and (unicode) strings are the only native Python objects that can directly be used as parameters in these function calls. `None` is passed as a C `NULL` pointer, bytes objects and strings are passed as pointer to the memory block that contains their data (char* or wchar_t*). Python integers are passed as the platform's default C int type, their value is masked to fit into the C type.

Before we move on calling functions with other parameter types, we have to learn more about **`ctypes`** data types.

Tipos de datos fundamentales

`ctypes` defines a number of primitive C compatible data types:

Tipo ctypes	Tipo C	Tipo Python	`_type_`
`c_bool`	_Bool	`bool`	`'?'`
`c_char`	char	1-character `bytes`	`'c'`
`c_wchar`	`wchar_t`	1-character `str`	`'u'`
`c_byte`	char	`int`	`'b'`
`c_ubyte`	unsigned char	`int`	`'B'`
`c_short`	short	`int`	`'h'`
`c_ushort`	unsigned short	`int`	`'H'`
`c_int`	int	`int`	`'i'` *
`c_int8`	`int8_t`	`int`	*
`c_int16`	`int16_t`	`int`	*
`c_int32`	`int32_t`	`int`	*
`c_int64`	`int64_t`	`int`	*
`c_uint`	unsigned int	`int`	`'I'` *
`c_uint8`	`uint8_t`	`int`	*
`c_uint16`	`uint16_t`	`int`	*
`c_uint32`	`uint32_t`	`int`	*
`c_uint64`	`uint64_t`	`int`	*
`c_long`	long	`int`	`'l'`
`c_ulong`	unsigned long	`int`	`'L'`
`c_longlong`	long long	`int`	`'q'` *
`c_ulonglong`	unsigned long long	`int`	`'Q'` *
`c_size_t`	`size_t`	`int`	*
`c_ssize_t`	`Py_ssize_t`	`int`	*
`c_time_t`	`time_t`	`int`	*

`c_float`	float	float
`c_double`	double	float
`c_longdouble`	long double	float
`c_char_p`	char* (terminado en NUL)	bytes object or `None`
`c_wchar_p`	wchar_t* (terminado en NUL)	str or `None`
`c_void_p`	void*	int or `None`
`py_object`	PyObject*	object
VARIANT_BOOL	short int	**`bool`**

Additionally, if IEC 60559 compatible complex arithmetic (Annex G) is supported in both C and `libffi`, the following complex types are available:

Tipo ctypes	Tipo C	Tipo Python
`c_float_complex`	float complex	complex
`c_double_complex`	double complex	complex
`c_longdouble_complex`	long double complex	complex

Todos estos tipos pueden ser creados llamándolos con un inicializador opcional del tipo y valor correctos:

```
>>> c_int()
c_long(0)
>>> c_wchar_p("Hello, World")
c_wchar_p(140018365411392)
>>> c_ushort(-3)
c_ushort(65533)
>>>
```

The constructors for numeric types will convert input using **`__bool__()`**, **`__index__()`** (for `int`), **`__float__()`** or **`__complex__()`**. This means **`c_bool`** accepts any object with a truth value:

```
>>> empty_list = []
>>> c_bool(empty_list)
c_bool(False)
```

Dado que estos tipos son mutables, su valor también puede ser cambiado después:

```
>>> i = c_int(42)
>>> print(i)
c_long(42)
>>> print(i.value)
42
>>> i.value = -99
>>> print(i.value)
-99
>>>
```

Assigning a new value to instances of the pointer types `c_char_p`, `c_wchar_p`, and `c_void_p` changes the *memory location* they point to, *not the contents* of the memory block (of course not, because Python string objects are immutable):

```
>>> s = "Hello, World"
>>> c_s = c_wchar_p(s)
>>> print(c_s)
c_wchar_p(139966785747344)
>>> print(c_s.value)
Hello World
>>> c_s.value = "Hi, there"
>>> print(c_s)              # the memory location has ch
c_wchar_p(139966783348904)
>>> print(c_s.value)
Hi, there
>>> print(s)                # first object is unchanged
Hello, World
>>>
```

Sin embargo, debe tener cuidado de no pasarlos a funciones que esperan punteros a la memoria mutable. Si necesitas bloques de memoria mutables, ctypes tiene una función `create_string_buffer()` que los crea de varias maneras. El contenido actual del bloque de memoria puede ser accedido (o cambiado) con la propiedad `raw`; si quieres acceder a él como cadena terminada NUL, usa la propiedad `value`:

```
>>> from ctypes import *
>>> p = create_string_buffer(3)            # create a 3
>>> print(sizeof(p), repr(p.raw))
3 b'\x00\x00\x00'
>>> p = create_string_buffer(b"Hello")     # create a bu
>>> print(sizeof(p), repr(p.raw))
6 b'Hello\x00'
>>> print(repr(p.value))
b'Hello'
>>> p = create_string_buffer(b"Hello", 10) # create a 1
>>> print(sizeof(p), repr(p.raw))
10 b'Hello\x00\x00\x00\x00\x00'
>>> p.value = b"Hi"
>>> print(sizeof(p), repr(p.raw))
10 b'Hi\x00lo\x00\x00\x00\x00\x00'
>>>
```

The `create_string_buffer()` function replaces the old **`c_buffer()`** function (which is still available as an alias). To create a mutable memory block containing unicode characters of the C type **`wchar_t`**, use the `create_unicode_buffer()` function.

Funciones de llamada, continuación

Note que printf imprime al canal de salida estándar real, *no* a `sys.stdout`, por lo que estos ejemplos sólo funcionarán en el prompt de la consola, no desde dentro de *IDLE* o *PythonWin*:

```
>>> printf = libc.printf
>>> printf(b"Hello, %s\n", b"World!")
Hello, World!
14
>>> printf(b"Hello, %S\n", "World!")
Hello, World!
14
>>> printf(b"%d bottles of beer\n", 42)
42 bottles of beer
19
>>> printf(b"%f bottles of beer\n", 42.5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ctypes.ArgumentError: argument 2: TypeError: Don't know
>>>
```

As has been mentioned before, all Python types except integers, strings, and bytes objects have to be wrapped in their corresponding **ctypes** type, so that they can be converted to the required C data type:

```
>>> printf(b"An int %d, a double %f\n", 1234, c_double(3
An int 1234, a double 3.140000
31
>>>
```

Calling variadic functions

On a lot of platforms calling variadic functions through ctypes is exactly the same as calling functions with a fixed number of parameters. On some platforms, and in particular ARM64 for Apple Platforms, the calling convention for variadic functions is different than that for regular functions.

On those platforms it is required to specify the **argtypes** attribute for the regular, non-variadic, function arguments:

```
libc.printf.argtypes = [ctypes.c_char_p]
```

Because specifying the attribute does not inhibit portability it is advised to always specify **argtypes** for all variadic functions.

Funciones de llamada con sus propios tipos de datos personalizados

You can also customize **ctypes** argument conversion to allow instances of your own classes be used as function arguments. **ctypes** looks for an **_as_parameter_** attribute and uses this as the function argument. The attribute must be an integer, string, bytes, a **ctypes** instance, or an object with an **_as_parameter_** attribute:

```
>>> class Bottles:
...     def __init__(self, number):
...         self._as_parameter_ = number
...
>>> bottles = Bottles(42)
>>> printf(b"%d bottles of beer\n", bottles)
42 bottles of beer
19
>>>
```

If you don’t want to store the instance’s data in the **`_as_parameter_`** instance variable, you could define a **`property`** which makes the attribute available on request.

Especificar los tipos de argumentos requeridos (prototipos de funciones)

It is possible to specify the required argument types of functions exported from DLLs by setting the **`argtypes`** attribute.

`argtypes` must be a sequence of C data types (the **`printf()`** function is probably not a good example here, because it takes a variable number and different types of parameters depending on the format string, on the other hand this is quite handy to experiment with this feature):

```
>>> printf.argtypes = [c_char_p, c_char_p, c_int, c_doub
>>> printf(b"String '%s', Int %d, Double %f\n", b"Hi", 1
String 'Hi', Int 10, Double 2.200000
37
>>>
```

La especificación de un formato protege contra los tipos de argumentos incompatibles (al igual que un prototipo para una función C), e intenta convertir los argumentos en tipos válidos:

```
>>> printf(b"%d %d %d", 1, 2, 3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ctypes.ArgumentError: argument 2: TypeError: 'int' objec
```

```
>>> printf(b"%s %d %f\n", b"X", 2, 3)
X 2 3.000000
13
>>>
```

If you have defined your own classes which you pass to function calls, you have to implement a **`from_param()`** class method for them to be able to use them in the **`argtypes`** sequence. The **`from_param()`** class method receives the Python object passed to the function call, it should do a typecheck or whatever is needed to make sure this object is acceptable, and then return the object itself, its **`_as_parameter_`** attribute, or whatever you want to pass as the C function argument in this case. Again, the result should be an integer, string, bytes, a **`ctypes`** instance, or an object with an **`_as_parameter_`** attribute.

Tipos de retorno

By default functions are assumed to return the C int type. Other return types can be specified by setting the **`restype`** attribute of the function object.

The C prototype of **`time()`** is `time_t time(time_t *)`. Because **`time_t`** might be of a different type than the default return type int, you should specify the **`restype`** attribute:

```
>>> libc.time.restype = c_time_t
```

The argument types can be specified using **`argtypes`**:

```
>>> libc.time.argtypes = (POINTER(c_time_t),)
```

To call the function with a `NULL` pointer as first argument, use `None`:

```
>>> print(libc.time(None))
1150640792
```

Here is a more advanced example, it uses the **`strchr()`** function, which expects a string pointer and a char, and returns a pointer to a string:

```
>>> strchr = libc.strchr
>>> strchr(b"abcdef", ord("d"))
8059983
>>> strchr.restype = c_char_p    # c_char_p is a pointer
>>> strchr(b"abcdef", ord("d"))
b'def'
>>> print(strchr(b"abcdef", ord("x")))
None
>>>
```

If you want to avoid the **`ord("x")`** calls above, you can set the **`argtypes`** attribute, and the second argument will be converted from a single character Python bytes object into a C char:

```
>>> strchr.restype = c_char_p
>>> strchr.argtypes = [c_char_p, c_char]
>>> strchr(b"abcdef", b"d")
b'def'
>>> strchr(b"abcdef", b"def")
Traceback (most recent call last):
ctypes.ArgumentError: argument 2: TypeError: one charact
>>> print(strchr(b"abcdef", b"x"))
None
>>> strchr(b"abcdef", b"d")
b'def'
>>>
```

You can also use a callable Python object (a function or a class for example) as the **`restype`** attribute, if the foreign function returns an integer. The callable will be called with the *integer* the C function returns, and the result of this call will be used as the result of your function call. This is useful to check for error return values and automatically raise an exception:

```
>>> GetModuleHandle = windll.kernel32.GetModuleHandleA
>>> def ValidHandle(value):
...     if value == 0:
...         raise WinError()
...     return value
```

```
...
>>>
>>> GetModuleHandle.restype = ValidHandle
>>> GetModuleHandle(None)
486539264
>>> GetModuleHandle("something silly")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in ValidHandle
OSError: [Errno 126] The specified module could not be f
>>>
```

`WinError` es una función que llamará a la api Windows `FormatMessage` para obtener la representación de la cadena de un código de error, y retornará una excepción. `WinError` toma un parámetro de código de error opcional, si no se usa ninguno, llama a **`GetLastError`()** para recuperarlo.

Please note that a much more powerful error checking mechanism is available through the **`errcheck`** attribute; see the reference manual for details.

Pasar los punteros (o: pasar los parámetros por referencia)

A veces una función api C espera un *puntero* a un tipo de datos como parámetro, probablemente para escribir en el lugar correspondiente, o si los datos son demasiado grandes para ser pasados por valor. Esto también se conoce cómo *pasar parámetros por referencia*.

`ctypes` exports the **`byref()`** function which is used to pass parameters by reference. The same effect can be achieved with the **`pointer()`** function, although **`pointer()`** does a lot more work since it constructs a real pointer object, so it is faster to use **`byref()`** if you don't need the pointer object in Python itself:

```
>>> i = c_int()
>>> f = c_float()
>>> s = create_string_buffer(b'\000' * 32)
>>> print(i.value, f.value, repr(s.value))
0 0.0 b''
>>> libc.sscanf(b"1 3.14 Hello", b"%d %f %s",
...             byref(i), byref(f), s)
3
>>> print(i.value, f.value, repr(s.value))
1 3.1400001049 b'Hello'
>>>
```

Estructuras y uniones

Structures and unions must derive from the `Structure` and `Union` base classes which are defined in the **`ctypes`** module. Each subclass must define a `_fields_` attribute. **`_fields_`** must be a list of *2-tuples*, containing a *field name* and a *field type*.

The field type must be a **`ctypes`** type like `c_int`, or any other derived **`ctypes`** type: structure, union, array, pointer.

Aquí hay un ejemplo simple de una estructura POINT, que contiene dos enteros llamados *x* y *y*, y también muestra cómo inicializar una estructura en el constructor:

```
>>> from ctypes import *
>>> class POINT(Structure):
...     _fields_ = [("x", c_int),
...                 ("y", c_int)]
...
>>> point = POINT(10, 20)
>>> print(point.x, point.y)
10 20
>>> point = POINT(y=5)
>>> print(point.x, point.y)
0 5
>>> POINT(1, 2, 3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: too many initializers
>>>
```

Sin embargo, se pueden construir estructuras mucho más complicadas. Una estructura puede contener por sí misma otras estructuras usando una estructura como tipo de campo.

Aquí hay una estructura RECT que contiene dos POINTs llamados *upperleft* (superior izquierda)y *lowerright* (abajo a la derecha):

```
>>> class RECT(Structure):
...     _fields_ = [("upperleft", POINT),
...                 ("lowerright", POINT)]
...
>>> rc = RECT(point)
>>> print(rc.upperleft.x, rc.upperleft.y)
0 5
>>> print(rc.lowerright.x, rc.lowerright.y)
0 0
>>>
```

Las estructuras anidadas también pueden ser inicializadas en el constructor de varias maneras:

```
>>> r = RECT(POINT(1, 2), POINT(3, 4))
>>> r = RECT((1, 2), (3, 4))
```

Field descriptors can be retrieved from the *class*, they are useful for debugging because they can provide useful information. See **`CField`**:

```
>>> POINT.x
<ctypes.CField 'x' type=c_int, ofs=0, size=4>
>>> POINT.y
<ctypes.CField 'y' type=c_int, ofs=4, size=4>
>>>
```

Advertencia

`ctypes` does not support passing unions or structures with bit-fields to functions by value. While this may work on 32-bit x86, it’s not guaranteed by the library to work in the general case. Unions and structures with bit-fields should always be passed to

functions by pointer.

Structure/union layout, alignment and byte order

By default, Structure and Union fields are laid out in the same way the C compiler does it. It is possible to override this behavior entirely by specifying a `_layout_` class attribute in the subclass definition; see the attribute documentation for details.

It is possible to specify the maximum alignment for the fields and/or for the structure itself by setting the class attributes `_pack_` and/or `_align_`, respectively. See the attribute documentation for details.

ctypes uses the native byte order for Structures and Unions. To build structures with non-native byte order, you can use one of the `BigEndianStructure`, `LittleEndianStructure`, `BigEndianUnion`, and `LittleEndianUnion` base classes. These classes cannot contain pointer fields.

Campos de bits en estructuras y uniones

It is possible to create structures and unions containing bit fields. Bit fields are only possible for integer fields, the bit width is specified as the third item in the `_fields_` tuples:

```
>>> class Int(Structure):
...     _fields_ = [("first_16", c_int, 16),
...                 ("second_16", c_int, 16)]
...
>>> print(Int.first_16)
<ctypes.CField 'first_16' type=c_int, ofs=0, bit_size=16
>>> print(Int.second_16)
<ctypes.CField 'second_16' type=c_int, ofs=0, bit_size=1
```

It is important to note that bit field allocation and layout in memory are not defined as a C standard; their implementation is compiler-specific. By default, Python will attempt to match the behavior of a «native» compiler for the current platform. See the `_layout_` attribute for details on the default behavior and how to

change it.

Arreglos

Los arreglos son secuencias, que contienen un número fijo de instancias del mismo tipo.

La forma recomendada de crear tipos de arreglos es multiplicando un tipo de dato por un entero positivo:

```
TenPointsArrayType = POINT * 10
```

Aquí hay un ejemplo de un tipo de datos algo artificial, una estructura que contiene 4 POINTs entre otras cosas:

```
>>> from ctypes import *
>>> class POINT(Structure):
...     _fields_ = ("x", c_int), ("y", c_int)
...
>>> class MyStruct(Structure):
...     _fields_ = [("a", c_int),
...                 ("b", c_float),
...                 ("point_array", POINT * 4)]
>>>
>>> print(len(MyStruct().point_array))
4
>>>
```

Las instancias se crean de la manera habitual, llamando a la clase:

```
arr = TenPointsArrayType()
for pt in arr:
    print(pt.x, pt.y)
```

El código anterior imprime una serie de líneas `0 0`, porque el contenido del arreglos se inicializa con ceros.

También se pueden especificar inicializadores del tipo correcto:

```
>>> from ctypes import *
>>> TenIntegers = c_int * 10
>>> ii = TenIntegers(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
>>> print(ii)
<c_long_Array_10 object at 0x...>
>>> for i in ii: print(i, end=" ")
...
1 2 3 4 5 6 7 8 9 10
>>>
```

Punteros

Pointer instances are created by calling the `pointer()` function on a **`ctypes`** type:

```
>>> from ctypes import *
>>> i = c_int(42)
>>> pi = pointer(i)
>>>
```

Las instancias del puntero tienen un atributo `contents` que retorna el objeto al que apunta el puntero, el objeto `i` arriba:

```
>>> pi.contents
c_long(42)
>>>
```

Note that **`ctypes`** does not have OOR (original object return), it constructs a new, equivalent object each time you retrieve an attribute:

```
>>> pi.contents is i
False
>>> pi.contents is pi.contents
False
>>>
```

Asignar otra instancia `c_int` al atributo de contenido del puntero causaría que el puntero apunte al lugar de memoria donde se almacena:

```
>>> i = c_int(99)
>>> pi.contents = i
>>> pi.contents
c_long(99)
>>>
```

Las instancias de puntero también pueden ser indexadas con números enteros:

```
>>> pi[0]
99
>>>
```

Asignando a un índice entero cambia el valor señalado:

```
>>> print(i)
c_long(99)
>>> pi[0] = 22
>>> print(i)
c_long(22)
>>>
```

También es posible usar índices diferentes de 0, pero debes saber lo que estás haciendo, al igual que en C: Puedes acceder o cambiar arbitrariamente las ubicaciones de memoria. Generalmente sólo usas esta característica si recibes un puntero de una función C, y *sabes* que el puntero en realidad apunta a un arreglo en lugar de a un solo elemento.

Behind the scenes, the `pointer()` function does more than simply create pointer instances, it has to create pointer *types* first. This is done with the `POINTER()` function, which accepts any **`ctypes`** type, and returns a new type:

```
>>> PI = POINTER(c_int)
>>> PI
<class 'ctypes.LP_c_long'>
>>> PI(42)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: expected c_long instead of int
```

```
>>> PI(c_int(42))
<ctypes.LP_c_long object at 0x...>
>>>
```

Llamar al tipo de puntero sin un argumento crea un puntero `NULL`. Los punteros `NULL` tienen un valor booleano falso..:

```
>>> null_ptr = POINTER(c_int)()
>>> print(bool(null_ptr))
False
>>>
```

`ctypes` checks for `NULL` when dereferencing pointers (but dereferencing invalid non-`NULL` pointers would crash Python):

```
>>> null_ptr[0]
Traceback (most recent call last):
    ....
ValueError: NULL pointer access
>>>

>>> null_ptr[0] = 1234
Traceback (most recent call last):
    ....
ValueError: NULL pointer access
>>>
```

Thread safety without the GIL

From Python 3.13 onward, the GIL can be disabled on the free-threaded build. In ctypes, reads and writes to a single object concurrently is safe, but not across multiple objects:

```
>>> number = c_int(42)
>>> pointer_a = pointer(number)
>>> pointer_b = pointer(number)
```

In the above, it’s only safe for one object to read and write to the address at once if the GIL is disabled. So, `pointer_a` can be shared and written to across multiple threads, but only if

`pointer_b` is not also attempting to do the same. If this is an issue, consider using a **`threading.Lock`** to synchronize access to memory:

```
>>> import threading
>>> lock = threading.Lock()
>>> # Thread 1
>>> with lock:
...     pointer_a.contents = 24
>>> # Thread 2
>>> with lock:
...     pointer_b.contents = 42
```

Conversiones de tipos

Usually, ctypes does strict type checking. This means, if you have `POINTER(c_int)` in the **`argtypes`** list of a function or as the type of a member field in a structure definition, only instances of exactly the same type are accepted. There are some exceptions to this rule, where ctypes accepts other objects. For example, you can pass compatible array instances instead of pointer types. So, for `POINTER(c_int)`, ctypes accepts an array of c_int:

```
>>> class Bar(Structure):
...     _fields_ = [("count", c_int), ("values", POINTER
...
>>> bar = Bar()
>>> bar.values = (c_int * 3)(1, 2, 3)
>>> bar.count = 3
>>> for i in range(bar.count):
...     print(bar.values[i])
...
1
2
3
>>>
```

In addition, if a function argument is explicitly declared to be a pointer type (such as `POINTER(c_int)`) in **`argtypes`**, an object

of the pointed type (`c_int` in this case) can be passed to the function. ctypes will apply the required **`byref()`** conversion in this case automatically.

Para poner un campo de tipo POINTER a `NULL`, puedes asignar `None`:

```
>>> bar.values = None
>>>
```

Sometimes you have instances of incompatible types. In C, you can cast one type into another type. **`ctypes`** provides a **`cast()`** function which can be used in the same way. The `Bar` structure defined above accepts `POINTER(c_int)` pointers or **`c_int`** arrays for its `values` field, but not instances of other types:

```
>>> bar.values = (c_byte * 4)()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: incompatible types, c_byte_Array_4 instance i
>>>
```

Para estos casos, la función **`cast()`** es muy útil.

La función **`cast()`** puede ser usada para lanzar una instancia ctypes en un puntero a un tipo de datos ctypes diferente. **`cast()`** toma dos parámetros, un objeto ctypes que es o puede ser convertido en un puntero de algún tipo, y un tipo de puntero ctypes. retorna una instancia del segundo argumento, que hace referencia al mismo bloque de memoria que el primer argumento:

```
>>> a = (c_byte * 4)()
>>> cast(a, POINTER(c_int))
<ctypes.LP_c_long object at ...>
>>>
```

Así, **`cast()`** puede ser usado para asignar al campo `values` de `Bar` la estructura:

```
>>> bar = Bar()
>>> bar.values = cast((c_byte * 4)(), POINTER(c_int))
>>> print(bar.values[0])
0
>>>
```

Tipos incompletos

Los Tipos Incompletos son estructuras, uniones o matrices cuyos miembros aún no están especificados. En C, se especifican mediante declaraciones a futuro, que se definen más adelante:

```
struct cell; /* forward declaration */

struct cell {
    char *name;
    struct cell *next;
};
```

La traducción directa al código de ctypes sería esta, pero no funciona:

```
>>> class cell(Structure):
...     _fields_ = [("name", c_char_p),
...                 ("next", POINTER(cell))]
...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in cell
NameError: name 'cell' is not defined
>>>
```

because the new `class cell` is not available in the class statement itself. In **`ctypes`**, we can define the `cell` class and set the **`_fields_`** attribute later, after the class statement:

```
>>> from ctypes import *
>>> class cell(Structure):
...     pass
...
>>> cell._fields_ = [("name", c_char_p),
...                  ("next", POINTER(cell))]
```

```
>>>
```

Vamos a intentarlo. Creamos dos instancias de `cell`, y dejamos que se apunten una a la otra, y finalmente seguimos la cadena de punteros unas cuantas veces:

```
>>> c1 = cell()
>>> c1.name = b"foo"
>>> c2 = cell()
>>> c2.name = b"bar"
>>> c1.next = pointer(c2)
>>> c2.next = pointer(c1)
>>> p = c1
>>> for i in range(8):
...     print(p.name, end=" ")
...     p = p.next[0]
...
foo bar foo bar foo bar foo bar
>>>
```

Funciones de retrollamadas (*callback*)

`ctypes` allows creating C callable function pointers from Python callables. These are sometimes called *callback functions*.

Primero, debes crear una clase para la función de retrollamada. La clase conoce la convención de llamada, el tipo de retorno, y el número y tipos de argumentos que esta función recibirá.

La función de fábrica **`CFUNCTYPE`()`** crea tipos para las funciones de retrollamada usando la convención de llamada `cdecl`. En Windows, la función de fábrica **`WINFUNCTYPE()`** crea tipos para funciones de retrollamadas usando la convención de llamadas `stdcall`.

Ambas funciones de fábrica se llaman con el tipo de resultado como primer argumento, y las funciones de llamada de retorno con los tipos de argumentos esperados como los argumentos restantes.

I will present an example here which uses the standard C library’s

`qsort()` function, that is used to sort items with the help of a callback function. **`qsort()`** will be used to sort an array of integers:

```
>>> IntArray5 = c_int * 5
>>> ia = IntArray5(5, 1, 7, 33, 99)
>>> qsort = libc.qsort
>>> qsort.restype = None
>>>
```

`qsort()` must be called with a pointer to the data to sort, the number of items in the data array, the size of one item, and a pointer to the comparison function, the callback. The callback will then be called with two pointers to items, and it must return a negative integer if the first item is smaller than the second, a zero if they are equal, and a positive integer otherwise.

Así que nuestra función de retrollamada recibe punteros a números enteros, y debe retornar un número entero. Primero creamos el `tipo` para la función de retrollamada:

```
>>> CMPFUNC = CFUNCTYPE(c_int, POINTER(c_int), POINTER(c
>>>
```

Para empezar, aquí hay una simple llamada que muestra los valores que se pasan:

```
>>> def py_cmp_func(a, b):
...     print("py_cmp_func", a[0], b[0])
...     return 0
...
>>> cmp_func = CMPFUNC(py_cmp_func)
>>>
```

El resultado:

```
>>> qsort(ia, len(ia), sizeof(c_int), cmp_func)
py_cmp_func 5 1
py_cmp_func 33 99
py_cmp_func 7 33
py_cmp_func 5 7
py_cmp_func 1 7
>>>
```

Ahora podemos comparar los dos artículos y obtener un resultado útil:

```
>>> def py_cmp_func(a, b):
...     print("py_cmp_func", a[0], b[0])
...     return a[0] - b[0]
...
>>>
>>> qsort(ia, len(ia), sizeof(c_int), CMPFUNC(py_cmp_fu
py_cmp_func 5 1
py_cmp_func 33 99
py_cmp_func 7 33
py_cmp_func 1 7
py_cmp_func 5 7
>>>
```

Como podemos comprobar fácilmente, nuestro arreglo está ordenado ahora:

```
>>> for i in ia: print(i, end=" ")
...
1 5 7 33 99
>>>
```

Las funciones de fabrica pueden ser usadas como decoradores de fabrica, así que podemos escribir:

```
>>> @CFUNCTYPE(c_int, POINTER(c_int), POINTER(c_int))
... def py_cmp_func(a, b):
...     print("py_cmp_func", a[0], b[0])
...     return a[0] - b[0]
...
>>> qsort(ia, len(ia), sizeof(c_int), py_cmp_func)
py_cmp_func 5 1
py_cmp_func 33 99
py_cmp_func 7 33
py_cmp_func 1 7
py_cmp_func 5 7
>>>
```

Nota

Make sure you keep references to `CFUNCTYPE()` objects as long as they are used from C code. **`ctypes`** doesn't, and if you don't, they may be garbage collected, crashing your program when a callback is made.

Además, nótese que sí se llama a la función de retrollamada en un hilo creado fuera del control de Python (por ejemplo, por el código foráneo que llama a la retrollamada), ctypes crea un nuevo hilo Python tonto en cada invocación. Este comportamiento es correcto para la mayoría de los propósitos, pero significa que los valores almacenados con `threading.local` *no* sobreviven a través de diferentes llamadas de retorno, incluso cuando esas llamadas se hacen desde el mismo hilo C.

Acceder a los valores exportados de los dlls

Some shared libraries not only export functions, they also export variables. An example in the Python library itself is the `Py_Version`, Python runtime version number encoded in a single constant integer.

`ctypes` can access values like this with the `in_dll()` class methods of the type. *pythonapi* is a predefined symbol giving access to the Python C api:

```
>>> version = ctypes.c_int.in_dll(ctypes.pythonapi, "Py_
>>> print(hex(version.value))
0x30c00a0
```

Un ejemplo extendido que también demuestra el uso de punteros accediendo al puntero `PyImport_FrozenModules` exportado por Python.

Citando los documentos para ese valor:

> Este puntero se inicializa para apuntar a un arreglo de registros **`_frozen`**, terminados por uno cuyos miembros son todos `NULL` o cero. Cuando se importa un módulo congelado, se busca en esta tabla. El código de terceros podría jugar trucos con esto para proporcionar una colección de módulos congelados creada dinámicamente.

So manipulating this pointer could even prove useful. To restrict the example size, we show only how this table can be read with **`ctypes`**:

```
>>> from ctypes import *
>>>
>>> class struct_frozen(Structure):
...     _fields_ = [("name", c_char_p),
...                 ("code", POINTER(c_ubyte)),
...                 ("size", c_int),
...                 ("get_code", POINTER(c_ubyte)),  # F
...                ]
...
>>>
```

Hemos definido el tipo de datos **`_frozen`**, por lo que podemos obtener el puntero a la tabla:

```
>>> FrozenTable = POINTER(struct_frozen)
>>> table = FrozenTable.in_dll(pythonapi, "_PyImport_Fro
>>>
```

Como `tabla` es un `puntero` al arreglo de registros `struct_frozen`, podemos iterar sobre ella, pero sólo tenemos que asegurarnos de que nuestro bucle termine, porque los punteros no tienen tamaño. Tarde o temprano, probablemente se caerá con una violación de acceso o lo que sea, así que es mejor salir del bucle cuando le demos a la entrada `NULL`:

```
>>> for item in table:
...     if item.name is None:
...         break
...     print(item.name.decode("ascii"), item.size)
...
_frozen_importlib 31764
_frozen_importlib_external 41499
zipimport 12345
>>>
```

El hecho de que la Python estándar tenga un módulo congelado y un paquete congelado (indicado por el miembro `tamaño` negativo) no se conoce bien, sólo se usa para hacer pruebas. Pruébalo con `import __hello__` por ejemplo.

Sorpresas

There are some edges in **`ctypes`** where you might expect something other than what actually happens.

Considere el siguiente ejemplo:

```
>>> from ctypes import *
>>> class POINT(Structure):
...     _fields_ = ("x", c_int), ("y", c_int)
...
>>> class RECT(Structure):
...     _fields_ = ("a", POINT), ("b", POINT)
...
>>> p1 = POINT(1, 2)
>>> p2 = POINT(3, 4)
>>> rc = RECT(p1, p2)
>>> print(rc.a.x, rc.a.y, rc.b.x, rc.b.y)
1 2 3 4
>>> # now swap the two points
>>> rc.a, rc.b = rc.b, rc.a
>>> print(rc.a.x, rc.a.y, rc.b.x, rc.b.y)
3 4 3 4
>>>
```

Hm. Ciertamente esperábamos que la última declaración imprimiera `3 4 1 2`. ¿Qué ha pasado? Aquí están los pasos de la línea `rc.a, rc.b = rc.b, rc.a` arriba:

```
>>> temp0, temp1 = rc.b, rc.a
>>> rc.a = temp0
>>> rc.b = temp1
>>>
```

Note que `temp0` y `temp1` son objetos que todavía usan el buffer interno del objeto `rc` de arriba. Así que ejecutando `rc.a = temp0` copia el contenido del buffer de `temp0` en el buffer de `rc`. Esto, a su vez, cambia el contenido de `temp1`. Por lo tanto, la última asignación `rc.b = temp1`, no tiene el efecto esperado.

Tengan en cuenta que la recuperación de subobjetos de Estructuras, Uniones y Arreglos no *copia* el subobjeto, sino que recupera un objeto contenido que accede al búfer subyacente del objeto raíz.

Otro ejemplo que puede comportarse de manera diferente a lo que uno esperaría es este:

```
>>> s = c_char_p()
>>> s.value = b"abc def ghi"
>>> s.value
b'abc def ghi'
>>> s.value is s.value
False
>>>
```

Nota

Los objetos instanciados desde **`c_char_p`** sólo pueden tener su valor fijado en bytes o enteros.

¿Por qué está imprimiendo `False`? Las instancias ctypes son objetos que contienen un bloque de memoria más algunos descriptors que acceden al contenido de la memoria. Almacenar un objeto Python en el bloque de memoria no almacena el objeto en sí mismo, en su lugar se almacenan los `contenidos` del objeto. ¡Acceder a los contenidos de nuevo construye un nuevo objeto Python cada vez!

Tipos de datos de tamaño variable

`ctypes` provides some support for variable-sized arrays and structures.

La función **`resize()`** puede ser usada para redimensionar el buffer de memoria de un objeto ctypes existente. La función toma el objeto como primer argumento, y el tamaño solicitado en bytes como segundo argumento. El bloque de memoria no puede hacerse más pequeño que el bloque de memoria natural especificado por el tipo de objeto, se lanza un **`ValueError`** si se intenta:

```
>>> short_array = (c_short * 4)()
>>> print(sizeof(short_array))
8
>>> resize(short_array, 4)
Traceback (most recent call last):
    ...
ValueError: minimum size is 8
>>> resize(short_array, 32)
>>> sizeof(short_array)
32
>>> sizeof(type(short_array))
8
>>>
```

Esto está bien, pero ¿cómo se puede acceder a los elementos adicionales contenidos en este arreglo? Dado que el tipo todavía sabe sólo 4 elementos, obtenemos errores al acceder a otros elementos:

```
>>> short_array[:]
[0, 0, 0, 0]
>>> short_array[7]
Traceback (most recent call last):
    ...
IndexError: invalid index
>>>
```

Another way to use variable-sized data types with **`ctypes`** is to use the dynamic nature of Python, and (re-)define the data type after the required size is already known, on a case by case basis.

referencia ctypes

Encontrar bibliotecas compartidas

Cuando se programa en un lenguaje compilado, se accede a las bibliotecas compartidas cuando se compila/enlaza un programa, y cuándo se ejecuta el programa.

The purpose of the `find_library()` function is to locate a library in a way similar to what the compiler or runtime loader does (on platforms with several versions of a shared library the most recent should be loaded), while the ctypes library loaders act like when a program is run, and call the runtime loader directly.

The **`ctypes.util`** module provides a function which can help to determine the library to load.

ctypes.util.find_library(*name*)

> Intenta encontrar una biblioteca y retornar un nombre. *name* es el nombre de la biblioteca sin ningún prefijo como *lib*, sufijo como `.so`, `.dylib` o número de versión (esta es la forma usada para la opción del enlazador posix **`-l`**). Si no se puede encontrar ninguna biblioteca, retorna `None`.

La funcionalidad exacta depende del sistema.

On Linux, `find_library()` tries to run external programs (`/sbin/ldconfig`, `gcc`, `objdump` and `ld`) to find the library file. It returns the filename of the library file.

Note that if the output of these programs does not correspond to the dynamic linker used by Python, the result of this function may be misleading.

Distinto en la versión 3.6: En Linux, el valor de la variable de entorno `LD_LIBRARY_PATH` se utiliza cuando se buscan bibliotecas, si una biblioteca no puede ser encontrada por ningún otro medio.

Aquí hay algunos ejemplos:

```
>>> from ctypes.util import find_library
>>> find_library("m")
'libm.so.6'
>>> find_library("c")
'libc.so.6'
>>> find_library("bz2")
'libbz2.so.1.0'
>>>
```

On macOS and Android, **`find_library()`** uses the system's standard naming schemes and paths to locate the library, and returns a full pathname if successful:

```
>>> from ctypes.util import find_library
>>> find_library("c")
'/usr/lib/libc.dylib'
>>> find_library("m")
'/usr/lib/libm.dylib'
>>> find_library("bz2")
'/usr/lib/libbz2.dylib'
>>> find_library("AGL")
'/System/Library/Frameworks/AGL.framework/AGL'
>>>
```

On Windows, **`find_library()`** searches along the system search path, and returns the full pathname, but since there is no predefined naming scheme a call like `find_library("c")` will fail and return `None`.

If wrapping a shared library with **`ctypes`**, it *may* be better to determine the shared library name at development time, and hardcode that into the wrapper module instead of using **`find_library()`** to locate the library at runtime.

Listing loaded shared libraries

When writing code that relies on code loaded from shared libraries, it can be useful to know which shared libraries have already been loaded into the current process.

The **`ctypes.util`** module provides the **`dllist()`** function, which calls the different APIs provided by the various platforms to help determine which shared libraries have already been loaded into the current process.

The exact output of this function will be system dependent. On most platforms, the first entry of this list represents the current process itself, which may be an empty string. For example, on glibc-based Linux, the return may look like:

```
>>> from ctypes.util import dllist
>>> dllist()
['', 'linux-vdso.so.1', '/lib/x86_64-linux-gnu/libm.so.6
```

Cargando bibliotecas compartidas

There are several ways to load shared libraries into the Python process. One way is to instantiate **`CDLL`** or one of its subclasses:

class ctypes.CDLL(*name*, *mode=DEFAULT_MODE*, *handle=None*, *use_errno=False*, *use_last_error=False*, *winmode=None*)

> Represents a loaded shared library.
>
> Functions in this library use the standard C calling convention, and are assumed to return int. The Python global interpreter lock is released before calling any function exported by these libraries, and reacquired afterwards. For different function behavior, use a subclass: **`OleDLL`**, **`WinDLL`**, or **`PyDLL`**.
>
> If you have an existing **`handle`** to an already loaded shared library, it can be passed as the *handle* argument to wrap the opened library in a new **`CDLL`** object. In this case, *name* is only used to set the **`_name`** attribute, but it may be adjusted and/or validated.
>
> If *handle* is `None`, the underlying platform's *dlopen(3)* *[https://manpages.debian.org/dlopen(3)]* or **`LoadLibrary()`** function is

used to load the library into the process, and to get a handle to it.

name is the pathname of the shared library to open. If *name* does not contain a path separator, the library is found in a platform-specific way.

On non-Windows systems, *name* can be `None`. In this case, **`dlopen()`** is called with `NULL`, which opens the main program as a «library». (Some systems do the same is *name* is empty; `None`/`NULL` is more portable.)

CPython implementation detail

Since CPython is linked to `libc`, a `None` *name* is often used to access the C standard library:

```
>>> printf = ctypes.CDLL(None).printf
>>> printf.argtypes = [ctypes.c_char_p]
>>> printf(b"hello\n")
hello
6
```

To access the Python C API, prefer **`ctypes.pythonapi`** which works across platforms.

El parámetro *mode* puede utilizarse para especificar cómo se carga la biblioteca. Para más detalles, consulte la página *dlopen(3)* *[https://manpages.debian.org/dlopen(3)]* del manual. En Windows, *mode* es ignorado. En los sistemas posix, RTLD_NOW siempre se agrega, y no es configurable.

The *use_errno* parameter, when set to true, enables a ctypes mechanism that allows accessing the system **`errno`** error number in a safe way. **`ctypes`** maintains a thread-local copy of the system's **`errno`** variable; if you call foreign functions created with `use_errno=True` then the **`errno`** value before the function call is swapped with the ctypes private copy, the same happens immediately after the function call.

La función **`ctypes.get_errno()`** retorna el valor de la copia privada de ctypes, y la función `ctypes.set_errno()` cambia la copia privada de ctypes a un nuevo valor y retorna el valor anterior.

The *use_last_error* parameter, when set to true, enables the same mechanism for the Windows error code which is managed by the `GetLastError()` and **`SetLastError()`** Windows API functions; `ctypes.get_last_error()` and `ctypes.set_last_error()` are used to request and change the ctypes private copy of the windows error code.

The *winmode* parameter is used on Windows to specify how the library is loaded (since *mode* is ignored). It takes any value that is valid for the Win32 API `LoadLibraryEx` flags parameter. When omitted, the default is to use the flags that result in the most secure DLL load, which avoids issues such as DLL hijacking. Passing the full path to the DLL is the safest way to ensure the correct library and dependencies are loaded.

En Windows, la creación de una instancia `CDLL` puede fallar incluso si existe el nombre de la DLL. Cuando no se encuentra una DLL dependiente de la DLL cargada, se lanza un error `OSError` con el mensaje *«[WinError 126] No se pudo encontrar el módulo especificado»*. Este mensaje de error no contiene el nombre de DLL que falta porque la API de Windows no retorna esta información, lo que dificulta el diagnóstico de este error. Para resolver este error y determinar qué DLL no se encuentra, debe buscar la lista de DLL dependientes y determinar cuál no se encuentra utilizando las herramientas de depuración y seguimiento de Windows.

Ver también

Microsoft DUMPBIN tool [https://learn.microsoft.com/en-us/cpp/build/reference/dumpbin-reference?view=msvc-170] – A tool to find DLL dependents.

Distinto en la versión 3.8: Añadido el parámetro *winmode*.

Distinto en la versión 3.12: The *name* parameter can now be a

path-like object.

Instances of this class have no public methods. Functions exported by the shared library can be accessed as attributes or by index. Please note that accessing the function through an attribute caches the result and therefore accessing it repeatedly returns the same object each time. On the other hand, accessing it through an index returns a new object each time:

```
>>> from ctypes import CDLL
>>> libc = CDLL("libc.so.6")  # On Linux
>>> libc.time == libc.time
True
>>> libc['time'] == libc['time']
False
```

The following public attributes are available. Their name starts with an underscore to not clash with exported function names:

_handle

El manejador del sistema usado para acceder a la biblioteca.

_name

El nombre de la biblioteca pasado en el constructor.

class ctypes.OleDLL

See **`CDLL`**, the superclass, for common information.

Functions in this library use the `stdcall` calling convention, and are assumed to return the windows specific **`HRESULT`** code. **`HRESULT`** values contain information specifying whether the function call failed or succeeded, together with additional error code. If the return value signals a failure, an **`OSError`** is automatically raised.

Availability: Windows

Distinto en la versión 3.3: **`WindowsError`** used to be raised, which is now an alias of **`OSError`**.

class ctypes.WinDLL

See **`CDLL`**, the superclass, for common information.

Functions in these libraries use the `stdcall` calling convention, and are assumed to return int by default.

Availability: Windows

class ctypes.PyDLL

See **`CDLL`**, the superclass, for common information.

When functions in this library are called, the Python GIL is *not* released during the function call, and after the function execution the Python error flag is checked. If the error flag is set, a Python exception is raised.

Thus, this is only useful to call Python C API functions directly.

ctypes.RTLD_GLOBAL

Flag para usar como parámetro *modo*. En las plataformas en las que esta bandera no está disponible, se define como el cero entero.

ctypes.RTLD_LOCAL

Flag para usar como parámetro *modo*. En las plataformas en las que esto no está disponible, es lo mismo que *RTLD_GLOBAL*.

ctypes.DEFAULT_MODE

El modo por defecto que se utiliza para cargar las bibliotecas compartidas. En OSX 10.3, esto es *RTLD_GLOBAL*, de lo contrario es lo mismo que *RTLD_LOCAL*.

Shared libraries can also be loaded by using one of the prefabricated objects, which are instances of the **`LibraryLoader`** class, either by calling the `LoadLibrary()` method, or by retrieving the library as attribute of the loader instance.

class ctypes.LibraryLoader(*dlltype*)

Clase que carga bibliotecas compartidas. *dlltype* debe ser uno de los tipos `CDLL`, `PyDLL`, `WinDLL`, o `OleDLL`.

`__getattr__()` has special behavior: It allows loading a shared library by accessing it as attribute of a library loader instance. The result is cached, so repeated attribute accesses return the same library each time.

LoadLibrary(*name*)

Carga una biblioteca compartida en el proceso y la retorna. Este método siempre retorna una nueva instancia de la biblioteca.

Estos cargadores prefabricados de bibliotecas están disponibles:

ctypes.cdll

Crea instancias de `CDLL`.

ctypes.windll

Creates `WinDLL` instances.

Availability: Windows

ctypes.oledll

Creates `OleDLL` instances.

Availability: Windows

ctypes.pydll

Crea instancias de `PyDLL`.

Para acceder directamente a la API C de Python, se dispone de un objeto de biblioteca compartida de Python listo-para-usar:

ctypes.pythonapi

An instance of **`PyDLL`** that exposes Python C API functions as attributes. Note that all these functions are assumed to return C int, which is of course not always the truth, so you have to assign the correct **`restype`** attribute to use these functions.

Loading a library through any of these objects raises an auditing event `ctypes.dlopen` with string argument `name`, the name used to load the library.

Accessing a function on a loaded library raises an auditing event `ctypes.dlsym` with arguments `library` (the library object) and `name` (the symbol's name as a string or integer).

En los casos en los que sólo está disponible el manejador de la biblioteca en lugar del objeto, al acceder a una función se produce un evento de auditoría `ctypes.dlsym/handle` con los argumentos `handle` (el manejador de la biblioteca en bruto) y `name`.

Funciones foráneas

As explained in the previous section, foreign functions can be accessed as attributes of loaded shared libraries. The function objects created in this way by default accept any number of arguments, accept any ctypes data instances as arguments, and return the default result type specified by the library loader.

They are instances of a private local class **`_FuncPtr`** (not exposed in **`ctypes`**) which inherits from the private **`_CFuncPtr`** class:

```
>>> import ctypes
>>> lib = ctypes.CDLL(None)
>>> issubclass(lib._FuncPtr, ctypes._CFuncPtr)
True
>>> lib._FuncPtr is ctypes._CFuncPtr
False
```

class ctypes._CFuncPtr

Clase base para funciones foráneas C invocables.

Las instancias de funciones foráneas también son tipos de datos compatibles con C; representan punteros de funciones C.

Este comportamiento puede personalizarse asignando a los atributos especiales del objeto de la función foránea.

restype

Asigne un tipo ctypes para especificar el tipo de resultado de la función externa. Use `None` para void, una función que no retorna nada.

It is possible to assign a callable Python object that is not a ctypes type, in this case the function is assumed to return a C int, and the callable will be called with this integer, allowing further processing or error checking. Using this is deprecated, for more flexible post processing or error checking use a ctypes data type as **`restype`** and assign a callable to the **`errcheck`** attribute.

argtypes

Asigne una tupla de tipos ctypes para especificar los tipos de argumentos que acepta la función. Las funciones que utilizan la convención de llamada `stdcall` sólo pueden ser llamadas con el mismo número de argumentos que la longitud de esta tupla; las funciones que utilizan la convención de llamada C aceptan también argumentos adicionales no especificados.

When a foreign function is called, each actual argument is passed to the **`from_param()`** class method of the items in the **`argtypes`** tuple, this method allows adapting the actual argument to an object that the foreign function accepts. For example, a **`c_char_p`** item in the **`argtypes`** tuple will convert a string passed as argument into a bytes object using ctypes conversion rules.

New: It is now possible to put items in argtypes which are not ctypes types, but each item must have a `from_param()` method which returns a value usable as argument (integer, string, ctypes instance). This allows defining adapters that can adapt custom objects as function parameters.

errcheck

Asigne una función Python u otra llamada a este atributo. El invocable será llamado con tres o más argumentos:

callable(*result*, *func*, *arguments*)

result is what the foreign function returns, as specified by the `restype` attribute.

func es el propio objeto de la función foránea, lo que permite reutilizar el mismo objeto invocable para comprobar o postprocesar los resultados de varias funciones.

arguments es una tupla que contiene los parámetros originalmente pasados a la llamada de la función, esto permite especializar el comportamiento en los argumentos utilizados.

El objeto que retorna esta función será retornado por la llamada de la función foránea, pero también puede comprobar el valor del resultado y hacer una excepción si la llamada de la función foránea ha fallado.

En Windows, cuando una llamada a una función foránea plantea una excepción de sistema (por ejemplo, debido a una violación de acceso), será capturada y sustituida por una excepción Python adecuada. Además, un evento de auditoría `ctypes.set_exception` con el argumento `code` será levantado, permitiendo que un gancho de auditoría reemplace la excepción con la suya propia.

Some ways to invoke foreign function calls as well as some of the functions in this module may raise an auditing event `ctypes.call_function` with arguments `function pointer` and `arguments`.

Prototipos de funciones

Las funciones foráneas también pueden crearse mediante la instanciación de prototipos de funciones. Los prototipos de funciones son similares a los prototipos de funciones en C; describen una función (tipo de retorno, tipos de argumentos, convención de llamada) sin definir una implementación. Las funciones de fábrica deben ser llamadas con el tipo de resultado deseado y los tipos de argumento de la función, y pueden ser usadas como fábricas de decoradores, y como tales, ser aplicadas a las funciones a través de la sintaxis `@wrapper`. Ver Funciones de retrollamadas (callback) para ejemplos.

ctypes.CFUNCTYPE(*restype*, **argtypes*, *use_errno=False*, *use_last_error=False*)

> El prototipo de función retornado crea funciones que usan la convención de llamada C estándar. La función liberará el GIL durante la llamada. Si *use_errno* se configura a true, la copia privada de ctypes de la variable del sistema **`errno`** se intercambia con el valor real **`errno`** antes y después de la llamada; *use_last_error* hace lo mismo con el código de error de Windows.

ctypes.WINFUNCTYPE(*restype*, **argtypes*, *use_errno=False*, *use_last_error=False*)

> The returned function prototype creates functions that use the `stdcall` calling convention. The function will release the GIL during the call. *use_errno* and *use_last_error* have the same meaning as above.
>
> Availability: Windows

ctypes.PYFUNCTYPE(*restype*, **argtypes*)

> El prototipo de función retornado crea funciones que usan la

convención de llamadas de Python. La función *no* liberará el GIL durante la llamada.

Los prototipos de funciones creados por estas funciones de fábrica pueden ser instanciados de diferentes maneras, dependiendo del tipo y el número de los parámetros en la llamada:

prototype(*address*)

Retorna una función foránea en la dirección especificada que debe ser un número entero.

prototype(*callable*)

Crear una función de llamada C (una función de retrollamada) a partir de un *callable* Python.

prototype(*func_spec*[, *paramflags*])

Retorna una función foránea exportada por una biblioteca compartida. *func_spec* debe ser un 2-tupla `(name_or_ordinal, library)`. El primer elemento es el nombre de la función exportada como cadena, o el ordinal de la función exportada como entero pequeño. El segundo elemento es la instancia de la biblioteca compartida.

prototype(*vtbl_index*, *name*[, *paramflags*[, *iid*]])

Retorna una función foránea que llamará a un método COM. *vtbl_index* es el índice de la tabla de funciones virtuales, un pequeño entero no negativo. *name* es el nombre del método COM. *iid* es un puntero opcional para el identificador de la interfaz que se utiliza en el informe de errores extendido.

If *iid* is not specified, an **`OSError`** is raised if the COM method call fails. If *iid* is specified, a **`COMError`** is raised instead.

COM methods use a special calling convention: They require a pointer to the COM interface as first argument, in addition to those parameters that are specified in the **`argtypes`** tuple.

Availability: Windows

El parámetro opcional *paramflags* crea envoltorios de funciones foráneas con mucha más funcionalidad que las características descritas anteriormente.

paramflags must be a tuple of the same length as **`argtypes`**.

Cada elemento de esta tupla contiene más información sobre un parámetro, debe ser una tupla que contenga uno, dos o tres elementos.

El primer elemento es un entero que contiene una combinación de flags de dirección para el parámetro:

1
: Especifica un parámetro de entrada a la función.

2
: Parámetro de salida. La función foránea rellena un valor.

4
: Parámetro de entrada que por defecto es el cero entero.

El segundo elemento opcional es el nombre del parámetro como cadena. Si se especifica esto, se puede llamar a la función foránea con parámetros con nombre.

El tercer elemento opcional es el valor por defecto de este parámetro.

The following example demonstrates how to wrap the Windows `MessageBoxW` function so that it supports default parameters and named arguments. The C declaration from the windows header file is this:

```
WINUSERAPI int WINAPI
MessageBoxW(
    HWND hWnd,
    LPCWSTR lpText,
    LPCWSTR lpCaption,
    UINT uType);
```

Here is the wrapping with **`ctypes`**:

```
>>> from ctypes import c_int, WINFUNCTYPE, windll
>>> from ctypes.wintypes import HWND, LPCWSTR, UINT
>>> prototype = WINFUNCTYPE(c_int, HWND, LPCWSTR, LPCWST
>>> paramflags = (1, "hwnd", 0), (1, "text", "Hi"), (1,
>>> MessageBox = prototype(("MessageBoxW", windll.user32
```

La función foránea de `MessageBox` puede ser llamada de esta manera:

```
>>> MessageBox()
>>> MessageBox(text="Spam, spam, spam")
>>> MessageBox(flags=2, text="foo bar")
```

Un segundo ejemplo demuestra los parámetros de salida. La función `GetWindowRect` de win32 retorna las dimensiones de una ventana especificada copiándolas en la estructura `RECT` que la persona que llama tiene que suministrar. Aquí está la declaración C:

```
WINUSERAPI BOOL WINAPI
GetWindowRect(
     HWND hWnd,
     LPRECT lpRect);
```

Here is the wrapping with **`ctypes`**:

```
>>> from ctypes import POINTER, WINFUNCTYPE, windll, Wir
>>> from ctypes.wintypes import BOOL, HWND, RECT
>>> prototype = WINFUNCTYPE(BOOL, HWND, POINTER(RECT))
>>> paramflags = (1, "hwnd"), (2, "lprect")
>>> GetWindowRect = prototype(("GetWindowRect", windll.u
>>>
```

Las funciones con parámetros de salida retornarán automáticamente el valor del parámetro de salida si hay uno solo, o una tupla que contiene los valores del parámetro de salida cuando hay más de uno, por lo que la función GetWindowRect retorna ahora una instancia RECT, cuando se llama.

Output parameters can be combined with the **`errcheck`** protocol to do further output processing and error checking. The win32 `GetWindowRect` api function returns a `BOOL` to signal success or failure, so this function could do the error checking, and raises an exception when the api call failed:

```
>>> def errcheck(result, func, args):
...     if not result:
...         raise WinError()
...     return args
...
>>> GetWindowRect.errcheck = errcheck
>>>
```

If the **`errcheck`** function returns the argument tuple it receives unchanged, **`ctypes`** continues the normal processing it does on the output parameters. If you want to return a tuple of window coordinates instead of a `RECT` instance, you can retrieve the fields in the function and return them instead, the normal processing will no longer take place:

```
>>> def errcheck(result, func, args):
...     if not result:
...         raise WinError()
...     rc = args[1]
...     return rc.left, rc.top, rc.bottom, rc.right
...
>>> GetWindowRect.errcheck = errcheck
>>>
```

Funciones de utilidad

ctypes.addressof(*obj*)

Retorna la dirección del buffer de memoria como un entero. *obj* debe ser una instancia de tipo ctypes.

Lanza un auditing event `ctypes.addressof` con el argumento `obj`.

ctypes.alignment(*obj_or_type*)

Retorna los requerimientos de alineación de un tipo de ctypes. *obj_or_type* debe ser un tipo o instancia ctypes.

ctypes.byref(*obj*[, *offset*])

Retorna un puntero ligero a *obj*, que debe ser un ejemplo de un tipo de ctypes. *offset* es por defecto cero, y debe ser un entero que se añadirá al valor del puntero interno.

`byref(obj, offset)` corresponde a este código C:

```
(((char *)&obj) + offset)
```

El objeto retornado sólo puede ser utilizado como un parámetro de llamada de función foránea. Se comporta de manera similar a `pointer(obj)`, pero la construcción es mucho más rápida.

ctypes.CopyComPointer(*src*, *dst*)

Copies a COM pointer from *src* to *dst* and returns the Windows specific **`HRESULT`** value.

If *src* is not `NULL`, its `AddRef` method is called, incrementing the reference count.

In contrast, the reference count of *dst* will not be decremented before assigning the new value. Unless *dst* is `NULL`, the caller is responsible for decrementing the reference count by calling its `Release` method when necessary.

Availability: Windows

Added in version 3.14.

ctypes.cast(*obj*, *type*)

Esta función es similar a la del operador de reparto en C.

retorna una nueva instancia de *type* que apunta al mismo bloque de memoria que *obj*. *type* debe ser un tipo de puntero, y *obj* debe ser un objeto que pueda ser interpretado como un puntero.

ctypes.create_string_buffer(*init*, *size=None*)
ctypes.create_string_buffer(*size*)

Esta función crea un búfer de caracteres mutables. El objeto retornado es un arreglo de ctypes de **`c_char`**.

If *size* is given (and not `None`), it must be an **`int`**. It specifies the size of the returned array.

If the *init* argument is given, it must be **`bytes`**. It is used to initialize the array items. Bytes not initialized this way are set to zero (NUL).

If *size* is not given (or if it is `None`), the buffer is made one element larger than *init*, effectively adding a NUL terminator.

If both arguments are given, *size* must not be less than `len(init)`.

Advertencia

If *size* is equal to `len(init)`, a NUL terminator is not added. Do not treat such a buffer as a C string.

For example:

```
>>> bytes(create_string_buffer(2))
b'\x00\x00'
>>> bytes(create_string_buffer(b'ab'))
b'ab\x00'
>>> bytes(create_string_buffer(b'ab', 2))
b'ab'
>>> bytes(create_string_buffer(b'ab', 4))
b'ab\x00\x00'
>>> bytes(create_string_buffer(b'abcdef', 2))
Traceback (most recent call last):
   ...
ValueError: byte string too long
```

Lanza un auditing event `ctypes.create_string_buffer` con argumentos `init`, `size`.

ctypes.create_unicode_buffer(*init*, *size=None*)
ctypes.create_unicode_buffer(*size*)

Esta función crea un búfer de caracteres unicode mutable. El objeto retornado es un arreglo de ctypes de **`c_wchar`**.

The function takes the same arguments as **`create_string_buffer()`** except *init* must be a string and *size* counts **`c_wchar`**.

Lanza un auditing event `ctypes.create_unicode_buffer` con argumentos `init`, `size`.

ctypes.DllCanUnloadNow()

This function is a hook which allows implementing in-process COM servers with ctypes. It is called from the DllCanUnloadNow function that the _ctypes extension dll exports.

Availability: Windows

ctypes.DllGetClassObject()

This function is a hook which allows implementing in-process COM servers with ctypes. It is called from the DllGetClassObject function that the `_ctypes` extension dll exports.

Availability: Windows

ctypes.util.find_library(*name*)

Intenta encontrar una biblioteca y retornar un nombre de ruta. *name* es el nombre de la biblioteca sin ningún prefijo como `lib`, sufijo como `.so`, `.dylib` o número de versión (esta es la forma usada para la opción del enlazador posix `-l`). Si no se puede encontrar ninguna biblioteca, retorna `None`.

La funcionalidad exacta depende del sistema.

See Encontrar bibliotecas compartidas for complete documentation.

ctypes.util.find_msvcrt()

Returns the filename of the VC runtime library used by Python, and by the extension modules. If the name of the library cannot be determined, `None` is returned.

Si necesita liberar memoria, por ejemplo, asignada por un módulo de extensión con una llamada al `free(void *)`, es importante que utilice la función en la misma biblioteca que asignó la memoria.

Availability: Windows

ctypes.util.dllist()

Try to provide a list of paths of the shared libraries loaded into the current process. These paths are not normalized or processed in any way. The function can raise **`OSError`** if the underlying platform APIs fail. The exact functionality is system dependent.

On most platforms, the first element of the list represents the current executable file. It may be an empty string.

Availability: Windows, macOS, iOS, glibc, BSD libc, musl

Added in version 3.14.

ctypes.FormatError([*code*])

Returns a textual description of the error code *code*. If no error code is specified, the last error code is used by calling the Windows API function **`GetLastError()`**.

Availability: Windows

ctypes.GetLastError()

Returns the last error code set by Windows in the calling thread. This function calls the Windows `GetLastError()` function directly, it does not return the ctypes-private copy of the error code.

Availability: Windows

ctypes.get_errno()

Retorna el valor actual de la copia ctypes-private de la variable de sistema **`errno`** en el hilo de llamada.

Lanza un auditing event `ctypes.get_errno` sin argumentos.

ctypes.get_last_error()

Returns the current value of the ctypes-private copy of the system **`LastError`** variable in the calling thread.

Availability: Windows

Lanza un auditing event `ctypes.get_last_error` sin argumentos.

ctypes.memmove(*dst*, *src*, *count*)

Igual que la función de la biblioteca estándar de C *memmove*: copia *count* bytes de *src* a *dst*. *dst* y *src* deben ser enteros o instancias ctypes que pueden ser convertidos en punteros.

ctypes.memset(*dst*, *c*, *count*)

Igual que la función de la biblioteca estándar de C *memset* C: llena el bloque de memoria en la dirección *dst* con *count* bytes de valor *c*. *dst* debe ser un número entero que especifique una dirección, o una instancia ctypes.

ctypes.POINTER(*type*, */*)

Create or return a ctypes pointer type. Pointer types are cached and reused internally, so calling this function repeatedly is cheap. *type* must be a ctypes type.

Detalles de implementación de CPython: The resulting pointer type is cached in the `__pointer_type__` attribute of *type*. It is possible to set this attribute before the first call to `POINTER` in order to set a custom pointer type. However, doing this is discouraged: manually creating a suitable pointer type is difficult without relying on implementation details that may change in future Python versions.

ctypes.pointer(*obj*, */*)

Create a new pointer instance, pointing to *obj*. The returned object is of the type `POINTER(type(obj))`.

Nota: Si sólo quieres pasar un puntero a un objeto a una llamada de función foránea, deberías usar `byref(obj)` que es mucho más rápido.

ctypes.resize(*obj*, *size*)

Esta función redimensiona el búfer de memoria interna de *obj*, que debe ser una instancia de tipo ctypes. No es posible hacer el buffer más pequeño que el tamaño nativo del tipo de objetos, como lo indica `size of (type(obj))`, pero es posible agrandar el buffer.

ctypes.set_errno(*value*)

Poner el valor actual de la copia ctypes-private de la variable del sistema **`errno`** en el hilo de llamada a *valor* y retornar el valor anterior.

Lanza un auditing event `ctypes.set_errno` con argumento `errno`.

ctypes.set_last_error(*value*)

Sets the current value of the ctypes-private copy of the system

`LastError` variable in the calling thread to *value* and return the previous value.

Availability: Windows

Lanza un auditing event `ctypes.set_last_error` con argumento `error`.

ctypes.sizeof(*obj_or_type*)

Retorna el tamaño en bytes de un buffer de memoria tipo ctypes o instancia. Hace lo mismo que el operador C `sizeof`.

ctypes.string_at(*ptr*, *size=-1*)

Return the byte string at *void *ptr*. If *size* is specified, it is used as size, otherwise the string is assumed to be zero-terminated.

Raises an auditing event `ctypes.string_at` with arguments `ptr`, `size`.

ctypes.WinError(*code=None*, *descr=None*)

Creates an instance of **`OSError`**. If *code* is not specified, **`GetLastError()`** is called to determine the error code. If *descr* is not specified, **`FormatError()`** is called to get a textual description of the error.

Availability: Windows

Distinto en la versión 3.3: An instance of **`WindowsError`** used to be created, which is now an alias of **`OSError`**.

ctypes.wstring_at(*ptr*, *size=-1*)

Return the wide-character string at *void *ptr*. If *size* is specified, it is used as the number of characters of the string, otherwise the string is assumed to be zero-terminated.

Raises an auditing event `ctypes.wstring_at` with arguments `ptr`, `size`.

ctypes.memoryview_at(*ptr*, *size*, *readonly=False*)

Return a **`memoryview`** object of length *size* that references memory starting at *void *ptr*.

If *readonly* is true, the returned **`memoryview`** object can not be used to modify the underlying memory. (Changes made by other means will still be reflected in the returned object.)

This function is similar to **`string_at()`** with the key difference of not making a copy of the specified memory. It is a semantically equivalent (but more efficient) alternative to `memoryview((c_byte * size).from_address(ptr))`. (While **`from_address()`** only takes integers, *ptr* can also be given as a **`ctypes.POINTER`** or a **`byref()`** object.)

Raises an auditing event `ctypes.memoryview_at` with arguments `address`, `size`, `readonly`.

Added in version 3.14.

Tipos de datos

class ctypes._CData

Esta clase no pública es la clase de base común de todos los tipos de datos de los ctypes. Entre otras cosas, todas las instancias de tipo ctypes contienen un bloque de memoria que contiene datos compatibles con C; la dirección del bloque de memoria es retornada por la función de ayuda **`addressof()`**. Otra variable de instancia se expone como **`_objetos`**; ésta contiene otros objetos de Python que deben mantenerse vivos en caso de que el bloque de memoria contenga punteros.

Métodos comunes de tipos de datos ctypes, estos son todos métodos de clase (para ser exactos, son métodos del metaclass):

from_buffer(*source*[, *offset*])

Este método retorna una instancia ctypes que comparte el buffer del objeto *source*. El objeto *source* debe soportar la interfaz del buffer de escritura. El parámetro opcional *offset* especifica un offset en el buffer de la fuente en bytes; el valor por defecto es cero. Si el buffer de la fuente no es lo suficientemente grande se lanza un **`ValueError`**.

Lanza un auditing event `ctypes.cdata/buffer` con argumentos `pointer`, `size`, `offset`.

from_buffer_copy(*source*[, *offset*])

Este método crea una instancia ctypes, copiando el buffer del buffer de objetos *source* que debe ser legible. El parámetro opcional *offset* especifica un offset en el buffer de origen en bytes; el valor por defecto es cero. Si el buffer de fuente no es lo suficientemente grande se lanza un **`ValueError`**.

Lanza un auditing event `ctypes.cdata/buffer` con argumentos `pointer`, `size`, `offset`.

from_address(*address*)

Este método retorna una instancia de tipo ctypes utilizando la memoria especificada por *address* que debe ser un entero.

Este método, y otros que indirectamente llaman a este método, lanzan un auditing event `ctypes.cdata` con argumento `address`.

from_param(*obj*)

This method adapts *obj* to a ctypes type. It is called with the actual object used in a foreign function call when the type is present in the foreign function's **`argtypes`** tuple; it must return an object that can be used as a function call parameter.

Todos los tipos de datos ctypes tienen una implementación por defecto de este método de clase que normalmente retorna *obj* si es una instancia del tipo. Algunos tipos aceptan también otros objetos.

in_dll(*library*, *name*)

Este método retorna una instancia de tipo ctypes exportada por una biblioteca compartida. *name* es el nombre del símbolo que exporta los datos, *library* es la biblioteca compartida cargada.

Common class variables of ctypes data types:

_pointer_type_

The pointer type that was created by calling **`POINTER()`** for corresponding ctypes data type. If a pointer type was not yet created, the attribute is missing.

Added in version 3.14.

Variables de instancia común de los tipos de datos de ctypes:

_b_base_

A veces, las instancias de datos ctypes no poseen el bloque de memoria que contienen, sino que comparten parte del bloque de memoria de un objeto base. El miembro de sólo lectura **`_b_base_`** es el objeto raíz ctypes que posee el bloque de memoria.

_b_needsfree_

Esta variable de sólo lectura es verdadera cuando la instancia de datos ctypes ha sido asignada a el propio bloque de memoria, falsa en caso contrario.

_objects

Este miembro es `None` o un diccionario que contiene objetos de Python que deben mantenerse vivos para que el contenido del bloque de memoria sea válido. Este objeto sólo se expone para su depuración; nunca modifique el contenido de este diccionario.

Tipos de datos fundamentales

class ctypes._SimpleCData

Esta clase no pública es la clase base de todos los tipos de datos de ctypes fundamentales. Se menciona aquí porque contiene los atributos comunes de los tipos de datos de ctypes fundamentales. **`_SimpleCData`** es una subclase de **`_CData`**, por lo que hereda sus métodos y atributos. Los tipos de datos ctypes que no son y no contienen punteros ahora pueden ser archivados.

Los instancias tienen un solo atributo:

value

Este atributo contiene el valor real de la instancia. Para los tipos enteros y punteros, es un entero, para los tipos de caracteres, es un objeto o cadena de bytes de un solo carácter, para los tipos de punteros de caracteres es un objeto o cadena de bytes de Python.

When the `value` attribute is retrieved from a ctypes instance, usually a new object is returned each time. **`ctypes`** does *not* implement original object return, always a new object is constructed. The same is true for all other ctypes object instances.

Each subclass has a class attribute:

type

Class attribute that contains an internal type code, as a single-character string. See Tipos de datos fundamentales for a summary.

Types marked * in the summary may be (or always are) aliases of a different **`_SimpleCData`** subclass, and will not necessarily use the listed type code. For example, if the platform’s long, long long and time_t C types are the same, then **`c_long`**, **`c_longlong`** and **`c_time_t`** all refer to a single class, `c_long`, whose `_type_` code is `'l'`. The `'L'` code will be unused.

Ver también

The `array` and struct modules, as well as third-party modules like numpy [https://numpy.org/doc/stable/reference/arrays.interface.html#object.__array_interface__], use similar – but slightly different – type codes.

Fundamental data types, when returned as foreign function call results, or, for example, by retrieving structure field members or array items, are transparently converted to native Python types. In other words, if a foreign function has a `restype` of `c_char_p`, you will always receive a Python bytes object, *not* a `c_char_p` instance.

Subclasses of fundamental data types do *not* inherit this behavior. So, if a foreign functions `restype` is a subclass of `c_void_p`, you will receive an instance of this subclass from the function call. Of course, you can get the value of the pointer by accessing the `value` attribute.

Estos son los tipos de datos fundamentales de ctypes:

class ctypes.c_byte

Representa el tipo de datos C signed char e interpreta el valor como un entero pequeño. El constructor acepta un inicializador entero opcional; no se realiza ninguna comprobación de desbordamiento.

class ctypes.c_char

Representa el tipo de datos C char e interpreta el valor como un solo carácter. El constructor acepta un inicializador de cadena opcional, la longitud de la cadena debe ser exactamente un carácter.

class ctypes.c_char_p

Representa el tipo de datos C char* cuando apunta a una cadena terminada en cero. Para un puntero de carácter general que también puede apuntar a datos binarios, se debe usar `POINTER(c_char)`. El constructor acepta una dirección entera o un objeto de bytes.

class ctypes.c_double

Representa el tipo de datos C double. El constructor acepta un inicializador flotante opcional.

class ctypes.c_longdouble

Representa el tipo de datos C long double. El constructor acepta un inicializador flotante opcional. En plataformas donde `sizeof(long double) == sizeof(double)` es un alias de **`c_double`**.

class ctypes.c_float

Representa el tipo de datos C float. El constructor acepta un inicializador flotante opcional.

class ctypes.c_double_complex

Represents the C double complex datatype, if available. The constructor accepts an optional **`complex`** initializer.

Added in version 3.14.

class ctypes.c_float_complex

Represents the C float complex datatype, if available. The constructor accepts an optional **`complex`** initializer.

Added in version 3.14.

class ctypes.c_longdouble_complex

Represents the C long double complex datatype, if available. The constructor accepts an optional **`complex`** initializer.

Added in version 3.14.

class ctypes.c_int

Representa el tipo de datos C signed int. El constructor acepta un inicializador entero opcional; no se realiza ninguna comprobación de desbordamiento. En plataformas donde `sizeof(int) == sizeof(long)` es un alias de **`c_long`**.

class ctypes.c_int8

Represents the C 8-bit signed int datatype. It is an alias for **`c_byte`**.

class ctypes.c_int16

Representa el tipo de datos signed int de C de 16 bits. Por lo general, un alias para **`c_short`**.

class ctypes.c_int32

Representa el tipo de datos signed int de C de 32 bits. Por lo general, un alias para **`c_int`**.

class ctypes.c_int64

Representa el tipo de datos signed int de C de 64 bits. Por lo general, un alias para **`c_longlong`**.

class ctypes.c_long

Representa el tipo de datos C signed long. El constructor acepta un inicializador entero opcional; no se realiza ninguna comprobación de desbordamiento.

class ctypes.c_longlong

Represents the C signed long long datatype. The constructor accepts an optional integer initializer; no overflow checking is done. On platforms where `sizeof(long long) == sizeof(long)` it is an alias to **`c_long`**.

class ctypes.c_short

Representa el tipo de datos C signed short. El constructor acepta un inicializador entero opcional; no se realiza ninguna comprobación de desbordamiento.

class ctypes.c_size_t

Represents the C **`size_t`** datatype. Usually an alias for another unsigned integer type.

class ctypes.c_ssize_t

Represents the **`Py_ssize_t`** datatype. This is a signed version of **`size_t`**; that is, the POSIX **`ssize_t`** type. Usually an alias for another integer type.

Added in version 3.2.

class ctypes.c_time_t

Represents the C **`time_t`** datatype. Usually an alias for another integer type.

Added in version 3.12.

class ctypes.c_ubyte

Representa el tipo de datos C unsigned char, interpreta el valor como un entero pequeño. El constructor acepta un inicializador entero opcional; no se realiza ninguna comprobación de desbordamiento.

class ctypes.c_uint

Representa el tipo de datos C unsigned int. El constructor acepta un inicializador entero opcional; no se realiza ninguna comprobación de desbordamiento. En plataformas donde `sizeof(int) == sizeof(long)` es un alias para **`c_ulong`**.

class ctypes.c_uint8

Represents the C 8-bit unsigned int datatype. It is an alias for **`c_ubyte`**.

class ctypes.c_uint16

Representa el tipo de datos unsigned int de C de 16 bits. Por lo general, un alias para **`c_ushort`**.

class ctypes.c_uint32

Representa el tipo de datos unsigned int de C de 32 bits. Por lo general, un alias para `c_uint`.

class ctypes.c_uint64

Representa el tipo de datos unsigned int de C de 64 bits. Por lo general, un alias para `c_ulonglong`.

class ctypes.c_ulong

Representa el tipo de datos C unsigned long. El constructor acepta un inicializador entero opcional; no se realiza ninguna comprobación de desbordamiento.

class ctypes.c_ulonglong

Represents the C unsigned long long datatype. The constructor accepts an optional integer initializer; no overflow checking is done. On platforms where `sizeof(long long) == sizeof(long)` it is an alias to `c_long`.

class ctypes.c_ushort

Representa el tipo de datos C unsigned short. El constructor acepta un inicializador entero opcional; no se realiza ninguna comprobación de desbordamiento.

class ctypes.c_void_p

Representa el tipo C void*. El valor se representa como un número entero. El constructor acepta un inicializador entero opcional.

class ctypes.c_wchar

Represents the C **`wchar_t`** datatype, and interprets the value as a single character unicode string. The constructor accepts an optional string initializer, the length of the string must be exactly one character.

class ctypes.c_wchar_p

Representa el tipo de datos C wchar_t*, que debe ser un puntero a una cadena de caracteres anchos terminada en cero. El constructor acepta una dirección entera o una cadena.

class ctypes.c_bool

Representa el tipo de dato C bool (más precisamente, _Bool de C99). Su valor puede ser `True` o `False`, y el constructor acepta cualquier objeto que tenga un valor de verdad.

class ctypes.HRESULT

Represents a **`HRESULT`** value, which contains success or error information for a function or method call.

Availability: Windows

class ctypes.py_object

Representa el tipo de dato de C PyObject*. Llamar esto sin un argumento crea un puntero PyObject* `NULL`.

Distinto en la versión 3.14: **`py_object`** is now a generic type.

The **`ctypes.wintypes`** module provides quite some other Windows specific data types, for example **`HWND`**, **`WPARAM`**, **`VARIANT_BOOL`** or **`DWORD`**. Some useful structures like **`MSG`** or **`RECT`** are also defined.

Tipos de datos estructurados

class ctypes.Union(**args*, ***kw*)

Clase base abstracta para uniones en orden de bytes nativos.

Unions share common attributes and behavior with structures; see **`Structure`** documentation for details.

class ctypes.BigEndianUnion(**args*, ***kw*)

Clase base abstracta para uniones en orden de bytes *big endian*.

Added in version 3.11.

class ctypes.LittleEndianUnion(**args*, ***kw*)

Clase base abstracta para uniones en orden de bytes *little endian*.

Added in version 3.11.

class ctypes.BigEndianStructure(**args*, ***kw*)

Clase base abstracta para estructuras en orden de bytes *big endian*.

class ctypes.LittleEndianStructure(**args*, ***kw*)

Clase base abstracta para estructuras en orden de bytes *little endian*.

Las estructuras y uniones con un orden de bytes no nativo no pueden contener campos de tipo puntero ni ningún otro tipo de datos que contenga campos de tipo puntero.

class ctypes.Structure(**args*, ***kw*)

Clase base abstracta para estructuras en orden de bytes *native*.

Concrete structure and union types must be created by subclassing one of these types, and at least define a **`_fields_`** class variable. **`ctypes`** will create descriptors which allow reading and writing the fields by direct attribute accesses. These are the

fields

Una secuencia que define los campos de estructura. Los elementos deben ser de 2 o 3 tuplas. El primer ítem es el nombre del campo, el segundo ítem especifica el tipo de campo; puede ser cualquier tipo de datos ctypes.

Para los campos de tipo entero como **`c_int`**, se puede dar un tercer elemento opcional. Debe ser un pequeño entero positivo que defina el ancho de bit del campo.

Los nombres de los campos deben ser únicos dentro de una estructura o unión. Esto no se comprueba, sólo se puede acceder a un campo cuando los nombres se repiten.

Es posible definir la variable de clase `_fields_` *después* de la sentencia de clase que define la subclase Estructura, esto permite crear tipos de datos que se refieren directa o indirectamente a sí mismos:

```
class List(Structure):
    pass
List._fields_ = [("pnext", POINTER(List)),
                 ...
                ]
```

The **`_fields_`** class variable can only be set once. Later assignments will raise an `AttributeError`.

Additionally, the **`_fields_`** class variable must be defined before the structure or union type is first used: an instance or subclass is created, `sizeof()` is called on it, and so on. Later assignments to **`_fields_`** will raise an `AttributeError`. If **`_fields_`** has not been set before such use, the structure or union will have no own fields, as if **`_fields_`** was empty.

Sub-subclasses of structure types inherit the fields of the base class plus the `_fields_` defined in the sub-subclass, if any.

pack

An optional small integer that allows overriding the alignment of structure fields in the instance.

This is only implemented for the MSVC-compatible memory layout (see `_layout_`).

Setting **`_pack_`** to 0 is the same as not setting it at all. Otherwise, the value must be a positive power of two. The effect is equivalent to `#pragma pack(N)` in C,

except **`ctypes`** may allow larger *n* than what the compiler accepts.

`_pack_` must already be defined when **`_fields_`** is assigned, otherwise it will have no effect.

Deprecated since version 3.14, will be removed in version 3.19: For historical reasons, if **`_pack_`** is non-zero, the MSVC-compatible layout will be used by default. On non-Windows platforms, this default is deprecated and is slated to become an error in Python 3.19. If it is intended, set **`_layout_`** to `'ms'` explicitly.

align

An optional small integer that allows increasing the alignment of the structure when being packed or unpacked to/from memory.

The value must not be negative. The effect is equivalent to `__attribute__((aligned(N)))` on GCC or `#pragma align(N)` on MSVC, except **`ctypes`** may allow values that the compiler would reject.

`_align_` can only *increase* a structure's alignment requirements. Setting it to 0 or 1 has no effect.

Using values that are not powers of two is discouraged and may lead to surprising behavior.

`_align_` must already be defined when **`_fields_`** is assigned, otherwise it will have no effect.

Added in version 3.13.

layout

An optional string naming the struct/union layout. It can currently be set to:

- `"ms"`: the layout used by the Microsoft compiler (MSVC). On GCC and Clang, this layout can be selected with

  `__attribute__((ms_struct))`.
- `"gcc-sysv"`: the layout used by GCC with the System V or “SysV-like” data model, as used on Linux and macOS. With this layout, **`_pack_`** must be unset or zero.

If not set explicitly, `ctypes` will use a default that matches the platform conventions. This default may change in future Python releases (for example, when a new platform gains official support, or when a difference between similar platforms is found). Currently the default will be:

- On Windows: `"ms"`
- When **`_pack_`** is specified: `"ms"`. (This is deprecated; see **`_pack_`** documentation.)
- Otherwise: `"gcc-sysv"`

`_layout_` must already be defined when **`_fields_`** is assigned, otherwise it will have no effect.

Added in version 3.14.

anonymous

Una secuencia opcional que enumera los nombres de los campos sin nombre (anónimos). **`_anonymous_`** debe estar ya definida cuando se asigna **`_fields_`**, de lo contrario no tendrá ningún efecto.

The fields listed in this variable must be structure or union type fields. **`ctypes`** will create descriptors in the structure type that allows accessing the nested fields directly, without the need to create the structure or union field.

Aquí hay un tipo de ejemplo (Windows):

```
class _U(Union):
    _fields_ = [("lptdesc", POINTER(TYPEDESC)),
                ("lpadesc", POINTER(ARRAYDESC)
                ("hreftype", HREFTYPE)]
```

```
class TYPEDESC(Structure):
    _anonymous_ = ("u",)
    _fields_ = [("u", _U),
                ("vt", VARTYPE)]
```

La estructura `TYPEDESC` describe un tipo de datos COM, el campo `vt` especifica cuál de los campos de unión es válido. Como el campo `u` está definido como campo anónimo, ahora es posible acceder a los miembros directamente desde la instancia TYPEDESC. `td.lptdesc` y `td.u.lptdesc` son equivalentes, pero el primero es más rápido ya que no necesita crear una instancia de unión temporal:

```
td = TYPEDESC()
td.vt = VT_PTR
td.lptdesc = POINTER(some_type)
td.u.lptdesc = POINTER(some_type)
```

Es posible definir subclases de estructuras, que heredan los campos de la clase base. Si la definición de la subclase tiene una variable **`_fields_`** separada, los campos especificados en ella se añaden a los campos de la clase base.

Los constructores de estructuras y uniones aceptan tanto argumentos posicionales como de palabras clave. Los argumentos posicionales se usan para inicializar los campos de los miembros en el mismo orden en que aparecen en **`_fields_`**. Los argumentos de palabras clave en el constructor se interpretan como asignaciones de atributos, por lo que inicializarán **`_fields_`** con el mismo nombre, o crearán nuevos atributos para nombres no presentes en **`_fields_`**.

class ctypes.CField(**args*, ***kw*)

Descriptor for fields of a **`Structure`** and **`Union`**. For example:

```
>>> class Color(Structure):
```

```
...     _fields_ = (
...         ('red', c_uint8),
...         ('green', c_uint8),
...         ('blue', c_uint8),
...         ('intense', c_bool, 1),
...         ('blinking', c_bool, 1),
...    )
...
>>> Color.red
<ctypes.CField 'red' type=c_ubyte, ofs=0, size=1>
>>> Color.green.type
<class 'ctypes.c_ubyte'>
>>> Color.blue.byte_offset
2
>>> Color.intense
<ctypes.CField 'intense' type=c_bool, ofs=3, bit_si
>>> Color.blinking.bit_offset
1
```

All attributes are read-only.

`CField` objects are created via **`_fields_`**; do not instantiate the class directly.

Added in version 3.14: Previously, descriptors only had `offset` and `size` attributes and a readable string representation; the **`CField`** class was not available directly.

name

Name of the field, as a string.

type

Type of the field, as a ctypes class.

offset
byte_offset

Offset of the field, in bytes.

For bitfields, this is the offset of the underlying bytealigned *storage unit*; see `bit_offset`.

byte_size

Size of the field, in bytes.

For bitfields, this is the size of the underlying *storage unit*. Typically, it has the same size as the bitfield's type.

size

For non-bitfields, equivalent to `byte_size`.

For bitfields, this contains a backwards-compatible bit-packed value that combines `bit_size` and `bit_offset`. Prefer using the explicit attributes instead.

is_bitfield

True if this is a bitfield.

bit_offset
bit_size

The location of a bitfield within its *storage unit*, that is, within `byte_size` bytes of memory starting at `byte_offset`.

To get the field's value, read the storage unit as an integer, shift left by **`bit_offset`** and take the **`bit_size`** least significant bits.

For non-bitfields, **`bit_offset`** is zero and **`bit_size`** is equal to `byte_size * 8`.

is_anonymous

True if this field is anonymous, that is, it contains nested sub-fields that should be merged into a containing structure or union.

Arreglos y punteros

class ctypes.Array(**args*)

Clase base abstracta para arreglos.

The recommended way to create concrete array types is by multiplying any **`ctypes`** data type with a non-negative integer. Alternatively, you can subclass this type and define **`_length_`** and **`_type_`** class variables. Array elements can be read and written using standard subscript and slice accesses; for slice reads, the resulting object is *not* itself an **`Array`**.

length

Un número entero positivo que especifica el número de elementos del conjunto. Los subíndices fuera de rango dan como resultado un **`IndexError`**. Será retornado por **`len()`**.

type

Especifica el tipo de cada elemento del arreglo.

Los constructores de subclases de arreglos aceptan argumentos posicionales, usados para inicializar los elementos en orden.

ctypes.ARRAY(*type*, *length*)

Create an array. Equivalent to `type * length`, where *type* is a **`ctypes`** data type and *length* an integer.

This function is soft deprecated in favor of multiplication. There are no plans to remove it.

class ctypes._Pointer

Clase base, privada y abstracta para punteros.

Los tipos de punteros concretos se crean llamando a **`POINTER()`** con el tipo que será apuntado; esto se hace automáticamente por `pointer()`.

Si un puntero apunta a un arreglo, sus elementos pueden ser leídos y escritos usando accesos de subíndices y cortes estándar. Los objetos punteros no tienen tamaño, así que `len()` lanzará un `TypeError`. Los subíndices negativos se leerán de la memoria *antes* que el puntero (como en C), y los subíndices fuera de rango probablemente se bloqueen con una violación de acceso (si tienes suerte).

type

Especifica el tipo apuntado.

contents

Retorna el objeto al que el puntero apunta. Asignando a este atributo cambia el puntero para que apunte al objeto asignado.

Exceptions

exception ctypes.ArgumentError

Esta excepción se lanza cuando una llamada a una función foránea no puede convertir uno de los argumentos pasados.

exception ctypes.COMError(*hresult*, *text*, *details*)

This exception is raised when a COM method call failed.

hresult

The integer value representing the error code.

text

The error message.

details

The 5-tuple `(descr, source, helpfile, helpcontext, progid)`.

descr is the textual description. *source* is the language-

dependent `ProgID` for the class or application that raised the error. *helpfile* is the path of the help file. *helpcontext* is the help context identifier. *progid* is the `ProgID` of the interface that defined the error.

Availability: Windows

Added in version 3.14.

Command-line interface libraries

The modules described in this chapter assist with implementing command line and terminal interfaces for applications.

Here’s an overview:

- **`argparse`** — Parser for command-line options, arguments and subcommands
- **`optparse`** — Parser for command line options
- **`getpass`** — Portable password input
- **`fileinput`** — Iterate over lines from multiple input streams
- **`curses`** — Terminal handling for character-cell displays
- **`curses.textpad`** — Text input widget for curses programs
- **`curses.ascii`** — Utilities for ASCII characters
- **`curses.panel`** — A panel stack extension for curses
- **`cmd`** — Support for line-oriented command interpreters

`argparse` — Parser for command-line options, arguments and subcommands

Added in version 3.2.

Código fuente: Lib/argparse.py [https://github.com/python/cpython/tree/3.14/Lib/argparse.py]

Nota

While **`argparse`** is the default recommended standard library module for implementing basic command line applications, authors with more exacting requirements for exactly how their command line applications behave may find it doesn't provide the necessary level of control. Refer to Choosing an argument parsing library for alternatives to consider when `argparse` doesn't support behaviors that the application requires (such as entirely disabling support for interspersed options and positional arguments, or accepting option parameter values that start with – even when they correspond to another defined option).

Tutorial

Esta página contiene la información de referencia de la API. Para una introducción más amigable al análisis de la línea de comandos de Python, echa un vistazo al argparse tutorial.

The **`argparse`** module makes it easy to write user-friendly command-line interfaces. The program defines what arguments it requires, and **`argparse`** will figure out how to parse those out of **`sys.argv`**. The **`argparse`** module also automatically generates help and usage messages. The module will also issue errors when users give the program invalid arguments.

The **`argparse`** module’s support for command-line interfaces is built around an instance of **`argparse.ArgumentParser`**. It is a container for argument specifications and has options that apply to the parser as whole:

```
parser = argparse.ArgumentParser(
                    prog='ProgramName',
                    description='What the program does',
                    epilog='Text at the bottom of help')
```

El método **`ArgumentParser.add_argument()`** adjunta especificaciones de argumentos individuales al analizador. Soporta argumentos posicionales, opciones que aceptan valores, y banderas de activación y desactivación:

```
parser.add_argument('filename')           # positional a
parser.add_argument('-c', '--count')      # option that
parser.add_argument('-v', '--verbose',
                    action='store_true')  # on/off flag
```

El método **`ArgumentParser.parse_args()`** corre el analizador y coloca los datos extraídos en un objeto **`argparse.Namespace`**:

```
args = parser.parse_args()
print(args.filename, args.count, args.verbose)
```

Nota

If you’re looking for a guide about how to upgrade **`optparse`** code to **`argparse`**, see Upgrading Optparse Code.

Objetos *ArgumentParser*

class argparse.ArgumentParser(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=[]*, *formatter_class=argparse.HelpFormatter*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error', add_help=True, allow_abbrev=True, exit_on_error=True, *, suggest_on_error=False, color=True*)

Crea un nuevo objeto **`ArgumentParser`**. Todos los parámetros deben pasarse como argumentos de palabra clave. Cada parámetro tiene su propia descripción más detallada a continuación, pero en resumen son:

- prog - The name of the program (default: generated from the `__main__` module attributes and `sys.argv[0]`)
- usage - La cadena de caracteres que describe el uso del programa (por defecto: generado a partir de los argumentos añadidos al analizador)
- description - Texto a mostrar antes del argumento ayuda (por defecto, ninguno)
- epilog - Texto a mostrar después del argumento ayuda (por defecto, ninguno)
- parents - Una lista de objetos **`ArgumentParser`** cuyos argumentos también deberían ser incluidos
- formatter_class - Una clase para personalizar la salida de la ayuda
- prefix_chars - El conjunto de caracteres que preceden a los argumentos opcionales (por defecto: ‘-‘)
- fromfile_prefix_chars - El conjunto de caracteres que preceden a los archivos de los cuales se deberían leer los argumentos adicionales (por defecto: `None`)
- argument_default - El valor global por defecto de los argumentos (por defecto: `None`)
- conflict_handler - La estrategia para resolver los opcionales conflictivos (normalmente es innecesaria)
- add_help - Añade una opción `-h/--help` al analizador (por defecto: `True`)
- allow_abbrev - Allows long options to be abbreviated if the abbreviation is unambiguous (default: `True`)
- exit_on_error - Determines whether or not **`ArgumentParser`** exits with error info when an error occurs. (default: `True`)
- suggest_on_error - Enables suggestions for mistyped argument choices and subparser names (default: `False`)

- color - Allow color output (default: `True`)

Distinto en la versión 3.5: se añadió el parámetro *allow_abbrev*.

Distinto en la versión 3.8: En versiones anteriores, *allow_abbrev* también deshabilitaba la agrupación de banderas (*flags*) cortas como `-vv` para que sea `-v -v`.

Distinto en la versión 3.9: Se agregó el parámetro *exit_on_error*.

Distinto en la versión 3.14: suggest_on_error and *color* parameters were added.

En las siguientes secciones se describe cómo se utiliza cada una de ellas.

prog

By default, **`ArgumentParser`** calculates the name of the program to display in help messages depending on the way the Python interpreter was run:

- The **`base name`** of `sys.argv[0]` if a file was passed as argument.
- The Python interpreter name followed by `sys.argv[0]` if a directory or a zipfile was passed as argument.
- The Python interpreter name followed by `-m` followed by the module or package name if the **`-m`** option was used.

This default is almost always desirable because it will make the help messages match the string that was used to invoke the program on the command line. However, to change this default behavior, another value can be supplied using the `prog=` argument to **`ArgumentParser`**:

```
>>> parser = argparse.ArgumentParser(prog='myprogram')
>>> parser.print_help()
usage: myprogram [-h]

options:
 -h, --help  show this help message and exit
```

Note that the program name, whether determined from `sys.argv[0]`, from the `__main__` module attributes or from the `prog=` argument, is available to help messages using the `%(prog)s` format specifier.

```
>>> parser = argparse.ArgumentParser(prog='myprogram')
>>> parser.add_argument('--foo', help='foo of the %(prog
>>> parser.print_help()
usage: myprogram [-h] [--foo FOO]

options:
 -h, --help  show this help message and exit
 --foo FOO   foo of the myprogram program
```

Distinto en la versión 3.14: The default `prog` value now reflects how `__main__` was actually executed, rather than always being `os.path.basename(sys.argv[0])`.

uso

By default, **`ArgumentParser`** calculates the usage message from the arguments it contains. The default message can be overridden with the `usage=` keyword argument:

```
>>> parser = argparse.ArgumentParser(prog='PROG', usage=
>>> parser.add_argument('--foo', nargs='?', help='foo he
>>> parser.add_argument('bar', nargs='+', help='bar help
>>> parser.print_help()
usage: PROG [options]

positional arguments:
 bar          bar help

options:
 -h, --help   show this help message and exit
 --foo [FOO]  foo help
```

El especificador de formato `%(prog)s` está preparado para introducir el nombre del programa en los mensajes de ayuda.

When a custom usage message is specified for the main parser, you may also want to consider passing the `prog` argument to **`add_subparsers()`** or the `prog` and the `usage` arguments to **`add_parser()`**, to ensure consistent command prefixes and usage information across subparsers.

description

Most calls to the **`ArgumentParser`** constructor will use the `description=` keyword argument. This argument gives a brief description of what the program does and how it works. In help messages, the description is displayed between the command-line usage string and the help messages for the various arguments.

Por defecto, la descripción será ajustada a una línea para que encaje en el espacio dado. Para cambiar este comportamiento, revisa el argumento formatter_class.

epilog

A algunos programas les gusta mostrar una descripción adicional del programa después de la descripción de los argumentos. Dicho texto puede ser especificado usando el argumento `epilog=` para **`ArgumentParser`**:

```
>>> parser = argparse.ArgumentParser(
...     description='A foo that bars',
...     epilog="And that's how you'd foo a bar")
>>> parser.print_help()
usage: argparse.py [-h]

A foo that bars

options:
 -h, --help  show this help message and exit

And that's how you'd foo a bar
```

Al igual que con el argumento description, el texto `epilog=` está por defecto ajustado a una línea, pero este comportamiento puede ser modificado con el argumento formatter_class para **`ArgumentParser`**.

parents

A veces, varios analizadores comparten un conjunto de argumentos comunes. En lugar de repetir las definiciones de estos argumentos, se puede usar un único analizador con todos los argumentos compartidos y pasarlo en el argumento `parents=` a **`ArgumentParser`**. El argumento `parents=` toma una lista de objetos **`ArgumentParser`**, recoge todas las acciones de posición y de opción de éstos, y añade estas acciones al objeto **`ArgumentParser`** que se está construyendo:

```
>>> parent_parser = argparse.ArgumentParser(add_help=Fal
>>> parent_parser.add_argument('--parent', type=int)

>>> foo_parser = argparse.ArgumentParser(parents=[parent
>>> foo_parser.add_argument('foo')
>>> foo_parser.parse_args(['--parent', '2', 'XXX'])
Namespace(foo='XXX', parent=2)

>>> bar_parser = argparse.ArgumentParser(parents=[parent
>>> bar_parser.add_argument('--bar')
>>> bar_parser.parse_args(['--bar', 'YYY'])
Namespace(bar='YYY', parent=None)
```

Ten en cuenta que la mayoría de los analizadores padre especificarán `add_help=False`. De lo contrario, el **`ArgumentParser`** verá dos opciones `-h/--help` (una para el padre y otra para el hijo) y generará un error.

Nota

Debes inicializar completamente los analizadores antes de pasarlos a través de `parents=`. Si cambias los analizadores padre después del analizador hijo, esos cambios no se reflejarán en el hijo.

formatter_class

Los objetos **`ArgumentParser`** permiten personalizar el formato de la ayuda especificando una clase de formato alternativa. Actualmente, hay cuatro clases de este tipo:

class argparse.RawDescriptionHelpFormatter
class argparse.RawTextHelpFormatter
class argparse.ArgumentDefaultsHelpFormatter
class argparse.MetavarTypeHelpFormatter

`RawDescriptionHelpFormatter` y **`RawTextHelpFormatter`** dan más control sobre cómo se muestran las descripciones de texto. Por defecto, los objetos **`ArgumentParser`** ajustan a la línea los textos de description y epilog en los mensajes de ayuda de la línea de comandos:

```
>>> parser = argparse.ArgumentParser(
...     prog='PROG',
...     description='''this description
...         was indented weird
...             but that is okay''',
...     epilog='''
...             likewise for this epilog whose whitespac
...         be cleaned up and whose words will be wrappe
...         across a couple lines''')
>>> parser.print_help()
usage: PROG [-h]

this description was indented weird but that is okay

options:
 -h, --help  show this help message and exit

likewise for this epilog whose whitespace will be cleane
will be wrapped across a couple lines
```

Pasar **`RawDescriptionHelpFormatter`** como `formatter_class=` indica que description y epilog ya tienen el formato correcto y no deben ser ajustados a la línea:

```
>>> parser = argparse.ArgumentParser(
...     prog='PROG',
...     formatter_class=argparse.RawDescriptionHelpForma
...     description=textwrap.dedent('''\
...         Please do not mess up this text!
...         --------------------------------
...             I have indented it
...             exactly the way
...             I want it
...         '''))
>>> parser.print_help()
usage: PROG [-h]

Please do not mess up this text!
--------------------------------
   I have indented it
   exactly the way
   I want it

options:
 -h, --help  show this help message and exit
```

RawTextHelpFormatter maintains whitespace for all sorts of help text, including argument descriptions. However, multiple newlines are replaced with one. If you wish to preserve multiple blank lines, add spaces between the newlines.

ArgumentDefaultsHelpFormatter añade automáticamente información sobre los valores por defecto a cada uno de los mensajes de ayuda de los argumentos:

```
>>> parser = argparse.ArgumentParser(
...     prog='PROG',
...     formatter_class=argparse.ArgumentDefaultsHelpFor
>>> parser.add_argument('--foo', type=int, default=42, h
>>> parser.add_argument('bar', nargs='*', default=[1, 2,
>>> parser.print_help()
```

```
usage: PROG [-h] [--foo FOO] [bar ...]

positional arguments:
 bar          BAR! (default: [1, 2, 3])

options:
 -h, --help   show this help message and exit
 --foo FOO    FOO! (default: 42)
```

`MetavarTypeHelpFormatter` utiliza el nombre del parámetro type para cada argumento como el nombre a mostrar para sus valores (en lugar de utilizar dest como lo hace el formato habitual):

```
>>> parser = argparse.ArgumentParser(
...     prog='PROG',
...     formatter_class=argparse.MetavarTypeHelpFormatte
>>> parser.add_argument('--foo', type=int)
>>> parser.add_argument('bar', type=float)
>>> parser.print_help()
usage: PROG [-h] [--foo int] float

positional arguments:
  float

options:
  -h, --help  show this help message and exit
  --foo int
```

prefix_chars

Most command-line options will use `-` as the prefix, e.g. `-f/--foo`. Parsers that need to support different or additional prefix characters, e.g. for options like `+f` or `/foo`, may specify them using the `prefix_chars=` argument to the **`ArgumentParser`** constructor:

```
>>> parser = argparse.ArgumentParser(prog='PROG', prefix
>>> parser.add_argument('+f')
>>> parser.add_argument('++bar')
>>> parser.parse_args('+f X ++bar Y'.split())
Namespace(bar='Y', f='X')
```

El argumento `prefix_chars=` tiene un valor por defecto de `'-'`. Proporcionar un conjunto de caracteres que no incluya `-` causará que las opciones `-f/--foo` no sean inhabilitadas.

fromfile_prefix_chars

Algunas veces, cuando se trata de una lista de argumentos particularmente larga, puede tener sentido mantener la lista de argumentos en un archivo en lugar de escribirla en la línea de comandos. Si el argumento `fromfile_prefix_chars=` se da al constructor **`ArgumentParser`**, entonces los argumentos que empiezan con cualquiera de los caracteres especificados se tratarán como archivos, y serán reemplazados por los argumentos que contienen. Por ejemplo:

```
>>> with open('args.txt', 'w', encoding=sys.getfilesyste
...     fp.write('-f\nbar')
...
>>> parser = argparse.ArgumentParser(fromfile_prefix_cha
>>> parser.add_argument('-f')
>>> parser.parse_args(['-f', 'foo', '@args.txt'])
Namespace(f='bar')
```

Arguments read from a file must be one per line by default (but see also **`convert_arg_line_to_args()`**) and are treated as if they were in the same place as the original file referencing argument on the command line. So in the example above, the expression `['-f', 'foo', '@args.txt']` is considered equivalent to the expression `['-f', 'foo', '-f', 'bar']`.

Nota

Empty lines are treated as empty strings (`''`), which are allowed as values but not as arguments. Empty lines that are read as arguments will result in an «unrecognized arguments» error.

`ArgumentParser` utiliza codificación del sistema de archivos y manejador de errores para leer el archivo que contiene argumentos.

El argumento `fromfile_prefix_chars=` por defecto es `None`, lo que significa que los argumentos nunca serán tratados como referencias de archivos.

Distinto en la versión 3.12: **`ArgumentParser`** changed encoding and errors to read arguments files from default (e.g. **`locale.getpreferredencoding(False)`** and `"strict"`) to the filesystem encoding and error handler. Arguments file should be encoded in UTF-8 instead of ANSI Codepage on Windows.

argument_default

Generalmente, los valores por defecto de los argumentos se especifican ya sea pasando un valor por defecto a **`add_argument()`** o llamando a los métodos **`set_defaults()`** con un conjunto específico de pares nombre-valor. A veces, sin embargo, puede ser útil especificar un único valor por defecto para todos los argumentos del analizador. Esto se puede lograr pasando el argumento de palabra clave `argument_default=` a **`ArgumentParser`**. Por ejemplo, para suprimir globalmente la creación de atributos en las llamadas a **`parse_args()`** , proporcionamos el argumento `argument_default=SUPPRESS`:

```
>>> parser = argparse.ArgumentParser(argument_default=ar
>>> parser.add_argument('--foo')
>>> parser.add_argument('bar', nargs='?')
>>> parser.parse_args(['--foo', '1', 'BAR'])
Namespace(bar='BAR', foo='1')
>>> parser.parse_args([])
Namespace()
```

allow_abbrev

Normalmente, cuando pasas una lista de argumentos al método **`parse_args()`** de un **`ArgumentParser`**, reconoce las abreviaturas de las opciones largas.

Esta característica puede ser desactivada poniendo `allow_abbrev` a `False`:

```
>>> parser = argparse.ArgumentParser(prog='PROG', allow_
>>> parser.add_argument('--foobar', action='store_true')
>>> parser.add_argument('--foonley', action='store_false
>>> parser.parse_args(['--foon'])
usage: PROG [-h] [--foobar] [--foonley]
PROG: error: unrecognized arguments: --foon
```

Added in version 3.5.

conflict_handler

Los objetos **`ArgumentParser`** no permiten dos acciones con la misma cadena de caracteres de opción. Por defecto, los objetos **`ArgumentParser`** lanzan una excepción si se intenta crear un argumento con una cadena de caracteres de opción que ya está en uso:

```
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> parser.add_argument('-f', '--foo', help='old foo hel
>>> parser.add_argument('--foo', help='new foo help')
Traceback (most recent call last):
 ..
ArgumentError: argument --foo: conflicting option string
```

A veces (por ejemplo, cuando se utiliza parents) puede ser útil anular simplemente cualquier argumento antiguo con la misma cadena de caracteres de opción. Para lograr este comportamiento, se puede suministrar el valor `'resolve'` al argumento `conflict_handler=` de **`ArgumentParser`**:

```
>>> parser = argparse.ArgumentParser(prog='PROG', confli
>>> parser.add_argument('-f', '--foo', help='old foo hel
>>> parser.add_argument('--foo', help='new foo help')
>>> parser.print_help()
usage: PROG [-h] [-f FOO] [--foo FOO]

options:
 -h, --help  show this help message and exit
 -f FOO      old foo help
 --foo FOO   new foo help
```

Ten en cuenta que los objetos **`ArgumentParser`** sólo eliminan una acción si todas sus cadenas de caracteres de opción están anuladas. Así, en el ejemplo anterior, la antigua acción `-f/--foo` se mantiene como la acción `-f`, porque sólo la cadena de caracteres de opción `--foo` fue anulada.

add_help

By default, **`ArgumentParser`** objects add an option which simply displays the parser's help message. If `-h` or `--help` is supplied at the command line, the **`ArgumentParser`** help will be printed.

Ocasionalmente, puede ser útil desactivar la inclusión de esta opción de ayuda. Esto se puede lograr pasando `False` como argumento de `add_help=` a **`ArgumentParser`**:

```
>>> parser = argparse.ArgumentParser(prog='PROG', add_he
>>> parser.add_argument('--foo', help='foo help')
>>> parser.print_help()
usage: PROG [--foo FOO]

options:
 --foo FOO  foo help
```

La opción de ayuda es típicamente `-h/--help`. La excepción a esto es si `prefix_chars=` se especifica y no incluye `-`, en cuyo caso `-h` y `--help` no son opciones válidas. En este caso, el primer carácter en `prefix_chars` se utiliza para preceder a las opciones de ayuda:

```
>>> parser = argparse.ArgumentParser(prog='PROG', prefix
>>> parser.print_help()
usage: PROG [+h]

options:
  +h, ++help  show this help message and exit
```

exit_on_error

Normally, when you pass an invalid argument list to the **`parse_args()`** method of an **`ArgumentParser`**, it will print a *message* to **`sys.stderr`** and exit with a status code of 2.

Si el usuario desea detectar errores manualmente, la función se puede habilitar configurando `exit_on_error` en `False`

```
>>> parser = argparse.ArgumentParser(exit_on_error=False
>>> parser.add_argument('--integers', type=int)
_StoreAction(option_strings=['--integers'], dest='intege
>>> try:
...     parser.parse_args('--integers a'.split())
... except argparse.ArgumentError:
...     print('Catching an argumentError')
...
Catching an argumentError
```

Added in version 3.9.

suggest_on_error

By default, when a user passes an invalid argument choice or subparser name, **`ArgumentParser`** will exit with error info and list the permissible argument choices (if specified) or subparser names as part of the error message.

If the user would like to enable suggestions for mistyped argument choices and subparser names, the feature can be enabled by setting `suggest_on_error` to `True`. Note that this only applies for arguments when the choices specified are strings:

```
>>> parser = argparse.ArgumentParser(suggest_on_error=Tr
>>> parser.add_argument('--action', choices=['debug', 'd
>>> parser.parse_args(['--action', 'debugg'])
usage: tester.py [-h] [--action {debug,dryrun}]
tester.py: error: argument --action: invalid choice: 'de
```

If you’re writing code that needs to be compatible with older

Python versions and want to opportunistically use `suggest_on_error` when it's available, you can set it as an attribute after initializing the parser instead of using the keyword argument:

```
>>> parser = argparse.ArgumentParser(description='Proces
>>> parser.suggest_on_error = True
```

Added in version 3.14.

color

By default, the help message is printed in color using ANSI escape sequences [https://en.wikipedia.org/wiki/ANSI_escape_code]. If you want plain text help messages, you can disable this in your local environment, or in the argument parser itself by setting `color` to `False`:

```
>>> parser = argparse.ArgumentParser(description='Proces
...                                  color=False)
>>> parser.add_argument('--action', choices=['sum', 'max
>>> parser.add_argument('integers', metavar='N', type=ir
...                     help='an integer for the accumul
>>> parser.parse_args(['--help'])
```

Note that when `color=True`, colored output depends on both environment variables and terminal capabilities. However, if `color=False`, colored output is always disabled, even if environment variables like `FORCE_COLOR` are set.

Nota

Error messages will include color codes when redirecting stderr to a file. To avoid this, set the **`NO_COLOR`** [https://no-color.org/] or **`PYTHON_COLORS`** environment variable (for example, `NO_COLOR=1 python script.py 2> errors.txt`).

Added in version 3.14.

El método *add_argument()*

ArgumentParser.add_argument(*name or flags...*, *[, *action*] [, *nargs*] [, *const*] [, *default*] [, *type*] [, *choices*] [, *required*] [, *help*] [, *metavar*] [, *dest*] [, *deprecated*])

Define cómo se debe interpretar un determinado argumento de línea de comandos. Cada parámetro tiene su propia descripción más detallada a continuación, pero en resumen son:

- name or flags - Either a name or a list of option strings, e.g. `'foo'` or `'-f', '--foo'`.
- action - El tipo básico de acción a tomar cuando este argumento se encuentra en la línea de comandos.
- nargs - El número de argumentos de la línea de comandos que deben ser consumidos.
- const - Un valor fijo requerido por algunas selecciones de action y nargs.
- default - El valor producido si el argumento está ausente en la línea de comando y si está ausente en el objeto de espacio de nombres.
- type - El tipo al que debe convertirse el argumento de la línea de comandos.
- choices - Una secuencia de valores permitidos para el argumento.
- required - Si se puede omitir o no la opción de la línea de comandos (sólo opcionales).
- help - Una breve descripción de lo que hace el argumento.
- metavar - Un nombre para el argumento en los mensajes de uso.
- dest - El nombre del atributo que será añadido al objeto retornado por **`parse_args()`**.
- deprecated - Whether or not use of the argument is deprecated.

The method returns an **`Action`** object representing the argument.

En las siguientes secciones se describe cómo se utiliza cada una de ellas.

name or flags

El método **`add_argument()`** debe saber si espera un argumento opcional, como `-f` o `--foo`, o un argumento posicional, como una lista de nombres de archivos. Los primeros argumentos que se pasan a **`add_argument()`** deben ser o bien una serie de banderas (*flags*), o un simple nombre de un argumento (*name*).

Por ejemplo, se puede crear un argumento opcional como:

```
>>> parser.add_argument('-f', '--foo')
```

mientras que un argumento posicional podría ser creado como:

```
>>> parser.add_argument('bar')
```

Cuando se llama a **`parse_args()`** , los argumentos opcionales serán identificados por el prefijo `-`, y el resto de los argumentos serán asumidos como posicionales:

```
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> parser.add_argument('-f', '--foo')
>>> parser.add_argument('bar')
>>> parser.parse_args(['BAR'])
Namespace(bar='BAR', foo=None)
>>> parser.parse_args(['BAR', '--foo', 'FOO'])
Namespace(bar='BAR', foo='FOO')
>>> parser.parse_args(['--foo', 'FOO'])
usage: PROG [-h] [-f FOO] bar
PROG: error: the following arguments are required: bar
```

By default, **`argparse`** automatically handles the internal naming and display names of arguments, simplifying the process without requiring additional configuration. As such, you do not need to specify the dest and metavar parameters. For optional arguments, the dest parameter defaults to the argument name, with underscores _ replacing hyphens `-`. The metavar parameter defaults to the upper-cased name. For example:

```
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> parser.add_argument('--foo-bar')
>>> parser.parse_args(['--foo-bar', 'FOO-BAR'])
Namespace(foo_bar='FOO-BAR')
>>> parser.print_help()
usage:  [-h] [--foo-bar FOO-BAR]

optional arguments:
 -h, --help  show this help message and exit
 --foo-bar FOO-BAR
```

action

Los objetos **`ArgumentParser`** asocian los argumentos de la línea de comandos con las acciones. Esta acciones pueden hacer casi cualquier cosa con los argumentos de línea de comandos asociados a ellas, aunque la mayoría de las acciones simplemente añaden un atributo al objeto retornado por **`parse_args()`**. El argumento de palabra clave `action` especifica cómo deben ser manejados los argumentos de la línea de comandos. Las acciones proporcionadas son:

- `'store'` - This just stores the argument's value. This is the default action.

- `'store_const'` - Esto almacena el valor especificado por el argumento de palabra clave const; nótese que el argumento de palabra clave const por defecto es `None`. La acción `'store_const'` se usa comúnmente con argumentos opcionales que especifican algún tipo de indicador (*flag*). Por ejemplo:

  ```
  >>> parser = argparse.ArgumentParser()
  >>> parser.add_argument('--foo', action='store_cons
  >>> parser.parse_args(['--foo'])
  Namespace(foo=42)
  ```

- `'store_true'` and `'store_false'` - These are special cases of `'store_const'` that respectively store the values `True` and `False` with default values of `False` and `True`:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo', action='store_true
>>> parser.add_argument('--bar', action='store_fals
>>> parser.add_argument('--baz', action='store_fals
>>> parser.parse_args('--foo --bar'.split())
Namespace(foo=True, bar=False, baz=True)
```

- `'append'` - This appends each argument value to a list. It is useful for allowing an option to be specified multiple times. If the default value is a non-empty list, the parsed value will start with the default list's elements and any values from the command line will be appended after those default values. Example usage:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo', action='append', d
>>> parser.parse_args('--foo 1 --foo 2'.split())
Namespace(foo=['0', '1', '2'])
```

- `'append_const'` - This appends the value specified by the const keyword argument to a list; note that the const keyword argument defaults to `None`. The `'append_const'` action is typically useful when multiple arguments need to store constants to the same list. For example:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--str', dest='types', acti
>>> parser.add_argument('--int', dest='types', acti
>>> parser.parse_args('--str --int'.split())
Namespace(types=[<class 'str'>, <class 'int'>])
```

- `'extend'` - This appends each item from a multi-value argument to a list. The `'extend'` action is typically used with the nargs keyword argument value `'+'` or `'*'`. Note that when nargs is `None` (the default) or `'?'`, each character of the argument string will be appended to the list. Example usage:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument("--foo", action="extend", n
>>> parser.parse_args(["--foo", "f1", "--foo", "f2"
Namespace(foo=['f1', 'f2', 'f3', 'f4'])
```

Added in version 3.8.

- `'count'` - This counts the number of times an argument occurs. For example, this is useful for increasing verbosity levels:

  ```
  >>> parser = argparse.ArgumentParser()
  >>> parser.add_argument('--verbose', '-v', action='
  >>> parser.parse_args(['-vvv'])
  Namespace(verbose=3)
  ```

  Observa, *default* (el valor por defecto) será `None` a menos que explícitamente se establezca como *0*.

- `'help'` - Esta imprime un mensaje de ayuda completo para todas las opciones del analizador actual y luego termina. Por defecto, se añade una acción de ayuda automáticamente al analizador. Ver **`ArgumentParser`** para detalles de cómo se genera la salida.

- `’version’` - Esta espera un argumento de palabra clave `version=` en la llamada **`add_argument()`**, e imprime la información de la versión y finaliza cuando es invocada:

  ```
  >>> import argparse
  >>> parser = argparse.ArgumentParser(prog='PROG')
  >>> parser.add_argument('--version', action='versio
  >>> parser.parse_args(['--version'])
  PROG 2.0
  ```

You may also specify an arbitrary action by passing an **`Action`** subclass (e.g. **`BooleanOptionalAction`**) or other object that implements the same interface. Only actions that consume command-line arguments (e.g. `'store'`, `'append'`, `'extend'`, or custom actions with non-zero `nargs`) can be used with positional arguments.

The recommended way to create a custom action is to extend

`Action`, overriding the **`__call__()`** method and optionally the **`__init__()`** and **`format_usage()`** methods. You can also register custom actions using the **`register()`** method and reference them by their registered name.

Un ejemplo de una acción personalizada:

```
>>> class FooAction(argparse.Action):
...     def __init__(self, option_strings, dest, nargs=N
...         if nargs is not None:
...             raise ValueError("nargs not allowed")
...         super().__init__(option_strings, dest, **kwa
...     def __call__(self, parser, namespace, values, op
...         print('%r %r %r' % (namespace, values, optio
...         setattr(namespace, self.dest, values)
...
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo', action=FooAction)
>>> parser.add_argument('bar', action=FooAction)
>>> args = parser.parse_args('1 --foo 2'.split())
Namespace(bar=None, foo=None) '1' None
Namespace(bar='1', foo=None) '2' '--foo'
>>> args
Namespace(bar='1', foo='2')
```

Para más detalles, ver **`Action`**.

nargs

`ArgumentParser` objects usually associate a single command-line argument with a single action to be taken. The `nargs` keyword argument associates a different number of command-line arguments with a single action. See also Especificando argumentos ambiguos. The supported values are:

- `N` (un entero). `N` argumentos de la línea de comandos se agruparán en una lista. Por ejemplo:

  ```
  >>> parser = argparse.ArgumentParser()
  >>> parser.add_argument('--foo', nargs=2)
  ```

```
>>> parser.add_argument('bar', nargs=1)
>>> parser.parse_args('c --foo a b'.split())
Namespace(bar=['c'], foo=['a', 'b'])
```

Ten en cuenta que `nargs=1` produce una lista de un elemento. Esto es diferente del valor por defecto, en el que el elemento se produce por sí mismo.

- `'?'`. Un argumento se consumirá desde la línea de comandos si es posible, y se generará como un sólo elemento. Si no hay ningún argumento de línea de comandos, se obtendrá el valor de default. Ten en cuenta que para los argumentos opcionales, hay un caso adicional - la cadena de caracteres de opción está presente pero no va seguida de un argumento de línea de comandos. En este caso se obtendrá el valor de const. Algunos ejemplos para ilustrar esto:

  ```
  >>> parser = argparse.ArgumentParser()
  >>> parser.add_argument('--foo', nargs='?', const='
  >>> parser.add_argument('bar', nargs='?', default='
  >>> parser.parse_args(['XX', '--foo', 'YY'])
  Namespace(bar='XX', foo='YY')
  >>> parser.parse_args(['XX', '--foo'])
  Namespace(bar='XX', foo='c')
  >>> parser.parse_args([])
  Namespace(bar='d', foo='d')
  ```

  Uno de los usos más comunes de `nargs='?'` es permitir archivos de entrada y salida opcionales:

  ```
  >>> parser = argparse.ArgumentParser()
  >>> parser.add_argument('infile', nargs='?')
  >>> parser.add_argument('outfile', nargs='?')
  >>> parser.parse_args(['input.txt', 'output.txt'])
  Namespace(infile='input.txt', outfile='output.txt')
  >>> parser.parse_args(['input.txt'])
  Namespace(infile='input.txt', outfile=None)
  >>> parser.parse_args([])
  Namespace(infile=None, outfile=None)
  ```

- `’*’`. Todos los argumentos presentes en la línea de comandos se recogen en una lista. Ten en cuenta que generalmente no tiene mucho sentido tener más de un argumento posicional con `nargs=‘*’`, pero es posible tener múltiples argumentos opcionales con `nargs=‘*’`. Por ejemplo:

  ```
  >>> parser = argparse.ArgumentParser()
  >>> parser.add_argument('--foo', nargs='*')
  >>> parser.add_argument('--bar', nargs='*')
  >>> parser.add_argument('baz', nargs='*')
  >>> parser.parse_args('a b --foo x y --bar 1 2'.spl
  Namespace(bar=['1', '2'], baz=['a', 'b'], foo=['x',
  ```

- `'+'`. Just like `'*'`, all command-line arguments present are gathered into a list. Additionally, an error message will be generated if there wasn’t at least one command-line argument present. For example:

  ```
  >>> parser = argparse.ArgumentParser(prog='PROG')
  >>> parser.add_argument('foo', nargs='+')
  >>> parser.parse_args(['a', 'b'])
  Namespace(foo=['a', 'b'])
  >>> parser.parse_args([])
  usage: PROG [-h] foo [foo ...]
  PROG: error: the following arguments are required:
  ```

If the `nargs` keyword argument is not provided, the number of arguments consumed is determined by the action. Generally this means a single command-line argument will be consumed and a single item (not a list) will be produced. Actions that do not consume command-line arguments (e.g. `'store_const'`) set `nargs=0`.

const

El argumento `const` de **`add_argument()`** se usa para mantener valores constantes que no se leen desde la línea de comandos pero que son necesarios para las diversas acciones de **`ArgumentParser`**. Los dos usos más comunes son:

- Cuando **`add_argument()`** es llamado con `action='store_const'` o `action='append_const'`. Estas acciones añaden el valor `const` a uno de los atributos del objeto retornado por **`parse_args()`**. Mira la descripción action para ver ejemplos. Si `const` no es proporcionado a **`add_argument()`**, este recibirá el valor por defecto `None`.
- When **`add_argument()`** is called with option strings (like `-f` or `--foo`) and `nargs='?'`. This creates an optional argument that can be followed by zero or one command-line arguments. When parsing the command line, if the option string is encountered with no command-line argument following it, the value from `const` will be used. See the nargs description for examples.

Distinto en la versión 3.11: Por defecto `const=None`, incluyendo cuando `action='append_const'` o `action='store_const'`.

default

Todos los argumentos opcionales y algunos argumentos posicionales pueden ser omitidos en la línea de comandos. El argumento de palabra clave `default` de **`add_argument()`**, cuyo valor por defecto es `None`, especifica qué valor debe usarse si el argumento de línea de comandos no está presente. Para los argumentos opcionales, se usa el valor `default` cuando la cadena de caracteres de opción no está presente en la línea de comandos:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo', default=42)
>>> parser.parse_args(['--foo', '2'])
Namespace(foo='2')
>>> parser.parse_args([])
Namespace(foo=42)
```

If the target namespace already has an attribute set, the action *default* will not overwrite it:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo', default=42)
>>> parser.parse_args([], namespace=argparse.Namespace(f
Namespace(foo=101)
```

Si el valor `default` es una cadena de caracteres, el analizador procesa el valor como si fuera un argumento de la línea de comandos. En particular, el analizador aplica cualquier argumento de conversión type , si se proporciona, antes de establecer el atributo en el valor de retorno **`Namespace`**. En caso contrario, el analizador utiliza el valor tal y como es:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--length', default='10', type=i
>>> parser.add_argument('--width', default=10.5, type=ir
>>> parser.parse_args()
Namespace(length=10, width=10.5)
```

Para argumentos posiciones con nargs igual a `?` o `*`, el valor `default` se utiliza cuando no hay ningún argumento de línea de comandos presente:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('foo', nargs='?', default=42)
>>> parser.parse_args(['a'])
Namespace(foo='a')
>>> parser.parse_args([])
Namespace(foo=42)
```

For required arguments, the `default` value is ignored. For example, this applies to positional arguments with nargs values other than `?` or `*`, or optional arguments marked as `required=True`.

Proporcionar `default=argparse.SUPPRESS` causa que no se agregue ningún atributo si el argumento de la línea de comandos no está presente:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo', default=argparse.SUPPRE
>>> parser.parse_args([])
Namespace()
```

```
>>> parser.parse_args(['--foo', '1'])
Namespace(foo='1')
```

type

De forma predeterminada, el analizador lee los argumentos de la línea de comandos como cadenas simples. Sin embargo, a menudo, la cadena de la línea de comandos debe interpretarse como otro tipo, como **`float`** o **`int`**. La palabra clave `type` para **`add_argument()`** permite realizar cualquier verificación de tipo y conversión de tipo necesaria.

Si la palabra clave type se usa con la palabra clave default, el convertidor de tipos solo se aplica si el valor predeterminado es una cadena de caracteres.

The argument to `type` can be a callable that accepts a single string or the name of a registered type (see **`register()`**) If the function raises **`ArgumentTypeError`**, **`TypeError`**, or **`ValueError`**, the exception is caught and a nicely formatted error message is displayed. Other exception types are not handled.

Los tipos y funciones incorporados comunes se pueden utilizar como convertidores de tipos:

```
import argparse
import pathlib

parser = argparse.ArgumentParser()
parser.add_argument('count', type=int)
parser.add_argument('distance', type=float)
parser.add_argument('street', type=ascii)
parser.add_argument('code_point', type=ord)
parser.add_argument('datapath', type=pathlib.Path)
```

Las funciones definidas por el usuario también se pueden utilizar:

```
>>> def hyphenated(string):
...     return '-'.join([word[:4] for word in string.cas
...
```

```
>>> parser = argparse.ArgumentParser()
>>> _ = parser.add_argument('short_title', type=hyphenat
>>> parser.parse_args(['"The Tale of Two Cities"'])
Namespace(short_title='"the-tale-of-two-citi')
```

The **`bool()`** function is not recommended as a type converter. All it does is convert empty strings to `False` and non-empty strings to `True`. This is usually not what is desired:

```
>>> parser = argparse.ArgumentParser()
>>> _ = parser.add_argument('--verbose', type=bool)
>>> parser.parse_args(['--verbose', 'False'])
Namespace(verbose=True)
```

See **`BooleanOptionalAction`** or `action='store_true'` for common alternatives.

En general, la palabra clave `type` es una conveniencia que solo debe usarse para conversiones simples que solo pueden generar una de las tres excepciones admitidas. Cualquier cosa con un manejo de errores o de recursos más interesante debe hacerse en sentido descendente después de analizar los argumentos.

Por ejemplo, las conversiones JSON o YAML tienen casos de error complejos que requieren mejores informes que los que puede proporcionar la palabra clave `type`. Un **`JSONDecodeError`** no estaría bien formateado y un **`FileNotFoundError`** no se manejaría en absoluto.

Even **`FileType`** has its limitations for use with the `type` keyword. If one argument uses **`FileType`** and then a subsequent argument fails, an error is reported but the file is not automatically closed. In this case, it would be better to wait until after the parser has run and then use the **`with`**-statement to manage the files.

Para los verificadores de tipo que simplemente verifican un conjunto fijo de valores, considere usar la palabra clave choices en su lugar.

choices

Algunos argumentos de la línea de comandos deberían seleccionarse de un conjunto restringido de valores. Estos pueden ser manejados pasando un objeto de secuencia como el argumento de palabra clave *choices* a `add_argument()`. Cuando se analiza la línea de comandos, se comprueban los valores de los argumentos y se muestra un mensaje de error si el argumento no era uno de los valores aceptables:

```
>>> parser = argparse.ArgumentParser(prog='game.py')
>>> parser.add_argument('move', choices=['rock', 'paper'
>>> parser.parse_args(['rock'])
Namespace(move='rock')
>>> parser.parse_args(['fire'])
usage: game.py [-h] {rock,paper,scissors}
game.py: error: argument move: invalid choice: 'fire' (c
'paper', 'scissors')
```

Se puede pasar cualquier secuencia como el valor para *choices*, así que los objetos `list`, `tuple` , y las secuencias personalizadas están soportados.

No se recomienda el uso de `enum.Enum` porque es difícil controlar su apariencia en el uso, la ayuda y los mensajes de error.

Note that *choices* are checked after any type conversions have been performed, so objects in *choices* should match the type specified. This can make *choices* appear unfamiliar in usage, help, or error messages.

To keep *choices* user-friendly, consider a custom type wrapper that converts and formats values, or omit type and handle conversion in your application code.

Las opciones formateadas anulan el *metavar* predeterminado que normalmente se deriva de *dest*. Esto suele ser lo que se quiere porque el usuario nunca ve el parámetro *dest*. Si esta visualización no es deseable (quizás porque hay muchas opciones), simplemente especifique un metavar explícito.

required

In general, the **`argparse`** module assumes that flags like `-f` and `--bar` indicate *optional* arguments, which can always be omitted at the command line. To make an option *required*, `True` can be specified for the `required=` keyword argument to **`add_argument()`**:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo', required=True)
>>> parser.parse_args(['--foo', 'BAR'])
Namespace(foo='BAR')
>>> parser.parse_args([])
usage: [-h] --foo FOO
: error: the following arguments are required: --foo
```

Como muestra el ejemplo, si una opción está marcada como `required`, **`parse_args()`** informará de un error si esa opción no está presente en la línea de comandos.

Nota

Las opciones requeridas están consideradas generalmente mala práctica porque los usuarios esperan que las *opciones* sean *opcionales*, y por lo tanto deberían ser evitadas cuando sea posible.

help

The `help` value is a string containing a brief description of the argument. When a user requests help (usually by using `-h` or `--help` at the command line), these `help` descriptions will be displayed with each argument.

Las cadenas de texto `help` pueden incluir varios descriptores de formato para evitar la repetición de cosas como el nombre del programa o el argumento default. Los descriptores disponibles incluyen el nombre del programa, `%(prog)s` y la mayoría de los argumentos de palabra clave de **`add_argument()`**, por ejemplo `%(default)s`, `%(type)s`, etc.:

```
>>> parser = argparse.ArgumentParser(prog='frobble')
>>> parser.add_argument('bar', nargs='?', type=int, defa
...                         help='the bar to %(prog)s (defau
>>> parser.print_help()
usage: frobble [-h] [bar]

positional arguments:
 bar     the bar to frobble (default: 42)

options:
 -h, --help  show this help message and exit
```

Como la cadena de caracteres de ayuda soporta el formato-%, si se quiere que aparezca un `%` literal en la cadena de caracteres de ayuda, se debe escribir como `%%`.

argparse supports silencing the help entry for certain options, by setting the `help` value to `argparse.SUPPRESS`:

```
>>> parser = argparse.ArgumentParser(prog='frobble')
>>> parser.add_argument('--foo', help=argparse.SUPPRESS)
>>> parser.print_help()
usage: frobble [-h]

options:
  -h, --help  show this help message and exit
```

metavar

When **ArgumentParser** generates help messages, it needs some way to refer to each expected argument. By default, **ArgumentParser** objects use the dest value as the «name» of each object. By default, for positional argument actions, the dest value is used directly, and for optional argument actions, the dest value is uppercased. So, a single positional argument with `dest='bar'` will be referred to as `bar`. A single optional argument `--foo` that should be followed by a single command-line argument will be referred to as `FOO`. An example:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo')
>>> parser.add_argument('bar')
>>> parser.parse_args('X --foo Y'.split())
Namespace(bar='X', foo='Y')
>>> parser.print_help()
usage:  [-h] [--foo FOO] bar

positional arguments:
 bar

options:
 -h, --help  show this help message and exit
 --foo FOO
```

Un nombre alternativo se puede especificar con `metavar`:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo', metavar='YYY')
>>> parser.add_argument('bar', metavar='XXX')
>>> parser.parse_args('X --foo Y'.split())
Namespace(bar='X', foo='Y')
>>> parser.print_help()
usage:  [-h] [--foo YYY] XXX

positional arguments:
 XXX

options:
 -h, --help  show this help message and exit
 --foo YYY
```

Ten en cuenta que `metavar` sólo cambia el nombre *mostrado* - el nombre del atributo en el objeto **`parse_args()`** sigue estando determinado por el valor dest.

Diferentes valores de `nargs` pueden causar que *metavar* sea usado múltiples veces. Proporcionar una tupla a `metavar` especifica una visualización diferente para cada uno de los argumentos:

```
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> parser.add_argument('-x', nargs=2)
>>> parser.add_argument('--foo', nargs=2, metavar=('bar'
>>> parser.print_help()
usage: PROG [-h] [-x X X] [--foo bar baz]

options:
 -h, --help     show this help message and exit
 -x X X
 --foo bar baz
```

dest

La mayoría de las acciones **`ArgumentParser`** añaden algún valor como atributo del objeto retornado por **`parse_args()`**. El nombre de este atributo está determinado por el argumento de palabra clave `dest` de **`add_argument()`**. Para acciones de argumento posicional, se proporciona `dest` normalmente como primer argumento de **`add_argument()`**:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('bar')
>>> parser.parse_args(['XXX'])
Namespace(bar='XXX')
```

Para las acciones de argumentos opcionales, el valor de `dest` se infiere normalmente de las cadenas de caracteres de opción. **`ArgumentParser`** genera el valor de `dest` tomando la primera cadena de caracteres de opción larga y quitando la cadena inicial `--`. Si no se proporcionaron cadenas de caracteres de opción largas, `dest` se derivará de la primera cadena de caracteres de opción corta quitando el carácter `-`. Cualquier carácter `-` interno se convertirá a caracteres `_` para asegurarse de que la cadena de caracteres es un nombre de atributo válido. Los ejemplos siguientes ilustran este comportamiento:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('-f', '--foo-bar', '--foo')
>>> parser.add_argument('-x', '-y')
>>> parser.parse_args('-f 1 -x 2'.split())
Namespace(foo_bar='1', x='2')
```

```
>>> parser.parse_args('--foo 1 -y 2'.split())
Namespace(foo_bar='1', x='2')
```

`dest` permite que se proporcione un nombre de atributo personalizado:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo', dest='bar')
>>> parser.parse_args('--foo XXX'.split())
Namespace(bar='XXX')
```

deprecated

During a project's lifetime, some arguments may need to be removed from the command line. Before removing them, you should inform your users that the arguments are deprecated and will be removed. The `deprecated` keyword argument of **`add_argument()`**, which defaults to `False`, specifies if the argument is deprecated and will be removed in the future. For arguments, if `deprecated` is `True`, then a warning will be printed to **`sys.stderr`** when the argument is used:

```
>>> import argparse
>>> parser = argparse.ArgumentParser(prog='snake.py')
>>> parser.add_argument('--legs', default=0, type=int,
>>> parser.parse_args([])
Namespace(legs=0)
>>> parser.parse_args(['--legs', '4'])
snake.py: warning: option '--legs' is deprecated
Namespace(legs=4)
```

Added in version 3.13.

Las clases *Action*

`Action` classes implement the Action API, a callable which returns a callable which processes arguments from the command-line. Any object which follows this API may be passed as the `action` parameter to **`add_argument()`**.

class argparse.Action(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

`Action` objects are used by an **`ArgumentParser`** to represent the information needed to parse a single argument from one or more strings from the command line. The **`Action`** class must accept the two positional arguments plus any keyword arguments passed to **`ArgumentParser.add_argument()`** except for the `action` itself.

Instances of **`Action`** (or return value of any callable to the `action` parameter) should have attributes **`dest`**, **`option_strings`**, **`default`**, **`type`**, **`required`**, **`help`**, etc. defined. The easiest way to ensure these attributes are defined is to call **`Action.__init__()`**.

call(*parser*, *namespace*, *values*, *option_string=None*)

`Action` instances should be callable, so subclasses must override the **`__call__()`** method, which should accept four parameters:

- *parser* - The **`ArgumentParser`** object which contains this action.
- *namespace* - The **`Namespace`** object that will be returned by **`parse_args()`**. Most actions add an attribute to this object using **`setattr()`**.
- *values* - The associated command-line arguments, with any type conversions applied. Type conversions are specified with the type keyword argument to **`add_argument()`**.
- *option_string* - The option string that was used to invoke this action. The `option_string` argument is optional, and will be absent if the action is associated with a positional argument.

The **`__call__()`** method may perform arbitrary actions, but will typically set attributes on the `namespace` based on `dest` and `values`.

format_usage()

`Action` subclasses can define a **`format_usage()`** method that takes no argument and return a string which will be used when printing the usage of the program. If such method is not provided, a sensible default will be used.

class argparse.BooleanOptionalAction

A subclass of **`Action`** for handling boolean flags with positive and negative options. Adding a single argument such as `--foo` automatically creates both `--foo` and `--no-foo` options, storing `True` and `False` respectively:

```
>>> import argparse
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo', action=argparse.Bo
>>> parser.parse_args(['--no-foo'])
Namespace(foo=False)
```

Added in version 3.9.

El método *parse_args()*

ArgumentParser.parse_args(*args=None*, *namespace=None*)

Convierte las cadenas de caracteres de argumentos en objetos y los asigna como atributos del espacio de nombres (*namespace*). Retorna el espacio de nombres (*namespace*) ocupado.

Previous calls to **`add_argument()`** determine exactly what objects are created and how they are assigned. See the documentation for **`add_argument()`** for details.

- args - Lista de cadenas de caracteres para analizar. El valor por defecto se toma de **`sys.argv`**.
- namespace - Un objeto para obtener los atributos. Por defecto es un nuevo objeto vacío **`Namespace`**.

Sintaxis del valor de la opción

El método `parse_args()` soporta diversas formas de especificar el valor de una opción (si requiere uno). En el caso más simple, la opción y su valor se pasan como dos argumentos separados:

```
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> parser.add_argument('-x')
>>> parser.add_argument('--foo')
>>> parser.parse_args(['-x', 'X'])
Namespace(foo=None, x='X')
>>> parser.parse_args(['--foo', 'FOO'])
Namespace(foo='FOO', x=None)
```

En el caso de opciones largas (opciones con nombres más largos que un sólo carácter), la opción y el valor también se pueden pasar como un sólo argumento de línea de comandos, utilizando `=` para separarlos:

```
>>> parser.parse_args(['--foo=FOO'])
Namespace(foo='FOO', x=None)
```

Para las opciones cortas (opciones de un sólo carácter de largo), la opción y su valor pueden ser concatenados:

```
>>> parser.parse_args(['-xX'])
Namespace(foo=None, x='X')
```

Se pueden unir varias opciones cortas, usando un sólo prefijo `-`, siempre y cuando sólo la última opción (o ninguna de ellas) requiera un valor:

```
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> parser.add_argument('-x', action='store_true')
>>> parser.add_argument('-y', action='store_true')
>>> parser.add_argument('-z')
>>> parser.parse_args(['-xyzZ'])
Namespace(x=True, y=True, z='Z')
```

Argumentos no válidos

Mientras analiza la línea de comandos, `parse_args()` comprueba una variedad de errores, incluyendo opciones ambiguas, tipos no válidos, opciones no válidas, número incorrecto de argumentos de posición, etc. Cuando encuentra un error de este tipo, termina y muestra el error junto con un mensaje de uso:

```
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> parser.add_argument('--foo', type=int)
>>> parser.add_argument('bar', nargs='?')

>>> # invalid type
>>> parser.parse_args(['--foo', 'spam'])
usage: PROG [-h] [--foo FOO] [bar]
PROG: error: argument --foo: invalid int value: 'spam'

>>> # invalid option
>>> parser.parse_args(['--bar'])
usage: PROG [-h] [--foo FOO] [bar]
PROG: error: no such option: --bar

>>> # wrong number of arguments
>>> parser.parse_args(['spam', 'badger'])
usage: PROG [-h] [--foo FOO] [bar]
PROG: error: extra arguments found: badger
```

Argumentos conteniendo `-`

El método `parse_args()` busca generar errores cuando el usuario ha cometido claramente una equivocación, pero algunas situaciones son inherentemente ambiguas. Por ejemplo, el argumento de línea de comandos `-1` podría ser un intento de especificar una opción o un intento de proporcionar un argumento de posición. El método **`parse_args()`** es cauteloso aquí: los argumentos de posición sólo pueden comenzar con `-` si se ven como números negativos y no hay opciones en el analizador que se puedan ver como números negativos

```
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> parser.add_argument('-x')
>>> parser.add_argument('foo', nargs='?')

>>> # no negative number options, so -1 is a positional
>>> parser.parse_args(['-x', '-1'])
Namespace(foo=None, x='-1')

>>> # no negative number options, so -1 and -5 are posit
>>> parser.parse_args(['-x', '-1', '-5'])
Namespace(foo='-5', x='-1')

>>> parser = argparse.ArgumentParser(prog='PROG')
>>> parser.add_argument('-1', dest='one')
>>> parser.add_argument('foo', nargs='?')

>>> # negative number options present, so -1 is an optic
>>> parser.parse_args(['-1', 'X'])
Namespace(foo=None, one='X')

>>> # negative number options present, so -2 is an optic
>>> parser.parse_args(['-2'])
usage: PROG [-h] [-1 ONE] [foo]
PROG: error: no such option: -2

>>> # negative number options present, so both -1s are c
>>> parser.parse_args(['-1', '-1'])
usage: PROG [-h] [-1 ONE] [foo]
PROG: error: argument -1: expected one argument
```

Si tienes argumentos de posición que deben comenzar con `-` y no parecen números negativos, puedes insertar el pseudo-argumento `'--'` que indica a **`parse_args()`** que todo lo que sigue es un argumento de posición:

```
>>> parser.parse_args(['--', '-f'])
Namespace(foo='-f', one=None)
```

También ver la guía de argparse sobre cómo manejar argumentos ambiguos para más detalles.

Abreviaturas de los argumentos (coincidencia de prefijos)

El método `parse_args()` por defecto permite abreviar las opciones largas a un prefijo, si la abreviatura es inequívoca (el prefijo coincide con una opción única):

```
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> parser.add_argument('-bacon')
>>> parser.add_argument('-badger')
>>> parser.parse_args('-bac MMM'.split())
Namespace(bacon='MMM', badger=None)
>>> parser.parse_args('-bad WOOD'.split())
Namespace(bacon=None, badger='WOOD')
>>> parser.parse_args('-ba BA'.split())
usage: PROG [-h] [-bacon BACON] [-badger BADGER]
PROG: error: ambiguous option: -ba could match -badger,
```

Se incurre en un error por argumentos que podrían derivar en más de una opción. Esta característica puede ser desactivada poniendo allow_abbrev a `False`.

Más allá de `sys.argv`

Sometimes it may be useful to have an `ArgumentParser` parse arguments other than those of `sys.argv`. This can be accomplished by passing a list of strings to `parse_args()`. This is useful for testing at the interactive prompt:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument(
...     'integers', metavar='int', type=int, choices=ran
...     nargs='+', help='an integer in the range 0..9')
>>> parser.add_argument(
...     '--sum', dest='accumulate', action='store_const'
...     default=max, help='sum the integers (default: fi
>>> parser.parse_args(['1', '2', '3', '4'])
Namespace(accumulate=<built-in function max>, integers=[
>>> parser.parse_args(['1', '2', '3', '4', '--sum'])
```

```
Namespace(accumulate=<built-in function sum>, integers=[
```

El objeto *Namespace*

class argparse.Namespace

Clase simple utilizada por defecto por **`parse_args()`** para crear un objeto que contenga atributos y retornarlo.

Esta clase es deliberadamente simple, sólo una subclase **`object`** con una representación de cadena de texto legible. Si prefieres tener una vista en forma de diccionario de los atributos, puedes usar el lenguaje estándar de Python, **`vars()`**:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo')
>>> args = parser.parse_args(['--foo', 'BAR'])
>>> vars(args)
{'foo': 'BAR'}
```

También puede ser útil tener un **`ArgumentParser`** que asigne atributos a un objeto ya existente, en lugar de un nuevo objeto **`Namespace`**. Esto se puede lograr especificando el argumento de palabra clave `namespace=`:

```
>>> class C:
...     pass
...
>>> c = C()
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo')
>>> parser.parse_args(args=['--foo', 'BAR'], namesp
>>> c.foo
'BAR'
```

Otras utilidades

Subcommands

ArgumentParser.add_subparsers(*[, *title*] [, *description*] [, *prog*] [, *parser_class*] [, *action*] [, *dest*] [, *required*] [, *help*] [, *metavar*])

Many programs split up their functionality into a number of subcommands, for example, the `svn` program can invoke subcommands like `svn checkout`, `svn update`, and `svn commit`. Splitting up functionality this way can be a particularly good idea when a program performs several different functions which require different kinds of command-line arguments. **`ArgumentParser`** supports the creation of such subcommands with the **`add_subparsers()`** method. The **`add_subparsers()`** method is normally called with no arguments and returns a special action object. This object has a single method, **`add_parser()`**, which takes a command name and any **`ArgumentParser`** constructor arguments, and returns an **`ArgumentParser`** object that can be modified as usual.

Descripción de los parámetros:

- *title* - title for the sub-parser group in help output; by default «subcommands» if description is provided, otherwise uses title for positional arguments
- *description* - description for the sub-parser group in help output, by default `None`
- *prog* - usage information that will be displayed with subcommand help, by default the name of the program and any positional arguments before the subparser argument
- *parser_class* - class which will be used to create sub-parser instances, by default the class of the current parser (e.g. **`ArgumentParser`**)
- action - el tipo básico de acción a tomar cuando este argumento se encuentre en la línea de comandos
- dest - name of the attribute under which subcommand name will be stored; by default `None` and no value is stored
- required - Si se debe proporcionar o no un sub-comando, por defecto `False` (añadido en 3.7)
- help - ayuda para el grupo de análisis secundario en la

salida de la ayuda, por defecto `None`
- metavar - string presenting available subcommands in help; by default it is `None` and presents subcommands in form {cmd1, cmd2, ..}

Algún ejemplo de uso:

```
>>> # create the top-level parser
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> parser.add_argument('--foo', action='store_true
>>> subparsers = parser.add_subparsers(help='subcom
>>>
>>> # create the parser for the "a" command
>>> parser_a = subparsers.add_parser('a', help='a h
>>> parser_a.add_argument('bar', type=int, help='ba
>>>
>>> # create the parser for the "b" command
>>> parser_b = subparsers.add_parser('b', help='b h
>>> parser_b.add_argument('--baz', choices=('X', 'Y
>>>
>>> # parse some argument lists
>>> parser.parse_args(['a', '12'])
Namespace(bar=12, foo=False)
>>> parser.parse_args(['--foo', 'b', '--baz', 'Z'])
Namespace(baz='Z', foo=True)
```

Ten en cuenta que el objeto retornado por **`parse_args()`** sólo contendrá atributos para el analizador principal y el analizador secundario que fue seleccionado por la línea de comandos (y no cualquier otro analizador secundario). Así que en el ejemplo anterior, cuando se especifica el comando `a`, sólo están presentes los atributos `foo` y `bar`, y cuando se especifica el comando `b`, sólo están presentes los atributos `foo` y `baz`.

Del mismo modo, cuando se solicita un mensaje de ayuda de un analizador secundario, sólo se imprimirá la ayuda para ese analizador en particular. El mensaje de ayuda no incluirá mensajes del analizador principal o de analizadores relacionados. (Sin embargo, se puede dar un mensaje de ayuda para cada comando del analizador secundario suministrando el argumento `help=` a **`add_parser()`** como se ha indicado anteriormente).

```
>>> parser.parse_args(['--help'])
usage: PROG [-h] [--foo] {a,b} ...

positional arguments:
  {a,b}   subcommand help
    a     a help
    b     b help

options:
  -h, --help  show this help message and exit
  --foo   foo help

>>> parser.parse_args(['a', '--help'])
usage: PROG a [-h] bar

positional arguments:
  bar     bar help

options:
  -h, --help  show this help message and exit

>>> parser.parse_args(['b', '--help'])
usage: PROG b [-h] [--baz {X,Y,Z}]

options:
  -h, --help     show this help message and exit
  --baz {X,Y,Z}  baz help
```

El método **`add_subparsers()`** también soporta los argumentos de palabra clave `title` y `description`. Cuando cualquiera de los dos esté presente, los comandos del analizador secundario aparecerán en su propio grupo en la salida de la ayuda. Por ejemplo:

```
>>> parser = argparse.ArgumentParser()
>>> subparsers = parser.add_subparsers(title='subco
...                                     description=
...                                     help='additi
>>> subparsers.add_parser('foo')
>>> subparsers.add_parser('bar')
>>> parser.parse_args(['-h'])
usage:  [-h] {foo,bar} ...

options:
  -h, --help  show this help message and exit

subcommands:
  valid subcommands

  {foo,bar}   additional help
```

Furthermore, **`add_parser()`** supports an additional *aliases* argument, which allows multiple strings to refer to the same subparser. This example, like `svn`, aliases `co` as a shorthand for `checkout`:

```
>>> parser = argparse.ArgumentParser()
>>> subparsers = parser.add_subparsers()
>>> checkout = subparsers.add_parser('checkout', al
>>> checkout.add_argument('foo')
>>> parser.parse_args(['co', 'bar'])
Namespace(foo='bar')
```

`add_parser()` supports also an additional *deprecated* argument, which allows to deprecate the subparser.

```
>>> import argparse
>>> parser = argparse.ArgumentParser(prog='chicken.
>>> subparsers = parser.add_subparsers()
>>> run = subparsers.add_parser('run')
>>> fly = subparsers.add_parser('fly', deprecated=T
>>> parser.parse_args(['fly'])
chicken.py: warning: command 'fly' is deprecated
Namespace()
```

Added in version 3.13.

One particularly effective way of handling subcommands is to combine the use of the **`add_subparsers()`** method with calls to `set_defaults()` so that each subparser knows which Python function it should execute. For example:

```
>>> # subcommand functions
>>> def foo(args):
...     print(args.x * args.y)
...
>>> def bar(args):
...     print('((%s))' % args.z)
...
>>> # create the top-level parser
>>> parser = argparse.ArgumentParser()
>>> subparsers = parser.add_subparsers(required=Tru
>>>
>>> # create the parser for the "foo" command
>>> parser_foo = subparsers.add_parser('foo')
>>> parser_foo.add_argument('-x', type=int, default
>>> parser_foo.add_argument('y', type=float)
>>> parser_foo.set_defaults(func=foo)
>>>
>>> # create the parser for the "bar" command
>>> parser_bar = subparsers.add_parser('bar')
>>> parser_bar.add_argument('z')
>>> parser_bar.set_defaults(func=bar)
>>>
>>> # parse the args and call whatever function was
>>> args = parser.parse_args('foo 1 -x 2'.split())
>>> args.func(args)
2.0
>>>
>>> # parse the args and call whatever function was
>>> args = parser.parse_args('bar XYZYX'.split())
>>> args.func(args)
((XYZYX))
```

De esta manera, puedes dejar que **`parse_args()`** haga el trabajo de llamar a la función apropiada después de que el análisis de los argumentos se haya completado. Asociar funciones con acciones como esta es típicamente la forma más fácil de manejar las diferentes acciones para cada uno de tus analizadores secundarios. Sin embargo, si es necesario comprobar el nombre del analizador secundario que se ha invocado, el argumento de palabra clave `dest` a la llamada **`add_subparsers()`** hará el trabajo:

```
>>> parser = argparse.ArgumentParser()
>>> subparsers = parser.add_subparsers(dest='subpar
>>> subparser1 = subparsers.add_parser('1')
>>> subparser1.add_argument('-x')
>>> subparser2 = subparsers.add_parser('2')
>>> subparser2.add_argument('y')
>>> parser.parse_args(['2', 'frobble'])
Namespace(subparser_name='2', y='frobble')
```

Distinto en la versión 3.7: New *required* keyword-only parameter.

Distinto en la versión 3.14: Subparser's *prog* is no longer affected by a custom usage message in the main parser.

Objetos *FileType*

class argparse.FileType(*mode='r'*, *bufsize=-1*, *encoding=None*, *errors=None*)

El generador **`FileType`** crea objetos que pueden ser transferidos al argumento tipo de **`ArgumentParser.add_argument()`**. Los argumentos que tienen objetos **`FileType`** como su tipo abrirán los argumentos de líneas de comandos como archivos con los modos, tamaños de búfer, codificaciones y manejo de errores solicitados (véase la función **`open()`** para más detalles):

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--raw', type=argparse.File
>>> parser.add_argument('out', type=argparse.FileTy
>>> parser.parse_args(['--raw', 'raw.dat', 'file.tx
Namespace(out=<_io.TextIOWrapper name='file.txt' mo
```

Los objetos *FileType* entienden el pseudo-argumento `'-'` y lo convierten automáticamente en `sys.stdin` para objetos de lectura `FileType` y `sys.stdout` para objetos de escritura **`FileType`**:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('infile', type=argparse.Fil
>>> parser.parse_args(['-'])
Namespace(infile=<_io.TextIOWrapper name='<stdin>'
```

Nota

If one argument uses *FileType* and then a subsequent argument fails, an error is reported but the file is not automatically closed. This can also clobber the output files. In this case, it would be better to wait until after the parser has run and then use the `with`-statement to manage the files.

Distinto en la versión 3.4: Added the *encoding* and *errors* parameters.

Obsoleto desde la versión 3.14.

Grupos de argumentos

ArgumentParser.add_argument_group(*title=None*, *description=None*, *[, *argument_default*] [, *conflict_handler*])

By default, `ArgumentParser` groups command-line arguments into «positional arguments» and «options» when displaying help messages. When there is a better conceptual grouping of arguments than this default one, appropriate groups can be created using the **`add_argument_group()`** method:

```
>>> parser = argparse.ArgumentParser(prog='PROG', a
>>> group = parser.add_argument_group('group')
>>> group.add_argument('--foo', help='foo help')
>>> group.add_argument('bar', help='bar help')
>>> parser.print_help()
usage: PROG [--foo FOO] bar

group:
  bar        bar help
  --foo FOO  foo help
```

The **`add_argument_group()`** method returns an argument group object which has an **`add_argument()`** method just like a regular **`ArgumentParser`**. When an argument is added to the group, the parser treats it just like a normal argument, but displays the argument in a separate group for help messages. The **`add_argument_group()`** method accepts *title* and *description* arguments which can be used to customize this display:

```
>>> parser = argparse.ArgumentParser(prog='PROG', a
>>> group1 = parser.add_argument_group('group1', 'g
>>> group1.add_argument('foo', help='foo help')
>>> group2 = parser.add_argument_group('group2', 'g
>>> group2.add_argument('--bar', help='bar help')
>>> parser.print_help()
usage: PROG [--bar BAR] foo

group1:
  group1 description

  foo        foo help

group2:
  group2 description

  --bar BAR  bar help
```

The optional, keyword-only parameters argument_default and

conflict_handler allow for finer-grained control of the behavior of the argument group. These parameters have the same meaning as in the `ArgumentParser` constructor, but apply specifically to the argument group rather than the entire parser.

Ten en cuenta que cualquier argumento que no esté en los grupos definidos por el usuario terminará en las secciones habituales de «argumentos de posición» y «argumentos opcionales».

Within each argument group, arguments are displayed in help output in the order in which they are added.

Deprecated since version 3.11, removed in version 3.14: Calling **`add_argument_group()`** on an argument group now raises an exception. This nesting was never supported, often failed to work correctly, and was unintentionally exposed through inheritance.

Obsoleto desde la versión 3.14: Passing prefix_chars to **`add_argument_group()`** is now deprecated.

Exclusión mutua

ArgumentParser.add_mutually_exclusive_group(*required=False*)

Create a mutually exclusive group. **`argparse`** will make sure that only one of the arguments in the mutually exclusive group was present on the command line:

```
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> group = parser.add_mutually_exclusive_group()
>>> group.add_argument('--foo', action='store_true'
>>> group.add_argument('--bar', action='store_false
>>> parser.parse_args(['--foo'])
Namespace(bar=True, foo=True)
>>> parser.parse_args(['--bar'])
Namespace(bar=False, foo=False)
>>> parser.parse_args(['--foo', '--bar'])
usage: PROG [-h] [--foo | --bar]
```

```
PROG: error: argument --bar: not allowed with argum
```

El método **add_mutually_exclusive_group()** también acepta un argumento *required*, para indicar que se requiere al menos uno de los argumentos mutuamente exclusivos:

```
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> group = parser.add_mutually_exclusive_group(req
>>> group.add_argument('--foo', action='store_true'
>>> group.add_argument('--bar', action='store_false
>>> parser.parse_args([])
usage: PROG [-h] (--foo | --bar)
PROG: error: one of the arguments --foo --bar is re
```

Tomar en cuenta que actualmente los grupos de argumentos mutuamente exclusivos no admiten los argumentos *title* y *description* de **add_argument_group()**. Sin embargo, se puede agregar un grupo mutuamente exclusivo a un grupo de argumentos que tenga un título y una descripción. Por ejemplo:

```
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> group = parser.add_argument_group('Group title'
>>> exclusive_group = group.add_mutually_exclusive_
>>> exclusive_group.add_argument('--foo', help='foo
>>> exclusive_group.add_argument('--bar', help='bar
>>> parser.print_help()
usage: PROG [-h] (--foo FOO | --bar BAR)

options:
  -h, --help  show this help message and exit

Group title:
  Group description

  --foo FOO   foo help
  --bar BAR   bar help
```

Deprecated since version 3.11, removed in version 3.14: Calling **add_argument_group()** or

`add_mutually_exclusive_group()` on a mutually exclusive group now raises an exception. This nesting was never supported, often failed to work correctly, and was unintentionally exposed through inheritance.

Valores por defecto del analizador

ArgumentParser.set_defaults(***kwargs*)

La mayoría de las veces, los atributos del objeto retornado por **`parse_args()`** se determinarán completamente inspeccionando los argumentos de la línea de comandos y las acciones de los argumentos. **`set_defaults()`** permite que se añadan algunos atributos adicionales que se determinan sin ninguna inspección de la línea de comandos:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('foo', type=int)
>>> parser.set_defaults(bar=42, baz='badger')
>>> parser.parse_args(['736'])
Namespace(bar=42, baz='badger', foo=736)
```

Note that defaults can be set at both the parser level using **`set_defaults()`** and at the argument level using **`add_argument()`**. If both are called for the same argument, the last default set for an argument is used:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo', default='bar')
>>> parser.set_defaults(foo='spam')
>>> parser.parse_args([])
Namespace(foo='spam')
```

Los valores por defecto a nivel analizador pueden ser muy útiles cuando se trabaja con varios analizadores. Consulta el método **`add_subparsers()`** para ver un ejemplo de este tipo.

ArgumentParser.get_default(*dest*)

Obtiene el valor por defecto para un atributo del espacio de nombres (*namespace*), establecido ya sea por `add_argument()` o por `set_defaults()`:

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo', default='badger')
>>> parser.get_default('foo')
'badger'
```

Mostrando la ayuda

En la mayoría de las aplicaciones típicas, `parse_args()` se encargará de dar formato y mostrar cualquier mensaje de uso o de error. Sin embargo, hay varios métodos para dar formato disponibles:

ArgumentParser.print_usage(*file=None*)

> Muestra una breve descripción de cómo se debe invocar el `ArgumentParser` en la línea de comandos. Si *file* es `None`, se asume `sys.stdout`.

ArgumentParser.print_help(*file=None*)

> Muestra un mensaje de ayuda, incluyendo el uso del programa e información sobre los argumentos registrados en el `ArgumentParser`. Si *file* es `None`, se asume `sys.stdout`.

También hay variantes de estos métodos que simplemente retornan una cadena de caracteres en lugar de mostrarla:

ArgumentParser.format_usage()

> Retorna una cadena de caracteres que contiene una breve descripción de cómo se debe invocar el `ArgumentParser` en la línea de comandos.

ArgumentParser.format_help()

> Retorna una cadena de caracteres que contiene un mensaje de ayuda, incluyendo el uso del programa e información sobre los argumentos registrados en el `ArgumentParser`.

Análisis parcial

ArgumentParser.parse_known_args(*args*=*None*, *namespace*=*None*)

Sometimes a script only needs to handle a specific set of command-line arguments, leaving any unrecognized arguments for another script or program. In these cases, the **`parse_known_args()`** method can be useful.

This method works similarly to **`parse_args()`**, but it does not raise an error for extra, unrecognized arguments. Instead, it parses the known arguments and returns a two item tuple that contains the populated namespace and the list of any unrecognized arguments.

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo', action='store_true
>>> parser.add_argument('bar')
>>> parser.parse_known_args(['--foo', '--badger', '
(Namespace(bar='BAR', foo=True), ['--badger', 'spam
```

Advertencia

Coincidencia de prefijos las reglas se aplican a **`parse_known_args()`**. El analizador puede consumir una opción aunque sea sólo un prefijo de una de sus opciones conocidas, en lugar de dejarla en la lista de argumentos restantes.

Personalizando el análisis de archivos

ArgumentParser.convert_arg_line_to_args(*arg_line*)

Los argumentos que se leen de un archivo (mira el argumento de palabra clave *fromfile_prefix_chars* para el constructor **`ArgumentParser`**) se leen uno por línea. **`convert_arg_line_to_args()`** puede ser invalidado para una lectura más elegante.

Este método utiliza un sólo argumento *arg_line* que es una cadena de caracteres leída desde el archivo de argumentos. Retorna una lista de argumentos analizados a partir de esta cadena de caracteres. El método se llama una vez por línea leída del fichero de argumentos, en orden.

Una alternativa útil de este método es la que trata cada palabra separada por un espacio como un argumento. El siguiente ejemplo demuestra cómo hacerlo:

```
class MyArgumentParser(argparse.ArgumentParser):
    def convert_arg_line_to_args(self, arg_line):
        return arg_line.split()
```

Métodos de salida

ArgumentParser.exit(*status=0*, *message=None*)

This method terminates the program, exiting with the specified *status* and, if given, it prints a *message* to `sys.stderr` before that. The user can override this method to handle these steps differently:

```
class ErrorCatchingArgumentParser(argparse.Argument
    def exit(self, status=0, message=None):
        if status:
            raise Exception(f'Exiting because of an
        exit(status)
```

ArgumentParser.error(*message*)

This method prints a usage message, including the *message*, to `sys.stderr` and terminates the program with a status code of 2.

Análisis entremezclado

ArgumentParser.parse_intermixed_args(*args=None*, *namespace=None*)

ArgumentParser.parse_known_intermixed_args(*args=None*, *namespace=None*)

Una serie de comandos *Unix* permiten al usuario mezclar argumentos opcionales con argumentos de posición. Los métodos **`parse_intermixed_args()`** y **`parse_known_intermixed_args()`** soportan este modo de análisis.

These parsers do not support all the **`argparse`** features, and will raise exceptions if unsupported features are used. In particular, subparsers, and mutually exclusive groups that include both optionals and positionals are not supported.

El siguiente ejemplo muestra la diferencia entre **`parse_known_args()`** y **`parse_intermixed_args()`**: el primero retorna `['2', '3']` como argumentos sin procesar, mientras que el segundo recoge todos los de posición en `rest`.

```
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('--foo')
>>> parser.add_argument('cmd')
>>> parser.add_argument('rest', nargs='*', type=int
>>> parser.parse_known_args('doit 1 --foo bar 2 3'.
(Namespace(cmd='doit', foo='bar', rest=[1]), ['2',
>>> parser.parse_intermixed_args('doit 1 --foo bar
Namespace(cmd='doit', foo='bar', rest=[1, 2, 3])
```

`parse_known_intermixed_args()` retorna una tupla de dos elementos que contiene el espacio de nombres poblado y la lista de los restantes argumentos de cadenas de caracteres. **`parse_intermixed_args()`** arroja un error si quedan argumentos de cadenas de caracteres sin procesar.

Added in version 3.7.

Registering custom types or actions

ArgumentParser.register(*registry_name*, *value*, *object*)

Sometimes it’s desirable to use a custom string in error messages to provide more user-friendly output. In these cases, **`register()`** can be used to register custom actions or types with a parser and allow you to reference the type by their registered name instead of their callable name.

The **`register()`** method accepts three arguments - a *registry_name*, specifying the internal registry where the object will be stored (e.g., `action`, `type`), *value*, which is the key under which the object will be registered, and object, the callable to be registered.

The following example shows how to register a custom type with a parser:

```
>>> import argparse
>>> parser = argparse.ArgumentParser()
>>> parser.register('type', 'hexadecimal integer',
>>> parser.add_argument('--foo', type='hexadecimal
_StoreAction(option_strings=['--foo'], dest='foo',
>>> parser.parse_args(['--foo', '0xFA'])
Namespace(foo=250)
>>> parser.parse_args(['--foo', '1.2'])
usage: PROG [-h] [--foo FOO]
PROG: error: argument --foo: invalid 'hexadecimal i
```

Excepciones

exception argparse.ArgumentError

Un error al crear o usar un argumento (opcional o posicional).

El valor de la cadena de caracteres de esta excepción es el mensaje, ampliado con información sobre el argumento que lo causó.

exception argparse.ArgumentTypeError

Se lanza cuando algo sale mal al convertir una cadena de caracteres de la línea de comandos a un tipo.

Guides and Tutorials

- Tutorial de *argparse*
- Migrating **`optparse`** code to **`argparse`**

Tutorial de *argparse*

autor:

Tshepang Mbambo

Este tutorial pretende ser una leve introducción a **`argparse`**, el módulo de análisis (*parsing*) de línea de comandos recomendado en la biblioteca estándar de Python.

Nota

The standard library includes two other libraries directly related to command-line parameter processing: the lower level **`optparse`** module (which may require more code to configure for a given application, but also allows an application to request behaviors that `argparse` doesn't support), and the very low level **`getopt`** (which specifically serves as an equivalent to the **`getopt()`** family of functions available to C programmers). While neither of those modules is covered directly in this guide, many of the core concepts in `argparse` first originated in `optparse`, so some aspects of this tutorial will also be relevant to `optparse` users.

Conceptos

Vamos a mostrar el tipo de funcionalidad que vamos a explorar en este tutorial introductorio haciendo uso del comando **ls**:

```
$ ls
cpython  devguide  prog.py  pypy  rm-unused-function.pat
$ ls pypy
ctypes_configure  demo  dotviewer  include  lib_pypy  li
$ ls -l
total 20
```

```
drwxr-xr-x 19 wena wena 4096 Feb 18 18:51 cpython
drwxr-xr-x  4 wena wena 4096 Feb  8 12:04 devguide
-rwxr-xr-x  1 wena wena  535 Feb 19 00:05 prog.py
drwxr-xr-x 14 wena wena 4096 Feb  7 00:59 pypy
-rw-r--r--  1 wena wena  741 Feb 18 01:01 rm-unused-func
$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory
Sort entries alphabetically if none of -cftuvSUX nor --s
...
```

Algunos conceptos que podemos aprender de los cuatro comandos:

- El comando **ls** es útil cuando se ejecuta sin ninguna opción en absoluto. Por defecto muestra el contenido del directorio actual.
- Si queremos hacer algo, mas allá de lo que provee por defecto, le contamos un poco mas. En este caso, queremos mostrar un directorio diferente, `pypy`. Lo que hicimos fue especificar lo que se conoce como argumento posicional. Se llama así porque el programa debe saber que hacer con ese valor, basado únicamente en función de donde aparece en la línea de comandos. Este concepto es mas relevante para un comando como **cp**, cuyo uso mas básico es `cp SRC DEST`. La primer posición es *lo que quieres copiar*, y la segunda posición es *a donde lo quieres copiar*.
- Ahora, digamos que queremos cambiar el comportamiento del programa. En nuestro ejemplo, mostramos mas información para cada archivo en lugar de solo mostrar los nombres de los archivos. El argumento `-l` en ese caso se conoce como argumento opcional.
- Este es un fragmento del texto de ayuda. Es muy útil porque puedes encontrar un programa que nunca has usado antes, y puedes darte cuenta de como funciona simplemente leyendo el texto de ayuda.

Las bases

Comencemos con un simple ejemplo, el cual no hace (casi) nada:

```
import argparse
parser = argparse.ArgumentParser()
parser.parse_args()
```

Lo siguiente es el resultado de ejecutar el código:

```
$ python prog.py
$ python prog.py --help
usage: prog.py [-h]

options:
  -h, --help  show this help message and exit
$ python prog.py --verbose
usage: prog.py [-h]
prog.py: error: unrecognized arguments: --verbose
$ python prog.py foo
usage: prog.py [-h]
prog.py: error: unrecognized arguments: foo
```

Esto es lo que está pasando:

- Ejecutar el script sin ninguna opción da como resultado que no se muestra nada en *stdout*. No es tan útil.
- El segundo comienza a mostrar la utilidad del módulo **`argparse`**. No hemos hecho casi nada, pero ya recibimos un buen mensaje de ayuda.
- La opción `--help`, que también puede ser abreviada como `-h`, es la única opción que tenemos gratis (es decir, no necesitamos especificarla). Especificar cualquier otra cosa da como resultado un error. Pero aún así, recibimos un mensaje útil, también gratis.

Introducción a los argumentos posicionales

Un ejemplo:

```
import argparse
parser = argparse.ArgumentParser()
```

```
parser.add_argument("echo")
args = parser.parse_args()
print(args.echo)
```

Y ejecutando el código:

```
$ python prog.py
usage: prog.py [-h] echo
prog.py: error: the following arguments are required: ec
$ python prog.py --help
usage: prog.py [-h] echo

positional arguments:
  echo

options:
  -h, --help  show this help message and exit
$ python prog.py foo
foo
```

Aquí está lo que está sucediendo:

- Hemos agregado el método **`add_argument()`**, el cual es el que usamos para especificar cuales opciones de la línea de comandos el programa está dispuesto a aceptar. En este caso, lo he llamado `echo` para que concuerde con su función.
- Llamar nuestro programa ahora requiere que especifiquemos una opción.
- El método **`parse_args()`** retorna de hecho algunos datos de las opciones especificadas, en este caso, `echo`.
- La variable es una forma de "magia" que **`argparse`** se realiza de forma gratuita (es decir, no es necesario especificar en qué variable se almacena ese valor). También notará que su nombre coincide con el argumento de cadena dado al método, `echo`.

Sin embargo, tenga en cuenta que, aunque la pantalla de ayuda luce bien y todo, en realidad no es tan útil como podría ser. Por ejemplo, vemos que tenemos `echo` como un argumento posicional, pero no sabemos lo que hace, de otra manera que no sea adivinar o leer el código fuente. Entonces, vamos a hacerlo un poco mas útil:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("echo", help="echo the string you us
args = parser.parse_args()
print(args.echo)
```

Y la salida:

```
$ python prog.py -h
usage: prog.py [-h] echo

positional arguments:
  echo        echo the string you use here

options:
  -h, --help  show this help message and exit
```

Ahora, que tal si hacemos algo más útil:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", help="display a square of
args = parser.parse_args()
print(args.square**2)
```

Lo siguiente es el resultado de ejecutar el código:

```
$ python prog.py 4
Traceback (most recent call last):
  File "prog.py", line 5, in <module>
    print(args.square**2)
TypeError: unsupported operand type(s) for ** or pow():
```

Eso no fue tan bien. Esto es porque **`argparse`** trata las opciones que le damos como cadenas, a menos que le digamos otra cosa. Entonces, vamos a llamar a **`argparse`** para tratar esa entrada como un entero:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", help="display a square of
                    type=int)
args = parser.parse_args()
print(args.square**2)
```

Lo siguiente es el resultado de ejecutar el código:

```
$ python prog.py 4
16
$ python prog.py four
usage: prog.py [-h] square
prog.py: error: argument square: invalid int value: 'fou
```

Eso fue bien. El programa ahora aún se cierra útilmente en caso de una entrada ilegal incorrecta antes de proceder.

Introducción a los argumentos opcionales

Hasta ahora hemos estado jugando con argumentos posicionales. Vamos a darle una mirada a como agregar los opcionales:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("--verbosity", help="increase output
args = parser.parse_args()
if args.verbosity:
    print("verbosity turned on")
```

Y la salida:

```
$ python prog.py --verbosity 1
verbosity turned on
$ python prog.py
$ python prog.py --help
usage: prog.py [-h] [--verbosity VERBOSITY]

options:
  -h, --help            show this help message and exit
  --verbosity VERBOSITY
                        increase output verbosity
$ python prog.py --verbosity
usage: prog.py [-h] [--verbosity VERBOSITY]
prog.py: error: argument --verbosity: expected one argum
```

Esto es lo que está pasando:

- El programa está escrito para mostrar algo cuando `--verbosity` sea especificado y no mostrar nada cuando no.
- Para mostrar que la opción es de hecho opcional, no hay ningún error al ejecutar el programa sin ella. Tenga en cuenta que por defecto, si un argumento opcional no es usado, la variable relevante, en este caso `args.verbosity`, se le da `None` como valor, razón por la cual falla la prueba de verdad del **`if`**.
- El mensaje de ayuda es un poco diferente.
- Cuando usamos la opción `--verbosity`, también se debe especificar un valor, cualquier valor.

El ejemplo anterior acepta arbitrariamente valores enteros para `--verbosity`, pero para nuestro simple programa, solo dos valores son realmente útiles, `True` o `False`. Modifiquemos el código de acuerdo a esto:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("--verbose", help="increase output v
                    action="store_true")
args = parser.parse_args()
if args.verbose:
    print("verbosity turned on")
```

Y la salida:

```
$ python prog.py --verbose
verbosity turned on
$ python prog.py --verbose 1
usage: prog.py [-h] [--verbose]
```

```
prog.py: error: unrecognized arguments: 1
$ python prog.py --help
usage: prog.py [-h] [--verbose]

options:
  -h, --help  show this help message and exit
  --verbose   increase output verbosity
```

Esto es lo que está pasando:

- La opción ahora es más una bandera que algo que requiere un valor. Incluso cambiamos el nombre de la opción para que coincida con esa idea. Tenga en cuenta que ahora especificamos una nueva palabra clave, `action`, y le dimos el valor `"store_true"`. Esto significa que, si la opción es especificada, se asigna el valor `True` a `args.verbose`. No especificarlo implica `False`.
- Se queja cuando se especifica un valor, en verdadero espíritu de lo que realmente son los flags.
- Observe los diferentes textos de ayuda.

Opciones cortas

Si estas familiarizado con el uso de la línea de comandos, podrás observar que aún no he tocado el tema de las versiones cortas de las opciones. Es bastante simple:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("-v", "--verbose", help="increase ou
                    action="store_true")
args = parser.parse_args()
if args.verbose:
    print("verbosity turned on")
```

Y aquí va:

```
$ python prog.py -v
verbosity turned on
$ python prog.py --help
```

```
usage: prog.py [-h] [-v]

options:
  -h, --help     show this help message and exit
  -v, --verbose  increase output verbosity
```

Tenga en cuenta que la nueva habilidad es también reflejada en el texto de ayuda.

Combinar argumentos opcionales y posicionales

Nuestro programa sigue creciendo en complejidad:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given nu
parser.add_argument("-v", "--verbose", action="store_tru
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbose:
    print(f"the square of {args.square} equals {answer}"
else:
    print(answer)
```

Y ahora la salida:

```
$ python prog.py
usage: prog.py [-h] [-v] square
prog.py: error: the following arguments are required: sq
$ python prog.py 4
16
$ python prog.py 4 --verbose
the square of 4 equals 16
$ python prog.py --verbose 4
the square of 4 equals 16
```

- Hemos traído de vuelta un argumento posicional, de ahí la queja.
- Tenga en cuenta que el orden no importa.

Que tal si le retornamos a nuestro programa la capacidad de tener múltiples valores de verbosidad, y realmente usarlos:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given nu
parser.add_argument("-v", "--verbosity", type=int,
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity == 2:
    print(f"the square of {args.square} equals {answer}"
elif args.verbosity == 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

Y la salida:

```
$ python prog.py 4
16
$ python prog.py 4 -v
usage: prog.py [-h] [-v VERBOSITY] square
prog.py: error: argument -v/--verbosity: expected one ar
$ python prog.py 4 -v 1
4^2 == 16
$ python prog.py 4 -v 2
the square of 4 equals 16
$ python prog.py 4 -v 3
16
```

Todos estos se ven bien, excepto el último, que expone un error en nuestro programa. Corrijamos esto restringiendo los valores que la opción `--verbosity` puede aceptar:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given nu
parser.add_argument("-v", "--verbosity", type=int, choic
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity == 2:
    print(f"the square of {args.square} equals {answer}"
elif args.verbosity == 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

Y la salida:

```
$ python prog.py 4 -v 3
usage: prog.py [-h] [-v {0,1,2}] square
prog.py: error: argument -v/--verbosity: invalid choice:
$ python prog.py 4 -h
usage: prog.py [-h] [-v {0,1,2}] square

positional arguments:
  square                display a square of a given numb

options:
  -h, --help            show this help message and exit
  -v, --verbosity {0,1,2}
                        increase output verbosity
```

Tenga en cuenta que el cambio se refleja tanto en el mensaje de error como en la cadena de ayuda.

Ahora, usemos un enfoque diferente para jugar con la verbosidad, lo cual es bastante común. También coincide con la forma en que el ejecutable de CPython maneja su propio argumento de verbosidad (verifique el resultado de `python --help`):

```
import argparse
```

```
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display the square of a given
parser.add_argument("-v", "--verbosity", action="count",
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity == 2:
    print(f"the square of {args.square} equals {answer}"
elif args.verbosity == 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

Hemos introducido otra acción, «count», para contar el número de apariciones de opciones específicas.

```
$ python prog.py 4
16
$ python prog.py 4 -v
4^2 == 16
$ python prog.py 4 -vv
the square of 4 equals 16
$ python prog.py 4 --verbosity --verbosity
the square of 4 equals 16
$ python prog.py 4 -v 1
usage: prog.py [-h] [-v] square
prog.py: error: unrecognized arguments: 1
$ python prog.py 4 -h
usage: prog.py [-h] [-v] square

positional arguments:
  square           display a square of a given number

options:
  -h, --help       show this help message and exit
  -v, --verbosity  increase output verbosity
$ python prog.py 4 -vvv
16
```

- Si, ahora es mas una bandera (similar a `action="store_true"`) en la versión anterior de nuestro script. Esto debería explicar la queja.
- También se comporta de manera similar a la acción `"store_true"`.
- Ahora aquí una demostración de lo que la acción `"count"` da. Probablemente haya visto esta clase de uso antes.
- Y si no especificas la bandera `-v`, se considera que esa bandera tiene el valor `None`.
- Como debería esperarse, especificando la forma larga de la bandera, obtendríamos el mismo resultado.
- Lamentablemente, nuestra salida de ayuda no es muy informativa sobre la nueva capacidad que ha adquirido nuestro script, pero eso siempre se puede solucionar mejorando la documentación de nuestro script (por ejemplo, a través del argumento de la palabra clave `help`).
- La última salida expone un error en nuestro programa.

Vamos a arreglarlo:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given nu
parser.add_argument("-v", "--verbosity", action="count",
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2

# bugfix: replace == with >=
if args.verbosity >= 2:
    print(f"the square of {args.square} equals {answer}"
elif args.verbosity >= 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

Y esto es lo que da:

```
$ python prog.py 4 -vvv
```

```
the square of 4 equals 16
$ python prog.py 4 -vvvv
the square of 4 equals 16
$ python prog.py 4
Traceback (most recent call last):
  File "prog.py", line 11, in <module>
    if args.verbosity >= 2:
TypeError: '>=' not supported between instances of 'None
```

- La primer salida fue correcta, y corrigió el error que teníamos antes. Es decir, queremos que cualquier valor $> = 2$ sea lo más detallado posible.
- Tercer salida no tan buena.

Vamos a arreglar ese error:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given nu
parser.add_argument("-v", "--verbosity", action="count",
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity >= 2:
    print(f"the square of {args.square} equals {answer}"
elif args.verbosity >= 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

Acabamos de introducir otra palabra clave, `default`. Lo hemos configurado en `0` para que sea comparable con otros valores int. Recuerde que por defecto, si un argumento opcional no es especificado, obtiene el valor `None`, y eso no puede ser comparado con un valor int (de ahí la excepción **`TypeError`**).

Y:

```
$ python prog.py 4
```

Tu puedes llegar bastante lejos con lo que hemos aprendido hasta ahora, y solo arañado la superficie. El módulo **`argparse`** es muy poderoso, y exploraremos un poco mas antes de finalizar este tutorial.

Un poco mas avanzado

Qué pasaría si quisiéramos expandir nuestro pequeño programa para que tenga otros poderes, no solo cuadrados:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
parser.add_argument("-v", "--verbosity", action="count",
args = parser.parse_args()
answer = args.x**args.y
if args.verbosity >= 2:
    print(f"{args.x} to the power {args.y} equals {answe
elif args.verbosity >= 1:
    print(f"{args.x}^{args.y} == {answer}")
else:
    print(answer)
```

Salida:

```
$ python prog.py
usage: prog.py [-h] [-v] x y
prog.py: error: the following arguments are required: x,
$ python prog.py -h
usage: prog.py [-h] [-v] x y

positional arguments:
  x                 the base
  y                 the exponent

options:
  -h, --help       show this help message and exit
  -v, --verbosity
$ python prog.py 4 2 -v
4^2 == 16
```

Tenga en cuenta que hasta ahora hemos estado usando el nivel de verbosidad para *cambiar* el texto que se muestra. El siguiente ejemplo en lugar de usar nivel de verbosidad para mostrar *mas* texto en su lugar:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
parser.add_argument("-v", "--verbosity", action="count",
args = parser.parse_args()
answer = args.x**args.y
if args.verbosity >= 2:
    print(f"Running '{__file__}'")
if args.verbosity >= 1:
    print(f"{args.x}^{args.y} == ", end="")
print(answer)
```

Salida:

```
$ python prog.py 4 2
16
$ python prog.py 4 2 -v
4^2 == 16
$ python prog.py 4 2 -vv
Running 'prog.py'
4^2 == 16
```

Especificando argumentos ambiguos

Cuando hay ambigüedad al decidir si un argumento es posicional o para un argumento, `--` se puede usar para indicar **`parse_args()`** que todo lo que sigue a eso es un argumento posicional:

```
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> parser.add_argument('-n', nargs='+')
>>> parser.add_argument('args', nargs='*')

>>> # ambiguo, parse_args asume que es una  opción
>>> parser.parse_args(['-f'])
usage: PROG [-h] [-n N [N ...]] [args ...]
PROG: error: unrecognized arguments: -f

>>> parser.parse_args(['--', '-f'])
Namespace(args=['-f'], n=None)

>>> # ambiguo, la opción -n option acepta argumentos ávi
>>> parser.parse_args(['-n', '1', '2', '3'])
Namespace(args=[], n=['1', '2', '3'])

>>> parser.parse_args(['-n', '1', '--', '2', '3'])
Namespace(args=['2', '3'], n=['1'])
```

Opciones conflictivas

Hasta ahora, hemos estado trabajando con dos métodos de una instancia de **`argparse.ArgumentParser`**. Vamos a introducir un tercer método, **`add_mutually_exclusive_group()`**. Nos permite especificar opciones que entran en conflicto entre sí. Cambiemos también el resto del programa para que la nueva funcionalidad tenga mas sentido: presentamos la opción `--quiet`, la cual es lo opuesto a la opción `--verbose`:

```
import argparse

parser = argparse.ArgumentParser()
group = parser.add_mutually_exclusive_group()
group.add_argument("-v", "--verbose", action="store_true
group.add_argument("-q", "--quiet", action="store_true")
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
args = parser.parse_args()
answer = args.x**args.y
```

```
if args.quiet:
    print(answer)
elif args.verbose:
    print(f"{args.x} to the power {args.y} equals {answe
else:
    print(f"{args.x}^{args.y} == {answer}")
```

Nuestro programa ahora es mas simple, y perdimos algunas funcionalidades en aras de la demostración. De todos modos, aquí esta el resultado:

```
$ python prog.py 4 2
4^2 == 16
$ python prog.py 4 2 -q
16
$ python prog.py 4 2 -v
4 to the power 2 equals 16
$ python prog.py 4 2 -vq
usage: prog.py [-h] [-v | -q] x y
prog.py: error: argument -q/--quiet: not allowed with ar
$ python prog.py 4 2 -v --quiet
usage: prog.py [-h] [-v | -q] x y
prog.py: error: argument -q/--quiet: not allowed with ar
```

Esto debería ser sencillo de seguir. He agregado esa última salida para que se pueda ver el tipo de flexibilidad que obtiene, es decir, mezclar opciones de forma larga con opciones de forma corta.

Antes de concluir, probablemente quiera contarle a sus usuarios el propósito principal de su programa, solo en caso de que no lo supieran:

```
import argparse

parser = argparse.ArgumentParser(description="calculate 
group = parser.add_mutually_exclusive_group()
group.add_argument("-v", "--verbose", action="store_true
group.add_argument("-q", "--quiet", action="store_true")
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
```

```
args = parser.parse_args()
answer = args.x**args.y

if args.quiet:
    print(answer)
elif args.verbose:
    print(f"{args.x} to the power {args.y} equals {answe
else:
    print(f"{args.x}^{args.y} == {answer}")
```

Tenga en cuenta la ligera diferencia en el uso del texto. Tenga en cuenta `[-v | -q]`, lo cual nos indica que podemos usar `-v` o `-q`, pero no ambos al mismo tiempo:

```
$ python prog.py --help
usage: prog.py [-h] [-v | -q] x y

calculate X to the power of Y

positional arguments:
  x              the base
  y              the exponent

options:
  -h, --help     show this help message and exit
  -v, --verbose
  -q, --quiet
```

Cómo traducir la salida de argparse

La salida del módulo **`argparse`**, como su texto de ayuda y mensajes de error, se pueden traducir usando el módulo **`gettext`**. Esto permite que las aplicaciones localicen fácilmente los mensajes producidos por **`argparse`**. Consulte Internacionalizando sus programas y módulos para más información.

Por ejemplo, en esta salida **`argparse`**:

```
$ python prog.py --help
```

```
usage: prog.py [-h] [-v | -q] x y

calculate X to the power of Y

positional arguments:
  x               the base
  y               the exponent

options:
  -h, --help      show this help message and exit
  -v, --verbose
  -q, --quiet
```

Las cadenas de caracteres `usage:`, `positional arguments:`, `options:` y `show this help message and exit`` son todas traducibles.

Para traducir estas cadenas de caracteres, primero se deben extraer en un archivo `.po`. Por ejemplo, usando Babel [https://babel.pocoo.org/], ejecute este comando:

```
$ pybabel extract -o messages.po /usr/lib/python3.12/arg
```

Este comando extraerá todas las cadenas de caracteres traducibles del módulo **`argparse`** y las generará en un archivo llamado `messages.po`. Este comando asume que su instalación de Python está en `/usr/lib`.

Puede encontrar la ubicación del módulo **`argparse`** en su sistema usando este script:

```
import argparse
print(argparse.__file__)
```

Una vez que los mensajes en el archivo `.po` estén traducidos y las traducciones estén instaladas usando **`gettext`**, **`argparse`** podrá mostrar los mensajes traducidos.

Para traducir sus propias cadenas de caracteres en la salida **`argparse`**, use **`gettext`**.

Custom type converters

The `argparse` module allows you to specify custom type converters for your command-line arguments. This allows you to modify user input before it's stored in the `argparse.Namespace`. This can be useful when you need to pre-process the input before it is used in your program.

When using a custom type converter, you can use any callable that takes a single string argument (the argument value) and returns the converted value. However, if you need to handle more complex scenarios, you can use a custom action class with the **action** parameter instead.

For example, let's say you want to handle arguments with different prefixes and process them accordingly:

```
import argparse

parser = argparse.ArgumentParser(prefix_chars='-+')

parser.add_argument('-a', metavar='<value>', action='app
                    type=lambda x: ('-', x))
parser.add_argument('+a', metavar='<value>', action='app
                    type=lambda x: ('+', x))

args = parser.parse_args()
print(args)
```

Salida:

```
$ python prog.py -a value1 +a value2
Namespace(a=[('-', 'value1'), ('+', 'value2')])
```

In this example, we:

- Created a parser with custom prefix characters using the `prefix_chars` parameter.
- Defined two arguments, `-a` and `+a`, which used the `type` parameter to create custom type converters to store the value

in a tuple with the prefix.

Without the custom type converters, the arguments would have treated the `-a` and `+a` as the same argument, which would have been undesirable. By using custom type converters, we were able to differentiate between the two arguments.

Conclusión

El módulo **`argparse`** ofrece mucho más que solo lo mostrado aquí. Su documentación es bastante detallada y completa, y está llena de ejemplos. Habiendo seguido este tutorial, debe digerirlos fácilmente sin sentirse abrumado.

Migrating `optparse` code to `argparse`

The `argparse` module offers several higher level features not natively provided by the `optparse` module, including:

- Handling positional arguments.
- Supporting subcommands.
- Allowing alternative option prefixes like `+` and `/`.
- Handling zero-or-more and one-or-more style arguments.
- Producing more informative usage messages.
- Providing a much simpler interface for custom `type` and `action`.

Originally, the `argparse` module attempted to maintain compatibility with `optparse`. However, the fundamental design differences between supporting declarative command line option processing (while leaving positional argument processing to application code), and supporting both named options and positional arguments in the declarative interface mean that the API has diverged from that of `optparse` over time.

As described in Choosing an argument parsing library, applications that are currently using `optparse` and are happy with the way it works can just continue to use `optparse`.

Application developers that are considering migrating should also review the list of intrinsic behavioural differences described in that section before deciding whether or not migration is desirable.

For applications that do choose to migrate from `optparse` to `argparse`, the following suggestions should be helpful:

- Replace all `optparse.OptionParser.add_option()` calls with `ArgumentParser.add_argument()` calls.
- Replace `(options, args) = parser.parse_args()`

with `args = parser.parse_args()` and add additional **`ArgumentParser.add_argument()`** calls for the positional arguments. Keep in mind that what was previously called `options`, now in the **`argparse`** context is called `args`.

- Replace **`optparse.OptionParser.disable_interspersed_args()`** by using **`parse_intermixed_args()`** instead of **`parse_args()`**.
- Replace callback actions and the `callback_*` keyword arguments with `type` or `action` arguments.
- Replace string names for `type` keyword arguments with the corresponding type objects (e.g. int, float, complex, etc).
- Replace **`optparse.Values`** with **`Namespace`** and **`optparse.OptionError`** and **`optparse.OptionValueError`** with **`ArgumentError`**.
- Replace strings with implicit arguments such as `%default` or `%prog` with the standard Python syntax to use dictionaries to format strings, that is, `%(default)s` and `%(prog)s`.
- Replace the OptionParser constructor `version` argument with a call to `parser.add_argument('--version', action='version', version='<the version>')`.

`optparse` — Parser for command line options

Source code: Lib/optparse.py [https://github.com/python/cpython/tree/3.14/Lib/optparse.py]

Choosing an argument parsing library

The standard library includes three argument parsing libraries:

- **`getopt`**: a module that closely mirrors the procedural C `getopt` API. Included in the standard library since before the initial Python 1.0 release.
- **`optparse`**: a declarative replacement for `getopt` that provides equivalent functionality without requiring each application to implement its own procedural option parsing logic. Included in the standard library since the Python 2.3 release.
- **`argparse`**: a more opinionated alternative to `optparse` that provides more functionality by default, at the expense of reduced application flexibility in controlling exactly how arguments are processed. Included in the standard library since the Python 2.7 and Python 3.2 releases.

In the absence of more specific argument parsing design constraints, **`argparse`** is the recommended choice for implementing command line applications, as it offers the highest level of baseline functionality with the least application level code.

`getopt` is retained almost entirely for backwards compatibility reasons. However, it also serves a niche use case as a tool for prototyping and testing command line argument handling in `getopt`-based C applications.

`optparse` should be considered as an alternative to **`argparse`** in the following cases:

- an application is already using **`optparse`** and doesn’t want to risk the subtle behavioural changes that may arise when migrating to **`argparse`**
- the application requires additional control over the way options and positional parameters are interleaved on the command line (including the ability to disable the interleaving feature completely)
- the application requires additional control over the incremental parsing of command line elements (while `argparse` does support this, the exact way it works in practice is undesirable for some use cases)
- the application requires additional control over the handling of options which accept parameter values that may start with – (such as delegated options to be passed to invoked subprocesses)
- the application requires some other command line parameter processing behavior which `argparse` does not support, but which can be implemented in terms of the lower level interface offered by `optparse`

These considerations also mean that **`optparse`** is likely to provide a better foundation for library authors writing third party command line argument processing libraries.

As a concrete example, consider the following two command line argument parsing configurations, the first using `optparse`, and the second using `argparse`:

```
import optparse

if __name__ == '__main__':
    parser = optparse.OptionParser()
    parser.add_option('-o', '--output')
    parser.add_option('-v', dest='verbose', action='stor
    opts, args = parser.parse_args()
    process(args, output=opts.output, verbose=opts.verbo
import argparse

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-o', '--output')
    parser.add_argument('-v', dest='verbose', action='st
    parser.add_argument('rest', nargs='*')
    args = parser.parse_args()
    process(args.rest, output=args.output, verbose=args.
```

The most obvious difference is that in the `optparse` version, the non-option arguments are processed separately by the application after the option processing is complete. In the `argparse` version, positional arguments are declared and processed in the same way as the named options.

However, the `argparse` version will also handle some parameter combination differently from the way the `optparse` version would handle them. For example (amongst other differences):

- supplying `-o -v` gives `output="-v"` and `verbose=False` when using `optparse`, but a usage error with `argparse` (complaining that no value has been supplied for `-o/--output`, since `-v` is interpreted as meaning the verbosity flag)
- similarly, supplying `-o --` gives `output="--"` and `args=()` when using `optparse`, but a usage error with `argparse` (also complaining that no value has been supplied for `-o/--output`, since `--` is interpreted as terminating the option processing and treating all remaining values as positional arguments)
- supplying `-o=foo` gives `output="=foo"` when using `optparse`, but gives `output="foo"` with `argparse` (since `=` is special cased as an alternative separator for option parameter values)

Whether these differing behaviors in the `argparse` version are considered desirable or a problem will depend on the specific command line application use case.

Ver también

click [https://pypi.org/project/click/] is a third party argument processing library (originally based on `optparse`), which allows command line applications to be developed as a set of decorated command implementation functions.

Other third party libraries, such as typer [https://pypi.org/project/typer/] or msgspec-click [https://pypi.org/project/msgspec-click/], allow command line interfaces to be specified in ways that more effectively integrate with static checking of Python type annotations.

Introduction

`optparse` is a more convenient, flexible, and powerful library for parsing command-line options than the minimalist **`getopt`** module. **`optparse`** uses a more declarative style of command-line parsing: you create an instance of **`OptionParser`**, populate it with options, and parse the command line. **`optparse`** allows users to specify options in the conventional GNU/POSIX syntax, and additionally generates usage and help messages for you.

Here's an example of using **`optparse`** in a simple script:

```
from optparse import OptionParser
...
parser = OptionParser()
parser.add_option("-f", "--file", dest="filename",
                  help="write report to FILE", metavar="
parser.add_option("-q", "--quiet",
                  action="store_false", dest="verbose",
                  help="don't print status messages to s

(options, args) = parser.parse_args()
```

Con estas pocas líneas de código, los usuarios de tu script ahora pueden hacer un uso «normal» del mismo mediante la línea de comandos, por ejemplo:

```
<yourscript> --file=outfile -q
```

As it parses the command line, **`optparse`** sets attributes of the `options` object returned by **`parse_args()`** based on user-supplied command-line values. When **`parse_args()`** returns from parsing this command line, `options.filename` will be `"outfile"` and `options.verbose` will be `False`. **`optparse`** supports both long and short options, allows short options to be merged together, and allows options to be associated with their arguments in a variety of ways. Thus, the following command lines are all equivalent to the above example:

```
<yourscript> -f outfile --quiet
<yourscript> --quiet --file outfile
<yourscript> -q -foutfile
<yourscript> -qfoutfile
```

Además, los usuarios pueden ejecutar uno de los siguientes:

```
<yourscript> -h
<yourscript> --help
```

and **`optparse`** will print out a brief summary of your script's options:

```
Usage: <yourscript> [options]

Options:
  -h, --help            show this help message and exit
  -f FILE, --file=FILE  write report to FILE
  -q, --quiet           don't print status messages to s
```

donde el valor de *yourscript* se determina en tiempo de ejecución (normalmente a partir de `sys.argv [0]`).

Contexto

`optparse` was explicitly designed to encourage the creation of programs with straightforward command-line interfaces that follow the conventions established by the **`getopt()`** family of functions

available to C developers. To that end, it supports only the most common command-line syntax and semantics conventionally used under Unix. If you are unfamiliar with these conventions, reading this section will allow you to acquaint yourself with them.

Terminología

argumento

una cadena de caracteres ingresada en la línea de comandos y pasada mediante la shell a `execl()` o `execv()`. En Python, los argumentos son elementos de `sys.argv[1:]` (dado que `sys.argv[0]` es el propio nombre del programa que se está ejecutando). Las shells de Unix también usan el término «*word*» ("palabra") para referirse a ellos.

En ocasiones es deseable proporcionar una lista de argumentos que no sea `sys.argv[1:]`, por lo que deberías considerar un "argumento" como "un elemento de `sys.argv[1:]` o de alguna otra lista proporcionada como sustituto de `sys.argv[1:]`".

opción

an argument used to supply extra information to guide or customize the execution of a program. There are many different syntaxes for options; the traditional Unix syntax is a hyphen («-») followed by a single letter, e.g. `-x` or `-F`. Also, traditional Unix syntax allows multiple options to be merged into a single argument, e.g. `-x -F` is equivalent to `-xF`. The GNU project introduced `--` followed by a series of hyphen-separated words, e.g. `--file` or `--dry-run`. These are the only two option syntaxes provided by **`optparse`**.

Algunas de las otras sintaxis para opciones que el mundo ha visto son:

- un guion seguido de algunas letras, por ejemplo `-pf` (esto *no* es lo mismo que múltiples opciones fusionadas en un solo argumento)
- un guion seguido de una palabra completa, por ejemplo `-file` (esto es técnicamente equivalente a la sintaxis

anterior, pero generalmente no se ven ambas en un mismo programa)
- un signo más seguido de una sola letra, unas pocas letras o una palabra, por ejemplo `+f` o `+rgb`
- una barra seguida de una letra, de unas pocas letras o de una palabra, por ejemplo `/f` o `/file`

These option syntaxes are not supported by **`optparse`**, and they never will be. This is deliberate: the first three are non-standard on any environment, and the last only makes sense if you're exclusively targeting Windows or certain legacy platforms (e.g. VMS, MS-DOS).

argumento de opción

an argument that follows an option, is closely associated with that option, and is consumed from the argument list when that option is. With **`optparse`**, option arguments may either be in a separate argument from their option:

```
-f foo
--file foo
```

o incluidos en el mismo argumento:

```
-ffoo
--file=foo
```

Typically, a given option either takes an argument or it doesn't. Lots of people want an «optional option arguments» feature, meaning that some options will take an argument if they see it, and won't if they don't. This is somewhat controversial, because it makes parsing ambiguous: if `-a` takes an optional argument and `-b` is another option entirely, how do we interpret `-ab`? Because of this ambiguity, **`optparse`** does not support this feature.

argumento posicional

es algo que queda en la lista de argumentos después de que las opciones hayan sido analizadas sintácticamente, es decir, después de que las opciones y sus argumentos hayan sido analizados y eliminados de la lista de argumentos.

opción requerida

> an option that must be supplied on the command-line; note that the phrase «required option» is self-contradictory in English. **`optparse`** doesn't prevent you from implementing required options, but doesn't give you much help at it either.

Por ejemplo, considera esta hipotética linea de comandos:

```
prog -v --report report.txt foo bar
```

`-v` y `--report` son ambas opciones. `report.txt` es un argumento de opción, suponiendo que `--report` toma un argumento. En cambio, `foo` y `bar` son ambos argumentos posicionales.

¿Qué finalidad tienen las opciones?

Las opciones se utilizan para poder proporcionar información adicional con el fin de ajustar o personalizar la ejecución de un programa. Por si aún no ha quedado claro, las opciones suelen ser *opcionales*. Un programa debería poder ejecutarse sin problemas sin ninguna opción. (Elija un programa aleatorio del conjunto de herramientas de Unix o GNU. ¿Puede ejecutarse sin ninguna opción y aún así tener sentido? Las principales excepciones son `find`, `tar` y `dd`—los cuales son todos bichos raros mutantes que han sido apropiadamente criticados por su sintaxis no estándar y por tener interfaces confusas).

Como se ha comentado, mucha gente quiere que sus programas tengan «opciones requeridas». Pero pensemos en ello detenidamente. ¡Si es necesario, entonces *no es opcional*! Si hay una pieza de información absolutamente requerida para que tu programa pueda ejecutarse correctamente no uses opciones, para eso están los argumentos posicionales.

Como ejemplo de un buen diseño de una interfaz de línea de comandos, considera la humilde herramienta `cp` para copiar archivos. No tiene mucho sentido intentar copiar archivos sin proporcionar un destino y al menos una fuente de origen. Por lo tanto, `cp` falla si lo ejecutas sin argumentos. Sin embargo, tiene una sintaxis flexible y útil que no requiere ninguna opción:

```
cp SOURCE DEST
cp SOURCE ... DEST-DIR
```

Puedes hacer mucho simplemente con eso. La mayoría de las implementaciones de `cp` proporcionan un montón de opciones para modificar exactamente cómo se copian los archivos: se puede preservar el modo y la fecha de modificación, evitar que se sigan enlaces simbólicos, preguntar antes de sobrescribir el contenido de archivos existentes, etc. Pero nada de esto distrae de la misión principal de `cp`, que consiste en copiar uno o varios archivos en otro directorio.

¿Qué finalidad tienen los argumentos posicionales?

Los argumentos posicionales son adecuados para aquellas piezas de información que tu programa, absolutamente y sin duda alguna, requiere para funcionar.

Una buena interfaz de usuario debería tener la menor cantidad de requisitos absolutos posibles. Si tu programa requiere 17 piezas distintas de información para ejecutarse correctamente, no importa mucho *cómo* obtengas esa información del usuario; la mayoría de ellos se rendirán y se irán antes de ejecutar con éxito el programa. Esto se aplica tanto si la interfaz de usuario es una línea de comandos, un archivo de configuración o una GUI: si hace demasiadas demandas a sus usuarios, la mayoría simplemente se rendirá.

En resumen, trata de minimizar la cantidad de información que los usuarios están absolutamente obligados a proporcionar, utiliza valores predeterminados sensatos siempre que sea posible. Como es natural, también deseas que tus programas sean razonablemente flexibles, para eso están las opciones. De nuevo, no importa si son entradas en un archivo de configuración, widgets en un cuadro de diálogo de «Preferencias» de una GUI u opciones en la línea de comandos; cuantas más opciones implementes, más flexible será tu programa y más complicada se vuelve su implementación. Una excesiva flexibilidad evidentemente también tiene inconvenientes, demasiadas opciones pueden abrumar a los usuarios y hacer que tu código sea mucho más difícil de mantener.

Tutorial

While **`optparse`** is quite flexible and powerful, it's also straightforward to use in most cases. This section covers the code patterns that are common to any **`optparse`**-based program.

En primer lugar, necesitas importar la clase OptionParser y luego, al comienzo del programa principal, crear una instancia de ella:

```
from optparse import OptionParser
...
parser = OptionParser()
```

Ahora ya puedes comenzar a definir opciones. La sintaxis básica es:

```
parser.add_option(opt_str, ...,
                  attr=value, ...)
```

Each option has one or more option strings, such as `-f` or `--file`, and several option attributes that tell **`optparse`** what to expect and what to do when it encounters that option on the command line.

Normalmente, cada opción tendrá una cadena de opción corta y una cadena de opción larga, por ejemplo:

```
parser.add_option("-f", "--file", ...)
```

Puedes definir tantas cadenas de opción cortas y tantas largas como desees (incluso ninguna), siempre que haya al menos una cadena de opción en total.

The option strings passed to **`OptionParser.add_option()`** are effectively labels for the option defined by that call. For brevity, we will frequently refer to *encountering an option* on the command line; in reality, **`optparse`** encounters *option strings* and looks up options from them.

Once all of your options are defined, instruct **`optparse`** to parse your program's command line:

```
(options, args) = parser.parse_args()
```

(Si lo deseas, puedes pasar una lista de argumentos personalizada a **`parse_args()`**, pero eso rara vez es necesario: por defecto se usa `sys.argv [1:]`.)

`parse_args()` retorna dos valores:

- `options`, un objeto que contiene valores para todas tus opciones. Por ejemplo, si `--file` toma un argumento de una sola cadena de caracteres, entonces `options.file` será el nombre del archivo proporcionado por el usuario o `None` si el usuario no proporcionó esa opción en la linea de comandos
- `args`, la lista de argumentos posicionales que quedan después de analizar las opciones

Este tutorial solo cubre los cuatro atributos de opción más importantes: **`action`**, **`type`**, **`dest`** (destino) y **`help`**. De todos ellos, **`action`** es el fundamental.

Comprendiendo las acciones de opción

Actions tell **`optparse`** what to do when it encounters an option on the command line. There is a fixed set of actions hard-coded into **`optparse`**; adding new actions is an advanced topic covered in section Extending optparse. Most actions tell **`optparse`** to store a value in some variable—for example, take a string from the command line and store it in an attribute of `options`.

If you don't specify an option action, **`optparse`** defaults to `store`.

La acción store

The most common option action is `store`, which tells **`optparse`** to take the next argument (or the remainder of the current argument), ensure that it is of the correct type, and store it to your chosen destination.

Por ejemplo:

```
parser.add_option("-f", "--file",
                  action="store", type="string", dest="f
```

Now let’s make up a fake command line and ask **`optparse`** to parse it:

```
args = ["-f", "foo.txt"]
(options, args) = parser.parse_args(args)
```

When **`optparse`** sees the option string `-f`, it consumes the next argument, `foo.txt`, and stores it in `options.filename`. So, after this call to **`parse_args()`**, `options.filename` is `"foo.txt"`.

Some other option types supported by **`optparse`** are `int` and `float`. Here’s an option that expects an integer argument:

```
parser.add_option("-n", type="int", dest="num")
```

Ten en cuenta que esta opción no tiene una cadena de opción larga, lo cual es perfectamente aceptable. Además, no hay ninguna acción explícita, ya que el valor predeterminado es `store`.

Analicemos otra línea de comandos simulada. En esta ocasión, vamos a proporcionar el argumento de la opción pegado junto a la misma, sin separación entre ambos: dado que `-n42` (un argumento) es equivalente a `-n 42` (dos argumentos), el código:

```
(options, args) = parser.parse_args(["-n42"])
print(options.num)
```

imprimirá `42`.

If you don’t specify a type, **`optparse`** assumes `string`. Combined with the fact that the default action is `store`, that means our first example can be a lot shorter:

```
parser.add_option("-f", "--file", dest="filename")
```

If you don’t supply a destination, **`optparse`** figures out a sensible default from the option strings: if the first long option string is `--foo-bar`, then the default destination is `foo_bar`. If there are no long option strings, **`optparse`** looks at the first short option string: the default destination for `-f` is `f`.

`optparse` also includes the built-in `complex` type. Adding types is covered in section Extending optparse.

Manejo de opciones booleanas (flags)

Flag options—set a variable to true or false when a particular option is seen—are quite common. **`optparse`** supports them with two separate actions, `store_true` and `store_false`. For example, you might have a `verbose` flag that is turned on with `-v` and off with `-q`:

```
parser.add_option("-v", action="store_true", dest="verbo
parser.add_option("-q", action="store_false", dest="verb
```

Aquí tenemos dos opciones diferentes con el mismo destino, lo cual es totalmente correcto. Solo significa que debes tener un poco de cuidado al establecer los valores predeterminados, lo veremos a continuación.

When **`optparse`** encounters `-v` on the command line, it sets `options.verbose` to `True`; when it encounters `-q`, `options.verbose` is set to `False`.

Otras acciones

Some other actions supported by **`optparse`** are:

`"store_const"`
: almacena un valor de constante, preestablecida vía **`Option.const`**

`"append"`
: agrega el argumento de esta opción a una lista

`"count"`
: incrementa un contador en uno

`"callback"`

llama a una función específica

Estas acciones se tratan en la sección Guía de referencia y en la sección Retrollamadas de opción.

Valores por defecto

All of the above examples involve setting some variable (the «destination») when certain command-line options are seen. What happens if those options are never seen? Since we didn't supply any defaults, they are all set to `None`. This is usually fine, but sometimes you want more control. **`optparse`** lets you supply a default value for each destination, which is assigned before the command line is parsed.

First, consider the verbose/quiet example. If we want **`optparse`** to set `verbose` to `True` unless `-q` is seen, then we can do this:

```
parser.add_option("-v", action="store_true", dest="verbo
parser.add_option("-q", action="store_false", dest="verb
```

Dado que los valores predeterminados se aplican a *destination* en lugar de a cualquier opción en particular y que estas dos opciones tienen el mismo destino, lo anterior es exactamente equivalente a:

```
parser.add_option("-v", action="store_true", dest="verbo
parser.add_option("-q", action="store_false", dest="verb
```

Considera lo siguiente:

```
parser.add_option("-v", action="store_true", dest="verbo
parser.add_option("-q", action="store_false", dest="verb
```

Nuevamente, el valor predeterminado para `verbose` será `True`: el último valor predeterminado proporcionado para cualquier destino en particular es el único que se tendrá en cuenta.

Una forma más clara de especificar valores predeterminados es el método **`set_defaults()`** de OptionParser, al que puedes llamar en cualquier momento antes de llamar a **`parse_args()`**:

```
parser.set_defaults(verbose=True)
parser.add_option(...)
(options, args) = parser.parse_args()
```

Como vimos antes, el último valor especificado para un destino de opción dado es el que cuenta. Para mayor claridad, intenta utilizar un método u otro para establecer valores predeterminados, no ambos.

Generando ayuda

`optparse`'s ability to generate help and usage text automatically is useful for creating user-friendly command-line interfaces. All you have to do is supply a `help` value for each option, and optionally a short usage message for your whole program. Here's an OptionParser populated with user-friendly (documented) options:

```
usage = "usage: %prog [options] arg1 arg2"
parser = OptionParser(usage=usage)
parser.add_option("-v", "--verbose",
                  action="store_true", dest="verbose", d
                  help="make lots of noise [default]")
parser.add_option("-q", "--quiet",
                  action="store_false", dest="verbose",
                  help="be vewwy quiet (I'm hunting wabb
parser.add_option("-f", "--filename",
                  metavar="FILE", help="write output to
parser.add_option("-m", "--mode",
                  default="intermediate",
                  help="interaction mode: novice, interm
                       "or expert [default: %default]")
```

If **`optparse`** encounters either `-h` or `--help` on the command-line, or if you just call **`parser.print_help()`**, it prints the following to standard output:

```
Usage: <yourscript> [options] arg1 arg2

Options:
  -h, --help            show this help message and exit
```

```
-v, --verbose         make lots of noise [default]
-q, --quiet           be vewwy quiet (I'm hunting wabb
-f FILE, --filename=FILE
                      write output to FILE
-m MODE, --mode=MODE  interaction mode: novice, interm
                      expert [default: intermediate]
```

(If the help output is triggered by a help option, **optparse** exits after printing the help text.)

There's a lot going on here to help **optparse** generate the best possible help message:

- el script define su propio mensaje de uso:

  ```
  usage = "usage: %prog [options] arg1 arg2"
  ```

  optparse expands `%prog` in the usage string to the name of the current program, i.e. `os.path.basename(sys.argv[0])`. The expanded string is then printed before the detailed option help.

  If you don't supply a usage string, **optparse** uses a bland but sensible default: `"Usage: %prog [options]"`, which is fine if your script doesn't take any positional arguments.

- every option defines a help string, and doesn't worry about line-wrapping—**optparse** takes care of wrapping lines and making the help output look good.

- las opciones que toman un valor indican este hecho en su mensaje de ayuda generado automáticamente, por ejemplo, para la opción «mode»:

  ```
  -m MODE, --mode=MODE
  ```

  Here, «MODE» is called the meta-variable: it stands for the argument that the user is expected to supply to `-m/--mode`. By default, **optparse** converts the destination variable name to uppercase and uses that for the meta-variable. Sometimes, that's not what you want—for example, the `--`

`filename` option explicitly sets `metavar="FILE"`, resulting in this automatically generated option description:

```
-f FILE, --filename=FILE
```

Sin embargo, esto es importante para algo más que para ahorrar espacio: el texto de ayuda escrito manualmente utiliza la metavariable `FILE` para indicarle al usuario que hay una conexión entre la sintaxis semiformal `-f FILE` y la descripción semántica informal «escribir la salida en FILE». Esta es una manera simple pero efectiva de hacer que tu texto de ayuda sea mucho más claro y útil para los usuarios finales.

- options that have a default value can include `%default` in the help string—**`optparse`** will replace it with **`str()`** of the option's default value. If an option has no default value (or the default value is `None`), `%default` expands to `none`.

Agrupando opciones

Cuando se trabaja con muchas opciones, suele ser conveniente agruparlas para obtener una mejor salida de ayuda. La clase **`OptionParser`** puede contener varios grupos de opciones, cada uno de los cuales puede contener múltiples opciones.

Podemos obtener un grupo de opciones usando la clase **`OptionGroup`**:

class optparse.OptionGroup(*parser*, *title*, *description=None*)

donde

- *parser* es la instancia de **`OptionParser`** en la que se insertará el grupo
- *title* es el título dado al grupo
- *description*, opcional, es la descripción larga del grupo

la clase **`OptionGroup`** hereda de **`OptionContainer`** (al igual que **`OptionParser`**), por lo que el método **`add_option()`** se puede usar para agregar una opción al grupo.

Una vez que se han declarado todas las opciones, usando el método **`add_option_group()`** de la clase `OptionParser` el grupo se agrega al analizador sintáctico previamente definido.

Agregar un `OptionGroup` a un analizador es fácil, continuando con el analizador definido en la sección anterior:

```
group = OptionGroup(parser, "Dangerous Options",
                    "Caution: use these options at your
                    "It is believed that some of them bi
group.add_option("-g", action="store_true", help="Group
parser.add_option_group(group)
```

Esto daría como resultado la siguiente salida de ayuda:

```
Usage: <yourscript> [options] arg1 arg2

Options:
  -h, --help            show this help message and exit
  -v, --verbose         make lots of noise [default]
  -q, --quiet           be vewwy quiet (I'm hunting wabb
  -f FILE, --filename=FILE
                        write output to FILE
  -m MODE, --mode=MODE  interaction mode: novice, interm
                        expert [default: intermediate]

  Dangerous Options:
    Caution: use these options at your own risk.  It is
    of them bite.

    -g                  Group option.
```

Un ejemplo un poco más completo podría implicar el uso de más de un grupo, ampliando el ejemplo anterior:

```
group = OptionGroup(parser, "Dangerous Options",
                    "Caution: use these options at your
                    "It is believed that some of them bi
group.add_option("-g", action="store_true", help="Group
parser.add_option_group(group)
```

```
group = OptionGroup(parser, "Debug Options")
group.add_option("-d", "--debug", action="store_true",
                 help="Print debug information")
group.add_option("-s", "--sql", action="store_true",
                 help="Print all SQL statements execute
group.add_option("-e", action="store_true", help="Print
parser.add_option_group(group)
```

lo que da como resultado la siguiente salida:

```
Usage: <yourscript> [options] arg1 arg2

Options:
  -h, --help            show this help message and exit
  -v, --verbose         make lots of noise [default]
  -q, --quiet           be vewwy quiet (I'm hunting wabb
  -f FILE, --filename=FILE
                        write output to FILE
  -m MODE, --mode=MODE  interaction mode: novice, interm
                        [default: intermediate]

  Dangerous Options:
    Caution: use these options at your own risk.  It is
    of them bite.

    -g                  Group option.

  Debug Options:
    -d, --debug         Print debug information
    -s, --sql           Print all SQL statements execute
    -e                  Print every action done
```

Otro método interesante, particularmente cuando se trabaja programáticamente con grupos de opciones, es:

OptionParser.get_option_group(*opt_str*)

Retorna el **`OptionGroup`** al que pertenece la cadena de opción corta o larga *opt_str* (por ejemplo, `'-o'` o `' --`

`option'`). Si no existe dicho **`OptionGroup`**, el método retorna `None`.

Imprimir una cadena de caracteres con la versión del programa

Similar to the brief usage string, **`optparse`** can also print a version string for your program. You have to supply the string as the `version` argument to OptionParser:

```
parser = OptionParser(usage="%prog [-f] [-q]", version="
```

`%prog` is expanded just like it is in `usage`. Apart from that, `version` can contain anything you like. When you supply it, **`optparse`** automatically adds a `--version` option to your parser. If it encounters this option on the command line, it expands your `version` string (by replacing `%prog`), prints it to stdout, and exits.

Por ejemplo, si tu script se llama `/usr/bin/foo`:

```
$ /usr/bin/foo --version
foo 1.0
```

Para imprimir y obtener la cadena de caracteres `version`, se pueden utilizar cualquiera de los siguientes métodos:

OptionParser.print_version(*file*=*None*)

> Imprime el mensaje con la versión del programa actual (`self.version`) en *file* (por defecto, la salida estándar). Al igual que con **`print_usage()`**, cualquier aparición de `%prog` en `self.version` se reemplaza con el nombre del programa actual. El método no hace nada si `self.version` está vacío o no está definido.

OptionParser.get_version()

> Igual que **`print_version()`**, pero retorna la cadena de versión en lugar de imprimirla.

How **`optparse`** handles errors

There are two broad classes of errors that **`optparse`** has to worry about: programmer errors and user errors. Programmer errors are usually erroneous calls to `OptionParser.add_option()`, e.g. invalid option strings, unknown option attributes, missing option attributes, etc. These are dealt with in the usual way: raise an exception (either `optparse.OptionError` or `TypeError`) and let the program crash.

Handling user errors is much more important, since they are guaranteed to happen no matter how stable your code is. **`optparse`** can automatically detect some user errors, such as bad option arguments (passing `-n 4x` where `-n` takes an integer argument), missing arguments (`-n` at the end of the command line, where `-n` takes an argument of any type). Also, you can call **`OptionParser.error()`** to signal an application-defined error condition:

```
(options, args) = parser.parse_args()
...
if options.a and options.b:
    parser.error("options -a and -b are mutually exclusi
```

In either case, **`optparse`** handles the error the same way: it prints the program’s usage message and an error message to standard error and exits with error status 2.

Considera el primero de los dos ejemplos anteriores, donde el usuario pasa `4x` a una opción que toma un número entero:

```
$ /usr/bin/foo -n 4x
Usage: foo [options]

foo: error: option -n: invalid integer value: '4x'
```

O, en el caso en el que el usuario definitivamente no pase ningún valor:

```
$ /usr/bin/foo -n
Usage: foo [options]

foo: error: -n option requires an argument
```

`optparse`-generated error messages take care always to mention the option involved in the error; be sure to do the same when calling **`OptionParser.error()`** from your application code.

If **`optparse`**'s default error-handling behaviour does not suit your needs, you'll need to subclass OptionParser and override its **`exit()`** and/or **`error()`** methods.

Reuniendo todas las piezas

Here's what **`optparse`**-based scripts usually look like:

```
from optparse import OptionParser
...
def main():
    usage = "usage: %prog [options] arg"
    parser = OptionParser(usage)
    parser.add_option("-f", "--file", dest="filename",
                      help="read data from FILENAME")
    parser.add_option("-v", "--verbose",
                      action="store_true", dest="verbose
    parser.add_option("-q", "--quiet",
                      action="store_false", dest="verbos
    ...
    (options, args) = parser.parse_args()
    if len(args) != 1:
        parser.error("incorrect number of arguments")
    if options.verbose:
        print("reading %s..." % options.filename)
    ...

if __name__ == "__main__":
    main()
```

Guía de referencia

Creando el analizador sintáctico (parser)

The first step in using **`optparse`** is to create an OptionParser

instance.

class optparse.OptionParser(...)

El constructor de la clase OptionParser no tiene argumentos obligatorios, pero sí varios argumentos opcionales por palabra clave. Siempre deben pasarse como argumentos por palabras clave, es decir, no se debe confiar nunca en el orden en que se declaran los argumentos.

`usage` (por defecto: `"%prog [options]"`)

The usage summary to print when your program is run incorrectly or with a help option. When **`optparse`** prints the usage string, it expands `%prog` to `os.path.basename(sys.argv[0])` (or to `prog` if you passed that keyword argument). To suppress a usage message, pass the special value **`optparse.SUPPRESS_USAGE`**.

`option_list` (por defecto: `[]`)

Una lista de objetos Option con los que poblar el analizador. Las opciones en `option_list` se agregan después de cualquier opción en `standard_option_list` (un atributo de clase que puede ser establecido por las subclases de OptionParser), pero antes de cualquier versión u opción de ayuda. Obsoleto: usar en su lugar el método **`add_option()`**, una vez creado el analizador.

`option_class` (por defecto: *optparse.Option*)

Clase usada por el método **`add_option()`** para añadir opciones al analizador.

`version` (por defecto: `None`)

A version string to print when the user supplies a version option. If you supply a true value for `version`, **`optparse`** automatically adds a version option with the single option string `--version`. The substring `%prog` is expanded the same as for `usage`.

`conflict_handler` (por defecto: `"error"`)

Especifica qué hacer cuando se agregan al analizador opciones con cadenas de opción en conflicto entre si. Ver sección Conflictos entre opciones.

`description` (por defecto: `None`)
A paragraph of text giving a brief overview of your program. **`optparse`** reformats this paragraph to fit the current terminal width and prints it when the user requests help (after `usage`, but before the list of options).

`formatter` (por defecto: una nueva instancia de la clase **`IndentedHelpFormatter`**)
An instance of optparse.HelpFormatter that will be used for printing help text. **`optparse`** provides two concrete classes for this purpose: IndentedHelpFormatter and TitledHelpFormatter.

`add_help_option` (por defecto: `True`)
If true, **`optparse`** will add a help option (with option strings `-h` and `--help`) to the parser.

`prog`
La cadena de caracteres a usar como substituta de `os.path.basename(sys.argv[0])` cuando se expanda `%prog` en `usage` y en `version`.

`epilog` (por defecto: `None`)
Un párrafo con texto de ayuda que se imprimirá después de la opción de ayuda.

Completando el analizador con opciones

Hay varias formas de agregar las opciones al analizador. La forma preferida es mediante el método **`OptionParser.add_option()`**, tal como se muestra en la sección del Tutorial. El método **`add_option()`** se puede llamar de dos formas diferentes:

- pasándole una instancia de Option (como la que retorna **`make_option()`**)
- pasándole cualquier combinación de argumentos posicionales y por palabra clave que sean aceptables para

`make_option()` (es decir, para el constructor de la clase Option), lo que creará la instancia Option automáticamente

La otra alternativa es pasar una lista de instancias de Option previamente construidas al constructor OptionParser, como en el siguiente ejemplo:

```
option_list = [
    make_option("-f", "--filename",
                action="store", type="string", dest="fil
    make_option("-q", "--quiet",
                action="store_false", dest="verbose"),
    ]
parser = OptionParser(option_list=option_list)
```

(**`make_option()`** is a factory function for creating Option instances; currently it is an alias for the Option constructor. A future version of **`optparse`** may split Option into several classes, and **`make_option()`** will pick the right class to instantiate. Do not instantiate Option directly.)

Definiendo las opciones

Cada instancia de Option representa un conjunto de cadenas de opción sinónimas para la línea de comandos, por ejemplo `-f` y `--file`. Se puede especificar cualquier número de cadenas de opción cortas o largas, pero se debe proporcionar al menos una cadena de opción en total.

La forma canónica de crear una instancia de **`Option`** es mediante el método **`add_option()`** de la clase **`OptionParser`**.

OptionParser.add_option(*option*)
OptionParser.add_option(**opt_str*, *attr*=*value*, *...*)

Para definir una opción con solo una cadena de opción corta:

```
parser.add_option("-f", attr=value, ...)
```

Y para definir una opción con solo una cadena de opción larga:

```
parser.add_option("--foo", attr=value, ...)
```

The keyword arguments define attributes of the new Option object. The most important option attribute is **`action`**, and it largely determines which other attributes are relevant or required. If you pass irrelevant option attributes, or fail to pass required ones, **`optparse`** raises an **`OptionError`** exception explaining your mistake.

An option's *action* determines what **`optparse`** does when it encounters this option on the command-line. The standard option actions hard-coded into **`optparse`** are:

`"store"`
: almacena el argumento de esta opción (por defecto)

`"store_const"`
: almacena un valor de constante, preestablecida vía **`Option.const`**

`"store_true"`
: almacena `True`

`"store_false"`
: almacena `False`

`"append"`
: agrega el argumento de esta opción a una lista

`"append_const"`
: agrega un valor constante a una lista, preestablecido vía **`Option.const`**

`"count"`
: incrementa un contador en uno

`"callback"`
: llama a una función específica

`"help"`
: imprime un mensaje de uso que incluye todas las opciones y la documentación correspondiente

(Si no se proporciona una acción, el valor predeterminado es `"store"`. Para esta acción en concreto, también se pueden proporcionar los atributos de opción **`type`** y **`dest`**. Consultar Acciones de opción estándares para más información.)

As you can see, most actions involve storing or updating a value somewhere. **`optparse`** always creates a special object for this, conventionally called `options`, which is an instance of **`optparse.Values`**.

class optparse.Values

Un objeto que mantiene como atributos a los nombres de los argumentos y los valores analizados. Normalmente se crean por invocación cuando llamamos **`OptionParser.parse_args()`**, y pueden ser sobreescritos por una subclase personalizada pasada al argumento *values* de **`OptionParser.parse_args()`** (como se describe en Analizando los argumentos).

Los argumentos de opción (y algunos otros valores) se almacenan como atributos de este objeto, de acuerdo con el atributo de opción **`dest`** (destino) establecido.

Por ejemplo, cuando se llama a:

```
parser.parse_args()
```

one of the first things **`optparse`** does is create the `options` object:

```
options = Values()
```

Si una de las opciones de este analizador es definida con:

```
parser.add_option("-f", "--file", action="store", type="
```

y la línea de comandos que se analiza incluye cualquiera de las siguientes variantes:

```
-ffoo
```

```
-f foo
--file=foo
--file foo
```

then **`optparse`**, on seeing this option, will do the equivalent of

```
options.filename = "foo"
```

Los atributos de opción **`type`** y **`dest`** son casi tan importantes como **`action`**, pero **`action`** es el único de ellos que es apropiado para *todas* las opciones.

Atributos de opción

class optparse.Option

Un único argumento de línea de comando, con varios atributos pasados como palabras clave al constructor. Normalmente se crea con **`OptionParser.add_option()`** en lugar de directamente, y puede ser sobrescrito por una clase personalizada mediante el argumento *option_class* de **`OptionParser`**.

The following option attributes may be passed as keyword arguments to **`OptionParser.add_option()`**. If you pass an option attribute that is not relevant to a particular option, or fail to pass a required option attribute, **`optparse`** raises **`OptionError`**.

Option.action

(por defecto: `"store"`)

Determines **`optparse`**’s behaviour when this option is seen on the command line; the available options are documented here.

Option.type

(por defecto: `"string"`)

El tipo de argumento esperado para esta opción (por ejemplo, `"string"` o `"int"`). Los tipos de opción disponibles están documentados aquí.

Option.dest

(por defecto: derivado de las cadenas de opción)

If the option's action implies writing or modifying a value somewhere, this tells **`optparse`** where to write it: **`dest`** names an attribute of the `options` object that **`optparse`** builds as it parses the command line.

Option.default

El valor que se utilizará como destino de esta opción si la opción no aparece en la línea de comandos. Ver también **`OptionParser.set_defaults()`**.

Option.nargs

(por defecto: 1)

How many arguments of type **`type`** should be consumed when this option is seen. If > 1, **`optparse`** will store a tuple of values to **`dest`**.

Option.const

Para acciones que almacenan un valor constante, el valor constante a almacenar.

Option.choices

Para opciones de tipo `"choice"`, la lista con las cadenas que el usuario puede elegir.

Option.callback

Para las opciones con la acción `"callback"` asignada, el objeto invocable a llamar cuando se encuentra esta opción. Consultar la sección Retrollamadas de opción para obtener más detalles sobre los argumentos que son pasados al objeto invocable.

Option.callback_args
Option.callback_kwargs

Argumentos posicionales y por palabra clave adicionales para ser pasados a `callback` después de los cuatro argumentos pasados a la retrollamada estándar.

Option.help

Texto de ayuda a imprimir para esta opción cuando se enumeran todas las opciones disponibles, después de que el usuario proporcione una opción **`help`** (como `--help`). Si no se proporciona ningún texto de ayuda, la opción seguirá apareciendo, solo que sin texto de ayuda. Para ocultar esta opción completamente, se debe asignar al atributo el valor especial **`optparse.SUPPRESS_HELP`**.

Option.metavar

(por defecto: derivado de las cadenas de opción)

Reemplazo para los argumentos de opción que se utilizará al imprimir el texto de ayuda. Consultar la sección Tutorial para ver un ejemplo.

Acciones de opción estándares

The various option actions all have slightly different requirements and effects. Most actions have several relevant option attributes which you may specify to guide **`optparse`**'s behaviour; a few have required attributes, which you must specify for any option using that action.

- `"store"` [atributos relacionados: **`type`**, **`dest`**, **`nargs`**, **`choices`**]

  La opción debe ir seguida de un argumento, que se convierte en un valor de acuerdo a **`type`** y se almacena en **`dest`**. Si **`nargs`** es mayor que 1, se consumirán varios argumentos de la línea de comandos. Todos ellos se convertirán de acuerdo a **`type`** y se almacenarán en **`dest`** como una tupla. Consultar la sección Tipos de opción estándares para más información.

  Si se proporciona **`choices`** (una lista o tupla de cadenas de caracteres), el tipo por defecto es `"choice"`.

Si no se proporciona `type`, el tipo por defecto es `"string"`.

If `dest` is not supplied, **`optparse`** derives a destination from the first long option string (e.g., `--foo-bar` implies `foo_bar`). If there are no long option strings, **`optparse`** derives a destination from the first short option string (e.g., `-f` implies `f`).

Ejemplo:

```
parser.add_option("-f")
parser.add_option("-p", type="float", nargs=3, dest
```

Mientras analiza la línea de comandos:

```
-f foo.txt -p 1 -3.5 4 -fbar.txt
```

`optparse` will set

```
options.f = "foo.txt"
options.point = (1.0, -3.5, 4.0)
options.f = "bar.txt"
```

- `"store_const"` [atributo requerido: `const`; atributo relacionado: `dest`]

  El valor `const` es almacenado en `dest`.

  Ejemplo:

```
parser.add_option("-q", "--quiet",
                  action="store_const", const=0, de
parser.add_option("-v", "--verbose",
                  action="store_const", const=1, de
parser.add_option("--noisy",
                  action="store_const", const=2, de
```

  If `--noisy` is seen, **`optparse`** will set

```
options.verbose = 2
```

- `"store_true"` [atributo relacionado: **`dest`**]

  Un caso especial de `"store_const"` que almacena `True` en **`dest`**.

- `"store_false"` [atributo relacionado: **`dest`**]

  Como `"store_true"`, pero almacena `False`.

  Ejemplo:

  ```
  parser.add_option("--clobber", action="store_true",
  parser.add_option("--no-clobber", action="store_fal
  ```

- `"append"` [atributos relacionados: **`type`**, **`dest`**, **`nargs`**, **`choices`**]

  The option must be followed by an argument, which is appended to the list in **`dest`**. If no default value for **`dest`** is supplied, an empty list is automatically created when **`optparse`** first encounters this option on the command-line. If **`nargs`** > 1, multiple arguments are consumed, and a tuple of length **`nargs`** is appended to **`dest`**.

  Los valores por defecto para los atributos **`type`** y **`dest`** son los mismos que para la acción `"store"`.

  Ejemplo:

  ```
  parser.add_option("-t", "--tracks", action="append"
  ```

  If `-t3` is seen on the command-line, **`optparse`** does the equivalent of:

  ```
  options.tracks = []
  options.tracks.append(int("3"))
  ```

  Si, un poco más adelante, se encuentra `--tracks=4`, procede así:

  ```
  options.tracks.append(int("4"))
  ```

La acción `append` llama al método `append` con el valor actual de la opción. Esto significa que cualquier valor por defecto especificado debe tener un método `append`. También significa que si existe un valor por defecto, los elementos por defecto estarán presentes en el valor analizado para la opción, con todos los valores de la línea de comandos agregados a la lista a continuación de ellos:

```
>>> parser.add_option("--files", action="append", d
>>> opts, args = parser.parse_args(['--files', 'ove
>>> opts.files
['~/.mypkg/defaults', 'overrides.mypkg']
```

- `"append_const"` [atributo requerido: **`const`**; atributo relacionado: **`dest`**]

  Igual que `"store_const"`, pero el valor **`const`** se agrega a **`dest`**. Como ocurre con `"append"`, **`dest`** por defecto es `None` y se crea automáticamente una lista vacía la primera vez que se encuentra la opción en la linea de comandos.

- `"count"` [atributo relacionado: **`dest`**]

  Incrementa el entero almacenado en **`dest`**. Si no se proporciona un valor por defecto, **`dest`** se establece en cero antes de incrementarse por primera vez.

  Ejemplo:

  ```
  parser.add_option("-v", action="count", dest="verbo
  ```

  The first time `-v` is seen on the command line, **`optparse`** does the equivalent of:

  ```
  options.verbosity = 0
  options.verbosity += 1
  ```

  Cada aparición posterior de `-v` da como resultado:

  ```
  options.verbosity += 1
  ```

- `"callback"` [atributo requerido: **`callback`**; atributos relacionados: **`type`**, **`nargs`**, **`callback_args`**, **`callback_kwargs`**]

Llama a la función especificada por **`callback`**, que es llamada como:

```
func(option, opt_str, value, parser, *args, **kwarg
```

Ver sección Retrollamadas de opción para más detalles.

- `"help"`

  Imprime un mensaje de ayuda completo para todas las opciones presentes en el analizador de opciones actual. El mensaje de ayuda se construye a partir de la cadena `usage`, pasada al constructor de OptionParser, y la cadena **`help`**, pasada a cada opción.

  Si no se proporciona una cadena **`help`** para una opción, dicha opción seguirá apareciendo en el mensaje de ayuda. Para omitir una opción por completo, debe usarse el valor especial **`optparse.SUPPRESS_HELP`**.

  `optparse` automatically adds a **`help`** option to all OptionParsers, so you do not normally need to create one.

  Ejemplo:

```
from optparse import OptionParser, SUPPRESS_HELP

# usually, a help option is added automatically, bu
# be suppressed using the add_help_option argument
parser = OptionParser(add_help_option=False)

parser.add_option("-h", "--help", action="help")
parser.add_option("-v", action="store_true", dest="
                  help="Be moderately verbose")
parser.add_option("--file", dest="filename",
                  help="Input file to read data fro
parser.add_option("--secret", help=SUPPRESS_HELP)
```

If **`optparse`** sees either `-h` or `--help` on the command line, it will print something like the following help message to stdout (assuming `sys.argv[0]` is `"foo.py"`):

```
Usage: foo.py [options]

Options:
  -h, --help        Show this help message and exit
  -v                Be moderately verbose
  --file=FILENAME   Input file to read data from
```

After printing the help message, **`optparse`** terminates your process with `sys.exit(0)`.

- `"version"`

  Prints the version number supplied to the OptionParser to stdout and exits. The version number is actually formatted and printed by the `print_version()` method of OptionParser. Generally only relevant if the `version` argument is supplied to the OptionParser constructor. As with **`help`** options, you will rarely create `version` options, since **`optparse`** automatically adds them when needed.

Tipos de opción estándares

`optparse` has five built-in option types: `"string"`, `"int"`, `"choice"`, `"float"` and `"complex"`. If you need to add new option types, see section Extending optparse.

Los argumentos de las opciones en la cadena ingresada no se verifican ni se convierten de ninguna manera: el texto de la línea de comandos se almacena en el destino (o se pasa a la retrollamada) tal cual.

Los argumentos enteros (tipo `"int"`) se analizan de la siguiente manera:

- si el número comienza con `0x`, se analiza como un número hexadecimal
- si el número comienza con `0`, se analiza como un número

octal
- si el número comienza con `0b`, se analiza como un número binario
- en cualquier otro caso, el número se analiza como un número decimal

The conversion is done by calling **`int()`** with the appropriate base (2, 8, 10, or 16). If this fails, so will **`optparse`**, although with a more useful error message.

Los argumentos de las opciones de tipo `"float"` y `"complex"` se convierten directamente usando **`float()`** y **`complex()`** respectivamente, con un manejo de errores similar.

Las opciones de tipo `"choice"` son un subtipo de las opciones `"string"`. El atributo de opción **`choices`** (que es una secuencia de cadenas) define el conjunto de argumentos de opción permitidos. Posteriormente, **`optparse.check_choice()`** comparará los argumentos de las opciones proporcionadas por el usuario con esta lista maestra y lanzará una excepción **`OptionValueError`** si se proporciona una cadena no válida.

Analizando los argumentos

El objetivo primario de crear y agregar opciones a un OptionParser es llamar a su método **`parse_args()`**.

OptionParser.parse_args(*args*=*None*, *values*=*None*)

Analiza las opciones de la línea de comando encontradas en *args*.

Los parámetros de entrada son

`args`

la lista de argumentos a procesar (por defecto: `sys.argv [1:]`)

`values`

a **`Values`** object to store option arguments in (default: a new instance of **`Values`**) – if you give an existing object, the option defaults will not be initialized on it y los valores de retorno es un par `(options, args)` donde

`options`
: the same object that was passed in as *values*, or the `optparse.Values` instance created by **`optparse`**

`args`
: los argumentos posicionales que quedan en la linea de comandos después de que se hayan procesado todas las opciones

El uso más habitual es no proporcionar ningún argumento por palabra clave. Si se proporciona `values`, dicho argumento será modificado mediante llamadas repetidas a **`setattr()`** (aproximadamente una por cada argumento de opción a almacenar en un destino de opción) y finalmente será retornado por el método **`parse_args()`**.

Si el método **`parse_args()`** encuentra algún error en la lista de argumentos, llama al método **`error()`** de OptionParser con un mensaje de error apropiado para el usuario final. Esto causa que el proceso termine con un estado de salida de 2 (el estado de salida tradicional en Unix para errores en la línea de comandos).

Consultar y manipular el analizador de opciones

El comportamiento predeterminado del analizador de opciones se puede personalizar ligeramente. También se puede indagar en el analizador de opciones y ver qué hay en él. OptionParser proporciona varios métodos para ayudar con éstos propósitos:

OptionParser.disable_interspersed_args()

Set parsing to stop on the first non-option. For example, if `-a` and `-b` are both simple options that take no arguments, **`optparse`** normally accepts this syntax:

```
prog -a arg1 -b arg2
```

y la trata de forma equivalente a:

```
prog -a -b arg1 arg2
```

Para deshabilitar esta funcionalidad, se debe llamar al método **`disable_interspersed_args()`**. Esto restaura la sintaxis tradicional usada en Unix, donde el análisis de opción se detiene con el primer argumento que no es una opción.

Se debe usar este método si se dispone de un procesador de comandos que ejecuta otro comando con sus propias opciones y se desea asegurarse de que estas opciones no se confunden entre si. Lo que puede ocurrir si, por ejemplo, cada comando tiene un conjunto diferente de opciones.

OptionParser.enable_interspersed_args()

Configura el análisis para que no se detenga si encuentra un argumento que no sea una opción, lo que permite intercalar modificadores con argumentos de linea de comandos. Este es el comportamiento por defecto.

OptionParser.get_option(*opt_str*)

Retorna la instancia de Option con la cadena de opción *opt_str*, o `None` si ninguna opción tiene esa cadena de opción.

OptionParser.has_option(*opt_str*)

Retorna `True` si OptionParser tiene una opción con la cadena de opción *opt_str* (por ejemplo, `-q` o `--verbose`).

OptionParser.remove_option(*opt_str*)

Si **`OptionParser`** tiene una opción correspondiente a *opt_str*, esa opción es eliminada. Si esa opción proporcionó cualquier otra cadena de opción, todas esas cadenas de opción quedan invalidadas. Si *opt_str* no aparece en ninguna opción que pertenezca a este **`OptionParser`**, se lanza una excepción **`ValueError`**.

Conflictos entre opciones

Si no se tiene cuidado, es fácil definir opciones con cadenas de opción en conflicto entre si:

```
parser.add_option("-n", "--dry-run", ...)
...
parser.add_option("-n", "--noisy", ...)
```

(Esto es particularmente cierto si se ha definido una subclase propia de OptionParser con algunas opciones estándar.)

Every time you add an option, **`optparse`** checks for conflicts with existing options. If it finds any, it invokes the current conflict-handling mechanism. You can set the conflict-handling mechanism either in the constructor:

```
parser = OptionParser(..., conflict_handler=handler)
```

o mediante una llamada separada:

```
parser.set_conflict_handler(handler)
```

Los administradores de conflictos disponibles son:

`"error"` (por defecto)
: se asume que los conflictos entre opciones son un error de programación y, por tanto, generarán una excepción **`OptionConflictError`**

`"resolve"`
: resuelve conflictos de opciones de forma inteligente (ver más abajo)

Como ejemplo, vamos a definir un **`OptionParser`** que resuelva conflictos de manera inteligente y agregaremos algunas opciones conflictivas:

```
parser = OptionParser(conflict_handler="resolve")
parser.add_option("-n", "--dry-run", ..., help="do no ha
parser.add_option("-n", "--noisy", ..., help="be noisy")
```

At this point, **`optparse`** detects that a previously added option is already using the `-n` option string. Since `conflict_handler` is `"resolve"`, it resolves the situation by removing `-n` from the earlier option's list of option strings. Now `--dry-run` is the only way for the user to activate that option. If the user asks for help, the help message will reflect that:

```
Options:
  --dry-run     do no harm
  ...
  -n, --noisy   be noisy
```

It's possible to whittle away the option strings for a previously added option until there are none left, and the user has no way of invoking that option from the command-line. In that case, **`optparse`** removes that option completely, so it doesn't show up in help text or anywhere else. Carrying on with our existing OptionParser:

```
parser.add_option("--dry-run", ..., help="new dry-run op
```

At this point, the original `-n/--dry-run` option is no longer accessible, so **`optparse`** removes it, leaving this help text:

```
Options:
  ...
  -n, --noisy   be noisy
  --dry-run     new dry-run option
```

Limpieza

Las instancias de OptionParser tienen varias referencias cíclicas. Esto no debería ser un problema para el recolector de basura de Python, pero es posible que se desee romper las referencias cíclicas explícitamente llamando al método **`destroy()`** de la instancia OptionParser una vez que se haya terminado. Esto es particularmente útil en aplicaciones de larga ejecución en las que OptionParser puede terminar accediendo a grafos de objetos considerablemente grandes.

Otros métodos

OptionParser admite varios métodos públicos más:

OptionParser.set_usage(*usage*)

Establece la cadena de caracteres de uso de acuerdo a las reglas descritas anteriormente para el argumento por palabra clave `usage` del constructor. Pasando `None` se establece la cadena de uso por defecto; use el valor especial **`optparse.SUPPRESS_USAGE`** para suprimir el mensaje de uso.

OptionParser.print_usage(*file=None*)

Imprime el mensaje de uso del programa actual (`self.usage`) en *file* (que por defecto es la salida estándar). Cualquier aparición de la cadena de caracteres `%prog` en `self.usage` es reemplazada con el nombre del programa actual. No hace nada si `self.usage` está vacío o no ha sido definido.

OptionParser.get_usage()

Igual que **`print_usage()`** pero retorna la cadena de uso en vez de imprimirla.

OptionParser.set_defaults(*dest=value*, ...)

Establece valores por defecto para varios destinos de opción a la vez. Usar el método **`set_defaults()`** es la forma preferida de establecer valores por defecto para las opciones, ya que varias opciones pueden compartir el mismo destino. Por ejemplo, si varias opciones de `mode` tienen el mismo destino, cualquiera de ellas puede establecer el valor por defecto, pero el último establecido es el que finalmente queda establecido:

```
parser.add_option("--advanced", action="store_const
                  dest="mode", const="advanced",
                  default="novice")    # overridden
parser.add_option("--novice", action="store_const",
                  dest="mode", const="novice",
                  default="advanced")  # overrides
```

Para evitar esta confusión, usa el método **`set_defaults()`**:

```
parser.set_defaults(mode="advanced")
parser.add_option("--advanced", action="store_const
                  dest="mode", const="advanced")
parser.add_option("--novice", action="store_const",
                  dest="mode", const="novice")
```

Retrollamadas de opción

When **`optparse`**’s built-in actions and types aren’t quite enough for your needs, you have two choices: extend **`optparse`** or define a callback option. Extending **`optparse`** is more general, but overkill for a lot of simple cases. Quite often a simple callback is all you need.

Hay dos pasos a seguir para definir una opción con retrollamada:

- definir la opción en sí usando la acción `"callback"`
- escribir la retrollamada. Esta es una función (o método) que toma al menos cuatro argumentos, los cuales son descritos a continuación

Definición de una opción con retrollamada

Generalmente, la forma más sencilla de definir una opción con retrollamada es mediante el método **`OptionParser.add_option()`**. Aparte de **`action`**, el único atributo de opción que debes especificar es `callback`, que es la función a llamar:

```
parser.add_option("-c", action="callback", callback=my_c
```

`callback` is a function (or other callable object), so you must have already defined `my_callback()` when you create this callback option. In this simple case, **`optparse`** doesn’t even know if `-c` takes any arguments, which usually means that the option takes no arguments—the mere presence of `-c` on the command-line is all it needs to know. In some circumstances, though, you might want your callback to consume an arbitrary number of command-line arguments. This is where writing callbacks gets tricky; it’s covered later in this section.

`optparse` always passes four particular arguments to your callback, and it will only pass additional arguments if you specify them via `callback_args` and `callback_kwargs`. Thus, the minimal callback function signature is:

```
def my_callback(option, opt, value, parser):
```

Los cuatro argumentos para la retrollamada se describen a continuación.

Hay varios atributos de opción adicionales que se pueden proporcionar cuando se define una opción con retrollamada:

`type`
: has its usual meaning: as with the `"store"` or `"append"` actions, it instructs **`optparse`** to consume one argument and convert it to `type`. Rather than storing the converted value(s) anywhere, though, **`optparse`** passes it to your callback function.

`nargs`
: also has its usual meaning: if it is supplied and > 1, **`optparse`** will consume `nargs` arguments, each of which must be convertible to `type`. It then passes a tuple of converted values to your callback.

`callback_args`
: una tupla con los argumentos posicionales adicionales para pasar a la retrollamada

`callback_kwargs`
: un diccionario con los argumentos por palabra clave para pasar a la retrollamada

Cómo son invocadas las retrollamadas

Todas las retrollamadas son invocadas de la siguiente forma:

```
func(option, opt_str, value, parser, *args, **kwargs)
```

donde

`option`
: es la instancia de Option que invoca a la retrollamada

`opt_str`
: es la cadena de opción encontrada en la línea de comandos que activa la retrollamada. (Si se utilizó una opción larga abreviada, `opt_str` será la cadena de opción canónica completa— por ejemplo, si el usuario ingresa `--foo` en la línea de comandos como una abreviatura de `--foobar`, entonces `opt_str` será `"--foobar"`.)

`value`
: is the argument to this option seen on the command-line. **`optparse`** will only expect an argument if **`type`** is set; the type of `value` will be the type implied by the option's type. If **`type`** for this option is `None` (no argument expected), then `value` will be `None`. If **`nargs`** > 1, `value` will be a tuple of values of the appropriate type.

`parser`
: es la instancia de OptionParser que controla todo. Su utilidad principal radica en que permite acceder a otros datos de interés a través de sus atributos de instancia:

  `parser.largs`
  : la lista actual de argumentos que sobran, es decir, argumentos que se han consumido pero que no son opciones ni argumentos de opción. Siéntete libre de modificar `parser.largs`, por ejemplo, agregando más argumentos. (Esta lista se convertirá en `args`, el segundo valor de retorno del método **`parse_args()`**.)

  `parser.rargs`
  : la lista actual de argumentos restantes, es decir, los argumentos que quedan a continuación de `opt_str` y `value` (si corresponde), una vez eliminados ambos.

Siéntete libre de modificar `parser.rargs`, por ejemplo, consumiendo más argumentos.

`parser.values`

the object where option values are by default stored (an instance of optparse.OptionValues). This lets callbacks use the same mechanism as the rest of **optparse** for storing option values; you don’t need to mess around with globals or closures. You can also access or modify the value(s) of any options already encountered on the command-line.

`args`

es una tupla de argumentos posicionales arbitrarios suministrados a través del atributo de opción **`callback_args`**.

`kwargs`

es un diccionario con argumentos por palabra clave arbitrarios proporcionados por **`callback_kwargs`**.

Lanzando errores en una retrollamada

The callback function should raise **`OptionValueError`** if there are any problems with the option or its argument(s). **optparse** catches this and terminates the program, printing the error message you supply to stderr. Your message should be clear, concise, accurate, and mention the option at fault. Otherwise, the user will have a hard time figuring out what they did wrong.

Ejemplo de retrollamada 1: una retrollamada trivial

Aquí hay un ejemplo de una opción con retrollamada que no tiene argumentos y simplemente registra que se encontró la opción en la línea de comandos:

```
def record_foo_seen(option, opt_str, value, parser):
    parser.values.saw_foo = True
parser.add_option("--foo", action="callback", callback=r
```

Ciertamente, se puede hacer lo mismo simplemente con la acción `"store_true"`.

Ejemplo de retrollamada 2: comprobar el orden de las opciones

Aquí tenemos un ejemplo un poco más interesante: registra el hecho de que se ha encontrado `-a`, pero lanza un error si viene después de `-b` en la línea de comandos

```
def check_order(option, opt_str, value, parser):
    if parser.values.b:
        raise OptionValueError("can't use -a after -b")
    parser.values.a = 1
...
parser.add_option("-a", action="callback", callback=chec
parser.add_option("-b", action="store_true", dest="b")
```

Ejemplo de retrollamada 3: comprobar el orden de las opciones (generalizado)

If you want to reuse this callback for several similar options (set a flag, but blow up if `-b` has already been seen), it needs a bit of work: the error message and the flag that it sets must be generalized.

```
def check_order(option, opt_str, value, parser):
    if parser.values.b:
        raise OptionValueError("can't use %s after -b" %
    setattr(parser.values, option.dest, 1)
...
parser.add_option("-a", action="callback", callback=chec
parser.add_option("-b", action="store_true", dest="b")
parser.add_option("-c", action="callback", callback=chec
```

Ejemplo de retrollamada 4: comprobar una condición arbitraria

Por supuesto, puedes poner cualquier condición aquí, no estás limitado a verificar los valores de las opciones previamente definidas. Por ejemplo, si tienes opciones que no deberían llamarse cuando hay luna llena, todo lo que tienes que hacer es lo siguiente:

```
def check_moon(option, opt_str, value, parser):
    if is_moon_full():
        raise OptionValueError("%s option invalid when m
                               % opt_str)
    setattr(parser.values, option.dest, 1)
...
parser.add_option("--foo",
                  action="callback", callback=check_moon
```

(La definición de `is_moon_full()` se deja como ejercicio para el lector).

Ejemplo de retrollamada 5: argumentos fijos

Las cosas se ponen un poco más interesantes cuando se definen opciones con retrollamada que toman un número fijo de argumentos. Especificar que una opción con retrollamada toma argumentos es similar a definir una opción `"store"` o `"append"`: si se define **`type`**, entonces la opción toma un argumento que debe poder convertirse a ese tipo; si además se define **`nargs`**, entonces la opción toma **`nargs`** argumentos.

Aquí hay un ejemplo que simplemente emula la acción `"store"` estándar:

```
def store_value(option, opt_str, value, parser):
    setattr(parser.values, option.dest, value)
...
parser.add_option("--foo",
                  action="callback", callback=store_valu
                  type="int", nargs=3, dest="foo")
```

Note that **`optparse`** takes care of consuming 3 arguments and converting them to integers for you; all you have to do is store them. (Or whatever; obviously you don’t need a callback for this example.)

Ejemplo de retrollamada 6: argumentos variables

Things get hairy when you want an option to take a variable number of arguments. For this case, you must write a callback, as **optparse** doesn't provide any built-in capabilities for it. And you have to deal with certain intricacies of conventional Unix command-line parsing that **optparse** normally handles for you. In particular, callbacks should implement the conventional rules for bare `--` and `-` arguments:

- tanto `--` como `-` pueden ser argumentos de opción
- `--` desnudo (si no es el argumento de alguna opción): detener el procesamiento de la línea de comandos y descartar el `--`
- `-` desnudo (si no es el argumento de alguna opción): detener el procesamiento de la línea de comandos pero mantener el `-` (añadiéndolo a `parser.largs`)

If you want an option that takes a variable number of arguments, there are several subtle, tricky issues to worry about. The exact implementation you choose will be based on which trade-offs you're willing to make for your application (which is why **optparse** doesn't support this sort of thing directly).

En cualquier caso, aquí hay un intento de una retrollamada para una opción con un número de argumentos variable:

```
def vararg_callback(option, opt_str, value, parser):
    assert value is None
    value = []

    def floatable(str):
        try:
            float(str)
            return True
        except ValueError:
            return False

    for arg in parser.rargs:
        # stop on --foo like options
        if arg[:2] == "--" and len(arg) > 2:
            break
        # stop on -a, but not on -3 or -3.0
        if arg[:1] == "-" and len(arg) > 1 and not float
            break
        value.append(arg)

    del parser.rargs[:len(value)]
    setattr(parser.values, option.dest, value)

...
parser.add_option("-c", "--callback", dest="vararg_attr"
                  action="callback", callback=vararg_cal
```

Extending `optparse`

Since the two major controlling factors in how **`optparse`** interprets command-line options are the action and type of each option, the most likely direction of extension is to add new actions and new types.

Agregando nuevos tipos

To add new types, you need to define your own subclass of **`optparse`**'s `Option` class. This class has a couple of attributes that define **`optparse`**'s types: **`TYPES`** and **`TYPE_CHECKER`**.

Option.TYPES

Una tupla con nombres de tipos. En tu subclase, simplemente define una nueva tupla **`TYPES`** que se base en la estándar.

Option.TYPE_CHECKER

Un diccionario que asigna nombres de tipos a funciones de verificación de tipo. Una función de verificación de tipo tiene la siguiente firma:

```
def check_mytype(option, opt, value)
```

donde `option` es una instancia de **`Option`**, `opt` es una cadena de opción (por ejemplo, `-f`) y `value` es la cadena de caracteres de la línea de comandos que debe comprobarse y convertirse al tipo deseado. `check_mytype()` debería retornar un objeto del tipo hipotético `mytype`. El valor retornado por una función de verificación de tipo terminará formando parte de la instancia de OptionValues retornada por el método **`OptionParser.parse_args()`** o será pasada a una retrollamada como parámetro `value`.

Tu función de verificación de tipo debería lanzar una excepción **`OptionValueError`** si encuentra algún problema. **`OptionValueError`** toma una cadena de caracteres como único argumento, que es pasada tal cual al método **`error()`** de la clase **`OptionParser`**, que a su vez antepone a la misma el nombre del programa y la cadena `"error:"` e imprime todo en la salida de error estándar antes de finalizar el proceso.

Here's a silly example that demonstrates adding a `"complex"` option type to parse Python-style complex numbers on the command line. (This is even sillier than it used to be, because **`optparse`** 1.3 added built-in support for complex numbers, but never mind.)

Primero, las importaciones necesarias:

```
from copy import copy
from optparse import Option, OptionValueError
```

En primer lugar, debes definir tu verificador de tipo, ya que se hace referencia a él más adelante (en el atributo de clase **`TYPE_CHECKER`** de tu subclase de Option):

```
def check_complex(option, opt, value):
    try:
        return complex(value)
    except ValueError:
        raise OptionValueError(
            "option %s: invalid complex value: %r" % (op
```

Finalmente, la subclase de Option:

```
class MyOption (Option):
    TYPES = Option.TYPES + ("complex",)
    TYPE_CHECKER = copy(Option.TYPE_CHECKER)
    TYPE_CHECKER["complex"] = check_complex
```

(If we didn't make a `copy()` of `Option.TYPE_CHECKER`, we would end up modifying the `TYPE_CHECKER` attribute of `optparse`'s Option class. This being Python, nothing stops you from doing that except good manners and common sense.)

That's it! Now you can write a script that uses the new option type just like any other `optparse`-based script, except you have to instruct your OptionParser to use MyOption instead of Option:

```
parser = OptionParser(option_class=MyOption)
parser.add_option("-c", type="complex")
```

Alternativamente, puedes crear tu propia lista de opciones y pasarla a OptionParser; si no usas `add_option()` de la manera anterior, no necesitas decirle a OptionParser qué clase de opción usar:

```
option_list = [MyOption("-c", action="store", type="comp
parser = OptionParser(option_list=option_list)
```

Agregando nuevas acciones

Adding new actions is a bit trickier, because you have to understand that `optparse` has a couple of classifications for actions:

Acciones «store»

> actions that result in `optparse` storing a value to an attribute of the current OptionValues instance; these options require a `dest` attribute to be supplied to the Option constructor.

Acciones «typed»

> acciones que toman un valor de la línea de comandos, esperando que sea de cierto tipo; o mejor dicho, una cadena de caracteres que se pueda convertir a un determinado tipo.

Estas opciones requieren un atributo `type` para el constructor de Option.

Ambas categorías se solapan entre si: algunas acciones `store` predeterminadas son `"store"`, `"store_const"`, `"append"` y `"count"`, mientras que las acciones «typed» predeterminadas son `"store"`, `"append"` y `"callback"`.

Cuando agregas una acción, debes categorizarla enumerándola en al menos uno de los siguientes atributos de clase de la clase Option (todos ellos son listas de cadenas de caracteres):

Option.ACTIONS

Todas las acciones deben aparecer en ACTIONS.

Option.STORE_ACTIONS

Las acciones «store» también se enumeran aquí.

Option.TYPED_ACTIONS

Las acciones «typed» también se enumeran aquí.

Option.ALWAYS_TYPED_ACTIONS

Actions that always take a type (i.e. whose options always take a value) are additionally listed here. The only effect of this is that **`optparse`** assigns the default type, `"string"`, to options with no explicit type whose action is listed in `ALWAYS_TYPED_ACTIONS`.

Para implementar realmente tu nueva acción, debes redefinir el método **`take_action()`** de Option, implementando un nuevo método que reconozca tu acción.

Por ejemplo, agreguemos una nueva acción `"extend"`. Esta es similar a la acción estándar `"append"`, pero en lugar de tomar un solo valor de la línea de comandos y agregarlo a una lista existente, `"extend"` tomará múltiples valores en una sola cadena de caracteres delimitada por comas y amplía una lista previamente existente con ellos. Es decir, si `--names` es una opción `"extend"` de tipo `"string"`, la línea de comandos

```
--names=foo,bar --names blah --names ding,dong
```

daría como resultado una lista como la siguiente:

```
["foo", "bar", "blah", "ding", "dong"]
```

De nuevo, definimos una subclase de Option:

```
class MyOption(Option):

    ACTIONS = Option.ACTIONS + ("extend",)
    STORE_ACTIONS = Option.STORE_ACTIONS + ("extend",)
    TYPED_ACTIONS = Option.TYPED_ACTIONS + ("extend",)
    ALWAYS_TYPED_ACTIONS = Option.ALWAYS_TYPED_ACTIONS +

    def take_action(self, action, dest, opt, value, valu
        if action == "extend":
            lvalue = value.split(",")
            values.ensure_value(dest, []).extend(lvalue)
        else:
            Option.take_action(
                self, action, dest, opt, value, values,
```

Detalles a tener en cuenta:

- `"extend"` espera un valor en la línea de comandos y también almacena ese valor en algún lugar, por lo que lo agregamos tanto a **`STORE_ACTIONS`** como a **`TYPED_ACTIONS`**.

- to ensure that **`optparse`** assigns the default type of `"string"` to `"extend"` actions, we put the `"extend"` action in **`ALWAYS_TYPED_ACTIONS`** as well.

- **`MyOption.take_action()`** implements just this one new action, and passes control back to **`Option.take_action()`** for the standard **`optparse`** actions.

- `values` es una instancia de la clase *optparse_parser.Values*, que proporciona el útil método **`ensure_value()`**. El método **`ensure_value()`** es en esencia lo mismo que `getattr()` pero con un mecanismo de seguridad agregado. Es llamado como:

```
values.ensure_value(attr, value)
```

Si el atributo `attr` de `values` no existe o es `None`, entonces *ensure_value()* primero lo establece en `value` y luego retorna el atributo actualizado (`value`). Esto es muy útil para acciones como `"extend"`, `"append"` y `"count"`, dado que todas ellas acumulan datos en una variable y esperan que esa variable sea de cierto tipo (una lista para las dos primeras, un número entero para la última). Usar el método **`ensure_value()`** significa que los scripts que usan tu acción no tienen que preocuparse por establecer un valor predeterminado para los destinos de opción en cuestión; simplemente pueden dejar el valor predeterminado como `None` y **`secure_value()`** se encargará de que todo esté correcto cuando sea necesario.

Excepciones

exception optparse.OptionError

Lanzada si se crea con argumentos inválidos o inconsistentes una instancia de **`Option`**.

exception optparse.OptionConflictError

Lanzada si se añaden opciones que entren en conflicto en una **`OptionParser`**.

exception optparse.OptionValueError

Lanzada si se encuentra una opción inválida en la línea de comandos.

exception optparse.BadOptionError

Lanzada si se pasa una opción inválida en la línea de comandos.

exception optparse.AmbiguousOptionError

Lanzada si se pasa una opción ambigua en la línea de comandos.

`getpass` — Portable password input

Código fuente: Lib/getpass.py [https://github.com/python/cpython/tree/3.14/Lib/getpass.py]

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

The **`getpass`** module provides two functions:

getpass.getpass(*prompt='Password: '*, *stream=None*, ***, *echo_char=None*)

> Solicita al usuario una contraseña sin hacer eco. Se solicita al usuario mediante la cadena *prompt*, que por defecto es `'Password: '`. En Unix, el indicador se escribe en el objeto similar a un archivo *stream* usando el controlador de errores de reemplazo si es necesario. *stream* toma por defecto el terminal de control (`/dev/tty`) o si no está disponible para `sys.stderr` (este argumento se ignora en Windows).
>
> The *echo_char* argument controls how user input is displayed while typing. If *echo_char* is `None` (default), input remains hidden. Otherwise, *echo_char* must be a single printable ASCII character and each typed character is replaced by it. For example, `echo_char='*'` will display asterisks instead of the actual input.
>
> Si la entrada sin *echo* no está disponible, getpass() recurre a imprimir un mensaje de advertencia en *stream* y leer de `sys.stdin` y lanza un **`GetPassWarning`**.

Nota

Si llama a getpass desde IDLE, la entrada puede realizarse en la terminal desde la que inició IDLE en lugar de en la ventana inactiva en sí.

Nota

On Unix systems, when *echo_char* is set, the terminal will be configured to operate in *noncanonical mode [https://manpages.debian.org/termios(3)#Canonical_and_noncanonical_mode]*. In particular, this means that line editing shortcuts such as `Ctrl`+`U` will not work and may insert unexpected characters into the input.

Distinto en la versión 3.14: Added the *echo_char* parameter for keyboard feedback.

exception getpass.GetPassWarning

Una subclase **`UserWarning`** lanzada cuando la entrada de la contraseña puede repetirse.

getpass.getuser()

Retorna el «nombre de inicio de sesión» del usuario.

This function checks the environment variables **`LOGNAME`**, **`USER`**, **`LNAME`** and **`USERNAME`**, in order, and returns the value of the first one which is set to a non-empty string. If none are set, the login name from the password database is returned on systems which support the **`pwd`** module, otherwise, an **`OSError`** is raised.

In general, this function should be preferred over **`os.getlogin()`**.

Distinto en la versión 3.13: Previously, various exceptions beyond just **`OSError`** were raised.

`fileinput` — Iterate over lines from multiple input streams

Código fuente: Lib/fileinput.py [https://github.com/python/cpython/tree/3.14/Lib/fileinput.py]

Este módulo implementa una clase auxiliar y funciones para escribir rápidamente un bucle sobre una entrada estándar o una lista de archivos. Si solo quiere leer o escribir un archivo, vea **`open()`**.

El uso común es:

```
import fileinput
for line in fileinput.input(encoding="utf-8"):
    process(line)
```

Esto itera sobre las líneas de todos los archivos enumerados en `sys.argv[1:]`, por defecto a `sys.stdin` si la lista está vacía. Si un nombre de archivo es `'-'`, también se reemplaza por `sys.stdin` y los argumentos opcionales *mode* y *openhook* se ignoran. Para especificar una lista alternativa de nombres de archivo, se pasa como primer argumento a **`input()`**. También se permite un único nombre de archivo.

Todos los archivos se abren en modo texto de manera predeterminada, pero puede anular esto especificando el parámetro *mode* en la llamada a **`input()`** o **`FileInput`**. Si se produce un error de E/S durante la apertura o lectura de un archivo, se lanza **`OSError`**.

Distinto en la versión 3.3: **`IOError`** solía ser lanzado; ahora es un alias de **`OSError`**.

Si `sys.stdin` se usa más de una vez, el segundo y siguientes usos no retornarán líneas, excepto tal vez para uso interactivo, o si se ha reiniciado explícitamente (por ejemplo, usando `sys.stdin.seek(0)`).

Los archivos vacíos se abren e inmediatamente se cierran; la única vez que su presencia en la lista de nombres de archivo es notable es cuando el último archivo abierto está vacío.

Las líneas se retornan con cualquier nueva línea intacta, lo que significa que la última línea en un archivo puede no tener una.

You can control how files are opened by providing an opening hook via the *openhook* parameter to `fileinput.input()` or `FileInput()`. The hook must be a function that takes two arguments, *filename* and *mode*, and returns an accordingly opened file-like object. If *encoding* and/or *errors* are specified, they will be passed to the hook as additional keyword arguments. This module provides a `hook_compressed()` to support compressed files.

La siguiente función es la interfaz principal de este módulo:

fileinput.input(*files=None, inplace=False, backup='', *, mode='r', openhook=None, encoding=None, errors=None*)

> Crea una instancia de la clase `FileInput`. La instancia se usará como estado global para las funciones de este módulo y también se volverá a usar durante la iteración. Los parámetros de esta función se pasarán al constructor de la clase **`FileInput`**.
>
> La instancia `FileInput` se puede usar como gestor de contexto en la declaración `with`. En este ejemplo, *input* se cierra después de salir de la instrucción **`with`**, incluso si se produce una excepción:
>
> ```
> with fileinput.input(files=('spam.txt', 'eggs.txt')
>     for line in f:
>         process(line)
> ```
>
> *Distinto en la versión 3.2:* Se puede usar como gestor de contexto.
>
> *Distinto en la versión 3.8:* Los parámetros de palabras clave

mode y *openhook* ahora son solo palabras clave.

Distinto en la versión 3.10: Los parámetros exclusivos de palabra clave *encoding* y *errors* son añadidos.

Las siguientes funciones utilizan el estado global creado por **`fileinput.input()`**; si no hay estado activo, es lanzado **`RuntimeError`**.

fileinput.filename()

Retorna el nombre del archivo que se está leyendo actualmente. Antes de leer la primera línea, retorna `None`.

fileinput.fileno()

Retorna el entero «file descriptor» para el archivo actual. Cuando no se abre ningún archivo (antes de la primera línea y entre archivos), retorna `-1`.

fileinput.lineno()

Retorna el número de línea acumulativa de la línea que se acaba de leer. Antes de que se haya leído la primera línea, retorna `0`. Después de leer la última línea del último archivo, retorna el número de línea de esa línea.

fileinput.filelineno()

Retorna el número de línea en el archivo actual. Antes de que se haya leído la primera línea, retorna `0`. Después de leer la última línea del último archivo, retorna el número de línea de esa línea dentro del archivo.

fileinput.isfirstline()

Retorna `True` si la línea que acaba de leer es la primera línea de su archivo; de lo contrario, retorna `False`.

fileinput.isstdin()

Retorna `True` si la última línea se leyó de `sys.stdin`, de lo contrario, retorna `False`.

fileinput.nextfile()

Cierra el archivo actual para que la próxima iteración lea la primera línea del siguiente archivo (si corresponde); las líneas no leídas del archivo no contarán para el recuento de líneas acumuladas. El nombre del archivo no se cambia hasta que se haya leído la primera línea del siguiente archivo. Antes de que se haya leído la primera línea, esta función no tiene efecto; no se puede usar para omitir el primer archivo. Después de leer la última línea del último archivo, esta función no tiene efecto.

fileinput.close()

Cierra la secuencia.

La clase que implementa el comportamiento de secuencia proporcionado por el módulo también está disponible para la subclasificación:

class fileinput.FileInput(*files=None*, *inplace=False*, *backup=''*, ***, *mode='r'*, *openhook=None*, *encoding=None*, *errors=None*)

La Clase `FileInput` es la implementación; sus métodos `filename()`, `fileno()`, `lineno()`, `filelineno()`, `isfirstline()`, `isstdin()`, `nextfile()` y `close()` corresponden a las funciones del mismo nombre en el módulo. Además es iterable y tiene un método `readline()` que retorna la siguiente línea de entrada. Se debe acceder a la secuencia en orden estrictamente secuencial; acceso aleatorio y `readline()` no se pueden mezclar.

Con *mode* puede especificar qué modo de archivo se pasará a `open()`. Debe ser uno de `'r'` y `'rb'`.

El *openhook*, cuando se proporciona, debe ser una función que tome dos argumentos, *filename* y *mode*, y retorne un objeto similar a un archivo abierto en consecuencia. No puede usar *inplace* y *openhook* juntos.

Puedes especificar *encoding* y *errors* que son pasados a `open()` o *openhook*.

Una instancia **`FileInput`** se puede usar como gestor de contexto en la instrucción **`with`**. En este ejemplo, *input* se cierra después de salir de la palabra **`with`**, incluso si se produce una excepción:

```
with FileInput(files=('spam.txt', 'eggs.txt')) as i
    process(input)
```

Distinto en la versión 3.2: Se puede usar como gestor de contexto.

Distinto en la versión 3.8: El parámetro de palabra clave *mode* y *openhook* ahora son solo palabras clave.

Distinto en la versión 3.10: Los parámetros exclusivos de palabra clave *encoding* y *errors* son añadidos.

Distinto en la versión 3.11: Los modos `'rU'` y `'U'` y el método **`__getitem__()`** han sido eliminados.

Filtrado al instante opcional: si el argumento de la palabra clave `inplace=True` se pasa a **`fileinput.input()`** o al constructor **`FileInput`**, el archivo se mueve a una copia de seguridad y la salida estándar es dirigida al archivo de entrada (si ya existe un archivo con el mismo nombre que el archivo de copia de seguridad, se reemplazará en silencio). Esto hace posible escribir un filtro que reescribe su archivo de entrada en su lugar. Si se proporciona el parámetro *backup* (generalmente como `backup='.<some extension>'`), este especifica la extensión para el archivo de respaldo y el archivo de respaldo permanece; de forma predeterminada, la extensión es `'.bak'` y se elimina cuando se cierra el archivo de salida. El filtrado en el lugar se desactiva cuando se lee la entrada estándar.

Este módulo proporciona los dos enlaces de apertura siguientes:

fileinput.hook_compressed(*filename*, *mode*, *, *encoding=None*, *errors=None*)

Abre de forma transparente archivos comprimidos con *gzip* y *bzip2* (reconocidos por las extensiones `'.gz'` and `'.bz2'`) utilizando los módulos **`gzip`** y **`bz2`**. Si la extensión del nombre de archivo no es `'.gz'` or `'.bz2'`, el archivo se abre normalmente (es decir, usando **`open()`** sin descompresión).

Los valores *encoding* y *errors* se pasan a **`io.TextIOWrapper`** para archivos comprimidos y para abrir archivos normales.

Ejemplo de uso: `fi = fileinput.FileInput(openhook=fileinput.hook_compres encoding="utf-8")`

Distinto en la versión 3.10: Los parámetros exclusivos de palabra clave *encoding* y *errors* son añadidos.

fileinput.hook_encoded(*encoding*, *errors=None*)

Retorna un enlace que abre cada archivo con **`open()`**, usando el *encoding* y *errors* dados para leer el archivo.

Ejemplo de uso: `fi = fileinput.FileInput(openhook=fileinput.hook_encoded "surrogateescape"))`

Distinto en la versión 3.6: Se agregó el parámetro opcional *errors*.

Obsoleto desde la versión 3.10: Esta función está en desuso ya que **`fileinput.input()`** y **`FileInput`** ahora tienen los parámetros *encoding* y *errors*.

`curses` — Terminal handling for character-cell displays

Source code: Lib/curses [https://github.com/python/cpython/tree/3.14/Lib/curses]

The `curses` module provides an interface to the curses library, the de-facto standard for portable advanced terminal handling.

Mientras que curses es más ampliamente usado en los ambientes Unix, las versiones están disponibles para Windows, DOS, y posiblemente para otros sistemas también. Esta extensión del módulo es diseñada para coincidir con el API de ncurses, una librería de código abierto almacenada en Linux y las variantes BSD de Unix.

Availability: not Android, not iOS, not WASI.

This module is not supported on mobile platforms or WebAssembly platforms.

This is an optional module. If it is missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

Availability: Unix.

Nota

Cuando la documentación menciona un *carácter*, esto puede ser especificado como un entero, una cadena Unicode de un carácter o una cadena de bytes de un byte.

Cuando la documentación menciona una *cadena de caracteres*, esto puede ser especificado como una cadena Unicode o una cadena de bytes.

Ver también

Módulo `curses.ascii`
Utilidades para trabajar con caracteres ASCII, independientemente de tu configuración local.

Módulo `curses.panel`
Una extensión de la pila de paneles que añade profundidad a las ventanas de curses.

Módulo `curses.textpad`
Widget de texto editable para apoyar curses **Emacs**- como enlaces.

Programación de curses con Python
Material del tutorial usando curses con Python, por *Andrew Kuchling* y *Eric Raymond*.

Funciones

The module `curses` defines the following exception:

exception curses.error

Una excepción se lanza cuando una función de la librería curses retorna un error.

Nota

Cuando los argumentos *x* o *y* para una función o un método son opcionales, se predetermina la ubicación actual del cursor. Cuando *attr* es opcional, por defecto es `A_NORMAL`.

The module `curses` defines the following functions:

curses.assume_default_colors(*fg*, *bg*, */*)

Allow use of default values for colors on terminals supporting this feature. Use this to support transparency in your application.

- Assign terminal default foreground/background colors to color number `-1`. So `init_pair(x, COLOR_RED, -1)` will initialize pair *x* as red on default background and `init_pair(x, -1, COLOR_BLUE)` will initialize pair *x* as default foreground on blue.
- Change the definition of the color-pair `0` to `(fg, bg)`.

Added in version 3.14.

curses.baudrate()

Retorna la velocidad de salida de la terminal en bits por segundo. En emuladores de terminal de software esto tendrá un alto valor fijado. Incluido por razones históricas; en tiempos pasados, esto fue usado para escribir los ciclos de salida por retrasos de tiempo y ocasionalmente para cambiar interfaces dependiendo de la velocidad en la línea.

curses.beep()

Emite un corto sonido de atención.

curses.can_change_color()

Retorna `True` o `False`, dependiendo ya sea que el programador puede cambiar los colores presentados por la terminal.

curses.cbreak()

Entra al modo *cbreak*. En el modo *cbreak* (a veces llamado modo «raro») del buffer de línea *tty* normal es apagado y los caracteres están disponibles para ser leídos uno por uno. Sin embargo, a diferencia del modo raw, los caracteres especiales (interrumpir, salir, suspender y control de flujo) retiene sus efectos en el manejador *tty* y el programa de llamada.

Llamando primero `raw()` luego `cbreak()` dejando la terminal en modo *cbreak*.

curses.color_content(*color_number*)

Retorna la intensidad de los componentes rojo, verde y azul (RGB) en el color *color_number*, que debe estar entre `0` y `COLORS-1`. Retorna una tupla de 3, que contiene los valores R,G,B para el color dado, que estará entre `0` (sin componente) y `1000` (cantidad máxima de componente).

curses.color_pair(*pair_number*)

Return the attribute value for displaying text in the specified color pair. Only the first 256 color pairs are supported. This attribute value can be combined with **`A_STANDOUT`**, **`A_REVERSE`**, and the other **`A_*`** attributes. `pair_number()` is the counterpart to this function.

curses.curs_set(*visibility*)

Configura el estado del cursor. *visibilidad* puede estar configurado para «0», «1» o «2», para invisible, normal o muy visible. Si la terminal soporta la visibilidad requerida, retorna el estado del cursor previo; de otra manera ejecuta una excepción. En muchos terminales, el modo «visible» es un cursor subrayado y el modo «muy visible» es un cursor de bloque.

curses.def_prog_mode()

Guardar el modo del terminal actual como el modo «program», el modo cuando el programa corriendo está usando curses. (Su contraparte es el modo «shell», para cuando el programa no está en curses.) Seguido de la llamada a `reset_prog_mode()` restaurará este modo.

curses.def_shell_mode()

Guarde el modo de terminal actual como el modo «shell», el modo en el que el programa en ejecución no utiliza curses. (Su contraparte es el modo «program», cuando el programa está usando las capacidades de curses.) Las llamadas subsecuentes a `reset_shell_mode()` restaurarán este modo.

curses.delay_output(*ms*)

Inserte una pausa en milisegundo *ms* en la salida.

curses.doupdate()

Actualiza la pantalla física. La librería curses mantiene dos estructuras de datos, uno representa el contenido de la pantalla física actual y una pantalla virtual representa el próximo estado deseado. La base `doupdate()` actualiza la pantalla física para comparar la pantalla virtual.

La pantalla virtual puede ser actualizada por una llamada `noutrefresh()` después de escribir las operaciones tales como `addstr()` ha sido ejecutada en una ventana. La llamada normal `refresh()` es simplemente `noutrefresh()` seguido por `doupdate()`; si tienes para actualizar múltiples ventanas, puedes aumentar el rendimiento y quizás reducir los parpadeos de la pantalla usando la llamada `noutrefresh()` en todas las ventanas, seguido por un simple `doupdate()`.

curses.echo()

Entrar en modo echo. En modo echo, cada caracter de entrada es repercutido a la pantalla como este es introducido.

curses.endwin()

Desinicializa la librería y retorne el terminal al estado normal.

curses.erasechar()

Retorne el carácter borrado del usuario actual como un objeto de bytes de un byte. Bajo el sistema de operaciones Unix esta es una propiedad de el controlador tty de el programa curses, y no es configurado por la librería curses en sí misma.

curses.filter()

La rutina **`filter()`**, si es usada, debe ser llamada antes que **`initscr()`** sea llamada. El efecto es que durante estas llamadas, **`LINES`** es configurada para `1`; las capacidades `clear`, `cup`, `cud`, `cud1`, `cuu1`, `cuu`, `vpa` son desactivadas; y la cadena `home` es configurada para el valor de `cr`. El efecto es que el cursor es confinado para la línea actual, y también las pantallas son actualizadas. Este puede ser usado para habilitar la línea editando el carácter en un tiempo sin tocar el resto de las pantallas.

curses.flash()

La pantalla Flash. Eso es, cambiar lo a video inverso y luego cambie lo de nuevo en un corto intervalo. Algunas personas prefieren como “campana visible” para la señal de atención audible producida por **`beep()`**.

curses.flushinp()

Vacíe todos los búferes de entrada. Esto desecha cualquier mecanografiado que ha sido escrito por el usuario y no ha sido aún procesado por el programa.

curses.getmouse()

Después de **`getch()`** retorna **`KEY_MOUSE`** para señalar un evento de mouse, se debe llamar a este método para recuperar el evento de mouse en cola, representado como una tupla de 5 `(id, x, y, z, bstate)`. *id* es un valor de ID que se utiliza para distinguir varios dispositivos, y *x*, *y*, *z* son las coordenadas del evento. (*z* no se usa actualmente.) *bstate* es un valor entero cuyos bits se establecerán para indicar el tipo de evento, y será el OR bit a bit de una o más de las siguientes constantes, donde *n* es el botón número de 1 a 5: **`BUTTONn_PRESSED`**, **`BUTTONn_RELEASED`**, **`BUTTONn_CLICKED`**, **`BUTTONn_DOUBLE_CLICKED`**, **`BUTTONn_TRIPLE_CLICKED`**, **`BUTTON_SHIFT`**, **`BUTTON_CTRL`**, **`BUTTON_ALT`**.

Distinto en la versión 3.10: Las constantes `BUTTON5_*` ahora están expuestas si son proporcionadas por la biblioteca curses subyacente.

curses.getsyx()

Retorna las coordenadas actuales del cursor en la pantalla virtual como una tupla `(y, x)`. Si **`leaveok`** es actualmente `True` entonces retorna `(-1,-1)`.

curses.getwin(*file*)

Read window related data stored in the file by an earlier **`window.putwin()`** call. The routine then creates and initializes a new window using that data, returning the new window object.

curses.has_colors()

Retorna `True` si el terminal puede desplegar colores, en caso contrario, retorna `False`.

curses.has_extended_color_support()

Retorna `True` si el módulo admite soporte de color extendido; de lo contrario retorna `False`. El soporte de color extendido permite más de 256 pares de colores para terminales que admiten más de 16 colores (por ejemplo, xterm-256color).

El soporte de color extendido requiere ncurses versión 6.1 o posterior.

Added in version 3.10.

curses.has_ic()

Retorna `True` si el terminal tiene la capacidad de insertar y eliminar caracteres. Esta función es incluida por razones históricas solamente, ya que todos los emuladores de la terminal de software modernos tienen tales capacidades.

curses.has_il()

Retorna `True` si la terminal tiene la capacidad de insertar y eliminar caracteres o pueden simularlos usando las regiones de desplazamiento. Esta función es incluida por razones históricas solamente, como todos los emuladores de terminales de software modernos tienen tales capacidades.

curses.has_key(*ch*)

Toma una clave valor *ch*, y retorna `True` si el tipo de terminal actual reconoce una clave con ese valor.

curses.halfdelay(*tenths*)

Usado por el modo de medio retardo, el cual es similar al modo *cbreak* en que los caracteres escritos por el usuario están disponibles inmediatamente para el programa. Sin embargo, después de bloquearlos por *tenths* décimas de segundos, se lanza una excepción si nada ha sido escrito. El valor de *tenths* debe ser un número entre `1` y `255`. Use **`nocbreak()`** para salir del modo de medio retardo.

curses.init_color(*color_number*, *r*, *g*, *b*)

Change the definition of a color, taking the number of the color to be changed followed by three RGB values (for the amounts of red, green, and blue components). The value of *color_number* must be between `0` and `COLORS - 1`. Each of *r*, *g*, *b*, must be a value between `0` and `1000`. When **`init_color()`** is used, all occurrences of that color on the screen immediately change to the new definition. This function is a no-op on most terminals; it is active only if **`can_change_color()`** returns `True`.

curses.init_pair(*pair_number*, *fg*, *bg*)

Change the definition of a color-pair. It takes three arguments: the number of the color-pair to be changed, the foreground color number, and the background color number. The value of *pair_number* must be between `1` and `COLOR_PAIRS - 1` (the `0` color pair can only be changed by **`use_default_colors()`** and **`assume_default_colors()`**). The value of *fg* and *bg* arguments must be between `0` and `COLORS - 1`, or, after calling **`use_default_colors()`** or **`assume_default_colors()`**, `-1`. If the color-pair was previously initialized, the screen is refreshed and all occurrences of that color-pair are changed to the new definition.

curses.initscr()

Inicializa la librería. Retorna un objeto window el cual representa a toda la pantalla.

Nota

Si hay un error al abrir el terminal, la librería curses subyacente puede causar que el interprete salga.

curses.is_term_resized(*nlines*, *ncols*)

Retorna `True` si **`resize_term()`** modificaría la estructura de la ventana, `False` en caso contrario.

curses.isendwin()

Retorna `True` si **`endwin()`** ha sido llamado (eso es que la librería curses ha sido desinicializada).

curses.keyname(*k*)

Retorna el nombre de la tecla enumerada *k* como un objeto de bytes. El nombre de una tecla que genera un carácter ASCII imprimible es el carácter de la tecla. El nombre de una combinación de teclas de control es un consistente objeto de bytes de dos bytes de un signo de intercalación (`b'^'`) seguido por el correspondiente carácter ASCII imprimible. El nombre de una combinación de tecla *alt* (128-255) es un objeto de bytes consistente del prefijo `b'M-'` seguido por el nombre del correspondiente carácter ASCII.

curses.killchar()

Retorna el carácter de eliminación de la línea actual del usuario como un objeto de bytes de un byte. Bajo el sistema operativo Unix esta es una propiedad del controlador *tty* del programa *curses*, y no está configurado por la librería *curses* por sí mismo.

curses.longname()

Retorna un objeto de bytes que contiene el campo de nombre largo *terminfo* que describe el terminal actual. La longitud máxima de una descripción verbosa es 128 caracteres. Esto es definido solamente después de la llamada a **`initscr()`**.

curses.meta(*flag*)

Si *flag* es `True`, permite caracteres de 8 bits para ser introducidos. Si *flag* es `False`, permite solamente caracteres de 7 bits.

curses.mouseinterval(*interval*)

Set the maximum time in milliseconds that can elapse between press and release events in order for them to be recognized as a click, and return the previous interval value. The default value is 200 milliseconds, or one fifth of a second.

curses.mousemask(*mousemask*)

Configure los eventos del mouse para ser reportados, y retorna una tupla `(availmask, oldmask)`. *availmask* indica cual de los eventos del ratón especificados pueden ser reportados; en caso de falla completa retorna `0`. *oldmask* es el valor previo de la máscara de evento del mouse de la ventana dada. Si esta función nunca es llamada, los eventos del mouse nunca son reportados.

curses.napms(*ms*)

Duerme durante *ms* milisegundos.

curses.newpad(*nlines*, *ncols*)

Crea y retorna un apuntador para una nueva estructura de datos de *pad* con el número dado de líneas y columnas. Retorna un *pad* como un objeto de ventana.

Una almohadilla es como una ventana, excepto que no es restringida por el tamaño de la pantalla, y no está necesariamente asociada con una parte particular de la pantalla. Las almohadillas pueden ser usadas cuando un ventana grande es necesitada, y solamente una parte de la ventana estará en la pantalla de una sola vez. Actualizaciones automáticas de almohadillas (tales desde el desplazamiento o haciendo eco de la entrada) no ocurre. Los métodos **`refresh()`** y **`noutrefresh()`** de una almohadilla requiere 6 argumentos para especificar la parte de la almohadilla a ser mostrada y la locación en la ventana que se utilizará para la visualización. Los argumentos son *pminrow*, *pmincol*, *sminrow*, *smincol*, *smaxrow*, *smaxcol*; el argumento *p* se refiere a la esquina superior izquierda de la región de la almohadilla a ser mostrada y el argumento *s* define una caja de recorte en la pantalla con la cual la región de la almohadilla será mostrada.

curses.newwin(*nlines*, *ncols*)
curses.newwin(*nlines*, *ncols*, *begin_y*, *begin_x*)

Retorna una nueva window, cuya esquina superior izquierda esta en `(begin_y, begin_x)`, y cuyo alto/ancho es *nlines*/*ncols*.

Por defecto, la ventana extenderá desde la posición especificada para la esquina inferior derecha de la pantalla.

curses.nl()

Ingrese al modo de línea nueva. Este modo pone la tecla de retorno en una nueva línea en la entrada, y traduce la nueva línea en retorno y avance de línea en la salida. El modo de nueva línea está inicialmente encendida.

curses.nocbreak()

Salir del modo *cbreak*. Retorne al modo normal «cooked» con la línea del búfer.

curses.noecho()

Salir del modo echo. El eco de los caracteres de entrada está desactivado.

curses.nonl()

Dejar el modo de nueva línea. Desactiva la traducción de retorno en una nueva línea en la entrada, y desactiva la traducción a bajo nivel de una nueva línea en nueva línea/ retorno en la salida (pero esto no cambia el comportamiento de `addch('\n')`, el cual siempre es el equivalente de retornar y avanzar la línea en la pantalla virtual). Con las traducciones apagadas, *curses* puede aumentar algunas veces la velocidad del movimiento vertical un poco; también, estará disponible para detectar la tecla de retorno en la entrada.

curses.noqiflush()

Cuando la rutina **`noqiflush()`** es usada, descarga normal de colas de entrada y salida asociadas con el `INTR`, los caracteres `QUIT` and `SUSP` no serán hechos. Puedes querer llamar **`noqiflush()`** en un manejador de señales si quieres que la salida continúe como si la interrupción no hubiera ocurrido después de que el manejador exista.

curses.noraw()

Salir del modo raw. Retorna al modo normal «cooked» con la línea del búfer.

curses.pair_content(*pair_number*)

Retorna una tupla `(fg, bg)` conteniendo los colores del par de color solicitado. El valor de *pair_number* debe ser entre `0` y `COLOR_PAIRS-1`.

curses.pair_number(*attr*)

Retorna el numero del conjunto de pares de colores para el valor del atributo *attr*. **`color_pair()`** es la contraparte de esta función.

curses.putp(*str*)

Equivalente a `tputs(str, 1, putchar)`; emite el valor de una capacidad especificada *terminfo* para el terminal actual. Nota que la salida de **`putp()`** siempre va a la salida estándar.

curses.qiflush([*flag*])

Si *flag* es `False`, el efecto es el mismo como llamar **`noqiflush()`**. Si *flag* es `True`, o el argumento no es proveído, la cola será nivelada cuando estos caracteres de control son leídos.

curses.raw()

Entrar al modo crudo. En el modo crudo, el almacenamiento en búfer de la línea normal y procesamiento de las teclas de interrupción, salida, suspensión y control de flujo son apagadas; los caracteres son presentados a la función de entrada de *curses* una por una.

curses.reset_prog_mode()

Restaura la terminal para el modo «program», anteriormente guardado por **`def_prog_mode()`**.

curses.reset_shell_mode()

Restablece el terminal al modo «shell» como lo guardó previamente **`def_shell_mode()`**.

curses.resetty()

Restablece el estado del modo del terminal para lo que esto fue en el último llamado a **`savetty()`**.

curses.resize_term(*nlines*, *ncols*)

La función *backend* usada por **`resizeterm()`**, caracteriza más de el trabajo; cuando se redimensiona la ventana, `resize_term()` los rellenos blancos de las áreas que son extendidas. La aplicación de llamada llenaría en estas áreas con datos apropiados. La función `resize_term()` intenta redimensionar todas las ventanas. Sin embargo, debido a la convención de llamadas del las almohadillas, esto no es posible para redimensionar estos sin interacciones adicionales con la aplicación.

curses.resizeterm(*nlines*, *ncols*)

Ajusta el tamaño al estándar y la ventana actual para las dimensiones especificadas, y ajusta otros datos contables usados por la librería *curses* que registra las dimensiones de la ventana (en particular el manejador SIGWINCH).

curses.savetty()

Guarda el estado actual del modo del terminal en un búfer, usable por `resetty()`.

curses.get_escdelay()

Recupera el valor establecido por `set_escdelay()`.

Added in version 3.9.

curses.set_escdelay(*ms*)

Establece el número de milisegundos de espera después de leer un carácter de escape, para distinguir entre un carácter de escape individual ingresado en el teclado de las secuencias de escape enviadas por el cursor y las teclas de función.

Added in version 3.9.

curses.get_tabsize()

Recupera el valor establecido por `set_tabsize()`.

Added in version 3.9.

curses.set_tabsize(*size*)

Establece el número de columnas utilizadas por la biblioteca de curses al convertir un carácter de tabulación en espacios, ya que agrega la tabulación a una ventana.

Added in version 3.9.

curses.setsyx(*y*, *x*)

Fija el cursor de la pantalla virtual para *y*, *x*. Si *y* y *x* son ambos «-1», entonces **`leaveok`** es configurado `True`.

curses.setupterm(*term=None*, *fd=-1*)

Inicializa la terminal. *term* es una cadena de caracteres dando el nombre de la terminal, o `None`; si es omitido o `None`, el valor de la variable de entorno **`TERM`** será usada. *fd* es el archivo descriptor al cual alguna secuencia de inicialización será enviada; si no es suministrada o `-1`, el archivo descriptor para `sys.stdout` será usado.

curses.start_color()

Debe ser llamado si el programador quiere usar colores, y antes de que cualquier otra rutina de manipulación de colores sea llamada. Esta es una buena práctica para llamar esta rutina inmediatamente después de **`initscr()`**.

`start_color()` initializes eight basic colors (black, red, green, yellow, blue, magenta, cyan, and white), and two global variables in the **`curses`** module, **`COLORS`** and **`COLOR_PAIRS`**, containing the maximum number of colors and color-pairs the terminal can support. It also restores the colors on the terminal to the values they had when the terminal was just turned on.

curses.termattrs()

Retorna un *OR* lógico de todos los atributos del video soportados por el terminal. Esta información es útil cuando un programa *curses* necesita completar el control sobre la apariencia de la pantalla.

curses.termname()

Retorna el valor de la variable de entorno **TERM**, como un objeto de bytes, trucado para 14 caracteres.

curses.tigetflag(*capname*)

Retorna el valor de la capacidad booleana correspondiente al nombre de la capacidad *terminfo capname* como un número entero. Retorna el valor `-1` si *capname* no es una capacidad booleana, o `0` si es cancelada o ausente desde la descripción de la terminal.

curses.tigetnum(*capname*)

Retorna el valor de la capacidad numérica correspondiente al nombre de la capacidad *terminfo capname* como un entero. Regresa el valor `-2` si *capname* no es una capacidad numérica, o `-1` si esta es cancelada o ausente desde la descripción del terminal.

curses.tigetstr(*capname*)

Retorna el valor de la capacidad de la cadena de caracteres correspondiente al nombre de la capacidad *terminfo capname* como un objeto de bytes. Retorna `None` si *capname* no es una «capacidad de cadena de caracteres» de *terminfo*, o es cancelada o ausente desde la descripción de la terminal.

curses.tparm(*str*[, ...])

Instancia del objeto de bytes *str* con los parámetros suministrados, dónde *str* sería un cadena de caracteres parametrizada obtenida desde la base de datos de *terminfo*. E.g. `tparm(tigetstr("cup"), 5, 3)` podría resultar en `b'\033[6;4H'`, el resultado exacto depende del tipo de terminal.

curses.typeahead(*fd*)

Especifica que el descriptor de archivo *fd* es usado para la verificación anticipada. Si *fd* es «-1», entonces no se realiza ninguna verificación anticipada.

La librería *curses* hace «la optimización del rompimiento de línea» por la búsqueda para mecanografiar periódicamente mientras se actualiza la pantalla. Si la entrada es encontrada, y viene desde un *tty*, la actualización actual es pospuesta hasta que se vuelva a llamar la actualización, permitiendo la respuesta más rápida para comandos escritos en avance. Esta función permite especificar un archivo diferente al descriptor para la verificación anticipada.

curses.unctrl(*ch*)

Regresa un objeto de bytes el cual es una representación imprimible del carácter *ch*. Los caracteres de control son representados como un símbolo de intercalación seguido del carácter, por ejemplo como `b'^C'`. La impresión de caracteres son dejados como están.

curses.ungetch(*ch*)

Presiona *ch* para que el siguiente `getch()` lo retorne.

Nota

Solamente un *ch* puede ser colocado antes **`getch()`** es llamada.

curses.update_lines_cols()

Update the `LINES` and `COLS` module variables. Useful for detecting manual screen resize.

Added in version 3.5.

curses.unget_wch(*ch*)

Presiona *ch* para que el siguiente `get_wch()` lo retorne.

Nota

Solamente un *ch* puede ser presionado antes **`get_wch()`** es llamada.

Added in version 3.3.

curses.ungetmouse(*id*, *x*, *y*, *z*, *bstate*)

Coloca un evento `KEY_MOUSE` en la cola de entrada, asociando el estado de los datos dados con esto.

curses.use_env(*flag*)

Si es usado, esta función sería llamada antes de `initscr()` o *newterm* son llamados. Cuando *flag* es `False`, los valores de líneas y columnas especificadas en la base de datos *terminfo* será usado, incluso si las variables de entorno **`LINES`** y **`COLUMNS`** (usadas por defecto) son configuradas, o si *curses* está corriendo en una ventana (en cuyo caso el comportamiento por defecto sería usar el tamaño de ventana si **`LINES`** y **`COLUMNS`** no están configuradas).

curses.use_default_colors()

Equivalent to `assume_default_colors(-1, -1)`.

curses.wrapper(*func*, /, **args*, ***kwargs*)

Inicializa *curses* y llama a otro objeto invocable, *func*, el cual debería ser el resto de tu aplicación que usa *curses*. Si la aplicación lanza una excepción, esta función restaurará la terminal para un estado sano antes de re-lanzar la excepción y generar un una pista. El objeto invocable *func* es entonces pasado a la ventana principal "stdscr" como su primer argumento, seguido por algún otro argumento pasado a **`wrapper()`**. Antes de llamar a *func*, **`wrapper()`** habilita el modo *cbreak*, desactiva echo, permite el teclado del terminal, e inicializa los colores si la terminal tiene soporte de color. En salida (ya sea normal o por excepción) esto restaura el modo *cooked*, habilita el echo, y desactiva el teclado del terminal.

Objetos de ventana

class curses.window

Los objetos ventana, retornados por `initscr()` y `newwin()` anteriores, tienen los siguientes métodos y atributos:

window.addch(*ch*[, *attr*])

window.addch(*y*, *x*, *ch*[, *attr*])

Pinta el carácter *ch* en `(y, x)` con atributos *attr*, sobrescribiendo cualquier carácter pintado previamente en esa ubicación. De forma predeterminada, la posición y los atributos del carácter son la configuración actual del objeto ventana.

Nota

Escribiendo afuera de la ventana, sub-ventana, o *pad* genera un `curses.error`. Intentando escribir en la esquina inferior derecha de una ventana, sub-ventana, o *pad* causará una excepción a ser generada después de que el carácter es pintado.

window.addnstr(*str*, *n*[, *attr*])

window.addnstr(*y*, *x*, *str*, *n*[, *attr*])

Pintar como máximo *n* caracteres de la cadena de texto *str* en «(y,x)» con atributos *attr*, sobrescribiendo algo previamente en la pantalla.

window.addstr(*str*[, *attr*])

window.addstr(*y*, *x*, *str*[, *attr*])

Dibuja la cadena de caracteres *str* en «(y,x)» con atributos *attr*, sobrescribiendo cualquier cosa previamente en la pantalla.

Nota

- Escribiendo afuera de la ventana, sub-ventana, o *pad* genera un `curses.error`. Intentando escribir en la esquina inferior derecha de una ventana, sub-ventana, o *pad* causará una excepción a ser generada

después de que la cadena de caracteres es pintada.
- Un bug en *ncurses* [https://bugs.python.org/issue35924], el *backend* para este módulo de Python, puede causar *SegFaults* cuando re-dimensionan las ventanas. Esto es reparado en ncurses-6.1-20190511. Si tu estás atascado con un *ncurses* anterior, puedes evitar desencadenar este si no llamas `addstr()` con un *str* que tiene embebido nuevas líneas. En su lugar, llama **`addstr()`** separadamente por cada línea.

window.attroff(*attr*)

Remueve el atributo *attr* desde el conjunto «background» aplicado a todos los escritos para la ventana actual.

window.attron(*attr*)

Add attribute *attr* to the «background» set applied to all writes to the current window.

window.attrset(*attr*)

Establezca el conjunto de atributos «background para *attr*. Este conjunto es inicialmente «0» (sin atributos).

window.bkgd(*ch*[, *attr*])

Configura la propiedad de fondo de la ventana para el carácter *ch*, con atributos *atte*. El cambio es entonces aplicado para la posición de cada carácter en esa ventana:

- El atributo de cada carácter en la ventana es cambiado por el nuevo atributo de fondo.
- Dónde quiera que el carácter del fondo anterior aparezca, es cambiado al nuevo carácter de fondo.

window.bkgdset(*ch*[, *attr*])

Configura el fondo de la ventana. Un fondo de ventana consiste de un carácter y cualquier combinación de atributos. La parte del atributo del fondo es combinado (OR” ed) con todos los caracteres que no son blancos escritos en la ventana. Tanto las partes del carácter y del atributo del fondo son combinados con los caracteres en blanco. El fondo se convierte en una propiedad del carácter y se mueve con el carácter a través de cualquier operación de desplazamiento e inserción/eliminación de línea/carácter.

window.border([*ls*[, *rs*[, *ts*[, *bs*[, *tl*[, *tr*[, *bl*[, *br*]]]]]]]])

Dibuja un borde alrededor de las orillas de la ventana. Cada parámetro especifica el carácter a usar para una parte específica del borde; ve la tabla abajo para más detalles.

Nota

Un valor `0` para cualquier parámetro causará el carácter por defecto a ser usado para ese parámetro. parámetros de palabras clave pueden *no* ser usados. Los valores predeterminados son listados en esta tabla:

Parámetro	**Descripción**	**Valor por defecto**
ls	Lado izquierdo	`ACS_VLINE`
rs	Lado derecho	`ACS_VLINE`
ts	Arriba	`ACS_HLINE`
bs	Abajo	`ACS_HLINE`
tl	Esquina superior izquierda	`ACS_ULCORNER`
tr	Esquina superior derecha	`ACS_URCORNER`
bl	Esquina inferior izquierda	`ACS_LLCORNER`
br	Esquina inferior derecha	`ACS_LRCORNER`

window.box([*vertch*, *horch*])

Similar a `border()`, pero ambos *ls y *rs* son *vertch* y ambos *ts* y *bs* son *horch*. Los caracteres de esquina predeterminados son siempre usados por esta función.

window.chgat(*attr*)
window.chgat(*num*, *attr*)
window.chgat(*y*, *x*, *attr*)
window.chgat(*y*, *x*, *num*, *attr*)

Configura los atributos de *num* de caracteres en la posición del cursor, o una posición `(y, x)` si es suministrado. Si *num* no es dado o es `-1`, el atributo será configurado en todos los caracteres para el final de la línea. Esta función mueve el cursor a la posición `(y, x)` si es suministrado. La línea cambiada será tocada usando el método **`touchline()`** tal que el contenido será vuelto a mostrar por la actualización de la siguiente ventana.

window.clear()

Semejante a **`erase()`**, pero también causa que toda la ventana sea repintada sobre la siguiente llamada para **`refresh()`**.

window.clearok(*flag*)

Si *flag* es `True`, la siguiente llamada para **`refresh()`** limpiará la ventana completamente.

window.clrtobot()

Borrar desde el cursor hasta el final de la ventana: todas las líneas bajo el cursor son eliminadas, y entonces el equivalente de **`clrtoeol()`** es realizado.

window.clrtoeol()

Borrar desde el cursor hasta el final de la línea.

window.cursyncup()

Actualice la posición actual del cursor de todos los ancestros de la ventana para reflejar la posición actual del cursor de la ventana.

window.delch([*y*, *x*])

Borrar cualquier carácter en «(y,x)».

window.deleteln()

Borra la línea bajo el cursor. Todas las líneas siguientes son movidas una línea hacía arriba.

window.derwin(*begin_y*, *begin_x*)
window.derwin(*nlines*, *ncols*, *begin_y*, *begin_x*)

Una abreviación para «ventana derivada», **`derwin()`** es el mismo cómo llamar **`subwin()`**, excepto que *begin_y* y *begin_x* son relativos al origen de la ventana, más bien relativo a la pantalla completa. Retorna un objeto ventana para la ventana derivada.

window.echochar(*ch*[, *attr*])

Añada el carácter *ch* con atributo *attr*, e inmediatamente llame a **`refresh()`** en la ventana.

window.enclose(*y*, *x*)

Pruebe si el par dado de las coordenadas de celda del carácter relativas de la ventana son encerrados por la ventana dada, retornando `True` o `False`. Esto es útil para determinar que subconjunto de las ventanas de pantalla encierran la locación de un evento del mouse.

Distinto en la versión 3.10: Anteriormente retornaba `1` o `0` en lugar de `True` o `False`.

window.encoding

Encoding used to encode method arguments (Unicode strings and characters). The encoding attribute is inherited from the parent window when a subwindow is created, for example with **`window.subwin()`**. By default, current locale encoding is used (see **`locale.getencoding()`**).

Added in version 3.3.

window.erase()

Limpiar la ventana.

window.getbegyx()

Return a tuple `(y, x)` of coordinates of upper-left corner.

window.getbkgd()

Retorna el par carácter/atributo dada la ventana actual.

window.getch([*y*, *x*])

Obtener un carácter. Nota que el entero retornado *no* tiene que estar en el rango ASCII: teclas de función, claves de teclado y los demás son representados por números mayores que 255. En el modo sin demora, retorna `-1` si no hay entrada, en caso contrario espere hasta que una tecla es presionada.

window.get_wch([*y*, *x*])

Obtener un carácter amplio. Retorna un carácter para la mayoría de las teclas, o un entero para las teclas de función, teclas del teclado, y otras teclas especiales. En modo de no retorna, genera una excepción si no hay entrada.

Added in version 3.3.

window.getkey([*y*, *x*])

Obtener un carácter, retornando una cadena de caracteres en lugar de un entero, como **`getch()`** lo hace. Las teclas de función, teclas del teclado y otras teclas especiales retornan una cadena *multibyte* conteniendo el nombre de la tecla. En modo de no retardo, genera una excepción si no hay entrada.

window.getmaxyx()

Retorna una tupla «(y,x)» de el alto y ancho de la ventana.

window.getparyx()

Retorne las coordenadas iniciales de esta ventana relativa para su ventana padre como una tupla `(y, x)`. Retorna `(-1, -1)` si esta ventana no tiene padre.

window.getstr()
window.getstr(*n*)
window.getstr(*y*, *x*)
window.getstr(*y*, *x*, *n*)

Read a bytes object from the user, with primitive line editing capacity. The maximum value for *n* is 2047.

Distinto en la versión 3.14: The maximum value for *n* was increased from 1023 to 2047.

window.getyx()

Retorna una tupla `(y, x)` de la posición actual del cursor relativa para la esquina superior izquierda de la ventana.

window.hline(*ch*, *n*)
window.hline(*y*, *x*, *ch*, *n*)

Muestra una línea horizontal iniciando en `(y, x)` con longitud *n* consistente de el carácter *ch*.

window.idcok(*flag*)

Si *flag* es `False`, *curses* ya no considera el uso de la función de insertar/eliminar caracteres de hardware del terminal; si *flag* `True`, el uso de inserción y borrado de caracteres está habilitado. Cuando *curses* es inicializado primero, el uso de caracteres de inserción / borrado está habilitado por defecto.

window.idlok(*flag*)

If *flag* is `True`, **`curses`** will try and use hardware line editing facilities. Otherwise, line insertion/deletion are disabled.

window.immedok(*flag*)

Si *flag* es `True`, cualquier cambio en la imagen de la ventana automáticamente causará que la ventana sea refrescada; ya no tienes que llamar a **`refresh()`** por ti mismo. Sin embargo, esto puede degradar el rendimiento considerablemente, debido a las repeticiones de llamada a *wrefresh*. Esta opción está deshabilitada por defecto.

window.inch([*y*, *x*])

Retorna el carácter en la posición dada en la ventana. Los 8bits inferiores son los caracteres propiamente dichos, y los bits superiores son los atributos.

window.insch(*ch*[, *attr*])
window.insch(*y*, *x*, *ch*[, *attr*])

Pinta el carácter *ch* en `(y, x)` con atributos *attr*, moviendo la línea desde la posición *x* a la derecha un carácter.

window.insdelln(*nlines*)

Inserta *nlines* líneas en la ventana especificada arriba de la línea actual. Las *nlines* líneas de fondo se pierden. Para *nlines* negativos, elimine *nlines* líneas comenzando con la que está debajo del cursor, y mueve las restantes hacia arriba. Se borran las *nlines* líneas inferiores. La posición del cursor actual se mantiene igual.

window.insertln()

Inserte una línea en blanco bajo el cursos. Las líneas siguientes se moverán una línea hacía abajo.

window.insnstr(*str*, *n*[, *attr*])
window.insnstr(*y*, *x*, *str*, *n*[, *attr*])

Inserte una cadena de caracteres (tantos caracteres quepan en la línea) antes los caracteres bajo el cursor, sobre los *n* caracteres. Si *n* es cero o negativo, la cadena entera es insertada. Todos los caracteres a la derecha de el cursor son desplazados a la derecha, perdiéndose los caracteres de la derecha en la línea. La posición del cursor no cambia (después de mover a *y*, *x*, si es especificado).

window.insstr(*str*[, *attr*])

window.insstr(*y*, *x*, *str*[, *attr*])

Inserte una cadena de caracteres (tantos caracteres encajen en la línea) antes los caracteres bajo el cursor. Todos los caracteres a la derecha de el cursor son desplazados a la derecha, perdiéndose los caracteres de la derecha en línea.

window.instr([*n*])
window.instr(*y*, *x*[, *n*])

Return a bytes object of characters, extracted from the window starting at the current cursor position, or at *y*, *x* if specified. Attributes are stripped from the characters. If *n* is specified, **`instr()`** returns a string at most *n* characters long (exclusive of the trailing NUL). The maximum value for *n* is 2047.

Distinto en la versión 3.14: The maximum value for *n* was increased from 1023 to 2047.

window.is_linetouched(*line*)

Retorna `True` si la línea especificada fue modificada desde la última llamada a **`refresh()`**; de otra manera retorna `False`. Genera una excepción **`curses.error`** si *line* no es válida para la ventana dada.

window.is_wintouched()

Retorna `True` si la ventana especificada fue modificada desde la última llamada para **`refresh()`**; en caso contrario, retorna `False`.

window.keypad(*flag*)

If *flag* is `True`, escape sequences generated by some keys (keypad, function keys) will be interpreted by **`curses`**. If *flag* is `False`, escape sequences will be left as is in the input stream.

window.leaveok(*flag*)

Si *flag* es `True`, el cursor es dejado donde está en la actualización, por ejemplo estando en «cursor position.» Esto reduce el movimiento del cursor siempre que sea posible. Si es posible, el cursor se hará invisible.

Si *flag* es `False`, el cursor siempre estará en «cursor position» después de una actualización.

window.move(*new_y*, *new_x*)

Mueve el cursor a `(new_y, new_x)`.

window.mvderwin(*y*, *x*)

Mueve la ventana adentro de su ventana padre. Los parámetros relativos a la pantalla de la ventana no son cambiados. Esta rutina es usada para mostrar diferentes partes de la ventana padre en la misma posición física en la pantalla.

window.mvwin(*new_y*, *new_x*)

Mueve la ventana a su esquina superior izquierda que está en `(new_y,new_x)`.

window.nodelay(*flag*)

Si *flag* es `True`, **`getch()`** no será bloqueada.

window.notimeout(*flag*)

Si *flag* es `True`, las secuencias de escape no serán agotadas.

Si *flag* es `False`, después de pocos milisegundos, una secuencia de escape no será interpretada, y será dejada en el flujo de entrada como está.

window.noutrefresh()

Marque para actualizar pero espere. Esta función actualiza la estructura de datos representando el estado deseado de la ventana, pero no fuerza una actualización en la pantalla física. Para realizar eso, llame **`doupdate()`**.

window.overlay(*destwin*[, *sminrow*, *smincol*, *dminrow*, *dmincol*, *dmaxrow*, *dmaxcol*])

Cubre la ventana en la parte superior de *destwin*. La ventana no necesita ser del mismo tamaño, solamente se solapa la región copiada. Esta copia no es destructiva, lo que significa que el carácter de fondo actual no sobre-escribe el contenido viejo de *destwin*.

Para obtener el control de grano fino sobre la región copiada, la segunda forma de `overlay()` puede ser usada. *sminrow* y *smincol* son las coordenadas superior izquierda de la ventana de origen, y las otras variables marcan un rectángulo en la ventana destino.

window.overwrite(*destwin*[, *sminrow*, *smincol*, *dminrow*, *dmincol*, *dmaxrow*, *dmaxcol*])

Sobre-escribe la ventana en la parte superior de *destwin*. La ventana no necesita ser del mismo tamaño, en cuyo caso solamente se sobrepone la región que es copiada. Esta copia es destructiva, lo cual significa que el carácter de fondo actual sobre-escribe el contenido viejo de *destwin*.

Para obtener el control de grano fino sobre la región copiada, la segunda forma de `overwrite()` puede ser usada. *sminrow* y *smincol* son las coordenadas superior izquierda de la ventana origen, las otras variables marcan un rectángulo en la ventana de destino.

window.putwin(*file*)

Escribe todos los datos asociados con la ventana en el objeto de archivo proveído. Esta información puede estar después recuperada usando la función `getwin()`.

window.redrawln(*beg*, *num*)

Indica que el *num* de líneas de la pantalla, empiezan en la línea *beg*, son corruptos y deberían ser completamente redibujado en la siguiente llamada `refresh()`.

window.redrawwin()

Toca la ventana completa, causando que se vuelva a dibujar completamente en la siguiente llamada `refresh()` .

window.refresh([*pminrow*, *pmincol*, *sminrow*, *smincol*, *smaxrow*, *smaxcol*])

Actualiza la pantalla inmediatamente (sincronice la pantalla actual con los métodos previos de dibujar/borrar).

Los 6 argumentos opcionales pueden ser solamente especificados cuando la ventana es una almohadilla creada con `newpad()`. Los parámetros adicionales son necesarios para indicar que parte de la almohadilla y pantalla son envueltos. *pminrow* y *pmincol* especifica la esquina superior de la mano izquierda del rectángulo a ser mostrado en el *pad*. *sminrow*, *smincol*, *smaxrow*, y *smaxcol* especifica los bordes del rectángulo a ser mostrados en la pantalla. La esquina inferior de la mano derecha del rectángulo a ser mostrado en el *pad* es calculado desde las coordenadas de la pantalla, desde que el rectángulo debe ser del mismo tamaño. Ambos rectángulos deben ser completamente contenidos con su estructura respectiva. Valores negativos de *pminrow*, *pmincol*, *sminrow*, o *smincol* son tratados como si fueran cero.

window.resize(*nlines*, *ncols*)

Redistribuir el almacenamiento para una ventana *curses* para ajustar su dimensión para los valores especificados. Si la dimensión también es mayor que los valores actuales, los datos de la ventana serán llenados con espacios en blanco que tiene la ejecución del fondo actual (como ajustado por `bkgdset()`) uniéndolos.

window.scroll([*lines* = *1*])

Desplace la pantalla o la región de desplazamiento hacia arriba por *lines* líneas.

window.scrollok(*flag*)

Control que sucede cuando el cursor de una ventana es movida fuera del borde de la ventana o región de desplazamiento, también como un resultado de una acción de nueva línea en la línea inferior, o escribiendo el último carácter de la última línea. Si *flag* es `False`, el cursor está a la izquierda sobre la línea inferior. Si *flag* es `True`, la ventana será desplazada hacía arriba por una línea. Note que en orden para obtener el efecto del desplazamiento físico en el terminal, también será necesario llamar **`idlok()`**.

window.setscrreg(*top*, *bottom*)

Configura la región de desplazamiento desde la línea *top* hasta la línea *bottom*. Todas las acciones de desplazamiento tomarán lugar en esta región.

window.standend()

Desactive el atributo destacado. En algunos terminales esto tiene el efecto secundario de desactivar todos los atributos.

window.standout()

Active el atributo *A_STANDOUT*.

window.subpad(*begin_y*, *begin_x*)
window.subpad(*nlines*, *ncols*, *begin_y*, *begin_x*)

Retorna la sub-ventana, cuya esquina superior izquierda esta en `(begin_y, begin_x)`, y cuyo ancho/alto es *ncols*/*nlines*.

window.subwin(*begin_y*, *begin_x*)
window.subwin(*nlines*, *ncols*, *begin_y*, *begin_x*)

Retorna la sub-ventana, cuya esquina superior izquierda esta en `(begin_y, begin_x)`, y cuyo ancho/alto es *ncols*/*nlines*.

Por defecto, la sub-ventana extenderá desde la posición especificada para la esquina inferior derecha de la ventana.

window.syncdown()

Toca cada ubicación en la ventana que ha sido tocado por alguna de sus ventanas padres. Esta rutina es llamada por **`refresh()`**, esto casi nunca debería ser necesario para llamarlo manualmente.

window.syncok(*flag*)

Si *flag* es `True`, entonces **`syncup()`** es llamada automáticamente cuando hay un cambio en la ventana.

window.syncup()

Toque todas las locaciones de los ancestros de la ventana que han sido cambiados en la ventana.

window.timeout(*delay*)

Configura el bloqueo o no bloqueo del comportamiento para la ventana. Si el *delay* es negativo, bloqueando la lectura usada (el cual esperará indefinidamente para la entrada). Si *delay* es cero, entonces no se bloqueará la lectura usada, y **`getch()`** retornará `-1` si la entrada no está esperando. Si *delay* es positivo, entonces **`getch()`** se bloqueará por *delay* milisegundos, y retorna `-1` si aún no entra en el final de ese tiempo.

window.touchline(*start*, *count*[, *changed*])

Supone *count* líneas han sido cambiadas, iniciando con la línea *start*. Si *changed* es suministrado, específica que las líneas afectadas son marcadas como hayan sido cambiadas (*changed*=`True`) o no cambiadas (*changed*=`False`).

window.touchwin()

Suponga que toda la ventana ha sido cambiada, con el fin de optimizar el dibujo.

window.untouchwin()

Marque todas las líneas en la ventana como no cambiadas desde la última llamada para **`refresh()`**.

window.vline(*ch*, *n*[, *attr*])

window.vline(*y*, *x*, *ch*, *n*[, *attr*])

Display a vertical line starting at `(y, x)` with length *n* consisting of the character *ch* with attributes *attr*.

Constantes

The **`curses`** module defines the following data members:

curses.ERR

Algunas rutinas de *curses* que retornan un entero, tales como **`getch()`**, retorna **`ERR`** sobre el fallo.

curses.OK

Algunas rutinas de *curses* que retornan un entero, tales como **`napms()`**, retorna **`OK`** tras el éxito.

curses.version

A bytes object representing the current version of the module.

curses.ncurses_version

Una tupla nombrada contiene los 3 componentes de la versión de la librería *ncurses major*, *minor*, y *patch*. Todos los valores son enteros. Los componentes pueden también ser accedidos por nombre, así `curses.ncurses_version[0]` es equivalente a `curses.ncurses_version.major` y así.

Disponibilidad: si la librería ncurses es usada.

Added in version 3.8.

curses.COLORS

The maximum number of colors the terminal can support. It is defined only after the call to **`start_color()`**.

curses.COLOR_PAIRS

> The maximum number of color pairs the terminal can support. It is defined only after the call to `start_color()`.

curses.COLS

> The width of the screen, i.e., the number of columns. It is defined only after the call to `initscr()`. Updated by `update_lines_cols()`, `resizeterm()` and `resize_term()`.

curses.LINES

> The height of the screen, i.e., the number of lines. It is defined only after the call to `initscr()`. Updated by `update_lines_cols()`, `resizeterm()` and `resize_term()`.

Algunas constantes están disponibles para especificar los atributos del celdas de caracteres. Las constantes exactas disponible dependen del sistema.

Atributo	**Significado**
curses.A_ALTCHARSET	Modo de conjunto de caracteres alternativo
curses.A_BLINK	Modo parpadeo
curses.A_BOLD	Modo negrita
curses.A_DIM	Modo tenue
curses.A_INVIS	Modo invisible o en blanco
curses.A_ITALIC	Modo cursiva
curses.A_NORMAL	Atributo normal
curses.A_PROTECT	Modo protegido
curses.A_REVERSE	Fondo inverso y colores de primer plano
curses.A_STANDOUT	Modo destacado
curses.A_UNDERLINE	Modo subrayado
curses.A_HORIZONTAL	Resaltado horizontal
curses.A_LEFT	Resaltado a la izquierda
curses.A_LOW	Resaltado bajo
curses.A_RIGHT	Resaltado a la derecha
curses.A_TOP	Resaltado arriba
curses.A_VERTICAL	Resaltado vertical

Added in version 3.7: `A_ITALIC` fue añadido.

Varias constantes están disponibles para extraer los atributos correspondientes retornados por algunos métodos.

Máscara de bits	**Significado**
`A_ATTRIBUTES`	Máscara de bits para extraer atributos
`A_CHARTEXT`	Máscara de bits para extraer un caracter
`A_COLOR`	Máscara de bits para extraer información de campo de pares de colores

Las teclas se denominan constantes enteras con nombres que comienzan con `KEY_`. Las teclas exactas disponibles son dependientes del sistema.

Clave constante	**Clave**
`KEY_MIN`	Valor mínimo de la clave
`KEY_BREAK`	Clave rota (no confiable)
`KEY_DOWN`	Flecha hacia abajo
`KEY_UP`	Flecha hacia arriba
`KEY_LEFT`	Flecha hacia la izquierda
`KEY_RIGHT`	Flecha hacia la derecha
`KEY_HOME`	Tecla de inicio (flecha hacia arriba + flecha hacia la izquierda)
`KEY_BACKSPACE`	Retroceso (no confiable)
`KEY_F0`	Teclas de función. Más de 64 teclas de función son soportadas.
`KEY_Fn`	Valor de tecla de función *n*
`KEY_DL`	Borrar línea
`KEY_IL`	Insertar línea
`KEY_DC`	Borrar caracter
`KEY_IC`	Insertar caracter o ingresara al modo de inserción
`KEY_EIC`	Salir del modo de inserción de caracteres
`KEY_CLEAR`	Limpiar pantalla
`KEY_EOS`	Limpiar al final de pantalla
`KEY_EOL`	Limpiar al final de la línea
`KEY_SF`	Desplazar una línea hacia adelante
`KEY_SR`	Desplazar una línea hacia atrás (reversa)
`KEY_NPAGE`	Página siguiente
`KEY_PPAGE`	Página anterior
`KEY_STAB`	Establecer pestaña
`KEY_CTAB`	Limpiar pestaña
`KEY_CATAB`	Limpiar todas las pestañas

curses.KEY_ENTER	Ingresar o enviar (no confiable)
curses.KEY_SRESET	Reinicio suave (parcial) (no confiable)
curses.KEY_RESET	Reinicio o reinicio fuerte (no confiable)
curses.KEY_PRINT	Imprimir
curses.KEY_LL	Inicio abajo o abajo (abajo a la izquierda)
curses.KEY_A1	Superior izquierda del teclado
curses.KEY_A3	Superior derecha de el teclado
curses.KEY_B2	Centro del teclado
curses.KEY_C1	Inferior izquierdo del teclado
curses.KEY_C3	Inferior derecho del teclado
curses.KEY_BTAB	Pestaña trasera
curses.KEY_BEG	Mendigar (comienzo)
curses.KEY_CANCEL	Cancelar
curses.KEY_CLOSE	Cerrar
curses.KEY_COMMAND	Cmd (Comando)
curses.KEY_COPY	Copiar
curses.KEY_CREATE	Crear
curses.KEY_END	Final
curses.KEY_EXIT	Salir
curses.KEY_FIND	Encontrar
curses.KEY_HELP	Ayudar
curses.KEY_MARK	Marca
curses.KEY_MESSAGE	Mensaje
curses.KEY_MOVE	Mover
curses.KEY_NEXT	Siguiente
curses.KEY_OPEN	Abrir
curses.KEY_OPTIONS	Opciones
curses.KEY_PREVIOUS	Anterior
curses.KEY_REDO	Rehacer
curses.KEY_REFERENCE	Referencia
curses.KEY_REFRESH	Refrescar
curses.KEY_REPLACE	Reemplazar
curses.KEY_RESTART	Reiniciar
curses.KEY_RESUME	Resumir
curses.KEY_SAVE	Guardar
curses.KEY_SBEG	Mendicidad desplazada (comienzo)
curses.KEY_SCANCEL	Cancelar cambiado
curses.KEY_SCOMMAND	Comando cambiado
curses.KEY_SCOPY	Copiado cambiado
curses.KEY_SCREATE	Crear con desplazamiento

KEY_SDC	Borrar carácter desplazado
KEY_SDL	Borrar línea desplazada
KEY_SELECT	Seleccionar
KEY_SEND	Final desplazado
KEY_SEOL	Limpiar línea desplazada
KEY_SEXIT	Salida desplazada
KEY_SFIND	Búsqueda desplazada
KEY_SHELP	Ayuda desplazada
KEY_SHOME	Inicio cambiado
KEY_SIC	Entrada cambiada
KEY_SLEFT	Flecha hacia la izquierda desplazada
KEY_SMESSAGE	Mensaje cambiado
KEY_SMOVE	Movimiento desplazado
KEY_SNEXT	Siguiente desplazado
KEY_SOPTIONS	Opciones cambiadas
KEY_SPREVIOUS	Anterior cambiado
KEY_SPRINT	Imprimir desplazado
KEY_SREDO	Rehacer cambiado
KEY_SREPLACE	Reemplazo cambiado
KEY_SRIGHT	Flecha hacia la derecha cambiada
KEY_SRSUME	Resumen cambiado
KEY_SSAVE	Guardar desplazado
KEY_SSUSPEND	Suspender cambiado
KEY_SUNDO	Deshacer desplazado
KEY_SUSPEND	Suspender
KEY_UNDO	Deshacer
KEY_MOUSE	Evento del ratón ha ocurrido
KEY_RESIZE	Evento de cambio de tamaño del terminal
KEY_MAX	Valor máximo de clave

On VT100s and their software emulations, such as X terminal emulators, there are normally at least four function keys (**KEY_F1**, **KEY_F2**, **KEY_F3**, **KEY_F4**) available, and the arrow keys mapped to **KEY_UP**, **KEY_DOWN**, **KEY_LEFT** and **KEY_RIGHT** in the obvious way. If your machine has a PC keyboard, it is safe to expect arrow keys and twelve function keys (older PC keyboards may have only ten function keys); also, the following keypad mappings are standard:

Tecla	**Constante**

KEY_IC	Insert
KEY_DC	Delete
KEY_HOME	Home
KEY_END	End
KEY_PPAGE	Page Up
KEY_NPAGE	Page Down

La siguiente tabla lista los caracteres desde el conjunto alterno de caracteres. Estos son heredados desde el terminal VT100, y generalmente estará disponible en emuladores de software tales como terminales X. Cuando no hay gráficos disponibles, *curses* cae en una aproximación cruda de ASCII imprimibles.

Nota

Estos están disponibles solamente después de que `initscr()` ha sido llamada.

Código ACS	**Significado**
curses.ACS_BBSS	nombre alternativo para la esquina superior derecha
curses.ACS_BLOCK	bloque cuadrado sólido
curses.ACS_BOARD	tablero de cuadrados
curses.ACS_BSBS	nombre alternativo para línea horizontal
curses.ACS_BSSB	nombre alternativo para esquina superior izquierda
curses.ACS_BSSS	nombre alternativo para el soporte superior
curses.ACS_BTEE	soporte inferior
curses.ACS_BULLET	bala
curses.ACS_CKBOARD	tablero inspector (punteado)
curses.ACS_DARROW	flecha apuntando hacia abajo
curses.ACS_DEGREE	símbolo de grado
curses.ACS_DIAMOND	diamante
curses.ACS_GEQUAL	mayor que o igual a
curses.ACS_HLINE	línea horizontal
curses.ACS_LANTERN	símbolo de la linterna
curses.ACS_LARROW	flecha izquierda
curses.ACS_LEQUAL	menor que o igual a
curses.ACS_LLCORNER	esquina inferior izquierda
curses.ACS_LRCORNER	esquina inferior derecha
curses.ACS_LTEE	soporte izquierdo
curses.ACS_NEQUAL	signo no igual

curses.ACS_PI	letra pi
curses.ACS_PLMINUS	signo más o menos
curses.ACS_PLUS	gran signo más
curses.ACS_RARROW	flecha hacia la derecha
curses.ACS_RTEE	soporte derecho
curses.ACS_S1	escanear línea 1
curses.ACS_S3	escanear línea 3
curses.ACS_S7	escanear línea 7
curses.ACS_S9	escanear línea 9
curses.ACS_SBBS	nombre alterno para la esquina inferior derecha
curses.ACS_SBSB	nombre alterno para la línea vertical
curses.ACS_SBSS	nombre alterno para el soporte derecho
curses.ACS_SSBB	nombre alterno para la esquina inferior izquierda
curses.ACS_SSBS	nombre alterno para el soporte inferior
curses.ACS_SSSB	nombre alterno para el soporte izquierdo
curses.ACS_SSSS	nombre alterno para *crossover* o un gran más
curses.ACS_STERLING	libra esterlina
curses.ACS_TTEE	soporte superior
curses.ACS_UARROW	flecha hacia arriba
curses.ACS_ULCORNER	esquina superior izquierda
curses.ACS_URCORNER	esquina superior derecha
curses.ACS_VLINE	línea vertical

The following table lists mouse button constants used by `getmouse()`:

Mouse button constant	**Significado**
curses.BUTTONn_PRESSED	Mouse button *n* pressed
curses.BUTTONn_RELEASED	Mouse button *n* released
curses.BUTTONn_CLICKED	Mouse button *n* clicked
curses.BUTTONn_DOUBLE_CLICKED	Mouse button *n* double clicked
curses.BUTTONn_TRIPLE_CLICKED	Mouse button *n* triple clicked
curses.BUTTON_SHIFT	Shift was down during button state change
curses.BUTTON_CTRL	Control was down during button state change
curses.BUTTON_ALT	Control was down during button state change

Distinto en la versión 3.10: Las constantes `BUTTON5_*` ahora están expuestas si son proporcionadas por la biblioteca curses subyacente.

La siguiente tabla lista los colores pre-definidos:

Color	Constante
Negro	curses.COLOR_BLACK
Azul	curses.COLOR_BLUE
Cian (azul verdoso claro)	curses.COLOR_CYAN
Verde	curses.COLOR_GREEN
Magenta (rojo violáceo)	curses.COLOR_MAGENTA
Rojo	curses.COLOR_RED
Blanco	curses.COLOR_WHITE
Amarillo	curses.COLOR_YELLOW

`curses.textpad` — Text input widget for curses programs

The `curses.textpad` module provides a `Textbox` class that handles elementary text editing in a curses window, supporting a set of keybindings resembling those of Emacs (thus, also of Netscape Navigator, BBedit 6.x, FrameMaker, and many other programs). The module also provides a rectangle-drawing function useful for framing text boxes or for other purposes.

The module `curses.textpad` defines the following function:

curses.textpad.rectangle(*win*, *uly*, *ulx*, *lry*, *lrx*)

> Dibuja un rectángulo. El primer argumento debe ser un objeto de ventana, los argumentos restantes son coordenadas relativas para esa ventana. El segundo y tercer argumento son las coordenadas "x" y "y" de la esquina superior de la mano izquierda del rectángulo a ser dibujado, el cuarto y quinto argumento son las coordenadas "x" y "y" de la esquina inferior de la mano derecha. El rectángulo será dibujado usando formularios de caracteres VT100/IBM PC en terminales que hace esto posible (incluyendo *xterm* y mas otro emulador de terminal de software). Por otra parte será dibujado con guiones, barras verticales, y signos de más de ASCII.

Objeto de caja de texto

Tú puedes instanciar un objeto `Textbox` como sigue:

class curses.textpad.Textbox(*win*)

Retorna un objeto *widget* de caja de texto. El argumento *win* debería ser un objeto *curses* window en el cual la caja de texto es para ser contenido. El cursor de la caja de texto está inicialmente localizado en la esquina superior de la mano izquierda de la ventana que lo contiene, con coordenadas `(0, 0)`. La bandera instanciada `stripspaces` está inicialmente encendida.

Objeto `Textbox` tiene los siguientes métodos:

edit([*validator*])

Este es el punto de entrada que normalmente usarás. Esto acepta la edición de las pulsaciones de tecla hasta que uno de las pulsaciones de finalización es introducida. Si *validator* es suministrado, esto debe ser una función. Será llamado para cada pulsación de tecla introducida con la pulsación de tecla como un parámetro; el comando ejecutado está hecho en el resultado. Este método retorna el contenido de la ventana como una cadena de caracteres; si se incluyen espacios en blanco en la ventana se ve afectado por el atributo `stripspaces`.

do_command(*ch*)

Procesa un simple comando al pulsar una tecla. Aquí está las funciones de tecla especiales admitidas:

Pulsación de tecla	**Acción**
`Control-A`	Ve al borde izquierdo de la ventana.
`Control-B`	Cursor hacia la izquierda, pasando a la línea anterior si corresponde.
`Control-D`	Borra el carácter bajo el cursor.
`Control-E`	Ve al borde derecho (quita los espacios) o al final de línea (eliminar los espacios).

`Control-F`	Cursor hacia la derecha, pasando a la línea siguiente cuando corresponda.
`Control-G`	Terminar, retornando el contenido de la ventana.
`Control-H`	Eliminar carácter al revés.
`Control-J`	Terminar si la ventana es de 1 línea, en caso contrario, inserte una nueva línea.
`Control-K`	Si la línea es blanca bórrela, en caso contrario, limpie hasta el final de línea.
`Control-L`	Refrescar la pantalla.
`Control-N`	Cursor hacia abajo; mueve hacia abajo una línea.
`Control-O`	Inserte una línea en blanco en la posición del cursor.
`Control-P`	Cursor hacia arriba; mueve hacia arriba una línea.

Las operaciones de movimiento no hacen nada si el cursor está en un borde donde el movimiento no es posible. Los siguientes sinónimos están apoyados siempre que sea posible:

Constante	**Pulsación de tecla**
`KEY_LEFT`	`Control-B`
`KEY_RIGHT`	`Control-F`
`KEY_UP`	`Control-P`
`KEY_DOWN`	`Control-N`
`KEY_BACKSPACE`	`Control-h`

Todas las otras pulsaciones de teclas están tratadas como un comando para insertar el carácter dado y muévase a la derecha (con ajuste en la línea).

gather()

Retorne el contenido de la ventana como una cadena de texto; si se incluyen espacios en blanco en la ventana se ve afectado por el miembro `stripspaces`.

stripspaces

Este atributo es una bandera que controla la interpretación de los espacios en blanco en la ventana. Cuando está encendido, los espacios en blanco al final en cada línea son ignorados; cualquier movimiento del cursor que lo coloque en un espacio en blanco final va hasta el final de esa línea, y los espacios en blanco finales son despejados cuando se recopila el contenido de la ventana.

`curses.ascii` — Utilities for ASCII characters

Source code: Lib/curses/ascii.py [https://github.com/python/cpython/tree/3.14/Lib/curses/ascii.py]

The `curses.ascii` module supplies name constants for ASCII characters and functions to test membership in various ASCII character classes. The constants supplied are names for control characters as follows:

Nombre	Significado
curses.ascii.NUL	
curses.ascii.SOH	Inicio del encabezado, interrupción de la consola
curses.ascii.STX	Inicio del texto
curses.ascii.ETX	Final del texto
curses.ascii.EOT	Fin de la transmisión
curses.ascii.ENQ	Consulta, va con el control de flujo `ACK`
curses.ascii.ACK	Reconocimiento
curses.ascii.BEL	Campana
curses.ascii.BS	Retroceso
curses.ascii.TAB	Tabulación
curses.ascii.HT	Alias para `TAB`: «Tabulación horizontal»
curses.ascii.LF	Línea de alimentación
curses.ascii.NL	Alias para `LF`: «Nueva línea»
curses.ascii.VT	Tabulación vertical
curses.ascii.FF	Alimentación de formulario
curses.ascii.CR	Retorno de carro (*Carriage return* en inglés)
curses.ascii.SO	*Shift-out*, comenzar un conjunto de caracteres alternativo
curses.ascii.SI	*Shift-in*, reanudar el conjunto de caracteres predeterminado
curses.ascii.DLE	Escape de enlace de datos
curses.ascii.DC1	XON, para control de flujo
curses.ascii.DC2	Control de dispositivo 2, control de flujo en modo bloque
curses.ascii.DC3	XOFF, para control de flujo

curses.ascii.DC4	Control de dispositivo 4
curses.ascii.NAK	Reconocimiento negativo
curses.ascii.SYN	Inactivo sincrónico
curses.ascii.ETB	Bloque de transmisión final
curses.ascii.CAN	Cancelar
curses.ascii.EM	Fin del medio
curses.ascii.SUB	Sustituir
curses.ascii.ESC	Escapar
curses.ascii.FS	Separador de archivos
curses.ascii.GS	Separador de grupos
curses.ascii.RS	Separador de registros, finalizador en modo bloque
curses.ascii.US	Separador de unidades
curses.ascii.SP	Espacio
curses.ascii.DEL	Eliminar

Tenga en cuenta que muchos de estos tienen poca importancia práctica en el uso moderno. Los mnemónicos se derivan de las convenciones de la teleimpresora que son anteriores a las computadoras digitales.

El módulo proporciona las siguientes funciones, siguiendo el patrón de las de la biblioteca C estándar:

curses.ascii.isalnum(*c*)

> Comprueba un carácter alfanumérico ASCII; esto es equivalente a `isalpha(c) or isdigit(c)`.

curses.ascii.isalpha(*c*)

> Comprueba si hay un carácter alfabético ASCII; es equivalente a `isupper(c) or islower(c)`.

curses.ascii.isascii(*c*)

> Comprueba un valor de carácter que se ajuste al conjunto ASCII de 7 bits.

curses.ascii.isblank(*c*)

> Comprueba si hay un carácter de espacio en blanco ASCII; espacio o tabulación horizontal.

curses.ascii.iscntrl(*c*)

Comprueba un carácter de control ASCII (en el rango de 0x00 a 0x1f o 0x7f).

curses.ascii.isdigit(*c*)

Comprueba si hay un dígito decimal ASCII, desde `'0'` hasta `'9'`. Esto es equivalente a `c in string.digits`.

curses.ascii.isgraph(*c*)

Comprueba en ASCII cualquier carácter imprimible excepto el espacio.

curses.ascii.islower(*c*)

Comprueba un carácter ASCII en minúscula.

curses.ascii.isprint(*c*)

Comprueba cualquier carácter imprimible ASCII, incluido el espacio.

curses.ascii.ispunct(*c*)

Comprueba si hay algún carácter ASCII imprimible que no sea un espacio o un carácter alfanumérico.

curses.ascii.isspace(*c*)

Comprueba los caracteres de espacio en blanco ASCII; espacio, línea de alimentación, retorno de carro, formulario de alimentación, tabulación horizontal, tabulación vertical.

curses.ascii.isupper(*c*)

Comprueba una letra mayúscula ASCII.

curses.ascii.isxdigit(*c*)

Comprueba si hay un dígito hexadecimal ASCII. Esto es equivalente a `c in string.hexdigits`.

curses.ascii.isctrl(*c*)

> Comprueba un carácter de control ASCII (valores ordinales de 0 a 31)

curses.ascii.ismeta(*c*)

> Comprueba si hay un carácter no ASCII (valores ordinales 0x80 y superiores).

Estas funciones aceptan enteros o cadenas de un solo carácter; cuando el argumento es una cadena de caracteres, primero se convierte utilizando la función *built-in* `ord()`.

Tenga en cuenta que todas estas funciones verifican los valores de bits ordinales derivados del carácter de la cadena que ingresa; en realidad, no saben nada sobre la codificación de caracteres de la máquina host.

Las siguientes dos funciones toman una cadena de un solo carácter o un valor de byte entero; devuelven un valor del mismo tipo.

curses.ascii.ascii(*c*)

> Retorna el valor ASCII correspondiente a los 7 bits bajos de *c*.

curses.ascii.ctrl(*c*)

> Retorna el carácter de control correspondiente al carácter dado (el valor del bit del carácter es bit a bit (*bitwise-anded*) con 0x1f).

curses.ascii.alt(*c*)

> Retorna el carácter de 8 bits correspondiente al carácter ASCII dado (el valor del bit de carácter se escribe bit a bit (*bitwise-ored*) con 0x80).

La siguiente función toma una cadena de un solo carácter o un valor entero; devuelve una cadena.

curses.ascii.unctrl(*c*)

Retorna una representación de cadena del carácter ASCII *c*. Si *c* es imprimible, esta cadena es el propio carácter. Si el carácter es un carácter de control (0x00–0x1f) la cadena consta de un signo de intercalación (`'^'`) seguido de la letra mayúscula correspondiente. Si el carácter es una eliminación ASCII (0x7f), la cadena es `'^?'`. Si el carácter tiene su meta bit establecido (0x80), el meta bit se elimina, se aplican las reglas anteriores y se antepone `'!'` al resultado.

curses.ascii.controlnames

Una matriz de cadena de caracteres de 33 elementos que contiene los mnemónicos ASCII para los treinta y dos caracteres de control ASCII desde 0 (NUL) a 0x1f (US), en orden, más el mnemónico `SP` para el carácter de espacio.

`curses.panel` — A panel stack extension for curses

Los paneles son ventanas con la característica de profundidad añadida, por lo que se pueden apilar una encima de la otra, y solo se mostrarán las partes visibles de cada ventana. Los paneles se pueden agregar, mover hacia arriba o hacia abajo en la pila y eliminarse.

Funciones

The module `curses.panel` defines the following functions:

curses.panel.bottom_panel()

Retorna el panel inferior en la pila del panel.

curses.panel.new_panel(*win*)

Retorna un objeto de panel, asociándolo con la ventana dada *win*. Tenga en cuenta que debe mantener explícitamente el objeto de panel retornado al que se hace referencia. Si no lo hace, el objeto de panel se recoge como basura y elimina de la pila del panel.

curses.panel.top_panel()

Retorna el panel superior de la pila de paneles.

curses.panel.update_panels()

Actualiza la pantalla virtual después de los cambios en la pila del panel. Esto no llama a `curses.doupdate()`, por lo que tendrás que hacerlo tú mismo.

Objetos de panel

Los objetos panel, retornados por `new_panel()` arriba, son ventanas con un orden de apilamiento. Siempre hay una ventana asociada a un panel que determina el contenido, mientras que los métodos del panel son responsables de la profundidad de la ventana en la pila del panel.

Los objetos de panel tienen los siguientes métodos:

Panel.above()

> Retorna el panel situado encima del panel actual.

Panel.below()

> Retorna el panel debajo del panel actual.

Panel.bottom()

> Empuja el panel hasta la parte inferior de la pila.

Panel.hidden()

> Retorna `True` si el panel está oculto (no visible), `False` en caso contrario.

Panel.hide()

> Ocultar el panel. Esto no elimina el objeto, solo hace que la ventana en la pantalla sea invisible.

Panel.move(*y*, *x*)

> Mueve el panel a las coordenadas de pantalla `(y, x)`.

Panel.replace(*win*)

> Cambia la ventana asociada con el panel a la ventana *win*.

Panel.set_userptr(*obj*)

> Establece el puntero de usuario del panel en *obj*. Esto se usa

para asociar un dato arbitrario con el panel y puede ser cualquier objeto de Python.

Panel.show()

Muestra el panel (que podría haber estado oculto).

Panel.top()

Empuja el panel hacia la parte superior de la pila.

Panel.userptr()

Retorna el puntero del usuario para el panel. Puede ser cualquier objeto de Python.

Panel.window()

Retorna el objeto de ventana asociado con el panel.

`cmd` — Support for line-oriented command interpreters

Código fuente: Lib/cmd.py [https://github.com/python/cpython/tree/3.14/Lib/cmd.py]

La clase `Cmd` proporciona un *framework* simple para escribir intérpretes de comandos orientados a línea. A menudo son útiles para agentes de prueba, herramientas administrativas y prototipos que luego se incluirán en una interfaz más sofisticada.

class cmd.Cmd(*completekey='tab'*, *stdin=None*, *stdout=None*)

Una instancia `Cmd` o subclase de la instancia es un *framework* intérprete orientado a líneas. No hay una buena razón para crear instancias `Cmd`; mas bien, es útil como una super clase de una clase intérprete que usted define con el fin de heredar métodos `Cmd` y encapsular métodos de acción.

El argumento opcional *completekey* es el nombre `readline` de una llave de finalización; por defecto es `Tab`. Si *completekey* no es `None` y `readline` está disponible, el comando de finalización es hecho automáticamente.

The default, `'tab'`, is treated specially, so that it refers to the `Tab` key on every `readline.backend`. Specifically, if `readline.backend` is `editline`, `Cmd` will use `'^I'` instead of `'tab'`. Note that other values are not treated this way, and might only work with a specific backend.

Los argumentos opcionales *stdin* y *stdout* especifican la entrada y salida de los objetos archivo que la instancia Cmd o la instancia subclase usará para la entrada y salida. Si no está especificado, se predeterminarán a `sys.stdin` y `sys.stdout`.

Si desea un *stdin* dado a ser usado, asegúrese de establecer las instancias atributo `use_rawinput` a `False`, de lo contrario *stdin* será ignorado.

Distinto en la versión 3.13: `completekey='tab'` is replaced by `'^I'` for `editline`.

Objetos Cmd

Una instancia `Cmd` tiene los siguientes métodos:

Cmd.cmdloop(*intro=None*)

Emita repetidamente una solicitud, acepte la entrada, analice un prefijo inicial de la entrada recibida y envíe a los métodos de acción, pasándoles el resto de la línea como argumento.

El argumento opcional es un *banner* o cadena de caracteres introductoria que se tramitará antes del primer mensaje (esto anula el atributo de clase `intro`).

Si el módulo `readline` es cargado, la entrada heredará automáticamente **bash**-como edición historia-lista (e.g. `Control-P` retorna al último comando, `Control-N` adelanta al siguiente, `Control-F` mueve el cursor a la derecha no destructivamente, `Control-B` mueve el cursor a la izquierda no destructivamente, etc.).

Un fin-de-archivo en la entrada es retornado como cadena de caracteres `’EOF’`.

An interpreter instance will recognize a command name `foo` if and only if it has a method `do_foo()`. As a special case, a line beginning with the character `'?'` is dispatched to the method `do_help()`. As another special case, a line beginning with the character `'!'` is dispatched to the method `do_shell()` (if such a method is defined).

This method will return when the `postcmd()` method returns a true value. The *stop* argument to `postcmd()` is the return value from the command’s corresponding `do_*()`

method.

If completion is enabled, completing commands will be done automatically, and completing of commands args is done by calling **`complete_foo()`** with arguments *text*, *line*, *begidx*, and *endidx*. *text* is the string prefix we are attempting to match: all returned matches must begin with it. *line* is the current input line with leading whitespace removed, *begidx* and *endidx* are the beginning and ending indexes of the prefix text, which could be used to provide different completion depending upon which position the argument is in.

Cmd.do_help(*arg*)

All subclasses of **`Cmd`** inherit a predefined **`do_help()`**. This method, called with an argument `'bar'`, invokes the corresponding method **`help_bar()`**, and if that is not present, prints the docstring of **`do_bar()`**, if available. With no argument, **`do_help()`** lists all available help topics (that is, all commands with corresponding **`help_*()`** methods or commands that have docstrings), and also lists any undocumented commands.

Cmd.onecmd(*str*)

Interpret the argument as though it had been typed in response to the prompt. This may be overridden, but should not normally need to be; see the **`precmd()`** and **`postcmd()`** methods for useful execution hooks. The return value is a flag indicating whether interpretation of commands by the interpreter should stop. If there is a **`do_*()`** method for the command *str*, the return value of that method is returned, otherwise the return value from the **`default()`** method is returned.

Cmd.emptyline()

Método llamado cuando se ingresa una línea vacía en respuesta a la solicitud. Si este método no se anula, repite el último comando no vacío ingresado.

Cmd.default(*line*)

Método llamado en una línea de entrada cuando no se reconoce el prefijo del comando. Si este método no se anula, imprime un mensaje de error y retorna.

Cmd.completedefault(*text*, *line*, *begidx*, *endidx*)

Method called to complete an input line when no command-specific `complete_*()` method is available. By default, it returns an empty list.

Cmd.columnize(*list*, *displaywidth=80*)

Método llamado para mostrar una lista de cadenas de caracteres como un set compacto de columnas. Cada columna es tan amplia como sea necesaria. Las columnas se separan por dos espacios para facilidad de lectura.

Cmd.precmd(*line*)

Método de enlace ejecutado justo antes de que se interprete la línea de comando *line*, pero después de que se genere y emita la solicitud de entrada. Este método es un trozo en `Cmd`; existe para ser anulado por subclases. El valor de retorno es utilizado como el comando que se ejecutará mediante el método `onecmd()`; la implementación `precmd()` puede reescribir el comando o simplemente retornar *line* sin cambios.

Cmd.postcmd(*stop*, *line*)

Método de enlace ejecutado justo después de que finaliza un despacho de comando. Este método es un trozo en `Cmd`; existe para ser anulado por subclases. *line* es la línea de comando que se ejecutó, y *stop* es una bandera que indica si la ejecución finalizará después de la llamada a `postcmd()`; este será el valor de retorno del método `onecmd()`. El valor de retorno de este método será usado como el nuevo valor para la bandera interna que corresponde a *stop*; retornando falso hará que la interpretación continúe.

Cmd.preloop()

Método de enlace ejecutado una vez cuando **`cmdloop()`** es llamado. Este método es un trozo en **`Cmd`**; existe para ser anulado por subclases.

Cmd.postloop()

Método de enlace ejecutado una vez cuando **`cmdloop()`** está a punto de retornar. Este método es un trozo en **`Cmd`**; existe para ser anulado por subclases.

Instancias de subclases **`Cmd`** tienen algunas variables de instancia públicas:

Cmd.prompt

El aviso emitido para solicitar entradas.

Cmd.identchars

La cadena de caracteres aceptada para el prefijo del comando.

Cmd.lastcmd

El último prefijo de comando no vacío visto.

Cmd.cmdqueue

Una lista de líneas de entrada puestas en cola. La lista *cmdqueue* es verificada en **`cmdloop()`** cuando una nueva entrada es necesitada; Si es no vacía, sus elementos serán procesados en orden, como si se ingresara en la solicitud.

Cmd.intro

Una cadena para emitir como introducción o *banner*. Puede ser anulado dando un argumento al método **`cmdloop()`**.

Cmd.doc_header

El encabezado a tramitar si la salida de ayuda tiene una sección para comandos documentados.

Cmd.misc_header

The header to issue if the help output has a section for miscellaneous help topics (that is, there are `help_*()` methods without corresponding `do_*()` methods).

Cmd.undoc_header

The header to issue if the help output has a section for undocumented commands (that is, there are `do_*()` methods without corresponding `help_*()` methods).

Cmd.ruler

El carácter utilizado para dibujar líneas separadoras debajo de los encabezados mensajes-ayuda. Si está vacío, no se dibuja una línea regla. El valor predeterminado es `'='`.

Cmd.use_rawinput

A flag, defaulting to true. If true, `cmdloop()` uses `input()` to display a prompt and read the next command; if false, `sys.stdout.write()` and `sys.stdin.readline()` are used. (This means that by importing `readline`, on systems that support it, the interpreter will automatically support **Emacs**-like line editing and command-history keystrokes.)

Ejemplo Cmd

The `cmd` module is mainly useful for building custom shells that let a user work with a program interactively.

Esta sección presenta un ejemplo simple de cómo construir un *shell* alrededor de algunos de los comandos en el módulo `turtle`.

Basic turtle commands such as `forward()` are added to a `Cmd` subclass with method named `do_forward()`. The argument is converted to a number and dispatched to the turtle module. The docstring is used in the help utility provided by the shell.

The example also includes a basic record and playback facility implemented with the `precmd()` method which is responsible for converting the input to lowercase and writing the commands to a file. The `do_playback()` method reads the file and adds the

recorded commands to the **`cmdqueue`** for immediate playback:

```
import cmd, sys
from turtle import *

class TurtleShell(cmd.Cmd):
    intro = 'Welcome to the turtle shell.   Type help or
    prompt = '(turtle) '
    file = None

    # ----- basic turtle commands -----
    def do_forward(self, arg):
        'Move the turtle forward by the specified distar
        forward(*parse(arg))
    def do_right(self, arg):
        'Turn turtle right by given number of degrees:
        right(*parse(arg))
    def do_left(self, arg):
        'Turn turtle left by given number of degrees:  I
        left(*parse(arg))
    def do_goto(self, arg):
        'Move turtle to an absolute position with changi
        goto(*parse(arg))
    def do_home(self, arg):
        'Return turtle to the home position:  HOME'
        home()
    def do_circle(self, arg):
        'Draw circle with given radius an options extent
        circle(*parse(arg))
    def do_position(self, arg):
        'Print the current turtle position:  POSITION'
        print('Current position is %d %d\n' % position()
    def do_heading(self, arg):
        'Print the current turtle heading in degrees:  H
        print('Current heading is %d\n' % (heading(),))
    def do_color(self, arg):
        'Set the color:  COLOR BLUE'
        color(arg.lower())
    def do_undo(self, arg):
        'Undo (repeatedly) the last turtle action(s):  U
    def do_reset(self, arg):
        'Clear the screen and return turtle to center:
        reset()
    def do_bye(self, arg):
        'Stop recording, close the turtle window, and ex
        print('Thank you for using Turtle')
        self.close()
        bye()
        return True

    # ----- record and playback -----
    def do_record(self, arg):
        'Save future commands to filename:  RECORD rose.
        self.file = open(arg, 'w')
    def do_playback(self, arg):
        'Playback commands from a file:  PLAYBACK rose.c
        self.close()
        with open(arg) as f:
            self.cmdqueue.extend(f.read().splitlines())
    def precmd(self, line):
        line = line.lower()
        if self.file and 'playback' not in line:
            print(line, file=self.file)
        return line
    def close(self):
        if self.file:
            self.file.close()
            self.file = None

def parse(arg):
    'Convert a series of zero or more numbers to an argu
    return tuple(map(int, arg.split()))

if __name__ == '__main__':
    TurtleShell().cmdloop()
```

Aquí hay una sesión de muestra con la *turle shell* mostrando las funciones de ayuda, de ayuda, usando líneas en blanco para repetir comandos, un registro simple y facilidad de reproducción:

```
Welcome to the turtle shell.   Type help or ? to list co

(turtle) ?

Documented commands (type help <topic>):
========================================
bye     color    goto     home  playback  record  right
circle  forward  heading  left  position  reset   undo

(turtle) help forward
Move the turtle forward by the specified distance:  FORW
(turtle) record spiral.cmd
(turtle) position
Current position is 0 0

(turtle) heading
Current heading is 0

(turtle) reset
(turtle) circle 20
(turtle) right 30
(turtle) circle 40
(turtle) right 30
(turtle) circle 60
(turtle) right 30
(turtle) circle 80
(turtle) right 30
(turtle) circle 100
(turtle) right 30
(turtle) circle 120
(turtle) right 30
(turtle) circle 120
(turtle) heading
Current heading is 180

(turtle) forward 100
(turtle)
```

```
(turtle) right 90
(turtle) forward 100
(turtle)
(turtle) right 90
(turtle) forward 400
(turtle) right 90
(turtle) forward 500
(turtle) right 90
(turtle) forward 400
(turtle) right 90
(turtle) forward 300
(turtle) playback spiral.cmd
Current position is 0 0

Current heading is 0

Current heading is 180

(turtle) bye
Thank you for using Turtle
```

Ejecución concurrente

Los módulos descritos en este capítulo proveen soporte para la ejecución concurrente de código. La elección de qué herramienta utilizar depende de la tarea a ejecutar (vinculada a CPU o vinculada a E/S) y del estilo preferido de desarrollo (multi-tarea cooperativa o multi-tarea apropiativa). A continuación se muestra un resumen:

- `threading` — Thread-based parallelism
    - Introduction
    - GIL and performance considerations
    - Reference
        - Thread-local data
        - Thread objects
        - Lock objects
        - RLock objects
        - Condition objects
        - Semaphore objects
        - `Semaphore` example
        - Event objects
        - Timer objects
        - Barrier objects
    - Uso de *locks*, condiciones y semáforos en la declaración `with`
- `multiprocessing` — Process-based parallelism
    - Introducción
        - La clase `Process`
        - Contextos y métodos de inicio
        - Intercambiando objetos entre procesos
        - Sincronización entre procesos
        - Compartiendo estado entre procesos

    - Usando una piscina de trabajadores (*pool of workers*)
- Referencia
    - Global start method
    - **`Process`** y excepciones
    - Tuberías (*Pipes*) y Colas (*Queues*)
    - Miscelánea
    - Objetos de conexión *Connection Objects*
    - Primitivas de sincronización (*Synchronization primitives*)
    - Objetos compartidos **`ctypes`**
        - The **`multiprocessing.sharedctypes`** module
    - Administradores (*Managers*)
        - Administradores customizables (*Customized managers*)
        - Utilizando un administrador remoto
    - Objetos Proxy (*Proxy Objects*)
        - Limpieza (*Cleanup*)
    - Piscinas de procesos (*Process Pools*)
    - Oyentes y clientes (*Listeners and Clients*)
        - Formatos de dirección (*Address formats*)
    - Llaves de autentificación
    - *Logging*
    - The **`multiprocessing.dummy`** module
- Pautas de programación
    - Todos los métodos de inicio
    - Los métodos de inicio *spawn* y *forkserver*
- Ejemplos

- **`multiprocessing.shared_memory`** — Shared memory for direct access across processes
- El paquete **`concurrent`**
- **`concurrent.futures`** — Launching parallel tasks
  - ○ Objetos ejecutores
  - ○ ThreadPoolExecutor
    - ■ Ejemplo de ThreadPoolExecutor
  - ○ InterpreterPoolExecutor
  - ○ ProcessPoolExecutor
    - ■ Ejemplo de *ProcessPoolExecutor*
  - ○ Objetos futuro
  - ○ Funciones del módulo
  - ○ Clases de Excepciones
- **`concurrent.interpreters`** — Multiple interpreters in the same process
  - ○ Key details
  - ○ Introduction
    - ■ Multiple Interpreters and Isolation
    - ■ Running in an Interpreter
    - ■ Concurrency and Parallelism
    - ■ Communication Between Interpreters
    - ■ «Sharing» Objects
  - ○ Reference
    - ■ Interpreter objects
    - ■ Exceptions
    - ■ Communicating Between Interpreters
  - ○ Basic usage
- **`subprocess`** — Subprocess management
  - ○ Using the **`subprocess`** Module

        - Argumentos frecuentemente empleados
        - El constructor de Popen
        - Excepciones
    - Consideraciones sobre seguridad
    - Objetos Popen
    - Elementos auxiliares de Popen en Windows
        - Constantes de Windows
    - Antigua API de alto nivel
    - Replacing Older Functions with the `subprocess` Module
        - Cómo reemplazar la sustitución de órdenes de **/bin/sh**
        - Cómo remplazar los flujos de la shell
        - Cómo reemplazar `os.system()`
        - Cómo reemplazar la familia `os.spawn`
        - Replacing `os.popen()`
    - Funciones de llamada a la shell de retrocompatibilidad
    - Notas
        - Timeout Behavior
        - Cómo convertir una secuencia de argumentos a una cadena en Windows
        - Disable use of `posix_spawn()`
- `sched` — Event scheduler
    - Objetos de `Scheduler`
- `queue` — A synchronized queue class
    - Objetos de la cola
        - Waiting for task completion
        - Terminating queues
    - Objetos de cola simple

- **`contextvars`** — Context Variables
    - variables de contexto
    - Gestión de contexto manual
    - Soporte asyncio

He aquí módulos de apoyo para algunos de los servicios mencionados:

- **`_thread`** — Low-level threading API

`threading` — Thread-based parallelism

Código fuente: Lib/threading.py [https://github.com/python/cpython/tree/3.14/Lib/threading.py]

Este módulo construye interfaces de hilado de alto nivel sobre el módulo de más bajo nivel **`_thread`**.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

Introduction

The **`threading`** module provides a way to run multiple threads [https://en.wikipedia.org/wiki/Thread_(computing)] (smaller units of a process) concurrently within a single process. It allows for the creation and management of threads, making it possible to execute tasks in parallel, sharing memory space. Threads are particularly useful when tasks are I/O bound, such as file operations or making network requests, where much of the time is spent waiting for external resources.

A typical use case for **`threading`** includes managing a pool of worker threads that can process multiple tasks concurrently. Here's a basic example of creating and starting threads using **`Thread`**:

```
import threading
import time

def crawl(link, delay=3):
    print(f"crawl started for {link}")
    time.sleep(delay)  # Blocking I/O (simulating a netw
    print(f"crawl ended for {link}")

links = [
    "https://python.org",
    "https://docs.python.org",
    "https://peps.python.org",
]

# Start threads for each link
threads = []
for link in links:
    # Using `args` to pass positional arguments and `kwa
    t = threading.Thread(target=crawl, args=(link,), kwa
    threads.append(t)

# Start each thread
for t in threads:
    t.start()

# Wait for all threads to finish
for t in threads:
    t.join()
```

Distinto en la versión 3.7: Este módulo solía ser opcional, ahora está siempre disponible.

Ver también

`concurrent.futures.ThreadPoolExecutor` ofrece una interfaz a mas alto nivel para enviar tareas a un hilo en segundo plano sin bloquear la ejecución del hilo de llamada, pero manteniendo la capacidad de recuperar sus resultados cuando sea necesario.

`queue` proporciona una interfaz segura a nivel de hilos intercambiar datos entre hilos en ejecución.

`asyncio` ofrece un enfoque alternativo para lograr la concurrencia a nivel de tarea sin requerir el uso de múltiples subprocesos del sistema operativo.

Nota

En la serie Python 2.x, este módulo contenía nombres `camelCase` para algunos métodos y funciones. Estos están obsoletos a partir de Python 3.10, pero aún son compatibles por compatibilidad con Python 2.5 y versiones anteriores.

En CPython, debido al Candado de intérprete global, solo un hilo puede ejecutar código Python a la vez (aunque ciertas bibliotecas orientadas al rendimiento pueden superar esta limitación). Si desea que su aplicación haga un mejor uso de los recursos computacionales de las máquinas multinúcleo, se recomienda utilizar **`multiprocessing`** o **`concurrent.futures.ProcessPoolExecutor`**. Sin embargo, el subproceso sigue siendo un modelo apropiado si desea ejecutar varias tareas vinculadas a E/S simultáneamente.

GIL and performance considerations

Unlike the **`multiprocessing`** module, which uses separate processes to bypass the global interpreter lock (GIL), the threading module operates within a single process, meaning that all threads share the same memory space. However, the GIL limits the performance gains of threading when it comes to CPU-bound tasks, as only one thread can execute Python bytecode at a time. Despite this, threads remain a useful tool for achieving concurrency in many scenarios.

As of Python 3.13, free-threaded builds can disable the GIL, enabling true parallel execution of threads, but this feature is not available by default (see **PEP 703** [https://peps.python.org/pep-0703/]).

Reference

Este módulo define las siguientes funciones:

threading.active_count()

Retorna el número de objetos **`Thread`** actualmente con vida. La cuenta retornada es igual al largo de la lista retornada por **`enumerate()`**.

La función `activeCount` es un alias obsoleto para esta función.

threading.current_thread()

Return the current **`Thread`** object, corresponding to the caller's thread of control. If the caller's thread of control was not created through the **`threading`** module, a dummy thread object with limited functionality is returned.

La función `currentThread` es un alias obsoleto para esta función.

threading.excepthook(*args*, /)

Gestiona una excepción lanzada por **`Thread.run()`**.

El argumento *args* posee los siguientes atributos:

- *exc_type*: Tipo de excepción.
- *exc_value*: Valor de la excepción, puede ser `None`.
- *exc_traceback*: Rastreo de la excepción, puede ser `None`.
- *thread*: El hilo que ha lanzado la excepción, puede ser `None`.

Si *exc_type* es **`SystemExit`**, la excepción es silenciosamente ignorada. De otro modo, la excepción se imprime en **`sys.stderr`**.

Si esta función lanza una excepción, se llama a **`sys.excepthook()`** para manejarla.

`threading.excepthook()` se puede sobrescribir para controlar cómo se gestionan las excepciones levantadas por **`Thread.run()`**.

Guarda *exc_value* usando un *hook* personalizado puede crear un ciclo de referencias. Debe ser aclarado explícitamente que se rompa el ciclo de referencias cuando la excepción ya no se necesite.

Guardando *thread* usando un *hook* personalizado puede resucitarlo si se asigna a un objeto que esté siendo finalizado. Evítese que *thread* sea almacenado después de que el *hook* personalizado se complete para evitar resucitar objetos.

Ver también

`sys.excepthook()` gestiona excepciones no capturadas.

Added in version 3.8.

threading.__excepthook__

Mantiene el valor original de `threading.excepthook()`. Se guarda para que se pueda restaurar el valor original en caso de que se reemplacen con objetos rotos o alternativos.

Added in version 3.10.

threading.get_ident()

Retorna el "identificador de hilo" del hilo actual. Éste es un entero distinto de cero. Su valor no tiene un significado directo; ha sido pensado como una *cookie* mágica para usarse, por ejemplo, en indexar un diccionario con datos específicos del hilo. Los identificadores de hilo pueden ser reciclados cuando se abandona un hilo y se crea otro hilo.

Added in version 3.3.

threading.get_native_id()

Retorna la ID de Hilo (*Thread ID*) nativo integral del hilo actual asignado por el kernel. Este es un entero distinto de cero. Su valor puede utilizarse para identificar de forma única a este hilo en particular a través de todo el sistema (hasta que el hilo termine, luego de lo cual el valor puede ser reciclado por el SO).

Availability: Windows, FreeBSD, Linux, macOS, OpenBSD, NetBSD, AIX, DragonFlyBSD, GNU/kFreeBSD.

Added in version 3.8.

Distinto en la versión 3.13: Added support for GNU/kFreeBSD.

threading.enumerate()

Retorna una lista de todos los objetos `Thread` actualmente activos. La lista incluye subprocesos demoníacos y objetos de subprocesos ficticios creados por `current_thread()`. Excluye los subprocesos terminados y los subprocesos que aún no se han iniciado. Sin embargo, el hilo principal siempre es parte del resultado, incluso cuando se termina.

threading.main_thread()

Retorna el objeto `Thread` principal. En condiciones normales, el hilo principal es el hilo desde el que fue inicializado el intérprete de Python.

Added in version 3.4.

threading.settrace(*func*)

Set a trace function for all threads started from the **`threading`** module. The *func* will be passed to `sys.settrace()` for each thread, before its `run()` method is called.

threading.settrace_all_threads(*func*)

Set a trace function for all threads started from the **`threading`** module and all Python threads that are currently executing.

La *func* será pasada a `sys.settrace()` para cada hilo, antes de que su método `run()` sea invocado.

Added in version 3.12.

threading.gettrace()

Obtiene la función de rastreo según lo establecido por `settrace()`.

Added in version 3.10.

threading.setprofile(*func*)

Set a profile function for all threads started from the **`threading`** module. The *func* will be passed to `sys.setprofile()` for each thread, before its `run()` method is called.

threading.setprofile_all_threads(*func*)

Set a profile function for all threads started from the **`threading`** module and all Python threads that are currently executing.

La *func* se pasará a `sys.setprofile()` para cada hilo, antes de que se llame a su método `run()`.

Added in version 3.12.

threading.getprofile()

Obtiene la función de generador de perfiles establecida por `setprofile()`.

Added in version 3.10.

threading.stack_size([*size*])

Retorna el tamaño de pila usado para crear nuevos hilos. El argumento opcional *size* (tamaño) especifica el tamaño de pila a ser utilizado para hilos creados posteriormente, y debe ser 0 (usar el valor por defecto de la plataforma o el configurado) o un valor entero positivo de al menos 32.768 (32KiB). Si no se especifica *size*, se usará 0. Si no existe soporte para cambiar el tamaño de pila, se lanzará un `RuntimeError`. Si el tamaño de pila especificado es inválido, se lanzará un `ValueError` y el tamaño de pila no será modificado. El tamaño mínimo de pila actualmente soportado es de 32KiB para garantizar suficiente espacio de pila para el intérprete mismo. Nótese que algunas plataformas pueden tener restricciones particulares de valores para tamaños de pila, como requerir un tamaño de pila > 32KiB, o requerir una asignación en múltiplos del tamaño de página de la memoria del sistema. Debe consultarse la documentación de cada plataforma para mayor información (páginas de 4KiB son comunes; se recomienda el uso de múltiplos de 4096 para el tamaño de pila en ausencia de información más específica)

Availability: Windows, pthreads.

Plataformas Unix con soporte para subprocesos POSIX.

Este módulo también define la siguiente constante:

threading.TIMEOUT_MAX

El máximo valor permitido para el parámetro *timeout* de las funciones bloqueantes (`Lock.acquire()`, `RLock.acquire()`, `Condition.wait()`, etc.). La especificación de un tiempo de espera mayor a este valor lanzará un `OverflowError`.

Added in version 3.2.

Este módulo define un número de clases, las cuales son detalladas en las siguientes secciones.

El diseño de este módulo está vagamente basado en el modelo de hilos de Java. Sin embargo, donde Java hace que los bloqueos y las variables de condición sean comportamientos básicos de cada objeto, en Python son objetos separados. La clase `Thread` de Python soporta un subconjunto del comportamiento de la clase Thread de Java; actualmente, no hay prioridades, no hay grupos de hilos, y los hilos no pueden ser destruidos, detenidos, suspendidos, reanudados o interrumpidos. Los métodos estáticos de la clase Thread de Java, cuando se implementan, se asignan a funciones de nivel de módulo.

Todos los métodos descritos abajo son ejecutados de manera atómica.

Thread-local data

Thread-local data is data whose values are thread specific. If you have data that you want to be local to a thread, create a **`local`** object and use its attributes:

```
>>> mydata = local()
>>> mydata.number = 42
>>> mydata.number
42
```

You can also access the **`local`**-object's dictionary:

```
>>> mydata.__dict__
{'number': 42}
>>> mydata.__dict__.setdefault('widgets', [])
[]
>>> mydata.widgets
[]
```

If we access the data in a different thread:

```
>>> log = []
>>> def f():
...     items = sorted(mydata.__dict__.items())
...     log.append(items)
...     mydata.number = 11
...     log.append(mydata.number)

>>> import threading
>>> thread = threading.Thread(target=f)
>>> thread.start()
>>> thread.join()
>>> log
[[], 11]
```

we get different data. Furthermore, changes made in the other thread don’t affect data seen in this thread:

```
>>> mydata.number
42
```

Of course, values you get from a `local` object, including their `__dict__` attribute, are for whatever thread was current at the time the attribute was read. For that reason, you generally don’t want to save these values across threads, as they apply only to the thread they came from.

You can create custom `local` objects by subclassing the `local` class:

```
>>> class MyLocal(local):
...     number = 2
...     def __init__(self, /, **kw):
...         self.__dict__.update(kw)
...     def squared(self):
...         return self.number ** 2
```

This can be useful to support default values, methods and initialization. Note that if you define an `__init__()` method, it will be called each time the `local` object is used in a separate thread. This is necessary to initialize each thread’s dictionary.

Now if we create a `local` object:

```
>>> mydata = MyLocal(color='red')
```

we have a default number:

```
>>> mydata.number
2
```

an initial color:

```
>>> mydata.color
'red'
>>> del mydata.color
```

And a method that operates on the data:

```
>>> mydata.squared()
4
```

As before, we can access the data in a separate thread:

```
>>> log = []
>>> thread = threading.Thread(target=f)
>>> thread.start()
>>> thread.join()
>>> log
[[('color', 'red')], 11]
```

without affecting this thread's data:

```
>>> mydata.number
2
>>> mydata.color
Traceback (most recent call last):
...
AttributeError: 'MyLocal' object has no attribute 'color
```

Note that subclasses can define __slots__, but they are not thread local. They are shared across threads:

```
>>> class MyLocal(local):
...     __slots__ = 'number'

>>> mydata = MyLocal()
>>> mydata.number = 42
>>> mydata.color = 'red'
```

So, the separate thread:

```
>>> thread = threading.Thread(target=f)
>>> thread.start()
>>> thread.join()
```

affects what we see:

```
>>> mydata.number
11
```

class threading.local

Una clase que representa datos locales de hilo.

Thread objects

La clase `Thread` representa una actividad que corre en un hilo de control separado. Hay dos manera de especificar la actividad: pasando un objeto invocable al constructor, o sobrescribiendo el método `run()` en una subclase. Ningún otro método (a excepción del constructor) deberá ser sobrescrito en una subclase. En otras palabras, *solo* sobrescribir los métodos `__init__()` y **`run()`** de esta clase.

Una vez que un objeto *thread* es creado, su actividad debe ser iniciada llamando al método `start()` del hilo. Ésto invoca el método `run()` en un hilo de control separado.

Una vez que la actividad del hilo ha sido iniciada, el hilo se considerará "vivo". Deja de estar vivo cuando su método `run()` termina – ya sea normalmente, o por lanzar una excepción no manejada. El método `is_alive()` verifica si acaso el hilo está vivo.

Otras hilos pueden llamar al método `join()`. Esto bloquea el hilo que llama hasta que el hilo cuyo método **`join()`** es llamado termina.

Un hilo tiene un nombre. El nombre puede ser pasado al constructor y leído o cambiado a través del atributo `name`.

Si el método `run()` lanza una excepción, se llama a `threading.excepthook()` para gestionarla. Por defecto, **`threading.excepthook()`** ignora silenciosamente a `SystemExit`.

Un hilo puede ser marcado como un «hilo demonio». El significado de esta marca es que la totalidad del programa de Python finalizará cuando solo queden hilos demonio. El valor inicial es heredado del hilo creador. La marca puede ser establecida a través de la propiedad **`daemon`** o del argumento *daemon* en el constructor.

Nota

Los hilos demonio son detenidos abruptamente al momento del cierre. Sus recursos (tales como archivos abiertos, transacciones con bases de datos, etc.) pueden no ser liberados adecuadamente. Si se requiere que los hilos se detengan con gracia, háganse no-demoníacos y úsese un mecanismo de señalización adecuado tal como un **`Event`**.

Existe un objeto «hilo principal»; éste corresponde al hilo de control inicial del programa de Python. No es un hilo demonio.

Existe la posibilidad de crear «objetos de hilos *dummy*». Estos son objetos hilo correspondientes a «hilos extranjeros», que son hilos de control iniciados afuera del modulo *threading*, por ejemplo directamente de código en C. Los objetos de hilos *dummy* tienen funcionalidad limitada; siempre se consideran vivos y demoníacos, y no pueden se les puede aplicar el método joined. Nunca son eliminados, ya que es imposible detectar la terminación de hilos extranjeros.

class threading.Thread(*group=None, target=None, name=None, args=(), kwargs={}, *, daemon=None, context=None*)

Este constructor siempre debe ser llamado con argumentos de palabra clave. Los argumentos son:

group debe ser *None*; reservado para una futura extensión cuando se implemente una clase **`ThreadGroup`**.

target es el objeto invocable a ser invocado por el método **`run()`**. Por defecto es `None`, lo que significa que nada es llamado.

name es el nombre del hilo. De forma predeterminada, se construye un nombre único con el formato «Hilo-*N*», donde *N* es un número decimal pequeño, o «Hilo-*N* (target)» donde «target» es `target.__name__` si se especifica el argumento *target*.

args es una lista o tupla de argumentos para la invocación de destino. Por defecto es `()`.

kwargs es un diccionario de argumentos de palabra clave para la invocación objetivo. Por defecto es `{}`.

Si no es `None`, *daemon* establece explícitamente si el hilo es demoníaco. Si es `None` (el valor por defecto), la propiedad demoníaca es heredada del hilo actual.

context is the **`Context`** value to use when starting the thread. The default value is `None` which indicates that the **`sys.flags.thread_inherit_context`** flag controls the behaviour. If the flag is true, threads will start with a copy of the context of the caller of **`start()`**. If false, they will start with an empty context. To explicitly start with an empty context, pass a new instance of **`Context()`**. To explicitly start with a copy of the current context, pass the value from **`copy_context()`**. The flag defaults true on free-threaded builds and false otherwise.

Si la subclase sobrescribe el constructor, debe asegurarse de invocar al constructor de la clase base (`Thread.__init__()`) antes de hacer cualquier otra cosa al hilo.

Distinto en la versión 3.3: Added the *daemon* parameter.

Distinto en la versión 3.10: Utilice el nombre *target* si se omite el argumento *name*.

Distinto en la versión 3.14: Added the *context* parameter.

start()

Inicia la actividad del hilo.

Debe ser llamada máximo una vez por objeto hilo. Se encarga de que el método **`run()`** del objeto sea invocado en un hilo de control separado.

Este método lanzará un `RuntimeError` si se llama más de una vez en el mismo objeto hilo.

If supported, set the operating system thread name to `threading.Thread.name`. The name can be truncated depending on the operating system thread name limits.

Distinto en la versión 3.14: Set the operating system thread name.

run()

Método que representa la actividad del hilo.

Se puede sobrescribir este método en una subclase. El método estándar `run()` invoca el objeto invocable pasado al constructor del objeto como argumento *target*, si lo hay, con argumentos posicionales y de palabra clave tomados de los argumentos *args* y *kwargs*, respectivamente.

Usar una lista o tupla como el argumento *args* que se pasa a `Thread` podría lograr el mismo efecto.

Ejemplo:

```
>>> from threading import Thread
>>> t = Thread(target=print, args=[1])
>>> t.run()
1
>>> t = Thread(target=print, args=(1,))
>>> t.run()
1
```

join(*timeout=None*)

Espera a que el hilo termine. Esto bloquea el hilo llamador hasta que el hilo cuyo método `join()` es llamado finalice – ya sea normalmente o a través de una excepción no gestionada – o hasta que el tiempo de espera opcional caduque.

When the *timeout* argument is present and not `None`, it should be a floating-point number specifying a timeout for the operation in seconds (or fractions thereof). As **`join()`** always returns `None`, you must call **`is_alive()`** after **`join()`** to decide whether a timeout happened – if the thread is still alive, the **`join()`** call timed out.

Cuando el argumento *timeout* no se presenta o es `None`, la operación bloqueará hasta que el hilo termine.

Un hilo puede unirse varias veces.

`join()` lanza un **`RuntimeError`** si se intenta unir el hilo actual ya que ello generaría un punto muerto. También es un error aplicar **`join()`** a un hilo antes de que haya sido iniciado y los intentos de hacerlo lanzaran la misma excepción.

If an attempt is made to join a running daemonic thread in late stages of Python finalization **`join()`** raises a **`PythonFinalizationError`**.

Distinto en la versión 3.14: May raise **`PythonFinalizationError`**.

name

Un *string* utilizado con propósitos de identificación. No posee semántica. Se puede dar el mismo nombre a múltiples hilos. El nombre inicial es establecido por el constructor.

On some platforms, the thread name is set at the operating system level when the thread starts, so that it is visible in task managers. This name may be truncated to fit in a system-specific limit (for example, 15 bytes on Linux or 63 bytes on macOS).

Changes to *name* are only reflected at the OS level when the currently running thread is renamed. (Setting the *name* attribute of a different thread only updates the Python Thread object.)

getName()
setName()

API getter/setter obsoleta para **`name`**; utilícelo directamente como una propiedad en su lugar.

Obsoleto desde la versión 3.10.

ident

El "identificador de hilo" de este hilo o `None` si el hilo no ha sido iniciado. Es un entero distinto de cero. Ver la función **`get_ident()`**. Los identificadores de hilos pueden ser reciclados cuando un hilo finaliza y otro hilo es creado. El identificador está disponible incuso después de que el hilo ha abandonado.

native_id

El ID de subproceso (`TID`) de este subproceso, según lo asignado por el sistema operativo (kernel). Este es un número entero no negativo, o `None` si el hilo no se ha iniciado. Consulte la función **`get_native_id()`**. Este valor se puede usar para identificar de forma única este hilo en particular en todo el sistema (hasta que el hilo termine, después de lo cual el sistema operativo puede reciclar el valor).

Nota

Similar a las *Process IDs*, las *Thread IDs* sólo son válidas (garantizadas como únicas a través de todo el sistema) desde el momento en que se crea el hilo hasta que el hilo es finalizado.

Availability: Windows, FreeBSD, Linux, macOS, OpenBSD, NetBSD, AIX, DragonFlyBSD.

Added in version 3.8.

is_alive()

Retornar si acaso el hilo está vivo.

Este método retorna `True` desde justo antes de que el método `run()` inicie hasta junto antes de que el método **`run()`** termine. La función `enumerate()` del módulo retorna una lista de todos los hilos vivos.

daemon

Un valor booleano que indica si este hilo es un hilo demonio (*True*) o no (*False*). Debe ser establecido antes de que se llame a `start()`, de lo contrario se lanzará un `RuntimeError`. Su valor inicial se hereda del hilo creador; el hilo principal no es un hilo demonio y por lo tanto todos los hilos creados en el hilo principal tienen por defecto un valor `daemon` = `False`.

El programa de Python en su totalidad finaliza cuando no queda ningún hilo no-demonio vivo.

isDaemon()

setDaemon()

API getter/setter obsoleta para `daemon`; utilícelo directamente como una propiedad en su lugar.

Obsoleto desde la versión 3.10.

Lock objects

Una primitiva *lock*, es una primitiva de sincronización que no pertenece a ningún hilo en particular cuando está cerrado. En Python, es la primitiva de sincronización de más bajo nivel actualmente disponible, implementado directamente por el módulo de extensión `_thread`.

Una primitiva *lock* está en uno de dos estados, «cerrado» o «abierto» (*locked/unlocked*). Se crea en estado abierto. Tiene dos métodos básicos, `acquire()` (adquirir) y `release()` (liberar). Cuando el estado es *abierto*, **`acquire()`** cambia el estado a cerrado y retorna inmediatamente. Cuando el estado es *cerrado*, `acquire()` bloquea hasta que una llamada a `release()` en otro hilo lo cambie a abierto, luego la llamada a `acquire()` lo restablece a cerrado y retorna. El método `release()` sólo debe ser llamado en el estado cerrado; cambia el estado a abierto y retorna inmediatamente. Si se realiza un intento de liberar un *lock* abierto, se lanzará un `RuntimeError`.

Los *locks* también soportan el protocolo de gestión de contexto.

Cuando más de un hilo está bloqueado en `acquire()` esperando que el estado sea abierto, sólo un hilo procederá cuando una llamada a `release()` restablezca el estado a abierto; cuál de los hilos en espera procederá no está definido, y puede variar a través de las implementaciones.

Todos los métodos se ejecutan de manera atómica.

class threading.Lock

La clase que implemente los objetos de la primitiva *lock*. Una vez que un hilo ha adquirido un *lock*, intentos subsecuentes por adquirirlo bloquearán, hasta que sea liberado; cualquier hilo puede liberarlo.

Distinto en la versión 3.13: `Lock` is now a class. In earlier Pythons, `Lock` was a factory function which returned an instance of the underlying private lock type.

acquire(*blocking=True*, *timeout=-1*)

Adquirir un *lock*, bloqueante o no bloqueante.

Cuando se invoca con el argumento *blocking* establecido como `True` (el valor por defecto), bloquea hasta que el *lock* se abra, luego lo establece como cerrado y retorna `True`.

Cuando es invocado con el argumento *blocking* como `False`, no bloquea. Si una llamada con *blocking* establecido como `True` bloqueara, retorna `Falso` inmediatamente; de otro modo, cierra el *lock* y retorna `True`.

Cuando se invoca con el argumento de punto flotante *timeout* fijado a un valor positivo, bloquea por a lo más el número de segundos especificado en *timeout* y mientras el *lock* no pueda ser adquirido. Un argumento *timeout* de «-1» especifica una espera ilimitada. No está admitido especificar un *timeout* cuando *blocking* es `False`.

El valor de retorno es `True` si el *lock* es adquirido con éxito, `Falso` si no (por ejemplo si *timeout* expiró).

Distinto en la versión 3.2: El parámetro *timeout* es nuevo.

Distinto en la versión 3.2: La adquisición de un *lock* ahora puede ser interrumpida por señales en POSIX si la implementación de hilado subyacente lo soporta.

Distinto en la versión 3.14: Lock acquisition can now be interrupted by signals on Windows.

release()

Libera un *lock*. Puede ser llamado desde cualquier hilo, no solo el hilo que ha adquirido el *lock*.

Cuando el *lock* está cerrado, lo restablece a abierto, y retorna. Si cualquier otro hilo está bloqueado esperando que el *lock* se abra, permite que exactamente uno de ellos proceda.

Cuando se invoca en un *lock* abierto, se lanza un **`RuntimeError`**.

No hay valor de retorno.

locked()

Retorna `True` si se adquiere el bloqueo.

RLock objects

Un *lock* reentrante es una primitiva de sincronización que puede ser adquirido múltiples veces por el mismo hilo. Internamente, utiliza el concepto de «hilo dueño» y «nivel de recursividad» además del estado abierto/cerrado utilizado por las primitivas *locks*. Si está en estado cerrado, algún hilo es dueño del *lock*; si está en estado abierto, ningún hilo es dueño.

Threads call a lock's `acquire()` method to lock it, and its `release()` method to unlock it.

Nota

Reentrant locks support the context management protocol, so it is recommended to use `with` instead of manually calling `acquire()` and `release()` to handle acquiring and releasing the lock for a block of code.

RLock's `acquire()/release()` call pairs may be nested, unlike Lock's `acquire()/release()`. Only the final `release()` (the `release()` of the outermost pair) resets the lock to an unlocked state and allows another thread blocked in `acquire()` to proceed.

`acquire()/release()` must be used in pairs: each acquire must have a release in the thread that has acquired the lock. Failing to call release as many times the lock has been acquired can lead to deadlock.

class threading.RLock

Esta clase implementa objetos tipo *lock* reentrantes. Un *lock* reentrante debe ser liberado por el hilo que lo adquirió. Una vez que un hilo ha adquirido un *lock* reentrante, el mismo hilo puede adquirirlo otra vez sin bloquearse; el hilo debe liberarlo una vez por vez que lo adquiere.

Nótese que `RLock` en realidad es una función fábrica que retorna una instancia de la versión más eficiente de la clase RLock concreta que sea soportada por la plataforma.

acquire(*blocking=True*, *timeout=-1*)

Adquirir un *lock*, bloqueante o no bloqueante.

Ver también

Using RLock as a context manager
Recommended over manual **`acquire()`** and `release()` calls whenever practical.

When invoked with the *blocking* argument set to `True` (the default):

- If no thread owns the lock, acquire the lock and return immediately.
- If another thread owns the lock, block until we are able to acquire lock, or *timeout*, if set to a positive float value.
- If the same thread owns the lock, acquire the lock again, and return immediately. This is the difference between `Lock` and **`RLock`**; **`Lock`** handles this case the same as the previous, blocking until the lock can be acquired.

When invoked with the *blocking* argument set to `False`:

- If no thread owns the lock, acquire the lock and return immediately.
- If another thread owns the lock, return immediately.
- If the same thread owns the lock, acquire the lock again and return immediately.

In all cases, if the thread was able to acquire the lock, return `True`. If the thread was unable to acquire the lock (i.e. if not blocking or the timeout was reached)

return `False`.

If called multiple times, failing to call **`release()`** as many times may lead to deadlock. Consider using **`RLock`** as a context manager rather than calling acquire/release directly.

Distinto en la versión 3.2: El parámetro *timeout* es nuevo.

release()

Libera un *lock*, disminuyendo el nivel de recursividad. Si después de la disminución es cero, restablece el *lock* a abierto (no perteneciente a ningún hilo), y si cualquier otro hilo está bloqueado esperando que se abra el *lock*, permite que exactamente uno de ellos proceda. Si luego de la disminución el nivel de recursividad todavía no es cero, el *lock* permanece cerrado y perteneciente al hilo llamador.

Only call this method when the calling thread owns the lock. A **`RuntimeError`** is raised if this method is called when the lock is not acquired.

No hay valor de retorno.

locked()

Return a boolean indicating whether this object is locked right now.

Added in version 3.14.

Condition objects

Una condición variable siempre va asociada a algún tipo de *lock*. éste puede ser provisto o se creará uno por defecto. Proveer uno es útil cuando varias variables de condición deben compartir el mismo *lock*. El *lock* es parte del objeto condicional: no es necesario rastrearlo por separado.

Una condición variable obedece el protocolo de gestión de contexto:

al usar la declaración `with` se adquiere el *lock* asociado por la duración del bloque contenido. Los métodos **`acquire()`** y **`release()`** también llaman los métodos correspondientes del *lock* asociado.

Otros métodos deben llamarse con el *lock* asociado conservado. El método **`wait()`** libera el *lock*, y luego bloquea hasta que otro hilo lo despierte llamando **`notify()`** o **`notify_all()`**. Una vez que ha sido despertado, **`wait()`** re-adquiere el *lock* y retorna. También es posible especificar un tiempo de espera.

El método **`notify()`** despierta a uno de los hilos que esperan a la condición variable, si es que alguno espera. El método **`notify_all()`** despierta a todos los hilos que estén esperando a la condición variable.

Nota: Los métodos **`notify()`** y **`notify_all()`** no liberan el *lock*; esto significa que el hilo o los hilos que han sido despertados no retornaran de su llamada de **`wait()`** inmediatamente, sino solo una vez que el hilo que haya llamado a **`notify()`** o **`notify_all()`** renuncie finalmente a la propiedad del *lock*.

El estilo típico de programación con variables condicionales utiliza el *lock* para sincronizar el acceso a algún estado compartido; hilos que estén interesados en un cambio de estado en particular llamarán a **`wait()`** reiteradamente hasta que vean el estado deseado, mientras que los hilos que modifiquen el estado llamarán a **`notify()`** o a **`notify_all()`** cuando cambien el estado de modo que pudiera ser que el el estado sea el deseado por alguno de los hilos en espera. Por ejemplo, el siguiente código es una situación genérica de productor-consumidor con capacidad de búfer ilimitada:

```
# Consume one item
with cv:
    while not an_item_is_available():
        cv.wait()
    get_an_available_item()

# Produce one item
```

```
with cv:
    make_an_item_available()
    cv.notify()
```

El bucle `while` que verifica la condición de la aplicación es necesario porque **`wait()`** puede retornar después de una cantidad arbitraria de tiempo, y la condición que dio pie a la llamada de **`notify()`** puede ya no ser verdadera. Esto es inherente a la programación multi-hilo. El método **`wait_for()`** puede usarse para automatizar la revisión de condiciones, y facilita la computación de tiempos de espera:

```
# Consume an item
with cv:
    cv.wait_for(an_item_is_available)
    get_an_available_item()
```

Para elegir entre **`notify()`** y **`notify_all()`**, considérese si un cambio de estado puede ser interesante para uno o varios hilos en espera. Por ejemplo en una típica situación productor-consumidor, agregar un elemento al búfer sólo necesita despertar un hilo consumidor.

class threading.Condition(*lock=None*)

Esta clase implementa objetos de condición variable. Una condición variable permite que uno o más hilos esperen hasta que sean notificados por otro hilo.

Si se provee un argumento *lock* distinto de `None`, debe ser un objeto **`Lock`** o **`RLock`**, y se utiliza como el *lock* subyacente. De otro modo, se crea un nuevo objeto **`RLock`** y se utiliza como el *lock* subyacente.

Distinto en la versión 3.3: cambiado de función de fábrica a una clase.

acquire(**args*)

Adquiere el *lock* subyacente. Este método llama al método correspondiente sobre el *lock* subyacente; el valor de retorno es lo que retorne aquel método.

release()

Libera el *lock* subyacente. Este método llama al método correspondiente en el *lock* subyacente; no tiene valor de retorno.

locked()

Return a boolean indicating whether this object is locked right now.

Added in version 3.14.

wait(*timeout=None*)

Espera hasta ser notificado o hasta que el tiempo de espera caduque. Si el hilo invocador no ha adquirido el *lock* cuando este método es llamado, se lanza un **`RuntimeError`**.

Este método libera el *lock* subyacente, y luego bloquea hasta ser despertado por una llamada a **`notify()`** o **`notify_all()`** para la misma condición variable en otro hilo, o hasta que el tiempo de espera opcional se cumpla. Una vez que ha sido despertado o el tiempo de espera ha pasado, re-adquiere el *lock* y retorna.

When the *timeout* argument is present and not `None`, it should be a floating-point number specifying a timeout for the operation in seconds (or fractions thereof).

Cuando el *lock* subyacente es un **`RLock`**, no se libera utilizando su método **`release()`**, ya que esto podría no abrir realmente el *lock* cuando haya sido adquirido múltiples veces recursivamente. En cambio, se usa una interfaz interna de la clase **`RLock`**, que lo abre realmente incluso cuando haya sido adquirido múltiples veces recursivamente. Otra interfaz interna se usa luego para restablecer el nivel de recursividad cuando el *lock* es readquirido.

El valor de retorno es `True` a menos que un *timeout* dado haya expirado, en cuyo caso será `False`.

Distinto en la versión 3.2: Previamente, el método siempre retornaba `None`.

wait_for(*predicate*, *timeout=None*)

Espera a que una condición se evalúe como verdadera. *predicate* debe ser un invocable cuyo resultado se interpretará como un valor booleano. Se puede proveer un *timeout* que especifique el máximo tiempo de espera.

Este método utilitario puede llamar a **`wait()`** reiteradas veces hasta que se satisfaga el predicado, o hasta que la espera caduque. El valor de retorno es el último valor de retorno del predicado y se evaluará a `False` si el método ha caducado.

Al ignorar la propiedad *feature*, llamar a este método equivale vagamente a escribir:

```
while not predicate():
    cv.wait()
```

Por ende, aplican las mismas reglas que con **`wait()`**: El *lock* debe ser conservado cuando se llame y es re-adquirido al momento del retorno. El predicado se evalúa con el *lock* conservado.

Added in version 3.2.

notify(*n=1*)

Por defecto, despierta a un hilo que esté esperando por esta condición, si lo existe. Si el hilo llamador no ha adquirido el *lock* cuando se llama este método, se lanza un **`RuntimeError`**.

Este método despierta como máximo *n* de los hilos que estén esperando por la condición variable; no es una opción si no hay hilos esperando.

La implementación actual despierta exactamente *n* hilos, si hay por lo menos *n* hilos esperando. Sin embargo, no es seguro apoyarse en este comportamiento. A futuro, una implementación optimizada podría ocasionalmente despertar a más de *n* hilos.

Nota: un hilo que ha sido despertado no retorna realmente de su llamada a **`wait()`** hasta que pueda readquirir el *lock*. Ya que **`notify()`** no libera el *lock*, su llamador debiera hacerlo.

notify_all()

Despierta a todos los hilos que esperen por esta condición. Este método actúa como **`notify()`**, pero despierta a todos los hilos en espera en vez de a uno. Si el hilo llamador no ha adquirido el *lock* cuando se llama a este método, se lanza un **`RuntimeError`**.

El método `notifyAll` es un alias obsoleto para este método.

Semaphore objects

Éste es uno de las primitivas de sincronización más antiguos en la historia de las ciencias de la computación, inventado por el pionero en ciencias de la computación holandés Edsger W. Dijkstra (él utilizó los nombres `P()` y `V()` en lugar de **`acquire()`** y **`release()`**)

Un semáforo administra un contador interno que se disminuye por cada llamada a **`acquire()`** y se incrementa por cada llamada a **`release()`**. El contador no puede bajar de cero; cuando **`acquire()`** lo encuentra en cero, bloquea, esperando hasta que otro hilo llame **`release()`**.

Los semáforos también tienen soporte para el protocolo de gestión de contexto.

class threading.Semaphore(*value=1*)

Esta clase implementa los objetos semáforo. Un semáforo gestiona un contador atómico que representa el número de llamadas a **`release()`** menos el número de llamadas a **`acquire()`**, más un valor inicial. El método **`acquire()`** bloquea si es necesario, hasta que pueda retornar sin volver el contador negativo. Si no es provisto, el valor por defecto de *value* será 1.

El argumento opcional da el *value* inicial al contador interno; por defecto es `1`. Si el *value* provisto es menor a 0; se lanza un **`ValueError`**.

Distinto en la versión 3.3: cambiado de función de fábrica a una clase.

acquire(*blocking=True*, *timeout=None*)

Adquirir un semáforo.

Cuando se invoca sin argumentos:

- Si el contador interno es mayor a cero de entrada, lo disminuye en uno y retorna `True` inmediatamente.
- Si el contador interno es cero de entrada, bloquea hasta ser despertado por una llamada a **`release()`**. Una vez despierto (y el contador sea mayor a 0), disminuye el contador en 1 y retorna `True`. Se despertará exactamente un hilo por cada llamada a **`release()`**. No debiese confiarse en el orden en que los hilos sean despertados.

Cuando se invoca con *blocking* fijado en `False`, no bloquea. Si una llamada sin un argumento bloquease, retorna `False` inmediatamente; de otro modo, hace lo mismo que cuando se llama sin argumentos, y retorna `True`.

Cuando se invoca con *timeout* distinto de `None`, bloqueará por un tiempo máximo en segundos fijados en *timeout*. Si *acquire* no se completa exitosamente en ese intervalo, retorna `False`. De otro modo retorna `True`.

Distinto en la versión 3.2: El parámetro *timeout* es nuevo.

release(*n=1*)

Suelta un semáforo, incrementando el contador interno por *n*. Cuando era cero en la entrada y otros subprocesos están esperando que vuelva a ser mayor que cero, active *n* de esos subprocesos.

Distinto en la versión 3.9: Se agregó el parámetro *n* para liberar varios subprocesos en espera a la vez.

class threading.BoundedSemaphore(*value=1*)

Clase que implementa objetos de semáforo delimitados. Un semáforo delimitado verifica que su valor actual no exceda su valor inicial. Si lo hace, se lanza un **`ValueError`**. En la mayoría de las situaciones se utilizan los semáforos para cuidar recursos con capacidad limitada. Si se libera el semáforo demasiadas veces es signo de un *bug*. Si no se provee, el valor por defecto de *value* será 1.

Distinto en la versión 3.3: cambiado de función de fábrica a una clase.

`Semaphore` example

Los semáforos suelen utilizarse para cuidar recursos con capacidad limitada, por ejemplo, un servidor de base de datos. En cualquier situación en que el tamaño de los recursos sea fijo, se debe usar un semáforo delimitado. Antes de generar cualquier hilo de trabajo, tu hilo principal debe inicializar el semáforo:

```
maxconnections = 5
# ...
pool_sema = BoundedSemaphore(value=maxconnections)
```

Una vez que han sido generados, los hilos de trabajo llaman a los métodos *acquire* y *release* cuando necesitan conectarse al servidor:

```
with pool_sema:
    conn = connectdb()
    try:
        # ... use connection ...
    finally:
        conn.close()
```

El uso de semáforos delimitados reduce la posibilidad de que pase inadvertido un error de programación que cause que el semáforo sea liberado más veces de las que sea adquirido.

Event objects

Éste es uno de los mecanismos más simples de comunicación entre hilos: un hilo señala un evento y otro hilo lo espera.

Un objeto de evento maneja una marca interna que puede ser establecida como verdadera mediante el método **`set()`** y restablecida a falsa mediante el método **`clear()`**. El método **`wait()`** bloquea hasta que la marca sea *true*.

class threading.Event

Clase que implementa los objetos de evento. Un evento gestiona un indicador que puede ser establecido a verdadero mediante el método **`set()`** y restablecido a falso con el método **`clear()`**. El método **`wait()`** bloquea hasta que el indicador sea verdadero. El indicador es inicialmente falso.

Distinto en la versión 3.3: cambiado de función de fábrica a una clase.

is_set()

Retorna `True` exclusivamente si el indicador interno es verdadero.

El método `isSet` es un alias obsoleto para este método.

set()

Establece el indicador interno a verdadero. Todos los hilos que estén esperando que se vuelva verdadero serán despertados. Los hilos que llaman a `wait()` una vez que el indicador marca verdadero no bloquearán.

clear()

Restablece el indicador a falso. Posteriormente, los hilos que llamen a `wait()` bloquearán hasta que se llame a `set()` para establecer el indicador interno a verdadero nuevamente.

wait(*timeout=None*)

Block as long as the internal flag is false and the timeout, if given, has not expired. The return value represents the reason that this blocking method returned; `True` if returning because the internal flag is set to true, or `False` if a timeout is given and the internal flag did not become true within the given wait time.

When the timeout argument is present and not `None`, it should be a floating-point number specifying a timeout for the operation in seconds, or fractions thereof.

Distinto en la versión 3.1: Previamente, el método siempre retornaba `None`.

Timer objects

Esta clase representa una acción que sólo debe ejecutarse luego de que una cierta cantidad de tiempo transcurra — un temporizador. `Timer` es una subclase de `Thread` y en tanto tal también funciona como un ejemplo de creación de hilos personalizados.

Los temporizadores son iniciados, tal como los hilos, al llamarse su método `Timer.start`. El temporizador puede ser detenido (antes de que su acción haya comenzado) al llamar al método `cancel()`.

El intervalo que el temporizador esperará antes de ejecutar su acción puede no ser exactamente el mismo que el intervalo especificado por el usuario.

Por ejemplo:

```
def hello():
    print("hello, world")

t = Timer(30.0, hello)
t.start()  # after 30 seconds, "hello, world" will be pr
```

class threading.Timer(*interval*, *function*, *args=None*, *kwargs=None*)

Crear un temporizador que ejecutará *function* con los argumentos *args* y los argumentos de palabra clave *kwargs*, luego de que una cantidad *interval* de segundos hayan transcurrido. Si *args* es `None` (por defecto) se utilizará una lista vacía. Si *kwargs* es `None` (por defecto) se utilizará un *dict* vacío.

Distinto en la versión 3.3: cambiado de función de fábrica a una clase.

cancel()

Detiene el temporizador, y cancela la ejecución de la acción del temporizador. Esto sólo funcionará si el temporizador está en etapa de espera.

Barrier objects

Added in version 3.2.

Esta clase provee una primitiva de sincronización simple para ser usado por un número fijo de hilos que necesitan esperarse entre ellos. Cada uno de los hilos intenta pasar la barrera llamando al método **`wait()`** y bloqueará hasta que todos los hilos hayan hecho sus respectivas llamadas a **`wait()`**. En este punto, los hilos son liberados simultáneamente.

La barrera puede ser reutilizada cualquier número de veces para el mismo número de hilos.

Como ejemplo, aquí hay una manera simple de sincronizar un hilo cliente con uno servidor:

```
b = Barrier(2, timeout=5)

def server():
    start_server()
    b.wait()
    while True:
        connection = accept_connection()
        process_server_connection(connection)

def client():
    b.wait()
    while True:
        connection = make_connection()
        process_client_connection(connection)
```

class threading.Barrier(*parties*, *action=None*, *timeout=None*)

Crear un objeto de barrera para un número *parties* de hilos. Una *action*, si es provista, es un invocable a ser llamado por uno de los hilos cuando sean liberados. *timeout* es el valor de tiempo de espera máximo por defecto si no se especifica uno en el método **`wait()`**.

wait(*timeout=None*)

Pasa la barrera. Cuando todos los hilos involucrados en el objeto barrera han llamado esta función, se liberan todos simultáneamente. Si se provee un valor *timeout*, se utilizará con preferencia sobre cualquiera que haya sido suministrado al constructor de la clase.

El valor de retorno es un entero en el rango desde 0 hasta *parties* – 1, diferente para cada hilo. Puede ser utilizado para seleccionar a un hilo para que haga alguna limpieza especial, por ejemplo:

```
i = barrier.wait()
if i == 0:
    # Only one thread needs to print this
    print("passed the barrier")
```

Se se provee una *action* al constructor, uno de los hilos la habrá llamado antes de ser liberado. Si acaso esta llamada lanzara un error, la barrera entra en estado *broken* (roto).

Si se agota el tiempo de llamada, la barrera pasa al estado roto.

Este método podría lanzar una excepción `BrokenBarrierError` si la barrera está rota o si se reinicia mientras el hilo está esperando.

reset()

Retorna la barrera al estado por defecto, vacío. Cualquier hilo que esté a su espera recibirá la excepción `BrokenBarrierError`.

Nótese que utilizar esta función podría requerir alguna sincronización externa si existen otros hilos cuyos estados sean desconocidos. Si una barrera se rompe puede ser mejor abandonarla y crear una nueva.

abort()

Coloca la barrera en estado roto. Esto causa que cualquier llamada activa o futura a `wait()` falle con el error `BrokenBarrierError`. Úsese por ejemplo si uno de los hilos necesita abortar, para evitar que la aplicación quede en punto muerto.

Puede ser preferible simplemente crear la barrera con un valor *timeout* sensato para cuidarse automáticamente de que uno de los hilos falle.

parties

El número de hilos requeridos para pasar la barrera.

n_waiting

El número de hilos actualmente esperando en la barrera.

broken

Un valor booleano que será `True` si la barrera está en el estado roto.

exception threading.BrokenBarrierError

Esta excepción, una subclase de **`RuntimeError`**, se lanza cuando el objeto **`Barrier`** se restablece o se rompe.

Uso de *locks*, condiciones y semáforos en la declaración `with`

Todos los objetos provistos por este módulo que tienen métodos `acquire` y `release` pueden ser utilizados como administradores de contexto para una declaración **`with`**. El método `acquire` será llamado cuando se ingresa al bloque y el método `release` será llamado cuando se abandona el bloque. De ahí que, el siguiente fragmento:

```
with some_lock:
    # do something...
```

sea equivalente a:

```
some_lock.acquire()
try:
    # do something...
finally:
    some_lock.release()
```

Actualmente, los objetos **`Lock`**, **`RLock`**, **`Condition`**, **`Semaphore`**, y **`BoundedSemaphore`** pueden ser utilizados como gestores de contexto con declaraciones **`with`**.

`multiprocessing` — Process-based parallelism

Código fuente: Lib/multiprocessing/ [https://github.com/python/cpython/tree/3.14/Lib/multiprocessing/]

Availability: not Android, not iOS, not WASI.

This module is not supported on mobile platforms or WebAssembly platforms.

Introducción

`multiprocessing` is a package that supports spawning processes using an API similar to the **`threading`** module. The **`multiprocessing`** package offers both local and remote concurrency, effectively side-stepping the Global Interpreter Lock by using subprocesses instead of threads. Due to this, the **`multiprocessing`** module allows the programmer to fully leverage multiple processors on a given machine. It runs on both POSIX and Windows.

The **`multiprocessing`** module also introduces the **`Pool`** object which offers a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism). The following example demonstrates the common practice of defining such functions in a module so that child processes can successfully import that module. This basic example of data parallelism using **`Pool`**,

```
from multiprocessing import Pool

def f(x):
    return x*x
```

```
if __name__ == '__main__':
    with Pool(5) as p:
        print(p.map(f, [1, 2, 3]))
```

imprimirá la salida estándar

```
[1, 4, 9]
```

The **`multiprocessing`** module also introduces APIs which do not have analogs in the `threading` module, like the ability to `terminate`, `interrupt` or `kill` a running process.

Ver también

`concurrent.futures.ProcessPoolExecutor` ofrece una interfaz de nivel superior para enviar tareas a un proceso en segundo plano sin bloquear la ejecución del proceso de llamada. En comparación con el uso directo de la interfaz `Pool`, la API `concurrent.futures` permite separar el envío de trabajo al grupo de procesos subyacente de la espera de los resultados.

La clase `Process`

In **`multiprocessing`**, processes are spawned by creating a `Process` object and then calling its `start()` method. **`Process`** follows the API of `threading.Thread`. A trivial example of a multiprocess program is

```
from multiprocessing import Process

def f(name):
    print('hello', name)

if __name__ == '__main__':
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()
```

Para mostrar las IDs individuales involucradas en el proceso, aquí hay un ejemplo ampliado:

```
from multiprocessing import Process
import os

def info(title):
    print(title)
    print('module name:', __name__)
    print('parent process:', os.getppid())
    print('process id:', os.getpid())

def f(name):
    info('function f')
    print('hello', name)

if __name__ == '__main__':
    info('main line')
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()
```

Para obtener una explicación de por qué es necesaria la parte `if __name__ == '__main__'`, consulte Pautas de programación.

The arguments to **`Process`** usually need to be unpickleable from within the child process. If you tried typing the above example directly into a REPL it could lead to an **`AttributeError`** in the child process trying to locate the *f* function in the `__main__` module.

Contextos y métodos de inicio

Depending on the platform, **`multiprocessing`** supports three ways to start a process. These *start methods* are

Generación (*spawn*)

El proceso principal inicia un nuevo proceso de interpretación de Python. El proceso secundario solo heredará los recursos necesarios para ejecutar el método `run()` del objeto de proceso. En particular, los identificadores y descriptores de archivo innecesarios del proceso principal no se heredarán. Iniciar un proceso con este método es bastante lento en comparación con *fork* o *forkserver*.

Available on POSIX and Windows platforms. The default on Windows and macOS.

fork

El proceso parental usa `os.fork()` para bifurcar el intérprete de Python. El proceso hijo, cuando comienza, es efectivamente idéntico al proceso parental. Todos los recursos del proceso parental son heredados por el proceso hijo. Tenga en cuenta que bifurcar (*forking*) de forma segura un proceso multihilo es problemático.

Available on POSIX systems.

Distinto en la versión 3.14: This is no longer the default start method on any platform. Code that requires *fork* must explicitly specify that via `get_context()` or `set_start_method()`.

Distinto en la versión 3.12: If Python is able to detect that your process has multiple threads, the `os.fork()` function that this start method calls internally will raise a `DeprecationWarning`. Use a different start method. See the `os.fork()` documentation for further explanation.

forkserver

When the program starts and selects the *forkserver* start method, a server process is

> spawned. From then on, whenever a new process is needed, the parent process connects to the server and requests that it fork a new process. The fork server process is single threaded unless system libraries or preloaded imports spawn threads as a side-effect so it is generally safe for it to use `os.fork()`. No unnecessary resources are inherited.
>
> Available on POSIX platforms which support passing file descriptors over Unix pipes such as Linux. The default on those.
>
> *Distinto en la versión 3.14:* This became the default start method on POSIX platforms.

Distinto en la versión 3.4: spawn added on all POSIX platforms, and *forkserver* added for some POSIX platforms. Child processes no longer inherit all of the parents inheritable handles on Windows.

Distinto en la versión 3.8: On macOS, the *spawn* start method is now the default. The *fork* start method should be considered unsafe as it can lead to crashes of the subprocess as macOS system libraries may start threads. See bpo-33725 [https://bugs.python.org/issue?@action=redirect&bpo=33725].

Distinto en la versión 3.14: On POSIX platforms the default start method was changed from *fork* to *forkserver* to retain the performance but avoid common multithreaded process incompatibilities. See gh-84559 [https://github.com/python/cpython/issues/84559].

On POSIX using the *spawn* or *forkserver* start methods will also start a *resource tracker* process which tracks the unlinked named system resources (such as named semaphores or `SharedMemory` objects) created by processes of the program. When all processes have exited the resource tracker unlinks any remaining tracked object. Usually there should be none, but if a process was killed by a signal there may be some «leaked» resources. (Neither leaked semaphores nor shared memory segments will be automatically unlinked until the next reboot. This is problematic for both objects because the system allows only a limited number of named semaphores, and shared memory segments occupy some space in the main memory.)

Para seleccionar un método de inicio, utilice la función **`set_start_method()`** en la cláusula `if __name__ == '__main__'` del módulo principal. Por ejemplo:

```
import multiprocessing as mp

def foo(q):
    q.put('hello')

if __name__ == '__main__':
    mp.set_start_method('spawn')
    q = mp.Queue()
    p = mp.Process(target=foo, args=(q,))
    p.start()
    print(q.get())
    p.join()
```

`set_start_method()` no debería ser usada más de una vez en el programa.

Alternativamente, se puede usar **`get_context()`** para obtener un objeto de contexto. Los objetos de contexto tienen la misma API que el módulo de multiprocesamiento, y permiten utilizar múltiples métodos de inicio en el mismo programa.

```
import multiprocessing as mp

def foo(q):
    q.put('hello')

if __name__ == '__main__':
    ctx = mp.get_context('spawn')
    q = ctx.Queue()
    p = ctx.Process(target=foo, args=(q,))
    p.start()
    print(q.get())
    p.join()
```

Tenga en cuenta que los objetos relacionados con un contexto pueden no ser compatibles con procesos para un contexto diferente. En particular, los *locks* creados con el contexto *fork* no se pueden pasar a los procesos iniciados con los métodos de inicio *spawn* o *forkserver* .

Libraries using `multiprocessing` or `ProcessPoolExecutor` should be designed to allow their users to provide their own multiprocessing context. Using a specific context of your own within a library can lead to incompatibilities with the rest of the library user's application. Always document if your library requires a specific start method.

Advertencia

The `'spawn'` and `'forkserver'` start methods generally cannot be used with «frozen» executables (i.e., binaries produced by packages like **PyInstaller** and **cx_Freeze**) on POSIX systems. The `'fork'` start method may work if code does not use threads.

Intercambiando objetos entre procesos

`multiprocessing` supports two types of communication channel between processes:

Colas (*queues*)

La clase `Queue` es prácticamente un clon de `queue.Queue`. Por ejemplo:

```
from multiprocessing import Process, Queue

def f(q):
    q.put([42, None, 'hello'])

if __name__ == '__main__':
    q = Queue()
    p = Process(target=f, args=(q,))
    p.start()
    print(q.get())    # prints "[42, None, 'hello'
    p.join()
```

Queues are thread and process safe. Any object put into a **`multiprocessing`** queue will be serialized.

Tuberías (*Pipes*)

La función `Pipe()` retorna un par de objetos de conexión conectados por una tubería (*pipe*) que, por defecto, es un dúplex (bidireccional). Por ejemplo:

```
from multiprocessing import Process, Pipe

def f(conn):
    conn.send([42, None, 'hello'])
    conn.close()

if __name__ == '__main__':
    parent_conn, child_conn = Pipe()
    p = Process(target=f, args=(child_conn,))
    p.start()
    print(parent_conn.recv())   # prints "[42, Nor
    p.join()
```

Los dos objetos de conexión retornados por `Pipe()` representan los dos extremos de la tubería (*pipe*). Cada objeto de conexión tiene los métodos **`send()`** y **`recv()`** (entre otros). Tenga en cuenta que los datos en una tubería pueden corromperse si dos procesos (o hilos) intentan leer o escribir en el *mismo* (*same*) extremo de la tubería al mismo tiempo. Por supuesto, no hay riesgo de corrupción por procesos que utilizan diferentes extremos de la tubería (*pipe*) al mismo tiempo.

The **`send()`** method serializes the object and

`recv()` re-creates the object.

Sincronización entre procesos

`multiprocessing` contains equivalents of all the synchronization primitives from **`threading`**. For instance one can use a lock to ensure that only one process prints to standard output at a time:

```
from multiprocessing import Process, Lock

def f(l, i):
    l.acquire()
    try:
        print('hello world', i)
    finally:
        l.release()

if __name__ == '__main__':
    lock = Lock()

    for num in range(10):
        Process(target=f, args=(lock, num)).start()
```

Sin usar el candado (*lock*) de salida de los diferentes procesos, es probable que todo se mezcle.

Compartiendo estado entre procesos

Como se mencionó anteriormente, cuando se realiza una programación concurrente, generalmente es mejor evitar el uso del estado compartido en la medida de lo posible. Esto es particularmente cierto cuando se utilizan múltiples procesos.

However, if you really do need to use some shared data then **`multiprocessing`** provides a couple of ways of doing so.

Memoria compartida

Los datos se pueden almacenar en un mapa de memoria compartida usando **`Value`** o **`Array`**. Por ejemplo, el siguiente código

```
from multiprocessing import Process, Value, Array

def f(n, a):
    n.value = 3.1415927
    for i in range(len(a)):
        a[i] = -a[i]

if __name__ == '__main__':
    num = Value('d', 0.0)
    arr = Array('i', range(10))

    p = Process(target=f, args=(num, arr))
    p.start()
    p.join()

    print(num.value)
    print(arr[:])
```

imprimirá

```
3.1415927
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

Los argumentos `'d'` y `'i'` utilizados al crear `num` y `arr` son códigos de tipo del tipo utilizado por **`array`** module: `'d'` indica un flotador de doble precisión y `'i'` indica un entero con signo. Estos objetos compartidos serán seguros para procesos y subprocesos.

Para una mayor flexibilidad en el uso de la memoria compartida, se puede usar el módulo **`multiprocessing.sharedctypes`** que admite la creación arbitraria de objetos *ctypes* asignados desde la memoria compartida.

Proceso servidor (*Server process*)

Un objeto de administrador retornado por **`Manager()`** controla un proceso de servidor que contiene objetos de Python y permite que otros procesos los manipulen usando proxies.

A manager returned by **`Manager()`** will support types **`list`**, **`dict`**, **`set`**, **`Namespace`**, **`Lock`**, **`RLock`**, **`Semaphore`**, **`BoundedSemaphore`**, **`Condition`**, **`Event`**, **`Barrier`**, **`Queue`**, **`Value`** and **`Array`**. For example,

```
from multiprocessing import Process, Manager

def f(d, l, s):
    d[1] = '1'
    d['2'] = 2
    d[0.25] = None
    l.reverse()
    s.add('a')
    s.add('b')

if __name__ == '__main__':
    with Manager() as manager:
        d = manager.dict()
        l = manager.list(range(10))
        s = manager.set()

        p = Process(target=f, args=(d, l, s))
        p.start()
        p.join()

        print(d)
        print(l)
        print(s)
```

imprimirá

```
{0.25: None, 1: '1', '2': 2}
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
{'a', 'b'}
```

> Los administradores de procesos del servidor son más flexibles que el uso de objetos de memoria compartida porque pueden hacerse para admitir tipos de objetos arbitrarios. Por lo tanto, un solo administrador puede ser compartido por procesos en diferentes ordenadores a través de una red. Sin embargo, son más lentos que usar memoria compartida.

Usando una piscina de trabajadores (*pool of workers*)

La clase `Pool` representa procesos de piscina de trabajadores (*pool of workers*). Tiene métodos que permiten que las tareas se descarguen a los procesos de trabajo de diferentes maneras.

Por ejemplo:

```
from multiprocessing import Pool, TimeoutError
import time
import os

def f(x):
    return x*x

if __name__ == '__main__':
    # start 4 worker processes
    with Pool(processes=4) as pool:

        # print "[0, 1, 4,..., 81]"
        print(pool.map(f, range(10)))

        # print same numbers in arbitrary order
        for i in pool.imap_unordered(f, range(10)):
            print(i)

        # evaluate "f(20)" asynchronously
        res = pool.apply_async(f, (20,))      # runs in
        print(res.get(timeout=1))             # prints "
        # evaluate "os.getpid()" asynchronously
        res = pool.apply_async(os.getpid, ()) # runs in
        print(res.get(timeout=1))             # prints t

        # launching multiple evaluations asynchronously
        multiple_results = [pool.apply_async(os.getpid,
        print([res.get(timeout=1) for res in multiple_re

        # make a single worker sleep for 10 seconds
        res = pool.apply_async(time.sleep, (10,))
        try:
            print(res.get(timeout=1))
        except TimeoutError:
            print("We lacked patience and got a multipro

        print("For the moment, the pool remains availabl

    # exiting the 'with'-block has stopped the pool
    print("Now the pool is closed and no longer availabl
```

Tenga en cuenta que los métodos de una piscina (*pool*) solo deben ser utilizados por el proceso que lo creó.

Nota

La funcionalidad en este paquete requiere que los procesos hijos (*children*) puedan importar el módulo `__main__`. Esto está cubierto en Pautas de programación sin embargo, vale la pena señalarlo aquí. Esto significa que algunos ejemplos, como **`multiprocessing.pool.Pool`** no funcionarán en el intérprete interactivo. Por ejemplo:

```
>>> from multiprocessing import Pool
>>> p = Pool(5)
>>> def f(x):
...     return x*x
...
>>> with p:
...     p.map(f, [1,2,3])
```

```
Process PoolWorker-1:
Process PoolWorker-2:
Process PoolWorker-3:
Traceback (most recent call last):
Traceback (most recent call last):
Traceback (most recent call last):
AttributeError: Can't get attribute 'f' on <module '__m
AttributeError: Can't get attribute 'f' on <module '__m
AttributeError: Can't get attribute 'f' on <module '__m
```

(Si intenta esto, en realidad generará tres trazas completas intercaladas de forma semialeatoria, y luego tendrá que detener el proceso principal de alguna manera)

Referencia

The `multiprocessing` package mostly replicates the API of the `threading` module.

Global start method

Python supports several ways to create and initialize a process. The global start method sets the default mechanism for creating a process.

Several multiprocessing functions and methods that may also instantiate certain objects will implicitly set the global start method to the system's default, if it hasn't been set already. The global start method can only be set once. If you need to change the start method from the system default, you must proactively set the global start method before calling functions or methods, or creating these objects.

`Process` y excepciones

class multiprocessing.Process(*group=None*, *target=None*, *name=None*, *args=()*, *kwargs={}*, ***, *daemon=None*)

Los objetos de proceso representan la actividad que se ejecuta en un proceso separado. La clase **`Process`** tiene equivalentes para todos los métodos **`threading.Thread`**.

The constructor should always be called with keyword arguments. *group* should always be `None`; it exists solely for compatibility with **`threading.Thread`**. *target* is the callable object to be invoked by the **`run()`** method. It defaults to `None`, meaning nothing is called. *name* is the process name (see **`name`** for more details). *args* is the argument tuple for the target invocation. *kwargs* is a dictionary of keyword arguments for the target invocation. If provided, the keyword-only *daemon* argument sets the process **`daemon`** flag to `True` or `False`. If `None` (the default), this flag will be inherited from the creating process.

De forma predeterminada, no se pasan argumentos a *target*. El argumento *args*, que por defecto es `()`, se puede usar para especificar una lista o tupla de los argumentos para pasar a *target*.

If a subclass overrides the constructor, it must make sure it invokes the base class constructor (`super().__init__()`) before doing anything else to the process.

Nota

In general, all arguments to **`Process`** must be picklable. This is frequently observed when trying to create a **`Process`** or use a **`concurrent.futures.ProcessPoolExecutor`** from a REPL with a locally defined *target* function.

Passing a callable object defined in the current REPL session causes the child process to die via an uncaught **`AttributeError`** exception when starting as *target* must have been defined within an importable module in order to be loaded during unpickling.

Example of this uncatchable error from the child:

```
>>> import multiprocessing as mp
>>> def knigit():
...     print("Ni!")
...
>>> process = mp.Process(target=knigit)
>>> process.start()
>>> Traceback (most recent call last):
  File ".../multiprocessing/spawn.py", line ..., i
  File ".../multiprocessing/spawn.py", line ..., i
AttributeError: module '__main__' has no attribute
>>> process
<SpawnProcess name='SpawnProcess-1' pid=379473 par
```

See Los métodos de inicio spawn y forkserver. While this restriction is not true if using the `"fork"` start method, as of Python `3.14` that is no longer the default on any platform. See Contextos y métodos de inicio. See also gh-132898 [https://github.com/python/cpython/issues/132898].

Distinto en la versión 3.3: Added the *daemon* parameter.

run()

Método que representa la actividad del proceso.

Puede anular este método en una subclase. El método estándar **`run()`** invoca el objeto invocable pasado al constructor del objeto como argumento objetivo, si lo hay, con argumentos posicionares y de palabras clave tomados de los argumentos *args* y *kwargs*, respectivamente.

El uso de una lista o tupla como argumento *args* pasado a **`Process`** logra el mismo efecto.

Ejemplo:

```
>>> from multiprocessing import Process
>>> p = Process(target=print, args=[1])
>>> p.run()
1
```

```
>>> p = Process(target=print, args=(1,))
>>> p.run()
1
```

start()

Comienza la actividad del proceso.

Esto debe llamarse como máximo una vez por objeto de proceso. Organiza la invocación del método `run()` del objeto en un proceso separado.

join([*timeout*])

Si el argumento opcional *timeout* es `None` (el valor predeterminado), el método se bloquea hasta que el proceso cuyo método `join()` se llama termina. Si *timeout* es un número positivo, bloquea como máximo *timeout* segundos. Tenga en cuenta que el método retorna `None` si su proceso finaliza o si el método agota el tiempo de espera. Verifique el proceso `exitcode` para determinar si terminó.

Un proceso puede unirse muchas veces.

Un proceso no puede unirse a sí mismo porque esto provocaría un punto muerto. Es un error intentar unirse a un proceso antes de que se haya iniciado.

name

El nombre del proceso. El nombre es una cadena utilizada solo con fines de identificación. No tiene semántica. Múltiples procesos pueden tener el mismo nombre.

El nombre inicial es establecido por el constructor. Si no se proporciona un nombre explícito al constructor, se forma un nombre usando *“Process-N:sub:`1`:N:sub:`2`:…:N:sub:`k`”* se construye, donde cada *N:sub:`k`* es el *N-enésimo* hijo del parental.

is_alive()

Retorna si el proceso está vivo.

Aproximadamente, un objeto de proceso está vivo desde el momento en que el método `start()` retorna hasta que finaliza el proceso hijo.

daemon

La bandera del proceso *daemon* , es un valor booleano. Esto debe establecerse antes de que `start()` sea llamado.

El valor inicial se hereda del proceso de creación.

Cuando un proceso sale, intenta terminar todos sus procesos demoníacos (*daemonic*) hijos.

Tenga en cuenta que un proceso demoníaco (*daemonic*) no puede crear procesos hijos. De lo contrario, un proceso demoníaco (*daemonic*) dejaría a sus hijos huérfanos si se termina cuando finaliza su proceso parental. Además, estos **no** son *daemons* o servicios de Unix, son procesos normales qué finalizarán (y no se unirán) si los procesos no demoníacos han salido.

Además de API `threading.Thread`, los objetos `Process` también admiten los siguientes atributos y métodos:

pid

Retorna el ID del proceso. Antes de que se genere el proceso, esto será `None`.

exitcode

El código de salida del proceso secundario. Será `None` si el proceso aún no ha finalizado.

Si el método `run()` del proceso secundario se retornó normalmente, el código de salida será 0. Si terminó a través de `sys.exit()` con un argumento entero *N*, el código de salida será *N*.

Si el proceso secundario finalizó debido a una excepción no capturada dentro de `run()`, el código de salida será 1. Si fue terminado por la señal *N*, el código de salida será el valor negativo *-N*.

authkey

La clave de autenticación del proceso (una cadena de bytes).

When **`multiprocessing`** is initialized the main process is assigned a random string using `os.urandom()`.

Cuando se crea un objeto `Process`, este heredará la clave de autenticación de su proceso parental, aunque esto puede cambiarse configurando `authkey` a otra cadena de bytes.

Consulte Llaves de autentificación.

sentinel

Un identificador numérico de un objeto del sistema que estará «listo» cuando finalice el proceso.

You can use this value if you want to wait on several events at once using `multiprocessing.connection.wait()`. Otherwise calling `join()` is simpler.

On Windows, this is an OS handle usable with the `WaitForSingleObject` and `WaitForMultipleObjects` family of API calls. On POSIX, this is a file descriptor usable with primitives from the `select` module.

Added in version 3.3.

interrupt()

Terminate the process. Works on POSIX using the `SIGINT` signal. Behavior on Windows is undefined.

By default, this terminates the child process by raising `KeyboardInterrupt`. This behavior can be altered by setting the respective signal handler in the child process `signal.signal()` for `SIGINT`.

Note: if the child process catches and discards `KeyboardInterrupt`, the process will not be terminated.

Note: the default behavior will also set `exitcode` to `1` as if an uncaught exception was raised in the child process. To have a different `exitcode` you may simply catch `KeyboardInterrupt` and call `exit(your_code)`.

Added in version 3.14.

terminate()

Terminate the process. On POSIX this is done using the `SIGTERM` signal; on Windows `TerminateProcess()` is used. Note that exit handlers and finally clauses, etc., will not be executed.

Tenga en cuenta que los procesos descendientes del proceso *no* finalizarán – simplemente quedarán huérfanos.

Advertencia

Si este método se usa cuando el proceso asociado está usando una tubería (*pipe*) o una cola(*queue*), entonces la tubería o la cola pueden corromperse y pueden quedar inutilizables por otro proceso. Del mismo modo, si el proceso ha adquirido un *lock* o un semáforo, etc., su finalización puede provocar el bloqueo de otros procesos.

kill()

Same as `terminate()` but using the `SIGKILL` signal on POSIX.

Added in version 3.7.

close()

El cierre del objeto `Process`, libera todos los recursos asociados a él. `ValueError` se lanza si el proceso subyacente aún se está ejecutando. Una vez `close()` se retorna con éxito, la mayoría de los otros métodos y atributos del objeto `Process` lanzará `ValueError`.

Added in version 3.7.

Tenga en cuenta que los métodos `start()`, `join()`, `is_alive()`, `terminate()` y `exitcode` deberían solo ser llamados por el proceso que creó el objeto del proceso.

Ejemplo de uso de algunos métodos de la `Process`:

```
>>> import multiprocessing, time, signal
>>> mp_context = multiprocessing.get_context('spawn
>>> p = mp_context.Process(target=time.sleep, args=
>>> print(p, p.is_alive())
<...Process ... initial> False
>>> p.start()
>>> print(p, p.is_alive())
<...Process ... started> True
>>> p.terminate()
>>> time.sleep(0.1)
>>> print(p, p.is_alive())
<...Process ... stopped exitcode=-SIGTERM> False
>>> p.exitcode == -signal.SIGTERM
True
```

exception multiprocessing.ProcessError

The base class of all `multiprocessing` exceptions.

exception multiprocessing.BufferTooShort

Exception raised by `Connection.recv_bytes_into()` when the supplied buffer object is too small for the message read.

Si `e` es una instancia de `BufferTooShort` entonces `e.args[0]` dará el mensaje como una cadena de *bytes*.

exception multiprocessing.AuthenticationError

Lanzada cuando hay un error de autenticación.

exception multiprocessing.TimeoutError

Lanzada por métodos con un tiempo de espera (*timeout*) cuando este expira.

Tuberías (*Pipes*) y Colas (*Queues*)

Cuando se usan múltiples procesos, uno generalmente usa el paso de mensajes para la comunicación entre procesos y evita tener que usar primitivas de sincronización como *locks*.

Para pasar mensajes se puede usar `Pipe()` (para una conexión entre dos procesos) o una cola (*queue*)(que permite múltiples productores y consumidores).

Los tipos `Queue`, `SimpleQueue` y `JoinableQueue` son colas multi-productor, multi-consumidor FIFO (primero en entrar, primero en salir) modeladas en `queue.Queue` en la biblioteca estándar. Se diferencian en que `Queue` carece de `task_done()` y `join()` métodos introducidos en Python 2.5 `queue.Queue` class.

Si usa `JoinableQueue`, entonces **debe** llamar a `JoinableQueue.task_done()` para cada tarea eliminada de la cola (*queue*) o de lo contrario el semáforo utilizado para contar el número de tareas sin terminar puede eventualmente desbordarse, lanzando un excepción.

One difference from other Python queue implementations, is that `multiprocessing` queues serializes all objects that are put into them using `pickle`. The object returned by the get method is a recreated object that does not share memory with the original object.

Tenga en cuenta que también se puede crear una cola compartida mediante el uso de un objeto de administrador – consulte Administradores (Managers).

Nota

`multiprocessing` uses the usual `queue.Empty` and `queue.Full` exceptions to signal a timeout. They are not available in the `multiprocessing` namespace so you need to import them from `queue`.

Nota

Cuando un objeto se coloca en una cola (*queue*), el objeto se serializa (*pickled*) y luego un subproceso de fondo vacía los datos serializados a una tubería (*pipe*) subyacente. Esto tiene algunas consecuencias que son un poco sorprendentes, pero no deberían causar dificultades prácticas: si realmente causan molestias, se puede usar una cola creada con un manager.

1. Después de poner un objeto en una cola vacía, puede haber un retraso infinitesimal antes de que el método de la cola `empty()` retorne `False` y `get_nowait()` puede retornar sin lanzar `queue.Empty`.
2. Si varios procesos están poniendo en cola objetos, es posible que los objetos se reciban en el otro extremo fuera de orden. Sin embargo, los objetos en cola por el mismo proceso siempre estarán en el orden esperado entre sí.

Advertencia

Si se termina un proceso usando `Process.terminate()` o `os.kill()` mientras intenta usar una clase `Queue`, es probable que los datos de la cola(*queue*) se corrompan. Esto puede hacer que cualquier otro proceso obtenga una excepción cuando intente usar la cola más adelante.

Advertencia

Como se mencionó anteriormente, si un proceso hijo ha puesto elementos en una cola (*queue*) (y no ha utilizado `JoinableQueue.cancel_join_thread`), entonces ese proceso no terminará hasta que todos los elementos almacenados en búfer hayan sido vaciados a la tubería (*pipe*).

Esto significa que si intenta unirse a ese proceso, puede obtener un punto muerto a menos que esté seguro de que todos los elementos que se han puesto en la cola (*queue*) se han consumido. Del mismo modo, si el proceso hijo no es *daemonic*, el proceso parental puede bloquearse en la salida cuando intenta unir todos sus elementos hijos no *daemonic*.

Tenga en cuenta que una cola (*queue*) creada con un administrador no tiene este problema. Consulte Pautas de programación.

Para ver un ejemplo del uso de colas para la comunicación entre procesos, consulte Ejemplos.

multiprocessing.Pipe(*duplex=True*)

Retorna un par de objetos ``*(conn1, conn2)*`` de la **`Connection`** que representan los extremos de una tubería (*pipe*).

Si *duplex* es `True` (el valor predeterminado), entonces la tubería (*pipe*) es bidireccional. Si *duplex* es `False`, entonces la tubería es unidireccional: `conn1` solo se puede usar para recibir mensajes y `conn2` solo se puede usar para enviar mensajes.

The **`send()`** method serializes the object using **`pickle`** and the **`recv()`** re-creates the object.

class multiprocessing.Queue(*[maxsize]*)

Retorna un proceso de cola (*queue*) compartida implementado utilizando una tubería (*pipe*) y algunos candados/semáforos (*locks/semaphores*). Cuando un proceso pone por primera vez un elemento en la cola, se inicia un hilo alimentador que transfiere objetos desde un búfer a la tubería.

Instantiating this class may set the global start method. See Global start method for more details.

Las excepciones habituales **`queue.Empty`** y **`queue.Full`** del módulo de la biblioteca estándar **`queue`** se generan para indicar tiempos de espera.

`Queue` implements all the methods of **`queue.Queue`** except for **`task_done()`**, **`join()`**, and **`shutdown()`**.

qsize()

> Retorna el tamaño aproximado de la cola (*queue*). Debido a la semántica multiproceso/multiprocesamiento, este número no es confiable.
>
> Note that this may raise **`NotImplementedError`** on platforms like macOS where `sem_getvalue()` is not implemented.

empty()

> Retorna `True` si la cola (*queue*) está vacía, de lo contrario retorna `False`. Debido a la semántica multiproceso/multiprocesamiento, esto no es confiable.
>
> May raise an **`OSError`** on closed queues. (not guaranteed)

full()

> Retorna `True` si la cola (*queue*) está llena, de lo contrario retorna `False`. Debido a la semántica multiproceso/multiprocesamiento, esto no es confiable.

put(*obj*[, *block*[, *timeout*]])

> Pone *obj* en la cola. Si el argumento opcional *block* es `True` (el valor predeterminado) y *timeout* es `None` (el

valor predeterminado), se bloquea si es necesario hasta que haya un espacio disponible. Si *timeout* es un número positivo, bloquea a lo sumo *timeout* segundos y genera la excepción **`queue.Full`** si no hay espacio libre disponible en ese tiempo. De lo contrario (*block* es `False`), y coloca un elemento en la cola si hay un espacio libre disponible de inmediato, de lo contrario, genera la excepción **`queue.Full`** (*timeout* se ignora en ese caso).

Distinto en la versión 3.8: Si la cola (*queue*) está cerrada, **`ValueError`** se lanza en lugar de **`AssertionError`**.

put_nowait(*obj*)

Equivalente a `put(obj, False)`.

get([*block*[, *timeout*]])

Elimina y retorna un artículo de la cola (*queue*). Si un argumento opcional *block* es `True` (el valor predeterminado) y el *timeout* es `None` (el valor predeterminado), es bloqueado si es necesario hasta que un elemento esté disponible. Si el *timeout* es un número positivo, bloquea a lo sumo segundos y genera la excepción **`queue.Empty`** si no había ningún elemento disponible dentro de ese tiempo. De lo contrario (el bloque es `False`), retorna un elemento si hay uno disponible de inmediato, de lo contrario, levante la excepción **`queue.Empty`** (*timeout* se ignora en ese caso).

Distinto en la versión 3.8: Si la cola está cerrada, se lanza **`ValueError`** en lugar de **`OSError`**.

get_nowait()

Equivalente a `get(False)`.

La **`multiprocessing.Queue`** tiene algunos métodos adicionales que no se encuentran en **`queue.Queue`**. Estos métodos suelen ser innecesarios para la mayoría de los códigos:

close()

Cierra la cola: libera recursos internos.

A queue must not be used anymore after it is closed. For example, `get()`, `put()` and `empty()` methods must no longer be called.

The background thread will quit once it has flushed all buffered data to the pipe. This is called automatically when the queue is garbage collected.

join_thread()

Unifica al hilo de fondo. Esto solo se puede usar después de que se ha llamado a `close()`. Esto se bloquea hasta que salga el hilo de fondo, asegurando que todos los datos en el búfer se hayan vaciado a la tubería (*pipe*).

Por defecto, si un proceso no es el creador de la cola (*queue*), al salir intentará unirse al hilo de fondo de la cola. El proceso puede llamar a `cancel_join_thread()` para hacer que el `join_thread()` no haga nada.

cancel_join_thread()

Evita que `join_thread()` bloquee. En particular, esto evita que el subproceso en segundo plano se una automáticamente cuando finaliza el proceso; consulte **`join_thread()`**.

Un mejor nombre para este método podría ser `allow_exit_without_flush()`. Es probable que provoque la pérdida de datos encolados, y es casi seguro que no necesitará usarlo. Realmente solo está allí si necesita que el proceso actual salga inmediatamente sin esperar a vaciar los datos en en la tubería (*pipe*) subyacente, y no le importan los datos perdidos.

Nota

La funcionalidad de esta clase requiere una implementación de semáforo compartido en funcionamiento en el sistema operativo que es huésped (*host*). Sin uno, la funcionalidad en esta clase se deshabilitará, y los intentos de instanciar a **`Queue`** resultarán en **`ImportError`**. Consulte bpo-3770 [https://bugs.python.org/issue?@action=redirect&bpo=3770] para información adicional. Lo mismo es válido para cualquiera de los tipos de cola especializados que se enumeran a continuación.

class multiprocessing.SimpleQueue

Es un tipo simplificado **`Queue`**, muy similar a un *lock* de **`Pipe`**.

Instantiating this class may set the global start method. See Global start method for more details.

close()

Cierra la cola: libera recursos internos.

Una cola no se debe usar más después de ser cerrada. Por ejemplo los métodos **`get()`**, **`put()`** y **`empty()`** no deben ser llamados.

Added in version 3.9.

empty()

Retorna `True` si la cola (*queue*) está vacía, de otra manera retorna `False`.

Always raises an **`OSError`** if the SimpleQueue is closed.

get()

Eliminar y retornar un artículo de la cola (*queue*).

put(*item*)

Pone *item* en la cola.

class multiprocessing.JoinableQueue([*maxsize*])

`JoinableQueue`, una subclase `Queue`, es una cola (*queue*) que además tiene los métodos `task_done()` y `join()`.

Instantiating this class may set the global start method. See Global start method for more details.

task_done()

Indica que una tarea anteriormente en cola (*queue*) está completa. Usado por los consumidores de la cola. Por cada `get()` utilizado para recuperar una tarea, una llamada posterior a `task_done()` le dice a la cola que el procesamiento de la tarea se ha completado.

Si un `join()` se está bloqueando actualmente, se reanudará cuando se hayan procesado todos los elementos (lo que significa que `task_done()` es llamado para cada elemento que había sido puesto en cola (*queue*) por `put()`).

Lanza un `ValueError` si es llamado más veces que elementos hay en una cola.

join()

Se bloquea hasta que todos los elementos en una cola han sido recibidos y procesados.

El recuento de tareas no finalizadas aumenta cada vez que se agrega un elemento a la cola. El recuento disminuye cada vez que un consumidor llama a `task_done()` para indicar que el artículo se recuperó y todo el trabajo en él está completo. Cuando el recuento de tareas inacabadas cae a cero, `join()` se desbloquea.

Miscelánea

multiprocessing.active_children()

Retorna una lista con todos los hijos del proceso actual.

Llamar a esto tiene el efecto secundario de «unir» (*joining*) cualquier proceso que ya haya finalizado.

multiprocessing.cpu_count()

Retorna el número de CPU en el sistema.

This number is not equivalent to the number of CPUs the current process can use. The number of usable CPUs can be obtained with `os.process_cpu_count()` (or `len(os.sched_getaffinity(0))`).

Cuando no se puede determinar el número de CPUs, se lanza un `NotImplementedError`.

Ver también

`os.cpu_count()` `os.process_cpu_count()`

Distinto en la versión 3.13: The return value can also be overridden using the `-X cpu_count` flag or `PYTHON_CPU_COUNT` as this is merely a wrapper around the `os` cpu count APIs.

multiprocessing.current_process()

Retorna el objeto de la `Process` correspondiente al proceso actual.

Un análogo de la `threading.current_thread()`.

multiprocessing.parent_process()

Retorna el objeto de la `Process` correspondiente al proceso parental de `current_process()`. Para el proceso principal, `parent_process` será `None`.

Added in version 3.8.

multiprocessing.freeze_support()

Add support for when a program which uses **`multiprocessing`** has been frozen to produce an executable. (Has been tested with **py2exe**, **PyInstaller** and **cx_Freeze**.)

Es necesario llamar a esta función inmediatamente después de la línea principal del módulo `if __name__ == '__main__'`. Por ejemplo:

```
from multiprocessing import Process, freeze_support

def f():
    print('hello world!')

if __name__ == '__main__':
    freeze_support()
    Process(target=f).start()
```

Si se omite la línea `freeze_support()` entonces se intenta comenzar el ejecutable congelado que lanzará `RuntimeError`.

Calling `freeze_support()` has no effect when the start method is not *spawn*. In addition, if the module is being run normally by the Python interpreter (the program has not been frozen), then `freeze_support()` has no effect.

multiprocessing.get_all_start_methods()

Returns a list of the supported start methods, the first of which is the default. The possible start methods are `'fork'`, `'spawn'` and `'forkserver'`. Not all platforms support all methods. See Contextos y métodos de inicio.

Added in version 3.4.

multiprocessing.get_context(*method=None*)

Return a context object which has the same attributes as the **`multiprocessing`** module.

If *method* is `None` then the default context is returned. Note that if the global start method has not been set, this will set it to the system default See Global start method for more details. Otherwise *method* should be `'fork'`, `'spawn'`, `'forkserver'`. **`ValueError`** is raised if the specified start method is not available. See Contextos y métodos de inicio.

Added in version 3.4.

multiprocessing.get_start_method(*allow_none=False*)

Retorna el nombre del método de inicio que es utilizado para iniciar procesos.

If the global start method is not set and *allow_none* is `False`, the global start method is set to the default, and its name is returned. See Global start method for more details.

The return value can be `'fork'`, `'spawn'`, `'forkserver'` or `None`. See Contextos y métodos de inicio.

Added in version 3.4.

Distinto en la versión 3.8: En macOS, el método de inicio *spawn* ahora es el predeterminado. El método de inicio *fork* debe considerarse inseguro ya que puede provocar bloqueos del subproceso. Consulte bpo-33725 [https://bugs.python.org/issue?@action=redirect&bpo=33725].

multiprocessing.set_executable(*executable*)

Set the path of the Python interpreter to use when starting a child process. (By default **`sys.executable`** is used). Embedders will probably need to do something like

```
set_executable(os.path.join(sys.exec_prefix, 'pytho
```

antes ellos pueden crear procesos hijos.

Distinto en la versión 3.4: Now supported on POSIX when the `'spawn'` start method is used.

Distinto en la versión 3.11: Acepta un path-like object.

multiprocessing.set_forkserver_preload(*module_names*)

Set a list of module names for the forkserver main process to attempt to import so that their already imported state is inherited by forked processes. Any **`ImportError`** when doing so is silently ignored. This can be used as a performance enhancement to avoid repeated work in every process.

For this to work, it must be called before the forkserver process has been launched (before creating a **`Pool`** or starting a **`Process`**).

Only meaningful when using the `'forkserver'` start method. See Contextos y métodos de inicio.

Added in version 3.4.

multiprocessing.set_start_method(*method*, *force=False*)

Set the method which should be used to start child processes. The *method* argument can be `'fork'`, `'spawn'` or `'forkserver'`. Raises **`RuntimeError`** if the start method has already been set and *force* is not `True`. If *method* is `None` and *force* is `True` then the start method is set to `None`. If *method* is `None` and *force* is `False` then the context is set to the default context.

Tenga en cuenta que esto debería llamarse como máximo una vez, y debería protegerse dentro de la cláusula `if __name__ == '__main__'` del módulo principal.

See Contextos y métodos de inicio.

Added in version 3.4.

Nota

`multiprocessing` contains no analogues of `threading.active_count()`, `threading.enumerate()`, `threading.settrace()`, `threading.setprofile()`, `threading.Timer`, or `threading.local`.

Objetos de conexión *Connection Objects*

Los objetos de conexión permiten el envío y la recepción de objetos serializables (*pickable*) o cadenas de caracteres seleccionables. Pueden considerarse como sockets conectados orientados a mensajes.

Los objetos de conexión usualmente son creados usando `Pipe` – ver también Oyentes y clientes (Listeners and Clients).

class multiprocessing.connection.Connection

send(*obj*)

Envía un objeto al otro extremo de la conexión que debe leerse usando `recv()`.

El objeto debe ser serializable (*pickable*). Los serializados (*pickable*) muy grandes (aproximadamente 32 *MiB*+ , aunque depende del sistema operativo) pueden generar una excepción `ValueError`.

recv()

Retorna un objeto enviado desde el otro extremo de la conexión usando `send()`. Se bloquea hasta que haya algo para recibir. Se lanza `EOFError` si no queda nada por recibir y el otro extremo está cerrado.

fileno()

Retorna el descriptor de archivo o identificador utilizado por la conexión.

close()

Cierra la conexión.

Esto se llama automáticamente cuando la conexión es basura recolectada.

poll([*timeout*])

Retorna si hay datos disponibles para leer.

Si no se especifica *timeout*, se retornará de inmediato. Si *timeout* es un número, esto especifica el tiempo máximo en segundos para bloquear. Si *timeout* es `None`, se usa un tiempo de espera infinito.

Tenga en cuenta que se pueden sondear varios objetos de conexión a la vez utilizando **`multiprocessing.connection.wait()`**.

send_bytes(*buf*[, *offset*[, *size*]])

Envía datos de *bytes* desde a bytes-like object como un mensaje completo.

If *offset* is given then data is read from that position in *buf*. If *size* is given then that many bytes will be read from *buf*. Very large buffers (approximately 32 MiB+, though it depends on the OS) may raise a **`ValueError`** exception

recv_bytes([*maxlength*])

Retorna un mensaje completo de datos de *bytes* enviados desde el otro extremo de la conexión como una cadena de caracteres. Se bloquea hasta que haya algo para recibir. Aumenta **`EOFError`** si no queda nada por recibir y el otro extremo se ha cerrado.

Si se especifica *maxlength* y el mensaje es más largo que *maxlength*, entonces se lanza un **`OSError`** y la conexión ya no será legible.

Distinto en la versión 3.3: Esta función solía lanzar un **`IOError`**, que ahora es un alias de **`OSError`**.

recv_bytes_into(*buf*[, *offset*])

Read into *buf* a complete message of byte data sent from the other end of the connection and return the number of bytes in the message. Blocks until there is something to receive. Raises **`EOFError`** if there is nothing left to receive and the other end was closed.

buf must be a writable bytes-like object. If *offset* is given then the message will be written into the buffer from that position. Offset must be a non-negative integer less than the length of *buf* (in bytes).

Si el búfer es demasiado corto, se genera una excepción **`BufferTooShort`** y el mensaje completo está disponible como `e.args[0]` donde `e` es la instancia de excepción.

Distinto en la versión 3.3: Los objetos de conexión ahora pueden transferirse entre procesos usando **`Connection.send()`** y **`Connection.recv()`**.

Connection objects also now support the context management protocol – see Tipos gestores de contexto. **`__enter__()`** returns the connection object, and **`__exit__()`** calls **`close()`**.

Por ejemplo:

```
>>> from multiprocessing import Pipe
>>> a, b = Pipe()
>>> a.send([1, 'hello', None])
>>> b.recv()
[1, 'hello', None]
>>> b.send_bytes(b'thank you')
>>> a.recv_bytes()
b'thank you'
>>> import array
>>> arr1 = array.array('i', range(5))
>>> arr2 = array.array('i', [0] * 10)
>>> a.send_bytes(arr1)
>>> count = b.recv_bytes_into(arr2)
>>> assert count == len(arr1) * arr1.itemsize
>>> arr2
array('i', [0, 1, 2, 3, 4, 0, 0, 0, 0, 0])
```

Advertencia

El método `Connection.recv()` desempaqueta automáticamente los datos que recibe, lo que puede ser un riesgo de seguridad a menos que pueda confiar en el proceso que envió el mensaje.

Por lo tanto, a menos que el objeto de conexión se haya producido usando `Pipe()` solo debe usar los métodos `recv()` y `send()` después de realizar algún tipo de autenticación. Consulte Llaves de autentificación.

Advertencia

Si se mata un proceso mientras intenta leer o escribir en una tubería (*pipe*), entonces es probable que los datos en la tubería se corrompan, porque puede ser imposible estar seguro de dónde se encuentran los límites del mensaje.

Primitivas de sincronización (*Synchronization primitives*)

En general, las primitivas de sincronización no son tan necesarias en un programa multiproceso como en un programa *multihilos* (*multithreaded*). Consulte la documentación para `threading`.

Tenga en cuenta que también se pueden crear primitivas de sincronización utilizando un objeto administrador – consulte Administradores (Managers).

class multiprocessing.Barrier(*parties*[, *action*[, *timeout*]])

Un objeto de barrera: un clon de `threading.Barrier`.

Instantiating this class may set the global start method. See Global start method for more details.

Added in version 3.3.

class multiprocessing.BoundedSemaphore([*value*])

Un objeto semáforo (*semaphore object*) acotado: un análogo cercano de la `threading.BoundedSemaphore`.

Instantiating this class may set the global start method. See Global start method for more details.

Existe una diferencia solitaria de su análogo cercano: el primer argumento de su método `acquire` es nombrado *block*, es consistente con `Lock.acquire()`.

locked()

Return a boolean indicating whether this object is locked right now.

Added in version 3.14.

Nota

En macOS, esto no se puede distinguir de `Semaphore` porque `sem_getvalue()` no está implementado en esa plataforma.

class multiprocessing.Condition([*lock*])

Una variable de condición: un alias para la `threading.Condition`.

If *lock* is specified then it should be a `Lock` or `RLock` object from **`multiprocessing`**.

Instantiating this class may set the global start method. See Global start method for more details.

Distinto en la versión 3.3: El método `wait_for()` fue a añadido.

class multiprocessing.Event

Un clon de `threading.Event`.

Instantiating this class may set the global start method. See Global start method for more details.

class multiprocessing.Lock

Un objeto candado no recursivo: un análogo cercano de `threading.Lock` . Una vez que un proceso o subproceso ha adquirido un bloqueo, los intentos posteriores de adquirirlo de cualquier proceso o subproceso se bloquearán hasta que se libere; cualquier proceso o hilo puede liberarlo. Los conceptos y comportamientos de **`threading.Lock`** como se aplica a los subprocesos se replican aquí en `multiprocessing.Lock` como se aplica a los procesos o subprocesos, excepto como se indica.

Tenga en cuenta que `Lock` es en realidad una función de fábrica que retorna una instancia de `multiprocessing.synchronize.Lock` inicializada con un contexto predeterminado.

Instantiating this class may set the global start method. See Global start method for more details.

La `Lock` soporta el protocolo context manager y, por lo tanto, se puede usar en la declaración `with`.

acquire(*block=True*, *timeout=None*)

Adquiriendo un candado (*lock*), bloqueante o no bloqueante.

Con el argumento *block* establecido en `True` (el valor predeterminado), la llamada al método se bloqueará hasta que el bloqueo esté en un estado desbloqueado, luego configúrelo como bloqueado y retorne `True`. Tenga en cuenta que el nombre de este primer argumento difiere del que aparece en **`threading.Lock.acquire()`**.

Con el argumento *block* establecido en `False`, la llamada al método no se bloquea. Si el bloqueo está actualmente en un estado bloqueado, retorna `False`; de lo contrario, configure el bloqueo en un estado bloqueado y retorne `True`.

Cuando se invoca con un valor positivo de punto flotante para *timeout*, bloquea como máximo el número de segundos especificado por *timeout* siempre que no se pueda obtener el bloqueo. Las invocaciones con un valor negativo para *timeout* de cero. Las invocaciones con un valor *timeout* de `None` (el valor predeterminado) establecen el período de tiempo de espera en infinito. Tenga en cuenta que el tratamiento de valores negativos o `None` para *timeout* difiere del comportamiento implementado en **`threading.Lock.acquire()`**. El argumento *timeout* no tiene implicaciones prácticas si el argumento *block* se establece en `False` y, por lo tanto, se ignora. Retorna `True` si se ha adquirido el candado o `False` si ha transcurrido el tiempo de espera.

release()

Suelta un candado. Esto se puede llamar desde cualquier proceso o subproceso, no solo desde el proceso o subproceso que originalmente adquirió el candado.

El comportamiento es el mismo que en **`threading.Lock.release()`** excepto que cuando se invoca en un bloqueo desbloqueado, se genera a **`ValueError`**.

locked()

Return a boolean indicating whether this object is locked right now.

Added in version 3.14.

class multiprocessing.RLock

Un objeto de candado recursivo: un análogo cercano a **`threading.RLock`**. El proceso o el hilo que lo adquirió debe liberar un candado recursivo. Una vez que un proceso o subproceso ha adquirido un candado recursivo, el mismo proceso o subproceso puede volver a adquirirlo sin bloquearlo; ese proceso o hilo debe liberarlo una vez por cada vez que se haya adquirido.

Tenga en cuenta que **`RLock`** es en realidad una función de fábrica que retorna una instancia de `multiprocessing.synchronize.RLock` inicializada con un contexto predeterminado.

Instantiating this class may set the global start method. See Global start method for more details.

La **`RLock`** admite el protocolo context manager y, por lo tanto, puede usarse en **`with`**.

acquire(*block=True*, *timeout=None*)

Adquiriendo un candado (*lock*), bloqueante o no bloqueante.

Cuando se invoca con el argumento *block* establecido en `True` , bloquea hasta que el candado esté en un estado desbloqueado (que no sea propiedad de ningún proceso o subproceso) a menos que el candado o el subproceso actual ya sea de su propiedad. El proceso o subproceso actual se apropia del candado (si aún no lo tiene) y el nivel de recursión dentro de este aumenta en uno, lo que da como resultado un valor de retorno de `True`. Tenga en cuenta que hay varias diferencias en el comportamiento de este primer argumento en comparación con la implementación de `threading.RLock.acquire()`, comenzando con el nombre del argumento en sí.

Cuando se invoca con el argumento *block* establecido en `False`, no bloquea. Si el candado ya ha sido adquirido (y por lo tanto es propiedad) de otro proceso o subproceso, el proceso o subproceso actual no se apropia y el nivel de recursión dentro del candado no cambia, lo que resulta en un valor de retorno de `False`. Si el candado está en un estado desbloqueado, el proceso o subproceso actual toma posesión y el nivel de recurrencia se incrementa, lo que resulta en un valor de retorno de `True`.

El uso y los comportamientos del argumento *timeout* son los mismos que en `Lock.acquire()`. Tenga en cuenta que algunos de estos comportamientos de *timeout* difieren de los comportamientos implementados en `threading.RLock.acquire()`.

release()

Libera un candado, disminuyendo el nivel de recursión. Si después del decremento el nivel de recursión es cero, restablece el candado a desbloqueado (que no sea propiedad de ningún proceso o subproceso) y si se bloquean otros procesos o subprocesos esperando que el candado se desbloquee, permite que continúe exactamente uno de ellos. Si después del decremento el nivel de recursión sigue siendo distinto de cero, el candado permanece bloqueado y pertenece al proceso de llamada o subproceso.

Solo llame a este método cuando el proceso o subproceso de llamada sea el propietario del candado. Se lanza un `AssertionError` si se llama a este método mediante un proceso o subproceso que no sea el propietario o si el candado está en un estado desbloqueado (sin propietario). Tenga en cuenta que el tipo de excepción planteada en esta situación difiere del comportamiento implementado en `threading.RLock.release()`.

locked()

Return a boolean indicating whether this object is locked right now.

Added in version 3.14.

class multiprocessing.Semaphore([*value*])

Un objeto semáforo: un análogo cercano de `threading.Semaphore`.

Instantiating this class may set the global start method. See Global start method for more details.

Existe una diferencia solitaria de su análogo cercano: el primer argumento de su método `acquire` es nombrado *block*, es consistente con `Lock.acquire()`.

get_value()

Return the current value of semaphore.

Note that this may raise `NotImplementedError` on platforms like macOS where `sem_getvalue()` is not implemented.

locked()

Return a boolean indicating whether this object is locked right now.

Added in version 3.14.

Nota

En macOS, `sem_timedwait` no es compatible, por lo que llamar a `acquire()` con un tiempo de espera emulará el comportamiento de esa función utilizando un bucle inactivo.

Nota

Parte de la funcionalidad de este paquete requiere una implementación de semáforo compartido que funcione en el sistema operativo. Sin uno, el módulo **`multiprocessing.synchronize`** se desactivará, y los intentos de importarlo darán como resultado **`ImportError`**. Consulte bpo-3770 [https://bugs.python.org/issue?@action=redirect&bpo=3770] para información adicional.

Objetos compartidos `ctypes`

Es posible crear objetos compartidos utilizando memoria compartida que puede ser heredada por procesos secundarios.

multiprocessing.Value(*typecode_or_type*, **args*, *lock=True*)

Retorna un objeto **`ctypes`** asignado desde la memoria compartida. Por defecto, el valor de retorno es en realidad un contenedor sincronizado para el objeto. Se puede acceder al objeto en sí a través del atributo *value* de la **`Value`**.

typecode_or_type determina el tipo del objeto retornado: es un tipo *ctypes* o un código de tipo de un carácter del tipo utilizado por el módulo **`array`**. **args* se pasa al constructor para el tipo.

Si *lock* es `True` (el valor predeterminado), se crea un nuevo objeto de candado recursivo para sincronizar el acceso al valor. Si *lock* es un objeto **`Lock`** o **`RLock`**, se usará para sincronizar el acceso al valor. Si *lock* es `False`, entonces el acceso al objeto retornado no estará protegido automáticamente por un candado, por lo que no será necesariamente «proceso-seguro».

Operaciones como `+=` que implican una lectura y escritura no son atómicas. Entonces, si, por ejemplo, desea incrementar atómicamente un valor compartido, es insuficiente simplemente hacer:

```
counter.value += 1
```

Suponiendo que el candado asociado es recursivo (que es por defecto), puede hacer

```
with counter.get_lock():
    counter.value += 1
```

Véase que *lock* es un argumento de solo una palabra clave.

multiprocessing.Array(*typecode_or_type*, *size_or_initializer*, *, *lock=True*)

Retorna una matriz *ctypes* asignada desde la memoria compartida. Por defecto, el valor de retorno es en realidad un contenedor sincronizado para el arreglo.

typecode_or_type determines the type of the elements of the returned array: it is either a ctypes type or a one character typecode of the kind used by the **`array`** module with the exception of `'w'`, which is not supported. In addition, the `'c'` typecode is an alias for **`ctypes.c_char`**. If *size_or_initializer* is an integer, then it determines the length of the array, and the array will be initially zeroed. Otherwise, *size_or_initializer* is a sequence which is used to initialize the array and whose length determines the length of the array.

Si *lock* es `True` (el valor predeterminado), se crea un nuevo objeto de bloqueo para sincronizar el acceso al valor. Si *lock* es un objeto **`Lock`** o **`RLock`**, se usará para sincronizar el acceso al valor. Si *lock* es `False`, entonces el acceso al objeto retornado no estará protegido automáticamente por un candado, por lo que no será necesariamente «proceso seguro».

Véase que *lock* es un argumento de solo una palabra clave.

Tenga en cuenta que una matriz de **`ctypes.c_char`** tiene atributos *value* y *raw* que le permiten a uno usarlo para almacenar y recuperar cadenas de caracteres.

The `multiprocessing.sharedctypes` module

The **`multiprocessing.sharedctypes`** module provides functions for allocating **`ctypes`** objects from shared memory which can be inherited by child processes.

Nota

Aunque es posible almacenar un puntero en la memoria compartida, recuerde que esto se referirá a una ubicación en el espacio de direcciones de un proceso específico. Sin embargo, es muy probable que el puntero sea inválido en el contexto de un segundo proceso y tratar de desreferenciar el puntero del segundo proceso puede causar un bloqueo.

multiprocessing.sharedctypes.RawArray(*typecode_or_type*, *size_or_initializer*)

Retorna una matriz *ctypes* asignada desde la memoria compartida.

typecode_or_type determina el tipo de los elementos de la matriz retornada: es un tipo de tipo *ctypes* o un código de tipo de un carácter del tipo utilizado por el módulo **`array`**. Si *size_or_initializer* es un entero, entonces determina la longitud de la matriz, y la matriz se pondrá a cero inicialmente. De lo contrario, *size_or_initializer* es una secuencia que se usa para inicializar la matriz y cuya longitud determina la longitud del arreglo.

Tenga en cuenta que configurar y obtener un elemento es potencialmente no atómico – utiliza **`Array()`** en su lugar para asegurarse de que el acceso se sincronice automáticamente mediante un candado.

multiprocessing.sharedctypes.RawValue(*typecode_or_type*, **args*)

Retorna un objeto *ctypes* asignado desde la memoria compartida.

typecode_or_type determina el tipo del objeto retornado: es un tipo *ctypes* o un código de tipo de un carácter del tipo utilizado por el módulo **`array`**. **args* se pasa al constructor para el tipo.

Tenga en cuenta que configurar y obtener el valor es potencialmente no atómico – use **`Value()`** en su lugar para asegurarse de que el acceso se sincronice automáticamente mediante un candado.

Tenga en cuenta que una matriz de **`ctypes.c_char`** tiene atributos `value` y `raw` que le permiten a uno usarlo para almacenar y recuperar cadenas de caracteres – consulte la documentación para **`ctypes`**.

multiprocessing.sharedctypes.Array(*typecode_or_type*, *size_or_initializer*, *, *lock=True*, *ctx=None*)

Lo mismo que **`RawArray()`**, excepto que, dependiendo del valor de *lock*, se puede retornar un contenedor de sincronización seguro para el proceso en lugar de un arreglo de tipos crudos.

Si *lock* es `True` (el valor predeterminado), se crea un nuevo objeto candado para sincronizar el acceso al valor. Si *lock* es un objeto **`Lock`** o **`RLock`**, se utilizará para sincronizar el acceso al valor. Si *lock* es `False`, entonces el acceso al objeto retornado no estará protegido automáticamente por un candado, por lo que no será necesariamente «seguro para el proceso».

ctx is a context object, or `None` (use the current context). If `None`, calling this may set the global start method. See Global start method for more details.

Note that *lock* and *ctx* are keyword-only parameters.

multiprocessing.sharedctypes.Value(*typecode_or_type*, **args*, *lock=True*, *ctx=None*)

Lo mismo que **`RawValue()`** excepto que, dependiendo del valor de *lock*, se puede retornar una envoltura de sincronización segura para el proceso en lugar de un objeto *ctypes* sin procesar.

Si *lock* es `True` (el valor predeterminado), se crea un nuevo objeto candado para sincronizar el acceso al valor. Si *lock* es un objeto **`Lock`** o **`RLock`**, se utilizará para sincronizar el acceso al valor. Si *lock* es `False`, entonces el acceso al objeto retornado no estará protegido automáticamente por un candado, por lo que no será necesariamente «seguro para el proceso».

ctx is a context object, or `None` (use the current context). If `None`, calling this may set the global start method. See Global start method for more details.

Note that *lock* and *ctx* are keyword-only parameters.

multiprocessing.sharedctypes.copy(*obj*)

Retorna un objeto *ctypes* asignado de la memoria compartida, que es una copia del objeto *ctypes obj*.

multiprocessing.sharedctypes.synchronized(*obj*, *lock=None*, *ctx=None*)

Retorna un objeto contenedor seguro para un objeto *ctypes* que usa *lock* para sincronizar el acceso. Si *lock* es `None` (el valor predeterminado), se crea automáticamente un objeto **`multiprocessing.RLock`**.

ctx is a context object, or `None` (use the current context). If `None`, calling this may set the global start method. See Global start method for more details.

Un contenedor sincronizado tendrá dos métodos además de los del objeto que envuelve: **`get_obj()`** retorna el objeto envuelto y **`get_lock()`** retorna el objeto de bloqueo utilizado para la sincronización.

Tenga en cuenta que acceder al objeto *ctypes* a través del contenedor puede ser mucho más lento que acceder al objeto *ctypes* sin formato.

Distinto en la versión 3.5: Los objetos sincronizados admiten el protocolo: context manager.

La siguiente tabla compara la sintaxis para crear objetos *ctypes* compartidos desde la memoria compartida con la sintaxis *ctypes* normal. (En la tabla `MyStruct` hay alguna subclase de **`ctypes.Structure`**.)

ctypes	**sharedctypes usando *type***	**sharedctypes usando *typecode***
c_double(2.4)	*RawValue(c_double, 2.4)*	*RawValue('d', 2.4)*
MyStruct(4, 6)	*RawValue(MyStruct, 4, 6)*	
*(c_short * 7)()*	*RawArray(c_short, 7)*	*RawArray('h', 7)*
*(c_int * 3)(9, 2, 8)*	*RawArray(c_int, (9, 2, 8))*	*RawArray('i', (9, 2, 8))*

A continuación se muestra un ejemplo donde un número de objetos *ctypes* son modificados por un proceso hijo:

```
from multiprocessing import Process, Lock
from multiprocessing.sharedctypes import Value, Array
from ctypes import Structure, c_double

class Point(Structure):
    _fields_ = [('x', c_double), ('y', c_double)]

def modify(n, x, s, A):
    n.value **= 2
    x.value **= 2
    s.value = s.value.upper()
    for a in A:
        a.x **= 2
        a.y **= 2

if __name__ == '__main__':
    lock = Lock()

    n = Value('i', 7)
    x = Value(c_double, 1.0/3.0, lock=False)
    s = Array('c', b'hello world', lock=lock)
    A = Array(Point, [(1.875,-6.25), (-5.75,2.0), (2.375

    p = Process(target=modify, args=(n, x, s, A))
    p.start()
    p.join()

    print(n.value)
    print(x.value)
    print(s.value)
    print([(a.x, a.y) for a in A])
```

Los resultados impresos son

```
49
0.1111111111111111
HELLO WORLD
[(3.515625, 39.0625), (33.0625, 4.0), (5.640625, 90.25)
```

Administradores (*Managers*)

Los administradores (*managers*) proporcionan una forma de crear datos que se pueden compartir entre diferentes procesos, incluido el intercambio en una red entre procesos que se ejecutan en diferentes máquinas. Un objeto administrador controla un proceso de servidor que gestiona *shared objects* (*objetos compartidos*). Otros procesos pueden acceder a los objetos compartidos mediante el uso de servidores proxy.

multiprocessing.Manager()

> Retorna un objeto iniciado `SyncManager` que se puede usar para compartir objetos entre procesos. El objeto administrador retornado corresponde a un proceso hijo generado y tiene métodos que crearán objetos compartidos y retornarán los proxies correspondientes.

Los procesos del administrador se cerrarán tan pronto como se recolecte la basura o salga su proceso padre. Las clases de administrador se definen en el módulo `multiprocessing.managers`:

class multiprocessing.managers.BaseManager(*address=None, authkey=None, serializer='pickle', ctx=None, *, shutdown_timeout=1.0*)

Crear un objeto *BaseManager*.

Una vez creado, debe llamar a **`start()`** o `get_server().serve_forever()` para asegurarse de que el objeto de administrador se refiera a un proceso de administrador iniciado.

address es la dirección en la que el proceso del administrador escucha las nuevas conexiones. Si *address* es `None`, se elige una arbitrariamente.

authkey es la clave de autenticación que se utilizará para verificar la validez de las conexiones entrantes al proceso del servidor. Si *authkey* es `None`, entonces se usa `current_process().authkey`. De lo contrario, se usa *authkey* y debe ser una cadena de *bytes*.

serializer debe ser `'pickle'` (usar serialización **`pickle`**) o `'xmlrpclib'` (usar serialización **`xmlrpc.client`**).

ctx is a context object, or `None` (use the current context). If `None`, calling this may set the global start method. See Global start method for more details.

shutdown_timeout es un tiempo de espera en segundos que se utiliza para esperar hasta que el proceso utilizado por el administrador se complete en el método **`shutdown()`**. Si se agota el tiempo de apagado, el proceso finaliza. Si la finalización del proceso también supera el tiempo de espera, el proceso se cancela.

Distinto en la versión 3.11: Se agregó el parámetro *shutdown_timeout*.

start([*initializer*[, *initargs*]])

Se inicia un subproceso para iniciar el administrador. Si *initializer* no es `None`, entonces el subproceso llamará `initializer(*initargs)` cuando se inicie.

get_server()

Retorna un objeto **`Server`** que representa el servidor real bajo el control del Administrador. El objeto **`Server`** admite el método **`serve_forever()`**:

```
>>> from multiprocessing.managers import BaseMa
>>> manager = BaseManager(address=('', 50000),
>>> server = manager.get_server()
>>> server.serve_forever()
```

`Server` tiene un atributo adicional **`address`**.

connect()

Conecta un objeto de administrador local (*manager*) local a un proceso de administrador remoto:

```
>>> from multiprocessing.managers import BaseMa
>>> m = BaseManager(address=('127.0.0.1', 5000
>>> m.connect()
```

shutdown()

Detiene el proceso utilizado por el gerente (*manager*). Esto solo está disponible si **`start()`** se ha utilizado para iniciar el proceso del servidor.

Esto se puede llamar múltiples veces.

register(*typeid*[, *callable*[, *proxytype*[, *exposed*[, *method_to_typeid*[, *create_method*]]]]])

Un método de clase que puede usarse para registrar un tipo o invocarse con la clase de administrador (*manager*).

typeid es un «identificador de tipo» que se utiliza para identificar un tipo particular de objeto compartido. Esto debe ser una cadena de caracteres.

callable es un invocable utilizado para crear objetos para este identificador de tipo. Si una instancia de administrador se conectará al servidor utilizando el método `connect()`, o si el argumento *create_method* es `False`, esto se puede dejar como `None`.

proxytype es una subclase de `BaseProxy` que se usa para crear *proxies* para objetos compartidos con este *typeid*. Si `None`, se crea automáticamente una clase proxy.

Se utiliza *exposed* para especificar una secuencia de nombres de métodos a los que se debe permitir el acceso de los servidores proxy para este tipo de identificación utilizando `BaseProxy._callmethod()`. (Si *exposed* es `None`, entonces **`proxytype._exposed_`** se usa en su lugar si existe). En el caso de que no se especifique una lista expuesta, todos los «métodos públicos» del objeto compartido serán accesibles . (Aquí un «método público» significa cualquier atributo que tenga un método `__call__()` y cuyo nombre no comience con `'_'`.)

El *method_to_typeid* es una asignación utilizada para especificar el tipo de retorno de los métodos expuestos que deberían retornar un proxy. Asigna nombres de métodos a cadenas *typeid*. (Si *method_to_typeid* es `None` entonces **`proxytype._method_to_typeid`** se usa en su lugar si existe). Si el nombre de un método no es una clave de esta asignación o si la asignación es `None` entonces el objeto retornado por el método se copiará por valor.

create_method determina si un método debe crearse con el nombre *typeid* que se puede usar para indicarle al proceso del servidor que cree un nuevo objeto compartido y retornando un proxy para él. Por defecto es `True`.

`BaseManager` las instancias también tienen una propiedad de solo lectura:

address

La dirección utilizada por el administrador.

Distinto en la versión 3.3: Los objetos de administrador admiten el protocolo de gestión de contexto; consulte Tipos gestores de contexto. `__enter__()` inicia el proceso del servidor (si aún no se ha iniciado) y luego retorna el objeto de administrador. `__exit__()` llama `shutdown()`.

En versiones anteriores `__enter__()` no iniciaba el proceso del servidor del administrador si aún no se había iniciado.

class multiprocessing.managers.SyncManager

Una subclase de `BaseManager` que se puede utilizar para la sincronización de procesos. Los objetos de este tipo son retornados por `multiprocessing.Manager()`.

Sus métodos crean y retornan Objetos Proxy (Proxy Objects) para varios tipos de datos de uso común que se sincronizarán entre procesos. Esto incluye notablemente listas compartidas y diccionarios.

Barrier(*parties*[, *action*[, *timeout*]])

`threading.Barrier` crea un objeto compartido y retorna un proxy para él.

Added in version 3.3.

BoundedSemaphore([*value*])

Crea un objeto compartido `threading.BoundedSemaphore` y retorna un proxy para él.

Condition([*lock*])

Crea un objeto compartido `threading.Condition` y retorna un proxy para él.

Si se proporciona *lock*, debería ser un proxy para un objeto `threading.Lock` o `threading.RLock`.

Distinto en la versión 3.3: El método **`wait_for()`** fue a añadido.

Event()

Crea un objeto compartido **`threading.Event`** y retorna un proxy para él.

Lock()

Crea un objeto compartido **`threading.Lock`** y retorna un proxy para él.

Namespace()

Crea un objeto compartido **`Namespace`** y retorna un proxy para él.

Queue([*maxsize*])

Crea un objeto compartido **`queue.Queue`** y retorna un proxy para él.

RLock()

Crea un objeto compartido **`threading.RLock`** y retorna un proxy para él.

Semaphore([*value*])

Crea un objeto compartido **`threading.Semaphore`** y retorna un proxy para él.

Array(*typecode, sequence*)

Crea un arreglo y retorna un proxy para ello.

Value(*typecode, value*)

Crea un objeto con un atributo de escritura `value` y retorna un proxy para él.

dict()

dict(*mapping*)
dict(*sequence*)

Crea un objeto compartido `dict` y retorna un proxy para él.

list()
list(*sequence*)

Crea un objeto compartido `list` y retorna un proxy para él.

set()
set(*sequence*)
set(*mapping*)

Create a shared `set` object and return a proxy for it.

Added in version 3.14: `set` support was added.

Distinto en la versión 3.6: Los objetos compartidos pueden anidarse. Por ejemplo, un objeto contenedor compartido, como una lista compartida, puede contener otros objetos compartidos que serán administrados y sincronizados por `SyncManager`.

class multiprocessing.managers.Namespace

Un tipo que puede registrarse con `SyncManager`.

Un objeto de espacio de nombres no tiene métodos públicos, pero tiene atributos de escritura. Su representación muestra los valores de sus atributos.

Sin embargo, cuando se usa un proxy para un objeto de espacio de nombres, un atributo que comience con `'_'` será un atributo del proxy y no un atributo del referente:

```
>>> mp_context = multiprocessing.get_context('spawn
>>> manager = mp_context.Manager()
>>> Global = manager.Namespace()
>>> Global.x = 10
>>> Global.y = 'hello'
>>> Global._z = 12.3    # this is an attribute of t
>>> print(Global)
Namespace(x=10, y='hello')
```

Administradores customizables (*Customized managers*)

Para crear su propio administrador, uno crea una subclase de `BaseManager` y utiliza el método de clase `register()` para registrar nuevos tipos o llamadas con la clase de administrador. Por ejemplo:

```
from multiprocessing.managers import BaseManager

class MathsClass:
    def add(self, x, y):
        return x + y
    def mul(self, x, y):
        return x * y

class MyManager(BaseManager):
    pass

MyManager.register('Maths', MathsClass)

if __name__ == '__main__':
    with MyManager() as manager:
        maths = manager.Maths()
        print(maths.add(4, 3))         # prints 7
        print(maths.mul(7, 8))         # prints 56
```

Utilizando un administrador remoto

Es posible ejecutar un servidor administrador en una máquina y hacer que los clientes lo usen desde otras máquinas (suponiendo que los cortafuegos involucrados lo permitan).

La ejecución de los siguientes comandos crea un servidor para una única cola compartida a la que los clientes remotos pueden acceder:

```
>>> from multiprocessing.managers import BaseManager
>>> from queue import Queue
>>> queue = Queue()
>>> class QueueManager(BaseManager): pass
>>> QueueManager.register('get_queue', callable=lambda:q
>>> m = QueueManager(address=('', 50000), authkey=b'abra
>>> s = m.get_server()
>>> s.serve_forever()
```

Un cliente puede tener accesos al servidor de la siguiente manera:

```
>>> from multiprocessing.managers import BaseManager
>>> class QueueManager(BaseManager): pass
>>> QueueManager.register('get_queue')
>>> m = QueueManager(address=('foo.bar.org', 50000), aut
>>> m.connect()
>>> queue = m.get_queue()
>>> queue.put('hello')
```

Otro cliente puede también usarlo:

```
>>> from multiprocessing.managers import BaseManager
>>> class QueueManager(BaseManager): pass
>>> QueueManager.register('get_queue')
>>> m = QueueManager(address=('foo.bar.org', 50000), aut
>>> m.connect()
>>> queue = m.get_queue()
>>> queue.get()
'hello'
```

Los procesos locales también pueden acceder a esa cola (*queue*), utilizando el código de arriba en el cliente para acceder de forma remota:

```
>>> from multiprocessing import Process, Queue
>>> from multiprocessing.managers import BaseManager
>>> class Worker(Process):
...     def __init__(self, q):
...         self.q = q
...         super().__init__()
...     def run(self):
...         self.q.put('local hello')
...
>>> queue = Queue()
>>> w = Worker(queue)
>>> w.start()
>>> class QueueManager(BaseManager): pass
...
>>> QueueManager.register('get_queue', callable=lambda:
>>> m = QueueManager(address=('', 50000), authkey=b'abra
>>> s = m.get_server()
>>> s.serve_forever()
```

Objetos Proxy (*Proxy Objects*)

Un proxy es un objeto que *se refiere* a un objeto compartido que vive (presumiblemente) en un proceso diferente. Se dice que el objeto compartido es el *referente* del proxy. Varios objetos proxy pueden tener el mismo referente.

Un objeto proxy tiene métodos que invocan los métodos correspondientes de su referente (aunque no todos los métodos del referente estarán necesariamente disponibles a través del proxy). De esta manera, un proxy se puede usar al igual que su referente:

```
>>> mp_context = multiprocessing.get_context('spawn')
>>> manager = mp_context.Manager()
>>> l = manager.list([i*i for i in range(10)])
>>> print(l)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> print(repr(l))
<ListProxy object, typeid 'list' at 0x...>
>>> l[4]
16
>>> l[2:5]
[4, 9, 16]
```

Tenga en cuenta que la aplicación **`str()`** a un proxy retornará la representación del referente, mientras que la aplicación **`repr()`**

retornará la representación del proxy.

Una característica importante de los objetos proxy es que son seleccionables para que puedan pasarse entre procesos. Como tal, un referente puede contener Objetos Proxy (Proxy Objects). Esto permite anidar estas listas administradas, dictados y otros Objetos Proxy (Proxy Objects):

```
>>> a = manager.list()
>>> b = manager.list()
>>> a.append(b)         # referent of a now contains ref
>>> print(a, b)
[<ListProxy object, typeid 'list' at ...>] []
>>> b.append('hello')
>>> print(a[0], b)
['hello'] ['hello']
```

Del mismo modo, los proxies *dict* y *list* pueden estar anidados uno dentro del otro:

```
>>> l_outer = manager.list([ manager.dict() for i in rar
>>> d_first_inner = l_outer[0]
>>> d_first_inner['a'] = 1
>>> d_first_inner['b'] = 2
>>> l_outer[1]['c'] = 3
>>> l_outer[1]['z'] = 26
>>> print(l_outer[0])
{'a': 1, 'b': 2}
>>> print(l_outer[1])
{'c': 3, 'z': 26}
```

Si los objetos estándar (no proxy) **`list`** or **`dict`** están contenidos en un referente, las modificaciones a esos valores mutables no se propagarán a través del administrador porque el proxy no tiene forma de saber cuándo los valores contenidos dentro son modificados. Sin embargo, almacenar un valor en un proxy de contenedor (que desencadena un `__setitem__` en el objeto proxy) se propaga a través del administrador y, por lo tanto, para modificar efectivamente dicho elemento, uno podría reasignar el valor modificado al proxy de contenedor:

```
# create a list proxy and append a mutable object (a dic
lproxy = manager.list()
lproxy.append({})
# now mutate the dictionary
d = lproxy[0]
d['a'] = 1
d['b'] = 2
# at this point, the changes to d are not yet synced, bu
# updating the dictionary, the proxy is notified of the
lproxy[0] = d
```

Este enfoque es quizás menos conveniente que emplear anidado Objetos Proxy (Proxy Objects) para la mayoría de los casos de uso, pero también demuestra un nivel de control sobre la sincronización.

Nota

The proxy types in **`multiprocessing`** do nothing to support comparisons by value. So, for instance, we have:

```
>>> manager.list([1,2,3]) == [1,2,3]
False
```

En su lugar, se debe usar una copia del referente al hacer comparaciones.

class multiprocessing.managers.BaseProxy

Los objetos proxy son instancias de subclases de **`BaseProxy`**.

_callmethod(*methodname*[, *args*[, *kwds*]])

Llama y retorna el resultado de un método del referente del proxy.

Si `proxy` es un proxy cuyo referente es `obj` entonces la expresión

```
proxy._callmethod(methodname, args, kwds)
```

evaluará la expresión

```
getattr(obj, methodname)(*args, **kwds)
```

en el proceso del administrador.

El valor retornado será una copia del resultado de la llamada o un proxy a un nuevo objeto compartido; consulte la documentación del argumento *method_to_typeid* de `BaseManager.register()`.

Si la llamada genera una excepción, entonces se vuelve a generar `_callmethod()`. Si se genera alguna otra excepción en el proceso del administrador, esto se convierte en una excepción **`RemoteError`** y se genera mediante **`_callmethod()`**.

Tenga en cuenta en particular que se lanzará una excepción si *methodname* no ha sido *exposed*.

Un ejemplo de uso de `_callmethod()`:

```
>>> l = manager.list(range(10))
>>> l._callmethod('__len__')
10
>>> l._callmethod('__getitem__', (slice(2, 7),)
[2, 3, 4, 5, 6]
>>> l._callmethod('__getitem__', (20,))
Traceback (most recent call last):
...
IndexError: list index out of range
```

_getvalue()

Retorna una copia del referente.

Si el referente no se puede deserializar (*unpicklable*), esto lanzará una excepción.

repr()

Retorna una representación de un objeto proxy.

__str__()

Retorna una representación del referente.

Limpieza (*Cleanup*)

Un objeto proxy utiliza una devolución de llamada (*callback*) de referencia débil (*weakref*) para que cuando sea recolectado por el recolector de basura se da de baja del administrador que posee su referente.

Un objeto compartido se elimina del proceso del administrador cuando ya no hay ningún proxy que se refiera a él.

Piscinas de procesos (*Process Pools*)

Se puede crear un grupo de procesos que llevarán a cabo las tareas que se le presenten con la `Pool` class.

class multiprocessing.pool.Pool([*processes*[, *initializer*[, *initargs*[, *maxtasksperchild*[, *context*]]]]])

Un objeto de grupo de procesos que controla un grupo de procesos de trabajo a los que se pueden enviar trabajos. Admite resultados asincrónicos con tiempos de espera y devoluciones de llamada y tiene una implementación de mapa paralelo.

processes is the number of worker processes to use. If *processes* is `None` then the number returned by `os.process_cpu_count()` is used.

Si *initializer* no es `None`, cada proceso de trabajo llamará `initializer(*initargs)` cuando se inicie.

maxtasksperchild es el número de tareas que un proceso de trabajo puede completar antes de salir y ser reemplazado por un proceso de trabajo nuevo, para permitir que se liberen los recursos no utilizados. El valor predeterminado *maxtasksperchild* es `None`, lo que significa que los procesos de trabajo vivirán tanto tiempo como el grupo.

context can be used to specify the context used for starting the worker processes. Usually a pool is created using the function **`multiprocessing.Pool()`** or the **`Pool()`** method of a context object. In both cases *context* is set appropriately. If `None`, calling this function will have the side effect of setting the current global start method if it has not been set already. See the **`get_context()`** function.

Tenga en cuenta que los métodos del objeto de grupo solo deben ser invocados por el proceso que creó el grupo.

Advertencia

Los objetos **`multiprocessing.pool`** tienen recursos internos que necesitan ser administrados adecuadamente (como cualquier otro recurso) utilizando el grupo como administrador de contexto o llamando a **`close()`** y **`terminate()`** manualmente. De lo contrario, el proceso puede demorarse en la finalización.

Tenga en cuenta que **no es correcto** confiar en el recolector de basura para destruir el grupo ya que *CPython* no asegura que se llamará al finalizador del grupo (consulte **`object.__del__()`** para obtener más información).

Distinto en la versión 3.2: Added the *maxtasksperchild* parameter.

Distinto en la versión 3.4: Added the *context* parameter.

Distinto en la versión 3.13: processes uses **`os.process_cpu_count()`** by default, instead of **`os.cpu_count()`**.

Nota

Worker processes within a **`Pool`** typically live for the complete duration of the Pool's work queue. A frequent pattern found in other systems (such as Apache, mod_wsgi, etc) to free resources held by workers is to allow a worker within a pool to complete only a set amount of work before exiting, being cleaned up and a new process spawned to replace the old one. The *maxtasksperchild* argument to the **`Pool`** exposes this ability to the end user.

apply(*func*[, *args*[, *kwds*]])

Llama a *func* con argumentos *args* y argumentos de palabras clave *kwds*. Se bloquea hasta que el resultado esté listo. Dados estos bloques, **`apply_async()`** es más adecuado para realizar trabajos en paralelo. Además, *func* solo se ejecuta en uno de los trabajadores de piscina.

apply_async(*func*[, *args*[, *kwds*[, *callback*[, *error_callback*]]]])

Una variante del método **`apply()`** que retorna un objeto **`AsyncResult`**.

Si se especifica *callback*, debería ser un invocable que acepte un único argumento. Cuando el resultado está listo, se le aplica *callback*, a menos que la llamada falle, en cuyo caso se aplica *error_callback*.

Si se especifica *error_callback*, debería ser un invocable que acepte un único argumento. Si la función de destino falla, se llama a *error_callback* con la instancia de excepción.

Las devoluciones de llamada deben completarse inmediatamente ya que de lo contrario el hilo que maneja los resultados se bloqueará.

map(*func*, *iterable*[, *chunksize*])

Un equivalente paralelo de la función incorporada **`map()`** (aunque solo admite un argumento *iterable*, para varios iterables consulte **`starmap()`**). Bloquea hasta que el resultado esté listo.

Este método corta el iterable en varios trozos que envía al grupo de procesos como tareas separadas. El tamaño (aproximado) de estos fragmentos se puede especificar estableciendo *chunksize* en un entero positivo.

Tenga en cuenta que puede causar un alto uso de memoria para iterables muy largos. Considere usar **`imap()`** o **`imap_unordered()`** con la opción explícita *chunksize* para una mejor eficiencia.

map_async(*func*, *iterable*[, *chunksize*[, *callback*[, *error_callback*]]])

Una variante del método **`map()`** que retorna un objeto **`AsyncResult`**.

Si se especifica *callback*, debería ser un invocable que acepte un único argumento. Cuando el resultado está listo, se le aplica *callback*, a menos que la llamada falle, en cuyo caso se aplica *error_callback*.

Si se especifica *error_callback*, debería ser un invocable que acepte un único argumento. Si la función de destino falla, se llama a *error_callback* con la instancia de excepción.

Las devoluciones de llamada deben completarse inmediatamente ya que de lo contrario el hilo que maneja los resultados se bloqueará.

imap(*func*, *iterable*[, *chunksize*])

Una versión más perezosa (*lazier*) de **`map()`**.

El argumento *chunksize* es el mismo que el utilizado por el método **`map()`**. Para iterables muy largos, usar un valor grande para *chunksize* puede hacer que el trabajo se complete **much** (**mucho**) más rápido que usar el valor predeterminado de `1`.

Además, si *chunksize* es `1`, el método **`next()`** del iterador retornado por el método **`imap()`** tiene un parámetro opcional *timeout* :`next (timeout)` lanzará **`multiprocessing.TimeoutError`** si el resultado no puede retornarse dentro de *timeout* segundos.

imap_unordered(*func*, *iterable*[, *chunksize*])

Lo mismo que **`imap()`**, excepto que el orden de los resultados del iterador retornado debe considerarse arbitrario. (Solo cuando hay un solo proceso de trabajo se garantiza que el orden sea «correcto»).

starmap(*func*, *iterable*[, *chunksize*])

Como **`map()`**, excepto que se espera que los elementos de *iterable* sean iterables que se desempaquetan como argumentos.

Por lo tanto, un *iterable* de `[(1,2), (3, 4)]` da como resultado `[func(1,2), func(3,4)]`.

Added in version 3.3.

starmap_async(*func*, *iterable*[, *chunksize*[, *callback*[, *error_callback*]]])

Una combinación de **`starmap()`** y **`map_async()`** que itera sobre *iterable* de iterables y llama a *func* con los iterables desempaquetados. Como resultado se retorna un objeto.

Added in version 3.3.

close()

Impide que se envíen más tareas a la piscina (*pool*). Una vez que se hayan completado todas las tareas, se cerrarán los procesos de trabajo.

terminate()

Detiene los procesos de trabajo inmediatamente sin completar el trabajo pendiente. Cuando el objeto del grupo es basura recolectada `terminate()` se llamará inmediatamente.

join()

Espera a que salgan los procesos de trabajo. Se debe llamar `close()` o `terminate()` antes de usar `join()`.

Distinto en la versión 3.3: Los objetos de piscina (*pool*) ahora admiten el protocolo de administración de contexto; consulte Tipos gestores de contexto. **`__enter__()`** retorna el objeto de grupo, y **`__exit__()`** llama `terminate()`.

class multiprocessing.pool.AsyncResult

La clase del resultado retornado por `Pool.apply_async()` y `Pool.map_async()`.

get([*timeout*])

Retorna el resultado cuando llegue. Si *timeout* no es `None` y el resultado no llega dentro de *timeout* segundos, entonces se lanza `multiprocessing.TimeoutError`. Si la llamada remota generó una excepción, esa excepción se volverá a plantear mediante `get()`.

wait([*timeout*])

Espera hasta que el resultado esté disponible o hasta que pase *timeout* segundos.

ready()

Retorna si la llamada se ha completado.

successful()

Retorna si la llamada se completó sin generar una excepción. Lanzará `ValueError` si el resultado no está listo.

Distinto en la versión 3.7: Si el resultado no está listo **`ValueError`** aparece en lugar de **`AssertionError`**.

El siguiente ejemplo demuestra el uso de una piscina(*pool*):

```
from multiprocessing import Pool
import time

def f(x):
    return x*x

if __name__ == '__main__':
    with Pool(processes=4) as pool:         # start 4 wo
        result = pool.apply_async(f, (10,)) # evaluate "
        print(result.get(timeout=1))        # prints "10

        print(pool.map(f, range(10)))       # prints "[0

        it = pool.imap(f, range(10))
        print(next(it))                     # prints "0"
        print(next(it))                     # prints "1"
        print(it.next(timeout=1))           # prints "4"

        result = pool.apply_async(time.sleep, (10,))
        print(result.get(timeout=1))        # raises mul
```

Oyentes y clientes (*Listeners and Clients*)

Por lo general, el paso de mensajes entre procesos se realiza mediante colas o mediante objetos **`Connection`** retornados por **`Pipe()`**.

However, the **`multiprocessing.connection`** module allows some extra flexibility. It basically gives a high level message oriented API for dealing with sockets or Windows named pipes. It also has support for *digest authentication* using the **`hmac`** module, and for polling multiple connections at the same time.

multiprocessing.connection.deliver_challenge(*connection*, *authkey*)

Envía un mensaje generado aleatoriamente al otro extremo de la conexión y espera una respuesta.

Si la respuesta coincide con el resumen del mensaje utilizando *authkey* como clave, se envía un mensaje de bienvenida al otro extremo de la conexión. De lo contrario se lanza **`AuthenticationError`**.

multiprocessing.connection.answer_challenge(*connection*, *authkey*)

Recibe un mensaje, calcula el resumen del mensaje usando *authkey* como la clave y luego envía el resumen de vuelta.

Si no se recibe un mensaje de bienvenida, se lanza **`AuthenticationError`**.

multiprocessing.connection.Client(*address*[, *family*[, *authkey*]])

Se intenta configurar una conexión con el oyente que utiliza la dirección *address*, retornando **`Connection`**.

El tipo de conexión está determinado por el argumento *family*, pero esto generalmente se puede omitir ya que generalmente se puede inferir del formato de *address*. (Consulte Formatos de dirección (Address formats))

If *authkey* is given and not `None`, it should be a byte string and will be used as the secret key for an HMAC-based authentication challenge. No authentication is done if *authkey* is `None`. **`AuthenticationError`** is raised if authentication fails. See Llaves de autentificación.

class multiprocessing.connection.Listener([*address*[, *family*[, *backlog*[, *authkey*]]]])

Un contenedor para un *socket* vinculado o una tubería (*pipe*) con nombre de Windows que está "escuchando" las conexiones.

address es la dirección que utilizará el *socket* vinculado o la conocida tubería (*pipe*) con nombre del objeto de escucha.

Nota

Si se usa una dirección de “0.0.0.0” , la dirección no será un punto final conectable en Windows. Si necesita un punto final conectable, debe usar “127.0.0.1”.

family es el tipo de socket (o tubería con nombre) a utilizar. Esta puede ser una de las cadenas de caracteres `'AF_INET'` (para un socket TCP), `' AF_UNIX'` (para un socket de dominio Unix) o `'AF_PIPE'` (para una tubería con nombre de Windows) . De estos, solo el primero está garantizado para estar disponible. Si *family* es `None` , *family* se deduce del formato de *address*. Si *address* también es `None` , se elige un valor predeterminado. Este valor predeterminado es *family* con la opción más rápida disponible. Consulte Formatos de dirección (Address formats). Tenga en cuenta que si *family* es `'AF_UNIX'` y la dirección es `None`, el *socket* se creará en un directorio temporal privado usando `tempfile.mkstemp()`.

Si el objeto de escucha utiliza un *socket*, entonces *backlog* (1 por defecto) se pasa al método `listen()` del *socket* una vez que se ha vinculado.

If *authkey* is given and not `None`, it should be a byte string and will be used as the secret key for an HMAC-based authentication challenge. No authentication is done if *authkey* is `None`. `AuthenticationError` is raised if authentication fails. See Llaves de autentificación.

accept()

> Acepta una conexión en el *socket* vinculado o canalización con nombre del objeto de escucha y retorne un objeto `Connection`. Si se intenta la autenticación y falla, entonces se lanza una `AuthenticationError`.

close()

> Cierra el socket vinculado o la tubería con nombre del objeto de escucha. Esto se llama automáticamente cuando el oyente es recolectado por el recolector de

basura. Sin embargo, es aconsejable llamarlo explícitamente.

Los objetos de escucha tienen las siguientes propiedades de solo lectura:

address

La dirección que está utilizando el objeto *Listener*.

last_accepted

La dirección de donde vino la última conexión aceptada. Si esto no está disponible, entonces es `None`.

Distinto en la versión 3.3: Los objetos de escucha ahora admiten el protocolo de gestión de contexto – consulte Tipos gestores de contexto. El objeto *lISTENER* retorna **`__enter__()`**, y **`__exit__()`** llama a **`close()`**.

multiprocessing.connection.wait(*object_list*, *timeout=None*)

Espera hasta que un objeto en *object_list* esté listo. Retorna la lista de esos objetos en *object_list* que están listos. Si *timeout* es flotante, la llamada se bloquea durante como máximo tantos segundos. Si *timeout* es `` `None``, se bloqueará por un período ilimitado. Un tiempo de espera negativo es equivalente a un tiempo de espera cero.

For both POSIX and Windows, an object can appear in *object_list* if it is

- un objeto legible de **`Connection`**;
- un objeto conectado y legible de **`socket.socket`**; o
- el atributo **`sentinel`** de un objeto **`Process`**.

Un objeto de conexión o *socket* está listo cuando hay datos disponibles para leer, o el otro extremo se ha cerrado.

POSIX: `wait(object_list, timeout)` almost equivalent `select.select(object_list, [], [], timeout)`. The difference is that, if **`select.select()`** is interrupted by a signal, it can raise **`OSError`** with an error number of `EINTR`, whereas **`wait()`** will not.

Windows: An item in *object_list* must either be an integer handle which is waitable (according to the definition used by the documentation of the Win32 function `WaitForMultipleObjects()`) or it can be an object with a **`fileno()`** method which returns a socket handle or pipe handle. (Note that pipe handles and socket handles are **not** waitable handles.)

Added in version 3.3.

Ejemplos

El siguiente código de servidor crea un escucha que utiliza `'secret password'` como clave de autenticación. Luego espera una conexión y envía algunos datos al cliente:

```
from multiprocessing.connection import Listener
from array import array

address = ('localhost', 6000)     # family is deduced to

with Listener(address, authkey=b'secret password') as li
    with listener.accept() as conn:
        print('connection accepted from', listener.last_

        conn.send([2.25, None, 'junk', float])

        conn.send_bytes(b'hello')

        conn.send_bytes(array('i', [42, 1729]))
```

El siguiente código se conecta al servidor y recibe algunos datos del servidor:

```
from multiprocessing.connection import Client
from array import array
```

```
address = ('localhost', 6000)

with Client(address, authkey=b'secret password') as conn
    print(conn.recv())                  # => [2.25, None

    print(conn.recv_bytes())            # => 'hello'

    arr = array('i', [0, 0, 0, 0, 0])
    print(conn.recv_bytes_into(arr))    # => 8
    print(arr)                          # => array('i',
```

El siguiente código utiliza **`wait()`** para esperar mensajes de múltiples procesos a la vez:

```
from multiprocessing import Process, Pipe, current_proce
from multiprocessing.connection import wait

def foo(w):
    for i in range(10):
        w.send((i, current_process().name))
    w.close()

if __name__ == '__main__':
    readers = []

    for i in range(4):
        r, w = Pipe(duplex=False)
        readers.append(r)
        p = Process(target=foo, args=(w,))
        p.start()
        # We close the writable end of the pipe now to b
        # p is the only process which owns a handle for
        # ensures that when p closes its handle for the
        # wait() will promptly report the readable end a
        w.close()

    while readers:
        for r in wait(readers):
            try:
```

```
        msg = r.recv()
    except EOFError:
        readers.remove(r)
    else:
        print(msg)
```

Formatos de dirección (*Address formats*)

- Una dirección `'AF_INET'` es una tupla de la forma `(hostname, port)` donde *hostname* es una cadena de caracteres y *port* es un número entero.
- Una dirección `'AF_UNIX'` es una cadena que representa un nombre de archivo en el sistema de archivos.
- Una dirección `'AF_PIPE'` es una cadena con el formato `r'\\.\pipe\PipeName'`. Para usar **`Client()`** para conectarse a una tubería con nombre en una computadora remota llamada *ServerName*, se debe usar una dirección de la forma `r'\\ServerName\pipe\PipeName'` en cambio.

Tenga en cuenta que cualquier cadena que comience con dos barras inclinadas invertidas se asume por defecto como una dirección `'AF_PIPE'` en lugar de una dirección `'AF_UNIX'`.

Llaves de autentificación

Cuando uno usa **`Connection.recv`**, los datos recibidos se desbloquean automáticamente. Desafortunadamente, la eliminación de datos de una fuente no confiable es un riesgo de seguridad. Por lo tanto **`Listener`** y **`Client()`** usan el módulo **`hmac`** para proporcionar autenticación de resumen.

Una clave de autenticación es una cadena de bytes que se puede considerar como una contraseña: una vez que se establece una conexión, ambos extremos exigirán pruebas de que el otro conoce la clave de autenticación. (Demostrar que ambos extremos están usando la misma clave **no** implica enviar la clave a través de la conexión).

Si se solicita la autenticación pero no se especifica una clave de autenticación, se utiliza el valor de retorno de `current_process().authkey` (consulte `Process`). Este valor será heredado automáticamente por cualquier objeto **`Process`** que crea el proceso actual. Esto significa que (por defecto) todos los procesos de un programa multiproceso compartirán una única clave de autenticación que se puede usar al configurar conexiones entre ellos.

Las claves de autenticación adecuadas también se pueden generar utilizando `os.urandom()`.

This authentication protects `Listener` and `Client()` connections, which are reachable by address. It is not applied to the anonymous pipes created by `Pipe()` or used internally by `Queue`. **`multiprocessing`** treats all local processes running as the same user as trusted; on most operating systems such processes can access each other's pipe file descriptors regardless. Applications that require isolation between processes of the same user must arrange it at the operating-system level – for example, by running workers under a different user account or in a sandbox.

Logging

Existe cierto soporte para el registro. Sin embargo, tenga en cuenta que el paquete `logging` no utiliza candados compartidos de proceso, por lo que es posible (dependiendo del tipo de controlador) que los mensajes de diferentes procesos se mezclen.

multiprocessing.get_logger()

> Returns the logger used by **`multiprocessing`**. If necessary, a new one will be created.
>
> When first created the logger has level `logging.NOTSET` and no default handler. Messages sent to this logger will not by default propagate to the root logger.
>
> Tenga en cuenta que en Windows los procesos hijos solo heredarán el nivel del registrador del proceso parental – no se heredará ninguna otra personalización del registrador.

multiprocessing.log_to_stderr(*level=None*)

Esta función realiza una llamada a `get_logger()` pero además de retornar el registrador creado por get_logger, agrega un controlador que envía la salida a `sys.stderr` usando el formato `'[%(levelname)s/%(processName)s] %(message)s'`. Puede modificar `levelname` del registrador pasando un argumento `level`.

A continuación se muestra una sesión de ejemplo con el registro activado:

```
>>> import multiprocessing, logging
>>> logger = multiprocessing.log_to_stderr()
>>> logger.setLevel(logging.INFO)
>>> logger.warning('doomed')
[WARNING/MainProcess] doomed
>>> m = multiprocessing.Manager()
[INFO/SyncManager-...] child process calling self.run()
[INFO/SyncManager-...] created temp directory /.../pymp-
[INFO/SyncManager-...] manager serving at '/.../listener
>>> del m
[INFO/MainProcess] sending shutdown message to manager
[INFO/SyncManager-...] manager exiting with exitcode 0
```

Para obtener una tabla completa de niveles de registro, consulte el módulo `logging`.

The `multiprocessing.dummy` module

`multiprocessing.dummy` replicates the API of **`multiprocessing`** but is no more than a wrapper around the `threading` module.

In particular, the `Pool` function provided by **`multiprocessing.dummy`** returns an instance of `ThreadPool`, which is a subclass of `Pool` that supports all the same method calls but uses a pool of worker threads rather than worker processes.

class multiprocessing.pool.ThreadPool([*processes*[, *initializer*[, *initargs*]]])

Un objeto de *pool* de hebras que controla un *pool* de hebras trabajadoras a las cuales se pueden enviar trabajos. La interfaz de las instancias de `ThreadPool` son totalmente compatibles con las instancias de `Pool`, y sus recursos también deben ser debidamente administrador, ya sea usando el *pool* como un gestor de contexto, o llamando a `close()` y `terminate()` manualmente.

processes is the number of worker threads to use. If *processes* is `None` then the number returned by `os.process_cpu_count()` is used.

Si *initializer* no es `None`, cada proceso de trabajo llamará `initializer(*initargs)` cuando se inicie.

A diferencia de `Pool`, *maxtasksperchild* and *context* no se pueden entregar.

Nota

Un `ThreadPool` comparte la misma interfaz que `Pool`, la cual está diseñada alrededor de un *pool* de procesos, y precede la introducción del módulo `concurrent.futures`. Como tal, hereda algunas operaciones que no tienen sentido para un *pool* basado en hebras, y tiene su propio tipo para representar el estado de trabajos asíncronos, `AsyncResult`, el cual no es entendido por otras librerías.

Los usuarios deberían por lo general preferir usar `concurrent.futures.ThreadPoolExecutor`, el cual tiene una interfaz más simple que fue diseñada alrededor de hebras desde un principio, y que retorna instancias de `concurrent.futures.Future`, las que son compatibles con muchas más librerías, incluyendo `asyncio`.

Pautas de programación

There are certain guidelines and idioms which should be adhered to when using `multiprocessing`.

Todos los métodos de inicio

Lo siguiente se aplica a todos los métodos de inicio.

Evita estado compartido

> En la medida de lo posible, se debe tratar de evitar el desplazamiento de grandes cantidades de datos entre procesos.
>
> Probablemente sea mejor seguir usando colas (*queues*) o tuberías (*pipes*) para la comunicación entre procesos en lugar de usar las primitivas de sincronización de nivel inferior.

Serialización (*picklability*)

> Asegúrese que todos los argumentos de los métodos de *proxies* son serializables (*pickable*)

Seguridad de hilos de *proxies*

> No usa un objeto proxy de más de un hilo a menos que lo proteja con un candado (*lock*).
>
> (Nunca hay un problema con diferentes procesos que usan el *mismo* proxy.)

Uniéndose a procesos zombies

> On POSIX when a process finishes but has not been joined it becomes a zombie. There should never be very many because each time a new process starts (or `active_children()` is called) all completed processes which have not yet been joined will be joined. Also calling a finished process's `Process.is_alive` will join the process. Even so it is probably good practice to explicitly join all the processes that you start.

Mejor heredar que serializar/deserializar (*pickle/unpickle*)

> When using the *spawn* or *forkserver* start methods many types from `multiprocessing` need to be picklable so that child processes can use them. However, one should generally avoid sending shared objects to other processes using pipes or queues. Instead you should arrange the program so that a process which needs access to a shared resource created elsewhere can inherit it from an ancestor process.

Evita procesos de finalización

> El uso del método `Process.terminate` para detener un proceso puede causar que los recursos compartidos (como candados, semáforos, tuberías y colas) que el proceso utiliza actualmente se rompan o no disponible para otros procesos.
>
> Por lo tanto, probablemente sea mejor considerar usar `Process.terminate` en procesos que nunca usan recursos compartidos.

Unirse a procesos que usan colas

> Tenga en cuenta que un proceso que ha puesto elementos en una cola esperará antes de finalizar hasta que todos los elementos almacenados en búfer sean alimentados por el hilo «alimentador» a la tubería subyacente. (El proceso secundario puede llamar al método `Queue.cancel_join_thread` de la cola para evitar este comportamiento).
>
> Esto significa que siempre que use una cola debe asegurarse de que todos los elementos que se hayan puesto en la cola se eliminarán antes de unirse al proceso. De lo contrario, no puede estar seguro de que los procesos que han puesto elementos en la cola finalizarán. Recuerde también que los procesos no demoníacos se unirán automáticamente.

Un ejemplo que de bloqueo mutuo (*deadlock*) es el siguiente

```
from multiprocessing import Process, Queue

def f(q):
    q.put('X' * 1000000)

if __name__ == '__main__':
    queue = Queue()
    p = Process(target=f, args=(queue,))
    p.start()
    p.join()                    # this deadlocks
    obj = queue.get()
```

Una solución aquí sería intercambiar las dos últimas líneas (o simplemente eliminar la línea `p.join()`).

Se pasan recursos explícitamente a procesos hijos

On POSIX using the *fork* start method, a child process can make use of a shared resource created in a parent process using a global resource. However, it is better to pass the object as an argument to the constructor for the child process.

Además de hacer que el código (potencialmente) sea compatible con Windows y los otros métodos de inicio, esto también garantiza que mientras el proceso secundario siga vivo, el objeto no se recolectará en el proceso primario. Esto podría ser importante si se libera algún recurso cuando el objeto es basura recolectada en el proceso padre.

Entonces por ejemplo

```
from multiprocessing import Process, Lock

def f():
    ... do something using "lock" ...

if __name__ == '__main__':
    lock = Lock()
    for i in range(10):
        Process(target=f).start()
```

debería ser reescrito como

```
from multiprocessing import Process, Lock

def f(l):
    ... do something using "l" ...

if __name__ == '__main__':
    lock = Lock()
    for i in range(10):
        Process(target=f, args=(lock,)).start()
```

Tenga cuidado de reemplazar `sys.stdin` con un *«file like object»*

`multiprocessing` originally unconditionally called:

```
os.close(sys.stdin.fileno())
```

en el método **`multiprocessing.Process._bootstrap()`** — Esto dio lugar a problemas con los procesos en proceso. Esto ha sido cambiado a:

```
sys.stdin.close()
sys.stdin = open(os.open(os.devnull, os.O_RDONLY),
```

Which solves the fundamental issue of processes colliding with each other resulting in a bad file descriptor error, but introduces a potential danger to applications which replace `sys.stdin()` with a «file-like object» with output buffering. This danger is that if multiple processes call `close()` on this file-like object, it could result in the same data being flushed to the object multiple times, resulting in corruption.

Si escribe un objeto similar a un archivo e implementa su propio almacenamiento en caché, puede hacer que sea seguro para la bifurcación (*fork-safe*) almacenando el pid cada vez que se agrega al caché y descartando el caché cuando cambia el pid. Por ejemplo:

```
@property
def cache(self):
    pid = os.getpid()
    if pid != self._pid:
        self._pid = pid
        self._cache = []
    return self._cache
```

Para más información, consulte bpo-5155 [https://bugs.python.org/issue?@action=redirect&bpo=5155], bpo-5313 [https://bugs.python.org/issue?@action=redirect&bpo=5313] y bpo-5331 [https://bugs.python.org/issue?@action=redirect&bpo=5331]

Los métodos de inicio *spawn* y *forkserver*

There are a few extra restrictions which don't apply to the *fork* start method.

Más serialización (*pickability*)

Ensure that all arguments to **`Process`** are picklable. Also, if you subclass `Process.__init__`, you must make sure that instances will be picklable when the **`Process.start`** method is called.

Variables globales

Tenga en cuenta que si el código que se ejecuta en un proceso secundario intenta acceder a una variable global, entonces el valor que ve (si lo hay) puede no ser el mismo que el valor en el proceso primario en el momento en que fue llamado **`Process.start`**.

Sin embargo, las variables globales que son solo constantes de nivel de módulo no causan problemas.

Importando de manera segura el módulo principal

Make sure that the main module can be safely imported by a new Python interpreter without causing unintended side effects (such as starting a new process).

Por ejemplo, usando el método de inicio *spawn* o *forkserver* ejecutando este módulo fallaría produciendo **`RuntimeError`**:

```
from multiprocessing import Process

def foo():
    print('hello')

p = Process(target=foo)
p.start()
```

En su lugar, se debe proteger el «punto de entrada» (*«entry point»*) del programa utilizando como sigue

```
if __name__ == '__main__':
```

```
from multiprocessing import Process, freeze_suppor

def foo():
    print('hello')

if __name__ == '__main__':
    freeze_support()
    set_start_method('spawn')
    p = Process(target=foo)
    p.start()
```

(La línea `freeze_support()` puede omitirse si el programa se ejecuta normalmente en lugar de congelarse).

Esto permite que el intérprete de Python recién generado importe de forma segura el módulo y luego ejecute la función del módulo `foo()`.

Se aplican restricciones similares si se crea un grupo o administrador en el módulo principal.

Ejemplos

Demostración de cómo crear y usar gerentes y proxies personalizados:

```
from multiprocessing import freeze_support
from multiprocessing.managers import BaseManager, BasePr
import operator

##

class Foo:
    def f(self):
        print('you called Foo.f()')
    def g(self):
        print('you called Foo.g()')
    def _h(self):
        print('you called Foo._h()')

# A simple generator function
def baz():
    for i in range(10):
        yield i*i

# Proxy type for generator objects
```

```
class GeneratorProxy(BaseProxy):
    _exposed_ = ['__next__']
    def __iter__(self):
        return self
    def __next__(self):
        return self._callmethod('__next__')

# Function to return the operator module
def get_operator_module():
    return operator

##

class MyManager(BaseManager):
    pass

# register the Foo class; make `f()` and `g()` accessibl
MyManager.register('Foo1', Foo)

# register the Foo class; make `g()` and `_h()` accessib
MyManager.register('Foo2', Foo, exposed=('g', '_h'))

# register the generator function baz; use `GeneratorPro
MyManager.register('baz', baz, proxytype=GeneratorProxy)

# register get_operator_module(); make public functions
MyManager.register('operator', get_operator_module)

##

def test():
    manager = MyManager()
    manager.start()

    print('-' * 20)

    f1 = manager.Foo1()
    f1.f()
    f1.g()
    assert not hasattr(f1, '_h')
    assert sorted(f1._exposed_) == sorted(['f', 'g'])

    print('-' * 20)

    f2 = manager.Foo2()
    f2.g()
    f2._h()
    assert not hasattr(f2, 'f')
    assert sorted(f2._exposed_) == sorted(['g', '_h'])

    print('-' * 20)

    it = manager.baz()
    for i in it:
        print('<%d>' % i, end=' ')
    print()

    print('-' * 20)

    op = manager.operator()
    print('op.add(23, 45) =', op.add(23, 45))
    print('op.pow(2, 94) =', op.pow(2, 94))
    print('op._exposed_ =', op._exposed_)

##

if __name__ == '__main__':
    freeze_support()
    test()
```

Usando **`Pool`**:

```
import multiprocessing
import time
import random
import sys
```

```
#
# Functions used by test code
#

def calculate(func, args):
    result = func(*args)
    return '%s says that %s%s = %s' % (
        multiprocessing.current_process().name,
        func.__name__, args, result
        )

def calculatestar(args):
    return calculate(*args)

def mul(a, b):
    time.sleep(0.5 * random.random())
    return a * b

def plus(a, b):
    time.sleep(0.5 * random.random())
    return a + b

def f(x):
    return 1.0 / (x - 5.0)

def pow3(x):
    return x ** 3

def noop(x):
    pass

#
# Test code
#

def test():
    PROCESSES = 4
    print('Creating pool with %d processes\n' % PROCESSE
with multiprocessing.Pool(PROCESSES) as pool:
    #
    # Tests
    #

    TASKS = [(mul, (i, 7)) for i in range(10)] + \
            [(plus, (i, 8)) for i in range(10)]

    results = [pool.apply_async(calculate, t) for t
    imap_it = pool.imap(calculatestar, TASKS)
    imap_unordered_it = pool.imap_unordered(calculat

    print('Ordered results using pool.apply_async():
    for r in results:
        print('\t', r.get())
    print()

    print('Ordered results using pool.imap():')
    for x in imap_it:
        print('\t', x)
    print()

    print('Unordered results using pool.imap_unorder
    for x in imap_unordered_it:
        print('\t', x)
    print()

    print('Ordered results using pool.map() --- will
    for x in pool.map(calculatestar, TASKS):
        print('\t', x)
    print()

    #
    # Test error handling
    #

    print('Testing error handling:')
```

```
try:
    print(pool.apply(f, (5,)))
except ZeroDivisionError:
    print('\tGot ZeroDivisionError as expected f
else:
    raise AssertionError('expected ZeroDivisionE

try:
    print(pool.map(f, list(range(10))))
except ZeroDivisionError:
    print('\tGot ZeroDivisionError as expected f
else:
    raise AssertionError('expected ZeroDivisionE

try:
    print(list(pool.imap(f, list(range(10)))))
except ZeroDivisionError:
    print('\tGot ZeroDivisionError as expected f
else:
    raise AssertionError('expected ZeroDivisionE

it = pool.imap(f, list(range(10)))
for i in range(10):
    try:
        x = next(it)
    except ZeroDivisionError:
        if i == 5:
            pass
    except StopIteration:
        break
    else:
        if i == 5:
            raise AssertionError('expected ZeroD

assert i == 9
print('\tGot ZeroDivisionError as expected from
print()
```

```
        #
        # Testing timeouts
        #

        print('Testing ApplyResult.get() with timeout:',
        res = pool.apply_async(calculate, TASKS[0])
        while 1:
            sys.stdout.flush()
            try:
                sys.stdout.write('\n\t%s' % res.get(0.02
                break
            except multiprocessing.TimeoutError:
                sys.stdout.write('.')
        print()
        print()

        print('Testing IMapIterator.next() with timeout:
        it = pool.imap(calculatestar, TASKS)
        while 1:
            sys.stdout.flush()
            try:
                sys.stdout.write('\n\t%s' % it.next(0.02
            except StopIteration:
                break
            except multiprocessing.TimeoutError:
                sys.stdout.write('.')
        print()
        print()


if __name__ == '__main__':
    multiprocessing.freeze_support()
    test()
```

Un ejemplo que muestra cómo usar las colas para alimentar tareas a una colección de procesos de trabajo y recopilar los resultados:

```
import time
import random

from multiprocessing import Process, Queue, current_proc

#
# Function run by worker processes
#

def worker(input, output):
    for func, args in iter(input.get, 'STOP'):
        result = calculate(func, args)
        output.put(result)

#
# Function used to calculate result
#

def calculate(func, args):
    result = func(*args)
    return '%s says that %s%s = %s' % \
        (current_process().name, func.__name__, args, re

#
# Functions referenced by tasks
#

def mul(a, b):
    time.sleep(0.5*random.random())
    return a * b

def plus(a, b):
    time.sleep(0.5*random.random())
    return a + b

#
#
#
```

```
def test():
    NUMBER_OF_PROCESSES = 4
    TASKS1 = [(mul, (i, 7)) for i in range(20)]
    TASKS2 = [(plus, (i, 8)) for i in range(10)]

    # Create queues
    task_queue = Queue()
    done_queue = Queue()

    # Submit tasks
    for task in TASKS1:
        task_queue.put(task)

    # Start worker processes
    for i in range(NUMBER_OF_PROCESSES):
        Process(target=worker, args=(task_queue, done_qu

    # Get and print results
    print('Unordered results:')
    for i in range(len(TASKS1)):
        print('\t', done_queue.get())

    # Add more tasks using `put()`
    for task in TASKS2:
        task_queue.put(task)

    # Get and print some more results
    for i in range(len(TASKS2)):
        print('\t', done_queue.get())

    # Tell child processes to stop
    for i in range(NUMBER_OF_PROCESSES):
        task_queue.put('STOP')


if __name__ == '__main__':
    freeze_support()
    test()
```

`multiprocessing.shared_memor` — Shared memory for direct access across processes

Código fuente: Lib/multiprocessing/shared_memory.py [https://github.com/python/cpython/tree/3.14/Lib/multiprocessing/shared_memory.py]

Added in version 3.8.

This module provides a class, **`SharedMemory`**, for the allocation and management of shared memory to be accessed by one or more processes on a multicore or symmetric multiprocessor (SMP) machine. To assist with the life-cycle management of shared memory especially across distinct processes, a **`BaseManager`** subclass, **`SharedMemoryManager`**, is also provided in the **`multiprocessing.managers`** module.

In this module, shared memory refers to «POSIX style» shared memory blocks (though is not necessarily implemented explicitly as such) and does not refer to «distributed shared memory». This style of shared memory permits distinct processes to potentially read and write to a common (or shared) region of volatile memory. Processes are conventionally limited to only have access to their own process memory space but shared memory permits the sharing of data between processes, avoiding the need to instead send messages between processes containing that data. Sharing data directly via memory can provide significant performance benefits compared to sharing data via disk or socket or other communications requiring the serialization/deserialization and copying of data.

class multiprocessing.shared_memory.SharedMemory(*name=None*, *create=False*, *size=0*, ***, *track=True*)

Create an instance of the **`SharedMemory`** class for either creating a new shared memory block or attaching to an existing shared memory block. Each shared memory block is assigned a unique name. In this way, one process can create a shared memory block with a particular name and a different process can attach to that same shared memory block using that same name.

As a resource for sharing data across processes, shared memory blocks may outlive the original process that created them. When one process no longer needs access to a shared memory block that might still be needed by other processes, the **`close()`** method should be called. When a shared memory block is no longer needed by any process, the **`unlink()`** method should be called to ensure proper cleanup.

Parámetros:

- **name** (*str* | *None*) – The unique name for the requested shared memory, specified as a string. When creating a new shared memory block, if `None` (the default) is supplied for the name, a novel name will be generated.
- **create** (*bool*) – Control whether a new shared memory block is created (`True`) or an existing shared memory block is attached (`False`).
- **size** (*int*) – The requested number of bytes when creating a new shared memory block. Because some platforms choose to allocate chunks of memory based upon that platform's memory page size, the exact size of the shared memory block may be larger or equal to the size requested. When attaching to an existing shared memory block, the *size* parameter is ignored.
- **track** (*bool*) – When `True`, register the shared memory block with a resource tracker process on platforms where the OS does not do this automatically. The

resource tracker ensures proper cleanup of the shared memory even if all other processes with access to the memory exit without doing so. Python processes created from a common ancestor using `multiprocessing` facilities share a single resource tracker process, and the lifetime of shared memory segments is handled automatically among these processes. Python processes created in any other way will receive their own resource tracker when accessing shared memory with *track* enabled. This will cause the shared memory to be deleted by the resource tracker of the first process that terminates. To avoid this issue, users of `subprocess` or standalone Python processes should set *track* to `False` when there is already another process in place that does the bookkeeping. *track* is ignored on Windows, which has its own tracking and automatically deletes shared memory when all handles to it have been closed.

Distinto en la versión 3.13: Added the *track* parameter.

close()

Close the file descriptor/handle to the shared memory from this instance. **`close()`** should be called once access to the shared memory block from this instance is no longer needed. Depending on operating system, the underlying memory may or may not be freed even if all handles to it have been closed. To ensure proper cleanup, use the `unlink()` method.

unlink()

Delete the underlying shared memory block. This should be called only once per shared memory block regardless of the number of handles to it, even in other processes. **`unlink()`** and `close()` can be called in any order, but trying to access data inside a shared memory block after **`unlink()`** may result in memory access errors, depending on platform.

This method has no effect on Windows, where the only way to delete a shared memory block is to close all handles.

buf

Un *memoryview* del contenido del bloque de memoria compartida.

name

Acceso de solo lectura al nombre único del bloque de memoria compartida.

size

Acceso de solo lectura al tamaño en bytes del bloque de memoria compartida.

El siguiente ejemplo muestra el uso de bajo nivel de instancias de `SharedMemory`:

```
>>> from multiprocessing import shared_memory
>>> shm_a = shared_memory.SharedMemory(create=True, size
>>> type(shm_a.buf)
<class 'memoryview'>
>>> buffer = shm_a.buf
>>> len(buffer)
10
>>> buffer[:4] = bytearray([22, 33, 44, 55])  # Modify m
>>> buffer[4] = 100                           # Modify s
>>> # Attach to an existing shared memory block
>>> shm_b = shared_memory.SharedMemory(shm_a.name)
>>> import array
>>> array.array('b', shm_b.buf[:5])  # Copy the data int
array('b', [22, 33, 44, 55, 100])
>>> shm_b.buf[:5] = b'howdy'  # Modify via shm_b using b
>>> bytes(shm_a.buf[:5])      # Access via shm_a
b'howdy'
>>> shm_b.close()   # Close each SharedMemory instance
>>> shm_a.close()
>>> shm_a.unlink()  # Call unlink only once to release t
```

The following example demonstrates a practical use of the **`SharedMemory`** class with NumPy arrays [https://numpy.org/], accessing the same **`numpy.ndarray`** from two distinct Python shells:

```
>>> # In the first Python interactive shell
>>> import numpy as np
>>> a = np.array([1, 1, 2, 3, 5, 8])  # Start with an ex
>>> from multiprocessing import shared_memory
>>> shm = shared_memory.SharedMemory(create=True, size=a
>>> # Now create a NumPy array backed by shared memory
>>> b = np.ndarray(a.shape, dtype=a.dtype, buffer=shm.bu
>>> b[:] = a[:]  # Copy the original data into shared me
>>> b
array([1, 1, 2, 3, 5, 8])
>>> type(b)
<class 'numpy.ndarray'>
>>> type(a)
<class 'numpy.ndarray'>
>>> shm.name  # We did not specify a name so one was cho
'psm_21467_46075'

>>> # In either the same shell or a new Python shell on
>>> import numpy as np
>>> from multiprocessing import shared_memory
>>> # Attach to the existing shared memory block
>>> existing_shm = shared_memory.SharedMemory(name='psm_
>>> # Note that a.shape is (6,) and a.dtype is np.int64
>>> c = np.ndarray((6,), dtype=np.int64, buffer=existing
>>> c
```

```
array([1, 1, 2, 3, 5, 8])
>>> c[-1] = 888
>>> c
array([  1,   1,   2,   3,   5, 888])

>>> # Back in the first Python interactive shell, b refl
>>> b
array([  1,   1,   2,   3,   5, 888])

>>> # Clean up from within the second Python shell
>>> del c  # Unnecessary; merely emphasizing the array i
>>> existing_shm.close()

>>> # Clean up from within the first Python shell
>>> del b  # Unnecessary; merely emphasizing the array i
>>> shm.close()
>>> shm.unlink()  # Free and release the shared memory b
```

class multiprocessing.managers.SharedMemoryManager([*address*[, *authkey*]])

A subclass of `multiprocessing.managers.BaseManager` which can be used for the management of shared memory blocks across processes.

A call to `start()` on a **`SharedMemoryManager`** instance causes a new process to be started. This new process's sole purpose is to manage the life cycle of all shared memory blocks created through it. To trigger the release of all shared memory blocks managed by that process, call `shutdown()` on the instance. This triggers a `unlink()` call on all of the `SharedMemory` objects managed by that process and then stops the process itself. By creating **`SharedMemory`** instances through a **`SharedMemoryManager`**, we avoid the need to manually track and trigger the freeing of shared memory resources.

Esta clase proporciona métodos para crear y retornar instancias `SharedMemory` y para crear un objeto de tipo lista (`ShareableList`) basados en memoria compartida.

Refer to `BaseManager` for a description of the inherited *address* and *authkey* optional input arguments and how they may be used to connect to an existing **`SharedMemoryManager`** service from other processes.

SharedMemory(*size*)

Create and return a new `SharedMemory` object with the specified *size* in bytes.

ShareableList(*sequence*)

Create and return a new `ShareableList` object, initialized by the values from the input *sequence*.

The following example demonstrates the basic mechanisms of a `SharedMemoryManager`:

```
>>> from multiprocessing.managers import SharedMemoryMar
>>> smm = SharedMemoryManager()
>>> smm.start()  # Start the process that manages the sh
>>> sl = smm.ShareableList(range(4))
>>> sl
ShareableList([0, 1, 2, 3], name='psm_6572_7512')
>>> raw_shm = smm.SharedMemory(size=128)
>>> another_sl = smm.ShareableList('alpha')
>>> another_sl
ShareableList(['a', 'l', 'p', 'h', 'a'], name='psm_6572_
>>> smm.shutdown()  # Calls unlink() on sl, raw_shm, and
```

The following example depicts a potentially more convenient pattern for using `SharedMemoryManager` objects via the `with` statement to ensure that all shared memory blocks are released after they are no longer needed:

```
>>> with SharedMemoryManager() as smm:
...     sl = smm.ShareableList(range(2000))
...     # Divide the work among two processes, storing p
...     p1 = Process(target=do_work, args=(sl, 0, 1000))
...     p2 = Process(target=do_work, args=(sl, 1000, 200
...     p1.start()
...     p2.start()  # A multiprocessing.Pool might be mo
...     p1.join()
...     p2.join()   # Wait for all work to complete in b
...     total_result = sum(sl)  # Consolidate the partia
```

When using a `SharedMemoryManager` in a `with` statement, the shared memory blocks created using that manager are all released when the `with` statement's code block finishes execution.

class multiprocessing.shared_memory.ShareableList(*sequence=None*, *, *name=None*)

Provide a mutable list-like object where all values stored within are stored in a shared memory block. This constrains storable values to the following built-in data types:

- `int` (signed 64-bit)
- `float`
- `bool`
- `str` (less than 10M bytes each when encoded as UTF-8)
- `bytes` (less than 10M bytes each)
- `None`

It also notably differs from the built-in `list` type in that these lists can not change their overall length (i.e. no `append()`, `insert()`, etc.) and do not support the dynamic creation of new `ShareableList` instances via slicing.

sequence is used in populating a new `ShareableList` full of values. Set to `None` to instead attach to an already existing `ShareableList` by its unique shared memory name.

name is the unique name for the requested shared memory, as described in the definition for `SharedMemory`. When attaching to an existing `ShareableList`, specify its shared memory block's unique name while leaving *sequence* set to `None`.

Nota

A known issue exists for `bytes` and `str` values. If they end with `\x00` nul bytes or characters, those may be *silently stripped* when fetching them by index from the **`ShareableList`**. This `.rstrip(b'\x00')` behavior is considered a bug and may go away in the future. See gh-106939 [https://github.com/python/cpython/issues/106939].

Para aplicaciones donde eliminar espacios nulos finales es un problema, solucionarlo añadiendo siempre incondicionalmente un byte extra que no sea 0 al final de dichos valores al almacenar y eliminándolo incondicionalmente al recuperarlos.

```
>>> from multiprocessing import shared_memory
>>> nul_bug_demo = shared_memory.ShareableList(['?\
>>> nul_bug_demo[0]
'?'
>>> nul_bug_demo[1]
b'\x03\x02\x01'
>>> nul_bug_demo.shm.unlink()
>>> padded = shared_memory.ShareableList(['?\x00\x0
>>> padded[0][:-1]
'?\x00'
>>> padded[1][:-1]
b'\x03\x02\x01\x00\x00\x00'
>>> padded.shm.unlink()
```

count(*value*)

Return the number of occurrences of *value*.

index(*value*)

Return first index position of *value*. Raise `ValueError` if *value* is not present.

format

Atributo de solo lectura que contiene el formato de empaquetado `struct` utilizado por todos los valores almacenados actualmente.

shm

La instancia de **SharedMemory** donde se almacenan los valores.

El siguiente ejemplo muestra el uso básico de una instancia **ShareableList**:

```
>>> from multiprocessing import shared_memory
>>> a = shared_memory.ShareableList(['howdy', b'HoWdY',
>>> [ type(entry) for entry in a ]
[<class 'str'>, <class 'bytes'>, <class 'float'>, <class
>>> a[2]
-273.154
>>> a[2] = -78.5
>>> a[2]
-78.5
>>> a[2] = 'dry ice'  # Changing data types is supporte
>>> a[2]
'dry ice'
>>> a[2] = 'larger than previously allocated storage spa
Traceback (most recent call last):
  ...
ValueError: exceeds available storage for existing str
>>> a[2]
'dry ice'
>>> len(a)
7
>>> a.index(42)
6
>>> a.count(b'howdy')
0
>>> a.count(b'HoWdY')
1
>>> a.shm.close()
>>> a.shm.unlink()
>>> del a  # Use of a ShareableList after call to unlin
```

El siguiente ejemplo muestra cómo uno, dos o muchos procesos pueden acceder al mismo **ShareableList** al proporcionar el nombre del bloque de memoria compartida detrás de él:

```
>>> b = shared_memory.ShareableList(range(5))         #
>>> c = shared_memory.ShareableList(name=b.shm.name)  #
>>> c
ShareableList([0, 1, 2, 3, 4], name='...')
>>> c[-1] = -999
>>> b[-1]
-999
>>> b.shm.close()
>>> c.shm.close()
>>> c.shm.unlink()
```

The following examples demonstrates that **`ShareableList`** (and underlying **`SharedMemory`**) objects can be pickled and unpickled if needed. Note, that it will still be the same shared object. This happens, because the deserialized object has the same unique name and is just attached to an existing object with the same name (if the object is still alive):

```
>>> import pickle
>>> from multiprocessing import shared_memory
>>> sl = shared_memory.ShareableList(range(10))
>>> list(sl)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> deserialized_sl = pickle.loads(pickle.dumps(sl))
>>> list(deserialized_sl)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> sl[0] = -1
>>> deserialized_sl[1] = -2
>>> list(sl)
[-1, -2, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(deserialized_sl)
[-1, -2, 2, 3, 4, 5, 6, 7, 8, 9]

>>> sl.shm.close()
>>> sl.shm.unlink()
```

El paquete `concurrent`

This package contains the following modules:

- `concurrent.futures` – Lanzamiento de tareas paralelas
- `concurrent.interpreters` – Multiple interpreters in the same process

`concurrent.futures` — Launching parallel tasks

Added in version 3.2.

Source code: Lib/concurrent/futures/thread.py [https://github.com/python/cpython/tree/3.14/Lib/concurrent/futures/thread.py], Lib/concurrent/futures/process.py [https://github.com/python/cpython/tree/3.14/Lib/concurrent/futures/process.py], and Lib/concurrent/futures/interpreter.py [https://github.com/python/cpython/tree/3.14/Lib/concurrent/futures/interpreter.py]

The `concurrent.futures` module provides a high-level interface for asynchronously executing callables.

The asynchronous execution can be performed with threads, using `ThreadPoolExecutor` or `InterpreterPoolExecutor`, or separate processes, using `ProcessPoolExecutor`. Each implements the same interface, which is defined by the abstract `Executor` class.

`concurrent.futures.Future` must not be confused with `asyncio.Future`, which is designed for use with `asyncio` tasks and coroutines. See the asyncio's Future documentation for a detailed comparison of the two.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

Objetos ejecutores

class concurrent.futures.Executor

Una clase abstracta que proporciona métodos para ejecutar llamadas de forma asincrónica. No debe ser utilizada directamente, sino a través de sus subclases.

submit(*fn*, /, **args*, ***kwargs*)

Programa la invocación de *fn*, que será ejecutada como `fn(*args, **kwargs)` y retorna un objeto **`Future`** que representa la ejecución del invocable.

```
with ThreadPoolExecutor(max_workers=1) as execu
    future = executor.submit(pow, 323, 1235)
    print(future.result())
```

map(*fn*, **iterables*, *timeout=None*, *chunksize=1*, *buffersize=None*)

Similar to **`map(fn, *iterables)`** except:

- The *iterables* are collected immediately rather than lazily, unless a *buffersize* is specified to limit the number of submitted tasks whose results have not yet been yielded. If the buffer is full, iteration over the *iterables* pauses until a result is yielded from the buffer.
- *fn* is executed asynchronously and several calls to *fn* may be made concurrently.

El iterador retornado lanza **`TimeoutError`** si **`__next__()`** es llamado y el resultado no está disponible luego de *timeout* segundos luego de la llamada original a **`Executor.map()`**. *timeout* puede ser un int o un float. Si no se provee un *timeout* o es `None`, no hay limite de espera.

If a *fn* call raises an exception, then that exception will be raised when its value is retrieved from the iterator.

When using **`ProcessPoolExecutor`**, this method chops *iterables* into a number of chunks which it submits to the pool as separate tasks. The (approximate) size of these chunks can be specified by setting *chunksize* to a positive integer. For very long iterables, using a large value for *chunksize* can significantly improve performance compared to the default size of 1. With `ThreadPoolExecutor` and `InterpreterPoolExecutor`, *chunksize* has no effect.

Distinto en la versión 3.5: Added the *chunksize* parameter.

Distinto en la versión 3.14: Added the *buffersize* parameter.

shutdown(*wait=True*, ***, *cancel_futures=False*)

Indica al ejecutor que debe liberar todos los recursos que está utilizando cuando los futuros actualmente pendientes de ejecución finalicen. Las llamadas a `Executor.submit()` y `Executor.map()` realizadas después del apagado lanzarán `RuntimeError`.

Si *wait* es `True` este método no retornará hasta que todos los futuros pendientes hayan terminado su ejecución y los recursos asociados al ejecutor hayan sido liberados. Si *wait* es `False`, este método retornará de inmediato y los recursos asociados al ejecutor se liberarán cuando todos los futuros asociados hayan finalizado su ejecución. Independientemente del valor de *wait*, el programa Python entero no finalizará hasta que todos los futuros pendientes hayan finalizado su ejecución.

Si *cancel_futures* es `True`, este método cancelará todos los futuros pendientes que el ejecutor no ha comenzado a ejecutar. Los futuros que se completen o estén en ejecución no se cancelarán, independientemente del valor de *cancel_futures*.

Si tanto *cancel_futures* como *wait* son `True`, todos los futuros que el ejecutor ha comenzado a ejecutar se completarán antes de que regrese este método. Los futuros restantes se cancelan.

You can avoid having to call this method explicitly if you use the executor as a context manager via the **`with`** statement, which will shutdown the **`Executor`** (waiting as if **`Executor.shutdown()`** were called with *wait* set to `True`):

```
import shutil
with ThreadPoolExecutor(max_workers=4) as e:
    e.submit(shutil.copy, 'src1.txt', 'dest1.tx
    e.submit(shutil.copy, 'src2.txt', 'dest2.tx
    e.submit(shutil.copy, 'src3.txt', 'dest3.tx
    e.submit(shutil.copy, 'src4.txt', 'dest4.tx
```

Distinto en la versión 3.9: Agregado *cancel_futures*.

ThreadPoolExecutor

`ThreadPoolExecutor` es una subclase de **`Executor`** que usa un grupo de hilos para ejecutar llamadas de forma asincrónica.

Pueden ocurrir bloqueos mutuos cuando la llamada asociada a un **`Future`** espera el resultado de otro **`Future`**. Por ejemplo:

```
import time
def wait_on_b():
    time.sleep(5)
    print(b.result())  # b will never complete because i
    return 5

def wait_on_a():
    time.sleep(5)
    print(a.result())  # a will never complete because i
    return 6


executor = ThreadPoolExecutor(max_workers=2)
a = executor.submit(wait_on_b)
b = executor.submit(wait_on_a)
```

Y:

```
def wait_on_future():
    f = executor.submit(pow, 5, 2)
    # This will never complete because there is only one
    # it is executing this function.
    print(f.result())

executor = ThreadPoolExecutor(max_workers=1)
future = executor.submit(wait_on_future)
# Note: calling future.result() would also cause a deadl
# the single worker thread is already waiting for wait_c
```

class concurrent.futures.ThreadPoolExecutor(*max_workers=None*, *thread_name_prefix=''*, *initializer=None*, *initargs=()*)

Subclase de **`Executor`** que utiliza un grupo de hilos de *max_workers* como máximo para ejecutar llamadas de forma asincrónica.

Todos los hilos que se hayan puesto en cola en `ThreadPoolExecutor` se unirán antes de que el intérprete pueda salir. Tenga en cuenta que el controlador de salida que hace esto se ejecuta *antes* de cualquier controlador de salida añadido mediante `atexit`. Esto significa que las excepciones en el hilo principal deben ser capturadas y manejadas para indicar a los hilos que salgan correctamente. Por esta razón, se recomienda no utilizar `ThreadPoolExecutor` para tareas de larga duración.

initializer es un invocable opcional que es llamado al comienzo de cada hilo de trabajo; *initargs* es una tupla de argumentos pasados al inicializador. Si el *initializer* lanza una excepción, todos los trabajos actualmente pendientes lanzarán **`BrokenThreadPool`**, así como cualquier intento de enviar más trabajos al grupo.

Distinto en la versión 3.5: Si *max_workers* es `None` o no es especificado, se tomará por defecto el número de procesadores de la máquina, multiplicado por `5`, asumiendo que **`ThreadPoolExecutor`** a menudo se utiliza para paralelizar E/S en lugar de trabajo de CPU y que el numero de trabajadores debe ser mayor que el número de trabajadores para **`ProcessPoolExecutor`**.

Distinto en la versión 3.6: Added the *thread_name_prefix* parameter to allow users to control the **`threading.Thread`** names for worker threads created by the pool for easier debugging.

Distinto en la versión 3.7: Se agregaron los argumentos *initializer* y *initargs*.

Distinto en la versión 3.8: El valor predeterminado de *max_workers* fue reemplazado por `min(32, os.cpu_count() + 4)`. Este valor predeterminado conserva al menos 5 trabajadores para las tareas vinculadas de E/S. Utiliza como máximo 32 núcleos de CPU para tareas vinculadas a la CPU que liberan el GIL. Y evita utilizar recursos muy grandes implícitamente en máquinas de muchos núcleos.

ThreadPoolExecutor ahora también reutiliza hilos inactivos antes de crear *max_workers* hilos de trabajo.

Distinto en la versión 3.13: Default value of *max_workers* is changed to `min(32, (os.process_cpu_count() or 1) + 4)`.

Ejemplo de ThreadPoolExecutor

```
import concurrent.futures
import urllib.request

URLS = ['http://www.foxnews.com/',
        'http://www.cnn.com/',
        'http://europe.wsj.com/',
        'http://www.bbc.co.uk/',
        'http://nonexistent-subdomain.python.org/']

# Retrieve a single page and report the URL and contents
```

```
def load_url(url, timeout):
    with urllib.request.urlopen(url, timeout=timeout) as
        return conn.read()

# We can use a with statement to ensure threads are clea
with concurrent.futures.ThreadPoolExecutor(max_workers=5
    # Start the load operations and mark each future wit
    future_to_url = {executor.submit(load_url, url, 60):
    for future in concurrent.futures.as_completed(future
        url = future_to_url[future]
        try:
            data = future.result()
        except Exception as exc:
            print('%r generated an exception: %s' % (url
        else:
            print('%r page is %d bytes' % (url, len(data
```

InterpreterPoolExecutor

Added in version 3.14.

The **`InterpreterPoolExecutor`** class uses a pool of interpreters to execute calls asynchronously. It is a **`ThreadPoolExecutor`** subclass, which means each worker is running in its own thread. The difference here is that each worker has its own interpreter, and runs each task using that interpreter.

The biggest benefit to using interpreters instead of only threads is true multi-core parallelism. Each interpreter has its own Global Interpreter Lock, so code running in one interpreter can run on one CPU core, while code in another interpreter runs unblocked on a different core.

The tradeoff is that writing concurrent code for use with multiple interpreters can take extra effort. However, this is because it forces you to be deliberate about how and when interpreters interact, and to be explicit about what data is shared between interpreters. This results in several benefits that help balance the extra effort, including true multi-core parallelism, For example, code written this way can make it easier to reason about concurrency. Another major benefit is that you don't have to deal with several of the big pain points of using threads, like race conditions.

Each worker's interpreter is isolated from all the other interpreters. «Isolated» means each interpreter has its own runtime state and operates completely independently. For example, if you redirect **`sys.stdout`** in one interpreter, it will not be automatically redirected to any other interpreter. If you import a module in one interpreter, it is not automatically imported in any other. You would need to import the module separately in interpreter where you need it. In fact, each module imported in an interpreter is a completely separate object from the same module in a different interpreter, including **`sys`**, **`builtins`**, and even `__main__`.

Isolation means a mutable object, or other data, cannot be used by more than one interpreter at the same time. That effectively means interpreters cannot actually share such objects or data. Instead, each interpreter must have its own copy, and you will have to synchronize any changes between the copies manually. Immutable objects and data, like the builtin singletons, strings, and tuples of immutable objects, don't have these limitations.

Communicating and synchronizing between interpreters is most effectively done using dedicated tools, like those proposed in **PEP 734** [https://peps.python.org/pep-0734/]. One less efficient alternative is to serialize with **`pickle`** and then send the bytes over a shared **`socket`** or **`pipe`**.

class concurrent.futures.InterpreterPoolExecutor(*max_workers=None, thread_name_prefix='', initializer=None, initargs=()*)

A **`ThreadPoolExecutor`** subclass that executes calls asynchronously using a pool of at most *max_workers* threads. Each thread runs tasks in its own interpreter. The worker interpreters are isolated from each other, which means each has its own runtime state and that they can't share any mutable objects or other data. Each interpreter has its own Global Interpreter Lock, which means code run with this executor has true multi-core parallelism.

The optional *initializer* and *initargs* arguments have the same meaning as for `ThreadPoolExecutor`: the initializer is run when each worker is created, though in this case it is run in the worker's interpreter. The executor serializes the *initializer* and *initargs* using `pickle` when sending them to the worker's interpreter.

Nota

The executor may replace uncaught exceptions from *initializer* with `ExecutionFailed`.

Other caveats from parent `ThreadPoolExecutor` apply here.

`submit()` and `map()` work like normal, except the worker serializes the callable and arguments using `pickle` when sending them to its interpreter. The worker likewise serializes the return value when sending it back.

When a worker's current task raises an uncaught exception, the worker always tries to preserve the exception as-is. If that is successful then it also sets the `__cause__` to a corresponding `ExecutionFailed` instance, which contains a summary of the original exception. In the uncommon case that the worker is not able to preserve the original as-is then it directly preserves the corresponding `ExecutionFailed` instance instead.

ProcessPoolExecutor

La clase `ProcessPoolExecutor` es una subclase `Executor` que usa un grupo de procesos para ejecutar llamadas de forma asíncrona. `ProcessPoolExecutor` usa el módulo `multiprocessing`, que le permite eludir el Global Interpreter Lock pero también significa que solo los objetos seleccionables pueden ser ejecutados y retornados.

El módulo `__main__` debe ser importable por los subprocesos trabajadores. Esto significa que `ProcessPoolExecutor` no funcionará en el intérprete interactivo.

Llamar a métodos de `Executor` o `Future` desde el invocable enviado a `ProcessPoolExecutor` resultará en bloqueos mutuos.

Note that the restrictions on functions and arguments needing to picklable as per `multiprocessing.Process` apply when using `submit()` and `map()` on a `ProcessPoolExecutor`. A function defined in a REPL or a lambda should not be expected to work.

class concurrent.futures.ProcessPoolExecutor(*max_workers=None*, *mp_context=None*, *initializer=None*, *initargs=()*, *max_tasks_per_child=None*)

An `Executor` subclass that executes calls asynchronously using a pool of at most *max_workers* processes. If *max_workers* is `None` or not given, it will default to `os.process_cpu_count()`. If *max_workers* is less than or equal to `0`, then a `ValueError` will be raised. On Windows, *max_workers* must be less than or equal to `61`. If it is not then `ValueError` will be raised. If *max_workers* is `None`, then the default chosen will be at most `61`, even if more processors are available. *mp_context* can be a `multiprocessing` context or `None`. It will be used to launch the workers. If *mp_context* is `None` or not given, the default `multiprocessing` context is used. See Contextos y métodos de inicio.

initializer es un invocable opcional que es llamado al comienzo de cada proceso trabajador; *initargs* es una tupla de argumentos pasados al inicializador. Si el *initializer* lanza una excepción, todos los trabajos actualmente pendientes lanzarán `BrokenProcessPool`, así como cualquier intento de enviar más trabajos al grupo.

max_tasks_per_child es un argumento opcional que especifica el número máximo de tareas que un único proceso puede ejecutar antes de salir y ser reemplazado por un nuevo proceso de trabajo. Por defecto, *max_tasks_per_child* es `None`, lo que significa que los procesos de trabajo vivirán tanto como el pool. Cuando se especifica un máximo, el método de inicio de multiprocesamiento «spawn» se utilizará por defecto en ausencia de un parámetro *mp_context*. Esta característica es incompatible con el método de inicio «fork».

Nota

Bugs have been reported when using the *max_tasks_per_child* feature that can result in the **`ProcessPoolExecutor`** hanging in some circumstances. Follow its eventual resolution in gh-115634 [https://github.com/python/cpython/issues/115634].

Distinto en la versión 3.3: When one of the worker processes terminates abruptly, a `BrokenProcessPool` error is now raised. Previously, behaviour was undefined but operations on the executor or its futures would often freeze or deadlock.

Distinto en la versión 3.7: El argumento *mp_context* se agregó para permitir a los usuarios controlar el método de iniciación para procesos de trabajo creados en el grupo.

Se agregaron los argumentos *initializer* y *initargs*.

Distinto en la versión 3.11: El argumento *max_tasks_per_child* se agregó para permitir a los usuarios controlar el tiempo de vida para procesos de trabajo creados en el grupo.

Distinto en la versión 3.12: En sistemas POSIX, si su aplicación tiene múltiples hilos y el contexto `multiprocessing` utiliza el método de inicio `"fork"`: La función `os.fork()` llamada internamente para generar trabajadores puede lanzar un mensaje `DeprecationWarning`. Pase un *mp_context* configurado para utilizar un método de inicio diferente. Consulte la documentación de **`os.fork()`** para más información.

Distinto en la versión 3.13: *max_workers* uses `os.process_cpu_count()` by default, instead of `os.cpu_count()`.

Distinto en la versión 3.14: The default process start method

(see Contextos y métodos de inicio) changed away from *fork*. If you require the *fork* start method for **`ProcessPoolExecutor`** you must explicitly pass `mp_context=multiprocessing.get_context("fork")`.

terminate_workers()

Attempt to terminate all living worker processes immediately by calling **`Process.terminate`** on each of them. Internally, it will also call **`Executor.shutdown()`** to ensure that all other resources associated with the executor are freed.

After calling this method the caller should no longer submit tasks to the executor.

Added in version 3.14.

kill_workers()

Attempt to kill all living worker processes immediately by calling **`Process.kill`** on each of them. Internally, it will also call **`Executor.shutdown()`** to ensure that all other resources associated with the executor are freed.

After calling this method the caller should no longer submit tasks to the executor.

Added in version 3.14.

Ejemplo de *ProcessPoolExecutor*

```
import concurrent.futures
import math

PRIMES = [
    112272535095293,
    112582705942171,
    112272535095293,
    115280095190773,
    115797848077099,
    1099726899285419]

def is_prime(n):
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False

    sqrt_n = int(math.floor(math.sqrt(n)))
    for i in range(3, sqrt_n + 1, 2):
        if n % i == 0:
            return False
    return True

def main():
    with concurrent.futures.ProcessPoolExecutor() as exe
        for number, prime in zip(PRIMES, executor.map(is
            print('%d is prime: %s' % (number, prime))

if __name__ == '__main__':
    main()
```

Objetos futuro

La clase `Future` encapsula la ejecución asincrónica del invocable. Las instancias de `Future` son creadas por `Executor.submit()`.

class concurrent.futures.Future

Encapsula la ejecución asincrónica del invocable. Las instancias de `Future` son creadas por `Executor.submit()` y no deberían ser creadas directamente, excepto para pruebas.

cancel()

Intenta cancelar la llamada. Si el invocable está siendo ejecutado o ha finalizado su ejecución y no puede ser cancelado el método retornará `False`, de lo contrario la llamada será cancelada y el método retornará `True`.

cancelled()

Retorna `True` si la llamada fue cancelada exitosamente.

running()

Retorna `True` si la llamada está siendo ejecutada y no puede ser cancelada.

done()

Retorna `True` si la llamada fue cancelada exitosamente o terminó su ejecución.

result(*timeout=None*)

Retorna el valor retornado por la llamada. Si la llamada aún no se ha completado, este método esperará hasta *timeout* segundos. Si la llamada no se ha completado en *timeout* segundos, entonces se lanzará un **`TimeoutError`**. *timeout* puede ser un int o un float. Si *timeout* no se especifica o es `None`, no hay límite para el tiempo de espera.

Si el futuro es cancelado antes de finalizar su ejecución, **`CancelledError`** será lanzada.

Si la llamada lanzó una excepción, este método lanzará la misma excepción.

exception(*timeout=None*)

Retorna la excepción lanzada por la llamada. Si la llamada aún no se ha completado, este método esperará hasta *timeout* segundos. Si la llamada no se ha completado en *timeout* segundos, entonces se lanzará un **`TimeoutError`**. *timeout* puede ser un int o un float. Si *timeout* no se especifica o es `None`, no hay límite para el tiempo de espera.

Si el futuro es cancelado antes de finalizar su ejecución, **`CancelledError`** será lanzada.

Si la llamada es completada sin excepciones, se retornará `` `None ``.

add_done_callback(*fn*)

Asocia el invocable *fn* al futuro. *fn* va a ser llamada, con el futuro como su único argumento, cuando el futuro sea cancelado o finalice su ejecución.

Los invocables agregados se llaman en el orden en que se agregaron y siempre se llaman en un hilo que pertenece al proceso que los agregó. Si el invocable genera una subclase **`Exception`**, se registrará y se ignorará. Si el invocable genera una subclase **`BaseException`**, el comportamiento no está definido.

Si el futuro ya ha finalizado su ejecución o fue cancelado, *fn* retornará inmediatamente.

Los siguientes métodos de **`Future`** están pensados para ser usados en pruebas unitarias e implementaciones de **`Executor`**.

set_running_or_notify_cancel()

Este método sólo debe ser llamado en implementaciones de **`Executor`** antes de ejecutar el trabajo asociado al **`Future`** y por las pruebas unitarias.

Si el método retorna `False` entonces el **`Future`** fue cancelado, es decir, **`Future.cancel()`** fue llamado y retornó `True`. Cualquier hilo que esté esperando a que el **`Future`** se complete (es decir, a través de **`as_completed()`** o **`wait()`**) será despertado.

Si el método retorna `True` entonces el **`Future`** no se ha cancelado y se ha puesto en estado de ejecución, es decir, las llamadas a **`Future.running`()`** retornarán `True`.

Este método solo puede ser llamado una sola vez y no puede ser llamado luego de haber llamado a **`Future.set_result()`** o a **`Future.set_exception()`**.

set_result(*result*)

Establece *result* como el resultado del trabajo asociado al **`Future`**.

Este método solo debe ser usado por implementaciones de **`Executor`** y pruebas unitarias.

Distinto en la versión 3.8: Este método lanza **`concurrent.futures.InvalidStateError`** si **`Future`** ya ha finalizado su ejecución.

set_exception(*exception*)

Establece *exception*, subclase de **`Exception`**, como el resultado del trabajo asociado al **`Future`**.

Este método solo debe ser usado por implementaciones de **`Executor`** y pruebas unitarias.

Distinto en la versión 3.8: Este método lanza **`concurrent.futures.InvalidStateError`** si **`Future`** ya ha finalizado su ejecución.

Funciones del módulo

concurrent.futures.wait(*fs*, *timeout=None*, *return_when=ALL_COMPLETED*)

Espera a que se completen las instancias **`Future`** (posiblemente creadas por diferentes instancias **`Executor`**) dadas por *fs*. Los futuros duplicados dados a *fs* se eliminan y sólo se devolverán una vez. Retorna una tupla nombrada de 2 conjuntos. El primer conjunto, llamado `done`, contiene los futuros que se han completado (futuros finalizados o cancelados) antes de que se complete la espera. El segundo conjunto, llamado `not_done`, contiene los futuros que no se completaron (futuros pendientes o en ejecución).

El argumento *timeout* puede ser usado para controlar la espera máxima en segundos antes de retornar. *timeout* puede ser un int o un float. Si *timeout* no es especificado o es `None`, no hay limite en el tiempo de espera.

return_when indica cuando debe retornar esta función. Debe ser alguna de las siguientes constantes:

Constante	**Descripción**
concurrent.futures.FIRST_COMPLETED	La función retornará cuando cualquier futuro finalice o sea cancelado.
concurrent.futures.FIRST_EXCEPTION	The function will return when any future finishes by raising an exception. If no future raises an exception then it is equivalent to **`ALL_COMPLETED`**.
concurrent.futures.ALL_COMPLETED	La función retornará cuando todos los futuros finalicen o sean cancelados.

concurrent.futures.as_completed(*fs*, *timeout=None*)

Retorna un iterador sobre las instancias de **`Future`** (posiblemente creadas por distintas instancias de **`Executor`**) dadas por *fs* que produce futuros a medida que van finalizando (normalmente o cancelados). Cualquier futuro dado por *fs* que esté duplicado será retornado una sola vez. Los futuros que hayan finalizado antes de la llamada a **`as_completed()`** serán entregados primero. El iterador retornado lanzará **`TimeoutError`** si **`__next__()`** es llamado y el resultado no está disponible luego de *timeout* segundos a partir de la llamada original a **`as_completed()`**. *timeout* puede ser un int o un float. Si *timeout* no es especificado o es `None`, no hay límite en el tiempo de espera.

Ver también

PEP 3148 [https://peps.python.org/pep-3148/] – futuros - ejecutar cómputos asincrónicamente

La propuesta que describe esta propuesta de inclusión en la biblioteca estándar de Python.

Clases de Excepciones

exception concurrent.futures.CancelledError

Lanzada cuando un futuro es cancelado.

exception concurrent.futures.TimeoutError

Un alias obsoleto de `TimeoutError`, lanzado cuando una operación en un futuro excede el tiempo de espera dado.

Distinto en la versión 3.11: Esta clase se convirtió en un alias de `TimeoutError`.

exception concurrent.futures.BrokenExecutor

Derivada de `RuntimeError`, esta excepción es lanzada cuando un ejecutor se encuentra corrupto por algún motivo y no puede ser utilizado para enviar o ejecutar nuevas tareas.

Added in version 3.7.

exception concurrent.futures.InvalidStateError

Lanzada cuando una operación es realizada sobre un futuro que no permite dicha operación en el estado actual.

Added in version 3.8.

exception concurrent.futures.thread.BrokenThreadPool

Derived from `BrokenExecutor`, this exception class is raised when one of the workers of a `ThreadPoolExecutor` has failed initializing.

Added in version 3.7.

exception concurrent.futures.interpreter.BrokenInterpreterPool

Derived from **`BrokenThreadPool`**, this exception class is raised when one of the workers of a **`InterpreterPoolExecutor`** has failed initializing.

Added in version 3.14.

exception concurrent.futures.process.BrokenProcessPool

Derived from **`BrokenExecutor`** (formerly **`RuntimeError`**), this exception class is raised when one of the workers of a **`ProcessPoolExecutor`** has terminated in a non-clean fashion (for example, if it was killed from the outside).

Added in version 3.3.

`concurrent.interpreters` — Multiple interpreters in the same process

Added in version 3.14.

Source code: Lib/concurrent/interpreters [https://github.com/python/cpython/tree/3.14/Lib/concurrent/interpreters]

The `concurrent.interpreters` module constructs higher-level interfaces on top of the lower level `_interpreters` module.

The module is primarily meant to provide a basic API for managing interpreters (AKA «subinterpreters») and running things in them. Running mostly involves switching to an interpreter (in the current thread) and calling a function in that execution context.

For concurrency, interpreters themselves (and this module) don't provide much more than isolation, which on its own isn't useful. Actual concurrency is available separately through `threads` See below

Ver también

`InterpreterPoolExecutor`
Combines threads with interpreters in a familiar interface.

Aislamiento de módulos de extensión
How to update an extension module to support multiple interpreters.

PEP 554 [https://peps.python.org/pep-0554/]

PEP 734 [https://peps.python.org/pep-0734/]

PEP 684 [https://peps.python.org/pep-0684/]

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

Key details

Before we dive in further, there are a small number of details to keep in mind about using multiple interpreters:

- isolated, by default
- no implicit threads
- not all PyPI packages support use in multiple interpreters yet

Introduction

An «interpreter» is effectively the execution context of the Python runtime. It contains all of the state the runtime needs to execute a program. This includes things like the import state and builtins. (Each thread, even if there's only the main thread, has some extra runtime state, in addition to the current interpreter, related to the current exception and the bytecode eval loop.)

The concept and functionality of the interpreter have been a part of Python since version 2.2, but the feature was only available through the C-API and not well known, and the isolation was relatively incomplete until version 3.12.

Multiple Interpreters and Isolation

A Python implementation may support using multiple interpreters in the same process. CPython has this support. Each interpreter is effectively isolated from the others (with a limited number of carefully managed process-global exceptions to the rule).

That isolation is primarily useful as a strong separation between distinct logical components of a program, where you want to have careful control of how those components interact.

Nota

Interpreters in the same process can technically never be strictly isolated from one another since there are few restrictions on memory access within the same process. The Python runtime makes a best effort at isolation but extension modules may easily violate that. Therefore, do not use multiple interpreters in security-sensitive situations, where they shouldn't have access to each other's data.

Running in an Interpreter

Running in a different interpreter involves switching to it in the current thread and then calling some function. The runtime will execute the function using the current interpreter's state. The `concurrent.interpreters` module provides a basic API for creating and managing interpreters, as well as the switch-and-call operation.

No other threads are automatically started for the operation. There is a helper for that though. There is another dedicated helper for calling the builtin `exec()` in an interpreter.

When `exec()` (or `eval()`) are called in an interpreter, they run using the interpreter's `__main__` module as the «globals» namespace. The same is true for functions that aren't associated with any module. This is the same as how scripts invoked from the command-line run in the `__main__` module.

Concurrency and Parallelism

As noted earlier, interpreters do not provide any concurrency on their own. They strictly represent the isolated execution context the runtime will use *in the current thread*. That isolation makes them similar to processes, but they still enjoy in-process efficiency, like threads.

All that said, interpreters do naturally support certain flavors of concurrency. There's a powerful side effect of that isolation. It enables a different approach to concurrency than you can take with async or threads. It's a similar concurrency model to CSP or the actor model, a model which is relatively easy to reason about.

You can take advantage of that concurrency model in a single thread, switching back and forth between interpreters, Stackless-style. However, this model is more useful when you combine interpreters with multiple threads. This mostly involves starting a new thread, where you switch to another interpreter and run what you want there.

Each actual thread in Python, even if you're only running in the main thread, has its own *current* execution context. Multiple threads can use the same interpreter or different ones.

At a high level, you can think of the combination of threads and interpreters as threads with opt-in sharing.

As a significant bonus, interpreters are sufficiently isolated that they do not share the GIL, which means combining threads with multiple interpreters enables full multi-core parallelism. (This has been the case since Python 3.12.)

Communication Between Interpreters

In practice, multiple interpreters are useful only if we have a way to communicate between them. This usually involves some form of message passing, but can even mean sharing data in some carefully managed way.

With this in mind, the `concurrent.interpreters` module provides a `queue.Queue` implementation, available through `create_queue()`.

«Sharing» Objects

Any data actually shared between interpreters loses the thread-safety provided by the GIL. There are various options for dealing with this in extension modules. However, from Python code the lack of thread-safety means objects can't actually be shared, with a few exceptions. Instead, a copy must be created, which means mutable objects won't stay in sync.

By default, most objects are copied with `pickle` when they are passed to another interpreter. Nearly all of the immutable builtin objects are either directly shared or copied efficiently. For example:

- `None`
- `bool` (`True` and `False`)
- `bytes`
- `str`
- `int`
- `float`
- `tuple` (of similarly supported objects)

There is a small number of Python types that actually share mutable data between interpreters:

- `memoryview`
- `Queue`

Reference

This module defines the following functions:

concurrent.interpreters.list_all()

Return a `list` of `Interpreter` objects, one for each existing interpreter.

concurrent.interpreters.get_current()

Return an `Interpreter` object for the currently running interpreter.

concurrent.interpreters.get_main()

Return an `Interpreter` object for the main interpreter. This is the interpreter the runtime created to run the REPL or the script given at the command-line. It is usually the only one.

concurrent.interpreters.create()

Initialize a new (idle) Python interpreter and return a `Interpreter` object for it.

concurrent.interpreters.create_queue()

Initialize a new cross-interpreter queue and return a `Queue` object for it.

Interpreter objects

class concurrent.interpreters.Interpreter(*id*)

A single interpreter in the current process.

Generally, **`Interpreter`** shouldn't be called directly. Instead, use `create()` or one of the other module functions.

id

(read-only)

The underlying interpreter's ID.

whence

(read-only)

A string describing where the interpreter came from.

is_running()

Return `True` if the interpreter is currently executing code in its **`__main__`** module and `False` otherwise.

close()

Finalize and destroy the interpreter.

prepare_main(*ns=None*, ***kwargs*)

Bind objects in the interpreter's **`__main__`** module.

Some objects are actually shared and some are copied efficiently, but most are copied via `pickle`. See «Sharing» Objects.

exec(*code*, */*, *dedent=True*)

Run the given source code in the interpreter (in the current thread).

call(*callable*, */*, **args*, ***kwargs*)

Return the result of calling running the given function in the interpreter (in the current thread).

call_in_thread(*callable*, */*, **args*, ***kwargs*)

Run the given function in the interpreter (in a new thread).

Exceptions

exception concurrent.interpreters.InterpreterError

This exception, a subclass of `Exception`, is raised when an interpreter-related error happens.

exception concurrent.interpreters.InterpreterNotFoundError

This exception, a subclass of `InterpreterError`, is raised when the targeted interpreter no longer exists.

exception concurrent.interpreters.ExecutionFailed

This exception, a subclass of `InterpreterError`, is raised when the running code raised an uncaught exception.

excinfo

A basic snapshot of the exception raised in the other interpreter.

exception concurrent.interpreters.NotShareableError

This exception, a subclass of `TypeError`, is raised when an object cannot be sent to another interpreter.

Communicating Between Interpreters

class concurrent.interpreters.Queue(*id*)

A wrapper around a low-level, cross-interpreter queue, which implements the `queue.Queue` interface. The underlying queue can only be created through `create_queue()`.

Some objects are actually shared and some are copied efficiently, but most are copied via `pickle`. See «Sharing» Objects.

id

(read-only)

The queue's ID.

exception concurrent.interpreters.QueueEmptyError

This exception, a subclass of `queue.Empty`, is raised from `Queue.get()` and `Queue.get_nowait()` when the queue is empty.

exception concurrent.interpreters.QueueFullError

This exception, a subclass of `queue.Full`, is raised from `Queue.put()` and `Queue.put_nowait()` when the queue is full.

Basic usage

Creating an interpreter and running code in it:

```
from concurrent import interpreters

interp = interpreters.create()

# Run in the current OS thread.

interp.exec('print("spam!")')
interp.exec("""if True:
    print('spam!')
    """)

from textwrap import dedent
interp.exec(dedent("""
    print('spam!')
    """))

def run(arg):
    return arg

res = interp.call(run, 'spam!')
print(res)

def run():
    print('spam!')

interp.call(run)

# Run in new OS thread.

t = interp.call_in_thread(run)
t.join()
```

`subprocess` — Subprocess management

Código fuente: Lib/subprocess.py [https://github.com/python/cpython/tree/3.14/Lib/subprocess.py]

The `subprocess` module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. This module intends to replace several older modules and functions:

```
os.system
os.spawn*
```

Information about how the `subprocess` module can be used to replace these modules and functions can be found in the following sections.

Ver también

PEP 324 [https://peps.python.org/pep-0324/] – PEP de proposición del módulo *subprocess*

Availability: not Android, not iOS, not WASI.

This module is not supported on mobile platforms or WebAssembly platforms.

Using the `subprocess` Module

La opción recomendada para invocar subprocesos es utilizar la función `run()` para todos los casos al alcance de ésta. Para usos más avanzados, se puede utilizar la interfaz de más bajo nivel `Popen`.

subprocess.run(*args*, ***, *stdin=None*, *input=None*, *stdout=None*, *stderr=None*, *capture_output=False*, *shell=False*, *cwd=None*, *timeout=None*, *check=False*, *encoding=None*, *errors=None*, *text=None*, *env=None*, *universal_newlines=None*, ***other_popen_kwargs*)

Ejecuta la orden descrita por *args*. Espera a que termine y retorna una instancia de `CompletedProcess`.

Los argumentos mostrados en la definición superior son sólo los más comunes; que se describen posteriormente en Argumentos frecuentemente empleados (de ahí el uso de la notación por clave en la signatura abreviada). La signatura completa de la función es a grandes rasgos la misma que la del constructor de `Popen`; la mayoría de los argumentos de esta función se pasan a esa interfaz (no es el caso de *timeout*, *input*, *check* ni *capture_output*).

If *capture_output* is true, stdout and stderr will be captured. When used, the internal `Popen` object is automatically created with *stdout* and *stderr* both set to `PIPE`. The *stdout* and *stderr* arguments may not be supplied at the same time as *capture_output*. If you wish to capture and combine both streams into one, set *stdout* to `PIPE` and *stderr* to `STDOUT`, instead of using *capture_output*.

A *timeout* may be specified in seconds, it is internally passed on to `Popen.communicate()`. If the timeout expires, the child process will be killed and waited for. The `TimeoutExpired` exception will be re-raised after the child process has terminated. The initial process creation itself cannot be interrupted on many platform APIs so you are not guaranteed to see a timeout exception until at least after however long process creation takes.

The *input* argument is passed to `Popen.communicate()` and thus to the subprocess's stdin. If used it must be a byte sequence, or a string if *encoding* or *errors* is specified or *text* is true. When used, the internal `Popen` object is automatically created with *stdin* set to `PIPE`, and the *stdin* argument may not be used as well.

Si *check* es verdadero y el proceso retorna un resultado distinto de cero, se lanzará una excepción **`CalledProcessError`**. Los atributos de dicha excepción contendrán los argumentos, el código retornado y tanto stdout como stderr si se capturaron.

Si se especifican *encoding* o *errors* o *text* es verdadero, se abrirán los objetos fichero para stdin, stdout y stderr en modo texto, con los *encoding* y *errors* especificados o los valores predeterminados de **`io.TextIOWrapper`**. El argumento *universal_newlines* equivale a *text* y se admite por compatibilidad hacia atrás. Los objetos fichero se abren en modo binario por defecto.

If *env* is not `None`, it must be a mapping that defines the environment variables for the new process; these are used instead of the default behavior of inheriting the current process” environment. It is passed directly to **`Popen`**. This mapping can be str to str on any platform or bytes to bytes on POSIX platforms much like **`os.environ`** or **`os.environb`**.

Ejemplos:

```
>>> subprocess.run(["ls", "-l"])  # doesn't capture
CompletedProcess(args=['ls', '-l'], returncode=0)

>>> subprocess.run("exit 1", shell=True, check=True
Traceback (most recent call last):
  ...
subprocess.CalledProcessError: Command 'exit 1' ret

>>> subprocess.run(["ls", "-l", "/dev/null"], captu
CompletedProcess(args=['ls', '-l', '/dev/null'], re
stdout=b'crw-rw-rw- 1 root root 1, 3 Jan 23 16:23 /
```

Added in version 3.5.

Distinto en la versión 3.6: Se añadieron los parámetros *encoding* y *errors*

Distinto en la versión 3.7: Se añadió el parámetro *text* como alias más comprensible de *universal_newlines*. Se añadió el parámetro *capture_output*.

Distinto en la versión 3.12: Changed Windows shell search order for `shell=True`. The current directory and `%PATH%` are replaced with `%COMSPEC%` and `%SystemRoot%\System32\cmd.exe`. As a result, dropping a malicious program named `cmd.exe` into a current directory no longer works.

class subprocess.CompletedProcess

El valor retornado por **`run()`**, que representa un proceso ya terminado.

args

Los argumentos utilizados para lanzar el proceso. Pueden ser una lista o una cadena.

returncode

Estado de salida del proceso hijo. Típicamente, un estado de salida 0 indica que la ejecución tuvo éxito.

Un valor negativo `-N` indica que el hijo fue forzado a terminar con la señal `N` (solamente POSIX).

stdout

La salida estándar capturada del proceso hijo. Una secuencia de bytes, o una cadena si se llamó a **`run()`** con *encoding*, *errors*, o *text* establecidos a `True`. `None` si no se capturó el error estándar.

Si se ejecutó el proceso con `stderr=subprocess.STDOUT`, stdout y stderr se combinarán en este atributo, y **`stderr`** será `None`.

stderr

El error estándar capturado del proceso hijo. Una secuencia de bytes, o una cadena si se llamó a **`run()`** con *encoding*, *errors*, o *text* establecidos a `True`. `None` si no se capturó el error estándar.

check_returncode()

Si **`returncode`** no es cero, lanza un **`CalledProcessError`**.

Added in version 3.5.

subprocess.DEVNULL

Valor especial que se puede usar como argumento *stdin*, *stdout* o *stderr* de **`Popen`** y que indica que se usará el fichero especial **`os.devnull`**.

Added in version 3.3.

subprocess.PIPE

Valor especial que se puede usar como argumento *stdin*, *stdout* o *stderr* de **`Popen`** y que indica que se abrirá un pipe al flujo indicado. Es útil para usarlo con **`Popen.communicate()`**.

subprocess.STDOUT

Valor especial que se puede usar de argumento *stderr* a **`Popen`** y que indica que el error estándar debería ir al mismo gestor que la salida estándar.

exception subprocess.SubprocessError

Clase base para el resto de excepciones de este módulo.

Added in version 3.3.

exception subprocess.TimeoutExpired

Subclase de **`SubprocessError`**, se lanza cuando expira un plazo de ejecución esperando a un proceso hijo.

cmd

Orden que se utilizó para lanzar el proceso hijo.

timeout

Plazo de ejecución en segundos.

output

Salida del proceso hijo si fue capturado por **`run()`** o **`check_output()`**. De lo contrario, `None`. Siempre es **`bytes`** cuando se capturó cualquier salida, independientemente de la configuración de `text=True`. Puede permanecer `None` en lugar de `b''` cuando no se observó ningún resultado.

stdout

Alias de *output*, por simetría con **`stderr`**.

stderr

Salida Stderr del proceso hijo si fue capturado por **`run()`**. De lo contrario, `None`. Siempre es **`bytes`** cuando se capturó la salida de stderr independientemente de la configuración de `text=True`. Puede permanecer como `None` en lugar de `b''` cuando no se observó ninguna salida de stderr.

Added in version 3.3.

Distinto en la versión 3.5: Se añadieron los atributos *stdout* y *stderr*

exception subprocess.CalledProcessError

Subclase de **`SubprocessError`**, lanzada cuando un proceso ejecutado por **`check_call()`**, **`check_output()`** o **`run()`** (con `check=True`) devuelve un estado de salida distinto de cero.

returncode

Estado de salida del proceso hijo. Si el proceso terminó por causa de una señal, el estado será el número de la señal en negativo.

cmd

Orden que se utilizó para lanzar el proceso hijo.

output

Salida del proceso hijo si fue capturada por `run()` o `check_output()`. De otro modo, `None`.

stdout

Alias de *output*, por simetría con `stderr`.

stderr

Salida de *stderr* del proceso hijo si fue capturada por `run()`. De otro modo, `None`.

Distinto en la versión 3.5: Se añadieron los atributos *stdout* y *stderr*

Argumentos frecuentemente empleados

Para permitir una gran variedad de usos, el constructor de `Popen` (y las funciones asociadas) aceptan un gran número de argumentos opcionales. Para los usos más habituales, se pueden dejar de forma segura los valores por defecto. Los argumentos más frecuentemente necesarios son:

> *args* se requiere en todas las llamadas; debe ser una cadena o una secuencia de argumentos al programa. En general, es mejor proporcionar una secuencia de argumentos porque permite que el módulo se ocupe de las secuencias de escape y los entrecomillados de los argumentos (por ejemplo, para permitir espacios en los nombres de fichero). Si se pasa una cadena simple, se ha de especificar *shell* como `True` (ver más adelante) o la cadena debe ser el nombre del programa a ejecutar sin especificar ningún argumento.
>
> *stdin*, *stdout* and *stderr* specify the executed program's standard input, standard output and standard error file handles, respectively. Valid

values are `None`, **`PIPE`**, **`DEVNULL`**, an existing file descriptor (a positive integer), and an existing file object with a valid file descriptor. With the default settings of `None`, no redirection will occur. **`PIPE`** indicates that a new pipe to the child should be created. **`DEVNULL`** indicates that the special file **`os.devnull`** will be used. Additionally, *stderr* can be **`STDOUT`**, which indicates that the stderr data from the child process should be captured into the same file handle as for *stdout*.

Si se especifican *encoding* o *errors*, o *text* (o su alias *universal_newlines*) es verdadero, se abrirán en modo texto los objetos fichero *stdin*, *stdout* y *stderr* usando los *encoding* y *errors* especificados en la llamada o los valores predeterminados de **`io.TextIOWrapper`**.

Para *stdin*, los saltos del línea `'\n'` de la entrada serán convertidos al separador de línea predeterminado **`os.linesep`**. Para *stdout* y *stderr*, todos los saltos de línea de la salida serán convertidos a `'\n'`. Hay más información en la documentación de la clase **`io.TextIOWrapper`** para el caso en que el argumento *newline* de su constructor es `None`.

Si no se usa el modo texto, *stdin*, *stdout* y *stderr* se abrirán como flujos binarios. No se realizará ninguna codificación ni conversión de salto de línea.

Distinto en la versión 3.6: Added the *encoding* and *errors* parameters.

Distinto en la versión 3.7: Se añadió el parámetro *text* como alias de *universal_newlines*.

Nota

El atributo *newlines* de los objetos fichero

`Popen.stdin`, `Popen.stdout` y `Popen.stderr` no es actualizado por el método `Popen.communicate()`.

Si *shell* es `True`, la orden especificada se ejecutará pasando por la shell. Esto tiene utilidad si se usa Python principalmente por el flujo de control mejorado sobre la mayoría de las shell de sistema, pero se desea también un acceso práctico a otras características de la shell, como pipes, nombres de fichero con comodines, expansión de variables de entorno o expansión de `~` al directorio *home* del usuario. Sin embargo, no se debe olvidar que el propio Python tiene implementaciones de muchas características tipo shell (en particular, `glob`, `fnmatch`, `os.walk()`, `os.path.expandvars()`, `os.path.expanduser()`, y `shutil`).

Distinto en la versión 3.3: Cuando *universal_newlines* es `True`, la clase usa la codificación `locale.getpreferredencoding(False)` en lugar de `locale.getpreferredencoding()`. Ver la clase `io.TextIOWrapper` para obtener más información sobre este cambio.

Nota

Leer la sección Consideraciones sobre la seguridad antes de usar `shell=True`.

Estas opciones y el resto se describen con más detalle en la documentación del constructor de `Popen`.

El constructor de Popen

El proceso interno de creación y gestión de este módulo lo gestiona la clase `Popen`. Proporciona una gran flexibilidad para que los desarrolladores sean capaces de gestionar los casos menos comunes que quedan sin cubrir por las funciones auxiliares.

class subprocess.Popen(*args*, *bufsize=-1*, *executable=None*, *stdin=None*, *stdout=None*, *stderr=None*, *preexec_fn=None*, *close_fds=True*, *shell=False*, *cwd=None*, *env=None*, *universal_newlines=None*, *startupinfo=None*, *creationflags=0*, *restore_signals=True*, *start_new_session=False*, *pass_fds=()*, ***, *group=None*, *extra_groups=None*, *user=None*, *umask=-1*, *encoding=None*, *errors=None*, *text=None*, *pipesize=-1*, *process_group=None*)

Ejecuta un programa hijo en un proceso nuevo. En POSIX, la clase usa `os.execvp()` como comportamiento para lanzar el proceso hijo. En Windows, la clase usa la función `CreateProcess()` de Windows. Los argumentos de `Popen` son los siguientes.

args debe ser o una secuencia de argumentos de programa o una cadena simple o un objeto tipo ruta. Por omisión, el programa a ejecutar es el primer elemento de *args* si *args* es una secuencia. Si *args* es una cadena, la interpretación es dependiente de la plataforma, según se describe más abajo. Véase los argumentos *shell* y *executable* para más información sobre el comportamiento por defecto. Salvo que se indique, se recomienda pasar los *args* como una secuencia.

Advertencia

Para obtener la máxima confiabilidad, use una ruta completa para el ejecutable. Para buscar un nombre no calificado en **`PATH`**, use `shutil.which()`. En todas las plataformas, pasar `sys.executable` es la forma recomendada de iniciar de nuevo el intérprete de Python actual y usar el formato de línea de comandos `-m` para iniciar un módulo instalado.

La resolución de la ruta de *executable* (o el primer elemento de *args*) depende de la plataforma. Para POSIX, consulte `os.execvpe()`, y tenga en cuenta que al resolver o buscar la ruta ejecutable, *cwd* anula el directorio de trabajo actual y *env* puede anular la variable de entorno `PATH`. Para Windows, consulte la documentación de los parámetros `lpApplicationName` e `lpCommandLine` de WinAPI `CreateProcess`, y tenga en cuenta que al resolver o buscar la ruta ejecutable con `shell = False`, *cwd* no anula el directorio de trabajo actual y *env* no puede anular la variable de entorno `PATH`. El uso de una ruta completa evita todas estas variaciones.

Un ejemplo del paso de argumentos a un programa externo mediante una secuencia:

```
Popen(["/usr/bin/git", "commit", "-m", "Fixes a bug
```

En POSIX, si *args* es una cadena se interpreta como el nombre o la ruta del programa que ejecutar. Sin embargo, esto solamente funciona si no hay que pasar argumentos al programa.

Nota

Puede que no resulte evidente cómo descomponer una orden de la shell en una secuencia de argumentos, especialmente en casos complejos. **`shlex.split()`** puede aclarar cómo determinar la descomposición en tokens de *args*:

```
>>> import shlex, subprocess
>>> command_line = input()
/bin/vikings -input eggs.txt -output "spam spam.tx
>>> args = shlex.split(command_line)
>>> print(args)
['/bin/vikings', '-input', 'eggs.txt', '-output',
>>> p = subprocess.Popen(args) # Success!
```

Hay que destacar en particular que las opciones (como -*input*) y los argumentos (como *eggs.txt*) que van separados por espacio en blanco en la shell van en elementos de lista separados, mientras los argumentos que necesitan entrecomillado o escapado de espacios cuando se usan en la shell (como los nombres de ficheros con espacios o la orden *echo* anteriormente mostrada) son elementos simples de la lista.

En Windows, si *args* es una secuencia, se convertirá a cadena del modo descrito en Cómo convertir una secuencia de argumentos a una cadena en Windows. Esto es así porque la función de bajo nivel `CreateProcess()` funciona sobre cadenas.

Distinto en la versión 3.6: El parámetro *args* toma un objeto tipo ruta si *shell* es `False` y una secuencia de objetos tipo fichero en POSIX.

Distinto en la versión 3.8: El parámetro *args* toma un objeto tipo ruta si *shell* es `False` y una secuencia de bytes y objetos tipo fichero en Windows.

El argumento *shell* (`False` por defecto) especifica si usar la shell como programa a ejecutar. Si *shell* es `True`, se recomienda pasar *args* como cadena mejor que como secuencia.

En POSIX con `shell=True`, la shell predeterminada es `/bin/sh`. Si *args* es una cadena, ésta especifica la orden a ejecutar por la shell. Esto significa que la cadena tiene que tener el formato que tendría si se tecleara en la línea de órdenes. Esto incluye, por ejemplo, el entrecomillado y las secuencias de escape necesarias para los nombres de fichero que contengan espacios. Si *args* es una secuencia, el primer elemento especifica la cadena de la orden y cualquier otro elemento será tratado como argumentos adicionales a la propia shell. De este modo, **`Popen`** hace el equivalente de:

```
Popen(['/bin/sh', '-c', args[0], args[1], ...])
```

En Windows con `shell=True`, la variable de entorno **COMSPEC** especifica la shell predeterminada. Solamente hace falta especificar `shell=True` en Windows cuando la orden que se desea ejecutar es interna a la shell (como. **dir** o **copy**). No hace falta especificar `shell=True` para ejecutar un fichero por lotes o un ejecutable de consola.

Nota

Leer la sección Consideraciones sobre la seguridad antes de usar `shell=True`.

Se proporcionará *bufsize* como el argumento correspondiente a la función `open()` cuando se creen los objetos fichero de los flujos stdin/stdout/stderr:

- `0` means unbuffered (read and write are one system call and can return short)
- `1` means line buffered (only usable if `text=True` or `universal_newlines=True`)
- cualquier otro valor positivo indica que hay que usar un búfer de aproximadamente dicho tamaño
- *bufsize* negativo (el valor por defecto) indica que se use el valor predeterminado del sistema, *io.DEFAULT_BUFFER_SIZE*.

Distinto en la versión 3.3.1: bufsize now defaults to -1 to enable buffering by default to match the behavior that most code expects. In versions prior to Python 3.2.4 and 3.3.1 it incorrectly defaulted to `0` which was unbuffered and allowed short reads. This was unintentional and did not match the behavior of Python 2 as most code expected.

El argumento *executable* especifica un programa de reemplazo que ejecutar. Esto es muy poco frecuente. Cuando `shell=False`, *executable* reemplaza al programa especificado por *args*. Sin embargo, se pasan los *args* originales al programa. La mayoría de los programas tratan el programa especificado en los *args* como el nombre del programa, que puede ser diferente del programa realmente ejecutado. En POSIX, el nombre en *args* funciona como nombre visible en utilidades como **ps**. Si `shell=True`, en POSIX el argumento *executable* especifica una shell de reemplazo de la predeterminada `/bin/sh`.

Distinto en la versión 3.6: El parámetro *executable* acepta un objeto tipo ruta en POSIX.

Distinto en la versión 3.8: El parámetro *executable* acepta bytes y un objeto tipo ruta en POSIX.

Distinto en la versión 3.12: Changed Windows shell search order for `shell=True`. The current directory and `%PATH%` are replaced with `%COMSPEC%` and `%SystemRoot%\System32\cmd.exe`. As a result, dropping a malicious program named `cmd.exe` into a current directory no longer works.

stdin, *stdout* and *stderr* specify the executed program's standard input, standard output and standard error file handles, respectively. Valid values are `None`, **`PIPE`**, **`DEVNULL`**, an existing file descriptor (a positive integer), and an existing file object with a valid file descriptor. With the default settings of `None`, no redirection will occur. **`PIPE`** indicates that a new pipe to the child should be created. **`DEVNULL`** indicates that the special file **`os.devnull`** will be used. Additionally, *stderr* can be **`STDOUT`**, which indicates that the stderr data from the applications should be captured into the same file handle as for *stdout*.

Si se establece *preexec_fn* a un objeto invocable, se llamará a dicho objeto en el proceso hijo justo antes de que se ejecute el hijo. (solamente POSIX)

Advertencia

El parámetro *preexec_fn* NO ES SEGURO para usar en presencia de hilos en su aplicación. El subproceso podría bloquearse antes de que se llame a exec.

Nota

Si necesita modificar el entorno para el subproceso, use el parámetro *env* en lugar de hacerlo en un *preexec_fn*. Los parámetros *start_new_session* y *process_group* deben ocupar el lugar del código que usa *preexec_fn* para llamar a **`os.setsid()`** o **`os.setpgid()`** en el subproceso

Distinto en la versión 3.8: Se ha abandonado el soporte de *preexec_fn* en subintérpretes. El uso de dicho parámetro en un subintérprete lanza **`RuntimeError`**. La nueva restricción puede afectar a aplicaciones desplegadas en *mod_wsgi*, *uWSGI* y otros entornos incrustados.

If *close_fds* is true, all file descriptors except `0`, `1` and `2` will be closed before the child process is executed. Otherwise when *close_fds* is false, file descriptors obey their inheritable flag as described in Herencia de los descriptores de archivos.

En Windows, si *close_fds* es verdadero el proceso hijo no heredará ningún gestor de fichero salvo que se le pasen explícitamente en el elemento `handle_list` de **`STARTUPINFO.lpAttributeList`**, o mediante redirección estándar.

Distinto en la versión 3.2: El valor predeterminado de *close_fds* se cambió de **`False`** a lo antes descrito.

Distinto en la versión 3.7: En Windows, el valor predeterminado de *close_fds* se cambió de **`False`** a **`True`** al redirigir los gestores estándar. Ahora es posible establecer *close_fds* a **`True`** cuando se redirigen los gestores estándar.

pass_fds es una secuencia de descriptor de ficheros opcional que han de mantenerse abiertos entre el proceso padre e hijo. Si se proporciona un valor a *pass_fds* se fuerza *close_fds* a **`True`**. (solamente POSIX)

Distinto en la versión 3.2: Se añadió el parámetro *pass_fds*.

Si *cwd* no es `None`, la función cambia el directorio de trabajo a *cwd* antes de ejecutar el proceso hijo. *cwd* puede ser una cadena, o un objeto bytes o tipo ruta. En POSIX, la función busca *executable* (o el primer elemento de *args*) relativo a *cwd* si la ruta del ejecutable es relativa.

Distinto en la versión 3.6: El parámetro *cwd* acepta un objeto tipo ruta en POSIX.

Distinto en la versión 3.7: El parámetro *cwd* acepta un objeto tipo ruta en Windows.

Distinto en la versión 3.8: El parámetro *cwd* acepta un objeto bytes en Windows.

Si *restore_signals* es verdadero (el valor por defecto) todas las señales que Python ha establecido a SIG_IGN se restauran a SIG_DFL en el proceso hijo antes del *exec*. En la actualidad, esto incluye las señales SIGPIPE, SIGXFZ y SIGXFSZ (solamente POSIX).

Distinto en la versión 3.2: Se añadió *restore_signals*.

Si *start_new_session* es verdadero, la llamada al sistema `setsid()` se realizará en el proceso secundario antes de la ejecución del subproceso.

Availability: POSIX

Distinto en la versión 3.2: Se añadió *start_new_session*.

Si *process_group* es un entero no negativo, la llamada al sistema `setpgid(0, value)` se realizará en el proceso secundario antes de la ejecución del subproceso.

Availability: POSIX

Distinto en la versión 3.11: Se agregó *process_group*.

If *group* is not `None`, the setregid() system call will be made in the child process prior to the execution of the subprocess. If the provided value is a string, it will be looked up via **`grp.getgrnam()`** and the value in `gr_gid` will be used. If the value is an integer, it will be passed verbatim. (POSIX only)

Availability: POSIX

Added in version 3.9.

If *extra_groups* is not `None`, the setgroups() system call will be

made in the child process prior to the execution of the subprocess. Strings provided in *extra_groups* will be looked up via **`grp.getgrnam()`** and the values in `gr_gid` will be used. Integer values will be passed verbatim. (POSIX only)

Availability: POSIX

Added in version 3.9.

If *user* is not `None`, the setreuid() system call will be made in the child process prior to the execution of the subprocess. If the provided value is a string, it will be looked up via **`pwd.getpwnam()`** and the value in `pw_uid` will be used. If the value is an integer, it will be passed verbatim. (POSIX only)

Nota

Specifying *user* will not drop existing supplementary group memberships! The caller must also pass `extra_groups=()` to reduce the group membership of the child process for security purposes.

Availability: POSIX

Added in version 3.9.

Si *umask* no es negativo, la llamada a sistema umask() se hará en el proceso hijo antes de la ejecución del subproceso.

Availability: POSIX

Added in version 3.9.

If *env* is not `None`, it must be a mapping that defines the environment variables for the new process; these are used instead of the default behavior of inheriting the current process" environment. This mapping can be str to str on any platform or bytes to bytes on POSIX platforms much like **`os.environ`** or **`os.environb`**.

Nota

If specified, *env* must provide any variables required for the program to execute. On Windows, in order to run a side-by-side assembly [https://en.wikipedia.org/wiki/Side-by-Side_Assembly] the specified *env* **must** include a valid `%SystemRoot%`.

Si se especifican *encoding* o *errors*, o *text* es verdadero, los objetos de archivo *stdin*, *stdout* y *stderr* se abren en modo de texto con los *encoding* y *errors* especificados, como se describe anteriormente en Argumentos frecuentemente empleados. El argumento *universal_newlines* es equivalente a *text* y se proporciona para compatibilidad con versiones anteriores. De forma predeterminada, los objetos de archivo se abren en modo binario.

Added in version 3.6: Se añadieron *encoding* y *errors*.

Added in version 3.7: Se añadió *text* como alias más legible de *universal_newlines*.

If given, *startupinfo* will be a **`STARTUPINFO`** object, which is passed to the underlying `CreateProcess` function.

If given, *creationflags*, can be one or more of the following flags:

- **`CREATE_NEW_CONSOLE`**
- **`CREATE_NEW_PROCESS_GROUP`**
- **`ABOVE_NORMAL_PRIORITY_CLASS`**
- **`BELOW_NORMAL_PRIORITY_CLASS`**
- **`HIGH_PRIORITY_CLASS`**
- **`IDLE_PRIORITY_CLASS`**
- **`NORMAL_PRIORITY_CLASS`**
- **`REALTIME_PRIORITY_CLASS`**
- **`CREATE_NO_WINDOW`**
- **`DETACHED_PROCESS`**
- **`CREATE_DEFAULT_ERROR_MODE`**
- **`CREATE_BREAKAWAY_FROM_JOB`**

pipeize se puede usar para cambiar el tamaño de la tubería cuando **`PIPE`** se usa para *stdin*, *stdout* o *stderr*. El tamaño de la tubería solo se cambia en plataformas que lo admiten (solo Linux en este momento de redacción). Otras plataformas ignorarán este parámetro.

Distinto en la versión 3.10: Added the *pipesize* parameter.

Se puede usar los objetos Popen como gestores de contexto mediante sentencia **`with`**: a la salida, los descriptores de flujo se cierran y se espera a que acabe el proceso.

```
with Popen(["ifconfig"], stdout=PIPE) as proc:
    log.write(proc.stdout.read())
```

Popen y el resto de las funciones de este módulo que la usan lanzan un evento de auditoría `subprocess.Popen` con argumentos `executable`, `args`, `cwd` y `env`. El valor de `args` puede ser una cadena simple o un alista de cadenas, dependiendo de la plataforma.

Distinto en la versión 3.2: Se añadió la funcionalidad de gestor de contexto.

Distinto en la versión 3.6: El destructor de Popen ahora emite una advertencia **`ResourceWarning`** si el proceso hijo todavía se está ejecutando.

Distinto en la versión 3.8: Popen puede usar **`os.posix_spawn()`** en algunos casos para obtener mejor rendimiento. En el Subsistema de Windows para Linux (WSL) y en la emulación del modo usuario de QEMU, el constructor de Popen que use **`os.posix_spawn()`** ya no lanzará una excepción cuando se den errores tales como que un programa no esté, sino que el proceso hijo fracasará con un **`returncode`** distinto de cero.

Excepciones

Las excepciones lanzadas en el proceso hijo, antes de que el nuevo programa haya empezado a ejecutarse, se relanzarán en el padre.

La excepción más común que se lanza es `OSError`. Esto ocurre, por ejemplo, al intentar ejecutar un archivo inexistente. Las aplicaciones deben prepararse para excepciones **`OSError`**. Tenga en cuenta que, cuando `shell = True`, **`OSError`** será lanzado por el hijo solo si no se encontró el shell seleccionado. Para determinar si el shell no pudo encontrar la aplicación solicitada, es necesario verificar el código de retorno o la salida del subproceso.

Se lanzará un `ValueError` si se llama a `Popen` con argumentos no válidos.

`check_call()` y `check_output()` lanzarán un `CalledProcessError` si el proceso invocado retorna un código de retorno distinto de cero.

All of the functions and methods that accept a *timeout* parameter, such as `run()` and `Popen.communicate()` will raise `TimeoutExpired` if the timeout expires before the process exits.

Todas las excepciones definidas en este módulo heredan de `SubprocessError`.

Added in version 3.3: Se añadió la clase base `SubprocessError`.

Consideraciones sobre seguridad

Unlike some other popen functions, this library will not implicitly choose to call a system shell. This means that all characters, including shell metacharacters, can safely be passed to child processes. If the shell is invoked explicitly, via `shell=True`, it is the application’s responsibility to ensure that all whitespace and metacharacters are quoted appropriately to avoid shell injection [https://en.wikipedia.org/wiki/Shell_injection#Shell_injection] vulnerabilities. On some platforms, it is possible to use `shlex.quote()` for this escaping.

On Windows, batch files (`*.bat` or `*.cmd`) may be launched by the operating system in a system shell regardless of the arguments passed to this library. This could result in arguments being parsed according to shell rules, but without any escaping added by Python.

If you are intentionally launching a batch file with arguments from untrusted sources, consider passing `shell=True` to allow Python to escape special characters. See gh-114539 [https://github.com/python/cpython/issues/114539] for additional discussion.

Objetos Popen

Las instancias de la clase **`Popen`** cuentan con los siguientes métodos:

Popen.poll()

Comprueba si el proceso hijo ha finalizado. Establece y retorna el atributo **`returncode`**. De lo contrario, retorna `None`.

Popen.wait(*timeout=None*)

Espera a que termine el proceso hijo. Establece y retorna el atributo **`returncode`**.

Si el proceso no finaliza tras *timeout* segundos, lanza una excepción **`TimeoutExpired`**. Se puede capturar esta excepción para reintentar la espera.

Nota

Esto causará un bloqueo cuando se use `stdout=PIPE` o `stderr=PIPE` y el proceso hijo genere suficiente salida hacia un pipe como para bloquear esperando que el búfer del pipe del SO acepte más datos. Se debe usar **`Popen.communicate()`** cuando se usen pipes para evitar esto.

Nota

When the `timeout` parameter is not `None`, then (on POSIX) the function is implemented using a busy loop (non-blocking call and short sleeps). Use the **`asyncio`** module for an asynchronous wait: see **`asyncio.create_subprocess_exec`**.

Distinto en la versión 3.3: Se añadió *timeout*.

Popen.communicate(*input=None*, *timeout=None*)

Interactúa con el proceso: Envía datos a stdin. Lee datos de stdout y stderr hasta encontrar un fin-de-fichero. Espera a que termine el proceso y escribe el atributo **`returncode`**. El argumento opcional *input* debe contener los datos que hay que enviar al proceso hijo o `None` si no hay que enviar datos al proceso hijo. Si se abrieron los flujos en modo texto, *input* ha de ser una cadena de caracteres. En caso contrario, debe contener bytes.

`communicate()` retorna una tupla `(stdout_data, stderr_data)`. Los datos serán cadenas si se abrieron los flujos en modo texto, en caso contrario serán bytes.

Adviértase que si se desea enviar datos al stdin del proceso, se ha de crear el objeto Popen con `stdin=PIPE`. Análogamente, para obtener algo diferente de `None` en la tupla del resultado, hay que suministrar `stdout=PIPE` o `stderr=PIPE` también.

If the process does not terminate after *timeout* seconds, a **`TimeoutExpired`** exception will be raised. Catching this exception and retrying communication will not lose any output. Supplying *input* to a subsequent post-timeout **`communicate()`** call is in undefined behavior and may become an error in the future.

No se matará el proceso si vence el plazo de ejecución, así que para hacer limpieza, una aplicación correcta debería matar el proceso y terminar la comunicación:

```
proc = subprocess.Popen(...)
try:
    outs, errs = proc.communicate(timeout=15)
except TimeoutExpired:
    proc.kill()
    outs, errs = proc.communicate()
```

After a call to `communicate()` raises `TimeoutExpired`, do not call `wait()`. Use an additional `communicate()` call to finish handling pipes and populate the `returncode` attribute.

Nota

Los datos leídos pasan por un búfer en memoria, así que no se ha de usar este método para un tamaño de datos grande o ilimitado.

Distinto en la versión 3.3: Se añadió *timeout*.

Popen.send_signal(*signal*)

Envía la señal *signal* al proceso hijo.

No hace nada si el proceso ya ha terminado.

Nota

En Windows, SIGTERM es un alias de `terminate()`. CTRL_C_EVENT y CTRL_BREAK_EVENT se pueden enviar a procesos iniciados con un parámetro *creationflags* que incluye `CREATE_NEW_PROCESS_GROUP`.

Popen.terminate()

Stop the child. On POSIX OSs the method sends `SIGTERM` to the child. On Windows the Win32 API function `TerminateProcess()` is called to stop the child.

Popen.kill()

Mata el proceso hijo. En sistemas operativos POSIX la función envía SIGKILL al proceso hijo. En Windows `kill()` es un alias de `terminate()`.

The following attributes are also set by the class for you to access. Reassigning them to new values is unsupported:

Popen.args

El argumento *args* según se pasó a `Popen`: o una secuencia de argumentos del programa o una cadena sencilla.

Added in version 3.3.

Popen.stdin

If the *stdin* argument was `PIPE`, this attribute is a writeable stream object as returned by `open()`. If the *encoding* or *errors* arguments were specified or the *text* or *universal_newlines* argument was `True`, the stream is a text stream, otherwise it is a byte stream. If the *stdin* argument was not **`PIPE`**, this attribute is `None`.

Popen.stdout

If the *stdout* argument was `PIPE`, this attribute is a readable stream object as returned by `open()`. Reading from the stream provides output from the child process. If the *encoding* or *errors* arguments were specified or the *text* or *universal_newlines* argument was `True`, the stream is a text stream, otherwise it is a byte stream. If the *stdout* argument was not **`PIPE`**, this attribute is `None`.

Popen.stderr

If the *stderr* argument was `PIPE`, this attribute is a readable stream object as returned by `open()`. Reading from the stream provides error output from the child process. If the *encoding* or *errors* arguments were specified or the *text* or *universal_newlines* argument was `True`, the stream is a text stream, otherwise it is a byte stream. If the *stderr* argument was not **`PIPE`**, this attribute is `None`.

Advertencia

Se ha de usar `communicate()` en lugar de `.stdin.write`, `.stdout.read` o `.stderr.read` para evitar bloqueos por búfer de pipes del SO llenos que puedan bloquear el proceso hijo.

Popen.pid

El ID de proceso del hijo.

Adviértase que si se establece el argumento *shell* a `True`, éste es el ID de proceso del la shell generada.

Popen.returncode

The child return code. Initially `None`, **`returncode`** is set by a call to the **`poll()`**, **`wait()`**, or **`communicate()`** methods if they detect that the process has terminated.

A `None` value indicates that the process hadn't yet terminated at the time of the last method call.

Un valor negativo `-N` indica que el hijo fue forzado a terminar con la señal `N` (solamente POSIX).

When `shell=True`, the return code reflects the exit status of the shell itself (e.g. `/bin/sh`), which may map signals to codes such as `128+N`. See the documentation of the shell (for example, the Bash manual's Exit Status) for details.

Elementos auxiliares de Popen en Windows

La clase **`STARTUPINFO`** y las siguientes constantes sólo están disponibles en Windows.

class subprocess.STARTUPINFO(**, dwFlags=0, hStdInput=None, hStdOutput=None, hStdError=None, wShowWindow=0, lpAttributeList=None*)

Se utiliza un soporte parcial de la estructura STARTUPINFO [https://msdn.microsoft.com/en-us/library/ms686331(v=vs.85).aspx] de Windows para la creación de **`Popen`**. Se pueden establecer los siguientes atributos pasándolos como argumentos sólo por clave.

Distinto en la versión 3.7: Se añadió el soporte de argumentos sólo por clave.

dwFlags

Un campo bit que determina si se usan ciertos atributos de **`STARTUPINFO`** cuando el proceso crea una ventana.

```
si = subprocess.STARTUPINFO()
si.dwFlags = subprocess.STARTF_USESTDHANDLES |
```

hStdInput

Si **`dwFlags`** especifica **`STARTF_USESTDHANDLES`**, este atributo es el gestor de entrada estándar del proceso. Si no se especifica **`STARTF_USESTDHANDLES`**, el valor predeterminado de entrada estándar es el búfer de teclado.

hStdOutput

Si **`dwFlags`** especifica **`STARTF_USESTDHANDLES`**, este atributo es el gestor de salida estándar del proceso. En caso contrario, se hace caso omiso del atributo y el valor predeterminado de salida estándar es el búfer de ventana.

hStdError

Si **`dwFlags`** especifica **`STARTF_USESTDHANDLES`**, este atributo es el gestor de error estándar del proceso. En caso contrario, se hace caso omiso del atributo y el valor predeterminado de error estándar es el búfer de ventana.

wShowWindow

Si **`dwFlags`** especifica **`STARTF_USESHOWWINDOW`**, este atributo puede ser cualquiera de los valores que pueden especificarse en el parámetro `nCmdShow` para la función ShowWindow [https://msdn.microsoft.com/en-us/library/ms633548(v=vs.85).aspx], salvo `SW_SHOWDEFAULT`. De otro modo, se hace caso omiso del atributo.

Se proporciona **`SW_HIDE`** para este atributo. Se usa cuando se llama a **`Popen`** con `shell=True`.

lpAttributeList

Un diccionario de atributos adicionales para la creación del proceso, según `STARTUPINFOEX`, véase UpdateProcThreadAttribute [https://msdn.microsoft.com/en-us/library/windows/desktop/ms686880(v=vs.85).aspx].

Atributos admitidos:

handle_list

Una secuencia de gestores que se heredará. *close_fds* debe ser verdadero si no viene vacío.

Los gestores deben hacerse temporalmente heredables por **`os.set_handle_inheritable()`** cuando se pasan al constructor de **`Popen`** o de lo contrario, se lanzará **`OSError`** con el error de Windows `ERROR_INVALID_PARAMETER` (87).

Advertencia

En un proceso multihilo, hay que evitar filtrar gestores no heredables cuando se combina esta característica con llamadas concurrentes a otras funciones de creación de procesos que heredan todos los gestores, como **`os.system()`**. Esto también rige para la redirección de gestores estándar, que crea temporalmente gestores heredables.

Added in version 3.7.

Constantes de Windows

The **`subprocess`** module exposes the following constants.

subprocess.STD_INPUT_HANDLE

El dispositivo de entrada estándar. Inicialmente, es el búfer de entrada de la consola, `CONIN$`.

subprocess.STD_OUTPUT_HANDLE

El dispositivo de salida estándar. Inicialmente, es el búfer de pantalla de la consola activa, `CONOUT$`.

subprocess.STD_ERROR_HANDLE

El dispositivo de error estándar. Inicialmente, es el búfer de pantalla de la consola activa, `CONOUT$`.

subprocess.SW_HIDE

Oculta la ventana. Se activará otra ventana.

subprocess.STARTF_USESTDHANDLES

Especifica que los atributos **`STARTUPINFO.hStdInput`**, **`STARTUPINFO.hStdOutput`**, y **`STARTUPINFO.hStdError`** contienen información adicional.

subprocess.STARTF_USESHOWWINDOW

Especifica que el atributo **`STARTUPINFO.wShowWindow`** contiene información adicional.

subprocess.STARTF_FORCEONFEEDBACK

A **`STARTUPINFO.dwFlags`** parameter to specify that the *Working in Background* mouse cursor will be displayed while a process is launching. This is the default behavior for GUI processes.

Added in version 3.13.

subprocess.STARTF_FORCEOFFFEEDBACK

A **`STARTUPINFO.dwFlags`** parameter to specify that the mouse cursor will not be changed when launching a process.

Added in version 3.13.

subprocess.CREATE_NEW_CONSOLE

El nuevo proceso obtiene una nueva consola, en lugar de heredar la consola del padre (que es el comportamiento predeterminado).

subprocess.CREATE_NEW_PROCESS_GROUP

Un parámetro **`Popen`** `creationflags` para especificar que se cree un nuevo grupo de procesos. Este indicador es necesario para usar **`os.kill()`** sobre el subproceso.

Este indicador no se tiene en cuenta si se especifica **`CREATE_NEW_CONSOLE`**.

subprocess.ABOVE_NORMAL_PRIORITY_CLASS

Parámetro **`Popen`** `creationflags` para especificar que el nuevo proceso tendrá una prioridad superior a la media.

Added in version 3.7.

subprocess.BELOW_NORMAL_PRIORITY_CLASS

Parámetro **`Popen`** `creationflags` para especificar que el nuevo proceso tendrá una prioridad inferior a la media.

Added in version 3.7.

subprocess.HIGH_PRIORITY_CLASS

Parámetro **`Popen`** `creationflags` para especificar que el nuevo proceso tendrá una prioridad elevada.

Added in version 3.7.

subprocess.IDLE_PRIORITY_CLASS

Parámetro **`Popen`** `creationflags` para especificar que el nuevo proceso tendrá una la mínima prioridad.

Added in version 3.7.

subprocess.NORMAL_PRIORITY_CLASS

A **`Popen`** `creationflags` parameter to specify that a new process will have a normal priority. (default)

Added in version 3.7.

subprocess.REALTIME_PRIORITY_CLASS

Parámetro **Popen** `creationflags` para especificar que el nuevo proceso tendrá una prioridad de tiempo real. Raramente se debería usar REALTIME_PRIORITY_CLASS, porque esto interrumpe los hilos que gestionan la entrada del ratón y del teclado, así como la gestión del volcado de búfer del disco. Esta clase es apropiada para aplicaciones que se comunican directamente con el hardware o que llevan a cabo tareas breves que no admitan interrupciones.

Added in version 3.7.

subprocess.CREATE_NO_WINDOW

Parámetro **Popen** `creationflags` para especificar que el nuevo proceso no creará una ventana.

Added in version 3.7.

subprocess.DETACHED_PROCESS

Parámetro **Popen** `creationflags` para especificar que el nuevo proceso no heredará la consola del padre. Este valor es incompatible con CREATE_NEW_CONSOLE.

Added in version 3.7.

subprocess.CREATE_DEFAULT_ERROR_MODE

Parámetro **Popen** `creationflags` para especificar que el nuevo proceso no hereda el modo de error del proceso padre. En su lugar, el proceso parte del modo de error predeterminado. Esta característica es particularmente útil para aplicaciones de shell multihilo que se ejecutan con los errores "duros" desactivados.

Added in version 3.7.

subprocess.CREATE_BREAKAWAY_FROM_JOB

Parámetro **Popen** `creationflags` para especificar que el nuevo proceso no está asociado a la tarea.

Added in version 3.7.

Antigua API de alto nivel

Antes de Python 3.5, estas tres funciones conformaban la API de alto nivel para subprocesos. Ahora se puede usar `run()` en muchos casos, pero hay mucho código escrito con estas funciones.

subprocess.call(*args*, ***, *stdin=None*, *stdout=None*, *stderr=None*, *shell=False*, *cwd=None*, *timeout=None*, ***other_popen_kwargs*)

Ejecutar la orden descrita por *args*. Esperar que la orden se complete y retornar al atributo `returncode`.

El código que requiera capturar stdout o stderr debería usar `run()` en su lugar:

```
run(...).returncode
```

Para suprimir stdout o stderr se ha de proporcionar un valor de `DEVNULL`.

Se muestran algunos argumentos comunes. La signatura completa de la función es la misma que la del constructor de `Popen`; esta función pasa todos los argumentos proporcionados (salvo *timeout*) directamente a esa interfaz.

Nota

No usar `stdout=PIPE` ni `stderr=PIPE` con esta función. El proceso hijo se bloqueará si genera suficiente salida a un pipe como para saturar el búfer del pipe del sistema operativo mientras no se lee de los pipes.

Distinto en la versión 3.3: Se añadió *timeout*.

Distinto en la versión 3.12: Changed Windows shell search order for `shell=True`. The current directory and `%PATH%` are replaced with `%COMSPEC%` and `%SystemRoot%`

`\System32\cmd.exe`. As a result, dropping a malicious program named `cmd.exe` into a current directory no longer works.

subprocess.check_call(*args*, ***, *stdin=None*, *stdout=None*, *stderr=None*, *shell=False*, *cwd=None*, *timeout=None*, ***other_popen_kwargs*)

Ejecuta la instrucción con argumentos. Espera a que la instrucción se complete. Si el código de retorno es cero retorna; en caso contrario, lanza **`CalledProcessError`**. El objeto **`CalledProcessError`** tendrá el código de retorno en el atributo **`returncode`**.Si **`check_call()`** no pudo iniciar el proceso, propagará la excepción que fue lanzada.

El código que requiera capturar stdout o stderr debería usar **`run()`** en su lugar:

```
run(..., check=True)
```

Para suprimir stdout o stderr se ha de proporcionar un valor de **`DEVNULL`**.

Se muestran algunos argumentos comunes. La signatura completa de la función es la misma que la del constructor de **`Popen`**; esta función pasa todos los argumentos proporcionados (salvo *timeout*) directamente a esa interfaz.

Nota

No usar `stdout=PIPE` ni `stderr=PIPE` con esta función. El proceso hijo se bloqueará si genera suficiente salida a un pipe como para saturar el búfer del pipe del sistema operativo mientras no se lee de los pipes.

Distinto en la versión 3.3: Se añadió *timeout*.

Distinto en la versión 3.12: Changed Windows shell search order for `shell=True`. The current directory and `%PATH%` are replaced with `%COMSPEC%` and `%SystemRoot%\System32\cmd.exe`. As a result, dropping a malicious program named `cmd.exe` into a current directory no longer works.

subprocess.check_output(*args*, **, *stdin=None*, *stderr=None*, *shell=False*, *cwd=None*, *encoding=None*, *errors=None*, *universal_newlines=None*, *timeout=None*, *text=None*, ***other_popen_kwargs*)

Ejecuta orden con argumentos y retorna su salida.

Si el código de retorno fue diferente de cero lanza un **`CalledProcessError`**. El objeto **`CalledProcessError`** tendrá el código de retorno en el atributo **`returncode`** y los datos de salida en el atributo **`output`**.

Esto equivale a:

```
run(..., check=True, stdout=PIPE).stdout
```

Los argumentos mostrados arriba son meramente algunos de los comunes. La signatura completa de la función es casi la misma que la de **`run()`** - la mayoría de los argumentos se pasa directamente a esa interfaz. Existe una diferencia con la interfaz de **`run()`** respecto al comportamiento: pasar `input=None` genera el mismo comportamiento que `input=b''` (o `input=''`, dependiendo de otros argumentos) en vez de usar el flujo entrada estándar del padre.

Por omisión, esta función retornará los datos como bytes codificados. La codificación real de los datos podría depender de la orden invocada, por lo que la decodificación a texto se deberá hacer al nivel de la aplicación.

Este comportamiento se puede modificar estableciendo *text*, *encoding*, *errors*, o *universal_newlines* a `True`, como se describe en Argumentos frecuentemente empleados y **`run()`**.

Para capturar también el error estándar del resultado se debe usar `stderr=subprocess.STDOUT`:

```
>>> subprocess.check_output(
...     "ls non_existent_file; exit 0",
...     stderr=subprocess.STDOUT,
...     shell=True)
'ls: non_existent_file: No such file or directory\n
```

Added in version 3.1.

Distinto en la versión 3.3: Se añadió *timeout*.

Distinto en la versión 3.4: Se añadió soporte para el argumento por clave *input*.

Distinto en la versión 3.6: Se añadieron *encoding* y *errors*. Ver **`run()`** para más detalles.

Added in version 3.7: Se añadió *text* como alias más legible de *universal_newlines*.

Distinto en la versión 3.12: Changed Windows shell search order for `shell=True`. The current directory and `%PATH%` are replaced with `%COMSPEC%` and `%SystemRoot%\System32\cmd.exe`. As a result, dropping a malicious program named `cmd.exe` into a current directory no longer works.

Replacing Older Functions with the `subprocess` Module

En esta sección, «a se convierte en b» significa que b se puede usar en lugar de.

Nota

Todas las funciones «a» de esta sección fracasan silenciosamente (o casi) si no se halla el programa ejecutado; las funciones de reemplazo «b» lanzan **`OSError`** en lugar de esto.

Además, las funciones de reemplazo que usan **`check_output()`** fracasarán con un error

`CalledProcessError` si la operación solicitada produce un código de retorno diferente de cero. La salida queda disponible en el atributo `output` de la excepción lanzada.

In the following examples, we assume that the relevant functions have already been imported from the **subprocess** module.

Cómo reemplazar la sustitución de órdenes de /bin/sh

```
output=$(mycmd myarg)
```

se convierte en:

```
output = check_output(["mycmd", "myarg"])
```

Cómo remplazar los flujos de la shell

```
output=$(dmesg | grep hda)
```

se convierte en:

```
p1 = Popen(["dmesg"], stdout=PIPE)
p2 = Popen(["grep", "hda"], stdin=p1.stdout, stdout=PIPE
p1.stdout.close()  # Allow p1 to receive a SIGPIPE if p2
output = p2.communicate()[0]
```

La llamada a `p1.stdout.close()` tras lanzar *p2* es importante para que *p1* reciba un SIGPIPE si *p2* retorna antes que *p1*.

Alternativamente, para entrada de confianza, se puede usar el propio soporte de pipeline de la shell directamente:

```
output=$(dmesg | grep hda)
```

se convierte en:

```
output = check_output("dmesg | grep hda", shell=True)
```

Cómo reemplazar `os.system()`

```
sts = os.system("mycmd" + " myarg")
# becomes
retcode = call("mycmd" + " myarg", shell=True)
```

Notas:

- No suele hacer falta llamar al programa a través de la shell.
- El valor de retorno de `call()` se codifica de forma diferente al de `os.system()`.
- The `os.system()` function ignores SIGINT and SIGQUIT signals while the command is running, but the caller must do this separately when using the **`subprocess`** module.

Un ejemplo más creíble:

```
try:
    retcode = call("mycmd" + " myarg", shell=True)
    if retcode < 0:
        print("Child was terminated by signal", -retcode
    else:
        print("Child returned", retcode, file=sys.stderr
except OSError as e:
    print("Execution failed:", e, file=sys.stderr)
```

Cómo reemplazar la familia `os.spawn`

Ejemplo de P_NOWAIT:

```
pid = os.spawnlp(os.P_NOWAIT, "/bin/mycmd", "mycmd", "my
==>
pid = Popen(["/bin/mycmd", "myarg"]).pid
```

Ejemplo de P_WAIT:

```
retcode = os.spawnlp(os.P_WAIT, "/bin/mycmd", "mycmd", "
==>
retcode = call(["/bin/mycmd", "myarg"])
```

Ejemplo de vector:

```
os.spawnvp(os.P_NOWAIT, path, args)
```

```
==>
Popen([path] + args[1:])
```

Ejemplo de entorno:

```
os.spawnlpe(os.P_NOWAIT, "/bin/mycmd", "mycmd", "myarg",
==>
Popen(["/bin/mycmd", "myarg"], env={"PATH": "/usr/bin"})
```

Replacing `os.popen()`

La gestión del código de retorno se traduce así:

```
pipe = os.popen(cmd, 'w')
...
rc = pipe.close()
if rc is not None and rc >> 8:
    print("There were some errors")
==>
process = Popen(cmd, stdin=PIPE)
...
process.stdin.close()
if process.wait() != 0:
    print("There were some errors")
```

Funciones de llamada a la shell de retrocompatibilidad

Este módulo también proporciona las siguientes funciones de compatibilidad del módulo `commands` de las versiones 2.X. Estas operaciones invocan implícitamente a la shell del sistema y no se les aplican ninguna de las garantías descritas anteriormente respecto a seguridad o consistencia en la gestión de excepciones.

subprocess.getstatusoutput(*cmd*, *, *encoding=None*, *errors=None*)

Retorna `(exitcode, output)` de ejecutar *cmd* en una shell.

Execute the string *cmd* in a shell with **`check_output()`** and return a 2-tuple `(exitcode, output)`. *encoding* and *errors* are used to decode output; see the notes on Argumentos frecuentemente empleados for more details.

Se elimina un salto de línea final de la salida. El código de salida de la orden se puede interpretar como el código de retorno del subproceso. Por ejemplo:

```
>>> subprocess.getstatusoutput('ls /bin/ls')
(0, '/bin/ls')
>>> subprocess.getstatusoutput('cat /bin/junk')
(1, 'cat: /bin/junk: No such file or directory')
>>> subprocess.getstatusoutput('/bin/junk')
(127, 'sh: /bin/junk: not found')
>>> subprocess.getstatusoutput('/bin/kill $$')
(-15, '')
```

Availability: Unix, Windows.

Distinto en la versión 3.3.4: Se añadió soporte de Windows.

La función ahora retorna `(exitcode, output)` en lugar de `(status, output)` como en Python 3.3.3 y anteriores. *exitcode* tiene el mismo valor que **`returncode`**.

Distinto en la versión 3.11: Added the *encoding* and *errors* parameters.

subprocess.getoutput(*cmd*, ***, *encoding=None*, *errors=None*)

Retorna la salida (stdout y stderr) de ejecutar *cmd* en una shell.

Como **`getstatusoutput()`**, salvo que se ignora el código de salida y el valor retornado es una cadena que contiene la salida del comando. Por ejemplo:

```
>>> subprocess.getoutput('ls /bin/ls')
'/bin/ls'
```

Availability: Unix, Windows.

Distinto en la versión 3.3.4: Se añadió soporte de Windows

Distinto en la versión 3.11: Added the *encoding* and *errors* parameters.

Notas

Timeout Behavior

When using the `timeout` parameter in functions like `run()`, `Popen.wait()`, or `Popen.communicate()`, users should be aware of the following behaviors:

1. **Process Creation Delay**: The initial process creation itself cannot be interrupted on many platform APIs. This means that even when specifying a timeout, you are not guaranteed to see a timeout exception until at least after however long process creation takes.
2. **Extremely Small Timeout Values**: Setting very small timeout values (such as a few milliseconds) may result in almost immediate `TimeoutExpired` exceptions because process creation and system scheduling inherently require time.

Cómo convertir una secuencia de argumentos a una cadena en Windows

En Windows, para convertir una secuencia *args* en una cadena que puede ser analizada con las siguientes reglas (correspondientes a las reglas que usa la biblioteca de ejecución de MS C):

1. Los argumentos se separan por espacio en blanco, que debe ser un espacio o un tabulador.
2. Una cadena entre comillas dobles se interpreta como un argumento simple, sin importar los espacios en blanco que contenga. Se puede incrustar una cadena entre comillas en un argumento.
3. Una comilla doble precedida de una barra invertida se

interpreta literalmente como una comilla doble.

4. Las barras invertidas se interpretan literalmente, salvo que precedan a una comilla doble.
5. Si las barras invertidas preceden inmediatamente a una doble comilla, cada par de barras invertidas se interpreta como una barra invertida literal. Si el número de barras invertidas es impar, la última barra invertida escapa la siguiente comilla doble según se describe en la regla 3.

Ver también

`shlex`
Módulo que proporciona una función para analizar y escapar líneas de órdenes.

Disable use of `posix_spawn()`

On Linux, **`subprocess`** defaults to using the `vfork()` system call internally when it is safe to do so rather than `fork()`. This greatly improves performance.

```
subprocess._USE_POSIX_SPAWN = False  # See CPython issue
```

It is safe to set this to false on any Python version. It will have no effect on older or newer versions where unsupported. Do not assume the attribute is available to read. Despite the name, a true value does not indicate the corresponding function will be used, only that it may be.

Presente los problemas cada vez que tenga que usar estas perillas privadas con una forma de reproducir el problema que estaba viendo. Enlace a ese problema desde un comentario en su código.

Added in version 3.8: `_USE_POSIX_SPAWN`

`sched` — Event scheduler

Código fuente: Lib/sched.py [https://github.com/python/cpython/tree/3.14/Lib/sched.py]

The **`sched`** module defines a class which implements a general purpose event scheduler:

class sched.scheduler(*timefunc=time.monotonic*, *delayfunc=time.sleep*)

La clase **`scheduler`** define una interfaz genérica para planificar eventos. Necesita dos funciones para tratar con el «mundo exterior» — *timefunc* debe poder llamarse sin argumentos y retornar un número (el «time», en cualquier unidad). La función *delayfunc* debería ser invocable con un argumento, compatible con la salida de *timefunc*, y debería retrasar tantas unidades de tiempo. *delayfunc* también se llamará con el argumento `0` después de que se ejecute cada evento para permitir que otros hilos tengan la oportunidad de ejecutarse en aplicaciones multihilo.

Distinto en la versión 3.3: Los argumentos *timefunc* and *delayfunc* son opcionales.

Distinto en la versión 3.3: **`scheduler`** La clase se puede usar de forma segura en entornos multihilo.

Ejemplo:

```
>>> import sched, time
>>> s = sched.scheduler(time.time, time.sleep)
>>> def print_time(a='default'):
...     print("From print_time", time.time(), a)
...
>>> def print_some_times():
```

```
...     print(time.time())
...     s.enter(10, 1, print_time)
...     s.enter(5, 2, print_time, argument=('positional'
...     # despite having higher priority, 'keyword' runs
...     s.enter(5, 1, print_time, kwargs={'a': 'keyword'
...     s.enterabs(1_650_000_000, 10, print_time, argume
...     s.enterabs(1_650_000_000, 5, print_time, argumen
...     s.run()
...     print(time.time())
...
>>> print_some_times()
1652342830.3640375
From print_time 1652342830.3642538 second enterabs
From print_time 1652342830.3643398 first enterabs
From print_time 1652342835.3694863 positional
From print_time 1652342835.3696074 keyword
From print_time 1652342840.369612 default
1652342840.3697174
```

Objetos de `Scheduler`

`scheduler` Las sentencias tienen los siguientes métodos y atributos:

scheduler.enterabs(*time*, *priority*, *action*, *argument=()*, *kwargs={}*)

Planifica un nuevo evento. El argumento *time* debe ser un tipo numérico compatible con el valor de retorno de la función *timefunc* que se pasa al constructor. Los eventos planificados para la misma *time* se ejecutarán en el orden de su *priority*. Un número más bajo representa una prioridad más alta.

Ejecutar el evento significa ejecutar `action(*argument, **kwargs)`. *argument* es una secuencia que contiene los argumentos posicionales para *action*. *kwargs* es un diccionario que contiene los argumentos de palabras clave para *action*.

El valor de retorno es un evento que puede usarse para una cancelación posterior del evento (ver **`cancel()`**).

Distinto en la versión 3.3: El argumento *argument* es opcional.

Distinto en la versión 3.3: Se agregó el argumento *kwargs*.

scheduler.enter(*delay*, *priority*, *action*, *argument=()*, *kwargs={}*)

Planifica un evento *delay* para más unidades de tiempo. Aparte del tiempo relativo, los otros argumentos, el efecto y el valor de retorno son los mismos que para **`enterabs()`**.

Distinto en la versión 3.3: El argumento *argument* es opcional.

Distinto en la versión 3.3: Se agregó el argumento *kwargs*.

scheduler.cancel(*event*)

Elimina el evento de la cola. Si *event* no es un evento actualmente en la cola, este método lanzará un **`ValueError`**.

scheduler.empty()

Retorna `True` si la cola de eventos está vacía.

scheduler.run(*blocking=True*)

Ejecuta todos los eventos planificados. Este método esperará (usando la función *delayfunc* enviada al constructor) al próximo evento, luego lo ejecutará y así sucesivamente hasta que no haya más eventos planificados.

Si *blocking* es falso, se ejecutan los eventos planificados que expiran mas pronto (si corresponde) y luego retorna la fecha límite de la próxima llamada programada en el planificador (si corresponde).

action o *delayfunc* pueden generar una excepción. En cualquier caso, el planificador mantendrá un estado consistente y propagará la excepción. Si *action* genera una excepción, el evento no se intentará en futuras llamadas a **`run()`**.

Si una secuencia de eventos tarda más en ejecutarse que el tiempo disponible antes del próximo evento, el planificador simplemente se retrasará. No se descartarán eventos; el código de llamada es responsable de cancelar eventos que ya no son oportunos.

Distinto en la versión 3.3: Se agregó el argumento *blocking*.

scheduler.queue

Atributo de solo lectura que retorna una lista de los próximos eventos en el orden en que se ejecutarán. Cada evento se muestra como un named tuple con los siguientes campos : *time, priority, action, argument, kwargs*.

`queue` — A synchronized queue class

Código fuente: Lib/queue.py [https://github.com/python/cpython/tree/3.14/Lib/queue.py]

The **`queue`** module implements multi-producer, multi-consumer queues. It is especially useful in threaded programming when information must be exchanged safely between multiple threads. The **`Queue`** class in this module implements all the required locking semantics.

El módulo implementa tres tipos de cola, que difieren sólo en el orden en que se recuperan las entradas. En una cola FIFO, las primeras tareas añadidas son las primeras recuperadas. En una cola LIFO, la última entrada añadida es la primera recuperada (operando como una pila). En una cola de prioridad, las entradas se mantienen ordenadas (usando el módulo **`heapq`**) y la entrada de menor valor se recupera primero.

Internamente, estos tres tipos de colas utilizan bloqueos para bloquear temporalmente los hilos que compiten entre sí; sin embargo, no están diseñadas para manejar la reposición dentro de un hilo.

Además, el módulo implementa un tipo de cola «simple» FIFO, **`SimpleQueue`**, cuya implementación específica proporciona garantías adicionales a cambio de una funcionalidad menor.

The **`queue`** module defines the following classes and exceptions:

class queue.Queue(*maxsize=0*)

Constructor para una cola FIFO. *maxsize* es un número entero que establece el límite superior del número de elementos que pueden ser colocados en la cola. La inserción se bloqueará una vez que se haya alcanzado este tamaño, hasta que se consuman los elementos de la cola. Si *maxsize* es menor o igual a cero, el tamaño de la cola es infinito.

class queue.LifoQueue(*maxsize=0*)

Constructor para una cola LIFO. *maxsize* es un número entero que establece el límite superior del número de elementos que pueden ser colocados en la cola. La inserción se bloqueará una vez que se haya alcanzado este tamaño, hasta que se consuman los elementos de la cola. Si *maxsize* es menor o igual a cero, el tamaño de la cola es infinito.

class queue.PriorityQueue(*maxsize=0*)

Constructor para una cola de prioridad. *maxsize* es un número entero que establece el límite superior del número de elementos que pueden ser colocados en la cola. La inserción se bloqueará una vez que se haya alcanzado este tamaño, hasta que se consuman los elementos de la cola. Si *maxsize* es menor o igual a cero, el tamaño de la cola es infinito.

Las entradas de menor valor se recuperan primero (la entrada de menor valor es la retornada por `min(entries)`). Un patrón típico para las entradas es una tupla en la forma: `(priority_number, data)`.

Si los elementos de *datos* no son comparables, los datos pueden envolverse en una clase que ignore el elemento de datos y sólo compare el número de prioridad:

```
from dataclasses import dataclass, field
from typing import Any

@dataclass(order=True)
class PrioritizedItem:
    priority: int
    item: Any=field(compare=False)
```

class queue.SimpleQueue

Constructor de una cola sin límites FIFO. Las colas simples carecen de funcionalidad avanzada como el seguimiento de tareas.

Added in version 3.7.

exception queue.Empty

Excepción lanzada cuando el objeto `get()` (o `get_nowait()`) que no se bloquea es llamado en un objeto `Queue` que está vacío.

exception queue.Full

Excepción lanzada cuando el objeto `put()` (o `put_nowait()`) que no se bloquea es llamado en un objeto `Queue` que está lleno.

exception queue.ShutDown

Exception raised when `put()` or `get()` is called on a `Queue` object which has been shut down.

Added in version 3.13.

Objetos de la cola

Los objetos de la cola (`Queue`, `LifoQueue`, o `PriorityQueue`) proporcionan los métodos públicos descritos a continuación.

Queue.qsize()

Retorna el tamaño aproximado de la cola. Nota, qsize() > 0 no garantiza que un get() posterior no se bloquee, ni *qsize()* < *maxsize* garantiza que put() no se bloquee.

Queue.empty()

Retorna `True` si la cola está vacía, `False` si no. Si empty() retorna `True` no garantiza que una llamada posterior a put() no se bloquee. Del mismo modo, si empty() retorna `False` no garantiza que una llamada posterior a get() no se bloquee.

Queue.full()

Retorna `True` si la cola está llena, `False` si no. Si full() retorna `True` no garantiza que una llamada posterior a get() no se bloquee. Del mismo modo, si full() retorna `False` no garantiza que una llamada posterior a put() no se bloquee.

Queue.put(*item*, *block=True*, *timeout=None*)

Pone el *item* en la cola. Si el argumento opcional *block* es verdadero y *timeout* es `None` (el predeterminado), bloquea si es necesario hasta que un espacio libre esté disponible. Si *timeout* es un número positivo, bloquea como máximo *timeout* segundos y lanza la excepción **`Full`** si no había ningún espacio libre disponible en ese tiempo. De lo contrario (*block* es falso), pone un elemento en la cola si un espacio libre está disponible inmediatamente, o bien lanza la excepción **`Full`** (*timeout* es ignorado en ese caso).

Raises **`ShutDown`** if the queue has been shut down.

Queue.put_nowait(*item*)

Equivalente a `put(item, block=False)`.

Queue.get(*block=True*, *timeout=None*)

Retira y retorna un elemento de la cola. Si los argumentos opcionales *block* son true y *timeout* es `None` (el predeterminado), bloquea si es necesario hasta que un elemento esté disponible. Si *timeout* es un número positivo, bloquea como máximo *timeout* segundos y lanza la excepción **`Empty`** si no había ningún elemento disponible en ese tiempo. De lo contrario (*block* es falso), retorna un elemento si uno está disponible inmediatamente, o bien lanza la excepción **`Empty`** (*timeout* es ignorado en ese caso).

Antes de la 3.0 en los sistemas POSIX, y para todas las versiones en Windows, si *block* es verdadero y *timeout* es `None`, esta operación entra en una espera no interrumpible en una cerradura subyacente. Esto significa que no puede haber excepciones, y en particular una SIGINT no disparará una

`KeyboardInterrupt`.

Raises **`ShutDown`** if the queue has been shut down and is empty, or if the queue has been shut down immediately.

Queue.get_nowait()

Equivalente a `get(False)`.

Se ofrecen dos métodos para apoyar el seguimiento si las tareas en cola han sido completamente procesadas por hilos *daemon* de consumo.

Queue.task_done()

Indica que una tarea anteriormente en cola está completa. Utilizado por los hilos de la cola de consumo. Por cada **`get()`** usado para recuperar una tarea, una llamada posterior a **`task_done()`** le dice a la cola que el procesamiento de la tarea está completo.

Si un **`join()`** se está bloqueando actualmente, se reanudará cuando todos los ítems hayan sido procesados (lo que significa que se recibió una llamada **`task_done()`** por cada ítem que había sido **`put()`** en la cola).

Lanza un **`ValueError`** si se llama más veces de las que hay elementos colocados en la cola.

Queue.join()

Bloquea hasta que todos los artículos de la cola se hayan obtenido y procesado.

El conteo de tareas sin terminar sube cada vez que se añade un elemento a la cola. El conteo baja cuando un hilo de consumidor llama **`task_done()`** para indicar que el elemento fue recuperado y todo el trabajo en él está completo. Cuando el conteo de tareas sin terminar cae a cero, **`join()`** se desbloquea.

Waiting for task completion

Ejemplo de cómo esperar a que se completen las tareas en cola:

```
import threading
import queue

q = queue.Queue()

def worker():
    while True:
        item = q.get()
        print(f'Working on {item}')
        print(f'Finished {item}')
        q.task_done()

# Turn-on the worker thread.
threading.Thread(target=worker, daemon=True).start()

# Send thirty task requests to the worker.
for item in range(30):
    q.put(item)

# Block until all tasks are done.
q.join()
print('All work completed')
```

Terminating queues

When no longer needed, **`Queue`** objects can be wound down until empty or terminated immediately with a hard shutdown.

Queue.shutdown(*immediate=False*)

Put a **`Queue`** instance into a shutdown mode.

The queue can no longer grow. Future calls to **`put()`** raise **`ShutDown`**. Currently blocked callers of **`put()`** will be unblocked and will raise **`ShutDown`** in the formerly blocked thread.

If *immediate* is false (the default), the queue can be wound

down normally with `get()` calls to extract tasks that have already been loaded.

And if `task_done()` is called for each remaining task, a pending `join()` will be unblocked normally.

Once the queue is empty, future calls to `get()` will raise `ShutDown`.

If *immediate* is true, the queue is terminated immediately. The queue is drained to be completely empty and the count of unfinished tasks is reduced by the number of tasks drained. If unfinished tasks is zero, callers of `join()` are unblocked. Also, blocked callers of `get()` are unblocked and will raise `ShutDown` because the queue is empty.

Use caution when using `join()` with *immediate* set to true. This unblocks the join even when no work has been done on the tasks, violating the usual invariant for joining a queue.

Added in version 3.13.

Objetos de cola simple

Los objetos `SimpleQueue` proporcionan los métodos públicos descritos a continuación.

SimpleQueue.qsize()

Retorna el tamaño aproximado de la cola. Nota, qsize() > 0 no garantiza que un get() posterior no se bloquee.

SimpleQueue.empty()

Retorna `True` si la cola está vacía, `False` si no. Si empty() retorna `False` no garantiza que una llamada posterior a get() no se bloquee.

SimpleQueue.put(*item*, *block*=*True*, *timeout*=*None*)

Pone el elemento en la cola. El método nunca se bloquea y siempre tiene éxito (excepto por posibles errores de bajo nivel como la falta de asignación de memoria). Los argumentos opcionales *block* y *timeout* son ignorados y sólo se proporcionan por compatibilidad con `Queue.put()`.

Este método tiene una implementación en C la cual ha sido usada anteriormente. Los llamados `put()` o `get()` pueden ser interrumpidos por otro llamado `put()` en el mismo hilo sin bloquear o corromper el estado interno dentro de la fila. Esto lo hace apropiado para su uso en destructores como los métodos `__del__` o las retrollamadas `weakref`.

SimpleQueue.put_nowait(*item*)

Equivalente a `put(item, block=False)`, siempre y cuando sea compatible con **`Queue.put_nowait`()`**.

SimpleQueue.get(*block=True*, *timeout=None*)

Retira y retorna un elemento de la cola. Si los argumentos opcionales *block* son true y *timeout* es `None` (el predeterminado), bloquea si es necesario hasta que un elemento esté disponible. Si *timeout* es un número positivo, bloquea como máximo *timeout* segundos y lanza la excepción `Empty` si no había ningún elemento disponible en ese tiempo. De lo contrario (*block* es falso), retorna un elemento si uno está disponible inmediatamente, o bien lanza la excepción **`Empty`** (*timeout* es ignorado en ese caso).

SimpleQueue.get_nowait()

Equivalente a `get(False)`.

Ver también

Clase `multiprocessing.Queue`
Una clase de cola para su uso en un contexto de multiprocesamiento (en lugar de multihilo).

`collections.deque` es una implementación alternativa de colas sin límites con operaciones atómicas rápidas `append()` y `popleft()` que no requieren bloqueo y también soportan indexación.

`contextvars` — Context Variables

Este módulo proporciona APIs para gestionar, almacenar y acceder a estados en el contexto local. La clase `ContextVar` se utiliza para declarar y trabajar con Variables de Contexto (*Context Variables*). La función `copy_context()` y la clase `Context` deberían ser utilizadas para gestionar el contexto actual en frameworks asíncronos.

Context managers that have state should use Context Variables instead of `threading.local()` to prevent their state from bleeding to other code unexpectedly, when used in concurrent code.

Ver **PEP 567** [https://peps.python.org/pep-0567/] para más detalles.

Added in version 3.7.

variables de contexto

class contextvars.ContextVar(*name*[, *, *default*])

Esta clase se utiliza para declarar una nueva Variable de Contexto, por ejemplo:

```
var: ContextVar[int] = ContextVar('var', default=42
```

El parámetro obligatorio *name* se utiliza para introspección y depuración.

El parámetro opcional de sólo palabra clave *default* es utilizado por `ContextVar.get()`, cuando en el contexto actual no se encuentra ningún valor para la variable.

Importante: las Variables de Contexto deberían ser creadas en lo más alto a nivel de módulo y nunca en clausura. Los objetos **`Context`** mantienen referencias a variables de contexto, lo cual no permitiría que estas variables de contexto sean limpiadas por el recolector de basura.

name

El nombre de la variable. Propiedad de sólo lectura.

Added in version 3.7.1.

get([*default*])

Retorna un valor para la variable de contexto en el contexto actual.

Si la variable no tiene ningún valor en el contexto actual, el método:

- retornará el valor del argumento *default* del método, si alguno fue dado; o
- retornará el valor por defecto de la variable de contexto, si ésta fue creada con alguno; o
- lanzará **`LookupError`**.

set(*value*)

Establece un nuevo valor para la variable de contexto en el contexto actual.

El argumento obligatorio *value* es el nuevo valor de la variable de contexto.

Retorna un objeto **`Token`** que puede utilizarse para restaurar la variable a su valor anterior, utilizando el método **`ContextVar.reset()`**.

For convenience, the token object can be used as a context manager to avoid calling **`ContextVar.reset()`** manually:

```
var = ContextVar('var', default='default value'
with var.set('new value'):
    assert var.get() == 'new value'

assert var.get() == 'default value'
```

It is a shorthand for:

```
var = ContextVar('var', default='default value'

token = var.set('new value')
try:
    assert var.get() == 'new value'
finally:
    var.reset(token)

assert var.get() == 'default value'
```

Added in version 3.14: Added support for using tokens as context managers.

reset(*token*)

Restablece la variable de contexto al valor que tenía antes de llamar al método **`ContextVar.set()`**, que creó el *token* utilizado.

Por ejemplo:

```
var = ContextVar('var')

token = var.set('new value')
# code that uses 'var'; var.get() returns 'new 
var.reset(token)

# After the reset call the var has no value aga
# var.get() would raise a LookupError.
```

The same *token* cannot be used twice.

class contextvars.Token

Token objects are returned by the `ContextVar.set()` method. They can be passed to the `ContextVar.reset()` method to revert the value of the variable to what it was before the corresponding *set*. A single token cannot reset a context variable more than once.

Tokens support the context manager protocol to automatically reset context variables. See `ContextVar.set()`.

Added in version 3.14: Added support for usage as a context manager.

var

Propiedad de sólo lectura. Apunta al objeto `ContextVar` que creó el *token*.

old_value

Propiedad de sólo lectura. Es el valor que la variable tenía antes de llamar al método `ContextVar.set()` que creó el *token*. Apunta a `Token.MISSING` si la variable no estaba establecida antes de la llamada.

MISSING

Marcador utilizado por `Token.old_value`.

Gestión de contexto manual

contextvars.copy_context()

Retorna una copia del objeto `Context` actual.

El siguiente código obtiene una copia del contexto actual e imprime todas las variables y sus valores establecidos en el contexto:

```
ctx: Context = copy_context()
print(list(ctx.items()))
```

The function has an $O(1)$ complexity, i.e. works equally fast for contexts with a few context variables and for contexts that have a lot of them.

class contextvars.Context

Mapeo de `ContextVars` con sus valores.

`Context()` crea un contexto vacío sin valores. Para obtener una copia del contexto actual, se puede utilizar la función `copy_context()`.

Each thread has its own effective stack of **`Context`** objects. The current context is the **`Context`** object at the top of the current thread's stack. All **`Context`** objects in the stacks are considered to be *entered*.

Entering a context, which can be done by calling its `run()` method, makes the context the current context by pushing it onto the top of the current thread's context stack.

Exiting from the current context, which can be done by returning from the callback passed to the `run()` method, restores the current context to what it was before the context was entered by popping the context off the top of the context stack.

Since each thread has its own context stack, `ContextVar` objects behave in a similar fashion to `threading.local()` when values are assigned in different threads.

Attempting to enter an already entered context, including contexts entered in other threads, raises a `RuntimeError`.

After exiting a context, it can later be re-entered (from any thread).

Any changes to `ContextVar` values via the `ContextVar.set()` method are recorded in the current context. The `ContextVar.get()` method returns the value associated with the current context. Exiting a context effectively reverts any changes made to context variables while the context was entered (if needed, the values can be

restored by re-entering the context).

Context implementa la interfaz **`collections.abc.Mapping`**.

run(*callable*, **args*, ***kwargs*)

Enters the Context, executes `callable(*args, **kwargs)`, then exits the Context. Returns *callable*'s return value, or propagates an exception if one occurred.

Example:

```
import contextvars

var = contextvars.ContextVar('var')
var.set('spam')
print(var.get())  # 'spam'

ctx = contextvars.copy_context()

def main():
    # 'var' was set to 'spam' before
    # calling 'copy_context()' and 'ctx.run(ma
    print(var.get())  # 'spam'
    print(ctx[var])  # 'spam'

    var.set('ham')

    # Now, after setting 'var' to 'ham':
    print(var.get())  # 'ham'
    print(ctx[var])  # 'ham'

# Any changes that the 'main' function makes t
# will be contained in 'ctx'.
ctx.run(main)

# The 'main()' function was run in the 'ctx' c
# so changes to 'var' are contained in it:
```

```
print(ctx[var])  # 'ham'

# However, outside of 'ctx', 'var' is still set
print(var.get())  # 'spam'
```

copy()

Retorna una copia superficial (*shallow copy*) del objeto de contexto.

var in context

Retorna `True` si *context* tiene un valor establecido para *var*; de lo contrario, retorna `False`.

context[var]

Retorna el valor de la variable **`ContextVar`** *var*. Si la variable no está establecida en el contexto actual, se lanzará **`KeyError`**.

get(*var*[, *default*])

Retorna el valor de *var*, si *var* tiene el valor en el objeto de contexto; de lo contrario, retorna *default*. Si *default* no es dado, retorna `None`.

iter(context)

Retorna un iterador de las variables almacenadas en el objeto de contexto.

len(proxy)

Retorna el número de variables establecidas en el objeto de contexto.

keys()

Retorna un listado de todas las variables en el objeto de contexto.

values()

Retorna un listado de los valores de todas las variables en el objeto de contexto.

items()

Retorna un listado de dos tuplas que contienen todas las variables y sus variables en el contexto actual.

Soporte asyncio

Las variables de contexto están soportadas de forma nativa en **`asyncio`** y se pueden utilizar sin ninguna configuración adicional. Por ejemplo, el siguiente código crea un servidor simple de respuesta, que utiliza una variable de contexto que hace que la dirección del cliente remoto esté disponible en la *Task* que gestiona al cliente:

```
import asyncio
import contextvars

client_addr_var = contextvars.ContextVar('client_addr')

def render_goodbye():
    # The address of the currently handled client can be
    # without passing it explicitly to this function.

    client_addr = client_addr_var.get()
    return f'Good bye, client @ {client_addr}\r\n'.encod

async def handle_request(reader, writer):
    addr = writer.transport.get_extra_info('socket').get
    client_addr_var.set(addr)

    # In any code that we call is now possible to get
    # client's address by calling 'client_addr_var.get()

    while True:
        line = await reader.readline()
        print(line)
        if not line.strip():
            break

    writer.write(b'HTTP/1.1 200 OK\r\n')  # status line
    writer.write(b'\r\n')  # headers
    writer.write(render_goodbye())  # body
    writer.close()

async def main():
    srv = await asyncio.start_server(
        handle_request, '127.0.0.1', 8081)

    async with srv:
        await srv.serve_forever()

asyncio.run(main())

# To test it you can use telnet or curl:
#     telnet 127.0.0.1 8081
#     curl 127.0.0.1:8081
```

`_thread` — Low-level threading API

Este módulo ofrece primitivas de bajo nivel para trabajar con múltiples *threads* o hilos (también llamados *light-weight processes* o *tasks*) – múltiples hilos de control compartiendo su espacio de datos global. Para sincronizar, provee «candados» simples (también llamados *mutexes* o *binary semaphores*). El módulo `threading` provee una API de manejo de hilos más fácil de usar y de más alto nivel, construida sobre este módulo.

Distinto en la versión 3.7: Este módulo solía ser opcional, pero ahora está siempre disponible.

Este módulo define las siguientes constantes y funciones:

exception _thread.error

> Lanzado ante errores específicos de un hilo.
>
> *Distinto en la versión 3.3:* Ahora es un sinónimo de la excepción incorporada `RuntimeError`.

_thread.LockType

> Este es el tipo de los objetos candado (*lock objects*).

_thread.start_new_thread(*function*, *args*[, *kwargs*])

> Inicia un nuevo hilo y retorna su identificador. El hilo ejecuta la función *function* con la lista de argumentos *args* (que debe ser una tupla). El argumento opcional *kwargs* especifica un diccionario de argumentos por palabras clave.
>
> Cuando la función retorna, el hilo finaliza silenciosamente.
>
> Cuando la función termina con una excepción no gestionada,

se invoca a **`sys.unraisablehook()`** para que gestione la excepción. El atributo *object* del argumento gancho (*hook*), es *function*. Por defecto, se muestra un seguimiento de pila y luego el hilo sale (pero los otros hilos continúan funcionando).

Cuando la función lanza una excepción **`SystemExit`**, se ignora silenciosamente.

Lanza un evento de auditoría `_thread.start_new_thread` con los argumentos `function`, `args`, `kwargs`.

Distinto en la versión 3.8: Ahora se utiliza **`sys.unraisablehook()`** para gestionar las excepciones no gestionadas.

_thread.interrupt_main(*signum=signal.SIGINT*, */*)

Simular el efecto de una señal que llega al hilo principal. Un hilo puede usar esta función para interrumpir el hilo principal, aunque no hay garantías de que la interrupción ocurrirá inmediatamente.

Si es dada, *signum* es el número de la señal a simular. Si *signum* no es dada, **`signal.SIGINT`** es simulado.

Si la señal dada no es manejada por Python (se estableció en **`signal.SIG_DFL`** o **`signal.SIG_IGN`**), esta función no hace nada.

Distinto en la versión 3.10: Se agrega el argumento *signum* para personalizar el número de señal.

Nota

Esto no emite la señal correspondiente, sino que programa una llamada al controlador asociado (si existe). Si realmente se quiere emitir la señal, se utiliza **`signal.raise_signal()`**.

_thread.exit()

Lanza la excepción `SystemExit`. Cuando no es gestionada, causa que el hilo salga silenciosamente.

_thread.allocate_lock()

Retorna un nuevo objeto candado (*lock object*). Los métodos de los candados se describen más abajo. El candado está abierto al inicio.

_thread.get_ident()

Retorna el "identificador de hilo" (*thread identifier*) del hilo actual. Es un entero distinto de cero. Su valor no tiene un significado directo, tiene la intención de ser utilizada como una *cookie* mágica para, por ejemplo, indexar un diccionario con datos específicos del hilo. Los identificadores de hilo pueden reciclarse cuando un hilo sale y otro se crea.

_thread.get_native_id()

Retorna el ID de hilo nativo integral del hilo asignado por el kernel. Es un entero no-negativo. Su valor puede utilizarse para identificar inequívocamente este hilo en particular en todo el sistema (hasta que el hilo termine, luego de lo cual el valor puede ser reciclado por el Sistema Operativo).

Availability: Windows, FreeBSD, Linux, macOS, OpenBSD, NetBSD, AIX, DragonFlyBSD, GNU/kFreeBSD.

Added in version 3.8.

Distinto en la versión 3.13: Added support for GNU/kFreeBSD.

_thread.stack_size([*size*])

Retorna el tamaño de la pila del hilo (*thread stack*) utilizada al crear nuevos hilos. El argumento opcional *size* especifica el tamaño de la pila a utilizar en los hilos que se creen a continuación, y debe ser 0 (utiliza el valor por defecto de la plataforma o el configurado) o un entero positivo de al menos 32768 (32KiB). Si *size* no se especifica, se utiliza 0. Si no está soportado el cambio del tamaño de pila del hilo, se lanza una excepción **`RuntimeError`**. Si la pila especificada es inválida se lanza un **`ValueError`** y el tamaño de la pila no se modifica. 32KiB es actualmente el menor valor soportado para el tamaño de la pila, para garantizar suficiente espacio en la misma para que quepa el propio intérprete. Tenga en cuenta que alguna plataformas pueden tener restricciones particulares en los valores para el tamaño de la pila, como requerir un mínimo que supere los 32KiB, o requerir una asignación en múltiplos del tamaño de página de memoria del sistema. Es necesario consultar la documentación de la plataforma para mayor información (son habituales las páginas de 4KiB; usar múltiplos de 4096 para el tamaño de pila es la estrategia sugerida si no se cuenta con información más específica).

Availability: Windows, pthreads.

Plataformas Unix con soporte para hilos POSIX.

_thread.TIMEOUT_MAX

El máximo valor permitido para el parámetro *timeout* de **`Lock.acquire`**. Especificar un tiempo de espera (*timeout*) mayor que este valor lanzará una excepción **`OverflowError`**.

Added in version 3.2.

Los objetos candado (*lock objects*) tienen los siguientes métodos:

lock.acquire(*blocking=True*, *timeout=-1*)

Sin ningún argumento opcional, este método adquiere el candado incondicionalmente, si es necesario esperando que éste sea liberado por otro hilo (solamente un hilo por vez puede adquirir un candado; para eso existen).

If the *blocking* argument is present, the action depends on its value: if it is false, the lock is only acquired if it can be acquired immediately without waiting, while if it is true, the lock is acquired unconditionally as above.

If the floating-point *timeout* argument is present and positive, it specifies the maximum wait time in seconds before returning. A negative *timeout* argument specifies an unbounded wait. You cannot specify a *timeout* if *blocking* is false.

El valor de retorno es `True` si el candado (*lock*) se adquirió exitosamente, `False` de lo contrario.

Distinto en la versión 3.2: El parámetro *timeout* es nuevo.

Distinto en la versión 3.2: La adquisición de candados ahora puede ser interrumpida por señales en POSIX.

Distinto en la versión 3.14: Lock acquires can now be interrupted by signals on Windows.

lock.release()

Libera el candado. El candado debe haber sido adquirido previamente, pero no necesariamente por el mismo hilo.

lock.locked()

Retorna el estado del candado: `True` si ha sido adquirido por algún hilo, `False` de lo contrario.

Además de estos métodos, los objetos candado pueden ser utilizados mediante la declaración **`with`**, por ejemplo:

```
import _thread

a_lock = _thread.allocate_lock()

with a_lock:
    print("a_lock is locked while this executes")
```

Salvedades:

- Interrupts always go to the main thread (the **`KeyboardInterrupt`** exception will be received by that thread.)

- Invocar a `sys.exit()` o lanzar la excepción `SystemExit` equivale a invocar `_thread.exit()`.
- Cuando el hilo principal sale, ¿sobreviven los otros hilos? Depende de cómo esté definido por el sistema. En la mayoría de los sistemas, los hilos se cierran inmediatamente (*killed*), sin ejecutar las cláusulas `try` … `finally` o los destructores del objeto.

Comunicación en redes y entre procesos

Los módulos descritos en este capítulo proveen los mecanismos para la comunicación en red y entre procesos.

Algunos módulos solo funcionan para dos procesos que están en una misma máquina, e.g. `signal` y `mmap`. Otros módulos soportan protocolos de red que dos o mas procesos pueden utilizar para comunicarse entre máquinas.

La lista de módulos descritos en este capítulo es:

- `asyncio` — Asynchronous I/O
- `socket` — Low-level networking interface
- `ssl` — TLS/SSL wrapper for socket objects
- `select` — Waiting for I/O completion
- `selectors` — High-level I/O multiplexing
- `signal` — Set handlers for asynchronous events
- `mmap` — Memory-mapped file support

`asyncio` — Asynchronous I/O

¡Hola Mundo!

```
import asyncio

async def main():
    print('Hello ...')
    await asyncio.sleep(1)
    print('... World!')

asyncio.run(main())
```

asyncio es una biblioteca para escribir código **concurrente** utilizando la sintaxis **async/await**.

asyncio es utilizado como base en múltiples *frameworks* asíncronos de Python y provee un alto rendimiento en redes y servidores web, bibliotecas de conexión de base de datos, colas de tareas distribuidas, etc.

asyncio suele encajar perfectamente para operaciones con límite de E/S y código de red **estructurado** de alto nivel.

Ver también

A Conceptual Overview of asyncio
Explanation of the fundamentals of asyncio.

asyncio provee un conjunto de *APIs* de **alto nivel** para:

- ejecutar corutinas de Python de manera concurrente y tener control total sobre su ejecución;
- realizar redes E/S y comunicación entre procesos(IPC);
- controlar subprocesos;

- distribuir tareas a través de colas;
- sincronizar código concurrente;

Adicionalmente, existen *APIs* de **bajo nivel** para *desarrolladores de bibliotecas y frameworks* para:

- create and manage event loops, which provide asynchronous APIs for networking, running subprocesses, handling OS signals, etc;
- implementar protocolos eficientes utilizando transportes;
- Bibliotecas puente basadas en retrollamadas y código con sintaxis *async/wait.*

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

asyncio REPL

You can experiment with an `asyncio` concurrent context in the REPL:

```
$ python -m asyncio
asyncio REPL ...
Use "await" directly instead of "asyncio.run()".
Type "help", "copyright", "credits" or "license" for mor
>>> import asyncio
>>> await asyncio.sleep(10, result='hello')
'hello'
```

This REPL provides limited compatibility with **`PYTHON_BASIC_REPL`**. It is recommended that the default REPL is used for full functionality and the latest features.

Raises an auditing event `cpython.run_stdin` with no arguments.

Distinto en la versión 3.12.5: (also 3.11.10, 3.10.15, 3.9.20, and 3.8.20) Emits audit events.

Distinto en la versión 3.13: Uses PyREPL if possible, in which case **`PYTHONSTARTUP`** is also executed. Emits audit events.

Referencias

APIs de alto nivel

- Runners
- Coroutines and tasks
- Streams
- Primitivas de sincronización
- Sub-procesos
- Colas
- Excepciones
- Call Graph Introspection

APIs de bajo nivel

- Event loop
- Futures
- Transportes y protocolos
- Políticas
- Soporte de plataforma
- Extensión

Guías y tutoriales

- Índice de API de alto nivel
- Índice de API de bajo nivel
- Desarrollando con asyncio

Nota

El código fuente para asyncio puede encontrarse en Lib/asyncio/ [https://github.com/python/cpython/tree/3.14/Lib/asyncio/].

Runners

Código fuente: Lib/asyncio/runners.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/runners.py]

Esta sección describe las primitivas asyncio de alto nivel para ejecutar código asyncio.

Están construidos sobre un event loop con el objetivo de simplificar el uso de código async para escenarios comunes de alta difusión.

- Ejecutando un programa asyncio
- Gestor de contexto del runner
- Manejando interrupciones de teclado

Ejecutando un programa asyncio

asyncio.run(*coro*, **, *debug=None*, *loop_factory=None*)

Execute *coro* in an asyncio event loop and return the result.

The argument can be any awaitable object.

This function runs the awaitable, taking care of managing the asyncio event loop, *finalizing asynchronous generators*, and closing the executor.

Esta función no puede ser llamada cuando otro bucle de eventos asyncio está corriendo en el mismo hilo.

Si *debug* es `True`, el bucle de eventos se ejecutará en modo depuración. `False` deshabilita el modo depuración de manera explícita. `None` se usa para respetar la configuración global Modo depuración.

If *loop_factory* is not `None`, it is used to create a new event loop; otherwise **`asyncio.new_event_loop()`** is used. The loop is closed at the end. This function should be used as a main entry point for asyncio programs, and should ideally only be called once. It is recommended to use *loop_factory* to configure the event loop instead of policies. Passing **`asyncio.EventLoop`** allows running asyncio without the policy system.

Al ejecutor se le da un tiempo de espera de 5 minutos para apagarse. Si el ejecutor no ha finalizado en ese tiempo, se emite una advertencia y se cierra el ejecutor.

Ejemplo:

```
async def main():
    await asyncio.sleep(1)
    print('hello')

asyncio.run(main())
```

Added in version 3.7.

Distinto en la versión 3.9: Actualizado para usar **`loop.shutdown_default_executor()`**.

Distinto en la versión 3.10: *debug* es `None` por defecto para respetar la configuración global del modo depuración.

Distinto en la versión 3.12: Añadido el parámetro *loop_factory*.

Distinto en la versión 3.14: *coro* can be any awaitable object.

Nota

The **`asyncio`** policy system is deprecated and will be removed in Python 3.16; from there on, an explicit *loop_factory* is needed to configure the event loop.

Gestor de contexto del runner

class asyncio.Runner(**, debug=None, loop_factory=None*)

Un gestor de contexto que simplifica *múltiples* llamadas a funciones asíncronas en el mismo contexto.

A veces varias funciones asíncronas de alto nivel deben ser llamadas en el mismo event loop y **`contextvars.Context`**.

Si *debug* es `True`, el bucle de eventos se ejecutará en modo depuración. `False` deshabilita el modo depuración de manera explícita. `None` se usa para respetar la configuración global Modo depuración.

loop_factory puede ser usado para redefinir la creación de bucles. Es responsabilidad del *loop_factory* establecer el bucle creado como el actual. Por defecto **`asyncio.new_event_loop()`** es usado y configura el nuevo bucle de eventos como el actual con **`asyncio.set_event_loop()`** si *loop_factory* es `None`.

Basically, **`asyncio.run()`** example can be rewritten with the runner usage:

```
async def main():
    await asyncio.sleep(1)
    print('hello')

with asyncio.Runner() as runner:
    runner.run(main())
```

Added in version 3.11.

run(*coro, *, context=None*)

Execute *coro* in the embedded event loop.

The argument can be any awaitable object.

If the argument is a coroutine, it is wrapped in a Task.

An optional keyword-only *context* argument allows specifying a custom **`contextvars.Context`** for the code to run in. The runner's default context is used if context is `None`.

Returns the awaitable's result or raises an exception.

Esta función no puede ser llamada cuando otro bucle de eventos asyncio está corriendo en el mismo hilo.

Distinto en la versión 3.14: coro can be any awaitable object.

close()

Cierra el runner.

Termina los generadores asíncronos, apaga el ejecutor por defecto, cierra el bucle de eventos y libera el `contextvars.Context` embebido.

get_loop()

Retorna el bucle de eventos asociado a la instancia del runner.

Nota

`Runner` usa una estrategia de inicialización perezosa, su constructor no inicializa las estructuras de bajo nivel subyacentes.

El *loop* y el *context* embebidos son creados al entrar al cuerpo `with` o en la primera llamada a `run()` o a `get_loop()`.

Manejando interrupciones de teclado

Added in version 3.11.

Cuando `signal.SIGINT` es lanzada por `Ctrl-C`, la excepción `KeyboardInterrupt` es lanzada en el hilo principal por defecto.

Sin embargo, esto no funciona con `asyncio` porque puede interrumpir las funciones internas a asyncio e impedir la salida del programa.

Para mitigar este problema, `asyncio` maneja `signal.SIGINT` de la siguiente forma:

1. `asyncio.Runner.run()` instala un administrador `signal.SIGINT` personalizado antes que cualquier código de usuario sea ejecutado y lo remueve a la salida de la función.
2. La `Runner` crea la tarea principal que será pasada a la co-rutina para su ejecución.
3. Cuando `signal.SIGINT` es lanzada por `Ctrl-C`, el administrador de señales personalizado cancela la tarea principal llamando a `asyncio.Task.cancel()` que lanza `asyncio.CancelledError` dentro de la tarea principal. Esto hace que la pila de Python se desenrolle, los bloques `try/except` y `try/finally` se pueden utilizar para la limpieza de recursos. Luego que la tarea principal es cancelada, `asyncio.Runner.run()` lanza `KeyboardInterrupt`.
4. Un usuario podría escribir un bucle cerrado que no puede ser interrumpido por `asyncio.Task.cancel()`, en cuyo caso la segunda llamada a `Ctrl-C` lanza inmediatamente `KeyboardInterrupt` sin cancelar la tarea principal.

Coroutines and tasks

Esta sección describe las API de asyncio de alto nivel para trabajar con corrutinas y tareas.

- Corrutinas
- Esperables
- Creating tasks
- Task cancellation
- Task groups
- Durmiendo
- Running tasks concurrently
- Eager task factory
- Shielding from cancellation
- Tiempo agotado
- Waiting primitives
- Running in threads
- Scheduling from other threads
- Introspección
- Task object

Corrutinas

Source code: Lib/asyncio/coroutines.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/coroutines.py]

Coroutines declarado con la sintaxis async/await es la forma preferida de escribir aplicaciones asyncio. Por ejemplo, el siguiente fragmento de código imprime «hola», espera 1 segundo y luego imprime «mundo»:

```
>>> import asyncio

>>> async def main():
...     print('hello')
...     await asyncio.sleep(1)
...     print('world')

>>> asyncio.run(main())
hello
world
```

Tenga en cuenta que simplemente llamando a una corrutina no programará para que se ejecute:

```
>>> main()
<coroutine object main at 0x1053bb7c8>
```

Para ejecutar realmente una corutina, asyncio proporciona los siguientes mecanismos:

- La función `asyncio.run()` para ejecutar la función de punto de entrada de nivel superior «main()» (consulte el ejemplo anterior.)
- Esperando en una corrutina. El siguiente fragmento de código imprimirá «hola» después de esperar 1 segundo y luego imprimirá «mundo» después de esperar *otros* 2 segundos:

  ```
  import asyncio
  import time

  async def say_after(delay, what):
      await asyncio.sleep(delay)
      print(what)

  async def main():
      print(f"started at {time.strftime('%X')}")

      await say_after(1, 'hello')
      await say_after(2, 'world')

      print(f"finished at {time.strftime('%X')}")

  asyncio.run(main())
  ```

Salida esperada:

```
started at 17:13:52
hello
world
finished at 17:13:55
```

- La función **`asyncio.create_task()`** para ejecutar corrutinas concurrentemente como asyncio **`Tasks`**.

  Modifiquemos el ejemplo anterior y ejecutemos dos corrutinas `say_after` *concurrentemente*:

  ```
  async def main():
      task1 = asyncio.create_task(
          say_after(1, 'hello'))

      task2 = asyncio.create_task(
          say_after(2, 'world'))

      print(f"started at {time.strftime('%X')}")

      # Wait until both tasks are completed (should t
      # around 2 seconds.)
      await task1
      await task2

      print(f"finished at {time.strftime('%X')}")
  ```

  Tenga en cuenta que la salida esperada ahora muestra que el fragmento de código se ejecuta 1 segundo más rápido que antes:

  ```
  started at 17:14:32
  hello
  world
  finished at 17:14:34
  ```

- La clase **`asyncio.TaskGroup`** proporciona una alternativa más moderna a **`create_task()`**. Usando esta API, el último ejemplo se convierte en:

```
async def main():
    async with asyncio.TaskGroup() as tg:
        task1 = tg.create_task(
            say_after(1, 'hello'))

        task2 = tg.create_task(
            say_after(2, 'world'))

        print(f"started at {time.strftime('%X')}")

    # The await is implicit when the context manage

    print(f"finished at {time.strftime('%X')}")
```

El tiempo y la salida deben ser los mismos que para la versión anterior.

Added in version 3.11: `asyncio.TaskGroup`.

Esperables

Decimos que un objeto es un objeto **esperable** si se puede utilizar en una expresión `await`. Muchas API de asyncio están diseñadas para aceptar los valores esperables.

Hay tres tipos principales de objetos *esperables*: **corrutinas**, **Tareas** y **Futures**.

Corrutinas

Las corrutinas de Python son *esperables* y por lo tanto se pueden esperar de otras corrutinas:

```
import asyncio

async def nested():
    return 42

async def main():
    # Nothing happens if we just call "nested()".
    # A coroutine object is created but not awaited,
    # so it *won't run at all*.
    nested()  # will raise a "RuntimeWarning".

    # Let's do it differently now and await it:
    print(await nested())  # will print "42".

asyncio.run(main())
```

Importante

En esta documentación se puede utilizar el término «corrutina» para dos conceptos estrechamente relacionados:

- una *función corrutina*: una función **`async def`**;
- un *objeto corrutina*: un objeto retornado llamando a una *función corrutina*.

Tareas

Las tareas se utilizan para programar corrutinas *concurrentemente*.

Cuando una corrutina se envuelve en una *Tarea* con funciones como **`asyncio.create_task()`** la corrutina se programa automáticamente para ejecutarse pronto:

```
import asyncio

async def nested():
    return 42

async def main():
    # Schedule nested() to run soon concurrently
    # with "main()".
    task = asyncio.create_task(nested())

    # "task" can now be used to cancel "nested()", or
    # can simply be awaited to wait until it is complete
    await task

asyncio.run(main())
```

Futures

Un `Future` es un objeto esperable especial de **bajo-nivel** que representa un **resultado eventual** de una operación asíncrona.

Cuando un objeto Future es *esperado* significa que la corrutina esperará hasta que el Future se resuelva en algún otro lugar.

Los objetos Future de asyncio son necesarios para permitir que el código basado en retro llamada se use con async/await.

Normalmente , **no es necesario** crear objetos Future en el código de nivel de aplicación.

Los objetos Future, a veces expuestos por bibliotecas y algunas API de asyncio, pueden ser esperados:

```
async def main():
    await function_that_returns_a_future_object()

    # this is also valid:
    await asyncio.gather(
        function_that_returns_a_future_object(),
        some_python_coroutine()
    )
```

Un buen ejemplo de una función de bajo nivel que retorna un objeto Future es `loop.run_in_executor()`.

Creating tasks

Source code: Lib/asyncio/tasks.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/tasks.py]

---

asyncio.create_task(*coro*, ***, *name=None*, *context=None*, *eager_start=None*, ***kwargs*)

Envuelve una coroutine *coro* en una `Task` y programa su ejecución. Retorna el objeto Tarea.

The full function signature is largely the same as that of the `Task` constructor (or factory) - all of the keyword arguments to this function are passed through to that interface.

Un argumento *context* opcional de solo palabra clave permite especificar un `contextvars.Context` personalizado para que se ejecute el *coro*. La copia de contexto actual se crea cuando no se proporciona *context*.

An optional keyword-only *eager_start* argument allows specifying if the task should execute eagerly during the call to create_task, or be scheduled later. If *eager_start* is not passed the mode set by `loop.set_task_factory()` will be used.

La tarea se ejecuta en el bucle retornado por `get_running_loop()`, `RuntimeError` se genera si no hay ningún bucle en ejecución en el subproceso actual.

Nota

`asyncio.TaskGroup.create_task()` es una alternativa más nueva que permite una espera conveniente para un grupo de tareas relacionadas.

Importante

Guarde una referencia al resultado de esta función, para evitar que una tarea desaparezca en medio de la ejecución. El bucle de eventos solo mantiene referencias débiles a las tareas. Una tarea a la que no se hace referencia en ningún otro lugar puede ser recolectada por el recolector de basura en cualquier momento, incluso antes de que se complete. Para tareas confiables en segundo plano, de tipo «lanzar y olvidar», reúnalas en una colección:

```
background_tasks = set()

for i in range(10):
    task = asyncio.create_task(some_coro(param=i))

    # Add task to the set. This creates a strong r
    background_tasks.add(task)

    # To prevent keeping references to finished ta
    # make each task remove its own reference from
    # completion:
    task.add_done_callback(background_tasks.discar
```

Added in version 3.7.

Distinto en la versión 3.8: Se ha añadido el parámetro *name*.

Distinto en la versión 3.11: Se ha añadido el parámetro *context*.

Distinto en la versión 3.14: Added the *eager_start* parameter by passing on all *kwargs*.

Task cancellation

Las tareas se pueden cancelar de forma fácil y segura. Cuando se cancela una tarea, se generará `asyncio.CancelledError` en la tarea en la próxima oportunidad.

Se recomienda que las corrutinas utilicen bloques `try/finally` para realizar de forma sólida la lógica de limpieza. En caso de que `asyncio.CancelledError` se detecte explícitamente, generalmente debería propagarse cuando se complete la limpieza. **`asyncio.CancelledError`** subclasifica directamente a `BaseException`, por lo que la mayor parte del código no necesitará tenerlo en cuenta.

Los componentes asyncio que permiten la simultaneidad estructurada, como **`asyncio.TaskGroup`** y **`asyncio.timeout()`**, se implementan mediante cancelación internamente y podrían comportarse mal si una rutina traga **`asyncio.CancelledError`**. De manera similar, el código de usuario generalmente no debería llamar a **`uncancel`**. Sin embargo, en los casos en los que realmente se desea suprimir **`asyncio.CancelledError`**, es necesario llamar también a `uncancel()` para eliminar completamente el estado de cancelación.

Task groups

Los grupos de tareas combinan una API de creación de tareas con una forma conveniente y confiable de esperar a que finalicen todas las tareas del grupo.

class asyncio.TaskGroup

Un asynchronous context manager que contiene un grupo de tareas. Las tareas se pueden agregar al grupo usando **`create_task()`**. Se esperan todas las tareas cuando sale el administrador de contexto.

Added in version 3.11.

create_task(*coro*, ***, *name=None*, *context=None*, *eager_start=None*, ***kwargs*)

Create a task in this task group. The signature matches that of **`asyncio.create_task()`**. If the task group is inactive (e.g. not yet entered, already finished, or in the process of shutting down), we will close the given `coro`.

Distinto en la versión 3.13: Close the given coroutine if the task group is not active.

Distinto en la versión 3.14: Passes on all *kwargs* to **`loop.create_task()`**

Ejemplo:

```
async def main():
    async with asyncio.TaskGroup() as tg:
        task1 = tg.create_task(some_coro(...))
        task2 = tg.create_task(another_coro(...))
    print(f"Both tasks have completed now: {task1.result
```

La instrucción `async with` esperará a que finalicen todas las tareas del grupo. Mientras espera, aún se pueden agregar nuevas tareas al grupo (por ejemplo, pasando `tg` a una de las corrutinas y llamando a `tg.create_task()` en esa corrutina). Una vez finalizada la última tarea y salido del bloque `async with`, no se podrán añadir nuevas tareas al grupo.

La primera vez que alguna de las tareas pertenecientes al grupo falla con una excepción que no sea **`asyncio.CancelledError`**, las tareas restantes del grupo se cancelan. No se pueden añadir más tareas al grupo. En este punto, si el cuerpo de la instrucción `async with` aún está activo (es decir, aún no se ha llamado a **`__aexit__()`**), la tarea que contiene directamente la instrucción `async with` también se cancela. El **`asyncio.CancelledError`** resultante interrumpirá un `await`, pero no saldrá de la instrucción `async with` que lo contiene.

Una vez que todas las tareas han finalizado, si alguna tarea ha fallado con una excepción que no sea **`asyncio.CancelledError`**, esas excepciones se combinan en un **`ExceptionGroup`** o **`BaseExceptionGroup`** (según corresponda; consulte su documentación) que luego se genera.

Dos excepciones básicas se tratan de manera especial: si alguna tarea falla con **`KeyboardInterrupt`** o **`SystemExit`**, el grupo de tareas aún cancela las tareas restantes y las espera, pero luego se vuelve a generar el **`KeyboardInterrupt`** o **`SystemExit`** inicial en lugar de **`ExceptionGroup`** o **`BaseExceptionGroup`**.

Si el cuerpo de la instrucción `async with` finaliza con una excepción (por lo que se llama a **`__aexit__()`** con un conjunto de excepciones), esto se trata igual que si una de las tareas fallara: las tareas restantes se cancelan y luego se esperan, y las excepciones de no cancelación se agrupan en un grupo de excepción y se generan. La excepción pasada a `__aexit__()`, a menos que sea `asyncio.CancelledError`, también se incluye en el grupo de excepciones. Se hace el mismo caso especial para `KeyboardInterrupt` y `SystemExit` que en el párrafo anterior.

Task groups are careful not to mix up the internal cancellation used to «wake up» their `__aexit__()` with cancellation requests for the task in which they are running made by other parties. In particular, when one task group is syntactically nested in another, and both experience an exception in one of their child tasks simultaneously, the inner task group will process its exceptions, and then the outer task group will receive another cancellation and process its own exceptions.

In the case where a task group is cancelled externally and also must raise an `ExceptionGroup`, it will call the parent task's `cancel()` method. This ensures that a `asyncio.CancelledError` will be raised at the next `await`, so the cancellation is not lost.

Task groups preserve the cancellation count reported by `asyncio.Task.cancelling()`.

Distinto en la versión 3.13: Improved handling of simultaneous internal and external cancellations and correct preservation of cancellation counts.

Terminating a task group

While terminating a task group is not natively supported by the standard library, termination can be achieved by adding an exception-raising task to the task group and ignoring the raised exception:

```
import asyncio
from asyncio import TaskGroup

class TerminateTaskGroup(Exception):
    """Exception raised to terminate a task group."""
```

```
async def force_terminate_task_group():
    """Used to force termination of a task group."""
    raise TerminateTaskGroup()

async def job(task_id, sleep_time):
    print(f'Task {task_id}: start')
    await asyncio.sleep(sleep_time)
    print(f'Task {task_id}: done')

async def main():
    try:
        async with TaskGroup() as group:
            # spawn some tasks
            group.create_task(job(1, 0.5))
            group.create_task(job(2, 1.5))
            # sleep for 1 second
            await asyncio.sleep(1)
            # add an exception-raising task to force the
            group.create_task(force_terminate_task_group
    except* TerminateTaskGroup:
        pass

asyncio.run(main())
```

Expected output:

```
Task 1: start
Task 2: start
Task 1: done
```

Durmiendo

async asyncio.sleep(*delay*, *result=None*)

Bloquea por *delay* segundos.

Si se proporciona *result*, se retorna al autor de la llamada cuando se completa la corrutina.

`sleep()` siempre suspende la tarea actual, permitiendo que se ejecuten otras tareas.

Establecer el retraso en 0 proporciona una ruta optimizada para permitir que se ejecuten otras tareas. Esto puede ser utilizado por funciones de ejecución prolongada para evitar bloquear el bucle de eventos durante toda la duración de la llamada a la función.

Ejemplo de una rutina que muestra la fecha actual cada segundo durante 5 segundos:

```
import asyncio
import datetime as dt

async def display_date():
    loop = asyncio.get_running_loop()
    end_time = loop.time() + 5.0
    while True:
        print(dt.datetime.now())
        if (loop.time() + 1.0) >= end_time:
            break
        await asyncio.sleep(1)

asyncio.run(display_date())
```

Distinto en la versión 3.10: Se quitó el parámetro *loop*.

Distinto en la versión 3.13: Raises **`ValueError`** if *delay* is **`nan`**.

Running tasks concurrently

awaitable asyncio.gather(**aws*, *return_exceptions=False*)

Ejecute objetos esperables en la secuencia *aws* de forma *concurrently*.

Si cualquier esperable en *aws* es una corrutina, se programa automáticamente como una Tarea.

Si todos los esperables se completan correctamente, el resultado es una lista agregada de valores retornados. El orden de los valores de resultado corresponde al orden de esperables en *aws*.

Si *return_exceptions* es `False` (valor predeterminado), la primera excepción provocada se propaga inmediatamente a la tarea que espera en `gather()`. Otros esperables en la secuencia *aws* **no se cancelarán** y continuarán ejecutándose.

Si *return_exceptions* es `True`, las excepciones se tratan igual que los resultados correctos y se agregan en la lista de resultados.

Si `gather()` es *cancelado*, todos los esperables enviados (que aún no se han completado) también se *cancelan*.

Si alguna Tarea o Future de la secuencia *aws* se *cancela*, se trata como si se lanzara **`CancelledError`** – la llamada `gather()` **no** se cancela en este caso. Esto es para evitar la cancelación de una Tarea/Future enviada para hacer que otras Tareas/Futures sean canceladas.

Nota

A new alternative to create and run tasks concurrently and wait for their completion is **`asyncio.TaskGroup`**. *TaskGroup* provides stronger safety guarantees than *gather* for scheduling a nesting of subtasks: if a task (or a subtask, a task scheduled by a task) raises an exception, *TaskGroup* will, while *gather* will not, cancel the remaining scheduled tasks.

Ejemplo:

```
import asyncio

async def factorial(name, number):
    f = 1
    for i in range(2, number + 1):
        print(f"Task {name}: Compute factorial({num
        await asyncio.sleep(1)
        f *= i
    print(f"Task {name}: factorial({number}) = {f}"
    return f

async def main():
    # Schedule three calls *concurrently*:
    L = await asyncio.gather(
        factorial("A", 2),
        factorial("B", 3),
        factorial("C", 4),
    )
    print(L)

asyncio.run(main())

# Expected output:
#
#     Task A: Compute factorial(2), currently i=2..
#     Task B: Compute factorial(3), currently i=2..
#     Task C: Compute factorial(4), currently i=2..
#     Task A: factorial(2) = 2
#     Task B: Compute factorial(3), currently i=3..
#     Task C: Compute factorial(4), currently i=3..
#     Task B: factorial(3) = 6
#     Task C: Compute factorial(4), currently i=4..
#     Task C: factorial(4) = 24
#     [2, 6, 24]
```

Nota

If *return_exceptions* is false, cancelling gather() after it has been marked done won't cancel any submitted awaitables. For instance, gather can be marked done after propagating an exception to the caller, therefore, calling `gather.cancel()` after catching an exception (raised by one of the awaitables) from gather won't cancel any other awaitables.

Distinto en la versión 3.7: Si se cancela el propio *gather*, la cancelación se propaga independientemente de *return_exceptions*.

Distinto en la versión 3.10: Se quitó el parámetro *loop*.

Obsoleto desde la versión 3.10: Se emite una advertencia de obsolescencia si no se proporcionan argumentos posicionales o no todos los argumentos posicionales son objetos de tipo Future y no hay un bucle de eventos en ejecución.

Eager task factory

asyncio.eager_task_factory(*loop*, *coro*, ***, *name=None*, *context=None*)

Una fábrica de tareas para una ejecución entusiasta de tareas.

Cuando se utiliza esta fábrica (a través de `loop.set_task_factory(asyncio.eager_task_factory)`), las corrutinas comienzan a ejecutarse sincrónicamente durante la construcción de `Task`. Las tareas sólo se programan en el bucle de eventos si se bloquean. Esto puede suponer una mejora del rendimiento, ya que se evita la sobrecarga de la programación del bucle para las corrutinas que se completan sincrónicamente.

Un ejemplo común en el que esto resulta beneficioso son las rutinas que emplean almacenamiento en caché o memorización para evitar E/S reales cuando sea posible.

Nota

La ejecución inmediata de la corrutina es un cambio semántico. Si la rutina regresa o se activa, la tarea nunca se programa en el bucle de eventos. Si la ejecución de la rutina se bloquea, la tarea se programa en el bucle de eventos. Este cambio puede introducir cambios de comportamiento en las aplicaciones existentes. Por ejemplo, es probable que cambie el orden de ejecución de las tareas de la aplicación.

Added in version 3.12.

asyncio.create_eager_task_factory(*custom_task_constructor*)

Cree una fábrica de tareas entusiastas, similar a **`eager_task_factory()`**, utilizando el *custom_task_constructor* proporcionado al crear una nueva tarea en lugar del **`Task`** predeterminado.

custom_task_constructor debe ser un *callable* con la firma que coincida con la firma de **`Task.__init__`**. El invocable debe devolver un objeto compatible con **`asyncio.Task`**.

Esta función devuelve un *callable* destinado a ser utilizado como fábrica de tareas de un bucle de eventos a través de **`loop.set_task_factory(factory)`**).

Added in version 3.12.

Shielding from cancellation

awaitable asyncio.shield(*aw*)

Protege un objeto esperable de ser **`cancelado`**.

Si *aw* es una corrutina, se programa automáticamente como una Tarea.

La declaración:

```
task = asyncio.create_task(something())
res = await shield(task)
```

es equivalente a:

```
res = await something()
```

excepto que si la corrutina que lo contiene se cancela, la tarea que se ejecuta en `something()` no se cancela. Desde el punto de vista de `something()`, la cancelación no ocurrió. Aunque su invocador siga cancelado, por lo que la expresión «await» sigue generando un **`CancelledError`**.

Si `something()` se cancela por otros medios (es decir, desde dentro de sí mismo) eso también cancelaría `shield()`.

Si se desea ignorar por completo la cancelación (no se recomienda) la función `shield()` debe combinarse con una cláusula try/except, como se indica a continuación:

```
task = asyncio.create_task(something())
try:
    res = await shield(task)
except CancelledError:
    res = None
```

Importante

Guarde una referencia a las tareas pasadas a esta función, para evitar que una tarea desaparezca a mitad de la ejecución. El bucle de eventos solo mantiene referencias débiles a las tareas. Una tarea a la que no se hace referencia en ningún otro lugar puede ser recolectada por el recolector de basura en cualquier momento, incluso antes de que se complete.

Distinto en la versión 3.10: Se quitó el parámetro *loop*.

Obsoleto desde la versión 3.10: Se emite una advertencia de obsolescencia si *aw* no es un objeto similares a Futures y no hay un bucle de eventos en ejecución.

Tiempo agotado

asyncio.timeout(*delay*)

Un asynchronous context manager que se puede usar para limitar la cantidad de tiempo que se pasa esperando algo.

delay puede ser `None` o un número flotante/int de segundos de espera. Si *delay* es `None`, no se aplicará ningún límite de tiempo; esto puede ser útil si se desconoce el retraso cuando se crea el administrador de contexto.

En cualquier caso, el administrador de contexto se puede reprogramar después de la creación mediante `Timeout.reschedule()`.

Ejemplo:

```
async def main():
    async with asyncio.timeout(10):
        await long_running_task()
```

Si `long_running_task` tarda más de 10 segundos en completarse, el administrador de contexto cancelará la tarea actual y manejará internamente el `asyncio.CancelledError` resultante, transformándolo en un `asyncio.TimeoutError` que se puede capturar y manejar.

Nota

El administrador de contexto `asyncio.timeout()` es lo que transforma el `asyncio.CancelledError` en un `asyncio.TimeoutError`, lo que significa que el **`asyncio.TimeoutError`** solo puede capturarse *outside* del administrador de contexto.

Ejemplo de captura de `TimeoutError`:

```
async def main():
    try:
        async with asyncio.timeout(10):
            await long_running_task()
    except TimeoutError:
        print("The long operation timed out, but we

    print("This statement will run regardless.")
```

El administrador de contexto producido por `asyncio.timeout()` puede reprogramarse para una fecha límite diferente e inspeccionarse.

class asyncio.Timeout(*when*)

Un asynchronous context manager para cancelar corrutinas vencidas.

Prefer using **`asyncio.timeout()`** or **`asyncio.timeout_at()`** rather than instantiating **`Timeout`** directly.

`when` debe ser un tiempo absoluto en el que el contexto debe expirar, según lo medido por el reloj del bucle de eventos:

- Si `when` es `None`, el tiempo de espera nunca se activará.
- Si `when < loop.time()`, el tiempo de espera se activará en la próxima iteración del bucle de eventos

when() → float | None

Retorna la fecha límite actual, o `None` si la fecha límite actual no está establecida.

reschedule(*when: float* | *None*)

Reprogramar el tiempo de espera.

expired() → bool

Retorna si el administrador de contexto ha excedido su fecha límite (caducada).

Ejemplo:

```
async def main():
    try:
        # We do not know the timeout when starting,
        async with asyncio.timeout(None) as cm:
            # We know the timeout now, so we resche
            new_deadline = get_running_loop().time(
            cm.reschedule(new_deadline)

            await long_running_task()
    except TimeoutError:
        pass

    if cm.expired():
        print("Looks like we haven't finished on ti
```

Los administradores de contexto de tiempo de espera se pueden anidar de forma segura.

Added in version 3.11.

asyncio.timeout_at(*when*)

Similar a **`asyncio.timeout()`**, excepto que *when* es el tiempo absoluto para dejar de esperar, o `None`.

Ejemplo:

```
async def main():
    loop = get_running_loop()
    deadline = loop.time() + 20
    try:
        async with asyncio.timeout_at(deadline):
            await long_running_task()
    except TimeoutError:
        print("The long operation timed out, but we

    print("This statement will run regardless.")
```

Added in version 3.11.

async asyncio.wait_for(*aw*, *timeout*)

Espere a que el *aw* esperable se complete con un tiempo agotado.

Si *aw* es una corrutina, se programa automáticamente como una Tarea.

timeout puede ser `None` o punto flotante o un número entero de segundos a esperar. Si *timeout* es `None`, se bloquea hasta que Future se completa.

Si se agota el tiempo de espera, cancela la tarea y lanza **`TimeoutError`**.

Para evitar la **`cancelación`** de la tarea , envuélvala en **`shield()`**.

La función esperará hasta que se cancele el Future, por lo que el tiempo de espera total puede exceder el *timeout*. Si ocurre una excepción durante la cancelación, se propaga.

Si se cancela la espera, el Future *aw* también se cancela.

Ejemplo:

```
async def eternity():
    # Sleep for one hour
    await asyncio.sleep(3600)
    print('yay!')

async def main():
    # Wait for at most 1 second
    try:
        await asyncio.wait_for(eternity(), timeout=
    except TimeoutError:
        print('timeout!')

asyncio.run(main())

# Expected output:
#
#     timeout!
```

Distinto en la versión 3.7: Cuando se cancela *aw* debido a un tiempo de espera, `wait_for` espera a que se cancele *aw*.

Anteriormente, lanzaba **`TimeoutError`** inmediatamente.

Distinto en la versión 3.10: Se quitó el parámetro *loop*.

Distinto en la versión 3.11: Raises **`TimeoutError`** instead of **`asyncio.TimeoutError`**.

Waiting primitives

async asyncio.wait(*aws*, **, *timeout=None*, *return_when=ALL_COMPLETED*)

Ejecuta instancias **`Future`** y **`Task`** en el iterable *aws* simultáneamente y bloquea hasta la condición especificada por *return_when*.

El iterable *aws* no debe estar vacío.

Retorna dos conjuntos de Tareas/Futures: `(done, pending)`.

Uso:

```
done, pending = await asyncio.wait(aws)
```

timeout (un punto flotante o int), si se especifica, se puede utilizar para controlar el número máximo de segundos que hay que esperar antes de retornar.

Tenga en cuenta que esta función no lanza **`TimeoutError`**. Los futuros o las tareas que no se realizan cuando se agota el tiempo de espera simplemente se retornan en el segundo conjunto.

return_when indica cuándo debe retornar esta función. Debe ser una de las siguientes constantes:

Constante	Descripción
asyncio.FIRST_COMPLETED	La función retornará cuando cualquier Future termine o se cancele.
asyncio.FIRST_EXCEPTION	The function will return when any future finishes by

raising an exception. If no future raises an exception then it is equivalent to **ALL_COMPLETED**.

asyncio.ALL_COMPLETED

La función retornará cuando todos los Futures terminen o se cancelen.

A diferencia de **wait_for()**, `wait()` no cancela los Futures cuando se produce un agotamiento de tiempo.

Distinto en la versión 3.10: Se quitó el parámetro *loop*.

Distinto en la versión 3.11: Está prohibido pasar objetos de rutina a `wait()` directamente.

Distinto en la versión 3.12: Se agregó soporte para generadores que generan tareas.

asyncio.as_completed(*aws*, *, *timeout=None*)

Run awaitable objects in the *aws* iterable concurrently. The returned object can be iterated to obtain the results of the awaitables as they finish.

The object returned by `as_completed()` can be iterated as an asynchronous iterator or a plain iterator. When asynchronous iteration is used, the originally-supplied awaitables are yielded if they are tasks or futures. This makes it easy to correlate previously-scheduled tasks with their results. Example:

```
ipv4_connect = create_task(open_connection("127.0.0
ipv6_connect = create_task(open_connection("::1", 8
tasks = [ipv4_connect, ipv6_connect]

async for earliest_connect in as_completed(tasks):
    # earliest_connect is done. The result can be o
    # awaiting it or calling earliest_connect.resul
    reader, writer = await earliest_connect

    if earliest_connect is ipv6_connect:
        print("IPv6 connection established.")
    else:
        print("IPv4 connection established.")
```

During asynchronous iteration, implicitly-created tasks will be yielded for supplied awaitables that aren't tasks or futures.

When used as a plain iterator, each iteration yields a new coroutine that returns the result or raises the exception of the next completed awaitable. This pattern is compatible with Python versions older than 3.13:

```
ipv4_connect = create_task(open_connection("127.0.0
ipv6_connect = create_task(open_connection("::1", 8
tasks = [ipv4_connect, ipv6_connect]

for next_connect in as_completed(tasks):
    # next_connect is not one of the original task
    # awaited to obtain the result value or raise t
    # awaitable that finishes next.
    reader, writer = await next_connect
```

A **`TimeoutError`** is raised if the timeout occurs before all awaitables are done. This is raised by the `async for` loop during asynchronous iteration or by the coroutines yielded during plain iteration.

Distinto en la versión 3.10: Se quitó el parámetro *loop*.

Obsoleto desde la versión 3.10: Se emite una advertencia de obsolescencia si no todos los objetos en espera en el iterable *aws* son objetos de tipo Future y no hay un bucle de eventos en ejecución.

Distinto en la versión 3.12: Se agregó soporte para generadores que generan tareas.

Distinto en la versión 3.13: The result can now be used as either an asynchronous iterator or as a plain iterator (previously it was only a plain iterator).

Running in threads

async asyncio.to_thread(*func*, /, **args*, ***kwargs*)

Ejecutar asincrónicamente la función *func* en un hilo separado.

Cualquier *args y **kwargs suministrados para esta función se pasan directamente a *func*. Además, el current **`contextvars.Context`** se propaga, lo que permite acceder a las variables de contexto del subproceso del bucle de eventos en el subproceso separado.

Retorna una corrutina que se puede esperar para obtener el resultado final de *func*.

Esta función de rutina está diseñada principalmente para ejecutar funciones/métodos vinculados a IO que, de otro modo, bloquearían el bucle de eventos si se ejecutaran en el subproceso principal. Por ejemplo:

```
def blocking_io():
    print(f"start blocking_io at {time.strftime('%X
    # Note that time.sleep() can be replaced with a
    # IO-bound operation, such as file operations.
    time.sleep(1)
    print(f"blocking_io complete at {time.strftime(

async def main():
    print(f"started main at {time.strftime('%X')}")

    await asyncio.gather(
        asyncio.to_thread(blocking_io),
        asyncio.sleep(1))

    print(f"finished main at {time.strftime('%X')}"


asyncio.run(main())

# Expected output:
#
```

```
# started main at 19:50:53
# start blocking_io at 19:50:53
# blocking_io complete at 19:50:54
# finished main at 19:50:54
```

Llamar directamente a `blocking_io()` en cualquier rutina bloquearía el bucle de eventos durante su duración, lo que daría como resultado 1 segundo adicional de tiempo de ejecución. En cambio, al usar `asyncio.to_thread()`, podemos ejecutarlo en un subproceso separado sin bloquear el ciclo de eventos.

Nota

Debido a GIL, `asyncio.to_thread()` generalmente solo se puede usar para hacer que las funciones vinculadas a IO no bloqueen. Sin embargo, para los módulos de extensión que lanzan GIL o implementaciones alternativas de Python que no tienen uno, `asyncio.to_thread()` también se puede usar para funciones vinculadas a la CPU.

Added in version 3.9.

Scheduling from other threads

asyncio.run_coroutine_threadsafe(*coro*, *loop*)

Envía una corrutina al bucle de eventos especificado. Seguro para Hilos.

Retorna **`concurrent.futures.Future`** para esperar el resultado de otro hilo del SO (Sistema Operativo).

Esta función está pensada para llamarse desde un hilo del SO diferente al que se ejecuta el bucle de eventos. Ejemplo:

```
def in_thread(loop: asyncio.AbstractEventLoop) -> N
    # Run some blocking IO
    pathlib.Path("example.txt").write_text("hello w
    # Create a coroutine
    coro = asyncio.sleep(1, result=3)

    # Submit the coroutine to a given loop
    future = asyncio.run_coroutine_threadsafe(coro,

    # Wait for the result with an optional timeout
    assert future.result(timeout=2) == 3

async def amain() -> None:
    # Get the running loop
    loop = asyncio.get_running_loop()

    # Run something in a thread
    await asyncio.to_thread(in_thread, loop)
```

It’s also possible to run the other way around. Example:

```
@contextlib.contextmanager
def loop_in_thread() -> Generator[asyncio.AbstractE
    loop_fut = concurrent.futures.Future[asyncio.Ab
    stop_event = asyncio.Event()

    async def main() -> None:
        loop_fut.set_result(asyncio.get_running_loo
        await stop_event.wait()

    with concurrent.futures.ThreadPoolExecutor(1) a
        complete_fut = tpe.submit(asyncio.run, main
        for fut in concurrent.futures.as_completed(
            if fut is loop_fut:
                loop = loop_fut.result()
                try:
                    yield loop
                finally:
                    loop.call_soon_threadsafe(stop_
            else:
                fut.result()
```

```
# Create a loop in another thread
with loop_in_thread() as loop:
    # Create a coroutine
    coro = asyncio.sleep(1, result=3)

    # Submit the coroutine to a given loop
    future = asyncio.run_coroutine_threadsafe(coro,

    # Wait for the result with an optional timeout
    assert future.result(timeout=2) == 3
```

Si se lanza una excepción en la corrutina, el Future retornado será notificado. También se puede utilizar para cancelar la tarea en el bucle de eventos:

```
try:
    result = future.result(timeout)
except TimeoutError:
    print('The coroutine took too long, cancelling
    future.cancel()
except Exception as exc:
    print(f'The coroutine raised an exception: {exc
else:
    print(f'The coroutine returned: {result!r}')
```

Consulte la sección de la documentación Concurrencia y multi hilos.

A diferencia de otras funciones asyncio, esta función requiere que el argumento *loop* se pase explícitamente.

Added in version 3.5.1.

Introspección

asyncio.current_task(*loop=None*)

Retorna la instancia **`Task`** actualmente en ejecución o `None` si no se está ejecutando ninguna tarea.

Si *loop* es `None` **`get_running_loop()`** se utiliza para obtener el bucle actual.

Added in version 3.7.

asyncio.all_tasks(*loop=None*)

Retorna un conjunto de objetos **`Task`** que se ejecutan por el bucle.

Si *loop* es `None`, **`get_running_loop()`** se utiliza para obtener el bucle actual.

Added in version 3.7.

asyncio.iscoroutine(*obj*)

Retorna `True` si *obj* es un objeto corutina.

Added in version 3.4.

Task object

class asyncio.Task(*coro*, ***, *loop=None*, *name=None*, *context=None*, *eager_start=False*)

Un objeto **`similar a Future`** que ejecuta Python coroutine. No es seguro hilos.

Las tareas se utilizan para ejecutar corrutinas en bucles de eventos. Si una corrutina aguarda en un Future, la Tarea suspende la ejecución de la corrutina y espera la finalización del Future. Cuando el Future *termina*, se reanuda la ejecución de la corrutina envuelta.

Los bucles de eventos usan la programación cooperativa: un bucle de eventos ejecuta una tarea a la vez. Mientras una Tarea espera para la finalización de un Future, el bucle de eventos ejecuta otras tareas, retorno de llamada o realiza operaciones de E/S.

Utilice la función de alto nivel **`asyncio.create_task()`**

para crear Tareas, o las funciones de bajo nivel `loop.create_task()` o `ensure_future()`. Se desaconseja la creación de instancias manuales de Tareas.

Para cancelar una Tarea en ejecución, utilice el método `cancel()`. Llamarlo hará que la tarea lance una excepción `CancelledError` en la corrutina contenida. Si una corrutina está esperando en un objeto Future durante la cancelación, se cancelará el objeto Future.

`cancelled()` se puede utilizar para comprobar si la Tarea fue cancelada. El método retorna `True` si la corrutina contenida no suprimió la excepción `CancelledError` y se canceló realmente.

`asyncio.Task` hereda de `Future` todas sus API excepto `Future.set_result()` y `Future.set_exception()`.

Un argumento *context* opcional de solo palabra clave permite especificar un `contextvars.Context` personalizado para que se ejecute *coro*. Si no se proporciona ningún *context*, la tarea copia el contexto actual y luego ejecuta su rutina en el contexto copiado.

Un argumento *eager_start* opcional de solo palabra clave permite iniciar con entusiasmo la ejecución de `asyncio.Task` en el momento de la creación de la tarea. Si se establece en `True` y el bucle de eventos se está ejecutando, la tarea comenzará a ejecutar la corrutina inmediatamente, hasta la primera vez que la corrutina se bloquee. Si la rutina regresa o se activa sin bloquearse, la tarea finalizará con entusiasmo y saltará la programación al bucle de eventos.

Distinto en la versión 3.7: Agregado soporte para el módulo `contextvars`.

Distinto en la versión 3.8: Se ha añadido el parámetro *name*.

Obsoleto desde la versión 3.10: Se emite una advertencia de obsolescencia si no se especifica *loop* y no hay un bucle de eventos en ejecución.

Distinto en la versión 3.11: Se ha añadido el parámetro *context*.

Distinto en la versión 3.12: Se ha añadido el parámetro *eager_start*.

done()

Retorna `True` si la Tarea está *finalizada*.

Una tarea está *finalizada* cuando la corrutina contenida retornó un valor, lanzó una excepción, o se canceló la Tarea.

result()

Retorna el resultado de la Tarea.

Si la tarea está *terminada*, se retorna el resultado de la corrutina contenida (o si la corrutina lanzó una excepción, esa excepción se vuelve a relanzar.)

Si la Tarea ha sido *cancelada*, este método lanza una excepción **`CancelledError`**.

If the Task's result isn't yet available, this method raises an **`InvalidStateError`** exception.

exception()

Retorna la excepción de la Tarea.

Si la corrutina contenida lanzó una excepción, esa excepción es retornada. Si la corrutina contenida retorna normalmente, este método retorna `None`.

Si la Tarea ha sido *cancelada*, este método lanza una excepción **`CancelledError`**.

Si la Tarea aún no está *terminada*, este método lanza una excepción **`InvalidStateError`**.

add_done_callback(*callback*, *, *context=None*)

Agrega una retro llamada que se ejecutará cuando la Tarea esté *terminada*.

Este método solo se debe usar en código basado en retrollamada de bajo nivel.

Consulte la documentación de `Future.add_done_callback()` para obtener más detalles.

remove_done_callback(*callback*)

Remueve la *retrollamada* de la lista de retrollamadas.

Este método solo se debe usar en código basado en retrollamada de bajo nivel.

Consulte la documentación de `Future.remove_done_callback()` para obtener más detalles.

get_stack(*, *limit=None*)

Retorna la lista de marcos de pila para esta tarea.

Si la corrutina contenida no se termina, esto retorna la pila donde se suspende. Si la corrutina se ha completado correctamente o se ha cancelado, retorna una lista vacía. Si la corrutina terminó por una excepción, esto retorna la lista de marcos de seguimiento.

Los marcos siempre se ordenan de más antiguo a más nuevo.

Solo se retorna un marco de pila para una corrutina suspendida.

El argumento opcional *limit* establece el número máximo de marcos que se retornarán; de forma predeterminada se retornan todos los marcos disponibles. El orden de la lista retornada varía en función de si se retorna una pila o un *traceback*: se retornan los marcos más recientes de una pila, pero se retornan los marcos más antiguos de un *traceback*. (Esto coincide con el comportamiento del módulo traceback.)ss

print_stack(*, *limit=None*, *file=None*)

Imprime la pila o el seguimiento de esta tarea.

Esto produce una salida similar a la del módulo traceback para los marcos recuperados por **`get_stack()`**.

El argumento *limit* se pasa directamente a **`get_stack()`**.

El argumento *file* es un flujo de E/S en el que se escribe la salida; por defecto, la salida se escribe en **`sys.stdout`**.

get_coro()

Retorna el objeto corrutina contenido por **`Task`**.

Nota

Esto devolverá `None` para las tareas que ya se han completado con entusiasmo. Consulte el Eager Task Factory.

Added in version 3.8.

Distinto en la versión 3.12: La ejecución ansiosa de la tarea recientemente agregada significa que el resultado puede ser `None`.

get_context()

Devuelve el objeto **`contextvars.Context`** asociado con la tarea.

Added in version 3.12.

get_name()

Retorna el nombre de la Tarea.

Si no se ha asignado explícitamente ningún nombre a la Tarea, la implementación de Tarea asyncio predeterminada genera un nombre predeterminado durante la creación de instancias.

Added in version 3.8.

set_name(*value*)

Establece el nombre de la Tarea.

El argumento *value* puede ser cualquier objeto, que luego se convierte en una cadena.

En la implementación de Task predeterminada, el nombre será visible en la salida **`repr()`** de un objeto de tarea.

Added in version 3.8.

cancel(*msg=None*)

Solicita que se cancele la Tarea.

If the Task is already *done* or *cancelled*, return `False`, otherwise, return `True`.

The method arranges for a **`CancelledError`** exception to be thrown into the wrapped coroutine on the next cycle of the event loop.

Luego, la corrutina tiene la oportunidad de limpiar o incluso denegar la solicitud suprimiendo la excepción con un bloque **`try`**… … `except CancelledError`… **`finally`**. Por lo tanto, a diferencia de **`Future.cancel()`**, **`Task.cancel()`** no garantiza que la tarea se cancelará, aunque suprimir la cancelación por completo no es común y se desaconseja activamente. Sin embargo, si la rutina decide suprimir la cancelación, debe llamar a **`Task.uncancel()`** además de detectar la excepción.

Distinto en la versión 3.9: Se agregó el parámetro *msg*.

Distinto en la versión 3.11: El parámetro `msg` se propaga desde la tarea cancelada a su espera.

En el ejemplo siguiente se muestra cómo las corrutinas pueden interceptar la solicitud de cancelación:

```
async def cancel_me():
    print('cancel_me(): before sleep')

    try:
        # Wait for 1 hour
        await asyncio.sleep(3600)
    except asyncio.CancelledError:
        print('cancel_me(): cancel sleep')
        raise
    finally:
        print('cancel_me(): after sleep')

async def main():
    # Create a "cancel_me" Task
    task = asyncio.create_task(cancel_me())

    # Wait for 1 second
    await asyncio.sleep(1)

    task.cancel()
    try:
        await task
    except asyncio.CancelledError:
        print("main(): cancel_me is cancelled r

asyncio.run(main())
```

```
# Expected output:
#
#     cancel_me(): before sleep
#     cancel_me(): cancel sleep
#     cancel_me(): after sleep
#     main(): cancel_me is cancelled now
```

cancelled()

Retorna `True` si la Tarea se *cancela*.

La tarea se *cancela* cuando se solicitó la cancelación con **`cancel()`** y la corrutina contenida propagó la excepción **`CancelledError`** que se le ha lanzado.

uncancel()

Disminuye el recuento de solicitudes de cancelación a esta tarea.

Retorna el número restante de solicitudes de cancelación.

Tenga en cuenta que una vez que se completa la ejecución de una tarea cancelada, las llamadas posteriores a **`uncancel()`** no son efectivas.

Added in version 3.11.

Este método lo usan los componentes internos de asyncio y no se espera que lo use el código del usuario final. En particular, si una Tarea se cancela con éxito, esto permite que elementos de concurrencia estructurada como Task groups y **`asyncio.timeout()`** continúen ejecutándose, aislando la cancelación al bloque estructurado respectivo. Por ejemplo:

```
async def make_request_with_timeout():
    try:
        async with asyncio.timeout(1):
            # Structured block affected by the
            await make_request()
            await make_another_request()
    except TimeoutError:
        log("There was a timeout")
    # Outer code not affected by the timeout:
    await unrelated_code()
```

Si bien el bloque con `make_request()` y `make_another_request()` podría cancelarse debido al tiempo de espera, `unrelated_code()` debería continuar ejecutándose incluso en caso de que se agote el tiempo de espera. Esto se implementa con `uncancel()`. Los administradores de contexto `TaskGroup` usan `uncancel()` de manera similar.

If end-user code is, for some reason, suppressing cancellation by catching `CancelledError`, it needs to call this method to remove the cancellation state.

When this method decrements the cancellation count to zero, the method checks if a previous `cancel()` call had arranged for `CancelledError` to be thrown into the task. If it hasn't been thrown yet, that arrangement will be rescinded (by resetting the internal `_must_cancel` flag).

Distinto en la versión 3.13: Changed to rescind pending cancellation requests upon reaching zero.

cancelling()

Retorna el número de solicitudes de cancelación pendientes a esta Tarea, es decir, el número de llamadas a `cancel()` menos el número de llamadas a `uncancel()`.

Tenga en cuenta que si este número es mayor que cero pero la tarea aún se está ejecutando, `cancelled()` aún retornará `False`. Esto se debe a que este número se puede reducir llamando a `uncancel()`, lo que puede provocar que la tarea no se cancele después de todo si las solicitudes de cancelación se reducen a cero.

Este método lo utilizan las partes internas de asyncio y no se espera que lo utilice el código del usuario final. Consulte `uncancel()` para obtener más detalles.

Added in version 3.11.

Streams

Código fuente: Lib/asyncio/streams.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/streams.py]

Los *streams* son async/await primitivos de alto nivel para trabajar con conexiones de red. Los *streams* permiten enviar y recibir datos sin utilizar *callbacks* o protocolos y transportes de bajo nivel.

Aquí hay un ejemplo de un cliente eco TCP escrito utilizando *streams* asyncio:

```
import asyncio

async def tcp_echo_client(message):
    reader, writer = await asyncio.open_connection(
        '127.0.0.1', 8888)

    print(f'Send: {message!r}')
    writer.write(message.encode())
    await writer.drain()

    data = await reader.read(100)
    print(f'Received: {data.decode()!r}')

    print('Close the connection')
    writer.close()
    await writer.wait_closed()

asyncio.run(tcp_echo_client('Hello World!'))
```

Consulte también la sección de Examples a continuación.

Funciones *stream*

Las siguientes funciones asyncio de nivel superior se pueden utilizar para crear y trabajar con *streams*:

async asyncio.open_connection(*host=None*, *port=None*, ***, *limit=None*, *ssl=None*, *family=0*, *proto=0*, *flags=0*, *sock=None*, *local_addr=None*, *server_hostname=None*, *ssl_handshake_timeout=None*, *ssl_shutdown_timeout=None*, *happy_eyeballs_delay=None*, *interleave=None*)

Establece una conexión de red y retorna un par de objetos `(reader, writer)`.

Los objetos *reader* y *writer* retornados son instancias de las clases **`StreamReader`** y **`StreamWriter`**.

limit determina el límite de tamaño del búfer utilizado por la instancia de **`StreamReader`** retornada. De forma predeterminada, *limit* está establecido en 64 KiB.

El resto de los argumentos se pasan directamente a **`loop.create_connection()`**.

Nota

El argumento *sock* transfiere la propiedad del socket al **`StreamWriter`** creado. Para cerrar el socket, llame a su método **`close()`**.

Distinto en la versión 3.7: Se agregó el parámetro *ssl_handshake_timeout*.

Distinto en la versión 3.8: Added the *happy_eyeballs_delay* and *interleave* parameters.

Distinto en la versión 3.10: Se eliminó el parámetro *loop*.

Distinto en la versión 3.11: Added the *ssl_shutdown_timeout* parameter.

async asyncio.start_server(*client_connected_cb*, *host=None*, *port=None*, ***, *limit=None*, *family=socket.AF_UNSPEC*, *flags=socket.AI_PASSIVE, sock=None, backlog=100, ssl=None, reuse_address=None, reuse_port=None, keep_alive=None, ssl_handshake_timeout=None, ssl_shutdown_timeout=None, start_serving=True*)

Inicia un servidor socket.

La retrollamada *client_connected_cb* se llama siempre que se establece una nueva conexión de cliente. Recibe un par `(reader, writer)` como dos argumentos, instancias de las clases **`StreamReader`** y **`StreamWriter`**.

client_connected_cb puede ser una función simple invocable o de corrutina; si es una función de corrutina, se programará automáticamente como un **`Task`**.

limit determina el límite de tamaño del búfer utilizado por la instancia de **`StreamReader`** retornada. De forma predeterminada, *limit* está establecido en 64 KiB.

El resto de los argumentos se pasan directamente a **`loop.create_server()`**.

Nota

El argumento *sock* transfiere la propiedad del socket al servidor creado. Para cerrar el socket, llame al método **`close()`** del servidor.

Distinto en la versión 3.7: Se agregaron los parámetros *ssl_handshake_timeout* y *start_serving*.

Distinto en la versión 3.10: Se eliminó el parámetro *loop*.

Distinto en la versión 3.11: Added the *ssl_shutdown_timeout* parameter.

Distinto en la versión 3.13: Added the *keep_alive* parameter.

Sockets Unix

async asyncio.open_unix_connection(*path=None*, ***, *limit=None*, *ssl=None*, *sock=None*, *server_hostname=None*, *ssl_handshake_timeout=None*, *ssl_shutdown_timeout=None*)

Establece una conexión de socket Unix y retorna un par de `(reader, writer)`.

Similar a `open_connection()` pero opera en sockets Unix.

Consulte también la documentación de `loop.create_unix_connection()`.

Nota

El argumento *sock* transfiere la propiedad del socket al `StreamWriter` creado. Para cerrar el socket, llame a su método `close()`.

Availability: Unix.

Distinto en la versión 3.7: Se agregó el parámetro *ssl_handshake_timeout*. El parámetro *path* ahora puede ser un path-like object

Distinto en la versión 3.10: Se eliminó el parámetro *loop*.

Distinto en la versión 3.11: Added the *ssl_shutdown_timeout* parameter.

async asyncio.start_unix_server(*client_connected_cb*, *path=None*, ***, *limit=None*, *sock=None*, *backlog=100*, *ssl=None*, *ssl_handshake_timeout=None*, *ssl_shutdown_timeout=None*, *start_serving=True*, *cleanup_socket=True*)

Inicia un servidor socket Unix.

Similar a `start_server()` pero funciona con sockets Unix.

If *cleanup_socket* is true then the Unix socket will automatically be removed from the filesystem when the server is closed, unless the socket has been replaced after the server has been created.

Consulte también la documentación de `loop.create_unix_server()`.

Nota

El argumento *sock* transfiere la propiedad del socket al servidor creado. Para cerrar el socket, llame al método `close()` del servidor.

Availability: Unix.

Distinto en la versión 3.7: Se agregaron los parámetros *ssl_handshake_timeout* y *start_serving*. El parámetro *path* ahora puede ser un path-like object.

Distinto en la versión 3.10: Se eliminó el parámetro *loop*.

Distinto en la versión 3.11: Added the *ssl_shutdown_timeout* parameter.

Distinto en la versión 3.13: Added the *cleanup_socket* parameter.

StreamReader

class asyncio.StreamReader

Represents a reader object that provides APIs to read data from the IO stream. As an asynchronous iterable, the object supports the `async for` statement.

No se recomienda crear instancias de objetos *StreamReader* directamente; utilice `open_connection()` y `start_server()` en su lugar.

feed_eof()

Acknowledge the EOF.

async read(*n=-1*)

Read up to *n* bytes from the stream.

If *n* is not provided or set to `-1`, read until EOF, then return all read **`bytes`**. If EOF was received and the internal buffer is empty, return an empty `bytes` object.

If *n* is `0`, return an empty `bytes` object immediately.

If *n* is positive, return at most *n* available `bytes` as soon as at least 1 byte is available in the internal buffer. If EOF is received before any byte is read, return an empty `bytes` object.

async readline()

Lea una línea, donde «línea» es una secuencia de bytes que termina en `\n`.

Si se recibe EOF (final del archivo) y no se encontró `\n`, el método retorna datos leídos parcialmente.

Si se recibe EOF (final de archivo) y el búfer interno está vacío, retorna un objeto `bytes` vacío.

async readexactly(*n*)

Lee exactamente *n* bytes.

Genera un **`IncompleteReadError`** si se alcanza EOF (final del archivo) antes de que se pueda leer *n*. Utilice el atributo **`IncompleteReadError.partial`** para obtener los datos leídos parcialmente.

async readuntil(*separator=b'\n'*)

Lee datos de la secuencia hasta que se encuentre el separador (*separator*).

En caso de éxito, los datos y el separador se eliminarán del búfer interno (consumido). Los datos retornados incluirán el separador al final.

Si la cantidad de datos leídos excede el límite de flujo configurado, se genera una excepción `LimitOverrunError` y los datos se dejan en el búfer interno y se pueden leer nuevamente.

Si se alcanza EOF (final del archivo) antes de que se encuentre el separador completo, se genera una excepción `IncompleteReadError` y se restablece el búfer interno. El atributo `IncompleteReadError.partial` puede contener una parte del separador.

The *separator* may also be a tuple of separators. In this case the return value will be the shortest possible that has any separator as the suffix. For the purposes of `LimitOverrunError`, the shortest possible separator is considered to be the one that matched.

Added in version 3.5.2.

Distinto en la versión 3.13: The *separator* parameter may now be a `tuple` of separators.

at_eof()

Retorna `True` si el buffer está vacío y `feed_eof()` fue llamado.

StreamWriter

class asyncio.StreamWriter

Representa un objeto de escritura que proporciona APIs para escribir datos en el flujo de E/S.

No se recomienda crear instancias de objetos *StreamWriter* directamente; use `open_connection()` y `start_server()` en su lugar.

write(*data*)

El método intenta escribir los datos (*data*) en el socket subyacente inmediatamente. Si eso falla, los datos se ponen en cola en un búfer de escritura interno hasta que se puedan enviar.

The *data* buffer should be a bytes, bytearray, or C-contiguous one-dimensional memoryview object.

El método debe usarse junto con el método `drain()`:

```
stream.write(data)
await stream.drain()
```

writelines(*data*)

El método escribe una lista (o cualquier iterable) de bytes en el socket subyacente inmediatamente. Si eso falla, los datos se ponen en cola en un búfer de escritura interno hasta que se puedan enviar.

El método debe usarse junto con el método `drain()`:

```
stream.writelines(lines)
await stream.drain()
```

close()

El método cierra la secuencia y el socket subyacente.

The method should be used, though not mandatory, along with the `wait_closed()` method:

```
stream.close()
await stream.wait_closed()
```

can_write_eof()

Retorna `True` si el transporte subyacente admite el método **`write_eof()`**, `False` en caso contrario.

write_eof()

Cierra la escritura de la secuencia después de que se vacíen los datos de escritura almacenados en búfer.

transport

Retorna el transporte asyncio subyacente.

get_extra_info(*name*, *default=None*)

Accede a información de transporte opcional; consulte **`BaseTransport.get_extra_info()`** para obtener más detalles.

async drain()

Espera hasta que sea apropiado reanudar la escritura en la transmisión. Ejemplo:

```
writer.write(data)
await writer.drain()
```

Este es un método de control de flujo que interactúa con el búfer de escritura de E/S subyacente. Cuando el tamaño del búfer alcanza la marca de agua alta, *drain()* bloquea hasta que el tamaño del búfer se agota hasta la marca de agua baja y se pueda reanudar la escritura. Cuando no hay nada que esperar, **`drain()`** regresa inmediatamente.

async start_tls(*sslcontext*, ***, *server_hostname=None*, *ssl_handshake_timeout=None*, *ssl_shutdown_timeout=None*)

Actualiza una conexión existente basada en flujo a TLS.

Parámetros:

- *sslcontext*: una instancia configurada de **`SSLContext`**.
- *server_hostname*: establece o sustituye el nombre de host con el que se comparará el certificado del servidor de destino.
- *ssl_handshake_timeout* es el tiempo en segundos que se espera a que se complete el protocolo TLS antes de abortar la conexión. `60.0` segundos si `None` (por defecto).

- *ssl_shutdown_timeout* is the time in seconds to wait for the SSL shutdown to complete before aborting the connection. `30.0` seconds if `None` (default).

Added in version 3.11.

Distinto en la versión 3.12: Added the *ssl_shutdown_timeout* parameter.

is_closing()

Retorna `True` si la secuencia está cerrada o en proceso de cerrarse.

Added in version 3.7.

async wait_closed()

Espera hasta que se cierre la secuencia.

Should be called after **`close()`** to wait until the underlying connection is closed, ensuring that all data has been flushed before e.g. exiting the program.

Added in version 3.7.

Ejemplos

Cliente eco TCP usando *streams*

Cliente eco TCP usando la función **`asyncio.open_connection()`**:

```
import asyncio

async def tcp_echo_client(message):
    reader, writer = await asyncio.open_connection(
        '127.0.0.1', 8888)

    print(f'Send: {message!r}')
    writer.write(message.encode())
    await writer.drain()

    data = await reader.read(100)
    print(f'Received: {data.decode()!r}')

    print('Close the connection')
    writer.close()
    await writer.wait_closed()

asyncio.run(tcp_echo_client('Hello World!'))
```

Ver también

El ejemplo del protocolo de cliente eco TCP utiliza el método `loop.create_connection()` de bajo nivel.

Servidor eco TCP usando *streams*

Servidor eco TCP usando la función `asyncio.start_server()`:

```
import asyncio

async def handle_echo(reader, writer):
    data = await reader.read(100)
    message = data.decode()
    addr = writer.get_extra_info('peername')

    print(f"Received {message!r} from {addr!r}")

    print(f"Send: {message!r}")
    writer.write(data)
    await writer.drain()

    print("Close the connection")
    writer.close()
    await writer.wait_closed()

async def main():
    server = await asyncio.start_server(
        handle_echo, '127.0.0.1', 8888)

    addrs = ', '.join(str(sock.getsockname()) for sock i
    print(f'Serving on {addrs}')

    async with server:
        await server.serve_forever()

asyncio.run(main())
```

Ver también

El ejemplo del protocolo de servidor eco TCP utiliza el método `loop.create_server()`.

Obtener encabezados HTTP

Ejemplo simple de consulta de encabezados HTTP de la URL pasada en la línea de comando:

```
import asyncio
import urllib.parse
import sys

async def print_http_headers(url):
    url = urllib.parse.urlsplit(url)
    if url.scheme == 'https':
        reader, writer = await asyncio.open_connection(
            url.hostname, 443, ssl=True)
    else:
        reader, writer = await asyncio.open_connection(
            url.hostname, 80)

    query = (
        f"HEAD {url.path or '/'} HTTP/1.0\r\n"
        f"Host: {url.hostname}\r\n"
        f"\r\n"
    )

    writer.write(query.encode('latin-1'))
    while True:
        line = await reader.readline()
        if not line:
            break

        line = line.decode('latin1').rstrip()
        if line:
            print(f'HTTP header> {line}')

    # Ignore the body, close the socket
    writer.close()
    await writer.wait_closed()

url = sys.argv[1]
asyncio.run(print_http_headers(url))
```

Uso:

```
python example.py http://example.com/path/page.html
```

o con HTTPS:

```
python example.py https://example.com/path/page.html
```

Registrar un socket abierto para esperar datos usando *streams*

Corutina esperando hasta que un socket reciba datos usando la función `open_connection()` function:

```
import asyncio
import socket

async def wait_for_data():
    # Get a reference to the current event loop because
    # we want to access low-level APIs.
    loop = asyncio.get_running_loop()

    # Create a pair of connected sockets.
    rsock, wsock = socket.socketpair()

    # Register the open socket to wait for data.
    reader, writer = await asyncio.open_connection(sock=

    # Simulate the reception of data from the network
    loop.call_soon(wsock.send, 'abc'.encode())

    # Wait for data
    data = await reader.read(100)

    # Got data, we are done: close the socket
    print("Received:", data.decode())
    writer.close()
    await writer.wait_closed()

    # Close the second socket
    wsock.close()

asyncio.run(wait_for_data())
```

Ver también

El ejemplo de registro de un socket abierto para esperar datos usando un protocolo utiliza un protocolo de bajo nivel y el método **`loop.create_connection()`**.

El ejemplo de observar un descriptor de archivo para leer eventos utiliza el método **`loop.add_reader()`** de bajo nivel para ver un descriptor de archivo.

Primitivas de sincronización

Código fuente: Lib/asyncio/locks.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/locks.py]

---

las primitivas de sincronización de asyncio están diseñadas para ser similares a las del módulo `threading`, con dos importantes advertencias:

- las primitivas de asyncio no son seguras en hilos, por lo tanto, no deben ser usadas para la sincronización de hilos del sistema operativo (usa `threading` para esto);
- los métodos de estas primitivas de sincronización no aceptan el argumento *timeout*. Usa la función `asyncio.wait_for()` para realizar operaciones que involucren tiempos de espera.

asyncio tiene las siguientes primitivas de sincronización básicas:

- `Lock`
- `Event`
- `Condition`
- `Semaphore`
- `BoundedSemaphore`
- `Barrier`

Lock

class asyncio.Lock

Implementa un cierre de exclusión mutua para tareas asyncio. No es seguro en hilos.

Un cierre asyncio puede usarse para garantizar el acceso en exclusiva a un recurso compartido.

La forma preferida de usar un Lock es mediante una declaración **`async with`**:

```
lock = asyncio.Lock()

# ... later
async with lock:
    # access shared state
```

lo que es equivalente a:

```
lock = asyncio.Lock()

# ... later
await lock.acquire()
try:
    # access shared state
finally:
    lock.release()
```

Distinto en la versión 3.10: Eliminado el parámetro *loop*.

async acquire()

Adquiere el cierre.

Este método espera hasta que el cierre está *abierto*, lo establece como *cerrado* y retorna `True`.

Cuando más de una corrutina está bloqueada en **`acquire()`**, esperando a que el cierre se abra, solo una de las corrutinas proseguirá finalmente.

Adquirir un cierre es *justo*: la corrutina que prosigue será la primera corrutina que comenzó a esperarlo.

release()

Libera el cierre.

Cuando el cierre está *cerrado*, lo restablece al estado *abierto* y retorna.

Si el cierre está *abierto*, se lanza una excepción **`RuntimeError`**.

locked()

Retorna `True` si el cierre está *cerrado*.

Event

class asyncio.Event

Un objeto de eventos. No es seguro en hilos.

Un evento asyncio puede usarse para notificar a múltiples tareas asyncio que ha ocurrido algún evento.

Un objeto Event administra una bandera interna que se puede establecer en *true* con el método **`set()`** y se restablece en *false* con el método **`clear()`**. El método **`wait()`** se bloquea hasta que la bandera se establece en *true*. El flag se establece en *false* inicialmente.

Distinto en la versión 3.10: Eliminado el parámetro *loop*.

Ejemplo:

```
async def waiter(event):
    print('waiting for it ...')
    await event.wait()
    print('... got it!')

async def main():
    # Create an Event object.
    event = asyncio.Event()

    # Spawn a Task to wait until 'event' is set.
    waiter_task = asyncio.create_task(waiter(event)

    # Sleep for 1 second and set the event.
    await asyncio.sleep(1)
    event.set()
    # Wait until the waiter task is finished.
    await waiter_task

asyncio.run(main())
```

async wait()

Espera hasta que se establezca el evento.

Si el evento está configurado, retorna `True` inmediatamente. De lo contrario, bloquea hasta que otra tarea llame a **`set()`**.

set()

Establece el evento.

Todas las tareas esperando a que el evento se establezca serán activadas inmediatamente.

clear()

Borra (restablece) el evento.

Subsequent tasks awaiting on **`wait()`** will now block until the **`set()`** method is called again.

is_set()

Retorna `True` si el evento está establecido.

Condition

class asyncio.Condition(*lock=None*)

Un objeto Condition. No seguro en hilos.

Una tarea puede usar una condición primitiva de asyncio para esperar a que suceda algún evento y luego obtener acceso exclusivo a un recurso compartido.

En esencia, un objeto Condition combina la funcionalidad de un objeto **`Event`** y un objeto **`Lock`**. Es posible tener múltiples objetos Condition que compartan un mismo Lock, lo que permite coordinar el acceso exclusivo a un recurso compartido entre diferentes tareas interesadas en estados particulares de ese recurso compartido.

El argumento opcional *lock* debe ser un objeto **`Lock`** o `None`. En este último caso, se crea automáticamente un nuevo objeto Lock.

Distinto en la versión 3.10: Eliminado el parámetro *loop*.

La forma preferida de usar una condición es mediante una declaración **`async with`**:

```
cond = asyncio.Condition()

# ... later
async with cond:
    await cond.wait()
```

lo que es equivalente a:

```
cond = asyncio.Condition()

# ... later
await cond.acquire()
try:
    await cond.wait()
finally:
    cond.release()
```

async acquire()

Adquiere el cierre (lock) subyacente.

Este método espera hasta que el cierre subyacente esté *abierto*, lo establece en *cerrado* y retorna `True`.

notify(*n=1*)

Wake up *n* tasks (1 by default) waiting on this condition. If fewer than *n* tasks are waiting they are all awakened.

El cierre debe adquirirse antes de llamar a este método y liberarse poco después. Si se llama con un cierre *abierto*, se lanza una excepción **`RuntimeError`**.

locked()

Retorna `True` si el cierre subyacente está adquirido.

notify_all()

Despierta todas las tareas que esperan a esta condición.

Este método actúa como **`notify()`**, pero despierta todas las tareas en espera.

El cierre debe adquirirse antes de llamar a este método y liberarse poco después. Si se llama con un cierre *abierto*, se lanza una excepción **`RuntimeError`**.

release()

Libera el cierre subyacente.

Cuando se invoca en un cierre abierto, se lanza una excepción **`RuntimeError`**.

async wait()

Espera hasta que se le notifique.

Si la tarea que llama no ha adquirido el cierre cuando se llama a este método, se lanza una excepción **`RuntimeError`**.

Este método libera el cierre subyacente y luego se bloquea hasta que lo despierte una llamada **`notify()`** o **`notify_all()`**. Una vez despertado, la condición vuelve a adquirir su cierre y este método retorna `True`.

Note that a task *may* return from this call spuriously, which is why the caller should always re-check the state and be prepared to **`wait()`** again. For this reason, you may prefer to use `wait_for()` instead.

async wait_for(*predicate*)

Espera hasta que un predicado se vuelva *verdadero*.

The predicate must be a callable which result will be interpreted as a boolean value. The method will repeatedly `wait()` until the predicate evaluates to *true*. The final value is the return value.

Semaphore

class asyncio.Semaphore(*value=1*)

Un objeto Semaphore. No es seguro en hilos.

Un semáforo gestiona un contador interno que se reduce en cada llamada al método `acquire()` y se incrementa en cada llamada al método `release()`. El contador nunca puede bajar de cero, cuando **`acquire()`** encuentra que es cero, se bloquea, esperando hasta que alguna tarea llame a **`release()`**.

El argumento opcional *value* proporciona el valor inicial para el contador interno (`1` por defecto). Si el valor dado es menor que `0` se lanza una excepción `ValueError`.

Distinto en la versión 3.10: Eliminado el parámetro *loop*.

La forma preferida de utilizar un semáforo es mediante una declaración `async with`:

```
sem = asyncio.Semaphore(10)

# ... later
async with sem:
    # work with shared resource
```

lo que es equivalente a:

```
sem = asyncio.Semaphore(10)

# ... later
await sem.acquire()
try:
    # work with shared resource
finally:
    sem.release()
```

async acquire()

Adquiere un semáforo.

Si el contador interno es mayor que cero, lo reduce en uno y retorna `True` inmediatamente. Si es cero, espera hasta que se llame al método **`release()`** y retorna `True`.

locked()

Retorna `True` si el semáforo no se puede adquirir de inmediato.

release()

Libera un semáforo, incrementando el contador interno en uno. Puede despertar una tarea que esté a la espera para adquirir el semáforo.

A diferencia de **`BoundedSemaphore`**, **`Semaphore`** permite hacer más llamadas `release()` que llamadas `acquire()`.

BoundedSemaphore

class asyncio.BoundedSemaphore(*value=1*)

Un objeto semáforo delimitado. No es seguro en hilos.

BoundedSemaphore es una versión de la clase **`Semaphore`** que lanza una excepción `ValueError` en `release()` si aumenta el contador interno por encima del *valor* inicial.

Distinto en la versión 3.10: Eliminado el parámetro *loop*.

Barrera

class asyncio.Barrier(*parties*)

Un objeto barrera. No es seguro en hilos.

Una barrera es una primitiva de sincronización simple que permite bloquear hasta que un número *parties* de tareas estén esperando en ella. Las tareas pueden esperar en el método `wait()` y se bloquearán hasta que el número especificado de tareas termine esperando en `wait()`. En ese momento, todas las tareas en espera se desbloquearán simultáneamente.

`async with` puede utilizarse como alternativa a esperar en `wait()`.

La barrera puede reutilizarse tantas veces como se desee.

Ejemplo:

```
async def example_barrier():
   # barrier with 3 parties
   b = asyncio.Barrier(3)

   # create 2 new waiting tasks
   asyncio.create_task(b.wait())
   asyncio.create_task(b.wait())

   await asyncio.sleep(0)
   print(b)

   # The third .wait() call passes the barrier
   await b.wait()
   print(b)
   print("barrier passed")
```

```
    await asyncio.sleep(0)
    print(b)

asyncio.run(example_barrier())
```

El resultado de este ejemplo es:

```
<asyncio.locks.Barrier object at 0x... [filling, wa
<asyncio.locks.Barrier object at 0x... [draining, w
barrier passed
<asyncio.locks.Barrier object at 0x... [filling, wa
```

Added in version 3.11.

async wait()

Pasa la barrera. Cuando todas las tareas que forman parte de la barrera han llamado a esta función, todas se desbloquean simultáneamente.

Cuando se cancela una tarea en espera o bloqueada en la barrera, esta tarea sale de la barrera que permanece en el mismo estado. Si el estado de la barrera es «filling», el número de tareas en espera disminuye en 1.

El valor de retorno es un entero en el rango de 0 a `parties-1`, diferente para cada tarea. Esto se puede utilizar para seleccionar una tarea para hacer algún mantenimiento especial, por ejemplo:

```
...
async with barrier as position:
    if position == 0:
        # Only one task prints this
        print('End of *draining phase*')
```

Este método puede lanzar una excepción **`BrokenBarrierError`** si la barrera se rompe o se restablece mientras una tarea está en espera. Podría lanzar una excepción **`CancelledError`** si se cancela una tarea.

async reset()

Devuelve la barrera al estado vacío por defecto. Cualquier tarea que espere en ella recibirá la excepción **`BrokenBarrierError`**.

Si se rompe una barrera, puede ser mejor dejarla y crear una nueva.

async abort()

Put the barrier into a broken state. This causes any active or future calls to **`wait()`** to fail with the **`BrokenBarrierError`**. Use this for example if one of the tasks needs to abort, to avoid infinite waiting tasks.

parties

El número de tareas necesarias para pasar la barrera.

n_waiting

El número de tareas que esperan actualmente en la barrera mientras se llena.

broken

Un booleano que es `True` si la barrera está en estado roto.

exception asyncio.BrokenBarrierError

Esta excepción, una subclase de **`RuntimeError`**, se lanza cuando el objeto **`Barrier`** se reinicia o se rompe.

Distinto en la versión 3.9: Adquirir un bloqueo usando `await lock` o `yield from lock` o una declaración **`with`** (`with await lock`, `with (yield from lock)`) se eliminó . En su lugar, use `async with lock`.

Sub-procesos

Código fuente: Lib/asyncio/subprocess.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/subprocess.py], Lib/asyncio/base_subprocess.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/base_subprocess.py]

Esta sección describe APIs de alto nivel de *async/await* de asyncio para crear y gestionar sub-procesos.

Aquí podemos encontrar un ejemplo de cómo asyncio puede ejecutar un comando de *shell* y obtener su resultado:

```
import asyncio

async def run(cmd):
    proc = await asyncio.create_subprocess_shell(
        cmd,
        stdout=asyncio.subprocess.PIPE,
        stderr=asyncio.subprocess.PIPE)

    stdout, stderr = await proc.communicate()

    print(f'[{cmd!r} exited with {proc.returncode}]')
    if stdout:
        print(f'[stdout]\n{stdout.decode()}')
    if stderr:
        print(f'[stderr]\n{stderr.decode()}')

asyncio.run(run('ls /zzz'))
```

mostrará en pantalla:

```
['ls /zzz' exited with 1]
[stderr]
```

```
ls: /zzz: No such file or directory
```

Ya que todos las funciones de sub-procesos de asyncio son asíncronas y asyncio proporciona herramientas para trabajar con tales funciones, es fácil ejecutar y monitorear múltiples subprocesos en paralelo. De hecho es trivial modificar los ejemplos de arriba para ejecutar varios comandos simultáneamente:

```
async def main():
    await asyncio.gather(
        run('ls /zzz'),
        run('sleep 1; echo "hello"'))

asyncio.run(main())
```

Véase también la subsección Examples.

Creando sub-procesos

async asyncio.create_subprocess_exec(*program*, **args*, *stdin=None*, *stdout=None*, *stderr=None*, *limit=None*, ***kwds*)

Crea un sub-proceso.

The *limit* argument sets the buffer limit for `StreamReader` wrappers for `stdout` and `stderr` (if `subprocess.PIPE` is passed to *stdout* and *stderr* arguments).

Retorna una instancia de `Process`.

Véase la documentación de `loop.subprocess_exec()` para los otros parámetros.

If the process object is garbage collected while the process is still running, the child process will be killed.

Distinto en la versión 3.10: Se eliminó el parámetro *loop*.

async asyncio.create_subprocess_shell(*cmd*, *stdin=None*, *stdout=None*, *stderr=None*, *limit=None*, ***kwds*)

Ejecuta el comando de shell *cmd*.

The *limit* argument sets the buffer limit for `StreamReader` wrappers for `stdout` and `stderr` (if `subprocess.PIPE` is passed to *stdout* and *stderr* arguments).

Retorna una instancia de `Process`.

Véase la documentación de `loop.subprocess_shell()` para los otros parámetros.

If the process object is garbage collected while the process is still running, the child process will be killed.

Importante

Es la responsabilidad de la aplicación asegurarse que todos los espacios en blanco y caracteres especiales estén citados apropiadamente para evitar vulnerabilidades de inyección de instrucciones [https://es.wikipedia.org/wiki/Inyecci%C3%B3n_de_c%C3%B3digo#Inyecci%C3%B3n_de_instrucciones]. La función `shlex.quote()` puede ser usada para escapar caracteres en blanco y caracteres especiales de *shell* en cadenas de caracteres a ser usadas para construir comandos de *shell*.

Distinto en la versión 3.10: Se eliminó el parámetro *loop*.

Nota

Los subprocesos están disponibles para Windows si se utiliza un `ProactorEventLoop`. Consulte Soporte de subprocesos en Windows para obtener más detalles.

Ver también

asyncio también tiene las siguientes APIs de *bajo nivel* para trabajar con sub-procesos: `loop.subprocess_exec()`, `loop.subprocess_shell()`, `loop.connect_read_pipe()`, `loop.connect_write_pipe()`, así como también los Sub-procesos de Transportes y los Sub-procesos de Protocolos.

Constantes

asyncio.subprocess.PIPE

Se le puede pasar a los parámetros *stding*, *stdout* o *stderr*.

If *PIPE* is passed to *stdin* argument, the `Process.stdin` attribute will point to a `StreamWriter` instance.

If *PIPE* is passed to *stdout* or *stderr* arguments, the `Process.stdout` and `Process.stderr` attributes will point to `StreamReader` instances.

asyncio.subprocess.STDOUT

Un valor especial que puede ser usado como el argumento de *stderr* e indica que los errores estándar deben ser redireccionados en la salida estándar.

asyncio.subprocess.DEVNULL

Un valor especial que puede ser usado como el argumento de *stdin*, *stdout*, *stderr* para procesar funciones de creación. Indica que el archivo especial `os.devnull` será usado para la correspondiente transmisión del sub-proceso.

Interactuando con Subprocesos

Las dos funciones `create_subprocess_exec()` y `create_subprocess_shell()` retornan instancias de la clase *Process*. *Process* es un envoltorio de alto nivel que permite comunicarse con los subprocesos y estar atento a sus términos.

class asyncio.subprocess.Process

An object that wraps OS processes created by the `create_subprocess_exec()` and `create_subprocess_shell()` functions.

Esta clase está diseñada para tener un API similar a la clase `subprocess.Popen`, pero hay algunas diferencias notables:

- a diferencia de Popen, las instancias de Process no tiene un equivalente del método `poll()`;
- los métodos `communicate()` y `wait()` no tienen un parámetro *timeout*: use la función `wait_for()`;
- el método `Process.wait()` es asíncrono, mientras que el método `subprocess.Popen.wait()` es implementado como un bucle de espera activa;
- el parámetro *universal_newlines* no es compatible.

Esta clase no es segura para subprocesos (not thread safe).

Véase también la sección Subprocesos e Hilos.

async wait()

Espera a que el proceso hijo termine.

Define y retorna el atributo `returncode`.

Nota

Este método puede bloquearse cuando usa `stdout=PIPE` o `stderr=PIPE` y el proceso hijo genera tanta salida que se bloquea esperando a que la tubería de búfer del SO acepte más datos. Use el método `communicate()` cuando use tuberías para evitar esta condición.

async communicate(*input=None*)

Interactuar con el proceso:

1. envía datos a *stdin* (si *input* no es `None`);
2. cierra *stdin*;
3. lee datos desde *stdout* y *stderr*, hasta que el llegue al EOF;

4. espera a que el proceso termine.

El argumento opcional *input* son los datos (objetos **`bytes`**) que serán enviados a los procesos hijos.

Retorna una tupla `(stdout_data, stderr_data)`.

Si se lanza la excepción **`BrokenPipeError`** o **`ConnectionResetError`** cuando se escribe *input* en *stdin*, la excepción es ignorada. Esta condición ocurre cuando el proceso finaliza antes de que todos los datos sean escritos en *stdin*.

Si se desea enviar datos al *stdin* del proceso, el proceso necesita ser creado con `stdin=PIPE`. De manera similar, para obtener cualquier cosa excepto `None` en la tupla del resultado, el proceso tiene que ser creado con los argumentos `stdout=PIPE` y/o `stderr=PIPE`.

Tenga en cuenta que los datos leídos son almacenados en memoria, por lo que no utilice este método si el tamaño de los datos es más grande o ilimitado.

Distinto en la versión 3.12: stdin gets closed when `input=None` too.

send_signal(*signal*)

Envíe la señal *signal* al proceso hijo.

Nota

On Windows, **`SIGTERM`** is an alias for **`terminate()`**. `CTRL_C_EVENT` and `CTRL_BREAK_EVENT` can be sent to processes started with a *creationflags* parameter which includes `CREATE_NEW_PROCESS_GROUP`.

terminate()

Para al proceso hijo.

On POSIX systems this method sends **`SIGTERM`** to the child process.

On Windows the Win32 API function **`TerminateProcess()`** is called to stop the child process.

kill()

Mata el proceso hijo.

On POSIX systems this method sends **`SIGKILL`** to the child process.

En Windows este método es un alias para **`terminate()`**.

stdin

Standard input stream (**`StreamWriter`**) or `None` if the process was created with `stdin=None`.

stdout

Standard output stream (**`StreamReader`**) or `None` if the process was created with `stdout=None`.

stderr

Standard error stream (**`StreamReader`**) or `None` if the process was created with `stderr=None`.

Advertencia

Utilice el método **`communicate()`** en vez de **`process.stdin.write()`**, **`await process.stdout.read()`** o **`await process.stderr.read`**. Esto evita bloqueos debido a que los flujos pausan la lectura o escritura y bloqueando al proceso hijo.

pid

Número de identificación del Proceso (PID por sus siglas en inglés).

Note that for processes created by the **`create_subprocess_shell()`** function, this attribute is the PID of the spawned shell.

returncode

Código de retorno del proceso cuando finaliza.

Un valor `None` indica que el proceso todavía no ha finalizado.

For processes created with **`create_subprocess_exec()`**, a negative value `-N` indicates that the child was terminated by signal `N` (POSIX only).

For processes created with **`create_subprocess_shell()`**, the return code reflects the exit status of the shell itself (e.g. `/bin/sh`), which may map signals to codes such as `128+N`. See the documentation of the shell (for example, the Bash manual's Exit Status) for details.

Subprocesos y Hilos

Por defecto el bucle de eventos de asyncio estándar permite correr subprocesos desde hilos diferentes.

En Windows, sólo **`ProactorEventLoop`** proporciona subprocesos (por defecto), **`SelectorEventLoop`** no es compatible con subprocesos.

Tenga en cuenta que implementaciones alternativas del bucle de eventos pueden tener limitaciones propias; por favor consulte su documentación.

Ver también

La sección Concurrencia y multihilos en asyncio.

Ejemplos

Un ejemplo usando la clase **`Process`** para controlar un subproceso y la clase **`StreamReader`** para leer de su salida estándar.

El subproceso es creado por la función **`create_subprocess_exec()`**:

```
import asyncio
import sys

async def get_date():
    code = 'import datetime as dt; print(dt.datetime.now

    # Create the subprocess; redirect the standard outpu
    # into a pipe.
    proc = await asyncio.create_subprocess_exec(
        sys.executable, '-c', code,
        stdout=asyncio.subprocess.PIPE)

    # Read one line of output.
    data = await proc.stdout.readline()
    line = data.decode('ascii').rstrip()

    # Wait for the subprocess exit.
    await proc.wait()
    return line

date = asyncio.run(get_date())
print(f"Current date: {date}")
```

Véase también los ejemplos de prueba escritos usando APIs de bajo nivel.

Colas

Código fuente: Lib/asyncio/queue.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/queue.py]

Las colas asyncio son diseñadas para ser similares a clases del módulo `queue`. Sin embargo las colas asyncio no son seguras para hilos, son diseñadas para usar específicamente en código async/await.

Nota que los métodos de colas de asyncio no tienen un parámetro *timeout*; usa la función `asyncio.wait_for()` para hacer operaciones de cola con un tiempo de espera.

Ver también la sección Examples a continuación.

Cola

class asyncio.Queue(*maxsize=0*)

Una cola primero en entrar, primero en salir (PEPS, o *FIFO* en inglés).

Si *maxsize* es menor que o igual a cero, el tamaño de la cola es infinito. Si es un entero mayor a `0`, entonces `await put()` se bloquea cuando una cola alcanza su *maxsize* hasta que un elemento es removido por `get()`.

Diferente de los subprocesos de la biblioteca estándar `queue`, el tamaño de la cola siempre es conocido y puede ser retornado llamando el método `qsize()`.

Distinto en la versión 3.10: Excluido el parámetro *loop*.

Esta clase es no segura para hilos.

maxsize

Número de ítems permitidos en la cola.

empty()

Retorna `True` si la cola es vacía, o `False` en caso contrario.

full()

Retorna `True` si hay **`maxsize`** ítems en la cola.

If the queue was initialized with `maxsize=0` (the default), then **`full()`** never returns `True`.

async get()

Remueve y retorna un ítem de la cola. Si la cola es vacía, espera hasta que un ítem esté disponible.

Raises **`QueueShutDown`** if the queue has been shut down and is empty, or if the queue has been shut down immediately.

get_nowait()

Retorna un ítem si uno está inmediatamente disponible, de otra manera levanta **`QueueEmpty`**.

async join()

Se bloquea hasta que todos los ítems en la cola han sido recibidos y procesados.

El conteo de tareas no terminadas sube siempre que un ítem es agregado a la cola. El conteo baja siempre que la ejecución de una corrutina **`task_done()`** para indicar que el ítem fue recuperado y todo el trabajo en él está completo. Cuando el conteo de tareas inacabadas llega a cero, **`join()`** se desbloquea.

async put(*item*)

Pone un ítem en la cola. Si la cola está completa, espera hasta que una entrada vacía esté disponible antes de agregar el ítem.

Raises `QueueShutDown` if the queue has been shut down.

put_nowait(*item*)

Pone un ítem en la cola sin bloquearse.

Si no hay inmediatamente disponibles entradas vacías, lanza `QueueFull`.

qsize()

Retorna el número de ítems en la cola.

shutdown(*immediate=False*)

Put a `Queue` instance into a shutdown mode.

The queue can no longer grow. Future calls to `put()` raise `QueueShutDown`. Currently blocked callers of `put()` will be unblocked and will raise `QueueShutDown` in the formerly awaiting task.

If *immediate* is false (the default), the queue can be wound down normally with `get()` calls to extract tasks that have already been loaded.

And if `task_done()` is called for each remaining task, a pending `join()` will be unblocked normally.

Once the queue is empty, future calls to `get()` will raise `QueueShutDown`.

If *immediate* is true, the queue is terminated immediately. The queue is drained to be completely empty and the count of unfinished tasks is reduced by the number of tasks drained. If unfinished tasks is zero, callers of `join()` are unblocked. Also, blocked callers of `get()` are unblocked and will raise `QueueShutDown` because the queue is empty.

Use caution when using `join()` with *immediate* set to true. This unblocks the join even when no work has been done on the tasks, violating the usual invariant for joining a queue.

Added in version 3.13.

task_done()

Indicate that a formerly enqueued work item is complete.

Used by queue consumers. For each `get()` used to fetch a work item, a subsequent call to `task_done()` tells the queue that the processing on the work item is complete.

Si un `join()` está actualmente bloqueando, éste se resumirá cuando todos los ítems han sido procesados (implicado que un método `task_done()` fue recibido por cada ítem que ha sido `put()` en la cola.

Lanza `ValueError` si fue llamado más veces que los ítems en la cola.

Cola de prioridad

class asyncio.PriorityQueue

Una variante de `Queue`; recupera entradas en su orden de prioridad (el más bajo primero).

Las entradas son típicamente tuplas de la forma `(priority_number, data)`.

Cola UEPA (o *LIFO* en inglés)

class asyncio.LifoQueue

Una variante de una `Queue` que recupera primero el elemento agregado más reciente (último en entrar, primero en salir).

Excepciones

exception asyncio.QueueEmpty

Esta excepción es lanzada cuando el método `get_nowait()` es ejecutado en una cola vacía.

exception asyncio.QueueFull

Las excepciones son lanzadas cuando el método `put_nowait()` es lanzado en una cola que ha alcanzado su *maxsize*.

exception asyncio.QueueShutDown

Exception raised when `put()` or `get()` is called on a queue which has been shut down.

Added in version 3.13.

Ejemplos

Las colas pueden ser usadas para distribuir cargas de trabajo entre múltiples tareas concurrentes:

```
import asyncio
import random
import time


async def worker(name, queue):
    while True:
        # Get a "work item" out of the queue.
        sleep_for = await queue.get()

        # Sleep for the "sleep_for" seconds.
        await asyncio.sleep(sleep_for)

        # Notify the queue that the "work item" has been
        queue.task_done()

        print(f'{name} has slept for {sleep_for:.2f} sec


async def main():
    # Create a queue that we will use to store our "work
    queue = asyncio.Queue()

    # Generate random timings and put them into the queu
    total_sleep_time = 0
    for _ in range(20):
        sleep_for = random.uniform(0.05, 1.0)
        total_sleep_time += sleep_for
        queue.put_nowait(sleep_for)

    # Create three worker tasks to process the queue con
    tasks = []
    for i in range(3):
        task = asyncio.create_task(worker(f'worker-{i}',
        tasks.append(task)

    # Wait until the queue is fully processed.
    started_at = time.monotonic()
    await queue.join()
    total_slept_for = time.monotonic() - started_at

    # Cancel our worker tasks.
    for task in tasks:
        task.cancel()
    # Wait until all worker tasks are cancelled.
    await asyncio.gather(*tasks, return_exceptions=True)

    print('====')
    print(f'3 workers slept in parallel for {total_slept
    print(f'total expected sleep time: {total_sleep_time

asyncio.run(main())
```

Excepciones

Código Fuente Lib/asyncio/exceptions.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/exceptions.py]

---

exception asyncio.TimeoutError

Un alias obsoleto de `TimeoutError`, lanzado cuando la operación ha superado el plazo establecido.

Distinto en la versión 3.11: Esta clase es un alias de `TimeoutError`.

exception asyncio.CancelledError

La operación ha sido cancelada.

Esta excepción se puede capturar para realizar operaciones personalizadas cuando se cancelan las tareas de asyncio. En casi todas las situaciones, la excepción debe volver a lanzarse.

Distinto en la versión 3.8: `CancelledError` es ahora una subclase de `BaseException` en lugar de `Exception`.

exception asyncio.InvalidStateError

Estado Interno no válido de `Task` o `Future`.

Se puede lanzar en situaciones como establecer un valor de resultado para un objeto *Future* que ya tiene un valor de resultado establecido.

exception asyncio.SendfileNotAvailableError

La llamada al sistema «sendfile» no esta disponible desde el *socket* o tipo de archivo dado.

Una subclase de `RuntimeError`.

exception asyncio.IncompleteReadError

La operación de lectura solicitada no se completó completamente.

Lanzado por la asyncio stream APIs.

La excepción es una subclase de `EOFError`.

expected

El número total (`int`) de bytes esperados.

partial

Un cadena de `bytes` leída antes de que alcance al final del flujo.

exception asyncio.LimitOverrunError

Alcanzó el límite de tamaño del búfer mientras buscaba un separador.

Lanzado por asyncio stream APIs.

consumed

El número total de bytes que se consumirán.

Call Graph Introspection

Source code: Lib/asyncio/graph.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/graph.py]

---

asyncio has powerful runtime call graph introspection utilities to trace the entire call graph of a running *coroutine* or *task*, or a suspended *future*. These utilities and the underlying machinery can be used from within a Python program or by external profilers and debuggers.

Added in version 3.14.

asyncio.print_call_graph(*future=None*, */*, ***, *file=None*, *depth=1*, *limit=None*)

> Print the async call graph for the current task or the provided **`Task`** or **`Future`**.
>
> This function prints entries starting from the top frame and going down towards the invocation point.
>
> The function receives an optional *future* argument. If not passed, the current running task will be used.
>
> If the function is called on *the current task*, the optional keyword-only *depth* argument can be used to skip the specified number of frames from top of the stack.
>
> If the optional keyword-only *limit* argument is provided, each call stack in the resulting graph is truncated to include at most `abs(limit)` entries. If *limit* is positive, the entries left are the closest to the invocation point. If *limit* is negative, the topmost entries are left. If *limit* is omitted or `None`, all entries are present. If *limit* is `0`, the call stack is not printed at all, only «awaited by» information is printed.

If *file* is omitted or `None`, the function will print to `sys.stdout`.

Example:

The following Python code:

```
import asyncio

async def test():
    asyncio.print_call_graph()

async def main():
    async with asyncio.TaskGroup() as g:
        g.create_task(test(), name='test')

asyncio.run(main())
```

will print:

```
* Task(name='test', id=0x1039f0fe0)
+ Call stack:
|   File 't2.py', line 4, in async test()
+ Awaited by:
   * Task(name='Task-1', id=0x103a5e060)
      + Call stack:
      |   File 'taskgroups.py', line 107, in async
      |   File 't2.py', line 7, in async main()
```

asyncio.format_call_graph(*future=None*, */*, ***, *depth=1*, *limit=None*)

Like `print_call_graph()`, but returns a string. If *future* is `None` and there's no current task, the function returns an empty string.

asyncio.capture_call_graph(*future=None*, */*, ***, *depth=1*, *limit=None*)

Capture the async call graph for the current task or the provided `Task` or `Future`.

The function receives an optional *future* argument. If not passed, the current running task will be used. If there's no current task, the function returns `None`.

If the function is called on *the current task*, the optional keyword-only *depth* argument can be used to skip the specified number of frames from top of the stack.

Returns a `FutureCallGraph` data class object:

- `FutureCallGraph(future, call_stack, awaited_by)`

  Where *future* is a reference to a **`Future`** or a **`Task`** (or their subclasses.)

  `call_stack` is a tuple of `FrameCallGraphEntry` objects.

  `awaited_by` is a tuple of `FutureCallGraph` objects.

- `FrameCallGraphEntry(frame)`

  Where *frame* is a frame object of a regular Python function in the call stack.

Low level utility functions

To introspect an async call graph asyncio requires cooperation from control flow structures, such as **`shield()`** or **`TaskGroup`**. Any time an intermediate **`Future`** object with low-level APIs like **`Future.add_done_callback()`** is involved, the following two functions should be used to inform asyncio about how exactly such intermediate future objects are connected with the tasks they wrap or control.

asyncio.future_add_to_awaited_by(*future*, *waiter*, */*)

Record that *future* is awaited on by *waiter*.

Both *future* and *waiter* must be instances of **`Future`** or **`Task`** or their subclasses, otherwise the call would have no effect.

A call to `future_add_to_awaited_by()` must be followed by an eventual call to the **`future_discard_from_awaited_by()`** function with the same arguments.

asyncio.future_discard_from_awaited_by(*future*, *waiter*, /)

Record that *future* is no longer awaited on by *waiter*.

Both *future* and *waiter* must be instances of **`Future`** or **`Task`** or their subclasses, otherwise the call would have no effect.

Event loop

Código fuente: Lib/asyncio/events.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/events.py], Lib/asyncio/base_events.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/base_events.py]

Prólogo

El bucle de eventos es el núcleo de cada aplicación asyncio. Los bucles de eventos ejecutan tareas asíncronas y llamadas de retorno, realizan operaciones de E/S de red y ejecutan subprocesos.

Los desarrolladores de aplicaciones normalmente deberían usar las funciones asyncio de alto nivel, como: `asyncio.run()`, y rara vez deberían necesitar hacer referencia al objeto de bucle o llamar a sus métodos. Esta sección esta dirigida principalmente a autores de código de nivel inferior, bibliotecas y frameworks, quienes necesitan un control mas preciso sobre el comportamiento del bucle de eventos.

Obtención del bucle de eventos

Las siguientes funciones de bajo nivel se pueden utilizar para obtener, establecer o crear un bucle de eventos:

asyncio.get_running_loop()

> Retorna el bucle de eventos en ejecución en el hilo del sistema operativo actual.
>
> Raise a `RuntimeError` if there is no running event loop.
>
> This function can only be called from a coroutine or a callback.

Added in version 3.7.

asyncio.get_event_loop()

Obtiene bucle de eventos actual.

When called from a coroutine or a callback (e.g. scheduled with call_soon or similar API), this function will always return the running event loop.

If there is no running event loop set, the function will return the result of the `get_event_loop_policy().get_event_loop()` call.

Dado que esta función tiene un comportamiento bastante complejo (especialmente cuando están en uso las políticas de bucle de eventos personalizadas), usar la función **`get_running_loop()`** es preferible antes que **`get_event_loop()`** en corrutinas y llamadas de retorno.

As noted above, consider using the higher-level **`asyncio.run()`** function, instead of using these lower level functions to manually create and close an event loop.

Distinto en la versión 3.14: Raises a **`RuntimeError`** if there is no current event loop.

Nota

The **`asyncio`** policy system is deprecated and will be removed in Python 3.16; from there on, this function will return the current running event loop if present else it will return the loop set by **`set_event_loop()`**.

asyncio.set_event_loop(*loop*)

Set *loop* as the current event loop for the current OS thread.

asyncio.new_event_loop()

Crea y retorna un nuevo objeto de bucle de eventos.

Tenga en cuenta que el comportamiento de las funciones `get_event_loop()`, `set_event_loop()`, y `new_event_loop()` puede ser modificado mediante establecienda una política de bucle de eventos personalizada.

Contenidos

Esta página de documentación contiene las siguientes secciones:

- La sección Métodos del bucle de eventos es la documentación de referencia de las APIs del bucle de eventos;
- La sección Callback Handles documenta las instancias `Handle` y `TimerHandle` las cuales son retornadas por métodos planificados como `loop.call_soon()` y `loop.call_later()`;
- La sección Objetos del servidor documenta tipos retornados por los métodos del bucle de eventos como `loop.create_server()`;
- La sección Implementaciones de bucle de eventos documenta las clases `SelectorEventLoop` y `ProactorEventLoop`;
- La sección Ejemplos muestra como trabajar con algunas APIs de bucle de eventos.

Event loop methods

Los bucles de eventos tienen APIs de **bajo nivel** para lo siguiente:

- Iniciar y para el bucle
- Programación de llamadas de retorno
- Planificando llamadas retardadas
- Creating futures and tasks
- Abriendo conexiones de red
- Creando servidores de red
- Transfiriendo archivos
- TLS upgrade
- Viendo descriptores de archivos
- Trabajar con objetos sockets directamente
- DNS
- Trabajando con tuberías

- Señales Unix
- Ejecutando código en un hilos o grupos de procesos
- Error handling API
- Habilitando el modo depuración
- Running subprocesses

Iniciar y para el bucle

loop.run_until_complete(*future*)

Se ejecuta hasta que *future* (una instancia de `Future`) se haya completado.

Si el argumento es un objeto corrutina está implícitamente planificado para ejecutarse como una `asyncio.Task`.

Retorna el resultado del Futuro o genera una excepción.

loop.run_forever()

Ejecuta el bucle de eventos hasta que `stop()` es llamado.

If `stop()` is called before `run_forever()` is called, the loop will poll the I/O selector once with a timeout of zero, run all callbacks scheduled in response to I/O events (and those that were already scheduled), and then exit.

Si `stop()` es llamado mientras `run_forever()` se está ejecutando, el loop ejecutará el lote actual de llamadas y después finalizará. Tenga en cuenta que llamadas planificadas por otras llamadas no se ejecutarán en este caso; en su lugar, ellas correrán la próxima vez que `run_forever()` o `run_until_complete()` sean llamados.

loop.stop()

Detener el bucle de eventos.

loop.is_running()

Retorna `True` si el bucle de eventos esta en ejecución actualmente.

loop.is_closed()

Retorna `True` si el bucle de eventos se cerró.

loop.close()

Cierra el bucle de eventos.

El bucle no debe estar en ejecución cuando se llama a esta función. Cualquier llamada de retorno pendiente será descartada.

Este método limpia todas las colas y apaga el ejecutor, pero no espera a que el ejecutor termine.

Este método es idempotente e irreversible. No se debe llamar ningún otro método después que el bucle de eventos es cerrado.

async loop.shutdown_asyncgens()

Schedule all currently open asynchronous generator objects to close with an **`aclose()`** call. After calling this method, the event loop will issue a warning if a new asynchronous generator is iterated. This should be used to reliably finalize all scheduled asynchronous generators.

Tenga en cuenta que no hay necesidad de llamar esta función cuando **`asyncio.run()`** es utilizado.

Ejemplo:

```
try:
    loop.run_forever()
finally:
    loop.run_until_complete(loop.shutdown_asyncgens
    loop.close()
```

Added in version 3.6.

async loop.shutdown_default_executor(*timeout=None*)

Schedule the closure of the default executor and wait for it to join all of the threads in the `ThreadPoolExecutor`. Once this method has been called, using the default executor with `loop.run_in_executor()` will raise a `RuntimeError`.

The *timeout* parameter specifies the amount of time (in `float` seconds) the executor will be given to finish joining. With the default, `None`, the executor is allowed an unlimited amount of time.

If the *timeout* is reached, a `RuntimeWarning` is emitted and the default executor is terminated without waiting for its threads to finish joining.

Nota

Do not call this method when using `asyncio.run()`, as the latter handles default executor shutdown automatically.

Added in version 3.9.

Distinto en la versión 3.12: Added the *timeout* parameter.

Programación de llamadas de retorno

loop.call_soon(*callback*, **args*, *context=None*)

Programa el *callback* (retrollamada) callback para que se llame con argumentos *args* en la próxima iteración del ciclo de eventos.

Return an instance of `asyncio.Handle`, which can be used later to cancel the callback.

Llamadas que son ejecutadas en el orden en el que fueron registradas. Cada llamada será ejecutada exactamente una sola vez.

The optional keyword-only *context* argument specifies a custom `contextvars.Context` for the *callback* to run in. Callbacks use the current context when no *context* is provided.

Unlike **`call_soon_threadsafe()`**, this method is not thread-safe.

loop.call_soon_threadsafe(*callback*, **args*, *context=None*)

A thread-safe variant of **`call_soon()`**. When scheduling callbacks from another thread, this function *must* be used, since **`call_soon()`** is not thread-safe.

This function is safe to be called from a reentrant context or signal handler, however, it is not safe or fruitful to use the returned handle in such contexts.

Lanza **`RuntimeError`** si se llama en un bucle que ha sido cerrado. Esto puede suceder en un hilo secundario cuando la aplicación principal se está apagando.

Vea sección concurrencia y multiproceso de la documentación.

Distinto en la versión 3.7: Fue agregado el parámetro solo de palabra clave *context*. Vea **PEP 567** [https://peps.python.org/pep-0567/] para mas detalles.

Nota

La mayoría de las funciones planificadas de **`asyncio`** no permiten pasar argumentos de palabra clave. Para hacer eso utilice **`functools.partial()`**:

```
# will schedule "print("Hello", flush=True)"
loop.call_soon(
    functools.partial(print, "Hello", flush=True))
```

El uso de objetos parciales es usualmente mas conveniente que utilizar lambdas, ya que asyncio puede renderizar mejor objetos parciales en mensajes de depuración y error.

Planificando llamadas retardadas

El bucle de eventos provee mecanismos para planificar funciones de llamadas que serán ejecutadas en algún punto en el futuro. El bucle de eventos usa relojes monotónicos para seguir el tiempo.

loop.call_later(*delay*, *callback*, **args*, *context=None*)

Planifica *callback* para ser ejecutada luego de *delay* número de segundos (puede ser tanto un entero como un flotante).

Una instancia de `asyncio.TimerHandle` es retornada, la que puede ser utilizada para cancelar la ejecución.

callback será ejecutada exactamente una sola vez. Si dos llamadas son planificadas para el mismo momento exacto, el orden en el que son ejecutadas es indefinido.

The optional positional *args* will be passed to the callback when it is called. Use `functools.partial()` to pass keyword arguments to *callback*.

Un argumento *context* opcional y solo de palabra clave que permite especificar una clase `contextvars.Context` personalizada en la cual *callback* será ejecutada. Cuando no se provee *context* el contexto actual es utilizado.

Nota

For performance, callbacks scheduled with **`loop.call_later()`** may run up to one clock-resolution early (see `time.get_clock_info('monotonic').resolution`).

Distinto en la versión 3.7: Fue agregado el parámetro solo de palabra clave *context*. Vea **PEP 567** [https://peps.python.org/pep-0567/] para mas detalles.

Distinto en la versión 3.8: En Python 3.7 y versiones anteriores con la implementación del bucle de eventos predeterminada, el *delay* no puede exceder un día. Esto fue arreglado en

Python 3.8.

loop.call_at(*when*, *callback*, **args*, *context=None*)

Planifica *callback* para ser ejecutada en una marca de tiempo absoluta *when* (un entero o un flotante), usando la misma referencia de tiempo que `loop.time()`.

El comportamiento de este método es el mismo que `call_later()`.

Una instancia de `asyncio.TimerHandle` es retornada, la que puede ser utilizada para cancelar la ejecución.

Nota

For performance, callbacks scheduled with **`loop.call_at()`** may run up to one clock-resolution early (see `time.get_clock_info('monotonic').resolution`).

Distinto en la versión 3.7: Fue agregado el parámetro solo de palabra clave *context*. Vea **PEP 567** [https://peps.python.org/pep-0567/] para mas detalles.

Distinto en la versión 3.8: En Python 3.7 y versiones anteriores con la implementación del bucle de eventos predeterminada, la diferencia entre *when* y el tiempo actual no puede exceder un día. Esto fue arreglado en Python 3.8.

loop.time()

Retorna el tiempo actual, como un `float`, de acuerdo al reloj monotónico interno del bucle de evento.

Nota

Distinto en la versión 3.8: En Python 3.7 y versiones anteriores los tiempos de espera (*delay* relativo o *when* absoluto) no deben exceder un día. Esto fue arreglado en Python 3.8.

Ver también

La función `asyncio.sleep()`.

Creating futures and tasks

loop.create_future()

Crea un objeto `asyncio.Future` adjunto al bucle de eventos.

Esta es la manera preferida de crear Futures en asyncio. Esto permite que bucles de eventos de terceros provean implementaciones alternativas del objeto Future (con mejor rendimiento o instrumentación).

Added in version 3.5.2.

loop.create_task(*coro*, ***, *name=None*, *context=None*, *eager_start=None*, ***kwargs*)

Planifica la ejecución de la corrutina *coro*. Retorna un objeto `Task`.

Bucles de eventos de terceros pueden usar sus propias subclases de `Task` por interoperabilidad. En este caso, el tipo de resultado es una subclase de `Task`.

The full function signature is largely the same as that of the `Task` constructor (or factory) - all of the keyword arguments to this function are passed through to that interface.

Si el argumento *name* es provisto y no `None`, se establece como el nombre de la tarea usando `Task.set_name()`.

Un argumento opcional y solo de palabra clave *context* que permite especificar una clase `contextvars.Context` personalizada en la cual *coro* será ejecutada. Cuando no se provee *context* el contexto actual es utilizado.

An optional keyword-only *eager_start* argument allows specifying if the task should execute eagerly during the call to create_task, or be scheduled later. If *eager_start* is not passed the mode set by **`loop.set_task_factory()`** will be used.

Distinto en la versión 3.8: Agregado el parámetro *name*.

Distinto en la versión 3.11: Agregado el parámetro *context*.

Distinto en la versión 3.13.3: Added `kwargs` which passes on arbitrary extra parameters, including `name` and `context`.

Distinto en la versión 3.13.4: Rolled back the change that passes on *name* and *context* (if it is None), while still passing on other arbitrary keyword arguments (to avoid breaking backwards compatibility with 3.13.3).

Distinto en la versión 3.14: All *kwargs* are now passed on. The *eager_start* parameter works with eager task factories.

loop.set_task_factory(*factory*)

Establece una fábrica de tareas que será utilizada por **`loop.create_task()`**.

If *factory* is `None` the default task factory will be set. Otherwise, *factory* must be a *callable* with the signature matching `(loop, coro, **kwargs)`, where *loop* is a reference to the active event loop, and *coro* is a coroutine object. The callable must pass on all *kwargs*, and return a **`asyncio.Task`**-compatible object.

Distinto en la versión 3.13.3: Required that all *kwargs* are passed on to **`asyncio.Task`**.

Distinto en la versión 3.13.4: *name* is no longer passed to task factories. *context* is no longer passed to task factories if it is `None`.

Distinto en la versión 3.14: *name* and *context* are now unconditionally passed on to task factories again.

loop.get_task_factory()

Retorna una fábrica de tareas o `None` si la predefinida está en uso.

Abriendo conexiones de red

async loop.create_connection(*protocol_factory*, *host=None*, *port=None*, ***, *ssl=None*, *family=0*, *proto=0*, *flags=0*, *sock=None*, *local_addr=None*, *server_hostname=None*, *ssl_handshake_timeout=None*, *ssl_shutdown_timeout=None*, *happy_eyeballs_delay=None*, *interleave=None*, *all_errors=False*)

Abre una conexión de transmisión de transporte a una dirección especificada por *host* y *port*.

The socket family can be either **`AF_INET`** or **`AF_INET6`** depending on *host* (or the *family* argument, if provided).

The socket type will be **`SOCK_STREAM`**.

protocol_factory debe ser un ejecutable que retorna una implementación del asyncio protocol.

Este método tratará de establecer la conexión en un segundo plano. Cuando es exitosa, retorna un par `(transport, protocol)`.

La sinopsis cronológica de las operaciones subyacentes es como sigue:

1. La conexión es establecida y un transporte es creado para ello.
2. *protocol_factory* es llamado sin argumentos y se espera que retorne una instancia de protocol.
3. La instancia del protocolo se acopla con el transporte mediante el llamado de su método **`connection_made()`**.
4. Una tupla `(transport, protocol)` es retornada cuando se tiene éxito.

El transporte creado es una transmisión (*stream*) bidireccional que depende de la implementación.

Otros argumentos:

- *ssl*: si se provee y no es falso, un transporte SSL/TLS es creado (de manera predeterminada se crea un transporte TCP plano). Si *ssl* es un objeto `ssl.SSLContext`, este contexto es utilizado para crear el transporte; si *ssl* es `True`, se utiliza un contexto predeterminado retornado por `ssl.create_default_context()`.

  Ver también

  Consideraciones de seguridad SSL/TLS

- *server_hostname* establece o reemplaza el nombre de servidor (*hostname*) contra el cual el certificado del servidor de destino será comparado. Sólo debería ser pasado si *ssl* no es `None`. De manera predeterminada es usado el valor del argumento *host*. Si *host* está vacío, no hay valor predeterminado y debes pasar un valor para *server_hostname*. Si *server_hostname* es una cadena vacía, la comparación de nombres de servidores es deshabilitada (lo que es un riesgo de seguridad serio, permitiendo potenciales ataques de hombre-en-el-medio, *man-in-the-middle attacks*).

- *family*, *proto*, *flags* son dirección de familia, protocolo y banderas opcionales que serán pasadas a través de *getaddrinfo()* para la resolución de *host*. Si están dados, todos ellos deberían ser enteros de las constantes del módulo `socket` correspondiente.

- *happy_eyeballs_delay*, si se proporciona, habilita Happy Eyeballs para esta conexión. Debe ser un número de punto flotante que represente la cantidad de tiempo en segundos para esperar a que se complete un intento de conexión, antes de comenzar el siguiente intento en paralelo. Este es el «Retraso de intento de conexión» como se define en **RFC 8305** [https://datatracker.ietf.org/doc/html/rfc8305.html]. Un valor predeterminado sensato recomendado por el RFC es `0.25` (250 milisegundos).

- *interleave* controla reordenamientos de dirección cuando un nombre de servidor resuelve a múltiples direcciones IP. Si es `0` o no es especificado, no se hace ningún reordenamiento, y las direcciones son intentadas en el orden retornado por `getaddrinfo()`. Si un entero positivo es especificado, las direcciones son intercaladas por dirección de familia, y el entero dado es interpretado como «Número de familias de la primera dirección» (*First Address Family Count*) como es definida en **RFC 8305** [https://datatracker.ietf.org/doc/html/rfc8305.html]. El valor predefinido es `0` si *happy_eyeballs_delay* no es especificado, y `1` si lo es.

- *sock*, si está dado, debe ser un objeto `socket.socket` existente y ya conectado, que será utilizado por el transporte. Si *sock* es dado, ningún *host*, *port*, *family*, *proto*, *flags*, *happy_eyeballs_delay*, *interleave* o *local_addr* deben ser especificados.

  Nota

  El argumento *sock* transfiere la propiedad del socket al transporte creado. Para cerrar el socket, llame al método `close()` del transporte.

- *local_addr*, si está dado, es una tupla `(local_host, local_port)` usada para enlazar el socket localmente. Los *local_host* y *local_port* son buscados usando `getaddrinfo()`, de manera similar que con *host* y *port*.

- *ssl_handshake_timeout* es (para una conexión TLS) el tiempo en segundos a esperar que se complete el *handshake* TLS antes de abortar la conexión. `60.0` segundos si es `None` (predefinido).

- *ssl_shutdown_timeout* es el tiempo en segundos a esperar a que se complete el apagado SSL antes de abortar la conexión. `30.0` segundos si es `None` (predefinido).

- *all_errors* determines what exceptions are raised when a connection cannot be created. By default, only a single `Exception` is raised: the first exception if there is only one or all errors have same message, or a single `OSError` with the error messages combined. When `all_errors` is `True`, an `ExceptionGroup` will be raised containing all exceptions (even if there is only one).

Distinto en la versión 3.5: Agregado el soporte para SSL/TLS en **`ProactorEventLoop`**.

Distinto en la versión 3.6: The socket option socket.TCP_NODELAY is set by default for all TCP connections.

Distinto en la versión 3.7: Agregado el parámetro *ssl_handshake_timeout*.

Distinto en la versión 3.8: Agregados los parámetros *happy_eyeballs_delay* y *interleave*.

Happy Eyeballs Algorithm: Success with Dual-Stack Hosts. When a server's IPv4 path and protocol are working, but the server's IPv6 path and protocol are not working, a dual-stack client application experiences significant connection delay compared to an IPv4-only client. This is undesirable because it causes the dual-stack client to have a worse user experience. This document specifies requirements for algorithms that reduce this user-visible delay and provides an algorithm.

For more information: https://datatracker.ietf.org/doc/html/rfc6555

Distinto en la versión 3.11: Agregado el parámetro *ssl_shutdown_timeout*.

Distinto en la versión 3.12: all_errors was added.

Ver también

La función `open_connection()` es una API alternativa de alto nivel. Retorna un par de (`StreamReader`, `StreamWriter`) que puede ser usado directamente en código async/await.

async loop.create_datagram_endpoint(*protocol_factory*, *local_addr=None*, *remote_addr=None*, ***, *family=0*, *proto=0*, *flags=0*, *reuse_port=None*, *allow_broadcast=None*, *sock=None*)

Crea un datagrama de conexión.

The socket family can be either `AF_INET`, `AF_INET6`, or `AF_UNIX`, depending on *host* (or the *family* argument, if provided).

The socket type will be `SOCK_DGRAM`.

protocol_factory debe ser un ejecutable que retorne una implementación de protocol.

Una tupla de `(transport, protocol)` es retornada cuando se tiene éxito.

Otros argumentos:

- *local_addr*, si está dado, es una tupla `(local_host, local_port)` usada para enlazar el socket localmente. Los *local_host* y *local_port* son buscados utilizando `getaddrinfo()`.

  Nota

  On Windows, when using the proactor event loop with `local_addr=None`, an `OSError` with **`errno.WSAEINVAL`** will be raised when running it.

- *remote_addr*, si está dado, es una tupla `(remote_host, remote_port)` utilizada para conectar el socket a una dirección remota. Los *remote_host* y *remote_port* son buscados utilizando `getaddrinfo()`.

- *family*, *proto*, *flags* son direcciones de familia, protocolo y banderas opcionales que serán pasadas a través de `getaddrinfo()` para la resolución de *host*. Si está dado, estos deben ser todos enteros de las constantes del módulo `socket` correspondiente.
- *reuse_port* tells the kernel to allow this endpoint to be bound to the same port as other existing endpoints are bound to, so long as they all set this flag when being created. This option is not supported on Windows and some Unixes. If the socket.SO_REUSEPORT constant is not defined then this capability is unsupported.
- *allow_broadcast* dice al kernel que habilite este punto de conexión para enviar mensajes a la dirección de transmisión (*broadcast*).
- *sock* puede opcionalmente ser especificado para usar un objeto `socket.socket` preexistente y ya conectado que será utilizado por el transporte. Si están especificados, *local_addr* y *remote_addr* deben ser omitidos (tienen que ser `None`).

Nota

El argumento *sock* transfiere la propiedad del socket al transporte creado. Para cerrar el socket, llame al método `close()` del transporte.

Refiérase a los ejemplos UDP echo client protocol y UDP echo server protocol.

Distinto en la versión 3.4.4: Los parámetros *family*, *proto*, *flags*, *reuse_address*, *reuse_port*, *allow_broadcast* y *sock* fueron agregados.

Distinto en la versión 3.8: Se agregó soporte para Windows.

Distinto en la versión 3.8.1: The *reuse_address* parameter is no longer supported, as using socket.SO_REUSEADDR poses a significant security concern for UDP. Explicitly passing `reuse_address=True` will raise an exception.

Cuando múltiples procesos con UIDs diferentes asignan sockets a una misma dirección socket UDP con `SO_REUSEADDR`, los paquetes entrantes pueden distribuirse aleatoriamente entre los sockets.

For supported platforms, *reuse_port* can be used as a replacement for similar functionality. With *reuse_port*, socket.SO_REUSEPORT is used instead, which specifically prevents processes with differing UIDs from assigning sockets to the same socket address.

Distinto en la versión 3.11: The *reuse_address* parameter, disabled since Python 3.8.1, 3.7.6 and 3.6.10, has been entirely removed.

async loop.create_unix_connection(*protocol_factory*, *path=None*, ***, *ssl=None*, *sock=None*, *server_hostname=None*, *ssl_handshake_timeout=None*, *ssl_shutdown_timeout=None*)

Crear una conexión Unix.

The socket family will be **`AF_UNIX`**; socket type will be **`SOCK_STREAM`**.

Una tupla de `(transport, protocol)` es retornada cuando se tiene éxito.

path es el nombre de un dominio de un socket Unix y es requerido, a menos que un parámetro *sock* sea especificado. Los socket Unix abstractos, **`str`**, **`bytes`**, y **`Path`** son soportados.

Vea la documentación del método **`loop.create_connection()`** para información acerca de los argumentos de este método.

Availability: Unix.

Distinto en la versión 3.7: Agregado el parámetro *ssl_handshake_timeout*. El parámetro *path* ahora puede ser un path-like object.

Distinto en la versión 3.11: Agregado el parámetro *ssl_shutdown_timeout*.

Creando servidores de red

async loop.create_server(*protocol_factory*, *host=None*, *port=None*, ***, *family=socket.AF_UNSPEC*, *flags=socket.AI_PASSIVE*, *sock=None*, *backlog=100*, *ssl=None*, *reuse_address=None*, *reuse_port=None*, *keep_alive=None*, *ssl_handshake_timeout=None*, *ssl_shutdown_timeout=None*, *start_serving=True*)

Create a TCP server (socket type **`SOCK_STREAM`**) listening on *port* of the *host* address.

Retorna un objeto **`Server`**.

Argumentos:

- *protocol_factory* debe ser un ejecutable que retorne una implementación de protocol.
- El parámetro *host* puede ser establecido a distintos tipos que determinan donde el servidor estaría escuchando:
  - Si *host* es una cadena, el servidor TCP está enlazado a una sola interfaz de red especificada por *host*.
  - Si *host* es una secuencia de cadenas, el servidor TCP está enlazado a todas las interfaces de red especificadas por la secuencia.
  - Si *host* es una cadena vacía o `None`, se asumen todas las interfaces y una lista con múltiples sockets será retornada (mas probablemente uno para IPv4 y otro para IPv6).

- El parámetro *port* puede establecerse para especificar en qué puerto debe escuchar el servidor. Si es `0` o `None` (por defecto), se seleccionará un puerto aleatorio no utilizado (tenga en cuenta que si *host* resuelve a múltiples interfaces de red, se seleccionará un puerto aleatorio diferente para cada interfaz).

- *family* can be set to either `socket.AF_INET` or `AF_INET6` to force the socket to use IPv4 or IPv6. If not set, the *family* will be determined from host name (defaults to `AF_UNSPEC`).

- *flags* es una máscara de bits para `getaddrinfo()`.

- *sock* puede ser especificado opcionalmente para usar objetos socket preexistentes. Si se utiliza, entonces *host* y *port* no deben ser especificados.

  Nota

  El argumento *sock* transfiere la propiedad del socket al servidor creado. Para cerrar el socket, llame al método `close()` del servidor.

- *backlog* es el número máximo de conexiones encoladas pasadas a `listen()` (el valor predeterminado es 100).

- *ssl* puede ser establecido como una instancia de `SSLContext` para habilitar TLS sobre las conexiones aceptadas.

- *reuse_address* indica al kernel que reutilice un socket local en estado `TIME_WAIT`, sin esperar que su plazo de ejecución expire. Si no es especificado será establecido automáticamente como `True` en Unix.

- *reuse_port* dice al kernel que habilite este punto de conexión para ser unido al mismo puerto de la misma forma que otros puntos de conexión existentes también están unidos, siempre y cuando todos ellos establezcan esta bandera al ser creados.

- *keep_alive* set to `True` keeps connections active by enabling the periodic transmission of messages.

Distinto en la versión 3.13: Added the *keep_alive* parameter.

- *ssl_handshake_timeout* es (para un servidor TLS) el tiempo en segundos a esperar por el apretón de manos (*handshake*) TLS a ser completado antes de abortar la conexión. `60.0` si es `None` (su valor predeterminado).
- *ssl_shutdown_timeout* es el tiempo en segundos a esperar a que se complete el apagado SSL antes de abortar la conexión. `30.0` segundos si es `None` (predefinido).
- *start_serving* establecido como `True` (de manera predeterminada) produce que los servidores creados comiencen a aceptar conexiones inmediatamente. Si es establecido como `False`, el usuario debe esperar por **`Server.start_serving()`** o **`Server.serve_forever()`** para que el servidor comience a aceptar conexiones.

Distinto en la versión 3.5: Agregado el soporte para SSL/TLS en **`ProactorEventLoop`**.

Distinto en la versión 3.5.1: El parámetro *host* puede ser una secuencia de cadenas.

Distinto en la versión 3.6: Added *ssl_handshake_timeout* and *start_serving* parameters. The socket option socket.TCP_NODELAY is set by default for all TCP connections.

Distinto en la versión 3.11: Agregado el parámetro *ssl_shutdown_timeout*.

Ver también

La función **`start_server()`** es una API alternativa de alto nivel que retorna un par de **`StreamReader`** y **`StreamWriter`** que pueden ser usados en código async/await.

async loop.create_unix_server(*protocol_factory*, *path=None*, ***, *sock=None*, *backlog=100*, *ssl=None*, *ssl_handshake_timeout=None*, *ssl_shutdown_timeout=None*, *start_serving=True*, *cleanup_socket=True*)

Similar to `loop.create_server()` but works with the `AF_UNIX` socket family.

path es el nombre de un dominio de socket Unix, y es requerido a menos que el argumento *sock* sea provisto. Son soportados sockets unix abstractos, `str`, `bytes`, y rutas `Path`.

If *cleanup_socket* is true then the Unix socket will automatically be removed from the filesystem when the server is closed, unless the socket has been replaced after the server has been created.

Vea la documentación de el método `loop.create_server()` para mas información acerca de los argumentos de este método.

Availability: Unix.

Distinto en la versión 3.7: Agregados los parámetros *ssl_handshake_timeout* y *start_serving*. El parámetro *path* ahora puede ser un objeto `Path`.

Distinto en la versión 3.11: Agregado el parámetro *ssl_shutdown_timeout*.

Distinto en la versión 3.13: Added the *cleanup_socket* parameter.

async loop.connect_accepted_socket(*protocol_factory*, *sock*, ***, *ssl=None*, *ssl_handshake_timeout=None*, *ssl_shutdown_timeout=None*)

Envuelve una conexión ya aceptada en un par de transporte/protocolo.

Este método puede ser usado por servidores que acepten conexiones por fuera de asyncio, pero que usen asyncio para manejarlas.

Parámetros:

- *protocol_factory* debe ser un ejecutable que retorne una implementación de protocol.
- *sock* es un objeto socket preexistente retornado por `socket.accept`.

  Nota

  El argumento *sock* transfiere la propiedad del socket al transporte creado. Para cerrar el socket, llame al método `close()` del transporte.

- *ssl* puede ser establecido como un `SSLContext` para habilitar SSL sobre las conexiones aceptadas.
- *ssl_handshake_timeout* es (para una conexión SSL) el tiempo en segundos que se esperará para que se complete el apretón de manos (*handshake*) SSL antes de abortar la conexión. `60.0` si es `None` (su valor predeterminado).
- *ssl_shutdown_timeout* es el tiempo en segundos a esperar a que se complete el apagado SSL antes de abortar la conexión. `30.0` segundos si es `None` (predefinido).

Retorna un par `(transport, protocol)`.

Added in version 3.5.3.

Distinto en la versión 3.7: Agregado el parámetro *ssl_handshake_timeout*.

Distinto en la versión 3.11: Agregado el parámetro *ssl_shutdown_timeout*.

Transfiriendo archivos

async loop.sendfile(*transport*, *file*, *offset=0*, *count=None*, ***, *fallback=True*)

Envía un *file* a través de un *transport*. Retorna el numero total de bytes enviados.

El método usa `os.sendfile()` de alto rendimiento si está disponible.

file debe ser un objeto de archivo regular abierto en modo binario.

offset indica desde donde se empezará a leer el archivo. Si es especificado, *count* es el número total de bytes a transmitir en contraposición con enviar el archivo hasta que se alcance EOF. La posición del archivo es actualizada siempre, incluso cuando este método genere un error, y `file.tell()` puede ser usado para obtener el número de bytes enviados hasta el momento.

fallback establecido como `True` hace que asyncio lea y envíe el archivo manualmente cuando la plataforma no soporta la llamada de envío de archivos del sistema (por ejemplo, Windows o sockets SSL en Unix).

Lanza `SendfileNotAvailableError` si el sistema no soporta la llamada de envío de archivos del sistema y *fallback* es `True`.

Added in version 3.7.

TLS upgrade

async loop.start_tls(*transport*, *protocol*, *sslcontext*, ***, *server_side=False*, *server_hostname=None*, *ssl_handshake_timeout=None*, *ssl_shutdown_timeout=None*)

Actualiza una conexión basada en transporte ya existente a TLS.

Create a TLS coder/decoder instance and insert it between the *transport* and the *protocol*. The coder/decoder implements both *transport*-facing protocol and *protocol*-facing transport.

Return the created two-interface instance. After *await*, the *protocol* must stop using the original *transport* and communicate with the returned object only because the coder caches *protocol*-side data and sporadically exchanges extra TLS session packets with *transport*.

In some situations (e.g. when the passed transport is already closing) this may return `None`.

Parámetros:

- Las instancias *transport* y *protocol* que retornan los métodos como **`create_server()`** y **`create_connection()`**.
- *sslcontext*: una instancia configurada de **`SSLContext`**.
- *server_side* pasa *True* cuando se actualiza una conexión del lado del servidor (como en el caso de una creada por **`create_server()`**).
- *server_hostname*: establece o reemplaza el nombre del host contra el cual se compara el certificado del servidor de destino.
- *ssl_handshake_timeout* es (para una conexión TLS) el tiempo en segundos a esperar que se complete el *handshake* TLS antes de abortar la conexión. `60.0` segundos si es `None` (predefinido).
- *ssl_shutdown_timeout* es el tiempo en segundos a esperar a que se complete el apagado SSL antes de abortar la conexión. `30.0` segundos si es `None` (predefinido).

Added in version 3.7.

Distinto en la versión 3.11: Agregado el parámetro *ssl_shutdown_timeout*.

Viendo descriptores de archivos

loop.add_reader(*fd*, *callback*, **args*)

Empieza a monitorear el descriptor de archivos *fd* para disponibilidad de lectura e invoca *callback* con los argumentos especificados una vez que *fd* está habilitado para ser leído.

Any preexisting callback registered for *fd* is cancelled and replaced by *callback*.

loop.remove_reader(*fd*)

Stop monitoring the *fd* file descriptor for read availability. Returns `True` if *fd* was previously being monitored for reads.

loop.add_writer(*fd*, *callback*, **args*)

Start monitoring the *fd* file descriptor for write availability and invoke *callback* with the specified arguments *args* once *fd* is available for writing.

Any preexisting callback registered for *fd* is cancelled and replaced by *callback*.

Use **`functools.partial()`** para pasar argumentos de palabra clave a *callback*.

loop.remove_writer(*fd*)

Stop monitoring the *fd* file descriptor for write availability. Returns `True` if *fd* was previously being monitored for writes.

Vea también la sección Soporte de plataforma para algunas limitaciones de estos métodos.

Trabajar con objetos sockets directamente

En general, implementaciones de protocolo que usen APIs basadas en transporte como **`loop.create_connection()`** y **`loop.create_server()`** son mas rápidas que aquellas implementaciones que trabajan con directamente con sockets. De cualquier forma, hay algunos casos de uso en los cuales el rendimiento no es crítico, y trabajar directamente con objetos **`socket`** es mas conveniente.

async loop.sock_recv(*sock*, *nbytes*)

Recibe hasta *nbytes* de *sock*. Versión asíncrona de `socket.recv()`.

Retorna los datos recibidos como un objeto bytes.

sock debe ser un socket no bloqueante.

Distinto en la versión 3.7: A pesar de que este método siempre fue documentado como un método de corrutina, los lanzamientos previos a Python 3.7 retornaban un `Future`. Desde Python 3.7 este es un método `async def`.

async loop.sock_recv_into(*sock*, *buf*)

Recibe datos desde *sock* en el búfer *buf*. Modelado después del método bloqueante `socket.recv_into()`.

Retorna el número de bytes escritos en el búfer.

sock debe ser un socket no bloqueante.

Added in version 3.7.

async loop.sock_recvfrom(*sock*, *bufsize*)

Recibe un datagrama de hasta *bufsize* de *sock*. Versión asíncrona de `socket.recvfrom()`.

Retorna una tupla de (datos recibidos, dirección remota).

sock debe ser un socket no bloqueante.

Added in version 3.11.

async loop.sock_recvfrom_into(*sock*, *buf*, *nbytes=0*)

Recibe un datagrama de hasta *nbytes* de *sock* en *buf*. Versión asíncrona de `socket.recvfrom_into()`.

Retorna una dupla de (número de bytes recibidos, dirección remota).

sock debe ser un socket no bloqueante.

Added in version 3.11.

async loop.sock_sendall(*sock*, *data*)

Envía *data* al socket *sock*. Versión asíncrona de **`socket.sendall()`**.

Este método continua enviando al socket hasta que se hayan enviado todos los datos en *data* u ocurra un error. `None` es retornado cuando se tiene éxito. Cuando ocurre un error, se lanza una excepción. Adicionalmente, no hay manera de determinar cuantos datos, si es que se hubo alguno, se procesaron correctamente por el extremo receptor de la conexión.

sock debe ser un socket no bloqueante.

Distinto en la versión 3.7: A pesar de que este método siempre fue documentado como un método de corrutina, antes de Python 3.7 retorna un **`Future`**. Desde Python 3.7, este es un método `async def`.

async loop.sock_sendto(*sock*, *data*, *address*)

Envía un datagrama desde *sock* a *address*. Versión asíncrona de **`socket.sendto()`**.

Retorna el número de bytes enviados.

sock debe ser un socket no bloqueante.

Added in version 3.11.

async loop.sock_connect(*sock*, *address*)

Conecta *sock* a un socket remoto en *address*.

Versión asíncrona de **`socket.connect()`**.

sock debe ser un socket no bloqueante.

Distinto en la versión 3.5.2: `address` ya no necesita ser resuelto. `sock_connect` va a intentar verificar si *address* ya fue resuelto a partir del llamado de **`socket.inet_pton()`**. Si no lo fue, se utilizará **`loop.getaddrinfo()`** ara resolver *address*.

Ver también

`loop.create_connection()` y **`asyncio.open_connection()`**.

async loop.sock_accept(*sock*)

Acepta una conexión. Modelado después del método bloqueante **`socket.accept()`**.

The socket must be bound to an address and listening for connections. The return value is a pair `(conn, address)` where *conn* is a *new* socket object usable to send and receive data on the connection, and *address* is the address bound to the socket on the other end of the connection.

sock debe ser un socket no bloqueante.

Distinto en la versión 3.7: A pesar de que este método siempre fue documentado como un método de corrutina, antes de Python 3.7 retorna un **`Future`**. Desde Python 3.7, este es un método `async def`.

Ver también

`loop.create_server()` y **`start_server()`**.

async loop.sock_sendfile(*sock*, *file*, *offset=0*, *count=None*, ***, *fallback=True*)

Envía un archivo usando **`os.sendfile`** de alto rendimiento si es posible. Retorna el número total de bytes enviados.

Versión asíncrona de `socket.sendfile()`.

sock debe ser un `socket.SOCK_STREAM` `socket` no bloqueante.

file debe ser un objeto de archivo regular abierto en modo binario.

offset indica desde donde se empezará a leer el archivo. Si es especificado, *count* es el número total de bytes a transmitir en contraposición con enviar el archivo hasta que se alcance EOF. La posición del archivo es actualizada siempre, incluso cuando este método genere un error, y `file.tell()` puede ser usado para obtener el número de bytes enviados hasta el momento.

fallback, cuando es establecida como `True`, hace que asyncio lea y escriba el archivo manualmente cuando el sistema no soporta la llamada de envío de archivos del sistema (por ejemplo, Windows o sockets SSL en Unix).

Lanza `SendfileNotAvailableError` si el sistema no soporta la llamada de envío de archivos del sistema *sendfile* y *fallback* es `False`.

sock debe ser un socket no bloqueante.

Added in version 3.7.

DNS

async loop.getaddrinfo(*host*, *port*, ***, *family=0*, *type=0*, *proto=0*, *flags=0*)

Versión asíncrona de `socket.getaddrinfo()`.

async loop.getnameinfo(*sockaddr*, *flags=0*)

Asynchronous version of `socket.getnameinfo()`.

Nota

Both *getaddrinfo* and *getnameinfo* internally utilize their synchronous versions through the loop's default thread pool executor. When this executor is saturated, these methods may experience delays, which higher-level networking libraries may report as increased timeouts. To mitigate this, consider using a custom executor for other user tasks, or setting a default executor with a larger number of workers.

Distinto en la versión 3.7: Ambos métodos *getaddrinfo* y *getnameinfo* siempre fueron documentados para retornar una corrutina, pero antes de Python 3.7 retornaban, de hecho, objetos **`Future`**. A partir de Python 3.7, ambos métodos son corrutinas.

Trabajando con tuberías

async loop.connect_read_pipe(*protocol_factory*, *pipe*)

Registra el fin de lectura de *pipe* en el bucle de eventos.

protocol_factory debe ser un ejecutable que retorna una implementación del asyncio protocol.

pipe es un objeto de tipo archivo.

Retorna un par `(transport, protocol)`, donde *transport* soporta la interface **`ReadTransport`** y *protocol* es un objeto instanciado por *protocol_factory*.

Con el bucle de eventos **`SelectorEventLoop`**, el *pipe* es establecido en modo no bloqueante.

async loop.connect_write_pipe(*protocol_factory*, *pipe*)

Registra el fin de escritura de *pipe* en el bucle de eventos.

protocol_factory debe ser un ejecutable que retorna una implementación del asyncio protocol.

pipe es un objeto de tipo archivo.

Retorna un par `(transport, protocol)`, donde *transport* soporta la interface **`WriteTransport`** y *protocol* es un objeto inicializado por *protocol_factory*.

Con el bucle de eventos `SelectorEventLoop`, el *pipe* es establecido en modo no bloqueante.

Nota

`SelectorEventLoop` no soporta los métodos anteriores en windows. En su lugar, use `ProactorEventLoop` para Windows.

Ver también

Los métodos `loop.subprocess_exec()` y `loop.subprocess_shell()`.

Señales Unix

loop.add_signal_handler(*signum*, *callback*, **args*)

Set *callback* as the handler for the *signum* signal, passing *args* as positional arguments.

La llamada será invocada por *loop*, junto con otras llamadas encoladas y corrutinas ejecutables de ese bucle de eventos. A menos que los gestores de señal la registren usando `signal.signal()`, una llamada registrada con esta función tiene permitido interactuar con el bucle de eventos.

Lanza `ValueError` si el número de señal es invalido o inalcanzable. Lanza `RuntimeError` si hay algún problema preparando el gestor.

Use `functools.partial()` para pasar argumentos de palabra clave a *callback*.

Como `signal.signal()`, esta función debe ser invocada en el hilo principal.

loop.remove_signal_handler(*sig*)

Elimina el gestor para la señal *sig*.

Retorna `True` si el gestor de señal fue eliminado, o `False` si no se estableció gestor para la señal dada.

Availability: Unix.

Ver también

El módulo **`signal`**.

Ejecutando código en un hilos o grupos de procesos

awaitable loop.run_in_executor(*executor*, *func*, **args*)

Arrange for *func* to be called in the specified executor passing *args* as positional arguments.

The *executor* argument should be an **`concurrent.futures.Executor`** instance. The default executor is used if *executor* is `None`. The default executor can be set by **`loop.set_default_executor()`**, otherwise, a **`concurrent.futures.ThreadPoolExecutor`** will be lazy-initialized and used by **`run_in_executor()`** if needed.

Ejemplo:

```
import asyncio
import concurrent.futures

def blocking_io():
    # File operations (such as logging) can block t
    # event loop: run them in a thread pool.
    with open('/dev/urandom', 'rb') as f:
        return f.read(100)

def cpu_bound():
    # CPU-bound operations will block the event loo
    # in general it is preferable to run them in a
    # process pool.
    return sum(i * i for i in range(10 ** 7))

async def main():
    loop = asyncio.get_running_loop()

    ## Options:

    # 1. Run in the default loop's executor:
    result = await loop.run_in_executor(
        None, blocking_io)
    print('default thread pool', result)

    # 2. Run in a custom thread pool:
    with concurrent.futures.ThreadPoolExecutor() as
        result = await loop.run_in_executor(
            pool, blocking_io)
        print('custom thread pool', result)

    # 3. Run in a custom process pool:
    with concurrent.futures.ProcessPoolExecutor() a
        result = await loop.run_in_executor(
            pool, cpu_bound)
        print('custom process pool', result)

    # 4. Run in a custom interpreter pool:
    with concurrent.futures.InterpreterPoolExecutor
        result = await loop.run_in_executor(
            pool, cpu_bound)
        print('custom interpreter pool', result)

if __name__ == '__main__':
    asyncio.run(main())
```

Tenga en cuenta que la protección del punto de entrada (`if __name__ == '__main__'`) es requerida para la opción 3 debido a las peculiaridades de **`multiprocessing`**, que es utilizado por **`ProcessPoolExecutor`**. Vea Importación segura del módulo principal.

Este método retorna un objeto **`asyncio.Future`**.

Use **`functools.partial()`** para pasar argumentos de palabra clave a *func*.

Distinto en la versión 3.5.3: **`loop.run_in_executor()`** ya no configura el `max_workers` del ejecutor del grupo de subprocesos que crea, sino que lo deja en manos del ejecutor del grupo de subprocesos (**`ThreadPoolExecutor`**) para establecer el valor por defecto.

loop.set_default_executor(*executor*)

Set *executor* as the default executor used by **`run_in_executor()`**. *executor* must be an instance of **`ThreadPoolExecutor`**, which includes **`InterpreterPoolExecutor`**.

Distinto en la versión 3.11: executor debe ser una instancia de **`ThreadPoolExecutor`**.

Error handling API

Permite personalizar como son manejadas las excepciones en el bucle de eventos.

loop.set_exception_handler(*handler*)

Establece *handler* como el nuevo gestor de excepciones del bucle de eventos.

Si *handler* es `None`, se establecerá el gestor de excepciones predeterminado. De otro modo, *handler* debe ser un invocable con la misma firma `(loop, context)`, donde `loop` es una referencia al bucle de eventos activo, y `context` es un objeto `dict` que contiene los detalles de la excepción (vea la documentación de **`call_exception_handler()`** para detalles acerca del contexto).

If the handler is called on behalf of a **`Task`** or **`Handle`**, it is run in the **`contextvars.Context`** of that task or callback handle.

Distinto en la versión 3.12: The handler may be called in the **`Context`** of the task or handle where the exception originated.

loop.get_exception_handler()

Retorna el gesto de excepciones actual, o `None` si no fue establecido ningún gestor de excepciones personalizado.

Added in version 3.5.2.

loop.default_exception_handler(*context*)

Gestor de excepciones por defecto.

Esto es llamado cuando ocurre una excepción y no se estableció ningún gestor de excepciones. Esto puede ser llamado por un gestor de excepciones personalizado que quiera cambiar el comportamiento del gestor predeterminado.

El parámetro *context* tiene el mismo significado que en **`call_exception_handler()`**.

loop.call_exception_handler(*context*)

Llama al gestor de excepciones del bucle de eventos actual.

context es un objeto `dict` conteniendo las siguientes claves (en futuras versiones de Python podrían introducirse nuevas claves):

- "message": Mensaje de error;
- "exception" (opcional): Objeto de excepción;
- "future" (opcional): instancia de **`asyncio.Future`**;
- "task" (opcional): instancia de **`asyncio.Task`**;
- "handle" (opcional): instancia de **`asyncio.Handle`**;
- "protocol" (opcional): instancia de Protocol;
- "transport" (opcional): instancia de Transport;

- “socket” (opcional): instancia de `socket.socket`;
- “source_traceback” (optional): Traceback of the source;
- “handle_traceback” (optional): Traceback of the handle;

- “asyncgen” (opcional): Generador asíncrono que causó la excepción.

Nota

This method should not be overloaded in subclassed event loops. For custom exception handling, use the `set_exception_handler()` method.

Habilitando el modo depuración

loop.get_debug()

Obtiene el modo depuración (`bool`) del bucle de eventos.

El valor predeterminado es `True` si la variable de entorno **PYTHONASYNCIODEBUG** es establecida a una cadena no vacía, de otro modo será `False`.

loop.set_debug(*enabled: bool*)

Establece el modo de depuración del bucle de eventos.

Distinto en la versión 3.7: El nuevo Python Modo de Desarrollo ahora también se puede usar para habilitar el modo de depuración.

loop.slow_callback_duration

This attribute can be used to set the minimum execution duration in seconds that is considered «slow». When debug mode is enabled, «slow» callbacks are logged.

Default value is 100 milliseconds.

Ver también

El modo depuración de asyncio.

Running subprocesses

Los métodos descritos en esta subsección son de bajo nivel. En código async/await regular considere usar las convenientes funciones de alto nivel `asyncio.create_subprocess_shell()` y `asyncio.create_subprocess_exec()`.

Nota

En Windows, el bucle de eventos por defecto `ProactorEventLoop` soporta subprocesos, mientras que `SelectorEventLoop` no. Vea Soporte de subprocesos en Windows para más detalles.

async loop.subprocess_exec(*protocol_factory*, **args*, *stdin=subprocess.PIPE*, *stdout=subprocess.PIPE*, *stderr=subprocess.PIPE*, ***kwargs*)

Crea un subproceso de uno o mas argumentos de cadena especificados por *args*.

args debe ser una lista de cadenas representadas por:

- `str`;
- o `bytes`, codificados a la codificación del sistema de archivos.

La primer cadena especifica el programa ejecutable, y las cadenas restantes especifican los argumentos. En conjunto, los argumentos de cadena forman el `argv` del programa.

Esto es similar a la clase de la librería estándar `subprocess.Popen` llamada con `shell=False` y la lista de cadenas pasadas como el primer argumento; de cualquier forma, cuando `Popen` toma un sólo argumento que es una lista de cadenas, *subprocess_exec* toma múltiples cadenas como argumentos.

El *protocol_factory* debe ser un ejecutable que retorne una subclase de la clase **`asyncio.SubprocessProtocol`**.

Otros parámetros:

- *stdin* puede ser cualquier de estos:
  - a file-like object
  - an existing file descriptor (a positive integer), for example those created with **`os.pipe()`**
  - la constante **`subprocess.PIPE`** (predeterminado) que creará una tubería nueva y la conectará,
  - el valor `None` que hará que el subproceso herede el descriptor de archivo de este proceso
  - la constante **`subprocess.DEVNULL`** que indica que el archivo especial **`os.devnull`** será utilizado
- *stdout* puede ser cualquier de estos:
  - a file-like object
  - la constante **`subprocess.PIPE`** (predeterminado) que creará una tubería nueva y la conectará,
  - el valor `None` que hará que el subproceso herede el descriptor de archivo de este proceso
  - la constante **`subprocess.DEVNULL`** que indica que el archivo especial **`os.devnull`** será utilizado
- *stderr* puede ser cualquier de estos:
  - a file-like object
  - la constante **`subprocess.PIPE`** (predeterminado) que creará una tubería nueva y la conectará,
  - el valor `None` que hará que el subproceso herede el descriptor de archivo de este proceso
  - la constante **`subprocess.DEVNULL`** que indica que el archivo especial **`os.devnull`** será

utilizado

    - ○ la constante `subprocess.STDOUT` que conectará el flujo de errores predeterminado al flujo de salida predeterminado del proceso

- El resto de argumentos de palabra clave son pasados a `subprocess.Popen` sin interpretación, excepto por *bufsize*, *universal_newlines*, *shell*, *text*, *encoding* y *errors*, que no deben ser especificados en lo absoluto.

  La API subproceso `asyncio` no soporta decodificar los flujos como texto. `bytes.decode()` puede ser usado para convertir a texto los bytes retornados por el flujo.

If a file-like object passed as *stdin*, *stdout* or *stderr* represents a pipe, then the other side of this pipe should be registered with `connect_write_pipe()` or `connect_read_pipe()` for use with the event loop.

Vea el constructor de la clase `subprocess.Popen` para documentación acerca de otros argumentos.

Retorna un par de `(transport, protocol)`, donde *transport* se ajusta a la clase base `asyncio.SubprocessTransport` y *protocol* es un objeto instanciado por *protocol_factory*.

If the transport is closed or is garbage collected, the child process is killed if it is still running.

async loop.subprocess_shell(*protocol_factory*, *cmd*, ***, *stdin=subprocess.PIPE*, *stdout=subprocess.PIPE*, *stderr=subprocess.PIPE*, ***kwargs*)

Crea un subproceso desde *cmd*, que puede ser una cadena `str` o `bytes` codificado a la codificación del sistema de archivos, usando la sintaxis «shell» de la plataforma.

Esto es similar a la clase de la librería estándar `subprocess.Popen` llamada con `shell=True`.

El *protocol_factory* debe ser un ejecutable que retorne una subclase de la clase `asyncio.SubprocessProtocol`.

Vea `subprocess_exec()` para mas detalles acerca de los argumentos restantes.

Retorna un par de `(transport, protocol)`, donde *transport* se ajusta a la clase base `SubprocessTransport` y *protocol* es un objeto instanciado por *protocol_factory*.

If the transport is closed or is garbage collected, the child process is killed if it is still running.

Nota

Es responsabilidad de la aplicación asegurar que todos los espacios en blanco y caracteres especiales estén escapados correctamente para evitar vulnerabilidades de inyección de código [https://en.wikipedia.org/wiki/Shell_injection#Shell_injection]. La función `shlex.quote()` puede ser usada para escapar apropiadamente espacios en blanco y caracteres especiales en cadenas que van a ser usadas para construir comandos de consola.

Callback handles

class asyncio.Handle

Un objeto de contenedor de llamada retornado por `loop.call_soon()`, `loop.call_soon_threadsafe()`.

get_context()

Return the `contextvars.Context` object associated with the handle.

Added in version 3.12.

cancel()

Cancela la llamada. Si la llamada ya fue cancelada o ejecutada, este método no tiene efecto.

cancelled()

Retorna `True` si la llamada fue cancelada.

Added in version 3.7.

class asyncio.TimerHandle

Un objeto de contenedor de llamada retornado por **`loop.call_later()`**, and **`loop.call_at()`**.

Esta clase es una subclase de **`Handle`**.

when()

Retorna el tiempo de una llamada planificada como **`float`** segundos.

El tiempo es una marca de tiempo absoluta, usando la misma referencia de tiempo que **`loop.time()`**.

Added in version 3.7.

Server objects

Los objetos de servidor son creados por las funciones **`loop.create_server()`**, **`loop.create_unix_server()`**, **`start_server()`**, y **`start_unix_server()`**.

Do not instantiate the **`Server`** class directly.

class asyncio.Server

Los objetos *Server* son gestores de asíncronos de contexto. Cuando son usados en una declaración `async with`, está garantizado que el objeto Servidor está cerrado y no está aceptando nuevas conexiones cuando la declaración `async with` es completada:

```
srv = await loop.create_server(...)

async with srv:
    # some code
# At this point, srv is closed and no longer accept
```

Distinto en la versión 3.7: El objeto Servidor es un gestor asíncrono de contexto desde Python 3.7.

Distinto en la versión 3.11: This class was exposed publicly as `asyncio.Server` in Python 3.9.11, 3.10.3 and 3.11.

close()

Deja de servir: deja de escuchar sockets y establece el atributo **`sockets`** a `None`.

Los sockets que representan conexiones entrantes existentes de clientes se dejan abiertas.

The server is closed asynchronously; use the **`wait_closed()`** coroutine to wait until the server is closed (and no more connections are active).

close_clients()

Close all existing incoming client connections.

Calls **`close()`** on all associated transports.

`close()` should be called before **`close_clients()`** when closing the server to avoid races with new clients connecting.

Added in version 3.13.

abort_clients()

Close all existing incoming client connections immediately, without waiting for pending operations to complete.

Calls **`abort()`** on all associated transports.

`close()` should be called before **`abort_clients()`** when closing the server to avoid races with new clients connecting.

Added in version 3.13.

get_loop()

Retorna el bucle de eventos asociado con el objeto Servidor.

Added in version 3.7.

async start_serving()

Comienza a aceptar conexiones.

Este método es idempotente, así que puede ser llamado cuando el servidor ya está sirviendo.

El parámetro sólo de palabra clave *start_serving* de **`loop.create_server()`** y **`asyncio.start_server()`** permite crear un objeto Servidor que no está aceptando conexiones inicialmente. En este caso `Server.start_serving()`, o **`Server.serve_forever()`** pueden ser usados para hacer que el servidor empiece a aceptar conexiones.

Added in version 3.7.

async serve_forever()

Comienza a aceptar conexiones hasta que la corrutina sea cancelada. La cancelación de la tarea `serve_forever` hace que el servidor sea cerrado.

Este método puede ser llamado si el servidor ya está aceptando conexiones. Solamente una tarea `serve_forever` puede existir para un objeto *Server*.

Ejemplo:

```
async def client_connected(reader, writer):
    # Communicate with the client with
```

```
        # reader/writer streams.  For example:
        await reader.readline()

    async def main(host, port):
        srv = await asyncio.start_server(
            client_connected, host, port)
        await srv.serve_forever()

    asyncio.run(main('127.0.0.1', 0))
```

Added in version 3.7.

is_serving()

Retorna `True` si el servidor está aceptando nuevas conexiones.

Added in version 3.7.

async wait_closed()

Wait until the **`close()`** method completes and all active connections have finished.

sockets

List of socket-like objects, `asyncio.trsock.TransportSocket`, which the server is listening on.

Distinto en la versión 3.7: Antes de Python 3.7 `Server.sockets` solía retornar directamente una lista interna de servidores socket. En 3.7 se retorna una copia de esa lista.

Event loop implementations

asyncio viene con dos implementaciones diferentes del bucle de eventos: **`SelectorEventLoop`** y **`ProactorEventLoop`**.

By default asyncio is configured to use **`EventLoop`**.

class asyncio.SelectorEventLoop

A subclass of **`AbstractEventLoop`** based on the **`selectors`** module.

Usa el *selector* disponible mas eficiente para la plataforma dada. También es posible configurar manualmente la implementación exacta del selector a utilizar:

```
import asyncio
import selectors

async def main():
   ...

loop_factory = lambda: asyncio.SelectorEventLoop(se
asyncio.run(main(), loop_factory=loop_factory)
```

Availability: Unix, Windows.

class asyncio.ProactorEventLoop

A subclass of **`AbstractEventLoop`** for Windows that uses «I/O Completion Ports» (IOCP).

Availability: Windows.

Ver también

MSDN documentation on I/O Completion Ports [https://learn.microsoft.com/windows/win32/fileio/i-o-completion-ports].

class asyncio.EventLoop

An alias to the most efficient available subclass of **`AbstractEventLoop`** for the given platform.

It is an alias to **`SelectorEventLoop`** on Unix and **`ProactorEventLoop`** on Windows.

Added in version 3.13.

class asyncio.AbstractEventLoop

Clase base abstracta para bucles de evento compatibles con asyncio.

La sección Event loop methods lista todos los métodos que una implementación alternativa de `AbstractEventLoop` debería tener definidos.

Examples

Nótese que todos los ejemplos en esta sección muestran **a propósito** como usar las APIs de bucle de eventos de bajo nivel, como ser **`loop.run_forever()`** y **`loop.call_soon()`**. Aplicaciones asyncio modernas raramente necesitan ser escritas de esta manera; considere utilizar funciones de alto nivel como **`asyncio.run()`**.

Hola Mundo con call_soon()

Un ejemplo usando el método **`loop.call_soon()`** para planificar una llamada. La llamada muestra `"Hello World"` y luego para el bucle de eventos:

```
import asyncio

def hello_world(loop):
    """A callback to print 'Hello World' and stop the ev
    print('Hello World')
    loop.stop()

loop = asyncio.new_event_loop()

# Schedule a call to hello_world()
loop.call_soon(hello_world, loop)

# Blocking call interrupted by loop.stop()
try:
    loop.run_forever()
```

```
finally:
    loop.close()
```

Ver también

Un ejemplo similar de Hola Mundo creado con una corrutina y la función `run()`.

Muestra la fecha actual con call_later()

Un ejemplo de llamada mostrando la fecha actual cada un segundo. La llamada usa el método `loop.call_later()` para volver a planificarse después de 5 segundos, y después para el bucle de eventos:

```
import asyncio
import datetime as dt

def display_date(end_time, loop):
    print(dt.datetime.now())
    if (loop.time() + 1.0) < end_time:
        loop.call_later(1, display_date, end_time, loop)
    else:
        loop.stop()

loop = asyncio.new_event_loop()

# Schedule the first call to display_date()
end_time = loop.time() + 5.0
loop.call_soon(display_date, end_time, loop)

# Blocking call interrupted by loop.stop()
try:
    loop.run_forever()
finally:
    loop.close()
```

Ver también

Un ejemplo similar a fecha actual creado con una corrutina y la función `run()`.

Mirar un descriptor de archivo para leer eventos

Espera hasta que el descriptor de archivo reciba algún dato usando el método `loop.add_reader()` y entonces cierra el bucle de eventos:

```
import asyncio
from socket import socketpair

# Create a pair of connected file descriptors
rsock, wsock = socketpair()

loop = asyncio.new_event_loop()

def reader():
    data = rsock.recv(100)
    print("Received:", data.decode())

    # We are done: unregister the file descriptor
    loop.remove_reader(rsock)

    # Stop the event loop
    loop.stop()

# Register the file descriptor for read event
loop.add_reader(rsock, reader)

# Simulate the reception of data from the network
loop.call_soon(wsock.send, 'abc'.encode())

try:
    # Run the event loop
    loop.run_forever()
finally:
    # We are done. Close sockets and the event loop.
    rsock.close()
    wsock.close()
    loop.close()
```

Ver también

- Un ejemplo similar usando transportes, protocolos y el método `loop.create_connection()`.
- Otro ejemplo similar usando la función de alto nivel `asyncio.open_connection()` y transmisiones.

Establece los gestores de señal para SIGINT y SIGTERM

(This `signal` example only works on Unix.)

Register handlers for signals `SIGINT` and `SIGTERM` using the `loop.add_signal_handler()` method:

```
import asyncio
import functools
import os
import signal

def ask_exit(signame, loop):
    print("got signal %s: exit" % signame)
    loop.stop()

async def main():
    loop = asyncio.get_running_loop()

    for signame in {'SIGINT', 'SIGTERM'}:
        loop.add_signal_handler(
            getattr(signal, signame),
            functools.partial(ask_exit, signame, loop))

    await asyncio.sleep(3600)
```

```
print("Event loop running for 1 hour, press Ctrl+C to ir
print(f"pid {os.getpid()}: send SIGINT or SIGTERM to exi

asyncio.run(main())
```

Futures

Código fuente: Lib/asyncio/futures.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/futures.py], Lib/asyncio/base_futures.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/base_futures.py]

Los objetos *Future* se utilizan para conectar **código basado en retrollamadas de bajo nivel** (*low-level callback-based code*) con código *async/await* de alto nivel.

Funciones Future

asyncio.isfuture(*obj*)

Retorna `True` si *obj* es uno de los siguientes:

- una instancia de **`asyncio.Future`**,
- una instancia de **`asyncio.Task`**,
- un objeto tipo Future con un atributo `_asyncio_future_blocking`.

Added in version 3.5.

asyncio.ensure_future(*obj*, ***, *loop=None*)

Retorna:

- el argumento *obj* inalterado, si *obj* es una **`Future`**, **`Task`**, o un objeto tipo Future (esto se puede verificar con **`isfuture()`**.)
- un objeto **`Task`** envolviendo *obj*, si *obj* es una corrutina (esto se puede verificar con **`iscoroutine()`**); en este caso, la corrutina será programada por `ensure_future()`.
- un objeto **`Task`** que aguardará a *obj*, si *obj* es aguardable (esto se puede verificar con

`inspect.isawaitable()`.)

Si *obj* no es ninguno de los superiores, se lanzará `TypeError`.

Importante

Guarda una referencia al resultado de esta función, evita que una tarea desaparezca a mitad de su ejecución.

See also the `create_task()` function which is the preferred way for creating new tasks or use `asyncio.TaskGroup` which keeps reference to the task internally.

Distinto en la versión 3.5.1: La función acepta cualquier objeto awaitable.

Obsoleto desde la versión 3.10: Se emite una advertencia de desaprobación si *obj* no es un objeto tipo Future y no se especifica *loop* y no hay un bucle de eventos en ejecución.

asyncio.wrap_future(*future*, *, *loop=None*)

Envuelve un objeto `concurrent.futures.Future` en un objeto `asyncio.Future`.

Obsoleto desde la versión 3.10: Se emite una advertencia de desaprobación si *future* no es un objeto tipo Future y *loop* no se especifica y no hay un bucle de eventos en ejecución.

Objeto Future

class asyncio.Future(*, *loop=None*)

Un Future representa un resultado eventual de una operación asíncrona. No es seguro en hilos (*thread-safe*).

Future es un objeto awaitable. Las corrutinas pueden esperar a objetos Future hasta que obtengan un resultado o excepción, o hasta que se cancelen. Un Future puede esperarse varias veces y el resultado es el mismo.

Normalmente, los Futures se utilizan para permitir que código basado en retrollamadas de bajo nivel (*low-level callback-based code*) (por ejemplo, en protocolos implementados utilizando *asyncio* transports) interactúe con código *async/await* de alto nivel.

Es recomendable no exponer nunca objetos Future en APIs expuestas al usuario, y la forma recomendada de crear un objeto Future es llamando a `loop.create_future()`. De esta forma, implementaciones alternativas de bucles de eventos (*event loop*) pueden inyectar sus propias implementaciones optimizadas de un objeto Future.

Distinto en la versión 3.7: Añadido soporte para el módulo `contextvars`.

Obsoleto desde la versión 3.10: Se emite una advertencia de desaprobación si no se especifica *loop* y no hay un bucle de eventos en ejecución.

result()

Retorna el resultado del Future.

Si el Future es *done* y tiene un resultado establecido por el método `set_result()`, el valor resultante es retornado.

Si el Future es *done* y tiene una excepción establecida por el método `set_exception()`, este método lanzará esta excepción.

Si un evento es *cancelled*, este método lanzará una excepción `CancelledError`.

If the Future's result isn't yet available, this method raises an `InvalidStateError` exception.

set_result(*result*)

Marca el Future como *done* y establece su resultado.

Raises an **`InvalidStateError`** error if the Future is already *done*.

set_exception(*exception*)

Marca el Future como *done* y establece una excepción.

Raises an **`InvalidStateError`** error if the Future is already *done*.

done()

Retorna `True` si el Future está *done*.

Un Future está *done* si estaba *cancelled* o si tiene un resultado o excepción establecidos mediante llamadas a **`set_result()`** o **`set_exception()`**.

cancelled()

Retorna `True` si el Future fue *cancelled*.

El método suele utilizarse para comprobar que un Future no es *cancelled* antes de establecer un resultado o excepción al mismo:

```
if not fut.cancelled():
    fut.set_result(42)
```

add_done_callback(*callback*, ***, *context=None*)

Añade una retrollamada (*callback*) a ser ejecutada cuando el Future es *done*.

La retrollamada (*callback*) es llamada con el objeto Future como su único argumento.

Si el Future ya es *done* cuando se llama a este método, la retrollamada (*callback*) es programada con **`loop.call_soon()`**.

Un argumento opcional de contexto, por palabra clave, permite especificar un `contextvars.Context` personalizado para ser ejecutado en la retrollamada (*callback*). El contexto actual se utiliza cuando no se provee un contexto (*context*).

`functools.partial()` se puede utilizar para dar parámetros a la retrollamada (*callback*), por ejemplo:

```
# Call 'print("Future:", fut)' when "fut" is do
fut.add_done_callback(
    functools.partial(print, "Future:"))
```

Distinto en la versión 3.7: El parámetro de contexto (*context*) por palabra clave fue añadido. Ver **PEP 567** [https://peps.python.org/pep-0567/] para más detalles.

remove_done_callback(*callback*)

Elimina la retrollamada (*callback*) de la lista de retrollamadas.

Retorna el número de retrollamadas (*callbacks*) eliminadas, que normalmente es 1, excepto si una retrollamada fue añadida más de una vez.

cancel(*msg=None*)

Cancela el Future y programa retrollamadas (*callbacks*).

Si el Future ya está *done* o *cancelled*, retorna `False`. De lo contrario, cambia el estado del Future a *cancelled*, programa las retrollamadas, y retorna `True`.

The optional string argument *msg* is passed as the argument to the `CancelledError` exception raised when a cancelled Future is awaited.

Distinto en la versión 3.9: Se agregó el parámetro *msg*.

exception()

Retorna la excepción definida en este Future.

La excepción (o `None` si no se había establecido ninguna excepción) es retornada sólo si Future es *done*.

Si un evento es *cancelled*, este método lanzará una excepción **`CancelledError`**.

Si el Future todavía no es *done*, este método lanza una excepción **`InvalidStateError`**.

get_loop()

Retorna el bucle de eventos (*event loop*) al cual el objeto Future está asociado.

Added in version 3.7.

Este ejemplo crea un objeto Future, crea y programa una Task asíncrona para establecer el resultado para el Future, y espera hasta que el Future tenga un resultado:

```
async def set_after(fut, delay, value):
    # Sleep for *delay* seconds.
    await asyncio.sleep(delay)

    # Set *value* as a result of *fut* Future.
    fut.set_result(value)

async def main():
    # Get the current event loop.
    loop = asyncio.get_running_loop()

    # Create a new Future object.
    fut = loop.create_future()

    # Run "set_after()" coroutine in a parallel Task.
    # We are using the low-level "loop.create_task()" AP
    # we already have a reference to the event loop at h
    # Otherwise we could have just used "asyncio.create_
    loop.create_task(
        set_after(fut, 1, '... world'))

    print('hello ...')

    # Wait until *fut* has a result (1 second) and print
    print(await fut)

asyncio.run(main())
```

Importante

El objeto Future fue diseñado para imitar a `concurrent.futures.Future`. Entre las principales diferencias están:

- al contrario que Futures de *asyncio*, las instancias de `concurrent.futures.Future` no son aguardables (*await*).
- `asyncio.Future.result()` y `asyncio.Future.exception()` no aceptan el argumento *timeout*.
- `asyncio.Future.result()` y `asyncio.Future.exception()` lanzan una excepción `InvalidStateError` cuando el Future no es *done*.
- Las retrollamadas (*callbacks*) registradas con `asyncio.Future.add_done_callback()` no son llamadas inmediatamente, sino que son programadas con `loop.call_soon()`.
- *asyncio* Future no es compatible con las funciones `concurrent.futures.wait()` ni `concurrent.futures.as_completed()`.
- `asyncio.Future.cancel()` acepta un argumento opcional `msg`, pero `concurrent.futures.Future.cancel()` no.

Transportes y protocolos

Prefacio

Los transportes y protocolos son utilizados por las APIs de **bajo nivel** de los bucles de eventos, como `loop.create_connection()`. Utilizan un estilo de programación basado en retrollamadas y permiten implementaciones de alto rendimiento de protocolos de red o IPC (p. ej. HTTP).

Esencialmente, los transportes y protocolos solo deben usarse en bibliotecas y frameworks, nunca en aplicaciones asyncio de alto nivel.

Esta página de la documentación cubre tanto Transports como Protocols.

Introducción

En el nivel más alto, el transporte se ocupa de *cómo* se transmiten los bytes, mientras que el protocolo determina *qué* bytes transmitir (y hasta cierto punto cuándo).

Una forma diferente de decir lo mismo: Un transporte es una abstracción para un socket (o un punto final de E/S similar) mientras que un protocolo es una abstracción para una aplicación, desde el punto de vista del transporte.

Otro punto de vista más es que las interfaces de transporte y protocolo definen juntas una interfaz abstracta para usar E/S de red y E/S entre procesos.

Siempre existe una relación 1:1 entre el transporte y los objetos protocolo: el protocolo llama a los métodos del transporte para enviar datos, mientras que el transporte llama a los métodos del protocolo para enviarle los datos que se han recibido.

La mayoría de los métodos del bucle de eventos orientados a la conexión (como `loop.create_connection()`) aceptan generalmente un argumento *protocol_factory* que es usado para crear un objeto *Protocol* para una conexión aceptada, representada por un objeto *Transport*. Estos métodos suelen retornar una tupla de la forma `(transporte, protocolo)`.

Contenidos

Esta página de la documentación contiene las siguientes secciones:

- La sección Transports documenta las clases asyncio `BaseTransport`, `ReadTransport`, `WriteTransport`, `Transport`, `DatagramTransport` y `SubprocessTransport`.
- La sección Protocols documenta las clases asyncio `BaseProtocol`, `Protocol`, `BufferedProtocol`, `DatagramProtocol` y `SubprocessProtocol`.
- La sección Examples muestra cómo trabajar con transportes, protocolos y las APIs de bajo nivel del bucle de eventos.

Transportes

Código fuente: Lib/asyncio/transports.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/transports.py]

Los transportes son clases proporcionadas por `asyncio` para abstraer varios tipos de canales de comunicación.

Los objetos transporte siempre son instanciados por un bucle de eventos asyncio.

asyncio implementa transportes para TCP, UDP, SSL y pipes de subprocesos. Los métodos disponibles en un transporte dependen del tipo de transporte.

Las clases transporte no son seguras en hilos.

Jerarquía de transportes

class asyncio.BaseTransport

Clase base para todos los transportes. Contiene métodos que todos los transportes asyncio comparten.

class asyncio.WriteTransport(*BaseTransport*)

Un transporte base para conexiones de solo escritura.

Las instancias de la clase *WriteTransport* se retornan desde el método del bucle de eventos `loop.connect_write_pipe()` y también se utilizan en métodos relacionados con subprocesos como `loop.subprocess_exec()`.

class asyncio.ReadTransport(*BaseTransport*)

Un transporte base para conexiones de solo lectura.

Las instancias de la clase *ReadTransport* se retornan desde el método del bucle de eventos `loop.connect_read_pipe()` y también se utilizan en métodos relacionados con subprocesos como `loop.subprocess_exec()`.

class asyncio.Transport(*WriteTransport*, *ReadTransport*)

Interfaz que representa un transporte bidireccional, como una conexión TCP.

El usuario no crea una instancia de transporte directamente; en su lugar se llama a una función de utilidad, pasándole una fábrica de protocolos junto a otra información necesaria para crear el transporte y el protocolo.

Las instancias de la clase *Transport* son retornadas o utilizadas por métodos del bucle de eventos como `loop.create_connection()`, `loop.create_unix_connection()`, `loop.create_server()`, `loop.sendfile()`, etc.

class asyncio.DatagramTransport(*BaseTransport*)

Un transporte para conexiones de datagramas (UDP).

Las instancias de la clase *DatagramTransport* se retornan desde el método `loop.create_datagram_endpoint()` del bucle de eventos.

class asyncio.SubprocessTransport(*BaseTransport*)

Una abstracción para representar una conexión entre un proceso padre y su proceso OS hijo.

Las instancias de la clase *SubprocessTransport* se retornan desde los métodos del bucle de eventos `loop.subprocess_shell()` y `loop.subprocess_exec()`.

Transporte base

BaseTransport.close()

Cierra el transporte.

Si el transporte tiene un búfer para datos de salida, los datos almacenados en el búfer se eliminarán de forma asincrónica. No se recibirán más datos. Después de que se vacíen todos los datos almacenados en el búfer, el método `protocol.connection_lost()` del protocolo se llamará con `None` como argumento. El transporte no debería ser usado luego de haber sido cerrado.

BaseTransport.is_closing()

Retorna `True` si el transporte se está cerrando o está ya cerrado.

BaseTransport.get_extra_info(*name*, *default=None*)

Retorna información sobre el transporte o los recursos subyacentes que utiliza.

name es una cadena de caracteres que representa la información específica del transporte que se va a obtener.

default es el valor que se retornará si la información no está disponible o si el transporte no admite la consulta con la implementación del bucle de eventos de terceros dada o en la plataforma actual.

Por ejemplo, el siguiente código intenta obtener el objeto socket subyacente del transporte:

```
sock = transport.get_extra_info('socket')
if sock is not None:
    print(sock.getsockopt(...))
```

Categorías de información que se pueden consultar sobre algunos transportes:

- socket:
  - `'peername'`: la dirección remota a la que está conectado el socket, resultado de **`socket.socket.getpeername()`** (`None` en caso de error)
  - `'socket'`: instancia de **`socket.socket`**
  - `'sockname'`: la dirección propia del socket, resultado de **`socket.socket.getsockname()`**
- socket SSL:
  - `'compression'`: el algoritmo de compresión que se está usando como una cadena de caracteres o `None` si la conexión no está comprimida; resultado de **`ssl.SSLSocket.compression()`**
  - `'cipher'`: una tupla de tres valores que contiene el nombre del cifrado que se está utilizando, la versión del protocolo SSL que define su uso y la cantidad de bits de la clave secreta que se utilizan; resultado de **`ssl.SSLSocket.cipher()`**

    - `'peercert'`: certificado de pares; resultado de **`ssl.SSLSocket.getpeercert()`**
    - `'sslcontext'`: instancia de **`ssl.SSLContext`**
    - `'ssl_object'`: instancia de **`ssl.SSLObject`** o **`ssl.SSLSocket`**
- pipe:
    - `'pipe'`: objeto pipe
- subproceso:
    - `'subprocess'`: instancia de **`subprocess.Popen`**

BaseTransport.set_protocol(*protocol*)

Establece un nuevo protocolo.

El cambio de protocolo solo debe realizarse cuando esté documentado que ambos protocolos admiten el cambio.

BaseTransport.get_protocol()

Retorna el protocolo actual.

Transportes de solo lectura

ReadTransport.is_reading()

Retorna `True` si el transporte está recibiendo nuevos datos.

Added in version 3.7.

ReadTransport.pause_reading()

Pausa el extremo receptor del transporte. No se pasarán datos al método **`protocol.data_received()`** del protocolo hasta que se llame a **`resume_reading()`**.

Distinto en la versión 3.7: El método es idempotente, es decir, se puede llamar cuando el transporte ya está en pausa o cerrado.

ReadTransport.resume_reading()

Reanuda el extremo receptor. El método `protocol.data_received()` del protocolo se llamará una vez más si hay algunos datos disponibles para su lectura.

Distinto en la versión 3.7: El método es idempotente, es decir, se puede llamar cuando el transporte está leyendo.

Transportes de solo escritura

WriteTransport.abort()

Cierra el transporte inmediatamente, sin esperar a que finalicen las operaciones pendientes. Se perderán los datos almacenados en el búfer. No se recibirán más datos. El método `protocol.connection_lost()` del protocolo será llamado eventualmente con `None` como argumento.

WriteTransport.can_write_eof()

Retorna `True` si el transporte admite `write_eof()`, en caso contrario `False`.

WriteTransport.get_write_buffer_size()

Retorna el tamaño actual del búfer de salida utilizado por el transporte.

WriteTransport.get_write_buffer_limits()

Obtiene los límites superior e inferior para el control del flujo de escritura. Retorna una tupla `(low, high)` donde *low* ("inferior") y *high* ("superior") son un número de bytes positivo.

Usa `set_write_buffer_limits()` para establecer los límites.

Added in version 3.4.2.

WriteTransport.set_write_buffer_limits(*high=None*, *low=None*)

Establece los límites *high* ("superior") y *low* ("inferior") para el control del flujo de escritura.

Estos dos valores (medidos en número de bytes) controlan cuándo se llaman los métodos `protocol.pause_writing()` y `protocol.resume_writing()` del protocolo . Si se especifica, el límite inferior debe ser menor o igual que el límite superior. Ni *high* ni *low* pueden ser negativos.

`pause_writing()` se llama cuando el tamaño del búfer es mayor o igual que el valor *high* ("superior"). Si se ha pausado la escritura, se llama a `resume_writing()` cuando el tamaño del búfer es menor o igual que el valor *low* ("inferior").

Los valores por defecto son específicos de la implementación. Si solo se proporciona el límite superior, el inferior toma de forma predeterminada un valor específico, dependiente de la implementación, menor o igual que el límite superior. Establecer *high* ("superior") en cero fuerza *low* ("inferior") a cero también y hace que `pause_writing()` sea llamado siempre que el búfer no esté vacío. Establecer *low* ("inferior") en cero hace que `resume_writing()` sea llamado únicamente cuando el búfer esté vacío. El uso de cero para cualquiera de los límites es generalmente subóptimo, ya que reduce las oportunidades para realizar E/S y cálculos simultáneamente.

Usa `get_write_buffer_limits()` para obtener los límites.

WriteTransport.write(*data*)

Escribe los bytes de *data* en el transporte.

Este método no bloquea; almacena los datos en el búfer y organiza que se envíen de forma asincrónica.

WriteTransport.writelines(*list_of_data*)

Escribe una lista (o cualquier iterable) de bytes de datos en el transporte. Esto es funcionalmente equivalente a llamar a `write()` en cada elemento generado por el iterable, pero puede ser implementado de manera más eficiente.

WriteTransport.write_eof()

Cierra el extremo de escritura del transporte después de vaciar todos los datos almacenados en el búfer. Aún es posible recibir datos.

Este método puede lanzar una excepción `NotImplementedError` si el transporte (p. ej. SSL) no soporta conexiones semicerradas (*half-closed*).

Transportes de datagramas

DatagramTransport.sendto(*data*, *addr=None*)

Envía los bytes *data* al par remoto proporcionado por *addr* (una dirección de destino dependiente del transporte). Si *addr* es `None`, los datos se envían a la dirección de destino proporcionada en la creación del transporte.

Este método no bloquea; almacena los datos en el búfer y organiza que se envíen de forma asincrónica.

Distinto en la versión 3.13: This method can be called with an empty bytes object to send a zero-length datagram. The buffer size calculation used for flow control is also updated to account for the datagram header.

DatagramTransport.abort()

Cierra el transporte inmediatamente, sin esperar a que finalicen las operaciones pendientes. Se perderán los datos almacenados en el búfer. No se recibirán más datos. El método `protocol.connection_lost()` del protocolo será llamado eventualmente con `None` como argumento.

Transportes de subprocesos

SubprocessTransport.get_pid()

Retorna la id del subproceso como un número entero.

SubprocessTransport.get_pipe_transport(*fd*)

Retorna el transporte para la pipe de comunicación correspondiente al descriptor de archivo entero *fd*:

- `0`: writable streaming transport of the standard input (*stdin*), or `None` if the subprocess was not created with `stdin=PIPE`
- `1`: readable streaming transport of the standard output (*stdout*), or `None` if the subprocess was not created with `stdout=PIPE`
- `2`: readable streaming transport of the standard error (*stderr*), or `None` if the subprocess was not created with `stderr=PIPE`
- otro *fd*: `None`

SubprocessTransport.get_returncode()

Retorna el código de retorno del subproceso como un entero o `None` si no ha retornado aún, lo que es similar al atributo `subprocess.Popen.returncode`.

SubprocessTransport.kill()

Mata al subproceso.

En los sistemas POSIX, la función envía SIGKILL al subproceso. En Windows, este método es un alias para `terminate()`.

Ver también `subprocess.Popen.kill()`.

SubprocessTransport.send_signal(*signal*)

Envía el número de *señal* al subproceso, como en `subprocess.Popen.send_signal()`.

SubprocessTransport.terminate()

Detiene el subproceso.

On POSIX systems, this method sends `SIGTERM` to the subprocess. On Windows, the Windows API function **`TerminateProcess()`** is called to stop the subprocess.

Ver también `subprocess.Popen.terminate()`.

SubprocessTransport.close()

Mata al subproceso llamando al método `kill()`.

Si el subproceso aún no ha retornado, cierra los transportes de las pipes *stdin*, *stdout* y *stderr*.

Protocolos

Código fuente: Lib/asyncio/protocols.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/protocols.py]

---

asyncio proporciona un conjunto de clases base abstractas que pueden usarse para implementar protocolos de red. Estas clases están destinadas a ser utilizadas junto con los transportes.

Las subclases de las clases abstractas de protocolos base pueden implementar algunos o todos los métodos. Todos estos métodos son retrollamadas: son llamados por los transportes en ciertos eventos, por ejemplo, cuando se reciben algunos datos. Un método del protocolo base debe ser llamado por el transporte correspondiente.

Protocolos base

class asyncio.BaseProtocol

Protocolo base con métodos que comparten todos los demás protocolos.

class asyncio.Protocol(*BaseProtocol*)

La clase base para implementar protocolos de *streaming* (TCP, sockets Unix, etc).

class asyncio.BufferedProtocol(*BaseProtocol*)

Una clase base para implementar protocolos de *streaming* con control manual del búfer de recepción.

class asyncio.DatagramProtocol(*BaseProtocol*)

La clase base para implementar protocolos de datagramas (UDP).

class asyncio.SubprocessProtocol(*BaseProtocol*)

La clase base para implementar protocolos que se comunican con procesos secundarios (pipes unidireccionales).

Protocolo base

Todos los protocolos asyncio pueden implementar las retrollamadas del protocolo base.

Retrollamadas de conexión

Las retrollamadas de conexión son llamadas exactamente una vez por conexión establecida en todos los protocolos. Todas las demás retrollamadas del protocolo solo pueden ser llamadas entre estos dos métodos.

BaseProtocol.connection_made(*transport*)

Se llama cuando se establece una conexión.

El argumento *transport* es el transporte que representa la conexión. El protocolo se encarga de almacenar la referencia a su propio transporte.

BaseProtocol.connection_lost(*exc*)

Se llama cuando la conexión se pierde o se cierra.

El argumento es un objeto excepción o `None`. Esto último significa que se recibió un EOF regular o que la conexión fue cancelada o cerrada por este lado de la conexión.

Retrollamadas de control de flujo

Los transportes pueden llamar a las retrollamadas de control de flujo para pausar o reanudar la escritura llevada a cabo por el protocolo.

Consulta la documentación del método `set_write_buffer_limits()` para obtener más detalles.

BaseProtocol.pause_writing()

> Se llama cuando el búfer del transporte supera el límite superior.

BaseProtocol.resume_writing()

> Se llama cuando el búfer del transporte se vacía por debajo del límite inferior.

Si el tamaño del búfer es igual al límite superior, `pause_writing()` no será llamado: el tamaño del búfer debe superarse estrictamente.

Por el contrario, se llama a `resume_writing()` cuando el tamaño del búfer es igual o menor que el límite inferior. Estas condiciones finales son importantes para garantizar que todo salga como se espera cuando cualquiera de los dos límites sea cero.

Protocolos de *streaming*

Los métodos de eventos, como `loop.create_server()`, `loop.create_unix_server()`, `loop.create_connection()`, `loop.create_unix_connection()`, `loop.connect_accepted_socket()`, `loop.connect_read_pipe()`, y `loop.connect_write_pipe()` aceptan fábricas que retornan protocolos de *streaming*.

Protocol.data_received(*data*)

> Se llama cuando se reciben algunos datos. *data* es un objeto

bytes no vacío que contiene los datos entrantes.

Que los datos se almacenen en un búfer, que se fragmenten o se vuelvan a ensamblar depende del transporte. En general, no debe confiar en semánticas específicas y, en cambio, hacer que su análisis sea genérico y flexible. Sin embargo, los datos siempre se reciben en el orden correcto.

El método se puede llamar un número arbitrario de veces mientras una conexión esté abierta.

Sin embargo, **`protocol.eof_received()`** se llama como máximo una vez. Luego de llamar a `eof_received()`, ya no se llama más a `data_received()`.

Protocol.eof_received()

Se llama cuando el otro extremo indica que no enviará más datos (por ejemplo, llamando a **`transport.write_eof()`** si el otro extremo también usa asyncio).

Este método puede retornar un valor falso (incluido `None`), en cuyo caso el transporte se cerrará solo. Por el contrario, si este método retorna un valor verdadero, el protocolo utilizado determina si se debe cerrar el transporte. Dado que la implementación por defecto retorna `None`, en éste caso, se cierra implícitamente la conexión.

Algunos transportes, incluido SSL, no admiten conexiones semicerradas (*half-closed*), en cuyo caso retornar verdadero desde este método resultará en el cierre de la conexión.

Máquina de estado:

```
start -> connection_made
    [-> data_received]*
    [-> eof_received]?
-> connection_lost -> end
```

Protocolos de *streaming* mediante búfer

Added in version 3.7.

Los protocolos que hacen uso de un búfer se pueden utilizar con cualquier método del bucle de eventos que admita Streaming Protocols.

Las implementaciones de `BufferedProtocol` permiten la asignación manual explícita y el control del búfer de recepción. Los bucles de eventos pueden utilizar el búfer proporcionado por el protocolo para evitar copias de datos innecesarias. Esto puede resultar en una mejora notable del rendimiento de los protocolos que reciben grandes cantidades de datos. Las implementaciones de protocolos sofisticados pueden reducir significativamente la cantidad de asignaciones de búfer.

Las siguientes retrollamadas son llamadas en instancias **`BufferedProtocol`**:

BufferedProtocol.get_buffer(*sizehint*)

> Se llama para asignar un nuevo búfer de recepción.
>
> *sizehint* es el tamaño mínimo recomendado para el búfer retornado. Es aceptable retornar búferes más pequeños o más grandes de lo que sugiere *sizehint*. Cuando se establece en -1, el tamaño del búfer puede ser arbitrario. Es un error retornar un búfer con tamaño cero.
>
> `get_buffer()` debe retornar un objeto que implemente el protocolo de búfer.

BufferedProtocol.buffer_updated(*nbytes*)

> Se llama cuando el búfer se ha actualizado con los datos recibidos.
>
> *nbytes* es el número total de bytes que se escribieron en el búfer.

BufferedProtocol.eof_received()

> Consulte la documentación del método **`protocol.eof_received()`**.

`get_buffer()` se puede llamar un número arbitrario de veces durante una conexión. Sin embargo, `protocol.eof_received()` se llama como máximo una vez y, si se llama, `get_buffer()` y `buffer_updated()` no serán llamados después de eso.

Máquina de estado:

```
start -> connection_made
    [-> get_buffer
        [-> buffer_updated]?
    ]*
    [-> eof_received]?
-> connection_lost -> end
```

Protocolos de datagramas

Las instancias del protocolo de datagramas deben ser construidas por fábricas de protocolos pasadas al método `loop.create_datagram_endpoint()`.

DatagramProtocol.datagram_received(*data*, *addr*)

Se llama cuando se recibe un datagrama. *data* es un objeto bytes que contiene los datos entrantes. *addr* es la dirección del par que envía los datos; el formato exacto depende del transporte.

DatagramProtocol.error_received(*exc*)

Se llama cuando una operación de envío o recepción anterior genera una `OSError`. *exc* es la instancia `OSError`.

Este método se llama en condiciones excepcionales, cuando el transporte (por ejemplo, UDP) detecta que un datagrama no se pudo entregar a su destinatario. Sin embargo, en la mayoría de casos, los datagramas que no se puedan entregar se eliminarán silenciosamente.

Nota

En los sistemas BSD (macOS, FreeBSD, etc.) el control de flujo no es compatible con los protocolos de datagramas, esto se debe a que no hay una forma confiable de detectar fallos de envío causados por escribir demasiados paquetes.

El socket siempre aparece como disponible ("ready") y se eliminan los paquetes sobrantes. Un error `OSError` con `errno` establecido en `errno.ENOBUFS` puede o no ser generado; si se genera, se informará a `DatagramProtocol.error_received()` pero en caso contrario se ignorará.

Protocolos de subprocesos

Las instancias de protocolo de subproceso deben ser construidas por fábricas de protocolos pasadas a los métodos `loop.subprocess_exec()` y `loop.subprocess_shell()`.

SubprocessProtocol.pipe_data_received(*fd*, *data*)

Se llama cuando el proceso hijo escribe datos en su pipe stdout o stderr.

fd es el descriptor de archivo entero de la pipe.

data es un objeto bytes no vacío que contiene los datos recibidos.

SubprocessProtocol.pipe_connection_lost(*fd*, *exc*)

Se llama cuando se cierra una de las pipes que se comunican con el proceso hijo.

fd es el descriptor de archivo entero que se cerró.

SubprocessProtocol.process_exited()

Se llama cuando el proceso hijo ha finalizado.

It can be called before `pipe_data_received()` and `pipe_connection_lost()` methods.

Ejemplos

Servidor de eco TCP

Crear un servidor de eco TCP usando el método `loop.create_server()`, enviar de vuelta los datos recibidos y cerrar la conexión:

```
import asyncio


class EchoServerProtocol(asyncio.Protocol):
    def connection_made(self, transport):
        peername = transport.get_extra_info('peername')
        print('Connection from {}'.format(peername))
        self.transport = transport

    def data_received(self, data):
        message = data.decode()
        print('Data received: {!r}'.format(message))

        print('Send: {!r}'.format(message))
        self.transport.write(data)

        print('Close the client socket')
        self.transport.close()


async def main():
    # Get a reference to the event loop as we plan to us
    # low-level APIs.
    loop = asyncio.get_running_loop()

    server = await loop.create_server(
        EchoServerProtocol,
        '127.0.0.1', 8888)

    async with server:
        await server.serve_forever()


asyncio.run(main())
```

Ver también

El ejemplo Servidor de eco TCP usando streams usa la función de alto nivel `asyncio.start_server()`.

Cliente de eco TCP

Un cliente de eco TCP usando el método `loop.create_connection()`, envía datos y espera hasta que la conexión se cierre:

```
import asyncio


class EchoClientProtocol(asyncio.Protocol):
    def __init__(self, message, on_con_lost):
        self.message = message
        self.on_con_lost = on_con_lost

    def connection_made(self, transport):
        transport.write(self.message.encode())
        print('Data sent: {!r}'.format(self.message))

    def data_received(self, data):
        print('Data received: {!r}'.format(data.decode()

    def connection_lost(self, exc):
        print('The server closed the connection')
        self.on_con_lost.set_result(True)


async def main():
    # Get a reference to the event loop as we plan to us
    # low-level APIs.
    loop = asyncio.get_running_loop()

    on_con_lost = loop.create_future()
    message = 'Hello World!'

    transport, protocol = await loop.create_connection(
        lambda: EchoClientProtocol(message, on_con_lost)
        '127.0.0.1', 8888)

    # Wait until the protocol signals that the connectio
    # is lost and close the transport.
    try:
        await on_con_lost
    finally:
        transport.close()


asyncio.run(main())
```

Ver también

El ejemplo Cliente de eco TCP usando streams usa la función de alto nivel `asyncio.open_connection()`.

Servidor de eco UDP

Un servidor de eco UDP, usando el método `loop.create_datagram_endpoint()`, envía de vuelta los datos recibidos:

```
import asyncio


class EchoServerProtocol:
    def connection_made(self, transport):
        self.transport = transport
    def datagram_received(self, data, addr):
        message = data.decode()
        print('Received %r from %s' % (message, addr))
        print('Send %r to %s' % (message, addr))
        self.transport.sendto(data, addr)


async def main():
    print("Starting UDP server")

    # Get a reference to the event loop as we plan to us
    # low-level APIs.
    loop = asyncio.get_running_loop()

    # One protocol instance will be created to serve all
    # client requests.
    transport, protocol = await loop.create_datagram_enc
        EchoServerProtocol,
        local_addr=('127.0.0.1', 9999))

    try:
        await asyncio.sleep(3600)  # Serve for 1 hour.
    finally:
        transport.close()


asyncio.run(main())
```

Cliente de eco UDP

Un cliente de eco UDP, usando el método **`loop.create_datagram_endpoint()`**, envía datos y cierra el transporte cuando recibe la respuesta:

```
import asyncio


class EchoClientProtocol:
    def __init__(self, message, on_con_lost):
        self.message = message
        self.on_con_lost = on_con_lost
        self.transport = None

    def connection_made(self, transport):
        self.transport = transport
        print('Send:', self.message)
        self.transport.sendto(self.message.encode())

    def datagram_received(self, data, addr):
        print("Received:", data.decode())

        print("Close the socket")
        self.transport.close()

    def error_received(self, exc):
        print('Error received:', exc)

    def connection_lost(self, exc):
        print("Connection closed")
        self.on_con_lost.set_result(True)


async def main():
    # Get a reference to the event loop as we plan to us
    # low-level APIs.
    loop = asyncio.get_running_loop()

    on_con_lost = loop.create_future()
    message = "Hello World!"

    transport, protocol = await loop.create_datagram_enc
        lambda: EchoClientProtocol(message, on_con_lost)
        remote_addr=('127.0.0.1', 9999))

    try:
        await on_con_lost
    finally:
        transport.close()


asyncio.run(main())
```

Conectando sockets existentes

Espera hasta que un socket reciba datos usando el método **`loop.create_connection()`** mediante un protocolo:

```
import asyncio
import socket


class MyProtocol(asyncio.Protocol):

    def __init__(self, on_con_lost):
        self.transport = None
        self.on_con_lost = on_con_lost

    def connection_made(self, transport):
        self.transport = transport

    def data_received(self, data):
        print("Received:", data.decode())

        # We are done: close the transport;
        # connection_lost() will be called automatically
        self.transport.close()

    def connection_lost(self, exc):
        # The socket has been closed
        self.on_con_lost.set_result(True)


async def main():
    # Get a reference to the event loop as we plan to us
    # low-level APIs.
    loop = asyncio.get_running_loop()
    on_con_lost = loop.create_future()

    # Create a pair of connected sockets
    rsock, wsock = socket.socketpair()

    # Register the socket to wait for data.
    transport, protocol = await loop.create_connection(
        lambda: MyProtocol(on_con_lost), sock=rsock)

    # Simulate the reception of data from the network.
    loop.call_soon(wsock.send, 'abc'.encode())

    try:
        await protocol.on_con_lost
    finally:
        transport.close()
        wsock.close()

asyncio.run(main())
```

Ver también

El ejemplo monitorizar eventos de lectura en un descriptor de archivo utiliza el método de bajo nivel `loop.add_reader()` para registrar un descriptor de archivo.

El ejemplo registrar un socket abierto a la espera de datos usando streams usa *streams* de alto nivel creados por la función `open_connection()` en una corrutina.

loop.subprocess_exec() y *SubprocessProtocol*

Un ejemplo de un protocolo de subproceso que se utiliza para obtener la salida de un subproceso y esperar su terminación.

El subproceso es creado por el método `loop.subprocess_exec()`

```
import asyncio
import sys

class DateProtocol(asyncio.SubprocessProtocol):
    def __init__(self, exit_future):
        self.exit_future = exit_future
        self.output = bytearray()
        self.pipe_closed = False
        self.exited = False

    def pipe_connection_lost(self, fd, exc):
        self.pipe_closed = True
        self.check_for_exit()

    def pipe_data_received(self, fd, data):
        self.output.extend(data)

    def process_exited(self):
        self.exited = True
        # process_exited() method can be called before
        # pipe_connection_lost() method: wait until both
        # called.
        self.check_for_exit()

    def check_for_exit(self):
        if self.pipe_closed and self.exited:
            self.exit_future.set_result(True)

async def get_date():
    # Get a reference to the event loop as we plan to us
    # low-level APIs.
    loop = asyncio.get_running_loop()

    code = 'import datetime as dt; print(dt.datetime.now
    exit_future = asyncio.Future(loop=loop)

    # Create the subprocess controlled by DateProtocol;
    # redirect the standard output into a pipe.
    transport, protocol = await loop.subprocess_exec(
        lambda: DateProtocol(exit_future),
        sys.executable, '-c', code,
        stdin=None, stderr=None)

    # Wait for the subprocess exit using the process_exi
    # method of the protocol.
    await exit_future

    # Close the stdout pipe.
    transport.close()

    # Read the output which was collected by the
    # pipe_data_received() method of the protocol.
    data = bytes(protocol.output)
    return data.decode('ascii').rstrip()

date = asyncio.run(get_date())
print(f"Current date: {date}")
```

Consulte también el mismo ejemplo escrito utilizando la API de alto nivel.

Políticas

Advertencia

Policies are deprecated and will be removed in Python 3.16. Users are encouraged to use the `asyncio.run()` function or the `asyncio.Runner` with *loop_factory* to use the desired loop implementation.

Una política de bucle de eventos es un objeto global que se utiliza para obtener y establecer el bucle de eventos actual o para crear nuevos bucles de eventos. La política preestablecida puede ser reemplazada con alternativas built-in para usar diferentes implementaciones de bucles de eventos, o sustituida por una política personalizada la cual puede anular estos comportamientos.

El objeto de política obtiene y establece un bucle de eventos separado por *contexto*. Por defecto, esto se realiza por hilo, aunque las políticas personalizadas pueden definir el *contexto* de una forma diferente.

Las políticas de bucle de eventos personalizadas pueden controlar el comportamiento de `get_event_loop()`, `set_event_loop()`, y `new_event_loop()`.

Los objetos de política deberían implementar las APIs definidas en la clase abstracta base `AbstractEventLoopPolicy`.

Obteniendo y configurando la política

Las siguientes funciones pueden ser usadas para obtener y configurar la política de los procesos actuales:

asyncio.get_event_loop_policy()

Retorna la política actual en todo el proceso.

Obsoleto desde la versión 3.14: The **`get_event_loop_policy()`** function is deprecated and will be removed in Python 3.16.

asyncio.set_event_loop_policy(*policy*)

Establece la política actual en todo el proceso a *policy*.

Si *policy* está configurado a `None`, la política por defecto se reestablece.

Obsoleto desde la versión 3.14: The **`set_event_loop_policy()`** function is deprecated and will be removed in Python 3.16.

Objetos de política

La clase base de política de bucle de eventos abstractos se define de la siguiente manera:

class asyncio.AbstractEventLoopPolicy

Una clase base abstracta para políticas asyncio.

get_event_loop()

Retorna el bucle de eventos para el contexto actual.

Retorna un objeto bucle de eventos implementando la interfaz **`AbstractEventLoop`**.

Este método nunca debería retornar `None`.

Distinto en la versión 3.6.

set_event_loop(*loop*)

Establece el bucle de eventos para el contexto a *loop*.

new_event_loop()

Crea y retorna un nuevo objeto de bucle de eventos.

Este método nunca debería retornar `None`.

Obsoleto desde la versión 3.14: The **`AbstractEventLoopPolicy`** class is deprecated and will be removed in Python 3.16.

asyncio se envía con las siguientes políticas integradas:

class asyncio.DefaultEventLoopPolicy

La política por defecto asyncio. Usa `SelectorEventLoop` en Unix y `ProactorEventLoop` en Windows.

No hay necesidad de instalar la política por defecto manualmente. asyncio está configurado para usar la política por defecto automáticamente.

Distinto en la versión 3.8: En Windows, `ProactorEventLoop` ahora se usa por defecto.

Distinto en la versión 3.14: The `get_event_loop()` method of the default asyncio policy now raises a `RuntimeError` if there is no set event loop.

Obsoleto desde la versión 3.14: The **`DefaultEventLoopPolicy`** class is deprecated and will be removed in Python 3.16.

class asyncio.WindowsSelectorEventLoopPolicy

Una política de bucle de eventos alternativa que usa la implementación de bucle de eventos `SelectorEventLoop`.

Availability: Windows.

Obsoleto desde la versión 3.14: The **`WindowsSelectorEventLoopPolicy`** class is deprecated and will be removed in Python 3.16.

class asyncio.WindowsProactorEventLoopPolicy

Una política de bucle de eventos alternativa que usa la implementación de bucle de eventos `ProactorEventLoop`.

Availability: Windows.

Obsoleto desde la versión 3.14: The **`WindowsProactorEventLoopPolicy`** class is deprecated and will be removed in Python 3.16.

Personalizar Políticas

Para implementar una nueva política de bucle de eventos, se recomienda heredar **`DefaultEventLoopPolicy`** y sobreescribir los métodos para los cuales se desea una conducta personalizada, por ejemplo:

```
class MyEventLoopPolicy(asyncio.DefaultEventLoopPolicy):

    def get_event_loop(self):
        """Get the event loop.

        This may be None or an instance of EventLoop.
        """
        loop = super().get_event_loop()
        # Do something with loop ...
        return loop

asyncio.set_event_loop_policy(MyEventLoopPolicy())
```

Soporte de plataforma

El módulo `asyncio` está diseñado para ser portátil, pero algunas plataformas tienen diferencias y limitaciones sutiles debido a la arquitectura y las capacidades subyacentes de las plataformas.

Todas las plataformas

- `loop.add_reader()` y `loop.add_writer()` no se pueden utilizar para supervisar la E/S del archivo.

Windows

Código fuente: Lib/asyncio/proactor_events.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/proactor_events.py], Lib/asyncio/windows_events.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/windows_events.py], Lib/asyncio/windows_utils.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/windows_utils.py]

Distinto en la versión 3.8: En Windows, `ProactorEventLoop` es ahora el bucle de eventos predeterminado.

Todos los bucles de eventos en Windows no admiten los métodos siguientes:

- `loop.create_unix_connection()` y `loop.create_unix_server()` no son compatibles. La familia de sockets `socket.AF_UNIX` es específica de Unix.
- `loop.add_signal_handler()` y `loop.remove_signal_handler()` no son compatibles.

`SelectorEventLoop` tiene las siguientes limitaciones:

- `SelectSelector` se utiliza para esperar los eventos de los sockets: soporta los sockets y está limitado a 512 sockets.

- `loop.add_reader()` y `loop.add_writer()` sólo aceptan manejadores de sockets (por ejemplo, los descriptores de archivos de tuberías no están soportados).
- Las tuberías no están soportadas, por lo que los métodos `loop.connect_read_pipe()` y `loop.connect_write_pipe()` no están implementados.
- Subprocesos no están soportados, es decir, los métodos `loop.subprocess_exec()` y `loop.subprocess_shell()` no están implementados.

`ProactorEventLoop` tiene las siguientes limitaciones:

- Los métodos `loop.add_reader()` y `loop.add_writer()` no están soportados.

La resolución del reloj monótono de Windows suele ser de unos 15,6 milisegundos. La mejor resolución es de 0,5 milisegundos. La resolución depende del hardware (disponibilidad de HPET [https://es.wikipedia.org/wiki/High_Precision_Event_Timer]) y de la configuración de Windows.

Soporte de sub-procesos en Windows

En Windows, el bucle de eventos por defecto `ProactorEventLoop` soporta subprocesos, mientras que `SelectorEventLoop` no lo hace.

macOS

Las versiones modernas de MacOS son totalmente compatibles.

macOS $<=$ 10.8

En macOS 10.6, 10.7 y 10.8, el bucle de eventos por defecto utiliza `selectors.KqueueSelector`, que no soporta dispositivos de caracteres en estas versiones. El `SelectorEventLoop` puede ser configurado manualmente para usar `SelectSelector` o `PollSelector` para soportar dispositivos de caracteres en estas versiones antiguas de macOS. Ejemplo:

```
import asyncio
import selectors

selector = selectors.SelectSelector()
loop = asyncio.SelectorEventLoop(selector)
asyncio.set_event_loop(loop)
```

Extensión

La dirección principal para extender **`asyncio`** es escribir clases *event loop* personalizadas. Asyncio tiene ayudantes que podrían usarse para simplificar esta tarea.

Nota

Los terceros deben reutilizar el código asyncio existente con precaución, una nueva versión de Python es gratuita para romper la compatibilidad con versiones anteriores en la parte *internal* de la API.

Escribir un bucle de eventos personalizado

`asyncio.AbstractEventLoop` declara muchos métodos. Implementarlos todos desde cero es un trabajo tedioso.

Un bucle puede obtener la implementación de muchos métodos comunes de forma gratuita al heredar de **`asyncio.BaseEventLoop`**.

A su vez, el sucesor debería implementar un montón de métodos *privados* declarados pero no implementados en **`asyncio.BaseEventLoop`**.

Por ejemplo, `loop.create_connection()` comprueba los argumentos, resuelve las direcciones DNS y llama a `loop._make_socket_transport()` que debería implementar la clase heredada. El método `_make_socket_transport()` no está documentado y se considera como una API *internal*.

Constructores privados Future y Task

`asyncio.Future` y `asyncio.Task` nunca deben crearse directamente; en su lugar, utilice las fábricas `loop.create_future()` y `loop.create_task()` o `asyncio.create_task()` correspondientes.

Sin embargo, *event loops* de terceros puede *reuse* incorporar futuras implementaciones y tareas con el fin de obtener un código complejo y altamente optimizado de forma gratuita.

Para ello se listan los siguientes constructores *private*:

Future._init_(*, *loop=None*)

Cree una instancia futura integrada.

loop es una instancia de bucle de eventos opcional.

Task._init_(*coro*, *, *loop=None*, *name=None*, *context=None*)

Cree una instancia de tarea integrada.

loop es una instancia de bucle de eventos opcional. El resto de argumentos se describen en la descripción de `loop.create_task()`.

Distinto en la versión 3.11: Se agrega el argumento *context*.

Soporte de por vida de tareas

La implementación de una tarea de terceros debe llamar a las siguientes funciones para mantener una tarea visible para `asyncio.all_tasks()` y `asyncio.current_task()`:

asyncio._register_task(*task*)

Registre un nuevo *task* como administrado por *asyncio*.

Llame a la función desde un constructor de tareas.

asyncio._unregister_task(*task*)

Anule el registro de un *task* de las estructuras internas de *asyncio*.

La función debe llamarse cuando una tarea está a punto de finalizar.

asyncio._enter_task(*loop*, *task*)

Cambie la tarea actual al argumento *task*.

Llame a la función justo antes de ejecutar una parte del *coroutine* incrustado (**`coroutine.send()`** o **`coroutine.throw()`**).

asyncio._leave_task(*loop*, *task*)

Vuelva a cambiar la tarea actual de *task* a `None`.

Llame a la función justo después de la ejecución de **`coroutine.send()`** o **`coroutine.throw()`**.

Índice de API de alto nivel

Esta página lista todas las APIs async/await habilitadas de alto nivel.

Tareas

Utilidades para ejecutar programas asyncio, crear Tareas, y esperar en múltiples cosas con tiempos de espera.

`run()`	Crea un loop de eventos, ejecuta una corrutina, cierra el loop.
`Runner`	Un administrador de contexto que simplifica múltiples llamadas a funciones asíncronas.
`Task`	Objeto tarea.
`TaskGroup`	Un administrador de contexto que contiene un grupo de tareas. Proporciona una forma conveniente y confiable de esperar a que finalicen todas las tareas del grupo.
`create_task()`	Inicia una tarea asyncio, luego la retorna.
`current_task()`	Retorna la tarea actual.
`all_tasks()`	Retorna todas las tareas que aún no han terminado para un bucle de eventos.
`await` `sleep()`	Duerme por un número de segundos.
`await` `gather()`	Programa y espera por cosas concurrentemente.
`await` `wait_for()`	Ejecuta con un tiempo de espera.
`await` `shield()`	Protege de la cancelación.
`await` `wait()`	Monitorea la completitud.
`timeout()`	Ejecuta con un tiempo de espera. Útil en los casos en que `wait_for` no es adecuado.
`to_thread()`	Ejecute de forma asincrónica una función en un subproceso del sistema operativo independiente.
`run_coroutine_threadsafe()`	Programa una corrutina desde otro hilo del sistema operativo.
`for in` `as_completed()`	Monitorea por completitud con un loop `for`.

Ejemplos

- Usando asyncio.gather() para ejecutar cosas en paralelo.
- Usando asyncio.wait_for() para forzar un tiempo de expiración.
- Cancelación.
- Usando asyncio.sleep().
- Ver también la página principal de documentación de Tareas.

Colas

Las colas deberían ser usadas para distribuir trabajo entre múltiples Tareas asyncio, implementar pools de conexiones y patrones pub/sub.

`Queue`	Una cola FIFO.
`PriorityQueue`	Una cola de prioridad.
`LifoQueue`	Una cola LIFO.

Ejemplos

- Usando asyncio.Queue para distribuir carga de trabajo entre varias Tareas.
- Ver también la página de documentación de Colas.

Subprocesos

Utilidades para generar subprocesos y ejecutar comandos de consola.

`await` **`create_subprocess_exec()`**	Crea un subproceso.
`await` **`create_subprocess_shell()`**	Ejecuta un comando de consola.

Ejemplos

- Ejecutando un comando de consola.
- Ver también la documentación de las APIs de subprocesos.

Flujos

APIs de alto nivel para trabajar con IO de red.

await `open_connection()`	Establece una conexión TCP.
await `open_unix_connection()`	Establece una conexión de un socket Unix.
await `start_server()`	Lanza un servidor TCP.
await `start_unix_server()`	Lanza un servidor de sockets Unix.
`StreamReader`	Objeto de alto nivel async/await para recibir datos de red.
`StreamWriter`	Objeto de alto nivel async/await para enviar datos de red.

Ejemplos

- Cliente TCP de ejemplo.
- Ver también la documentación de APIs de flujos.

Sincronización

Primitivas de sincronización al estilo hilos que pueden ser usadas en Tareas.

`Lock`	Un bloqueo mutex.
`Event`	Un objeto de evento.
`Condition`	Un objeto de condición.
`Semaphore`	Un semáforo.
`BoundedSemaphore`	Un semáforo acotado.
`Barrier`	Un objeto barrera.

Ejemplos

- Usando asyncio.Event.
- Usando asyncio.Barrier.
- Ver también la documentación de las primitivas de sincronización de asyncio.

Excepciones

`asyncio.CancelledError`	Lanzada cuando una Tarea es cancelada. Ver también `Task.cancel()`.
`asyncio.BrokenBarrierError`	Lanzada cuando se rompe una barrera. Véase también `Barrier.wait()`.

Ejemplos

- Gestionando CancelledError para ejecutar código en petición de cancelación.
- Ver también la lista completa de excepciones específicas de asyncio.

Índice de API de bajo nivel

Esta página enumera todas las APIs de asyncio de bajo nivel.

Obtención del bucle de eventos

`asyncio.get_running_loop()`	La función **preferida** para obtener el bucle de eventos en ejecución.
`asyncio.get_event_loop()`	Obtiene una instancia del bucle de eventos (en ejecución o actual mediante la política actual).
`asyncio.set_event_loop()`	Establece el bucle de eventos como actual a través de la política del bucle.
`asyncio.new_event_loop()`	Crea un nuevo bucle de eventos.

Ejemplos

- Usando asyncio.get_running_loop().

Métodos del bucle de eventos

Consulte también la sección de documentación principal sobre Event loop methods.

Ciclo de vida

`loop.run_until_complete()`	Ejecuta un Future/Tarea/aguardable (*awaitable*) hasta que se complete.
`loop.run_forever()`	Ejecuta el bucle de eventos para siempre.
`loop.stop()`	Detiene el bucle de eventos.
`loop.close()`	Cierra el bucle de eventos.
`loop.is_running()`	Retorna `True` si el bucle de eventos se está ejecutando.
`loop.is_closed()`	Retorna `True` si el bucle de eventos está cerrado.
`await loop.shutdown_asyncgens()`	Cierra generadores asincrónicos.

Depuración

`loop.set_debug()`	Habilita o deshabilita el modo de depuración.
`loop.get_debug()`	Obtiene el modo de depuración actual.

Programación de devoluciones de llamada

`loop.call_soon()`	Invoca una devolución de llamada *soon*.
`loop.call_soon_threadsafe()`	Una variante segura para subprocesos de `loop.call_soon()`.
`loop.call_later()`	Invoca una devolución de llamada *después* del tiempo especificado.
`loop.call_at()`	Invoca una devolución de llamada *en* el tiempo especificado.

Thread/Interpreter/Process Pool

`await loop.run_in_executor()`	Ejecuta una función de bloqueo vinculada a la CPU o de otro tipo en un ejecutor `concurrent.futures`.
`loop.set_default_executor()`	Establece el ejecutor predeterminado para `loop.run_in_executor()`.

Tareas y Futures

`loop.create_future()`	Crea un objeto `Future`.
`loop.create_task()`	Programa una corrutina como `Task`.
`loop.set_task_factory()`	Establece una fábrica utilizada por `loop.create_task()` para crear `Tareas`.
`loop.get_task_factory()`	Obtiene la fábrica `loop.create_task()` que se usa para crear `Tareas`.

DNS

`await loop.getaddrinfo()`	Versión asincrónica de `socket.getaddrinfo()`.
`await loop.getnameinfo()`	Versión asincrónica de `socket.getnameinfo()`.

Redes e IPC

`loop.create_connection()`	Abre una conexión TCP.
`loop.create_server()`	Crea un servidor TCP.
`loop.create_unix_connection()`	Abre una conexión de socket Unix.
`loop.create_unix_server()`	Crea un servidor socket de Unix.
`loop.connect_accepted_socket()`	Envuelve un `socket` en un par `(transport, protocol)`.
`loop.create_datagram_endpoint()`	Abre una conexión de datagramas (UDP).
`loop.sendfile()`	Envía un archivo a través del transporte.
`loop.start_tls()`	Actualiza una conexión existente a TLS.
`loop.connect_read_pipe()`	Envuelve el fin de lectura de *pipe* en un par `(transport, protocol)`.
`loop.connect_write_pipe()`	Envuelve el fin de escritura de *pipe* en un par `(transport, protocol)`.

Sockets

`loop.sock_recv()`	Recibe datos de `socket`.
`loop.sock_recv_into()`	Recibe datos de `socket` en un buffer.
`loop.sock_recvfrom()`	Recibe un datagrama desde `socket`.
`loop.sock_recvfrom_into()`	Recibe un datagrama desde `socket` en un buffer.
`loop.sock_sendall()`	Envía datos a `socket`.
`loop.sock_sendto()`	Envía un datagrama a través de `socket` a la dirección indicada.
`loop.sock_connect()`	Conecta con `socket`.
`loop.sock_accept()`	Acepta una conexión `socket`.
`loop.sock_sendfile()`	Envía un archivo a través de `socket`.
`loop.add_reader()`	Comienza a monitorear un descriptor de archivo para ver la disponibilidad de lectura.
`loop.remove_reader()`	Detiene el monitoreo del descriptor de archivo para ver la disponibilidad de lectura.
`loop.add_writer()`	Comienza a monitorear un descriptor de archivo para ver la disponibilidad de escritura.
`loop.remove_writer()`	Detiene el monitoreo del descriptor de archivo para ver la disponibilidad de escritura.

Señales Unix

`loop.add_signal_handler()`	Añade un gestor para `signal`.

`loop.remove_signal_handler()`	Elimina un gestor para .

Subprocesos

`loop.subprocess_exec()`	Genera un subproceso.
`loop.subprocess_shell()`	Genera un subproceso desde un comando de shell.

Gestor de errores

`loop.call_exception_handler()`	Invoca al gestor de excepciones.
`loop.set_exception_handler()`	Establece un nuevo gestor de excepciones.
`loop.get_exception_handler()`	Obtiene el gestor de excepciones actual.
`loop.default_exception_handler()`	La implementación predetermina del gestor de excepciones.

Ejemplos

- Usando asyncio.new_event_loop() y loop.run_forever().
- Usando loop.call_later().
- Usando `loop.create_connection()` para implementar un cliente de eco.
- Usando `loop.create_connection()` para conectar a un socket.
- Usando add_reader() para mirar un FD y leer eventos.
- Usando loop.add_signal_handler().
- Usando loop.subprocess_exec().

Transportes

Todos los transportes implementan los siguientes métodos:

`transport.close()`	Cierra el transporte.
`transport.is_closing()`	Retorna `True` si el transporte está cerrado o se está cerrando.
`transport.get_extra_info()`	Solicita información sobre el transporte.
`transport.set_protocol()`	Establece un nuevo protocolo.
`transport.get_protocol()`	Retorna el protocolo actual.

Transportes que pueden recibir datos (conexiones TCP y Unix, *pipes*, etc). Retornan de métodos como `loop.create_connection()`, `loop.create_unix_connection()`, **`loop.connect_read()`**, etc:

Leer transportes

`transport.is_reading()`	Retorna `True` si el transporte está recibiendo.
`transport.pause_reading()`	Pausa la recepción.
`transport.resume_reading()`	Reanuda la recepción.

Transportes que pueden enviar datos (conexiones TCP y Unix, *pipes*, etc). Retornan de métodos como `loop.create_connection()`, `loop.create_unix_connection()`, `loop.connect_write_pipe()`, etc:

Escribir transportes

`transport.write()`	Escribe datos en el transporte.
`transport.writelines()`	Escribe búferes en el transporte.
`transport.can_write_eof()`	Retorna `True` si el transporte admite el envío de EOF.
`transport.write_eof()`	Cierra y envía EOF después de vaciar los datos almacenados en búfer.
`transport.abort()`	Cierra el transporte inmediatamente.
`transport.get_write_buffer_size()`	Retorna el tamaño actual del búfer de salida.
`transport.get_write_buffer_limits()`	Retorna los límites superior e inferior para controlar el flujo de escritura.
`transport.set_write_buffer_limits()`	Establece nuevos límites superior e inferior para el control del flujo de escritura.

Transportes retornados por `loop.create_datagram_endpoint()`:

Transportes de datagramas

`transport.sendto()`	Envía datos al par remoto.
`transport.abort()`	Cierra el transporte inmediatamente.

Abstracción de transporte de bajo nivel sobre subprocesos. Retornado por `loop.subprocess_exec()` y `loop.subprocess_shell()`:

Transportes de subprocesos

`transport.get_pid()`	Retorna el id de proceso del subproceso.
`transport.get_pipe_transport()`	Retorna el transporte para la *pipe* de comunicación solicitada (*stdin*, *stdout* o *stderr*).
`transport.get_returncode()`	Retorna el código de retorno del subproceso.
`transport.kill()`	Mata el subproceso.
`transport.send_signal()`	Envía una señal al subproceso.
`transport.terminate()`	Detiene el subproceso.
`transport.close()`	Mata el subproceso y cierra todas las *pipes*.

Protocolos

Las clases de protocolo pueden implementar los siguientes **métodos de devolución de llamada**:

callback `connection_made()`	Se llama cuando se establece una conexión.
callback `connection_lost()`	Se llama cuando la conexión se pierde o cierra.
callback `pause_writing()`	Se llama cuando el búfer del transporte excede el límite superior.
callback `resume_writing()`	Se llama cuando el búfer del transporte se vacía por debajo del límite inferior.

Protocolos de streaming (TCP, Unix Sockets, Pipes)

callback `data_received()`	Se llama cuando se reciben algunos datos.
callback `eof_received()`	Se llama cuando se recibe un EOF.

Protocolos de streaming en búfer

callback `get_buffer()`	Se llama para asignar un nuevo búfer de recepción.
callback `buffer_updated()`	Se llama cuando el búfer se actualizó con los datos recibidos.
callback `eof_received()`	Se llama cuando se recibe un EOF.

Protocolos de datagramas

callback `datagram_received()`	Se llama cuando se recibe un datagrama.
callback `error_received()`	Se llama cuando una operación de envío o recepción anterior genera un `OSError`.

Protocolos de subprocesos

callback `pipe_data_received()`	Se llama cuando el proceso hijo escribe datos en su *pipe stdout* o *stderr*.
callback `pipe_connection_lost()`	Se llama cuando se cierra un *pipe* que se comunica con el proceso hijo.
callback `process_exited()`	Called when the child process has exited. It can be called before `pipe_data_received()` and `pipe_connection_lost()` methods.

Políticas de bucle de eventos

Las políticas son un mecanismo de bajo nivel para alterar el comportamiento de funciones como `asyncio.get_event_loop()`. Vea también la sección principal políticas para más detalles.

Acceso a políticas

`asyncio.get_event_loop_policy()`	Retorna la política actual en todo el proceso.
`asyncio.set_event_loop_policy()`	Establece una nueva política para todo el proceso.
`AbstractEventLoopPolicy`	Clase base para objetos de política.

Desarrollando con asyncio

La programación asincrónica es diferente a la programación «secuencial» clásica.

Esta página enumera errores y trampas comunes y explica cómo evitarlos.

Modo depuración

Por defecto asyncio se ejecuta en modo producción. Para facilitar el desarrollo asyncio tiene un *modo depuración*.

Hay varias maneras de habilitar el modo depuración de asyncio:

- Definiendo la variable de entorno **`PYTHONASYNCIODEBUG`** a `1`.
- Usando Modo de Desarrollo de Python.
- Pasando `debug=True` a **`asyncio.run()`**.
- Invocando **`loop.set_debug()`**.

Además de habilitar el modo depuración, considere también:

- definir el nivel de log del asyncio logger a **`logging.DEBUG`**, por ejemplo el siguiente fragmento de código puede ser ejecutado al inicio de la aplicación:

  ```
  logging.basicConfig(level=logging.DEBUG)
  ```

- configurando el módulo **`warnings`** para mostrar advertencias **`ResourceWarning`**. Una forma de hacerlo es usando la opción **`-W`** `default` de la línea de comandos.

Cuando el modo depuración está habilitado:

- Muchas APIs asyncio que no son seguras para hilos (como los métodos **`loop.call_soon()`** y **`loop.call_at()`**)

generan una excepción si son llamados desde un hilo equivocado.
- El tiempo de ejecución del selector E/S es registrado si tarda demasiado tiempo en realizar una operación E/S.
- Los callbacks que tardan más de 100ms son registrados. El atributo **`loop.slow_callback_duration`** puede ser usado para definir la duración mínima de ejecución en segundos que se considere como «lenta».

Concurrencia y multihilo

Un bucle de eventos se ejecuta en un hilo (generalmente el hilo principal) y ejecuta todos los callbacks y las Tareas en su hilo. Mientras una Tarea está ejecutándose en el bucle de eventos, ninguna otra Tarea puede ejecutarse en el mismo hilo. Cuando una Tarea ejecuta una expresión `await`, la Tarea en ejecución se suspende y el bucle de eventos ejecuta la siguiente Tarea.

Para programar un callback desde otro hilo del SO, se debe usar el método **`loop.call_soon_threadsafe()`**. Ejemplo:

```
loop.call_soon_threadsafe(callback, *args)
```

Casi ningún objeto asyncio es seguro entre hilos (*thread safe*), lo cual generalmente no es un problema a no ser que haya código que trabaje con ellos desde fuera de una Tarea o un callback. Si tal código necesita llamar a la API de asyncio de bajo nivel, se debe usar el método **`loop.call_soon_threadsafe()`**, por ejemplo:

```
loop.call_soon_threadsafe(fut.cancel)
```

Para programar un objeto de corrutina desde una hilo diferente del sistema operativo se debe usar la función **`run_coroutine_threadsafe()`**. Esta retorna un **`concurrent.futures.Future`** para acceder al resultado:

```
async def coro_func():
     return await asyncio.sleep(1, 42)

# Later in another OS thread:
```

```
future = asyncio.run_coroutine_threadsafe(coro_func(), l
# Wait for the result:
result = future.result()
```

Para manejar señales el bucle de eventos debe ser ejecutado en el hilo principal.

The `loop.run_in_executor()` method can be used with a `concurrent.futures.ThreadPoolExecutor` or `InterpreterPoolExecutor` to execute blocking code in a different OS thread without blocking the OS thread that the event loop runs in.

Actualmente no hay forma de programar corrutinas o retrollamadas directamente desde un proceso diferente (como uno que haya comenzado con `multiprocessing`). La sección Event loop methods enumera las API que pueden leer desde tuberías y descriptores de archivos sin bloquear el bucle de eventos. Además, las API Subprocess de asyncio proporcionan una forma de iniciar un proceso y comunicarse con él desde el bucle de eventos. Por último, el método `loop.run_in_executor()` mencionado anteriormente también se puede usar con un `concurrent.futures.ProcessPoolExecutor` para ejecutar código en un proceso diferente.

Ejecutando código bloqueante

Código bloqueante (dependiente de la CPU) no debe ser ejecutado directamente. Por ejemplo, si una función realiza un cálculo intensivo de CPU durante 1 segundo, todas las Tareas y operaciones de Entrada/Salida (IO) concurrentes se retrasarían 1 segundo.

An executor can be used to run a task in a different thread, including in a different interpreter, or even in a different process to avoid blocking the OS thread with the event loop. See the `loop.run_in_executor()` method for more details.

Logueando

asyncio usa el módulo `logging` y todo el logueo es realizado mediante el logger `"asyncio"`.

El nivel de log por defecto es `logging.INFO`, el cual puede ser fácilmente ajustado:

```
logging.getLogger("asyncio").setLevel(logging.WARNING)
```

Loguear por la red puede bloquear el bucle de eventos. Se recomienda usar un subproceso separado para manejar registros o usar sin bloqueo IO. Por ejemplo, consulte Tratar con gestores que bloquean.

Detectar corrutinas no esperadas

Cuando una función de corrutina es invocada, pero no esperada (por ejemplo `coro()` en lugar de `await coro()`) o la corrutina no es programada con `asyncio.create_task()`, asyncio emitirá una `RuntimeWarning`:

```
import asyncio

async def test():
    print("never scheduled")

async def main():
    test()

asyncio.run(main())
```

Salida:

```
test.py:7: RuntimeWarning: coroutine 'test' was never aw
  test()
```

Salida en modo depuración:

```
test.py:7: RuntimeWarning: coroutine 'test' was never aw
Coroutine created at (most recent call last)
  File "../t.py", line 9, in <module>
```

```
    asyncio.run(main(), debug=True)

  < .. >

  File "../t.py", line 7, in main
    test()
  test()
```

La solución habitual es esperar la corrutina o llamar a la función `asyncio.create_task()`:

```
async def main():
    await test()
```

Detectar excepciones nunca recuperadas

Si un `Future.set_exception()` es invocado pero el objeto Futuro nunca es esperado, la excepción nunca será propagada al código del usuario. En este caso, asyncio emitiría un mensaje de registro cuando el objeto Futuro fuera recolectado como basura.

Ejemplo de una excepción no manejada:

```
import asyncio

async def bug():
    raise Exception("not consumed")

async def main():
    asyncio.create_task(bug())

asyncio.run(main())
```

Salida:

```
Task exception was never retrieved
future: <Task finished coro=<bug() done, defined at test
  exception=Exception('not consumed')>
```

```
Traceback (most recent call last):
  File "test.py", line 4, in bug
    raise Exception("not consumed")
Exception: not consumed
```

Habilita el modo depuración para obtener el seguimiento de pila (*traceback*) donde la tarea fue creada:

```
asyncio.run(main(), debug=True)
```

Salida en modo depuración:

```
Task exception was never retrieved
future: <Task finished coro=<bug() done, defined at test
    exception=Exception('not consumed') created at async

source_traceback: Object created at (most recent call la
  File "../t.py", line 9, in <module>
    asyncio.run(main(), debug=True)

< .. >

Traceback (most recent call last):
  File "../t.py", line 4, in bug
    raise Exception("not consumed")
Exception: not consumed
```

Asynchronous generators best practices

Writing correct and efficient asyncio code requires awareness of certain pitfalls. This section outlines essential best practices that can save you hours of debugging.

Close asynchronous generators explicitly

It is recommended to manually close the asynchronous generator. If a generator exits early - for example, due to an exception raised in the body of an `async for` loop - its asynchronous cleanup code may run in an unexpected context. This can occur after the tasks it depends on have completed, or during the event loop shutdown when the async-generator's garbage collection hook is called.

To avoid this, explicitly close the generator by calling its **`aclose()`** method, or use the **`contextlib.aclosing()`** context manager:

```
import asyncio
import contextlib

async def gen():
  yield 1
  yield 2

async def func():
  async with contextlib.aclosing(gen()) as g:
    async for x in g:
      break  # Don't iterate until the end

asyncio.run(func())
```

As noted above, the cleanup code for these asynchronous generators is deferred. The following example demonstrates that the finalization of an asynchronous generator can occur in an unexpected order:

```
import asyncio
work_done = False

async def cursor():
    try:
        yield 1
    finally:
        assert work_done

async def rows():
    global work_done
    try:
        yield 2
    finally:
        await asyncio.sleep(0.1) # immitate some async w


async def main():
    async for c in cursor():
        async for r in rows():
            break
        break

asyncio.run(main())
```

For this example, we get the following output:

```
unhandled exception during asyncio.run() shutdown
task: <Task finished name='Task-3' coro=<<async_generato
Traceback (most recent call last):
  File "example.py", line 6, in cursor
    yield 1
asyncio.exceptions.CancelledError

During handling of the above exception, another exceptio

Traceback (most recent call last):
  File "example.py", line 8, in cursor
    assert work_done
           ^^^^^^^^^
AssertionError
```

The `cursor()` asynchronous generator was finalized before the `rows` generator - an unexpected behavior.

The example can be fixed by explicitly closing the `cursor` and `rows` async-generators:

```
async def main():
    async with contextlib.aclosing(cursor()) as cursor_g
        async for c in cursor_gen:
            async with contextlib.aclosing(rows()) as ro
```

```
                async for r in rows_gen:
                    break
            break
```

Create asynchronous generators only when the event loop is running

It is recommended to create asynchronous generators only after the event loop has been created.

To ensure that asynchronous generators close reliably, the event loop uses the `sys.set_asyncgen_hooks()` function to register callback functions. These callbacks update the list of running asynchronous generators to keep it in a consistent state.

When the `loop.shutdown_asyncgens()` function is called, the running generators are stopped gracefully and the list is cleared.

The asynchronous generator invokes the corresponding system hook during its first iteration. At the same time, the generator records that the hook has been called and does not call it again.

Therefore, if iteration begins before the event loop is created, the event loop will not be able to add the generator to its list of active generators because the hooks are set after the generator attempts to call them. Consequently, the event loop will not be able to terminate the generator if necessary.

Consider the following example:

```
import asyncio

async def agenfn():
    try:
        yield 10
    finally:
        await asyncio.sleep(0)


with asyncio.Runner() as runner:
    agen = agenfn()
    print(runner.run(anext(agen)))
    del agen
```

Salida:

```
10
Exception ignored while closing generator <async_generat
Traceback (most recent call last):
  File "example.py", line 13, in <module>
    del agen
        ^^^^
RuntimeError: async generator ignored GeneratorExit
```

This example can be fixed as follows:

```
import asyncio

async def agenfn():
    try:
        yield 10
    finally:
        await asyncio.sleep(0)

async def main():
    agen = agenfn()
    print(await anext(agen))
    del agen

asyncio.run(main())
```

Avoid concurrent iteration and closure of the same generator

Async generators may be reentered while another `__anext__()` / `athrow()` / `aclose()` call is in progress. This may lead to an inconsistent state of the async generator and can cause errors.

Let’s consider the following example:

```
import asyncio

async def consumer():
    for idx in range(100):
        await asyncio.sleep(0)
        message = yield idx
        print('received', message)

async def amain():
    agenerator = consumer()
    await agenerator.asend(None)

    fa = asyncio.create_task(agenerator.asend('A'))
    fb = asyncio.create_task(agenerator.asend('B'))
    await fa
    await fb

asyncio.run(amain())
```

Salida:

```
received A
Traceback (most recent call last):
  File "test.py", line 38, in <module>
    asyncio.run(amain())
    ~~~~~~~~~~~^^^^^^^^^
  File "Lib/asyncio/runners.py", line 204, in run
    return runner.run(main)
           ~~~~~~~~~~^^^^^^
  File "Lib/asyncio/runners.py", line 127, in run
    return self._loop.run_until_complete(task)
           ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^
  File "Lib/asyncio/base_events.py", line 719, in run_un
    return future.result()
           ~~~~~~~~~~~~~^^
  File "test.py", line 36, in amain
    await fb
RuntimeError: anext(): asynchronous generator is already
```

Therefore, it is recommended to avoid using asynchronous generators in parallel tasks or across multiple event loops.

`socket` — Low-level networking interface

Código fuente: Lib/socket.py [https://github.com/python/cpython/tree/3.14/Lib/socket.py]

Este módulo proporciona acceso a la interfaz BSD *socket*. Está disponible en todos los sistemas Unix modernos, Windows, MacOS y probablemente plataformas adicionales.

Nota

Algunos comportamientos pueden depender de la plataforma, ya que las llamadas se realizan a las API de socket del sistema operativo.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

The Python interface is a straightforward transliteration of the Unix system call and library interface for sockets to Python's object-oriented style: the `socket()` function returns a *socket object* whose methods implement the various socket system calls. Parameter types are somewhat higher-level than in the C interface: as with **`read()`** and **`write()`** operations on Python files, buffer allocation on receive operations is automatic, and buffer length is implicit on send operations.

Ver también

Módulo `socketserver`
Clases que simplifican la escritura de servidores de red.

Módulo `ssl`
Un contenedor TLS/SSL para objetos de socket.

Familias Socket

Dependiendo del sistema y de las opciones de compilación, este módulo admite varias familias de sockets.

El formato de dirección requerido por un objeto de socket determinado se selecciona automáticamente en función de la familia de direcciones especificada cuando se creó el objeto de socket. Las direcciones de socket se representan de la siguiente manera:

- La dirección de un socket `AF_UNIX` enlazado a un nodo del sistema de archivos es representado como una cadena de caracteres, utilizando la codificación del sistema de archivos y el controlador de errores `'surrogateescape'` (Observar **PEP 383** [https://peps.python.org/pep-0383/]). Una dirección en el espacio de nombre abstracto de Linux es devuelvo como un bytes-like object con un byte inicial nulo; tenga en cuenta que los sockets en este nombre de espacio puede comunicarse con sockets normales del sistema de archivos, así que los programas destinados a correr en Linux podrían necesitar tratar con ambos tipos de direcciones. Se puede pasar un objeto similar a una cadena de caracteres o bytes para cualquier tipo de dirección al pasarlo como argumento.

  Distinto en la versión 3.3: Anteriormente, se suponía que las rutas de socket `AF_UNIX` utilizaban codificación UTF-8.

  Distinto en la versión 3.5: Ahora se acepta la grabación bytes-like object.

- Se utiliza un par `(host, port)` para la familia de direcciones `AF_INET`, donde *host* es una cadena que representa un nombre de host en notación de dominio de Internet como `'daring.cwi.nl'` o una dirección IPv4 como `'100.50.200.5'`, y *port* es un número entero.

  - Para direcciones IPv4, se aceptan dos formas especiales en lugar de una dirección de host: `''` representa **INADDR_ANY**, que se utiliza para enlazar a todas las interfaces, y la cadena de caracteres `'<broadcast>'` representa **INADDR_BROADCAST**. Este comportamiento no es compatible con IPv6, por lo tanto, es posible que desee evitarlos sí tiene la intención de admitir IPv6 con sus programas Python.

- For **AF_INET6** address family, a four-tuple `(host, port, flowinfo, scope_id)` is used, where *flowinfo* and *scope_id* represent the `sin6_flowinfo` and `sin6_scope_id` members in **struct sockaddr_in6** in C. For **socket** module methods, *flowinfo* and *scope_id* can be omitted just for backward compatibility. Note, however, omission of *scope_id* can cause problems in manipulating scoped IPv6 addresses.

  Distinto en la versión 3.7: Para direcciones de multidifusión (con *scopeid* significativo) *address* puede no contener la parte `%scope` (o `zone id`). Esta información es superflua y puede omitirse de forma segura (recomendado).

- **AF_NETLINK** sockets se representan como pares `(pid, groups)`.

- La compatibilidad con LINUX solo para TIPC está disponible mediante la familia de direcciones **AF_TIPC**. TIPC es un protocolo en red abierto y no basado en IP diseñado para su uso en entornos informáticos agrupados. Las direcciones se representan mediante una tupla y los campos dependen del tipo de dirección. El formulario de tupla general es `(addr_type, v1, v2, v3 [, scope])`, donde:

  - *addr_type* es uno de **TIPC_ADDR_NAMESEQ**, **TIPC_ADDR_NAME**, o **TIPC_ADDR_ID**.

  - *scope* es una de **TIPC_ZONE_SCOPE**, **TIPC_CLUSTER_SCOPE**, y **TIPC_NODE_SCOPE**.

  - Si *addr_type* es **TIPC_ADDR_NAME**, entonces *v1* es el tipo de servidor, *v2* es el identificador de puerto, y *v3* debe ser 0.

Si *addr_type* es **TIPC_ADDR_NAMESEQ**, entonces *v1* es el tipo de servidor, *v2* es el numero de puerto inferior, y *v3* es el numero de puerto superior.

Si *addr_type* es **TIPC_ADDR_ID**, *v1* es el nodo, *v2* es la referencia y *v3* debe establecerse en 0.

- Una tupla `(interface, )` es usada para la dirección de familia **AF_CAN**, donde *interface* es una cadena de caracteres representando a un nombre de interfaz de red como `'can0'`. La interfaz de red llamada `''` puede ser usada para recibir paquetes de todas las interfaces de red de esta familia.
    - **CAN_ISOTP** protocol requires a tuple `(interface, rx_addr, tx_addr)` where both additional parameters are unsigned long integer that represent a CAN identifier (standard or extended).
    - **CAN_J1939** protocol requires a tuple `(interface, name, pgn, addr)` where additional parameters are 64-bit unsigned integer representing the ECU name, a 32-bit unsigned integer representing the Parameter Group Number (PGN), and an 8-bit integer representing the address.

- Se utiliza una cadena o una tupla `(id, unit)` para el protocolo **SYSPROTO_CONTROL** de la familia **PF_SYSTEM**. La cadena es el nombre de un control de kernel mediante un ID asignado dinámicamente. La tupla se puede utilizar si se conoce el ID y el número de unidad del control del kernel o si se utiliza un ID registrado.

  Added in version 3.3.

- **AF_BLUETOOTH** admite los siguientes protocolos y formatos de dirección:
    - **BTPROTO_L2CAP** accepts a tuple `(bdaddr, psm[, cid[, bdaddr_type]])` where:

    - `bdaddr` is a string specifying the Bluetooth address.
    - `psm` is an integer specifying the Protocol/Service Multiplexer.
    - `cid` is an optional integer specifying the Channel Identifier. If not given, defaults to zero.
    - `bdaddr_type` is an optional integer specifying the address type; one of **BDADDR_BREDR** (default), **BDADDR_LE_PUBLIC**, **BDADDR_LE_RANDOM**.

  Distinto en la versión 3.14: Added `cid` and `bdaddr_type` fields.

- **BTPROTO_RFCOMM** acepta `(bdaddr, channel)` donde `bdaddr` es la dirección Bluetooth como una cadena de caracteres y `channel` es un entero.

- **BTPROTO_HCI** accepts a format that depends on your OS.

    - On Linux it accepts an integer `device_id` or a tuple `(device_id, [channel])` where `device_id` specifies the number of the Bluetooth device, and `channel` is an optional integer specifying the HCI channel (**HCI_CHANNEL_RAW** by default).
    - On FreeBSD, NetBSD and DragonFly BSD it accepts `bdaddr` where `bdaddr` is the Bluetooth address as a string.

  Distinto en la versión 3.2: Se ha añadido compatibilidad con NetBSD y DragonFlyBSD.

  Distinto en la versión 3.13.3: FreeBSD support added.

  Distinto en la versión 3.14: Added `channel` field. `device_id` not packed in a tuple is now accepted.

- **BTPROTO_SCO** accepts `bdaddr` where `bdaddr` is the Bluetooth address as a string or a **bytes** object. (ex.

`'12:23:34:45:56:67'` or `b'12:23:34:45:56:67'`)

Distinto en la versión 3.14: FreeBSD support added.

- **`AF_ALG`** es una interfaz basada en socket solo Linux para la criptografía del núcleo. Un socket de algoritmo se configura con una tupla de dos a cuatro elementos `(type, name [, feat [, mask]])`, donde:

  - *type* es el tipo de algoritmos como cadenas de caracteres, e.g. `aead`, `hash`, `skcipher` o `rng`.
  - *name* es el nombre del algoritmo y el modo de operación como cadena de caracteres, e.g. `sha256`, `hmac(sha256)`, `cbc(aes)` o `drbg_nopr_ctr_aes256`.
  - *feat* y *mask* son enteros de 32 bits sin signo.

  Availability: Linux >= 2.6.38.

  Algunos tipos de algoritmos requieren Kernels mas recientes.

  Added in version 3.6.

- **`AF_VSOCK`** permite comunicación entre maquinas virtuales y sus hosts. Los sockets están representando como una tupla `(CID, port)` donde el contexto del ID o CID y el puerto son enteros.

  Availability: Linux >= 3.9

  Véase *vsock(7)* *[https://manpages.debian.org/vsock(7)]*

  Added in version 3.7.

- **`AF_PACKET`** is a low-level interface directly to network devices. The addresses are represented by the tuple `(ifname, proto[, pkttype[, hatype[, addr]]])` where:

  - *ifname* - Cadena que especifica el nombre del dispositivo.

  - *proto* - The Ethernet protocol number. May be **`ETH_P_ALL`** to capture all protocols, one of the ETHERTYPE_* constants or any other Ethernet protocol number.
  - *pkttype* - Entero opcional especificando el tipo de paquete:
    - `PACKET_HOST` (por defecto) - Paquetes diseccionado al local host.
    - `PACKET_BROADCAST` - Paquete de transmisión de la capa física.
    - `PACKET_MULTICAST` - Paquete enviado a una dirección de multidifusión de capa física.
    - `PACKET_OTHERHOST` - Paquete a otro host que haya sido capturado por un controlador de dispositivo en modo promiscuo.
    - `PACKET_OUTGOING` - Paquete originalmente desde el local host que se enlaza de nuevo a un conector de paquetes.
  - *hatype* - Entero opcional que especifica el tipo de dirección de hardware ARP.
  - *addr* - Objeto opcional en forma de bytes que especifica la dirección física del hardware, cuya interpretación depende del dispositivo.

  Availability: Linux > = 2.2.

- **`AF_QIPCRTR`** es una interfaz basada en sockets solo para Linux para comunicarse con servicios que se ejecutan en co-procesadores en plataformas Qualcomm. La familia de direcciones se representa como una tupla `(node, port)` donde el *node* y *port* son enteros no negativos.

  Availability: Linux > = 4.7.

  Added in version 3.8.

- **`IPPROTO_UDPLITE`** es una variante de UDO que te permite especificar que porción del paquete es cubierta con la suma de comprobación. Esto agrega dos opciones al socket que pueden cambiar. `self.setsockopt(IPPROTO_UDPLITE, UDPLITE_SEND_CSCOV, length)` cambiara que parte de los paquetes salientes están cubierta por la suma de comprobación y `self.setsockopt(IPPROTO_UDPLITE, UDPLITE_RECV_CSCOV, length)` filtrara los paquetes que permitirá cubrir una pequeña parte de tu datos. En ambos casos `length` deben estar en `range(8, 2**16, 8)`.

Tal socket debe construirse como `socket(AF_INET, SOCK_DGRAM, IPPROTO_UDPLITE)` para IPV4 o `socket(AF_INET6, SOCK_DGRAM, IPPROTO_UDPLITE)` para IPV6.

Availability: Linux > = 2.6.20, FreeBSD > = 10.1

Added in version 3.9.

- **`AF_HYPERV`** es una interfaz basada en sockets exclusiva de Windows para comunicarse con hosts e invitados de Hyper-V. La familia de direcciones se representa como una tupla `(vm_id, service_id)` donde `vm_id` y `service_id` son cadenas UUID.

  El `vm_id` es el identificador de la máquina virtual o un conjunto de VMID conocidos valores si el objetivo no es una máquina virtual específica. Constantes VMID conocidas definidos en `socket` son:

  ○ `HV_GUID_ZERO`
  ○ `HV_GUID_BROADCAST`
  ○ `HV_GUID_WILDCARD` - Se utiliza para vincularse a sí mismo y aceptar conexiones de todas las particiones.
  ○ `HV_GUID_CHILDREN` - Se utiliza para vincularse a sí mismo y aceptar conexiones desde particiones secundarias.
  ○ `HV_GUID_LOOPBACK` - Se utiliza como objetivo para sí mismo.
  ○ `HV_GUID_PARENT` - Cuando se usa como enlace, acepta la conexión del padre partición. Cuando se utiliza como dirección de destino, se conectará al padre dividir.

  El `service_id` es el identificador del servicio registrado.

Added in version 3.12.

Si utiliza un nombre de host en la parte *host* de la dirección de socket IPv4/v6, el programa puede mostrar un comportamiento no determinista, ya que Python utiliza la primera dirección devuelta por la resolución DNS. La dirección del socket se resolverá de manera diferente en una dirección IPv4/v6 real, dependiendo de los resultados de la resolución DNS y/o la configuración del host. Para un comportamiento determinista, utilice una dirección numérica en la parte *host*.

Todos los errores generan excepciones. Las excepciones normales para tipos de argumentos inválidos y condiciones de falta de memoria pueden ser lanzadas. Los errores relacionados con la semántica de los sockets o de las direcciones lanzan `OSError` o una de sus subclases.

El modo de no bloqueo es compatible a través de `setblocking()`. Se admite una generalización de esto basada en los tiempos de espera a través de `settimeout()`.

Contenido del módulo

The module `socket` exports the following elements.

Excepciones

exception socket.error

Un alias en desuso de `OSError`.

Distinto en la versión 3.3: Siguiendo **PEP 3151** [https://peps.python.org/pep-3151/], es clase fue creada como un alias de `OSError`.

exception socket.herror

Una subclase de `OSError`, esta excepción se produce para los errores relacionados con la dirección, es decir, para las funciones que utilizan *h_errno* en la API de POSIX C, incluidas `gethostbyname_ex()` y `gethostbyaddr()`. El valor adjunto es un par `(h_errno, string)` que representa un error devuelto por una llamada a la biblioteca. *h_errno* es un valor numérico, mientras que *string* representa la descripción de *h_errno*, devuelta por la función **`hstrerror()`** C.

Distinto en la versión 3.3: Esta clase fue creada como una subclase de **`OSError`**.

exception socket.gaierror

A subclass of **`OSError`**, this exception is raised for address-related errors by **`getaddrinfo()`** and **`getnameinfo()`**. The accompanying value is a pair `(error, string)` representing an error returned by a library call. *string* represents the description of *error*, as returned by the **`gai_strerror()`** C function. The numeric *error* value will match one of the **`EAI_*`** constants defined in this module.

Distinto en la versión 3.3: Esta clase fue creada como una subclase de **`OSError`**.

exception socket.timeout

Un alias obsoleto de **`TimeoutError`**.

Una subclase de **`OSError`**, esta excepción se genera cuando ocurre un *timeout* en un socket que ha tenido tiempos de espera habilitados a través de una llamada previa a **`settimeout()`** (o implícitamente mediante **`setdefaulttimeout()`**). El valor del acompañamiento es una cadena de caracteres cuyo valor es actualmente siempre "tiempo de espera".

Distinto en la versión 3.3: Esta clase fue creada como una subclase de **`OSError`**.

Distinto en la versión 3.10: Esta clase se convirtió en un alias de **`TimeoutError`**.

Constantes

Las constantes AF_* y SOCK_* ahora son colecciones:

`AddressFamily` y `SocketKind` `IntEnum`.

Added in version 3.4.

socket.AF_UNIX
socket.AF_INET
socket.AF_INET6

These constants represent the address (and protocol) families, used for the first argument to `socket()`. If the `AF_UNIX` constant is not defined then this protocol is unsupported. More constants may be available depending on the system.

socket.AF_UNSPEC

`AF_UNSPEC` means that `getaddrinfo()` should return socket addresses for any address family (either IPv4, IPv6, or any other) that can be used.

socket.SOCK_STREAM
socket.SOCK_DGRAM
socket.SOCK_RAW
socket.SOCK_RDM
socket.SOCK_SEQPACKET

These constants represent the socket types, used for the second argument to `socket()`. More constants may be available depending on the system. (Only `SOCK_STREAM` and `SOCK_DGRAM` appear to be generally useful.)

socket.SOCK_CLOEXEC
socket.SOCK_NONBLOCK

Estas dos constantes, si se definen, se pueden combinar con los tipos de socket y le permiten establecer algunas banderas atómicamente (evitando así posibles condiciones de carrera y la necesidad de llamadas separadas).

Ver también

Secure File Descriptor Handling [http://udrepper.livejournal.com/20407.html] para una explicación más completa.

Availability: Linux > = 2.6.27.

Added in version 3.2.

SO_*
socket.SOMAXCONN
MSG_*
SOL_*
SCM_*
IPPROTO_*
IPPORT_*
INADDR_*
IP_*
IPV6_*
EAI_*
AI_*
NI_*
TCP_*

Many constants of these forms, documented in the Unix documentation on sockets and/or the IP protocol, are also defined in the socket module. They are generally used in arguments to the **`setsockopt()`** and **`getsockopt()`** methods of socket objects. In most cases, only those symbols that are defined in the Unix header files are defined; for a few symbols, default values are provided.

Distinto en la versión 3.6: `SO_DOMAIN`, `SO_PROTOCOL`, `SO_PEERSEC`, `SO_PASSSEC`, `TCP_USER_TIMEOUT`, `TCP_CONGESTION` han sido agregados.

Distinto en la versión 3.6.5: Added support for `TCP_FASTOPEN`, `TCP_KEEPCNT` on Windows platforms when available.

Distinto en la versión 3.7: `TCP_NOTSENT_LOWAT` ha sido agregada.

Added support for `TCP_KEEPIDLE`, `TCP_KEEPINTVL` on Windows platforms when available.

Distinto en la versión 3.10: Se agregó `IP_RECVTOS`. Se agregó

`TCP_KEEPALIVE`. En MacOS, esta constante se puede utilizar de la misma forma que `TCP_KEEPIDLE` en Linux.

Distinto en la versión 3.11: Se agregó `TCP_CONNECTION_INFO`. En MacOS esta constante se puede utilizar de la misma manera que `TCP_INFO` se utiliza en Linux y BSD.

Distinto en la versión 3.12: Se agregaron `SO_RTABLE` y `SO_USER_COOKIE`. En OpenBSD y FreeBSD respectivamente, esas constantes se pueden usar de la misma manera que `SO_MARK` es usado en Linux. También se agregaron opciones de socket TCP faltantes de Linux: `TCP_MD5SIG`, `TCP_THIN_LINEAR_TIMEOUTS`, `TCP_THIN_DUPACK`, `TCP_REPAIR`, `TCP_REPAIR_QUEUE`, `TCP_QUEUE_SEQ`, `TCP_REPAIR_OPTIONS`, `TCP_TIMESTAMP`, `TCP_CC_INFO`, `TCP_SAVE_SYN`, `TCP_SAVED_SYN`, `TCP_REPAIR_WINDOW`, `TCP_FASTOPEN_CONNECT`, `TCP_ULP`, `TCP_MD5SIG_EXT`, `TCP_FASTOPEN_KEY`, `TCP_FASTOPEN_NO_COOKIE`, `TCP_ZEROCOPY_RECEIVE`, `TCP_INQ`, `TCP_TX_DELAY`. Se agregó `IP_PKTINFO`, `IP_UNBLOCK_SOURCE`, `IP_BLOCK_SOURCE`, `IP_ADD_SOURCE_MEMBERSHIP`, `IP_DROP_SOURCE_MEMBERSHIP`.

Distinto en la versión 3.13: Added `SO_BINDTOIFINDEX`. On Linux this constant can be used in the same way that `SO_BINDTODEVICE` is used, but with the index of a network interface instead of its name.

Distinto en la versión 3.14: Added missing `IP_FREEBIND`, `IP_RECVERR`, `IPV6_RECVERR`, `IP_RECVTTL`, and `IP_RECVORIGDSTADDR` on Linux.

Distinto en la versión 3.14: Added support for `TCP_QUICKACK` on Windows platforms when available.

socket.AF_CAN
socket.PF_CAN
SOL_CAN_*
CAN_*

Muchas constantes de estos formularios, documentadas en la documentación de Linux, también se definen en el módulo de socket.

Availability: Linux > = 2.6.25, NetBSD > = 8.

Added in version 3.3.

Distinto en la versión 3.11: Se ha agregado compatibilidad con NetBSD.

Distinto en la versión 3.14: Restored missing `CAN_RAW_ERR_FILTER` on Linux.

socket.CAN_BCM
CAN_BCM_*

CAN_BCM, en la familia de protocolo CAN, es el protocolo del administrador de difusión (BCM. Las constantes del administrador de difusión, documentada en la documentación de Linux, también esta definidos en el modulo socket.

Availability: Linux > = 2.6.25.

Nota

El indicador **`CAN_BCM_CAN_FD_FRAME`** esta solamente disponible en Linux > = 4.8.

Added in version 3.4.

socket.CAN_RAW_FD_FRAMES

Habilita la compatibilidad con CAN FD en un socket CAN_RAW. Esta opción está deshabilitada de forma predeterminada. Esto permite que la aplicación envíe tramas CAN y CAN FD; sin embargo, debe aceptar las tramas CAN y CAN FD al leer desde el socket.

Esta constante se documenta en la documentación de Linux.

Availability: Linux > = 3.6.

Added in version 3.5.

socket.CAN_RAW_JOIN_FILTERS

Se une a los filtros CAN aplicados de modo que solo las tramas CAN que coinciden con todos los filtros CAN dados se pasan al espacio del usuario.

Esta constante se documenta en la documentación de Linux.

Availability: Linux > = 4.1.

Added in version 3.9.

socket.CAN_ISOTP

CAN_ISOTP, en el protocolo de familia CAN, es el protocolo ISO-TP (ISO 15765-2). Constantes ISO-TP, documentadas en la documentación Linux.

Availability: Linux > = 2.6.25.

Added in version 3.7.

socket.CAN_J1939

CAN_J1939, en el protocolo de familias CAN, es el protocolo SAE J1939. Constantes J1939, documentadas en el documentación Linux.

Availability: Linux > = 5.4.

Added in version 3.9.

socket.AF_DIVERT
socket.PF_DIVERT

Estas dos constantes, documentadas en la página del manual de FreeBSD divert(4), también están definidas en el módulo de socket.

Availability: FreeBSD > = 14.0.

Added in version 3.12.

socket.AF_PACKET
socket.PF_PACKET
PACKET_*

Muchas constantes de estos formularios, documentadas en la documentación de Linux, también se definen en el módulo de socket.

Availability: Linux > = 2.2.

socket.ETH_P_ALL

El `service_id` es el identificador del servicio registrado.:data:*!ETH_P_ALL* se puede utilizar en el constructor **`socket`** como *proto* para la familia **`AF_PACKET`** para capturar cada paquete, independientemente del protocolo.

Para obtener más información, consulte la página de manual *packet(7)* *[https://manpages.debian.org/packet(7)]*.

Availability: Linux.

Added in version 3.12.

socket.AF_RDS
socket.PF_RDS
socket.SOL_RDS
RDS_*

Muchas constantes de estos formularios, documentadas en la documentación de Linux, también se definen en el módulo de socket.

Availability: Linux > = 2.6.30.

Added in version 3.3.

socket.SIO_RCVALL
socket.SIO_KEEPALIVE_VALS
socket.SIO_LOOPBACK_FAST_PATH
RCVALL_*

Constantes para Windows' WSAIoctl(). Las constantes se utiliza como argumentos al método `ioctl()` de objetos de sockets.

Distinto en la versión 3.6: `SIO_LOOPBACK_FAST_PATH` ha sido agregado.

TIPC_*

LAS constantes relacionadas con TIPC, que coinciden con las exportadas por la API de socket de C. Consulte la documentación de TIPC para obtener más información.

socket.AF_ALG
socket.SOL_ALG
ALG_*

Constantes para la criptográfica del Kernel de Linux.

Availability: Linux > = 2.6.38.

Added in version 3.6.

socket.AF_VSOCK
socket.IOCTL_VM_SOCKETS_GET_LOCAL_CID
VMADDR*
SO_VM*

Constantes para la comunicación host/invitado de Linux.

Availability: Linux > = 4.8.

Added in version 3.7.

socket.AF_LINK

Availability: BSD, macOS.

Added in version 3.4.

socket.has_ipv6

Esta constante contiene un valor booleano que indica si IPv6 se admite en esta plataforma.

socket.AF_BLUETOOTH
socket.BTPROTO_L2CAP
socket.BTPROTO_RFCOMM
socket.BTPROTO_HCI
socket.BTPROTO_SCO

Integer constants for use with Bluetooth addresses.

socket.BDADDR_ANY
socket.BDADDR_LOCAL

Estas son constantes de cadenas que contienen direcciones Bluetooth con significados especiales. Por ejemplo `BDADDR_ANY` son usados para indicar cualquier dirección al especificar el socket vinculante con `BTPROTO_RFCOMM`.

socket.BDADDR_BREDR
socket.BDADDR_LE_PUBLIC
socket.BDADDR_LE_RANDOM

These constants describe the Bluetooth address type when binding or connecting a `BTPROTO_L2CAP` socket.

Availability: Linux, FreeBSD

Added in version 3.14.

socket.SOL_RFCOMM
socket.SOL_L2CAP
socket.SOL_HCI
socket.SOL_SCO
socket.SOL_BLUETOOTH

Used in the level argument to the `setsockopt()` and `getsockopt()` methods of Bluetooth socket objects.

`SOL_BLUETOOTH` is only available on Linux. Other constants are available if the corresponding protocol is supported.

SO_L2CAP_*
socket.L2CAP_LM
L2CAP_LM_*
SO_RFCOMM_*

RFCOMM_LM_*
SO_SCO_*
SO_BTH_*
BT_*

Used in the option name and value argument to the `setsockopt()` and `getsockopt()` methods of Bluetooth socket objects.

`BT_*` and `L2CAP_LM` are only available on Linux. **`SO_BTH_*`** are only available on Windows. Other constants may be available on Linux and various BSD platforms.

Added in version 3.14.

socket.HCI_FILTER
socket.HCI_TIME_STAMP
socket.HCI_DATA_DIR
socket.SO_HCI_EVT_FILTER
socket.SO_HCI_PKT_FILTER

Option names for use with `BTPROTO_HCI`. Availability and format of the option values depend on platform.

Distinto en la versión 3.14: Added **`SO_HCI_EVT_FILTER`** and **`SO_HCI_PKT_FILTER`** on NetBSD and DragonFly BSD. Added **`HCI_DATA_DIR`** on FreeBSD, NetBSD and DragonFly BSD.

socket.HCI_DEV_NONE

The `device_id` value used to create an HCI socket that isn't specific to a single Bluetooth adapter.

Availability: Linux

Added in version 3.14.

socket.HCI_CHANNEL_RAW
socket.HCI_CHANNEL_USER
socket.HCI_CHANNEL_MONITOR
socket.HCI_CHANNEL_CONTROL
socket.HCI_CHANNEL_LOGGING

Possible values for `channel` field in the **`BTPROTO_HCI`** address.

Availability: Linux

Added in version 3.14.

socket.AF_QIPCRTR

Constante para el protocolo de router IPC de Qualcomm, que se utiliza para comunicarse con procesadores remotos que brindan servicios.

Availability: Linux >= 4.7.

socket.SCM_CREDS2
socket.LOCAL_CREDS
socket.LOCAL_CREDS_PERSISTENT

LOCAL_CREDS y LOCAL_CREDS_PERSISTENT pueden usarse con sockets SOCK_DGRAM, SOCK_STREAM, equivalente a Linux/DragonFlyBSD SO_PASSCRED, mientras que LOCAL_CREDS envía las credenciales en la primera lectura, LOCAL_CREDS_PERSISTENT envía para cada lectura, SCM_CREDS2 debe entonces ser usado para este último para el tipo de mensaje.

Added in version 3.11.

Availability: FreeBSD.

socket.SO_INCOMING_CPU

Constante para optimizar la localidad CPU, a ser usada en conjunto con **`SO_REUSEPORT`**.

Added in version 3.11.

Availability: Linux >= 3.9

socket.SO_REUSEPORT_LB

Constant to enable duplicate address and port bindings with load balancing.

Added in version 3.14.

Availability: FreeBSD > = 12.0

socket.AF_HYPERV
socket.HV_PROTOCOL_RAW
socket.HVSOCKET_CONNECT_TIMEOUT
socket.HVSOCKET_CONNECT_TIMEOUT_MAX
socket.HVSOCKET_CONNECTED_SUSPEND
socket.HVSOCKET_ADDRESS_FLAG_PASSTHRU
socket.HV_GUID_ZERO
socket.HV_GUID_WILDCARD
socket.HV_GUID_BROADCAST
socket.HV_GUID_CHILDREN
socket.HV_GUID_LOOPBACK
socket.HV_GUID_PARENT

Constantes para sockets Windows Hyper-V para comunicaciones host/invitado.

Availability: Windows.

Added in version 3.12.

socket.ETHERTYPE_ARP
socket.ETHERTYPE_IP
socket.ETHERTYPE_IPV6
socket.ETHERTYPE_VLAN

Número de protocolo IEEE 802.3 [https://www.iana.org/assignments/ieee-802-números/ieee-802-numbers.txt]. constantes.

Availability: Linux, FreeBSD, macOS.

Added in version 3.12.

socket.SHUT_RD
socket.SHUT_WR
socket.SHUT_RDWR

These constants are used by the `shutdown()` method of socket objects.

Availability: not WASI.

Funciones

Creación de sockets

Todas las siguientes funciones crean socket objects.

class socket.socket(*family=AF_INET*, *type=SOCK_STREAM*, *proto=0*, *fileno=None*)

Crear un nuevo socket usando la dirección de familia dada, tipo de socket y el numero de protocolo. La dirección de familia debería ser **`AF_INET`** (por defecto), **`AF_INET6`**, **`AF_UNIX`**, **`AF_CAN`**, **`AF_PACKET`**, o **`AF_RDS`**. El tipo de socket debería ser **`SOCK_STREAM`** (por defecto), **`SOCK_DGRAM`**, **`SOCK_RAW`** o quizás una de las otras constantes `SOCK_`. El numero de protocolo es usualmente cero u omitirse o en el caso donde la familia de dirección es **`AF_CAN`** el protocolo debería ser uno de **`CAN_RAW`**, **`CAN_BCM`**, **`CAN_ISOTP`** o **`CAN_J1939`**.

If *fileno* is specified, the values for *family*, *type*, and *proto* are auto-detected from the specified file descriptor. Auto-detection can be overruled by calling the function with explicit *family*, *type*, or *proto* arguments. This only affects how Python represents e.g. the return value of **`socket.getpeername()`** but not the actual OS resource. Unlike **`socket.fromfd()`**, *fileno* will return the same socket and not a duplicate. This may help close a detached socket using **`socket.close()`**.

El socket recién creado es non-inheritable.

Genera un auditing event `socket.__new__` con los argumentos `self`, `family`, `type`, `protocol`.

Distinto en la versión 3.3: Se añadió la familia AF_CAN. Se añadió la familia AF_RDS.

Distinto en la versión 3.4: El protocolo CAN_BCM ha sido agregado.

Distinto en la versión 3.4: Los sockets devueltos ahora no son heredables.

Distinto en la versión 3.7: El protocolo CAN_ISOTP ha sido agregado.

Distinto en la versión 3.7: Cuando las banderas bit **`SOCK_NONBLOCK`** or **`SOCK_CLOEXEC`** están aplicadas a *type*, se borran, y **`socket.type`** no las reflejará. Igual se pasan a la llamada *socket ()* del sistema subyacente. Por lo tanto,

```
sock = socket.socket(
    socket.AF_INET,
    socket.SOCK_STREAM | socket.SOCK_NONBLOCK)
```

seguirá creando un socket sin bloqueo en los sistemas operativos que admiten `SOCK_NONBLOCK`, pero `sock.type` se establecerá en `socket.SOCK_STREAM`.

Distinto en la versión 3.9: El protocolo CAN_J1939 ha sido agregado.

Distinto en la versión 3.10: Se agregó el protocolo IPPROTO_MPTCP.

socket.socketpair([*family*[, *type*[, *proto*]]])

Build a pair of connected socket objects using the given address family, socket type, and protocol number. Address family, socket type, and protocol number are as for the **`socket()`** function. The default family is **`AF_UNIX`** if defined on the platform; otherwise, the default is **`AF_INET`**.

Los sockets creados recientemente son non-inheritable.

Distinto en la versión 3.2: Los objetos de socket devueltos ahora admiten toda la API de socket, en lugar de un subconjunto.

Distinto en la versión 3.4: Los sockets devueltos ahora no son heredables.

Distinto en la versión 3.5: Se ha agregado compatibilidad con Windows.

socket.create_connection(*address*, *timeout=GLOBAL_DEFAULT*, *source_address=None*, ***, *all_errors=False*)

Se conecta a un servicio TCP que esté escuchando en Internet *address* (un `(host, port)` de 2 tuplas) y retorna el objeto de socket. Esta es una función de nivel superior que **`socket.connect()`**: si *host* es un nombre de host no numérico, intentará resolverlo para **`AF_INET`** y **`AF_INET6`**, y luego intentará conectarse a todas las direcciones posibles sucesivamente hasta que la conexión se realice correctamente. Esto facilita la escritura de clientes que sean compatibles con IPv4 e IPv6.

Pasando el parámetro opcional *timeout* establece el tiempo de espera dentro de la instancia del socket. Si no es proporcionado *timeout*, la configuración global de tiempo de espera predeterminada retornada por **`getdefaulttimeout()`** es usada.

Si se suministra, *source_address* debe ser una “”(host, puerto)”” de 2 tuplas para que el socket se enlace como su dirección de origen antes de conectarse. Si el host o el puerto son “” o 0 respectivamente, se utilizará el comportamiento predeterminado del sistema operativo.

Cuando una conexión no puede ser creada, se lanza una exception. Por defecto, es la excepción de la última dirección en la lista. Si *all_errors* es `True`, es un **`ExceptionGroup`** conteniendo los errores de todos los intentos.

Distinto en la versión 3.2: source_address ha sido agregado.

Distinto en la versión 3.11: all_errors ha sido agregado.

socket.create_server(*address*, ***, *family=AF_INET*, *backlog=None*, *reuse_port=False, dualstack_ipv6=False*)

Función conveniente que crea un socket TCP vinculado a *dirección* (un `(host, puerto)` de 2 tuplas) y devuelve el objeto del socket.

familia debe ser **`AF_INET`** o **`AF_INET6`**. *backlog* es el tamaño de la cola pasado a **`socket.listen()`**; cuando `0` se elige un valor predeterminado razonable. *reuse_port* dicta si se debe establecer la opción de socket **`SO_REUSEPORT`**.

If *dualstack_ipv6* is true, *family* is **`AF_INET6`** and the platform supports it the socket will be able to accept both IPv4 and IPv6 connections, else it will raise **`ValueError`**. Most POSIX platforms and Windows are supposed to support this functionality. When this functionality is enabled the address returned by **`socket.getpeername()`** when an IPv4 connection occurs will be an IPv6 address represented as an IPv4-mapped IPv6 address. If *dualstack_ipv6* is false it will explicitly disable this functionality on platforms that enable it by default (e.g. Linux). This parameter can be used in conjunction with **`has_dualstack_ipv6()`**:

```
import socket

addr = ("", 8080)  # all interfaces, port 8080
if socket.has_dualstack_ipv6():
    s = socket.create_server(addr, family=socket.AF
else:
    s = socket.create_server(addr)
```

Nota

En plataformas POSIX la opción del socket **`SO_REUSEADDR`** está configurado para inmediatamente rehusar los sockets previos que estaban vinculados la misma *address* y permanecer en estado TIME_WAIT.

Added in version 3.8.

socket.has_dualstack_ipv6()

Retorna `True` si la plataforma admite la creación de un socket TCP que pueda manejar conexiones IPv4 e IPv6.

Added in version 3.8.

socket.fromfd(*fd*, *family*, *type*, *proto=0*)

Duplicate the file descriptor *fd* (an integer as returned by a file object's **`fileno()`** method) and build a socket object from the result. Address family, socket type and protocol number are as for the **`socket()`** function. The file descriptor should refer to a socket, but this is not checked — subsequent operations on the object may fail if the file descriptor is invalid. This function is rarely needed, but can be used to get or set socket options on a socket passed to a program as standard input or output (such as a server started by the Unix inet daemon). The socket is assumed to be in blocking mode.

El socket recién creado es non-inheritable.

Distinto en la versión 3.4: Los sockets devueltos ahora no son heredables.

socket.fromshare(*data*)

Cree una instancia de un socket a partir de los datos obtenidos del método **`socket.share()`**. Se supone que el socket está en modo de bloqueo.

Availability: Windows.

Added in version 3.3.

socket.SocketType

Este es un tipo de objeto Python que representa el tipo de objeto del socket. Es lo mismo que decir `type(socket(...))`.

Otras funciones

The **`socket`** module also offers various network-related services:

socket.close(*fd*)

Close a socket file descriptor. This is like **`os.close()`**, but for sockets. On some platforms (most notably Windows) **`os.close()`** does not work for socket file descriptors.

Added in version 3.7.

socket.getaddrinfo(*host*, *port*, *family=AF_UNSPEC*, *type=0*, *proto=0*, *flags=0*)

This function wraps the C function `getaddrinfo` of the underlying system.

Traduce el argumento *host/port* dentro de una secuencia de 5 tuplas que contiene todo los argumentos necesarios para crear un socket conectado a ese servicio. *host* es un nombre de dominio, una cadena en representación de una dirección IPV4/IPV6 o `None`. *port* es una nombre de una cadena de servicio como `'http'`, un numero de puerto numérico o `None`. Pasando `None` como el valor del *host* y *port*, pasando `NULL` a la API C subyacente.

The *family*, *type* and *proto* arguments can be optionally specified in order to provide options and limit the list of addresses returned. Pass their default values (**`AF_UNSPEC`**, 0, and 0, respectively) to not limit the results. See the note below for details.

The *flags* argument can be one or several of the `AI_*` constants, and will influence how results are computed and returned. For example, **`AI_NUMERICHOST`** will disable domain name resolution and will raise an error if *host* is a domain name.

La función devuelve una lista de 5 tuplas con la siguiente estructura:

```
(family, type, proto, canonname, sockaddr)
```

In these tuples, *family*, *type*, *proto* are all integers and are meant to be passed to the `socket()` function. *canonname* will be a string representing the canonical name of the *host* if **`AI_CANONNAME`** is part of the *flags* argument; else *canonname* will be empty. *sockaddr* is a tuple describing a socket address, whose format depends on the returned *family* (a `(address, port)` 2-tuple for `AF_INET`, a `(address, port, flowinfo, scope_id)` 4-tuple for `AF_INET6`), and is meant to be passed to the `socket.connect()` method.

> **Nota**
>
> If you intend to use results from **`getaddrinfo()`** to create a socket (rather than, for example, retrieve *canonname*), consider limiting the results by *type* (e.g. `SOCK_STREAM` or `SOCK_DGRAM`) and/or *proto* (e.g. **`IPPROTO_TCP`** or **`IPPROTO_UDP`**) that your application can handle.
>
> The behavior with default values of *family*, *type*, *proto* and *flags* is system-specific.
>
> Many systems (for example, most Linux configurations) will return a sorted list of all matching addresses. These addresses should generally be tried in order until a connection succeeds (possibly tried in parallel, for example, using a Happy Eyeballs [https://en.wikipedia.org/wiki/Happy_Eyeballs] algorithm). In these cases, limiting the *type* and/or *proto* can help eliminate unsuccessful or unusable connection attempts.
>
> Some systems will, however, only return a single address. (For example, this was reported on Solaris and AIX configurations.) On these systems, limiting the *type* and/or *proto* helps ensure that this address is usable.

Genera un auditing event `socket.getaddrinfo` con los argumentos `host`, `port`, `family`, `type`, `protocol`.

En el ejemplo siguiente se obtiene información de dirección para una conexión TCP hipotética a `example.org` en el puerto 80 (los resultados pueden diferir en el sistema si IPv6 no está habilitado):

```
>>> socket.getaddrinfo("example.org", 80, proto=soc
[(socket.AF_INET6, socket.SOCK_STREAM,
 6, '', ('2606:2800:220:1:248:1893:25c8:1946', 80,
 (socket.AF_INET, socket.SOCK_STREAM,
 6, '', ('93.184.216.34', 80))]
```

Distinto en la versión 3.2: los parámetros ahora se pueden pasar mediante argumentos de palabra clave.

Distinto en la versión 3.7: para direcciones de multidifusión IPv6, la cadena que representa una dirección no contendrá partes `%scope_id`.

socket.getfqdn([*name*])

Retorna un nombre de dominio completo para *name*. Si *name* se omite o está vacío, se interpreta como el host local. Para encontrar el nombre completo, se comprueba el nombre de host retornado por **`gethostbyaddr()`**, seguido de los alias del host, si están disponibles. Se selecciona el primer nombre que incluye un punto. En caso de que no haya disponible un nombre de dominio completo y se haya proporcionado *name*, se retorna sin cambios. Si *name* estaba vacío o era igual a `'0.0.0.0'`, se retorna el nombre de host de **`gethostname()`**.

socket.gethostbyname(*hostname*)

Traduce un nombre de host a un formato de dirección IPV4. La dirección IPV4 es retornada como una cadena, como `’100.50.200.5’`. Si el nombre de host es una dirección IPV4 en sí, se devuelve sin cambios. Observa **`gethostbyname_ex()`** para una interfaz mas completa. **`gethostbyname()`** no soporta la resolución de nombres IPV6, y **`getaddrinfo()`** debe utilizarse en su lugar para compatibilidad con doble pila IPv4/v6.

Genera un evento auditing `socket.gethostbyname` con el argumento `hostname`.

Availability: not WASI.

socket.gethostbyname_ex(*hostname*)

Traducir un nombre de host a IPv4, formato de dirección de interfaz extendida. Devuelve una tupla de 3 elementos `(hostname, aliaslist, ipaddrlist)` donde *hostname* es el nombre de host principal del host, *aliaslist* es una lista (posiblemente vacía) de nombres de host alternativos para la misma dirección y *ipaddrlist* es una lista de direcciones IPv4 para la misma interfaz en el mismo host (a menudo, pero no siempre, una única dirección). **`gethostbyname_ex()`** no admite la resolución de nombres IPv6 y **`getaddrinfo()`** debe usarse en su lugar para la compatibilidad con doble pila IPv4/v6.

Genera un evento auditing `socket.gethostbyname` con el argumento `hostname`.

Availability: not WASI.

socket.gethostname()

Retorna una cadena que contenga el nombre de host de la máquina donde se está ejecutando actualmente el intérprete de Python.

Genera un auditing event `socket.gethostname` sin argumentos.

Nota: **`gethostname()`** no siempre retorna el nombre de dominio completo, usa **`getfqdn()`** para eso.

Availability: not WASI.

socket.gethostbyaddr(*ip_address*)

Devuelve una tupla de 3 elementos `(nombre de host, lista de alias, lista de ipaddr)` donde *nombre de host* es el nombre de host principal que responde a la *dirección_ip* dada, *lista de alias* es una lista (posiblemente vacía) de nombres de host alternativos para el mismo dirección, y *ipaddrlist* es una lista de direcciones IPv4/v6 para la misma interfaz en el mismo host (lo más probable es que contenga solo una dirección). Para encontrar el nombre de dominio completo, utilice la función **`getfqdn()`**. **`gethostbyaddr()`** admite tanto IPv4 como IPv6.

Generar un auditing event `socket.gethostbyaddr` con los argumentos `ip_address`.

Availability: not WASI.

socket.getnameinfo(*sockaddr*, *flags*)

Traduce una dirección de socket *sockaddr* en una 2-tupla `(host, port)`. Dependiendo de la configuraciones de *flags*, el resultado puede contener un nombre de dominio completo o una representación numérica de la dirección en *host*. De igual manera, *port* puede contener un nombre de puerto de cadena de caracteres o un numero de puerto numérico.

Para las direcciones IPv6, `%scope` se anexa a la parte del host si *sockaddr* contiene *scopeid* significativo. Generalmente esto sucede para las direcciones de multidifusión.

Para mas información sobre *flags* pueden consultar *getnameinfo(3)* *[https://manpages.debian.org/getnameinfo(3)]*.

Plantea un auditing event `socket.getnameinfo` con el argumento `sockaddr`.

Availability: not WASI.

socket.getprotobyname(*protocolname*)

Translate an internet protocol name (for example, `'icmp'`) to a constant suitable for passing as the (optional) third argument to the **`socket()`** function. This is usually only needed for sockets opened in «raw» mode (**`SOCK_RAW`**); for the normal socket modes, the correct protocol is chosen automatically if the protocol is omitted or zero.

Availability: not WASI.

socket.getservbyname(*servicename*[, *protocolname*])

Traduzca un nombre de servicio de Internet y un nombre de protocolo a un número de puerto para ese servicio. El nombre de protocolo opcional, si se proporciona, debe ser `'tcp'` o `'udp'`; de lo contrario, cualquier protocolo coincidirá.

Genera un auditing event `socket.getservbyname` con los argumentos `servicename`, `protocolname`.

Availability: not WASI.

socket.getservbyport(*port*[, *protocolname*])

Traduzca un número de puerto de Internet y un nombre de protocolo a un nombre de servicio para ese servicio. El nombre de protocolo opcional, si se proporciona, debe ser `'tcp'` o `'udp'`; de lo contrario, cualquier protocolo coincidirá.

Genera un auditing event `socket.getservbyport` con los argumentos `port`, `protocolname`.

Availability: not WASI.

socket.ntohl(*x*)

Convierta enteros positivos de 32 bits de red a orden de bytes de host. En equipos donde el orden de bytes de host es el mismo que el orden de bytes de red, se trata de un no-op; de lo contrario, realiza una operación de intercambio de 4 bytes.

socket.ntohs(*x*)

Convierta enteros positivos de 16 bits de red a orden de bytes de host. En equipos donde el orden de bytes de host es el mismo que el orden de bytes de red, se trata de un no-op; de lo contrario, realiza una operación de intercambio de 2 bytes.

Distinto en la versión 3.10: Lanza **`OverflowError`** si *x* no cabe en un entero sin signo de 16 bits.

socket.htonl(*x*)

Convierta enteros positivos de 32 bits del host al orden de bytes de red. En equipos donde el orden de bytes de host es el mismo que el orden de bytes de red, se trata de un no-op; de lo contrario, realiza una operación de intercambio de 4 bytes.

socket.htons(*x*)

Convierta enteros positivos de 16 bits del host al orden de bytes de red. En equipos donde el orden de bytes de host es el mismo que el orden de bytes de red, se trata de un no-op; de lo contrario, realiza una operación de intercambio de 2 bytes.

Distinto en la versión 3.10: Lanza `OverflowError` si *x* no cabe en un entero sin signo de 16 bits.

socket.inet_aton(*ip_string*)

Convierte una dirección IPv4 desde el formato de cadena de cuatro puntos (por ejemplo, '123.45.67.89') a formato binario empaquetado de 32 bits, como un objeto de bytes de cuatro caracteres de longitud. Esto es útil cuando se convierte con un programa que usa la librería estándar C y necesita objetos de tipo `in_addr`, que es el tipo C para el binario empaquetado de 32 bits que devuelve esta función.

Ademas `` inet_aton`acepta cadena de caracteres con menos de tres puntos, observar la pagina del manual Unix :manpage:`inet(3)() `` para mas detalles.

Si la cadena de dirección IPv4 es pasada a esta función es invalido, `OSError` se lanzará. Tenga en cuenta que exactamente lo que es valido depende de la implementación C de `inet_aton()`.

`inet_aton()` no admite IPV6, y `inet_pton()` deberían utilizarse en su lugar para compatibilidad con doble pilas IPV4/v6.

socket.inet_ntoa(*packed_ip*)

Convierte una dirección IPv4 empaquetada de 32 bits (un bytes-like object cuatro bytes de longitud) a su representación estándar de cadena de cuatro puntos (por ejemplo '123.45.67.89'). Esto es útil cuando convertimos con un programa que usa la librería estándar C y necesita objetos de tipo `in_addr`, que es el tipo C para los datos binarios empaquetados de 32 bits que esta función toma como argumento.

Si la secuencia de byte pasada a esta función no es exactamente 4 bytes de longitud `OSError` podría generarse `inet_ntoa()` no soporta IPV6, y `inet_ntop()` debe utilizarse en su lugar para compatibilidad con doble pila IPv4 / v6.

Distinto en la versión 3.5: Ahora se acepta la grabación bytes-like object.

socket.inet_pton(*address_family*, *ip_string*)

Convierte una dirección IP desde su formato de cadena específico de la familia a un formato binario empaquetado. `inet_pton()` es útil cuando una librería o protocolo de red llama desde un objeto de tipo `in_addr` (similar a `inet_aton()`) o `in6_addr`.

Los Valores soportados para *address_family* son actualmente `AF_INET` y `AF_INET6`. Si la cadena de dirección IP *ip_string* no es válida, `OSError` se genera. Tenga en cuenta que exactamente lo que es válido depende tanto del valor de *address_family* como de la implementación subyacente de `inet_pton()`.

Availability: Unix, Windows.

Distinto en la versión 3.4: Se ha añadido compatibilidad con Windows

socket.inet_ntop(*address_family*, *packed_ip*)

Convierte una dirección IP empaquetada (un bytes-like object de algún numero de bytes) a su representación estándar de cadena específica de la familia (por ejemplo `'7.10.0.5'` o `'5aef:2b::8'`). **`inet_ntop()`** es útil cuando una librería o protocolo de red retorna un objeto de tipo **`in_addr`** (similar para **`inet_ntoa()`**) o **`in6_addr`**.

Los valores soportados para *address_family* actualmente son **`AF_INET`** y **`AF_INET6`**. Si los objetos de bytes *packed_ip* no tienen la longitud correcta para la familia de direcciones especificas, **`ValueError`** podría generarse. **`OSError`** se genera para errores desde la llamada a **`inet_ntop()`**.

Availability: Unix, Windows.

Distinto en la versión 3.4: Se ha añadido compatibilidad con Windows

Distinto en la versión 3.5: Ahora se acepta la grabación bytes-like object.

socket.CMSG_LEN(*length*)

Retorna la longitud total, sin relleno de arrastre, de un elemento de datos auxiliares con datos asociados del *length*. Este valor se puede utilizar a menudo como tamaño de búfer para **`recvmsg()`** para recibir un solo valor de datos auxiliares, pero **RFC 3542** [https://datatracker.ietf.org/doc/html/rfc3542.html] requiere aplicaciones portables para usar **`CMSG_SPACE()`** y así incluir espacio para el relleno, incluso cuando el elemento será el último en el búfer. Genera **`OverflowError`** si *length* está fuera del rango de valores permitido.

Availability: Unix, not WASI.

La mayoría de las plataformas Unix.

Added in version 3.3.

socket.CMSG_SPACE(*length*)

Retorna el tamaño del buffer necesario por **`recvmsg()`** para recibir un elemento de datos auxiliares con datos asociados del *length* dado, junto con cualquier relleno final. El espacio de buffer necesario para recibir múltiples elementos es la suma de los valores de **`CMSG_SPACE()`** para los datos asociados con la longitudes. Genera **`OverflowError`** si *length* está fuera del rango de valores permitido.

Tenga en cuenta que algunos sistemas pueden admitir datos auxiliares sin proporcionar esta función. Tenga en cuenta también que establecer el tamaño del búfer utilizando los resultados de esta función puede no limitar con precisión la cantidad de datos auxiliares que se pueden recibir, ya que los datos adicionales pueden caber en el área de relleno.

Availability: Unix, not WASI.

la mayoría de las plataformas Unix.

Added in version 3.3.

socket.getdefaulttimeout()

Retorna el tiempo de espera por defecto en segundos (flotante) para los objetos de un nuevo socket. Un valor de `None` indicada que los objetos del nuevo socket no tiene un tiempo de espera. Cuando el modulo socket es importado primero, por defecto es `None`.

socket.setdefaulttimeout(*timeout*)

Selecciona el tiempo de espera por defecto en segundos (flotante) para los objetos nuevos del socket. Cuando el modulo socket es importado primero, el valor por defecto es `None`. Observar **`settimeout()`** para posible valores y sus respectivos significados.

socket.sethostname(*name*)

Establece el nombre de host de la maquina en *name*. Esto genera un **`OSError`** si no tiene suficientes derechos.

Plantea un auditing event `socket.sethostname` con el argumento `name`.

Availability: Unix, not Android.

Added in version 3.3.

socket.if_nameindex()

Retorna una lista de tuplas de información de interfaz de red (índice int, cadena de nombre). **`OSError`** si se produce un error en la llamada del sistema.

Availability: Unix, Windows, not WASI.

Added in version 3.3.

Distinto en la versión 3.8: Se ha agregado compatibilidad con Windows.

Nota

En Windows las interfaces de redes tienen diferentes nombres en diferentes contextos (todos los nombres son ejemplos):

- UUID: `{FB605B73-AAC2-49A6-9A2F-25416AEA0573}`
- nombre: `ethernet_32770`
- nombre amigable: `vEthernet (nat)`
- descripción: `Hyper-V Virtual Ethernet Adapter`

Esta función retorna los nombres del segundo formulario de la lista, en este caso de ejemplo `ethernet_32770`.

socket.if_nametoindex(*if_name*)

Retorna un número de índice de interfaz de red correspondiente a un nombre de interfaz. **`OSError`** si no existe ninguna interfaz con el nombre especificado.

Availability: Unix, Windows, not WASI.

Added in version 3.3.

Distinto en la versión 3.8: Se ha agregado compatibilidad con Windows.

Ver también

“”Interface name” es un nombre como se documenta en `if_nameindex()`.

socket.if_indextoname(*if_index*)

Retorna un nombre de interfaz de red correspondiente a un número de índice de interfaz. `OSError` si no existe ninguna interfaz con el índice dado.

Availability: Unix, Windows, not WASI.

Added in version 3.3.

Distinto en la versión 3.8: Se ha agregado compatibilidad con Windows.

Ver también

“”Interface name” es un nombre como se documenta en `if_nameindex()`.

socket.send_fds(*sock*, *buffers*, *fds*[, *flags*[, *address*]])

Send the list of file descriptors *fds* over an `AF_UNIX` socket *sock*. The *fds* parameter is a sequence of file descriptors. Consult `sendmsg()` for the documentation of these parameters.

Availability: Unix, not WASI.

Plataformas Unix que soporten `sendmsg()` y el mecanismo

`SCM_RIGHTS`.

Added in version 3.9.

socket.recv_fds(*sock*, *bufsize*, *maxfds*[, *flags*])

Receive up to *maxfds* file descriptors from an **`AF_UNIX`** socket *sock*. Return `(msg, list(fds), flags, addr)`. Consult **`recvmsg()`** for the documentation of these parameters.

Availability: Unix, not WASI.

Unix platforms supporting **`recvmsg()`** and **`SCM_RIGHTS`** mechanism.

Added in version 3.9.

Nota

Cualquier número entero truncado al final de la lista de descriptores de archivo.

Objetos Socket

Los objetos socket tienen los siguientes métodos. Excepto para **`makefile()`**, esto corresponde al sistema de llamadas Unix para sockets.

Distinto en la versión 3.2: El soporte para el protocolo context manager ha sido agregado. Salir del gestor de contexto es equivalente para el llamado **`close()`**.

socket.accept()

Acepta una conexión. El socket debe estar vinculado a una dirección y estar escuchando las conexiones. El valor de retorno es el par `(conn, address)` cuando *conn* es un *new* objeto socket usado para enviar y recibir información en la conexión, y *address* es la dirección vinculada al socket en el extremo de la conexión.

El socket recién creado es non-inheritable.

Distinto en la versión 3.4: El socket ahora no es heredable.

Distinto en la versión 3.5: Si se interrumpe la llamada del sistema y el controlador de señal no genera una excepción, el método ahora vuelve a intentar la llamada del sistema en lugar de generar una excepción **`InterruptedError`** (consulte **PEP 475** [https://peps.python.org/pep-0475/] para la lógica).

socket.bind(*address*)

Bind the socket to *address*. The socket must not already be bound. The format of *address* depends on the address family — see Familias Socket.

Genera un auditing event `socket.getaddrinfo` con los argumentos `host`, `port`, `family`, `type`, `protocol`.

Availability: not WASI.

socket.close()

Mark the socket closed. The underlying system resource (e.g. a file descriptor) is also closed when all file objects from **`makefile()`** are closed. Once that happens, all future operations on the socket object will fail. The remote end will receive no more data (after queued data is flushed).

Los sockets se cierran automáticamente cuando se recogen basura, pero se recomienda **`cerrarlos()`** explícitamente, o usar una instrucción **`with`** alrededor de ellos.

Distinto en la versión 3.6: **`OSError`** ahora se lanza si se produce un error cuando se realiza la llamada **`close()`** subyacente.

Nota

`close()` releases the resource associated with a connection but does not necessarily close the connection immediately. If you want to close the connection in a timely fashion, call **`shutdown()`** before **`close()`**.

socket.connect(*address*)

Connect to a remote socket at *address*. The format of *address* depends on the address family — see Familias Socket.

If the connection is interrupted by a signal, the method waits until the connection completes, or raises a **`TimeoutError`** on timeout, if the signal handler doesn't raise an exception and the socket is blocking or has a timeout. For non-blocking sockets, the method raises an **`InterruptedError`** exception if the connection is interrupted by a signal (or the exception raised by the signal handler).

Genera un auditing event `socket.connect` con los argumentos `self`, `address`.

Distinto en la versión 3.5: El método ahora espera hasta que se completa la conexión en lugar de generar una excepción **`InterruptedError`** si la conexión se interrumpe por una señal, el controlador de señal no genera una excepción y el socket está bloqueando o tiene un tiempo de espera (consulte el **PEP 475** [https://peps.python.org/pep-0475/] para la razón de ser).

Availability: not WASI.

socket.connect_ex(*address*)

Similar a `connect(address)`, pero retorna un indicador de error en lugar de generar una excepción para los errores retornados por la llamada de nivel C **`connect()`** (otros problemas, como "host no encontrado", aún pueden generar excepciones). El indicador de error es `0` si la operación tuvo éxito, caso contrario es el valor de la variable **`errno`**. Esto es útil para admitir, por ejemplo, conexiones asincrónicas.

Genera un auditing event `socket.connect` con los argumentos `self`, `address`.

Availability: not WASI.

socket.detach()

Coloque el objeto de socket en estado cerrado sin cerrar realmente el descriptor de archivo subyacente. Se devuelve el descriptor de archivo y se puede reutilizar para otros fines.

Added in version 3.2.

socket.dup()

Duplica el socket.

El socket recién creado es non-inheritable.

Distinto en la versión 3.4: El socket ahora no es heredable.

Availability: not WASI.

socket.fileno()

Retorna un archivo descriptor del socket (un entero pequeño), o -1 si falla. Esto es útil con **`select.select()`**.

En Windows el pequeño entero retornado por este método no puede ser usado donde un descriptor de un archivo pueda ser usado (como una **`os.fdopen()`**). Unix no tiene esta limitación.

socket.get_inheritable()

Obtiene el inheritable flag del descriptor del archivo del socket o el controlador del socket: `True` si el socket puede ser heredada en procesos secundarios, `False` si falla.

Added in version 3.4.

socket.getpeername()

Return the remote address to which the socket is connected. This is useful to find out the port number of a remote IPv4/v6 socket, for instance. The format of the address returned depends on the address family — see Familias Socket. On some systems this function is not supported.

socket.getsockname()

Return the socket's own address. This is useful to find out the port number of an IPv4/v6 socket, for instance. The format of the address returned depends on the address family — see Familias Socket.

socket.getsockopt(*level*, *optname*[, *buflen*])

Return the value of the given socket option (see the Unix man page *getsockopt(2)* *[https://manpages.debian.org/getsockopt(2)]*). The needed symbolic constants (SO_* etc.) are defined in this module. If *buflen* is absent, an integer option is assumed and its integer value is returned by the function. If *buflen* is present, it specifies the maximum length of the buffer used to receive the option in, and this buffer is returned as a bytes object. It is up to the caller to decode the contents of the buffer (see the optional built-in module **`struct`** for a way to decode C structures encoded as byte strings).

Availability: not WASI.

socket.getblocking()

Retorna `True` si el socket está en modo de bloqueo, `False` si está en sin bloqueo.

Esto es equivalente a comprobar `socket.gettimeout() != 0`.

Added in version 3.7.

socket.gettimeout()

Retorna el tiempo de espera en segundos (flotante) asociado con las operaciones del socket, o `None` si el tiempo de espera

no es seleccionado. Esto refleja la ultima llamada al `setblocking()` o `settimeout()`.

socket.ioctl(*control*, *option*)

El método `ioctl()` es una interfaz limitada para el sistema de interfaces WSAIoctl. Por favor refiérase a Win32 documentation [https://msdn.microsoft.com/en-us/library/ms741621%28VS.85%29.aspx] para mas información.

En otras plataformas, las funciones genéricas `fcntl.fcntl()` y `fcntl.ioctl()` podrían ser usadas; ellas aceptan un objeto socket como su primer argumento.

Actualmente solo el siguiente control de códigos está soportados: `SIO_RCVALL`, `SIO_KEEPALIVE_VALS`, y `SIO_LOOPBACK_FAST_PATH`.

Availability: Windows

Distinto en la versión 3.6: `SIO_LOOPBACK_FAST_PATH` ha sido agregado.

socket.listen([*backlog*])

Habilita un servidor para aceptar conexiones. Si *backlog* es especifico, debe ser al menos 0 (si es menor, se establece en 0); especifica el número de conexiones no aceptadas que permitirá el sistema antes de rechazar nuevas conexiones. Si no se especifica, se elige un valor razonable predeterminado.

Availability: not WASI.

Distinto en la versión 3.5: El parámetro *backlog* ahora es opcional.

socket.makefile(*mode='r'*, *buffering=None*, ***, *encoding=None*, *errors=None*, *newline=None*)

Return a file object associated with the socket. The exact returned type depends on the arguments given to `makefile()`. These arguments are interpreted the same way as by the built-in `open()` function, except the only supported *mode* values are `'r'` (default), `'w'`, `'b'`, or a combination of those.

El socket debe estar en modo de bloqueo; puede tener un tiempo de espera, pero el búfer interno del objeto de archivo puede terminar en un estado incoherente si se produce un tiempo de espera.

Cerrar el objeto de archivo devuelto por `makefile()` no cerrará el socket original a menos que se hayan cerrado todos los demás objetos de archivo y `socket.close()` se haya llamado al objeto socket.

Nota

En Windows, el objeto similar a un archivo creado por `makefile()` no se puede utilizar cuando se espera un objeto de archivo con un descriptor de archivo, como los argumentos de secuencia de `subprocess.Popen()`.

socket.recv(*bufsize*[, *flags*])

Receive data from the socket. The return value is a bytes object representing the data received. The maximum amount of data to be received at once is specified by *bufsize*. A returned empty bytes object indicates that the client has disconnected. See the Unix manual page *recv(2)* *[https://manpages.debian.org/recv(2)]* for the meaning of the optional argument *flags*; it defaults to zero.

Distinto en la versión 3.5: Si se interrumpe la llamada del sistema y el controlador de señal no genera una excepción, el método ahora vuelve a intentar la llamada del sistema en lugar de generar una excepción `InterruptedError` (consulte **PEP 475** [https://peps.python.org/pep-0475/] para la lógica).

socket.recvfrom(*bufsize*[, *flags*])

Receive data from the socket. The return value is a pair `(bytes, address)` where *bytes* is a bytes object representing the data received and *address* is the address of the socket sending the data. See the Unix manual page *recv(2)* *[https://manpages.debian.org/recv(2)]* for the meaning of the optional argument *flags*; it defaults to zero. The format of *address* depends on the address family — see Familias Socket.

Distinto en la versión 3.5: Si se interrumpe la llamada del sistema y el controlador de señal no genera una excepción, el método ahora vuelve a intentar la llamada del sistema en lugar de generar una excepción **`InterruptedError`** (consulte **PEP 475** [https://peps.python.org/pep-0475/] para la lógica).

Distinto en la versión 3.7: Para direcciones IPv6 de multidifusión, el primer elemento de *address* ya no contiene la parte `%scope_id`. Para obtener la dirección IPV6 completa, use **`getnameinfo()`**.

socket.recvmsg(*bufsize*[, *ancbufsize*[, *flags*]])

Reciba datos normales (hasta *bufsize* bytes) y datos auxiliares del socket. El argumento *ancbufsize* establece el tamaño en bytes del búfer interno utilizado para recibir los datos auxiliares; el valor predeterminado es 0, lo que significa que no se recibirán datos auxiliares. Los tamaños de búfer adecuados para los datos auxiliares se pueden calcular mediante **`CMSG_SPACE()`** o **`CMSG_LEN()`**, y los elementos que no caben en el búfer pueden truncarse o descartarse. El valor predeterminado del argumento *flags* es 0 y tiene el mismo significado que para **`recv()`**.

El valor de retorno es una tupla de 4: `(data, ancdata, msg_flags, address)`. El valor *data* es un objeto **`bytes`** que contiene los datos no auxiliares recibidos. El valor *ancdata* es una lista de cero o mas tuplas `(cmsg_level, cmsg_type, cmsg_data)` representado los datos auxiliares (control de mensajes) recibidos: *cmsg_level* y *cmsg_type* son enteros especificando el nivel de protocolo y tipo específico de protocolo respectivamente, y *cmsg_data* es un objeto

`bytes` sosteniendo los datos asociados. El valor *msg_flags* es el OR bit a bit de varios indicadores que indican condiciones en el mensaje recibido; consulte la documentación de su sistema para obtener más detalles. Si la toma de recepción no está conectada, *address* es la dirección de el socket enviado, si está disponible; de lo contrario, su valor no se especifica.

En algunos sistemas, **`sendmsg()`** y **`recvmsg()`** pueden utilizarse para pasar descriptores de fichero entre procesos a través de un socket **`AF_UNIX`**. Cuando se utiliza esta funcionalidad (a menudo está restringida a los sockets **`SOCK_STREAM`**), **`recvmsg()`** devolverá, en sus datos auxiliares, elementos del formulario `(socket.SOL_SOCKET, socket.SCM_RIGHTS, fds)`, donde *fds* es un objeto **`bytes`** que representa los nuevos descriptores de archivo como una arreglo binario del tipo nativo C int. Si **`recvmsg()`** lanza una excepción después del retorno de la llamada al sistema, primero intentará cerrar cualquier descriptor de fichero recibido a través de este mecanismo.

Algunos sistemas no indican la longitud truncada de los elementos de datos auxiliares que solo se han recibido parcialmente. Si un elemento parece extenderse más allá del final del búfer, **`recvmsg()`** emitirá un **`RuntimeWarning`**, y devolverá la parte de él que está dentro del búfer siempre que no se haya truncado antes del inicio de sus datos asociados.

En sistemas donde soporta el mecanismo **`SCM_RIGHTS`**, la siguiente función recibirá un descriptor de archivos *maxfds*, devolviendo el mensaje de datos y un lista que contiene los descriptores (mientras se ignoran las condiciones inesperadas, como la recepción de mensajes de control no relacionados). Ver también **`sendmsg()`**.

```
import socket, array

def recv_fds(sock, msglen, maxfds):
    fds = array.array("i")   # Array of ints
    msg, ancdata, flags, addr = sock.recvmsg(msglen
    for cmsg_level, cmsg_type, cmsg_data in ancdata
        if cmsg_level == socket.SOL_SOCKET and cmsg
            # Append data, ignoring any truncated i
            fds.frombytes(cmsg_data[:len(cmsg_data)
    return msg, list(fds)
```

Availability: Unix.

La mayoría de las plataformas Unix.

Added in version 3.3.

Distinto en la versión 3.5: Si se interrumpe la llamada del sistema y el controlador de señal no genera una excepción, el método ahora vuelve a intentar la llamada del sistema en lugar de generar una excepción **`InterruptedError`** (consulte **PEP 475** [https://peps.python.org/pep-0475/] para la lógica).

socket.recvmsg_into(*buffers*[, *ancbufsize*[, *flags*]])

Recibir datos normales y datos auxiliares desde el socket, comportándose como **`recvmsg()`** lo haría, pero dispersar los datos no auxiliares en una serie de buffers en lugar de devolver un nuevo objeto bytes. El argumento *buffers* debe ser un iterable de objetos que exportan buffers de escritura (por ejemplo, objetos **`bytearray`**); estos se llenarán con fragmentos sucesivos de los datos no auxiliares hasta que se hayan escrito todos o no haya más buffers. El sistema operativo puede establecer un límite (**`sysconf()`** valor `SC_IOV_MAX`) en el número de buffers que se pueden utilizar. Los argumentos *ancbufsize* y *flags* tienen el mismo significado que para **`recvmsg()`**.

El valor de retorno es tupla de 4: `(nbytes, ancdata, msg_flags, address)`, donde *nbytes* es el numero total de bytes de datos no auxiliares escrito dentro de los bufetes, y *ancdata*, *msg_flags* y *address* son lo mismo que para **`recvmsg()`**.

Ejemplo:

```
>>> import socket
>>> s1, s2 = socket.socketpair()
>>> b1 = bytearray(b'----')
>>> b2 = bytearray(b'0123456789')
>>> b3 = bytearray(b'--------------')
>>> s1.send(b'Mary had a little lamb')
22
>>> s2.recvmsg_into([b1, memoryview(b2)[2:9], b3])
(22, [], 0, None)
>>> [b1, b2, b3]
[bytearray(b'Mary'), bytearray(b'01 had a 9'), byte
```

Availability: Unix.

La mayoría de las plataformas Unix.

Added in version 3.3.

socket.recvfrom_into(*buffer*[, *nbytes*[, *flags*]])

Receive data from the socket, writing it into *buffer* instead of creating a new bytestring. The return value is a pair `(nbytes, address)` where *nbytes* is the number of bytes received and *address* is the address of the socket sending the data. See the Unix manual page *recv(2)* *[https://manpages.debian.org/recv(2)]* for the meaning of the optional argument *flags*; it defaults to zero. The format of *address* depends on the address family — see Familias Socket.

socket.recv_into(*buffer*[, *nbytes*[, *flags*]])

Recibe hasta *nbytes* bytes desde el socket, almacenado los datos en un búfer en lugar de crear una nueva cadena de bytes. Si *nbytes* no esta especificado (o 0), recibir hasta el tamaño disponible en el búfer dado. Retorna el número de bytes recibidos. Ver la página del manual de Unix *recv(2)* *[https://manpages.debian.org/recv(2)]* para el significado del argumento opcional *flags*; por defecto es cero.

socket.send(*bytes*[, *flags*])

Send data to the socket. The socket must be connected to a remote socket. The optional *flags* argument has the same meaning as for **`recv()`**. Returns the number of bytes sent. Applications are responsible for checking that all data has been sent; if only some of the data was transmitted, the application needs to attempt delivery of the remaining data. For further information on this topic, consult the HOW TO - Programación con sockets.

Distinto en la versión 3.5: Si se interrumpe la llamada del sistema y el controlador de señal no genera una excepción, el método ahora vuelve a intentar la llamada del sistema en lugar de generar una excepción **`InterruptedError`** (consulte **PEP 475** [https://peps.python.org/pep-0475/] para la lógica).

socket.sendall(*bytes*[, *flags*])

Send data to the socket. The socket must be connected to a remote socket. The optional *flags* argument has the same meaning as for **`recv()`**. Unlike **`send()`**, this method continues to send data from *bytes* until either all data has been sent or an error occurs. `None` is returned on success. On error, an exception is raised, and there is no way to determine how much data, if any, was successfully sent.

Distinto en la versión 3.5: El tiempo de espera del socket ya no se restablece cada vez que los datos se envían correctamente. El tiempo de espera del socket es ahora la duración total máxima para enviar todos los datos.

Distinto en la versión 3.5: Si se interrumpe la llamada del sistema y el controlador de señal no genera una excepción, el método ahora vuelve a intentar la llamada del sistema en lugar de generar una excepción **`InterruptedError`** (consulte **PEP 475** [https://peps.python.org/pep-0475/] para la lógica).

socket.sendto(*bytes*, *address*)
socket.sendto(*bytes*, *flags*, *address*)

Send data to the socket. The socket should not be connected to a remote socket, since the destination socket is specified by *address*. The optional *flags* argument has the same meaning as for **`recv()`**. Return the number of bytes sent. The format of *address* depends on the address family — see Familias Socket.

Genera un auditing event `socket.sendto` con los argumentos `self`, `address`.

Distinto en la versión 3.5: Si se interrumpe la llamada del sistema y el controlador de señal no genera una excepción, el método ahora vuelve a intentar la llamada del sistema en lugar de generar una excepción **`InterruptedError`** (consulte **PEP 475** [https://peps.python.org/pep-0475/] para la lógica).

socket.sendmsg(*buffers*[, *ancdata*[, *flags*[, *address*]]])

Enviar datos normales y auxiliares al socket, recopilar los datos no auxiliares de una serie de buffers y concatenando en un único mensaje. El argumento *buffers* especifica los datos no auxiliares como un iterable de bytes-como objetos (por ejemplo: objetos **`bytes`**); el sistema operativo puede establecer un límite (**`os.sysconf()`** valor `SC_IOV_MAX`) en el número de buffers que se pueden utilizar. El argumento *ancdata* especifica los datos auxiliares (mensajes de control) como un iterable de cero o más tuplas `(cmsg_level, cmsg_type, cmsg_data)`, donde *cmsg_level* y *cmsg_type* son enteros que especifican el nivel de protocolo y el tipo específico del protocolo respectivamente, y *cmsg_data* es un objeto similar a bytes que contiene los datos asociados. Tenga en cuenta que algunos sistemas (en particular, sistemas sin **`CMSG_SPACE()`**) pueden admitir el envío de solo un mensaje de control por llamada. El valor predeterminado del argumento *flags* es 0 y tiene el mismo significado que para **`send()`**. Si se proporciona *address* y no `None`, establece una dirección de destino para el mensaje. El valor devuelto es el número de bytes de datos no auxiliares enviados.

La siguiente función envía la lista de descriptores de archivos *fds* sobre un socket **`AF_UNIX`**, estos sistemas pueden soportar la mecánica **`SCM_RIGHTS`**. Observar también **`recvmsg()`**.

```
import socket, array

def send_fds(sock, msg, fds):
    return sock.sendmsg([msg], [(socket.SOL_SOCKET,
```

Availability: Unix, not WASI.

La mayoría de las plataformas Unix.

Genera un auditing event `socket.sendmsg` con los argumentos `self`, `address`.

Added in version 3.3.

Distinto en la versión 3.5: Si se interrumpe la llamada del sistema y el controlador de señal no genera una excepción, el método ahora vuelve a intentar la llamada del sistema en lugar de generar una excepción **`InterruptedError`** (consulte **PEP 475** [https://peps.python.org/pep-0475/] para la lógica).

socket.sendmsg_afalg([*msg*,]*, *op*[, *iv*[, *assoclen*[, *flags*]]])

Versión especializada de **`sendmsg()`** para el socket **`AF_ALG`**. Modo de ajuste, IV, longitud de datos asociados a AEAD y banderas para el socket **`AF_ALG`**.

Availability: Linux > = 2.6.38.

Added in version 3.6.

socket.sendfile(*file*, *offset=0*, *count=None*)

Enviar un archivo hasta que se alcance EOF mediante el uso de alto rendimiento **`os.sendfile`** y devolver el número total de bytes que se enviaron. *file* debe ser un objeto de archivo normal abierto en modo binario. Si **`os.sendfile`** no está disponible (por ejemplo, Windows) o *file* no es un archivo normal **`send()`** se utilizará en su lugar. *offset* indica desde dónde empezar a leer el archivo. Si se especifica, *count* es el número total de bytes para transmitir en lugar de enviar el archivo hasta que se alcance EOF. La posición del archivo se actualiza a la vuelta o también en caso de error en cuyo caso **`file.tell()`** se puede utilizar para averiguar el número de bytes que se enviaron. El socket debe ser de tipo **`SOCK_STREAM`** No se admiten sockets sin bloqueo.

Added in version 3.5.

socket.set_inheritable(*inheritable*)

Selecciona el inheritable flag descriptor del archivo del socket o el controlador del socket.

Added in version 3.4.

socket.setblocking(*flag*)

Establecer el modo de bloqueo o no bloqueo del socket: si *flag* es false, el socket se establece en modo sin bloqueo, de lo contrario en modo de bloqueo.

El método es una abreviatura para ciertas llamadas **`settimeout()`**:

- `sock.setblocking(True)` es equivalente a `sock.settimeout(None)`
- `sock.setblocking(False)` es equivalente a `sock.settimeout(0.0)`

Distinto en la versión 3.7: El método ya no aplica la bandera **`SOCK_NONBLOCK`** en **`socket.type`**.

socket.settimeout(*value*)

Set a timeout on blocking socket operations. The *value* argument can be a nonnegative floating-point number expressing seconds, or `None`. If a non-zero value is given, subsequent socket operations will raise a **`timeout`** exception if the timeout period *value* has elapsed before the operation has completed. If zero is given, the socket is put in non-blocking mode. If `None` is given, the socket is put in blocking mode.

Para obtener más información, consulte las notas notas sobre los tiempos de espera del socket.

Distinto en la versión 3.7: El método ya no cambia la bandera **`SOCK_NONBLOCK`** en **`socket.type`**.

socket.setsockopt(*level*, *optname*, *value: int | Buffer*)
socket.setsockopt(*level*, *optname*, *None*, *optlen: int*)

Set the value of the given socket option (see the Unix manual page *setsockopt(2)* *[https://manpages.debian.org/setsockopt(2)]*). The needed symbolic constants are defined in this module (SO_* etc. <socket-unix-constants>). The value can be an integer, `None` or a bytes-like object representing a buffer. In the latter case it is up to the caller to ensure that the bytestring contains the proper bits (see the optional built-in module **`struct`** for a way to encode C structures as bytestrings). When *value* is set to `None`, *optlen* argument is required. It's equivalent to calling **`setsockopt()`** C function with `optval=NULL` and `optlen=optlen`.

Distinto en la versión 3.5: Ahora se acepta la grabación bytes-like object.

Distinto en la versión 3.6: setsockopt(level, optname, None, optlen: int) form added.

Availability: not WASI.

socket.shutdown(*how*)

Apague una o ambas mitades de la conexión. Si *how* es **`SHUT_RD`**, más recibe no se permiten. Si *how* es **`SHUT_WR`**, mas recibe no se permiten. Si *how* es **`SHUT_RDWR`**, más recibe no se permiten.

Availability: not WASI.

socket.share(*process_id*)

Duplica un socket y lo prepara para compartirlo con el proceso de destino. El proceso de destino debe estar provisto de *process_id*. el objeto de bytes resultante luego se puede pasar al proceso de destino usando alguna forma de comunicación entre procesos y el socket se puede recrear allí usando `fromshare()`. Una vez que se ha llamado a este método, es seguro cerrar el socket ya que el sistema operativo ya lo ha duplicado para el proceso de destino.

Availability: Windows.

Added in version 3.3.

Tenga en cuenta que no hay métodos `read()` o `write()`; use `recv()` y `send()` sin el argumento *flags* en su lugar.

Los objetos de socket también tienen estos atributos (de solo lectura) que corresponden a los valores dados al constructor `socket`.

socket.family

La familia socket.

socket.type

El tipo de socket.

socket.proto

The socket protocol.

Notas sobre los tiempos de espera del socket

Un objeto de socket puede estar en uno de los tres modos: bloqueo, no bloqueo o tiempo de espera. Los sockets se crean de forma predeterminada siempre en modo de bloqueo, pero esto se puede cambiar llamando a `setdefaulttimeout()`.

- En el modo *bloqueo*, las operaciones se bloquean hasta que se completan o el sistema devuelve un error (como tiempo de espera de conexión agotado).

- En *modo sin bloqueo*, las operaciones fallan (con un error que desafortunadamente depende del sistema) si no se pueden completar inmediatamente: las funciones del módulo `select` se pueden usar para saber cuándo y si un socket está disponible para leer o escribir.
- En *timeout mode*, las operaciones fallan si no se puede completan el tiempo de espera especifico para el socket (ellos lanzan una excepción `timeout`) o si el sistema devuelve un error.

Nota

At the operating system level, sockets in *timeout mode* are internally set in non-blocking mode. Also, the blocking and timeout modes are shared between file descriptors and socket objects that refer to the same network endpoint. This implementation detail can have visible consequences if e.g. you decide to use the `fileno()` of a socket.

Tiempos de espera y el método `connect`

La operación `connect()` también está sujeta a la configuración de tiempo de espera, y en general se recomienda llamar a `settimeout()` antes de llamar a `connect()` o pasar un parámetro de tiempo de espera a `create_connection()`. Sin embargo, la pila de red del sistema también puede devolver un error de tiempo de espera de conexión propio independientemente de cualquier configuración de tiempo de espera del socket de Python.

Tiempos de espera y el método `accept`

Si `getdefaulttimeout()` no es una `None`, los sockets devuelto por el método `accept()` heredan ese tiempo de espera. De lo contrario, el comportamiento depende de la configuración de la toma de escucha:

- si los sockets están escuchando en *blocking mode* o en el *timeout mode*, el socket devuelve el `accept()` en un *blocking mode*;

- si los sockets están escuchando en *non-blocking mode*, ya sea el socket devuelto por `accept()` es un modo de bloqueo o no bloque depende del sistema operativo. Si desea garantizar un comportamiento multiplataforma, se recomienda que se anule manualmente esta configuración.

Ejemplo

Here are four minimal example programs using the TCP/IP protocol: a server that echoes all data that it receives back (servicing only one client), and a client using it. Note that a server must perform the sequence `socket()`, `bind()`, `listen()`, `accept()` (possibly repeating the `accept()` to service more than one client), while a client only needs the sequence `socket()`, `connect()`. Also note that the server does not `sendall()`/`recv()` on the socket it is listening on but on the new socket returned by `accept()`.

Los dos primeros ejemplos solo admiten IPv4.

```
# Echo server program
import socket

HOST = ''                 # Symbolic name meaning all av
PORT = 50007              # Arbitrary non-privileged por
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) a
    s.bind((HOST, PORT))
    s.listen(1)
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data: break
            conn.sendall(data)

# Echo client program
```

```
import socket

HOST = 'daring.cwi.nl'    # The remote host
PORT = 50007              # The same port as used by the
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) a
    s.connect((HOST, PORT))
    s.sendall(b'Hello, world')
    data = s.recv(1024)
print('Received', repr(data))
```

The next two examples are identical to the above two, but support both IPv4 and IPv6. The server side will listen to the first address family available (it should listen to both instead). On most of IPv6-ready systems, IPv6 will take precedence and the server may not accept IPv4 traffic. The client side will try to connect to all the addresses returned as a result of the name resolution, and sends traffic to the first one connected successfully.

```
# Echo server program
import socket
import sys

HOST = None               # Symbolic name meaning all av
PORT = 50007              # Arbitrary non-privileged por
s = None
for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSP
                              socket.SOCK_STREAM, 0, soc
    af, socktype, proto, canonname, sa = res
    try:
        s = socket.socket(af, socktype, proto)
    except OSError as msg:
        s = None
        continue
    try:
        s.bind(sa)
        s.listen(1)
    except OSError as msg:
        s.close()
        s = None
        continue
    break
if s is None:
    print('could not open socket')
    sys.exit(1)
conn, addr = s.accept()
with conn:
    print('Connected by', addr)
    while True:
        data = conn.recv(1024)
        if not data: break
        conn.send(data)
```

```
# Echo client program
import socket
import sys

HOST = 'daring.cwi.nl'    # The remote host
PORT = 50007              # The same port as used by the
s = None
for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSP
    af, socktype, proto, canonname, sa = res
    try:
        s = socket.socket(af, socktype, proto)
    except OSError as msg:
        s = None
        continue
    try:
        s.connect(sa)
    except OSError as msg:
        s.close()
        s = None
        continue
    break
if s is None:
    print('could not open socket')
    sys.exit(1)
with s:
    s.sendall(b'Hello, world')
    data = s.recv(1024)
print('Received', repr(data))
```

El siguiente ejemplo muestra cómo escribir un rastreador de red muy simple con sockets sin procesar en Windows. El ejemplo requiere privilegios de administrador para modificar la interfaz:

```
import socket

# the public network interface
HOST = socket.gethostbyname(socket.gethostname())

# create a raw socket and bind it to the public interfac
s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socke
s.bind((HOST, 0))

# Include IP headers
s.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

# receive all packets
s.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)

# receive a packet
print(s.recvfrom(65565))

# disabled promiscuous mode
s.ioctl(socket.SIO_RCVALL, socket.RCVALL_OFF)
```

En el ejemplo siguiente se muestra cómo utilizar la interfaz de socket para comunicarse con una red CAN mediante el protocolo de socket sin procesar. Para utilizar CAN con el protocolo de gestor de difusión en su lugar, abra un socket con:

```
socket.socket(socket.AF_CAN, socket.SOCK_DGRAM, socket.C
```

Después de vincular (**`CAN_RAW`**) o conectar (**`CAN_BCM`**) el socket, puede usar las operaciones **`socket.send()`** y **`socket.recv()`** (y sus contrapartes) en el objeto socket como de costumbre.

Este último ejemplo puede requerir privilegios especiales:

```
import socket
import struct


# CAN frame packing/unpacking (see 'struct can_frame' ir

can_frame_fmt = "=IB3x8s"
can_frame_size = struct.calcsize(can_frame_fmt)

def build_can_frame(can_id, data):
    can_dlc = len(data)
    data = data.ljust(8, b'\x00')
    return struct.pack(can_frame_fmt, can_id, can_dlc, c

def dissect_can_frame(frame):
    can_id, can_dlc, data = struct.unpack(can_frame_fmt,
    return (can_id, can_dlc, data[:can_dlc])


# create a raw socket and bind it to the 'vcan0' interfa
s = socket.socket(socket.AF_CAN, socket.SOCK_RAW, socket
s.bind(('vcan0',))

while True:
    cf, addr = s.recvfrom(can_frame_size)

    print('Received: can_id=%x, can_dlc=%x, data=%s' % c

    try:
        s.send(cf)
    except OSError:
        print('Error sending CAN frame')

    try:
        s.send(build_can_frame(0x01, b'\x01\x02\x03'))
    except OSError:
        print('Error sending CAN frame')
```

Ejecutar un ejemplo varias veces con un retraso demasiado pequeño entre ejecuciones, podría dar lugar a este error:

```
OSError: [Errno 98] Address already in use
```

Esto se debe a que la ejecución anterior ha dejado el socket en un estado `TIME_WAIT` y no se puede reutilizar inmediatamente.

There is a **`socket`** flag to set, in order to prevent this, **`socket.SO_REUSEADDR`**:

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT))
```

el indicador **`SO_REUSEADDR`** indica al kernel que reutilice un socket local en estado `TIME_WAIT`, sin esperar a que expire su tiempo de espera natural.

Ver también

Para obtener una introducción a la programación de sockets (en C), consulte los siguientes documentos:

- *An Introductory 4.3BSD Interprocess Communication Tutorial*, por Stuart Sechrest
- *An Advanced 4.3BSD Interprocess Communication Tutorial*, por Samuel J. Leffler et al,

ambos en el manual del programador de Unix, documentos suplementarios 1 (secciones PS1:7 y PS1:8). La plataforma especifica material de referencia para las diversas llamadas al sistema también son una valiosa fuente de información en los detalles de la semántica del socket. Para Unix, referencia a las paginas del manual, para Windows, observa la especificación WinSock (o WinSock 2). Para APIS listas IPV6, los lectores pueden querer referirse al titulado Extensiones básicas de interfaz de socket para IPv6 **RFC 3493** [https://datatracker.ietf.org/doc/html/rfc3493.html] .

`ssl` — TLS/SSL wrapper for socket objects

Código fuente: Lib/ssl.py [https://github.com/python/cpython/tree/3.14/Lib/ssl.py]

This module provides access to Transport Layer Security (often known as «Secure Sockets Layer») encryption and peer authentication facilities for network sockets, both client-side and server-side. This module uses the OpenSSL library.

This is an optional module. If it is missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

Nota

Some behavior may be platform dependent, since calls are made to the operating system socket APIs. The installed version of OpenSSL may also cause variations in behavior. For example, TLSv1.3 comes with OpenSSL version 1.1.1.

Advertencia

No use este módulo sin leer Consideraciones de seguridad. Hacerlo puede generar una falsa sensación de seguridad, ya que la configuración predeterminada del módulo ssl no es necesariamente la adecuada para su aplicación.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See

Plataformas WebAssembly for more information.

Esta sección documenta los objetos y funciones en el módulo `ssl`; para más información general sobre TLS,SSL, y certificados, se recomienda que el lector acuda a la sección «Ver también» al final de la página.

This module provides a class, **`ssl.SSLSocket`**, which is derived from the **`socket.socket`** type, and provides a socket-like wrapper that also encrypts and decrypts the data going over the socket with SSL. It supports additional methods such as **`getpeercert()`**, which retrieves the certificate of the other side of the connection, **`cipher()`**, which retrieves the cipher being used for the secure connection or **`get_verified_chain()`**, **`get_unverified_chain()`** which retrieves certificate chain.

Para aplicaciones más sofisticadas, la clase **`ssl.SSLContext`** ayuda a administrar la configuración y los certificados, que luego pueden ser heredados por sockets SSL creados a través del método **`SSLContext.wrap_socket()`**.

Distinto en la versión 3.5.3: Actualizado para admitir la vinculación con OpenSSL 1.1.0

Distinto en la versión 3.6: OpenSSL 0.9.8, 1.0.0 y 1.0.1 son obsoletos y no son compatibles. En el futuro, el módulo ssl requerirá al menos OpenSSL 1.0.2 o 1.1.0.

Distinto en la versión 3.10: Se ha implementado **PEP 644** [https://peps.python.org/pep-0644/]. El módulo ssl requiere OpenSSL 1.1.1 o versiones más recientes.

El uso de constantes y funciones obsoletas genera advertencias de obsolescencia.

Functions, constants, and exceptions

Creación de sockets

Instances of **`SSLSocket`** must be created using the

`SSLContext.wrap_socket()` method. The helper function **`create_default_context()`** returns a new context with secure default settings.

Ejemplo de socket cliente con contexto por defecto y doble pila IPv4/IPv6:

```
import socket
import ssl

hostname = 'www.python.org'
context = ssl.create_default_context()

with socket.create_connection((hostname, 443)) as sock:
    with context.wrap_socket(sock, server_hostname=hostn
        print(ssock.version())
```

Ejemplo de socket cliente con contexto personalizado y IPv4:

```
hostname = 'www.python.org'
# PROTOCOL_TLS_CLIENT requires valid cert chain and host
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
context.load_verify_locations('path/to/cabundle.pem')

with socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0
    with context.wrap_socket(sock, server_hostname=hostn
        print(ssock.version())
```

Ejemplo de socket servidor escuchando en localhost IPv4:

```
context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.load_cert_chain('/path/to/certchain.pem', '/path

with socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0
    sock.bind(('127.0.0.1', 8443))
    sock.listen(5)
    with context.wrap_socket(sock, server_side=True) as
        conn, addr = ssock.accept()
        ...
```

Creación de contexto

Una función conveniente ayuda a crear objetos `SSLContext` para propósitos comunes.

ssl.create_default_context(*purpose=Purpose.SERVER_AUTH*, *, *cafile=None*, *capath=None*, *cadata=None*)

Return a new `SSLContext` object with default settings for the given *purpose*. The settings are chosen by the **`ssl`** module, and usually represent a higher security level than when calling the **`SSLContext`** constructor directly.

cafile, *capath*, *cadata* representan certificados CA opcionales para confiar en la verificación de certificados, como en `SSLContext.load_verify_locations()`. Si los tres son `None` al mismo tiempo, esta función puede optar por confiar en su lugar en los certificados CA por defecto del sistema.

The settings are: `PROTOCOL_TLS_CLIENT` or `PROTOCOL_TLS_SERVER`, `OP_NO_SSLv2`, and `OP_NO_SSLv3` with high encryption cipher suites without RC4 and without unauthenticated cipher suites. Passing `SERVER_AUTH` as *purpose* sets `verify_mode` to `CERT_REQUIRED` and either loads CA certificates (when at least one of *cafile*, *capath* or *cadata* is given) or uses `SSLContext.load_default_certs()` to load default CA certificates.

Cuando `keylog_filename` es soportado y la variable de entorno **`SSLKEYLOGFILE`** está establecida, `create_default_context()` activa el registro de claves.

The default settings for this context include `VERIFY_X509_PARTIAL_CHAIN` and `VERIFY_X509_STRICT`. These make the underlying OpenSSL implementation behave more like a conforming implementation of RFC 5280 [https://datatracker.ietf.org/doc/html/rfc5280.html], in exchange for a small amount of incompatibility with older X.509 certificates.

Nota

El protocolo, las opciones, el cifrado y otros ajustes pueden cambiar a valores mas restrictivos en cualquier momento sin previa obsolescencia. Los valores representan un equilibrio justo entre compatibilidad y seguridad.

Si su aplicación necesita ajustes específicos, debe crear un `SSLContext` y aplicar los ajustes usted mismo.

Nota

Si encuentra que cuando ciertos clientes o servidores antiguos intentan conectarse con un `SSLContext` creado con esta función obtienen un error indicando *Protocol or cipher suite mismatch*, puede ser que estos sólo soportan SSL3.0 el cual esta función excluye utilizando `OP_NO_SSLv3`. SSL3.0 está ampliamente considerado como completamente roto [https://en.wikipedia.org/wiki/POODLE]. Si todavía desea seguir utilizando esta función pero permitir conexiones SSL 3.0, puede volver a activarlas mediante:

```
ctx = ssl.create_default_context(Purpose.CLIENT_AU
ctx.options &= ~ssl.OP_NO_SSLv3
```

Nota

This context enables `VERIFY_X509_STRICT` by default, which may reject pre-**RFC 5280** [https://datatracker.ietf.org/doc/html/rfc5280.html] or malformed certificates that the underlying OpenSSL implementation otherwise would accept. While disabling this is not recommended, you can do so using:

```
ctx = ssl.create_default_context()
ctx.verify_flags &= ~ssl.VERIFY_X509_STRICT
```

Added in version 3.4.

Distinto en la versión 3.4.4: RC4 ha sido abandonado de la cadena de cifrado por defecto.

Distinto en la versión 3.6: ChaCha20/Poly1305 ha sido agregado a la cadena de caracteres de cifrado por defecto.

3DES ha sido abandonado de la cadena de caracteres de cifrado por defecto.

Distinto en la versión 3.8: Soporte del registro de claves en **`SSLKEYLOGFILE`** ha sido agregado.

Distinto en la versión 3.10: El contexto ahora usa el protocolo `PROTOCOL_TLS_CLIENT` o `PROTOCOL_TLS_SERVER` en lugar del `PROTOCOL_TLS` genérico.

Distinto en la versión 3.13: The context now uses `VERIFY_X509_PARTIAL_CHAIN` and `VERIFY_X509_STRICT` in its default verify flags.

Excepciones

exception ssl.SSLError

Se lanza para señalar un error de la implementación de SSL subyacente (actualmente proporcionada por la biblioteca OpenSSL). Esto indica algún problema en la capa de cifrado y autenticación de alto nivel que se superpone a la conexión de red subyacente. Este error es un subtipo de `OSError`. El código de error y el mensaje de las instancias de `SSLError` son proporcionados por la biblioteca OpenSSL.

Distinto en la versión 3.3: `SSLError` era un subtipo de `socket.error`.

library

Una cadena de caracteres mnemotécnica que designa el submódulo de OpenSSL en el que se ha producido el error, como `SSL`, `PEM` o `X509`. El rango de valores posibles depende de la versión de OpenSSL.

Added in version 3.3.

reason

Una cadena de caracteres mnemotécnica que designa la razón por la que se produjo el error, por ejemplo `CERTIFICATE_VERIFY_FAILED`. El rango de valores posibles depende de la versión de OpenSSL.

Added in version 3.3.

exception ssl.SSLZeroReturnError

Una subclase de **`SSLError`** lanzada cuando se intenta leer o escribir y la conexión SSL ha sido cerrada limpiamente. Tenga en cuenta que esto no significa que el transporte subyacente (lectura TCP) haya sido cerrado.

Added in version 3.3.

exception ssl.SSLWantReadError

Una subclase de **`SSLError`** lanzada por un socket SSL no bloqueante cuando se intenta leer o escribir datos, pero mas datos necesitan ser recibidos en el transporte TCP subyacente antes de que la solicitud pueda ser completada.

Added in version 3.3.

exception ssl.SSLWantWriteError

Una subclase de **`SSLError`** lanzada por un socket SSL no bloqueante cuando se intenta leer o escribir datos, pero mas datos necesitan ser enviados en el transporte TCP subyacente antes de que la solicitud pueda ser completada.

Added in version 3.3.

exception ssl.SSLSyscallError

Una subclase de **`SSLError`** lanzada cuando se encuentra un error del sistema mientras se intenta completar una operación en un socket SSL. Por desgracia, no hay una manera fácil de inspeccionar el número errno original.

Added in version 3.3.

exception ssl.SSLEOFError

Una subclase de `SSLError` lanzada cuando la conexión SSL ha sido cancelada abruptamente. Generalmente, no debería intentar reutilizar el transporte subyacente cuando este error se produce.

Added in version 3.3.

exception ssl.SSLCertVerificationError

Una subclase de `SSLError` lanzada cuando la validación del certificado ha fallado.

Added in version 3.7.

verify_code

Un número de error numérico que indica el error de verificación.

verify_message

Una cadena de caracteres legible del error de verificación.

exception ssl.CertificateError

Un alias para `SSLCertVerificationError`.

Distinto en la versión 3.7: La excepción es ahora un alias para `SSLCertVerificationError`.

Generación aleatoria

ssl.RAND_bytes(*num*, */*)

Retorna *num* bytes pseudoaleatorios criptográficamente fuertes. Lanza un `SSLError` si el PRNG no a sido sembrado con suficiente datos o si la operación no es soportada por el método RAND actual. `RAND_status()` puede ser usada para verificar el estado de PRNG y `RAND_add()` puede ser usada para sembrar el PRNG.

Para casi todas las aplicaciones `os.urandom()` es preferible.

Léase el artículo Wikipedia, Generador de números pseudoaleatorios criptográficamente seguro (CSPRNG) [https://es.wikipedia.org/wiki/Generador_de_n%C3%BAmeros_pseudoaleatorios_criptogr%C3%A1ficamente_seguro], para obtener los requisitos para un generador criptográficamente seguro.

Added in version 3.3.

ssl.RAND_status()

Retorna `True` si el generador de números pseudoaleatorios SSL a sido sembrado con "suficiente" aleatoriedad, y `False` de lo contrario. Puede utilizarse **`ssl.RAND_egd()`** y **`ssl.RAND_add()`** para aumentar la aleatoriedad del generador de números pseudoaleatorios.

ssl.RAND_add(*bytes*, *entropy*, */*)

Mix the given *bytes* into the SSL pseudo-random number generator. The parameter *entropy* (a float) is a lower bound on the entropy contained in string (so you can always use `0.0`). See **RFC 1750** [https://datatracker.ietf.org/doc/html/rfc1750.html] for more information on sources of entropy.

Distinto en la versión 3.5: Ahora se acepta bytes-like object modificable.

Gestión de certificados

ssl.cert_time_to_seconds(*cert_time*)

Return the time in seconds since the epoch, given the `cert_time` string representing the «notBefore» or «notAfter» date from a certificate in `"%b %d %H:%M:%S %Y %Z"` strptime format (C locale).

He aquí un ejemplo:

```
>>> import ssl
>>> import datetime as dt
>>> timestamp = ssl.cert_time_to_seconds("Jan  5 09
>>> timestamp
1515144883
>>> print(dt.datetime.fromtimestamp(timestamp, dt.U
2018-01-05 09:34:43+00:00
```

Las fechas *notBefore* o *notAfter* deben utilizar GMT (**RFC 5280** [https://datatracker.ietf.org/doc/html/rfc5280.html]).

Distinto en la versión 3.5: Interpreta la hora de entrada como una hora en UTC según lo especificado por la zona horaria "GMT" en la cadena de caracteres de entrada. Anteriormente se utilizaba la zona horaria local. Retorna un número entero (sin fracciones de segundo en el formato de entrada)

ssl.get_server_certificate(*addr*, *ssl_version=PROTOCOL_TLS_CLIENT*, *ca_certs=None*[, *timeout*])

Given the address `addr` of an SSL-protected server, as a (*hostname*, *port-number*) pair, fetches the server's certificate, and returns it as a PEM-encoded string. If `ssl_version` is specified, uses that version of the SSL protocol to attempt to connect to the server. If *ca_certs* is specified, it should be a file containing a list of root certificates, the same format as used for the *cafile* parameter in **`SSLContext.load_verify_locations()`**. The call will attempt to validate the server certificate against that set of root certificates, and will fail if the validation attempt fails. A timeout can be specified with the `timeout` parameter.

Distinto en la versión 3.3: Esta función es ahora compatible IPv6.

Distinto en la versión 3.5: La *ssl_version* por defecto se cambia de **`PROTOCOL_SSLv3`** a **`PROTOCOL_TLS`** para una máxima compatibilidad con los servidores modernos.

Distinto en la versión 3.10: Se agregó el argumento *session*.

ssl.DER_cert_to_PEM_cert(*der_cert_bytes*)

Dado un certificado como blob de bytes codificado en DER, retorna una versión de cadena de caracteres codificada en PEM del mismo certificado.

ssl.PEM_cert_to_DER_cert(*pem_cert_string*)

Dado un certificado como cadena de caracteres ASCII PEM, retorna una secuencia de bytes codificada con DER para ese mismo certificado.

ssl.get_default_verify_paths()

Retorna una tupla con nombre con las rutas por defecto de cafile y capath de OpenSSL. Las rutas son las mismas que las usadas por **`SSLContext.set_default_verify_paths()`**. El valor de retorno es una named tuple `DefaultVerifyPaths`:

- **`cafile`** - ruta resuelta a cafile o `None` si el archivo no existe,
- **`capath`** - ruta resuelta a capath o `None` si el directorio no existe,
- **`openssl_cafile_env`** - clave de entorno de OpenSSL que apunta a un cafile,
- **`openssl_cafile`** - camino codificado de forma rígida a un cafile,
- **`openssl_capath_env`** - clave de entorno de OpenSSL que apunta a un capath,
- **`openssl_capath`** - camino codificado de forma rígida a un directorio capath

Added in version 3.4.

ssl.enum_certificates(*store_name*)

Recupera los certificados del almacén de certificados del sistema de Windows. *store_name* puede ser uno de los siguientes: `CA`, `ROOT` o `MY`. Windows también puede proporcionar almacenes de certificados adicionales.

La función retorna una lista de tuplas (cert_bytes, encoding_type, trust). El encoding_type especifica la codificación de cert_bytes. Es **`x509_asn`** para datos X.509 ASN.1 o **`pkcs_7_asn`** para datos PKCS#7 ASN.1. Trust especifica el propósito del certificado como un conjunto de OIDS o exactamente `True` si el certificado es de confianza para todos los propósitos.

Ejemplo:

```
>>> ssl.enum_certificates("CA")
[(b'data...', 'x509_asn', {'1.3.6.1.5.5.7.3.1', '1.
 (b'data...', 'x509_asn', True)]
```

Availability: Windows.

Added in version 3.4.

ssl.enum_crls(*store_name*)

Obtiene CRLs del almacén de certificados del sistema de Windows. *store_name* puede ser uno de los siguientes: `CA`, `ROOT` o `MY`. Windows también puede proporcionar almacenes de certificados adicionales.

La función retorna una lista de tuplas (cert_bytes, encoding_type, trust). El encoding_type especifica la codificación de cert_bytes. Es **`x509_asn`** para datos X.509 ASN.1 o **`pkcs_7_asn`** para datos PKCS#7 ASN.1.

Availability: Windows.

Added in version 3.4.

Constantes

Todas las constantes son ahora colecciones **`enum.IntEnum`** o **`enum.IntFlag`**.

Added in version 3.6.

ssl.CERT_NONE

Possible value for `SSLContext.verify_mode`. Except for `PROTOCOL_TLS_CLIENT`, it is the default mode. With client-side sockets, just about any cert is accepted. Validation errors, such as untrusted or expired cert, are ignored and do not abort the TLS/SSL handshake.

En modo servidor, no se solicita ningún certificado al cliente, por lo que el cliente no envía ninguno para la autenticación del certificado del cliente.

Vea la discusión sobre Consideraciones de seguridad más abajo.

ssl.CERT_OPTIONAL

Possible value for `SSLContext.verify_mode`. In client mode, `CERT_OPTIONAL` has the same meaning as `CERT_REQUIRED`. It is recommended to use `CERT_REQUIRED` for client-side sockets instead.

En el modo servidor, se envía una solicitud de certificado de cliente al cliente. El cliente puede ignorar la solicitud o enviar un certificado para realizar la autenticación de certificado de cliente TLS. Si el cliente opta por enviar un certificado, éste se verifica. Cualquier error de verificación aborta inmediatamente el handshake TLS.

Use of this setting requires a valid set of CA certificates to be passed to `SSLContext.load_verify_locations()`.

ssl.CERT_REQUIRED

Possible value for `SSLContext.verify_mode`. In this mode, certificates are required from the other side of the socket connection; an `SSLError` will be raised if no certificate is provided, or if its validation fails. This mode is **not** sufficient to verify a certificate in client mode as it does not match hostnames. `check_hostname` must be enabled as well to verify the authenticity of a cert. `PROTOCOL_TLS_CLIENT` uses `CERT_REQUIRED` and enables `check_hostname` by default.

Con socket servidor, este modo proporciona una autenticación obligatoria de certificado de cliente TLS. Se envía una solicitud de certificado de cliente al cliente y el cliente debe proporcionar un certificado válido y de confianza.

Use of this setting requires a valid set of CA certificates to be passed to `SSLContext.load_verify_locations()`.

class ssl.VerifyMode

Colección `enum.IntEnum` de constantes CERT_*.

Added in version 3.6.

ssl.VERIFY_DEFAULT

Valor posible para `SSLContext.verify_flags`. En este modo, las listas de revocación de certificado (CRLs) no son verificadas. Por defecto OpenSSL no requiere ni verifica CRLs.

Added in version 3.4.

ssl.VERIFY_CRL_CHECK_LEAF

Valor posible para `SSLContext.verify_flags`. En este modo, sólo el certificado de pares es verificado pero ninguno de los certificados CA intermedios. El modo requiere una CRL válida que esté firmada por el emisor del certificado de pares (su CA antecesora directa). Si no se ha cargado una CRL adecuada con `SSLContext.load_verify_locations`, la validación fallará.

Added in version 3.4.

ssl.VERIFY_CRL_CHECK_CHAIN

Valor posible para `SSLContext.verify_flags`. En este modo, las CRLs de todos los certificados en la cadena de certificado de pares son verificadas.

Added in version 3.4.

ssl.VERIFY_X509_STRICT

Valor posible para `SSLContext.verify_flags` para desactivar soluciones alternativas para certificados X.509 rotos.

Added in version 3.4.

ssl.VERIFY_ALLOW_PROXY_CERTS

Valor posible para `SSLContext.verify_flags` para desactivar soluciones alternativas para certificados X.509 rotos.

Added in version 3.10.

ssl.VERIFY_X509_TRUSTED_FIRST

Valor posible para `SSLContext.verify_flags`. Indica a OpenSSL de preferir certificados de confianza al construir la cadena de confianza para validar un certificado. Esta opción está activada por defecto.

Added in version 3.4.4.

ssl.VERIFY_X509_PARTIAL_CHAIN

Valor posible para `SSLContext.verify_flags`. Indica a OpenSSL que acepte CA intermedias en el almacén de confianza para que se traten como anclajes de confianza, de la misma manera que los certificados de CA raíz autofirmados. Esto hace posible confiar en los certificados emitidos por una CA intermedia sin tener que confiar en su CA raíz antecesora.

Added in version 3.10.

class ssl.VerifyFlags

Colección `enum.IntFlag` de constantes VERIFY_*.

Added in version 3.6.

ssl.PROTOCOL_TLS

Selecciona la versión mas alta del protocolo soportada tanto por el cliente como por el servidor. A pesar de su nombre, esta opción puede seleccionar ambos protocolos «SSL» y «TLS».

Added in version 3.6.

Obsoleto desde la versión 3.10: Los clientes y servidores TLS requieren diferentes configuraciones predeterminadas para una comunicación segura. La constante del protocolo TLS genérico está en desuso en favor de `PROTOCOL_TLS_CLIENT` y `PROTOCOL_TLS_SERVER`.

ssl.PROTOCOL_TLS_CLIENT

Negocie automáticamente la versión de protocolo más alta que admiten tanto el cliente como el servidor, y configure las conexiones del lado del cliente de contexto. El protocolo habilita `CERT_REQUIRED` y `check_hostname` de forma predeterminada.

Added in version 3.6.

ssl.PROTOCOL_TLS_SERVER

Selecciona la versión mas alta del protocolo soportada tanto por el cliente como por el servidor. A pesar de su nombre, esta opción puede seleccionar ambos protocolos «SSL» y «TLS».

Added in version 3.6.

ssl.PROTOCOL_SSLv23

Alias para `PROTOCOL_TLS`.

Obsoleto desde la versión 3.6: Utilice en su lugar `PROTOCOL_TLS`.

ssl.PROTOCOL_SSLv3

Selecciona la versión 3 de SSL como protocolo de cifrado del canal.

Este protocolo no está disponible si se compila OpenSSL con la opción `no-ssl3`.

Advertencia

La versión 3 de SSL es insegura. Su uso es muy desaconsejado.

Obsoleto desde la versión 3.6: OpenSSL ha descontinuado todos los protocolos específicos de la versión. Utilice el protocolo predeterminado **`PROTOCOL_TLS_SERVER`** o **`PROTOCOL_TLS_CLIENT`** con **`SSLContext.minimum_version`** y **`SSLContext.maximum_version`** en su lugar.

ssl.PROTOCOL_TLSv1

Selecciona la versión 1.0 de TLS como protocolo de cifrado del canal.

Obsoleto desde la versión 3.6: OpenSSL ha descontinuado todos los protocolos específicos de la versión.

ssl.PROTOCOL_TLSv1_1

Selecciona la versión 1.1 de TLS como protocolo de cifrado del canal. Disponible sólo con openssl en versión 1.0.1+.

Added in version 3.4.

Obsoleto desde la versión 3.6: OpenSSL ha descontinuado todos los protocolos específicos de la versión.

ssl.PROTOCOL_TLSv1_2

Selecciona la versión 1.1 de TLS como protocolo de cifrado del canal. Disponible sólo con openssl en versión 1.0.1+.

Added in version 3.4.

Obsoleto desde la versión 3.6: OpenSSL ha descontinuado todos los protocolos específicos de la versión.

ssl.OP_ALL

Activa soluciones alternativas para varios errores presentes en otras implementaciones SSL. Esta opción esta activada por defecto. No necesariamente activa las mismas opciones como la constante `SSL_OP_ALL` de OpenSSL.

Added in version 3.2.

ssl.OP_NO_SSLv2

Evita una conexión SSLv2. Esta opción sólo es aplicable junto con **`PROTOCOL_TLS`**. Evita que los pares elijan SSLv2 como versión del protocolo.

Added in version 3.2.

Obsoleto desde la versión 3.6: SSLv2 es obsoleto

ssl.OP_NO_SSLv3

Evita una conexión SSLv3. Esta opción sólo es aplicable junto con **`PROTOCOL_TLS`**. Evita que los pares elijan SSLv3 como versión del protocolo.

Added in version 3.2.

Obsoleto desde la versión 3.6: SSLv3 es obsoleto

ssl.OP_NO_TLSv1

Evita una conexión TLSv1. Esta opción sólo es aplicable junto con **`PROTOCOL_TLS`**. Evita que los pares elijan TLSv1 como versión del protocolo.

Added in version 3.2.

Obsoleto desde la versión 3.7: Esta opción es obsoleta desde OpenSSL 1.1.0, utilice en su lugar los nuevos **`SSLContext.minimum_version`** y **`SSLContext.maximum_version`**.

ssl.OP_NO_TLSv1_1

Evita una conexión TLSv1.1. Esta opción sólo es aplicable junto con **`PROTOCOL_TLS`**. Evita que los pares elijan TLSv1.1 como versión del protocolo. Disponible sólo con openssl en versión 1.0.1+.

Added in version 3.4.

Obsoleto desde la versión 3.7: Esta opción es obsoleta desde OpenSSL 1.1.0.

ssl.OP_NO_TLSv1_2

Evita una conexión TLSv1.2. Esta opción sólo es aplicable junto con **`PROTOCOL_TLS`**. Evita que los pares elijan TLSv1.2 como versión del protocolo. Disponible sólo con openssl en versión 1.0.1+.

Added in version 3.4.

Obsoleto desde la versión 3.7: Esta opción es obsoleta desde OpenSSL 1.1.0.

ssl.OP_NO_TLSv1_3

Evita una conexión TLSv1.3. Esta opción sólo es aplicable junto con **`PROTOCOL_TLS`**. Evita que los pares elijan TLSv1.3 como versión del protocolo. TLS 1.3 está disponible con OpenSSL 1.1.1 o superior. Cuando Python es compilado contra una versión mas antigua de OpenSSL, la opción vale *0* por defecto.

Added in version 3.6.3.

Obsoleto desde la versión 3.7: The option is deprecated since OpenSSL 1.1.0. It was added to 2.7.15 and 3.6.3 for backwards compatibility with OpenSSL 1.0.2.

ssl.OP_NO_RENEGOTIATION

Desactiva toda re-negociación en TLSv1.2 y anteriores. No envía mensajes HelloRequest e ignora solicitudes de renegociación vía ClientHello.

Esta opción sólo está disponible con OpenSSL 1.1.0h y posteriores.

Added in version 3.7.

ssl.OP_CIPHER_SERVER_PREFERENCE

Utiliza la preferencia de ordenación de cifrado del servidor, en lugar de la del cliente. Esta opción no tiene efecto en los sockets del cliente ni en los sockets del servidor SSLv2.

Added in version 3.3.

ssl.OP_SINGLE_DH_USE

Prevents reuse of the same DH key for distinct SSL sessions. This improves forward secrecy but requires more computational resources. This option only applies to server sockets.

Added in version 3.3.

ssl.OP_SINGLE_ECDH_USE

Prevents reuse of the same ECDH key for distinct SSL sessions. This improves forward secrecy but requires more computational resources. This option only applies to server sockets.

Added in version 3.3.

ssl.OP_ENABLE_MIDDLEBOX_COMPAT

Enviar mensajes Change Cipher Spec (CCS) ficticios en el handshake de TLS 1.3 para que una conexión TLS 1.3 se parezca más a una conexión TLS 1.2.

Esta opción sólo está disponible con OpenSSL 1.1.1 y posteriores.

Added in version 3.8.

ssl.OP_NO_COMPRESSION

Desactivar la compresión en el canal SSL. Esto es útil si el protocolo de la aplicación soporta su propio esquema de compresión.

Added in version 3.3.

class ssl.Options

Colección `enum.IntFlag` de constantes OP_*.

ssl.OP_NO_TICKET

Evita que el lado del cliente solicite un ticket de sesión.

Added in version 3.6.

ssl.OP_IGNORE_UNEXPECTED_EOF

Ignore el cierre inesperado de las conexiones TLS.

Esta opción sólo está disponible con OpenSSL 1.0.0 y posteriores.

Added in version 3.10.

ssl.OP_ENABLE_KTLS

Enable the use of the kernel TLS. To benefit from the feature, OpenSSL must have been compiled with support for it, and the negotiated cipher suites and extensions must be supported by it (a list of supported ones may vary by platform and kernel version).

Note that with enabled kernel TLS some cryptographic operations are performed by the kernel directly and not via any available OpenSSL Providers. This might be undesirable if, for example, the application requires all cryptographic operations to be performed by the FIPS provider.

Esta opción sólo está disponible con OpenSSL 1.0.0 y posteriores.

Added in version 3.12.

ssl.OP_LEGACY_SERVER_CONNECT

Allow legacy insecure renegotiation between OpenSSL and unpatched servers only.

Added in version 3.12.

ssl.HAS_ALPN

Si la biblioteca OpenSSL tiene soporte incorporado para la extensión TLS *Application-Layer Protocol Negotiation* como se describe en **RFC 7301** [https://datatracker.ietf.org/doc/html/rfc7301.html].

Added in version 3.5.

ssl.HAS_NEVER_CHECK_COMMON_NAME

Si la biblioteca OpenSSL tiene soporte incorporado para no comprobar el nombre común del sujeto y `SSLContext.hostname_checks_common_name` es modificable.

Added in version 3.7.

ssl.HAS_ECDH

Si la biblioteca OpenSSL tiene soporte incorporado para el intercambio de claves Diffie-Hellman basado en Elliptic Curve. Esto debería ser cierto a menos que la función haya sido desactivada explícitamente por el distribuidor.

Added in version 3.3.

ssl.HAS_SNI

Si la biblioteca OpenSSL tiene soporte incorporado para la extensión *Server Name Indication* (como se define en **RFC 6066** [https://datatracker.ietf.org/doc/html/rfc6066.html]).

Added in version 3.2.

ssl.HAS_NPN

Si la biblioteca OpenSSL tiene soporte incorporado para *Next*

Protocol Negotiation como se describe en Application Layer Protocol Negotiation [https://en.wikipedia.org/wiki/Application-Layer_Protocol_Negotiation]. Cuando es verdadero, puede utilizar el método `SSLContext.set_npn_protocols()` para anunciar los protocolos que desea soportar.

Added in version 3.3.

ssl.HAS_SSLv2

Si la biblioteca OpenSSL tiene soporte incorporado para el protocolo SSL 2.0.

Added in version 3.7.

ssl.HAS_SSLv3

Si la biblioteca OpenSSL tiene soporte incorporado para el protocolo SSL 3.0.

Added in version 3.7.

ssl.HAS_TLSv1

Si la biblioteca OpenSSL tiene soporte incorporado para el protocolo TLS 1.0.

Added in version 3.7.

ssl.HAS_TLSv1_1

Si la biblioteca OpenSSL tiene soporte incorporado para el protocolo TLS 1.1.

Added in version 3.7.

ssl.HAS_TLSv1_2

Si la biblioteca OpenSSL tiene soporte incorporado para el protocolo TLS 1.2.

Added in version 3.7.

ssl.HAS_TLSv1_3

Si la biblioteca OpenSSL tiene soporte incorporado para el protocolo TLS 1.3.

Added in version 3.7.

ssl.HAS_PSK

Whether the OpenSSL library has built-in support for TLS-PSK.

Added in version 3.13.

ssl.HAS_PHA

Whether the OpenSSL library has built-in support for TLS-PHA.

Added in version 3.14.

ssl.CHANNEL_BINDING_TYPES

Lista de tipos de enlace de canales TLS admitidos. Las cadenas de caracteres en esta lista pueden ser usadas como argumentos para **`SSLSocket.get_channel_binding()`**.

Added in version 3.3.

ssl.OPENSSL_VERSION

La cadena de versión de la biblioteca OpenSSL cargada por el intérprete:

```
>>> ssl.OPENSSL_VERSION
'OpenSSL 1.0.2k  26 Jan 2017'
```

Added in version 3.2.

ssl.OPENSSL_VERSION_INFO

Una tupla de cinco números enteros representando la información de versión de la biblioteca OpenSSL:

```
>>> ssl.OPENSSL_VERSION_INFO
(1, 0, 2, 11, 15)
```

Added in version 3.2.

ssl.OPENSSL_VERSION_NUMBER

El número de versión en bruto de la biblioteca OpenSSL, como un único número entero:

```
>>> ssl.OPENSSL_VERSION_NUMBER
268443839
>>> hex(ssl.OPENSSL_VERSION_NUMBER)
'0x100020bf'
```

Added in version 3.2.

ssl.ALERT_DESCRIPTION_HANDSHAKE_FAILURE
ssl.ALERT_DESCRIPTION_INTERNAL_ERROR
ALERT_DESCRIPTION_*

Descripciones de alertas de **RFC 5246** [https://datatracker.ietf.org/doc/html/rfc5246.html] y otras. El IANA TLS Alert Registry [https://www.iana.org/assignments/tls-parameters/tls-parameters.xml#tls-parameters-6] contiene esta lista y las referencias a las RFC donde se define su significado.

Se utiliza como valor de retorno de la función callback en `SSLContext.set_servername_callback()`.

Added in version 3.4.

class ssl.AlertDescription

Colección `enum.IntEnum` de constantes ALERT_DESCRIPTION_*.

Added in version 3.6.

Purpose.SERVER_AUTH

Opción para `create_default_context()` y `SSLContext.load_default_certs()`. Este valor indica que el contexto puede utilizarse para autenticar servidores web (por lo tanto, se utilizará para crear sockets del lado del cliente).

Added in version 3.4.

Purpose.CLIENT_AUTH

Opción para `create_default_context()` y `SSLContext.load_default_certs()`. Este valor indica que el contexto puede utilizarse para autenticar clientes web (por lo tanto, se utilizará para crear sockets del lado del servidor).

Added in version 3.4.

class ssl.SSLErrorNumber

Colección `enum.IntEnum` de constantes SSL_ERROR_*.

Added in version 3.6.

class ssl.TLSVersion

Colección `enum.IntEnum` de versiones SSL y TLS para `SSLContext.maximum_version` y `SSLContext.minimum_version`.

Added in version 3.7.

TLSVersion.MINIMUM_SUPPORTED

TLSVersion.MAXIMUM_SUPPORTED

La mínima o máxima versión soportada de SSL o TLS. Estas son constantes mágicas. Sus valores no reflejan la mas baja o mas alta versión TLS/SSL disponible.

TLSVersion.SSLv3

TLSVersion.TLSv1

TLSVersion.TLSv1_1

TLSVersion.TLSv1_2

TLSVersion.TLSv1_3

SSL 3.0 a TLS 1.3.

Obsoleto desde la versión 3.10: Todos los miembros de **`TLSVersion`**, excepto **`TLSVersion.TLSv1_2`** y **`TLSVersion.TLSv1_3`**, están en desuso.

SSL sockets

class ssl.SSLSocket(*socket.socket*)

Los sockets SSL proporcionan los siguientes métodos de Objetos Socket:

- **`accept()`**
- **`bind()`**
- **`close()`**
- **`connect()`**
- **`detach()`**
- **`fileno()`**
- **`getpeername()`**, **`getsockname()`**
- **`getsockopt()`**, **`setsockopt()`**
- **`gettimeout()`**, **`settimeout()`**, **`setblocking()`**
- **`listen()`**
- **`makefile()`**
- **`recv()`**, **`recv_into()`** (but passing a non-zero `flags` argument is not allowed)
- **`send()`**, **`sendall()`** (with the same limitation)
- **`sendfile()`** (but **`os.sendfile`** will be used for plain-text sockets only, else **`send()`** will be used)
- **`shutdown()`**

Sin embargo, dado que el protocolo SSL (y TLS) tiene su propia estructura encima de TCP, la abstracción de los sockets SSL puede, en ciertos aspectos, divergir de la especificación de los sockets normales a nivel de SO. Ver especialmente las notas sobre sockets no bloqueantes.

Instancias de **`SSLSocket`** deben ser creadas usando el método **`SSLContext.wrap_socket()`**.

Distinto en la versión 3.5: El método **`sendfile()`** ha sido agregado.

Distinto en la versión 3.5: The **`shutdown()`** does not reset the socket timeout each time bytes are received or sent. The socket timeout is now the maximum total duration of the shutdown.

Obsoleto desde la versión 3.6: Crear una instancia de **`SSLSocket`** directamente es obsoleto, utilice **`SSLContext.wrap_socket()`** para envolver un socket.

Distinto en la versión 3.7: Las instancias de **`SSLSocket`** deben crearse con **`wrap_socket()`**. En versiones anteriores, era posible crear instancias directamente. Esto nunca fue documentado ni soportado oficialmente.

Distinto en la versión 3.10: Python ahora usa `SSL_read_ex` y `SSL_write_ex` internamente. Las funciones admiten la lectura y escritura de datos de más de 2 GB. La escritura de datos de longitud cero ya no falla con un error de violación de protocolo.

Los sockets SSL tienen también los siguientes métodos y atributos adicionales:

SSLSocket.read(*len=1024*, *buffer=None*)

Lee hasta *len* bytes de datos del socket SSL y retorna el resultado como una instancia `bytes`. Si *buffer* es especificado, entonces se lee hacia el búfer en su lugar, y retorna el número de bytes leídos.

Lanza **`SSLWantReadError`** o **`SSLWantWriteError`** si el socket es no-bloqueante y la lectura se bloquearía.

Como en cualquier momento es posible una re-negociación, una llamada a **`read()`** también puede provocar operaciones de escritura.

Distinto en la versión 3.5: The socket timeout is no longer reset each time bytes are received or sent. The socket timeout is now the maximum total duration to read up to *len* bytes.

Obsoleto desde la versión 3.6: Utilice `recv()` en lugar de `read()`.

SSLSocket.write(*data*)

Write *data* to the SSL socket and return the number of bytes written. The *data* argument must be an object supporting the buffer interface.

Lanza `SSLWantReadError` o `SSLWantWriteError` si el socket es no-bloqueante y la escritura se bloquearía.

Como en cualquier momento es posible una re-negociación, una llamada a `write()` también puede provocar operaciones de lectura.

Distinto en la versión 3.5: The socket timeout is no longer reset each time bytes are received or sent. The socket timeout is now the maximum total duration to write *data*.

Obsoleto desde la versión 3.6: Utilice `send()` en lugar de `write()`.

Nota

Los métodos `read()` y `write()` son los métodos de bajo nivel que leen y escriben datos no cifrados a nivel de aplicación y los descifran/cifran a datos cifrados a nivel de cable. Estos métodos requieren una conexión SSL activa, es decir, que se haya completado el handshake y no se haya llamado a `SSLSocket.unwrap()`.

Normalmente se deberían utilizar los métodos de la API de sockets como `recv()` y `send()` en lugar de estos métodos.

SSLSocket.do_handshake(*block=False*)

Realiza el handshake de configuración SSL.

If *block* is true and the timeout obtained by `gettimeout()` is zero, the socket is set in blocking mode until the handshake is performed.

Distinto en la versión 3.4: The handshake method also performs **`match_hostname()`** when the `check_hostname` attribute of the socket's `context` is true.

Distinto en la versión 3.5: The socket timeout is no longer reset each time bytes are received or sent. The socket timeout is now the maximum total duration of the handshake.

Distinto en la versión 3.7: Hostname or IP address is matched by OpenSSL during handshake. The function **`match_hostname()`** is no longer used. In case OpenSSL refuses a hostname or IP address, the handshake is aborted early and a TLS alert message is sent to the peer.

SSLSocket.getpeercert(*binary_form=False*)

Si no hay un certificado para el peer en el otro extremo de la conexión, retorna `None`. Si el handshake SSL no se ha realizado todavía, lanza `ValueError`.

Si el parámetro `binary_form` es `False`, y se ha recibido un certificado del peer, este método retorna una instancia `dict`. Si el certificado no fue validado, el dict está vacío. Si el certificado fue validado, retorna un dict con varias claves, entre ellas `subject` (la entidad para la que se emitió el certificado) y `issuer` (la entidad que emite el certificado). Si un certificado contiene una instancia de la extensión *Subject Alternative Name* (véase **RFC 3280** [https://datatracker.ietf.org/doc/html/rfc3280.html]), también habrá una clave `subjectAltName` en el diccionario.

Los campos `subject` y `issuer` son tuplas que contienen la secuencia de nombres distinguidos relativos (RDNs) indicados en la estructura de datos del certificado para los campos respectivos, y cada RDN es una secuencia de pares nombre-valor. Este es un ejemplo del mundo real:

```
{'issuer': ((('countryName', 'IL'),),
            (('organizationName', 'StartCom Ltd.'),
            (('organizationalUnitName',
              'Secure Digital Certificate Signing')
            (('commonName',
              'StartCom Class 2 Primary Intermediat
 'notAfter': 'Nov 22 08:15:19 2013 GMT',
 'notBefore': 'Nov 21 03:09:52 2011 GMT',
 'serialNumber': '95F0',
 'subject': ((('description', '571208-SLe257oHY9fVQ
             (('countryName', 'US'),),
             (('stateOrProvinceName', 'California')
             (('localityName', 'San Francisco'),),
             (('organizationName', 'Electronic Fron
             (('commonName', '*.eff.org'),),
             (('emailAddress', 'hostmaster@eff.org'
 'subjectAltName': (('DNS', '*.eff.org'), ('DNS', '
 'version': 3}
```

Si el parámetro `binary_form` es **`True`**, y se proporcionó un certificado, este método retorna la forma codificada en DER del certificado completo como una secuencia de bytes, o **`None`** si el par no proporcionó un certificado. El hecho de que el par proporcione un certificado depende del rol del socket SSL:

- para un socket SSL cliente, el servidor siempre proporcionará un certificado, independientemente de si se requirió la validación;
- para un socket SSL servidor, el cliente sólo proporcionará un certificado cuando lo solicite el servidor; por lo tanto **`getpeercert()`** devolverá **`None`** si ha utilizado **`CERT_NONE`** (en lugar de **`CERT_OPTIONAL`** o **`CERT REQUIRED`**).

See also **`SSLContext.check_hostname`**.

Distinto en la versión 3.2: El diccionario retornado incluye elementos adicionales tales como `issuer` y `notBefore`.

Distinto en la versión 3.4: `ValueError` se lanza cuando no se realiza el handshake. El diccionario retornado incluye elementos de extensión X509v3 adicionales como `crlDistributionPoints`, `caIssuers` y `OCSP` URIs.

Distinto en la versión 3.9: Las cadenas de direcciones IPv6 ya no tienen una nueva línea al final.

SSLSocket.get_verified_chain()

Returns verified certificate chain provided by the other end of the SSL channel as a list of DER-encoded bytes. If certificate verification was disabled method acts the same as **`get_unverified_chain()`**.

Added in version 3.13.

SSLSocket.get_unverified_chain()

Returns raw certificate chain provided by the other end of the SSL channel as a list of DER-encoded bytes.

Added in version 3.13.

SSLSocket.cipher()

Retorna una tupla de tres valores que contiene el nombre del cifrado que se está utilizando, la versión del protocolo SSL que define su uso y el número de bits secretos que se están utilizando. Si no se ha establecido ninguna conexión, retorna `None`.

SSLSocket.shared_ciphers()

Return the list of ciphers available in both the client and server. Each entry of the returned list is a three-value tuple containing the name of the cipher, the version of the SSL protocol that defines its use, and the number of secret bits the cipher uses. **`shared_ciphers()`** returns `None` if no connection has been established or the socket is a client socket.

Added in version 3.5.

SSLSocket.compression()

Retorna el algoritmo de compresión utilizado como una cadena de caracteres, o `None` si la conexión no está comprimida.

Si el protocolo de nivel superior soporta su propio mecanismo de compresión, puede utilizar **`OP_NO_COMPRESSION`** para desactivar la compresión a nivel de SSL.

Added in version 3.3.

SSLSocket.get_channel_binding(*cb_type='tls-unique'*)

Obtiene los datos de enlace del canal para la conexión actual, como un objeto bytes. Retorna `None` si no está conectado o no se ha completado el handshake.

El parámetro *cb_type* permite seleccionar el tipo de enlace de canal deseado. Los tipos de enlace de canal válidos se enumeran en la lista **`CHANNEL_BINDING_TYPES`**. Actualmente, sólo se admite la vinculación de canal "tls-unique", definida por **RFC 5929** [https://datatracker.ietf.org/doc/html/rfc5929.html]. **`ValueError`** se lanzará si se solicita un tipo de vinculación de canal no admitido.

Added in version 3.3.

SSLSocket.selected_alpn_protocol()

Retorna el protocolo que fue seleccionado durante el handshake TLS. Si no se ha llamado a **`SSLContext.set_alpn_protocols()`**, si la otra parte no soporta ALPN, si este socket no soporta ninguno de los protocolos propuestos por el cliente, o si el handshake no ha ocurrido todavía, se retorna `None`.

Added in version 3.5.

SSLSocket.selected_npn_protocol()

Retorna el protocolo de nivel superior que se seleccionó durante el handshake TLS/SSL. Si no se llamó a `SSLContext.set_npn_protocols()`, o si la otra parte no soporta NPN, o si el handshake aún no ha ocurrido, esto devolverá `None`.

Added in version 3.3.

Obsoleto desde la versión 3.10: NPN ha sido reemplazada por ALPN

SSLSocket.unwrap()

Realiza el handshake de cierre de SSL, que elimina la capa TLS del socket subyacente, y retorna el objeto socket subyacente. Esto puede utilizarse para pasar de una operación encriptada sobre una conexión a una sin encriptar. El socket retornado debe utilizarse siempre para la comunicación posterior con el otro lado de la conexión, en lugar del socket original.

SSLSocket.verify_client_post_handshake()

Solicita la autenticación post-handshake (PHA) de un cliente TLS 1.3. PHA sólo puede iniciarse para una conexión TLS 1.3 desde un socket del lado del servidor, después del handshake TLS inicial y con PHA habilitado en ambos lados, ver `SSLContext.post_handshake_auth`.

El método no realiza un intercambio de certificados inmediatamente. El lado del servidor envía una CertificateRequest durante el siguiente evento de escritura y espera que el cliente responda con un certificado en el siguiente evento de lectura.

Si alguna precondición no se cumple (por ejemplo, no es TLS 1.3, PHA no está habilitado), se genera un `SSLError`.

Nota

Sólo está disponible con OpenSSL 1.1.1 y TLS 1.3 habilitados. Sin el soporte de TLS 1.3, el método lanza

`NotImplementedError`.

Added in version 3.8.

SSLSocket.version()

Retorna la versión real del protocolo SSL negociada por la conexión como una cadena, o `None` si no se establece una conexión segura. Al momento de escribir, los posibles valores de retorno incluyen `"SSLv2"`, `"SSLv3"`, `"TLSv1"`, `"TLSv1.1"` y `"TLSv1.2"`. Las versiones recientes de OpenSSL pueden definir más valores de retorno.

Added in version 3.5.

SSLSocket.pending()

Retorna el número de bytes ya descifrados disponibles para leer, pendientes de la conexión.

SSLSocket.context

The **`SSLContext`** object this SSL socket is tied to.

Added in version 3.2.

SSLSocket.server_side

Un booleano que es `True` para los sockets del lado del servidor y `False` para los sockets del lado del cliente.

Added in version 3.2.

SSLSocket.server_hostname

Hostname del servidor: tipo **`str`**, o `None` para el socket del lado del servidor o si el hostname no fue especificado en el constructor.

Added in version 3.2.

Distinto en la versión 3.7: El atributo es ahora siempre texto ASCII. Cuando `server_hostname` es un nombre de dominio internacionalizado (IDN), este atributo almacena ahora la forma de etiqueta A (`"xn--pythn-mua.org"`), en lugar de la forma de etiqueta U (`"pythön.org"`).

SSLSocket.session

La **`SSLSession`** para esta conexión SSL. La sesión está disponible para los sockets del lado del cliente y del servidor después de que se haya realizado el handshake TLS. Para los sockets del cliente la sesión puede ser establecida antes de que **`do_handshake()`** haya sido llamado para reutilizar una sesión.

Added in version 3.6.

SSLSocket.session_reused

Added in version 3.6.

SSL contexts

Added in version 3.2.

Un contexto SSL contiene varios datos más duraderos que las conexiones SSL individuales, como opciones de configuración SSL, certificado(s) y clave(s) privada(s). También gestiona un cache de sesiones SSL para sockets del lado del servidor, para acelerar conexiones repetidas de los mismos clientes.

class ssl.SSLContext(*protocol=None*)

Crea un nuevo contexto SSL. Puede pasar *protocolo* que debe ser una de las constantes `PROTOCOL_*` definidas en este módulo. El parámetro especifica la versión del protocolo SSL a utilizar. Típicamente, el servidor elige una versión particular del protocolo, y el cliente debe adaptarse a la elección del servidor. La mayoría de las versiones no son interoperables con las demás. Si no se especifica, el valor por defecto es **`PROTOCOL_TLS`**; proporciona la mayor compatibilidad con otras versiones.

Esta es una tabla que muestra qué versiones de un cliente (en la parte inferior) pueden conectarse a qué versiones de un servidor (en la parte superior):

cliente / **servidor**	**SSLv2**	**SSLv3**	**TLS** [3]	**TLSv1**	**TLSv1.1**	**TLSv1.2**
SSLv2	sí	no	no [1]	no	no	no
SSLv3	no	sí	no [2]	no	no	no
TLS (*SSLv23*) [3]	no [1]	no [2]	sí	sí	sí	sí
TLSv1	no	no	sí	sí	no	no
TLSv1.1	no	no	sí	no	sí	no
TLSv1.2	no	no	sí	no	no	sí

Notas a pie de página

[1] (1,2) [2] (1,2) [3] (1,2)
`SSLContext` desactiva SSLv2 con **`OP_NO_SSLv2`** por defecto.

`SSLContext` desactiva SSLv3 con **`OP_NO_SSLv3`** por defecto.

El protocolo TLS 1.3 estará disponible con **`PROTOCOL_TLS`** en OpenSSL > = 1.1.1. No existe una constante PROTOCOL dedicada sólo a TLS 1.3.

Ver también

`create_default_context()` lets the **`ssl`** module choose security settings for a given purpose.

Distinto en la versión 3.6: The context is created with secure default values. The options **`OP_NO_COMPRESSION`**, **`OP_CIPHER_SERVER_PREFERENCE`**, **`OP_SINGLE_DH_USE`**, **`OP_SINGLE_ECDH_USE`**, **`OP_NO_SSLv2`**, and **`OP_NO_SSLv3`** (except for **`PROTOCOL_SSLv3`**) are set by default. The initial cipher suite list contains only `HIGH` ciphers, no `NULL` ciphers and no `MD5` ciphers.

Obsoleto desde la versión 3.10: **`SSLContext`** sin argumento de protocolo está en desuso. La clase de contexto requerirá el protocolo `PROTOCOL_TLS_CLIENT` o `PROTOCOL_TLS_SERVER` en el futuro.

Distinto en la versión 3.10: Los conjuntos de cifrado predeterminados ahora incluyen solo cifrados AES y ChaCha20 seguros con secreto directo y nivel de seguridad 2. Están prohibidas las claves RSA y DH con menos de 2048 bits y las claves ECC con menos de 224 bits. `PROTOCOL_TLS`, `PROTOCOL_TLS_CLIENT` y `PROTOCOL_TLS_SERVER` usan TLS 1.2 como versión mínima de TLS.

> **Nota**
>
> **`SSLContext`** only supports limited mutation once it has been used by a connection. Adding new certificates to the internal trust store is allowed, but changing ciphers, verification settings, or mTLS certificates may result in surprising behavior.

> **Nota**
>
> **`SSLContext`** is designed to be shared and used by multiple connections. Thus, it is thread-safe as long as it is not reconfigured after being used by a connection.

Los objetos `SSLContext` tienen los siguientes métodos y atributos:

SSLContext.cert_store_stats()

Obtiene estadísticas sobre las cantidades de certificados X.509 cargados, la cantidad de certificados X.509 marcados como certificados CA y las listas de revocación de certificados como diccionario.

Ejemplo para un contexto con un certificado CA y otro certificado:

```
>>> context.cert_store_stats()
{'crl': 0, 'x509_ca': 1, 'x509': 2}
```

Added in version 3.4.

SSLContext.load_cert_chain(*certfile*, *keyfile=None*, *password=None*)

Carga una clave privada y el certificado correspondiente. La cadena *certfile* debe ser la ruta a un solo archivo en formato PEM que contenga el certificado, así como cualquier número de certificados de CA necesarios para establecer la autenticidad del certificado. La cadena *keyfile*, si está presente, debe apuntar a un archivo que contenga la clave privada. De lo contrario, la clave privada también se tomará de *certfile*. Consulte la discusión de Certificados para obtener más información sobre cómo se almacena el certificado en *certfile*.

El argumento *password* puede ser una función a la que llamar para obtener la contraseña para descifrar la clave privada. Sólo se llamará si la clave privada está encriptada y se necesita una contraseña. Se llamará sin argumentos, y deberá devolver una cadena de caracteres, bytes o bytearray. Si el valor retornado es una cadena de caracteres, se codificará como UTF-8 antes de utilizarlo para descifrar la clave. Alternativamente, se puede suministrar un valor de cadena de caracteres, bytes o bytearray directamente como argumento *password*. Se ignorará si la clave privada no está cifrada y no se necesita una contraseña.

Si el argumento *password* no es especificado y una contraseña es requerida, el mecanismo de solicitud de contraseña incorporado de OpenSSL se usará para solicitarle una contraseña al usuario de forma interactiva.

Un **`SSLError`** es lanzado si la clave privada no coincide con el certificado.

Distinto en la versión 3.3: Nuevo argumento opcional *password*.

SSLContext.load_default_certs(*purpose=Purpose.SERVER_AUTH*)

Carga un conjunto de certificados predeterminados de «autoridad de certificación» (CA) desde ubicaciones predeterminadas. En Windows, carga certificados de CA de los almacenes del sistema `CA` y `ROOT`. En todos los sistemas llama a **`SSLContext.set_default_verify_paths()`**. En el futuro, el método también puede cargar certificados de CA desde otras ubicaciones.

The *purpose* flag specifies what kind of CA certificates are loaded. The default settings **`Purpose.SERVER_AUTH`** loads certificates, that are flagged and trusted for TLS web server authentication (client side sockets). **`Purpose.CLIENT_AUTH`** loads CA certificates for client certificate verification on the server side.

Added in version 3.4.

SSLContext.load_verify_locations(*cafile=None*, *capath=None*, *cadata=None*)

Carga un conjunto de certificados de «autoridad de certificación» (CA) usados para validar certificados de otros pares cuando **`verify_mode`** es distinto de **`CERT_NONE`**. Debe especificarse al menos uno de *cafile* o *capath*.

Este método puede cargar también listas de revocación de certificados (CRLs) en formato PEM o DER. Para poder usar CRLs, **`SSLContext.verify_flags`** debe ser configurado correctamente.

La cadena de caracteres *cafile*, si está presente, es la ruta a un archivo de certificados CA concatenados en formato PEM. Vea la discusión de Certificados para más información acerca de como organizar los certificados en este archivo.

The *capath* string, if present, is the path to a directory containing several CA certificates in PEM format, following an OpenSSL specific layout [https://docs.openssl.org/master/man3/SSL_CTX_load_verify_locations/].

El objeto *cadata*, si está presente, es una cadena de caracteres ASCII de uno o más certificados codificados en PEM o un bytes-like object de certificados codificados en DER. Al igual que con *capath*, las líneas adicionales alrededor de los certificados codificados en PEM se ignoran, pero debe haber al menos un certificado.

Distinto en la versión 3.4: Nuevo argumento opcional *cadata*

SSLContext.get_ca_certs(*binary_form=False*)

Obtiene una lista de certificados de «autoridad de certificación» (CA) cargados. Si el parámetro `binary_form` es **`False`** cada entrada de la lista es un diccionario como la salida de **`SSLSocket.getpeercert()`**. En caso contrario, el método retorna una lista de certificados codificados con DER. La lista retornada no contiene certificados de *capath* a menos que un certificado haya sido solicitado y cargado por una conexión SSL.

Nota

Los certificados de un directorio capath no se cargan a menos que se hayan utilizado al menos una vez.

Added in version 3.4.

SSLContext.get_ciphers()

Obtiene una lista de cifrados habilitados. La lista está en orden de prioridad de cifrado. Véase **`SSLContext.set_ciphers()`**.

Ejemplo:

```
>>> ctx = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
>>> ctx.set_ciphers('ECDHE+AESGCM:!ECDSA')
>>> ctx.get_ciphers()
[{'aead': True,
  'alg_bits': 256,
  'auth': 'auth-rsa',
  'description': 'ECDHE-RSA-AES256-GCM-SHA384 TLSv1
                 'Enc=AESGCM(256) Mac=AEAD',
  'digest': None,
  'id': 50380848,
  'kea': 'kx-ecdhe',
  'name': 'ECDHE-RSA-AES256-GCM-SHA384',
  'protocol': 'TLSv1.2',
  'strength_bits': 256,
  'symmetric': 'aes-256-gcm'},
 {'aead': True,
  'alg_bits': 128,
  'auth': 'auth-rsa',
  'description': 'ECDHE-RSA-AES128-GCM-SHA256 TLSv1
                 'Enc=AESGCM(128) Mac=AEAD',
  'digest': None,
  'id': 50380847,
  'kea': 'kx-ecdhe',
  'name': 'ECDHE-RSA-AES128-GCM-SHA256',
  'protocol': 'TLSv1.2',
  'strength_bits': 128,
  'symmetric': 'aes-128-gcm'}]
```

Added in version 3.6.

SSLContext.set_default_verify_paths()

Carga un conjunto de certificados de «autoridad de certificación» (CA) por defecto desde una ruta del sistema de archivos definida al construir la biblioteca OpenSSL. Desafortunadamente, no hay una manera fácil de saber si este método tiene éxito: no se retorna ningún error si no se encuentran certificados. Sin embargo, cuando la biblioteca OpenSSL se proporciona como parte del sistema operativo, es probable que esté configurada correctamente.

SSLContext.set_ciphers(*ciphers*, */*)

Set the available ciphers for sockets created with this context. It should be a string in the OpenSSL cipher list format [https://docs.openssl.org/master/man1/ciphers/]. If no cipher can be selected (because compile-time options or other configuration forbids use of all the specified ciphers), an **`SSLError`** will be raised.

Nota

cuando se conecta, el método **`SSLSocket.cipher()`** de los sockets SSL dará el cifrado actualmente seleccionado.

OpenSSL 1.1.1 tiene suites de cifrado TLS 1.3 habilitadas por defecto. Las suites no se pueden desactivar con **`set_ciphers()`**.

SSLContext.set_alpn_protocols(*alpn_protocols*)

Especifica qué protocolos debe anunciar el socket durante el handshake SSL/TLS. Debe ser una lista de cadenas de caracteres ASCII, como `['http/1.1', 'spdy/2']`, ordenadas por preferencia. La selección de un protocolo ocurrirá durante el handshake, y se desarrollará de acuerdo con **RFC 7301** [https://datatracker.ietf.org/doc/html/rfc7301.html]. Después de un handshake exitoso, el método **`SSLSocket.selected_alpn_protocol()`** devolverá el protocolo acordado.

Este método lanzará **`NotImplementedError`** si **`HAS_ALPN`** es `False`.

Added in version 3.5.

SSLContext.set_npn_protocols(*npn_protocols*)

Especifica qué protocolos debe anunciar el socket durante el handshake SSL/TLS. Debe ser una lista de cadenas, como `['http/1.1', 'spdy/2']`, ordenadas por preferencia. La selección de un protocolo ocurrirá durante el handshake, y se desarrollará de acuerdo a la Negociación del Protocolo de la Capa de Aplicación [https://en.wikipedia.org/wiki/Application-Layer_Protocol_Negotiation]. Después de un handshake exitoso, el método **`SSLSocket.selected_npn_protocol()`** devolverá el protocolo acordado.

Este método lanzará **`NotImplementedError`** si **`HAS_NPN`** es `False`.

Added in version 3.3.

Obsoleto desde la versión 3.10: NPN ha sido reemplazada por ALPN

SSLContext.sni_callback

Registra una función callback que se llamará después de que el servidor SSL/TLS haya recibido el mensaje de diálogo TLS Client Hello cuando el cliente TLS especifique una indicación de nombre de servidor. El mecanismo de indicación de nombre de servidor se especifica en **RFC 6066** [https://datatracker.ietf.org/doc/html/rfc6066.html] sección 3 - Server Name Indication.

Sólo se puede establecer una función callback por `SSLContext`. Si *sni_callback* se establece como `None`, la función callback se desactiva. Si se llama a esta función una vez más, se desactivará la función callback registrada anteriormente.

La función callback será llamada con tres argumentos; el primero es el `ssl.SSLSocket`, el segundo es una cadena que representa el nombre del servidor con el que el cliente pretende comunicarse (o `None` si el TLS Client Hello no contiene un nombre de servidor) y el tercer argumento es el `SSLContext` original. El argumento del nombre del servidor es un texto. En el caso de los nombres de dominio internacionalizados, el nombre del servidor es un IDN etiqueta A (`"xn--pythn-mua.org"`).

Un uso típico de esta función callback es cambiar el atributo `SSLSocket.context` de `ssl.SSLSocket` por un nuevo objeto de tipo `SSLContext` que representa una cadena de certificados que coincide con el nombre del servidor.

Due to the early negotiation phase of the TLS connection, only limited methods and attributes are usable like `SSLSocket.selected_alpn_protocol()` and `SSLSocket.context`. The `SSLSocket.getpeercert()`, `SSLSocket.get_verified_chain()`, `SSLSocket.get_unverified_chain()` `SSLSocket.cipher()` and `SSLSocket.compression()` methods require that the TLS connection has progressed beyond the TLS Client Hello and therefore will not return meaningful values nor can they be called safely.

La función *sni_callback* debe devolver `None` para permitir que la negociación TLS continúe. Si se requiere un fallo TLS, se puede devolver una constante `ALERT_DESCRIPTION_*`. Otros valores de retorno resultarán en un error fatal TLS con **ALERT_DESCRIPTION_INTERNAL_ERROR**.

Si se lanza una excepción desde la función *sni_callback* la conexión TLS terminará con un mensaje de alerta TLS fatal `ALERT_DESCRIPTION_HANDSHAKE_FAILURE`.

Este método lanzará `NotImplementedError` si la biblioteca OpenSSL tenía definido OPENSSL_NO_TLSEXT cuando se construyó.

Added in version 3.7.

SSLContext.set_servername_callback(*server_name_callback*)

Se trata de una API heredada que se mantiene por compatibilidad con versiones anteriores. Cuando sea posible, debería utilizar `sni_callback` en su lugar. El *server_name_callback* dado es similar a *sni_callback*, excepto que cuando el nombre del servidor es un nombre de dominio internacionalizado codificado con IDN, el *server_name_callback* recibe una etiqueta U decodificada (`"pythön.org"`).

If there is a decoding error on the server name, the TLS connection will terminate with an `ALERT_DESCRIPTION_INTERNAL_ERROR` fatal TLS alert message to the client.

Added in version 3.4.

SSLContext.load_dh_params(*dhfile*, */*)

Carga los parámetros de generación de claves para el intercambio de claves Diffie-Hellman (DH). El uso del intercambio de claves DH mejora el secreto hacia adelante a expensas de recursos computacionales (tanto en el servidor como en el cliente). El parámetro *dhfile* debe ser la ruta de un archivo que contenga los parámetros DH en formato PEM.

Esta configuración no se aplica a los sockets de los clientes. También puede utilizar la opción **`OP_SINGLE_DH_USE`** para mejorar aún más la seguridad.

Added in version 3.3.

SSLContext.set_ecdh_curve(*curve_name*, */*)

Establece el nombre de la curva para el intercambio de claves Diffie-Hellman basado en la curva elíptica (ECDH). ECDH es significativamente más rápido que el DH normal, aunque podría decirse que es igual de seguro. El parámetro *curve_name* debe ser una cadena que describa una curva elíptica conocida, por ejemplo `prime256v1` para una curva ampliamente soportada.

Esta configuración no se aplica a los sockets de los clientes. También puede utilizar la opción **`OP_SINGLE_ECDH_USE`** para mejorar aún más la seguridad.

Este método no está disponible si **`HAS_ECDH`** es `False`.

Added in version 3.3.

Ver también

SSL/TLS & Perfect Forward Secrecy [https://vincent.bernat.ch/en/blog/2011-ssl-perfect-forward-secrecy]
Vincent Bernat.

SSLContext.wrap_socket(*sock*, *server_side=False*, *do_handshake_on_connect=True*, *suppress_ragged_eofs=True*, *server_hostname=None*, *session=None*)

Wrap an existing Python socket *sock* and return an instance of **`SSLContext.sslsocket_class`** (default **`SSLSocket`**). The returned SSL socket is tied to the context, its settings and certificates. *sock* must be a **`SOCK_STREAM`** socket; other socket types are unsupported.

El parámetro `server_side` es un booleano que identifica si se desea un comportamiento del lado del servidor o del lado del cliente en este socket.

Para los sockets del lado del cliente, la construcción del contexto es perezosa; si el socket subyacente no está conectado todavía, la construcción del contexto se realizará después de llamar a **`connect()`** en el socket. Para los sockets del lado del servidor, si el socket no tiene un par remoto, se asume que es un socket a la escucha, y la envoltura SSL del lado del servidor se realiza automáticamente en las conexiones del cliente aceptadas a través del método **`accept()`**. El método puede lanzar **`SSLError`**.

En las conexiones de cliente, el parámetro opcional *server_hostname* especifica el nombre del servicio al que nos estamos conectando. Esto permite que un único servidor aloje varios servicios basados en SSL con certificados distintos, de forma similar a los hosts virtuales HTTP. Al especificar *server_hostname* se producirá un **`ValueError`** si *server_side* es verdadero.

El parámetro `do_handshake_on_connect` especifica si se hace el handshake SSL automáticamente después de hacer un **`socket.connect()`**, o si el programa de aplicación lo llamará explícitamente, invocando el método **`SSLSocket.do_handshake()`**. Llamar explícitamente a **`SSLSocket.do_handshake()`** da al programa el control sobre el comportamiento de bloqueo de la E/S del socket involucrada en el handshake.

El parámetro `suppress_ragged_eofs` especifica cómo el método **`SSLSocket.recv()`** debe señalar los EOF inesperados desde el otro extremo de la conexión. Si se especifica como `True` (el valor por defecto), retorna un EOF normal (un objeto bytes vacío) en respuesta a los errores EOF inesperados que se produzcan desde el socket subyacente; si `False`, lanzará las excepciones al llamador.

session, véase `session`.

To wrap an `SSLSocket` in another **`SSLSocket`**, use `SSLContext.wrap_bio()`.

Distinto en la versión 3.5: Siempre permite pasar un server_hostname, incluso si OpenSSL no tiene SNI.

Distinto en la versión 3.6: Se agregó el argumento *session*.

Distinto en la versión 3.7: El método retorna una instancia de `SSLContext.sslsocket_class` en lugar de `SSLSocket` codificado de forma rígida.

SSLContext.sslsocket_class

The return type of `SSLContext.wrap_socket()`, defaults to `SSLSocket`. The attribute can be assigned to on instances of `SSLContext` in order to return a custom subclass of **`SSLSocket`**.

Added in version 3.7.

SSLContext.wrap_bio(*incoming*, *outgoing*, *server_side=False*, *server_hostname=None*, *session=None*)

Envuelve los objetos BIO *incoming* y *outgoing* y retorna una instancia de `SSLContext.sslobject_class` (por defecto `SSLObject`). Las rutinas SSL leerán los datos de entrada de la BIO entrante y escribirán los datos en la BIO saliente.

Los parámetros *server_side*, *server_hostname* y *session* tienen el mismo significado que en `SSLContext.wrap_socket()`.

Distinto en la versión 3.6: Se agregó el argumento *session*.

Distinto en la versión 3.7: El método retorna una instancia de

`SSLContext.sslobject_class` en lugar de `SSLObject` codificado de forma rígida.

SSLContext.sslobject_class

El tipo de retorno de `SSLContext.wrap_bio()`, por defecto es `SSLObject`. El atributo puede anularse en la instancia de la clase para devolver una subclase personalizada de **`SSLObject`**.

Added in version 3.7.

SSLContext.session_stats()

Get statistics about the SSL sessions created or managed by this context. A dictionary is returned which maps the names of each piece of information [https://docs.openssl.org/1.1.1/man3/SSL_CTX_sess_number/] to their numeric values. For example, here is the total number of hits and misses in the session cache since the context was created:

```
>>> stats = context.session_stats()
>>> stats['hits'], stats['misses']
(0, 0)
```

SSLContext.check_hostname

Si debe coincidir con el nombre de host del certificado de pares en `SSLSocket.do_handshake()`. El `verify_mode` del contexto debe establecerse en `CERT_OPTIONAL` o `CERT_REQUIRED`, y debe pasar *server_hostname* a `wrap_socket()` para que coincida con el nombre de host. La activación de la comprobación del nombre de host configura automáticamente **`verify_mode`** de `CERT_NONE` a **`CERT_REQUIRED`**. No se puede volver a establecer en **`CERT_NONE`** siempre que la comprobación del nombre de host esté habilitada. El protocolo `PROTOCOL_TLS_CLIENT` habilita la verificación del nombre de host de forma predeterminada. Con otros protocolos, la verificación del nombre de host debe habilitarse explícitamente.

Ejemplo:

```
import socket, ssl

context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = True
context.load_default_certs()

s = socket.socket(socket.AF_INET, socket.SOCK_STREA
ssl_sock = context.wrap_socket(s, server_hostname='
ssl_sock.connect(('www.verisign.com', 443))
```

Added in version 3.4.

Distinto en la versión 3.7: `verify_mode` ahora se cambia automáticamente a `CERT_REQUIRED` cuando la comprobación del hostname está activada y **`verify_mode`** es `CERT_NONE`. Anteriormente la misma operación habría fallado con un `ValueError`.

SSLContext.keylog_filename

Escribe las claves TLS en un archivo keylog, siempre que se genere o reciba material de claves. El archivo keylog está diseñado únicamente para fines de depuración. El formato del archivo está especificado por NSS y es utilizado por muchos analizadores de tráfico como Wireshark. El archivo de registro se abre en modo sólo añadir. Las escrituras se sincronizan entre hilos, pero no entre procesos.

Added in version 3.8.

SSLContext.maximum_version

A `TLSVersion` enum member representing the highest supported TLS version. The value defaults to `TLSVersion.MAXIMUM_SUPPORTED`. The attribute is read-only for protocols other than `PROTOCOL_TLS`, `PROTOCOL_TLS_CLIENT`, and `PROTOCOL_TLS_SERVER`.

Los atributos `maximum_version`, `minimum_version` y `SSLContext.options` afectan a las versiones SSL y TLS soportadas del contexto. La implementación no evita la combinación inválida. Por ejemplo, un contexto con `OP_NO_TLSv1_2` en **`options`** y **`maximum_version`** establecido en `TLSVersion.TLSv1_2` no podrá establecer una conexión TLS 1.2.

Added in version 3.7.

SSLContext.minimum_version

Igual que `SSLContext.maximum_version` excepto que es la versión más baja soportada o `TLSVersion.MINIMUM_SUPPORTED`.

Added in version 3.7.

SSLContext.num_tickets

Control the number of TLS 1.3 session tickets of a `PROTOCOL_TLS_SERVER` context. The setting has no impact on TLS 1.0 to 1.2 connections.

Added in version 3.8.

SSLContext.options

Un número entero que representa el conjunto de opciones SSL habilitadas en este contexto. El valor por defecto es `OP_ALL`, pero se pueden especificar otras opciones como `OP_NO_SSLv2` mediante la combinación OR.

Distinto en la versión 3.6: `SSLContext.options` retorna opciones `Options`:

```
>>> ssl.create_default_context().options
<Options.OP_ALL|OP_NO_SSLv3|OP_NO_SSLv2|OP_NO_COMPR
```

Obsoleto desde la versión 3.7: Esta opción es obsoleta desde OpenSSL 1.1.0, utilice en su lugar los nuevos `SSLContext.minimum_version` y `SSLContext.maximum_version`.

SSLContext.post_handshake_auth

Habilita la autenticación del cliente TLS 1.3 post-handshake. La autenticación post-handshake está deshabilitada por defecto y un servidor sólo puede solicitar un certificado de cliente TLS durante el handshake inicial. Cuando se habilita, un servidor puede solicitar un certificado de cliente TLS en cualquier momento después del handshake.

Cuando se activa en los sockets del lado del cliente, el cliente indica al servidor que soporta la autenticación post-handshake.

Cuando se activa en los sockets del lado del servidor, `SSLContext.verify_mode` debe establecerse también como `CERT_OPTIONAL` o `CERT_REQUIRED`. El intercambio real de certificados del cliente se retrasa hasta que se llama a `SSLSocket.verify_client_post_handshake()` y se realiza alguna E/S.

Added in version 3.8.

SSLContext.protocol

La versión del protocolo elegida cuando se construyó el contexto. Este atributo es de sólo lectura.

SSLContext.hostname_checks_common_name

Si `check_hostname` vuelve a verificar el nombre común del sujeto del certificado en ausencia de una extensión de nombre alternativo del sujeto (por defecto: true).

Added in version 3.7.

Distinto en la versión 3.10: The flag had no effect with OpenSSL before version 1.1.1l. Python 3.8.9, 3.9.3, and 3.10 include workarounds for previous versions.

SSLContext.security_level

An integer representing the security level [https://docs.openssl.org/master/man3/SSL_CTX_get_security_level/] for the context. This attribute is read-only.

Added in version 3.10.

SSLContext.verify_flags

Los indicadores para las operaciones de verificación de certificados. Se pueden establecer indicadores como **VERIFY_CRL_CHECK_LEAF** mediante la combinación OR. Por defecto, OpenSSL no requiere ni verifica las listas de revocación de certificados (CRL). Disponible sólo con la versión 0.9.8+ de openssl.

Added in version 3.4.

Distinto en la versión 3.6: **SSLContext.verify_flags** retorna opciones **VerifyFlags**:

```
>>> ssl.create_default_context().verify_flags
<VerifyFlags.VERIFY_X509_TRUSTED_FIRST: 32768>
```

SSLContext.verify_mode

Si se intenta verificar los certificados de otros pares y cómo comportarse si la verificación falla. Este atributo debe ser uno de los siguientes: **CERT_NONE**, **CERT_OPTIONAL** o **CERT_REQUIRED**.

Distinto en la versión 3.6: **SSLContext.verify_mode** retorna **VerifyMode** enum:

```
>>> ssl.create_default_context().verify_mode
<VerifyMode.CERT_REQUIRED: 2>
```

SSLContext.set_psk_client_callback(*callback*)

Enables TLS-PSK (pre-shared key) authentication on a client-side connection.

In general, certificate based authentication should be preferred over this method.

The parameter `callback` is a callable object with the signature: `def callback(hint: str | None) -> tuple[str | None, bytes]`. The `hint` parameter is an

optional identity hint sent by the server. The return value is a tuple in the form (client-identity, psk). Client-identity is an optional string which may be used by the server to select a corresponding PSK for the client. The string must be less than or equal to `256` octets when UTF-8 encoded. PSK is a bytes-like object representing the pre-shared key. Return a zero length PSK to reject the connection.

Setting `callback` to **`None`** removes any existing callback.

Nota

When using TLS 1.3:

- the `hint` parameter is always **`None`**.
- client-identity must be a non-empty string.

Example usage:

```
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
context.check_hostname = False
context.verify_mode = ssl.CERT_NONE
context.maximum_version = ssl.TLSVersion.TLSv1_2
context.set_ciphers('PSK')

# A simple lambda:
psk = bytes.fromhex('c0ffee')
context.set_psk_client_callback(lambda hint: (None,

# A table using the hint from the server:
psk_table = { 'ServerId_1': bytes.fromhex('c0ffee')
              'ServerId_2': bytes.fromhex('facade')
}
def callback(hint):
    return 'ClientId_1', psk_table.get(hint, b'')
context.set_psk_client_callback(callback)
```

This method will raise **`NotImplementedError`** if **`HAS_PSK`** is `False`.

Added in version 3.13.

SSLContext.set_psk_server_callback(*callback*, *identity_hint=None*)

Enables TLS-PSK (pre-shared key) authentication on a server-side connection.

In general, certificate based authentication should be preferred over this method.

The parameter `callback` is a callable object with the signature: `def callback(identity: str | None) -> bytes`. The `identity` parameter is an optional identity sent by the client which can be used to select a corresponding PSK. The return value is a bytes-like object representing the pre-shared key. Return a zero length PSK to reject the connection.

Setting `callback` to **`None`** removes any existing callback.

The parameter `identity_hint` is an optional identity hint string sent to the client. The string must be less than or equal to `256` octets when UTF-8 encoded.

Nota

When using TLS 1.3 the `identity_hint` parameter is not sent to the client.

Example usage:

```
context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.maximum_version = ssl.TLSVersion.TLSv1_2
context.set_ciphers('PSK')

# A simple lambda:
psk = bytes.fromhex('c0ffee')
context.set_psk_server_callback(lambda identity: ps

# A table using the identity of the client:
psk_table = { 'ClientId_1': bytes.fromhex('c0ffee')
```

```
                  'ClientId_2': bytes.fromhex('facade')
}
def callback(identity):
    return psk_table.get(identity, b'')
context.set_psk_server_callback(callback, 'ServerId
```

This method will raise **`NotImplementedError`** if **`HAS_PSK`** is `False`.

Added in version 3.13.

Certificados

En general, los certificados forman parte de un sistema de clave pública/clave privada. En este sistema, a cada *principal* (que puede ser una máquina, una persona o una organización) se le asigna una clave de cifrado única de dos partes. Una parte de la clave es pública y se llama *clave pública*; la otra parte se mantiene en secreto y se llama *clave privada*. Las dos partes están relacionadas, en el sentido de que si se cifra un mensaje con una de las partes, se puede descifrar con la otra parte, y **sólo** con la otra parte.

Un certificado contiene información sobre dos sujetos. Contiene el nombre de un *sujeto*, y la clave pública del sujeto. También contiene una declaración de un segundo titular, el *emisor*, de que el sujeto es quien dice ser, y de que ésta es efectivamente la clave pública del sujeto. La declaración del emisor está firmada con su clave privada, que sólo el emisor conoce. Sin embargo, cualquiera puede verificar la declaración del emisor encontrando la clave pública del emisor, descifrando la declaración con ella y comparándola con el resto de la información del certificado. El certificado también contiene información sobre el periodo de tiempo en el que es válido. Esto se expresa en dos campos, llamados *notBefore* y *notAfter*.

En el uso de certificados en Python, un cliente o servidor puede utilizar un certificado para demostrar quién es. El otro lado de una conexión de red también puede ser requerido para producir un certificado, y ese certificado puede ser validado a la satisfacción del cliente o servidor que requiere dicha validación. El intento de conexión puede configurarse para que lance una excepción si la validación falla. La validación se realiza automáticamente, por el subyacente framework OpenSSL; la aplicación no necesita preocuparse de su mecánica. Sin embargo, la aplicación normalmente necesita proporcionar conjuntos de certificados para permitir que este proceso tenga lugar.

Python utiliza archivos para contener certificados. Deben ser formateados como «PEM» (ver **RFC 1422** [https://datatracker.ietf.org/doc/html/rfc1422.html]), que es una forma codificada en base-64 envuelta con una línea de cabecera y una línea de pie de página:

```
-----BEGIN CERTIFICATE-----
... (certificate in base64 PEM encoding) ...
-----END CERTIFICATE-----
```

Cadenas de certificados

Los archivos de Python que contienen certificados pueden contener una secuencia de certificados, a veces llamada *cadena de certificados*. Esta cadena debería empezar con el certificado específico para el principal que «es» el cliente o servidor, y luego el certificado para el emisor de ese certificado, y luego el certificado para el emisor de *ese* certificado, y así sucesivamente hasta llegar a un certificado que es *auto-firmado*, es decir, un certificado que tiene el mismo sujeto y emisor, a veces llamado *certificado raíz*. Los certificados sólo deben concatenarse en el archivo de certificados. Por ejemplo, supongamos que tenemos una cadena de tres certificados, desde el certificado de nuestro servidor al certificado de la autoridad de certificación que firmó nuestro certificado del servidor, hasta el certificado raíz de la agencia que emitió el certificado de la autoridad de certificación:

```
-----BEGIN CERTIFICATE-----
... (certificate for your server)...
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
... (the certificate for the CA)...
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
... (the root certificate for the CA's issuer)...
```

```
-----END CERTIFICATE-----
```

Certificados CA

Si se requiere la validación del certificado del otro lado de la conexión, se necesita proporcionar un archivo «CA certs», con las cadenas de certificados para cada emisor en el que se está dispuesto a confiar. De nuevo, este archivo sólo contiene estas cadenas concatenadas. Para la validación, Python utilizará la primera cadena que encuentre en el archivo que coincida. El archivo de certificados de la plataforma se puede utilizar llamando a **`SSLContext.load_default_certs()`**, esto se hace automáticamente con **`create_default_context()`**.

Clave y certificado combinados

Often the private key is stored in the same file as the certificate; in this case, only the `certfile` parameter to **`SSLContext.load_cert_chain()`** needs to be passed. If the private key is stored with the certificate, it should come before the first certificate in the certificate chain:

```
-----BEGIN RSA PRIVATE KEY-----
... (private key in base64 encoding) ...
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
... (certificate in base64 PEM encoding) ...
-----END CERTIFICATE-----
```

Certificados auto-firmados

Si va a crear un servidor que proporcione servicios de conexión encriptada SSL, necesitará adquirir un certificado para ese servicio. Hay muchas formas de adquirir los certificados adecuados, como comprar uno a una autoridad de certificación. Otra práctica común es generar un certificado auto-firmado. La forma más sencilla de hacerlo es con el paquete OpenSSL, utilizando algo como lo siguiente:

```
% openssl req -new -x509 -days 365 -nodes -out cert.pem
```

```
Generating a 1024 bit RSA private key
.......++++++
.............................++++++
writing new private key to 'cert.pem'
-----
You are about to be asked to enter information that will
into your certificate request.
What you are about to enter is what is called a Distingu
There are quite a few fields but you can leave some blan
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:MyState
Locality Name (eg, city) []:Some City
Organization Name (eg, company) [Internet Widgits Pty Lt
Organizational Unit Name (eg, section) []:My Group
Common Name (eg, YOUR name) []:myserver.mygroup.myorgani
Email Address []:ops@myserver.mygroup.myorganization.com
%
```

La desventaja de un certificado auto-firmado es que es su propio certificado raíz, y nadie más lo tendrá en su caché de certificados raíz conocidos (y de confianza).

Ejemplos

Pruebas de compatibilidad con SSL

Para comprobar la presencia de soporte SSL en una instalación de Python, el código del usuario debe utilizar el siguiente modismo:

```
try:
    import ssl
except ImportError:
    pass
else:
    ...  # do something that requires SSL support
```

Operación del lado del cliente

Este ejemplo crea un contexto SSL con la configuración de seguridad recomendada para los sockets del cliente, incluyendo la verificación automática de certificados:

```
>>> context = ssl.create_default_context()
```

Si prefieres ajustar la configuración de seguridad tú mismo, puedes crear un contexto desde cero (pero ten en cuenta que podrías no acertar con la configuración):

```
>>> context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
>>> context.load_verify_locations("/etc/ssl/certs/ca-bur
```

(este fragmento asume que tu sistema operativo coloca un paquete de todos los certificados CA en `/etc/ssl/certs/ca-bundle.crt`; si no es así, obtendrá un error y tendrá que ajustar la ubicación)

El protocolo **`PROTOCOL_TLS_CLIENT`** configura el contexto para la validación de certificados y la verificación del hostname. **`verify_mode`** se establece en **`CERT_REQUIRED`** y **`check_hostname`** se establece en `True`. Todos los demás protocolos crean contextos SSL con valores predeterminados inseguros.

Cuando se utiliza el contexto para conectarse a un servidor, **`CERT_REQUIRED`** y **`check_hostname`** validan el certificado del servidor: se asegura de que el certificado del servidor se ha firmado con uno de los certificados de la CA, se comprueba que la firma es correcta y se verifican otras propiedades como la validez y la identidad del hostname:

```
>>> conn = context.wrap_socket(socket.socket(socket.AF_I
...                            server_hostname="www.pyth
>>> conn.connect(("www.python.org", 443))
```

A continuación, puede obtener el certificado:

```
>>> cert = conn.getpeercert()
```

La inspección visual muestra que el certificado sí identifica el servicio deseado (es decir, el host HTTPS `www.python.org`):

```
>>> pprint.pprint(cert)
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/DigiCertSHA2
 'crlDistributionPoints': ('http://crl3.digicert.com/sha
                           'http://crl4.digicert.com/sha
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'DigiCert Inc'),),
            (('organizationalUnitName', 'www.digicert.co
            (('commonName', 'DigiCert SHA2 Extended Vali
 'notAfter': 'Sep  9 12:00:00 2016 GMT',
 'notBefore': 'Sep  5 00:00:00 2014 GMT',
 'serialNumber': '01BB6F00122B177F36CAB49CEA8B6B26',
 'subject': ((('businessCategory', 'Private Organization
             (('1.3.6.1.4.1.311.60.2.1.3', 'US'),),
             (('1.3.6.1.4.1.311.60.2.1.2', 'Delaware'),)
             (('serialNumber', '3359300'),),
             (('streetAddress', '16 Allen Rd'),),
             (('postalCode', '03894-4801'),),
             (('countryName', 'US'),),
             (('stateOrProvinceName', 'NH'),),
             (('localityName', 'Wolfeboro'),),
             (('organizationName', 'Python Software Foun
             (('commonName', 'www.python.org'),)),
 'subjectAltName': (('DNS', 'www.python.org'),
                    ('DNS', 'python.org'),
                    ('DNS', 'pypi.org'),
                    ('DNS', 'docs.python.org'),
                    ('DNS', 'testpypi.org'),
                    ('DNS', 'bugs.python.org'),
                    ('DNS', 'wiki.python.org'),
                    ('DNS', 'hg.python.org'),
                    ('DNS', 'mail.python.org'),
                    ('DNS', 'packaging.python.org'),
                    ('DNS', 'pythonhosted.org'),
                    ('DNS', 'www.pythonhosted.org'),
                    ('DNS', 'test.pythonhosted.org'),
                    ('DNS', 'us.pycon.org'),
                    ('DNS', 'id.python.org')),
 'version': 3}
```

Ahora que se ha establecido el canal SSL y se ha verificado el certificado, se puede proceder a hablar con el servidor:

```
>>> conn.sendall(b"HEAD / HTTP/1.0\r\nHost: linuxfr.org\
>>> pprint.pprint(conn.recv(1024).split(b"\r\n"))
[b'HTTP/1.1 200 OK',
 b'Date: Sat, 18 Oct 2014 18:27:20 GMT',
 b'Server: nginx',
 b'Content-Type: text/html; charset=utf-8',
 b'X-Frame-Options: SAMEORIGIN',
 b'Content-Length: 45679',
 b'Accept-Ranges: bytes',
 b'Via: 1.1 varnish',
 b'Age: 2188',
 b'X-Served-By: cache-lcy1134-LCY',
 b'X-Cache: HIT',
 b'X-Cache-Hits: 11',
 b'Vary: Cookie',
 b'Strict-Transport-Security: max-age=63072000; includeS
 b'Connection: close',
 b'',
 b'']
```

Vea la discusión sobre Consideraciones de seguridad más abajo.

Operación del lado del servidor

Para el funcionamiento del servidor, normalmente necesitarás tener un certificado de servidor y una clave privada, cada uno en un archivo. Primero crearás un contexto que contenga la clave y el certificado, para que los clientes puedan comprobar tu autenticidad. Entonces abrirás un socket, lo enlazarás a un puerto, llamarás a **`listen()`** en él, y empezarás a esperar a que los clientes se conecten:

```
import socket, ssl
```

```
context = ssl.create_default_context(ssl.Purpose.CLIENT_
context.load_cert_chain(certfile="mycertfile", keyfile="

bindsocket = socket.socket()
bindsocket.bind(('myaddr.example.com', 10023))
bindsocket.listen(5)
```

Cuando un cliente se conecta, llamarás a **`accept()`** en el socket para obtener el nuevo socket del otro extremo, y utilizarás el método **`SSLContext.wrap_socket()`** del contexto para crear un socket SSL del lado del servidor para la conexión:

```
while True:
    newsocket, fromaddr = bindsocket.accept()
    connstream = context.wrap_socket(newsocket, server_s
    try:
        deal_with_client(connstream)
    finally:
        connstream.shutdown(socket.SHUT_RDWR)
        connstream.close()
```

Entonces leerás los datos del `connstream` y harás algo con ellos hasta que hayas terminado con el cliente (o el cliente haya terminado contigo):

```
def deal_with_client(connstream):
    data = connstream.recv(1024)
    # empty data means the client is finished with us
    while data:
        if not do_something(connstream, data):
            # we'll assume do_something returns False
            # when we're finished with client
            break
        data = connstream.recv(1024)
    # finished with client
```

Y volver a escuchar nuevas conexiones de clientes (por supuesto, un servidor real probablemente manejaría cada conexión de cliente en un hilo separado, o pondría los sockets en modo no-bloqueo y usaría un bucle de eventos).

Notas sobre los sockets no bloqueantes

Los sockets SSL se comportan de forma ligeramente diferente a los sockets normales en modo no bloqueante. Cuando se trabaja con sockets no bloqueantes, hay varias cosas que hay que tener en cuenta:

- La mayoría de los métodos de `SSLSocket` lanzarán `SSLWantWriteError` o `SSLWantReadError` en lugar de `BlockingIOError` si una operación de E/S se bloquea. `SSLWantReadError` se lanzará si es necesaria una operación de lectura en el socket subyacente, y `SSLWantWriteError` para una operación de escritura en el socket subyacente. Tenga en cuenta que los intentos de *escribir* en un socket SSL pueden requerir *leer* del socket subyacente primero, y los intentos de *leer* del socket SSL pueden requerir una *escritura* previa en el socket subyacente.

  Distinto en la versión 3.5: En versiones anteriores de Python, el método `SSLSocket.send()` devolvía cero en lugar de lanzar `SSLWantWriteError` o `SSLWantReadError`.

- Llamar a `select()` le indica que se puede leer (o escribir) en el socket a nivel del SO, pero no implica que haya suficientes datos en la capa superior SSL. Por ejemplo, puede que sólo haya llegado una parte de una trama SSL. Por lo tanto, debe estar preparado para manejar los fallos de `SSLSocket.recv()` y `SSLSocket.send()`, y re-intentar después de otra llamada a `select()`.

- Por el contrario, dado que la capa SSL tiene su propia estructura, un socket SSL puede tener datos disponibles para leer sin que `select()` lo sepa. Por lo tanto, debería llamar primero a `SSLSocket.recv()` para drenar cualquier dato potencialmente disponible, y luego sólo bloquear en una llamada a `select()` si todavía es necesario.

  (por supuesto, se aplican disposiciones similares cuando se utilizan otras primitivas como `poll()`, o las del módulo `selectors`)

- El handshake SSL en sí mismo será no bloqueante: el método `SSLSocket.do_handshake()` tiene que ser re-intentado hasta que regrese con éxito. Aquí hay una sinopsis usando `select()` para esperar la disponibilidad del socket:

```
while True:
    try:
        sock.do_handshake()
        break
    except ssl.SSLWantReadError:
        select.select([sock], [], [])
    except ssl.SSLWantWriteError:
        select.select([], [sock], [])
```

Ver también

The `asyncio` module supports non-blocking SSL sockets and provides a higher level Streams API. It polls for events using the `selectors` module and handles `SSLWantWriteError`, `SSLWantReadError` and `BlockingIOError` exceptions. It runs the SSL handshake asynchronously as well.

Memory BIO support

Added in version 3.5.

Desde que se introdujo el módulo SSL en Python 2.6, la clase `SSLSocket` ha proporcionado dos áreas de funcionalidad relacionadas pero distintas:

- Manejo del protocolo SSL
- E/S de red

La API de E/S de red es idéntica a la proporcionada por `socket.socket`, de la que también hereda `SSLSocket`. Esto permite que un socket SSL sea utilizado como un reemplazo de un socket normal, haciendo que sea muy fácil añadir soporte SSL a una aplicación existente.

La combinación del manejo del protocolo SSL y la E/S de red suele funcionar bien, pero hay algunos casos en los que no es así. Un ejemplo son los frameworks de E/S asíncronos que quieren utilizar un modelo de multiplexación de E/S diferente al modelo «selección/consulta de un descriptor de archivo» (basado en la preparación) que es asumido por `socket.socket` y por las rutinas internas de E/S de socket de OpenSSL. Esto es principalmente relevante para plataformas como Windows donde este modelo no es eficiente. Para este propósito, se proporciona una variante de ámbito reducido de `SSLSocket` llamada `SSLObject`.

class ssl.SSLObject

Una variante de alcance reducido de `SSLSocket` que representa una instancia del protocolo SSL que no contiene ningún método de E/S de red. Esta clase suele ser utilizada por los autores de frameworks que quieren implementar E/S asíncrona para SSL a través de búfers de memoria.

Esta clase implementa una interfaz sobre un objeto SSL de bajo nivel como el implementado por OpenSSL. Este objeto captura el estado de una conexión SSL pero no proporciona ninguna E/S de red en sí misma. La E/S debe realizarse a través de objetos «BIO» separados que son la capa de abstracción de E/S de OpenSSL.

Esta clase no tiene un constructor público. Se debe crear una instancia `SSLObject` utilizando el método `wrap_bio()`. Este método creará la instancia `SSLObject` y la vinculará a un par de BIOs. El BIO *de entrada* se utiliza para pasar los datos de Python a la instancia del protocolo SSL, mientras que el BIO *de salida* se utiliza para pasar los datos a la inversa.

Los siguientes métodos son disponibles:

- `context`
- `server_side`
- `server_hostname`
- `session`

- **`session_reused`**
- **`read()`**
- **`write()`**
- **`getpeercert()`**
- **`get_verified_chain()`**
- **`get_unverified_chain()`**
- **`selected_alpn_protocol()`**
- **`selected_npn_protocol()`**
- **`cipher()`**
- **`shared_ciphers()`**
- **`compression()`**
- **`pending()`**
- **`do_handshake()`**
- **`verify_client_post_handshake()`**
- **`unwrap()`**
- **`get_channel_binding()`**
- **`version()`**

En comparación con **`SSLSocket`**, este objeto carece de las siguientes características:

- Cualquier forma de E/S de red; `recv()` y `send()` leen y escriben sólo en los búfers subyacentes de **`MemoryBIO`**.
- No existe la maquinaria *do_handshake_on_connect*. Siempre hay que llamar manualmente a **`do_handshake()`** para iniciar el handshake.
- No hay manejo de *suppress_ragged_eofs*. Todas las condiciones de fin de archivo que violan el protocolo se reportan a través de la excepción **`SSLEOFError`**.
- La llamada al método **`unwrap()`** no retorna nada, a diferencia de lo que ocurre con un socket SSL, que retorna el socket subyacente.
- La función callback *server_name_callback* pasada a **`SSLContext.set_servername_callback()`** obtendrá una instancia de **`SSLObject`** en lugar de una instancia de **`SSLSocket`** como primer parámetro.

Algunas notas relacionadas con el uso de **`SSLObject`**:

- Toda la E/S en un **`SSLObject`** es non-blocking. Esto significa que, por ejemplo, `read()` lanzará un `SSLWantReadError` si necesita más datos de los que la BIO entrante tiene disponibles.

Distinto en la versión 3.7: `SSLObject` instances must be created with `wrap_bio()`. In earlier versions, it was possible to create instances directly. This was never documented or officially supported.

Un SSLObject se comunica con el mundo exterior utilizando búfers de memoria. La clase `MemoryBIO` proporciona un búfer de memoria que puede ser utilizado para este propósito. Envuelve un objeto BIO (Basic IO) de memoria de OpenSSL:

class ssl.MemoryBIO

Un búfer de memoria que se puede utilizar para pasar datos entre Python y una instancia del protocolo SSL.

pending

Retorna el número de bytes que se encuentran actualmente en la memoria búfer.

eof

Un booleano que indica si la memoria BIO es actual en la posición de fin de archivo.

read(*n=-1*, */*)

Lee hasta *n* bytes del búfer de memoria. Si *n* no se especifica o es negativo, se retornan todos los bytes.

write(*buf*, */*)

Escribe los bytes de *buf* en la memoria BIO. El argumento *buf* debe ser un objeto que soporte el protocolo de búfer.

El valor de retorno es el número de bytes escritos, que siempre es igual a la longitud de *buf*.

write_eof()

Escribe un marcador EOF en la memoria BIO. Después de llamar a este método, es ilegal llamar a `write()`. El atributo `eof` se convertirá en verdadero después de que se hayan leído todos los datos que hay actualmente en el búfer.

Sesión SSL

Added in version 3.6.

class ssl.SSLSession

Objeto sesión usado por `session`.

id

time

timeout

ticket_lifetime_hint

has_ticket

Consideraciones de seguridad

Los mejores valores por defecto

Para el **uso en el cliente**, si no tiene ningún requisito especial para su política de seguridad, es muy recomendable que utilice la función `create_default_context()` para crear su contexto SSL. Cargará los certificados CA de confianza del sistema, habilitará la validación de certificados y la comprobación del hostname, e intentará elegir una configuración de protocolo y cifrado razonablemente segura.

Por ejemplo, así es como se utiliza la clase `smtplib.SMTP` para crear una conexión segura y de confianza con un servidor SMTP:

```
>>> import ssl, smtplib
>>> smtp = smtplib.SMTP("mail.python.org", port=587)
>>> context = ssl.create_default_context()
>>> smtp.starttls(context=context)
(220, b'2.0.0 Ready to start TLS')
```

Si se necesita un certificado de cliente para la conexión, se puede añadir con `SSLContext.load_cert_chain()`.

Por el contrario, si crea el contexto SSL llamando usted mismo al constructor `SSLContext`, no tendrá activada por defecto la validación de certificados ni la comprobación de hostname. Si lo hace, lea los párrafos siguientes para conseguir un buen nivel de seguridad.

Ajustes manuales

Verificación de certificados

When calling the `SSLContext` constructor directly, `CERT_NONE` is the default. Since it does not authenticate the other peer, it can be insecure, especially in client mode where most of the time you would like to ensure the authenticity of the server you're talking to. Therefore, when in client mode, it is highly recommended to use `CERT_REQUIRED`. However, it is in itself not sufficient; you also have to check that the server certificate, which can be obtained by calling `SSLSocket.getpeercert()`, matches the desired service. For many protocols and applications, the service can be identified by the hostname. This common check is automatically performed when `SSLContext.check_hostname` is enabled.

Distinto en la versión 3.7: Hostname matchings is now performed by OpenSSL. Python no longer uses `match_hostname()`.

En el modo servidor, si quiere autenticar a sus clientes utilizando la capa SSL (en lugar de utilizar un mecanismo de autenticación de nivel superior), también tendrá que especificar `CERT_REQUIRED` y comprobar de forma similar el certificado del cliente.

Versiones del protocolo

Las versiones 2 y 3 de SSL se consideran inseguras y, por lo tanto, su uso es peligroso. Si desea la máxima compatibilidad entre clientes y servidores, se recomienda utilizar **`PROTOCOL_TLS_CLIENT`** o **`PROTOCOL_TLS_SERVER`** como versión del protocolo. SSLv2 y SSLv3 están desactivados por defecto.

```
>>> client_context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLI
>>> client_context.minimum_version = ssl.TLSVersion.TLSv
>>> client_context.maximum_version = ssl.TLSVersion.TLSv
```

The SSL context created above will only allow TLSv1.3 and later (if supported by your system) connections to a server. **`PROTOCOL_TLS_CLIENT`** implies certificate validation and hostname checks by default. You have to load certificates into the context.

Selección de cifrado

If you have advanced security requirements, fine-tuning of the ciphers enabled when negotiating a SSL session is possible through the **`SSLContext.set_ciphers()`** method. Starting from Python 3.2.3, the ssl module disables certain weak ciphers by default, but you may want to further restrict the cipher choice. Be sure to read OpenSSL’s documentation about the cipher list format [https://docs.openssl.org/1.1.1/man1/ciphers/#cipher-list-format]. If you want to check which ciphers are enabled by a given cipher list, use **`SSLContext.get_ciphers()`** or the `openssl ciphers` command on your system.

Multiprocesamiento

If using this module as part of a multi-processed application (using, for example the **`multiprocessing`** or **`concurrent.futures`** modules), be aware that OpenSSL’s internal random number generator does not properly handle forked processes. Applications must change the PRNG state of the parent process if they use any SSL feature with **`os.fork()`**. Any successful call of **`RAND_add()`** or **`RAND_bytes()`** is sufficient.

TLS 1.3

Added in version 3.7.

Python tiene soporte provisional y experimental para TLS 1.3 con OpenSSL 1.1.1. El nuevo protocolo se comporta de forma ligeramente diferente a la versión anterior de TLS/SSL. Algunas de las nuevas características de TLS 1.3 aún no están disponibles.

- TLS 1.3 utiliza un conjunto disjunto de suites de cifrado. Todas las suites de cifrado AES-GCM y ChaCha20 están habilitadas por defecto. El método `SSLContext.set_ciphers()` aún no puede habilitar o deshabilitar ningún cifrado de TLS 1.3, pero `SSLContext.get_ciphers()` los retorna.
- Los tickets de sesión ya no se envían como parte del handshake inicial y se manejan de forma diferente. `SSLSocket.session` y `SSLSession` no son compatibles con TLS 1.3.
- Los certificados del lado del cliente ya no se verifican durante el handshake inicial. Un servidor puede solicitar un certificado en cualquier momento. Los clientes procesan las solicitudes de certificados mientras envían o reciben datos de la aplicación desde el servidor.
- Las funciones de TLS 1.3, como los datos anticipados, la solicitud de certificado de cliente TLS diferida, la configuración del algoritmo de firma y la repetición de claves, aún no son compatibles.

Ver también

Clase `socket.socket`
Documentación de la clase `socket` subyacente

SSL/TLS Strong Encryption: An Introduction [https://httpd.apache.org/docs/trunk/en/ssl/ssl_intro.html]
Introducción de la documentación del servidor HTTP Apache

RFC 1422: Privacy Enhancement for Internet Electronic

Mail: Part II: Certificate-Based Key Management **[https://datatracker.ietf.org/doc/html/rfc1422.html]**
Steve Kent

RFC 4086: Randomness Requirements for Security **[https://datatracker.ietf.org/doc/html/rfc4086.html]**
Donald E. Eastlake, Jeffrey I. Schiller, Steve Crocker

RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile **[https://datatracker.ietf.org/doc/html/rfc5280.html]**
David Cooper et al.

RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2 **[https://datatracker.ietf.org/doc/html/rfc5246.html]**
Tim Dierks and Eric Rescorla.

RFC 6066: Transport Layer Security (TLS) Extensions **[https://datatracker.ietf.org/doc/html/rfc6066.html]**
Donald E. Eastlake

IANA TLS: Transport Layer Security (TLS) Parameters **[https://www.iana.org/assignments/tls-parameters/tls-parameters.xml]**
IANA

RFC 7525: Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) **[https://datatracker.ietf.org/doc/html/rfc7525.html]**
IETF

Mozilla’s Server Side TLS recommendations **[https://wiki.mozilla.org/Security/Server_Side_TLS]**
Mozilla

`select` — Waiting for I/O completion

This module provides access to the `select()` and `poll()` functions available in most operating systems, `devpoll()` available on Solaris and derivatives, `epoll()` available on Linux 2.5+ and `kqueue()` available on most BSD. Note that on Windows, it only works for sockets; on other operating systems, it also works for other file types (in particular, on Unix, it works on pipes). It cannot be used on regular files to determine whether a file has grown since it was last read.

Nota

The `selectors` module allows high-level and efficient I/O multiplexing, built upon the `select` module primitives. Users are encouraged to use the `selectors` module instead, unless they want precise control over the OS-level primitives used.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

El módulo define lo siguiente:

exception select.error

Un alias en desuso de `OSError`.

Distinto en la versión 3.3: Siguiente **PEP 3151** [https://peps.python.org/pep-3151/], esta clase se convirtió en un alias de `OSError`.

select.devpoll()

(Solo se admite en Solaris y derivados). Retorna un objeto de sondeo `/dev/poll`; vea la sección Objetos de sondeo /dev/poll a continuación para conocer los métodos admitidos por los objetos *devpoll*.

`devpoll()` objects are linked to the number of file descriptors allowed at the time of instantiation. If your program reduces this value, **`devpoll()`** will fail. If your program increases this value, **`devpoll()`** may return an incomplete list of active file descriptors.

El nuevo descriptor del archivo es non-inheritable.

Added in version 3.3.

Distinto en la versión 3.4: El nuevo descriptor de archivo ahora no es heredable.

select.epoll(*sizehint=-1*, *flags=0*)

(Solo se admite en Linux 2.5.44 y versiones posteriores). Retorna un objeto de sondeo de borde, que se puede usar como interfaz de disparo de nivel o de borde para eventos de E/S.

sizehint informs epoll about the expected number of events to be registered. It must be positive, or `-1` to use the default. It is only used on older systems where **`epoll_create1()`** is not available; otherwise it has no effect (though its value is still checked).

flags está en desuso y se ignora por completo. Sin embargo, cuando se proporciona, su valor debe ser `0` o `select.EPOLL_CLOEXEC`, de lo contrario, se lanzará `OSError`.

Consulte la sección Objetos de sondeo de Edge y Level Trigger (epoll) a continuación para conocer los métodos admitidos por los objetos epolling.

Los objetos `epoll` admiten el protocolo *context management*: cuando se usa en una declaración **`with`**, el nuevo descriptor de archivo se cierra automáticamente al final del bloque.

El nuevo descriptor del archivo es non-inheritable.

Distinto en la versión 3.3: Se agregó el parámetro *flags*.

Distinto en la versión 3.4: Se agregó soporte para la declaración **`with`**. El nuevo descriptor de archivo ahora no es heredable.

Obsoleto desde la versión 3.4: El parámetro *flags*. `select.EPOLL_CLOEXEC` se usa por defecto ahora. Use **`os.set_inheritable()`** para hacer que el descriptor de archivo sea heredable.

select.poll()

(No es compatible con todos los sistemas operativos). Retorna un objeto de sondeo, que admite registrar y anular el registro de descriptores de archivo, y luego sondearlos para eventos de I/O; vea la sección Sondeo de objetos a continuación para conocer los métodos admitidos por los objetos de sondeo.

select.kqueue()

(Solo se admite en BSD). Retorna un objeto de cola del kernel; vea la sección Objetos Kqueue a continuación para conocer los métodos admitidos por los objetos kqueue.

El nuevo descriptor del archivo es non-inheritable.

Distinto en la versión 3.4: El nuevo descriptor de archivo ahora no es heredable.

select.kevent(*ident*, *filter=KQ_FILTER_READ*, *flags=KQ_EV_ADD*, *fflags=0*, *data=0*, *udata=0*)

(Solo se admite en BSD). Retorna un objeto de evento del kernel; vea la sección Objetos Kevent a continuación para conocer los métodos admitidos por los objetos kevent.

select.select(*rlist*, *wlist*, *xlist*, *timeout=None*)

This is a straightforward interface to the Unix `select()` system call. The first three arguments are iterables of "waitable objects": either integers representing file descriptors or objects with a parameterless method named `fileno()` returning such an integer:

- *rlist*: espera hasta que esté listo para leer
- *wlist*: espera hasta que esté listo para escribir
- *xlist*: espera una «condición excepcional» (consulte la página del manual para ver lo que su sistema considera tal condición)

Empty iterables are allowed, but acceptance of three empty iterables is platform-dependent. (It is known to work on Unix but not on Windows.) The optional *timeout* argument specifies a time-out as a floating-point number in seconds. When the *timeout* argument is omitted or `None`, the function blocks until at least one file descriptor is ready. A time-out value of zero specifies a poll and never blocks.

El valor de retorno es un triple de listas de objetos que están listos: subconjuntos de los primeros tres argumentos. Cuando se alcanza el tiempo de espera sin que esté listo un descriptor de archivo, se retornan tres listas vacías.

Entre los tipos de objetos aceptables en los iterables se encuentran Python objetos de archivo (por ejemplo, `sys.stdin`, u objetos retornados por `open()` o `os.popen()`), objetos de socket retornados por `socket.socket()`. También puede definir una clase *wrapper* usted mismo, siempre que tenga un método `fileno()` apropiado (que realmente retorne un descriptor de archivo, no solo un entero aleatorio).

Nota

File objects on Windows are not acceptable, but sockets are. On Windows, the underlying `select()` function is provided by the WinSock library, and does not handle file descriptors that don't originate from WinSock.

Distinto en la versión 3.5: La función ahora se vuelve a intentar con un tiempo de espera (*timeout*) recalculado cuando se interrumpe por una señal, excepto si el controlador de señal genera una excepción (ver **PEP 475** [https://peps.python.org/pep-0475/] para la justificación), en lugar de generar **`InterruptedError`**.

select.PIPE_BUF

The minimum number of bytes which can be written without blocking to a pipe when the pipe has been reported as ready for writing by **`select()`**, **`poll()`** or another interface in this module. This doesn't apply to other kinds of file-like objects such as sockets.

Este valor es garantizado por POSIX para ser menor a 512.

Availability: Unix

Added in version 3.2.

Objetos de sondeo `/dev/poll`

Solaris and derivatives have `/dev/poll`. While **`select()`** is *O*(*highest file descriptor*) and **`poll()`** is *O*(*number of file descriptors*), `/dev/poll` is *O*(*active file descriptors*).

`/dev/poll` behaviour is very close to the standard **`poll()`** object.

devpoll.close()

Cierra el descriptor de archivo del objeto de sondeo.

Added in version 3.4.

devpoll.closed

`True` si el objeto de sondeo está cerrado.

Added in version 3.4.

devpoll.fileno()

Retorna el número de descriptor de archivo del objeto de sondeo.

Added in version 3.4.

devpoll.register(*fd*[, *eventmask*])

Registra un descriptor de archivo con el objeto de sondeo. Las futuras llamadas al método `poll()` comprobarán si el descriptor del archivo tiene algún evento I/O pendiente. *fd* puede ser un entero o un objeto con un método `fileno()` que retorna un entero. Los objetos de archivo implementan **`Fileno()`**, por lo que también pueden usarse como argumento.

eventmask is an optional bitmask describing the type of events you want to check for. The constants are the same as with **`poll()`** object. The default value is a combination of the constants **`POLLIN`**, **`POLLPRI`**, and **`POLLOUT`**.

Advertencia

Registering a file descriptor that's already registered is not an error, but the result is undefined. The appropriate action is to unregister or modify it first. This is an important difference compared with **`poll()`**.

devpoll.modify(*fd*[, *eventmask*])

This method does an `unregister()` followed by a `register()`. It is (a bit) more efficient than doing the same explicitly.

devpoll.unregister(*fd*)

Elimina un descriptor de archivo que está siendo rastreado por un objeto de sondeo. Al igual que el método `register()`, *fd* puede ser un entero o un objeto con un método `fileno()` que retorna un entero.

Al intentar eliminar un descriptor de archivo que nunca se registró se ignora de forma segura.

devpoll.poll([*timeout*])

Sondea el conjunto de descriptores de archivos registrados y retorna una lista posiblemente vacía que contiene tuplas de dos elementos `(fd, event)` para los descriptores que tienen eventos o errores para informar. *fd* es el descriptor de archivo, y *event* es una máscara de bits con bits configurados para los eventos informados para ese descriptor — **`POLLIN`** para entrada en espera, **`POLLOUT`** para indicar que se puede escribir en el descriptor, y así sucesivamente. Una lista vacía indica que se agotó el tiempo de espera de la llamada y que ningún descriptor de archivo tuvo eventos para informar. Si se proporciona *timeout*, especifica el período de tiempo en milisegundos que el sistema esperará por los eventos antes de regresar. Si se omite *timeout*, -1 o **`None`**, la llamada se bloqueará hasta que haya un evento para este objeto de sondeo.

Distinto en la versión 3.5: La función ahora se vuelve a intentar con un tiempo de espera (*timeout*) recalculado cuando se interrumpe por una señal, excepto si el controlador de señal genera una excepción (ver **PEP 475** [https://peps.python.org/pep-0475/] para la justificación), en lugar de generar **`InterruptedError`**.

Objetos de sondeo de *Edge* y *Level Trigger* (*epoll*)

https://linux.die.net/man/4/epoll

eventmask

Constante	**Significado**
`EPOLLIN`	Disponible para lectura
`EPOLLOUT`	Disponible para escritura
`EPOLLPRI`	Urgente para lectura
`EPOLLERR`	Condición de error ocurrió en la asociación. fd

EPOLLHUP	Se colgó en la asociación. fd
EPOLLET	Establece el comportamiento en *Edge Trigger*, el valor predeterminado es *Level Trigger*
EPOLLONESHOT	Establece el comportamiento en *one-shot*. Después de que se retira un evento, el fd se deshabilita internamente
EPOLLEXCLUSIVE	Despierta solo un objeto epoll cuando el fd asociado tiene un evento. El valor predeterminado (si este flag no está configurado) es activar todos los objetos epoll que sondean en un fd.
EPOLLRDHUP	Socket de flujo de conexión cerrada por pares o apagado escribiendo la mitad de la conexión.
EPOLLRDNORM	Equivalente a **EPOLLIN**
EPOLLRDBAND	La banda de datos de prioridad se puede leer.
EPOLLWRNORM	Equivalente a **EPOLLOUT**
EPOLLWRBAND	Se puede escribir datos de prioridad.
EPOLLMSG	Ignorado.
EPOLLWAKEUP	Prevents sleep during event waiting.

Added in version 3.6: **EPOLLEXCLUSIVE** fue agregado. Solo es compatible con Linux Kernel 4.5 o posterior.

Added in version 3.14: **EPOLLWAKEUP** was added. It's only supported by Linux Kernel 3.5 or later.

epoll.close()

Cierra el descriptor del archivo de control del objeto epoll.

epoll.closed

`True` si el objeto epoll está cerrado.

epoll.fileno()

Retorna el número de descriptor de archivo del control fd.

epoll.fromfd(*fd*)

Crea un objeto epoll a partir de un descriptor de archivo dado.

epoll.register(*fd*[, *eventmask*])

Registra un descriptor fd con el objeto epoll.

epoll.modify(*fd*, *eventmask*)

Modifica un descriptor de archivo registrado.

epoll.unregister(*fd*)

Elimina un descriptor de archivo registrado del objeto epoll.

Distinto en la versión 3.9: El método ya no ignora el error **`EBADF`**.

epoll.poll(*timeout=None*, *maxevents=-1*)

Espera los eventos. tiempo de espera (*timeout*) en segundos (float)

Distinto en la versión 3.5: La función ahora se vuelve a intentar con un tiempo de espera (*timeout*) recalculado cuando se interrumpe por una señal, excepto si el controlador de señal genera una excepción (ver **PEP 475** [https://peps.python.org/pep-0475/] para la justificación), en lugar de generar **`InterruptedError`**.

Sondeo de objetos

The **`poll()`** system call, supported on most Unix systems, provides better scalability for network servers that service many, many clients at the same time. **`poll()`** scales better because the system call only requires listing the file descriptors of interest, while **`select()`** builds a bitmap, turns on bits for the fds of interest, and then afterward the whole bitmap has to be linearly scanned again. **`select()`** is *O*(*highest file descriptor*), while **`poll()`** is *O*(*number of file descriptors*).

poll.register(*fd*[, *eventmask*])

Registra un descriptor de archivo con el objeto de sondeo. Las futuras llamadas al método **`poll()`** comprobarán si el descriptor del archivo tiene algún evento I/O pendiente. *fd* puede ser un entero o un objeto con un método **`fileno()`** que retorna un entero. Los objetos de archivo implementan **`Fileno()`**, por lo que también pueden usarse como argumento.

eventmask es una máscara de bits opcional que describe el tipo de eventos que se desea verificar y puede ser una combinación de las constantes **`POLLIN`**, **`POLLPRI`**, y **`POLLOUT`**, descrito en la mesa de abajo. Si no se especifica, el valor predeterminado utilizado verificará los 3 tipos de eventos.

Constante	**Significado**
`POLLIN`	Hay datos para leer
`POLLPRI`	Hay datos urgentes para leer
`POLLOUT`	Lista para la salida: la escritura no bloqueará
`POLLERR`	Condición de error de algún tipo
`POLLHUP`	Colgado
`POLLRDHUP`	Socket de flujo de conexión cerrada por pares, o apagado escribiendo la mitad de la conexión
`POLLNVAL`	Solicitud no válida: descriptor no abierto

Al registrar un descriptor de archivo que ya está registrado no es un error y tiene el mismo efecto que registrar el descriptor exactamente una vez.

poll.modify(*fd*, *eventmask*)

Modifica un fd ya registrado. Esto tiene el mismo efecto que `register(fd, eventmask)`. Si se intenta modificar un descriptor de archivo que nunca se registró, se genera una excepción **`OSError`** con *errno* **`ENOENT`**.

poll.unregister(*fd*)

Elimina un descriptor de archivo que está siendo rastreado por un objeto de sondeo. Al igual que el método **`register()`**, *fd* puede ser un entero o un objeto con un método **`fileno()`** que retorna un entero.

Intenta eliminar un descriptor de archivo que nunca se registró provoca una excepción `KeyError`.

poll.poll([*timeout*])

Sondea el conjunto de descriptores de archivos registrados y retorna una lista posiblemente vacía que contiene tuplas de 2 elementos `(fd, event)` para los descriptores que tienen eventos o errores para informar. *fd* es el descriptor de archivo, y *event* es una máscara de bits con bits configurados para los eventos informados para ese descriptor — **`POLLIN`** para entrada en espera, **`POLLOUT`** para indicar que se puede escribir en el descriptor, y así sucesivamente. Una lista vacía indica que se agotó el tiempo de espera de la llamada y que ningún descriptor de archivo tuvo eventos para informar. Si se proporciona *timeout*, especifica el período de tiempo en milisegundos que el sistema esperará por los eventos antes de regresar. Si se omite *timeout*, es negativo o `None`, la llamada se bloqueará hasta que haya un evento para este objeto de sondeo.

Distinto en la versión 3.5: La función ahora se vuelve a intentar con un tiempo de espera (*timeout*) recalculado cuando se interrumpe por una señal, excepto si el controlador de señal genera una excepción (ver PEP 475 [https://peps.python.org/pep-0475/] para la justificación), en lugar de generar `InterruptedError`.

Objetos Kqueue

kqueue.close()

Cierra el descriptor del archivo de control del objeto kqueue.

kqueue.closed

`True` si el objeto *kqueue* está cerrado.

kqueue.fileno()

Retorna el número de descriptor de archivo del control fd.

kqueue.fromfd(*fd*)

Crea un objeto *kqueue* a partir de un descriptor de archivo dado.

kqueue.control(*changelist*, *max_events*[, *timeout*]) → eventlist

Interfaz de bajo nivel para kevent

- la lista de cambios (*changelist*) debe ser iterable de objetos kevent o `None`
- max_events debe ser 0 o un entero positivo
- tiempo de espera (*timeout*) en segundos (posible con *floats*); el valor predeterminado es `None`, para esperar para siempre

Distinto en la versión 3.5: La función ahora se vuelve a intentar con un tiempo de espera (*timeout*) recalculado cuando se interrumpe por una señal, excepto si el controlador de señal genera una excepción (ver **PEP 475** [https://peps.python.org/pep-0475/] para la justificación), en lugar de generar `InterruptedError`.

Objetos Kevent

https://man.freebsd.org/cgi/man.cgi?query=kqueue&sektion=2

kevent.ident

Valor utilizado para identificar el evento. La interpretación depende del filtro, pero generalmente es el descriptor de archivo. En el constructor, ident puede ser un int o un objeto con un método `fileno()`. kevent almacena el entero internamente.

kevent.filter

Nombre del filtro del kernel.

Constante	Significado
`KQ_FILTER_READ`	Toma un descriptor y retorna cada vez que hay datos disponibles para leer

KQ_FILTER_WRITE	Toma un descriptor y retorna cada vez que hay datos disponibles para escribir
KQ_FILTER_AIO	Solicitud de AIO
KQ_FILTER_VNODE	Retorna cuando ocurre uno o más de los eventos solicitados observados en *fflag*
KQ_FILTER_PROC	Vigila los eventos en un id de proceso
KQ_FILTER_NETDEV	Vigila los eventos en un dispositivo de red [no disponible en macOS]
KQ_FILTER_SIGNAL	Retorna cada vez que la señal observada se entrega al proceso
KQ_FILTER_TIMER	Establece un temporizador arbitrario

kevent.flags

Acción de filtro.

Constante	**Significado**
KQ_EV_ADD	Agrega o modifica un evento
KQ_EV_DELETE	Elimina un evento de la cola
KQ_EV_ENABLE	Permitscontrol() to return the event
KQ_EV_DISABLE	Disablesevent
KQ_EV_ONESHOT	Elimina evento después de la primera aparición
KQ_EV_CLEAR	Restablece el estado después de recuperar un evento
KQ_EV_SYSFLAGS	evento interno
KQ_EV_FLAG1	evento interno
KQ_EV_EOF	Filtrar la condición específica de EOF
KQ_EV_ERROR	Ver valores de retorno

kevent.fflags

Filtrar flags específicas.

KQ_FILTER_READ y **KQ_FILTER_WRITE** bandera de filtro:

Constante	**Significado**
KQ_NOTE_LOWAT	marca de agua baja de un *socket buffer*

KQ_FILTER_VNODE bandera de filtro:

Constante	**Significado**
KQ_NOTE_DELETE	*unlink()* fue llamado
KQ_NOTE_WRITE	una escritura ha ocurrido

Constante	Significado
KQ_NOTE_EXTEND	el archivo fue extendido
KQ_NOTE_ATTRIB	un atributo ha cambiado
KQ_NOTE_LINK	el recuento de enlaces ha cambiado
KQ_NOTE_RENAME	el archivo fue renombrado
KQ_NOTE_REVOKE	se revocó el acceso al archivo

KQ_FILTER_PROC bandera de filtro:

Constante	Significado
KQ_NOTE_EXIT	el proceso ha terminado (*exited*)
KQ_NOTE_FORK	el proceso ha llamado a *fork()*
KQ_NOTE_EXEC	el proceso ha ejecutado un nuevo proceso
KQ_NOTE_PCTRLMASK	flag de filtro interno
KQ_NOTE_PDATAMASK	flag de filtro interno
KQ_NOTE_TRACK	sigue un proceso a través de *fork()*
KQ_NOTE_CHILD	retornado en el proceso hijo para *NOTE_TRACK*
KQ_NOTE_TRACKERR	incapaz de adjuntar a un proceso hijo

Indicadores de filtro **KQ_FILTER_NETDEV** (no disponible en macOS):

Constante	Significado
KQ_NOTE_LINKUP	el enlace está funcionando
KQ_NOTE_LINKDOWN	el enlace está caído
KQ_NOTE_LINKINV	el estado del enlace es invalido

kevent.data

Filtrar datos específicos.

kevent.udata

Valor definido por el usuario.

`selectors` — High-level I/O multiplexing

Added in version 3.4.

Código fuente: Lib/selectors.py [https://github.com/python/cpython/tree/3.14/Lib/selectors.py]

Introducción

Este módulo permite multiplexación de E/S eficiente y de alto nivel, basada en los primitivos del módulo `select`. Se recomienda a los usuarios utilizar este módulo en su lugar, a menos que deseen un control preciso sobre los primitivos a nivel de sistema operativo utilizados.

It defines a `BaseSelector` abstract base class, along with several concrete implementations (`KqueueSelector`, `EpollSelector`…), that can be used to wait for I/O readiness notification on multiple file objects. In the following, «file object» refers to any object with a `fileno()` method, or a raw file descriptor. See file object.

`DefaultSelector` es un alias de la implementación más eficiente disponible en la plataforma actual: esta debería ser la opción predeterminada para la mayoría de los usuarios.

Nota

El tipo de objetos de archivo admitidos depende de la plataforma: en Windows, se admiten sockets, pero no *pipes*, mientras que en Unix, ambos son compatibles (también se pueden admitir algunos otros tipos, como FIFOs o dispositivos de archivo especiales).

Ver también

`select`
Módulo de multiplexación de E/S de bajo nivel.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

Clases

Jerarquía de clases:

```
BaseSelector
+-- SelectSelector
+-- PollSelector
+-- EpollSelector
+-- DevpollSelector
+-- KqueueSelector
```

In the following, *events* is a bitwise mask indicating which I/O events should be waited for on a given file object. It can be a combination of the module's constants below:

Constante	**Significado**
selectors.EVENT_READ	Disponible para lectura
selectors.EVENT_WRITE	Disponible para escritura

class selectors.SelectorKey

La clase **`SelectorKey`** es una **`namedtuple`** que se utiliza para asociar un objeto de archivo a su descriptor de archivo subyacente, máscara de evento seleccionada y datos adjuntos. Es retornada por varios métodos **`BaseSelector`**.

fileobj
Objeto de archivo registrado.

fd

Descriptor de archivo subyacente.

events

Eventos que se deben esperar en este objeto de archivo.

data

Datos opacos opcionales asociados a este objeto de archivo: por ejemplo, podría usarse para almacenar un ID de sesión por cliente.

class selectors.BaseSelector

Un **`BaseSelector`** se usa para esperar a que el evento de E/S esté listo en varios objetos de archivo. Admite el registro y la cancelación del registro de secuencias de archivos y un método para esperar eventos de E/S en esas secuencias, con un tiempo de espera opcional. Es una clase base abstracta, por lo que no se puede crear una instancia. Use **`DefaultSelector`** en su lugar, o un **`SelectSelector`**, **`KqueueSelector`**, etc. si deseas usar específicamente una implementación y su plataforma la admite. **`BaseSelector`** y sus implementaciones concretas soportan el protocolo context manager.

abstractmethod register(*fileobj*, *events*, *data=None*)

Registra un objeto de archivo para su selección y lo monitoriza en busca de eventos de E/S.

fileobj es el objeto de archivo a monitorear. Puede ser un descriptor de archivo de tipo entero o un objeto con un método `fileno()`. *events* es una máscara bit a bit de eventos para monitorear. *data* es un objeto opaco.

Esto retorna una nueva instancia **`SelectorKey`**, o lanza un **`ValueError`** en caso de una máscara de evento o un descriptor de archivo no válidos, o **`KeyError`** si el objeto de archivo ya está registrado.

abstractmethod unregister(*fileobj*)

Anula el registro de un objeto de archivo de la selección y lo elimina del monitoreo. Se anulará el registro de un objeto de archivo antes de cerrarlo.

fileobj debe ser un objeto de archivo previamente registrado.

Esto retorna la instancia **`SelectorKey`** asociada, o lanza un **`KeyError`** si *fileobj* no está registrado. Se lanzará un **`ValueError`** si *fileobj* no es válido (por ejemplo, no tiene el método `fileno ()` o su método `fileno ()` tiene un valor de retorno no válido).

modify(*fileobj*, *events*, *data=None*)

Cambia los eventos monitorizados de un objeto de archivo registrado o los datos adjuntos.

Esto es equivalente a *BaseSelector.unregister(fileobj)* seguido de *BaseSelector.register(fileobj, events, data)*, excepto que se puede implementar de manera más eficiente.

Esto retorna una nueva instancia de **`SelectorKey`**, o lanza un **`ValueError`** en caso de una máscara de evento o un descriptor de archivo no válidos, o **`KeyError`** si el objeto de archivo no está registrado.

abstractmethod select(*timeout=None*)

Espera hasta que algunos objetos de archivo registrados estén listos o el tiempo de espera expire.

Si `timeout > 0`, esto especifica el tiempo máximo de espera, en segundos. Si `timeout <= 0`, la llamada no se bloqueará y reportará los objetos de archivo actualmente listos. Si *timeout* es `None`, la llamada se bloqueará hasta que un objeto de archivo monitorizado esté listo.

Retorna una lista de tuplas `(key, events)`, una por cada objeto de archivo listo.

key es la instancia `SelectorKey` correspondiente a un objeto de archivo listo. *events* es una máscara de bits de eventos listos en este objeto de archivo.

Nota

Este método puede regresar antes de que cualquier objeto de archivo esté listo o haya transcurrido el tiempo de espera si el proceso actual recibe una señal: en este caso, se retornará una lista vacía.

Distinto en la versión 3.5: El selector ahora se reintenta con un tiempo de espera recalculado cuando es interrumpido por una señal si el gestor de señales no lanzó una excepción (ver **PEP 475** [https://peps.python.org/pep-0475/] para la justificación), en lugar de retornar una lista vacía de eventos antes del tiempo de espera.

close()

Cierra el selector.

Se debe llamar para asegurarse de que se libera cualquier recurso subyacente. El selector no se utilizará una vez cerrado.

get_key(*fileobj*)

Retorna la clave asociada con un objeto de archivo registrado.

Esto retorna la instancia `SelectorKey` asociada a este objeto de archivo, o lanza un `KeyError` si el objeto de archivo no está registrado.

abstractmethod get_map()

Retorna un mapeo asociando objetos de archivo a las llaves del selector.

Retorna una instancia de `Mapping` mapeando objetos de archivo registrados a su instancia `SelectorKey` asociada.

class selectors.DefaultSelector

La clase de selector predeterminada, utiliza la implementación más eficiente disponible en la plataforma actual. Esta debería ser la opción predeterminada para la mayoría de los usuarios.

class selectors.SelectSelector

Selector basado en `select.select()`.

class selectors.PollSelector

Selector basado en `select.poll()`.

class selectors.EpollSelector

Selector basado en `select.epoll()`.

fileno()

Esto retorna el descriptor de archivo utilizado por el objeto `select.epoll()` subyacente.

class selectors.DevpollSelector

Selector basado en `select.devpoll()`.

fileno()

Esto retorna el descriptor de archivo utilizado por el objeto `select.devpoll()` subyacente.

Added in version 3.5.

class selectors.KqueueSelector

Selector basado en `select.kqueue()`.

fileno()

Esto retorna el descriptor de archivo utilizado por el objeto `select.kqueue()` subyacente.

Ejemplos

Aquí hay una implementación simple de un servidor de eco:

```
import selectors
import socket

sel = selectors.DefaultSelector()

def accept(sock, mask):
    conn, addr = sock.accept()  # Should be ready
    print('accepted', conn, 'from', addr)
    conn.setblocking(False)
    sel.register(conn, selectors.EVENT_READ, read)

def read(conn, mask):
    data = conn.recv(1000)  # Should be ready
    if data:
        print('echoing', repr(data), 'to', conn)
        conn.send(data)  # Hope it won't block
    else:
        print('closing', conn)
        sel.unregister(conn)
        conn.close()

sock = socket.socket()
sock.bind(('localhost', 1234))
sock.listen(100)
sock.setblocking(False)
sel.register(sock, selectors.EVENT_READ, accept)

while True:
    events = sel.select()
    for key, mask in events:
        callback = key.data
        callback(key.fileobj, mask)
```

`signal` — Set handlers for asynchronous events

Código fuente: Lib/signal.py [https://github.com/python/cpython/tree/3.14/Lib/signal.py]

Este módulo proporciona mecanismos para usar gestores de señales en Python.

Reglas generales

La función `signal.signal()` permite definir gestores personalizados que serán ejecutados cuando una señal es recibida. Un pequeño número de gestores por defecto son instalados: `SIGPIPE` es ignorada (por lo que los errores de escritura en tuberías y sockets se pueden informar como excepciones ordinarias de Python) y `SIGINT` es trasladada en una excepción `KeyboardInterrupt` si el proceso padre no lo ha cambiado.

El gestor para una señal en particular, una vez establecido, continua instalado hasta que se resetea explícitamente (Python emula el estilo de interfaz BSD independientemente de la implementación subyacente), con la excepción del gestor para `SIGCHLD`, que sigue la implementación subyacente.

On WebAssembly platforms, signals are emulated and therefore behave differently. Several functions and signals are not available on these platforms.

Ejecución de los gestores de señales de Python

A Python signal handler does not get executed inside the low-level (C) signal handler. Instead, the low-level signal handler sets a flag which tells the virtual machine to execute the corresponding Python signal handler at a later point (for example, at the next bytecode instruction). This has consequences:

- Tiene poco sentido detectar errores sincrónicos como `SIGFPE` o `SIGSEGV` que son causados por una operación no válida en código C. Python retornará desde el gestor de señales a código C, que es probable que extienda la misma señal otra vez, ocasionando que Python se cuelgue aparentemente. Desde Python 3.3 en adelante, puedes usar el módulo `faulthandler` para reportar errores síncronos.
- Un cálculo de larga duración implementado exclusivamente en C (como una coincidencia de expresiones regulares en un gran cuerpo de texto) puede funcionar interrumpidamente durante una cantidad arbitraria de tiempo, independientemente de las señales recibidas. Los gestores de señales de Python serán llamados cuando el cálculo finalice.
- Si el gestor genera alguna excepción, será lanzada inesperadamente. Vea la nota abajo para discusión.

Señales e hilos

Los gestores de señales de Python se ejecutan siempre en el hilo principal de Python, incluso si la señal fue recibida desde otro hilo. Esto significa que las señales no pueden ser usadas como un medio de comunicación entre hilos. Puedes usar las primitivas de sincronización desde el módulo `threading` en su lugar.

Además, solo el hilo principal puede configurar un nuevo gestor de señal.

Advertencia

Synchronization primitives such as `threading.Lock` should not be used within signal handlers. Doing so can lead to unexpected deadlocks.

Contenidos del módulo

Distinto en la versión 3.5: Las constantes relacionadas con señal

(SIG*), gestor (`SIG_DFL`, `SIG_IGN`) y sigmask (`SIG_BLOCK`, `SIG_UNBLOCK`, `SIG_SETMASK`) enumeradas a continuación fueron convertidas en `enums` (`Signals`, `Handlers` y `Sigmasks`, respectivamente). Las funciones `getsignal()`, `pthread_sigmask()`, `sigpending()` y `sigwait()` devuelven **`enums`** legibles por humanos como objetos **`Signals`**.

El módulo signal define tres enums:

class signal.Signals

`enum.IntEnum` colección de constantes SIG* y colección de constantes CTRL_*.

Added in version 3.5.

class signal.Handlers

`enum.IntEnum` collection of the constants `SIG_DFL` and `SIG_IGN`.

Added in version 3.5.

class signal.Sigmasks

`enum.IntEnum` collection of the constants `SIG_BLOCK`, `SIG_UNBLOCK` and `SIG_SETMASK`.

Availability: Unix.

Consulta la página del manual *sigprocmask(2)* *[https://manpages.debian.org/sigprocmask(2)]* y *pthread_sigmask(3)* *[https://manpages.debian.org/pthread_sigmask(3)]* para obtener más información.

Added in version 3.5.

The variables defined in the **`signal`** module are:

signal.SIG_DFL

Ésta es una de las dos opciones estándar de manejo de señales; simplemente realizará la función predeterminada para la señal. Por ejemplo, en la mayoría de los sistemas, la acción predeterminada para **`SIGQUIT`** es volcar el núcleo y salir, mientras que la acción predeterminada para **`SIGCHLD`** es simplemente ignorarlo.

signal.SIG_IGN

Este es otro manejador de señales estándar, que simplemente ignorará la señal dada.

signal.SIGABRT

Abortar señal de *abort(3)* *[https://manpages.debian.org/abort(3)]*.

signal.SIGALRM

Señal de temporizador de *alarm(2)* *[https://manpages.debian.org/alarm(2)]*.

Availability: Unix.

signal.SIGBREAK

Interrumpir desde el teclado (CTRL + BREAK).

Availability: Windows.

signal.SIGBUS

Error de bus (mal acceso a la memoria).

Availability: Unix.

signal.SIGCHLD

El proceso hijo se detuvo o terminó.

Availability: Unix.

signal.SIGCLD

Alias para **`SIGCHLD`**.

Availability: not macOS.

signal.SIGCONT

Continuar el proceso si está detenido actualmente

Availability: Unix.

signal.SIGFPE

Excepción de coma flotante. Por ejemplo, división por cero.

Ver también

`ZeroDivisionError` se genera cuando el segundo argumento de una operación de división o módulo es cero.

signal.SIGHUP

Se detectó un bloqueo en el terminal de control o muerte del proceso de control.

Availability: Unix.

signal.SIGILL

Instrucción ilegal.

signal.SIGINT

Interrumpir desde el teclado (CTRL + C).

La acción predeterminada es generar `KeyboardInterrupt`.

signal.SIGKILL

Señal de muerte.

No se puede detectar, bloquear ni ignorar.

Availability: Unix.

signal.SIGPIPE

Tubería rota: escriba en la tubería sin lectores.

La acción predeterminada es ignorar la señal.

Availability: Unix.

signal.SIGPROF

Profiling timer expired.

Availability: Unix.

signal.SIGQUIT

Terminal quit signal.

Availability: Unix.

signal.SIGSEGV

Fallo de segmentación: referencia de memoria no válida.

signal.SIGSTOP

Stop executing (cannot be caught or ignored).

Availability: Unix.

signal.SIGSTKFLT

Fallo en el coprocesador del stack. El kernel de Linux no lanza esta señal,esta sólo puede ser lanzada en el espacio de usuario.

Availability: Linux.

En arquitecturas donde la señal está disponible. Consulta la página del manual *signal(7)* *[https://manpages.debian.org/signal(7)]* para obtener más información.

Added in version 3.11.

signal.SIGTERM

Señal de terminación.

signal.SIGUSR1

Señal definida por el usuario 1.

Availability: Unix.

signal.SIGUSR2

Señal definida por el usuario 2.

Availability: Unix.

signal.SIGVTALRM

Virtual timer expired.

Availability: Unix.

signal.SIGWINCH

Señal de cambio de tamaño de ventana.

Availability: Unix.

signal.SIGXCPU

CPU time limit exceeded.

Availability: Unix.

SIG*

All the signal numbers are defined symbolically. For example, the hangup signal is defined as **`signal.SIGHUP`**; the variable names are identical to the names used in C programs, as found in `<signal.h>`. The Unix man page for "`signal`" lists the existing signals (on some systems this is *signal(2)* *[https://manpages.debian.org/signal(2)]*, on others the list is in *signal(7)* *[https://manpages.debian.org/signal(7)]*). Note that not all systems define the same set of signal names; only those names defined by the system are defined by this module.

signal.CTRL_C_EVENT

La señal correspondiente al evento de pulsación de tecla `Ctrl + C`. Esta señal solo se puede utilizar con **`os.kill()`**.

Availability: Windows.

Added in version 3.2.

signal.CTRL_BREAK_EVENT

La señal correspondiente al evento de pulsación de tecla `Ctrl + Break`. Esta señal solo se puede utilizar con **`os.kill()`**.

Availability: Windows.

Added in version 3.2.

signal.NSIG

Un número más alto que el número más alto de señal. Use **`valid_signals()`** para obtener números válidos de señales.

signal.ITIMER_REAL

Reduce el temporizador de intervalo en tiempo real y entrega **`SIGALRM`** al vencimiento.

signal.ITIMER_VIRTUAL

Disminuye el temporizador de intervalo solo cuando el proceso se está ejecutando y entrega SIGVTALRM al vencimiento.

signal.ITIMER_PROF

Disminuye el temporizador de intervalo tanto cuando se ejecuta el proceso como cuando el sistema se está ejecutando en nombre del proceso. Junto con ITIMER_VIRTUAL, este temporizador generalmente se usa para perfilar el tiempo que pasa la aplicación en el espacio del usuario y del kernel. SIGPROF se entrega al vencimiento.

signal.SIG_BLOCK

Un valor posible para el parámetro *how* para **`pthread_sigmask()`** que indica que las señales deben bloquearse.

Added in version 3.3.

signal.SIG_UNBLOCK

Un valor posible para el parámetro *how* para `pthread_sigmask()` que indica que las señales deben desbloquearse.

Added in version 3.3.

signal.SIG_SETMASK

Un valor posible para el parámetro *how* para `pthread_sigmask()` que indica que la máscara de señal debe ser reemplazada.

Added in version 3.3.

The `signal` module defines one exception:

exception signal.ItimerError

Se genera para señalar un error de la implementación subyacente `setitimer()` o `getitimer()`. Espere este error si se pasa un temporizador de intervalo no válido o un tiempo negativo a `setitimer()`. Este error es un subtipo de `OSError`.

Added in version 3.3: Este error solía ser un subtipo de `IOError`, que ahora es un alias de `OSError`.

The `signal` module defines the following functions:

signal.alarm(*time*)

Si *time* es distinto de cero, esta función solicita que se envíe una señal `SIGALRM` al proceso en *time* segundos. Se cancela cualquier alarma programada previamente (solo se puede programar una alarma en cualquier momento). El valor retornado es entonces el número de segundos antes de que se entregara cualquier alarma previamente configurada. Si *time* es cero, no se programa ninguna alarma y se cancela cualquier alarma programada. Si el valor de retorno es cero, no hay ninguna alarma programada actualmente.

Availability: Unix.

Consulta la página del manual *alarm(2)* *[https://manpages.debian.org/alarm(2)]* para obtener más información.

signal.getsignal(*signalnum*)

Retorna el manejador de señales actual para la señal *signalnum*. El valor retornado puede ser un objeto de Python invocable o uno de los valores especiales `signal.SIG_IGN`, `signal.SIG_DFL` o `None`. Aquí, **`signal.SIG_IGN`** significa que la señal fue previamente ignorada, **`signal.SIG_DFL`** significa que la forma predeterminada de manejar la señal estaba en uso anteriormente, y el gestor de señales no se instaló desde Python.

signal.strsignal(*signalnum*)

Retorna la descripción de la señal *signalnum*, como «Interrupt» para `SIGINT`. Retorna `None` si *signalnum* no tiene descripción. Lanza `ValueError` si *signalnum* no es válido.

Added in version 3.8.

signal.valid_signals()

Retorna el conjunto de números de señal válidos en esta plataforma. Esto puede ser menor que `rango(1, NSIG)` si el sistema reserva algunas señales para uso interno.

Added in version 3.8.

signal.pause()

Hacer que el proceso duerma hasta que se reciba una señal; entonces se llamará al manejador apropiado. No retorna nada.

Availability: Unix.

Consulta la página del manual *signal(2)* *[https://manpages.debian.org/signal(2)]* para obtener más información.

Vea también `sigwait()`, `sigwaitinfo()`, `sigtimedwait()` y `sigpending()`.

signal.raise_signal(*signum*)

Envía una señal al proceso de llamada. No retorna nada.

Added in version 3.8.

signal.pidfd_send_signal(*pidfd*, *sig*, *siginfo=None*, *flags=0*)

Envía la señal *sig* al proceso referido por el descriptor de archivo *pidfd*. Python actualmente no soporta el parámetro *siginfo*; éste debe ser `None`. El argumento *flags* está proveído para extensiones futuras; no hay valores actualmente definidos para *flags*.

Para más información vea la página de manual *pidfd_send_signal(2)* *[https://manpages.debian.org/pidfd_send_signal(2)]*.

Availability: Linux >= 5.1, Android >= `build-time` API level 31

Added in version 3.9.

signal.pthread_kill(*thread_id*, *signalnum*)

Envíe la señal *signalnum* al hilo *thread_id*, otro hilo en el mismo proceso que el llamador. El hilo de destino puede ejecutar cualquier código (Python o no). Sin embargo, si el hilo de destino está ejecutando el intérprete de Python, los manejadores de señales de Python serán ejecutados por el hilo principal. Por lo tanto, el único punto de enviar una señal a un hilo de Python en particular sería forzar que una llamada al sistema en ejecución falle con `InterruptedError`.

Use `threading.get_ident()` or the `ident` attribute of `threading.Thread` objects to get a suitable value for *thread_id*.

Si *signalnum* es 0, no se envía ninguna señal, pero se sigue realizando la comprobación de errores; esto se puede usar para verificar si el hilo de destino aún se está ejecutando.

Genera un evento de auditoría `signal.pthread_kill` con argumentos `thread_id`, `signalnum`.

Availability: Unix.

Consulta la página del manual *pthread_kill(3)* *[https://manpages.debian.org/pthread_kill(3)]* para obtener más información.

Vea también **`os.kill()`**.

Added in version 3.3.

signal.pthread_sigmask(*how*, *mask*)

Busca o cambia la máscara de señal del hilo de llamada. La máscara de señal es el conjunto de señales cuya entrega está actualmente bloqueada para la persona que llama. Retorna la máscara de señal anterior como un conjunto de señales.

El comportamiento de la llamada depende del valor de *how*, como sigue.

- **`SIG_BLOCK`**: El conjunto de señales bloqueadas es la unión del conjunto actual y el argumento *mask*.
- **`SIG_UNBLOCK`**: Las señales en *mask* se eliminan del conjunto actual de señales bloqueadas. Está permitido intentar desbloquear una señal que no esté bloqueada.
- **`SIG_SETMASK`**: El conjunto de señales bloqueadas se establece en el argumento *mask*.

mask es un conjunto de números de señales (por ejemplo, {**`signal.SIGINT`**, **`signal.SIGTERM`**}). Utilice **`valid_signals()`** para una máscara completa que incluya todas las señales.

Por ejemplo, `signal.pthread_sigmask(signal.SIG_BLOCK, [])` lee la máscara de señal del hilo de llamada.

`SIGKILL` y `SIGSTOP` no se pueden bloquear.

Availability: Unix.

Consulta la página del manual *sigprocmask(2) [https://manpages.debian.org/sigprocmask(2)]* y *pthread_sigmask(3) [https://manpages.debian.org/pthread_sigmask(3)]* para obtener más información.

Vea también `pause()`, `sigpending()` y `sigwait()`.

Added in version 3.3.

signal.setitimer(*which*, *seconds*, *interval=0.0*)

Establece el temporizador de intervalo dado (uno de `signal.ITIMER_REAL`, `signal.ITIMER_VIRTUAL` o `signal.ITIMER_PROF`) especificado por *which* disparar después de *seconds* (se acepta el números de punto flotante, diferente de `alarm()`) y luego cada *interval* segundos (si *interval* es distinto de cero). El temporizador de intervalo especificado por *which* se puede borrar estableciendo *seconds* en cero.

Cuando se dispara un temporizador de intervalos, se envía una señal al proceso. La señal enviada depende del temporizador que se utilice; `signal.ITIMER_REAL` entregará `SIGALRM`, `signal.ITIMER_VIRTUAL` envía `SIGVTALRM`, y `signal.ITIMER_PROF` entregará `SIGPROF`.

Los valores antiguos se retornan como una tupla: (retraso, intervalo).

Si intenta pasar un temporizador de intervalo no válido, se producirá un `ItimerError`.

Availability: Unix.

signal.getitimer(*which*)

Retorna el valor actual de un temporizador de intervalo dado especificado por *which*.

Availability: Unix.

signal.set_wakeup_fd(*fd*, *, *warn_on_full_buffer=True*)

Set the wakeup file descriptor to *fd*. When a signal your program has registered a signal handler for is received, the signal number is written as a single byte into the fd. If you haven't registered a signal handler for the signals you care about, then nothing will be written to the wakeup fd. This can be used by a library to wakeup a poll or select call, allowing the signal to be fully processed.

Se retorna el antiguo fd de activación (o -1 si la activación del descriptor de archivo no estaba habilitada). Si *fd* es -1, la activación del descriptor de archivo está deshabilitada. Si no es -1, *fd* debe ser sin bloqueo. Depende de la biblioteca eliminar los bytes de *fd* antes de llamar a poll o seleccionar nuevamente.

Cuando los hilos están habilitados, esta función solo se puede llamar desde el subproceso principal del intérprete principal; intentar llamarlo desde otros hilos hará que se lance una excepción **`ValueError`**.

Hay dos formas habituales de utilizar esta función. En ambos enfoques, usa el fd para despertarse cuando llega una señal, pero luego difieren en cómo determinan *which* señal o señales han llegado.

En el primer enfoque, leemos los datos del búfer de fd, y los valores de bytes le dan los números de señal. Esto es simple, pero en casos raros puede surgir un problema: generalmente el fd tendrá una cantidad limitada de espacio en el búfer, y si llegan demasiadas señales demasiado rápido, entonces el búfer puede llenarse y algunas señales pueden perderse. Si usa este enfoque, entonces debe configurar `warn_on_full_buffer = True`, que al menos causará que se imprima una advertencia en stderr cuando se pierdan las señales.

En el segundo enfoque, usamos el wakeup fd *solo* para wakeups e ignoramos los valores de bytes reales. En este caso, lo único que nos importa es si el búfer de fd está vacío o no; un búfer lleno no indica ningún problema. Si utiliza este enfoque, debe configurar `warn_on_full_buffer = False`, para que sus usuarios no se confundan con mensajes de advertencia falsos.

Distinto en la versión 3.5: En Windows, la función ahora también admite identificadores de socket.

Distinto en la versión 3.7: Se agregó el parámetro `warn_on_full_buffer`.

signal.siginterrupt(*signalnum*, *flag*)

Cambiar el comportamiento de reinicio de la llamada al sistema: si *flag* es **`False`**, las llamadas al sistema se reiniciarán cuando las interrumpa la señal *signalnum*, de lo contrario, las llamadas al sistema se interrumpirán. No retorna nada.

Availability: Unix.

Consulta la página del manual *siginterrupt(3)* *[https://manpages.debian.org/siginterrupt(3)]* para obtener más información.

Tenga en cuenta que la instalación de un gestor de señales con **`signal()`** restablecerá el comportamiento de reinicio a interrumpible llamando implícitamente a **`siginterrupt()`** con un valor de *flag* verdadero para la señal dada.

signal.signal(*signalnum*, *handler*)

Establece el gestor de la señal *signalnum* en la función *handler* (manejador). *handler* puede ser un objeto de Python invocable que toma dos argumentos (ver más abajo), o uno de los valores especiales **`signal.SIG_IGN`** o **`signal.SIG_DFL`**. Se retornará el manejador de señales anterior (vea la descripción de **`getsignal()`** arriba). (Consulte la página del manual de Unix *signal(2)* *[https://manpages.debian.org/signal(2)]* para obtener más información).

Cuando los hilos están habilitados, esta función solo se puede llamar desde el subproceso principal del intérprete principal; intentar llamarlo desde otros hilos hará que se lance una excepción **`ValueError`**.

El *handler* se llama con dos argumentos: el número de señal y el marco de pila actual (`None` o un objeto marco; para obtener una descripción de los objetos marco, consulte la descripción en la jerarquía de tipos o vea las descripciones de los atributos en el módulo **`inspect`**).

En Windows, **`signal()`** solo se puede llamar con **`SIGABRT`**, **`SIGFPE`**, **`SIGILL`**, **`SIGINT`**, **`SIGSEGV`**, **`SIGTERM`**, o **`SIGBREAK`**. Un **`ValueError`** se lanzará en cualquier otro caso. Tenga en cuenta que no todos los sistemas definen el mismo conjunto de nombres de señales; un **`AttributeError`** se lanzará si un nombre de señal no está definido como constante de nivel de módulo `SIG*`.

signal.sigpending()

Examine el conjunto de señales que están pendientes de entrega al hilo de llamada (es decir, las señales que se han generado mientras estaban bloqueadas). Retorna el conjunto de señales pendientes.

Availability: Unix.

Consulta la página del manual *sigpending(2)* *[https://manpages.debian.org/sigpending(2)]* para obtener más información.

Vea también **`pause()`**, **`pthread_sigmask()`** y **`sigwait()`**.

Added in version 3.3.

signal.sigwait(*sigset*)

Suspende la ejecución del hilo de llamada hasta la entrega de una de las señales especificadas en el conjunto de señales *sigset*. La función acepta la señal (la elimina de la lista pendiente de señales) y retorna el número de señal.

Availability: Unix.

Consulta la página del manual *sigwait(3)* *[https://manpages.debian.org/sigwait(3)]* para obtener más información.

Vea también **`pause()`**, **`pthread_sigmask()`**, **`sigpending()`**, **`sigwaitinfo()`** y **`sigtimedwait()`**.

Added in version 3.3.

signal.sigwaitinfo(*sigset*)

Suspende la ejecución del hilo de llamada hasta la entrega de una de las señales especificadas en el conjunto de señales *sigset*. La función acepta la señal y la elimina de la lista de señales pendientes. Si una de las señales en *sigset* ya está pendiente para el hilo de llamada, la función regresará inmediatamente con información sobre esa señal. No se llama al gestor de señales para la señal enviada. La función genera un **`InterruptedError`** si es interrumpida por una señal que no está en *sigset*.

The return value is an object representing the data contained in the `siginfo_t` structure, namely: `si_signo`, `si_code`, `si_errno`, `si_pid`, `si_uid`, `si_status`, `si_band`.

Availability: Unix.

Consulta la página del manual *sigwaitinfo(2)* *[https://manpages.debian.org/sigwaitinfo(2)]* para obtener más información.

Vea también **`pause()`**, **`sigwait()`** y **`sigtimedwait()`**.

Added in version 3.3.

Distinto en la versión 3.5: La función ahora se vuelve a intentar si es interrumpida por una señal que no está en *sigset* y el manejador de señales no genera una excepción (ver **PEP 475** [https://peps.python.org/pep-0475/] para la justificación).

signal.sigtimedwait(*sigset*, *timeout*)

Similar a `sigwaitinfo()`, pero toma un argumento *timeout* adicional que especifica un tiempo de espera. Si *timeout* se especifica como `0`, se realiza una encuesta. Retorna `None` si se agota el tiempo de espera.

Availability: Unix.

Consulta la página del manual *sigtimedwait(2)* *[https://manpages.debian.org/sigtimedwait(2)]* para obtener más información.

Vea también `pause()`, `sigwait()` y `sigwaitinfo()`.

Added in version 3.3.

Distinto en la versión 3.5: La función ahora se reintenta con el *timeout* recalculado si se interrumpe por una señal que no está en *sigset* y el manejador de señales no genera una excepción (ver **PEP 475** [https://peps.python.org/pep-0475/] para la justificación).

Ejemplos

Aquí hay un programa de ejemplo mínimo. Utiliza la función `alarm()` para limitar el tiempo de espera para abrir un archivo; esto es útil si el archivo es para un dispositivo serial que puede no estar encendido, lo que normalmente haría que `os.open()` se cuelgue indefinidamente. La solución es configurar una alarma de 5 segundos antes de abrir el archivo; si la operación lleva demasiado tiempo, se enviará la señal de alarma y el gestor genera una excepción.

```
import signal, os

def handler(signum, frame):
    signame = signal.Signals(signum).name
    print(f'Signal handler called with signal {signame}
    raise OSError("Couldn't open device!")

# Set the signal handler and a 5-second alarm
signal.signal(signal.SIGALRM, handler)
signal.alarm(5)

# This open() may hang indefinitely
fd = os.open('/dev/ttyS0', os.O_RDWR)

signal.alarm(0)          # Disable the alarm
```

Nota sobre SIGPIPE

Canalizar la salida de su programa a herramientas como *head(1)* *[https://manpages.debian.org/head(1)]* hará que se envíe una señal **`SIGPIPE`** a su proceso cuando el receptor de su salida estándar se cierre antes. Esto da como resultado una excepción como `BrokenPipeError: [Errno 32] Broken pipe`. Para manejar este caso, envuelva su punto de entrada para detectar esta excepción de la siguiente manera:

```
import os
import sys

def main():
    try:
        # simulate large output (your code replaces this
        for x in range(10000):
            print("y")
        # flush output here to force SIGPIPE to be trigg
        # while inside this try block.
        sys.stdout.flush()
    except BrokenPipeError:
        # Python flushes standard streams on exit; redir
        # to devnull to avoid another BrokenPipeError at
        devnull = os.open(os.devnull, os.O_WRONLY)
        os.dup2(devnull, sys.stdout.fileno())
        sys.exit(1)  # Python exits with error code 1 or

if __name__ == '__main__':
    main()
```

No establezcas la disposición de **`SIGPIPE`** en **`SIG_DFL`** para evitar

`BrokenPipeError`. Hacerlo causaría que tu programa se cierre inesperadamente siempre que se interrumpa cualquier conexión de socket mientras su programa aún esté escribiendo en él.

Nota sobre Manejadores de Señales y Excepciones

Si un gestor de señal lanza una excepción, ésta excepción será propagada al hilo principal de ejecución y podría ser lanzado luego de cualquier instrucción de bytecode. Más notablemente, **`KeyboardInterrupt`** podría aparecer en cualquier momento de la ejecución. La mayoría del código python, incluida la librería estándar, no puede robustecerse en contra esto, y por esto **`KeyboardInterrupt`** (o cualquier otra excepción resultante de un gestor de señal) podría poner el programa en un estado inesperado en raras ocasiones.

Para ilustrar esto, considere el siguiente código:

```
class SpamContext:
    def __init__(self):
        self.lock = threading.Lock()

    def __enter__(self):
        # If KeyboardInterrupt occurs here, everything i
        self.lock.acquire()
        # If KeyboardInterrupt occurs here, __exit__ wil
        ...
        # KeyboardInterrupt could occur just before the

    def __exit__(self, exc_type, exc_val, exc_tb):
        ...
        self.lock.release()
```

Para muchos programas, especialmente para los que sólo quieren interrumpir la ejecución ante un **`KeyboardInterrupt`**, esto no es un problema, pero las aplicaciones que son complejas o requieren alta confiabilidad deben evitar lanzar excepciones desde los gestores de señales. También deben evitar atrapar

`KeyboardInterrupt` como medio de una terminación de ejecución airosa. En su lugar, deben instalar su propio gestor **`SIGINT`**. A continuación, un ejemplo de un servidor HTTP que evita el **`KeyboardInterrupt`**:

```
import signal
import socket
from selectors import DefaultSelector, EVENT_READ
from http.server import HTTPServer, SimpleHTTPRequestHar

interrupt_read, interrupt_write = socket.socketpair()

def handler(signum, frame):
    print('Signal handler called with signal', signum)
    interrupt_write.send(b'\0')
signal.signal(signal.SIGINT, handler)

def serve_forever(httpd):
    sel = DefaultSelector()
    sel.register(interrupt_read, EVENT_READ)
    sel.register(httpd, EVENT_READ)

    while True:
        for key, _ in sel.select():
            if key.fileobj == interrupt_read:
                interrupt_read.recv(1)
                return
            if key.fileobj == httpd:
                httpd.handle_request()

print("Serving on port 8000")
httpd = HTTPServer(('', 8000), SimpleHTTPRequestHandler)
serve_forever(httpd)
print("Shutdown...")
```

`mmap` — Memory-mapped file support

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

Los objetos de archivos mapeados en memoria se comportan como objetos de tipo `bytearray` y objetos archivo. Puedes usar objetos mmap en la mayoría de lugares donde se esperen objetos de tipo `bytearray`; por ejemplo, puedes usar el módulo `re` para buscar entre un archivo mapeado en memoria. También puedes cambiar un solo byte al hacer `obj[index]==97`, o cambiar una subsecuencia al asignarle una rebanada: `obj[i1:i2] = b'...'`. También puedes leer y escribir datos que empiezan en la posición del archivo actual, y utilizar el método `seek()` a través del archivo para cambiar la posición actual.

A memory-mapped file is created by the `mmap` constructor, which is different on Unix and on Windows. In either case you must provide a file descriptor for a file opened for update. If you wish to map an existing Python file object, use its `fileno()` method to obtain the correct value for the *fileno* parameter. Otherwise, you can open the file using the `os.open()` function, which returns a file descriptor directly (the file still needs to be closed when done).

Nota

Si quieres crear un mapeado en memoria para un archivo con permisos de escritura y en el búfer, debes ejecutar la función `flush()`. Es necesario para asegurar que las modificaciones locales a los búfer estén realmente disponible para el mapeado.

Para las versiones del constructor de tanto Unix como de Windows, *access* puede ser especificado como un parámetro nombrado opcional. *access* acepta uno de cuatro valores: **`ACCESS_READ`**, **`ACCESS_WRITE`**, o **`ACCESS_COPY`** para especificar una memoria de solo lectura, *write- through*, o *copy-on-write* respectivamente, o **`ACCES_DEFAULT`** para deferir a *prot*. El parámetro *access* se puede usar tanto en Unix como en Windows. Si *access* no es especificado, el mmap de Windows retorna un mapeado *write-through*. Los valores de la memoria inicial para los tres tipos de acceso son tomados del archivo especificado. La asignación a una mapa de memoria **`ACCESS_READ`** lanza una excepción **`TypeError`**. La asignación a un mapa de memoria **`ACCESS_WRITE`** afecta tanto a la memoria como al archivo subyacente. La asignación a un mapa de memoria **`ACCES_COPY`** afecta a la memoria pero no actualiza el archivo subyacente.

Distinto en la versión 3.7: Se añadió la constante **`ACCESS_DEFAULT`**.

Para mapear memoria anónima, se debe pasar -1 como el *fileno* junto con la longitud.

class mmap.mmap(*fileno*, *length*, *tagname=None*, *access=ACCESS_DEFAULT*, *offset=0*)

(En la versión de Windows) Mapea *length* bytes desde el archivo especificado por el gestor de archivo *fileno*, y crea un objeto mmap. Si *length* es más largo que el tamaño actual del archivo, el archivo es extendido para contener *length* bytes. Si *length* es `0`, la longitud máxima del map es la tamaño actual del archivo, salvo que si el archivo está vacío Windows lanza una excepción (no puedes crear un mapeado vacío en Windows)

tagname, if specified and not `None`, is a string giving a tag name for the mapping. Windows allows you to have many different mappings against the same file. If you specify the name of an existing tag, that tag is opened, otherwise a new tag of this name is created. If this parameter is omitted or `None`, the mapping is created without a name. Avoiding the use of the *tagname* parameter will assist in keeping your code portable between Unix and Windows.

offset puede ser especificado como un *offset* entero no negativo. las referencias de mmap serán relativas al *offset* desde el comienzo del archivo. *offset* es por defecto 0. *offset* debe ser un múltiplo de **`ALLOCATIONGRANULARITY`**.

Lanza un evento de inspección `mmap.__new__` con los argumentos `fileno`, `length`, `access`, `offset`.

class mmap.mmap(*fileno*, *length*, *flags=MAP_SHARED*, *prot=PROT_WRITE | PROT_READ*, *access=ACCESS_DEFAULT*, *offset=0*, ***, *trackfd=True*)

(En la versión de Unix) Mapea *length* bytes desde el archivo especificado por el descriptor de archivo *fileno*, y retorna un objeto mmap. Si *length* es `0`, la longitud máxima del map será el tamaño actual del archivo cuando **`mmap`** sea llamado.

flags especifica la naturaleza del mapeado. **`MAP_PRIVATE`** crea un mapeado *copy-on-write* privado, por lo que los cambios al contenido del objeto mmap serán privados para este proceso, y **`MAP_SHARED`** crea un mapeado que es compartido con todos los demás procesos que mapean las mismas áreas del archivo. El valor por defecto es **`MAP_SHARED`**. Algunos tipos tienen posibles banderas adicionales con la lista completa en MAP_* constants.

prot, si se especifica, proporciona la protección de memoria deseado; los dos valores más útiles son **`PROT_READ`** y **`PROT_WRITE`**, para especificar que las páginas puedan ser escritas o leídas. *prot* es por defecto **`PROT_READ`** | **`PROT_WRITE`**.

access puede ser especificado en lugar de *flags* y *prot* como un parámetro nombrado opcional. Es un error especificar tanto *flags*, *prot* como *access*. Véase la descripción de *access* arriba por información de cómo usar este parámetro.

offset puede ser especificado como un *offset* entero no negativo. Las referencias serán relativas al *offset* desde el comienzo del archivo. *offset* por defecto es 0. *offset* debe ser un múltiplo de **`ALLOCATIONGRANULARITY`** que es igual a

`PAGESIZE` en los sistemas Unix.

If *trackfd* is `False`, the file descriptor specified by *fileno* will not be duplicated, and the resulting **`mmap`** object will not be associated with the map's underlying file. This means that the **`size()`** and **`resize()`** methods will fail. This mode is useful to limit the number of open file descriptors.

To ensure validity of the created memory mapping the file specified by the descriptor *fileno* is internally automatically synchronized with the physical backing store on macOS.

Distinto en la versión 3.13: The *trackfd* parameter was added.

Este ejemplo muestra un forma simple de usar **`mmap`**:

```
import mmap

# write a simple example file
with open("hello.txt", "wb") as f:
    f.write(b"Hello Python!\n")

with open("hello.txt", "r+b") as f:
    # memory-map the file, size 0 means whole file
    mm = mmap.mmap(f.fileno(), 0)
    # read content via standard file methods
    print(mm.readline())  # prints b"Hello Python!\
    # read content via slice notation
    print(mm[:5])  # prints b"Hello"
    # update content using slice notation;
    # note that new content must have same size
    mm[6:] = b" world!\n"
    # ... and read again using standard file method
    mm.seek(0)
    print(mm.readline())  # prints b"Hello  world!\
    # close the map
    mm.close()
```

`mmap` también puede ser usado como un gestor de contexto en una sentencia **`with`**

```
import mmap

with mmap.mmap(-1, 13) as mm:
    mm.write(b"Hello world!")
```

Added in version 3.2: Soporte del gestor de contexto.

El siguiente ejemplo demuestra como crear un mapa anónimo y cambiar los datos entre los procesos padre e hijo:

```
import mmap
import os

mm = mmap.mmap(-1, 13)
mm.write(b"Hello world!")

pid = os.fork()

if pid == 0:  # In a child process
    mm.seek(0)
    print(mm.readline())

    mm.close()
```

Lanza un evento de inspección `mmap.__new__` con los argumentos `fileno`, `length`, `access`, `offset`.

Los objetos de archivos mapeados en memoria soportan los siguiente métodos:

close()

Cierra el mmap. Las llamadas posteriores a otros métodos del objeto resultarán en que se lance una excepción *ValueError*. Esto no cerrará el archivo abierto.

closed

`True` si el archivo está cerrado.

Added in version 3.2.

find(*sub*[, *start*[, *end*]])

Retorna el índice mínimo en el objeto donde la subsecuencia *sub* es hallada, tal que *sub* este contenido en el rango [*start*, *end*]. Los argumentos opcionales *start* y *end* son interpretados como en una notación de rebanada. Retorna `-1` si falla.

Distinto en la versión 3.5: Ahora el objeto bytes-like object con permisos de escritura se acepta.

flush()

flush(*offset*, *size*, */*)

Transmite los cambios hechos a la copia en memoria de una archivo de vuelta al archivo. Sin el uso de esta llamada no hay garantía que los cambios sean escritos de vuelta antes de que los objetos sean destruidos. Si *offset* y *size* son especificados, solo los cambios al rango de bytes dado serán transmitidos al disco; de otra forma, la extensión completa al mapeado se transmite. *offset* debe ser un múltiplo de la constante **`PAGESIZE`** o **`ALLOCATIONGRANULARITY`**.

Se retorna `None` para indicar éxito. Una excepción es lanzada cuando la llamada falla.

Distinto en la versión 3.8: Anteriormente, se retornaba un valor diferente de cero cuando era exitoso; se retornaba cero cuando pasaba un error en Windows. Se retornaba un valor de cero cuando era exitoso; se lanzaba una excepción cuando pasaba un error en Unix.

madvise(*option*[, *start*[, *length*]])

Envía un aviso *option* al kernel sobre la región de la memoria que comienza con *start* y se extiende *length* bytes. *option* debe ser una de las constantes MADV_* disponibles en el sistema. Si *start* y *end* se omiten, se abarca al mapeo entero. En algunos sistemas (incluyendo Linux), *start* debe ser un múltiplo de **`PAGESIZE`**.

Disponibilidad: Sistemas con la llamada al sistema `madvise()`.

Added in version 3.8.

move(*dest*, *src*, *count*)

Copia los *count* bytes empezando en el *offset src* al índice de destino *dest*. Si el mmap fue creado con **`ACCESS_READ`**, entonces las llamadas lanzaran una excepción **`TypeError`**.

read([*n*])

Retorna una clase **`bytes`** que contiene hasta *n* bytes empezando desde la posición del archivo actual. Si se omite el argumento, es `None` o negativo, retorna todos los bytes desde la posición actual del archivo hasta el final del mapeado. Se actualiza la posición del archivo para apuntar después de los bytes que se retornaron.

Distinto en la versión 3.3: El argumento puede ser omitido o ser `None`.

read_byte()

Retorna un byte en la posición actual del archivo como un entero, y avanza la posición del archivo por 1.

readline()

Retorna una sola línea, comenzando en la posición actual del archivo y hasta la siguiente nueva línea. La posición del archivo se actualiza para apuntar después de los bytes que se retornaron.

resize(*newsize*)

Resizes the map and the underlying file, if any.

Resizing a map created with *access* of **`ACCESS_READ`** or **`ACCESS_COPY`**, will raise a **`TypeError`** exception. Resizing a map created with *trackfd* set to `False`, will raise a `ValueError` exception.

En Windows: cambiar el tamaño del mapa generará un `OSError` si hay otros mapas contra el archivo con el mismo nombre. Cambiar el tamaño de un mapa anónimo (es decir, contra el archivo de paginación) creará silenciosamente un nuevo mapa con los datos originales copiado hasta la longitud del nuevo tamaño.

Distinto en la versión 3.11: Falla correctamente si se intenta cambiar el tamaño cuando se sostiene otro mapa. Permite cambiar el tamaño contra un mapa anónimo en Windows

rfind(*sub*[, *start*[, *end*]])

Retorna el índice más alto en el objeto donde la subsecuencia *sub* se encuentre, tal que *sub* sea contenido en el rango [*start*, *end*]. Los argumentos opcionales *start* y *end* son interpretados como un notación de rebanada. Retorna `-1` si falla.

Distinto en la versión 3.5: Ahora el objeto bytes-like object con permisos de escritura se acepta.

seek(*pos*[, *whence*])

Establece la posición actual del archivo. El argumento *whence* es opcional y es por defecto `os.SEEK_SET` o `0` (posicionamiento del archivo absoluto); otros valores son `os.SEEK_CUR` o `1` (búsqueda relativa a la posición actual) y `os.SEEK_END` o `2` (búsqueda relativa al final del archivo).

Distinto en la versión 3.13: Return the new absolute position instead of `None`.

seekable()

Return whether the file supports seeking, and the return value is always `True`.

Added in version 3.13.

size()

Retorna el tamaño del archivo, que puede ser más grande que el tamaño del área mapeado en memoria.

tell()

Retorna la posición actual del puntero del archivo.

write(*bytes*)

Escribe los bytes en *bytes* en memoria en la posición actual del puntero del archivo y retorna el números de bytes escritos (nunca menos que `len(bytes)`, ya que si la escritura falla, una excepción **`ValueError`** será lanzada). La posición del archivo es actualizada para apuntar después de los bytes escritos. Si el mmap fue creado con **`ACCESS_READ`**, entonces escribirlo lanzará una excepción **`TypeError`**.

Distinto en la versión 3.5: Ahora el objeto bytes-like object con permisos de escritura se acepta.

Distinto en la versión 3.6: Ahora se retorna el número de bytes escritos.

write_byte(*byte*)

Escribe el entero *byte* en la memoria en la posición actual del puntero del archivo; se avanza la posición del archivo por `1`. Si el mmap es creado con **`ACCES_READ`**, entonces escribirlo hará que se lance la excepción **`TypeError`**.

Constantes MADV_*

mmap.MADV_NORMAL
mmap.MADV_RANDOM
mmap.MADV_SEQUENTIAL

mmap.MADV_WILLNEED
mmap.MADV_DONTNEED
mmap.MADV_REMOVE
mmap.MADV_DONTFORK
mmap.MADV_DOFORK
mmap.MADV_HWPOISON
mmap.MADV_MERGEABLE
mmap.MADV_UNMERGEABLE
mmap.MADV_SOFT_OFFLINE
mmap.MADV_HUGEPAGE
mmap.MADV_NOHUGEPAGE
mmap.MADV_DONTDUMP
mmap.MADV_DODUMP
mmap.MADV_FREE
mmap.MADV_NOSYNC
mmap.MADV_AUTOSYNC
mmap.MADV_NOCORE
mmap.MADV_CORE
mmap.MADV_PROTECT
mmap.MADV_FREE_REUSABLE
mmap.MADV_FREE_REUSE

Se pueden pasar estas opciones al método **`mmap.madvise()`**. No todas las opciones estarán presentes en todos los sistemas.

Disponibilidad: Sistemas con la llamada al sistema *madvise()*.

Added in version 3.8.

Constantes MAP_*

mmap.MAP_SHARED
mmap.MAP_PRIVATE
mmap.MAP_32BIT
mmap.MAP_ALIGNED_SUPER
mmap.MAP_ANON
mmap.MAP_ANONYMOUS
mmap.MAP_CONCEAL
mmap.MAP_DENYWRITE

mmap.MAP_EXECUTABLE
mmap.MAP_HASSEMAPHORE
mmap.MAP_JIT
mmap.MAP_NOCACHE
mmap.MAP_NOEXTEND
mmap.MAP_NORESERVE
mmap.MAP_POPULATE
mmap.MAP_RESILIENT_CODESIGN
mmap.MAP_RESILIENT_MEDIA
mmap.MAP_STACK
mmap.MAP_TPRO
mmap.MAP_TRANSLATED_ALLOW_EXECUTE
mmap.MAP_UNIX03

These are the various flags that can be passed to `mmap.mmap()`. **`MAP_ALIGNED_SUPER`** is only available at FreeBSD and **`MAP_CONCEAL`** is only available at OpenBSD. Note that some options might not be present on some systems.

Distinto en la versión 3.10: Added **`MAP_POPULATE`** constant.

Added in version 3.11: Added **`MAP_STACK`** constant.

Added in version 3.12: Added **`MAP_ALIGNED_SUPER`** and **`MAP_CONCEAL`** constants.

Added in version 3.13: Added **`MAP_32BIT`**, **`MAP_HASSEMAPHORE`**, **`MAP_JIT`**, **`MAP_NOCACHE`**, **`MAP_NOEXTEND`**, **`MAP_NORESERVE`**, **`MAP_RESILIENT_CODESIGN`**, **`MAP_RESILIENT_MEDIA`**, **`MAP_TPRO`**, **`MAP_TRANSLATED_ALLOW_EXECUTE`**, and **`MAP_UNIX03`** constants.

Manejo de datos de internet

Este capítulo describe los módulos que admiten el manejo de formatos de datos comúnmente usados en Internet.

- `email` — An email and MIME handling package
    - `email.message`: Representing an email message
    - `email.parser`: Parsing email messages
        - API *FeedParser*
        - API *Parser*
        - Notas adicionales
    - `email.generator`: Generating MIME documents
    - `email.policy`: Policy Objects
    - `email.errors`: Exception and Defect classes
    - `email.headerregistry`: Custom Header Objects
    - `email.contentmanager`: Managing MIME Content
        - Instancias gestoras de contenido
    - `email`: Ejemplos
    - `email.message.Message`: Representar un mensaje de correo electrónico usando la API `compat32`
    - `email.mime`: Creating email and MIME objects from scratch
    - `email.header`: Internationalized headers
    - `email.charset`: Representing character sets
    - `email.encoders`: Encoders
    - `email.utils`: Miscellaneous utilities
    - `email.iterators`: Iterators
- `json` — JSON encoder and decoder
    - Uso básico
    - Codificadores y Decodificadores

    - Excepciones
    - Cumplimiento e interoperabilidad estándar
        - Codificaciones de caracteres
        - Valores de número infinito y NaN
        - Nombres repetidos dentro de un objeto
        - Valores de nivel superior No-Objeto , No-Arreglo
        - Limitaciones de la implementación
    - Command-line interface
        - Command-line options
- **`mailbox`** — Manipulate mailboxes in various formats
    - **`Mailbox`** objects
        - **`Maildir`** objects
        - **`mbox`** objects
        - **`MH`** objects
        - **`Babyl`** objects
        - **`MMDF`** objects
    - **`Message`** objects
        - **`MaildirMessage`** objects
        - **`mboxMessage`** objects
        - **`MHMessage`** objects
        - **`BabylMessage`** objects
        - **`MMDFMessage`** objects
    - Excepciones
    - Ejemplos
- **`mimetypes`** — Map filenames to MIME types
    - MimeTypes objects
    - Command-line usage
    - Command-line example
- **`base64`** — Base16, Base32, Base64, Base85 Data Encodings

    - RFC 4648 Encodings
    - Base85 Encodings
    - Legacy Interface
    - Consideraciones de Seguridad

- **`binascii`** — Convert between binary and ASCII
- **`quopri`** — Encode and decode MIME quoted-printable data

`email` — An email and MIME handling package

Código fuente Lib/email/_init_.py [https://github.com/python/cpython/tree/3.14/Lib/email/_init_.py]

The **`email`** package is a library for managing email messages. It is specifically *not* designed to do any sending of email messages to SMTP (**RFC 2821** [https://datatracker.ietf.org/doc/html/rfc2821.html]), NNTP, or other servers; those are functions of modules such as **`smtplib`**. The **`email`** package attempts to be as RFC-compliant as possible, supporting **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html] and **RFC 6532** [https://datatracker.ietf.org/doc/html/rfc6532.html], as well as such MIME-related RFCs as **RFC 2045** [https://datatracker.ietf.org/doc/html/rfc2045.html], **RFC 2046** [https://datatracker.ietf.org/doc/html/rfc2046.html], **RFC 2047** [https://datatracker.ietf.org/doc/html/rfc2047.html], **RFC 2183** [https://datatracker.ietf.org/doc/html/rfc2183.html], and **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html].

La estructura general del paquete de correo electrónico se puede dividir en tres componentes principales, más un cuarto componente que controla el comportamiento de los otros componentes.

El componente central del paquete es un «modelo de objetos» que representa los mensajes de correo electrónico. Una aplicación interactúa con el paquete principalmente a través de la interfaz del modelo de objetos definida en el submódulo **`message`**. La aplicación puede usar esta API para hacer preguntas sobre un correo electrónico existente, para construir un nuevo correo electrónico o para agregar o eliminar subcomponentes de correo electrónico que utilizan la misma interfaz de modelo de objetos. Es decir, siguiendo la naturaleza de los mensajes de correo electrónico y sus subcomponentes MIME, el modelo de objetos de correo electrónico es una estructura de árbol de objetos que proporcionan la API `EmailMessage`.

Los otros dos componentes principales del paquete son `parser` y `generator`. El parser toma la versión serializada de un mensaje de correo electrónico (una secuencia de bytes) Y la convierte en un árbol de objetos `EmailMessage`. El generador toma un `EmailMessage` y lo convierte de nuevo en un flujo de bytes serializado. (El analizador y el generador también manejan flujos de caracteres de texto, pero se desaconseja este uso ya que es demasiado fácil terminar con mensajes que no son válidos de una forma u otra).

El componente de control es el módulo de `policy`. Cada `EmailMessage` cada `generator`, y cada `parser` tiene un objeto de `policy` asociado que controla su comportamiento. Por lo general, una aplicación solo necesita especificar la política cuando se crea un `EmailMessage` , ya sea instanciando directamente un `EmailMessage` para crear un nuevo correo electrónico o analizando un flujo de entrada con un `parser`. Pero la política se puede cambiar cuando el mensaje se serializa mediante un `generator`. Esto permite, por ejemplo, analizar un mensaje de correo electrónico genérico desde el disco, pero serializarlo utilizando la configuración estándar de SMTP al enviarlo a un servidor de correo electrónico.

El paquete de correo electrónico hace todo lo posible para ocultar los detalles de las diversas RFC que rigen de la aplicación. Conceptualmente, la aplicación debería poder tratar el mensaje de correo electrónico como un árbol estructurado de texto Unicode y archivos adjuntos binarios, sin tener que preocuparse por cómo se representan estos cuando se serializan. En la práctica, sin embargo, a menudo es necesario conocer al menos algunas de las reglas que rigen los mensajes MIME y su estructura, específicamente los nombres y la naturaleza de los «tipos de contenido» MIME y cómo identifican los documentos de varias partes. En su mayor parte, este conocimiento solo debería ser necesario para aplicaciones más complejas, e incluso entonces debería ser solo la estructura de alto nivel en cuestión, y no los detalles de cómo se representan esas estructuras. Dado que los tipos de contenido MIME se utilizan ampliamente en el software moderno de Internet (no solo en el correo electrónico), este será un concepto familiar para muchos programadores.

The following sections describe the functionality of the **`email`** package. We start with the **`message`** object model, which is the primary interface an application will use, and follow that with the **`parser`** and **`generator`** components. Then we cover the **`policy`** controls, which completes the treatment of the main components of the library.

Las siguientes tres secciones cubren las excepciones que puede generar el paquete y los defectos (incumplimiento de las RFC) que el **`parser`** puede detectar. Luego cubrimos los subcomponentes **`headerregistry`** y **`contentmanager`**, que proporcionan herramientas para realizar una manipulación más detallada de los encabezados y cargas útiles, respectivamente. Ambos componentes contienen características relevantes para consumir y producir mensajes no triviales, pero también documentan sus API de extensibilidad, que serán de interés para aplicaciones avanzadas.

A continuación, se muestra un conjunto de ejemplos del uso de las partes fundamentales de las API cubiertas en las secciones anteriores.

Lo anterior representa la API moderna (compatible con Unicode) del paquete de correo electrónico. Las secciones restantes, comenzando con la clase **`Message`**, cubren la API **`compat32`** heredada que trata mucho más directamente con los detalles de cómo se representan los mensajes de correo electrónico. La API **`compat32`** no oculta los detalles de las RFC de la aplicación, pero para las aplicaciones que necesitan operar a ese nivel, pueden ser herramientas útiles. Esta documentación también es relevante para las aplicaciones que todavía usan la API **`compat32`** por razones de compatibilidad con versiones anteriores.

Distinto en la versión 3.6: Documentos reorganizados y reescritos para promover la nueva API **`EmailMessage`**/**`EmailPolicy`**.

Contents of the **`email`** package documentation:

- **`email.message`**: Representing an email message

- `email.parser`: Parsing email messages
    - API *FeedParser*
    - API *Parser*
    - Notas adicionales
- `email.generator`: Generating MIME documents
- `email.policy`: Policy Objects
- `email.errors`: Exception and Defect classes
- `email.headerregistry`: Custom Header Objects
- `email.contentmanager`: Managing MIME Content
    - Instancias gestoras de contenido
- `email`: Ejemplos

API heredada:

- `email.message.Message`: Representar un mensaje de correo electrónico usando la API `compat32`
- `email.mime`: Creating email and MIME objects from scratch
- `email.header`: Internationalized headers
- `email.charset`: Representing character sets
- `email.encoders`: Encoders
- `email.utils`: Miscellaneous utilities
- `email.iterators`: Iterators

Ver también

Módulo `smtplib`
Cliente SMTP (Protocolo simple de transporte de correo)

Módulo `poplib`
Cliente POP (Protocolo de oficina postal)

Módulo `imaplib`
Cliente IMAP (Protocolo de acceso a mensajes de Internet)

Módulo `mailbox`
Herramientas para crear, leer y administrar colecciones de mensajes en disco utilizando una variedad de formatos

estándar.

[1] [2]

`email.message`: Representing an email message

Código fuente: Lib/email/message.py [https://github.com/python/cpython/tree/3.14/Lib/email/message.py]

Added in version 3.6: [1]

The central class in the `email` package is the `EmailMessage` class, imported from the `email.message` module. It is the base class for the `email` object model. `EmailMessage` provides the core functionality for setting and querying header fields, for accessing message bodies, and for creating or modifying structured messages.

Un mensaje de correo electrónico consiste en *headers* y un *payload* (al que también nos referimos como *content*). *Headers* como **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html] o **RFC 6532** [https://datatracker.ietf.org/doc/html/rfc6532.html] son nombres de campos de estilo y valores, donde el nombre y valor están separados por un “:”. Los dos puntos no son parte ni del nombre ni del valor. El *payload* puede ser un simple mensaje, un objeto binario, o una secuencia estructurada de sub-mensajes, cada uno con su propio conjunto de *headers* y su propio *payload*. El último tipo de *payload* es indicado por el mensaje con un MIME como *multipart/** o *message/rfc822* .

El modelo conceptual provisto por un objeto `EmailMessage` es el de un diccionario ordenado de cabeceras emparejadas con una carga útil que representa al cuerpo del mensaje **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html], que podría ser una lista de objetos sub-`EmailMessage`. Además de los métodos normales de diccionario para acceder a las cabeceras y valores, tiene métodos para acceder a información especializada desde las cabeceras (por ejemplo, el tipo de contenido MIME), para operar en la carga útil, generar una versión serializada del mensaje, y recorrer recursivamente el árbol de objetos.

The **`EmailMessage`** dictionary-like interface is indexed by the header names, which must be ASCII values. The values of the dictionary are strings with some extra methods. Headers are stored and returned in case-preserving form, but field names are matched case-insensitively. The keys are ordered, but unlike a real dict, there can be duplicates. Additional methods are provided for working with headers that have duplicate keys.

La carga útil es un objeto de cadena de caracteres o bytes, en el caso de objetos de mensaje simples, o una lista de objetos **`EmailMessage`**, para documentos de contenedor MIME como *multipart/** y objetos de mensaje *message/rfc822*.

class email.message.EmailMessage(*policy=default*)

If *policy* is specified use the rules it specifies to update and serialize the representation of the message. If *policy* is not set, use the **`default`** policy, which follows the rules of the email RFCs except for line endings (instead of the RFC mandated `\r\n`, it uses the Python standard `\n` line endings). For more information see the **`policy`** documentation. [2]

as_string(*unixfrom=False, maxheaderlen=None, policy=None*)

Retorna el mensaje entero como cadena de caracteres. Cuando la opción *unixform* es verdadera, la cabecera está incluida en la cadena de caracteres retornada. *unixform* está predeterminado con valor `False`. Por compatibilidad con versiones anteriores, la clase base **`Message`** *maxheaderlen* es aceptada pero con valor `None` como predeterminado, por lo que la longitud de línea se controla mediante **`max_line_length`** de la política. El argumento *policy* puede ser usado para sobrescribir la política predeterminada obtenida de la instancia del mensaje. Esto puede ser usado para controlar parte del formato producido por el método, ya que la *policy* especificada pasará a **`Generator`**.

Aplanar el mensaje puede acarrear cambios en

`EmailMessage` si es necesario rellenar los valores predeterminados para completar la transformación a una cadena de caracteres (por ejemplo, se pueden generar o modificar límites MIME).

Tenga en cuenta que este método se proporciona como una comodidad y quizás no sea la forma más eficiente de serializar mensajes en su aplicación, especialmente si estás tratando con múltiples mensajes. Consulte **`Generator`** por una API más flexible para serializar mensajes. No olvide también que este método está restringido a producir mensajes serializados como «7 bit clean» cuando **`utf8`** es `False`, que es el valor predeterminado.

Distinto en la versión 3.6: el comportamiento predeterminado cuando *maxheaderlen* no está especificado cambió de 0 al valor de *max_line_length* de la política .

__str__()

Equivalente a `as_string(policy=self.policy.clone(utf8=True))`. Permite `str(msg)` para producir una cadena que contenga un mensaje serializado en un formato legible.

Distinto en la versión 3.4: el método se cambió para usar `utf8=True`, produciendo así un **RFC 6531** [https://datatracker.ietf.org/doc/html/rfc6531.html] como representación del mensaje, en vez de ser un alias de **`as_string()`**.

as_bytes(*unixfrom=False*, *policy=None*)

Retorna el mensaje plano como un objeto de bytes. Cuando *unixform* es verdadero, la cabecera es incluida en la cadena de caracteres retornada. El valor predeterminado de *unixform* es `False`. El argumento *policy* puede ser usado para sobreescribir el valor predeterminado obtenido de la instancia del mensaje.

Esto puede ser usado para controlar parte del formato producido por el método, ya que la política especificada pasará a **`Generator`**.

Aplanar el mensaje puede acarrear cambios en **`EmailMessage`** si es necesario rellenar los valores predeterminados para completar la transformación a una cadena de caracteres (por ejemplo, se pueden generar o modificar límites MIME).

Tenga en cuenta que este método se proporciona como una comodidad y quizás no sea la forma más eficiente de serializar mensajes en su aplicación, especialmente si estas tratando con múltiples mensajes. Consulte **`Generator`** por una API más flexible para serializar mensajes.

bytes()

Equivalent to **`as_bytes()`**. Allows `bytes(msg)` to produce a bytes object containing the serialized message.

is_multipart()

Retorna `True` si la carga útil del mensaje es una lista de objetos de sub-**`EmailMessage`**, de otra manera retorna `False`. Cuando **`is_multipart()`** retorna `False`, la carga útil deberá ser un objeto cadena de caracteres (que podría ser una carga útil binaria codificada con CTE). Note que si **`is_multipart()`** retorna `True` no necesariamente significa que «msg.get_content_maintype() = = "multipart"» retornará `True`. Por ejemplo, `is_multipart` retornará `True` cuando la **`EmailMessage`** sea del tipo `message/rfc822`.

set_unixfrom(*unixfrom*)

Configura la cabecera del mensaje a *unixform*, que debería ser una cadena de caracteres. (Consulte **`mboxMessage`** para una descripción de esta cabecera) get_unixfrom()

Retorna la cabecera del mensaje. Predeterminado `None` si la cabecera no ha sido configurada.

Los siguientes métodos implementan el mapeo como una interfaz para acceder a la cabecera del mensaje. Tenga en cuenta que hay algunas diferencias semánticas entre esos métodos y una interfaz de mapeo normal(es decir, diccionario). Por ejemplo, en un diccionario no hay claves duplicadas, pero pueden haber cabeceras duplicadas. Además, en los diccionarios no hay un orden garantizado para las claves retornadas por **`keys()`**, pero en un objeto **`EmailMessage`**, las cabeceras siempre regresan en orden de aparición en el mensaje original, o en el que fueron agregadas luego. Cualquier cabecera borrada y vuelta a añadir siempre se agrega al final de la lista.

Estas diferencias semánticas son intencionales y están sesgadas hacia la conveniencia en los casos de uso más comunes.

Note que en todos los casos, cualquier cabecera presente en el mensaje no se incluye en la interfaz de mapeo.

len()

Retorna el número total de cabeceras, incluidas las duplicadas.

contains(*name*)

Retorna `True` si el objeto del mensaje tiene un campo llamado “nombre”. La comparación se realiza sin tener en cuenta mayúsculas o minúsculas y “nombre” no incluye “:”. Se utiliza para el operador `in` Por ejemplo:

```
if 'message-id' in myMessage:
   print('Message-ID:', myMessage['message-id']
```

getitem(*name*)

Retorna el valor de la cabecera nombrada. *name* no incluye el separador de dos puntos, “:”. Si la cabecera se pierde, regresa `None`, un **`KeyError`** no se genera nunca.

Tenga en cuenta que si el campo nombrado aparece más de una vez en la cabecera del mensaje, esa cantidad de veces el valor regresado será indefinido. Use el método **`get_all()`** para obtener los valores de todas las cabeceras existentes llamadas *name*.

Usando el *standard non-“compat32”*, el valor regresado es una instancia de una subclase de **`email.headerregistry.BaseHeader`**.

setitem(*name*, *val*)

Agrega una cabecera al mensaje con un campo “nombre” y un valor “val”. El campo se agrega al final de las cabeceras existentes en el mensaje.

Tenga en cuenta que esto no sobrescribe ni borra ninguna cabecera con el mismo nombre. Si quiere asegurarse de que la nueva cabecera es la única en el mensaje con el campo “nombre”, borre el campo primero, por ejemplo:

```
del msg['subject']
msg['subject'] = 'Python roolz!'
```

Si el **`policy`** define ciertas cabeceras para ser únicas (como lo hacen las políticas estándares), este método puede generar un **`ValueError`** cuando se intenta asignar un valor a esta cabecera si una ya existe. Este comportamiento es intencional por consistencia, pero no dependa de ello, ya que podemos optar por hacer que tales asignaciones eliminen la cabecera en el futuro.

delitem(*name*)

Elimine todas las apariciones del campo “nombre” de las cabeceras del mensaje. No se genera ninguna excepción si el campo nombrado no está presente en las cabeceras.

keys()

Retorna una lista de los nombres de todos los campos de la cabecera del mensaje.

values()

Retorna una lista de todos los valores de los campos del mensaje.

items()

Retorna una lista de tuplas de dos elementos que contienen todos los campos cabeceras y valores del mensaje.

get(*name*, *failobj=None*)

Return the value of the named header field. This is identical to `__getitem__()` except that optional *failobj* is returned if the named header is missing (*failobj* defaults to `None`).

Aquí hay algunos métodos adicionales útiles relacionados con la cabecera:

get_all(*name*, *failobj=None*)

Retorna una lista de todos los valores para el campo *nombre*. Si no se nombran tales cabeceras en el mensaje, regresa *failobj* (por defecto `None`)

add_header(*_name*, *_value*, ***_params*)

Configuración extendida de cabeceras. Este método es similar a `__setitem__()`, excepto que se pueden proporcionar parámetros de cabecera adicionales como argumentos de palabras clave.

Por cada ítem en los parámetros del diccionario *_params*, la clave se toma como el nombre del parámetro, con barra baja ("_") convertidos a guiones ("-") (ya que en Python no se permiten "-" como identificadores). Normalmente, el parámetro debe ser añadido como `key='value'` a menos que el valor sea `None`, en ese caso solo la clave debe ser añadida.

Si el valor contiene caracteres no-ASCII, el *charset* y el lenguaje deben ser controlados especificando el valor como una triple tupla en formato `(CHARSET, LANGUAJE, VALUE)`, donde `CHARSET` es una *string* llamando al *charset* usado para codificar el valor, `LANGUAJE` generalmente se establece en `None` o en una cadena de caracteres vacía (consulte **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html] para más opciones), y `VALUE` es el valor de la cadena de caracteres que contiene puntos de código no-ASCII. Si la triple tupla no pasa y el valor contiene caracteres no-ASCII, es automáticamente codificada en formato **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html], usando `CHARSET` de `utf-8` y `LANGUAJE` `None`.

Aquí hay un ejemplo:

```
msg.add_header('Content-Disposition', 'attachme
```

Esto agregará una cabecera que se ve como:

```
Content-Disposition: attachment; filename="bud.
```

Un ejemplo de la interfaz extendida con caracteres no-ASCII:

```
msg.add_header('Content-Disposition', 'attachme
               filename=('iso-8859-1', '', 'Fuß
```

replace_header(*_name*, *_value*)

Reemplaza una cabecera. Reemplaza la primer cabecera encontrada en el mensaje que coincida con *_name*, conservando el orden de cabeceras y uso de minúsculas (y mayúsculas) del nombre de campo de la cabecera original. Si no hay coincidencia, se lanzará un `KeyError`.

get_content_type()

Retorna el tipo de contenido del mensaje, pasado a minúsculas de la forma *maintype/subtype*. Si no hay cabecera llamada *Content-Type* en el mensaje, regresa el valor de `get_default_type()`. Si *Content-Type* no es válido, retorna `text/plain`.

(De acuerdo con **RFC 2045** [https://datatracker.ietf.org/doc/html/rfc2045.html], los mensajes siempre tienen un tipo predeterminado, **`get_content_type`siempre retornará un valor. :rfc:`2045`define el tipo predeterminado de un mensaje como :mimetype:`text/plain()`** a menos que aparezca dentro de un contenedor *multipart/digest*, en cuyo caso sería *message/rfc822*. Si la cabecera *Content-Type* tiene una especificación de tipo que sea inválida, **RFC 2045** [https://datatracker.ietf.org/doc/html/rfc2045.html] exige que el tipo predeterminado sea *text/plain*.)

get_content_maintype()

Retorna el tipo de contenido principal del mensaje. Esta es la parte *maintype* de la cadena retornada por `get_content_type()`.

get_content_subtype()

Retorna el tipo del sub-contenido del mensaje. Esta es la parte *subtype* de la cadena retornada por `get_content_type()`.

get_default_type()

Retorna el tipo de contenido predeterminado. La mayoría de los mensajes tienen como tipo de contenido predeterminado a *text/plain*, a excepción de los mensajes que son sub-partes de contenedores *multipart/digest*. Estas sub-partes tienen como tipo de contenido predeterminado a *message/rfc822*.

set_default_type(*ctype*)

Establece el tipo de contenido predeterminado. *ctype* debería ser *text/plain* o *message/rfc822*, aunque esto no está impuesto. El tipo de contenido predeterminado no se almacena en la cabecera *Content-Type*, así que sólo afecta al valor de retorno de los métodos `get_content_type` cuando ninguna cabecera *Content-Type* está presente en el mensaje.

set_param(*param*, *value*, *header='Content-Type'*, *requote=True*, *charset=None*, *language=''*, *replace=False*)

Establece un parámetro en el encabezado *Content-Type*. Si el parámetro ya existe en el encabezado, reemplace su valor con *value*. Cuando *header* es `Content-Type` (el predeterminado) y el encabezado aún no existe en el mensaje, agréguelo, establece su valor en *text/plain* y agregue el nuevo valor del parámetro. *header* opcional especifica un encabezado alternativo a *Content-Type*.

Si el valor contiene caracteres *non-ASCII*, el *charset* y el lenguaje pueden ser especificados explícitamente con los parámetros *charset* y *language*. Opcionalmente *language* especifica el lenguaje **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html], por defecto una cadena vacía. Ambos *charset* y *language* deberán ser cadenas. los valores predeterminados son `utf8` para *charset* y `None` para *language*.

Si *replace* es `False` (predeterminado) la cabecera se mueve al final de la lista de cabeceras. Si *replace* es `True`, la cabecera se actualizará en su lugar.

El uso del parámetro *requote* con objetos **`EmailMessage`** está obsoleto.

Tenga en cuenta que se puede acceder a los parámetros existentes de las cabeceras a través del atributo **`params`** de la cabecera (por ejemplo, `msg['Content-Type'].params['charset']`).

Distinto en la versión 3.4: Se agregó la palabra clave `replace`.

del_param(*param*, *header='content-type'*, *requote=True*)

Elimina completamente el parámetro dado de la cabecera *Content-Type*. La cabecera será reescrita en su lugar, sin el parámetro o su valor. Opcionalmente *header* especifica una alternativa a *Content-Type*.

El uso del parámetro *requote* con objetos **`EmailMessage`** está obsoleto.

get_filename(*failobj=None*)

Retorna el valor del parámetro `filename` de la cabecera *Content-Disposition* del mensaje. Si la cabecera no tiene un parámetro `filename`, este método recae a buscar el parámetro `name` en la cabecera *Content-Type*. Si ninguno se encuentra o falta la cabecera, se retorna *failobj*. La cadena retornada siempre estará sin comillas según **`email.utils.unquote()`**.

get_boundary(*failobj=None*)

Retorna el parámetro `boundary` de la cabecera *Content-Type* del mensaje, o *failobj* si la cabecera no se encuentra o si no tiene un parámetro `boundary`. La cadena retornada siempre estará sin comillas según **`email.utils.unquote()`**.

set_boundary(*boundary*)

Establece el parámetro `boundary` de la cabecera *Content-Type* como *boundary*. **`set_boundary()`** siempre que sea necesario pondrá comillas a *boundary*. Si el objeto mensaje no tiene cabecera *Content-Type* se lanza **`HeaderParseError`**.

Note que usar este método es sutilmente diferente a borrar la cabecera *Content-Type* antigua y agregar una nueva con el nuevo límite utilizando **`add_header()`**, porque **`set_boundary()`** preserva el orden de la cabecera *Content-Type* en la lista de cabeceras.

get_content_charset(*failobj=None*)

Retorna el parámetro `charset` de la cabecera *Content-Type`forzado a minúsculas. Si no hay una cabecera :mailheader:`Content-Type*, o si esa cabecera no tiene el parámetro `charset`, se retorna *failobj*.

get_charsets(*failobj=None*)

Retorna una lista que contiene los nombres de los *charset* en el mensaje. Si el mensaje es *multipart*, la lista contendrá un elemento por cada subparte en la carga útil, de lo contrario será una lista de longitud 1.

Cada elemento de la lista será una cadena que tendrá el valor del parámetro `charset` en la cabecera *Content-Type* para la subparte representada. Si la subparte no tiene cabecera *Content-Type*, no tiene parámetro `charset`, o no es del tipo MIME principal *text*, entonces ese elemento en la lista retornada será *failobj*.

is_attachment()

Retorna `True` si hay una cabecera *Content-Disposition* y su valor (sensible a mayúsculas) es `attachment`, de lo contrario retorna `False`.

Distinto en la versión 3.4.2: is_attachment ahora es un método en lugar de una propiedad, por consistencia con **`is_multipart()`**.

get_content_disposition()

Retorna el valor en minúsculas (sin parámetros) de la cabecera *Content-Disposition* si el mensaje la tiene, de lo contrario retorna `None`. Los valores posibles para este método son *inline*, *attachment* o `None` si el mensaje sigue **RFC 2183** [https://datatracker.ietf.org/doc/html/rfc2183.html].

Added in version 3.5.

Los siguientes métodos se refieren a interrogar y manipular el contenido (*payload*) del mensaje.

walk()

El método **`walk()`** es un generador multipropósito que puede usarse para iterar sobre todas las partes y subpartes del árbol de un objeto mensaje, ordenando el recorrido en profundidad primero. Típicamente se usaría **`walk()`** como el iterador en un ciclo `for`; cada iteración retornará la siguiente subparte.

Aquí hay un ejemplo que imprime el tipo MIME de cada parte de una estructura de mensaje de varias partes:

```
>>> for part in msg.walk():
...     print(part.get_content_type())
multipart/report
text/plain
message/delivery-status
text/plain
text/plain
message/rfc822
text/plain
```

`walk` itera sobre las subpartes de cualquier parte donde **`is_multipart()`** retorna `True`, aun si `msg.get_content_maintype() == 'multipart'` pueda retornar `False`. Podemos ver esto en nuestro ejemplo al hacer uso de la función auxiliar de depuración `_structure`:

```
>>> from email.iterators import _structure
>>> for part in msg.walk():
...     print(part.get_content_maintype() == 'm
...           part.is_multipart())
True True
False False
False True
False False
False False
False True
False False
>>> _structure(msg)
multipart/report
    text/plain
    message/delivery-status
        text/plain
        text/plain
    message/rfc822
        text/plain
```

Aquí las partes `message` no son `multiparts`, pero sí contienen subpartes. `is_multipart()` retorna `True` y `walk` desciende a las subpartes.

get_body(*preferencelist=('related', 'html', 'plain')*)

Retorna la parte MIME que es la mejor candidata para ser el «cuerpo» del mensaje.

preferencelist debe ser una secuencia de cadenas del conjunto `related`, `html`, y `plain`, e Indica el orden de preferencia para el tipo de contenido de la parte retornada.

Empieza a buscar coincidencias candidatas con el objeto en el que se llama al método `get_body`”.

Si `related` no está incluido en la *preferencelist,* considere la parte raíz (o subparte de la parte raíz) de cualquier relacionado encontrado como candidato si la (sub-)parte coincide con una preferencia.

Cuando se encuentra una `multipart/related`, verifica el parámetro `start` y si se encuentra una parte con un *Content-ID* que coincida, considera solamente a ésta cuando se busca coincidencias candidatas. De otra forma considera sólo la primera parte (predeterminado *root*) de la `multipart/related`.

Si una parte tiene una cabecera *Content-Disposition* , sólo se considera a ésta parte como coincidencia candidata si el valor de la cabecera es `inline`.

Si ninguno de los candidatos coincide con ninguna de las preferencias en *preferencelist*, retorna `None`.

Notas: (1) Para la mayoría de las aplicaciones las únicas combinaciones de *preferencelist* que realmente tienen sentido son `('plain',)`, `('html', 'plain')`, y la predeterminada `('related', 'html', 'plain')`. (2) Debido a que la coincidencia comienza con el objeto en el que se llama a `get_body`, llamar a `get_body` en un `multipart/related` devolverá el objeto en sí a menos que la *preferencelist* tenga un valor no predeterminado. (3) Los mensajes (o partes de mensajes) que no especifican un *Content-Type* o cuya cabecera *Content-Type* sea inválida se tratarán como si fueran del tipo `text/plain`, que ocasionalmente puede ocasionar que `get_body` retorne resultados inesperados.

iter_attachments()

Devuelve un iterador sobre todas las subpartes inmediatas del mensaje que no son partes candidatas del «body». Es decir, omitir la primer ocurrencia de cada `text/plain`, `text/html`, `multipart/related`, o `multipart/alternative` (a menos que estén marcados explícitamente como adjuntos a través de *Content-Disposition: attachment*) y retornar todas las partes restantes. Cuando se aplica directamente a un `multipart/related`, retorna un iterador sobre todas las partes relacionadas excepto la parte raíz (es decir, la parte apuntada por el parámetro `start`, o la primera parte si no hay un parámetro `start` o el parámetro `start` no coincide con el *Content-ID* de ninguna de las partes). Cuando se aplica directamente a un `multipart/alternative` o a un no-`multipart`, retorna un iterador vacío.

iter_parts()

Retorna un iterador sobre todas las partes inmediatas del mensaje, que estará vacío para una no-`multipart`. (vea también **`walk()`**.)

get_content(**args*, *content_manager=None*, ***kw*)

Llama al método **`get_content()`** del *content_manager*, pasando *self* como el objeto del mensaje y pasando cualquier otro argumento o palabra clave como argumentos adicionales. Si no se especifica *content_manager* se usa el `content_manager` especificado por la **`policy`** actual.

set_content(**args*, *content_manager=None*, ***kw*)

Llama al método **`set_content()`** del *content_manager*, pasando *self* como el objeto *message* y pasando cualquier otro argumento o palabra clave como argumentos adicionales. Si no se especifica *content_manager* se usa el `content_manager` especificado por la **`policy`** actual.

make_related(*boundary=None*)

Convierte un mensaje no-`multipart` a un mensaje `multipart/related`, moviendo cualquier cabecera *Content-* y la carga útil a una (nueva) primera parte del `multipart`. Si *boundary* se especifica, se utiliza como la cadena límite en el *multipart*, de otra forma deja que el límite se cree automáticamente cuando se necesite (por ejemplo, cuando se serializa el mensaje).

make_alternative(*boundary=None*)

Convierte un mensaje no-`multipart` o un mensaje `multipart/related` a uno `multipart/alternative` moviendo cualquier cabecera *Content-* y la carga útil existentes a una (nueva) primera parte del `multipart`. Si *boundary* se especifica, se utiliza como la cadena límite en el *multipart*, de otra forma deja que el límite se cree automáticamente cuando se necesite (por ejemplo, cuando se serializa el mensaje).

make_mixed(*boundary=None*)

Convierte un mensaje no-`multipart`, `multipart/related` o `multipart-alternative` a uno `multipart/mixed` moviendo cualquier cabecera *Content-* y la carga útil existentes a una (nueva) primera parte del `multipart`. Si *boundary* se especifica, se utiliza como la cadena límite en el *multipart*, de otra forma deja que el límite se cree automáticamente cuando se necesite (por ejemplo, cuando se serializa el mensaje).

add_related(**args*, *content_manager=None*, ***kw*)

Si el mensaje es un `multipart/related`, crea un nuevo objeto mensaje, pasa todos los argumentos a su método **`set_content()`** y lo une al `multipart` con **`attach()`**. Si el mensaje es un no-`multipart`, llama a **`make_related()`** y procede como arriba. Si el mensaje es cualquier otro tipo de `multipart`, lanza **`TypeError`**. Si *content_manager* no es especificado, usa el `content_manager` especificado por la **`policy`** actual. Si la parte agregada no tiene un encabezado *Content-Disposition*, se agrega uno con el valor `inline`.

add_alternative(**args*, *content_manager=None*, ***kw*)

Si el mensaje es un `multipart/alternative`, crea un nuevo objeto mensaje, pasa todos los argumentos a su método **`set_content()`** y lo une al `multipart` con **`attach()`**. Si el mensaje es un no-`multipart` o `multipart/related`, llama a **`make_alternative()`** y procede como arriba. Si el mensaje es cualquier otro tipo de `multipart`, lanza **`TypeError`**. Si *content_manager* no es especificado, usa el `content_manager` especificado por la **`policy`** actual.

add_attachment(**args*, *content_manager=None*, ***kw*)

Si el mensaje es un `multipart/mixed`, cree un nuevo objeto de mensaje, pase todos los argumentos a su método **`set_content()`** y **`attach()`** al `multipart`. Si el mensaje no es `multipart`, `multipart/related` o `multipart/alternative`, llame **`make_mixed()`** y luego proceda como se indicó anteriormente. Si no se especifica *content_manager*, use el `content_manager` especificado por el actual **`policy`**. Si la parte agregada no tiene encabezado *Content-Disposition*, agregue uno con el valor `attachment`. Este método se puede utilizar tanto para adjuntos explícitos (*Content-Disposition: attachment*) como para adjuntos `inline` (*Content-Disposition: inline*), pasando las opciones apropiadas al `content_manager`.

clear()

Elimina la carga útil y todas las cabeceras.

clear_content()

Elimina la carga útil y todos las cabeceras *!Content-*, dejando a las demás cabeceras intactas y en su orden original.

Los objetos **`EmailMessage`** tienen los siguientes atributos de instancia:

preamble

El formato de un documento MIME permite algo de texto entre la linea en blanco después de las cabeceras y la primera cadena límite de multiparte. Normalmente, este texto nunca es visible en un lector de correo compatible con MIME porque queda fuera de la armadura MIME estándar. Sin embargo, al ver el texto sin formato del mensaje, o al ver el mensaje en un lector no compatible con MIME, este texto puede volverse visible.

El atributo *preamble* contiene este texto extra para documentos MIME. Cuando el **`Parser`** descubre texto después de las cabeceras pero antes de la primer cadena límite, asigna este texto al atributo *preamble* del mensaje. Cuando el **`Generator`** escribe la representación de texto sin formato de un mensaje MIME y encuentra que el mensaje tiene un atributo *preamble* escribirá este texto en el área entre las cabeceras y el primer límite. Véase **`email.parser`** y **`email.generator`** para conocer detalles.

Cabe señalar que si el objeto *message* no tiene preámbulo, el atributo *preamble* será igual a `None`.

epilogue

El atributo *epilogue* actúa de igual forma al atributo *preamble* con la excepción de que contiene texto que aparece entre el último límite y el fin del mensaje. Como con el **`preamble`**, si no hay texto de epílogo este atributo será `None`.

defects

El atributo *defects* contiene una lista de todos los errores encontrados al hacer el análisis sintáctico del mensaje. Vea **`email.errors`** para una descripción detallada de los posibles efectos del análisis sintáctico.

class email.message.MIMEPart(*policy=default*)

Esta clase representa una subparte de un mensaje MIME. Es idéntica a `EmailMessage`, excepto que las cabeceras *MIME-Version* son añadidas cuando se llama a `set_content()`, ya que las subpartes no necesitan su propia cabecera *MIME-Version*.

Notas al pie

Añadido originalmente en la versión 3.4 como un módulo provisional. La documentación para la clase heredada *message* se movió a email.message.Message: Representar un mensaje de correo electrónico usando la API compat32.

The `EmailMessage` class requires a policy that provides a `content_manager` attribute for content management methods like `set_content()` and `get_content()` to work. The legacy `compat32` policy does not support these methods and should not be used with **`EmailMessage`**.

`email.parser`: Parsing email messages

Código fuente: Lib/email/parser.py [https://github.com/python/cpython/tree/3.14/Lib/email/parser.py]

Se pueden construir estructuras de objetos de mensaje de dos formas: pueden ser creados de puro invento al crear un objeto **`EmailMessage`**, añadir encabezados usando la interfaz de diccionario, y añadir carga(s) usando el método **`set_content()`** y otros relacionados, o pueden ser creados al analizar una representación serializada de un mensaje de correo electrónico.

El paquete **`email`** proporciona un analizador estándar que entiende la mayoría de estructuras de documentos de correo electrónico, incluyendo documentos MIME. Le puedes pasar al analizador bytes, una cadena de caracteres o una archivo de objeto, y el analizador te retornará la instancia **`EmailMessage`** raíz de la estructura del objeto. Para mensajes simples que no sean MIME, la carga de su objeto raíz probablemente será una cadena de caracteres conteniendo el texto o el mensaje. Para mensajes MIME, el objeto raíz retornará `True` de su método **`is_multipart()`**, y las subpartes pueden ser accedidas a través de los métodos de manipulación de carga, tales como **`get_body()`**, **`iter_parts()`**, y **`walk()`**.

De hecho hay dos interfaces de analizadores disponibles para usar, la API **`Parser`** y la API progresiva **`FeedParser`**. La API **`Parser`** es más útil si tú tienes el texto del mensaje entero en memoria, o si el mensaje entero reside en un archivo en el sistema. **`FeedParser`** es más apropiado cuando estás leyendo el mensaje de un *stream* que puede ser bloqueado esperando más entrada (tal como leer un mensaje de correo electrónico de un socket). El **`FeedParser`** puede consumir y analizar el mensaje de forma progresiva, y sólo retorna el objeto raíz cuando cierras el analizador.

Tenga en cuenta que el analizador puede ser extendido en formas limitadas, y por supuesto puedes implementar tu propio analizador completamente desde cero. Toda la lógica que conecta el analizador empaquetado del paquete `email` y la clase `EmailMessage` está encarnada en la clase `Policy`, por lo que un analizador personalizado puede crear árboles de objetos mensaje en cualquier forma que encuentre necesario al implementar versiones personalizadas de los métodos apropiados de **`Policy`**.

API *FeedParser*

La clase `BytesFeedParser`, importado del módulo **`email.feedparser`**, proporciona una API que es propicia para el análisis progresivo de mensajes de correo electrónico, tal como sería necesario cuando se esté leyendo el texto de un mensaje de correo electrónico de una fuente que puede bloquear (tal como un socket). Desde luego se puede usar la clase **`BytesFeedParser`** para analizar un mensaje de correo electrónico completamente contenido en un bytes-like object, cadena de caracteres, o archivo, pero la API `BytesParser` puede ser más conveniente para tales casos de uso. Las semánticas y resultados de las dos API de los analizadores son idénticas.

La API de `BytesFeedParser` es simple; puedes crear una instancia, le proporcionas un montón de bytes hasta que no haya más necesidad de hacerlo, entonces cierras el analizador para recuperar el objeto del mensaje raíz. El **`BytesFeedParser`** es extremadamente preciso cuando está analizando mensajes conformes al estándar, y hace un buen trabajo al analizar mensajes no conformes, proporcionando información acerca de cómo un mensaje fue considerado inservible. Ingresará una lista de cualquier problema que encontró en el atributo `defects` del objeto mensaje. Véase el módulo `email.errors` para la lista de defectos que puede encontrar.

Aquí está el API para `BytesFeedParser`:

class email.parser.BytesFeedParser(*_factory=None*, *, *policy=policy.compat32*)

Crea una instancia de `BytesFeedParser`. El argumento opcional *_factory* es un invocable sin argumentos; si no se especifica, usa el `message_factory` de *policy*. Llama a *_factory* cuando sea necesario un nuevo objeto mensaje.

Si se especifica *policy*, usa las reglas que especifica para actualizar la representación del mensaje. Si *policy* no está puesta, usa la política (*policy*) `compat32`, que mantiene compatibilidad con la versión 3.2 de Python del paquete de correo electrónico y proporciona a `Message` como la fábrica por defecto. Todas las otras políticas proveen a `EmailMessage` como el *_factory* por defecto. Para más información en lo demás que *policy* controla, véase la documentación `policy`.

Nota: **La palabra clave *policy* siempre debe estar especificada**; El valor por defecto cambiará a `email.policy.default` en una versión futura de Python.

Added in version 3.2.

Distinto en la versión 3.3: Se añadió la palabra clave *policy*.

Distinto en la versión 3.6: *_factory* es por defecto la *policy* `message_factory`.

feed(*data*)

Le proporciona al analizador algunos datos más. *data* debe ser un bytes-like object conteniendo una o más líneas. Las líneas pueden ser parciales y el analizador va a juntar tales líneas parciales apropiadamente. las líneas pueden tener cualquiera de las tres terminaciones de línea comunes: retorno de cargo (*retorno de cargo*), nueva línea (*newline*), o retorno de cargo y nueva línea (pueden ser mezclados).

close()

Completa el análisis de todos los datos previamente proporcionados y retorna la raíz del objeto mensaje. No está definido lo que pasa si se llama a `feed()` después de que este método haya sido llamado.

class email.parser.FeedParser(*_factory=None*, ***, *policy=policy.compat32*)

Works like **`BytesFeedParser`** except that the input to the **`feed()`** method must be a string. This is of limited utility, since the only way for such a message to be valid is for it to contain only ASCII text or, if **`utf8`** is `True`, no binary attachments.

Distinto en la versión 3.3: Se añadió la palabra clave *policy*.

API *Parser*

The **`BytesParser`** class, imported from the **`email.parser`** module, provides an API that can be used to parse a message when the complete contents of the message are available in a bytes-like object or file. The **`email.parser`** module also provides **`Parser`** for parsing strings, and header-only parsers, **`BytesHeaderParser`** and **`HeaderParser`**, which can be used if you're only interested in the headers of the message. **`BytesHeaderParser`** and **`HeaderParser`** can be much faster in these situations, since they do not attempt to parse the message body, instead setting the payload to the raw body.

class email.parser.BytesParser(*_class=None*, ***, *policy=policy.compat32*)

Crea una instancia de **`BytesParser`**. Los argumentos *_class* y *policy* tiene el mismo significado y semántica que los argumentos *_factory* y *policy* de **`BytesFeedParser`**.

Nota: **La palabra clave *policy* siempre debe estar especificada**; El valor por defecto cambiará a **`email.policy.default`** en una versión futura de Python.

Distinto en la versión 3.3: Se eliminó el argumento *strict* que fue deprecado en 2.4. Se añadió la palabra clave *policy*.

Distinto en la versión 3.6: *_class* es por defecto la política `message_factory`.

parse(*fp*, *headersonly=False*)

Read all the data from the binary file-like object *fp*, parse the resulting bytes, and return the message object. *fp* must support both the **`readline()`** and the **`read()`** methods.

The bytes contained in *fp* must be formatted as a block of **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html] (or, if **`utf8`** is `True`, **RFC 6532** [https://datatracker.ietf.org/doc/html/rfc6532.html]) style headers and header continuation lines, optionally preceded by an envelope header. The header block is terminated either by the end of the data or by a blank line. Following the header block is the body of the message (which may contain MIME-encoded subparts, including subparts with a *Content-Transfer-Encoding* of `8bit`).

El argumento opcional *headersonly* es un flag que especifica si se debe analizar después de leer las cabeceras o no. El valor por defecto es `False`, significando que analiza el contenido entero del archivo.

parsebytes(*bytes*, *headersonly=False*)

Similar al método **`parse()`**, excepto que toma un bytes-like object en vez de un objeto similar a un archivo. Llamar a este método en un bytes-like object es equivalente a envolver a *bytes* en una instancia de **`BytesIO`** primero y llamar a **`parse()`**.

El argumento opcional *headersonly* es como el método **`parse()`**.

Added in version 3.2.

class email.parser.BytesHeaderParser(*_class=None*, **, *policy=policy.compat32*)

Exactamente como **`BytesParser`**, excepto que *headersonly* es por defecto `True`.

Added in version 3.3.

class email.parser.Parser(*_class=None*, ***, *policy=policy.compat32*)

Esta clase es paralela a **`BytesParser`**, pero trata entradas de cadenas de caracteres.

Distinto en la versión 3.3: Se eliminó el argumento *strict*. Se añadió la palabra clave *policy*.

Distinto en la versión 3.6: *_class* es por defecto la política `message_factory`.

parse(*fp*, *headersonly=False*)

Lee todos los datos del modo texto del objeto parecido a archivo *fp*, analiza el texto resultante, y retorna el objeto mensaje raíz. *fp* debe soportar tanto el método **`readline()`** y el método **`read()`** en objetos parecidos a archivos.

Además de el requisito del modo texto, este método opera como **`BytesParser.parse()`**.

parsestr(*text*, *headersonly=False*)

Similar al método **`parse()`**, excepto que toma un objeto de cadena de caracteres de un objeto similar a un archivo. Llamar a este método en una cadena de caracteres es equivalente a envolver a *text* en una instancia de **`StringIO`** primero y llamar a **`parse()`**.

El argumento opcional *headersonly* es como el método **`parse()`**.

class email.parser.HeaderParser(*_class=None*, ***, *policy=policy.compat32*)

Exactamente como **`Parser`**, excepto que *headersonly* es por defecto `True`.

Ya que crear una estructura de un objeto mensaje de una cadena de caracteres o un objeto archivo es una tarea tan común, Se proporcionaron 4 funciones como una conveniencia. Están disponibles en paquete de espacio de nombres de alto nivel **`email`**.

email.message_from_bytes(*s*, *_class=None*, ***, *policy=policy.compat32*)

Retorna una estructura del objeto mensaje de un bytes-like object. Esto es equivalente a `BytesParser().parsebytes(s)`. El argumento opcional *_class* y *policy* son interpretados como sucede con el constructor de clase **`BytesParser`**.

Added in version 3.2.

Distinto en la versión 3.3: Se eliminó el argumento *strict*. Se añadió la palabra clave *policy*.

email.message_from_binary_file(*fp*, *_class=None*, ***, *policy=policy.compat32*)

Retorna una estructura árbol del objeto mensaje de un file object binario abierto. Esto es equivalente a `BytesParser().parse(fp)`. *_class* y *policy* son interpretados como sucede con el constructor de clase **`BytesParser`**.

Added in version 3.2.

Distinto en la versión 3.3: Se eliminó el argumento *strict*. Se añadió la palabra clave *policy*.

email.message_from_string(*s*, *_class=None*, ***, *policy=policy.compat32*)

Retorna una estructura del objeto mensaje de una cadena de caracteres. Esto es equivalente a `Parser().parsestr(s)`. *_class* y *policy* son interpretados como sucede con el constructor de clase **`Parser`**.

Distinto en la versión 3.3: Se eliminó el argumento *strict*. Se añadió la palabra clave *policy*.

email.message_from_file(*fp*, *_class=None*, ***, *policy=policy.compat32*)

Retorna una estructura árbol del objeto mensaje de un file object abierto. Esto es equivalente a `Parser().parse(fp)`. *_class* y *policy* son interpretados como sucede con el constructor de clase **`Parser`**.

Distinto en la versión 3.3: Se eliminó el argumento *strict*. Se añadió la palabra clave *policy*.

Distinto en la versión 3.6: *_class* es por defecto la política `message_factory`.

Aquí está un ejemplo de cómo puedes usar **`message_from_bytes()`** en una entrada interactiva de Python:

```
>>> import email
>>> msg = email.message_from_bytes(myBytes)
```

Notas adicionales

Aquí están algunas notas sobre la semántica del análisis:

- La mayoría de los mensajes de tipo que no son *multipart* son actualizados como un solo objeto mensaje con una carga de cadena de caracteres. Estos objetos retornarán `False` para **`is_multipart()`**, y **`iter_parts()`** cederá (*yield*) una lista vacía.
- Todos los mensajes de tipo *multipart* serán analizados como un objeto mensaje contenedor con una lista de objetos submensajes para sus cargas. El mensaje del contenedor externo retornará `True` para **`is_multipart()`**, y **`iter_parts()`** cederá (*yield*) una lista de subpartes.
- La mayoría de mensajes con una tipo de contenido de *message/** (tal como *message/delivery-status* y *message/rfc822*) también serán analizados como objetos contenedores que contienen una lista de cargas de longitud 1. Su método **`is_multipart()`** retornará `True`. El único elemento cedido

(*yielded*) por `iter_parts()` será un objeto sub-mensaje.

- Algunos mensajes de conformidad no estándar pueden no ser internamente consistentes acerca de su *multipart*-idad. Tales mensajes pueden tener una cabecera *Content-Type* de tipo *multipart*, pero su método `is_multipart()` puede retornar `False`. Si tales mensajes son analizados con `FeedParser`, tendrán una instancia de la clase `MultipartInvariantViolationDefect` en su lista de atributos *defects*. Véase `email.errors` para más detalles.

[1]

`email.generator`: Generating MIME documents

Código fuente: Lib/email/generator.py [https://github.com/python/cpython/tree/3.14/Lib/email/generator.py]

One of the most common tasks is to generate the flat (serialized) version of the email message represented by a message object structure. You will need to do this if you want to send your message via `smtplib.SMTP.sendmail()`, or print the message on the console. Taking a message object structure and producing a serialized representation is the job of the generator classes.

Al igual que con el módulo `email.parser`, no se limita a la funcionalidad del generador incluido; se podría escribir uno desde cero. Sin embargo, el generador incluido sabe cómo generar la mayoría del correo electrónico de una manera compatible con los estándares, debería controlar los mensajes de correo electrónico MIME y no MIME bien. Está diseñado para que las operaciones de análisis y generación orientadas a bytes sean inversas, asumiendo que se utilice la misma `policy` de no transformación para ambos. Es decir, analizar la secuencia de bytes serializada a través de la clase `BytesParser` y, a continuación, regenerar la secuencia de bytes serializada mediante `BytesGenerator` debe producir una salida idéntica a la entrada [1]. (Por otro lado, el uso del generador en un `EmailMessage` construido por el programa puede dar lugar a cambios en el objeto `EmailMessage` a medida que se rellenan los valores predeterminados.)

La clase `Generator` se puede utilizar para acoplar un mensaje en una representación serializada de texto (a diferencia de la binaria). Sin embargo, como Unicode no puede representar datos binarios directamente, el mensaje es necesariamente transformado en algo que contiene sólo caracteres ASCII. Se utiliza las técnicas de codificación de transferencia de contenido RFC de correo electrónico estándar para codificar mensajes de correo electrónico para el transporte a través de canales que no son «8 bits limpios».

Para adaptar procesamiento reproducible de mensajes firmados por SMIME, **`Generator`** deshabilita el encabezamiento para las partes del mensaje de tipo `multipart/signed` y todas sus subpartes.

class email.generator.BytesGenerator(*outfp*, *mangle_from_=None*, *maxheaderlen=None*, ***, *policy=None*)

> Retorna un objeto **`BytesGenerator`** que escribirá cualquier mensaje provisto por el método **`flatten()`**, o cualquier texto de escape sustituto cifrado con el método **`write()`**, al file-like object *outfp*. *outfp* debe soportar un método `write` que acepte datos binarios.
>
> Si *mangle_from_* opcional es `True`, se coloca un carácter “”>”” delante de cualquier línea del cuerpo que comience con la cadena exacta `"From "`, es decir, `From` seguido de un espacio al principio de una línea. *mangle_from_* vuelve de forma predeterminada al valor de la configuración **`mangle_from_`** de la *policy* (que es `True` para la política **`compat32`** y `False` para todas las demás). *mangle_from_* está diseñado para su uso cuando los mensajes se almacenan en formato mbox de Unix (consulte **`mailbox`** y WHY THE CONTENT-LENGTH FORMAT IS BAD [https://www.jwz.org/doc/content-length.html]).
>
> Si *maxheaderlen* no es `None`, se repliega las líneas de encabezado que son mayores que *maxheaderlen*, o si es `0`, no se reenvuelve ningún encabezado. Si *manheaderlen* es `None` (predeterminado), se envuelven los encabezados y otras líneas de mensajes de acuerdo a los ajustes de *policy*.
>
> Si la *norma* es especificada, se usa esa norma para controlar la generación de mensajes. Si la *norma* es `None` (predeterminado), se usa la norma asociada con el **`Message`** o el objeto **`EmailMessage`** pasado para `flatten` para controlar la generación del mensaje. Se puede ver **`email.policy`** para detalles de que controla la *norma*.

Added in version 3.2.

Distinto en la versión 3.3: Agregada la palabra clave *norma*.

Distinto en la versión 3.6: El comportamiento predeterminado de los parámetros *mangle_from_* y *maxheaderlen* es para seguir la norma.

flatten(*msg*, *unixfrom=False*, *linesep=None*)

Imprime la representación textual de la estructura del objeto de mensaje originada en *msg* al archivo de salida especificado cuando se creó la instancia **`BytesGenerator`**.

Si el tipo **`cte_type`** de la opción **`policy`** es `8bit` (valor predeterminado), se copia los encabezados en el mensaje analizado original que no se hayan modificado con ningún bytes a la salida con el conjunto de bits altos, reproducido como en el original, y se conserva el *Content-Transfer-Encoding* no ASCII de cualquier parte del cuerpo que los tenga. Si `cte_type` es `7bit`, se convierte los bytes con el conjunto de bits altos según sea necesario utilizando un *Content-Transfer-Encoding* compatible con ASCII . Es decir, se transforma partes con *Content-Transfer-Encoding* no ASCII (*Content-Transfer-Encoding: 8bit*) en un conjunto de caracteres compatible con ASCII *Content-Transfer-Encoding*, y se codifica bytes inválidos RFC no ASCII en encabezados mediante el conjunto de caracteres MIME `unknown-8bit`, lo que los convierte en compatibles con RFC.

Si *unixfrom* es `True`, se imprime el delimitador de encabezado de sobre utilizado por el formato de buzón de correo Unix (consulta **`mailbox`**) antes del primero de los encabezados **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html] del objeto de mensaje raíz. Si el objeto raíz no tiene encabezado de sobre, se crea uno estándar. El valor predeterminado es `False`. Tener en cuenta que para las subpartes, nunca se imprime ningún encabezado de sobre.

Si *linesep* no es `None`, se usa como caracter separador entre todas las líneas del mensaje acoplado. Si *linesep* es `None` (predeterminado), se usa el valor especificado en la *norma*.

clone(*fp*)

Retorna un clon independiente de esta instancia de **`BytesGenerator`** con las mismas configuraciones exactas, y *fp* como el nuevo *outfp*.

write(*s*)

Codifica *s* usando el códec `ASCII` u el manipulador de error `escape sustituto`, y lo pasa al método *write* del *outfp* pasado al constructor de la clase **`BytesGenerator`**.

Como conveniencia, **`EmailMessage`** provee los métodos **`as_bytes()`** y `bytes(aMessage)` (también conocido como **`__bytes__()`**) que simplifican la generación de la representación serializada binaria de un objeto mensaje. Para más detalle, ver **`email.message`**.

Dado que las cadenas de caracteres no pueden representar datos binarios, la clase **`Generator`** debe convertir los datos binarios en cualquier mensaje que aplane a un formato compatible con ASCII, convirtiéndolos en un *Content-Transfer_Encoding* compatible con ASCII. Usando la terminología de RFC de correo electrónico, se puede pensar en esto como **`Generator`** serializando a una secuencia de E/S que no es *«8 bit clean»*. En otras palabras, la mayoría de las aplicaciones querrán usar **`BytesGenerator`**, y no **`Generator`**.

class email.generator.Generator(*outfp*, *mangle_from_=None*, *maxheaderlen=None*, ***, *policy=None*)

Retorna un objeto **`Generator`** que escribirá cualquier mensaje provisto al método **`flatten()`**, o cualquier texto provisto al método **`write()`**, al file-like object *outfp*. *outfp* debe soportar un método `write` que acepte datos de cadena de caracteres.

Si *mangle_from_* opcional es `True`, se coloca un carácter “”>”” delante de cualquier línea del cuerpo que comience con la cadena exacta `"From "`, es decir, `From` seguido de un espacio al principio de una línea. *mangle_from_* vuelve de forma predeterminada al valor de la configuración **`mangle_from_`** de la *policy* (que es `True` para la política **`compat32`** y `False` para todas las demás). *mangle_from_* está diseñado para su uso cuando los mensajes se almacenan en formato mbox de Unix (consulte **`mailbox`** y WHY THE CONTENT-LENGTH FORMAT IS BAD [https://www.jwz.org/doc/content-length.html]).

Si *maxheaderlen* no es `None`, se repliega las líneas de encabezado que son mayores que *maxheaderlen*, o si es `0`, no se reenvuelve ningún encabezado. Si *manheaderlen* es `None` (predeterminado), se envuelven los encabezados y otras líneas de mensajes de acuerdo a los ajustes de *policy*.

Si la *norma* es especificada, se usa esa norma para controlar la generación de mensajes. Si la *norma* es `None` (predeterminado), se usa la norma asociada con el **`Message`** o el objeto **`EmailMessage`** pasado para `flatten` para controlar la generación del mensaje. Se puede ver **`email.policy`** para detalles de que controla la *norma*.

Distinto en la versión 3.3: Agregada la palabra clave *norma*.

Distinto en la versión 3.6: El comportamiento predeterminado de los parámetros *mangle_from_* y *maxheaderlen* es para seguir la norma.

flatten(*msg*, *unixfrom=False*, *linesep=None*)

Imprime la representación textual de la estructura del objeto de mensaje originado en el *msg* al archivo especificado de salida cuando la instancia de **`Generator`** fue creada.

Si la opción **`policy`** **`cte_type`** es `8bit`, se genera el mensaje como si la opción estuviera establecida en `7bit`. (Esto es necesario porque las cadenas de caracteres no pueden representar bytes que no sean ASCII). Convierte cualesquiera bytes con el conjunto de bits alto según sea necesario utilizando un *Content-Transfer-Encoding* compatible con ASCII. Es decir, transforma partes con *Content-Transfer-Encoding* (*Content-Transfer-Encoding: 8bit*) no ASCII en un conjunto de caracteres *Content-Transfer-Encoding* compatible con ASCII. Y codifica bytes no ASCII RFC inválidos ASCII en encabezados mediante el conjunto de caracteres MIME `unknown-8bit`, lo que los convierte en compatibles con RFC.

Si *unixfrom* es `True`, se imprime el delimitador de encabezado de sobre utilizado por el formato de buzón de correo Unix (consulta **`mailbox`**) antes del primero de los encabezados **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html] del objeto de mensaje raíz. Si el objeto raíz no tiene encabezado de sobre, se crea uno estándar. El valor predeterminado es `False`. Tener en cuenta que para las subpartes, nunca se imprime ningún encabezado de sobre.

Si *linesep* no es `None`, se usa como caracter separador entre todas las líneas del mensaje acoplado. Si *linesep* es `None` (predeterminado), se usa el valor especificado en la *norma*.

Distinto en la versión 3.2: Agrega soporte para el recodificado de cuerpos de mensajes `8bit`, y el argumento *linesep*.

clone(*fp*)

Retorna un clon independiente de esta instancia de **`Generator`** con las mismas opciones exactas y *fp* como la nueva *outfp*.

write(*s*)

Escribe *s* al método *write* del *outfp* pasado al constructor de la `Generator`. Esto provee justo la suficiente API de tipo archivo para instancias de **`Generator`** para ser usadas en la función `print()`.

Para conveniencia, `EmailMessage` provee los métodos `as_string()` y `str(aMessage)` (también conocido como `__str__()`), que simplifican la generación de una representación de una cadena de caracteres formateada de un objeto mensaje. Para más detalles, ver `email.message`.

The **`email.generator`** module also provides a derived class, `DecodedGenerator`, which is like the `Generator` base class, except that non-*text* parts are not serialized, but are instead represented in the output stream by a string derived from a template filled in with information about the part.

class email.generator.DecodedGenerator(*outfp*, *mangle_from_=None*, *maxheaderlen=None*, *fmt=None*, ***, *policy=None*)

Se actúa como `Generator`, excepto para cualquier subparte del mensaje pasado a `Generator.flatten()`, si la subparte es de tipo principal *text*, se imprime la carga útil decodificada de la subparte, y si el tipo principal no es *text*, en lugar de imprimirlo se rellena la cadena de caracteres *fmt* utilizando la información de la parte y se imprime la cadena de caracteres rellenada resultante.

Para llenar *fmt*, se ejecuta `fmt % part_info`, donde `part_info` es un diccionario compuesto por las siguientes claves y valores:

- `type` – Todo el tipo MIME de la parte no*text*
- `maintype` – Principal tipo MIME de la parte no*text*
- `subtype` – Tipo sub-MIME de la parte no-*text*
- `filename` – Nombre de archivo de la parte no*text*
- `description` – Descripción asociada con la parte no*text*
- `encoding` – Codificado de la transferencia del contenido de la parte no-*text*

Si *fmt* es `None`, se usa el siguiente *fmt* predeterminado:

> «[Non-text (%(type)s) part of message omitted, filename %(filename)s]»

Los opcionales *_mangle_from_* y *maxheaderlen* son como en la clase base **`Generator`**.

Notas al pie

Esta instrucción supone que se utiliza la configuración adecuada para `unixfrom`, y que no hay ninguna configuración **`email.policy`** que llame a ajustes automáticos (por ejemplo, **`refold_source`** debe ser `none`, que es *no* es el valor predeterminado). Esto tampoco es 100% verdadero, ya que si el mensaje no se ajusta a los estándares RFC ocasionalmente la información sobre el texto original exacto se pierde durante la el análisis de recuperación de errores. Es un objetivo fijar estos últimos casos extremos cuando sea posible.

[1]

`email.policy`: Policy Objects

Added in version 3.3.

Código fuente: Lib/email/policy.py [https://github.com/python/cpython/tree/3.14/Lib/email/policy.py]

El enfoque principal del paquete `email` es la gestión de mensajes de correo electrónico como se describe por los varios RFC de correos electrónicos y MIME. Sin embargo, el formato general de los mensajes de correo electrónico (un bloque de campos de cabecera cada uno consistiendo en un nombre seguido de dos puntos seguido de un valor, el bloque entero seguido por una línea blanca y un "cuerpo" arbitrario), es un formato que ha encontrado utilidad fuera del campo de correos electrónicos. Algunos de estos usos cumplen bastante cerca los RFC de correos electrónicos principales, alguno no. Incluso cuando se trabajo con correos electrónicos, hay veces cuando es deseable romper la estricta conformidad con los RFC, tal como generar correos electrónicos que se integran con servidores de correos electrónicos que no siguen los estándares, o que implementan extensiones que quieres usar en formas que violan los estándares.

Los objetos *policy* dan al paquete de correos electrónicos la flexibilidad de manejar todos estos casos de uso diversos.

Un objeto `Policy` encapsula un conjunto de atributos y métodos que controlan el comportamiento de varios componentes del paquete de correos electrónicos durante el uso. Las instancias `Policy` pueden ser pasadas a varias clases y métodos en el paquete de correos electrónicos para alterar el comportamiento por defecto. Los valores que se pueden configurar y sus valores por defecto se describen abajo.

Hay una *policy* por defecto usada por todas las clases en el paquete de correos electrónicos. Para todos las clases `Parser` y las funciones de conveniencia relacionadas, y para la clase `Message`, esta es una *policy* `Compat32`, a través de sus correspondientes instancias `compat32` pre-definidas. Esta política mantiene una compatibilidad (en algunos casos, compatibilidad con los bugs) con el paquete de correos electrónicos de las versiones anteriores de Python3.3.

El valor por defecto para la palabra clave *policy* de `EmailMessage` es la *policy* `EmailPolicy`, a través de su instancia pre-definida `default`.

Cuando un objeto `Message` o `EmailMessage` es creado, adquiere un *policy*. Si el mensaje es creado por un `parser`, un *policy* pasado al analizador será el *policy* usado por el mensaje que cree. Si el mensaje es creado por el programa, entonces el *policy* puede ser especificado cuando sea creado. Cuando un mensaje es pasado a un `generator`, el generador usa el *policy* del mensaje por defecto, pero también puedes pasar un *policy* específico al generador que anulará el que está guardado en el objeto mensaje.

El valor por defecto del argumento por palabra clave *policy* para las clases `email.parser` y las funciones de conveniencia del analizador **se cambiarán** en una futura versión de Python. Por consiguiente, **siempre debes especificar explícitamente qué policy quieres usar** cuando se llama a cualquiera de las clases y funciones descritas en el módulo `parser`.

La primera parte de esta documentación cubre las características de `Policy`, un abstract base class que define las características que son comunes a todos los objetos *policy*, incluyendo `compat32`. Esto incluye ciertos métodos gancho (*hook*) que son llamados internamente por el paquete de correos electrónicos, que un *policy* personalizado puede invalidar para obtener un comportamiento diferente. La segunda parte describe las clases concretas `EmailPolicy` y `Compat32`, que implementan los ganchos (*hooks*) que proporcionan el comportamiento estándar y el comportamiento y características compatibles hacia atrás, respectivamente.

Las instancias `Policy` son inmutables, pero pueden ser clonadas, aceptando los mismos argumentos de palabra clave que el constructor de clase y retorna una nueva instancia `Policy` que es una copia del original pero con los valores de atributos específicos cambiados.

Como un ejemplo, el siguiente código puede ser usado para leer un mensaje de correo electrónico de un archivo en disco y pasarlo al programa de sistema `sendmail` en un sistema Unix:

```
>>> from email import message_from_binary_file
>>> from email.generator import BytesGenerator
>>> from email import policy
>>> from subprocess import Popen, PIPE
>>> with open('mymsg.txt', 'rb') as f:
...     msg = message_from_binary_file(f, policy=policy.
...
>>> p = Popen(['sendmail', msg['To'].addresses[0]], stdi
>>> g = BytesGenerator(p.stdin, policy=msg.policy.clone(
>>> g.flatten(msg)
>>> p.stdin.close()
>>> rc = p.wait()
```

Aquí le estamos diciendo a **`BytesGenerator`** que use los caracteres de separación de línea correctos de RFC cuando crea la cadena binaria para alimentar a `stding` de `sendmail` (`sendmails's stdin`), donde el *policy* por defecto usaría separadores de línea `\n`.

Algunos métodos del paquete de correos electrónicos aceptan un argumento de palabra clave *policy*, permitiendo que el *policy* pueda ser anulado para ese método. Por ejemplo, el siguiente código usa el método **`as_bytes()`** del objeto *msg* del ejemplo anterior y escribe el mensaje en un archivo usando los separadores de línea nativos para la plataforma en el que esté corriendo:

```
>>> import os
>>> with open('converted.txt', 'wb') as f:
...     f.write(msg.as_bytes(policy=msg.policy.clone(lin
17
```

Los objetos *policy* puede ser combinados usando el operador de adición, produciendo un objeto *policy* cuya configuración es una combinación de los valores que de los objetos sumados:

```
>>> compat_SMTP = policy.compat32.clone(linesep='\r\n')
>>> compat_strict = policy.compat32.clone(raise_on_defec
>>> compat_strict_SMTP = compat_SMTP + compat_strict
```

Esta operación no es conmutativa; es decir, el orden en el que los objetos son añadidos importa. Para ilustrar:

```
>>> policy100 = policy.compat32.clone(max_line_length=10
>>> policy80 = policy.compat32.clone(max_line_length=80)
>>> apolicy = policy100 + policy80
>>> apolicy.max_line_length
80
>>> apolicy = policy80 + policy100
>>> apolicy.max_line_length
100
```

class email.policy.Policy(***kw*)

Este es el abstract base class para todas las clases *policy*. Proporciona las implementaciones por defecto para un par de métodos triviales, también como la implementación de las propiedades de inmutabilidad, el método **`clone()`**, y las semánticas del constructor.

Se pueden pasar varios argumentos de palabra clave al constructor de una clase *policy*. Los argumentos que pueden ser especificados son cualquier propiedad que no sea método en esta clase, además de cualquier otra propiedad adicional que no sea método en la clase concreta. Un valor especificado en el constructor anulará el valor por defecto para los atributos correspondientes.

Esta clase define las siguientes propiedades, y por consiguiente los valores a continuación pueden ser pasados al constructor de cualquier clase *policy*:

max_line_length

El tamaño máximo de cualquier línea en la salida serializada, sin contar el fin de la línea de carácter(res).

Por defecto es 78, por el **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html]. Un valor de `0` o **`None`** indica que ningún envolvimiento de líneas puede ser hecha en lo más mínimo.

linesep

La cadena de caracteres a ser usada para terminar las líneas en una salida serializada. El valor por defecto es `\n` porque esa es la disciplina del fin de línea interna usada por Python, aunque `\r\n` es requerida por los RFCs.

cte_type

Controla el tipo de Codificaciones de Transferencia de Contenido que pueden ser o son necesarios para ser usados. Los valores posibles son:

`7bit`
todos los datos deben ser «compatibles con 7 bit (*7 bit clean*)» (ASCII-only). Esto significa que donde sea necesario, los datos serán codificados usando o codificación imprimible entre comillas o base64.

`8bit`
los datos no son limitados a ser compatibles con 7 bits (*7 bit clean*). Se requiere que los datos en las cabeceras todavía sean sólo ASCII y por lo tanto estarán codificadas (véase **`fold_binary()`** y **`utf8`** abajo por las excepciones), pero las partes del cuerpo pueden usar `8bit` CTE.

Un valor `cte_type` de `` `8bit `` sólo trabaja con `BytesGenerator`, no `Generator`, porque las cadenas no pueden contener datos binarios. Si un `Generator` está operando bajo un *policy* que especifica `cte_type=8bit`, actuará como si `cte_type` fuese `7bit`.

raise_on_defect

Si es **`True`**, cualquier defecto encontrado será lanzado como error. Si es **`False`** (el valor por defecto), los defectos serán pasados al método `register_defect()`.

mangle_from_

Si es `True`, las líneas empezando con *«From «* en el cuerpo son saltados al poner un > en frente de ellos. Este parámetro es usado cuando el mensaje está siendo serializado por un generador. El valor por defecto es: `False`.

Added in version 3.5.

message_factory

Una función de fábrica para construir un nuevo objeto mensaje vacío. Usado por el analizador cuando construye mensajes. Por defecto es `None`, en cuyo caso `Message` es usado.

Added in version 3.6.

verify_generated_headers

If `True` (the default), the generator will raise `HeaderWriteError` instead of writing a header that is improperly folded or delimited, such that it would be parsed as multiple headers or joined with adjacent data. Such headers can be generated by custom header classes or bugs in the `email` module.

As it's a security feature, this defaults to `True` even in the `Compat32` policy. For backwards compatible, but unsafe, behavior, it must be set to `False` explicitly.

Added in version 3.13.

El siguiente método de `Policy` está destinado a ser llamado por código usando la librería de correos electrónicos para crear instancias de *policy* con configuraciones personalizadas:

clone(***kw*)

Retorna una nueva instancia de `Policy` cuyos atributos tienen los mismos valores que la instancia actual, excepto donde se le dan, a esos atributos, nuevos valores por los argumentos de palabra clave.

Los métodos de **`Policy`** restantes son llamados por el código de paquete de correos electrónicos, y no están destinados a ser llamados por una aplicación usando el paquete de correos electrónicos. Un *policy* personalizado debe implementar todos estos métodos.

handle_defect(*obj*, *defect*)

Handle a *defect* found on *obj*. When the email package calls this method, *defect* will always be a subclass of **`MessageDefect`**.

La implementación por defecto verifica la bandera **`raise_on_defect`**. Si es `True`, *defect* es lanzado como una excepción. Si es `False` (el valor por defecto), *obj* y *defect* son pasados a **`register_defect()`**.

register_defect(*obj*, *defect*)

Register a *defect* on *obj*. In the email package, *defect* will always be a subclass of **`MessageDefect`**.

La implementación por defecto llama al método `append` del atributo `defects` de *obj*. Cuando un paquete de correos electrónicos llama a **`handle_defect`**, *obj* normalmente tendrá un atributo `defects` que tiene un método `append`. Los tipos de objetos personalizados usados con el paquete de correos electrónicos (por ejemplo, objetos `Message` personalizados) también deben proporcionar tal atributo, de otro modo, los defectos en los mensajes analizados levantarán errores no esperados.

header_max_count(*name*)

Retorna el máximo número permitido de cabeceras llamadas *name*.

Llamado cuando una cabecera es añadida a un objeto **`EmailMessage`** o **`Message`**. Si el valor retornado no es `0` o `None`, y ya hay un número de cabeceras con el nombre *name* mayor o igual que el valor retornado, un **`ValueError`** es lanzado.

Porque el comportamiento por defecto de `Message.__setitem__` es agregar el valor a la lista de cabeceras, es fácil crear cabeceras duplicadas sin darse cuenta. Este método permite que ciertas cabeceras sean limitadas en el número de instancias de esa cabecera que puede ser agregada a `Message` programáticamente. (El límite no es observado por el analizador, que fielmente producirá tantas cabeceras como existen en el mensaje siendo analizado.)

La implementación por defecto retorna `None` para todos los nombres de cabeceras.

header_source_parse(*sourcelines*)

El paquete de correos electrónicos llama a este método con una lista de cadenas de caracteres, cada terminación de cadena con los caracteres de separación de línea encontrada en la fuente siendo analizada. La primera línea incluye el campo del nombre de cabecera y el separador. Todos los espacios en blanco en la fuente son preservados. El método debe retornar la tupla `(name,value)` que va a ser guardada en el `Message` para representar la cabecera analizada.

Si una implementación desea mantener compatibilidad con los *policy* del paquete de correos electrónicos existentes, *name* debe ser el nombre con las letras preservadas (todos los caracteres hasta el separador "`:`"), mientras que *value* debe ser el valor desdoblado (todos los caracteres de separación de líneas eliminados, pero los espacios en blanco intactos), privado de espacios en blanco al comienzo.

sourcelines puede contener datos binarios *surrogateescaped.*

No hay implementación por defecto

header_store_parse(*name*, *value*)

El paquete de correos electrónicos llama a este método con el nombre y valor proporcionados por el programa de aplicación cuando la aplicación está modificando un `Message` programáticamente (en lugar de un `Message` creado por un analizador). El método debe retornar la tupla `(name,value)` que va a ser almacenada en el `Message` para representar la cabecera.

Si una implementación desea retener compatibilidad con los *policy* del paquete de correos electrónicos existentes, el *name* y *value* deben ser cadenas de caracteres o subclases de cadenas que no cambien el contenido de los pasados en los argumentos.

No hay implementación por defecto

header_fetch_parse(*name*, *value*)

El paquete de correos electrónicos llama a este método con el *name* y *valor* actualmente guardados en `Message` (el mensaje) cuando la cabecera es solicitada por el programa de aplicación, y lo que sea que el método retorne es lo que es pasado de vuelta a la aplicación como el valor de la cabecera siendo recuperado. Note que puede haber más de una cabecera con el mismo nombre guardado en `Message`; el método es pasado al nombre específico y valor de la cabecera destinado a ser retornado a la aplicación.

value puede contener datos binarios *surrogateescaped.* No debe haber datos binarios *surrogateescaped* en el valor retornado por el método.

No hay implementación por defecto

fold(*name*, *value*)

El paquete de correos electrónicos llama a este método con los *name* y *value* actualmente guardados en `Message` para una cabecera dada. El método debe retornar una cadena de caracteres que represente esa cabecera «doblada» correctamente (de acuerdo a los ajustes del *policy*) al componer *name* con *value* e insertar caracteres `linesep` en los lugares apropiados. Véase **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html] para una discusión de las reglas para doblar cabeceras de correos electrónicos.

value puede contener datos binarios *surrogateescaped*. No debe haber datos binarios *surrogateescaped* en la cadena de caracteres retornada por el método.

fold_binary(*name*, *value*)

Igual que `fold()`, excepto que el valor retornado debe ser un objeto bytes en vez de una cadena de caracteres.

value puede contener datos binarios *surrogateescaped*. Estos pueden ser convertidos de vuelta a datos binarios en el objeto de bytes retornado.

class email.policy.EmailPolicy(***kw*)

Este `Policy` concreto proporciona el comportamiento que sirve para cumplir con los RFCs actuales para correos electrónicos. Estos incluyen (pero no están limitados a) **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html], **RFC 2047** [https://datatracker.ietf.org/doc/html/rfc2047.html], y los actuales RFCs MIME.

Esta *policy* incorpora nuevos analizadores de cabeceras y algoritmos de doblado. En vez de cadenas de caracteres simples, las cabeceras son subclases de `str` con atributos que dependen del tipo del campo. El analizado y algoritmo de doblado implementan los **RFC 2047** [https://datatracker.ietf.org/doc/html/rfc2047.html] y **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html] por completo.

El valor por defecto para el atributo **`message_factory`** es **`EmailMessage`**.

Además de los atributos que se pueden configurar listados arriba que aplican a todas los *policies*, este *policy* añade los siguientes atributos adicionales:

Added in version 3.6: [1]

utf8

Si es `False`, se sigue el **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html], siendo compatible con caracteres non-ASCII en las cabeceras al codificarlas como «palabras codificadas». Si es `True`, sigue el **RFC 6532** [https://datatracker.ietf.org/doc/html/rfc6532.html] y usa el formato de codificación `utf-8` para las cabeceras. Los mensajes formateados de esta manera puede ser pasados a servidores SMTP que admitan la extensión `SMTPUTF8` (**RFC 6531** [https://datatracker.ietf.org/doc/html/rfc6531.html]).

refold_source

Si el valor para una cabecera en el objeto `Message` se originó de un **`parser`** (en lugar de ser establecido por el programa), este atributo indica tanto si un generador debe redoblar ese valor cuando se transforma al mensaje de vuelta a la forma serializada o no. Los valores posibles son:

`none`	todos los valores de la fuente usan el doblamiento original
`long`	los valores de la fuente que tengan una línea que sea más grande que `max_line_length` serán redoblados
`all`	todos los valores son redoblados.

El valor por defecto es `long`.

header_factory

Un invocable que toma dos argumentos, `name` y `value`, donde `name` es un nombre de campo de cabecera y `value` es un valor del campo de la cabecera no doblado, y retorna una subclase de cadenas de caracteres que representan la cabecera. Un valor por defecto `header_factory` (véase **`headerregistry`**) es proporcionado que admite el análisis personalizado para los varios tipos de cabecera RFC 5322 [https://datatracker.ietf.org/doc/html/rfc5322.html] de direcciones y fechas, y los tipos del cabeceras MIME principales. La compatibilidad de analizadores personalizados adicionales será agregada en el futuro.

content_manager

Un objeto con al menos dos métodos: get_content and set_content. Cuando los métodos **`get_content()`** o **`set_content()`** del objeto **`messageEmailMessage`** son llamados, llama al método correspondiente de este objeto, pasándole el mensaje objeto como su primer argumento, y cualquier argumento o palabra clave que fuese pasado como un argumento adicional. Por defecto `content_manager` se pone a **`raw_data_manager`**.

Added in version 3.4.

La clase proporciona las siguientes implementaciones concretas de los métodos abstractos de **`Policy`**:

header_max_count(*name*)

Retorna el valor del atributo **`max_count`** de la clase especializada usada para representar la cabecera con el nombre dado.

header_source_parse(*sourcelines*)

El nombre es analizado como todo hasta el "`:`" y retornado inalterado. El valor es determinado al remover los espacios en blanco al principio del resto de la primera línea, juntando todas las subsecuentes líneas, y removiendo cualquier carácter CR o LF.

header_store_parse(*name, value*)

El nombre es retornado inalterado. Si el valor de la entrada tiene un atributo `name` y coincide con *name* ignorando mayúsculas y minúsculas, el valor es retornado inalterado. Si no, el *name* y *value* son pasados a `header_factory` , y el objeto cabecera resultante es retornado como el valor. En este caso un `ValueError` es lanzado si el valor de la entrada contiene caracteres CR o LF.

header_fetch_parse(*name, value*)

Si el valor tiene un atributo `name`, es retornado inalterado. De otra manera el *name*, y el *value* con cualquier carácter CR o LF eliminado, son pasados a la `header_factory`, y el objeto cabecera resultante es retornado. Cualquier byte *surrogateescaped* es convertido al glifo de carácter desconocido de unicode.

fold(*name, value*)

El doblamiento de la cabecera es controlado por la configuración de *policy* **`refold_source`**. Se considera que un valor es *source value* (valor fuente) si y sólo si no tiene un atributo `name` (tener un atributo `name` significa que es un objeto cabecera de algún tipo). Si un valor fuente necesita ser redoblado de acuerdo a la política, es convertido en un objeto cabecera al pasarle el *name* y *value* con cualquier carácter CR y LF eliminado al `header_factory`. El doblamiento de un objeto cabecera es hecho al llamar a su método `fold` con el *policy* actual.

Los valores fuente son separadas en líneas usando el método **`splitlines()`**. Si el valor no va a ser redoblado, las líneas son unidas de nuevo usando el `linesep` del *policy* y retornadas. La excepción son las líneas que contienen datos binarios non-ascii. En ese caso el valor es redoblado a pesar de la configuración de `refold_source`, que causa que los datos binarios sean codificados CTE usando el juego de caracteres (charset) `unknown-8bit`.

fold_binary(*name*, *value*)

Igual que **`fold()`** si **`cte_type`** fuese `7bit`, excepto que el valor retornado son bytes.

Si **`cte_type`** es `8bit`, los datos binarios non-ASCII son convertidos de vuelta a bytes. Las cabeceras con datos binarios son redoblados, sin considerar la configuración de `refold_header`, ya que no hay forma de saber si los datos binarios consisten en caracteres de un sólo byte o caracteres con múltiples bytes.

Las siguientes instancias de **`EmailPolicy`** proporcionan valores por defecto apropiados para dominios de aplicaciones específicas. Note que en el futuro el comportamiento de estas instancias (en particular la instancia `HTTP`) puede ser ajustado para cumplir incluso más de cerca a los RFC relevantes a sus dominios.

email.policy.default

Una instancia de `EmailPolicy` con todos los valores por defecto sin cambiar. Este *policy* usa la terminación de línea `\n` estándar de Python en vez de correcto por el RFC `\r\n`.

email.policy.SMTP

Apropiado para la serialización de mensajes en cumplimiento con los RFC de correos electrónicos. Como `default`, pero con `linesep` puesto en `\r\n`, que es conforme con el RFC.

email.policy.SMTPUTF8

Igual que `SMTP` excepto que **`utf8`** es `True`. Útil para serializar mensajes a un almacén de mensajes sin usar palabras codificadas en las cabeceras. Sólo debe ser utilizada para transmisiones por SMTP si las direcciones del remitente o recipiente tienen caracteres non-ASCII (el método **`smtplib.SMTP.send_message()`** gestiona esto automáticamente).

email.policy.HTTP

Apropiado para serializar cabeceras con uso en tráfico de HTTP. Como `SMTP` excepto que `max_line_length` es puesto en `None` (ilimitado).

email.policy.strict

Instancias de conveniencia. Igual que `default` excepto que `raise_of_defect` es puesto en `True`. Esto permite que cualquier *policy* sea hecho estricto al escribir:

```
somepolicy + policy.strict
```

Con todos estos **`EmailPolicies`**, el API efectivo del paquete de correos electrónicos es cambiado del API de Python 3.2 en las siguientes maneras:

- Estableciendo una cabecera en una **`Message`** resulta en que la cabecera sea analizada y un objeto cabecera sea creado.
- Buscar una valor de cabecera de un **`Message`** resulta en que la cabecera sea analizada y un objeto cabecera sea creado y retornado.
- Cualquier objeto cabecera, o cualquier cabecera que sea redoblada debido a las configuraciones del *policy*, es doblado usando un algoritmo que implementa el algoritmo de doblado del RFC por completo, incluyendo saber dónde las palabras codificadas son requeridas y permitidas.

Desde la vista de la aplicación, significa que cualquier cabecera obtenida a través de **`EmailMessage`** es un objeto cabecera con atributos extra, cuyo valor de cadena es el valor Unicode de la cabecera completamente decodificada. Asimismo, se le puede asignar un nuevo valor a una cabecera, o a una nueva cabecera creada, usando una cadena de caracteres Unicode, y el *policy* se ocupará de convertir la cadena Unicode en la forma decodificada correcta según el RFC.

Los objetos cabecera y sus atributos son descritos en **`headerrregistry`**.

class email.policy.Compat32(***kw*)

This concrete `Policy` is the backward compatibility policy. It replicates the behavior of the email package in Python 3.2. The **`policy`** module also defines an instance of this class, `compat32`, that is used as the default policy. Thus the default behavior of the email package is to maintain compatibility with Python 3.2.

Los siguientes atributos tienen valores que son diferentes del `Policy` por defecto:

mangle_from_

El valor por defecto es `True`.

La clase proporciona las siguientes implementaciones concretas de los métodos abstractos de `Policy`:

header_source_parse(*sourcelines*)

El nombre es analizado como todo hasta el "`:`" y retornado inalterado. El valor es determinado al remover los espacios en blanco al principio del resto de la primera línea, juntando todas las subsecuentes líneas, y removiendo cualquier carácter CR o LF.

header_store_parse(*name*, *value*)

El nombre y valor son retornados inalterados.

header_fetch_parse(*name*, *value*)

Si el valor contiene datos binarios, es convertido a un objeto `Header` usando el juego de caracteres (*charset*) `unknown-8bit`. De otro modo, es retornado sin modificar.

fold(*name*, *value*)

Las cabeceras son dobladas usando el algoritmo de doblado de `Header`, lo que preserva los saltos de líneas existentes en el valor, y envuelve cada línea resultante hasta el `max_line_length`. Los datos binarios Non-ASCII son codificados por *CTE* usando el juego de caracteres (*charset*) `unknown-8bit`.

fold_binary(*name*, *value*)

Las cabeceras son dobladas usando el algoritmo de doblado **`Header`**, lo que preserva los saltos de línea existentes en el valor, y envuelve cada línea resultante hasta `max_line_length`. Si `cte_type` es `7bit`, los datos binarios non-ascii son codificados por *CTE* usando el juego de caracteres `unknown-8bit`. De otro modo, la cabecera fuente original es usada, con sus saltos de línea existentes y cualquier (inválido según el RFC) dato binario que puede contener.

email.policy.compat32

Una instancia de **`Compat32`**, proporcionando compatibilidad hacia atrás con el comportamiento del paquete de correos electrónicos en Python 3.2.

Nota

The **`compat32`** policy should not be used as a policy for **`EmailMessage`** objects, and should only be used to serialize messages that were created using the **`compat32`** policy.

Notas al pie de página

Se añadió originalmente en 3.3 como un característica provisional.

`email.errors`: Exception and Defect classes

Código fuente: Lib/email/errors.py [https://github.com/python/cpython/tree/3.14/Lib/email/errors.py]

The following exception classes are defined in the `email.errors` module:

exception email.errors.MessageError

Esta es la clase base para todas las excepciones que el paquete `email` puede lanzar. Se deriva de la clase estándar `Exception` y no define métodos adicionales.

exception email.errors.MessageParseError

Esta es la clase base para las excepciones generadas por la clase `Parser`. Se deriva de `MessageError`. Esta clase también es utilizada internamente por el analizador sintáctico utilizado por `headerregistry`.

exception email.errors.HeaderParseError

Lanzada en algunas condiciones de error al analizar los encabezados RFC 5322 [https://datatracker.ietf.org/doc/html/rfc5322.html] de un mensaje, esta clase se deriva de `MessageParseError`. El método `set_boundary()` lanzará este error si el tipo de contenido es desconocido cuando se llama al método. `Header` puede lanzar este error para ciertos errores de decodificación de base64, y cuando se intenta crear un encabezado que parece contener un encabezado incrustado (es decir, hay lo que se supone que es un línea de continuación que no tiene espacios en blanco iniciales y parece un encabezado).

exception email.errors.BoundaryError

Deprecada y no utilizada actualmente.

exception email.errors.MultipartConversionError

Raised if the `attach()` method is called on an instance of a class derived from `MIMENonMultipart` (e.g. `MIMEImage`). **`MultipartConversionError`** multiply inherits from `MessageError` and the built-in `TypeError`.

exception email.errors.HeaderWriteError

Raised when an error occurs when the `generator` outputs headers.

exception email.errors.MessageDefect

This is the base class for all defects found when parsing email messages. It is derived from `ValueError`.

exception email.errors.HeaderDefect

This is the base class for all defects found when parsing email headers. It is derived from `MessageDefect`.

Aquí está la lista de defectos que `FeedParser` puede encontrar mientras analiza mensajes. Tenga en cuenta que los defectos se agregan al mensaje donde se encontró el problema, por ejemplo, si un mensaje anidado dentro de un *multipart/alternative* tenía un encabezado mal formado, ese objeto de mensaje anidado tendría un defecto, pero el contenido los mensajes no lo harían.

Todas las clases de defectos se derivan de `email.errors.MessageDefect`.

exception email.errors.NoBoundaryInMultipartDefect

A message claimed to be a multipart, but had no *boundary* parameter.

exception email.errors.StartBoundaryNotFoundDefect

The start boundary claimed in the *Content-Type* header was never found.

exception email.errors.CloseBoundaryNotFoundDefect

A start boundary was found, but no corresponding close boundary was ever found.

Added in version 3.3.

exception email.errors.FirstHeaderLineIsContinuationDefect

The message had a continuation line as its first header line.

exception email.errors.MisplacedEnvelopeHeaderDefect

A «Unix From» header was found in the middle of a header block.

exception email.errors.MissingHeaderBodySeparatorDefect

A line was found while parsing headers that had no leading white space but contained no ":". Parsing continues assuming that the line represents the first line of the body.

Added in version 3.3.

exception email.errors.MalformedHeaderDefect

A header was found that was missing a colon, or was otherwise malformed.

Obsoleto desde la versión 3.3: Este defecto no se ha utilizado por varias versiones de Python.

exception email.errors.MultipartInvariantViolationDefect

A message claimed to be a *multipart*, but no subparts were found. Note that when a message has this defect, its **`is_multipart()`** method may return `False` even though its content type claims to be *multipart*.

exception email.errors.InvalidBase64PaddingDefect

When decoding a block of base64 encoded bytes, the padding was not correct. Enough padding is added to perform the decode, but the resulting decoded bytes may be invalid.

exception email.errors.InvalidBase64CharactersDefect

When decoding a block of base64 encoded bytes, characters outside the base64 alphabet were encountered. The characters are ignored, but the resulting decoded bytes may be invalid.

exception email.errors.InvalidBase64LengthDefect

When decoding a block of base64 encoded bytes, the number of non-padding base64 characters was invalid (1 more than a multiple of 4). The encoded block was kept as-is.

exception email.errors.InvalidDateDefect

When decoding an invalid or unparsable date field. The original value is kept as-is.

[1]

`email.headerregistry`: Custom Header Objects

Código fuente: Lib/email/headerregistry.py [https://github.com/python/cpython/tree/3.14/Lib/email/headerregistry.py]

Added in version 3.6: [1]

Los encabezados están representados por subclases personalizadas de **`str`**. La clase particular utilizada para representar un encabezado dado está determinada por **`header_factory`** del **`policy`** vigente cuando se crean los encabezados. Esta sección documenta el `header_factory` particular implementado por el paquete de correo electrónico para el manejo mensajes de correo electrónico compatibles con **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html], que no solo proporciona objetos de encabezado personalizados para varios tipos de encabezados, sino que también proporciona un mecanismo de extensión para que las aplicaciones agreguen sus propios tipos de encabezados personalizados.

Cuando se utiliza cualquiera de los objetos de política derivados de **`EmailPolicy`**, todos los encabezados son producidos por **`HeaderRegistry`** y tienen **`BaseHeader`** como su última clase base. Cada clase de encabezado tiene una clase base adicional que está determinada por el tipo de encabezado. Por ejemplo, muchos encabezados tienen la clase **`UnstructuredHeader`** como su otra clase base. La segunda clase especializada para un encabezado está determinada por el nombre del encabezado, utilizando una tabla de búsqueda almacenada en **`HeaderRegistry`**. Todo esto se gestiona de forma transparente para el programa de aplicación típico, pero se proporcionan interfaces para modificar el comportamiento predeterminado para su uso por aplicaciones más complejas.

Las secciones a continuación primero documentan las clases base de encabezados y sus atributos, seguidas por la API para modificar el comportamiento de **`HeaderRegistry`**, y finalmente las clases de soporte utilizadas para representar los datos analizados a partir de encabezados estructurados.

class email.headerregistry.BaseHeader(*name*, *value*)

name y *value* se pasan a `BaseHeader` desde la llamada **`header_factory`**. El valor de cadena de caracteres de cualquier objeto de encabezado es el *value* completamente descodificado en unicode.

Esta clase base define las siguientes propiedades de solo lectura:

name

El nombre del encabezado (la parte del campo antes del “:”). Este es exactamente el valor pasado en **`header_factory`** llamada para *name*; es decir, se conserva el caso.

defects

Una tupla de instancias **`HeaderDefect`** que informan sobre cualquier problema de cumplimiento de RFC que se encuentre durante el análisis. El paquete de correo electrónico intenta estar completo para detectar problemas de cumplimiento. Vea el módulo **`errors`** para una discusión de los tipos de defectos que pueden ser reportados.

max_count

El número máximo de encabezados de este tipo que pueden tener el mismo `name`. Un valor de `None` significa ilimitado. El valor de `BaseHeader` para este atributo es `None`; se espera que las clases de encabezado especializadas anulen este valor según sea necesario.

`BaseHeader` también proporciona el siguiente método, que es llamado por el código de la biblioteca de correo electrónico y, en general, no debe ser llamado por programas de aplicación:

fold(*, *policy*)

Retorna una cadena que contenga **`linesep`** caracteres según sea necesario para doblar correctamente el encabezado de acuerdo con *policy*. Un atributo **`cte_type`** de `8bit` se tratará como si fuera `7bit`, ya que los encabezados no pueden contener datos binarios arbitrarios. Si **`utf8`** es `False`, los datos no ASCII estarán codificados RFC 2047 [https://datatracker.ietf.org/doc/html/rfc2047.html].

`BaseHeader` por sí solo no se puede utilizar para crear un objeto de encabezado. Define un protocolo con el que coopera cada encabezado especializado para producir el objeto de encabezado. Específicamente, `BaseHeader` requiere que la clase especializada proporcione un **`classmethod()`** llamado `parse`. Este método se llama de la siguiente manera:

```
parse(string, kwds)
```

`kwds` es un diccionario que contiene una clave preinicializada, `defects`. `defects` es una lista vacía. El método de análisis debe agregar cualquier defecto detectado a esta lista. A la devolución, el diccionario `kwds` *debe* (*must*) contener valores para al menos las claves `decoded` y `defects`. `decoded` debe ser el valor de cadena para el encabezado (es decir, el valor del encabezado completamente decodificado a Unicode). El método de análisis debe asumir que *string* puede contener partes codificadas por transferencia de contenido, pero también debe manejar correctamente todos los caracteres Unicode válidos para que pueda analizar valores de encabezado no codificados.

Entonces, el `__new__` de `BaseHeader` crea la instancia del encabezado y llama a su método `init`. La clase especializada solo necesita proporcionar un método `init` si desea establecer atributos adicionales más allá de los proporcionados por `BaseHeader`. Tal método `init` debería verse así:

```
def init(self, /, *args, **kw):
    self._myattr = kw.pop('myattr')
    super().init(*args, **kw)
```

Es decir, cualquier cosa adicional que la clase especializada ponga en el diccionario `kwds` debe eliminarse y manejarse, y el contenido restante de `kw` (y `args`) debe pasar `BaseHeader` al método `init`.

class email.headerregistry.UnstructuredHeader

Un encabezado «sin estructura» es el tipo predeterminado de encabezado en **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html]. Cualquier encabezado que no tenga una sintaxis especificada se trata como no estructurado. El ejemplo clásico de un encabezado no estructurado es el encabezado *Subject*.

En **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html], un encabezado no estructurado es una ejecución de texto arbitrario en el conjunto de caracteres ASCII. **RFC 2047** [https://datatracker.ietf.org/doc/html/rfc2047.html], sin embargo, tiene un mecanismo compatible **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html] para codificar texto no ASCII como caracteres ASCII dentro de un valor de encabezado. Cuando un *value* que contiene palabras codificadas se pasa al constructor, el analizador `UnstructuredHeader` convierte dichas palabras codificadas en unicode, siguiendo las reglas **RFC 2047** [https://datatracker.ietf.org/doc/html/rfc2047.html] para texto no estructurado. El analizador utiliza heurística para intentar decodificar ciertas palabras codificadas no compatibles. Los defectos se registran en tales casos, así como defectos por problemas como caracteres no válidos dentro de las palabras codificadas o el texto no codificado.

Este tipo de encabezado no proporciona atributos adicionales.

class email.headerregistry.DateHeader

RFC 5322 [https://datatracker.ietf.org/doc/html/rfc5322.html] especifica un formato muy específico para las fechas dentro de los encabezados de correo electrónico. El analizador `DateHeader` reconoce ese formato de fecha, además de reconocer una serie de formas variantes que a veces se encuentran «en la naturaleza» (*«in the wild»*).

Este tipo de encabezado proporciona los siguientes atributos adicionales:

datetime

> Si el valor del encabezado puede reconocerse como una fecha válida de una forma u otra, este atributo contendrá una instancia **`datetime`** que representa esa fecha. Si la zona horaria de la fecha de entrada se especifica como `-0000` (lo que indica que está en UTC pero no contiene información sobre la zona horaria de origen), entonces **`datetime`** será un ingenuo **`datetime`**. Si se encuentra un desplazamiento de zona horaria específico (incluido `+0000`), entonces **`datetime`** contendrá un `datetime` consciente que usa **`datetime.timezone`** para registrar el desplazamiento de la zona horaria.

El valor `decoded` del encabezado se determina formateando la `datetime` de acuerdo con las reglas **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html]; es decir, esto se establece en:

```
email.utils.format_datetime(self.datetime)
```

Al crear un `DateHeader`, *value* puede ser **`datetime`** instancia. Esto significa, por ejemplo, que el siguiente código es válido y hace lo que cabría esperar:

```
msg['Date'] = datetime(2011, 7, 15, 21)
```

Debido a que se trata de una `datetime` naif, se interpretará como una marca de tiempo UTC, y el valor resultante tendrá una zona horaria de `-0000`. Mucho más útil es usar la función **`localtime()`** del módulo **`utils`**:

```
msg['Date'] = utils.localtime()
```

Este ejemplo establece el encabezado de la fecha en la hora y fecha actuales utilizando el desplazamiento de zona horaria actual.

class email.headerregistry.AddressHeader

Los encabezados de dirección son uno de los tipos de encabezados estructurados más complejos. La clase `AddressHeader` proporciona una interfaz genérica para cualquier encabezado de dirección.

Este tipo de encabezado proporciona los siguientes atributos adicionales:

groups

Una tupla de objetos **`Group`** que codifican las direcciones y los grupos que se encuentran en el valor del encabezado. Las direcciones que no forman parte de un grupo se representan en esta lista como `Groups` de una sola dirección cuyo **`display_name`** es `None`.

addresses

Una tupla de objetos **`Address`** que codifican todas las direcciones individuales del valor del encabezado. Si el valor del encabezado contiene algún grupo, las direcciones individuales del grupo se incluyen en la lista en el punto donde el grupo aparece en el valor (es decir, la lista de direcciones se «aplana» (*«flattened»*) en una lista unidimensional).

El valor `decoded` del encabezado tendrá todas las palabras codificadas decodificadas a Unicode. Los nombres de dominio codificados **`idna`** también se decodifican en Unicode. El valor `decoded` se establece concatenando el valor **`str`** de los elementos del atributo `groups` con `', '`.

Se puede utilizar una lista de objetos **`Address`** y **`Group`** en cualquier combinación para establecer el valor de un encabezado de dirección. Los objetos `Group` cuyo `display_name` sea `None` se interpretarán como direcciones únicas, lo que permite copiar una lista de direcciones con los grupos intactos utilizando la lista obtenida del atributo `groups` de el encabezado de origen.

class email.headerregistry.SingleAddressHeader

Una subclase de **`AddressHeader`** que agrega un atributo adicional:

address

La única dirección codificada por el valor del encabezado. Si el valor del encabezado en realidad contiene más de una dirección (lo que sería una violación del RFC bajo el valor predeterminado **`policy`**), acceder a este atributo resultará en un **`ValueError`**.

Muchas de las clases anteriores también tienen una variante `Unique` (por ejemplo,``UniqueUnstructuredHeader``). La única diferencia es que en la variante `Unique`, **`max_count`** se establece en 1.

class email.headerregistry.MIMEVersionHeader

En realidad, solo hay un valor válido para el encabezado *MIME-Version*, y ese es `1.0`. Para pruebas futuras, esta clase de encabezado admite otros números de versión válidos. Si un número de versión tiene un valor válido por **RFC 2045** [https://datatracker.ietf.org/doc/html/rfc2045.html], entonces el objeto de encabezado tendrá valores distintos de `None` para los siguientes atributos:

version

El número de versión como una cadena de caracteres, con cualquier espacio en blanco y/o comentarios eliminados.

major

El número de versión mayor como un entero minor

El número de versión menor como un entero

class email.headerregistry.ParameterizedMIMEHeader

Todos los encabezados *MIME* comienzan con el prefijo *"Content-"*. Cada encabezado específico tiene un valor determinado, que se describe en la clase de ese encabezado. Algunos también pueden tomar una lista de parámetros complementarios, que tienen un formato común. Esta clase sirve como base para todos los encabezados *MIME* que toman parámetros.

params

Un diccionario que asigna nombres de parámetros a valores de parámetros.

class email.headerregistry.ContentTypeHeader

Una clase **`ParameterizedMIMEHeader`** que maneja el encabezado *Content-Type*.

content_type

La cadena de caracteres de tipo de contenido, en la forma `maintype/subtype`.

maintype

subtype

class email.headerregistry.ContentDispositionHeader

Una clase **`ParameterizedMIMEHeader`** que maneja el encabezado *Content-Disposition*.

content_disposition

`inline` y `attachment` son los únicos valores válidos de uso común.

class email.headerregistry.ContentTransferEncodingHeader

Maneja el encabezado de *Content-Transfer-Encoding*.

cte

Los valores válidos son `7bit`, `8bit`, `base64`, y `quoted-printable`. Consulte **RFC 2045** [https://datatracker.ietf.org/doc/html/rfc2045.html] para obtener más información.

class email.headerregistry.HeaderRegistry(*base_class=BaseHeader*, *default_class=UnstructuredHeader*, *use_default_map=True*)

This is the factory used by `EmailPolicy` by default. `HeaderRegistry` builds the class used to create a header instance dynamically, using *base_class* and a specialized class retrieved from a registry that it holds. When a given header name does not appear in the registry, the class specified by *default_class* is used as the specialized class. When *use_default_map* is `True` (the default), the standard mapping of header names to classes is copied in to the registry during initialization. *base_class* is always the last class in the generated class's `__bases__` list.

Las asignaciones predeterminadas son:

subject: UniqueUnstructuredHeader
date: UniqueDateHeader
resent-date: DateHeader
orig-date: UniqueDateHeader
sender: UniqueSingleAddressHeader
resent-sender: SingleAddressHeader
to: UniqueAddressHeader
resent-to: AddressHeader
cc: UniqueAddressHeader

resent-cc:
AddressHeader
bcc:
UniqueAddressHeader
resent-bcc:
AddressHeader
from:
UniqueAddressHeader
resent-from:
AddressHeader
reply-to:
UniqueAddressHeader
mime-version:
MIMEVersionHeader
content-type:
ContentTypeHeader
content-disposition:
ContentDispositionHeader
content-transfer-encoding:
ContentTransferEncodingHeader
message-id:
MessageIDHeader

`HeaderRegistry` tiene los siguientes métodos:

map_to_type(*self*, *name*, *cls*)

> *name* es el nombre del encabezado que se asignará. Se convertirá a minúsculas en el registro. *cls* es la clase especializada que se utilizará, junto con *base_class*, para crear la clase utilizada para instanciar encabezados que coincidan con *name*.

__getitem__(*name*)

> Construye y retorna una clase para manejar la creación de un encabezado *nombre*.

__call__(*name*, *value*)

> Recupera el encabezado especializado asociado con

name del registro (usando *default_class* si *name* no aparece en el registro) y lo compone con *base_class* para producir una clase, llama al constructor de la clase construida, pasándole el mismo lista de argumentos y, finalmente, retorna la instancia de clase creada de ese modo.

Las siguientes clases son las clases que se utilizan para representar datos analizados a partir de encabezados estructurados y, en general, pueden ser utilizadas por un programa de aplicación para construir valores estructurados para asignar a encabezados específicos.

class email.headerregistry.Address(*display_name=''*, *username=''*, *domain=''*, *addr_spec=None*)

La clase utilizada para representar una dirección de correo electrónico. La forma general de una dirección es:

```
[display_name] <username@domain>
```

or:

```
username@domain
```

donde cada parte debe ajustarse a reglas de sintaxis específicas explicadas en **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html].

Para su comodidad, se puede especificar *addr_spec* en lugar de *username* y *domain*, en cuyo caso *username* y *domain* se analizarán a partir de *addr_spec*. Un *addr_spec* debe ser una cadena de caracteres entre comillas RFC adecuada; si no es `Address`, se lanzará un error. Se permiten caracteres Unicode y se codificarán como propiedad cuando se serialicen. Sin embargo, según las RFC, *no* se permite unicode en la parte del nombre de usuario de la dirección.

display_name

La parte del nombre para mostrar de la dirección, si la hubiera, con todas las citas eliminadas. Si la dirección no tiene un nombre para mostrar, este atributo será una cadena vacía.

username

La parte del `username` de la dirección, con todas las citas eliminadas.

domain

La parte de `domain` de la dirección.

addr_spec

La parte de la dirección `username@domain`, citada correctamente para usarla como dirección simple (el segundo formulario que se muestra arriba). Este atributo no es mutable.

str()

El valor `str` del objeto es la dirección citada de acuerdo con las reglas **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html], pero sin codificación de transferencia de contenido de ningún carácter que no sea ASCII.

Para admitir SMTP (**RFC 5321** [https://datatracker.ietf.org/doc/html/rfc5321.html]), `Address` maneja un caso especial: si `username` y `domain` son ambos la cadena de caracteres vacía (o `None`), entonces el valor de cadena de caracteres `Address` es `<>`.

class email.headerregistry.Group(*display_name=None*, *addresses=None*)

La clase utilizada para representar un grupo de direcciones. La forma general de un grupo de direcciones es:

```
display_name: [address-list];
```

Para facilitar el procesamiento de listas de direcciones que constan de una mezcla de grupos y direcciones únicas, también se puede utilizar un `Group` para representar direcciones únicas que no forman parte de un grupo al establecer *display_name* en `None` y proporcionando una lista de direcciones únicas como *addresses*.

display_name

El `display_name` del grupo. Si es `None` y hay exactamente una `Address` en `addresses`, entonces el `Group` representa una única dirección que no está en un grupo.

addresses

Posiblemente una tupla vacía de **`Address`** que representan las direcciones en el grupo.

__str__()

El valor `str` de un `Group` se formatea de acuerdo con **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html], pero sin codificación de transferencia de contenido de ningún carácter que no sea ASCII. Si `display_name` no es ninguno y hay una sola `Address` en la lista de `addresses`, el valor de `str` será el mismo que el `str` de ese single `Address`.

Pie de notas

Originalmente añadido en 3.3 como módulo provisional provisional module

[1]

`email.contentmanager`: Managing MIME Content

Código fuente: Lib/email/contentmanager.py [https://github.com/python/cpython/tree/3.14/Lib/email/contentmanager.py]

Added in version 3.6: [1]

class email.contentmanager.ContentManager

Clase base para gestores de contenido. Proporciona los mecanismos de registro estándar para registrar convertidores entre contenido MIME y otras representaciones, así como los métodos de envío `get_content` y `set_content`.

get_content(*msg*, **args*, ***kw*)

Busca una función de controlador basada en el `mimetype` de *msg* (ver el siguiente párrafo), la llama, le pasa todos los argumentos y retorna el resultado de la llamada. La expectativa es que el controlador extraiga la carga útil de *msg* y retorne un objeto que codifica información sobre los datos extraídos.

Para encontrar el controlador, busca las siguientes llaves en el registro, deteniéndose con la primera que encuentre:

- la cadena que representa el tipo MIME completo (`maintype/subtype`)
- la cadena de caracteres que representa el `maintype`
- la cadena de caracteres vacía

Si ninguna de estas llaves produce un controlador, se lanza una excepción **`KeyError`** para el tipo MIME completo.

set_content(*msg*, *obj*, **args*, ***kw*)

Si el `maintype` es `multipart`, se lanza un **`TypeError`**; de lo contrario, busca una función de controlador basada en el tipo de *obj* (ver el siguiente párrafo), llama a **`clear_content()`** en el *msg* y llama a la función de controlador, pasando todos los argumentos. La expectativa es que el controlador transforme y almacene *obj* en *msg*, posiblemente realizando otros cambios a *msg* también, como agregar varios encabezados MIME para codificar la información necesaria para interpretar los datos almacenados.

Para encontrar el controlador, obtiene el tipo de *obj* (`typ = type(obj)`), y busca las siguientes llaves en el registro, deteniéndose con la primera encontrada:

- el tipo en sí (`typ`)
- el nombre completo de calificación del tipo (`typ.__module__ + '.' + typ.__qualname__`).
- the type's **`qualname`** (`typ.__qualname__`)
- the type's **`name`** (`typ.__name__`).

If none of the above match, repeat all of the checks above for each of the types in the MRO (**`typ.__mro__`**). Finally, if no other key yields a handler, check for a handler for the key `None`. If there is no handler for `None`, raise a **`KeyError`** for the fully qualified name of the type.

También agrega un encabezado *MIME-Version* si no hay uno presente (vea también **`MIMEPart`**).

add_get_handler(*key*, *handler*)

Registra el *handler* de funciones como el manejador de *key*. Para los posibles valores de *key*, consulte **`get_content()`**.

add_set_handler(*typekey*, *handler*)

Registra el *handler* como la función a llamar cuando un objeto de un tipo coincidente *typekey* se pasa a `set_content()`. Para los posibles valores de *typekey*, consulte **`set_content()`**.

Instancias gestoras de contenido

Actualmente, el paquete de correo electrónico solo proporciona un administrador de contenido concreto, **`raw_data_manager`**, aunque en el futuro se pueden agregar más. **`raw_data_manager`** es el **`content_manager`** proporcionado por **`EmailPolicy`** y sus derivados.

email.contentmanager.raw_data_manager

Este administrador de contenido proporciona sólo una interfaz mínima más allá de la proporcionada por **`Message`** en sí: trata solo con texto, cadenas de bytes sin procesar, y objetos **`Message`**. Sin embargo, proporciona ventajas significativas en comparación con la API base: `get_content` en una parte de texto retornará una cadena de caracteres unicode sin que la aplicación tenga que decodificarla manualmente, `set_content` proporciona un amplio conjunto de opciones para controlar los encabezados añadidos a una parte y controlar la codificación de transferencia de contenido, y permite el uso de los diversos métodos `add_`, simplificando así la creación de mensajes multiparte.

email.contentmanager.get_content(*msg*, *errors='replace'*)

Retorna la carga útil de la parte como una cadena de caracteres (para partes de `text`), un objeto **`EmailMessage`** (para partes de `message/rfc822`), o un objeto de `bytes` (para todos los demás tipos que no son multiparte). Lanza un **`KeyError`** si se llama en un `multipart`. Si la parte es una parte de `text` y se especifica *errors*, se usa como el controlador de errores al decodificar la carga útil a unicode. El controlador de errores predeterminado es `replace`.

email.contentmanager.set_content(*msg*, *<'str'>*, *subtype="plain"*, *charset='utf-8'*, *cte=None*, *disposition=None*, *filename=None*, *cid=None*, *params=None*, *headers=None*)
email.contentmanager.set_content(*msg*, *<'bytes'>*, *maintype*, *subtype*, *cte="base64"*, *disposition=None*, *filename=None*, *cid=None*, *params=None*, *headers=None*)
email.contentmanager.set_content(*msg*, *<'EmailMessage'>*, *cte=None*, *disposition=None*, *filename=None*, *cid=None*, *params=None*, *headers=None*)

Añade cabeceras y carga útil al *msg*:

Añade un encabezado *Content-Type* con un valor `maintype/subtype`.

- Para `str`, establece el `maintype` de MIME en `text`, y establece el subtipo en *subtype* si se especifica, o `plain` si no está presente.
- Para `bytes`, usa el *maintype* y *subtype* especificados, o lanza un **`TypeError`** si no se especifican.
- Para objetos **`EmailMessage`**, establece el *maintype* en `message`, y establece el *subtype* en *subtype* si se especifica o `rfc822` si no se especifica. Si *subtype* es `partial`, se lanza un error (los objetos de `bytes` deben usarse para construir partes `message/partial`).

Si se proporciona *charset* (lo cual solo es válido para `str`), codifica la cadena de caracteres en bytes utilizando el conjunto de caracteres especificado. El valor por defecto es `utf-8`. Si el *charset* especificado es un alias conocido del nombre de un conjunto de caracteres del estándar MIME, utiliza el conjunto de caracteres estándar en su lugar.

Si se establece *cte*, codifica la carga útil mediante la codificación de transferencia de contenido especificada y establece el encabezado *Content-Transfer-Encoding* en ese valor. Los valores posibles para *cte* son `quoted-`

`printable`, `base64`, `7bit`, `8bit`, y `binary`. Si la entrada no se puede codificar en la codificación especificada (por ejemplo, especificando un *cte* de `7bit` para una entrada que contiene valores no ASCII), se lanza un **`ValueError`**.

- For `str` objects, if *cte* is not set use heuristics to determine the most compact encoding. Prior to encoding, **`str.splitlines()`** is used to normalize all line boundaries, ensuring that each line of the payload is terminated by the current policy's **`linesep`** property (even if the original string did not end with one).
- For `bytes` objects, *cte* is taken to be base64 if not set, and the aforementioned newline translation is not performed.
- Para **`EmailMessage`**, según **RFC 2046** [https://datatracker.ietf.org/doc/html/rfc2046.html], se lanza un error si se solicita un *cte* de `quoted-printable` o `base64` para el *subtype* `rfc822`, y para cualquier *cte* que no sea `7bit` para el *subtype* `external-body`. Para `message/rfc822`, se usa `8bit` si no se especifica *cte*. Para todos los demás valores de *subtype*, se usa `7bit`.

Nota

Un *cte* de `binary` todavía no funciona correctamente. El objeto `EmailMessage` modificado por `set_content` es correcto, pero **`BytesGenerator`** no lo serializa correctamente.

Si se establece *disposición*, se usa como valor del encabezado *Content-Disposition*. Si no se especifica y se especifica *filename*, agrega el encabezado con el valor `attachment`. Si no se especifica *disposition* y tampoco se especifica *filename*, no agrega el encabezado. Los únicos valores válidos para *disposition* son `attachment` e `inline`.

Si se especifica el *filename*, se usa como el valor del parámetro `filename` del encabezado *Content-Disposition*.

Si se especifica *cid*, agrega un encabezado *Content-ID* con valor *cid*.

Si se especifica *params*, itera su método `items` y use los pares resultantes `(key, value)` para establecer parámetros adicionales en el encabezado *Content-Type*.

Si se especifica *headers* y es una lista de cadenas de caracteres de la forma `headername: headervalue` o una lista de objetos `header` (que se distinguen de las cadenas de caracteres por tener un atributo `name`), agrega los encabezados a *msg*.

Notas al pie de página

Originalmente añadido en la versión 3.4 como un módulo provisional

[1]

`email`: Ejemplos

Aquí hay algunos ejemplos de cómo usar el paquete `email` para leer, escribir y enviar mensajes de correo electrónico simples, así como mensajes MIME más complejos.

Primero, veamos cómo crear y enviar un mensaje de texto simple (tanto el contenido de texto como las direcciones pueden contener caracteres unicode):

```
# Import smtplib for the actual sending function
import smtplib

# Import the email modules we'll need
from email.message import EmailMessage

# Open the plain text file whose name is in textfile for
with open(textfile) as fp:
    # Create a text/plain message
    msg = EmailMessage()
    msg.set_content(fp.read())

# me == the sender's email address
# you == the recipient's email address
msg['Subject'] = f'The contents of {textfile}'
msg['From'] = me
msg['To'] = you

# Send the message via our own SMTP server.
s = smtplib.SMTP('localhost')
s.send_message(msg)
s.quit()
```

El análisis de los encabezados **RFC 822** [https://datatracker.ietf.org/doc/html/rfc822.html] se pueden hacer fácilmente usando las clases del módulo **`parser`**:

```
# Import the email modules we'll need
#from email.parser import BytesParser
from email.parser import Parser
from email.policy import default

# If the e-mail headers are in a file, uncomment these t
# with open(messagefile, 'rb') as fp:
#     headers = BytesParser(policy=default).parse(fp)

#  Or for parsing headers in a string (this is an uncomm
headers = Parser(policy=default).parsestr(
        'From: Foo Bar <user@example.com>\n'
        'To: <someone_else@example.com>\n'
        'Subject: Test message\n'
        '\n'
        'Body would go here\n')

#  Now the header items can be accessed as a dictionary:
print('To: {}'.format(headers['to']))
print('From: {}'.format(headers['from']))
print('Subject: {}'.format(headers['subject']))

# You can also access the parts of the addresses:
print('Recipient username: {}'.format(headers['to'].addr
print('Sender name: {}'.format(headers['from'].addresses
```

Aquí hay un ejemplo de cómo enviar un mensaje MIME que contiene un grupo de fotos familiares que pueden residir en un directorio:

```
# Import smtplib for the actual sending function.
import smtplib

# Here are the email package modules we'll need.
from email.message import EmailMessage

# Create the container email message.
```

```
msg = EmailMessage()
msg['Subject'] = 'Our family reunion'
# me == the sender's email address
# family = the list of all recipients' email addresses
msg['From'] = me
msg['To'] = ', '.join(family)
msg.preamble = 'You will not see this in a MIME-aware ma

# Open the files in binary mode.  You can also omit the
# if you want MIMEImage to guess it.
for file in pngfiles:
    with open(file, 'rb') as fp:
        img_data = fp.read()
    msg.add_attachment(img_data, maintype='image',
                                 subtype='png')

# Send the email via our own SMTP server.
with smtplib.SMTP('localhost') as s:
    s.send_message(msg)
```

Aquí hay un ejemplo de cómo enviar todo el contenido de un directorio como un mensaje de correo electrónico: [1]

```
#!/usr/bin/env python3

"""Send the contents of a directory as a MIME message.""

import os
import smtplib
# For guessing MIME type based on file name extension
import mimetypes

from argparse import ArgumentParser

from email.message import EmailMessage
from email.policy import SMTP


def main():
    parser = ArgumentParser(description="""\
Send the contents of a directory as a MIME message.
Unless the -o option is given, the email is sent by forw
SMTP server, which then does the normal delivery process
must be running an SMTP server.
""")
    parser.add_argument('-d', '--directory',
                        help="""Mail the contents of the
                        otherwise use the current direct
                        files in the directory are sent,
                        subdirectories.""")
    parser.add_argument('-o', '--output',
                        metavar='FILE',
                        help="""Print the composed messa
                        sending the message to the SMTP
    parser.add_argument('-s', '--sender', required=True,
                        help='The value of the From: hea
    parser.add_argument('-r', '--recipient', required=Tr
                        action='append', metavar='RECIPI
                        default=[], dest='recipients',
                        help='A To: header value (at lea
    args = parser.parse_args()
    directory = args.directory
    if not directory:
        directory = '.'
    # Create the message
    msg = EmailMessage()
    msg['Subject'] = f'Contents of directory {os.path.ab
    msg['To'] = ', '.join(args.recipients)
    msg['From'] = args.sender
    msg.preamble = 'You will not see this in a MIME-awar

    for filename in os.listdir(directory):
        path = os.path.join(directory, filename)
        if not os.path.isfile(path):
            continue
        # Guess the content type based on the file's ext
        # will be ignored, although we should check for
        # gzip'd or compressed files.
        ctype, encoding = mimetypes.guess_file_type(path
        if ctype is None or encoding is not None:
            # No guess could be made, or the file is enc
            # use a generic bag-of-bits type.
            ctype = 'application/octet-stream'
        maintype, subtype = ctype.split('/', 1)
        with open(path, 'rb') as fp:
            msg.add_attachment(fp.read(),
                               maintype=maintype,
                               subtype=subtype,
                               filename=filename)
    # Now send or store the message
    if args.output:
        with open(args.output, 'wb') as fp:
            fp.write(msg.as_bytes(policy=SMTP))
    else:
        with smtplib.SMTP('localhost') as s:
            s.send_message(msg)


if __name__ == '__main__':
    main()
```

Aquí hay un ejemplo de cómo descomprimir un mensaje MIME como el anterior, en un directorio de archivos:

```
#!/usr/bin/env python3

"""Unpack a MIME message into a directory of files."""

import os
import email
import mimetypes

from email.policy import default

from argparse import ArgumentParser
```

```
def main():
    parser = ArgumentParser(description="""\
Unpack a MIME message into a directory of files.
""")
    parser.add_argument('-d', '--directory', required=Tr
                        help="""Unpack the MIME message
                        directory, which will be create
                        exist.""")
    parser.add_argument('msgfile')
    args = parser.parse_args()

    with open(args.msgfile, 'rb') as fp:
        msg = email.message_from_binary_file(fp, policy=

    try:
        os.mkdir(args.directory)
    except FileExistsError:
        pass

    counter = 1
    for part in msg.walk():
        # multipart/* are just containers
        if part.get_content_maintype() == 'multipart':
            continue
        # Applications should really sanitize the given
        # email message can't be used to overwrite impor
        filename = part.get_filename()
        if not filename:
            ext = mimetypes.guess_extension(part.get_con
            if not ext:
                # Use a generic bag-of-bits extension
                ext = '.bin'
            filename = f'part-{counter:03d}{ext}'
        counter += 1
        with open(os.path.join(args.directory, filename)
            fp.write(part.get_payload(decode=True))
```

```
if __name__ == '__main__':
    main()
```

Aquí hay un ejemplo de cómo crear un mensaje HTML con una versión alternativa de texto sin formato. Para hacer las cosas un poco más interesantes, incluimos una imagen relacionada en la parte html, y guardamos una copia de lo que vamos a enviar en el disco, además de enviarlo.

```
#!/usr/bin/env python3

import smtplib

from email.message import EmailMessage
from email.headerregistry import Address
from email.utils import make_msgid

# Create the base text message.
msg = EmailMessage()
msg['Subject'] = "Pourquoi pas des asperges pour ce midi
msg['From'] = Address("Pepé Le Pew", "pepe", "example.co
msg['To'] = (Address("Penelope Pussycat", "penelope", "e
             Address("Fabrette Pussycat", "fabrette", "e
msg.set_content("""\
Salut!

Cette recette [1] sera sûrement un très bon repas.

[1] http://www.yummly.com/recipe/Roasted-Asparagus-Epicu

--Pepé
""")

# Add the html version.  This converts the message into
# container, with the original text message as the first
# message as the second part.
asparagus_cid = make_msgid()
msg.add_alternative("""\
<html>
  <head></head>
  <body>
    <p>Salut!</p>
    <p>Cette
        <a href="http://www.yummly.com/recipe/Roasted-As
            recette
        </a> sera sûrement un très bon repas.
    </p>
    <img src="cid:{asparagus_cid}">
  </body>
</html>
""".format(asparagus_cid=asparagus_cid[1:-1]), subtype='
# note that we needed to peel the <> off the msgid for u

# Now add the related image to the html part.
with open("roasted-asparagus.jpg", 'rb') as img:
    msg.get_payload()[1].add_related(img.read(), 'image'
                                     cid=asparagus_cid)

# Make a local copy of what we are going to send.
with open('outgoing.msg', 'wb') as f:
    f.write(bytes(msg))

# Send the message via local SMTP server.
with smtplib.SMTP('localhost') as s:
    s.send_message(msg)
```

Si nos enviaron el mensaje del último ejemplo, aquí hay una forma en que podríamos procesarlo:

```
import os
import sys
import tempfile
import mimetypes
import webbrowser

# Import the email modules we'll need
from email import policy
from email.parser import BytesParser


def magic_html_parser(html_text, partfiles):
    """Return safety-sanitized html linked to partfiles.

    Rewrite the href="cid:...." attributes to point to t
    Though not trivial, this should be possible using ht
    """
    raise NotImplementedError("Add the magic needed")


# In a real program you'd get the filename from the argu
with open('outgoing.msg', 'rb') as fp:
    msg = BytesParser(policy=policy.default).parse(fp)

# Now the header items can be accessed as a dictionary,
# be converted to unicode:
print('To:', msg['to'])
print('From:', msg['from'])
print('Subject:', msg['subject'])

# If we want to print a preview of the message content,
# the least formatted payload is and print the first thr
# if the message has no plain text part printing the fir
# is probably useless, but this is just a conceptual exa
simplest = msg.get_body(preferencelist=('plain', 'html')
print()
print(''.join(simplest.get_content().splitlines(keepends

ans = input("View full message?")
if ans.lower()[0] == 'n':
    sys.exit()

# We can extract the richest alternative in order to dis
richest = msg.get_body()
partfiles = {}
if richest['content-type'].maintype == 'text':
    if richest['content-type'].subtype == 'plain':
        for line in richest.get_content().splitlines():
            print(line)
        sys.exit()
    elif richest['content-type'].subtype == 'html':
        body = richest
    else:
        print("Don't know how to display {}".format(rich
        sys.exit()
elif richest['content-type'].content_type == 'multipart/
    body = richest.get_body(preferencelist=('html'))
    for part in richest.iter_attachments():
        fn = part.get_filename()
        if fn:
            extension = os.path.splitext(part.get_filena
        else:
            extension = mimetypes.guess_extension(part.g
        with tempfile.NamedTemporaryFile(suffix=extensio
            f.write(part.get_content())
            # again strip the <> to go from email form c
            partfiles[part['content-id'][1:-1]] = f.name
else:
    print("Don't know how to display {}".format(richest.
    sys.exit()
with tempfile.NamedTemporaryFile(mode='w', delete=False)
    f.write(magic_html_parser(body.get_content(), partfi
webbrowser.open(f.name)
os.remove(f.name)
for fn in partfiles.values():
    os.remove(fn)

# Of course, there are lots of email messages that coulc
# minded program, but it will handle the most common one
```

Hasta el aviso, el resultado de lo anterior es:

```
To: Penelope Pussycat <penelope@example.com>, Fabrette P
From: Pepé Le Pew <pepe@example.com>
Subject: Pourquoi pas des asperges pour ce midi ?
```

```
Salut!

Cette recette [1] sera sûrement un très bon repas.
```

Notas al pie

Gracias a *Matthew Dixon Cowles* por la inspiración y ejemplos originales.

`email.message.Message`: Representar un mensaje de correo electrónico usando la API `compat32`

La clase `Message` es muy similar a la clase `EmailMessage`, sin los métodos añadidos por esa clase y con el comportamiento predeterminado de algunos otros métodos siendo ligeramente diferente. También documentamos aquí algunos métodos que, aun siendo soportados por **`EmailMessage`**, no están recomendados a no ser que estés lidiando con código heredado.

Por lo demás, la filosofía y estructura de las dos clases es la misma.

Este documento describe el comportamiento bajo la política por defecto (para `Message`) `Compat32`. Si vas a usar otra política, deberías estar usando la clase `EmailMessage` en su lugar.

Un mensaje de correo electrónico consta de *headers* y *payload*. Los encabezados deben ser nombres y valores de estilo RFC 5322 [https://datatracker.ietf.org/doc/html/rfc5322.html], donde el nombre del campo y el valor están separados por dos puntos. Los dos puntos no forman parte del nombre del campo ni del valor del campo. La carga útil puede ser un mensaje de texto simple, un objeto binario o una secuencia estructurada de submensajes, cada uno con su propio conjunto de encabezados y su propia carga útil. El último tipo de carga útil se indica mediante el mensaje que tiene un tipo MIME como *multipart/** o *message/rfc822*.

El modelo conceptual proporcionado por un objeto `Message` es el de un diccionario ordenado de encabezados con métodos adicionales para acceder a información especializada de los encabezados, para acceder a la carga, para generar una versión serializada del mensaje y para recorrer recursivamente el árbol del objeto. Ten en cuenta que son soportados encabezados duplicados pero deben ser usados métodos especiales para acceder a ellos.

El pseudodiccionario `Message` es indexado por los nombres de encabezados, los cuales deben ser valores ASCII. Los valores del diccionario son cadenas que se supone que contienen sólo caracteres ASCII; hay algún manejo especial para la entrada no ASCII, pero esta no siempre produce los resultados correctos. Los encabezados son almacenados y retornados preservando mayúsculas y minúsculas, pero los nombres de campos son emparejados sin distinción entre mayúsculas y minúsculas. También puede haber sólo un encabezado de envoltura, también conocido como el encabezado *Unix-From* o el encabezado `From_`. La carga (*payload*) es una cadena o bytes, en el caso de objetos de mensajes simples, o una lista de objetos **`Message`**, para contenedores de documentos MIME (ej. *multipart/** y *message/rfc822*).

Aquí están los métodos de la clase `Message`:

class email.message.Message(*policy=compat32*)

Si se especifica *policy* (debe ser una instancia de una clase `policy`) utiliza las reglas que especifica para actualizar y serializar la representación del mensaje. Si no se define *policy*, utiliza la política `compat32`, la cual mantiene compatibilidad con la versión de Python 3.2 del paquete email. Para más información consulta la documentación de **`policy`**.

Distinto en la versión 3.3: El argumento de palabra clave *policy* fue añadido.

as_string(*unixfrom=False, maxheaderlen=0, policy=None*)

Retorna el mensaje completo aplanado como una cadena. Cuando el parámetro opcional *unixfrom* es verdadero, el encabezado de envoltura se incluye en la cadena retornada. *unixfrom* es por defecto `False`. Por razones de compatibilidad con versiones anteriores, *maxheaderlen* es `0` por defecto, por lo que si quieres un valor diferente debes debes sobreescribirlo explícitamente (el valor especificado por *max_line_length* en la política será ignorado por este método). El argumento *policy* puede ser usado para sobrescribir la política por defecto obtenida de la instancia del mensaje. Esto puede ser usado para controlar algo del formato producido por el método, ya que la *policy* especificada puede ser pasada al `Generator`.

Aplanar el mensaje puede desencadenar cambios en **`Message`** si por defecto necesita ser rellenado para completar la transformación a una cadena (por ejemplo, límites MIME pueden ser generados o modificados).

Ten en cuenta que este método es proporcionado como conveniencia y puede no siempre formatear el mensaje de la forma que quieres. Por ejemplo, de forma predeterminada no realiza la manipulación de líneas que comienzan con `From` que es requerida por el formato mbox de Unix. Para mayor flexibilidad, instancia un **`Generator`** y utiliza su método **`flatten()`** directamente. Por ejemplo:

```
from io import StringIO
from email.generator import Generator
fp = StringIO()
g = Generator(fp, mangle_from_=True, maxheaderl
g.flatten(msg)
text = fp.getvalue()
```

Si el objeto de mensaje contiene datos binarios que no están codificados de acuerdo a los estándares RFC, los datos no compatibles serán reemplazados por puntos de código Unicode de «carácter desconocido». (Consulta también **`as_bytes()`** y **`BytesGenerator`**.)

Distinto en la versión 3.4: el argumento de palabra clave *policy* fue añadido.

__str__()

Equivalent to **`as_string()`**. Allows `str(msg)` to produce a string containing the formatted message.

as_bytes(*unixfrom=False, policy=None*)

Retorna el mensaje completo aplanado como un objeto de bytes. Cuando el argumento opcional *unixfrom* es verdadero, el encabezado de envoltura se incluye en la cadena retornada. *unixfrom* es por defecto `False`. El argumento *policy* puede ser usado para sobrescribir la política por defecto obtenida desde la instancia del mensaje. Esto puede ser usado para controlar algo del formato producido por el método, ya que el *policy* especificado será pasado al `BytesGenerator`.

Aplanar el mensaje puede desencadenar cambios en **`Message`** si por defecto necesita ser rellenado para completar la transformación a una cadena (por ejemplo, límites MIME pueden ser generados o modificados).

Ten en cuenta que este método es proporcionado como conveniencia y puede no siempre formatear el mensaje de la forma que quieres. Por ejemplo, de forma predeterminada no realiza la manipulación de líneas que comienzan con `From` que es requerida por el formato mbox de Unix. Para mayor flexibilidad, instancia un **`BytesGenerator`** y utiliza su método **`flatten()`** directamente. Por ejemplo:

```
from io import BytesIO
from email.generator import BytesGenerator
fp = BytesIO()
g = BytesGenerator(fp, mangle_from_=True, maxhe
g.flatten(msg)
text = fp.getvalue()
```

Added in version 3.4.

bytes()

Equivalent to **`as_bytes()`**. Allows `bytes(msg)` to produce a bytes object containing the formatted message.

Added in version 3.4.

is_multipart()

Retorna `True` si la carga del mensaje es una lista de objetos heredados de **`Message`**, si no retorna `False`. Cuando **`is_multipart()`** retorna `False`, la carga debe ser un objeto de cadena (el cual puede ser una carga CTE codificada en binario). (Ten en cuenta que **`is_multipart()`** retornando `True` no significa necesariamente que «msg.get_content_maintype() == "multipart"» retornará `True`. Por ejemplo, `is_multipart` retornará `True` cuando el **`Message`** es de tipo `message/rfc822`.)

set_unixfrom(*unixfrom*)

Establece el mensaje del encabezado de envoltura a *unixfrom*, el cual debe ser una cadena.

get_unixfrom()

Retorna el mensaje del encabezado de envoltura. Por defecto a `None` si el encabezado de envoltura nunca fue definido.

attach(*payload*)

Añade el *payload* dado a la carga actual, la cual debe ser `None` o una lista de objetos **`Message`** antes de la invocación. Después de la invocación, la carga siempre será una lista de objetos **`Message`**. Si quieres definir la carga a un objeto escalar (ej. una cadena), usa **`set_payload()`** en su lugar.

This is a legacy method. On the **`EmailMessage`** class its functionality is replaced by **`set_content()`** and the related `make` and `add` methods.

get_payload(*i*=*None*, *decode*=*False*)

Retorna la carga (*payload*) actual, la cual será una lista de objetos **`Message`** cuando **`is_multipart()`** es `True`, o una cadena cuando **`is_multipart()`** es `False`. Si la carga es una lista y mutas el objeto de lista, modificarás la carga del mensaje.

Con el argumento opcional *i*, **`get_payload()`** retornará el elemento número *i* de la carga (*payload*), contando desde cero, si **`is_multipart()`** es `True`. Un **`IndexError`** será generado si *i* es menor que 0 ó mayor o igual que el número de elementos en la carga. Si la carga es una cadena (ej. **`is_multipart()`** es `False`) y se define *i*, se genera un **`TypeError`**.

El argumento opcional *decode* es un indicador que determina si una carga debería ser decodificada o no, de acuerdo al encabezado *Content-Transfer-Encoding*. Cuando es `True` y el mensaje no es multiparte, la carga será decodificada si el valor de su encabezado es `quoted-printable` o `base64`. Si se usa alguna otra codificación o falta el encabezado *Content-Transfer-Encoding*, la carga es retornada tal cual (sin decodificar). En todos los casos el valor retornado son datos binarios. Si el mensaje es multiparte y el indicador *decode* es `True`, entonces se retorna `None`. Si la carga es base64 y no fue perfectamente formada (falta relleno, tiene caracteres fuera del alfabeto base64), entonces un defecto apropiado será añadido a la propiedad defect del mensaje (**`InvalidBase64PaddingDefect`** o **`InvalidBase64CharactersDefect`**, respectivamente).

Cuando *decode* es `False` (por defecto) el cuerpo es retornado como una cadena sin decodificar el *Content-Transfer-Encoding*. Sin embargo, para un *Content-Transfer-Encoding* de 8bit, se realiza un intento para decodificar los bytes originales usando el `charset` especificado por el encabezado *Content-Type*, usando el manejador de error `replace`. Si ningún `charset` es especificado o si el `charset` dado no es reconocido por el paquete email, el cuerpo es decodificado usando el conjunto de caracteres ASCII por defecto.

This is a legacy method. On the **`EmailMessage`** class its functionality is replaced by **`get_content()`** and **`iter_parts()`**.

set_payload(*payload*, *charset=None*)

Define la carga completa del objeto mensaje a *payload*. Es responsabilidad del cliente asegurar invariantes de carga. El argumento opcional *charset* define el conjunto de caracteres por defecto del mensaje; consulta **`set_charset()`** para más detalles.

This is a legacy method. On the **`EmailMessage`** class its functionality is replaced by **`set_content()`**.

set_charset(*charset*)

Define el junto de caracteres de la carga a *charset*, el cual puede ser tanto una instancia **`Charset`** (ver **`email.charset`**), una cadena denominando un conjunto de caracteres, o `None`. Si es una cadena, será convertida a una instancia **`Charset`**. Si *charset* es `None`, el parámetro `charset` será eliminado del encabezado *Content-Type* (el mensaje no será modificado de otra manera). Cualquier otro valor lanzará un **`TypeError`**.

Si no hay un encabezado existente *MIME-Version*, será añadido uno. Si no hay un encabezado existente *Content-Type*, será añadido uno con valor *text/plain*. Tanto como si el encabezado *Content-Type* existe actualmente como si no, su parámetro `charset` será establecido a *charset.output_charset*. Si *charset.input_charset* y *charset.output_charset* difieren, la carga será recodificada al *output_charset*. Si no hay un encabezado existente *Content-Transfer-Encoding*, entonces la carga será codificada por transferencia, si es necesario, usando el `Charset` especificado y un encabezado con el valor apropiado será añadido. Si ya existe un encabezado *Content-Transfer-Encoding*, la carga se asume que ya está correctamente codificada usando ese *Content-Transfer-Encoding* y no es modificada.

This is a legacy method. On the `EmailMessage` class its functionality is replaced by the *charset* parameter of the `email.message.EmailMessage.set_content()` method.

get_charset()

Retorna la instancia `Charset` asociada con la carga del mensaje.

This is a legacy method. On the `EmailMessage` class it always returns `None`.

Los siguientes métodos implementan una interfaz parecida a un mapeo para acceder a los encabezados RFC 2822 [https://datatracker.ietf.org/doc/html/rfc2822.html] del mensaje. Ten en cuenta que hay algunas diferencias semánticas entre esos métodos y una interfaz de mapeo normal (ej. diccionario). Por ejemplo, en un diccionario no hay claves duplicadas, pero aquí pueden haber encabezados de mensaje duplicados. También, en diccionarios no hay un orden garantizado de las claves retornadas por `keys()`, pero en un objeto `Message`, los encabezados siempre son retornados en el orden que aparecieron en el mensaje original, o en el que fueron añadidos al mensaje más tarde. Cualquier encabezado eliminado y vuelto a adicionar siempre es añadido al final de la lista de encabezados.

Esas diferencias semánticas son intencionales y están sesgadas hacia la máxima comodidad.

Ten en cuenta que en todos los casos, cualquier encabezado de envoltura presente en el mensaje no está incluido en la interfaz de mapeo.

En un modelo generado desde bytes, cualesquiera valores de encabezado que (en contravención de los RFCs) contienen bytes no ASCII serán representados, cuando sean obtenidos mediante esta interfaz, como objetos **`Header`** con un conjunto de caracteres `unknown-8bit`.

len()

Retorna el número total de encabezados, incluyendo duplicados.

contains(*name*)

Retorna `True` si el objeto mensaje tiene un campo llamado *name*. La concordancia se realiza sin distinguir mayúsculas de minúsculas y *name* no debería incluir el caracter de doble punto final. Usado para el operador `in`, ej:

```
if 'message-id' in myMessage:
   print('Message-ID:', myMessage['message-id']
```

getitem(*name*)

Retorna el valor del campo del encabezado nombrado. *name* no debe incluir el separador de campo de doble punto. Si falta un encabezado, se retorna `None`; nunca se genera un error **`KeyError`**.

Ten en cuenta que si el campo nombrado aparece más de una vez en los encabezados del mensaje, no se define cuales serán exactamente aquellos valores de campos retornados. Usa el método **`get_all()`** para obtener los valores de todos los encabezados nombrados existentes.

setitem(*name*, *val*)

Añade un encabezado al mensaje con el nombre de campo *name* y el valor *val*. El campo es añadido al final de los campos existentes del mensaje.

Ten en cuenta que esto no sobreescribe ni elimina ningún encabezado existente con el mismo nombre. Si quieres asegurar que el nuevo encabezado es el único presente en el mensaje con el nombre de campo *name*, elimina el campo primero, ej:

```
del msg['subject']
msg['subject'] = 'Python roolz!'
```

delitem(*name*)

Elimina todas las ocurrencias de un campo con el nombre *name* de los encabezados del mensaje. No se genera ninguna excepción si el encabezado nombrado no está presente en los encabezados.

keys()

Retorna una lista de todos los nombres de campos de encabezados del mensaje.

values()

Retorna una lista de todos los valores de campos del mensaje.

items()

Retorna una lista de tuplas de dos elementos conteniendo todos los campos y valores de encabezados del mensaje.

get(*name*, *failobj=None*)

Return the value of the named header field. This is identical to **`__getitem__()`** except that optional *failobj* is returned if the named header is missing (defaults to `None`).

Aquí hay algunos métodos útiles adicionales:

get_all(*name*, *failobj=None*)

Retorna una lista de todos los valores para el campo denominado *name*. Si no hay tales encabezados nombrados en el mensaje, retorna *failobj* (por defecto `None`).

add_header(*_name*, *_value*, ***_params*)

Configuración de encabezado extendida. Este método es similar a **`__setitem__()`** excepto que pueden ser provistos parámetros adicionales de encabezado como argumentos de palabra clave. *_name* es el campo de encabezado a añadir y *_value* es el valor *primario* para el encabezado.

Para cada elemento en el diccionario de argumentos de palabra clave *_params*, la clave se toma como el nombre del parámetro con guiones bajos convertidos a guiones medios (ya que los guiones medios son ilegales como identificadores en Python). Normalmente, el parámetro será añadido como `key="value"` a no ser que el valor sea `None`, en cuyo caso sólo la clave será añadida. Si el valor contiene caracteres no ASCII, puede ser especificado como una tupla de tres elementos en el formato `(CHARSET, LANGUAGE, VALUE)`, donde `CHARSET` es una cadena que nombra el conjunto de caracteres a ser usado al codificar el valor, `LANGUAGE` puede normalmente ser definido a `None` o una cadena vacía (ver **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html] para otras posibilidades) y `VALUE` es la cadena del valor conteniendo puntos de caracteres no ASCII. Si no se pasa una tupla de tres elementos y el valor contiene caracteres no ASCII, se codifica automáticamente en formato **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html] usando `CHARSET` como `utf-8` y `LANGUAGE` como `None`.

Aquí hay un ejemplo:

```
msg.add_header('Content-Disposition', 'attachme
```

Esto añadirá un encabezado que se verá como

```
Content-Disposition: attachment; filename="bud.
```

Un ejemplo con caracteres no ASCII:

```
msg.add_header('Content-Disposition', 'attachme
               filename=('iso-8859-1', '', 'Fu
```

Lo que produce

```
Content-Disposition: attachment; filename*="isc
```

replace_header(_*name*, _*value*)

Reemplaza un encabezado. Reemplaza el primer encabezado encontrado en el mensaje que concuerda con _*name*, conservando el orden del encabezado y el nombre del campo. Si no se encuentra ningún encabezado que concuerde, se genera un error **`KeyError`**.

get_content_type()

Retorna el tipo de contenido del mensaje. La cadena retornada se fuerza a letras minúsculas de la forma *maintype/subtype*. Si no hay ningún encabezado *Content-Type* en el mensaje será retornado el tipo por defecto como es dado por **`get_default_type()`**. Dado que según **RFC 2045** [https://datatracker.ietf.org/doc/html/rfc2045.html], los mensajes tienen siempre un tipo predeterminado, **`get_content_type()`** siempre retornará un valor.

RFC 2045 [https://datatracker.ietf.org/doc/html/rfc2045.html] define el tipo predeterminado del mensaje a *text/plain* a no ser que aparezca dentro de un contenedor *multipart/digest*, en cuyo caso sería *message/rfc822*. Si el encabezado *Content-Type* tiene una especificación de tipo inválido, **RFC 2045** [https://datatracker.ietf.org/doc/

html/rfc2045.html] ordena que el tipo por defecto sea *text/plain*.

get_content_maintype()

Retorna el tipo de contenido principal del mensaje. Esta es la parte *maintype* de la cadena retornada por `get_content_type()`.

get_content_subtype()

Retorna el tipo del subcontenido del mensaje. Esta es la parte *subtype* de la cadena retornada por `get_content_type()`.

get_default_type()

Retorna el tipo del contenido por defecto. La mayoría de mensajes tienen un tipo de contenido por defecto de *text/plain*, excepto para mensajes que son subpartes de contenedores *multipart/digest*. Tales subpartes tienen como tipo de contenido predeterminado *message/rfc822*.

set_default_type(*ctype*)

Establece el tipo de contenido por defecto. *ctype* debería ser *text/plain* o *message/rfc822*, aunque esto no es obligatorio. El tipo de contenido predeterminado no se almacena en el encabezado *Content-Type*.

get_params(*failobj=None*, *header='content-type'*, *unquote=True*)

Retorna los parámetros del *Content-Type* del mensaje como una lista. Los elementos de la lista retornada son tuplas de dos elementos de pares clave/valor, tal y como son partidas por el signo `'='`. El lado izquierdo del `'='` es la clave, mientras el lado derecho es el valor. Si no hay signo `'='` en el parámetro, el valor es la cadena vacía, en caso contrario el valor es como se describe en `get_param()` y no está citado si el parámetro opcional *unquote* es `True` (por defecto).

El parámetro opcional *failobj* es el objeto a retornar si no hay encabezado *Content-Type*. El parámetro opcional *header* es el encabezado a buscar en lugar de *Content-Type*.

This is a legacy method. On the **`EmailMessage`** class its functionality is replaced by the *params* property of the individual header objects returned by the header access methods.

get_param(*param*, *failobj=None*, *header='content-type'*, *unquote=True*)

Retorna el valor del parámetro *param* del encabezado *Content-Type* como una cadena. Si el mensaje no tiene encabezado *Content-Type* o si no existe tal parámetro, entonces se retorna *failobj* (por defecto es `None`).

El parámetro opcional *header*, si es definido, especifica el encabezado del mensaje a usar en lugar de *Content-Type*.

Las claves de parámetros siempre son comparadas distinguiendo entre mayúsculas y minúsculas. El valor de retorno puede ser una cadena, una tupla de 3 elementos si el parámetro fue codificado según **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html]. Cuando es una tupla de 3 elementos, los elementos del valor tienen la forma `(CHARSET, LANGUAGE, VALUE)`. Ten en cuenta que `CHARSET` y `LANGUAGE` pueden ser `None`, en cuyo caso debes considerar `VALUE` como codificado en el conjunto de caracteres `us-ascii`. Generalmente puedes ignorar `LANGUAGE`.

Si a tu aplicación no le importa si el parámetro fue codificado según **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html], puedes contraer el valor del parámetro invocando **`email.utils.collapse_rfc2231_value()`**, pasando el valor de retorno desde **`get_param()`**. Esto retornará una cadena Unicode convenientemente decodificada cuando el valor es una tupla o la cadena original sin entrecomillar si no lo es. Por ejemplo:

```
rawparam = msg.get_param('foo')
param = email.utils.collapse_rfc2231_value(rawp
```

En cualquier caso, el valor del parámetro (tanto la cadena retornada o el elemento `VALUE` en la tupla de 3 elementos) siempre está sin entrecomillar, a no ser que *unquote* está establecido a `False`.

This is a legacy method. On the **`EmailMessage`** class its functionality is replaced by the *params* property of the individual header objects returned by the header access methods.

set_param(*param*, *value*, *header='Content-Type'*, *requote=True*, *charset=None*, *language=''*, *replace=False*)

Establece un parámetro en el encabezado *Content-Type*. Si el parámetro ya existe en el encabezado, su valor será remplazado con *value*. Si el encabezado *Content-Type* no ha sido definido todavía para este mensaje, será establecido a *text/plain* y el nuevo valor del parámetro será añadido según **RFC 2045** [https://datatracker.ietf.org/doc/html/rfc2045.html].

El parámetro opcional *header* especifica una alternativa a *Content-Type* y todos los parámetros serán entrecomillados si es necesario a no ser que el parámetro opcional *requote* sea `False` (por defecto es `True`).

Si se especifica el parámetro opcional *charset*, el parámetro será codificado de acuerdo a **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html]. El parámetro opcional *language* especifica el lenguaje RFC 2231, por defecto una cadena vacía. Tanto *charset* como *language* deberían ser cadenas.

Si *replace* es `False` (por defecto) el encabezado será movido al final de la lista de encabezados. Si *replace* es `True`, el encabezado será actualizado.

Distinto en la versión 3.4: el parámetro de palabra clave `replace` fue añadido.

del_param(*param, header='content-type', requote=True*)

Elimina el parámetro dado completamente del encabezado *Content-Type*. El encabezado será reescrito en sí mismo sin el parámetro o su valor. Todos los valores serán entrecomillados si es necesario a no ser que *requote* sea `False` (por defecto es `True`). El parámetro opcional *header* especifica una alternativa a *Content-Type*.

set_type(*type, header='Content-Type', requote=True*)

Establece el tipo y subtipo principal para el encabezado *Content-Type*. *type* debe ser una cadena de la forma *maintype/subtype*, si no será generado un **`ValueError`**.

Este método remplaza el encabezado *Content-Type*, manteniendo todos los parámetros en su lugar. Si *requote* es `False`, este dejará el encabezado existente tal como está, en caso contrario los parámetros serán entrecomillados (por defecto).

Un encabezado alternativo puede ser especificado en el argumento *header*. Cuando el encabezado *Content-Type* es definido, un encabezado *MIME-Version* también es añadido.

This is a legacy method. On the **`EmailMessage`** class its functionality is replaced by the `make_` and `add_` methods.

get_filename(*failobj=None*)

Retorna el valor del parámetro `filename` del encabezado *Content-Disposition* del mensaje. Si el encabezado no tiene un parámetro `filename`, este método recurre a buscar el parámetro `name` en el encabezado *Content-Type*. Si tampoco se encuentra o falta el encabezado, entonces retorna *failobj*. La cadena retornada siempre será sin entrecomillar según **`email.utils.unquote()`**.

get_boundary(*failobj=None*)

Retorna el valor del parámetro `boundary` del encabezado *Content-Type* del mensaje o *failobj* tanto si falta el encabezado como si no tiene parámetro `boundary`. La cadena retornada siempre será sin entrecomillar según **`email.utils.unquote()`**.

set_boundary(*boundary*)

Establece el parámetro `boundary` del encabezado *Content-Type* a *boundary*. **`set_boundary()`** siempre entrecomillará *boundary* si es necesario. Se genera **`HeaderParseError`** si el objeto de mensaje no tiene encabezado *Content-Type*.

Ten en cuenta que usar este método es sutilmente diferente a borrar el antiguo encabezado *Content-Type* y añadir uno nuevo con el nuevo límite mediante **`add_header()`** porque **`set_boundary()`** preserva el orden del encabezado *Content-Type* en la lista de encabezados. Sin embargo, no conserva ninguna línea de continuación que pueden haber estado presentes en el encabezado original *Content-Type*.

get_content_charset(*failobj=None*)

Retorna el parámetro `charset` del encabezado *Content-Type*, forzado a letras minúsculas. Si no hay un encabezado *Content-Type* o si ese encabezado no tiene parámetro `charset`, se retorna *failobj*.

Ten en cuenta que este método difiere de **`get_charset()`**, el cual retorna la instancia **`Charset`** para la codificación por defecto del cuerpo del mensaje.

get_charsets(*failobj=None*)

Retorna una lista conteniendo los nombres de los conjuntos de caracteres en el mensaje. Si el mensaje es *multipart*, entonces la lista contendrá un elemento para cada subparte en la carga (*payload*), en caso contrario será una lista de un elemento.

Cada elemento en la lista será una cadena la cual es el valor del parámetro `charset` en el encabezado *Content-Type* para la subparte representada. Sin embargo, si la subparte no tiene encabezado *Content-Type*, no tiene parámetro `charset` o no es del tipo MIME *text* principal, entonces ese elemento en la lista retornada será *failobj*.

get_content_disposition()

Retorna el valor en minúsculas (sin parámetros) del encabezado del mensaje *Content-Disposition* si tiene uno o `None`. Los valores posibles para este método son *inline*, *attachment* o `None` si el mensaje sigue el **RFC 2183** [https://datatracker.ietf.org/doc/html/rfc2183.html].

Added in version 3.5.

walk()

El método **`walk()`** es un generador de todo propósito el cual puede ser usado para iterar sobre todas las partes y subpartes de árbol de objeto de mensaje, en orden de recorrido de profundidad primero. Siempre usarás típicamente **`walk()`** como iterador en un bucle `for`; cada iteración retorna la siguiente subparte.

Aquí hay un ejemplo que imprime el tipo MIME de cada parte de una estructura de mensaje multiparte:

```
>>> for part in msg.walk():
...     print(part.get_content_type())
```

```
multipart/report
text/plain
message/delivery-status
text/plain
text/plain
message/rfc822
text/plain
```

`walk` itera sobre las subpartes de cualquier parte donde **`is_multipart()`** retorna `True`, aunque `msg.get_content_maintype() == 'multipart'` puede retornar `False`. Vemos esto en nuestro ejemplo haciendo uso de la función de ayuda de depuración `_structure`:

```
>>> for part in msg.walk():
...     print(part.get_content_maintype() == '
...           part.is_multipart())
True True
False False
False True
False False
False False
False True
False False
>>> _structure(msg)
multipart/report
    text/plain
    message/delivery-status
        text/plain
        text/plain
    message/rfc822
        text/plain
```

Aquí las partes de `message` no son `multiparts`, pero contienen subpartes. `is_multipart()` retorna `True` y `walk` desciende a las subpartes.

Los objetos **`Message`** pueden contener opcionalmente dos atributos de instancia, los cuales pueden ser usados al generar el texto plano de un mensaje MIME.

preamble

El formato de un documento MIME permite algo de texto entre la línea en blanco que sigue a los encabezados y la primera cadena límite multiparte. Normalmente, este texto nunca es visible en un lector de correo compatible con MIME porque queda fuera de la armadura MIME estándar. Sin embargo, viendo el texto del mensaje en crudo o viendo el mensaje en un lector no compatible con MIME, este texto puede volverse visible.

El atributo *preamble* contiene este texto de refuerzo adicional para documentos MIME. Cuando el `Parser` descubre algo de texto después de los encabezados pero antes de la primera cadena límite, asigna este texto al atributo *preamble* del mensaje. Cuando el `Generator` está escribiendo la representación de texto sin formato de un mensaje MIME y puede encontrar el mensaje como un atributo *preamble*, escribirá este texto en el área entre los encabezados y el primer límite. Consulta `email.parser` y `email.generator` para más detalles.

Ten en cuenta que si el objeto de mensaje no tiene preámbulo, el atributo *preamble* será `None`.

epilogue

El atributo *epilogue* actúa de la misma manera que el atributo *preamble*, excepto que contiene texto que aparece entre el último límite y el fin del mensaje.

No necesitas establecer el epílogo de la cadena vacía en orden para el `Generator` para imprimir una nueva línea al final del archivo.

defects

El atributo *defects* contiene una lista de todos los problemas encontrados al analizar este mensaje. Consulta `email.errors` para una descripción detallada de los posibles defectos de análisis.

`email.mime`: Creating email and MIME objects from scratch

Código fuente: Lib/email/mime/ [https://github.com/python/cpython/tree/3.14/Lib/email/mime/]

Este módulo forma parte de la API heredada de correo electrónico (`Compat32`). Su funcionalidad se sustituye parcialmente por el **`contentmanager`** en la nueva API, pero en ciertas aplicaciones estas clases pueden seguir siendo útiles, incluso en código no heredado.

Normalmente, se obtiene una estructura de objeto de mensaje pasando un archivo o algún texto a un analizador, que analiza el texto y retorna el objeto de mensaje principal. Sin embargo, también puede crear una estructura de mensajes completa desde cero, o incluso objetos individuales **`Message`** a mano. De hecho, también puede tomar una estructura existente y agregar nuevos objetos **`Message`**, moverlos, etc. Esto compone una interfaz muy conveniente para cortar y gestionar mensajes MIME.

Puede crear una nueva estructura de objeto mediante la creación de una instancia de **`Message`**, añadiendo accesorios y todos los encabezados necesarios manualmente. Para mensajes MIME sin embargo, el paquete **`email`** proporciona subclases convenientes para facilitar las cosas.

Descripción de las clases:

class email.mime.base.MIMEBase(*_maintype*, *_subtype*, ***, *policy=compat32*, ***_params*)

Módulo: **`email.mime.base`**

Esta es la clase básica para todas las subclases específicas

`Message` para MIME. Normalmente no creará instancias específicas de la `MIMEBase`, aunque es posible. La **`MIMEBase`** se proporciona principalmente como una clase básica conveniente para subclases más específicas, apropiadas para MIME.

_maintype es el tipo principal *Content-Type* (por ejemplo *text* o *image*), y *_subtype* es el tipo menor *Content-Type* (por ejemplo *plain* o *gif*). *_params* es un diccionario de parámetro clave/valor y se pasa directamente a `Message.add_header`.

Si se especifica *policy*, (por defecto, la directiva `compat32`) se pasará a `Message`.

La clase `MIMEBase` siempre agrega un encabezado *Content-Type* (basado en *_maintype*, *_subtype* y *_params*), y un encabezado *MIME-Versión* (siempre establecido en `1.0`).

Distinto en la versión 3.6: Se ha añadido el parámetro *policy* de solo palabra clave.

class email.mime.nonmultipart.MIMENonMultipart

Módulo: `email.mime.nonmultipart`

Una subclase de `MIMEBase`, es una clase base intermedia para los mensajes MIME que no son *multipart*. El propósito principal de esta clase es evitar el uso del método `attach()`, que solo tiene sentido para los mensajes *multipart*. Si se llama a **`attach()`**, se lanza una excepción `MultipartConversionError`.

class email.mime.multipart.MIMEMultipart(*_subtype='mixed', boundary=None, _subparts=None, *, policy=compat32, **_params*)

Módulo: `email.mime.multipart`

Una subclase de `MIMEBase`, se trata de una clase base intermedia para los mensajes MIME que son *multipart*. El valor predeterminado opcional de *_subtype* es *mixed*, pero se puede utilizar para especificar el subtipo del mensaje. Se agregará un encabezado *Content-Type* de *multipart/_subtype* al objeto del mensaje. También se agregará un encabezado *MIME-Version*.

El *boundary* opcional es la cadena de límite multiparte. Cuando `None` (valor predeterminado), el límite se calcula cuando es necesario (por ejemplo, cuando se serializa el mensaje).

_subparts es una secuencia de subpartes iniciales para la carga útil. Debe ser posible convertir esta secuencia en una lista. Siempre puede adjuntar nuevas subpartes al mensaje mediante el método **`Message.attach`**.

El valor predeterminado del argumento *policy* opcional es **`compat32`**.

Los parámetros adicionales para el encabezado *Content-Type* se toman de los argumentos de palabra clave, o se pasan al argumento *_params*, que es un diccionario de palabras clave.

Distinto en la versión 3.6: Se ha añadido el parámetro *policy* de solo palabra clave.

class email.mime.application.MIMEApplication(*_data*, *_subtype='octet-stream'*, *_encoder=email.encoders.encode_base64*, ***, *policy=compat32*, ***_params*)

Módulo: **`email.mime.application`**

Una subclase de **`MIMENonMultipart`**, la clase **`MIMEApplication`** se utiliza para representar objetos de mensaje MIME de tipo principal *application*. *_data* contiene los bytes de la aplicación sin procesar. *_subtype* opcional especifica el subtipo MIME y el valor predeterminado es *octet-stream*.

_encoder opcional es una recurso invocable (es decir, una función) que realizará la codificación real de los datos para el transporte. Este invocable toma un argumento, que es la instancia **`MIMEApplication`**. Debe utilizar **`get_payload()`** y **`set_payload()`** para cambiar la carga útil a la forma codificada. También debe agregar cualquier *Content-Transfer-Encoding* u otros encabezados al objeto del mensaje según sea necesario. La codificación predeterminada es base64. Consulte el módulo **`email.encoders`** para obtener una lista de los codificadores integrados.

El valor predeterminado del argumento *policy* opcional es **`compat32`**.

_params se pasan directamente al constructor de la clase base.

Distinto en la versión 3.6: Se ha añadido el parámetro *policy* de solo palabra clave.

class email.mime.audio.MIMEAudio(*_audiodata*, *_subtype=None*, *_encoder=email.encoders.encode_base64*, ***, *policy=compat32*, ***_params*)

Módulo: **`email.mime.audio`**

Una subclase de **`MIMENonMultipart`**, la clase **`MIMEAudio`** se utiliza para crear objetos de mensaje MIME de tipo principal *audio*. *_audiodata* contiene los bytes del audio sin procesar. Si estos datos pueden ser decodificados como au, wav, aiff, or aifc, entonces el subtipo se incluirá automáticamente en el encabezado *Content-Type*. De lo contrario, puede especificar explícitamente el subtipo de audio mediante el argumento *_subtype*. Si no se pudo adivinar el tipo secundario y no se ha proporcionado *_subtype*, se lanza **`TypeError`**.

_encoder opcional es un recurso invocable (es decir, una función) que realizará la codificación real de los datos de audio para el transporte. Este invocable toma un argumento, que es la instancia **`MIMEAudio`**. Debe utilizar **`get_payload()`** y **`set_payload()`** para cambiar la carga útil a la forma codificada. También debe agregar cualquier *Content-Transfer-Encoding* u otros encabezados al objeto del mensaje según sea necesario. La codificación predeterminada es base64. Consulte el módulo **`email.encoders`** para obtener una lista de los codificadores integrados.

El valor predeterminado del argumento *policy* opcional es `compat32`.

_params se pasan directamente al constructor de la clase base.

Distinto en la versión 3.6: Se ha añadido el parámetro *policy* de solo palabra clave.

class email.mime.image.MIMEImage(*_imagedata*, *_subtype=None*, *_encoder=email.encoders.encode_base64*, ***, *policy=compat32*, ***_params*)

Módulo: `email.mime.image`

Una subclase de `MIMENonMultipart`, la clase `MIMEImage` se utiliza para crear objetos de mensaje MIME de tipo principal *image*. *_imagedata* contiene los bytes de la imagen sin procesar. Si se puede detectar el tipo de dato (intentando con jpeg, png, gif, tiff, rgb, pbm, pgm, ppm, rast, xbm, bmp, webp, y exr), entonces el subtipo se incluirá automáticamente en el encabezado *Content-Type*. De lo contrario, puede especificar explícitamente el subtipo de imagen mediante el argumento *_subtype*. Si no se pudo adivinar el tipo secundario y no se ha proporcionado *_subtype*, se lanza `TypeError`.

_encoder opcional es un recurso invocable (es decir, una función) que realizará la codificación real de los datos de imagen para el transporte. Este invocable toma un argumento, que es la instancia `MIMEImage`. Debe utilizar `get_payload()` y `set_payload()` para cambiar la carga útil a la forma codificada. También debe agregar cualquier *Content-Transfer-Encoding* u otros encabezados al objeto del mensaje según sea necesario. La codificación predeterminada es base64. Consulte el módulo `email.encoders` para obtener una lista de los codificadores integrados.

El valor predeterminado del argumento *policy* opcional es `compat32`.

_params se pasan directamente al constructor `MIMEBase`.

Distinto en la versión 3.6: Se ha añadido el parámetro *policy* de solo palabra clave.

class email.mime.message.MIMEMessage(*_msg*, *_subtype='rfc822'*, ***, *policy=compat32*)

Módulo: **`email.mime.message`**

Una subclase de **`MIMENonMultipart`**, la clase **`MIMEMessage`** se utiliza para crear objetos MIME de tipo principal *message*. *_msg* se utiliza como la carga y debe ser una instancia de la clase **`Message`** (o una subclase de la misma), de lo contrario se lanza un **`TypeError`**.

_subtype opcional establece el subtipo del mensaje; por defecto es *rfc822*.

El valor predeterminado del argumento *policy* opcional es **`compat32`**.

Distinto en la versión 3.6: Se ha añadido el parámetro *policy* de solo palabra clave.

class email.mime.text.MIMEText(*_text*, *_subtype='plain'*, *_charset=None*, ***, *policy=compat32*)

Módulo: **`email.mime.text`**

Una subclase de **`MIMENonMultipart`**, la clase **`MIMEText`** se utiliza para crear objetos MIME de tipo principal *text*. *_text* es la cadena de caracteres de la carga útil. *_subtype* es el tipo menor y el valor predeterminado es *plain*. *_charset* es el paquete de caracteres del texto y se pasa como argumento al constructor **`MIMENonMultipart`**; el valor predeterminado es `us-ascii` si la cadena contiene sólo puntos de código `ascii`, y `utf-8` en caso contrario. El parámetro *_charset* acepta una cadena o una instancia **`Charset`**.

A menos que el argumento *_charset* se establezca explícitamente como `None`, el objeto MIMEText creado tendrá a la vez un encabezado *Content-Type* con un parámetro `charset`, y un encabezado *Content-Transfer-Encoding*. Esto significa que una llamada posterior a `set_payload` no dará lugar a una carga codificada, incluso si se pasa un conjunto de caracteres en el comando `set_payload`. Puede «restablecer» este comportamiento eliminando el encabezado `Content-Transfer-Encoding`, después de lo cual una llamada a `set_payload` codificará automáticamente la nueva carga útil (y agregará un nuevo encabezado *Content-Transfer-Encoding*).

El valor predeterminado del argumento *policy* opcional es **`compat32`**.

Distinto en la versión 3.5: _charset también acepta instancias **`Charset`**.

Distinto en la versión 3.6: Se ha añadido el parámetro *policy* de solo palabra clave.

`email.header`: Internationalized headers

Código fuente: Lib/email/header.py [https://github.com/python/cpython/tree/3.14/Lib/email/header.py]

Este módulo es parte de la API de email heredada (`Compat32`). En la API actual, la codificación y decodificación de las cabeceras se gestiona de forma transparente por la API de tipo diccionario de la clase **`EmailMessage`**. Además de los usos del código heredado, este módulo puede ser útil en aplicaciones que necesiten controlar completamente el conjunto de caracteres usado cuando se codifican las cabeceras.

El resto del texto de esta sección es la documentación original del módulo.

RFC 2822 [https://datatracker.ietf.org/doc/html/rfc2822.html] es el estándar base que describe el formato de los mensajes de correo electrónico. Deriva del estándar anterior **RFC 822** [https://datatracker.ietf.org/doc/html/rfc822.html], cuyo uso se generalizó durante una época en la que la mayoría del correo electrónico se componía únicamente de caracteres ASCII. **RFC 2822** [https://datatracker.ietf.org/doc/html/rfc2822.html] es una especificación que se escribió asumiendo que el correo electrónico contiene solo caracteres ASCII 7-bit.

Of course, as email has been deployed worldwide, it has become internationalized, such that language specific character sets can now be used in email messages. The base standard still requires email messages to be transferred using only 7-bit ASCII characters, so a slew of RFCs have been written describing how to encode email containing non-ASCII characters into **RFC 2822** [https://datatracker.ietf.org/doc/html/rfc2822.html]-compliant format. These RFCs include **RFC 2045** [https://datatracker.ietf.org/doc/html/rfc2045.html], **RFC 2046** [https://datatracker.ietf.org/doc/html/rfc2046.html], **RFC 2047** [https://

datatracker.ietf.org/doc/html/rfc2047.html], and **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html]. The `email` package supports these standards in its `email.header` and `email.charset` modules.

If you want to include non-ASCII characters in your email headers, say in the *Subject* or *To* fields, you should use the `Header` class and assign the field in the `Message` object to an instance of `Header` instead of using a string for the header value. Import the `Header` class from the `email.header` module. For example:

```
>>> from email.message import Message
>>> from email.header import Header
>>> msg = Message()
>>> h = Header('p\xf6stal', 'iso-8859-1')
>>> msg['Subject'] = h
>>> msg.as_string()
'Subject: =?iso-8859-1?q?p=F6stal?=\n\n'
```

¿Has visto cómo hemos hecho que el campo *Subject* contuviera un caracter no ASCII? Lo hemos hecho creando una instancia de `Header` y pasándole el conjunto de caracteres en los que estaba codificado la cadena de bytes. Cuando la instancia de `Message` subsecuente se ha aplanado, el campo *Subject* se ha codificado en **RFC 2047** [https://datatracker.ietf.org/doc/html/rfc2047.html] adecuadamente. Los lectores de correo que soportan MIME deberían mostrar esta cabecera usando el caracter ISO-8859-1 incrustado.

Aquí está la descripción de la clase `Header`:

class email.header.Header(*s=None*, *charset=None*, *maxlinelen=None*, *header_name=None*, *continuation_ws=' '*, *errors='strict'*)

> Crea una cabecera conforme a las especificaciones MIME que contiene cadenas de caracteres en diferentes conjuntos de caracteres.
>
> El argumento opcional *s* es el valor inicial de la cabecera. Si es `None` (el valor por defecto), el valor inicial de la cabecera quedará sin asignar. Puedes añadirlo luego a la cabecera con

llamadas al método `append()`. *s* debe ser una instancia de `bytes` o `str`, pero lee la documentación de **`append()`** para conocer los detalles semánticos.

El argumento opcional *charset* sirve para dos propósitos: tiene el mismo significado que el argumento *charset* en el método `append()`. También asigna el conjunto de caracteres por defecto para todas las llamadas subsecuentes a **`append()`** que omitan el argumento *charset*. Si no se proporciona *charset* en el constructor (por defecto), el conjunto de caracteres `us-ascii` se usa tanto para el conjunto de caracteres inicial de *s* como por defecto para las llamadas subsecuentes a **`append()`**.

The maximum line length can be specified explicitly via *maxlinelen*. For splitting the first line to a shorter value (to account for the field header which isn't included in *s*, e.g. *Subject*) pass in the name of the field in *header_name*. The default *maxlinelen* is 78, and the default value for *header_name* is `None`, meaning it is not taken into account for the first line of a long, split header.

El argumento opcional *continuation_ws* debe ser conforme a las normas de espacios en blanco plegables de la RFC 2822 [https://datatracker.ietf.org/doc/html/rfc2822.html], y normalmente es o un espacio o un caracter tabulador. Este caracter se antepondrá delante de las líneas continuadas. El valor por defecto de *continuation_ws* es un solo espacio.

El argumento opcional *errors* se pasa directamente a través del método `append()`.

append(*s*, *charset=None*, *errors='strict'*)

> Añade la cadena de caracteres *s* a la cabecera MIME.
>
> El argumento opcional *charset*, si se proporciona, debe ser una instancia de `Charset` (ver `email.charset`) o el nombre de un conjunto de datos, que deberá ser convertido a una instancia de **`Charset`**. Un valor de `None` (el valor por defecto) significa que se usará el *charset* dado en el constructor.

s puede ser una instancia de `bytes` o de `str`. Si es una instancia de `bytes`, entonces *charset* es la codificación de esa cadena de bytes, y se lanzará un `UnicodeError` si la cadena de caracteres no puede decodificarse con ese conjunto de caracteres.

Si *s* es una instancia de `str`, entonces *charset* es una sugerencia que especifica el conjunto de caracteres usando en la cadena de caracteres.

En cualquier caso, cuando se produce una cabecera conforme a la **RFC 2822** [https://datatracker.ietf.org/doc/html/rfc2822.html] usando las reglas de la **RFC 2047** [https://datatracker.ietf.org/doc/html/rfc2047.html], la cadena de caracteres se codificará usando el códec de salida del conjunto de caracteres. Si la cadena de caracteres no puede codificarse usando el códec de salida, se lanzará un UnicodeError.

El argumento opcional *errors* se pasa como el argumento de errores para la llamada de decodificación si *s* es una cadena de bits.

encode(*splitchars=';, \t', maxlinelen=None, linesep='\n'*)

Codifica un mensaje de la cabecera en un formato conforme a RFC, posiblemente envolviendo las líneas largas y encapsulando las partes no ASCII en base64 o en codificaciones imprimibles entrecomilladas.

El argumento opcional *splitchars* es una cadena de caracteres que contiene caracteres a los que el algoritmo de separación debería asignar espacio extra durante la encapsulación de la cabecera normal. Esto da un basto soporte a los saltos sintácticos de alto nivel de la **RFC 2822** [https://datatracker.ietf.org/doc/html/rfc2822.html]: los puntos de separación precedidos por un caracter separador tienen preferencia durante la separación de la línea, con preferencia de caracteres en el orden en el que aparecen en la cadena de caracteres. El espacio y el tabulador pueden incluirse en la cadena de caracteres para indicar si se debería dar preferencia a uno sobre el otro como punto de separación cuando no aparezcan otros caracteres separadores en la línea que se está dividiendo. Los caracteres separadores no afectan a las líneas codificadas de la **RFC 2047** [https://datatracker.ietf.org/doc/html/rfc2047.html].

maxlinelen, si se proporciona, sobrescribe el valor de longitud de línea máxima para la instancia.

linesep especifica los caracteres usados para separar las líneas de la cabecera plegada. Su valor por defecto es el valor más útil para el código de una aplicación Python (`\n`), pero se puede especificar `\r\n` para producir cabeceras con separadores de línea conforme a RFCs.

Distinto en la versión 3.2: Argumento *linesep* añadido.

La clase **`Header`** también proporciona una serie de métodos para soportar operaciones estándar y funciones incorporadas.

__str__()

Retorna una aproximación de **`Header`** como una cadena de caracteres, usando una longitud de línea ilimitada. Todas las piezas se convierten a unicode utilizando la codificación especificada y unidas adecuadamente. Todas las piezas con un conjunto de caracteres `'unknown-8bit'` se decodifican como ASCII usando el gestor de errores de `'replace'`.

Distinto en la versión 3.2: Añadida gestión del conjunto de caracteres `'unknown-8bit'`.

__eq__(*other*)

Este método permite comparar si dos instancias de **`Header`** son iguales.

__ne__(*other*)

Este método permite comparar si dos instancias de

`Header` no son iguales.

The `email.header` module also provides the following convenient functions.

email.header.decode_header(*header*)

Decodifica el valor de un mensaje de la cabecera sin convertir el conjunto de caracteres. El valor de la cabecera está en *header*.

For historical reasons, this function may return either:

1. A list of pairs containing each of the decoded parts of the header, `(decoded_bytes, charset)`, where *decoded_bytes* is always an instance of `bytes`, and *charset* is either:

    - A lower case string containing the name of the character set specified.
    - `None` for non-encoded parts of the header.

4. A list of length 1 containing a pair `(string, None)`, where *string* is always an instance of `str`.

An `email.errors.HeaderParseError` may be raised when certain decoding errors occur (e.g. a base64 decoding exception).

Here are examples:

```
>>> from email.header import decode_header
>>> decode_header('=?iso-8859-1?q?p=F6stal?=')
[(b'p\xf6stal', 'iso-8859-1')]
>>> decode_header('unencoded_string')
[('unencoded_string', None)]
>>> decode_header('bar =?utf-8?B?ZsOzbw==?=')
[(b'bar ', None), (b'f\xc3\xb3o', 'utf-8')]
```

Nota

This function exists for backwards compatibility only. For new code, we recommend using `email.headerregistry.HeaderRegistry`.

email.header.make_header(*decoded_seq*, *maxlinelen=None*, *header_name=None*, *continuation_ws=' '*)

Crea una instancia de `Header` a partir de una secuencia de duplas como las retornadas por `decode_header()`.

`decode_header()` toma el valor de una cadena de caracteres de la cabecera y retorna una secuencia de duplas con el formato `(decoded_string, charset)`, donde *charset* es el nombre del conjunto de caracteres.

Esta función toma una de esas secuencias de duplas y retorna una instancia de `Header`. Los argumentos opcionales *maxlinelen*, *header_name* y *continuation_ws* son como los del constructor `Header`.

Nota

This function exists for backwards compatibility only, and is not recommended for use in new code.

`email.charset`: Representing character sets

Código fuente: Lib/email/charset.py [https://github.com/python/cpython/tree/3.14/Lib/email/charset.py]

Este módulo es parte de la API de email heredada (`Compat32`). En la nueva API solo se usa la tabla de alias.

El texto restante de esta sección corresponde a la documentación original del módulo.

Este módulo proporciona una clase **`Charset`** para representar conjuntos de caracteres y conversiones de conjuntos de caracteres en mensajes de correo electrónico, así como un registro de conjuntos de caracteres y varios métodos de conveniencia para manipular este registro. Las instancias de **`Charset`** se utilizan en varios otros módulos dentro del paquete **`email`**.

Import this class from the **`email.charset`** module.

class email.charset.Charset(*input_charset=DEFAULT_CHARSET*)

Asigna conjuntos de caracteres a sus propiedades de correo electrónico.

Esta clase proporciona información sobre los requisitos impuestos al correo electrónico para un conjunto de caracteres específico. También proporciona rutinas de conveniencia para convertir entre juegos de caracteres, dada la disponibilidad de los códecs aplicables. Dado un conjunto de caracteres, hará todo lo posible para proporcionar información sobre cómo utilizar ese conjunto de caracteres en un mensaje de correo electrónico de forma compatible con RFC.

Ciertos conjuntos de caracteres deben codificarse con quoted-printable o base64 cuando se usan en encabezados o cuerpos de correo electrónico. Ciertos conjuntos de caracteres deben convertirse directamente y no están permitidos en el correo electrónico.

Opcional *input_charset* es como se describe a continuación; siempre se convierte a minúsculas. Después de que el alias sea normalizado también se utiliza como una búsqueda en el registro de conjuntos de caracteres para averiguar la codificación del encabezado, codificación de cuerpo, y códec de conversión de salida que se usarán para el conjunto de caracteres. Por ejemplo, si *input_charset* es `iso-8859-1`, los encabezados y cuerpos se codificarán mediante quoted-printable y no es necesario ningún códec de conversión de salida. Si *input_charset* es `euc-jp`, los encabezados se codificarán con base64, los cuerpos no se codificarán, pero el texto de salida se convertirá del conjunto de caracteres `euc-jp` al conjunto de caracteres `iso-2022-jp`.

Las instancias **`Charset`** tienen los siguientes atributos de datos:

input_charset

> El conjunto de caracteres inicial especificado. Los alias comunes se convierten a sus nombres de correo electrónico *Official* (por ejemplo, `latin_1` se convierte a `iso-8859-1`). El valor predeterminado es `us-ascii` de 7 bits.

header_encoding

> Si el conjunto de caracteres debe codificarse antes de que pueda usarse en un encabezado de correo electrónico, este atributo se establecerá a `charset.QP` (para imprimible con comillas), `charset.BASE64` (para codificación base64), o `charset.SHORTEST` para la codificación más corta QP o BASE64. De lo contrario será `None`.

body_encoding

Igual que *header_encoding*, pero describe la codificación del cuerpo del mensaje de correo, que de hecho puede ser diferente a la codificación del encabezado. `charset.SHORTEST` no está permitido para *body_encoding*.

output_charset

Algunos conjuntos de caracteres deben ser convertidos antes de poder ser usados en cabeceras o cuerpos de correos. Si el *input_charset* es uno de ellos, este atributo contendrá el nombre del conjunto de caracteres al que será convertido. De lo contrario, será `None`.

input_codec

El nombre del códec de Python usado para convertir el *input_charset* a Unicode. Si no es necesario un códec de conversión, este atributo será `None`.

output_codec

El nombre del códec de Python usado para convertir Unicode a *output_charset*. Si no es necesario un códec de conversión, este atributo tendrá el mismo valor que *input_codec*.

Las instancias **`Charset`** además tienen los siguientes métodos:

get_body_encoding()

Retorna la codificación de transferencia de contenido usada para la codificación del cuerpo.

Esta es la cadena de caracteres `quoted-printable` o `base64` dependiendo de la codificación usada, o es una función, en cuyo caso se deberá llamar a la función con un solo argumento, el objeto Mensaje a ser codificado. La función deberá luego establecer el encabezado *Content-Transfer-Encoding* en lo que sea apropiado.

Retorna la cadena `quoted-printable` si

body_encoding es `QP`, retorna la cadena `base64` si *body_encoding* es `BASE64`, y retorna la cadena `7bit` en caso contrario.

get_output_charset()

Return the output character set.

Este es el atributo *output_charset* si no es `None`, en caso contrario es *input_charset*.

header_encode(*string*)

Codifica como encabezado la cadena de caracteres *string*.

El tipo de codificación (base64 o quoted-printable) se basará en el atributo *header_encoding*.

header_encode_lines(*string*, *maxlengths*)

Codifica como encabezado *string* convirtiéndolo primero a bytes.

Es similar a **`header_encode()`** excepto que la cadena se ajusta a las longitudes máximas indicadas en el argumento *maxlengths*, el cual debe ser un iterador: cada elemento retornado por este iterador proporcionará la siguiente longitud máxima de línea.

body_encode(*string*)

Codifica como cuerpo la cadena *string*.

El tipo de codificación (base64 o quoted-printable) se basará en el atributo *body_encoding*.

La clase **`Charset`** también proporciona una serie de métodos para soportar operaciones estándar y funciones integradas.

str()

Retorna *input_charset* como una cadena de caracteres convertida a minúsculas. **`__repr__()`** es un alias para **`__str__()`**.

_eq_(*other*)

Este método le permite comparar dos instancias `Charset` por igualdad.

_ne_(*other*)

Este método le permite comparar dos instancias `Charset` por desigualdad.

The **`email.charset`** module also provides the following functions for adding new entries to the global character set, alias, and codec registries:

email.charset.add_charset(*charset*, *header_enc=None*, *body_enc=None*, *output_charset=None*)

Añade propiedades de carácter al registro global.

charset es el conjunto de caracteres de entrada, y debe ser el nombre canónico del conjunto de caracteres.

Opcional *header_enc* y *body_enc* es `charset.QP` para imprimibles entre comillas, `charset.BASE64` para codificación base64, `charset.SHORTEST` para la codificación más corta entre imprimible entre comillas o base64, o `None` para no codificar. `SHORTEST` solo es válido para *header_enc*. El valor predeterminado es `None` para no codificar.

Opcional *output_charset* es es el conjunto de caracteres en el que debe estar la salida. Las conversiones proceden del conjunto de caracteres de entrada, a Unicode, al conjunto de caracteres de salida cuando se llama al método **`Charset.convert()`**. El valor predeterminado es la salida en el mismo conjunto de caracteres que la entrada.

Tanto *input_charset* y *output_charset* deben tener entradas de códec Unicode en el conjunto de caracteres del módulo para la asignación del códec; use `add_codec()` para agregar códecs que el módulo no conozca. Consulte la documentación del módulo `codecs` para más información.

El registro global de conjuntos de caracteres se mantiene en el módulo global diccionario `CHARSETS`.

email.charset.add_alias(*alias*, *canonical*)

Añade un alias al conjunto de caracteres. *alias* es el nombre del alias, p. ej. `latin-1`. *canonical* es el nombre canónico del conjunto de caracteres, p. ej. `iso-8859-1`.

El registro de alias global de conjuntos de caracteres se mantiene en el módulo global diccionario `ALIASES`.

email.charset.add_codec(*charset*, *codecname*)

Añade un códec que asigna caracteres en el conjunto de caracteres dado hacia y desde Unicode.

charset es el nombre canónico de un conjunto de caracteres. *codename* es el nombre de un códec de Python, según corresponda para el segundo argumento del método `str` de `encode()`.

[1]

`email.encoders`: Encoders

Código fuente: Lib/email/encoders.py [https://github.com/python/cpython/tree/3.14/Lib/email/encoders.py]

Este módulo forma parte de la anterior API de correo electrónico (`Compat32`). En la nueva API, la funcionalidad la proporciona el parámetro *cte* del método **`set_content()`**.

Este módulo está obsoleto en Python 3. Las funciones que aparecen aquí no deberían ser llamadas explícitamente ya que la clase **`MIMEText`** establece el tipo de contenido y el encabezado CTE utilizando los valores del *_subtype* y del *_charset* que se pasan cuando se instancia esa clase.

El texto que viene a continuación corresponde a la documentación original del módulo.

Cuando se crean objetos **`Message`** desde 0, a menudo se necesita codificar el contenido del mensaje para transportarlo a través de servidores de correo electrónico adecuados. Esto es así especialmente para el tipo de mensajes *image/** y *text/** que contienen datos binarios.

The **`email`** package provides some convenient encoders in its **`encoders`** module. These encoders are actually used by the **`MIMEAudio`** and **`MIMEImage`** class constructors to provide default encodings. All encoder functions take exactly one argument, the message object to encode. They usually extract the payload, encode it, and reset the payload to this newly encoded value. They should also set the *Content-Transfer-Encoding* header as appropriate.

Ten en cuenta que estas funciones no sirven para un mensaje con múltiples partes. En lugar de aplicarlo al mensaje completo, las funciones deben aplicarse a cada subparte individual. Si se pasa un mensaje de múltiples partes como argumento se activara un mensaje de error `TypeError`.

A continuación, una lista de las funciones de codificación facilitadas:

email.encoders.encode_quopri(*msg*)

Codifica el contenido en formularios entrecomillados e imprimibles y marca el encabezado *Content-Transfer-Encoding* como `quoted-printable` [1]. Es un buen codificador para usar cuando la mayoría del contenido son datos imprimibles normales pero hay algún dato que no es imprimible.

email.encoders.encode_base64(*msg*)

Codifica el contenido en un formulario base64 y marca el encabezado *Content-Transfer-Encoding* como `base64`. Esta codificación es buena cuando la mayoría del contenido son datos no imprimibles ya que es un formulario más compacto que formularios entrecomillados e imprimibles. La desventaja es que incluye el texto que no es leíble por los humanos.

email.encoders.encode_7or8bit(*msg*)

Esto, en realidad, no modifica el contenido del mensaje, pero fija el encabezado *Content-Transfer-Encoding* a `7bit` u `8bit`, lo que considere más adecuado en función del contenido del mensaje.

email.encoders.encode_noop(*msg*)

Esto no hace nada; ni siquiera fija el encabezado *Content-Transfer-Encoding*.

Notas

El codificado con `encode_quopri()` también codifica todas las tabulaciones y caracteres de espacios en los datos.

[1]

`email.utils`: Miscellaneous utilities

Código fuente: Lib/email/utils.py [https://github.com/python/cpython/tree/3.14/Lib/email/utils.py]

There are a couple of useful utilities provided in the **`email.utils`** module:

email.utils.localtime(*dt=None*)

Return local time as an aware datetime object. If called without arguments, return current time. Otherwise *dt* argument should be a **`datetime`** instance, and it is converted to the local time zone according to the system time zone database. If *dt* is naive (that is, `dt.tzinfo` is `None`), it is assumed to be in local time.

Added in version 3.3.

Deprecated since version 3.12, removed in version 3.14: El parámetro *isdst*.

email.utils.make_msgid(*idstring=None*, *domain=None*)

Retorna una cadena de caracteres apropiada para una cabecera *Message-ID* que cumpla con **RFC 2822** [https://datatracker.ietf.org/doc/html/rfc2822.html]. Si se especifica un *idstring* opcional, es una cadena de caracteres utilizada para reforzar la unicidad del identificador del mensaje. Si se especifica un *domain* opcional provee la porción del msgid después de “@”. El predeterminado es el hostname local. Normalmente no es necesario especificar este valor, pero puede ser útil en algunos casos, como cuando se está construyendo un sistema distribuido que utiliza un nombre de dominio consistente a lo largo de varios ordenadores.

Distinto en la versión 3.2: Se añadió la palabra clave *domain*.

Las funciones que quedan son parte de la API de correo electrónico heredada (`Compat32`) . No hay necesidad de utilizarlas directamente con el nuevo API, dado que el análisis sintáctico y formateo que proporcionan es realizado automáticamente por el mecanismo de análisis sintáctico de la nueva API.

email.utils.quote(*str*)

Retorna una nueva cadena de caracteres con barras invertidas (\) en *str* reemplazado por dos barras invertidas, y comillas dobles reemplazadas por barra invertida seguido de comillas dobles.

email.utils.unquote(*str*)

Retorna una nueva cadena de caracteres que es una versión *unquoted* de *str*. Si *str* comienza y termina con comillas dobles, éstas son eliminadas. De igual manera si *str* empieza y termina con comillas angulares (< y >), éstas son eliminadas.

email.utils.parseaddr(*address*, ***, *strict=True*)

Interpreta *address* – la cual debe ser el valor de un campo que contenga un campo tal como *To* o *Cc* – para separarlo en sus componentes *realname* y *email address*. Retorna una tupla de información, a menos que la interpretación falle, en cuyo caso una 2-tupla de `('','')` es retornada.

If *strict* is true, use a strict parser which rejects malformed inputs.

Distinto en la versión 3.13: Add *strict* optional parameter and reject malformed inputs by default.

email.utils.formataddr(*pair*, *charset='utf-8'*)

El inverso de **`parseaddr()`**, este toma una 2-tupla de la forma `(realname,real, email_address)` y retorna una cadena de caracteres válido para una cabecera *To* o *Cc*. Si el primer elemento de *pair* es falso, entonces el segundo elemento es retornado sin modificación.

El *charset* opcional es el conjunto de caracteres que será usado en la codificación **RFC 2047** [https://datatracker.ietf.org/doc/html/rfc2047.html] del `real_name` si el `real_name` contiene caracteres que no sean ASCII. Puede ser una instancia de **`str`** o **`Charset`**. El valor predeterminado es `utf-8`.

Distinto en la versión 3.3: Se añadió la opción *charset*.

email.utils.getaddresses(*fieldvalues*, *, *strict=True*)

This method returns a list of 2-tuples of the form returned by `parseaddr()`. *fieldvalues* is a sequence of header field values as might be returned by **`Message.get_all`**.

If *strict* is true, use a strict parser which rejects malformed inputs.

Here's a simple example that gets all the recipients of a message:

```
from email.utils import getaddresses

tos = msg.get_all('to', [])
ccs = msg.get_all('cc', [])
resent_tos = msg.get_all('resent-to', [])
resent_ccs = msg.get_all('resent-cc', [])
all_recipients = getaddresses(tos + ccs + resent_to
```

Distinto en la versión 3.13: Add *strict* optional parameter and reject malformed inputs by default.

email.utils.parsedate(*date*)

Intenta interpretar una fecha de acuerdo a las reglas en **RFC 2822** [https://datatracker.ietf.org/doc/html/rfc2822.html]. Sin embargo, algunos clientes de correo no siguen ese formato como está especificado, por lo cual **`parsedate()`** intenta adivinar correctamente en esos casos. *date* es una secuencia de caracteres que contiene una fecha **RFC 2822** [https://datatracker.ietf.org/doc/html/rfc2822.html] como `"Mon, 20 Nov`

`1995 19:12:08 -0500"`. Si tiene éxito en interpretar la fecha, **`parsedate()`** retorna una 9-tupla que puede ser pasada directamente a **`time.mktime()`**; de lo contrario `None` es retornado. Observar que los índices 6,7 y 8 de la tupla resultante no son utilizables.

email.utils.parsedate_tz(*date*)

Realiza la misma función que **`parsedate()`**, pero retorna `None` o una 10-tupla; los primeros 9 elementos forman una tupla que puede ser pasada directamente a **`time.mktime()`**, y el décimo es el desfase de la zona horaria de la fecha con respecto a UTC (que es el término oficial para Greenwich Mean Time) [1]. Si la cadena de caracteres de entrada no tiene zona horaria, el último elemento de la tupla retornada es `0`, el cual representa UTC. Nótese que los índices 6, 7 y 8 de la tupla resultante no son utilizables.

email.utils.parsedate_to_datetime(*date*)

Lo inverso de **`format_datetime()`**. Realiza la misma función que **`parsedate()`**, pero en caso de éxito retorna un **`datetime`**; de lo contrario, `ValueError` se lanza si *date* contiene un valor no válido, como una hora superior a 23 o una diferencia de zona horaria no comprendida entre -24 y 24 horas. Si la fecha de entrada tiene una zona horaria de `-0000`, el `datetime` será un `datetime` ingenuo, y si la fecha se ajusta a las RFC, representará una hora en UTC pero sin indicación de la zona horaria de origen real del mensaje de donde proviene la fecha. . Si la fecha de entrada tiene cualquier otra compensación de zona horaria válida, el `datetime` será un `datetime` consciente con el correspondiente **`timezone`** **`tzinfo`**.

Added in version 3.3.

email.utils.mktime_tz(*tuple*)

Cambia una 10-tupla, como la retornada por **`parsedate_tz()`** a una marca de tiempo UTC (segundos desde la Época). Si la zona horaria en la tupla es `None`, asume el tiempo local.

email.utils.formatdate(*timeval=None*, *localtime=False*, *usegmt=False*)

Retorna una fecha como una cadena de caracteres de acuerdo a **RFC 2822** [https://datatracker.ietf.org/doc/html/rfc2822.html], por ejemplo:

```
Fri, 09 Nov 2001 01:08:47 -0000
```

Optional *timeval* if given is a floating-point time value as accepted by **`time.gmtime()`** and **`time.localtime()`**, otherwise the current time is used.

localtime opcional, es una bandera que cuando es `True`, interpreta *timeval* y retorna una fecha relativa a la zona horaria local en lugar de UTC, tomando apropiadamente en cuenta el horario de verano. El valor predeterminado es `False` con lo cual UTC es utilizado.

usegmt opcional, es una bandera que cuando es `True` retorna una fecha como cadena de caracteres con la zona horaria como una cadena de caracteres ascii `GMT`, en lugar de un numérico `-0000`. Esto es necesario para algunos protocolos (como HTTP). Sólo aplica cuando *localtime* es `False`. El valor predeterminado es `False`.

email.utils.format_datetime(*dt*, *usegmt=False*)

Similar a `formatdate`, pero la entrada es una instancia **`datetime`**. Si es un datetime naíf, se asume ser «UTC sin información sobre la zona horaria de origen», y el `-0000` convencional es usado para la zona horaria. Si es un `datetime` consciente entonces el desfase numérico de la zona horaria es utilizado. Si es una zona horaria consciente con un desfase cero, entonces *usegmt* puede ser `True`, en cuyo caso la cadena de caracteres `GMT` es utilizada en lugar del desfase numérico de zona horaria. Esto provee una manera de generar cabeceras de fecha HTTP conforme estándares.

Added in version 3.3.

email.utils.decode_rfc2231(*s*)

Decodifica la cadena de caracteres *s* de acuerdo a **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html].

email.utils.encode_rfc2231(*s*, *charset=None*, *language=None*)

Codifica la cadena de caracteres *s* de acuerdo a **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html]. *charset* y *language* opcionales, si son dados es el conjunto de caracteres y nombre del lenguaje a utilizar. Si ninguno es dado, *s* es retornado sin modificar. Si *charset* es dado pero *language* no, la cadena de caracteres es codificada usando la cadena de caracteres vacía para *language*.

email.utils.collapse_rfc2231_value(*value*, *errors='replace'*, *fallback_charset='us-ascii'*)

Cuando un parámetro de cabecera está codificado en formato **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html], `Message.get_param` puede retornar una 3-tupla conteniendo el conjunto de caracteres, lenguaje y valor. `collapse_rfc2231_value()` convierte esto en una cadena de caracteres unicode. El parámetro opcional *errors* es pasado al argumento *errors* del método `encode()` de `str`; de manera predeterminada recae en `'replace'`. *fallback_charset* opcional, especifica el conjunto de caracteres a utilizar si el especificado en la cabecera **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html] no es conocido por Python; su valor predeterminado es `'us-ascii'`.

Por conveniencia, si el *value* pasado a `collapse_rfc2231_value()` no es una tupla, debería ser una cadena de caracteres y se retorna sin citar.

email.utils.decode_params(*params*)

Decodifica la lista de parámetros de acuerdo a **RFC 2231** [https://datatracker.ietf.org/doc/html/rfc2231.html]. *params* es una secuencia de 2-tuplas conteniendo elementos de la forma

```
(content-type, string-value).
```

Notas a pie de página

Nótese que el signo del desfase de la zona horaria es opuesto al signo de la variable `time.timezone` para la misma zona horaria; este último sigue el estándar POSIX mientras que este módulo sigue **RFC 2822** [https://datatracker.ietf.org/doc/html/rfc2822.html].

`email.iterators`: Iterators

Código fuente: Lib/email/iterators.py [https://github.com/python/cpython/tree/3.14/Lib/email/iterators.py]

Iterating over a message object tree is fairly easy with the `Message.walk` method. The **`email.iterators`** module provides some useful higher level iterations over message object trees.

email.iterators.body_line_iterator(*msg*, *decode=False*)

Itera sobre todas las cargas útiles de todas las subpartes de *msg*, retornando las cargas útiles en cadenas de caracteres línea por línea. Descarta todas las cabeceras de las subpartes, y descarta cualquier subparte con una carga útil que no sea una cadena de caracteres de Python. Esto de alguna forma es equivalente a leer la representación en texto plano del mensaje desde un fichero usando `readline()`, descartando todas las cabeceras intermedias.

El argumento opcional *decode* se pasa a través de `Message.get_payload`.

email.iterators.typed_subpart_iterator(*msg*, *maintype='text'*, *subtype=None*)

Itera sobre todas las subpartes de *msg*, retornando solo las subpartes que coincidan el tipo MIME especificado por *maintype* y *subtype*.

Note que *subtype* es opcional; si se omite, entonces se comprobará si el tipo MIME de las subpartes coincide con el tipo principal solamente. *maintype* es opcional también; su valor por defecto es *text*.

Por tanto, por defecto `typed_subpart_iterator()`

retorna cada parte que tenga un tipo MIME *text/**.

La siguiente función se ha añadido como una útil herramienta de depuración. *No* debe ser considerada parte de la interfaz pública soportada para este paquete.

email.iterators._structure(*msg*, *fp=None*, *level=0*, *include_default=False*)

Imprime una representación con sangrías de los tipos del contenido de la estructura de objetos mensaje. Por ejemplo:

```
>>> msg = email.message_from_file(somefile)
>>> _structure(msg)
multipart/mixed
    text/plain
    text/plain
    multipart/digest
        message/rfc822
            text/plain
        message/rfc822
            text/plain
        message/rfc822
            text/plain
        message/rfc822
            text/plain
        message/rfc822
            text/plain
    text/plain
```

El argumento opcional *fp* es un objeto similar a un fichero en el que imprimir la salida. Debe ser adecuado para la función de Python `print()`. Se usa *level* internamente. *include_default*, si tiene su valor a verdadero, imprime el tipo por defecto también.

[1]

`json` — JSON encoder and decoder

Código fuente: Lib/json/__init__.py [https://github.com/python/cpython/tree/3.14/Lib/json/__init__.py]

JSON (JavaScript Object Notation) [https://json.org], specified by **RFC 7159** [https://datatracker.ietf.org/doc/html/rfc7159.html] (which obsoletes **RFC 4627** [https://datatracker.ietf.org/doc/html/rfc4627.html]) and by ECMA-404 [https://ecma-international.org/publications-and-standards/standards/ecma-404/], is a lightweight data interchange format inspired by JavaScript [https://en.wikipedia.org/wiki/JavaScript] object literal syntax (although it is not a strict subset of JavaScript [1]).

Nota

The term «object» in the context of JSON processing in Python can be ambiguous. All values in Python are objects. In JSON, an object refers to any data wrapped in curly braces, similar to a Python dictionary.

Advertencia

Se debe tener cuidado al analizar datos JSON de fuentes que no son de confianza. Una cadena JSON maliciosa puede hacer que el decodificador consuma considerables recursos del CPU y la memoria. Se recomienda limitar el tamaño de los datos que se van a analizar.

This module exposes an API familiar to users of the standard library **`marshal`** and **`pickle`** modules.

Codificación de jerarquías básicas de objetos de Python:

```
>>> import json
>>> json.dumps(['foo', {'bar': ('baz', None, 1.0, 2)}])
'["foo", {"bar": ["baz", null, 1.0, 2]}]'
>>> print(json.dumps("\"foo\bar"))
"\"foo\bar"
>>> print(json.dumps('\u1234'))
"\u1234"
>>> print(json.dumps('\\'))
"\\"
>>> print(json.dumps({"c": 0, "b": 0, "a": 0}, sort_keys
{"a": 0, "b": 0, "c": 0}
>>> from io import StringIO
>>> io = StringIO()
>>> json.dump(['streaming API'], io)
>>> io.getvalue()
'["streaming API"]'
```

Codificación compacta:

```
>>> import json
>>> json.dumps([1, 2, 3, {'4': 5, '6': 7}], separators=
'[1,2,3,{"4":5,"6":7}]'
```

Impresión linda:

```
>>> import json
>>> print(json.dumps({'6': 7, '4': 5}, sort_keys=True, i
{
    "4": 5,
    "6": 7
}
```

Customizing JSON object encoding:

```
>>> import json
>>> def custom_json(obj):
...     if isinstance(obj, complex):
...         return {'__complex__': True, 'real': obj.rea
...     raise TypeError(f'Cannot serialize object of {ty
...
```

```
>>> json.dumps(1 + 2j, default=custom_json)
'{"__complex__": true, "real": 1.0, "imag": 2.0}'
```

Decodificación JSON:

```
>>> import json
>>> json.loads('["foo", {"bar":["baz", null, 1.0, 2]}]')
['foo', {'bar': ['baz', None, 1.0, 2]}]
>>> json.loads('"\\"foo\\bar"')
'"foo\x08ar'
>>> from io import StringIO
>>> io = StringIO('["streaming API"]')
>>> json.load(io)
['streaming API']
```

Customizing JSON object decoding:

```
>>> import json
>>> def as_complex(dct):
...     if '__complex__' in dct:
...         return complex(dct['real'], dct['imag'])
...     return dct
...
>>> json.loads('{"__complex__": true, "real": 1, "imag":
...     object_hook=as_complex)
(1+2j)
>>> import decimal
>>> json.loads('1.1', parse_float=decimal.Decimal)
Decimal('1.1')
```

Extendiendo **`JSONEncoder`**:

```
>>> import json
>>> class ComplexEncoder(json.JSONEncoder):
...     def default(self, obj):
...         if isinstance(obj, complex):
...             return [obj.real, obj.imag]
...         # Let the base class default method raise th
...         return super().default(obj)
...
```

```
>>> json.dumps(2 + 1j, cls=ComplexEncoder)
'[2.0, 1.0]'
>>> ComplexEncoder().encode(2 + 1j)
'[2.0, 1.0]'
>>> list(ComplexEncoder().iterencode(2 + 1j))
['[2.0', ', 1.0', ']']
```

Using `json` from the shell to validate and pretty-print:

```
$ echo '{"json":"obj"}' | python -m json
{
    "json": "obj"
}
$ echo '{1.2:3.4}' | python -m json
Expecting property name enclosed in double quotes: line
```

Consulte Command-line interface para obtener documentación detallada.

Nota

JSON es un subconjunto de YAML [http://yaml.org/] 1.2. El JSON producido por la configuración predeterminada de este módulo (en particular, el valor predeterminado *separators*) también es un subconjunto de YAML 1.0 y 1.1. Por lo tanto, este módulo también se puede utilizar como un serializador YAML.

Nota

Los codificadores y decodificadores de este módulo conservan el orden de entrada y salida de forma predeterminada. El orden solo se pierde si los contenedores subyacentes no están ordenados.

Uso básico

json.dump(*obj*, *fp*, *, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None, default=None, sort_keys=False, **kw*)

Serialize *obj* as a JSON formatted stream to *fp* (a `.write()`-supporting file-like object) using this Python-to-JSON conversion table.

Nota

A diferencia de **`pickle`** y **`marshal`**, JSON no es un protocolo enmarcado, por lo que intentar serializar varios objetos con llamadas repetidas a **`dump()`** utilizando el mismo *fp* dará como resultado un archivo JSON no válido.

Parámetros:

- **obj** (*object*) – The Python object to be serialized.
- **fp** (file-like object) – The file-like object *obj* will be serialized to. The **`json`** module always produces **`str`** objects, not **`bytes`** objects, therefore `fp.write()` must support **`str`** input.
- **skipkeys** (*bool*) – If `True`, keys that are not of a basic type (**`str`**, **`int`**, **`float`**, **`bool`**, `None`) will be skipped instead of raising a **`TypeError`**. Default `False`.
- **ensure_ascii** (*bool*) – If `True` (the default), the output is guaranteed to have all incoming non-ASCII and non-printable characters escaped. If `False`, all characters will be outputted as-is, except for the characters that must be escaped: quotation mark, reverse solidus, and the control characters U+0000 through U+001F.
- **check_circular** (*bool*) – If `False`, the circular reference check for container types is skipped and a circular reference will result in a **`RecursionError`** (or worse). Default

`True`.

- **allow_nan** (*bool*) – If `False`, serialization of out-of-range **`float`** values (`nan`, `inf`, `-inf`) will result in a **`ValueError`**, in strict compliance with the JSON specification. If `True` (the default), their JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`) are used.
- **cls** (a **`JSONEncoder`** subclass) – If set, a custom JSON encoder with the **`default()`** method overridden, for serializing into custom datatypes. If `None` (the default), **`JSONEncoder`** is used.
- **indent** (*int* | *str* | *None*) – If a positive integer or string, JSON array elements and object members will be pretty-printed with that indent level. A positive integer indents that many spaces per level; a string (such as `"\t"`) is used to indent each level. If zero, negative, or `""` (the empty string), only newlines are inserted. If `None` (the default), the most compact representation is used.
- **separators** (*tuple* | *None*) – A two-tuple: `(item_separator, key_separator)`. If `None` (the default), *separators* defaults to `(', ', ': ')` if *indent* is `None`, and `(',', ': ')` otherwise. For the most compact JSON, specify `(',', ':')` to eliminate whitespace.
- **default** (callable | None) – A function that is called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a **`TypeError`**. If `None` (the default), **`TypeError`** is raised.

- **sort_keys** (*bool*) – If `True`, dictionaries will be outputted sorted by key. Default `False`.

Distinto en la versión 3.2: Permite cadenas de caracteres para *indent* además de enteros.

Distinto en la versión 3.4: Usa `(',', ': ')` como predeterminado si *indent* no es `None`.

Distinto en la versión 3.6: Todos los parámetros opcionales son ahora palabra-clave-solamente.

json.dumps(*obj*, ***, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serializa *obj* en un `str` con formato JSON usando esta conversion table. Los argumentos tienen el mismo significado que en `dump()`.

Nota

Las llaves de los pares llave/valor de JSON siempre son del tipo `str`. Cuando un diccionario se convierte en JSON, todas las llaves del diccionario se convierten en cadenas. Como resultado de esto, si un diccionario se convierte en JSON y, a continuación, se convierte nuevamente en un diccionario, el diccionario puede que no sea igual al original. Es decir, `loads(dumps(x)) != x` si x tiene llaves que no son de tipo cadena de caracteres.

json.load(*fp*, ***, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize *fp* to a Python object using the JSON-to-Python conversion table.

Parámetros:

- **fp** (file-like object) – A `.read()`-supporting text file or binary file containing the JSON document to be deserialized.
- **cls** (a **`JSONDecoder`** subclass) – If set, a custom JSON decoder. Additional keyword arguments to **`load()`** will be passed to the constructor of *cls*. If `None` (the default), **`JSONDecoder`** is used.
- **object_hook** (callable | None) – If set, a function that is called with the result of any JSON object literal decoded (a **`dict`**). The return value of this function will be used instead of the **`dict`**. This feature can be used to implement custom decoders, for example JSON-RPC [https://www.jsonrpc.org] class hinting. Default `None`.
- **object_pairs_hook** (callable | None) – If set, a function that is called with the result of any JSON object literal decoded with an ordered list of pairs. The return value of this function will be used instead of the **`dict`**. This feature can be used to implement custom decoders. If *object_hook* is also set, *object_pairs_hook* takes priority. Default `None`.
- **parse_float** (callable | None) – If set, a function that is called with the string of every JSON float to be decoded. If `None` (the default), it is equivalent to `float(num_str)`. This can be used to parse JSON floats into custom datatypes, for example **`decimal.Decimal`**.
- **parse_int** (callable | None) – If set, a function that is called with the string of

every JSON int to be decoded. If `None` (the default), it is equivalent to `int(num_str)`. This can be used to parse JSON integers into custom datatypes, for example **`float`**.

- **parse_constant** (callable | None) – If set, a function that is called with one of the following strings: `'-Infinity'`, `'Infinity'`, or `'NaN'`. This can be used to raise an exception if invalid JSON numbers are encountered. Default `None`.

Muestra:

- **JSONDecodeError** – When the data being deserialized is not a valid JSON document.
- **UnicodeDecodeError** – When the data being deserialized does not contain UTF-8, UTF-16 or UTF-32 encoded data.

Distinto en la versión 3.1:

- Added the optional *object_pairs_hook* parameter.
- *parse_constant* ya no es llamado en "null", "true", "false".

Distinto en la versión 3.6:

- Todos los parámetros opcionales son ahora palabra-clave-solamente.
- *fp* ahora puede ser un binary file. La codificación de entrada debe ser UTF-8, UTF-16 o UTF-32.

Distinto en la versión 3.11: El *parse_int* predeterminado de **`int()`** ahora limita la longitud máxima de la cadena de enteros a través de la integer string conversion length limitation del intérprete para ayudar a evitar ataques de denegación de servicio.

json.loads(*s*, ***, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Identical to `load()`, but instead of a file-like object, deserialize *s* (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object using this conversion table.

Distinto en la versión 3.6: s ahora puede ser de tipo `bytes` o `bytearray`. La codificación de entrada debe ser UTF-8, UTF-16 o UTF-32.

Distinto en la versión 3.9: El argumento de palabra llave *encoding* ha sido removido.

Codificadores y Decodificadores

class json.JSONDecoder(***, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *strict=True*, *object_pairs_hook=None*)

Decodificador JSON simple.

Realiza las siguientes traducciones en la decodificación de forma predeterminada:

JSON	Python
object	dict
array	list
string	str
número (int)	int
número (real)	float
true	True
false	False
null	None

También entiende `NaN`, `Infinity` y `-Infinity` como sus correspondientes valores `float`, que está fuera de la especificación JSON.

object_hook is an optional function that will be called with the result of every JSON object decoded and its return value will be used in place of the given **`dict`**. This can be used to provide custom deserializations (e.g. to support JSON-RPC [https://www.jsonrpc.org] class hinting).

object_pairs_hook is an optional function that will be called with the result of every JSON object decoded with an ordered list of pairs. The return value of *object_pairs_hook* will be used instead of the **`dict`**. This feature can be used to implement custom decoders. If *object_hook* is also defined, the *object_pairs_hook* takes priority.

Distinto en la versión 3.1: Soporte agregado para *object_pairs_hook*.

parse_float is an optional function that will be called with the string of every JSON float to be decoded. By default, this is equivalent to `float(num_str)`. This can be used to use another datatype or parser for JSON floats (e.g. **`decimal.Decimal`**).

parse_int is an optional function that will be called with the string of every JSON int to be decoded. By default, this is equivalent to `int(num_str)`. This can be used to use another datatype or parser for JSON integers (e.g. **`float`**).

parse_constant is an optional function that will be called with one of the following strings: `'-Infinity'`, `'Infinity'`, `'NaN'`. This can be used to raise an exception if invalid JSON numbers are encountered.

Si *strict* es falso (“”True”” es el valor predeterminado), se permitirán caracteres de control dentro de cadenas. Los caracteres de control en este contexto son aquellos con códigos de caracteres en el rango 0–31, incluyendo `'\t'` (tab), `'\n'`, `'\r'` and `'\0'`.

Si los datos que se deserializan no constituyen un documento JSON válido, se lanzará un **`JSONDecodeError`**.

Distinto en la versión 3.6: Todos los parámetros son ahora palabra-clave-solamente.

decode(*s*)

Retorna la representación Python de *s* (una instancia `str` que contiene un documento JSON).

`JSONDecodeError` se producirá si el documento JSON entregado es invalido.

raw_decode(*s*)

Decodifica un documento JSON de *s* (un `str` comenzando con un documento JSON) y retorna una tupla de 2 de la representación Python y el índice en *s* donde terminó el documento.

Esto se puede usar para decodificar un documento JSON de una cadena de caracteres que puede tener datos extraños al final.

class json.JSONEncoder(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, default=None*)

Codificador JSON extensible para estructuras de datos de Python.

Admite los siguientes objetos y tipos de forma predeterminada:

Python	**JSON**
dict	object
list, tuple	array
str	string
int, float, Enums derivadas de int o float	number
True	true
False	false
None	null

Distinto en la versión 3.4: Compatibilidad añadida con las clases Enum derivadas de int y float.

A fin de extender esto para reconocer otros objetos, implementar una subclase con un método `default()` con otro método que retorna un objeto serializable para “”o”” si es posible, de lo contrario debe llamar a la implementación de superclase (para lanzar `TypeError`).

If *skipkeys* is false (the default), a `TypeError` will be raised when trying to encode keys that are not `str`, `int`, `float`, `bool` or `None`. If *skipkeys* is true, such items are simply skipped.

If *ensure_ascii* is true (the default), the output is guaranteed to have all incoming non-ASCII and non-printable characters escaped. If *ensure_ascii* is false, all characters will be output as-is, except for the characters that must be escaped: quotation mark, reverse solidus, and the control characters U +0000 through U+001F.

Si *check_circular* es cierto (valor predeterminado), se comprobarán las listas, los diccionarios y los objetos codificados personalizados en busca de referencias circulares durante la codificación para evitar una recursividad infinita (lo que provocaría un `RecursionError`). De lo contrario, no se realiza ninguna comprobación de este tipo.

Si *allow_nan* es cierto (valor predeterminado), `NaN`, `Infinity` y `-Infinity` se codificarán como tales. Este comportamiento no es compatible con las especificaciones JSON, pero es coherente con la mayoría de los codificadores y decodificadores basados en JavaScript. De lo contrario, codificar dichos puntos flotantes provocará un `ValueError`.

Si *sort_keys* es cierto (predeterminado: `False`), la salida de los diccionarios se ordenará por clave; esto es útil para las pruebas de regresión para garantizar que las serializaciones JSON se pueden comparar en el día a día.

Si *indent* es un entero no negativo o una cadena, los elementos del arreglo JSON y los miembros del objeto se imprimirán con ese nivel de sangría. Un nivel de sangría de 0, negativo o `""` solo insertará nuevas líneas. `None` (el valor predeterminado) selecciona la representación más compacta. El uso de una sangría de entero positivo agrega sangrías de muchos espacios por nivel. Si *indent* es una cadena (como `"\t"`), esa cadena se usa para agregarle sangría a cada nivel.

Distinto en la versión 3.2: Permite cadenas de caracteres para *indent* además de enteros.

Si se especifica, *separators* debe ser una tupla `(separador_elemento, separador_llave)`. El valor predeterminado es `(', ', ': ')` si *indent* es `None` y `(',', ': ')` de lo contrario. Para obtener la representación JSON más compacta, debe especificar `(',', ':')` para eliminar espacios en blanco.

Distinto en la versión 3.4: Usa `(',', ': ')` como predeterminado si *indent* no es `None`.

Si se especifica, *default* debería ser una función que se llama para objetos que de otro modo no se pueden serializar. Debería retornar una versión codificable JSON del objeto o generar un **`TypeError`**. Si no se especifica, produce **`TypeError`**.

Distinto en la versión 3.6: Todos los parámetros son ahora palabra-clave-solamente.

default(*o*)

Implemente este método en una subclase de modo que retorne un objeto serializable para *o*, o llame a la implementación base (para generar un **`TypeError`**).

Por ejemplo, para admitir iteradores arbitrarios, podría implementar **`default()`** de esta manera:

```
def default(self, o):
   try:
       iterable = iter(o)
   except TypeError:
```

```
            pass
        else:
            return list(iterable)
        # Let the base class default method raise th
        return super().default(o)
```

encode(*o*)

Retorna una representación de cadena de caracteres JSON de una estructura de datos de Python, *o*. Por ejemplo:

```
>>> json.JSONEncoder().encode({"foo": ["bar", '
'{"foo": ["bar", "baz"]}'
```

iterencode(*o*)

Codifica el objeto dado, *o*, y produce cada representación de cadena como disponible. Por ejemplo:

```
for chunk in json.JSONEncoder().iterencode(bigo
    mysocket.write(chunk)
```

Excepciones

exception json.JSONDecodeError(*msg*, *doc*, *pos*)

Subclase de `ValueError` con los siguientes atributos adicionales:

msg

El mensaje de error sin formato.

doc

El documento JSON que se está analizando.

pos

El índice de inicio de *doc* donde se produjo un error en el análisis.

lineno

La línea correspondiente a *pos*.

colno

La columna correspondiente a *pos*.

Added in version 3.5.

Cumplimiento e interoperabilidad estándar

The JSON format is specified by **RFC 7159** [https://datatracker.ietf.org/doc/html/rfc7159.html] and by ECMA-404 [https://ecma-international.org/publications-and-standards/standards/ecma-404/]. This section details this module's level of compliance with the RFC. For simplicity, **`JSONEncoder`** and **`JSONDecoder`** subclasses, and parameters other than those explicitly mentioned, are not considered.

Este módulo no cumple con la RFC de forma estricta, implementando algunas extensiones que son válidas en JavaScript pero no son válidas en JSON. En particular:

- Se aceptan y se envían valores de números Infinitos y NaN;
- Se aceptan nombres repetidos dentro de un objeto y solo se utiliza el valor del último par nombre-valor.

Puesto que el RFC permite a los analizadores compatibles con RFC aceptar textos de entrada que no son compatibles con RFC, el deserializador de este módulo es técnicamente compatible con RFC bajo la configuración predeterminada.

Codificaciones de caracteres

La RFC requiere que JSON se represente mediante UTF-8, UTF-16 o UTF-32, siendo UTF-8 el valor predeterminado recomendado para la máxima interoperabilidad.

As permitted, though not required, by the RFC, this module's serializer sets *ensure_ascii*=*True* by default, thus escaping the output so that the resulting strings only contain printable ASCII characters.

Aparte del parámetro *ensure_ascii*, este módulo se define estrictamente en términos de conversión entre objetos Python y `Unicode strings`, y por lo tanto no aborda directamente el problema de las codificaciones de caracteres.

La RFC prohíbe agregar una marca de orden byte (BOM, por sus siglas en inglés) al inicio de un texto JSON y el serializador de este módulo no agrega una BOM a su salida. La RFC permite, pero no requiere, deserializadores JSON para omitir una BOM inicial en su entrada. El deserializador de este módulo genera un `ValueError` cuando hay una lista de materiales inicial.

La RFC no prohíbe explícitamente las cadenas JSON que contienen secuencias de bytes que no corresponden a caracteres Unicode válidos (por ejemplo, sustitutos UTF-16 no espaciados), pero sí tiene en cuenta que pueden causar problemas de interoperabilidad. De forma predeterminada, este módulo acepta y genera puntos de código (cuando está presente en el original `str`) para dichas secuencias.

Valores de número infinito y NaN

El RFC no permite la representación de los valores de número infinito o NaN. A pesar de eso, de forma predeterminada, este módulo acepta y genera `Infinity`, `-Infinity` y `NaN` como si fueran valores literales de número JSON válidos:

```
>>> # Neither of these calls raises an exception, but th
>>> json.dumps(float('-inf'))
'-Infinity'
>>> json.dumps(float('nan'))
'NaN'
>>> # Same when deserializing
>>> json.loads('-Infinity')
-inf
>>> json.loads('NaN')
nan
```

En el serializador, el parámetro *allow_nan* se puede utilizar para modificar este comportamiento. En el deserializador, se puede utilizar el parámetro *parse_constant* para modificar este comportamiento.

Nombres repetidos dentro de un objeto

La RFC especifica que los nombres dentro de un objeto JSON deben ser únicos, pero no exige cómo se deben controlar los nombres repetidos en los objetos JSON. De forma predeterminada, este módulo no genera una excepción; en su lugar, ignora todo excepto el último par nombre-valor para un nombre dado:

```
>>> weird_json = '{"x": 1, "x": 2, "x": 3}'
>>> json.loads(weird_json)
{'x': 3}
```

El parámetro *object_pairs_hook* se puede utilizar para alterar este comportamiento.

Valores de nivel superior No-Objeto , No-Arreglo

La versión anterior de JSON especificada por el obsoleto **RFC 4627** [https://datatracker.ietf.org/doc/html/rfc4627.html] requería que el valor de nivel superior de un texto JSON fuera un objeto JSON o un arreglo (Python `dict` o `list`), y no podía ser un valor JSON nulo, booleano, numérico o de cadena. **RFC 7159** [https://datatracker.ietf.org/doc/html/rfc7159.html] eliminó esa restricción, y este módulo no ha implementado ni ha implementado nunca esa restricción en su serializador o en su deserializador.

Independientemente, para lograr la máxima interoperabilidad, es posible que usted desee adherirse voluntariamente a la restricción.

Limitaciones de la implementación

Algunas implementaciones del deserializador JSON pueden establecer límites en:

- el tamaño de los textos JSON aceptados

- el nivel máximo de anidamiento de objetos y arreglos JSON
- el rango y precisión de los números JSON
- el contenido y la longitud máxima de las cadenas de caracteres JSON

Este módulo no impone tales límites más allá de los propios tipos de datos de Python relevantes o del propio intérprete de Python.

Al serializar en JSON, tenga en cuenta las limitaciones en las aplicaciones que pueden consumir su JSON. En particular, es común que los números JSON se deserialicen en números de doble precisión IEEE 754 y, por lo tanto, estén sujetos al rango y las limitaciones de precisión de esa representación. Esto es especialmente relevante cuando se serializan valores de Python `int` de magnitud extremadamente grande, o cuando se serializan instancias de tipos numéricos «exóticos» como `decimal.Decimal`.

Command-line interface

Código fuente: Lib/json/tool.py [https://github.com/python/cpython/tree/3.14/Lib/json/tool.py]

The **`json`** module can be invoked as a script via `python -m json` to validate and pretty-print JSON objects. The **`json.tool`** submodule implements this interface.

Si no se especifican los argumentos opcionales `infile` y `outfile`, se utilizarán `sys.stdin` y `sys.stdout` respectivamente:

```
$ echo '{"json": "obj"}' | python -m json
{
    "json": "obj"
}
$ echo '{1.2:3.4}' | python -m json
Expecting property name enclosed in double quotes: line
```

Distinto en la versión 3.5: La salida está ahora en el mismo orden que la entrada. Utilice la opción **`--sort-keys`** para ordenar la salida de los diccionarios alfabéticamente por llave.

Distinto en la versión 3.14: The **`json`** module may now be directly executed as `python -m json`. For backwards compatibility, invoking the CLI as `python -m json.tool` remains supported.

Command-line options

infile

El archivo JSON que se va a validar o imprimir con impresión linda:

```
$ python -m json mp_films.json
[
    {
        "title": "And Now for Something Completely
        "year": 1971
    },
    {
        "title": "Monty Python and the Holy Grail",
        "year": 1975
    }
]
```

Si no se especifica *infile*, lee **`sys.stdin`**.

outfile

Escribe la salida de *infile* en el *outfile* dado. De lo contrario, lo escribe en **`sys.stdout`**.

--sort-keys

Ordena la salida de los diccionarios alfabéticamente por llave.

Added in version 3.5.

--no-ensure-ascii

Deshabilita el escape de caracteres que no sean ascii, véase **`json.dumps()`** para más información.

Added in version 3.9.

--json-lines

Analiza cada línea de entrada como objeto JSON independiente.

Added in version 3.8.

--indent, --tab, --no-indent, --compact

Opciones mutuamente exclusivas para control de espacios en blanco.

Added in version 3.9.

-h, --help

Muestra el mensaje de ayuda.

Notas de pie de página

Como se indica en la errata para RFC 7159 [https://www.rfc-editor.org/errata_search.php?rfc-7159], JSON permite caracteres literales U+2028 (SEPARADOR DE LINEA) y U+2029 (SEPARADOR DE PÁRRAFO) en cadenas, mientras que JavaScript (a partir de ECMAScript Edición 5.1) no lo hace.

`mailbox` — Manipulate mailboxes in various formats

Código fuente: Lib/mailbox.py [https://github.com/python/cpython/tree/3.14/Lib/mailbox.py]

This module defines two classes, `Mailbox` and `Message`, for accessing and manipulating on-disk mailboxes and the messages they contain. `Mailbox` offers a dictionary-like mapping from keys to messages. `Message` extends the `email.message` module's `Message` class with format-specific state and behavior. Supported mailbox formats are Maildir, mbox, MH, Babyl, and MMDF.

Ver también

Módulo `email`
Representar y manipular mensajes.

`Mailbox` objects

class mailbox.Mailbox

Un buzón de correo, que se puede inspeccionar y modificar.

The `Mailbox` class defines an interface and is not intended to be instantiated. Instead, format-specific subclasses should inherit from `Mailbox` and your code should instantiate a particular subclass.

The `Mailbox` interface is dictionary-like, with small keys corresponding to messages. Keys are issued by the `Mailbox` instance with which they will be used and are only meaningful to that `Mailbox` instance. A key continues to identify a message even if the corresponding message is modified, such as by replacing it with another message.

Messages may be added to a **`Mailbox`** instance using the set-like method `add()` and removed using a `del` statement or the set-like methods `remove()` and `discard()`.

`Mailbox` interface semantics differ from dictionary semantics in some noteworthy ways. Each time a message is requested, a new representation (typically a `Message` instance) is generated based upon the current state of the mailbox. Similarly, when a message is added to a **`Mailbox`** instance, the provided message representation's contents are copied. In neither case is a reference to the message representation kept by the **`Mailbox`** instance.

The default **`Mailbox`** iterator iterates over message representations, not keys as the default `dictionary` iterator does. Moreover, modification of a mailbox during iteration is safe and well-defined. Messages added to the mailbox after an iterator is created will not be seen by the iterator. Messages removed from the mailbox before the iterator yields them will be silently skipped, though using a key from an iterator may result in a `KeyError` exception if the corresponding message is subsequently removed.

Advertencia

Be very cautious when modifying mailboxes that might be simultaneously changed by some other process. The safest mailbox format to use for such tasks is `Maildir`; try to avoid using single-file formats such as `mbox` for concurrent writing. If you're modifying a mailbox, you *must* lock it by calling the `lock()` and `unlock()` methods *before* reading any messages in the file or making any changes by adding or deleting a message. Failing to lock the mailbox runs the risk of losing messages or corrupting the entire mailbox.

`Mailbox` instances have the following methods:

add(*message*)

Añade *message* al buzón de correo y retorna la clave que se le ha asignado.

El parámetro *message* puede ser una instancia `Message`, una instancia `email.message.Message`, una cadena, una cadena de bytes o un objeto tipo archivo (que debe estar abierto en modo binario). Si *message* es una instancia de la subclase `Message` con el formato apropiado (por ejemplo, si es una instancia `mboxMessage` y ésta es una instancia `mbox`), se utiliza su información de formato específico. En caso contrario, se utilizan valores por defecto razonables para la información específica del formato.

Distinto en la versión 3.2: Se añadió el soporte para la entrada binaria.

remove(*key*)
__delitem__(*key*)
discard(*key*)

Borre el mensaje correspondiente a la *key* del buzón de correo.

Si no existe tal mensaje, se lanza una excepción `KeyError` si el método se llamó `remove()` o `__delitem__()` pero no se lanza una excepción si el método se llamó `discard()`. El comportamiento de `discard()` puede ser preferido si el formato de buzón subyacente soporta la modificación concurrente por otros procesos.

__setitem__(*key*, *message*)

Reemplaza el mensaje correspondiente a *key* por *message*. Levante una excepción `KeyError` si ningún mensaje ya corresponde a *key*.

Al igual que `add()`, el parámetro *message* puede ser una instancia `Message`, una instancia `email.message.Message`, una cadena, una cadena de bytes, o un objeto tipo archivo (que debe estar abierto en modo binario). Si *message* es una instancia de la subclase **`Message`** con el formato apropiado (por ejemplo, si es una instancia `mboxMessage` y ésta es una instancia `mbox`), se utiliza su información de formato específico. En caso contrario, la información específica del formato del mensaje que actualmente corresponde a *key* se deja sin cambios.

iterkeys()

Return an iterator over all keys

keys()

The same as `iterkeys()`, except that a `list` is returned rather than an iterator

itervalues()
__iter__()

Return an iterator over representations of all messages. The messages are represented as instances of the appropriate format-specific `Message` subclass unless a custom message factory was specified when the **`Mailbox`** instance was initialized.

Nota

El comportamiento de `__iter__()` es diferente al de los diccionarios, que iteran sobre las claves.

values()

The same as `itervalues()`, except that a `list` is returned rather than an iterator

iteritems()

Return an iterator over (*key*, *message*) pairs, where *key* is a key and *message* is a message representation. The messages are represented as instances of the appropriate format-specific `Message` subclass unless a custom message factory was specified when the `Mailbox` instance was initialized.

items()

The same as `iteritems()`, except that a `list` of pairs is returned rather than an iterator of pairs.

get(*key*, *default=None*)
__getitem__(*key*)

Return a representation of the message corresponding to *key*. If no such message exists, *default* is returned if the method was called as `get()` and a `KeyError` exception is raised if the method was called as `__getitem__()`. The message is represented as an instance of the appropriate format-specific `Message` subclass unless a custom message factory was specified when the `Mailbox` instance was initialized.

get_message(*key*)

Retorna una representación del mensaje correspondiente a *key* como una instancia de la subclase `Message` específica del formato, o lanza una excepción `KeyError` si no existe tal mensaje.

get_bytes(*key*)

Retorna una representación en bytes del mensaje correspondiente a *key*, o lanza una excepción `KeyError` si no existe tal mensaje.

Added in version 3.2.

get_string(*key*)

Retorna una representación en cadena del mensaje correspondiente a *key*, o lanza una excepción

`KeyError` si no existe tal mensaje. El mensaje se procesa a través de `email.message.Message` para convertirlo en una representación limpia de 7 bits.

get_file(*key*)

Return a file-like representation of the message corresponding to *key*, or raise a `KeyError` exception if no such message exists. The file-like object behaves as if open in binary mode. This file should be closed once it is no longer needed.

Distinto en la versión 3.2: The file object really is a binary file; previously it was incorrectly returned in text mode. Also, the file-like object now supports the context manager protocol: you can use a `with` statement to automatically close it.

Nota

Unlike other representations of messages, file-like representations are not necessarily independent of the `Mailbox` instance that created them or of the underlying mailbox. More specific documentation is provided by each subclass.

__contains__(*key*)

Retorna `True` si «key» corresponde a un mensaje, si no `False`.

__len__()

Retorna un recuento de los mensajes en el buzón de correo.

clear()

Borrar todos los mensajes del buzón.

pop(*key*, *default=None*)

Return a representation of the message corresponding to *key* and delete the message. If no such message exists, return *default*. The message is represented as an instance of the appropriate format-specific `Message` subclass unless a custom message factory was specified when the `Mailbox` instance was initialized.

popitem()

Return an arbitrary (*key*, *message*) pair, where *key* is a key and *message* is a message representation, and delete the corresponding message. If the mailbox is empty, raise a `KeyError` exception. The message is represented as an instance of the appropriate format-specific `Message` subclass unless a custom message factory was specified when the `Mailbox` instance was initialized.

update(*arg*)

Parameter *arg* should be a *key*-to-*message* mapping or an iterable of (*key*, *message*) pairs. Updates the mailbox so that, for each given *key* and *message*, the message corresponding to *key* is set to *message* as if by using `__setitem__()`. As with `__setitem__()`, each *key* must already correspond to a message in the mailbox or else a `KeyError` exception will be raised, so in general it is incorrect for *arg* to be a `Mailbox` instance.

Nota

A diferencia de los diccionarios, los argumentos de las palabras clave no están soportados.

flush()

Write any pending changes to the filesystem. For some `Mailbox` subclasses, changes are always written

immediately and **`flush()`** does nothing, but you should still make a habit of calling this method.

lock()

Adquiera un aviso exclusivo de bloqueo en el buzón de correo para que otros procesos sepan que no deben modificarlo. Un **`ExternalClashError`** se lanza si el bloqueo no está disponible. Los mecanismos de bloqueo particulares utilizados dependen del formato del buzón de correo. Deberías *siempre* bloquear el buzón antes de hacer cualquier modificación a su contenido.

unlock()

Libera el bloqueo del buzón de correo, si lo hay.

close()

Flush the mailbox, unlock it if necessary, and close any open files. For some **`Mailbox`** subclasses, this method does nothing.

`Maildir` objects

class mailbox.Maildir(*dirname*, *factory=None*, *create=True*)

Una subclase de **`Mailbox`** para los buzones de correo en formato Maildir. El parámetro *factory* es un objeto invocable que acepta una representación de mensaje tipo archivo (que se comporta como si se abriera en modo binario) y retorna una representación personalizada. Si *factory* es `None`, **`MaildirMessage`** se utiliza como representación de mensaje por defecto. Si *create* es `True`, el buzón se crea si no existe.

Si *create* es `True` y la ruta de *dirname* existe, será tratado como un *maildir* existente sin intentar verificar su diseño de directorio.

Es por razones históricas que *dirname* es nombrado como tal en lugar de *path*.

Maildir es un formato de buzón de correo basado en un directorio inventado para el agente de transferencia de correo qmail y ahora ampliamente soportado por otros programas. Los mensajes en un buzón de correo de Maildir se almacenan en archivos separados dentro de una estructura de directorio común. Este diseño permite que los buzones de Maildir sean accedidos y modificados por múltiples programas no relacionados sin corrupción de datos, por lo que el bloqueo de archivos es innecesario.

Los buzones de correo de Maildir contienen tres subdirectorios, a saber: `tmp`, `new`, y `cur`. Los mensajes se crean momentáneamente en el subdirectorio `tmp` y luego se mueven al subdirectorio `new` para finalizar la entrega. Un agente de usuario de correo puede posteriormente mover el mensaje al subdirectorio `cur` y almacenar la información sobre el estado del mensaje en una sección especial «info» adjunta a su nombre de archivo.

Folders of the style introduced by the Courier mail transfer agent are also supported. Any subdirectory of the main mailbox is considered a folder if `'.'` is the first character in its name. Folder names are represented by **`Maildir`** without the leading `'.'`. Each folder is itself a Maildir mailbox but should not contain other folders. Instead, a logical nesting is indicated using `'.'` to delimit levels, e.g., «Archived.2005.07».

colon

> La especificación Maildir requiere el uso de dos puntos (`':'`) en ciertos nombres de archivos de mensajes. Sin embargo, algunos sistemas operativos no permiten este carácter en los nombres de archivo, si desea utilizar un formato similar a Maildir en dicho sistema operativo, debe especificar otro carácter para utilizarlo en su lugar. El signo de exclamación (`'!'`) es una elección popular. Por ejemplo:

```
import mailbox
mailbox.Maildir.colon = '!'
```

The **`colon`** attribute may also be set on a per-instance basis.

Distinto en la versión 3.13: **`Maildir`** now ignores files with a leading dot.

`Maildir` instances have all of the methods of **`Mailbox`** in addition to the following:

list_folders()

Retorna una lista con los nombres de todas las carpetas.

get_folder(*folder*)

Return a **`Maildir`** instance representing the folder whose name is *folder*. A **`NoSuchMailboxError`** exception is raised if the folder does not exist.

add_folder(*folder*)

Create a folder whose name is *folder* and return a **`Maildir`** instance representing it.

remove_folder(*folder*)

Elimina la carpeta cuyo nombre es *folder*. Si la carpeta contiene algún mensaje, se lanzará una excepción **`NotEmptyError`** y la carpeta no se borrará.

clean()

Borra los archivos temporales del buzón de correo que no han sido accedidos en las últimas 36 horas. La especificación Maildir dice que los programas de lectura de correo deben hacer esto ocasionalmente.

get_flags(*key*)

Return as a string the flags that are set on the message corresponding to *key*. This is the same as `get_message(key).get_flags()` but much faster, because it does not open the message file. Use this

method when iterating over the keys to determine which messages are interesting to get.

If you do have a **`MaildirMessage`** object, use its **`get_flags()`** method instead, because changes made by the message's **`set_flags()`**, **`add_flag()`** and **`remove_flag()`** methods are not reflected here until the mailbox's **`__setitem__()`** method is called.

Added in version 3.13.

set_flags(*key*, *flags*)

On the message corresponding to *key*, set the flags specified by *flags* and unset all others. Calling `some_mailbox.set_flags(key, flags)` is similar to

```
one_message = some_mailbox.get_message(key)
one_message.set_flags(flags)
some_mailbox[key] = one_message
```

but faster, because it does not open the message file.

If you do have a **`MaildirMessage`** object, use its **`set_flags()`** method instead, because changes made with this mailbox method will not be visible to the message object's method, **`get_flags()`**.

Added in version 3.13.

add_flag(*key*, *flag*)

On the message corresponding to *key*, set the flags specified by *flag* without changing other flags. To add more than one flag at a time, *flag* may be a string of more than one character.

Considerations for using this method versus the message object's **`add_flag()`** method are similar to those for **`set_flags()`**; see the discussion there.

Added in version 3.13.

remove_flag(*key*, *flag*)

On the message corresponding to *key*, unset the flags specified by *flag* without changing other flags. To remove more than one flag at a time, *flag* may be a string of more than one character.

Considerations for using this method versus the message object's **`remove_flag()`** method are similar to those for **`set_flags()`**; see the discussion there.

Added in version 3.13.

get_info(*key*)

Return a string containing the info for the message corresponding to *key*. This is the same as `get_message(key).get_info()` but much faster, because it does not open the message file. Use this method when iterating over the keys to determine which messages are interesting to get.

If you do have a **`MaildirMessage`** object, use its **`get_info()`** method instead, because changes made by the message's **`set_info()`** method are not reflected here until the mailbox's **`__setitem__()`** method is called.

Added in version 3.13.

set_info(*key*, *info*)

Set the info of the message corresponding to *key* to *info*. Calling `some_mailbox.set_info(key, flags)` is similar to

```
one_message = some_mailbox.get_message(key)
one_message.set_info(info)
some_mailbox[key] = one_message
```

but faster, because it does not open the message file.

If you do have a `MaildirMessage` object, use its `set_info()` method instead, because changes made with this mailbox method will not be visible to the message object's method, `get_info()`.

Added in version 3.13.

Some `Mailbox` methods implemented by **`Maildir`** deserve special remarks:

add(*message*)
__setitem__(*key*, *message*)
update(*arg*)

Advertencia

Estos métodos generan nombres de archivo únicos basados en el ID del proceso actual. Cuando se utilizan varios hilos, pueden producirse conflictos de nombres no detectados y causar la corrupción del buzón de correo a menos que se coordinen los hilos para evitar que se utilicen estos métodos para manipular el mismo buzón de correo simultáneamente.

flush()

Todos los cambios en los buzones de Maildir se aplican inmediatamente, así que este método no hace nada.

lock()
unlock()

Los buzones de Maildir no admiten (o requieren) bloqueo, por lo que estos métodos no hacen nada.

close()

`Maildir` instances do not keep any open files and the underlying mailboxes do not support locking, so this method does nothing.

get_file(*key*)

Dependiendo de la plataforma del host, puede que no sea posible modificar o eliminar el mensaje subyacente mientras el archivo retornado permanezca abierto.

Ver también

pagina web maildir de Courier [https://www.courier-mta.org/maildir.html]
Una especificación del formato. Describe una extensión común para admitir carpetas.

Utilizando el formato maildir [https://cr.yp.to/proto/maildir.html]
Notas sobre Maildir por su inventor. Incluye un esquema actualizado de creación de nombres y detalles sobre la «info» de la semántica».

`mbox` objects

class mailbox.mbox(*path*, *factory=None*, *create=True*)

Una subclase de **`Mailbox`** para los buzones de correo en formato mbox. El parámetro *factory* es un objeto invocable que acepta una representación de mensaje tipo archivo (que se comporta como si se abriera en modo binario) y retorna una representación personalizada. Si *factory* es `None`, **`mboxMessage`** se utiliza como representación de mensaje por defecto. Si *create* es `True`, el buzón de correo se crea si no existe.

El formato mbox es el formato clásico para almacenar correo en sistemas Unix. Todos los mensajes de un buzón de correo mbox se almacenan en un único archivo con el comienzo de cada mensaje indicado por una línea cuyos cinco primeros caracteres son «From».

Several variations of the mbox format exist to address perceived shortcomings in the original. In the interest of compatibility, `mbox` implements the original format, which is sometimes referred to as *mboxo*. This means that the *Content-Length* header, if present, is ignored and that any occurrences of «From « at the beginning of a line in a message body are transformed to «>From « when storing the message, although occurrences of «>From « are not transformed to «From « when reading the message.

Some `Mailbox` methods implemented by `mbox` deserve special remarks:

get_bytes(*key*, *from_*=*False*)

> Note: This method has an extra parameter (*from_*) compared with other classes. The first line of an mbox file entry is the Unix «From « line. If *from_* is False, the first line of the file is dropped.

get_file(*key*, *from_*=*False*)

> Using the file after calling `flush()` or `close()` on the `mbox` instance may yield unpredictable results or raise an exception.
>
> Note: This method has an extra parameter (*from_*) compared with other classes. The first line of an mbox file entry is the Unix «From « line. If *from_* is False, the first line of the file is dropped.

get_string(*key*, *from_*=*False*)

> Note: This method has an extra parameter (*from_*) compared with other classes. The first line of an mbox file entry is the Unix «From « line. If *from_* is False, the first line of the file is dropped.

lock()
unlock()

> Se utilizan tres mecanismos de bloqueo—el bloqueo por

puntos y, si está disponible, las llamadas del sistema **`flock()`** y **`lockf()`**.

Ver también

pagina web mbox de tin [http://www.tin.org/bin/man.cgi?section=5&topic=mbox]
Una especificación del formato, con detalles sobre el bloqueo.

Configurando el correo de Netscape en Unix: Por qué el formato de longitud de contenido es malo [https://www.jwz.org/doc/content-length.html]
Un argumento para usar el formato original mbox en lugar de una variación.

«mbox» es una familia de varios formatos de buzón de correo mutuamente incompatibles [https://www.loc.gov/preservation/digital/formats/fdd/fdd000383.shtml]
Una historia de variaciones de mbox.

`MH` objects

class mailbox.MH(*path*, *factory=None*, *create=True*)

Una subclase de **`Mailbox`** para los buzones de correo en formato MH. El parámetro *factory* es un objeto invocable que acepta una representación de mensaje tipo archivo (que se comporta como si se abriera en modo binario) y retorna una representación personalizada. Si *factory* es `None`, **`MHMessage`** se utiliza como representación de mensaje por defecto. Si *create* es `True`, el buzón de correo se crea si no existe.

MH es un formato de buzón de correo basado en un directorio inventado para el Sistema de Manejo de Mensajes MH, un agente de usuario de correo. Cada mensaje de un buzón de correo MH reside en su propio archivo. Un buzón de correo MH puede contener otros buzones de correos MH (llamados *folders*) además de los mensajes. Las carpetas pueden anidarse indefinidamente. Los buzones de correo MH también soportan *sequences*, que son listas con nombre usadas para agrupar lógicamente los mensajes sin moverlos a subcarpetas. Las secuencias se definen en un archivo llamado `.mh_sequences` en cada carpeta.

The **`MH`** class manipulates MH mailboxes, but it does not attempt to emulate all of **mh**'s behaviors. In particular, it does not modify and is not affected by the `context` or `.mh_profile` files that are used by **mh** to store its state and configuration.

`MH` instances have all of the methods of **`Mailbox`** in addition to the following:

Distinto en la versión 3.13: Supported folders that don't contain a `.mh_sequences` file.

list_folders()

Retorna una lista con los nombres de todas las carpetas.

get_folder(*folder*)

Return an **`MH`** instance representing the folder whose name is *folder*. A **`NoSuchMailboxError`** exception is raised if the folder does not exist.

add_folder(*folder*)

Create a folder whose name is *folder* and return an **`MH`** instance representing it.

remove_folder(*folder*)

Elimina la carpeta cuyo nombre es *folder*. Si la carpeta contiene algún mensaje, se lanzará una excepción **`NotEmptyError`** y la carpeta no se borrará.

get_sequences()

Retorna un diccionario de nombres de secuencias mapeadas a listas clave. Si no hay secuencias, se retorna el diccionario vacío.

set_sequences(*sequences*)

Re-define las secuencias que existen en el buzón de correo basado en *sequences*, un diccionario de nombres mapeados a listas de claves, como las retornadas por `get_sequences()`.

pack()

Renombra los mensajes en el buzón de correo según sea necesario para eliminar los huecos en la numeración. Las entradas en la lista de secuencias se actualizan correspondientemente.

Nota

Las llaves ya emitidas quedan invalidadas por esta operación y no deben utilizarse posteriormente.

Some `Mailbox` methods implemented by `MH` deserve special remarks:

remove(*key*)
delitem(*key*)
discard(*key*)

Estos métodos borran inmediatamente el mensaje. No se utiliza la convención del MH de marcar un mensaje para borrarlo poniendo una coma en su nombre.

lock()
unlock()

Se utilizan tres mecanismos de bloqueo—el bloqueo por puntos y, si está disponible, las llamadas del sistema `flock()` y `lockf()`. Para los buzones de correos MH, bloquear el buzón de correo significa bloquear el archivo `.mh_sequences` y, sólo durante la duración de cualquier operación que les afecte, bloquear los archivos de mensajes individuales.

get_file(*key*)

Dependiendo de la plataforma anfitriona, puede que no sea posible eliminar el mensaje subyacente mientras el archivo retornado permanezca abierto.

flush()

Todos los cambios en los buzones de correos de Maildir se aplican inmediatamente, así que este método no hace nada.

close()

`MH` instances do not keep any open files, so this method is equivalent to **`unlock()`**.

Ver también

nmh - Sistema de Manejo de Mensajes **[https://www.nongnu.org/nmh/]**

Página principal de **nmh**, una versión actualizada del original **mh**.

MH & nmh: Correo electrónico para usuarios y programadores **[https://rand-mh.sourceforge.io/book/]**

Un libro con licencia GPL sobre **mh** y **nmh**, con alguna información sobre el formato del buzón.

`Babyl` objects

class mailbox.Babyl(*path*, *factory=None*, *create=True*)

Una subclase de **`Mailbox`** para los buzones en formato Babyl. El parámetro *factory* es un objeto invocable que acepta una representación de mensaje tipo archivo (que se comporta como si se abriera en modo binario) y retorna una representación personalizada. Si *factory* es `None`, `BabylMessage` se utiliza como representación de mensaje por defecto. Si *create* es `True`, el buzón se crea si no existe.

Babyl es un formato de buzón de un solo archivo usado por el agente de usuario de correo de Rmail incluido en Emacs. El comienzo de un mensaje se indica con una línea que contiene los dos caracteres Control-Underscore (`'\037'`) y Control-L (`'\014'`). El final de un mensaje se indica con el comienzo del siguiente mensaje o, en el caso del último mensaje, una línea que contiene un carácter Control-Underscore (`'\037'`).

Los mensajes en un buzón de correo de Babyl tienen dos juegos de encabezados, los encabezados originales y los llamados encabezados visibles. Los encabezados visibles son típicamente un subconjunto de los encabezados originales que han sido reformateados o abreviados para ser más atractivos. Cada mensaje de un buzón de correo de Babyl también tiene una lista de *labels*, o cadenas cortas que registran información adicional sobre el mensaje, y una lista de todas las etiquetas definidas por el usuario que se encuentran en el buzón de correo se mantiene en la sección de opciones de Babyl.

`Babyl` instances have all of the methods of `Mailbox` in addition to the following:

get_labels()

> Retorna una lista de los nombres de todas las etiquetas definidas por el usuario utilizadas en el buzón de correo.
>
> **Nota**
>
> Los mensajes actuales se inspeccionan para determinar qué etiquetas existen en el buzón de correo en lugar de consultar la lista de etiquetas en la sección de opciones de Babyl, pero la sección de Babyl se actualiza cada vez que se modifica el buzón de correo.

Some `Mailbox` methods implemented by `Babyl` deserve special remarks:

get_file(*key*)

En los buzones de correos de Babyl, los encabezados de un mensaje no se almacenan contiguamente al cuerpo del mensaje. Para generar una representación tipo archivo, las cabeceras y el cuerpo se copian juntos en una instancia `io.BytesIO`, que tiene una API idéntica a la de un archivo. Como resultado, el objeto similar a un archivo es verdaderamente independiente del buzón de correo subyacente, pero no ahorra memoria en comparación con una representación en cadena.

lock()
unlock()

Se utilizan tres mecanismos de bloqueo—el bloqueo por puntos y, si está disponible, las llamadas del sistema `flock()` y `lockf()`.

Ver también

Formato de la versión 5 de los archivos de Babyl [https://quimby.gnus.org/notes/BABYL]

Una especificación del formato Babyl.

Leyendo el correo con Rmail [https://www.gnu.org/software/emacs/manual/html_node/emacs/Rmail.html]

El manual de Rmail, con cierta información sobre la semántica Babyl.

`MMDF` objects

class mailbox.MMDF(*path*, *factory=None*, *create=True*)

Una subclase de `Mailbox` para buzones de correos en formato MMDF. El parámetro *factory* es un objeto al que se puede llamar que acepta una representación de mensaje similar a un archivo (que se comporta como si se abriera en modo binario) y retorna una representación personalizada. Si *factory* es `None`, **`MMDFMessage`** se utiliza como representación de mensaje predeterminada. Si *create* es `True`, el buzón de correo se crea si no existe.

MMDF es un formato de buzón de correo de un solo archivo inventado para el centro de distribución de memorandos multicanal, un agente de transferencia de correo. Cada mensaje está en la misma forma que un mensaje de mbox, pero está entre corchetes antes y después por líneas que contienen cuatro caracteres Control-A (`'\001'`). Al igual que con el formato mbox, el principio de cada mensaje se indica mediante una línea cuyos primeros cinco caracteres son «From «, pero las apariciones adicionales de «From» no se transforman en «>From» al almacenar mensajes porque las líneas de separador de mensajes adicionales impiden confundir tales ocurrencias para los inicios de los mensajes posteriores.

Some **`Mailbox`** methods implemented by **`MMDF`** deserve special remarks:

get_bytes(*key*, *from_=False*)

> Note: This method has an extra parameter (*from_*) compared with other classes. The first line of an mbox file entry is the Unix «From « line. If *from_* is False, the first line of the file is dropped.

get_file(*key*, *from_=False*)

> Using the file after calling **`flush()`** or **`close()`** on the **`MMDF`** instance may yield unpredictable results or raise an exception.
>
> Note: This method has an extra parameter (*from_*) compared with other classes. The first line of an mbox file entry is the Unix «From « line. If *from_* is False, the first line of the file is dropped.

lock()

unlock()

Se utilizan tres mecanismos de bloqueo—el bloqueo por puntos y, si está disponible, las llamadas del sistema `flock()` y `lockf()`.

Ver también

Página web de mmdf de tin [http://www.tin.org/bin/man.cgi?section-5&topic-mmdf]
Una especificación del formato MMDF de la documentación de tin, un lector de noticias.

MMDF [https://en.wikipedia.org/wiki/MMDF]
Un artículo de Wikipedia que describe el Centro de Distribución de Memorandos Multicanal.

Message objects

class mailbox.Message(*message=None*)

A subclass of the `email.message` module's `Message`. Subclasses of `mailbox.Message` add mailbox-format-specific state and behavior.

If *message* is omitted, the new instance is created in a default, empty state. If *message* is an `email.message.Message` instance, its contents are copied; furthermore, any format-specific information is converted insofar as possible if *message* is a `Message` instance. If *message* is a string, a byte string, or a file, it should contain an RFC 5322 [https://datatracker.ietf.org/doc/html/rfc5322.html]-compliant message, which is read and parsed. Files should be open in binary mode, but text mode files are accepted for backward compatibility.

El estado y los comportamientos específicos del formato que ofrecen las subclases varían, pero en general sólo se admiten las propiedades que no son específicas de un buzón de correo concreto (aunque presumiblemente las propiedades son específicas de un formato de buzón de correo concreto). Por ejemplo, no se conservan las compensaciones de archivos para los formatos de buzón de correo de un solo archivo ni los nombres de archivo para los formatos de buzón de correo basados en directorios, porque sólo son aplicables al buzón de correo original. Pero sí se conservan las declaraciones tales como si un mensaje ha sido leído por el usuario o marcado como importante, porque se aplican al propio mensaje.

There is no requirement that **`Message`** instances be used to represent messages retrieved using **`Mailbox`** instances. In some situations, the time and memory required to generate **`Message`** representations might not be acceptable. For such situations, **`Mailbox`** instances also offer string and file-like representations, and a custom message factory may be specified when a **`Mailbox`** instance is initialized.

`MaildirMessage` objects

class mailbox.MaildirMessage(*message=None*)

Un mensaje con comportamientos específicos de Maildir. El parámetro *message* tiene el mismo significado que con el constructor **`Message`**.

Típicamente, una aplicación de agente de usuario de correo mueve todos los mensajes del subdirectorio `new` al subdirectorio `cur` después de la primera vez que el usuario abre y cierra el buzón de coreo, registrando que los mensajes son antiguos, tanto si han sido leídos como si no. Cada mensaje en `cur` tiene una sección «info» añadida a su nombre de archivo para almacenar información sobre su estado. (Algunos lectores de correo también pueden añadir una sección «info» a los mensajes en `new`.) La sección «info» puede tomar una de dos formas: puede contener «2», seguido de una lista de flags estandarizados (por ejemplo, «2,FR») o puede contener «1», seguido de la llamada información experimental. Los flags normalizados para los mensajes de Maildir son los siguientes:

Flag	Significado	Explicación
	Borrador	Bajo composición

F	Marcado	Marcado como importante
P	Aprobado	Enviado, reenviado o rebotado
R	Contestado	Contestado a
S	Visto	Leído
T	Destruido	Marcado para su posterior eliminación

`MaildirMessage` instances offer the following methods:

get_subdir()

Retorna «new» (si el mensaje debe ser almacenado en el subdirectorio `new`) o «cur» (si el mensaje debe ser almacenado en el subdirectorio `cur`).

Nota

A message is typically moved from `new` to `cur` after its mailbox has been accessed, whether or not the message has been read. A message `msg` has been read if `"S" in msg.get_flags()` is `True`.

set_subdir(*subdir*)

Establece el subdirectorio en el que debe almacenarse el mensaje. El parámetro *subdir* debe ser «new» o «cur».

get_flags()

Retorna una cadena que especifica los flags que están actualmente establecidos. Si el mensaje cumple con el formato estándar de Maildir, el resultado es la concatenación en orden alfabético de cero o una ocurrencia de cada una de los flags `'D'`, `'F'`, `'P'`, `'R'`, `'S'`, y `'T'`. La cadena vacía se retorna si no hay flags o si «info» contiene semántica experimental.

set_flags(*flags*)

Establece los flags especificados por *flags* y desactiva todas las demás.

add_flag(*flag*)

Establece lo(s) flag(s) especificado(s) por *flags* sin cambiar otros flags. Para añadir más de un indicador a la vez, *flag* puede ser una cadena de más de un carácter. La «info» actual se sobrescribe si contiene o no información experimental en lugar de flags.

remove_flag(*flag*)

Unset the flag(s) specified by *flag* without changing other flags. To remove more than one flag at a time, *flag* may be a string of more than one character. If «info» contains experimental information rather than flags, the current «info» is not modified.

get_date()

Retorna la fecha de entrega del mensaje como un número de punto flotante que representa los segundos desde la época.

set_date(*date*)

Establece la fecha de entrega del mensaje en *date*, un número de punto flotante que representa los segundos desde la época.

get_info()

Retorna una cadena que contiene la «info» de un mensaje. Esto es útil para acceder y modificar la «info» que es experimental (es decir, no una lista de flags).

set_info(*info*)

Establece «info» en *info*, que debería ser una cadena.

When a **`MaildirMessage`** instance is created based upon an **`mboxMessage`** or **`MMDFMessage`** instance, the *Status* and *X-Status* headers are omitted and the following conversions take place:

Estado resultante	**`mboxMessage`** o **`MMDFMessage`**

subdirectorio «cur»	flag O
flag F	flag F
flag R	flag A
flag S	flag R
flag T	flag D

When a **`MaildirMessage`** instance is created based upon an `MHMessage` instance, the following conversions take place:

Estado resultante	**`MHMessage`**
subdirectorio «cur»	secuencia «*unseen*» (no vista)
subdirectorio «cur» y marcador S	no hay secuencia «*unseen*» (invisible)
flag F	secuencia «*flagged*» (marcada)
flag R	secuencia «*replied*» (respondida)

When a **`MaildirMessage`** instance is created based upon a `BabylMessage` instance, the following conversions take place:

Estado resultante	**`BabylMessage`**
subdirectorio «cur»	etiqueta «*unseen*» (invisible)
subdirectorio «cur» y marcador S	no hay etiqueta «*unseen*» (invisible)
flag P	etiqueta «*forwarded*» o «*resent*» (reenviado)
flag R	etiqueta de «*answered*» (contestado)
flag T	etiqueta «deleted» (borrado)

`mboxMessage` objects

class mailbox.mboxMessage(*message*=*None*)

Un mensaje con comportamientos específicos de mbox. El parámetro *message* tiene el mismo significado que con el constructor `Message`.

Los mensajes en un buzón de correo de mbox se almacenan juntos en un solo archivo. La dirección del sobre del remitente y la hora de entrega se almacenan normalmente en una línea que comienza con «From» que se utiliza para indicar el comienzo de un mensaje, aunque hay una variación considerable en el formato exacto de estos datos entre las implementaciones de mbox. Los flags del estado del mensaje, como por ejemplo si ha sido leído o marcado como importante, se almacenan típicamente en las cabeceras *Status* y *X-Status*.

Los flags convencionales para los mensajes de mbox son los siguientes:

Flag	**Significado**	**Explicación**
R	Leído	Leído
O	Antiguo	Anteriormente detectado por MUA
D	Borrado	Marcado para su posterior eliminación
F	Marcado	Marcado como importante
A	Respondido	Contestado a

Los flags «R» y «O» se almacenan en el encabezado *Status* y los flags «D», «F» y «A» se almacenan en el encabezado *X-Status*. Los flags y los encabezados aparecen típicamente en el orden mencionado.

`mboxMessage` instances offer the following methods:

get_from()

Retorna una cadena que representa la línea «From» que marca el inicio del mensaje en un buzón de correo de mbox. El «From» inicial y la nueva línea final están excluidas.

set_from(*from_*, *time_ =None*)

Set the «From « line to *from_*, which should be specified without a leading «From « or trailing newline. For convenience, *time_* may be specified and will be formatted appropriately and appended to *from_*. If *time_* is specified, it should be a **`time.struct_time`** instance, a tuple suitable for passing to **`time.strftime()`**, or `True` (to use **`time.gmtime()`**).

get_flags()

Retorna una cadena que especifica los flags que están actualmente establecidos. Si el mensaje cumple con el formato convencional, el resultado es la concatenación en el siguiente orden de cero o una ocurrencia de cada uno de los flags `'R'`, `'O'`, `'D'`, `'F'`, and `'A'`.

set_flags(*flags*)

Establece los flags especificadas por *flags* y desactiva todos las demás. El parámetro *flags* debe ser la concatenación en cualquier orden de cero o más ocurrencias de cada uno de los flags `'R'`, `'O'`, `'D'`, `'F'`, and `'A'`.

add_flag(*flag*)

Establece lo(s) flag(s) especificado(s) por *flag* sin cambiar otros flags. Para añadir más de un indicador a la vez, *flag* puede ser una cadena de más de un carácter.

remove_flag(*flag*)

Unset the flag(s) specified by *flag* without changing other flags. To remove more than one flag at a time, *flag* may be a string of more than one character.

When an **`mboxMessage`** instance is created based upon a **`MaildirMessage`** instance, a «From « line is generated based upon the **`MaildirMessage`** instance's delivery date, and the following conversions take place:

Estado resultante	**Estado de `MaildirMessage`**
flag R	flag S
flag O	subdirectorio «cur»
flag D	flag T
flag F	flag F
flag A	flag R

When an **`mboxMessage`** instance is created based upon an **`MHMessage`** instance, the following conversions take place:

Estado resultante	**Estado `MHMessage`**
flag R y flag O	no hay una secuencia «*unseen*» (invisible)
flag O	secuencia «*unseen*» (no vista)

flag F	secuencia «*flagged*» (marcada)
flag A	secuencia «*replied*» (respondida)

When an `mboxMessage` instance is created based upon a `BabylMessage` instance, the following conversions take place:

Estado resultante	`BabylMessage`
flag R y flag O	no hay una etiqueta «*unseen*» (invisible)
flag O	etiqueta «*unseen*» (invisible)
flag D	etiqueta «deleted» (borrado)
flag A	etiqueta de «*answered*» (contestado)

When a `mboxMessage` instance is created based upon an `MMDFMessage` instance, the «From « line is copied and all flags directly correspond:

Estado resultante	Estado de `MMDFMessage`
flag R	flag R
flag O	flag O
flag D	flag D
flag F	flag F
flag A	flag A

`MHMessage` objects

class mailbox.MHMessage(*message=None*)

Un mensaje con comportamientos específicos de la HM. El parámetro *message* tiene el mismo significado que con el constructor `Message`.

Los mensajes de MH no soportan marcas o flags en el sentido tradicional, pero sí secuencias, que son agrupaciones lógicas de mensajes arbitrarios. Algunos programas de lectura de correo (aunque no los estándares **mh** y **nmh**) usan secuencias de manera muy similar a los flags que se usan con otros formatos, como sigue:

Secuencia	Explicación
unseen (no visto)	No leído, pero previamente detectado por la MUA
replied (contestado)	Contestado

flagged (marcado)	Marcado como importante

`MHMessage` instances offer the following methods:

get_sequences()

Retorna una lista de los nombres de las secuencias que incluyen este mensaje.

set_sequences(*sequences*)

Establece la lista de secuencias que incluyen este mensaje.

add_sequence(*sequence*)

Añade *sequence* a la lista de secuencias que incluyen este mensaje.

remove_sequence(*sequence*)

Elimina *sequence* de la lista de secuencias que incluyen este mensaje.

When an **`MHMessage`** instance is created based upon a `MaildirMessage` instance, the following conversions take place:

Estado resultante	**Estado de `MaildirMessage`**
secuencia «*unseen*» (no vista)	no hay indicador S
secuencia «*replied*» (respondida)	flag R
secuencia «*flagged*» (marcada)	flag F

When an **`MHMessage`** instance is created based upon an `mboxMessage` or `MMDFMessage` instance, the *Status* and *X-Status* headers are omitted and the following conversions take place:

Estado resultante	**Estado `mboxMessage` o `MMDFMessage`**
secuencia «*unseen*» (no vista)	Sin indicador R
secuencia «*replied*» (respondida)	flag A
secuencia «*flagged*» (marcada)	flag F

When an **`MHMessage`** instance is created based upon a `BabylMessage` instance, the following conversions take place:

Estado resultante	`BabylMessage`
Secuencia «*unseen*» (invisible)	etiqueta «*unseen*» (no vista)
Secuencia «*replied*» (respondida)	etiqueta de «*answered*» (contestado)

`BabylMessage` objects

class mailbox.BabylMessage(*message=None*)

Un mensaje con comportamientos específicos de Babyl. El parámetro *message* tiene el mismo significado que con el constructor `Message`.

Ciertas etiquetas de mensajes, llamadas *attributes*, están definidas por convención para tener significados especiales. Los atributos son los siguientes:

Etiqueta	Explicación
unseen (no visto)	No leído, pero previamente detectado por la MUA
deleted (borrado)	Marcado para su posterior eliminación
filed (archivado)	Copiado a otro archivo o buzón de correo
answered (contestado)	Contestado a
forwarded (reenviado)	Reenviado
edited (editado)	Modificado por el usuario
resent (reenviado)	Reenviado

By default, Rmail displays only visible headers. The **`BabylMessage`** class, though, uses the original headers because they are more complete. Visible headers may be accessed explicitly if desired.

`BabylMessage` instances offer the following methods:

get_labels()

Retorna una lista de etiquetas en el mensaje.

set_labels(*labels*)

Establece la lista de etiquetas del mensaje en *labels*.

add_label(*label*)

Añade *label* a la lista de etiquetas del mensaje.

remove_label(*label*)

Eliminar *label* de la lista de etiquetas del mensaje.

get_visible()

Return a `Message` instance whose headers are the message’s visible headers and whose body is empty.

set_visible(*visible*)

Establece los encabezados visibles del mensaje para que sean los mismos que los del *message*. El parámetro *visible* debe ser una instancia `Message`, una instancia `email.message.Message`, una cadena, o un objeto tipo archivo (que debe estar abierto en modo texto).

update_visible()

When a **`BabylMessage`** instance’s original headers are modified, the visible headers are not automatically modified to correspond. This method updates the visible headers as follows: each visible header with a corresponding original header is set to the value of the original header, each visible header without a corresponding original header is removed, and any of *Date*, *From*, *Reply-To*, *To*, *CC*, and *Subject* that are present in the original headers but not the visible headers are added to the visible headers.

When a **`BabylMessage`** instance is created based upon a `MaildirMessage` instance, the following conversions take place:

Estado resultante	**Estado de `MaildirMessage`**
etiqueta «*unseen*» (invisible)	no hay indicador S
etiqueta «deleted» (borrado)	flag T
etiqueta de «*answered*» (contestado)	flag R
etiqueta «*forwarded*» (reenviado)	flag P

When a **`BabylMessage`** instance is created based upon an `mboxMessage` or `MMDFMessage` instance, the *Status* and *X-Status* headers are omitted and the following conversions take place:

Estado resultante	`mboxMessage` o `MMDFMessage`
etiqueta «*unseen*» (invisible)	sin indicador R
etiqueta «deleted» (borrado)	flag D
etiqueta de «*answered*» (contestado)	flag A

When a **`BabylMessage`** instance is created based upon an `MHMessage` instance, the following conversions take place:

Estado resultante	`MHMessage`
etiqueta «*unseen*» (invisible)	Secuencia «*unseen*» (no vista)
etiqueta de «*answered*» (contestado)	Secuencia «*replied*» (respondida)

`MMDFMessage` objects

class mailbox.MMDFMessage(*message*=*None*)

Un mensaje con comportamientos específicos de MMDF. El parámetro *message* tiene el mismo significado que con el constructor `Message`.

Al igual que los mensajes en un buzón de correo de mbox, los mensajes MMDF se almacenan con la dirección del remitente y la fecha de entrega en una línea inicial que comienza con «From». De la misma manera, los flags que indican el estado del mensaje se almacenan típicamente en las cabeceras *Status* y *X-Status*.

Los flags convencionales para los mensajes MMDF son idénticos a las de los mensajes de mbox y son los siguientes:

Flag	Significado	Explicación
R	Leído	
O	Antiguo	Anteriormente detectado por MUA
D	Borrado	Marcado para su posterior eliminación
F	Marcado	Marcado como importante
A	Contestado	Respondido a

Los flags «R» y «O» se almacenan en el encabezado *Status* y los flags «D», «F» y «A» se almacenan en el encabezado *X-Status*. Los flags y los encabezados aparecen típicamente en el orden mencionado.

`MMDFMessage` instances offer the following methods, which are identical to those offered by `mboxMessage`:

get_from()

Retorna una cadena que representa la línea «From» que marca el inicio del mensaje en un buzón de correo de mbox. El «From» inicial y la nueva línea final están excluidas.

set_from(*from_*, *time_ =None*)

Set the «From « line to *from_*, which should be specified without a leading «From « or trailing newline. For convenience, *time_* may be specified and will be formatted appropriately and appended to *from_*. If *time_* is specified, it should be a `time.struct_time` instance, a tuple suitable for passing to `time.strftime()`, or `True` (to use `time.gmtime()`).

get_flags()

Retorna una cadena que especifica los flags que están actualmente establecidos. Si el mensaje cumple con el formato convencional, el resultado es la concatenación en el siguiente orden de cero o una ocurrencia de cada uno de los flags `'R'`, `'O'`, `'D'`, `'F'`, and `'A'`.

set_flags(*flags*)

Establece los flags especificadas por *flags* y desactiva todos las demás. El parámetro *flags* debe ser la concatenación en cualquier orden de cero o más ocurrencias de cada uno de los flags `'R'`, `'O'`, `'D'`, `'F'`, and `'A'`.

add_flag(*flag*)

Establece lo(s) flag(s) especificado(s) por *flag* sin cambiar otros flags. Para añadir más de un indicador a la vez, *flag* puede ser una cadena de más de un carácter.

remove_flag(*flag*)

Unset the flag(s) specified by *flag* without changing other flags. To remove more than one flag at a time, *flag* may be a string of more than one character.

When an **`MMDFMessage`** instance is created based upon a **`MaildirMessage`** instance, a «From « line is generated based upon the **`MaildirMessage`** instance's delivery date, and the following conversions take place:

Estado resultante	Estado de **`MaildirMessage`**
flag R	flag S
flag O	subdirectorio «cur»
flag D	flag T
flag F	flag F
flag A	flag R

When an **`MMDFMessage`** instance is created based upon an **`MHMessage`** instance, the following conversions take place:

Estado resultante	Estado **`MHMessage`**
flag R y flag O	no hay una secuencia «*unseen*» (invisible)
flag O	secuencia «*unseen*» (no vista)
flag F	secuencia «*flagged*» (marcada)
flag A	secuencia «*replied*» (respondida)

When an **`MMDFMessage`** instance is created based upon a **`BabylMessage`** instance, the following conversions take place:

Estado resultante	Estado **`BabylMessage`**
flag R y flag O	no hay una etiqueta «*unseen*» (invisible)
flag O	etiqueta «*unseen*» (invisible)
flag D	etiqueta «deleted» (borrado)
flag A	etiqueta de «*answered*» (contestado)

When an **`MMDFMessage`** instance is created based upon an **`mboxMessage`** instance, the «From « line is copied and all flags directly correspond:

Estado resultante	Estado de **`mboxMessage`**
flag R	flag R

flag O
flag D
flag F
flag A

Excepciones

The following exception classes are defined in the **`mailbox`** module:

exception mailbox.Error

The base class for all other module-specific exceptions.

exception mailbox.NoSuchMailboxError

Se lanza cuando se espera un buzón de correo pero no se encuentra, como cuando se instancia una subclase **`Mailbox`** con una ruta que no existe (y con el parámetro *create* establecido en `False`), o cuando se abre una carpeta que no existe.

exception mailbox.NotEmptyError

Se lanza cuando un buzón de correo no está vacío, pero se espera que lo esté, como cuando se elimina una carpeta que contiene mensajes.

exception mailbox.ExternalClashError

Raised when some mailbox-related condition beyond the control of the program causes it to be unable to proceed, such as when failing to acquire a lock that another program already holds, or when a uniquely generated file name already exists.

exception mailbox.FormatError

Se lanza cuando los datos de un archivo no pueden ser analizados, como cuando una instancia **`MH`** intenta leer un archivo `.mh_sequences` corrupto.

Ejemplos

Un simple ejemplo de impresión de los temas de todos los mensajes en un buzón de correo que parecen interesantes:

```
import mailbox
for message in mailbox.mbox('~/mbox'):
    subject = message['subject']       # Could possibly
    if subject and 'python' in subject.lower():
        print(subject)
```

Para copiar todo el correo de un buzón de Babyl a un buzón de MH, convirtiendo toda la información de formato específico que puede ser convertida:

```
import mailbox
destination = mailbox.MH('~/Mail')
destination.lock()
for message in mailbox.Babyl('~/RMAIL'):
    destination.add(mailbox.MHMessage(message))
destination.flush()
destination.unlock()
```

Este ejemplo clasifica el correo de varias listas de correo en diferentes buzones, teniendo cuidado de evitar la corrupción del correo debido a la modificación simultánea por otros programas, la pérdida de correo debido a la interrupción del programa, o la terminación prematura debido a mensajes malformados en el buzón:

```
import mailbox
import email.errors

list_names = ('python-list', 'python-dev', 'python-bugs'

boxes = {name: mailbox.mbox('~/email/%s' % name) for nam
inbox = mailbox.Maildir('~/Maildir', factory=None)

for key in inbox.iterkeys():
    try:
        message = inbox[key]
    except email.errors.MessageParseError:
        continue                # The message is malform

    for name in list_names:
        list_id = message['list-id']
        if list_id and name in list_id:
            # Get mailbox to use
            box = boxes[name]

            # Write copy to disk before removing origina
            # If there's a crash, you might duplicate a
            # that's better than losing a message comple
            box.lock()
            box.add(message)
            box.flush()
            box.unlock()

            # Remove original message
            inbox.lock()
            inbox.discard(key)
            inbox.flush()
            inbox.unlock()
            break               # Found destination, so

for box in boxes.itervalues():
    box.close()
```

`mimetypes` — Map filenames to MIME types

Código fuente: Lib/mimetypes.py [https://github.com/python/cpython/tree/3.14/Lib/mimetypes.py]

The **`mimetypes`** module converts between a filename or URL and the MIME type associated with the filename extension. Conversions are provided from filename to MIME type and from MIME type to filename extension; encodings are not supported for the latter conversion.

El módulo proporciona una clase y varias funciones de conveniencia. Las funciones son la interfaz normal de este módulo, pero algunas aplicaciones pueden estar interesadas en la clase también.

Las funciones que se describen a continuación constituyen la interfaz principal de este módulo. Si el módulo no ha sido inicializado, llamarán **`init()`** si se basan en la información que **`init()`** establece.

mimetypes.guess_type(*url*, *strict=True*)

Adivina el tipo de un archivo basado en su nombre de archivo, ruta o URL, dado por *url*. La URL puede ser una cadena o un objeto path-like object.

El valor de retorno es una tupla `(type, encoding)` donde *type* es `None` si el tipo no puede ser adivinado (por sufijo ausente o desconocido) o una cadena de la forma `type/subtype'`, utilizable para un encabezado MIME *content-type*.

encoding es `None` para no codificar o el nombre del programa usado para codificar (por ejemplo **compress** o **gzip**). La codificación es adecuada para ser usada como una cabecera

de *Content-Encoding*, **no** como una cabecera de *Content-Transfer-Encoding*. Los mapeos son conducidos por tablas. Los sufijos de codificación son sensibles a las mayúsculas y minúsculas; los sufijos de tipo se prueban primero distinguiendo entre mayúsculas y minúsculas, y luego sin dicha distinción.

The optional *strict* argument is a flag specifying whether the list of known MIME types is limited to only the official types registered with IANA [https://www.iana.org/assignments/media-types/media-types.xhtml]. However, the behavior of this module also depends on the underlying operating system. Only file types recognized by the OS or explicitly registered with Python's internal database can be identified. When *strict* is `True` (the default), only the IANA types are supported; when *strict* is `False`, some additional non-standard but commonly used MIME types are also recognized.

Distinto en la versión 3.8: Added support for *url* being a path-like object.

Soft deprecated since version 3.13: Passing a file path instead of URL is soft deprecated. Use **`guess_file_type()`** for this.

mimetypes.guess_file_type(*path*, ***, *strict=True*)

Guess the type of a file based on its path, given by *path*. Similar to the **`guess_type()`** function, but accepts a path instead of URL. Path can be a string, a bytes object or a path-like object.

Added in version 3.13.

mimetypes.guess_all_extensions(*type*, *strict=True*)

Guess the extensions for a file based on its MIME type, given by *type*. The return value is a list of strings giving all possible filename extensions, including the leading dot (`'.'`). The extensions are not guaranteed to have been associated with any particular data stream, but would be mapped to the MIME type *type* by **`guess_type()`** and

`guess_file_type()`.

El argumento opcional *strict* tiene el mismo significado que con la función `guess_type()`.

mimetypes.guess_extension(*type*, *strict=True*)

Guess the extension for a file based on its MIME type, given by *type*. The return value is a string giving a filename extension, including the leading dot (`'.'`). The extension is not guaranteed to have been associated with any particular data stream, but would be mapped to the MIME type *type* by `guess_type()` and `guess_file_type()`. If no extension can be guessed for *type*, `None` is returned.

El argumento opcional *strict* tiene el mismo significado que con la función `guess_type()`.

Algunas funciones adicionales y elementos de datos están disponibles para controlar el comportamiento del módulo.

mimetypes.init(*files=None*)

Inicializa las estructuras de datos internos. Si se proporciona *files* debe ser una secuencia de nombres de archivos que deben utilizarse para aumentar el mapa de tipo por defecto. Si se omite, los nombres de archivo a utilizar se toman de `knownfiles`; en Windows, se cargan las configuraciones actuales del registro. Cada archivo nombrado en *files* o **`knownfiles`** tiene prioridad sobre los nombrados antes de él. Se permite llamar repetidamente a `init()`.

Si se especifica una lista vacía para *files* se evitará que se apliquen los valores predeterminados del sistema: sólo estarán presentes los valores conocidos de una lista incorporada.

Si *files* es `None` la estructura interna de datos se reconstruye completamente a su valor inicial por defecto. Esta es una operación estable y producirá los mismos resultados cuando se llame varias veces.

Distinto en la versión 3.2: Anteriormente, la configuración del registro de Windows se ignoraba.

mimetypes.read_mime_types(*filename*)

Carga el mapa de tipo dado en el archivo *filename*, si existe. El mapa de tipos es retornado como un diccionario que mapea las extensiones de los nombres de archivo, incluyendo el punto inicial (`'.'`), a las cadenas de la forma `'type/subtype'`. Si el archivo *filename* no existe o no puede ser leído, se retorna `None`.

mimetypes.add_type(*type*, *ext*, *strict=True*)

Añade un mapeo del tipo MIME *type* a la extensión *ext*. Cuando la extensión ya se conoce, el nuevo tipo reemplazará al antiguo. Cuando el tipo ya se conoce la extensión se añadirá a la lista de extensiones conocidas.

Cuando *strict* es `True` (el valor por defecto), el mapeo se añadirá a los tipos MIME oficiales, de lo contrario a los no estándar.

mimetypes.inited

Flag que indica si se han inicializado o no las estructuras de datos globales. Esto se establece como «True» por **`init()`**.

mimetypes.knownfiles

Lista de los nombres de los archivos de mapas de tipo comúnmente instalados. Estos archivos se llaman típicamente `mime.types` y se instalan en diferentes lugares por diferentes paquetes.

mimetypes.suffix_map

Diccionario que mapea sufijos a sufijos. Se utiliza para permitir el reconocimiento de archivos codificados cuya codificación y tipo se indican con la misma extensión. Por ejemplo, la extensión `.tgz` se mapea a `.tar.gz` para permitir que la codificación y el tipo se reconozcan por separado.

mimetypes.encodings_map

El diccionario mapea las extensiones de los nombres de archivo a los tipos de codificación.

mimetypes.types_map

Diccionario que mapea extensiones de los nombres de archivo a tipos MIME.

mimetypes.common_types

Diccionario que mapea extensiones de los nombres de archivo a tipos MIME no estándar, pero comúnmente encontrados.

Un ejemplo de utilización del módulo:

```
>>> import mimetypes
>>> mimetypes.init()
>>> mimetypes.knownfiles
['/etc/mime.types', '/etc/httpd/mime.types', ... ]
>>> mimetypes.suffix_map['.tgz']
'.tar.gz'
>>> mimetypes.encodings_map['.gz']
'gzip'
>>> mimetypes.types_map['.tgz']
'application/x-tar-gz'
```

MimeTypes objects

The `MimeTypes` class may be useful for applications which may want more than one MIME-type database; it provides an interface similar to the one of the **`mimetypes`** module.

class mimetypes.MimeTypes(*filenames=(), strict=True*)

Esta clase representa una base de datos de tipos MIME. Por defecto, proporciona acceso a la misma base de datos que el resto de este módulo. La base de datos inicial es una copia de la proporcionada por el módulo, y puede ser extendida cargando archivos adicionales de tipo `mime.types` en la base de datos usando los métodos `read()` o `readfp()`. Los diccionarios de mapeo también pueden ser borrados antes de cargar datos adicionales si no se desean los datos por defecto.

El parámetro opcional *filenames* puede utilizarse para hacer que se carguen archivos adicionales «encima» de la base de datos predeterminada.

suffix_map

> El diccionario mapea sufijos a sufijos. Se utiliza para permitir el reconocimiento de archivos codificados cuya codificación y tipo se indican con la misma extensión. Por ejemplo, la extensión `.tgz` se mapea a `.tar.gz` para permitir que la codificación y el tipo se reconozcan por separado. Esto es inicialmente una copia del global **`suffix_map`** definido en el módulo.

encodings_map

> El diccionario mapea las extensiones de los nombres de archivo a los tipos de codificación. Es inicialmente una copia del global **`encodings_map`** definido en el módulo.

types_map

> Una tupla que contiene dos diccionarios, mapeando las extensiones de los nombres de archivo a los tipos MIME: el primer diccionario es para los tipos no estándar y el segundo para los tipos estándar. Están inicializados por **`common_types`** y **`types_map`**.

types_map_inv

> Una tupla que contiene dos diccionarios, mapeando los tipos MIME a una lista de extensiones de nombres de archivos: el primer diccionario es para los tipos no estándar y el segundo para los tipos estándar. Están inicializados por **`common_types`** y **`types_map`**.

guess_extension(*type*, *strict=True*)

> Similar a la función **`guess_extension()`**, usando las

tablas almacenadas como parte del objeto.

guess_type(*url*, *strict=True*)

Similar a la función `guess_type()`, usando las tablas almacenadas como parte del objeto.

guess_file_type(*path*, *, *strict=True*)

Similar to the `guess_file_type()` function, using the tables stored as part of the object.

Added in version 3.13.

guess_all_extensions(*type*, *strict=True*)

Similar a la función `guess_all_extensions()`, usando las tablas almacenadas como parte del objeto.

read(*filename*, *strict=True*)

Carga información MIME de un archivo llamado *filename*. Esto usa `readfp()` para analizar el archivo.

Si *strict* es `True`, la información se añadirá a la lista de tipos estándar, si no a la lista de tipos no estándar.

readfp(*fp*, *strict=True*)

Carga información de tipo MIME de un archivo abierto *fp*. El archivo debe tener el formato de los archivos estándar `mime.types`.

Si *strict* es `True`, la información se va a añadir a la lista de tipos estándar, de otro modo se añadirá a la lista de tipos no estándar.

read_windows_registry(*strict=True*)

Carga información desde el registro de Windows del tipo de metadato MIME.

Availability: Windows.

Si *strict* es `True`, la información se va a añadir a la lista de tipos estándar, de otro modo se añadirá a la lista de tipos no estándar.

Added in version 3.2.

add_type(*type*, *ext*, *strict=True*)

Add a mapping from the MIME type *type* to the extension *ext*. Valid extensions start with a "." or are empty. When the extension is already known, the new type will replace the old one. When the type is already known the extension will be added to the list of known extensions.

Cuando *strict* es `True` (el valor por defecto), el mapeo se añadirá a los tipos MIME oficiales, de lo contrario a los no estándar.

Deprecated since version 3.14, will be removed in version 3.16: Invalid, undotted extensions will raise a **`ValueError`** in Python 3.16.

Command-line usage

The **`mimetypes`** module can be executed as a script from the command line.

```
python -m mimetypes [-h] [-e] [-l] type [type ...]
```

The following options are accepted:

-h
--help

Show the help message and exit.

-e
--extension

Guess extension instead of type.

-l

--lenient

Additionally search for some common, but non-standard types.

By default the script converts MIME types to file extensions. However, if `--extension` is specified, it converts file extensions to MIME types.

For each `type` entry, the script writes a line into the standard output stream. If an unknown type occurs, it writes an error message into the standard error stream and exits with the return code `1`.

Command-line example

Here are some examples of typical usage of the **`mimetypes`** command-line interface:

```
$ # get a MIME type by a file name
$ python -m mimetypes filename.png
type: image/png encoding: None

$ # get a MIME type by a URL
$ python -m mimetypes https://example.com/filename.txt
type: text/plain encoding: None

$ # get a complex MIME type
$ python -m mimetypes filename.tar.gz
type: application/x-tar encoding: gzip

$ # get a MIME type for a rare file extension
$ python -m mimetypes filename.pict
error: unknown extension of filename.pict

$ # now look in the extended database built into Python
$ python -m mimetypes --lenient filename.pict
type: image/pict encoding: None
```

```
$ # get a file extension by a MIME type
$ python -m mimetypes --extension text/javascript
.js

$ # get a file extension by a rare MIME type
$ python -m mimetypes --extension text/xul
error: unknown type text/xul

$ # now look in the extended database again
$ python -m mimetypes --extension --lenient text/xul
.xul

$ # try to feed an unknown file extension
$ python -m mimetypes filename.sh filename.nc filename.x
type: application/x-sh encoding: None
type: application/x-netcdf encoding: None
error: unknown extension of filename.xxx

$ # try to feed an unknown MIME type
$ python -m mimetypes --extension audio/aac audio/opus a
.aac
.opus
error: unknown type audio/future
```

`base64` — Base16, Base32, Base64, Base85 Data Encodings

Código fuente: Lib/base64.py [https://github.com/python/cpython/tree/3.14/Lib/base64.py]

This module provides functions for encoding binary data to printable ASCII characters and decoding such encodings back to binary data. This includes the encodings specified in **RFC 4648** [https://datatracker.ietf.org/doc/html/rfc4648.html] (Base64, Base32 and Base16) and the non-standard Base85 encodings.

Hay dos interfaces proporcionadas por este módulo. La interfaz moderna admite la codificación de objetos similares a bytes a ASCII **`bytes`** y la decodificación de objetos similares a bytes o cadenas que contienen ASCII a **`bytes`**. Se admiten los dos alfabetos base 64 definidos en **RFC 4648** [https://datatracker.ietf.org/doc/html/rfc4648.html] (normal y seguro para URL y sistema de archivos).

The legacy interface does not support decoding from strings, but it does provide functions for encoding and decoding to and from file objects. It only supports the Base64 standard alphabet, and it adds newlines every 76 characters as per **RFC 2045** [https://datatracker.ietf.org/doc/html/rfc2045.html]. Note that if you are looking for **RFC 2045** [https://datatracker.ietf.org/doc/html/rfc2045.html] support you probably want to be looking at the **`email`** package instead.

Distinto en la versión 3.3: Las cadenas de caracteres Unicode de solo ASCII ahora son aceptadas por las funciones de decodificación de la interfaz moderna.

Distinto en la versión 3.4: Cualquier objeto similar a bytes ahora son aceptados por todas las funciones de codificación y decodificación en este módulo. Ascii85/Base85 soporte agregado.

RFC 4648 Encodings

The **RFC 4648** [https://datatracker.ietf.org/doc/html/rfc4648.html] encodings are suitable for encoding binary data so that it can be safely sent by email, used as parts of URLs, or included as part of an HTTP POST request.

base64.b64encode(*s*, *altchars=None*)

Codifica el objeto similar a bytes *s* utilizando Base64 y retorna los **`bytes`** codificados.

Optional *altchars* must be a bytes-like object of length 2 which specifies an alternative alphabet for the `+` and `/` characters. This allows an application to e.g. generate URL or filesystem safe Base64 strings. The default is `None`, for which the standard Base64 alphabet is used.

May assert or raise a **`ValueError`** if the length of *altchars* is not 2. Raises a **`TypeError`** if *altchars* is not a bytes-like object.

base64.b64decode(*s*, *altchars=None*, *validate=False*)

Decodifica el objeto similar a bytes codificado en Base64 o cadena de caracteres ASCII *s* y retorna los **`bytes`** decodificados.

Optional *altchars* must be a bytes-like object or ASCII string of length 2 which specifies the alternative alphabet used instead of the `+` and `/` characters.

Una excepción **`binascii.Error`** se lanza si *s* está incorrectamente rellenado (*padded*).

Si *validate* es `False` (el valor predeterminado), los caracteres que no están en el alfabeto normal de base 64 ni en el alfabeto alternativo se descartan antes de la verificación del relleno. Si *validate* es `True`, estos caracteres no alfabéticos en la entrada dan como resultado **`binascii.Error`**.

Para más información sobre la verificación estricta de base64, véase `binascii.a2b_base64()`

May assert or raise a `ValueError` if the length of *altchars* is not 2.

base64.standard_b64encode(*s*)

Codifica el objeto similar a bytes *s* usando el alfabeto estándar Base64 y retorna los `bytes` codificados.

base64.standard_b64decode(*s*)

Decodifica un bytes-like object o cadena de caracteres ASCII *s* utilizando el alfabeto estándar Base64 y retorna los `bytes` decodificados.

base64.urlsafe_b64encode(*s*)

Codifica el objeto similar a bytes *s* usando el alfabeto seguro para URL y sistemas de archivos, que sustituye a `-` en lugar de `+` y `_` en lugar de `/` en el alfabeto estándar de Base64, y retorna los `bytes` codificados. El resultado aún puede contener `=`.

base64.urlsafe_b64decode(*s*)

Decodifica objeto similar a bytes o cadena de caracteres ASCII *s* utilizando el alfabeto seguro para URL y sistema de archivos, que sustituye `-` en lugar de `+` y `_` en lugar de `/` en el alfabeto estándar de Base64, y retorna los `bytes` decodificados.

base64.b32encode(*s*)

Codifica el objeto similar a bytes *s* utilizando Base32 y retorna los `bytes` codificados.

base64.b32decode(*s*, *casefold=False*, *map01=None*)

Decodifica el objeto similar a bytes codificado en Base32 o cadena de caracteres ASCII *s* y retorna los `bytes` decodificados.

El opcional *casefold* es un flag que especifica si un alfabeto en minúscula es aceptable como entrada. Por motivos de seguridad, el valor predeterminado es `Falso`.

RFC 4648 [https://datatracker.ietf.org/doc/html/rfc4648.html] permite la asignación opcional del dígito 0 (cero) a la letra O (oh), y la asignación opcional del dígito 1 (uno) a la letra I (ojo) o la letra L (el) . El argumento opcional *map01* cuando no es `None`, especifica la letra a la cual el dígito 1 debería mapearse(cuando *map01* no es `None`, el dígito 0 siempre se asigna a la letra O). Por motivos de seguridad, el valor predeterminado es `None`, por lo que 0 y 1 no están permitidos en la entrada.

Una **`binascii.Error`** se lanza si *s* está incorrectamente rellenado (*padded*) o si hay caracteres no alfabéticos presentes en la entrada.

base64.b32hexencode(*s*)

Similar a **`b32encode()`** pero usa el Alfabeto Hexagonal Extendido, como se define en **RFC 4648** [https://datatracker.ietf.org/doc/html/rfc4648.html].

Added in version 3.10.

base64.b32hexdecode(*s*, *casefold=False*)

Similar a **`b32decode()`** pero usa el Alfabeto Hexagonal Extendido, como se define en **RFC 4648** [https://datatracker.ietf.org/doc/html/rfc4648.html].

Esta versión no permite el dígito 0 (cero) a la letra O (oh) y el dígito 1 (uno) a las asignaciones de la letra I (ojo) o la letra L (el), todos estos caracteres están incluidos en el Alfabeto Hexagonal Extendido y no son intercambiables.

Added in version 3.10.

base64.b16encode(*s*)

Codifica el objeto similar a bytes *s* utilizando Base16 y retorna los **`bytes`** codificados.

base64.b16decode(*s*, *casefold=False*)

Decodifica el objeto similar a bytes codificado en Base16 o cadena de caracteres ASCII *s* y retorna los **`bytes`** decodificados.

El opcional *casefold* es un flag que especifica si un alfabeto en minúscula es aceptable como entrada. Por motivos de seguridad, el valor predeterminado es `Falso`.

Una **`binascii.Error`** se lanza si *s* está incorrectamente rellenado (*padded*) o si hay caracteres no alfabéticos presentes en la entrada.

Base85 Encodings

Base85 encoding is not formally specified but rather a de facto standard, thus different systems perform the encoding differently.

The **`a85encode()`** and **`b85encode()`** functions in this module are two implementations of the de facto standard. You should call the function with the Base85 implementation used by the software you intend to work with.

The two functions present in this module differ in how they handle the following:

- Whether to include enclosing `<~` and `~>` markers
- Whether to include newline characters
- The set of ASCII characters used for encoding
- Handling of null bytes

Refer to the documentation of the individual functions for more information.

base64.a85encode(*b*, ***, *foldspaces=False*, *wrapcol=0*, *pad=False*, *adobe=False*)

Codifica el objeto similar a bytes *b* utilizando Ascii85 y retorna los **`bytes`** codificados.

foldspaces es un flag opcional que utiliza la secuencia corta especial "y" en lugar de 4 espacios consecutivos (ASCII 0x20) como lo admite "btoa". Esta característica no es compatible con la codificación Ascii85 «estándar».

wrapcol controls whether the output should have newline (`b'\n'`) characters added to it. If this is non-zero, each output line will be at most this many characters long, excluding the trailing newline.

pad controla si la entrada se rellena (*padded*) a un múltiplo de 4 antes de la codificación. Tenga en cuenta que la implementación de `btoa` siempre es rellenada (*pads*).

adobe controla si la secuencia de bytes codificada está enmarcada con `<~` y `~>`, que es utilizada por la implementación de Adobe.

Added in version 3.4.

base64.a85decode(*b*, ***, *foldspaces=False*, *adobe=False*, *ignorechars=b' \t\n\r\x0b'*)

Decodifica el objeto similar a bytes codificado en Ascii85 o cadena de caracteres ASCII *b* y retorna los **`bytes`** decodificados.

foldspaces es un flag que especifica si la secuencia corta "y" debe aceptarse como abreviatura durante 4 espacios consecutivos (ASCII 0x20). Esta característica no es compatible con la codificación Ascii85 «estándar».

adobe controla si la secuencia de entrada está en formato Adobe Ascii85 (es decir, se enmarca con <~ y ~>).

ignorechars should be a byte string containing characters to ignore from the input. This should only contain whitespace characters, and by default contains all whitespace characters in ASCII.

Added in version 3.4.

base64.b85encode(*b*, *pad=False*)

Codifica el objeto similar a bytes *b* utilizando base85 (como se usa en por ejemplo, diferencias binarias de estilo git) y retorna los **`bytes`** codificados.

Si *pad* es verdadero, la entrada se rellena con `b'\0'`, por lo que su longitud es un múltiplo de 4 bytes antes de la codificación.

Added in version 3.4.

base64.b85decode(*b*)

Decodifica el objeto similar a bytes codificado en base85 o cadena de caracteres ASCII *b* y retorna los **`bytes`** decodificados. El relleno se elimina implícitamente, si es necesario.

Added in version 3.4.

base64.z85encode(*s*)

Encode the bytes-like object *s* using Z85 (as used in ZeroMQ) and return the encoded **`bytes`**. See Z85 specification [https://rfc.zeromq.org/spec/32/] for more information.

Added in version 3.13.

base64.z85decode(*s*)

Decode the Z85-encoded bytes-like object or ASCII string *s* and return the decoded **`bytes`**. See Z85 specification [https://rfc.zeromq.org/spec/32/] for more information.

Added in version 3.13.

Legacy Interface

base64.decode(*input*, *output*)

Decodifica el contenido del archivo binario *input* y escribe los datos binarios resultantes en el archivo *output*. *input* y *output* deben ser objetos archivo. *input* se leerá hasta que `input.readline()` retorne un objeto de bytes vacío.

base64.decodebytes(*s*)

Decodifica el objeto similar a bytes *s*, que debe contener una o más líneas de datos codificados en base64, y retornará los **`bytes`** decodificados.

Added in version 3.1.

base64.encode(*input*, *output*)

Codifica el contenido del archivo binario *input* y escribe los datos codificados en base64 resultantes en el archivo *output*. *input* y *output* deben ser objetos archivos. *input* se leerá hasta que `input.read()` retorna un objeto de bytes vacío. **`encode()`** inserta un carácter de nueva línea (`b'\n'`) después de cada 76 bytes de la salida, además de garantizar que la salida siempre termine con una nueva línea, según **RFC 2045** [https://datatracker.ietf.org/doc/html/rfc2045.html] (MIME).

base64.encodebytes(*s*)

Codifica el objeto similar a bytes *s*, que puede contener datos binarios arbitrarios, y retorna **`bytes`** que contienen los datos codificados en base64, con líneas nuevas (`b'\n'`) insertado después de cada 76 bytes de salida, y asegurando que haya una nueva línea final, según **RFC 2045** [https://datatracker.ietf.org/doc/html/rfc2045.html] (MIME).

Added in version 3.1.

Un ejemplo de uso del módulo:

```
>>> import base64
>>> encoded = base64.b64encode(b'data to be encoded')
>>> encoded
b'ZGF0YSB0byBiZSBlbmNvZGVk'
```

```
>>> data = base64.b64decode(encoded)
>>> data
b'data to be encoded'
```

Consideraciones de Seguridad

Se agregó una nueva sección de consideraciones de seguridad a **RFC 4648** [https://datatracker.ietf.org/doc/html/rfc4648.html] (sección 12); se recomienda revisar la sección de seguridad para cualquier código implementado en producción.

Ver también

Módulo `binascii`
Módulo de soporte que contiene conversiones de ASCII a binario y binario a ASCII.

RFC 1521 [https://datatracker.ietf.org/doc/html/rfc1521.html] - MIME (Extensiones multipropósito de correo de Internet) Parte uno: Mecanismos para especificar y describir el formato de los cuerpos de mensajes de Internet
La Sección 5.2, «Codificación de transferencia de contenido Base64», proporciona la definición de la codificación base64.

`binascii` — Convert between binary and ASCII

The **`binascii`** module contains a number of methods to convert between binary and various ASCII-encoded binary representations. Normally, you will not use these functions directly but use wrapper modules like **`base64`** instead. The **`binascii`** module contains low-level functions written in C for greater speed that are used by the higher-level modules.

Nota

Las funciones `a2b_*` aceptan cadenas Unicode que contienen solo caracteres ASCII. Otras funciones solo aceptan objetos tipo binarios (como **`bytes`**, **`bytearray`** y otros objetos que admiten el protocolo de búfer).

Distinto en la versión 3.3: Las cadenas unicode con sólo caracteres ASCII son ahora aceptadas por las funciones `a2b_*`.

The **`binascii`** module defines the following functions:

binascii.a2b_uu(*string*)

Convierte una sola línea de datos uuencoded de nuevo a binarios y retorna los datos binarios. Las líneas normalmente contienen 45 bytes (binarios), excepto por la última línea. Los datos de línea pueden ir seguidos de espacios en blanco.

binascii.b2a_uu(*data*, ***, *backtick=False*)

Convierte datos binarios a una línea de caracteres ASCII, el valor retornado es la línea convertida, incluido un carácter de nueva línea. La longitud de *data* debe ser como máximo 45. Si *backtick* es verdadero, los ceros se representan mediante `` '`' `` en lugar de espacios.

Distinto en la versión 3.7: Se ha añadido el parámetro *backtick*.

binascii.a2b_base64(*string*, */*, ***, *strict_mode=False*)

Convierte un bloque de datos en base64 de nuevo a binario y retorna los datos binarios. Se puede pasar más de una línea a la vez.

Si *strict_mode* es verdadero, solo se convertirán los datos en base64 válidos. Los datos en base64 no válidos lanzarán `binascii.Error`.

base64 válido:

- De acuerdo a **RFC 3548** [https://datatracker.ietf.org/doc/html/rfc3548.html].
- Contiene solo caracteres del alfabeto base64.
- No contiene exceso de datos después del relleno (incluye el exceso de relleno, nuevas líneas, etc.).
- No empieza con un relleno.

Distinto en la versión 3.11: Se ha añadido el parámetro *strick_mode*.

binascii.b2a_base64(*data*, ***, *newline=True*)

Convierte datos binarios en una línea de caracteres ASCII en codificación base64. El valor retornado es la línea convertida, incluido un carácter de nueva línea si *newline* es verdadero. La salida de esta función se ajusta a **RFC 3548** [https://datatracker.ietf.org/doc/html/rfc3548.html].

Distinto en la versión 3.6: Se ha añadido el parámetro *newline*.

binascii.a2b_qp(*data*, *header=False*)

Convierte un bloque de datos imprimibles entre comillas a binario y retorna los datos binarios. Se puede pasar más de una línea a la vez. Si el argumento opcional *header* está presente y es verdadero, los guiones bajos se decodificarán como espacios.

binascii.b2a_qp(*data*, *quotetabs=False*, *istext=True*, *header=False*)

Convierte datos binarios en una(s) línea(s) de caracteres ASCII en codificación imprimible entre comillas. El valor de retorno son las líneas convertidas. Si el argumento opcional *quotetabs* está presente y es verdadero, se codificarán todas los tabs y espacios. Si el argumento opcional *istext* está presente y es verdadero, las nuevas líneas no se codifican, pero se codificarán los espacios en blanco finales. Si el argumento opcional *header* está presente y es verdadero, los espacios se codificarán como guiones bajos por: rfc: *1522*. Si el argumento opcional *header* está presente y es falso, los caracteres de nueva línea también se codificarán; de lo contrario, la conversión de salto de línea podría dañar el flujo de datos binarios.

binascii.crc_hqx(*data*, *value*)

Calcula un valor CRC de 16 bits de *data*, comenzando con *value* como el CRC inicial, y retorna el resultado. Utiliza el polinomio CRC-CCITT $x^{16} + x^{12} + x^5 + 1$, a menudo representado como 0x1021. Este CRC se utiliza en el formato binhex4.

binascii.crc32(*data*[, *value*])

Calcula CRC-32, la suma de comprobación de 32 bits de *data* que no tiene signo, comenzando con un CRC inicial de *value*. El CRC inicial predeterminado es cero. El algoritmo es consistente con la suma de verificación del archivo ZIP. Dado que el algoritmo está diseñado para usarse como un algoritmo de suma de verificación, no es adecuado para usarlo como algoritmo hash general. Úselo de la siguiente manera:

```
print(binascii.crc32(b"hello world"))
# Or, in two pieces:
crc = binascii.crc32(b"hello")
crc = binascii.crc32(b" world", crc)
print('crc32 = {:#010x}'.format(crc))
```

Distinto en la versión 3.0: El resultado siempre es sin signo.

binascii.b2a_hex(*data*[, *sep*[, *bytes_per_sep=1*]])
binascii.hexlify(*data*[, *sep*[, *bytes_per_sep=1*]])

Retorna la representación hexadecimal del binario *data*. Cada byte de *data* se convierte en la representación hexadecimal de 2 dígitos correspondiente. Por lo tanto, el objeto de bytes retornado es el doble de largo que la longitud de *data*.

Una funcionalidad similar (pero que retorna una cadena de texto) también es convenientemente accesible usando el método **`bytes.hex()`**.

Si se especifica *sep*, debe ser un solo carácter *str* o un objeto de bytes. Se insertará en la salida después de cada *bytes_per_sep* bytes de entrada . La ubicación del separador se cuenta desde el extremo derecho de la salida de forma predeterminada; si desea contar desde el izquierdo, proporcione un valor negativo *bytes_per_sep*.

```
>>> import binascii
>>> binascii.b2a_hex(b'\xb9\x01\xef')
b'b901ef'
>>> binascii.hexlify(b'\xb9\x01\xef', '-')
b'b9-01-ef'
>>> binascii.b2a_hex(b'\xb9\x01\xef', b'_', 2)
b'b9_01ef'
>>> binascii.b2a_hex(b'\xb9\x01\xef', b' ', -2)
b'b901 ef'
```

Distinto en la versión 3.8: Se agregaron los parámetros *sep* y *bytes_per_sep*.

binascii.a2b_hex(*hexstr*)
binascii.unhexlify(*hexstr*)

Retorna los datos binarios representados por la cadena hexadecimal *hexstr*. Esta función es la inversa de **`b2a_hex()`**. *hexstr* debe contener un número par de dígitos hexadecimales (que pueden ser mayúsculas o minúsculas), de lo contrario se produce una excepción `Error`.

Funcionalidad similar (aceptar sólo argumentos de cadena de texto, pero más liberal hacia espacios en blanco) también es accesible mediante el método de clase `bytes.fromhex()`.

exception binascii.Error

Excepción provocada por errores. Estos suelen ser errores de programación.

exception binascii.Incomplete

Excepción provocada por datos incompletos. Por lo general, estos no son errores de programación, pero se pueden controlar leyendo un poco más de datos e intentándolo de nuevo.

Ver también

Módulo `base64`

Soporte para compatibilidad con RFC de codificación de estilo base64 en base 16, 32, 64 y 85.

Módulo `quopri`

Soporte para codificación imprimible entre comillas utilizada en mensajes de correo electrónico MIME.

`quopri` — Encode and decode MIME quoted-printable data

Código fuente: Lib/quopri.py [https://github.com/python/cpython/tree/3.14/Lib/quopri.py]

Este módulo realiza la codificación y descodificación de transporte imprimible entre comillas, tal como se define en **RFC 1521** [https://datatracker.ietf.org/doc/html/rfc1521.html]: «MIME (Multipurpose Internet Mail Extensions) Parte Uno: Mecanismos para especificar y describir el formato de los cuerpos de mensajes de Internet». La codificación imprimible entre comillas está diseñada para datos donde hay relativamente pocos caracteres no imprimibles; el esquema de codificación base64 disponible a través del módulo **`base64`** es más compacto si hay muchos caracteres de este tipo, como cuando se envía un archivo gráfico.

quopri.decode(*input*, *output*, *header=False*)

Descodificar el contenido del archivo *input* y escribir los datos binarios descodificados resultantes en el archivo *output*. *input* y *output* deben ser objetos de archivo binario. Si el argumento opcional *header* está presente y true, el carácter de subrayado se descodificará como espacio. Esto se utiliza para decodificar encabezados codificados en «Q» como se describe en **RFC 1522** [https://datatracker.ietf.org/doc/html/rfc1522.html]: «MIME (Multipurpose Internet Mail Extensions) Parte dos: Extensiones de encabezado de mensaje para texto no ASCII».

quopri.encode(*input*, *output*, *quotetabs*, *header=False*)

Codifique el contenido del archivo *input* y escriba los datos imprimibles entre comillas resultantes en el archivo *output*. *input* y *output* deben ser objetos de archivo binario. *quotetabs*, un indicador no opcional que controla si codificar espacios incrustados y pestañas; cuando true codifica dicho espacio en blanco incrustado, y cuando false los deja sin codificar. Tenga en cuenta que los espacios y pestañas que aparecen al final de las líneas siempre están codificados, según **RFC 1521** [https://datatracker.ietf.org/doc/html/rfc1521.html]. *header* es un indicador que controla si los espacios están codificados como guiones bajos según **RFC 1522** [https://datatracker.ietf.org/doc/html/rfc1522.html].

quopri.decodestring(*s*, *header=False*)

Como `decode()`, excepto que acepta una fuente `bytes` y retorna el correspondiente `bytes` decodificado.

quopri.encodestring(*s*, *quotetabs=False*, *header=False*)

Como `encode()`, excepto que acepta un origen `bytes` y retorna el codificado correspondiente `bytes`. De forma predeterminada, envía un valor `False` al parámetro *quotetabs* de la función `encode()`.

Ver también

Módulo `base64`

Codificar y decodificar datos MIME base64

Herramientas Para Procesar Formatos de Marcado Estructurado

Python soporta una variedad de módulos para trabajar con varias formas de almacenar datos de forma estructurada. Esto incluye módulos para trabajar con el Lenguaje de Marcado Estructurado General (SGML) y el Lenguaje de de Marcado de Hipertexto (HTML), y varias interfaces para trabajar con el Lenguaje de Marcado Estructurado Extensible (XML).

- **`html`** — HyperText Markup Language support
- **`html.parser`** — Simple HTML and XHTML parser
    - Aplicación ejemplo de un analizador sintáctico (*parser*) de HTML
    - Métodos **`HTMLParser`**
    - Ejemplos
- **`html.entities`** — Definitions of HTML general entities
- Módulos de procesamiento XML
    - XML security
- **`xml.etree.ElementTree`** — The ElementTree XML API
    - Tutorial
        - Árbol y elementos XML
        - Procesando XML
        - API de consulta para un procesamiento no bloqueante
        - Encontrando elementos interesantes
        - Modificando un archivo XML
        - Construyendo documentos XML

        - Procesando XML con espacio de nombres
    - Soporte de XPath
        - Ejemplo
        - Sintaxis XPath soportada
    - Referencia
        - Funciones
    - Soporte de XInclude
        - Ejemplo
    - Referencia
        - Funciones
        - Objetos Element
        - Objetos ElementTree
        - Objetos QName
        - Objetos TreeBuilder
        - Objetos XMLParser
        - Objetos XMLPullParser
        - Excepciones
- **`xml.dom`** — The Document Object Model API
    - Contenido del módulo
    - Objetos en el DOM
        - Objetos *DOMImplementation*
        - Objetos nodo
        - Objetos *NodeList*
        - Objetos *DocumentType*
        - Objetos documento
        - Objetos elemento
        - Objetos atributo
        - Objetos *NamedNodeMap*
        - Objetos comentario
        - Objetos Texto y *CDATASection*
        - Objetos *ProcessingInstruction*

        - Excepciones
    - Conformidad
        - Mapeo de tipos
        - Métodos de acceso (*accessor*)
- **`xml.dom.minidom`** — Minimal DOM implementation
    - Objetos del DOM
    - Ejemplo de DOM
    - minidom y el estándar DOM
- **`xml.dom.pulldom`** — Support for building partial DOM trees
    - Objetos DOMEventStream
- **`xml.sax`** — Support for SAX2 parsers
    - Objetos SAXException
- **`xml.sax.handler`** — Base classes for SAX handlers
    - Objetos ContentHandler
    - Objetos DTDHandler
    - Objetos EntityResolver
    - Objetos ErrorHandler
    - Objetos DTDHandler
- **`xml.sax.saxutils`** — SAX Utilities
- **`xml.sax.xmlreader`** — Interface for XML parsers
    - Objetos XMLReader
    - Objetos IncrementalParser
    - Objetos localizadores
    - Objetos InputSource
    - La Interfaz **`Attributes`**
    - La Interfaz **`AttributesNS`**
- **`xml.parsers.expat`** — Fast XML parsing using Expat

  - Objetos XMLParser
  - Excepciones de ExpatError
  - Ejemplo
  - Descripciones del modelo de contenido
  - Constantes de error de expansión

`html` — HyperText Markup Language support

Código fuente: Lib/html/__init__.py [https://github.com/python/cpython/tree/3.14/Lib/html/__init__.py]

Este módulo define utilidades para manipular HTML.

html.escape(*s*, *quote=True*)

Convert the characters `&`, `<` and `>` in string *s* to HTML-safe sequences. Use this if you need to display text that might contain such characters in HTML. If the optional flag *quote* is true (the default), the characters (`"`) and (`'`) are also translated; this helps for inclusion in an HTML attribute value delimited by quotes, as in `<a href="...">`. If *quote* is set to false, the characters (`"`) and (`'`) are not translated.

Added in version 3.2.

html.unescape(*s*)

Convierte todas las referencias de caracteres numéricos y con nombre (por ejemplo `&gt;`, `&#62;`, `&#x3e;`) de la cadena de caracteres *s* a los caracteres Unicode correspondientes. Esta función utiliza las reglas definidas por el estándar HTML 5 para las referencias de caracteres válidas e inválidas, y la **`lista de referencia de caracteres con nombre de HTML 5`**.

Added in version 3.4.

Los submódulos del paquete `html` son:

- **`html.parser`** – Analizador sintáctico simple de HTML y

XHTML

- **`html.entities`** – Definición general de entidades HTML

`html.parser` — Simple HTML and XHTML parser

Código fuente: Lib/html/parser.py [https://github.com/python/cpython/tree/3.14/Lib/html/parser.py]

Este módulo define una clase **`HTMLParser`** que sirve como base para analizar archivos de texto formateados en HTML (*HyperText Mark-up Language*) y XHTML.

class html.parser.HTMLParser(**, convert_charrefs=True, scripting=False*)

Cree una instancia de analizador capaz de analizar marcado no válido.

If *convert_charrefs* is true (the default), all character references (except the ones in elements like `script` and `style`) are automatically converted to the corresponding Unicode characters.

If *scripting* is false (the default), the content of the `noscript` element is parsed normally; if it's true, it's returned as is without being parsed.

Una instancia de **`HTMLParser`** se alimenta de datos HTML y llama a métodos de manejo cuando se encuentran etiquetas de inicio, etiquetas finales, texto, comentarios y otros elementos de marcado. El usuario debe subclasificar **`HTMLParser`** y anular sus métodos para implementar el comportamiento deseado.

Este analizador no verifica que las etiquetas finales coincidan con las etiquetas iniciales ni llame al manejador de etiquetas finales para los elementos que se cierran implícitamente al cerrar un elemento externo.

Distinto en la versión 3.4: argumento de palabra clave *convert_charrefs* agregado.

Distinto en la versión 3.5: El valor predeterminado para el argumento *convert_charrefs* ahora es `True`.

Distinto en la versión 3.14.1: Added the *scripting* parameter.

Aplicación ejemplo de un analizador sintáctico (*parser*) de HTML

As a basic example, below is a simple HTML parser that uses the **`HTMLParser`** class to print out start tags, end tags, and data as they are encountered:

```
from html.parser import HTMLParser

class MyHTMLParser(HTMLParser):
    def handle_starttag(self, tag, attrs):
        print("Encountered a start tag:", tag)

    def handle_endtag(self, tag):
        print("Encountered an end tag :", tag)

    def handle_data(self, data):
        print("Encountered some data  :", data)

parser = MyHTMLParser()
parser.feed('<html><head><title>Test</title></head>'
            '<body><h1>Parse me!</h1></body></html>')
```

La salida será entonces:

```
Encountered a start tag: html
Encountered a start tag: head
Encountered a start tag: title
Encountered some data  : Test
Encountered an end tag : title
Encountered an end tag : head
Encountered a start tag: body
Encountered a start tag: h1
Encountered some data  : Parse me!
Encountered an end tag : h1
Encountered an end tag : body
Encountered an end tag : html
```

Métodos `HTMLParser`

instancias de `HTMLParser` tienen los siguientes métodos:

HTMLParser.feed(*data*)

> Alimente un poco de texto al analizador. Se procesa en la medida en que consta de elementos completos; los datos incompletos se almacenan en el búfer hasta que se introducen más datos o se llama a `close()`. *data* debe ser `str`.

HTMLParser.close()

> Fuerce el procesamiento de todos los datos almacenados como si fueran seguidos por una marca de fin de archivo. Este método puede ser redefinido por una clase derivada para definir un procesamiento adicional al final de la entrada, pero la versión redefinida siempre debe llamar a `HTMLParser` método de clase base `close()`.

HTMLParser.reset()

> Restablecer la instancia. Pierde todos los datos no procesados. Esto se llama implícitamente en el momento de la instanciación.

HTMLParser.getpos()

> Retorna el número de línea actual y el desplazamiento.

HTMLParser.get_starttag_text()

> Retorna el texto de la etiqueta de inicio abierta más

recientemente. Normalmente, esto no debería ser necesario para el procesamiento estructurado, pero puede ser útil para tratar con HTML «como implementado» o para volver a generar entradas con cambios mínimos (se puede preservar el espacio en blanco entre los atributos, etc.).

Los siguientes métodos se invocan cuando se encuentran datos o elementos de marcado y deben anularse en una subclase. Las implementaciones de la clase base no hacen nada (excepto **`handle_startendtag()`**):

HTMLParser.handle_starttag(*tag*, *attrs*)

This method is called to handle the start tag of an element (e.g. `<div id="main">`).

The *tag* argument is the name of the tag converted to lower case. The *attrs* argument is a list of `(name, value)` pairs containing the attributes found inside the tag's `<>` brackets. The *name* will be translated to lower case, and quotes in the *value* have been removed, and character and entity references have been replaced. For empty attributes, *value* is `None`.

Por ejemplo, para la etiqueta `<A HREF="https://www.cwi.nl/">`, este método se llamaría como `handle_starttag('a', [('href', 'https : // www.cwi.nl/ ')])`.

Todas las referencias de entidad de **`html.entities`** se reemplazan en los valores de los atributos.

HTMLParser.handle_endtag(*tag*)

Este método se llama para manejar la etiqueta final de un elemento (por ejemplo, `</div>`)

El argumento *tag* es el nombre de la etiqueta convertida a minúsculas.

HTMLParser.handle_startendtag(*tag*, *attrs*)

Similar a **`handle_starttag()`**, pero llamado cuando el analizador encuentra una etiqueta vacía de estilo XHTML (`<img .../>`). Este método puede ser anulado por subclases que requieren esta información léxica particular; la implementación predeterminada simplemente llama **`handle_starttag()`** y **`handle_endtag()`**.

HTMLParser.handle_data(*data*)

This method is called to process arbitrary data (e.g. text nodes and the content of elements like `script` and `style`).

HTMLParser.handle_entityref(*name*)

This method is called to process a named character reference of the form `&name;` (e.g. `&gt;`), where *name* is a general entity reference (e.g. `'gt'`). This method is only called if *convert_charrefs* is false.

HTMLParser.handle_charref(*name*)

This method is called to process decimal and hexadecimal numeric character references of the form `&#NNN;` and `&#xNNN;`. For example, the decimal equivalent for `&gt;` is `&#62;`, whereas the hexadecimal is `&#x3E;`; in this case the method will receive `'62'` or `'x3E'`. This method is only called if *convert_charrefs* is false.

HTMLParser.handle_comment(*data*)

Este método se llama cuando se encuentra un comentario (por ejemplo, `<!--comment-->`).

Por ejemplo, el comentario `<! - comment ->` hará que se llame a este método con el argumento `'comment'`.

El contenido de los comentarios condicionales de Internet Explorer (*condcoms*) también se enviará a este método, por lo tanto, para `<!--[if IE 9]>IE9-specific content<![endif]-->`, este método recibirá `'[if IE 9]>IE9-specific content<![endif]'`.

HTMLParser.handle_decl(*decl*)

Este método se llama para manejar una declaración de tipo de documento HTML (por ejemplo, `<!DOCTYPE html>`).

El parámetro *decl* será todo el contenido de la declaración dentro del `<!...>` *markup* (por ejemplo, `'DOCTYPE html'`).

HTMLParser.handle_pi(*data*)

Método llamado cuando se encuentra una instrucción de procesamiento. El parámetro *data* contendrá toda la instrucción de procesamiento. Por ejemplo, para la instrucción de procesamiento `<?proc color='red'>`, este método se llamaría como `handle_pi("proc color='red'")`. Está destinado a ser anulado por una clase derivada; La implementación de la clase base no hace nada.

Nota

La clase **`HTMLParser`** utiliza las reglas sintácticas SGML para procesar instrucciones. Una instrucción de procesamiento XHTML que use el `'?'` final hará que se incluya el `'?'` en *data*.

HTMLParser.unknown_decl(*data*)

Se llama a este método cuando el analizador lee una declaración no reconocida.

El parámetro *data* será el contenido completo de la declaración dentro del marcado `<![...]>`. A veces es útil ser reemplazado por una clase derivada. La implementación de la clase base no hace nada.

Ejemplos

The following class implements a parser that will be used to illustrate more examples:

```
from html.parser import HTMLParser
from html.entities import name2codepoint

class MyHTMLParser(HTMLParser):
    def handle_starttag(self, tag, attrs):
        print("Start tag:", tag)
        for attr in attrs:
            print("     attr:", attr)

    def handle_endtag(self, tag):
        print("End tag  :", tag)

    def handle_data(self, data):
        print("Data     :", data)

    def handle_comment(self, data):
        print("Comment  :", data)

    def handle_entityref(self, name):
        c = chr(name2codepoint[name])
        print("Named ent:", c)

    def handle_charref(self, name):
        if name.startswith('x'):
            c = chr(int(name[1:], 16))
        else:
            c = chr(int(name))
        print("Num ent  :", c)

    def handle_decl(self, data):
        print("Decl     :", data)

parser = MyHTMLParser()
```

Parsing a doctype:

```
>>> parser.feed('<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
...             '"http://www.w3.org/TR/html4/strict.dtd"
Decl     : DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//E
```

Parsing an element with a few attributes and a title:

```
>>> parser.feed('<img src="python-logo.png" alt="The Pyt
Start tag: img
     attr: ('src', 'python-logo.png')
     attr: ('alt', 'The Python logo')
>>>
>>> parser.feed('<h1>Python</h1>')
Start tag: h1
Data     : Python
End tag  : h1
```

The content of elements like `script` and `style` is returned as is, without further parsing:

```
>>> parser.feed('<style type="text/css">#python { color:
Start tag: style
     attr: ('type', 'text/css')
Data     : #python { color: green }
End tag  : style

>>> parser.feed('<script type="text/javascript">'
...             'alert("<strong>hello! &#9786;</strong>"
Start tag: script
     attr: ('type', 'text/javascript')
Data     : alert("<strong>hello! &#9786;</strong>");
End tag  : script
```

Attribute names are converted to lowercase, quotes from attribute values removed, and `None` is returned as *value* for empty attributes (such as `checked`):

```
>>> parser.feed("<input TYPE='checkbox' checked requirec
Start tag: input
     attr: ('type', 'checkbox')
     attr: ('checked', None)
     attr: ('required', '')
     attr: ('disabled', 'disabled')
```

Parsing comments:

```
>>> parser.feed('<!--a comment-->'
...             '<!--[if IE 9]>IE-specific content<![enc
Comment   : a comment
Comment   : [if IE 9]>IE-specific content<![endif]
```

Parsing named and numeric character references and converting them to the correct char (note: these 3 references are all equivalent to `'>'`):

```
>>> parser = MyHTMLParser()
>>> parser.feed('&gt;&#62;&#x3E;')
Data     : >>>

>>> parser = MyHTMLParser(convert_charrefs=False)
>>> parser.feed('&gt;&#62;&#x3E;')
Named ent: >
Num ent  : >
Num ent  : >
```

Feeding incomplete chunks to **`feed()`** works, but **`handle_data()`** might be called more than once if *convert_charrefs* is false:

```
>>> for chunk in ['<sp', 'an>buff', 'ered', ' text</s',
...     parser.feed(chunk)
...
Start tag: span
Data     : buff
Data     : ered
Data     :  text
End tag  : span
```

Parsing invalid HTML (e.g. unquoted attributes) also works:

```
>>> parser.feed('<p><a class=link href=#main>tag soup</p
Start tag: p
Start tag: a
     attr: ('class', 'link')
     attr: ('href', '#main')
Data     : tag soup
```

```
End tag  : p
End tag  : a
```

[1]

`html.entities` — Definitions of HTML general entities

Código fuente: Lib/html/entities.py [https://github.com/python/cpython/tree/3.14/Lib/html/entities.py]

Este módulo define cuatro diccionarios **`html5`**, **`name2codepoint`**, **`codepoint2name`**, y **`entitydefs`**.

html.entities.html5

Un diccionario que asigna referencias de caracteres con nombre HTML5 [1] a los caracteres Unicode equivalentes, p. Ej. `html5['gt;'] == '>'`. Tenga en cuenta que el punto y coma al final está incluido en el nombre (por ejemplo, `'gt;'`), sin embargo, algunos de los nombres son aceptados por el estándar incluso sin el punto y coma: en este caso, el nombre está presente con y sin el `';'`. Consulte también **`html.unescape()`**.

Added in version 3.3.

html.entities.entitydefs

Un diccionario que asigna definiciones de entidad XHTML 1.0 a su texto de reemplazo en ISO Latin-1.

html.entities.name2codepoint

Un diccionario que asigna nombres de entidades HTML4 a los puntos de código Unicode.

html.entities.codepoint2name

Un diccionario que asigna puntos de código Unicode a nombres de entidades HTML4.

Notas al pie

Vea https://html.spec.whatwg.org/multipage/named-characters.html#named-character-references

Módulos de procesamiento XML

Código fuente: Lib/xml/ [https://github.com/python/cpython/tree/3.14/Lib/xml/]

Las interfaces de Python para procesar XML están agrupadas en el paquete `xml`.

Nota

If you need to parse untrusted or unauthenticated data, see XML security.

It is important to note that modules in the **`xml`** package require that there be at least one SAX-compliant XML parser available. The Expat parser is included with Python, so the **`xml.parsers.expat`** module will always be available.

La documentación de los paquetes **`xml.dom`** y **`xml.sax`** es la definición de los enlaces de Python para las interfaces DOM y SAX.

Los submódulos de manejo de XML son:

- **`xml.etree.ElementTree`**: la API ElementTree, un procesador de XML simple y ligero

- **`xml.dom`**: la definición de la API DOM
- **`xml.dom.minidom`**: una implementación mínima de DOM
- **`xml.dom.pulldom`**: soporte para la construcción de árboles DOM parciales

- **`xml.sax`**: clases base SAX2 y funciones de conveniencia
- **`xml.parsers.expat`**: el enlace del analizador Expat

XML security

An attacker can abuse XML features to carry out denial of service attacks, access local files, generate network connections to other machines, or circumvent firewalls when attacker-controlled XML is being parsed, in Python or elsewhere.

The built-in XML parsers of Python rely on the library libexpat [https://github.com/libexpat/libexpat], commonly called Expat, for parsing XML.

By default, Expat itself does not access local files or create network connections.

Expat versions lower than 2.7.2 may be vulnerable to the «billion laughs», «quadratic blowup» and «large tokens» vulnerabilities, or to disproportional use of dynamic memory. Python bundles a copy of Expat, and whether Python uses the bundled or a system-wide Expat, depends on how the Python interpreter `has been configured` in your environment. Python may be vulnerable if it uses such older versions of Expat. Check **`pyexpat.EXPAT_VERSION`**.

`xmlrpc` is **vulnerable** to the «decompression bomb» attack.

mil millones de risas / expansión exponencial de entidad

El ataque Billion Laughs [https://en.wikipedia.org/wiki/Billion_laughs], también conocido como expansión exponencial de entidades, utiliza varios niveles de entidades anidadas. Cada entidad hace referencia a otra entidad varias veces y la definición de entidad final contiene una cadena pequeña. La expansión exponencial da como resultado varios gigabytes de texto y consume mucha memoria y tiempo de CPU.

expansión de entidad de explosión cuadrática

Un ataque de explosión cuadrática es similar a un ataque de Billion Laughs [https://en.wikipedia.org/wiki/Billion_laughs]; también abusa de la expansión de entidad. En lugar de entidades anidadas, repite una entidad grande con un par de miles de caracteres una y otra vez. El ataque no es tan eficaz como el caso exponencial, pero evita desencadenar contramedidas del analizador que prohíben entidades profundamente anidadas.

bomba de descompresión

Las bombas de descompresión (también conocidas como ZIP bomb [https://en.wikipedia.org/wiki/Zip_bomb]) se aplican a todas las bibliotecas XML que pueden analizar secuencias XML comprimidas, como secuencias HTTP comprimidas con gzip o archivos comprimidos por LZMA. Para un atacante puede reducir la cantidad de datos transmitidos en magnitudes de tres o más.

large tokens

Expat needs to re-parse unfinished tokens; without the protection introduced in Expat 2.6.0, this can lead to quadratic runtime that can be used to cause denial of service in the application parsing XML. The issue is known as **CVE 2023-52425** [https://www.cve.org/CVERecord?id=CVE-2023-52425].

[1] (1,2,3,4)

`xml.etree.ElementTree` — The ElementTree XML API

Código fuente: Lib/xml/etree/ElementTree.py [https://github.com/python/cpython/tree/3.14/Lib/xml/etree/ElementTree.py]

The `xml.etree.ElementTree` module implements a simple and efficient API for parsing and creating XML data.

Distinto en la versión 3.3: Este módulo utilizará una implementación rápida siempre que esté disponible.

Obsoleto desde la versión 3.3: The `xml.etree.cElementTree` module is deprecated.

Nota

If you need to parse untrusted or unauthenticated data, see XML security.

Tutorial

This is a short tutorial for using `xml.etree.ElementTree` (`ET` in short). The goal is to demonstrate some of the building blocks and basic concepts of the module.

Árbol y elementos XML

XML es un formato de datos inherentemente jerárquico, y la forma más natural de representarlo es con un árbol. `ET` tiene dos clases para este propósito - `ElementTree` representa todo el documento XML como un árbol, y `Element` representa un solo nodo en este árbol. Las interacciones con todo el documento (leer y escribir en/

desde archivos) se realizan normalmente en el nivel de **`ElementTree`**. Las interacciones con un solo elemento XML y sus sub-elementos se realizan en el nivel **`Element`**.

Procesando XML

We'll be using the fictive `country_data.xml` XML document as the sample data for this section:

```
<?xml version="1.0"?>
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```

Podemos importar estos datos leyendo desde un archivo:

```
import xml.etree.ElementTree as ET
tree = ET.parse('country_data.xml')
root = tree.getroot()
```

O directamente desde una cadena de caracteres:

```
root = ET.fromstring(country_data_as_string)
```

`fromstring()` analiza el XML de una cadena de caracteres directamente en un **`Element`**, que es el elemento raíz del árbol analizado. Otras funciones de análisis pueden crear un **`ElementTree`**. Compruebe la documentación para estar seguro.

Como un **`Element`**, `root` tiene una etiqueta y un diccionario de atributos:

```
>>> root.tag
'data'
>>> root.attrib
{}
```

También tiene nodos hijos sobre los cuales podemos iterar:

```
>>> for child in root:
...     print(child.tag, child.attrib)
...
country {'name': 'Liechtenstein'}
country {'name': 'Singapore'}
country {'name': 'Panama'}
```

Los hijos están anidados, y podemos acceder a nodos hijos específicos por el índice:

```
>>> root[0][1].text
'2008'
```

Nota

No todos los elementos de la entrada XML acabarán siendo elementos del árbol analizado. Actualmente, este módulo omite cualquier comentario XML, instrucciones de procesamiento y declaraciones de tipo documento en la entrada. Sin embargo, los árboles construidos utilizando la API de este módulo, en lugar de analizar el texto XML, pueden contener comentarios e instrucciones de procesamiento, que se incluirán al generar la salida XML. Se puede acceder a una declaración de tipo documento pasando una instancia **`TreeBuilder`** personalizada al constructor **`XMLParser`**.

API de consulta para un procesamiento no bloqueante

La mayoría de las funciones de análisis proporcionadas por este módulo requieren que se lea todo el documento a la vez antes de retornar cualquier resultado. Es posible utilizar un **`XMLParser`** y alimentar los datos en él de forma incremental, pero se trata de una *push* API que llama a métodos en un objetivo invocable, que es demasiado bajo nivel e inconveniente para la mayoría de las necesidades. A veces, lo que el usuario realmente quiere es ser capaz de analizar XML de forma incremental, sin bloquear las operaciones, mientras disfruta de la comodidad de los objetos **`Element`** totalmente construidos.

La herramienta más potente para hacer esto es **`XMLPullParser`**. No requiere una lectura de bloqueo para obtener los datos XML, y en su lugar se alimenta de datos de forma incremental con llamadas a **`XMLPullParser.feed()`**. Para obtener los elementos XML analizados, llama a **`XMLPullParser.read_events()`**. He aquí un ejemplo:

```
>>> parser = ET.XMLPullParser(['start', 'end'])
>>> parser.feed('<mytag>sometext')
>>> list(parser.read_events())
[('start', <Element 'mytag' at 0x7fa66db2be58>)]
>>> parser.feed(' more text</mytag>')
>>> for event, elem in parser.read_events():
...     print(event)
...     print(elem.tag, 'text=', elem.text)
...
end
mytag text= sometext more text
```

El caso de uso obvio es el de las aplicaciones que operan de forma no bloqueante, donde los datos XML se reciben de un socket o se leen de forma incremental desde algún dispositivo de almacenamiento. En estos casos, las lecturas bloqueantes son inaceptables.

Debido a su flexibilidad, **`XMLPullParser`** puede ser un inconveniente para los casos de uso más simples. Si no te importa que tu aplicación se bloquee en la lectura de datos XML pero te gustaría tener capacidades de análisis incremental, echa un vistazo a **`iterparse()`**. Puede ser útil cuando estás leyendo un documento XML grande y no quieres mantenerlo completamente en memoria.

Where *immediate* feedback through events is wanted, calling method **`XMLPullParser.flush()`** can help reduce delay; please make sure to study the related security notes.

Encontrando elementos interesantes

`Element` tiene algunos métodos útiles que ayudan a iterar recursivamente sobre todo el sub-árbol por debajo de él (sus hijos, los hijos de sus hijos, y así sucesivamente). Por ejemplo, **`Element.iter()`**:

```
>>> for neighbor in root.iter('neighbor'):
...     print(neighbor.attrib)
...
{'name': 'Austria', 'direction': 'E'}
{'name': 'Switzerland', 'direction': 'W'}
{'name': 'Malaysia', 'direction': 'N'}
{'name': 'Costa Rica', 'direction': 'W'}
{'name': 'Colombia', 'direction': 'E'}
```

`Element.findall()` encuentra sólo los elementos con una etiqueta que son hijos directos del elemento actual. **`Element.find()`** encuentra el *primer* hijo con una etiqueta determinada, y **`Element.text`** accede al contenido de texto del elemento. **`Element.get()`** accede a los atributos del elemento:

```
>>> for country in root.findall('country'):
...     rank = country.find('rank').text
...     name = country.get('name')
...     print(name, rank)
...
Liechtenstein 1
Singapore 4
Panama 68
```

Es posible especificar de forma más sofisticada qué elementos buscar utilizando XPath.

Modificando un archivo XML

`ElementTree` proporciona una forma sencilla de construir documentos XML y escribirlos en archivos. El método **`ElementTree.write()`** sirve para este propósito.

Una vez creado, un objeto **`Element`** puede ser manipulado cambiando directamente sus campos (como **`Element.text`**), añadiendo y modificando atributos (método **`Element.set()`**), así como añadiendo nuevos hijos (por ejemplo con **`Element.append()`**).

Digamos que queremos añadir uno al rango de cada país, y añadir un atributo `updated` al elemento rango:

```
>>> for rank in root.iter('rank'):
...     new_rank = int(rank.text) + 1
...     rank.text = str(new_rank)
...     rank.set('updated', 'yes')
...
>>> tree.write('output.xml')
```

Nuestro XML tiene ahora este aspecto:

```
<?xml version="1.0"?>
<data>
    <country name="Liechtenstein">
        <rank updated="yes">2</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank updated="yes">5</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank updated="yes">69</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```

Podemos eliminar elementos utilizando **`Element.remove()`**. Digamos que queremos eliminar todos los países con un rango superior a 50:

```
>>> for country in root.findall('country'):
...     # using root.findall() to avoid removal during t
...     rank = int(country.find('rank').text)
...     if rank > 50:
...         root.remove(country)
...
>>> tree.write('output.xml')
```

Tenga en cuenta que la modificación concurrente mientras se itera puede conducir a problemas, al igual que cuando se itera y modifica listas o diccionarios de Python. Por lo tanto, el ejemplo recoge primero todos los elementos coincidentes con `root.findall()`, y sólo entonces itera sobre la lista de coincidencias.

Nuestro XML tiene ahora este aspecto:

```
<?xml version="1.0"?>
<data>
    <country name="Liechtenstein">
```

```
        <rank updated="yes">2</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank updated="yes">5</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
</data>
```

Construyendo documentos XML

La función `SubElement()` también proporciona una forma cómoda de crear nuevos sub-elementos para un elemento dado:

```
>>> a = ET.Element('a')
>>> b = ET.SubElement(a, 'b')
>>> c = ET.SubElement(a, 'c')
>>> d = ET.SubElement(c, 'd')
>>> ET.dump(a)
<a><b /><c><d /></c></a>
```

Procesando XML con espacio de nombres

Si la entrada XML tiene espacio de nombres [https://es.wikipedia.org/wiki/Espacio_de_nombres_XML], las etiquetas y los atributos con prefijos de la forma `prefix:sometag` se expanden a `{uri}sometag` donde el *prefix* se sustituye por el *URI* completo. Además, si hay un espacio de nombre por defecto [https://www.w3.org/TR/xml-names/#defaulting], ese URI completo se antepone a todas las etiquetas sin prefijo.

A continuación se muestra un ejemplo de XML que incorpora dos espacios de nombres, uno con el prefijo «fictional» y el otro que sirve como espacio de nombres por defecto:

```
<?xml version="1.0"?>
<actors xmlns:fictional="http://characters.example.com"
        xmlns="http://people.example.com">
    <actor>
        <name>John Cleese</name>
        <fictional:character>Lancelot</fictional:charact
        <fictional:character>Archie Leach</fictional:cha
    </actor>
    <actor>
        <name>Eric Idle</name>
        <fictional:character>Sir Robin</fictional:charac
        <fictional:character>Gunther</fictional:characte
        <fictional:character>Commander Clement</fictiona
    </actor>
</actors>
```

Una forma de buscar y explorar este ejemplo XML es añadir manualmente el URI a cada etiqueta o atributo en el XPath de un **`find()`** o **`findall()`**:

```
root = fromstring(xml_text)
for actor in root.findall('{http://people.example.com}ac
    name = actor.find('{http://people.example.com}name')
    print(name.text)
    for char in actor.findall('{http://characters.exampl
        print(' |-->', char.text)
```

Una mejor manera de buscar en el ejemplo de XML con espacio para nombres es crear un diccionario con sus propios prefijos y utilizarlos en las funciones de búsqueda:

```
ns = {'real_person': 'http://people.example.com',
      'role': 'http://characters.example.com'}

for actor in root.findall('real_person:actor', ns):
    name = actor.find('real_person:name', ns)
    print(name.text)
    for char in actor.findall('role:character', ns):
        print(' |-->', char.text)
```

Estos dos enfoques dan como resultado:

```
John Cleese
 |--> Lancelot
 |--> Archie Leach
Eric Idle
 |--> Sir Robin
 |--> Gunther
 |--> Commander Clement
```

Soporte de XPath

Este módulo proporciona un soporte limitado para las expresiones XPath [https://www.w3.org/TR/xpath] para localizar elementos en un árbol. El objetivo es soportar un pequeño subconjunto de la sintaxis abreviada; un motor XPath completo está fuera del alcance del módulo.

Ejemplo

A continuación se muestra un ejemplo que demuestra algunas de las capacidades de XPath del módulo. Utilizaremos el documento XML `countrydata` de la sección Parsing XML:

```
import xml.etree.ElementTree as ET

root = ET.fromstring(countrydata)

# Top-level elements
root.findall(".")

# All 'neighbor' grand-children of 'country' children of
# elements
root.findall("./country/neighbor")

# Nodes with name='Singapore' that have a 'year' child
root.findall(".//year/..[@name='Singapore']")

# 'year' nodes that are children of nodes with name='Sir
```

```
root.findall(".//*[@name='Singapore']/year")

# All 'neighbor' nodes that are the second child of thei
root.findall(".//neighbor[2]")
```

Para XML con espacios de nombre, use la notación calificada habitual `{namespace}tag`:

```
# All dublin-core "title" tags in the document
root.findall(".//{http://purl.org/dc/elements/1.1/}title
```

Sintaxis XPath soportada

Sintaxis	Significado
`tag`	Selecciona todos los elementos hijos con la etiqueta dada. Por ejemplo, `spam` selecciona todos los elementos hijos llamados `spam`, y `spam/egg` selecciona todos los nietos llamados `egg` en todos los hijos llamados `spam`. `{namespace}*` selecciona todas las etiquetas en el espacio de nombres dado, `{*}spam` selecciona las etiquetas llamadas `spam` en cualquier (o ningún) espacio de nombres, y `{}*` sólo selecciona las etiquetas que no están en un espacio de nombres. *Distinto en la versión 3.8:* Se ha añadido la posibilidad de utilizar comodines asterisco.
`*`	Selecciona todos los elementos hijos, incluidos los comentarios y las instrucciones de procesamiento. Por ejemplo, `*/egg` selecciona todos los hijos llamados `egg`.
`.`	Selecciona el nodo actual. Esto es útil sobre todo al principio de la ruta, para indicar que es una ruta relativa.
`//`	Selecciona todos los sub-elementos, en todos los niveles por debajo del elemento actual. Por ejemplo, `.//egg` selecciona todos los elementos `egg` en todo el árbol.
`..`	Selecciona el elemento padre. Retorna `None` si la ruta intenta llegar a los ancestros del elemento inicial (el elemento `find` fue invocado).
`[@attrib]`	Selecciona todos los elementos que tengan el atributo dado.
`[@attrib='value']`	Selecciona todos los elementos para los que el atributo dado tiene el valor dado. El valor no puede contener comillas.
`[@attrib!='value']`	Selecciona todos los elementos para los que el atributo dado no tiene el valor dado. El valor no puede contener comillas.

	Added in version 3.10.
`[tag]`	Selecciona todos los elementos que tienen un hijo llamado `tag`. Sólo se admiten los hijos inmediatos.
`[.='text']`	Selecciona todos los elementos cuyo contenido de texto completo, incluyendo los descendientes, es igual al «texto» dado. *Added in version 3.7.*
`[.!='text']`	Selecciona todos los elementos cuyo contenido de texto completo, incluidos los descendientes, no es igual al `text` proporcionado. *Added in version 3.10.*
`[tag='text']`	Selecciona todos los elementos que tienen un hijo llamado `tag` cuyo contenido de texto completo, incluyendo los descendientes, es igual al `text` dado.
`[tag!='text']`	Selecciona todos los elementos que tienen un hijo llamado `tag` cuyo contenido de texto completo, incluidos los descendientes, no es igual al `text` dado. *Added in version 3.10.*
`[position]`	Selecciona todos los elementos que se encuentran en la posición dada. La posición puede ser un número entero (1 es la primera posición), la expresión `last()` (para la última posición), o una posición relativa a la última posición (por ejemplo, `last()-1`).

Los predicados (expresiones entre corchetes) deben ir precedidos de un nombre de etiqueta, un asterisco u otro predicado. Los predicados `position` deben ir precedidos de un nombre de etiqueta.

Referencia

Funciones

xml.etree.ElementTree.canonicalize(*xml_data=None*, ***, *out=None*, *from_file=None*, ***options*)

Función de transformación C14N 2.0 [https://www.w3.org/TR/xml-c14n2/].

Canonicalization is a way to normalise XML output in a way that allows byte-by-byte comparisons and digital signatures. It reduces the freedom that XML serializers have and instead generates a more constrained XML representation. The main restrictions regard the placement of namespace declarations, the ordering of attributes, and ignorable whitespace.

Esta función toma una cadena de datos XML (*xml_data*) o una ruta de archivo o un objeto tipo archivo (*from_file*) como entrada, la convierte a la forma canónica y la escribe utilizando el objeto archivo (o tipo archivo) *out*, si se proporciona, o la devuelve como una cadena de texto si no. El archivo de salida recibe texto, no bytes. Por tanto, debe abrirse en modo texto con codificación `utf-8`.

Usos típicos:

```
xml_data = "<root>...</root>"
print(canonicalize(xml_data))

with open("c14n_output.xml", mode='w', encoding='ut
    canonicalize(xml_data, out=out_file)

with open("c14n_output.xml", mode='w', encoding='ut
    canonicalize(from_file="inputfile.xml", out=out
```

Las opciones de configuración *options* son las siguientes:

- *with_comments*: configurar a `True` para incluir los comentarios (por defecto: `False`)
- *strip_text*: configurar a `True` para eliminar los espacios en blanco antes y después del contenido del texto (por defecto: `False`)
- *rewrite_prefixes*: configurar a `True` para sustituir los prefijos de espacios de nombres por «n{number}» (por defecto: `False`)
- *qname_aware_tags*: un conjunto de nombres de etiquetas conscientes de qname en el que los prefijos deben ser reemplazados en el contenido del texto (por defecto: vacío)

- *qname_aware_attrs*: un conjunto de nombres de atributos conscientes de qname en el que los prefijos
  deben ser reemplazados en el contenido del texto (por defecto: vacío)

- *exclude_attrs*: un conjunto de nombres de atributos que no deben serializarse
- *exclude_tags*: un conjunto de nombres de etiquetas que no deben serializarse

En la lista de opciones anterior, «un conjunto» se refiere a cualquier colección o iterable de cadenas, no se espera ningún orden.

Added in version 3.8.

xml.etree.ElementTree.Comment(*text=None*)

Fábrica de elementos de comentario. Esta función de fábrica crea un elemento especial que será serializado como un comentario XML por el serializador estándar. La cadena de comentario puede ser una cadena de bytes o una cadena Unicode. *text* es una cadena que contiene la cadena de comentario. Devuelve una instancia de elemento que representa un comentario.

Tenga en cuenta que `XMLParser` omite los comentarios en la entrada en lugar de crear objetos de comentario para ellos. Un `ElementTree` sólo contendrá nodos de comentario si se han insertado en el árbol utilizando uno de los métodos `Element`.

xml.etree.ElementTree.dump(*elem*)

Escribe un árbol de elementos o una estructura de elementos en sys.stdout. Esta función debe utilizarse únicamente para debugging.

El formato de salida exacto depende de la implementación. En esta versión, se escribe como un archivo XML ordinario.

elem es un árbol de elementos o un elemento individual.

Distinto en la versión 3.8: La función `dump()` ahora preserva el orden de atributos especificado por el usuario.

xml.etree.ElementTree.fromstring(*text*, *parser=None*)

Analiza una sección XML a partir de una constante de cadena. Igual que `XML()`. *text* es una cadena que contiene datos XML. *parser* es una instancia de parser opcional. Si no se da, se utiliza el analizador estándar `XMLParser`. Devuelve una instancia de `Element`.

xml.etree.ElementTree.fromstringlist(*sequence*, *parser=None*)

Analiza un documento XML a partir de una secuencia de fragmentos de cadena de caracteres. *sequence* es una lista u otra secuencia que contiene fragmentos de datos XML. *parser* es una instancia de parser opcional. Si no se da, se utiliza el analizador estándar `XMLParser`. Retorna una instancia de `Element`.

Added in version 3.2.

xml.etree.ElementTree.indent(*tree*, *space=' '*, *level=0*)

Añade espacios en blanco al subárbol para indentar el árbol visualmente. Esto puede utilizarse para generar una salida XML con una impresión bonita. *tree* puede ser un Element o ElementTree. *space* es la cadena de espacio en blanco que se insertará para cada nivel de indentación, dos caracteres de espacio por defecto. Para indentar subárboles parciales dentro de un árbol ya indentado, pase el nivel de indentación inicial como *level*.

Added in version 3.9.

xml.etree.ElementTree.iselement(*element*)

Comprueba si un objeto parece ser un objeto elemento válido. *element* es una instancia de elemento. Retorna `True` si se trata de un objeto elemento.

xml.etree.ElementTree.iterparse(*source*, *events=None*, *parser=None*)

Parses an XML section into an element tree incrementally, and reports what's going on to the user. *source* is a filename or file object containing XML data. *events* is a sequence of events to report back. The supported events are the strings `"start"`, `"end"`, `"comment"`, `"pi"`, `"start-ns"` and `"end-ns"` (the «ns» events are used to get detailed namespace information). If *events* is omitted, only `"end"` events are reported. *parser* is an optional parser instance. If not given, the standard **`XMLParser`** parser is used. *parser* must be a subclass of **`XMLParser`** and can only use the default **`TreeBuilder`** as a target. Returns an iterator providing `(event, elem)` pairs; it has a `root` attribute that references the root element of the resulting XML tree once *source* is fully read. The iterator has the **`close()`** method that closes the internal file object if *source* is a filename.

Tenga en cuenta que mientras **`iterparse()`** construye el árbol de forma incremental, emite lecturas de bloqueo en la *source* (o en el fichero que nombra). Por lo tanto, no es adecuado para aplicaciones en las que no se pueden realizar lecturas de bloqueo. Para un análisis completamente no bloqueante, véase **`XMLPullParser`**.

> **Nota**
>
> **`iterparse()`** sólo garantiza que ha visto el carácter «>» de una etiqueta de inicio cuando emite un evento «start», por lo que los atributos están definidos, pero el contenido de los atributos text y tail está indefinido en ese momento. Lo mismo ocurre con los hijos del elemento; pueden estar presentes o no.
>
> Si necesita un elemento totalmente poblado, busque los eventos «end» en su lugar.

Obsoleto desde la versión 3.4: El argumento *parser*.

Distinto en la versión 3.8: Los eventos `comment` y `pi` han sido añadidos.

Distinto en la versión 3.13: Added the **`close()`** method.

xml.etree.ElementTree.parse(*source*, *parser=None*)

Analiza una sección XML en un árbol de elementos. *source* es un nombre de archivo o un objeto de archivo que contiene datos XML. *parser* es una instancia de parser opcional. Si no se da, se utiliza el analizador estándar **`XMLParser`**. Devuelve una instancia de **`ElementTree`**.

xml.etree.ElementTree.ProcessingInstruction(*target*, *text=None*)

Fábrica de elementos PI. Esta función de fábrica crea un elemento especial que será serializado como una instrucción de procesamiento XML. *target* es una cadena que contiene el objetivo de PI. *text* es una cadena que contiene el contenido de PI, si se da. Devuelve una instancia de elemento, representando una instrucción de procesamiento.

Note that **`XMLParser`** skips over processing instructions in the input instead of creating PI objects for them. An **`ElementTree`** will only contain processing instruction nodes if they have been inserted into to the tree using one of the **`Element`** methods.

xml.etree.ElementTree.register_namespace(*prefix*, *uri*)

Registra un prefijo de espacio de nombres. El registro es global, y cualquier asignación existente para el prefijo dado o el URI del espacio de nombres será eliminado. *prefix* es un prefijo de espacio de nombres. *uri* es una uri del espacio de nombres. Las etiquetas y los atributos de este espacio de nombres se serializarán con el prefijo dado, si es posible.

Added in version 3.2.

xml.etree.ElementTree.SubElement(*parent*, *tag*, *attrib={}*, ***extra*)

Fábrica de sub-elementos. Esta función crea una instancia de elemento y la añade a un elemento existente.

El nombre del elemento, los nombres de los atributos y los valores de los atributos pueden ser cadenas de bytes o cadenas de caracteres Unicode. *parent* es el elemento padre. *tag* es el nombre del sub-elemento. *attrib* es un diccionario opcional que contiene los atributos del elemento. *extra* contiene atributos adicionales, dados como argumentos de palabras clave. Devuelve una instancia de elemento.

xml.etree.ElementTree.tostring(*element*, *encoding='us-ascii'*, *method='xml'*, ***, *xml_declaration=None*, *default_namespace=None*, *short_empty_elements=True*)

Genera una representación de cadena de caracteres de un elemento XML, incluyendo todos los sub-elementos. *element* es una instancia de **`Element`**. *encoding* [1] es la codificación de salida (por defecto es US-ASCII). Utilice `encoding="unicode"` para generar una cadena de caracteres Unicode (de lo contrario, se genera una cadena de bytes). *method* es `"xml"`, `"html"` o `"text"` (por defecto es `"xml"`). *xml_declaration*, *default_namespace* y *short_empty_elements* tienen el mismo significado que en **`ElementTree.write()`**. Devuelve una cadena (opcionalmente) codificada que contiene los datos XML.

Distinto en la versión 3.4: Added the *short_empty_elements* parameter.

Distinto en la versión 3.8: Added the *xml_declaration* and *default_namespace* parameters.

Distinto en la versión 3.8: La función **`tostring()`** ahora preserva el orden de atributos especificado por el usuario.

xml.etree.ElementTree.tostringlist(*element*, *encoding='us-ascii'*, *method='xml'*, ***, *xml_declaration=None*, *default_namespace=None*, *short_empty_elements=True*)

Genera una representación de cadena de caracteres de un elemento XML, incluyendo todos los sub-elementos. *element* es una instancia de **`Element`**. *encoding* [1] es la codificación de salida (por defecto es US-ASCII). Utilice `encoding="unicode"` para generar una cadena de caracteres Unicode (de lo contrario, se genera una cadena de bytes). *method* es `"xml"`, `"html"` o `"text"` (por defecto es `"xml"`). *xml_declaration*, *default_namespace* y *short_empty_elements* tienen el mismo significado que en **`ElementTree.write()`**. Devuelve una lista de cadenas (opcionalmente) codificadas que contienen los datos XML. No garantiza ninguna secuencia específica, excepto que `b"".join(tostringlist(element)) == tostring(element)`.

Added in version 3.2.

Distinto en la versión 3.4: Added the *short_empty_elements* parameter.

Distinto en la versión 3.8: Added the *xml_declaration* and *default_namespace* parameters.

Distinto en la versión 3.8: La función **`tostringlist()`** ahora preserva el orden de atributos especificado por el usuario.

xml.etree.ElementTree.XML(*text*, *parser=None*)

Analiza una sección XML a partir de una constante de cadena de caracteres. Esta función puede utilizarse para incrustar «literales XML» en el código de Python. *text* es una cadena de caracteres que contiene datos XML. *parser* es una instancia de parser opcional. Si no se da, se utiliza el analizador estándar **`XMLParser`**. Devuelve una instancia de **`Element`**.

xml.etree.ElementTree.XMLID(*text*, *parser=None*)

Analiza una sección XML a partir de una constante de cadena de caracteres, y también devuelve un diccionario que mapea los id:s de elementos a elementos. *text* es una cadena de caracteres que contiene datos XML. *parser* es una instancia de parser opcional. Si no se da, se utiliza el analizador estándar de **`XMLParser`**. Devuelve una tupla que contiene una instancia de **`Element`** y un diccionario.

Soporte de XInclude

Este módulo proporciona un soporte limitado para las directivas XInclude [https://www.w3.org/TR/xinclude/], a través del módulo de ayuda **`xml.etree.ElementInclude`**. Este módulo puede utilizarse para insertar subárboles y cadenas de texto en árboles de elementos, basándose en la información del árbol.

Ejemplo

Aquí hay un ejemplo que demuestra el uso del módulo XInclude. Para incluir un documento XML en el documento actual, utilice el elemento `{http://www.w3.org/2001/XInclude}include` y establezca el atributo **parse** como `"xml"`, y utilice el atributo **href** para especificar el documento a incluir.

```
<?xml version="1.0"?>
<document xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include href="source.xml" parse="xml" />
</document>
```

Por defecto, el atributo **href** se trata como un nombre de archivo. Puede utilizar cargadores personalizados para anular este comportamiento. También tenga en cuenta que el ayudante estándar no soporta la sintaxis XPointer.

To process this file, load it as usual, and pass the root element to the **`xml.etree.ElementTree`** module:

```
from xml.etree import ElementTree, ElementInclude

tree = ElementTree.parse("document.xml")
root = tree.getroot()

ElementInclude.include(root)
```

El módulo ElementInclude sustituye el elemento `{http://www.w3.org/2001/XInclude}include` por el elemento raíz del documento **source.xml**. El resultado podría ser algo así:

```
<document xmlns:xi="http://www.w3.org/2001/XInclude">
  <para>This is a paragraph.</para>
```

```
</document>
```

Si se omite el atributo **parse**, el valor por defecto es «xml». El atributo href es obligatorio.

Para incluir un documento de texto, utilice el elemento `{http://www.w3.org/2001/XInclude}include` y establezca el atributo **parse** como «text»:

```
<?xml version="1.0"?>
<document xmlns:xi="http://www.w3.org/2001/XInclude">
  Copyright (c) <xi:include href="year.txt" parse="text"
</document>
```

El resultado podría ser algo así:

```
<document xmlns:xi="http://www.w3.org/2001/XInclude">
  Copyright (c) 2003.
</document>
```

Referencia

Funciones

xml.etree.ElementInclude.default_loader(*href*, *parse*, *encoding=None*)

> Default loader. This default loader reads an included resource from disk. *href* is a URL. *parse* is for parse mode either «xml» or «text». *encoding* is an optional text encoding. If not given, encoding is `utf-8`. Returns the expanded resource. If the parse mode is `"xml"`, this is an **`Element`** instance. If the parse mode is `"text"`, this is a string. If the loader fails, it can return `None` or raise an exception.

xml.etree.ElementInclude.include(*elem*, *loader=None*, *base_url=None*, *max_depth=6*)

> This function expands XInclude directives in-place in tree pointed by *elem*. *elem* is either the root **`Element`** or an

`ElementTree` instance to find such element. *loader* is an optional resource loader. If omitted, it defaults to `default_loader()`. If given, it should be a callable that implements the same interface as **`default_loader()`**. *base_url* is base URL of the original file, to resolve relative include file references. *max_depth* is the maximum number of recursive inclusions. Limited to reduce the risk of malicious content explosion. Pass `None` to disable the limitation.

Distinto en la versión 3.9: Added the *base_url* and *max_depth* parameters.

Objetos Element

class xml.etree.ElementTree.Element(*tag*, *attrib={}*, ***extra*)

Clase Element. Esta clase define la interfaz Element, y provee una implementación de referencia de esta interfaz.

El nombre del elemento, los nombres de los atributos y los valores de los atributos pueden ser cadenas de bytes o cadenas Unicode. *tag* es el nombre del elemento. *attrib* es un diccionario opcional que contiene los atributos del elemento. *extra* contiene atributos adicionales, dados como argumentos de palabras clave.

tag

Una cadena de caracteres que identifica qué tipo de datos representa este elemento (el tipo de elemento, en otras palabras).

text
tail

Estos atributos pueden utilizarse para contener datos adicionales asociados al elemento. Sus valores suelen ser cadenas, pero pueden ser cualquier objeto específico de la aplicación. Si el elemento se crea a partir de un archivo XML, el atributo *text* contiene el texto entre la etiqueta inicial del elemento y su primera etiqueta hija o final, o `None`, y el atributo *tail* contiene el texto entre la etiqueta final del elemento y la siguiente etiqueta, o `None`. Para los datos XML

```
<a><b>1<c>2<d/>3</c></b>4</a>
```

el elemento *a* tiene `None` para los atributos *text* y *tail*, el elemento *b* tiene *text* `"1"` y *tail* `"4"`, el elemento *c* tiene *text* `"2"` y *tail* `None`, y el elemento *d* tiene *text* `None` y *tail* `"3"`.

Para recoger el texto interior de un elemento, véase **`itertext()`**, por ejemplo `"".join(element.itertext())`.

Las aplicaciones pueden almacenar objetos arbitrarios en estos atributos.

attrib

Un diccionario que contiene los atributos del elemento. Ten en cuenta que aunque el valor *attrib* es siempre un diccionario mutable real de Python, una implementación de ElementTree puede elegir utilizar otra representación interna, y crear el diccionario sólo si alguien lo pide. Para aprovechar este tipo de implementaciones, utiliza los métodos de diccionario que aparecen a continuación siempre que sea posible.

Los siguientes métodos tipo diccionario funcionan con los atributos de los elementos.

clear()

Restablece un elemento. Esta función elimina todos los sub-elementos, borra todos los atributos y establece los atributos de texto y cola como `None`.

get(*key*, *default=None*)

Obtiene el atributo del elemento llamado *key*.

Retorna el valor del atributo, o *default* si el atributo no fue encontrado.

items()

Retorna los atributos del elemento como una secuencia de pares (nombre, valor). Los atributos se retornan en un orden arbitrario.

keys()

Retorna los nombres de los atributos de los elementos como una lista. Los nombres se retornan en un orden arbitrario.

set(*key*, *value*)

Establecer el atributo *key* en el elemento a *value*.

Los siguientes métodos actúan sobre los hijos del elemento (sub-elementos).

append(*subelement*)

Añade el elemento *subelement* al final de la lista interna de sub-elementos de este elemento. Lanza **`TypeError`** si *subelement* no es un **`Element`**.

extend(*subelements*)

Appends *subelements* from an iterable of elements. Raises **`TypeError`** if a subelement is not an **`Element`**.

Added in version 3.2.

find(*match*, *namespaces=None*)

Encuentra el primer sub-elemento que coincide con *match*. *match* puede ser un nombre de etiqueta o un path. Retorna una instancia de elemento o `None`. *namespaces* es un mapeo opcional del prefijo del espacio de nombres al nombre completo. Pasa `''` como prefijo para mover todos los nombres de etiquetas sin prefijo en la expresión al espacio de nombres dado.

findall(*match*, *namespaces*=*None*)

Encuentra todos los sub-elementos coincidentes, por nombre de etiqueta o path. Retorna una lista que contiene todos los elementos coincidentes en el orden del documento. *namespaces* es un mapeo opcional del prefijo del espacio de nombres al nombre completo. Pasa `''` como prefijo para mover todos los nombres de etiquetas sin prefijo en la expresión al espacio de nombres dado.

findtext(*match*, *default*=*None*, *namespaces*=*None*)

Busca el texto del primer sub-elemento que coincida con *match*. *match* puede ser un nombre de etiqueta o un path. Retorna el contenido del texto del primer elemento que coincida, o *default* si no se encuentra ningún elemento. Tenga en cuenta que si el elemento coincidente no tiene contenido de texto se devuelve una cadena vacía. *namespaces* es un mapeo opcional del prefijo del espacio de nombres al nombre completo. Pasa `''` como prefijo para mover todos los nombres de etiquetas sin prefijo en la expresión al espacio de nombres dado.

insert(*index*, *subelement*)

Inserta *subelement* en la posición dada en este elemento. Lanza **`TypeError`** si *subelement* no es un **`Element`**.

iter(*tag*=*None*)

Crea un árbol iterator con el elemento actual como raíz. El iterador itera sobre este elemento y todos los elementos por debajo de él, en el orden del documento (profundidad primero). Si *tag* no es `None` o `'*'`, sólo los elementos cuya etiqueta es igual a *tag* son retornados por el iterador. Si la estructura del árbol se modifica durante la iteración, el resultado es indefinido.

Added in version 3.2.

iterfind(*match*, *namespaces=None*)

Encuentra todos los subelementos que coinciden, por nombre de etiqueta o ruta. Retorna un iterable con todos los elementos coincidentes en el orden del documento. *namespaces* es un mapeo opcional del prefijo del espacio de nombres al nombre completo.

Added in version 3.2.

itertext()

Crea un iterador de texto. El iterador hace un bucle sobre este elemento y todos los subelementos, en el orden del documento, y retorna todo el texto interior.

Added in version 3.2.

makeelement(*tag*, *attrib*)

Crea un nuevo objeto elemento del mismo tipo que este elemento. No llame a este método, utilice la función de fábrica **`SubElement()`** en su lugar.

remove(*subelement*)

Elimina el *subelement* del elemento. A diferencia de los métodos find*, este método compara los elementos basándose en la identidad de la instancia, no en el valor de la etiqueta o el contenido.

Los objetos **`Element`** también soportan los siguientes métodos de tipo secuencia para trabajar con subelementos: **`__delitem__()`**, **`__getitem__()`**, **`__setitem__()`**, **`__len__()`**.

Caution: Elements with no subelements will test as `False`. In a future release of Python, all elements will test as `True` regardless of whether subelements exist. Instead, prefer explicit `len(elem)` or `elem is not None` tests.:

```
element = root.find('foo')

if not element:  # careful!
    print("element not found, or element has no sub

if element is None:
    print("element not found")
```

Distinto en la versión 3.12: Testing the truth value of an Element emits **`DeprecationWarning`**.

Antes de Python 3.8, el orden de serialización de los atributos XML de los elementos se hacía predecible artificialmente ordenando los atributos por su nombre. Basado en el ordenamiento -ahora garantizado- de los diccionarios, este reordenamiento arbitrario fue eliminado en Python 3.8 para preservar el orden en que los atributos fueron originalmente analizados o creados por el código del usuario.

En general, el código del usuario debería intentar no depender de un orden específico de los atributos, dado que el XML Information Set [https://www.w3.org/TR/xml-infoset/] excluye explícitamente el orden de los atributos para transmitir información. El código debe estar preparado para hacer frente a cualquier orden en la entrada. En los casos en los que se requiere una salida XML determinista, por ejemplo, para la firma criptográfica o los conjuntos de datos de prueba, la serialización canónica está disponible con la función **`canonicalize()`**.

En los casos en los que la salida canónica no es aplicable, pero un orden de atributos específico sigue siendo deseable en la salida, el código debe tratar de crear los atributos directamente en el orden deseado, para evitar desajustes perceptivos para los lectores del código. En los casos en que esto es difícil de lograr, una receta como la siguiente se puede aplicar antes de la serialización para hacer cumplir un orden independientemente de la creación de elementos:

```
def reorder_attributes(root):
```

```
for el in root.iter():
    attrib = el.attrib
    if len(attrib) > 1:
        # adjust attribute order, e.g. by sorti
        attribs = sorted(attrib.items())
        attrib.clear()
        attrib.update(attribs)
```

Objetos ElementTree

class xml.etree.ElementTree.ElementTree(*element=None*, *file=None*)

Clase envoltorio de un ElementTree. Esta clase representa una jerarquía de elementos completa, y añade algún soporte extra para la serialización hacia y desde XML estándar.

element es el elemento raíz. El árbol se inicializa con el contenido del *file* XML si se da.

_setroot(*element*)

Reemplaza el elemento raíz de este árbol. Esto descarta el contenido actual del árbol, y lo reemplaza con el elemento dado. Utilícelo con cuidado. *element* es una instancia de elemento.

find(*match*, *namespaces=None*)

Igual que **`Element.find()`**, empezando por la raíz del árbol.

findall(*match*, *namespaces=None*)

Igual que **`Element.findall()`**, empezando por la raíz del árbol.

findtext(*match*, *default=None*, *namespaces=None*)

Igual que **`Element.findtext()`**, empezando por la raíz del árbol.

getroot()

Retorna el elemento raíz de este árbol.

iter(*tag=None*)

Crea y retorna un iterador de árbol para el elemento raíz. El iterador recorre todos los elementos de este árbol, en orden de sección. *tag* es la etiqueta a buscar (por defecto devuelve todos los elementos).

iterfind(*match, namespaces=None*)

Igual que **`Element.iterfind()`**, empezando por la raíz del árbol.

Added in version 3.2.

parse(*source, parser=None*)

Carga una sección XML externa en este árbol de elementos. *source* es un nombre de archivo o un file object. *parser* es una instancia opcional del analizador. Si no se da, se utiliza el analizador estándar **`XMLParser`**. Retorna el elemento raíz de la sección.

write(*file, encoding='us-ascii', xml_declaration=None, default_namespace=None, method='xml', *, short_empty_elements=True*)

Escribe el árbol de elementos en un archivo, como XML. *file* es un nombre de archivo, o un file object abierto para escritura. *encoding* [1] es la codificación de salida (por defecto es US-ASCII). *xml_declaration* controla si se debe añadir una declaración XML al archivo. Utilice `False` para nunca, `True` para siempre, `None` para sólo si no es US-ASCII o UTF-8 o Unicode (por defecto es `None`). *default_namespace* establece el espacio de nombres XML por defecto (para «xmlns»). *method* es `"xml"`, `"html"` o `"text"` (por defecto es `"xml"`). El parámetro *short_empty_elements* controla el formato de los elementos sin contenido. Si es `True` (por defecto), se emiten como una sola etiqueta autocerrada, de lo contrario se emiten como un par de etiquetas de inicio/fin.

La salida es una cadena de caracteres (**`str`**) o binaria (**`bytes`**). Esto es controlado por el argumento *encoding*. Si *encoding* es `unicode`, la salida es una cadena de caracteres; en caso contrario, es binaria. Tenga en cuenta que esto puede entrar en conflicto con el tipo de *file* si es un file object abierto; asegúrese de no intentar escribir una cadena en un flujo binario y viceversa.

Distinto en la versión 3.4: Added the *short_empty_elements* parameter.

Distinto en la versión 3.8: El método **`write()`** ahora conserva el orden de los atributos especificado por el usuario.

Este es el archivo XML que será manipulado:

```
<html>
    <head>
        <title>Example page</title>
    </head>
    <body>
        <p>Moved to <a href="http://example.org/">exampl
        or <a href="http://example.com/">example.com</a>
    </body>
</html>
```

Ejemplo de cambio del atributo «target» de cada enlace en el primer párrafo:

```
>>> from xml.etree.ElementTree import ElementTree
>>> tree = ElementTree()
>>> tree.parse("index.xhtml")
<Element 'html' at 0xb77e6fac>
>>> p = tree.find("body/p")     # Finds first occurrence
>>> p
```

```
<Element 'p' at 0xb77ec26c>
>>> links = list(p.iter("a"))   # Returns list of all li
>>> links
[<Element 'a' at 0xb77ec2ac>, <Element 'a' at 0xb77ec1cc
>>> for i in links:             # Iterates through all f
...     i.attrib["target"] = "blank"
...
>>> tree.write("output.xhtml")
```

Objetos QName

class xml.etree.ElementTree.QName(*text_or_uri*, *tag=None*)

QName wrapper. Se puede utilizar para envolver un valor de atributo QName, con el fin de obtener un manejo adecuado del espacio de nombres en la salida. *text_or_uri* es una cadena de caracteres que contiene el valor QName, en la forma {uri}local, o, si se da el argumento tag, la parte URI de un QName. Si se da *tag*, el primer argumento se interpreta como un URI, y este argumento se interpreta como un nombre local. Las instancias de `QName` son opacas.

Objetos TreeBuilder

class xml.etree.ElementTree.TreeBuilder(*element_factory=None*, ***, *comment_factory=None*, *pi_factory=None*, *insert_comments=False*, *insert_pis=False*)

Constructor genérico de estructuras de elementos. Este constructor convierte una secuencia de llamadas a los métodos start, data, end, comment y pi en una estructura de elementos bien formada. Puedes utilizar esta clase para construir una estructura de elementos utilizando un analizador XML personalizado, o un analizador para algún otro formato similar a XML.

element_factory, cuando se da, debe ser un callable que acepta dos argumentos posicionales: una etiqueta y un diccionario de atributos. Se espera que retorne una nueva instancia de elemento.

Las funciones *comment_factory* y *pi_factory*, cuando se dan, deben comportarse como las funciones **`Comment()`** y **`ProcessingInstruction()`** para crear comentarios e instrucciones de procesamiento. Si no se dan, se utilizarán las fábricas por defecto. Cuando *insert_comments* y/o *insert_pis* es verdadero, los comentarios/pis se insertarán en el árbol si aparecen dentro del elemento raíz (pero no fuera de él).

close()

> Vacía los buffers del constructor y retorna el elemento del documento de nivel superior. Retorna una instancia de **`Element`**.

data(*data*)

> Añade texto al elemento actual. *data* es una cadena. Debe ser una cadena de bytes o una cadena Unicode.

end(*tag*)

> Cierra el elemento actual. *tag* es el nombre del elemento. Retorna el elemento cerrado.

start(*tag*, *attrs*)

> Abre un nuevo elemento. *tag* es el nombre del elemento. *attrs* es un diccionario que contiene los atributos del elemento. Retorna el elemento abierto.

comment(*text*)

> Crea un comentario con el *texto* dado. Si `insert_comments` es verdadero, esto también lo añadirá al árbol.
>
> *Added in version 3.8.*

pi(*target*, *text*)

> Creates a process instruction with the given *target* name and *text*. If `insert_pis` is true, this will also add it to the tree.

Added in version 3.8.

Además, un objeto **`TreeBuilder`** personalizado puede proporcionar los siguientes métodos:

doctype(*name*, *pubid*, *system*)

Maneja una declaración de doctype. *name* es el nombre del doctype. *pubid* es el identificador público. *system* es el identificador del sistema. Este método no existe en la clase por defecto **`TreeBuilder`**.

Added in version 3.2.

start_ns(*prefix*, *uri*)

Se llama cada vez que el analizador encuentra una nueva declaración de espacio de nombres, antes de la llamada de retorno `start()` para el elemento de apertura que lo define. *prefix* es `''` para el espacio de nombres por defecto y el nombre del prefijo del espacio de nombres declarado en caso contrario. *uri* es el URI del espacio de nombres.

Added in version 3.8.

end_ns(*prefix*)

Se llama después de la llamada de retorno `end()` de un elemento que declaró un mapeo de prefijo de espacio de nombres, con el nombre del *prefijo* que salió del ámbito.

Added in version 3.8.

class xml.etree.ElementTree.C14NWriterTarget(*write*, ***, *with_comments=False*, *strip_text=False*, *rewrite_prefixes=False*, *qname_aware_tags=None*, *qname_aware_attrs=None*, *exclude_attrs=None*, *exclude_tags=None*)

Un escritor C14N 2.0 [https://www.w3.org/TR/xml-c14n2/]. Los argumentos son los mismos que para la función `canonicalize()`. Esta clase no construye un árbol, sino que traduce los eventos de devolución de llamada directamente a una forma serializada utilizando la función *write*.

Added in version 3.8.

Objetos XMLParser

class xml.etree.ElementTree.XMLParser(*, *target=None*, *encoding=None*)

Esta clase es el bloque de construcción de bajo nivel del módulo. Utiliza `xml.parsers.expat` para un análisis eficiente de XML basado en eventos. Puede ser alimentada con datos XML de forma incremental con el método `feed()`, y los eventos de análisis se traducen en una API push - invocando callbacks en el objeto *target*. Si se omite *target*, se utiliza la clase estándar `TreeBuilder`. Si se da *encoding* [1], el valor anula la codificación especificada en el archivo XML.

Distinto en la versión 3.8: Parameters are now keyword-only. The *html* argument is no longer supported.

close()

Finaliza la alimentación de datos al analizador. Retorna el resultado de llamar al método `close()` del *target* pasado durante la construcción; por defecto, es el elemento del documento de nivel superior.

feed(*data*)

Introduce los datos en el analizador sintáctico. *data* son datos codificados.

flush()

Triggers parsing of any previously fed unparsed data, which can be used to ensure more immediate feedback, in particular with Expat $>=$2.6.0. The implementation of **`flush()`** temporarily disables reparse deferral with Expat (if currently enabled) and triggers a reparse.

Disabling reparse deferral has security consequences; please see `xml.parsers.expat.xmlparser.SetReparseDeferralE` for details.

Note that **`flush()`** has been backported to some prior releases of CPython as a security fix. Check for availability of **`flush()`** using **`hasattr()`** if used in code running across a variety of Python versions.

Added in version 3.13.

`XMLParser.feed()` llama al método `start(tag, attrs_dict)` de *target* para cada etiqueta de apertura, a su método `end(tag)` para cada etiqueta de cierre, y los datos son procesados por el método `data(data)`. Para más métodos de callback soportados, véase la clase **`TreeBuilder`**. **`XMLParser.close()`** llama al método `close()` de *target*. **`XMLParser`** puede utilizarse no sólo para construir una estructura de árbol. Este es un ejemplo de contar la profundidad máxima de un archivo XML:

```
>>> from xml.etree.ElementTree import XMLParser
>>> class MaxDepth:                     # The targe
...     maxDepth = 0
...     depth = 0
...     def start(self, tag, attrib):   # Called fo
...         self.depth += 1
...         if self.depth > self.maxDepth:
...             self.maxDepth = self.depth
...     def end(self, tag):             # Called fo
...         self.depth -= 1
...     def data(self, data):
...         pass            # We do not need to do
...     def close(self):    # Called when all data
...         return self.maxDepth
...
>>> target = MaxDepth()
>>> parser = XMLParser(target=target)
>>> exampleXml = """
```

```
... <a>
...   <b>
...   </b>
...   <b>
...     <c>
...       <d>
...       </d>
...     </c>
...   </b>
... </a>"""
>>> parser.feed(exampleXml)
>>> parser.close()
4
```

Objetos XMLPullParser

class xml.etree.ElementTree.XMLPullParser(*events=None*)

Un analizador sintáctico pull adecuado para aplicaciones no bloqueantes. Su API de entrada es similar a la de `XMLParser`, pero en lugar de enviar llamadas a un objetivo de devolución de llamada, `XMLPullParser` recoge una lista interna de eventos de análisis y permite al usuario leer de ella. *events* son una secuencia de eventos a reportar. Los eventos soportados son las cadenas `"start"`, `"end"`, `"comment"`, `"pi"`, `"start-ns"` y `"end-ns"` (los eventos «ns» se utilizan para obtener información detallada del espacio de nombres). Si se omite *events*, sólo se informará de los eventos `"end"`.

feed(*data*)

Introduce los datos de los bytes dados en el analizador.

flush()

Triggers parsing of any previously fed unparsed data, which can be used to ensure more immediate feedback, in particular with Expat > =2.6.0. The implementation of **`flush()`** temporarily disables reparse deferral with Expat (if currently enabled) and triggers a reparse.

Disabling reparse deferral has security consequences; please see `xml.parsers.expat.xmlparser.SetReparseDeferralE` for details.

Note that `flush()` has been backported to some prior releases of CPython as a security fix. Check for availability of `flush()` using `hasattr()` if used in code running across a variety of Python versions.

Added in version 3.13.

close()

Señala al analizador que el flujo de datos ha terminado. A diferencia de `XMLParser.close()`, este método siempre retorna `None`. Cualquier evento que no haya sido recuperado cuando el analizador se cierra puede ser leído con `read_events()`.

read_events()

Retorna un iterador sobre los eventos que se han encontrado en los datos alimentados al analizador. El iterador retorna pares `(event, elem)`, donde *event* es una cadena de caracteres que representa el tipo de evento (por ejemplo, `"fin"`) y *elem* es el objeto `Element` encontrado, u otro valor de contexto como el siguiente.

- `start`, `end`: el Element actual.
- `comment`, `pi`: el comentario / la instrucción de procesamiento actual
- `start-ns`: una tupla `(prefix, uri)` que nombra el mapeo del espacio de nombres declarado.
- `end-ns`: `None` (esto puede cambiar en una versión futura)

Los eventos proporcionados en una llamada anterior a `read_events()` no serán retornados nuevamente. Los eventos se consumen de la cola interna sólo cuando se recuperan del iterador, por lo que múltiples lectores iterando en paralelo sobre iteradores obtenidos de **`read_events()`** tendrán resultados impredecibles.

Nota

`XMLPullParser` sólo garantiza que ha visto el carácter «>» de una etiqueta de inicio cuando emite un evento «start», por lo que los atributos están definidos, pero el contenido de los atributos text y tail está indefinido en ese momento. Lo mismo ocurre con los hijos del elemento; pueden estar presentes o no.

Si necesita un elemento totalmente poblado, busque los eventos «end» en su lugar.

Added in version 3.4.

Distinto en la versión 3.8: Los eventos `comment` y `pi` han sido añadidos.

Excepciones

class xml.etree.ElementTree.ParseError

Error de análisis de XML, lanzado por los distintos métodos de análisis de este módulo cuando el análisis falla. La representación en cadena de caracteres de una instancia de esta excepción contendrá un mensaje de error fácil de entender. Además, tendrá los siguientes atributos disponibles:

code

Un código de error numérico del analizador sintáctico expat. Consulte la documentación de **`xml.parsers.expat`** para ver la lista de códigos de error y sus significados.

position

Una tupla de números de *line*, *column*, que especifica dónde se produjo el error.

Notas al pie de página

La cadena de caracteres de codificación incluida en la salida XML debe ajustarse a los estándares adecuados. Por ejemplo, «UTF-8» es válido, pero «UTF8» no lo es. Consulte https://www.w3.org/TR/2006/REC-xml11-20060816/#NT-EncodingDecl y https://www.iana.org/assignments/character-sets/character-sets.xhtml.

`xml.dom` — The Document Object Model API

Código Fuente: Lib/xml/dom/__init__.py [https://github.com/python/cpython/tree/3.14/Lib/xml/dom/__init__.py]

El Modelo de Objetos del Documento, o «DOM» por sus siglas en inglés, es un lenguaje API del Consorcio *World Wide Web* (W3C) para acceder y modificar documentos XML. Una implementación del DOM presenta los documento XML como un árbol, o permite al código cliente construir dichas estructuras desde cero para luego darles acceso a la estructura a través de un conjunto de objetos que implementaron interfaces conocidas.

El DOM es extremadamente útil para aplicaciones de acceso directo. SAX sólo te permite la vista de una parte del documento a la vez. Si estás mirando un elemento SAX, no tienes acceso a otro. Si estás viendo un nodo de texto, no tienes acceso al elemento contenedor. Cuando desarrollas una aplicación SAX, necesitas registrar la posición de tu programa en el documento en algún lado de tu código. SAX no lo hace por ti. Además, desafortunadamente no podrás mirar hacia adelante (*look ahead*) en el documento XML.

Algunas aplicaciones son imposibles en un modelo orientado a eventos sin acceso a un árbol. Por supuesto que puedes construir algún tipo de árbol por tu cuenta en eventos SAX, pero el DOM te evita escribir ese código. El DOM es una representación de árbol estándar para datos XML.

El Modelo de Objetos del Documento es definido por el *W3C* en fases, o «niveles» en su terminología. El mapeado de Python de la API está basado en la recomendación del DOM nivel 2.

Las aplicaciones DOM típicamente empiezan al diseccionar (*parse*) el XML en un DOM. Cómo esto funciona no está incluido en el DOM nivel 1, y el nivel 2 provee mejoras limitadas. Existe una clase objeto llamada `DOMImplementation` que da acceso a métodos de creación de `Document`, pero de ninguna forma da acceso a los constructores (*builders*) de *reader/parser/Document* de una forma independiente a la implementación. No hay una forma clara para acceder a estos método sin un objeto `Document` existente. En Python, cada implementación del DOM proporcionará una función `getDOMImplementation()`. El DOM de nivel 3 añade una especificación para Cargar(*Load*)/Guardar(*Store*), que define una interfaz al lector (*reader*), pero no está disponible aún en la librería estándar de Python.

Una vez que tengas un objeto del documento del DOM, puedes acceder a las partes de tu documento XML a través de sus propiedades y métodos. Estas propiedades están definidas en la especificación del DOM; está porción del manual describe la interpretación de la especificación en Python.

La especificación estipulada por el *W3C* define la *DOM API* para Java, ECMAScript, y OMG IDL. El mapeo de Python definido aquí está basado en gran parte en la versión IDL de la especificación, pero no se requiere el cumplimiento estricto (aunque las implementaciones son libres de soportar el mapeo estricto de IDL). Véase la sección Conformidad para una discusión detallada del mapeo de los requisitos.

Ver también

Document Object Model (DOM) Level 2 Specification [https://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/]

La recomendación del *W3C* con la cual se basa el *DOM API* de Python.

Document Object Model (DOM) Level 1 Specification [https://www.w3.org/TR/REC-DOM-Level-1/]

La recomendación del W3C para el DOM soportada por `xml.dom.minidom`.

Python Language Mapping Specification [https://www.omg.org/spec/PYTH/1.2/PDF]

Este documento especifica el mapeo de OMG IDL a Python.

Contenido del módulo

The `xml.dom` contains the following functions:

xml.dom.registerDOMImplementation(*name*, *factory*)

Registra la función *factory* con el nombre *name*. La función fábrica (*factory*) debe retornar un objeto que implemente la interfaz **`DOMImplementation`**. La función fábrica puede retornar el mismo objeto cada vez que se llame, o uno nuevo por cada llamada, según sea apropiado para la implementación específica (e.g. si la implementación soporta algunas personalizaciones).

xml.dom.getDOMImplementation(*name=None*, *features=()*)

Retorna una implementación del DOM apropiada. El *name* es o bien conocido, el nombre del módulo de una implementación DOM, o `None`. Si no es `None` importa el módulo correspondiente y retorna un objeto **`DomImplementation`** si la importación tiene éxito. Si no se le pasa un nombre, y el entorno de variable **`PYTHON_DOM`** ha sido puesto, dicha variable es usada para encontrar la información de la implementación.

Si no se le pasa un nombre, examina las implementaciones disponibles para encontrar uno con el conjunto de características requeridas. Si no se encuentra ninguna implementación, lanza una excepción **`ImportError`**. La lista de características debe ser una secuencia de pares `(feature,version)` que son pasados al método **`hasFeature()`** en objetos disponibles de **`DOMImplementation`**.

Algunas constantes convenientes son proporcionadas:

xml.dom.EMPTY_NAMESPACE

El valor usado para indicar que ningún espacio de nombres es asociado con un nodo en el DOM. Se encuentra típicamente con el **`namespaceURI`** de un nodo, o usado como el parámetro *namespaceURI* para un método específico del *namespace*.

xml.dom.XML_NAMESPACE

El espacio de nombres de la URI asociada con el prefijo `xml`, como se define por Namespaces in XML [https://www.w3.org/TR/REC-xml-names/] (sección 4).

xml.dom.XMLNS_NAMESPACE

El espacio de nombres del URI para declaraciones del espacio de nombres, como se define en Document Object Model (DOM) Level 2 Core Specification [https://www.w3.org/TR/DOM-Level-2-Core/core.html] (sección 1.1.8).

xml.dom.XHTML_NAMESPACE

El URI del espacio de nombres del XHTML como se define en XHTML 1.0: The Extensible HyperText Markup Language [https://www.w3.org/TR/xhtml1/] (sección 3.1.1).

In addition, **`xml.dom`** contains a base **`Node`** class and the DOM exception classes. The **`Node`** class provided by this module does not implement any of the methods or attributes defined by the DOM specification; concrete DOM implementations must provide those. The **`Node`** class provided as part of this module does provide the constants used for the **`nodeType`** attribute on concrete **`Node`** objects; they are located within the class rather than at the module level to conform with the DOM specifications.

Objetos en el DOM

La documentación definitiva para el DOM es la especificación del DOM del W3C.

Note que los atributos del DOM también pueden ser manipulados como nodos en vez de simples cadenas de caracteres (*strings*). Sin embargo, es bastante raro que tengas que hacer esto, por lo que su uso aún no está documentado.

Interfaz	**Sección**	**Propósito**

DOMImplementation	Objetos DOMImplementation	Interfaz para las implementaciones subyacentes.
Node	Objetos nodo	Interfaz base para la mayoría de objetos en un documento.
NodeList	Objetos NodeList	Interfaz para una secuencia de nodos.
DocumentType	Objetos DocumentType	Información acerca de la declaraciones necesarias para procesar un documento.
Document	Objetos documento	Objeto que representa un documento entero.
Element	Objetos elemento	Nodos elemento en la jerarquía del documento.
Attr	Objetos atributo	Nodos de los valores de los atributos en los elementos nodo.
Comment	Objetos comentario	Representación de los comentarios en el documento fuente.
Text	Objetos Texto y CDATASection	Nodos con contenido textual del documento.
ProcessingInstruction	Objetos ProcessingInstruction	Representación de instrucción del procesamiento.

Una sección adicional describe las excepciones definidas para trabajar con el *DOM* en Python.

Objetos *DOMImplementation*

La interfaz **`DOMImplementation`** proporciona una forma para que las aplicaciones determinen la disponibilidad de características particulares en el *DOM* que están usando. El *DOM* nivel 2 añadió la habilidad de crear nuevos objetos **`Document`** y **`DocumentType`** usando **`DOMImplementation`** también.

DOMImplementation.hasFeature(*feature*, *version*)

> Retorna `True` si la característica identificada por el par de cadenas de caracteres *feature* y *version* está implementada.

DOMImplementation.createDocument(*namespaceUri*, *qualifiedName*, *doctype*)

> Retorna un nuevo objeto **`Document`** (la raíz del DOM), con un hijo objeto **`Element`** teniendo el *namespaceUri* y *qualifiedName* dados. El *doctype* debe ser un **`DocumentType`** creado por **`createDocumentType()`** o `None`. En la *DOM API* de Python, los primeros argumentos pueden ser `None` para indicar que ningún hijo **`Element`** va a ser creado.

DOMImplementation.createDocumentType(*qualifiedName*, *publicId*, *systemId*)

Retorna un nuevo objeto **`DocumentType`** que encapsula las cadenas de caracteres *qualifiedName*, *publicId*, y *systemId* dadas, representando la información contenida en un tipo de declaración de documento XML.

Objetos nodo

Todos los componentes de un documento XML son sub-clases de **`Node`**.

Node.nodeType

Un entero representando el tipo de nodo. Las Constantes simbólicas para los tipos están en el objeto **`Nodo`**: **`ELEMENT_NODE`**, **`ATTRIBUTE_NODE`**, **`TEXT_NODE`**, **`CDATA_SECTION_NODE`**, **`ENTITY_NODE`**, **`PROCESSING_INSTRUCTION_NODE`**, **`COMMENT_NODE`**, **`DOCUMENT_NODE`**, **`DOCUMENT_TYPE_NODE`**, **`NOTATION_NODE`**. Este es un atributo sólo de lectura.

Node.parentNode

El padre del nodo actual, o `None` para el nodo del documento. El valor es siempre un objeto **`Node`** o `None`. Para los nodos **`Element`**, este será el elemento padre, excepto para el elemento raíz, en cuyo caso será el objeto **`Document`**. Para los nodos **`Attr`**, este siempre es `None`. Este es un atributo de sólo lectura.

Node.attributes

Un **`NamedNodeMap`** de objetos de atributos. Sólo los elementos tienes un valor real para esto; otros nodos proporcionan `None` para este atributo. Este es un atributo de sólo lectura.

Node.previousSibling

El nodo que precede inmediatamente este nodo con el mismo padre. Por ejemplo el elemento con una etiqueta final que viene justo antes de la etiqueta del comienzo del elemento *self*. Por supuesto, los documentos XML está hechos de más que sólo elementos por lo que el hermano anterior puede ser un texto, un comentario, o algo más. Si este nodo es el primer hijo del padre, este atributo será `None`. Este es un atributo de sólo lectura.

Node.nextSibling

El nodo que sigue inmediatamente este nodo con el mismo padre. Véase también **`previousSibling`**. Si este es el último hijo del padre, este atributo será `None`. Este es un atributo de sólo lectura.

Node.childNodes

Una lista de nodos contenidos dentro de este nodo. Este es un atributo de sólo lectura.

Node.firstChild

El primer hijo del nodo, si hay alguno, o `None`. Este es un atributo de sólo lectura.

Node.lastChild

El último hijo del nodo, si hay alguno, o `None`. Este es un atributo de sólo lectura.

Node.localName

La parte del **`tagName`** seguido de los dos puntos si hay uno, si no, el **`tagName`** entero. El valor es una cadena de caracteres.

Node.prefix

La parte del **`tagName`** antes de los dos puntos si hay uno, si no, la cadena de caracteres vacía. El valor es una cadena, o `None`.

Node.namespaceURI

El espacio de nombres asociado con el nombre del elemento. Este será una cadena de caracteres o `None`. Este es una atributo de sólo lectura.

Node.nodeName

Este tiene un significado diferente para cada tipo de nodo; véase la especificación del DOM para los detalles. Siempre puedes obtener la información que obtendrías aquí desde otra propiedad como la propiedad **`tagName`** para elementos o la propiedad **`name`** para atributos. Para todos los tipos de nodo, el valor de este atributo tendrá o una cadena de caracteres o `None`. Este es un atributo de sólo lectura.

Node.nodeValue

Este tiene un significado diferente para cada tipo de nodo; véase la especificación del DOM para los detalles. La situación es similar a ese con **`nodeName`**. El valor es una cadena de caracteres o `None`.

Node.hasAttributes()

Retorna `True` si el nodo tiene algún atributo.

Node.hasChildNodes()

Retorna `True` si el nodo tiene algún nodo hijo.

Node.isSameNode(*other*)

Retorna `True` si *other* hace referencia al mismo nodo como este nodo. Esto es especialmente útil para las implementaciones DOM que usan una arquitectura *proxy* de cualquier tipo (porque más de un objeto puede hacer referencia al mismo nodo).

Nota

Esto se basa en una *DOM API* de nivel 3 propuesta que está en la etapa de «borrador de trabajo» (*«working draft»*), pero esta interfaz en particular no parece controversial. Los cambios del *W3C* necesariamente no afectaran este método en la interfaz del *DOM* del Python (aunque cualquier nueva *API* del *W3C* para esto también sería soportado).

Node.appendChild(*newChild*)

Añade un nuevo nodo hijo a este nodo al final de la lista de hijos, retornando *newChild*. Si el nodo ya estaba en el árbol, este se remueve primero.

Node.insertBefore(*newChild*, *refChild*)

Inserta un nuevo nodo hijo antes de un hijo existente. Debe ser el caso que *refChild* sea un hijo de este nodo; si no, **`ValueError`** es lanzado. *newChild* es retornado. Si *refChild* es `None`, se inserta a *newChild* al final de la lista de hijos.

Node.removeChild(*oldChild*)

Elimina un nodo hijo. *oldChild* debe ser un hijo de este nodo; si no, **`ValueError`** es lanzado. *oldChild* es retornado si tiene éxito. Si *oldChild* no será usado más adelante, se debe llamar a su método **`unlink()`**.

Node.replaceChild(*newChild*, *oldChild*)

Reemplaza un nodo existente con un nuevo nodo. Debe ser el caso que *oldChild* sea un hijo de este nodo; si no, **`ValueError`** es lanzado.

Node.normalize()

Une nodos de texto adyacentes para que todos los tramos de texto sean guardados como únicas instancias de **`Text`**. Esto simplifica el procesamiento de texto de un árbol del *DOM* para muchas aplicaciones.

Node.cloneNode(*deep*)

Clona este nodo. Poner *deep* significa clonar todos los nodos hijo también. Esto retorna el clon.

Objetos *NodeList*

Un **`NodeList`** representa una secuencia de nodos. Estos objetos se usan de dos formas en la recomendación principal del *DOM*: un objeto **`Element`** proporciona uno como su lista de nodos hijo, y los métodos **`getElementsByTagName()`** y **`getElementsByTagNameNS()`** de **`Node`** retornan objetos con esta interfaz para representar resultados de consulta.

La recomendación del *DOM* nivel 2 define un método y un atributo para estos objetos:

NodeList.item(*i*)

> Retorna el i-ésimo *item* de la secuencia, si hay uno, o `None`. El índice *i* no puede ser menor que cero o mayor o igual que el tamaño de la secuencia.

NodeList.length

> El número de nodos en la secuencia.

Además, la interfaz *DOM* de Python requiere que un algún soporte adicional sea proporcionado para que los objetos **`NodeList`** puedan ser usados como secuencias de Python. Todas las implementaciones de **`NodeList`** deben incluir soporte para **`__len__()`** y **`__getitem__()`**; esto permite la iteración de **`NodeList`** en sentencias con **`for`** y un soporte apropiado para la función incorporada **`len()`**.

Si una implementación DOM soporta la modificación del documento, la implementación de **`NodeList`** debe también soportar los métodos **`__setitem__()`** y **`__delitem__()`**.

Objetos *DocumentType*

La información acerca de las notaciones y entidades declaradas por un documento (incluido el subconjunto externo si el analizador lo usa y puede proporcionar información) está disponible desde un objeto **`DocumentType`**. El **`DocumentType`** para un documento está disponible desde el atributo **`doctype`** del objeto **`Document`**; si no hay ninguna declaración `DOCTYPE` para el documento, el atributo **`doctype`** del documento se pondrá como `None` en vez de una instancia de esta interfaz.

`DocumentType` es una especialización de **`Node`**, y añade los siguientes atributos:

DocumentType.publicId

El identificador público para el subconjunto externo de la definición del tipo de documento. Esto será una cadena de caracteres o `None`.

DocumentType.systemId

El identificador del sistema para el subconjunto externo de la definición del tipo de documento. Esto será una *URI* como una cadena de caracteres, o `None`.

DocumentType.internalSubset

Una cadena de caracteres proporcionando el subconjunto interno completo del documento. Esto no incluye los paréntesis que cierran el subconjunto. Si el documento no tiene ningún subconjunto interno, debe ser `None`.

DocumentType.name

El nombre del elemento raíz como se indica en la declaración `DOCTYPE`, si está presente.

DocumentType.entities

Este es un **`NamedNodeMap`** proporcionando las definiciones de las entidades externas. Para nombres de entidades definidas más de una vez, sólo la primera definición es proporcionada (el resto es ignorado como se requiere por la recomendación de *XML*). Puede ser `None` si el analizador no proporciona la información, o si ninguna entidad es definida.

DocumentType.notations

Este es un **`NamedNodeMap`** proporcionando las definiciones de las notaciones. Para nombres de notaciones definidas más de una vez, sólo la primera definición es proporcionada (el resto es ignorado como se requiere por la recomendación *XML*). Puede ser `None` si el analizador no proporciona la información, o si no hay notaciones definidas.

Objetos documento

Un **`Documento`** representa un documento *XML* entero, incluyendo sus elementos constituyentes, atributos, instrucciones de procesamiento, comentarios, etc. Recuerda que este hereda propiedades de **`Node`**.

Document.documentElement

El único elemento raíz del documento.

Document.createElement(*tagName*)

Crea y retorna un nuevo elemento nodo. El elemento no se inserta en el documento cuando es creado. Necesitas insertarlo explícitamente con uno de los otros métodos como **`insertBefore()`** o **`appendChild()`**.

Document.createElementNS(*namespaceURI*, *tagName*)

Crea y retorna un nuevo elemento con un espacio de nombres. El *tagName* puede tener un prefijo. El elemento no se inserta en el documento cuando es creado. Necesitas insertarlo explícitamente con uno de los otros métodos como **`insertBefore()`** o **`appendChild()`**.

Document.createTextNode(*data*)

Crea y retorna un nodo texto conteniendo los datos pasados como parámetros, Como con los otros métodos de creación, este no inserta el nodo en el árbol.

Document.createComment(*data*)

Crea y retorna un nodo comentario conteniendo los datos pasados como parámetros. Como con los otros métodos de creación, este no inserta el nodo en el árbol.

Document.createProcessingInstruction(*target*, *data*)

Crea y retorna una instrucción de procesamiento conteniendo el *target* y *data* pasados como parámetros. Como con los otros métodos de creación, este no inserta en nodo en el árbol.

Document.createAttribute(*name*)

Crea y retorna un nodo atributo. Este método no asocia el nodo atributo con ningún elemento particular. Debes usar **`setAttributeNode()`** en el objeto **`Element`** apropiado para usar la instancia del atributo recién creada.

Document.createAttributeNS(*namespaceURI*, *qualifiedName*)

Crea y retorna un nodo atributo con un espacio de nombres. El *tagName* puede ser un prefijo. Este método no asocia el nodo atributo con ningún elemento en particular. Debes usar **`setAttributeNode()`** en el objeto **`Element`** apropiado para usar la instancia del atributo recién creada.

Document.getElementsByTagName(*tagName*)

Busca todos los descendientes (hijos directos, hijos de los hijos, etc.) con un nombre del tipo de elemento particular.

Document.getElementsByTagNameNS(*namespaceURI*, *localName*)

Busca todos los descendientes (hijos directos, hijos de hijos, etc.) con un espacio de nombres URI particular (*namespaceURI*) y nombre local (*localname*). El nombre local es parte del espacio de nombres después del prefijo.

Objetos elemento

`Element` es una subclase de **`Node`**, por lo que hereda todos los atributos de esa clase.

Element.tagName

El nombre del tipo de elemento. En un documento que usa espacios de nombres este puede tener varios dos puntos en el. El valor es una cadena de caracteres.

Element.getElementsByTagName(*tagName*)

Igual al método equivalente en la clase **`Document`**.

Element.getElementsByTagNameNS(*namespaceURI*, *localName*)

Igual al método equivalente en la clase **`Document`**.

Element.hasAttribute(*name*)

Retorna `True` si el elemento tiene un atributo nombrado *name*.

Element.hasAttributeNS(*namespaceURI, localName*)

Retorna `True` si el elemento tiene un atributo nombrado por *namespaceURI* y *localName*.

Element.getAttribute(*name*)

Retorna el valor del atributo nombrado por *name* como una cadena de caracteres. Si no existe dicho atributo, una cadena vacía es retornada, como si el atributo no tuviera valor.

Element.getAttributeNode(*attrname*)

Retorna el nodo **`Attr`** para el atributo nombrado por *attrname*.

Element.getAttributeNS(*namespaceURI*, *localName*)

Retorna el valor del atributo nombrado por *namespaceURI* y *localName* como una cadena de caracteres. Si no existe dicho atributo, una cadena vacía es retornada, como si el atributo no tuviera valor.

Element.getAttributeNodeNS(*namespaceURI, localName*)

Retorna un valor de atributo como nodo, dado un *namespaceURI* y *localName*.

Element.removeAttribute(*name*)

Remueve un atributo por nombre (*name*). Si no hay un atributo correspondiente, un **`NotFoundErr`** es lanzado.

Element.removeAttributeNode(*oldAttr*)

Remueve y retorna *oldAttr* de la lista de atributos, si está presenta. Si *oldAttr* no está presente, **`NotFoundErr`** es lanzado.

Element.removeAttributeNS(*namespaceURI*, *localName*)

Remueve un atributo por nombre (*name*). Note que esto usa un *localName*, no un *qname*. Ninguna excepción es lanzada si no existe el atributo correspondiente.

Element.setAttribute(*name*, *value*)

Pone un valor de atributo como una cadena de caracteres.

Element.setAttributeNode(*newAttr*)

Añade un nuevo atributo nodo al elemento, reemplazando un atributo existente si es necesario si el atributo **`name`** coincide. Si un reemplazo ocurre, el viejo atributo será retornado. Si *newAttr* ya está en uso, **`InuseAttributeErr`** será lanzado.

Element.setAttributeNodeNS(*newAttr*)

Añade un nuevo nodo atributo al elemento, reemplazando un atributo existente si es necesario, si el **`namesapceURI`** y **`localName`** coinciden. Si un reemplazo ocurre, el viejo atributo será retornado. Si *newAttr* está en uso, **`InuseAttributeErr`** será lanzado.

Element.setAttributeNS(*namespaceURI*, *qname*, *value*)

Pone un valor de atributo a partir de una cadena de caracteres, dados un *namespaceURI* y *qname*. Note que un *qname* es el nombre completo del atributo. Esto es diferente al de arriba.

Objetos atributo

`Attr` hereda de **`Node`**, por lo que hereda todos sus atributos.

Attr.name

El nombre del atributo. En un documento que usa espacio de nombres, puede incluir dos puntos.

Attr.localName

La parte del nombre seguido después de los dos puntos si hay uno, si no el nombre entero. Este es un atributo de sólo lectura.

Attr.prefix

La parte del nombre que precede los dos puntos si hay uno, si no la cadena de caracteres vacía.

Attr.value

El valor textual del atributo. Este es un sinónimo para el atributo `nodeValue`.

Objetos *NamedNodeMap*

`NamedNodeMap` *no* hereda de `Node`.

NamedNodeMap.length

La longitud de la lista de atributos.

NamedNodeMap.item(*index*)

Retorna un atributo con un índice particular. El orden en los que obtienes los atributos es arbitrario pero será consistente en la vida de un *DOM*. Cada *item* es un nodo atributo. Obtén su valor con el atributo `value`.

There are also experimental methods that give this class more mapping behavior. You can use them or you can use the standardized `getAttribute*()` family of methods on the `Element` objects.

Objetos comentario

`Comment` representa un comentario en el documento *XML*. Es una subclase de `Node`, pero no puede tener hijos nodo.

Comment.data

El contenido del comentario como una cadena de caracteres.

El atributo contiene todos los caracteres entre el `<!--` que empieza y el `-->` que termina, pero no los incluye.

Objetos Texto y *CDATASection*

La interfaz **`Text`** representa el texto en el documento *XML*. Si el analizador y la implementación del *DOM* soporta la extensión *XML* del *DOM*, las porciones de texto rodeadas secciones marcadas como *CDATA* se guardan en objetos **`CDATASection`**. Estas dos interfaces son idénticas, pero proveen valores diferentes para el atributo **`nodeType`**.

Estas interfaces extienden la interfaz **`Node`**. No pueden tener nodos hijo.

Text.data

El contenido del nodo texto como una cadena de caracteres.

Nota

El uso de un nodo **`CDATASection`** no indica que el nodo represente una sección completa marcada como *CDATA*, sólo que el contenido del nodo fue parte de una sección *CDATA*. Una sola sección *CDATA* puede ser representada por más de un nodo en el árbol del documento. No hay manera de determinar si dos nodos adyacentes **`CDATASection`** son representados diferentes a secciones marcadas como *CDATA*.

Objetos *ProcessingInstruction*

Representa una instrucción de procesamiento en el documento *XML*; hereda de la interfaz **`Node`** y no puede tener hijos.

ProcessingInstruction.target

El contenido de la instrucción de procesamiento hasta el carácter en blanco. Este es un atributo de sólo lectura.

ProcessingInstruction.data

El contenido de la instrucción de procesamiento después del primer carácter en blanco.

Excepciones

La recomendación del *DOM* nivel 2 define una sola excepción, `DOMException`, y un número de constantes que permite que las aplicaciones determinen qué tipo de error ocurrió. las instancias de `DOMException` llevan un atributo `code` que proporciona el valor apropiado para la excepción específica.

La interfaz *DOM* de Python provee las constantes, pero también expande el conjunto de excepciones para que exista una excepción específica para cada uno de los códigos de excepción definidos por el *DOM*. Las implementaciones deben lanzar la excepción específica apropiada, cada uno de los cuales lleva el valor apropiado para el atributo `code`.

exception xml.dom.DOMException

Clase base de excepción usada para todas las excepciones del *DOM* específicas. Esta clase de excepción no puede ser instanciada directamente.

exception xml.dom.DomstringSizeErr

Lanzado cuando un rango de texto específico no cabe en una cadena de caracteres. No se sabe si se usa in las implementación *DOM* de Python, pero puede ser recibido de otras implementaciones *DOM* que no hayan sido escritas en Python.

exception xml.dom.HierarchyRequestErr

Lanzado cuando se intenta insertar un nodo donde el tipo de nodo no es permitido.

exception xml.dom.IndexSizeErr

Lanzado cuando un parámetro del índice o tamaño de un método es negativo o excede los valores permitidos.

exception xml.dom.InuseAttributeErr

Lanzado cuando se intenta insertar un nodo **`Attr`** que está presente en algún lado en el documento.

exception xml.dom.InvalidAccessErr

Lanzado si un parámetro o una operación no es soportada por el objeto subyacente.

exception xml.dom.InvalidCharacterErr

Esta excepción es lanzada cuando un parámetro de cadena de caracteres contiene un carácter que no está permitido en el contexto que está siendo usado por la recomendación *XML 1.0*. Por ejemplo, intentar crear un nodo **`Element`** con un espacio en el nombre del tipo de elemento causará que se lance este error.

exception xml.dom.InvalidModificationErr

Lanzado cuando se intenta modificar el tipo de un nodo.

exception xml.dom.InvalidStateErr

Lanzado cuando se intenta usar un objeto que no está definido o ya no es usable.

exception xml.dom.NamespaceErr

Si se intenta cambiar cualquier objeto de forma que no sea permitida con respecto a la recomendación Namespaces in XML [https://www.w3.org/TR/REC-xml-names/], esta excepción es lanzada.

exception xml.dom.NotFoundErr

Excepción cuando un nodo no existe en el contexto referenciado. Por ejemplo, **`NamedNodeMap.removeNamedItem()`** será lanzado si el nodo pasado no existe en el mapa.

exception xml.dom.NotSupportedErr

Lanzado cuando la implementación no soporta el tipo requerido del objeto u operación.

exception xml.dom.NoDataAllowedErr

Es lanzado si se especifican datos para un nodo que no soporta datos.

exception xml.dom.NoModificationAllowedErr

Lanzado cuando se intenta modificar un objeto donde las modificaciones no son permitidas (tal como los nodos de sólo-lectura).

exception xml.dom.SyntaxErr

Lanzado cuando se especifica una cadena de caracteres inválida o ilegal.

exception xml.dom.WrongDocumentErr

Lanzado cuando un nodo es insertado en un documento diferente al que este actualmente pertenece, y la implementación no soporta migrar el nodo de un documento a otro.

Los códigos de excepción definidos en la recomendación del *DOM* se mapean a las excepciones descritas arriba de acuerdo a esta tabla:

Constante	**Excepción**
`DOMSTRING_SIZE_ERR`	`DomstringSizeErr`
`HIERARCHY_REQUEST_ERR`	`HierarchyRequestErr`
`INDEX_SIZE_ERR`	`IndexSizeErr`
`INUSE_ATTRIBUTE_ERR`	`InuseAttributeErr`
`INVALID_ACCESS_ERR`	`InvalidAccessErr`
`INVALID_CHARACTER_ERR`	`InvalidCharacterErr`
`INVALID_MODIFICATION_ERR`	`InvalidModificationErr`
`INVALID_STATE_ERR`	`InvalidStateErr`
`NAMESPACE_ERR`	`NamespaceErr`
`NOT_FOUND_ERR`	`NotFoundErr`
`NOT_SUPPORTED_ERR`	`NotSupportedErr`
`NO_DATA_ALLOWED_ERR`	`NoDataAllowedErr`
`NO_MODIFICATION_ALLOWED_ERR`	`NoModificationAllowedErr`
`SYNTAX_ERR`	`SyntaxErr`
`WRONG_DOCUMENT_ERR`	`WrongDocumentErr`

Conformidad

Esta sección describe los requisitos de conformidad y las relaciones entre el *DOM API* de Python, las recomendaciones del *DOM* del *W3C*, y el mapeo *OMG IDL* para Python.

Mapeo de tipos

Los tipos IDL usados en la especificación del *DOM* son mapeados a los tipos de tipos de Python de acuerdo a la siguiente tabla.

Tipo IDL	Tipo en Python
`boolean`	`bool o int`
`int`	`int`
`long int`	`int`
`unsigned int`	`int`
`DOMString`	`str o bytes`
`null`	`None`

Métodos de acceso (*accessor*)

El mapeo de *OMG IDL* a python define funciones de acceso para las declaraciones del atributo *IDL* de la que misma forma en que el mapeo de Java lo hace. Mapear las declaraciones IDL:

```
readonly attribute string someValue;
         attribute string anotherValue;
```

produce tres funciones de acceso: un método «get» para **`someValue`** (**`_get_someValue()`**), y métodos «get» y «set» para **`anotherValue`** (**`_get_anotherValue()`** y **`_set_anotherValue()`**). El mapeado, en particular, no requiere que los atributos *IDL* sean accesibles como los atributos normales de Python: No es obligatorio que `object.someValue` funcione, y puede lanzar un **`AttributeError`**.

El *DOM API* de Python, sin embargo, *si* requiere que los atributos de acceso normales funcionen. Esto significa que no es probable que los típicos sustitutos generados por compiladores de *IDL* en Python funcionen, y los objetos envoltorio (*wrapper*) pueden ser necesarios en el cliente si los objetos del *DOM* son accedidos mediante *CORBA*. Mientras que esto requiere consideraciones adicionales para clientes *DOM* en *CORBA*, los implementadores con experiencia que usen *DOM* por encima de *CORBA* desde Python no lo consideran un problema. Los atributos que se declaran `readonly` pueden no restringir el acceso de escritura en todas las implementaciones DOM.

En el *DOM API* de Python, las funciones de acceso no son obligatorias. Si se proveen, deben tomar la forma definida por el mapeo *IDL* de Python, pero estos métodos se consideran innecesarios debido a que los atributos son accesibles directamente desde Python. Nunca se deben proporcionar métodos de acceso (*accessor*) «Set» para los atributos `readonly`.

Las definiciones de IDL no encarnan los requisitos del *DOM API* del *W3C* por completo, como las nociones de ciertos objetos, como el valor de retorno **`getElementsByTagName()`**, siendo «live». El DOM API de Python no requiere que las implementaciones hagan cumplir tales requisitos.

[1]

`xml.dom.minidom` — Minimal DOM implementation

Código fuente: Lib/xml/dom/minidom.py [https://github.com/python/cpython/tree/3.14/Lib/xml/dom/minidom.py]

`xml.dom.minidom` is a minimal implementation of the Document Object Model interface, with an API similar to that in other languages. It is intended to be simpler than the full DOM and also significantly smaller. Users who are not already proficient with the DOM should consider using the **`xml.etree.ElementTree`** module for their XML processing instead.

Nota

If you need to parse untrusted or unauthenticated data, see XML security.

DOM applications typically start by parsing some XML into a DOM. With **`xml.dom.minidom`**, this is done through the parse functions:

```
from xml.dom.minidom import parse, parseString

dom1 = parse('c:\\temp\\mydata.xml')  # parse an XML fil

datasource = open('c:\\temp\\mydata.xml')
dom2 = parse(datasource)  # parse an open file

dom3 = parseString('<myxml>Some data<empty/> some more
```

La función **`parse()`** puede tomar un nombre de archivo o un objeto de archivo previamente abierto.

xml.dom.minidom.parse(*filename_or_file*, *parser=None*,

bufsize=None)

Retorna un **`Document`** a partir de la entrada dada. *filename_or_file* puede ser un nombre de archivo o un objeto similar a un archivo. *parser*, si se proporciona, debe ser un objeto de un analizador sintáctico SAX2. Esta función intercambiará el controlador de documentos del analizador sintáctico y activará el soporte con el espacio de nombres. Otras configuraciones del analizador sintáctico (como configurar un solucionador de entidades) deben haberse realizado de antemano.

Si tienes XML en una cadena de caracteres, puedes usar la función `parseString()` en su lugar:

xml.dom.minidom.parseString(*string*, *parser=None*)

Retorna un objeto **`Document`** que representa a *string*. Este método crea un objeto `io.StringIO` para la cadena de caracteres y lo pasa a `parse()`.

Ambas funciones retornan un objeto **`Document`** que representa el contenido del documento.

What the `parse()` and `parseString()` functions do is connect an XML parser with a «DOM builder» that can accept parse events from any SAX parser and convert them into a DOM tree. The names of the functions are perhaps misleading, but are easy to grasp when learning the interfaces. The parsing of the document will be completed before these functions return; it's simply that these functions do not provide a parser implementation themselves.

You can also create a **`Document`** by calling a method on a «DOM Implementation» object. You can get this object either by calling the **`getDOMImplementation()`** function in the `xml.dom` package or the **`xml.dom.minidom`** module. Once you have a **`Document`**, you can add child nodes to it to populate the DOM:

```
from xml.dom.minidom import getDOMImplementation

impl = getDOMImplementation()
```

```
newdoc = impl.createDocument(None, "some_tag", None)
top_element = newdoc.documentElement
text = newdoc.createTextNode('Some textual content.')
top_element.appendChild(text)
```

Una vez que tengas un objeto del documento DOM, puedes acceder a las partes de tu documento XML a través de sus propiedades y métodos. Estas propiedades se definen en la especificación DOM. La propiedad principal del objeto del documento es **`documentElement`**. Te proporciona el elemento principal en el documento XML: el que contiene a todos los demás. Aquí hay un programa de ejemplo:

```
dom3 = parseString("<myxml>Some data</myxml>")
assert dom3.documentElement.tagName == "myxml"
```

When you are finished with a DOM tree, you may optionally call the **`unlink()`** method to encourage early cleanup of the now-unneeded objects. **`unlink()`** is an **`xml.dom.minidom`**-specific extension to the DOM API that renders the node and its descendants essentially useless. Otherwise, Python's garbage collector will eventually take care of the objects in the tree.

Ver también

Document Object Model (DOM) Level 1 Specification [https://www.w3.org/TR/REC-DOM-Level-1/]
The W3C recommendation for the DOM supported by **`xml.dom.minidom`**.

Objetos del DOM

The definition of the DOM API for Python is given as part of the **`xml.dom`** module documentation. This section lists the differences between the API and **`xml.dom.minidom`**.

Node.unlink()

Rompe las referencias internas dentro del DOM para recolectarlo como basura en las versiones de Python sin recolector de basura cíclico. Incluso cuando se dispone del mismo, su uso puede hacer que grandes cantidades de memoria estén disponibles antes, por lo que es una buena práctica invocar este método en objetos DOM, tan pronto como ya no se necesiten. Solo necesita ser invocado en el objeto `Document`, pero se puede llamar en los nodos hijos para descartar los hijos de ese nodo concreto.

Puedes evitar invocar este método explícitamente utilizando la declaración `with`. El siguiente código desvinculará automáticamente *dom* cuando se salga del bloque `with`:

```
with xml.dom.minidom.parse(datasource) as dom:
    ... # Work with dom.
```

Node.writexml(*writer*, *indent=''*, *addindent=''*, *newl=''*, *encoding=None*, *standalone=None*)

Escribe XML en el objeto escritor. El escritor recibe texto pero no bytes como entrada, debe tener un método `write()` que coincida con el de la interfaz del objeto de archivo. El parámetro *indent* es la sangría del nodo actual. El parámetro *addindent* es la sangría incremental que se utilizará para los subnodos del nodo actual. El parámetro *newl* especifica la cadena que se utilizará como terminación de las nuevas líneas.

Para el nodo `Document`, se puede usar el argumento por palabra clave adicional *encoding* para especificar el valor del campo de codificación del encabezado XML.

De manera similar, al indicar explícitamente el argumento *standalone*, las declaraciones del documento independiente se agregan al prólogo del documento XML. Si el valor se establece en `True`, se agrega `standalone="yes"`; de lo contrario, se establece en `"no"`. No declarar el argumento omitirá la declaración del documento.

Distinto en la versión 3.8: El método `writexml()` ahora conserva el orden de los atributos especificado por el usuario.

Distinto en la versión 3.9: Se agregó el parámetro *standalone*.

Node.toxml(*encoding=None*, *standalone=None*)

Retorna una cadena de caracteres o una cadena de bytes que contiene el XML representado por el nodo DOM.

Si se proporciona de forma explícita un valor para el argumento *encoding* [1], el resultado es una cadena de bytes con la codificación especificada. Si no se proporciona el argumento *encoding*, el resultado es una cadena Unicode y la declaración XML en la cadena resultante no especifica una codificación. Codificar esta cadena en una codificación que no sea UTF-8 probablemente sea una práctica incorrecta, ya que UTF-8 es la codificación predeterminada para XML.

El argumento *standalone* se comporta exactamente como en **`writexml()`**.

Distinto en la versión 3.8: El método **`toxml()`** ahora conserva el orden de los atributos especificado por el usuario.

Distinto en la versión 3.9: Se agregó el parámetro *standalone*.

Node.toprettyxml(*indent='\t'*, *newl='\n'*, *encoding=None*, *standalone=None*)

Retorna una versión impresa elegante del documento. *indent* especifica la cadena de caracteres a usar como sangría y es una tabulación por defecto; *newl* especifica la cadena de caracteres emitida al final de cada línea y es `\n` por defecto.

El argumento *encoding* se comporta como el argumento correspondiente del método **`toxml()`**.

El argumento *standalone* se comporta exactamente como en **`writexml()`**.

Distinto en la versión 3.8: El método **`toprettyxml()`** ahora conserva el orden de los atributos especificado por el usuario.

Distinto en la versión 3.9: Se agregó el parámetro *standalone*.

Ejemplo de DOM

Este programa de ejemplo es una demostración bastante realista de un programa simple. En este caso particular, no aprovechamos mucho la flexibilidad del DOM.

```
import xml.dom.minidom

document = """\
<slideshow>
<title>Demo slideshow</title>
<slide><title>Slide title</title>
<point>This is a demo</point>
<point>Of a program for processing slides</point>
</slide>

<slide><title>Another demo slide</title>
<point>It is important</point>
<point>To have more than</point>
<point>one slide</point>
</slide>
</slideshow>
"""

dom = xml.dom.minidom.parseString(document)

def getText(nodelist):
    rc = []
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            rc.append(node.data)
    return ''.join(rc)

def handleSlideshow(slideshow):
    print("<html>")
    handleSlideshowTitle(slideshow.getElementsByTagName(
    slides = slideshow.getElementsByTagName("slide")
    handleToc(slides)
    handleSlides(slides)
    print("</html>")

def handleSlides(slides):
    for slide in slides:
        handleSlide(slide)

def handleSlide(slide):
    handleSlideTitle(slide.getElementsByTagName("title")
    handlePoints(slide.getElementsByTagName("point"))

def handleSlideshowTitle(title):
    print(f"<title>{getText(title.childNodes)}</title>")

def handleSlideTitle(title):
    print(f"<h2>{getText(title.childNodes)}</h2>")

def handlePoints(points):
    print("<ul>")
    for point in points:
        handlePoint(point)
    print("</ul>")

def handlePoint(point):
    print(f"<li>{getText(point.childNodes)}</li>")

def handleToc(slides):
    for slide in slides:
        title = slide.getElementsByTagName("title")[0]
        print(f"<p>{getText(title.childNodes)}</p>")

handleSlideshow(dom)
```

minidom y el estándar DOM

The `xml.dom.minidom` module is essentially a DOM 1.0-compatible DOM with some DOM 2 features (primarily namespace features).

El uso de la interfaz DOM en Python es sencillo. Se aplican las siguientes reglas de mapeo:

- Se accede a las interfaces a través de objetos de instancia. Las aplicaciones no deben instanciar las clases en sí mismas; deben usar las funciones de creación disponibles en el objeto **`Document`**. Las interfaces derivadas admiten todas las operaciones (y atributos) de las interfaces base, además de cualquier operación nueva.
- Las operaciones se utilizan como métodos. Dado que el DOM usa solo parámetros **`in`**, los argumentos se pasan en el orden normal (de izquierda a derecha). No hay argumentos opcionales. Las operaciones `void` retornan `None`.
- Los atributos IDL se asignan a atributos de instancia. Por compatibilidad con el mapeo del lenguaje OMG IDL para Python, también se puede acceder a un atributo `foo` a través de los métodos de acceso **`_get_foo()`** y **`_set_foo()`**. Los atributos `readonly` no deben modificarse; esto no se aplica en tiempo de ejecución.
- Los tipos `short int`, `unsigned int`, `unsigned long long` y `boolean` se asignan todos a objetos enteros de Python.
- The type `DOMString` maps to Python strings. **`xml.dom.minidom`** supports either bytes or strings, but will normally produce strings. Values of type `DOMString` may also be `None` where allowed to have the IDL `null` value by the DOM specification from the W3C.
- Las declaraciones `const` se asignan a variables en su ámbito respectivo (por ejemplo, `xml.dom.minidom.Node.PROCESSING_INSTRUCTION_NODE`); no deben modificarse.
- `DOMException` is currently not supported in **`xml.dom.minidom`**. Instead, **`xml.dom.minidom`** uses standard Python exceptions such as **`TypeError`** and **`AttributeError`**.
- Los objetos de la clase **`NodeList`** se implementan usando el tipo lista incorporado de Python. Estos objetos proporcionan la interfaz definida en la especificación DOM, pero en versiones anteriores de Python no son compatibles con la API oficial. Sin embargo, son mucho más «pythónicas» que la

interfaz definida en las recomendaciones del W3C.

The following interfaces have no implementation in **`xml.dom.minidom`**:

- **`DOMTimeStamp`**
- **`EntityReference`**

La mayoría de ellas reflejan información en el documento XML que generalmente no es de utilidad para la mayoría de los usuarios de DOM.

Notas al pie

El nombre de codificación incluido en la salida XML debe cumplir con los estándares apropiados. Por ejemplo, «UTF-8» es válido, pero «UTF8» no es válido en la declaración de un documento XML, aunque Python lo acepta como nombre de codificación. Para más detalles, consulta https://www.w3.org/TR/2006/REC-xml11-20060816/#NT-EncodingDecl y https://www.iana.org/assignments/character-sets/character-sets.xhtml.

`xml.dom.pulldom` — Support for building partial DOM trees

Source code: Lib/xml/dom/pulldom.py [https://github.com/python/cpython/tree/3.14/Lib/xml/dom/pulldom.py]

The **`xml.dom.pulldom`** module provides a «pull parser» which can also be asked to produce DOM-accessible fragments of the document where necessary. The basic concept involves pulling «events» from a stream of incoming XML and processing them. In contrast to SAX which also employs an event-driven processing model together with callbacks, the user of a pull parser is responsible for explicitly pulling events from the stream, looping over those events until either processing is finished or an error condition occurs.

Nota

If you need to parse untrusted or unauthenticated data, see XML security.

Distinto en la versión 3.7.1: El analizador SAX ya no procesa entidades externas generales de forma predeterminada para aumentar la seguridad de forma predeterminada. Para habilitar el procesamiento de entidades externas, pase una instancia de analizador personalizada (*custom parser instance in::*)

```
from xml.dom.pulldom import parse
from xml.sax import make_parser
from xml.sax.handler import feature_external_ges

parser = make_parser()
parser.setFeature(feature_external_ges, True)
parse(filename, parser=parser)
```

Ejemplo:

```
from xml.dom import pulldom

doc = pulldom.parse('sales_items.xml')
for event, node in doc:
    if event == pulldom.START_ELEMENT and node.tagName =
        if int(node.getAttribute('price')) > 50:
            doc.expandNode(node)
            print(node.toxml())
```

`event` es una constante y puede ser uno de:

- **START_ELEMENT** (Iniciar elemento)
- **END_ELEMENT** (Finalizar elemento)
- **COMMENT** (comentario)
- **START_DOCUMENT** (Iniciar documento)
- **END_DOCUMENT** (finalizar documento)
- **CHARACTERS** (caracteres)
- **PROCESSING_INSTRUCTION** (instrucción de procesamiento)
- **IGNORABLE_WHITESPACE** (Espacio en blanco que puede ignorarse)

`node` es un objeto del tipo **xml.dom.minidom.Document**, **xml.dom.minidom.Element** ó **xml.dom.minidom.Text**.

Puesto que el documento se trata como una secuencia «flat» (plana) de eventos, el documento «tree» (árbol) se atraviesa implícitamente y los elementos deseados se encuentran independientemente de su profundidad en el árbol. En otras palabras, no es necesario tener en cuenta cuestiones jerárquicas como la búsqueda recursiva de los nodos de documento, aunque si el contexto de los elementos fuera importante, es necesario mantener algún estado relacionado con el contexto (es decir, recordar dónde se encuentra en el documento en un momento dado) o hacer uso del método **DOMEventStream.expandNode()** y cambiar al procesamiento relacionado con DOM.

class xml.dom.pulldom.PullDOM(*documentFactory=None*)

Subclase de **xml.sax.handler.ContentHandler**.

class xml.dom.pulldom.SAX2DOM(*documentFactory=None*)

Subclase de `xml.sax.handler.ContentHandler`.

xml.dom.pulldom.parse(*stream_or_string, parser=None, bufsize=None*)

Retorna un `DOMEventStream` de la entrada dada. *stream_or_string* (secuencia o cadena) puede ser un nombre de archivo o un objeto similar a un archivo, *parser*, si se indica, debe ser un objeto `XMLReader`. Esta función cambiará el controlador de documentos del analizador y activará el soporte de espacios de nombres; otra configuración del analizador (como establecer un solucionador de entidades) debe haberse realizado de antemano.

Si tiene XML en una cadena, puede usar en su lugar la función `parseString()`:

xml.dom.pulldom.parseString(*string, parser=None*)

Retorna una: clase `DOMEventStream` que representa la cadena (Unicode) *strnig* (cadena)

xml.dom.pulldom.default_bufsize

Valor predeterminado para el parámetro *bufsize* para `parse()`.

El valor de las variables puede ser cambiado antes de llamar a `parse()` y el nuevo valor tendrá efecto.

Objetos DOMEventStream

class xml.dom.pulldom.DOMEventStream(*stream, parser, bufsize*)

Distinto en la versión 3.11: Support for `__getitem__()` method has been removed.

getEvent()

Retorna el contenido de la tupla *event* y del *node* corriente como `xml.dom.minidom.Document` si el evento es igual a **`START_DOCUMENT`**, **`xml.dom.minidom.Element`** si el evento es igual a **`START_ELEMENT`** o **`END_ELEMENT`** o **`xml.dom.minidom.Text`** si el evento es igual a **`CHARACTERS`**. El nodo actual no es contiene información sobre sus hijos a menos que se llame a la función **`expandNode()`**.

expandNode(*node*)

Expande todos los hijos de *node* en *node* (nodo en nodo). Ejemplo:

```
from xml.dom import pulldom

xml = '<html><title>Foo</title> <p>Some text <d
doc = pulldom.parseString(xml)
for event, node in doc:
    if event == pulldom.START_ELEMENT and node.
        # Following statement only prints '<p/>
        print(node.toxml())
        doc.expandNode(node)
        # Following statement prints node with
        print(node.toxml())
```

reset()

`xml.sax` — Support for SAX2 parsers

Código fuente: Lib/xml/sax/__init__.py [https://github.com/python/cpython/tree/3.14/Lib/xml/sax/__init__.py]

The `xml.sax` package provides a number of modules which implement the Simple API for XML (SAX) interface for Python. The package itself provides the SAX exceptions and the convenience functions which will be most used by users of the SAX API.

Nota

If you need to parse untrusted or unauthenticated data, see XML security.

Distinto en la versión 3.7.1: El analizador SAX ya no procesa entidades generales externas por defecto para incrementar seguridad. Antes, el analizador creaba conexiones de red para buscar archivos remotos o archivos locales cargados del sistema de archivos para DTD y entidades. La característica puede ser activadas de nuevo con el método `setFeature()` en el objeto analizador y el argumento `feature_external_ges`.

Las funciones de conveniencia son:

xml.sax.make_parser(*parser_list=[]*)

Crea y retorna un objeto SAX `XMLReader`. El primer analizador encontrado será el que se use. Si se provee *parser_list*, debe ser un iterable de cadenas de caracteres el cual nombra módulos que tienen una función llamada `créate_parser()`. Los módulos listados en *parser_list* serán usados antes de los módulos en la lista de analizadores por defecto.

Distinto en la versión 3.8: El argumento *parser_list* puede ser cualquier iterable, no sólo una lista.

xml.sax.parse(*filename_or_stream, handler, error_handler=handler.ErrorHandler()*)

Crea un analizador SAD y úsalo para analizar un documento. El documento, aprobado como *filename_or_steam*, puede ser un nombre de archivo o un objeto de archivo. El parámetro *handler* necesita ser una instancia SAX `ContentHandler`. Si se da *error_handler*, debe ser una instancia `ErrorHandler` SAX; si es omitido, se lanzará `SAXParseException` en todos los errores. No hay valor retornado; toda tarea debe ser realizada por el *handler* aprobado.

xml.sax.parseString(*string, handler, error_handler=handler.ErrorHandler()*)

Similar a `parser()`, pero analiza desde un búfer *string* recibido como un parámetro. *string* debe ser una instancia `str` o un bytes-like object.

Distinto en la versión 3.5: Agregado soporte de instancias `str`.

Una aplicación SAX típica usa tres tipos de objetos: lectores, gestores y fuentes de entrada. «Lector» en este contexto es otro término para analizador, por ejemplo, alguna pieza de código que lee los bytes o caracteres de la fuente de entrada, y produce una secuencia de eventos. Los eventos luego se distribuyen a los objetos gestores, por ejemplo el lector invoca un método en el gestor. Una aplicación SAX debe por tanto obtener un objeto lector, crear o abrir una fuente de entrada, crear los gestores, y conectar esos objetos juntos. Como paso final de preparación, el lector es llamado para analizar la entrada. Durante el análisis, los métodos en los objetos gestores son llamados basados en eventos estructurales y sintácticos de los datos introducidos.

For these objects, only the interfaces are relevant; they are normally not instantiated by the application itself. Since Python does not have an explicit notion of interface, they are formally introduced as classes, but applications may use implementations which do not inherit from the provided classes. The `InputSource`, `Locator`, **`Attributes`**, **`AttributesNS`**, and `XMLReader` interfaces are defined in the module `xml.sax.xmlreader`. The handler interfaces are defined in `xml.sax.handler`. For convenience, **`InputSource`** (which is often instantiated directly) and the handler classes are also available from **`xml.sax`**. These interfaces are described below.

In addition to these classes, **`xml.sax`** provides the following exception classes.

exception xml.sax.SAXException(*msg*, *exception=None*)

> Encapsula un error XML o advertencia. Esta clase puede contener errores básicos o información de advertencias ya sea para el analizador XML o la aplicación: esto puede ser heredado para proveer funcionalidad adicionar o para agregar localización. Nota que a pesar de los analizadores definidos en la interfaz `ErrorHandler` recibe instancias de esta excepción, no es requerido para lanzar la excepción — esto es algo útil como un contenedor para información.
>
> Cuando es instanciado, *msg* debería ser una descripción del error legible para humanos. El parámetro opcional *exception*, si es dado, debería ser `None` o una excepción que fue atrapada por el código analizador y se transmite como información.
>
> Esta es la clase base para las otras clases excepción SAX.

exception xml.sax.SAXParseException(*msg*, *exception*, *locator*)

> Subclase de `SAXException` levantada en errores de análisis. Las instancias de esta clase son pasadas a los métodos de las interfaces SAX `ErrorHandler` para proveer información sobre el error de análisis. Esta clase soporta la interfaz SAX `Locator` así como la interfaz `SAXException`.

exception xml.sax.SAXNotRecognizedException(*msg*, *exception=None*)

> Subclase de `SAXException` lanzada cuando una SAX

`XMLReader` es confrontada con una propiedad o característica no reconocida. Las aplicaciones SAX y extensiones pueden usar esta clase para propósitos similares.

exception xml.sax.SAXNotSupportedException(*msg*, *exception=None*)

Las subclases de `SAXException` se lanzan cuando un SAX `sax` se pregunta para habilitar una característica que no tiene soporte, o para establecer una propiedad a un valor que la implementación no da soporte. Las aplicaciones sAX y las extensiones pueden usar esta clase para propósitos similares.

Ver también

SAX: The Simple API for XML [http://www.saxproject.org/]
Este sitio es el punto focal para la definición de la API SAX. Provee una implementación Java y documentación en línea. Los enlaces para implementaciones e información histórica también están disponibles.

Módulo `xml.sax.handler`
Definiciones de las interfaces para objetos proporcionados por aplicaciones.

Módulo `xml.sax.saxutils`
Funciones de conveniencia para usar en aplicaciones SAX.

Módulo `xml.sax.xmlreader`
Definiciones de las interfaces para objetos que proveen analizadores.

Objetos SAXException

La clase de excepción `SAXException` da soporte a los siguientes métodos:

SAXException.getMessage()

Retorna un mensaje legible para humanos describiendo la condición de error.

SAXException.getException()

Retorna un objeto excepción encapsulado, o `None`.

`xml.sax.handler` — Base classes for SAX handlers

Source code Lib/xml/sax/handler.py [https://github.com/python/cpython/tree/3.14/Lib/xml/sax/handler.py]

The SAX API defines five kinds of handlers: content handlers, DTD handlers, error handlers, entity resolvers and lexical handlers. Applications normally only need to implement those interfaces whose events they are interested in; they can implement the interfaces in a single object or in multiple objects. Handler implementations should inherit from the base classes provided in the module **`xml.sax.handler`**, so that all methods get default implementations.

class xml.sax.handler.ContentHandler

> Esta es la principal interfaz de devolución de llamada en el SAX, y la más importante para las aplicaciones. El orden de los eventos en esta interfaz refleja el orden de la información en el documento.

class xml.sax.handler.DTDHandler

> Manejar eventos de DTD.
>
> Esta interfaz especifica solo los eventos DTD necesarios para el análisis básico (entidades y atributos no analizados).

class xml.sax.handler.EntityResolver

> Interfaz básica para resolver entidades. Si creas un objeto implementando esta interfaz, y luego registras el objeto con tu Parser, el parser (analizador) invocara al método en tu objeto para resolver todas las entidades externas.

class xml.sax.handler.ErrorHandler

Interfaz utilizada por el parser para presentar mensajes de error y advertencia a la aplicación. Los métodos de este objeto controlan si los errores se convierten inmediatamente en excepciones o se manejan de alguna otra manera.

class xml.sax.handler.LexicalHandler

Interface usada por el parser para representar eventos de baja frecuencia que pueden no ser de interés para muchas aplicaciones.

In addition to these classes, **`xml.sax.handler`** provides symbolic constants for the feature and property names.

xml.sax.handler.feature_namespaces

value: `"http://xml.org/sax/features/namespaces"`
true: Realizar el procesamiento del Namespace (espacio de nombres).
false: Opcionalmente no realizar el procesamiento del Namespace (espacio de nombres) (implica prefijos de espacio de nombres; por defecto).
acceso: (parsing) solo de lectura; (not parsing) lectura/ escritura

xml.sax.handler.feature_namespace_prefixes

value: `"http://xml.org/sax/features/namespace-prefixes"`
true: Reporte de los nombres prefijados originales y los atributos utilizados para las declaraciones del Namespace (espacio de nombres).
false: No informar de los atributos utilizados para las declaraciones del Namespace (espacio de nombres), y opcionalmente no informar de los nombres prefijados originales (por defecto).
acceso: (parsing) solo de lectura; (not parsing) lectura/ escritura

xml.sax.handler.feature_string_interning

value: `"http://xml.org/sax/features/string-interning"`
true: Todos los nombres de elementos, prefijos, nombres de atributos, URIs del Namespace (espacio de nombres) y nombres locales son internados usando la función interna incorporada.
false: Los nombres no están necesariamente internados, aunque pueden estarlo (por defecto).
acceso: (parsing) solo de lectura; (not parsing) lectura/ escritura

xml.sax.handler.feature_validation

value: `"http://xml.org/sax/features/validation"`
true: Reportar de todos los errores de validación (implica entidades generales externas y entidades de parámetros externos).
false: No informe de los errores de validación.
acceso: (parsing) solo de lectura; (not parsing) lectura/ escritura

xml.sax.handler.feature_external_ges

Advertencia

Enabling opens a vulnerability to external entity attacks [https://en.wikipedia.org/wiki/XML_external_entity_attack] if the parser is used with user-provided XML content. Please reflect on your threat model [https://en.wikipedia.org/wiki/Threat_model] before enabling this feature.

value: `"http://xml.org/sax/features/external-parameter-entities"`
true: Incluye todas las entidades externas generales de texto (text).
false: No incluya entidades generales externas.
acceso: (parsing) solo de lectura; (not parsing) lectura/ escritura

xml.sax.handler.feature_external_pes

value: `"http://xml.org/sax/features/external-parameter-entities"`
true: Incluye todas las entidades paramétricas externas, incluyendo el subconjunto DTD externo.
false: No incluya ninguna entidad paramétrica externa, ni siquiera el subconjunto DTD externo.
acceso: (parsing) solo de lectura; (not parsing) lectura/escritura

xml.sax.handler.all_features

Lista de todas las características.

xml.sax.handler.property_lexical_handler

value: `"http://xml.org/sax/properties/lexical-handler"`
data type: xml.sax.sax2lib.LexicalHandler (no soportado en Python 2)
descripción: Un handler (manipulador) de extensión opcional para eventos léxicos como los comentarios.
acceso: read/write (leer/escribir)

xml.sax.handler.property_declaration_handler

value: `"http://xml.org/sax/properties/declaration-handler"`
data type: xml.sax.sax2lib.DeclHandler (no soportado en Python 2)
description: Un gestor de extensión opcional para eventos relacionados con la DTD que no sean anotaciones y entidades no preparadas.
acceso: read/write (leer/escribir)

xml.sax.handler.property_dom_node

value: `"http://xml.org/sax/properties/dom-node"`
data type: org.w3c.dom.Node (not supported in Python 2)
descripción: Cuando se analiza, el nodo DOM actual que se

visita si se trata de un iterador DOM; cuando no se analiza, el nodo DOM raíz para la iteración.
acceso: (parsing) solo de lectura; (not parsing) lectura/escritura

xml.sax.handler.property_xml_string

value: `"http://xml.org/sax/properties/xml-string"`
tipo de datos: Bytes
description: La cadena literal de caracteres que fue la fuente del evento actual.
acceso: solo-lectura

xml.sax.handler.all_properties

Lista de todos los nombres de propiedades conocidas.

Objetos ContentHandler

Se espera que los usuarios subclasifiquen **`ContentHandler`** para apoyar su aplicación. Los siguientes métodos son llamados por el analizador en los eventos apropiados en el documento de entrada:

ContentHandler.setDocumentLocator(*locator*)

Invocado por el parser (analizador) para dar a la aplicación un localizador para determinar el origen de los eventos del documento.

Se recomienda encarecidamente a los parsers (analizadores) SAX (aunque no es absolutamente necesario) que proporcionen un localizador: si lo hace, debe proporcionar el localizador a la aplicación invocando este método antes de invocar cualquiera de los otros métodos de la interfaz DocumentHandler.

El localizador permite a la aplicación determinar la posición final de cualquier evento relacionado con el documento, incluso si el analizador no informa de un error. Normalmente, la aplicación utilizará esta información para informar de sus propios errores (como el contenido de los caracteres que no coinciden con las reglas de negocio de la aplicación). Es probable que la información retornada por el localizador no sea suficiente para su uso con un motor de búsqueda.

Tenga en cuenta que el localizador retornará la información correcta solo durante la invocación de los eventos en esta interfaz. La aplicación no debe intentar utilizarlo en ningún otro momento.

ContentHandler.startDocument()

Recibir la notificación del inicio de un documento.

El parser (analizador) SAX invocará este método solo una vez, antes que cualquier otro método en esta interfaz o en DTDHandler (excepto **`setDocumentLocator()`**).

ContentHandler.endDocument()

Recibir una notificación del final de un documento.

El analizador (parser) SAX invocará este método solo una vez, y será el último método invocado durante el análisis. El analizador no invocará este método hasta que no haya abandonado el análisis sintáctico (debido a un error irrecuperable) o haya llegado al final de la entrada.

ContentHandler.startPrefixMapping(*prefix*, *uri*)

Comienza el alcance de un mapeo del prefijo-URI Namespace.

La información de este evento no es necesaria para el procesamiento normal del Namespace (espacio de nombres): el lector de XML SAX sustituirá automáticamente los prefijos de los nombres de elementos y atributos cuando se active la función `feature_namespaces` default (el valor por defecto).

Sin embargo, hay casos en que las aplicaciones necesitan utilizar prefijos en los datos de los caracteres o en los valores de los atributos, en los que no se pueden expandir automáticamente de forma segura; los eventos `startPrefixMapping()` y `endPrefixMapping()` suministran la información a la aplicación para expandir los prefijos en esos contextos por sí mismos, si es necesario.

Note que los eventos `startPrefixMapping()` y `endPrefixMapping()` no están garantizados de estar apropiadamente anidados en relación a cada uno: todos los eventos `startPrefixMapping()` ocurrirán antes del correspondiente evento `startElement()`, y todos los eventos `endPrefixMapping()` ocurrirán después del correspondiente evento `endElement()`, pero su orden no está garantizado.

ContentHandler.endPrefixMapping(*prefix*)

Terminar con el alcance de un mapeo de prefijos-URI.

Ver `startPrefixMapping()` para más detalles. Este evento siempre ocurrirá después del correspondiente evento `endElement()`, pero el orden de los eventos `endPrefixMapping()` no está garantizado.

ContentHandler.startElement(*name*, *attrs*)

Señala el inicio de un elemento en modo no espacial.

The *name* parameter contains the raw XML 1.0 name of the element type as a string and the *attrs* parameter holds an object of the Attributes interface containing the attributes of the element. The object passed as *attrs* may be re-used by the parser; holding on to a reference to it is not a reliable way to keep a copy of the attributes. To keep a copy of the attributes, use the `copy()` method of the *attrs* object.

ContentHandler.endElement(*name*)

Señala el final de un elemento en modo no espacial.

El parámetro *name* contiene el nombre del tipo de elemento, al igual que con el evento `startElement()`.

ContentHandler.startElementNS(*name*, *qname*, *attrs*)

Señala el inicio de un elemento en el modo de Namespace (espacio de nombres).

The *name* parameter contains the name of the element type as a `(uri, localname)` tuple, the *qname* parameter contains the raw XML 1.0 name used in the source document, and the *attrs* parameter holds an instance of the AttributesNS interface containing the attributes of the element. If no namespace is associated with the element, the *uri* component of *name* will be `None`. The object passed as *attrs* may be re-used by the parser; holding on to a reference to it is not a reliable way to keep a copy of the attributes. To keep a copy of the attributes, use the **`copy()`** method of the *attrs* object.

Los parsers (analizadores) pueden establecer el parámetro *qname* como `None`, a menos que se active la función `feature_namespace_prefixes`.

ContentHandler.endElementNS(*name*, *qname*)

Señala el final de un elemento en el modo de namespace.

El parámetro *name* contiene el nombre del tipo de elemento, al igual que con el método **`startElementNS()`**, así como el parámetro *qname*.

ContentHandler.characters(*content*)

Recibir la notificación de los datos de los caracteres.

El Parser (analizador) llamará a este método para informar de cada dato de carácter. Los parsers (analizadores) SAX pueden retornar todos los datos de caracteres contiguos en un solo trozo, o pueden dividirlos en varios trozos; sin embargo, todos los caracteres de un mismo evento deben provenir de la misma entidad externa para que el Localizador proporcione información útil.

content puede ser una cadena o una instancia de bytes; el módulo lector de «Expat» siempre produce cadenas.

Nota

La anterior interfaz SAX 1 proporcionada por el Grupo de Interés Especial de Python XML utilizaba una interfaz más parecida a Java para este método. Dado que la mayoría de los parsers (analizadores) utilizados de Python no aprovecharon la interfaz más antigua, se eligió la firma más simple para reemplazarla. Para convertir el código antiguo en la nueva interfaz, se utilizó *content* en lugar de cortar el contenido con los antiguos parámetros *offset* y *length*.

ContentHandler.ignorableWhitespace(*whitespace*)

Recibir notificación de espacios en blanco ignorables en el contenido de los elementos.

Los parsers validadores deben utilizar este método para informar de cada trozo de espacio en blanco desconocido (véase la recomendación W3C XML 1.0, sección 2.10): los analizadores no validadores también pueden utilizar este método si son capaces de analizar y utilizar modelos de contenido.

Los parsers (analizadores) SAX pueden retornar todos los espacios blancos contiguos en un solo trozo, o pueden dividirlo en varios trozos; sin embargo, todos los caracteres de un mismo evento deben provenir de la misma entidad externa, para que el Localizador proporcione información útil.

ContentHandler.processingInstruction(*target*, *data*)

Recibir la notificación de una instrucción de procesamiento.

El Parser (analizador) invocará este método una vez por cada instrucción de procesamiento encontrada: nótese que las instrucciones de procesamiento pueden ocurrir antes o después del elemento principal del documento.

Un parser (analizador) SAX nunca debe informar de una declaración XML (XML 1.0, sección 2.8) o una declaración de texto (XML 1.0, sección 4.3.1) utilizando este método.

ContentHandler.skippedEntity(*name*)

Recibir la notificación de una entidad salteada.

El Parser invocará este método una vez por cada entidad omitida. Los procesos no validadores pueden omitir entidades si no han visto las declaraciones (porque, por ejemplo, la entidad fue declarada en un subconjunto DTD externo). Todos los procesos pueden omitir entidades externas, dependiendo de los valores de las propiedades `feature_external_ges` y `feature_external_pes`.

Objetos DTDHandler

`DTDHandler` la instancia provee los siguientes métodos:

DTDHandler.notationDecl(*name*, *publicId*, *systemId*)

Manejar un evento de declaración de anotación.

DTDHandler.unparsedEntityDecl(*name*, *publicId*, *systemId*, *ndata*)

Manejar un evento de declaración de entidad no preparada.

Objetos EntityResolver

EntityResolver.resolveEntity(*publicId*, *systemId*)

Resolver el identificador de sistema de una entidad y retornar ya sea el identificador de sistema para leerlo como una cadena, o una fuente de entrada para leerlo. La implementación por defecto retorna *systemId*.

Objetos ErrorHandler

Los objetos con esta interfaz se usan para recibir información de error y advertencia del **`XMLReader`**. Si creas un objeto que implemente esta interfaz, y luego registras el objeto con tu **`XMLReader`**, el parser (analizador) invocará a los métodos de tu objeto para informar de todas las advertencias y errores. Hay tres niveles de errores disponibles: advertencias, errores (posiblemente) recuperables y errores no recuperables. Todos los métodos toman un **`SAXParseException`** como único parámetro. Los errores y advertencias pueden ser convertidos en una excepción lanzando el objeto de excepción pasado.

ErrorHandler.error(*exception*)

> Invocación cuando el parser (analizador) encuentra un error recuperable. Si este método no lanza una excepción, el análisis sintáctico puede continuar, pero la aplicación no debe esperar más información del documento. Permitir que el parser (analizador) continúe puede permitir que se descubran errores adicionales en el documento de entrada.

ErrorHandler.fatalError(*exception*)

> Invocación cuando el parser (analizador) encuentra un error del que no puede recuperarse; se espera que el análisis sintáctico termine cuando vuelva este método.

ErrorHandler.warning(*exception*)

> Invocación cuando el parser (analizador) presenta información de advertencia menor a la aplicación. Se espera que el análisis sintáctico continúe cuando vuelva este método, y la información del documento seguirá pasando a la aplicación. Si se lanza una excepción con este método, el análisis sintáctico terminará.

Objetos DTDHandler

Handler (manipulador) de extensión opcional para eventos léxicos SAX2.

Este handler se usa para obtener información léxica de un documento XML. La información léxica incluye información que describe la codificación del documento usado u los comentarios

XML embebidos en el documento, como también una sección para el DTD y para cada sección CDTA. Estos manipuladores léxicos son usado de la misma forma que los manipuladores de contenido (content handlers).

Configurar el LexicalHandlerde un XMLReader usando el método setProperty con la propiedad identificadora `'http://xml.org/sax/properties/lexical-handler'`.

LexicalHandler.comment(*content*)

> Reporta un comentario en cualquier lugar en el documento (incluyendo el DTD y elementos por fuera del documento).

LexicalHandler.startDTD(*name*, *public_id*, *system_id*)

> Reporta el estado de las declaraciones del DTD en caso de tenga asociado un DTD.

LexicalHandler.endDTD()

> Reporta el fin de la declaración del DTD.

LexicalHandler.startCDATA()

> Reporta el inicio de la sección CDATA marcada.
>
> El contenido de la sección marcada del CDATA se reportará a través de manipulador de caracteres.

LexicalHandler.endCDATA()

> Reporta el final de la sección marcada del CDATA.

`xml.sax.saxutils` — SAX Utilities

Código fuente: Lib/xml/sax/saxutils.py [https://github.com/python/cpython/tree/3.14/Lib/xml/sax/saxutils.py]

The module **`xml.sax.saxutils`** contains a number of classes and functions that are commonly useful when creating SAX applications, either in direct use, or as base classes.

xml.sax.saxutils.escape(*data*, *entities={}*)

Escapar `'&'`, `'<'` y `'>'` en una cadena de datos.

Puede escapar otras cadenas de datos pasando un diccionario como el parámetro opcional *entities*. Las claves y los valores deben ser cadenas; cada clave será reemplazada por su valor correspondiente. Los caracteres `'&'`, `'<'` y `'>'` siempre se escapan, incluso si se proporciona *entities*.

Nota

Esta función debería ser usada únicamente para escapar caracteres que no pueden ser usados directamente en XML. No use esta función como una función general de traducción de cadenas de caracteres.

xml.sax.saxutils.unescape(*data*, *entities={}*)

Quitar el escape `'&amp;'`, `'&lt;'`, y `'&gt;'` en una cadena de datos.

You can unescape other strings of data by passing a dictionary as the optional *entities* parameter. The keys and values must all be strings; each key will be replaced with its corresponding value. `'&amp;'`, `'&lt;'`, and `'&gt;'` are always unescaped, even if *entities* is provided.

xml.sax.saxutils.quoteattr(*data*, *entities={}*)

Similar a `escape()`, pero también prepara *data* para usarse como un valor de atributo. El valor devuelto es una versión entrecomillada de *data* con los reemplazos adicionales necesarios. `quoteattr()` seleccionará un carácter de comillas basado en el contenido de *data*, intentando evitar codificar los caracteres de comillas en la cadena. Si los caracteres de comillas simples y dobles ya están en *data*, los caracteres de comillas dobles se codificarán y *data* se envolverá entre comillas dobles. La cadena resultante se puede utilizar directamente como un valor de atributo:

```
>>> print("<element attr=%s>" % quoteattr("ab ' cd
<element attr="ab ' cd &quot; ef">
```

Esta función es útil al generar valores de atributo para HTML o cualquier SGML utilizando la sintaxis concreta de referencia.

class xml.sax.saxutils.XMLGenerator(*out=None*, *encoding='iso-8859-1'*, *short_empty_elements=False*)

Esta clase implementa la interfaz `ContentHandler` escribiendo eventos SAX en un documento XML. En otras palabras, el uso de un `XMLGenerator` como controlador de contenido reproducirá el documento original que se está analizando. *out* debe ser un objeto similar a un archivo que por defecto sea *sys.stdout*. *encoding* es la codificación de la secuencia de salida que tiene como valor predeterminado `'iso-8859-1'`. *short_empty_elements* controla el formato de los elementos que no contienen contenido: si `False` (valor predeterminado) se emiten como un par de etiquetas de inicio/fin, si se establece en `True` se emiten como una sola etiqueta autocerrada.

Distinto en la versión 3.2: Added the *short_empty_elements* parameter.

class xml.sax.saxutils.XMLFilterBase(*base*)

Esta clase está diseñada para situarse entre un `XMLReader` y los manejadores de eventos de la aplicación cliente. Por defecto, no hace más que pasar las peticiones al lector y los eventos a los manejadores sin modificar, pero las subclases pueden sobrescribir métodos específicos para modificar el flujo de eventos o las peticiones de configuración a medida que pasan.

xml.sax.saxutils.prepare_input_source(*source*, *base*=")

Esta función toma una fuente de entrada y, opcionalmente, una URL base y retorna un objeto `InputSource` completamente resuelto y listo para ser leído. La fuente de entrada puede ser proporcionada como una cadena de caracteres, un objeto tipo archivo , o un objeto **`InputSource`**; los analizadores usarán esta función para implementar el argumento polimórfico *source* a su método `parse()`.

`xml.sax.xmlreader` — Interface for XML parsers

Código fuente: Lib/xml/sax/xmlreader.py [https://github.com/python/cpython/tree/3.14/Lib/xml/sax/xmlreader.py]

Los analizadores SAX implementan la interfaz `XMLReader`. Están implementados en un módulo Python, que debe proveer una función `create_parser()`. Esta función es invocada por `xml.sax.make_parser()` sin argumentos para crear un nuevo objeto analizador.

class xml.sax.xmlreader.XMLReader

Clase base que puede ser heredada por analizadores SAX.

class xml.sax.xmlreader.IncrementalParser

En algunos casos, es deseable no analizar una fuente de entrada a la vez, si no alimentar partes del documento a medida que estén disponibles. Tenga en cuenta que el lector normalmente no leerá el fichero completo, si no que también lo leerá por partes, aún así `parse()` no retornará hasta que el documento por completo es procesado. Por lo tanto, estas interfaces deben utilizarse si el comportamiento de bloqueo `parse()` no es deseable.

Cuando se crea una instancia del analizador, está listo para comenzar a aceptar información desde el método de alimentación inmediatamente. Después de que el análisis ha finalizado con una llamada para cerrar, se debe llamar al método de reinicio para que el analizador esté listo para aceptar información nueva, ya sea de la fuente o utilizando el método de análisis.

Tenga en cuenta que estos métodos *no* deben ser llamados durante el análisis, es decir, después de que el análisis ha sido llamado y antes de que regrese.

Por defecto, la clase también implementa el método de análisis de la interfaz XMLReader utilizando los métodos de alimentación, cierre y reinicio de la interfaz IncrementalParser en conveniencia a los escritores de controlador SAX 2.0.

class xml.sax.xmlreader.Locator

La interfaz para asociar un evento SAX con una ubicación del documento. Un objeto localizador retornará resultados válidos sólo durante llamadas a métodos DocumentHandler; en cualquier otro momento, los resultados son impredecibles. Si la información no está disponible, los métodos pueden retornar `None`.

class xml.sax.xmlreader.InputSource(*system_id=None*)

La encapsulación de la información necesaria por el **`XMLReader`** para leer entidades.

Esta clase puede incluir información sobre el identificador público, identificador del sistema, flujo de bytes (posiblemente con la información de codificación de caracteres) y/o el flujo de caracteres de una entidad.

Las aplicaciones crearán objetos de esta clase para uso en el método **`XMLReader.parse()`** y para retornar desde EntityResolver.resolveEntity.

Una **`InputSource`** pertenece a la aplicación, el **`XMLReader`** no tiene permitido modificar objetos **`InputSource`** pasados desde la aplicación, a pesar de que puede hacer copias y modificarlas.

class xml.sax.xmlreader.AttributesImpl(*attrs*)

Esta es una implementación de la interfaz **`Attributes`** (vea la sección La Interfaz Attributes). Este es un objeto de tipo diccionario que representa los atributos de elemento en una llamada `startElement()`. En adición a las operaciones de diccionario más útiles, soporta una serie de otros métodos como se describe en la interfaz. Los lectores deben crear una instancia de los objetos de esta clase; *attrs* debe ser un objeto de tipo diccionario que contenga un mapeo de nombres de atributo a valores de atributo.

class xml.sax.xmlreader.AttributesNSImpl(*attrs*, *qnames*)

Variante consciente del espacio de nombres de `AttributesImpl`, que se pasará a `startElementNS()`. Es derivada de `AttributesImpl`, pero entiende nombres de atributo como dos tuplas de *namespaceURI* y *localname*. En adición, provee una serie de métodos esperando nombres calificados como aparecen en el documento original. Esta clase implementa la interfaz `AttributesNS` (vea la sección La Interfaz AttributesNS).

Objetos XMLReader

La interfaz `XMLReader` soporta los siguientes métodos:

XMLReader.parse(*source*)

Procesa una fuente de entrada, produciendo eventos SAX. El objeto *source* puede ser un identificador de sistema (una cadena identificando la fuente de entrada – típicamente un nombre de fichero o una URL), un `pathlib.Path` o un objeto path-like, o un objeto `InputSource`. Cuando `parse()` retorna, la entrada es procesada completamente, y el objeto analizador puede ser descartado o reiniciado.

Distinto en la versión 3.5: Agregado soporte de flujo de caracteres.

Distinto en la versión 3.8: Agregado soporte de objetos path-like.

XMLReader.getContentHandler()

Retorna el `ContentHandler` actual.

XMLReader.setContentHandler(*handler*)

Establece el `ContentHandler` actual. Si ningún **`ContentHandler`** es establecido, los eventos de contenido serán descartados.

XMLReader.getDTDHandler()

Retorna el `DTDHandler` actual.

XMLReader.setDTDHandler(*handler*)

Establece el `DTDHandler` actual. Si ningún **`DTDHandler`** es establecido, los eventos DTD serán descartados.

XMLReader.getEntityResolver()

Retorna el `EntityResolver` actual.

XMLReader.setEntityResolver(*handler*)

Establece el `EntityResolver` actual. Si ningún **`EntityResolver`** es establecido, los intentos de resolver una entidad externa resultarán en la apertura del identificador de sistema para la entidad, y un error si no está disponible.

XMLReader.getErrorHandler()

Retorna el `ErrorHandler` actual.

XMLReader.setErrorHandler(*handler*)

Establece el manejador de errores actual. Si ningún `ErrorHandler` es establecido, se lanzarán errores como excepciones, y se imprimirán alertas.

XMLReader.setLocale(*locale*)

Permite a una aplicación establecer la configuración local para errores y alertas.

Analizadores SAX no son requeridos para proveer localización para errores y alertas; si no pueden soportar la configuración local solicitada, de cualquier forma, lanzarán una excepción

SAX. Las aplicaciones pueden solicitar un cambio local en medio del análisis.

XMLReader.getFeature(*featurename*)

Retorna la configuración actual para la característica *featurename*. Si la característica no es reconocida, **`SAXNotRecognizedException`** es lanzada. Los bien conocidos featurenames son listados en el módulo **`xml.sax.handler`**.

XMLReader.setFeature(*featurename*, *value*)

Establece el *featurename* a *value*. Si la característica no es reconocida, **`SAXNotRecognizedException`** es lanzada. Si la característica o su configuración no es soportada por el analizador, *SAXNotSupportedException* es lanzada.

XMLReader.getProperty(*propertyname*)

Retorna la configuración actual para la propiedad *propertyname*. Si la configuración no es reconocida, una **`SAXNotRecognizedException`** es lanzada. Las bien conocidas propertynames son listadas en el módulo **`xml.sax.handler`**.

XMLReader.setProperty(*propertyname*, *value*)

Establece el *propertyname* a *value*. Si la propiedad no es reconocida, **`SAXNotRecognizedException`** es lanzada. Si la propiedad o su configuración no es soportada por el analizador, *SAXNotSupportedException* es lanzada.

Objetos IncrementalParser

Las instancias de **`IncrementalParser`** ofrecen los siguientes métodos adicionales:

IncrementalParser.feed(*data*)

Procesa una parte de *data*.

IncrementalParser.close()

Asume el fin del documento. Eso verificará las condiciones bien formadas que pueden ser verificadas sólo al final, invocar manejadores, y puede limpiar los recursos asignados durante el análisis.

IncrementalParser.reset()

Este método es llamado después de que el cierre ha sido llamado para restablecer el analizador, de forma que esté listo para analizar nuevos documentos. Los resultados de llamar parse o feed después del cierre sin llamar a reset son indefinidos.

Objetos localizadores

Las instancias de `Locator` proveen estos métodos:

Locator.getColumnNumber()

Retorna el número de columna donde el evento actual comienza.

Locator.getLineNumber()

Retorna el número de línea donde el evento actual comienza.

Locator.getPublicId()

Retorna el identificador público para el evento actual.

Locator.getSystemId()

Retorna el identificador de sistema para el evento actual.

Objetos InputSource

InputSource.setPublicId(*id*)

Establece el identificador público de esta `InputSource`.

InputSource.getPublicId()

Retorna el identificador público para esta `InputSource`.

InputSource.setSystemId(*id*)

Establece el identificador de sistema de esta `InputSource`.

InputSource.getSystemId()

Retorna el identificador de sistema para esta `InputSource`.

InputSource.setEncoding(*encoding*)

Establece la codificación de caracteres para esta `InputSource`.

La codificación debe ser una cadena aceptable para una declaración de codificación XML (vea la sección 4.3.3 de la recomendación XML).

El atributo de codificación de la `InputSource` es ignorado si la `InputSource` contiene también un flujo de caracteres.

InputSource.getEncoding()

Obtiene la codificación de caracteres de esta InputSource.

InputSource.setByteStream(*bytefile*)

Establece el flujo de bytes (un binary file) para esta fuente de entrada.

El analizador SAX ignorará esto si existe también un flujo de caracteres especificado, pero utilizará un flujo de bytes en preferencia para abrir una conexión URI en sí.

Si la aplicación conoce la codificación de caracteres del flujo de bytes, debería establecerla con el método setEncoding.

InputSource.getByteStream()

Obtiene el flujo de bytes para esta fuente de entrada.

El método getEncoding retornará la codificación de caracteres para este flujo de bytes, o `None` si se desconoce.

InputSource.setCharacterStream(*charfile*)

Establece el flujo de caracteres (un text file) para esta fuente de entrada.

Si existe un flujo de caracteres especificado, el analizador SAX ignorará cualquier flujo de bytes y no intentará abrir una conexión URI al identificador de sistema.

InputSource.getCharacterStream()

Obtiene el flujo de caracteres para esta fuente de entrada.

La Interfaz `Attributes`

Los objetos **`Attributes`** implementa una porción del mapping protocol, incluyendo los métodos **`copy()`**, **`get()`**, **`__contains__()`**, **`items()`**, **`keys()`**, y **`values()`**. Los siguientes métodos también son provistos:

Attributes.getLength()

Retorna el número de atributos.

Attributes.getNames()

Retorna los nombres de los atributos.

Attributes.getType(*name*)

Retorna el tipo del atributo *name*, que es normalmente `'CDATA'`.

Attributes.getValue(*name*)

Retorna el valor del atributo *name*.

La Interfaz `AttributesNS`

Esta interfaz es un subtipo de la interfaz **`Attributes`** (vea la sección La Interfaz Attributes). Todos los métodos soportados por la interfaz están también disponibles en los objetos **`AttributesNS`**.

Los siguientes métodos están también disponibles:

AttributesNS.getValueByQName(*name*)

Retorna el valor para un nombre cualificado.

AttributesNS.getNameByQName(*name*)

Retorna el par `(namespace, localname)` para un *name* cualificado.

AttributesNS.getQNameByName(*name*)

Retorna el nombre cualificado para un par `(namespace, localname)`.

AttributesNS.getQNames()

Retorna los nombres cualificados para todos los atributos.

[1]

`xml.parsers.expat` — Fast XML parsing using Expat

Nota

If you need to parse untrusted or unauthenticated data, see XML security.

The `xml.parsers.expat` module is a Python interface to the Expat non-validating XML parser. The module provides a single extension type, `xmlparser`, that represents the current state of an XML parser. After an `xmlparser` object has been created, various attributes of the object can be set to handler functions. When an XML document is then fed to the parser, the handler functions are called for the character data and markup in the XML document.

Este módulo utiliza el módulo `pyexpat` para proporcionar acceso al analizador Expat. El uso directo del módulo `pyexpat` está obsoleto.

Este módulo proporciona una excepción y un tipo de objeto:

exception xml.parsers.expat.ExpatError

La excepción que se lanza cuando Expat informa un error. Consulte la sección Excepciones de ExpatError para obtener más información sobre cómo interpretar los errores de Expat.

exception xml.parsers.expat.error

Alias para `ExpatError`.

xml.parsers.expat.XMLParserType

El tipo de los valores de retorno de la función `ParserCreate()`.

The **`xml.parsers.expat`** module contains two functions:

xml.parsers.expat.ErrorString(*errno*)

Retorna una cadena explicativa para un número de error dado *errno*.

xml.parsers.expat.ParserCreate(*encoding=None*, *namespace_separator=None*)

Crea y retorna un nuevo objeto **`xmlparser`**. *encoding*, si se especifica, debe ser una cadena que nombre la codificación utilizada por los datos XML. Expat no admite tantas codificaciones como Python, y su repertorio de codificaciones no se puede ampliar; es compatible con UTF-8, UTF-16, ISO-8859-1 (Latin1) y ASCII. Si se proporciona *encoding* [1], anulará la codificación implícita o explícita del documento.

Parsers created through **`ParserCreate()`** are called «root» parsers, in the sense that they do not have any parent parser attached. Non-root parsers are created by **`parser.ExternalEntityParserCreate`**.

Expat puede, opcionalmente, realizar el procesamiento del espacio de nombres XML por usted, habilitado al proporcionar un valor para *namespace_separator*. El valor debe ser una cadena de un carácter; a **`ValueError`** se lanzará si la cadena tiene una longitud ilegal (`None` se considera lo mismo que una omisión). Cuando el procesamiento de espacios de nombres está habilitado, se expandirán los nombres de tipos de elementos y los nombres de atributos que pertenecen a un espacio de nombres. El nombre del elemento pasado a los controladores de elementos **`StartElementHandler`** y **`EndElementHandler`** será la concatenación del URI del espacio de nombres, el carácter separador del espacio de nombres y la parte local del nombre. Si el separador del espacio de nombres es un byte cero (`chr(0)`), el URI del espacio de nombres y la parte local se concatenarán sin ningún separador.

Por ejemplo, si *namespace_separator* se establece en un carácter de espacio (`' '`) y se analiza el siguiente documento:

```
<?xml version="1.0"?>
<root xmlns    = "http://default-namespace.org/"
      xmlns:py = "http://www.python.org/ns/">
  <py:elem1 />
  <elem2 xmlns="" />
</root>
```

`StartElementHandler` recibirá las siguientes cadenas para cada elemento:

```
http://default-namespace.org/ root
http://www.python.org/ns/ elem1
elem2
```

Debido a las limitaciones en la biblioteca `Expat` utilizada por **`pyexpat`**, la instancia **`xmlparser`** retorna solo se puede usar para analizar un solo documento XML. Llame a `ParserCreate` para cada documento para proporcionar instancias de analizador únicas.

Ver también

El Expat XML Parser **[http://www.libexpat.org/]**
Página de inicio del proyecto Expat.

Objetos XMLParser

Los objetos **`xmlparser`** tienen los siguientes métodos:

xmlparser.Parse(*data*[, *isfinal*])

Analiza el contenido de la cadena *data*, llamando a las funciones del controlador apropiadas para procesar los datos analizados. *isfinal* debe ser verdadero en la última llamada a este método; permite el análisis de un solo archivo en fragmentos, no el envío de varios archivos. *data* puede ser la cadena vacía en cualquier momento.

xmlparser.ParseFile(*file*)

Analizar la lectura de datos XML del objeto *file*. *file* solo necesita proporcionar el método `read(nbytes)`, devolviendo la cadena vacía cuando no hay más datos.

xmlparser.SetBase(*base*)

Establece la base que se utilizará para resolver URIs relativos en identificadores de sistema en declaraciones. La resolución de los identificadores relativos se deja en manos de la aplicación: este valor se pasará como el argumento *base* a las funciones **`ExternalEntityRefHandler()`**, **`NotationDeclHandler()`**, y **`UnparsedEntityDeclHandler()`**.

xmlparser.GetBase()

Retorna una cadena que contiene la base establecida por una llamada anterior a **`SetBase()`**, o `None` si no se ha llamado a **`SetBase()`**.

xmlparser.GetInputContext()

Retorna los datos de entrada que generaron el evento actual como una cadena. Los datos están en la codificación de la entidad que contiene el texto. Cuando se llama mientras un controlador de eventos no está activo, el valor de retorno es `None`.

xmlparser.ExternalEntityParserCreate(*context*[, *encoding*])

Cree un analizador «child» que se pueda utilizar para analizar una entidad analizada externa a la que hace referencia el contenido analizado por el analizador principal. El parámetro *context* debe ser la cadena pasada a la función del controlador **`ExternalEntityRefHandler()`**, que se describe a continuación. El analizador secundario se crea con **`order_attributes`** y **`specific_attributes`** establecidos en los valores de este analizador.

xmlparser.SetParamEntityParsing(*flag*)

Controle el análisis de las entidades de parámetros (incluido el subconjunto DTD externo). Los posibles valores de *flag* son `XML_PARAM_ENTITY_PARSING_NEVER`, `XML_PARAM_ENTITY_PARSING_UNLESS_STANDALONE` y `XML_PARAM_ENTITY_PARSING_ALWAYS`. Retorna verdadero si el establecimiento de la bandera fue exitoso.

xmlparser.UseForeignDTD([*flag*])

Llamar a esto con un valor verdadero para *flag* (el predeterminado) hará que Expat llame a `ExternalEntityRefHandler` con `None` para todos los argumentos para permitir que se cargue una DTD alternativa. Si el documento no contiene una declaración de tipo de documento, se seguirá llamando a `ExternalEntityRefHandler`, pero no se llamará a `StartDoctypeDeclHandler` y `EndDoctypeDeclHandler`.

Pasar un valor falso para *flag* cancelará una llamada anterior que pasó un valor verdadero, pero por lo demás no tiene ningún efecto.

Este método sólo se puede llamar antes de que se llamen los métodos `Parse()` o `ParseFile()`; llamarlo después de que cualquiera de ellos haya sido llamado causa que `ExpatError` se lanza con el atributo `code` establecido en `errors.codes[errors.XML_ERROR_CANT_CHANGE_FEATURE_O`

xmlparser.SetReparseDeferralEnabled(*enabled*)

Advertencia

Calling `SetReparseDeferralEnabled(False)` has security implications, as detailed below; please make sure to understand these consequences prior to using the `SetReparseDeferralEnabled` method.

Expat 2.6.0 introduced a security mechanism called «reparse deferral» where instead of causing denial of service through

quadratic runtime from reparsing large tokens, reparsing of unfinished tokens is now delayed by default until a sufficient amount of input is reached. Due to this delay, registered handlers may — depending of the sizing of input chunks pushed to Expat — no longer be called right after pushing new input to the parser. Where immediate feedback and taking over responsibility of protecting against denial of service from large tokens are both wanted, calling `SetReparseDeferralEnabled(False)` disables reparse deferral for the current Expat parser instance, temporarily or altogether. Calling `SetReparseDeferralEnabled(True)` allows re-enabling reparse deferral.

Note that **`SetReparseDeferralEnabled()`** has been backported to some prior releases of CPython as a security fix. Check for availability of **`SetReparseDeferralEnabled()`** using **`hasattr()`** if used in code running across a variety of Python versions.

Added in version 3.13.

xmlparser.GetReparseDeferralEnabled()

Returns whether reparse deferral is currently enabled for the given Expat parser instance.

Added in version 3.13.

`xmlparser` objects have the following methods to mitigate some common XML vulnerabilities.

xmlparser.SetAllocTrackerActivationThreshold(*threshold, /*)

Sets the number of allocated bytes of dynamic memory needed to activate protection against disproportionate use of RAM.

By default, parser objects have an allocation activation threshold of 64 MiB, or equivalently 67,108,864 bytes.

An **`ExpatError`** is raised if this method is called on a non-root parser. The corresponding **`lineno`** and **`offset`** should

not be used as they may have no special meaning.

Added in version 3.14.1.

xmlparser.SetAllocTrackerMaximumAmplification(*max_factor*, */*)

Sets the maximum amplification factor between direct input and bytes of dynamic memory allocated.

The amplification factor is calculated as `allocated / direct` while parsing, where `direct` is the number of bytes read from the primary document in parsing and `allocated` is the number of bytes of dynamic memory allocated in the parser hierarchy.

The *max_factor* value must be a non-NaN `float` value greater than or equal to 1.0. Amplification factors greater than 100.0 can be observed near the start of parsing even with benign files in practice. In particular, the activation threshold should be carefully chosen to avoid false positives.

By default, parser objects have a maximum amplification factor of 100.0.

An `ExpatError` is raised if this method is called on a non-root parser or if *max_factor* is outside the valid range. The corresponding `lineno` and `offset` should not be used as they may have no special meaning.

Nota

The maximum amplification factor is only considered if the threshold that can be adjusted by `SetAllocTrackerActivationThreshold()` is exceeded.

Added in version 3.14.1.

`xmlparser` los objetos tienen los siguientes atributos:

xmlparser.buffer_size

El tamaño del búfer usado cuando `buffer_text` es verdadero. Se puede establecer un nuevo tamaño de búfer asignando un nuevo valor entero a este atributo. Cuando se cambia el tamaño, el búfer se vaciará.

xmlparser.buffer_text

Setting this to true causes the **`xmlparser`** object to buffer textual content returned by Expat to avoid multiple calls to the `CharacterDataHandler()` callback whenever possible. This can improve performance substantially since Expat normally breaks character data into chunks at every line ending. This attribute is false by default, and may be changed at any time. Note that when it is false, data that does not contain newlines may be chunked too.

xmlparser.buffer_used

Si `buffer_text` está habilitado, el número de bytes almacenados en el búfer. Estos bytes representan texto codificado en UTF-8. Este atributo no tiene una interpretación significativa cuando **`buffer_text`** es falso.

xmlparser.ordered_attributes

Establecer este atributo en un número entero distinto de cero hace que los atributos se informen como una lista en lugar de un diccionario. Los atributos se presentan en el orden que se encuentran en el texto del documento. Para cada atributo, se presentan dos entradas de lista: el nombre del atributo y el valor del atributo. (Las versiones anteriores de este módulo también usaban este formato). De forma predeterminada, este atributo es falso; se puede cambiar en cualquier momento.

xmlparser.specified_attributes

Si se establece en un número entero distinto de cero, el analizador informará solo los atributos que se especificaron en la instancia del documento y no los que se derivaron de declaraciones de atributos. Las aplicaciones que establecen esto deben tener especial cuidado al utilizar la información adicional disponible en las declaraciones según sea necesario para cumplir con los estándares para el comportamiento de los procesadores XML. De forma predeterminada, este atributo es falso; se puede cambiar en cualquier momento.

Los siguientes atributos contienen valores relacionados con el error más reciente encontrado por un objeto **`xmlparser`**, y solo tendrán los valores correctos una vez que una llamada a **`Parse()`** o **`ParseFile()`** haya lanzado una excepción **`xml.parsers.expat.ExpatError`**.

xmlparser.ErrorByteIndex

Índice de bytes en el que se produjo un error.

xmlparser.ErrorCode

Código numérico que especifica el problema. Este valor puede pasarse a la función **`ErrorString()`**, o compararse con una de las constantes definidas en el objeto `errors`.

xmlparser.ErrorColumnNumber

Número de columna en la que se produjo un error.

xmlparser.ErrorLineNumber

Número de línea en la que ocurrió un error.

Los siguientes atributos contienen valores relacionados con la ubicación actual del análisis en un objeto **`xmlparser`**. Durante una devolución de llamada que informa un evento de análisis, indican la ubicación del primero de la secuencia de caracteres que generó el evento. Cuando se llama fuera de una devolución de llamada, la posición indicada estará justo después del último evento de análisis (independientemente de si hubo una devolución de llamada asociada).

xmlparser.CurrentByteIndex

Índice de bytes actual en la entrada del analizador.

xmlparser.CurrentColumnNumber

Número de columna actual en la entrada del analizador.

xmlparser.CurrentLineNumber

Número de línea actual en la entrada del analizador.

Aquí está la lista de controladores que se pueden configurar. Para configurar un controlador en un objeto **`xmlparser`** *o*, use `o.handlername = func`. *handlername* debe tomarse de la siguiente lista, y *func* debe ser un objeto invocable que acepte el número correcto de argumentos. Los argumentos son todos cadenas, a menos que se indique lo contrario.

xmlparser.XmlDeclHandler(*version*, *encoding*, *standalone*)

Se llama cuando se analiza la declaración XML. La declaración XML es la declaración (opcional) de la versión aplicable de la recomendación XML, la codificación del texto del documento y una declaración «independiente» opcional. *version* y *encoding* serán cadenas, y *standalone* será `1` si el documento se declara independiente, `0` si se declara no independiente o `-1` si se omitió la cláusula independiente. Esto solo está disponible con la versión Expat 1.95.0 o más reciente.

xmlparser.StartDoctypeDeclHandler(*doctypeName*, *systemId*, *publicId*, *has_internal_subset*)

Called when Expat begins parsing the document type declaration (`<!DOCTYPE ...`). The *doctypeName* is provided exactly as presented. The *systemId* and *publicId* parameters give the system and public identifiers if specified, or `None` if omitted. *has_internal_subset* will be true if the document contains an internal document declaration subset. This requires Expat version 1.2 or newer.

xmlparser.EndDoctypeDeclHandler()

Se llama cuando Expat termina de analizar la declaración del tipo de documento. Esto requiere Expat versión 1.2 o más reciente.

xmlparser.ElementDeclHandler(*name*, *model*)

Se llama una vez para cada declaración de tipo de elemento. *name* es el nombre del tipo de elemento y *model* es una representación del modelo de contenido.

xmlparser.AttlistDeclHandler(*elname*, *attname*, *type*, *default*, *required*)

Se llama para cada atributo declarado para un tipo de elemento. Si una declaración de lista de atributos declara tres atributos, este controlador se llama tres veces, una para cada atributo. *elname* es el nombre del elemento al que se aplica la declaración y *attname* es el nombre del atributo declarado. El tipo de atributo es una cadena pasada como *type*; los valores posibles son `'CDATA'`, `'ID'`, `'IDREF'`, … *default* da el valor predeterminado para el atributo utilizado cuando el atributo no está especificado por el instancia de documento, o `None` si no hay un valor predeterminado (valores `#IMPLIED`). Si se requiere que el atributo se proporcione en la instancia del documento, *required* será verdadero. Esto requiere la versión Expat 1.95.0 o más reciente.

xmlparser.StartElementHandler(*name*, *attributes*)

Llamado para el inicio de cada elemento. *name* es una cadena que contiene el nombre del elemento, y *attributes* son los atributos del elemento. Si **`order_attributes`** es verdadero, esta es una lista (ver **`order_attributes`** para una descripción completa). De lo contrario, es un diccionario que asigna nombres a valores.

xmlparser.EndElementHandler(*name*)

Llamado al final de cada elemento.

xmlparser.ProcessingInstructionHandler(*target*, *data*)

Llamado para cada instrucción de procesamiento.

xmlparser.CharacterDataHandler(*data*)

Called for character data. This will be called for normal character data, CDATA marked content, and ignorable whitespace. Applications which must distinguish these cases can use the **`StartCdataSectionHandler`**, **`EndCdataSectionHandler`**, and **`ElementDeclHandler`** callbacks to collect the required information. Note that the character data may be chunked even if it is short and so you may receive more than one call to **`CharacterDataHandler()`**. Set the **`buffer_text`** instance attribute to `True` to avoid that.

xmlparser.UnparsedEntityDeclHandler(*entityName, base, systemId, publicId, notationName*)

Se llama para declaraciones de entidad sin analizar (NDATA). Esto solo está presente para la versión 1.2 de la biblioteca Expat; para versiones más recientes, use **`EntityDeclHandler`** en su lugar. (La función subyacente en la biblioteca Expat se ha declarado obsoleta).

xmlparser.EntityDeclHandler(*entityName, is_parameter_entity, value, base, systemId, publicId, notationName*)

Llamado para todas las declaraciones de entidad. Para parámetros y entidades internas, *value* será una cadena que proporciona el contenido declarado de la entidad; esto será `None` para entidades externas. El parámetro *notationName* será `None` para las entidades analizadas y el nombre de la notación para las entidades no analizadas. *is_parameter_entity* será verdadero si la entidad es una entidad de parámetro o falso para las entidades generales (la mayoría de las aplicaciones solo deben preocuparse por las entidades generales). Esto solo está disponible a partir de la versión 1.95.0 de la biblioteca Expat.

xmlparser.NotationDeclHandler(*notationName, base, systemId, publicId*)

Se llama para declaraciones de notación. *notationName*, *base* y *systemId* y *publicId* son cadenas si se dan. Si se omite el identificador público, *publicId* será `None`.

xmlparser.StartNamespaceDeclHandler(*prefix*, *uri*)

Se llama cuando un elemento contiene una declaración de espacio de nombres. Las declaraciones de espacio de nombres se procesan antes de que se llame a **`StartElementHandler`** para el elemento en el que se colocan las declaraciones.

xmlparser.EndNamespaceDeclHandler(*prefix*)

Se llama cuando se alcanza la etiqueta de cierre para un elemento que contiene una declaración de espacio de nombres. Esto se llama una vez para cada declaración de espacio de nombres en el elemento en el orden inverso al que se llamó **`StartNamespaceDeclHandler`** para indicar el inicio del alcance de cada declaración de espacio de nombres. Las llamadas a este controlador se realizan después del correspondiente **`EndElementHandler`** para el final del elemento.

xmlparser.CommentHandler(*data*)

Llamado para comentarios. *data* es el texto del comentario, excluyendo el `'<!--'` inicial y el final `'-->'`.

xmlparser.StartCdataSectionHandler()

Llamado al comienzo de una sección CDATA. Esto y **`EndCdataSectionHandler`** son necesarios para poder identificar el inicio sintáctico y el final de las secciones CDATA.

xmlparser.EndCdataSectionHandler()

Llamado al final de una sección CDATA.

xmlparser.DefaultHandler(*data*)

Se invoca por cualquier carácter del documento XML para el que no se ha especificado ningún controlador aplicable. Esto significa caracteres que forman parte de una construcción que se podría informar, pero para los que no se ha proporcionado ningún controlador.

xmlparser.DefaultHandlerExpand(*data*)

Es lo mismo que **`DefaultHandler()`**, pero no inhibe la expansión de entidades internas. La referencia de la entidad no se pasará al controlador predeterminado.

xmlparser.NotStandaloneHandler()

Se llama si el documento XML no se ha declarado como un documento independiente. Esto sucede cuando hay un subconjunto externo o una referencia a una entidad de parámetro, pero la declaración XML no establece independiente en `yes` en una declaración XML. Si este controlador retorna `0`, el analizador lanzará un error **`XML_ERROR_NOT_STANDALONE`**. Si este controlador no está configurado, el analizador no lanza ninguna excepción para esta condición.

xmlparser.ExternalEntityRefHandler(*context*, *base*, *systemId*, *publicId*)

Advertencia

Implementing a handler that accesses local files and/or the network may create a vulnerability to external entity attacks [https://en.wikipedia.org/wiki/XML_external_entity_attack] if **`xmlparser`** is used with user-provided XML content. Please reflect on your threat model [https://en.wikipedia.org/wiki/Threat_model] before implementing this handler.

Llamado para referencias a entidades externas. *base* es la base actual, según lo establecido por una llamada anterior a **`SetBase()`**. Los identificadores público y del sistema, *systemId* y *publicId*, son cadenas si se dan; si no se proporciona el identificador público, *publicId* será `None`. El valor *context* es opaco y solo debe usarse como se describe a continuación.

Para que se analicen las entidades externas, se debe implementar este controlador. Es responsable de crear el sub-analizador usando

`ExternalEntityParserCreate(context)`, inicializándolo con las devoluciones de llamada apropiadas y analizando la entidad. Este controlador debería devolver un número entero; si retorna `0`, el analizador lanzará un error **`XML_ERROR_EXTERNAL_ENTITY_HANDLING`**; de lo contrario, el análisis continuará.

Si no se proporciona este controlador, las entidades externas se informan mediante la devolución de llamada **`DefaultHandler`**, si se proporciona.

Excepciones de ExpatError

Las excepciones **`ExpatError`** tienen una serie de atributos interesantes:

ExpatError.code

Número de error interno del expatriado para el error específico. El diccionario **`errors.messages`** asigna estos números de error a los mensajes de error de Expat. Por ejemplo:

```
from xml.parsers.expat import ParserCreate, ExpatEr

p = ParserCreate()
try:
    p.Parse(some_xml_document)
except ExpatError as err:
    print("Error:", errors.messages[err.code])
```

El módulo **`errors`** también proporciona constantes de mensajes de error y un diccionario **`codes`** mapeando estos mensajes a los códigos de error, ver más abajo.

ExpatError.lineno

Número de línea en la que se detectó el error. La primera línea está numerada como `1`.

ExpatError.offset

Carácter desplazado en la línea donde ocurrió el error. La primera columna está numerada como `0`.

Ejemplo

El siguiente programa define tres controladores que simplemente imprimen sus argumentos.

```
import xml.parsers.expat

# 3 handler functions
def start_element(name, attrs):
    print('Start element:', name, attrs)
def end_element(name):
    print('End element:', name)
def char_data(data):
    print('Character data:', repr(data))

p = xml.parsers.expat.ParserCreate()

p.StartElementHandler = start_element
p.EndElementHandler = end_element
p.CharacterDataHandler = char_data

p.Parse("""<?xml version="1.0"?>
<parent id="top"><child1 name="paul">Text goes here</chi
<child2 name="fred">More text</child2>
</parent>""", 1)
```

La salida de este programa es:

```
Start element: parent {'id': 'top'}
Start element: child1 {'name': 'paul'}
Character data: 'Text goes here'
End element: child1
Character data: '\n'
Start element: child2 {'name': 'fred'}
Character data: 'More text'
End element: child2
```

```
Character data: '\n'
End element: parent
```

Descripciones del modelo de contenido

Los modelos de contenido se describen mediante tuplas anidadas. Cada tupla contiene cuatro valores: el tipo, el cuantificador, el nombre y una tupla de niños. Los niños son simplemente descripciones adicionales del modelo de contenido.

The values of the first two fields are constants defined in the **`xml.parsers.expat.model`** module. These constants can be collected in two groups: the model type group and the quantifier group.

Las constantes en el grupo de tipos de modelo son:

xml.parsers.expat.model.XML_CTYPE_ANY

> Se declaró que el elemento nombrado por el nombre del modelo tiene un modelo de contenido de `ANY`.

xml.parsers.expat.model.XML_CTYPE_CHOICE

> El elemento nombrado permite elegir entre varias opciones; se utiliza para modelos de contenido como `(A | B | C)`.

xml.parsers.expat.model.XML_CTYPE_EMPTY

> Los elementos que se declaran `EMPTY` tienen este tipo de modelo.

xml.parsers.expat.model.XML_CTYPE_MIXED

xml.parsers.expat.model.XML_CTYPE_NAME

xml.parsers.expat.model.XML_CTYPE_SEQ

> Los modelos que representan una serie de modelos que siguen uno tras otro se indican con este tipo de modelo. Se utiliza para modelos como `(A, B, C)`.

Las constantes en el grupo cuantificador son:

xml.parsers.expat.model.XML_CQUANT_NONE

No se proporciona ningún modificador, por lo que puede aparecer exactamente una vez, como para `A`.

xml.parsers.expat.model.XML_CQUANT_OPT

El modelo es opcional: puede aparecer una vez o no aparecer, como para `A?`.

xml.parsers.expat.model.XML_CQUANT_PLUS

El modelo debe aparecer una o más veces (como `A+`).

xml.parsers.expat.model.XML_CQUANT_REP

El modelo debe aparecer cero o más veces, como en `A*`.

Constantes de error de expansión

The following constants are provided in the **`xml.parsers.expat.errors`** module. These constants are useful in interpreting some of the attributes of the **`ExpatError`** exception objects raised when an error has occurred. Since for backwards compatibility reasons, the constants" value is the error *message* and not the numeric error *code*, you do this by comparing its **`code`** attribute with `errors.codes[errors.XML_ERROR_CONSTANT_NAME]`.

El módulo `errors` tiene los siguientes atributos:

xml.parsers.expat.errors.codes

Un diccionario que asigna descripciones de cadenas a sus códigos de error.

Added in version 3.2.

xml.parsers.expat.errors.messages

Un diccionario que asigna códigos de error numéricos a sus descripciones de cadenas.

Added in version 3.2.

xml.parsers.expat.errors.XML_ERROR_ASYNC_ENTITY

xml.parsers.expat.errors.XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF

Una referencia de entidad en un valor de atributo se refiere a una entidad externa en lugar de una entidad interna.

xml.parsers.expat.errors.XML_ERROR_BAD_CHAR_REF

Una referencia de carácter se refiere a un carácter que es ilegal en XML (por ejemplo, carácter `0`, o "`&#0;`").

xml.parsers.expat.errors.XML_ERROR_BINARY_ENTITY_REF

Una referencia de entidad se refería a una entidad que se declaró con una notación, por lo que no se puede analizar.

xml.parsers.expat.errors.XML_ERROR_DUPLICATE_ATTRIBUTE

Un atributo se utilizó más de una vez en una etiqueta de inicio.

xml.parsers.expat.errors.XML_ERROR_INCORRECT_ENCODING

xml.parsers.expat.errors.XML_ERROR_INVALID_TOKEN

Se lanza cuando un byte de entrada no se puede asignar correctamente a un carácter; por ejemplo, un byte NUL (valor `0`) en un flujo de entrada UTF-8.

xml.parsers.expat.errors.XML_ERROR_JUNK_AFTER_DOC_ELEMENT

Se produjo algo diferente a los espacios en blanco después del elemento del documento.

xml.parsers.expat.errors.XML_ERROR_MISPLACED_XML_PI

Se encontró una declaración XML en algún lugar que no sea el comienzo de los datos de entrada.

xml.parsers.expat.errors.XML_ERROR_NO_ELEMENTS

El documento no contiene elementos (XML requiere que todos los documentos contengan exactamente un elemento de nivel superior)..

xml.parsers.expat.errors.XML_ERROR_NO_MEMORY

Expat no pudo asignar memoria internamente.

xml.parsers.expat.errors.XML_ERROR_PARAM_ENTITY_REF

Se encontró una referencia de entidad de parámetro donde no estaba permitida.

xml.parsers.expat.errors.XML_ERROR_PARTIAL_CHAR

Se encontró un carácter incompleto en la entrada.

xml.parsers.expat.errors.XML_ERROR_RECURSIVE_ENTITY_REF

Una referencia de entidad contenía otra referencia a la misma entidad; posiblemente a través de un nombre diferente, y posiblemente indirectamente.

xml.parsers.expat.errors.XML_ERROR_SYNTAX

Se encontró algún error de sintaxis no especificado.

xml.parsers.expat.errors.XML_ERROR_TAG_MISMATCH

Una etiqueta final no coincidía con la etiqueta inicial abierta más interna.

xml.parsers.expat.errors.XML_ERROR_UNCLOSED_TOKEN

Algún token (como una etiqueta de inicio) no se cerró antes del final de la transmisión o se encontró el siguiente token.

xml.parsers.expat.errors.XML_ERROR_UNDEFINED_ENTITY

Se hizo referencia a una entidad que no estaba definida.

xml.parsers.expat.errors.XML_ERROR_UNKNOWN_ENCODING

La codificación del documento no es compatible con Expat.

xml.parsers.expat.errors.XML_ERROR_UNCLOSED_CDATA_SECTION

No se cerró una sección marcada con CDATA.

xml.parsers.expat.errors.XML_ERROR_EXTERNAL_ENTITY_HANDLING

xml.parsers.expat.errors.XML_ERROR_NOT_STANDALONE

El analizador determinó que el documento no era «independiente» aunque se declaró en la declaración XML, y el **`NotStandaloneHandler`** se estableció y devolvió `0`.

xml.parsers.expat.errors.XML_ERROR_UNEXPECTED_STATE

xml.parsers.expat.errors.XML_ERROR_ENTITY_DECLARED_IN_PE

xml.parsers.expat.errors.XML_ERROR_FEATURE_REQUIRES_XML_DTD

An operation was requested that requires DTD support to be compiled in, but Expat was configured without DTD support. This should never be reported by a standard build of the **`xml.parsers.expat`** module.

xml.parsers.expat.errors.XML_ERROR_CANT_CHANGE_FEATURE_ONCE_PA

Se solicitó un cambio de comportamiento después de que comenzó el análisis que solo se puede cambiar antes de que haya comenzado el análisis. Esto (actualmente) solo lanzado por **`UseForeignDTD()`**.

xml.parsers.expat.errors.XML_ERROR_UNBOUND_PREFIX

Se encontró un prefijo no declarado cuando se habilitó el procesamiento del espacio de nombres.

xml.parsers.expat.errors.XML_ERROR_UNDECLARING_PREFIX

El documento intentó eliminar la declaración de espacio de nombres asociada con un prefijo.

xml.parsers.expat.errors.XML_ERROR_INCOMPLETE_PE

El documento no contenía ningún elemento de documento.

xml.parsers.expat.errors.XML_ERROR_XML_DECL

El documento no contenía ningún elemento de documento.

xml.parsers.expat.errors.XML_ERROR_TEXT_DECL

Se produjo un error al analizar una declaración de texto en una entidad externa.

xml.parsers.expat.errors.XML_ERROR_PUBLICID

Se encontraron caracteres en la identificación pública que no están permitidos.

xml.parsers.expat.errors.XML_ERROR_SUSPENDED

La operación solicitada se realizó en un analizador suspendido, pero no está permitida. Esto incluye intentos de proporcionar información adicional o detener el analizador.

xml.parsers.expat.errors.XML_ERROR_NOT_SUSPENDED

Se realizó un intento de reanudar el analizador cuando no se había suspendido.

xml.parsers.expat.errors.XML_ERROR_ABORTED

Esto no se debe informar a las aplicaciones Python.

xml.parsers.expat.errors.XML_ERROR_FINISHED

La operación solicitada se realizó en un analizador que terminó de analizar la entrada, pero no está permitido. Esto incluye intentos de proporcionar información adicional o detener el analizador.

xml.parsers.expat.errors.XML_ERROR_SUSPEND_PE

xml.parsers.expat.errors.XML_ERROR_RESERVED_PREFIX_XML

Se hizo un intento de declarar o remover el prefijo del namespace reservado `xml` o de enlazarlo con otro namespace URI.

xml.parsers.expat.errors.XML_ERROR_RESERVED_PREFIX_XMLNS

Se hizo un intento por declarar o remover el prefijo del namespace reservado `xmlns`.

xml.parsers.expat.errors.XML_ERROR_RESERVED_NAMESPACE_URI

Se hizo un intento de enlace entre la URI y uno de los prefijos del namespace reservado `xml` y `xmlns` a otro prefijo de namespace.

xml.parsers.expat.errors.XML_ERROR_INVALID_ARGUMENT

Esto no se debe informar a las aplicaciones Python.

xml.parsers.expat.errors.XML_ERROR_NO_BUFFER

Esto no se debe informar a las aplicaciones Python.

xml.parsers.expat.errors.XML_ERROR_AMPLIFICATION_LIMIT_BREACH

El límite en el factor de amplificación de entrada (de DTD y entidades) ha sido sobrepasada.

xml.parsers.expat.errors.XML_ERROR_NOT_STARTED

The parser was tried to be stopped or suspended before it started.

Added in version 3.14.

Notas al pie

La cadena de codificación incluida en la salida XML debe cumplir con los estándares apropiados. Por ejemplo, «UTF-8» es válido, pero «UTF8» no lo es. Consulte https://www.w3.org/TR/2006/REC-xml11-20060816/#NT-EncodingDecl y https://www.iana.org/assignments/character-sets/character-sets.xhtml.

Protocolos y soporte de Internet

Los módulos descritos en este capítulo implementan protocolos de Internet y son compatibles con la tecnología relacionada. Todos están implementados en Python. La mayoría de estos módulos requieren la presencia del módulo dependiente del sistema **`socket`**, que actualmente es compatible con las plataformas más populares. Aquí hay una descripción general:

- **`webbrowser`** — Convenient web-browser controller
    - Command-line interface
    - Browser controller objects
- **`wsgiref`** — WSGI Utilities and Reference Implementation
    - **`wsgiref.util`** – WSGI environment utilities
    - **`wsgiref.headers`** – WSGI response header tools
    - **`wsgiref.simple_server`** – a simple WSGI HTTP server
    - **`wsgiref.validate`** — WSGI conformance checker
    - **`wsgiref.handlers`** – server/gateway base classes
    - **`wsgiref.types`** – WSGI types for static type checking
    - Ejemplos
- **`urllib`** — URL handling modules
- **`urllib.request`** — Extensible library for opening URLs
    - Objetos Request
    - Objetos OpenerDirector
    - Objetos BaseHandler
    - Objetos HTTPRedirectHandler
    - Objetos HTTPCookieProcessor
    - Objetos ProxyHandler
    - Objetos HTTPPasswordMgr

  - Objetos HTTPPasswordMgrWithPriorAuth
  - Objetos AbstractBasicAuthHandler
  - Objetos HTTPBasicAuthHandler
  - Objetos ProxyBasicAuthHandler
  - Objetos AbstractDigestAuthHandler
  - Objetos HTTPDigestAuthHandler
  - Objetos ProxyDigestAuthHandler
  - Objetos HTTPHandler
  - Objetos HTTPSHandler
  - Objetos FileHandler
  - Objetos DataHandler
  - Objetos FTPHandler
  - Objetos CacheFTPHandler
  - Objetos UnknownHandler
  - Objetos HTTPErrorProcessor
  - Ejemplos
  - Interfaz heredada
  - `urllib.request` Restrictions

- `urllib.response` — Response classes used by urllib
- `urllib.parse` — Parse URLs into components

  - Análisis de URL
  - Análisis de seguridad de URL
  - Análisis de bytes codificados ASCII
  - Resultados del análisis estructurado
  - Cita de URL

- `urllib.error` — Exception classes raised by urllib.request
- `urllib.robotparser` — Parser for robots.txt
- `http` — HTTP modules

  - Códigos de estado HTTP
  - Categoría de estado HTTP
  - Métodos HTTP

- `http.client` — HTTP protocol client

  - Objetos de `HTTPConnection`
  - Objetos de `HTTPResponse`
  - Ejemplos

    - Objetos de **`HTTPMessage`**

- **`ftplib`** — FTP protocol client

    - Reference

        - FTP objects
        - FTP_TLS objects
        - Module variables

- **`poplib`** — POP3 protocol client

    - Objetos POP3
    - Ejemplo POP3

- **`imaplib`** — IMAP4 protocol client

    - Objetos de IMAP4
    - Ejemplo IMAP4

- **`smtplib`** — SMTP protocol client

    - Objetos SMTP
    - Ejemplo SMTP

- **`uuid`** — UUID objects according to **RFC 9562**

    - Uso de la línea de comandos
    - Ejemplo
    - Ejemplo de línea de comandos

- **`socketserver`** — A framework for network servers

    - Notas de creación del servidor
    - Objetos de servidor
    - Solicitar objetos de controlador
    - Ejemplos

        - **`socketserver.TCPServer`** Ejemplo
        - **`socketserver.UDPServer`** Ejemplo
        - Mixins asincrónicos

- `http.server` — HTTP servers
    - Command-line interface
    - Security considerations
- `http.cookies` — HTTP state management
    - Objetos de cookie
    - Objetos Morsel
    - Ejemplo
- `http.cookiejar` — Cookie handling for HTTP clients
    - Objetos CookieJar y FileCookieJar
    - Subclases FileCookieJar y co-operación con navegadores web
    - Objetos CookiePolicy
    - Objetos DefaultCookiePolicy
    - Objetos Cookie
    - Ejemplos
- `xmlrpc` — Módulos XMLRPC para cliente y servidor
- `xmlrpc.client` — XML-RPC client access
    - Objetos *ServerProxy*
    - Objetos *DateTime*
    - Objetos binarios
    - Objetos Faults
    - Objetos ProtocolError
    - Objetos MultiCall
    - Funciones de Conveniencia
    - Ejemplo de uso de cliente
    - Ejemplo de uso de cliente y servidor
- `xmlrpc.server` — Basic XML-RPC servers
    - SimpleXMLRPCServer objects
        - SimpleXMLRPCServer example
    - CGIXMLRPCRequestHandler
    - Documentando el servidor XMLRPC

    - DocXMLRPCServer objects
    - DocCGIXMLRPCRequestHandler
- **`ipaddress`** — IPv4/IPv6 manipulation library
    - Funciones de fábrica de conveniencia
    - Direcciones IP
        - Objetos de dirección
        - Conversión a cadenas y enteros
        - Operadores
            - Operadores de comparación
            - Operadores aritméticos
    - Definiciones de red IP
        - Prefijo, máscara de red y máscara de host
        - Objetos de red
        - Operadores
            - Operadores lógicos
            - Iteración
            - Redes como contenedores de direcciones
    - Objetos de interfaz
        - Operadores
            - Operadores lógicos
    - Otras funciones de nivel de módulo
    - Excepciones personalizadas

[1]

`webbrowser` — Convenient web-browser controller

Código fuente: Lib/webbrowser.py [https://github.com/python/cpython/tree/3.14/Lib/webbrowser.py]

The **`webbrowser`** module provides a high-level interface to allow displaying web-based documents to users. Under most circumstances, simply calling the **`open()`** function from this module will do the right thing.

En Unix, son privilegiados los navegadores gráficos bajo X11, pero serán usados navegadores en modo texto si los navegadores gráficos no están disponibles o un *display* X11 no está disponible. Si se usan navegadores en modo texto, el proceso invocador será bloqueado hasta que el usuario salga del navegador.

If the environment variable **`BROWSER`** exists, it is interpreted as the **`os.pathsep`**-separated list of browsers to try ahead of the platform defaults. When the value of a list part contains the string `%s`, then it is interpreted as a literal browser command line to be used with the argument URL substituted for `%s`; if the value is a single word that refers to one of the already registered browsers this browser is added to the front of the search list; if the part does not contain `%s`, it is simply interpreted as the name of the browser to launch. [1]

Distinto en la versión 3.14: The **`BROWSER`** variable can now also be used to reorder the list of platform defaults. This is particularly useful on macOS where the platform defaults do not refer to command-line tools on **`PATH`**.

En plataformas no Unix o cuando está disponible un navegador remoto en Unix, el proceso de control no esperará a que el usuario finalice el navegador, sino que permitirá al navegador remoto mantener su propia ventana en la pantalla. Si no están disponibles navegadores remotos en Unix, el proceso de control lanzará un nuevo navegador y esperará.

On iOS, the `BROWSER` environment variable, as well as any arguments controlling autoraise, browser preference, and new tab/window creation will be ignored. Web pages will *always* be opened in the user's preferred browser, in a new tab, with the browser being brought to the foreground. The use of the `webbrowser` module on iOS requires the `ctypes` module. If `ctypes` isn't available, calls to `open()` will fail.

Command-line interface

The script **webbrowser** can be used as a command-line interface for the module. It accepts a URL as the argument. It accepts the following optional parameters:

-n, --new-window

Opens the URL in a new browser window, if possible.

-t, --new-tab

Opens the URL in a new browser tab.

The options are, naturally, mutually exclusive. Usage example:

```
python -m webbrowser -t "https://www.python.org"
```

Availability: not WASI, not Android.

La siguiente excepción es definida:

exception webbrowser.Error

Excepción generada cuando ocurre un error de control de navegador.

Las siguientes funciones son definidas:

webbrowser.open(*url*, *new=0*, *autoraise=True*)

Muestra *url* usando el navegador por defecto. Si *new* es 0, se abre la *url* en la misma ventana del navegador si es posible. Si *new* es 1, se abre una nueva ventana del navegador si es posible. Si *new* es 2, se abre una nueva página del navegador («pestaña») si es posible. Si *autoraise* es `True`, la ventana es lanzada si es posible (ten en cuenta que bajo muchos gestores de ventana esto ocurrirá independientemente de la configuración de esta variable).

Returns `True` if a browser was successfully launched, `False` otherwise.

Ten en cuenta que en algunas plataformas, tratar de abrir un nombre de archivo usando esta función puede funcionar e iniciar el programa asociado del sistema operativo. Sin embargo, esto no es soportado ni portable.

Lanza un evento de auditoría `webbrowser.open` con el argumento `url`.

webbrowser.open_new(*url*)

Abre *url* en una nueva ventana del navegador por defecto, si es posible, si no, abre *url* en la única ventana del navegador.

Returns `True` if a browser was successfully launched, `False` otherwise.

webbrowser.open_new_tab(*url*)

Abre *url* en una nueva página («pestaña») del navegador por defecto, si es posible, si no equivale a **`open_new()`**.

Returns `True` if a browser was successfully launched, `False` otherwise.

webbrowser.get(*using=None*)

Retorna un objeto de controlador para el tipo de navegador *using*. Si *using* es `None`, retorna un controlador de un navegador por defecto apropiado para el entorno del invocador.

webbrowser.register(*name*, *constructor*, *instance=None*, ***, *preferred=False*)

Registra el tipo de navegador *name*. Una vez que el tipo de navegador es registrado, la función `get()` puede retornar un controlador para ese tipo de navegador. Si no se provee *instance* o es `None`, *constructor* será invocado sin parámetros al crear una instancia cuando sea necesario. Si se provee *instance*, *constructor* no será nunca invocado y puede ser `None`.

Definir *preferred* a `True` hace de este navegador un resultado preferido para una invocación `get()` sin argumento. De otra manera, este punto de entrada sólo es útil si planeas definir la variable **BROWSER** o invocar `get()` con un argumento no vacío correspondiendo con el nombre de un manejador que declaras.

Distinto en la versión 3.7: fue añadido el parámetro sólo de palabra clave *preferred*.

Un número de tipos de navegador son predefinidos. Esta tabla muestra los nombres de los tipos que se pueden pasar a la función `get()` y las instanciaciones correspondientes para las clases de los controladores, todas definidas en este módulo.

Nombre de tipo	Nombre de clase	Nota
`'mozilla'`	`Mozilla('mozilla')`	
`'firefox'`	`Mozilla('mozilla')`	
`'epiphany'`	`Epiphany('epiphany')`	
`'kfmclient'`	`Konqueror()`	(1)
`'konqueror'`	`Konqueror()`	(1)
`'kfm'`	`Konqueror()`	(1)
`'opera'`	`Opera()`	
`'links'`	`GenericBrowser('links')`	
`'elinks'`	`Elinks('elinks')`	
`'lynx'`	`GenericBrowser('lynx')`	
`'w3m'`	`GenericBrowser('w3m')`	
`'windows-default'`	`WindowsDefault`	(2)
`'macosx'`	`MacOSXOSAScript('default')`	(3)
`'safari'`	`MacOSXOSAScript('safari')`	(3)

`'google-chrome'`	`Chrome('google-chrome')`	
`'chrome'`	`Chrome('chrome')`	
`'chromium'`	`Chromium('chromium')`	
`'chromium-browser'`	`Chromium('chromium-browser')`	
`'iosbrowser'`	`IOSBrowser`	(4)

Notas:

1. «Konqueror» is the file manager for the KDE desktop environment for Unix, and only makes sense to use if KDE is running. Some way of reliably detecting KDE would be nice; the **`KDEDIR`** variable is not sufficient. Note also that the name «kfm» is used even when using the **konqueror** command with KDE 2 — the implementation selects the best strategy for running Konqueror.
2. Sólo en plataformas Windows.
3. Only on macOS.
4. Only on iOS.

Added in version 3.2: A new **`MacOSXOSAScript`** class has been added and is used on Mac instead of the previous **`MacOSX`** class. This adds support for opening browsers not currently set as the OS default.

Added in version 3.3: Ha sido añadido soporte para Chrome/ Chromium.

Distinto en la versión 3.12: Se ha eliminado el soporte para varios navegadores obsoletos. Los navegadores eliminados son Grail, Mosaic, Netscape, Galeon, Skipstone, Iceape y Firefox versión 35 e inferiores.

Distinto en la versión 3.13: Support for iOS has been added.

Aquí están algunos ejemplos simples:

```
url = 'https://docs.python.org/'

# Open URL in a new tab, if a browser window is already
webbrowser.open_new_tab(url)
```

```
# Open URL in new window, raising the window if possible
webbrowser.open_new(url)
```

Browser controller objects

Browser controllers provide the `name` attribute, and the following three methods which parallel module-level convenience functions:

controller.name

Nombre dependiente del sistema para el navegador.

controller.open(*url*, *new = 0*, *autoraise = True*)

Despliega *url* usando el navegador manejado por este controlador. Si *new* es 1, se abre una nueva ventana del navegador si es posible. Si *new* es 2, se abre una nueva página del navegador («pestaña») si es posible.

controller.open_new(*url*)

Abre *url* en una nueva ventana del navegador manejado por este controlador, si es posible, si no, abre *url* en la única ventana del navegador. Alias `open_new()`.

controller.open_new_tab(*url*)

Abre *url* en una nueva página («pestaña») del navegador manejado por este controlador, si es posible, si no equivale a `open_new()`.

Notas al pie

Los ejecutables nombrados aquí sin una ruta completa serán buscados en los directorios dados en la variable de entorno **`PATH`**.

`wsgiref` — WSGI Utilities and Reference Implementation

Código fuente: Lib/wsgiref [https://github.com/python/cpython/tree/3.14/Lib/wsgiref]

Advertencia

`wsgiref` is a reference implementation and is not recommended for production. The module only implements basic security checks.

La Interfaz de Pasarela del Servidor Web, o *Web Server Gateway Interface* en inglés (WSGI), es una interfaz estándar entre el servidor web y aplicaciones web escritas en Python. Con una interfaz estándar es más sencillo usar una aplicación que soporte WSGI con diferentes servidores web.

Sólo los autores de servidores y frameworks web necesitan conocer cada detalle y caso límite del diseño WSGI. No es necesario conocer cada detalle de WSGI sólo para instalar o escribir una aplicación web usando un *framework* existente.

`wsgiref` is a reference implementation of the WSGI specification that can be used to add WSGI support to a web server or framework. It provides utilities for manipulating WSGI environment variables and response headers, base classes for implementing WSGI servers, a demo HTTP server that serves WSGI applications, types for static type checking, and a validation tool that checks WSGI servers and applications for conformance to the WSGI specification (**PEP 3333** [https://peps.python.org/pep-3333/]).

Vea wsgi.readthedocs.io [https://wsgi.readthedocs.io/] para más información sobre WSGI, así como enlaces a tutoriales y otros recursos.

`wsgiref.util` – WSGI environment utilities

Este módulo ofrece una variedad de funciones útiles para trabajar con entornos WSGI. Un entorno WSGI es un diccionario que contiene variables de la petición HTTP, descrito en **PEP 3333** [https://peps.python.org/pep-3333/]. Todas las funciones que aceptan un parámetro *environ* esperan un diccionario compatible con WSGI. Por favor, consulte la especificación detallada en **PEP 3333** [https://peps.python.org/pep-3333/], y los alias de tipos que pueden usarse en las anotaciones de tipo en `WSGIEnvironment`.

wsgiref.util.guess_scheme(*environ*)

Retorna una deducción del valor para `wsgi.url_scheme` que debería ser «http» o «https», buscando la variable de entorno `HTTPS` en el diccionario *environ*. El valor de retorno es una cadena.

Esta función es útil al crear un *gateway* que envuelve CGI o un protocolo similar como FastCGI. Habitualmente, los servidores que ofrecen estos protocolos incluyen una variable `HTTPS` con el valor «1», «yes», o «on» cuando reciben una petición vía SSL. Así, esta función retorna «https» si encuentra ese valor, o «http», en caso contrario.

wsgiref.util.request_uri(*environ*, *include_query=True*)

Retorna la URI completa de la petición, opcionalmente la cadena de consulta, usando el algoritmo encontrado en la sección «URL Reconstruction» de **PEP 3333** [https://peps.python.org/pep-3333/]. Si *include_query* es falso, la cadena de consulta no se incluye en la URI resultante.

wsgiref.util.application_uri(*environ*)

Similar a `request_uri()` excepto que se ignoran las variables `PATH_INFO` y `QUERY_STRING`. El resultado es la URI base del objeto de aplicación indicado en la petición.

wsgiref.util.shift_path_info(*environ*)

Desplaza un solo nombre de `PATH_INFO` a `SCRIPT_NAME` y retorna el nombre. Se modifica el diccionario *environ*, por lo que se deberá usar una copia si se quiere mantener `PATH_INFO` o `SCRIPT_NAME` intactos.

Si no quedan segmentos en `PATH_INFO`, retornará `None`.

Habitualmente, esta rutina se usa para procesar cada porción de la ruta del URI de la petición, por ejemplo, para usar la ruta como una serie de claves en un diccionario. Esta rutina modifica el entorno pasado para permitir invocar otra aplicación WSGI que esté localizada en la URI objetivo. Por ejemplo, si hay una aplicación WSGI en `/foo`, y la ruta es `/foo/bar/baz`, y la aplicación WSGI en `/foo` llama a **`shift_path_info()`**, recibirá la cadena «bar», y se actualizará el entorno para pasarlo a una aplicación WSGI en `/foo/bar`. Es decir, `SCRIPT_NAME` cambiará de `/foo` a `/foo/bar` y `PATH_INFO` cambiará de `/bar/baz` a `/baz`.

Cuando `PATH_INFO` es únicamente «/», esta rutina retornará una cadena vacía y añadirá una barra final a `SCRIPT_NAME`, incluso cuando normalmente los segmentos de ruta vacíos se ignoran y `SCRIPT_NAME` no acaba con una barra. Este comportamiento es intencional, para asegurar que una aplicación puede diferenciar entre URIs que acaban en `/x` de las que acaban en `/x/` cuando usan esta rutina para atravesar objetos.

wsgiref.util.setup_testing_defaults(*environ*)

Actualiza *environ* con valores por defecto triviales con propósito de prueba.

Esta rutina añade varios parámetros requeridos por WSGI, incluyendo `HTTP_POST`, `SERVER_NAME`, `SERVER_PORT`, `REQUEST_METHOD`, `SCRIPT_NAME`, `PATH_INFO`, y todas las variables `wsgi.*` definidas en **PEP 3333** [https://peps.python.org/pep-3333/]. Sólo ofrece valores por defecto y no reemplaza ningún valor existente en esas variables.

Esta rutina pretende facilitar la preparación de entornos ficticios para pruebas unitarias de servidores y aplicaciones WSGI. NO debería usarse en servidores y aplicaciones WSGI reales, ya que los valores son ficticios!

Example usage (see also **`demo_app()`** for another example):

```
from wsgiref.util import setup_testing_defaults
from wsgiref.simple_server import make_server

# A relatively simple WSGI application. It's going
# environment dictionary after being updated by set
def simple_app(environ, start_response):
    setup_testing_defaults(environ)

    status = '200 OK'
    headers = [('Content-type', 'text/plain; charse

    start_response(status, headers)

    ret = [("%s: %s\n" % (key, value)).encode("utf-
           for key, value in environ.items()]
    return ret

with make_server('', 8000, simple_app) as httpd:
    print("Serving on port 8000...")
    httpd.serve_forever()
```

In addition to the environment functions above, the **`wsgiref.util`** module also provides these miscellaneous utilities:

wsgiref.util.is_hop_by_hop(*header_name*)

Retorna `True` si "header_name" es una cabecera «Hop-by-Hop» HTTP/1.1, como se define en **RFC 2616** [https://datatracker.ietf.org/doc/html/rfc2616.html].

class wsgiref.util.FileWrapper(*filelike*, *blksize=8192*)

Una implementación concreta del protocolo `wsgiref.types.FileWrapper` usado para convertir objetos de tipo archivo en un iterator. Los objetos resultantes son iterables. A medida que se itera sobre el objeto, el parámetro opcional *blksize* se pasará repetidamente al método **`read()`** del objeto archivo para obtener cadenas de bytes para entregar. Cuando **`read()`** retorna una cadena de bytes vacía, la iteración finalizará y no se podrá reiniciar.

Si *filelike* tiene un método **`close()`**, el objeto retornado también tendrá un método **`close()`** que llamará al método **`close()`** del objeto archivo subyacente.

Ejemplo de uso:

```
from io import StringIO
from wsgiref.util import FileWrapper

# We're using a StringIO-buffer for as the file-lik
filelike = StringIO("This is an example file-like o
wrapper = FileWrapper(filelike, blksize=5)

for chunk in wrapper:
    print(chunk)
```

Distinto en la versión 3.11: Support for `__getitem__()` method has been removed.

`wsgiref.headers` – WSGI response header tools

Este módulo ofrece una sola clase, `Headers`, para la manipulación de cabeceras de respuesta WSGI usando un interfaz de mapa.

class wsgiref.headers.Headers([*headers*])

Crea un objeto con interfaz de mapa envolviendo *headers*, que debe ser una lista de tuplas nombre/valor de las cabeceras, como se describe en **PEP 3333** [https://peps.python.org/pep-3333/]. El valor por defecto de *headers* es una lista vacía.

`Headers` objects support typical mapping operations including `__getitem__()`, `get()`, `__setitem__()`, `setdefault()`, `__delitem__()` and `__contains__()`. For each of these methods, the key is the header name (treated case-insensitively), and the value is the first value associated with that header name. Setting a header deletes any existing values for that header, then adds a new value at the end of the wrapped header list. Headers” existing order is generally maintained, with new headers added to the end of the wrapped list.

A diferencia de un diccionario, los objetos `Headers` no lanzan un error cuando se intenta obtener o eliminar una clave que no está en la lista de cabeceras subyacente. Al intentar obtener una clave inexistente simplemente se retornará `None`, mientras que el intento de eliminación de una cabecera inexistente simplemente no tendrá ningún efecto.

Los objetos `Headers` también soportan los métodos **`keys()`**, **`values()`**, y **`items()`**. Las listas retornadas por **`keys()`** y **`items()`** pueden incluir la misma clave más de una vez si se trata de una cabecera de valor múltiple. La `len()` de un objeto **`Headers`** es la misma que la longitud de sus **`items()`**, que es la misma que la longitud de la lista de cabeceras envuelta. De hecho, el método **`items()`** simplemente retorna una copia de la lista de cabeceras.

La llamada `bytes()` sobre un objeto `Headers` retorna una cadena de bytes formateada y lista para su transmisión como cabeceras de respuesta HTTP. Cada cabecera se ubica en una línea con su valor separado por dos puntos y un espacio. Cada línea finaliza con un retorno de carro y un salto de línea, y la cadena de bytes finaliza con una línea en blanco.

Además de la interfaz de mapa y las funcionalidades de formateado, los objetos `Headers` también ofrecen los siguientes métodos para consultar y añadir cabeceras con múltiples valores, y para añadir cabeceras con parámetros MIME:

get_all(*name*)

Retorna una lista de todos los valores para la cabecera indicada.

La lista retornada tendrá los valores ordenados según la lista de cabeceras original o según se hayan añadido a esta instancia, y podrá contener duplicados. Cualquier campo eliminado y añadido de nuevo estará al final de la lista. Si no existen campos con el nombre indicado, retornará una lista vacía.

add_header(*name, value, **_params*)

Añade una cabecera, posiblemente de valor múltiple, con los parámetros MIME opcionales especificados vía argumentos por palabra clave.

name es el nombre de la cabecera a añadir. Se pueden usar argumentos por palabra clave para establecer parámetros MIME para la cabecera. Cada parámetro debe ser una cadena o `None`. Todos los guiones bajos en los nombres de parámetros se convierten en guiones, dado que los guiones son inválidos en identificadores Python y muchos parámetros MIME incluyen guiones. Si el valor del parámetro es una cadena, se añade a los parámetros del valor de la cabecera con la forma `nombre=valor`. Si es `None`, sólo se añade el nombre del parámetro, para reflejar parámetros MIME sin valor. Ejemplo de uso:

```
h.add_header('content-disposition', 'attachment
```

El código anterior añadirá una cabecera como la siguiente:

```
Content-Disposition: attachment; filename="bud.
```

Distinto en la versión 3.5: El parámetro *headers* es opcional.

`wsgiref.simple_server` – a simple

WSGI HTTP server

Este módulo implementa un servidor HTTP simple, basado en **`http.server`**, que sirve aplicaciones WSGI. Cada instancia del servidor sirve una aplicación WSGI simple en una máquina y puerto dados. Si se quiere servir múltiples aplicaciones en una misma máquina y puerto, se deberá crear una aplicación WSGI que analiza `PATH_INFO` para seleccionar qué aplicación invocar para cada petición. Por ejemplo, usando la función **`shift_path_info()`** de **`wsgiref.util`**.

wsgiref.simple_server.make_server(*host*, *port*, *app*, *server_class*=*WSGIServer*, *handler_class*=*WSGIRequestHandler*)

> Crea un nuevo servidor WSGI que sirve en *host* y *port*, aceptando conexiones para *app*. El valor de retorno es una instancia de *server_class* y procesará peticiones usando *handler_class*. *app* debe ser un objeto aplicación WSGI, como se define en **PEP 3333** [https://peps.python.org/pep-3333/].
>
> Ejemplo de uso:

```
from wsgiref.simple_server import make_server, demo

with make_server('', 8000, demo_app) as httpd:
    print("Serving HTTP on port 8000...")

    # Respond to requests until process is killed
    httpd.serve_forever()

    # Alternative: serve one request, then exit
    httpd.handle_request()
```

wsgiref.simple_server.demo_app(*environ*, *start_response*)

> This function is a small but complete WSGI application that returns a text page containing the message «Hello world!» and a list of the key/value pairs provided in the *environ* parameter. It’s useful for verifying that a WSGI server (such as **`wsgiref.simple_server`**) is able to run a simple WSGI

application correctly.

The *start_response* callable should follow the `StartResponse` protocol.

class wsgiref.simple_server.WSGIServer(*server_address*, *RequestHandlerClass*)

Crea una instancia de `WSGIServer`. *server_address* debe ser una tupla `(máquina,puerto)` y *RequestHandlerClass* debe ser la subclase de `http.server.BaseHTTPRequestHandler` que se usará para procesar peticiones.

Normalmente, no es necesario invocar este constructor, ya que la función `make_server()` puede gestionar todos los detalles.

`WSGIServer` es una subclase de `http.server.HTTPServer`, por lo que todos sus métodos, como **`serve_forever()`** y **`handle_request()`**, están disponibles. **`WSGIServer`** también ofrece los siguientes métodos específicos de WSGI:

set_app(*application*)

Establece el invocable *application* como la aplicación WSGI que recibirá las peticiones.

get_app()

Retorna la aplicación invocable actualmente establecida.

Habitualmente, sin embargo, no es necesario usar estos métodos adicionales, ya que `set_app()` se llama desde `make_server()` y `get_app()` existe sobretodo para el beneficio de instancias del gestor de peticiones.

class wsgiref.simple_server.WSGIRequestHandler(*request*, *client_address*, *server*)

Crea un gestor HTTP para la *request* indicada (es decir un socket), *client_address (una tupla ``(máquina,puerto)``), y *server* (una instancia `WSGIServer`).

No es necesario crear instancias de esta clase directamente. Se crean automáticamente bajo demanda por objetos `WSGIServer`. Sin embargo, se pueden crear subclases de esta clase y proveerlas como *handler_class* a la función `make_server()`. Algunos métodos posiblemente relevantes para sobreescribir en estas subclases:

get_environ()

> Retorna un diccionario `WSGIEnvironment` para una petición. La implementación por defecto copia el contenido del diccionario atributo **`base_environ`** del objeto **`WSGIserver`** y añade varias cabeceras derivadas de la petición HTTP. Cada llamada a este método debe retornar un nuevo diccionario con todas las variables de entorno CGI relevantes especificadas en **PEP 3333** [https://peps.python.org/pep-3333/].

get_stderr()

> Retorna el objeto que debe usarse como el flujo de `wsgi.errors`. La implementación por defecto retorna simplemente `sys.stderr`.

handle()

> Procesa la petición HTTP. La implementación por defecto crea una instancia gestora usando una clase `wsgiref.handlers` para implementar la interfaz de aplicación WSGI real.

`wsgiref.validate` — WSGI conformance checker

When creating new WSGI application objects, frameworks, servers, or middleware, it can be useful to validate the new code's conformance using **`wsgiref.validate`**. This module provides a function that creates WSGI application objects that validate communications between a WSGI server or gateway and a WSGI application object, to check both sides for protocol conformance.

Hay que observar que esta utilidad no garantiza compatibilidad completa con **PEP 3333** [https://peps.python.org/pep-3333/]. La ausencia de errores usando este módulo no implica que no existan errores. Sin embargo, si este módulo produce errores, implica que o el servidor o la aplicación no son 100% compatibles.

Este módulo se basa en el módulo **`paste.lint`** de la librería *Python Paste* de *Ian Bicking*.

wsgiref.validate.validator(*application*)

Envuelve *application* y retorna un nuevo objeto de aplicación WSGI. La aplicación retornada reenviará todas las peticiones a la *application* original y comprobará que tanto la *application* como el servidor que la llama son compatibles con la especificación WSGI y con **RFC 2616** [https://datatracker.ietf.org/doc/html/rfc2616.html].

Cualquier incompatibilidad detectada provocará el lanzamiento de un **`AssertionError`**. Nótese que, sin embargo, cómo se gestionan estos errores depende del servidor. Por ejemplo, **`wsgiref.simple_server`** y otros servidores basados en **`wsgiref.handlers`**, que no sobreescriben los métodos de gestión de errores para hacer otras cosas, simplemente escribirán un mensaje de que el error ha ocurrido y volcarán la traza de error en `sys.stderr` o algún otro flujo de errores.

Este *wrapper* puede también generar salidas usando el módulo **`warnings`** para señalar comportamientos que son cuestionables pero que no están realmente prohibidos por **PEP 3333** [https://peps.python.org/pep-3333/]. A no ser que se suprima esta salida usando opciones de línea de comandos de Python o la API de **`warnings`**, cualquier aviso se escribirá en `sys.stderr`, en lugar de en `wsgi.errors`, aunque sean el mismo objeto.

Ejemplo de uso:

```
from wsgiref.validate import validator
from wsgiref.simple_server import make_server

# Our callable object which is intentionally not co
# standard, so the validator is going to break
def simple_app(environ, start_response):
    status = '200 OK'  # HTTP Status
    headers = [('Content-type', 'text/plain')]  # H
    start_response(status, headers)

    # This is going to break because we need to ret
    # the validator is going to inform us
    return b"Hello World"

# This is the application wrapped in a validator
validator_app = validator(simple_app)

with make_server('', 8000, validator_app) as httpd:
    print("Listening on port 8000....")
    httpd.serve_forever()
```

`wsgiref.handlers` – server/gateway base classes

Este módulo ofrece clases gestoras base para implementar servidores y *gateways* WSGI. Estas clases base gestionan la mayoría del trabajo de comunicarse con una aplicación WSGI, siempre que se les dé un entorno CGI, junto con una entrada, una salida, y un flujo de errores.

class wsgiref.handlers.CGIHandler

Invocación basada en CGI vía `sys.stdin`, `sys.stdout`, `sys.stderr` y `os.environ`. Esto es útil cuando se quiere ejecutar una aplicación WSGI como un script CGI. Simplemente es necesario invocar `CGIHandler().run(app)`, donde `app` es el objeto de aplicación WSGI que se quiere invocar.

Esta clase es una subclase de **`BaseCGIHandler`** que establece `wsgi.run_once` a cierto, `wsgi.multithread` a falso, y `wsgi.multiprocess` a cierto, y siempre usa **`sys`** y **`os`** para obtener los flujos y entorno CGI necesarios.

class wsgiref.handlers.IISCGIHandler

Una alternativa especializada a **`CGIHandler`**, para usar al desplegar en servidores web Microsoft IIS, sin establecer la opción de configuración `allowPathInfo` (IIS>=7) o la meta-base `allowPathInfoForScriptMappings` (IIS<7).

Por defecto, IIS entrega `PATH_INFO` que duplica `SCRIPT_NAME` al principio, causando problemas con aplicaciones WSGI que implementan enrutamiento. Este gestor elimina las partes duplicadas de la ruta.

IIS se puede configurar para pasar el `PATH_INFO` correcto, pero esto causa otro error donde `PATH_TRANSLATED` es incorrecto. Afortunadamente, esta variable rara vez se usa y no está garantizada por WSGI. Sin embargo, en IIS<7, la configuración solo se puede realizar en un nivel de vhost, lo que afecta a todas las demás asignaciones de scripts, muchas de las cuales se rompen cuando se exponen al error `PATH_TRANSLATED`. Por esta razón, IIS<7 casi nunca se implementa con la solución (incluso IIS7 rara vez lo usa porque todavía no tiene una interfaz de usuario).

No hay forma de que el código CGI detecte cuándo la opción está activada, por lo que se ofrece una clase gestora separada. Se usa de la misma forma que **`CGIHandler`**, p.e. llamando a `IISCGIHandler().run(app)`, donde `app` es el objeto aplicación WSGI que se desea invocar.

Added in version 3.2.

class wsgiref.handlers.BaseCGIHandler(*stdin*, *stdout*, *stderr*, *environ*, *multithread=True*, *multiprocess=False*)

Similar a **`CGIHandler`**, pero en lugar de usar los módulos **`sys`** y **`os`**, el entorno CGI y los flujos de E/S se especifican explícitamente. Los valores *multithread* y *multiprocess* se pasan a las aplicaciones ejecutadas por la instancia de gestión como `wsgi.multithread` y `wsgi.multiprocess`.

Esta clase es una subclase de **`SimpleHandler`** para usarse con servidores HTTP que no reciben peticiones directas de Internet, *HTTP origin servers*. Al escribir una implementación de un protocolo de pasarela, como CGI, FastCGI, SCGI, etcétera, que use una cabecera `Status:` para enviar un estado HTTP, probablemente sea adecuado heredar de esta clase, en lugar de **`SimpleHandler`**.

class wsgiref.handlers.SimpleHandler(*stdin*, *stdout*, *stderr*, *environ*, *multithread*=*True*, *multiprocess*=*False*)

Similar a **`BaseCGIHandler`** pero diseñada para usarse con servidores que reciban peticiones directamente de Internet, o *HTTP origin servers*. Al escribir una implementación de un servidor HTTP, probablemente prefiera heredar de esta clase en lugar de **`BaseCGIHandler`**.

This class is a subclass of **`BaseHandler`**. It overrides the **`__init__()`**, **`get_stdin()`**, **`get_stderr()`**, **`add_cgi_vars()`**, **`_write()`**, and **`_flush()`** methods to support explicitly setting the environment and streams via the constructor. The supplied environment and streams are stored in the **`stdin`**, **`stdout`**, **`stderr`**, and **`environ`** attributes.

El método **`write()`** de *stdout* podría escribir cada fragmento de información completamente, como **`io.BufferedIOBase`**.

class wsgiref.handlers.BaseHandler

Esta es una clase base abstracta para ejecutar aplicaciones WSGI. Cada instancia gestionará una sola petición HTTP, aunque en principio se podría crear una subclase reutilizable para múltiples peticiones.

Las instancias de **`BaseHandler`** tiene un sólo método para uso externo:

run(*app*)

Ejecuta la aplicación WSGI *app* indicada.

Todos los otros métodos de `BaseHandler` se invocan por este método en el proceso de ejecutar la aplicación, es decir, existen principalmente para permitir personalizar el proceso.

Los métodos siguientes DEBEN sobrescribirse en una subclase:

_write(*data*)

Enviar los bytes *data* para la transmisión al cliente. Es correcto que este método realmente transmita la información. La clase `BaseHandler` simplemente separa las operaciones de escritura y liberación para una mayor eficiencia cuando el sistema subyacente realmente hace esa distinción.

_flush()

Forzar la transmisión de los datos en el búfer al cliente. Es correcto si este método no es operativo, p.e., si `_write()` realmente envía los datos.

get_stdin()

Retorna un objeto compatible con `InputStream`, apropiado para usar como `wsgi.input` de la petición en proceso actualmente.

get_stderr()

Retorna un objeto compatible con `ErrorStream`, apropiado para usar como `wsgi.errors` de la petición en proceso actualmente.

add_cgi_vars()

Inserta las variables CGI para la petición actual en el atributo `environ`.

A continuación se describen algunos métodos y atributos más que podría ser interesante sobrescribir. Esta lista es sólo un sumario, sin embargo, y no incluye cada método que puede ser sobrescrito. Conviene consultar las docstrings y el código fuente para obtener información adicional antes de intentar crear una subclase de **`BaseHandler`** personalizada.

Atributos y métodos para personalizar el entorno WSGI:

wsgi_multithread

> El valor a usar en la variable de entorno `wsgi.multithread`. Por defecto es cierto en **`BaseHandler`**, pero puede tener un valor por defecto distinto (o establecerse en el constructor) en otras subclases.

wsgi_multiprocess

> El valor a usar en la variable de entorno `wsgi.multiprocess`. Por defecto es cierto en **`BaseHandler`**, pero puede tener un valor por defecto distinto (o establecerse en el constructor) en otras subclases.

wsgi_run_once

> El valor a usar en la variable de entorno `wsgi.run_once`. Por defecto es falso en **`BaseHandler`**, pero en **`CGIHandler`** es cierto por defecto.

os_environ

> The default environment variables to be included in every request’s WSGI environment. By default, this is a copy of `os.environ` at the time that **`wsgiref.handlers`** was imported, but subclasses can either create their own at the class or instance level. Note that the dictionary should be considered read-only, since the default value is shared between multiple classes and instances.

server_software

Si el atributo **`origin_server`** tiene valor, éste se usa para establecer el valor por defecto de la variable de entorno WSGI `SERVER_SOFTWARE` y un cabecera `Server:` por defecto en las respuestas HTTP. Las clases gestoras que no son *HTTP origin servers*, como **`BaseCGIHandler`** y **`CGIHandler`**, ignoran este atributo.

Distinto en la versión 3.3: El término «Python» se reemplaza con el término específico correspondiente a la implementación del intérprete, como «CPython», «Jython», etc.

get_scheme()

Retorna el esquema usado en la URL para la petición actual. La implementación por defecto utiliza la función **`guess_scheme()`** de **`wsgiref.util`** para adivinar si el esquema debería ser «http» o «https», basándose en las variables de **`environ`** de la petición actual.

setup_environ()

Establece el atributo **`environ`** a un entorno WSGI completo. La implementación por defecto utiliza todos los métodos y atributos anteriormente mencionados, más los métodos **`get_stdin()`**, **`get_stderr()`** y **`add_cgi_vars()`**, y el atributo **`wsgi_file_wrapper`**. También incluye una clave `SERVER_SOFTWARE` si no existe, siempre y cuando el atributo **`origin_server`** tiene un valor válido y el atributo **`server_software`** está establecido.

Métodos y atributos para personalizar el manejo de excepciones:

log_exception(*exc_info*)

Envía la tupla *exc_info* al registro del servidor. *exc_info* es un tupla `(type, value, traceback)`. La implementación por defecto simplemente escribe el seguimiento de la pila en el flujo `wsgi.errors` de la petición y lo vacía. Las subclases pueden sobrescribir éste método para cambiar el formato o redirigir la salida, enviar mensajes de correo con el seguimiento de pila a un administrador o cualquier otra acción que se considere adecuada.

traceback_limit

El máximo número de marcos a incluir en la salida de seguimientos de pilas por el método por defecto **`log_exception()`**. Si vale `None`, se incluyen todos los marcos.

error_output(*environ*, *start_response*)

Este método es una aplicación WSGI para generar una página de error para el usuario. Sólo se invoca si un error ocurre antes de enviar las cabeceras al cliente.

This method can access the current error using `sys.exception()`, and should pass that information to *start_response* when calling it (as described in the «Error Handling» section of **PEP 3333** [https://peps.python.org/pep-3333/]). In particular, the *start_response* callable should follow the **`StartResponse`** protocol.

La implementación por defecto sólo utiliza los atributos **`error_status`**, **`error_headers`** y **`error_body`** para generar una página de salida. Las subclases pueden sobrescribir éste para producir una salida de error dinámica mejor.

Hay que tener en cuenta, sin embargo, que no se recomienda, desde una perspectiva de seguridad, mostrar información de diagnóstico a cualquier usuario antiguo. Idealmente, se debería hacer algo especial para activar la salida de información de diagnóstico, motivo por el que la implementación por defecto no incluye ninguna.

error_status

El estado HTTP utilizado para las respuestas de error. Se debería utilizar una de las cadenas de estado definidas en **PEP 3333** [https://peps.python.org/pep-3333/]. Por defecto es un código 500 y un mensaje.

error_headers

Las cabeceras HTTP utilizadas por las respuestas de error. Debería tratarse de una lista de cabeceras de respuesta WSGI (tuplas `(name, value)`), tal y como se describe en **PEP 3333** [https://peps.python.org/pep-3333/]. La lista por defecto simplemente establece el tipo de contenido a `text/plain`.

error_body

El cuerpo de la respuesta de error. Debería ser una cadena de bytes con el cuerpo de la respuesta HTTP. Por defecto contiene el texto plano *A server error occurred. Please contact the administrator.*

Métodos y atributos para la funcionalidad «Optional Platform-Specific File Handling» de **PEP 3333** [https://peps.python.org/pep-3333/]:

wsgi_file_wrapper

Una fábrica `wsgi.file_wrapper` compatible con **`wsgiref.types.FileWrapper`**, o `None`. El valor por defecto de este atributo es la clase **`wsgiref.util.FileWrapper`**.

sendfile()

Sobrescribir para implementar la transmisión de ficheros específica para la plataforma. Este método se llama sólo si el valor de retorno de la aplicación es una instancia de la clase especificada por el atributo **`wsgi_file_wrapper`**. Debería retornar un valor cierto si fue capaz de transmitir correctamente el fichero, de modo que el código por defecto de transmisión no será ejecutado. La implementación por defecto de este método simplemente retorna un valor falso.

Métodos y atributos varios:

origin_server

Este atributo debería establecerse a cierto si los métodos **`_write()`** y **`_flush()`** están siendo usados para comunicar directamente al cliente, en lugar de usar un protocolo de pasarela CGI que requiere el estado HTTP en una cabecera `Status:` especial.

El valor por defecto de este atributo es cierto en **`BaseHandler`**, pero en **`BaseCGIHandler`** y **`CGIHandler`** es falso.

http_version

Si **`origin_server`** es cierto, este atributo de tipo cadena se usa para establecer la versión HTTP de la respuesta enviada al cliente. Por defecto es `"1.0"`.

wsgiref.handlers.read_environ()

Transcodifica las variables CGI de `os.environ` a cadenas «bytes in unicode» definidas en **PEP 3333** [https://peps.python.org/pep-3333/], retornando un nuevo diccionario. Esta función se usa en **`CGIHandler`** y **`IISCGIHandler`** en lugar de usar directamente `os.environ`, lo que no es necesariamente compatible con EWGI en todas las plataformas y servidores web usando Python 3 – específicamente, aquellas en las que el entorno actual del sistema operativo es Unicode, p.e. Windows, o en las que el entorno está en bytes, pero la codificación del sistema usada por Python para descodificar lo es cualquier otro que ISO-8859-1, por ejemplo, sistemas UNIX que usen UTF-8.

Cuando se está implementando un gestor basado en CGI propio, probablemente se requiera usar esta rutina en lugar de sólo copiar directamente los valores de `os.environ`.

Added in version 3.2.

`wsgiref.types` – WSGI types for static type checking

Este módulo provee varios tipos para validadores estáticos de tipos, como se describe en **PEP 3333** [https://peps.python.org/pep-3333/].

Added in version 3.11.

class wsgiref.types.StartResponse

A **`typing.Protocol`** describing **start_response()** [https://peps.python.org/pep-3333/#the-start-response-callable] callables (**PEP 3333** [https://peps.python.org/pep-3333/]).

wsgiref.types.WSGIEnvironment

Un alias de tipo que describe un diccionario de entorno WSGI.

wsgiref.types.WSGIApplication

Un alias de tipo que describe una aplicación WSGI invocable.

class wsgiref.types.InputStream

A **`typing.Protocol`** describing a **WSGI Input Stream** [https://peps.python.org/pep-3333/#input-and-error-streams].

class wsgiref.types.ErrorStream

A **`typing.Protocol`** describing a **WSGI Error Stream** [https://peps.python.org/pep-3333/#input-and-error-streams].

class wsgiref.types.FileWrapper

A **`typing.Protocol`** describing a **file wrapper** [https://peps.python.org/pep-3333/#optional-platform-specific-file-handling]. See **`wsgiref.util.FileWrapper`** for a concrete implementation of this protocol.

Ejemplos

This is a working «Hello World» WSGI application, where the *start_response* callable should follow the `StartResponse` protocol:

```
"""
Every WSGI application must have an application object -
object that accepts two arguments. For that purpose, we'
use a function (note that you're not limited to a functi
use a class for example). The first argument passed to t
is a dictionary containing CGI-style environment variabl
second variable is the callable object.
"""
from wsgiref.simple_server import make_server


def hello_world_app(environ, start_response):
    status = "200 OK"  # HTTP Status
    headers = [("Content-type", "text/plain; charset=utf
    start_response(status, headers)

    # The returned object is going to be printed
    return [b"Hello World"]

with make_server("", 8000, hello_world_app) as httpd:
    print("Serving on port 8000...")

    # Serve until process is killed
    httpd.serve_forever()
```

Ejemplo de una aplicación WSGI que sirve el directorio actual, acepta un directorio opcional y un número de puerto (default: 8000) en la línea de comandos:

```
"""
Small wsgiref based web server. Takes a path to serve fr
optional port number (defaults to 8000), then tries to s
MIME types are guessed from the file names, 404 errors a
if the file is not found.
```

```
"""
import mimetypes
import os
import sys
from wsgiref import simple_server, util


def app(environ, respond):
    # Get the file name and MIME type
    fn = os.path.join(path, environ["PATH_INFO"][1:])
    if "." not in fn.split(os.path.sep)[-1]:
        fn = os.path.join(fn, "index.html")
    mime_type = mimetypes.guess_file_type(fn)[0]

    # Return 200 OK if file exists, otherwise 404 Not Fo
    if os.path.exists(fn):
        respond("200 OK", [("Content-Type", mime_type)])
        return util.FileWrapper(open(fn, "rb"))
    else:
        respond("404 Not Found", [("Content-Type", "text
        return [b"not found"]


if __name__ == "__main__":
    # Get the path and port from command-line arguments
    path = sys.argv[1] if len(sys.argv) > 1 else os.getc
    port = int(sys.argv[2]) if len(sys.argv) > 2 else 80

    # Make and start the server until control-c
    httpd = simple_server.make_server("", port, app)
    print(f"Serving {path} on port {port}, control-C to
    try:
        httpd.serve_forever()
    except KeyboardInterrupt:
        print("Shutting down.")
        httpd.server_close()
```

`urllib` — URL handling modules

Código fuente: Lib/urllib/ [https://github.com/python/cpython/tree/3.14/Lib/urllib/]

`urllib` es un paquete que reúne varios módulos para trabajar con URLs:

- **`urllib.request`** para abrir y leer URLs
- **`urllib.error`** contiene las excepciones propuestas por **`urllib.request`**
- **`urllib.parse`** para parsear URLs
- **`urllib.robotparser`** para parsear archivos `robots.txt`

`urllib.request` — Extensible library for opening URLs

Código fuente: Lib/urllib/request.py [https://github.com/python/cpython/tree/3.14/Lib/urllib/request.py]

The `urllib.request` module defines functions and classes which help in opening URLs (mostly HTTP) in a complex world — basic and digest authentication, redirections, cookies and more.

Ver también

Se recomienda el paquete Requests [https://requests.readthedocs.io/en/master/] para una interfaz de cliente HTTP de mayor nivel.

Advertencia

On macOS it is unsafe to use this module in programs using `os.fork()` because the `getproxies()` implementation for macOS uses a higher-level system API. Set the environment variable `no_proxy` to `*` to avoid this problem (e.g. `os.environ["no_proxy"] = "*"`).

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

The `urllib.request` module defines the following functions:

urllib.request.urlopen(*url*, *data=None*, [*timeout*,]*, *context=None*)

Abre la *url*, la cual puede ser una cadena de texto que contenga una URL válida y correctamente codificada, o un objeto **`Request`**.

data debe ser un objeto que especifique datos adicionales a ser enviados al servidor o `None` si no se necesitan tales datos. Vea **`Request`** para más detalles.

El módulo urllib.request usa HTTP/1.1 e incluye el encabezado `Connection:close` en sus peticiones HTTP.

El parámetro opcional *timeout* especifica un tiempo de expiración en segundos para operaciones bloqueantes como el intento de conexión (si no se especifica, será usado el tiempo de expiración global predeterminado). Esto actualmente sólo funciona para conexiones HTTP, HTTPS y FTP.

Si se especifica *context*, debe ser una instancia **`ssl.SSLContext`** describiendo las diferentes opciones SSL. Vea **`HTTPSConnection`** para más detalles.

Esta función siempre retorna un objeto que puede actuar como un gestor de contexto, y tiene las propiedades *url*, *headers* y *status*. Véase **`urllib.response.addinfourl`** para más detalles sobre estas propiedades.

Para URLs HTTP y HTTPS, esta función retorna un objeto **`http.client.HTTPResponse`** ligeramente modificado. Adicionalmente a los tres nuevos métodos anteriores, el atributo msg contiene la misma información que el atributo **`reason`** — la frase de motivo devuelta por el servidor — en lugar de los encabezados de la respuesta como se especifica en la documentación para **`HTTPResponse`**.

For FTP, file, and data URLs, this function returns a **`urllib.response.addinfourl`** object.

Genera **`URLError`** en errores de protocolo.

Tenga en cuenta que `None` puede ser retornado si ningún manejador gestiona la petición (aunque el **`OpenerDirector`** global instalado de manera predeterminada usa **`UnknownHandler`** para asegurar que esto nunca suceda).

In addition, if proxy settings are detected (for example, when a `*_proxy` environment variable like **`http_proxy`** is set), **`ProxyHandler`** is default installed and makes sure the requests are handled through the proxy.

La función heredada de Python 2.6 y anteriores `urllib.urlopen` ha sido descontinuada, **`urllib.request.urlopen()`** corresponde a la antigua `urllib2.urlopen`. La gestión de proxy, la cual se hacía pasando un parámetro diccionario a `urllib.urlopen`, puede ser obtenida usando objetos **`ProxyHandler`**.

El abridor predeterminado genera un evento de auditoría `urllib.Request` con los argumentos `fullurl`, `data`, `headers`, `method` tomados del objeto de la petición.

Distinto en la versión 3.2: cafile y *capath* fueron añadidos.

Los hosts virtuales HTTPS ahora están soportados si es posible (esto es, si **`ssl.HAS_SNI`** es verdadero).

data puede ser un objeto iterable.

Distinto en la versión 3.3: cadefault fue añadido.

Distinto en la versión 3.4.3: context fue añadido.

Distinto en la versión 3.10: HTTPS connection now send an ALPN extension with protocol indicator `http/1.1` when no *context* is given. Custom *context* should set ALPN protocols with **`set_alpn_protocols()`**.

Distinto en la versión 3.13: Remove *cafile*, *capath* and *cadefault* parameters: use the *context* parameter instead.

urllib.request.install_opener(*opener*)

Instala una instancia **`OpenerDirector`** como el abridor global predeterminado. Instalar un abridor sólo es necesario si quieres que urlopen use ese abridor; si no, simplemente invoca `OpenerDirector.open()` en lugar de `urlopen()`. El código no comprueba por un **`OpenerDirector`** real y cualquier clase con la interfaz apropiada funcionará.

urllib.request.build_opener([*handler*, ...])

Retorna una instancia `OpenerDirector`, la cual encadena los manejadores en el orden dado. *handlers* pueden ser tanto instancias de `BaseHandler` o subclases de **`BaseHandler`** (en cuyo caso debe ser posible invocar el constructor sin ningún parámetro). Instancias de las siguientes clases estarán delante del *handlers*, a no ser que el *handlers* las contenga, instancias o subclases de ellas: `ProxyHandler` (si son detectadas configuraciones de proxy), `UnknownHandler`, `HTTPHandler`, `HTTPDefaultErrorHandler`, `HTTPRedirectHandler`, `FTPHandler`, `FileHandler`, `HTTPErrorProcessor`.

Si la instalación de Python tiene soporte SSL (ej. si se puede importar el módulo `ssl`), también será añadida `HTTPSHandler`.

Una subclase de `BaseHandler` puede cambiar también su atributo **`handler_order`** para modificar su posición en la lista de manejadores.

urllib.request.pathname2url(*path*, ***, *add_scheme=False*)

Convert the given local path to a `file:` URL. This function uses `quote()` function to encode the path.

If *add_scheme* is false (the default), the return value omits the `file:` scheme prefix. Set *add_scheme* to true to return a complete URL.

This example shows the function being used on Windows:

```
>>> from urllib.request import pathname2url
>>> path = 'C:\\Program Files'
>>> pathname2url(path, add_scheme=True)
```

```
'file:///C:/Program%20Files'
```

Distinto en la versión 3.14: Windows drive letters are no longer converted to uppercase, and `:` characters not following a drive letter no longer cause an **`OSError`** exception to be raised on Windows.

Distinto en la versión 3.14: Paths beginning with a slash are converted to URLs with authority sections. For example, the path `/etc/hosts` is converted to the URL `///etc/hosts`.

Distinto en la versión 3.14: The *add_scheme* parameter was added.

urllib.request.url2pathname(*url*, ***, *require_scheme=False*, *resolve_host=False*)

Convert the given `file:` URL to a local path. This function uses **`unquote()`** to decode the URL.

If *require_scheme* is false (the default), the given value should omit a `file:` scheme prefix. If *require_scheme* is set to true, the given value should include the prefix; a **`URLError`** is raised if it doesn't.

The URL authority is discarded if it is empty, `localhost`, or the local hostname. Otherwise, if *resolve_host* is set to true, the authority is resolved using **`socket.gethostbyname()`** and discarded if it matches a local IP address (as per **RFC 8089 §3** [https://datatracker.ietf.org/doc/html/rfc8089.html#section-3]). If the authority is still unhandled, then on Windows a UNC path is returned, and on other platforms a **`URLError`** is raised.

This example shows the function being used on Windows:

```
>>> from urllib.request import url2pathname
>>> url = 'file:///C:/Program%20Files'
>>> url2pathname(url, require_scheme=True)
'C:\\Program Files'
```

Distinto en la versión 3.14: Windows drive letters are no longer

converted to uppercase, and `:` characters not following a drive letter no longer cause an **`OSError`** exception to be raised on Windows.

Distinto en la versión 3.14: The URL authority is discarded if it matches the local hostname. Otherwise, if the authority isn't empty or `localhost`, then on Windows a UNC path is returned (as before), and on other platforms a **`URLError`** is raised.

Distinto en la versión 3.14: The URL query and fragment components are discarded if present.

Distinto en la versión 3.14: The *require_scheme* and *resolve_host* parameters were added.

urllib.request.getproxies()

Esta función auxiliar retorna un diccionario de esquema para las asignaciones de URL del servidor proxy. Escanea el entorno en busca de variables denominadas `<scheme>_proxy`, tomando en cuenta diferencia entre mayúsculas y minúsculas, para todos los sistemas operativos primero, y cuando no pueden encontrarla, buscan información proxy desde la Configuración del Sistema de macOS y desde Registros del Sistema para Windows. Si existen variables de entorno tanto en mayúsculas como en minúsculas (y no concuerdan), las minúsculas son preferidas.

Nota

Si la variable del entorno `REQUEST_METHOD` está definida, lo cual usualmente indica que tu script está ejecutándose en un entorno CGI, la variable de entorno `HTTP_PROXY` (mayúsculas `_PROXY`) será ignorada. Esto es porque esa variable puede ser inyectada por un cliente usando el encabezado HTTP «Proxy:». Si necesitas usar un proxy HTTP en un entorno CGI, usa `ProxyHandler` explícitamente o asegúrate de que el nombre de la variable está en minúsculas (o al menos el sufijo `_proxy`).

Se proveen las siguientes clases:

class urllib.request.Request(*url*, *data=None*, *headers={}*, *origin_req_host=None*, *unverifiable=False*, *method=None*)

Esta clase es un abstracción de una petición URL.

url debe ser una cadena de texto que contenga una URL válida y correctamente codificada.

data debe ser un objeto que especifique datos adicionales a enviar al servidor o `None` si no se necesitan tales datos. Actualmente las peticiones HTTP son las únicas que usan *data*. Los tipos de objetos soportados incluyen bytes, objetos como archivos e iterables de objetos como bytes. Si no se ha provisto el campo de encabezado `Content-Length` ni `Transfer-Encoding`, **`HTTPHandler`** establecerá estos encabezados de acuerdo al tipo de *data*. `Content-Length` será usado para enviar objetos de bytes, mientras `Transfer-Encoding: chunked` como se especifica en **RFC 7230** [https://datatracker.ietf.org/doc/html/rfc7230.html], Sección 3.3.1 será usado para enviar archivos y otros iterables.

Para un método de una petición HTTP POST, *data* debe ser un buffer en el formato estándar *application/x-www-form-urlencoded*. La función **`urllib.parse.urlencode()`** toma un mapeo o una secuencia de tuplas de dos valores y retorna una cadena de caracteres ASCII en este formato. Debe ser codificada a bytes antes de ser usada como el parámetro *data*.

headers debe ser un diccionario, y será tratado como si **`add_header()`** fuera invocado con cada clave y valor como argumentos. Esto es usado frecuentemente para «parodiar» el valor de encabezado `User-Agent`, el cual es usado por un navegador para identificarse a sí mismo – algunos servidores HTTP sólo permiten peticiones que vienen de navegadores comunes a diferencia de los scripts. Por ejemplo, Mozilla Firefox puede identificarse a sí mismo como `"Mozilla/5.0 (X11; U; Linux i686) Gecko/20071127 Firefox/2.0.0.11"`, mientras el agente de usuario predeterminado de **`urllib`** es `"Python-urllib/2.6"` (en

Python 2.6). Todas las claves de los encabezados se envían en camel case.

An appropriate `Content-Type` header should be included if the *data* argument is present. If this header has not been provided and *data* is not `None`, `Content-Type: application/x-www-form-urlencoded` will be added as a default.

Los siguientes dos argumentos sólo tienen interés para la gestión correcta de cookies HTTP de terceros:

origin_req_host debe ser el host de la petición de la transacción origen, como define **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html]. Por defecto es `http.cookiejar.request_host(self)`. Este es el nombre de host o la dirección IP de la petición original que fue iniciada por el usuario. Por ejemplo, si la petición es para una imagen en un documento HTML, debe ser el host de la petición para la página que contiene la imagen.

unverifiable debe indicar si la petición no es verificable, como define **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html]. Por defecto es `False`. Una petición no verificable es una cuya URL el usuario no tuvo opción de aprobar. Por ejemplo, si la petición es por una imagen en un documento HTML y el usuario no tuvo opción de aprobar la obtención automática de la imagen, este debe ser verdadero.

method should be a string that indicates the HTTP request method that will be used (e.g. `'HEAD'`). If provided, its value is stored in the **`method`** attribute and is used by **`get_method()`**. The default is `'GET'` if *data* is `None` or `'POST'` otherwise. Subclasses may indicate a different default method by setting the **`method`** attribute in the class itself.

Nota

La petición no funcionará como se espera si el objeto de datos es incapaz de entregar su contenido más de una vez (ej. un archivo o un iterable que puede producir el contenido sólo una vez) y la petición se reintentará para redirecciones HTTP o autenticación. El *data* es enviado al servidor HTTP directamente después de los encabezados. No hay soporte para una expectativa de funcionamiento 100% continuo en la biblioteca.

Distinto en la versión 3.3: El argumento **`Request.method`** es añadido a la clase Request.

Distinto en la versión 3.4: El atributo predeterminado **`Request.method`** puede ser indicado a nivel de clase.

Distinto en la versión 3.6: No se genera un error si el `Content-Length` no ha sido provisto y *data* no es `None` ni un objeto de bytes. En su lugar recurre a la codificación de transferencia fragmentada.

class urllib.request.OpenerDirector

La clase **`OpenerDirector`** abre URLs mediante la encadenación conjunta de **`BaseHandler`**. Este maneja el encadenamiento de manejadores y la recuperación de errores.

class urllib.request.BaseHandler

Esta es la clase base para todos los manejadores registrados — y manejan sólo las mecánicas simples del registro.

class urllib.request.HTTPDefaultErrorHandler

Una clase la cual define un manejador predeterminado para los errores de respuesta HTTP; todas las respuestas son convertidas en excepciones **`HTTPError`**.

class urllib.request.HTTPRedirectHandler

Una clase para manejar redirecciones.

class urllib.request.HTTPCookieProcessor(*cookiejar=None*)

Una clase para manejar Cookies HTTP.

class urllib.request.ProxyHandler(*proxies=None*)

Causa que las peticiones vayan a través de un proxy. Si se provee *proxies*, debe ser un diccionario mapeando nombres de protocolos a URLs de proxies. Por defecto lee la lista de proxies de las variables de entorno `<protocol>_proxy`. Si no se establecen variables de entorno de proxy, entonces se obtienen las configuraciones de proxy en un entorno Windows desde la sección del registro de Configuraciones de Internet y en un entorno macOS se obtiene la información de proxy desde el Framework de Configuración del Sistema.

Para deshabilitar la detección automática de proxy pasa un diccionario vacío.

La variable de entorno **`no_proxy`** puede ser usada para especificar hosts los cuales no deben ser alcanzados mediante proxy; si se establece, debe ser una lista separada por comas de sufijos de nombres de host, con `:port` añadidos opcionalmente, por ejemplo `cern.ch,ncsa.uiuc.edu,some.host:8080`.

Nota

`HTTP_PROXY` será ignorado si se establece una variable `REQUEST_METHOD`; vea la documentación de **`getproxies()`**.

class urllib.request.HTTPPasswordMgr

Mantiene una base de datos de mapeos `(realm, uri) -> (user, password)`.

class urllib.request.HTTPPasswordMgrWithDefaultRealm

Mantiene una base de datos de mapeos `(realm, uri) -> (user, password)`. Un reino de `None` se considera un reino caza todo, el cual es buscado si ningún otro reino encaja.

class urllib.request.HTTPPasswordMgrWithPriorAuth

Una variante de **`HTTPPasswordMgrWithDefaultRealm`** que también tiene una base de datos de mapeos `uri -> is_authenticated`. Puede ser usada por un manejador BasicAuth para determinar cuando enviar credenciales de autenticación inmediatamente en lugar de esperar primero a una respuesta `401`.

Added in version 3.5.

class urllib.request.AbstractBasicAuthHandler(*password_mgr=None*)

Esta es una clase mixin que ayuda con la autenticación HTTP, tanto al host remoto y a un proxy. Si se proporciona *password_mgr*, debe ser algo compatible con **`HTTPPasswordMgr`**; refiera a la sección Objetos HTTPPasswordMgr para información sobre la interfaz que debe ser soportada. Si *passwd_mgr* proporciona también métodos `is_authenticated` y `update_authenticated` (vea Objetos HTTPPasswordMgrWithPriorAuth), entonces el manejador usará el `is_authenticated` resultado para una URI dada para determinar el envío o no de credenciales de autenticación con la petición. Si `is_authenticated` retorna `True` para la URI, las peticiones subsecuentes a la URI o cualquiera de las super URIs incluirán automáticamente los credenciales de autenticación.

Added in version 3.5: Añadido soporte `is_authenticated`.

class urllib.request.HTTPBasicAuthHandler(*password_mgr=None*)

Administra autenticación con el host remoto. Si se proporciona *password_mgr*, debe ser compatible con **`HTTPPasswordMgr`**; refiera a la sección Objetos HTTPPasswordMgr para información sobre la interfaz que debe ser soportada. HTTPBasicAuthHandler lanzará un **`ValueError`** cuando se presente con un esquema de Autenticación incorrecto.

class urllib.request.ProxyBasicAuthHandler(*password_mgr=None*)

Administra autenticación con el proxy. Si se proporciona *password_mgr* debe ser compatible con **`HTTPPasswordMgr`**;

refiera a la sección Objetos HTTPPasswordMgr para información sobre la interfaz que debe ser soportada.

class urllib.request.AbstractDigestAuthHandler(*password_mgr=None*)

Esto es una clase mixin que ayuda con la autenticación HTTP, tanto al host remoto como a un proxy. Si se proporciona *password_mgr* debe ser compatible con **`HTTPPasswordMgr`**; refiera a la sección Objetos HTTPPasswordMgr para información sobre la interfaz que debe ser soportada.

Distinto en la versión 3.14: Added support for HTTP digest authentication algorithm `SHA-256`.

class urllib.request.HTTPDigestAuthHandler(*password_mgr=None*)

Maneja autenticación con el host remoto. Si se proporciona *password_mgr* debe ser compatible con **`HTTPPasswordMgr`**; refiera a la sección Objetos HTTPPasswordMgr para información sobre la interfaz que debe ser soportada. Cuando se añaden tanto el Manejador de Autenticación Digest (*Digest Authentication Handler*) como el Manejador de Autenticación Básico (*Basic Authentication Handler*) la Autenticación Digest siempre se intenta primero. Si la Autenticación Digest retorna una respuesta 40x de nuevo, se envía al controlador de Autenticación Básica para Manejar. Este método Handler lanzará un **`ValueError`** cuando sea presentado con un esquema de autenticación diferente a Digest o Básico.

Distinto en la versión 3.3: Genera **`ValueError`** en Esquema de Autenticación no soportado.

class urllib.request.ProxyDigestAuthHandler(*password_mgr=None*)

Administra autenticación con el proxy. Si se proporciona *password_mgr* debe ser compatible con **`HTTPPasswordMgr`**; refiera a la sección Objetos HTTPPasswordMgr para información sobre la interfaz que debe ser soportada.

class urllib.request.HTTPHandler

Una clase para gestionar apertura de URLs HTTP.

class urllib.request.HTTPSHandler(*debuglevel=0*, *context=None*, *check_hostname=None*)

Una clase para gestionar apertura de URLs HTTPS. *context* y *check_hostname* tienen el mismo significado que en `http.client.HTTPSConnection`.

Distinto en la versión 3.2: context y *check_hostname* fueron añadidos.

class urllib.request.FileHandler

Abre archivos locales.

class urllib.request.DataHandler

Abre URLs de datos.

Added in version 3.4.

class urllib.request.FTPHandler

Abre URLs FTP.

class urllib.request.CacheFTPHandler

Abre URLs FTP, manteniendo una caché de conexiones FTP abiertas para minimizar retrasos.

class urllib.request.UnknownHandler

Una clase caza todo para gestionar URLs desconocidas.

class urllib.request.HTTPErrorProcessor

Procesa errores de respuestas HTTP.

Objetos Request

Los siguientes métodos describen la interfaz pública de `Request` por lo que pueden ser sobrescritos en subclases. También define varios atributos públicos que pueden ser usado por clientes para inspeccionar la respuesta analizada.

Request.full_url

La URL original pasada al constructor.

Distinto en la versión 3.4.

Request.full_url es una propiedad con setter, getter y deleter. Obtener `full_url` retorna la petición URL original con el fragmento, si este estaba presente.

Request.type

El esquema de URI.

Request.host

La autoridad de URI, típicamente un host, pero también puede contener un puerto separado por un caracter de doble punto.

Request.origin_req_host

El host original de la petición, sin puerto.

Request.selector

La ruta de URI. Si `Request` usa un proxy, entonces selector será la URL completa que se pasa al proxy.

Request.data

El cuerpo de la entidad para la solicitud o `None` si no es especificado.

Distinto en la versión 3.4: Cambiar el valor de `Request.data` elimina ahora el encabezado «Content-Length» si fue establecido o calculado previamente.

Request.unverifiable

booleano, indica si la petición no es verificable como se define por **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html].

Request.method

El método de petición HTTP a usar. Por defecto su valor es `None`, lo que significa que `get_method()` realizará su cálculo normal del método a usar. Su valor puede ser definido (sobrescribiendo así el cálculo predeterminado en `get_method()`) tanto proporcionando un valor por defecto estableciéndolo a nivel de clase en una subclase de `Request` o pasando un valor al constructor de `Request` por medio del argumento *method*.

Added in version 3.3.

Distinto en la versión 3.4: Un valor predeterminado puede ser establecido ahora en subclases; previamente sólo podía ser definido mediante el argumento del constructor.

Request.get_method()

Retorna una cadena indicando el método de petición HTTP. Si `Request.method` no es `None`, retorna su valor, de otra forma retorna `'GET'` si `Request.data` es `None` o `'POST'` si no lo es. Esto sólo es significativo para peticiones HTTP.

Distinto en la versión 3.3: get_method ahora mira el valor de `Request.method`.

Request.add_header(*key*, *val*)

Añade otro encabezado a la petición. Los encabezados actualmente son ignorados por todos los manejadores excepto los manejadores HTTP, donde son añadidos a la lista de encabezados enviados al servidor. Tenga en cuenta que no puede haber más de un encabezado con el mismo nombre y las invocaciones posteriores sobrescribirán las invocaciones previas en caso de que *key* colisione. Actualmente, esto no es una pérdida de funcionalidad HTTP, ya que todos los encabezados que tienen sentido cuando son usados más de una vez tienen una manera (específica de encabezado) de ganar la misma funcionalidad usando sólo un encabezado. Tenga en cuenta que los encabezados agregados con este método también se agregan a las solicitudes redirigidas.

Request.add_unredirected_header(*key*, *header*)

Añade un encabezado que no será añadido a una petición redireccionada.

Request.has_header(*header*)

Retorna si la instancia tiene el encabezado nombrado (comprueba tanto regular como no redirigido).

Request.remove_header(*header*)

Elimina el encabezado nombrado de la instancia de la petición (desde encabezados regulares y no redireccionados).

Added in version 3.4.

Request.get_full_url()

Retorna la URL dada en el constructor.

Distinto en la versión 3.4.

Retorna `Request.full_url`

Request.set_proxy(*host*, *type*)

Prepara la petición conectando a un servidor proxy. Los *host* y *type* reemplazarán aquellos de la instancia y el selector de la instancia será la URL original dada en el constructor.

Request.get_header(*header_name*, *default=None*)

Retorna el valor del encabezado dado.

Request.header_items()

Retorna una lista de tuplas (header_name, header_value) de los encabezados de la Petición.

Distinto en la versión 3.4: Los métodos de petición add_data, has_data, get_data, get_type, get_host, get_selector, get_origin_req_host y is_unverifiable que quedaron obsoletos desde 3.3 han sido eliminados.

Objetos OpenerDirector

Las instancias de `OpenerDirector` tienen los siguientes métodos:

OpenerDirector.add_handler(*handler*)

handler debe ser una instancia de `BaseHandler`. Los siguientes métodos son buscados y añadidos a las cadenas posibles (tenga en cuenta que los errores HTTP son un caso espacial). Tenga en cuenta que, en los siguientes, *protocol* debe ser remplazado con el protocolo actual a manejar, por ejemplo `http_response()` sería el protocolo HTTP del manejador de respuesta. También *type* debe ser remplazado con el código HTTP actual, por ejemplo `http_error_404()` manejaría errores HTTP 404.

- **`<protocol>_open()`** — signal that the handler knows how to open *protocol* URLs.

  Vea `BaseHandler.<protocol>_open()` para más información.

- **`http_error_<type>()`** — signal that the handler knows how to handle HTTP errors with HTTP error code *type*.

  Vea `BaseHandler.http_error_<nnn>()` para más información.

- **`<protocol>_error()`** — signal that the handler knows how to handle errors from (non-`http`) *protocol*.

- **`<protocol>_request()`** — signal that the handler knows how to pre-process *protocol* requests.

  Vea `BaseHandler.<protocol>_request()` para más información.

- **`<protocol>_response()`** — signal that the handler knows how to post-process *protocol* responses.

  Vea `BaseHandler.<protocol>_response()` para más información.

OpenerDirector.open(*url*, *data=None*[, *timeout*])

Abre la *url* dada (la cual puede ser un objeto de petición o una cadena de caracteres), pasando opcionalmente el *data* dado. Los argumentos, los valores de retorno y las excepciones generadas son las mismas que aquellas de `urlopen()` (las cuales simplemente invocan el método `open()` en el `OpenerDirector` instalado global). El parámetro opcional *timeout* especifica un tiempo de expiración en segundos para operaciones bloqueantes como el intento de conexión (si no se especifica, el tiempo de expiración global será usado). La característica de tiempo de expiración actualmente funciona sólo para conexiones HTTP, HTTPS y FTP.

OpenerDirector.error(*proto*, **args*)

Handle an error of the given protocol. This will call the registered error handlers for the given protocol with the given arguments (which are protocol specific). The HTTP protocol is a special case which uses the HTTP response code to determine the specific error handler; refer to the **`http_error_<type>()`** methods of the handler classes.

Retorna si los valores y excepciones generadas son las mismas que aquellas de `urlopen()`.

Los objetos OpenerDirector abren URLs en tres etapas:

El orden en el cual esos métodos son invocados dentro de cada etapa es determinado ordenando las instancias manejadoras.

1. Every handler with a method named like **`<protocol>_request()`** has that method called to pre-process the request.

2. Handlers with a method named like `<protocol>_open()` are called to handle the request. This stage ends when a handler either returns a non-`None` value (ie. a response), or raises an exception (usually `URLError`). Exceptions are allowed to propagate.

   In fact, the above algorithm is first tried for methods named `default_open()`. If all such methods return `None`, the algorithm is repeated for methods named like `<protocol>_open()`. If all such methods return `None`, the algorithm is repeated for methods named `unknown_open()`.

   Tenga en cuenta que la implementación de esos métodos puede involucrar invocaciones de los métodos `open()` y `error()` de la instancia `OpenerDirector` padre.

3. Every handler with a method named like `<protocol>_response()` has that method called to post-process the response.

Objetos BaseHandler

Los objetos `BaseHandler` proporcionan un par de métodos que son útiles directamente y otros que están destinados a ser utilizados por clases derivadas. Estos están pensados para uso directo:

BaseHandler.add_parent(*director*)

Añade un director como padre.

BaseHandler.close()

Elimina cualquier padre.

El siguiente atributo y los siguientes métodos sólo deben ser usados por clases derivadas de `BaseHandler`.

Nota

The convention has been adopted that subclasses defining `<protocol>_request()` or `<protocol>_response()`

methods are named **`*Processor`**; all others are named **`*Handler`**.

BaseHandler.parent

Un `OpenerDirector` válido, el cual puede ser utilizado para abrir usando un protocolo diferente, o para manejar errores.

BaseHandler.default_open(*req*)

Este método no es definido en `BaseHandler`, pero las subclases deben definirlo si quieren cazar todas las URLs.

This method, if implemented, will be called by the parent `OpenerDirector`. It should return a file-like object as described in the return value of the `open()` method of **`OpenerDirector`**, or `None`. It should raise `URLError`, unless a truly exceptional thing happens (for example, `MemoryError` should not be mapped to **`URLError`**).

Este método será invocado antes de cualquier método de apertura específico de protocolo.

BaseHandler.<protocol>_open(req)

Este método no está definido en `BaseHandler`, pero las subclases deben definirlo si quieren manejar URLs con el protocolo dado.

This method, if defined, will be called by the parent `OpenerDirector`. Return values should be the same as for `default_open()`.

BaseHandler.unknown_open(*req*)

Este método *no* está definido en `BaseHandler`, pero las subclases deben definirlo si quieren cazar todas las URLs sin manejador registrado para abrirlo.

Este método, si está implementado, será invocado por el `parent` de `OpenerDirector`. Los valores retornados deben ser los mismos que para `default_open()`.

BaseHandler.http_error_default(*req*, *fp*, *code*, *msg*, *hdrs*)

Este método *no* está definido en `BaseHandler`, pero las subclases deben sobreescribirlo si pretenden proporcionar una solución general para los errores HTTP que de otro modo no se manejarían. Sería invocado automáticamente por el `OpenerDirector` obteniendo el error y no debe ser invocado normalmente en otras circunstancias.

`OpenerDirector` will call this method with five positional arguments:

1. a `Request` object,
2. a file-like object with the HTTP error body,
3. the three-digit code of the error, as a string,
4. the user-visible explanation of the code, as a string, and
5. the headers of the error, as a mapping object.

Los valores de retorno y las excepciones generadas deben ser los mismos que aquellos de `urlopen()`.

BaseHandler.http_error_<nnn>(req, fp, code, msg, hdrs)

nnn debe ser un código de error HTTP de tres dígitos. Este método tampoco está definido en `BaseHandler`, pero será invocado, si existe, en una instancia de una subclase, cuando ocurra un error HTTP con código *nnn*.

Las subclases deben sobrescribir este método para manejar errores HTTP específicos.

Arguments, return values and exceptions raised should be the same as for `http_error_default()`.

BaseHandler.<protocol>_request(req)

Este método *no* está definido en `BaseHandler`, pero las subclases deben definirlo si pretenden preprocesar peticiones del protocolo dado.

Este método, si está definido, será invocado por el `OpenerDirector` padre. *req* será un objeto `Request`. El valor retornado debe ser un objeto `Request`.

BaseHandler.<protocol>_response(req, response)

Este método *no* está definido en `BaseHandler`, pero las subclases deben definirlo si quieren postprocesar respuestas del protocolo dado.

Este método, si está definido, será invocado por el `OpenerDirector` padre. *req* será un objeto `Request`. *response* será un objeto que implementa la misma interfaz que el valor retornado de `urlopen()`. El valor retornado debe implementar la misma interfaz que el valor retornado de **`urlopen()`**.

Objetos HTTPRedirectHandler

Nota

Algunas redirecciones HTTP requieren acción desde el código del módulo del cliente. Si este es el caso, se genera `HTTPError`. Vea RFC 2616 [https://datatracker.ietf.org/doc/html/rfc2616.html] para más detalles de los significados precisos de los diferentes códigos de redirección.

An `HTTPError` exception raised as a security consideration if the HTTPRedirectHandler is presented with a redirected URL which is not an HTTP, HTTPS or FTP URL.

HTTPRedirectHandler.redirect_request(*req*, *fp*, *code*, *msg*, *hdrs*, *newurl*)

Return a `Request` or `None` in response to a redirect. This is called by the default implementations of the **`http_error_30*()`** methods when a redirection is received from the server. If a redirection should take place, return a new **`Request`** to allow **`http_error_30*()`** to perform the redirect to *newurl*. Otherwise, raise `HTTPError` if no other handler should try to handle this URL, or return `None` if you can't but another handler might.

Nota

La implementación predeterminada de este método no sigue estrictamente **RFC 2616** [https://datatracker.ietf.org/doc/html/rfc2616.html], la cual dice que las respuestas 301 y 302 a peticiones POST no deben ser redirigidas automáticamente sin confirmación por el usuario. En realidad, los navegadores permiten redirección automática de esas respuestas, cambiando el POST a un `GET` y la implementación predeterminada reproduce este comportamiento.

HTTPRedirectHandler.http_error_301(*req*, *fp*, *code*, *msg*, *hdrs*)

Redirecciona a la URL `Location:` o `URI:`. Este método es invocado por el **`OpenerDirector`** padre al obtener una respuesta HTTP "moved permanently".

HTTPRedirectHandler.http_error_302(*req*, *fp*, *code*, *msg*, *hdrs*)

Lo mismo que **`http_error_301()`**, pero invocado para la respuesta "found".

HTTPRedirectHandler.http_error_303(*req*, *fp*, *code*, *msg*, *hdrs*)

Lo mismo que **`http_error_301()`**, pero invocado para la respuesta "see other".

HTTPRedirectHandler.http_error_307(*req*, *fp*, *code*, *msg*, *hdrs*)

Lo mismo que **`http_error_301()`**, pero invocado para la respuesta "redirección temporal". No permite cambiar el método de solicitud de `POST` a `GET`.

HTTPRedirectHandler.http_error_308(*req*, *fp*, *code*, *msg*, *hdrs*)

Lo mismo que **`http_error_301()`**, pero invocado para la respuesta "redirección permanente". No permite cambiar el método de solicitud de `POST` a `GET`.

Added in version 3.11.

Objetos HTTPCookieProcessor

Las instancias `HTTPCookieProcessor` tienen un atributo:

HTTPCookieProcessor.cookiejar

El `http.cookiejar.CookieJar` en el cual las cookies están almacenadas.

Objetos ProxyHandler

ProxyHandler.<protocol>_open(request)

The `ProxyHandler` will have a method **`<protocol>_open()`** for every *protocol* which has a proxy in the *proxies* dictionary given in the constructor. The method will modify requests to go through the proxy, by calling `request.set_proxy()`, and call the next handler in the chain to actually execute the protocol.

Objetos HTTPPasswordMgr

Estos métodos están disponibles en los objetos `HTTPPasswordMgr` y `HTTPPasswordMgrWithDefaultRealm`.

HTTPPasswordMgr.add_password(*realm*, *uri*, *user*, *passwd*)

uri puede ser una única URI o una secuencia de URIs. *realm*, *user* y *passwd* deben ser cadenas. Esto causa que `(user, passwd)` se utilice como tokens de autenticación cuando la autenticación para *realm* y para una super URI de ninguna de las URIs dadas es provista.

HTTPPasswordMgr.find_user_password(*realm*, *authuri*)

Obtener usuario/contraseña para el reino y URI dados, si alguno ha sido dado. Este método retornará `(None, None)` si no hay usuario/contraseña concordante.

Para objetos `HTTPPasswordMgrWithDefaultRealm`, el reino `None` será buscado si el *realm* dado no tiene usuario/

contraseña concordante.

Objetos HTTPPasswordMgrWithPriorAuth

Esta manejador de contraseña extiende **`HTTPPasswordMgrWithDefaultRealm`** para soportar el seguimiento de URIs para las cuales deben ser enviadas siempre credenciales de autenticación.

HTTPPasswordMgrWithPriorAuth.add_password(*realm*, *uri*, *user*, *passwd*, *is_authenticated=False*)

> *realm*, *uri*, *user*, *passwd* son como para **`HTTPPasswordMgr.add_password()`**. *is_authenticated* establece el valor inicial del indicador `is_authenticated` para la URI o lista de URIs dadas. Si se especifica *is_authenticated* como `True`, *realm* se ignora.

HTTPPasswordMgrWithPriorAuth.find_user_password(*realm*, *authuri*)

> Lo mismo que para objetos **`HTTPPasswordMgrWithDefaultRealm`**

HTTPPasswordMgrWithPriorAuth.update_authenticated(*self*, *uri*, *is_authenticated=False*)

> Actualiza el indicador `is_authenticated` para la *uri* o lista de URIs dadas.

HTTPPasswordMgrWithPriorAuth.is_authenticated(*self*, *authuri*)

> Retorna el estado actual del indicador `is_authenticated` para la URI dada.

Objetos AbstractBasicAuthHandler

AbstractBasicAuthHandler.http_error_auth_reqed(*authreq*, *host*, *req*, *headers*)

Maneja una autenticación de petición obteniendo un par usuario/contraseña y reintentando la petición. *authreq* debe ser el nombre del encabezado donde la información sobre el reino se incluye en la petición, *host* especifica la URL y ruta para la cual autenticar, *req* debe ser el objeto **`Request`** (fallido) y *headers* deben ser los encabezados de error.

host es una autoridad (ej. `"python.org"`) o una URL conteniendo un componente de autoridad (ej. `"http://python.org/"`). En cualquier caso, la autoridad no debe contener un componente userinfo (por lo que `"python.org"` y `"python.org:80"` están bien, `"joe:password@python.org"` no).

Objetos HTTPBasicAuthHandler

HTTPBasicAuthHandler.http_error_401(*req*, *fp*, *code*, *msg*, *hdrs*)

Reintenta la petición con la información de autenticación, si está disponible.

Objetos ProxyBasicAuthHandler

ProxyBasicAuthHandler.http_error_407(*req*, *fp*, *code*, *msg*, *hdrs*)

Reintenta la petición con la información de autenticación, si está disponible.

Objetos AbstractDigestAuthHandler

AbstractDigestAuthHandler.http_error_auth_reqed(*authreq*, *host*, *req*, *headers*)

authreq debe ser el nombre del encabezado donde la información sobre el reino está incluida en la petición, *host* debe ser el host al que autenticar, *req* debe ser el objeto **`Request`** (fallido) y *headers* deben ser los encabezados de error.

Objetos HTTPDigestAuthHandler

HTTPDigestAuthHandler.http_error_401(*req, fp, code, msg, hdrs*)

Reintenta la petición con la información de autenticación, si está disponible.

Objetos ProxyDigestAuthHandler

ProxyDigestAuthHandler.http_error_407(*req, fp, code, msg, hdrs*)

Reintenta la petición con la información de autenticación, si está disponible.

Objetos HTTPHandler

HTTPHandler.http_open(*req*)

Send an HTTP request, which can be either GET or POST, depending on `req.data`.

Objetos HTTPSHandler

HTTPSHandler.https_open(*req*)

Send an HTTPS request, which can be either GET or POST, depending on `req.data`.

Objetos FileHandler

FileHandler.file_open(*req*)

Abre el archivo localmente, si no hay nombre de host, o el nombre de host es `'localhost'`.

Distinto en la versión 3.2: This method is applicable only for local hostnames. When a remote hostname is given, a **`URLError`** is raised.

Objetos DataHandler

DataHandler.data_open(*req*)

Read a data URL. This kind of URL contains the content encoded in the URL itself. The data URL syntax is specified in **RFC 2397** [https://datatracker.ietf.org/doc/html/rfc2397.html]. This implementation ignores white spaces in base64 encoded data URLs so the URL may be wrapped in whatever source file it comes from. But even though some browsers don't mind about a missing padding at the end of a base64 encoded data URL, this implementation will raise a `ValueError` in that case.

Objetos FTPHandler

FTPHandler.ftp_open(*req*)

Abre el archivo FTP indicado por *req*. El inicio de sesión siempre se realiza con un usuario y contraseña vacíos.

Objetos CacheFTPHandler

Los objetos `CacheFTPHandler` son objetos `FTPHandler` con los siguientes métodos adicionales:

CacheFTPHandler.setTimeout(*t*)

Establece el tiempo de expiración de conexiones a *t* segundos.

CacheFTPHandler.setMaxConns(*m*)

Establece el número máximo de conexiones cacheadas a *m*.

Objetos UnknownHandler

UnknownHandler.unknown_open()

Genera una excepción `URLError`.

Objetos HTTPErrorProcessor

HTTPErrorProcessor.http_response(*request*, *response*)

Procesa errores de respuestas HTTP.

Para códigos de error que no están en el rango de los 200, el objeto de respuesta es retornado inmediatamente.

For non-200 error codes, this simply passes the job on to the **`http_error_<type>()`** handler methods, via `OpenerDirector.error()`. Eventually, `HTTPDefaultErrorHandler` will raise an `HTTPError` if no other handler handles the error.

HTTPErrorProcessor.https_response(*request*, *response*)

Procesa los errores HTTPS de las respuestas.

Este comportamiento es el mismo que `http_response()`.

Ejemplos

Adicionalmente a los ejemplos siguientes, se dan más ejemplos en HOWTO - Cómo obtener recursos de Internet con el paquete urllib.

This example gets the python.org main page and displays the first 300 bytes of it:

```
>>> import urllib.request
>>> with urllib.request.urlopen('http://www.python.org/'
...     print(f.read(300))
...
b'<!doctype html>\n<!--[if lt IE 7]>   <html class="no-
```

Tenga en cuenta que urlopen retorna un objeto de bytes. Esto es porque no hay forma para urlopen de determinar automáticamente la codificación del flujo de bytes que recibe del servidor HTTP. En general, un programa decodificará el objeto de bytes retornado a cadena de caracteres una vez que determine o adivine la codificación apropiada.

The following HTML spec document, https://html.spec.whatwg.org/#charset, lists the various ways in which an HTML or an XML document could have specified its encoding information.

For additional information, see the W3C document: https://www.w3.org/International/questions/qa-html-encoding-declarations.

As the python.org website uses *utf-8* encoding as specified in its meta tag, we will use the same for decoding the bytes object:

```
>>> with urllib.request.urlopen('http://www.python.org/'
...     print(f.read(100).decode('utf-8'))
...
<!doctype html>
<!--[if lt IE 7]>   <html class="no-js ie6 lt-ie7 lt-ie8
<!-
```

It is also possible to achieve the same result without using the context manager approach:

```
>>> import urllib.request
>>> f = urllib.request.urlopen('http://www.python.org/')
>>> try:
...     print(f.read(100).decode('utf-8'))
... finally:
...     f.close()
...
<!doctype html>
<!--[if lt IE 7]>   <html class="no-js ie6 lt-ie7 lt-ie8
<!--
```

En el siguiente ejemplo, estamos enviando un flujo de datos a la entrada estándar de un CGI y leyendo los datos que nos retorna. Tenga en cuenta que este ejemplo sólo funcionará cuando la instalación de Python soporte SSL.

```
>>> import urllib.request
>>> req = urllib.request.Request(url='https://localhost/
...                        data=b'This data is passed to
>>> with urllib.request.urlopen(req) as f:
...     print(f.read().decode('utf-8'))
...
Got Data: "This data is passed to stdin of the CGI"
```

El código para el CGI de muestra usado en el ejemplo anterior es:

```
#!/usr/bin/env python
import sys
data = sys.stdin.read()
print('Content-type: text/plain\n\nGot Data: "%s"' % dat
```

Aquí hay un ejemplo de realizar una petición `PUT` usando **`Request`**:

```
import urllib.request
DATA = b'some data'
req = urllib.request.Request(url='http://localhost:8080'
with urllib.request.urlopen(req) as f:
    pass
print(f.status)
print(f.reason)
```

Uso de Autenticación HTTP Básica:

```
import urllib.request
# Create an OpenerDirector with support for Basic HTTP A
auth_handler = urllib.request.HTTPBasicAuthHandler()
auth_handler.add_password(realm='PDQ Application',
                          uri='https://mahler:8092/site-
                          user='klem',
                          passwd='kadidd!ehopper')
opener = urllib.request.build_opener(auth_handler)
# ...and install it globally so it can be used with urlo
urllib.request.install_opener(opener)
with urllib.request.urlopen('http://www.example.com/logi
    print(f.read().decode('utf-8'))
```

`build_opener()` provides many handlers by default, including a

ProxyHandler. By default, **ProxyHandler** uses the environment variables named `<scheme>_proxy`, where `<scheme>` is the URL scheme involved. For example, the **http_proxy** environment variable is read to obtain the HTTP proxy's URL.

Este ejemplo reemplaza el **ProxyHandler** predeterminado por uno que usa URLs de proxy suministradas mediante programación y añade soporte de autorización de proxy con **ProxyBasicAuthHandler**.

```
proxy_handler = urllib.request.ProxyHandler({'http': 'ht
proxy_auth_handler = urllib.request.ProxyBasicAuthHandle
proxy_auth_handler.add_password('realm', 'host', 'userna

opener = urllib.request.build_opener(proxy_handler, prox
# This time, rather than install the OpenerDirector, we 
with opener.open('http://www.example.com/login.html') as
   print(f.read().decode('utf-8'))
```

Añadiendo encabezados HTTP:

Usa el argumento *headers* en el constructor de **Request**, o:

```
import urllib.request
req = urllib.request.Request('http://www.example.com/')
req.add_header('Referer', 'http://www.python.org/')
# Customize the default User-Agent header value:
req.add_header('User-Agent', 'urllib-example/0.1 (Contac
with urllib.request.urlopen(req) as f:
    print(f.read().decode('utf-8'))
```

OpenerDirector añade automáticamente un encabezado *User-Agent* a cada **Request**. Para cambiar esto:

```
import urllib.request
opener = urllib.request.build_opener()
opener.addheaders = [('User-agent', 'Mozilla/5.0')]
with opener.open('http://www.example.com/') as f:
   print(f.read().decode('utf-8'))
```

También, recuerda que algunos encabezados estándar (*Content-Length*, *Content-Type* y *Host*) son añadidos cuando se pasa **`Request`** a **`urlopen()`** (o **`OpenerDirector.open()`**).

Aquí hay un ejemplo de sesión que usa el método `GET` para obtener una URL que contiene los parámetros:

```
>>> import urllib.request
>>> import urllib.parse
>>> params = urllib.parse.urlencode({'spam': 1, 'eggs':
>>> url = "http://www.musi-cal.com/cgi-bin/query?%s" % p
>>> with urllib.request.urlopen(url) as f:
...     print(f.read().decode('utf-8'))
...
```

El siguiente ejemplo usa el método POST en su lugar. Tenga en cuenta que la salida de parámetros desde urlencode es codificada a bytes antes de ser enviados a urlopen como datos:

```
>>> import urllib.request
>>> import urllib.parse
>>> data = urllib.parse.urlencode({'spam': 1, 'eggs': 2,
>>> data = data.encode('ascii')
>>> with urllib.request.urlopen("http://requestb.in/xrbl
...     print(f.read().decode('utf-8'))
...
```

El siguiente ejemplo usa un proxy HTTP especificado, sobrescribiendo las configuraciones de entorno:

```
>>> import urllib.request
>>> proxies = {'http': 'http://proxy.example.com:8080/'}
>>> opener = urllib.request.build_opener(urllib.request.
>>> with opener.open("http://www.python.org") as f:
...     f.read().decode('utf-8')
...
```

El siguiente ejemplo no usa proxies, sobrescribiendo las variables de entorno:

```
>>> import urllib.request
>>> opener = urllib.request.build_opener(urllib.request.
>>> with opener.open("http://www.python.org/") as f:
...     f.read().decode('utf-8')
...
```

Interfaz heredada

Las siguientes funciones y clases están portadas desde el módulo `urllib` de Python 2 (en oposición a `urllib2`). Ellas pueden estar obsoletas en algún punto del futuro.

urllib.request.urlretrieve(*url*, *filename=None*, *reporthook=None*, *data=None*)

Copy a network object denoted by a URL to a local file. If the URL points to a local file, the object will not be copied unless filename is supplied. Return a tuple `(filename, headers)` where *filename* is the local file name under which the object can be found, and *headers* is whatever the **`info()`** method of the object returned by **`urlopen()`** returned (for a remote object). Exceptions are the same as for **`urlopen()`**.

El segundo argumento, si está presente, especifica la localización a la que será copiada el objeto (si está ausente, la localización será un archivo temporal con un nombre generado). El tercer argumento, si está presente, es un objeto invocable que será invocado una vez que se establezca la conexión de red y después de eso una vez después de cada lectura de bloque. Al invocable se le pasarán tres argumentos; una cuenta de los bloques transferidos hasta el momento, un tamaño de bloque en bytes y el tamaño total del archivo. El tercer argumento puede ser `-1` en servidores FTP antiguos los cuales no retornan un tamaño de archivo en respuesta a una solicitud de recuperación.

El siguiente ejemplo ilustra el escenario de uso más común:

```
>>> import urllib.request
>>> local_filename, headers = urllib.request.urlret
>>> html = open(local_filename)
>>> html.close()
```

Si la *url* usa el esquema de identificador `http:`, el argumento opcional *data* puede ser dado para especificar una petición `POST` (normalmente el tipo de petición es `GET`). El argumento *data* debe ser un objeto de bytes en formato *application/x-www-form-urlencoded*; vea la función **`urllib.parse.urlencode()`**.

`urlretrieve()` will raise **`ContentTooShortError`** when it detects that the amount of data available was less than the expected amount (which is the size reported by a *Content-Length* header). This can occur, for example, when the download is interrupted.

El *Content-Length* es tratado como un límite inferior: si no hay más datos a leer, urlretrieve lee más datos, pero si están disponibles menos datos, se genera la excepción.

You can still retrieve the downloaded data in this case, it is stored in the **`content`** attribute of the exception instance.

Si no fue proporcionado el encabezado *Content-Length*, urlretrieve no puede comprobar el tamaño de los datos que han sido descargados, y sólo los retorna. En este caso sólo tienes que asumir que la descarga fue exitosa.

urllib.request.urlcleanup()

Limpia archivos temporales que pueden haber quedado tras llamadas anteriores a **`urlretrieve()`**.

`urllib.request` Restrictions

- Actualmente, sólo uno de los siguientes protocolo están soportados: HTTP (versiones 0.9 y 1.0), FTP, archivos locales y URLs de datos.

  Distinto en la versión 3.4: Añadido soporte para URLs de datos.

- La característica de caché de `urlretrieve()` ha sido deshabilitada hasta que alguien encuentre el tiempo para hackear el procesamiento adecuado de los encabezados de tiempo de Expiración.

- Debería haber una función para consultar si una URL en particular está en la caché.

- Para compatibilidad con versiones anteriores, si una URL parece apuntar a un archivo local pero el archivo no puede ser abierto, la URL es reinterpretada usando el protocolo FTP. Esto a veces puede causar mensajes de error confusos.

- Las funciones `urlopen()` y `urlretrieve()` pueden causar retrasos arbitrariamente largos mientras esperan a que se configure una conexión de red. Esto significa que es difícil construir un cliente Web interactivo usando estas funciones sin utilizar hilos.

- Los datos retornados por `urlopen()` o `urlretrieve()` son los datos en crudo retornados por el servidor. Estos pueden ser datos binarios (como una imagen), texto plano o (por ejemplo) HTML. El protocolo HTTP provee información de tipo en el encabezado de respuesta, el cual puede ser inspeccionado mirando el encabezado *Content-Type*. Si los datos retornados son HTML, puedes usar el módulo `html.parser` para analizarlos.

- The code handling the FTP protocol cannot differentiate between a file and a directory. This can lead to unexpected behavior when attempting to read a URL that points to a file that is not accessible. If the URL ends in a `/`, it is assumed to refer to a directory and will be handled accordingly. But if an attempt to read a file leads to a 550 error (meaning the URL cannot be found or is not accessible, often for permission reasons), then the path is treated as a directory in order to handle the case when a directory is specified by a URL but the trailing `/` has been left off. This can cause misleading results when you try to fetch a file whose read permissions make it inaccessible; the FTP code will try to read it, fail with a 550 error, and then perform a directory listing for the unreadable file. If fine-grained control is needed, consider using the **`ftplib`** module.

`urllib.response` — Response classes used by urllib

The **`urllib.response`** module defines functions and classes which define a minimal file-like interface, including `read()` and `readline()`. Functions defined by this module are used internally by the **`urllib.request`** module. The typical response object is a **`urllib.response.addinfourl`** instance:

class urllib.response.addinfourl

url

:URL del recurso obtenido, comúnmente usado para determinar si se ha seguido una redirección.

headers

Retorna las cabeceras de la respuesta en la forma de una instancia de **`EmailMessage`**.

status

Added in version 3.9.

Código de estado retornado por el servidor.

geturl()

Obsoleto desde la versión 3.9: Obsoleto en favor de **`url`**.

info()

Obsoleto desde la versión 3.9: Obsoleto en favor de **`headers`**.

code

Obsoleto desde la versión 3.9: Obsoleto en favor de **`status`**.

getcode()

Obsoleto desde la versión 3.9: Obsoleto en favor de **`status`**.

`urllib.parse` — Parse URLs into components

Código fuente: Lib/urllib/parse.py [https://github.com/python/cpython/tree/3.14/Lib/urllib/parse.py]

Este modulo define una interfaz estándar para separar cadenas de texto del Localizador de recursos uniforme (más conocido por las siglas URL, del inglés *Uniform Resource Locator*) en componentes (esquema de dirección, ubicación de red, ruta de acceso, etc.), para poder combinar los componentes nuevamente en una cadena de texto URL, y convertir una «URL relativa» a una URL absoluta a partir de una «URL base».

The module has been designed to match the internet RFC on Relative Uniform Resource Locators. It supports the following URL schemes: `file`, `ftp`, `gopher`, `hdl`, `http`, `https`, `imap`, `itms-services`, `mailto`, `mms`, `news`, `nntp`, `prospero`, `rsync`, `rtsp`, `rtsps`, `rtspu`, `sftp`, `shttp`, `sip`, `sips`, `snews`, `svn`, `svn+ssh`, `telnet`, `wais`, `ws`, `wss`.

Detalles de implementación de CPython: The inclusion of the `itms-services` URL scheme can prevent an app from passing Apple's App Store review process for the macOS and iOS App Stores. Handling for the `itms-services` scheme is always removed on iOS; on macOS, it *may* be removed if CPython has been built with the **`--with-app-store-compliance`** option.

The **`urllib.parse`** module defines functions that fall into two broad categories: URL parsing and URL quoting. These are covered in detail in the following sections.

This module's functions use the deprecated term `netloc` (or `net_loc`), which was introduced in **RFC 1808** [https://datatracker.ietf.org/doc/html/rfc1808.html]. However, this term has been obsoleted by **RFC 3986** [https://datatracker.ietf.org/doc/html/rfc3986.html], which introduced the term `authority` as its replacement. The use of `netloc` is continued for backward compatibility.

Análisis de URL

Las funciones de análisis de url se centran en dividir una cadena de dirección URL en sus componentes o en combinar componentes de dirección URL en una cadena de dirección URL.

urllib.parse.urlsplit(*urlstring*, *scheme=None*, *allow_fragments=True*)

Parse a URL into five components, returning a 5-item named tuple **`SplitResult`** or **`SplitResultBytes`**. This corresponds to the general structure of a URL: `scheme://netloc/path?query#fragment`. Each tuple item is a string, possibly empty. The components are not broken up into smaller parts (for example, the network location is a single string), and % escapes are not expanded. The delimiters as shown above are not part of the result, except for a leading slash in the *path* component, which is retained if present. For example:

```
>>> from urllib.parse import urlsplit
>>> urlsplit("scheme://netloc/path?query#fragment")
SplitResult(scheme='scheme', netloc='netloc', path=
            query='query', fragment='fragment')
>>> o = urlsplit("http://docs.python.org:80/3/libra
...                  "highlight=params#url-parsing")
>>> o
SplitResult(scheme='http', netloc='docs.python.org:
            path='/3/library/urllib.parse.html',
            query='highlight=params', fragment='url
>>> o.scheme
'http'
>>> o.netloc
'docs.python.org:80'
>>> o.hostname
'docs.python.org'
>>> o.port
```

```
80
>>> o._replace(fragment="").geturl()
'http://docs.python.org:80/3/library/urllib.parse.h
```

Following the syntax specifications in **RFC 1808** [https://datatracker.ietf.org/doc/html/rfc1808.html], **`urlsplit()`** recognizes a netloc only if it is properly introduced by “//”. Otherwise the input is presumed to be a relative URL and thus to start with a path component.

```
>>> from urllib.parse import urlsplit
>>> urlsplit('//www.cwi.nl:80/%7Eguido/Python.html'
SplitResult(scheme='', netloc='www.cwi.nl:80', path
            query='', fragment='')
>>> urlsplit('www.cwi.nl/%7Eguido/Python.html')
SplitResult(scheme='', netloc='', path='www.cwi.nl/
            query='', fragment='')
>>> urlsplit('help/Python.html')
SplitResult(scheme='', netloc='', path='help/Python
            query='', fragment='')
```

El argumento *scheme* proporciona el esquema de direccionamiento predeterminado, que solo se utilizará si la dirección URL no especifica uno. Debe ser del mismo tipo (texto o bytes) que *urlstring*, excepto que el valor predeterminado `''` siempre está permitido, y se convierte automáticamente a `b''` si aplica.

Si el argumento *allow_fragments* es falso, los identificadores de fragmento no son reconocidos. Sino que, son analizados como parte de la ruta, parámetros o componentes de la consulta y el **`fragment`** es configurado como una cadena vacía en el valor retornado.

El valor retornado es un named tuple, lo que significa que se puede tener acceso a sus elementos por índice o como atributos con nombre, que son:

Atributo	Índice	Valor	Valor si no está presente
scheme	0	Especificador de esquema de URL	parámetro *scheme*

netloc	1	Parte de ubicación de red	cadena vacía
path	2	Hierarchical path	cadena vacía
query	3	Componente de consulta	cadena vacía
fragment	4	Identificador de fragmento	cadena vacía
username		Nombre de usuario	**None**
password		Contraseña	**None**
hostname		Nombre de host (minúsculas)	**None**
port		Número de puerto como entero, si está presente	**None**

La lectura del atributo **`port`** lanzará un **`ValueError`** si se especifica un puerto no válido en la dirección URL. Consulte la sección Resultados del análisis estructurado para obtener más información sobre el objeto de resultado.

Los corchetes no coincidentes en el atributo **`netloc`** lanzarán un **`ValueError`**.

Los caracteres del atributo **`netloc`** que se descomponen en la normalización de NFKC (según lo utilizado por la codificación IDNA) en cualquiera de `/`, `?`, `#`, `@` o `:` lanzará un **`ValueError`**. Si la dirección URL se descompone antes del análisis, no se producirá ningún error.

Following some of the WHATWG spec [https://url.spec.whatwg.org/#concept-basic-url-parser] that updates **RFC 3986** [https://datatracker.ietf.org/doc/html/rfc3986.html], leading C0 control and space characters are stripped from the URL. `\n`, `\r` and tab `\t` characters are removed from the URL at any position.

As is the case with all named tuples, the subclass has a few additional methods and attributes that are particularly useful. One such method is **`_replace()`**. The **`_replace()`** method will return a new **`SplitResult`** object replacing specified fields with new values.

```
>>> from urllib.parse import urlsplit
>>> u = urlsplit('//www.cwi.nl:80/%7Eguido/Python.h
>>> u
SplitResult(scheme='', netloc='www.cwi.nl:80', path
            query='', fragment='')
```

```
>>> u._replace(scheme='http')
SplitResult(scheme='http', netloc='www.cwi.nl:80', 
            query='', fragment='')
```

Advertencia

`urlsplit()` no realiza validaciones. Mire URL parsing security para más información.

Distinto en la versión 3.2: Se han añadido capacidades de análisis de URL IPv6.

Distinto en la versión 3.3: The fragment is now parsed for all URL schemes (unless *allow_fragments* is false), in accordance with **RFC 3986** [https://datatracker.ietf.org/doc/html/rfc3986.html]. Previously, an allowlist of schemes that support fragments existed.

Distinto en la versión 3.6: Los números de puerto fuera de rango ahora generan **`ValueError`**, en lugar de retornar **`None`**.

Distinto en la versión 3.8: Los caracteres que afectan al análisis de netloc en la normalización de NFKC ahora lanzarán **`ValueError`**.

Distinto en la versión 3.10: Los caracteres ASCII de nueva línea y tab se eliminan de la URL.

Distinto en la versión 3.12: Los caracteres de control WHATWG C0 y de espacio se eliminan de la URL.

urllib.parse.parse_qs(*qs*, *keep_blank_values=False*, *strict_parsing=False*, *encoding='utf-8'*, *errors='replace'*, *max_num_fields=None*, *separator='&'*)

Analizar una cadena de consulta proporcionada como argumento de cadena (datos de tipo *application/x-www-form-urlencoded*). Los datos se retornan como un diccionario. Las claves de diccionario son los nombres de variable de consulta únicos y los valores son listas de valores para cada nombre.

El argumento opcional *keep_blank_values* es un indicador que indica si los valores en blanco de las consultas codificadas por porcentaje deben tratarse como cadenas en blanco. Un valor verdadero indica que los espacios en blanco deben conservarse como cadenas en blanco. El valor falso predeterminado indica que los valores en blanco deben omitirse y tratarse como si no se hubieran incluido.

El argumento opcional *strict_parsing* es una marca que indica qué hacer con los errores de análisis. Si es falso (valor predeterminado), los errores se omiten silenciosamente. Si es verdadero, los errores generan una excepción **`ValueError`**.

Los parámetros opcionales *encoding* y *errors* especifican cómo descodificar secuencias codificadas porcentualmente en caracteres Unicode, tal como lo acepta el método **`bytes.decode()`**.

El argumento opcional *max_num_fields* es el número máximo de campos que se van a leer. Si se establece, se produce un **`ValueError`** si hay más de *max_num_fields* campos leídos.

El argumento opcional *separator* es el símbolo que se usa para separar los argumentos de la cadena de consulta. El valor por defecto es `&`.

Utilice la función **`urllib.parse.urlencode()`** (con el parámetro `doseq` establecido en `True`) para convertir dichos diccionarios en cadenas de consulta.

Distinto en la versión 3.2: Agregue los parámetros *encoding* y *errors*.

Distinto en la versión 3.8: Añadido parámetro *max_num_fields*.

Distinto en la versión 3.10: Añadido parámetro *separator* con un valor por defecto de `&`. Versiones de Python anterior a Python 3.10 permitían el uso de `;` y `&` como separadores de la cadena de consulta. Esto ha sido cambiado para aceptar sólo una llave separadora, siendo `&` el separador por defecto.

Obsoleto desde la versión 3.14: Accepting objects with false values (like `0` and `[]`) except empty strings and byte-like objects and `None` is now deprecated.

urllib.parse.parse_qsl(*qs*, *keep_blank_values=False*, *strict_parsing=False*, *encoding='utf-8'*, *errors='replace'*, *max_num_fields=None*, *separator='&'*)

Analizar una cadena de consulta proporcionada como argumento de cadena (datos de tipo *application/x-www-form-urlencoded*). Los datos se retornan como una lista de pares de nombre y valor.

El argumento opcional *keep_blank_values* es un indicador que indica si los valores en blanco de las consultas codificadas por porcentaje deben tratarse como cadenas en blanco. Un valor verdadero indica que los espacios en blanco deben conservarse como cadenas en blanco. El valor falso predeterminado indica que los valores en blanco deben omitirse y tratarse como si no se hubieran incluido.

El argumento opcional *strict_parsing* es una marca que indica qué hacer con los errores de análisis. Si es falso (valor predeterminado), los errores se omiten silenciosamente. Si es verdadero, los errores generan una excepción **`ValueError`**.

Los parámetros opcionales *encoding* y *errors* especifican cómo descodificar secuencias codificadas porcentualmente en caracteres Unicode, tal como lo acepta el método **`bytes.decode()`**.

El argumento opcional *max_num_fields* es el número máximo de campos que se van a leer. Si se establece, se produce un **`ValueError`** si hay más de *max_num_fields* campos leídos.

El argumento opcional *separator* es el símbolo que se usa para separar los argumentos de la cadena de consulta. El valor por defecto es `&`.

Utilice la función **`urllib.parse.urlencode()`** para convertir dichas listas de pares en cadenas de consulta.

Distinto en la versión 3.2: Agregue los parámetros *encoding* y *errors*.

Distinto en la versión 3.8: Añadido parámetro *max_num_fields*.

Distinto en la versión 3.10: Añadido parámetro *separator* con un valor por defecto de `&`. Versiones de Python anterior a Python 3.10 permitían el uso de `;` y `&` como separadores de la cadena de consulta. Esto ha sido cambiado para aceptar sólo una llave separadora, siendo `&` el separador por defecto.

urllib.parse.urlunsplit(*parts*)

Construct a URL from a tuple as returned by `urlsplit()`. The *parts* argument can be any five-item iterable. This may result in a slightly different, but equivalent URL, if the URL that was parsed originally had unnecessary delimiters (for example, a `?` with an empty query; the RFC states that these are equivalent).

urllib.parse.urlparse(*urlstring*, *scheme=None*, *allow_fragments=True*)

This is similar to **`urlsplit()`**, but additionally splits the *path* component on *path* and *params*. This function returns a 6-item named tuple **`ParseResult`** or **`ParseResultBytes`**. Its items are the same as for the **`urlsplit()`** result, except that *params* is inserted at index 3, between *path* and *query*.

This function is based on obsoleted **RFC 1738** [https://datatracker.ietf.org/doc/html/rfc1738.html] and **RFC 1808** [https://datatracker.ietf.org/doc/html/rfc1808.html], which listed *params* as the main URL component. The more recent URL syntax allows parameters to be applied to each segment of the *path* portion of the URL (see **RFC 3986** [https://datatracker.ietf.org/doc/html/rfc3986.html]). **`urlsplit()`** should generally be used instead of **`urlparse()`**. A separate function is needed to separate the path segments and parameters.

urllib.parse.urlunparse(*parts*)

Combine the elements of a tuple as returned by **`urlparse()`** into a complete URL as a string. The *parts* argument can be any six-item iterable. This may result in a slightly different, but equivalent URL, if the URL that was parsed originally had unnecessary delimiters (for example, a ? with an empty query; the RFC states that these are equivalent).

urllib.parse.urljoin(*base*, *url*, *allow_fragments*=*True*)

Construya una URL completa («absoluta») combinando una «URL base» (*base*) con otra URL (*url*). Informalmente, esto utiliza componentes de la dirección URL base, en particular el esquema de direccionamiento, la ubicación de red y (parte de) la ruta de acceso, para proporcionar los componentes que faltan en la dirección URL relativa. Por ejemplo:

```
>>> from urllib.parse import urljoin
>>> urljoin('http://www.cwi.nl/%7Eguido/Python.html
'http://www.cwi.nl/%7Eguido/FAQ.html'
```

The *allow_fragments* argument has the same meaning and default as for **`urlsplit()`**.

Nota

Si *url* es una URL absoluta (es decir, comienza con `//` o `scheme://`), el nombre de host y/o esquema de *url* estará presente en el resultado. Por ejemplo:

```
>>> urljoin('http://www.cwi.nl/%7Eguido/Python.htm
...         '//www.python.org/%7Eguido')
'http://www.python.org/%7Eguido'
```

Si no desea ese comportamiento, preprocesa las partes *url* con **`urlsplit()`** y **`urlunsplit()`**, eliminando posibles partes *esquema* y *netloc*.

Advertencia

Because an absolute URL may be passed as the `url` parameter, it is generally **not secure** to use `urljoin` with an attacker-controlled `url`. For example in, `urljoin("https://website.com/users/",` `username)`, if `username` can contain an absolute URL, the result of `urljoin` will be the absolute URL.

Distinto en la versión 3.5: Comportamiento actualizado para coincidir con la semántica definida en **RFC 3986** [https://datatracker.ietf.org/doc/html/rfc3986.html].

urllib.parse.urldefrag(*url*)

Si *url* contiene un identificador de fragmento, retorne una versión modificada de *url* sin identificador de fragmento y el identificador de fragmento como una cadena independiente. Si no hay ningún identificador de fragmento en *url*, retorne *url* sin modificar y una cadena vacía.

El valor retornado es un named tuple, se puede acceder a sus elementos por índice o como atributos con nombre:

Atributo	**Índice**	**Valor**	**Valor si no está presente**
url	0	URL sin fragmento	cadena vacía
fragment	1	Identificador de fragmento	cadena vacía

Consulte la sección Resultados del análisis estructurado para obtener más información sobre el objeto de resultado.

Distinto en la versión 3.2: El resultado es un objeto estructurado en lugar de una simple tupla de 2.

urllib.parse.unwrap(*url*)

Extrae la url de una URL envuelta (es decir, una cadena de caracteres formateada como `<URL:scheme://host/path>`, `<scheme://host/path>`, `URL:scheme://host/path` or `scheme://host/path`). Si *url* no es una URL envuelta, se retorna sin cambios.

Análisis de seguridad de URL

Las APIs **`urlsplit()`** y **`urlparse()`** no realizan **validación** de las entradas. Es posible que no produzcan errores en entradas que otras aplicaciones consideren inválidas. También pueden tener éxito en algunas entradas que podrían no ser consideradas URLs en otros lugares. Su propósito es más la funcionalidad práctica que la pureza.

En vez de generar una excepción en una entrada inusual, pueden devolver algunas partes componentes como una cadenas vacías. O los componentes pueden contener más de lo que deberían.

Se recomienda que los usuarios de estas APIs, en las que los valores se pueden usar en cualquier lugar con implicaciones de seguridad, codifiquen de forma defensiva. Realicen algunas comprobaciones en su código antes de confiar en un componente devuelto. ¿Tiene sentido ese `scheme`? ¿Es ese `path` sensible? ¿Hay algo extraño en ese `hostname`? etc.

Lo que constituye una URL no está universalmente bien definido. Aplicaciones diferentes tienen diferentes necesidades y restricciones deseadas. Por ejemplo, la WHATWG spec [https://url.spec.whatwg.org/#concept-basic-url-parser] describe lo que requieren los clientes web orientados al usuario, como un navegador web. Mientras **RFC 3986** [https://datatracker.ietf.org/doc/html/rfc3986.html] es más general. Estas funciones incorporan algunos aspectos de ambos, pero no se puede afirmar que cumpla con ninguno de los dos. Las API y el código de usuario existente con expectativas sobre comportamientos específicos son anteriores a ambos estándares, lo que nos lleva a ser muy cautelosos realizando cambios en el comportamiento de la API.

Análisis de bytes codificados ASCII

Las funciones de análisis de URL se diseñaron originalmente para funcionar solo en cadenas de caracteres. En la práctica, es útil poder manipular las direcciones URL correctamente citadas y codificadas como secuencias de bytes ASCII. En consecuencia, las funciones de análisis de URL de este módulo funcionan en los objetos **`bytes`** y **`bytearray`**, además de los objetos **`str`**.

Si se pasan datos **`str`**, el resultado también contendrá solo los datos **`str`**. Si se pasan datos **`bytes`** o **`bytearray`**, el resultado contendrá solo los datos **`bytes`**.

Si intenta mezclar datos `str` con `bytes` o `bytearray` en una sola llamada de función, se producirá un `TypeError`, mientras que al intentar pasar valores de bytes no ASCII se desencadenará `UnicodeDecodeError`.

Para admitir una conversión más sencilla de objetos de resultado entre `str` y `bytes`, todos los valores retornados de las funciones de análisis de URL proporcionan un método `encode()` (cuando el resultado contiene `str` data) o un método `decode()` (cuando el resultado contiene datos `bytes`). Las firmas de estos métodos coinciden con las de los métodos correspondientes `str` y `bytes` (excepto que la codificación predeterminada es `'ascii`” en lugar de `'utf-8`”). Cada uno produce un valor de un tipo correspondiente que contiene datos `bytes` (para los métodos `encode()`) o `str` (para `decode()` métodos).

Las aplicaciones que necesitan operar en direcciones URL potencialmente citadas incorrectamente que pueden contener datos no ASCII tendrán que realizar su propia decodificación de bytes a caracteres antes de invocar los métodos de análisis de URL.

El comportamiento descrito en esta sección solo se aplica a las funciones de análisis de URL. Las funciones de citación de URL utilizan sus propias reglas al producir o consumir secuencias de bytes como se detalla en la documentación de las funciones de citación de URL individuales.

Distinto en la versión 3.2: Las funciones de análisis de URL ahora aceptan secuencias de bytes codificadas en ASCII

Resultados del análisis estructurado

The result objects from the `urlsplit()`, `urlparse()` and `urldefrag()` functions are subclasses of the `tuple` type. These subclasses add the attributes listed in the documentation for those functions, the encoding and decoding support described in the previous section, as well as an additional method:

urllib.parse.SplitResult.geturl()

Retorna la versión re-combinada de la dirección URL original como una cadena. Esto puede diferir de la dirección URL original en que el esquema se puede normalizar a minúsculas y los componentes vacíos pueden descartarse. En concreto, se quitarán los parámetros vacíos, las consultas y los identificadores de fragmento.

Para los resultados `urldefrag()`, solo se eliminarán los identificadores de fragmento vacíos. Para los resultados `urlsplit()` y `urlparse()`, todos los cambios observados se realizarán en la URL retornada por este método.

El resultado de este método permanece inalterado si se pasa de nuevo a través de la función de análisis original:

```
>>> from urllib.parse import urlsplit
>>> url = 'HTTP://www.Python.org/doc/#'
>>> r1 = urlsplit(url)
>>> r1.geturl()
'http://www.Python.org/doc/'
>>> r2 = urlsplit(r1.geturl())
>>> r2.geturl()
'http://www.Python.org/doc/'
```

Las clases siguientes proporcionan las implementaciones de los resultados del análisis estructurado cuando se opera en objetos `str`:

class urllib.parse.DefragResult(*url*, *fragment*)

Clase concreta para los resultados de `urldefrag()` que contienen datos `str`. El método **`encode()`** retorna una instancia `DefragResultBytes`.

Added in version 3.2.

class urllib.parse.ParseResult(*scheme*, *netloc*, *path*, *params*, *query*, *fragment*)

Clase concreta para los resultados de `urlparse()` que contiene `str` data. El método **`encode()`** retorna una instancia `ParseResultBytes`.

class urllib.parse.SplitResult(*scheme*, *netloc*, *path*, *query*, *fragment*)

Clase concreta para los resultados de `urlsplit()` que contiene `str` data. El método **`encode()`** retorna una instancia `SplitResultBytes`.

Las clases siguientes proporcionan las implementaciones de los resultados del análisis cuando se opera en objetos `bytes` o `bytearray`:

class urllib.parse.DefragResultBytes(*url*, *fragment*)

Clase concreta para los resultados de `urldefrag()` que contienen datos `bytes`. El método **`decode()`** retorna una instancia `DefragResult`.

Added in version 3.2.

class urllib.parse.ParseResultBytes(*scheme*, *netloc*, *path*, *params*, *query*, *fragment*)

Clase concreta para los resultados `urlparse()` que contienen datos `bytes`. El método **`decode()`** retorna una instancia `ParseResult`.

Added in version 3.2.

class urllib.parse.SplitResultBytes(*scheme*, *netloc*, *path*, *query*, *fragment*)

Clase concreta para los resultados de `urlsplit()` que contienen datos `bytes`. El método **`decode()`** retorna una instancia `SplitResult`.

Added in version 3.2.

Cita de URL

Las funciones de citación de URL se centran en tomar datos del programa y hacerlos seguros para su uso como componentes URL citando caracteres especiales y codificando adecuadamente texto no

ASCII. También admiten la inversión de estas operaciones para volver a crear los datos originales a partir del contenido de un componente de dirección URL si esa tarea aún no está cubierta por las funciones de análisis de URL anteriores.

urllib.parse.quote(*string*, *safe*='/', *encoding*=*None*, *errors*=*None*)

Reemplaza caracteres especiales en *string* con la secuencia de escape `%xx`. Las letras, los dígitos y los caracteres `'_.-~'` nunca se citan. De forma predeterminada, esta función está pensada para citar la sección de ruta de acceso de la dirección URL. El parámetro opcional *safe* especifica caracteres ASCII adicionales que no se deben citar — su valor predeterminado es `'/'`.

string puede ser un objeto **`str`** o **`bytes`**.

Distinto en la versión 3.7: Se ha movido de **RFC 2396** [https://datatracker.ietf.org/doc/html/rfc2396.html] a **RFC 3986** [https://datatracker.ietf.org/doc/html/rfc3986.html] para citar cadenas URL. Ahora se incluye «~» en el conjunto de caracteres reservados.

Los parámetros opcionales *encoding* y *errors* especifican cómo tratar con caracteres no ASCII, tal como lo acepta el método **`str.encode()`**. *encoding* por defecto es `'utf-8'`. *errors* tiene como valor predeterminado `'strict'`, lo que significa que los caracteres no admitidos generan un **`UnicodeEncodeError`**. *encoding* y *errors* no se deben proporcionar si *string* es **`bytes`** o se genera **`TypeError`**.

Tenga en cuenta que `quote(string, safe, encoding, errors)` es equivalente a `quote_from_bytes(string.encode(encoding, errors), safe)`.

Ejemplo: `quote('/El Niño/')` produce `'/El%20Ni%C3%B1o/'`.

urllib.parse.quote_plus(*string*, *safe*='', *encoding*=*None*, *errors*=*None*)

Como **`quote()`**, pero también reemplaza espacios con signos más, como se requiere para citar valores de formularios

HTML al momento de construir una cadena de consulta que será parte de una URL. Los signos más en la cadena de caracteres original se escapan, a no ser que se incluyan en *safe*. Además el valor de *safe* no es / por defecto.

Ejemplo: `quote_plus('/El Niño/')` produce `'%2FEl+Ni%C3%B1o%2F'`.

urllib.parse.quote_from_bytes(*bytes*, *safe='/'*)

Como **`quote()`**, pero acepta un objeto **`bytes`** en lugar de un **`str`**, y no realiza la codificación de cadena a bytes.

Ejemplo: `quote_from_bytes(b'a&\xef')` produce `'a%26%EF'`.

urllib.parse.unquote(*string*, *encoding='utf-8'*, *errors='replace'*)

Reemplaza secuencias de escape `%xx` con los caracteres individuales correspondientes. Los parámetros opcionales *encoding* y *errors* especifican cómo descodificar secuencias codificadas porcentualmente a caracteres Unicode, tal como lo acepta el método **`bytes.decode()`**.

string puede ser un objeto **`str`** o **`bytes`**.

encoding por defecto es `'utf-8`”. *errors* por defecto es `'replace`”, lo que significa que las secuencias no válidas se reemplazan por un carácter de marcador de posición.

Ejemplo: `unquote('/El%20Ni%C3%B1o/')` produce `'/El Niño/'`.

Distinto en la versión 3.9: El parámetro *string* admite bytes y cadenas de caracteres (anteriormente sólo cadenas de caracteres).

urllib.parse.unquote_plus(*string*, *encoding='utf-8'*, *errors='replace'*)

Como **`unquote()`**, pero también reemplaza los signos más por espacios, como es requerido al decodificar valores de formularios HTML.

string debe ser `str`.

Ejemplo: `unquote_plus('/El+Ni%C3%B1o/')` produce `'/El Niño/'`.

urllib.parse.unquote_to_bytes(*string*)

Reemplaza los escapes `%xx` por sus equivalentes de un solo octeto y retorna un objeto `bytes`.

string puede ser un objeto `str` o `bytes`.

Si es un `str`, los caracteres no ASCII sin escapar en *string* se codifican en bytes UTF-8.

Ejemplo: `unquote_to_bytes('a%26%EF')` produce `b'a&\xef'`.

urllib.parse.urlencode(*query*, *doseq=False*, *safe=''*, *encoding=None*, *errors=None*, *quote_via=quote_plus*)

Convierta un objeto de asignación o una secuencia de tuplas de dos elementos, que pueden contener objetos `str` o `bytes`, en una cadena de texto ASCII codificada porcentualmente. Si la cadena resultante se va a utilizar como una operación *data* para post con la función **`URLlib.request.urlopen()`**, entonces debe codificarse en bytes, de lo contrario resultaría en un `TypeError`.

La cadena resultante es una serie de pares `key=value` separados por caracteres `'&'`, donde tanto *key* como *value* se citan mediante la función *quote_via*. De forma predeterminada, `quote_plus()` se utiliza para citar los valores, lo que significa que los espacios se citan como un carácter `'+'` y "/" los caracteres se codifican como `%2F`, que sigue el estándar para las solicitudes GET (`application/x-www-form-urlencoded`). Una función alternativa que se puede pasar como *quote_via* es `quote()`, que codificará espacios como `%20` y no codificará caracteres "/". Para obtener el máximo control de lo que se cita, utilice `quote` y especifique un valor para *safe*.

Cuando se utiliza una secuencia de tuplas de dos elementos como argumento *query*, el primer elemento de cada tupla es una clave y el segundo es un valor. El valor en sí mismo puede ser una secuencia y, en ese caso, si el parámetro opcional *doseq* se evalúa como `True`, se generan pares individuales `key=value` separados por `'&'` para cada elemento de la secuencia de valores de la clave. El orden de los parámetros de la cadena codificada coincidirá con el orden de las tuplas de parámetros de la secuencia.

Los parámetros *safe*, *encoding* y *errors* se pasan a *quote_via* (los parámetros *encoding* y *errors* solo se pasan cuando un elemento de consulta es **`str`**).

Para revertir este proceso de codificación, en este módulo se proporcionan **`parse_qs()`** y **`parse_qsl()`** para analizar cadenas de consulta en estructuras de datos de Python.

Consulte urllib examples para averiguar cómo se puede utilizar el método **`urllib.parse.urlencode()`** para generar una cadena de consulta para una dirección URL o datos para POST.

Distinto en la versión 3.2: El parámetro *query* admite bytes y objetos de cadena.

Distinto en la versión 3.5: Added the *quote_via* parameter.

Obsoleto desde la versión 3.14: Accepting objects with false values (like `0` and `[]`) except empty strings and byte-like objects and `None` is now deprecated.

Ver también

WHATWG [https://url.spec.whatwg.org/] - URL estándar actual

Grupo de trabajo para el estándar URL que define URLs, dominio, dirección IP, formato *application/x-www-form-urlencoded* y su API.

RFC 3986 [https://datatracker.ietf.org/doc/html/rfc3986.html] - Identificadores uniformes de recursos

Este es el estándar actual (STD66). Cualquier cambio en el módulo urllib.parse debe ajustarse a esto. Se podrían observar ciertas desviaciones, que son principalmente para fines de compatibilidad con versiones anteriores y para ciertos requisitos de análisis de facto como se observa comúnmente en los principales navegadores.

RFC 2732 [https://datatracker.ietf.org/doc/html/rfc2732.html] - Formato de direcciones IPv6 literales en URL.

Esto especifica los requisitos de análisis de las direcciones URL IPv6.

RFC 2396 [https://datatracker.ietf.org/doc/html/rfc2396.html] - Identificadores uniformes de recursos (URI): Sintaxis genérica

Documento que describe los requisitos sintácticos genéricos para los nombres de recursos uniformes (URL) y los localizadores uniformes de recursos (URL).

RFC 2368 [https://datatracker.ietf.org/doc/html/rfc2368.html] - El esquema mailto URL.

Análisis de requisitos para esquemas de URL de correo a correo.

RFC 1808 [https://datatracker.ietf.org/doc/html/rfc1808.html] - Localizadores uniformes de recursos relativos

Esta solicitud de comentarios incluye las reglas para unirse a una URL absoluta y relativa, incluyendo un buen número de «Ejemplos anormales» que rigen el tratamiento de los casos fronterizos.

RFC 1738 [https://datatracker.ietf.org/doc/html/rfc1738.html] - Localizadores uniformes de recursos (URL)

Esto especifica la sintaxis formal y la semántica de las direcciones URL absolutas.

`urllib.error` — Exception classes raised by urllib.request

Código fuente: Lib/urllib/error.py [https://github.com/python/cpython/tree/3.14/Lib/urllib/error.py]

The `urllib.error` module defines the exception classes for exceptions raised by `urllib.request`. The base exception class is `URLError`.

The following exceptions are raised by `urllib.error` as appropriate:

exception urllib.error.URLError

> Los gestores lanzan esta excepción (o excepciones derivadas) cuando encuentran un problema. Es una subclase de `OSError`.
>
> reason
>
> > El motivo de este error. Puede ser una cadena de mensaje u otra instancia de una excepción.
>
> *Distinto en la versión 3.3:* `URLError` used to be a subtype of `IOError`, which is now an alias of `OSError`.

exception urllib.error.HTTPError(*url*, *code*, *msg*, *hdrs*, *fp*)

> A pesar de ser una excepción (una subclase de `URLError`), un `HTTPError` también puede funcionar como un valor de retorno no excepcional de tipo archivo (lo mismo que retorna `urlopen()`). Esto es útil para gestionar errores HTTP exóticos, como peticiones de autentificación.
>
> url
>
> > Contiene la URL de la solicitud. Un alias para el

atributo *filename*.

code

Un código de estado HTTP como los definidos en **RFC 2616** [https://datatracker.ietf.org/doc/html/rfc2616.html]. Este valor numérico se corresponde con un valor de un diccionario de códigos como el que hay en **`http.server.BaseHTTPRequestHandler.responses`**.

reason

Normalmente esto es una cadena de caracteres que explica el motivo de este error. Un alias para el atributo *msg*.

headers

Las cabeceras de la respuesta HTTP de la petición HTTP que causó el **`HTTPError`**. Un alias para el atributo *hdrs*.

Added in version 3.4.

fp

Un objeto similar a un archivo donde se puede leer el cuerpo del error HTTP.

exception urllib.error.ContentTooShortError(*msg*, *content*)

Esta excepción se lanza cuando la función **`urlretrieve()`** detecta que la cantidad de datos descargados es menor que la esperada (dada por la cabecera *Content-Length*).

content

Los datos descargados (y supuestamente truncados).

`urllib.robotparser` — Parser for robots.txt

Código fuente: Lib/urllib/robotparser.py [https://github.com/python/cpython/tree/3.14/Lib/urllib/robotparser.py]

This module provides a single class, **`RobotFileParser`**, which answers questions about whether or not a particular user agent can fetch a URL on the website that published the `robots.txt` file. For more details on the structure of `robots.txt` files, see http://www.robotstxt.org/orig.html.

class urllib.robotparser.RobotFileParser(*url=''*)

Esta clase proporciona métodos para leer, analizar y responder preguntas acerca de `robots.txt`

set_url(*url*)

Establece la URL que hace referencia a un archivo `robots.txt`.

read()

Lee la URL `robots.txt` y la envía al analizador.

parse(*lines*)

Analiza el argumento *lines*.

can_fetch(*useragent*, *url*)

Retorna `True` si el *useragent* tiene permiso para buscar la *url* de acuerdo con las reglas contenidas en el archivo `robots.txt` analizado.

mtime()

Retorna la hora en que se recuperó por última vez el archivo `robots.txt`. Esto es útil para arañas web de larga duración que necesitan buscar nuevos archivos `robots.txt` periódicamente.

modified()

Establece la hora a la que se recuperó por última vez el archivo `robots.txt` hasta la hora actual.

crawl_delay(*useragent*)

Retorna el valor del parámetro `Crawl-delay` de `robots.txt` para el *useragent* en cuestión. Si no existe tal parámetro o no se aplica al *useragent* especificado o la entrada `robots.txt` para este parámetro tiene una sintaxis no válida, devuelve `None`.

Added in version 3.6.

request_rate(*useragent*)

Retorna el contenido del parámetro `Request-rate` de `robots.txt` como una tupla nombrada `RequestRate(requests, seconds)`. Si no existe tal parámetro o no se aplica al *useragent* especificado o la entrada `robots.txt` para este parámetro tiene una sintaxis no válida, devuelve `None`.

Added in version 3.6.

site_maps()

Retorna el contenido del parámetro `Sitemap` de `robots.txt` en forma de **`list()`**. Si no existe tal parámetro o la entrada `robots.txt` para este parámetro tiene una sintaxis no válida, devuelve `None`.

Added in version 3.8.

El siguiente ejemplo demuestra el uso básico de la clase **`RobotFileParser`**:

```
>>> import urllib.robotparser
>>> rp = urllib.robotparser.RobotFileParser()
>>> rp.set_url("http://www.pythontest.net/robots.txt")
>>> rp.read()
>>> rrate = rp.request_rate("*")
>>> rrate.requests
1
>>> rrate.seconds
1
>>> rp.crawl_delay("*")
6
>>> rp.can_fetch("*", "http://www.pythontest.net/")
True
>>> rp.can_fetch("*", "http://www.pythontest.net/no-robo
False
```

`http` — HTTP modules

Código fuente: Lib/http/__init__.py [https://github.com/python/cpython/tree/3.14/Lib/http/__init__.py]

`http` is a package that collects several modules for working with the HyperText Transfer Protocol:

- `http.client` es un cliente del protocolo HTTP de bajo nivel; para la apertura de URL de alto nivel use `urllib.request`
- `http.server` contiene clases de servidor HTTP básicas basadas en `socketserver`
- `http.cookies` tiene utilidades para implementar la gestión de estados mediante cookies
- `http.cookiejar` provee persistencia de cookies

The `http` module also defines the following enums that help you work with http related code:

class http.HTTPStatus

Added in version 3.5.

Una subclase de `enum.IntEnum` que define un conjunto de códigos de estado HTTP, frases de motivo y descripciones largas escritas en inglés.

Uso:

```
>>> from http import HTTPStatus
>>> HTTPStatus.OK
HTTPStatus.OK
>>> HTTPStatus.OK == 200
True
>>> HTTPStatus.OK.value
200
```

```
>>> HTTPStatus.OK.phrase
'OK'
>>> HTTPStatus.OK.description
'Request fulfilled, document follows'
>>> list(HTTPStatus)
[HTTPStatus.CONTINUE, HTTPStatus.SWITCHING_PROTOCOL
```

Códigos de estado HTTP

Los códigos de estado registrados por IANA [https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml] disponibles en **`http.HTTPStatus`** son:

Código	Nombre de la enumeración	Detalles
100	CONTINUE	HTTP Semantics **RFC 9110**, Section 15.2.1
101	SWITCHING_PROTOCOLS	HTTP Semantics **RFC 9110**, Section 15.2.2
102	PROCESSING	WebDAV **RFC 2518**, Sección 10.1
103	EARLY_HINTS	Un código de estado HTTP para indicar pistas **RFC 8297**
200	OK	HTTP Semantics **RFC 9110**, Section 15.3.1
201	CREATED	HTTP Semantics **RFC 9110**, Section 15.3.2
202	ACCEPTED	HTTP Semantics **RFC 9110**, Section 15.3.3
203	NON_AUTHORITATIVE_INFORMATION	HTTP Semantics **RFC 9110**, Section 15.3.4
204	NO_CONTENT	HTTP Semantics **RFC 9110**, Section 15.3.5
205	RESET_CONTENT	HTTP Semantics **RFC 9110**, Section 15.3.6
206	PARTIAL_CONTENT	HTTP Semantics **RFC 9110**, Section 15.3.7
207	MULTI_STATUS	WebDAV **RFC 4918**, Sección 11.1
208	ALREADY_REPORTED	Extensiones de enlace a WebDAV **RFC 5842**, Sección 7.1 (Experimental)
226	IM_USED	Codificación delta en HTTP **RFC 3229**, Sección 10.4.1
300	MULTIPLE_CHOICES	HTTP Semantics **RFC 9110**, Section 15.4.1
301	MOVED_PERMANENTLY	HTTP Semantics **RFC 9110**, Section 15.4.2
302	FOUND	HTTP Semantics **RFC 9110**, Section 15.4.3
303	SEE_OTHER	HTTP Semantics **RFC 9110**, Section 15.4.4
304	NOT_MODIFIED	HTTP Semantics **RFC 9110**, Section 15.4.5
305	USE_PROXY	HTTP Semantics **RFC 9110**, Section 15.4.6
307	TEMPORARY_REDIRECT	HTTP Semantics **RFC 9110**, Section 15.4.8
308	PERMANENT_REDIRECT	HTTP Semantics **RFC 9110**, Section 15.4.9
400	BAD_REQUEST	HTTP Semantics **RFC 9110**, Section 15.5.1
401	UNAUTHORIZED	HTTP Semantics **RFC 9110**, Section 15.5.2

402 PAYMENT_REQUIRED	HTTP Semantics **RFC 9110**, Section 15.5.3
403 FORBIDDEN	HTTP Semantics **RFC 9110**, Section 15.5.4
404 NOT_FOUND	HTTP Semantics **RFC 9110**, Section 15.5.5
405 METHOD_NOT_ALLOWED	HTTP Semantics **RFC 9110**, Section 15.5.6
406 NOT_ACCEPTABLE	HTTP Semantics **RFC 9110**, Section 15.5.7
407 PROXY_AUTHENTICATION_REQUIRED	HTTP Semantics **RFC 9110**, Section 15.5.8
408 REQUEST_TIMEOUT	HTTP Semantics **RFC 9110**, Section 15.5.9
409 CONFLICT	HTTP Semantics **RFC 9110**, Section 15.5.10
410 GONE	HTTP Semantics **RFC 9110**, Section 15.5.11
411 LENGTH_REQUIRED	HTTP Semantics **RFC 9110**, Section 15.5.12
412 PRECONDITION_FAILED	HTTP Semantics **RFC 9110**, Section 15.5.13
413 CONTENT_TOO_LARGE	HTTP Semantics **RFC 9110**, Section 15.5.14
414 URI_TOO_LONG	HTTP Semantics **RFC 9110**, Section 15.5.15
415 UNSUPPORTED_MEDIA_TYPE	HTTP Semantics **RFC 9110**, Section 15.5.16
416 RANGE_NOT_SATISFIABLE	HTTP Semantics **RFC 9110**, Section 15.5.17
417 EXPECTATION_FAILED	HTTP Semantics **RFC 9110**, Section 15.5.18
418 IM_A_TEAPOT	HTCPCP/1.0 **RFC 2324**, Sección 2.3.2
421 MISDIRECTED_REQUEST	HTTP Semantics **RFC 9110**, Section 15.5.20
422 UNPROCESSABLE_CONTENT	HTTP Semantics **RFC 9110**, Section 15.5.21
423 LOCKED	WebDAV **RFC 4918**, Sección 11.3
424 FAILED_DEPENDENCY	WebDAV **RFC 4918**, Sección 11.4
425 TOO_EARLY	Uso de datos iniciales en HTTP **RFC 8470**
426 UPGRADE_REQUIRED	HTTP Semantics **RFC 9110**, Section 15.5.22
428 PRECONDITION_REQUIRED	Códigos de estado HTTP adicionales **RFC 6585**
429 TOO_MANY_REQUESTS	Códigos de estado HTTP adicionales **RFC 6585**
431 REQUEST_HEADER_FIELDS_TOO_LARGE	Códigos de estado HTTP adicionales **RFC 6585**
451 UNAVAILABLE_FOR_LEGAL_REASONS	Un código de estado HTTP para reportar obstáculos legales **RFC 7725**
500 INTERNAL_SERVER_ERROR	HTTP Semantics **RFC 9110**, Section 15.6.1
501 NOT_IMPLEMENTED	HTTP Semantics **RFC 9110**, Section 15.6.2
502 BAD_GATEWAY	HTTP Semantics **RFC 9110**, Section 15.6.3
503 SERVICE_UNAVAILABLE	HTTP Semantics **RFC 9110**, Section 15.6.4
504 GATEWAY_TIMEOUT	HTTP Semantics **RFC 9110**, Section 15.6.5
505 HTTP_VERSION_NOT_SUPPORTED	HTTP Semantics **RFC 9110**, Section 15.6.6
506 VARIANT_ALSO_NEGOTIATES	Negociación transparente de contenido en HTTP **RFC 2295**, Sección 8.1 (Experimental)
507 INSUFFICIENT_STORAGE	WebDAV **RFC 4918**, Sección 11.5
508 LOOP_DETECTED	Extensiones de unión WebDAV **RFC 5842**, Sección 7.2 (Experimental)
510 NOT_EXTENDED	Un framework de extensión HTTP **RFC 2774**, Sección 7

(Experimental)	
`NETWORK_AUTHENTICATION_REQUIRED`	RFC 6585, Sección 6

Códigos de estado HTTP adicionales

Con el fin de preservar la compatibilidad con versiones anteriores, los valores de la enumeración están también presentes en el módulo **`http.client`** en forma de constantes. El nombre de la enumeración es el mismo que el nombre de la constante (ej. `http.HTTPStatus.OK` se encuentra también disponible como `http.client.OK`).

Distinto en la versión 3.7: Se agregó el código de estado `421 MISDIRECTED_REQUEST`.

Added in version 3.8: Se agregó el código de estado `451 UNAVAILABLE_FOR_LEGAL_REASONS`.

Added in version 3.9: Agregados códigos de estado `103 EARLY_HINTS`, `418 IM_A_TEAPOT` y `425 TOO_EARLY`.

Distinto en la versión 3.13: Implemented RFC9110 naming for status constants. Old constant names are preserved for backwards compatibility: `413 REQUEST_ENTITY_TOO_LARGE`, `414 REQUEST_URI_TOO_LONG`, `416 REQUESTED_RANGE_NOT_SATISFIABLE` and `422 UNPROCESSABLE_ENTITY`.

Categoría de estado HTTP

Added in version 3.12.

Los valores enum tienen varias propiedades para indicar la categoría de estado HTTP:

Propiedad	**Indica que**	**Detalles**
`is_informational`	`100 <= status <= 199`	HTTP Semantics RFC 9110, Section 15
`is_success`	`200 <= status <= 299`	HTTP Semantics RFC 9110, Section 15
`is_redirection`	`300 <= status <= 399`	HTTP Semantics RFC 9110, Section 15
`is_client_error`	`400 <= status <= 499`	HTTP Semantics RFC 9110, Section 15
`is_server_error`	`500 <= status <= 599`	HTTP Semantics RFC 9110, Section 15

Uso:

```
>>> from http import HTTPStatus
>>> HTTPStatus.OK.is_success
True
>>> HTTPStatus.OK.is_client_error
False
```

class http.HTTPMethod

Added in version 3.11.

Una subclase de **`enum.StrEnum`** que define un conjunto de métodos HTTP y sus descripciones escritas en inglés.

Uso:

```
>>> from http import HTTPMethod
>>>
>>> HTTPMethod.GET
<HTTPMethod.GET>
>>> HTTPMethod.GET == 'GET'
True
>>> HTTPMethod.GET.value
'GET'
>>> HTTPMethod.GET.description
'Retrieve the target.'
>>> list(HTTPMethod)
[<HTTPMethod.CONNECT>,
 <HTTPMethod.DELETE>,
 <HTTPMethod.GET>,
 <HTTPMethod.HEAD>,
 <HTTPMethod.OPTIONS>,
 <HTTPMethod.PATCH>,
 <HTTPMethod.POST>,
 <HTTPMethod.PUT>,
 <HTTPMethod.TRACE>]
```

Métodos HTTP

Los métodos registrados por IANA [https://www.iana.org/assignments/http-methods/http-methods.xhtml] disponibles en **`http.HTTPMethod`** son:

Nombre de la enumeración	Método	Detalle
	GET	HTTP Semantics **RFC 9110**, Section 9.3.1
	HEAD	HTTP Semantics **RFC 9110**, Section 9.3.2
	POST	HTTP Semantics **RFC 9110**, Section 9.3.3
	PUT	HTTP Semantics **RFC 9110**, Section 9.3.4
	DELETE	HTTP Semantics **RFC 9110**, Section 9.3.5
	CONNECT	HTTP Semantics **RFC 9110**, Section 9.3.6
	OPTIONS	HTTP Semantics **RFC 9110**, Section 9.3.7
	TRACE	HTTP Semantics **RFC 9110**, Section 9.3.8
	PATCH	HTTP/1.1 **RFC 5789**

`http.client` — HTTP protocol client

Código fuente: Lib/http/client.py [https://github.com/python/cpython/tree/3.14/Lib/http/client.py]

Este módulo define clases que implementan el lado del cliente de los protocolos HTTP y HTTPS. Normalmente no se usa directamente — el módulo `urllib.request` lo usa para gestionar URLs que utilizan HTTP y HTTPS.

Ver también

The Requests package [https://requests.readthedocs.io/en/latest/] is recommended for a higher-level HTTP client interface.

Nota

El soporte HTTPS solo está disponible si Python se compiló con soporte SSL (a través del módulo `ssl`).

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

El módulo proporciona las siguientes clases:

class http.client.HTTPConnection(*host*, *port=None*, [*timeout*,]*source_address=None*, *blocksize=8192*)

Una instancia `HTTPConnection` representa una transacción con un servidor HTTP. Se debe instanciar pasándole un host y un número de puerto opcional. Si no se pasa ningún número de puerto, el puerto se extrae de la cadena de host si tiene la forma `host:port`; de lo contrario, se utiliza el puerto HTTP predeterminado (80). Si se proporciona el parámetro opcional *timeout*, las operaciones de bloqueo (como los intentos de conexión) expirarán después de esos segundos (si no se proporciona, se usa la configuración de tiempo de espera global predeterminada). El parámetro opcional *source_address* puede ser una tupla de un (host, puerto) para usar como la dirección de origen desde la que se realiza la conexión HTTP. El parámetro opcional *blocksize* establece el tamaño del búfer en bytes para enviar un cuerpo de mensaje similar a un archivo.

Por ejemplo, las siguientes llamadas crean instancias que se conectan al servidor en el mismo host y puerto:

```
>>> h1 = http.client.HTTPConnection('www.python.org
>>> h2 = http.client.HTTPConnection('www.python.org
>>> h3 = http.client.HTTPConnection('www.python.org
>>> h4 = http.client.HTTPConnection('www.python.org
```

Distinto en la versión 3.2: source_address fue adicionado.

Distinto en la versión 3.4: Se eliminó el argumento *strict*. Las «Respuestas Simples» de estilo HTTP 0.9 ya no son compatibles.

Distinto en la versión 3.7: argumento *blocksize* fue adicionado.

class http.client.HTTPSConnection(*host*, *port=None*, ***, [*timeout*,]*source_address=None*, *context=None*, *blocksize=8192*)

Una subclase de **`HTTPConnection`** que usa SSL para la comunicación con servidores seguros. El puerto predeterminado es `443`. Si se especifica *context*, debe ser una instancia de **`ssl.SSLContext`** que describa las diversas opciones de SSL.

Por favor lea Consideraciones de seguridad para obtener más información sobre las mejores prácticas.

Distinto en la versión 3.2: source_address, *context* y *check_hostname* fueron adicionados.

Distinto en la versión 3.2: This class now supports HTTPS virtual hosts if possible (that is, if **`ssl.HAS_SNI`** is true).

Distinto en la versión 3.4: Se eliminó el argumento *strict*. Las «Respuestas Simples» de estilo HTTP 0.9 ya no son compatibles.

Distinto en la versión 3.4.3: This class now performs all the necessary certificate and hostname checks by default. To revert to the previous, unverified, behavior **`ssl._create_unverified_context()`** can be passed to the *context* parameter.

Distinto en la versión 3.8: Esta clase ahora habilita TLS 1.3 **`ssl.SSLContext.post_handshake_auth`** para el *context* predeterminado o cuando *cert_file* se pasa con un *context* personalizado.

Distinto en la versión 3.10: This class now sends an ALPN extension with protocol indicator `http/1.1` when no *context* is given. Custom *context* should set ALPN protocols with **`set_alpn_protocols()`**.

Distinto en la versión 3.12: The deprecated *key_file*, *cert_file* and *check_hostname* parameters have been removed.

class http.client.HTTPResponse(*sock*, *debuglevel=0*, *method=None*, *url=None*)

Clase cuyas instancias se retornan tras una conexión exitosa. No son instancias realizadas directamente por el usuario.

Distinto en la versión 3.4: Se eliminó el argumento *strict*. Las «Respuestas Simples» del estilo HTTP 0.9 ya no están soportadas.

Este módulo proporciona la siguiente función:

http.client.parse_headers(*fp*)

Parse the headers from a file pointer *fp* representing a HTTP request/response. The file has to be a `BufferedIOBase` reader (i.e. not text) and must provide a valid **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html] style header.

Esta función retorna una instancia de `http.client.HTTPMessage` que contiene los campos de encabezado, pero no un *payload* (lo mismo que `HTTPResponse.msg` y `http.server.BaseHTTPRequestHandler.headers`) Después de regresar, el puntero de archivo *fp* está listo para leer el cuerpo HTTP.

Nota

`parse_headers()` no analiza la línea de inicio de un mensaje HTTP; solo analiza las líneas `Name: value`. El archivo tiene que estar listo para leer estas líneas de campo, por lo que la primera línea ya debe consumirse antes de llamar a la función.

Las siguientes excepciones son lanzadas según corresponda:

exception http.client.HTTPException

La clase base de las otras excepciones en este módulo. Es una subclase de `Exception`.

exception http.client.NotConnected

Una subclase de `HTTPException`.

exception http.client.InvalidURL

Una subclase de `HTTPException`, es lanzada si se proporciona un puerto y no es numérico o está vacío.

exception http.client.UnknownProtocol

Una subclase de `HTTPException`.

exception http.client.UnknownTransferEncoding

Una subclase de `HTTPException`.

exception http.client.UnimplementedFileMode

Una subclase de `HTTPException`.

exception http.client.IncompleteRead

Una subclase de `HTTPException`.

exception http.client.ImproperConnectionState

Una subclase de `HTTPException`.

exception http.client.CannotSendRequest

Una subclase de `ImproperConnectionState`.

exception http.client.CannotSendHeader

Una subclase de `ImproperConnectionState`.

exception http.client.ResponseNotReady

Una subclase de `ImproperConnectionState`.

exception http.client.BadStatusLine

Una subclase de `HTTPException`. Es lanzada si un servidor responde con un código de estado HTTP que no entendemos.

exception http.client.LineTooLong

Una subclase de `HTTPException`. Es lanzada si se recibe una línea excesivamente larga en el protocolo HTTP del servidor.

exception http.client.RemoteDisconnected

Una subclase de `ConnectionResetError` y `BadStatusLine`. Lanzada por `HTTPConnection.getresponse()` cuando el intento de leer la respuesta no produce datos leídos de la conexión, indica que el extremo remoto ha cerrado la conexión.

Added in version 3.5: Previamente, **`BadStatusLine`**`('')` fue lanzada.

Las constantes definidas en este módulo son:

http.client.HTTP_PORT

El puerto predeterminado para el protocolo HTTP (siempre `80`).

http.client.HTTPS_PORT

El puerto predeterminado para el protocolo HTTPS (siempre `443`).

http.client.responses

Este diccionario asigna los códigos de estado HTTP 1.1 a los nombres W3C.

Ejemplo: `http.client.responses[http.client.NOT_FOUND]` es `'Not Found'`.

Consulte Códigos de estado HTTP para obtener una lista de los códigos de estado HTTP que están disponibles en este módulo como constantes.

Objetos de `HTTPConnection`

Las instancias **`HTTPConnection`** tienen los siguientes métodos:

HTTPConnection.request(*method*, *url*, *body=None*, *headers={}*, ***, *encode_chunked=False*)

This will send a request to the server using the HTTP request method *method* and the request URI *url*. The provided *url* must be an absolute path to conform with **RFC 2616 §5.1.2** [https://datatracker.ietf.org/doc/html/rfc2616.html#section-5.1.2] (unless connecting to an HTTP proxy server or using the `OPTIONS` or `CONNECT` methods).

Si se especifica *body*, los datos especificados se envían una vez finalizados los encabezados. Puede ser **`str`**, un objeto bytes-like object, un objeto file object abierto, o un iterable de **`bytes`**. Si *body* es una cadena, se codifica como ISO-8859-1, el valor predeterminado para HTTP. Si es un objeto de tipo bytes, los bytes se envían tal cual. Si es un objeto file object, se envía el contenido del archivo; Este objeto de archivo debe soportar al menos el método `read()`. Si el objeto de archivo es una instancia de **`io.TextIOBase`**, los datos retornados por el método `read()` se codificarán como ISO-8859-1, de lo contrario, los datos retornados por `read()` se envía como está. Si *body* es un iterable, los elementos del iterable se envían tal cual hasta que se agota el iterable.

The *headers* argument should be a mapping of extra HTTP headers to send with the request. A **Host header** [https://datatracker.ietf.org/doc/html/rfc2616.html#section-14.23] must be provided to conform with **RFC 2616 §5.1.2** [https://datatracker.ietf.org/doc/html/rfc2616.html#section-5.1.2] (unless connecting to an HTTP proxy server or using the `OPTIONS` or `CONNECT` methods).

Si *headers* no contiene `Content-Length` ni Transfer-Encoding, pero hay un cuerpo de solicitud, uno de esos campos de encabezado se agregará automáticamente. Si *body* es `None`, el encabezado `Content-Length` se establece en `0` para los métodos que esperan un cuerpo (`PUT`, `POST` y `PATCH`) . Si *body* es una cadena de caracteres o un objeto similar a bytes que no es también un file, el encabezado `Content-Length` se establece en su longitud. Cualquier otro tipo de *body* (archivos e iterables en general) se codificará en fragmentos, y el encabezado `Transfer-Encoding` se establecerá automáticamente en lugar de `Content-Length`.

El argumento *encode_chunked* solo es relevante si `Transfer-Encoding` se especifica en *headers*. Si *encode_chunked* es `False`, el objeto `HTTPConnection` supone que toda la codificación es manejada por el código de llamada. Si es `True`, el cuerpo estará codificado en fragmentos.

For example, to perform a `GET` request to `https://docs.python.org/3/`:

```
>>> import http.client
>>> host = "docs.python.org"
>>> conn = http.client.HTTPSConnection(host)
>>> conn.request("GET", "/3/", headers={"Host": hos
>>> response = conn.getresponse()
>>> print(response.status, response.reason)
200 OK
```

Nota

La codificación de transferencia fragmentada se ha agregado al protocolo HTTP versión 1.1. A menos que se sepa que el servidor HTTP maneja HTTP 1.1, el llamador debe especificar la longitud del contenido o debe pasar un **`str`** o un objeto similar a bytes que no sea también un archivo como la representación del cuerpo.

Nota

Note that you must have read the whole response or call **`close()`** if **`getresponse()`** raised an non-**`ConnectionError`** exception before you can send a new request to the server.

Distinto en la versión 3.2: body ahora puede ser un iterable.

Distinto en la versión 3.6: Si ni `Content-Length` ni `Transfer-Encoding` están configurados en *headers*, el archivo y los objetos iterables *body* ahora están codificados en fragmentos. Se agregó el argumento *encode_chunked*. No se intenta determinar la longitud del contenido para los objetos de archivo.

HTTPConnection.getresponse()

Debe llamarse después de enviar una solicitud para obtener la respuesta del servidor. Retorna una instancia de

`HTTPResponse`.

Distinto en la versión 3.5: Si una **`ConnectionError`** o una subclase fue lanzada, el objeto **`HTTPConnection`** estará listo para volver a conectarse cuando se envíe una nueva solicitud.

Note that this does not apply to **`OSError`**s raised by the underlying socket. Instead the caller is responsible to call **`close()`** on the existing connection.

HTTPConnection.set_debuglevel(*level*)

Establecer el nivel de depuración. El nivel de depuración predeterminado es `0`, lo que significa que no se imprime ninguna salida de depuración. Cualquier valor mayor que `0` hará que todos los resultados de depuración definidos actualmente se impriman en *stdout*. El `debuglevel` se pasa a cualquier objeto nuevo **`HTTPResponse`** que se cree.

Added in version 3.1.

HTTPConnection.set_tunnel(*host*, *port=None*, *headers=None*)

Configure el host y el puerto para el túnel de conexión HTTP. Esto permite ejecutar la conexión a través de un servidor proxy.

The *host* and *port* arguments specify the endpoint of the tunneled connection (i.e. the address included in the CONNECT request, *not* the address of the proxy server).

The *headers* argument should be a mapping of extra HTTP headers to send with the CONNECT request.

As HTTP/1.1 is used for HTTP CONNECT tunnelling request, as per the RFC [https://datatracker.ietf.org/doc/html/rfc7231#section-4.3.6], a HTTP `Host:` header must be provided, matching the authority-form of the request target provided as the destination for the CONNECT request. If a HTTP `Host:` header is not provided via the headers argument, one is generated and transmitted automatically.

Por ejemplo, para hacer un túnel a través de un servidor proxy HTTPS que se ejecuta localmente en el puerto 8080, pasaríamos la dirección del proxy al constructor **`HTTPSConnection`**, y la dirección del host al que finalmente queremos llegar al método **`set_tunnel()`**:

```
>>> import http.client
>>> conn = http.client.HTTPSConnection("localhost",
>>> conn.set_tunnel("www.python.org")
>>> conn.request("HEAD","/index.html")
```

Added in version 3.2.

Distinto en la versión 3.12: HTTP CONNECT tunnelling requests use protocol HTTP/1.1, upgraded from protocol HTTP/1.0. `Host:` HTTP headers are mandatory for HTTP/1.1, so one will be automatically generated and transmitted if not provided in the headers argument.

HTTPConnection.get_proxy_response_headers()

Returns a dictionary with the headers of the response received from the proxy server to the CONNECT request.

If the CONNECT request was not sent, the method returns `None`.

Added in version 3.12.

HTTPConnection.connect()

Se conecta al servidor especificado cuando el objeto fue creado. Por defecto, esto se llama automáticamente cuando se realiza una solicitud si el cliente aún no tiene una conexión.

Lanza un evento de auditoría `http.client.connect` con los argumentos `self`, `host`, `port`.

HTTPConnection.close()

Cierre la conexión al servidor.

HTTPConnection.blocksize

Tamaño del búfer en bytes para enviar un archivo como cuerpo del mensaje.

Added in version 3.7.

As an alternative to using the `request()` method described above, you can also send your request step by step, by using the four functions below.

HTTPConnection.putrequest(*method*, *url*, *skip_host=False*, *skip_accept_encoding=False*)

Esta debería ser la primera llamada después de que se haya realizado la conexión al servidor. Envía una línea al servidor que consta de la cadena de caracteres *method*, la cadena de caracteres *url* y la versión HTTP (`HTTP/1.1`). Para deshabilitar el envío automático de encabezados `Host:` o `Accept-Encoding:` (por ejemplo, para aceptar codificaciones de contenido adicionales), especifique *skip_host* o *skip_accept_encoding* con valores no Falsos.

HTTPConnection.putheader(*header*, *argument*[, ...])

Envía un encabezado de estilo **RFC 822** [https://datatracker.ietf.org/doc/html/rfc822.html]al servidor. Este envía una línea al servidor que consta del encabezado, dos puntos y un espacio, y el primer argumento. Si se dan más argumentos, se envían líneas de continuación, cada una de las cuales consta de tabulación y un argumento.

HTTPConnection.endheaders(*message_body=None*, *, *encode_chunked=False*)

Envía una línea en blanco al servidor, señalando el final de los encabezados. El argumento opcional *message_body* se puede usar para pasar un cuerpo de mensaje asociado a la solicitud.

Si *encode_chunked* es `True`, el resultado de cada iteración de *message_body* se codificará en fragmentos como se especifica en **RFC 7230** [https://datatracker.ietf.org/doc/html/rfc7230.html], Sección 3.3.1. La forma en que se codifican los datos depende del tipo de *message_body*. Si *message_body* implementa buffer interface la codificación dará como resultado un solo fragmento. Si *message_body* es una **`collections.abc.Iterable`**, cada iteración de *message_body* dará como resultado un fragmento. Si *message_body* es un objeto file object, cada llamada a `.read()` dará como resultado un fragmento. El método señala automáticamente el final de los datos codificados en fragmentos inmediatamente después de *message_body*.

Nota

Debido a la especificación de codificación fragmentada, fragmentos vacíos producidos por un cuerpo iterador será ignorado por el codificador de fragmentos. Esto es para evitar la terminación prematura de la lectura de la solicitud por parte del servidor de destino debido a una codificación con formato incorrecto.

Distinto en la versión 3.6: Added chunked encoding support and the *encode_chunked* parameter.

HTTPConnection.send(*data*)

Envía datos al servidor. Esto debe usarse directamente solo después de que se haya llamado al método **`endheaders()`** y antes de que se llame al método **`getresponse()`**.

Lanza un evento de auditoría `http.client.send` con los argumentos `self`, `data`.

Objetos de `HTTPResponse`

Una instancia de **`HTTPResponse`** envuelve la respuesta HTTP del servidor. Proporciona acceso a los encabezados de la solicitud y al cuerpo de la entidad. La respuesta es un objeto iterable y puede usarse en una declaración `with`.

Distinto en la versión 3.5: La interfaz `io.BufferedIOBase` ahora está implementada y todas sus operaciones de lectura están soportadas.

HTTPResponse.read([*amt*])

Lee y retorna el cuerpo de respuesta, o hasta los siguientes bytes *amt*.

HTTPResponse.readinto(*b*)

Lee hasta los siguientes bytes `len(b)` del cuerpo de respuesta en el búfer *b*. Retorna el número de bytes leídos.

Added in version 3.3.

HTTPResponse.getheader(*name*, *default=None*)

Retorna el valor del encabezado *name*, o *default* si no hay un encabezado que coincida con *name*. Si hay más de un encabezado con el nombre *name*, retorne todos los valores unidos por ", ". Si "default" es cualquier iterable que no sea una sola cadena de caracteres sus elementos se retornan, de manera similar, unidos por comas.

HTTPResponse.getheaders()

Retorna una lista de tuplas (encabezado, valor).

HTTPResponse.fileno()

Retorna el `fileno` del socket implícito.

HTTPResponse.msg

Una instancia `http.client.HTTPMessage` que contiene los encabezados de respuesta. **`http.client.HTTPMessage`** es una subclase de `email.message.Message`.

HTTPResponse.version

Versión del protocolo HTTP utilizada por el servidor. 10 para HTTP/1.0, 11 para HTTP/1.1.

HTTPResponse.url

URL del recurso recuperado, comúnmente utilizado para determinar si se siguió una redirección.

HTTPResponse.headers

Cabeceras de la respuesta en forma de una instancia **`email.message.EmailMessage`**.

HTTPResponse.status

Código del estado retornado por el servidor.

HTTPResponse.reason

Una frase de la razón es retornada por el servidor.

HTTPResponse.debuglevel

Un depurador. Si **`debuglevel`** es mayor que cero, los mensajes se imprimirán en `stdout` a medida que se lee y analiza la respuesta.

HTTPResponse.closed

Es `True` si la transmisión está cerrada.

HTTPResponse.geturl()

Obsoleto desde la versión 3.9: Deprecada a favor de **`url`**.

HTTPResponse.info()

Obsoleto desde la versión 3.9: Deprecada a favor de **`headers`**.

HTTPResponse.getcode()

Obsoleto desde la versión 3.9: Deprecada a favor de **`status`**.

Ejemplos

Aquí hay una sesión de ejemplo que usa el método `GET` *method*:

```
>>> import http.client
>>> conn = http.client.HTTPSConnection("www.python.org")
>>> conn.request("GET", "/")
>>> r1 = conn.getresponse()
>>> print(r1.status, r1.reason)
200 OK
>>> data1 = r1.read()  # This will return entire content
>>> # The following example demonstrates reading data in
>>> conn.request("GET", "/")
>>> r1 = conn.getresponse()
>>> while chunk := r1.read(200):
...     print(repr(chunk))
b'<!doctype html>\n<!--[if"...
...
>>> # Example of an invalid request
>>> conn = http.client.HTTPSConnection("docs.python.org"
>>> conn.request("GET", "/parrot.spam")
>>> r2 = conn.getresponse()
>>> print(r2.status, r2.reason)
404 Not Found
>>> data2 = r2.read()
>>> conn.close()
```

Aquí hay una sesión de ejemplo que usa el método `HEAD`. Tenga en cuenta que el método `HEAD` nunca retorna ningún dato.

```
>>> import http.client
>>> conn = http.client.HTTPSConnection("www.python.org")
>>> conn.request("HEAD", "/")
>>> res = conn.getresponse()
>>> print(res.status, res.reason)
200 OK
>>> data = res.read()
>>> print(len(data))
0
>>> data == b''
True
```

Aquí hay una sesión de ejemplo que usa el método `POST`:

```
>>> import http.client, urllib.parse
>>> params = urllib.parse.urlencode({'@number': 12524, '
>>> headers = {"Content-type": "application/x-www-form-u
...            "Accept": "text/plain"}
>>> conn = http.client.HTTPConnection("bugs.python.org")
>>> conn.request("POST", "", params, headers)
>>> response = conn.getresponse()
>>> print(response.status, response.reason)
302 Found
>>> data = response.read()
>>> data
b'Redirecting to <a href="https://bugs.python.org/issue1
>>> conn.close()
```

Las solicitudes HTTP `PUT` del lado del cliente son muy similares a las solicitudes `POST`. La diferencia radica solo en el lado del servidor donde los servidores HTTP permitirán que se creen recursos a través de la solicitud `PUT`. Cabe señalar que los métodos HTTP personalizados también se manejan en **`urllib.request.Request`** configurando el atributo de método apropiado. Aquí hay una sesión de ejemplo que muestra cómo enviar una solicitud `PUT`:

```
>>> # This creates an HTTP request
>>> # with the content of BODY as the enclosed represent
>>> # for the resource http://localhost:8080/file
...
>>> import http.client
>>> BODY = "***filecontents***"
>>> conn = http.client.HTTPConnection("localhost", 8080)
>>> conn.request("PUT", "/file", BODY)
>>> response = conn.getresponse()
>>> print(response.status, response.reason)
200, OK
```

Objetos de `HTTPMessage`

class http.client.HTTPMessage(*email.message.Message*)

Una instancia de `http.client.HTTPMessage` contiene los encabezados de una respuesta HTTP. Se implementa utilizando la clase `email.message.Message`.

`ftplib` — FTP protocol client

Código fuente Lib/ftplib.py [https://github.com/python/cpython/tree/3.14/Lib/ftplib.py]

Este módulo define la clase `FTP` y algunos elementos relacionados. La clase `FTP` implementa el lado cliente del protocolo FTP. Puede usar esto para escribir programas de Python que realicen una variedad de trabajos FTP automatizados, como duplicar otros servidores FTP. También es utilizado por el módulo `urllib.request` para manejar URL que usan FTP. Para más información sobre FTP (File Transfer Protocol), ver internet **RFC 959** [https://datatracker.ietf.org/doc/html/rfc959.html].

La codificación predeterminada es UTF-8, siguiendo **RFC 2640** [https://datatracker.ietf.org/doc/html/rfc2640.html].

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

Here’s a sample session using the `ftplib` module:

```
>>> from ftplib import FTP
>>> ftp = FTP('ftp.us.debian.org')  # connect to host, d
>>> ftp.login()                     # user anonymous, pa
'230 Login successful.'
>>> ftp.cwd('debian')               # change into "debia
'250 Directory successfully changed.'
>>> ftp.retrlines('LIST')           # list directory con
-rw-rw-r--    1 1176     1176         1063 Jun 15 10:18
...
drwxr-sr-x    5 1176     1176         4096 Dec 19  2000
drwxr-sr-x    4 1176     1176         4096 Nov 17  2008
drwxr-xr-x    3 1176     1176         4096 Oct 10  2012
```

```
'226 Directory send OK.'
>>> with open('README', 'wb') as fp:
>>>     ftp.retrbinary('RETR README', fp.write)
'226 Transfer complete.'
>>> ftp.quit()
'221 Goodbye.'
```

Reference

FTP objects

class ftplib.FTP(*host=''*, *user=''*, *passwd=''*, *acct=''*, *timeout=None*, *source_address=None*, ***, *encoding='utf-8'*)

Return a new instance of the **`FTP`** class.

Parámetros:

- **host** (*str*) – The hostname to connect to. If given, `connect(host)` is implicitly called by the constructor.
- **user** (*str*) – The username to log in with (default: `'anonymous'`). If given, `login(host, passwd, acct)` is implicitly called by the constructor.
- **passwd** (*str*) – The password to use when logging in. If not given, and if *passwd* is the empty string or `"-"`, a password will be automatically generated.
- **acct** (*str*) – Account information to be used for the `ACCT` FTP command. Few systems implement this. See RFC-959 [https://datatracker.ietf.org/doc/html/rfc959.html] for more details.
- **timeout** (*float* | *None*) – A timeout in seconds for blocking operations like **`connect()`** (default: the global default timeout setting).
- **source_address** (*tuple* | *None*) – A 2-

tuple `(host, port)` for the socket to bind to as its source address before connecting.

- **encoding** (*str*) – The encoding for directories and filenames (default: `'utf-8'`).

La clase **`FTP`** admite la instrucción **`with`**, por ejemplo:

```
>>> from ftplib import FTP
>>> with FTP("ftp1.at.proftpd.org") as ftp:
...     ftp.login()
...     ftp.dir()
...
'230 Anonymous login ok, restrictions apply.'
dr-xr-xr-x   9 ftp      ftp           154 May  6 10
dr-xr-xr-x   9 ftp      ftp           154 May  6 10
dr-xr-xr-x   5 ftp      ftp          4096 May  6 10
dr-xr-xr-x   3 ftp      ftp            18 Jul 10  2
>>>
```

Distinto en la versión 3.2: Se agregó compatibilidad con la instrucción **`with`**.

Distinto en la versión 3.3: Se agregó el parámetro *source_address*.

Distinto en la versión 3.9: Si el parámetro *timeout* se establece en cero, lanzará un **`ValueError`** para evitar la creación de un socket sin bloqueo. Se agregó el parámetro *encoding*, y el valor predeterminado se cambió de Latin-1 a UTF-8 para seguir **RFC 2640** [https://datatracker.ietf.org/doc/html/rfc2640.html].

Several **`FTP`** methods are available in two flavors: one for handling text files and another for binary files. The methods are named for the command which is used followed by `lines` for the text version or `binary` for the binary version.

Las instancias de **`FTP`** tienen los siguientes métodos:

set_debuglevel(*level*)

Set the instance's debugging level as an `int`. This controls the amount of debugging output printed. The debug levels are:

- `0` (default): No debug output.
- `1`: Produce a moderate amount of debug output, generally a single line per request.
- `2` or higher: Produce the maximum amount of debugging output, logging each line sent and received on the control connection.

connect(*host=''*, *port=0*, *timeout=None*, *source_address=None*)

Connect to the given host and port. This function should be called only once for each instance; it should not be called if a *host* argument was given when the `FTP` instance was created. All other `FTP` methods can only be called after a connection has successfully been made.

Parámetros:

- **host** (*str*) – The host to connect to.
- **port** (*int*) – The TCP port to connect to (default: `21`, as specified by the FTP protocol specification). It is rarely needed to specify a different port number.
- **timeout** (*float* | *None*) – A timeout in seconds for the connection attempt (default: the global default timeout setting).
- **source_address** (*tuple* | *None*) – A 2-tuple `(host, port)` for the socket to bind to as its source address before connecting.

Lanza un evento auditor `ftplib.connect` con los argumentos `self`, `host`, `port`.

Distinto en la versión 3.3: Se agregó el parámetro *source_address*.

getwelcome()

Retornar el mensaje de bienvenida enviado por el servidor como respuesta a la conexión inicial. (Este mensaje a veces contiene renuncias de responsabilidad o información de ayuda que puede ser relevante para el usuario.)

login(*user='anonymous'*, *passwd=''*, *acct=''*)

Log on to the connected FTP server. This function should be called only once for each instance, after a connection has been established; it should not be called if the *host* and *user* arguments were given when the `FTP` instance was created. Most FTP commands are only allowed after the client has logged in.

Parámetros:

- **user** (*str*) – The username to log in with (default: `'anonymous'`).
- **passwd** (*str*) – The password to use when logging in. If not given, and if *passwd* is the empty string or `"-"`, a password will be automatically generated.
- **acct** (*str*) – Account information to be used for the `ACCT` FTP command. Few systems implement this. See RFC-959 [https://datatracker.ietf.org/doc/html/rfc959.html] for more details.

abort()

Anula una transferencia de archivo que está en progreso. Usarlo no siempre funciona, pero vale la pena intentarlo.

sendcmd(*cmd*)

Envía una cadena de comando simple al servidor y retorna la cadena de caracteres de respuesta.

Genera un evento auditor `ftplib.sendcmd` con los argumentos `self`, `cmd`.

voidcmd(*cmd*)

Send a simple command string to the server and handle the response. Return the response string if the response code corresponds to success (codes in the range 200–299). Raise **`error_reply`** otherwise.

Genera un evento auditor `ftplib.sendcmd` con los argumentos `self`, `cmd`.

retrbinary(*cmd*, *callback*, *blocksize=8192*, *rest=None*)

Retrieve a file in binary transfer mode.

Parámetros:

- **cmd** (*str*) – An appropriate `RETR` command: `"RETR filename"`.
- **callback** (callable) – A single parameter callable that is called for each block of data received, with its single argument being the data as **`bytes`**.
- **blocksize** (*int*) – The maximum chunk size to read on the low-level **`socket`** object created to do the actual transfer. This also corresponds to the largest size of data that will be passed to *callback*. Defaults to `8192`.
- **rest** (*int*) – A `REST` command to be sent to the server. See the

documentation for the *rest* parameter of the **`transfercmd()`** method.

retrlines(*cmd*, *callback=None*)

Retrieve a file or directory listing in the encoding specified by the *encoding* parameter at initialization. *cmd* should be an appropriate `RETR` command (see **`retrbinary()`**) or a command such as `LIST` or `NLST` (usually just the string `'LIST'`). `LIST` retrieves a list of files and information about those files. `NLST` retrieves a list of file names. The *callback* function is called for each line with a string argument containing the line with the trailing CRLF stripped. The default *callback* prints the line to **`sys.stdout`**.

set_pasv(*val*)

Habilita el modo pasivo si *val* es verdadero, de lo contrario lo inhabilita. El modo pasivo es el valor predeterminado.

storbinary(*cmd*, *fp*, *blocksize=8192*, *callback=None*, *rest=None*)

Store a file in binary transfer mode.

Parámetros:

- **cmd** (*str*) – An appropriate `STOR` command: `"STOR *filename*"`.
- **fp** (file object) – A file object (opened in binary mode) which is read until EOF, using its **`read()`** method in blocks of size *blocksize* to provide the data to be stored.
- **blocksize** (*int*) – The read block size. Defaults to `8192`.
- **callback** (callable) – A single parameter callable that is called

for each block of data sent, with its single argument being the data as `bytes`.
- **rest** (*int*) – A `REST` command to be sent to the server. See the documentation for the *rest* parameter of the `transfercmd()` method.

Distinto en la versión 3.2: The *rest* parameter was added.

storlines(*cmd*, *fp*, *callback=None*)

Almacena un archivo en modo de línea. *cmd* debe ser un comando `STOR` apropiado (ver `storbinary()`). Las líneas se leen hasta EOF del file object *fp* (abierto en modo binario) usando su método `readline()` para proporcionar los datos que se almacenarán. *callback* es un parámetro único opcional que se puede llamar en cada línea después de su envío.

transfercmd(*cmd*, *rest=None*)

Inicia una transferencia sobre la conexión de datos. Si la transferencia es activa, envía un comando `EPRT` o `PORT` y el comando de transferencia especificado por *cmd*, y acepta la conexión. Si el servidor es pasivo, envía un comando `EPSV` o `PASV`, lo conecta, e inicia el comando de transferencia. De cualquier manera, retorna el socket para la conexión.

Si se proporciona *rest* opcional, se envía un comando `REST` al servidor, pasando *rest* como argumento. *rest* suele ser un desplazamiento de bytes en el archivo solicitado, que le indica al servidor que reinicie el envío de los bytes del archivo en el desplazamiento solicitado, omitiendo los bytes iniciales. Sin embargo, tenga en cuenta que el método `transfercmd()` convierte *rest* en una cadena de caracteres con el parámetro *encoding* especificado en la inicialización, pero no se realiza ninguna comprobación del contenido de la cadena de caracteres. Si el servidor no reconoce el comando `REST`, se lanzará una excepción **`error_reply`**. Si esto sucede, simplemente llame a **`transfercmd()`** sin un argumento *rest*.

ntransfercmd(*cmd*, *rest=None*)

Como **`transfercmd()`**, pero retorna una tupla de conexión de datos y el tamaño esperado de los datos. Si el tamaño esperado no se pudo computar, retornará `None` como tal. *cmd* y *rest* significan lo mismo que en **`transfercmd()`**.

mlsd(*path=''*, *facts=[]*)

Enumera un directorio en un formato estandarizado usando el comando `MLSD` (**RFC 3659** [https://datatracker.ietf.org/doc/html/rfc3659.html]). Si se omite *path*, se asume el directorio actual. *facts* es una lista de cadenas de caracteres que representan el tipo de información deseada (por ejemplo, `["type", "size", "perm"]`). Retorna un objeto generador que produce una tupla de dos elementos por cada archivo encontrado en la ruta. El primer elemento es el nombre del archivo, el segundo es un diccionario que contiene datos sobre el nombre del archivo. El contenido de este diccionario puede estar limitado por el argumento *facts*, pero no se garantiza que el servidor retorne todos los datos solicitados.

Added in version 3.3.

nlst(*argument*[, ...])

Retorna una lista de nombres de archivos tal como los retorna el comando `NLST`. El *argument* opcional es un directorio para listar (el predeterminado es el directorio del servidor actual). Se pueden usar varios argumentos para pasar opciones no estándar al comando `NLST`.

Nota

Si tu servidor admite el comando, `mlsd()` ofrece una API mejor.

dir(*argument*[, ...])

Produce a directory listing as returned by the `LIST` command, printing it to standard output. The optional *argument* is a directory to list (default is the current server directory). Multiple arguments can be used to pass non-standard options to the `LIST` command. If the last argument is a function, it is used as a *callback* function as for `retrlines()`; the default prints to `sys.stdout`. This method returns `None`.

Nota

Si tu servidor admite el comando, `mlsd()` ofrece una API mejor.

rename(*fromname*, *toname*)

Asigna un nombre nuevo al archivo en el servidor desde *fromname* a *toname*.

delete(*filename*)

Remueve el archivo nombrado *filename* del servidor. De ser exitoso, retorna el texto de la respuesta, de lo contrario, lanza `error_perm` sobre errores de permiso o `error_reply` sobre otros errores.

cwd(*pathname*)

Configura el directorio actual en el servidor.

mkd(*pathname*)

Crea un nuevo directorio en el servidor.

pwd()

Retorna el nombre de ruta del directorio actual en el servidor.

rmd(*dirname*)

Elimina el directorio en el servidor llamado *dirname*.

size(*filename*)

Solicita el tamaño del archivo llamado *filename* en el servidor. De ser exitoso, se retorna el tamaño del archivo como un entero, de lo contrario retorna `None`. Nótese que el comando `SIZE` no está estandarizado, pero es admitido por muchas implementaciones comunes de servidor.

quit()

Envía un comando `QUIT` al servidor y cierra la conexión. Esta es la forma «políticamente correcta» de cerrar una conexión, pero puede lanzar una excepción si el servidor responde con un error al comando `QUIT`. Esto implica una llamada al método **`close()`** que hace inútil la instancia de **`FTP`** para llamadas posteriores (véase más adelante).

close()

Cierra la conexión de forma unilateral. Esto no debería aplicarse a una conexión ya cerrada, como luego de una llamada exitosa a **`quit()`**. Después de esta llamada, la instancia **`FTP`** ya no debería utilizarse (luego de una llamada a **`close()`** o **`quit()`** no puedes abrir nuevamente la conexión emitiendo otro método **`login()`**.

FTP_TLS objects

class ftplib.FTP_TLS(*host=''*, *user=''*, *passwd=''*, *acct=''*, ***, *context=None*, *timeout=None*, *source_address=None*,

encoding='utf-8')

An `FTP` subclass which adds TLS support to FTP as described in RFC 4217 [https://datatracker.ietf.org/doc/html/rfc4217.html]. Connect to port 21 implicitly securing the FTP control connection before authenticating.

Nota

The user must explicitly secure the data connection by calling the `prot_p()` method.

Parámetros:

- **host** (*str*) – The hostname to connect to. If given, `connect(host)` is implicitly called by the constructor.
- **user** (*str*) – The username to log in with (default: `'anonymous'`). If given, `login(host, passwd, acct)` is implicitly called by the constructor.
- **passwd** (*str*) – The password to use when logging in. If not given, and if *passwd* is the empty string or `"-"`, a password will be automatically generated.
- **acct** (*str*) – Account information to be used for the `ACCT` FTP command. Few systems implement this. See RFC-959 [https://datatracker.ietf.org/doc/html/rfc959.html] for more details.
- **context** (`ssl.SSLContext`) – An SSL context object which allows bundling SSL configuration options, certificates and private keys into a single, potentially long-lived, structure. Please read Consideraciones de seguridad for best practices.
- **timeout** (*float* | *None*) – A timeout in seconds for blocking operations like `connect()` (default: the global default

timeout setting).

- **source_address** (*tuple* | *None*) – A 2-tuple `(host, port)` for the socket to bind to as its source address before connecting.
- **encoding** (*str*) – The encoding for directories and filenames (default: `'utf-8'`).

Added in version 3.2.

Distinto en la versión 3.3: Added the *source_address* parameter.

Distinto en la versión 3.4: La clase ahora admite el chequeo del nombre con **`ssl.SSLContext.check_hostname`** y *Server Name Indication* (véase **`ssl.HAS_SNI`**).

Distinto en la versión 3.9: Si el parámetro *timeout* se establece en cero, lanzará un **`ValueError`** para evitar la creación de un socket sin bloqueo. Se agregó el parámetro *encoding*, y el valor predeterminado se cambió de Latin-1 a UTF-8 para seguir **RFC 2640** [https://datatracker.ietf.org/doc/html/rfc2640.html].

Distinto en la versión 3.12: Se han eliminado los parámetros obsoletos *keyfile* y *certfile*.

Aquí hay una sesión de ejemplo que usa la clase **`FTP_TLS`**:

```
>>> ftps = FTP_TLS('ftp.pureftpd.org')
>>> ftps.login()
'230 Anonymous user logged in'
>>> ftps.prot_p()
'200 Data protection level set to "private"'
>>> ftps.nlst()
['6jack', 'OpenBSD', 'antilink', 'blogbench', 'bsdc
```

`FTP_TLS` class inherits from **`FTP`**, defining these additional methods and attributes:

ssl_version

La versión SSL para usar (toma como predeterminado `ssl.PROTOCOL_SSLv23`).

auth()

Establece una conexión de control segura usando TLS o SSL, dependiendo de qué esté especificado en el atributo `ssl_version`.

Distinto en la versión 3.4: El método ahora admite el chequeo del nombre con `ssl.SSLContext.check_hostname` y *Server Name Indication* (véase `ssl.HAS_SNI`).

ccc()

Revierte el canal de control a texto plano. Esto puede ser útil para aprovechar cortafuegos que saben manejar NAT con FTP no-seguro sin abrir puertos fijos.

Added in version 3.3.

prot_p()

Configura conexión de datos segura.

prot_c()

Configura la conexión de datos de tipo texto común.

Module variables

exception ftplib.error_reply

Se lanza una excepción cuando una respuesta inesperada se recibe del servidor.

exception ftplib.error_temp

Se genera una excepción cuando se recibe un código de error que refiere a un error temporal (códigos de respuesta en el rango 400-499).

exception ftplib.error_perm

Se lanza una excepción cuando se recibe un código de error que refiere a un error permanente (códigos de respuesta en el rango 500–599).

exception ftplib.error_proto

Se lanza una excepción cuando se recibe una respuesta del servidor que no coincide con las especificaciones de respuesta del protocolo de transferencia de archivos (FTP), es decir, que comienza con un dígito en el rango 1–5.

ftplib.all_errors

El conjunto de todas las excepciones (como una tupla) que los métodos de instancias `FTP` pueden lanzar como resultado de problemas con la conexión FTP (a diferencia de los errores de programación hechos por el autor de la llamada). Este conjunto incluye las cuatro excepciones enumeradas anteriormente, como también `OSError` y `EOFError`.

Ver también

Módulo `netrc`

Analizador para el formato de archivo `.netrc`. El archivo `.netrc` suele ser utilizado por clientes FTP para cargar la información de autenticación de usuario antes de solicitarlo al usuario.

`poplib` — POP3 protocol client

Código fuente: Lib/poplib.py [https://github.com/python/cpython/tree/3.14/Lib/poplib.py]

Este módulo define una clase, `POP3`, que encapsula una conexión a un servidor POP3 e implementa el protocolo como está definido en **RFC 1939** [https://datatracker.ietf.org/doc/html/rfc1939.html]. La clase `POP3` soporta los mínimos y opcionales conjuntos de comandos de **RFC 1939** [https://datatracker.ietf.org/doc/html/rfc1939.html]. La clase `POP3` también soporta el comando `STLS` introducido en **RFC 2595** [https://datatracker.ietf.org/doc/html/rfc2595.html] para habilitar comunicación encriptada en una conexión ya establecida.

Adicionalmente, este módulo provee una clase `POP3_SSL`, que provee soporte para conectar servidores POP3 que usan SSL como una capa de protocolo subyacente.

Note que POP3, aunque ampliamente soportado, es obsoleto. La calidad de implementación de servidores POP3 varía ampliamente, y muchos son bastante pobres. Si su servidor de correo soporta IMAP, sería mejor utilizar la clase `imaplib.IMAP4`, ya que los servidores IMAP tienden a estar mejor implementados.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

The `poplib` module provides two classes:

class poplib.POP3(*host*, *port=POP3_PORT*[, *timeout*])

Esta clase implementa el protocolo POP3 actual. La conexión es creada cuando la instancia es inicializada. Si *port* se omite, el puerto POP3 estándar (110) es utilizado. El parámetro opcional *timeout* especifica un tiempo de espera en segundos para el intento de conexión (si no se especifica, la configuración global de tiempo de espera será utilizada).

Genera un evento de auditoría `poplib.connect` con argumentos `self`, `host`, `port`.

Todos los comandos lanzarán un evento de auditoría `poplib.putline` con argumentos `self` y `line`, donde `line` son los bytes a ser enviados al host remoto.

Distinto en la versión 3.9: Si el parámetro *timeout* se establece en cero, lanzará un **`ValueError`** para evitar la creación de un socket sin bloqueo.

class poplib.POP3_SSL(*host*, *port=POP3_SSL_PORT*, ***, *timeout=None*, *context=None*)

Esta es una subclase de **`POP3`** que conecta al servidor sobre un socket SSL encriptado. Si *port* no es especificado, 995, el puerto POP3-over-SSL es utilizado. *timeout* funciona como en el constructor de la clase **`POP3`**. *context* es un objeto **`ssl.SSLContext`** opcional que permite empaquetar opciones de configuración SSL, certificados y llaves privadas en una única (potencialmente longeva) estructura. Por favor lea Consideraciones de seguridad para buenas prácticas.

Genera un evento de auditoría `poplib.connect` con argumentos `self`, `host`, `port`.

Todos los comandos lanzarán un evento de auditoría `poplib.putline` con argumentos `self` y `line`, donde `line` son los bytes a ser enviados al host remoto.

Distinto en la versión 3.2: Parámetro *context* agregado.

Distinto en la versión 3.4: The class now supports hostname check with **`ssl.SSLContext.check_hostname`** and *Server Name Indication* (see **`ssl.HAS_SNI`**).

Distinto en la versión 3.9: Si el parámetro *timeout* se establece en cero, lanzará un **`ValueError`** para evitar la creación de un socket sin bloqueo.

Distinto en la versión 3.12: The deprecated *keyfile* and *certfile* parameters have been removed.

One exception is defined as an attribute of the `poplib` module:

exception poplib.error_proto

Excepción generada en cualquier error de este módulo (errores del módulo `socket` no son capturadas). La razón para la excepción es pasada al constructor como una cadena.

Ver también

Módulo `imaplib`

El módulo IMAP de Python.

Preguntas Frecuentes Sobre Fetchmail **[http://www.catb.org/~esr/fetchmail/fetchmail-FAQ.html]**

Las preguntas frecuentes para el cliente POP/IMAP **fetchmail** colecciona información en las variaciones del servidor POP3 e incumplimiento de RFC que puede ser útil si usted necesita escribir una aplicación basada en el protocolo POP.

Objetos POP3

Todos los comandos POP3 están representados por métodos del mismo nombre, en minúsculas; la mayoría retorna el texto de respuesta enviado por el servidor.

Una instancia `POP3` tiene los siguientes métodos:

POP3.set_debuglevel(*level*)

Establece el nivel de depuración de la instancia. Esto controla la cantidad de salida de depuración impresa. Por defecto, `0`, no produce salida de depuración. Un valor de `1` produce una moderada cantidad de salida de depuración, generalmente una única línea por solicitud. Un valor de `2` o mayor produce la máxima cantidad de salida de depuración, registrando cada línea enviada y recibida en la conexión de control.

POP3.getwelcome()

Retorna la cadena de saludo enviada por el servidor POP3.

POP3.capa()

Consulta las capacidades del servidor como está especificado en **RFC 2449** [https://datatracker.ietf.org/doc/html/rfc2449.html]. Retorna un diccionario en la forma `{'nombre': ['param'...]}`.

Added in version 3.4.

POP3.user(*username*)

Envía el comando del usuario, la respuesta debería indicar que una contraseña es requerida.

POP3.pass_(*password*)

Send password, response includes message count and mailbox size. Note: the mailbox on the server is locked until **`quit()`** is called.

POP3.apop(*user*, *secret*)

Utiliza la autenticación APOP (más segura) para registrar en el servidor POP3.

POP3.rpop(*user*)

Utiliza autenticación RPOP (similar a los comandos r de UNIX) para registrar en el servidor POP3.

POP3.stat()

Obtiene el estado del buzón de correo. El resultado es una tupla de 2 enteros: `(conteo de mensaje, tamaño del buzón de correo)`.

POP3.list([*which*])

Solicita lista de mensajes, el resultado es en la forma `(respuesta, ['mesg_num octets', ...], octets)`. Si *which* está establecido, es el mensaje a listar.

POP3.retr(*which*)

Recupera el número de mensaje completo *which*, y establece marca de visto. El resultado es en la forma `(respuesta, ['line', ...], octets)`.

POP3.dele(*which*)

Marca el número de mensaje *which* para eliminación. En la mayoría de los servidores las eliminaciones no están actualmente presentadas hasta QUIT (la mayor excepción es Eudora QPOP, que deliberadamente viola las RFC haciendo eliminaciones pendientes en cada desconexión).

POP3.rset()

Remueve las marcas de eliminación para el buzón de correo.

POP3.noop()

No hace nada. Puede ser utilizado como keep-alive.

POP3.quit()

Cierra sesión: envía los cambios, desbloquea el buzón de correo, desconecta.

POP3.top(*which, howmuch*)

Recupera la cabecera del mensaje mas *howmuch* las líneas del mensaje después del cabecera del número de mensajes *which*. El resultado es en la forma `(respuesta, ['línea', ...]. octets)`.

El comando TOP POP3 que este método utiliza, a diferencia del comando RETR, no establece la marca de visto del mensaje; desafortunadamente, TOP está pobremente especificado en las RFC y se rompe con frecuencia en servidores off-brand. Pruebe este método a mano contra los servidores POP3 que usted utilizará antes de confiar en él.

POP3.uidl(*which=None*)

Retorna la lista del resumen de mensajes (id único). Si *which* es especificado, el resultado contiene el id único para ese mensaje en la forma `'response mesgnum uid`, de otra forma el resultado es una lista `(respuesta, ['mengnum uid', ...], octets)`.

POP3.utf8()

Trata de cambiar al modo UTF-8. Retorna la respuesta del servidor si es exitosa, genera **`error_proto`** si no. Especificado en **RFC 6856** [https://datatracker.ietf.org/doc/html/rfc6856.html].

Added in version 3.5.

POP3.stls(*context=None*)

Comienza una sesión TLS en la conexión activa como está especificado en **RFC 2595** [https://datatracker.ietf.org/doc/html/rfc2595.html]. Esto es únicamente permitido antes de la autenticación de usuario

El parámetro *context* es un objeto **`ssl.SSLContext`** que permite empaquetar opciones de configuración SSL, certificados y llaves privadas en una única (potencialmente longeva) estructura. Por favor lea Consideraciones de seguridad para buenas prácticas.

This method supports hostname checking via **`ssl.SSLContext.check_hostname`** and *Server Name Indication* (see **`ssl.HAS_SNI`**).

Added in version 3.4.

Instancias de **`POP3_SSL`** no tienen métodos adicionales. La interfaz de esta subclase es idéntica a su padre.

Ejemplo POP3

Este es un ejemplo mínimo (sin chequeo de errores) que abre un buzón de correo y retorna e imprime todos los mensajes:

```
import getpass, poplib

M = poplib.POP3('localhost')
M.user(getpass.getuser())
M.pass_(getpass.getpass())
numMessages = len(M.list()[1])
for i in range(numMessages):
    for j in M.retr(i+1)[1]:
        print(j)
```

Al final del módulo, hay una sección de test que contiene un ejemplo más extensivo de uso.

`imaplib` — IMAP4 protocol client

Código fuente: Lib/imaplib.py [https://github.com/python/cpython/tree/3.14/Lib/imaplib.py]

Este módulo define tres clases **`IMAP4`**, **`IMAP4_SSL`** y **`IMAP4_stream`**, que encapsula una conexión a un servidor IMAP4 e implementa un gran subconjunto del protocolo de cliente IMAP4rev1 como se define en :rfc :*2060*. Es compatible con los servidores IMAP4 (**RFC 1730** [https://datatracker.ietf.org/doc/html/rfc1730.html]), pero tenga en cuenta que el comando `STATUS` no es compatible con IMAP4.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

Three classes are provided by the **`imaplib`** module, **`IMAP4`** is the base class:

class imaplib.IMAP4(*host=''*, *port=IMAP4_PORT*, *timeout=None*)

This class implements the actual IMAP4 protocol. The connection is created and protocol version (IMAP4 or IMAP4rev1) is determined when the instance is initialized. If *host* is not specified, `''` (the local host) is used. If *port* is omitted, the standard IMAP4 port (143) is used. The optional *timeout* parameter specifies a timeout in seconds for the connection attempt. If timeout is not given or is `None`, the global default socket timeout is used.

La clase **`IMAP4`** soporta la sentencia **`with`**. Cuando se usa de esta manera, el comando IMAP4 `LOGOUT` se emite automáticamente cuando se cierra la declaración **`with`**. P.ej.:

```
>>> from imaplib import IMAP4
>>> with IMAP4("domain.org") as M:
...     M.noop()
...
('OK', [b'Nothing Accomplished. d25if65hy903weo.87'
```

Distinto en la versión 3.5: Se agregó soporte para la sentencia **`with`**.

Distinto en la versión 3.9: El parámetro opcional *timeout* fue agregado.

Se definen tres excepciones como atributos de la clase **`IMAP4`**:

exception IMAP4.error

Excepción lanzada por cualquier error. El motivo de la excepción se pasa al constructor como una cadena de caracteres.

exception IMAP4.abort

Los errores del servidor IMAP4 causan que esta excepción sea lanzada. Esta es una subclase de **`IMAP4.error`**. Tenga en cuenta que cerrar la instancia e instanciar una nueva generalmente permitirá la recuperación de esta excepción.

exception IMAP4.readonly

Esta excepción es lanzada cuando el servidor cambia el estado de un buzón de correo de escritura. Esta es una subclase de **`IMAP4.error`**. Algún otro cliente ahora tiene permiso de escritura y será necesario volver a abrir el buzón para volver a obtener el permiso de escritura.

También hay una subclase para conexiones seguras:

class imaplib.IMAP4_SSL(*host=''*, *port=IMAP4_SSL_PORT*, ***, *ssl_context=None*, *timeout=None*)

Esta es una subclase derivada de **`IMAP4`** que se conecta a través de un socket cifrado SSL (para usar esta clase necesita un módulo de socket que se compiló con soporte SSL). Si no se especifica *host*, se usa `''` (el host local). Si se omite *port*, se usa el puerto IMAP4 estándar sobre SSL (993). *ssl_context* es un objeto **`ssl.SSLContext`** que permite agrupar opciones de configuración SSL, certificados y claves privadas en una estructura única (potencialmente de larga duración). Leer Consideraciones de seguridad para conocer las mejores prácticas.

The optional *timeout* parameter specifies a timeout in seconds for the connection attempt. If timeout is not given or is `None`, the global default socket timeout is used.

Distinto en la versión 3.3: El parámetro *ssl_context* fue agregado.

Distinto en la versión 3.4: La clase ahora admite la verificación del nombre de host con **`ssl.SSLContext.check_hostname`** y *Server Name Indication* (ver **`ssl.HAS_SNI`**).

Distinto en la versión 3.9: El parámetro opcional *timeout* fue agregado.

Distinto en la versión 3.12: Se han eliminado los parámetros obsoletos *keyfile* y *certfile*.

La segunda subclase permite conexiones creadas por un proceso hijo:

class imaplib.IMAP4_stream(*command*)

Esta es una subclase derivada de **`IMAP4`** que se conecta a los descriptores de archivo `stdin/stdout` creados al pasar *command* a `subprocess.Popen ()`.

Se definen las siguientes funciones de utilidad:

imaplib.Internaldate2tuple(*datestr*)

Analiza una cadena de caracteres IMAP4 `INTERNALDATE` y retorna la hora local correspondiente. El valor de retorno es una tupla **`time.struct_time`** o `None` si la cadena de caracteres tiene un formato incorrecto.

imaplib.Int2AP(*num*)

Convierte un número entero en una representación de bytes utilizando caracteres del conjunto [`A` .. `P`].

imaplib.ParseFlags(*flagstr*)

Convierte una respuesta IMAP4 `FLAGS` a una tupla de indicadores individuales.

imaplib.Time2Internaldate(*date_time*)

Convierte *date_time* en una representación IMAP4 `INTERNALDATE`. El valor de retorno es una cadena de caracteres en la forma: `"DD-Mmm-YYYY HH:MM:SS +HHMM"` (incluyendo comillas dobles). El argumento *date_time* puede ser un número (int o float) que representa segundos en un espacio de tiempo (como lo retorna `time.time()`), una tupla de 9 que representa la hora local como una instancia de `time.struct_time` (según lo retornado por `time.localtime()`), una instancia actualizada de `datetime.datetime`, o una cadena de caracteres entre comillas dobles. En el último caso, se supone que ya está en el formato correcto.

Tenga en cuenta que los números de mensaje IMAP4 cambian a medida que cambia el buzón de correo; en particular, después de que un comando `EXPUNGE` realiza eliminaciones, los mensajes restantes se vuelven a numerar. Por lo tanto, es muy recomendable usar UIDs en su lugar, con el comando UID.

Al final del módulo, hay una sección de prueba que contiene un ejemplo más extenso de uso.

Ver también

Documentos describiendo el protocolo, y fuentes de servidores que lo implementan, del Centro de Información IMAP de la Universidad de Washington, pueden ser encontrados en (**Código**

Fuente) https://github.com/uw-imap/imap (**Fuera de mantención**).

Objetos de IMAP4

Todos los comandos IMAP4rev1 están representados por métodos del mismo nombre, ya sea en mayúsculas o minúsculas.

Todos los argumentos de los comandos se convierten en cadenas de caracteres, excepto `AUTENTICATE` y el último argumento de `APPEND` que se pasa como un literal IMAP4. Si es necesario (la cadena de caracteres contiene caracteres sensibles al protocolo IMAP4 y no está entre paréntesis ni comillas dobles), se cita cada cadena. Sin embargo, siempre se cita el argumento *password* para el comando `LOGIN`. Si desea evitar que se cite una cadena de argumento (por ejemplo: el argumento *flags* para `STORE`), encierre la cadena entre paréntesis (por ejemplo: `r'(\Deleted)'`).

Most commands return a tuple: `(type, [data, ...])` where *type* is usually `'OK'` or `'NO'`, and *data* is either the text from the command response, or mandated results from the command. Each *data* is either a `bytes`, or a tuple. If a tuple, then the first part is the header of the response, and the second part contains the data (ie: “literal” value).

Las opciones *message_set* de los siguientes comandos son una cadena de caracteres que especifica uno o más mensajes sobre los que se debe actuar. Puede ser un número de mensaje simple (`'1'`), un rango de números de mensaje (`'2:4'`) o un grupo de rangos no contiguos separados por comas (`'1:3,6:9'`). Un rango puede contener un asterisco para indicar un límite superior infinito (`'3:*'`).

Una instancia de **`IMAP4`** tiene los siguientes métodos:

IMAP4.append(*mailbox*, *flags*, *date_time*, *message*)

Agregar *mensaje* al buzón de correo con nombre.

IMAP4.authenticate(*mechanism*, *authobject*)

Autenticar comando — requiere procesamiento de respuesta.

mechanism especifica qué mecanismo de autenticación se utilizará; debe aparecer en la variable de instancia `capabilities` en la forma `AUTH=mechanism`.

authobject debe ser un objeto invocable:

```
data = authobject(response)
```

Se llamará para procesar las respuestas de continuación del servidor; el argumento *response* que se pasa será `bytes`. Debería retornar `bytes` *data* que se codificarán en base64 y se enviarán al servidor. Debería retornar `None` si la respuesta de cancelación de cliente `*` se debe enviar en su lugar.

Distinto en la versión 3.5: los nombres de usuario y las contraseñas de cadena de caracteres ahora están codificados para `utf-8` en lugar de limitarse a ASCII.

IMAP4.check()

Control del buzón de correo en el servidor.

IMAP4.close()

Cerrar el buzón de correo seleccionado actualmente. Los mensajes eliminados se eliminan del buzón de correo de escritura. Este es el comando recomendado antes de `LOGOUT`.

IMAP4.copy(*message_set*, *new_mailbox*)

Copia mensajes *message_set* al final de *new_mailbox*.

IMAP4.create(*mailbox*)

Crea un nuevo buzón de correo llamado *mailbox*.

IMAP4.delete(*mailbox*)

Elimina el buzón de correo antiguo llamado *mailbox*.

IMAP4.deleteacl(*mailbox*, *who*)

Elimina las ACLs (elimina cualquier derecho) establecidas para quién en el buzón de correo.

IMAP4.enable(*capability*)

Habilita *capability* (ver **RFC 5161** [https://datatracker.ietf.org/doc/html/rfc5161.html]). La mayoría de las capacidades no necesitan estar habilitadas. Actualmente solo esta soportada la capacidad `UTF8=ACCEPT` (consulte **RFC 6855** [https://datatracker.ietf.org/doc/html/rfc6855.html]).

Added in version 3.5: El método **`enable()`** en sí, y soporte **RFC 6855** [https://datatracker.ietf.org/doc/html/rfc6855.html].

IMAP4.expunge()

Elimina permanentemente los elementos eliminados del buzón de correo seleccionado. Genera una respuesta `EXPUNGE` para cada mensaje eliminado. Los datos retornados contienen una lista de números de mensaje `EXPUNGE` en el orden recibido.

IMAP4.fetch(*message_set, message_parts*)

Obtiene (partes de) mensajes. *message_parts* debe ser una cadena de nombres de partes de mensajes encerrados entre paréntesis, por ejemplo: `"(UID BODY[TEXT])"`. Los datos retornados son una tupla de mensaje parte sobre y datos.

IMAP4.getacl(*mailbox*)

Obtiene la `ACLs` para *mailbox*. El método no es estándar, pero es compatible con el servidor `Cyrus`.

IMAP4.getannotation(*mailbox, entry, attribute*)

Recupera la `ANNOTATIONs` especificada para *mailbox*. El método no es estándar, pero es compatible con el servidor `Cyrus`.

IMAP4.getquota(*root*)

Obtiene el uso y los límites de los recursos de la `quota` de

root. Este método es parte de la extensión IMAP4 QUOTA definida en rfc2087.

IMAP4.getquotaroot(*mailbox*)

Obtiene la lista de `quota roots` para el nombrado *mailbox*. Este método es parte de la extensión IMAP4 QUOTA definida en rfc2087.

IMAP4.idle(*duration=None*)

Return an **`Idler`**: an iterable context manager implementing the IMAP4 `IDLE` command as defined in **RFC 2177** [https://datatracker.ietf.org/doc/html/rfc2177.html].

The returned object sends the `IDLE` command when activated by the **`with`** statement, produces IMAP untagged responses via the iterator protocol, and sends `DONE` upon context exit.

All untagged responses that arrive after sending the `IDLE` command (including any that arrive before the server acknowledges the command) will be available via iteration. Any leftover responses (those not iterated in the **`with`** context) can be retrieved in the usual way after `IDLE` ends, using **`IMAP4.response()`**.

Responses are represented as `(type, [data, ...])` tuples, as described in IMAP4 Objects.

The *duration* argument sets a maximum duration (in seconds) to keep idling, after which any ongoing iteration will stop. It can be an **`int`** or **`float`**, or `None` for no time limit. Callers wishing to avoid inactivity timeouts on servers that impose them should keep this at most 29 minutes (1740 seconds). Requires a socket connection; *duration* must be `None` on **`IMAP4_stream`** connections.

```
>>> with M.idle(duration=29 * 60) as idler:
...     for typ, data in idler:
...         print(typ, data)
```

```
...
EXISTS [b'1']
RECENT [b'1']
```

Idler.burst(*interval=0.1*)

Yield a burst of responses no more than *interval* seconds apart (expressed as an **`int`** or **`float`**).

This generator is an alternative to iterating one response at a time, intended to aid in efficient batch processing. It retrieves the next response along with any immediately available subsequent responses. (For example, a rapid series of `EXPUNGE` responses after a bulk delete.)

Requires a socket connection; does not work on **`IMAP4_stream`** connections.

```
>>> with M.idle() as idler:
...     # get a response and any others followi
...     batch = list(idler.burst())
...     print(f'processing {len(batch)} respons
...     print(batch)
...
processing 3 responses...
[('EXPUNGE', [b'2']), ('EXPUNGE', [b'1']), ('RE
```

Truco

The `IDLE` context's maximum duration, as passed to **`IMAP4.idle()`**, is respected when waiting for the first response in a burst. Therefore, an expired **`Idler`** will cause this generator to return immediately without producing anything. Callers should consider this if using it in a loop.

Nota

The iterator returned by **`IMAP4.idle()`** is usable only

within a `with` statement. Before or after that context, unsolicited responses are collected internally whenever a command finishes, and can be retrieved with `IMAP4.response()`.

Nota

The `Idler` class name and structure are internal interfaces, subject to change. Calling code can rely on its context management, iteration, and public method to remain stable, but should not subclass, instantiate, compare, or otherwise directly reference the class.

Added in version 3.14.

IMAP4.list([*directory*[, *pattern*]])

Lista los nombres de buzones de correo en *directory* coincidiendo *pattern*. *directory* por defecto es la carpeta de correo de nivel superior, y *pattern* por defecto coincide con cualquier cosa. Los datos retornados contienen una lista de respuestas `LIST`.

IMAP4.login(*user*, *password*)

Identifica al cliente con una contraseña de texto sin formato. El *password* será citado.

IMAP4.login_cram_md5(*user*, *password*)

Fuerza el uso de la autenticación `CRAM-MD5` al identificar al cliente para proteger la contraseña. Solo funcionará si la respuesta `CAPABILITY` del servidor incluye la frase `AUTH=CRAM-MD5`.

Distinto en la versión 3.14: An `IMAP4.error` is raised if MD5 support is not available.

IMAP4.logout()

Cierra la conexión al servidor. Retorna la respuesta `BYE` desde el servidor .

Distinto en la versión 3.8: El método ya no ignora las excepciones silenciosamente arbitrarias.

IMAP4.lsub(*directory*='""', *pattern*='*')

Lista los nombres de buzones de correos suscritos en el patrón de coincidencia del directorio *directory* por defecto para el directorio de nivel superior y *pattern* por defecto para que coincida con cualquier buzón de correo. Los datos retornados son una tupla de mensaje parte sobre y datos.

IMAP4.myrights(*mailbox*)

Muestra mis ACLs para un buzón de correo (es decir, los derechos que tengo sobre el buzón de correo).

IMAP4.namespace()

Retorna espacios de nombres IMAP como se define en **RFC 2342** [https://datatracker.ietf.org/doc/html/rfc2342.html].

IMAP4.noop()

Envía `NOOP` al servidor.

IMAP4.open(*host*, *port*, *timeout*=*None*)

Opens socket to *port* at *host*. The optional *timeout* parameter specifies a timeout in seconds for the connection attempt. If timeout is not given or is `None`, the global default socket timeout is used. Also note that if the *timeout* parameter is set to be zero, it will raise a **`ValueError`** to reject creating a non-blocking socket. This method is implicitly called by the **`IMAP4`** constructor. The connection objects established by this method will be used in the **`IMAP4.read()`**, **`IMAP4.readline()`**, **`IMAP4.send()`**, and **`IMAP4.shutdown()`** methods. You may override this method.

Genera un evento de auditoría `imaplib.open` con argumentos `self, host, port`.

Distinto en la versión 3.9: El parámetro *timeout* fue agregado.

IMAP4.partial(*message_num, message_part, start, length*)

Obtiene partes truncadas de un mensaje. Los datos retornados son una tupla de mensaje parte sobre y datos.

IMAP4.proxyauth(*user*)

Asume la autenticación como *user*. Permite a un administrador autorizado hacer un proxy en el buzón de correo de cualquier usuario.

IMAP4.read(*size*)

Lee *size* bytes del servidor remoto. Podemos sobrescribir este método.

IMAP4.readline()

Lee una línea del servidor remoto. Podemos sobrescribir este método.

IMAP4.recent()

Solicita al servidor una actualización. Los datos retornados son `None` si no hay mensajes nuevos, de lo contrario el valor de respuesta es `RECENT`.

IMAP4.rename(*oldmailbox, newmailbox*)

Cambia el nombre del buzón de correo llamado *oldmailbox* a *newmailbox*.

IMAP4.response(*code*)

Retorna los datos para la respuesta *code* si se recibió, o `None`. Retorna el código dado, en lugar del tipo habitual.

IMAP4.search(*charset, criterion*[, ...])

Busca en el buzón de correo mensajes coincidentes. El *charset* puede ser `None`, en cuyo caso no se especificará `CHARSET` en la solicitud al servidor. El protocolo IMAP requiere que se especifique al menos un criterio; se lanzará una excepción cuando el servidor retorne un error. *charset* debe ser `None` si la capacidad `UTF8=ACCEPT` se habilitó utilizando el comando **`enable()`**.

Ejemplo:

```
# M is a connected IMAP4 instance...
typ, msgnums = M.search(None, 'FROM', '"LDJ"')

# or:
typ, msgnums = M.search(None, '(FROM "LDJ")')
```

IMAP4.select(*mailbox='INBOX'*, *readonly=False*)

Seleccione un buzón de correo. Los datos retornados son el recuento de mensajes en *mailbox* (respuesta `EXISTS`). El *mailbox* predeterminado es `'INBOX'`. Si se establece el indicador *readonly*, no se permiten modificaciones en el buzón de correo.

IMAP4.send(*data*)

Envía `data` al servidor remoto. Podemos sobrescribir este método.

Lanza un evento de auditoría `imaplib.send` con argumentos `self`, `data`.

IMAP4.setacl(*mailbox*, *who*, *what*)

Establece una `ACL` para *mailbox*. El método no es estándar, pero es compatible con el servidor `Cyrus`.

IMAP4.setannotation(*mailbox*, *entry*, *attribute*[, ...])

Establece `ANNOTATION`s para *mailbox*. El método no es estándar, pero es compatible con el servidor `Cyrus`.

IMAP4.setquota(*root*, *limits*)

Establece los recursos *limits* de la `quota` de los *root*. Este método es parte de la extensión IMAP4 QUOTA definida en rfc2087.

IMAP4.shutdown()

Cierra la conexión establecida en `open`. Este método es llamado implícitamente por **`IMAP4.logout()`**. Podemos sobrescribir este método.

IMAP4.socket()

Retorna la instancia de socket utilizada para conectarse al servidor.

IMAP4.sort(*sort_criteria*, *charset*, *search_criterion*[, ...])

El comando `sort` es una variante de `search` con semántica de clasificación para los resultados. Los datos retornados contienen una lista separada por espacios de números de mensajes coincidentes.

Sort tiene dos argumentos antes del argumento (s) *search_criterion*; una lista entre paréntesis de *sort_criteria*, y la búsqueda del *charset*. Tenga en cuenta que, a diferencia de `search`, el argumento de búsqueda *charset* es obligatorio. También hay un comando `uid sort` que corresponde a `sort` de la misma manera que `uid search` corresponde a `search`. El comando `sort` primero busca en el buzón de correo mensajes que coincidan con los criterios de búsqueda dados utilizando el argumento *charset* para la interpretación de cadenas de caracteres en los criterios de búsqueda. Luego retorna los números de mensajes coincidentes.

Este es un comando de extensión `IMAP4rev1`.

IMAP4.starttls(*ssl_context=None*)

Envía un comando `STARTTLS`. El argumento *ssl_context* es opcional y debe ser un objeto **`ssl.SSLContext`**. Esto habilitará el cifrado en la conexión IMAP. Leer Consideraciones de seguridad para conocer las mejores prácticas.

Added in version 3.2.

Distinto en la versión 3.4: El método ahora admite la verificación del nombre de host con `ssl.SSLContext.check_hostname` y *Server Name Indication* (ver `ssl.HAS_SNI`).

IMAP4.status(*mailbox*, *names*)

Solicita condiciones de estado con nombre para *mailbox*.

IMAP4.store(*message_set*, *command*, *flag_list*)

Altera las disposiciones de los indicadores para los mensajes en el buzón de correo. *command* esta especificado en la sección 6.4.6 de **RFC 2060** [https://datatracker.ietf.org/doc/html/rfc2060.html] siendo como uno de «FLAGS», «+FLAGS» o «-FLAGS», opcionalmente con un sufijo «.SILENT».

Por ejemplo, para establecer el indicador de eliminación en todos los mensajes:

```
typ, data = M.search(None, 'ALL')
for num in data[0].split():
   M.store(num, '+FLAGS', '\\Deleted')
M.expunge()
```

Nota

Creating flags containing "]" (for example: «[test]») violates **RFC 3501** [https://datatracker.ietf.org/doc/html/rfc3501.html] (the IMAP protocol). However, imaplib has historically allowed creation of such tags, and popular IMAP servers, such as Gmail, accept and produce such flags. There are non-Python programs which also create such tags. Although it is an RFC violation and IMAP clients and servers are supposed to be strict, imaplib still continues to allow such tags to be created for backward compatibility reasons, and as of Python 3.6, handles them if they are sent from the server, since this improves realworld compatibility.

IMAP4.subscribe(*mailbox*)

Suscribe al nuevo buzón de correo.

IMAP4.thread(*threading_algorithm*, *charset*, *search_criterion*[, ...])

El comando `thread` es una variante de `search` con semántica de hilos para los resultados. Los datos retornados contienen una lista de miembros de hilos separados por espacios.

Los miembros de del hilo (*thread*) consisten en cero o más números de mensajes, delimitados por espacios, que indican sucesivos padres e hijos.

Thread tiene dos argumentos antes del argumento(s) *search_criterion*; un *threading_algorithm*, y la búsqueda del *charset*. Tenga en cuenta que, a diferencia de `search`, el argumento de búsqueda *charset* es obligatorio. También hay un comando `uid thread` que corresponde a `thread` de la misma manera que `uid search` corresponde a `search`. El comando `thread` primero busca en el buzón de correo mensajes que coincidan con los criterios de búsqueda dados utilizando el argumento *charset* para la interpretación de cadenas de caracteres en los criterios de búsqueda. Luego retorna los mensajes coincidentes enfilados según el algoritmo de subproceso especificado.

Este es un comando de extensión `IMAP4rev1`.

IMAP4.uid(*command*, *arg*[, ...])

Ejecuta argumentos de comando con mensajes identificados por UID, en lugar de número de mensaje. Retorna la respuesta apropiada al comando. Se debe proporcionar al menos un argumento; Si no se proporciona ninguno, el servidor retornará un error y se lanzará una excepción.

IMAP4.unsubscribe(*mailbox*)

Darse de baja del antiguo buzón de correo.

IMAP4.unselect()

`imaplib.IMAP4.unselect()` libera recursos del servidor asociados al buzón de correo seleccionado y devuelve el servidor al estado autenticado. Este comando realiza las mismas acciones que **`imaplib.IMAP4.close()`**, con la excepción de que ningún mensaje es permanentemente borrado del buzón de correo actualmente seleccionado.

Added in version 3.9.

IMAP4.xatom(*name*[, ...])

Permite comandos de extensión simples notificados por el servidor en la respuesta `CAPABILITY`.

Los siguientes atributos se definen en instancias de **`IMAP4`**:

IMAP4.PROTOCOL_VERSION

El protocolo mas recientemente admitido en la respuesta `CAPABILITY` desde el servidor.

IMAP4.debug

Valor entero para controlar la salida de depuración. El valor de inicialización se toma de la variable del módulo `Debug`. Valores mayores de tres rastrean cada comando.

IMAP4.utf8_enabled

Valor booleano que normalmente es `False`, pero se establece en `True` si un comando **`enable()`** es exitosamente emitido para la capacidad `UTF8=ACCEPT`.

Added in version 3.5.

Ejemplo IMAP4

Aquí hay un ejemplo mínimo (sin verificación de errores) que abre un buzón de correo y recupera e imprime todos los mensajes:

```
import getpass, imaplib

M = imaplib.IMAP4(host='example.org')
M.login(getpass.getuser(), getpass.getpass())
M.select()
typ, data = M.search(None, 'ALL')
for num in data[0].split():
    typ, data = M.fetch(num, '(RFC822)')
    print('Message %s\n%s\n' % (num, data[0][1]))
M.close()
M.logout()
```

`smtplib` — SMTP protocol client

Código fuente: Lib/smtplib.py [https://github.com/python/cpython/tree/3.14/Lib/smtplib.py]

The `smtplib` module defines an SMTP client session object that can be used to send mail to any internet machine with an SMTP or ESMTP listener daemon. For details of SMTP and ESMTP operation, consult **RFC 821** [https://datatracker.ietf.org/doc/html/rfc821.html] (Simple Mail Transfer Protocol) and **RFC 1869** [https://datatracker.ietf.org/doc/html/rfc1869.html] (SMTP Service Extensions).

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

class smtplib.SMTP(*host=''*, *port=0*, *local_hostname=None*, [*timeout*,]*source_address=None*)

> Una instancia `SMTP` encapsula una conexión SMTP. Tiene métodos que admiten un repertorio completo de operaciones SMTP y ESMTP. Si se proporcionan los parámetros de *port* y *host* opcionales, se llama al método SMTP `connect()` con esos parámetros durante la inicialización. Si se especifica, *local_hostname* se utiliza como FQDN del host local en el comando HELO /EHLO. De lo contrario, el nombre de host local se encuentra mediante `socket.getfqdn()`. Si la llamada `connect()` retorna algo que no sea un código de éxito, se lanza un `SMTPConnectError`. El parámetro opcional *timeout* especifica un tiempo de espera en segundos para bloquear operaciones como el intento de conexión (si no se especifica, se utilizará la configuración de tiempo de espera global predeterminada). Si expira el tiempo de espera, se

lanza **`TimeoutError`**. El parámetro opcional *source_address* permite la vinculación a alguna dirección de origen específica en una máquina con múltiples interfaces de red y/o algún puerto TCP de origen específico. Se necesita una tupla de 2 (host, port), para que el socket se vincule como su dirección de origen antes de conectarse. Si se omite (o si el *host* o el *port* son `''` y/o 0 respectivamente), se utilizará el comportamiento predeterminado del sistema operativo.

Para un uso normal, solo debe requerir los métodos initialization/connect, **`sendmail()`** y **`SMTP.quit()`**. A continuación se incluye un ejemplo.

La clase **`SMTP`** admite la instrucción **`with`**. Cuando se usa así, el comando SMTP `QUIT` se emite automáticamente cuando la **`with`** sale de la instrucción. por ejemplo:

```
>>> from smtplib import SMTP
>>> with SMTP("domain.org") as smtp:
...     smtp.noop()
...
(250, b'Ok')
>>>
```

Todos los comandos lanzarán un evento de auditoría `smtplib.SMTP.send` con argumentos `self` y `data`, donde `data` son los bytes que están a punto de ser enviado al host remoto.

Distinto en la versión 3.3: Se agregó soporte para la sentencia **`with`**.

Distinto en la versión 3.3: Se agregó el argumento *source_address*.

Added in version 3.5: La extensión SMTPUTF8 (**RFC 6531** [https://datatracker.ietf.org/doc/html/rfc6531.html]) ahora es compatible.

Distinto en la versión 3.9: Si el parámetro *timeout* se define a cero, lanzará un **`ValueError`** para evitar la creación de un socket no bloqueado.

class smtplib.SMTP_SSL(*host=''*, *port=0*, *local_hostname=None*, ***, [*timeout*,]*context=None*, *source_address=None*)

An `SMTP_SSL` instance behaves exactly the same as instances of `SMTP`. `SMTP_SSL` should be used for situations where SSL is required from the beginning of the connection and using `starttls()` is not appropriate. If *host* is not specified, the local host is used. If *port* is zero, the standard SMTP-over-SSL port (465) is used. The optional arguments *local_hostname*, *timeout* and *source_address* have the same meaning as they do in the `SMTP` class. *context*, also optional, can contain a `SSLContext` and allows configuring various aspects of the secure connection. Please read Consideraciones de seguridad for best practices.

Distinto en la versión 3.3: se agregó *contexto*.

Distinto en la versión 3.3: Se agregó el argumento *source_address*.

Distinto en la versión 3.4: La clase ahora admite la verificación del nombre de host con `ssl.SSLContext.check_hostname` y *Server Name Indication* (ver `ssl.HAS_SNI`).

Distinto en la versión 3.9: Si el parámetro *timeout* se define a cero, lanzará un `ValueError` para evitar la creación de un socket no bloqueado

Distinto en la versión 3.12: Los parámetros obsoletos *keyfile y *certifile* se han eliminado.

class smtplib.LMTP(*host=''*, *port=LMTP_PORT*, *local_hostname=None*, *source_address=None*[, *timeout*])

The LMTP protocol, which is very similar to ESMTP, is heavily based on the standard SMTP client. It's common to use Unix sockets for LMTP, so our `connect()` method must support that as well as a regular host:port server. The optional arguments *local_hostname* and *source_address* have the same meaning as they do in the `SMTP` class. To specify a Unix socket, you must use an absolute path for *host*, starting with a "/".

Se admite la autenticación mediante el mecanismo SMTP habitual. Cuando se usa un socket Unix, LMTP generalmente no admite ni requiere autenticación, pero su millaje puede variar.

Distinto en la versión 3.9: Se ha añadido el parámetro opcional *timeout*.

También se define una buena selección de excepciones:

exception smtplib.SMTPException

Subclase de `OSError` que es la clase de excepción base para todas las demás excepciones proporcionadas por este módulo.

Distinto en la versión 3.4: SMTPException se convirtió en subclase de `OSError`

exception smtplib.SMTPServerDisconnected

Esta excepción se genera cuando el servidor se desconecta inesperadamente o cuando se intenta usar la instancia `SMTP` antes de conectarlo a un servidor.

exception smtplib.SMTPResponseException

Base class for all exceptions that include an SMTP error code. These exceptions are generated in some instances when the SMTP server returns an error code.

smtp_code

The error code.

smtp_error

The error message.

exception smtplib.SMTPSenderRefused

Dirección del remitente rechazada. Además de los atributos establecidos por todas las excepciones **`SMTPResponseException`**, éste establece "remitente" para la cadena de caracteres que el servidor SMTP rechazó.

exception smtplib.SMTPRecipientsRefused

All recipient addresses refused.

recipients

A dictionary of exactly the same sort as returned by **`SMTP.sendmail()`** containing the errors for each recipient.

exception smtplib.SMTPDataError

El servidor SMTP se negó a aceptar los datos del mensaje.

exception smtplib.SMTPConnectError

Se produjo un error durante el establecimiento de conexión con el servidor.

exception smtplib.SMTPHeloError

El servidor rechazó nuestro mensaje `HELO`.

exception smtplib.SMTPNotSupportedError

El servidor no admite el comando o la opción que se intentó.

Added in version 3.5.

exception smtplib.SMTPAuthenticationError

La autenticación SMTP salió mal. Lo más probable es que el servidor no aceptó la combinación proporcionada de username/password.

Ver también

RFC 821 [https://datatracker.ietf.org/doc/html/rfc821.html] - Protocolo Simple de Transferencia

Definición de protocolo para SMTP. Este documento cubre el modelo, el procedimiento operativo y los detalles del protocolo para SMTP.

RFC 1869 [https://datatracker.ietf.org/doc/html/rfc1869.html] - Extensiones de Servicio SMTP

Definición de las extensiones ESMTP para SMTP. Esto describe un marco para extender SMTP con nuevos comandos, que admite el descubrimiento dinámico de los comandos proporcionados por el servidor y define algunos comandos adicionales.

Objetos SMTP

Una instancia `SMTP` tiene los siguientes métodos:

SMTP.set_debuglevel(*level*)

Establezca el nivel de salida de depuración. Un valor de 1 o `True` para *level* da como resultado mensajes de depuración para la conexión y para todos los mensajes enviados y recibidos desde el servidor. Un valor de 2 para *level* da como resultado que estos mensajes tengan una marca de tiempo.

Distinto en la versión 3.5: Se agregó el nivel de depuración 2.

SMTP.docmd(*cmd*, *args*=*''*)

Envía un comando *cmd* al servidor. El argumento opcional *args* simplemente se concatena con el comando, separado por un espacio.

Esto devuelve una tupla de 2 compuestos por un código de respuesta numérico y la línea de respuesta real (las respuestas de varias líneas se unen en una línea larga).

En funcionamiento normal, no debería ser necesario llamar a este método explícitamente. Se utiliza para implementar otros métodos y puede resultar útil para probar extensiones privadas.

Si se pierde la conexión con el servidor mientras se espera la respuesta, se activará **`SMTPServerDisconnected`**.

SMTP.connect(*host='localhost'*, *port=0*)

Conéctese a un host en un puerto determinado. Los valores predeterminados son para conectarse al host local en el puerto SMTP estándar (25). Si el nombre de host termina con dos puntos (`':'`) seguido de un número, ese sufijo se eliminará y el número se interpretará como el número de puerto a utilizar. El constructor invoca automáticamente este método si se especifica un host durante la instanciación. Devuelve una tupla de 2 del código de respuesta y el mensaje enviado por el servidor en su respuesta de conexión.

Genera un evento de auditoría `smtplib.connect` con argumentos `self`, `host`, `port`.

SMTP.helo(*name=''*)

Identifíquese en el servidor SMTP usando `HELO`. El argumento del nombre de host tiene como valor predeterminado el nombre de dominio completo del host local. El mensaje devuelto por el servidor se almacena como el atributo **`helo_resp`** del objeto.

En funcionamiento normal, no debería ser necesario llamar a este método explícitamente. Será llamado implícitamente por **`sendmail()`** cuando sea necesario.

SMTP.ehlo(*name=''*)

Identifíquese en un servidor ESMTP mediante `EHLO`. El argumento del nombre de host tiene como valor predeterminado el nombre de dominio completo del host local. Examine la respuesta para la opción ESMTP y guárdelos para que los utilice **`has_extn()`**. También establece varios atributos informativos: el mensaje retornado por el servidor se almacena como el atributo **`ehlo_resp`**, **`does_esmtp`** se establece en `True` o `False` dependiendo de si el servidor admite ESMTP, y **`esmtp_features`** será un diccionario que contiene los nombres de las extensiones de servicio SMTP de este servidor soportes, y sus parámetros (si los hay).

A menos que desee utilizar `has_extn()` antes de enviar correo, no debería ser necesario llamar a este método explícitamente. Se llamará implícitamente por `sendmail()` cuando sea necesario.

SMTP.ehlo_or_helo_if_needed()

Este método llama a `ehlo()` o `helo()` si no ha habido ningún comando `EHLO` o `HELO` anterior en esta sesión. Primero prueba ESMTP `EHLO`.

`SMTPHeloError`
El servidor no respondió correctamente al saludo `HELO`.

SMTP.has_extn(*name*)

Retorna `True` si *name* está en el conjunto de extensiones de servicio SMTP devueltas por el servidor, `False` en caso contrario. El método es insensible a la presencia de mayúsculas en *name*.

SMTP.verify(*address*)

Verifique la validez de una dirección en este servidor usando SMTP `VRFY`. Retorna una tupla que consta del código 250 y una dirección completa **RFC 822** [https://datatracker.ietf.org/doc/html/rfc822.html] (incluido el nombre humano) si la dirección del usuario es válida. De lo contrario, devuelve un código de error SMTP de 400 o más y una cadena de error.

Nota

Muchos sitios desactivan SMTP `VRFY` para frustrar a los spammers.

SMTP.login(*user*, *password*, ***, *initial_response_ok=True*)

Inicie sesión en un servidor SMTP que requiera autenticación.

Los argumentos son el nombre de usuario y la contraseña para autenticarse. Si no ha habido ningún comando `EHLO` o `HELO` anterior en esta sesión, este método prueba primero ESMTP `EHLO`. Este método regresará normalmente si la autenticación fue exitosa o puede generar las siguientes excepciones:

`SMTPHeloError`
: El servidor no respondió correctamente al saludo `HELO`.

`SMTPAuthenticationError`
: El servidor no aceptó la combinación de nombre de username/password.

`SMTPNotSupportedError`
: El servidor no admite el comando `AUTH`.

`SMTPException`
: No se encontró ningún método de autenticación adecuado.

Each of the authentication methods supported by **`smtplib`** are tried in turn if they are advertised as supported by the server. See **`auth()`** for a list of supported authentication methods. *initial_response_ok* is passed through to **`auth()`**.

El argumento de palabra clave opcional *initial_response_ok* especifica si, para los métodos de autenticación que lo admiten, se puede enviar una «respuesta inicial» como se especifica en RFC 4954 [https://datatracker.ietf.org/doc/html/rfc4954.html] junto con el comando `AUTH`, en lugar de requerir un desafío/respuesta .

Distinto en la versión 3.5: **`SMTPNotSupportedError`** se puede generar y se agregó el parámetro *initial_response_ok*.

SMTP.auth(*mechanism*, *authobject*, *, *initial_response_ok*=*True*)

Emita un comando `SMTP AUTH` para el *mechanism* de autenticación especificado y maneje la respuesta de desafío a través de *authobject*.

mechanism especifica qué mecanismo de autenticación se utilizará como argumento para el comando `AUTH`; los valores válidos son los enumerados en el elemento `auth` de **`esmtp_features`**.

authobject debe ser un objeto que se pueda invocar y que tome un único argumento opcional:

```
data = authobject(challenge=None)
```

Si la verificación de respuesta inicial devuelve `None`, o si *initial_response_ok* es falso, se llamará a `authobject()` para procesar la respuesta de desafío del servidor; el argumento *challenge* que se pasa será un `bytes`. Debería devolver *data* ASCII `str` que serán codificados en base64 y enviados al servidor.

Si la verificación de respuesta inicial devuelve `None`, o si *initial_response_ok* es falso, se llamará a `authobject()` para procesar la respuesta de desafío del servidor; el argumento *challenge* que se pasa será un `bytes`. Debería devolver *data* ASCII `str` que serán codificados en base64 y enviados al servidor.

La clase `SMTP` proporciona `authobjects` para los mecanismos `CRAM-MD5`, `PLAIN` y `LOGIN`; se denominan `SMTP.auth_cram_md5`, `SMTP.auth_plain` y `SMTP.auth_login` respectivamente. Todos requieren que las propiedades de `user` y `password` de la instancia `SMTP` se establezcan en los valores adecuados.

User code does not normally need to call `auth` directly, but can instead call the **`login()`** method, which will try each of the above mechanisms in turn, in the order listed. `auth` is exposed to facilitate the implementation of authentication methods not (or not yet) supported directly by **`smtplib`**.

Added in version 3.5.

SMTP.starttls(*, *context=None*)

Ponga la conexión SMTP en modo TLS (Seguridad de la capa de transporte). Todos los comandos SMTP que siguen se cifrarán. Entonces deberías llamar a `ehlo()` de nuevo.

Si se proporcionan *keyfile* y *certfile*, se utilizan para crear una `ssl.SSLContext`.

El parámetro *context* opcional es un objeto `ssl.SSLContext`; Esta es una alternativa al uso de un archivo de claves y un archivo de certificado y, si se especifica, tanto *keyfile* como *certfile* deben ser `None`.

Si no ha habido ningún comando `EHLO` o `HELO` anterior en esta sesión, este método intenta ESMTP `EHLO` primero.

Distinto en la versión 3.12: Los parámetros obsoletos *keyfile y *certifile* se han eliminado.

`SMTPHeloError`
El servidor no respondió correctamente al saludo `HELO`.

`SMTPNotSupportedError`
El servidor no admite la extensión STARTTLS.

`RuntimeError`
La compatibilidad con SSL/TLS no está disponible para su intérprete de Python.

Distinto en la versión 3.3: se agregó *contexto*.

Distinto en la versión 3.4: The method now supports hostname check with `ssl.SSLContext.check_hostname` and *Server Name Indicator* (see `HAS_SNI`).

Distinto en la versión 3.5: El error generado por falta de compatibilidad con STARTTLS ahora es la subclase `SMTPNotSupportedError` en lugar de la base `SMTPException`.

SMTP.sendmail(*from_addr*, *to_addrs*, *msg*, *mail_options=()*, *rcpt_options=()*)

Send mail. The required arguments are an **RFC 822** [https://datatracker.ietf.org/doc/html/rfc822.html] from-address string, a list of **RFC 822** [https://datatracker.ietf.org/doc/html/rfc822.html] to-address strings (a bare string will be treated as a list with 1 address), and a message string. The caller may pass a list of ESMTP options (such as `"8bitmime"`) to be used in `MAIL FROM` commands as *mail_options*. ESMTP options (such as `DSN` commands) that should be used with all `RCPT` commands can be passed as *rcpt_options*. Each option should be passed as a string containing the full text of the option, including any potential key (for instance, `"NOTIFY=SUCCESS,FAILURE"`). (If you need to use different ESMTP options to different recipients you have to use the low-level methods such as **`mail()`**, **`rcpt()`** and **`data()`** to send the message.)

Nota

Los parámetros *from_addr* y *to_addrs* se utilizan para construir el sobre del mensaje utilizado por los agentes de transporte. `sendmail` no modifica los encabezados de los mensajes de ninguna manera.

msg puede ser una cadena que contenga caracteres en el rango ASCII o una cadena de bytes. Una cadena se codifica en bytes utilizando el códec ascii, y los caracteres `\r` y `\n` solitarios se convierten en caracteres `\ r\n`. Una cadena de bytes no se modifica.

Si no ha habido ningún comando `EHLO` o `HELO` anterior en esta sesión, este método prueba primero ESMTP `EHLO`. Si el servidor utiliza ESMTP, se le pasará el tamaño del mensaje y cada una de las opciones especificadas (si la opción está en el conjunto de funciones que anuncia el servidor). Si `EHLO` falla, se probará `HELO` y se eliminarán las opciones de ESMTP.

Este método volverá normalmente si se acepta el correo para al menos un destinatario. De lo contrario, lanzará una excepción. Es decir, si este método no genera una excepción, alguien debería recibir su correo. Si este método no genera una excepción, devuelve un diccionario, con una entrada para cada destinatario rechazado. Cada entrada contiene una tupla del código de error SMTP y el mensaje de error adjunto enviado por el servidor.

Si se incluye `SMTPUTF8`” en *mail_options * y el servidor lo admite, *from_addr* y *to_addrs* pueden contener caracteres no ASCII.

Este método puede lanzar las siguientes excepciones:

`SMTPRecipientsRefused`
All recipients were refused. Nobody got the mail.

`SMTPHeloError`
El servidor no respondió correctamente al saludo `HELO`.

`SMTPSenderRefused`
El servidor no aceptó el *from_addr*.

`SMTPDataError`
El servidor respondió con un código de error inesperado (que no sea el rechazo de un destinatario).

`SMTPNotSupportedError`
Se proporcionó `SMTPUTF8` en *mail_options* pero el servidor no lo admite.

A menos que se indique lo contrario, la conexión estará abierta incluso después de que se lance una excepción.

Distinto en la versión 3.2: msg puede ser una cadena de bytes.

Distinto en la versión 3.5: Se agregó compatibilidad con `SMTPUTF8`, así que la **`SMTPNotSupportedError`** puede ser lanzada si se especifica `SMTPUTF8` ya que el servidor no lo admite.

SMTP.send_message(*msg*, *from_addr=None*, *to_addrs=None*, *mail_options=()*, *rcpt_options=()*)

Este es un método conveniente para llamar a **`sendmail()`** con el mensaje representado por un objeto **`email.message.Message`**. Los argumentos tienen el mismo significado que para **`sendmail()`**, excepto que *msg* es un objeto `Mensaje`.

Si *from_addr* es `None` o *to_addrs* es `None`, `send_message` llena esos argumentos con direcciones extraídas de los encabezados de *msg* como se especifica en **RFC 5322** [https://datatracker.ietf.org/doc/html/rfc5322.html]: *from_addr* se establece en el campo *Sender* si está presente, y de lo contrario, en el campo *From*. *to_addrs* combina los valores (si los hay) de los campos *To*, *Cc* y *Bcc* de *msg*. Si aparece exactamente un conjunto de encabezados *Resent-** en el mensaje, los encabezados normales se ignoran y en su lugar se utilizan los encabezados *Resent- **. Si el mensaje contiene más de un conjunto de encabezados *Resent-**, se lanza un **`ValueError`**, ya que no hay forma de detectar sin ambigüedades el conjunto más reciente de encabezados *Resent-*.

`send_message` serializes *msg* using **`BytesGenerator`** with `\r\n` as the *linesep*, and calls **`sendmail()`** to transmit the resulting message. Regardless of the values of *from_addr* and *to_addrs*, `send_message` does not transmit any *Bcc* or *Resent-Bcc* headers that may appear in *msg*. If any of the addresses in *from_addr* and *to_addrs* contain non-ASCII characters and the server does not advertise `SMTPUTF8` support, an **`SMTPNotSupportedError`** is raised. Otherwise the `Message` is serialized with a clone of its **`policy`** with the **`utf8`** attribute set to `True`, and `SMTPUTF8` and `BODY=8BITMIME` are added to *mail_options*.

Added in version 3.2.

Added in version 3.5: Soporte para direcciones internacionalizadas (`SMTPUTF8`).

SMTP.quit()

Termine la sesión SMTP y cierre la conexión. Retorna el resultado del comando SMTP `QUIT`.

Los métodos de bajo nivel correspondientes a los comandos estándar SMTP/ESMTP `` ` ``HELP``, `RSET`, `NOOP`, `MAIL`, `RCPT`, y `DATA` también están soportados. Normalmente, no es necesario llamarlos directamente, por lo que no se documentan aquí. Para más detalles, consulte el código del módulo.

Additionally, an SMTP instance has the following attributes:

SMTP.helo_resp

The response to the `HELO` command, see **`helo()`**.

SMTP.ehlo_resp

The response to the `EHLO` command, see **`ehlo()`**.

SMTP.does_esmtp

A boolean value indicating whether the server supports ESMTP, see **`ehlo()`**.

SMTP.esmtp_features

A dictionary of the names of SMTP service extensions supported by the server, see **`ehlo()`**.

Ejemplo SMTP

This example prompts the user for addresses needed in the message envelope (“To” and “From” addresses), and the message to be delivered. Note that the headers to be included with the message must be included in the message as entered; this example doesn’t do any processing of the **RFC 822** [https://datatracker.ietf.org/doc/html/rfc822.html] headers. In particular, the “To” and “From” addresses must be included in the message headers explicitly:

```
import smtplib

def prompt(title):
    return input(title).strip()

from_addr = prompt("From: ")
to_addrs  = prompt("To: ").split()
print("Enter message, end with ^D (Unix) or ^Z (Windows)

# Add the From: and To: headers at the start!
lines = [f"From: {from_addr}", f"To: {', '.join(to_addrs
while True:
    try:
        line = input()
    except EOFError:
        break
    else:
        lines.append(line)

msg = "\r\n".join(lines)
print("Message length is", len(msg))

server = smtplib.SMTP("localhost")
server.set_debuglevel(1)
server.sendmail(from_addr, to_addrs, msg)
server.quit()
```

Nota

En general, querrá usar las características del paquete **`email`** para construir un mensaje de correo electrónico, que luego puede enviar a través de **`send_message()`**; ver email: Ejemplos.

`uuid` — UUID objects according to RFC 9562 [https://datatracker.ietf.org/doc/html/rfc9562.html]

Código fuente: Lib/uuid.py [https://github.com/python/cpython/tree/3.14/Lib/uuid.py]

This module provides immutable `UUID` objects (the `UUID` class) and functions for generating UUIDs corresponding to a specific UUID version as specified in **RFC 9562** [https://datatracker.ietf.org/doc/html/rfc9562.html] (which supersedes **RFC 4122** [https://datatracker.ietf.org/doc/html/rfc4122.html]), for example, `uuid1()` for UUID version 1, `uuid3()` for UUID version 3, and so on. Note that UUID version 2 is deliberately omitted as it is outside the scope of the RFC.

Si todo lo que quieres es una identificación única, probablemente deberías llamar a `uuuid1()` o `uuid4()`. Ten en cuenta que `uuid1()` puede comprometer la privacidad ya que crea un UUID que contiene la dirección de red del ordenador. `uuid4()` crea un UUID aleatorio.

Dependiendo del soporte de la plataforma subyacente, `uuid1()` puede o no retornar un UUID «seguro». Un UUID seguro es aquel que se genera mediante métodos de sincronización que aseguran que ningún proceso pueda obtener el mismo UUID. Todas las instancias de `UUID` tienen un atributo `is_safe` que transmite cualquier información sobre la seguridad del UUID, usando esta enumeración:

class uuid.SafeUUID

Added in version 3.7.

safe

El UUID fue generado por la plataforma de una manera segura para multiprocesamiento.

unsafe

El UUID no fue generado por la plataforma de una manera segura de multiprocesamiento.

unknown

La plataforma no proporciona información sobre si el UUID se generó de forma segura o no.

class uuid.UUID(*hex=None*, *bytes=None*, *bytes_le=None*, *fields=None*, *int=None*, *version=None*, ***, *is_safe=SafeUUID.unknown*)

Crea un UUID a partir de una cadena de 32 dígitos hexadecimales, una cadena de 16 bytes en orden *big-endian* como el argumento *bytes*, una cadena de 16 bytes en orden *little-endian* como el argumento *bytes_le*, una tupla de seis enteros (32-bit *time_low*, 16-bit *time_mid*, 16-bit *time_hi_version*, 8-bit *clock_seq_hi_variant*, 8-bit *clock_seq_low*, 48-bit *node*) como el argumento *fields*, o un solo entero de 128-bit como el argumento *int*. Cuando se da una cadena de dígitos hexadecimales, los corchetes, los guiones y el prefijo URN son todos opcionales. Por ejemplo, todas estas expresiones producen el mismo UUID:

```
UUID('{12345678-1234-5678-1234-567812345678}')
UUID('12345678123456781234567812345678')
UUID('urn:uuid:12345678-1234-5678-1234-567812345678
UUID(bytes=b'\x12\x34\x56\x78'*4)
UUID(bytes_le=b'\x78\x56\x34\x12\x34\x12\x78\x56' +
              b'\x12\x34\x56\x78\x12\x34\x56\x78')
UUID(fields=(0x12345678, 0x1234, 0x5678, 0x12, 0x34
UUID(int=0x12345678123456781234567812345678)
```

Exactly one of *hex*, *bytes*, *bytes_le*, *fields*, or *int* must be given. The *version* argument is optional; if given, the resulting UUID will have its variant and version number set according to **RFC 9562** [https://datatracker.ietf.org/doc/html/rfc9562.html], overriding bits in the given *hex*, *bytes*, *bytes_le*, *fields*, or *int*.

La comparación de los objetos UUID se hace mediante la comparación de sus atributos `UID.int`. La comparación con un objeto no UUID produce un `TypeError`.

`str(uuid)` retorna una cadena en la forma `12345678-1234-5678-1234-567812345678` donde los 32 dígitos hexadecimales representan el UUID.

Las instancias de `UUID` tienen estos atributos de sólo lectura:

UUID.bytes

El UUID como una cadena de 16 bytes (que contiene los seis campos enteros en orden de bytes *big-endian*).

UUID.bytes_le

El UUID como una cadena de 16 bytes (con *time_low*, *time_mid*, y *time_hi_version* en el orden de bytes *little-endian*).

UUID.fields

Una tupla de los seis campos enteros de la UUID, que también están disponibles como seis atributos individuales y dos atributos derivados:

Campo	Significado
UUID.time_low	The first 32 bits of the UUID. Only relevant to version 1.
UUID.time_mid	The next 16 bits of the UUID. Only relevant to version 1.
UUID.time_hi_version	The next 16 bits of the UUID. Only relevant to version 1.
UUID.clock_seq_hi_variant	The next 8 bits of the UUID. Only relevant to versions 1 and 6.
UUID.clock_seq_low	The next 8 bits of the UUID. Only relevant to versions 1 and 6.
UUID.node	The last 48 bits of the UUID. Only relevant to version 1.
UUID.time	The 60-bit timestamp as a count of 100-nanosecond intervals since Gregorian epoch (1582-10-15 00:00:00) for versions 1 and 6, or the 48-bit timestamp in milliseconds since Unix epoch (1970-01-01 00:00:00) for version 7.
UUID.clock_seq	The 14-bit sequence number. Only relevant to versions 1 and 6.

UUID.hex

El UUID como una cadena hexadecimal en minúsculas de 32 caracteres.

UUID.int

El UUID como un entero de 128 bits.

UUID.urn

The UUID as a URN as specified in **RFC 9562** [https://datatracker.ietf.org/doc/html/rfc9562.html].

UUID.variant

La variante UUID, que determina la disposición interna del UUID. Esta será una de las constantes `RESERVED_NCS`, `RFC_4122`, `RESERVED_MICROSOFT`, o `RESERVED_FUTURE`.

UUID.version

The UUID version number (1 through 8, meaningful only when the variant is `RFC_4122`).

Distinto en la versión 3.14: Added UUID versions 6, 7 and 8.

UUID.is_safe

Una enumeración de `SafeUUID` que indica si la plataforma generó el UUID de forma segura para el multiprocesamiento.

Added in version 3.7.

The `uuid` module defines the following functions:

uuid.getnode()

Obtiene la dirección del hardware como un entero positivo de 48 bits. La primera vez que esto se ejecute, puede lanzar un programa separado, que podría ser bastante lento. Si todos los intentos de obtener la dirección de hardware fallan, elegimos un número aleatorio de 48 bits con el bit multicast (el bit menos significativo del primer octeto) puesto a 1 como se recomienda en **RFC 4122** [https://datatracker.ietf.org/doc/html/

rfc4122.html]. «Dirección de hardware» significa la dirección MAC de una interfaz de red. En una máquina con múltiples interfaces de red, las direcciones MAC administradas universalmente (es decir, donde el segundo bit menos significativo del primer octeto es *unset*) se preferirán a las direcciones MAC administradas localmente, pero sin otras garantías de orden.

Distinto en la versión 3.7: Se prefieren las direcciones MAC administradas universalmente a las administradas localmente, ya que se garantiza que las primeras son únicas a nivel mundial, mientras que las segundas no lo son.

uuid.uuid1(*node=None*, *clock_seq=None*)

Generate a UUID from a host ID, sequence number, and the current time according to **RFC 9562, §5.1** [https://datatracker.ietf.org/doc/html/rfc9562.html#section-5.1].

When *node* is not specified, `getnode()` is used to obtain the hardware address as a 48-bit positive integer. When a sequence number *clock_seq* is not specified, a pseudo-random 14-bit positive integer is generated.

If *node* or *clock_seq* exceed their expected bit count, only their least significant bits are kept.

uuid.uuid3(*namespace*, *name*)

Generate a UUID based on the MD5 hash of a namespace identifier (which is a UUID) and a name (which is a `bytes` object or a string that will be encoded using UTF-8) according to **RFC 9562, §5.3** [https://datatracker.ietf.org/doc/html/rfc9562.html#section-5.3].

uuid.uuid4()

Generate a random UUID in a cryptographically-secure method according to **RFC 9562, §5.4** [https://datatracker.ietf.org/doc/html/rfc9562.html#section-5.4].

uuid.uuid5(*namespace*, *name*)

Generate a UUID based on the SHA-1 hash of a namespace identifier (which is a UUID) and a name (which is a `bytes` object or a string that will be encoded using UTF-8) according to **RFC 9562, §5.5** [https://datatracker.ietf.org/doc/html/rfc9562.html#section-5.5].

uuid.uuid6(*node=None*, *clock_seq=None*)

Generate a UUID from a sequence number and the current time according to **RFC 9562, §5.6** [https://datatracker.ietf.org/doc/html/rfc9562.html#section-5.6].

This is an alternative to `uuid1()` to improve database locality.

When *node* is not specified, `getnode()` is used to obtain the hardware address as a 48-bit positive integer. When a sequence number *clock_seq* is not specified, a pseudo-random 14-bit positive integer is generated.

If *node* or *clock_seq* exceed their expected bit count, only their least significant bits are kept.

Added in version 3.14.

uuid.uuid7()

Generate a time-based UUID according to **RFC 9562, §5.7** [https://datatracker.ietf.org/doc/html/rfc9562.html#section-5.7].

For portability across platforms lacking sub-millisecond precision, UUIDs produced by this function embed a 48-bit timestamp and use a 42-bit counter to guarantee monotonicity within a millisecond.

Added in version 3.14.

uuid.uuid8(*a=None*, *b=None*, *c=None*)

Generate a pseudo-random UUID according to **RFC 9562, §5.8** [https://datatracker.ietf.org/doc/html/rfc9562.html#section-5.8].

When specified, the parameters *a*, *b* and *c* are expected to be positive integers of 48, 12 and 62 bits respectively. If they exceed their expected bit count, only their least significant bits are kept; non-specified arguments are substituted for a pseudo-random integer of appropriate size.

By default, *a*, *b* and *c* are not generated by a cryptographically secure pseudo-random number generator (CSPRNG). Use `uuid4()` when a UUID needs to be used in a security-sensitive context.

Added in version 3.14.

The **`uuid`** module defines the following namespace identifiers for use with `uuid3()` or `uuid5()`.

uuid.NAMESPACE_DNS

Cuando se especifica este espacio de nombres, la cadena *name* es un nombre de dominio completamente calificado.

uuid.NAMESPACE_URL

Cuando se especifica este espacio de nombres, la cadena *name* es una URL.

uuid.NAMESPACE_OID

Cuando se especifica este espacio de nombres, la cadena *name* es un OID ISO.

uuid.NAMESPACE_X500

Cuando se especifica este espacio de nombres, la cadena *name* es un X.500 DN en DER o un formato de salida de texto.

The **`uuid`** module defines the following constants for the possible values of the `variant` attribute:

uuid.RESERVED_NCS

Reservado para la compatibilidad con NCS.

uuid.RFC_4122

Specifies the UUID layout given in **RFC 4122** [https://datatracker.ietf.org/doc/html/rfc4122.html]. This constant is kept for backward compatibility even though **RFC 4122** [https://datatracker.ietf.org/doc/html/rfc4122.html] has been superseded by **RFC 9562** [https://datatracker.ietf.org/doc/html/rfc9562.html].

uuid.RESERVED_MICROSOFT

Reservado para la compatibilidad con Microsoft.

uuid.RESERVED_FUTURE

Reservado para una futura definición.

The `uuid` module defines the special Nil and Max UUID values:

uuid.NIL

A special form of UUID that is specified to have all 128 bits set to zero according to **RFC 9562, §5.9** [https://datatracker.ietf.org/doc/html/rfc9562.html#section-5.9].

Added in version 3.14.

uuid.MAX

A special form of UUID that is specified to have all 128 bits set to one according to **RFC 9562, §5.10** [https://datatracker.ietf.org/doc/html/rfc9562.html#section-5.10].

Added in version 3.14.

Ver también

RFC 9562 [https://datatracker.ietf.org/doc/html/rfc9562.html] - A Universally Unique IDentifier (UUID) URN Namespace

Esta especificación define un espacio de nombres de recursos uniforme para los UUID, el formato interno de los UUID y los métodos de generación de los UUID.

Uso de la línea de comandos

Added in version 3.12.

The **`uuid`** module can be executed as a script from the command line.

```
python -m uuid [-h] [-u {uuid1,uuid3,uuid4,uuid5,uuid6,u
```

Se aceptan las siguientes opciones:

-h, --help

Muestra el mensaje de ayuda y libera la linea de comando.

-u <uuid>
--uuid <uuid>

Especifica el nombre de la función que se utilizará para generar el uuid. Por defecto se utiliza **`uuid4()`**.

Distinto en la versión 3.14: Allow generating UUID versions 6, 7 and 8.

-n <namespace>
--namespace <namespace>

El espacio de nombres es un `UUID` o `@ns` donde `ns` es un UUID predefinido conocido dirigido por el nombre del espacio de nombres. Como `@dns`, `@url`, `@oid` y `@x500`. Solo se requiere para las funciones **`uuid3()`** / **`uuid5()`**.

-N <name>
--name <name>

El nombre utilizado como parte de la generación del uuid. Solo se requiere para las funciones **`uuid3()`** / **`uuid5()`**.

-C <num>
--count <num>

Generate *num* fresh UUIDs.

Added in version 3.14.

Ejemplo

Here are some examples of typical usage of the **`uuid`** module:

```
>>> import uuid

>>> # make a UUID based on the host ID and current time
>>> uuid.uuid1()
UUID('a8098c1a-f86e-11da-bd1a-00112444be1e')

>>> # make a UUID using an MD5 hash of a namespace UUID
>>> uuid.uuid3(uuid.NAMESPACE_DNS, 'python.org')
UUID('6fa459ea-ee8a-3ca4-894e-db77e160355e')

>>> # make a random UUID
>>> uuid.uuid4()
UUID('16fd2706-8baf-433b-82eb-8c7fada847da')

>>> # make a UUID using a SHA-1 hash of a namespace UUID
>>> uuid.uuid5(uuid.NAMESPACE_DNS, 'python.org')
UUID('886313e1-3b8a-5372-9b90-0c9aee199e5d')

>>> # make a UUID from a string of hex digits (braces ar
>>> x = uuid.UUID('{00010203-0405-0607-0809-0a0b0c0d0e0f

>>> # convert a UUID to a string of hex digits in standa
>>> str(x)
'00010203-0405-0607-0809-0a0b0c0d0e0f'

>>> # get the raw 16 bytes of the UUID
>>> x.bytes
b'\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e

>>> # make a UUID from a 16-byte string
>>> uuid.UUID(bytes=x.bytes)
UUID('00010203-0405-0607-0809-0a0b0c0d0e0f')

>>> # get the Nil UUID
>>> uuid.NIL
UUID('00000000-0000-0000-0000-000000000000')
```

```
>>> # get the Max UUID
>>> uuid.MAX
UUID('ffffffff-ffff-ffff-ffff-ffffffffffff')

>>> # same as UUIDv1 but with fields reordered to improv
>>> uuid.uuid6()
UUID('1f0799c0-98b9-62db-92c6-a0d365b91053')

>>> # get UUIDv7 creation (local) time as a timestamp i
>>> u = uuid.uuid7()
>>> u.time
1743936859822

>>> # get UUIDv7 creation (local) time as a datetime ob
>>> import datetime as dt
>>> dt.datetime.fromtimestamp(u.time / 1000)
datetime.datetime(...)

>>> # make a UUID with custom blocks
>>> uuid.uuid8(0x12345678, 0x9abcdef0, 0x11223344)
UUID('00001234-5678-8ef0-8000-000011223344')
```

Ejemplo de línea de comandos

Here are some examples of typical usage of the **`uuid`** command-line interface:

```
# generate a random UUID - by default uuid4() is used
$ python -m uuid

# generate a UUID using uuid1()
$ python -m uuid -u uuid1

# generate a UUID using uuid5
$ python -m uuid -u uuid5 -n @url -N example.com

# generate 42 random UUIDs
$ python -m uuid -C 42
```

`socketserver` — A framework for network servers

Código fuente: Lib/socketserver.py [https://github.com/python/cpython/tree/3.14/Lib/socketserver.py]

The `socketserver` module simplifies the task of writing network servers.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

Hay cuatro clases básicas de servidores concretos:

class socketserver.TCPServer(*server_address*, *RequestHandlerClass*, *bind_and_activate=True*)

> Esto utiliza el protocolo TCP de Internet, que proporciona flujos continuos de datos entre el cliente y el servidor. Si *bind_and_activate* es verdadero, el constructor intenta invocar automáticamente `server_bind()` y `server_activate()`. Los otros parámetros se pasan a la clase base `BaseServer`.

class socketserver.UDPServer(*server_address*, *RequestHandlerClass*, *bind_and_activate=True*)

> Esto utiliza datagramas, que son paquetes discretos de información que pueden llegar fuera de servicio o perderse durante el tránsito. Los parámetros son los mismos que para `TCPServer`.

class socketserver.UnixStreamServer(*server_address*, *RequestHandlerClass*, *bind_and_activate=True*)

class socketserver.UnixDatagramServer(*server_address*, *RequestHandlerClass*, *bind_and_activate=True*)

> Estas clases que se usan con menos frecuencia son similares a las clases TCP y UDP, pero usan sockets de dominio Unix; no están disponibles en plataformas que no sean Unix. Los parámetros son los mismos que para `TCPServer`.

Estas cuatro clases procesan solicitudes *sincrónicamente*; cada solicitud debe completarse antes de que se pueda iniciar la siguiente. Esto no es adecuado si cada solicitud tarda mucho en completarse, porque requiere mucho cálculo o porque retorna muchos datos que el cliente tarda en procesar. La solución es crear un proceso o hilo independiente para manejar cada solicitud; las clases mixtas `ForkingMixIn` y `ThreadingMixIn` pueden usarse para soportar el comportamiento asincrónico.

La creación de un servidor requiere varios pasos. Primero, debes crear una clase de controlador de solicitudes subclasificando la clase `BaseRequestHandler` y anulando su método `handle()`; este método procesará las solicitudes entrantes. En segundo lugar, debe crear una instancia de una de las clases de servidor, pasándole la dirección del servidor y la clase del controlador de solicitudes. Se recomienda utilizar el servidor en una declaración `with`. Luego llame al método `handle_request()` o `serve_forever()` del objeto de servidor para procesar una o muchas solicitudes. Finalmente, llame a `server_close()` para cerrar el socket (a menos que haya usado una `with` declaración).

Al heredar de `ThreadingMixIn` para el comportamiento de la conexión con subprocesos, debe declarar explícitamente cómo desea que se comporten sus subprocesos en un cierre abrupto. La clase `ThreadingMixIn` define un atributo *daemon_threads*, que indica si el servidor debe esperar o no la terminación del hilo. Debe establecer la bandera explícitamente si desea que los subprocesos se comporten de forma autónoma; el valor predeterminado es `False`, lo que significa que Python no se cerrará hasta que todos los hilos creados por `ThreadingMixIn` hayan salido.

Las clases de servidor tienen los mismos métodos y atributos externos, independientemente del protocolo de red que utilicen.

Notas de creación del servidor

Hay cinco clases en un diagrama de herencia, cuatro de las cuales representan servidores síncronos de cuatro tipos:

```
+------------+
| BaseServer |
+------------+
      |
      v
+-----------+        +------------------+
| TCPServer |------->| UnixStreamServer |
+-----------+        +------------------+
      |
      v
+-----------+        +--------------------+
| UDPServer |------->| UnixDatagramServer |
+-----------+        +--------------------+
```

Tenga en cuenta que `UnixDatagramServer` deriva de `UDPServer`, no de `UnixStreamServer` — la única diferencia entre un servidor IP y uno Unix es la familia de direcciones.

class socketserver.ForkingMixIn
class socketserver.ThreadingMixIn

> Se pueden crear versiones de Forking y threading de cada tipo de servidor utilizando estas clases mixtas. Por ejemplo, `ThreadingUDPServer` se crea de la siguiente manera:
>
> ```
> class ThreadingUDPServer(ThreadingMixIn, UDPServer)
>     pass
> ```
>
> La clase de mezcla es lo primero, ya que anula un método definido en `UDPServer`. La configuración de los distintos atributos también cambia el comportamiento del mecanismo del servidor subyacente.

`ForkingMixIn` y las clases de Forking mencionadas a continuación solo están disponibles en plataformas POSIX que admiten `os.fork()`.

block_on_close

> `ForkingMixIn.server_close` waits until all child processes complete, except if `block_on_close` attribute is `False`.
>
> `ThreadingMixIn.server_close` waits until all non-daemon threads complete, except if `block_on_close` attribute is `False`.

max_children

> Specify how many child processes will exist to handle requests at a time for **`ForkingMixIn`**. If the limit is reached, new requests will wait until one child process has finished.

daemon_threads

> For **`ThreadingMixIn`** use daemonic threads by setting `ThreadingMixIn.daemon_threads` to `True` to not wait until threads complete.

Distinto en la versión 3.7: `ForkingMixIn.server_close` and **`ThreadingMixIn.server_close`** now waits until all child processes and non-daemonic threads complete. Add a new `ForkingMixIn.block_on_close` class attribute to opt-in for the pre-3.7 behaviour.

class socketserver.ForkingTCPServer
class socketserver.ForkingUDPServer
class socketserver.ThreadingTCPServer
class socketserver.ThreadingUDPServer
class socketserver.ForkingUnixStreamServer
class socketserver.ForkingUnixDatagramServer
class socketserver.ThreadingUnixStreamServer
class socketserver.ThreadingUnixDatagramServer

Estas clases están predefinidas utilizando las clases mixtas.

Added in version 3.12: Las clases `ForkingUnixStreamServer` y `ForkingUnixDatagramServer` fueron agregadas.

Para implementar un servicio, debes derivar una clase de **`BaseRequestHandler`** y redefinir su método **`handle()`**. Luego, puede ejecutar varias versiones del servicio combinando una de las clases de servidor con su clase de controlador de solicitudes. La clase del controlador de solicitudes debe ser diferente para los servicios de datagramas o flujos. Esto se puede ocultar usando las subclases del controlador **`StreamRequestHandler`** o **`DatagramRequestHandler`**.

Por supuesto, ¡todavía tienes que usar la cabeza! Por ejemplo, no tiene sentido usar un servidor de bifurcación si el servicio contiene un estado en la memoria que puede ser modificado por diferentes solicitudes, ya que las modificaciones en el proceso hijo nunca alcanzarían el estado inicial que se mantiene en el proceso padre y se pasa a cada hijo. En este caso, puede usar un servidor de subprocesos, pero probablemente tendrá que usar bloqueos para proteger la integridad de los datos compartidos.

Por otro lado, si está creando un servidor HTTP donde todos los datos se almacenan externamente (por ejemplo, en el sistema de archivos), una clase síncrona esencialmente hará que el servicio sea «sordo» mientras se maneja una solicitud, que puede ser durante mucho tiempo si un cliente tarda en recibir todos los datos que ha solicitado. Aquí es apropiado un servidor de enhebrado o bifurcación.

En algunos casos, puede ser apropiado procesar parte de una solicitud de forma síncrona, pero para finalizar el procesamiento en un hijo bifurcado según los datos de la solicitud. Esto se puede implementar usando un servidor sincrónico y haciendo un fork explicito en la clase del controlador de solicitudes el método **`handle()`**.

Otro enfoque para gestionar múltiples solicitudes simultáneas en un entorno que no admite subprocesos ni **`fork()`** (o donde estos son demasiado costosos o inadecuados para el servicio) es mantener una tabla explícita de solicitudes parcialmente terminadas y utilizar **`selectors`** para decidir en qué solicitud trabajar a continuación (o si manejar una nueva solicitud entrante). Esto es particularmente importante para los servicios de transmisión en los que cada cliente puede potencialmente estar conectado durante mucho tiempo (si no se pueden utilizar subprocesos o subprocesos).

Objetos de servidor

class socketserver.BaseServer(*server_address, RequestHandlerClass*)

Esta es la superclase de todos los objetos de servidor en el módulo. Define la interfaz, que se indica a continuación, pero no implementa la mayoría de los métodos, que se realiza en subclases. Los dos parámetros se almacenan en los atributos respectivos `server_address` y `RequestHandlerClass`.

fileno()

Retorna un descriptor de archivo entero para el socket en el que escucha el servidor. Esta función se pasa comúnmente a `selectors`, para permitir monitorear múltiples servidores en el mismo proceso.

handle_request()

Procese una sola solicitud. Esta función llama a los siguientes métodos en orden: `get_request()`, `verify_request()`, y `process_request()`. Si el método proporcionado por el usuario `handle()` de la clase del controlador lanza una excepción, se llamará al método `handle_error()` del servidor. Si no se recibe ninguna solicitud en `timeout` segundos, se llamará a `handle_timeout()` y `handle_request()` regresará.

serve_forever(*poll_interval=0.5*)

Manejar solicitudes hasta una solicitud explícita `shutdown()`. Sondeo de apagado cada *poll_interval* segundos. Ignora el atributo `timeout`. También llama `service_actions()`, que puede ser utilizado por una subclase o mixin para proporcionar acciones específicas para un servicio dado. Por ejemplo, la clase

`ForkingMixIn` usa **`service_actions()`** para limpiar procesos secundarios zombies.

Distinto en la versión 3.3: Se agregó la llamada `service_actions` al método `serve_forever`.

service_actions()

Esto se llama en el ciclo `serve_forever()`. Este método puede ser reemplazado por subclases o clases mixtas para realizar acciones específicas para un servicio dado, como acciones de limpieza.

Added in version 3.3.

shutdown()

Dile al bucle `serve_forever()` que se detenga y espere hasta que lo haga. `shutdown()` debe llamarse mientras **`serve_forever()`** se está ejecutando en un hilo diferente, de lo contrario, se bloqueará.

server_close()

Limpiar el servidor. Puede ser sobrescrito.

address_family

The family of protocols to which the server's socket belongs. Common examples are `socket.AF_INET`, `socket.AF_INET6`, and `socket.AF_UNIX`. Subclass the TCP or UDP server classes in this module with class attribute `address_family = AF_INET6` set if you want IPv6 server classes.

RequestHandlerClass

La clase de controlador de solicitudes proporcionada por el usuario; se crea una instancia de esta clase para cada solicitud.

server_address

La dirección en la que escucha el servidor. El formato de las direcciones varía según la familia de protocolos; consulte la documentación del módulo `socket` para obtener más detalles. Para los protocolos de Internet, esta es una tupla que contiene una cadena que da la dirección y un número de puerto entero: `('127.0.0.1', 80)`, por ejemplo.

socket

El objeto de socket en el que el servidor escuchará las solicitudes entrantes.

Las clases de servidor admiten las siguientes variables de clase:

allow_reuse_address

Si el servidor permitirá la reutilización de una dirección. Este valor predeterminado es `False`, y se puede establecer en subclases para cambiar la política.

request_queue_size

El tamaño de la cola de solicitudes. Si toma mucho tiempo procesar una sola solicitud, cualquier solicitud que llegue mientras el servidor está ocupado se coloca en una cola, hasta `request_queue_size`. Una vez que la cola está llena, más solicitudes de clientes obtendrán un error de «Conexión denegada». El valor predeterminado suele ser 5, pero las subclases pueden anularlo.

socket_type

El tipo de socket utilizado por el servidor; `socket.SOCK_STREAM` y `socket.SOCK_DGRAM` son dos valores comunes.

timeout

Duración del tiempo de espera, medida en segundos, o `None` si no se desea un tiempo de espera. Si `handle_request()` no recibe solicitudes entrantes dentro del período de tiempo de espera, se llama al método `handle_timeout()`.

Hay varios métodos de servidor que pueden ser anulados por subclases de clases de servidor base como `TCPServer`; estos métodos no son útiles para los usuarios externos del objeto de servidor.

finish_request(*request, client_address*)

En realidad, procesa la solicitud creando instancias `RequestHandlerClass` y llamando a su método `handle()`.

get_request()

Debe aceptar una solicitud del socket y retornar una tupla de 2 que contenga el objeto de socket *nuevo* que se utilizará para comunicarse con el cliente y la dirección del cliente.

handle_error(*request, client_address*)

Esta función se llama si el método `handle()` de una instancia `RequestHandlerClass` lanza una excepción. La acción predeterminada es imprimir el rastreo al error estándar y continuar manejando más solicitudes.

Distinto en la versión 3.6: Ahora solo se solicitan excepciones derivadas de la clase `Exception`.

handle_timeout()

Esta función se llama cuando el atributo `timeout` se ha establecido en un valor distinto de `None` y el tiempo de espera ha pasado sin que se reciban solicitudes. La acción predeterminada para los servidores de forking es recopilar el estado de cualquier proceso hijo que haya salido, mientras que en los servidores de threading este método no hace nada.

process_request(*request, client_address*)

Llama a `finish_request()` para crear una instancia de `RequestHandlerClass`. Si lo desea, esta función puede crear un nuevo proceso o hilo para manejar la solicitud; las clases `ForkingMixIn` y `ThreadingMixIn` hacen esto.

server_activate()

Lo llama el constructor del servidor para activar el servidor. El comportamiento predeterminado para un servidor TCP simplemente invoca `listen()` en el socket del servidor. Puede anularse.

server_bind()

Lo llama el constructor del servidor para vincular el socket a la dirección deseada. Puede anularse.

verify_request(*request, client_address*)

Debe retornar un valor booleano; si el valor es `True`, la solicitud se procesará, y si es `False`, la solicitud será denegada. Esta función se puede anular para implementar controles de acceso para un servidor. La implementación predeterminada siempre retorna `True`.

Distinto en la versión 3.6: Se agregó soporte para el protocolo context manager. Salir del administrador de contexto es equivalente a llamar a `server_close()`.

Solicitar objetos de controlador

class socketserver.BaseRequestHandler

Esta es la superclase de todos los objetos de manejo de solicitudes. Define la interfaz, que se muestra a continuación. Una subclase de controlador de solicitudes concreta debe definir un nuevo método `handle()` y puede anular cualquiera de los otros métodos. Se crea una nueva instancia de la subclase para cada solicitud.

setup()

Se llama antes del método `handle()` para realizar las acciones de inicialización necesarias. La implementación predeterminada no hace nada.

handle()

This function must do all the work required to service a request. The default implementation does nothing. Several instance attributes are available to it; the request is available as `request`; the client address as `client_address`; and the server instance as `server`, in case it needs access to per-server information.

The type of `request` is different for datagram or stream services. For stream services, `request` is a socket object; for datagram services, `request` is a pair of string and socket.

finish()

Se llama después del método `handle()` para realizar las acciones de limpieza necesarias. La implementación predeterminada no hace nada. Si `setup()` lanza una excepción, no se llamará a esta función.

request

The *new* `socket.socket` object to be used to communicate with the client.

client_address

Client address returned by `BaseServer.get_request()`.

server

`BaseServer` object used for handling the request.

class socketserver.StreamRequestHandler *class* socketserver.DatagramRequestHandler

These `BaseRequestHandler` subclasses override the `setup()` and `finish()` methods, and provide `rfile` and `wfile` attributes.

rfile

A file object from which receives the request is read. Support the `io.BufferedIOBase` readable interface.

wfile

A file object to which the reply is written. Support the `io.BufferedIOBase` writable interface

Distinto en la versión 3.6: `wfile` also supports the `io.BufferedIOBase` writable interface.

Ejemplos

`socketserver.TCPServer` Ejemplo

Este es el lado del servidor:

```
import socketserver

class MyTCPHandler(socketserver.BaseRequestHandler):
    """
    The request handler class for our server.

    It is instantiated once per connection to the serve
    override the handle() method to implement communicat
    client.
    """

    def handle(self):
        # self.request is the TCP socket connected to th
        pieces = [b'']
        total = 0
        while b'\n' not in pieces[-1] and total < 10_00
```

```
            pieces.append(self.request.recv(2000))
            total += len(pieces[-1])
        self.data = b''.join(pieces)
        print(f"Received from {self.client_address[0]}:"
        print(self.data.decode("utf-8"))
        # just send back the same data, but upper-cased
        self.request.sendall(self.data.upper())
        # after we return, the socket will be closed.

if __name__ == "__main__":
    HOST, PORT = "localhost", 9999

    # Create the server, binding to localhost on port 99
    with socketserver.TCPServer((HOST, PORT), MyTCPHandl
        # Activate the server; this will keep running un
        # interrupt the program with Ctrl-C
        server.serve_forever()
```

Una clase de controlador de solicitudes alternativa que hace uso de secuencias (objetos similares a archivos que simplifican la comunicación al proporcionar la interfaz de archivo estándar):

```
class MyTCPHandler(socketserver.StreamRequestHandler):

    def handle(self):
        # self.rfile is a file-like object created by th
        # We can now use e.g. readline() instead of raw
        # We limit ourselves to 10000 bytes to avoid abu
        self.data = self.rfile.readline(10000).rstrip()
        print(f"{self.client_address[0]} wrote:")
        print(self.data.decode("utf-8"))
        # Likewise, self.wfile is a file-like object use
        # to the client
        self.wfile.write(self.data.upper())
```

The difference is that the `readline()` call in the second handler will call `recv()` multiple times until it encounters a newline character, while the first handler had to use a `recv()` loop to accumulate data until a newline itself. If it had just used a single `recv()` without the loop it would just have returned what has been received so far from the client. TCP is stream based: data arrives in the order it was sent, but there is no correlation between client `send()` or `sendall()` calls and the number of `recv()` calls on the server required to receive it.

Este es el lado del cliente:

```
import socket
import sys

HOST, PORT = "localhost", 9999
data = " ".join(sys.argv[1:])

# Create a socket (SOCK_STREAM means a TCP socket)
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) a
    # Connect to server and send data
    sock.connect((HOST, PORT))
    sock.sendall(bytes(data, "utf-8"))
    sock.sendall(b"\n")

    # Receive data from the server and shut down
    received = str(sock.recv(1024), "utf-8")

print("Sent:    ", data)
print("Received:", received)
```

La salida del ejemplo debería verse así:

Servidor:

```
$ python TCPServer.py
127.0.0.1 wrote:
b'hello world with TCP'
127.0.0.1 wrote:
b'python is nice'
```

Cliente:

```
$ python TCPClient.py hello world with TCP
```

```
Sent:     hello world with TCP
Received: HELLO WORLD WITH TCP
$ python TCPClient.py python is nice
Sent:     python is nice
Received: PYTHON IS NICE
```

`socketserver.UDPServer` Ejemplo

Este es el lado del servidor:

```
import socketserver

class MyUDPHandler(socketserver.BaseRequestHandler):
    """
    This class works similar to the TCP handler class, e
    self.request consists of a pair of data and client s
    there is no connection the client address must be gi
    when sending data back via sendto().
    """

    def handle(self):
        data = self.request[0].strip()
        socket = self.request[1]
        print(f"{self.client_address[0]} wrote:")
        print(data)
        socket.sendto(data.upper(), self.client_address)

if __name__ == "__main__":
    HOST, PORT = "localhost", 9999
    with socketserver.UDPServer((HOST, PORT), MyUDPHandl
        server.serve_forever()
```

Este es el lado del cliente:

```
import socket
import sys

HOST, PORT = "localhost", 9999
data = " ".join(sys.argv[1:])
```

```
# SOCK_DGRAM is the socket type to use for UDP sockets
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# As you can see, there is no connect() call; UDP has no
# Instead, data is directly sent to the recipient via se
sock.sendto(bytes(data + "\n", "utf-8"), (HOST, PORT))
received = str(sock.recv(1024), "utf-8")

print("Sent:     ", data)
print("Received:", received)
```

La salida del ejemplo debería verse exactamente como en el ejemplo del servidor TCP.

Mixins asincrónicos

Para construir controladores asincrónicos, use las clases **`ThreadingMixIn`** y **`ForkingMixIn`**.

Un ejemplo para la clase **`ThreadingMixIn`** class

```
import socket
import threading
import socketserver

class ThreadedTCPRequestHandler(socketserver.BaseRequest

    def handle(self):
        data = str(self.request.recv(1024), 'ascii')
        cur_thread = threading.current_thread()
        response = bytes("{}: {}".format(cur_thread.name
        self.request.sendall(response)

class ThreadedTCPServer(socketserver.ThreadingMixIn, soc
    pass

def client(ip, port, message):
    with socket.socket(socket.AF_INET, socket.SOCK_STREA
        sock.connect((ip, port))
        sock.sendall(bytes(message, 'ascii'))
        response = str(sock.recv(1024), 'ascii')
        print("Received: {}".format(response))

if __name__ == "__main__":
    # Port 0 means to select an arbitrary unused port
    HOST, PORT = "localhost", 0

    server = ThreadedTCPServer((HOST, PORT), ThreadedTCF
    with server:
        ip, port = server.server_address

        # Start a thread with the server -- that thread
        # more thread for each request
        server_thread = threading.Thread(target=server.s
        # Exit the server thread when the main thread te
        server_thread.daemon = True
        server_thread.start()
        print("Server loop running in thread:", server_t

        client(ip, port, "Hello World 1")
        client(ip, port, "Hello World 2")
        client(ip, port, "Hello World 3")

        server.shutdown()
```

La salida del ejemplo debería verse así:

```
$ python ThreadedTCPServer.py
Server loop running in thread: Thread-1
Received: Thread-2: Hello World 1
Received: Thread-3: Hello World 2
Received: Thread-4: Hello World 3
```

La clase `ForkingMixIn` se usa de la misma manera, excepto que el servidor generará un nuevo proceso para cada solicitud. Disponible solo en plataformas POSIX que admitan `fork()`.

`http.server` — HTTP servers

Código fuente: Lib/http/server.py [https://github.com/python/cpython/tree/3.14/Lib/http/server.py]

Este módulo define clases para implementar servidores HTTP.

Advertencia

`http.server` is not recommended for production. It only implements basic security checks.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

Una clase, `HTTPServer`, es una subclase `socketserver.TCPServer`. Crea y escucha en el socket HTTP, enviando las peticiones a un handler. El código para crear y ejecutar el servidor se ve así:

```
def run(server_class=HTTPServer, handler_class=BaseHTTP
    server_address = ('', 8000)
    httpd = server_class(server_address, handler_class)
    httpd.serve_forever()
```

class http.server.HTTPServer(*server_address*, *RequestHandlerClass*)

Esta clase se basa en la clase `TCPServer` almacenando la dirección del servidor como variables de instancia llamadas `nombre_del_servidor` y `puerto_del_servidor`. El servidor es accesible por el handler, típicamente a través de la variable de instancia `servidor` del handler.

class http.server.ThreadingHTTPServer(*server_address*, *RequestHandlerClass*)

Esta clase es idéntica a HTTPServer, pero utiliza subprocesos para controlar las solicitudes mediante el uso de `ThreadingMixIn`. Esto es útil para controlar los sockets de pre-apertura de los navegadores web, en los que `HTTPServer` esperaría indefinidamente.

Added in version 3.7.

class http.server.HTTPSServer(*server_address*, *RequestHandlerClass*, *bind_and_activate=True*, ***, *certfile*, *keyfile=None*, *password=None*, *alpn_protocols=None*)

Subclass of `HTTPServer` with a wrapped socket using the `ssl` module. If the `ssl` module is not available, instantiating a `HTTPSServer` object fails with a `RuntimeError`.

The *certfile* argument is the path to the SSL certificate chain file, and the *keyfile* is the path to file containing the private key.

A *password* can be specified for files protected and wrapped with PKCS#8, but beware that this could possibly expose hardcoded passwords in clear.

Ver también

See `ssl.SSLContext.load_cert_chain()` for additional information on the accepted values for *certfile*, *keyfile* and *password*.

When specified, the *alpn_protocols* argument must be a sequence of strings specifying the «Application-Layer Protocol Negotiation» (ALPN) protocols supported by the server. ALPN allows the server and the client to negotiate the application protocol during the TLS handshake.

By default, it is set to `["http/1.1"]`, meaning the server supports HTTP/1.1.

Added in version 3.14.

class http.server.ThreadingHTTPSServer(*server_address*, *RequestHandlerClass*, *bind_and_activate=True*, ***, *certfile*, *keyfile=None*, *password=None*, *alpn_protocols=None*)

This class is identical to `HTTPSServer` but uses threads to handle requests by inheriting from `ThreadingMixIn`. This is analogous to `ThreadingHTTPServer` only using **`HTTPSServer`**.

Added in version 3.14.

The `HTTPServer`, `ThreadingHTTPServer`, `HTTPSServer` and `ThreadingHTTPSServer` must be given a *RequestHandlerClass* on instantiation, of which this module provides three different variants:

class http.server.BaseHTTPRequestHandler(*request*, *client_address*, *server*)

This class is used to handle the HTTP requests that arrive at the server. By itself, it cannot respond to any actual HTTP requests; it must be subclassed to handle each request method (for example, `'GET'` or `'POST'`). **`BaseHTTPRequestHandler`** provides a number of class and instance variables, and methods for use by subclasses.

The handler will parse the request and the headers, then call a method specific to the request type. The method name is constructed from the request. For example, for the request method `SPAM`, the **`do_SPAM()`** method will be called with no arguments. All of the relevant information is stored in instance variables of the handler. Subclasses should not need to override or extend the **`__init__()`** method.

`BaseHTTPRequestHandler` tiene las siguientes variables de instancia:

client_address

Contiene una tupla con el formato `(host, port)` que

hace referencia a la dirección del cliente.

server

Contiene la instancia del servidor.

close_connection

Booleano que se debe establecer antes de **`handle_one_request()`** retorna, que indica si se puede esperar otra solicitud o si la conexión debe cerrarse.

requestline

Contiene la representación de cadena de la línea de solicitud HTTP. Se elimina el CRLF de terminación. Este atributo debe establecerse mediante **`handle_one_request()`**. Si no se ha procesado ninguna línea de solicitud válida, debe establecerse en la cadena vacía.

command

Contiene el comando (tipo de petición). Por ejemplo, `'GET'`.

path

Contiene la ruta de la solicitud. Si el componente de consulta de la URL está presente, `path` incluye la consulta. Usando la terminología de: rfc: *3986*, `path` aquí incluye `hier-part` y `query`.

request_version

Contiene la versión de la cadena de caracteres para la petición. Por ejemplo, `HTTP/1.0'`.

headers

Holds an instance of the class specified by the **`MessageClass`** class variable. This instance parses and manages the headers in the HTTP request. The `parse_headers()` function from `http.client` is used to parse the headers and it requires that the HTTP request provide a valid RFC 5322 [https://datatracker.ietf.org/doc/html/rfc5322.html] style header.

rfile

Un flujo de entrada `io.BufferedIOBase`, listo para leer desde el inicio de los datos de entrada opcionales.

wfile

Contiene el flujo de salida para escribir una respuesta al cliente. Se debe utilizar la adherencia apropiada al protocolo HTTP cuando se escribe en este flujo para lograr una interoperación exitosa con los clientes HTTP.

Distinto en la versión 3.6: Este es un flujo de `io.BufferedIOBase`.

`BaseHTTPRequestHandler` tiene los siguientes atributos:

server_version

Especifica la versión del software del servidor. Es posible que desee anular esto. El formato es de múltiples cadenas separadas por espacio en blanco, donde cada cadena es de la forma nombre[/versión]. Por ejemplo, `BaseHTTP/0.2'`.

sys_version

Contiene la versión del sistema Python, en una forma utilizable por el método `version_string` y la variable de clase `server_version`. Por ejemplo, `Python/1.4'`.

error_message_format

Especifica una cadena de formato que debe ser usada por el método `send_error()` para construir una respuesta de error al cliente. La cadena se rellena por defecto con variables de `responses` basadas en el código de estado que pasó a **`send_error()`**.

error_content_type

Especifica el encabezado *HTTP Content-Type* de las respuestas de error enviadas al cliente. El valor predeterminado es `'text/html'`.

protocol_version

Especifica la versión de HTTP con la cual el servidor es conforme. Es enviada como respuesta para informarle al cliente sobre las capacidades de comunicación del servidor para siguientes peticiones. Si se establece en `'HTTP/1.1'`, el servidor permitirá conexiones persistentes HTTP; sin embargo, el servidor *debe* incluir un encabezado exacto `Content-Length` (usando **`send_header()`**) en todas sus respuestas a los clientes. Para la compatibilidad con versiones anteriores, el valor predeterminado es `'HTTP/1.0'`.

MessageClass

Especifica una **`email.message.Message`**-como clase para analizar los encabezados HTTP. Típicamente, esto no es anulado, y por defecto es **`http.client.HTTPMessage`**.

responses

Este atributo contiene una asignación de enteros de código de error a tuplas de dos elementos que contienen un mensaje corto y largo. Por ejemplo, `{code (shortmessage, longmessage)}`. El *shortmessage* se utiliza normalmente como la clave *message* en una respuesta de error, y *longmessage* como la clave *explain*. Es utilizado por **`send_response_only()`** y **`send_error()`** métodos.

Una instancia **`BaseHTTPRequestHandler`** tiene los siguientes métodos:

handle()

Calls `handle_one_request()` once (or, if persistent connections are enabled, multiple times) to handle incoming HTTP requests. You should never need to override it; instead, implement appropriate **`do_*()`** methods.

handle_one_request()

This method will parse and dispatch the request to the appropriate **`do_*()`** method. You should never need to override it.

handle_expect_100()

When an HTTP/1.1 conformant server receives an `Expect: 100-continue` request header it responds back with a `100 Continue` followed by `200 OK` headers. This method can be overridden to raise an error if the server does not want the client to continue. For example, the server can choose to send `417 Expectation Failed` as a response header and `return False`.

Added in version 3.2.

send_error(*code*, *message*=*None*, *explain*=*None*)

Sends and logs a complete error reply to the client. The numeric *code* specifies the HTTP error code, with *message* as an optional, short, human readable description of the error. The *explain* argument can be used to provide more detailed information about the error; it will be formatted using the `error_message_format` attribute and emitted, after a complete set of headers, as the response body. The `responses` attribute holds the default values for *message* and *explain* that will be used if no value is provided; for unknown codes the default value for both is the string `???`. The body will be empty if the method is HEAD or the response code is one of the

following: `1xx`, `204 No Content`, `205 Reset Content`, `304 Not Modified`.

Distinto en la versión 3.4: La respuesta de error incluye un encabezado de longitud de contenido. Añadido el argumento *explain*.

send_response(*code*, *message=None*)

Agrega un encabezado de respuesta al búfer de encabezados y registra la solicitud aceptada. La línea de respuesta HTTP se escribe en el búfer interno, seguido de los encabezados *Server* y *Date*. Los valores de estos dos encabezados se recogen de los métodos `version_string()` y `date_time_string()`, respectivamente. Si el servidor no tiene la intención de enviar ningún otro encabezado utilizando el método `send_header()`, entonces `send_response()` debe ir seguido de una llamada `end_headers()`.

Distinto en la versión 3.3: Los encabezados se almacenan en un búfer interno y `end_headers()` debe llamarse explícitamente.

send_header(*keyword*, *value*)

Agrega el encabezado HTTP a un búfer interno que se escribirá en la secuencia de salida cuando se invoca `end_headers()` o `flush_headers()`. *keyword* debe especificar la palabra clave *header*, con *value* especificando su valor. Tenga en cuenta que, después de que se realizan las llamadas *send_header*, **`end_headers()`** DEBE llamarse para completar la operación.

This method does not reject input containing CRLF sequences.

Distinto en la versión 3.2: Los encabezados se almacenan en un búfer interno.

send_response_only(*code*, *message=None*)

Envía el encabezado de respuesta solamente, usado para los propósitos cuando la respuesta `100 Continue` es enviada por el servidor al cliente. Los encabezados no se almacenan en el buffer y envían directamente el flujo de salida. Si no se especifica el *message*, se envía el mensaje HTTP correspondiente al *code* de respuesta.

This method does not reject *message* containing CRLF sequences.

Added in version 3.2.

end_headers()

Adds a blank line (indicating the end of the HTTP headers in the response) to the headers buffer and calls **`flush_headers()`**.

Distinto en la versión 3.2: Los encabezados del buffer se escriben en el flujo de salida.

flush_headers()

Finalmente envía los encabezados al flujo de salida y limpia el buffer interno de los cabezales.

Added in version 3.3.

log_request(*code='-'*, *size='-'*)

Registra una solicitud aceptada (exitosa). El *code* debe especificar el código numérico HTTP asociado a la respuesta. Si un tamaño de la respuesta está disponible, entonces debe ser pasado como el parámetro *size*.

log_error(...)

Registra un error cuando una solicitud no puede ser cumplida. Por defecto, pasa el mensaje a **`log_message()`**, por lo que toma los mismos argumentos (*format* y valores adicionales).

log_message(*format*, ...)

Registra un mensaje arbitrario en `sys.stderr`. Normalmente se anula para crear mecanismos personalizados de registro de errores. El argumento *format* es una cadena de formato estándar de estilo de impresión, donde los argumentos adicionales a **`log_message()`** se aplican como entradas al formato. La dirección ip del cliente y la fecha y hora actual son prefijadas a cada mensaje registrado.

version_string()

Retorna la cadena de versiones del software del servidor. Esta es una combinación de los atributos **`server_version`** y **`sys_version`**.

date_time_string(*timestamp=None*)

Retorna la fecha y la hora dadas por *timestamp* (que debe ser `None` o en el formato retornado por **`time.time`()**), formateado para un encabezado de mensaje. Si se omite *timestamp*, utiliza la fecha y la hora actuales.

El resultado se muestra como `Sun, 06 Nov 1994 08:49:37 GMT'`.

log_date_time_string()

Retorna la fecha y la hora actuales, formateadas para el registro.

address_string()

Retorna la dirección del cliente.

Distinto en la versión 3.3: Anteriormente, se realizó una búsqueda de nombres. Para evitar retrasos en la resolución del nombre, ahora siempre retorna la dirección IP.

class http.server.SimpleHTTPRequestHandler(*request*, *client_address*, *server*, *directory=None*)

Esta clase sirve archivos del directorio *directory* e inferior, o del directorio actual si no se proporciona *directory*, mapeando directamente la estructura del directorio a las solicitudes HTTP.

Distinto en la versión 3.7: Added the *directory* parameter.

Distinto en la versión 3.9: El parámetro *directory* acepta un path-like object.

La carga de trabajo, como el análisis de la solicitud, lo hace la clase base **`BaseHTTPRequestHandler`**. Esta clase implementa las funciones **`do_GET()`** y **`do_HEAD()`**.

Los siguientes se definen como atributos de clase de **`SimpleHTTPRequestHandler`**:

server_version

Esto sería `"SimpleHTTP/" + __version__`, donde `__version__` se define a nivel de módulo.

extensions_map

Un diccionario que asigna sufijos a tipos MIME contiene sobreescrituras personalizadas para las asignaciones predeterminadas del sistema. El mapeo se usa sin distinción entre mayúsculas y minúsculas, por lo que solo debe contener claves en minúsculas.

Distinto en la versión 3.9: Este diccionario ya no contiene las asignaciones predeterminadas del sistema, sino que solo contiene anulaciones.

Una instancia **`SimpleHTTPRequestHandler`** tiene los siguientes métodos:

do_HEAD()

Este método sirve para el tipo de petición: `'HEAD'` envía los encabezados que enviaría para la petición equivalente `GET`. Ver el método **`do_GET()`** para una explicación más completa de los posibles encabezados.

do_GET()

La solicitud se asigna a un archivo local interpretando la solicitud como una ruta relativa al directorio de trabajo actual.

Si la solicitud fue mapeada a un directorio, el directorio se comprueba para un archivo llamado `index.html` or `index.htm` (en ese orden). Si se encuentra, se retorna el contenido del archivo; de lo contrario, se genera un listado del directorio llamando al método **`list_directory()`**. Este método utiliza **`os.listdir()`** para escanear el directorio, y retorna una respuesta de error `404` si falla el **`listdir()`**.

If the request was mapped to a file, it is opened. Any **`OSError`** exception in opening the requested file is mapped to a `404`, `'File not found'` error. If there was an `'If-Modified-Since'` header in the request, and the file was not modified after this time, a `304`, `'Not Modified'` response is sent. Otherwise, the content type is guessed by calling the **`guess_type()`** method, which in turn uses the *extensions_map* variable, and the file contents are returned.

Un encabezado de `'Content-type:'` con el tipo de contenido adivinado, seguido de un encabezado de `'Content-Length:'` con el tamaño del archivo y un encabezado de `'Last-Modified:'` con el tiempo de modificación del archivo.

Then follows a blank line signifying the end of the headers, and then the contents of the file are output.

Por ejemplo, ver la implementación de la función `test` en Lib/http/server.py [https://github.com/python/cpython/tree/3.14/Lib/http/server.py].

Distinto en la versión 3.7: Soporta la cabecera `'If-Modified-Since'`.

La clase **`SimpleHTTPRequestHandler`** puede ser usada de la siguiente manera para crear un servidor web muy básico que sirva archivos relativos al directorio actual:

```
import http.server
import socketserver

PORT = 8000

Handler = http.server.SimpleHTTPRequestHandler

with socketserver.TCPServer(("", PORT), Handler) as http
    print("serving at port", PORT)
    httpd.serve_forever()
```

`SimpleHTTPRequestHandler` can also be subclassed to enhance behavior, such as using different index file names by overriding the class attribute **`index_pages`**.

class http.server.CGIHTTPRequestHandler(*request*, *client_address*, *server*)

Esta clase se utiliza para servir tanto a los archivos como a la salida de los scripts CGI del directorio actual y del siguiente. Note que el mapeo de la estructura jerárquica de HTTP a la estructura del directorio local es exactamente como en **`SimpleHTTPRequestHandler`**.

Nota

Los scripts CGI ejecutados por la clase **`CGIHTTPRequestHandler`** no pueden ejecutar redirecciones (código HTTP 302), porque el código 200 (la salida del script sigue) se envía antes de la ejecución del script CGI. Esto adelanta el código de estado.

La clase, sin embargo, ejecutará el script CGI, en lugar de servirlo como un archivo, si adivina que es un script CGI. Sólo se usan CGI basados en directorios — la otra configuración común del servidor es tratar las extensiones especiales como denotando los scripts CGI.

Las funciones **`do_GET()`** y **`do_HEAD()`** se modifican para ejecutar scripts CGI y servir la salida, en lugar de servir archivos, si la petición lleva a algún lugar por debajo de la ruta `cgi_directories`.

La **`CGIHTTPRequestHandler`** define el siguiente miembro de datos:

cgi_directories

> Esto por defecto es `['/cgi-bin', '/htbin']` y describe los directorios a tratar como si contuvieran scripts CGI.

La **`CGIHTTPRequestHandler`** define el siguiente método:

do_POST()

> Este método sirve para el tipo de petición `'POST'`, sólo permitido para scripts CGI. El error 501, «Can only POST to CGI scripts», se produce cuando se intenta enviar a una url no CGI.

Tenga en cuenta que los scripts CGI se ejecutarán con UID de usuario *nobody*, por razones de seguridad. Los problemas con el script CGI serán traducidos al error 403.

Deprecated since version 3.13, will be removed in version 3.15: **`CGIHTTPRequestHandler`** is being removed in 3.15. CGI has not been considered a good way to do things for well over a decade. This code has been unmaintained for a while now and sees very little practical use. Retaining it could lead to further security considerations.

Command-line interface

`http.server` can also be invoked directly using the **`-m`** switch of the interpreter. The following example illustrates how to serve files relative to the current directory:

```
python -m http.server [OPTIONS] [port]
```

The following options are accepted:

port

> The server listens to port 8000 by default. The default can be overridden by passing the desired port number as an argument:
>
> ```
> python -m http.server 9000
> ```

-b, --bind <address>

> Specifies a specific address to which it should bind. Both IPv4 and IPv6 addresses are supported. By default, the server binds itself to all interfaces. For example, the following command causes the server to bind to localhost only:
>
> ```
> python -m http.server --bind 127.0.0.1
> ```
>
> *Added in version 3.4.*
>
> *Distinto en la versión 3.8:* Support IPv6 in the `--bind` option.

-d, --directory <dir>

> Specifies a directory to which it should serve the files. By default, the server uses the current directory. For example, the following command uses a specific directory:
>
> ```
> python -m http.server --directory /tmp/
> ```
>
> *Added in version 3.7.*

-p, --protocol <version>

Specifies the HTTP version to which the server is conformant. By default, the server is conformant to HTTP/1.0. For example, the following command runs an HTTP/1.1 conformant server:

```
python -m http.server --protocol HTTP/1.1
```

Added in version 3.11.

--cgi

`CGIHTTPRequestHandler` puede ser activado en la línea de comandos pasando la opción `--cgi`:

```
python -m http.server --cgi
```

Deprecated since version 3.13, will be removed in version 3.15: **`http.server`** command line `--cgi` support is being removed because **`CGIHTTPRequestHandler`** is being removed.

Advertencia

`CGIHTTPRequestHandler` and the `--cgi` command-line option are not intended for use by untrusted clients and may be vulnerable to exploitation. Always use within a secure environment.

--tls-cert

Specifies a TLS certificate chain for HTTPS connections:

```
python -m http.server --tls-cert fullchain.pem
```

Added in version 3.14.

--tls-key

Specifies a private key file for HTTPS connections.

This option requires `--tls-cert` to be specified.

Added in version 3.14.

--tls-password-file

Specifies the password file for password-protected private keys:

```
python -m http.server \
       --tls-cert cert.pem \
       --tls-key key.pem \
       --tls-password-file password.txt
```

This option requires `--tls-cert` to be specified.

Added in version 3.14.

Security considerations

`SimpleHTTPRequestHandler` seguirá los enlaces simbólicos cuando gestione peticiones, esto hace posible que se sirvan ficheros fuera del directorio especificado.

Methods **`BaseHTTPRequestHandler.send_header()`** and **`BaseHTTPRequestHandler.send_response_only()`** assume sanitized input and do not perform input validation such as checking for the presence of CRLF sequences. Untrusted input may result in HTTP Header injection attacks.

Earlier versions of Python did not scrub control characters from the log messages emitted to stderr from `python -m http.server` or the default **`BaseHTTPRequestHandler`** `.log_message` implementation. This could allow remote clients connecting to your server to send nefarious control codes to your terminal.

Distinto en la versión 3.12: Control characters are scrubbed in stderr logs.

`http.cookies` — HTTP state management

Source code: Lib/http/cookies.py [https://github.com/python/cpython/tree/3.14/Lib/http/cookies.py]

The `http.cookies` module defines classes for abstracting the concept of cookies, an HTTP state management mechanism. It supports both simple string-only cookies, and provides an abstraction for having any serializable data-type as cookie value.

The module formerly strictly applied the parsing rules described in the **RFC 2109** [https://datatracker.ietf.org/doc/html/rfc2109.html] and **RFC 2068** [https://datatracker.ietf.org/doc/html/rfc2068.html] specifications. It has since been discovered that MSIE 3.0x didn't follow the character rules outlined in those specs; many current-day browsers and servers have also relaxed parsing rules when it comes to cookie handling. As a result, this module now uses parsing rules that are a bit less strict than they once were.

The character set, `string.ascii_letters`, `string.digits` and ``!#$%&'*+-.^_`|~:`` denote the set of valid characters allowed by this module in a cookie name (as `key`).

Distinto en la versión 3.3: Allowed ":" as a valid cookie name character.

Nota

Al encontrar una cookie no válida, se lanza `CookieError`, por lo que si los datos de su cookie provienen de un navegador, siempre debe prepararse para los datos no válidos y detectar `CookieError` en el análisis.

exception http.cookies.CookieError

Error de excepción debido a la invalidez de **RFC 2109** [https://datatracker.ietf.org/doc/html/rfc2109.html]: atributos incorrectos, encabezado *Set-Cookie* incorrecto, etc.

class http.cookies.BaseCookie([*input*])

Esta clase es un objeto similar a un diccionario cuyas claves son cadenas de caracteres y cuyos valores son `Morsel`. Tenga en cuenta que al establecer una clave en un valor, el valor se convierte primero en `Morsel` que contiene la clave y el valor.

Si se proporciona *input*, se pasa al método `load()`.

class http.cookies.SimpleCookie([*input*])

This class derives from `BaseCookie` and overrides `value_decode()` and `value_encode()`. `SimpleCookie` supports strings as cookie values. When setting the value, `SimpleCookie` calls the builtin `str()` to convert the value to a string. Values received from HTTP are kept as strings.

Ver también

Módulo `http.cookiejar`

HTTP cookie handling for web *clients*. The `http.cookiejar` and `http.cookies` modules do not depend on each other.

RFC 2109 [https://datatracker.ietf.org/doc/html/rfc2109.html] - Mecanismo de gestión de estado HTTP

Esta es la especificación de gestión de estado implementada por este módulo.

Objetos de cookie

BaseCookie.value_decode(*val*)

Retorna una tupla `(real_value, coded_value)` de una representación de cadena de caracteres. `real_value` puede ser de cualquier tipo. Este método no decodifica en **`BaseCookie`** — existe por lo que puede ser anulado.

BaseCookie.value_encode(*val*)

Retorna una tupla `(real_value, coded_value)`. *val* puede ser de cualquier tipo, pero `coded_value` siempre se convertirá en una cadena de caracteres. Este método no codifica en **`BaseCookie`** — existe por lo que se puede anular.

En general, debería darse el caso de que **`value_encode()`** y **`value_decode()`** sean inversas en el rango de *value_decode*.

BaseCookie.output(*attrs=None*, *header='Set-Cookie:'*, *sep='\r\n'*)

Return a string representation suitable to be sent as HTTP headers. *attrs* and *header* are sent to each **`Morsel`**'s **`output()`** method. *sep* is used to join the headers together, and is by default the combination `'\r\n'` (CRLF).

BaseCookie.js_output(*attrs=None*)

Retorna un fragmento de código JavaScript que, si se ejecuta en un navegador que admita JavaScript, actuará de la misma forma que si se enviaran los encabezados HTTP.

El significado de *attrs* es el mismo que en **`output()`**.

BaseCookie.load(*rawdata*)

Si *rawdata* es una cadena de caracteres, analícela como un `HTTP_COOKIE` y agregue los valores que se encuentran allí como **`Morsel`**s. Si es un diccionario, equivale a:

```
for k, v in rawdata.items():
    cookie[k] = v
```

Objetos Morsel

class http.cookies.Morsel

Resumen de un par clave/valor, que tiene algunos atributos **RFC 2109** [https://datatracker.ietf.org/doc/html/rfc2109.html].

Morsels are dictionary-like objects, whose set of keys is constant — the valid **RFC 2109** [https://datatracker.ietf.org/doc/html/rfc2109.html] attributes, which are:

expires
path
comment
domain
max-age
secure
version
httponly
samesite
partitioned

El atributo **`httponly`** especifica que la cookie solo se transfiere en solicitudes HTTP y no es accesible a través de JavaScript. Esto tiene como objetivo mitigar algunas formas de secuencias de comandos entre sitios.

The attribute **`samesite`** controls when the browser sends the cookie with cross-site requests. This helps to mitigate CSRF attacks. Valid values are «Strict» (only sent with same-site requests), «Lax» (sent with same-site requests and top-level navigations), and «None» (sent with same-site and cross-site requests). When using «None», the «secure» attribute must also be set, as required by modern browsers.

The attribute **`partitioned`** indicates to user agents that these cross-site cookies *should* only be available in the same top-level context that the cookie was first set in. For this to be accepted by the user agent, you **must** also set `Secure`.

In addition, it is recommended to use the `__Host` prefix when setting partitioned cookies to make them bound to the hostname and not the registrable domain. Read CHIPS

(Cookies Having Independent Partitioned State) [https://github.com/privacycg/CHIPS/blob/main/README.md] for full details and examples.

Las claves no distinguen entre mayúsculas y minúsculas y su valor predeterminado es `''`.

Distinto en la versión 3.5: `__eq__()` now takes `key` and `value` into account.

Distinto en la versión 3.7: Los atributos `key`, `value` y `coded_value` son de solo lectura. Utilice `set()` para configurarlos.

Distinto en la versión 3.8: Se agregó soporte para el atributo `samesite`.

Distinto en la versión 3.14: Added support for the `partitioned` attribute.

Morsel.value

El valor de la cookie.

Morsel.coded_value

El valor codificado de la cookie — esto es lo que se debe enviar.

Morsel.key

El nombre de la cookie.

Morsel.set(*key*, *value*, *coded_value*)

Establezca los atributos *key*, *value* y *coded_value*.

Morsel.isReservedKey(*K*)

Si *K* es miembro del conjunto de claves de una `Morsel`.

Morsel.output(*attrs=None*, *header='Set-Cookie:'*)

Retorna una representación de cadena de caracteres del

Morsel, adecuada para enviarse como un encabezado HTTP. De forma predeterminada, se incluyen todos los atributos, a menos que se proporcione *attrs*, en cuyo caso debería ser una lista de atributos a utilizar. *header* es por defecto `"Set-Cookie:"`.

Morsel.js_output(*attrs=None*)

Retorna un fragmento de código JavaScript que, si se ejecuta en un navegador que admita JavaScript, actuará de la misma forma que si se hubiera enviado el encabezado HTTP.

El significado de *attrs* es el mismo que en **`output()`**.

Morsel.OutputString(*attrs=None*)

Retorna una cadena de caracteres que representa el Morsel, sin ningún HTTP o JavaScript circundante.

El significado de *attrs* es el mismo que en **`output()`**.

Morsel.update(*values*)

Actualice los valores en el diccionario Morsel con los valores en el diccionario *values*. Lanza un error si alguna de las claves en el *values* dict no es un atributo válido **RFC 2109** [https://datatracker.ietf.org/doc/html/rfc2109.html].

Distinto en la versión 3.5: se lanza un error para claves no válidas.

Morsel.copy(*value*)

Retorna una copia superficial del objeto Morsel.

Distinto en la versión 3.5: retorna un objeto Morsel en lugar de un dict.

Morsel.setdefault(*key*, *value=None*)

Lanza un error si la clave no es un atributo válido **RFC 2109** [https://datatracker.ietf.org/doc/html/rfc2109.html]; de lo contrario, se comporta igual que **`dict.setdefault()`**.

Ejemplo

The following example demonstrates how to use the **`http.cookies`** module.

```
>>> from http import cookies
>>> C = cookies.SimpleCookie()
>>> C["fig"] = "newton"
>>> C["sugar"] = "wafer"
>>> print(C) # generate HTTP headers
Set-Cookie: fig=newton
Set-Cookie: sugar=wafer
>>> print(C.output()) # same thing
Set-Cookie: fig=newton
Set-Cookie: sugar=wafer
>>> C = cookies.SimpleCookie()
>>> C["rocky"] = "road"
>>> C["rocky"]["path"] = "/cookie"
>>> print(C.output(header="Cookie:"))
Cookie: rocky=road; Path=/cookie
>>> print(C.output(attrs=[], header="Cookie:"))
Cookie: rocky=road
>>> C = cookies.SimpleCookie()
>>> C.load("chips=ahoy; vienna=finger") # load from a st
>>> print(C)
Set-Cookie: chips=ahoy
Set-Cookie: vienna=finger
>>> C = cookies.SimpleCookie()
>>> C.load('keebler="E=everybody; L=\\"Loves\\"; fudge=;
>>> print(C)
Set-Cookie: keebler="E=everybody; L=\"Loves\"; fudge=;"
>>> C = cookies.SimpleCookie()
>>> C["oreo"] = "doublestuff"
>>> C["oreo"]["path"] = "/"
>>> print(C)
Set-Cookie: oreo=doublestuff; Path=/
>>> C = cookies.SimpleCookie()
>>> C["twix"] = "none for you"
>>> C["twix"].value
'none for you'
>>> C = cookies.SimpleCookie()
>>> C["number"] = 7 # equivalent to C["number"] = str(7)
>>> C["string"] = "seven"
>>> C["number"].value
'7'
>>> C["string"].value
'seven'
>>> print(C)
Set-Cookie: number=7
Set-Cookie: string=seven
```

`http.cookiejar` — Cookie handling for HTTP clients

Código fuente: Lib/http/cookiejar.py [https://github.com/python/cpython/tree/3.14/Lib/http/cookiejar.py]

The **`http.cookiejar`** module defines classes for automatic handling of HTTP cookies. It is useful for accessing websites that require small pieces of data – *cookies* – to be set on the client machine by an HTTP response from a web server, and then returned to the server in later HTTP requests.

Both the regular Netscape cookie protocol and the protocol defined by **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html] are handled. RFC 2965 handling is switched off by default. **RFC 2109** [https://datatracker.ietf.org/doc/html/rfc2109.html] cookies are parsed as Netscape cookies and subsequently treated either as Netscape or RFC 2965 cookies according to the "policy" in effect. Note that the great majority of cookies on the internet are Netscape cookies. **`http.cookiejar`** attempts to follow the de-facto Netscape cookie protocol (which differs substantially from that set out in the original Netscape specification), including taking note of the `max-age` and `port` cookie-attributes introduced with RFC 2965.

Nota

Diversos parámetros nombrados que se encuentran en *Set-Cookie* y *Set-Cookie2* (ejem. `domain` y `expires`) son convencionalmente referidos como *attributes*. Para distinguirlos de los atributos de Python, la documentación de este módulo utiliza el término *cookie-attribute*.

El módulo define la siguiente excepción:

exception http.cookiejar.LoadError

Las instancias de `FileCookieJar` provocan esta excepción al no cargar las cookies desde un archivo. `LoadError` es una subclase de `OSError`.

Distinto en la versión 3.3: `LoadError` used to be a subtype of `IOError`, which is now an alias of `OSError`.

Se proporcionan las siguientes clases:

class http.cookiejar.CookieJar(*policy=None*)

policy es un objeto implementando la interfaz `CookiePolicy`.

La clase `CookieJar` almacena HTTP cookies. Ésta extrae cookies de los request de HTTP, y las retorna en forma de responses de HTTP. Las instancias de `CookieJar` automáticamente expiran las cookies contenidas cuando es necesario. Las subclases también son responsables de almacenar y recuperar cookies de un archivo o base de datos.

class http.cookiejar.FileCookieJar(*filename=None*, *delayload=None*, *policy=None*)

policy es un objeto implementando la interfaz `CookiePolicy`. Para los otros argumentos, te recomendamos visitar la documentación correspondiente a los atributos.

Una `CookieJar` puede cargar cookies de, y posiblemente almacenar, un archivo en el disco. Las cookies **NO** son cargadas desde un archivo nombrado hasta que alguno de los métodos `load()` o `revert()` es llamado. Subclases de esta clase son documentadas en la sección Subclases FileCookieJar y co-operación con navegadores web.

Esto no debe inicializarse directamente; en su lugar, use sus subclases a continuación.

Distinto en la versión 3.8: El parámetro filename soporta un path-like object.

class http.cookiejar.CookiePolicy

Esta clase es responsable de decidir si cada cookie debe ser aceptada del servidor, o retornada al mismo.

class http.cookiejar.DefaultCookiePolicy(*blocked_domains=None, allowed_domains=None, netscape=True, rfc2965=False, rfc2109_as_netscape=None, hide_cookie2=False, strict_domain=False, strict_rfc2965_unverifiable=True, strict_ns_unverifiable=False, strict_ns_domain=DefaultCookiePolicy.DomainLiberal, strict_ns_set_initial_dollar=False, strict_ns_set_path=False, secure_protocols=('https', 'wss')*)

Los argumentos del constructor sólo deben ser dados como argumentos clave. *blocked_domains* es una secuencia de nombres de dominio cuyas cookies nunca deben de ser aceptadas o retornadas. *allowed_domains* y no `None`, es una secuencia de sólo dominios que podemos aceptar y retornar cookies. *secure_protocols* es una secuencia de protocolos para los que se pueden agregar cookies seguras. De manera predeterminada *https* y *wss* (websocket seguro) son considerados protocolos seguros. Para todos los demás argumentos, te recomendamos visitar la documentación para los objetos `CookiePolicy` y `DefaultCookiePolicy`.

`DefaultCookiePolicy` implementa el estándar de reglas aceptar/rechazar para cookies Netscape y **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html]. De manera predeterminada, las cookies **RFC 2109** [https://datatracker.ietf.org/doc/html/rfc2109.html] (es decir, las cookies que reciben una cabecera *Set-Cookie* con una versión cookie-attribute de 1) son tratadas de acuerdo con las reglas del RFC 2965. Sin embargo, si el manejo del RFC2965 se encuentra apagado o `rfc2109_as_netscape` es `True`, las cookies RFC 2109 son "degradadas" por la instancia `CookieJar` a las cookies Netscape, configurando la `version` del atributo de la instancia `Cookie` como 0. `DefaultCookiePolicy` también provee algunos parámetros que permiten fine-tunning en el policy.

class http.cookiejar.Cookie

This class represents Netscape, RFC 2109 [https://datatracker.ietf.org/doc/html/rfc2109.html] and RFC 2965 [https://datatracker.ietf.org/doc/html/rfc2965.html] cookies. It is not expected that users of `http.cookiejar` construct their own `Cookie` instances. Instead, if necessary, call `make_cookies()` on a `CookieJar` instance.

Ver también

Módulo `urllib.request`

Apertura de URL con manejo automático de cookies.

Módulo `http.cookies`

HTTP cookie classes, principally useful for server-side code. The `http.cookiejar` and `http.cookies` modules do not depend on each other.

https://curl.se/rfc/cookie_spec.html

The specification of the original Netscape cookie protocol. Though this is still the dominant protocol, the "Netscape cookie protocol" implemented by all the major browsers (and `http.cookiejar`) only bears a passing resemblance to the one sketched out in `cookie_spec.html`.

RFC 2109 [https://datatracker.ietf.org/doc/html/rfc2109.html] - Mecanismo de Gestión de Estados HTTP

Obsoleto por RFC 2965 [https://datatracker.ietf.org/doc/html/rfc2965.html]. Usa *Set-Cookie* con versión=1.

RFC 2965 [https://datatracker.ietf.org/doc/html/rfc2965.html] - Mecanismo de Gestión de Estados HTTP

El protocolo Netscape con los errores solucionados. Usa *Set-Cookie2* en lugar de *Set-Cookie*. No es ampliamente usado.

https://kristol.org/cookie/errata.html

Fe de erratas sin finalizar hasta el RFC 2965 [https://datatracker.ietf.org/doc/html/rfc2965.html].

RFC 2964 [https://datatracker.ietf.org/doc/html/rfc2964.html] - Mecanismo de Gestión de Estados HTTP

Objetos CookieJar y FileCookieJar

Los objetos `CookieJar` soportan el protocolo iterator para iterar sobre los objetos `Cookie` contenidos.

`CookieJar` tiene los siguientes métodos:

CookieJar.add_cookie_header(*request*)

Añade la cookie correcta a la cabecera del request.

Si el *policy* admite (ie. los atributos `rfc2965` y `hide_cookie2` de la instancia `CookieJar` del `CookiePolicy` son de manera respectiva verdadero y falso), la cabecera *Cookie2* es igualmente añadido cuando es apropiado.

El objeto *request* (usualmente una instancia de `urllib.request.Request`) debe soportar los métodos `get_full_url()`, `has_header()`, `get_header()`, `header_items()`, `add_unredirected_header()` y los atributos `host`, `type`, `unverifiable` y `origin_req_host` según lo documentado por `urllib.request`.

Distinto en la versión 3.3: el objeto *request* necesita del atributo `origin_req_host`. La dependencia en el método deprecado `get_origin_req_host()` ha sido removida.

CookieJar.extract_cookies(*response*, *request*)

Extrae las cookies del *response* HTTP y las almacena en el `CookieJar`, donde está permitido por la policy.

El `CookieJar` buscará por cabeceras *Set-Cookie* y *Set-Cookie2* permitidos en el argumento *response*, y almacena las cookies cuando es apropiado (sujeto a la aprobación del método `CookiePolicy.set_ok()`).

El objeto *response* (usualmente el resultado de llamar `urllib.request.urlopen()`, o similares) debe de soportar un método `info()`, el cual retorna una instancia `email.message.Message`.

El objeto *request* (usualmente una instancia de `urllib.request.Request`) debe soportar los métodos `get_full_url()` y los atributos `host`, `unverifiable` y `origin_req_host` según lo documentado por `urllib.request`. La solicitud se utiliza para establecer valores predeterminados para los atributos de las cookies, así como para verificar si la cookie puede establecerse.

Distinto en la versión 3.3: el objeto *request* necesita del atributo `origin_req_host`. La dependencia en el método deprecado `get_origin_req_host()` ha sido removida.

CookieJar.set_policy(*policy*)

Establece la instancia `CookiePolicy` para ser usada.

CookieJar.make_cookies(*response*, *request*)

Retorna una secuencia de objetos `Cookie` extraída del objeto *response*.

Revisa la documentación `extract_cookies()` para conocer las interfaces necesarias de los argumentos *response* y *request*.

CookieJar.set_cookie_if_ok(*cookie*, *request*)

Establece una `Cookie` si la política (*policy*) dice OK para hacerlo.

CookieJar.set_cookie(*cookie*)

Establece una `Cookie`, sin consultar con la política (*policy*) para ver si se debe de establecer o no.

CookieJar.clear([*domain*[, *path*[, *name*]]])

Borra algunas cookies.

Si se invoca sin argumentos, borra todas las cookies. Si se le da un solo argumento. sólo las cookies que pertenecen a tal *domain* serán removidas. Si se le da dos argumentos, las cookies pertenecientes al especificado *domain* y *path* URL serán removidas. Si se le da tres argumentos, entonces la cookie con el especificado *domain*, *path* y *name* será removida.

Lanza **`KeyError`** si no existe ninguna cookie que coincida.

CookieJar.clear_session_cookies()

Descarta todas las cookies de la sesión.

Descarta todas las cookies que tengan un atributo **`discard`** true (usualmente porque no tienen `max-age` o `expires` atributos’cookie, o un explícito atributo-cookie `discard`). Para los navegadores interactivos, el fin de la sesión usualmente corresponde con el cierre de la ventana del navegador.

Nota que el método **`save()`** no guardará las cookies de la sesión de todas maneras, a menos que de otra manera preguntes para pasar un argumento *ignore_discard* como true.

`FileCookieJar` implementa los siguientes métodos adicionales:

FileCookieJar.save(*filename=None*, *ignore_discard=False*, *ignore_expires=False*)

Guarda las cookies en un archivo.

Esta clase base genera **`NotImplementedError`**. Las subclases podrían dejar este método sin implementar.

filename es el nombre del archivo en el que se guardarán las cookies. Si *filename* no es especificado, **`self.filename`** es utilizado (cuyo valor predeterminado es el valor que será pasado al constructor, si hay alguno); si **`self.filename`** es **`None`**, se genera un **`ValueError`**.

ignore_discard: almacena incluso las cookies configuradas para ser descartadas. *ignore_expires*: almacena las cookies que han caducado

El archivo se sobrescribe si ya existe, borrando así todas las cookies que contiene. Las cookies guardadas se pueden restaurar después utilizando los métodos `load()` o `revert()`.

FileCookieJar.load(*filename=None*, *ignore_discard=False*, *ignore_expires=False*)

Carga las cookies desde un archivo.

Las viejas cookies son guardadas a menos que sean sobre escritas por nuevas recién cargadas.

Los argumentos son en cuanto a `save()`.

El archivo nombrado debe estar en un formato entendible para la clase, o se lanzará un `LoadError`. Además, puede generarse un `OSError`, por ejemplo, si el archivo no existe.

Distinto en la versión 3.3: Solía levantarse un `IOError`, pero ahora es un alias de `OSError`.

FileCookieJar.revert(*filename=None*, *ignore_discard=False*, *ignore_expires=False*)

Limpia todas las cookies y recarga las cookies de un archivo guardado.

`revert()` puede levantar las mismas excepciones que `load()`. Si hay un fallo. el estado del objeto no se alterará.

Instancias `FileCookieJar` tienen los siguientes atributos públicos:

FileCookieJar.filename

Nombre del archivo predeterminado para el archivo en donde se almacenan las cookies. Este atributo podría ser asignado.

FileCookieJar.delayload

Si es true, carga perezosamente las cookies desde el disco. Este atributo no se debería de asignar. Esta es solo una pista, debido a que solo el rendimiento, y no al comportamiento (a menos que las cookies en el disco sean cambiadas). Un objeto **`CookieJar`** puede ignorarlo. Ninguna de las clases de **`FileCookieJar`** incluidas en la librería estándar carga las cookies de manera perezosa.

Subclases FileCookieJar y co-operación con navegadores web

Las siguientes subclases **`CookieJar`** son proveídas para lectura y escritura.

class http.cookiejar.MozillaCookieJar(*filename=None*, *delayload=None*, *policy=None*)

Un **`FileCookieJar`** que puede cargar y guardar cookies al disco en el formato de archivo `cookies.txt` de Mozilla (que también es utilizado por curl y los navegadores Lynx y Netscape).

Nota

Esto puede perder información acerca de cookies **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html], y también sobre atributos nuevos o no estándar como `port`.

Advertencia

Realiza una copia de seguridad de tus cookies antes de guardarlas, especialmente si tienes cookies cuya perdida / corrupción puede ser inconveniente (hay algunas sutilezas que pueden conducir a ligeros cambios en el archivo sobre una carga / guardado round-trip).

También toma en cuenta que las cookies guardadas mientras Mozilla se está ejecutando se volverán torpes debido a Mozilla.

class http.cookiejar.LWPCookieJar(*filename=None, delayload=None, policy=None*)

Un **`FileCookieJar`** que puede cargar y guardar en disco en formato compatible con el formato de archivo `Set-Cookie3` de la librería libwww-perl. Estos es conveniente si tu quieres guardar cookies en un archivo legible para los humanos.

Distinto en la versión 3.8: El parámetro filename soporta un path-like object.

Objetos CookiePolicy

Objetos implementando la interfaz **`CookiePolicy`** tienen los siguientes métodos:

CookiePolicy.set_ok(*cookie, request*)

Retorna un valor booleano indicando si la cookie debe ser aceptada o no desde el servidor.

cookie es una instancia de **`Cookie`**. *request* es un objeto implementando la interfaz definida por la documentación de **`CookieJar.extract_cookies()`**.

CookiePolicy.return_ok(*cookie, request*)

Retorna un valor booleano indicando si la cookie debe ser retornada o no al servidor.

cookie es una instancia de **`Cookie`**. *request* es un objeto implementando la interfaz definida por la documentación de **`CookieJar.add_cookie_header()`**.

CookiePolicy.domain_return_ok(*domain, request*)

Retorna `False` si las cookies no deben de ser retornadas, a partir del dominio cookie dado.

Este método es una optimización. Elimina la necesidad de verificar cada cookie con un domino en particular (lo cual puede involucrar la lectura de muchos archivos). Retornando true de `domain_return_ok()` y `path_return_ok()` deja todo el trabajo a `return_ok()`.

Si `domain_return_ok()` retorna true para el dominio de cookies, `path_return_ok()` es llamado para el path de cookies. De otra manera, **`path_return_ok()`** y `return_ok()` no son nunca llamados para ese dominio de cookies. Si **`path_return_ok()`** retorna true, **`return_ok()`** es llamado con el objeto `Cookie` en sí para una revisión completa. De otra manera, **`return_ok()`** es nunca llamado para ese path de cookies.

Ten en cuenta que `domain_return_ok()` es llamado para cada dominio *cookie*, no solamente para el dominio *request*. Por ejemplo, la función podría ser llamada con ambos `".example.com"` y `"www.example.com"` si el dominio request es `"www.example.com"`. Lo mismo se aplica para `path_return_ok()`.

El argumento *request* es tal y como es documentado para `return_ok()`.

CookiePolicy.path_return_ok(*path*, *request*)

Retorna `False` si las cookies no deben de ser retornadas, dado el path de las cookies.

Revisa la documentación para `domain_return_ok()`.

Adicionalmente a los métodos implementados anteriormente, las implementaciones de la interfaz `CookiePolicy` también deben de proporcionar los siguientes atributos, indicando cuales protocolos deben de ser utilizados, y como. Todos estos atributos se pueden asignar.

CookiePolicy.netscape

Implementa el protocolo Netscape.

CookiePolicy.rfc2965

Implementa el protocolo **RFC 2965** [https://datatracker.ietf.org/

doc/html/rfc2965.html].

CookiePolicy.hide_cookie2

No añadas la cabecera *Cookie2* a las requests (la presencia de esta cabecera indica al servidor que nosotros entendemos las cookies **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html]).

El camino más útil para definir una clase `CookiePolicy` es haciendo una subclase de `DefaultCookiePolicy` y sobre escribir algunos o todos los métodos anteriores. `CookiePolicy` en sí misma puede ser utilizada como una “null policy” para permitir establecer y recibir algunas o todas las cookies (esto raramente puede ser útil).

Objetos DefaultCookiePolicy

Implementa las reglas estándar para aceptar y retornar cookies.

Ambas cookies, **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html] y Netscape son cubiertas. El manejo del RFC 2965 está desactivado de forma predeterminada.

La forma más sencilla de proporcionar tu propia política (*policy*) es sobre escribir esta clase y llamar sus métodos en tus implementaciones modificadas antes de añadir tu propias comprobaciones adicionales:

```
import http.cookiejar
class MyCookiePolicy(http.cookiejar.DefaultCookiePolicy)
    def set_ok(self, cookie, request):
        if not http.cookiejar.DefaultCookiePolicy.set_ok
            return False
        if i_dont_want_to_store_this_cookie(cookie):
            return False
        return True
```

En adición a las características requeridas para implementar la interfaz `CookiePolicy`, esta clase te permite que bloquees o permitas dominios de establecer y recibir cookies. También hay algunos interruptores estrictos que te permiten ajustar un poco las reglas flojas del protocolo Netscape (a costa de bloquear algunas cookies benignas).

Se proporciona una lista de bloqueo y una lista de permitidos del dominio (ambos desactivados de forma predeterminada). Solo los dominios que no están en la lista de bloqueo y que están presentes en la lista de permisos (si la lista de permisos está activa) participan en la configuración y devolución de cookies. Utilice el argumento del constructor *blocked_domains* y los métodos **`blocked_domains()`** y **`set_blocked_domains()`** (y el argumento y los métodos correspondientes para *allowed_domains*). Si configura una lista de permitidos, puede desactivarla nuevamente configurándola en `None`.

Los dominios en listas de bloqueos o permitidos que no comienzan con un punto deben ser iguales al dominio de la cookie para que coincidan. Por ejemplo, `"example.com"` coincide con una entrada de lista de bloqueo de `"example.com"`, pero `"www.example.com"` no. Los dominios que comienzan con un punto también se corresponden con dominios más específicos. Por ejemplo, tanto `"www.example.com"` como `"www.coyote.example.com"` coinciden con `".example.com"` (pero el propio `"example.com"` no lo hace). Las direcciones IP son una excepción y deben coincidir exactamente. Por ejemplo, si block_domains contiene `"192.168.1.2"` y `".168.1.2"`, 192.168.1.2 está bloqueado, pero 193.168.1.2 no.

`DefaultCookiePolicy` implementa los siguientes métodos adicionales:

DefaultCookiePolicy.blocked_domains()

> Retorna una secuencia de dominios bloqueados (en forma de tupla).

DefaultCookiePolicy.set_blocked_domains(*blocked_domains*)

> Establece la secuencia de dominios bloqueados.

DefaultCookiePolicy.is_blocked(*domain*)

Retorna `True` si *domain* está en la lista de bloqueo para establecer o recibir cookies.

DefaultCookiePolicy.allowed_domains()

Retorna `None`, o la secuencia de los dominios permitidos (en forma de tupla).

DefaultCookiePolicy.set_allowed_domains(*allowed_domains*)

Establece la secuencia de los dominios permitidos, o `None`.

DefaultCookiePolicy.is_not_allowed(*domain*)

Retorna `True` si *domain* no está en la lista de permitidos para establecer o recibir cookies.

Las instancias `DefaultCookiePolicy` tienen los siguientes atributos, que son todos inicializados desde los argumentos del constructor del mismo nombre, y el cual todos pueden ser asignados a.

DefaultCookiePolicy.rfc2109_as_netscape

Si es true, solicita que la instancia `CookieJar` degrade las cookies **RFC 2109** [https://datatracker.ietf.org/doc/html/rfc2109.html] (es decir, las cookies recibidas en una cabecera *Set-Cookie* con un cookie-attribute versión de 1) a cookies Netscape al establecer la versión del atributo de la instancia `Cookie` a 0. El valor predeterminado es `None`, en cuyo caso las cookies RFC 2109 son degradadas si y sólo si el control de **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html] se encuentra apagado. Por lo tanto, las cookies RFC 2109 son degradadas de forma predeterminada.

Interruptores generales de rigurosidad:

DefaultCookiePolicy.strict_domain

No permite que los sitios establezcan dominios de dos componentes con dominios country-code top-level como

`.co.uk`, `.gov.uk`, `.co.nz`.etc. ¡Esto está lejos de ser perfecto y no está garantizado a que vaya a funcionar!

Interruptores de rigurosidad del protocolo **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html]:

DefaultCookiePolicy.strict_rfc2965_unverifiable

Siga las reglas **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html] sobre transacciones no verificables (normalmente, una transacción no verificable es una del resultado de una redirección o una solicitud de una imagen alojada en otro sitio). Si esto es falso, las cookies *nunca* se bloquean en base a la verificabilidad

Interruptores de rigurosidad del protocolo Netscape:

DefaultCookiePolicy.strict_ns_unverifiable

Aplica las reglas del **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html] a transacciones no verificables incluso a cookies Netscape.

DefaultCookiePolicy.strict_ns_domain

Indicadores que señalan que tan estricto deben ser con las reglas de domain-matching para cookies Netscape. Consulte a continuación los valores aceptables.

DefaultCookiePolicy.strict_ns_set_initial_dollar

Ignora las cookies en las cabeceras Set-Cookie: que tengan nombres que comienzan con `'$'`.

DefaultCookiePolicy.strict_ns_set_path

No permite establecer cookies cuyo path no concuerde con el request URI.

`strict_ns_domain` is a collection of flags. Its value is constructed by or-ing together (for example, `DomainStrictNoDots|DomainStrictNonDomain` means both flags are set).

DefaultCookiePolicy.DomainStrictNoDots

Cuando configuras las cookies, el “host prefix” no debe de contener un punto (ejem. `www.foo.bar.com` no puede establecer una cookie para `.bar.com`, porque `www.foo` tiene un punto).

DefaultCookiePolicy.DomainStrictNonDomain

Las cookies que no especifican explícitamente un cookie-attribute `domain` sólo se pueden retornar a un dominio igual al que estableció la cookie (ejem. `spam.example.com` no retornará las cookies de `example.com` que no tuvieran un cookie-attribute `domain`)

DefaultCookiePolicy.DomainRFC2965Match

Cuando se establecen las cookies, requiere de un completo emparejamiento de dominio **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html].

Los siguientes atributos son proveídos por conveniencia, y son las combinaciones más útiles de los indicadores anteriores:

DefaultCookiePolicy.DomainLiberal

Equivalente a 0 (es decir, todos los anteriores indicadores de rigurosidad de los dominios Netscape están apagados).

DefaultCookiePolicy.DomainStrict

Equivalente a `DomainStrictNoDots` | `DomainStrictNonDomain`.

Objetos Cookie

`Cookie` instances have Python attributes roughly corresponding to the standard cookie-attributes specified in the various cookie standards. The correspondence is not one-to-one, because there are complicated rules for assigning default values, because the `max-age` and `expires` cookie-attributes contain equivalent information, and because **RFC 2109** [https://datatracker.ietf.org/doc/html/rfc2109.html] cookies may be “downgraded” by **`http.cookiejar`** from version 1 to version 0 (Netscape) cookies.

La asignación a estos atributos no debe de ser necesario excepto en raras circunstancias en un método `CookiePolicy`. La clase no aplica consistencia interna, por lo que tienes que saber lo que estás haciendo si lo estas aplicando.

Cookie.version

Integer or `None`. Netscape cookies have `version` 0. **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html] and **RFC 2109** [https://datatracker.ietf.org/doc/html/rfc2109.html] cookies have a `version` cookie-attribute of 1. However, note that **`http.cookiejar`** may "downgrade" RFC 2109 cookies to Netscape cookies, in which case `version` is 0.

Cookie.name

Nombre de la cookie (un string).

Cookie.value

El valor de la cookie (una string), o `None`.

Cookie.port

String que representa un puerto o un conjunto de puertos (ejem. "80", o "80,8080") o incluso `None`.

Cookie.domain

Cookie domain (a string).

Cookie.path

Ruta de la cookie (un string, ejem. `'/acme/rocket_launchers'`).

Cookie.secure

`True` si cookie sólo debe de ser retornada sobre una conexión segura.

Cookie.expires

Fecha de caducidad entera en segundos desde la parte temporal, o `None`. Véase también el método `is_expired()`.

Cookie.discard

`True` si esta es una sesión de cookies.

Cookie.comment

Comentario string del servidor explicando la función de esta cookie, o `None`.

Cookie.comment_url

URL que enlaza a un comentario desde un servidor explicando la función de esta cookie, o `None`.

Cookie.rfc2109

`True` if this cookie was received as an RFC 2109 [https://datatracker.ietf.org/doc/html/rfc2109.html] cookie (ie. the cookie arrived in a *Set-Cookie* header, and the value of the Version cookie-attribute in that header was 1). This attribute is provided because **`http.cookiejar`** may “downgrade” RFC 2109 cookies to Netscape cookies, in which case `version` is 0.

Cookie.port_specified

`True` si un puerto o un conjunto de puertos fue explícitamente especificado por el servidor (en la cabecera *Set-Cookie* / *Set-Cookie2*).

Cookie.domain_specified

`True` si un domino fue explícitamente especificado por el servidor.

Cookie.domain_initial_dot

`True` si el dominio explícitamente especificado por el servidor empieza con un punto (`'.'`).

Las cookies pueden tener cookie-attribute no estándar adicionales.

Se pueden acceder a estos usando los siguientes métodos:

Cookie.has_nonstandard_attr(*name*)

Retorna `True` si cookie tiene nombrado el cookie-attribute.

Cookie.get_nonstandard_attr(*name*, *default=None*)

Si la cookie tiene nombrado el cookie-attribute, retorna su valor. De otra manera, retorna *default*.

Cookie.set_nonstandard_attr(*name*, *value*)

Retorna el valor del nombrado cookie-attribute.

La clase `Cookie` también define el siguiente método:

Cookie.is_expired(*now=None*)

`True` si la cookie ha excedido el tiempo que el servidor solicitó para que expirara. Si *now* es dado (en segundos desde la parte temporal), retorna si la cookie ha expirado en el momento especificado.

Ejemplos

The first example shows the most common usage of **`http.cookiejar`**:

```
import http.cookiejar, urllib.request
cj = http.cookiejar.CookieJar()
opener = urllib.request.build_opener(urllib.request.HTTP
r = opener.open("http://example.com/")
```

Este ejemplo ilustra como abrir una URL usando tus cookies Netscape, Mozilla, o Lynx (asume la convención Unix/Netscape para la ubicación del archivo de cookies):

```
import os, http.cookiejar, urllib.request
cj = http.cookiejar.MozillaCookieJar()
cj.load(os.path.join(os.path.expanduser("~"), ".netscape
opener = urllib.request.build_opener(urllib.request.HTTP
r = opener.open("http://example.com/")
```

El siguiente ejemplo ilustra el uso de **`DefaultCookiePolicy`**. Activa las cookies **RFC 2965** [https://datatracker.ietf.org/doc/html/rfc2965.html], es más estricto con respecto a los dominios cuando se establecen o se retornan cookies Netscape, y bloquea algunos dominios para establecer o retornar cookies:

```
import urllib.request
from http.cookiejar import CookieJar, DefaultCookiePolic
policy = DefaultCookiePolicy(
    rfc2965=True, strict_ns_domain=Policy.DomainStrict,
    blocked_domains=["ads.net", ".ads.net"])
cj = CookieJar(policy)
opener = urllib.request.build_opener(urllib.request.HTTP
r = opener.open("http://example.com/")
```

`xmlrpc` — Módulos XMLRPC para cliente y servidor

XML-RPC es un método de Llamada a Procedimiento Remoto (o RPC en inglés) que usa XML usando HTTP como vía de transporte. Con esto, un cliente puede llamar a método con parámetros en un servidor remoto (el servidor se identifica mediante una URI) y traer de vuelta datos de forma estructurada.

`xmlrpc` es un paquete que agrupa los módulos, tanto de cliente como de servidor, que implementan XML-RPC. Los módulos son:

- **`xmlrpc.client`**
- **`xmlrpc.server`**

[1]

`xmlrpc.client` — XML-RPC client access

Source code: Lib/xmlrpc/client.py [https://github.com/python/cpython/tree/3.14/Lib/xmlrpc/client.py]

XML-RPC es un método de llamada a procedimiento remoto que utiliza XML pasado a través de HTTP(S) como transporte. Con él, un cliente puede llamar a métodos con parámetros en un servidor remoto (el servidor es nombrado por un URI) y recuperar datos estructurados. Este módulo admite la escritura de código de cliente XML-RPC; maneja todos los detalles de la traducción entre objetos de Python conformes y XML en el cable.

Advertencia

The **`xmlrpc.client`** module is not secure against maliciously constructed data. If you need to parse untrusted or unauthenticated data, see XML security.

Distinto en la versión 3.5: For HTTPS URIs, **`xmlrpc.client`** now performs all the necessary certificate and hostname checks by default.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

class xmlrpc.client.ServerProxy(*uri*, *transport=None*, *encoding=None*, *verbose=False*, *allow_none=False*, *use_datetime=False*, *use_builtin_types=False*, *, *headers=()*, *context=None*)

Una **`ServerProxy`** instancia un objeto que gestiona la comunicación con un servidor XML-RPC remoto. El primer argumento requerido es un URI (*Uniform Resource Indicator*) y normalmente será la URL del servidor. El segundo argumento opcional es una instancia de fábrica de transporte; de forma predeterminada, es una instancia interna **`SafeTransport`** para https: URL y una instancia interna HTTP **`Transport`** en caso contrario. El tercer argumento opcional es una codificación, por defecto UTF-8. El cuarto argumento opcional es un indicador de depuración.

The following parameters govern the use of the returned proxy instance. If *allow_none* is true, the Python constant `None` will be translated into XML; the default behaviour is for `None` to raise a **`TypeError`**. This is a commonly used extension to the XML-RPC specification, but isn't supported by all clients and servers; see http://ontosys.com/xml-rpc/extensions.php [https://web.archive.org/web/20130120074804/http://ontosys.com/xml-rpc/extensions.php] for a description. The *use_builtin_types* flag can be used to cause date/time values to be presented as **`datetime.datetime`** objects and binary data to be presented as **`bytes`** objects; this flag is false by default. **`datetime.datetime`**, **`bytes`** and **`bytearray`** objects may be passed to calls. The *headers* parameter is an optional sequence of HTTP headers to send with each request, expressed as a sequence of 2-tuples representing the header name and value. (e.g. `[('Header-Name', 'value')]`). If an HTTPS URL is provided, *context* may be **`ssl.SSLContext`** and configures the SSL settings of the underlying HTTPS connection. The obsolete *use_datetime* flag is similar to *use_builtin_types* but it applies only to date/time values.

Distinto en la versión 3.3: La opción *use_builtin_types* fue añadida.

Distinto en la versión 3.8: El parámetro *headers* fue añadida.

Both the HTTP and HTTPS transports support the URL syntax extension for HTTP Basic Authentication: `http://user:pass@host:port/path`. The `user:pass` portion will be base64-encoded as an HTTP "Authorization" header, and sent to the remote server as part of the connection process when invoking an XML-RPC method. You only need to use this if the remote server requires a Basic Authentication user and password.

La instancia retornada es un objeto proxy con métodos que se pueden utilizar para invocar las correspondientes llamadas RPC en el servidor remoto. Si el servidor remoto admite la API de introspección, el proxy también se puede utilizar para consultar al servidor remoto los métodos que admite (descubrimiento de servicios) y recuperar otros metadatos asociados al servidor.

Los tipos que son conformes (por ejemplo, que se pueden clasificar a través de XML) incluyen lo siguiente (y, excepto donde se indique, no se clasifican como el mismo tipo de Python):

Tipo XML-RPC	Tipo de Python
`boolean`	`bool`
`int`, `i1`, `i2`, `i4`, `i8` o `biginteger`	`int` en el rango de -2147483648 a 2147483647. Los valores obtienen la etiqueta `<int>`.
`double` o `float`	`float`. Los valores obtienen la etiqueta `<double>`.
`string`	`str`
`array`	`list` o `tuple` que contiene elementos determinados. Las matrices se retornan como **`lists`**.
`struct`	`dict`. Las claves deben ser cadenas de caracteres, los valores pueden ser de cualquier tipo determinado. Pueden pasarse objetos de clases definidas por el usuario; sólo se transmite su atributo `__dict__`.
`dateTime.iso8601`	`DateTime` o `datetime.datetime`. El tipo retornado depende de los valores de los indicadores *use_builtin_types* y *use_datetime*.
`base64`	`Binary`, `bytes` o `bytearray`. El tipo retornado depende del valor de la marca *use_builtin_types*.
`nil`	La constante `None`. Solo se permite pasar si *allow_none* es verdadero.
`bigdecimal`	`decimal.Decimal`. Retornado solo el tipo.

Este es el conjunto completo de tipos de datos admitidos por XML-RPC. Las llamadas a métodos también pueden generar una instancia especial `Fault`, que se usa para señalar errores del servidor XML-RPC, o `ProtocolError` que se usa para señalar un error en la capa de transporte HTTP/HTTPS. Ambos **`Fault`** y **`ProtocolError`** derivan de una clase base llamada **`Error`**. Tenga en cuenta que el módulo de cliente xmlrpc actualmente no clasifica instancias de subclases de tipos integrados.

Al pasar cadenas de caracteres, los caracteres especiales de XML como <, > y `&` se escaparán automáticamente. Sin embargo, es responsabilidad de la persona que llama asegurarse de que la cadena de caracteres esté libre de caracteres que no están permitidos en XML, como los caracteres de control con valores ASCII entre 0 y 31 (excepto, por supuesto, tabulación, nueva línea y retorno de carro); no hacer esto resultará en una solicitud XML-RPC que no es XML bien formado. Si tiene que pasar bytes arbitrarios a través de XML-RPC, use las clases `bytes` o `bytearray` o la clase contenedora `Binary` descrita a continuación.

`Server` se conserva como un alias para `ServerProxy` para compatibilidad con versiones anteriores. El nuevo código debe usar **`ServerProxy`**.

Distinto en la versión 3.5: Se agregó el argumento *context*.

Distinto en la versión 3.6: Se agregó soporte para etiquetas de tipo con prefijos (por ejemplo. `ex:nil`). Se agregó soporte para desagrupar los tipos adicionales utilizados por la implementación Apache XML-RPC para números: `i1`, `i2`, `i8`, `biginteger`, `float` y `bigdecimal`. Consulte http://ws.apache.org/xmlrpc/types.html para obtener una descripción.

Ver también

XML-RPC HOWTO [https://tldp.org/HOWTO/XML-RPC-HOWTO/index.html]

Una buena descripción del funcionamiento de XML-RPC y del software cliente en varios idiomas. Contiene prácticamente todo lo que un desarrollador de cliente XML-RPC necesita saber.

XML-RPC Introspection [https://xmlrpc-c.sourceforge.io/introspection.html]
Describe la extensión del protocolo XML-RPC para la introspección.

XML-RPC Specification [http://xmlrpc.scripting.com/spec.html]
La especificación oficial.

Objetos *ServerProxy*

La instancia de `ServerProxy` tiene un método correspondiente a cada llamada de procedimiento remoto aceptada por el servidor XML-RPC. Llamar al método realiza un RPC, enviado por nombre y firma de argumento (por ejemplo, el mismo nombre de método puede sobrecargarse con múltiples firmas de argumento). El RPC finaliza retornando un valor, que puede ser datos retornados en un tipo conforme o un objeto `Fault` o `ProtocolError` que indica un error.

Los servidores que admiten la API de introspección XML admiten algunos métodos comunes agrupados bajo el atributo reservado **`system`**:

ServerProxy.system.listMethods()

Este método retorna una lista de cadenas, una para cada método (que no es del sistema) admitido por el servidor XML-RPC.

ServerProxy.system.methodSignature(*name*)

Este método toma un parámetro, el nombre de un método implementado por el servidor XML-RPC. Retorna una matriz de posibles firmas para este método. Una firma es una variedad de tipos. El primero de estos tipos es el tipo de retorno del método, el resto son parámetros.

Debido a que se permiten múltiples firmas (es decir, sobrecarga), este método retorna una lista de firmas en lugar de un singleton.

Las propias firmas están restringidas a los parámetros de nivel superior esperados por un método. Por ejemplo, si un método espera una matriz de estructuras como parámetro y retorna una cadena de caracteres, su firma es simplemente «cadena, matriz». Si espera tres enteros y retorna una cadena, su firma es «string, int, int, int».

Si no se define una firma para el método, se retorna un valor que no es una matriz. En Python, esto significa que el tipo de valor retornado será diferente a una lista.

ServerProxy.system.methodHelp(*name*)

Este método toma un parámetro, el nombre de un método implementado por el servidor XML-RPC. Retorna una cadena de caracteres de documentación que describe el uso de ese método. Si no hay tal cadena de caracteres disponible, se retorna una cadena de caracteres vacía. La cadena de caracteres de documentación puede contener marcado HTML.

Distinto en la versión 3.5: Las instancias de **`ServerProxy`** admiten el protocolo context manager para cerrar el transporte subyacente.

A continuación se muestra un ejemplo práctico. El código del servidor:

```
from xmlrpc.server import SimpleXMLRPCServer

def is_even(n):
    return n % 2 == 0

server = SimpleXMLRPCServer(("localhost", 8000))
print("Listening on port 8000...")
server.register_function(is_even, "is_even")
server.serve_forever()
```

El código de cliente para el servidor anterior:

```
import xmlrpc.client

with xmlrpc.client.ServerProxy("http://localhost:8000/")
    print("3 is even: %s" % str(proxy.is_even(3)))
    print("100 is even: %s" % str(proxy.is_even(100)))
```

Objetos *DateTime*

class xmlrpc.client.DateTime

Esta clase puede inicializarse con segundos desde la época, una tupla de tiempo, una cadena de fecha/hora ISO 8601 o una instancia **`datetime.datetime`**. Tiene los siguientes métodos, soportados principalmente para uso interno por el código de clasificación/eliminación de clasificación:

decode(*string*)

Acepta una cadena de caracteres como el nuevo valor de tiempo de la instancia.

encode(*out*)

Escribe la codificación XML-RPC de este elemento **`DateTime`** en el objeto de flujo *out*.

It also supports certain of Python’s built-in operators through **`rich comparison`** and **`__repr__()`** methods.

A continuación se muestra un ejemplo práctico. El código del servidor:

```
import datetime as dt
from xmlrpc.server import SimpleXMLRPCServer
import xmlrpc.client

def today():
    today = dt.datetime.today()
    return xmlrpc.client.DateTime(today)

server = SimpleXMLRPCServer(("localhost", 8000))
```

```
print("Listening on port 8000...")
server.register_function(today, "today")
server.serve_forever()
```

El código de cliente para el servidor anterior:

```
import xmlrpc.client
import datetime as dt

proxy = xmlrpc.client.ServerProxy("http://localhost:8000

today = proxy.today()
# convert the ISO 8601 string to a datetime object
converted = dt.datetime.strptime(today.value, "%Y%m%dT%H
print(f"Today: {converted.strftime('%d.%m.%Y, %H:%M')}")
```

Objetos binarios

class xmlrpc.client.Binary

Esta clase puede inicializarse a partir de datos de bytes (que pueden incluir NUL). El acceso principal al contenido de un objeto `Binary` lo proporciona un atributo:

data

Los datos binarios encapsulados por la instancia `Binary`. Los datos se proporcionan como un objeto `bytes`.

Los objetos `Binary` tienen los siguientes métodos, soportados principalmente para uso interno por el código de clasificación/desagrupación:

decode(*bytes*)

Acepta un objeto base64 `bytes` y se descodifica como los nuevos datos de la instancia.

encode(*out*)

Escribe la codificación XML-RPC base 64 de este elemento binario en el objeto de flujo *out*.

Los datos codificados tendrán líneas nuevas cada 76 caracteres según RFC 2045 sección 6.8 **RFC 2045 Section 6.8** [https://datatracker.ietf.org/doc/html/rfc2045.html#section-6.8], que era la especificación estándar de facto base64 cuando se escribió la especificación XML-RPC.

It also supports certain of Python’s built-in operators through `__eq__()` and `__ne__()` methods.

Ejemplo de uso de los objetos binarios. Vamos a transferir una imagen sobre XMLRPC:

```
from xmlrpc.server import SimpleXMLRPCServer
import xmlrpc.client

def python_logo():
    with open("python_logo.jpg", "rb") as handle:
        return xmlrpc.client.Binary(handle.read())

server = SimpleXMLRPCServer(("localhost", 8000))
print("Listening on port 8000...")
server.register_function(python_logo, 'python_logo')

server.serve_forever()
```

El cliente obtiene la imagen y la guarda en un archivo:

```
import xmlrpc.client

proxy = xmlrpc.client.ServerProxy("http://localhost:8000
with open("fetched_python_logo.jpg", "wb") as handle:
    handle.write(proxy.python_logo().data)
```

Objetos Faults

class xmlrpc.client.Fault

Un objeto **`Fault`** encapsula el contenido de una etiqueta de error XML-RPC. Los objetos de error tienen los siguientes atributos:

faultCode

Un entero que indica el tipo de falla.

faultString

Una cadena de caracteres que contiene un mensaje de diagnóstico asociado con el fallo.

En el siguiente ejemplo vamos a causar intencionalmente un **`Fault`** al retornar un objeto de tipo complejo. El código del servidor:

```
from xmlrpc.server import SimpleXMLRPCServer

# A marshalling error is going to occur because we're re
# complex number
def add(x, y):
    return x+y+0j

server = SimpleXMLRPCServer(("localhost", 8000))
print("Listening on port 8000...")
server.register_function(add, 'add')

server.serve_forever()
```

El código de cliente para el servidor anterior:

```
import xmlrpc.client

proxy = xmlrpc.client.ServerProxy("http://localhost:8000
try:
    proxy.add(2, 5)
except xmlrpc.client.Fault as err:
    print("A fault occurred")
    print("Fault code: %d" % err.faultCode)
    print("Fault string: %s" % err.faultString)
```

Objetos ProtocolError

class xmlrpc.client.ProtocolError

El objeto **`ProtocolError`** describe un error de protocolo en la capa de transporte subyacente (como un error 404 “no encontrado” si el servidor nombrado por el URI no existe). Tiene los siguientes atributos:

url

El URI o URL que provocó el error.

errcode

El código de error.

errmsg

El mensaje de error o la cadena de caracteres de diagnóstico.

headers

Un diccionario que contiene los encabezados de la solicitud HTTP/HTTPS que desencadenó el error.

En el siguiente ejemplo, vamos a causar intencionalmente un **`ProtocolError`** proporcionando un URI inválido:

```
import xmlrpc.client

# create a ServerProxy with a URI that doesn't respond t
proxy = xmlrpc.client.ServerProxy("http://google.com/")

try:
    proxy.some_method()
except xmlrpc.client.ProtocolError as err:
    print("A protocol error occurred")
    print("URL: %s" % err.url)
    print("HTTP/HTTPS headers: %s" % err.headers)
    print("Error code: %d" % err.errcode)
    print("Error message: %s" % err.errmsg)
```

Objetos MultiCall

El objeto **`MultiCall`** proporciona una forma de encapsular múltiples llamadas a un servidor remoto en una sola solicitud [1].

class xmlrpc.client.MultiCall(*server*)

> Crea un objeto usado para llamadas al método boxcar. *server* es el objetivo final de la llamada. Se pueden realizar llamadas al objeto de resultado, pero retornarán inmediatamente `None` y solo almacenarán el nombre y los parámetros de la llamada en el objeto **`MultiCall`**. Llamar al objeto en sí hace que todas las llamadas almacenadas se transmitan como una única solicitud de `system.multicall`. El resultado de esta llamada es un generator; iterar sobre este generador produce los resultados individuales.

A continuación se muestra un ejemplo de uso de esta clase. El código del servidor:

```
from xmlrpc.server import SimpleXMLRPCServer

def add(x, y):
    return x + y

def subtract(x, y):
    return x - y

def multiply(x, y):
    return x * y

def divide(x, y):
    return x // y

# A simple server with simple arithmetic functions
server = SimpleXMLRPCServer(("localhost", 8000))
print("Listening on port 8000...")
server.register_multicall_functions()
server.register_function(add, 'add')
server.register_function(subtract, 'subtract')
server.register_function(multiply, 'multiply')
server.register_function(divide, 'divide')
server.serve_forever()
```

El código de cliente para el servidor anterior:

```
import xmlrpc.client

proxy = xmlrpc.client.ServerProxy("http://localhost:8000
multicall = xmlrpc.client.MultiCall(proxy)
multicall.add(7, 3)
multicall.subtract(7, 3)
multicall.multiply(7, 3)
multicall.divide(7, 3)
result = multicall()

print("7+3=%d, 7-3=%d, 7*3=%d, 7//3=%d" % tuple(result))
```

Funciones de Conveniencia

xmlrpc.client.dumps(*params*, *methodname=None*, *methodresponse=None*, *encoding=None*, *allow_none=False*)

> Convert *params* into an XML-RPC request, or into a response if *methodresponse* is true. *params* can be either a tuple of arguments or an instance of the **`Fault`** exception class. If *methodresponse* is true, only a single value can be returned, meaning that *params* must be of length 1. *encoding*, if supplied, is the encoding to use in the generated XML; the default is UTF-8. Python’s **`None`** value cannot be used in standard XML-RPC; to allow using it via an extension, provide a true value for *allow_none*.

xmlrpc.client.loads(*data*, *use_datetime=False*, *use_builtin_types=False*)

> Convierte una solicitud o respuesta XML-RPC en objetos Python, un `(params, methodname)`. *params* es una tupla de argumento; *methodname* es una cadena de caracteres, o

`None` si no hay ningún nombre de método presente en el paquete. Si el paquete XML-RPC representa una condición de falla, esta función lanzará una excepción `Fault`. La opción *use_builtin_types* puede usarse para hacer que los valores de fecha/hora se presenten como objetos de `datetime.datetime` y datos binarios que se presenten como objetos de `bytes`; esta opción es falsa por defecto.

La opción obsoleta *use_datetime* es similar a *use_builtin_types* pero esto aplica solo a valores fecha/hora.

Distinto en la versión 3.3: La opción *use_builtin_types* fue añadida.

Ejemplo de uso de cliente

```
# simple test program (from the XML-RPC specification)
from xmlrpc.client import ServerProxy, Error

# server = ServerProxy("http://localhost:8000") # local
with ServerProxy("http://betty.userland.com") as proxy:

    print(proxy)

    try:
        print(proxy.examples.getStateName(41))
    except Error as v:
        print("ERROR", v)
```

Para acceder a un servidor XML-RPC a través de un proxy HTTP, debe definir un transporte personalizado. El siguiente ejemplo muestra cómo:

```
import http.client
import xmlrpc.client

class ProxiedTransport(xmlrpc.client.Transport):

    def set_proxy(self, host, port=None, headers=None):
        self.proxy = host, port
        self.proxy_headers = headers

    def make_connection(self, host):
        connection = http.client.HTTPConnection(*self.pr
        connection.set_tunnel(host, headers=self.proxy_h
        self._connection = host, connection
        return connection

transport = ProxiedTransport()
transport.set_proxy('proxy-server', 8080)
server = xmlrpc.client.ServerProxy('http://betty.userlar
print(server.examples.getStateName(41))
```

Ejemplo de uso de cliente y servidor

Vea SimpleXMLRPCServer example.

Notas al pie

Este enfoque se presentó por primera vez en una discusión en xmlrpc.com [https://web.archive.org/web/20060624230303/http://www.xmlrpc.com/discuss/msgReader$1208?mode=topic].

`xmlrpc.server` — Basic XML-RPC servers

Código fuente: Lib/xmlrpc/server.py [https://github.com/python/cpython/tree/3.14/Lib/xmlrpc/server.py]

The `xmlrpc.server` module provides a basic server framework for XML-RPC servers written in Python. Servers can either be free standing, using `SimpleXMLRPCServer`, or embedded in a CGI environment, using `CGIXMLRPCRequestHandler`.

Advertencia

The `xmlrpc.server` module is not secure against maliciously constructed data. If you need to parse untrusted or unauthenticated data, see XML security.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

class xmlrpc.server.SimpleXMLRPCServer(*addr*, *requestHandler=SimpleXMLRPCRequestHandler*, *logRequests=True*, *allow_none=False*, *encoding=None*, *bind_and_activate=True*, *use_builtin_types=False*)

Crea una nueva instancia de servidor. Esta clase proporciona métodos para el registro de funciones que pueden ser llamados por el protocolo XML-RPC. El parámetro *requestHandler* debe ser un generador para las instancias del controlador de solicitudes; el valor predeterminado es `SimpleXMLRPCRequestHandler`. Los parámetros *addr* y *requestHandler* se pasan al constructor `socketserver.`

`TCPServer`. Si *logRequests* es verdadero (valor predeterminado), se registrarán las solicitudes; establecer este parámetro como falso desactivará el registro. Los parámetros *allow_none* y *encoding* se pasan a **`xmlrpc.client`** y controlan las respuestas XML-RPC que se devolverán desde el servidor. El parámetro *bind_and_activate* controla si **`server_bind()`** y **`server_activate()`** son llamados inmediatamente por el constructor; por defecto es verdadero. Establecerlo como false permite que el código manipule la variable de clase *allow_reuse_address* antes de enlazar la dirección. El parámetro *use_builtin_types* se pasa a la función **`loads()`** y controla qué tipos se procesan cuando se reciben valores de fecha y hora o datos binarios; el valor predeterminado es false.

Distinto en la versión 3.3: Se ha añadido el indicador *use_builtin_types*.

class xmlrpc.server.CGIXMLRPCRequestHandler(*allow_none=False*, *encoding=None*, *use_builtin_types=False*)

Crea una nueva instancia para gestionar solicitudes XML-RPC en un entorno CGI. Los parámetros *allow_none* y *encoding* se pasan a **`xmlrpc.client`** y controlan las respuestas XML-RPC que se devolverán desde el servidor. El parámetro *use_builtin_types* se pasa a la función **`loads()`** y controla qué tipos se procesan cuando se reciben valores de fecha y hora o datos binarios; el valor predeterminado es falso.

Distinto en la versión 3.3: Se ha añadido el indicador *use_builtin_types*.

class xmlrpc.server.SimpleXMLRPCRequestHandler

Crea una nueva instancia del controlador de solicitudes. Este controlador de solicitudes admite solicitudes `POST` y modifica el registro para que se respete el parámetro *logRequests* del parámetro constructor **`SimpleXMLRPCServer`**.

SimpleXMLRPCServer objects

La clase `SimpleXMLRPCServer` se basa en `socketserver.TCPServer` y proporciona un medio para crear servidores XML-RPC simples e independientes.

SimpleXMLRPCServer.register_function(*function=None*, *name=None*)

> Register a function that can respond to XML-RPC requests. If *name* is given, it will be the method name associated with *function*, otherwise `function.__name__` will be used. *name* is a string, and may contain characters not legal in Python identifiers, including the period character.
>
> This method can also be used as a decorator. When used as a decorator, *name* can only be given as a keyword argument to register *function* under *name*. If no *name* is given, `function.__name__` will be used.
>
> *Distinto en la versión 3.7:* `register_function()` puede ser usado como decorador.

SimpleXMLRPCServer.register_instance(*instance*, *allow_dotted_names=False*)

> Registre un objeto que se usa para exponer nombre de métodos que no se han registrado usando `register_function()`. Si *instance* contiene un método `_dispatch()`, este será llamado con el nombre del método solicitado y los parámetros de la solicitud. Su API es `def _dispatch(self, method, params)` (tenga en cuenta que *params* no representa una lista de argumentos variables). Si se invoca a una función subyacente para realizar su tarea, esa función es llamada como `func(*params)`, expandiendo la lista de parámetros. El valor de retorno de `_dispatch()` se retorna al cliente como resultado. Si *instance* no tiene un método `_dispatch()`, se busca un atributo que coincida con el nombre del método solicitado.
>
> Si el argumento opcional *allow_dotted_names* es verdadero y la instancia no tiene un método `_dispatch()`, entonces si el nombre solicitado contiene puntos, cada componente del

nombre del método se busca individualmente, con el efecto con el efecto que produce una búsqueda jerárquica simple. El valor encontrado en esta búsqueda es entonces llamado con los parámetros de la solicitud y el valor de retorno se devuelve al cliente.

Advertencia

Habilitando la opción *allow_dotted_names* permite a los intrusos acceder a las variables globales de su módulo y puede permitir que los intrusos ejecuten código arbitrario en su máquina. Utilice esta opción únicamente en una red cerrada y segura.

SimpleXMLRPCServer.register_introspection_functions()

Registre las funciones de introspección XML-RPC `system.listMethods`, `system.methodHelp` y `system.methodSignature`.

SimpleXMLRPCServer.register_multicall_functions()

Registre la función de llamada múltiple XML-RPC system.multicall.

SimpleXMLRPCRequestHandler.rpc_paths

Un valor de atributo que debe ser una tupla que enumere porciones de rota válidas de la URL para recibir solicitudes XML-RPC. Las solicitudes publicadas en otras rutas darán como resultado un error HTTP 404 «no existe tal página». Si esta tupla está vacía, todas las rutas se considerarán válidas. El valor predeterminado es `('/', '/RPC2')`.

SimpleXMLRPCServer example

Código del servidor:

```
from xmlrpc.server import SimpleXMLRPCServer
from xmlrpc.server import SimpleXMLRPCRequestHandler
```

```
# Restrict to a particular path.
class RequestHandler(SimpleXMLRPCRequestHandler):
    rpc_paths = ('/RPC2',)

# Create server
with SimpleXMLRPCServer(('localhost', 8000),
                        requestHandler=RequestHandler) a
    server.register_introspection_functions()

    # Register pow() function; this will use the value c
    # pow.__name__ as the name, which is just 'pow'.
    server.register_function(pow)

    # Register a function under a different name
    def adder_function(x, y):
        return x + y
    server.register_function(adder_function, 'add')

    # Register an instance; all the methods of the insta
    # published as XML-RPC methods (in this case, just '
    class MyFuncs:
        def mul(self, x, y):
            return x * y

    server.register_instance(MyFuncs())

    # Run the server's main loop
    server.serve_forever()
```

El siguiente código de cliente llamará a los métodos disponibles por el servidor anterior:

```
import xmlrpc.client

s = xmlrpc.client.ServerProxy('http://localhost:8000')
print(s.pow(2,3))  # Returns 2**3 = 8
print(s.add(2,3))  # Returns 5
print(s.mul(5,2))  # Returns 5*2 = 10
```

```
# Print list of available methods
print(s.system.listMethods())
```

`register_function()` también se puede utilizar como decorador. El ejemplo de servidor anterior puede registrar funciones a modo de decorador:

```
from xmlrpc.server import SimpleXMLRPCServer
from xmlrpc.server import SimpleXMLRPCRequestHandler

class RequestHandler(SimpleXMLRPCRequestHandler):
    rpc_paths = ('/RPC2',)

with SimpleXMLRPCServer(('localhost', 8000),
                        requestHandler=RequestHandler) a
    server.register_introspection_functions()

    # Register pow() function; this will use the value o
    # pow.__name__ as the name, which is just 'pow'.
    server.register_function(pow)

    # Register a function under a different name, using
    # register_function as a decorator. *name* can only
    # as a keyword argument.
    @server.register_function(name='add')
    def adder_function(x, y):
        return x + y

    # Register a function under function.__name__.
    @server.register_function
    def mul(x, y):
        return x * y

    server.serve_forever()
```

El siguiente ejemplo incluido en el módulo `Lib/xmlrpc/server.py` muestra un servidor que permite nombres con puntos y registra una función de llamada múltiple.

Advertencia

Enabling the *allow_dotted_names* option allows intruders to access your module’s global variables and may allow intruders to execute arbitrary code on your machine. Only use this example within a secure, closed network.

```
import datetime as dt

class ExampleService:
    def getData(self):
        return '42'

    class currentTime:
        @staticmethod
        def getCurrentTime():
            return dt.datetime.now()

with SimpleXMLRPCServer(("localhost", 8000)) as server:
    server.register_function(pow)
    server.register_function(lambda x,y: x+y, 'add')
    server.register_instance(ExampleService(), allow_dot
    server.register_multicall_functions()
    print('Serving XML-RPC on localhost port 8000')
    try:
        server.serve_forever()
    except KeyboardInterrupt:
        print("\nKeyboard interrupt received, exiting.")
        sys.exit(0)
```

Esta demostración de ExampleService se puede invocar desde la línea de comando:

```
python -m xmlrpc.server
```

El cliente que interactúa con el servidor anterior está incluido en *Lib/xmlrpc/client.py*:

```
server = ServerProxy("http://localhost:8000")
```

```
try:
    print(server.currentTime.getCurrentTime())
except Error as v:
    print("ERROR", v)

multi = MultiCall(server)
multi.getData()
multi.pow(2,9)
multi.add(1,2)
try:
    for response in multi():
        print(response)
except Error as v:
    print("ERROR", v)
```

Este cliente que interactúa con el servidor XMLRPC de demostración se puede invocar como:

```
python -m xmlrpc.client
```

CGIXMLRPCRequestHandler

La clase **`CGIXMLRPCRequestHandler`** se puede usar para manejar solicitudes XML-RPC enviadas a scripts Python CGI.

CGIXMLRPCRequestHandler.register_function(*function=None*, *name=None*)

> Register a function that can respond to XML-RPC requests. If *name* is given, it will be the method name associated with *function*, otherwise **`function.__name__`** will be used. *name* is a string, and may contain characters not legal in Python identifiers, including the period character.
>
> This method can also be used as a decorator. When used as a decorator, *name* can only be given as a keyword argument to register *function* under *name*. If no *name* is given, **`function.__name__`** will be used.

Distinto en la versión 3.7: `register_function()` puede ser usado como decorador.

CGIXMLRPCRequestHandler.register_instance(*instance*)

Registra un objeto que se usa para exponer nombres de métodos que no se han registrado usando `register_function()`. Si la instancia contiene un método **`_dispatch()`**, se llama con el nombre del método solicitado y los parámetros de la solicitud; el valor de retorno se devuelve al cliente como resultado. Si la instancia no tiene un método **`_dispatch()`**, se busca un atributo que coincida con el nombre del método solicitado; si el nombre del método contiene puntos, cada componente del nombre del método se busca individualmente, con el efecto con el efecto que produce una búsqueda jerárquica simple. El valor encontrado en esta búsqueda es entonces llamado con los parámetros de la solicitud y el valor de retorno se devuelve al cliente.

CGIXMLRPCRequestHandler.register_introspection_functions()

Registra las funciones de introspección `system.listMethods`, `system.methodHelp` y `system.methodSignature`.

CGIXMLRPCRequestHandler.register_multicall_functions()

Registra la función de llamada múltiple XML-RPC `system.multicall`.

CGIXMLRPCRequestHandler.handle_request(*request_text=None*)

Maneja una solicitud XML-RPC. Si se proporciona *request_text*, deberían ser los datos POST proporcionados por el servidor HTTP, de lo contrario se utilizará el contenido de stdin.

Ejemplo:

```
class MyFuncs:
    def mul(self, x, y):
        return x * y
```

```
handler = CGIXMLRPCRequestHandler()
handler.register_function(pow)
handler.register_function(lambda x,y: x+y, 'add')
handler.register_introspection_functions()
handler.register_instance(MyFuncs())
handler.handle_request()
```

Documentando el servidor XMLRPC

Estas clases amplían las clases anteriores para proporcionar documentación HTML en respuesta a solicitudes HTTP GET. Los servidores pueden ser independientes, usando **`DocXMLRPCServer`**, o integrados en un entorno CGI, usando **`DocCGIXMLRPCRequestHandler`**.

class xmlrpc.server.DocXMLRPCServer(*addr*, *requestHandler=DocXMLRPCRequestHandler*, *logRequests=True*, *allow_none=False*, *encoding=None*, *bind_and_activate=True*, *use_builtin_types=True*)

> Crea una nueva instancia de servidor. Todos los parámetros tienen el mismo significado que para **`SimpleXMLRPCServer`**; *requestHandler* tiene como valor predeterminado **`DocXMLRPCRequestHandler`**.
>
> *Distinto en la versión 3.3:* Se ha añadido el indicador *use_builtin_types*.

class xmlrpc.server.DocCGIXMLRPCRequestHandler

> Crea una nueva instancia para manejar solicitudes XML-RPC en un entorno CGI.

class xmlrpc.server.DocXMLRPCRequestHandler

> Crea una nueva instancia de controlador de solicitudes. Este controlador de solicitudes admite solicitudes XML-RPC POST, solicitudes GET de documentación y modifica el registro para que se respete el parámetro *logRequests* del parámetro constructor **`DocXMLRPCServer`**.

DocXMLRPCServer objects

La clase `DocXMLRPCServer` se deriva de `SimpleXMLRPCServer` y proporciona un medio para crear servidores XML-RPC autónomos y autodocumentados. Las solicitudes HTTP POST se manejan como llamadas al método XML-RPC. Las solicitudes HTTP GET se manejan generando documentación HTML al estilo pydoc. Esto permite que un servidor proporcione su propia documentación basada en web.

DocXMLRPCServer.set_server_title(*server_title*)

> Establezca el título utilizado en la documentación HTML generada. Este título se utilizará dentro del elemento HTML «title».

DocXMLRPCServer.set_server_name(*server_name*)

> Establezca el nombre utilizado en la documentación HTML generada. Este nombre aparecerá en la parte superior de la documentación generada dentro de un elemento «h1».

DocXMLRPCServer.set_server_documentation(*server_documentation*)

> Establezca la descripción utilizada en la documentación HTML generada. Esta descripción aparecerá como un párrafo, debajo del nombre del servidor, en la documentación.

DocCGIXMLRPCRequestHandler

La clase `DocCGIXMLRPCRequestHandler` se deriva de `CGIXMLRPCRequestHandler` y proporciona un medio para crear scripts CGI XML-RPC autodocumentados. Las solicitudes HTTP POST se manejan como llamadas al método XML-RPC. Las solicitudes HTTP GET se manejan generando documentación HTML al estilo pydoc. Esto permite que un servidor proporcione su propia documentación basada en web.

DocCGIXMLRPCRequestHandler.set_server_title(*server_title*)

> Establezca el título utilizado en la documentación HTML

generada. Este título se utilizará dentro del elemento HTML «title».

DocCGIXMLRPCRequestHandler.set_server_name(*server_name*)

Establezca el nombre utilizado en la documentación HTML generada. Este nombre aparecerá en la parte superior de la documentación generada dentro de un elemento «h1».

DocCGIXMLRPCRequestHandler.set_server_documentation(*server_documenta*

Establezca la descripción utilizada en la documentación HTML generada. Esta descripción aparecerá como un párrafo, debajo del nombre del servidor, en la documentación.

`ipaddress` — IPv4/IPv6 manipulation library

Source code: Lib/ipaddress.py [https://github.com/python/cpython/tree/3.14/Lib/ipaddress.py]

`ipaddress` provides the capabilities to create, manipulate and operate on IPv4 and IPv6 addresses and networks.

Las funciones y clases de este módulo facilitan el manejo de varias tareas relacionadas con las direcciones IP, incluida la verificación de si dos hosts están en la misma subred, la iteración sobre todos los hosts de una subred en particular, la verificación de si una cadena representa un valor válido o no. Dirección IP o definición de red, etc.

Esta es la referencia completa de la API del módulo; para obtener una descripción general y una introducción, consulte Introducción al modulo ipaddress.

Added in version 3.3.

Funciones de fábrica de conveniencia

The **`ipaddress`** module provides factory functions to conveniently create IP addresses, networks and interfaces:

ipaddress.ip_address(*address*)

Retorna un objeto **`IPv4Address`** o **`IPv6Address`** según la dirección IP pasada como argumento. Se pueden proporcionar direcciones IPv4 o IPv6; los enteros menores que `2**32` se considerarán IPv4 de forma predeterminada. Se lanza un **`ValueError`** si *address* no representa una dirección IPv4 o IPv6 válida.

```
>>> ipaddress.ip_address('192.168.0.1')
IPv4Address('192.168.0.1')
>>> ipaddress.ip_address('2001:db8::')
IPv6Address('2001:db8::')
```

ipaddress.ip_network(*address*, *strict=True*)

Retorna un objeto `IPv4Network` o `IPv6Network` dependiendo de la dirección IP pasada como argumento. *address* es una cadena o entero que representa la red IP. Se pueden suministrar redes IPv4 o IPv6; los enteros menores que `2**32` se considerarán IPv4 de forma predeterminada. *strict* se pasa a **`IPv4Network`** o **`IPv6Network`** constructor. Se genera un `ValueError` si *address* no representa una dirección IPv4 o IPv6 válida, o si la red tiene bits de host establecidos.

```
>>> ipaddress.ip_network('192.168.0.0/28')
IPv4Network('192.168.0.0/28')
```

ipaddress.ip_interface(*address*)

Retorna un objeto `IPv4Interface` o `IPv6Interface` dependiendo de la dirección IP pasada como argumento. *address* es una cadena o entero que representa la dirección IP. Se pueden proporcionar direcciones IPv4 o IPv6; los enteros menores que `2**32` se considerarán IPv4 de forma predeterminada. Se genera un `ValueError` si *address* no representa una dirección IPv4 o IPv6 válida.

Una desventaja de estas funciones de conveniencia es que la necesidad de manejar los formatos IPv4 e IPv6 significa que los mensajes de error brindan información mínima sobre el error exacto, ya que las funciones no saben si se pretendía usar el formato IPv4 o IPv6. Se pueden obtener informes de errores más detallados llamando directamente a los constructores de clases específicos de la versión adecuada.

Direcciones IP

Objetos de dirección

Los objetos `IPv4Address` y `IPv6Address` comparten muchos atributos comunes. Algunos atributos que solo son significativos para direcciones IPv6 también son implementados por objetos **`IPv4Address`**, a fin de facilitar la escritura de código que maneje ambas versiones de IP correctamente. Los objetos de dirección son hashable, por lo que se pueden usar como claves en diccionarios.

class ipaddress.IPv4Address(*address*)

Construya una dirección IPv4. Se genera un `AddressValueError` si *address* no es una dirección IPv4 válida.

Lo siguiente constituye una dirección IPv4 válida:

1. Una cadena en notación de punto decimal, que consta de cuatro enteros decimales en el rango inclusivo 0-255, separados por puntos (por ejemplo, `192.168.0.1`). Cada entero representa un octeto (byte) en la dirección. No se toleran ceros iniciales para evitar confusiones con la notación octal.
2. Un número entero que cabe en 32 bits.
3. Un entero empaquetado en un objeto `bytes` de longitud 4 (primero el octeto más significativo).

```
>>> ipaddress.IPv4Address('192.168.0.1')
IPv4Address('192.168.0.1')
>>> ipaddress.IPv4Address(3232235521)
IPv4Address('192.168.0.1')
>>> ipaddress.IPv4Address(b'\xC0\xA8\x00\x01')
IPv4Address('192.168.0.1')
```

Distinto en la versión 3.8: Se toleran los ceros iniciales, incluso en casos ambiguos que parecen notación octal.

Distinto en la versión 3.9.5: Los ceros iniciales ya no se toleran y se tratan como un error. Las cadenas de direcciones IPv4 ahora se analizan tan estrictamente como glibc `inet_pton()`.

version

El número de versión apropiado: `4` para IPv4, `6` para IPv6.

Distinto en la versión 3.14: Made available on the class.

max_prefixlen

El número total de bits en la representación de direcciones para esta versión: `32` para IPv4, `128` para IPv6.

El prefijo define el número de bits iniciales en una dirección que se comparan para determinar si una dirección es parte de una red o no.

Distinto en la versión 3.14: Made available on the class.

compressed

exploded

Representación de cadena en notación decimal con puntos. Los ceros iniciales nunca se incluyen en la representación.

Como IPv4 no define una notación abreviada para direcciones con octetos establecidos en cero, estos dos atributos son siempre los mismos que `str(addr)` para direcciones IPv4. La exposición de estos atributos facilita la escritura de código de visualización que pueda manejar direcciones IPv4 e IPv6.

packed

La representación binaria de esta dirección: un objeto **`bytes`** de la longitud adecuada (primero el octeto más significativo). Son 4 bytes para IPv4 y 16 bytes para IPv6.

reverse_pointer

El nombre del registro PTR de DNS inverso para la dirección IP, por ejemplo:

```
>>> ipaddress.ip_address("127.0.0.1").reverse_p
'1.0.0.127.in-addr.arpa'
>>> ipaddress.ip_address("2001:db8::1").reverse
'1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
```

Este es el nombre que podría usarse para realizar una búsqueda PTR, no el nombre de host resuelto en sí.

Added in version 3.5.

is_multicast

`True` si la dirección está reservada para uso de multidifusión. Consulte **RFC 3171** [https://datatracker.ietf.org/doc/html/rfc3171.html] (para IPv4) o **RFC 2373** [https://datatracker.ietf.org/doc/html/rfc2373.html] (para IPv6).

is_private

`True` if the address is defined as not globally reachable by iana-ipv4-special-registry [https://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.xhtml] (for IPv4) or iana-ipv6-special-registry [https://www.iana.org/assignments/iana-ipv6-special-registry/iana-ipv6-special-registry.xhtml] (for IPv6) with the following exceptions:

- `is_private` is `False` for the shared address space (`100.64.0.0/10`)
- For IPv4-mapped IPv6-addresses the `is_private` value is determined by the semantics of the underlying IPv4 addresses and the following condition holds (see **`IPv6Address.ipv4_mapped`**):

  ```
  address.is_private == address.ipv4_mapped.
  ```

`is_private` has value opposite to **`is_global`**, except for the shared address space (`100.64.0.0/10`

range) where they are both `False`.

Distinto en la versión 3.13: Fixed some false positives and false negatives.

- `192.0.0.0/24` is considered private with the exception of `192.0.0.9/32` and `192.0.0.10/32` (previously: only the `192.0.0.0/29` sub-range was considered private).
- `64:ff9b:1::/48` is considered private.
- `2002::/16` is considered private.
- There are exceptions within `2001::/23` (otherwise considered private): `2001:1::1/128`, `2001:1::2/128`, `2001:3::/32`, `2001:4:112::/48`, `2001:20::/28`, `2001:30::/28`. The exceptions are not considered private.

is_global

`True` if the address is defined as globally reachable by iana-ipv4-special-registry [https://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.xhtml] (for IPv4) or iana-ipv6-special-registry [https://www.iana.org/assignments/iana-ipv6-special-registry/iana-ipv6-special-registry.xhtml] (for IPv6) with the following exception:

For IPv4-mapped IPv6-addresses the `is_private` value is determined by the semantics of the underlying IPv4 addresses and the following condition holds (see **`IPv6Address.ipv4_mapped`**):

```
address.is_global == address.ipv4_mapped.is_glo
```

`is_global` has value opposite to **`is_private`**, except for the shared address space (`100.64.0.0/10` range) where they are both `False`.

Added in version 3.4.

Distinto en la versión 3.13: Fixed some false positives

and false negatives, see **`is_private`** for details.

is_unspecified

`True` si la dirección no está especificada. Consulte **RFC 5735** [https://datatracker.ietf.org/doc/html/rfc5735.html] (para IPv4) o **RFC 2373** [https://datatracker.ietf.org/doc/html/rfc2373.html] (para IPv6).

is_reserved

`True` if the address is noted as reserved by the IETF. For IPv4, this is only `240.0.0.0/4`, the `Reserved` address block. For IPv6, this is all addresses allocated [https://www.iana.org/assignments/ipv6-address-space/ipv6-address-space.xhtml] as `Reserved by IETF` for future use.

Nota

For IPv4, `is_reserved` is not related to the address block value of the `Reserved-by-Protocol` column in iana-ipv4-special-registry [https://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.xhtml].

Prudencia

For IPv6, `fec0::/10` a former Site-Local scoped address prefix is currently excluded from that list (see **`is_site_local`** & **RFC 3879** [https://datatracker.ietf.org/doc/html/rfc3879.html]).

is_loopback

`True` si se trata de una dirección de bucle invertido. Consulte **RFC 3330** [https://datatracker.ietf.org/doc/html/rfc3330.html] (para IPv4) o **RFC 2373** [https://datatracker.ietf.org/doc/html/rfc2373.html] (para IPv6).

is_link_local

`True` si se trata de una dirección de bucle invertido. Consulte **RFC 3330** [https://datatracker.ietf.org/doc/html/rfc3330.html] (para IPv4) o **RFC 2373** [https://datatracker.ietf.org/doc/html/rfc2373.html] (para IPv6). `True` si la dirección está reservada para uso local de enlace. Ver **RFC 3927** [https://datatracker.ietf.org/doc/html/rfc3927.html].

ipv6_mapped

`IPv4Address` object representing the IPv4-mapped IPv6 address. See **RFC 4291** [https://datatracker.ietf.org/doc/html/rfc4291.html].

Added in version 3.13.

IPv4Address.__format__(*fmt*)

Devuelve una representación de cadena de la dirección IP, controlada por una cadena de formato explícito. *fmt* puede ser uno de los siguientes: `'s'`, la opción predeterminada, equivalente a **`str()`**, `'b'` para una cadena binaria con relleno de ceros, `'X'` o `'x'` para una representación hexadecimal en mayúsculas o minúsculas, o `'n'`, que es equivalente a `'b'` para direcciones IPv4 y `'x'` para IPv6. Para representaciones binarias y hexadecimales, el especificador de formulario `'#'` y la opción de agrupación `'_'` están disponibles. `__format__` es utilizado por `format`, `str.format` y f-strings.

```
>>> format(ipaddress.IPv4Address('192.168.0.1'))
'192.168.0.1'
>>> '{:#b}'.format(ipaddress.IPv4Address('192.168.0
'0b11000000101010000000000000000001'
>>> f'{ipaddress.IPv6Address("2001:db8::1000"):s}'
'2001:db8::1000'
>>> format(ipaddress.IPv6Address('2001:db8::1000'),
'2001_0DB8_0000_0000_0000_0000_0000_1000'
>>> '{:#_n}'.format(ipaddress.IPv6Address('2001:db8
'0x2001_0db8_0000_0000_0000_0000_0000_1000'
```

Added in version 3.9.

class ipaddress.IPv6Address(*address*)

Construya una dirección IPv6. Se genera un **`AddressValueError`** si *address* no es una dirección IPv6 válida.

Lo siguiente constituye una dirección IPv6 válida:

1. Una cadena que consta de ocho grupos de cuatro dígitos hexadecimales, cada grupo representa 16 bits. Los grupos están separados por dos puntos. Esto describe una notación *explotada* (a mano). La cadena también se puede *comprimir* (notación abreviada) por varios medios. Consulte **RFC 4291** [https://datatracker.ietf.org/doc/html/rfc4291.html] para obtener más detalles. Por ejemplo, `"0000:0000:0000:0000:0000:0abc:0007:0def"` se puede comprimir en `"::abc:7:def"`.

   Opcionalmente, la cadena también puede tener un ID de zona de alcance, expresado con un sufijo `%scope_id`. Si está presente, el ID de alcance no debe estar vacío y no puede contener `%`. Consulte **RFC 4007** [https://datatracker.ietf.org/doc/html/rfc4007.html] para obtener más detalles. Por ejemplo, `fe80::1234%1` podría identificar la dirección `fe80::1234` en el primer enlace del nodo.

2. Un número entero que cabe en 128 bits.

3. Un entero empaquetado en un objeto **`bytes`** de longitud 16, big-endian.

```
>>> ipaddress.IPv6Address('2001:db8::1000')
IPv6Address('2001:db8::1000')
>>> ipaddress.IPv6Address('ff02::5678%1')
IPv6Address('ff02::5678%1')
```

compressed

La forma corta de la representación de la dirección, con ceros a la izquierda en los grupos omitidos y la secuencia más larga de grupos que consta completamente de ceros colapsó en un solo grupo vacío.

Este también es el valor devuelto por `str(addr)` para las direcciones IPv6.

exploded

La forma larga de la representación de la dirección, con todos los ceros a la izquierda y los grupos que constan completamente de ceros incluidos.

Para los siguientes atributos y métodos, consulte la documentación correspondiente de la clase **`IPv4Address`**:

packed

reverse_pointer

version

max_prefixlen

is_multicast

is_private

is_global

Added in version 3.4.

is_unspecified

is_reserved

is_loopback

is_link_local

is_site_local

`True` si la dirección está reservada para uso local del sitio. Tenga en cuenta que el espacio de direcciones local del sitio ha quedado obsoleto por **RFC 3879** [https://datatracker.ietf.org/doc/html/rfc3879.html]. Utilice **`is_private`** para probar si esta dirección está en el espacio de direcciones locales únicas como se define en **RFC 4193** [https://datatracker.ietf.org/doc/html/rfc4193.html].

ipv4_mapped

Para las direcciones que parecen ser direcciones IPv4 asignadas (que comienzan con `::FFFF/96`), esta propiedad informará la dirección IPv4 incrustada. Para cualquier otra dirección, esta propiedad será `None`.

scope_id

Para las direcciones de ámbito definido por **RFC 4007** [https://datatracker.ietf.org/doc/html/rfc4007.html], esta propiedad identifica la zona particular del ámbito de la dirección a la que pertenece la dirección, como una cadena. Cuando no se especifica ninguna zona de alcance, esta propiedad será `None`.

sixtofour

Para direcciones que parecen ser direcciones 6to4 (comenzando con `2002::/16`) según lo definido por **RFC 3056** [https://datatracker.ietf.org/doc/html/rfc3056.html], esta propiedad informará la dirección IPv4 incrustada. Para cualquier otra dirección, esta propiedad será `None`.

teredo

Para las direcciones que parecen ser direcciones Teredo (que comienzan con `2001::/32`) según lo definido por **RFC 4380** [https://datatracker.ietf.org/doc/html/rfc4380.html], esta propiedad informará el par de direcciones IP `(servidor, cliente)` incrustado. Para cualquier otra dirección, esta propiedad será

`None.`

IPv6Address.__format__(*fmt*)

Consulte la documentación del método correspondiente en **`IPv4Address`**.

Added in version 3.9.

Conversión a cadenas y enteros

Para interoperar con interfaces de red como el módulo de socket, las direcciones deben convertirse en cadenas o números enteros. Esto se maneja usando las funciones integradas **`str()`** y **`int()`**

```
>>> str(ipaddress.IPv4Address('192.168.0.1'))
'192.168.0.1'
>>> int(ipaddress.IPv4Address('192.168.0.1'))
3232235521
>>> str(ipaddress.IPv6Address('::1'))
'::1'
>>> int(ipaddress.IPv6Address('::1'))
1
```

Tenga en cuenta que las direcciones de ámbito IPv6 se convierten en números enteros sin ID de zona de ámbito.

Operadores

Los objetos de dirección admiten algunos operadores. A menos que se indique lo contrario, los operadores solo se pueden aplicar entre objetos compatibles (es decir, IPv4 con IPv4, IPv6 con IPv6).

Operadores de comparación

Los objetos de dirección se pueden comparar con el conjunto habitual de operadores de comparación. Las mismas direcciones IPv6 con diferentes ID de zona de alcance no son iguales. Algunos ejemplos:

```
>>> IPv4Address('127.0.0.2') > IPv4Address('127.0.0.1')
True
>>> IPv4Address('127.0.0.2') == IPv4Address('127.0.0.1')
False
>>> IPv4Address('127.0.0.2') != IPv4Address('127.0.0.1')
True
>>> IPv6Address('fe80::1234') == IPv6Address('fe80::1234
False
>>> IPv6Address('fe80::1234%1') != IPv6Address('fe80::12
True
```

Operadores aritméticos

Los números enteros se pueden sumar o restar de los objetos de dirección. Algunos ejemplos:

```
>>> IPv4Address('127.0.0.2') + 3
IPv4Address('127.0.0.5')
>>> IPv4Address('127.0.0.2') - 3
IPv4Address('126.255.255.255')
>>> IPv4Address('255.255.255.255') + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ipaddress.AddressValueError: 4294967296 (>= 2**32) is no
```

Definiciones de red IP

Los objetos **`IPv4Network`** y **`IPv6Network`** proporcionan un mecanismo para definir e inspeccionar las definiciones de red IP. Una definición de red consta de una * máscara * y una * dirección de red * y, como tal, define un rango de direcciones IP que son iguales a la dirección de red cuando están enmascaradas (Y binario) con la máscara. Por ejemplo, una definición de red con la máscara `255.255.255.0` y la dirección de red `192.168.1.0` consta de direcciones IP en el rango inclusivo de `192.168.1.0` a `192.168.1.255`.

Prefijo, máscara de red y máscara de host

Hay varias formas equivalentes de especificar máscaras de red IP. Un *prefix* `/<nbits>` es una notación que indica cuántos bits de orden superior se establecen en la máscara de red. Una máscara de red (*mask*) es una dirección IP con una cierta cantidad de bits de orden superior establecidos. Por lo tanto, el prefijo `/24` es equivalente a la máscara de red `255.255.255.0` en IPv4, o `ffff:ff00::` en IPv6. Además, una * máscara de host * es el inverso lógico de una * máscara de red * y, a veces, se utiliza (por ejemplo, en las listas de control de acceso de Cisco) para indicar una máscara de red. La máscara de host equivalente a `/24` en IPv4 es `0.0.0.255`.

Objetos de red

Todos los atributos implementados por objetos de dirección también son implementados por objetos de red. Además, los objetos de red implementan atributos adicionales. Todos estos son comunes entre **`IPv4Network`** y **`IPv6Network`**, por lo que para evitar la duplicación solo se documentan para **`IPv4Network`**. Los objetos de red son hashable, por lo que se pueden usar como claves en diccionarios.

class ipaddress.IPv4Network(*address*, *strict=True*)

Construya una definición de red IPv4. *address* puede ser una de las siguientes:

1. Una cadena que consta de una dirección IP y una máscara opcional, separadas por una barra (`/`). La dirección IP es la dirección de red y la máscara puede ser un solo número, lo que significa que es un *prefix*, o una representación de cadena de caracteres de una dirección IPv4. Si es la última, la máscara es interpretada como una *net mask* si comienza con un campo no nulo, o como una *host mask* si comienza con un campo cero, con la excepción de una máscara de solo ceros la cual es tratada como una *net mask*. Si no se proporciona una máscara, se considera `/32`.

   Por ejemplo, las siguientes especificaciones de *address* son equivalentes: `192.168.1.0/24`, `192.168.1.0/255.255.255.0` y `192.168.1.0/0.0.0.255`.

2. Un número entero que cabe en 32 bits. Esto es equivalente a una red de una sola dirección, con la dirección de red *address* y la máscara `/32`.

3. Un entero empaquetado en un objeto **`bytes`** de longitud 4, big-endian. La interpretación es similar a una *address* entera.

4. A two-tuple of an address description and a netmask, where the address description is either a string, a 32-bits integer, a 4-bytes packed integer, or an existing **`IPv4Address`** object; and the netmask is either an integer representing the prefix length (e.g. `24`) or a string representing the prefix mask (e.g. `255.255.255.0`).

Se genera un **`AddressValueError`** si *address* no es una dirección IPv4 válida. A **`NetmaskValueError`** se genera si la máscara no es válida para una dirección IPv4.

Si *strict* es `True` y los bits de host están configurados en la dirección proporcionada, entonces **`ValueError`** se genera. De lo contrario, los bits del host se enmascaran para determinar la dirección de red adecuada.

A menos que se indique lo contrario, todos los métodos de red que acepten otros objetos de red / dirección generarán **`TypeError`** si la versión de IP del argumento es incompatible con `self`.

Distinto en la versión 3.5: Se agregó la forma de dos tuplas para el parámetro de constructor *address*.

version

max_prefixlen

Consulte la documentación del atributo correspondiente en **`IPv4Address`**.

is_multicast

is_private

is_unspecified

is_reserved

is_loopback

is_link_local

> Estos atributos son verdaderos para la red en su conjunto si lo son tanto para la dirección de red como para la dirección de transmisión.

network_address

> La dirección de red de la red. La dirección de red y la longitud del prefijo juntos definen de forma única una red.

broadcast_address

> La dirección de transmisión de la red. Todos los hosts de la red deben recibir los paquetes enviados a la dirección de transmisión.

hostmask

> La máscara de host, como un objeto `IPv4Address`.

netmask

> La máscara de red, como un objeto `IPv4Address`.

with_prefixlen

compressed

exploded

> Una representación de cadena de la red, con la máscara

en notación de prefijo.

`with_prefixlen` y `compressed` son siempre lo mismo que `str(red)`. `exploded` utiliza la forma explosionada de la dirección de red.

with_netmask

Una representación de cadena de la red, con la máscara en notación de máscara de red.

with_hostmask

Una representación de cadena de la red, con la máscara en notación de máscara de host.

num_addresses

El número total de direcciones en la red.

prefixlen

Longitud del prefijo de red, en bits.

hosts()

Longitud de Devuelve un iterador sobre los hosts utilizables en la red. Los hosts utilizables son todas las direcciones IP que pertenecen a la red, excepto la dirección de red en sí y la dirección de transmisión de la red. Para redes con una longitud de máscara de 31, la dirección de red y la dirección de transmisión de red también se incluyen en el resultado. Las redes con una máscara de 32 devolverán una lista que contiene la dirección de host única, el prefijo de red, en bits.

```
>>> list(ip_network('192.0.2.0/29').hosts())
[IPv4Address('192.0.2.1'), IPv4Address('192.0.2
 IPv4Address('192.0.2.3'), IPv4Address('192.0.2
 IPv4Address('192.0.2.5'), IPv4Address('192.0.2
>>> list(ip_network('192.0.2.0/31').hosts())
[IPv4Address('192.0.2.0'), IPv4Address('192.0.2
>>> list(ip_network('192.0.2.1/32').hosts())
```

```
[IPv4Address('192.0.2.1')]
```

overlaps(*other*)

`True` si esta red está total o parcialmente contenida en *other* o *other* está completamente contenido en esta red.

address_exclude(*network*)

Calcula las definiciones de red resultantes de eliminar la *network* dada de esta. Devuelve un iterador de objetos de red. Genera **`ValueError`** si *network* no está completamente contenido en esta red.

```
>>> n1 = ip_network('192.0.2.0/28')
>>> n2 = ip_network('192.0.2.1/32')
>>> list(n1.address_exclude(n2))
[IPv4Network('192.0.2.8/29'), IPv4Network('192.
 IPv4Network('192.0.2.2/31'), IPv4Network('192.
```

subnets(*prefixlen_diff=1*, *new_prefix=None*)

Las subredes que se unen para crear la definición de red actual, según los valores de los argumentos. *prefixlen_diff* es la cantidad en la que se debe aumentar la longitud de nuestro prefijo. *new_prefix* es el nuevo prefijo deseado de las subredes; debe ser más grande que nuestro prefijo. Se debe establecer uno y solo uno de *prefixlen_diff* y *new_prefix*. Devuelve un iterador de objetos de red.

```
>>> list(ip_network('192.0.2.0/24').subnets())
[IPv4Network('192.0.2.0/25'), IPv4Network('192.
>>> list(ip_network('192.0.2.0/24').subnets(pre
[IPv4Network('192.0.2.0/26'), IPv4Network('192.
 IPv4Network('192.0.2.128/26'), IPv4Network('19
>>> list(ip_network('192.0.2.0/24').subnets(new
[IPv4Network('192.0.2.0/26'), IPv4Network('192.
 IPv4Network('192.0.2.128/26'), IPv4Network('19
>>> list(ip_network('192.0.2.0/24').subnets(new
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    raise ValueError('new prefix must be longer
ValueError: new prefix must be longer
>>> list(ip_network('192.0.2.0/24').subnets(new
[IPv4Network('192.0.2.0/25'), IPv4Network('192.
```

supernet(*prefixlen_diff=1*, *new_prefix=None*)

La superred que contiene esta definición de red, según los valores de los argumentos. *prefixlen_diff* es la cantidad en la que se debe reducir la longitud de nuestro prefijo. *new_prefix* es el nuevo prefijo deseado de la superred; debe ser más pequeño que nuestro prefijo. Se debe establecer uno y solo uno de *prefixlen_diff* y *new_prefix*. Devuelve un solo objeto de red.

```
>>> ip_network('192.0.2.0/24').supernet()
IPv4Network('192.0.2.0/23')
>>> ip_network('192.0.2.0/24').supernet(prefixl
IPv4Network('192.0.0.0/22')
>>> ip_network('192.0.2.0/24').supernet(new_pre
IPv4Network('192.0.0.0/20')
```

subnet_of(*other*)

Devuelve `True` si esta red es una subred de *other*.

```
>>> a = ip_network('192.168.1.0/24')
>>> b = ip_network('192.168.1.128/30')
>>> b.subnet_of(a)
True
```

Added in version 3.7.

supernet_of(*other*)

Devuelve `True` si esta red es una superred de *other*.

```
>>> a = ip_network('192.168.1.0/24')
>>> b = ip_network('192.168.1.128/30')
>>> a.supernet_of(b)
True
```

Added in version 3.7.

compare_networks(*other*)

Compare esta red con *other*. En esta comparación solo se consideran las direcciones de red; los bits de host no lo son. Devuelve `-1`, `0` o `1`.

```
>>> ip_network('192.0.2.1/32').compare_networks
-1
>>> ip_network('192.0.2.1/32').compare_networks
1
>>> ip_network('192.0.2.1/32').compare_networks
0
```

Obsoleto desde la versión 3.7: Utiliza el mismo algoritmo de ordenación y comparación que «<», «= =» y «>»

class ipaddress.IPv6Network(*address*, *strict=True*)

Construya una definición de red IPv6. *address* puede ser una de las siguientes:

1. Una cadena que consta de una dirección IP y una longitud de prefijo opcional, separados por una barra (/). La dirección IP es la dirección de red y la longitud del prefijo debe ser un solo número, el *prefix*. Si no se proporciona una longitud de prefijo, se considera que es `/128`.

   Tenga en cuenta que las máscaras de red expandidas actualmente no son compatibles. Eso significa que `2001:db00::0/24` es un argumento válido mientras que `2001:db00::0/ffff:ff00::` no lo es.

2. Un número entero que cabe en 128 bits. Esto es equivalente a una red de una sola dirección, con la dirección de red *address* y la máscara `/128`.

3. Un entero empaquetado en un objeto **`bytes`** de longitud 16, big-endian. La interpretación es similar a una *address* entera.

4. A two-tuple of an address description and a netmask, where the address description is either a string, a 128-bits integer, a 16-bytes packed integer, or an existing **`IPv6Address`** object; and the netmask is an integer representing the prefix length.

Se genera un **`AddressValueError`** si *address* no es una dirección IPv6 válida. A **`NetmaskValueError`** se genera si la máscara no es válida para una dirección IPv6.

Si *strict* es `True` y los bits de host están configurados en la dirección proporcionada, entonces **`ValueError`** se genera. De lo contrario, los bits del host se enmascaran para determinar la dirección de red adecuada.

Distinto en la versión 3.5: Se agregó la forma de dos tuplas para el parámetro de constructor *address*.

version

max_prefixlen

is_multicast

is_private

is_unspecified

is_reserved

is_loopback

is_link_local

network_address

broadcast_address

hostmask

netmask

with_prefixlen

compressed

exploded

with_netmask

with_hostmask

num_addresses

prefixlen

hosts()

> Devuelve un iterador sobre los hosts utilizables en la red. Los hosts utilizables son todas las direcciones IP que pertenecen a la red, excepto la dirección Anycast del Subnet-Router. Para redes con una longitud de máscara de 127, la dirección anycast del Subnet-Router también se incluye en el resultado. Las redes con una máscara de 128 devolverán una lista que contiene la única dirección de host.

overlaps(*other*)

address_exclude(*network*)

subnets(*prefixlen_diff=1*, *new_prefix=None*)

supernet(*prefixlen_diff=1*, *new_prefix=None*) subnet_of(*other*)

supernet_of(*other*)

compare_networks(*other*)

Consulte la documentación del atributo correspondiente en `IPv4Network`.

is_site_local

This attribute is true for the network as a whole if it is true for both the network address and the broadcast address.

Operadores

Este atributo es verdadero para la red en su conjunto si es cierto para ambos objetos de red que admiten algunos operadores. A menos que se indique lo contrario, los operadores solo se pueden aplicar entre objetos compatibles (es decir, IPv4 con IPv4, IPv6 con IPv6).

Operadores lógicos

Los objetos de red se pueden comparar con el conjunto habitual de operadores lógicos. Los objetos de red se ordenan primero por dirección de red y luego por máscara de red.

Iteración

Los objetos de red se pueden iterar para enumerar todas las direcciones que pertenecen a la red. Para la iteración, se devuelven *todos* los hosts, incluidos los hosts inutilizables (para hosts utilizables, use el método `hosts()`). Un ejemplo:

```
>>> for addr in IPv4Network('192.0.2.0/28'):
...     addr
...
IPv4Address('192.0.2.0')
```

```
IPv4Address('192.0.2.1')
IPv4Address('192.0.2.2')
IPv4Address('192.0.2.3')
IPv4Address('192.0.2.4')
IPv4Address('192.0.2.5')
IPv4Address('192.0.2.6')
IPv4Address('192.0.2.7')
IPv4Address('192.0.2.8')
IPv4Address('192.0.2.9')
IPv4Address('192.0.2.10')
IPv4Address('192.0.2.11')
IPv4Address('192.0.2.12')
IPv4Address('192.0.2.13')
IPv4Address('192.0.2.14')
IPv4Address('192.0.2.15')
```

Redes como contenedores de direcciones

Los objetos de red pueden actuar como contenedores de direcciones. Algunos ejemplos:

```
>>> IPv4Network('192.0.2.0/28')[0]
IPv4Address('192.0.2.0')
>>> IPv4Network('192.0.2.0/28')[15]
IPv4Address('192.0.2.15')
>>> IPv4Address('192.0.2.6') in IPv4Network('192.0.2.0/2
True
>>> IPv4Address('192.0.3.6') in IPv4Network('192.0.2.0/2
False
```

Objetos de interfaz

Los objetos de interfaz son hashable, por lo que se pueden usar como claves en diccionarios.

class ipaddress.IPv4Interface(*address*)

Construya una interfaz IPv4. El significado de *address* es como en el constructor de `IPv4Network`, excepto que siempre se aceptan direcciones de host arbitrarias.

`IPv4Interface` es una subclase de **`IPv4Address`**, por lo que hereda todos los atributos de esa clase. Además, están disponibles los siguientes atributos:

ip

The address (**`IPv4Address`**) without network information.

```
>>> interface = IPv4Interface('192.0.2.5/24')
>>> interface.ip
IPv4Address('192.0.2.5')
```

network

La red (**`IPv4Network`**) a la que pertenece esta interfaz.

```
>>> interface = IPv4Interface('192.0.2.5/24')
>>> interface.network
IPv4Network('192.0.2.0/24')
```

with_prefixlen

Una representación de cadena de la interfaz con la máscara en notación de prefijo.

```
>>> interface = IPv4Interface('192.0.2.5/24')
>>> interface.with_prefixlen
'192.0.2.5/24'
```

with_netmask

Una representación de cadena de la interfaz con la red como máscara de red.

```
>>> interface = IPv4Interface('192.0.2.5/24')
>>> interface.with_netmask
'192.0.2.5/255.255.255.0'
```

with_hostmask

Una representación de cadena de la interfaz con la red como máscara de host.

```
>>> interface = IPv4Interface('192.0.2.5/24')
>>> interface.with_hostmask
'192.0.2.5/0.0.0.255'
```

class ipaddress.IPv6Interface(*address*)

Construya una interfaz IPv6. El significado de *address* es como en el constructor de **`IPv6Network`**, excepto que siempre se aceptan direcciones de host arbitrarias.

`IPv6Interface` es una subclase de **`IPv6Address`**, por lo que hereda todos los atributos de esa clase. Además, están disponibles los siguientes atributos:

ip

network

with_prefixlen

with_netmask

with_hostmask

Consulte la documentación del atributo correspondiente en **`IPv4Interface`**.

Operadores

Los objetos de interfaz admiten algunos operadores. A menos que se indique lo contrario, los operadores solo se pueden aplicar entre objetos compatibles (es decir, IPv4 con IPv4, IPv6 con IPv6).

Operadores lógicos

Los objetos de interfaz se pueden comparar con el conjunto habitual de operadores lógicos.

Para la comparación de igualdad (== y `!=`), Tanto la dirección IP como la red deben ser iguales para que los objetos sean iguales. Una interfaz no se comparará con ninguna dirección u objeto de red.

Para ordenar (<, >, etc) las reglas son diferentes. Los objetos de interfaz y dirección con la misma versión de IP se pueden comparar, y los objetos de dirección siempre se clasificarán antes que los objetos de interfaz. Primero se comparan dos objetos de interfaz por sus redes y, si son iguales, luego por sus direcciones IP.

Otras funciones de nivel de módulo

El módulo también proporciona las siguientes funciones de nivel de módulo:

ipaddress.v4_int_to_packed(*address*)

> Represente una dirección como 4 bytes empaquetados en orden de red (big-endian). *address* es una representación entera de una dirección IP IPv4. A `ValueError` se genera si el número entero es negativo o demasiado grande para ser una dirección IP IPv4.
>
> ```
> >>> ipaddress.ip_address(3221225985)
> IPv4Address('192.0.2.1')
> >>> ipaddress.v4_int_to_packed(3221225985)
> b'\xc0\x00\x02\x01'
> ```

ipaddress.v6_int_to_packed(*address*)

> Representa una dirección como 16 bytes empaquetados en orden de red (big-endian). *address* es una representación entera de una dirección IP IPv6. A `ValueError` se genera si el número entero es negativo o demasiado grande para ser una dirección IP IPv6.

ipaddress.summarize_address_range(*first, last*)

> Devuelve un iterador del rango de red resumido dadas la primera y la última dirección IP. *first* es el primero `IPv4Address` o `IPv6Address` en el rango y *last* es el último `IPv4Address` o `IPv6Address` en el rango. A

`TypeError` se genera si *first* o *last* no son direcciones IP o no son de la misma versión. A `ValueError` se genera si *last* no es mayor que *first* o si la *first* versión de la dirección no es 4 o 6.

```
>>> [ipaddr for ipaddr in ipaddress.summarize_addre
...    ipaddress.IPv4Address('192.0.2.0'),
...    ipaddress.IPv4Address('192.0.2.130'))]
[IPv4Network('192.0.2.0/25'), IPv4Network('192.0.2.
```

ipaddress.collapse_addresses(*addresses*)

Return an iterator of the collapsed `IPv4Network` or `IPv6Network` objects. *addresses* is an iterable of **`IPv4Network`** or **`IPv6Network`** objects. A `TypeError` is raised if *addresses* contains mixed version objects.

```
>>> [ipaddr for ipaddr in
... ipaddress.collapse_addresses([ipaddress.IPv4Net
... ipaddress.IPv4Network('192.0.2.128/25')])]
[IPv4Network('192.0.2.0/24')]
```

ipaddress.get_mixed_type_key(*obj*)

Devuelve una clave adecuada para clasificar entre redes y direcciones. Los objetos de dirección y red no se pueden ordenar de forma predeterminada; son fundamentalmente diferentes, por lo que la expresión

```
IPv4Address('192.0.2.0') <= IPv4Network('192.0.2.0/
```

doesn't make sense. There are some times however, where you may wish to have **`ipaddress`** sort these anyway. If you need to do this, you can use this function as the *key* argument to `sorted()`.

obj es un objeto de red o de dirección.

Excepciones personalizadas

Para admitir informes de errores más específicos de los constructores de clases, el módulo define las siguientes excepciones:

exception ipaddress.AddressValueError(*ValueError*)

Cualquier error de valor relacionado con la dirección.

exception ipaddress.NetmaskValueError(*ValueError*)

Cualquier error de valor relacionado con la máscara de red.

Servicios Multimedia

Los módulos descritos en este capítulo implementan varios algoritmos o interfaces que son útiles principalmente para aplicaciones multimedia. Su disponibilidad depende de la instalación. He aquí una visión general:

- **`wave`** — Read and write WAV files
    - Los objetos *Wave_read*
    - Los objetos *Wave_write*
- **`colorsys`** — Conversions between color systems

`wave` — Read and write WAV files

Código fuente: Lib/wave.py [https://github.com/python/cpython/tree/3.14/Lib/wave.py]

The **`wave`** module provides a convenient interface to the Waveform Audio «WAVE» (or «WAV») file format. Only uncompressed PCM encoded wave files are supported.

Distinto en la versión 3.12: Support for `WAVE_FORMAT_EXTENSIBLE` headers was added, provided that the extended format is `KSDATAFORMAT_SUBTYPE_PCM`.

The **`wave`** module defines the following function and exception:

wave.open(*file*, *mode=None*)

Si *file* es una cadena, abra el archivo con ese nombre, de lo contrario trátelo como un objeto similar a un archivo. *mode* puede ser:

`'rb'`
Modo de solo lectura.

`'wb'`
Modo de solo escritura.

Tenga en cuenta que no permite archivos WAV de lectura/escritura.

Un *mode* de `'rb'` retorna un objeto **`Wave_read`**, mientras que un *mode* de `'wb'` retorna un objeto **`Wave_write`**. Si se omite *mode* y se pasa un objeto similar a un archivo como *file*, `file.mode` se usa como el valor predeterminado para *mode*.

If you pass in a file-like object, the wave object will not close it when its `close()` method is called; it is the caller's responsibility to close the file object.

The **`open()`** function may be used in a **`with`** statement. When the **`with`** block completes, the **`Wave_read.close()`** or **`Wave_write.close()`** method is called.

Distinto en la versión 3.4: Se agregó soporte para archivos no encontrados.

exception wave.Error

Error que se produce cuando algo es imposible porque viola la especificación WAV o alcanza una deficiencia de implementación.

Los objetos *Wave_read*

class wave.Wave_read

Read a WAV file.

Los objetos *Wave_read*, tal como lo retorna **`open()`**, tienen los siguientes métodos:

close()

Close the stream if it was opened by **`wave`**, and make the instance unusable. This is called automatically on object collection.

getnchannels()

Retorna el número de canales de audio (`1` para mono, `2` para estéreo).

getsampwidth()

Retorna el ancho de la muestra en bytes.

getframerate()

Retorna la frecuencia del muestreo.

getnframes()

Retorna el número de cuadros del audio.

getcomptype()

Retorna el tipo de compresión (`'NONE'` es el único tipo admitido).

getcompname()

Versión legible para humanos de **`getcomptype()`**. Generalmente `'not compressed'` significa `'NONE'`.

getparams()

Returns a **`namedtuple()`** `(nchannels, sampwidth, framerate, nframes, comptype, compname)`, equivalent to output of the `get*()` methods.

readframes(*n*)

Lee y retorna como máximo *n* cuadros de audio, como un objeto **`bytes`**.

rewind()

Rebobina el puntero del archivo hasta el principio de la secuencia de audio.

The following two methods are defined for compatibility with the old **`aifc`** module, and don't do anything interesting.

getmarkers()

Retorna `None`.

Deprecated since version 3.13, will be removed in version 3.15: The method only existed for compatibility with the **`aifc`** module which has been removed in Python

3.13.

getmark(*id*)

Lanza un error.

Deprecated since version 3.13, will be removed in version 3.15: The method only existed for compatibility with the **`aifc`** module which has been removed in Python 3.13.

Los dos métodos siguientes definen un término «posición» que es compatible entre ellos y, es dependiente de la implementación.

setpos(*pos*)

Establece el puntero del archivo en la posición especificada.

tell()

Retorna la posición actual del puntero del archivo.

Los objetos *Wave_write*

class wave.Wave_write

Write a WAV file.

Wave_write objects, as returned by **`open()`**.

For seekable output streams, the `wave` header will automatically be updated to reflect the number of frames actually written. For unseekable streams, the *nframes* value must be accurate when the first frame data is written. An accurate *nframes* value can be achieved either by calling **`setnframes()`** or **`setparams()`** with the number of frames that will be written before **`close()`** is called and then using **`writeframesraw()`** to write the frame data, or by calling **`writeframes()`** with all of the frame data to be written. In the latter case **`writeframes()`** will calculate the number of frames in the data and set *nframes* accordingly before writing the frame data.

Distinto en la versión 3.4: Se agregó soporte para archivos no encontrados.

Wave_write objects have the following methods:

close()

> Make sure *nframes* is correct, and close the file if it was opened by `wave`. This method is called upon object collection. It will raise an exception if the output stream is not seekable and *nframes* does not match the number of frames actually written.

setnchannels(*n*)

> Configure el número de canales.

getnchannels()

> Return the number of channels.

setsampwidth(*n*)

> Establezca el ancho de la muestra en *n* bytes.

getsampwidth()

> Return the sample width in bytes.

setframerate(*n*)

> Establezca la velocidad del cuadro en *n*.
>
> *Distinto en la versión 3.2:* Una entrada no-entera para este método se redondea al número entero más cercano.

getframerate()

> Return the frame rate.

setnframes(*n*)

Establezca el número de cuadros en *n*. Esto se cambiará más adelante si el número de cuadros realmente escritos es diferente (este intento de actualización levantará un error si no se encuentra la secuencia de salida).

getnframes()

Return the number of audio frames written so far.

setcomptype(*type*, *name*)

Establece el tipo de compresión y la descripción. Por el momento, solo se admite el tipo de compresión `NONE`, lo que significa que no hay compresión.

getcomptype()

Return the compression type (`'NONE'`).

getcompname()

Return the human-readable compression type name.

setparams(*tuple*)

The *tuple* should be `(nchannels, sampwidth, framerate, nframes, comptype, compname)`, with values valid for the `set*()` methods. Sets all parameters.

getparams()

Return a **`namedtuple()`** `(nchannels, sampwidth, framerate, nframes, comptype, compname)` containing the current output parameters.

tell()

Retorna la posición actual en el archivo, con el mismo descargo para los métodos **`Wave_read.tell()`** y **`Wave_read.setpos()`**.

writeframesraw(*data*)

Escribe cuadros de audio, sin corregir *nframes*.

Distinto en la versión 3.4: Todo bytes-like object ahora es aceptado.

writeframes(*data*)

Escribe cuadros de audio y se asegura de que *nframes* sea correcto. Levantará un error si no se puede encontrar la secuencia de salida y si el número total de cuadros que se han escrito después de que se haya escrito *data* no coincide con el valor establecido previamente para *nframes*.

Distinto en la versión 3.4: Todo bytes-like object ahora es aceptado.

Tenga en cuenta que no es válido establecer ningún parámetro después de invocar a **`writeframes()`** o **`writeframesraw()`**, y cualquier intento de hacerlo levantará **`wave.Error`**.

`colorsys` — Conversions between color systems

Código fuente: Lib/colorsys.py [https://github.com/python/cpython/tree/3.14/Lib/colorsys.py]

The `colorsys` module defines bidirectional conversions of color values between colors expressed in the RGB (Red Green Blue) color space used in computer monitors and three other coordinate systems: YIQ, HLS (Hue Lightness Saturation) and HSV (Hue Saturation Value). Coordinates in all of these color spaces are floating-point values. In the YIQ space, the Y coordinate is between 0 and 1, but the I and Q coordinates can be positive or negative. In all other spaces, the coordinates are all between 0 and 1.

Ver también

Puede encontrar más información sobre los espacios de color en http://poynton.ca/ColorFAQ.html y https://www.cambridgeincolour.com/tutorials/color-spaces.htm.

The `colorsys` module defines the following functions:

colorsys.rgb_to_yiq(*r*, *g*, *b*)

Convierte el color de las coordenadas RGB en coordenadas YIQ.

colorsys.yiq_to_rgb(*y*, *i*, *q*)

Convierte el color de las coordenadas YIQ en coordenadas RGB.

colorsys.rgb_to_hls(*r*, *g*, *b*)

Convierte el color de las coordenadas RGB en coordenadas HLS.

colorsys.hls_to_rgb(*h*, *l*, *s*)

Convierte el color de las coordenadas HLS en coordenadas RGB.

colorsys.rgb_to_hsv(*r*, *g*, *b*)

Convierte el color de las coordenadas RGB en coordenadas HSV.

colorsys.hsv_to_rgb(*h*, *s*, *v*)

Convierte el color de las coordenadas HSV en coordenadas RGB.

Ejemplo:

```
>>> import colorsys
>>> colorsys.rgb_to_hsv(0.2, 0.4, 0.4)
(0.5, 0.5, 0.4)
>>> colorsys.hsv_to_rgb(0.5, 0.5, 0.4)
(0.2, 0.4, 0.4)
```

Internacionalización

Los módulos descritos en este capítulo te ayudan a escribir software que es independiente del idioma y lugar al proporcionar mecanismos para seleccionar un idioma a ser usado en mensajes de programas o adaptar la salida para que coincida con las convenciones locales.

La lista de módulos descritos en este capítulo es:

- `gettext` — Multilingual internationalization services
    - GNU API **gettext**
    - API basada en clases
        - La clase `NullTranslations`
        - La clase `GNUTranslations`
        - Soporte de catálogo de mensajes de Solaris
        - El constructor del catálogo
    - Internacionalizando sus programas y módulos
        - Agregar configuración regional a su módulo
        - Agregar configuración regional a su aplicación
        - Cambiar idiomas sobre la marcha
        - Traducciones diferidas
    - Agradecimientos
- `locale` — Internationalization services
    - Segundo plano, detalles, indicaciones, consejos y advertencias
    - Locale names
    - Para escritores de extensión y programas que incrustan Python
    - Acceso a los catálogos de mensajes

[1] [2]

`gettext` — Multilingual internationalization services

Código fuente: Lib/gettext.py [https://github.com/python/cpython/tree/3.14/Lib/gettext.py]

The `gettext` module provides internationalization (I18N) and localization (L10N) services for your Python modules and applications. It supports both the GNU **gettext** message catalog API and a higher level, class-based API that may be more appropriate for Python files. The interface described below allows you to write your module and application messages in one natural language, and provide a catalog of translated messages for running under different natural languages.

También se dan algunas sugerencias para localizar sus módulos y aplicaciones Python.

GNU API gettext

The `gettext` module defines the following API, which is very similar to the GNU **gettext** API. If you use this API you will affect the translation of your entire application globally. Often this is what you want if your application is monolingual, with the choice of language dependent on the locale of your user. If you are localizing a Python module, or if your application needs to switch languages on the fly, you probably want to use the class-based API instead.

gettext.bindtextdomain(*domain*, *localedir=None*)

Bind the *domain* to the locale directory *localedir*. More concretely, `gettext` will look for binary `.mo` files for the given domain using the path (on Unix): `localedir/language/LC_MESSAGES/domain.mo`, where *language* is searched for in the environment variables **LANGUAGE**,

`LC_ALL`, **`LC_MESSAGES`**, and **`LANG`** respectively.

Si *localedir* se omite o es `None`, se retorna el enlace actual para *domain*. [1]

gettext.textdomain(*domain=None*)

Cambia o consulta el dominio global actual. Si *domain* es `None`, se retorna el dominio global actual; de lo contrario, el dominio global se establece en *domain*, que se retorna.

gettext.gettext(*message*)

Retorna la traducción localizada de *message*, basada en el dominio global actual, el idioma y el directorio de configuración regional. Esta función suele tener un alias como **`_()`** en el espacio de nombres local (ver ejemplos a continuación).

gettext.dgettext(*domain, message*)

Como **`gettext()`**, pero busque el mensaje en el *domain* especificado.

gettext.ngettext(*singular, plural, n*)

Como **`gettext()`**, pero considere formas plurales. Si se encuentra una traducción, aplique la fórmula plural a *n* y retorna el mensaje resultante (algunos idiomas tienen más de dos formas plurales). Si no se encuentra ninguna traducción, retorna *singular* if *n* es 1; retorna *plural* de lo contrario.

La fórmula Plural se toma del encabezado del catálogo. Es una expresión C o Python que tiene una variable libre *n*; la expresión se evalúa como el índice del plural en el catálogo. Consulte la documentación de GNU gettext [https://www.gnu.org/software/gettext/manual/gettext.html] para conocer la sintaxis precisa que se utilizará en archivos `.po` y las fórmulas para un variedad de idiomas.

gettext.dngettext(*domain, singular, plural, n*)

Como `ngettext()`, pero busque el mensaje en el *domain* especificado.

gettext.pgettext(*context*, *message*)

gettext.dpgettext(*domain*, *context*, *message*)

gettext.npgettext(*context*, *singular*, *plural*, *n*)

gettext.dnpgettext(*domain*, *context*, *singular*, *plural*, *n*)

Similar a las funciones correspondientes sin la `p` en el prefijo (es decir, `gettext()`, `dgettext()`, `ngettext()`, `dngettext()`), pero la traducción está restringida al mensaje dado *context*.

Added in version 3.8.

Tenga en cuenta que **gettext** de GNU también define un método **`dcgettext()`**, pero esto no se consideró útil, por lo que actualmente no está implementado.

Aquí hay un ejemplo del uso típico de esta API:

```
import gettext
gettext.bindtextdomain('myapplication', '/path/to/my/lar
gettext.textdomain('myapplication')
_ = gettext.gettext
# ...
print(_('This is a translatable string.'))
```

API basada en clases

The class-based API of the **`gettext`** module gives you more flexibility and greater convenience than the GNU **gettext** API. It is the recommended way of localizing your Python applications and modules. **`gettext`** defines a `GNUTranslations` class which implements the parsing of GNU `.mo` format files, and has methods for returning strings. Instances of this class can also install themselves in the built-in namespace as the function `_()`.

gettext.find(*domain*, *localedir=None*, *languages=None*, *all=False*)

Esta función implementa el algoritmo de búsqueda de archivos estándar `.mo`. Toma un *domain*, idéntico al que toma `textdomain()`. Opcional *localedir* es como en `bindtextdomain()`. *languages* opcionales es una lista de cadenas, donde cada cadena es un código de idioma.

Si no se proporciona *localedir*, se utiliza el directorio de configuración regional predeterminado del sistema. [2] Si no se proporcionan *languages*, se buscan las siguientes variables de entorno: **`LANGUAGE`**, **`LC_ALL`**, **`LC_MESSAGES`**, y **`LANG`**. El primero que retorna un valor no vacío se usa para la variable *languages*. Las variables de entorno deben contener una lista de idiomas separados por dos puntos, que se dividirá en dos puntos para producir la lista esperada de cadenas de código de idioma.

`find()` luego expande y normaliza los idiomas, y luego los itera, buscando un archivo existente construido con estos componentes:

`localedir``/`*`language`*`/LC_MESSAGES/`*`domain`*`.mo`

El primer nombre de archivo que existe es retornado por `find()`. Si no se encuentra dicho archivo, se retorna `None`. Si se proporciona *all*, retorna una lista de todos los nombres de archivo, en el orden en que aparecen en la lista de idiomas o las variables de entorno.

gettext.translation(*domain*, *localedir=None*, *languages=None*, *class_=None*, *fallback=False*)

Retorna una instancia de `*Translations` basada en *domain*, *localedir* y *languages*, que primero se pasan a `find()` para obtener una lista de las rutas de archivos `.mo` asociadas. Las instancias con nombres de archivos `.mo` idénticos se almacenan en caché. La clase concreta instanciada es *class_* si se proporciona, de lo contrario será `GNUTranslations`. El constructor de esta clase debe aceptar un único argumento de tipo file object.

Si se encuentran varios archivos, los archivos posteriores se utilizan como retrocesos para los anteriores. Para permitir la configuración de la reserva, `copy.copy()` se utiliza para clonar cada objeto de traducción del caché; los datos de la instancia real aún se comparten con el caché.

Si no se encuentra el archivo `.mo`, esta función genera `OSError` si *fallback* es falso (que es el valor predeterminado) y retorna una instancia de `NullTranslations` si *fallback* is verdadero.

Distinto en la versión 3.3: `IOError` used to be raised, it is now an alias of `OSError`.

Distinto en la versión 3.11: el parámetro *codeset* se removió.

gettext.install(*domain*, *localedir=None*, *, *names=None*)

Esto instala la función `_()` en el espacio de nombres incorporado de Python, basado en *domain* y *localedir* que se pasan a la función `translation()`.

Para el parámetro *names*, consulte la descripción del método del objeto de traducción `install()`.

Como se ve a continuación, generalmente marca las cadenas en su aplicación que son candidatas para la traducción, envolviéndolas en una llamada a la función `_()`, como esta:

```
print(_('This string will be translated.'))
```

Para mayor comodidad, desea que la función `_()` se instale en el espacio de nombres integrado de Python, para que sea fácilmente accesible en todos los módulos de su aplicación.

Distinto en la versión 3.11: *names* ahora es un parámetro de solo palabra clave.

La clase `NullTranslations`

Las clases de traducción son las que realmente implementan la traducción de cadenas de mensajes del archivo fuente original a cadenas de mensajes traducidas. La clase base utilizada por todas las clases de traducción es `NullTranslations`; Esto proporciona la interfaz básica que puede utilizar para escribir sus propias clases de traducción especializadas. Estos son los métodos de **`NullTranslations`**:

class gettext.NullTranslations(*fp=None*)

> Toma un término opcional file object *fp*, que es ignorado por la clase base. Inicializa las variables de instancia «protegidas» *_info* y *_charset* que se establecen mediante clases derivadas, así como *_fallback*, que se establece a través de `add_fallback()`. Luego llama a `self._parse(fp)` if *fp* no es `None`.
>
> _parse(*fp*)
>
> > No operativo en la clase base, este método toma el objeto de archivo *fp* y lee los datos del archivo, inicializando su catálogo de mensajes. Si tiene un formato de archivo de catálogo de mensajes no admitido, debe sobrescribir este método para analizar su formato.
>
> add_fallback(*fallback*)
>
> > Agrega *fallback* como el objeto de respaldo para el objeto de traducción actual. Un objeto de traducción debe consultar la reserva si no puede proporcionar una traducción para un mensaje dado.
>
> gettext(*message*)
>
> > Si se ha establecido una reserva, reenvíe **`gettext()`** a la reserva. De lo contrario, retorna *message*. Anulado en clases derivadas.
>
> ngettext(*singular*, *plural*, *n*)
>
> > Si se ha establecido una reserva, reenvíe **`ngettext()`** a la reserva. De lo contrario, retorna *singular* si *n* es 1;

volver *plural* de lo contrario. Anulado en clases derivadas.

pgettext(*context*, *message*)

Si se ha establecido una reserva, reenvía **`pgettext()`** a la reserva. De lo contrario, retorna el mensaje traducido. Anulado en clases derivadas.

Added in version 3.8.

npgettext(*context*, *singular*, *plural*, *n*)

Si se ha establecido una reserva, reenvía **`npgettext()`** a la reserva. De lo contrario, retorna el mensaje traducido. Anulado en clases derivadas.

Added in version 3.8.

info()

Return a dictionary containing the metadata found in the message catalog file.

charset()

Retorna la codificación del archivo de catálogo de mensajes.

install(*names=None*)

Este método instala **`gettext()`** en el espacio de nombres incorporado, vinculándolo a `_`.

Si se proporciona el parámetro *names*, debe ser una secuencia que contenga los nombres de las funciones que desea instalar en el espacio de nombres incorporado, además de **`_()`**. Los nombres compatibles son `'gettext'`, `'ngettext'`, `'pgettext'` y `'npgettext'`.

Considere que esta es solo una manera, aunque la más conveniente, de hacer disponible la función **`_()`** en su aplicación. Ya que afecta a toda la aplicación de manera global, y en particular al espacio de nombres integrado, los módulos localizados nunca deben instalar `_()`. En lugar de ello, deberían utilizar este código para hacer que `_()` esté disponible en su módulo:

```
import gettext
t = gettext.translation('mymodule', ...)
_ = t.gettext
```

Esto pone `_()` solo en el espacio de nombres global del módulo y, por lo tanto, solo afecta las llamadas dentro de este módulo.

Distinto en la versión 3.8: Se agregó `'pgettext'` y `'npgettext'`.

La clase `GNUTranslations`

The **`gettext`** module provides one additional class derived from **`NullTranslations`**: **`GNUTranslations`**. This class overrides **`_parse()`** to enable reading GNU **gettext** format `.mo` files in both big-endian and little-endian format.

`GNUTranslations` parses optional metadata out of the translation catalog. It is convention with GNU **gettext** to include metadata as the translation for the empty string. This metadata is in RFC 822 [https://datatracker.ietf.org/doc/html/rfc822.html]-style `key: value` pairs, and should contain the `Project-Id-Version` key. If the key `Content-Type` is found, then the `charset` property is used to initialize the «protected» **`_charset`** instance variable, defaulting to `None` if not found. If the charset encoding is specified, then all message ids and message strings read from the catalog are converted to Unicode using this encoding, else ASCII is assumed.

Ya que los identificadores de mensajes también se leen como cadenas Unicode, todos los métodos `*gettext()` considerarán a los identificadores de mensajes como cadenas Unicode, y no como cadenas de bytes.

The entire set of key/value pairs are placed into a dictionary and set as the «protected» **`_info`** instance variable.

Si el número mágico del archivo `.mo` no es válido, el número de versión principal es inesperado, o si se producen otros problemas al leer el archivo, se crea una instancia de **`GNUTranslations`** puede generar **`OSError`**.

class gettext.GNUTranslations

Los siguientes métodos se anulan desde la implementación de la clase base:

gettext(*message*)

Busca el *message* id en el catálogo y retorna la cadena de caracteres de mensaje correspondiente, como una cadena Unicode. Si no hay una entrada en el catálogo para el *message* id, y se ha establecido una retroceso (*fallback*), la búsqueda se reenvía al método de retroceso **`gettext()`**. De lo contrario, se retorna el *message* id.

ngettext(*singular*, *plural*, *n*)

Realiza una búsqueda de formas plurales de una identificación de mensaje. *singular* se usa como la identificación del mensaje para fines de búsqueda en el catálogo, mientras que *n* se usa para determinar qué forma plural usar. La cadena del mensaje retornado es una cadena de caracteres Unicode.

Si la identificación del mensaje no se encuentra en el catálogo y se especifica una reserva, la solicitud se reenvía al método de reserva **`ngettext()`**. De lo contrario, cuando *n* es 1 se retorna *singular*, y *plural* se retorna en todos los demás casos.

Aquí hay un ejemplo:

```
n = len(os.listdir('.'))
cat = GNUTranslations(somefile)
message = cat.ngettext(
    'There is %(num)d file in this directory',
    'There are %(num)d files in this directory'
    n) % {'num': n}
```

pgettext(*context*, *message*)

Busca el *context* y *message* id en el catálogo y retorna la cadena de de caracteres mensaje correspondiente, como una cadena de caracteres Unicode. Si no hay ninguna entrada en el catálogo para el *message* id y *context*, y se ha establecido un retroceso (*fallback*), la búsqueda se reenvía al método de retroceso `pgettext()`. De lo contrario, se retorna el *message* id.

Added in version 3.8.

npgettext(*context*, *singular*, *plural*, *n*)

Realiza una búsqueda de formas plurales de una identificación de mensaje. *singular* se usa como la identificación del mensaje para fines de búsqueda en el catálogo, mientras que *n* se usa para determinar qué forma plural usar.

Si la identificación del mensaje para *context* no se encuentra en el catálogo y se especifica una reserva, la solicitud se reenvía al método de reserva `npgettext()`. De lo contrario, cuando *n * es 1 se retorna *singular*, y *plural* se retorna en todos los demás casos.

Added in version 3.8.

Soporte de catálogo de mensajes de Solaris

El sistema operativo Solaris define su propio formato de archivo binario `.mo`, pero como no se puede encontrar documentación sobre este formato, no es compatible en este momento.

El constructor del catálogo

GNOME uses a version of the **`gettext`** module by James

Henstridge, but this version has a slightly different API. Its documented usage was:

```
import gettext
cat = gettext.Catalog(domain, localedir)
_ = cat.gettext
print(_('hello world'))
```

Para la compatibilidad con este módulo anterior, la función **`Catalog()`** es un alias para la función `translation()` descrita anteriormente.

Una diferencia entre este módulo y el de Henstridge: sus objetos de catálogo admitían el acceso a través de una API de mapeo, pero esto parece estar sin usar y, por lo tanto, actualmente no es compatible.

Internacionalizando sus programas y módulos

La internacionalización (I18N) se refiere a la operación mediante la cual un programa conoce varios idiomas. La localización (L10N) se refiere a la adaptación de su programa, una vez internacionalizado, al idioma local y los hábitos culturales. Para proporcionar mensajes multilingües para sus programas de Python, debe seguir los siguientes pasos:

1. prepare su programa o módulo marcando especialmente cadenas traducibles
2. ejecuta un conjunto de herramientas sobre tus archivos marcados para generar catálogos de mensajes sin procesar
3. crear traducciones específicas del idioma de los catálogos de mensajes
4. use the **`gettext`** module so that message strings are properly translated

Para preparar su código para la internacionalización (I18N), necesita revisar todas las cadenas de texto en sus archivos. Cualquier cadena que requiera traducción debe ser marcada envolviéndola en `_('...')` —, lo que equivale a una llamada a la función `_`. Por ejemplo:

```
filename = 'mylog.txt'
message = _('writing a log message')
with open(filename, 'w') as fp:
    fp.write(message)
```

En este ejemplo, la cadena de caracteres `'writing a log message'` está marcada como candidata para la traducción, mientras que las cadenas `'mylog.txt'` y `'w'` no lo están.

Hay algunas herramientas para extraer las cadenas destinadas a la traducción. El GNU **gettext** original solo admitía el código fuente C o C++, pero su versión extendida **xgettext** escanea el código escrito en varios idiomas, incluido Python, para encontrar cadenas marcadas como traducibles. Babel [https://babel.pocoo.org/] es una biblioteca de internacionalización de Python que incluye un script `pybabel` para extraer y compilar catálogos de mensajes. El programa de François Pinard llamado **xpot** hace un trabajo similar y está disponible como parte de su po-utils package [https://github.com/pinard/po-utils].

(Python también incluye versiones de Python puro de estos programas, llamadas **pygettext.py** y **msgfmt.py**; algunas distribuciones de Python las instalarán por usted. **pygettext.py** es similar a **xgettext**, pero solo entiende el código fuente de Python y no puede manejar otros lenguajes de programación como C o C++. **pygettext.py** admite una interfaz de línea de comandos similar a **xgettext**; para detalles sobre su uso, ejecute `pygettext.py --help`. **msgfmt.py** es binario compatible con GNU **msgfmt**. Con estos dos programas, es posible que no necesite el paquete GNU **gettext** para internacionalizar sus aplicaciones Python.)

xgettext, **pygettext**, y herramientas similares generan archivos `.po` que son catálogos de mensajes. Son archivos estructurados legibles por humanos que contienen cada cadena marcada en el código fuente, junto con un marcador de posición para las versiones traducidas de estas cadenas.

Copies of these `.po` files are then handed over to the individual human translators who write translations for every supported natural language. They send back the completed language-specific versions as a `<language-name>.po` file that's compiled into a machine-readable `.mo` binary catalog file using the **msgfmt** program. The `.mo` files are used by the **`gettext`** module for the actual translation processing at run-time.

How you use the **`gettext`** module in your code depends on whether you are internationalizing a single module or your entire application. The next two sections will discuss each case.

Agregar configuración regional a su módulo

Si está aplicando configuración regional a su módulo, debe tener cuidado de no realizar cambios globales, por ejemplo, al espacio de nombres integrado. No debe utilizar la API GNU **gettext**, sino la API basada en clases.

Digamos que su módulo se llama «spam» y los diversos archivos `.mo` de traducciones del lenguaje natural del módulo residen en `/usr/share/locale` en formato GNU **gettext**. Esto es lo que pondría en la parte superior de su módulo:

```
import gettext
t = gettext.translation('spam', '/usr/share/locale')
_ = t.gettext
```

Agregar configuración regional a su aplicación

Si está localizando su aplicación, puede instalar globalmente la función **`_()`** en el espacio de nombres integrado, normalmente en el archivo principal de su aplicación. Esto hará que todos los archivos específicos de la aplicación puedan usar `_('...')` sin necesidad de instalarla explícitamente en cada uno de ellos.

En el caso simple, entonces, solo necesita agregar el siguiente bit de código al archivo del controlador principal de su aplicación:

```
import gettext
gettext.install('myapplication')
```

Si necesita establecer el directorio de configuración regional, puede pasarlo a la función **`install()`**:

```
import gettext
gettext.install('myapplication', '/usr/share/locale')
```

Cambiar idiomas sobre la marcha

Si su programa necesita admitir muchos idiomas al mismo tiempo, es posible que desee crear varias instancias de traducción y luego cambiar entre ellas explícitamente, de esta manera:

```
import gettext

lang1 = gettext.translation('myapplication', languages=[
lang2 = gettext.translation('myapplication', languages=[
lang3 = gettext.translation('myapplication', languages=[

# start by using language1
lang1.install()

# ... time goes by, user selects language 2
lang2.install()

# ... more time goes by, user selects language 3
lang3.install()
```

Traducciones diferidas

En la mayoría de las situaciones de codificación, las cadenas se traducen donde se codifican. Sin embargo, ocasionalmente, debe marcar cadenas para la traducción, pero diferir la traducción real hasta más tarde. Un ejemplo clásico es:

```
animals = ['mollusk',
           'albatross',
           'rat',
           'penguin',
           'python', ]
# ...
for a in animals:
    print(a)
```

Aquí, desea marcar las cadenas en la lista de `animals` como traducibles, pero en realidad no desea traducirlas hasta que se impriman.

Aquí hay una manera de manejar esta situación:

```
def _(message): return message

animals = [_('mollusk'),
           _('albatross'),
           _('rat'),
           _('penguin'),
           _('python'), ]

del _

# ...
for a in animals:
    print(_(a))
```

Esto funciona ya que la definición ficticia de `_()` devuelve simplemente la cadena sin modificar. Además, esta definición ficticia sobrescribirá de manera temporal cualquier definición de `_()` en el espacio de nombres integrado (hasta el uso del comando `del`). No obstante, tenga cuidado si ya existe una definición previa de `_()` en el espacio de nombres local.

Tenga en cuenta que el segundo uso de `_()` no identificará «a» como traducible al programa **gettext**, porque el parámetro no es un literal de cadena.

Otra forma de manejar esto es con el siguiente ejemplo:

```
def N_(message): return message

animals = [N_('mollusk'),
           N_('albatross'),
           N_('rat'),
           N_('penguin'),
           N_('python'), ]
```

```
# ...
for a in animals:
    print(_(a))
```

En este caso, está marcando las cadenas traducibles con la función **`N_()`**, la cual no generará conflictos con ninguna definición de **`_()`**. Sin embargo, deberá enseñar a su programa de extracción de mensajes a buscar cadenas traducibles marcadas con **`N_()`**. **xgettext**, **pygettext**, `pybabel extract`, y **xpot** todos soportan esto mediante el uso de la opción de línea de comandos **`-k`**. La elección de **`` N_`() ``** aquí es completamente arbitraria; podría haber sido también **`MarkThisStringForTranslation()`**.

Agradecimientos

Las siguientes personas contribuyeron con código, comentarios, sugerencias de diseño, implementaciones anteriores y una valiosa experiencia para la creación de este módulo:

- *Peter Funk*
- *James Henstridge*
- *Juan David Ibáñez Palomar*
- *Marc-André Lemburg*
- Martin von Löwis
- *François Pinard*
- *Barry Warsaw*
- *Gustavo Niemeyer*

Notas al pie

The default locale directory is system dependent; for example, on Red Hat Linux it is `/usr/share/locale`, but on Solaris it is `/usr/lib/locale`. The **`gettext`** module does not try to support these system dependent defaults; instead its default is `sys.base_prefix/share/locale` (see **`sys.base_prefix`**). For this reason, it is always best to call **`bindtextdomain()`** with an explicit absolute path at the start of your application.

Vea la nota al pie de página para `bindtextdomain()` arriba.

[1] [2] [3]

`locale` — Internationalization services

Source code: Lib/locale.py [https://github.com/python/cpython/tree/3.14/Lib/locale.py]

The `locale` module opens access to the POSIX locale database and functionality. The POSIX locale mechanism allows programmers to deal with certain cultural issues in an application, without requiring the programmer to know all the specifics of each country where the software is executed.

The `locale` module is implemented on top of the `_locale` module, which in turn uses an ANSI C locale implementation if available.

The `locale` module defines the following exception and functions:

exception locale.Error

> Cuando el *locale* pasado a `setlocale()` no es reconocido, se lanza una excepción.

locale.setlocale(*category*, *locale=None*)

> If *locale* is given and not `None`, `setlocale()` modifies the locale setting for the *category*. The available categories are listed in the data description below. *locale* may be a string, or a pair, language code and encoding. An empty string specifies the user’s default settings. If the modification of the locale fails, the exception `Error` is raised. If successful, the new locale setting is returned.
>
> If *locale* is a pair, it is converted to a locale name using the locale aliasing engine. The language code has the same format as a locale name, but without encoding and `@`-

modifier. The language code and encoding can be `None`.

Si se omite *locale* o es `None`, se retorna la configuración actual para *category*.

Ejemplo:

```
>>> import locale
>>> loc = locale.setlocale(locale.LC_ALL)  # get cu
# use German locale; name and availability varies w
>>> locale.setlocale(locale.LC_ALL, 'de_DE.UTF-8')
>>> locale.strcoll('f\xe4n', 'foo')  # compare a st
>>> locale.setlocale(locale.LC_ALL, '')   # use use
>>> locale.setlocale(locale.LC_ALL, 'C')  # use def
>>> locale.setlocale(locale.LC_ALL, loc)  # restore
```

`setlocale()` is not thread-safe on most systems. Applications typically start with a call of:

```
import locale
locale.setlocale(locale.LC_ALL, '')
```

Esto establece la configuración regional de todas las categorías en la configuración predeterminada del usuario (normalmente especificada en la variable de entorno **`LANG`**). Si la configuración regional no se cambia a partir de entonces, el uso de multiprocesamiento, no debería causar problemas.

locale.localeconv()

Retorna la base de datos de las convenciones locales como diccionario. Este diccionario tiene las siguientes cadenas de caracteres como claves:

Categoría	**Clave**	**Significado**
`LC_NUMERIC`	`'decimal_point'`	Carácter de punto decimal.
	`'grouping'`	Secuencia de números que especifica qué posiciones relativas se espera el `'miles_sep'`. Si la secuencia termina con **`CHAR_MAX`**, no se realiza ninguna agrupación adicional. Si la secuencia termina con un `0`, el último

		tamaño de grupo se usa repetidamente.
	`'thousands_sep'`	Carácter utilizado entre grupos.
LC_MONETARY	`'int_curr_symbol'`	Símbolo de moneda internacional.
	`'currency_symbol'`	Símbolo de moneda local.
	`'p_cs_precedes/n_cs_precedes'`	Si el símbolo de moneda precede al valor (para valores positivos o negativos).
	`'p_sep_by_space/n_sep_by_space'`	Si el símbolo de moneda está separado del valor por un espacio (para valores positivos o negativos).
	`'mon_decimal_point'`	Punto decimal utilizado para valores monetarios.
	`'frac_digits'`	Número de dígitos fraccionarios utilizados en el formateo local de valores monetarios.
	`'int_frac_digits'`	Número de dígitos fraccionarios utilizados en el formateo internacional de valores monetarios.
	`'mon_thousands_sep'`	Separador de grupo utilizado para valores monetarios.
	`'mon_grouping'`	Equivalente a `'grouping'`, utilizada para valores monetarios.
	`'positive_sign'`	Símbolo utilizado para anotar un valor monetario positivo.
	`'negative_sign'`	Símbolo utilizado para anotar un valor monetario negativo.
	`'p_sign_posn/n_sign_posn'`	La posición del signo (para respuestas positivas. valores negativos), ver abajo.

Todos los valores numéricos se pueden establecer a **CHAR_MAX** para indicar que no hay ningún valor especificado en esta configuración regional.

Los valores posibles para `'p_sign_posn'` y `'n_sign_posn'` se dan a continuación.

Valor	Explicación
`0`	Moneda y valor están rodeados por paréntesis.
`1`	El signo debe preceder al valor y al símbolo de moneda.
`2`	El signo debe seguir el valor y el símbolo de moneda.
`3`	El signo debe preceder inmediatamente al valor.
`4`	El signo debe seguir inmediatamente al valor.
`CHAR_MAX`	No se especifica nada en esta configuración regional.

The function temporarily sets the `LC_CTYPE` locale to the `LC_NUMERIC` locale or the `LC_MONETARY` locale if locales are different and numeric or monetary strings are non-ASCII. This temporary change affects other threads.

Distinto en la versión 3.7: The function now temporarily sets the `LC_CTYPE` locale to the `LC_NUMERIC` locale in some cases.

locale.nl_langinfo(*option*)

Retorna información específica de la configuración regional como una cadena de caracteres. Esta función no está disponible en todos los sistemas, y el conjunto de opciones posibles también puede variar entre plataformas. Los posibles valores de argumento son números, para los cuales las constantes simbólicas están disponibles en el módulo de configuración regional.

La función **`nl_langinfo()`** acepta una de las siguientes claves. La mayoría de las descripciones están tomadas de la descripción correspondiente en la biblioteca C de GNU.

locale.CODESET

Obtiene una cadena de caracteres con el nombre de la codificación de caracteres utilizada en la configuración regional seleccionada.

locale.D_T_FMT

Obtiene una cadena de caracteres que se puede utilizar como cadena de caracteres de formato para **`time.strftime()`** para representar la fecha y la hora de una manera específica de la configuración regional.

locale.D_FMT

Obtiene una cadena de caracteres que se puede utilizar como cadena de formato para **`time.strftime()`** para representar una fecha de una manera específica de la configuración regional.

locale.T_FMT

Obtiene una cadena de caracteres que se puede utilizar como cadena de formato para **`time.strftime()`** para representar un tiempo de una manera específica de la configuración regional.

locale.T_FMT_AMPM

Obtiene una cadena de caracteres de formato para `time.strftime()` para representar el tiempo en el formato am/pm.

locale.DAY_1
locale.DAY_2
locale.DAY_3
locale.DAY_4
locale.DAY_5
locale.DAY_6
locale.DAY_7

Consigue el nombre del n-ésimo día de la semana.

Nota

Esto sigue la convención estadounidense de `DAY_1` siendo domingo, no la convención internacional (ISO 8601) que el lunes es el primer día de la semana.

locale.ABDAY_1
locale.ABDAY_2
locale.ABDAY_3
locale.ABDAY_4
locale.ABDAY_5
locale.ABDAY_6
locale.ABDAY_7

Obtener el nombre abreviado del n-ésimo día de la semana.

locale.MON_1
locale.MON_2
locale.MON_3
locale.MON_4
locale.MON_5
locale.MON_6

locale.MON_7
locale.MON_8
locale.MON_9
locale.MON_10
locale.MON_11
locale.MON_12

Obtener el nombre del n-ésimo mes.

locale.ABMON_1
locale.ABMON_2
locale.ABMON_3
locale.ABMON_4
locale.ABMON_5
locale.ABMON_6
locale.ABMON_7
locale.ABMON_8
locale.ABMON_9
locale.ABMON_10
locale.ABMON_11
locale.ABMON_12

Obtener el nombre abreviado del enésimo mes.

locale.RADIXCHAR

Obtener el carácter radical (punto decimal, coma decimal, etc.).

locale.THOUSEP

Obtener el carácter separador de miles (grupos de tres dígitos).

locale.YESEXPR

Obtener una expresión regular que se pueda usar con la función regex para reconocer una respuesta positiva a una pregunta de sí / no.

locale.NOEXPR

Get a regular expression that can be used with the `regex(3)` function to recognize a negative response to a yes/no question.

Nota

The regular expressions for `YESEXPR` and `NOEXPR` use syntax suitable for the `regex` function from the C library, which might differ from the syntax used in `re`.

locale.CRNCYSTR

Obtener el símbolo de moneda, precedido por «-» si el símbolo debe aparecer antes del valor, «+» si el símbolo debe aparecer después del valor, o «.» si el símbolo debe reemplazar el carácter raíz.

locale.ERA

Get a string which describes how years are counted and displayed for each era in a locale.

La mayoría de las configuraciones regionales no definen este valor. Un ejemplo de una localidad que sí define este valor es la japonesa. En Japón, la representación tradicional de fechas incluye el nombre de la época correspondiente al reinado del entonces emperador.

Normally it should not be necessary to use this value directly. Specifying the `E` modifier in their format strings causes the `time.strftime()` function to use this information. The format of the returned string is specified in *The Open Group Base Specifications Issue 8*, paragraph 7.3.5.2 LC_TIME C-Language Access [https://pubs.opengroup.org/onlinepubs/9799919799/basedefs/V1_chap07.html#tag_07_03_05_02].

locale.ERA_D_T_FMT

Obtener una cadena de caracteres de formato para `time.strftime()` para representar la fecha y la hora de una manera específica de la era.

locale.ERA_D_FMT

Obtener una cadena de caracteres de formato para **`time.strftime()`** para representar una fecha de una manera basada en una era específica.

locale.ERA_T_FMT

Obtener una cadena de caracteres de formato para **`time.strftime()`** para representar una hora de una manera basada en una era específica.

locale.ALT_DIGITS

Get a string consisting of up to 100 semicolon-separated symbols used to represent the values 0 to 99 in a locale-specific way. In most locales this is an empty string.

The function temporarily sets the `LC_CTYPE` locale to the locale of the category that determines the requested value (`LC_TIME`, `LC_NUMERIC`, `LC_MONETARY` or `LC_MESSAGES`) if locales are different and the resulting string is non-ASCII. This temporary change affects other threads.

Distinto en la versión 3.14: The function now temporarily sets the `LC_CTYPE` locale in some cases.

locale.getdefaultlocale([*envvars*])

Intenta determinar la configuración regional por defecto y los retorna como una tupla del formulario `(código de idioma, codificación)`.

De acuerdo con POSIX, un programa que no ha llamado a `setlocale(LC_ALL, '')` se ejecuta utilizando la configuración regional portátil `'C'`. Llamar a `setlocale(LC_ALL, ' ')` le permite usar la configuración regional predeterminada definida por la variable **`LANG`**. Dado que no queremos interferir con la configuración de configuración regional actual, emulamos el comportamiento de la manera descrita anteriormente.

Para mantener la compatibilidad con otras plataformas, no sólo se prueba la variable **LANG** sino una lista de variables dadas como parámetro *envvars*. Se utilizará la primera que se encuentre definida. *envvars* por defecto a la ruta de búsqueda utilizada en GNU gettext; siempre debe contener el nombre de la variable `'LANG '`. La ruta de búsqueda GNU gettext contiene `'LC_ALL '`, `'LC_CTYPE '`, `'LANG '` y `'LANGUAGE '`, en ese orden.

The language code has the same format as a locale name, but without encoding and `@`-modifier. The language code and encoding may be `None` if their values cannot be determined. The «C» locale is represented as `(None, None)`.

Deprecated since version 3.11, will be removed in version 3.15.

locale.getlocale(*category=LC_CTYPE*)

Returns the current setting for the given locale category as a tuple containing the language code and encoding. *category* may be one of the **`LC_*`** values except **`LC_ALL`**. It defaults to **`LC_CTYPE`**.

The language code has the same format as a locale name, but without encoding and `@`-modifier. The language code and encoding may be `None` if their values cannot be determined. The «C» locale is represented as `(None, None)`.

locale.getpreferredencoding(*do_setlocale=True*)

Retorna la locale encoding utilizada para los datos de texto, según las preferencias del usuario. Las preferencias del usuario se expresan de manera diferente en diferentes sistemas, y puede que no estén disponibles programáticamente en algunos sistemas, por lo que esta función solo retorna una suposición.

En algunos sistemas, es necesario invocar **`setlocale()`** para obtener las preferencias del usuario, por lo que esta función no es segura para subprocesos. Si invocar *setlocale* no es necesario o deseado, *do_setlocale* se debe ajustar a `False`.

En Android o si Python UTF-8 Mode está habilitado, siempre retorna `'utf-8'`, el argumento locale encoding y *do_setlocale* se ignoran.

La pre-inicializacioń de Python configura la configuración regional LC_CTYPE. Véase también la codificación del sistema de archivos y manejador de errores.

Distinto en la versión 3.7: La función ahora siempre retorna `"utf-8"` en Android o si Python UTF-8 Mode está habilitado.

locale.getencoding()

Obtenga el locale encoding actual:

- En Android y VxWorks, retorna `"utf-8"`.
- En Unix, retorna la codificación de la configuración regional **`LC_CTYPE`** actual. Retorna `"utf-8"` si `nl_langinfo(CODESET)` retorna una cadena vacía: por ejemplo, si no se admite la configuración regional actual de LC_CTYPE.
- En Windows, retorna la página de códigos ANSI.

La pre-inicializacioń de Python configura la configuración regional LC_CTYPE. Véase también la codificación del sistema de archivos y manejador de errores.

Esta función es similar a **`getpreferredencoding(False)`** excepto que esta función ignora Python UTF-8 Mode.

Added in version 3.11.

locale.normalize(*localename*)

Retorna un código de configuración regional normalizado para el nombre configuración regional dado. El código de configuración regional retornado está formateado para usarse con **`setlocale()`**. Si la normalización falla, el nombre original se retorna sin cambios.

Si no se conoce la codificación dada, la función por defecto codifica el código de configuración regional como

`setlocale()`.

locale.strcoll(*string1*, *string2*)

Compara dos cadenas según la configuración actual **`LC_COLLATE`** . Como cualquier otra función de comparación, retorna un valor negativo o positivo, o `0`, dependiendo de si *string1* compila antes o después de *string2* o es igual a él.

locale.strxfrm(*string*)

Transforma una cadena en una que se puede utilizar en comparaciones de localización. Por ejemplo, `strxfrm(s1) < strxfrm(s2)` es equivalente a `strcoll(s1, s2) < 0`. Esta función se puede utilizar cuando la misma cadena se compara repetidamente, p. ej., al agrupar una secuencia de cadenas.

locale.format_string(*format*, *val*, *grouping=False*, *monetary=False*)

Formats a number *val* according to the current **`LC_NUMERIC`** setting. The format follows the conventions of the `%` operator. For floating-point values, the decimal point is modified if appropriate. If *grouping* is `True`, also takes the grouping into account.

Si *monetary* es verdadero, la conversión utiliza separador monetario de miles y cadenas de caracteres de agrupación.

Procesa especificadores de formato como en `format % val`, pero tiene en cuenta los ajustes de configuración regional actuales.

Distinto en la versión 3.7: Se añadió el parámetro de la palabra clave *monetary*.

locale.currency(*val*, *symbol=True*, *grouping=False*, *international=False*)

Formatea un número *val* según la configuración actual **`LC_MONETARY`** .

The returned string includes the currency symbol if *symbol* is true, which is the default. If *grouping* is `True` (which is not the default), grouping is done with the value. If *international* is `True` (which is not the default), the international currency symbol is used.

Nota

This function will not work with the “C” locale, so you have to set a locale via **`setlocale()`** first.

locale.str(*float*)

Formats a floating-point number using the same format as the built-in function `str(float)`, but takes the decimal point into account.

locale.delocalize(*string*)

Convierte una cadena de caracteres en una cadena de números normalizada, siguiendo la configuración **`LC_NUMERIC`**.

Added in version 3.5.

locale.localize(*string*, *grouping=False*, *monetary=False*)

Convierte una cadena de números normalizada en una cadena de caracteres formateada siguiendo la configuración **`LC_NUMERIC`**.

Added in version 3.10.

locale.atof(*string*, *func=float*)

Convierte una cadena en un número, siguiendo la configuración de **`LC_NUMERIC`**, llamando a *func* según el resultado de llamar a **`delocalize()`** en *string*.

locale.atoi(*string*)

Convierte una cadena de caracteres a un entero, siguiendo las convenciones `LC_NUMERIC` .

locale.LC_CTYPE

Locale category for the character type functions. Most importantly, this category defines the text encoding, i.e. how bytes are interpreted as Unicode codepoints. See **PEP 538** [https://peps.python.org/pep-0538/] and **PEP 540** [https://peps.python.org/pep-0540/] for how this variable might be automatically coerced to `C.UTF-8` to avoid issues created by invalid settings in containers or incompatible settings passed over remote SSH connections.

Python doesn't internally use locale-dependent character transformation functions from `ctype.h`. Instead, `pyctype.h` provides locale-independent equivalents like `Py_TOLOWER`.

locale.LC_COLLATE

Locale category for sorting strings. The functions `strcoll()` and `strxfrm()` of the `locale` module are affected.

locale.LC_TIME

Categoría de configuración regional para el formateo de hora. La función `time.strftime()` sigue estas convenciones.

locale.LC_MONETARY

Categoría de configuración regional para el formateo de valores monetarios. Las opciones disponibles están disponibles en la función `localeconv()` .

locale.LC_MESSAGES

Categoría de configuración regional para visualización de mensajes. Python actualmente no admite mensajes de configuración regional específicos de la aplicación. Los mensajes mostrados por el sistema operativo, como los retornados por `os.strerror()` podrían verse afectados por esta categoría.

This value may not be available on operating systems not conforming to the POSIX standard, most notably Windows.

locale.LC_NUMERIC

Locale category for formatting numbers. The functions `format_string()`, `atoi()`, `atof()` and `str()` of the **`locale`** module are affected by that category. All other numeric formatting operations are not affected.

locale.LC_ALL

Combinación de todos los ajustes de configuración regional. Si se utiliza este indicador cuando se cambia la localización, se intenta establecer la localización para todas las categorías. Si eso falla para cualquier categoría, ninguna categoría se cambia en absoluto. Cuando la configuración regional se recupera usando este indicador, se retorna una cadena de caracteres que indica la configuración para todas las categorías. Esta cadena de caracteres se puede usar más tarde para restaurar la configuración.

locale.CHAR_MAX

Esta es una constante simbólica utilizada para diferentes valores retornados por `localeconv()`.

Segundo plano, detalles, indicaciones, consejos y advertencias

El estándar C define la configuración regional como una propiedad de todo el programa que puede ser relativamente costosa de cambiar. Además de eso, alguna implementación se rompe de tal manera que los cambios de configuración local frecuentes pueden causar volcados de núcleo. Esto hace que la configuración regional sea algo dolorosa de usar correctamente.

Inicialmente, cuando se inicia un programa, la configuración regional es la configuración regional `C`, no importa cuál sea la configuración regional preferida por el usuario. Hay una excepción: la categoría **`LC_CTYPE`** se cambia al inicio para establecer la codificación de la configuración regional actual en la codificación de configuración regional preferida del usuario. El programa debe decir explícitamente que quiere la configuración regional preferida del usuario para otras categorías llamando a `setlocale(LC_ALL, '')`.

Generalmente es una mala idea llamar **`setlocale()`** en alguna rutina de biblioteca, ya que como efecto secundario afecta a todo el programa. Guardar y restaurar es casi igual de malo: es caro y afecta a otros hilos que se ejecutan antes de que los ajustes se hayan restaurado.

Si, al codificar un módulo para uso general, se necesita una versión configuración regional independiente de una operación que se ve afectada por la localización (como ciertos formatos utilizados con **`time.strftime()`**), tendrá que encontrar una manera de hacerlo sin usar la rutina de biblioteca estándar. Aún mejor es convencerse a sí mismo de que el uso de la configuración regional está bien. Sólo como último recurso debe documentar que su módulo no es compatible con la configuración regional no- `C`.

The only way to perform numeric operations according to the locale is to use the special functions defined by this module: **`atof()`**, **`atoi()`**, **`format_string()`**, **`str()`**.

No hay manera de realizar conversiones de casos y clasificaciones de caracteres de acuerdo a la configuración regional. Para cadenas de texto (Unicode) estas se hacen de acuerdo con el valor de carácter solamente, mientras que para cadenas de bytes, las conversiones y clasificaciones se hacen de acuerdo con el valor ASCII del byte, y bytes cuyo bit alto se establece (es decir, bytes no ASCII) nunca se convierten o se consideran parte de una clase de caracteres como letra o espacio en blanco.

Locale names

The format of the locale name is platform dependent, and the set of supported locales can depend on the system configuration.

On Posix platforms, it usually has the format [1]:

```
language ["_" territory] ["." charset] ["@" modifier]
```

where *language* is a two- or three-letter language code from ISO 639 [https://www.iso.org/iso-639-language-code], *territory* is a two-letter country or region code from ISO 3166 [https://www.iso.org/iso-3166-country-codes.html], *charset* is a locale encoding, and *modifier* is a script name, a language subtag, a sort order identifier, or other locale modifier (for example, «latin», «valencia», «stroke» and «euro»).

On Windows, several formats are supported. [2] [3] A subset of IETF BCP 47 [https://www.rfc-editor.org/info/bcp47] tags:

```
language ["-" script] ["-" territory] ["." charset]
language ["-" script] "-" territory "-" modifier
```

where *language* and *territory* have the same meaning as in Posix, *script* is a four-letter script code from ISO 15924 [https://www.unicode.org/iso15924/], and *modifier* is a language subtag, a sort order identifier or custom modifier (for example, «valencia», «stroke» or «x-python»). Both hyphen (`'-'`) and underscore (`'_'`) separators are supported. Only UTF-8 encoding is allowed for BCP 47 tags.

Windows also supports locale names in the format:

```
language ["_" territory] ["." charset]
```

where *language* and *territory* are full names, such as «English» and «United States», and *charset* is either a code page number (for example, «1252») or UTF-8. Only the underscore separator is supported in this format.

The «C» locale is supported on all platforms.

IEEE Std 1003.1-2024; 8.2 Internationalization Variables [https://pubs.opengroup.org/onlinepubs/9799919799/basedefs/V1_chap08.html#tag_08_02]

UCRT Locale names, Languages, and Country/Region strings [https://learn.microsoft.com/en-us/cpp/c-runtime-library/locale-names-languages-and-country-region-strings]

Locale Names [https://learn.microsoft.com/en-us/windows/win32/intl/locale-names]

Para escritores de extensión y programas que incrustan Python

Los módulos de extensión nunca deben llamar a `setlocale()`, excepto para averiguar cuál es la configuración regional actual. Pero dado que el valor de retorno sólo se puede utilizar portablemente para restaurarlo, eso no es muy útil (excepto quizás para averiguar si la localización es o no es `C`).

When Python code uses the **`locale`** module to change the locale, this also affects the embedding application. If the embedding application doesn't want this to happen, it should remove the **`_locale`** extension module (which does all the work) from the table of built-in modules in the `config.c` file, and make sure that the **`_locale`** module is not accessible as a shared library.

Acceso a los catálogos de mensajes

locale.gettext(*msg*)

locale.dgettext(*domain*, *msg*)

locale.dcgettext(*domain*, *msg*, *category*)

locale.textdomain(*domain*)

locale.bindtextdomain(*domain*, *dir*)

locale.bind_textdomain_codeset(*domain*, *codeset*)

The locale module exposes the C library's gettext interface on systems that provide this interface. It consists of the functions **`gettext()`**, **`dgettext()`**, **`dcgettext()`**, **`textdomain()`**, **`bindtextdomain()`**, and **`bind_textdomain_codeset()`**. These are similar to the same functions in the **`gettext`** module, but use the C library's binary format for message catalogs, and the C library's search algorithms for locating message catalogs.

Python applications should normally find no need to invoke these functions, and should use **`gettext`** instead. A known exception to this rule are applications that link with additional C libraries which internally invoke C functions `gettext` or `dcgettext`. For these applications, it may be necessary to bind the text domain, so that the libraries can properly locate their message catalogs.

Graphical user interfaces with Tk

Tk/Tcl has long been an integral part of Python. It provides a robust and platform independent windowing toolkit, that is available to Python programmers using the **`tkinter`** package, and its extension, the **`tkinter.ttk`** module.

El paquete **`tkinter`** es una fina capa orientada a objetos encima de Tcl/Tk. Para usar **`tkinter`**, no necesita escribir código Tcl, pero deberá consultar la documentación de Tk y, ocasionalmente, la documentación de Tcl. **`tkinter`** es un conjunto de envoltorios que implementan los widgets Tk como clases de Python.

Las principales virtudes de **`tkinter`** son que es rápido y que generalmente viene incluido con Python. Aunque su documentación estándar es débil, se dispone de buen material, que incluye: referencias, tutoriales, un libro y otros. **`tkinter`** también es famoso por tener un aspecto y una sensación obsoletos, que se ha mejorado enormemente en Tk 8.5. Sin embargo, hay muchas otras bibliotecas GUI en las que podría estar interesado. La wiki de Python enumera varias GUI frameworks and tools [https://wiki.python.org/moin/GuiProgramming] alternativas.

- **`tkinter`** — Python interface to Tcl/Tk
    - Arquitectura
    - Módulos Tkinter
    - Guía de supervivencia de Tkinter
        - Un programa simple de Hola Mundo
        - Conceptos importantes de Tk
        - Entendiendo como funcionan los empaquetadores de Tcl/Tk
        - ¿Cómo lo hago?, ¿Cómo funciona?
        - Navegando en el manual de referencia de Tcl/Tk

    - Modelo de subprocesamiento
    - Guía práctica
        - Configuración de opciones
        - El empaquetador
        - Opciones del empaquetador
        - Asociación de variables de widget
        - El gestor de ventanas
        - Tipos de datos de opciones Tk
        - Enlaces y eventos
        - El parámetro índice
        - Imágenes
    - Gestor de archivos
- **`tkinter.colorchooser`** — Color choosing dialog
- **`tkinter.font`** — Tkinter font wrapper
- Diálogos tkinter
    - **`tkinter.simpledialog`** — Standard Tkinter input dialogs
    - **`tkinter.filedialog`** — File selection dialogs
        - Diálogos nativos de carga/guardado
    - **`tkinter.commondialog`** — Dialog window templates
- **`tkinter.messagebox`** — Tkinter message prompts
- **`tkinter.scrolledtext`** — Scrolled Text Widget
- **`tkinter.dnd`** — Drag and drop support
- **`tkinter.ttk`** — Tk themed widgets
    - Uso de Ttk
    - Ttk widgets
    - Widget
        - Opciones estándar
        - Opciones de widgets desplegables
        - Opciones de etiqueta
        - Opciones de compatibilidad
        - Estados del widget

    - ttk.Widget
  - Combobox
    - Opciones
    - Eventos virtuales
    - ttk.Combobox
  - Spinbox
    - Opciones
    - Eventos virtuales
    - ttk.Spinbox
  - Notebook
    - Opciones
    - Opciones de pestañas
    - Identificadores de pestañas
    - Eventos virtuales
    - ttk.Notebook
  - Progressbar
    - Opciones
    - ttk.Progressbar
  - Separator
    - Opciones
  - Sizegrip
    - Notas específicas por plataforma
    - Errores detectados
  - Treeview
    - Opciones
    - Opciones de elementos
    - Opciones de etiqueta
    - Identificadores de columna

        - Eventos virtuales
        - ttk.Treeview
    - Ttk Styling
        - Diseños
- IDLE — Python editor and shell
    - Menús
        - Menú de archivo (Shell y Editor)
        - Menú editar (Shell y Editor)
        - Menú de formato (solo ventana del Editor)
        - Menú ejecutar (solo ventana Editor)
        - Menú de shell (solo ventana de shell)
        - Menú de depuración (solo ventana de shell)
        - Menú de opciones (Shell y editor)
        - Menú de ventana (shell y editor)
        - Menú de ayuda (shell y editor)
        - Menús contextuales
    - Edición y navegación
        - Ventana del editor
        - Atajos de teclado
        - Indentación automática
        - Buscar y reemplazar
        - Terminaciones
        - Sugerencias de llamada
        - Format block
        - Contexto del código
        - Ventana de Shell
        - Colores del texto
    - Inicio y ejecución de código
        - Command-line usage
        - Error de inicio
        - Ejecutando código del usuario
        - Salida del usuario en consola

        - Desarrollando aplicaciones tkinter
        - Ejecutando sin un subproceso
    - Ayuda y preferencias
        - Recursos de ayuda
        - Preferencias de configuración
        - IDLE en macOS
        - Extensiones
    - idlelib — implementation of IDLE application
- **`turtle`** — Turtle graphics
    - Introducción
    - Empezar
    - Tutorial
        - Iniciando un entorno para tortugas
        - Dibujo básico
            - Control del lápiz
            - La posición de la tortuga
        - Creando patrones algorítmicos
    - Cómo…
        - Empiece lo antes posible
        - Automatically begin and end filling
        - Utilice el espacio de nombres del módulo **`turtle`**
        - Utilice gráficos de tortugas en un guión
        - Utilice gráficos de tortuga orientados a objetos
    - Referencia gráficos de tortugas
        - Métodos Turtle
        - Métodos de TurtleScreen/Screen
    - Métodos de *RawTurtle/Turtle* Y sus correspondientes funciones

        - Movimiento de Turtle
        - Mostrar el estado de la tortuga
        - Configuración de las medidas
        - Control del lápiz
            - Estado de dibujo
            - Control del color
            - Relleno
            - Más controles de dibujo
        - Estado de la Tortuga
            - Visibilidad
            - Apariencia
        - Usando eventos
        - Métodos especiales de *Turtle*
        - Formas compuestas
    - Métodos de *TurtleScreen/Screen* y sus correspondientes funciones
        - Control de ventana
        - Control de animación
        - Usando eventos de pantalla
        - Métodos de entrada
        - Configuración y métodos especiales
        - Métodos específicos de *Screen*, no heredados de *TurtleScreen*
    - Clases públicas
    - Explicación
    - Ayuda y configuración
        - Cómo usar la ayuda
        - Traducción de cadenas de documentos a diferentes idiomas
        - Cómo configurar Screen and Turtles
    - **`turtledemo`** — Demo scripts
    - Cambios desde Python 2.6

○ Cambios desde Python 3.0

`tkinter` — Python interface to Tcl/Tk

Código fuente: Lib/tkinter/_init_.py [https://github.com/python/cpython/tree/3.14/Lib/tkinter/_init_.py]

The **`tkinter`** package («Tk interface») is the standard Python interface to the Tcl/Tk GUI toolkit. Both Tk and **`tkinter`** are available on most Unix platforms, including macOS, as well as on Windows systems.

Running `python -m tkinter` from the command line should open a window demonstrating a simple Tk interface, letting you know that **`tkinter`** is properly installed on your system, and also showing what version of Tcl/Tk is installed, so you can read the Tcl/Tk documentation specific to that version.

Tkinter soporta un amplio rango de versiones TCL/TK, construidos con o sin soporte de hilo. La versión oficial del binario de python incluye Tcl/Tk 8.6 con subprocesos. Vea el código fuente para el módulo **`_tkinter`** para mayor información acerca de las versiones soportadas.

Tkinter no es un subproceso liviano pero agrega una buena cantidad de su propia lógica para hacer la experiencia mas pythónico. Esta documentación se concentrará en estas adiciones y cambios, refiriéndose sobre la documentación oficial de Tcl-Tk para los detalles que no han sido cambiados.

Nota

Tcl/Tk 8.5 (2007) introdujo un moderno conjunto de componentes temáticos de interfaz de usuario en conjunto con una nueva API para ser usada. Ambas API tanto la antigua como la nueva están disponibles. La mayor parte de la documentación que podrá encontrar en línea aún usa la antigua versión de la API y por lo tanto podría estar desactualizada.

This is an optional module. If it is missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

Ver también

- **TkDocs [https://tkdocs.com/]**
  Es un extenso tutorial sobre como crear interfaces de usuario con Tkinter. Explica conceptos clave y muestra enfoques recomendados para usar la API moderna.
- **Tkinter 8.5 reference: a GUI for Python [https://www.tkdocs.com/shipman/]**
  Documentación de referencia sobre Tkinter 8.5 donde se detallan clases disponibles, métodos y opciones.

Recursos de Tcl/Tk:

- **Comandos Tk [https://www.tcl.tk/man/tcl8.6/TkCmd/contents.htm]**
  Referencia exhaustiva para cada uno de los comandos subyacentes utilizados por Tkinter.
- **Tcl/Tk Home Page [https://www.tcl.tk]**
  Documentación adicional y enlaces al núcleo de desarrollo de Tcl/Tk.

Libros:

- **Modern Tkinter for Busy Python Developers [https://www.amazon.com/Modern-Tkinter-Python-Developers-ebook/dp/B0071QDNLO/]**
  Escrito por Mark Roseman (ISBN 978-1999149567)
- **Python GUI programming with Tkinter [https://www.packtpub.com/en-us/product/python-gui-programming-with-**

tkinter-9781788835886]
By Alan D. Moore. (ISBN 978-1788835886)

- **Programming Python** **[https://learning-python.com/about-pp4e.html]**
  Escrito por Mark Lutz, que cubre los tópicos de Tkinter de forma excelente. (ISBN 978-0596158101)

- **Tcl and the Tk Toolkit** **[https://www.amazon.com/exec/obidos/ASIN/020163337X]**
  Escrito por John Ousterhout, creador de Tcl/Tk y Kevin Jones, el cual no cubre Tkinter. (ISBN 978-0321336330)

Arquitectura

Tcl/Tk no es una biblioteca única mas bien consiste en unos pocos módulos distintivos entre si. cada uno separados por funcionalidades y que poseen su documentación oficial. Las versiones del binario de Python también incluyen un módulo adicional con esto.

Tcl

Tcl es un lenguaje de programación interpretado y dinámico, tal como Python. Aunque puede ser usado en si mismo como un lenguaje de programación de uso general, lo más común es que se utilice en aplicaciones del lenguaje C como un motor de secuencias de comandos o como una interfaz hacia el kit de herramientas de Tk. La librería TCL contiene una interfaz con C que crea y gestiona una o más instancias del intérprete de TCL, corre los comandos en TCL en esas instancias y agrega comandos personalizados tanto en Tcl o C. Cada intérprete tiene su cola de eventos y entrega facilidades para enviar procesos a la cola y procesarlos. A diferencia de Python, el modelo de ejecución en Tcl es diseñado alrededor de cooperar en múltiples asignaciones y Tkinter ayuda a salvar estas diferencias (ver Threading model para más detalles).

Tk

Tk es un paquete de Tcl [https://wiki.tcl-lang.org/37432] implementado en C que agrega comandos personalizados que permiten manipular widget en la GUI. Cada objeto de la clase `Tk` agrega su instancia de interprete Tcl con Tk cargado en el. Los widget de Tk son muy personalizables, aunque a costa de una apariencia anticuada. Tk usa eventos de la cola de Tcl para generar y procesar eventos de la GUI.

Ttk

El nuevo tema Tk (Ttk) es una nueva familia de widgets de Tk que proveen una mejor apariencia en diferentes plataformas mas que varios de los widgets clásicos de Tk. Ttk es distribuido como parte de Tk estando disponible a partir de la versión 8.5. Enlaces para Python son entregados en un módulo aparte, `tkinter.ttk`.

Internamente, Tk y Ttk usan funcionalidades del sistema operativo subyacente, en otras palabras, Xlib en Unix/X11, Cocoa en macOS o GDI en Windows.

When your Python application uses a class in Tkinter, e.g., to create a widget, the `tkinter` module first assembles a Tcl/Tk command string. It passes that Tcl command string to an internal `_tkinter` binary module, which then calls the Tcl interpreter to evaluate it. The Tcl interpreter will then call into the Tk and/or Ttk packages, which will in turn make calls to Xlib, Cocoa, or GDI.

Módulos Tkinter

Support for Tkinter is spread across several modules. Most applications will need the main `tkinter` module, as well as the `tkinter.ttk` module, which provides the modern themed widget set and API:

```
from tkinter import *
from tkinter import ttk
```

class tkinter.Tk(*screenName=None*, *baseName=None*, *className='Tk'*, *useTk=True*, *sync=False*, *use=None*)

Esto crea un widget de nivel superior de Tk que generalmente es la ventana principal de una aplicación. Cada instancia tiene su propio intérprete Tcl asociado.

La clase **`Tk`** normalmente se instancia utilizando los valores predeterminados. No obstante, se reconocen los siguientes argumentos de palabra clave:

screenName

Cuando se proporciona (como cadena de caracteres), establece la variable de entorno **`DISPLAY`**. (solo X11)

baseName

Nombre del archivo de perfil. De forma predeterminada, *baseName* se deriva del nombre del programa (`sys.argv[0]`).

className

Nombre de la clase del widget. Se utiliza como archivo de perfil y como nombre con el que se invoca a Tcl (*argv0* en *interp*).

useTk

Si es *True*, inicializa el subsistema Tk. La función **`tkinter.Tcl()`** se establece en *False*.

sync

Si es `True`, ejecuta todos los comandos del servidor X de forma síncrona, por lo tanto, los errores se notifican inmediatamente. Se puede utilizar en el proceso de depuración. (solo X11)

use

Especifica el *id* de la ventana en la que se incrustar la aplicación, en lugar de crearla como una ventana independiente de nivel superior. El *id* debe especificarse de la misma manera que el valor para la opción -use de los widgets de nivel superior (es decir, emplea el formato de retorno de **`winfo_id()`**).

Hay que tener en cuenta que en algunas plataformas, esto solo funcionará correctamente si el *id* se refiere a un marco Tk o a un nivel superior que tenga habilitada la opción -container.

`Tk` lee e interpreta los archivos de perfil, nombrados `.className.tcl` y `. baseName.tcl`, en el intérprete Tcl y llama a **`exec()`** sobre el contenido de `.className.py` y `.baseName.py`. La ruta de los archivos de perfil se encuentra en la variable de entorno **`HOME`** o, si no está definida, en **`os.curdir`**.

tk

El objeto de la aplicación Tk creado al instanciar **`Tk`**. Esto proporciona acceso al intérprete de Tcl. Todos los widget que se adjuntan a la misma instancia de **`Tk`** tienen el mismo valor para el atributo **`tk`**.

master

The widget object that contains this widget. For **`Tk`**, the **`master`** is **`None`** because it is the main window. The terms *master* and *parent* are similar and sometimes used interchangeably as argument names; however, calling **`winfo_parent()`** returns a string of the widget name whereas **`master`** returns the object. *parent/child* reflects the tree-like relationship while *master* (or *container*)/*content* reflects the container structure.

children

Los descendientes directos de este widget como **`dict`** con los nombres del widget secundario como clave y los objetos de instancia secundarios como valor.

tkinter.Tcl(*screenName=None, baseName=None, className='Tk', useTk=False*)

La función **`Tcl()`** es una función de fábrica que crea un objeto muy similar al creado por la clase **`Tk`**, excepto que no inicializa el subsistema Tk. Esto suele ser útil cuando se maneja el intérprete de Tcl en un entorno en el que no se desean crear ventanas de nivel superior extrañas o donde no se puede (como los sistemas Unix/Linux sin un servidor X). Un objeto creado por el objeto `Tcl()` puede tener una ventana *Toplevel* creada (y el subsistema Tk inicializado) llamando a su método `loadtk()`.

Otros módulos que proporcionan soporte a Tk incluyen:

`tkinter`
Módulos de Tkinter.

`tkinter.colorchooser`
Cuadro de diálogo que permite al usuario elegir un color.

`tkinter.commondialog`
Clase base para cuadros de diálogo definidos en los otros módulos listados aquí.

`tkinter.filedialog`
Cuadros de diálogo por defecto que permiten al usuario especificar un archivo para abrir o guardar.

`tkinter.font`
Utilidades para ayudar a trabajar con fuentes.

`tkinter.messagebox`
Acceso a cuadros de diálogo estándar de Tk.

`tkinter.scrolledtext`
Widget de texto con una barra de desplazamiento vertical integrada.

`tkinter.simpledialog`
Cuadros de diálogo simples y funciones útiles.

`tkinter.ttk`
Themed widget set introduced in Tk 8.5, providing modern alternatives for many of the classic widgets in the main `tkinter` module.

Módulos adicionales:

`_tkinter`
A binary module that contains the low-level interface to Tcl/Tk. It is automatically imported by the main `tkinter` module, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter.

`idlelib`
Python’s Integrated Development and Learning Environment (IDLE). Based on `tkinter`.

`tkinter.constants`
Symbolic constants that can be used in place of strings when passing various parameters to Tkinter calls. Automatically imported by the main `tkinter` module.

`tkinter.dnd`
(experimental) Drag-and-drop support for `tkinter`. This will become deprecated when it is replaced with the Tk DND.

`turtle`
Gráficos de tortuga en una ventana Tk.

Guía de supervivencia de Tkinter

Esta sección no fue pensada en ser un tutorial exhaustivo sobre cada tópico de Tk o Tkinter. Para esto, puede consultar en alguno de los recursos externos que fueron presentados anteriormente. En vez de eso, esta apartado entrega una orientación breve sobre como es ve una aplicación en Tkinter, identifica conceptos claves de Tk y explica como el empaquetador de Tkinter está estructurado.

Es resto de esta sección lo ayudará a identificar las clases, métodos y opciones que necesitará en una aplicación de Tkinter y donde se puede encontrar documentación más detallada sobre esto, incluida en la guía de referencia oficial de Tcl/Tk.

Un programa simple de Hola Mundo

Comenzaremos a explorar Tkinter a través de una simple aplicación «Hola Mundo». Esta no es la aplicación mas pequeña que podríamos escribir, pero es suficiente para ilustrar algunos conceptos claves que necesitará saber.

```
from tkinter import *
from tkinter import ttk
root = Tk()
frm = ttk.Frame(root, padding=10)
frm.grid()
ttk.Label(frm, text="Hello World!").grid(column=0, row=
ttk.Button(frm, text="Quit", command=root.destroy).grid(
root.mainloop()
```

Después de las declaraciones de import, la línea siguiente será la de crear la instancia de la clase **`Tk`** la cual iniciativa Tk y crea el interprete asociado a Tcl. Esto también crea una ventana de nivel superior, conocida como la ventana raíz la cual sirve como la ventana principal de la aplicación.

En la siguiente línea se crea el marco del widget, el cual en este caso contendrá la etiqueta y el botón que crearemos después. El marco encaja dentro de la ventana raíz.

La próxima línea crea una etiqueta para el widget que contiene una cadena de texto estática. El método **`grid()`** es usado para especificar la posición del diseño de la etiqueta que está dentro del marco del widget, similar a como trabajan las tablas en HTML.

Es entonces cuando el botón del widget es creado y dejado a la derecha de la etiqueta. Una vez pulsado llamará al método **`destroy()`** de la ventana raíz.

Finalmente el método **`mainloop()`** muestra todo en pantalla y responde a la entrada del usuario hasta que el programa termina.

Conceptos importantes de Tk

Incluso con este sencillo programa se pueden mostrar conceptos claves de Tk:

widgets

Una interfaz de usuario de Tkinter está compuesta de varios *widgets* individuales. Cada widget es representado como un objeto de Python, instanciado desde clases tales como **`ttk.Frame`**, **`ttk.Label`** y **`ttk.Button`**.

jerarquía de los widgets

Los widgets se organizan en *jerarquías*. La etiqueta y el botón estaban dentro del marco el que a su vez estaba contenido dentro del marco raíz. Cuando se crea cada *child* del widget su *parent* es pasado como el primer argumento del constructor del widget.

opciones de configuración

Los widgets tienen *opciones de configuración* los cuales modifican su apariencia y comportamiento, como el texto que se despliega en la etiqueta o el botón. Diferentes clases de widgets tendrán diferentes conjuntos de opciones.

gestión del diseño de formularios

Los widgets no son añadidos automáticamente a la interfaz de usuario una vez que han sido creados. Un *administrador de geometrías* como `grid` controla donde la interfaz de usuario es colocada.

ejecución del evento

Tkinter reacciona a la entrada del usuario, realiza los cambios del programa e incluso actualiza lo que se muestra en pantalla solo cuando se ejecuta activamente un *event loop*. Si el programa no esta ejecutando el evento, su interfaz de usuario no se actualizará.

Entendiendo como funcionan los empaquetadores de Tcl/Tk

Cuando su aplicación utiliza las clases y métodos de Tkinter, internamente Tkinter está ensamblando cadenas que representan comandos Tcl/Tk, y ejecutando esos comandos en el intérprete Tcl adjunto a la instancia **`Tk`** de su aplicación.

Tanto si se está intentando navegar en la documentación de referencia o intentando encontrar el método u opción correcta o adaptando algún código ya existente o depurando su aplicación en Tkinter, hay veces que es útil entender como son esos comandos subyacentes de Tcl/Tk.

Para ilustrar, aquí se tiene un equivalente en Tcl/Tk de la parte principal del script Tkinter anterior.

```
ttk::frame .frm -padding 10
grid .frm
grid [ttk::label .frm.lbl -text "Hello World!"] -column
grid [ttk::button .frm.btn -text "Quit" -command "destro
```

La sintaxis es similar a varios lenguajes de línea de comandos, donde la primera palabra es el comando que debe ser ejecutado con los argumentos que le siguen al comando, separados por espacios, Sin entrar en demasiados detalles, note lo siguiente:

- Los comandos usados para crear widgets (tales como `ttk::frame`) corresponden a las clases de widgets en Tkinter.
- Opciones de widget en Tcl (por ejemplo `-text`) corresponden a argumentos clave en Tkinter.
- Los widgets son reverenciados con un *pathname* in Tcl (por ejemplo `.frm.btn`) mientras que Tkinter no usa nombres sino que referencia a objetos.
- El lugar del widget en la jerarquía es codificada (jerárquicamente) en nombre de archivo, usando un `.` (punto) como separador de búsqueda. El nombre de ruta del archivo para la ventana raíz es un `.` (punto), la jerarquía no es definida por la ruta del archivo pero si por la especificación del widget padre cuando se crea cada widget hijo.
- Operations which are implemented as separate *commands* in Tcl (like `grid` or `destroy`) are represented as *methods* on Tkinter widget objects. As you’ll see shortly, at other times Tcl uses what appear to be method calls on widget objects,

which more closely mirror what is used in Tkinter.

¿Cómo lo hago?, ¿Cómo funciona?

Si no esta seguro sobre como realizar ciertas acciones en Tkinter y no se puede encontrar información de forma rápida en el tutorial o en la documentación de referencias que estas usando, aquí hay varias estrategias que pueden ser de mucha ayuda.

Primero, recordar que los detalles sobre como cada widget trabaja en las diferentes versiones de Tcl/Tk pueden variar según la versión. Si está buscando información, asegúrese de que corresponde a la versión tanto de Python como de Tcl/Tk instaladas en su sistema.

Cuando se busca sobre como usar una API, puede ser muy útil el saber exactamente el nombre de la clase, opción o método que está utilizando. Tener esto en cuenta de antemano, ya sea en un shell de Python interactivo o con `print()`, puede ayudar a identificar lo que necesitas.

Para encontrar que opciones de configuración están disponibles en cada widget, se utiliza la llamada al método **`configure()`**, el cual retorna un diccionario que contiene una gran variedad de información sobre cada objeto, incluyendo valores por defecto y actuales. Utilice el **`keys()`** para saber los nombres de cada opción.

```
btn = ttk.Button(frm, ...)
print(btn.configure().keys())
```

La mayoría de los widgets tienen varias opciones de configuraciones en común, puede ser útil averiguar cuales son específicos de una clase de widget en particular. Comparando la lista de opciones de un widget sencillo como por ejemplo un frame es una de las formas de hacerlo.

```
print(set(btn.configure().keys()) - set(frm.configure().
```

De manera similar se pueden encontrar los métodos disponibles para un objeto de un widget utilizando la función standard `dir()`. Si lo intentas podrás ver alrededor de 200 métodos comunes para los widgets por lo que de nuevo, es muy importante identificar las especificaciones de un widget.

```
print(dir(btn))
print(set(dir(btn)) - set(dir(frm)))
```

Navegando en el manual de referencia de Tcl/Tk

Como se ha señalado, el manual oficial de referencia de Tk commands [https://www.tcl.tk/man/tcl8.6/TkCmd/contents.htm] (en las páginas del manual) es a menudo la descripción mas precisa sobre que hacen las operaciones especificas de los widgets. Incluso sabiendo el nombre de la opción o método que necesitas, es posible que puedas encontrar información adicional.

Mientras todas las operaciones en Tkinter son implementadas como llamadas de método sobre objetos del widget, se puede ver cuantas operaciones de Tcl/Tk aparecen como comandos que toman el nombre de ruta del widget como el primer parámetro, seguido de parámetros opcionales, por ejemplo.

```
destroy .
grid .frm.btn -column 0 -row 0
```

Otros, sin embargo, se parecen más a métodos llamados en un objeto del widget (de hecho, cuando se crea un widget en Tcl / Tk, este a su vez crea un comando Tcl con el nombre de la ruta del widget, siendo el primer parámetro de ese comando el nombre de un método para llamar).

```
.frm.btn invoke
.frm.lbl configure -text "Goodbye"
```

En la documentación oficial de referencia para Tcl/Tk, se podrán encontrar mas operaciones que son similares a las llamadas a métodos de la pagina principal de un widget en especifico (p.ej., se podrá encontrar el método **`invoke()`** en la página principal de ttk::button [https://www.tcl.tk/man/tcl8.6/TkCmd/ttk_button.htm]) mientras que las funciones que toman un widget como un parámetro a menudo tienen su propia página principal (p.ej., grid [https://www.tcl.tk/man/tcl8.6/TkCmd/grid.htm]).

Se podrán encontrar varias opciones en común y metidos en las páginas del manual options [https://www.tcl.tk/man/tcl8.6/TkCmd/options.htm] o ttk::widget [https://www.tcl.tk/man/tcl8.6/TkCmd/ttk_widget.htm], mientras que otras opciones se encuentran en las páginas del manual de una clase de widget específica.

También se puede encontrar cuantos métodos de Tkinter poseen nombres compuestos, p.ej, `winfo_x()`, `winfo_height()`, `winfo_viewable()`. Usted también encuentra información para todo esto en las páginas del manual en winfo <https://www.tcl.tk/man/tcl8.6/TkCmd/winfo.htm>`_.

Nota

Puede ser algo confuso, pero también hay métodos en todos los widgets de Tkinter los cuales no operan directamente en el widget pero operan a nivel global, independiente del widget. Ejemplos de esto son los métodos para acceder al portapapeles o a llamados del sistema. (Ellos aparecen al ser implementados como métodos en la clase base `Widget` clase que todos los widgets en Tkinter heredan).

Modelo de subprocesamiento

Python and Tcl/Tk have very different threading models, which `tkinter` tries to bridge. If you use threads, you may need to be aware of this.

Un interprete de Python puede tener varios hilos asociados. En Tcl, diferentes hilos pueden ser creados, pero cada hilo tiene una instancia del interprete de Tcl asociada por separado. Los hilos también pueden crear mas de una instancia del interprete, aunque cada instancia del intérprete puede ser usada solo por el hilo que ha sido creado.

Each `Tk` object created by `tkinter` contains a Tcl interpreter. It also keeps track of which thread created that interpreter. Calls to `tkinter` can be made from any Python thread. Internally, if a call comes from a thread other than the one that created the `Tk` object, an event is posted to the interpreter’s event queue, and when executed, the result is returned to the calling Python thread.

Las aplicaciones de Tcl/Tk son normalmente basados en eventos, lo que significa que después de la inicialización, el interprete corre un bucle de eventos (p.ej. `Tk.mainloop()`) que responde a los eventos. Al ser un evento de un solo proceso, los controladores responden rápidamente, en caso contrario ellos bloquearán el procesamiento de otros eventos. Para evitar esto, cualquier procesamiento de larga duración no deben ser procesados en el controlador de eventos, o en caso contrario ser separados en pequeñas piezas usando temporizadores o correr el proceso en otro hilo. Esto es diferente a muchos kits de herramientas de GUI donde GUI se ejecuta en un hilo completamente separado del código de la aplicación, incluido el controlador de eventos.

If the Tcl interpreter is not running the event loop and processing events, any `tkinter` calls made from threads other than the one running the Tcl interpreter will fail.

Existen varios casos especiales:

- Tcl/Tk libraries can be built so they are not thread-aware. In this case, `tkinter` calls the library from the originating Python thread, even if this is different than the thread that created the Tcl interpreter. A global lock ensures only one call occurs at a time.
- While `tkinter` allows you to create more than one instance of a `Tk` object (with its own interpreter), all interpreters that are part of the same thread share a common event queue, which gets ugly fast. In practice, don’t create more than one instance of `Tk` at a time. Otherwise, it’s best to create them in separate threads and ensure you’re running a thread-aware Tcl/Tk build.
- Bloquear los controladores de eventos no son la única manera de prevenir que el intérprete de Tcl reingrese al bucle de eventos. Es incluso posible ejecutar una cantidad de bucles de eventos anidados o abandonar el bucle de forma completa, Si está realizando alguna tarea complicada cuando se trata de eventos o hilos hay que tener en cuenta estas posibilidades.
- There are a few select `tkinter` functions that presently

work only when called from the thread that created the Tcl interpreter.

Guía práctica

Configuración de opciones

Las opciones controlan parámetros como el color y el ancho del borde de un widget. Las opciones se pueden configurar de tres maneras:

En el momento de la creación del objeto, utilizando argumentos de palabras clave

```
fred = Button(self, fg="red", bg="blue")
```

Después de la creación del objeto, tratando el nombre de la opción como un índice de diccionario

```
fred["fg"] = "red"
fred["bg"] = "blue"
```

Usando el método *config()* para actualizar múltiples atributos después de la creación del objeto

```
fred.config(fg="red", bg="blue")
```

Para obtener una explicación completa de las opciones y su comportamiento, consulte las páginas de manual de Tk para el widget en cuestión.

Tenga en cuenta que las páginas del manual enumeran «OPCIONES ESTÁNDAR» y «OPCIONES ESPECÍFICAS DE WIDGET» para cada widget. La primera es una lista de opciones que son comunes a muchos widgets, la segunda son las opciones que son específicas para ese widget en particular. Las opciones estándar están documentadas en la página del manual *options(3)* *[https://manpages.debian.org/options(3)]*.

No se hace distinción entre las opciones estándar y las opciones específicas del widget en este documento. Algunas opciones no se aplican a algunos tipos de widgets. Si un determinado widget responde a una opción particular depende de la clase del widget. Los botones tienen la opción `command`, las etiquetas no.

Las opciones admitidas por un widget dado se enumeran en la página de manual de ese widget, o se pueden consultar en tiempo de ejecución llamando al método **`config()`** sin argumentos, o llamando al método **`keys()`** en ese widget. El valor retornado en esas llamadas es un diccionario cuya clave es el nombre de la opción como una cadena (por ejemplo, `'relief'`) y cuyo valor es una tupla de 5 elementos.

Algunas opciones, como `bg`, son sinónimos de opciones comunes con nombres largos (`bg` es la abreviatura de *«background»*). Al pasar el método `config()`, el nombre de una opción abreviada retornará una tupla de 2 elementos (en lugar de 5). Esta tupla contiene nombres de sinónimos y nombres de opciones «reales» (como `('bg', 'background')`).

Índice	**Ejemplo**	**Significado**
0	`'relief'`	nombre de la opción
1	`'relief'`	nombre de la opción para la búsqueda en la base de datos
2	`'Relief'`	clase de la opción para la consulta de base de datos
3	`'raised'`	valor por defecto
4	`'groove'`	valor actual

Ejemplo:

```
>>> print(fred.config())
{'relief': ('relief', 'relief', 'Relief', 'raised', 'gro
```

Por supuesto, el diccionario impreso incluirá todas las opciones disponibles y sus valores. Esto es solo un ejemplo.

El empaquetador

The packer is one of Tk’s geometry-management mechanisms. Geometry managers are used to specify the relative positioning of widgets within their container. In contrast to the more cumbersome *placer* (which is used less commonly, and we do not cover here), the packer takes qualitative relationship specification - *above, to the left of*, *filling*, etc - and works everything out to determine the exact placement coordinates for you.

The size of any container widget is determined by the size of the «content widgets» inside. The packer is used to control where content widgets appear inside the container into which they are packed. You can pack widgets into frames, and frames into other frames, in order to achieve the kind of layout you desire. Additionally, the arrangement is dynamically adjusted to accommodate incremental changes to the configuration, once it is packed.

Tenga en cuenta que los widgets no aparecen hasta que su geometría no se haya especificado con un administrador de diseño de pantalla. Es un error común de principiante ignorar la especificación de la geometría, y luego sorprenderse cuando se crea el widget pero no aparece nada. Un objeto gráfico solo aparece después que, por ejemplo, se le haya aplicado el método **`pack()`** del empaquetador.

Se puede llamar al método *pack()* con pares palabra-clave/valor que controlan dónde debe aparecer el widget dentro de su contenedor y cómo se comportará cuando se cambie el tamaño de la ventana principal de la aplicación. Aquí hay unos ejemplos:

```
fred.pack()                     # defaults to side = "tc
fred.pack(side="left")
fred.pack(expand=1)
```

Opciones del empaquetador

Para obtener más información sobre el empaquetador y las opciones que puede tomar, consulte el manual y la página 183 del libro de John Ousterhout.

anchor

Anchor type. Denotes where the packer is to place each content in its parcel.

expand

Un valor booleano, `0` o `1`.

fill

Los valores legales son: `'x'`, `'y'`, `'both'`, `'none'`.

ipadx y *ipady*

A distance - designating internal padding on each side of the content.

padx y *pady*

A distance - designating external padding on each side of the content.

side

Los valores legales son: `'left'`, `'right'`, `'top'`, `'bottom'`.

Asociación de variables de widget

La asignación de un valor a ciertos objetos gráficos (como los widgets de entrada de texto) se puede vincular directamente a variables en su aplicación utilizando opciones especiales. Estas opciones son `variable`, `textvariable`, `onvalue`, `offvalue`, y `value`. Esta conexión funciona en ambos sentidos: si la variable cambia por algún motivo, el widget al que está conectado se actualizará para reflejar el nuevo valor.

Unfortunately, in the current implementation of **`tkinter`** it is not possible to hand over an arbitrary Python variable to a widget through a `variable` or `textvariable` option. The only kinds of variables for which this works are variables that are subclassed from a class called Variable, defined in **`tkinter`**.

Hay muchas subclases útiles de Variable ya definidas: **`StringVar`**, **`IntVar`**, **`DoubleVar`**, and **`BooleanVar`**. Para leer el valor actual de dicha variable, es necesario llamar al método **`get()`**, y para cambiar su valor, al método **`set()`**. Si se sigue este protocolo, el widget siempre rastreará el valor de la variable, sin ser necesaria ninguna otra intervención.

Por ejemplo:

```
import tkinter as tk

class App(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        self.pack()

        self.entrythingy = tk.Entry()
        self.entrythingy.pack()

        # Create the application variable.
        self.contents = tk.StringVar()
        # Set it to some value.
        self.contents.set("this is a variable")
        # Tell the entry widget to watch this variable.
        self.entrythingy["textvariable"] = self.contents

        # Define a callback for when the user hits retur
        # It prints the current value of the variable.
        self.entrythingy.bind('<Key-Return>',
                             self.print_contents)

    def print_contents(self, event):
        print("Hi. The current entry content is:",
              self.contents.get())

root = tk.Tk()
myapp = App(root)
myapp.mainloop()
```

El gestor de ventanas

In Tk, there is a utility command, `wm`, for interacting with the window manager. Options to the `wm` command allow you to control things like titles, placement, icon bitmaps, and the like. In **`tkinter`**, these commands have been implemented as methods on the **`Wm`** class. Toplevel widgets are subclassed from the **`Wm`** class, and so can call the **`Wm`** methods directly.

To get at the toplevel window that contains a given widget, you can often just refer to the widget's **`master`**. Of course if the widget has been packed inside of a frame, the **`master`** won't represent a toplevel window. To get at the toplevel window that contains an arbitrary widget, you can call the **`_root()`** method. This method begins with an underscore to denote the fact that this function is part of the implementation, and not an interface to Tk functionality.

Aquí hay algunos ejemplos típicos:

```
import tkinter as tk

class App(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.pack()

# create the application
myapp = App()

#
# here are method calls to the window manager class
#
myapp.master.title("My Do-Nothing Application")
myapp.master.maxsize(1000, 400)

# start the program
myapp.mainloop()
```

Tipos de datos de opciones Tk

anchor

Los valores legales son los puntos de la brújula: `"n"`, `"ne"`, `"e"`, `"se"`, `"s"`, `"sw"`, `"w"`, `"nw"`, y también `"center"`.

bitmap

Hay ocho nombres de *bitmaps* integrados: `'error'`, `'gray25'`, `'gray50'`, `'hourglass'`, `'info'`, `'questhead'`, `'question'`, `'warning'`. Para especificar un nombre de archivo de mapa de bits de X, indique la ruta completa del archivo, precedida por una `@`, como en `"@/usr/contrib/bitmap/gumby.bit"`.

boolean

Se puede pasar enteros 0 o 1 o las cadenas `"yes"` or `"no"`.

callback

Esto es cualquier función de Python que no toma argumentos. Por ejemplo:

```
def print_it():
    print("hi there")
fred["command"] = print_it
```

color

Los colores se pueden dar como nombres de colores de X en el archivo `rgb.txt`, o como cadenas que representan valores RGB. La cadena que representa el valor RGB toma un rango de 4 bits:`"#RGB"`, 8 bits: `"#RRGGBB"`, 12 bits» `"#RRRGGGBBB"`, o 16 bits `"#RRRRGGGGBBBB"` donde R, G, B representan cualquier dígito hexadecimal legal. Consulte la página 160 del libro de Ousterhout para más detalles.

cursor

Los nombres estándar del cursor X de `cursorfont.h` se pueden usar sin el prefijo `XC_`. Por ejemplo, para obtener un cursor de mano (**`XC_hand2`**), use la cadena `"hand2"`. También se puede especificar su propio mapa de bits y archivo de máscara. Ver página 179 del libro de Ousterhout.

distance

Las distancias de pantalla se pueden especificar tanto en píxeles como en distancias absolutas. Los píxeles se dan como números y las distancias absolutas como cadenas con el carácter final que indica unidades: `c` para centímetros, `i` para pulgadas, `m` para milímetros, `p` para puntos de impresora. Por ejemplo, 3.5 pulgadas se expresa como `"3.5i"`.

font

Tk usa un formato de lista para los nombres de fuentes, como `{courier 10 bold}`. Los tamaños de fuente expresados en números positivos se miden en puntos, mientras que los tamaños con números negativos se miden en píxeles.

geometry

Es una cadena de caracteres del estilo `widthxheight`, donde el ancho y la altura se miden en píxeles para la mayoría de los widgets (en caracteres para widgets que muestran texto). Por ejemplo: `fred["geometry"] = "200x100"`.

justify

Legal values are the strings: `"left"`, `"center"`, and `"right"`.

region

Es una cadena de caracteres con cuatro elementos delimitados por espacios, cada uno de ellos es una distancia legal (ver arriba). Por ejemplo: `"2 3 4 5"`, `"3i 2i 4.5i 2i"` y `"3c 2c 4c 10.43c"` son todas regiones legales.

relief

Determina cuál será el estilo de borde de un widget. Los valores legales son: `"raised"`, `"sunken"`, `"flat"`, `"groove"`, y `"ridge"`.

scrollcommand

Este es casi siempre el método **`set()`** de algún widget de barra de desplazamiento, pero puede ser cualquier método de un widget que tome un solo argumento.

wrap

Debe ser uno de estos: `"none"`, `"char"`, o `"word"`.

Enlaces y eventos

El método de enlace (*binding*) del comando del widget le permite observar ciertos eventos y hacer que la función de devolución de llamada se active cuando se produce ese tipo de evento. La forma del método de enlace es:

```
def bind(self, sequence, func, add=''):
```

donde:

sequence

es una cadena que denota el tipo de evento objetivo. Para obtener más información, consulte la página del manual de Bind y la página 201 del libro de John Ousterhout.

func

es una función de Python que toma un argumento y se llama cuando ocurre el evento. La instancia del evento se pasa como el argumento mencionado. (Las funciones implementadas de esta manera a menudo se llaman *callbacks*.)

add

es opcional, ya sea `''` o `'+'`. Pasar una cadena de caracteres vacía indica que este enlace anulará cualquier otro enlace asociado con este evento. Pasar `'+'` agrega esta función a la lista de funciones vinculadas a este tipo de evento.

Por ejemplo:

```
def turn_red(self, event):
    event.widget["activeforeground"] = "red"

self.button.bind("<Enter>", self.turn_red)
```

Observe cómo se accede al campo del widget del evento en la devolución de llamada `turn_red()`. Este campo contiene el widget que capturó el evento de X. La siguiente tabla enumera los otros campos de eventos a los que puede acceder y cómo se indican en Tk, lo que puede ser útil cuando se hace referencia a las páginas del manual de Tk.

Tk	**Campo evento de Tkinter**

%f	focus	%A	char
%h	height	%E	send_event
%k	keycode	%K	keysym
%s	state	%N	keysym_num
%t	time	%T	type
%w	width	%W	widget
%x	x	%X	x_root
%y	y	%Y	y_root

El parámetro índice

Muchos widgets requieren que se pase un parámetro de tipo índice. Esto se utiliza para señalar ubicaciones específicas en el widget de texto, caracteres específicos en el widget de entrada, o elementos particulares en el widget de menú.

Índice de widget de entrada (índice, índice de vista, etc.)

Entry widgets have options that refer to character positions in the text being displayed. You can use these **`tkinter`** functions to access these special points in text widgets:

Índice de widget de texto

La notación de índice del widget de texto es muy rica y se detalla mejor en las páginas del manual de Tk.

Índices de menú (*menu.invoke()*, *menu.entryconfig()*, etc.)

Algunas opciones y métodos para menús manipulan entradas de menú específicas. Cada vez que se necesita un índice de menú para una opción o un parámetro, se puede pasar:

- un número entero que se refiere a la posición numérica de la entrada en el widget, contada desde arriba, comenzando con 0;
- la cadena `"active"`, que se refiere a la posición del menú que está actualmente debajo del cursor;
- la cadena de caracteres `"last"` que se refiere al último elemento del menú;
- Un número entero precedido por `@`, como en `@6`, donde el entero es interpretado como una coordenada *y* de píxeles en el sistema de coordenadas del menú;
- la cadena de caracteres `"none"`, que indica que no hay

entrada de menú, usado frecuentemente con *menu.activate()* para desactivar todas las entradas; y, finalmente,
- una cadena de texto cuyo patrón coincide con la etiqueta de la entrada del menú, tal como se explora desde la parte superior del menú hasta la parte inferior. Tenga en cuenta que este tipo de índice se considera después de todos los demás, lo que significa que las coincidencias para los elementos del menú etiquetados `last`, `active`, o `none` pueden interpretarse de hecho como los literales anteriores.

Imágenes

Se pueden crear imágenes de diferentes formatos a través de la correspondiente subclase de **`tkinter.Image`**:

- **`BitmapImage`** para imágenes en formato XBM.
- **`PhotoImage`** para imágenes en formatos PGM, PPM, GIF y PNG. Este último es compatible a partir de Tk 8.6.

Cualquier tipo de imagen se crea a través de la opción `file` o `data` (también hay otras opciones disponibles).

Distinto en la versión 3.13: Added the **`PhotoImage`** method **`copy_replace()`** to copy a region from one image to other image, possibly with pixel zooming and/or subsampling. Add *from_coords* parameter to **`PhotoImage`** methods **`copy()`**, **`zoom()`** and **`subsample()`**. Add *zoom* and *subsample* parameters to **`PhotoImage`** method **`copy()`**.

El objeto imagen se puede usar siempre que algún widget admita una opción de `image` (por ejemplo, etiquetas, botones, menús). En estos casos, Tk no mantendrá una referencia a la imagen. Cuando se elimina la última referencia de Python al objeto de imagen, los datos de la imagen también se eliminan, y Tk mostrará un cuadro vacío donde se utilizó la imagen.

Ver también

El paquete Pillow [https://python-pillow.org/] añade soporte para los formatos del tipo BMP, JPEG, TIFF, y WebP, entre otros.

Gestor de archivos

Tk permite registrar y anular el registro de una función de devolución de llamada que se llamará desde el mainloop de Tk cuando la E/S sea posible en un descriptor de archivo. Solo se puede registrar un controlador por descriptor de archivo. Código de ejemplo:

```
import tkinter
widget = tkinter.Tk()
mask = tkinter.READABLE | tkinter.WRITABLE
widget.tk.createfilehandler(file, mask, callback)
...
widget.tk.deletefilehandler(file)
```

Esta función no está disponible en Windows.

Dado que no se sabe cuántos bytes están disponibles para su lectura, no use métodos de `BufferedIOBase` o `TextIOBase` `read()` o `readline()`, ya que estos requieren leer un número predefinido de bytes. Para *sockets*, los métodos `recv()` o `recvfrom()` trabajan bien; para otros archivos, use lectura sin formato o `os.read(file.fileno(), maxbytecount)`.

Widget.tk.createfilehandler(*file*, *mask*, *func*)

Registra la función *callback* gestor de archivos *func*. El argumento *file* puede ser un objeto con un método **`fileno()`** (como un archivo u objeto de socket), o un descriptor de archivo entero. El argumento *mask* es una combinación ORed de cualquiera de las tres constantes que siguen. La retrollamada se llama de la siguiente manera:

```
callback(file, mask)
```

Widget.tk.deletefilehandler(*file*)

Anula el registro de un gestor de archivos.

_tkinter.READABLE
_tkinter.WRITABLE
_tkinter.EXCEPTION

Constantes utilizadas en los argumentos *mask*.

`tkinter.colorchooser` — Color choosing dialog

Código fuente: Lib/tkinter/colorchooser.py [https://github.com/python/cpython/tree/3.14/Lib/tkinter/colorchooser.py]

The **`tkinter.colorchooser`** module provides the **`Chooser`** class as an interface to the native color picker dialog. `Chooser` implements a modal color choosing dialog window. The `Chooser` class inherits from the **`Dialog`** class.

class tkinter.colorchooser.Chooser(*master=None*, ***options*)

tkinter.colorchooser.askcolor(*color=None*, ***options*)

Cree un cuadro de diálogo de elección de color. Una llamada a este método mostrará la ventana, esperará a que el usuario haga una selección y devolverá el color seleccionado (o `None`) a la persona que llama.

Ver también

Módulo `tkinter.commondialog`
Módulo de diálogo estándar de Tkinter

`tkinter.font` — Tkinter font wrapper

Código fuente: Lib/tkinter/font.py [https://github.com/python/cpython/tree/3.14/Lib/tkinter/font.py]

The `tkinter.font` module provides the `Font` class for creating and using named fonts.

Los diferentes pesos e inclinaciones de la fuente son:

tkinter.font.NORMAL
tkinter.font.BOLD
tkinter.font.ITALIC
tkinter.font.ROMAN

class tkinter.font.Font(*root=None*, *font=None*, *name=None*, *exists=False*, ***options*)

La clase `Font` representa una fuente con nombre. Las instancias *Font* reciben nombres únicos y se pueden especificar por su configuración de familia, tamaño y estilo. Las fuentes con nombre son el método de Tk para crear e identificar fuentes como un solo objeto, en lugar de especificar una fuente por sus atributos con cada aparición.

argumentos:

font - tupla de especificador de fuente (familia, tamaño, opciones)
name - nombre de fuente único
exists - self apunta a la fuente con nombre existente si es

verdadera

opciones de palabras clave adicionales
(ignoradas si se especifica *font*):

family - familia de la fuente,
es decir, Courier, Times
size - tamaño de fuente
    Si *size* es positivo, se
    interpreta como tamaño
    en puntos.
    Si *size* es un número
    negativo, se trata su valor
    absoluto
    como tamaño en píxeles.
weight - énfasis de fuente
(NORMAL, BOLD)
slant - ROMAN, ITALIC
underline - subrayado de
fuente (0 - ninguno, 1 -
subrayado)
overstrike - tachado de fuente
(0 - ninguno, 1 - tachado)

actual(*option=None, displayof=None*)

Retorna los atributos de la fuente.

cget(*option*)

Recupera un atributo de la fuente.

config(***options*)

Modifica los atributos de la fuente.

copy()

Retorna una nueva instancia de la fuente actual.

measure(*text, displayof=None*)

Retorna la cantidad de espacio que ocuparía el texto en la pantalla especificada cuando se formatee en la fuente actual. Si no se especifica ninguna pantalla, se asume la ventana principal de la aplicación.

metrics(**options*, ***kw*)

Retorna datos específicos de la fuente. Las opciones incluyen:

ascent - distancia entre la línea de base y el punto más alto que un
el carácter de la fuente puede ocupar

descent - distancia entre la línea de base y el punto más bajo que un
el carácter de la fuente puede ocupar

linespace - separación vertical mínima necesaria entre dos
caracteres de la fuente que aseguran que no haya superposición vertical entre líneas.

fixed - 1 si la fuente es de ancho fijo en caso contrario 0

tkinter.font.families(*root=None*, *displayof=None*)

Retorna las diferentes familias de fuentes.

tkinter.font.names(*root=None*)

Retorna los nombres de las fuentes definidas.

tkinter.font.nametofont(*name*, *root=None*)

Retorna una representación `Font` de una fuente con nombre tk.

Distinto en la versión 3.10: Se agregó el parámetro *root*.

Diálogos tkinter

`tkinter.simpledialog` — Standard Tkinter input dialogs

Código fuente: Lib/tkinter/simpledialog.py [https://github.com/python/cpython/tree/3.14/Lib/tkinter/simpledialog.py]

The **`tkinter.simpledialog`** module contains convenience classes and functions for creating simple modal dialogs to get a value from the user.

tkinter.simpledialog.askfloat(*title*, *prompt*, ***kw*)
tkinter.simpledialog.askinteger(*title*, *prompt*, ***kw*)
tkinter.simpledialog.askstring(*title*, *prompt*, ***kw*)

> Las tres funciones anteriores proporcionan diálogos que piden al usuario que introduzca un valor del tipo deseado.

class tkinter.simpledialog.Dialog(*parent*, *title=None*)

> La clase base para los diálogos personalizados.
>
> body(*master*)
>
> > Sobrescribir para construir la interfaz del dialogo y retornar el widget que debe tener el foco inicial.
>
> buttonbox()
>
> > El comportamiento por defecto añade los botones OK y Cancelar. Sobrescribir para diseños de botones personalizados.

`tkinter.filedialog` — File selection

dialogs

Código fuente: Lib/tkinter/filedialog.py [https://github.com/python/cpython/tree/3.14/Lib/tkinter/filedialog.py]

The `tkinter.filedialog` module provides classes and factory functions for creating file/directory selection windows.

Diálogos nativos de carga/guardado

Las siguientes clases y funciones proporcionan ventanas de diálogo de archivos que combinan un aspecto nativo con opciones de configuración para personalizar el comportamiento. Los siguientes argumentos son aplicables a las clases y funciones enumeradas a continuación:

> *parent* - la ventana sobre la que se situará el diálogo
>
> *title* - el título de la ventana
>
> *initialdir* - el directorio en el que inicia el diálogo
>
> *initialfile* - el archivo seleccionado al abrir el diálogo
>
> *filetypes* - una secuencia de tuplas (label, pattern), se permite el comodín "*"
>
> *defaultextension* - extensión por defecto para añadir al archivo (diálogos de guardado)
>
> *multiple* - cuando es verdadero, se permite la selección de múltiples elementos

Funciones estáticas de factoría

Las siguientes funciones, al ser invocadas, crean un diálogo modal de aspecto nativo, esperan la selección del usuario y retornan el(los) valor(es) seleccionado(s) o `Ninguno` a la llamada.

tkinter.filedialog.askopenfile(*mode='r'*, ***options*)

tkinter.filedialog.askopenfiles(*mode='r'*, ***options*)

Las dos funciones anteriores crean un diálogo `Open` y retornan el(los) objeto(s) de archivo(s) abierto(s) en modo de sólo lectura.

tkinter.filedialog.asksaveasfile(*mode='w'*, ***options*)

Crea un diálogo `SaveAs` y retorna un objeto de un archivo abierto en modo de sólo escritura.

tkinter.filedialog.askopenfilename(***options*)

tkinter.filedialog.askopenfilenames(***options*)

Las dos funciones anteriores crean un diálogo `Open` y retornan el(los) nombre(s) de archivo(s) seleccionado(s) que corresponde(n) a un(os) archivo(s) existente(s).

tkinter.filedialog.asksaveasfilename(***options*)

Crea un diálogo `SaveAs` y retorna el nombre del archivo seleccionado.

tkinter.filedialog.askdirectory(***options*)

Pide al usuario que seleccione un directorio.
Argumento opcional adicional:

- *mustexist* - determina si la selección debe ser un directorio existente.

class tkinter.filedialog.Open(*master=None*, ***options*)

class tkinter.filedialog.SaveAs(*master=None*, ***options*)

Las dos clases anteriores proporcionan ventanas de diálogo nativas para guardar y cargar archivos.

Clases de conveniencia

Las siguientes clases se utilizan para crear ventanas de archivos/directorios desde cero. Estas no emulan el aspecto nativo de la plataforma.

class tkinter.filedialog.Directory(*master=None*, ***options*)

Crear un diálogo que solicite al usuario que seleccione un directorio.

Nota

La clase *FileDialog* debe ser subclasificada para el manejo de eventos y comportamiento personalizados.

class tkinter.filedialog.FileDialog(*master*, *title=None*)

Crear un diálogo básico de selección de archivos.

cancel_command(*event=None*)

Activa la terminación de la ventana de diálogo.

dirs_double_event(*event*)

Manejador de eventos para el evento de doble clic sobre un directorio.

dirs_select_event(*event*)

Manejador de eventos para el evento de clic sobre un directorio.

files_double_event(*event*)

Manejador de eventos para el evento de doble clic sobre un archivo.

files_select_event(*event*)

Manejador de eventos para el evento de un solo clic sobre un archivo.

filter_command(*event=None*)

Filtra los archivos por directorio.

get_filter()

Recupera el filtro de archivos que se está utilizando actualmente.

get_selection()

Recupera el elemento seleccionado actualmente.

go(*dir_or_file=os.curdir*, *pattern='*'*, *default=''*, *key=None*)

Renderiza el diálogo e inicia el bucle de eventos.

ok_event(*event*)

Salir del diálogo retornando la selección actual.

quit(*how=None*)

Salir del diálogo retornando el nombre del archivo, si lo hay.

set_filter(*dir*, *pat*)

Establece el filtro de archivos.

set_selection(*file*)

Actualiza la selección de archivos actual a *archivo*.

class tkinter.filedialog.LoadFileDialog(*master*, *title=None*)

Una subclase de FileDialog que crea una ventana de diálogo para seleccionar un archivo existente.

ok_command()

Comprueba que se proporciona un archivo y que la selección indica un archivo ya existente.

class tkinter.filedialog.SaveFileDialog(*master*, *title=None*)

Una subclase de FileDialog que crea una ventana de diálogo para seleccionar un archivo de destino.

ok_command()

Comprueba si la selección apunta a un archivo válido que no es un directorio. Se requiere confirmación si se selecciona un archivo ya existente.

`tkinter.commondialog` — Dialog window templates

Código fuente: Lib/tkinter/commondialog.py [https://github.com/python/cpython/tree/3.14/Lib/tkinter/commondialog.py]

The `tkinter.commondialog` module provides the `Dialog` class that is the base class for dialogs defined in other supporting modules.

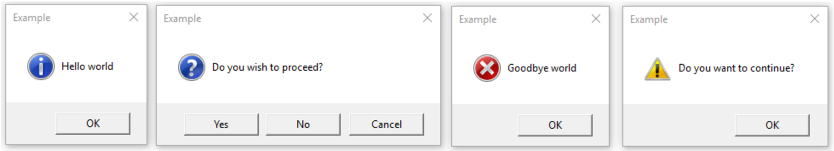
class tkinter.commondialog.Dialog(*master=None*, ***options*)

show(***options*)

Renderiza la ventana de diálogo.

Ver también

Módulos `tkinter.messagebox`, Leyendo y escribiendo archivos



Hello world

Do you wish to proceed?

Goodbye world

Do you want to continue?

`tkinter.messagebox` — Tkinter message prompts

Código fuente: Lib/tkinter/messagebox.py [https://github.com/python/cpython/tree/3.14/Lib/tkinter/messagebox.py]

The **`tkinter.messagebox`** module provides a template base class as well as a variety of convenience methods for commonly used configurations. The message boxes are modal and will return a subset of (`True`, `False`, `None`, **`OK`**, **`CANCEL`**, **`YES`**, **`NO`**) based on the user's selection. Common message box styles and layouts include but are not limited to:

class tkinter.messagebox.Message(*master=None*, ***options*)

Create a message window with an application-specified message, an icon and a set of buttons. Each of the buttons in the message window is identified by a unique symbolic name (see the *type* options).

The following options are supported:

command

Specifies the function to invoke when the user closes the dialog. The name of the button clicked by the user to close the dialog is passed as argument. This is only available on macOS.

default

Gives the symbolic name of the default button for this message window (**`OK`**, `CANCEL`, and so on). If this option is not specified, the first button in the dialog will be made the default.

detail

Specifies an auxiliary message to the main message given by the *message* option. The message detail will be presented beneath the main message and, where supported by the OS, in a less emphasized font than the main message.

icon

Specifies an icon to display. If this option is not specified, then the `INFO` icon will be displayed.

message

Specifies the message to display in this message box. The default value is an empty string.

parent

Makes the specified window the logical parent of the message box. The message box is displayed on top of its parent window.

title

Specifies a string to display as the title of the message box. This option is ignored on macOS, where platform guidelines forbid the use of a title on this kind of dialog.

type

Arranges for a predefined set of buttons to be displayed.

show(***options*)

Display a message window and wait for the user to select one of the buttons. Then return the symbolic name of the selected button. Keyword arguments can override options specified in the constructor.

Cuadro de mensaje de información

tkinter.messagebox.showinfo(*title=None*, *message=None*, ***options*)

Creates and displays an information message box with the specified title and message.

Cuadros de mensaje de advertencia

tkinter.messagebox.showwarning(*title=None*, *message=None*, ***options*)

Creates and displays a warning message box with the specified title and message.

tkinter.messagebox.showerror(*title=None*, *message=None*, ***options*)

Creates and displays an error message box with the specified title and message.

Cuadros de mensajes de preguntas

tkinter.messagebox.askquestion(*title=None*, *message=None*, ***, *type=YESNO*, ***options*)

Ask a question. By default shows buttons **`YES`** and **`NO`**. Returns the symbolic name of the selected button.

tkinter.messagebox.askokcancel(*title=None*, *message=None*, ***options*)

Ask if operation should proceed. Shows buttons **`OK`** and **`CANCEL`**. Returns `True` if the answer is ok and `False` otherwise.

tkinter.messagebox.askretrycancel(*title=None*, *message=None*,

***options*)

Ask if operation should be retried. Shows buttons **RETRY** and **CANCEL**. Return `True` if the answer is yes and `False` otherwise.

tkinter.messagebox.askyesno(*title=None*, *message=None*, ***options*)

Ask a question. Shows buttons **YES** and **NO**. Returns `True` if the answer is yes and `False` otherwise.

tkinter.messagebox.askyesnocancel(*title=None*, *message=None*, ***options*)

Ask a question. Shows buttons **YES**, **NO** and **CANCEL**. Return `True` if the answer is yes, `None` if cancelled, and `False` otherwise.

Symbolic names of buttons:

tkinter.messagebox.ABORT = *'abort'*

tkinter.messagebox.RETRY = *'retry'*

tkinter.messagebox.IGNORE = *'ignore'*

tkinter.messagebox.OK = *'ok'*

tkinter.messagebox.CANCEL = *'cancel'*

tkinter.messagebox.YES = *'yes'*

tkinter.messagebox.NO = *'no'*

Predefined sets of buttons:

tkinter.messagebox.ABORTRETRYIGNORE = *'abortretryignore'*

Displays three buttons whose symbolic names are **ABORT**, **RETRY** and **IGNORE**.

tkinter.messagebox.OK = *'ok'*

Displays one button whose symbolic name is `OK`.

tkinter.messagebox.OKCANCEL = *'okcancel'*

Displays two buttons whose symbolic names are `OK` and `CANCEL`.

tkinter.messagebox.RETRYCANCEL = *'retrycancel'*

Displays two buttons whose symbolic names are `RETRY` and `CANCEL`.

tkinter.messagebox.YESNO = *'yesno'*

Displays two buttons whose symbolic names are `YES` and `NO`.

tkinter.messagebox.YESNOCANCEL = *'yesnocancel'*

Displays three buttons whose symbolic names are `YES`, `NO` and `CANCEL`.

Icon images:

tkinter.messagebox.ERROR = *'error'*

tkinter.messagebox.INFO = *'info'*

tkinter.messagebox.QUESTION = *'question'*

tkinter.messagebox.WARNING = *'warning'*

`tkinter.scrolledtext` — Scrolled Text Widget

Código fuente: Lib/tkinter/scrolledtext.py [https://github.com/python/cpython/tree/3.14/Lib/tkinter/scrolledtext.py]

The `tkinter.scrolledtext` module provides a class of the same name which implements a basic text widget which has a vertical scroll bar configured to do the «right thing.» Using the `ScrolledText` class is a lot easier than setting up a text widget and scroll bar directly.

El widget de texto y la barra de desplazamiento se empaquetan juntos en un `Frame`, y los métodos de los administradores de geometría `Grid` y `Pack` se adquieren del objeto `Frame`. Esto permite que el widget `ScrolledText` sea usado directamente para lograr el funcionamiento más normal del administrador de geometría.

Si fuera necesario un control más específico, los siguientes atributos están disponibles:

class tkinter.scrolledtext.ScrolledText(*master=None*, ***kw*)

frame

El marco que rodea el texto y los widgets de la barra de desplazamiento.

vbar

El widget de la barra de desplazamiento.

`tkinter.dnd` — Drag and drop support

Código fuente: Lib/tkinter/dnd.py [https://github.com/python/cpython/tree/3.14/Lib/tkinter/dnd.py]

Nota

Esto es experimental y quedará obsoleto cuando se sustituya por el Tk DND.

The **`tkinter.dnd`** module provides drag-and-drop support for objects within a single application, within the same window or between windows. To enable an object to be dragged, you must create an event binding for it that starts the drag-and-drop process. Typically, you bind a ButtonPress event to a callback function that you write (see Enlaces y eventos). The function should call **`dnd_start()`**, where "source" is the object to be dragged, and "event" is the event that invoked the call (the argument to your callback function).

La selección de un objeto de destino ocurre de la siguiente manera:

1. Búsqueda de arriba hacia abajo del área debajo del mouse para el widget de destino

    - El widget de destino debe tener un atributo *dnd_accept* invocable
    - If *dnd_accept* is not present or returns `None`, search moves to parent widget
    - If no target widget is found, then the target object is `None`

1. Llama a *<old_target>.dnd_leave(source, event)*

2. Llama a *<new_target>.dnd_enter(source, event)*
3. Llama a *<target>.dnd_commit(source, event)* para notificar la caída
4. Llama a *<source>.dnd_end(target, event)* para señalar el final de arrastrar y soltar

class tkinter.dnd.DndHandler(*source*, *event*)

La clase *DndHandler* maneja los eventos de arrastrar y soltar que rastrean los eventos Motion y ButtonRelease en la raíz del widget de eventos.

cancel(*event=None*)

Cancela el proceso de arrastrar y soltar.

finish(*event*, *commit=0*)

Ejecuta el fin de las funciones de arrastrar y soltar.

on_motion(*event*)

Inspecciona el área debajo del mouse en busca de objetos de destino mientras se realiza el arrastre.

on_release(*event*)

Señal de fin de arrastre cuando se activa el patrón de liberación.

tkinter.dnd.dnd_start(*source*, *event*)

Función de fábrica para el proceso de arrastrar y soltar.

Ver también

Enlaces y eventos

`tkinter.ttk` — Tk themed widgets

Código fuente: Lib/tkinter/ttk.py [https://github.com/python/cpython/tree/3.14/Lib/tkinter/ttk.py]

The **`tkinter.ttk`** module provides access to the Tk themed widget set, introduced in Tk 8.5. It provides additional benefits including anti-aliased font rendering under X11 and window transparency (requiring a composition window manager on X11).

The basic idea for **`tkinter.ttk`** is to separate, to the extent possible, the code implementing a widget's behavior from the code implementing its appearance.

Ver también

Soporte para estilos de Widgets Tk [https://core.tcl.tk/tips/doc/trunk/tip/48.md]
Un documento que presenta apoyo temático para Tk

Uso de Ttk

Para empezar a utilizar Ttk, hay que importar su módulo:

```
from tkinter import ttk
```

Para anular los widgets Tk básicos, la importación debe seguir la importación de Tk:

```
from tkinter import *
from tkinter.ttk import *
```

That code causes several **`tkinter.ttk`** widgets (**`Button`**,

`Checkbutton`, **`Entry`**, **`Frame`**, **`Label`**, **`LabelFrame`**, **`Menubutton`**, **`PanedWindow`**, **`Radiobutton`**, **`Scale`** and **`Scrollbar`**) to automatically replace the Tk widgets.

Esto tiene el beneficio de usar los nuevos widgets que dan una mejor apariencia en todas las plataformas; sin embargo, el reemplazo de widgets no es completamente compatible. La principal diferencia es que las opciones de widgets como «fg», «bg» y otras relacionadas con el estilo del widget ya no están presentes en los widgets de Ttk. En su lugar, utiliza la clase **`ttk.Style`** para mejorar los efectos de estilo.

Ver también

Conversión de aplicaciones existentes para usar widgets Tile
[http://tktable.sourceforge.net/tile/doc/converting.txt]
Una monografía (utilizando la terminología Tcl) sobre las diferencias que normalmente se encuentran al modificar aplicaciones para usar los nuevos widgets.

Ttk widgets

Ttk viene con 18 widgets, doce de los cuales ya existían en tkinter: **`Button`**, **`Checkbutton`**, **`Entry`**, **`Frame`**, **`Label`**, **`LabelFrame`**, **`Menubutton`**, **`PanedWindow`**, **`Radiobutton`**, **`Scale`**, **`Scrollbar`** y **`Spinbox`**. Los otros seis son nuevos: **`Combobox`**, **`Notebook`**, **`Progressbar`**, **`Separator`**, **`Sizegrip`** y **`Treeview`**. Y todas ellas son subclases de **`Widget`**.

El uso de los widgets Ttk le da a la aplicación un aspecto mejorado. Como se ha mencionado anteriormente, hay diferencias en cómo se codifica el estilo.

Código Tk:

```
l1 = tkinter.Label(text="Test", fg="black", bg="white")
l2 = tkinter.Label(text="Test", fg="black", bg="white")
```

Código Ttk:

```
style = ttk.Style()
style.configure("BW.TLabel", foreground="black", backgro

l1 = ttk.Label(text="Test", style="BW.TLabel")
l2 = ttk.Label(text="Test", style="BW.TLabel")
```

Para obtener más información acerca de TtkStyling, consulta la documentación de la clase `Style`.

Widget

`ttk.Widget` define las opciones y métodos estándar compatibles con los widgets temáticos de Tk y no se crea una instancia directamente.

Opciones estándar

Todos los Widgets **`ttk`** aceptan las siguientes opciones:

Opción	Descripción
class	Especifica la clase de ventana. La clase se usa cuando se consulta la base de datos de opciones para las otras opciones de la ventana, para determinar las etiquetas de enlace (*bindtags*) predeterminadas para la ventana y para seleccionar el diseño y estilo predeterminados del widget. Esta opción es de solo lectura y solo se puede especificar cuando se crea la ventana.
cursor	Especifica el cursor del mouse que se utilizará para el widget. Si se establece en la cadena vacía (el valor predeterminado), el cursor se hereda para el widget principal.
takefocus	Determina si la ventana acepta el foco durante el recorrido del teclado. Se retorna 0, 1 o una cadena vacía. Si se retorna 0, significa que la ventana debe omitirse por completo durante el recorrido del teclado. Si es 1, significa que la ventana debe recibir el foco de entrada siempre que sea visible. Y una cadena vacía significa que los scripts transversales toman la decisión sobre si enfocarse o no en la ventana.
style	Se puede usar para especificar un estilo personalizado para el widget.

Opciones de widgets desplegables

Los widgets controlados por una barra deslizante presentan las siguientes opciones.

Opción	Descripción
xscrollcommand	Se usa para interactuar con las barras deslizantes horizontales. Cuando la vista en la ventana del widget cambia, el widget generará un comando Tcl basado en el scrollcommand. Por lo general, esta opción consiste en el método **`Scrollbar.set()`** de barras deslizantes. Esto hará que la barra deslizante se actualice cada vez que cambie la vista de la ventana.
yscrollcommand	Se utiliza para comunicarse con las barras deslizantes verticales. Para obtener más información, consulta más arriba.

Opciones de etiqueta

Las siguientes opciones son compatibles con etiquetas, botones y otros widgets similares a botones.

Opción	Descripción
text	Especifica una cadena de texto que se mostrará dentro del widget.
textvariable	Especifica un nombre cuyo valor se utilizará en lugar del recurso de opción de texto.
underline	Si se activa, especifica el índice (empezando por 0) de un carácter que se va a subrayar en la cadena de texto. El carácter subrayado se utiliza para la activación mnemotécnica.
image	Especifica una imagen que se va a mostrar. Es una lista de 1 o más elementos. El primer elemento es el nombre de imagen predeterminado. El resto de la lista es una secuencia de pares statespec/value según lo definido por `Style.map()`, especificando diferentes imágenes para usar cuando el widget está en un estado determinado o una combinación de estados. Todas las imágenes de la lista deben tener el mismo tamaño.
compound	Especifica cómo mostrar la imagen en relación con el texto, en el caso de que estén presentes las opciones de texto e imágenes. Los valores válidos son: • text: mostrar solo texto

- image: mostrar solo la imagen
- top, bottom, left, right: muestra la imagen por encima, por debajo, a la izquierda o a la derecha del texto, respectivamente.
- none: valor predeterminado. Mostrar la imagen si está presente, de lo contrario el texto.

width	Si es mayor que cero, especifica cuánto espacio, en ancho de caracteres, se debe asignar para la etiqueta de texto; si es menor que cero, especifica un ancho mínimo. Si es cero o no se especifica, se utiliza el ancho natural de la etiqueta de texto.

Opciones de compatibilidad

Opción	**Descripción**
state	Se puede establecer en «normal» o «deshabilitado» para controlar el bit de estado «deshabilitado». Esta es una opción de solo escritura: establecerlo cambia el estado del widget, pero el método `Widget.state()` no afecta a esta opción.

Estados del widget

El estado del widget es un mapa de bits de indicadores de estado independientes.

Indicador de estado	**Descripción**
active	El puntero está sobre el widget y clickeando sobre él producirá alguna acción
disabled	El widget está desactivado bajo el control del programa
focus	El widget tiene el enfoque del teclado
pressed	El widget está siendo pulsado
selected	«On», «true» o «current» para aspectos como Checkbuttons y radiobuttons
background	Windows y Mac tienen el concepto de ventana «activa» o en primer plano. El estado *background* se establece para los widgets en una ventana de fondo y se borra para los que están en la ventana en primer plano
readonly	El widget no debe permitir la modificación del usuario
alternate	Un formato de visualización alternativo específico del widget
invalid	El valor del widget no es válido

Una especificación de estado es una secuencia de nombres de estado, opcionalmente prefijados con un signo de exclamación que indica que el bit está desactivado.

ttk.Widget

Además de los métodos descritos a continuación, el **`ttk.Widget`** soporta los métodos **`tkinter.Widget.cget()`** y **`tkinter.Widget.configure()`**.

class tkinter.ttk.Widget

identify(*x*, *y*)

Retorna el nombre del elemento en la posición *x* *y* o la cadena vacía si el punto no se encuentra dentro de ningún elemento.

x e *y* son coordenadas en píxeles relativas al widget.

instate(*statespec*, *callback=None*, **args*, ***kw*)

Prueba el estado del widget. Si no se especifica una retrollamada, retorna `True` si el estado del widget coincide con *statespec* y `False` en caso contrario. Si se especifica la retrollamada, se llama con argumentos (*args*) si el estado del widget coincide con *statespec*.

state(*statespec=None*)

Modifica o pregunta el estado del widget. Si se especifica *statespec*, establece el estado del widget según éste y retorna un nuevo *statespec* informando cuales indicadores se han cambiado. Si no se especifica *statespec*, retorna los indicadores de estado habilitados actualmente.

statespec es generalmente una lista o una tupla.

Combobox

El widget **`ttk.Combobox`** combina un campo de texto con una lista desplegable de valores. Este widget es una subclase de **`Entry`**.

Además de los métodos heredados de **`Widget`**: **`Widget.cget()`**, **`Widget.configure()`**, **`Widget.identify()`**, **`Widget.instate()`** y **`Widget.state()`**, y los siguientes heredados de **`Entry`**: **`Entry.bbox()`**, **`Entry.delete()`**, **`Entry.icursor()`**, **`Entry.index()`**, **`Entry.insert()`**, **`Entry.selection()`**, **`Entry.xview()`**, tiene algunos otros métodos, descritos en **`ttk.Combobox`**.

Opciones

Este widget acepta las siguientes opciones específicas:

Opción	**Descripción**
exportselection	Valor booleano. Si se establece, la selección del widget está vinculada a la selección del Administrador de ventanas (que se puede retornar invocando Misc.selection_get, por ejemplo).
justify	Especifica cómo el texto se alinea en el widget. Uno de «left», «center» o «right».
height	Especifica el largo/altura del cuadro de la lista desplegable, en filas.
postcommand	Un script (posiblemente registrado con Misc.register) que se llama inmediatamente antes de mostrar los valores. Puede especificar qué valores mostrar.
state	Uno de «normal», «readonly» o «disabled». En el estado «readonly» el valor no se puede editar directamente y el usuario solo puede seleccionar los valores de la lista desplegable. En el estado «normal» el campo de texto se puede editar directamente. En el estado «deshabilitado» no es posible ninguna interacción.
textvariable	Especifica un nombre cuyo valor está vinculado al valor del widget. Cada vez que cambia el valor asociado a ese nombre, se actualiza el valor del widget y viceversa. Véase **`tkinter.StringVar`**.
values	Especifica la lista de valores que se mostrarán en el cuadro de lista desplegable.
width	Especifica un valor entero que indica el ancho deseado de la ventana de entrada, en caracteres de tamaño medio de la fuente del widget.

Eventos virtuales

Los widgets de cuadro combinado generan un evento virtual <<**ComboboxSelected**>> cuando el usuario selecciona un elemento de la lista de valores.

ttk.Combobox

class tkinter.ttk.Combobox

current(*newindex=None*)

Si se especifica *newindex*, establece el valor del cuadro combinado en la posición del elemento *newindex*. De lo contrario, retorna el índice del valor actual o -1 si el valor actual no está en la lista de valores.

get()

Retorna el valor actual del cuadro combinado.

set(*value*)

Establece el valor del cuadro combinado a *value*.

Spinbox

El widget **`ttk.Spinbox`** es un **`ttk.Entry`** mejorado con flechas de incremento y decremento. Se puede utilizar para números o listas de valores de cadena. Este widget es una subclase de **`Entry`**.

Además de los métodos heredados de **`Widget`**: **`Widget.cget()`**, **`Widget.configure()`**, **`Widget.identify()`**, **`Widget.instate()`** y **`Widget.state()`**, y los siguientes heredados de **`Entry`**: **`Entry.bbox()`**, **`Entry.delete()`**, **`Entry.icursor()`**, **`Entry.index()`**, **`Entry.insert()`**, **`Entry.xview()`**, tiene algunos otros métodos, descritos en **`ttk.Spinbox`**.

Opciones

Este widget acepta las siguientes opciones específicas:

Opción	Descripción
from	Valor flotante. Si se establece, este es el valor mínimo al que se reducirá el botón de disminución. Debe escribirse como `from_` cuando se usa como argumento, ya que `from` es una palabra clave de Python.
to	Valor flotante. Si se establece, este es el valor máximo al que se incrementará el botón de incremento.
increment	Valor flotante. Especifica la cantidad por la cual los botones de incremento/disminución cambian el valor. Por defecto la cantidad es de 1.0.
values	Secuencia de valores de cadena o flotantes. Si se especifica, los botones de incremento/disminución recorrerán los elementos de esta secuencia en lugar de incrementar o disminuir los números.
wrap	Valor booleano. Si es `True`, los botones de incremento y disminución pasarán del valor `to` al valor `from` o del valor `from` al valor `to`, respectivamente.
format	Valor de cadena. Especifica el formato de los números establecidos por los botones de incremento/disminución. Debe tener la forma «%W.Pf», donde W es el ancho de relleno del valor, P es la precisión, y "%" y "f" son literales.
command	Python invocable. Se llamará sin argumentos cada vez que se presione alguno de los botones de incremento o disminución.

Eventos virtuales

El widget spinbox genera un evento virtual <<**Increment**>> cuando el usuario presiona <Up>, y un evento virtual <<**Decrement**>> cuando el usuario presiona <Down>.

ttk.Spinbox

class tkinter.ttk.Spinbox

get()

Retorna el valor actual del spinbox.

set(*value*)

Establece el valor del spinbox a *value*.

Notebook

El widget Ttk Notebook administra una colección de ventanas y muestra una sola a la vez. Cada ventana secundaria está asociada a una pestaña, que el usuario puede seleccionar para cambiar la ventana que se muestra actualmente.

Opciones

Este widget acepta las siguientes opciones específicas:

Opción	Descripción
height	Si está presente y es mayor que cero, especifica la altura deseada del área del panel (sin incluir el relleno interno o las pestañas). De lo contrario, se utiliza la altura máxima de todos los paneles.
padding	Especifica la cantidad de espacio adicional que se va a agregar alrededor del exterior del bloc de notas. El relleno es una lista de hasta cuatro especificaciones de longitud izquierda superior derecha inferior. Si se especifican menos de cuatro elementos, el valor predeterminado inferior es el superior, el valor predeterminado de la derecha es el de la izquierda y el valor predeterminado superior es el de la izquierda.
width	Si está presente y es mayor que cero, especifica el ancho deseado del área del panel (sin incluir el relleno interno). De lo contrario, se utiliza el ancho máximo de todos los paneles.

Opciones de pestañas

Las opciones específicas para pestañas son:

Opción	Descripción
state	O bien «normal», «disabled» o «hidden». Si es «disabled», la pestaña no se puede seleccionar. Si es «hidden», la pestaña no se muestra.
sticky	Especifica cómo se coloca la ventana secundaria dentro del área del panel. *Value* es una cadena que contiene cero o más de los caracteres «n», «s», «e» o «w». Cada letra se refiere a un lado

	(norte, sur, este u oeste) al que se pegará la ventana secundaria, según el administrador de geometría `grid()`.
padding	Especifica la cantidad de espacio adicional que se va a agregar entre el notebook y este panel. La sintaxis es la misma que para el relleno de opciones utilizado por Notebook.
text	Especifica un texto que se muestra en la pestaña.
image	Especifica una imagen que se muestra en la pestaña. Consulta la opción de imagen descrita en `Widget`.
compound	Especifica cómo mostrar la imagen en relación con el texto, en el caso de que tanto el texto como la imagen estén presentes. Consulta Label Options para obtener valores válidos.
underline	Especifica el índice (basado en 0) de un carácter que se va a subrayar en la cadena de texto. El carácter subrayado se utiliza para la activación mnemotécnica si se llama a `Notebook.enable_traversal()`.

Identificadores de pestañas

El tab_id presente en varios métodos de `ttk.Notebook` puede tener cualquiera de las siguientes formas:

- Un entero entre cero y el número de pestañas
- El nombre de una ventana secundaria
- Un especificación de posición de la forma «@x,y» que identifique la pestaña
- El valor «current» el cual identifica la pestaña seleccionada actualmente
- El valor «end» que retorna el número de pestañas (válido solo para `Notebook.index()`)

Eventos virtuales

Este widget genera un evento virtual <<**NotebookTabChanged**>> después de seleccionar una nueva pestaña.

ttk.Notebook

class tkinter.ttk.Notebook

add(*child*, ***kw*)

Añade una nueva pestaña al notebook.

Si la ventana está actualmente administrada por el notebook pero oculta, se restaura a su posición anterior.

Consulta Tab Options para la lista de opciones disponibles.

forget(*tab_id*)

Quita la pestaña especificada por *tab_id*, desasigna y quita la ventana asociada.

hide(*tab_id*)

Oculta la pestaña especificada por *tab_id*.

La pestaña no se mostrará, pero la ventana asociada permanece administrada por el notebook y se recordará su configuración. Las pestañas ocultas se pueden restaurar con el comando **`add()`**.

identify(*x*, *y*)

Retorna el nombre del elemento de la pestaña en la posición *x*, *y* o cadena vacía si no hay ninguno.

index(*tab_id*)

Retorna el índice numérico de la pestaña especificada por *tab_id*, o el número total de pestañas si *tab_id* es la cadena «end».

insert(*pos*, *child*, ***kw*)

Añade un panel en la posición especificada.

pos es la cadena «end», un índice entero o el nombre de un elemento secundario administrado. Si el bloc de notas ya administra *child*, lo mueve a la posición especificada.

Consulta Tab Options para la lista de opciones disponibles.

select(*tab_id=None*)

Selecciona el *tab_id* especificado.

Se mostrará la ventana secundaria asociada y la ventana previamente seleccionada (si es diferente) no se debe asignar. Si se omite *tab_id*, retorna el nombre del widget del panel seleccionado actualmente.

tab(*tab_id*, *option=None*, ***kw*)

Consultar o modificar las opciones del *tab_id* específico.

Si no se proporciona *kw*, retorna un diccionario de los valores de las opciones de pestañas. Si se especifica *option*, retorna el valor de esa *option*. De lo contrario, establece las opciones en los valores correspondientes.

tabs()

Retorna una lista de ventanas administradas por el notebook.

enable_traversal()

Habilita la tabulación para una ventana de nivel superior que contenga este notebook.

Esto extenderá los enlaces para la ventana de nivel superior que contiene el notebook del siguiente modo:

- `Control-Tab`: selecciona la pestaña siguiente a la seleccionada actualmente.
- `Shift-Control-Tab`: selecciona la pestaña precedente a la seleccionada actualmente.
- `Alt-K`: donde *K* es el carácter mnemotécnico (subrayado) de cualquier pestaña, seleccionará esa pestaña.

Se pueden habilitar varios notebooks en un único nivel superior para la tabulación, incluidos los notebooks anidados. Sin embargo, la tabulación entre notebooks solo funciona correctamente si todos los paneles tienen el notebook en el que se encuentran como maestro.

Progressbar

El widget **`ttk.Progressbar`** muestra el estado de una operación de larga duración. Puede funcionar en dos modos: 1) el modo determinado que muestra la cantidad completada en función de la cantidad total de trabajo a realizar y 2) el modo indeterminado que proporciona una pantalla animada para que el usuario sepa que la operación está en curso.

Opciones

Este widget acepta las siguientes opciones específicas:

Opción	Descripción
orient	Puede ser «horizontal» o «vertical». Especifica la orientación de la barra de progreso.
length	Especifica la longitud del eje largo de la barra de progreso (ancho si horizontal, alto si es vertical).
mode	Puede ser «determinate» o «indeterminate».
maximum	Número que especifica el valor máximo. Por defecto el valor es 100.
value	El valor actual de la barra de progreso. En el modo «determinado», representa la cantidad de trabajo completado. En el modo «indeterminado», se interpreta como módulo *maximum*, es decir, la barra de progreso completa un «ciclo» cuando su valor aumenta en *maximum*.
variable	Nombre vinculado al valor de la opción. Si se especifica, el valor de la barra de progreso se establece automáticamente en el valor de este nombre cada vez que se modifica dicho valor.
phase	Variable de solo lectura. El widget incrementa periódicamente el valor de esta opción siempre que su valor sea mayor que 0 y, en modo determinado, menor que el máximo. Esta opción puede ser utilizada por el tema actual para proporcionar efectos de

animación adicionales.

ttk.Progressbar

class tkinter.ttk.Progressbar

start(*interval=None*)

Inicia el modo de incremento automático: programa un evento *timer* periódico que llama a `Progressbar.step()` cada *interval* milisegundos. Si se omite, *interval* tiene como valor predeterminado 50 milisegundos.

step(*amount=None*)

Incrementa el valor de la barra de progreso en *amount*.

amount vale 1.0 por defecto si se omite.

stop()

Para el modo de incremento automático: cancela cualquier evento *timer* periódico iniciado por `Progressbar.start()` para la barra de progreso en cuestión.

Separator

El widget **`ttk.Separator`** muestra una barra de separación horizontal o vertical.

No tiene otros métodos aparte de aquellos heredados de **`ttk.Widget`**.

Opciones

Este widget acepta las opción específica siguiente:

Opción	Descripción
orient	Puede ser «horizontal» o «vertical». Especifica la orientación del

separador.

Sizegrip

El widget **`ttk.Sizegrip`** (también conocido como *grow box*) permite al usuario cambiar el tamaño de la ventana de nivel superior que lo contiene presionando y arrastrando la esquina.

Este widget no tiene ni opciones ni métodos específicos, aparte de aquellos heredados de **`ttk.Widget`**.

Notas específicas por plataforma

- En macOS, las ventanas de nivel superior incluyen automáticamente un control de tamaño integrado de forma predeterminada. Agregar un **`Sizegrip`** es inofensivo, ya que el agarre integrado solo enmascara el widget.

Errores detectados

- Si la posición del nivel superior que lo contiene se especifica en relación con la parte derecha o inferior de la pantalla (por ejemplo,…), el widget **`Sizegrip`** no cambiará el tamaño de la ventana.
- El widget solo soporta el cambio de tamaño desde la esquina inferior derecha.

Treeview

El widget **`ttk.Treeview`** muestra una colección en árbol de elementos. Cada elemento tiene una etiqueta textual, una imagen opcional y una lista opcional de valores de datos. Los valores de datos se muestran en columnas sucesivas después de la etiqueta de árbol.

El orden en que se muestran los valores de datos se puede controlar estableciendo la opción de widget `displaycolumns`. El Treeview también puede mostrar encabezados. Se puede acceder a las columnas por número o nombres simbólicos enumerados en las columnas de opciones del widget. Consulte Column Identifiers.

Cada elemento se identifica con un nombre único. El widget genera IDs para los elementos si no se proporcionan en la declaración. Hay un elemento raíz distinguido, denominado `{}`. El elemento raíz en sí no se muestra; sus hijos aparecen en el nivel superior de la jerarquía.

Cada elemento también tiene una lista de etiquetas que se pueden usar para asociar enlaces de eventos con elementos individuales y controlar la apariencia del elemento.

El widget Treeview admite el deslizamiento horizontal y vertical, según las opciones descritas en Scrollable Widget Options y los métodos **`Treeview.xview()`** y **`Treeview.yview()`**.

Opciones

Este widget acepta las siguientes opciones específicas:

Opción	Descripción
columns	Una lista de identificadores de columna, que especifican el número de columnas y sus nombres.
displaycolumns	Una lista de identificadores de columna (índices simbólicos o enteros) que especifican qué columnas de datos se muestran y el orden en que aparecen, o la cadena «#all».
height	Especifica el número de filas que deben estar visibles. Nota: el ancho solicitado se determina a partir de la suma de los anchos de columna.
padding	Especifica el relleno interno del widget. El relleno es una lista de hasta cuatro especificaciones de longitud.
selectmode	Controla cómo los enlaces de clase integrados administran la selección. Puede ser «extended», «browse» o «none». Si se establece en «extended» (valor predeterminado), se pueden seleccionar varios elementos. Si se elige «browse», se seleccionará un solo elemento a la vez. Si se elige «none», la selección no se cambiará. Tenga en cuenta que el código de la aplicación y los enlaces de etiqueta pueden establecer la selección como deseen, independientemente del valor de esta opción.
show	Una lista que contiene cero o más de los siguientes valores,

especificando qué elementos del árbol se van a mostrar.

- tree: muestra las etiquetas del árbol en la columna #0.
- headings: muestra la fila de encabezado.

El valor predeterminado es «tree headings», es decir, mostrar todos los elementos.

Nota: la columna #0 siempre hace referencia a la columna del árbol, incluso si no se especifica show=»tree».

Opciones de elementos

Las siguientes opciones de elemento se pueden especificar para los elementos de los comandos de inserción y de elementos del widget.

Opción	**Descripción**
text	La etiqueta textual que se va a mostrar para el elemento.
image	Una imagen Tk, que se muestra a la izquierda de la etiqueta.
values	La lista de valores asociados al elemento. Cada elemento debe tener el mismo número de valores que las columnas de opciones del widget. Si hay menos valores que columnas, los valores restantes se asumen vacíos. Si hay más valores que columnas, se omiten los valores adicionales.
open	Valor `True/False` que indica si los elementos secundarios deben mostrarse u ocultarse.
tags	La lista de etiquetas asociadas al elemento.

Opciones de etiqueta

Se pueden especificar las opciones siguientes para etiquetas:

Opción	**Descripción**
foreground	Especifica el color de primer plano del texto.
background	Especifica el color de fondo de la celda o elemento.
font	Especifica la fuente que se utilizará al añadir texto.
image	Especifica la imagen del elemento, en caso de que la opción de imagen del elemento esté vacía.

Identificadores de columna

Los identificadores de columna toman cualquiera de los siguientes formas:

- Un nombre simbólico de la lista de opciones de columna.
- Un entero n, especificando la enésima columna de datos.
- Una cadena de la forma #n, donde n es un entero, especificando la enésima columna mostrada.

Notas:

- Los valores de opciones del elemento se pueden mostrar en un orden diferente al orden en el que se almacenan.
- La columna #0 siempre hace referencia a la columna del árbol, incluso si no se especifica show=»tree».

Un número de columna de datos es un índice en la lista de valores de opciones de un elemento; el número de columna se visualiza en el árbol donde se muestran los valores. Las etiquetas de árbol se muestran en la columna #0. Si no se establece la opción displaycolumns, la columna de datos n se muestra en la columna #n +1. Una vez más, **la columna #0 siempre hace referencia a la columna del árbol**.

Eventos virtuales

El widget Treeview genera los siguientes eventos virtuales.

Evento	**Descripción**
<<TreeviewSelect>>	Se genera cada vez que cambia la selección.
<<TreeviewOpen>>	Generado justo antes configurar el elemento de resalto a open=True.
<<TreeviewClose>>	Generado justo después de establecer el elemento de resalto a open=False.

Los métodos `Treeview.focus()` y `Treeview.selection()` se pueden utilizar para determinar el elemento o elementos afectados.

ttk.Treeview

class tkinter.ttk.Treeview

bbox(*item*, *column=None*)

Retorna el cuadro delimitador (en relación con la ventana del widget de Treeview) del *item* especificado con el formato (x, y, ancho, alto).

Si se especifica *column*, retorna el cuadro delimitador de esa celda. Si el *item* no está visible (es decir, si es descendiente de un elemento cerrado o se desplaza fuera de la pantalla), retorna una cadena vacía.

get_children(*item=None*)

Retorna la lista de elementos secundarios que pertenecen a *item*.

Si no se especifica *item*, retorna la raíz.

set_children(*item, *newchildren*)

Reemplaza el elemento secundario de *item* por *newchildren*.

Los elementos secundarios presentes en *item* que no están presentes en *newchildren* se separan del árbol. Ningún elemento de *newchildren* puede ser un antecesor de *item*. Tenga en cuenta que si no se especifica *newchildren* se desasocian los elementos secundarios de *item*.

column(*column, option=None, **kw*)

Consultar o modificar las opciones para la *columna* especificada.

Si no se proporciona *kw*, retorna un diccionario de los valores de opciones de columna. Si se especifica *option*, se retorna el valor de esa *option*. De lo contrario, establece las opciones en los valores correspondientes.

Las opciones/valores válidos son:

id

Retorna el nombre de la columna. Esta es una opción de solo lectura.

anchor: One of the standard Tk anchor values.

Especifica cómo se debe alinear el texto de esta columna con respecto a la celda.

minwidth: width

El ancho mínimo de la columna en píxeles. El widget Treeview no hará que la columna sea más pequeña de lo especificado por esta opción cuando se cambie el tamaño del widget o el usuario arrastre una columna.

stretch: `True/False`

Especifica si el ancho de la columna debe ajustarse cuando se cambia el tamaño del widget.

width: width

El ancho de la columna en píxeles.

Se puede establecer *column* = *«#0»* para configurar el árbol de la columna.

delete(**items*)

Elimina todos los *items* especificados y todos sus descendientes.

El elemento raíz no se elimina.

detach(**items*)

Desvincula todos los *items* especificados del árbol.

Los objetos y todos sus descendientes todavía existen, y pueden ser reinsertados en otro punto del árbol, pero no se mostrarán.

El elemento raíz no se desvincula.

exists(*item*)

Retorna `True` si el *item* especificado está presente en el árbol.

focus(*item=None*)

Si se especifica *item*, establece el elemento de foco en *item*. De lo contrario, retorna el elemento de foco actual o “” si no hay ninguno.

heading(*column*, *option=None*, ***kw*)

Consulta o modifica las opciones de encabezado para la *column* especificada.

Si no se proporciona *kw*, retorna un diccionario de los valores de opciones de encabezado. Si se especifica *option*, se retorna el valor de esa *option*. De lo contrario, establece las opciones en los valores correspondientes.

Las opciones/valores válidos son:

text: text
: El texto se muestra en el encabezado de la columna.

image: imageName
: Especifica una imagen para mostrar en la parte derecha del encabezado de la columna.

anchor: anchor
: Especifica el alineamiento del texto del encabezado. Es uno de los valores estándar de Tk anchor.

command: callback
: Una retrollamada que se ejecutará cuando se presione la etiqueta de encabezado.

Se especifica *column =»#0»* para configurar el encabezado de la columna de árbol.

identify(*component*, *x*, *y*)

Retorna una descripción del *component* especificado bajo el punto dado por *x* e *y*, o la cadena vacía si no existe dicho *component* en esa posición.

identify_row(*y*)

Retorna el identificador del elemento en la posición *y*.

identify_column(*x*)

Retorna el identificador de los datos de columna de la celda en la posición *x*.

La columna del árbol tiene ID #0.

identify_region(*x*, *y*)

Retorna uno de:

zona	**significado**
heading	Zona de encabezado del árbol.
separator	Espacio entre dos encabezados de columna.
tree	La zona del árbol.
cell	Datos de celda.

Disponibilidad: Tk 8.6.

identify_element(*x*, *y*)

Retorna el elemento en la posición *x*, *y*.

Disponibilidad: Tk 8.6.

index(*item*)

Retorna el índice entero de *item* dentro de la lista de elementos secundarios de su elemento primario.

insert(*parent*, *index*, *iid=None*, ***kw*)

Crea un nuevo elemento y retorna el identificador del elemento recién creado.

parent es el identificador del elemento primario o la cadena vacía para crear un nuevo elemento de nivel superior. *index* es un entero, o el valor «end», especificando dónde insertar el nuevo elemento en la lista de elementos secundarios del elemento primario. Si *index* es menor o igual que cero, el nuevo nodo se inserta al principio; si *index* es mayor o igual que el número actual de elementos secundarios, se inserta al final. Si se especifica *iid*, se utiliza como identificador de elemento; *iid* no debe existir en el árbol previamente. De lo contrario, se genera un nuevo identificador único.

See Item Options for the list of available options.

item(*item*, *option=None*, ***kw*)

Consulta o modifica las opciones para el *item* especificado.

Si no se proporciona ninguna opción, se retorna un diccionario con opciones/valores para el elemento. Si se especifica *option*, se retorna el valor de esa opción. De lo contrario, establece las opciones en los valores correspondientes según *kw*.

move(*item*, *parent*, *index*)

Mueve *item* a la posición *index* en la lista de elementos secundarios de *parent*.

No se permite mover un elemento bajo uno de sus descendientes. Si *index* es menor o igual que cero, *item* se mueve al principio; si es mayor o igual que el número de hijos, se mueve hasta el final. Si *item* se desvincula, se vuelve a conectar.

next(*item*)

Retorna el identificador del siguiente elemento de *item*, o “” si *item* es el último elemento secundario de su elemento primario.

parent(*item*)

Retorna el identificador del elemento primario de *item* o “” si *item* está en el nivel superior de la jerarquía.

prev(*item*)

Retorna el identificador del elemento anterior de *item*, o “” si *item* es el primer elemento secundario de su elemento primario.

reattach(*item*, *parent*, *index*)

Un alias para **`Treeview.move()`**.

see(*item*)

Garantiza que *item* está visible.

Establece todas las opciones modificables de los antecesores de *item* a `True` y desplaza el widget si es necesario para que *item* esté dentro de la parte visible del árbol.

selection()

Retorna una tupla de los elementos seleccionados.

Distinto en la versión 3.8: `selection()` ya no toma argumentos. Para cambiar el estado de selección, utiliza los siguientes métodos de selección.

selection_set(**items*)

items se convierte en la nueva selección.

Distinto en la versión 3.6: items se pueden pasar como argumentos separados, no solo como una sola tupla.

selection_add(**items*)

Añade *items* a la selección.

Distinto en la versión 3.6: items se pueden pasar como argumentos separados, no solo como una sola tupla.

selection_remove(*items*)

Elimina *elementos* de la selección.

Distinto en la versión 3.6: items se pueden pasar como argumentos separados, no solo como una sola tupla.

selection_toggle(*items*)

Alterna el estado de selección de cada elemento en *items*.

Distinto en la versión 3.6: items se pueden pasar como argumentos separados, no solo como una sola tupla.

set(*item, column=None, value=None*)

Con un argumento, retorna un diccionario de pares columna/valor para el *item* especificado. Con dos argumentos, retorna el valor actual de la *columna* especificada. Con tres argumentos, establece el valor de determinado *column* en un *item* determinado en el *value* especificado.

tag_bind(*tagname, sequence=None, callback=None*)

Enlaza una retrollamada para el evento *sequence* a la etiqueta *tagname*. Cuando se entrega un evento a un elemento, se llama a las retrollamadas de cada una de las etiquetas del elemento.

tag_configure(*tagname, option=None, **kw*)

Consulta o modifica las opciones para el *tagname* especificado.

Si no se proporciona *kw*, retorna un diccionario de la configuración de opciones para *tagname*. Si se especifica *option*, retorna el valor de esa *option* para el *tagname* especificado. De lo contrario, establece las opciones en los valores correspondientes para el *tagname* dado.

tag_has(*tagname*, *item=None*)

Si se especifica *item*, retorna 1 o 0 dependiendo de si el *item* tiene el *tagname* especificado. De lo contrario, retorna una lista de todos los elementos que tienen la etiqueta especificada.

Disponibilidad: Tk 8.6

xview(**args*)

Consulta o modifica la posición horizontal de la vista de árbol.

yview(**args*)

Consulta o modifica la posición vertical de la vista de árbol.

Ttk Styling

A cada widget en `ttk` se le asigna un estilo, que especifica el conjunto de elementos que componen el widget y cómo se organizan, junto con la configuración dinámica y predeterminada para las opciones de elemento. De forma predeterminada, el nombre del estilo es el mismo que el nombre de clase del widget, pero puede ser reemplazado por la opción de estilo del widget. Si no conoces el nombre de clase de un widget, utiliza el método `Misc.winfo_class()` (somewidget.winfo_class()).

Ver también

Presentación de la conferencia Tcl'2004 [http://tktable.sourceforge.net/tile/tile-tcl2004.pdf]

Este documento explica cómo funciona el motor de temas

class tkinter.ttk.Style

Esta clase se utiliza para manipular la base de datos de estilos.

configure(*style*, *query_opt=None*, ***kw*)

Consulta o establece el valor predeterminado de las opciones especificadas en *style*.

Cada clave en *kw* es una opción y cada valor es una cadena que identifica el valor de esa opción.

Por ejemplo, para cambiar cualquier botón por defecto a un botón plano con borde interno y un color de fondo distinto:

```
from tkinter import ttk
import tkinter

root = tkinter.Tk()

ttk.Style().configure("TButton", padding=6, rel
   background="#ccc")

btn = ttk.Button(text="Sample")
btn.pack()

root.mainloop()
```

map(*style*, *query_opt=None*, ***kw*)

Consulta o establece valores dinámicos de opciones específicas en *style*.

Cada clave en *kw* es una opción y cada valor es una lista o tupla (generalmente) que contiene *statespecs* agrupados en tuplas, listas o alguna otra preferencia. Una *statespec* es un compuesto de uno o más estados y un valor.

Un ejemplo puede hacerlo más comprensible:

```
import tkinter
from tkinter import ttk
```

```
root = tkinter.Tk()

style = ttk.Style()
style.map("C.TButton",
    foreground=[('pressed', 'red'), ('active',
    background=[('pressed', '!disabled', 'blac
    )

colored_btn = ttk.Button(text="Test", style="C.

root.mainloop()
```

Ten en cuenta que el orden de las secuencias (estado, valor) para una opción es importante, si se cambia el orden a `[('active', 'blue'), ('pressed', 'red')]` en la opción en primer plano, por ejemplo, el resultado sería un primer plano azul cuando el widget se encuentre en los estados *active* o *pressed*.

lookup(*style, option, state=None, default=None*)

Retorna el valor especificado para *option* en *style*.

Si *state* se especifica, se espera que sea una secuencia de uno o más estados. Si se establece el argumento *default*, se usa como valor de reserva en caso de que no se especifique una opción.

Para verificar qué fuente usa un Button por defecto:

```
from tkinter import ttk

print(ttk.Style().lookup("TButton", "font"))
```

layout(*style, layoutspec=None*)

Define el diseño del widget para un *estilo* dado. Si se omite *layoutspec*, retorna la especificación de diseño para un estilo determinado.

layoutspec, si se especifica, se espera que sea una lista o algún otro tipo de secuencia (excluyendo cadenas), donde cada elemento debe ser una tupla y el primer elemento es el nombre de diseño y el segundo elemento debe tener el formato descrito en Layouts.

Para entender el formato, consulta el siguiente ejemplo (no está pensado para hacer nada útil):

```
from tkinter import ttk
import tkinter

root = tkinter.Tk()

style = ttk.Style()
style.layout("TMenubutton", [
   ("Menubutton.background", None),
   ("Menubutton.button", {"children":
       [("Menubutton.focus", {"children":
           [("Menubutton.padding", {"children":
               [("Menubutton.label", {"side": '
           })]
       })]
   }),
])

mbtn = ttk.Menubutton(text='Text')
mbtn.pack()
root.mainloop()
```

element_create(*elementname*, *etype*, **args*, ***kw*)

Create a new element in the current theme, of the given *etype* which is expected to be either «image», «from» or «vsapi». The latter is only available in Tk 8.6 on Windows.

Si se utiliza «image», *args* debe contener el nombre de imagen predeterminado seguido de pares statespec/value (esta es imagespec), y *kw* puede tener las siguientes opciones:

border = padding

> el relleno interno es una lista de hasta cuatro enteros, especificando los bordes izquierdo, superior, derecho e inferior, respectivamente.

height = height

> Especifica una altura mínima para el elemento. Si es menor que cero, la altura de la imagen base se utiliza como valor predeterminado.

padding = padding

> Especifica el relleno interior del elemento. El valor predeterminado es el valor del borde si no se especifica.

sticky = spec

> Especifica cómo se coloca la imagen dentro de la sección. *spec* contiene cero o más caracteres «n», «s», «w» o «e».

width = width

> Especifica un ancho mínimo para el elemento. Si es menor que cero, el ancho de la imagen base se utiliza como valor predeterminado.

Example:

```
img1 = tkinter.PhotoImage(master=root, file='bu
img1 = tkinter.PhotoImage(master=root, file='bu
img1 = tkinter.PhotoImage(master=root, file='bu
style = ttk.Style(root)
style.element_create('Button.button', 'image',
                     img1, ('pressed', img2), (
                     border=(2, 4), sticky='we'
```

Si se utiliza «from» como valor de *etype*, **`element_create()`** clonará un elemento existente. *args* contiene un themename, desde el que se clonará el elemento y, opcionalmente, un elemento desde el que clonar. Si no se especifica este elemento para clonar, se usará un elemento vacío. *kw* se descarta.

Example:

```
style = ttk.Style(root)
style.element_create('plain.background', 'from'
```

If «vsapi» is used as the value of *etype*, **`element_create()`** will create a new element in the current theme whose visual appearance is drawn using the Microsoft Visual Styles API which is responsible for the themed styles on Windows XP and Vista. *args* is expected to contain the Visual Styles class and part as given in the Microsoft documentation followed by an optional sequence of tuples of ttk states and the corresponding Visual Styles API state value. *kw* may have the following options:

padding = padding

> Specify the element’s interior padding. *padding* is a list of up to four integers specifying the left, top, right and bottom padding quantities respectively. If fewer than four elements are specified, bottom defaults to top, right defaults to left, and top defaults to left. In other words, a list of three numbers specify the left, vertical, and right padding; a list of two numbers specify the horizontal and the vertical padding; a single number specifies the same padding all the way around the widget. This option may not be mixed with any other options.

margins = padding

> Specifies the elements exterior padding. *padding* is a list of up to four integers specifying the left, top, right and bottom padding quantities respectively. This option may not be mixed with any other options.

width = width

> Specifies the width for the element. If this option is set then the Visual Styles API will not be queried for the recommended size or the part. If

this option is set then *height* should also be set. The *width* and *height* options cannot be mixed with the *padding* or *margins* options.

height=height

Specifies the height of the element. See the comments for *width*.

Example:

```
style = ttk.Style(root)
style.element_create('pin', 'vsapi', 'EXPLORERE
                     ('pressed', '!selected', 3
                     ('active', '!selected', 2)
                     ('pressed', 'selected', 6)
                     ('active', 'selected', 5),
                     ('selected', 4),
                     ('', 1)])
style.layout('Explorer.Pin',
             [('Explorer.Pin.pin', {'sticky': '
pin = ttk.Checkbutton(style='Explorer.Pin')
pin.pack(expand=True, fill='both')
```

Distinto en la versión 3.13: Added support of the «vsapi» element factory.

element_names()

Retorna la lista de elementos definidos en le tema actual.

element_options(*elementname*)

Retorna la lista de opciones de *elementname*.

theme_create(*themename, parent=None, settings=None*)

Crea un tema nuevo.

Es un error si *themename* ya existe. Si se especifica *parent*, el nuevo tema heredará estilos, elementos y diseños del tema primario. Si se especifica *settings*, se espera que tengan la misma sintaxis utilizada para `theme_settings()`.

theme_settings(*themename*, *settings*)

Establece temporalmente el tema actual en *themename*, aplica los *settings* especificados y, a continuación, restaura el tema anterior.

Cada clave en *settings* es un estilo y cada valor puede contener las teclas “configure”, “map”, “layout” y “element create” y se espera que tengan el mismo formato especificado por los métodos `Style.configure()`, `Style.map()`, `Style.layout()` y `Style.element_create()` respectivamente.

Como ejemplo, vamos a cambiar un poco el Combobox por el tema predeterminado:

```
from tkinter import ttk
import tkinter

root = tkinter.Tk()

style = ttk.Style()
style.theme_settings("default", {
   "TCombobox": {
       "configure": {"padding": 5},
       "map": {
           "background": [("active", "green2"),
                          ("!disabled", "green4
           "fieldbackground": [("!disabled", "g
           "foreground": [("focus", "OliveDrab1
                          ("!disabled", "OliveD
       }
   }
})
```

```
combo = ttk.Combobox().pack()

root.mainloop()
```

theme_names()

Retorna una lista de temas conocidos.

theme_use(*themename=None*)

Si no se proporciona *themename*, retorna el tema en uso. De lo contrario, establece el tema actual en *themename*, actualiza todos los widgets y emite un evento <<ThemeChanged>>.

Diseños

A layout can be just `None`, if it takes no options, or a dict of options specifying how to arrange the element. The layout mechanism uses a simplified version of the pack geometry manager: given an initial cavity, each element is allocated a parcel.

Las opciones/valores válidos son:

side: whichside

Especifica en qué lado de la cavidad colocar el elemento; sea arriba, derecha, abajo o izquierda. Si se omite, el elemento ocupa toda la cavidad.

sticky: nswe

Especifica dónde se coloca el elemento dentro de su sección asignada.

unit: 0 or 1

Si se establece en 1, hace que el elemento y todos sus descendientes sean tratados como un único elemento en métodos como **`Widget.identify()`** y otros. Se utiliza para cosas como los *thumbs* de la barra de desplazamiento.

children: [sublayout…]

Especifica una lista de elementos para colocar dentro del elemento. Cada elemento es una tupla (u otro tipo de secuencia) donde el primer elemento es el nombre de diseño y el otro es un Layout.

IDLE — Python editor and shell

Código fuente: Lib/idlelib/ [https://github.com/python/cpython/tree/3.14/Lib/idlelib/]

IDLE es el entorno de desarrollo integrado de Python.

IDLE tiene las siguientes características:

- multiplataforma: funciona en su mayoría igual en Windows, Unix y macOS
- La ventana del shell de Python (interprete interactivo) con coloreado de código de entrada, salida y mensajes de error
- editor de texto multiventana con deshacer múltiple, coloreación Python, indentado inteligente, sugerencias de llamadas a funciones, autocompletado y otras características
- búsqueda dentro de cualquier ventana, reemplazo dentro de las ventanas del editor, y búsqueda a través de múltiples archivos (grep)
- depurador con breakpoints persistentes, por pasos y visualización de espacios de nombres globales y locales
- configuración, navegadores y otros cuadros de diálogo

The IDLE application is implemented in the `idlelib` package.

This is an optional module. If it is missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

Menús

IDLE tiene dos tipos de ventana principales, la ventana del shell y la ventana del editor. Es posible tener múltiples ventanas de edición simultáneamente. En Windows y Linux, cada una tiene su propio menú principal. Cada menú documentado abajo indica con cuál tipo de ventana está asociada esta.

Las ventanas de salida, como las que se usan para Editar = > Encontrar en archivos, son un subtipo de la ventana de edición. Actualmente tienen el mismo menú principal pero un título predeterminado y un menú contextual diferente.

En macOS, hay un menú de aplicación. Este cambia dinámicamente de acuerdo la ventana actualmente seleccionada. Tiene un menú IDLE y algunas de las entradas descritas a continuación se mueven de acuerdo con las pautas de Apple.

Menú de archivo (Shell y Editor)

Nuevo archivo

Crea una nueva ventana de edición de archivos.

Abrir…

Abre un archivo existente con un cuadro de dialogo para abrir.

Abrir módulo…

Abre un módulo existente (buscar en sys.path).

Archivos recientes

Abre una lista de archivos recientes. Haga click en alguno para abrirlo.

Navegador de módulo

Muestra las funciones, clases y métodos en estructura de árbol en el archivo actual del Editor. En el shell, primero abra un módulo.

Navegador de ruta

Muestra directorios, módulos, funciones, clases y métodos de sys.path en una estructura de árbol.

Guardar

Guarda la ventana actual en el archivo asociado, si existe alguno. Las ventanas que han sido modificadas desde que se abrieron o se guardaron por última vez tienen un * antes y después del título de la ventana. Si no hay un archivo asociado, ejecute Guardar como en su lugar.

Guardar como…

Save the current window with a Save As dialog. The file saved becomes the new associated file for the window. (If your file manager is set to hide extensions, the current extension will be omitted in the file name box. If the new filename has no “.”, “.py” and “.txt” will be added for Python and text files, except that on macOS Aqua,”.py” is added for all files.)

Guardar copia como…

Guarda la ventana actual en un archivo diferente sin cambiar el archivo asociado. (Consultar la nota anterior Guardar como sobre las extensiones de nombre de archivo.)

Imprimir ventana

Imprime la ventana actual en la impresora predeterminada.

Cerrar ventana

Cierra la ventana actual (si es un editor no guardado, solicita guardar; si es un Shell no guardado, solicita salir de la ejecución). Llamar a `exit()` o `close()` en la ventana de Shell también cierra el Shell. Si esta es la única ventana, también sale del IDLE.

Salir del IDLE

Cierra todas las ventanas y sale del IDLE (solicita guardar ventanas de edición no guardadas).

Menú editar (Shell y Editor)

Deshacer

Deshace el último cambio a la ventana actual. Se puede deshacer un máximo de 1000 cambios.

Rehacer

Vuelve a aplicar el último cambio deshecho a la ventana actual.

Seleccionar todo

Selecciona el contenido completo de la ventana actual.

Cortar

Copia la selección en el portapapeles global; después elimina la selección.

Copiar

Copia la selección en el portapapeles global.

Pegar

Inserta el contenido del portapapeles global en la ventana actual.

Las funciones del portapapeles también están disponibles en los menús contextuales.

Encontrar…

Abre un cuadro de diálogo de búsqueda con muchas opciones

Buscar de nuevo

Repite la última búsqueda, si hay alguna.

Encontrar selección

Busca la cadena de caracteres seleccionada actualmente, si hay alguna.

Encontrar en Archivos…

Abre un cuadro de diálogo de búsqueda de archivos. Presenta los resultados en una nueva ventana de salida.

Reemplazar…

Abre un cuadro de diálogo de búsqueda y reemplazo.

Ir a la línea

Mueve el cursor al inicio de la línea solicitada y hace que esa línea sea visible. Una solicitud a partir del final del archivo lo lleva al final. Borra cualquier selección y actualiza el estado de la línea y la columna.

Mostrar complementos

Open a scrollable list allowing selection of existing names. See Completions in the Editing and Navigation section below.

Expandir palabra

Completa un prefijo que se ha escrito para que coincida con una palabra completa en la misma ventana; intente nuevamente para obtener un complemento diferente.

Mostrar sugerencias de llamada

After an unclosed parenthesis for a function, open a small window with function parameter hints. See Calltips in the Editing and Navigation section below.

Mostrar los paréntesis alrededor

Resalta los paréntesis.

Menú de formato (solo ventana del Editor)

Formatear párrafo

Rewrap the text block containing the text insert cursor. Avoid code lines. See Format block in the Editing and Navigation section below.

Zona de indentación

Desplaza las líneas seleccionadas a la derecha en un nivel de indentación (por defecto 4 espacios).

Zona de deindentación

Desplaza las líneas seleccionadas hacia la izquierda en un nivel de indentación (por defecto 4 espacios).

Zona comentada

Inserta ## delante de las líneas seleccionadas.

Zona descomentada

Elimina los # o ## al inicio de las líneas seleccionadas.

Zona tabulada

Transforma los tramos de espacios *iniciales* en tabs. (Nota: Recomendamos usar 4 bloques de espacio para indentar el código de Python).

Zona destabulada

Transforma *todos* los tabs en el número correcto de espacios.

Alternar tabs

Abre un cuadro de diálogo para cambiar entre indentado con espacios y tabs.

Nuevo tamaño de indentación

Abre un cuadro de diálogo para cambiar el tamaño de indentación. El valor predeterminado aceptado por la comunidad de Python es de 4 espacios.

Strip Trailing Whitespace

Remove trailing space and other whitespace characters after the last non-whitespace character of a line by applying **`str.rstrip()`** to each line, including lines within multiline strings. Except for Shell windows, remove extra newlines at the end of the file.

Menú ejecutar (solo ventana Editor)

Módulo ejecutar

Hace lo que dice en Verificar módulo. Si no hay errores, reinicia el shell para limpiar el entorno, luego ejecuta el módulo. La salida es mostrada en la ventana de shell. Tener en cuenta que la visualización requiere el uso de `print` o `write`. Cuando finaliza la ejecución, el shell permanece activo y muestra un mensaje. En este punto, se puede explorar interactivamente el resultado de la ejecución. Esto es similar a ejecutar un archivo con `python -i file` en una línea de comando.

Ejecutar… Personalizado

Igual que Módulo ejecutar, pero ejecuta el módulo con parámetros personalizados. *Los argumentos de la línea de comandos* extienden **`sys.argv`** como si se pasaran por una línea de comando. El módulo se puede ejecutar en el shell sin reiniciar.

Verificar módulo

Comprueba la sintaxis del módulo actualmente abierto en la ventana de edición. Si el módulo no ha sido guardado IDLE solicitará al usuario que guarde o guarde automáticamente, como se seleccionó en la pestaña General del cuadro de diálogo Configuración de inactividad. Si hay algún error de sintaxis, la ubicación aproximada será indicada en la ventana del Editor.

Shell de Python

Abrir o activar la ventana del shell de Python.

Menú de shell (solo ventana de shell)

Ver el último reinicio

Navega la ventana del shell hasta el último reinicio del mismo.

Reiniciar shell

Reinicia el shell para limpiar el entorno.

Historial anterior

Recorre los comandos anteriores en el historial que coinciden con la entrada actual.

Historial siguiente

Recorre los comandos posteriores en el historial que coinciden con la entrada actual.

Ejecución de interrupción

Detiene un programa en ejecución.

Menú de depuración (solo ventana de shell)

Ir al Archivo/Línea

Busca en la línea actual, con el cursor y la línea de arriba para un nombre de archivo y número de línea. Si lo encuentra, abre el archivo si aún no está abierto y muestra la línea. Usa esto para ver las líneas de origen referenciadas en un rastreo de excepción y las líneas encontradas por Buscar en archivos. También disponible en el menú contextual de la ventana del shell y las ventanas de salida.

Depurador (alternar)
: Cuando esta función está habilitada, el código ingresado en el shell o ejecutado desde el editor se ejecutará con el depurador. En el editor, los breakpoints se pueden establecer con el menú contextual. Esta funcionalidad aún está incompleta y es en cierto modo experimental.

Visualizador de pila
: Muestra el seguimiento de la pila de la última excepción en un complemento de árbol, con acceso a locales y globales.

Auto-abrir visualizador de pila
: Activa/desactiva automáticamente el visualizador de pila en una excepción no controlada.

Menú de opciones (Shell y editor)

Configurar IDLE
: Abre un cuadro de diálogo de configuración y cambia las preferencias por lo siguiente: fuentes, indentación, combinaciones de teclas, temas de color de texto, ventanas y tamaño de inicio, fuentes de ayuda adicionales y extensiones. En macOS, abre el cuadro de diálogo de configuración seleccionando Preferencias en el menú de la aplicación. Para obtener más detalles, consultar Configurar preferencias en Ayuda y preferencias.

La mayoría de los ajustes de configuración se aplican a todas las ventanas, abiertas o no. Los siguientes elementos de opción se aplican solo a la ventana activa.

Mostrar/Ocultar el contexto del código (solo ventana del Editor)
: Abre un panel en la parte superior de la ventana de edición el cual muestra el contexto de bloque de código que se ha desplazado sobre la parte superior de la ventana. Consultar Contexto de código en la sección Edición y Navegación a continuación.

Mostrar/Ocultar números de línea (solo ventana del Editor)

Abre una columna a la izquierda de la ventana de edición que muestra el número de cada línea de texto. El valor por defecto de esta característica es desactivado, este puede modificarse en las preferencias (consultar Configuración de preferencias).

Ampliar/Restaurar altura

Alterna la ventana entre el tamaño normal y la altura máxima. El tamaño inicial predeterminado es 40 líneas por 80 caracteres a menos que se cambie en la pestaña General del cuadro de diálogo Configurar IDLE. La altura máxima para una pantalla es determinada maximizando momentáneamente una ventana la primera vez que se acerca la pantalla. Cambiar la configuración de la pantalla puede invalidar la altura guardada. Este alternado no tiene efecto cuando se maximiza una ventana.

Menú de ventana (shell y editor)

Enumera los nombres de todas las ventanas abiertas; seleccione uno para ponerlo en primer plano (deiconificándolo si es necesario).

Menú de ayuda (shell y editor)

Acerca de IDLE

Muestra la versión, derechos de autor, licencia, créditos y más.

Ayuda de IDLE

Muestra este documento IDLE, que detalla las opciones del menú, edición y navegación básica y otros consejos.

Documentación de Python

Accede a la documentación local de Python, si está instalada, o inicia un navegador web y abre docs.python.org mostrando la última documentación de Python.

Demostración Turtle

Ejecuta el módulo turtledemo con ejemplos de código Python y dibujos de tortugas.

Se pueden agregar fuentes de ayuda adicionales aquí con el cuadro de diálogo Configurar IDLE en la pestaña General. Consultar la subsección Fuentes de ayuda a continuación para obtener más información sobre las opciones del menú Ayuda.

Menús contextuales

Se abre un menú contextual haciendo click derecho en una ventana (Control-click en macOS). Los menús contextuales tienen las funciones estándar del portapapeles también en el menú Editar.

Cortar
: Copia la selección en el portapapeles global; después elimina la selección.

Copiar
: Copia la selección en el portapapeles global.

Pegar
: Inserta el contenido del portapapeles global en la ventana actual.

Las ventanas del editor también tienen funciones de breakpoint. Las líneas con un conjunto de breakpoints están especialmente marcadas. Los breakpoints solo tienen efecto cuando se ejecutan bajo el depurador. Los breakpoints para un archivo se guardan en el directorio `.idlerc` del usuario.

Establecer breakpoint
: Establecer un breakpoint en la línea actual.

Eliminar breakpoint
: Eliminar el breakpoint en esa línea.

Las ventanas de shell y de salida también tienen los siguientes elementos.

Ir a archivo/línea

Hace lo mismo que el menú depurar.

La ventana de shell también tiene una función de *squeezing* de salida explicada en la subsección *ventana de shell de Python* a continuación.

Exprimir
: Si el cursor está sobre una línea de salida, comprime toda la salida entre el código de arriba y el mensaje de abajo hasta la etiqueta “Texto *squeezed*”.

Edición y navegación

Ventana del editor

IDLE puede abrir ventanas del editor cuando se inicia, dependiendo de la configuración y de cómo inicies el IDLE. A partir de ahí, usar el menú Archivo. Solo puede haber una ventana de editor abierta para un archivo determinado.

La barra de título contiene el nombre del archivo, la ruta completa y la versión de Python e IDLE que ejecuta la ventana. La barra de estado contiene el número de línea (“Ln”) y el número de columna (“Col”). Los números de línea comienzan con 1; los números de columna con 0.

El IDLE supone que los archivos con una extensión .py* conocida contienen código Python y que los otros archivos no. Ejecuta el código Python con el menú Ejecutar.

Atajos de teclado

The IDLE insertion cursor is a thin vertical bar between character positions. When characters are entered, the insertion cursor and everything to its right moves right one character and the new character is entered in the new space.

Several non-character keys move the cursor and possibly delete characters. Deletion does not puts text on the clipboard, but IDLE has an undo list. Wherever this doc discusses keys, “C” refers to the `Control` key on Windows and Unix and the `Command` key on macOS. (And all such discussions assume that the keys have not been re-bound to something else.)

- Arrow keys move the cursor one character or line.
- `C-LeftArrow` and `C-RightArrow` moves left or right one word.
- `Home` and `End` go to the beginning or end of the line.
- `Page Up` and `Page Down` go up or down one screen.
- `C-Home` and `C-End` go to beginning or end of the file.
- `Backspace` and `Del` (or `C-d`) delete the previous or next character.
- `C-Backspace` and `C-Del` delete one word left or right.
- `C-k` deletes ("kills") everything to the right.

Las combinaciones de teclas estándar (como `C-c` para copiar y `C-v` para pegar) pueden funcionar. Las combinaciones de teclas se seleccionan en el cuadro de diálogo Configurar IDLE.

Indentación automática

Después de una declaración de apertura de bloque, la siguiente línea está indentada por 4 espacios (en la ventana del shell de Python por un tab). Después de ciertas palabras clave (saltar, retornar, etc.), la siguiente línea se deindenta. En la indentación principal, `Backspace` elimina hasta 4 espacios si están allí. `Tab` inserta espacios (en la ventana del shell de Python un tab), el número depende del tamaño de la indentación. Actualmente, los tabs están restringidos a cuatro espacios debido a las limitaciones de Tcl/Tk.

Consulte también los comandos de zona de indexación/deindentación en Menú de formato.

Buscar y reemplazar

Cualquier selección se convierte en un objetivo de búsqueda. Sin embargo, solo funcionan las selecciones dentro de una línea porque las búsquedas solo se realizan dentro de las líneas con la nueva línea de terminal eliminada. Si se marca la `[x] Expresión regular`, el objetivo se interpreta de acuerdo al módulo *re* de

Python.

Terminaciones

Cuando se solicitan y están disponibles, se suministran complementos para los nombres de los módulos, los atributos de las clases o las funciones, o los nombres de los archivos. Cada método de solicitud muestra un cuadro para completar con los nombres existentes. (Para cualquier cuadro, cambie el nombre que se está completando y el elemento resaltado en el cuadro escribiendo y borrando caracteres; pulsando las teclas `Up`, `Down`, `PageUp`, `PageDown`, `Home`, y `End`; y con un solo clic dentro del cuadro. Cierra la caja con las teclas `Escape`, `Enter` y doble `Tab` o con clics fuera de la caja. Un doble clic dentro de la caja selecciona y cierra.

Una forma de abrir una caja es escribir un carácter clave y esperar un intervalo predefinido. Este intervalo es por defecto de 2 segundos; se puede modificar en el diálogo de configuración. (Para evitar las ventanas emergentes automáticas, establezca el retardo a un número grande de milisegundos, como 100000000). Para nombres de módulos importados o atributos de clases o funciones, escriba ".". Para nombres de archivos en el directorio raíz, escriba **`os.sep`** o **`os.altsep`** inmediatamente después de una comilla de apertura. (En Windows, se puede especificar una unidad de disco primero.) Muévase a los subdirectorios escribiendo un nombre de directorio y un separador.

En lugar de esperar, o después de cerrar una caja, abra una caja de finalización inmediatamente con Mostrar finalizaciones en el menú Edición. La tecla rápida por defecto es `C-espacio`. Si se teclea un prefijo para el nombre deseado antes de abrir el cuadro, la primera coincidencia o casi coincidencia se hace visible. El resultado es el mismo que si se introduce un prefijo después de mostrar la caja. Mostrar las terminaciones después de una cita completa de los nombres de archivo en el directorio actual en lugar de un directorio raíz.

Pulsar `Tab` después de un prefijo suele tener el mismo efecto que el de mostrar las terminaciones. (Sin prefijo, se indenta.) Sin embargo, si solo hay una coincidencia con el prefijo, esa coincidencia se añade inmediatamente al texto del editor sin abrir una caja.

Al invocar “Mostrar las terminaciones”, o al pulsar `Tab` después de un prefijo, fuera de una cadena y sin un “.” precedente, se abre un cuadro con las palabras clave, los nombres incorporados y los nombres disponibles a nivel de módulo.

Cuando se edita el código en un editor (a diferencia de Shell), aumentar los nombres disponibles a nivel de módulo mediante la ejecución de su código y no reiniciar el Shell después. Esto es especialmente útil después de añadir importaciones en la parte superior de un archivo. Esto también aumenta las posibles terminaciones de atributos.

Los cuadros de excluyen inicialmente los valores que empiezan por “_” o, en el caso de los módulos, no incluidos en “_all_”. Se puede acceder a los nombres ocultos escribiendo “_” después de “.”, ya sea antes o después de abrir la caja.

Sugerencias de llamada

Se muestra una sugerencia de llamada cuando se escribe `(` después del nombre de una función *accesible*. Una expresión de nombre puede incluir puntos y subíndices. Una sugerencia permanece hasta que se hace click, el cursor se mueve fuera del área de argumento o es escrito `)`. Cada vez que el cursor está en la parte del argumento de una definición, el menú o acceso directo muestra una sugerencia de llamada.

La sugerencia de llamada consiste en la firma de la función y el docstring hasta la primera línea en blanco de este último o la quinta línea no en blanco. (Algunas funciones incorporadas carecen de una firma accesible.) Un “/” o “*” en la firma indica que los argumentos anteriores o posteriores se pasan solo por posición o por nombre (palabra clave). Los detalles están sujetos a cambios.

En Shell, las funciones accesibles dependen de los módulos que se hayan importado en el proceso del usuario, incluidos los importados por el propio Idle, y de las definiciones que se hayan ejecutado, todo ello desde el último reinicio.

Por ejemplo, reinicie el Shell e ingrese `itertools.count(`. Una sugerencia de llamada aparece porque el Idle importa itertools en el proceso del usuario para su propio uso. (Esto puede cambiar.) Al ingresar `turtle.write(` nada aparece. El Idle no importa turtle. El menú o el acceso directo tampoco hace nada. Al ingresar `import turtle` y luego `turtle.write(` funcionará.

En un editor, las declaraciones de importación no tienen efecto hasta que se ejecuta el archivo. Es posible que desee ejecutar un archivo después de escribir las declaraciones de importación en la parte superior o ejecutar inmediatamente un archivo existente antes de editarlo.

Format block

Reformat Paragraph rewraps a block (“paragraph”) of contiguous equally indented non-blank comments, a similar block of text within a multiline string, or a selected subset of either. If needed, add a blank line to separate string from code. Partial lines in a selection expand to complete lines. The resulting lines have the same indent as before but have maximum total length of N columns (characters). Change the default N of 72 on the Window tab of IDLE Settings.

Contexto del código

Dentro de una ventana del editor que contiene código Python, el contexto del código se puede alternar para mostrar u ocultar un panel en la parte superior de la ventana. Cuando se muestra, este panel congela las líneas de apertura por bloques de código, como aquellos que comienzan con las palabras clave `class`, `def`, o `if`, que de otro modo se habrían desplazado fuera de la vista. El tamaño del panel se expandirá y contraerá según sea necesario para mostrar todos los niveles actuales de contexto, hasta el número máximo de líneas definidas en el cuadro de diálogo Configurar IDLE (que por defecto es 15). Si no hay líneas de contexto actuales y la función está activada, se visualizará una sola línea en blanco. Al hacer click en una línea en el panel de contexto, esa línea se moverá a la parte superior del editor.

El texto y los colores de fondo para el panel de contexto se pueden configurar en la pestaña destacados en el cuadro de diálogo Configurar IDLE.

Ventana de Shell

Con el Shell del IDLE, se ingresa, edita y recupera declaraciones completas. (La mayoría de consolas y terminales solo funcionan con una sola línea física a la vez).

Envía una declaración de una sola línea para su ejecución al presionar `Return` con el cursor en cualquier parte de la línea. Si una línea se extiende con una barra invertida (`\`), el cursor debe estar en la última línea física. Envía una declaración compuesta de varias líneas al ingresar una línea vacía después de la declaración.

Cuando se pega código en el Shell, este no se compila y posiblemente se ejecuta hasta que se teclea `Return`, como se especifica arriba. Se puede editar el código pegado primero. Si se pega más de una declaración en el Shell, el resultado será un **`SyntaxError`** cuando se compilan varias declaraciones como si fueran una sola.

Las líneas que contienen `RESTART` significan que el proceso de ejecución del usuario se ha reiniciado. Esto ocurre cuando falla el proceso de ejecución del usuario, solicita un reinicio en el menú del Shell o ejecuta código en una ventana del editor.

The editing features described in previous subsections work when entering code interactively. IDLE’s Shell window also responds to the following:

- `C-c` attempts to interrupt statement execution (but may fail).
- `C-d` closes Shell if typed at a `>>>` prompt.
- `Alt-p` and `Alt-n` (`C-p` and `C-n` on macOS) retrieve to the current prompt the previous or next previously entered statement that matches anything already typed.
- `Return` while the cursor is on any previous statement appends the latter to anything already typed at the prompt.

Colores del texto

El idle por defecto es negro sobre texto blanco, pero colorea el texto con significados especiales. Para el shell, estos son: la salida del shell, error del shell, salida del usuario y error del usuario. Para el código Python, en el indicador de comandos de shell o en un editor, estos son: palabras clave, nombres de funciones y clases incorporadas, los siguientes nombres `class` and `def`, cadenas de caracteres y comentarios. Para cualquier ventana de texto, estos son: el cursor (cuando está presente), el texto encontrado (cuando sea posible) y el texto seleccionado.

IDLE también resalta las palabras claves ligeras **`match`**, **`case`**, y **`_`** en sentencias de patrones coincidentes. Sin embargo, este resaltado no es perfecto y será incorrecto en algunos raras ocasiones, incluyendo algunos `_`-s en patrones `case`.

La coloración del texto se realiza en segundo plano, por lo que ocasionalmente se puede ver el texto sin colorear. Para cambiar la combinación de colores, use la pestaña Resaltar en el cuadro de diálogo Configurar IDLE. El marcado de líneas de breakpoint del depurador en el editor y el texto en ventanas emergentes y cuadros de diálogo no es configurable por el usuario.

Inicio y ejecución de código

Al iniciar con la opción `-s`, el IDLE ejecutará el archivo al que hacen referencia las variables de entorno **`IDLESTARTUP`** o **`PYTHONSTARTUP`**. El IDLE primero verifica `IDLESTARTUP`; si `IDLESTARTUP` está presente, se ejecuta el archivo al que se hace referencia. Si `IDLESTARTUP` no está presente, IDLE verifica `PYTHONSTARTUP`. Los archivos referenciados por estas variables de entorno son lugares convenientes para almacenar funciones que son usadas con frecuencia desde el shell del IDLE o para ejecutar declaraciones importadas para importar módulos comunes.

Además, `Tk` también carga un archivo de inicio si este está presente. Tenga en cuenta que el archivo Tk se carga incondicionalmente. Este archivo adicional es `.Idle.py` y es buscado en el directorio de inicio del usuario. Las declaraciones en este archivo se ejecutarán en el espacio de nombres Tk, por lo que este archivo no es útil para importar funciones que se utilizarán

desde el shell de Python del IDLE.

Command-line usage

IDLE can be invoked from the command line with various options. The general syntax is:

```
python -m idlelib [options] [file ...]
```

The following options are available:

-c <command>

Run the specified Python command in the shell window. For example, pass `-c "print('Hello, World!')"`. On Windows, the outer quotes must be double quotes as shown.

-d

Enable the debugger and open the shell window.

-e

Open an editor window.

-h

Print a help message with legal combinations of options and exit.

-i

Open a shell window.

-r <file>

Run the specified file in the shell window.

-s

Run the startup file (as defined by the environment variables **`IDLESTARTUP`** or **`PYTHONSTARTUP`**) before opening the shell window.

-t <title>

Set the title of the shell window.

-

Read and execute standard input in the shell window. This option must be the last one before any arguments.

If arguments are provided:

- If `-`, `-c`, or `-r` is used, all arguments are placed in `sys.argv[1:]`, and `sys.argv[0]` is set to `''`, `'-c'`, or `'-r'` respectively. No editor window is opened, even if that is the default set in the *Options* dialog.
- Otherwise, arguments are treated as files to be opened for editing, and `sys.argv` reflects the arguments passed to IDLE itself.

Error de inicio

El IDLE usa un socket para comunicarse entre el proceso del GUI de IDLE y el proceso de ejecución del código de usuario. Se debe establecer una conexión cada vez que el Shell se inicia o reinicia. (Esto último se indica mediante una línea divisoria que dice “REINICIAR” (*RESTART*)). Si el proceso del usuario no se conecta al proceso de la GUI, muestra un cuadro de error `Tk` con un mensaje de “no se puede conectar” que dirige al usuario aquí. Y después se detiene.

Una falla de conexión específica en sistemas Unix es el resultado de una mala configuración de las reglas de enmascarado en algún lugar de la configuración del sistema de red. Cuando el IDLE inicia desde la terminal, puede verse un mensaje que comienza con `` Invalid host:. ``` The valid value is ``127.0.0.1 (idlelib.rpc.LOCALHOST)``**. Se puede diagnosticar con ``tcpconnect -irv 127.0.0.1 6543 ``` en una ventana de la terminal y `tcplisten <same args>` en otra.

Una causa común de falla se da cuando un archivo escrito por el usuario tiene el mismo nombre de un módulo de biblioteca estándar, como *random.py* y *tkinter.py*. Cuando dicho archivo se encuentra en el mismo directorio que un archivo que está por ejecutarse, el IDLE no puede importar el archivo stdlib. La solución actual es cambiar el nombre del archivo del usuario.

Aunque es menos común que en el pasado, un programa de antivirus o firewall puede detener la conexión. Si el programa no se puede configurar para permitir la conexión, debe estar apagado para que funcione el IDLE. Es seguro permitir esta conexión interna porque no hay datos visibles en puertos externos. Un problema similar es una configuración incorrecta de la red que bloquea las conexiones.

Los problemas de instalación de Python ocasionalmente detienen el IDLE: puede darse un conflicto entre versiones o una sola instalación puede requerir privilegios de administrador. Si se soluciona el conflicto o no se puede o quiere ejecutar como administrador, puede ser más fácil desinstalar completamente Python y comenzar de nuevo.

Un proceso zombie pythonw.exe podría ser un problema. En Windows, use el Administrador de tareas para buscar uno y detenerlo si lo hay. A veces, un reinicio iniciado por un bloqueo del programa o una interrupción del teclado (control-C) puede fallar al conectarse. Descartar el cuadro de error o usar Reiniciar Shell en el menú del Shell puede solucionar un problema temporal.

Cuando el IDLE se inicia por primera vez, intenta leer los archivos de configuración de usuario en `~/.idlerc/` (~ este es el directorio principal). Si hay un problema, se mostrará un mensaje de error. Dejando de lado los fallos aleatorios del disco, esto se puede evitar si nunca se editan los archivos a mano. En su lugar, utilice el cuadro de diálogo de configuración, en Opciones. Una vez que hay un error en un archivo de configuración de usuario, la mejor solución puede ser eliminarlo y empezar de nuevo con el cuadro de diálogo de configuración.

Si el IDLE se cierra sin mensaje y este no fue iniciado desde una consola, intente iniciarlo desde una consola o terminal (`python -m idlelib`) y observe si esto genera un mensaje de error.

En sistemas basados en Unix con tcl/tk más antiguos que `8.6.11` (ver `About IDLE`) ciertos caracteres de ciertas fuentes pueden causar un fallo a tk con un mensaje hacia la terminal. Esto puede pasar cuando se inicia el IDLE para editar un archivo con dicho carácter o luego cuando se ingresa ese carácter. Si no se puede actualizar tcl/tk, entonces re configurar el IDLE para usar una fuente que funcione mejor.

Ejecutando código del usuario

Con raras excepciones, el resultado de ejecutar el código de Python con el IDLE se supone es el mismo que al ejecutar el mismo código por el método predeterminado, directamente con Python en una consola de sistema en modo de texto o una ventana de terminal. Sin embargo, las diferentes interfaces y operaciones ocasionalmente afectan los resultados visibles. Por ejemplo, `sys.modules` comienza con más entradas y `threading.activeCount()` retorna 2 en lugar de 1.

De forma predeterminada, el IDLE ejecuta el código de usuario en un proceso separado del sistema operativo en lugar de hacerlo en el proceso de la interfaz de usuario que ejecuta el shell y el editor. En el proceso de ejecución, este reemplaza `sys.stdin`, `sys.stdout`, y `sys.stderr` con objetos que recuperan las entradas desde y envían las salidas hacia la ventana de la consola. Los valores originales almacenados en `sys.__stdin__`, `sys.__stdout__`, y `sys.__stderr__` no se ven afectados, pero pueden ser `None`.

El envío de la salida de impresión de un proceso a un widget de texto en otro es más lento que la impresión a un terminal del sistema en el mismo proceso. Esto tiene el mayor efecto cuando se imprimen múltiples argumentos, ya que la cadena de cada argumento, cada separador y la nueva línea se envían por separado. Para el desarrollo, esto no suele ser un problema, pero si uno quiere imprimir más rápido en IDLE, formatea y une todo lo que quiere mostrar junto y luego imprime una sola cadena. Tanto las cadenas de formato como **`str.join()`** pueden ayudar a combinar campos y líneas.

Las sustituciones de flujo estándar del IDLE no son heredadas por subprocesos creados en el proceso de ejecución, siendo directamente por código de usuario o por módulos como el multiprocesamiento. Si tal subproceso usa `input` desde sys.stdin o `print` o `write` hacia sys.stdout o sys.stderr, el IDLE debe ser iniciado en una ventana de línea de comando. (En Windows, utilice `python` o `py` en lugar de `pythonw` o `pyw`.) El subproceso secundario será adjuntado a esa ventana para entrada y salida.

Si `sys` es restablecido mediante un código de usuario, tal como `importlib.reload(sys)`, los cambios del IDLE se perderán y la entrada del teclado y la salida de la pantalla no funcionarán correctamente.

Cuando el shell está en primer plano, controla el teclado y la pantalla. Este es transparente normalmente, pero las funciones que acceden directamente al teclado y la pantalla no funcionarán. Estas incluyen funciones específicas del sistema que determinan si se ha presionado una tecla y de ser así, cuál.

La ejecución de código del IDLE en el proceso de ejecución agrega marcos a la pila de llamadas que de otro modo no estarían allí. el IDLE encapsula `sys.getrecursionlimit` y `sys.setrecursionlimit` para reducir el efecto de los marcos de pila adicionales.

Cuando el código de usuario genera SystemExit directamente o llamando a sys.exit, el IDLE regresa al visualizador del Shell en lugar de salir.

Salida del usuario en consola

Cuando un programa muestra texto, el resultado está determinado por el dispositivo de salida correspondiente. Cuando el IDLE ejecuta el código de usuario `sys.stdout` y `sys.stderr` se conectan al área de visualización del Shell del IDLE. Algunas de estas características son heredadas de los widgets de texto Tk subyacentes. Otras son adiciones programadas. Donde es importante, el shell está diseñado para el desarrollo en lugar de la ejecución de producción.

Por ejemplo, el Shell nunca elimina la salida. Un programa que envía salidas ilimitadas al Shell eventualmente llenará la memoria, lo que resultará en un error de memoria. Por el contrario, algunas ventanas de texto del sistema solo conservan las últimas n líneas de salida. Una consola de Windows, por ejemplo, mantiene una línea configurable por el usuario de 1 a 9999, con 300 por defecto.

Un widget de texto de Tk, y por lo tanto el Shell de IDLE, muestra caracteres (puntos de código) en el subconjunto BMP (plano multilingual básico) de Unicode. Aquellos caracteres que se muestran con un glifo adecuado y los cuales con una caja de reemplazo depende del sistema operativo y las fuentes instaladas. Los caracteres de tabulación hacen que el texto siguiente comience después de la siguiente tabulación. (Ocurre cada 8 “caracteres”). Los caracteres de nueva línea hacen que el texto siguiente aparezca en una nueva línea. Otros caracteres de control se omiten o se muestran como un espacio, cuadro u otra cosa, dependiendo del sistema operativo y la fuente. (Mover el cursor del texto a través de esa salida con las teclas de flecha puede mostrar algún comportamiento de espaciado sorpresivo.)

```
>>> s = 'a\tb\a<\x02><\r>\bc\nd'  # Enter 22 chars.
>>> len(s)
14
>>> s  # Display repr(s)
'a\tb\x07<\x02><\r>\x08c\nd'
>>> print(s, end='')  # Display s as is.
# Result varies by OS and font.  Try it.
```

La función `repr` es usada para la visualización interactiva del valor de las expresiones. Esta retorna una versión modificada de la cadena de caracteres de entrada en la que los códigos de control, algunos puntos de código BMP y todos los puntos de código que no son BMP son reemplazados con caracteres de escape. Como se demostró anteriormente, esto permite identificar los caracteres en una cadena de caracteres, independientemente de cómo sean mostrados.

La salida normal y de error generalmente se mantienen separadas (en líneas separadas) de la entrada de código y entre sí. Cada una obtiene diferentes colores de resaltado.

Para el *traceback* de SyntaxError, el marcado normal "^" dónde se detectó el error se reemplaza coloreando el texto con un resaltado de error. Cuando el código ejecutado desde un archivo causa otras excepciones, se puede hacer click derecho en un *traceback* para saltar a la línea correspondiente en un editor del IDLE. El archivo se abrirá si es necesario.

El Shell tiene una funcionalidad especial para exprimir las líneas de salida hasta una etiqueta de "Texto *Squeezed*". Esto se hace automáticamente para una salida sobre N líneas (N = 50 por defecto). N se puede cambiar en la sección PyShell de la página General del cuadro de diálogo Configuración. La salida con menos líneas puede ser *squeezed* haciendo click derecho en la salida. Esto puede ser útil en líneas suficientemente largas para bajar el tiempo de desplazamiento.

La salida *squeezed* se expande en su lugar haciendo doble click en la etiqueta. También se puede enviar al portapapeles o a una ventana de vista separada haciendo click derecho en la etiqueta.

Desarrollando aplicaciones tkinter

El IDLE es intencionalmente diferente de Python estándar para facilitar el desarrollo de programas tkinter. Ingrese `import tkinter as tk; root = tk.Tk()` en Python estándar y no aparecerá nada. Ingrese lo mismo en el IDLE y aparecerá una ventana tk. En Python estándar, también se debe ingresar `root.update()` para ver la ventana. El IDLE hace el equivalente en segundo plano, aproximadamente 20 veces por segundo, lo que es aproximadamente cada 50 milisegundos. Luego ingrese `b = tk.Button(root, text='button'); b.pack()`. Del mismo modo, no hay cambios visibles en Python estándar hasta que ingrese `root.update()`.

La mayoría de los programas tkinter ejecutan `root.mainloop()`, que generalmente no retorna hasta que se destruye la aplicación tk. Si el programa se ejecuta con `python -i` o desde un editor del IDLE, no aparecerá un mensaje `>>>` del shell hasta que retorne `mainloop()`, momento en el cual no queda nada con lo que interactuar.

Al ejecutar un programa tkinter desde un editor IDLE, se puede comentar la llamada de bucle principal. Luego se recibe un aviso del shell inmediatamente y se puede interactuar con la aplicación en vivo. Solo se debe recordar reactivar la llamada al bucle principal cuando se ejecuta en Python estándar.

Ejecutando sin un subproceso

Por defecto, el IDLE ejecuta el código de usuario en un subproceso separado a través de un socket, el cual utiliza la interfaz de bucle interno. Esta conexión no es visible externamente y no son enviados ni recibidos datos de Internet. Si el software de cortafuego sigue presentando problemas, puede ignorarlo.

Si el intento de conexión del socket falla, IDLE lo notificará. Este tipo de falla a veces es temporal, pero si persiste, el problema puede provenir de un cortafuego que bloquea la conexión o de una mala configuración en un sistema en particular. Hasta que se solucione el problema, se puede ejecutar el Idle con el modificador de línea de comandos -n.

Si el IDLE se inicia con el modificador de línea de comandos -n, se ejecutará en un único proceso y no creará el subproceso que ejecuta el servidor de ejecución de Python RPC. Esto puede ser útil si Python no puede crear el subproceso o la interfaz de socket RPC en su plataforma. Sin embargo, en este modo el código de usuario no está aislado en sí del IDLE. Además, el entorno no se reinicia cuando se selecciona Ejecutar/Ejecutar módulo (F5). Si el código se ha modificado, se debe volver a cargar() los módulos afectados y volver a importar cualquier elemento específico (por ejemplo, desde *foo* importar *baz*) para que los cambios surtan efecto. Por estas razones, es preferible ejecutar el IDLE con el subproceso predeterminado si es posible.

Obsoleto desde la versión 3.4.

Ayuda y preferencias

Recursos de ayuda

La entrada del menú Ayuda «Ayuda IDLE» muestra una versión HTML formateada del capítulo IDLE de la Referencia de la biblioteca. El resultado, en una ventana de texto tkinter de solo lectura, está cerca de lo que se ve en un navegador web. Navegue por el texto con la rueda del ratón, la barra de desplazamiento o presionando las teclas de flecha arriba y abajo. O haga click en el botón TDC (Tabla de contenido) y seleccione un encabezado de sección en el cuadro abierto.

La entrada del menú de ayuda «Documentos de Python» abre amplias fuentes de ayuda, incluyendo tutoriales, disponibles en `docs.python.org/x.y`, donde "x.y" es la versión de Python actualmente en ejecución. Si su sistema tiene una copia fuera de línea de los documentos (esta puede ser una opción de instalación), esta se abrirá en su lugar.

Las URL seleccionadas se pueden agregar o eliminar del menú de ayuda en cualquier momento utilizando la pestaña General del cuadro de diálogo Configurar IDLE.

Preferencias de configuración

Las preferencias de fuente, resaltado, teclas y preferencias generales se pueden cambiar en Configurar IDLE en el menú opción. Las configuraciones de usuario no predeterminadas se guardan en un directorio `.idlerc` en el directorio de inicio del usuario. Los problemas causados por archivos de configuración de usuario incorrectos se resuelven editando o eliminando uno o más de los archivos en `.idlerc`.

En la pestaña Fuentes (*Font*), consulte la muestra de texto para ver el efecto de la fuente y el tamaño de la fuente en varios caracteres en varios idiomas. Edite la muestra para agregar otros caracteres de interés personal. Use la muestra para seleccionar fuentes monoespaciadas. Si determinados caracteres tienen problemas en el shell o en el editor, agréguelos a la parte inicial de la muestra e intente cambiar primero el tamaño y luego la fuente.

En la pestaña Resaltado y Teclas (*Highlights and Keys*), seleccione un tema de color integrado o personalizado y un conjunto de teclas. Para utilizar un nuevo tema de color integrado o un conjunto de teclas con IDLE más antiguos, guárdelo como un nuevo tema personalizado o conjunto de teclas y será accesible para los IDLE más antiguos.

IDLE en macOS

En Preferencias del sistema: Dock, puede establecer «Preferencias de pestañas al abrir documentos» en el valor «Siempre». Este parámetro no es compatible con la interfaz gráfica de usuario del framework tk/tkinter utilizado por IDLE y rompe algunas funcionalidades del IDLE.

Extensiones

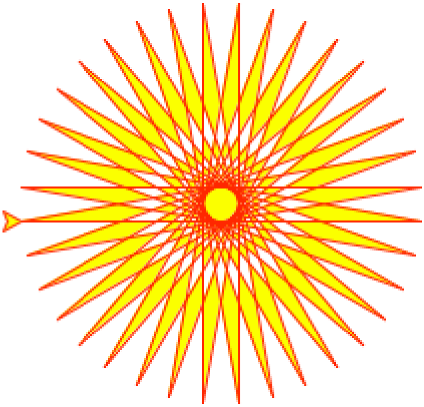
IDLE incluye una herramienta de extensiones. Las preferencias para las extensiones se pueden cambiar con la pestaña Extensiones de la ventana de preferencias. Lea el inicio de config-extensions.def en la carpeta idlelib para obtener más información. La única extensión utilizada actualmente por defecto es zzdummy, un ejemplo que también se utiliza para realizar pruebas.

idlelib — implementation of IDLE application

Código fuente: Lib/idlelib [https://github.com/python/cpython/tree/3.14/Lib/idlelib]

El paquete Lib/idlelib implementa la aplicación del IDLE. Consultar el resto de esta página para saber cómo utilizar el IDLE.

Los archivos en idlelib se describen en idlelib/README.txt. Accede a él en idlelib o haga clic en Ayuda = > Acerca del IDLE en el menú del IDLE. Este archivo también asigna elementos del menú del IDLE al código que implementa el elemento. Excepto para archivos enumerados en “Inicio”, el código de idlelib es “privado” en el sentido de que los cambios de características pueden ser portados (consultar **PEP 434** [https://peps.python.org/pep-0434/]).



`turtle` — Turtle graphics

Código fuente: Lib/turtle.py [https://github.com/python/cpython/tree/3.14/Lib/turtle.py]

Introducción

Los gráficos de tortuga son una implementación de the popular geometric drawing tools introduced in Logo [https://en.wikipedia.org/wiki/Turtle_(robot)], desarrollado por Wally Feurzeig, Seymour Papert y Cynthia Solomon en 1967.

This is an optional module. If it is missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

Empezar

Imagine una tortuga robótica que comienza en (0, 0) en el plano x-y. Después de un `import turtle`, dale el comando `turtle.forward(15)` y se moverá (¡en la pantalla!) 15 píxeles en la dirección en la que está mirando, dibujando una línea a medida que se mueve. Dale el comando `turtle.right(25)` y rotará en su lugar 25 grados en el sentido de las agujas del reloj.

Turtle star

Turtle puede dibujar figuras intrincadas usando programas que repiten movimientos simples.

En Python, los gráficos de tortuga proporcionan una representación de una «tortuga» física (un pequeño robot con un bolígrafo) que dibuja en una hoja de papel en el suelo.

Es una forma eficaz y probada para que los estudiantes conozcan los conceptos de programación y la interacción con el software, ya que proporciona una retroalimentación instantánea y visible. También proporciona un acceso cómodo a la salida gráfica en general.

El dibujo de tortugas se creó originalmente como una herramienta educativa para que la utilizaran los profesores en el aula. Para el programador que necesita producir algún resultado gráfico, puede ser una forma de hacerlo sin la sobrecarga que supone introducir bibliotecas externas o más complejas en su trabajo.

Tutorial

Los nuevos usuarios deberían empezar aquí. En este tutorial, exploraremos algunos de los conceptos básicos del dibujo de tortugas.

Iniciando un entorno para tortugas

En un shell de Python, importe todos los objetos del módulo `turtle`:

```
from turtle import *
```

Si se encuentra con un error `No module named '_tkinter'`, tendrá que instalar **`Tk interface package`** en su sistema.

Dibujo básico

Envía la tortuga hacia adelante 100 pasos:

```
forward(100)
```

Deberías ver (probablemente en una nueva ventana de tu pantalla) una línea dibujada por la tortuga, en dirección este. Cambia la dirección de la tortuga para que gire 120 grados hacia la izquierda (en sentido contrario a las agujas del reloj):

```
left(120)
```

Continuemos dibujando un triángulo:

```
forward(100)
left(120)
forward(100)
```

Observa cómo la tortuga, representada por una flecha, apunta en diferentes direcciones a medida que la diriges.

Experimente con esos comandos y también con `backward()` y `right()`.

Control del lápiz

Intente cambiar el color (por ejemplo, `color('blue')`) y el ancho de la línea (por ejemplo, `width(3)`) y luego vuelva a dibujar.

También puedes mover la tortuga sin dibujar, levantando el lápiz: `up()` antes de moverla. Para comenzar a dibujar nuevamente, usa `down()`.

La posición de la tortuga

Envía a tu tortuga de regreso a su punto de partida (útil si ha desaparecido de la pantalla):

```
home()
```

La posición inicial está en el centro de la pantalla de la tortuga. Si alguna vez necesitas saberlas, obtén las coordenadas x-y de la tortuga con:

```
pos()
```

El hogar está en `(0, 0)`.

Y después de un tiempo, probablemente ayudará a limpiar la ventana para que podamos comenzar de nuevo:

```
clearscreen()
```

Creando patrones algorítmicos

Utilizando bucles, es posible construir patrones geométricos:

```
for steps in range(100):
    for c in ('blue', 'red', 'green'):
        color(c)
        forward(steps)
        right(30)
```

- ¡que, por supuesto, están limitadas únicamente por la imaginación!

Dibujemos la forma de estrella en la parte superior de esta página. Queremos líneas rojas, rellenas con amarillo:

```
color('red')
fillcolor('yellow')
```

Así como `up()` y `down()` determinan si se dibujarán líneas, el relleno se puede activar y desactivar:

```
begin_fill()
```

A continuación crearemos un bucle:

```
while True:
    forward(200)
    left(170)
    if abs(pos()) < 1:
        break
```

`abs(pos()) < 1` es una buena manera de saber cuándo la tortuga ha regresado a su posición original.

Por último, completa el relleno:

```
end_fill()
```

(Tenga en cuenta que el llenado solo se realiza cuando se da el comando `end_fill()`).

Cómo…

Esta sección cubre algunos casos de uso y enfoques típicos de las tortugas.

Empiece lo antes posible

Una de las ventajas de los gráficos de tortuga es la respuesta visual inmediata que se obtiene con comandos simples: es una forma excelente de introducir a los niños a las ideas de programación, con un mínimo de gastos generales (no solo para los niños, por supuesto).

El módulo Turtle hace posible esto al exponer toda su funcionalidad básica como funciones, disponibles con `from turtle import *`. turtle graphics tutorial cubre este enfoque.

Vale la pena señalar que muchos de los comandos de la tortuga también tienen equivalentes aún más concisos, como `fd()` para **`forward()`**. Estos son especialmente útiles cuando se trabaja con estudiantes para quienes escribir a máquina no es una habilidad.

> Necesitará tener instalado **`Tk interface package`** en su sistema para que funcionen los gráficos de Turtle. Tenga en cuenta que esto no siempre es sencillo, por lo que debe verificar esto

con anticipación si planea usar gráficos de Turtle con un estudiante.

Automatically begin and end filling

Starting with Python 3.14, you can use the **`fill()`** context manager instead of **`begin_fill()`** and **`end_fill()`** to automatically begin and end fill. Here is an example:

```
with fill():
    for i in range(4):
        forward(100)
        right(90)

forward(200)
```

The code above is equivalent to:

```
begin_fill()
for i in range(4):
    forward(100)
    right(90)
end_fill()

forward(200)
```

Utilice el espacio de nombres del módulo `turtle`

Usar `from turtle import *` es conveniente, pero tenga cuidado, ya que importa una colección bastante grande de objetos y, si está haciendo otra cosa que no sean gráficos de tortuga, corre el riesgo de un conflicto de nombres (esto se vuelve aún más problemático si está usando gráficos de tortuga en un script donde se pueden importar otros módulos).

La solución es utilizar `import turtle`: `fd()` se convierte en `turtle.fd()`, `width()` se convierte en `turtle.width()` y así sucesivamente. (Si escribir «tortuga» una y otra vez se vuelve tedioso, utilice, por ejemplo, `import turtle as t` en su lugar).

Utilice gráficos de tortugas en un guión

Se recomienda utilizar el espacio de nombres del módulo `turtle` como se describe inmediatamente arriba, por ejemplo:

```
import turtle as t
from random import random

for i in range(100):
    steps = int(random() * 100)
    angle = int(random() * 360)
    t.right(angle)
    t.fd(steps)
```

Sin embargo, también se requiere otro paso: tan pronto como finalice el script, Python también cerrará la ventana de la tortuga. Agregar:

```
t.mainloop()
```

hasta el final del script. El script ahora esperará a que lo cierren y no saldrá hasta que lo hagan, por ejemplo, cerrando la ventana de gráficos de la tortuga.

Utilice gráficos de tortuga orientados a objetos

Ver también

Explanation of the object-oriented interface

Aparte de para propósitos introductorios muy básicos o para probar cosas lo más rápido posible, es más habitual y mucho más potente utilizar el enfoque orientado a objetos para los gráficos de tortugas. Por ejemplo, esto permite tener varias tortugas en la pantalla a la vez.

En este enfoque, los distintos comandos de la tortuga son métodos de objetos (en su mayoría objetos `Turtle`). *can* utiliza el enfoque orientado a objetos en el shell, pero sería más típico en un script de

Python.

El ejemplo anterior se convierte entonces en:

```
from turtle import Turtle
from random import random

t = Turtle()
for i in range(100):
    steps = int(random() * 100)
    angle = int(random() * 360)
    t.right(angle)
    t.fd(steps)

t.screen.mainloop()
```

Tenga en cuenta la última línea. `t.screen` es una instancia de **`Screen`** en la que existe una instancia de Turtle; se crea automáticamente junto con la tortuga.

La pantalla de la tortuga se puede personalizar, por ejemplo:

```
t.screen.title('Object-oriented turtle demo')
t.screen.bgcolor("orange")
```

Referencia gráficos de tortugas

Nota

En la siguiente documentación se proporciona la listas de argumentos para las funciones. Los métodos, por su puesto, tienen el argumento principal adicional *self* que se omite aquí.

Métodos Turtle

Movimiento de Turtle

Mover y dibujar

    - `forward()` | `fd()`
    - `backward()` | `bk()` | `back()`
    - `right()` | `rt()`
    - `left()` | `lt()`
    - `goto()` | `setpos()` | `setposition()`
    - `teleport()`
    - `setx()`
    - `sety()`
    - `setheading()` | `seth()`
    - `home()`
    - `circle()`
    - `dot()`
    - `stamp()`
    - `clearstamp()`
    - `clearstamps()`
    - `undo()`
    - `speed()`

  - Mostrar el estado de la tortuga

    - `position()` | `pos()`
    - `towards()`
    - `xcor()`
    - `ycor()`
    - `heading()`
    - `distance()`

  - Ajuste y unidades de medida

    - `degrees()`
    - `radians()`

- Control del lápiz

  - Estado de dibujo

    - `pendown()` | `pd()` | `down()`
    - `penup()` | `pu()` | `up()`
    - `pensize()` | `width()`
    - `pen()`
    - `isdown()`

Control del color

- `color()`
- `pencolor()`
- `fillcolor()`

Relleno

- `filling()`
- `fill()`
- `begin_fill()`
- `end_fill()`

Más controles de dibujo

- `reset()`
- `clear()`
- `write()`

Estado de la Tortuga

Visibilidad

- `showturtle()` | `st()`
- `hideturtle()` | `ht()`
- `isvisible()`

Apariencia

- `shape()`
- `resizemode()`
- `shapesize()` | `turtlesize()`
- `shearfactor()`
- `tiltangle()`
- `tilt()`
- `shapetransform()`
- `get_shapepoly()`

Usando eventos

- `onclick()`

`onrelease()`
`ondrag()`

Métodos especiales de *Turtle*

`poly()`
`begin_poly()`
`end_poly()`
`get_poly()`
`clone()`
`getturtle()` | `getpen()`
`getscreen()`
`setundobuffer()`
`undobufferentries()`

Métodos de TurtleScreen/Screen

Control de ventana

`bgcolor()`
`bgpic()`
`clearscreen()`
`resetscreen()`
`screensize()`
`setworldcoordinates()`

Control de animación

`no_animation()`
`delay()`
`tracer()`
`update()`

Usando eventos de pantalla

`listen()`
`onkey()` | `onkeyrelease()`
`onkeypress()`
`onclick()` | `onscreenclick()`
`ontimer()`
`mainloop()` | `done()`

Configuración y métodos especiales

- `mode()`
- `colormode()`
- `getcanvas()`
- `getshapes()`
- `register_shape()` | `addshape()`
- `turtles()`
- `window_height()`
- `window_width()`

Métodos de entrada

- `textinput()`
- `numinput()`

Métodos específicos para *Screen*

- `bye()`
- `exitonclick()`
- `save()`
- `setup()`
- `title()`

Métodos de *RawTurtle/Turtle* Y sus correspondientes funciones

Casi todos los ejemplos de esta sección se refieren a una instancia Turtle llamada `turtle`.

Movimiento de Turtle

turtle.forward(*distance*)
turtle.fd(*distance*)

Parámetros:

distance – un número (entero o flotante)

Mover hacia adelante la tortuga la *ditance* especificada, en la dirección en la que la tortuga apunta.

```
>>> turtle.position()
(0.00,0.00)
>>> turtle.forward(25)
>>> turtle.position()
(25.00,0.00)
>>> turtle.forward(-75)
>>> turtle.position()
(-50.00,0.00)
```

turtle.back(*distance*)
turtle.bk(*distance*)
turtle.backward(*distance*)

Parámetros:

distance – un número

Mover hacia atrás la tortuga la *distance* especificada, opuesta a la dirección en que la tortuga apunta. No cambia la dirección de la tortuga.

```
>>> turtle.position()
(0.00,0.00)
>>> turtle.backward(30)
>>> turtle.position()
(-30.00,0.00)
```

turtle.right(*angle*)
turtle.rt(*angle*)

Parámetros:

angle – un número (entero o flotante)

Gira la tortuga a la derecha tomando los *angle* como unidad de medida. (La unidad de medida por defecto son los grado, pero se puede configurar a través de las funciones `degrees()` y `radians()`.) La orientación de los ángulos depende del modo en que está la tortuga, ver `mode()`.

```
>>> turtle.heading()
22.0
>>> turtle.right(45)
>>> turtle.heading()
337.0
```

turtle.left(*angle*)
turtle.lt(*angle*)

Parámetros:

angle – un número (entero o flotante)

Gira la tortuga a la izquierda tomando los *ángulos* como unidad de medida. (La unidad de medida por defecto son los grado, pero se puede configurar a través de las funciones `degrees()` y `radians()`.) La orientación de los ángulos depende del modo en que está la tortuga, ver `mode()`.

```
>>> turtle.heading()
22.0
>>> turtle.left(45)
>>> turtle.heading()
67.0
```

turtle.goto(*x*, *y=None*)
turtle.setpos(*x*, *y=None*)
turtle.setposition(*x*, *y=None*)

Parámetros:

- **x** – un número o un par/vector de números
- **y** – un número o `None`

Si *y* es `None`, *x* debe ser un par de coordenadas o un `Vec2D` (ejemplo: según lo devuelto por `pos()`).

Mueve la tortuga a una posición absoluta. Si el lápiz está bajo, traza una linea. No cambia la orientación de la tortuga.

```
>>> tp = turtle.pos()
>>> tp
(0.00,0.00)
>>> turtle.setpos(60,30)
>>> turtle.pos()
(60.00,30.00)
>>> turtle.setpos((20,80))
>>> turtle.pos()
(20.00,80.00)
>>> turtle.setpos(tp)
>>> turtle.pos()
(0.00,0.00)
```

turtle.teleport(*x*, *y=None*, ***, *fill_gap=False*)

Parámetros:

- **x** – un número o `None`
- **y** – un número o `None`
- **fill_gap** – un booleano

Mueve la tortuga a una posición absoluta. A diferencia de goto(x, y), no se dibujará una línea. La orientación de la tortuga no cambia. Si se está llenando, el polígono o polígonos desde los que se teletransporta se llenarán después de salir y el llenado comenzará nuevamente después de teletransportarse. Esto se puede desactivar con fill_gap=True, que hace que la línea imaginaria recorrida durante el teletransporte actúe como una barrera de relleno como en goto(x, y).

```
>>> tp = turtle.pos()
>>> tp
(0.00,0.00)
>>> turtle.teleport(60)
>>> turtle.pos()
(60.00,0.00)
>>> turtle.teleport(y=10)
>>> turtle.pos()
(60.00,10.00)
```

```
>>> turtle.teleport(20, 30)
>>> turtle.pos()
(20.00,30.00)
```

Added in version 3.12.

turtle.setx(*x*)

Parámetros:
x – un número (entero o flotante)

Establece la primera coordenada de la tortuga a *x*, deja la segunda coordenada sin cambios.

```
>>> turtle.position()
(0.00,240.00)
>>> turtle.setx(10)
>>> turtle.position()
(10.00,240.00)
```

turtle.sety(*y*)

Parámetros:
y – un número (entero o flotante)

Establece la segunda coordenada de la tortuga a *y*, deja la primera coordenada sin cambios.

```
>>> turtle.position()
(0.00,40.00)
>>> turtle.sety(-10)
>>> turtle.position()
(0.00,-10.00)
```

turtle.setheading(*to_angle*)
turtle.seth(*to_angle*)

Parámetros:
to_angle – un número (entero o flotante)

Establece la orientación de la tortuga a *to_angle*. Aquí hay algunas direcciones comunes en grados:

modo estándar	modo logo
0 - este	0 - norte
90 - norte	90 - este
180 - oeste	180 - sur
270 - sur	270 - oeste

```
>>> turtle.setheading(90)
>>> turtle.heading()
90.0
```

turtle.home()

Mueve la tortuga al origen – coordenadas (0,0) – y establece la orientación a la orientación original (que depende del modo, ver `mode()`).

```
>>> turtle.heading()
90.0
>>> turtle.position()
(0.00,-10.00)
>>> turtle.home()
>>> turtle.position()
(0.00,0.00)
>>> turtle.heading()
0.0
```

turtle.circle(*radius*, *extent=None*, *steps=None*)

Parámetros:

- **radius** – un número
- **extent** – un número (o `None`)
- **steps** – un entero (o `None`)

Dibuja un círculo con el *radius* (radio) dado. El centro es *radius* unidades a la izquierda de la tortuga; *extent* – un ángulo – determina que parte del círculo se dibuja. Si no se pasa *extent*, dibuja el círculo entero. Si *extent* no es un círculo completo, un punto final del arco es la posición actual de lápiz. Dibuja el arco en dirección antihorario si *radius* es positivo, si no en dirección horaria. Finalmente la dirección de la tortuga es modificada por el aumento de *extent*.

Como el círculo se aproxima a un polígono regular inscripto, *steps* determina el número de pasos a usar. Si no se da, será calculado automáticamente. Puede ser usado para dibujar polígonos regulares.

```
>>> turtle.home()
>>> turtle.position()
(0.00,0.00)
>>> turtle.heading()
0.0
>>> turtle.circle(50)
>>> turtle.position()
(-0.00,0.00)
>>> turtle.heading()
0.0
>>> turtle.circle(120, 180)  # draw a semicircle
>>> turtle.position()
(0.00,240.00)
>>> turtle.heading()
180.0
```

turtle.dot()
turtle.dot(*size*)
turtle.dot(*color, /*)
turtle.dot(*size, color, /*)
turtle.dot(*size, r, g, b, /*)

Parámetros:

- **size** – un entero > = 1 (si se da)
- **color** – un *colorstring* o una tupla numérica de color

Draw a circular dot with diameter *size*, using *color*. If *size* is not given, the maximum of `pensize+4` and `2*pensize` is used.

```
>>> turtle.home()
>>> turtle.dot()
>>> turtle.fd(50); turtle.dot(20, "blue"); turtle.f
>>> turtle.position()
(100.00,-0.00)
>>> turtle.heading()
0.0
```

turtle.stamp()

Estampa una copia de la forma de la tortuga en el lienzo en la posición actual. Devuelve un stamp_id por cada estampa, que puede ser usado para borrarlo al llamar `clearstamp(stamp_id)`.

```
>>> turtle.color("blue")
>>> stamp_id = turtle.stamp()
>>> turtle.fd(50)
```

turtle.clearstamp(*stampid*)

Parámetros:

stampid – un entero, debe devolver el valor de la llamada previa de la función `stamp()` call

Borra la estampa con el *stampid* dado.

```
>>> turtle.position()
(150.00,-0.00)
>>> turtle.color("blue")
>>> astamp = turtle.stamp()
>>> turtle.fd(50)
>>> turtle.position()
(200.00,-0.00)
>>> turtle.clearstamp(astamp)
>>> turtle.position()
(200.00,-0.00)
```

turtle.clearstamps(*n=None*)

Parámetros:

n – un entero (o `None`)

Borra todas o las primeros/últimos *n* estampas de la tortuga. Si *n* es `None` , borra todas las estampas, Si $n > 0$ borra la primera estampa *n*, sino y $n < 0$ borra la última estampa *n*.

```
>>> for i in range(8):
...     unused_stamp_id = turtle.stamp()
...     turtle.fd(30)
>>> turtle.clearstamps(2)
>>> turtle.clearstamps(-2)
>>> turtle.clearstamps()
```

turtle.undo()

Deshace (repetidamente) la(s) última(s) acción(es) de la tortuga. El número de acciones a deshacer es determinado por el tamaño del *undobuffer*.

```
>>> for i in range(4):
...     turtle.fd(50); turtle.lt(80)
...
>>> for i in range(8):
...     turtle.undo()
```

turtle.speed(*speed=None*)

Parámetros:

speed – un entero en el rango 0..10 o un *speedstring* (ver abajo)

Establecer la velocidad de la tortuga a un valor entero en el rango 0..10. Si no se pasa ningún argumento, vuelve a la velocidad actual.

Si el parámetro de entrada es un número mayor que 10 o menor que 0.5, la velocidad se establece a 0. La frase acerca de la velocidad es mapeada a valores de velocidad de la siguiente manera:

- «fastest»: 0
- «fast»: 10
- «normal»: 6
- «slow»: 3
- «slowest»: 1

Velocidades de 1 a 10 generan una animación cada vez más rápida al dibujar las líneas y en el giro de la tortuga.

Atención: *speed* = 0 implica que *no* habrá ninguna animación. Los métodos *fordward/back* harán que la tortuga salte, de la misma manera que *left/right* hará que la tortuga gire instantáneamente.

```
>>> turtle.speed()
3
>>> turtle.speed('normal')
>>> turtle.speed()
6
>>> turtle.speed(9)
>>> turtle.speed()
9
```

Mostrar el estado de la tortuga

turtle.position()
turtle.pos()

Devuelve la posición actual de la tortuga (x,y) (como un vector `Vec2D`)

```
>>> turtle.pos()
(440.00,-0.00)
```

turtle.towards(*x*, *y=None*)

Parámetros:

- **x** – un número o par de vectores numéricos o una instancia de la tortuga
- **y** – un número si *x* es un número, si no `None`

Retorna el ángulo entre la línea en la posición de la tortuga a la posición especificada en (x, y), el vector o la otra tortuga. Esto depende de la posición inicial de la tortuga, que depende del modo - «standard»/»world» o «logo».

```
>>> turtle.goto(10, 10)
>>> turtle.towards(0,0)
225.0
```

turtle.xcor()

Devuelve la coordinada *x* de la tortuga.

```
>>> turtle.home()
>>> turtle.left(50)
>>> turtle.forward(100)
>>> turtle.pos()
(64.28,76.60)
>>> print(round(turtle.xcor(), 5))
64.27876
```

turtle.ycor()

Devuelve la coordenada *y* de la tortuga.

```
>>> turtle.home()
>>> turtle.left(60)
>>> turtle.forward(100)
>>> print(turtle.pos())
(50.00,86.60)
>>> print(round(turtle.ycor(), 5))
86.60254
```

turtle.heading()

Devuelve la orientación actual de la tortuga (el valor depende del modo de la tortuga, ver `mode()`).

```
>>> turtle.home()
>>> turtle.left(67)
>>> turtle.heading()
67.0
```

turtle.distance(*x*, *y=None*)

Parámetros:

- **x** – un número o par de vectores numéricos o una instancia de la tortuga
- **y** – un número si *x* es un número, si no `None`

Devuelve la distancia desde la tortuga al vector (x,y) dado, otra instancia de la tortuga, el valor es unidad pasos de tortuga.

```
>>> turtle.home()
>>> turtle.distance(30,40)
50.0
>>> turtle.distance((30,40))
50.0
>>> joe = Turtle()
>>> joe.forward(77)
>>> turtle.distance(joe)
77.0
```

Configuración de las medidas

turtle.degrees(*fullcircle=360.0*)

Parámetros:

fullcircle – un número

Establece la unidad de medida del ángulo, por ejemplo establece el número de «grados» para un círculo completo. El valor por defecto es 36 grados.

```
>>> turtle.home()
>>> turtle.left(90)
>>> turtle.heading()
90.0

>>> # Change angle measurement unit to grad (also k
>>> # grade, or gradian and equals 1/100-th of the
>>> turtle.degrees(400.0)
>>> turtle.heading()
100.0
>>> turtle.degrees(360)
>>> turtle.heading()
90.0
```

turtle.radians()

Establece la unidad de medida del ángulo a radianes. Equivalente a `degrees(2*math.pi)`.

```
>>> turtle.home()
>>> turtle.left(90)
>>> turtle.heading()
90.0
>>> turtle.radians()
>>> turtle.heading()
1.5707963267948966
```

Control del lápiz

Estado de dibujo

turtle.pendown()
turtle.pd()
turtle.down()

Baja el lápiz – dibuja mientras se mueve.

turtle.penup()
turtle.pu()

turtle.up()

Levanta el lápiz – no dibuja mientras se mueve.

turtle.pensize(*width=None*)
turtle.width(*width=None*)

Parámetros:

width – un número positivo

Establece el grosos de la línea a *width* o lo devuelve. Si *resizemode* se establece a «auto» y turtleshape es un polígono, ese polígono es dibujado con el mismo grosor de línea. Si no se dan argumentos, devuelve el grosor del lápiz actual.

```
>>> turtle.pensize()
1
>>> turtle.pensize(10)   # de aquí en adelante se d
```

turtle.pen(*pen=None, **pendict*)

Parámetros:

- **pen** – un diccionario con algunos o todos las claves listadas debajo
- **pendict** – uno o más argumentos-palabras claves con las claves listadas debajo como palabras claves

Devuelve o establece los atributos del lápiz en un «diccionario-lápiz» con el siguiente para de valores/claves:

- «shown»: True/False
- «pendown»: True/False
- «pencolor»: color-string or color-tuple
- «fillcolor»: color-string or color-tuple
- «pensize»: número positivo
- «speed»: número en el rango 0..10
- «resizemode»: «auto» or «user» or «noresize»
- «stretchfactor»: (número positivo, número positivo)
- «outline»: número positivo

- «tilt»: número

Este diccionario puede usarse como argumento de una llamada subsecuente a la función `pen()` para restaurar el estado anterior del lápiz. Más aún, uno o más de estos atributos pueden darse como argumentos claves. Esto puede usarse para establecer diferentes atributos del lápiz en una sola definición.

```
>>> turtle.pen(fillcolor="black", pencolor="red", p
>>> sorted(turtle.pen().items())
[('fillcolor', 'black'), ('outline', 1), ('pencolor
 ('pendown', True), ('pensize', 10), ('resizemode',
 ('shearfactor', 0.0), ('shown', True), ('speed', 9
 ('stretchfactor', (1.0, 1.0)), ('tilt', 0.0)]
>>> penstate=turtle.pen()
>>> turtle.color("yellow", "")
>>> turtle.penup()
>>> sorted(turtle.pen().items())[:3]
[('fillcolor', ''), ('outline', 1), ('pencolor', 'y
>>> turtle.pen(penstate, fillcolor="green")
>>> sorted(turtle.pen().items())[:3]
[('fillcolor', 'green'), ('outline', 1), ('pencolor
```

turtle.isdown()

Devuelve `True` si el lápiz está abajo, si está arriba `False`.

```
>>> turtle.penup()
>>> turtle.isdown()
False
>>> turtle.pendown()
>>> turtle.isdown()
True
```

Control del color

turtle.pencolor()
turtle.pencolor(*color*, */*)

turtle.pencolor(*r, g, b, /*)

Devuelve o establece el color del lápiz.

Se permiten cuatro formatos de entrada:

`pencolor()`

Return the current pencolor as color specification string or as a tuple (see example). May be used as input to another color/pencolor/fillcolor/bgcolor call.

`pencolor(colorstring)`

Establece el color del lápiz a *colorstring*, que es una palabra que especifica un color *Tk*, tales como `"red"`, `"yellow"`, o `"#33cc8c"`.

`pencolor((r, g, b))`

Establece el color del lápiz representado como una tupla de *r*, *g*, y *b*. Cada valor *r*, *g*, y *b* debe ser un valor entero en el rango 0..colormode, donde colormode es 1.0 o 255 (ver **`colormode()`**).

`pencolor(r, g, b)`

Establece el color del lápiz al color RGB representado por *r*, *g*, y *b*. Cada valor *r*, *g*, y *b* debe estar en el rango 0..colormode.

Si *turtleshape* es un polígono, la línea del polígono es dibujado con el muevo color de lápiz elegido.

```
>>> colormode()
1.0
>>> turtle.pencolor()
'red'
>>> turtle.pencolor("brown")
>>> turtle.pencolor()
'brown'
>>> tup = (0.2, 0.8, 0.55)
>>> turtle.pencolor(tup)
>>> turtle.pencolor()
(0.2, 0.8, 0.5490196078431373)
```

```
>>> colormode(255)
>>> turtle.pencolor()
(51.0, 204.0, 140.0)
>>> turtle.pencolor('#32c18f')
>>> turtle.pencolor()
(50.0, 193.0, 143.0)
```

turtle.fillcolor()
turtle.fillcolor(*color*, */*)
turtle.fillcolor(*r*, *g*, *b*, */*)

Return or set the fillcolor.

Se permiten cuatro formatos de entrada:

`fillcolor()`
Return the current fillcolor as color specification string, possibly in tuple format (see example). May be used as input to another color/pencolor/fillcolor/bgcolor call.

`fillcolor(colorstring)`
Establece *fillcolor* a *colorstring*, que es un color TK especificado como una palabra (en inglés), tales como *«red»*, *«yellow»*, o `"#33cc8c"`.

`fillcolor((r, g, b))`
Establece *fillcolor* al color RGB representado por la tupla *r*, *g*, y *b*. Cada uno de los valores *r*, *g*, y *b* debe estar en el rango 0..colormode, donde *colormode* es 1.0 o 255 (ver **`colormode()`**).

`fillcolor(r, g, b)`
Establece *fillcolor* al color RGB representado por *r*, *g*, y *b*. Cada uno de los valores *r*, *g* y *b* debe ser un valor en el rango 0..colormode.

Si *turtleshape* es un polígono, el interior de ese polígono es dibujado con el color establecido en *fillcolor*.

```
>>> turtle.fillcolor("violet")
>>> turtle.fillcolor()
```

```
'violet'
>>> turtle.pencolor()
(50.0, 193.0, 143.0)
>>> turtle.fillcolor((50, 193, 143))  # Integers, n
>>> turtle.fillcolor()
(50.0, 193.0, 143.0)
>>> turtle.fillcolor('#ffffff')
>>> turtle.fillcolor()
(255.0, 255.0, 255.0)
```

turtle.color()
turtle.color(*color, /*)
turtle.color(*r, g, b, /*)
turtle.color(*pencolor*, *fillcolor, /*)

Retorna o establece *pencolor* (el color del lápiz) y *fillcolor* (el color de relleno).

Se permiten varios formatos de entrada. Usan de 0 a 3 argumentos como se muestra a continuación:

`color()`
Devuelve el valor actual de *pencolor* y el valor actual de *fillcolor* como un par de colores especificados como palabras o tuplas, como devuelven las funciones **`pencolor()`** y **`fillcolor()`**.

`color(colorstring), color((r,g,b)), color(r,g,b)`
Entradas como en **`pencolor()`**, establece al valor dado tanto, *fillcolor* como *pencolor*.

`color(colorstring1, colorstring2), color((r1,g1,b1), (r2,g2,b2))`
Equivalente a `pencolor(colorstring1)` y `fillcolor(colorstring2)` y análogamente si se usa el otro formato de entrada.

Si *turtleshape* es un polígono, la línea y el interior de ese polígono e dibujado con los nuevos colores que se establecieron.

```
>>> turtle.color("red", "green")
>>> turtle.color()
('red', 'green')
>>> color("#285078", "#a0c8f0")
>>> color()
((40.0, 80.0, 120.0), (160.0, 200.0, 240.0))
```

Ver también: Método *Screeen* **`colormode()`**.

Relleno

turtle.filling()

Devuelve *fillstate* (`True` si está lleno, sino `False`).

```
>>> turtle.begin_fill()
>>> if turtle.filling():
...     turtle.pensize(5)
... else:
...     turtle.pensize(3)
```

turtle.fill()

Fill the shape drawn in the `with turtle.fill():` block.

```
>>> turtle.color("black", "red")
>>> with turtle.fill():
...      turtle.circle(80)
```

Using **`fill()`** is equivalent to adding the **`begin_fill()`** before the fill-block and **`end_fill()`** after the fill-block:

```
>>> turtle.color("black", "red")
>>> turtle.begin_fill()
>>> turtle.circle(80)
>>> turtle.end_fill()
```

Added in version 3.14.

turtle.begin_fill()

Para ser llamada justo antes de dibujar una forma a rellenar.

turtle.end_fill()

Rellena la forma dibujada después de última llamada a la función `begin_fill()`.

Superponer o no, regiones de polígonos auto-intersectados o múltiples formas, estas son rellenadas dependiendo de los gráficos del sistema operativo, tipo de superposición y número de superposiciones. Por ejemplo, la flecha de la tortuga de arriba, puede ser toda amarilla o tener algunas regiones blancas.

```
>>> turtle.color("black", "red")
>>> turtle.begin_fill()
>>> turtle.circle(80)
>>> turtle.end_fill()
```

Más controles de dibujo

turtle.reset()

Borra el dibujo de la tortuga de la pantalla, centra la tortuga y establece las variables a los valores por defecto.

```
>>> turtle.goto(0,-22)
>>> turtle.left(100)
>>> turtle.position()
(0.00,-22.00)
>>> turtle.heading()
100.0
>>> turtle.reset()
>>> turtle.position()
(0.00,0.00)
>>> turtle.heading()
0.0
```

turtle.clear()

Borra el dibujo de la tortuga de la pantalla. No mueve la tortuga. El estado y posición de la tortuga así como los todos los dibujos de las otras tortugas no son afectados.

turtle.write(*arg*, *move=False*, *align='left'*, *font=('Arial', 8, 'normal')*)

Parámetros:

- **arg** – objeto que se escribirá en *TurtleScreen*
- **move** – True/False
- **align** – una de las frases «*left*», «*center*» o «*right*»
- **font** – un trio (nombre de fuente, tamaño de fuente, tipo de fuente)

Escribe texto - la representación de cadena de caracteres de *arg* - en la posición actual de la tortuga de acuerdo con *align* («izquierda», «centro» o «derecha») y con la fuente dada. Si *mov* es verdadero, el lápiz se mueve a la esquina inferior derecha del texto. De forma predeterminada, *move* es `False`.

```
>>> turtle.write("Home = ", True, align="center")
>>> turtle.write((0,0), True)
```

Estado de la Tortuga

Visibilidad

turtle.hideturtle()
turtle.ht()

Hace invisible a la tortuga, Es una buena idea hacer eso mientras está haciendo dibujos complejos, ya que esconder a la tortuga acelera dibujo de manera observable.

```
>>> turtle.hideturtle()
```

turtle.showturtle()
turtle.st()

Hace visible la tortuga.

```
>>> turtle.showturtle()
```

turtle.isvisible()

Devuelve `True` si la tortuga se muestra, `False` si está oculta.

```
>>> turtle.hideturtle()
>>> turtle.isvisible()
False
>>> turtle.showturtle()
>>> turtle.isvisible()
True
```

Apariencia

turtle.shape(*name=None*)

Parámetros:

name – una cadena de caracteres que es un nombre de forma válido

Establece la forma de la tortuga al *name* que se establece o, si no se establece un nombre, devuelve el nombre actual de su forma. La forma *name* debe existir en el diccionario de formas de *TurtleScreen*. Inicialmente están las siguientes formas poligonales: «*arrow*», «*turtle*», «*circle*», «*square*», «*triangle*», «*classic*». Para aprender como trabajar con estas formas ver los métodos de Screen **`register_shape()`**.

```
>>> turtle.shape()
'classic'
>>> turtle.shape("turtle")
>>> turtle.shape()
'turtle'
```

turtle.resizemode(*rmode=None*)

Parámetros:

rmode – una de las cadenas «*auto*», «*user*», «*noresize*»

Establece *resizemode* a alguno de los valores: «*auto*», «*user*», «*noresize*». Si *mode* no se aporta, devuelve el actual *resizemode*. Distintos *resizemode* tienen los siguientes efectos:

- «*auto*»: adapta la apariencia de la tortuga al correspondiente valor del lápiz.
- «*user*»: adapta la apariencia de la tortuga de acuerdo a los valores de *sretchfactor* y *outlinewidth* (contorno), que se establece con la función **`shapesize()`**.
- «*noresize*»: no se adapta la apariencia de la tortuga.

`resizemode("user")` es llamado por la función **`shapesize()`** cuando se usa con argumentos.

```
>>> turtle.resizemode()
'noresize'
>>> turtle.resizemode("auto")
>>> turtle.resizemode()
'auto'
```

turtle.shapesize(*stretch_wid=None*, *stretch_len=None*, *outline=None*)
turtle.turtlesize(*stretch_wid=None*, *stretch_len=None*, *outline=None*)

Parámetros:

- **stretch_wid** – número positivo
- **stretch_len** – número positivo
- **outline** – número positivo

Devuelve o establece los atributos x/y-stretchfactors y/o outline del lápiz. Establece resizemode en «user». Si y solo si resizemode está establecido en «user», la tortuga se mostrará estirada de acuerdo con sus factores de estiramiento: *stretch_wid* es el factor de estiramiento perpendicular a su orientación, *stretch_len* es el factor de estiramiento en la dirección de su orientación, *outline* determina el ancho del contorno de la forma.

```
>>> turtle.shapesize()
(1.0, 1.0, 1)
>>> turtle.resizemode("user")
>>> turtle.shapesize(5, 5, 12)
>>> turtle.shapesize()
(5, 5, 12)
>>> turtle.shapesize(outline=8)
>>> turtle.shapesize()
(5, 5, 8)
```

turtle.shearfactor(*shear=None*)

Parámetros:
shear – número (opcional)

Establece o devuelve el valor actual del estiramiento. Estira la forma de la tortuga de acuerdo a la inclinación del factor de corte, que es la tangente del ángulo de corte. No cambia el rumbo de la tortuga (dirección del movimiento). Si no de da un valor de inclinación: devuelve el factor actual, por ejemplo la tangente del ángulo de inclinación, por el cual se cortan la líneas paralelas al rumbo de la tortuga.

```
>>> turtle.shape("circle")
>>> turtle.shapesize(5,2)
>>> turtle.shearfactor(0.5)
>>> turtle.shearfactor()
0.5
```

turtle.tilt(*angle*)

Parámetros:
angle – un número

Rota la forma de la tortuga en *ángulo* desde su ángulo de inclinación actual, pero no cambia el rumbo de la tortuga (dirección del movimiento).

```
>>> turtle.reset()
>>> turtle.shape("circle")
>>> turtle.shapesize(5,2)
>>> turtle.tilt(30)
>>> turtle.fd(50)
>>> turtle.tilt(30)
>>> turtle.fd(50)
```

turtle.tiltangle(*angle=None*)

Parámetros:

angle – un número (opcional)

Establece o devuelve el ángulo de inclinación actual. Si se otorga un ángulo, rota la forma de la tortuga para apuntar en la dirección del ángulo especificado, independientemente de su actual ángulo de inclinación. No cambia el rumbo de la tortuga (dirección del movimiento). Si no se da el ángulo: devuelve el ángulo de inclinación actual, por ejemplo: el ángulo entre la orientación de la forma de la tortuga y el rumbo de la tortuga (su dirección de movimiento).

```
>>> turtle.reset()
>>> turtle.shape("circle")
>>> turtle.shapesize(5,2)
>>> turtle.tilt(45)
>>> turtle.tiltangle()
45.0
```

turtle.shapetransform(*t11=None, t12=None, t21=None, t22=None*)

Parámetros:

- **t11** – un número (opcional)
- **t12** – un número (opcional)
- **t21** – un número (opcional)
- **t12** – un número (opcional)

Establece o devuelve la matriz de transformación actual de la forma de la tortuga.

Si no se proporciona ninguno de los elementos de la matriz, retorna la matriz de transformación como una tupla de 4 elementos. De lo contrario, establezca los elementos dados y transforme la forma de tortuga de acuerdo con la matriz que consta de la primera fila t11, t12 y la segunda fila t21, t22. El determinante t11 * t22 - t12 * t21 no debe ser cero, de lo contrario se genera un error. Modifique el factor de estiramiento, el factor de corte y el ángulo de inclinación de acuerdo con la matriz dada.

```
>>> turtle = Turtle()
>>> turtle.shape("square")
>>> turtle.shapesize(4,2)
>>> turtle.shearfactor(-0.5)
>>> turtle.shapetransform()
(4.0, -1.0, -0.0, 2.0)
```

turtle.get_shapepoly()

Devuelve el polígono de la forma actual como tupla de pares de coordenadas. Esto puede ser usado para definir una nueva forma o componentes de una forma compuesta.

```
>>> turtle.shape("square")
>>> turtle.shapetransform(4, -1, 0, 2)
>>> turtle.get_shapepoly()
((50, -20), (30, 20), (-50, 20), (-30, -20))
```

Usando eventos

turtle.onclick(*fun*, *btn=1*, *add=None*)

Parámetros:

- **fun** – una función con dos argumentos que se invocará con las coordenadas del punto en el que se hizo clic en el lienzo
- **btn** – número del botón del mouse, el valor predeterminado es 1 (botón izquierdo del mouse)

- **add** – `True` o `False` – si es `True`, se agrega un nuevo enlace, de lo contrario reemplazará el enlace anterior

Enlaza *acciones divertidas* a eventos de click en la tortuga. Si la *accion* es `None`, las acciones asociadas son borradas. Ejemplo para la tortuga anónima, en la forma procedimental:

```
>>> def turn(x, y):
...     left(180)
...
>>> onclick(turn)  # Now clicking into the turtle w
>>> onclick(None)  # event-binding will be removed
```

turtle.onrelease(*fun*, *btn=1*, *add=None*)

Parámetros:

- **fun** – una función con dos argumentos que se invocará con las coordenadas del punto en el que se hizo clic en el lienzo
- **btn** – número del botón del mouse, el valor predeterminado es 1 (botón izquierdo del mouse)
- **add** – `True` o `False` – si es `True`, se agrega un nuevo enlace, de lo contrario reemplazará el enlace anterior

Enlaza *acciones divertidas* a eventos del tipo soltar botón del mouse en la tortuga. SI la *acción* es `None`, las acciones asociadas son borradas.

```
>>> class MyTurtle(Turtle):
...     def glow(self,x,y):
...         self.fillcolor("red")
...     def unglow(self,x,y):
...         self.fillcolor("")
...
>>> turtle = MyTurtle()
>>> turtle.onclick(turtle.glow)     # clicking on t
>>> turtle.onrelease(turtle.unglow) # releasing tur
```

turtle.ondrag(*fun*, *btn=1*, *add=None*)

Parámetros:

- **fun** – una función con dos argumentos que se invocará con las coordenadas del punto en el que se hizo clic en el lienzo
- **btn** – número del botón del mouse, el valor predeterminado es 1 (botón izquierdo del mouse)
- **add** – `True` o `False` – si es `True`, se agrega un nuevo enlace, de lo contrario reemplazará el enlace anterior

Enlaza *acciones divertidas* a eventos en los movimientos del mouse. Si la *acción* es `None`, las acciones asociadas son borradas.

Observación: cada secuencia de los eventos de movimiento del mouse en una tortuga es precedida por un evento de click del mouse en esa tortuga.

```
>>> turtle.ondrag(turtle.goto)
```

Subsecuentemente, clickear y arrastrar la Tortuga la moverá a través de la pantalla produciendo dibujos a mano alzada (si el lápiz está abajo).

Métodos especiales de *Turtle*

turtle.poly()

Record the vertices of a polygon drawn in the `with turtle.poly():` block. The first and last vertices will be connected.

```
>>> with turtle.poly():
```

```
...     turtle.forward(100)
...     turtle.right(60)
...     turtle.forward(100)
```

Added in version 3.14.

turtle.begin_poly()

Comienza a grabar los vértices de un polígono. La posición actual de la tortuga es el primer vértice del polígono.

turtle.end_poly()

Deja de grabar los vértices de un polígono. La posición actual de la tortuga es el último vértice del polígono. Esto se conectará con el primer vértice.

turtle.get_poly()

Devuelve el último polígono grabado.

```
>>> turtle.home()
>>> turtle.begin_poly()
>>> turtle.fd(100)
>>> turtle.left(20)
>>> turtle.fd(30)
>>> turtle.left(60)
>>> turtle.fd(50)
>>> turtle.end_poly()
>>> p = turtle.get_poly()
>>> register_shape("myFavouriteShape", p)
```

turtle.clone()

Crea y devuelve un clon de la tortuga con la misma posición, dirección y propiedades de la tortuga.

```
>>> mick = Turtle()
>>> joe = mick.clone()
```

turtle.getturtle()

turtle.getpen()

Devuelve el objeto Tortuga en si. El único uso razonable: es como una función para devolver la «tortuga anónima»:

```
>>> pet = getturtle()
>>> pet.fd(50)
>>> pet
<turtle.Turtle object at 0x...>
```

turtle.getscreen()

Devuelve el objeto `TurtleScreen` sobre el cual la tortuga está dibujando. Los métodos *TurtleScreen* luego pueden ser llamados para ese objeto.

```
>>> ts = turtle.getscreen()
>>> ts
<turtle._Screen object at 0x...>
>>> ts.bgcolor("pink")
```

turtle.setundobuffer(*size*)

Parámetros:

size – un entero o `None`

Establecer o deshabilitar deshacer búfer. Si *size* es un número entero, se instala un búfer de deshacer vacío de un tamaño determinado. *size* da el número máximo de acciones de tortuga que se pueden deshacer mediante el método/función `undo()`. Si *size* es `None`, el búfer para deshacer está deshabilitado.

```
>>> turtle.setundobuffer(42)
```

turtle.undobufferentries()

Devuelve el número de entradas en el buffer para deshacer acciones.

```
>>> while undobufferentries():
```

```
...      undo()
```

Formas compuestas

Para usar formas complejas con la tortuga, que consiste en varios polígonos de diferentes colores, deberá usar la clase de ayuda `Shape` explícitamente como se describe debajo:

1. Crear una objeto de forma vacía del tipo *compound*.
2. Agregue tantos componentes a este objeto como desee, utilizando el método `addcomponent()`.

   Por ejemplo:

   ```
   >>> s = Shape("compound")
   >>> poly1 = ((0,0),(10,-5),(0,10),(-10,-5))
   >>> s.addcomponent(poly1, "red", "blue")
   >>> poly2 = ((0,0),(10,-5),(-10,-5))
   >>> s.addcomponent(poly2, "blue", "red")
   ```

3. Ahora agregar la forma a la lista de formas de la pantalla y úsela:

   ```
   >>> register_shape("myshape", s)
   >>> shape("myshape")
   ```

Nota

La clase `Shape` es usada internamente por el método `register_shape()` en maneras diferentes. El programador deberá lidiar con la clase *Shape* ¡solo cuando use formas compuestas como las que se mostraron arriba!

Métodos de *TurtleScreen/Screen* y sus correspondientes funciones

La mayoría de los ejemplos en esta sección se refieren a la instancia de *TurtleScreen* llamada `screen`.

Control de ventana

turtle.bgcolor()
turtle.bgcolor(*color*, */*)
turtle.bgcolor(*r*, *g*, *b*, */*)

Return or set the background color of the TurtleScreen.

Se permiten cuatro formatos de entrada:

`bgcolor()`
Return the current background color as color specification string or as a tuple (see example). May be used as input to another color/pencolor/fillcolor/bgcolor call.

`bgcolor(colorstring)`
Set the background color to *colorstring*, which is a Tk color specification string, such as `"red"`, `"yellow"`, or `"#33cc8c"`.

`bgcolor((r, g, b))`
Set the background color to the RGB color represented by the tuple of *r*, *g*, and *b*. Each of *r*, *g*, and *b* must be in the range 0..colormode, where colormode is either 1.0 or 255 (see `colormode()`).

`bgcolor(r, g, b)`
Set the background color to the RGB color represented by *r*, *g*, and *b*. Each of *r*, *g*, and *b* must be in the range 0..colormode.

```
>>> screen.bgcolor("orange")
>>> screen.bgcolor()
'orange'
>>> screen.bgcolor("#800080")
>>> screen.bgcolor()
(128.0, 0.0, 128.0)
```

turtle.bgpic(*picname=None*)

Parámetros:

picname – a string, name of an image file (PNG, GIF, PGM, and PPM) or `"nopic"`, or `None`

Establece la imagen de fondo o devuelve el nombre de la imagen de fondo actual. Si *picname* es un nombre de archivo, establece la imagen correspondiente como fondo. Si *picname* es `"nopic"`, borra la imagen de fondo, si hay alguna presente. Si *picname* es `None`, devuelve el nombre de archivo de la imagen de fondo actual.

```
>>> screen.bgpic()
'nopic'
>>> screen.bgpic("landscape.gif")
>>> screen.bgpic()
"landscape.gif"
```

turtle.clear()

Nota

Este método de *TurtleScreen* está disponible como una función global solo bajo el nombre `clearscreen`. La función global `clear` es otra, derivada del método de *Turtle* `clear`.

turtle.clearscreen()

Borra todos los dibujos y todas las tortugas del la pantalla de la tortuga. Reinicia la ahora vacía pantalla de la tortuga a su estado inicial: fondo blanco, sin imagen de fondo, sin enlaces a eventos o seguimientos.

turtle.reset()

Nota

Este método *TurtleScreen* esta disponible como una función global solo bajo el nombre `resetscreen`. La función global `reset` es otra, derivada del método *Turtle* `reset`.

turtle.resetscreen()

Reinicia todas las tortugas de la pantalla a su estado inicial.

turtle.screensize(*canvwidth=None*, *canvheight=None*, *bg=None*)

Parámetros:

- **canvwidth** – entero positivo, nueva anchura del lienzo en pixeles
- **canvheight** – entero positivo, nueva altura del lienzo en pixeles
- **bg** – *colorstrng* o tupla de color, muevo color de fondo

Si no se dan argumentos, devuelve el actual (ancho y alto del lienzo). Sino redimensiona el lienzo en el que la tortuga está dibujando. No altera la ventana de dibujo. Para ver las partes ocultas del lienzo, use la barra de desplazamiento. Con este método, se pueden hacer visibles aquellas partes de un dibujo que antes estaban fuera del lienzo.

```
>>> screen.screensize()
(400, 300)
>>> screen.screensize(2000,1500)
>>> screen.screensize()
(2000, 1500)
```

ej. buscar una tortuga que se escapó por error ;-)

turtle.setworldcoordinates(*llx*, *lly*, *urx*, *ury*)

Parámetros:

- **llx** – un número, coordenada x de la esquina inferior izquierda del lienzo
- **lly** – un número, coordenada y de la

esquina inferior izquierda del lienzo

- **urx** – un número, coordenada x de la esquina superior derecha del lienzo
- **ury** – un número, coordenada z de la esquina superior derecha del lienzo

Configura coordenadas definidas por el usuario y cambia al modo *world* si es necesario. Esto realiza un `screen.reset()`. Si el modo *world* ya está activo, todos los dibujos se re dibujan de acuerdo a las nuevas coordenadas.

ATENCIÓN: en los sistemas de coordenadas definidos por el usuario, los ángulos pueden aparecer distorsionados.

```
>>> screen.reset()
>>> screen.setworldcoordinates(-50,-7.5,50,7.5)
>>> for _ in range(72):
...     left(10)
...
>>> for _ in range(8):
...     left(45); fd(2)   # a regular octagon
```

Control de animación

turtle.no_animation()

Temporarily disable turtle animation. The code written inside the `no_animation` block will not be animated; once the code block is exited, the drawing will appear.

```
>>> with screen.no_animation():
...     for dist in range(2, 400, 2):
...         fd(dist)
...         rt(90)
```

Added in version 3.14.

turtle.delay(*delay=None*)

Parámetros:

delay – entero positivo

Establece o retorna el *retraso* del dibujo en mili segundos. (Este es aproximadamente, el tiempo de intervalo entre dos actualizaciones consecutivas del lienzo). Mientras más largo sea el retraso, más lenta la animación.

Argumento opcional:

```
>>> screen.delay()
10
>>> screen.delay(5)
>>> screen.delay()
5
```

turtle.tracer(*n*=*None*, *delay*=*None*)

Parámetros:

- **n** – entero no negativo
- **delay** – entero no negativo

Activa o desactiva la animación de la tortuga y establece el retraso para la actualización de los dibujos. Si se da *n*, solo cada enésima actualización regular de pantalla se realiza realmente. (Puede usarse para acelerar los dibujos de gráficos complejos). Cuando es llamada sin argumentos, devuelve el valor de *n* guardado actualmente. El segundo argumento establece el valor de retraso (ver `delay()`).

```
>>> screen.tracer(8, 25)
>>> dist = 2
>>> for i in range(200):
...     fd(dist)
...     rt(90)
...     dist += 2
```

turtle.update()

Realiza una actualización de la pantalla de la tortuga. Para ser usada cuando *tracer* está deshabilitada.

Ver también el método *RawTurtle/Turtle* `speed()`.

Usando eventos de pantalla

turtle.listen(*xdummy=None, ydummy=None*)

Establece el foco en la pantalla de la tortuga (para recoger eventos de teclado). Los argumentos *dummy* se proveen en orden de ser capaces de pasar `listen()` a los métodos *onclick*.

turtle.onkey(*fun, key*)

turtle.onkeyrelease(*fun, key*)

Parámetros:

- **fun** – una función sin argumentos o `None`
- **key** – una cadena de caracteres: tecla (por ejemplo, «a») o una acción del teclado (por ejemplo, «space»)

Vincula *fun* a un evento de liberación de una tecla. Si *fun* es `None`, los eventos vinculados son removidos. Aclaración: para poder registrar eventos de teclado, *TurtleScreen* tiene que tener el foco. (ver el método `listen()`.)

```
>>> def f():
...     fd(50)
...     lt(60)
...
>>> screen.onkey(f, "Up")
>>> screen.listen()
```

turtle.onkeypress(*fun, key=None*)

Parámetros:

- **fun** – una función sin argumentos o `None`
- **key** – una cadena de caracteres: tecla (por ejemplo, «a») o una acción del

teclado (por ejemplo, «space»)

Vincula *fun* a un evento de pulsado de una tecla, o a cualquier evento de pulsado de tecla si no se da una. Aclaración: para poder registrar eventos de teclado, *TurtleScreen* tiene que tener el foco. (ver el método **`listen()`**.)

```
>>> def f():
...     fd(50)
...
>>> screen.onkey(f, "Up")
>>> screen.listen()
```

turtle.onclick(*fun, btn=1, add=None*)
turtle.onscreenclick(*fun, btn=1, add=None*)

Parámetros:

- **fun** – una función con dos argumentos que se invocará con las coordenadas del punto en el que se hizo clic en el lienzo
- **btn** – número del botón del mouse, el valor predeterminado es 1 (botón izquierdo del mouse)
- **add** – `True` o `False` – si es `True`, se agrega un nuevo enlace, de lo contrario reemplazará el enlace anterior

Vincula *fun* a eventos de click del mouse en esta pantalla. Si *fun* es `None`, los vínculos existentes son removidos.

Ejemplo de una instancia TurtleScreen llamada `screen` y una instancia Turtle llamada `turtle`:

```
>>> screen.onclick(turtle.goto) # Subsequently clic
>>>                             # make the turtle m
>>> screen.onclick(None)        # remove event bind
```

Nota

Este método *TurtleScreen* está disponible como una función global solo bajo el nombre `onscreenclick`. La función global `onclick` es otra derivada del método Turtle `onclick`.

turtle.ontimer(*fun*, *t=0*)

Parámetros:

- **fun** – una función sin argumentos
- **t** – un número $>= 0$

Instala un temporizador que llama a *fun* cada *t* milisegundos.

```
>>> running = True
>>> def f():
...     if running:
...         fd(50)
...         lt(60)
...         screen.ontimer(f, 250)
>>> f()   ### makes the turtle march around
>>> running = False
```

turtle.mainloop()
turtle.done()

Comienza un bucle de evento - llamando a la función *mainloop* del *Tkinter*. Debe ser la última declaración en un programa gráfico de turtle. *No* debe ser usado si algún script es corrido dentro del IDLE en modo -n (Sin subproceso) - para uso interactivo de gráficos turtle.:

```
>>> screen.mainloop()
```

Métodos de entrada

turtle.textinput(*title*, *prompt*)

Parámetros:

- **title** – cadena de caracteres
- **prompt** – cadena de caracteres

Abre una ventana de diálogo para ingresar una cadena de caracteres. El parámetro *title* es el título de la ventana de diálogo, *prompt* es un texto que usualmente describe que información se debe ingresar. Devuelve la cadena ingresada. Si el diálogo es cancelado, devuelve `None`.

```
>>> screen.textinput("NIM", "Name of first player:"
```

turtle.numinput(*title*, *prompt*, *default=None*, *minval=None*, *maxval=None*)

Parámetros:

- **title** – cadena de caracteres
- **prompt** – cadena de caracteres
- **default** – número (opcional)
- **minval** – número (opcional)
- **maxval** – número (opcional)

Abre una ventana de diálogo para ingresar un número. El título es el título de la ventana de diálogo, el mensaje es un texto que describe principalmente qué información numérica ingresar. predeterminado: valor predeterminado, minval: valor mínimo para la entrada, maxval: valor máximo para la entrada. El número ingresado debe estar en el rango minval… maxval si se proporcionan estos. Si no, se emite una pista y el diálogo permanece abierto para su corrección. Devuelve el número ingresado. Si se cancela el diálogo, devuelve `None`.

```
>>> screen.numinput("Poker", "Your stakes:", 1000,
```

Configuración y métodos especiales

turtle.mode(*mode=None*)

Parámetros:

mode – una de las cadenas *«standard»*, *«logo»* o *«world»*

Establece el mode de la tortuga (*«standard»*, *«logo»* o *«world»*) y realiza un reinicio. Si no se da «*mode*», retorna el modo actual.

Mode «standard» is compatible with old **`turtle`**. Mode «logo» is compatible with most Logo turtle graphics. Mode «world» uses user-defined «world coordinates». **Attention**: in this mode angles appear distorted if `x/y` unit-ratio doesn't equal 1.

Modo	**Rumbo inicial de la tortuga**	**Ángulos positivos**
standard	hacia la derecha (este)	sentido antihorario
logo	hacia arriba (norte)	sentido horario

```
>>> mode("logo")   # reinicia la tortuga hacia el n
>>> mode()
'logo'
```

turtle.colormode(*cmode*=*None*)

Parámetros:

cmode – uno de los valores 1.0 o 255

Devuelve el modo de color o configúrelo en 1.0 o 255. Posteriormente, los valores *r*, *g*, *b* de los triples de color deben estar en el rango 0..*cmode*.

```
>>> screen.colormode(1)
>>> turtle.pencolor(240, 160, 80)
Traceback (most recent call last):
     ...
TurtleGraphicsError: bad color sequence: (240, 160,
>>> screen.colormode()
1.0
>>> screen.colormode(255)
>>> screen.colormode()
255
```

```
>>> turtle.pencolor(240,160,80)
```

turtle.getcanvas()

Devuelve el lienzo de este *TurtleScreen*. Útil para conocedores que saben que hace con un *TKinter Canvas*.

```
>>> cv = screen.getcanvas()
>>> cv
<turtle.ScrolledCanvas object ...>
```

turtle.getshapes()

Devuelve la lista de nombres de todas las formas de la tortuga actualmente disponibles.

```
>>> screen.getshapes()
['arrow', 'blank', 'circle', ..., 'turtle']
```

turtle.register_shape(*name*, *shape=None*)
turtle.addshape(*name*, *shape=None*)

There are four different ways to call this function:

1. *name* is the name of an image file (PNG, GIF, PGM, and PPM) and *shape* is `None`: Install the corresponding image shape.

   ```
   >>> screen.register_shape("turtle.gif")
   ```

   Nota

   Las imágenes de tipo *shapes* no rotan cuando la tortuga gira, así que ellas ¡no muestran el rumbo de la tortuga!

2. *name* is an arbitrary string and *shape* is the name of an image file (PNG, GIF, PGM, and PPM): Install the corresponding image shape.

   ```
   >>> screen.register_shape("turtle", "turtle.gi
   ```

Nota

Las imágenes de tipo *shapes* no rotan cuando la tortuga gira, así que ellas ¡no muestran el rumbo de la tortuga!

3. *name* es una cadena de caracteres arbitraria y *shape* es una tupla de pares de coordenadas: Instala la forma poligonal correspondiente.

   ```
   >>> screen.register_shape("triangle", ((5,-3),
   ```

4. *name* es una cadena arbitraria y *shape* es un objeto `Shape` (compuesto): instala la forma compuesta correspondiente.

Agregar una forma de tortuga a la lista de formas de *TurtleScreen*. Solo las formas registradas de esta manera pueden ser usadas invocando el comando `shape(shapename)`.

Distinto en la versión 3.14: Added support for PNG, PGM, and PPM image formats. Both a shape name and an image file name can be specified.

turtle.turtles()

Devuelve la lista de tortugas en la pantalla.

```
>>> for turtle in screen.turtles():
...     turtle.color("red")
```

turtle.window_height()

Devuelve la altura de la ventana de la tortuga.

```
>>> screen.window_height()
480
```

turtle.window_width()

Devuelve el ancho de la ventana de la tortuga.

```
>>> screen.window_width()
640
```

Métodos específicos de *Screen*, no heredados de *TurtleScreen*

turtle.bye()

Apaga la ventana gráfica de la tortuga.

turtle.exitonclick()

Ata el método `bye()` al click del ratón sobre la pantalla.

Si el valor *«using_IDLE»* en la configuración del diccionario es `False` (valor por defecto), también ingresa *mainloop*. Observaciones: si se usa la *IDLE* con el modificador (no subproceso) `-n`, este valor debe establecerse a `True` en `turtle.cfg`. En este caso el propio *mainloop* de la *IDLE* está activa también para script cliente.

turtle.save(*filename*, *overwrite=False*)

Save the current turtle drawing (and turtles) as a PostScript file.

Parámetros:

- **filename** – the path of the saved PostScript file
- **overwrite** – if `False` and there already exists a file with the given filename, then the function will raise a `FileExistsError`. If it is `True`, the file will be overwritten.

```
>>> screen.save("my_drawing.ps")
>>> screen.save("my_drawing.ps", overwrite=True)
```

Added in version 3.14.

turtle.setup(*width=_CFG['width']*, *height=_CFG['height']*, *startx=_CFG['leftright'], starty=_CFG['topbottom']*)

Establece el tamaño y la posición de la ventana principal. Los valores por defecto de los argumentos son guardados en el diccionario de configuración y puede ser cambiado a través del archivo `turtle.cfg`.

Parámetros:

- **width** – si es un entero, el tamaño en pixeles, si es un número flotante, una fracción de la pantalla; el valor por defecto es 50% de la pantalla
- **height** – si es un entero, la altura en pixeles, si es un número flotante, una fracción de la pantalla; el valor por defecto es 75% de la pantalla
- **startx** – si es positivo, punto de inicio en pixeles desde la esquina izquierda de la pantalla, si es negativo desde la esquina derecha de la pantalla, si es `None`, centra la ventana horizontalmente
- **starty** – si es positivo, punto de inicio en pixeles desde la parte superior de la pantalla, si es negativo desde la parte inferior de la pantalla, si es `None`, centra la ventana verticalmente

```
>>> screen.setup (width=200, height=200, startx=0,
>>>              # sets window to 200x200 pixels, i
>>> screen.setup(width=.75, height=0.5, startx=None
>>>              # sets window to 75% of screen by
```

turtle.title(*titlestring*)

Parámetros:

titlestring – una cadena de caracteres que se muestra en la barra de título de la ventana gráfica de la tortuga

Establece el título de la ventana de la tortuga a *titlestring*.

```
>>> screen.title("Welcome to the turtle zoo!")
```

Clases públicas

class turtle.RawTurtle(*canvas*)
class turtle.RawPen(*canvas*)

Parámetros:

canvas – un **`tkinter.Canvas`**, un **`ScrolledCanvas`** o un **`TurtleScreen`**

Crea una tortuga. La tortuga tiene todos los métodos descriptos anteriormente como «métodos de *«Turtle/RawTurtle»*.

class turtle.Turtle

Subclase de *RawTurtle*, tiene la misma interface pero dibuja sobre una objeto por defecto **`Screen`** creado automáticamente cuando es necesario por primera vez.

class turtle.TurtleScreen(*cv*)

Parámetros:

cv – un **`tkinter.Canvas`**

Proporciona métodos orientados a la pantalla como **`bgcolor()`**, etc., que se describen anteriormente.

class turtle.Screen

Subclase de *TurtleScreen*, con cuatro métodos agregados.

class turtle.ScrolledCanvas(*master*)

Parámetros:

master – algunos *widgets TKinter* para contener el *ScrollCanvas*, por ejemplo un lienzo *Tkinter* con barras de desplazamiento agregadas

Usado por la clase *Screen*, que proporciona automáticamente un *ScrolledCanvas* como espacio de trabajo para las tortugas.

class turtle.Shape(*type_*, *data*)

Parámetros:

type_ – una de las cadenas de caracteres *«polygon»*, *«image»*, *«compound»*

Estructura de datos que modela las formas. El par `(type_, data)` debe seguir estas especificaciones:

*type_*	*data*
«polygon»	una tupla para el polígono, por ejemplo: una tupla de par de coordenadas
«image»	una imagen (en esta forma solo se usa ¡internamente!)
«compound»	`None` (una forma compuesta tiene que ser construida usando el método `addcomponent()`)

addcomponent(*poly*, *fill*, *outline=None*)

Parámetros:

- **poly** – un polígono, por ejemplo una tupla de pares de números
- **fill** – un color con el que el polígono será llenado
- **outline** – un color con el que se delineará el polígono (se de ingresa)

Ejemplo:

```
>>> poly = ((0,0),(10,-5),(0,10),(-10,-5))
>>> s = Shape("compound")
>>> s.addcomponent(poly, "red", "blue")
>>> # ... add more components and then use regi
```

Ver Formas compuestas.

class turtle.Vec2D(*x*, *y*)

Una clase de vectores bidimensionales, usado como clase de ayuda para implementar gráficos de tortuga. Puede ser útil para los programas de gráficos de tortugas también. Derivado de la tupla, ¡por lo que un vector es una tupla!

Proporciona (para *a*, *b* vectores, *k* números)

- `a + b` suma de vectores
- `a - b` resta de vectores
- `a * b` producto interno
- `k * a` y `a * k` multiplicación con escalar
- `abs(a)` valor absoluto de a
- `a.rotate(angle)` rotación

Explicación

Un objeto tortuga se dibuja en un objeto de pantalla, y hay una serie de clases clave en la interfaz orientada a objetos tortuga que se pueden usar para crearlos y relacionarlos entre sí.

Una instancia **`Turtle`** creará automáticamente una instancia **`Screen`** si aún no hay una presente.

`Turtle` es una subclase de **`RawTurtle`**, que crea automáticamente una superficie de dibujo. Para ello, será necesario proporcionar o crear una *canvas*. La *canvas* puede ser una **`tkinter.Canvas`**, una **`ScrolledCanvas`** o una **`TurtleScreen`**.

`TurtleScreen` es la superficie de dibujo básica de una tortuga. **`Screen`** es una subclase de `TurtleScreen` e incluye some additional methods para gestionar su apariencia (incluido el tamaño y el título) y su comportamiento. El constructor de `TurtleScreen` necesita un **`tkinter.Canvas`** o un **`ScrolledCanvas`** como argumento.

La interfaz funcional para gráficos de tortugas utiliza los distintos métodos de `Turtle` y `TurtleScreen`/`Screen`. En segundo plano, se crea automáticamente un objeto de pantalla cada vez que se llama a una función derivada de un método `Screen`. De manera similar, se crea automáticamente un objeto de tortuga cada vez que se llama a cualquiera de las funciones derivadas de un método de Turtle.

Para utilizar varias tortugas en una pantalla, se debe utilizar la interfaz orientada a objetos.

Ayuda y configuración

Cómo usar la ayuda

Los métodos públicos de las clases Screen y Turtle están ampliamente documentados a través de cadenas de documentación. Por lo tanto, estos se pueden usar como ayuda en línea a través de las instalaciones de ayuda de Python:

- Cuando se usa IDLE, la información sobre herramientas muestra las firmas y las primeras líneas de las cadenas de documentos de las llamadas de función/método escritas.
- Llamar a **`help()`** en métodos o funciones muestra los docstrings:

  ```
  >>> help(Screen.bgcolor)
  Help on method bgcolor in module turtle:

  bgcolor(self, *args) unbound turtle.Screen method
      Set or return backgroundcolor of the TurtleScre

      Arguments (if given): a color string or three n
      in the range 0..colormode or a 3-tuple of such


      >>> screen.bgcolor("orange")
      >>> screen.bgcolor()
      "orange"
      >>> screen.bgcolor(0.5,0,0.5)
  ```

```
    >>> screen.bgcolor()
    "#800080"

>>> help(Turtle.penup)
Help on method penup in module turtle:

penup(self) unbound turtle.Turtle method
    Pull the pen up -- no drawing when moving.

    Aliases: penup | pu | up

    No argument

    >>> turtle.penup()
```

- Los docstrings de las funciones que se derivan de los métodos tienen una forma modificada:

```
>>> help(bgcolor)
Help on function bgcolor in module turtle:

bgcolor(*args)
    Set or return backgroundcolor of the TurtleScre

    Arguments (if given): a color string or three n
    in the range 0..colormode or a 3-tuple of such

    Example::

      >>> bgcolor("orange")
      >>> bgcolor()
      "orange"
      >>> bgcolor(0.5,0,0.5)
      >>> bgcolor()
      "#800080"

>>> help(penup)
Help on function penup in module turtle:
```

```
penup()
    Pull the pen up -- no drawing when moving.

    Aliases: penup | pu | up

    No argument

    Example:
    >>> penup()
```

Estos docstrings modificados se crean automáticamente junto con las definiciones de función que se derivan de los métodos en el momento de la importación.

Traducción de cadenas de documentos a diferentes idiomas

Existe una utilidad para crear un diccionario cuyas claves son los nombres de los métodos y cuyos valores son los docstrings de los métodos públicos de las clases Screen y Turtle.

turtle.write_docstringdict(*filename='turtle_docstringdict'*)

Parámetros:

filename – una cadena de caracteres, utilizada como nombre de archivo

Crea y escribe un diccionario-docstring en un script de Python con el nombre de archivo dado. Esta función tiene que ser llamada explícitamente (no es utilizada por las clases de gráficos de tortugas). El diccionario de docstrings se escribirá en el script de Python *`filename`*`.py`. Está destinado a servir como plantilla para la traducción de las cadenas de documentos a diferentes idiomas.

If you (or your students) want to use **`turtle`** with online help in your native language, you have to translate the docstrings and save the resulting file as e.g. `turtle_docstringdict_german.py`.

Si tiene una entrada adecuada en su archivo `turtle.cfg`, este diccionario se leerá en el momento de la importación y reemplazará los docstrings originales en inglés.

En el momento de escribir este artículo, existen diccionarios de docstrings en alemán e italiano. (Solicitudes por favor a glingl@aon.at.)

Cómo configurar Screen and Turtles

La configuración predeterminada incorporada imita la apariencia y el comportamiento del antiguo módulo de tortuga para mantener la mejor compatibilidad posible con él.

Si desea utilizar una configuración diferente que refleje mejor las características de este módulo o que se adapte mejor a sus necesidades, por ejemplo, para su uso en un aula, puede preparar un archivo de configuración `turtle.cfg` que se leerá en el momento de la importación y modificará la configuración de acuerdo con su configuración.

La configuración incorporada correspondería al siguiente `turtle.cfg`:

```
width = 0.5
height = 0.75
leftright = None
topbottom = None
canvwidth = 400
canvheight = 300
mode = standard
colormode = 1.0
delay = 10
undobuffersize = 1000
shape = classic
pencolor = black
fillcolor = black
resizemode = noresize
visible = True
language = english
```

```
exampleturtle = turtle
examplescreen = screen
title = Python Turtle Graphics
using_IDLE = False
```

Breve explicación de las entradas seleccionadas:

- Las primeras cuatro líneas corresponden a los argumentos del método `Screen.setup`.
- Las líneas 5 y 6 corresponden a los argumentos del método `Screen.screensize`.
- *shape* puede ser cualquiera de las formas integradas, por ejemplo: arrow, turtle, etc. Para obtener más información, pruebe con `help(shape)`.
- Si no desea utilizar ningún color de relleno (es decir, hacer que la tortuga sea transparente), debe escribir `fillcolor = ""` (pero todas las cadenas no vacías no deben tener comillas en el archivo cfg).
- Si desea reflejar el estado de la tortuga, debe usar `resizemode = auto`.
- If you set e.g. `language = italian` the docstringdict `turtle_docstringdict_italian.py` will be loaded at import time (if present on the import path, e.g. in the same directory as **`turtle`**).
- Las entradas *exampleturtle* y *examplescreen* definen los nombres de estos objetos a medida que aparecen en las cadenas de documentos. La transformación de método-docstrings en función-docstrings eliminará estos nombres de las docstrings.
- *using_IDLE*: Establezca esta opción en `True` si trabaja habitualmente con IDLE y su opción `-n` («sin subproceso»). Esto evitará que `exitonclick()` ingrese al bucle principal.

There can be a `turtle.cfg` file in the directory where **`turtle`** is stored and an additional one in the current working directory. The latter will override the settings of the first one.

El directorio `Lib/turtledemo` contiene un archivo `turtle.cfg`. Puede estudiarlo como un ejemplo y ver sus efectos al ejecutar las demostraciones (preferiblemente no desde el visor de demostraciones).

`turtledemo` — Demo scripts

The **`turtledemo`** package includes a set of demo scripts. These scripts can be run and viewed using the supplied demo viewer as follows:

```
python -m turtledemo
```

Alternativamente, puede ejecutar los scripts de demostración individualmente. Por ejemplo,

```
python -m turtledemo.bytedesign
```

The **`turtledemo`** package directory contains:

- Un visor de demostración `__main__.py` que se puede utilizar para ver el código fuente de los scripts y ejecutarlos al mismo tiempo.
- Multiple scripts demonstrating different features of the **`turtle`** module. Examples can be accessed via the Examples menu. They can also be run standalone.
- Un archivo `turtle.cfg` que sirve como ejemplo de cómo escribir y usar dichos archivos.

Los scripts de demostración son:

Nombre	Descripción	Características
bytedesign	patrón de gráficos de tortuga clásica compleja	`tracer()`, `delay()`, `update()`
chaos	gráficos dinámicos de Verhulst, muestra que los cálculos de la computadora pueden generar resultados a veces contra las expectativas del sentido común	coordenadas mundiales
clock	reloj analógico que muestra la hora de su computadora	turtles as clock's hands, `ontimer()`
colormixer	experimento con r, g, b	`ondrag()`
forest	3 árboles de ancho primero	aleatorización
fractalcurves	Curvas Hilbert & Koch	recursión
lindenmayer	etnomatemáticas (kolams indios)	Sistema-L
minimal_hanoi	Torres de Hanoi	Rectangular Turtles as Hanoi discs (`shape()`, `shapesize()`)
nim	juega el clásico juego de nim	tortugas como nimsticks, impulsado por eventos (mouse, teclado)

	la computadora.	
paint	programa de dibujo super minimalista	`onclick()`
peace	elemental	tortuga: apariencia y animación
penrose	embaldosado aperiódico con cometas y dardos	`stamp()`
planet_and_moon	simulación de sistema gravitacional	formas compuestas, `Vec2D`
rosette	un patrón del artículo de wikipedia sobre gráficos de tortuga (*turtle*)	`clone()`, `undo()`
round_dance	tortugas bailarinas que giran por parejas en dirección opuesta	formas compuestas, `clone()` `shapesize()`, `tilt()`, `get_shapepoly()`, `update()`
sorting_animate	demostración visual de diferentes métodos de ordenamiento	alineación simple, aleatorización
tree	un primer árbol de amplitud (gráfico, usando generadores)	`clone()`
two_canvases	diseño simple	tortugas en dos lienzos (*two_canvases*)
yinyang	otro ejemplo elemental	`circle()`

¡Diviértete!

Cambios desde Python 2.6

- Se han eliminado los métodos **`Turtle.tracer`**, **`Turtle.window_width`** y **`Turtle.window_height`**. Los métodos con estos nombres y funcionalidades ahora solo están disponibles como métodos de **`Screen`**. Las funciones derivadas de estos siguen estando disponibles. (De hecho, ya en Python 2.6 estos métodos eran meras duplicaciones de los métodos **`TurtleScreen`**/**`Screen`** correspondientes).
- Se ha eliminado el método **`Turtle.fill()`**. El comportamiento de **`begin_fill()`** y **`end_fill()`** ha cambiado ligeramente: ahora cada proceso de llenado debe completarse con una llamada a `end_fill()`.
- Se ha añadido un método **`Turtle.filling`**. Devuelve un valor booleano: `True` si se está realizando un proceso de llenado, `False` en caso contrario. Este comportamiento corresponde a una llamada `fill()` sin argumentos en Python 2.6.

Cambios desde Python 3.0

- Se han añadido los métodos **`shearfactor()`**, **`shapetransform()`** y **`get_shapepoly()`** de **`Turtle`**. De

esta forma, ahora está disponible toda la gama de transformaciones lineales regulares para transformar formas de tortugas. Se ha mejorado la funcionalidad de `tiltangle()`: ahora se puede utilizar para obtener o establecer el ángulo de inclinación.
- Se ha añadido el método `Screen onkeypress()` como complemento de `onkey()`. Como este último vincula acciones al evento de liberación de la tecla, también se le ha añadido un alias: `onkeyrelease()`.
- Se ha agregado el método `Screen.mainloop`, por lo que ya no es necesario utilizar la función independiente `mainloop()` al trabajar con objetos `Screen` y `Turtle`.
- Se han añadido dos métodos de entrada: `Screen.textinput` y `Screen.numinput`. Estos abren cuadros de diálogo de entrada y devuelven cadenas y números respectivamente.

Herramientas de desarrollo

Los módulos descritos en este capítulo le ayudan a escribir software. Por ejemplo, el módulo `pydoc` toma un módulo y genera documentación basada en el contenido del módulo. Los módulos `doctest` y `unittest` contienen frameworks para escribir pruebas unitarias que ejecutan y validan automáticamente el código, verificando que se produce la salida esperada.

La lista de módulos descritos en este capítulo es:

- `typing` — Support for type hints
    - Especificación del sistema de tipos de Python
    - Alias de tipo
    - NewType
    - Anotaciones en objetos invocables
    - Genéricos
    - Anotaciones en tuplas
    - El tipo de objetos de clase
    - Anotación de generadores y corrutinas
    - Tipos genéricos definidos por el usuario
    - El tipo `Any`
    - Subtipado nominal vs estructural
    - Contenido del módulo
        - Primitivos especiales de tipado
            - Tipos especiales
            - Formas especiales
            - Creación de tipos genéricos y alias de tipos
            - Otras directivas especiales
        - Protocolos
        - ABCs and Protocols for working with I/O
        - Funciones y decoradores
        - Ayudas de introspección

        - Constantes
        - Alias obsoletos
            - Alias de tipos integrados
            - Alias de tipos en **`collections`**
            - Alias a otros tipos concretos
            - Alias de ABCs de contenedores en **`collections.abc`**
            - Alias para ABCs asíncronos en **`collections.abc`**
            - Alias a otros ABCs en **`collections.abc`**
            - Alias de ABCs **`contextlib`**
    - Línea de tiempo de obsolescencia de características principales
- **`pydoc`** — Documentation generator and online help system
- Modo de desarrollo de Python
    - Efectos del modo de desarrollo de Python
    - Ejemplo de ResourceWarning
    - Ejemplo de error de descriptor de archivo incorrecto
- **`doctest`** — Test interactive Python examples
    - Uso simple: verificar ejemplos en docstrings
    - Uso Simple: Verificar ejemplos en un Archivo de Texto
    - Command-line Usage
    - Cómo funciona
        - ¿Qué docstrings son examinados?
        - ¿Cómo se reconocen los ejemplos de docstring?
        - ¿Cuál es el contexto de ejecución?
        - ¿Y las excepciones?
        - Banderas de Opción
        - Directivas
        - Advertencias
    - API básica
    - API de unittest
    - API avanzada

        - Objetos DocTest
        - Objetos *Example*
        - Objetos *DocTestFinder*
        - Objetos *DocTestParser*
        - TestResults objects
        - Objetos *DocTestRunner*
        - Objetos *OutputChecker*
    - Depuración
    - Plataforma improvisada
- **`unittest`** — Unit testing framework
    - Ejemplo sencillo
    - Interfaz de línea de comandos
        - Opciones de la línea de órdenes
    - Descubrimiento de pruebas
    - Organización del código de pruebas
    - Reutilización de código de prueba anterior
    - Omitir tests y fallos esperados
    - Distinguiendo iteraciones de tests empleando subtests
    - Clases y funciones
        - Casos de test
        - Agrupando tests
        - Cargando y ejecutando tests
            - load_tests protocolo
    - Instalaciones para clases y módulos
        - setUpClass y tearDownClass
        - setUpModule y tearDownModule
    - Manejo de señales
- **`unittest.mock`** — mock object library
    - Guía rápida
    - La clase Mock

    - Llamar a los objetos simulados
    - Eliminar atributos
    - Los nombres de los objetos simulados y el atributo *name*
    - Adjuntar objetos simulados como atributos

  - Parcheadores

    - patch
    - patch.object
    - patch.dict
    - patch.multiple
    - métodos start y stop de patch
    - parchear objetos incorporados (builtins)
    - TEST_PREFIX
    - Anidando decoradores patch
    - Dónde parchear
    - Parcheando descriptores y objetos proxy

  - MagicMock y el soporte de métodos mágicos

    - Simular métodos mágicos
    - Magic Mock

  - Ayudantes

    - sentinel
    - DEFAULT
    - call
    - create_autospec
    - ANY
    - FILTER_DIR
    - mock_open
    - Autoespecificación
    - Sellar objetos simulados

  - Order of precedence of **`side_effect`**, **`return_value`** and *wraps*

- **`unittest.mock`** — getting started

  - Usando mock
    - Mock patching methods
    - Mock for method calls on an object
    - Mocking classes
    - Nombrando tus mocks
    - Tracking all calls
    - Setting return values and attributes
    - Generar excepciones con mocks
    - Funciones de efectos secundarios e iterables
    - Iteradores asincrónicos de Mocking
    - El gestor de contexto asincrónico de Mocking
    - Creating a mock from an existing object
    - Uso de side_effect para devolver el contenido por archivo
  - Patch decorators
  - Further examples
    - Mocking de llamadas encadenadas
    - Mocking parcial
    - Mocking a generator method
    - Aplicar el mismo parche a cada método de prueba
    - Mocking unbound methods
    - Comprobación de varias llamadas con mock
    - Copiando con argumentos mutables
    - Nesting patches
    - Mocking a un diccionario usando MagickMock
    - Mock de subclases y sus atributos
    - Importaciones de Mocking con patch.dict
    - Seguimiento del orden de las llamadas y de las aserciones de llamadas menos detalladas
    - Coincidencia de argumentos más compleja
- **`test`** — Regression tests package for Python
  - Writing Unit Tests for the **`test`** package
  - Ejecución de pruebas utilizando la interfaz de línea de comandos

- **`test.support`** — Utilities for the Python test suite
- **`test.support.socket_helper`** — Utilities for socket tests
- **`test.support.script_helper`** — Utilities for the Python execution tests
- **`test.support.bytecode_helper`** — Support tools for testing correct bytecode generation
- **`test.support.threading_helper`** — Utilities for threading tests
- **`test.support.os_helper`** — Utilities for os tests
- **`test.support.import_helper`** — Utilities for import tests
- **`test.support.warnings_helper`** — Utilities for warnings tests

`typing` — Support for type hints

Added in version 3.5.

Source code: Lib/typing.py [https://github.com/python/cpython/tree/3.14/Lib/typing.py]

Nota

El entorno de ejecución de Python no exige anotaciones de tipos de funciones y variables. Pueden ser utilizadas por herramientas de terceros como type checkers, IDE, linters, etc.

Este módulo proporciona soporte en tiempo de ejecución para sugerencias de tipo.

Considere la función a continuación:

```
def surface_area_of_cube(edge_length: float) -> str:
    return f"The surface area of the cube is {6 * edge_l
```

La función `surface_area_of_cube` toma un argumento que se espera que sea una instancia de **`float`**, como lo indica type hint `edge_length: float`. Se espera que la función devuelva una instancia de **`str`**, como lo indica la sugerencia `-> str`.

Si bien las sugerencias de tipos pueden ser clases simples como **`float`** o **`str`**, también pueden ser más complejas. El módulo **`typing`** proporciona un vocabulario de sugerencias de tipos más avanzadas.

Con frecuencia se agregan nuevas funciones al módulo `typing`. El paquete typing_extensions [https://pypi.org/project/typing_extensions/]

proporciona versiones anteriores de estas nuevas funciones para versiones anteriores de Python.

Ver también

Typing cheat sheet [https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html]
Una visión general de los indicadores de tipo (hospedado por mypy docs, en inglés.)

Type System Reference section of the mypy docs [https://mypy.readthedocs.io/en/stable/index.html]
El sistema de tipos de Python es estandarizado por medio de las PEPs, así que esta referencia debe aplicarse a la mayoría de validadores de tipo de Python. (Algunas partes pueden referirse específicamente a mypy.)

Static Typing with Python [https://typing.python.org/en/latest/]
Documentación independiente del validador de tipos escrita por la comunidad que detalla las características del sistema de tipos, herramientas útiles relacionadas con el tipado y mejores prácticas.

Especificación del sistema de tipos de Python

The canonical, up-to-date specification of the Python type system can be found at Specification for the Python type system [https://typing.python.org/en/latest/spec/index.html].

Alias de tipo

Un alias de tipo se define usando la declaración `type`, el cual crea una instancia de `TypeAliasType`. En este ejemplo, `Vector` y `list[float]` serán tratados de manera equivalente a los validadores de tipo estático:

```
type Vector = list[float]
```

```
def scale(scalar: float, vector: Vector) -> Vector:
    return [scalar * num for num in vector]

# pasa las verificaciones de tipo; una lista de float ca
new_vector = scale(2.0, [1.0, -4.2, 5.4])
```

Los alias de tipo son útiles para simplificar firmas de tipo complejas. Por ejemplo:

```
from collections.abc import Sequence

type ConnectionOptions = dict[str, str]
type Address = tuple[str, int]
type Server = tuple[Address, ConnectionOptions]

def broadcast_message(message: str, servers: Sequence[Se
    ...

# El verificador de tipos estáticos tratará la firma de
# exactamente equivalente a esta.
def broadcast_message(
    message: str,
    servers: Sequence[tuple[tuple[str, int], dict[str, s
) -> None:
    ...
```

La declaración **`type`** es nueva en Python 3.12. Para compatibilidad con versiones anteriores, los alias de tipo también se pueden crear mediante una asignación simple.:

```
Vector = list[float]
```

O marcarse con **`TypeAlias`** para dejar explícito que se trata de un alias de tipo, no de una asignación de variable normal:

```
from typing import TypeAlias

Vector: TypeAlias = list[float]
```

NewType

Use la clase auxiliar **`NewType`** para crear tipos distintos:

```
from typing import NewType

UserId = NewType('UserId', int)
some_id = UserId(524313)
```

El validador estático de tipos tratará el nuevo tipo como si fuera una subclase del tipo original. Esto es útil para capturar errores lógicos:

```
def get_user_name(user_id: UserId) -> str:
    ...

# pasa la verificación de tipos
user_a = get_user_name(UserId(42351))

# falla la verificación de tipos; un int no es un UserI
user_b = get_user_name(-1)
```

Se pueden realizar todas las operaciones de `int` en una variable de tipo `UserId`, pero el resultado siempre será de tipo `int`. Esto permite pasar un `UserId` allí donde se espere un `int`, pero evitará la creación accidental de un `UserId` de manera incorrecta:

```
# 'output' es del tipo 'int', no 'UserId'
output = UserId(23413) + UserId(54341)
```

Tenga en cuenta que estas validaciones solo las aplica el validador de tipo estático. En tiempo de ejecución, la declaración `Derived = NewType('Derived', Base)` hará que `Derived` sea una clase que retorna inmediatamente cualquier parámetro que le pase. Eso significa que la expresión `Derived(some_value)` no crea una nueva clase ni introduce mucha sobrecarga más allá de la de una llamada de función regular.

Más concretamente, la expresión `some_value is Derived(some_value)` será siempre verdadera en tiempo de ejecución.

No es válido crear un subtipo de `Derived`:

```
from typing import NewType

UserId = NewType('UserId', int)

# Falla en runtime y no pasa la verificación de tipos
class AdminUserId(UserId): pass
```

Sin embargo, es posible crear un **`NewType`** basado en un `NewType` “derivado”:

```
from typing import NewType

UserId = NewType('UserId', int)

ProUserId = NewType('ProUserId', UserId)
```

y la comprobación de tipo para `ProUserId` funcionará como se espera.

Véase **PEP 484** [https://peps.python.org/pep-0484/] para más detalle.

Nota

Recuerde que el uso de alias de tipo implica que los dos tipos son *equivalentes* entre sí. Haciendo `type Alias = Original` provocará que el validador estático de tipos trate `Alias` como algo *exactamente equivalente* a `Original` en todos los casos. Esto es útil para cuando se quiera simplificar indicadores de tipo complejos.

En cambio, `NewType` declara un tipo que es *subtipo* de otro. Haciendo `Derived = NewType('Derived', Original)` hará que el Validador estático de tipos trate `Derived` como una *subclase* de `Original`, lo que implica que un valor de tipo `Original` no puede ser usado allí donde se espere un valor de tipo `Derived`. Esto es útil para prevenir errores lógicos con un coste de ejecución mínimo.

Added in version 3.5.2.

Distinto en la versión 3.10: `NewType` es ahora una clase en lugar de una función. Existe un costo de tiempo de ejecución adicional cuando se llama a `NewType` a través de una función normal.

Distinto en la versión 3.11: El rendimiento al llamar `NewType` ha sido restaurado a su nivel en Python 3.9.

Anotaciones en objetos invocables

Las funciones (u otros objetos callable) se pueden anotar utilizando **`collections.abc.Callable`** o el obsoleto **`typing.Callable`**. `Callable[[int], str]` significa una función que toma un único parámetro de tipo **`int`** y devuelve un **`str`**.

Por ejemplo:

```
from collections.abc import Callable, Awaitable

def feeder(get_next_item: Callable[[], str]) -> None:
    ...  # Body

def async_query(on_success: Callable[[int], None],
                on_error: Callable[[int, Exception], Nor
    ...  # Body

async def on_update(value: str) -> None:
    ...  # Body

callback: Callable[[str], Awaitable[None]] = on_update
```

The subscription syntax must always be used with exactly two values: the argument list and the return type. The argument list must be a list of types, a **`ParamSpec`**, **`Concatenate`**, or an ellipsis (`...`). The return type must be a single type.

Si se proporciona una elipsis como lista de argumentos, indica que sería aceptable un invocable con cualquier lista de parámetros arbitraria:

```
def concat(x: str, y: str) -> str:
    return x + y

x: Callable[..., str]
x = str      # OK
x = concat   # Also OK
```

`Callable` no puede expresar firmas complejas, como funciones que toman un número variádico de argumentos, overloaded functions o funciones que tienen parámetros que solo contienen palabras clave. Sin embargo, estas firmas se pueden expresar definiendo una clase `Protocol` con un método `__call__()`:

```
from collections.abc import Iterable
from typing import Protocol

class Combiner(Protocol):
    def __call__(self, *vals: bytes, maxlen: int | None

def batch_proc(data: Iterable[bytes], cb_results: Combi
    for item in data:
        ...

def good_cb(*vals: bytes, maxlen: int | None = None) ->
    ...
def bad_cb(*vals: bytes, maxitems: int | None) -> list[b
    ...

batch_proc([], good_cb)  # OK
batch_proc([], bad_cb)   # Error! El argumento 2 tiene u
                         # nombre y tipo diferentes en l
```

Los invocables que toman otros invocables como argumentos pueden indicar que sus tipos de parámetros dependen unos de otros utilizando `ParamSpec`. Además, si ese invocable agrega o elimina argumentos de otros invocables, se puede utilizar el operador

`Concatenate`. Toman la forma `Callable[ParamSpecVariable, ReturnType]` y `Callable[Concatenate[Arg1Type, Arg2Type, ..., ParamSpecVariable], ReturnType]` respectivamente.

Distinto en la versión 3.10: `Callable` ahora es compatible con **`ParamSpec`** y **`Concatenate`**. Consulte **PEP 612** [https://peps.python.org/pep-0612/] para obtener más información.

Ver también

La documentación de **`ParamSpec`** y **`Concatenate`** proporciona ejemplos de uso en `Callable`.

Genéricos

Dado que la información de tipo sobre los objetos guardados en contenedores no se puede inferir estáticamente de manera genérica, muchas clases de contenedor en la biblioteca estándar admiten suscripción para indicar los tipos esperados de elementos de contenedor.

```
from collections.abc import Mapping, Sequence

class Employee: ...

# Sequence[Employee] indica que todos los elementos de l
# deben ser instancias de "Employee".
# Mapping[str, str] indica que todas las claves y todos
# deben ser cadenas.
def notify_by_email(employees: Sequence[Employee],
                    overrides: Mapping[str, str]) -> Nor
```

Funciones y clases genéricas pueden ser parametrizadas usando type parameter syntax:

```
from collections.abc import Sequence

def first[T](l: Sequence[T]) -> T:  # La función es gené
```

```
    return l[0]
```

O utilizando directamente la fábrica **`TypeVar`**:

```
from collections.abc import Sequence
from typing import TypeVar

U = TypeVar('U')                  # Declara variable tip

def second(l: Sequence[U]) -> U:  # La función es genéri
    return l[1]
```

Distinto en la versión 3.12: El soporte sintáctico para genéricos es nuevo en Python 3.12.

Anotaciones en tuplas

En la mayoría de los contenedores de Python, el sistema de tipado supone que todos los elementos del contenedor serán del mismo tipo. Por ejemplo:

```
from collections.abc import Mapping

# El verificador de tipos inferirá que todos los element
x: list[int] = []

#Error del verificador de tipo: ``list`` solo acepta un
y: list[int, str] = [1, 'foo']

# El verificador de tipos inferirá que todas las claves
# y que todos los valores en ``z`` deben ser strings o i
z: Mapping[str, str | int] = {}
```

`list` solo acepta un argumento de tipo, por lo que un validador de tipos emitiría un error en la asignación `y` anterior. De manera similar, **`Mapping`** solo acepta dos argumentos de tipo: el primero indica el tipo de las claves y el segundo indica el tipo de los valores.

Sin embargo, a diferencia de la mayoría de los demás contenedores de Python, es común en el código idiomático de Python que las tuplas tengan elementos que no sean todos del mismo tipo. Por este motivo, las tuplas son un caso especial en el sistema de tipado de Python. **`tuple`** acepta *cualquier número* de argumentos de tipo:

```
# OK: ``x`` se asigna a una tupla de longitud 1 donde el
x: tuple[int] = (5,)

# OK: ``y`` se asigna a una tupla de longitud 2;
# el elemento 1 es un int, el elemento 2 es una cadena
y: tuple[int, str] = (5, "foo")

# Error: la anotación de tipo indica una tupla de longit
# pero ``z`` se ha asignado a una tupla de longitud 3
z: tuple[int] = (1, 2, 3)
```

To denote a tuple which could be of *any* length, and in which all elements are of the same type `T`, use the literal ellipsis `...`: `tuple[T, ...]`. To denote an empty tuple, use `tuple[()]`. Using plain `tuple` as an annotation is equivalent to using `tuple[Any, ...]`:

```
x: tuple[int, ...] = (1, 2)
# Estas reasignaciones están bien: ``tuple[int, ...]`` i
x = (1, 2, 3)
x = ()
# Esta reasignación es un error: todos los elementos en
x = ("foo", "bar")

# ``y`` solo se puede asignar a una tupla vacía
y: tuple[()] = ()

z: tuple = ("foo", "bar")
# Estas reasignaciones están bien: una ``tupla`` simple
z = (1, 2, 3)
z = ()
```

El tipo de objetos de clase

Una variable anotada con `C` puede aceptar un valor de tipo `C`. Por el contrario, una variable anotada con `type[C]` (o **`typing.Type[C]`** en desuso) puede aceptar valores que sean clases en sí mismos; específicamente, aceptará el *class object* de `C`. Por ejemplo:

```
a = 3         # Tiene tipo ``int``
b = int       # Tiene tipo ``type[int]``
c = type(a)   # Tambien tiene tipo ``type[int]``
```

Tenga en cuenta que `type[C]` es covariante:

```
class User: ...
class ProUser(User): ...
class TeamUser(User): ...

def make_new_user(user_class: type[User]) -> User:
    # ...
    return user_class()

make_new_user(User)        # OK
make_new_user(ProUser)     # También OK: ``type[ProUser]``
make_new_user(TeamUser)    # aún está bien
make_new_user(User())      # Error: se espera ``type[User]
make_new_user(int)         # Error: ``type[int]`` no es un
```

Los únicos parámetros legales para **`type`** son las clases, **`Any`**, variables de tipo y uniones de cualquiera de estos tipos. Por ejemplo:

```
def new_non_team_user(user_class: type[BasicUser | ProUs

new_non_team_user(BasicUser)   # OK
new_non_team_user(ProUser)     # OK
new_non_team_user(TeamUser)    # Error: ``type[TeamUser]`
                               # de ``type[BasicUser | Pr
new_non_team_user(User)        # También un error
```

`type[Any]` es equivalente a **`type`**, que es la raíz de la jerarquía de metaclases de Python.

Anotación de generadores y corrutinas

Se puede anotar un generador utilizando el tipo genérico `Generator[YieldType, SendType, ReturnType]`. Por ejemplo:

```
def echo_round() -> Generator[int, float, str]:
    sent = yield 0
    while sent >= 0:
        sent = yield round(sent)
    return 'Done'
```

Tenga en cuenta que, a diferencia de muchas otras clases genéricas en la biblioteca estándar, `SendType` de `Generator` se comporta de manera contravariante, no covariante o invariante.

Los parámetros `SendType` y `ReturnType` tienen como valor predeterminado **`None`**:

```
def infinite_stream(start: int) -> Generator[int]:
    while True:
        yield start
        start += 1
```

También es posible configurar estos tipos explícitamente:

```
def infinite_stream(start: int) -> Generator[int, None,
    while True:
        yield start
        start += 1
```

Los generadores simples que solo producen valores también se pueden anotar como que tienen un tipo de retorno de `Iterable[YieldType]` o `Iterator[YieldType]`:

```
def infinite_stream(start: int) -> Iterator[int]:
    while True:
        yield start
        start += 1
```

Los generadores asincrónicos se manejan de manera similar, pero no espere un argumento de tipo `ReturnType` (**`AsyncGenerator[YieldType, SendType]`**). El argumento `SendType` tiene como valor predeterminado **`None`**, por lo que las siguientes definiciones son equivalentes:

```
async def infinite_stream(start: int) -> AsyncGenerator[
    while True:
        yield start
        start = await increment(start)

async def infinite_stream(start: int) -> AsyncGenerator[
    while True:
        yield start
        start = await increment(start)
```

Al igual que en el caso sincrónico, también están disponibles **`AsyncIterable[YieldType]`** y **`AsyncIterator[YieldType]`**:

```
async def infinite_stream(start: int) -> AsyncIterator[i
    while True:
        yield start
        start = await increment(start)
```

Las corrutinas se pueden anotar utilizando **`Coroutine[YieldType, SendType, ReturnType]`**. Los argumentos genéricos corresponden a los de **`Generator`**, por ejemplo:

```
from collections.abc import Coroutine
c: Coroutine[list[str], str, int]  # Algunas corrutinas
x = c.send('hi')                   # El tipo inferido de
async def bar() -> None:
    y = await c                    # El tipo inferido de
```

Tipos genéricos definidos por el usuario

Una clase definida por el usuario puede ser definida como una clase genérica.

```
from logging import Logger

class LoggedVar[T]:
    def __init__(self, value: T, name: str, logger: Logg
        self.name = name
        self.logger = logger
        self.value = value

    def set(self, new: T) -> None:
        self.log('Set ' + repr(self.value))
        self.value = new

    def get(self) -> T:
        self.log('Get ' + repr(self.value))
        return self.value

    def log(self, message: str) -> None:
        self.logger.info('%s: %s', self.name, message)
```

Esta sintaxis indica que la clase `LoggedVar` está parametrizada en torno a un único type variable `T` . Esto también hace que `T` sea válido como tipo dentro del cuerpo de la clase.

Las clases genéricas heredan implícitamente de **`Generic`**. Para compatibilidad con Python 3.11 y versiones anteriores, también es posible heredar explícitamente de **`Generic`** para indicar una clase genérica:

```
from typing import TypeVar, Generic

T = TypeVar('T')

class LoggedVar(Generic[T]):
    ...
```

Las clases genéricas tienen métodos **`__class_getitem__()`**, lo que significa que se pueden parametrizar en tiempo de ejecución (por ejemplo, `LoggedVar[int]` a continuación):

```
from collections.abc import Iterable

def zero_all_vars(vars: Iterable[LoggedVar[int]]) -> Nor
    for var in vars:
        var.set(0)
```

Un tipo genérico puede tener un numero cualquiera de variables de tipo. Se permiten todas las variaciones de **`TypeVar`** para ser usadas como parámetros de un tipo genérico:

```
from typing import TypeVar, Generic, Sequence

class WeirdTrio[T, B: Sequence[bytes], S: (int, str)]:
    ...

OldT = TypeVar('OldT', contravariant=True)
OldB = TypeVar('OldB', bound=Sequence[bytes], covariant=
OldS = TypeVar('OldS', int, str)

class OldWeirdTrio(Generic[OldT, OldB, OldS]):
    ...
```

Cada argumento de variable de tipo de **`Generic`** debe ser distinto. Por lo tanto, esto no es válido:

```
from typing import TypeVar, Generic
...

class Pair[M, M]:  # SyntaxError
    ...

T = TypeVar('T')

class Pair(Generic[T, T]):   # INVALID
    ...
```

Las clases genéricas también pueden heredar de otras clases:

```
from collections.abc import Sized
```

```
class LinkedList[T](Sized):
    ...
```

Al heredar de clases genéricas, algunos parámetros de tipo podrían ser fijos:

```
from collections.abc import Mapping

class MyDict[T](Mapping[str, T]):
    ...
```

En este caso `MyDict` tiene un solo parámetro, `T`.

El uso de una clase genérica sin especificar parámetros de tipo supone **`Any`** para cada posición. En el siguiente ejemplo, `MyIterable` no es genérico, sino que hereda implícitamente de `Iterable[Any]`:

```
from collections.abc import Iterable

class MyIterable(Iterable): # Igual que Iterable[Any]
    ...
```

También se admiten alias de tipos genéricos definidos por el usuario. Ejemplos:

```
from collections.abc import Iterable

type Response[S] = Iterable[S] | int

# Return type here is same as Iterable[str] | int
def response(query: str) -> Response[str]:
    ...

type Vec[T] = Iterable[tuple[T, T]]

def inproduct[T: (int, float, complex)](v: Vec[T]) -> T:
    return sum(x*y for x, y in v)
```

Para compatibilidad con versiones anteriores, también se pueden crear alias de tipos genéricos mediante una asignación simple:

```
from collections.abc import Iterable
from typing import TypeVar

S = TypeVar("S")
Response = Iterable[S] | int
```

Distinto en la versión 3.7: **Generic** ya no posee una metaclase personalizable.

Distinto en la versión 3.12: La compatibilidad sintáctica con genéricos y alias de tipo es una novedad en la versión 3.12. Antes, las clases genéricas debían heredar explícitamente de **Generic** o contener una variable de tipo en una de sus bases.

User-defined generics for parameter expressions are also supported via parameter specification variables in the form `[**P]`. The behavior is consistent with type variables" described above as parameter specification variables are treated by the **typing** module as a specialized type variable. The one exception to this is that a list of types can be used to substitute a **ParamSpec**:

```
>>> class Z[T, **P]: ...  # T es un TypeVar; P es un Par
...
>>> Z[int, [dict, float]]
__main__.Z[int, [dict, float]]
```

También se pueden crear clases genéricas sobre **ParamSpec** utilizando herencia explícita de **Generic**. En este caso, no se utiliza `**`:

```
from typing import ParamSpec, Generic

P = ParamSpec('P')

class Z(Generic[P]):
    ...
```

Otra diferencia entre **TypeVar** y **ParamSpec** es que una variable genérica con una sola especificación de parámetros aceptará listas de parámetros en los formatos `X[[Type1, Type2, ...]]` y también `X[Type1, Type2, ...]` por razones estéticas. Internamente, el último se convierte al primero, por lo que los siguientes son equivalentes:

```
>>> class X[**P]: ...
...
>>> X[int, str]
__main__.X[[int, str]]
>>> X[[int, str]]
__main__.X[[int, str]]
```

Tenga en cuenta que los genéricos con **`ParamSpec`** pueden no tener `__parameters__` correctos después de la sustitución en algunos casos porque están destinados principalmente a la verificación de tipos estáticos.

Distinto en la versión 3.10: **`Generic`** ahora se puede parametrizar sobre expresiones de parámetros. Consulte **`ParamSpec`** y **PEP 612** [https://peps.python.org/pep-0612/] para obtener más detalles.

A user-defined generic class can have ABCs as base classes without a metaclass conflict. Generic metaclasses are not supported. The outcome of parameterizing generics is cached, and most types in the **`typing`** module are hashable and comparable for equality.

El tipo **`Any`**

Un caso especial de tipo es **`Any`**. Un Validador estático de tipos tratará cualquier tipo como compatible con **`Any`**, y **`Any`** como compatible con todos los tipos.

Esto significa que es posible realizar cualquier operación o llamada a un método en un valor de tipo **`Any`** y asignarlo a cualquier variable:

```
from typing import Any

a: Any = None
a = []          # OK
```

```
a = 2              # OK

s: str = ''
s = a              # OK

def foo(item: Any) -> int:
    # Pasa la verificación de tipos; 'item' puede ser de
    # y ese tipo puede tener el método 'bar'
    item.bar()
    ...
```

Nótese que no se realiza comprobación de tipo cuando se asigna un valor de tipo `Any` a un tipo más preciso. Por ejemplo, el Validador estático de tipos no reportó ningún error cuando se asignó `a` a `s`, aún cuando se declaró `s` como de tipo `str` y recibió un valor `int` en tiempo de ejecución!

Además, todas las funciones sin un tipo de retorno o tipos en los parámetros serán asignadas implícitamente a `Any` por defecto:

```
def legacy_parser(text):
    ...
    return data

# Un verificador de tipos estáticos tratará
# lo anterior como si tuviera la misma firma que:
def legacy_parser(text: Any) -> Any:
    ...
    return data
```

Este comportamiento permite que `Any` sea usado como una *vía de escape* cuando es necesario mezclar código tipado estática y dinámicamente.

Compárese el comportamiento de `Any` con el de `object`. De manera similar a `Any`, todo tipo es un subtipo de `object`. Sin embargo, en oposición a `Any`, lo contrario no es cierto: `object` *no* es un subtipo de ningún otro tipo.

Esto implica que cuando el tipo de un valor es `object`, un validador de tipos rechazará prácticamente todas las operaciones con él, y al asignarlo a una variable (o usarlo como valor de retorno) de un tipo más preciso será un error de tipo. Por ejemplo:

```
def hash_a(item: object) -> int:
    # No pasa la comprobación de tipos; un objeto no tie
    item.magic()
    ...

def hash_b(item: Any) -> int:
    # Pasa la veficiación de tipos
    item.magic()
    ...

# Pasa la verificación de tipo, ya que los ints y str so
hash_a(42)
hash_a("foo")

# Pasa la verificación de tipo, ya que Any es compatible
hash_b(42)
hash_b("foo")
```

Úsese **`object`** para indicar que un valor puede ser de cualquier tipo de manera segura. Úsese **`Any`** para indicar que un valor es de tipado dinámico.

Subtipado nominal vs estructural

Inicialmente, el **PEP 484** [https://peps.python.org/pep-0484/] definió el uso de *subtipado nominal* para el sistema de tipado estático de Python. Esto implica que una clase `A` será permitida allí donde se espere una clase `B` si y solo si `A` es una subclase de `B`.

Este requisito también se aplicaba anteriormente a clases base abstractas (ABC), tales como **`Iterable`**. El problema con esta estrategia es que una clase debía de ser marcada explícitamente para proporcionar esta funcionalidad, lo que resulta poco *pythónico* (idiomático) y poco ajustado a lo que uno normalmente haría en un código Python tipado dinámicamente. Por ejemplo, esto sí se ajusta al **PEP 484** [https://peps.python.org/pep-0484/]:

```
from collections.abc import Sized, Iterable, Iterator

class Bucket(Sized, Iterable[int]):
    ...
    def __len__(self) -> int: ...
    def __iter__(self) -> Iterator[int]: ...
```

PEP 544 [https://peps.python.org/pep-0544/] solves this problem by allowing users to write the above code without explicit base classes in the class definition, allowing `Bucket` to be implicitly considered a subtype of both `Sized` and `Iterable[int]` by static type checkers. This is known as *structural subtyping* (or static duck-typing):

```
from collections.abc import Iterator, Iterable

class Bucket:  # Nota: sin clases base
    ...
    def __len__(self) -> int: ...
    def __iter__(self) -> Iterator[int]: ...

def collect(items: Iterable[int]) -> int: ...
result = collect(Bucket())  # Pasa la verificación de ti
```

Asimismo, creando subclases de la clase especial **`Protocol`**, el usuario puede definir nuevos protocolos personalizados y beneficiarse del tipado estructural (véanse los ejemplos de abajo).

Contenido del módulo

El módulo `typing` define las siguientes clases, funciones y decoradores.

Primitivos especiales de tipado

Tipos especiales

Estos pueden ser usados como tipos en anotaciones. No soportan suscripción usando `[]`.

typing.Any

Tipo especial que indica un tipo sin restricciones.

- Todos los tipos son compatibles con **`Any`**.
- **`Any`** es compatible con todos los tipos.

Distinto en la versión 3.11: Ahora es posible utilizar **`Any`** como una clase base. Esto puede ser útil para evitar errores del validador de tipos con clases que pueden hacer uso del *duck typing* en cualquier punto, o que sean altamente dinámicas.

typing.AnyStr

Una variables de tipo restringida.

Definición:

```
AnyStr = TypeVar('AnyStr', str, bytes)
```

`AnyStr` está pensado para ser utilizado por funciones que pueden aceptar argumentos **`str`** o **`bytes`** pero que no puedan permitir que los dos se mezclen.

Por ejemplo:

```
def concat(a: AnyStr, b: AnyStr) -> AnyStr:
    return a + b

concat("foo", "bar")    # OK, output tiene tipo 'st
concat(b"foo", b"bar")  # OK, output tiene tipo 'by
concat("foo", b"bar")   # Error, no se puede mezcla
```

Tenga en cuenta que, a pesar de su nombre, `AnyStr` no tiene nada que ver con el tipo **`Any`**, ni significa "cualquier cadena de caracteres". En particular, `AnyStr` y `str | bytes` son diferentes entre sí y tienen diferentes casos de uso:

```
# Uso inválido de AnyStr:
# La variable de tipo se usa solo una vez en la fir
# por lo que el verificador de tipos no puede resol
def greet_bad(cond: bool) -> AnyStr:
    return "hi there!" if cond else b"greetings!"

# La mejor manera de anotar esta función:
def greet_proper(cond: bool) -> str | bytes:
    return "hi there!" if cond else b"greetings!"
```

Deprecated since version 3.13, will be removed in version 3.18: Obsoleto en favor del nuevo type parameter syntax. Utilice `class A[T: (str, bytes)]: ...` en lugar de importar `AnyStr`. Consulte **PEP 695** [https://peps.python.org/pep-0695/] para obtener más detalles.

En Python 3.16, `AnyStr` se eliminará de `typing.__all__` y se emitirán advertencias de desuso en tiempo de ejecución cuando se acceda a él o se importe desde `typing`. `AnyStr` se eliminará de `typing` en Python 3.18.

typing.LiteralString

Tipo especial que incluye sólo cadenas de caracteres literales.

Cualquier cadena de caracteres literal es compatible con `LiteralString`, al igual que cualquier otro `LiteralString`. Sin embargo, un objeto cuyo tipo sea simplemente `str` no lo es. Una cadena de caracteres creada mediante la composición de objetos cuyo tipo sea `LiteralString` también es aceptable como `LiteralString`.

Por ejemplo:

```
def run_query(sql: LiteralString) -> None:
    ...

def caller(arbitrary_string: str, literal_string: L
    run_query("SELECT * FROM students")  # OK
    run_query(literal_string)  # OK
    run_query("SELECT * FROM " + literal_string)  #
    run_query(arbitrary_string)  # type checker err
    run_query(  # type checker error
        f"SELECT * FROM students WHERE name = {arbi
    )
```

`LiteralString` es útil para API sensibles en las que cadenas de caracteres arbitrarias generadas por el usuario podrían generar problemas. Por ejemplo, los dos casos anteriores que generan errores de verificación de tipos podrían ser vulnerables a un ataque de inyección SQL.

Véase PEP 675 [https://peps.python.org/pep-0675/] para más detalle.

Added in version 3.11.

typing.Never
typing.NoReturn

`Never` y **`NoReturn`** representan bottom type [https://en.wikipedia.org/wiki/Bottom_type], un tipo que no tiene miembros.

Se pueden utilizar para indicar que una función nunca retorna, como **`sys.exit()`**:

```
from typing import Never  # o NoReturn

def stop() -> Never:
    raise RuntimeError('no way')
```

O definir una función que nunca debe llamarse, ya que no hay argumentos válidos, como **`assert_never()`**:

```
from typing import Never  # o NoReturn

def never_call_me(arg: Never) -> None:
    pass

def int_or_str(arg: int | str) -> None:
    never_call_me(arg)  # error del verificador de
    match arg:
        case int():
            print("It's an int")
        case str():
            print("It's a str")
        case _:
            never_call_me(arg)  # OK, arg es del ti
```

`Never` y **`NoReturn`** tienen el mismo significado en el sistema de tipos y los verificadores de tipos estáticos los tratan a ambos de manera equivalente.

Added in version 3.6.2: Se agregó **`NoReturn`**.

Added in version 3.11: Se agregó **`Never`**.

typing.Self

Tipo especial que representa la clase capturada actual.

Por ejemplo:

```
from typing import Self, reveal_type

class Foo:
    def return_self(self) -> Self:
        ...
        return self

class SubclassOfFoo(Foo): pass

reveal_type(Foo().return_self())  # Revealed type i
reveal_type(SubclassOfFoo().return_self())  # Tipo 
```

Esta anotación es semánticamente equivalente a lo siguiente, aunque de una manera más sucinta:

```
from typing import TypeVar

Self = TypeVar("Self", bound="Foo")

class Foo:
    def return_self(self: Self) -> Self:
        ...
        return self
```

En general, si algo devuelve `self`, como en los ejemplos anteriores, se debe utilizar `Self` en la anotación del retorno. Si `Foo.return_self` se anotó como que devuelve `"Foo"`, entonces el validador de tipos inferiría que el objeto devuelto desde `SubclassOfFoo.return_self` es del tipo `Foo` en lugar de `SubclassOfFoo`.

Otros casos de uso comunes incluyen:

- **`classmethod`** usados como constructores alternativos y retornan instancias del parámetro `cls`.
- Anotar un método **`__enter__()`** que retorna self.

No debe utilizar `Self` como anotación de retorno si no se garantiza que el método devuelva una instancia de una subclase cuando la clase sea heredada:

```
class Eggs:
    # Self would be an incorrect return annotation
    # as the object returned is always an instance
    # even in subclasses
    def returns_eggs(self) -> "Eggs":
        return Eggs()
```

Véase **PEP 673** [https://peps.python.org/pep-0673/] para más detalle.

Added in version 3.11.

typing.TypeAlias

Anotación especial para declarar explícitamente un alias de tipo.

Por ejemplo:

```
from typing import TypeAlias

Factors: TypeAlias = list[int]
```

`TypeAlias` es particularmente útil en versiones anteriores de Python para anotar alias que utilizan referencias para versiones posteriores, ya que puede ser difícil para los validadores de tipos distinguirlos de las asignaciones de variables normales:

```
from typing import Generic, TypeAlias, TypeVar

T = TypeVar("T")

# "Box" aún no existe,
# entonces tenemos que usar comillas para la refere
# Usando ``TypeAlias`` le dice al verificador de ti
# no una asignacin de variable a un string.
BoxOfStrings: TypeAlias = "Box[str]"

class Box(Generic[T]):
    @classmethod
    def make_box_of_strings(cls) -> BoxOfStrings: .
```

Ver **PEP 613** [https://peps.python.org/pep-0613/] para más detalle.

Added in version 3.10.

Obsoleto desde la versión 3.12: **`TypeAlias`** ha sido descontinuado en favor de la declaración **`type`**, la cual crea instancias de **`TypeAliasType`** y que admite de forma nativa referencias de versiones posteriores de Python. Tenga en cuenta que, si bien **`TypeAlias`** y **`TypeAliasType`** tienen propósitos similares y tienen nombres similares, son distintos y el último no es el tipo del primero. La eliminación de **`TypeAlias`** no está prevista actualmente, pero se recomienda a los usuarios que migren a las declaraciones **`type`**.

Formas especiales

Estos se pueden utilizar como tipos en anotaciones. Todos admiten la suscripción mediante `[]`, pero cada uno tiene una sintaxis única.

class typing.Union

Tipo de unión; `Union[X, Y]` es equivalente a `X | Y` y significa X o Y.

Para definir una unión, use p. ej. `Union[int, str]` o la abreviatura `int | str`. Se recomienda el uso de la abreviatura. Detalles:

- Los argumentos deben ser tipos y haber al menos uno.
- Las uniones de uniones se simplifican (se aplanan), p. ej.:

  ```
  Union[Union[int, str], float] == Union[int, st
  ```

  However, this does not apply to unions referenced through a type alias, to avoid forcing evaluation of the underlying **`TypeAliasType`**:

  ```
  type A = Union[int, str]
  Union[A, float] != Union[int, str, float]
  ```

- Las uniones con un solo argumento se eliminan, p. ej.:

  ```
  Union[int] == int  # El constructor de hecho r
  ```

- Argumentos repetidos se omiten, p. ej.:

  ```
  Union[int, str, int] == Union[int, str] == int
  ```

- Cuando se comparan uniones, el orden de los argumentos se ignoran, p. ej.:

  ```
  Union[int, str] == Union[str, int]
  ```

- No es posible crear una subclase o instanciar un `Union`.
- No es posible escribir `Union[X][Y]`.

Distinto en la versión 3.7: No elimina subclases explícitas de una unión en tiempo de ejecución.

Distinto en la versión 3.10: Las uniones ahora se pueden escribir como `X | Y`. Consulte union type expressions.

Distinto en la versión 3.14: **`types.UnionType`** is now an alias for **`Union`**, and both `Union[int, str]` and `int | str` create instances of the same class. To check whether an object is a `Union` at runtime, use `isinstance(obj, Union)`. For compatibility with earlier versions of Python, use `get_origin(obj) is typing.Union or get_origin(obj) is types.UnionType`.

typing.Optional

`Optional[X]` es equivalente a `X | None` (o `Union[X, None]`).

Nótese que no es lo mismo que un argumento opcional, que es aquel que tiene un valor por defecto. Un argumento opcional con un valor por defecto no necesita el indicador `Optional` en su anotación de tipo simplemente por que sea opcional. Por ejemplo:

```
def foo(arg: int = 0) -> None:
    ...
```

Por otro lado, si se permite un valor `None`, es apropiado el uso de `Optional`, independientemente de que sea opcional o no. Por ejemplo:

```
def foo(arg: Optional[int] = None) -> None:
    ...
```

Distinto en la versión 3.10: Optional ahora se puede escribir como `X | None`. Consulte union type expressions.

typing.Concatenate

Forma especial para anotar funciones de orden superior.

`Concatenate` can be used in conjunction with Callable and **`ParamSpec`** to annotate a higher-order callable which adds, removes, or transforms parameters of another callable. Usage is in the form `Concatenate[Arg1Type, Arg2Type, ..., ParamSpecVariable]`. `Concatenate` is valid when used in Callable type hints and when instantiating user-defined generic classes with **`ParamSpec`** parameters. The last parameter to `Concatenate` must be a **`ParamSpec`** or ellipsis (`...`).

Por ejemplo, para anotar un decorador `with_lock` que proporciona un **`threading.Lock`** a la función decorada, `Concatenate` puede usarse para indicar que `with_lock` espera un invocable que toma un `Lock` como primer argumento y retorna un invocable con un tipo de firma diferente. En este caso, el **`ParamSpec`** indica que los tipos de parámetros de los invocables retornados dependen de los tipos de parámetros de los invocables que se pasan en

```
from collections.abc import Callable
from threading import Lock
from typing import Concatenate

# Utilice este bloqueo para garantizar que solo un
# en cualquier momento.
my_lock = Lock()

def with_lock[**P, R](f: Callable[Concatenate[Lock,
    '''A type-safe decorator which provides a lock.
    def inner(*args: P.args, **kwargs: P.kwargs) ->
        # Proporcione el bloqueo como primer argume
        return f(my_lock, *args, **kwargs)
    return inner

@with_lock
def sum_threadsafe(lock: Lock, numbers: list[float]
    '''Add a list of numbers together in a thread-s
    with lock:
        return sum(numbers)

# No necesitamos pasar por la cerradura nosotros mi
sum_threadsafe([1.1, 2.2, 3.3])
```

Added in version 3.10.

Ver también

- **PEP 612** [https://peps.python.org/pep-0612/] - Variables de especificación de parámetros (el PEP que introdujo `ParamSpec` y `Concatenate`)
- **`ParamSpec`**
- Anotaciones en objetos invocables

typing.Literal

Tipo especial que solo incluye cadenas literales.

`Literal` se puede utilizar para indicar a los validadores de tipos que el objeto anotado tiene un valor equivalente a uno de los literales proporcionados.

Por ejemplo:

```
def validate_simple(data: Any) -> Literal[True]:  #
    ...

type Mode = Literal['r', 'rb', 'w', 'wb']
def open_helper(file: str, mode: Mode) -> str:
    ...

open_helper('/some/path', 'r')      # Pasa el verif
open_helper('/other/path', 'typo')  # Error en el v
```

`Literal[...]` no puede ser derivado. En tiempo de ejecución, se permite un valor arbitrario como argumento de tipo de `Literal[...]`, pero los validadores de tipos pueden imponer sus restricciones. Véase **PEP 585** [https://peps.python.org/pep-0585/] para más detalles sobre tipos literales.

Additional details:

- The arguments must be literal values and there must be at least one.

- Nested `Literal` types are flattened, e.g.:

  ```
  assert Literal[Literal[1, 2], 3] == Literal[1,
  ```

  However, this does not apply to `Literal` types referenced through a type alias, to avoid forcing evaluation of the underlying **`TypeAliasType`**:

  ```
  type A = Literal[1, 2]
  assert Literal[A, 3] != Literal[1, 2, 3]
  ```

- Argumentos repetidos se omiten, p. ej.:

  ```
  assert Literal[1, 2, 1] == Literal[1, 2]
  ```

- When comparing literals, the argument order is ignored, e.g.:

  ```
  assert Literal[1, 2] == Literal[2, 1]
  ```

- You cannot subclass or instantiate a `Literal`.

- You cannot write `Literal[X][Y]`.

Added in version 3.8.

Distinto en la versión 3.9.1: `Literal` ahora elimina los parámetros duplicados. Las comparaciones de igualdad de los objetos `Literal` ya no dependen del orden. Los objetos `Literal` ahora lanzarán una excepción **`TypeError`** durante las comparaciones de igualdad si uno de sus parámetros no es hashable.

typing.ClassVar

Construcción especial para tipado para marcar variables de clase.

Tal y como introduce **PEP 526** [https://peps.python.org/pep-0526/], una anotación de variable rodeada por ClassVar indica que la intención de un atributo dado es ser usado como variable de clase y que no debería ser modificado en las instancias de esa misma clase. Uso:

```
class Starship:
    stats: ClassVar[dict[str, int]] = {} # variable
    damage: int = 10                     # variable
```

`ClassVar` solo acepta tipos y no admite más niveles de subíndices.

`ClassVar` no es un clase en sí misma, y no debe ser usado con `isinstance()` o `issubclass()`. **`ClassVar`** no modifica el comportamiento de Python en tiempo de ejecución pero puede ser utilizado por validadores de terceros. Por ejemplo, un validador de tipos puede marcar el siguiente código como erróneo:

```
enterprise_d = Starship(3000)
enterprise_d.stats = {} # Error, establece variable
Starship.stats = {}     # Ésto está OK
```

Added in version 3.5.3.

Distinto en la versión 3.13: Ahora `ClassVar` se puede anidar en `Final` y viceversa.

typing.Final

Construcción de tipado especial para indicar nombres finales a los validadores de tipos.

Los nombres finales no se pueden reasignar en ningún ámbito. Los nombres finales declarados en ámbitos de clase no se pueden anular en subclases.

Por ejemplo:

```
MAX_SIZE: Final = 9000
MAX_SIZE += 1  # Error reportado por un verificador

class Connection:
    TIMEOUT: Final[int] = 10
```

```
class FastConnector(Connection):
    TIMEOUT = 1  # Error reportado por un verificad
```

No hay comprobación en tiempo de ejecución para estas propiedades. Véase **PEP 591** [https://peps.python.org/pep-0591/] para más detalles.

Added in version 3.8.

Distinto en la versión 3.13: Ahora `Final` se puede anidar en `ClassVar` y viceversa.

typing.Required

Construcción de tipado especial para marcar una clave `TypedDict` como requerida.

Esto es útil principalmente para TypedDicts `total=False`. Vea `TypedDict` y **PEP 655** [https://peps.python.org/pep-0655/] para obtener más detalles.

Added in version 3.11.

typing.NotRequired

Construcción de tipado especial para marcar una clave `TypedDict` como potencialmente faltante.

Véase `TypedDict` y **PEP 655** [https://peps.python.org/pep-0655/] para más detalle.

Added in version 3.11.

typing.ReadOnly

Una construcción de tipificación especial para marcar un elemento de un `TypedDict` como de solo lectura.

Por ejemplo:

```
class Movie(TypedDict):
   title: ReadOnly[str]
   year: int
```

```
def mutate_movie(m: Movie) -> None:
   m["year"] = 1999  # allowed
   m["title"] = "The Matrix"  # error del verificad
```

No hay ninguna comprobación en tiempo de ejecución para esta propiedad.

Vea **`TypedDict`** y **PEP 705** [https://peps.python.org/pep-0705/] para más detalle.

Added in version 3.13.

typing.Annotated

Forma de escritura especial para agregar metadatos específicos del contexto a una anotación.

Agregue metadatos `x` a un tipo `T` determinado mediante la anotación `Annotated[T, x]`. Los metadatos agregados mediante `Annotated` pueden usarse con herramientas de análisis estático o en tiempo de ejecución. En tiempo de ejecución, los metadatos se almacenan en un atributo **`__metadata__`**.

Si una biblioteca o herramienta encuentra una anotación `Annotated[T, x]` y no tiene una lógica especial para los metadatos, debe ignorar los metadatos y simplemente tratar la anotación como `T`. Como tal, `Annotated` puede ser útil para el código que desea usar anotaciones para fines fuera del sistema de tipado estático de Python.

Using `Annotated[T, x]` as an annotation still allows for static typechecking of `T`, as type checkers will simply ignore the metadata `x`. In this way, `Annotated` differs from the **`@no_type_check`** decorator, which can also be used for adding annotations outside the scope of the typing system, but completely disables typechecking for a function or class.

La responsabilidad de interpretar los metadatos recae en la herramienta o biblioteca que encuentra una anotación `Annotated`. Una herramienta o biblioteca que encuentra un tipo `Annotated` puede examinar los elementos de metadatos para determinar si son de interés (por ejemplo, utilizando **`isinstance()`**).

Annotated[<type>, <metadata>]

A continuación se muestra un ejemplo de cómo podría utilizar `Annotated` para agregar metadatos a las anotaciones de tipo si estuviera realizando un análisis de rango:

```
@dataclass
class ValueRange:
    lo: int
    hi: int

T1 = Annotated[int, ValueRange(-10, 5)]
T2 = Annotated[T1, ValueRange(-20, 3)]
```

The first argument to `Annotated` must be a valid type. Multiple metadata elements can be supplied as `Annotated` supports variadic arguments. The order of the metadata elements is preserved and matters for equality checks:

```
@dataclass
class ctype:
     kind: str

a1 = Annotated[int, ValueRange(3, 10), ctype("char"
a2 = Annotated[int, ctype("char"), ValueRange(3, 10

assert a1 != a2  # Order matters
```

Depende de la herramienta que consume las anotaciones decidir si el cliente puede agregar varios elementos de metadatos a una anotación y cómo fusionar esas anotaciones.

Los tipos anidados `Annotated` se aplanan. El orden de los elementos de metadatos comienza con la anotación más interna:

```
assert Annotated[Annotated[int, ValueRange(3, 10)],
    int, ValueRange(3, 10), ctype("char")
]
```

However, this does not apply to `Annotated` types referenced through a type alias, to avoid forcing evaluation of the underlying `TypeAliasType`:

```
type From3To10[T] = Annotated[T, ValueRange(3, 10)]
assert Annotated[From3To10[int], ctype("char")] != 
   int, ValueRange(3, 10), ctype("char")
]
```

Los elementos de metadatos duplicados no se eliminan:

```
assert Annotated[int, ValueRange(3, 10)] != Annotat
    int, ValueRange(3, 10), ValueRange(3, 10)
]
```

`Annotated` se puede utilizar con alias anidados y genéricos:

```
@dataclass
class MaxLen:
    value: int

type Vec[T] = Annotated[list[tuple[T, T]], Ma

# Cuando se utiliza en una anotación de tipo,
# ``Annotated[list[tuple[int, int]], MaxLen(1
type V = Vec[int]
```

No se puede utilizar `Annotated` con un `TypeVarTuple` descomprimido:

```
type Variadic[*Ts] = Annotated[*Ts, Ann1] = Annotat
```

where `T1`, `T2`, ... are `TypeVars`. This is invalid as only one type should be passed to Annotated.

De forma predeterminada, `get_type_hints()` elimina los metadatos de las anotaciones. Pase `include_extras=True` para conservar los metadatos:

```
>>> from typing import Annotated, get_type_hi
>>> def func(x: Annotated[int, "metadata"]) -
...
>>> get_type_hints(func)
{'x': <class 'int'>, 'return': <class 'NoneTy
>>> get_type_hints(func, include_extras=True)
{'x': typing.Annotated[int, 'metadata'], 'ret
```

En tiempo de ejecución, los metadatos asociados con un tipo `Annotated` se pueden recuperar a través del atributo **`__metadata__`**:

```
>>> from typing import Annotated
>>> X = Annotated[int, "very", "important", "
>>> X
typing.Annotated[int, 'very', 'important', 'm
>>> X.__metadata__
('very', 'important', 'metadata')
```

If you want to retrieve the original type wrapped by `Annotated`, use the **`__origin__`** attribute:

```
>>> from typing import Annotated, get_origin
>>> Password = Annotated[str, "secret"]
>>> Password.__origin__
<class 'str'>
```

Tenga en cuenta que el uso de **`get_origin()`** devolverá el mismo `Annotated`:

```
>>> get_origin(Password)
typing.Annotated
```

Ver también

PEP 593 [https://peps.python.org/pep-0593/] - Anotaciones flexibles de funciones y variables

El PEP introduce `Annotated` en la biblioteca estándar.

Added in version 3.9.

typing.TypeIs

Construcción de tipificación especial para marcar funciones de predicado de tipo definido por el usuario.

`TypeIs` se puede utilizar para anotar el tipo de retorno de una función de predicado de tipo definida por el usuario. `TypeIs` solo acepta un único argumento de tipo. En tiempo de ejecución, las funciones marcadas de esta manera deben devolver un valor booleano y tomar al menos un argumento posicional.

`TypeIs` tiene como objetivo beneficiar a *type narrowing*, una técnica utilizada por los verificadores de tipos estáticos para determinar un tipo más preciso de una expresión dentro del flujo de código de un programa. Por lo general, la restricción de tipos se realiza analizando el flujo de código condicional y aplicando la restricción a un bloque de código. La expresión condicional aquí a veces se denomina «predicado de tipo»:

```
def is_str(val: str | float):
    # "isinstance" predicado de tipo
    if isinstance(val, str):
        # Tipo de ``val`` se reduce a ``str``
        ...
    else:
        # De lo contrario, el tipo de ``val`` se li
        ...
```

A veces sería conveniente utilizar una función booleana definida por el usuario como predicado de tipo. Dicha función debería utilizar `TypeIs[...]` o **`TypeGuard`** como su tipo de retorno para alertar a los verificadores de tipos estáticos sobre esta intención. `TypeIs` suele tener un comportamiento más intuitivo que `TypeGuard`, pero no se puede utilizar cuando los tipos de entrada y salida son incompatibles (por ejemplo, `list[object]` a `list[int]`) o cuando la función no devuelve `True` para todas las instancias del tipo restringido.

El uso de `-> TypeIs[NarrowedType]` le indica al verificador de tipo estático que para una función determinada:

1. El valor de retorno es un booleano.
2. Si el valor de retorno es `True`, el tipo de su argumento es la intersección del tipo original del argumento y `NarrowedType`.
3. Si el valor de retorno es `False`, el tipo de su argumento se limita para excluir `NarrowedType`.

Por ejemplo:

```
from typing import assert_type, final, TypeIs

class Parent: pass
class Child(Parent): pass
@final
class Unrelated: pass

def is_parent(val: object) -> TypeIs[Parent]:
    return isinstance(val, Parent)

def run(arg: Child | Unrelated):
    if is_parent(arg):
        # El tipo de ``arg`` es reducido a la inter
        # de ``Parent`` y ``Child``, lo cual es equ
        # ``Child``.
        assert_type(arg, Child)
    else:
        # El tipo de  ``arg`` es reducido para excl
        # para que solo quede ``Unrelated``.
        assert_type(arg, Unrelated)
```

El tipo dentro de `TypeIs` debe ser coherente con el tipo del argumento de la función; si no lo es, los comprobadores de tipos estáticos generarán un error. Una función `TypeIs` escrita incorrectamente puede provocar un comportamiento incorrecto en el sistema de tipos; es responsabilidad del usuario escribir dichas funciones de manera segura.

Si una función `TypeIs` es un método de clase o instancia, entonces el tipo en `TypeIs` se asigna al tipo del segundo parámetro (después de `cls` o `self`).

En resumen, la forma `def foo(arg: TypeA) -> TypeIs[TypeB]: ...`, significa que si `foo(arg)` devuelve `True`, entonces `arg` es una instancia de `TypeB`, y si devuelve `False`, no es una instancia de `TypeB`.

`TypeIs` también funciona con variables de tipo. Para obtener más información, consulte **PEP 742** [https://peps.python.org/pep-0742/] (Restringir tipos con `TypeIs`).

Added in version 3.13.

typing.TypeGuard

Construcción de tipificación especial para marcar funciones de predicado de tipo definido por el usuario.

Las funciones de predicado de tipo son funciones definidas por el usuario que indican si su argumento es una instancia de un tipo en particular. `TypeGuard` funciona de manera similar a **`TypeIs`**, pero tiene efectos ligeramente diferentes en el comportamiento de verificación de tipo (ver a continuación).

El uso de `-> TypeGuard` le dice al validador de tipo estático que para una función determinada:

1. El valor de retorno es un booleano.
2. Si el valor de retorno es `True`, el tipo de su argumento es el tipo dentro de `TypeGuard`.

`TypeGuard` también funciona con variables de tipo. Véase **PEP 647** [https://peps.python.org/pep-0647/] para más detalles.

Por ejemplo:

```
def is_str_list(val: list[object]) -> TypeGuard[lis
    '''Determines whether all objects in the list a
    return all(isinstance(x, str) for x in val)

def func1(val: list[object]):
    if is_str_list(val):
        # El tipo de ``val`` es reducido a ``list[s
        print(" ".join(val))
    else:
        # El tipo de ``val`` se mantiene como ``lis
        print("Not a list of strings!")
```

`TypeIs` y `TypeGuard` se diferencian en los siguientes aspectos:

- `TypeIs` requiere que el tipo restringido sea un subtipo del tipo de entrada, mientras que `TypeGuard` no lo requiere. La razón principal es permitir cosas como restringir `list[object]` a `list[str]`, aunque este último no sea un subtipo del primero, ya que `list` es invariante.
- Cuando una función `TypeGuard` devuelve `True`, los verificadores de tipo limitan el tipo de la variable exactamente al tipo `TypeGuard`. Cuando una función `TypeIs` devuelve `True`, los verificadores de tipo pueden inferir un tipo más preciso combinando el tipo conocido previamente de la variable con el tipo `TypeIs`. (Técnicamente, esto se conoce como un tipo de intersección).
- Cuando una función `TypeGuard` devuelve `False`, los verificadores de tipo no pueden limitar el tipo de la variable en absoluto. Cuando una función `TypeIs` devuelve `False`, los verificadores de tipo pueden limitar el tipo de la variable para excluir el tipo `TypeIs`.

Added in version 3.10.

typing.Unpack

Tipado para marcar conceptualmente un objeto como si hubiera sido desempaquetado.

Por ejemplo, usar el operador de descompresión `*` en un type variable tuple es equivalente a usar `Unpack` para marcar la tupla de variable de tipo como descomprimida:

```
Ts = TypeVarTuple('Ts')
tup: tuple[*Ts]
# Effectively does:
tup: tuple[Unpack[Ts]]
```

De hecho, `Unpack` se puede usar indistintamente con `*` en el contexto de los tipos **`typing.TypeVarTuple`** y **`builtins.tuple`**. Es posible que veas que `Unpack` se usa explícitamente en versiones anteriores de Python, donde `*` no se podía usar en ciertos lugares:

```
# En versiones anteriores de Python, TypeVarTuple y
# se encuentran en el paquete de retroadaptación `t
from typing_extensions import TypeVarTuple, Unpack

Ts = TypeVarTuple('Ts')
tup: tuple[*Ts]         # Error de sintaxis en Pyth
tup: tuple[Unpack[Ts]]  # Equivalente semánticament
```

`Unpack` también se puede usar junto con **`typing.TypedDict`** para tipear `**kwargs` en una firma de función:

```
from typing import TypedDict, Unpack

class Movie(TypedDict):
    name: str
    year: int

# Esta función espera dos argumentos de palabras cl
# y `year` de tipo `int`.
def foo(**kwargs: Unpack[Movie]): ...
```

Consulte **PEP 692** [https://peps.python.org/pep-0692/] para obtener más información sobre el uso de `Unpack` para tipear `**kwargs`.

Added in version 3.11.

Creación de tipos genéricos y alias de tipos

Las siguientes clases no se deben utilizar directamente como anotaciones. Su finalidad es servir de bloques de construcción para crear tipos genéricos y alias de tipos.

Estos objetos se pueden crear mediante una sintaxis especial (type parameter lists y la declaración **`type`**). Para compatibilidad con Python 3.11 y versiones anteriores, también se pueden crear sin la sintaxis dedicada, como se documenta a continuación.

class typing.Generic

Clase base abstracta para tipos genéricos.

Un tipo genérico normalmente se declara agregando una lista de parámetros de tipo después del nombre de la clase:

```
class Mapping[KT, VT]:
    def __getitem__(self, key: KT) -> VT:
        ...
        # Etc.
```

Esta clase hereda implícitamente de `Generic`. La semántica de tiempo de ejecución de esta sintaxis se analiza en la Referencia del lenguaje.

Entonces, esta clase se puede usar como sigue:

```
def lookup_name[X, Y](mapping: Mapping[X, Y], key:
    try:
        return mapping[key]
    except KeyError:
        return default
```

Aquí los corchetes después del nombre de la función indican una función genérica.

Para compatibilidad con versiones anteriores, las clases genéricas también se pueden declarar heredando explícitamente de `Generic`. En este caso, los parámetros de tipo se deben declarar por separado:

```
KT = TypeVar('KT')
VT = TypeVar('VT')

class Mapping(Generic[KT, VT]):
    def __getitem__(self, key: KT) -> VT:
        ...
        # Etc.
```

class typing.TypeVar(*name*, **constraints*, *bound=None*, *covariant=False*, *contravariant=False*, *infer_variance=False*, *default=typing.NoDefault*)

Variable de tipo.

La forma preferida de construir una variable de tipo es a través de la sintaxis dedicada para funciones genéricas, clases genéricas y alias de tipo genérico:

```
class Sequence[T]:  # T is a TypeVar
    ...
```

This syntax can also be used to create bounded and constrained type variables:

```
class StrSequence[S: str]:  # S is a TypeVar with a
    ...                     # we can say that S is


class StrOrBytesSequence[A: (str, bytes)]:  # A is
    ...
```

Sin embargo, si se desea, también se pueden construir manualmente variables de tipo reutilizables, de la siguiente manera:

```
T = TypeVar('T')  # Puede ser cualquier cosa
```

```
S = TypeVar('S', bound=str)  # Puede ser cualquier
A = TypeVar('A', str, bytes)  # Tiene que ser exact
```

Las variables de tipo existen principalmente para el beneficio de los validadores de tipos estáticos. Sirven como parámetros para tipos genéricos, así como para definiciones de alias de tipo y funciones genéricas. Consulte **`Generic`** para obtener más información sobre tipos genéricos. Las funciones genéricas funcionan de la siguiente manera:

```
def repeat[T](x: T, n: int) -> Sequence[T]:
    """Return a list containing n references to x."
    return [x]*n


def print_capitalized[S: str](x: S) -> S:
    """Print x capitalized, and return x."""
    print(x.capitalize())
    return x


def concatenate[A: (str, bytes)](x: A, y: A) -> A:
    """Add two strings or bytes objects together.""
    return x + y
```

Note that type variables can be *bounded*, *constrained*, or neither, but cannot be both bounded *and* constrained.

La varianza de las variables de tipo es inferida por los validadores de tipo cuando se crean a través de la sintáxis de parámetros de tipo o cuando se pasa `infer_variance=True`. Las variables de tipo creadas manualmente se pueden marcar explícitamente como covariantes o contravariantes al pasar `covariant=True` o `contravariant=True`. De manera predeterminada, las variables de tipo creadas manualmente son invariantes. Consulte **PEP 484** [https://peps.python.org/pep-0484/] y **PEP 695** [https://peps.python.org/pep-0695/] para obtener más detalles.

Bounded type variables and constrained type variables have different semantics in several important ways. Using a *bounded* type variable means that the `TypeVar` will be solved using the most specific type possible:

```
x = print_capitalized('a string')
reveal_type(x)  # el tipo revelado es str

class StringSubclass(str):
    pass

y = print_capitalized(StringSubclass('another strin
reveal_type(y)  # revealed type is StringSubclass

z = print_capitalized(45)  # error: int no es un su
```

The upper bound of a type variable can be a concrete type, abstract type (ABC or Protocol), or even a union of types:

```
# Puede ser cualquier cosa con un método __abs__
def print_abs[T: SupportsAbs](arg: T) -> None:
    print("Absolute value:", abs(arg))

U = TypeVar('U', bound=str|bytes)  # Puede ser cual
V = TypeVar('V', bound=SupportsAbs)  # Puede ser cu
```

Sin embargo, usar una variable de tipo *constrained* significa que la `TypeVar` sólo podrá ser determinada como exactamente una de las restricciones dadas:

```
a = concatenate('one', 'two')
reveal_type(a)  # tipo revelado es str

b = concatenate(StringSubclass('one'), StringSubcla
reveal_type(b)  # tipo revelado es str, a pesar que

c = concatenate('one', b'two')  # error: la variabl
```

En tiempo de ejecución, `isinstance(x, T)` lanzará **`TypeError`**.

name

El nombre de la variable de tipo.

covariant

Si la variable de tipo ha sido marcado explícitamente como covariante.

contravariant

Si la variable de tipo ha sido marcado explícitamente como covariante.

_infer_variance_

Si los validadores de tipo deben inferir la variación de la variable de tipo.

Added in version 3.12.

bound

The upper bound of the type variable, if any.

Distinto en la versión 3.12: Para las variables de tipo creadas a través de sintáxis de parámetros de tipo, el límite se evalúa solo cuando se accede al atributo, no cuando se crea la variable de tipo (consulte Evaluación perezosa).

evaluate_bound()

An evaluate function corresponding to the `__bound__` attribute. When called directly, this method supports only the `VALUE` format, which is equivalent to accessing the `__bound__` attribute directly, but the method object can be passed to `annotationlib.call_evaluate_function()` to evaluate the value in a different format.

Added in version 3.14.

constraints

Una tupla que contiene las restricciones de la variable de tipo, si las hay.

Distinto en la versión 3.12: Para las variables de tipo creadas a través de la sintáxis de parámetros de tipo, las restricciones se evalúan solo cuando se accede al atributo, no cuando se crea la variable de tipo (consulte Evaluación perezosa).

evaluate_constraints()

An evaluate function corresponding to the **`__constraints__`** attribute. When called directly, this method supports only the **`VALUE`** format, which is equivalent to accessing the **`__constraints__`** attribute directly, but the method object can be passed to **`annotationlib.call_evaluate_function()`** to evaluate the value in a different format.

Added in version 3.14.

default

El valor predeterminado de la variable de tipo, o **`typing.NoDefault`** si no tiene valor predeterminado.

Added in version 3.13.

evaluate_default()

An evaluate function corresponding to the **`__default__`** attribute. When called directly, this method supports only the **`VALUE`** format, which is equivalent to accessing the **`__default__`** attribute directly, but the method object can be passed to **`annotationlib.call_evaluate_function()`** to evaluate the value in a different format.

Added in version 3.14.

has_default()

Devuelve si la variable de tipo tiene o no un valor predeterminado. Esto es equivalente a verificar si `__default__` no es el singleton `typing.NoDefault`, excepto que no fuerza la evaluación del valor predeterminado lazily evaluated.

Added in version 3.13.

Distinto en la versión 3.12: Ahora es posible declarar variables de tipo utilizando la sintaxis de parámetros de tipo introducida por **PEP 695** [https://peps.python.org/pep-0695/]. Se agregó el parámetro `infer_variance`.

Distinto en la versión 3.13: Se agregó soporte para valores predeterminados.

class typing.TypeVarTuple(*name*, *, *default=typing.NoDefault*)

Tupla de variable de tipo. Forma especializada de type variable que habilita genéricos de *variadic*.

Las tuplas de variables de tipo se pueden declarar en listas de parámetros de tipo usando un solo asterisco (`*`) antes del nombre:

```
def move_first_element_to_last[T, *Ts](tup: tuple[T
    return (*tup[1:], tup[0])
```

O invocando explícitamente el constructor `TypeVarTuple`:

```
T = TypeVar("T")
Ts = TypeVarTuple("Ts")

def move_first_element_to_last(tup: tuple[T, *Ts])
    return (*tup[1:], tup[0])
```

Una variable de tipo normal permite parametrizar con un solo tipo. Una tupla de variables de tipo, en contraste, permite la parametrización con un número *arbitrario* de tipos, al actuar como un número *arbitrario* de variables de tipo envueltas en una tupla. Por ejemplo:

```
# T está ligado a int, Ts está ligado a ()
# El valor de retorno es (1,), que tiene tipo tuple
move_first_element_to_last(tup=(1,))

# T está ligado a int, Ts está ligado a (str,)
# El valor de retorno es ('spam', 1), que tiene tip
move_first_element_to_last(tup=(1, 'spam'))

# T está ligado a int, Ts está ligado a (str, float
# El valor de retorno es ('spam', 3.0, 1), que tien
move_first_element_to_last(tup=(1, 'spam', 3.0))

# Esto falla al verificar el tipo (y falla en tiemp
# porque tuple[()] no es compatible con tuple[T, *T
# (se requiere al menos un elemento)
move_first_element_to_last(tup=())
```

Nótese el uso del operador de desempaquetado `*` en `tuple[T, *Ts]`. Conceptualmente, puede pensarse en `Ts` como una tupla de variables de tipo `(T1, T2, ...)`. `tuple[T, *Ts]` se convertiría en `tuple[T, *(T1, T2, ...)]`, lo que es equivalente a `tuple[T, T1, T2, ...]`. (Nótese que en versiones más antiguas de Python, ésto puede verse escrito usando en cambio **`Unpack`**, en la forma `Unpack[Ts]`.)

Las tuplas de variables de tipo *siempre* deben descomprimirse. Esto ayuda a distinguir las tuplas de variables de tipo, de las variables de tipo normales:

```
x: Ts          # No válido
x: tuple[Ts]   # No válido
x: tuple[*Ts]  # La forma correcta de hacerlo
```

Las tuplas de variables de tipo pueden ser utilizadas en los mismos contextos que las variables de tipo normales. Por ejemplo en definiciones de clases, argumentos y tipos de retorno:

```
class Array[*Shape]:
    def __getitem__(self, key: tuple[*Shape]) -> fl
    def __abs__(self) -> "Array[*Shape]": ...
    def get_shape(self) -> tuple[*Shape]: ...
```

Las tuplas de variables de tipo se pueden combinar sin problemas con variables de tipo normales:

```
class Array[DType, *Shape]:  # Esto está bien
    pass

class Array2[*Shape, DType]:  # Esto también está b
    pass

class Height: ...
class Width: ...

float_array_1d: Array[float, Height] = Array()
int_array_2d: Array[int, Height, Width] = Array()
```

Sin embargo, nótese que en una determinada lista de argumentos de tipo o de parámetros de tipo puede haber como máximo una tupla de variables de tipo:

```
x: tuple[*Ts, *Ts]            # Not valid
class Array[*Shape, *Shape]:  # Not valid
    pass
```

Finalmente, una tupla de variables de tipo desempaquetada puede ser utilizada como la anotación de tipo de `*args`:

```
def call_soon[*Ts](
    callback: Callable[[*Ts], None],
    *args: *Ts
) -> None:
    ...
    callback(*args)
```

En contraste con las anotaciones no-desempaquetadas de `*args`, por ej. `*args: int`, que especificaría que *todos* los argumentos son `int` - `*args: *Ts` permite referenciar los tipos de los argumentos *individuales* en `*args`. Aquí, ésto permite asegurarse de que los tipos de los `*args` que son pasados a `call_soon` calcen con los tipos de los argumentos (posicionales) de `callback`.

Véase **PEP 646** [https://peps.python.org/pep-0646/] para obtener más detalles sobre las tuplas de variables de tipo.

name

El nombre de la tupla de variables de tipo.

default

El valor predeterminado de la variable de tipo tupla, o **`typing.NoDefault`** si no tiene valor predeterminado.

Added in version 3.13.

evaluate_default()

An evaluate function corresponding to the **`__default__`** attribute. When called directly, this method supports only the **`VALUE`** format, which is equivalent to accessing the **`__default__`** attribute directly, but the method object can be passed to **`annotationlib.call_evaluate_function()`** to evaluate the value in a different format.

Added in version 3.14.

has_default()

Devuelve si la variable de tipo tupla tiene o no un valor predeterminado. Esto es equivalente a verificar si **`__default__`** no es el singleton **`typing.NoDefault`**, excepto que no fuerza la evaluación del valor predeterminado lazily evaluated.

Added in version 3.13.

Added in version 3.11.

Distinto en la versión 3.12: Ahora es posible declarar tuplas de variables de tipo utilizando la sintaxis de parámetros de tipo introducida por **PEP 695** [https://peps.python.org/pep-0695/].

Distinto en la versión 3.13: Se agregó soporte para valores predeterminados.

class typing.ParamSpec(*name*, **, *bound=None*, *covariant=False*, *contravariant=False*, *default=typing.NoDefault*)

Variable de especificación de parámetros. Versión especializada de type variables.

En las listas de parámetros de tipo, las especificaciones de parámetros se pueden declarar con dos asteriscos (`**`):

```
type IntFunc[**P] = Callable[P, int]
```

Para compatibilidad con Python 3.11 y versiones anteriores, los objetos `ParamSpec` también se pueden crear de la siguiente manera:

```
P = ParamSpec('P')
```

Las variables de especificación de parámetros existen principalmente para el beneficio de los validadores de tipo estático. Se utilizan para reenviar los tipos de parámetros de un invocable a otro invocable, un patrón que se encuentra comúnmente en funciones y decoradores de orden superior. Solo son válidos cuando se utilizan en `Concatenate`, o como primer argumento de `Callable`, o como parámetros para genéricos definidos por el usuario. Consulte **`Generic`** para obtener más información sobre tipos genéricos.

Por ejemplo, para agregar un registro básico a una función, se puede crear un decorador `add_logging` para registrar llamadas a funciones. La variable de especificación de parámetros le dice al validador de tipo que el invocable pasado al decorador y el nuevo invocable retornado por él tienen parámetros de tipo interdependientes:

```
from collections.abc import Callable
import logging

def add_logging[T, **P](f: Callable[P, T]) -> Calla
    '''A type-safe decorator to add logging to a fu
    def inner(*args: P.args, **kwargs: P.kwargs) ->
        logging.info(f'{f.__name__} was called')
        return f(*args, **kwargs)
    return inner

@add_logging
def add_two(x: float, y: float) -> float:
    '''Add two numbers together.'''
    return x + y
```

Without `ParamSpec`, the simplest way to annotate this previously was to use a **`TypeVar`** with upper bound `Callable[..., Any]`. However this causes two problems:

1. El validador de tipo no puede verificar la función `inner` porque `*args` y `**kwargs` deben escribirse **`Any`**.
2. Es posible que se requiera **`cast()`** en el cuerpo del decorador `add_logging` al retornar la función `inner`, o se debe indicar al validador de tipo estático que ignore el `return inner`.

args

kwargs

Dado que `ParamSpec` captura tanto parámetros posicionales como de palabras clave, `P.args` y `P.kwargs` se pueden utilizar para dividir un `ParamSpec` en sus componentes. `P.args` representa la tupla de parámetros posicionales en una llamada determinada y solo debe usarse para anotar `*args`. `P.kwargs` representa la asignación de parámetros de palabras clave a sus valores en una llamada determinada y solo debe usarse para anotar `**kwargs`. Ambos atributos requieren que el parámetro anotado esté dentro del alcance. En tiempo de ejecución, `P.args` y `P.kwargs` son instancias respectivamente de **`ParamSpecArgs`** y **`ParamSpecKwargs`**.

__name__

El nombre de la especificación del parámetro.

__default__

El valor predeterminado de la especificación del parámetro, o **`typing.NoDefault`** si no tiene valor predeterminado.

Added in version 3.13.

evaluate_default()

An evaluate function corresponding to the **`__default__`** attribute. When called directly, this method supports only the **`VALUE`** format, which is equivalent to accessing the **`__default__`** attribute directly, but the method object can be passed to **`annotationlib.call_evaluate_function()`** to evaluate the value in a different format.

Added in version 3.14.

has_default()

Devuelve si la especificación del parámetro tiene o no un valor predeterminado. Esto es equivalente a verificar si **`__default__`** no es el singleton **`typing.NoDefault`**, excepto que no fuerza la evaluación del valor predeterminado lazily evaluated.

Added in version 3.13.

Las variables de especificación de parámetros creadas con `covariant=True` o `contravariant=True` se pueden utilizar para declarar tipos genéricos covariantes o contravariantes. También se acepta el argumento `bound`, similar a **`TypeVar`**. Sin embargo, la semántica real de estas palabras clave aún no se ha decidido.

Added in version 3.10.

Distinto en la versión 3.12: Las especificaciones de parámetros ahora se pueden declarar utilizando la sintaxis de parámetros de tipo introducida por **PEP 695** [https://peps.python.org/pep-0695/].

Distinto en la versión 3.13: Se agregó soporte para valores predeterminados.

Nota

Solo las variables de especificación de parámetros definidas en el ámbito global pueden ser serializadas.

Ver también

- **PEP 612** [https://peps.python.org/pep-0612/] - Variables de especificación de parámetros (el PEP que introdujo `ParamSpec` y `Concatenate`)
- **`Concatenate`**
- Anotaciones en objetos invocables

typing.ParamSpecArgs
typing.ParamSpecKwargs

Argumentos y atributos de argumentos de palabras clave de un **`ParamSpec`**. El atributo `P.args` de un `ParamSpec` es una instancia de `ParamSpecArgs` y `P.kwargs` es una instancia de `ParamSpecKwargs`. Están pensados para la introspección en tiempo de ejecución y no tienen un significado especial para los validadores de tipo estático.

Llamar a **`get_origin()`** en cualquiera de estos objetos retornará el `ParamSpec` original:

```
>>> from typing import ParamSpec, get_origin
>>> P = ParamSpec("P")
>>> get_origin(P.args) is P
True
>>> get_origin(P.kwargs) is P
True
```

Added in version 3.10.

class typing.TypeAliasType(*name*, *value*, *, *type_params=()*)

El tipo de alias de tipo creado a través de la declaración **type**.

Por ejemplo:

```
>>> type Alias = int
>>> type(Alias)
<class 'typing.TypeAliasType'>
```

Added in version 3.12.

__name__

El nombre del alias de tipo:

```
>>> type Alias = int
>>> Alias.__name__
'Alias'
```

__module__

The name of the module in which the type alias was defined:

```
>>> type Alias = int
>>> Alias.__module__
'__main__'
```

__type_params__

Los parámetros de tipo del alias de tipo, o una tupla vacía si el alias no es genérico:

```
>>> type ListOrSet[T] = list[T] | set[T]
>>> ListOrSet.__type_params__
(T,)
>>> type NotGeneric = int
>>> NotGeneric.__type_params__
()
```

value

El valor del alias de tipo. Se evalúa de forma diferida, por lo que los nombres utilizados en la definición del alias no se resuelven hasta que se accede al atributo `__value__`:

```
>>> type Mutually = Recursive
>>> type Recursive = Mutually
>>> Mutually
Mutually
>>> Recursive
Recursive
>>> Mutually.__value__
Recursive
>>> Recursive.__value__
Mutually
```

evaluate_value()

An evaluate function corresponding to the **`__value__`** attribute. When called directly, this method supports only the **`VALUE`** format, which is equivalent to accessing the **`__value__`** attribute directly, but the method object can be passed to **`annotationlib.call_evaluate_function()`** to evaluate the value in a different format:

```
>>> type Alias = undefined
>>> Alias.__value__
Traceback (most recent call last):
...
NameError: name 'undefined' is not defined
```

```
>>> from annotationlib import Format, call_eval
>>> Alias.evaluate_value(Format.VALUE)
Traceback (most recent call last):
...
NameError: name 'undefined' is not defined
>>> call_evaluate_function(Alias.evaluate_value
ForwardRef('undefined')
```

Added in version 3.14.

Unpacking

Type aliases support star unpacking using the `*Alias` syntax. This is equivalent to using `Unpack[Alias]` directly:

```
>>> type Alias = tuple[int, str]
>>> type Unpacked = tuple[bool, *Alias]
>>> Unpacked.__value__
tuple[bool, typing.Unpack[Alias]]
```

Added in version 3.14.

Otras directivas especiales

Estas funciones y clases no se deben utilizar directamente como anotaciones. Su finalidad es servir de bloques de construcción para crear y declarar tipos.

class typing.NamedTuple

Versión para anotación de tipos de **`collections.namedtuple()`**.

Uso:

```
class Employee(NamedTuple):
    name: str
    id: int
```

Esto es equivalente a:

```
Employee = collections.namedtuple('Employee', ['nam
```

Para proporcionar a un campo un valor por defecto se puede asignar en el cuerpo de la clase:

```
class Employee(NamedTuple):
    name: str
    id: int = 3

employee = Employee('Guido')
assert employee.id == 3
```

Los campos con un valor por defecto deben ir después de los campos sin valor por defecto.

La clase resultante tiene un atributo extra `__annotations__` que proporciona un diccionario que mapea el nombre de los campos con sus respectivos tipos. (Los nombres de los campos están en el atributo `_fields` y sus valores por defecto en el atributo `_field_defaults`, ambos parte de la API **`namedtuple()`**.)

Las subclases de `NamedTuple` también pueden tener *docstrings* y métodos:

```
class Employee(NamedTuple):
    """Represents an employee."""
    name: str
    id: int = 3

    def __repr__(self) -> str:
        return f'<Employee {self.name}, id={self.id
```

Las subclases de `NamedTuple` pueden ser genéricas:

```
class Group[T](NamedTuple):
    key: T
    group: list[T]
```

Uso retrocompatible:

```
# Para crear un NamedTuple genérico en Python 3.11
T = TypeVar("T")

class Group(NamedTuple, Generic[T]):
    key: T
    group: list[T]

# También se admite una sintaxis funcional
Employee = NamedTuple('Employee', [('name', str), (
```

Distinto en la versión 3.6: Soporte añadido para la sintaxis de anotación de variables propuesto en **PEP 526** [https://peps.python.org/pep-0526/].

Distinto en la versión 3.6.1: Soporte añadido para valores por defecto, métodos y *docstrings*.

Distinto en la versión 3.8: Los atributos `_field_types` y `__annotations__` son simples diccionarios en vez de instancias de `OrderedDict`.

Distinto en la versión 3.9: Se remueve el atributo `_field_types` en favor del atributo más estándar `__annotations__` que tiene la misma información.

Distinto en la versión 3.9: `NamedTuple` is now a function rather than a class. It can still be used as a class base, as described above.

Distinto en la versión 3.11: Se agrega soporte para *namedtuples* genéricas.

Distinto en la versión 3.14: Using **`super()`** (and the `__class__` closure variable) in methods of `NamedTuple` subclasses is unsupported and causes a **`TypeError`**.

Deprecated since version 3.13, will be removed in version 3.15: La sintaxis de argumentos de palabras clave no documentada para crear clases NamedTuple (`NT = NamedTuple("NT", x=int)`) está obsoleta y no se permitirá en la versión 3.15. En su lugar, utilice la sintaxis basada en clases o la sintaxis funcional.

Deprecated since version 3.13, will be removed in version 3.15: Al utilizar la sintaxis funcional para crear una clase NamedTuple, no se permite pasar un valor al parámetro "campos" (`NT = NamedTuple("NT")`). También se permite pasar `None` al parámetro "campos" (`NT = NamedTuple("NT", None)`). Ambos métodos no estarán permitidos en Python 3.15. Para crear una clase NamedTuple con 0 campos, utilice `class NT(NamedTuple): pass` o `NT = NamedTuple("NT", [])`.

class typing.NewType(*name*, *tp*)

Clase auxiliar para crear tipos distintos con bajo consumo de recursos.

Un validador de tipos considera que un `NewType` es un tipo distinto. Sin embargo, en tiempo de ejecución, llamar a un `NewType` devuelve su argumento sin cambios.

Uso:

```
UserId = NewType('UserId', int)  # Declara el NewTy
first_user = UserId(1)  # "UserId" retorna el argum
```

__module__

The name of the module in which the new type is defined.

__name__

El nombre del nuevo tipo.

__supertype__

El tipo en el que se basa el nuevo tipo.

Added in version 3.5.2.

Distinto en la versión 3.10: `NewType` es ahora una clase en lugar de una función.

class typing.Protocol(*Generic*)

Clase base para clases de protocolo.

Las clases de protocolo se definen así:

```
class Proto(Protocol):
    def meth(self) -> int:
        ...
```

Tales clases son usadas principalmente con validadores estáticos de tipos que detectan subtipado estructural (*duck-typing* estático), por ejemplo:

```
class C:
    def meth(self) -> int:
        return 0

def func(x: Proto) -> int:
    return x.meth()

func(C())  # Pasa la verificación de tipos estática
```

See **PEP 544** [https://peps.python.org/pep-0544/] for more details. Protocol classes decorated with `runtime_checkable()` (described later) act as simple-minded runtime protocols that check only the presence of given attributes, ignoring their type signatures. Protocol classes without this decorator cannot be used as the second argument to `isinstance()` or `issubclass()`.

Las clases protocolo pueden ser genéricas, por ejemplo:

```
class GenProto[T](Protocol):
    def meth(self) -> T:
        ...
```

En el código que necesita ser compatible con Python 3.11 o anterior, los protocolos genéricos se pueden escribir de la siguiente manera:

```
T = TypeVar("T")

class GenProto(Protocol[T]):
    def meth(self) -> T:
        ...
```

Added in version 3.8.

@typing.runtime_checkable

Marca una clase protocolo como aplicable en tiempo de ejecución (lo convierte en un *runtime protocol*).

Such a protocol can be used with **`isinstance()`** and **`issubclass()`**. This allows a simple-minded structural check, very similar to «one trick ponies» in **`collections.abc`** such as **`Iterable`**. For example:

```
@runtime_checkable
class Closable(Protocol):
    def close(self): ...

assert isinstance(open('/some/file'), Closable)

@runtime_checkable
class Named(Protocol):
    name: str

import threading
assert isinstance(threading.Thread(name='Bob'), Nam
```

This decorator raises **`TypeError`** when applied to a non-protocol class.

Nota

`runtime_checkable()` comprobará únicamente la presencia de los métodos o atributos requeridos, no sus firmas de tipo o tipos. Por ejemplo, **`ssl.SSLObject`** es una clase, por lo tanto, pasa una comprobación **`issubclass()`** contra Callable. Sin embargo, el método `ssl.SSLObject.__init__` existe únicamente para generar un **`TypeError`** con un mensaje más informativo, por lo que es imposible llamar (instanciar) **`ssl.SSLObject`**.

Nota

Una verificación **`isinstance()`** contra un protocolo comprobable en tiempo de ejecución puede ser sorprendentemente lenta en comparación con una verificación `isinstance()` contra una clase que no es de protocolo. Considere utilizar expresiones alternativas como llamadas **`hasattr()`** para comprobaciones estructurales en código sensible al rendimiento.

Added in version 3.8.

Distinto en la versión 3.12: La implementación interna de las comprobaciones de **`isinstance()`** con protocolos que se pueden comprobar en tiempo de ejecución ahora utiliza **`inspect.getattr_static()`** para buscar atributos (anteriormente, se utilizaba **`hasattr()`**). Como resultado, algunos objetos que solían considerarse instancias de un protocolo que se podía comprobar en tiempo de ejecución ya no se consideran instancias de ese protocolo en Python 3.12+, y viceversa. Es poco probable que la mayoría de los usuarios se vean afectados por este cambio.

Distinto en la versión 3.12: The members of a runtime-checkable protocol are now considered «frozen» at runtime as soon as the class has been created. Monkey-patching attributes onto a runtime-checkable protocol will still work, but will have no impact on **`isinstance()`** checks comparing objects to the protocol. See What’s new in Python 3.12 for more details.

class typing.TypedDict(*dict*)

Special construct to add type hints to a dictionary. At runtime «**`TypedDict`** instances» are simply **`dicts`**.

`TypedDict` crea un tipo de diccionario que espera que todas sus instancias tenga un cierto conjunto de claves, donde cada clave está asociada con un valor de un tipo determinado. Esta exigencia no se comprueba en tiempo de ejecución y solo es aplicada por validadores de tipo. Uso:

```
class Point2D(TypedDict):
    x: int
    y: int
    label: str

a: Point2D = {'x': 1, 'y': 2, 'label': 'good'}  # O
b: Point2D = {'z': 3, 'label': 'bad'}           # F

assert Point2D(x=1, y=2, label='first') == dict(x=1
```

Una forma alternativa de crear un `TypedDict` es mediante la sintaxis de llamada de función. El segundo argumento debe ser un **`dict`** literal:

```
Punto2D = TypedDict('Punto2D', {'x': int, 'y': int,
```

This functional syntax allows defining keys which are not valid identifiers, for example because they are keywords or contain hyphens, or when key names must not be mangled like regular private names:

```
# raises SyntaxError
class Point2D(TypedDict):
    in: int  # 'in' is a keyword
    x-y: int  # name with hyphens

class Definition(TypedDict):
    __schema: str  # mangled to `_Definition__schem

# OK, functional syntax
Point2D = TypedDict('Point2D', {'in': int, 'x-y': i
Definition = TypedDict('Definition', {'__schema': s
```

De forma predeterminada, todas las llaves deben estar presentes en un `TypedDict`. Es posible marcar llaves individuales como *no requeridas* utilizando **`NotRequired`**:

```
class Point2D(TypedDict):
    x: int
    y: int
    label: NotRequired[str]

# Sintaxis alternativa
Point2D = TypedDict('Point2D', {'x': int, 'y': int,
```

Esto significa que en un `TypedDict` que sea una instancia de `Point2D`, será posible omitir la llave `label`.

Además, es posible marcar todas las llaves como no-requeridas por defecto, al especificar un valor de `False` en el argumento *total*:

```
class Point2D(TypedDict, total=False):
    x: int
    y: int

# Sintaxis alternativa
Point2D = TypedDict('Point2D', {'x': int, 'y': int}
```

Esto significa que un `TypedDict` `Point2D` puede tener cualquiera de las llaves omitidas. Solo se espera que un validador de tipo admita un `False` literal o `True` como valor del argumento `total`. `True` es el predeterminado y hace que todos los elementos definidos en el cuerpo de la clase sean obligatorios.

Las llaves individuales de un `TypedDict` `total=False` pueden ser marcadas como requeridas utilizando **`Required`**:

```
class Point2D(TypedDict, total=False):
    x: Required[int]
    y: Required[int]
    label: str
```

```
# Sintaxis alternativa
Point2D = TypedDict('Point2D', {
    'x': Required[int],
    'y': Required[int],
    'label': str
}, total=False)
```

Es posible que un tipo `TypedDict` herede de uno o más tipos `TypedDict` usando la sintaxis de clase. Uso:

```
class Point3D(Point2D):
    z: int
```

`Point3D` tiene tres elementos: `x`, `y` y `z`. Lo que es equivalente a la siguiente definición:

```
class Point3D(TypedDict):
    x: int
    y: int
    z: int
```

Un `TypedDict` no puede heredar de una clase que no sea una subclase de `TypedDict`, exceptuando **`Generic`**. Por ejemplo:

```
class X(TypedDict):
    x: int

class Y(TypedDict):
    y: int

class Z(object): pass  # Una clase  no-TypedDict

class XY(X, Y): pass  # OK

class XZ(X, Z): pass  # lanza TypeError
```

Un `TypedDict` puede ser genérico:

```
class Group[T](TypedDict):
    key: T
    group: list[T]
```

Para crear un `TypedDict` genérico que sea compatible con Python 3.11 o anterior, herede de **`Generic`** explícitamente:

```
T = TypeVar("T")

class Group(TypedDict, Generic[T]):
    key: T
    group: list[T]
```

A `TypedDict` can be introspected via annotations dicts (see Prácticas recomendadas para las anotaciones for more information on annotations best practices), **`__total__`**, **`__required_keys__`**, and **`__optional_keys__`**.

__total__

`Point2D.__total__` proporciona el valor del argumento `total`. Ejemplo:

```
>>> from typing import TypedDict
>>> class Point2D(TypedDict): pass
>>> Point2D.__total__
True
>>> class Point2D(TypedDict, total=False): pas
>>> Point2D.__total__
False
>>> class Point3D(Point2D): pass
>>> Point3D.__total__
True
```

Este atributo refleja el valor del argumento `total` de la clase `TypedDict` actual, no si la clase es semánticamente total. Por ejemplo, una `TypedDict` con `__total__` establecida en `True` puede tener claves marcadas con **`NotRequired`**, o puede heredar de otra `TypedDict` con `total=False`. Por lo tanto, generalmente es mejor usar **`__required_keys__`** y **`__optional_keys__`** para la introspección.

__required_keys__

Added in version 3.9.

__optional_keys__

`Point2D.__required_keys__` y `Point2D.__optional_keys__` retornan objetos de la clase **`frozenset`**, que contienen las llaves requeridas y no requeridas, respectivamente.

Las llaves marcadas con **`Required`** siempre aparecerán en `__required_keys__` y las llaves marcadas con **`NotRequired`** siempre aparecerán en `__optional_keys__`.

Para compatibilidad con versiones anteriores de Python 3.10, también es posible usar la herencia para declarar claves obligatorias y no obligatorias en el mismo `TypedDict`. Esto se hace declarando un `TypedDict` con un valor para el argumento `total` y luego heredando de él en otro `TypedDict` con un valor diferente para `total`:

```
>>> class Point2D(TypedDict, total=False):
...     x: int
...     y: int
...
>>> class Point3D(Point2D):
...     z: int
...
>>> Point3D.__required_keys__ == frozenset({'z'
True
>>> Point3D.__optional_keys__ == frozenset({'x'
True
```

Added in version 3.9.

Nota

Si se utiliza `from __future__ import annotations` o si las anotaciones se proporcionan como cadenas, las anotaciones no se evalúan cuando se define `TypedDict`. Por lo tanto, la introspección en tiempo de ejecución de la que dependen `__required_keys__` y `__optional_keys__` puede no funcionar correctamente y los valores de los atributos pueden ser incorrectos.

La compatibilidad con **`ReadOnly`** se refleja en los siguientes atributos:

__readonly_keys__

Un **`frozenset`** que contiene los nombres de todas las claves de solo lectura. Las claves son de solo lectura si llevan el calificador **`ReadOnly`**.

Added in version 3.13.

__mutable_keys__

Un **`frozenset`** que contiene los nombres de todas las claves mutables. Las claves son mutables si no llevan el calificador **`ReadOnly`**.

Added in version 3.13.

See the TypedDict [https://typing.python.org/en/latest/spec/typeddict.html#typeddict] section in the typing documentation for more examples and detailed rules.

Added in version 3.8.

Distinto en la versión 3.9: `TypedDict` is now a function rather than a class. It can still be used as a class base, as described above.

Distinto en la versión 3.11: Se agrega soporte para marcar llaves individuales como **`Required`** o **`NotRequired`**. Véase **PEP 655** [https://peps.python.org/pep-0655/].

Distinto en la versión 3.11: Se agrega soporte para

`TypedDict` genéricos.

Distinto en la versión 3.13: Se eliminó la compatibilidad con el método de argumento de palabra clave para crear `TypedDict`s.

Distinto en la versión 3.13: Se agregó soporte para el calificador **`ReadOnly`**.

Deprecated since version 3.13, will be removed in version 3.15: Al utilizar la sintaxis funcional para crear una clase TypedDict, no se permite pasar un valor al parámetro "campos" (`TD = TypedDict("TD")`). También se permite pasar `None` al parámetro "campos" (`TD = TypedDict("TD", None)`). Ambos métodos no estarán permitidos en Python 3.15. Para crear una clase TypedDict con 0 campos, utilice `class TD(TypedDict): pass` o `TD = TypedDict("TD", {})`.

Protocolos

The following protocols are provided by the **`typing`** module. All are decorated with **`@runtime_checkable`**.

class typing.SupportsAbs

Una ABC con un método abstracto `__abs__` que es covariante en su tipo retornado.

class typing.SupportsBytes

Una ABC con un método abstracto `__bytes__`.

class typing.SupportsComplex

Una ABC con un método abstracto `__complex__`.

class typing.SupportsFloat

Una ABC con un método abstracto `__float__`.

class typing.SupportsIndex

Una ABC con un método abstracto `__index__`.

Added in version 3.8.

class typing.SupportsInt

Una ABC con un método abstracto `__int__`.

class typing.SupportsRound

Una ABC con un método abstracto `__round__` que es covariantes en su tipo retornado.

ABCs and Protocols for working with I/O

class typing.IO[*AnyStr*]
class typing.TextIO
class typing.BinaryIO

Generic class `IO[AnyStr]` and its subclasses `TextIO(IO[str])` and `BinaryIO(IO[bytes])` represent the types of I/O streams such as returned by **`open()`**. Please note that these classes are not protocols, and their interface is fairly broad.

The protocols **`io.Reader`** and **`io.Writer`** offer a simpler alternative for argument types, when only the `read()` or `write()` methods are accessed, respectively:

```
def read_and_write(reader: Reader[str], writer: Writer[b
    data = reader.read()
    writer.write(data.encode())
```

Also consider using **`collections.abc.Iterable`** for iterating over the lines of an input stream:

```
def read_config(stream: Iterable[str]):
    for line in stream:
        ...
```

Funciones y decoradores

typing.cast(*typ*, *val*)

Convertir un valor a un tipo.

Esto retorna el valor sin modificar. Para el validador de tipos esto indica que el valor de retorno tiene el tipo señalado pero, de manera intencionada, no se comprobará en tiempo de ejecución (para maximizar la velocidad).

typing.assert_type(*val*, *typ*, */*)

Solicitar a un validador de tipos que confirme que *val* tiene *typ* por tipo inferido.

En tiempo de ejecución esto no hace nada: devuelve el primer argumento sin cambios, sin verificaciones ni efectos secundarios, sin importar el tipo real del argumento.

Cuando un validador de tipo estático encuentra una llamada a `assert_type()`, emite un error si el valor no es del tipo especificado:

```
def greet(name: str) -> None:
    assert_type(name, str)  # OK, tipo inferido de
    assert_type(name, int)  # error del verificador
```

Esta función es útil para asegurarse de que la comprensión que el validador de tipos tiene sobre un *script* está alineada con las intenciones de le desarrolladores:

```
def complex_function(arg: object):
    # Haz alguna lógica compleja de reducción de ti
    # despues de la cual esperamos que el tipo infe
    ...
    # Probar si el verificador de tipos entiende co
    assert_type(arg, int)
```

Added in version 3.11.

typing.assert_never(*arg*, */*)

Solicitar a un validador estático de tipos confirmar que una línea de código no es alcanzable.

Por ejemplo:

```
def int_or_str(arg: int | str) -> None:
    match arg:
        case int():
            print("It's an int")
        case str():
            print("It's a str")
        case _ as unreachable:
            assert_never(unreachable)
```

Aquí, las anotaciones permiten al validador de tipos inferir que el último caso nunca puede ejecutarse, porque `arg` es un **`int`** o un **`str`**, y ambas opciones están cubiertas por los casos anteriores.

Si un validador de tipos encuentra que una llamada a `assert_never()` es alcanzable, emitirá un error. Por ejemplo, si la anotación de tipo para `arg` fuese en cambio `int | str | float`, el validador de tipos emitiría un error que indicaría que `unreachable` es de tipo **`float`**. Para que una llamada a `assert_never` pase la verificación de tipos, el tipo inferido del argumento pasado debe ser el tipo inferior, **`Never`**, y nada más.

En tiempo de ejecución, ésto lanza una excepción cuando es llamado.

Ver también

Unreachable Code and Exhaustiveness Checking [https://typing.python.org/en/latest/guides/unreachable.html] has more information about exhaustiveness checking with static typing.

Added in version 3.11.

typing.reveal_type(*obj*, */*)

Pídale a un verificador de tipos estáticos que revele el tipo inferido de una expresión.

Cuando un verificador de tipos estáticos encuentra una llamada a esta función, emite un diagnóstico con el tipo inferido del argumento. Por ejemplo:

```
x: int = 1
reveal_type(x)  # Tipo revelado es "builtins.int"
```

Ésto puede ser de utilidad cuando se desea *debuguear* cómo tu validador de tipos maneja una pieza particular de código.

En tiempo de ejecución, esta función imprime el tipo de tiempo de ejecución de su argumento en **`sys.stderr`** y devuelve el argumento sin cambios (lo que permite que la llamada se use dentro de una expresión):

```
x = reveal_type(1)  # imprime "Runtime type is int"
print(x)  # prints "1"
```

Tenga en cuenta que el tipo de tiempo de ejecución puede ser diferente (más o menos específico) del tipo inferido estáticamente por un verificador de tipos.

La mayoría de los verificadores de tipos admiten `reveal_type()` en cualquier lugar, incluso si el nombre no se importa desde `typing`. Sin embargo, importar el nombre desde `typing` permite que el código se ejecute sin errores de tiempo de ejecución y comunica la intención con mayor claridad.

Added in version 3.11.

@typing.dataclass_transform(**, eq_default=True, order_default=False, kw_only_default=False, frozen_default=False, field_specifiers=(), **kwargs*)

Decorador para marcar un objeto como si proporcionara un comportamiento similar a **`dataclass`**.

`dataclass_transform` may be used to decorate a class, metaclass, or a function that is itself a decorator. The presence of `@dataclass_transform()` tells a static type checker that the decorated object performs runtime «magic» that transforms a class in a similar way to `@dataclasses.dataclass`.

Ejemplo de uso con una función decoradora:

```
@dataclass_transform()
def create_model[T](cls: type[T]) -> type[T]:
    ...
    return cls

@create_model
class CustomerModel:
    id: int
    name: str
```

En una clase base:

```
@dataclass_transform()
class ModelBase: ...

class CustomerModel(ModelBase):
    id: int
    name: str
```

En una metaclase:

```
@dataclass_transform()
class ModelMeta(type): ...

class ModelBase(metaclass=ModelMeta): ...

class CustomerModel(ModelBase):
    id: int
    name: str
```

The `CustomerModel` classes defined above will be treated by type checkers similarly to classes created with `@dataclasses.dataclass`. For example, type checkers will assume these classes have `__init__` methods that accept `id` and `name`.

The decorated class, metaclass, or function may accept the following bool arguments which type checkers will assume have the same effect as they would have on the **`@dataclasses.dataclass`** decorator: `init`, `eq`, `order`, `unsafe_hash`, `frozen`, `match_args`, `kw_only`, and `slots`. It must be possible for the value of these arguments (`True` or `False`) to be statically evaluated.

Es posible utilizar los argumentos del decorador `dataclass_transform` para personalizar los comportamientos por defecto de la clase, metaclase o función decorada:

Parámetros:

- **eq_default** (*bool*) – Indica si se asume que el parámetro `eq` es `True` o `False` si el llamador lo omite. El valor predeterminado es `True`.
- **order_default** (*bool*) – Indica si se asume que el parámetro `order` es `True` o `False` si el llamador lo omite. El valor predeterminado es `False`.
- **kw_only_default** (*bool*) – Indica si se asume que el parámetro `kw_only` es `True` o `False` si el llamador lo omite. El valor predeterminado es `False`.
- **frozen_default** (*bool*) – Indica si se asume que el parámetro `frozen` es `True` o `False` si el llamador lo omite. El valor predeterminado es `False`. .. Agregado en la versión:: 3.12
- **field_specifiers** (*tuple[Callable[..., Any], ...]*) – Especifica una lista estática de clases o funciones admitidas que describen campos, parecido con **`dataclasses.field()`**. El valor predeterminado es `()`.
- ****kwargs** (*Any*) – Es posible pasar

arbitrariamente otros argumentos nombrados para permitir posibles extensiones futuras.

Los validadores de tipos reconocen los siguientes parámetros opcionales en los especificadores de campo:

Nombre del parámetro	**Descripción**
`init`	Indica si el campo debe incluirse en el método `__init__` sintetizado. Si no se especifica, el valor predeterminado de `init` es `True`.
`default`	Proporciona el valor predeterminado para el campo.
`default_factory`	Proporciona una retrollamada en tiempo de ejecución que devuelve el valor predeterminado del campo. Si no se especifican `default` ni `default_factory`, se supone que el campo no tiene un valor predeterminado y se le debe proporcionar un valor cuando se cree una instancia de la clase.
`factory`	Un alias para el parámetro `default_factory` en los especificadores de campo.
`kw_only`	Indica si el campo debe marcarse como solo para palabras clave. Si es `True`, el campo será solo para palabras clave. Si es `False`, no será solo para palabras clave. Si no se especifica, se utilizará el valor del parámetro `kw_only` en el objeto decorado con `dataclass_transform`, o si no se especifica, se utilizará el valor de `kw_only_default` en `dataclass_transform`.
`alias`	Proporciona un nombre alternativo para el campo. Este nombre alternativo se utiliza en el método sintetizado `__init__`.

Parámetros reconocidos para especificadores de campo

En tiempo de ejecución, este decorador registra sus argumentos en el atributo `__dataclass_transform__` del objeto decorado. No tiene otro efecto en tiempo de ejecución.

Véase **PEP 681** [https://peps.python.org/pep-0681/] para más detalle.

Added in version 3.11.

@typing.overload

Decorador para crear funciones y métodos sobrecargados.

El decorador `@overload` permite describir funciones y métodos que admiten múltiples combinaciones diferentes de tipos de argumentos. Una serie de definiciones decoradas con `@overload` debe ir seguida de exactamente una definición que no esté decorada con `@overload` (para la misma función o método).

Las definiciones decoradas con `@overload` son solo para beneficio del validador de tipos, ya que serán sobrescritas por la definición no decorada con `@overload`. Mientras tanto, la definición no decorada con `@overload` se usará en tiempo de ejecución, pero el validador de tipos debe ignorarla. En tiempo de ejecución, llamar a una función decorada con `@overload` directamente generará **`NotImplementedError`**.

Un ejemplo de sobrecarga que proporciona un tipo más preciso que el que se puede expresar mediante una unión o una variable de tipo:

```
@overload
def process(response: None) -> None:
    ...
@overload
def process(response: int) -> tuple[int, str]:
    ...
@overload
def process(response: bytes) -> str:
    ...
def process(response):
    ...  # actual implementation goes here
```

Véase **PEP 484** [https://peps.python.org/pep-0484/] para más detalle y comparación con otras semánticas de tipado.

Distinto en la versión 3.11: Ahora es posible introspectar en tiempo de ejecución las funciones sobrecargadas utilizando `get_overloads()`.

typing.get_overloads(*func*)

Return a sequence of `@overload`-decorated definitions for *func*.

func is the function object for the implementation of the overloaded function. For example, given the definition of `process` in the documentation for `@overload`, `get_overloads(process)` will return a sequence of three function objects for the three defined overloads. If called on a function with no overloads, `get_overloads()` returns an empty sequence.

`get_overloads()` puede ser utilizada para introspectar en tiempo de ejecución una función sobrecargada.

Added in version 3.11.

typing.clear_overloads()

Borra todas las sobrecargas registradas en el registro interno.

Esto se puede utilizar para recuperar la memoria utilizada por el registro.

Added in version 3.11.

@typing.final

Decorador para indicar métodos finales y clases finales.

Decorar un método con `@final` indica a un validador de tipos que el método no se puede anular en una subclase. Decorar una clase con `@final` indica que no se puede subclasificar.

Por ejemplo:

```
class Base:
    @final
    def done(self) -> None:
        ...
class Sub(Base):
    def done(self) -> None:  # Error informado por
        ...

@final
class Leaf:
    ...
class Other(Leaf):  # Error informado por el verifi
    ...
```

No hay comprobación en tiempo de ejecución para estas propiedades. Véase **PEP 591** [https://peps.python.org/pep-0591/] para más detalles.

Added in version 3.8.

Distinto en la versión 3.11: El decorador intentará ahora establecer un atributo `__final__` como `True` en el objeto decorado. Por lo tanto, se puede utilizar una comprobación como `if getattr(obj, “__final__”, False)` en tiempo de ejecución para determinar si un objeto `obj` se ha marcado como final. Si el objeto decorado no admite la configuración de atributos, el decorador devuelve el objeto sin cambios sin lanzar una excepción.

@typing.no_type_check

Un decorador para indicar que las anotaciones no deben ser comprobadas como indicadores de tipo.

Esto funciona como un decorator (decorador) de clase o función. Con una clase, se aplica recursivamente a todos los métodos y clases definidos en esa clase (pero no a los métodos definidos en sus superclases o subclases). Los validadores de tipos ignorarán todas las anotaciones en una función o clase con este decorador.

`@no_type_check` muta el objeto decorado en su lugar.

@typing.no_type_check_decorator

Decorador para dar a otro decorador el efecto **`no_type_check()`**.

Esto hace que el decorador decorado añada el efecto de **`no_type_check()`** a la función decorada.

Deprecated since version 3.13, will be removed in version 3.15: Ningún verificador de tipos agregó compatibilidad con `@no_type_check_decorator`. Por lo tanto, está obsoleto y se eliminará en Python 3.15.

@typing.override

Decorador para indicar que un método en una subclase está destinado a anular un método o atributo en una superclase.

Los validadores de tipos deberían emitir un error si un método decorado con `@override` no anula nada. Esto ayuda a evitar errores que pueden ocurrir cuando se modifica una clase base sin un cambio equivalente en una clase secundaria.

Por ejemplo:

```
class Base:
    def log_status(self) -> None:
        ...

class Sub(Base):
    @override
    def log_status(self) -> None:  # Okay: sobreesc
        ...

    @override
    def done(self) -> None:  # Error reportado por
        ...
```

No hay ninguna comprobación en tiempo de ejecución de esta propiedad.

El decorador intentará establecer un atributo `__override__` en `True` en el objeto decorado. Por lo tanto, una comprobación como `if getattr(obj, “__override__”, False)` se puede utilizar en tiempo de ejecución para determinar si un objeto `obj` ha sido marcado como anulado. Si el objeto decorado no admite la configuración de atributos, el decorador devuelve el objeto sin cambios sin generar una excepción.

Vea **PEP 681** [https://peps.python.org/pep-0681/] para más información.

Added in version 3.12.

@typing.type_check_only

Decorador para marcar una clase o función como no disponible en tiempo de ejecución.

Este decorador no está disponible en tiempo de ejecución. Existe principalmente para marcar clases que se definen en archivos *stub* para cuando una implementación retorna una instancia de una clase privada:

```
@type_check_only
class Response:  # privado o no disponible en tiemp
    code: int
    def get_header(self, name: str) -> str: ...

def fetch_response() -> Response: ...
```

Nótese que no se recomienda retornar instancias de clases privadas. Normalmente es preferible convertirlas en clases públicas.

Ayudas de introspección

typing.get_type_hints(*obj*, *globalns=None*, *localns=None*, *include_extras=False*, ***, *format=Format.VALUE*)

Return a dictionary containing type hints for a function, method, module, class object, or other callable object.

This is often the same as `annotationlib.get_annotations()`, but this function makes the following changes to the annotations dictionary:

- Las referencias futuras codificadas como literales de cadena u objetos `ForwardRef` se manejan evaluándolas en el espacio de nombres *globalns*, *localns* y (cuando corresponda) el espacio de nombre type parameter *obj*. Si no se proporciona *globalns* o *localns*, los diccionarios de espacios de nombres apropiados se infieren de *obj*.
- `None` se reemplaza por `types.NoneType`.
- If `@no_type_check` has been applied to *obj*, an empty dictionary is returned.
- If *obj* is a class `C`, the function returns a dictionary that merges annotations from `C`'s base classes with those on `C` directly. This is done by traversing `C.__mro__` and iteratively combining annotations of each base class. Annotations on classes appearing earlier in the method resolution order always take precedence over annotations on classes appearing later in the method resolution order.
- The function recursively replaces all occurrences of `Annotated[T, ...]`, `Required[T]`, `NotRequired[T]`, and `ReadOnly[T]` with `T`, unless *include_extras* is set to `True` (see `Annotated` for more information).

Prudencia

This function may execute arbitrary code contained in annotations. See Security implications of introspecting annotations for more information.

Nota

If `Format.VALUE` is used and any forward references in the annotations of *obj* are not resolvable, a `NameError` exception is raised. For example, this can happen with names imported under `if TYPE_CHECKING`. More generally, any kind of exception can be raised if an annotation contains invalid Python code.

Nota

Calling **`get_type_hints()`** on an instance is not supported. To retrieve annotations for an instance, call **`get_type_hints()`** on the instance's class instead (for example, `get_type_hints(type(obj))`).

Distinto en la versión 3.9: Se agregó el parámetro `include_extras` como parte de **PEP 593** [https://peps.python.org/pep-0593/]. Consulte la documentación en **`Annotated`** para obtener más información.

Distinto en la versión 3.11: Anteriormente, se agregaba `Optional[t]` en las anotaciones de funciones o métodos si se establecía un valor por defecto igual a `None`. Ahora la anotación es retornada sin cambios.

Distinto en la versión 3.14: Added the `format` parameter. See the documentation on **`annotationlib.get_annotations()`** for more information.

Distinto en la versión 3.14: Calling **`get_type_hints()`** on instances is no longer supported. Some instances were accepted in earlier versions as an undocumented implementation detail.

typing.get_origin(*tp*)

Obtiene la versión sin suscripción de un tipo: para un objeto de tipado de la forma `X[Y, Z, ...]` devuelve `X`.

Si `X` es un alias de módulo de tipado para una clase incorporada o **`collections`**, se normalizará a la clase original. Si `X` es una instancia de **`ParamSpecArgs`** o **`ParamSpecKwargs`**, devuelve la **`ParamSpec`** subyacente. Devuelve `None` para objetos no soportados.

Por ejemplo:

```
assert get_origin(str) is None
assert get_origin(Dict[str, int]) is dict
assert get_origin(Union[int, str]) is Union
assert get_origin(Annotated[str, "metadata"]) is An
P = ParamSpec('P')
assert get_origin(P.args) is P
assert get_origin(P.kwargs) is P
```

Added in version 3.8.

typing.get_args(*tp*)

Obtiene los argumentos de tipo con todas las sustituciones realizadas: para un objeto de tipo con la forma `X[Y, Z, ...]` devuelve `(Y, Z, ...)`.

Si `X` es una unión o **`Literal`** contenida en otro tipo genérico, el orden de `(Y, Z, ...)` puede ser diferente del orden de los argumentos originales `[Y, Z, ...]` debido al almacenamiento en caché de tipos. Devuelve `()` para objetos no soportados.

Por ejemplo:

```
assert get_args(int) == ()
assert get_args(Dict[int, str]) == (int, str)
assert get_args(Union[int, str]) == (int, str)
```

Added in version 3.8.

typing.get_protocol_members(*tp*)

Devuelve el conjunto de miembros definidos en un **`Protocol`**.

```
>>> from typing import Protocol, get_protocol_membe
>>> class P(Protocol):
...     def a(self) -> str: ...
...     b: int
```

```
>>> get_protocol_members(P) == frozenset({'a', 'b'}
True
```

Genera **`TypeError`** para argumentos que no sean protocolos.

Added in version 3.13.

typing.is_protocol(*tp*)

Compruebe si un tipo es **`TypedDict`**.

Por ejemplo:

```
class P(Protocol):
    def a(self) -> str: ...
    b: int

is_protocol(P)     # => True
is_protocol(int)   # => False
```

Added in version 3.13.

typing.is_typeddict(*tp*)

Compruebe si un tipo es **`TypedDict`**.

Por ejemplo:

```
class Film(TypedDict):
    title: str
    year: int

assert is_typeddict(Film)
assert not is_typeddict(list | str)

# TypedDict es una fábrica para crear diccionarios
# no un diccionario tipado en si mismo
assert not is_typeddict(TypedDict)
```

Added in version 3.10.

class typing.ForwardRef

Clase utilizada para la representación interna de tipado de cadenas de caracteres en referencias futuras.

For example, `List["SomeClass"]` is implicitly transformed into `List[ForwardRef("SomeClass")]`. **`ForwardRef`** should not be instantiated by a user, but may be used by introspection tools.

Nota

Los tipos genéricos de **PEP 585** [https://peps.python.org/pep-0585/], como `list["SomeClass"]`, no se transformarán implícitamente en `list[ForwardRef("SomeClass")]` y, por lo tanto, no se resolverán automáticamente en `list[SomeClass]`.

Added in version 3.7.4.

Distinto en la versión 3.14: This is now an alias for **`annotationlib.ForwardRef`**. Several undocumented behaviors of this class have been changed; for example, after a `ForwardRef` has been evaluated, the evaluated value is no longer cached.

typing.evaluate_forward_ref(*forward_ref*, ***, *owner=None*, *globals=None*, *locals=None*, *type_params=None*, *format=annotationlib.Format.VALUE*)

Evaluate an **`annotationlib.ForwardRef`** as a type hint.

This is similar to calling **`annotationlib.ForwardRef.evaluate()`**, but unlike that method, **`evaluate_forward_ref()`** also recursively evaluates forward references nested within the type hint.

See the documentation for **`annotationlib.ForwardRef.evaluate()`** for the meaning of the *owner*, *globals*, *locals*, *type_params*, and *format* parameters.

Prudencia

This function may execute arbitrary code contained in annotations. See Security implications of introspecting annotations for more information.

Added in version 3.14.

typing.NoDefault

Un objeto centinela que se utiliza para indicar que un parámetro de tipo no tiene un valor predeterminado. Por ejemplo:

```
>>> T = TypeVar("T")
>>> T.__default__ is typing.NoDefault
True
>>> S = TypeVar("S", default=None)
>>> S.__default__ is None
True
```

Added in version 3.13.

Constantes

typing.TYPE_CHECKING

A special constant that is assumed to be `True` by 3rd party static type checkers. It's `False` at runtime.

A module which is expensive to import, and which only contain types used for typing annotations, can be safely imported inside an `if TYPE_CHECKING:` block. This prevents the module from actually being imported at runtime; annotations aren't eagerly evaluated (see **PEP 649** [https://peps.python.org/pep-0649/]) so using undefined symbols in annotations is harmless–as long as you don't later examine them. Your static type analysis tool will set `TYPE_CHECKING` to `True` during static type analysis, which means the module will be imported and the types will be checked properly during such analysis.

Uso:

```
if TYPE_CHECKING:
    import expensive_mod

def fun(arg: expensive_mod.SomeType) -> None:
    local_var: expensive_mod.AnotherType = other_fu
```

If you occasionally need to examine type annotations at runtime which may contain undefined symbols, use **`annotationlib.get_annotations()`** with a `format` parameter of **`annotationlib.Format.STRING`** or **`annotationlib.Format.FORWARDREF`** to safely retrieve the annotations without raising **`NameError`**.

Added in version 3.5.2.

Alias obsoletos

This module defines several deprecated aliases to pre-existing standard library classes. These were originally included in the **`typing`** module in order to support parameterizing these generic classes using `[]`. However, the aliases became redundant in Python 3.9 when the corresponding pre-existing classes were enhanced to support `[]` (see **PEP 585** [https://peps.python.org/pep-0585/]).

Los tipos redundantes están obsoletos a partir de Python 3.9. Sin embargo, si bien los alias pueden eliminarse en algún momento, actualmente no se planea eliminarlos. Por lo tanto, el intérprete no emite advertencias de obsolescencia para estos alias.

If at some point it is decided to remove these deprecated aliases, a deprecation warning will be issued by the interpreter for at least two releases prior to removal. The aliases are guaranteed to remain in the **`typing`** module without deprecation warnings until at least Python 3.14.

Se recomienda a los validadores de tipos que marquen los usos de los tipos obsoletos si el programa que están verificando apunta a una versión mínima de Python 3.9 o más reciente.

Alias de tipos integrados

class typing.Dict(*dict, MutableMapping[KT, VT]*)

Alias obsoleto de `dict`.

Tenga en cuenta que para anotar argumentos, se prefiere utilizar un tipo de colección abstracto como `Mapping` en lugar de utilizar `dict` o `typing.Dict`.

Obsoleto desde la versión 3.9: `builtins.dict` ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.List(*list, MutableSequence[T]*)

Alias obsoleto de `list`.

Tenga en cuenta que para anotar argumentos, se prefiere utilizar un tipo de colección abstracto como `Sequence` o `Iterable` en lugar de utilizar `list` o `typing.List`.

Obsoleto desde la versión 3.9: `builtins.list` ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.Set(*set, MutableSet[T]*)

Alias obsoleto de `builtins.set`.

Tenga en cuenta que para anotar argumentos, se prefiere utilizar un tipo de colección abstracto como `collections.abc.Set` en lugar de utilizar `set` o `typing.Set`.

Obsoleto desde la versión 3.9: `builtins.set` ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.FrozenSet(*frozenset, AbstractSet[T_co]*)

Alias obsoleto de `builtins.frozenset`.

Obsoleto desde la versión 3.9: **`builtins.frozenset`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

typing.Tuple

Alias obsoleto de **`tuple`**.

`tuple` y `Tuple` son casos especiales en el sistema de tipos; consulte Anotaciones en tuplas para más detalles.

Obsoleto desde la versión 3.9: **`builtins.tuple`** ahora soporta el uso de subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.Type(*Generic[CT_co]*)

Alias obsoleto de **`type`**.

Vea El tipo de objetos de clase para obtener detalles sobre el uso de **`type`** o `typing.Type` en anotaciones de tipo.

Added in version 3.5.2.

Obsoleto desde la versión 3.9: **`builtins.type`** ahora soporta el uso de subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

Alias de tipos en `collections`

class typing.DefaultDict(*collections.defaultdict, MutableMapping[KT, VT]*)

Alias obsoleto de **`collections.defaultdict`**.

Added in version 3.5.2.

Obsoleto desde la versión 3.9: **`collections.defaultdict`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.OrderedDict(*collections.OrderedDict,* *MutableMapping[KT, VT]*)

Alias obsoleto de `collections.OrderedDict`.

Added in version 3.7.2.

Obsoleto desde la versión 3.9: `collections.OrderedDict` ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.ChainMap(*collections.ChainMap, MutableMapping[KT, VT]*)

Alias obsoleto de `collections.ChainMap`.

Added in version 3.6.1.

Obsoleto desde la versión 3.9: `collections.ChainMap` ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.Counter(*collections.Counter, Dict[T, int]*)

Alias obsoleto de `collections.Counter`.

Added in version 3.6.1.

Obsoleto desde la versión 3.9: `collections.Counter` ahora soporta subíndices (`[]`)`. Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.Deque(*deque, MutableSequence[T]*)

Alias obsoleto de `collections.deque`.

Added in version 3.6.1.

Obsoleto desde la versión 3.9: `collections.deque` ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

Alias a otros tipos concretos

class typing.Pattern
class typing.Match

Alias obsoletos correspondientes a los tipos de retorno **`re.compile()`** y **`re.match()`**.

Estos tipos (y las funciones correspondientes) son genéricos sobre **`AnyStr`**. `Pattern` se puede especializar como `Pattern[str]` o `Pattern[bytes]`; `Match` se puede especializar como `Match[str]` o `Match[bytes]`.

Obsoleto desde la versión 3.9: Las clases `Pattern` y `Match` de **`re`** ahora soportan `[]`. Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.Text

Alias obsoleto para **`str`**.

Se indica `Text` para proporcionar compatibilidad con versiones de código posteriores a Python 2: en Python 2, `Text` es un alias para `unicode`.

Úsese `Text` para indicar que un valor debe contener una cadena de texto Unicode de manera que sea compatible con Python 2 y Python 3:

```
def add_unicode_checkmark(text: Text) -> Text:
    return text + u' \u2713'
```

Added in version 3.5.2.

Obsoleto desde la versión 3.11: Python 2 ya no es compatible, y la mayoría de los validadores de tipos tampoco admiten la verificación de tipos de código Python 2. La eliminación del alias no está planeada actualmente, pero se recomienda a los usuarios utilizar **`str`** en lugar de `Text`.

Alias de ABCs de contenedores en `collections.abc`

class typing.AbstractSet(*Collection[T_co]*)

Alias obsoleto de **`collections.abc.Set`**.

Obsoleto desde la versión 3.9: **`collections.abc.Set`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.ByteString(*Sequence[int]*)

Deprecated alias to **`collections.abc.ByteString`**.

Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use **`Buffer`** or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).

`ByteString` was originally intended to be an abstract class that would serve as a supertype of both **`bytes`** and **`bytearray`**. However, since the ABC never had any methods, knowing that an object was an instance of **`ByteString`** never actually told you anything useful about the object. Other common buffer types such as **`memoryview`** were also never understood as subtypes of **`ByteString`** (either at runtime or by static type checkers).

See **PEP 688** [https://peps.python.org/pep-0688/#current-options] for more details.

Deprecated since version 3.9, will be removed in version 3.17.

class typing.Collection(*Sized, Iterable[T_co], Container[T_co]*)

Alias obsoleto de **`collections.abc.Collection`**.

Added in version 3.6.

Obsoleto desde la versión 3.9: **`collections.abc.Collection`** ahora soporta la sintaxis de subíndice (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.Container(*Generic[T_co]*)

Alias obsoleto de **`collections.abc.Container`**.

Obsoleto desde la versión 3.9: **`collections.abc.Container`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.ItemsView(*MappingView, AbstractSet[tuple[KT_co, VT_co]]*)

Alias obsoleto de **`collections.abc.ItemsView`**.

Obsoleto desde la versión 3.9: **`collections.abc.ItemsView`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.KeysView(*MappingView, AbstractSet[KT_co]*)

Alias obsoleto de **`collections.abc.KeysView`**.

Obsoleto desde la versión 3.9: **`collections.abc.KeysView`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.Mapping(*Collection[KT], Generic[KT, VT_co]*)

Alias obsoleto de **`collections.abc.Mapping`**.

Obsoleto desde la versión 3.9: **`collections.abc.Mapping`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.MappingView(*Sized*)

Alias obsoleto de **`collections.abc.MappingView`**.

Obsoleto desde la versión 3.9: **`collections.abc.MappingView`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.MutableMapping(*Mapping[KT, VT]*)

Alias obsoleto de **`collections.abc.MutableMapping`**.

Obsoleto desde la versión 3.9: **`collections.abc.MutableMapping`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.MutableSequence(*Sequence[T]*)

Alias obsoleto de **`collections.abc.MutableSequence`**.

Obsoleto desde la versión 3.9: **`collections.abc.MutableSequence`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.MutableSet(*AbstractSet[T]*)

Alias obsoleto de **`collections.abc.MutableSet`**.

Obsoleto desde la versión 3.9: **`collections.abc.MutableSet`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.Sequence(*Reversible[T_co], Collection[T_co]*)

Alias obsoleto de **`collections.abc.Sequence`**.

Obsoleto desde la versión 3.9: **`collections.abc.Sequence`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.ValuesView(*MappingView, Collection[_VT_co]*)

Alias obsoleto de **`collections.abc.ValuesView`**.

Obsoleto desde la versión 3.9: **`collections.abc.ValuesView`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

Alias para ABCs asíncronos en `collections.abc`

class typing.Coroutine(*Awaitable[ReturnType], Generic[YieldType, SendType, ReturnType]*)

Alias obsoleto de **`collections.abc.Coroutine`**.

Consulte Anotación de generadores y corrutinas para obtener detalles sobre el uso de **`collections.abc.Coroutine`** y `typing.Coroutine` en anotaciones de tipo.

Added in version 3.5.3.

Obsoleto desde la versión 3.9: **`collections.abc.Coroutine`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.AsyncGenerator(*AsyncIterator[YieldType], Generic[YieldType, SendType]*)

Alias obsoleto de **`collections.abc.AsyncGenerator`**.

Consulte Anotación de generadores y corrutinas para obtener detalles sobre el uso de **`collections.abc.AsyncGenerator`** y `typing.AsyncGenerator` en anotaciones de tipo.

Added in version 3.6.1.

Obsoleto desde la versión 3.9: **`collections.abc.AsycGenerator`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

Distinto en la versión 3.13: El parámetro `SendType` ahora tiene un valor predeterminado.

class typing.AsyncIterable(*Generic[T_co]*)

Alias obsoleto de **`collections.abc.AsyncIterable`**.

Added in version 3.5.2.

Obsoleto desde la versión 3.9:

`collections.abc.AsyncIterable` ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.AsyncIterator(*AsyncIterable[T_co]*)

Alias obsoleto de `collections.abc.AsyncIterator`.

Added in version 3.5.2.

Obsoleto desde la versión 3.9: `collections.abc.AsyncIterator` ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.Awaitable(*Generic[T_co]*)

Alias obsoleto de `collections.abc.Awaitable`.

Added in version 3.5.2.

Obsoleto desde la versión 3.9: `collections.abc.Awaitable` ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

Alias a otros ABCs en `collections.abc`

class typing.Iterable(*Generic[T_co]*)

Alias obsoleto de `collections.abc.Iterable`.

Obsoleto desde la versión 3.9: `collections.abc.Iterable` ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.Iterator(*Iterable[T_co]*)

Alias obsoleto de `collections.abc.Iterator`.

Obsoleto desde la versión 3.9: `collections.abc.Iterator` ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

typing.Callable

Alias obsoleto de **`collections.abc.Callable`**.

Vea Anotaciones en objetos invocables para información detallada de cómo usar **`collections.abc.Callable`** y `typing.Callable` en anotaciones de tipo.

Obsoleto desde la versión 3.9: **`collections.abc.Callable`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

Distinto en la versión 3.10: `Callable` ahora es compatible con **`ParamSpec`** y **`Concatenate`**. Consulte **PEP 612** [https://peps.python.org/pep-0612/] para obtener más información.

class typing.Generator(*Iterator[YieldType], Generic[YieldType, SendType, ReturnType]*)

Alias obsoleto de **`collections.abc.Generator`**.

Consulte Anotación de generadores y corrutinas para obtener detalles sobre el uso de **`collections.abc.Generator`** y `typing.Generator` en anotaciones de tipo.

Obsoleto desde la versión 3.9: **`collections.abc.Generator`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

Distinto en la versión 3.13: Valores por defecto para los tipos send y return fueron agregados.

class typing.Hashable

Alias obsoleto de **`collections.abc.Hashable`**.

Obsoleto desde la versión 3.12: Use directamente **`collections.abc.Hashable`** en su lugar.

class typing.Reversible(*Iterable[T_co]*)

Alias obsoleto de **`collections.abc.Reversible`**.

Obsoleto desde la versión 3.9: **`collections.abc.Reversible`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

class typing.Sized

Alias obsoleto de **`collections.abc.Sized`**.

Obsoleto desde la versión 3.12: Use directamente **`collections.abc.Sized`** en su lugar.

Alias de ABCs `contextlib`

class typing.ContextManager(*Generic[T_co, ExitT_co]*)

Alias obsoleto de **`contextlib.AbstractContextManager`**.

El primer parámetro de tipo, `T_co`, representa el tipo devuelto por el método **`__enter__()`**. El segundo parámetro de tipo opcional, `ExitT_co`, cuyo valor predeterminado es `bool | None`, representa el tipo devuelto por el método **`__exit__()`**.

Added in version 3.5.4.

Obsoleto desde la versión 3.9: **`contextlib.AbstractContextManager`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

Distinto en la versión 3.13: Se agregó el segundo parámetro de tipo opcional, `ExitT_co`.

class typing.AsyncContextManager(*Generic[T_co, AExitT_co]*)

Alias obsoleto de **`contextlib.AbstractAsyncContextManager`**.

El primer parámetro de tipo, `T_co`, representa el tipo devuelto por el método **`__aenter__()`**. El segundo parámetro de tipo opcional, `AExitT_co`, cuyo valor predeterminado es `bool | None`, representa el tipo devuelto por el método **`__aexit__()`**.

Added in version 3.6.2.

Obsoleto desde la versión 3.9: **`contextlib.AbstractAsyncContextManager`** ahora soporta subíndices (`[]`). Véase **PEP 585** [https://peps.python.org/pep-0585/] y Tipo Alias Genérico.

Distinto en la versión 3.13: Se agregó el segundo parámetro de tipo opcional, `AExitT_co`.

Línea de tiempo de obsolescencia de características principales

Algunas características de `typing` están obsoletas y podrán ser removidas en versiones futuras de Python. Lo que sigue es una tabla que resume las principales obsolescencias para su conveniencia. Ésto está sujeto a cambio y no todas las obsolescencias están representadas.

Característica	Obsoleto desde	Remoción proyectada	PEP/issue
Versiones de `typing` de colecciones estándares	3.9	No decidido (ver Alias obsoletos para más información)	**PEP 585**
`typing.ByteString`	3.9	3.17	gh-91896
`typing.Text`	3.11	No decidido	gh-92332
`typing.Hashable` y **`typing.Sized`**	3.12	No decidido	gh-94309
`typing.TypeAlias`	3.12	No decidido	**PEP 695**
`@typing.no_type_check_decorator`	3.13	3.15	gh-106309
`typing.AnyStr`	3.13	3.18	gh-105578

`pydoc` — Documentation generator and online help system

Código fuente: Lib/pydoc.py [https://github.com/python/cpython/tree/3.14/Lib/pydoc.py]

The `pydoc` module automatically generates documentation from Python modules. The documentation can be presented as pages of text on the console, served to a web browser, or saved to HTML files.

For modules, classes, functions and methods, the displayed documentation is derived from the docstring (i.e. the `__doc__` attribute) of the object, and recursively of its documentable members. If there is no docstring, `pydoc` tries to obtain a description from the block of comment lines just above the definition of the class, function or method in the source file, or at the top of the module (see `inspect.getcomments()`).

The built-in function `help()` invokes the online help system in the interactive interpreter, which uses `pydoc` to generate its documentation as text on the console. The same text documentation can also be viewed from outside the Python interpreter by running **pydoc** as a script at the operating system's command prompt. For example, running

```
python -m pydoc sys
```

en la entrada de la consola mostrará la documentación sobre el módulo `sys`, en un estilo similar a las páginas del manual que se muestran por el comando **man** de Unix. El argumento de **pydoc** puede ser el nombre de una función, módulo, o paquete, o una referencia con puntos (*dotted reference*) de una clase, método, o función dentro de un módulo o módulo en un paquete. Si el argumento de **pydoc** se parece a una ruta (*path*) (es decir, que contiene el separador de rutas para tu sistema operativo como una barra en Unix), y hace referencia a un archivo fuente de Python existente, entonces se produce la documentación para ese archivo.

Nota

In order to find objects and their documentation, **`pydoc`** imports the module(s) to be documented. Therefore, any code on module level will be executed on that occasion. Use an `if __name__ == '__main__':` guard to only execute code when a file is invoked as a script and not just imported.

When printing output to the console, **pydoc** attempts to paginate the output for easier reading. If either the **`MANPAGER`** or the **`PAGER`** environment variable is set, **pydoc** will use its value as a pagination program. When both are set, **`MANPAGER`** is used.

Especificar un bandera (*flag*) `-w` antes del argumento hará que la documentación HTML sea escrita en un archivo de la carpeta actual, en vez de mostrar el texto en la consola.

Especificar una bandera (*flag*) `-k` antes del argumento buscará las líneas de la sinopsis de todos los módulos disponibles por la palabra clave (*keyword*) dada como argumento, de nuevo en una manera similar al comando **man** de Unix. La sinopsis de un módulo es la primera línea de su cadena de documentación.

You can also use **pydoc** to start an HTTP server on the local machine that will serve documentation to visiting web browsers. **python -m pydoc -p 1234** will start a HTTP server on port 1234, allowing you to browse the documentation at `http://localhost:1234/` in your preferred web browser. Specifying `0` as the port number will select an arbitrary unused port.

python -m pydoc -n <hostname> will start the server listening at the given hostname. By default the hostname is “localhost” but if you want the server to be reached from other machines, you may want to change the host name that the server responds to. During development this is especially useful if you want to run pydoc from within a container.

python -m pydoc -b will start the server and additionally open a web browser to a module index page. Each served page has a navigation bar at the top where you can *Get* help on an individual item, *Search* all modules with a keyword in their synopsis line, and go to the *Module index*, *Topics* and *Keywords* pages.

Cuando **pydoc** genera la documentación, se usa el entorno y ruta actual para localizar los módulos. Así, invocar **pydoc spam** precisamente documenta la versión del módulo que tu obtendrías si empezaras el interpretador de Python y escribieras `import spam`.

Module docs for core modules are assumed to reside in `https://docs.python.org/X.Y/library/` where `X` and `Y` are the major and minor version numbers of the Python interpreter. This can be overridden by setting the **`PYTHONDOCS`** environment variable to a different URL or to a local directory containing the Library Reference Manual pages.

Distinto en la versión 3.2: Se añadió la opción `-b`.

Distinto en la versión 3.3: La opción de la línea de comandos `-g` se eliminó.

Distinto en la versión 3.4: **`pydoc`** now uses **`inspect.signature()`** rather than **`inspect.getfullargspec()`** to extract signature information from callables.

Distinto en la versión 3.7: Se añadió la opción `-n`.

Modo de desarrollo de Python

Added in version 3.7.

El modo de desarrollo de Python introduce comprobaciones adicionales en tiempo de ejecución que son muy costosas para ser activadas por defecto. No debería ser más verboso que el predeterminado si el código es correcto; sólo se emiten nuevas advertencias cuando se detecta un problema.

Puede activarse mediante la opción de línea de comandos **`-X dev`** o estableciendo la variable de entorno **`PYTHONDEVMODE`** en `1`.

Consulte también Compilación de depuración de Python.

Efectos del modo de desarrollo de Python

Activar el modo de desarrollo de Python es similar al siguiente comando, pero con efectos adicionales que se describen a continuación:

```
PYTHONMALLOC=debug PYTHONASYNCIODEBUG=1 python -W defaul
```

Efectos del modo de desarrollo de Python:

- Añadir `default` filtro de avisos.Se muestran las siguientes advertencias:
  - **`DeprecationWarning`**
  - **`ImportWarning`**
  - **`PendingDeprecationWarning`**
  - **`ResourceWarning`**

  Normalmente, los advertencias son filtradas por defecto warning filters.

  Se comporta como si se utilizara la opción de línea de comandos **`-W default`**.

Utilice la opción de línea de comandos **`-W error`** o establezca la variable de entorno **`PYTHONWARNINGS`** en `error` para tratar las advertencias como errores.

- Instalar hooks(enganches) de depuración en los asignadores de memoria para comprobar:

  - Desbordamiento del búfer
  - Sobrecarga del búfer
  - Violación de la API del asignador de memoria
  - Uso inseguro del GIL

  Ver la función en C **`PyMem_SetupDebugHooks()`**.

  Se comporta como si la variable de entorno **`PYTHONMALLOC`** estuviera establecida en `debug`.

  Para activar el modo de desarrollo de Python sin instalar ganchos de depuración en los asignadores de memoria, establezca la variable de entorno **`PYTHONMALLOC`** a `default`.

- Llama a **`faulthandler.enable()`** al inicio de Python para instalar los handlers(manejadores) de las señales **`SIGSEGV`**, **`SIGFPE`**, **`SIGABRT`**, **`SIGBUS`** y **`SIGILL`** para volcar la traza de Python en caso de fallo.

  Se comporta como si se utilizara la opción de línea de comandos **`-X faulthandler`** o si la variable de entorno **`PYTHONFAULTHANDLER`** se establece en `1`.

- Habilitar asyncio debug mode. Por ejemplo, **`asyncio`** comprueba las corutinas que no fueron esperadas y las registra.

  Se comporta como si la variable de entorno **`PYTHONASYNCIODEBUG`** estuviera establecida en `1`.

- Comprueba los argumentos *encoding* y *errors* para las operaciones de codificación y decodificación de cadenas.

Ejemplos: **`open()`**, **`str.encode()`** y **`bytes.decode()`**.

Por defecto, para un mejor rendimiento, el argumento *errors* sólo se comprueba en el primer error de codificación/ decodificación y el argumento *encoding* a veces se ignora para las cadenas vacías.

- El destructor de **`io.IOBase`** registra las excepciones `close()`.
- Establece el atributo **`dev_mode`** de **`sys.flags`** a `True`.

El Modo de Desarrollo de Python no habilita el módulo **`tracemalloc`** por defecto, porque el costo de la sobrecarga (para el rendimiento y la memoria) sería demasiado grande. Activar el módulo **`tracemalloc`** proporciona información adicional sobre el origen de algunos errores. Por ejemplo, **`ResourceWarning`** registra la traza donde se asignó el recurso, y un error de desbordamiento de búfer registra la traza donde se asignó el bloque de memoria.

El modo de desarrollo de Python no impide que la opción de línea de comandos **`-O`** elimine las declaraciones **`assert`** ni que establezca **`__debug__`** a `False`.

El modo de desarrollo de Python solo se puede habilitar en el inicio de Python. Su valor se puede leer en **`sys.flags.dev_mode`**.

Distinto en la versión 3.8: El destructor de **`io.IOBase`** ahora registra las excepciones `close()`.

Distinto en la versión 3.9: Los argumentos *enconding* y *errors* se comprueban ahora para las operaciones de codificación y descodificación de cadenas.

Ejemplo de ResourceWarning

Ejemplo de un script que cuenta el número de líneas del archivo de texto especificado en la línea de comandos:

```
import sys
```

```
def main():
    fp = open(sys.argv[1])
    nlines = len(fp.readlines())
    print(nlines)
    # The file is closed implicitly

if __name__ == "__main__":
    main()
```

El script no cierra el archivo explícitamente. Por defecto, Python no emite ninguna advertencia. Ejemplo usando README.txt, que tiene 269 líneas:

```
$ python script.py README.txt
269
```

Al activar el modo de desarrollo de Python aparece una advertencia `ResourceWarning`:

```
$ python -X dev script.py README.txt
269
script.py:10: ResourceWarning: unclosed file <_io.TextIO
  main()
ResourceWarning: Enable tracemalloc to get the object al
```

Además, al activar `tracemalloc` se muestra la línea en la que se abrió el archivo:

```
$ python -X dev -X tracemalloc=5 script.py README.rst
269
script.py:10: ResourceWarning: unclosed file <_io.TextIO
  main()
Object allocated at (most recent call last):
  File "script.py", lineno 10
    main()
  File "script.py", lineno 4
    fp = open(sys.argv[1])
```

La solución es cerrar explícitamente el archivo. Ejemplo utilizando un gestor de contexto:

```
def main():
    # Close the file explicitly when exiting the with bl
    with open(sys.argv[1]) as fp:
        nlines = len(fp.readlines())
    print(nlines)
```

No cerrar un recurso explícitamente puede dejar un recurso abierto durante mucho más tiempo del estimado; puede causar graves problemas al salir de Python. Es malo en CPython, pero es aún peor en PyPy. Cerrar los recursos explícitamente hace que una aplicación sea más detallista y más fiable.

Ejemplo de error de descriptor de archivo incorrecto

Script que muestra la primera línea de sí mismo:

```
import os

def main():
    fp = open(__file__)
    firstline = fp.readline()
    print(firstline.rstrip())
    os.close(fp.fileno())
    # The file is closed implicitly

main()
```

Por defecto, Python no emite ninguna advertencia:

```
$ python script.py
import os
```

El modo de desarrollo de Python muestra un `ResourceWarning` y registra un error «Bad file descriptor» cuando termina el objeto archivo:

```
$ python -X dev script.py
import os
script.py:10: ResourceWarning: unclosed file <_io.TextIO
  main()
ResourceWarning: Enable tracemalloc to get the object al
Exception ignored in: <_io.TextIOWrapper name='script.py
Traceback (most recent call last):
  File "script.py", line 10, in <module>
    main()
OSError: [Errno 9] Bad file descriptor
```

`os.close(fp.fileno())` cierra el descriptor de archivo. Cuando el finalizador de objetos de archivo intenta cerrar el descriptor de archivo de nuevo, falla con el error `Bad file descriptor`. Un descriptor de archivo debe cerrarse sólo una vez. En el peor de los casos, cerrarlo dos veces puede provocar un fallo (ver bpo-18748 [https://bugs.python.org/issue?@action=redirect&bpo=18748] para un ejemplo).

La solución es eliminar la línea `os.close(fp.fileno())`, o abrir el archivo con `closefd=False`.

[1]

`doctest` — Test interactive Python examples

Código fuente: Lib/doctest.py [https://github.com/python/cpython/tree/3.14/Lib/doctest.py]

The `doctest` module searches for pieces of text that look like interactive Python sessions, and then executes those sessions to verify that they work exactly as shown. There are several common ways to use doctest:

- Para revisar que los docstrings de un módulo están al día al verificar que todos los ejemplos interactivos todavía trabajan como está documentado.
- Para realizar pruebas de regresión al verificar que los ejemplos interactivos de un archivo de pruebas o un objeto de pruebas trabaje como sea esperado.
- Para escribir documentación de tutorial para un paquete, generosamente ilustrado con ejemplos de entrada-salida. Dependiendo si los ejemplos o el texto expositivo son enfatizados, tiene el sabor de «pruebas literarias» o «documentación ejecutable».

Aquí hay un módulo de ejemplo completo pero pequeño:

```
"""
This is the "example" module.

The example module supplies one function, factorial().

>>> factorial(5)
120
"""
```

```
def factorial(n):
    """Return the factorial of n, an exact integer >= 0.

    >>> [factorial(n) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> factorial(30)
    265252859812191058636308480000000
    >>> factorial(-1)
    Traceback (most recent call last):
        ...
    ValueError: n must be >= 0

    Factorials of floats are OK, but the float must be a
    >>> factorial(30.1)
    Traceback (most recent call last):
        ...
    ValueError: n must be exact integer
    >>> factorial(30.0)
    265252859812191058636308480000000

    It must also not be ridiculously large:
    >>> factorial(1e100)
    Traceback (most recent call last):
        ...
    OverflowError: n too large
    """

    import math
    if not n >= 0:
        raise ValueError("n must be >= 0")
    if math.floor(n) != n:
        raise ValueError("n must be exact integer")
    if n+1 == n:  # catch a value like 1e300
        raise OverflowError("n too large")
    result = 1
    factor = 2
    while factor <= n:
        result *= factor
        factor += 1
    return result


if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

If you run `example.py` directly from the command line, **`doctest`** works its magic:

```
$ python example.py
$
```

There's no output! That's normal, and it means all the examples worked. Pass `-v` to the script, and **`doctest`** prints a detailed log of what it's trying, and prints a summary at the end:

```
$ python example.py -v
Trying:
    factorial(5)
Expecting:
    120
ok
Trying:
    [factorial(n) for n in range(6)]
Expecting:
    [1, 1, 2, 6, 24, 120]
ok
```

Y demás, eventualmente terminando con:

```
Trying:
    factorial(1e100)
Expecting:
    Traceback (most recent call last):
        ...
    OverflowError: n too large
ok
2 items passed all tests:
   1 test in __main__
   6 tests in __main__.factorial
7 tests in 2 items.
7 passed.
Test passed.
$
```

That's all you need to know to start making productive use of **`doctest`**! Jump in. The following sections provide full details. Note that there are many examples of doctests in the standard Python test suite and libraries. Especially useful examples can be found in the standard test file `Lib/test/test_doctest/test_doctest.py`.

Added in version 3.13: Output is colorized by default and can be controlled using environment variables.

Uso simple: verificar ejemplos en docstrings

The simplest way to start using doctest (but not necessarily the way you'll continue to do it) is to end each module **`M`** with:

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

`doctest` then examines docstrings in module **`M`**.

Ejecutar el módulo como un script causa que los ejemplos en los docstrings se ejecuten y sean verificados:

```
python M.py
```

No mostrará nada a menos que un ejemplo falle, en cuyo caso el ejemplo fallido o ejemplos fallidos y sus causas de la falla son imprimidas a stdout, y la línea final de salida es `***Test Failed*** N failures.`, donde *N* es el número de ejemplos que fallaron.

Ejecútalo con el modificador `-v` en su lugar:

```
python M.py -v
```

y un reporte detallado de todos los ejemplos intentados es impreso a la salida estándar, junto con resúmenes clasificados al final.

You can force verbose mode by passing `verbose=True` to **`testmod()`**, or prohibit it by passing `verbose=False`. In either of those cases, **`sys.argv`** is not examined by **`testmod()`** (so passing `-v` or not has no effect).

There is also a command line shortcut for running **`testmod()`**, see section Command-line Usage.

Para más información de **`testmod()`**, consulte la sección API básica.

Uso Simple: Verificar ejemplos en un Archivo de Texto

Otra simple aplicación de doctest es probar ejemplos interactivos en un archivo de texto. Esto puede ser hecho con la función **`testfile()`**:

```
import doctest
doctest.testfile("example.txt")
```

Este script corto ejecuta y verifica cualquier ejemplo interactivo de Python contenido en el archivo `example.txt`. El contenido del archivo es tratado como si fuese un solo gran docstring; ¡el archivo no necesita contener un programa de Python! Por ejemplo, tal vez `example.txt` contenga esto:

```
The ``example`` module
======================

Using ``factorial``
-------------------
```

```
This is an example text file in reStructuredText format.
``factorial`` from the ``example`` module:

    >>> from example import factorial

Now use it:

    >>> factorial(6)
    120
```

Ejecutar `doctest.testfile("example.txt")` entonces encuentra el error en esta documentación:

```
File "./example.txt", line 14, in example.txt
Failed example:
    factorial(6)
Expected:
    120
Got:
    720
```

As with **`testmod()`**, **`testfile()`** won’t display anything unless an example fails. If an example does fail, then the failing example(s) and the cause(s) of the failure(s) are printed to stdout, using the same format as **`testmod()`**.

Por defecto, **`testfile()`** busca archivos en el directorio del módulo al que se llama. Véase la sección API básica para una descripción de los argumentos opcionales que pueden ser usados para decirle que busque archivos en otros lugares.

Como **`testmod()`**, la verbosidad de **`testfile()`** puede ser establecida con el modificador de la línea de comandos `-v` o con el argumento por palabra clave opcional *verbose*.

There is also a command line shortcut for running **`testfile()`**, see section Command-line Usage.

Para más información en **`testfile()`**, véase la sección API básica.

Command-line Usage

The **`doctest`** module can be invoked as a script from the command line:

```
python -m doctest [-v] [-o OPTION] [-f] file [file ...]
```

-v, --verbose

Detailed report of all examples tried is printed to standard output, along with assorted summaries at the end:

```
python -m doctest -v example.py
```

This will import `example.py` as a standalone module and run **`testmod()`** on it. Note that this may not work correctly if the file is part of a package and imports other submodules from that package.

If the file name does not end with `.py`, **`doctest`** infers that it must be run with **`testfile()`** instead:

```
python -m doctest -v example.txt
```

-o, --option <option>

Option flags control various aspects of doctest’s behavior, see section Banderas de Opción.

Added in version 3.4.

-f, --fail-fast

This is shorthand for `-o FAIL_FAST`.

Added in version 3.4.

Cómo funciona

Esta sección examina en detalle cómo funciona doctest: qué docstrings revisa, cómo encuentra ejemplos interactivos, qué contexto de ejecución usa, cómo maneja las excepciones, y cómo las banderas pueden ser usadas para controlar su comportamiento. Esta es la información que necesitas saber para escribir ejemplos de doctest; para información sobre ejecutar doctest en estos ejemplos, véase las siguientes secciones.

¿Qué docstrings son examinados?

Se busca en el docstring del módulo, y todos los docstrings de las funciones, clases, y métodos. Los objetos importados en el módulo no se buscan.

In addition, there are cases when you want tests to be part of a module but not part of the help text, which requires that the tests not be included in the docstring. Doctest looks for a module-level variable called `__test__` and uses it to locate other tests. If `M.__test__` exists, it must be a dict, and each entry maps a (string) name to a function object, class object, or string. Function and class object docstrings found from `M.__test__` are searched, and strings are treated as if they were docstrings. In output, a key `K` in `M.__test__` appears with name `M.__test__.K`.

For example, place this block of code at the top of `example.py`:

```
__test__ = {
    'numbers': """
>>> factorial(6)
720

>>> [factorial(n) for n in range(6)]
[1, 1, 2, 6, 24, 120]
"""
}
```

The value of `example.__test__["numbers"]` will be treated as a docstring and all the tests inside it will be run. It is important to note that the value can be mapped to a function, class object, or module; if so, **`doctest`** searches them recursively for docstrings, which are then scanned for tests.

Todas las clases encontradas se buscan recursivamente de manera similar, para probar docstrings en sus métodos contenidos y clases anidadas.

Nota

`doctest` can only automatically discover classes and functions that are defined at the module level or inside other classes.

Since nested classes and functions only exist when an outer function is called, they cannot be discovered. Define them outside to make them visible.

¿Cómo se reconocen los ejemplos de docstring?

En la mayoría de los casos un copiar y pegar de una sesión de consola interactiva funciona bien, pero doctest no está intentando hacer una emulación exacta de ningún shell específico de Python.

```
>>> # comments are ignored
>>> x = 12
>>> x
12
>>> if x == 13:
...     print("yes")
... else:
...     print("no")
...     print("NO")
...     print("NO!!!")
...
no
NO
NO!!!
>>>
```

Cualquier salida esperada debe seguir inmediatamente el final de la línea `'>>>'` o `'...'` conteniendo el código, y la salida esperada (si la hubiera) se extiende hasta el siguiente `'>>>'` o la línea en blanco.

La letra pequeña:

- La salida esperada no puede contener una línea de espacios en blanco, ya que ese tipo de línea se toma para indicar el fin de la salida esperada. Si la salida esperada de verdad contiene una línea en blanco, pon `<BLANKLINE>` en tu ejemplo de doctest en cada lugar donde una línea en blanco sea esperada.

- Todos los caracteres de tabulación se expanden a espacios, usando paradas de tabulación de 8 -columnas. Las tabulaciones generadas por el código en pruebas no son modificadas. Ya que todas las tabulaciones en la salida de prueba *son* expandidas, significa que si el código de salida incluye tabulaciones, la única manera de que el doctest pueda pasar es si la opción **`NORMALIZE_WHITESPACE`** o directive está en efecto. Alternativamente, la prueba puede ser reescrita para capturar la salida y compararla a un valor esperado como parte de la prueba. Se llegó a este tratamiento de tabulaciones en la fuente a través de prueba y error, y ha demostrado ser la manera menos propensa a errores de manejarlos. Es posible usar un algoritmo diferente para manejar tabulaciones al escribir una clase **`DocTestParser`** personalizada.

- La salida a stdout es capturada, pero no la salida a stderr (los rastreos de la excepción son capturados a través de maneras diferentes).

- Si continuas una línea poniendo una barra invertida en una sesión interactiva, o por cualquier otra razón usas una barra invertida, debes usar un docstring crudo, que preservará tus barras invertidas exactamente como las escribes:

  ```
  >>> def f(x):
  ...     r'''Backslashes in a raw docstring: m\n'''
  ...
  >>> print(f.__doc__)
  Backslashes in a raw docstring: m\n
  ```

  De otra manera, la barra invertida será interpretada como parte de una cadena. Por ejemplo, el `\n` arriba sería interpretado como un carácter de nueva línea. Alternativamente, puedes duplicar cada barra invertida en la versión de doctest (y no usar una cadena de caracteres cruda):

```
>>> def f(x):
...     '''Backslashes in a raw docstring: m\\n'''
...
>>> print(f.__doc__)
Backslashes in a raw docstring: m\n
```

- La columna inicial no importa:

  ```
  >>> assert "Easy!"
        >>> import math
            >>> math.floor(1.9)
            1
  ```

  y tantos espacios en blanco al principio se eliminan de la salida esperada como aparece en la línea `'>>>'` inicial que empezó el ejemplo.

¿Cuál es el contexto de ejecución?

By default, each time **`doctest`** finds a docstring to test, it uses a *shallow copy* of **`M`**’s globals, so that running tests doesn’t change the module’s real globals, and so that one test in **`M`** can’t leave behind crumbs that accidentally allow another test to work. This means examples can freely use any names defined at top-level in **`M`**, and names defined earlier in the docstring being run. Examples cannot see names defined in other docstrings.

Puedes forzar el uso de tus propios diccionarios como contexto de ejecución al pasar `globs=your_dict` a **`testmod()`** o **`testfile()`** en su lugar.

¿Y las excepciones?

No hay problema, siempre que el rastreo sea la única salida producida por el ejemplo: sólo copia el rastreo. [1] Ya que los rastreos contienen detalles que probablemente cambien rápidamente (por ejemplo, rutas de archivos exactas y números de línea), este es un caso donde doctest trabaja duro para ser flexible en lo que acepta.

Ejemplo simple:

```
>>> [1, 2, 3].remove(42)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

El doctest tiene éxito si se lanza **`ValueError`**, con el detalle `list.remove(x): x not in list` como se muestra.

La salida esperada para una excepción debe empezar con una cabecera de rastreo, que puede ser una de las siguientes dos líneas, con el mismo sangrado de la primera línea del ejemplo:

```
Traceback (most recent call last):
Traceback (innermost last):
```

La cabecera de rastreo es seguida por una pila de rastreo opcional, cuyo contenido es ignorado por doctest. La pila de rastreo es típicamente omitida, o copiada palabra por palabra de una sesión interactiva.

La pila de rastreo es seguida por la parte más interesante: la línea o líneas conteniendo el tipo de excepción y detalle. Esto es usualmente la última línea de un rastreo, pero se puede extender a través de múltiples líneas si la excepción tiene un detalle de varias líneas:

```
>>> raise ValueError('multi\n    line\ndetail')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: multi
    line
detail
```

Las últimas tres líneas (empezando con **`ValueError`**) son comparados con el tipo de excepción y detalle, y el resto es ignorado.

La mejor práctica es omitir la pila de rastreo, a menos que añada valor de documentación significante al ejemplo. Por lo que el último ejemplo es probablemente mejor como:

```
>>> raise ValueError('multi\n    line\ndetail')
Traceback (most recent call last):
    ...
ValueError: multi
    line
detail
```

Note que los rastreos son tratados de manera especial. En particular, en el ejemplo reescrito, el uso de `...` es independiente de la opción **`ELLIPSIS`** de doctest. Se pueden excluir los puntos suspensivos en ese ejemplo, así como también pueden haber tres (o trescientas) comas o dígitos, o una transcripción sangrada de un *sketch* de Monty Python.

Algunos detalles que debes leer una vez, pero no necesitarás recordar:

- Doctest no puede adivinar si tu salida esperada vino de una excepción de rastreo o de una impresión ordinaria. Así que, un ejemplo que espera `ValueError: 42 is prime` pasará, ya sea si de hecho se lance **`ValueError`** o si el ejemplo simplemente imprime ese texto de rastreo. En la práctica, la salida ordinaria raramente comienza con una línea de cabecera de rastreo, por lo que esto no crea problemas reales.
- Cada línea de la pila de rastreo (si se presenta) debe estar más sangrada que la primera línea del ejemplo, *o* empezar con un carácter no alfanumérico. la primera línea que sigue a la cabecera de rastreo sangrada de igual forma y empezando con un alfanumérico es considerado el inicio del detalle de la excepción. Por supuesto que esto es lo correcto para rastreos genuinos.
- Cuando se especifica la opción **`IGNORE_EXCEPTION_DETAIL`** de doctest. todo lo que sigue a

los dos puntos más a la izquierda y cualquier otra información del módulo en el nombre de la excepción se ignora.

- The interactive shell omits the traceback header line for some **`SyntaxError`**s. But doctest uses the traceback header line to distinguish exceptions from non-exceptions. So in the rare case where you need to test a **`SyntaxError`** that omits the traceback header, you will need to manually add the traceback header line to your test example.

- En el caso de algunas excepciones Python señala la posición donde ocurrió el error usando un marcador `^`:

  ```
  >>> 1 + None
    File "<stdin>", line 1
      1 + None
      ~~^~~~~~
  TypeError: unsupported operand type(s) for +: 'int'
  ```

  Ya que las líneas mostrando la posición del error vienen antes del tipo de excepción y detalle, no son revisadas por doctest. Por ejemplo, el siguiente test pasaría, a pesar de que pone el marcador `^` en la posición equivocada:

  ```
  >>> 1 + None
    File "<stdin>", line 1
      1 + None
      ^~~~~~~~
  TypeError: unsupported operand type(s) for +: 'int'
  ```

Banderas de Opción

Varias banderas de opción controlan diversos aspectos del comportamiento de doctest. Los nombres simbólicos para las banderas son proporcionados como constantes del módulo, que se pueden conectar mediante *OR* bit a bit y pasar a varias funciones. Los nombres también pueden ser usados en las directivas de doctest, y se pueden pasar a la interfaz de la línea de comandos de doctest a través de la opción `-o`.

El primer grupo de opciones definen las semánticas de la prueba, controlando aspectos de cómo doctest decide si la salida de hecho concuerda con la salida esperada del ejemplo:

doctest.DONT_ACCEPT_TRUE_FOR_1

Por defecto, si un bloque de salida esperada contiene sólo `1`, se considera igual a un bloque de salida real conteniendo sólo `1` o `true`, y similarmente para `0` contra `False`. Cuando se especifica **`DONT_ACCEPT_TRUE_FOR_1`**`` ` ``, no se permite ninguna sustitución. El comportamiento por defecto atiende a que Python cambió el tipo de retorno de muchas funciones de enteros a booleanos; los doctest esperando salidas «de pequeño enteros» todavía trabajan en estos casos. Esta opción probablemente se vaya, pero no por muchos años.

doctest.DONT_ACCEPT_BLANKLINE

Por defecto, si un bloque de salida esperada contiene una línea que sólo tiene la cadena de caracteres `<BLANKLINE>`, entonces esa línea corresponderá a una línea en blanco en la salida real. Ya que una línea en blanca auténtica delimita la salida esperada, esta es la única manera de comunicar que una línea en blanco es esperada. Cuando se especifica **`DONT_ACCEPT_BLANKLINE`**, esta substitución no se permite.

doctest.NORMALIZE_WHITESPACE

Cuando se especifica, todas las secuencias de espacios en blanco (vacías y nuevas líneas) son tratadas como iguales. Cualquier secuencia de espacios en blanco dentro de la salida esperada corresponderá a cualquier secuencia de espacios en blanco dentro de la salida real. Por defecto, los espacios en blanco deben corresponderse exactamente. **`NORMALIZE_WHITESPACE`** es especialmente útil cuando una línea de la salida esperada es muy larga, y quieres envolverla a través de múltiples líneas en tu código fuente.

doctest.ELLIPSIS

Cuando se especifica, un marcador de puntos suspensivos (`...`) en la salida esperada puede corresponder a cualquier cadena de caracteres en la salida real. Esto incluye las cadenas de caracteres que abarcan límites de líneas, y cadenas de caracteres vacías, por lo que es mejor mantener su uso simple. Usos complicados pueden conducir a los mismo tipos de sorpresa de «ups, coincidió demasiado» que `.*` es propenso a hacer en expresiones regulares.

doctest.IGNORE_EXCEPTION_DETAIL

Cuando se especifica, las excepciones que doctest espera, pasarán siempre que sea lanzada una excepción del tipo esperado, incluso si los detalles (mensaje y nombre completo de la excepción) no coinciden.

Por ejemplo, si se espera `ValueError: 42`, el test pasará si la verdadera excepción lanzada es `ValueError: 3*14`, pero fallará si es lanzada un **`TypeError`**. También ignorará cualquier nombre completo incluido antes de la clase de la excepción, lo cual puede variar entre implementaciones y versiones de Python y bibliotecas/código en uso. Por lo tanto, las tres variaciones funcionarán con la bandera especificada:

```
>>> raise Exception('message')
Traceback (most recent call last):
Exception: message

>>> raise Exception('message')
Traceback (most recent call last):
builtins.Exception: message

>>> raise Exception('message')
Traceback (most recent call last):
__main__.Exception: message
```

Tenga en cuenta que **`ELLIPSIS`** también puede ser usado para ignorar detalles del mensaje de la excepción, pero dicha prueba aún podría fallar si el nombre del modulo está presente o si hay coincidencias exactas.

Distinto en la versión 3.2: **`IGNORE_EXCEPTION_DETAIL`** también ignora cualquier información relacionada al módulo conteniendo la excepción bajo prueba.

doctest.SKIP

Cuando se especifica, no ejecuta el ejemplo del todo. Puede ser útil en contextos donde los ejemplos de doctest sirven como documentación y casos de prueba a la vez, y un ejemplo debe ser incluido para propósitos de documentación, pero no debe ser revisado. P. ej., la salida del ejemplo puede ser aleatoria, o el ejemplo puede depender de recursos que no estarían disponibles para el controlador de pruebas.

La bandera *SKIP* también se puede usar para temporalmente «quitar» ejemplos.

doctest.COMPARISON_FLAGS

Una máscara de bits o juntadas lógicamente todas las banderas de arriba.

El segundo grupo de opciones controla cómo las fallas de las pruebas son reportadas:

doctest.REPORT_UDIFF

Cuando se especifica, las fallas que involucran salidas multilínea esperadas y reales son mostradas usando una diferencia (*diff*) unificada.

doctest.REPORT_CDIFF

Cuando se especifica, las fallas que involucran salidas multilínea esperadas y reales se mostrarán usando una diferencia (*diff*) contextual.

doctest.REPORT_NDIFF

Cuando se especifica, las diferencias son computadas por `difflib.Differ`, usando el mismo algoritmo que la popular utilidad `ndiff.py`. Este es el único método que marca diferencias dentro de líneas también como a través de líneas. Por ejemplo, si una línea de salida esperada contiene el dígito `1` donde la salida actual contiene la letra `l`, se inserta una línea con una marca de inserción marcando la posición de las columnas que no coinciden.

doctest.REPORT_ONLY_FIRST_FAILURE

Cuando se especifica, muestra el primer ejemplo fallido en cada doctest, pero suprime la salida para todos ejemplos restantes. Esto evitará que doctest reporte los ejemplos correctos que se rompen por causa de fallos tempranos; pero también puede esconder ejemplos incorrectos que fallen independientemente de la primera falla. Cuando se especifica **`REPORT_ONLY_FIRST_FAILURE`**, los ejemplos restantes aún se ejecutan, y aún cuentan para el número total de fallas reportadas, sólo se suprime la salida.

doctest.FAIL_FAST

Cuando se especifica, sale después del primer ejemplo fallido y no intenta ejecutar los ejemplos restantes. Por consiguiente, el número de fallas reportadas será como mucho 1. Esta bandera puede ser útil durante la depuración, ya que los ejemplos después de la primera falla ni siquiera producirán salida de depuración.

doctest.REPORTING_FLAGS

Una máscara de bits o todas las banderas de reporte arriba combinadas.

There is also a way to register new option flag names, though this isn't useful unless you intend to extend **`doctest`** internals via subclassing:

doctest.register_optionflag(*name*)

Crea una nueva bandera de opción con un nombre dado, y retorna el valor entero de la nueva bandera. se puede usar **`register_optionflag()`** cuando se hereda **`OutputChecker`** o **`DocTestRunner`** para crear nuevas opciones que sean compatibles con tus clases heredadas. **`register_optionflag()`** siempre debe ser llamado usando la siguiente expresión:

```
MY_FLAG = register_optionflag('MY_FLAG')
```

Directivas

Se pueden usar las directivas de doctest para modificar las banderas de opción para un ejemplo individual. Las directivas de doctest son comentarios de Python especiales que siguen el código fuente de un ejemplo:

```
directive:             "#" "doctest:" directive_options
directive_options:     directive_option ("," directive_o
directive_option:      on_or_off directive_option_name
on_or_off:             "+" | "-"
directive_option_name: "DONT_ACCEPT_BLANKLINE" | "NORMAL
```

No se permite el espacio en blanco entre el `+` o `-` y el nombre de la opción de directiva (*directive option name*). El nombre de la opción de directiva puede ser cualquiera de los nombres de las banderas de opciones explicadas arriba.

Las directivas de doctest de un ejemplo modifican el comportamiento de doctest para ese único ejemplo. Usa `+` para habilitar el comportamiento nombrado, o `-` para deshabilitarlo.

Por ejemplo, esta prueba se ejecuta con normalidad:

```
>>> print(list(range(20)))  # doctest: +NORMALIZE_WHITES
[0,   1,  2,  3,  4,  5,  6,  7,  8,  9,
10,  11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Sin la directiva esto fallaría, porque de hecho la salida verdadera no tiene dos espacios en blanco antes de cada elemento de la lista, y además porque la la salida verdadera está en una sola línea. Mientras tanto, la siguiente prueba también pasa, y de la misma forma requiere de directivas para hacerlo:

```
>>> print(list(range(20)))  # doctest: +ELLIPSIS
[0, 1, ..., 18, 19]
```

Se pueden usar varias directivas en una sola línea, separadas por comas:

```
>>> print(list(range(20)))  # doctest: +ELLIPSIS, +NORMA
[0,    1, ...,   18,    19]
```

Si las diferentes directivas se usan para un sólo ejemplo, entonces estas se combinarán:

```
>>> print(list(range(20)))  # doctest: +ELLIPSIS
...                         # doctest: +NORMALIZE_WHITES
[0,    1, ...,   18,    19]
```

Como se muestran en los ejemplos anteriores, puedes añadir líneas de `...` a tus ejemplos conteniendo sólo directivas. Puede ser útil cuando un ejemplo es demasiado largo para que una directiva pueda caber cómodamente en la misma línea:

```
>>> print(list(range(5)) + list(range(10, 20)) + list(ra
... # doctest: +ELLIPSIS
[0, ..., 4, 10, ..., 19, 30, ..., 39]
```

Tenga en cuenta que ya que todas las opciones están deshabilitadas por defecto, y las directivas sólo aplican a los ejemplos en los que aparecen, habilitarlas (a través de `+` en la directiva) usualmente es la única opción significativa. Sin embargo, las banderas de opciones también pueden ser pasadas a funciones que ejecutan doctests, estableciendo valores por defecto diferentes. En tales casos, deshabilitar una opción a través de `-` en una directiva puede ser útil.

Advertencias

`doctest` is serious about requiring exact matches in expected output. If even a single character doesn’t match, the test fails. This will probably surprise you a few times, as you learn exactly what Python does and doesn’t guarantee about output. For example, when printing a set, Python doesn’t guarantee that the element is printed in any particular order, so a test like

```
>>> foo()
{"spam", "eggs"}
```

¡es vulnerable! Una solución puede ser hacer:

```
>>> foo() == {"spam", "eggs"}
True
```

en su lugar. Otra es hacer

```
>>> d = sorted(foo())
>>> d
['eggs', 'spam']
```

Existen otros casos, pero ya captas la idea.

Otra mala idea es imprimir elementos que tienen la dirección de un objeto, como por ejemplo

```
>>> id(1.0)  # certain to fail some of the time
7948648
>>> class C: pass
>>> C()  # the default repr() for instances embeds an ac
<C object at 0x00AC18F0>
```

Con la directiva **`ELLIPSIS`** podemos sacar ventaja del último ejemplo:

```
>>> C()  # doctest: +ELLIPSIS
<C object at 0x...>
```

Floating-point numbers are also subject to small output variations across platforms, because Python defers to the platform C library for some floating-point calculations, and C libraries vary widely in quality here.

```
>>> 1000**0.1  # risky
1.9952623149688797
>>> round(1000**0.1, 9) # safer
1.995262315
>>> print(f'{1000**0.1:.4f}') # much safer
1.9953
```

Números de la forma `I/2.**J` son seguros a lo largo de todas las plataformas, y yo frecuentemente planeo ejemplos de doctest para producir números de esa forma:

```
>>> 3./4  # utterly safe
0.75
```

Las facciones simples también son más fáciles de entender para las personas, y eso conduce a una mejor documentación.

API básica

Las funciones **`testmod()`** y **`testfile()`** proporcionan una interfaz simple para doctest que debe ser suficiente para la mayoría de los usos básicos. Para una introducción menos formal a estas funciones, véase las secciones Uso simple: verificar ejemplos en docstrings y Uso Simple: Verificar ejemplos en un Archivo de Texto.

doctest.testfile(*filename*, *module_relative*=*True*, *name*=*None*, *package*=*None*, *globs*=*None*, *verbose*=*None*, *report*=*True*, *optionflags*=*0*, *extraglobs*=*None*, *raise_on_error*=*False*, *parser*=*DocTestParser()*, *encoding*=*None*)

> Todos los argumentos excepto *filename* son opcionales, y deben ser especificados en forma de palabras claves.
>
> Prueba los ejemplos en el archivo con nombre *filename*. Retorna `(failure_count, test_count)`.
>
> El argumento opcional *module_relative* especifica cómo el nombre de archivo debe ser interpretado:
>
> - Si *module_relative* es `True` (el valor por defecto), entonces *filename* especifica una ruta relativa al módulo que es independiente del SO. Por defecto, esta ruta es relativa al directorio del módulo que lo invoca; pero si el argumento *package* es especificado, entonces es relativo a ese paquete. Para asegurar la independencia del SO, *filename* debe usar caracteres `/` para separar segmentos, y no puede ser una ruta absoluta (por ejemplo., no puede empezar con `/`).
> - Si *module_relative* es `False`, entonces *filename* especifica una ruta especifica del SO. La ruta puede ser absoluta o relativa; las rutas relativas son resueltas con respecto al directorio de trabajo actual.

El argumento opcional *name* proporciona el nombre de la prueba; por defecto, o si es `None`, se usa `os.path.basename(filename)`.

El argumento opcional *package* es un paquete de Python o el nombre de una paquete de Python cuyo directorio debe ser usado como el directorio base para un nombre de archivo relativo al módulo. Si no se especifica ningún paquete, entonces el directorio del módulo que invoca se usa como el directorio base para los nombres de archivos relativos al módulo. Es un error especificar *package* si *module_relative* es `False`.

El argumento opcional *globs* proporciona un diccionario a ser usado como los globales cuando se ejecuten los ejemplos. Se crea una nueva copia superficial de este diccionario para el doctest, por lo que sus ejemplos empiezan con una pizarra en blanco. Por defecto, o si es `None`, se usa un nuevo diccionario vacío.

El argumento opcional *extraglobs* proporciona un diccionario mezclado con los globales usados para ejecutar ejemplos. Funciona como **`dict.update()`**: si *globs* y *extraglobs* tienen una clave en común, el valor asociado en *extraglobs* aparece en el diccionario combinado. Por defecto, o si es `None`, no se usa ninguna variable global. Es una característica avanzada que permite la parametrización de doctests. Por ejemplo, un doctest puede ser escribo para una clase base, usando un nombre genérico para la clase, y luego reusado para probar cualquier número de clases heredadas al pasar un diccionario de *extraglobs* mapeando el nombre genérico a la clase heredada para ser probada.

Optional argument *verbose* prints lots of stuff if true, and prints only failures if false; by default, or if `None`, it’s true if and only if `'-v'` is in **`sys.argv`**.

El argumento opcional *report* imprime un resumen al final cuando es verdadero; si no, no imprime nada al final. En modo verboso (*verbose*), el resumen es detallado; si no, el resumen es muy corto (de hecho, vacío si todos las pruebas pasan).

Optional argument *optionflags* (default value `0`) takes the bitwise OR of option flags. See section Banderas de Opción.

El argumento opcional *raise_on_error* tiene como valor por defecto *false*. Si es *true*, se lanza una excepción sobre la primera falla o excepción no esperada en un ejemplo. Esto permite que los fallos sean depurados en un análisis a posteriori. El comportamiento por defecto es continuar corriendo los ejemplos.

El argumento opcional *parser* especifica un **`DocTestParser`** (o subclase) que debe ser usado para extraer las pruebas de los archivos. Su valor por defecto es un analizador sintáctico normal (i.e., `DocTestParser()`).

El argumento opcional *encoding* especifica una codificación que debe ser usada para convertir el archivo a *unicode*.

doctest.testmod(*m=None, name=None, globs=None, verbose=None, report=True, optionflags=0, extraglobs=None, raise_on_error=False, exclude_empty=False*)

Todos los argumentos son opcionales, y todos excepto por *m* deben ser especificados en forma de palabras claves.

Prueba los ejemplos en los docstring de las funciones y clases alcanzables desde el módulo *m* (o desde el módulo **`__main__`** si *m* no es proporcionado o es `None`), empezando con `m.__doc__`.

Also test examples reachable from dict `m.__test__`, if it exists. `m.__test__` maps names (strings) to functions, classes and strings; function and class docstrings are searched for examples; strings are searched directly, as if they were docstrings.

Sólo se buscan los docstrings anexados a los objetos pertenecientes al módulo *m*.

Retorna `(failure_count, test_count)`.

El argumento opcional *name* proporciona el nombre del módulo; por defecto, o si es `None`, se usa `m.__name__`.

Optional argument *exclude_empty* defaults to false. If true, objects for which no doctests are found are excluded from consideration. The default is a backward compatibility hack, so that code still using **`doctest.master.summarize`** in conjunction with **`testmod()`** continues to get output for objects with no tests. The *exclude_empty* argument to the newer **`DocTestFinder`** constructor defaults to true.

Los argumentos opcionales *extraglobs*, *verbose*, *report*, *optionflags*, *raise_on_error*, y *globs* son los mismos en cuanto a la función **`testfile()`** arriba, excepto que *globs* es por defecto `m.__dict__`.

doctest.run_docstring_examples(*f*, *globs*, *verbose=False*, *name='NoName'*, *compileflags=None*, *optionflags=0*)

Prueba los ejemplos asociados con el objeto *f*; por ejemplo, *f* puede ser una cadena de caracteres, un módulo, una función, o un objeto clase.

Una copia superficial del diccionario del argumento *globs* se usa para la ejecución del contexto.

Se usa el argumento opcional *name* en mensajes de fallos, y por defecto es `"NoName"`.

Si el argumento opcional *verbose* es verdadero, la salida se genera incluso si no hay fallas. Por defecto, la salida se genera sólo en caso de la falla de un ejemplo.

El argumento opcional *compileflags* proporciona el conjunto de banderas que se deben usar por el compilador de Python cuando se corran los ejemplos. Por defecto, o si es `None`, las banderas se deducen correspondiendo al conjunto de características futuras encontradas en *globs*.

El argumento opcional *optionflags* trabaja con respecto a la función **`testfile()`** de arriba.

API de unittest

As your collection of doctest'ed modules grows, you'll want a way to run all their doctests systematically. **`doctest`** provides two functions that can be used to create **`unittest`** test suites from modules and text files containing doctests. To integrate with **`unittest`** test discovery, include a load_tests function in your test module:

```
import unittest
import doctest
import my_module_with_doctests

def load_tests(loader, tests, ignore):
    tests.addTests(doctest.DocTestSuite(my_module_with_c
    return tests
```

Hay dos funciones principales para crear instancias de **`unittest.TestSuite`** desde los archivos de texto y módulos con doctests:

doctest.DocFileSuite(**paths, module_relative=True, package=None, setUp=None, tearDown=None, globs=None, optionflags=0, parser=DocTestParser(), encoding=None*)

Convierte las pruebas de doctest de uno o más archivos de texto a una **`unittest.TestSuite`**.

The returned **`unittest.TestSuite`** is to be run by the unittest framework and runs the interactive examples in each file. If an example in any file fails, then the synthesized unit test fails, and a **`failureException`** exception is raised showing the name of the file containing the test and a (sometimes approximate) line number. If all the examples in a file are skipped, then the synthesized unit test is also marked as skipped.

Pasa una o más rutas (como cadenas de caracteres) a archivos de texto para ser examinados.

Se pueden proporcionar las opciones como argumentos por palabra clave:

El argumento opcional *module_relative* especifica cómo los nombres de archivos en *paths* se deben interpretar:

- Si *module_relative* is `True` (el valor por defecto), entonces cada archivo de nombre en *paths* especifica una ruta relativa al módulo independiente del SO. Por defecto, esta ruta es relativa al directorio del módulo lo está invocando; pero si se especifica el argumento *package*, entonces es relativo a ese paquete. Para asegurar la independencia del SO, cada nombre de archivo debe usar caracteres `/` para separar los segmentos de rutas, y puede no ser una ruta absoluta (i.e., puede no empezar con `/`).
- Si *module_relative* es `False`, entonces cada archivo de nombre en *paths* especifica una ruta especifica al SO. La ruta puede ser absoluta o relativa; las rutas relativas son resueltas con respecto a directorio de trabajo actual.

El argumento opcional *package* es un paquete de Python o el nombre de un paquete de Python cuyo directorio debe ser usado como el directorio base para los nombres de archivos relativos al módulo en *paths*. Si no se especifica ningún paquete, entonces el directorio del módulo que lo está invocando se usa como el directorio base para los archivos de nombres relativos al módulo. Es un error especificar *package* si *module_relative* es `False`.

Optional argument *setUp* specifies a set-up function for the test suite. This is called before running the tests in each file. The *setUp* function will be passed a **`DocTest`** object. The *setUp* function can access the test globals as the **`globs`** attribute of the test passed.

Optional argument *tearDown* specifies a tear-down function for the test suite. This is called after running the tests in each file. The *tearDown* function will be passed a **`DocTest`** object. The *tearDown* function can access the test globals as the

`globs` attribute of the test passed.

El argumento opcional *globs* es un diccionario que contiene las variables globales iniciales para las pruebas. Se crea una nueva copia de este diccionario para cada prueba. Por defecto, *globs* es un nuevo diccionario vacío.

El argumento opcional *optionflags* especifica las opciones de doctest por defecto para las pruebas, creado al juntar lógicamente las opciones de bandera individuales. Véase la sección Banderas de Opción. Véase la función **`set_unittest_reportflags()`** abajo para una mejor manera de definir las opciones de informe.

El argumento opcional *parser* especifica un **`DocTestParser`** (o subclase) que debe ser usado para extraer las pruebas de los archivos. Su valor por defecto es un analizador sintáctico normal (i.e., `DocTestParser()`).

El argumento opcional *encoding* especifica una codificación que debe ser usada para convertir el archivo a *unicode*.

Se añade el global `__file__` a los globales proporcionados a los doctests cargados desde un archivo de texto usando **`DocFileSuite()`**.

doctest.DocTestSuite(*module=None*, *globs=None*, *extraglobs=None*, *test_finder=None*, *setUp=None*, *tearDown=None*, *optionflags=0*, *checker=None*)

Convierte las pruebas de doctest para un módulo a un **`unittest.TestSuite`**.

The returned **`unittest.TestSuite`** is to be run by the unittest framework and runs each doctest in the module. Each docstring is run as a separate unit test. If any of the doctests fail, then the synthesized unit test fails, and a **`unittest.TestCase.failureException`** exception is raised showing the name of the file containing the test and a (sometimes approximate) line number. If all the examples in a docstring are skipped, then the

El argumento opcional *module* proporciona el módulo a probar. Puede ser un objeto de módulo o un nombre (posiblemente punteado) de módulo. Si no se especifica, se usa el módulo que invoca esta función.

Optional argument *globs* is a dictionary containing the initial global variables for the tests. A new copy of this dictionary is created for each test. By default, *globs* is the module's `__dict__`.

El argumento opcional *extraglobs* especifica un conjunto de variables globales adicionales que son mezcladas con *globs*. Por defecto, no se usa ningún global adicional.

El argumento opcional *test_finder* es el objeto `DocTestFinder` (o un reemplazo directo) que se usa para extraer doctests desde el módulo.

Optional arguments *setUp*, *tearDown*, and *optionflags* are the same as for function `DocFileSuite()` above, but they are called for each docstring.

Esta función usa la misma técnica de búsqueda que `testmod()`.

Distinto en la versión 3.5: `DocTestSuite()` retorna un `unittest.TestSuite` vacío si *module* no contiene ningún docstring en vez de lanzar un `ValueError`.

Under the covers, `DocTestSuite()` creates a `unittest.TestSuite` out of `doctest.DocTestCase` instances, and `DocTestCase` is a subclass of `unittest.TestCase`. `DocTestCase` isn't documented here (it's an internal detail), but studying its code can answer questions about the exact details of `unittest` integration.

Similarly, `DocFileSuite()` creates a `unittest.TestSuite` out of `doctest.DocFileCase` instances, and `DocFileCase` is a subclass of `DocTestCase`.

So both ways of creating a `unittest.TestSuite` run instances of `DocTestCase`. This is important for a subtle reason: when you run `doctest` functions yourself, you can control the `doctest` options in use directly, by passing option flags to `doctest` functions. However, if you're writing a `unittest` framework, `unittest` ultimately controls when and how tests get run. The framework author typically wants to control `doctest` reporting options (perhaps, e.g., specified by command line options), but there's no way to pass options through `unittest` to `doctest` test runners.

For this reason, `doctest` also supports a notion of `doctest` reporting flags specific to `unittest` support, via this function:

doctest.set_unittest_reportflags(*flags*)

> Set the `doctest` reporting flags to use.
>
> El argumento *flags* toma la combinación por el operador OR de las banderas de opciones. Véase la sección Banderas de Opción. Sólo se pueden usar las «banderas de informe».
>
> This is a module-global setting, and affects all future doctests run by module `unittest`: the `runTest()` method of `DocTestCase` looks at the option flags specified for the test case when the `DocTestCase` instance was constructed. If no reporting flags were specified (which is the typical and expected case), `doctest`'s `unittest` reporting flags are bitwise ORed into the option flags, and the option flags so augmented are passed to the `DocTestRunner` instance created to run the doctest. If any reporting flags were specified when the `DocTestCase` instance was constructed, `doctest`'s `unittest` reporting flags are ignored.
>
> La función retorna el valor de las banderas de informe de `unittest` en efecto antes de que la función fuera invocada.

API avanzada

La API básica es un simple envoltorio que sirve para hacer los doctest fáciles de usar. Es bastante flexible, y debe cumplir las necesidades de la mayoría de los usuarios; si requieres un control más preciso en las pruebas, o deseas extender las capacidades de doctest, entonces debes usar la API avanzada.

La API avanzada gira en torno a dos clases contenedoras, que se usan para guardar los ejemplos interactivos extraídos de los casos doctest:

- **`Example`**: Un statement de Python, emparejado con su salida esperada.
- **`DocTest`**: Una colección de clases **`Example`**, típicamente extraídos de un sólo docstring o archivo de texto.

Se definen clases de procesamiento adicionales para encontrar, analizar sintácticamente, y ejecutar, y comprobar ejemplos de doctest:

- **`DocTestFinder`**: Encuentra todos los docstrings en un módulo dado, y usa un **`DocTestParser`** para crear un **`DocTest`** de cada docstring que contiene ejemplos interactivos.
- **`DocTestParser`**: Crea un objeto **`DocTest`** de una cadena de caracteres (tal como un docstring de un objeto).
- **`DocTestRunner`**: Ejecuta los ejemplos en un **`DocTest`**, y usa un **`OutputChecker`** para verificar su salida.
- **`OutputChecker`**: Compara la salida real de un ejemplo de doctest con la salida esperada, y decide si coinciden.

Las relaciones entre estas clases de procesamiento se resumen en el siguiente diagrama:

```
                            list of:
+------+                   +---------+
|module| --DocTestFinder-> | DocTest | --DocTestRunner->
+------+    |        ^     +---------+     |       ^
            |        |     | Example |     |       |
            v        |     |   ...   |     v       |
           DocTestParser   | Example |   OutputChecker
                           +---------+
```

Objetos DocTest

class doctest.DocTest(*examples*, *globs*, *name*, *filename*, *lineno*, *docstring*)

Una colección de ejemplos de doctest que deben ejecutarse en un sólo nombre de espacios. Se usan los argumentos del constructor para inicializar los atributos de los mismos nombres.

`DocTest` define los siguientes atributos. Son inicializados por el constructor, y no deben ser modificados directamente.

examples

Una lista de objetos `Example` codificando los ejemplos interactivos de Python individuales que esta prueba debe ejecutar.

globs

El nombre de espacios (alias *globals*) en que los ejemplos se deben ejecutar. Este es un diccionario que mapea nombres a valores. Cualquier cambio al nombre de espacios hecho por los ejemplos (tal como juntar nuevas variables) se reflejará en `globs` después de que se ejecute la prueba.

name

Un nombre de cadena de caracteres que identifica el `DocTest`. Normalmente, este es el nombre del objeto o archivo del que se extrajo la prueba.

filename

The name of the file that this `DocTest` was extracted from; or `None` if the filename is unknown, or if the `DocTest` was not extracted from a file.

lineno

El número de línea dentro de `filename` donde este `DocTest` comienza, o `None` si el número de línea no está disponible. Este número de línea es comienza en 0 con respecto al comienzo del archivo.

docstring

La cadena de caracteres del que se extrajo la cadena, o `None` si la cadena no está disponible, o si la prueba no se extrajo de una cadena de caracteres.

Objetos *Example*

class doctest.Example(*source*, *want*, *exc_msg=None*, *lineno=0*, *indent=0*, *options=None*)

Un sólo ejemplo interactivo, que consta de una sentencia de Python y su salida esperada. Los argumentos del constructor se usan para inicializar los atributos del mismo nombre.

La clase **`Example`** define los siguientes atributos. Son inicializados por el constructor, y no deben ser modificados directamente.

source

Una cadena de caracteres que contiene el código fuente del ejemplo. Este código fuente consiste de una sola sentencia Python, y siempre termina en una nueva línea; el constructor añade una nueva línea cuando sea necesario.

want

La salida esperada de ejecutar el código fuente del ejemplo (o desde la salida estándar, o un seguimiento en caso de una excepción). **`wants`** termina con una nueva línea a menos que no se espera ninguna salida, en cuyo caso es una cadena vacía. El constructor añade una nueva línea cuando sea necesario.

exc_msg

El mensaje de excepción que el ejemplo genera, si se espera que el ejemplo genere una excepción; o `None` si no se espera que genere una excepción. Se compara este mensaje de excepción con el valor de retorno de **`traceback.format_exception_only()`**. **`exc_msg`** termina con una nueva línea a menos que sea `None`. El constructor añade una nueva línea si se necesita.

lineno

El número de línea dentro de la cadena de caracteres que contiene este ejemplo donde el ejemplo comienza. Este número de línea comienza en 0 con respecto al comienzo de la cadena que lo contiene.

indent

La sangría del ejemplo en la cadena que lo contiene; i.e., el número de caracteres de espacio que preceden la primera entrada del ejemplo.

options

A dictionary mapping from option flags to `True` or `False`, which is used to override default options for this example. Any option flags not contained in this dictionary are left at their default value (as specified by the **`DocTestRunner`**'s optionflags). By default, no options are set.

Objetos *DocTestFinder*

class doctest.DocTestFinder(*verbose=False, parser=DocTestParser(), recurse=True, exclude_empty=True*)

Una clase de procesamiento que se usa para extraer los **`DocTest`** que son relevantes para un objeto dado, desde su docstring y los docstring de sus objetos contenidos. Se puede extraer los **`DocTest`** de los módulos, clases, funciones, métodos, métodos estáticos, métodos de clase, y propiedades.

Se puede usar el argumento opcional *verbose* para mostrar los objetos buscados por *finder*. Su valor por defecto es `False` (ninguna salida).

El argumento opcional *parser* especifica el objeto **`DocTestParser`** (o un reemplazo directo) que se usa para extraer doctests desde docstrings.

Si el argumento opcional *recurse* es falso, entonces el método `DocTestFinder.find()` sólo examinará el objeto dado, y no cualquier objeto contenido.

Si el argumento opcional *exclude_empty* es falso, entonces `DocTestFinder.find()` incluirá pruebas para objetos con docstrings vacíos.

`DocTestFinder` define los siguientes métodos:

find(*obj[, name][, module][, globs][, extraglobs]*)

Retorna una lista de los **`Doctest`** que se definen por el docstring de *obj*, o por cualquiera de los docstring de sus objetos contenidos.

El argumento opcional *name* especifica el nombre del objeto; este nombre será usado para construir los nombres de los `DocTest` retornados. Si *name* no se especifica, entonces se usa `obj.__name__`.

El parámetro opcional *module* es el módulo que contiene el objeto dado. Si no se especifica el módulo o si es `None`, entonces el buscador de pruebas tratará de determinar automáticamente el módulo correcto. Se usa el módulo del objeto:

- Como un espacio de nombres por defecto, si no se especifica *globs*.
- Para evitar que *DocTestFinder* extraiga DocTests desde objetos que se importan desde otros módulos. (Se ignoran objetos contenidos con módulos aparte de *module*.)
- Para encontrar el nombre del archivo conteniendo el objeto.
- Para ayudar a encontrar el número de línea del objeto dentro de su archivo.

Si *module* es falso, no se hará ningún intento de encontrar el módulo. Es poco claro, de uso mayormente para probar doctest en si mismo: si *module* es `False`, o es `None` pero no se puede encontrar automáticamente, entonces todos los objetos se consideran que pertenecen al módulo (inexistente), por lo que todos los objetos contenidos se buscarán (recursivamente) por doctests.

The globals for each **`DocTest`** is formed by combining *globs* and *extraglobs* (bindings in *extraglobs* override bindings in *globs*). A new shallow copy of the globals dictionary is created for each **`DocTest`**. If *globs* is not specified, then it defaults to the module's **`__dict__`**, if specified, or `{}` otherwise. If *extraglobs* is not specified, then it defaults to `{}`.

Objetos *DocTestParser*

class doctest.DocTestParser

Un clase de procesamiento usada para extraer ejemplos interactivos de una cadena de caracteres, y usarlos para crear un objeto **`DocTest`**.

`DocTestParser` define los siguientes métodos:

get_doctest(*string*, *globs*, *name*, *filename*, *lineno*)

Extrae todos los ejemplos de *doctest* de una cadena dada, y los recolecta en un objeto **`DocTest`**.

globs, *name*, *filename*, and *lineno* are attributes for the new **`DocTest`** object. See the documentation for **`DocTest`** for more information.

get_examples(*string*, *name='<string>'*)

Extrae todos los ejemplos de la cadena de caracteres dada, y los retorna como una lista de objetos **`Example`**. Los números de línea empiezan en 0. El argumento opcional *name* es una nombre identificando esta cadena, y sólo es usada para mensajes de errores.

parse(*string*, *name='<string>'*)

Divide the given string into examples and intervening text, and return them as a list of alternating `Example`s and strings. Line numbers for the `Example`s are 0-based. The optional argument *name* is a name identifying this string, and is only used for error messages.

TestResults objects

class doctest.TestResults(*failed*, *attempted*)

failed

Number of failed tests.

attempted

Number of attempted tests.

skipped

Number of skipped tests.

Added in version 3.13.

Objetos *DocTestRunner*

class doctest.DocTestRunner(*checker=None*, *verbose=None*, *optionflags=0*)

Una clase de procesamiento usada para ejecutar y verificar los ejemplos interactivos en un `DocTest`.

The comparison between expected outputs and actual outputs is done by an `OutputChecker`. This comparison may be customized with a number of option flags; see section Banderas de Opción for more information. If the option flags are insufficient, then the comparison may also be customized by passing a subclass of `OutputChecker` to the constructor.

The test runner's display output can be controlled in two ways. First, an output function can be passed to `run()`; this

function will be called with strings that should be displayed. It defaults to `sys.stdout.write`. If capturing the output is not sufficient, then the display output can be also customized by subclassing DocTestRunner, and overriding the methods **`report_start()`**, **`report_success()`**, **`report_unexpected_exception()`**, and **`report_failure()`**.

El argumento por palabra clave opcional *checker* especifica el objeto **`OutputChecker`** (o un reemplazo directo) que se debe usar para comparar las salidas esperadas con las salidas reales de los ejemplos de doctest.

El argumento por palabra clave opcional *verbose* controla la verbosidad de **`DocTestRunner`**. Si *verbose* es `True`, entonces la información de cada ejemplo se imprime , mientras se ejecuta. Si *verbose* es `False`, entonces sólo las fallas se imprimen. Si *verbose* no se especifica, o es `None`, entonces la salida verbosa se usa si y sólo se usa el modificador de la línea de comandos `-v`.

El argumento por palabra clave opcional *optionflags* se puede usar para controlar cómo el *test runner* compara la salida esperada con una salida real, y cómo muestra las fallas. Para más información, véase la sección Banderas de Opción.

The test runner accumulates statistics. The aggregated number of attempted, failed and skipped examples is also available via the **`tries`**, **`failures`** and **`skips`** attributes. The **`run()`** and **`summarize()`** methods return a **`TestResults`** instance.

`DocTestRunner` defines the following methods:

report_start(*out*, *test*, *example*)

> Notifica que el *test runner* está a punto de procesar el ejemplo dado. Este método es proporcionado para permitir que clases heredadas de **`DocTestRunner`** personalicen su salida; no debe ser invocado directamente.

example is the example about to be processed. *test* is the test containing *example*. *out* is the output function that was passed to `DocTestRunner.run()`.

report_success(*out*, *test*, *example*, *got*)

Notifica que el ejemplo dado se ejecutó correctamente. Este método es proporcionado para permitir que las clases heredadas de `DocTestRunner` personalicen su salida; no debe ser invocado directamente.

example es el ejemplo a punto de ser procesado. *got* es la salida real del ejemplo. *test* es la prueba conteniendo *example*. *out* es la función de salida que se pasa a `DocTestRunner.run()`.

report_failure(*out*, *test*, *example*, *got*)

Notifica que el ejemplo dado falló. Este método es proporcionado para permitir que clases heredadas de `DocTestRunner` personalicen su salida; no debe ser invocado directamente.

example es el ejemplo a punto de ser procesado. *got* es la salida real del ejemplo. *test* es la prueba conteniendo *example*. *out* es la función de salida que se pasa a `DocTestRunner.run()`.

report_unexpected_exception(*out*, *test*, *example*, *exc_info*)

Notifica que el ejemplo dado lanzó una excepción inesperada. Este método es proporcionado para permitir que las clases heredadas de `DocTestRunner` personalicen su salida; no debe ser invocado directamente.

example es el ejemplo a punto de ser procesado, *exc_info* es una tupla que contiene información sobre la excepción inesperada (como se retorna por `sys.exc_info()`). *test* es la prueba conteniendo *example*. *out* es la función de salida que debe ser pasada a `DocTestRunner.run()`.

run(*test*, *compileflags=None*, *out=None*, *clear_globs=True*)

Run the examples in *test* (a `DocTest` object), and display the results using the writer function *out*. Return a `TestResults` instance.

Los ejemplo se ejecutan en el espacio de nombres `test.globs`. Si *clear_globs* es verdadero (el valor por defecto), entonces este espacio de nombres será limpiado después de la prueba se ejecute, para ayudar con la colección de basura. Si quisieras examinar el espacio de nombres después de que la prueba se complete, entonces use *clear_globs=False*.

compileflags da el conjunto de banderas que se deben usar por el compilador de Python cuando se ejecutan los ejemplos. Si no se especifica, entonces su valor por defecto será el conjunto de banderas de *future-import* que aplican a *globs*.

The output of each example is checked using the **`DocTestRunner`**'s output checker, and the results are formatted by the **`DocTestRunner.report_*()`** methods.

summarize(*verbose=None*)

Print a summary of all the test cases that have been run by this DocTestRunner, and return a `TestResults` instance.

El argumento opcional *verbose* controla qué tan detallado es el resumen. Si no se especifica la verbosidad, entonces se usa la verbosidad de `DocTestRunner`.

`DocTestParser` has the following attributes:

tries

Number of attempted examples.

failures

Number of failed examples.

skips

Number of skipped examples.

Added in version 3.13.

Objetos *OutputChecker*

class doctest.OutputChecker

Una clase que se usa para verificar si la salida real de un ejemplo de doctest coincide con la salida esperada. **`OutputChecker`** define dos métodos: **`check_output()`**, que compara un par de salidas dadas, y retorna `True` si coinciden; y **`output_difference()`**, que retorna una cadena que describe las diferencias entre las dos salidas.

`OutputChecker` define los siguientes métodos:

check_output(*want*, *got*, *optionflags*)

Retorna `True` si y sólo si la salida real de un ejemplo (*got*) coincide con la salida esperada (*want*). Siempre se considera que estas cadenas coinciden si son idénticas; pero dependiendo de qué banderas de opción el *test runner* esté usando, varias coincidencias inexactas son posibles. Véase la sección Banderas de Opción para más información sobre las banderas de opción.

output_difference(*example*, *got*, *optionflags*)

Retorna una cadena que describe las diferencias entre la salida esperada para un ejemplo dado (*example*) y la salida real (*got*). *optionflags* es el conjunto de banderas de opción usado para comparar *want* y *got*.

Depuración

Doctest proporciona varios mecanismos para depurar los ejemplos de doctest:

- Varias funciones convierten los doctest en programas de Python ejecutables, que pueden ser ejecutadas bajo el depurador de Python, **`pdb`**.
- La clase **`DebugRunner`** es una subclase de **`DocTestRunner`** que lanza una excepción por el primer ejemplo fallido, conteniendo información sobre ese ejemplo. Esta información se puede usar para realizar depuración a posteriori en el ejemplo.
- Los casos de **`unittest`** generados por **`DocTestSuite()`** admiten el método **`debug()`** definido por **`unittest.TestCase`**.
- Puedes añadir una llamada a **`pdb.set_trace()`** en un ejemplo de doctest, y bajarás al depurador de Python cuando esa línea sea ejecutada. Entonces puedes inspeccionar los valores de las variables, y demás. Por ejemplo, supongamos que `a.py` contiene sólo este docstring de módulo:

  ```
  """
  >>> def f(x):
  ...     g(x*2)
  >>> def g(x):
  ...     print(x+3)
  ...     import pdb; pdb.set_trace()
  >>> f(3)
  9
  """
  ```

  Entonces una sesión interactiva puede lucir como esta:

  ```
  >>> import a, doctest
  >>> doctest.testmod(a)
  --Return--
  > <doctest a[1]>(3)g()->None
  -> import pdb; pdb.set_trace()
  (Pdb) list
  ```

```
  1     def g(x):
  2         print(x+3)
  3  ->     import pdb; pdb.set_trace()
[EOF]
(Pdb) p x
6
(Pdb) step
--Return--
> <doctest a[0]>(2)f()->None
-> g(x*2)
(Pdb) list
  1     def f(x):
  2  ->     g(x*2)
[EOF]
(Pdb) p x
3
(Pdb) step
--Return--
> <doctest a[2]>(1)?()->None
-> f(3)
(Pdb) cont
(0, 3)
>>>
```

Funciones que convierten los doctest a código de Python, y posiblemente ejecuten el código sintetizado debajo del depurador:

doctest.script_from_examples(*s*)

Convierte texto con ejemplos a un script.

El argumento *s* es una cadena que contiene los ejemplos de doctest. La cadena se convierte a un script de Python, donde los ejemplos de doctest en *s* se convierten en código regular, y todo lo demás se convierte en comentarios de Python. El script generado se retorna como una cadena, Por ejemplo,

```
import doctest
print(doctest.script_from_examples(r"""
    Set x and y to 1 and 2.
    >>> x, y = 1, 2

    Print their sum:
    >>> print(x+y)
    3
"""))
```

muestra:

```
# Set x and y to 1 and 2.
x, y = 1, 2
#
# Print their sum:
print(x+y)
# Expected:
## 3
```

Esta función se usa internamente por otras funciones (véase más abajo), pero también pueden ser útiles cuando quieres transformar una sesión de Python interactiva en un script de Python.

doctest.testsource(*module*, *name*)

Convierte el doctest para un objeto en un script.

Argument *module* is a module object, or dotted name of a module, containing the object whose doctests are of interest. Argument *name* is the name (within the module) of the object with the doctests of interest. The result is a string, containing the object's docstring converted to a Python script, as described for **`script_from_examples()`** above. For example, if module `a.py` contains a top-level function **`f()`**, then

```
import a, doctest
print(doctest.testsource(a, "a.f"))
```

prints a script version of function **`f()`**"s docstring, with doctests converted to code, and the rest placed in comments.

doctest.debug(*module*, *name*, *pm=False*)

Depura los doctest para un objeto.

Los argumentos *module* y *name* son los mismos que para la función `testsource()` arriba. El script de Python sintetizado para el docstring del objeto nombrado es escrito en un archivo temporal, y entonces ese archivo es ejecutado bajo el control del depurador de PYthon, `pdb`.

Se usa una copia superficial de `module.__dict__` para el contexto de ejecución local y global.

El argumento opcional *pm* controla si se usa la depuración post-mortem. Si *pm* tiene un valor verdadero, el archivo de script es ejecutado directamente, y el depurador está involucrado sólo si el script termina a través del lanzamiento de una excepción. Si lo hace, entonces la depuración post-mortem es invocada, a través de `pdb.post_mortem()`, pasando el objeto de rastreo desde la excepción sin tratar. Si no se especifica *pm*, o si es falso, el script se ejecuta bajo el depurador desde el inicio, a través de una llamada de `exec()` apropiada a `pdb.run()`.

doctest.debug_src(*src*, *pm=False*, *globs=None*)

Depura los doctest en una cadena de caracteres.

Es como la función function `debug()` arriba, excepto que una cadena de caracteres que contiene los ejemplos de doctest se especifica directamente, a través del argumento *src*.

El argumento opcional *pm* tiene el mismo significado como en la función `debug()` arriba.

El argumento opcional *globs* proporciona un diccionario a usar como contexto de ejecución local y global. Si no se especifica, o es `None`, se usa un diccionario vacío. Si se especifica, se usa una copia superficial del diccionario.

La clase `DebugRunner`, y las excepciones especiales que puede lanzar, son de más interés a los autores de frameworks de pruebas, y sólo serán descritos brevemente aquí. Véase el código fuente, y especialmente el docstring de `DebugRunner` (¡que es un doctest!) para más detalles:

class doctest.DebugRunner(*checker=None*, *verbose=None*, *optionflags=0*)

> Una subclase de `DocTestRunner` que lanza una excepción tan pronto como se encuentra una falla. Si ocurre una excepción inesperada, se lanza una excepción `UnexpectedException`, conteniendo la prueba, el ejemplo, y la excepción original. Si la salida no coincide, entonces se lanza una excepción `DocTestFailure`, conteniendo la prueba, el ejemplo, y la salida real.
>
> Para información sobre los parámetros de construcción y los métodos, véase la documentación para `DocTestRunner` en la sección API avanzada.

Hay dos excepciones que se pueden lanzar por instancias de `DebugRunner`:

exception doctest.DocTestFailure(*test*, *example*, *got*)

> Una excepción lanzada por `DocTestRunner` para señalar que la salida real del ejemplo de un doctest no coincidió con su salida esperada. Los argumentos del constructor se usan para inicializar los atributos del mismo nombre.

`DocTestFailure` define los siguientes atributos:

DocTestFailure.test

> El objeto `DocTest` que estaba siendo ejecutado cuando el ejemplo falló.

DocTestFailure.example

> El objeto `Example` que falló.

DocTestFailure.got

> La salida real del ejemplo.

exception doctest.UnexpectedException(*test*, *example*, *exc_info*)

Una excepción lanzada por `DocTestRunner` para señalar que un ejemplo de doctest lanzó una excepción inesperada. Los argumentos del constructor se usan para inicializar los atributos del mismo nombre.

`UnexpectedException` define los siguientes atributos:

UnexpectedException.test

El objeto `DocTest` que estaba siendo ejecutado cuando el ejemplo falló.

UnexpectedException.example

El objeto `Example` que falló.

UnexpectedException.exc_info

Una tupla que contiene información sobre la excepción inesperada, como es retornado por `sys.exc_info()`.

Plataforma improvisada

As mentioned in the introduction, `doctest` has grown to have three primary uses:

1. Verificar los ejemplos en los docstring.
2. Pruebas de regresión.
3. Documentación ejecutable / pruebas literarias.

Estos usos tienen diferentes requerimientos, y es importante distinguirlos. En particular, llenar tus docstring con casos de prueba desconocidos conduce a mala documentación.

When writing a docstring, choose docstring examples with care. There’s an art to this that needs to be learned—it may not be natural at first. Examples should add genuine value to the documentation. A good example can often be worth many words. If done with care, the examples will be invaluable for your users, and will pay back the time it takes to collect them many times over as the years go by and things change. I’m still amazed at how often one of my `doctest` examples stops working after a «harmless» change.

Doctest también es una excelente herramienta para pruebas de regresión, especialmente si no escatimas en texto explicativo. Al intercalar prosa y ejemplos, se hace mucho más fácil mantener el seguimiento de lo que realmente se está probando, y por qué. Cuando una prueba falla, buena prosa puede hacer mucho más fácil comprender cuál es el problema, y cómo debe ser arreglado. Es verdad que puedes escribir comentarios extensos en pruebas basadas en código, pero pocos programadores lo hacen. Quizás es porque simplemente doctest hace escribir pruebas mucho más fácil que escribir código, mientras que escribir comentarios en código es mucho más difícil. Pienso que va más allá de eso: la actitud natural cuando se escribe una prueba basada en doctest es que quieres explicar los puntos finos de tu software, e ilustrarlos con ejemplos. Esto naturalmente lleva a archivos de pruebas que empiezan con las características más simples, y lógicamente progresan a complicaciones y casos extremos. Una narrativa coherente es el resultado, en vez de una colección de funciones aisladas que pruebas trozos aislados de funcionalidad aparentemente al azar. Es una actitud diferente, y produce resultados diferentes, difuminando la distinción entre probar y explicar.

Pruebas de regresión se limitan mejor a objetos o archivos dedicados. Hay varias opciones para organizar pruebas:

- Escribe archivos de texto que contienen los casos de prueba como ejemplos interactivos, y prueba los archivos usando `testfile()` o `DocFileSuite()`. Esto es lo recomendado, aunque es más fácil hacerlo para nuevos proyectos, diseñados desde el comienzo para usar doctest.
- Define funciones nombradas `_regrtest_topic` que consisten en docstrings únicas, que contienen casos de prueba por los tópicos nombrados. Estas funciones se pueden incluir en el mismo archivo que el módulo, o separadas en un archivo de prueba separado.
- Define a `__test__` dictionary mapping from regression test topics to docstrings containing test cases.

Cuando has puesto tus pruebas en un módulo, el módulo puede ser el mismo *test runner*. Cuando una prueba falla, puedes hacer que tu *test runner* vuelva a ejecutar sólo los doctest fallidos mientras que tu depuras el problema. Aquí hay un ejemplo mínimo de tal *test runner*:

```
if __name__ == '__main__':
    import doctest
    flags = doctest.REPORT_NDIFF|doctest.FAIL_FAST
    if len(sys.argv) > 1:
        name = sys.argv[1]
        if name in globals():
            obj = globals()[name]
        else:
            obj = __test__[name]
        doctest.run_docstring_examples(obj, globals(), r
                                       optionflags=flags
    else:
        fail, total = doctest.testmod(optionflags=flags)
        print(f"{fail} failures out of {total} tests")
```

Notas al pie de página

No se admiten los ejemplos que contienen una salida esperada y una excepción. Intentar adivinar dónde una termina y la otra empieza es muy propenso a errores, y da lugar a una prueba confusa.

`unittest` — Unit testing framework

Código fuente: Lib/unittest/__init__.py [https://github.com/python/cpython/tree/3.14/Lib/unittest/__init__.py]

(Si ya estás familiarizado con los conceptos básicos de realización de tests, puedes saltar a la lista de métodos de aserción.)

The `unittest` unit testing framework was originally inspired by JUnit and has a similar flavor as major unit testing frameworks in other languages. It supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework.

To achieve this, `unittest` supports some important concepts in an object-oriented way:

configuración de prueba (*fixture*)

Un *test fixture* representa los preparativos para realizar una o más pruebas y las acciones de limpieza asociadas. Esto puede incluir, por ejemplo, la creación de bases de datos temporales, directorios o el arranque de procesos del servidor.

caso de prueba

A *test case* is the individual unit of testing. It checks for a specific response to a particular set of inputs. `unittest` provides a base class, `TestCase`, which may be used to create new test cases.

conjunto de pruebas

Un *test suite* es una colección de casos de prueba, juegos de prueba o ambos. Se usa para agrupar pruebas que se han de ejecutar juntas.

ejecutor de pruebas

Un *test runner* es un componente que dirige la ejecución de las pruebas y proporciona un resultado. El ejecutor puede disponer de una interfaz gráfica, de texto o devolver un valor especial que indique el resultado de la ejecución de las pruebas.

Ver también

Módulo `doctest`

Otro módulo de soporte a pruebas con una solución muy diferente.

Simple Smalltalk Testing: With Patterns [https://web.archive.org/web/20150315073817/http://www.xprogramming.com/testfram.htm]

Kent Beck’s original paper on testing frameworks using the pattern shared by **`unittest`**.

pytest [https://docs.pytest.org/]

Una infraestructura de pruebas unitarias de otro proveedor con una sintaxis más ligera de escritura de pruebas, por ejemplo: `assert func(10) == 42`.

The Python Testing Tools Taxonomy [https://wiki.python.org/moin/PythonTestingToolsTaxonomy]

Una lista extensa de herramientas de prueba para Python incluyendo infraestructuras de pruebas funcionales y librerías de objetos sucedáneos.

Testing in Python Mailing List [http://lists.idyll.org/listinfo/testing-in-python]

Grupo especializado en debate sobre las pruebas y las herramientas de prueba de Python.

El script `Tools/unittestgui/unittestgui.py` en la distribución de código fuente de Python es una herramienta gráfica para el descubrimiento y ejecución de pruebas. Está orientado principalmente a facilitar su uso para aquellos que son nuevos en las pruebas unitarias. Para entornos de producción, se recomienda que las pruebas sean ejecutadas por un sistema de integración continua como Buildbot [https://buildbot.net/], Jenkins [https://www.jenkins.io/], GitHub Actions [https://github.com/features/actions], o AppVeyor [https://www.appveyor.com/].

Ejemplo sencillo

The **`unittest`** module provides a rich set of tools for constructing and running tests. This section demonstrates that a small subset of the tools suffice to meet the needs of most users.

He aquí un breve script para probar estros tres métodos de cadena:

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

A test case is created by subclassing **`unittest.TestCase`**. The three individual tests are defined with methods whose names start with the letters `test`. This naming convention informs the test runner about which methods represent tests.

El quid de cada test es la llamada a **`assertEqual()`** para verificar un resultado esperado; `assertTrue()` o `assertFalse()` para verificar una condición; o `assertRaises()` para asegurar que se lanza una excepción específica. Se utilizan estos métodos en lugar de la sentencia `assert` para que el ejecutor de pruebas pueda acumular todos los resultados de la prueba de cara a realizar un informe.

Los métodos `setUp()` y `tearDown()` permiten definir instrucciones que han de ser ejecutadas antes y después, respectivamente, de cada método de prueba. Se describen con mas detalle en la sección Organización del código de pruebas.

El bloque final muestra un modo sencillo de ejecutar las pruebas. `unittest.main()` proporciona una interfaz de línea de órdenes para el script de prueba. Cuando se ejecuta desde la línea de órdenes, el script anterior produce una salida como:

```
...
-------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Si se le pasa una opción `-v` al script de prueba, se establecerá un nivel mayor de detalle del proceso de `unittest.main()` y se obtendrá la siguiente salida:

```
test_isupper (__main__.TestStringMethods.test_isupper) .
test_split (__main__.TestStringMethods.test_split) ...
test_upper (__main__.TestStringMethods.test_upper) ...

-------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

The above examples show the most commonly used `unittest` features which are sufficient to meet many everyday testing needs. The remainder of the documentation explores the full feature set from first principles.

Distinto en la versión 3.11: El comportamiento de retornar un valor de un método de prueba (que no sea el valor por defecto `None`), ahora está obsoleto.

Interfaz de línea de comandos

Se puede usar el módulo *unittest* desde la línea de órdenes para ejecutar pruebas de módulos, clases o hasta métodos de prueba individuales:

```
python -m unittest test_module1 test_module2
python -m unittest test_module.TestClass
python -m unittest test_module.TestClass.test_method
```

Se puede pasar una lista con cualquier combinación de nombres de módulo, así como clases o métodos completamente cualificados.

Se puede especificar los módulos de prueba por ruta de fichero también:

```
python -m unittest tests/test_something.py
```

Esto permite usar el completado automático de nombre de fichero de la shell para especificar el módulo de pruebas. El fichero especificado aún debe ser susceptible de importarse como módulo. La ruta se convierte en nombre de módulo eliminando ".py" y convirtiendo el separador de directorios por ".". Si se desea ejecutar un fichero de prueba que no se puede importar como módulo, se ha de ejecutar el fichero directamente.

Se pueden ejecutar las pruebas con más nivel de detalle (mayor verbosidad) pasando el parámetro *-v*:

```
python -m unittest -v test_module
```

Cuando se ejecuta sin argumentos, se inicia Descubrimiento de pruebas:

```
python -m unittest
```

Para obtener una lista de todas las opciones de línea de órdenes:

```
python -m unittest -h
```

Distinto en la versión 3.2: En versiones anteriores sólo era posible ejecutar métodos de prueba individuales, pero no módulos ni clases.

Added in version 3.14: Output is colorized by default and can be controlled using environment variables.

Opciones de la línea de órdenes

unittest da soporte a las siguientes opciones de línea de órdenes:

-b, --buffer

Los flujos de datos de salida estándar y error estándar se acumulan en un búfer durante la ejecución de pruebas. La salida de las pruebas correctas se descarta. La salida de las pruebas que fallan o devuelven un error se añade a los mensajes de fallo.

-c, --catch

La pulsación de `Control-C` durante la ejecución de pruebas espera a que termine la prueba en curso y da un informe de los resultados hasta ese momento. Una segunda pulsación de `Control-C` lanza la excepción **`KeyboardInterrupt`** usual.

Consultar en Gestión de señales las funciones que proporcionan esta funcionalidad.

-f, --failfast

Finaliza la ejecución tras el primer error o fallo.

-k

Ejecutar solamente los métodos y clases de prueba que coincidan con el patrón o subcadena de caracteres. Esta opción se puede usar más de una vez, en cuyo caso se incluirán todos los casos de prueba que coincidan con cualquiera de los patrones dados.

Los patrones que contengan un comodín (`*`) se comprueban contra el nombre de la prueba usando `fnmatch.fnmatchcase()`; si no lo contienen, se usa una comprobación de contenido simple sensible a mayúsculas.

Los patrones se comprueban contra el nombre del método completamente cualificado como lo importa el cargador de pruebas.

Por ejemplo, `-k foo` coincide con `foo_tests.SomeTest.test_something` y con `bar_tests.SomeTest.test_foo`, pero no con `bar_tests.FooTest.test_something`.

--locals
    Muestra las variables locales en las trazas.

--durations N
    Muestra los N casos de prueba más lentos (N=0 para todos).

Added in version 3.2: Se añadieron las opciones de línea de órdenes `-b`, `-c` y `-f`.

Added in version 3.5: La opción de línea de órdenes `--locals`.

Added in version 3.7: La opción de línea de órdenes `-k`.

Added in version 3.12: La opción de línea de órdenes `--durations`.

La línea de órdenes también se puede usar para descubrimiento de pruebas, para ejecutar todas las pruebas de un proyecto o un subconjunto de éstas.

Descubrimiento de pruebas

Added in version 3.2.

Unittest da soporte al descubrimiento de pruebas simples. Para ser compatible con el descubrimiento de pruebas, todos los ficheros de prueba deben ser módulos o paquetes importables desde el directorio superior del proyecto (por lo que sus nombres han de ser identificadores válidos).

El descubrimiento de pruebas está implementado en **`TestLoader.discover()`**, pero también se puede usar desde la línea de órdenes. La línea de órdenes básica es:

```
cd project_directory
python -m unittest discover
```

Nota

Como atajo, `python -m unittest` es el equivalente de `python -m unittest discover`. Si se desea pasar argumentos al descubrimiento de pruebas, hay que pasar la sub-orden `discover` explícitamente.

La sub-orden `discover` cuenta con las siguientes opciones:

-v, --verbose

Salida verbosa

-s, --start-directory directory

Directorio de inicio para el descubrimiento (`.` si se omite)

-p, --pattern pattern

Patrón para la búsqueda de ficheros de prueba (`test*.py` si se omite)

-t, --top-level-directory directory

Directorio superior del proyecto (el directorio inicial si se omite)

Las opciones **`-s`**, **`-p`**, y **`-t`** se pueden pasar por posición en el orden mostrado. Las siguientes líneas de órdenes son equivalentes:

```
python -m unittest discover -s project_directory -p "*_t
python -m unittest discover project_directory "*_test.py
```

Además de pasar una ruta, es posible pasar el nombre de un paquete, por ejemplo `myproject.subpackage.test`, como directorio de inicio. El nombre de paquete proporcionado será importado y su ubicación en el sistema de archivos será utilizada como directorio de inicio.

Prudencia

El descubrimiento de pruebas carga las pruebas importándolas. Una vez que el descubrimiento ha encontrado todos los ficheros de prueba a partir del directorio inicial, especificado, convierte las rutas en nombres de paquetes a importar. Por ejemplo, `foo/bar/baz.py` será importado como `foo.bar.baz`.

Si hay un paquete instalado globalmente y se intenta hacer un descubrimiento sobre una copia diferente a la instalada del paquete, *podría* ocurrir que se importe la versión incorrecta. Si ocurre esto, el descubrimiento lanza una advertencia y abandona.

Si se proporciona el directorio de inicio como nombre de paquete en lugar de ruta a un directorio, el descubrimiento asume que la ubicación importada es la deseada, así que no se da la advertencia descrita.

Los módulos y paquetes de prueba pueden personalizar la carta y descubrimiento de pruebas mediante el protocolo load_tests.

Distinto en la versión 3.4: Test discovery supports namespace packages.

Distinto en la versión 3.11: Test discovery dropped the namespace packages support. It has been broken since Python 3.7. Start directory and its subdirectories containing tests must be regular package that have `__init__.py` file.

If the start directory is the dotted name of the package, the ancestor packages can be namespace packages.

Distinto en la versión 3.14: Test discovery supports namespace package as start directory again. To avoid scanning directories unrelated to Python, tests are not searched in subdirectories that do not contain `__init__.py`.

Organización del código de pruebas

The basic building blocks of unit testing are *test cases* — single scenarios that must be set up and checked for correctness. In **`unittest`**, test cases are represented by **`unittest.TestCase`** instances. To make your own test cases you must write subclasses of **`TestCase`** or use **`FunctionTestCase`**.

El código de pruebas de una instancia de **`TestCase`** debería ser completamente autónomo, de tal modo que se pueda ejecutar aisladamente o en una combinación arbitraria con otras clases de prueba.

La subclase más simple de **`TestCase`** se limita a implementar un método test (o un método que empiece por `test`) para realizar código de prueba específico:

```
import unittest

class DefaultWidgetSizeTestCase(unittest.TestCase):
    def test_default_widget_size(self):
        widget = Widget('The widget')
        self.assertEqual(widget.size(), (50, 50))
```

Note that in order to test something, we use one of the assert* methods provided by the **`TestCase`** base class. If the test fails, an exception will be raised with an explanatory message, and **`unittest`** will identify the test case as a *failure*. Any other exceptions will be treated as *errors*.

Las pruebas pueden ser muchas y su preparación puede ser repetitiva. Por suerte, se puede «sacar factor común» a la preparación implementando un método **`setUp()`**, al que la infraestructura de prueba llamará para cada prueba que se ejecute:

```
import unittest
```

```
class WidgetTestCase(unittest.TestCase):
    def setUp(self):
        self.widget = Widget('The widget')

    def test_default_widget_size(self):
        self.assertEqual(self.widget.size(), (50,50),
                         'incorrect default size')

    def test_widget_resize(self):
        self.widget.resize(100,150)
        self.assertEqual(self.widget.size(), (100,150),
                         'wrong size after resize')
```

Nota

El orden en que se ejecutan las diversas pruebas se determina por orden alfabético de los nombres de métodos de prueba.

Si el método **`setUp()`** lanza una excepción mientras se ejecuta la prueba, la infraestructura considerará que la prueba ha sufrido un error y no se ejecutará el método de prueba propiamente dicho.

Análogamente, se puede proporcionar un método **`tearDown()`** que haga limpieza después de que se ejecute el método de prueba:

```
import unittest

class WidgetTestCase(unittest.TestCase):
    def setUp(self):
        self.widget = Widget('The widget')

    def tearDown(self):
        self.widget.dispose()
```

Si **`setUp()`** se ejecuta sin errores, **`tearDown()`** se ejecutará tanto si el método de prueba tuvo éxito como si no.

Such a working environment for the testing code is called a *test* *fixture*. A new TestCase instance is created as a unique test fixture used to execute each individual test method. Thus `setUp()`, `tearDown()`, and `TestCase.__init__()` will be called once per test.

It is recommended that you use TestCase implementations to group tests together according to the features they test. `unittest` provides a mechanism for this: the *test suite*, represented by `unittest`'s `TestSuite` class. In most cases, calling `unittest.main()` will do the right thing and collect all the module's test cases for you and execute them.

Sin embargo, si se desea personalizar la construcción del paquete de pruebas, se puede hacer:

```
def suite():
    suite = unittest.TestSuite()
    suite.addTest(WidgetTestCase('test_default_widget_si
    suite.addTest(WidgetTestCase('test_widget_resize'))
    return suite

if __name__ == '__main__':
    runner = unittest.TextTestRunner()
    runner.run(suite())
```

Se puede poner las definiciones de los casos de prueba y de los paquetes de prueba en los mismos módulos que el código que prueban.(tal como `widget.py`), pero sacarlos a un módulo aparte como `test_widget.py` tiene diversas ventajas:

- El módulo de pruebas se puede ejecutar aisladamente desde la línea de órdenes.
- El código de pruebas se puede separar con más facilidad del código de producción.
- Disminuye la tentación de cambiar el código de prueba para ajustarse al código bajo prueba.
- El código de pruebas debería modificarse con mucha menor frecuencia que el código bajo prueba.
- Es más sencillo refactorizar el código bajo prueba.
- El código para probar módulos escritos en C ha de estar en

módulos aparte, así que ¿por qué no mantener la consistencia?
- Si cambia la estrategia de prueba, no hay razón para cambiar el código fuente principal.

Reutilización de código de prueba anterior

Some users will find that they have existing test code that they would like to run from **`unittest`**, without converting every old test function to a **`TestCase`** subclass.

For this reason, **`unittest`** provides a **`FunctionTestCase`** class. This subclass of **`TestCase`** can be used to wrap an existing test function. Set-up and tear-down functions can also be provided.

Dada la siguiente función de prueba:

```
def testSomething():
    something = makeSomething()
    assert something.name is not None
    # ...
```

se puede crear una instancia de caso de prueba de la siguiente manera, con métodos opcionales de preparación y desmontaje:

```
testcase = unittest.FunctionTestCase(testSomething,
                                     setUp=makeSomething
                                     tearDown=deleteSome
```

Nota

Even though **`FunctionTestCase`** can be used to quickly convert an existing test base over to a **`unittest`**-based system, this approach is not recommended. Taking the time to set up proper **`TestCase`** subclasses will make future test refactorings infinitely easier.

In some cases, the existing tests may have been written using the `doctest` module. If so, **`doctest`** provides a `DocTestSuite` class that can automatically build `unittest.TestSuite` instances from the existing **`doctest`**-based tests.

Omitir tests y fallos esperados

Added in version 3.1.

Unittest soporta omitir métodos individuales de tests e incluso clases completas de tests. Además, soporta marcar un test como un «fallo esperado», un test que está roto y va a fallar, pero no debería ser contado como fallo en `TestResult`.

Omitir un test es solo cuestión de emplear el decorator `skip()` o una de sus variantes condicionales, llamando a `TestCase.skipTest()` dentro de `setUp()` o en un método de test, o lanzando `SkipTest` directamente.

La omisión más básica tiene la siguiente forma:

```
class MyTestCase(unittest.TestCase):

    @unittest.skip("demonstrating skipping")
    def test_nothing(self):
        self.fail("shouldn't happen")

    @unittest.skipIf(mylib.__version__ < (1, 3),
                     "not supported in this library vers
    def test_format(self):
        # Tests that work for only a certain version of
        pass

    @unittest.skipUnless(sys.platform.startswith("win"),
    def test_windows_support(self):
        # windows specific testing code
        pass

    def test_maybe_skipped(self):
        if not external_resource_available():
            self.skipTest("external resource not availab
        # test code that depends on the external resourc
        pass
```

Esta es la salida de ejecutar el ejemplo superior en modo verboso:

```
test_format (__main__.MyTestCase.test_format) ... skippe
test_nothing (__main__.MyTestCase.test_nothing) ... skip
test_maybe_skipped (__main__.MyTestCase.test_maybe_skipp
test_windows_support (__main__.MyTestCase.test_windows_s

----------------------------------------------------------
Ran 4 tests in 0.005s

OK (skipped=4)
```

Las clases pueden ser omitidas igual que los métodos:

```
@unittest.skip("showing class skipping")
class MySkippedTestCase(unittest.TestCase):
    def test_not_run(self):
        pass
```

`TestCase.setUp()` puede omitir también el test. Esto es útil cuando un recurso que necesita ser puesto a punto no está disponible.

Los fallos esperados emplean el decorador **`expectedFailure()`**.

```
class ExpectedFailureTestCase(unittest.TestCase):
    @unittest.expectedFailure
    def test_fail(self):
        self.assertEqual(1, 0, "broken")
```

Es fácil lanzar tus propios decoradores que omitan haciendo un decorador que llame a **`skip()`** en el test cuando quiere ser omitido. Este decorador omite el test a menos que el objeto pasado tenga un cierto atributo:

```
def skipUnlessHasattr(obj, attr):
    if hasattr(obj, attr):
        return lambda func: func
    return unittest.skip("{!r} doesn't have {!r}".format
```

Los siguientes decoradores y la excepción implementan la omisión de tests y los fallos esperados:

@unittest.skip(*reason*)

Omitir incondicionalmente el test decorado. *reason* debe describir porqué el test está siendo omitido.

@unittest.skipIf(*condition*, *reason*)

Omitir el test decorado si *condition* es verdadero.

@unittest.skipUnless(*condition*, *reason*)

Omitir el test decorado a menos que *condition* sea verdadero.

@unittest.expectedFailure

Marca el test como un fallo esperado. Si falla el test o hay errores en la función de test misma (en lugar de en alguno de los métodos *test fixture*) será considerado un éxito. Si el test pasa, será considerado un fallo.

exception unittest.SkipTest(*reason*)

Esta excepción es lanzada para omitir un test.

Normalmente puedes usar directamente `TestCase.skipTest()` o uno de los decoradores de omisión en vez de lanzar esta excepción.

Los tests omitidos no ejecutarán `setUp()` o `tearDown()`. Las clases omitidas no ejecutarán `setUpClass()` o `tearDownClass()`. Los módulos omitidos no ejecutarán `setUpModule()` o `tearDownModule()`.

Distinguiendo iteraciones de tests empleando subtests

Added in version 3.4.

Cuando hay diferencias muy pequeñas entre tus tests, por ejemplo algunos parámetros, unittest te permite distinguirlos dentro del cuerpo de un método de test empleando el administrador de contexto **`subTest()`** .

Por ejemplo, el siguiente test:

```
class NumbersTest(unittest.TestCase):

    def test_even(self):
        """
        Test that numbers between 0 and 5 are all even.
        """
        for i in range(0, 6):
            with self.subTest(i=i):
                self.assertEqual(i % 2, 0)
```

producirá la siguiente salida:

```
========================================================
FAIL: test_even (__main__.NumbersTest.test_even) (i=1)
Test that numbers between 0 and 5 are all even.
--------------------------------------------------------
Traceback (most recent call last):
  File "subtests.py", line 11, in test_even
    self.assertEqual(i % 2, 0)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: 1 != 0

========================================================
FAIL: test_even (__main__.NumbersTest.test_even) (i=3)
Test that numbers between 0 and 5 are all even.
--------------------------------------------------------
Traceback (most recent call last):
  File "subtests.py", line 11, in test_even
    self.assertEqual(i % 2, 0)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: 1 != 0
```

```
========================================================
FAIL: test_even (__main__.NumbersTest.test_even) (i=5)
Test that numbers between 0 and 5 are all even.
--------------------------------------------------------
Traceback (most recent call last):
  File "subtests.py", line 11, in test_even
    self.assertEqual(i % 2, 0)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: 1 != 0
```

Sin usar un subtest, la ejecución se pararía después del primer fallo, y el error sería más difícil de diagnosticar porque el valor de `i` no se mostraría:

```
========================================================
FAIL: test_even (__main__.NumbersTest.test_even)
--------------------------------------------------------
Traceback (most recent call last):
  File "subtests.py", line 32, in test_even
    self.assertEqual(i % 2, 0)
AssertionError: 1 != 0
```

Clases y funciones

This section describes in depth the API of **`unittest`**.

Casos de test

class unittest.TestCase(*methodName='runTest'*)

Instances of the **`TestCase`** class represent the logical test units in the **`unittest`** universe. This class is intended to be used as a base class, with specific tests being implemented by concrete subclasses. This class implements the interface needed by the test runner to allow it to drive the tests, and methods that the test code can use to check for and report various kinds of failure.

Cada instancia de **`TestCase`** ejecutará un solo método base:

el método llamado *methodName*. En la mayoría de usos de **`TestCase`**, no tendrás que cambiar el *methodName* ni reimplementar el método por defecto `runTest()`.

Distinto en la versión 3.2: **`TestCase`** puede instancias con éxito sin dar un *methodName*. Esto permite experimentar de manera sencilla con **`TestCase`** en el intérprete interactivo.

Las instancias de **`TestCase`** proveen tres grupos de métodos: un grupo empleado para ejecutar el test, otro usado para que la implementación del test chequee condiciones y reporte fallos, y algunos métodos de indagación que permiten recopilar información sobre el test en si mismo.

Los métodos en el primer grupo (ejecutando el test) son:

setUp()

> Método llamado para preparar el banco de test. Es invocado inmediatamente antes de llamar al método de test; cualquier excepción lanzada por este método que no sea **`AssertionError`** o **`SkipTest`** será considerada un error en vez de un fallo del test. La implementación por defecto no hace nada.

tearDown()

> Método llamado inmediatamente después de que se haya llamado el método de prueba y se haya registrado el resultado. Se llama así aunque el método de ensayo haya planteado una excepción, por lo que la aplicación en las subclases puede tener que ser especialmente cuidadosa en cuanto a la comprobación del estado interno. Cualquier excepción, que no sea **`AssertionError`** o **`SkipTest`**, planteada por este método se considerará un error adicional en lugar de un fallo de la prueba (aumentando así el número total de errores reportados). Este método sólo se llamará si **`setUp()`** tiene éxito, independientemente del resultado del método de prueba. La implementación por defecto no hace nada.

setUpClass()

Un método de clase llamado antes de que los tests en una clase individual sean ejecutados. `setUpClass` es llamado con la clase como el único argumento y debe ser decorada como **`classmethod()`**:

```
@classmethod
def setUpClass(cls):
    ...
```

Vea Class and Module Fixtures para más detalles.

Added in version 3.2.

tearDownClass()

Un método de clase llamado después de que se hayan realizado tests en una clase individual. `tearDownClass` se llama con la clase como único argumento y debe ser decorado como un **`classmethod()`**:

```
@classmethod
def tearDownClass(cls):
    ...
```

Vea Class and Module Fixtures para más detalles.

Added in version 3.2.

run(*result=None*)

Ejecutar la prueba, recogiendo el resultado en el objeto **`TestResult`** pasado como *result*. Si se omite *result* o `None`, se crea un objeto resultado temporal (llamando al método **`defaultTestResult()`**) y se emplea ese. El objeto resultante se devuelve al invocador de **`run()`**.

El mismo efecto puede conseguirse simplemente llamando a la instancia **`TestCase`**.

Distinto en la versión 3.3: Las versiones previas de `run` no retornaban el resultado. Tampoco lo hacía la llamada a una instancia.

skipTest(*reason*)

Llamar a esto durante un método de prueba o **`setUp()`** se salta el test actual. Ver Omitir tests y fallos esperados para más información.

Added in version 3.1.

subTest(*msg=None*, ***params*)

Retorna un gestor de contexto que ejecuta el bloque de código adjunto como un subtest. *msg* y *params* son valores opcionales y arbitrarios que se muestran cuando falla un subtest, permitiéndole identificarlos claramente.

Un caso de test puede contener cualquier número de declaraciones de subtest, y pueden anidarse arbitrariamente.

Ver Distinguiendo iteraciones de tests empleando subtests para más información.

Added in version 3.4.

debug()

Realice el test sin recoger el resultado. Esto permite que las excepciones planteadas por el test se propaguen al invocado, y puede utilizarse para apoyar la ejecución de tests bajo un depurador.

La clase **`TestCase`** proporciona varios métodos de afirmación para comprobar y reportar fallos. En la siguiente tabla se enumeran los métodos más utilizados (consulte las tablas siguientes para ver más métodos de afirmación):

Método	Comprueba que	Nuevo en

assertEqual(a, b)	a == b	
assertNotEqual(a, b)	a != b	
assertTrue(x)	bool(x) is True	
assertFalse(x)	bool(x) is False	
assertIs(a, b)	a is b	3.1
assertIsNot(a, b)	a is not b	3.1
assertIsNone(x)	x is None	3.1
assertIsNotNone(x)	x is not None	3.1
assertIn(a, b)	a in b	3.1
assertNotIn(a, b)	a not in b	3.1
assertIsInstance(a, b)	isinstance(a, b)	3.2
assertNotIsInstance(a, b)	not isinstance(a, b)	3.2
assertIsSubclass(a, b)	issubclass(a, b)	3.14
assertNotIsSubclass(a, b)	not issubclass(a, b)	3.14

Todos los métodos de aserción aceptan un argumento *msg* que, si se especifica, se utiliza como mensaje de error en caso de fallo (véase también **`longMessage`**). Tenga en cuenta que el argumento de la palabra clave *msg* puede pasarse a **`assertRaises()`**, **`assertRaisesRegex()`**, **`assertWarns()`**, **`assertWarnsRegex()`** sólo cuando se utilizan como gestor de contexto.

assertEqual(*first*, *second*, *msg*=*None*)

Testea que *first* y *second* son iguales. Si los valores no comparan como iguales, el test fallará.

Además, si *first* y *second* son exactamente del mismo tipo y uno de lista, tuple, dict, set, frozenset o str o cualquier tipo que una subclase registre con **`addTypeEqualityFunc()`** se llamará a la función de igualdad específica del tipo para generar un mensaje de error por defecto más útil (véase también la lista de métodos específicos del tipo).

Distinto en la versión 3.1: Añadida la llamada automática de la función de igualdad de tipo específico.

Distinto en la versión 3.2: **`assertMultiLineEqual()`** añadido como la función por defecto para igualdad de tipos cuando se comparan cadenas.

assertNotEqual(*first*, *second*, *msg=None*)

Testea que *first* y *second* no son iguales. Si los valores son iguales, el test fallará.

assertTrue(*expr*, *msg=None*)
assertFalse(*expr*, *msg=None*)

Testea que *expr* es verdadero (o falso).

Note que esto es equivalente a `bool(expr) is True` y no a `expr is True` (use `assertIs(expr, True)` para lo último). Este método también debe evitarse cuando se disponga de métodos más específicos (por ejemplo, `assertEqual(a, b)` en lugar de `assertTrue(a == b)`), porque proporcionan un mejor mensaje de error en caso de fallo.

assertIs(*first*, *second*, *msg=None*)
assertIsNot(*first*, *second*, *msg=None*)

Testea si *first* y *second* son (o no) el mismo objeto.

Added in version 3.1.

assertIsNone(*expr*, *msg=None*)
assertIsNotNone(*expr*, *msg=None*)

Testea que *expr* es (o no es) `None`.

Added in version 3.1.

assertIn(*member*, *container*, *msg=None*)
assertNotIn(*member*, *container*, *msg=None*)

Testea que *member* está (o no está) en *container*.

Added in version 3.1.

assertIsInstance(*obj*, *cls*, *msg=None*) assertNotIsInstance(*obj*, *cls*, *msg=None*)

Testea que *obj* es (o no es) una instancia de *cls* (que puede ser una clase o una tupla de clases, de la misma forma que soporta `isinstance()`). Para chequear por el tipo exacto, use `assertIs(type(obj), cls)`.

Added in version 3.2.

assertIsSubclass(*cls*, *superclass*, *msg=None*)

assertNotIsSubclass(*cls*, *superclass*, *msg=None*)

Test that *cls* is (or is not) a subclass of *superclass* (which can be a class or a tuple of classes, as supported by `issubclass()`). To check for the exact type, use `assertIs(cls, superclass)`.

Added in version 3.14.

Es también posible chequear la producción de excepciones, advertencias y mensajes de log usando los siguientes métodos:

Método	**Chequea que**	**Nuevo en**
`assertRaises(exc, fun, *args, **kwds)`	`fun(*args, **kwds)` lanza *exc*	
`assertRaisesRegex(exc, r, fun, *args, **kwds)`	`fun(*args, **kwds)` lanza *exc* y el mensaje coincide con regex *r*	3.1
`assertWarns(warn, fun, *args, **kwds)`	`fun(*args, **kwds)` lanza *warn*	3.2
`assertWarnsRegex(warn, r, fun, *args, **kwds)`	`fun(*args, **kwds)` lanza *warn* y el mensaje coincide con regex *r*	3.2
`assertLogs(logger, level)`	El bloque `with` vuelca sus logs a *logger* con el *level* mínimo	3.4
`assertLogs(logger, level)`	El bloque `with` no ingresa El bloque `with` vuelca sus logs a *logger* con el *level* mínimo	3.10

assertRaises(*exception*, *callable*, **args*, ***kwds*)

assertRaises(*exception*, ***, *msg=None*)

Testea que se lanza una excepción cuando se llama a *callable* con cualquier argumento posicional o de palabra clave que también se pasa a `assertRaises()`. El test pasa si se lanza *exception*, es un error si se lanza otra excepción, o falla si no se lanza ninguna excepción. Para tener en cuenta cualquiera de un grupo de excepciones, una tupla que contenga las clases de excepción puede ser pasada como *exception*.

Si sólo se dan los argumentos de *exception* y posiblemente *msg*, retorna un administrador de contexto para que el código testado pueda ser escrito en línea en lugar de como una función:

```
with self.assertRaises(SomeException):
    do_something()
```

Cuando se emplea como un administrador de contexto, **`assertRaises()`** acepta el argumento por palabra clave adicional *msg*.

The context manager will store the caught exception object in its **`exception`** attribute. This can be useful if the intention is to perform additional checks on the exception raised:

```
with self.assertRaises(SomeException) as cm:
    do_something()

the_exception = cm.exception
self.assertEqual(the_exception.error_code, 3)
```

Distinto en la versión 3.1: Añadió la capacidad de usar **`assertRaises()`** como gestor de contexto.

Distinto en la versión 3.2: Added the **`exception`** attribute.

Distinto en la versión 3.3: Añadido el argumento por palabra clave *msg* cuando se emplea un gestor de contexto.

assertRaisesRegex(*exception*, *regex*, *callable*, **args*, ***kwds*)
assertRaisesRegex(*exception*, *regex*, ***, *msg=None*)

Como `assertRaises()` pero también testea que *regex* coincide en la representación de la cadena de la excepción planteada. *regex* puede ser un objeto de expresión regular o una cadena que contiene una expresión regular adecuada para ser usada por `re.search()`. Ejemplos:

```
self.assertRaisesRegex(ValueError, "invalid lit
                       int, 'XYZ')
```

o:

```
with self.assertRaisesRegex(ValueError, 'litera
   int('XYZ')
```

Added in version 3.1: Añadido bajo el nombre de `assertRaisesRegexp`.

Distinto en la versión 3.2: Renombrado a `assertRaisesRegex()`.

Distinto en la versión 3.3: Añadido el argumento por palabra clave *msg* cuando se emplea un gestor de contexto.

assertWarns(*warning*, *callable*, **args*, ***kwds*)
assertWarns(*warning*, *, *msg=None*)

Testea que una advertencia se activa cuando se llama a *callable* con cualquier argumento posicional o de palabra clave que también se pasa a `assertWarns()`. El test pasa si se activa el *warning* y falla si no lo hace. Cualquier excepción es un error. Para considerar cualquiera de un grupo de advertencias, una tupla que contenga las clases de advertencia puede ser pasada como *warnings*.

Si sólo se dan los argumentos de *advertencia* y *msg*, retorna un gestor de contexto para que el código testado pueda ser escrito en línea en lugar de como una función:

```
with self.assertWarns(SomeWarning):
    do_something()
```

Cuando se usa como gestor de contexto, `assertWarns()` acepta el argumento de palabra clave adicional *msg*.

The context manager will store the caught warning object in its **`warning`** attribute, and the source line which triggered the warnings in the **`filename`** and **`lineno`** attributes. This can be useful if the intention is to perform additional checks on the warning caught:

```
with self.assertWarns(SomeWarning) as cm:
    do_something()

self.assertIn('myfile.py', cm.filename)
self.assertEqual(320, cm.lineno)
```

Este método funciona independientemente de los filtros de aviso que estén en su lugar cuando se llame.

Added in version 3.2.

Distinto en la versión 3.3: Añadido el argumento por palabra clave *msg* cuando se emplea un gestor de contexto.

assertWarnsRegex(*warning*, *regex*, *callable*, **args*, ***kwds*)
assertWarnsRegex(*warning*, *regex*, ***, *msg=None*)

Como `assertWarns()` pero también testea que *regex* coincide en el mensaje del aviso disparado. *regex* puede ser un objeto de expresión regular o una cadena que contiene una expresión regular adecuada para ser usada por `re.search()`. Ejemplo:

```
self.assertWarnsRegex(DeprecationWarning,
                      r'legacy_function\(\) is
                      legacy_function, 'XYZ')
```

o:

```
with self.assertWarnsRegex(RuntimeWarning, 'uns
    frobnicate('/etc/passwd')
```

Added in version 3.2.

Distinto en la versión 3.3: Añadido el argumento por palabra clave *msg* cuando se emplea un gestor de contexto.

assertLogs(*logger=None*, *level=None*)

Un gestor de contexto para comprobar que al menos un mensaje está registrado en el *logger* o en uno de sus hijos, con al menos el *level* dado.

Si se da, *logger* debería ser un objeto **`logging.Logger`** o un **`str`** dando el nombre de un logger. El valor por defecto es el root logger, que captará todos los mensajes.

Si se da, *level* debería ser un nivel de registro numérico o su equivalente en cadena de caracteres (por ejemplo, ya sea `"ERROR"` o **`logging.ERROR`**). El valor predeterminado es **`logging.INFO`**.

El test pasa si al menos un mensaje emitido dentro del bloque `with` coincide con las condiciones de *logger* y *level*, de lo contrario falla.

El objeto devuelto por el gestor de contexto es un ayudante de grabación que lleva un registro de los mensajes de registro que coinciden. Tiene dos atributos:

records

Una lista de objetos **`logging.LogRecord`** de los mensajes de log coincidentes.

output

Una lista de objetos **`str`** con la salida forrajeada en los mensajes coincidentes.

Ejemplo:

```
with self.assertLogs('foo', level='INFO') as cm
    logging.getLogger('foo').info('first messag
    logging.getLogger('foo.bar').error('second 
self.assertEqual(cm.output, ['INFO:foo:first me
                             'ERROR:foo.bar:sec
```

Added in version 3.4.

assertNoLogs(*logger=None*, *level=None*)

Un gestor de contexto para comprobar que al menos un mensaje está registrado en el *logger* o en uno de sus hijos, con al menos el *level* dado.

Si se da, *logger* debería ser un objeto **`logging.Logger`** o un **`str`** dándole el nombre de un logger. El valor por defecto es el root logger, que captará todos los mensajes.

Si se da, *level* debería ser un nivel de registro numérico o su equivalente en cadena de caracteres (por ejemplo, ya sea `"ERROR"` o **`logging.ERROR`**). El valor predeterminado es **`logging.INFO`**.

A diferencia de **`assertLogs()`**, el gestor de contexto no devolverá nada.

Added in version 3.10.

Hay también otros métodos empleados para realizar comprobaciones más específicas, tales como:

Método	Comprueba que	Nuevo en
`assertAlmostEqual(a, b)`	`round(a-b, 7) == 0`	
`assertNotAlmostEqual(a, b)`	`round(a-b, 7) != 0`	
`assertGreater(a, b)`	`a > b`	3.1
`assertGreaterEqual(a, b)`	`a >= b`	3.1

`assertLess(a, b)`	a < b	3.1
`assertLessEqual(a, b)`	a <= b	3.1
`assertRegex(s, r)`	r.search(s)	3.1
`assertNotRegex(s, r)`	not r.search(s)	3.2
`assertCountEqual(a, b)`	a contains the same elements as *b*, regardless of their order.	3.2
`assertStartsWith(a, b)`	a.startswith(b)	3.14
`assertNotStartsWith(a, b)`	not a.startswith(b)	3.14
`assertEndsWith(a, b)`	a.endswith(b)	3.14
`assertNotEndsWith(a, b)`	not a.endswith(b)	3.14
`assertHasAttr(a, b)`	hastattr(a, b)	3.14
`assertNotHasAttr(a, b)`	not hastattr(a, b)	3.14

assertAlmostEqual(*first*, *second*, *places*=*7*, *msg*=*None*, *delta*=*None*)

assertNotAlmostEqual(*first*, *second*, *places*=*7*, *msg*=*None*, *delta*=*None*)

Testea que *first* y *second* son aproximadamente (o no aproximadamente) iguales calculando su diferencia, redondeando al número dado de puntos *places* decimales (por defecto 7), y comparado a cero. Nótese que estos métodos redondean los valores al número dado de *puntos decimales* (por ejemplo como la función `round()`) y no *cifras significativas*.

Si se suministra *delta* en vez de *places* que entonces la diferencia entre *first* y *second* deba ser menor o igual a (o mayor que) *delta*.

Suministrar tanto *delta* como *places* lanza un `TypeError`.

Distinto en la versión 3.2: `assertAlmostEqual()` considera automáticamente casi iguales a los objetos que se comparan igual. `assertNotAlmostEqual()` falla automáticamente si los objetos comparan iguales. Añadido el argumento de palabra clave *delta*.

assertGreater(*first*, *second*, *msg*=*None*)

assertGreaterEqual(*first*, *second*, *msg=None*)
assertLess(*first*, *second*, *msg=None*)
assertLessEqual(*first*, *second*, *msg=None*)

Prueba que *first* es respectivamente >, > =, < o < = que *second* dependiendo del nombre del método. Si no, el test fallará:

```
>>> self.assertGreaterEqual(3, 4)
AssertionError: "3" unexpectedly not greater th
```

Added in version 3.1.

assertRegex(*text*, *regex*, *msg=None*)
assertNotRegex(*text*, *regex*, *msg=None*)

Testea que una búsqueda *regex* coincide (o no coincide) con el *text*. En caso de fallo, el mensaje de error incluirá el patrón y el *text* (o el patrón y la parte de *text* que coincida inesperadamente). *regex* puede ser un objeto de expresión regular o una cadena que contiene una expresión regular adecuada para ser utilizada por **`re.search()`**.

Added in version 3.1: Añadido bajo el nombre de `assertRegexpMatches`.

Distinto en la versión 3.2: El método `assertRegexpMatches()` ha sido renombrado a **`assertRegex()`**.

Added in version 3.2: **`assertNotRegex()`**.

assertCountEqual(*first*, *second*, *msg=None*)

Testea que la secuencia *first* contiene los mismos elementos que *second*, independientemente de su orden. Cuando no lo hagan, se generará un mensaje de error con las diferencias entre las secuencias.

Los elementos duplicados *no* son ignorados cuando se comparan *first* y *second*. Verifica si cada elemento tiene la misma cuenta en ambas secuencias. Equivalente a: `assertEqual(Counter(list(first)), Counter(list(second)))` pero funciona también con secuencias de objetos que no son hashable.

Added in version 3.2.

assertStartsWith(*s*, *prefix*, *msg=None*)

assertNotStartsWith(*s*, *prefix*, *msg=None*)

Test that the Unicode or byte string *s* starts (or does not start) with a *prefix*. *prefix* can also be a tuple of strings to try.

Added in version 3.14.

assertEndsWith(*s*, *suffix*, *msg=None*)

assertNotEndsWith(*s*, *suffix*, *msg=None*)

Test that the Unicode or byte string *s* ends (or does not end) with a *suffix*. *suffix* can also be a tuple of strings to try.

Added in version 3.14.

assertHasAttr(*obj*, *name*, *msg=None*)

assertNotHasAttr(*obj*, *name*, *msg=None*)

Test that the object *obj* has (or has not) an attribute *name*.

Added in version 3.14.

El método **`assertEqual()`** envía la comprobación de igualdad de los objetos del mismo tipo a diferentes métodos específicos de tipo. Estos métodos ya están implementados para la mayoría de los tipos incorporados, pero también es posible registrar nuevos métodos usando `addTypeEqualityFunc()`:

addTypeEqualityFunc(*typeobj*, *function*)

Registra un método específico de tipo llamado por `assertEqual()` para comprobar si dos objetos del mismo *typeobj* (no subclases) comparan como iguales. *function* debe tomar dos argumentos posicionales y un tercer argumento de palabra clave msg=None tal y como lo hace **`assertEqual()`**. Debe lanzar `self.failureException(msg)` cuando se detecta una desigualdad entre los dos primeros parámetros, posiblemente proporcionando información útil y explicando las desigualdades en detalle en el mensaje de error.

Added in version 3.1.

La lista de métodos específicos de tipo utilizados automáticamente por `assertEqual()` se resumen en la siguiente tabla. Tenga en cuenta que normalmente no es necesario invocar estos métodos directamente.

Método	**Usado para comparar**	**Nuevo en**
`assertMultiLineEqual(a, b)`	strings	3.1
`assertSequenceEqual(a, b)`	sequences	3.1
`assertListEqual(a, b)`	lists	3.1
`assertTupleEqual(a, b)`	tuples	3.1
`assertSetEqual(a, b)`	sets or frozensets	3.1
`assertDictEqual(a, b)`	dicts	3.1

assertMultiLineEqual(*first*, *second*, *msg=None*)

Testea que la cadena multilínea *first* es igual a la cadena *second*. Cuando no sea igual, una diferencia de las dos cadenas que resalte las diferencias se incluirá en el mensaje de error. Este método se utiliza por defecto cuando se comparan cadenas con `assertEqual()`.

Added in version 3.1.

assertSequenceEqual(*first*, *second*, *msg=None*, *seq_type=None*)

Testea que dos secuencias son iguales. Si se suministra un *seq_type*, tanto *first* como *second* deben ser instancias de *seq_type* o se lanzará un fallo. Si las secuencias son diferentes se construye un mensaje de error que muestra la diferencia entre las dos.

Este método no es llamado directamente por `assertEqual()`, pero se usa para implementar `assertListEqual()` y `assertTupleEqual()`.

Added in version 3.1.

assertListEqual(*first*, *second*, *msg=None*)
assertTupleEqual(*first*, *second*, *msg=None*)

Testea que dos listas o tuplas son iguales. Si no es así, se construye un mensaje de error que muestra sólo las diferencias entre las dos. También se lanza un error si alguno de los parámetros es del tipo equivocado. Estos métodos se utilizan por defecto cuando se comparan listas o tuplas con `assertEqual()`.

Added in version 3.1.

assertSetEqual(*first*, *second*, *msg=None*)

Testea que dos conjuntos son iguales. Si no es así, se construye un mensaje de error que enumera las diferencias entre los conjuntos. Este método se utiliza por defecto cuando se comparan los conjuntos o frozensets con `assertEqual()`.

Fails if either of *first* or *second* does not have a `difference()` method.

Added in version 3.1.

assertDictEqual(*first*, *second*, *msg=None*)

Testea que dos diccionarios son iguales. Si no es así, se construye un mensaje de error que muestra las diferencias en los diccionarios. Este método se usará por defecto para comparar los diccionarios en las llamadas a **`assertEqual()`**.

Added in version 3.1.

Finalmente, **`TestCase`** proporciona los siguientes métodos y atributos:

fail(*msg*=*None*)

Señala un fallo del test incondicionalmente, con *msg* o `None` para el mensaje de error.

failureException

Este atributo de clase da la excepción lanzada por el método de test. Si un marco de pruebas necesita utilizar una excepción especializada, posiblemente para llevar información adicional, debe subclasificar esta excepción para «jugar limpio» con el marco. El valor inicial de este atributo es **`AssertionError`**.

longMessage

Este atributo de clase determina lo que ocurre cuando se pasa un mensaje de fallo personalizado como el argumento msg a una llamada assertXYY que falla. `True` es el valor por defecto. En este caso, el mensaje personalizado se añade al final del mensaje de fallo estándar. Cuando se establece en `False`, el mensaje personalizado reemplaza al mensaje estándar.

La configuración de la clase puede ser anulada en los métodos de test individuales asignando un atributo de instancia, self.longMessage, a `True` o `False` antes de llamar a los métodos assert.

La configuración de la clase se reajusta antes de cada llamada de test.

Added in version 3.1.

maxDiff

Este atributo controla la longitud máxima de las diferencias de salida de métodos assert que reportan diferencias en caso de fallo. El valor predeterminado es de 80*8 caracteres. Los métodos assert afectados por este atributo son **`assertSequenceEqual()`** (incluyendo todos los métodos de comparación de secuencias que le delegan), **`assertDictEqual()`** y **`assertMultiLineEqual()`**.

Poner `maxDiff` en `None` significa que no hay una longitud máxima de diferencias.

Added in version 3.2.

Los marcos de test pueden utilizar los siguientes métodos para reunir información sobre el test:

countTestCases()

Retorna el número de tests representados por este objeto de test. Para las instancias de **`TestCase`**, este siempre será `1`.

defaultTestResult()

Retorna una instancia de la clase de resultado de test que debería utilizarse para esta clase de caso de test (si no se proporciona otra instancia de resultado al método **`run()`**).

Para las instancias de **`TestCase`**, ésta siempre será una instancia de **`TestResult`**; las subclases de **`TestCase`** deben anular esto según sea necesario.

id()

Devuelva una cadena que identifique el caso de test específico. Normalmente es el nombre completo del método de test, incluyendo el nombre del módulo y de la clase.

shortDescription()

Devuelve una descripción de la prueba, o `None` si no se ha proporcionado ninguna descripción. La implementación por defecto de este método devuelve la primera línea de la docstring del método de test, si está disponible, o `None` .

Distinto en la versión 3.1: En 3.1 esto se cambió para añadir el nombre del test a la descripción corta incluso en presencia de una docstring. Esto causó problemas de compatibilidad con las extensiones de unittest y la adición del nombre de test fue movida a la **`TextTestResult`** en Python 3.2.

addCleanup(*function*, /, **args*, ***kwargs*)

Añade una función que se llamará después de **`tearDown()`** a los recursos de limpieza utilizados durante el test. Las funciones se llamarán en orden inverso al orden en que se agregan (LIFO). Se llaman con cualquier argumento y argumentos de palabra clave que se pase a **`addCleanup()`** cuando se agregan.

Si **`setUp()`** falla, lo que significa que **`tearDown()`** no se llama, entonces cualquier función de limpieza añadida seguirá siendo llamada.

Added in version 3.1.

enterContext(*cm*)

Introduce el gestor de contexto **`context()`** suministrado. Si es exitoso, añade también su método **`__exit__()`** como función de limpieza mediante **`addCleanup()`** y retorna el resultado del método **`__enter__()`**.

Added in version 3.11.

doCleanups()

Este método se llama incondicionalmente después de `tearDown()`, o después de `setUp()` si **`setUp()`** lanza una excepción.

Es responsable de llamar a todas las funciones de limpieza añadidas por `addCleanup()`. Si necesitas que las funciones de limpieza se llamen *con anterioridad* a `tearDown()` entonces puedes llamar a `doCleanups()` tú mismo.

`doCleanups()` saca los métodos de la pila de funciones de limpieza uno a uno, así que se puede llamar en cualquier momento.

Added in version 3.1.

classmethod addClassCleanup(*function*, */*, **args*, ***kwargs*)

Añade una función que se llamará después de `tearDownClass()` para limpiar recursos utilizados durante la clase de test. Las funciones se llamarán en orden inverso al orden en que se agregan (LIFO). Se llaman con cualquier argumento y argumento de palabra clave que se pase a `addClassCleanup()` cuando se añadan.

Si `setUpClass()` falla, lo que significa que `tearDownClass()` no se invoca, entonces cualquier función de limpieza añadida seguirá siendo llamada.

Added in version 3.8.

classmethod enterClassContext(*cm*)

Introduce el context manager suministrado. Si es exitoso, añade también su método `__exit__()` como función de limpieza mediante `addClassCleanup()` y retorna el resultado del método `__enter__()`.

Added in version 3.11.

classmethod doClassCleanups()

Este método se llama incondicionalmente después de `tearDownClass()`, o después de `setUpClass()` si **`setUpClass()`** lanza una excepción.

Es responsable de llamar a todas las funciones de limpieza añadidas por **`addCleanupClass()`**. Si necesitas que las funciones de limpieza se llamen *con anterioridad* a `tearDownClass()` entonces puedes llamar a **`doCleanupsClass()`** tú mismo.

`doCleanupsClass()` saca los métodos de la pila de funciones de limpieza de uno en uno, así que se puede llamar en cualquier momento.

Added in version 3.8.

class unittest.IsolatedAsyncioTestCase(*methodName='runTest'*)

Esta clase proporciona una API similar a `TestCase` y también acepta corutinas como funciones de test.

Added in version 3.8.

loop_factory

The *loop_factory* passed to `asyncio.Runner`. Override in subclasses with `asyncio.EventLoop` to avoid using the asyncio policy system.

Added in version 3.13.

async asyncSetUp()

Method called to prepare the test fixture. This is called after `TestCase.setUp()`. This is called immediately before calling the test method; other than `AssertionError` or `SkipTest`, any exception raised by this method will be considered an error rather than a test failure. The default implementation does nothing.

async asyncTearDown()

Method called immediately after the test method has been called and the result recorded. This is called before `tearDown()`. This is called even if the test method raised an exception, so the implementation in subclasses may need to be particularly careful about checking internal state. Any exception, other than `AssertionError` or `SkipTest`, raised by this method will be considered an additional error rather than a test failure (thus increasing the total number of reported errors). This method will only be called if the `asyncSetUp()` succeeds, regardless of the outcome of the test method. The default implementation does nothing.

addAsyncCleanup(*function*, */*, **args*, ***kwargs*)

Este método acepta una corutina que puede ser utilizada como función de limpieza.

async enterAsyncContext(*cm*)

Introduce el asynchronous context manager suministrado. Si es exitoso, añade también su método `__aexit__()` como función de limpieza mediante `addAsyncCleanup()` y retorna el resultado del método `__aenter__()`.

Added in version 3.11.

run(*result=None*)

Sets up a new event loop to run the test, collecting the result into the `TestResult` object passed as *result*. If *result* is omitted or `None`, a temporary result object is created (by calling the `defaultTestResult()` method) and used. The result object is returned to `run()`"s caller. At the end of the test all the tasks in the event loop are cancelled.

Un ejemplo ilustrando el orden:

```
from unittest import IsolatedAsyncioTestCase

events = []


class Test(IsolatedAsyncioTestCase):


    def setUp(self):
        events.append("setUp")

    async def asyncSetUp(self):
        self._async_connection = await AsyncConnect
        events.append("asyncSetUp")

    async def test_response(self):
        events.append("test_response")
        response = await self._async_connection.get
        self.assertEqual(response.status_code, 200)
        self.addAsyncCleanup(self.on_cleanup)

    def tearDown(self):
        events.append("tearDown")

    async def asyncTearDown(self):
        await self._async_connection.close()
        events.append("asyncTearDown")

    async def on_cleanup(self):
        events.append("cleanup")

if __name__ == "__main__":
    unittest.main()
```

Después de ejecutar el test, `events` contendría `[“setUp”, “asyncSetUp”, “test_response”, “asyncTearDown”, “tearDown”, “cleanup”]`.

class unittest.FunctionTestCase(*testFunc*, *setUp=None*, *tearDown=None*, *description=None*)

This class implements the portion of the `TestCase` interface which allows the test runner to drive the test, but does not provide the methods which test code can use to check and report errors. This is used to create test cases using legacy test code, allowing it to be integrated into a **`unittest`**-based test framework.

Agrupando tests

class unittest.TestSuite(*tests=()*)

Esta clase representa una agregación de casos de test individuales y conjuntos de tests. La clase presenta la interfaz que necesita el corredor de tests para poder ser ejecutado como cualquier otro caso de test. Ejecutar una instancia `TestSuite` es lo mismo que iterar sobre el conjunto, ejecutando cada test individualmente.

Si se indican *tests*, debe ser un iterable de casos de test individuales u otros conjuntos de tests que se usarán para construir el conjunto inicialmente. Se proporcionan métodos adicionales para añadir casos de test y conjuntos a la colección más adelante.

Los objetos de `TestSuite` se comportan de manera muy parecida a los objetos de `TestCase`, excepto que no implementan un test. En cambio, se usan para agregar tests en grupos de tests que deben ser ejecutados juntos. Existen algunos métodos adicionales para agregar tests a las instancias de **`TestSuite`**:

addTest(*test*)

Añade un `TestCase` o `TestSuite` al conjunto.

addTests(*tests*)

Añade todos los tests de un iterable de `TestCase` y `TestSuite` a este conjunto de tests.

Esto equivale a iterar sobre *tests*, llamando a `addTest()` para cada elemento.

`TestSuite` comparte los siguientes métodos con `TestCase`:

run(*result*)

Ejecuta los tests asociados a este conjunto, recogiendo el resultado en el objeto de resultado del test pasado como *result*. Tenga en cuenta que a diferencia de `TestCase.run()`, `TestSuite.run()` requiere que se pase el objeto resultado.

debug()

Ejecuta los tests asociados con este conjunto sin recoger los resultados. Esto permite que las excepciones lanzadas por este test sean propagadas al invocador y puedes ser usadas para apoyar tests que están ejecutándose con un debugger.

countTestCases()

Retorna el numero de tests representados por este objeto de test, incluidos todos los test individuales y los sub-conjuntos.

iter()

Tests grouped by a **`TestSuite`** are always accessed by iteration. Subclasses can lazily provide tests by overriding **`__iter__()`**. Note that this method may be called several times on a single suite (for example when counting tests or comparing for equality) so the tests returned by repeated iterations before `TestSuite.run()` must be the same for each call iteration. After **`TestSuite.run()`**, callers should not rely on the tests returned by this method unless the caller uses a subclass that overrides **`TestSuite._removeTestAtIndex()`** to preserve test references.

Distinto en la versión 3.2: In earlier versions the **`TestSuite`** accessed tests directly rather than through iteration, so overriding **`__iter__()`** wasn't sufficient for providing tests.

Distinto en la versión 3.4: In earlier versions the **`TestSuite`** held references to each **`TestCase`** after **`TestSuite.run()`**. Subclasses can restore that behavior by overriding **`TestSuite._removeTestAtIndex()`**.

In the typical usage of a **`TestSuite`** object, the **`run()`** method is invoked by a **`TestRunner`** rather than by the end-user test harness.

Cargando y ejecutando tests

class unittest.TestLoader

The **`TestLoader`** class is used to create test suites from classes and modules. Normally, there is no need to create an instance of this class; the **`unittest`** module provides an instance that can be shared as **`unittest.defaultTestLoader`**. Using a subclass or instance, however, allows customization of some configurable properties.

Los objetos **`TestLoader`** tienen los siguientes atributos:

errors

Una lista de los errores no fatales encontrados durante la carga de tests. No reseteados por el cargador en ningún momento. Los errores fatales son señalados por el método relevante que lanza una excepción al invocador. Los errores no fatales también son indicados por una prueba sintética que lanzará el error original cuando se ejecute.

Added in version 3.5.

Los objetos **`TestLoader`** tienen los siguientes métodos:

loadTestsFromTestCase(*testCaseClass*)

Return a suite of all test cases contained in the `TestCase`-derived `testCaseClass`.

A test case instance is created for each method named by `getTestCaseNames()`. By default these are the method names beginning with `test`. If `getTestCaseNames()` returns no methods, but the `runTest()` method is implemented, a single test case is created for that method instead.

loadTestsFromModule(*module*, ***, *pattern=None*)

Devuelva un conjunto de todos los casos de test contenidos en el módulo dado. Este método busca en *module* clases derivadas de `TestCase` y crea una instancia de la clase para cada método de test definido para la clase.

Nota

Aunque el uso de una jerarquía de clases derivadas de `TestCase` puede ser conveniente para compartir configuraciones y funciones de ayuda, la definición de métodos de test en clases base que no están destinadas a ser instanciadas directamente no complementa bien con este método. Hacerlo, sin embargo, puede ser útil cuando las configuraciones son diferentes y están definidas en subclases.

Si un módulo proporciona una función `load_tests` será llamado para cargar los tests. Esto permite a los módulos personalizar la carga de los tests. Este es el load_tests protocol. El argumento *pattern* se pasa como tercer argumento a `load_tests`.

Distinto en la versión 3.2: Se ha añadido soporte para `load_tests`.

Distinto en la versión 3.5: Se ha agregado soporte para un argumento *pattern* de solo palabra clave.

Distinto en la versión 3.12: El parámetro no documentado y no oficial *use_load_tests* ha sido eliminado.

loadTestsFromName(*name*, *module=None*)

Retorna un conjunto de todos los casos de test dado un especificador de cadena.

El especificador *name* es un «nombre punteado» que puede resolverse ya sea a un módulo, una clase de caso de test, un método de test dentro de una clase de caso de test, una instancia `TestSuite`, o un objeto invocable que devuelve una instancia `TestCase` o `TestSuite`. Estas comprobaciones se aplican en el orden que se indica aquí; es decir, un método en una posible clase de caso de test se recogerá como «un método de test dentro de una clase de caso de test", en lugar de «un objeto invocable".

For example, if you have a module `SampleTests` containing a `TestCase`-derived class `SampleTestCase` with three test methods (`test_one()`, `test_two()`, and `test_three()`), the specifier `'SampleTests.SampleTestCase'` would cause this method to return a suite which will run all three test methods. Using the specifier `'SampleTests.SampleTestCase.test_two'` would cause it to return a test suite which will run only the `test_two()` test method. The specifier can refer to modules and packages which have not been imported; they will be imported as a side-effect.

El método opcionalmente resuelve *name* relativo al *module* dado.

Distinto en la versión 3.5: Si un `ImportError` o `AttributeError` ocurre mientras atraviesa *name* entonces se devolverá un test sintético que lanza ese error cuando se ejecuta. Estos errores están incluidos en los errores acumulados por self.errors.

loadTestsFromNames(*names*, *module=None*)

Similar a **`loadTestsFromName()`**, pero toma una secuencia de nombres en lugar de un solo nombre. El valor de retorno es una suite de tests que soporta todos los test definidos para cada nombre.

getTestCaseNames(*testCaseClass*)

Devuelve una secuencia ordenada de nombres de métodos encontrados dentro de *testCaseClass*; esta debería ser una subclase de **`TestCase`**.

discover(*start_dir*, *pattern='test*.py'*, *top_level_dir=None*)

Encuentra todos los módulos de prueba recurriendo a subdirectorios del directorio de inicio especificado, y retorna un objeto de TestSuite que los contenga. Sólo se cargarán los archivos de test que coincidan con el *pattern*. (Utilizando la coincidencia de patrones de estilo de shell.) Sólo se cargarán los nombres de los módulos que sean importables (es decir, que sean identificadores Python válidos).

All test modules must be importable from the top level of the project. If the start directory is not the top level directory then *top_level_dir* must be specified separately.

Si la importación de un módulo falla, por ejemplo debido a un error de sintaxis, entonces esto se registrará como un error único y el descubrimiento continuará. Si el fallo en la importación se debe a que **`SkipTest`** se ha lanzado, se registrará como un salto en lugar de un error.

Si se encuentra un paquete (un directorio que contiene un archivo llamado `__init__.py`), se comprobará si el paquete tiene una función `load_tests`. Si existe, entonces se invocará `package.load_tests(loader, tests, pattern)`. Test discovery se encarga de asegurar que un paquete sólo se comprueba una vez durante una invocación, incluso si la propia función load_tests llama a `loader.discover`.

Si `load_tests` existe, entonces el descubrimiento *no* recurre en el paquete, `load_tests` es responsable de cargar todos los tests en el paquete.

The pattern is deliberately not stored as a loader attribute so that packages can continue discovery themselves.

top_level_dir is stored internally, and used as a default to any nested calls to `discover()`. That is, if a package's `load_tests` calls `loader.discover()`, it does not need to pass this argument.

start_dir puede ser un nombre de módulo punteado así como un directorio.

Added in version 3.2.

Distinto en la versión 3.4: Los módulos que lanzan **`SkipTest`** en la importación se registran como saltos, no como errores.

start_dir puede ser un paquete de espacios de nombres.

Las rutas se ordenan antes de ser importadas para que el orden de ejecución sea el mismo, incluso si el orden del sistema de archivos subyacente no depende del nombre del archivo.

Distinto en la versión 3.5: Los paquetes encontrados son ahora comprobados para `load_tests` sin importar si su ruta coincide con el *pattern*, porque es imposible que el nombre de un paquete coincida con el patrón por defecto.

Distinto en la versión 3.11: *start_dir* can not be a namespace packages. It has been broken since Python 3.7, and Python 3.11 officially removes it.

Distinto en la versión 3.13: top_level_dir is only stored for the duration of *discover* call.

Distinto en la versión 3.14: start_dir can once again be a namespace package.

Los siguientes atributos de un **`TestLoader`** pueden ser configurados ya sea por subclasificación o asignación en una instancia:

testMethodPrefix

Cadena que da el prefijo de los nombres de métodos que serán interpretados como métodos de test. El valor por defecto es `'test'`.

This affects **`getTestCaseNames()`** and all the `loadTestsFrom*` methods.

sortTestMethodsUsing

Function to be used to compare method names when sorting them in **`getTestCaseNames()`** and all the `loadTestsFrom*` methods.

suiteClass

Objeto invocable que construye un conjunto de pruebas a partir de una lista de pruebas. No se necesitan métodos en el objeto resultante. El valor por defecto es la clase **`TestSuite`**.

This affects all the `loadTestsFrom*` methods.

testNamePatterns

Lista de patrones de nombres de test de comodines al estilo del shell de Unix que los métodos de test tienen que coincidir con para ser incluidos en las suites de test (ver opción `-k`).

Si este atributo no es `None` (el predeterminado), todos los métodos de test que se incluyan en los paquetes de test deben coincidir con uno de los patrones de esta lista. Tenga en cuenta que las coincidencias se realizan siempre utilizando **`fnmatch.fnmatchcase()`**, por lo que a diferencia de los patrones pasados a la opción `-k`, los patrones de subcadena simple tendrán que ser convertidos utilizando comodines `*`.

This affects all the `loadTestsFrom*` methods.

Added in version 3.7.

class unittest.TestResult

Esta clase se utiliza para recopilar información sobre qué tests han tenido éxito y cuáles han fracasado.

Un objeto **`TestResult`** almacena los resultados de una serie de pruebas. Las clases **`TestCase`** y **`TestSuite`** aseguran que los resultados se registren correctamente; los autores de los tests no tienen que preocuparse de registrar el resultado de las mismas.

Testing frameworks built on top of **`unittest`** may want access to the **`TestResult`** object generated by running a set of tests for reporting purposes; a **`TestResult`** instance is returned by the **`TestRunner.run()`** method for this purpose.

Las instancias de **`TestResult`** tienen los siguientes atributos que serán de interés cuando se inspeccionen los resultados de la ejecución de un conjunto de tests:

errors

Una lista que contiene tuplas de 2 elementos de instancias **`TestCase`** y cadenas con formato de tracebacks. Cada tupla representa una prueba que lanzó una excepción inesperada.

failures

A list containing 2-tuples of **`TestCase`** instances and strings holding formatted tracebacks. Each tuple represents a test where a failure was explicitly signalled using the assert* methods.

skipped

Una lista que contiene tuplas de 2 elementos de instancias de **`TestCase`** y cadenas que contienen la razón para saltarse el test.

Added in version 3.1.

expectedFailures

Una lista que contiene tuplas de 2 elementos de instancias **`TestCase`** y cadenas con formato de traceback. Cada tupla representa un fallo esperado del caso de test.

unexpectedSuccesses

Una lista que contiene instancias de **`TestCase`** que fueron marcadas como fracasos esperados, pero tuvieron éxito.

collectedDurations

Una lista que contiene tuplas de 2 elementos con nombres de casos de prueba y números flotantes que representan el tiempo transcurrido de cada prueba que se ejecutó.

Added in version 3.12.

shouldStop

Puesto en `True` cuando la ejecución de los tests se detenga por **`stop()`**.

testsRun

El número total de tests realizados hasta ahora.

buffer

Si se ajusta a true, `sys.stdout` y `sys.stderr` serán almacenados entre **`startTest()`** y **`stopTest()`** siendo llamados. La salida recolectada sólo tendrá eco en el verdadero `sys.stdout` y `sys.stderr` si la prueba falla o se equivoca. Cualquier salida también se adjunta al mensaje de fallo / error.

Added in version 3.2.

failfast

Si se ajusta a true **`stop()`** se llamará al primer fallo o error, deteniendo la ejecución de la prueba.

Added in version 3.2.

tb_locals

Si se ajusta a true entonces las variables locales se mostrarán en los tracebacks.

Added in version 3.5.

wasSuccessful()

Devuelve `True` si todas las pruebas realizadas hasta ahora han pasado, de lo contrario devuelve `False`.

Distinto en la versión 3.4: Devuelve `False` si hubo algún **`unexpectedSuccesses`** de las pruebas marcadas con el decorador **`expectedFailure()`**.

stop()

This method can be called to signal that the set of tests being run should be aborted by setting the **`shouldStop`** attribute to `True`. **`TestRunner`** objects should respect this flag and return without running any additional tests.

For example, this feature is used by the **`TextTestRunner`** class to stop the test framework when the user signals an interrupt from the keyboard. Interactive tools which provide **`TestRunner`** implementations can use this in a similar manner.

Los siguientes métodos de la clase **`TestResult`** se utilizan para mantener las estructuras de datos internos, y pueden ampliarse en subclases para apoyar los requisitos de información adicionales. Esto es particularmente útil para construir herramientas que apoyen la presentación de informes interactivos mientras se ejecutan las pruebas.

startTest(*test*)

Llamado cuando el caso de prueba *test* está a punto de ser ejecutado.

stopTest(*test*)

Llamado después de que el caso de prueba *prueba* haya sido ejecutado, independientemente del resultado.

startTestRun()

Llamado una vez antes de que se ejecute cualquier prueba.

Added in version 3.1.

stopTestRun()

Llamado una vez después de que se ejecuten todas las pruebas.

Added in version 3.1.

addError(*test*, *err*)

Llamado cuando el caso de prueba *test* plantea una excepción inesperada. *err* es una tupla de la forma devuelta por **`sys.exc_info()`**: `(type, value, traceback)`.

La implementación por defecto añade una tupla `(test, formatted_err)` al atributo **`errors`** de la instancia, donde *formatted_err* es una traza formateada derivada de *err*.

addFailure(*test*, *err*)

Llamado cuando el caso de prueba *test* señala un fallo. *err* es una tupla de la forma devuelta por **`sys.exc_info()`**: `(type, value, traceback)`.

La implementación por defecto añade una tupla `(test, formatted_err)` al atributo **`failures`** de la instancia, donde *formatted_err* es una traza formateada derivada de *err*.

addSuccess(*test*)

Llamado cuando el caso de prueba *test* tenga éxito.

La implementación por defecto no hace nada.

addSkip(*test*, *reason*)

Llamado cuando se salta el caso de prueba *test*. *reason* es la razón que la prueba dio para saltarse.

La implementación por defecto añade una tupla `(test, reason)` al atributo **`skipped`** de la instancia.

addExpectedFailure(*test*, *err*)

Llamó cuando el caso de prueba *test* falla, pero fue marcado con el decorador **`expectedFailure()`**.

La implementación por defecto añade una tupla `(test, formatted_err)` al atributo **`expectedFailures`** de la instancia, donde *formatted_err* es una traza formateada derivada de *err*.

addUnexpectedSuccess(*test*)

Llamó cuando el caso de prueba *test* se marcó con el decorador `expectedFailure()`, pero tuvo éxito.

La implementación por defecto añade la prueba al atributo `unexpectedSuccesses` de la instancia.

addSubTest(*test*, *subtest*, *outcome*)

Llamado cuando termina una subtest. *test* es el caso de prueba correspondiente al método de test. *subtest* es una instancia personalizada `TestCase` que describe el test.

Si *outcome* es `None`, el subtest tuvo éxito. De lo contrario, falló con una excepción en la que *outcome* es una tupla de la forma devuelta por `sys.exc_info()`: `(type, value, traceback)`.

La implementación por defecto no hace nada cuando el resultado es un éxito, y registra los fallos del subtest como fallos normales.

Added in version 3.4.

addDuration(*test*, *elapsed*)

Llamado cuando el caso de prueba finaliza. *elapsed* es el tiempo representado en segundos e incluye la ejecución de funciones de limpieza.

Added in version 3.12.

class unittest.TextTestResult(*stream*, *descriptions*, *verbosity*, *, *durations=None*)

Una implementación concreta de `TestResult` utilizada por el `TextTestRunner`. Las subclases deberían aceptar `**kwargs` para garantizar la compatibilidad a medida que cambia la interfaz.

Added in version 3.2.

Distinto en la versión 3.12: Added the *durations* keyword parameter.

unittest.defaultTestLoader

Instancia de la clase `TestLoader` destinada a ser compartida. Si no es necesario personalizar la clase **`TestLoader`**, esta instancia puede utilizarse en lugar de crear repetidamente nuevas instancias.

class unittest.TextTestRunner(*stream=None*, *descriptions=True*, *verbosity=1*, *failfast=False*, *buffer=False*, *resultclass=None*, *warnings=None*, ***, *tb_locals=False*, *durations=None*)

Una implementación básica del test runner que produce resultados en una corriente. Si *stream* es `None`, el valor por defecto, `sys.stderr` se utiliza como flujo de salida. Esta clase tiene unos pocos parámetros configurables, pero es esencialmente muy simple. Las aplicaciones gráficas que ejecutan las suites de prueba deben proporcionar implementaciones alternativas. Tales implementaciones deberían aceptar `**kwargs` como interfaz para construir los cambios de los corredores cuando se añaden características a unittest.

Por defecto este runner muestra `DeprecationWarning`, `PendingDeprecationWarning`, `ResourceWarning` y `ImportWarning` incluso si están ignorados por defecto. Este comportamiento se puede anular utilizando las opciones **`-Wd`** o **`-Wa`** de Python (consulte Control de advertencias) y dejando *warnings* en `None`.

Distinto en la versión 3.2: Se añadió el argumento *warnings*.

Distinto en la versión 3.2: El flujo por defecto está configurado como `sys.stderr` en tiempo de instanciación en lugar de tiempo de importación.

Distinto en la versión 3.5: Se añadió el parámetro *tb_locals*.

Distinto en la versión 3.12: Se añadió el parámetro *durations*.

_makeResult()

Este método devuelve la instancia de `TestResult` usada por `run()`. No está destinado a ser llamado directamente, pero puede ser anulado en subclases para proporcionar un `TestResult` personalizado.

`_makeResult()` instanciando la clase o el pasaje llamado en el constructor `TextTestRunner` como el argumento de `resultclass`. Por defecto es **`TextTestResult`** si no se proporciona ninguna `resultclass`. La clase de resultado se instanciará con los siguientes argumentos:

```
stream, descriptions, verbosity
```

run(*test*)

Este método es la principal interfaz pública del `TextTestRunner`. Este método toma una instancia **`TestSuite`** o **`TestCase`**. Se crea una **`TestResult`** llamando a **`_makeResult()`** y se ejecuta(n) la(s) prueba(s) y se imprimen los resultados a stdout.

unittest.main(*module='_main_', defaultTest=None, argv=None, testRunner=None, testLoader=unittest.defaultTestLoader, exit=True, verbosity=1, failfast=None, catchbreak=None, buffer=None, warnings=None*)

Un programa de línea de comandos que carga un conjunto de pruebas de *módulo* y las ejecuta; esto es principalmente para hacer los módulos de prueba convenientemente ejecutables. El uso más simple de esta función es incluir la siguiente línea al final de un guión de prueba:

```
if __name__ == '__main__':
    unittest.main()
```

Puedes hacer pruebas con información más detallada pasando el argumento de la verbosity:

```
if __name__ == '__main__':
    unittest.main(verbosity=2)
```

El argumento *defaultTest* es el nombre de una prueba única o un iterable de nombres de pruebas a ejecutar si no se especifican nombres de pruebas a través de *argv*. Si no se especifica o `Ninguno` y no se proporcionan nombres de pruebas vía *argv*, se ejecutan todas las pruebas encontradas en *modulo*.

El argumento *argv* puede ser una lista de opciones pasadas al programa, siendo el primer elemento el nombre del programa. Si no se especifica o `Ninguno`, se utilizan los valores de **`sys.argv`**.

The *testRunner* argument can either be a test runner class or an already created instance of it. By default `main` calls **`sys.exit()`** with an exit code indicating success (0) or failure (1) of the tests run. An exit code of 5 indicates that no tests were run or skipped.

El argumento *testLoader* tiene que ser una instancia **`TestLoader`**, y por defecto **`defaultTestLoader`**.

`main` apoya el uso del intérprete interactivo pasando el argumento `exit=False`. Esto muestra el resultado en la salida estándar sin llamar a **`sys.exit()`**:

```
>>> from unittest import main
>>> main(module='test_module', exit=False)
```

Los parámetros *failfast*, *catchbreak* y *buffer* tienen el mismo efecto que las command-line options del mismo nombre.

El argumento *warnings* especifica el filtro de aviso que debe ser usado mientras se realizan los tests. Si no se especifica, permanecerá como `None` si se pasa una opción **`-W`** a **python** (ver Warning control), de lo contrario se establecerá como `'default'`.

Calling `main` returns an object with the `result` attribute that contains the result of the tests run as a **`unittest.TestResult`**.

Distinto en la versión 3.1: El parámetro *exit* fue añadido.

Distinto en la versión 3.2: Los parámetros *verbosity*, *failfast*, *catchbreak*, *buffer* y *warnings* fueron añadidos.

Distinto en la versión 3.4: El parámetro *defaultTest* fue cambiado para aceptar también un iterable de nombres de pruebas.

load_tests protocolo

Added in version 3.2.

Los módulos o paquetes pueden personalizar la forma en que se cargan las pruebas a partir de ellos durante las ejecuciones de prueba normales o el descubrimiento de pruebas mediante la implementación de una función llamada `load_tests`.

Si un módulo de tests define `load_tests` será llamado por **`TestLoader.loadTestsFromModule()`** con los siguientes argumentos:

```
load_tests(loader, standard_tests, pattern)
```

donde *pattern* se pasa directamente desde `loadTestsFromModule`. Por defecto es `None`.

Debe retornar una **`TestSuite`**.

loader es la instancia de **`TestLoader`** haciendo la carga. *standard_tests* son los tests que se cargarían por defecto desde el módulo. Es común que los módulos de test sólo quieran añadir o quitar tests del conjunto de tests estándar. El tercer argumento se usa cuando se cargan paquetes como parte del descubrimiento de tests.

Una típica función de `load_tests` que carga pruebas de un conjunto específico de **`TestCase`class:`TestCase`** puede ser como:

```
test_cases = (TestCase1, TestCase2, TestCase3)

def load_tests(loader, tests, pattern):
    suite = TestSuite()
    for test_class in test_cases:
        tests = loader.loadTestsFromTestCase(test_class)
        suite.addTests(tests)
    return suite
```

Si discovery se inicia en un directorio que contiene un paquete, ya sea desde la línea de comandos o llamando a **`TestLoader.discover()`**, entonces el paquete `__init__.py` se comprobará por `load_tests`. Si esa función no existe, discover se reincorporará al paquete como si fuera un directorio más. De lo contrario, el descubrimiento de los tests del paquete se dejará en `load_tests` que se llama con los siguientes argumentos:

```
load_tests(loader, standard_tests, pattern)
```

Esto debería devolver un **`TestSuite`** que represente todas las pruebas del paquete. (`test_estándar` sólo contendrá las pruebas recogidas de `__init__.py`.)

Debido a que el patrón se pasa a `load_tests` el paquete es libre de continuar (y potencialmente modificar) el descubrimiento de pruebas. Una función de "no hace nada" `load_test` para un paquete de pruebas se vería como:

```
def load_tests(loader, standard_tests, pattern):
    # top level directory cached on loader instance
    this_dir = os.path.dirname(__file__)
    package_tests = loader.discover(start_dir=this_dir,
    standard_tests.addTests(package_tests)
    return standard_tests
```

Distinto en la versión 3.5: Discovery ya no comprueba si los nombres de los paquetes coinciden con el *patrón* debido a la imposibilidad de que los nombres de los paquetes coincidan con el patrón por defecto.

Instalaciones para clases y módulos

Class and module level fixtures are implemented in **`TestSuite`**.

When the test suite encounters a test from a new class then **`tearDownClass()`** from the previous class (if there is one) is called, followed by **`setUpClass()`** from the new class.

Del mismo modo, si una prueba es de un módulo diferente de la prueba anterior, entonces se ejecuta `DesmontarMódulo` del módulo anterior, seguido de `DesmontarMódulo` del nuevo módulo.

Después de todas las pruebas, se ejecutan los últimos `tearDownClass` y `tearDownModule`.

Tenga en cuenta que los accesorios compartidos no juegan bien con las características [potenciales] como la paralelización de la prueba y rompen el aislamiento de la prueba. Deben ser usados con cuidado.

El orden por defecto de las pruebas creadas por los cargadores de pruebas unitarias es agrupar todas las pruebas de los mismos módulos y clases. Esto llevará a que `setUpClass` / `setUpModule` (etc) sea llamado exactamente una vez por clase y módulo. Si se aleatoriza el orden, de manera que las pruebas de diferentes módulos y clases sean adyacentes entre sí, entonces estas funciones compartidas de fixture pueden ser llamadas varias veces en una sola ejecución de prueba.

Los accesorios compartidos no están pensados para trabajar con suites con pedidos no estándar. Todavía existe una `BaseTestSuite` para los marcos de trabajo que no quieren soportar accesorios compartidos.

Si hay alguna excepción planteada durante una de las funciones compartidas del aparato, la prueba se notifica como un error. Debido a que no hay una instancia de prueba correspondiente, se crea un objeto `_ErrorHolder` (que tiene la misma interfaz que un **`TestCase`**) para representar el error. Si sólo estás usando el standard unittest test runner entonces este detalle no importa, pero si eres un autor de marcos de trabajo puede ser relevante.

setUpClass y tearDownClass

Estos deben ser implementados como métodos de clase:

```
import unittest

class Test(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls._connection = createExpensiveConnectionObjec

    @classmethod
    def tearDownClass(cls):
        cls._connection.destroy()
```

Si quieres que se invoque a `SetUpClass` y `BreakdownClass` en clases base, debes llamarlos tú mismo. Las implementaciones en **`TestCase`** están vacías.

Si se lanza una excepción durante una `setUpClass`, entonces los tests de la clase no se ejecutan y la `tearDownClass` no se ejecuta. Las clases que se salten no tendrán `setUpClass` o `tearDownClass`. Si la excepción es una **`SkipTest`** entonces la clase será reportada como salteada en lugar de como un error.

setUpModule y tearDownModule

Estos deben ser implementados como funciones:

```
def setUpModule():
    createConnection()

def tearDownModule():
    closeConnection()
```

Si se lanza una excepción en un `setUpModule`, entonces no se ejecutará ninguna de las pruebas del módulo y no se ejecutará el `tearDownModule`. Si la excepción es una **`SkipTest`** entonces el módulo será reportado como saltado en lugar de como un error.

Para agregar código de limpieza que se debe ejecutar incluso en el caso de una excepción, utilice `addModuleCleanup`:

unittest.addModuleCleanup(*function*, /, **args*, ***kwargs*)

Añade una función que se llamará después de `tearDownModule()` para limpiar los recursos utilizados durante la clase de test. Las funciones se llamarán en orden inverso al orden en que se agregan (LIFO). Se llaman con cualquier argumento y palabra clave que se pase a `addModuleCleanup()` cuando se añadan.

Si `setUpModule()` falla, lo que significa que `tearDownModule()` no se invoca, entonces cualquier función de limpieza añadida seguirá siendo invocada.

Added in version 3.8.

unittest.enterModuleContext(*cm*)

Introduce el context manager suministrado. Si es exitoso, añade también su método `__exit__()` como función de limpieza mediante `addModuleCleanup()` y retorna el resultado del método `__enter__()`.

Added in version 3.11.

unittest.doModuleCleanups()

Esta función se llama incondicionalmente después de `tearDownModule()`, o después de `setUpModule()` si **`setUpModule()`** lanza una excepción.

Es responsable de invocar a todas las funciones de limpieza añadidas por **`addCleanupModule()`**. Si necesitas que las funciones de limpieza se llamen *previamente* a `tearDownModule()` entonces puedes invocar a `doModuleCleanups()` tú mismo.

`doModuleCleanups()` saca los métodos de la pila de funciones de limpieza uno a uno, así que se puede llamar en cualquier momento.

Added in version 3.8.

Manejo de señales

Added in version 3.2.

The `-c/--catch` command-line option to unittest, along with the `catchbreak` parameter to `unittest.main()`, provide more friendly handling of control-C during a test run. With catch break behavior enabled control-C will allow the currently running test to complete, and the test run will then end and report all the results so far. A second control-c will raise a `KeyboardInterrupt` in the usual way.

El manejador de señales de manejo de control-c intenta permanecer compatible con el código o las pruebas que instalan su propio manejador `signal.SIGINT`. Si se llama al manejador `unittest` pero *no es* el manejador `signal.SIGINT` instalado, es decir, ha sido reemplazado por el sistema bajo test y delegado, entonces llama al manejador por defecto. Este será normalmente el comportamiento esperado por el código que reemplaza un manejador instalado y delega en él. Para las pruebas individuales que necesiten el manejo de control-c de `unittest` deshabilitado se puede usar el decorador `removeHandler()`.

Hay algunas funciones de utilidad para que los autores de marcos de trabajo habiliten la funcionalidad de control de control-c dentro de los marcos de prueba.

unittest.installHandler()

> Instala el controlador de control-c. Cuando se recibe una `signal.SIGINT` (normalmente en respuesta a que el usuario presione control-c) todos los resultados registrados tienen `stop()` llamado.

unittest.registerResult(*result*)

> Registrar un objeto `TestResult` para el manejo de control-c. El registro de un resultado almacena una referencia débil a él, por lo que no evita que el resultado sea recogido por el recolector de basura.

El registro de un objeto `TestResult` no tiene efectos secundarios si el manejo de control-c no está habilitado, por lo que los marcos de pruebas pueden registrar incondicionalmente todos los resultados que crean independientemente de si el manejo está habilitado o no.

unittest.removeResult(*result*)

Elimine un resultado registrado. Una vez que un resultado ha sido eliminado, `stop()` ya no se llamará en ese objeto de resultado en respuesta a un control-c.

unittest.removeHandler(*function=None*)

Cuando se llama sin argumentos, esta función quita el gestor control-c si se ha instalado. Esta función también se puede utilizar como decorador de tests para quitar temporalmente el controlador mientras se ejecuta el test:

```
@unittest.removeHandler
def test_signal_handling(self):
    ...
```

[1] [2] [3][4]

`unittest.mock` — mock object library

Added in version 3.3.

Source code: Lib/unittest/mock.py [https://github.com/python/cpython/tree/3.14/Lib/unittest/mock.py]

`unittest.mock` is a library for testing in Python. It allows you to replace parts of your system under test with mock objects and make assertions about how they have been used.

`unittest.mock` provides a core **`Mock`** class removing the need to create a host of stubs throughout your test suite. After performing an action, you can make assertions about which methods / attributes were used and arguments they were called with. You can also specify return values and set needed attributes in the normal way.

Además, mock proporciona un decorador **`patch()`** que puede manejar el parcheo de atributos a nivel de clase y de módulo dentro del ámbito de una prueba, junto con **`sentinel`** para crear objetos únicos. Consulta quick guide para ver algunos ejemplos de cómo utilizar **`Mock`**, **`MagicMock`** y **`patch()`**.

Mock está diseñado para ser utilizado junto a **`unittest`**. Mock se basa en el patrón "acción -> aserción" en lugar de usar el patrón "grabación -> reproducción" utilizado por muchos frameworks de simulación.

There is a backport of **`unittest.mock`** for earlier versions of Python, available as mock [https://pypi.org/project/mock/] on PyPI.

Guía rápida

Los objetos de las clases `Mock` y `MagicMock` van creando todos los atributos y métodos a medida que se accede a ellos y almacenan detalles de cómo se han utilizado. Puedes configurarlos para especificar valores de retorno o limitar qué atributos están disponibles y posteriormente hacer aserciones sobre cómo han sido utilizados:

```
>>> from unittest.mock import MagicMock
>>> thing = ProductionClass()
>>> thing.method = MagicMock(return_value=3)
>>> thing.method(3, 4, 5, key='value')
3
>>> thing.method.assert_called_with(3, 4, 5, key='value'
```

`side_effect` allows you to perform side effects, including raising an exception when a mock is called:

```
>>> from unittest.mock import Mock
>>> mock = Mock(side_effect=KeyError('foo'))
>>> mock()
Traceback (most recent call last):
  ...
KeyError: 'foo'

>>> values = {'a': 1, 'b': 2, 'c': 3}
>>> def side_effect(arg):
...     return values[arg]
...
>>> mock.side_effect = side_effect
>>> mock('a'), mock('b'), mock('c')
(1, 2, 3)
>>> mock.side_effect = [5, 4, 3, 2, 1]
>>> mock(), mock(), mock()
(5, 4, 3)
```

Existen muchas otras formas de configurar y controlar el comportamiento de Mock. Por ejemplo, el argumento *spec* configura el objeto simulado para que tome su especificación de otro objeto. Cualquier intento de acceder a atributos o métodos en el objeto simulado que no existan en la especificación fallará lanzando una excepción **`AttributeError`**.

El decorador / gestor de contexto **`patch()`** facilita la simulación de clases u objetos en un módulo bajo prueba. El objeto que especifiques será reemplazado por un objeto simulado (u otro objeto) durante la prueba y será restaurado cuando esta finalice:

```
>>> from unittest.mock import patch
>>> @patch('module.ClassName2')
... @patch('module.ClassName1')
... def test(MockClass1, MockClass2):
...     module.ClassName1()
...     module.ClassName2()
...     assert MockClass1 is module.ClassName1
...     assert MockClass2 is module.ClassName2
...     assert MockClass1.called
...     assert MockClass2.called
...
>>> test()
```

Nota

Cuando anidas decoradores patch, los objetos simulados se pasan a la función decorada en el mismo orden en el que fueron aplicados (el orden normal en el que se aplican los decoradores en *Python*). Esto significa de abajo hacia arriba, por lo que en el ejemplo anterior se pasa primero el objeto simulado para `module.ClassName1`.

Al usar **`patch()`** es importante que parchees los objetos en el espacio de nombres donde son buscados. Esto normalmente es sencillo, pero para una guía rápida, lee dónde parchear.

Además de decorador, la función **`patch()`** se puede usar como gestor de contexto en una declaración with:

```
>>> with patch.object(ProductionClass, 'method', return_
...     thing = ProductionClass()
...     thing.method(1, 2, 3)
```

```
...
>>> mock_method.assert_called_once_with(1, 2, 3)
```

También existe la función **`patch.dict()`** que permite establecer valores en un diccionario dentro de un ámbito y restaurar el diccionario a su estado original cuando finaliza la prueba:

```
>>> foo = {'key': 'value'}
>>> original = foo.copy()
>>> with patch.dict(foo, {'newkey': 'newvalue'}, clear=T
...     assert foo == {'newkey': 'newvalue'}
...
>>> assert foo == original
```

Mock admite la simulación de los métodos mágicos de Python. La forma más sencilla de utilizar métodos mágicos es mediante la clase **`MagicMock`**. Te permite hacer cosas como:

```
>>> mock = MagicMock()
>>> mock.__str__.return_value = 'foobarbaz'
>>> str(mock)
'foobarbaz'
>>> mock.__str__.assert_called_with()
```

Mock también permite asignar funciones (u otras instancias de Mock) a métodos mágicos y se asegura de que serán llamadas de forma apropiada. La clase **`MagicMock`** es solo una variante de Mock con la diferencia de que tiene todos los métodos mágicos previamente creados para ti (bueno, todos los que son útiles).

El siguiente es un ejemplo de uso de métodos mágicos utilizando la clase Mock ordinaria:

```
>>> mock = Mock()
>>> mock.__str__ = Mock(return_value='wheeeeee')
>>> str(mock)
'wheeeeee'
```

Para asegurarte de que los objetos simulados en tus pruebas tienen exactamente la misma API que los objetos que están reemplazando, puedes usar autoespecificación. La autoespecificación se puede realizar a través del argumento *autospec* de patch, o mediante la función `create_autospec()`. La autoespecificación crea objetos simulados que tienen los mismos atributos y métodos que los objetos que están reemplazando, y todas las función y métodos (incluidos los constructores) tienen la misma firma de llamada que los objetos reales.

Esto asegura que tus simulaciones fallarán, si se utilizan incorrectamente, de la misma manera que lo haría tu código en producción:

```
>>> from unittest.mock import create_autospec
>>> def function(a, b, c):
...     pass
...
>>> mock_function = create_autospec(function, return_val
>>> mock_function(1, 2, 3)
'fishy'
>>> mock_function.assert_called_once_with(1, 2, 3)
>>> mock_function('wrong arguments')
Traceback (most recent call last):
 ...
TypeError: missing a required argument: 'b'
```

`create_autospec()` también se puede usar en clases, donde copia la firma del método `__init__`, y en objetos invocables, donde copia la firma del método `__call__`.

La clase Mock

`Mock` es un objeto simulado flexible, destinado a reemplazar el uso de stubs y dobles de prueba en todo tu código. Los objetos Mock son invocables y crean atributos en el mismo momento que se accede a ellos como nuevos objetos Mock [1]. Acceder al mismo atributo siempre retornará el mismo objeto Mock. Además, registran cómo los usas, lo que te permite hacer aserciones sobre cómo tu código ha interaccionado con ellos.

`MagicMock` es una subclase de `Mock` con todos los métodos mágicos creados previamente y listos para ser usados. También hay variantes no invocables, útiles cuando se están simulando objetos que no se pueden llamar: `NonCallableMock` y `NonCallableMagicMock`

Los decoradores `patch()` facilitan la sustitución temporal de clases en un módulo en particular con un objeto `Mock`. Por defecto, **`patch()`** creará un objeto `MagicMock` automáticamente. Se puede especificar una clase alternativa a **`Mock`** usando el argumento *new_callable* de **`patch()`**.

class unittest.mock.Mock(*spec=None*, *side_effect=None*, *return_value=DEFAULT*, *wraps=None*, *name=None*, *spec_set=None*, *unsafe=False*, ***kwargs*)

> Crea un nuevo objeto `Mock`. **`Mock`** toma varios argumentos opcionales que especifican el comportamiento del objeto Mock:
>
> - *spec*: Puede ser una lista de cadenas de caracteres o un objeto existente previamente (una clase o una instancia) que actúa como la especificación del objeto simulado. Si pasas un objeto, se forma una lista de cadenas llamando a la función *dir* en el objeto (excluyendo los métodos y atributos mágicos no admitidos). Acceder a cualquier atributo que no esté en esta lista lanzará una excepción `AttributeError`.
>
>   If *spec* is an object (rather than a list of strings) then `__class__` returns the class of the spec object. This allows mocks to pass `isinstance()` tests.
>
> - *spec_set*: Una variante más estricta de *spec*. Si se utiliza, cualquier intento de *establecer* u obtener un atributo del objeto simulado que no esté en el objeto pasado como *spec_set* lanzará una excepción `AttributeError`.
>
> - *side_effect*: Una función que se llamará cada vez que el objeto simulado sea invocado. Consultar el atributo `side_effect` para más información. Es útil para lanzar excepciones o para cambiar dinámicamente valores de retorno. La función se llama con los mismos

argumentos que el objeto simulado, y a menos que retorne **`DEFAULT`**, su valor de retorno se utiliza como valor de retorno del propio objeto simulado.

Alternativamente *side_effect* puede ser una clase o instancia de excepción. En este caso, se lanza la excepción cuando se llama al objeto simulado.

Si *side_effect* es un iterable, cada llamada al objeto simulado retornará el siguiente valor del iterable.

Un *side_effect* se puede desactivar estableciéndolo en `None`.

- *return_value*: El valor retornado cuando se llama al objeto simulado. Por defecto, este es una nueva instancia de la clase Mock (creada en el primer acceso). Consultar el atributo **`return_value`** para más detalles.

- *unsafe*: Por defecto, el acceso a cualquier atributo cuyo nombre comience por *assert*, *assret*, *asert*, *aseert* o *assrt* generará un error **`AttributeError`**. Si se pasa `unsafe=True` se permitirá el acceso a estos atributos.

  Added in version 3.5.

- *wraps*: objeto a envolver (simular) por la instancia de Mock. Si *wraps* no es `None`, al llamar al objeto Mock se pasa la llamada a través del objeto envuelto (retornando el resultado real). Acceder a un atributo del objeto simulado retornará otro objeto Mock que envuelve al atributo correspondiente del objeto real envuelto (de modo que intentar acceder a un atributo que no existe lanzará una excepción **`AttributeError`**).

  Si el objeto simulado tiene un *return_value* explícito establecido, las llamadas no se pasan al objeto envuelto y *return_value* se retorna en su lugar.

- *name*: Si el objeto simulado tiene un nombre, será utilizado en la representación imprimible del mismo. Esto puede ser útil para la depuración. El nombre se propaga a los objetos simulados hijos.

Los objetos simulados también pueden ser invocados con argumentos por palabra clave arbitrarios. Estos serán utilizados para establecer atributos en el objeto simulado una vez creado. Consultar el método **`configure_mock()`** para más detalles.

assert_called()

Assert cuando el objeto simulado se ha invocado al menos una vez.

```
>>> mock = Mock()
>>> mock.method()
<Mock name='mock.method()' id='...'>
>>> mock.method.assert_called()
```

Added in version 3.6.

assert_called_once()

Assert si el objeto simulado se ha invocado exactamente una vez.

```
>>> mock = Mock()
>>> mock.method()
<Mock name='mock.method()' id='...'>
>>> mock.method.assert_called_once()
>>> mock.method()
<Mock name='mock.method()' id='...'>
>>> mock.method.assert_called_once()
Traceback (most recent call last):
...
AssertionError: Expected 'method' to have been
Calls: [call(), call()].
```

Added in version 3.6.

assert_called_with(*args*, ***kwargs*)

Este método es una manera apropiada de asertar si la última llamada se ha realizado de una manera particular:

```
>>> mock = Mock()
>>> mock.method(1, 2, 3, test='wow')
<Mock name='mock.method()' id='...'>
>>> mock.method.assert_called_with(1, 2, 3, tes
```

assert_called_once_with(*args*, ***kwargs*)

Assert si el objeto simulado se ha invocado exactamente una vez y si esa llamada se realizó con los argumentos especificados.

```
>>> mock = Mock(return_value=None)
>>> mock('foo', bar='baz')
>>> mock.assert_called_once_with('foo', bar='ba
>>> mock('other', bar='values')
>>> mock.assert_called_once_with('other', bar='
Traceback (most recent call last):
  ...
AssertionError: Expected 'mock' to be called or
Calls: [call('foo', bar='baz'), call('other', b
```

assert_any_call(*args*, ***kwargs*)

assert si el objeto simulado se ha invocado con los argumentos especificados.

La aserción pasa si el objeto simulado se ha invocado *en algún momento*, a diferencia de **`assert_called_with()`** y **`assert_called_once_with()`**, con los que solo pasa la aserción si la llamada es la más reciente, y en el caso de **`assert_called_once_with()`** también debe ser la única llamada realizada.

```
>>> mock = Mock(return_value=None)
>>> mock(1, 2, arg='thing')
>>> mock('some', 'thing', 'else')
>>> mock.assert_any_call(1, 2, arg='thing')
```

assert_has_calls(*calls*, *any_order=False*)

assert si el objeto simulado se ha invocado con las llamadas especificadas. La lista **`mock_calls`** se compara con la lista de llamadas.

Si *any_order* es falso entonces las llamadas deben ser secuenciales. No puede haber llamadas adicionales antes o después de las llamadas especificadas.

Si *any_order* es verdadero, las llamadas pueden estar en cualquier orden, pero deben aparecer todas en **`mock_calls`**.

```
>>> mock = Mock(return_value=None)
>>> mock(1)
>>> mock(2)
>>> mock(3)
>>> mock(4)
>>> calls = [call(2), call(3)]
>>> mock.assert_has_calls(calls)
>>> calls = [call(4), call(2), call(3)]
>>> mock.assert_has_calls(calls, any_order=Tru
```

assert_not_called()

Assert si el objeto simulado nunca fue invocado.

```
>>> m = Mock()
>>> m.hello.assert_not_called()
>>> obj = m.hello()
>>> m.hello.assert_not_called()
Traceback (most recent call last):
  ...
AssertionError: Expected 'hello' to not have be
Calls: [call()].
```

Added in version 3.5.

reset_mock(*, *return_value=False*, *side_effect=False*)

El método *reset_mock* restablece todos los atributos de llamada en un objeto simulado:

```
>>> mock = Mock(return_value=None)
>>> mock('hello')
>>> mock.called
True
>>> mock.reset_mock()
>>> mock.called
False
```

This can be useful where you want to make a series of assertions that reuse the same object.

return_value parameter when set to `True` resets **`return_value`**:

```
>>> mock = Mock(return_value=5)
>>> mock('hello')
5
>>> mock.reset_mock(return_value=True)
>>> mock('hello')
<Mock name='mock()' id='...'>
```

side_effect parameter when set to `True` resets **`side_effect`**:

```
>>> mock = Mock(side_effect=ValueError)
>>> mock('hello')
Traceback (most recent call last):
  ...
ValueError
>>> mock.reset_mock(side_effect=True)
>>> mock('hello')
<Mock name='mock()' id='...'>
```

Note that **`reset_mock()`** *doesn't* clear the `return_value`, `side_effect` or any child attributes you have set using normal assignment by default.

Child mocks are reset as well.

Distinto en la versión 3.6: Se añadieron dos argumentos de palabra clave a la función *reset_mock*.

mock_add_spec(*spec*, *spec_set=False*)

Agrega una especificación a un objeto simulado. *spec* puede ser un objeto o una lista de cadenas de caracteres. Solo los atributos presentes en *spec* pueden ser obtenidos desde el objeto simulado.

Si *spec_set* es verdadero, solo los atributos de la especificación pueden ser establecidos.

attach_mock(*mock*, *attribute*)

Adjunta otro objeto simulado como un atributo de la instancia actual, substituyendo su nombre y su padre. Las llamadas al objeto simulado adjuntado se registrarán en los atributos `method_calls` y `mock_calls` del padre.

configure_mock(***kwargs*)

Establece los atributos del objeto simulado por medio de argumentos por palabra clave.

Los atributos, los valores de retorno y los efectos de colaterales se pueden configurar en los objetos simulados hijos usando la notación de punto estándar y desempaquetando un diccionario en la llamada al método:

```
>>> mock = Mock()
>>> attrs = {'method.return_value': 3, 'other.s
>>> mock.configure_mock(**attrs)
>>> mock.method()
3
```

```
>>> mock.other()
Traceback (most recent call last):
  ...
KeyError
```

Lo mismo se puede lograr en la llamada al constructor de los objetos simulados:

```
>>> attrs = {'method.return_value': 3, 'other.s
>>> mock = Mock(some_attribute='eggs', **attrs)
>>> mock.some_attribute
'eggs'
>>> mock.method()
3
>>> mock.other()
Traceback (most recent call last):
  ...
KeyError
```

`configure_mock()` existe con el fin de facilitar la configuración después de que el objeto simulado haya sido creado.

dir()

Los objetos **`Mock`** limitan los resultados de `dir(some_mock)` a resultados útiles. Para los objetos simulados con una especificación (*spec*), esto incluye todos los atributos permitidos para el mismo.

Consultar **`FILTER_DIR`** para conocer que hace este filtrado y la forma de desactivarlo.

_get_child_mock(**kw)

Crea los objetos simulados hijos para los atributos y el valor de retorno. Por defecto los objetos simulados hijos serán del mismo tipo que el padre. Las subclases de Mock pueden redefinir este método para personalizar la forma en la que se construye el objeto simulado hijo.

Para objetos simulados no invocables la variante invocable será utilizada (en lugar de cualquier subclase personalizada).

called

Un booleano que representa si el objeto simulado ha sido invocado o no:

```
>>> mock = Mock(return_value=None)
>>> mock.called
False
>>> mock()
>>> mock.called
True
```

call_count

Un entero que le indica cuántas veces el objeto simulado ha sido invocado:

```
>>> mock = Mock(return_value=None)
>>> mock.call_count
0
>>> mock()
>>> mock()
>>> mock.call_count
2
```

return_value

Establece este atributo para configurar el valor a retornar cuando se llama al objeto simulado:

```
>>> mock = Mock()
>>> mock.return_value = 'fish'
>>> mock()
'fish'
```

El valor de retorno por defecto es otro objeto simulado y se puede configurar de forma habitual:

```
>>> mock = Mock()
>>> mock.return_value.attribute = sentinel.Attr
>>> mock.return_value()
<Mock name='mock()()' id='...'>
>>> mock.return_value.assert_called_with()
```

`return_value` también se puede establecer directamente en el constructor:

```
>>> mock = Mock(return_value=3)
>>> mock.return_value
3
>>> mock()
3
```

side_effect

Este atributo puede ser una función a ser llamada cuando se llame al objeto simulado, un iterable o una excepción (clase o instancia) para ser lanzada.

Si pasas una función, será llamada con los mismos argumentos que el objeto simulado y, a menos que la función retorne el singleton `DEFAULT`, la llamada al objeto simulado retornará lo mismo que retorna la función. En cambio, si la función retorna `DEFAULT`, entonces el objeto simulado retornará su valor normal (el del atributo `return_value`).

Si pasas un iterable, se utiliza para obtener un iterador a partir del mismo que debe producir un valor en cada llamada. Este valor puede ser una instancia de la excepción a ser lanzada o un valor a retornar al llamar al objeto simulado (el manejo de `DEFAULT` es igual que en el caso en el que se pasa una función).

Un ejemplo de un objeto simulado que genera una excepción (para probar el manejo de excepciones de una API):

```
>>> mock = Mock()
>>> mock.side_effect = Exception('Boom!')
>>> mock()
Traceback (most recent call last):
  ...
Exception: Boom!
```

Usando **`side_effect`** para retornar una secuencia de valores:

```
>>> mock = Mock()
>>> mock.side_effect = [3, 2, 1]
>>> mock(), mock(), mock()
(3, 2, 1)
```

Usando un objeto invocable:

```
>>> mock = Mock(return_value=3)
>>> def side_effect(*args, **kwargs):
...     return DEFAULT
...
>>> mock.side_effect = side_effect
>>> mock()
3
```

`side_effect` se puede establecer en el constructor. Aquí hay un ejemplo que suma uno al valor del objeto simulado invocado y lo retorna:

```
>>> side_effect = lambda value: value + 1
>>> mock = Mock(side_effect=side_effect)
>>> mock(3)
4
>>> mock(-8)
-7
```

Establecer **`side_effect`** en `None` lo desactiva:

```
>>> m = Mock(side_effect=KeyError, return_value
>>> m()
Traceback (most recent call last):
  ...
KeyError
>>> m.side_effect = None
>>> m()
3
```

call_args

This is either `None` (if the mock hasn't been called), or the arguments that the mock was last called with. This will be in the form of a tuple: the first member, which can also be accessed through the `args` property, is any positional arguments the mock was called with (or an empty tuple) and the second member, which can also be accessed through the `kwargs` property, is any keyword arguments (or an empty dictionary).

```
>>> mock = Mock(return_value=None)
>>> print(mock.call_args)
None
>>> mock()
>>> mock.call_args
call()
>>> mock.call_args == ()
True
>>> mock(3, 4)
>>> mock.call_args
call(3, 4)
>>> mock.call_args == ((3, 4),)
True
>>> mock.call_args.args
(3, 4)
>>> mock.call_args.kwargs
{}
>>> mock(3, 4, 5, key='fish', next='w00t!')
>>> mock.call_args
call(3, 4, 5, key='fish', next='w00t!')
>>> mock.call_args.args
(3, 4, 5)
>>> mock.call_args.kwargs
```

```
{'key': 'fish', 'next': 'w00t!'}
```

El atributo **`call_args`**, junto con los miembros de las listas **`call_args_list`**, **`method_calls`** y **`mock_calls`** son objetos **`call`**. Estos objetos son tuplas, con la finalidad de que puedan ser desempaquetadas para acceder a los argumentos individuales y hacer aserciones más complejas. Consultar objetos call como tuplas para más información.

Distinto en la versión 3.8: Propiedades `args` y `kwargs` agregadas.

call_args_list

Este argumento es una lista de todas las llamadas consecutivas realizadas al objeto simulado (por lo que la longitud de la lista es el número de veces que se ha invocado). Previamente a que se hayan realizado llamadas es una lista vacía. El objeto **`call`** se puede utilizar para construir convenientemente las listas de llamadas a comparar con **`call_args_list`**.

```
>>> mock = Mock(return_value=None)
>>> mock()
>>> mock(3, 4)
>>> mock(key='fish', next='w00t!')
>>> mock.call_args_list
[call(), call(3, 4), call(key='fish', next='w0
>>> expected = [(), ((3, 4),), ({'key': 'fish',
>>> mock.call_args_list == expected
True
```

Los miembros de **`call_args_list`** son objetos **`call`**. Estos pueden ser desempaquetados como tuplas para acceder a los argumentos individuales. Consultar objetos call como tuplas para más información.

method_calls

Igual que realizan un seguimiento de las llamadas hechas a sí mismos, los objetos simulados también realizan un seguimiento a *sus* métodos y atributos, así como de las llamadas hechas a los mismos:

```
>>> mock = Mock()
>>> mock.method()
<Mock name='mock.method()' id='...'>
>>> mock.property.method.attribute()
<Mock name='mock.property.method.attribute()' i
>>> mock.method_calls
[call.method(), call.property.method.attribute(
```

Los miembros de **`method_calls`** son objetos **`call`**. Estos pueden ser desempaquetados como tuplas para acceder a los atributos individuales. Consultar objetos call como tuplas para más información.

mock_calls

`mock_calls` registra *todas* las llamadas al objeto simulado, sus métodos, métodos mágicos *y* objetos simulados del valor de retorno.

```
>>> mock = MagicMock()
>>> result = mock(1, 2, 3)
>>> mock.first(a=3)
<MagicMock name='mock.first()' id='...'>
>>> mock.second()
<MagicMock name='mock.second()' id='...'>
>>> int(mock)
1
>>> result(1)
<MagicMock name='mock()()' id='...'>
>>> expected = [call(1, 2, 3), call.first(a=3),
... call.__int__(), call()(1)]
>>> mock.mock_calls == expected
True
```

Los miembros de **`mock_calls`** son objetos **`call`**.

Estos pueden ser desempaquetados como tuplas para acceder a los argumentos individuales. Consultar objetos call como tuplas para más información.

Nota

La forma como se registra el atributo **`mock_calls`** implica que cuando se realizan llamadas anidadas, los parámetros de las llamadas previas no se registran, por lo que siempre resultan iguales al comparar:

```
>>> mock = MagicMock()
>>> mock.top(a=3).bottom()
<MagicMock name='mock.top().bottom()' id='...
>>> mock.mock_calls
[call.top(a=3), call.top().bottom()]
>>> mock.mock_calls[-1] == call.top(a=-1).bott
True
```

class

Normally the **`__class__`** attribute of an object will return its type. For a mock object with a **`spec`**, **`__class__`** returns the spec class instead. This allows mock objects to pass **`isinstance()`** tests for the object they are replacing / masquerading as:

```
>>> mock = Mock(spec=3)
>>> isinstance(mock, int)
True
```

`__class__` is assignable to, this allows a mock to pass an **`isinstance()`** check without forcing you to use a spec:

```
>>> mock = Mock()
>>> mock.__class__ = dict
>>> isinstance(mock, dict)
True
```

class unittest.mock.NonCallableMock(*spec=None*, *wraps=None*, *name=None*, *spec_set=None*, ***kwargs*)

> Una versión no invocable de `Mock`. Los parámetros del constructor tienen el mismo significado que en **`Mock`**, con la excepción de *return_value* y *side_effect* que no tienen sentido en un objeto simulado no invocable.

Mock objects that use a class or an instance as a **`spec`** or **`spec_set`** are able to pass `isinstance()` tests:

```
>>> mock = Mock(spec=SomeClass)
>>> isinstance(mock, SomeClass)
True
>>> mock = Mock(spec_set=SomeClass())
>>> isinstance(mock, SomeClass)
True
```

Las clases `Mock` tienen soporte para simular los métodos mágicos. Consultar la sección dedicada a los métodos mágicos para ver los detalles.

Las clases simuladas y los decoradores `patch()` aceptan todas argumentos por palabra clave arbitrarios para la configuración. En los decoradores **`patch()`** los argumentos por palabra clave se pasan al constructor del objeto simulado creado. Estos argumentos se usan para configurar los atributos del propio objeto simulado:

```
>>> m = MagicMock(attribute=3, other='fish')
>>> m.attribute
3
>>> m.other
'fish'
```

Tanto el valor de retorno como el efecto colateral del objeto simulado pueden ser establecidos de la misma manera, mediante notación de punto. En cambio, si se quieren establecer mediante el constructor, dado que no se puede utilizar notación de punto directamente en una llamada, se tiene que crear un diccionario y desempaquetarlo usando `**`:

```
>>> attrs = {'method.return_value': 3, 'other.side_effec
>>> mock = Mock(some_attribute='eggs', **attrs)
>>> mock.some_attribute
'eggs'
>>> mock.method()
3
>>> mock.other()
Traceback (most recent call last):
  ...
KeyError
```

Un objeto simulado invocable que fue creado con un *spec* (o un *spec_set*) introspeccionará la firma del objeto de la especificación en el momento de emparejar las llamadas al objeto simulado. Esto le permite hacer coincidir sus argumentos con los argumentos de la llamada real, independientemente de si se pasaron por posición o por nombre:

```
>>> def f(a, b, c): pass
...
>>> mock = Mock(spec=f)
>>> mock(1, 2, c=3)
<Mock name='mock()' id='140161580456576'>
>>> mock.assert_called_with(1, 2, 3)
>>> mock.assert_called_with(a=1, b=2, c=3)
```

Esto se aplica a **`assert_called_with()`**, **`assert_called_once_with()`**, **`assert_has_calls()`** y **`assert_any_call()`**. Cuando se hace uso de Autoespecificación, también se aplicará a las llamadas a los métodos del objeto simulado.

Distinto en la versión 3.4: Se añadió introspección de firma en objetos simulados especificados y autoespecificados.

class unittest.mock.PropertyMock(**args*, ***kwargs*)

A mock intended to be used as a **`property`**, or other descriptor, on a class. **`PropertyMock`** provides **`__get__()`** and **`__set__()`** methods so you can specify a return value when it is fetched.

La obtención de una instancia `PropertyMock` desde un objeto ocasiona una llamada al objeto simulado, sin argumentos. Establecer su valor también llama al objeto simulado, con el valor a establecer como argumento.

```
>>> class Foo:
...     @property
...     def foo(self):
...         return 'something'
...     @foo.setter
...     def foo(self, value):
...         pass
...
>>> with patch('__main__.Foo.foo', new_callable=Pro
...     mock_foo.return_value = 'mockity-mock'
...     this_foo = Foo()
...     print(this_foo.foo)
...     this_foo.foo = 6
...
mockity-mock
>>> mock_foo.mock_calls
[call(), call(6)]
```

Debido a la forma en que se almacenan los atributos simulados, no es posible conectar directamente un `PropertyMock` a un objeto simulado. En su lugar se puede conectar al tipo (type) del objeto simulado:

```
>>> m = MagicMock()
>>> p = PropertyMock(return_value=3)
>>> type(m).foo = p
>>> m.foo
3
>>> p.assert_called_once_with()
```

Prudencia

If an `AttributeError` is raised by `PropertyMock`, it will be interpreted as a missing descriptor and `__getattr__()` will be called on the parent mock:

```
>>> m = MagicMock()
>>> no_attribute = PropertyMock(side_effect=AttributeEr
>>> type(m).my_property = no_attribute
>>> m.my_property
<MagicMock name='mock.my_property' id='140165240345424'
```

See **`__getattr__()`** for details.

class unittest.mock.AsyncMock(*spec=None, side_effect=None, return_value=DEFAULT, wraps=None, name=None, spec_set=None, unsafe=False, **kwargs*)

Una versión asíncrona de **`Mock`**. El objeto **`AsyncMock`** se comportará de tal modo que el objeto es reconocido como una función asíncrona y el resultado de su llamada es un objeto aguardable (awaitable).

```
>>> mock = AsyncMock()
>>> inspect.iscoroutinefunction(mock)
True
>>> inspect.isawaitable(mock())
True
```

El resultado de `mock()` es una función asíncrona que proporcionará el resultado de `side_effect` o de `return_value` después de haber sido aguardada:

- si `side_effect` es una función, la función asíncrona retornará el resultado de esa función,
- si `side_effect` es una excepción, la función asíncrona lanzará la excepción,
- si `side_effect` es un iterable, la función asíncrona retornará el siguiente valor del iterable, sin embargo, si se agota la secuencia de resultados, se lanza una excepción `StopAsyncIteration` inmediatamente,
- si `side_effect` no está definido, la función asíncrona retornará el valor definido por `return_value`, por lo tanto, la función asíncrona retorna un nuevo objeto

AsyncMock por defecto.

Establecer el argumento *spec* de un objeto **Mock** o **MagicMock** en una función asíncrona resultará en que un objeto corrutina será retornado después de la llamada al objeto.

```
>>> async def async_func(): pass
...
>>> mock = MagicMock(async_func)
>>> mock
<MagicMock spec='function' id='...'>
>>> mock()
<coroutine object AsyncMockMixin._mock_call at ...>
```

Establecer el argumento *spec* de un objeto **Mock**, **MagicMock** o **AsyncMock** en una clase que tiene simultáneamente funciones asíncronas y síncronas hará que se detecten automáticamente las funciones sincrónicas y las establecerá como `MagicMock` (si el objeto simulado padre es `AsyncMock` o `MagicMock`) o `Mock` (si el objeto simulado padre es `Mock`) . Todas las funciones asíncronas serán `AsyncMock`.

```
>>> class ExampleClass:
...     def sync_foo():
...         pass
...     async def async_foo():
...         pass
...
>>> a_mock = AsyncMock(ExampleClass)
>>> a_mock.sync_foo
<MagicMock name='mock.sync_foo' id='...'>
>>> a_mock.async_foo
<AsyncMock name='mock.async_foo' id='...'>
>>> mock = Mock(ExampleClass)
>>> mock.sync_foo
<Mock name='mock.sync_foo' id='...'>
>>> mock.async_foo
<AsyncMock name='mock.async_foo' id='...'>
```

Added in version 3.8.

assert_awaited()

Assert si el objeto simulado fue aguardado al menos una vez. Ten en cuenta que, independientemente del objeto que ha sido invocado, la palabra clave `await` debe ser utilizada:

```
>>> mock = AsyncMock()
>>> async def main(coroutine_mock):
...     await coroutine_mock
...
>>> coroutine_mock = mock()
>>> mock.called
True
>>> mock.assert_awaited()
Traceback (most recent call last):
...
AssertionError: Expected mock to have been awai
>>> asyncio.run(main(coroutine_mock))
>>> mock.assert_awaited()
```

assert_awaited_once()

Assert si el objeto simulado fue aguardado exactamente una vez.

```
>>> mock = AsyncMock()
>>> async def main():
...     await mock()
...
>>> asyncio.run(main())
>>> mock.assert_awaited_once()
>>> asyncio.run(main())
>>> mock.assert_awaited_once()
Traceback (most recent call last):
...
AssertionError: Expected mock to have been awai
```

assert_awaited_with(**args*, ***kwargs*)

Assert si el último aguardo (await) fue con los argumentos especificados.

```
>>> mock = AsyncMock()
>>> async def main(*args, **kwargs):
...     await mock(*args, **kwargs)
...
>>> asyncio.run(main('foo', bar='bar'))
>>> mock.assert_awaited_with('foo', bar='bar')
>>> mock.assert_awaited_with('other')
Traceback (most recent call last):
...
AssertionError: expected await not found.
Expected: mock('other')
Actual: mock('foo', bar='bar')
```

assert_awaited_once_with(**args*, ***kwargs*)

Assert si que el objeto simulado se ha aguardado exactamente una vez y con los argumentos especificados.

```
>>> mock = AsyncMock()
>>> async def main(*args, **kwargs):
...     await mock(*args, **kwargs)
...
>>> asyncio.run(main('foo', bar='bar'))
>>> mock.assert_awaited_once_with('foo', bar='b
>>> asyncio.run(main('foo', bar='bar'))
>>> mock.assert_awaited_once_with('foo', bar='b
Traceback (most recent call last):
...
AssertionError: Expected mock to have been awai
```

assert_any_await(**args*, ***kwargs*)

Assert si el objeto simulado nunca se ha aguardado con los argumentos especificados.

```
>>> mock = AsyncMock()
>>> async def main(*args, **kwargs):
...     await mock(*args, **kwargs)
...
>>> asyncio.run(main('foo', bar='bar'))
>>> asyncio.run(main('hello'))
>>> mock.assert_any_await('foo', bar='bar')
>>> mock.assert_any_await('other')
Traceback (most recent call last):
...
AssertionError: mock('other') await not found
```

assert_has_awaits(*calls*, *any_order=False*)

Assert si el objeto simulado ha sido aguardado con las llamadas especificadas. Para comprobar los aguardos (awaits) se utiliza la lista `await_args_list`.

Si *any_order* es falso, los aguardos (awaits) deben ser secuenciales. No puede haber llamadas adicionales antes o después de los aguardos especificados.

Si *any_order* es verdadero, entonces los aguardos (awaits) pueden estar en cualquier orden, pero deben aparecer todos en `await_args_list`.

```
>>> mock = AsyncMock()
>>> async def main(*args, **kwargs):
...     await mock(*args, **kwargs)
...
>>> calls = [call("foo"), call("bar")]
>>> mock.assert_has_awaits(calls)
Traceback (most recent call last):
...
AssertionError: Awaits not found.
Expected: [call('foo'), call('bar')]
Actual: []
>>> asyncio.run(main('foo'))
>>> asyncio.run(main('bar'))
>>> mock.assert_has_awaits(calls)
```

assert_not_awaited()

Assert si el objeto simulado nunca ha sido aguardado.

```
>>> mock = AsyncMock()
>>> mock.assert_not_awaited()
```

reset_mock(**args*, ***kwargs*)

Consultar **`Mock.reset_mock()`**. Además, también establece **`await_count`** a 0, **`await_args`** a None y borra **`await_args_list`**.

await_count

Un registro entero de cuántas veces se ha aguardado el objeto simulado.

```
>>> mock = AsyncMock()
>>> async def main():
...     await mock()
...
>>> asyncio.run(main())
>>> mock.await_count
1
>>> asyncio.run(main())
>>> mock.await_count
2
```

await_args

Este atributo es `None` (si el objeto simulado no se ha aguardado) o los argumentos con los que fue aguardado la última vez. Su funcionamiento es idéntico al de **`Mock.call_args`**.

```
>>> mock = AsyncMock()
>>> async def main(*args):
...     await mock(*args)
...
>>> mock.await_args
>>> asyncio.run(main('foo'))
>>> mock.await_args
call('foo')
>>> asyncio.run(main('bar'))
>>> mock.await_args
call('bar')
```

await_args_list

Es una lista de todas los aguardos (awaits) realizados en el objeto simulado de forma secuencial (por lo que la longitud de la lista es el número de veces que se ha aguardado el objeto). Si no se han realizado aguardos previos, es una lista vacía.

```
>>> mock = AsyncMock()
>>> async def main(*args):
...     await mock(*args)
...
>>> mock.await_args_list
[]
>>> asyncio.run(main('foo'))
>>> mock.await_args_list
[call('foo')]
>>> asyncio.run(main('bar'))
>>> mock.await_args_list
[call('foo'), call('bar')]
```

class unittest.mock.ThreadingMock(*spec=None*, *side_effect=None*, *return_value=DEFAULT*, *wraps=None*, *name=None*, *spec_set=None*, *unsafe=False*, ***, *timeout=UNSET*, ***kwargs*)

A version of **`MagicMock`** for multithreading tests. The **`ThreadingMock`** object provides extra methods to wait for a call to be invoked, rather than assert on it immediately.

The default timeout is specified by the `timeout` argument, or if unset by the **`ThreadingMock.DEFAULT_TIMEOUT`** attribute, which defaults to blocking (`None`).

You can configure the global default timeout by setting **`ThreadingMock.DEFAULT_TIMEOUT`**.

wait_until_called(*, *timeout=UNSET*)

Waits until the mock is called.

If a timeout was passed at the creation of the mock or if a timeout argument is passed to this function, the function raises an `AssertionError` if the call is not performed in time.

```
>>> mock = ThreadingMock()
>>> thread = threading.Thread(target=mock)
>>> thread.start()
>>> mock.wait_until_called(timeout=1)
>>> thread.join()
```

wait_until_any_call_with(**args*, ***kwargs*)

Waits until the mock is called with the specified arguments.

If a timeout was passed at the creation of the mock the function raises an `AssertionError` if the call is not performed in time.

```
>>> mock = ThreadingMock()
>>> thread = threading.Thread(target=mock, args
>>> thread.start()
>>> mock.wait_until_any_call_with("arg1", "arg2
>>> thread.join()
```

DEFAULT_TIMEOUT

Global default timeout in seconds to create instances of **`ThreadingMock`**.

Added in version 3.13.

Llamar a los objetos simulados

Los objetos Mock son invocables. La llamada a uno retornará el valor establecido en el atributo `return_value`. El valor de

retorno por defecto es un nuevo objeto Mock, el cual se crea la primera vez que se accede al valor de retorno (ya sea explícitamente o llamando al objeto Mock). Una vez creado, se almacena y el mismo objeto es retornado cada vez que se solicita.

Las llamadas realizadas al objeto serán registradas en los atributos `call_args` y `call_args_list`.

If `side_effect` is set then it will be called after the call has been recorded, so if `side_effect` raises an exception the call is still recorded.

La forma más sencilla de hacer que un objeto simulado lance de una excepción cuando sea invocado es establecer su atributo `side_effect` como una clase o instancia de excepción:

```
>>> m = MagicMock(side_effect=IndexError)
>>> m(1, 2, 3)
Traceback (most recent call last):
  ...
IndexError
>>> m.mock_calls
[call(1, 2, 3)]
>>> m.side_effect = KeyError('Bang!')
>>> m('two', 'three', 'four')
Traceback (most recent call last):
  ...
KeyError: 'Bang!'
>>> m.mock_calls
[call(1, 2, 3), call('two', 'three', 'four')]
```

If `side_effect` is a function then whatever that function returns is what calls to the mock return. The `side_effect` function is called with the same arguments as the mock. This allows you to vary the return value of the call dynamically, based on the input:

```
>>> def side_effect(value):
...     return value + 1
...
>>> m = MagicMock(side_effect=side_effect)
>>> m(1)
2
>>> m(2)
3
>>> m.mock_calls
[call(1), call(2)]
```

If you want the mock to still return the default return value (a new mock), or any set return value, then there are two ways of doing this. Either return **`return_value`** from inside **`side_effect`**, or return **`DEFAULT`**:

```
>>> m = MagicMock()
>>> def side_effect(*args, **kwargs):
...     return m.return_value
...
>>> m.side_effect = side_effect
>>> m.return_value = 3
>>> m()
3
>>> def side_effect(*args, **kwargs):
...     return DEFAULT
...
>>> m.side_effect = side_effect
>>> m()
3
```

To remove a **`side_effect`**, and return to the default behaviour, set the **`side_effect`** to `None`:

```
>>> m = MagicMock(return_value=6)
>>> def side_effect(*args, **kwargs):
...     return 3
...
>>> m.side_effect = side_effect
>>> m()
3
>>> m.side_effect = None
>>> m()
```

```
6
```

The `side_effect` can also be any iterable object. Repeated calls to the mock will return values from the iterable (until the iterable is exhausted and a `StopIteration` is raised):

```
>>> m = MagicMock(side_effect=[1, 2, 3])
>>> m()
1
>>> m()
2
>>> m()
3
>>> m()
Traceback (most recent call last):
  ...
StopIteration
```

Si cualquier miembro del iterable es una excepción, se lanzará en lugar de retornarse:

```
>>> iterable = (33, ValueError, 66)
>>> m = MagicMock(side_effect=iterable)
>>> m()
33
>>> m()
Traceback (most recent call last):
 ...
ValueError
>>> m()
66
```

Eliminar atributos

Los objetos simulados crean atributos en demanda. Esto les permite hacerse pasar por objetos de cualquier tipo.

You may want a mock object to return `False` to a `hasattr()` call, or raise an `AttributeError` when an attribute is fetched. You can do this by providing an object as a **`spec`** for a mock, but that isn’t always convenient.

Puedes «bloquear» atributos eliminándolos. Una vez eliminado, el acceso a un atributo lanzará una excepción **`AttributeError`**.

```
>>> mock = MagicMock()
>>> hasattr(mock, 'm')
True
>>> del mock.m
>>> hasattr(mock, 'm')
False
>>> del mock.f
>>> mock.f
Traceback (most recent call last):
    ...
AttributeError: f
```

Los nombres de los objetos simulados y el atributo *name*

Dado que «name» es un argumento para el constructor de la clase **`Mock`**, si quieres que tu objeto simulado tenga un atributo «name», no puedes simplemente pasarlo al constructor en tiempo de creación. Hay dos alternativas. Una opción es usar el método **`configure_mock()`**:

```
>>> mock = MagicMock()
>>> mock.configure_mock(name='my_name')
>>> mock.name
'my_name'
```

Una opción más sencilla es simplemente establecer el atributo «name» después de la creación del objeto simulado:

```
>>> mock = MagicMock()
>>> mock.name = "foo"
```

Adjuntar objetos simulados como atributos

Cuando se adjunta un objeto simulado como un atributo de otro objeto simulado (o como su valor de retorno) se convierte en un «hijo» del mismo. Las llamadas a los hijos se registran en los atributos **`method_calls`** y **`mock_calls`** del padre. Esto es útil para configurar objetos simulados hijos para después adjuntarlos al padre, o para adjuntar objetos simulados a un padre que se encargará de registrar todas las llamadas a los hijos, permitiéndote hacer aserciones sobre el orden de las llamadas entre objetos simulados:

```
>>> parent = MagicMock()
>>> child1 = MagicMock(return_value=None)
>>> child2 = MagicMock(return_value=None)
>>> parent.child1 = child1
>>> parent.child2 = child2
>>> child1(1)
>>> child2(2)
>>> parent.mock_calls
[call.child1(1), call.child2(2)]
```

La excepción a lo anterior es si el objeto simulado tiene un nombre. Esto te permite evitar el comportamiento de «parentesco» explicado previamente, si por alguna razón no deseas que suceda.

```
>>> mock = MagicMock()
>>> not_a_child = MagicMock(name='not-a-child')
>>> mock.attribute = not_a_child
>>> mock.attribute()
<MagicMock name='not-a-child()' id='...'>
>>> mock.mock_calls
[]
```

Los objetos simulados creados automáticamente por la función **`patch()`** también reciben nombres automáticamente. Para adjuntar un objeto simulado con nombre a un padre debes utilizar el método **`attach_mock()`**:

```
>>> thing1 = object()
>>> thing2 = object()
>>> parent = MagicMock()
>>> with patch('__main__.thing1', return_value=None) as
```

```
...     with patch('__main__.thing2', return_value=None)
...         parent.attach_mock(child1, 'child1')
...         parent.attach_mock(child2, 'child2')
...         child1('one')
...         child2('two')
...
>>> parent.mock_calls
[call.child1('one'), call.child2('two')]
```

Las únicas excepciones son los métodos y atributos mágicos (aquellos que tienen doble subrayado al principio y al final). Mock no los crea automáticamente, sino que lanza una excepción **`AttributeError`**. Esto se debe a que el intérprete a menudo solicitará implícitamente estos métodos y terminaría *muy* confundido si obtiene un nuevo objeto Mock cuando espera un método mágico. Si necesitas soporte para métodos mágicos, consulta métodos mágicos.

Parcheadores

Los decoradores patch se utilizan para parchear los objetos solo dentro del ámbito de la función que decoran. Se encargan automáticamente de desparchear una vez terminada la prueba, incluso si se lanzan excepciones. Todas estas funciones también se pueden utilizar con declaraciones o como decoradores de clase.

patch

Nota

La clave es realizar el parche en el mismo espacio de nombre. Para más detalles consultar la sección where to patch.

unittest.mock.patch(*target*, *new=DEFAULT*, *spec=None*, *create=False*, *spec_set=None*, *autospec=None*, *new_callable=None*, ***kwargs*)

`patch()` actúa como decorador de función, decorador de clase o como gestor de contexto. Ya sea en el interior del cuerpo de una función o dentro de una declaración with, *target* es parcheado con un objeto *new*. Cuando la función / declaración with termina su ejecución el parche se deshace automáticamente.

Si se omite *new*, entonces el objetivo se reemplaza por un objeto `AsyncMock` si el objeto parcheado es una función asíncrona, o con un objeto `MagicMock` en caso contrario. Si se utiliza `patch()` como decorador y se omite *new*, el objeto simulado creado se pasa como un argumento adicional a la función decorada. Si se utiliza `patch()` como un gestor de contexto, el objeto simulado creado es retornado por el gestor de contexto.

target debe ser una cadena de la forma `'paquete.modulo.NombreDeLaClase'`. *target* es importado y el objeto especificado reemplazado por el objeto *new*, por lo que *target* debe ser importable desde el entorno desde el cual estás llamando a `patch()`. Hay que tener presente que *target* es importado cuando se ejecuta la función decorada, no en tiempo de decoración.

Los argumentos *spec* y *spec_set* se pasan a `MagicMock` si patch está creando automáticamente uno para ti.

Además, puedes pasar `spec=True` o `spec_set=True`, lo que causa que patch pase el objeto que está siendo simulado como el objeto spec/spec_set.

new_callable te permite especificar una clase diferente, o un objeto invocable, que será invocada para crear el objeto *new*. Por defecto se utiliza `AsyncMock` para las funciones asíncronas y `MagicMock` para el resto.

Una variante más poderosa de *spec* es *autospec*. Si estableces `autospec=True` el objeto simulado se creará con una especificación del objeto que está siendo reemplazado. Todos los atributos del objeto simulado también tendrán la especificación del atributo correspondiente del objeto que está siendo reemplazado. Los argumentos de los métodos y funciones simulados son comprobados y se lanzará una excepción `TypeError` si se les llama con la firma incorrecta. Para los objetos simulados que sustituyen a una clase, su valor de retorno (la "instancia") tendrá la misma especificación que la clase. Consultar la función `create_autospec()` y Autoespecificación para más detalles.

En lugar de `autospec=True`, puedes pasar `autospec=some_object` para utilizar un objeto arbitrario como especificación en lugar del objeto reemplazado.

Por defecto, `patch()` fallará al intentar reemplazar atributos que no existen. Si pasas `create=True` y no existe el atributo, patch crea el atributo cuando la función se llama y lo elimina de nuevo en cuanto termina de ejecutarse . Esto es útil para implementar pruebas para atributos que tu código de producción crea en tiempo de ejecución. Está desactivado por defecto, ya que puede ser peligroso. ¡Al activarlo se pueden implementar pruebas que validen APIs que en realidad no existen!

Nota

Distinto en la versión 3.5: Si estás parcheando objetos incorporados (builtins) en un módulo, no es necesario pasar `create=True`, ya que en este caso se agrega de forma predeterminada.

Patch can be used as a `TestCase` class decorator. It works by decorating each test method in the class. This reduces the boilerplate code when your test methods share a common patchings set. `patch()` finds tests by looking for method names that start with `patch.TEST_PREFIX`. By default this is `'test'`, which matches the way `unittest` finds tests. You can specify an alternative prefix by setting `patch.TEST_PREFIX`.

Patch puede ser usado como un gestor de contexto, con la declaración with. En este caso el parcheo se aplica al bloque sangrado inmediatamente después de la declaración with. Si utilizas «as», el objeto parcheado será enlazado al nombre especificado después de «as»; muy útil si **`patch()`** está creando un objeto simulado automáticamente.

`patch()` toma argumentos por palabra clave arbitrarios. Estos serán pasados a **`AsyncMock`** si el objeto parcheado es asincrónico, si es de otra forma **`MagicMock`** o a un *new_callable* si es especificado.

`patch.dict(...)`, `patch.multiple(...)` y `patch.object(...)` están disponibles para casos de uso alternativos.

`patch()` como decorador de función, crea el objeto simulado por ti y lo pasa a la función decorada:

```
>>> @patch('__main__.SomeClass')
... def function(normal_argument, mock_class):
...     print(mock_class is SomeClass)
...
>>> function(None)
True
```

Parchear una clase sustituye a la clase por una *instancia* de **`MagicMock`**. Si la clase se instancia en el código bajo prueba, el atributo **`return_value`** del objeto simulado será el utilizado.

Si la clase se instancia en múltiples ocasiones, puedes utilizar el atributo **`side_effect`** para retornar un nuevo objeto simulado cada vez. O también puedes establecer *return_value* para que sea lo que tu quieras.

To configure return values on methods of *instances* on the patched class you must do this on the **`return_value`**. For example:

```
>>> class Class:
...     def method(self):
...         pass
...
>>> with patch('__main__.Class') as MockClass:
```

```
...     instance = MockClass.return_value
...     instance.method.return_value = 'foo'
...     assert Class() is instance
...     assert Class().method() == 'foo'
...
```

Si utilizas *spec* o *spec_set* y `patch()` está reemplazando una *clase*, el valor de retorno del objeto simulado creado tendrá las mismas especificaciones:

```
>>> Original = Class
>>> patcher = patch('__main__.Class', spec=True)
>>> MockClass = patcher.start()
>>> instance = MockClass()
>>> assert isinstance(instance, Original)
>>> patcher.stop()
```

El argumento *new_callable* es útil cuando deseas utilizar una clase por defecto alternativa a `MagicMock` para el objeto simulado creado. Por ejemplo, si quieres que se use una clase `NonCallableMock` por defecto:

```
>>> thing = object()
>>> with patch('__main__.thing', new_callable=NonCallabl
...     assert thing is mock_thing
...     thing()
...
Traceback (most recent call last):
  ...
TypeError: 'NonCallableMock' object is not callable
```

Otro caso de uso podría ser la sustitución de un objeto por una instancia de `io.StringIO`:

```
>>> from io import StringIO
>>> def foo():
...     print('Something')
...
>>> @patch('sys.stdout', new_callable=StringIO)
... def test(mock_stdout):
```

```
...     foo()
...     assert mock_stdout.getvalue() == 'Something\n'
...
>>> test()
```

Cuando `patch()` crea un objeto simulado para ti, a menudo lo primero que tienes que hacer es configurar el objeto simulado recién creado. Parte de la configuración se puede hacer en la propia llamada a patch. Cualquier palabra clave arbitraria que pases a la llamada será utilizada para establecer los atributos del objeto simulado creado:

```
>>> patcher = patch('__main__.thing', first='one', secon
>>> mock_thing = patcher.start()
>>> mock_thing.first
'one'
>>> mock_thing.second
'two'
```

Los atributos de los objetos simulados hijos, como `return_value` y `side_effect`, también puede ser configurados en la llamada, dado que los objetos simulados hijos son atributos en si mismos del objeto simulado padre creado. Eso si, estos no son sintácticamente válidos para ser pasados directamente como argumentos por palabras clave a la función patch, pero un diccionario con ellos como claves si que puede ser expandido en una llama a `patch()` usando `**`:

```
>>> config = {'method.return_value': 3, 'other.side_effe
>>> patcher = patch('__main__.thing', **config)
>>> mock_thing = patcher.start()
>>> mock_thing.method()
3
>>> mock_thing.other()
Traceback (most recent call last):
  ...
KeyError
```

Por defecto, el intento de parchear una función en un módulo (o un método o atributo en una clase) que no existe fallará lanzando una excepción **`AttributeError`**:

```
>>> @patch('sys.non_existing_attribute', 42)
... def test():
...     assert sys.non_existing_attribute == 42
...
>>> test()
Traceback (most recent call last):
  ...
AttributeError: <module 'sys' (built-in)> does not have
```

pero añadir `create=True` en la llamada a **`patch()`** hará que el ejemplo previo funcione como se esperaba:

```
>>> @patch('sys.non_existing_attribute', 42, create=True
... def test(mock_stdout):
...     assert sys.non_existing_attribute == 42
...
>>> test()
```

Distinto en la versión 3.8: **`patch()`** ahora retorna una instancia de **`AsyncMock`** si el objetivo es una función asíncrona.

patch.object

patch.object(*target*, *attribute*, *new=DEFAULT*, *spec=None*, *create=False*, *spec_set=None*, *autospec=None*, *new_callable=None*, ***kwargs*)

parchea el miembro *attribute* invocado de un objeto *target* con un objeto simulado.

`patch.object()` se puede utilizar como un decorador, decorador de clase o un gestor de contexto. Los argumentos *new*, *spec*, *create*, *spec_set*, *autospec* y *new_callable* tienen el mismo significado que en la función **`patch()`**. Al igual que **`patch()`**, **`patch.object()`** toma argumentos por palabras clave arbitrarios para la configuración del objeto simulado que crea.

Cuando se utiliza como un decorador de clase

`patch.object()` se rige por `patch.TEST_PREFIX` a la hora de elegir los métodos a envolver.

Puedes llamar a **`patch.object()`** con tres argumentos o con dos argumentos. La forma de tres argumento toma el objeto a parchear, el nombre del atributo y el objeto con el que reemplazar el atributo.

Al llamar usando la variante de dos argumento se omite el objeto de sustitución, por lo tanto se crea un objeto simulado automáticamente y se pasa como argumento adicional a la función decorada:

```
>>> @patch.object(SomeClass, 'class_method')
... def test(mock_method):
...     SomeClass.class_method(3)
...     mock_method.assert_called_with(3)
...
>>> test()
```

spec, *create* y el resto de argumentos de **`patch.object()`** tienen el mismo significado que en la función **`patch()`**.

patch.dict

patch.dict(*in_dict*, *values=()*, *clear=False*, ***kwargs*)

Patch a dictionary, or dictionary like object, and restore the dictionary to its original state after the test, where the restored dictionary is a copy of the dictionary as it was before the test.

in_dict puede ser un diccionario o un contenedor similar a uno. Si se trata de un objeto similar a un diccionario, debe soportar como mínimo el establecimiento, la obtención y la eliminación de elementos, además de permitir la iteración sobre las claves.

in_dict también puede ser una cadena que especifique el nombre del diccionario a parchear, el nombre es usado para obtener el diccionario mediante importación.

values puede ser otro diccionario con valores para ser añadidos al diccionario. *values* también puede ser un iterable de parejas `(clave, valor)`.

Si *clear* es verdadero, el contenido del diccionario se borrará antes de establecer los nuevos valores.

La función **`patch.dict()`** también puede ser invocada con argumentos por palabra clave arbitrarios para establecer los valores en el diccionario.

Distinto en la versión 3.8: La función **`patch.dict()`** ahora retorna el diccionario parcheado cuando se utiliza como gestor de contexto.

`patch.dict()` se puede utilizar como gestor de contexto, decorador o decorador de clase:

```
>>> foo = {}
>>> @patch.dict(foo, {'newkey': 'newvalue'})
... def test():
...     assert foo == {'newkey': 'newvalue'}
...
>>> test()
>>> assert foo == {}
```

Cuando se utiliza **`patch.dict()`** como decorador de clase, se rige por `patch.TEST_PREFIX` (por defecto `'test'`) a la hora de elegir qué métodos envolver:

```
>>> import os
>>> import unittest
>>> from unittest.mock import patch
>>> @patch.dict('os.environ', {'newkey': 'newvalue'})
... class TestSample(unittest.TestCase):
...     def test_sample(self):
...         self.assertEqual(os.environ['newkey'], 'newv
```

Si deseas utilizar un prefijo diferente para tu prueba, puede informar a los parcheadores del nuevo prefijo a usar estableciendo `patch.TEST_PREFIX`. Para más detalles sobre cómo cambiar el valor del atributo consultar TEST_PREFIX.

`patch.dict()` puede utilizarse para agregar miembros a un diccionario, o simplemente para dejar que una prueba modifique un diccionario y asegurarte de que el diccionario será restablecido cuando finalice la misma.

```
>>> foo = {}
>>> with patch.dict(foo, {'newkey': 'newvalue'}) as patc
...     assert foo == {'newkey': 'newvalue'}
...     assert patched_foo == {'newkey': 'newvalue'}
...     # You can add, update or delete keys of foo (or
...     patched_foo['spam'] = 'eggs'
...
>>> assert foo == {}
>>> assert patched_foo == {}
```

```
>>> import os
>>> with patch.dict('os.environ', {'newkey': 'newvalue'}
...     print(os.environ['newkey'])
...
newvalue
>>> assert 'newkey' not in os.environ
```

Se pueden utilizar argumentos por palabra clave en la llamada a **`patch.dict()`** para establecer valores en el diccionario:

```
>>> mymodule = MagicMock()
>>> mymodule.function.return_value = 'fish'
>>> with patch.dict('sys.modules', mymodule=mymodule):
...     import mymodule
...     mymodule.function('some', 'args')
...
'fish'
```

`patch.dict()` can be used with dictionary like objects that aren't actually dictionaries. At the very minimum they must support item getting, setting, deleting and either iteration or membership test. This corresponds to the magic methods **`__getitem__()`**,

`__setitem__()`, **`__delitem__()`** and either **`__iter__()`** or **`__contains__()`**.

```
>>> class Container:
...     def __init__(self):
...         self.values = {}
...     def __getitem__(self, name):
...         return self.values[name]
...     def __setitem__(self, name, value):
...         self.values[name] = value
...     def __delitem__(self, name):
...         del self.values[name]
...     def __iter__(self):
...         return iter(self.values)
...
>>> thing = Container()
>>> thing['one'] = 1
>>> with patch.dict(thing, one=2, two=3):
...     assert thing['one'] == 2
...     assert thing['two'] == 3
...
>>> assert thing['one'] == 1
>>> assert list(thing) == ['one']
```

patch.multiple

patch.multiple(*target*, *spec=None*, *create=False*, *spec_set=None*, *autospec=None*, *new_callable=None*, ***kwargs*)

Realiza múltiples parches en una sola llamada. Se toma el objeto a ser parcheado (ya sea como un objeto o una cadena de caracteres con el nombre del mismo para obtener el objeto mediante importación) y los argumentos por palabras clave para los parches:

```
with patch.multiple(settings, FIRST_PATCH='one', SE
    ...
```

Usa **`DEFAULT`** como valor si deseas que la función **`patch.multiple()`** cree objetos simulados automáticamente por ti. En este caso, los objetos simulados creados son pasados a la función decorada mediante palabra clave y se retorna un diccionario cuando `patch.multiple()` se utiliza como gestor de contexto.

La función `patch.multiple()` se puede utilizar como un decorador, decorador de clase o como gestor de contexto. La argumentos *spec*, *spec_set*, *create*, *autospec* y *new_callable* tienen el mismo significado que en la función `patch()`. Estos argumentos se aplicarán a *todos* los parches realizados por `patch.multiple()`.

Cuando se utiliza como decorador de clase, `patch.multiple()` se rige por `patch.TEST_PREFIX` a la hora de elegir qué métodos envolver.

Si deseas que `patch.multiple()` cree objetos simulados automáticamente por ti, puedes utilizar `DEFAULT` como valor. Si utilizas `patch.multiple()` como decorador, entonces los objetos simulados creados se pasan a la función decorada por palabra clave:

```
>>> thing = object()
>>> other = object()

>>> @patch.multiple('__main__', thing=DEFAULT, other=DEF
... def test_function(thing, other):
...     assert isinstance(thing, MagicMock)
...     assert isinstance(other, MagicMock)
...
>>> test_function()
```

`patch.multiple()` se puede anidar junto a otros decoradores `patch`, pero los argumentos pasados por palabra clave se deben agregar *después* de cualquier argumento estándar creado por `patch()`:

```
>>> @patch('sys.exit')
... @patch.multiple('__main__', thing=DEFAULT, other=DEF
... def test_function(mock_exit, other, thing):
...     assert 'other' in repr(other)
...     assert 'thing' in repr(thing)
...     assert 'exit' in repr(mock_exit)
...
>>> test_function()
```

Si `patch.multiple()` se utiliza como un gestor de contexto, el valor retornado por el mismo es un diccionario en el que los objetos simulados creados son almacenados, usando sus nombres como claves:

```
>>> with patch.multiple('__main__', thing=DEFAULT, other
...     assert 'other' in repr(values['other'])
...     assert 'thing' in repr(values['thing'])
...     assert values['thing'] is thing
...     assert values['other'] is other
...
```

métodos start y stop de patch

All the patchers have **`start()`** and **`stop()`** methods. These make it simpler to do patching in `setUp` methods or where you want to do multiple patches without nesting decorators or with statements.

To use them call `patch()`, `patch.object()` or `patch.dict()` as normal and keep a reference to the returned `patcher` object. You can then call **`start()`** to put the patch in place and **`stop()`** to undo it.

Si estás utilizando `patch()` para crear un objeto simulado automáticamente, este será retornado por la llamada a `patcher.start`:

```
>>> patcher = patch('package.module.ClassName')
>>> from package import module
>>> original = module.ClassName
>>> new_mock = patcher.start()
>>> assert module.ClassName is not original
>>> assert module.ClassName is new_mock
>>> patcher.stop()
>>> assert module.ClassName is original
>>> assert module.ClassName is not new_mock
```

A typical use case for this might be for doing multiple patches in the `setUp` method of a **`TestCase`**:

```
>>> class MyTest(unittest.TestCase):
...     def setUp(self):
...         self.patcher1 = patch('package.module.Class1
...         self.patcher2 = patch('package.module.Class2
...         self.MockClass1 = self.patcher1.start()
...         self.MockClass2 = self.patcher2.start()
...
...     def tearDown(self):
...         self.patcher1.stop()
...         self.patcher2.stop()
...
...     def test_something(self):
...         assert package.module.Class1 is self.MockCla
...         assert package.module.Class2 is self.MockCla
...
>>> MyTest('test_something').run()
```

Prudencia

Si se utiliza esta técnica debes asegurarte de que el parcheo está «sin aplicar» llamando a `stop`. Esto puede ser más complicado de lo que parece, ya que si se produce una excepción en el `setUp` no se llama a `tearDown` a continuación. **`unittest.TestCase.addCleanup()`** hace que esto sea más fácil:

```
>>> class MyTest(unittest.TestCase):
...     def setUp(self):
...         patcher = patch('package.module.Class')
...         self.MockClass = patcher.start()
...         self.addCleanup(patcher.stop)
...
...     def test_something(self):
...         assert package.module.Class is self.MockCla
```

```
...
```

Como beneficio adicional, ya no es necesario mantener una referencia al objeto `patcher`.

También es posible detener todos los parches que han sido iniciados usando **`patch.stopall()`**.

patch.stopall()

Detiene todos los parches activos. Solo se detienen parches iniciados con `start`.

parchear objetos incorporados (builtins)

Puedes parchear cualquier objeto incorporado dentro de un módulo. El siguiente ejemplo parchea la función incorporada **`ord()`**:

```
>>> @patch('__main__.ord')
... def test(mock_ord):
...     mock_ord.return_value = 101
...     print(ord('c'))
...
>>> test()
101
```

TEST_PREFIX

Todos los parcheadores se puede utilizar como decoradores de clase. Cuando se usan de esta forma, todos los métodos a probar en la clase son envueltos. Los parcheadores reconocen los métodos que comienzan con `'test'` como métodos a probar. Esta es la misma forma que la clase **`unittest.TestLoader`** usa para encontrar métodos de prueba por defecto.

Cabe la posibilidad de que desees utilizar un prefijo diferente para las pruebas. Puedes informar a los parcheadores del cambio de prefijo estableciendo el atributo `patch.TEST_PREFIX`:

```
>>> patch.TEST_PREFIX = 'foo'
>>> value = 3
>>>
>>> @patch('__main__.value', 'not three')
... class Thing:
...     def foo_one(self):
...         print(value)
...     def foo_two(self):
...         print(value)
...
>>>
>>> Thing().foo_one()
not three
>>> Thing().foo_two()
not three
>>> value
3
```

Anidando decoradores patch

Si deseas aplicar múltiples parches, solo tienes que apilar los decoradores.

Puede apilar múltiples decoradores patch usando el siguiente patrón:

```
>>> @patch.object(SomeClass, 'class_method')
... @patch.object(SomeClass, 'static_method')
... def test(mock1, mock2):
...     assert SomeClass.static_method is mock1
...     assert SomeClass.class_method is mock2
...     SomeClass.static_method('foo')
...     SomeClass.class_method('bar')
...     return mock1, mock2
...
>>> mock1, mock2 = test()
>>> mock1.assert_called_once_with('foo')
>>> mock2.assert_called_once_with('bar')
```

Ten en cuenta que los decoradores se aplican desde abajo hacia arriba. Esta es la manera estándar de Python para aplicar decoradores. El orden en el que los objetos simulados generados se pasan a la función de prueba coincide con este orden.

Dónde parchear

`patch()` funciona cambiando (temporalmente) el objeto al que apunta *name* con otro. Puede haber muchos nombres apuntando a cualquier objeto individual, por lo que para que el parcheo funcione, debes asegurarte de parchear el nombre utilizado para el objeto por el sistema concreto bajo prueba.

El principio básico es parchear donde un objeto es *buscado*, que no es necesariamente el mismo lugar donde está definido. Un par de ejemplos ayudarán a aclarar esto.

Imaginemos que tenemos un proyecto, que queremos probar, con la siguiente estructura:

```
a.py
    -> Defines SomeClass

b.py
    -> from a import SomeClass
    -> some_function instantiates SomeClass
```

Now we want to test `some_function` but we want to mock out `SomeClass` using **`patch()`**. The problem is that when we import module b, which we will have to do when it imports `SomeClass` from module a. If we use **`patch()`** to mock out `a.SomeClass` then it will have no effect on our test; module b already has a reference to the *real* `SomeClass` and it looks like our patching had no effect.

La clave es parchear `SomeClass` donde se utiliza (o donde es buscado). En este caso `some_function` realmente va a buscar `SomeClass` en el módulo b, donde lo hemos importado. La aplicación de parches debe ser similar a:

```
@patch('b.SomeClass')
```

Sin embargo, ten en cuenta el escenario alternativo donde en el módulo b en lugar de `from a import SomeClass` se hace `import a` y `some_function` usa `a.SomeClass`. Ambas formas de importación son comunes. En este caso, la clase que queremos parchear está siendo buscada en el módulo, por lo que tenemos que parchear `a.SomeClass`:

```
@patch('a.SomeClass')
```

Parcheando descriptores y objetos proxy

Tanto patch como patch.object parchean y restauran correctamente los descriptores: métodos de clase, métodos estáticos y propiedades. Debe parcharlos en el *class* en lugar de una instancia. También funcionan con objetos *some* que acceden a atributos de proxy, como django settings object [https://web.archive.org/web/20200603181648/http://www.voidspace.org.uk/python/weblog/arch_d7_2010_12_04.shtml#e1198].

MagicMock y el soporte de métodos mágicos

Simular métodos mágicos

`Mock` supports mocking the Python protocol methods, also known as «magic methods». This allows mock objects to replace containers or other objects that implement Python protocols.

Dado que los métodos mágicos se buscan de manera diferente a los métodos normales [2], este soporte ha sido especialmente implementado. Esto significa que solo es compatible con métodos mágicos específicos. La lista de métodos soportados incluye *casi* todos los existentes. Si hay alguno que falta y que consideras necesario, por favor háznoslo saber.

Puedes simular métodos mágicos estableciendo el método que te interesa en una función o en una instancia simulada. Si utilizas una función, *debe* aceptar `self` como primer argumento [3].

```
>>> def __str__(self):
```

```
...     return 'fooble'
...
>>> mock = Mock()
>>> mock.__str__ = __str__
>>> str(mock)
'fooble'
```

```
>>> mock = Mock()
>>> mock.__str__ = Mock()
>>> mock.__str__.return_value = 'fooble'
>>> str(mock)
'fooble'
```

```
>>> mock = Mock()
>>> mock.__iter__ = Mock(return_value=iter([]))
>>> list(mock)
[]
```

Un caso de uso para esto es simular objetos usados como gestores de contexto en una declaración **`with`**:

```
>>> mock = Mock()
>>> mock.__enter__ = Mock(return_value='foo')
>>> mock.__exit__ = Mock(return_value=False)
>>> with mock as m:
...     assert m == 'foo'
...
>>> mock.__enter__.assert_called_with()
>>> mock.__exit__.assert_called_with(None, None, None)
```

Las llamadas a métodos mágicos no aparecen registradas en **`method_calls`**, aunque si se registran en **`mock_calls`**.

Nota

Si se utiliza el argumento por palabra clave *spec* para crear un objeto simulado, intentar después establecer un método mágico que no está en la especificación lanzará una excepción **`AttributeError`**.

La lista completa de los métodos mágicos soportados es la siguiente:

- `__hash__`, `__sizeof__`, `__repr__` y `__str__`
- `__dir__`, `__format__` y `__subclasses__`
- `__round__`, `__floor__`, `__trunc__` y `__ceil__`
- Comparaciones: `__lt__`, `__gt__`, `__le__`, `__ge__`, `__eq__` y `__ne__`
- Métodos de contenedores: `__getitem__`, `__setitem__`, `__delitem__`, `__contains__`, `__len__`, `__iter__`, `__reversed__` y `__missing__`
- Administrador de contexto: `__enter__`, `__exit__`, `__aenter__` y `__aexit__`
- Métodos numéricos unarios: `__neg__`, `__pos__` y `__invert__`
- Los métodos numéricos (incluidas las variantes de mano derecha e in situ): `__add__`, `__sub__`, `__mul__`, `__matmul__`, `__truediv__`, `__floordiv__`, `__mod__`, `__divmod__`, `__lshift__`, `__rshift__`, `__and__`, `__xor__`, `__or__` y `__pow__`
- Métodos de conversión numérica: `__complex__`, `__int__`, `__float__` y `__index__`
- Métodos de descriptores: `__get__`, `__set__` y `__delete__`
- Pickling: `__reduce__`, `__reduce_ex__`, `__getinitargs__`, `__getnewargs__`, `__getstate__` y `__setstate__`
- Representación de rutas del sistema de archivos: `__fspath__`
- Métodos de iteración asíncronos: `__aiter__` y `__anext__`

Distinto en la versión 3.8: Se añadió soporte para **`os.PathLike.__fspath__()`**.

Distinto en la versión 3.8: Se añadió soporte para `__aenter__`, `__aexit__`, `__aiter__` y `__anext__`.

Los siguientes métodos existen pero *no* están soportados, ya sea porque son usados por el propio objeto simulado, porque no se pueden establecer dinámicamente o porque pueden causar problemas:

- `__getattr__`, `__setattr__`, `__init__` y `__new__`
- `__prepare__`, `__instancecheck__`, `__subclasscheck__` y `__del__`

Magic Mock

Hay dos variantes de `MagicMock`: **`MagicMock`** y **`NonCallableMagicMock`**.

class unittest.mock.MagicMock(**args*, ***kw*)

> `MagicMock` is a subclass of **`Mock`** with default implementations of most of the magic methods. You can use `MagicMock` without having to configure the magic methods yourself.
>
> Los parámetros del constructor tienen el mismo significado que en **`Mock`**.
>
> Si utilizas los argumentos *spec* o *spec_set* entonces *solo* los métodos mágicos que existan en la especificación serán creados.

class unittest.mock.NonCallableMagicMock(**args*, ***kw*)

> Una versión no invocable de **`MagicMock`**.
>
> Los parámetros del constructor tienen el mismo significado que en **`MagicMock`**, con la excepción de *return_value* y *side_effect* que no tienen significado en un objeto simulado no invocable.

Los métodos mágicos están implementados mediante objetos **`MagicMock`**, por lo que puedes configurarlos y utilizarlos de la forma habitual:

```
>>> mock = MagicMock()
>>> mock[3] = 'fish'
>>> mock.__setitem__.assert_called_with(3, 'fish')
>>> mock.__getitem__.return_value = 'result'
>>> mock[2]
```

```
'result'
```

Por defecto, muchos de los métodos de protocolo están obligados a retornar objetos de un tipo específico. Estos métodos están preconfigurados con un valor de retorno por defecto, de manera que puedan ser utilizados sin tener que hacer nada más, siempre que no estés interesado en el valor de retorno. En todo caso, puedes *establecer* el valor de retorno de forma manual si deseas cambiar el valor predeterminado.

Métodos y sus valores por defecto:

- `__lt__`: **`NotImplemented`**
- `__gt__`: **`NotImplemented`**
- `__le__`: **`NotImplemented`**
- `__ge__`: **`NotImplemented`**
- `__int__`: `1`
- `__contains__`: `False`
- `__len__`: `0`
- `__iter__`: `iter([])`
- `__exit__`: `False`
- `__aexit__`: `False`
- `__complex__`: `1j`
- `__float__`: `1.0`
- `__bool__`: `True`
- `__index__`: `1`
- `__hash__`: hash predeterminado del objeto simulado
- `__str__`: str predeterminado del objeto simulado
- `__sizeof__`: sizeof predeterminado del objeto simulado

Por ejemplo:

```
>>> mock = MagicMock()
>>> int(mock)
1
>>> len(mock)
0
>>> list(mock)
[]
>>> object() in mock
```

```
False
```

The two equality methods, **`__eq__()`** and **`__ne__()`**, are special. They do the default equality comparison on identity, using the **`side_effect`** attribute, unless you change their return value to return something else:

```
>>> MagicMock() == 3
False
>>> MagicMock() != 3
True
>>> mock = MagicMock()
>>> mock.__eq__.return_value = True
>>> mock == 3
True
```

El valor de retorno de **`MagicMock.__iter__()`** puede ser cualquier objeto iterable, no se requiere que sea un iterador:

```
>>> mock = MagicMock()
>>> mock.__iter__.return_value = ['a', 'b', 'c']
>>> list(mock)
['a', 'b', 'c']
>>> list(mock)
['a', 'b', 'c']
```

Si el valor de retorno *es* un iterador, iterar sobre él una vez que se consume, y cualquier iteración posterior, resultará en una lista vacía:

```
>>> mock.__iter__.return_value = iter(['a', 'b', 'c'])
>>> list(mock)
['a', 'b', 'c']
>>> list(mock)
[]
```

`MagicMock` tiene todos los métodos mágicos soportados configurados a excepción de algunos de los más oscuros y obsoletos. De todas formas, puedes configurar estos también si lo deseas.

Los métodos mágicos que son compatibles, pero que no están configurados por defecto en `MagicMock` son:

- `__subclasses__`
- `__dir__`
- `__format__`
- `__get__`, `__set__` y `__delete__`
- `__reversed__` y `__missing__`
- `__reduce__`, `__reduce_ex__`, `__getinitargs__`, `__getnewargs__`, `__getstate__` y `__setstate__`
- `__getformat__`

Los métodos mágicos *deben* ser buscados en la clase en lugar de en la instancia. Diferentes versiones de Python son inconsistentes sobre la aplicación de esta regla. Los métodos de protocolo soportados deben trabajar con todas las versiones de Python.

La función está básicamente conectada a la clase, pero cada instancia `Mock` se mantiene aislada de las demás.

Ayudantes

sentinel

unittest.mock.sentinel

> El objeto `sentinel` proporciona una manera conveniente de proporcionar objetos únicos para tus pruebas.
>
> Los atributos se crean a demanda, en el instante en el que se accede a ellos por su nombre por primera vez. El acceso al mismo atributo siempre retornará el mismo objeto. Los objetos retornados tienen una reproducción (repr) apropiada para que los mensajes de error de la prueba sean legibles.
>
> *Distinto en la versión 3.7:* Los atributos `sentinel` ahora preservan su identidad cuando son **`copiados`** o **`serializados con pickle`**.

A veces, cuando implementas pruebas, necesitas probar que un determinado objeto se pasa como argumento a otro método, o se retorna. Crear objetos centinela con un nombre para probar esto puede ser una práctica común. `sentinel` proporciona una manera conveniente de crear y probar la identidad de este tipo de objetos.

En el siguiente ejemplo, parcheamos `method` para retornar `sentinel.some_object`:

```
>>> real = ProductionClass()
>>> real.method = Mock(name="method")
>>> real.method.return_value = sentinel.some_object
>>> result = real.method()
>>> assert result is sentinel.some_object
>>> result
sentinel.some_object
```

DEFAULT

unittest.mock.DEFAULT

El objeto `DEFAULT` es un centinela pre-creado (actualmente `sentinel.DEFAULT`). Puede ser usado por las funciones `side_effect` para indicar que el valor de retorno normal debe ser utilizado.

call

unittest.mock.call(**args*, ***kwargs*)

`call()` es un objeto ayudante que puede usarse para hacer aserciones simples, para comparar con `call_args`, `call_args_list`, `mock_calls` y `method_calls`. **`call()`** y también se puede utilizar con `assert_has_calls()`.

```
>>> m = MagicMock(return_value=None)
>>> m(1, 2, a='foo', b='bar')
>>> m()
>>> m.call_args_list == [call(1, 2, a='foo', b='bar
True
```

call.call_list()

> Para un objeto call que representa múltiples llamadas, el método **`call_list()`** retorna una lista con todas las llamadas intermedias, así como la llamada final.

`call_list` es particularmente útil para hacer aserciones sobre «llamadas encadenadas». Una llamada encadenada es varias llamadas en una sola línea de código. Esto resulta en múltiples entradas en el atributo **`mock_calls`** de un objeto simulado. Construir manualmente la secuencia de llamadas puede ser tedioso.

`call_list()` puede construir la secuencia de llamadas desde la misma llamada encadenada:

```
>>> m = MagicMock()
>>> m(1).method(arg='foo').other('bar')(2.0)
<MagicMock name='mock().method().other()()' id='...'>
>>> kall = call(1).method(arg='foo').other('bar')(2.0)
>>> kall.call_list()
[call(1),
 call().method(arg='foo'),
 call().method().other('bar'),
 call().method().other()(2.0)]
>>> m.mock_calls == kall.call_list()
True
```

Un objeto `call`, dependiendo de cómo fuera construido, puede ser una tupla de la forma (argumentos posicionales, argumentos por palabras clave) o bien de la forma (nombre, argumentos posicionales, argumentos por palabras clave). Esto no es particularmente interesante cuando los construyes por ti mismo, pero la introspección de los objetos `call` que conforman los atributos **`Mock.call_args`**, **`Mock.call_args_list`** y **`Mock.mock_calls`** si pueden ser de utilidad para para acceder a los argumentos individuales que contienen.

Los objetos `call` en **`Mock.call_args`** y **`Mock.call_args_list`** están conformados por dos tuplas (argumentos posicionales, argumentos por palabra clave) mientras que los objetos `call` en **`Mock.mock_calls`**, junto con los que construyas por ti mismo, constan de tres tuplas (nombre, argumentos posicionales, argumentos por palabra clave).

Puedes utilizar esta presentación en forma de tuplas para obtener los argumentos individuales, con la finalidad de llevar a cabo introspección y aserciones más complejas. Los argumentos posicionales son una tupla (vacía si no hay ninguno) y los argumentos por palabra clave son un diccionario:

```
>>> m = MagicMock(return_value=None)
>>> m(1, 2, 3, arg='one', arg2='two')
>>> kall = m.call_args
>>> kall.args
(1, 2, 3)
>>> kall.kwargs
{'arg': 'one', 'arg2': 'two'}
>>> kall.args is kall[0]
True
>>> kall.kwargs is kall[1]
True

>>> m = MagicMock()
>>> m.foo(4, 5, 6, arg='two', arg2='three')
<MagicMock name='mock.foo()' id='...'>
>>> kall = m.mock_calls[0]
>>> name, args, kwargs = kall
>>> name
'foo'
>>> args
(4, 5, 6)
>>> kwargs
{'arg': 'two', 'arg2': 'three'}
>>> name is m.mock_calls[0][0]
True
```

create_autospec

unittest.mock.create_autospec(*spec*, *spec_set=False*, *instance=False*, ***kwargs*)

Crea un nuevo objeto simulado utilizando otro objeto (*spec*) como especificación. Los atributos del objeto simulado utilizarán el atributo correspondiente del objeto *spec* como su especificación.

Los argumentos de las funciones o métodos simulados se validarán para asegurarse de que son invocados con la firma correcta.

Si *spec_set* es `True`, tratar de establecer atributos que no existen en el objeto especificado lanzará una excepción **`AttributeError`**.

Si se utiliza una clase como especificación, el valor de retorno del objeto simulado (la instancia de la clase) tendrá la misma especificación. Se puede utilizar una clase como especificación de una instancia pasando `instance=True`. El objeto simulado retornado solo será invocable si las instancias del objeto simulado son también invocables.

`create_autospec()` también acepta argumentos por palabra clave arbitrarios, que son pasados al constructor del objeto simulado creado.

Consultar Autoespecificación para ver ejemplos de cómo utilizar la autoespecificación mediante **`create_autospec()`** y el argumento *autospec* de **`patch()`**.

Distinto en la versión 3.8: **`create_autospec()`** ahora retorna un objeto **`AsyncMock`** si el objetivo a simular es una función asíncrona.

ANY

unittest.mock.ANY

A veces puede que necesites hacer aserciones sobre *algunos* de los argumentos de una llamada al objeto simulado, pero sin preocuparte por el resto, o puede que quieras hacer uso de ellos de forma individual fuera de **`call_args`** y hacer aserciones más complejas con ellos.

Para ignorar ciertos argumentos puedes pasar objetos que se comparan como iguales con *cualquier cosa*. En este caso, las llamadas a `assert_called_with()` y `assert_called_once_with()` tendrán éxito sin importar lo que se haya pasado realmente.

```
>>> mock = Mock(return_value=None)
>>> mock('foo', bar=object())
>>> mock.assert_called_once_with('foo', bar=ANY)
```

`ANY` también se puede utilizar en las comparaciones con las listas de llamadas, como `mock_calls`:

```
>>> m = MagicMock(return_value=None)
>>> m(1)
>>> m(1, 2)
>>> m(object())
>>> m.mock_calls == [call(1), call(1, 2), ANY]
True
```

`ANY` is not limited to comparisons with call objects and so can also be used in test assertions:

```
class TestStringMethods(unittest.TestCase):

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', ANY])
```

FILTER_DIR

unittest.mock.FILTER_DIR

`FILTER_DIR` es una variable de nivel de módulo que controla la forma en que los objetos simulados responden a `dir()`. El valor predeterminado es `True`, que utiliza el filtrado que se describe a continuación para mostrar solo los miembros útiles. Si no le gusta este filtrado o necesita desactivarlo con fines de diagnóstico, configure `mock.FILTER_DIR = False`.

Con el filtrado activado, `dir(some_mock)` mostrará solo atributos útiles y además incluirá cualquier atributo creado dinámicamente, que normalmente no se mostraría. Si el objeto simulado fue creado con un *spec* (o *autospec*) se muestran todos los atributos del objeto original, incluso si no se ha accedido a ellos todavía:

```
>>> dir(Mock())
['assert_any_call',
 'assert_called',
 'assert_called_once',
 'assert_called_once_with',
 'assert_called_with',
 'assert_has_calls',
 'assert_not_called',
 'attach_mock',
 ...
>>> from urllib import request
>>> dir(Mock(spec=request))
['AbstractBasicAuthHandler',
 'AbstractDigestAuthHandler',
 'AbstractHTTPHandler',
 'BaseHandler',
 ...
```

Muchos de los atributos con subrayado y doble subrayado no muy útiles (atributos privados de **`Mock`** y no del objeto real que se está simulando) se han filtrado del resultado de llamar a **`dir()`** en un objeto **`Mock`**. Si no te gusta este comportamiento, puedes desactivarlo estableciendo el modificador a nivel de módulo **`FILTER_DIR`**:

```
>>> from unittest import mock
>>> mock.FILTER_DIR = False
>>> dir(mock.Mock())
['_NonCallableMock__get_return_value',
 '_NonCallableMock__get_side_effect',
 '_NonCallableMock__return_value_doc',
 '_NonCallableMock__set_return_value',
 '_NonCallableMock__set_side_effect',
'__call__',
'__class__',
...
```

Alternatively you can just use `vars(my_mock)` (instance members) and `dir(type(my_mock))` (type members) to bypass the filtering irrespective of **`FILTER_DIR`**.

mock_open

unittest.mock.mock_open(*mock=None*, *read_data=None*)

Una función auxiliar que permite crear un objeto simulado con la finalidad de reemplazar el uso de **`open()`**. Funciona para simular llamadas directas a **`open()`** y para aquellos casos en los que es utilizada como gestor de contexto.

El argumento *mock* es el objeto simulado a configurar. Si es `None` (por defecto) un objeto **`MagicMock`** se creará para ti, con la API limitada a métodos o atributos disponibles en los gestores de fichero estándares.

read_data is a string for the **`read()`**, **`readline()`**, and **`readlines()`** methods of the file handle to return. Calls to those methods will take data from *read_data* until it is depleted. The mock of these methods is pretty simplistic: every time the *mock* is called, the *read_data* is rewound to the start. If you need more control over the data that you are feeding to the tested code you will need to customize this mock for yourself. When that is insufficient, one of the in-memory filesystem packages on PyPI [https://pypi.org] can offer a realistic filesystem for testing.

Distinto en la versión 3.4: Added **`readline()`** and **`readlines()`** support. The mock of **`read()`** changed to consume *read_data* rather than returning it on each call.

Distinto en la versión 3.5: read_data ahora es restablecido en cada llamada al *mock*.

Distinto en la versión 3.8: Added **`__iter__()`** to implementation so that iteration (such as in for loops) correctly consumes *read_data*.

Usar `open()` como gestor de contexto es una buena manera de asegurarse de que los gestores de archivos se cierren correctamente al terminar y se está convirtiendo en una práctica común:

```
with open('/some/path', 'w') as f:
    f.write('something')
```

The issue is that even if you mock out the call to `open()` it is the *returned object* that is used as a context manager (and has `__enter__()` and `__exit__()` called).

Simular gestores de contexto con un `MagicMock` es lo suficientemente común y complicado como para que una función auxiliar sea útil:

```
>>> m = mock_open()
>>> with patch('__main__.open', m):
...     with open('foo', 'w') as h:
...         h.write('some stuff')
...
>>> m.mock_calls
[call('foo', 'w'),
 call().__enter__(),
 call().write('some stuff'),
 call().__exit__(None, None, None)]
>>> m.assert_called_once_with('foo', 'w')
>>> handle = m()
>>> handle.write.assert_called_once_with('some stuff')
```

Y para la lectura de archivos:

```
>>> with patch('__main__.open', mock_open(read_data='bib
...     with open('foo') as h:
...         result = h.read()
...
>>> m.assert_called_once_with('foo')
>>> assert result == 'bibble'
```

Autoespecificación

Autospeccing is based on the existing **`spec`** feature of mock. It limits the api of mocks to the api of an original object (the spec), but it is recursive (implemented lazily) so that attributes of mocks only have the same api as the attributes of the spec. In addition mocked functions / methods have the same call signature as the original so they raise a **`TypeError`** if they are called incorrectly.

Antes de explicar cómo funciona la autoespecificación, he aquí por qué se necesita.

`Mock` is a very powerful and flexible object, but it suffers from a flaw which is general to mocking. If you refactor some of your code, rename members and so on, any tests for code that is still using the *old api* but uses mocks instead of the real objects will still pass. This means your tests can all pass even though your code is broken.

Distinto en la versión 3.5: Before 3.5, tests with a typo in the word assert would silently pass when they should raise an error. You can still achieve this behavior by passing `unsafe=True` to Mock.

Ten en cuenta que esta es otra razón por la que necesitas pruebas de integración además de pruebas unitarias. Probar todo de forma aislada está muy bien, pero si no pruebas cómo están «conectadas entre sí» tus unidades, todavía hay mucho espacio para errores que las pruebas deberían haber detectado.

`unittest.mock` already provides a feature to help with this, called speccing. If you use a class or instance as the **`spec`** for a mock then you can only access attributes on the mock that exist on the real class:

```
>>> from urllib import request
>>> mock = Mock(spec=request.Request)
>>> mock.assret_called_with  # Intentional typo!
Traceback (most recent call last):
 ...
AttributeError: Mock object has no attribute 'assret_cal
```

La especificación solo se aplica al propio objeto simulado, por lo que aún tenemos el mismo problema con cualquiera de los métodos del mismo:

```
>>> mock.header_items()
<mock.Mock object at 0x...>
>>> mock.header_items.assret_called_with()  # Intentiona
```

La autoespecificación resuelve este problema. Puedes pasar `autospec=True` a **`patch()`** / **`patch.object()`** o utilizar la función **`create_autospec()`** para crear un objeto simulado con una especificación. Si utilizas el argumento `autospec=True` de **`patch()`**, el objeto que se va a reemplazar se utiliza como objeto de especificación. Debido a que la especificación se hace «perezosamente» (la especificación se crea en el instante en el que se accede a los atributos del objeto simulado, no antes), se puede utilizar con objetos muy complejos o anidadas (como módulos que importan módulos que, a su vez, importan otros módulos) sin un gran impacto en el rendimiento.

He aquí un ejemplo de ello en acción:

```
>>> from urllib import request
>>> patcher = patch('__main__.request', autospec=True)
>>> mock_request = patcher.start()
>>> request is mock_request
True
>>> mock_request.Request
<MagicMock name='request.Request' spec='Request' id='...
```

You can see that **`request.Request`** has a spec. **`request.Request`** takes two arguments in the constructor (one of which is *self*). Here’s what happens if we try to call it incorrectly:

```
>>> req = request.Request()
Traceback (most recent call last):
 ...
TypeError: <lambda>() takes at least 2 arguments (1 give
```

La especificación también se aplica a las clases instanciadas (es decir, el valor de retorno de los objetos simulados especificados):

```
>>> req = request.Request('foo')
>>> req
<NonCallableMagicMock name='request.Request()' spec='Req
```

`Request` objects are not callable, so the return value of instantiating our mocked out **`request.Request`** is a non-callable mock. With the spec in place any typos in our asserts will raise the correct error:

```
>>> req.add_header('spam', 'eggs')
<MagicMock name='request.Request().add_header()' id='...
>>> req.add_header.assret_called_with  # Intentional typ
Traceback (most recent call last):
 ...
AttributeError: Mock object has no attribute 'assret_cal
>>> req.add_header.assert_called_with('spam', 'eggs')
```

En muchos casos, podrás simplemente añadir `autospec=True` a tus llamadas **`patch()`** existentes y así estar protegido contra errores tipográficos y cambios de la API.

Además de poder usar *autospec* a través de **`patch()`**, existe la función **`create_autospec()`** para crear directamente objetos simulados autoespecificados:

```
>>> from urllib import request
>>> mock_request = create_autospec(request)
>>> mock_request.Request('foo', 'bar')
<NonCallableMagicMock name='mock.Request()' spec='Reques
```

Sin embargo, este no es el comportamiento predeterminado, ya que no está exento de advertencias y restricciones. Con el fin de conocer qué atributos están disponibles en el objeto especificado, autospec tiene que hacer uso de introspección sobre la especificación (acceder a los atributos). A medida que recorres los atributos en el objeto simulado, se produce paralelamente y de forma soterrada un recorrido del objeto original. Si alguno de tus objetos especificados tienen propiedades o descriptores que puedan desencadenar la ejecución de código, quizás no deberías utilizar la autoespecificación. De todas formas, es siempre mucho mejor diseñar tus objetos de modo que la introspección sea segura [4].

A more serious problem is that it is common for instance attributes to be created in the **`__init__()`** method and not to exist on the class at all. *autospec* can't know about any dynamically created attributes and restricts the api to visible attributes.

```
>>> class Something:
...   def __init__(self):
...     self.a = 33
...
>>> with patch('__main__.Something', autospec=True):
...   thing = Something()
...   thing.a
...
Traceback (most recent call last):
  ...
AttributeError: Mock object has no attribute 'a'
```

Hay diferentes formas de resolver este problema. La forma más sencilla, pero no necesariamente la menos molesta, es simplemente establecer los atributos necesarios en el objeto simulado después de la creación del mismo. El hecho de que *autospec* no te permita buscar atributos que no existen en la especificación no impide que los establezcas manualmente después:

```
>>> with patch('__main__.Something', autospec=True):
...   thing = Something()
...   thing.a = 33
...
```

Existe una versión más agresiva de *spec* y *autospec* que impide establecer atributos inexistentes. Esto es útil si deseas asegurarte de que tu código solo *establece* atributos válidos, pero obviamente evitando este escenario en particular:

```
>>> with patch('__main__.Something', autospec=True, spec
...   thing = Something()
...   thing.a = 33
...
```

```
Traceback (most recent call last):
  ...
AttributeError: Mock object has no attribute 'a'
```

Probably the best way of solving the problem is to add class attributes as default values for instance members initialised in **`__init__()`**. Note that if you are only setting default attributes in **`__init__()`** then providing them via class attributes (shared between instances of course) is faster too. e.g.

```
class Something:
    a = 33
```

Esto nos plantea otro problema. Es relativamente común proporcionar un valor predeterminado de `None` para los miembros que posteriormente se convertirán en objetos de un tipo diferente. `None` es inútil como especificación dado que no permite acceder a *ningún* atributo o método. Como `None` *nunca* será útil como especificación, y probablemente indica un miembro que normalmente será de algún otro tipo en el futuro, la autoespecificación no usa una especificación para aquellos miembros configurados con `None` como valor. Estos serán simplemente objetos simulados ordinarios (MagicMocks en realidad):

```
>>> class Something:
...     member = None
...
>>> mock = create_autospec(Something)
>>> mock.member.foo.bar.baz()
<MagicMock name='mock.member.foo.bar.baz()' id='...'>
```

Si modificar tus clases en producción para agregar valores predeterminados no es de tu agrado, dispones de otras opciones. Una de ellas es simplemente usar una instancia como especificación en lugar de la clase. La otra es crear una subclase de la clase en producción y agregar los valores predeterminados a la subclase, sin afectar a la clase en producción. Ambas alternativas requieren que uses un objeto alternativo como especificación. Afortunadamente **`patch()`** admite esto, simplemente tienes que pasar el objeto alternativo mediante el argumento *autospec*:

```
>>> class Something:
...   def __init__(self):
...     self.a = 33
...
>>> class SomethingForTest(Something):
...   a = 33
...
>>> p = patch('__main__.Something', autospec=SomethingFo
>>> mock = p.start()
>>> mock.a
<NonCallableMagicMock name='Something.a' spec='int' id='
```

Esto solo se aplica a las clases u objetos ya instanciados. Llamar a una clase simulada para crear una instancia simulada *no* crea una instancia real. Solo se llevan a cabo las búsqueda de atributos (mediante llamadas a `dir()`).

Sellar objetos simulados

unittest.mock.seal(*mock*)

> Seal desactivará la creación automática de objetos simulados al acceder a un atributo del objeto simulado que está siendo sellado, o a cualquiera de sus atributos que ya sean objetos simulados de forma recursiva.
>
> Si una instancia simulada con un nombre o una especificación es asignada a un atributo no será considerada en la cadena de sellado. Esto permite evitar que seal fije partes del objeto simulado.

```
>>> mock = Mock()
>>> mock.submock.attribute1 = 2
>>> mock.not_submock = mock.Mock(name="sample_name"
>>> seal(mock)
>>> mock.new_attribute  # This will raise Attribute
>>> mock.submock.attribute2  # This will raise Attr
>>> mock.not_submock.attribute2  # This won't raise
```

Added in version 3.7.

Order of precedence of `side_effect`, `return_value` and *wraps*

The order of their precedence is:

1. `side_effect`
2. `return_value`
3. *wraps*

If all three are set, mock will return the value from `side_effect`, ignoring `return_value` and the wrapped object altogether. If any two are set, the one with the higher precedence will return the value. Regardless of the order of which was set first, the order of precedence remains unchanged.

```
>>> from unittest.mock import Mock
>>> class Order:
...     @staticmethod
...     def get_value():
...         return "third"
...
>>> order_mock = Mock(spec=Order, wraps=Order)
>>> order_mock.get_value.side_effect = ["first"]
>>> order_mock.get_value.return_value = "second"
>>> order_mock.get_value()
'first'
```

As `None` is the default value of `side_effect`, if you reassign its value back to `None`, the order of precedence will be checked between `return_value` and the wrapped object, ignoring **`side_effect`**.

```
>>> order_mock.get_value.side_effect = None
>>> order_mock.get_value()
'second'
```

If the value being returned by `side_effect` is `DEFAULT`, it is ignored and the order of precedence moves to the successor to obtain the value to return.

```
>>> from unittest.mock import DEFAULT
>>> order_mock.get_value.side_effect = [DEFAULT]
>>> order_mock.get_value()
'second'
```

When **`Mock`** wraps an object, the default value of **`return_value`** will be **`DEFAULT`**.

```
>>> order_mock = Mock(spec=Order, wraps=Order)
>>> order_mock.return_value
sentinel.DEFAULT
>>> order_mock.get_value.return_value
sentinel.DEFAULT
```

The order of precedence will ignore this value and it will move to the last successor which is the wrapped object.

As the real call is being made to the wrapped object, creating an instance of this mock will return the real instance of the class. The positional arguments, if any, required by the wrapped object must be passed.

```
>>> order_mock_instance = order_mock()
>>> isinstance(order_mock_instance, Order)
True
>>> order_mock_instance.get_value()
'third'

>>> order_mock.get_value.return_value = DEFAULT
>>> order_mock.get_value()
'third'

>>> order_mock.get_value.return_value = "second"
>>> order_mock.get_value()
'second'
```

But if you assign `None` to it, this will not be ignored as it is an explicit assignment. So, the order of precedence will not move to the wrapped object.

```
>>> order_mock.get_value.return_value = None
>>> order_mock.get_value() is None
True
```

Even if you set all three at once when initializing the mock, the order of precedence remains the same:

```
>>> order_mock = Mock(spec=Order, wraps=Order,
...                   **{"get_value.side_effect": ["firs
...                      "get_value.return_value": "seco
...                   )
...
>>> order_mock.get_value()
'first'
>>> order_mock.get_value.side_effect = None
>>> order_mock.get_value()
'second'
>>> order_mock.get_value.return_value = DEFAULT
>>> order_mock.get_value()
'third'
```

If **`side_effect`** is exhausted, the order of precedence will not cause a value to be obtained from the successors. Instead, `StopIteration` exception is raised.

```
>>> order_mock = Mock(spec=Order, wraps=Order)
>>> order_mock.get_value.side_effect = ["first side effe
...                                     "another side ef
>>> order_mock.get_value.return_value = "second"

>>> order_mock.get_value()
'first side effect value'
>>> order_mock.get_value()
'another side effect value'

>>> order_mock.get_value()
Traceback (most recent call last):
 ...
```

```
StopIteration
```

[1][2]

`unittest.mock` — getting started

Added in version 3.3.

Usando mock

Mock patching methods

Usos comunes para objetos **`Mock`** incluye:

- Métodos de parcheo
- Métodos de grabación de llamadas sobre objetos

Es posible que desee reemplazar un método en un objeto para comprobar que se llama con los argumentos correctos por otra parte del sistema:

```
>>> real = SomeClass()
>>> real.method = MagicMock(name='method')
>>> real.method(3, 4, 5, key='value')
<MagicMock name='method()' id='...'>
```

Una vez que se ha utilizado nuestro mock (`real.method` en este ejemplo) tiene métodos y atributos que le permiten hacer afirmaciones sobre cómo se ha utilizado.

Nota

En la mayoría de estos ejemplos, las clases **`Mock`** y **`MagicMock`** son intercambiables. Como el `MagicMock` es la clase más capaz, hace que sea sensato usarlo por defecto.

Una vez que el mock ha sido llamado su atributo **`called`** se establece en `True`. Lo que es más importante, podemos usar el método **`assert_called_with()`** o **`assert_called_once_with()`** para comprobar que se llamó con los argumentos correctos.

En este ejemplo se prueba que llamar a `ProductionClass().method` da como resultado una llamada al método `something`:

```
>>> class ProductionClass:
...     def method(self):
...         self.something(1, 2, 3)
...     def something(self, a, b, c):
...         pass
...
>>> real = ProductionClass()
>>> real.something = MagicMock()
>>> real.method()
>>> real.something.assert_called_once_with(1, 2, 3)
```

Mock for method calls on an object

En el último ejemplo, parcheamos un método directamente en un objeto para comprobar que se llamó correctamente. Otro caso de uso común es pasar un objeto a un método (o a alguna parte del sistema sometido a prueba) y, a continuación, comprobar que se utiliza de la manera correcta.

La sencilla `ProductionClass` a continuación tiene un método `closer`. Si se llama con un objeto, entonces llama a `close` en él.

```
>>> class ProductionClass:
...     def closer(self, something):
...         something.close()
...
```

Así que para probarlo necesitamos pasar un objeto con un método `close` y comprobar que se llamó correctamente.

```
>>> real = ProductionClass()
>>> mock = Mock()
>>> real.closer(mock)
>>> mock.close.assert_called_with()
```

No tenemos que hacer ningún trabajo para proporcionar el método de “close” en nuestro mock. El acceso al cierre lo crea. Por lo tanto, si “close” aún no se ha llamado, entonces acceder a él en la prueba lo creará, pero **`assert_called_with()`** lanzará una excepción de error.

Mocking classes

Un caso de uso común es simular las clases que crea la instancia del código que se está probando. Cuando se aplica un parche a una clase, esa clase se reemplaza por un mock. Las instancias se crean *llamando a la clase*. Esto significa que tiene acceso a la «instancia mock» examinando el valor devuelto de la clase simulada.

En el ejemplo siguiente tenemos una función `some_function` que crea una instancia de `Foo` y llama a un método en él. La llamada a **`patch()`** reemplaza la clase `Foo` con un mock. La instancia `Foo` es el resultado de llamar al mock, por lo que se configura modificando el mock **`-Mock.return_value`**.

```
>>> def some_function():
...     instance = module.Foo()
...     return instance.method()
...
>>> with patch('module.Foo') as mock:
...     instance = mock.return_value
...     instance.method.return_value = 'the result'
...     result = some_function()
...     assert result == 'the result'
```

Nombrando tus mocks

Puede ser útil poner un nombre a tus mocks. El nombre se muestra en la reproducción del mock y puede ser útil cuando el mock aparece en los mensajes de error de la prueba. El nombre también se propaga a los atributos o métodos del mock:

```
>>> mock = MagicMock(name='foo')
>>> mock
<MagicMock name='foo' id='...'>
>>> mock.method
<MagicMock name='foo.method' id='...'>
```

Tracking all calls

A menudo, desea realizar un seguimiento de más de una llamada a un método. El atributo **`mock_calls`** registra todas las llamadas a los atributos secundarios del mock, y también a sus hijos.

```
>>> mock = MagicMock()
>>> mock.method()
<MagicMock name='mock.method()' id='...'>
>>> mock.attribute.method(10, x=53)
<MagicMock name='mock.attribute.method()' id='...'>
>>> mock.mock_calls
[call.method(), call.attribute.method(10, x=53)]
```

Si realiza una afirmación sobre `mock_calls` y se ha llamado a cualquier método inesperado, la aserción fallará. Esto es útil porque además de afirmar que se han realizado las llamadas que esperaba, también está comprobando que se hicieron en el orden correcto y sin llamadas adicionales:

Utiliza el objeto **`call`** para construir listas y compararlas con `mock_calls`:

```
>>> expected = [call.method(), call.attribute.method(10,
>>> mock.mock_calls == expected
True
```

Sin embargo, los parámetros de las llamadas que devuelven mocks no se registran, lo que significa que no es posible realizar un seguimiento de las llamadas anidadas donde los parámetros utilizados para crear ancestros son importantes:

```
>>> m = Mock()
>>> m.factory(important=True).deliver()
```

```
<Mock name='mock.factory().deliver()' id='...'>
>>> m.mock_calls[-1] == call.factory(important=False).de
True
```

Setting return values and attributes

Establecer los valores de retorno en un objeto mock es sumamente fácil:

```
>>> mock = Mock()
>>> mock.return_value = 3
>>> mock()
3
```

Por supuesto, puede hacer lo mismo con los métodos en el mock:

```
>>> mock = Mock()
>>> mock.method.return_value = 3
>>> mock.method()
3
```

El valor devuelto también se puede establecer en el constructor:

```
>>> mock = Mock(return_value=3)
>>> mock()
3
```

Si necesitas una configuración de atributo en su mock, simplemente haga:

```
>>> mock = Mock()
>>> mock.x = 3
>>> mock.x
3
```

A veces desea simular una situación más compleja, como por ejemplo `mock.connection.cursor().execute("SELECT 1")`. Si esperamos que esta llamada devuelva una lista, entonces tenemos que configurar el resultado de la llamada anidada.

Podemos usar **`call`** para construir el conjunto de llamadas en una «llamada encadenada» como esta para una fácil afirmación después:

```
>>> mock = Mock()
>>> cursor = mock.connection.cursor.return_value
>>> cursor.execute.return_value = ['foo']
>>> mock.connection.cursor().execute("SELECT 1")
['foo']
>>> expected = call.connection.cursor().execute("SELECT
>>> mock.mock_calls
[call.connection.cursor(), call.connection.cursor().exec
>>> mock.mock_calls == expected
True
```

Es la llamada a `.call_list()` la que convierte nuestro objeto de llamada en una lista de llamadas que representan las llamadas encadenadas.

Generar excepciones con mocks

Un atributo útil es **`side_effect`**. Si establece esto en una clase o instancia de excepción, se producirá la excepción cuando se llame al mock.

```
>>> mock = Mock(side_effect=Exception('Boom!'))
>>> mock()
Traceback (most recent call last):
  ...
Exception: Boom!
```

Funciones de efectos secundarios e iterables

`side_effect` también se puede asignar en una función o en una iterable. El caso de uso para `side_effect` como un iterable es donde se va a llamar a su mock varias veces, y desea que cada llamada devuelva un valor diferente. Cuando se asigna `side_effect` en un iterable cada llamada al mock devuelve el siguiente valor de lo iterable:

```
>>> mock = MagicMock(side_effect=[4, 5, 6])
>>> mock()
```

```
4
>>> mock()
5
>>> mock()
6
```

Para casos de uso más avanzados, como variar dinámicamente los valores devueltos en función de cómo se llame al mock, `side_effect` puede ser una función. Se llamará a la función con los mismos argumentos que el mock. Lo que sea que la función devuelve es lo que devuelve la llamada:

```
>>> vals = {(1, 2): 1, (2, 3): 2}
>>> def side_effect(*args):
...     return vals[args]
...
>>> mock = MagicMock(side_effect=side_effect)
>>> mock(1, 2)
1
>>> mock(2, 3)
2
```

Iteradores asincrónicos de Mocking

Desde Python 3.8, `AsyncMock` y `MagicMock` tienen soporte para mock Iteradores asíncronos through `__aiter__`. El **`return_value`** atributo de `__aiter__` puede ser usado para asignar los valores de retorno que podrían ser usados por iteración.

```
>>> mock = MagicMock()  # AsyncMock also works here
>>> mock.__aiter__.return_value = [1, 2, 3]
>>> async def main():
...     return [i async for i in mock]
...
>>> asyncio.run(main())
[1, 2, 3]
```

El gestor de contexto asincrónico de Mocking

Desde Python 3.8, `AsyncMock` y `MagicMock` tienen soporte para mock Gestores de contexto asíncronos a través de `__aenter__` y `__aexit__`. De forma predeterminada, las instancias `__aenter__` y `__aexit__` son instancias de `AsyncMock` que devuelven una función asincrónica.

```
>>> class AsyncContextManager:
...     async def __aenter__(self):
...         return self
...     async def __aexit__(self, exc_type, exc, tb):
...         pass
...
>>> mock_instance = MagicMock(AsyncContextManager())  #
>>> async def main():
...     async with mock_instance as result:
...         pass
...
>>> asyncio.run(main())
>>> mock_instance.__aenter__.assert_awaited_once()
>>> mock_instance.__aexit__.assert_awaited_once()
```

Creating a mock from an existing object

Un problema con el uso excesivo de mocking es que combina sus pruebas a la implementación de sus mocks en lugar de su código real. Supongamos que tiene una clase que implementa `some_method`. En una prueba para otra clase, se proporciona un mock de este objeto que *also* proporciona `some_method`. Si más tarde refactoriza la primera clase, para que ya no tenga `some_method` - entonces sus pruebas seguirán pasando a pesar de que su código está ahora roto!

`Mock` le permite proporcionar un objeto como especificación para el mock, utilizando el argumento de palabra clave *spec*. El acceso a métodos / atributos en el mock que no existen en el objeto de especificación lanzará inmediatamente un error de atributo. Si cambia la implementación de la especificación, las pruebas que usan esa clase comenzarán a fallar inmediatamente sin tener que crear instancias de la clase en esas pruebas.

```
>>> mock = Mock(spec=SomeClass)
>>> mock.old_method()
Traceback (most recent call last):
   ...
AttributeError: Mock object has no attribute 'old_method
```

El uso de una especificación también permite una coincidencia más inteligente de las llamadas realizadas al mock, independientemente de si algunos parámetros se pasaron como argumentos posicionales o con nombre:

```
>>> def f(a, b, c): pass
...
>>> mock = Mock(spec=f)
>>> mock(1, 2, 3)
<Mock name='mock()' id='140161580456576'>
>>> mock.assert_called_with(a=1, b=2, c=3)
```

Si desea que esta coincidencia más inteligente también funcione con llamadas de método en el mock, puede usar auto-speccing.

Si desea una forma más fuerte de especificación que impida la configuración de atributos arbitrarios, así como la obtención de ellos, entonces puede usar *spec_set* en lugar de *spec*.

Uso de side_effect para devolver el contenido por archivo

`mock_open()` se utiliza para parchear el método **`open()`**. **`side_effect`** se puede utilizar para devolver un nuevo objeto Mock por llamada. Esto puede usarse para devolver diferentes contenidos por fichero almacenado en un diccionario:

```
DEFAULT = "default"
data_dict = {"file1": "data1",
             "file2": "data2"}

def open_side_effect(name):
    return mock_open(read_data=data_dict.get(name, DEFAU
with patch("builtins.open", side_effect=open_side_effect
    with open("file1") as file1:
        assert file1.read() == "data1"

    with open("file2") as file2:
        assert file2.read() == "data2"

    with open("file3") as file2:
        assert file2.read() == "default"
```

Patch decorators

Nota

Con **`patch()`** importa que parchee objetos en el espacio de nombres donde se buscan. Esto es normalmente sencillo, pero para una guía rápida lea where to patch.

Una necesidad común en las pruebas es aplicar revisiones a un atributo de clase o a un atributo de módulo, por ejemplo, aplicar revisiones a una clase integrada o parchear una clase en un módulo para probar que se crea una instancia. Los módulos y las clases son efectivamente globales, por lo que el parcheo en ellos tiene que deshacerse después de la prueba o el parche persistirá en otras pruebas y causará problemas difíciles de diagnosticar.

mock proporciona tres decoradores convenientes para esto: **`patch()`**, **`patch.object()`** y **`patch.dict()`**. `patch` toma una sola cadena del formulario `package.module.Class.attribute` para especificar el atributo que está parcheando. También, opcionalmente, toma un valor con el que desea que se reemplace el atributo (o clase o lo que sea). “patch.object” toma un objeto y el nombre del atributo con el que desea parchear, además, opcionalmente, del valor con el que parcharlo.

`patch.object`:

```
>>> original = SomeClass.attribute
>>> @patch.object(SomeClass, 'attribute', sentinel.attri
... def test():
...     assert SomeClass.attribute == sentinel.attribute
...
>>> test()
>>> assert SomeClass.attribute == original

>>> @patch('package.module.attribute', sentinel.attribut
... def test():
...     from package.module import attribute
...     assert attribute is sentinel.attribute
...
>>> test()
```

Si está parcheando un módulo (incluyendo **`builtins`**) entonces use **`patch()`** en lugar de **`patch.object()`**:

```
>>> mock = MagicMock(return_value=sentinel.file_handle)
>>> with patch('builtins.open', mock):
...     handle = open('filename', 'r')
...
>>> mock.assert_called_with('filename', 'r')
>>> assert handle == sentinel.file_handle, "incorrect fi
```

EL nombre del módulo puede ser “dotted”, en el formulario `package.module` si es necesario:

```
>>> @patch('package.module.ClassName.attribute', sentine
... def test():
...     from package.module import ClassName
...     assert ClassName.attribute == sentinel.attribute
...
>>> test()
```

Un buen patrón en realidad es decorar los métodos de pruebas propios:

```
>>> class MyTest(unittest.TestCase):
...     @patch.object(SomeClass, 'attribute', sentinel.a
...     def test_something(self):
```

```
...         self.assertEqual(SomeClass.attribute, sentir
...
>>> original = SomeClass.attribute
>>> MyTest('test_something').test_something()
>>> assert SomeClass.attribute == original
```

Si desea parchear con un Mock, puede usar **`patch()`** con un solo argumento (o **`patch.object()`** con dos argumentos). El mock se creará para usted y se pasará a la función de prueba / método:

```
>>> class MyTest(unittest.TestCase):
...     @patch.object(SomeClass, 'static_method')
...     def test_something(self, mock_method):
...         SomeClass.static_method()
...         mock_method.assert_called_with()
...
>>> MyTest('test_something').test_something()
```

Puede apilar varios decoradores de parches utilizando este patrón:

```
>>> class MyTest(unittest.TestCase):
...     @patch('package.module.ClassName1')
...     @patch('package.module.ClassName2')
...     def test_something(self, MockClass2, MockClass1)
...         self.assertIs(package.module.ClassName1, Moc
...         self.assertIs(package.module.ClassName2, Moc
...
>>> MyTest('test_something').test_something()
```

Al anidar decoradores de parches, los mocks se pasan a la función decorada en el mismo orden en que se aplicaron (el orden normal *Python* que se aplican los decoradores). Esto significa de abajo hacia arriba, así que en el ejemplo anterior el simulacro de `test_module.ClassName2` se pasa primero.

También está **`patch.dict()`** para establecer valores en un diccionario solo durante un ámbito y restaurar el diccionario a su estado original cuando finaliza la prueba:

```
>>> foo = {'key': 'value'}
>>> original = foo.copy()
>>> with patch.dict(foo, {'newkey': 'newvalue'}, clear=T
...     assert foo == {'newkey': 'newvalue'}
...
>>> assert foo == original
```

`patch`, `patch.object` and `patch.dict` se pueden utilizar como gestores de contexto.

Donde utilice **`patch()`** para crear un simulacro para usted, puede obtener una referencia al simulacro utilizando la forma «as» de la instrucción with:

```
>>> class ProductionClass:
...     def method(self):
...         pass
...
>>> with patch.object(ProductionClass, 'method') as mock
...     mock_method.return_value = None
...     real = ProductionClass()
...     real.method(1, 2, 3)
...
>>> mock_method.assert_called_with(1, 2, 3)
```

Como alternativa `patch`, `patch.object` and `patch.dict` se pueden utilizar como decoradores de clase. Cuando se utiliza de esta manera es lo mismo que aplicar el decorador individualmente a cada método cuyo nombre comienza con «test».

Further examples

Estos son algunos ejemplos más para algunos escenarios ligeramente más avanzados.

Mocking de llamadas encadenadas

Mocking de las llamadas encadenadas es en realidad sencillo con mock una vez que entiende el atributo **`return_value`** . Cuando se llama a un mock por primera vez, o se obtiene su `return_value` antes de que se llame, se crea un nuevo **`Mock`**.

Esto significa que puede ver cómo se ha utilizado el objeto devuelto de una llamada a un objeto simulado interrogando el simulado `return_value`:

```
>>> mock = Mock()
>>> mock().foo(a=2, b=3)
<Mock name='mock().foo()' id='...'>
>>> mock.return_value.foo.assert_called_with(a=2, b=3)
```

Desde aquí es un paso simple para configurar y luego hacer aserciones sobre llamadas encadenadas. Por supuesto, otra alternativa es escribir su código de una manera más comprobable en primer lugar…

Por lo tanto, supongamos que tenemos algún código que se ve un poco como este:

```
>>> class Something:
...     def __init__(self):
...         self.backend = BackendProvider()
...     def method(self):
...         response = self.backend.get_endpoint('foobar
...         # more code
```

Suponiendo que `BackendProvider` ya está bien probado, ¿cómo probamos `method()`? En concreto, queremos probar que la sección de código `# more code` usa el objeto de respuesta de la manera correcta.

A medida que esta cadena de llamadas se realiza a partir de un atributo de instancia, podemos parchear el atributo `backend` en una instancia de `Something`. En este caso en particular sólo estamos interesados en el valor devuelto de la llamada final a `start_call` por lo que no tenemos mucha configuración que hacer. Supongamos que el objeto que devuelve es “similar a un archivo”, por lo que nos aseguraremos de que nuestro objeto de respuesta utilice el compilado **`open()`** as its `spec`.

Para ello creamos una instancia mock como nuestro back-end simulado y creamos un objeto de respuesta mock para ella. Para establecer la respuesta como el valor devuelto de ese `start_call` final podríamos hacer lo siguiente:

```
mock_backend.get_endpoint.return_value.create_call.retur
```

Podemos hacerlo de una manera un poco mejor usando el método **`configure_mock()`** para establecer directamente el valor devuelto para nosotros:

```
>>> something = Something()
>>> mock_response = Mock(spec=open)
>>> mock_backend = Mock()
>>> config = {'get_endpoint.return_value.create_call.ret
>>> mock_backend.configure_mock(**config)
```

Con estos monkey patch el «backend simulado» en su lugar y puede hacer la llamada real:

```
>>> something.backend = mock_backend
>>> something.method()
```

Usando **`mock_calls`** podemos comprobar la llamada encadenada con una sola afirmación. Una llamada encadenada es varias llamadas en una línea de código, por lo que habrá varias entradas en `mock_calls`. Podemos usar **`call.call_list()`** para crear esta lista de llamadas para nosotros:

```
>>> chained = call.get_endpoint('foobar').create_call('s
>>> call_list = chained.call_list()
>>> assert mock_backend.mock_calls == call_list
```

Mocking parcial

For some tests, you may want to mock out a call to **`datetime.date.today()`** to return a known date, but don't want to prevent the code under test from creating new date objects. Unfortunately **`datetime.date`** is written in C, so you cannot just monkey-patch out the static **`datetime.date.today()`** method.

Instead, you can effectively wrap the date class with a mock, while passing through calls to the constructor to the real class (and returning real instances).

El **`patch decorator`** se utiliza aquí como un mock de la clase `date` en el módulo en pruebas. Entonces, el atributo **`side_effect`** de la clase de fecha simulada se establece en una función lambda que devuelve una fecha real. Cuando la clase de fecha simulada es llamada, una fecha real se construirá y devolverá por `side_effect`.

```
>>> import datetime as dt
>>> with patch('mymodule.date') as mock_date:
...     mock_date.today.return_value = dt.date(2010, 10,
...     mock_date.side_effect = lambda *args, **kw: dt.
...
...     assert mymodule.date.today() == dt.date(2010, 1
...     assert mymodule.date(2009, 6, 8) == dt.date(200
```

Tenga en cuenta que no parcheamos **`datetime.date`** globalmente, parcheamos `date` en el módulo que *uses*. Consulte where to patch.

Cuando `date.today()` es llamada se retorna una fecha conocida, pero llama al constructor `date(...)` este constructor todavía devuelve fechas normales. Sin esto, puede encontrarse teniendo que calcular un resultado esperado utilizando exactamente el mismo algoritmo que el código en prueba, que es un anti-patrón de prueba clásico.

Las llamadas al constructor de fecha se registran en los atributos `mock_date` (`call_count` y amigos) que también pueden ser útiles para las pruebas.

Una forma alternativa de tratar con fechas de mock, u otras clases integradas, se discute en esta entrada de blog [https://williambert.online/2011/07/how-to-unit-testing-in-django-with-mocking-and-patching/].

Mocking a generator method

Un generador de Python es una función o método que utiliza la instrucción **`yield`** para devolver una serie de valores cuando se itera sobre [1].

Se llama a un método / función del generador para devolver el objeto generador. Es el objeto generador que luego se itera. El método de protocolo para la iteración es **`__iter__()`**, por lo que podemos hacer mock de esto usando una **`MagicMock`**.

Aquí hay una clase de ejemplo con un método «iter» implementado como generador:

```
>>> class Foo:
...     def iter(self):
...         for i in [1, 2, 3]:
...             yield i
...
>>> foo = Foo()
>>> list(foo.iter())
[1, 2, 3]
```

¿Cómo haríamos un mock de esta clase y, en particular, de su método «iter»?

Para configurar los valores devueltos desde la iteración (implícita en la llamada a **`list`**), necesitamos configurar el objeto devuelto por la llamada a `foo.iter()`.

```
>>> mock_foo = MagicMock()
>>> mock_foo.iter.return_value = iter([1, 2, 3])
>>> list(mock_foo.iter())
[1, 2, 3]
```

También hay expresiones generadoras y más usos avanzados <http://www.dabeaz.com/coroutines/index.html>”_ de generadores, pero no nos preocupan los que están aquí. Una muy buena introducción a los generadores y lo potentes que son es: Generator Tricks for Systems Programmers [http://www.dabeaz.com/generators/].

Aplicar el mismo parche a cada método de prueba

Si desea que se coloquen varias revisiones para varios métodos de prueba, la forma obvia es aplicar los decoradores de parches a cada método. Esto puede parecer una repetición innecesaria. En cambio, puede utilizar **`patch()`** (en todas sus diversas formas) como decorador de clases. Esto aplica las revisiones a todos los métodos de prueba de la clase. Un método de prueba se identifica mediante métodos cuyos nombres comienzan con `test`:

```
>>> @patch('mymodule.SomeClass')
... class MyTest(unittest.TestCase):
...
...     def test_one(self, MockSomeClass):
...         self.assertIs(mymodule.SomeClass, MockSomeCl
...
...     def test_two(self, MockSomeClass):
...         self.assertIs(mymodule.SomeClass, MockSomeCl
...
...     def not_a_test(self):
...         return 'something'
...
>>> MyTest('test_one').test_one()
>>> MyTest('test_two').test_two()
>>> MyTest('test_two').not_a_test()
'something'
```

Una forma alternativa de administrar parches es usar métodos start y stop de patch. Estos le permiten mover el parche en sus métodos `setUp` y `tearDown`.

```
>>> class MyTest(unittest.TestCase):
...     def setUp(self):
...         self.patcher = patch('mymodule.foo')
...         self.mock_foo = self.patcher.start()
...
...     def test_foo(self):
...         self.assertIs(mymodule.foo, self.mock_foo)
...
...     def tearDown(self):
...         self.patcher.stop()
```

```
...
>>> MyTest('test_foo').run()
```

Si utiliza esta técnica, debe asegurarse de que la aplicación de parches se «undone» llamando a `stop`. Esto puede ser más complicado de lo que podría pensar, porque si se produce una excepción en el setUp, no se llama a tearDown. **`unittest.TestCase.addCleanup()`** hace que esto sea más fácil:

```
>>> class MyTest(unittest.TestCase):
...     def setUp(self):
...         patcher = patch('mymodule.foo')
...         self.addCleanup(patcher.stop)
...         self.mock_foo = patcher.start()
...
...     def test_foo(self):
...         self.assertIs(mymodule.foo, self.mock_foo)
...
>>> MyTest('test_foo').run()
```

Mocking unbound methods

Sometimes a test needs to patch an *unbound method*, which means patching the method on the class rather than on the instance. In order to make assertions about which objects were calling this particular method, you need to pass `self` as the first argument. The issue is that you can’t patch with a mock for this, because if you replace an unbound method with a mock it doesn’t become a bound method when fetched from the instance, and so it doesn’t get `self` passed in. The workaround is to patch the unbound method with a real function instead. The **`patch()`** decorator makes it so simple to patch out methods with a mock that having to create a real function becomes a nuisance.

If you pass `autospec=True` to patch then it does the patching with a *real* function object. This function object has the same signature as the one it is replacing, but delegates to a mock under the hood. You still get your mock auto-created in exactly the same way as before. What it means though, is that if you use it to patch out an unbound method on a class the mocked function will be turned into a bound method if it is fetched from an instance. It will have `self` passed in as the first argument, which is exactly what was needed:

```
>>> class Foo:
...   def foo(self):
...     pass
...
>>> with patch.object(Foo, 'foo', autospec=True) as mock
...   mock_foo.return_value = 'foo'
...   foo = Foo()
...   foo.foo()
...
'foo'
>>> mock_foo.assert_called_once_with(foo)
```

Si no usamos `autospec=True` entonces el método independiente se parchea con una instancia de Mock en su lugar, y no se llama con `self`.

Comprobación de varias llamadas con mock

mock tiene una buena API para hacer aserciones sobre cómo se usan sus objetos ficticios.

```
>>> mock = Mock()
>>> mock.foo_bar.return_value = None
>>> mock.foo_bar('baz', spam='eggs')
>>> mock.foo_bar.assert_called_with('baz', spam='eggs')
```

Si su mock solo se llama una vez, puede usar el método **`assert_called_once_with()`** que también afirma que **`call_count`** es uno.

```
>>> mock.foo_bar.assert_called_once_with('baz', spam='eg
>>> mock.foo_bar()
>>> mock.foo_bar.assert_called_once_with('baz', spam='eg
Traceback (most recent call last):
    ...
```

```
AssertionError: Expected 'foo_bar' to be called once. Ca
Calls: [call('baz', spam='eggs'), call()].
```

Tanto `assert_called_with` como `assert_called_once_with` hacen afirmaciones sobre la llamada *más reciente*. Si su mock va a ser llamado varias veces, y desea hacer aserciones sobre *todas* esas llamadas que puede utilizar **`call_args_list`**:

```
>>> mock = Mock(return_value=None)
>>> mock(1, 2, 3)
>>> mock(4, 5, 6)
>>> mock()
>>> mock.call_args_list
[call(1, 2, 3), call(4, 5, 6), call()]
```

El helper **`call`** facilita la toma de aserciones sobre estas llamadas. Puede crear una lista de llamadas esperadas y compararla con `call_args_list`. Esto se ve notablemente similar al repr de la `call_args_list`:

```
>>> expected = [call(1, 2, 3), call(4, 5, 6), call()]
>>> mock.call_args_list == expected
True
```

Copiando con argumentos mutables

Otra situación es rara, pero puede morderte, es cuando se llama a tu mock con argumentos mutables. `call_args` y `call_args_list` almacenan *referencias* a los argumentos. Si el código sometido a prueba muta los argumentos ya no puede realizar aserciones sobre cuáles eran los valores cuando se llamó al mock.

Este es un código de ejemplo que muestra el problema. Imagine las siguientes funciones definidas en "mymodule":

```
def frob(val):
    pass

def grob(val):
    "First frob and then clear val"
    frob(val)
    val.clear()
```

Cuando tratamos de probar que `grob` llama `frob` con el argumento correcto mira lo que sucede:

```
>>> with patch('mymodule.frob') as mock_frob:
...     val = {6}
...     mymodule.grob(val)
...
>>> val
set()
>>> mock_frob.assert_called_with({6})
Traceback (most recent call last):
    ...
AssertionError: Expected: (({6},), {})
Called with: ((set(),), {})
```

Una posibilidad sería que el mock copiara los argumentos que pasa. Esto podría causar problemas si realiza aserciones que se basan en la identidad del objeto para la igualdad.

Aquí hay una solución que utiliza la funcionalidad **`side_effect`**. Si proporciona una función `side_effect` para un mock, se llamará a `side_effect` con los mismos argumentos que el mock. Esto nos da la oportunidad de copiar los argumentos y almacenarlos para aserciones posteriores. En este ejemplo estoy usando *otro* mock para almacenar los argumentos de modo que pueda usar los métodos ficticios para hacer la aserción. Una vez más, una función auxiliar configura esto para mí.

```
>>> from copy import deepcopy
>>> from unittest.mock import Mock, patch, DEFAULT
>>> def copy_call_args(mock):
...     new_mock = Mock()
...     def side_effect(*args, **kwargs):
...         args = deepcopy(args)
...         kwargs = deepcopy(kwargs)
...         new_mock(*args, **kwargs)
...             return DEFAULT
...         mock.side_effect = side_effect
...         return new_mock
...
>>> with patch('mymodule.frob') as mock_frob:
...         new_mock = copy_call_args(mock_frob)
...         val = {6}
...         mymodule.grob(val)
...
>>> new_mock.assert_called_with({6})
>>> new_mock.call_args
call({6})
```

`copy_call_args` se llama con el mock que se llamará. Devuelve un nuevo mock en el que hacemos la aserción. La función `side_effect` hace una copia de los args y llama a nuestro `new_mock` con la copia.

Nota

Si su simulacro solo se va a usar una vez, hay una forma más fácil de verificar los argumentos en el punto en que se llaman. Simplemente puede hacer la comprobación dentro de una función `side_effect`.

```
>>> def side_effect(arg):
...     assert arg == {6}
...
>>> mock = Mock(side_effect=side_effect)
>>> mock({6})
>>> mock(set())
Traceback (most recent call last):
    ...
AssertionError
```

Un enfoque alternativo es crear una subclase de **`Mock`** o **`MagicMock`** que copie (usando **`copy.deepcopy()`**) los argumentos. A continuación se muestra un ejemplo de implementación:

```
>>> from copy import deepcopy
>>> class CopyingMock(MagicMock):
...     def __call__(self, /, *args, **kwargs):
...         args = deepcopy(args)
...         kwargs = deepcopy(kwargs)
...         return super().__call__(*args, **kwargs)
...
>>> c = CopyingMock(return_value=None)
>>> arg = set()
>>> c(arg)
>>> arg.add(1)
>>> c.assert_called_with(set())
>>> c.assert_called_with(arg)
Traceback (most recent call last):
    ...
AssertionError: expected call not found.
Expected: mock({1})
Actual: mock(set())
>>> c.foo
<CopyingMock name='mock.foo' id='...'>
```

Cuando subclases `Mock` o `MagicMock` todos los atributos creados dinámicamente, y el `return_value` usará tu subclase automáticamente. Eso significa que todos los elementos secundarios de un `CopyingMock` también tendrán el tipo `CopyingMock`.

Nesting patches

Usar parches como administradores de contexto es bueno, pero si haces varios parches, puedes terminar con instrucciones anidadas que se sangran cada vez más a la derecha:

```
>>> class MyTest(unittest.TestCase):
...
...     def test_foo(self):
...         with patch('mymodule.Foo') as mock_foo:
...             with patch('mymodule.Bar') as mock_bar:
...                 with patch('mymodule.Spam') as mock_
...                     assert mymodule.Foo is mock_foo
...                         assert mymodule.Bar is mock_bar
...                         assert mymodule.Spam is mock_spa
...
>>> original = mymodule.Foo
>>> MyTest('test_foo').test_foo()
>>> assert mymodule.Foo is original
```

Con las funciones unittest `cleanup` y métodos start y stop de patch podemos lograr el mismo efecto sin la sangría anidada. Un método auxiliar simple, `create_patch`, pone el parche en su lugar y devuelve el mock creado para nosotros:

```
>>> class MyTest(unittest.TestCase):
...
...     def create_patch(self, name):
...         patcher = patch(name)
...         thing = patcher.start()
...         self.addCleanup(patcher.stop)
...         return thing
...
...     def test_foo(self):
...         mock_foo = self.create_patch('mymodule.Foo')
...         mock_bar = self.create_patch('mymodule.Bar')
...         mock_spam = self.create_patch('mymodule.Spam
...
...         assert mymodule.Foo is mock_foo
...         assert mymodule.Bar is mock_bar
...         assert mymodule.Spam is mock_spam
...
>>> original = mymodule.Foo
>>> MyTest('test_foo').run()
>>> assert mymodule.Foo is original
```

Mocking a un diccionario usando MagickMock

Es posible que desee simular un diccionario u otro objeto contenedor, registrando todo el acceso a él mientras todavía se comporta como un diccionario.

Podemos hacer esto con **`MagicMock`**, que se comportará como un diccionario, y usando **`side_effect`** para delegar el acceso del diccionario a un diccionario subyacente real que está bajo nuestro control.

Cuando se llama a los métodos **`__getitem__()`** y **`__setitem__()`** de nuestro `MagicMock` (acceso normal al diccionario), entonces se llama a `side_effect` con la clave (y el valor también en el caso de `__setitem__`). También podemos controlar lo que se devuelve.

Después de que se haya utilizado el `MagicMock` podemos usar atributos como **`call_args_list`** para afirmar cómo se usó el diccionario:

```
>>> my_dict = {'a': 1, 'b': 2, 'c': 3}
>>> def getitem(name):
...      return my_dict[name]
...
>>> def setitem(name, val):
...     my_dict[name] = val
...
>>> mock = MagicMock()
>>> mock.__getitem__.side_effect = getitem
>>> mock.__setitem__.side_effect = setitem
```

Nota

Una alternativa al uso de `MagicMock` es usar `Mock` y *solo* proporcionar los métodos mágicos que desea específicamente:

```
>>> mock = Mock()
>>> mock.__getitem__ = Mock(side_effect=getitem)
>>> mock.__setitem__ = Mock(side_effect=setitem)
```

Una *tercera* opción es usar `MagicMock` pero pasando `dict` como el argumento *spec* (o *spec_set*) para que el `MagicMock` creado solo tenga métodos mágicos de diccionario disponibles:

```
>>> mock = MagicMock(spec_set=dict)
```

```
 >>> mock.__getitem__.side_effect = getitem
 >>> mock.__setitem__.side_effect = setitem
```

Con estas funciones de efectos secundarios en su lugar, el `mock` se comportará como un diccionario normal pero registrando el acceso. Incluso genera un **`KeyError`** si intenta acceder a una clave que no existe.

```
>>> mock['a']
1
>>> mock['c']
3
>>> mock['d']
Traceback (most recent call last):
    ...
KeyError: 'd'
>>> mock['b'] = 'fish'
>>> mock['d'] = 'eggs'
>>> mock['b']
'fish'
>>> mock['d']
'eggs'
```

Una vez utilizado, puede realizar aserciones sobre el acceso utilizando los métodos y atributos mock normales:

```
>>> mock.__getitem__.call_args_list
[call('a'), call('c'), call('d'), call('b'), call('d')]
>>> mock.__setitem__.call_args_list
[call('b', 'fish'), call('d', 'eggs')]
>>> my_dict
{'a': 1, 'b': 'fish', 'c': 3, 'd': 'eggs'}
```

Mock de subclases y sus atributos

Hay varias razones por las que es posible que desee crear subclases **`Mock`** Una razón podría ser agregar métodos auxiliares. Aquí hay un ejemplo simple:

```
>>> class MyMock(MagicMock):
...     def has_been_called(self):
...         return self.called
...
>>> mymock = MyMock(return_value=None)
>>> mymock
<MyMock id='...'>
>>> mymock.has_been_called()
False
>>> mymock()
>>> mymock.has_been_called()
True
```

El comportamiento estándar para las instancias `Mock` es que los atributos y los simulacros de valor devuelto son del mismo tipo que el simulacro en el que se accede a ellos. Esto garantiza que los atributos `Mock` son `Mocks` y los atributos `MagicMock` son `MagicMocks` [2]. Por lo tanto, si está creando subclases para agregar métodos auxiliares, también estarán disponibles en los atributos y el valor devuelto mock de las instancias de su subclase.

```
>>> mymock.foo
<MyMock name='mock.foo' id='...'>
>>> mymock.foo.has_been_called()
False
>>> mymock.foo()
<MyMock name='mock.foo()' id='...'>
>>> mymock.foo.has_been_called()
True
```

A veces esto es inconveniente. Por ejemplo, un usuario [https://code.google.com/archive/p/mock/issues/105] está creando subclases de mock para crear un Twisted adaptor [https://twisted.org/documents/11.0.0/api/twisted.python.components.html]. Tener esto aplicado a los atributos también puede causar errores en realidad.

`Mock` (en todos sus sabores) utiliza un método llamado `_get_child_mock` para crear estos «sub-mocks» para atributos y valores devueltos. Puede evitar que la subclase se utilice para los atributos invalidando este método. La firma es que toma argumentos de palabra clave arbitrarios (`**kwargs`) que luego se pasan al constructor ficticio:

```
>>> class Subclass(MagicMock):
...     def _get_child_mock(self, /, **kwargs):
...         return MagicMock(**kwargs)
...
>>> mymock = Subclass()
>>> mymock.foo
<MagicMock name='mock.foo' id='...'>
>>> assert isinstance(mymock, Subclass)
>>> assert not isinstance(mymock.foo, Subclass)
>>> assert not isinstance(mymock(), Subclass)
```

Una excepción a esta regla son los mock no invocables. Los atributos usan la variante a la que se puede llamar porque, de lo contrario, los simulacros a los que no se puede llamar no podrían tener métodos a los que se puede llamar.

Importaciones de Mocking con patch.dict

Una situación en la que un mocking puede ser difícil es cuando tiene una importación local dentro de una función. Estos son más difíciles de simular porque no están usando un objeto del espacio de nombres del módulo que podemos revisar.

Por lo general, deben evitarse las importaciones locales. A veces se hacen para evitar dependencias circulares, para las que hay *generalmente* una manera mucho mejor de resolver el problema (refactorizar el código) o para evitar «costos iniciales» retrasando la importación. Esto también se puede resolver de mejores maneras que una importación local incondicional (almacenar el módulo como una clase o atributo de módulo y solo hacer la importación en el primer uso).

Aparte de eso, hay una manera de usar `mock` para afectar los resultados de una importación. La importación obtiene un *objeto* del diccionario **`sys.modules`**. Tenga en cuenta que obtiene un *objeto*, que no tiene por qué ser un módulo. La importación de un módulo por primera vez da como resultado que un objeto de módulo se coloque en `sys.modules`, por lo que normalmente cuando importe algo devuelve un modulo. Sin embargo, este no tiene por qué ser el caso.

Esto significa que puede usar **`patch.dict()`** para *temporalmente* poner un mock en su lugar sobre **`sys.modules`**. Cualquier importación mientras este parche está activo recuperará el mock. Cuando el parche está completo (la función decorada sale, el cuerpo de la instrucción with está completo o se llama a `patcher.stop()`) entonces lo que había anteriormente se restaurará de forma segura.

Aquí hay un ejemplo de mock del módulo “fooble”.

```
>>> import sys
>>> mock = Mock()
>>> with patch.dict('sys.modules', {'fooble': mock}):
...    import fooble
...    fooble.blob()
...
<Mock name='mock.blob()' id='...'>
>>> assert 'fooble' not in sys.modules
>>> mock.blob.assert_called_once_with()
```

Como puede ver, el `import fooble` tiene éxito, pero al salir no queda ningún “fooble” en **`sys.modules`**.

Esto también funciona para el formulario `from module import name`:

```
>>> mock = Mock()
>>> with patch.dict('sys.modules', {'fooble': mock}):
...    from fooble import blob
...    blob.blip()
...
<Mock name='mock.blob.blip()' id='...'>
>>> mock.blob.blip.assert_called_once_with()
```

Con un poco más de trabajo también puede simular las importaciones de paquetes:

```
>>> mock = Mock()
>>> modules = {'package': mock, 'package.module': mock.m
>>> with patch.dict('sys.modules', modules):
...     from package.module import fooble
...     fooble()
...
<Mock name='mock.module.fooble()' id='...'>
>>> mock.module.fooble.assert_called_once_with()
```

Seguimiento del orden de las llamadas y de las aserciones de llamadas menos detalladas

La clase **`Mock`** le permite realizar un seguimiento del *orden* de las llamadas a métodos en sus objetos mock a través del atributo **`method_calls`**. Esto no le permite rastrear el orden de las llamadas entre objetos simulados separados, sin embargo, podemos usar **`mock_calls`** para lograr el mismo efecto.

Debido a que los mocks rastrean las llamadas a mocks secundarios en `mock_calls`, y el acceso a un atributo arbitrario de un mock crea un mock secundario, podemos crear nuestros mocks separados de uno primario. Las llamadas a esos mocks hijos se grabarán, en orden, en el `mock_calls` del padre:

```
>>> manager = Mock()
>>> mock_foo = manager.foo
>>> mock_bar = manager.bar

>>> mock_foo.something()
<Mock name='mock.foo.something()' id='...'>
>>> mock_bar.other.thing()
<Mock name='mock.bar.other.thing()' id='...'>

>>> manager.mock_calls
[call.foo.something(), call.bar.other.thing()]
```

A continuación, podemos realizar afirmaciones sobre las llamadas, incluido el orden, comparando con el atributo `mock_calls` en el mock administrador:

```
>>> expected_calls = [call.foo.something(), call.bar.oth
>>> manager.mock_calls == expected_calls
True
```

Si `patch` está creando y poniendo en su lugar, sus mocks, puede adjuntarlos a un mock administrador usando el método **`attach_mock()`** . Después de adjuntar las llamadas se registrarán en `mock_calls` del administrador.

```
>>> manager = MagicMock()
>>> with patch('mymodule.Class1') as MockClass1:
...     with patch('mymodule.Class2') as MockClass2:
...         manager.attach_mock(MockClass1, 'MockClass1'
...         manager.attach_mock(MockClass2, 'MockClass2'
...         MockClass1().foo()
...         MockClass2().bar()
<MagicMock name='mock.MockClass1().foo()' id='...'>
<MagicMock name='mock.MockClass2().bar()' id='...'>
>>> manager.mock_calls
[call.MockClass1(),
call.MockClass1().foo(),
call.MockClass2(),
call.MockClass2().bar()]
```

Si se han realizado muchas llamadas, pero solo está interesado en una secuencia particular de ellas, entonces una alternativa es usar el método **`assert_has_calls()`** . Esto toma una lista de llamadas (construidas con el objeto **`call`**). Si esa secuencia de llamadas sobre **`mock_calls`**, la aserción se realiza correctamente.

```
>>> m = MagicMock()
>>> m().foo().bar().baz()
<MagicMock name='mock().foo().bar().baz()' id='...'>
>>> m.one().two().three()
<MagicMock name='mock.one().two().three()' id='...'>
>>> calls = call.one().two().three().call_list()
>>> m.assert_has_calls(calls)
```

Aunque la llamada encadenada `m.one().two().three()` no son las únicas llamadas que se han realizado al mock, la aserción aún tiene éxito.

A veces, un mock puede tener varias llamadas, y solo le interesa afirmar sobre *algunas* de esas llamadas. Puede que ni siquiera importe el pedido. En este caso, puede pasar `any_order=True` a `assert_has_calls`:

```
>>> m = MagicMock()
>>> m(1), m.two(2, 3), m.seven(7), m.fifty('50')
(...)
>>> calls = [call.fifty('50'), call(1), call.seven(7)]
>>> m.assert_has_calls(calls, any_order=True)
```

Coincidencia de argumentos más compleja

Usando el mismo concepto básico que **`ANY`** podemos implementar comparadores para hacer afirmaciones más complejas en objetos usados como argumentos para mocks.

Supongamos que esperamos que algún objeto se pase a un mock que, por defecto, se compara igual en función de la identidad del objeto (que es el valor predeterminado de Python para las clases definidas por el usuario). Para usar **`assert_called_with()`** tendríamos que pasar exactamente el mismo objeto. Si solo estamos interesados en algunos de los atributos de este objeto, podemos crear un comparador que verifique estos atributos por nosotros.

Puede ver en este ejemplo cómo una llamada “estándar” a `assert_called_with` no es suficiente:

```
>>> class Foo:
...     def __init__(self, a, b):
...         self.a, self.b = a, b
...
>>> mock = Mock(return_value=None)
>>> mock(Foo(1, 2))
>>> mock.assert_called_with(Foo(1, 2))
Traceback (most recent call last):
    ...
AssertionError: expected call not found.
```

```
Expected: mock(<__main__.Foo object at 0x...>)
Actual: mock(<__main__.Foo object at 0x...>)
```

Una función de comparación para nuestra clase `Foo` podría verse así:

```
>>> def compare(self, other):
...     if not type(self) == type(other):
...         return False
...     if self.a != other.a:
...         return False
...     if self.b != other.b:
...         return False
...     return True
...
```

Y un objeto comparador que puede usar funciones de comparación como esta para su operación de igualdad se vería así:

```
>>> class Matcher:
...     def __init__(self, compare, some_obj):
...         self.compare = compare
...         self.some_obj = some_obj
...     def __eq__(self, other):
...         return self.compare(self.some_obj, other)
...
```

Poniendo todo esto junto:

```
>>> match_foo = Matcher(compare, Foo(1, 2))
>>> mock.assert_called_with(match_foo)
```

El `Matcher` es instanciado con nuestra función de comparación y el objeto `Foo` con el que queremos comparar. En `assert_called_with` se llamará al método de igualdad `Matcher`, que compara el objeto con el que se llamó al mock con el que creamos nuestro matcher. Si coinciden, entonces pasa `assert_called_with`, y si no lo hacen, se lanzará un **`AssertionError`**:

```
>>> match_wrong = Matcher(compare, Foo(3, 4))
>>> mock.assert_called_with(match_wrong)
Traceback (most recent call last):
    ...
AssertionError: Expected: ((<Matcher object at 0x...>,),
Called with: ((<Foo object at 0x...>,), {})
```

Con un poco de ajuste, podría hacer que la función de comparación genere el `AssertionError` directamente y proporcione un mensaje de falla más útil.

A partir de la versión 1.5, la biblioteca de pruebas de Python PyHamcrest [https://pyhamcrest.readthedocs.io/] proporciona una funcionalidad similar, que puede ser útil aquí, en la forma de su comparador de igualdad (hamcrest.library.integration.match_equality [https://pyhamcrest.readthedocs.io/en/release-1.8/integration/#module-hamcrest.library.integration.match_equality]).

`test` — Regression tests package for Python

Nota

The `test` package is meant for internal use by Python only. It is documented for the benefit of the core developers of Python. Any use of this package outside of Python's standard library is discouraged as code mentioned here can change or be removed without notice between releases of Python.

The `test` package contains all regression tests for Python as well as the modules `test.support` and `test.regrtest`. `test.support` is used to enhance your tests while `test.regrtest` drives the testing suite.

Each module in the `test` package whose name starts with `test_` is a testing suite for a specific module or feature. All new tests should be written using the `unittest` or `doctest` module. Some older tests are written using a «traditional» testing style that compares output printed to `sys.stdout`; this style of test is considered deprecated.

Ver también

Módulo `unittest`
Escritura de pruebas de regresión de PyUnit.

Módulo `doctest`
Pruebas integradas en cadenas de caracteres de documentación.

Writing Unit Tests for the `test` package

Se prefiere que las pruebas que utilizan el módulo `unittest` sigan algunas pautas. Una es nombrar el módulo de prueba comenzándolo con `test_` y terminarlo con el nombre del módulo que se está probando. Los métodos de prueba en el módulo de prueba deben comenzar con `test_` y terminar con una descripción de lo que el método está probando. Esto es necesario para que el controlador de prueba reconozca los métodos como métodos de prueba. Por lo tanto, no se debe incluir una cadena de caracteres de documentación para el método. Se debe usar un comentario (como `# Tests function returns only True or False`) para proporcionar documentación para los métodos de prueba. Esto se hace porque las cadenas de documentación se imprimen si existen y, por lo tanto, no se indica qué prueba se está ejecutando.

A menudo se usa una plantilla básica:

```
import unittest
from test import support

class MyTestCase1(unittest.TestCase):

    # Only use setUp() and tearDown() if necessary

    def setUp(self):
        ... code to execute in preparation for tests ...

    def tearDown(self):
        ... code to execute to clean up after tests ...

    def test_feature_one(self):
        # Test feature one.
        ... testing code ...

    def test_feature_two(self):
        # Test feature two.
        ... testing code ...
    ... more test methods ...

class MyTestCase2(unittest.TestCase):
    ... same structure as MyTestCase1 ...

... more test classes ...

if __name__ == '__main__':
    unittest.main()
```

Este patrón de código permite que el conjunto de pruebas sea ejecutado por **`test.regrtest`**, como un script que admite la CLI **`unittest`**, o mediante la CLI `python -m unittest`.

El objetivo de las pruebas de regresión es intentar romper el código. Esto lleva a algunas pautas a seguir:

- El conjunto de pruebas debe ejercer todas las clases, funciones y constantes. Esto incluye no solo la API externa que se presentará al mundo exterior sino también el código «privado».
- Se prefiere la prueba de caja blanca (examinar el código que se prueba cuando se escriben las pruebas). Las pruebas de caja negra (probar solo la interfaz de usuario publicada) no son lo suficientemente completas como para garantizar que se prueben todos los casos límite y límite.
- Asegúrese de que todos los valores posibles son probados, incluidos los no válidos. Esto asegura que no solo todos los valores válidos sean aceptables, sino que los valores incorrectos se manejen correctamente.
- Agote tantas rutas de código cómo sea posible. Pruebe donde se produce la ramificación y, por lo tanto, adapte la entrada para asegurarse de que se toman muchas rutas diferentes a través del código.
- Añada una prueba explícita para cualquier error descubierto para el código probado. Esto asegurará que el error no vuelva a aparecer si el código se cambia en el futuro.

- Asegúrese de limpiar después de sus pruebas (así como cerrar y eliminar todos los archivos temporales).

- Si una prueba depende de una condición específica del sistema operativo, verifique que la condición ya existe antes de intentar la prueba.

- Importe la menor cantidad de módulos posible y hágalo lo antes posible. Esto minimiza las dependencias externas de las pruebas y también minimiza el posible comportamiento anómalo de los efectos secundarios de importar un módulo.

- Intente maximizar la reutilización del código. En ocasiones, las pruebas variarán en algo tan pequeño como qué tipo de entrada se utiliza. Minimice la duplicación de código usando como clase base una clase de prueba básica con una clase que especifique la entrada:

```
class TestFuncAcceptsSequencesMixin:

    func = mySuperWhammyFunction

    def test_func(self):
        self.func(self.arg)

class AcceptLists(TestFuncAcceptsSequencesMixin, un
    arg = [1, 2, 3]

class AcceptStrings(TestFuncAcceptsSequencesMixin,
    arg = 'abc'

class AcceptTuples(TestFuncAcceptsSequencesMixin, u
    arg = (1, 2, 3)
```

When using this pattern, remember that all classes that inherit from `unittest.TestCase` are run as tests. The **`TestFuncAcceptsSequencesMixin`** class in the example above does not have any data and so can't be run by itself, thus it does not inherit from **`unittest.TestCase`**.

Ver también

Desarrollo dirigido por pruebas (*Test Driven Development*)
Un libro de *Kent Beck* sobre pruebas de escritura antes del código.

Ejecución de pruebas utilizando la interfaz de línea de comandos

The `test` package can be run as a script to drive Python's regression test suite, thanks to the `-m` option: **python -m test**. Under the hood, it uses `test.regrtest`; the call **python -m test.regrtest** used in previous Python versions still works. Running the script by itself automatically starts running all regression tests in the `test` package. It does this by finding all modules in the package whose name starts with `test_`, importing them, and executing the function `test_main()` if present or loading the tests via unittest.TestLoader.loadTestsFromModule if `test_main` does not exist. The names of tests to execute may also be passed to the script. Specifying a single regression test (**python -m test test_spam**) will minimize output and only print whether the test passed or failed.

Running `test` directly allows what resources are available for tests to use to be set. You do this by using the `-u` command-line option. Specifying `all` as the value for the `-u` option enables all possible resources: **python -m test -uall**. If all but one resource is desired (a more common case), a comma-separated list of resources that are not desired may be listed after `all`. The command **python -m test -uall,-audio,-largefile** will run `test` with all resources except the `audio` and `largefile` resources. For a list of all resources and more command-line options, run **python -m test -h**.

Algunas otras formas de ejecutar las pruebas de regresión dependen de en qué plataforma se ejecuten las pruebas. En Unix, puede ejecutar **make test** en el directorio de nivel superior donde se construyó Python. En Windows, ejecutar **rt.bat** desde su directorio `PCbuild` ejecutará todas las pruebas de regresión.

Added in version 3.14: Output is colorized by default and can be controlled using environment variables.

`test.support` — Utilities for the Python test suite

The `test.support` module provides support for Python's regression test suite.

Nota

`test.support` is not a public module. It is documented here to help Python developers write tests. The API of this module is subject to change without backwards compatibility concerns between releases.

Este módulo define las siguientes excepciones:

exception test.support.TestFailed

Excepción que se lanzará cuando una prueba falle. Está en desuso a favor de pruebas basadas en `unittest`y los métodos de aserción de `unittest.TestCase`.

exception test.support.ResourceDenied

La subclase de `unittest.SkipTest`. Se lanza cuando un recurso (como una conexión de red) no está disponible. Lanzado por la función `requires()`.

The `test.support` module defines the following constants:

test.support.verbose

`True` cuando la salida detallada está habilitada. Debe verificarse cuando se desea información más detallada sobre una prueba en ejecución. *verbose* está establecido por `test.regrtest`.

test.support.is_jython

`True` si el intérprete en ejecución es Jython.

test.support.is_android

`True` if `sys.platform` is `android`.

test.support.is_emscripten

`True` if `sys.platform` is `emscripten`.

test.support.is_wasi

`True` if `sys.platform` is `wasi`.

test.support.is_apple_mobile

`True` if `sys.platform` is `ios`, `tvos`, or `watchos`.

test.support.is_apple

`True` if `sys.platform` is `darwin` or `is_apple_mobile` is `True`.

test.support.unix_shell

Ruta del shell si no está en Windows; de otra manera `None`.

test.support.LOOPBACK_TIMEOUT

Tiempo de espera en segundos para las pruebas que utilizan un servidor de red que escucha en la interfaz local de loopback de la red como `127.0.0.1`.

El tiempo de espera es lo suficientemente largo para evitar el fracaso de la prueba: tiene en cuenta que el cliente y el servidor pueden ejecutarse en diferentes hilos o incluso en diferentes procesos.

El tiempo de espera debe ser lo suficientemente largo para los métodos **`connect()`**, **`recv()`** y **`send()`** de **`socket.socket`**.

Su valor por defecto es de 5 segundos.

Ver también `INTERNET_TIMEOUT`.

test.support.INTERNET_TIMEOUT

Tiempo de espera en segundos para las solicitudes de red que van a Internet.

El tiempo de espera es lo suficientemente corto como para evitar que una prueba espere demasiado tiempo si la solicitud de Internet se bloquea por cualquier motivo.

Normalmente, un tiempo de espera utilizando `INTERNET_TIMEOUT` no debería marcar una prueba como fallida, sino que debería omitir la prueba: ver `transient_internet()`.

Su valor por defecto es de 1 minuto.

Ver también `LOOPBACK_TIMEOUT`.

test.support.SHORT_TIMEOUT

Tiempo de espera en segundos para marcar una prueba como fallida si la prueba tarda «demasiado».

El valor del tiempo de espera depende de la opción de línea de comandos regrtest `--timeout`.

Si una prueba que utiliza `SHORT_TIMEOUT` empieza a fallar aleatoriamente en buildbots lentos, utilice en su lugar `LONG_TIMEOUT`.

Su valor por defecto es de 30 segundos.

test.support.LONG_TIMEOUT

Tiempo de espera en segundos para detectar cuando se cuelga una prueba.

Es lo suficientemente largo como para reducir el riesgo de fracaso de la prueba en los buildbots de Python más lentos. No debe utilizarse para marcar una prueba como fallida si la prueba tarda «demasiado». El valor del tiempo de espera depende de la opción de línea de comandos regrtest `--timeout`.

Su valor por defecto es de 5 minutos.

Ver también **`LOOPBACK_TIMEOUT`**, **`INTERNET_TIMEOUT`** y **`SHORT_TIMEOUT`**.

test.support.PGO

Establecido cuando se pueden omitir las pruebas cuando no son útiles para *PGO*.

test.support.PIPE_MAX_SIZE

Una constante que probablemente sea más grande que el tamaño del búfer de la tubería (*pipe*) del sistema operativo subyacente, para bloquear las escrituras.

test.support.Py_DEBUG

`True` if Python was built with the **`Py_DEBUG`** macro defined, that is, if Python was built in debug mode.

Added in version 3.12.

test.support.SOCK_MAX_SIZE

Una constante que probablemente sea más grande que el tamaño del búfer del socket del sistema operativo subyacente para bloquear las escrituras.

test.support.TEST_SUPPORT_DIR

Set to the top level directory that contains **`test.support`**.

test.support.TEST_HOME_DIR

Establecido en el directorio de nivel superior para el paquete de prueba.

test.support.TEST_DATA_DIR

Establecido en el directorio `data` dentro del paquete de prueba.

test.support.MAX_Py_ssize_t

Establecido `sys.maxsize` para pruebas de gran memoria.

test.support.max_memuse

Establecido por `set_memlimit()` como límite de memoria para pruebas de memoria grande. Limitado por `MAX_Py_ssize_t`.

test.support.real_max_memuse

Establecido por `set_memlimit()` como límite de memoria para pruebas de memoria grande. No limitado por `MAX_Py_ssize_t`.

test.support.MISSING_C_DOCSTRINGS

Vale `True` si Python se construye sin cadenas de documentos (la macro `WITH_DOC_STRINGS` no está definida). Consulte la opción `configure --without-doc-strings`.

Consulte también la variable `HAVE_DOCSTRINGS`.

test.support.HAVE_DOCSTRINGS

Vale `True` si las cadenas de documentación de la función están disponibles. Consulte la opción `python -OO`, que elimina las cadenas de documentación de las funciones implementadas en Python.

Consulte también la variable `MISSING_C_DOCSTRINGS`.

test.support.TEST_HTTP_URL

Define la URL de un servidor HTTP dedicado para las pruebas de red.

test.support.ALWAYS_EQ

Objeto que es igual a cualquier cosa. Se utiliza para probar la comparación de tipos mixtos.

test.support.NEVER_EQ

Objeto que no es igual a nada (incluso a `ALWAYS_EQ`). Se utiliza para probar la comparación de tipos mixtos.

test.support.LARGEST

Objeto que es mayor que cualquier cosa (excepto a sí mismo). Se utiliza para probar la comparación de tipos mixtos.

test.support.SMALLEST

Objeto que es menor que cualquier cosa (excepto él mismo). Se utiliza para probar la comparación de tipos mixtos.

The **`test.support`** module defines the following functions:

test.support.busy_retry(*timeout*, *err_msg=None*, */*, ***, *error=True*)

Run the loop body until `break` stops the loop.

After *timeout* seconds, raise an **`AssertionError`** if *error* is true, or just stop the loop if *error* is false.

Example:

```
for _ in support.busy_retry(support.SHORT_TIMEOUT):
    if check():
        break
```

Example of error=False usage:

```
for _ in support.busy_retry(support.SHORT_TIMEOUT,
    if check():
        break
else:
    raise RuntimeError('my custom error')
```

test.support.sleeping_retry(*timeout*, *err_msg=None*, */*, ***, *init_delay=0.010*, *max_delay=1.0*, *error=True*)

Wait strategy that applies exponential backoff.

Run the loop body until `break` stops the loop. Sleep at each loop iteration, but not at the first iteration. The sleep delay is doubled at each iteration (up to *max_delay* seconds).

See `busy_retry()` documentation for the parameters usage.

Example raising an exception after SHORT_TIMEOUT seconds:

```
for _ in support.sleeping_retry(support.SHORT_TIMEO
    if check():
        break
```

Example of error=False usage:

```
for _ in support.sleeping_retry(support.SHORT_TIMEO
    if check():
        break
else:
    raise RuntimeError('my custom error')
```

test.support.is_resource_enabled(*resource*)

Retorna `True` si *resource* está habilitado y disponible. La lista de recursos disponibles solo se establece cuando **`test.regrtest`** está ejecutando las pruebas.

test.support.get_resource_value(*resource*)

Return the value specified for *resource* (as `-u` `resource=value`). Return `None` if *resource* is disabled or no value is specified.

test.support.python_is_optimized()

Retorna `True` si Python no fue construido con `-O0` o `-Og`.

test.support.with_pymalloc()

Return **`_testcapi.WITH_PYMALLOC`**.

test.support.requires(*resource*, *msg=None*)

Lanza **`ResourceDenied`** si *resource* no está disponible. *msg* es el argumento para **`ResourceDenied`** si se lanza. Siempre retorna `True` si es invocado por una función cuyo `__name__` es `'__main__'`. Se usa cuando se ejecutan pruebas por **`test.regrtest`**.

test.support.sortdict(*dict*)

Retorna una representación del *diccionario* con las claves ordenadas.

test.support.findfile(*filename*, *subdir=None*)

Retorna la ruta al archivo llamado *filename*. Si no se encuentra ninguna coincidencia, se retorna *filename*. Esto no equivale a un fallo, ya que podría ser la ruta al archivo.

La configuración *subdir* indica una ruta relativa a utilizar para encontrar el archivo en lugar de buscar directamente en los directorios de ruta.

test.support.get_pagesize()

Get size of a page in bytes.

Added in version 3.12.

test.support.setswitchinterval(*interval*)

Establecido **`sys.setswitchinterval()`** en el *intervalo* dado. Define un intervalo mínimo para los sistemas Android para evitar que el sistema se cuelgue.

test.support.check_impl_detail(***guards*)

Use esta verificación para proteger las pruebas específicas de implementación de CPython o para ejecutarlas solo en las implementaciones protegidas por los argumentos. Esta función retorna `True` o `False` según la plataforma del host. Ejemplo de uso:

```
check_impl_detail()               # Only on CPython
check_impl_detail(jython=True)    # Only on Jython.
check_impl_detail(cpython=False)  # Everywhere exce
```

test.support.set_memlimit(*limit*)

Establece los valores para `max_memuse` y `real_max_memuse` para pruebas de memoria grande.

test.support.record_original_stdout(*stdout*)

Almacene el valor de *stdout*. Está destinado a mantener el *stdout* en el momento en que comenzó el regrtest.

test.support.get_original_stdout()

Retorna el *stdout* original establecido por `record_original_stdout()` o `sys.stdout` si no está configurado.

test.support.args_from_interpreter_flags()

Retorna una lista de argumentos de línea de comandos que reproducen la configuración actual en `sys.flags` y `sys.warnoptions`.

test.support.optim_args_from_interpreter_flags()

Retorna una lista de argumentos de línea de comandos que reproducen la configuración de optimización actual en `sys.flags`.

test.support.captured_stdin()
test.support.captured_stdout()
test.support.captured_stderr()

Un administrador de contexto que reemplaza temporalmente la secuencia nombrada con un objeto `io.StringIO`.

Ejemplo de uso con flujos de salida:

```
with captured_stdout() as stdout, captured_stderr()
    print("hello")
    print("error", file=sys.stderr)
assert stdout.getvalue() == "hello\n"
assert stderr.getvalue() == "error\n"
```

Ejemplo de uso con flujo de entrada:

```
with captured_stdin() as stdin:
    stdin.write('hello\n')
    stdin.seek(0)
    # call test code that consumes from sys.stdin
    captured = input()
self.assertEqual(captured, "hello")
```

test.support.disable_faulthandler()

Un administrador de contexto que deshabilita temporalmente **`faulthandler`**.

test.support.gc_collect()

Se fuerza la mayor cantidad posible de objetos para ser recolectados. Esto es necesario porque el recolector de basura no garantiza la desasignación oportuna. Esto significa que los métodos `__del__` pueden llamarse más tarde de lo esperado y las referencias débiles pueden permanecer vivas por más tiempo de lo esperado.

test.support.disable_gc()

Un administrador de contexto que deshabilita el recolector de basura al entrar. Al salir, el recolector de basura se restaura a su estado anterior.

test.support.swap_attr(*obj*, *attr*, *new_val*)

Administrador de contexto para intercambiar un atributo con un nuevo objeto.

Uso:

```
with swap_attr(obj, "attr", 5):
    ...
```

Esto establecerá `obj.attr` en 5 durante la duración del bloque `with`, restaurando el valor anterior al final del bloque. Si `attr` no existe en `obj`, se creará y luego se eliminará al final del bloque.

El valor anterior (o `None` si no existe) se asignará al objetivo de la cláusula «como», si existe.

test.support.swap_item(*obj*, *attr*, *new_val*)

Administrador de contexto para intercambiar un elemento con un nuevo objeto.

Uso:

```
with swap_item(obj, "item", 5):
    ...
```

Esto establecerá `obj ["item"]` a 5 durante la duración del bloque `with`, restaurando el valor anterior al final del bloque. Si `item` no existe en `obj`, se creará y luego se eliminará al final del bloque.

El valor anterior (o `None` si no existe) se asignará al objetivo de la cláusula «como», si existe.

test.support.flush_std_streams()

Llama al método `flush()` en **`sys.stdout`** y luego en **`sys.stderr`**. Se puede usar para asegurarse de que el orden de los registros sea consistente antes de escribir en stderr.

Added in version 3.11.

test.support.print_warning(*msg*)

Imprime una advertencia en **`sys.__stderr__`**. Formatea el mensaje como: `f "Warning -- {msg}"`. Si *msg* se compone de varias líneas, añade el prefijo `"Warning --"` a cada línea.

Added in version 3.9.

test.support.wait_process(*pid*, ***, *exitcode*, *timeout=None*)

Espere hasta que el proceso *pid* termine y comprobará que el código de salida del proceso es *exitcode*.

Lanza un `AssertionError` si el código de salida del proceso no es igual a *exitcode*.

Si el proceso se ejecuta durante más tiempo que *timeout* en segundos (`SHORT_TIMEOUT` por defecto), mata el proceso y lanza un `AssertionError`. La función de tiempo de espera no está disponible en Windows.

Added in version 3.9.

test.support.calcobjsize(*fmt*)

Retorna el tamaño del `PyObject` cuyos miembros de estructura están definidos por *fmt*. El valor retornado incluye el tamaño del encabezado y la alineación del objeto de Python.

test.support.calcvobjsize(*fmt*)

Retorna el tamaño del `PyVarObject` cuyos miembros de estructura están definidos por *fmt*. El valor retornado incluye el tamaño del encabezado y la alineación del objeto de Python.

test.support.checksizeof(*test*, *o*, *size*)

Para el caso de prueba (*testcase*), se aserciona que el `sys.getsizeof` para *o* más el tamaño del encabezado *GC* es igual a *size*.

@test.support.anticipate_failure(*condition*)

Un decorador para marcar condicionalmente las pruebas con `unittest.expectedFailure()`. Cualquier uso de este decorador debe tener un comentario asociado que identifique el problema relevante del rastreador.

test.support.system_must_validate_cert(*f*)

Un decorador que se salta la prueba decorada en los errores de validación de la certificación TLS.

@test.support.run_with_locale(*catstr*, **locales*)

Un decorador para ejecutar una función en una configuración regional diferente, restableciéndola correctamente una vez que ha finalizado. *catstr* es la categoría de configuración regional como una cadena (por ejemplo, `"LC_ALL"`). Las *locales* aprobadas se probarán secuencialmente y se utilizará la primera configuración regional válida.

@test.support.run_with_tz(*tz*)

Un decorador para ejecutar una función en una zona horaria específica, restableciéndola correctamente una vez que haya finalizado.

@test.support.requires_freebsd_version(**min_version*)

Decorador para la versión mínima cuando se ejecuta la prueba en FreeBSD. Si la versión de FreeBSD es inferior a la mínima, se salta la prueba.

@test.support.requires_linux_version(**min_version*)

Decorador para la versión mínima cuando se ejecuta la prueba en Linux. Si la versión de Linux es inferior a la mínima, se omite la prueba.

@test.support.requires_mac_version(**min_version*)

Decorador para la versión mínima cuando se ejecuta la prueba en macOS. Si la versión de macOS es inferior a la mínima, se omite la prueba.

@test.support.requires_gil_enabled

Decorator for skipping tests on the free-threaded build. If the GIL is disabled, the test is skipped.

@test.support.requires_IEEE_754

Decorador para omitir pruebas en plataformas que no son *IEEE 754*.

@test.support.requires_zlib

Decorador para omitir pruebas si `zlib` no existe.

@test.support.requires_gzip

Decorador para omitir pruebas si `gzip` no existe.

@test.support.requires_bz2

Decorador para omitir pruebas si `bz2` no existe.

@test.support.requires_lzma

Decorador para omitir pruebas si `lzma` no existe.

@test.support.requires_resource(*resource*)

Decorador para omitir pruebas si *resource* no está disponible.

@test.support.requires_docstrings

Decorador para ejecutar solo la prueba si `HAVE_DOCSTRINGS`.

@test.support.requires_limited_api

Decorator for only running the test if Limited C API is available.

@test.support.cpython_only

Decorador para pruebas solo aplicable a CPython.

@test.support.impl_detail(*msg=None*, ***guards*)

Decorador para invocar `check_impl_detail()` en *guards*. Si eso retorna `False`, entonces usa *msg* como la razón para omitir la prueba.

@test.support.thread_unsafe(*reason=None*)

Decorator for marking tests as thread-unsafe. This test always runs in one thread even when invoked with `--parallel-threads`.

@test.support.no_tracing

Decorador para desactivar temporalmente el seguimiento durante la duración de la prueba.

@test.support.refcount_test

Decorador para pruebas que implican conteo de referencias. El decorador no ejecuta la prueba si CPython no la ejecuta. Cualquier función de rastreo no se establece durante la duración de la prueba para evitar conteos de referencia(*refcounts*) inesperados causados por la función de rastreo.

@test.support.bigmemtest(*size*, *memuse*, *dry_run* = *True*)

Decorador para pruebas *bigmem*.

size es un tamaño solicitado para la prueba (en unidades arbitrarias interpretadas por la prueba). *memuse* es el número de bytes por unidad para la prueba, o una buena estimación de la misma. Por ejemplo, una prueba que necesita dos búfers de byte, de 4 *GiB* cada uno, podría decorarse con `@bigmemtest(size=_4G, memuse=2)`.

El argumento *size* normalmente se pasa al método de prueba decorado como un argumento adicional. Si *dry_run* es `True`, el valor pasado al método de prueba puede ser menor que el valor solicitado. Si *dry_run* es `False`, significa que la prueba no admite ejecuciones ficticias cuando no se especifica `-M`.

@test.support.bigaddrspacetest

Decorador para pruebas que llenan el espacio de direcciones.

test.support.linked_to_musl()

Return `False` if there is no evidence the interpreter was compiled with `musl`, otherwise return a version triple, either `(0, 0, 0)` if the version is unknown, or the actual version if it is known. Intended for use in `skip` decorators. `emscripten` and `wasi` are assumed to be compiled with `musl`; otherwise `platform.libc_ver` is checked.

test.support.check_syntax_error(*testcase*, *statement*, *errtext*='', *, *lineno*=None, *offset*=None)

Prueba los errores de sintaxis en *statement* intentando compilar *statement*. *testcase* es la instancia **`unittest`** para la prueba. *errtext* es la expresión regular que debe coincidir con la representación de cadena de caracteres que es lanza en **`SyntaxError`**. Si *lineno* no es `None`, se compara con la línea de la excepción. Si *offset* no es `None`, se compara con el desplazamiento de la excepción.

test.support.open_urlresource(*url*, **args*, ***kw*)

Abre *url*. Si la apertura falla, se lanza **`TestFailed`**.

test.support.reap_children()

Se utiliza esto al final de `test_main` siempre que se inicien subprocesos. Esto ayudará a garantizar que ningún proceso hijo adicional (zombies) se quede para acumular recursos y crear problemas al buscar refleaks.

test.support.get_attribute(*obj*, *name*)

Obtiene un atributo, lanzando **`unittest.SkipTest`** si **`AttributeError`** está activado.

test.support.catch_unraisable_exception()

El administrador de contexto detectando excepciones imposibles de evaluar usando **`sys.unraisablehook()`**.

El almacenamiento del valor de excepción (`cm.unraisable.exc_value`) crea un ciclo de referencia. El ciclo de referencia se interrumpe explícitamente cuando sale el administrador de contexto.

El almacenamiento del objeto (`cm.unraisable.object`) puede resucitarlo si se establece en un objeto que se está finalizando. Salir del administrador de contexto borra el objeto almacenado.

Uso:

```
with support.catch_unraisable_exception() as cm:
    # code creating an "unraisable exception"
    ...

    # check the unraisable exception: use cm.unrais

# cm.unraisable attribute no longer exists at this 
# (to break a reference cycle)
```

Added in version 3.8.

test.support.load_package_tests(*pkg_dir*, *loader*, *standard_tests*, *pattern*)

La implementación genérica del protocolo **`unittest`** `load_tests` para usar en paquetes de prueba. *pkg_dir* es el directorio raíz del paquete; *loader*, *standard_tests* y *pattern* son los argumentos esperados por `load_tests`. En casos simples, el paquete de prueba `__init __. Py` puede ser el siguiente:

```
import os
from test.support import load_package_tests

def load_tests(*args):
    return load_package_tests(os.path.dirname(__fil
```

test.support.detect_api_mismatch(*ref_api*, *other_api*, ***, *ignore=()*)

Retorna el conjunto de atributos, funciones o métodos de *ref_api* que no se encuentra en *other_api *, excepto por una lista definida de elementos que se ignorarán en esta comprobación especificada en *ignore.*

De forma predeterminada, omite los atributos privados que comienzan con "_" pero incluye todos los métodos mágicos, es decir, los que comienzan y terminan en "_".

Added in version 3.5.

test.support.patch(*test_instance*, *object_to_patch*, *attr_name*, *new_value*)

Se anula *object_to_patch.attr_name* con *new_value*. Se agrega también el procedimiento de limpieza a *test_instance* para restaurar *object_to_patch* para *attr_name*. *Attr_name* debe ser un atributo válido para *object_to_patch*.

test.support.run_in_subinterp(*code*)

Ejecuta *code* en el subinterpretador. Lanza `unittest.SkipTest` si `tracemalloc` está habilitado.

test.support.check_free_after_iterating(*test*, *iter*, *cls*, *args=()*)

Las instancias de aserción de *cls* se desalojan después de la iteración.

test.support.missing_compiler_executable(*cmd_names=[]*)

Verifica la existencia de los ejecutables del compilador cuyos nombres figuran en *cmd_names* o todos los ejecutables del compilador cuando *cmd_names* está vacío y retorna el primer ejecutable faltante o `None` cuando no se encuentra ninguno.

test.support.check__all__(*test_case*, *module*, *name_of_module=None*, *extra=()*, *not_exported=()*)

Aserciona que la variable `__all__` de *module* contiene todos los nombres públicos.

Los nombres públicos del módulo (su API) se detectan automáticamente en función de si coinciden con la convención de nombres públicos y se definieron en *module*.

El argumento *name_of_module* puede especificar (como una cadena o tupla del mismo) qué módulo(s) se podría definir una API para ser detectada como una API pública. Un caso para esto es cuando *module* importa parte de su API pública desde otros módulos, posiblemente un *backend de C* (como `csv` y su `_csv`).

The *extra* argument can be a set of names that wouldn't otherwise be automatically detected as «public», like objects without a proper `__module__` attribute. If provided, it will be added to the automatically detected ones.

El argumento *not_exported* puede ser un conjunto de nombres que no deben tratarse como parte de la API pública aunque sus nombres indiquen lo contrario.

Ejemplo de uso:

```
import bar
import foo
import unittest
from test import support

class MiscTestCase(unittest.TestCase):
    def test__all__(self):
        support.check__all__(self, foo)

class OtherTestCase(unittest.TestCase):
    def test__all__(self):
        extra = {'BAR_CONST', 'FOO_CONST'}
        not_exported = {'baz'}  # Undocumented name
        # bar imports part of its API from _bar.
        support.check__all__(self, bar, ('bar', '_b
                             extra=extra, not_expor
```

Added in version 3.6.

test.support.skip_if_broken_multiprocessing_synchronize()

Omite las pruebas si el módulo no se encuentra al **`multiprocessing.synchronize`**, si no hay una implementación de semáforo disponible, o si crear un *lock* lanza un `OSError`.

Added in version 3.10.

test.support.check_disallow_instantiation(*test_case*, *tp*, **args*, ***kwds*)

Se aserciona que el tipo *tp* no pueda ser instanciado usando *args* y *kwds*.

Added in version 3.10.

test.support.adjust_int_max_str_digits(*max_digits*)

Esta función devuelve un administrador de contexto que cambiará la configuración global de **`sys.set_int_max_str_digits()`** durante la duración del contexto para permitir la ejecución de código de prueba que necesita un límite diferente en la cantidad de dígitos al convertir entre un número entero y una cadena.

Added in version 3.11.

The **`test.support`** module defines the following classes:

class test.support.SuppressCrashReport

Un administrador de contexto suele intentar evitar ventanas emergentes de diálogo de bloqueo en las pruebas que se espera que bloqueen un subproceso.

En Windows, deshabilita los cuadros de diálogo de Informe de errores de Windows usando SetErrorMode [https://msdn.microsoft.com/en-us/library/windows/desktop/ms680621.aspx].

On UNIX, **`resource.setrlimit()`** is used to set **`resource.RLIMIT_CORE`**'s soft limit to 0 to prevent coredump file creation.

On both platforms, the old value is restored by **`__exit__()`**.

class test.support.SaveSignals

Clase para guardar y restaurar manejadores (*handlers*) de señal registrados por el manejador de señal Python.

save(*self*)

Guarda los controladores de señales en un diccionario que asigna números de señales al controlador de señales actual.

restore(*self*)

Establece los números de señal del diccionario `save()` en el controlador guardado.

class test.support.Matcher

matches(*self*, *d*, ***kwargs*)

Intenta hacer coincidir una sola sentencia con los argumentos proporcionados.

match_value(*self*, *k*, *dv*, *v*)

Intenta hacer coincidir un único valor almacenado (*dv*) con un valor proporcionado (*v*).

`test.support.socket_helper` — Utilities for socket tests

The **`test.support.socket_helper`** module provides support for socket tests.

Added in version 3.9.

test.support.socket_helper.IPV6_ENABLED

Se establece como `True` si IPv6 está habilitado en este host, `False` en caso contrario.

test.support.socket_helper.find_unused_port(*family=socket.AF_INET*, *socktype=socket.SOCK_STREAM*)

Retorna un puerto no utilizado que debería ser adecuado para el enlace. Esto se logra creando un *socket* temporal con la misma familia y tipo que el parámetro `sock` (el valor predeterminado es `AF_INET`, `SOCK_STREAM`) y vinculándolo a la dirección de host especificada (por defecto es `0.0.0.0`) con el puerto establecido en 0, provocando un puerto efímero no utilizado del sistema operativo. El *socket* temporal se cierra y se elimina, y se retorna el puerto efímero.

Este método o `bind_port()` debe usarse para cualquier prueba en la que un *socket* del servidor deba estar vinculado a un puerto en particular durante la duración de la prueba. Cuál usar depende de si el código de llamada está creando un *socket* Python, o si un puerto no utilizado debe proporcionarse en un constructor o pasar a un programa externo (es decir, el argumento `-accept` al modo *s_server de openssl*). Siempre es preferible **`bind_port()`** sobre `find_unused_port()` donde sea posible. Se desaconseja el uso de un puerto codificado ya que puede hacer que varias instancias de la prueba sean imposibles de ejecutar simultáneamente, lo cual es un problema para los *buildbots*.

test.support.socket_helper.bind_port(*sock*, *host=HOST*)

Se enlaza el *socket* a un puerto libre y retorna el número de puerto. Se basa en puertos efímeros para garantizar que estemos utilizando un puerto independiente. Esto es importante ya que muchas pruebas pueden ejecutarse simultáneamente, especialmente en un entorno *buildbot*. Este método lanza una excepción si `sock.family` es **`AF_INET`** y `sock.type` es **`SOCK_STREAM`**, y el *socket* tiene **`SO_REUSEADDR`** o **`SO_REUSEPORT`** establecido en él. Las pruebas nunca deben configurar estas opciones de *socket* para los *sockets TCP/IP*. El único caso para configurar estas opciones es probar la multidifusión (*multicasting*) a través de múltiples *sockets UDP*.

Además, si la opción de *socket* **`SO_EXCLUSIVEADDRUSE`** está disponible (es decir, en Windows), se establecerá en el *socket*. Esto evitará que otras personas se vinculen a nuestro host/puerto mientras dure la prueba.

test.support.socket_helper.bind_unix_socket(*sock*, *addr*)

Enlaza un socket Unix, lanzando `unittest.SkipTest` si se lanza `PermissionError`.

@test.support.socket_helper.skip_unless_bind_unix_socket

Un decorador para ejecutar pruebas que requieren un enlace(`bind()`) funcional para sockets en sistemas Unix.

test.support.socket_helper.transient_internet(*resource_name*, *, *timeout=30.0*, *errnos=()*)

Un gestor de contexto que lanza **`ResourceDenied`** cuando varios problemas con la conexión a Internet se manifiestan como excepciones.

`test.support.script_helper` — Utilities for the Python execution tests

The **`test.support.script_helper`** module provides support for Python's script execution tests.

test.support.script_helper.interpreter_requires_environment()

Retorna `True` si el `sys.executable interpreter` requiere variables de entorno para poder ejecutarse.

Esto está diseñado para usarse con `@unittest.skipIf()` para anotar pruebas que necesitan usar una función `assert_python*()` para iniciar un modo aislado (`-I`) o sin entorno proceso de subinterpretador de modo (`-E`).

Una compilación y prueba normal no se encuentra en esta situación, pero puede suceder cuando se intenta ejecutar el conjunto de pruebas de biblioteca estándar desde un intérprete que no tiene un directorio principal obvio con la lógica de búsqueda de directorio principal actual de Python.

La configuración **`PYTHONHOME`** es una forma de hacer que la mayoría del *testuite* se ejecute en esa situación. **`PYTHONPATH`** o **`PYTHONUSERSITE`** son otras variables de entorno comunes que pueden afectar si el intérprete puede o no comenzar.

test.support.script_helper.run_python_until_end(**args*, ***env_vars*)

Configura el entorno basado en *env_vars* para ejecutar el intérprete en un subproceso. Los valores pueden incluir `__isolated`, `__cleanenv`, `__cwd` y `TERM`.

Distinto en la versión 3.9: La función ya no elimina los espacios en blanco de *stderr*.

test.support.script_helper.assert_python_ok(**args*, ***env_vars*)

Aserción de que ejecutar el intérprete con *arg* y variables de entorno opcionales *env_vars* tiene éxito (`rc == 0`) y retorna una tupla `(código de retorno, stdout, stderr)`.

Si se establece el parámetro de solo palabra clave *_cleanenv*, *env_vars* se usa como un entorno nuevo.

Python se inicia en modo aislado (opción de línea de comando `-I`), excepto si el parámetro de solo palabra clave *_isolated* se establece en `False`.

Distinto en la versión 3.9: La función ya no elimina los espacios en blanco de *stderr*.

test.support.script_helper.assert_python_failure(**args*, ***env_vars*)

Aserciona que la ejecución del intérprete con *args* y variables de entorno opcionales *env_vars* falla (`rc! = 0`) y retorna una tupla `(return code, stdout, stderr)`.

Consulte **`assert_python_ok()`** para más opciones.

Distinto en la versión 3.9: La función ya no elimina los espacios en blanco de *stderr*.

test.support.script_helper.spawn_python(**args*, *stdout=subprocess.PIPE*, *stderr=subprocess.STDOUT*, ***kw*)

Ejecuta un subproceso de Python con los argumentos dados.

kw es un argumento adicional de palabras clave para pasar a **`subprocess.Popen()`**. Retorna un objeto a **`subprocess.Popen`**.

test.support.script_helper.kill_python(*p*)

Ejecuta el proceso dado **`subprocess.Popen`** hasta que finalice y retorne *stdout*.

test.support.script_helper.make_script(*script_dir*, *script_basename*, *source*, *omit_suffix=False*)

Crea un script que contiene *source* en la ruta *script_dir* y *script_basename*. Si *omit_suffix* es `False`, agregue `.py` al nombre. Retorna la ruta completa del script.

test.support.script_helper.make_zip_script(*zip_dir*, *zip_basename*, *script_name*, *name_in_zip=None*)

Crea un archivo zip en *zip_dir* y *zip_basename* con la extensión `zip` que contiene los archivos en *script_name*. *name_in_zip* es el nombre del archivo. Retorna una tupla que contiene `(ruta completa, ruta completa del nombre del archivo)`.

test.support.script_helper.make_pkg(*pkg_dir*, *init_source=''*)

Crea un directorio llamado *pkg_dir* que contiene un archivo `__init__` con *init_source* como su contenido.

test.support.script_helper.make_zip_pkg(*zip_dir*, *zip_basename*, *pkg_name*, *script_basename*, *source*, *depth=1*, *compiled=False*)

Crea un directorio de paquete zip con una ruta de *zip_dir* y *zip_basename* que contiene un archivo `__init__` vacío y un archivo *script_basename* que contiene el *source*. Si *compilado* es `True`, ambos archivos fuente se compilarán y se agregarán al paquete zip. Retorna una tupla de la ruta zip completa y el nombre de archivo para el archivo zip.

`test.support.bytecode_helper` — Support tools for testing correct bytecode generation

The **`test.support.bytecode_helper`** module provides support for testing and inspecting bytecode generation.

Added in version 3.9.

El módulo define la siguiente clase:

class test.support.bytecode_helper.BytecodeTestCase(*unittest.TestCase*)

Esta clase tiene métodos de aserción personalizados para inspeccionar el código de bytes.

BytecodeTestCase.get_disassembly_as_string(*co*)

Retorna el desensamblaje de *co* como cadena.

BytecodeTestCase.assertInBytecode(*x*, *opname*, *argval=_UNSPECIFIED*)

Retorna instr si se encuentra *opname*, de lo contrario lanza **`AssertionError`**.

BytecodeTestCase.assertNotInBytecode(*x*, *opname*, *argval=_UNSPECIFIED*)

Lanza **`AssertionError`** si se encuentra *opname*.

`test.support.threading_helpe`

— Utilities for threading tests

The **`test.support.threading_helper`** module provides support for threading tests.

Added in version 3.10.

test.support.threading_helper.join_thread(*thread*, *timeout*=*None*)

Se une un *thread* dentro de *timeout*. Lanza un **`AssertionError`** si el hilo sigue vivo después de unos segundos de *timeout*.

@test.support.threading_helper.reap_threads

Decorador para garantizar que los hilos se limpien incluso si la prueba falla.

test.support.threading_helper.start_threads(*threads*, *unlock*=*None*)

Administrador de contexto para iniciar *threads*, que es una secuencia de subprocesos. *unlock* es una función a la que se llama después de que se inician los subprocesos, incluso si se generó una excepción; un ejemplo sería **`threading.Event.set()`**. `start_threads` intentará unirse a los subprocesos iniciados al salir.

test.support.threading_helper.threading_cleanup(**original_values*)

Se limpia los hilos no especificados en *original_values*. Diseñado para emitir una advertencia si una prueba deja hilos en ejecución en segundo plano.

test.support.threading_helper.threading_setup()

Retorna el recuento del hilo actual y copia de subprocesos colgantes.

test.support.threading_helper.wait_threads_exit(*timeout*=*None*)

El administrador de contexto debe esperar hasta que salgan todos los hilos creados en la declaración `with`.

test.support.threading_helper.catch_threading_exception()

El administrador de contexto captura **`threading.Thread`** excepción usando **`threading.excepthook()`**.

Atributos establecidos cuando se captura una excepción:

- `exc_type`
- `exc_value`
- `exc_traceback`
- `thread`

Consulte la documentación para **`threading.excepthook()`**.

Estos atributos se eliminan en la salida del administrador de contexto.

Uso:

```
with threading_helper.catch_threading_exception() a
    # code spawning a thread which raises an except
    ...

    # check the thread exception, use cm attributes
    # exc_type, exc_value, exc_traceback, thread
    ...

# exc_type, exc_value, exc_traceback, thread attrib
# exists at this point
# (to avoid reference cycles)
```

Added in version 3.8.

test.support.threading_helper.run_concurrently(*worker_func*, *nthreads*, *args=()*, *kwargs={}*)

Run the worker function concurrently in multiple threads. Reraises an exception if any thread raises one, after all threads have finished.

`test.support.os_helper` — Utilities for os tests

The `test.support.os_helper` module provides support for os tests.

Added in version 3.10.

test.support.os_helper.FS_NONASCII

Un carácter no codificable ASCII codificable por `os.fsencode()`.

test.support.os_helper.SAVEDCWD

Establecido `os.getcwd()`.

test.support.os_helper.TESTFN

Establecido a un nombre que sea seguro de usar como nombre de un archivo temporal. Cualquier archivo temporal que se cree debe cerrarse y desvincularse (eliminarse).

test.support.os_helper.TESTFN_NONASCII

Establecido en un nombre de archivo que contiene el carácter `FS_NONASCII`, si existe. Esto garantiza que, si existe el nombre de archivo, se puede codificar y decodificar con la codificación predeterminada del sistema de archivos. Esto permite omitir fácilmente las pruebas que requieren un nombre de archivo que no sea ASCII en plataformas donde no pueden funcionar.

test.support.os_helper.TESTFN_UNENCODABLE

Establecido un nombre de archivo (tipo *str*) que no se pueda codificar mediante la codificación del sistema de archivos en modo estricto. Puede ser `None` si no es posible generar dicho nombre de archivo.

test.support.os_helper.TESTFN_UNDECODABLE

Establecido un nombre de archivo (tipo de bytes) que no pueda descodificarse mediante la codificación del sistema de archivos en modo estricto. Puede ser `None` si no es posible generar dicho nombre de archivo.

test.support.os_helper.TESTFN_UNICODE

Establecido un nombre que no sea ASCII para un archivo temporal.

class test.support.os_helper.EnvironmentVarGuard

Clase utilizada para establecer o deshabilitar temporalmente las variables de entorno. Las instancias se pueden usar como un administrador de contexto y tienen una interfaz de diccionario completa para consultar/modificar el `os.environ` subyacente. Después de salir del administrador de contexto, todos los cambios en las variables de entorno realizados a través de esta instancia se revertirán.

Distinto en la versión 3.1: Añadido una interfaz de diccionario.

class test.support.os_helper.FakePath(*path*)

Simple path-like object. It implements the **`__fspath__()`** method which just returns the *path* argument. If *path* is an exception, it will be raised in **`__fspath__()`**.

EnvironmentVarGuard.set(*envvar*, *value*)

Se establece temporalmente la variable de entorno `envvar` en el valor de `value`.

EnvironmentVarGuard.unset(*envvar*, **others*)

Temporarily unset one or more environment variables.

Distinto en la versión 3.14: More than one environment variable can be unset.

test.support.os_helper.can_symlink()

Retorna `True` si el sistema operativo admite links simbólicos, de lo contrario `False`.

test.support.os_helper.can_xattr()

Retorna `True` si el sistema operativo admite *xattr*, de lo contrario `False`.

test.support.os_helper.change_cwd(*path*, *quiet=False*)

Un administrador de contexto que cambia temporalmente el directorio de trabajo actual a *path* y produce el directorio.

Si *quiet* es `False`, el administrador de contexto lanza una excepción en caso de error. De lo contrario, solo emite una advertencia y mantiene el directorio de trabajo actual igual.

test.support.os_helper.create_empty_file(*filename*)

Crea un archivo vacío con *filename*. Si ya existe, truncarlo.

test.support.os_helper.fd_count()

Cuenta el número de descriptores de archivo abiertos.

test.support.os_helper.fs_is_case_insensitive(*directory*)

Retorna `True` si el sistema de archivos para *directory* no distingue entre mayúsculas y minúsculas.

test.support.os_helper.make_bad_fd()

Se crea un descriptor de archivo no válido abriendo y cerrando un archivo temporal y retornando su descriptor.

test.support.os_helper.rmdir(*filename*)

Llama a **`os.rmdir()`** en *filename*. En las plataformas Windows, esto se envuelve con un ciclo de espera que verifica la existencia del archivo, que es necesario debido a los programas antivirus que pueden mantener los archivos abiertos y evitar que se eliminen.

test.support.os_helper.rmtree(*path*)

Llama a **`shutil.rmtree()`** en *path* o llama a **`os.lstat()`** y **`os.rmdir()`** para eliminar una ruta y su contenido. Al igual que con **`rmdir()`**, en las plataformas de Windows esto se envuelve con un ciclo de espera que verifica la existencia de los archivos.

@test.support.os_helper.skip_unless_symlink

Un decorador para ejecutar pruebas que requieren soporte para enlaces simbólicos.

@test.support.os_helper.skip_unless_xattr

Un decorador para ejecutar pruebas que requieren soporte para *xattr*.

test.support.os_helper.temp_cwd(*name='tempcwd'*, *quiet=False*)

Un administrador de contexto que crea temporalmente un nuevo directorio y cambia el directorio de trabajo actual (*CWD*).

El administrador de contexto crea un directorio temporal en el directorio actual con el nombre *name* antes de cambiar temporalmente el directorio de trabajo actual. Si *name* es `None` , el directorio temporal se crea usando **`tempfile.mkdtemp()`**.

Si *quiet* es `False` y no es posible crear o cambiar el *CWD*, se lanza un error. De lo contrario, solo se lanza una advertencia y se utiliza el *CWD* original.

test.support.os_helper.temp_dir(*path=None*, *quiet=False*)

Un administrador de contexto que crea un directorio temporal en *path* y produce el directorio.

Si *path* es `None`, el directorio temporal se crea usando **`tempfile.mkdtemp()`**. Si *quiet* es `False`, el administrador de contexto lanza una excepción en caso de error. De lo contrario, si se especifica *path* y no se puede crear, solo se emite una advertencia.

test.support.os_helper.temp_umask(*umask*)

Un administrador de contexto que establece temporalmente el proceso *umask*.

test.support.os_helper.unlink(*filename*)

Llama a `os.unlink()` en *filename*. Al igual que con `rmdir()`, en las plataformas Windows, esto se envuelve con un ciclo de espera que verifica la existencia del archivo.

`test.support.import_helper` — Utilities for import tests

The **`test.support.import_helper`** module provides support for import tests.

Added in version 3.10.

test.support.import_helper.forget(*module_name*)

Elimina el módulo llamado *module_name* de `sys.modules` y elimina los archivos compilados por bytes del módulo.

test.support.import_helper.import_fresh_module(*name*, *fresh=()*, *blocked=()*, *deprecated=False*)

Esta función importa y retorna una copia nueva del módulo Python nombrado eliminando el módulo nombrado de `sys.modules` antes de realizar la importación. Tenga en cuenta que a diferencia de **`reload()`**, el módulo original no se ve afectado por esta operación.

fresh es un iterable de nombres de módulos adicionales que también se eliminan del caché `sys.modules` antes de realizar la importación.

bloqueado es un iterable de nombres de módulos que se reemplazan con `None` en la memoria caché del módulo durante la importación para garantizar que los intentos de importarlos generen **`ImportError`**.

El módulo nombrado y los módulos nombrados en los parámetros *fresh* y *bloqueado* se guardan antes de comenzar la importación y luego se vuelven a insertar en `sys.modules` cuando se completa la importación fresca.

Los mensajes de deprecación de módulos y paquetes se suprimen durante esta importación si *deprecated* es `True`.

Esta función lanzará **`ImportError`** si el módulo nombrado no puede importarse.

Ejemplo de uso:

```
# Get copies of the warnings module for testing wit
# version being used by the rest of the test suite.
# C implementation, the other is forced to use the 
# implementation
py_warnings = import_fresh_module('warnings', block
c_warnings = import_fresh_module('warnings', fresh=
```

Added in version 3.1.

test.support.import_helper.import_module(*name*, *deprecated=False*, **, *required_on=()*)

Esta función importa y retorna el módulo nombrado. A diferencia de una importación normal, esta función lanza **`unittest.SkipTest`** si el módulo no se puede importar.

Los mensajes de deprecación de módulos y paquetes se suprimen durante esta importación si *deprecated* es `True`. Si se requiere un módulo en una plataforma pero es opcional para otros, establezca *required_on* en un iterable de prefijos de plataforma que se compararán con **`sys.platform`**.

Added in version 3.1.

test.support.import_helper.modules_setup()

Retorna una copia de `sys.modules`.

test.support.import_helper.modules_cleanup(*oldmodules*)

Elimina los módulos a excepción de *oldmodules* y `encodings` para preservar la memoria caché interna.

test.support.import_helper.unload(*name*)

Elimina *name* de `sys.modules`.

test.support.import_helper.make_legacy_pyc(*source*)

Mueve un archivo *pyc* de **PEP 3147** [https://peps.python.org/pep-3147/]/**PEP 488** [https://peps.python.org/pep-0488/] a su ubicación heredada y retorna la ruta del sistema de archivos al archivo de *pyc* heredado. El valor de origen es la ruta del sistema de archivos al archivo fuente. No es necesario que exista, sin embargo, debe existir el archivo PEP 3147/488 *pyc*.

class test.support.import_helper.CleanImport(**module_names*)

Un administrador de contexto para forzar la importación para retornar una nueva referencia de módulo. Esto es útil para probar comportamientos a nivel de módulo, como la emisión de un `DeprecationWarning` en la importación. Ejemplo de uso:

```
with CleanImport('foo'):
    importlib.import_module('foo')  # New reference
```

class test.support.import_helper.DirsOnSysPath(**paths*)

Un administrador de contexto para agregar directorios temporalmente a `sys.path`.

Esto hace una copia de `sys.path`, agrega cualquier directorio dado como argumento posicional, luego revierte **`sys.path`** a la configuración copiada cuando finaliza el contexto.

Tenga en cuenta que *all* `sys.path` produce modificaciones en el cuerpo del administrador de contexto, incluida la sustitución del objeto, se revertirán al final del bloque.

`test.support.warnings_helper` — Utilities for warnings tests

The **`test.support.warnings_helper`** module provides support for warnings tests.

Added in version 3.10.

test.support.warnings_helper.ignore_warnings(*, *category*)

Suppress warnings that are instances of *category*, which must be `Warning` or a subclass. Roughly equivalent to `warnings.catch_warnings()` with `warnings.simplefilter('ignore', category=category)`. For example:

```
@warning_helper.ignore_warnings(category=Deprecatio
def test_suppress_warning():
    # do something
```

Added in version 3.8.

test.support.warnings_helper.check_no_resource_warning(*testcase*)

Gestor de contexto para comprobar que no se ha lanzado un `ResourceWarning`. Debe eliminar el objeto que puede emitir **`ResourceWarning`** antes del final del administrador de contexto.

test.support.warnings_helper.check_syntax_warning(*testcase*, *statement*, *errtext*='', *, *lineno*=1, *offset*=None)

Prueba la advertencia de sintaxis en *statement* intentando compilar *statement*. Pruebe también que **`SyntaxWarning`** se emite solo una vez, y que se convertirá en **`SyntaxError`** cuando se convierta en error. *testcase* es la instancia **`unittest`** para la prueba. *errtext* es la expresión regular que debe coincidir con la representación de cadena del emitido **`SyntaxWarning`** y lanza **`SyntaxError`** . Si *lineno* no es `None`, se compara con la línea de advertencia y excepción. Si *offset* no es `None`, se compara con el desplazamiento de la excepción.

Added in version 3.8.

test.support.warnings_helper.check_warnings(**filters*, *quiet=True*)

A convenience wrapper for **`warnings.catch_warnings()`** that makes it easier to test that a warning was correctly raised. It is approximately equivalent to calling `warnings.catch_warnings(record=True)` with **`warnings.simplefilter()`** set to `always` and with the option to automatically validate the results that are recorded.

`check_warnings` acepta 2 tuplas de la forma `("mensaje regexp", WarningCategory)` como argumentos posicionales. Si se proporcionan uno o más *filters*, o si el argumento opcional de palabra clave *quiet* es `False`, se verifica para asegurarse de que las advertencias sean las esperadas: cada filtro especificado debe coincidir con al menos una de las advertencias lanzadas por el código adjunto o la prueba falla, y si se lanzan advertencias que no coinciden con ninguno de los filtros especificados, la prueba falla. Para deshabilitar la primera de estas comprobaciones, configure *quiet* en `True`.

Si no se especifican argumentos, el valor predeterminado es:

```
check_warnings(("", Warning), quiet=True)
```

En este caso, se capturan todas las advertencias y no se lanzaran errores.

En la entrada al administrador de contexto, se retorna una instancia de `WarningRecorder`. La lista de advertencias subyacentes de `catch_warnings()` está disponible a través del atributo `warnings` del objeto del registrador. Como conveniencia, también se puede acceder directamente a los atributos del objeto que representa la advertencia más reciente a través del objeto grabador (vea el ejemplo a continuación). Si no se ha lanzado ninguna advertencia, cualquiera de los atributos que de otro modo se esperarían en un objeto que representa una advertencia retornará `None`.

El objeto grabador (*recorder object*) también tiene un método `reset()`, que borra la lista de advertencias.

El administrador de contexto está diseñado para usarse así:

```
with check_warnings(("assertion is always true", Sy
                    ("", UserWarning)):
    exec('assert(False, "Hey!")')
    warnings.warn(UserWarning("Hide me!"))
```

En este caso, si no se generó ninguna advertencia, o si surgió alguna otra advertencia, `check_warnings()` lanzaría un error.

Cuando una prueba necesita analizar más profundamente las advertencias, en lugar de simplemente verificar si ocurrieron o no, se puede usar un código como este:

```
with check_warnings(quiet=True) as w:
    warnings.warn("foo")
    assert str(w.args[0]) == "foo"
    warnings.warn("bar")
    assert str(w.args[0]) == "bar"
    assert str(w.warnings[0].args[0]) == "foo"
    assert str(w.warnings[1].args[0]) == "bar"
    w.reset()
    assert len(w.warnings) == 0
```

Aquí se capturarán todas las advertencias, y el código de prueba prueba las advertencias capturadas directamente.

Distinto en la versión 3.2: Nuevos argumentos opcionales *filters* y *quiet*.

class test.support.warnings_helper.WarningsRecorder

La clase utilizada para registrar advertencias para pruebas unitarias. Consulte la documentación de `check_warnings()` arriba para obtener más detalles.

Depuración y perfilado

Estas bibliotecas le ayudan con el desarrollo de Python: el depurador le permite recorrer paso a paso el código, analizar marcos de pila y establecer puntos de interrupción, etc., y los perfiladores ejecutan código y le proporcionan un desglose detallado de los tiempos de ejecución, lo que le permite identificar cuellos de botella en sus programas. Los eventos de auditoría proporcionan visibilidad de los comportamientos en tiempo de ejecución que, de lo contrario, requerirían depuración o parches intrusivos.

- Tabla de auditoría de eventos
- **`bdb`** — Debugger framework
- **`faulthandler`** — Dump the Python traceback
    - Volcar el rastreo
    - Dumping the C stack
        - C Stack Compatibility
    - Estado del gestor de fallos
    - Volcar los rastreos después de un tiempo de espera
    - Volcar el rastreo en una señal del usuario
    - Problema con descriptores de archivo
    - Ejemplo
- **`pdb`** — The Python Debugger
    - Command-line interface
    - Debugger commands
- Los perfiladores de Python
    - Introducción a los perfiladores
    - Manual instantáneo de usuario
    - **`profile`** and **`cProfile`** Module Reference

  - La clase **`Stats`**
  - ¿Qué es el perfil determinista?
  - Limitaciones
  - Calibración
  - Usando un temporizador personalizado

- **`timeit`** — Measure execution time of small code snippets

  - Ejemplos básicos
  - Interfaz de Python
  - Interfaz de línea de comandos
  - Ejemplos

- **`trace`** — Trace or track Python statement execution

  - Uso de la línea de comandos

    - Opciones principales
    - Modificadores
    - Filtros

  - Interfaz programática

- **`tracemalloc`** — Trace memory allocations

  - Ejemplos

    - Mostrar los 10 principales
    - Calcula las diferencias
    - Consigue el seguimiento del bloque de memoria
    - **`Los 10 más bonitos`**

      - Graba los tamaños actual y máximo de todos los bloques de memoria rastreados

  - API

    - Funciones
    - Filtro de dominio
    - Filtro
    - Cuadro
    - Captura instantánea

- Estadística
- StatisticDiff
- Rastro
- Seguimiento

Tabla de auditoría de eventos

This table contains all events raised by **`sys.audit()`** or **`PySys_Audit()`** calls throughout the CPython runtime and the standard library. These calls were added in 3.8 or later (see **PEP 578** [https://peps.python.org/pep-0578/]).

Ver **`sys.addaudithook()`** y **`PySys_AddAuditHook()`** para informarse en el manejo de estos eventos.

Esta tabla es generada desde la documentación de CPython, y no debe guardar eventos lanzados por otras implementaciones. Ver la documentación de tus especificaciones en tiempo de ejecución para los eventos lanzados recientemente.

Audit event	Arguments	References
_thread.start_new_thread	`function, args, kwargs`	[1]
array.__new__	`typecode, initializer`	[1]
builtins.breakpoint	`breakpointhook`	[1]
builtins.id	`id`	[1]
builtins.input	`prompt`	[1]
builtins.input/result	`result`	[1]
code.__new__	`code, filename, name, argcount, posonlyargcount, kwonlyargcount, nlocals, stacksize, flags`	[1]
compile	`source, filename`	[1]
cpython.PyConfig_Set	`name, value`	[1]
cpython.PyInterpreterState_Clear		[1]
cpython.PyInterpreterState_New		[1]
cpython._PySys_ClearAuditHooks		[1]
cpython.remote_debugger_script	`script_path`	[1]
cpython.run_command	`command`	[1]
cpython.run_file	`filename`	[1]
cpython.run_interactivehook	`hook`	[1]
cpython.run_module	`module-name`	[1]
cpython.run_startup	`filename`	[1]
cpython.run_stdin		[1][2][3]

ctypes.addressof	`obj`	[1]
ctypes.call_function	`func_pointer, arguments`	[1]
ctypes.cdata	`address`	[1]
ctypes.cdata/buffer	`pointer, size, offset`	[1][2]
ctypes.create_string_buffer	`init, size`	[1]
ctypes.create_unicode_buffer	`init, size`	[1]
ctypes.dlopen	`name`	[1]
ctypes.dlsym	`library, name`	[1]
ctypes.dlsym/handle	`handle, name`	[1]
ctypes.get_errno		[1]
ctypes.get_last_error		[1]
ctypes.memoryview_at	`address, size, readonly`	[1]
ctypes.set_errno	`errno`	[1]
ctypes.set_exception	`code`	[1]
ctypes.set_last_error	`error`	[1]
ctypes.string_at	`ptr, size`	[1]
ctypes.wstring_at	`ptr, size`	[1]
ensurepip.bootstrap	`root`	[1]
exec	`code_object`	[1][2]
fcntl.fcntl	`fd, cmd, arg`	[1]
fcntl.flock	`fd, operation`	[1]
fcntl.ioctl	`fd, request, arg`	[1]
fcntl.lockf	`fd, cmd, len, start, whence`	[1]
ftplib.connect	`self, host, port`	[1]
ftplib.sendcmd	`self, cmd`	[1][2]
function.__new__	`code`	[1]
gc.get_objects	`generation`	[1]
gc.get_referents	`objs`	[1]
gc.get_referrers	`objs`	[1]
glob.glob	`pathname, recursive`	[1][2]
glob.glob/2	`pathname, recursive, root_dir, dir_fd`	[1][2]
http.client.connect	`self, host, port`	[1]
http.client.send	`self, data`	[1]
imaplib.open	`self, host, port`	[1]
imaplib.send	`self, data`	[1]
import	`module, filename, sys.path, sys.meta_path, sys.path_hooks`	[1]
marshal.dumps	`value, version`	[1]
marshal.load		[1]
marshal.loads	`bytes`	[1]

mmap.__new__	fileno, length, access, offset	[1]
msvcrt.get_osfhandle	fd	[1]
msvcrt.locking	fd, mode, nbytes	[1]
msvcrt.open_osfhandle	handle, flags	[1]
object.__delattr__	obj, name	[1]
object.__getattr__	obj, name	[1]
object.__setattr__	obj, name, value	[1]
open	path, mode, flags	[1][2][3]
os.add_dll_directory	path	[1]
os.chdir	path	[1][2]
os.chflags	path, flags	[1][2]
os.chmod	path, mode, dir_fd	[1][2][3]
os.chown	path, uid, gid, dir_fd	[1][2][3]
os.exec	path, args, env	[1]
os.fork		[1]
os.forkpty		[1]
os.fwalk	top, topdown, onerror, follow_symlinks, dir_fd	[1]
os.getxattr	path, attribute	[1]
os.kill	pid, sig	[1]
os.killpg	pgid, sig	[1]
os.link	src, dst, src_dir_fd, dst_dir_fd	[1]
os.listdir	path	[1]
os.listdrives		[1]
os.listmounts	volume	[1]
os.listvolumes		[1]
os.listxattr	path	[1]
os.lockf	fd, cmd, len	[1]
os.mkdir	path, mode, dir_fd	[1][2]
os.posix_spawn	path, argv, env	[1][2]
os.putenv	key, value	[1]
os.remove	path, dir_fd	[1][2][3]
os.removexattr	path, attribute	[1]
os.rename	src, dst, src_dir_fd, dst_dir_fd	[1][2][3]
os.rmdir	path, dir_fd	[1]
os.scandir	path	[1]
os.setxattr	path, attribute, value, flags	[1]
os.spawn	mode, path, args, env	[1]
os.startfile	path, operation	[1]
os.startfile/2	path, operation, arguments, cwd, show_cmd	[1]
os.symlink	src, dst, dir_fd	[1]

os.system	command	[1]
os.truncate	fd, length	[1][2]
os.unsetenv	key	[1]
os.utime	path, times, ns, dir_fd	[1]
os.walk	top, topdown, onerror, followlinks	[1]
pathlib.Path.glob	self, pattern	[1]
pathlib.Path.rglob	self, pattern	[1]
pdb.Pdb		[1]
pickle.find_class	module, name	[1]
poplib.connect	self, host, port	[1][2]
poplib.putline	self, line	[1][2]
pty.spawn	argv	[1]
resource.prlimit	pid, resource, limits	[1]
resource.setrlimit	resource, limits	[1]
setopencodehook		[1]
shutil.chown	path, user, group	[1]
shutil.copyfile	src, dst	[1][2][3]
shutil.copymode	src, dst	[1][2]
shutil.copystat	src, dst	[1][2]
shutil.copytree	src, dst	[1]
shutil.make_archive	base_name, format, root_dir, base_dir	[1]
shutil.move	src, dst	[1]
shutil.rmtree	path, dir_fd	[1]
shutil.unpack_archive	filename, extract_dir, format	[1]
signal.pthread_kill	thread_id, signalnum	[1]
smtplib.connect	self, host, port	[1]
smtplib.send	self, data	[1]
socket.__new__	self, family, type, protocol	[1]
socket.bind	self, address	[1]
socket.connect	self, address	[1][2]
socket.getaddrinfo	host, port, family, type, protocol	[1]
socket.gethostbyaddr	ip_address	[1]
socket.gethostbyname	hostname	[1][2]
socket.gethostname		[1]
socket.getnameinfo	sockaddr	[1]
socket.getservbyname	servicename, protocolname	[1]
socket.getservbyport	port, protocolname	[1]
socket.sendmsg	self, address	[1]
socket.sendto	self, address	[1]
socket.sethostname	name	[1]

sqlite3.connect	database	[1]
sqlite3.connect/handle	connection_handle	[1]
sqlite3.enable_load_extension	connection, enabled	[1]
sqlite3.load_extension	connection, path	[1]
subprocess.Popen	executable, args, cwd, env	[1]
sys._current_exceptions		[1]
sys._current_frames		[1]
sys._getframe	frame	[1]
sys._getframemodulename	depth	[1]
sys.addaudithook		[1][2]
sys.excepthook	hook, type, value, traceback	[1]
sys.monitoring.register_callback	func	[1]
sys.remote_exec	pid	[1]
sys.set_asyncgen_hooks_finalizer		[1]
sys.set_asyncgen_hooks_firstiter		[1]
sys.setprofile		[1]
sys.settrace		[1]
sys.unraisablehook	hook, unraisable	[1]
syslog.closelog		[1]
syslog.openlog	ident, logoption, facility	[1]
syslog.setlogmask	maskpri	[1]
syslog.syslog	priority, message	[1]
tempfile.mkdtemp	fullpath	[1][2]
tempfile.mkstemp	fullpath	[1][2][3]
time.sleep	secs	[1]
urllib.Request	fullurl, data, headers, method	[1]
webbrowser.open	url	[1]
winreg.ConnectRegistry	computer_name, key	[1]
winreg.CreateKey	key, sub_key, access	[1][2]
winreg.DeleteKey	key, sub_key, access	[1][2]
winreg.DeleteValue	key, value	[1]
winreg.DisableReflectionKey	key	[1]
winreg.EnableReflectionKey	key	[1]
winreg.EnumKey	key, index	[1]
winreg.EnumValue	key, index	[1]
winreg.ExpandEnvironmentStrings	str	[1]
winreg.LoadKey	key, sub_key, file_name	[1]
winreg.OpenKey	key, sub_key, access	[1]
winreg.OpenKey/result	key	[1][2][3]
winreg.PyHKEY.Detach	key	[1]

winreg.QueryInfoKey	`key`	[1]
winreg.QueryReflectionKey	`key`	[1]
winreg.QueryValue	`key, sub_key, value_name`	[1][2]
winreg.SaveKey	`key, file_name`	[1]
winreg.SetValue	`key, sub_key, type, value`	[1][2]

Los siguientes eventos se generan internamente y no corresponden a ninguna API pública de CPython:

Evento de auditoría	**Argumentos**
_winapi.CreateFile	`file_name, desired_access, share_mode, creation_disposition, flags_and_attributes`
_winapi.CreateJunction	`src_path, dst_path`
_winapi.CreateNamedPipe	`name, open_mode, pipe_mode`
_winapi.CreatePipe	
_winapi.CreateProcess	`application_name, command_line, current_directory`
_winapi.OpenProcess	`process_id, desired_access`
_winapi.TerminateProcess	`handle, exit_code`
_posixsubprocess.fork_exec	`exec_list, args, env`
ctypes.PyObj_FromPtr	`obj`

Added in version 3.14: The `_posixsubprocess.fork_exec` internal audit event.

`bdb` — Debugger framework

Source code: Lib/bdb.py [https://github.com/python/cpython/tree/3.14/Lib/bdb.py]

The `bdb` module handles basic debugger functions, like setting breakpoints or managing execution via the debugger.

La siguiente excepción es definida:

exception bdb.BdbQuit

Excepción lanzada por la clase `Bdb` al salir del depurador.

The `bdb` module also defines two classes:

class bdb.Breakpoint(*self*, *file*, *line*, *temporary=False*, *cond=None*, *funcname=None*)

Esta clase implementa puntos de interrupción temporales, permitiendo ignorar recuentos, desactivar y (re-)activar puntos de interrupción y definir condiciones de interrupción para los mismos.

Los puntos de interrupción se indexan numéricamente mediante una lista llamada `bpbynumber` y por pares `(archivo, linea)` mediante `bplist`. En el primer caso se apunta a una única instancia de `Breakpoint`. En el segundo caso se apunta a una lista de tales instancias, ya que puede haber más de un punto de interrupción por línea.

Al crear un punto de interrupción, el `nombre del archivo` asociado debe estar en su forma canónica. Si se define un `funcname`, se contará un `hit` de punto de interrupción cuando se ejecute la primera línea de esa función. Un punto de interrupción `condicional` siempre cuenta un `hit`.

Las instancias de la clase **`Breakpoint`** tienen los siguientes métodos:

deleteMe()

Elimina el punto de interrupción de la lista asociada a un archivo/línea. Si es el último punto de interrupción en esa posición, también borra la entrada correspondiente del archivo/línea correspondiente.

enable()

Marca el punto de interrupción como habilitado.

disable()

Marca el punto de interrupción como deshabilitado.

bpformat()

Retorna una cadena de caracteres que contiene toda la información sobre el punto de interrupción, apropiadamente formateada:

- El número del punto de interrupción.
- Estado temporal (del o keep)
- Posición del archivo/línea.
- Condición de interrupción.
- Número de veces para ignorar.
- Número de veces alcanzado.

Added in version 3.2.

bpprint(*out=None*)

Imprime la salida de **`bpformat()`** en el archivo *out*, o en la salida estándar si es `None`.

Las instancias de la clase **`Breakpoint`** tienen los siguientes atributos:

file

Nombre del archivo de la clase **`Breakpoint`**.

line

Número de línea de la clase `Breakpoint` dentro de `file`.

temporary

`True` if a `Breakpoint` at (file, line) is temporary.

cond

Condición para evaluar una clase `Breakpoint` en (file, line).

funcname

Nombre de la función que define si se alcanza un `Breakpoint` al ingresar a la función.

enabled

`True` if `Breakpoint` is enabled.

bpbynumber

Índice numérico para una sola instancia de un `Breakpoint`.

bplist

Diccionario de instancias `Breakpoint` indexadas por tuplas (`file`, `line`).

ignore

Número de veces para ignorar un `Breakpoint`.

hits

Recuento del número de veces que se alcanzó un `Breakpoint`.

class bdb.Bdb(*skip=None*, *backend='settrace'*)

La clase `Bdb` actúa como clase base del depurador genérico de Python.

Esta clase se encarga de los detalles de la funcionalidad de seguimiento; una clase derivada debería encargarse de implementar la interacción con el usuario. Un ejemplo es la clase de depuración estándar (**`pdb.Pdb`**).

El argumento *skip*, si se proporciona, debe ser un iterable con patrones de nombre de archivo, al estilo del módulo glob. El depurador no entrará en aquellos marcos de ejecución que se originen en un módulo que coincida con uno de estos patrones. Para determinar si un marco de ejecución se origina en un módulo determinado, se hace uso de `__name__` en las variables globales del marco de ejecución dado.

The *backend* argument specifies the backend to use for **`Bdb`**. It can be either `'settrace'` or `'monitoring'`. `'settrace'` uses **`sys.settrace()`** which has the best backward compatibility. The `'monitoring'` backend uses the new **`sys.monitoring`** that was introduced in Python 3.12, which can be much more efficient because it can disable unused events. We are trying to keep the exact interfaces for both backends, but there are some differences. The debugger developers are encouraged to use the `'monitoring'` backend to achieve better performance.

Distinto en la versión 3.1: Added the *skip* parameter.

Distinto en la versión 3.14: Added the *backend* parameter.

Los siguientes métodos de la clase **`Bdb`** normalmente no necesitan ser redefinidos.

canonic(*filename*)

Retorna la forma canónica de *filename*.

Para nombres de archivos reales, la forma canónica es una **`ruta absoluta normalizada entre mayúsculas y minúsculas`** que depende del sistema operativo. Un *filename* con corchetes angulares, como `"<stdin>"` generado en modo interactivo, se retorna sin cambios.

start_trace(*self*)

Start tracing. For `'settrace'` backend, this method is equivalent to `sys.settrace(self.trace_dispatch)`

Added in version 3.14.

stop_trace(*self*)

Stop tracing. For `'settrace'` backend, this method is equivalent to `sys.settrace(None)`

Added in version 3.14.

reset()

Set the **`botframe`**, **`stopframe`**, **`returnframe`** and **`quitting`** attributes with values ready to start debugging.

trace_dispatch(*frame*, *event*, *arg*)

Esta función se instala como función de seguimiento de los marcos de ejecución depurados. Su valor de retorno es la nueva función de seguimiento a usar (en la mayoría de los casos, ella misma).

La implementación predeterminada decide cómo despachar un marco de ejecución, dependiendo del tipo de evento (pasado como una cadena) que está a punto de ejecutarse. *event* puede tomar uno de los siguientes valores:

- `"line"`: Se va a ejecutar una nueva línea de código.
- `"call"`: Está a punto de llamarse a una función o se ha entrado en otro bloque de código.
- `"return"`: Una función u otro bloque de código está a punto de retornar.
- `"exception"`: Ha ocurrido una excepción.
- `"c_call"`: Una función de C está a punto de llamarse .

- `"c_return"`: Una función de C ha retornado.
- `"c_exception"`: Una función de C ha lanzado una excepción.

Para los eventos de Python, son llamadas funciones especializadas (ver más abajo). En cambio, para los eventos de C no se realiza ninguna acción.

El parámetro *arg* depende del evento previo.

Consulta la documentación de `sys.settrace()` para obtener más información sobre la función de seguimiento. Para obtener más información sobre los objetos código y los objetos marco consulte Jerarquía de tipos estándar.

dispatch_line(*frame*)

If the debugger should stop on the current line, invoke the `user_line()` method (which should be overridden in subclasses). Raise a `BdbQuit` exception if the `quitting` flag is set (which can be set from `user_line()`). Return a reference to the `trace_dispatch()` method for further tracing in that scope.

dispatch_call(*frame*, *arg*)

If the debugger should stop on this function call, invoke the `user_call()` method (which should be overridden in subclasses). Raise a `BdbQuit` exception if the `quitting` flag is set (which can be set from `user_call()`). Return a reference to the `trace_dispatch()` method for further tracing in that scope.

dispatch_return(*frame*, *arg*)

If the debugger should stop on this function return, invoke the `user_return()` method (which should be overridden in subclasses). Raise a `BdbQuit` exception if the `quitting` flag is set (which can be set from

`user_return()`). Return a reference to the `trace_dispatch()` method for further tracing in that scope.

dispatch_exception(*frame*, *arg*)

If the debugger should stop at this exception, invokes the `user_exception()` method (which should be overridden in subclasses). Raise a `BdbQuit` exception if the `quitting` flag is set (which can be set from **`user_exception()`**). Return a reference to the `trace_dispatch()` method for further tracing in that scope.

Las clases derivadas normalmente no necesitan redefinir los siguientes métodos, pero pueden hacerlo si necesitan redefinir la definición de parada y los puntos de interrupción.

is_skipped_module(*module_name*)

Return `True` if *module_name* matches any skip pattern.

stop_here(*frame*)

Return `True` if *frame* is below the starting frame in the stack.

break_here(*frame*)

Return `True` if there is an effective breakpoint for this line.

Comprueba si existe un punto de interrupción de línea o de una función y si está en vigor. Elimina puntos de interrupción temporales basados en la información de `effective()`.

break_anywhere(*frame*)

Return `True` if any breakpoint exists for *frame*'s filename.

Las clases derivadas deben redefinir estos métodos para adquirir el control sobre las operaciones de depuración.

user_call(*frame*, *argument_list*)

Se llama desde `dispatch_call()` si una interrupción puede detenerse dentro de la función llamada.

argument_list is not used anymore and will always be `None`. The argument is kept for backwards compatibility.

user_line(*frame*)

Se llama desde `dispatch_line()` cuando `stop_here()` o `break_here()` retornan `True`.

user_return(*frame*, *return_value*)

Se llama desde `dispatch_return()` cuando `stop_here()` retorna `True`.

user_exception(*frame*, *exc_info*)

Se llama desde `dispatch_exception()` cuando `stop_here()` retorna `True`.

do_clear(*arg*)

Maneja cómo un punto de interrupción debe ser eliminado cuando es temporal.

Este método debe ser implementado por las clases derivadas.

Las clases derivadas y los clientes pueden llamar a los siguientes métodos para influir en el estado de transición.

set_step()

Se detiene después de una línea de código.

set_next(*frame*)

Se detiene en la siguiente línea del marco de ejecución dado o en la de uno inferior.

set_return(*frame*)

Se detiene cuando se retorna desde el marco de ejecución dado.

set_until(*frame*, *lineno=None*)

Se detiene cuando se alcanza la línea con el *lineno* mayor que el actual o al retornar desde el marco actual.

set_trace([*frame*])

Inicia la depuración desde el marco de ejecución *frame*. Si *frame* no se especifica, la depuración se inicia desde el marco de ejecución que produce la llamada.

Distinto en la versión 3.13: **`set_trace()`** will enter the debugger immediately, rather than on the next line of code to be executed.

set_continue()

Se detiene solo en los puntos de interrupción o cuando haya terminado. Si no hay puntos de interrupción, se configura la función de seguimiento del sistema en `None`.

set_quit()

Set the **`quitting`** attribute to `True`. This raises **`BdbQuit`** in the next call to one of the **`dispatch_*()`** methods.

Las clases derivadas y los clientes pueden llamar a los siguientes métodos para manipular los puntos de interrupción. Estos métodos retornan una cadena de caracteres que contiene un mensaje de error si algo fue mal, o `None` si todo fue correctamente.

set_break(*filename*, *lineno*, *temporary=False*, *cond=None*, *funcname=None*)

Establece un nuevo punto de interrupción. Si la línea en la posición *lineno* no existe en el archivo con nombre *filename* pasado como argumento, se retorna un mensaje de error. *filename* debe estar en su forma canónica, tal como se describe en el método **`canonic()`**.

clear_break(*filename*, *lineno*)

Elimina los puntos de interrupción en *filename* y *lineno*. Si no se estableció ninguno, se retorna un mensaje de error.

clear_bpbynumber(*arg*)

Elimina el punto de interrupción que tiene el índice *arg* en **`Breakpoint.bpbynumber`**. Si *arg* no es un valor numérico o es un indice fuera de rango, se retorna un mensaje de error.

clear_all_file_breaks(*filename*)

Elimina todos los puntos de interrupción en *filename*. Si no se estableció ninguno, se retorna un mensaje de error.

clear_all_breaks()

Elimina todos los puntos de interrupción. Si no se estableció ninguno, se retorna un mensaje de error.

get_bpbynumber(*arg*)

Retorna un punto de interrupción especificado por el número dado. Si *arg* es una cadena de caracteres, será convertida en un número. Si *arg* es una cadena no numérica, o el punto de interrupción dado nunca existió o ya ha sido eliminado, se lanza una excepción **`ValueError`**.

Added in version 3.2.

get_break(*filename*, *lineno*)

Return `True` if there is a breakpoint for *lineno* in *filename*.

get_breaks(*filename*, *lineno*)

Retorna todos los puntos de interrupción en la línea número *lineno* del archivo *filename*, o una lista vacía si no se ha establecido ninguno.

get_file_breaks(*filename*)

Retorna todos los puntos de interrupción en el archivo *filename*, o una lista vacía si no hay ninguno establecido.

get_all_breaks()

Retorna todos los puntos de interrupción establecidos.

Derived classes and clients can call the following methods to disable and restart events to achieve better performance. These methods only work when using the `'monitoring'` backend.

disable_current_event()

Disable the current event until the next time **`restart_events()`** is called. This is helpful when the debugger is not interested in the current line.

Added in version 3.14.

restart_events()

Restart all the disabled events. This function is automatically called in `dispatch_*` methods after `user_*` methods are called. If the `dispatch_*` methods are not overridden, the disabled events will be restarted after each user interaction.

Added in version 3.14.

Las clases derivadas y clientes pueden llamar los siguientes métodos para obtener una estructura de datos que representa un seguimiento de pila.

get_stack(*f*, *t*)

Retorna una lista de tuplas (frame, lineno) en un seguimiento de pila y un tamaño.

El marco llamado más recientemente es el último en la lista. El tamaño es el número de marcos por debajo del marco en el que se invocó el depurador.

format_stack_entry(*frame_lineno*, *lprefix=': '*)

Retorna una cadena de caracteres con información sobre una entrada de pila, identificada por una tupla de la forma `(frame, lineno)`. La cadena de caracteres que retorna contiene:

- El nombre de archivo canónico que contiene el marco de ejecución.
- El nombre de la función o `"<lambda>"`.
- Los argumentos de entrada.
- El valor de retorno.
- La linea de código (si existe).

Los dos métodos descritos a continuación pueden ser llamados por los clientes para usar un depurador que se encargue de depurar un statement, proporcionado como una cadena de caracteres.

run(*cmd*, *globals=None*, *locals=None*)

Debug a statement executed via the **`exec()`** function. *globals* defaults to **`__main__.__dict__`**, *locals* defaults to *globals*.

runeval(*expr*, *globals=None*, *locals=None*)

Depura una expresión ejecutada mediante la función `eval()`. *globals* y *locals* tienen el mismo significado que en `run()`.

runctx(*cmd*, *globals*, *locals*)

Definido solo por compatibilidad con versiones anteriores. Llama al método `run()`.

runcall(*func*, /, **args*, ***kwds*)

Depura una única llamada a función y retorna su resultado.

Por último, el módulo también define las siguientes funciones:

bdb.checkfuncname(*b*, *frame*)

Return `True` if we should break here, depending on the way the `Breakpoint` *b* was set.

Si se estableció a través del número de línea, comprueba si `b.line` es el mismo que el de *frame*. Si el punto de interrupción se estableció a través del `nombre de la función`, se tiene que comprobar que estamos en el *frame* correcto (la función correcta) y si se está en su primera línea ejecutable.

bdb.effective(*file*, *line*, *frame*)

Retorna `(active breakpoint, delete temporary flag)` o `(None, None)` como el punto de interrupción para actuar.

The *active breakpoint* is the first entry in `bplist` for the (`file`, `line`) (which must exist) that is `enabled`, for which `checkfuncname()` is true, and that has neither a false `condition` nor positive `ignore` count. The *flag*, meaning that a temporary breakpoint should be deleted, is `False` only when the **`cond`** cannot be evaluated (in which case, **`ignore`** count is ignored).

If no such entry exists, then `(None, None)` is returned.

bdb.set_trace()

Inicia la depuración usando una instancia de **`Bdb`**, partiendo desde el marco de ejecución que realiza la llamada.

`faulthandler` — Dump the Python traceback

Added in version 3.3.

This module contains functions to dump Python tracebacks explicitly, on a fault, after a timeout, or on a user signal. Call **`faulthandler.enable()`** to install fault handlers for the **`SIGSEGV`**, **`SIGFPE`**, **`SIGABRT`**, **`SIGBUS`**, and **`SIGILL`** signals. You can also enable them at startup by setting the **`PYTHONFAULTHANDLER`** environment variable or by using the **`-X`** `faulthandler` command line option.

The fault handler is compatible with system fault handlers like Apport or the Windows fault handler. The module uses an alternative stack for signal handlers if the **`sigaltstack()`** function is available. This allows it to dump the traceback even on a stack overflow.

El gestor de fallos se llama en casos catastróficos y, por lo tanto, solo puede utilizar funciones seguras en señales (por ejemplo, no puede asignar memoria en el heap). Debido a esta limitación, el volcado del rastreo es mínimo comparado a los rastreos normales de Python:

- Solo se soporta ASCII. El gestor de errores `backslashreplace` se utiliza en la codificación.
- Cada cadena de caracteres está limitada a 500 caracteres.
- Solo se muestran el nombre de archivo, el nombre de la función y el número de línea. (sin código fuente)
- Está limitado a 100 frames y 100 hilos.
- El orden se invierte: la llamada más reciente se muestra primero.

Por defecto, el rastreo de Python se escribe en **`sys.stderr`**. Para ver los rastreos, las aplicaciones deben ejecutarse en la terminal. Alternativamente se puede pasar un archivo de registro a `faulthandler.enable()`.

El módulo está implementado en C, así los rastreos se pueden volcar en un fallo o cuando Python está en bloqueo mutuo.

El Modo de Desarrollo Python llama a `failurehandler.enable()` al inicio de Python.

Ver también

Módulo `pdb`
Depurador interactivo de código fuente para programas en Python.

Módulo `traceback`
Interfaz estándar para extraer, formatear e imprimir trazas de pila de programas en Python.

Volcar el rastreo

faulthandler.dump_traceback(*file=sys.stderr*, *all_threads=True*)

Vuelca los rastreos de todos los hilos en el archivo *file*. Si *all_threads* es `False`, vuelca solo el hilo actual.

Ver también

`traceback.print_tb()`, que se puede utilizar para imprimir un objeto de traza de pila.

Distinto en la versión 3.5: Se añadió soporte para pasar el descriptor de archivo a esta función.

Dumping the C stack

Added in version 3.14.

faulthandler.dump_c_stack(*file=sys.stderr*)

Dump the C stack trace of the current thread into *file*.

If the Python build does not support it or the operating system does not provide a stack trace, then this prints an error in place of a dumped C stack.

C Stack Compatibility

If the system does not support the C-level *backtrace(3)* *[https://manpages.debian.org/backtrace(3)]* or *dladdr1(3)* *[https://manpages.debian.org/dladdr1(3)]*, then C stack dumps will not work. An error will be printed instead of the stack.

Additionally, some compilers do not support CPython's implementation of C stack dumps. As a result, a different error may be printed instead of the stack, even if the operating system supports dumping stacks.

Nota

Dumping C stacks can be arbitrarily slow, depending on the DWARF level of the binaries in the call stack.

Estado del gestor de fallos

faulthandler.enable(*file=sys.stderr*, *all_threads=True*, *c_stack=True*)

Enable the fault handler: install handlers for the **`SIGSEGV`**, **`SIGFPE`**, **`SIGABRT`**, **`SIGBUS`** and **`SIGILL`** signals to dump the Python traceback. If *all_threads* is `True`, produce tracebacks for every running thread. Otherwise, dump only the current thread.

El archivo *file* se debe mantener abierto hasta que se desactive el gestor de fallos: ver problema con descriptores de archivo.

If *c_stack* is `True`, then the C stack trace is printed after the

Python traceback, unless the system does not support it. See **`dump_c_stack()`** for more information on compatibility.

Distinto en la versión 3.5: Se añadió soporte para pasar el descriptor de archivo a esta función.

Distinto en la versión 3.6: En Windows, también se instaló un gestor para la excepción de Windows.

Distinto en la versión 3.10: Ahora el volcado menciona si se está ejecutando una colección de recolectores de basura si *all_threads* es verdadero.

Distinto en la versión 3.14: Only the current thread is dumped if the GIL is disabled to prevent the risk of data races.

Distinto en la versión 3.14: The dump now displays the C stack trace if *c_stack* is true.

faulthandler.disable()

Desactiva el gestor de fallos: desinstala los gestores de señales instalados por **`enable()`**.

faulthandler.is_enabled()

Comprueba si el gestor de fallos está activado.

Volcar los rastreos después de un tiempo de espera

faulthandler.dump_traceback_later(*timeout*, *repeat=False*, *file=sys.stderr*, *exit=False*)

Dump the tracebacks of all threads, after a timeout of *timeout* seconds, or every *timeout* seconds if *repeat* is `True`. If *exit* is `True`, call **`_exit()`** with status=1 after dumping the tracebacks. (Note **`_exit()`** exits the process immediately, which means it doesn't do any cleanup like flushing file buffers.) If the function is called twice, the new call replaces previous parameters and resets the timeout. The timer has a sub-second resolution.

El archivo *file* se debe mantener abierto hasta que se vuelque el rastreo o se llame a **`cancel_dump_traceback_later()`**: ver problema con descriptores de archivo.

Esta función está implementada utilizando un hilo vigilante.

Distinto en la versión 3.5: Se añadió soporte para pasar el descriptor de archivo a esta función.

Distinto en la versión 3.7: Ahora esta función está siempre disponible.

faulthandler.cancel_dump_traceback_later()

Cancela la última llamada a **`dump_traceback_later()`**.

Volcar el rastreo en una señal del usuario

faulthandler.register(*signum*, *file*=*sys.stderr*, *all_threads*=*True*, *chain*=*False*)

Registra una señal del usuario: instala un gestor para la señal *signum* para volcar el rastreo de todos los hilos, o del hilo actual si *all_threads* es `False`, en el archivo file. Llama al gestor previo si chain es `True`.

El archivo *file* se debe mantener abierto hasta que la señal sea anulada por **`unregister()`**: ver problema con descriptores de archivo.

No está disponible en Windows.

Distinto en la versión 3.5: Se añadió soporte para pasar el descriptor de archivo a esta función.

faulthandler.unregister(*signum*)

Anula una señal del usuario: desinstala el gestor de la señal *signum* instalada por **`register()`**. Retorna `True` si la señal fue registrada, `False` en otro caso.

No está disponible en Windows.

Problema con descriptores de archivo

`enable()`, **`dump_traceback_later()`** y **`register()`** guardan el descriptor de archivo de su argumento *file*. Si se cierra el archivo y su descriptor de archivo es reutilizado por un nuevo archivo, o si se usa **`os.dup2()`** para reemplazar el descriptor de archivo, el rastreo se escribirá en un archivo diferente. Llame a estas funciones nuevamente cada vez que se reemplace el archivo.

Ejemplo

Ejemplo de un fallo de segmentación en Linux con y sin activar el gestor de fallos:

```
$ python -c "import ctypes; ctypes.string_at(0)"
Segmentation fault

$ python -q -X faulthandler
>>> import ctypes
>>> ctypes.string_at(0)
Fatal Python error: Segmentation fault

Current thread 0x00007fb899f39700 (most recent call firs
  File "/opt/python/Lib/ctypes/__init__.py", line 486 i
  File "<stdin>", line 1 in <module>

Current thread's C stack trace (most recent call first):
  Binary file "/opt/python/python", at _Py_DumpStack+0x4
  Binary file "/opt/python/python", at +0x32dcbd [0x5b27
  Binary file "/opt/python/python", at +0x32df8a [0x5b27
  Binary file "/usr/lib/libc.so.6", at +0x3def0 [0x77b73
  Binary file "/usr/lib/libc.so.6", at +0x17ef9c [0x77b7
  Binary file "/opt/python/build/lib.linux-x86_64-3.14/_
  Binary file "/usr/lib/libffi.so.8", at +0x7976 [0x77b7
```

```
    Binary file "/usr/lib/libffi.so.8", at +0x413c [0x77b7
    Binary file "/usr/lib/libffi.so.8", at ffi_call+0x12e
    Binary file "/opt/python/build/lib.linux-x86_64-3.14/_
    Binary file "/opt/python/build/lib.linux-x86_64-3.14/_
    Binary file "/opt/python/build/lib.linux-x86_64-3.14/_
    Binary file "/opt/python/python", at _PyObject_MakeTpC
    Binary file "/opt/python/python", at +0x11bab6 [0x5b27
    Binary file "/opt/python/python", at PyObject_Vectorca
    Binary file "/opt/python/python", at _PyEval_EvalFrame
    Binary file "/opt/python/python", at +0x2818e6 [0x5b27
    Binary file "/opt/python/python", at +0x281aab [0x5b27
    Binary file "/opt/python/python", at PyEval_EvalCode+0
    Binary file "/opt/python/python", at +0x255957 [0x5b27
    Binary file "/opt/python/python", at +0x255ab4 [0x5b27
    Binary file "/opt/python/python", at _PyEval_EvalFrame
    Binary file "/opt/python/python", at +0x2818e6 [0x5b27
    Binary file "/opt/python/python", at +0x281aab [0x5b27
    Binary file "/opt/python/python", at +0x11b6e1 [0x5b27
    Binary file "/opt/python/python", at +0x11d348 [0x5b27
    Binary file "/opt/python/python", at +0x11d626 [0x5b27
    Binary file "/opt/python/python", at PyObject_Call+0x2
    Binary file "/opt/python/python", at +0x32a67a [0x5b27
    Binary file "/opt/python/python", at +0x32a7f8 [0x5b27
    Binary file "/opt/python/python", at +0x32ac1b [0x5b27
    Binary file "/opt/python/python", at Py_RunMain+0x31 [
    <truncated rest of calls>
  Segmentation fault
```

[1]

pdb — The Python Debugger

Código fuente: Lib/unittest/mock.py [https://github.com/python/cpython/tree/3.14/Lib/unittest/mock.py]

The module **pdb** defines an interactive source code debugger for Python programs. It supports setting (conditional) breakpoints and single stepping at the source line level, inspection of stack frames, source code listing, and evaluation of arbitrary Python code in the context of any stack frame. It also supports post-mortem debugging and can be called under program control.

El depurador es extensible – en realidad se define como la clase **Pdb**. Esto no está actualmente documentado pero se entiende fácilmente leyendo la fuente. La interfaz de extensión usa los módulos **bdb** y **cmd**.

Ver también

Module faulthandler
Used to dump Python tracebacks explicitly, on a fault, after a timeout, or on a user signal.

Module traceback
Standard interface to extract, format and print stack traces of Python programs.

The typical usage to break into the debugger is to insert:

```
import pdb; pdb.set_trace()
```

Or:

```
breakpoint()
```

at the location you want to break into the debugger, and then run

the program. You can then step through the code following this statement, and continue running without the debugger using the **`continue`** command.

Distinto en la versión 3.7: The built-in **`breakpoint()`**, when called with defaults, can be used instead of `import pdb; pdb.set_trace()`.

```
def double(x):
   breakpoint()
   return x * 2
val = 3
print(f"{val} * 2 is {double(val)}")
```

The debugger's prompt is `(Pdb)`, which is the indicator that you are in debug mode:

```
> ...(2)double()
-> breakpoint()
(Pdb) p x
3
(Pdb) continue
3 * 2 is 6
```

Distinto en la versión 3.3: La finalización de tabulación a través del módulo **`readline`** está disponible para comandos y argumentos de comando, por ejemplo. Los nombres globales y locales actuales se ofrecen como argumentos del comando `p`.

Command-line interface

You can also invoke **`pdb`** from the command line to debug other scripts. For example:

```
python -m pdb [-c command] (-m module | -p pid | pyfile)
```

When invoked as a module, pdb will automatically enter post-mortem debugging if the program being debugged exits abnormally. After post-mortem debugging (or after normal exit of the program), pdb will restart the program. Automatic restarting preserves pdb's state (such as breakpoints) and in most cases is more useful than quitting the debugger upon program's exit.

-c, --command <command>

> To execute commands as if given in a `.pdbrc` file; see Debugger commands.
>
> *Distinto en la versión 3.2:* Added the `-c` option.

-m <module>

> To execute modules similar to the way `python -m` does. As with a script, the debugger will pause execution just before the first line of the module.
>
> *Distinto en la versión 3.7:* Added the `-m` option.

-p, --pid <pid>

> Attach to the process with the specified PID.
>
> *Added in version 3.14.*

To attach to a running Python process for remote debugging, use the `-p` or `--pid` option with the target process's PID:

```
python -m pdb -p 1234
```

Nota

Attaching to a process that is blocked in a system call or waiting for I/O will only work once the next bytecode instruction is executed or when the process receives a signal.

Typical usage to execute a statement under control of the debugger is:

```
>>> import pdb
>>> def f(x):
...     print(1 / x)
>>> pdb.run("f(2)")
```

```
> <string>(1)<module>()
(Pdb) continue
0.5
>>>
```

El uso típico para inspeccionar un programa que se ha estrellado es:

```
>>> import pdb
>>> def f(x):
...     print(1 / x)
...
>>> f(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in f
ZeroDivisionError: division by zero
>>> pdb.pm()
> <stdin>(2)f()
(Pdb) p x
0
(Pdb)
```

Distinto en la versión 3.13: The implementation of **PEP 667** [https://peps.python.org/pep-0667/] means that name assignments made via `pdb` will immediately affect the active scope, even when running inside an optimized scope.

El módulo define las siguientes funciones; cada una de ellas entra en el depurador de una manera ligeramente diferente:

pdb.run(*statement*, *globals=None*, *locals=None*)

> Ejecutar el *statement* (dado como una cadena o un objeto de código) bajo el control del depurador. El prompt del depurador aparece antes de que se ejecute cualquier código; puedes establecer puntos de ruptura y escribir **`continue`**, o puedes pasar por la declaración usando **`step`** o **`next`** (todos estos comandos se explican más abajo). Los argumentos opcionales *globals* y *locals* especifican el entorno en el que se ejecuta el código; por defecto se utiliza el diccionario del módulo **`__main__`**. (Ver la explicación de las funciones

incorporadas `exec()` o `eval()`).

pdb.runeval(*expression*, *globals=None*, *locals=None*)

Evaluate the *expression* (given as a string or a code object) under debugger control. When **`runeval()`** returns, it returns the value of the *expression*. Otherwise this function is similar to `run()`.

pdb.runcall(*function*, **args*, ***kwds*)

Llama a la *function* (un objeto de función o método, no una cadena) con los argumentos dados. Cuando `runcall()` retorna, retorna lo que sea que la llamada de la función haya retornado. El aviso del depurador aparece tan pronto como se introduce la función.

pdb.set_trace(*, *header=None*, *commands=None*)

Enter the debugger at the calling stack frame. This is useful to hard-code a breakpoint at a given point in a program, even if the code is not otherwise being debugged (e.g. when an assertion fails). If given, *header* is printed to the console just before debugging begins. The *commands* argument, if given, is a list of commands to execute when the debugger starts.

Distinto en la versión 3.7: El argumento de la palabra clave *header*.

Distinto en la versión 3.13: **`set_trace()`** will enter the debugger immediately, rather than on the next line of code to be executed.

Added in version 3.14: The *commands* argument.

awaitable pdb.set_trace_async(*, *header=None*, *commands=None*)

async version of `set_trace()`. This function should be used inside an async function with `await`.

```
async def f():
    await pdb.set_trace_async()
```

`await` statements are supported if the debugger is invoked by this function.

Added in version 3.14.

pdb.post_mortem(*t=None*)

Enter post-mortem debugging of the given exception or traceback object. If no value is given, it uses the exception that is currently being handled, or raises `ValueError` if there isn't one.

Distinto en la versión 3.13: Support for exception objects was added.

pdb.pm()

Enter post-mortem debugging of the exception found in `sys.last_exc`.

pdb.set_default_backend(*backend*)

There are two supported backends for pdb: `'settrace'` and `'monitoring'`. See `bdb.Bdb` for details. The user can set the default backend to use if none is specified when instantiating `Pdb`. If no backend is specified, the default is `'settrace'`.

Nota

`breakpoint()` and `set_trace()` will not be affected by this function. They always use `'monitoring'` backend.

Added in version 3.14.

pdb.get_default_backend()

Returns the default backend for pdb.

Added in version 3.14.

Las funciones `run*` y **`set_trace()`** son alias para instanciar la clase **`Pdb`** y llamar al método del mismo nombre. Si quieres acceder a más funciones, tienes que hacerlo tú mismo:

class pdb.Pdb(*completekey='tab'*, *stdin=None*, *stdout=None*, *skip=None*, *nosigint=False*, *readrc=True*, *mode=None*, *backend=None*, *colorize=False*)

`Pdb` es la clase de depuración.

Los argumentos *completekey*, *stdin* y *stdout* se pasan a la subyacente **`cmd.Cmd`** class; ver la descripción allí.

El argumento *skip*, si se da, debe ser un iterable de los patrones de nombre de los módulos de estilo global. El depurador no entrará en marcos que se originen en un módulo que coincida con uno de estos patrones. [1]

By default, Pdb sets a handler for the SIGINT signal (which is sent when the user presses `Ctrl-C` on the console) when you give a **`continue`** command. This allows you to break into the debugger again by pressing `Ctrl-C`. If you want Pdb not to touch the SIGINT handler, set *nosigint* to true.

El argumento * readrc * por defecto es verdadero y controla si Pdb cargará archivos .pdbrc desde el sistema de archivos.

The *mode* argument specifies how the debugger was invoked. It impacts the workings of some debugger commands. Valid values are `'inline'` (used by the breakpoint() builtin), `'cli'` (used by the command line invocation) or `None` (for backwards compatible behaviour, as before the *mode* argument was added).

The *backend* argument specifies the backend to use for the debugger. If `None` is passed, the default backend will be used. See **`set_default_backend()`**. Otherwise the supported backends are `'settrace'` and `'monitoring'`.

The *colorize* argument, if set to `True`, will enable colorized output in the debugger, if color is supported. This will highlight source code displayed in pdb.

Ejemplo de llamada para permitir el rastreo con *skip*:

```
import pdb; pdb.Pdb(skip=['django.*']).set_trace()
```

Levanta una auditing event `pdb.Pdb` sin argumentos.

Distinto en la versión 3.1: Added the *skip* parameter.

Distinto en la versión 3.2: Added the *nosigint* parameter. Previously, a SIGINT handler was never set by Pdb.

Distinto en la versión 3.6: El argumento *readrc*.

Added in version 3.14: Added the *mode* argument.

Added in version 3.14: Added the *backend* argument.

Added in version 3.14: Added the *colorize* argument.

Distinto en la versión 3.14: Inline breakpoints like **`breakpoint()`** or **`pdb.set_trace()`** will always stop the program at calling frame, ignoring the *skip* pattern (if any).

run(*statement*, *globals=None*, *locals=None*)
runeval(*expression*, *globals=None*, *locals=None*)
runcall(*function*, **args*, ***kwds*)
set_trace()

Véase la documentación para las funciones explicadas anteriormente.

Debugger commands

Los comandos reconocidos por el depurador se enumeran a continuación. La mayoría de los comandos pueden abreviarse a una o dos letras como se indica; por ejemplo, `h(elp)` significa que se puede usar `h` o `help` para introducir el comando de ayuda (pero no `he` o `hel`, ni `H` o `Help` o `HELP`). Los argumentos de los comandos deben estar separados por espacios en blanco (espacios o tabulaciones). Los argumentos opcionales están encerrados entre corchetes (`[]`) en la sintaxis del comando; los corchetes no deben escribirse. Las alternativas en la sintaxis de los comandos están separadas por una barra vertical (`|`).

Introducir una línea en blanco repite el último comando introducido. Excepción: si el último comando fue un **`list`** comando, las siguientes 11 líneas están listadas.

Se supone que los comandos que el depurador no reconoce son declaraciones en Python y se ejecutan en el contexto del programa que se está depurando. Las declaraciones en Python también pueden ser precedidas por un signo de exclamación (`!`). Esta es una manera poderosa de inspeccionar el programa que se está depurando; incluso es posible cambiar una variable o llamar a una función. Cuando se produce una excepción en una sentencia de este tipo, se imprime el nombre de la excepción pero no se cambia el estado del depurador.

Distinto en la versión 3.13: Expressions/Statements whose prefix is a pdb command are now correctly identified and executed.

El depurador soporta aliases. Los alias pueden tener parámetros que permiten un cierto nivel de adaptabilidad al contexto que se está examinando.

Multiple commands may be entered on a single line, separated by `;;`. (A single `;` is not used as it is the separator for multiple commands in a line that is passed to the Python parser.) No intelligence is applied to separating the commands; the input is split at the first `;;` pair, even if it is in the middle of a quoted string. A workaround for strings with double semicolons is to use implicit string concatenation `';'';'` or `";"";"`.

To set a temporary global variable, use a *convenience variable*. A *convenience variable* is a variable whose name starts with `$`. For example, `$foo = 1` sets a global variable `$foo` which you can use in the debugger session. The *convenience variables* are cleared when the program resumes execution so it’s less likely to interfere with your program compared to using normal variables like `foo = 1`.

There are four preset *convenience variables*:

- `$_frame`: the current frame you are debugging
- `$_retval`: the return value if the frame is returning
- `$_exception`: the exception if the frame is raising an exception
- `$_asynctask`: the asyncio task if pdb stops in an async function

Added in version 3.12: Added the *convenience variable* feature.

Added in version 3.14: Added the `$_asynctask` convenience variable.

If a file `.pdbrc` exists in the user's home directory or in the current directory, it is read with `'utf-8'` encoding and executed as if it had been typed at the debugger prompt, with the exception that empty lines and lines starting with `#` are ignored. This is particularly useful for aliases. If both files exist, the one in the home directory is read first and aliases defined there can be overridden by the local file.

Distinto en la versión 3.2: `.pdbrc` puede ahora contener comandos que continúan depurando, como **`continue`** o **`next`**. Anteriormente, estos comandos no tenían ningún efecto.

Distinto en la versión 3.11: `.pdbrc` es ahora leído con codificación `'utf-8'`. Antes, era leído con la codificación correspondiente a la configuración regional del sistema.

h(elp) [command]

> Sin argumento, imprime la lista de comandos disponibles. Con un *command* como argumento, imprime la ayuda sobre ese comando. `help pdb` muestra la documentación completa (el docstring del módulo **`pdb`**). Como el argumento *command* debe ser un identificador, hay que introducir `help exec` para obtener ayuda sobre el comando `!`.

w(here) [count]

> Print a stack trace, with the most recent frame at the bottom. if *count* is 0, print the current frame entry. If *count* is negative, print the least recent - *count* frames. If *count* is positive, print

the most recent *count* frames. An arrow (>) indicates the current frame, which determines the context of most commands.

Distinto en la versión 3.14: count argument is added.

d(own) [count]

Mueve los niveles del marco actual *count* (por defecto uno) hacia abajo en el trazado de la pila (*stack trace*) (a un marco más nuevo).

u(p) [count]

Mueve el marco actual *count* (por defecto uno) niveles hacia arriba en el trazado de la pila (*stack trace*) (a un marco más antiguo).

b(reak) [([filename:]lineno | function) [, condition]]

With a *lineno* argument, set a break at line *lineno* in the current file. The line number may be prefixed with a *filename* and a colon, to specify a breakpoint in another file (possibly one that hasn't been loaded yet). The file is searched on **`sys.path`**. Acceptable forms of *filename* are `/abspath/to/file.py`, `relpath/file.py`, `module` and `package.module`.

With a *function* argument, set a break at the first executable statement within that function. *function* can be any expression that evaluates to a function in the current namespace.

Si un segundo argumento está presente, es una expresión que debe ser evaluada como verdadera antes de que el punto de ruptura sea honrado.

Sin argumento, enumere todas las interrupciones, incluyendo para cada punto de interrupción, el número de veces que se ha alcanzado ese punto de interrupción, el conteo de ignorancia actual y la condición asociada, si la hay.

Each breakpoint is assigned a number to which all the other breakpoint commands refer.

tbreak [([filename:]lineno | function) [, condition]]

Punto de interrupción temporal, que se elimina automáticamente cuando se ejecuta por primera vez. Los argumentos son los mismos que para `break`.

cl(ear) [filename:lineno | bpnumber ...]

Con el argumento *filename:lineno*, despeja todos los puntos de interrupción en esta línea. Con una lista de números de puntos de interrupción separados por espacios, despeja esos puntos de interrupción. Sin el argumento, despeja todos los puntos de interrupción (pero primero pide confirmación).

disable bpnumber [bpnumber ...]

Deshabilitar los puntos de ruptura interrupción como una lista de números de puntos de ruptura separados por espacios. Desactivar un punto de interrupción significa que no puede hacer que el programa detenga la ejecución, pero a diferencia de borrar un punto de interrupción, permanece en la lista de puntos de interrupción y puede ser (re)activado.

enable bpnumber [bpnumber ...]

Habilitar los puntos de interrupción especificados.

ignore bpnumber [count]

Set the ignore count for the given breakpoint number. If *count* is omitted, the ignore count is set to 0. A breakpoint becomes active when the ignore count is zero. When non-zero, the *count* is decremented each time the breakpoint is reached and the breakpoint is not disabled and any associated condition evaluates to true.

condition bpnumber [condition]

Establece una nueva *condition* para el punto de interrupción, una expresión que debe evaluarse como verdadera antes de que el punto de ruptura sea honrado. Si la condición está ausente, se elimina cualquier condición existente, es decir, el punto de ruptura se hace incondicional.

commands [bpnumber]

Especifique una lista de comandos para el número del punto de interrupción *bpnumber*. Los comandos mismos aparecen en las siguientes líneas. Escriba una línea que contenga sólo `end` para terminar los comandos. Un ejemplo:

```
(Pdb) commands 1
(com) p some_variable
(com) end
(Pdb)
```

Para eliminar todos los comandos de un punto de interrupción, escribe `commands` y sigue inmediatamente con `end` , es decir, no des órdenes.

Sin el argumento *bpnumber*, `commands` se refiere al último punto de interrupción establecido.

Puede utilizar los comandos de punto de interrupción para iniciar el programa de nuevo. Simplemente usa el comando **`continue`**, o **`step`**, o cualquier otro comando que reanude la ejecución.

Specifying any command resuming execution (currently **`continue`**, **`step`**, **`next`**, **`return`**, **`until`**, **`jump`**, **`quit`** and their abbreviations) terminates the command list (as if that command was immediately followed by end). This is because any time you resume execution (even with a simple next or step), you may encounter another breakpoint—which could have its own command list, leading to ambiguities about which list to execute.

If the list of commands contains the `silent` command, or a command that resumes execution, then the breakpoint message containing information about the frame is not displayed.

Distinto en la versión 3.14: Frame information will not be displayed if a command that resumes execution is present in the command list.

s(tep)

Ejecutar la línea actual, detenerse en la primera ocasión posible (ya sea en una función que se llame o en la siguiente línea de la función actual).

n(ext)

Continúe la ejecución hasta que se alcance la siguiente línea de la función actual o vuelva. (La diferencia entre `next` y `step` es que `step` se detiene dentro de una función llamada, mientras que `next` ejecuta las funciones llamadas a (casi) toda velocidad, deteniéndose sólo en la siguiente línea de la función actual).

unt(il) [lineno]

Sin argumento, continúe la ejecución hasta que se alcance la línea con un número mayor que el actual.

With *lineno*, continue execution until a line with a number greater or equal to *lineno* is reached. In both cases, also stop when the current frame returns.

Distinto en la versión 3.2: Permita dar un número de línea explícito.

r(eturn)

Continúe la ejecución hasta que vuelva la función actual.

c(ont(inue))

Continúa la ejecución, sólo se detiene cuando se encuentra un punto de ruptura.

j(ump) lineno

Establezca la siguiente línea que será ejecutada. Sólo disponible en el marco de más bajo. Esto te permite saltar hacia atrás y ejecutar el código de nuevo, o saltar hacia adelante para saltar el código que no quieres ejecutar.

Cabe señalar que no todos los saltos están permitidos – por ejemplo, no es posible saltar en medio de un bucle `for` o fuera de una cláusula `finally`.

l(ist) [first[, last]]

Enumere el código fuente del archivo actual. Sin argumentos, enumere 11 líneas alrededor de la línea actual o continúe la lista anterior. Con `.` como argumento, enumere 11 líneas alrededor de la línea actual. Con un argumento, enumere 11 líneas alrededor de esa línea. Con dos argumentos, enumere el rango dado; si el segundo argumento es menor que el primero, se interpreta como un conteo.

La línea actual en el cuadro actual se indica con `->`. Si se está depurando una excepción, la línea donde la excepción fue originalmente planteada o propagada se indica con `>>`, si difiere de la línea actual.

Distinto en la versión 3.2: Added the `>>` marker.

ll | longlist

Enumere todos los códigos fuente de la función o marco actual. Las líneas interesantes están marcadas como `list`.

Added in version 3.2.

a(rgs)

Print the arguments of the current function and their current values.

p expression

Evaluate *expression* in the current context and print its value.

Nota

`print()` también se puede usar, pero no es un comando de depuración — esto ejecuta la función Python `print()`.

pp expression

Like the `p` command, except the value of *expression* is pretty-printed using the `pprint` module.

whatis expression

Print the type of *expression*.

source expression

Try to get source code of *expression* and display it.

Added in version 3.2.

display [expression]

Display the value of *expression* if it changed, each time execution stops in the current frame.

Without *expression*, list all display expressions for the current frame.

Nota

Display evaluates *expression* and compares to the result of the previous evaluation of *expression*, so when the result is mutable, display may not be able to pick up the changes.

Example:

```
lst = []
breakpoint()
pass
lst.append(1)
print(lst)
```

Display won't realize `lst` has been changed because the result of evaluation is modified in place by `lst.append(1)` before being compared:

```
> example.py(3)<module>()
-> pass
(Pdb) display lst
```

```
display lst: []
(Pdb) n
> example.py(4)<module>()
-> lst.append(1)
(Pdb) n
> example.py(5)<module>()
-> print(lst)
(Pdb)
```

You can do some tricks with copy mechanism to make it work:

```
> example.py(3)<module>()
-> pass
(Pdb) display lst[:]
display lst[:]: []
(Pdb) n
> example.py(4)<module>()
-> lst.append(1)
(Pdb) n
> example.py(5)<module>()
-> print(lst)
display lst[:]: [1]  [old: []]
(Pdb)
```

Added in version 3.2.

undisplay [expression]

Do not display *expression* anymore in the current frame. Without *expression*, clear all display expressions for the current frame.

Added in version 3.2.

interact

Start an interactive interpreter (using the `code` module) in a new global namespace initialised from the local and global namespaces for the current scope. Use `exit()` or `quit()` to exit the interpreter and return to the debugger.

Nota

As `interact` creates a new dedicated namespace for code execution, assignments to variables will not affect the original namespaces. However, modifications to any referenced mutable objects will be reflected in the original namespaces as usual.

Added in version 3.2.

Distinto en la versión 3.13: `exit()` and `quit()` can be used to exit the **`interact`** command.

Distinto en la versión 3.13: **`interact`** directs its output to the debugger's output channel rather than **`sys.stderr`**.

alias [name [command]]

Create an alias called *name* that executes *command*. The *command* must *not* be enclosed in quotes. Replaceable parameters can be indicated by `%1`, `%2`, … and `%9`, while `%*` is replaced by all the parameters. If *command* is omitted, the current alias for *name* is shown. If no arguments are given, all aliases are listed.

Los alias pueden anidarse y pueden contener cualquier cosa que se pueda teclear legalmente en el prompt de pdb. Tenga en cuenta que los comandos internos de pdb *pueden* ser anulados por los alias. Dicho comando se oculta hasta que se elimina el alias. El alias se aplica de forma recursiva a la primera palabra de la línea de comandos; todas las demás palabras de la línea se dejan en paz.

Como ejemplo, aquí hay dos alias útiles (especialmente cuando se colocan en el archivo `.pdbrc`):

```
# Print instance variables (usage "pi classInst")
alias pi for k in %1.__dict__.keys(): print(f"%1.{k
# Print instance variables in self
alias ps pi self
```

unalias name

Delete the specified alias *name*.

! statement

Execute the (one-line) *statement* in the context of the current stack frame. The exclamation point can be omitted unless the first word of the statement resembles a debugger command, e.g.:

```
(Pdb) ! n=42
(Pdb)
```

To set a global variable, you can prefix the assignment command with a **`global`** statement on the same line, e.g.:

```
(Pdb) global list_options; list_options = ['-l']
(Pdb)
```

run [args ...]
restart [args ...]

Restart the debugged Python program. If *args* is supplied, it is split with **`shlex`** and the result is used as the new **`sys.argv`**. History, breakpoints, actions and debugger options are preserved. **`restart`** is an alias for **`run`**.

Distinto en la versión 3.14: **`run`** and **`restart`** commands are disabled when the debugger is invoked in `'inline'` mode.

q(uit)

Quit from the debugger. The program being executed is aborted. An end-of-file input is equivalent to **`quit`**.

A confirmation prompt will be shown if the debugger is invoked in `'inline'` mode. Either `y`, `Y`, `<Enter>` or `EOF` will confirm the quit.

Distinto en la versión 3.14: A confirmation prompt will be shown if the debugger is invoked in `'inline'` mode. After the confirmation, the debugger will call **`sys.exit()`**

immediately, instead of raising **`bdb.BdbQuit`** in the next trace event.

debug code

Enter a recursive debugger that steps through *code* (which is an arbitrary expression or statement to be executed in the current environment).

retval

Print the return value for the last return of the current function.

exceptions [excnumber]

List or jump between chained exceptions.

When using `pdb.pm()` or `Pdb.post_mortem(...)` with a chained exception instead of a traceback, it allows the user to move between the chained exceptions using `exceptions` command to list exceptions, and `exceptions <number>` to switch to that exception.

Example:

```
def out():
    try:
        middle()
    except Exception as e:
        raise ValueError("reraise middle() error")

def middle():
    try:
        return inner(0)
    except Exception as e:
        raise ValueError("Middle fail")

def inner(x):
    1 / x

 out()
```

calling `pdb.pm()` will allow to move between exceptions:

```
> example.py(5)out()
-> raise ValueError("reraise middle() error") from

(Pdb) exceptions
  0 ZeroDivisionError('division by zero')
  1 ValueError('Middle fail')
> 2 ValueError('reraise middle() error')

(Pdb) exceptions 0
> example.py(16)inner()
-> 1 / x

(Pdb) up
> example.py(10)middle()
-> return inner(0)
```

Added in version 3.13.

Notas de pie de página

Si se considera que un marco se origina en un determinado módulo se determina por el `__name__` en el marcos globales.

Los perfiladores de Python

Código fuente: Lib/profile.py [https://github.com/python/cpython/tree/3.14/Lib/profile.py] y Lib/pstats.py [https://github.com/python/cpython/tree/3.14/Lib/pstats.py]

Introducción a los perfiladores

`cProfile` y `profile` proporcionan *deterministic profiling* de los programas de Python. *profile* es un conjunto de estadísticas que describe con qué frecuencia y durante cuánto tiempo se ejecutaron varias partes del programa. Estas estadísticas pueden formatearse en informes a través del módulo `pstats`.

La biblioteca estándar de Python proporciona dos implementaciones diferentes de la misma interfaz de creación de perfiles:

1. `cProfile` se recomienda para la mayoría de los usuarios; Es una extensión C con una sobrecarga razonable que la hace adecuada para perfilar programas de larga duración. Basado en `lsprof`, aportado por Brett Rosen y Ted Czotter.
2. `profile`, un módulo Python puro cuya interfaz es imitada por `cProfile`, pero que agrega una sobrecarga significativa a los programas perfilados. Si está intentando extender el generador de perfiles de alguna manera, la tarea podría ser más fácil con este módulo. Originalmente diseñado y escrito por Jim Roskind.

Nota

Los módulos del generador de perfiles están diseñados para proporcionar un perfil de ejecución para un programa determinado, no para fines de evaluación comparativa (para eso, está `timeit` que permite obtener resultados razonablemente precisos). Esto se aplica particularmente a la evaluación comparativa del código Python contra el código C: los perfiladores introducen gastos generales para el código Python, pero no para las funciones de nivel C, por lo que el código C parecería más rápido que cualquier Python.

Manual instantáneo de usuario

Esta sección se proporciona a los usuarios que «no quieren leer el manual». Proporciona una visión general muy breve y permite a un usuario realizar perfiles rápidamente en una aplicación existente.

Para perfilar una función que toma un solo argumento, puede hacer:

```
import cProfile
import re
cProfile.run('re.compile("foo|bar")')
```

(Use **`profile`** en lugar de **`cProfile`** si este último no está disponible en su sistema.)

La acción anterior ejecutaría **`re.compile()`** e imprime resultados de perfil como los siguientes:

```
      214 function calls (207 primitive calls) in 0.002

Ordered by: cumulative time

ncalls  tottime  percall  cumtime  percall filename:line
     1    0.000    0.000    0.002    0.002 {built-in met
     1    0.000    0.000    0.001    0.001 <string>:1(<m
     1    0.000    0.000    0.001    0.001 __init__.py:2
     1    0.000    0.000    0.001    0.001 __init__.py:2
     1    0.000    0.000    0.000    0.000 _compiler.py:
     1    0.000    0.000    0.000    0.000 _parser.py:93
     1    0.000    0.000    0.000    0.000 _compiler.py:
     1    0.000    0.000    0.000    0.000 _parser.py:43
```

The first line indicates that 214 calls were monitored. Of those calls, 207 were *primitive*, meaning that the call was not induced via recursion. The next line: `Ordered by: cumulative time` indicates the output is sorted by the `cumtime` values. The column headings include:

ncalls
: por el número de llamadas.

tottime
: para el tiempo total empleado en la función dada (y excluyendo el tiempo realizado en llamadas a subfunciones)

percall
: es el cociente de `tottime` dividido por `ncalls`

cumtime
: es el tiempo acumulado empleado en esta y todas las subfunciones (desde la invocación hasta la salida). Esta cifra es precisa *incluso* para funciones recursivas.

percall
: es el cociente de `cumtime` dividido por llamadas primitivas

filename:lineno(function)
: proporciona los datos respectivos de cada función

Cuando hay dos números en la primera columna (por ejemplo, `3/1`), significa que la función se repite. El segundo valor es el número de llamadas primitivas y el primero es el número total de llamadas. Tenga en cuenta que cuando la función no se repite, estos dos valores son iguales y solo se imprime la figura.

En lugar de imprimir la salida al final de la ejecución del perfil, puede guardar los resultados en un archivo especificando un nombre de archivo en la función **`run()`**:

```
import cProfile
import re
cProfile.run('re.compile("foo|bar")', 'restats')
```

La clase **`pstats.Stats`** lee los resultados del perfil desde un archivo y los formatea de varias maneras.

Los archivos `cProfile` y `profile` también se pueden invocar como un script para perfilar otro script. Por ejemplo:

```
python -m cProfile [-o output_file] [-s sort_order] (-m
```

-o <output_file>

Writes the profile results to a file instead of to stdout.

-s <sort_order>

Specifies one of the `sort_stats()` sort values to sort the output by. This only applies when `-o` is not supplied.

-m <module>

Specifies that a module is being profiled instead of a script.

Added in version 3.7: Se agregó la opción `-m` a `cProfile`.

Added in version 3.8: Se agregó la opción `-m` a `profile`.

Los módulos `pstats` clase `Stats` tienen una variedad de métodos para manipular e imprimir los datos guardados en un archivo de resultados de perfil:

```
import pstats
from pstats import SortKey
p = pstats.Stats('restats')
p.strip_dirs().sort_stats(-1).print_stats()
```

El método `strip_dirs()` eliminó la ruta extraña de todos los nombres de módulos. El método `sort_stats()` clasificó todas las entradas de acuerdo con el módulo/línea/nombre de cadena de caracteres estándar que se imprime. El método `print_stats()` imprimió todas las estadísticas. Puede intentar las siguientes llamadas de clasificación:

```
p.sort_stats(SortKey.NAME)
p.print_stats()
```

La primera llamada realmente ordenará la lista por nombre de función, y la segunda llamada imprimirá las estadísticas. Las siguientes son algunas llamadas interesantes para experimentar:

```
p.sort_stats(SortKey.CUMULATIVE).print_stats(10)
```

Esto ordena el perfil por tiempo acumulado en una función y luego solo imprime las diez líneas más significativas. Si desea comprender qué algoritmos están tomando tiempo, la línea anterior es lo que usaría.

Si estuviera buscando ver qué funciones se repetían mucho y toman mucho tiempo, usted haría:

```
p.sort_stats(SortKey.TIME).print_stats(10)
```

para ordenar según el tiempo dedicado a cada función y luego imprimir las estadísticas de las diez funciones principales.

También puedes probar:

```
p.sort_stats(SortKey.FILENAME).print_stats('__init__')
```

Esto ordenará todas las estadísticas por nombre de archivo y luego imprimirá estadísticas solo para los métodos de inicio de clase (ya que están escritos con `__init__` en ellos). Como último ejemplo, puedes probar:

```
p.sort_stats(SortKey.TIME, SortKey.CUMULATIVE).print_sta
```

Esta línea clasifica las estadísticas con una clave primaria de tiempo y una clave secundaria de tiempo acumulado, y luego imprime algunas de las estadísticas. Para ser específicos, la lista primero se reduce al 50% (re: `.5`) de su tamaño original, luego solo se mantienen las líneas que contienen `` `init `` y se imprime esa sub-sublista.

Si se preguntó qué funciones denominaban las funciones anteriores, ahora podría (`p` todavía está ordenada según el último criterio) hacer:

```
p.print_callers(.5, 'init')
```

y obtendría una lista de llamadas para cada una de las funciones enumeradas.

Si desea más funcionalidad, tendrá que leer el manual o adivinar lo que hacen las siguientes funciones:

```
p.print_callees()
p.add('restats')
```

Invocado como un script, el módulo `pstats` es un navegador de estadísticas para leer y examinar volcados de perfil. Tiene una interfaz simple orientada a la línea de comandos (implementada usando `cmd`) y ayuda interactiva.

`profile` and `cProfile` Module Reference

Both the `profile` and `cProfile` modules provide the following functions:

profile.run(*command*, *filename=None*, *sort=-1*)

Esta función toma un único argumento que se puede pasar a la función `exec()` y un nombre de archivo opcional. En todos los casos esta rutina ejecuta:

```
exec(command, __main__.__dict__, __main__.__dict__)
```

y recopila estadísticas de perfiles de la ejecución. Si no hay ningún nombre de archivo, esta función crea automáticamente una instancia de `Stats` e imprime un informe de perfil simple. Si se especifica el valor de clasificación, se pasa a esta instancia `Stats` para controlar cómo se ordenan los resultados.

profile.runctx(*command*, *globals*, *locals*, *filename=None*, *sort=-1*)

This function is similar to `run()`, with added arguments to supply the globals and locals mappings for the *command* string. This routine executes:

```
exec(command, globals, locals)
```

y recopila estadísticas de perfiles como en la función **`run()`** anterior.

class profile.Profile(*timer=None*, *timeunit=0.0*, *subcalls=True*, *builtins=True*)

Esta clase normalmente solo es usada si se necesita un control más preciso sobre la creación de perfiles que el que proporciona la función **`cProfile.run()`**.

Se puede suministrar un temporizador personalizado para medir cuánto tiempo tarda el código en ejecutarse mediante el argumento *timer*. Esta debe ser una función que retorna un solo número que representa la hora actual. Si el número es un entero, la *timeunit* especifica un multiplicador que especifica la duración de cada unidad de tiempo. Por ejemplo, si el temporizador retorna tiempos medidos en miles de segundos, la unidad de tiempo sería `.001`.

El uso directo de la clase **`Profile`** permite formatear los resultados del perfil sin escribir los datos del perfil en un archivo:

```
import cProfile, pstats, io
from pstats import SortKey
pr = cProfile.Profile()
pr.enable()
# ... do something ...
pr.disable()
s = io.StringIO()
sortby = SortKey.CUMULATIVE
ps = pstats.Stats(pr, stream=s).sort_stats(sortby)
ps.print_stats()
print(s.getvalue())
```

The **`Profile`** class can also be used as a context manager (supported only in **`cProfile`** module. see Tipos gestores de contexto):

```
import cProfile

with cProfile.Profile() as pr:
    # ... do something ...

    pr.print_stats()
```

Distinto en la versión 3.8: Se agregó soporte de administrador de contexto.

enable()

Start collecting profiling data. Only in **`cProfile`**.

disable()

Stop collecting profiling data. Only in **`cProfile`**.

create_stats()

Deja de recopilar datos de perfiles y registre los resultados internamente como el perfil actual.

print_stats(*sort=-1*)

Crea un objeto `Stats` en función del perfil actual e imprima los resultados en *stdout*.

The *sort* parameter specifies the sorting order of the displayed statistics. It accepts a single key or a tuple of keys to enable multi-level sorting, as in `Stats.sort_stats`.

Added in version 3.13: **`print_stats()`** now accepts a tuple of keys.

dump_stats(*filename*)

Escribe los resultados del perfil actual en *filename*.

run(*cmd*)

Perfila el cmd a través de `exec()`.

runctx(*cmd, globals, locals*)

Perfila el cmd a través de `exec()` con el entorno global y local especificado.

runcall(*func, /, *args, **kwargs*)

Perfila `func(*args, **kwargs)`

Tenga en cuenta que la creación de perfiles solo funcionará si el comando/función llamado realmente regresa. Si el intérprete se termina (por ejemplo, a través de una llamada a `sys.exit()` durante la ejecución del comando/función llamado) no se imprimirán resultados de generación de perfiles.

La clase `Stats`

El análisis de los datos del generador de perfiles es realizado utilizando la clase `Stats`.

class pstats.Stats(**filenames or profile, stream=sys.stdout*)

Este constructor de clase crea una instancia de un «objeto de estadísticas» a partir de un *filename* (o una lista de nombres de archivo) o de una instancia **`Profile`**. La salida se imprimirá en la secuencia especificada por *stream*.

El archivo seleccionado por el constructor anterior debe haber sido creado por la versión correspondiente de `profile` o `cProfile`. Para ser específicos, *no* hay compatibilidad de archivos garantizada con futuras versiones de este generador de perfiles, y no hay compatibilidad con archivos producidos por otros perfiladores, o el mismo generador de perfiles ejecutado en un sistema operativo diferente. Si se proporcionan varios archivos, se combinarán todas las estadísticas para funciones idénticas, de modo que se pueda considerar una vista general de varios procesos en un solo informe. Si es necesario combinar archivos adicionales con datos en un objeto existente `Stats`, se puede usar el método `add()`.

En lugar de leer los datos del perfil de un archivo, se puede usar un objeto `cProfile.Profile` o `profile.Profile` como fuente de datos del perfil.

los objetos `Stats` tienen los siguientes métodos:

strip_dirs()

> Este método para la clase `Stats` elimina toda la información de ruta principal de los nombres de archivo. Es muy útil para reducir el tamaño de la impresión para que quepa en (cerca de) 80 columnas. Este método modifica el objeto y se pierde la información eliminada. Después de realizar una operación de tira, se considera que el objeto tiene sus entradas en un orden «aleatorio», como lo fue justo después de la inicialización y carga del objeto. Si `strip_dirs()` hace que dos nombres de funciones no se puedan distinguir (están en la misma línea del mismo nombre de archivo y tienen el mismo nombre de función), entonces las estadísticas para estas dos entradas se acumulan en un sola entrada.

add(**filenames*)

> Este método de la clase `Stats` acumula información de perfil adicional en el objeto de perfil actual. Sus argumentos deben referirse a los nombres de archivo creados por la versión correspondiente de `profile.run()` or `cProfile.run()`. Las estadísticas para funciones con nombre idéntico (re: archivo, línea, nombre) se acumulan automáticamente en estadísticas de función única.

dump_stats(*filename*)

> Guarda los datos cargados en el objeto `Stats` en un archivo llamado *filename*. El archivo se crea si no existe y se sobrescribe si ya existe. Esto es equivalente al método del mismo nombre en las clases: class `profile.Profile` y `cProfile.Profile`.

sort_stats(**keys*)

Este método modifica el objeto `Stats` ordenándolo de acuerdo con los criterios proporcionados. El argumento puede ser una cadena de caracteres o una enumeración SortKey que identifica la base de una clasificación (ejemplo: `'time'`, `'name'`, `SortKey.TIME` o `SortKey.NAME`). El argumento enumeraciones SortKey tiene ventaja sobre el argumento de cadena de caracteres en que es más robusto y menos propenso a errores.

Cuando se proporciona más de una llave, se utilizan llaves adicionales como criterios secundarios cuando hay igualdad en todas las llaves seleccionadas antes que ellas. Por ejemplo, `sort_stats(SortKey.NAME, SortKey.FILE)` ordenará todas las entradas de acuerdo con el nombre de su función y resolverá todos los vínculos (nombres de función idénticos) clasificándolos por nombre de archivo.

Para el argumento de cadena de caracteres, se pueden usar abreviaturas para cualquier nombre de llaves, siempre que la abreviatura no sea ambigua.

Los siguientes son la cadena de caracteres válida y SortKey:

Arg de cadena de caracteres válido	Enumeración válida	Significado
`'calls'`	SortKey.CALLS	recuento de llamadas
`'cumulative'`	SortKey.CUMULATIVE	tiempo acumulado
`'cumtime'`	N/A	tiempo acumulado
`'file'`	N/A	nombre de archivo
`'filename'`	SortKey.FILENAME	nombre de archivo
`'module'`	N/A	nombre de archivo
`'ncalls'`	N/A	recuento de llamadas
`'pcalls'`	SortKey.PCALLS	recuento de llamadas primitivas
`'line'`	SortKey.LINE	número de línea
`'name'`	SortKey.NAME	nombre de la función
`'nfl'`	SortKey.NFL	nombre/archivo/línea
`'stdname'`	SortKey.STDNAME	nombre estándar

`'time'`	`SortKey.TIME`	tiempo interno
`'tottime'`	N/A	tiempo interno

Tenga en cuenta que todos los tipos de estadísticas están en orden descendente (colocando primero los elementos que requieren más tiempo), donde las búsquedas de nombre, archivo y número de línea están en orden ascendente (alfabético). La sutil distinción entre `SortKey.NFL` y `SortKey.STDNAME` es que el nombre estándar es una especie de nombre tal como está impreso, lo que significa que los números de línea incrustados se comparan de manera extraña. Por ejemplo, las líneas 3, 20 y 40 aparecerían (si los nombres de los archivos fueran los mismos) en el orden de las cadenas 20, 3 y 40. En contraste, `SortKey.NFL` hace una comparación numérica de los números de línea. De hecho, `sort_stats(SortKey.NFL)` es lo mismo que `sort_stats(SortKey.NAME, SortKey.FILENAME, SortKey.LINE)`.

Por razones de compatibilidad con versiones anteriores, se permiten los argumentos numéricos `-1`, `0`, `1`, y `2`. Se interpretan como `'stdname'`, `'calls'`, `'time'`, y `'cumulative'` respectivamente. Si se usa este formato de estilo antiguo (numérico), solo se usará una clave de clasificación (la tecla numérica) y los argumentos adicionales se ignorarán en silencio.

Added in version 3.7: Enumeración SortKey añadida.

reverse_order()

Este método para la clase **`Stats`** invierte el orden de la lista básica dentro del objeto. Tenga en cuenta que, de forma predeterminada, el orden ascendente vs descendente se selecciona correctamente en función de la llave de clasificación elegida.

print_stats(**restrictions*)

Este método para la clase **`Stats`** imprime un informe como se describe en la definición `profile.run()`.

El orden de la impresión se basa en la última operación `sort_stats()` realizada en el objeto (sujeto a advertencias en `add()` y `strip_dirs()`).

The arguments provided (if any) can be used to limit the list down to the significant entries. Initially, the list is taken to be the complete set of profiled functions. Each restriction is either an integer (to select a count of lines), or a decimal fraction between 0.0 and 1.0 inclusive (to select a percentage of lines), or a string that will be interpreted as a regular expression (to pattern match the standard name that is printed). If several restrictions are provided, then they are applied sequentially. For example:

```
print_stats(.1, 'foo:')
```

primero limitaría la impresión al primer 10% de la lista, y luego solo imprimiría las funciones que formaban parte del nombre del archivo `.*foo:`. En contraste, el comando:

```
print_stats('foo:', .1)
```

limitaría la lista a todas las funciones que tengan nombres de archivo: archivo `.*foo:`, y luego procedería a imprimir solo el primer 10% de ellas.

print_callers(**restrictions*)

Este método para la clase `Stats` imprime una lista de todas las funciones que llamaron a cada función en la base de datos perfilada. El orden es idéntico al proporcionado por `print_stats()`, y la definición del argumento de restricción también es idéntica. Cada persona que llama se informa en su propia línea. El formato difiere ligeramente según el generador de perfiles que produjo las estadísticas:

- Con `profile`, se muestra un número entre paréntesis después de cada llamada para mostrar cuántas veces se realizó esta llamada específica. Por conveniencia, un segundo número sin paréntesis repite el tiempo acumulado empleado en la función de la derecha.
- Con `cProfile`, cada persona que llama está precedida por tres números: la cantidad de veces que se realizó esta llamada específica y los tiempos totales y acumulativos gastados en la función actual mientras fue invocada por esta persona que llama.

print_callees(**restrictions*)

Este método para la clase `Stats` imprime una lista de todas las funciones que fueron llamadas por la función indicada. Aparte de esta inversión de la dirección de las llamadas (re: llamado vs fue llamado por), los argumentos y el orden son idénticos al método `print_callers()`.

get_stats_profile()

Este método retorna una instancia de StatsProfile, la cual contiene un mapeo de nombres de función a instancias de FunctionProfile. Cada instancia de FunctionProfile mantiene información relacionada al perfil de la función, como cuánto demoró la función en ejecutarse, cuantas veces fue llamada, etc…

Added in version 3.9: Añadidas las siguientes clases de datos: StatsProfile, FunctionProfile. Añadida la siguiente función: get_stats_profile.

¿Qué es el perfil determinista?

Deterministic profiling está destinada a reflejar el hecho de que se supervisan todos los eventos *function call*, *function return*, y *exception*, y se realizan temporizaciones precisas para los intervalos entre estos eventos (durante los cuales el código del usuario se está ejecutando). Por el contrario, *statistical profiling* (que no se realiza en este módulo) muestrea aleatoriamente el puntero de instrucción efectivo y deduce dónde se está gastando el tiempo. La última técnica tradicionalmente implica menos sobrecarga (ya que el código no necesita ser instrumentado), pero proporciona solo indicaciones relativas de dónde se está gastando el tiempo.

En Python, dado que hay un intérprete activo durante la ejecución, no se requiere la presencia de código instrumentado para realizar un perfil determinista. Python proporciona automáticamente un *hook* (retrollamada opcional) para cada evento. Además, la naturaleza interpretada de Python tiende a agregar tanta sobrecarga a la ejecución, que los perfiles deterministas tienden a agregar solo una pequeña sobrecarga de procesamiento en aplicaciones típicas. El resultado es que el perfil determinista no es tan costoso, pero proporciona estadísticas extensas de tiempo de ejecución sobre la ejecución de un programa Python.

Las estadísticas de recuento de llamadas se pueden usar para identificar errores en el código (recuentos sorprendentes) e identificar posibles puntos de expansión en línea (recuentos altos de llamadas). Las estadísticas de tiempo interno se pueden utilizar para identificar «bucles activos» que deben optimizarse cuidadosamente. Las estadísticas de tiempo acumulativo deben usarse para identificar errores de alto nivel en la selección de algoritmos. Tenga en cuenta que el manejo inusual de los tiempos acumulativos en este generador de perfiles permite que las estadísticas de implementaciones recursivas de algoritmos se comparen directamente con implementaciones iterativas.

Limitaciones

Una limitación tiene que ver con la precisión de la información de sincronización. Hay un problema fundamental con los perfiladores deterministas que implican precisión. La restricción más obvia es que el «reloj» subyacente solo funciona a una velocidad (típicamente) de aproximadamente 0,001 segundos. Por lo tanto, ninguna medición será más precisa que el reloj subyacente. Si se toman suficientes medidas, entonces el «error» tenderá a promediar.

Desafortunadamente, eliminar este primer error induce una segunda fuente de error.

El segundo problema es que «lleva un tiempo» desde que se distribuye un evento hasta que la llamada del generador de perfiles para obtener la hora en realidad *obtiene* el estado del reloj. Del mismo modo, hay un cierto retraso al salir del controlador de eventos del generador de perfiles desde el momento en que se obtuvo el valor del reloj (y luego se retiró), hasta que el código del usuario se está ejecutando nuevamente. Como resultado, las funciones que se llaman muchas veces, o llaman a muchas funciones, generalmente acumularán este error. El error que se acumula de esta manera es típicamente menor que la precisión del reloj (menos de un tic de reloj), pero *puede* acumularse y volverse muy significativo.

El problema es más importante con `profile` que con la sobrecarga inferior `cProfile`. Por esta razón, **`profile`** proporciona un medio para calibrarse a sí mismo para una plataforma dada para que este error pueda ser eliminado probabilísticamente (en promedio). Después de calibrar el generador de perfiles, será más preciso (en un sentido al menos cuadrado), pero a veces producirá números negativos (cuando el recuento de llamadas es excepcionalmente bajo y los dioses de la probabilidad trabajan en su contra :-)) *No* se alarme por los números negativos en el perfil. Deberían aparecer *solo* si ha calibrado su generador de perfiles, y los resultados son realmente mejores que sin calibración.

Calibración

El generador de perfiles del módulo `profile` resta una constante de cada tiempo de manejo de eventos para compensar la sobrecarga de llamar a la función de tiempo y eliminar los resultados. Por defecto, la constante es 0. El siguiente procedimiento se puede usar para obtener una mejor constante para una plataforma dada (ver Limitaciones).

```
import profile
pr = profile.Profile()
```

```
for i in range(5):
    print(pr.calibrate(10000))
```

El método ejecuta la cantidad de llamadas de Python dadas por el argumento, directamente y nuevamente bajo el generador de perfiles, midiendo el tiempo para ambos. Luego, calcula la sobrecarga oculta por evento del generador de perfiles y la retorna como un valor flotante. Por ejemplo, en un Intel Core i5 de 1.8Ghz que ejecuta macOS y usa el time.process_time () de Python como temporizador, el número mágico es aproximadamente 4.04e-6.

El objetivo de este ejercicio es obtener un resultado bastante consistente. Si su computadora es *muy* rápida, o su función de temporizador tiene una resolución pobre, es posible que tenga que pasar 100000, o incluso 1000000, para obtener resultados consistentes.

Cuando tiene una respuesta consistente, hay tres formas de usarla:

```
import profile

# 1. Apply computed bias to all Profile instances create
profile.Profile.bias = your_computed_bias

# 2. Apply computed bias to a specific Profile instance.
pr = profile.Profile()
pr.bias = your_computed_bias

# 3. Specify computed bias in instance constructor.
pr = profile.Profile(bias=your_computed_bias)
```

Si tiene una opción, es mejor que elija una constante más pequeña, y luego sus resultados «con menos frecuencia» se mostrarán como negativos en las estadísticas del perfil.

Usando un temporizador personalizado

Si desea cambiar la forma en que se determina el tiempo actual (por ejemplo, para forzar el uso del tiempo del reloj de pared o el tiempo transcurrido del proceso), pase la función de temporización que desee a la clase constructora **`Profile`**:

```
pr = profile.Profile(your_time_func)
```

El generador de perfiles resultante llamará a `your_time_func`. Dependiendo de si está utilizando **`profile.Profile`** o **`cProfile.Profile`**, el valor de retorno de `your_time_func` se interpretará de manera diferente:

`profile.Profile`

> `your_time_func` debería retornar un solo número o una lista de números cuya suma es la hora actual (como lo que **`os.times()`** retorna). Si la función retorna un número de tiempo único o la lista de números retornados tiene una longitud de 2, obtendrá una versión especialmente rápida de la rutina de envío.
>
> Be warned that you should calibrate the profiler class for the timer function that you choose (see Calibración). For most machines, a timer that returns a lone integer value will provide the best results in terms of low overhead during profiling. (**`os.times()`** is *pretty* bad, as it returns a tuple of floating-point values). If you want to substitute a better timer in the cleanest fashion, derive a class and hardwire a replacement dispatch method that best handles your timer call, along with the appropriate calibration constant.

`cProfile.Profile`

> `your_time_func` debería retornar un solo número. Si retorna enteros, también puede invocar al constructor de la clase con un segundo argumento que especifique la duración real de una unidad de tiempo. Por ejemplo, si `your_integer_time_func` retorna tiempos medidos en miles de segundos, construiría la instancia **`Profile`** de la siguiente manera:
>
> ```
> pr = cProfile.Profile(your_integer_time_func, 0.001
> ```
>
> As the **`cProfile.Profile`** class cannot be calibrated, custom timer functions should be used with care and should be as fast as possible. For the best results with a custom

timer, it might be necessary to hard-code it in the C source of the internal **`_lsprof`** module.

Python 3.3 agrega varias funciones nuevas en **`time`** que se puede usar para realizar mediciones precisas del proceso o el tiempo del reloj de pared (*wall-clock time*). Por ejemplo, vea **`time.perf_counter()`**.

`timeit` — Measure execution time of small code snippets

Código fuente: Lib/timeit.py [https://github.com/python/cpython/tree/3.14/Lib/timeit.py]

Este módulo proporciona una forma sencilla de cronometrar pequeños fragmentos de código Python. Tiene tanto una Interfaz de línea de comandos como una invocable. Evita una serie de trampas comunes para medir los tiempos de ejecución. Véase también la introducción de Tim Peters al capítulo «Algoritmos» en la segunda edición de *Python Cookbook*, publicado por O'Reilly.

Ejemplos básicos

En el ejemplo siguiente se muestra cómo se puede utilizar Interfaz de línea de comandos para comparar tres expresiones diferentes:

```
$ python -m timeit "'-'.join(str(n) for n in range(100))
10000 loops, best of 5: 30.2 usec per loop
$ python -m timeit "'-'.join([str(n) for n in range(100)
10000 loops, best of 5: 27.5 usec per loop
$ python -m timeit "'-'.join(map(str, range(100)))"
10000 loops, best of 5: 23.2 usec per loop
```

Esto se puede lograr desde Interfaz de Python con:

```
>>> import timeit
>>> timeit.timeit('"-".join(str(n) for n in range(100))'
0.3018611848820001
>>> timeit.timeit('"-".join([str(n) for n in range(100)]
0.2727368790656328
>>> timeit.timeit('"-".join(map(str, range(100)))', numb
0.23702679807320237
```

Un invocable también se puede pasar desde el Interfaz de Python:

```
>>> timeit.timeit(lambda: "-".join(map(str, range(100)))
0.19665591977536678
```

Sin embargo, tenga en cuenta que **`timeit()`** determinará automáticamente el número de repeticiones solo cuando se utiliza la interfaz de línea de comandos. En la sección Ejemplos puede encontrar ejemplos más avanzados.

Interfaz de Python

El módulo define tres funciones de conveniencia y una clase pública:

timeit.timeit(*stmt='pass', setup='pass', timer=<default timer>, number=1000000, globals=None*)

> Cree una instancia de **`Timer`** con la instrucción dada, el código *setup* y la función *timer* y ejecute su método **`timeit()`** con las ejecuciones *number*. El argumento *globals* opcional especifica un espacio de nombres en el que se ejecutará el código.
>
> *Distinto en la versión 3.5:* El parámetro opcional *globals* fue añadido.

timeit.repeat(*stmt='pass', setup='pass', timer=<default timer>, repeat=5, number=1000000, globals=None*)

> Crea una instancia de **`Timer`** con la instrucción dada, el código *setup* y la función *timer* y ejecuta su método **`repeat()`** con las ejecuciones *repeat* y *number* dadas. El argumento *globals* opcional especifica un espacio de nombres en el que se ejecutará el código.
>
> *Distinto en la versión 3.5:* El parámetro opcional *globals* fue añadido.
>
> *Distinto en la versión 3.7:* El valor por defecto para *repeat* cambió de 3 a 5.

timeit.default_timer()

El temporizador por defecto, que es siempre time.perf_counter(), retorna segundos como un número de punto flotante. Como alternativa, time.perf_counter_ns retorna un número entero de nanosegundos.

Distinto en la versión 3.3: `time.perf_counter()` es ahora el temporizador por defecto.

class timeit.Timer(*stmt='pass'*, *setup='pass'*, *timer=<timer function>*, *globals=None*)

Clase para la velocidad de tiempo de ejecución de pequeños fragmentos de código.

El constructor toma una instrucción que se cronometra, una instrucción adicional utilizada para la instalación, y una función de temporizador. Ambas instrucciones tienen como valor predeterminado `'pass'`; la función del temporizador depende de la plataforma (consulte la cadena *doc* del módulo). *stmt* y *setup* también pueden contener varias instrucciones separadas por `;` o líneas nuevas, siempre y cuando no contengan literales de cadena de varias líneas. La instrucción se ejecutará de forma predeterminada dentro del espacio de nombres timeit; este comportamiento se puede controlar pasando un espacio de nombres a *globals*.

Para medir el tiempo de ejecución de la primera instrucción, utilice el método `timeit()`. Los métodos `repeat()` y `autorange()` son métodos de conveniencia para llamar al método **`timeit()`** varias veces.

El tiempo de ejecución de *setup* se excluye de la ejecución de tiempo total.

Los parámetros *stmt* y *setup* también pueden tomar objetos a los que se puede llamar sin argumentos. Esto embebe llamadas a ellos en una función de temporizador que luego será ejecutada por `timeit()`. Tenga en cuenta que la sobrecarga de tiempo es un poco mayor en este caso debido a las llamadas de función adicionales.

Distinto en la versión 3.5: El parámetro opcional *globals* fue añadido.

timeit(*number=1000000*)

Registra la duración de *number* ejecuciones de la sentencia principal. Esto ejecuta la sentencia *setup* una vez y, a continuación, retorna el tiempo que se tarda en ejecutar la sentencia principal varias veces. El temporizador por defecto retorna segundos como un número de punto flotante. El argumento es el número de veces que se ejecuta el bucle, tomando por defecto un millón. La sentencia principal, la instrucción *setup* y la función *timer* que se va a utilizar se pasan al constructor.

Nota

De forma predeterminada, `timeit()` desactiva temporalmente garbage collection durante la medición. La ventaja de este enfoque es que hace que los tiempos independientes sean más comparables. La desventaja es que GC puede ser un componente importante del rendimiento de la función que se está midiendo. Si es así, GC se puede volver a habilitar como la primera instrucción en la cadena *setup*. Por ejemplo:

```
timeit.Timer('for i in range(10): oct(i)', 'g
```

autorange(*callback=None*)

Determina automáticamente cuántas veces llamar a `timeit()`.

This is a convenience function that calls `timeit()` repeatedly so that the total time >= 0.2 second, returning the eventual (number of loops, time taken for that number of loops). It calls **`timeit()`** with increasing numbers from the sequence 1, 2, 5, 10, 20, 50, … until the time taken is at least 0.2 seconds.

Si *callback* se da y no es `None`, se llamará después de cada prueba con dos argumentos: `callback(number, time_taken)`.

Added in version 3.6.

repeat(*repeat=5*, *number=1000000*)

Llama a **`timeit()`** algunas veces.

Esta es una función de conveniencia que llama a **`timeit()`** repetidamente, retornando una lista de resultados. El primer argumento especifica cuántas veces se debe llamar a **`timeit()`**. El segundo argumento especifica el argumento *number* para **`timeit()`**.

Nota

Es tentador calcular la media y la desviación estándar del vector de resultados e informar de estos. Sin embargo, esto no es muy útil. En un caso típico, el valor más bajo proporciona un límite inferior para la rapidez con la que el equipo puede ejecutar el fragmento de código especificado; valores más altos en el vector de resultado normalmente no son causados por la variabilidad en la velocidad de Python, sino por otros procesos que interfieren con su precisión de sincronización. Así que el **`min()`** del resultado es probablemente el único número que debería estar interesado en. Después de eso, usted debe mirar todo el vector y aplicar el sentido común en lugar de las estadísticas.

Distinto en la versión 3.7: El valor por defecto para *repeat* cambió de 3 a 5.

print_exc(*file=None*)

Ayudante para imprimir un seguimiento desde el código cronometrado.

Uso típico:

```
t = Timer(...)       # outside the try/except
try:
    t.timeit(...)    # or t.repeat(...)
except Exception:
    t.print_exc()
```

La ventaja sobre el seguimiento estándar es que se mostrarán las líneas de origen de la plantilla compilada. El argumento opcional *file* dirige dónde se envía el seguimiento; por defecto a `sys.stderr`.

Interfaz de línea de comandos

Cuando se llama como un programa desde la línea de comandos, se utiliza el siguiente formulario:

```
python -m timeit [-n N] [-r N] [-u U] [-s S] [-p] [-v] [
```

Cuando las siguientes opciones son entendidas:

-n N, --number=N

cuantas veces para ejecutar "*statement*"

-r N, --repeat=N

cuantas veces se va a repetir el temporizador (predeterminado 5)

-s S, --setup=S

instrucción a ser ejecutada una vez inicialmente (por defecto `pass`)

-p, --process

mide el tiempo de proceso, no el tiempo total de ejecución, utilizando `time.process_time()` en lugar de `time.perf_counter()`, que es el valor predeterminado

Added in version 3.3.

-u, --unit=U

especifica una unidad de tiempo para la salida del temporizador; puede seleccionar `nsec`, `usec`, `msec` o `sec`

Added in version 3.5.

-v, --verbose

imprime los resultados de tiempo en bruto; repetir para más dígitos de precisión

-h, --help

imprime un mensaje de uso corto y sale

Se puede dar una instrucción de varias líneas especificando cada línea como un argumento de instrucción independiente; Las líneas con sangría son posibles entrecomillando un argumento y utilizando espacios iniciales. Múltiples opciones `-s` se tratan de forma similar.

Si no se da `-n`, se calcula un número adecuado de bucles intentando aumentar los números de la secuencia 1, 2, 5, 10, 20, 50, … hasta que el tiempo total sea de al menos 0.2 segundos.

Las mediciones de `default_timer()` pueden verse afectadas por otros programas que se ejecutan en la misma máquina, por lo que lo mejor que se puede hacer cuando es necesario una medición de tiempo precisa es repetir la medición un par de veces y utilizar el mejor tiempo. La opción `-r` es buena para esto; el valor predeterminado de 5 repeticiones es probablemente suficiente en la mayoría de los casos. Puede usar `time.process_time()` para medir el tiempo de CPU.

Nota

Hay una cierta sobrecarga de línea base asociada con la ejecución de una instrucción pass. El código aquí no intenta ocultarlo, pero debe ser consciente de ello. La sobrecarga de línea base se puede medir invocando el programa sin argumentos y puede diferir entre versiones de Python.

Ejemplos

Es posible proporcionar una instrucción *setup* que se ejecuta solo una vez al inicio:

```
$ python -m timeit -s "text = 'sample string'; char = 'g
5000000 loops, best of 5: 0.0877 usec per loop
$ python -m timeit -s "text = 'sample string'; char = 'g
1000000 loops, best of 5: 0.342 usec per loop
```

En la salida, hay tres campos. El conteo del bucle, que indica cuántas veces se ejecutó el cuerpo de la declaración por tiempo de repetición del bucle. El conteo de repeticiones («el mejor de 5») el cual indica cuántas veces se repitió el ciclo de tiempo y, finalmente, el tiempo promedio que tomó el cuerpo de la declaración dentro de la mejor repetición del ciclo de tiempo. Es decir, el tiempo que tomó la repetición más rápida dividido por el número de bucles.

```
>>> import timeit
>>> timeit.timeit('char in text', setup='text = "sample
0.41440500499993504
>>> timeit.timeit('text.find(char)', setup='text = "samp
1.7246671520006203
```

Se puede hacer lo mismo usando la clase **`Timer`** y sus métodos:

```
>>> import timeit
>>> t = timeit.Timer('char in text', setup='text = "samp
>>> t.timeit()
0.3955516149999312
>>> t.repeat()
[0.40183617287970225, 0.37027556854118704, 0.38344867356
```

En los ejemplos siguientes se muestra cómo cronometrar expresiones que contienen varias líneas. Aquí comparamos el coste de usar **`hasattr()`** frente a **`try`**/**`except`** para probar los atributos de objeto faltantes y presentes:

```
$ python -m timeit "try:" "  str.__bool__" "except Attri
20000 loops, best of 5: 15.7 usec per loop
$ python -m timeit "if hasattr(str, '__bool__'): pass"
50000 loops, best of 5: 4.26 usec per loop

$ python -m timeit "try:" "  int.__bool__" "except Attri
200000 loops, best of 5: 1.43 usec per loop
$ python -m timeit "if hasattr(int, '__bool__'): pass"
100000 loops, best of 5: 2.23 usec per loop

>>> import timeit
>>> # attribute is missing
>>> s = """\
... try:
...     str.__bool__
... except AttributeError:
...     pass
... """
>>> timeit.timeit(stmt=s, number=100000)
0.9138244460009446
>>> s = "if hasattr(str, '__bool__'): pass"
>>> timeit.timeit(stmt=s, number=100000)
0.5829014980008651
>>>
>>> # attribute is present
>>> s = """\
... try:
...     int.__bool__
... except AttributeError:
...     pass
... """
>>> timeit.timeit(stmt=s, number=100000)
0.04215312199994514
>>> s = "if hasattr(int, '__bool__'): pass"
>>> timeit.timeit(stmt=s, number=100000)
0.08588060699912603
```

To give the `timeit` module access to functions you define, you can pass a *setup* parameter which contains an import statement:

```
def test():
    """Stupid test function"""
    L = [i for i in range(100)]

if __name__ == '__main__':
    import timeit
    print(timeit.timeit("test()", setup="from __main__ i
```

Otra opción es pasar `globals()` al parámetro *globals*, lo que hará que el código se ejecute dentro del espacio de nombres global actual. Esto puede ser más conveniente que especificar individualmente las importaciones:

```
def f(x):
    return x**2
def g(x):
    return x**4
def h(x):
    return x**8

import timeit
print(timeit.timeit('[func(42) for func in (f,g,h)]', gl
```

`trace` — Trace or track Python statement execution

Código fuente: Lib/trace.py [https://github.com/python/cpython/tree/3.14/Lib/trace.py]

The `trace` module allows you to trace program execution, generate annotated statement coverage listings, print caller/callee relationships and list functions executed during a program run. It can be used in another program or from the command line.

Ver también

Coverage.py [https://coverage.readthedocs.io/]
Una herramienta popular de terceros de cobertura que proporciona salida HTML junto con funciones avanzadas como cobertura de sucursales.

Uso de la línea de comandos

The `trace` module can be invoked from the command line. It can be as simple as

```
python -m trace --count -C . somefile.py ...
```

Lo anterior ejecutará `somefile.py` y generará listados anotados de todos los módulos de Python importados durante la ejecución en el directorio actual.

--help

Muestra uso y sale.

--version

Muestra la versión del módulo y sale.

Added in version 3.8: Added `--module` option that allows running an executable module.

Opciones principales

At least one of the following options must be specified when invoking **`trace`**. The `--listfuncs` option is mutually exclusive with the `--trace` and `--count` options. When **`--listfuncs`** is provided, neither **`--count`** nor **`--trace`** are accepted, and vice versa.

-c, --count

Genera un conjunto de archivos de lista anotados al finalizar el programa que muestra cuántas veces se ejecutó cada instrucción. Vea también `--coverdir`, `--file` y `--no-report` a continuación.

-t, --trace

Muestra las líneas a medida que se ejecutan.

-l, --listfuncs

Muestra las funciones ejecutadas al ejecutar el programa.

-r, --report

Genera una lista anotada de una ejecución del programa anterior que utilizó la opción `--count` y `--file`. Esto no ejecuta ningún código.

-T, --trackcalls

Muestra las relaciones de llamada expuestas al ejecutar el programa.

Modificadores

-f, --file=<file>

Nombre de un archivo para acumular recuentos durante varias ejecuciones de seguimiento. Debe usarse con la opción `--count`.

-C, --coverdir=<dir>

Directorio donde van los archivos del informe. El informe de cobertura para `paquete.modulo` se escribe en el archivo `directorio/paquete/modulo.cobertura`.

-m, --missing

Al generar listados anotados, marque las líneas que no se ejecutaron con `>>>>>>`.

-s, --summary

Cuando use `--count` o `--report`, escriba un breve resumen en stdout para cada archivo procesado.

-R, --no-report

No genera listados anotados. Esto es útil si tiene la intención de realizar varias ejecuciones con `--count`, y luego producir un solo conjunto de listados anotados al final.

-g, --timing

Prefija cada línea con la hora desde que se inició el programa. Solo se usa durante el rastreo.

Filtros

Estas opciones pueden repetirse varias veces.

--ignore-module=<mod>

Ignora cada uno de los nombres de módulo dados y sus submódulos (si es un paquete). El argumento puede ser una lista de nombres separados por una coma.

--ignore-dir=<dir>

Ignora todos los módulos y paquetes en el directorio y subdirectorios nombrados. El argumento puede ser una lista de directorios separados por `os.pathsep`.

Interfaz programática

class trace.Trace(*count=1, trace=1, countfuncs=0, countcallers=0, ignoremods=(), ignoredirs=(), infile=None, outfile=None, timing=False*)

Crea un objeto para rastrear la ejecución de una sola declaración o expresión. Todos los parámetros son opcionales. *count* habilita el conteo de números de línea. *trace* habilita el seguimiento de ejecución de línea. *countfuncs* habilita la lista de las funciones llamadas durante la ejecución. *countcallers* habilita el seguimiento de la relación de llamadas. *ignoremods* es una lista de módulos o paquetes para ignorar. *ignoredirs* es una lista de directorios cuyos módulos o paquetes deben ignorarse. *infile* es el nombre del archivo desde el cual leer la información de conteo almacenada. *outfile* es el nombre del archivo en el que se escribe la información de conteo actualizada. *timing* permite mostrar una marca de tiempo relativa al momento en que se inició el seguimiento.

run(*cmd*)

Ejecuta el comando y recopile estadísticas de la ejecución con los parámetros de seguimiento actuales. *cmd* debe ser una cadena o un objeto de código, adecuado para pasar a `exec()`.

runctx(*cmd, globals=None, locals=None*)

Ejecuta el comando y recopile estadísticas de la ejecución con los parámetros de seguimiento actuales, en los entornos globales y locales definidos. Si no está definido, *globals* y *locals* por defecto son diccionarios vacíos.

runfunc(*func, /, *args, **kwds*)

Llama a *func* con los argumentos dados bajo el control del objeto `Trace` con los parámetros de rastreo actuales.

results()

Retorna un objeto **`CoverageResults`** que contiene los resultados acumulativos de todas las llamadas anteriores a `run`, `runctx` y `runfunc` para la instancia dada **`Trace`**. No restablece los resultados de seguimiento acumulados.

class trace.CoverageResults

Un contenedor para los resultados de la cobertura, creado por **`Trace.results()`**. No debe ser creado directamente por el usuario.

update(*other*)

Fusiona datos de otro objeto **`CoverageResults`**.

write_results(*show_missing=True*, *summary=False*, *coverdir=None*, ***, *ignore_missing_files=False*)

Escribe los resultados de la cobertura. Configure *show_missing* para mostrar las líneas que no tuvieron coincidencias. Configurar *summary* para incluir en la salida el resumen de cobertura por módulo. *coverdir* especifica el directorio en el que se enviarán los archivos de resultado de cobertura. Si es `None`, los resultados de cada archivo fuente se colocan en su directorio.

If *ignore_missing_files* is `True`, coverage counts for files that no longer exist are silently ignored. Otherwise, a missing file will raise a **`FileNotFoundError`**.

Distinto en la versión 3.13: Added *ignore_missing_files* parameter.

Un ejemplo simple que demuestra el uso de la interfaz programática:

```
import sys
import trace

# create a Trace object, telling it what to ignore, and
# do tracing or line-counting or both.
tracer = trace.Trace(
    ignoredirs=[sys.prefix, sys.exec_prefix],
    trace=0,
    count=1)

# run the new command using the given tracer
tracer.run('main()')

# make a report, placing output in the current directory
r = tracer.results()
r.write_results(show_missing=True, coverdir=".")
```

`tracemalloc` — Trace memory allocations

Added in version 3.4.

Código fuente: Lib/tracemalloc.py [https://github.com/python/cpython/tree/3.14/Lib/tracemalloc.py]

El módulo tracemalloc es una herramienta de depuración para rastrear los espacios de memoria asignados por Python. Este proporciona la siguiente información:

- El rastreo al lugar de origen del objeto asignado
- Las estadísticas en los espacios de memoria asignados por nombre de archivo y por número de línea: tamaño total, número y tamaño promedio de los espacios de memoria asignados
- Calcula las diferencias entre dos informes instantáneos para detectar alguna filtración en la memoria

Para rastrear la mayoría de los espacios de memoria asignados por Python; el módulo debe empezar tan pronto como sea posible configurando la variable del entorno **`PYTHONTRACEMALLOC`** a `1`, o usando la opción del comando de línea **`-X`** `tracemalloc`. La función **`tracemalloc.start()`** puede ser llamada en tiempo de ejecución para empezar a rastrear las asignaciones de memoria de Python.

Por defecto, el rastreo de un espacio de memoria asignado solo guarda el cuadro mas reciente (1 cuadro). Para guardar 25 cuadros desde el **`` PYTHONTRACEMALLOC`comienzo, configura la variable del entorno a ``25` ``**, o usa **`-X`** `tracemalloc=25` en la opción de línea de comando.

Ejemplos

Mostrar los 10 principales

Mostrar los 10 archivos asignando la mayor cantidad de memoria:

```
import tracemalloc

tracemalloc.start()

# ... run your application ...

snapshot = tracemalloc.take_snapshot()
top_stats = snapshot.statistics('lineno')

print("[ Top 10 ]")
for stat in top_stats[:10]:
    print(stat)
```

Ejemplo de la salida del conjunto de pruebas de Python:

```
[ Top 10 ]
<frozen importlib._bootstrap>:716: size=4855 KiB, count=
<frozen importlib._bootstrap>:284: size=521 KiB, count=3
/usr/lib/python3.4/collections/__init__.py:368: size=244
/usr/lib/python3.4/unittest/case.py:381: size=185 KiB, c
/usr/lib/python3.4/unittest/case.py:402: size=154 KiB, c
/usr/lib/python3.4/abc.py:133: size=88.7 KiB, count=347,
<frozen importlib._bootstrap>:1446: size=70.4 KiB, count
<frozen importlib._bootstrap>:1454: size=52.0 KiB, count
<string>:5: size=49.7 KiB, count=148, average=344 B
/usr/lib/python3.4/sysconfig.py:411: size=48.0 KiB, coun
```

Se puede ver que Python ha cargado `4855 KiB` de data (código de bytes y constantes) desde los módulos y que el modulo **`collections`** asigno `24KiB` para crear tipos **`namedtuple`**.

Mira **`Snapshot.statistics()`** para más opciones.

Calcula las diferencias

Toma dos capturas instantáneas y muestra las diferencias:

```
import tracemalloc
tracemalloc.start()
# ... start your application ...

snapshot1 = tracemalloc.take_snapshot()
# ... call the function leaking memory ...
snapshot2 = tracemalloc.take_snapshot()

top_stats = snapshot2.compare_to(snapshot1, 'lineno')

print("[ Top 10 differences ]")
for stat in top_stats[:10]:
    print(stat)
```

Ejemplo de la salida antes y después de probar el conjunto de pruebas de Python:

```
[ Top 10 differences ]
<frozen importlib._bootstrap>:716: size=8173 KiB (+4428
/usr/lib/python3.4/linecache.py:127: size=940 KiB (+940
/usr/lib/python3.4/unittest/case.py:571: size=298 KiB (+
<frozen importlib._bootstrap>:284: size=1005 KiB (+166 K
/usr/lib/python3.4/mimetypes.py:217: size=112 KiB (+112
/usr/lib/python3.4/http/server.py:848: size=96.0 KiB (+9
/usr/lib/python3.4/inspect.py:1465: size=83.5 KiB (+83.5
/usr/lib/python3.4/unittest/mock.py:491: size=77.7 KiB (
/usr/lib/python3.4/urllib/parse.py:476: size=71.8 KiB (+
/usr/lib/python3.4/contextlib.py:38: size=67.2 KiB (+67.
```

Se puede ver que Python cargó `8173 KiB` de información del modulo (código de bytes y constantes), y que eso es `4428KiB` más de lo que ha sido cargado antes de los test, cuando la anterior captura de pantalla fue tomada. De manera similar, el modulo **`linecache`** ha almacenado en caché `940 KiB` del código fuente de Python para formatear los seguimientos, todo desde la captura instantánea.

Si el sistema tiene poca memoria libre, los informes instantáneos pueden ser escritos en el disco usando el método `Snapshot.dump()` para analizar el informe instantáneo offline. Después usa el método `Snapshot.load()` para actualizar el informe.

Consigue el seguimiento del bloque de memoria

Código para configurar el seguimiento del bloque de memoria más grande:

```
import tracemalloc

# Store 25 frames
tracemalloc.start(25)

# ... run your application ...

snapshot = tracemalloc.take_snapshot()
top_stats = snapshot.statistics('traceback')

# pick the biggest memory block
stat = top_stats[0]
print("%s memory blocks: %.1f KiB" % (stat.count, stat.s
for line in stat.traceback.format():
    print(line)
```

Ejemplo de la salida del conjunto de pruebas de Python (rastreo limitado a 25 cuadros):

```
903 memory blocks: 870.1 KiB
  File "<frozen importlib._bootstrap>", line 716
  File "<frozen importlib._bootstrap>", line 1036
  File "<frozen importlib._bootstrap>", line 934
  File "<frozen importlib._bootstrap>", line 1068
  File "<frozen importlib._bootstrap>", line 619
  File "<frozen importlib._bootstrap>", line 1581
  File "<frozen importlib._bootstrap>", line 1614
  File "/usr/lib/python3.4/doctest.py", line 101
    import pdb
  File "<frozen importlib._bootstrap>", line 284
  File "<frozen importlib._bootstrap>", line 938
  File "<frozen importlib._bootstrap>", line 1068
  File "<frozen importlib._bootstrap>", line 619
  File "<frozen importlib._bootstrap>", line 1581
  File "<frozen importlib._bootstrap>", line 1614
  File "/usr/lib/python3.4/test/support/__init__.py", li
    import doctest
  File "/usr/lib/python3.4/test/test_pickletools.py", li
    support.run_doctest(pickletools)
  File "/usr/lib/python3.4/test/regrtest.py", line 1276
    test_runner()
  File "/usr/lib/python3.4/test/regrtest.py", line 976
    display_failure=not verbose)
  File "/usr/lib/python3.4/test/regrtest.py", line 761
    match_tests=ns.match_tests)
  File "/usr/lib/python3.4/test/regrtest.py", line 1563
    main()
  File "/usr/lib/python3.4/test/__main__.py", line 3
    regrtest.main_in_temp_cwd()
  File "/usr/lib/python3.4/runpy.py", line 73
    exec(code, run_globals)
  File "/usr/lib/python3.4/runpy.py", line 160
    "__main__", fname, loader, pkg_name)
```

Se puede ver que la mayor parte de la memoria fue asignada en el módulo **`importlib`** para cargar datos (códigos de bytes y constantes) desde módulos `870.1 KiB`. El rastreo esta donde el módulo **`importlib`** cargó datos más recientemente: en la linea `import pdb` el módulo **`doctest`**. El rastreo puede cambiar si se carga un nuevo módulo.

Los 10 más bonitos

Codifica para configurar las 10 líneas que asignan gran parte de la memoria con una salida `bonita`, ignorando los archivos `<frozen importlib._bootstrap>` y `<unkownn>`:

```
import linecache
import os
import tracemalloc

def display_top(snapshot, key_type='lineno', limit=10):
    snapshot = snapshot.filter_traces((
        tracemalloc.Filter(False, "<frozen importlib._bo
        tracemalloc.Filter(False, "<unknown>"),
    ))
    top_stats = snapshot.statistics(key_type)

    print("Top %s lines" % limit)
    for index, stat in enumerate(top_stats[:limit], 1):
        frame = stat.traceback[0]
        print("#%s: %s:%s: %.1f KiB"
              % (index, frame.filename, frame.lineno, st
        line = linecache.getline(frame.filename, frame.l
        if line:
            print('    %s' % line)

    other = top_stats[limit:]
    if other:
        size = sum(stat.size for stat in other)
        print("%s other: %.1f KiB" % (len(other), size /
    total = sum(stat.size for stat in top_stats)
    print("Total allocated size: %.1f KiB" % (total / 10

tracemalloc.start()

# ... run your application ...

snapshot = tracemalloc.take_snapshot()
display_top(snapshot)
```

Ejemplo de la salida del conjunto de pruebas de Python:

```
Top 10 lines
#1: Lib/base64.py:414: 419.8 KiB
    _b85chars2 = [(a + b) for a in _b85chars for b in _b
```

```
#2: Lib/base64.py:306: 419.8 KiB
    _a85chars2 = [(a + b) for a in _a85chars for b in _a
#3: collections/__init__.py:368: 293.6 KiB
    exec(class_definition, namespace)
#4: Lib/abc.py:133: 115.2 KiB
    cls = super().__new__(mcls, name, bases, namespace)
#5: unittest/case.py:574: 103.1 KiB
    testMethod()
#6: Lib/linecache.py:127: 95.4 KiB
    lines = fp.readlines()
#7: urllib/parse.py:476: 71.8 KiB
    for a in _hexdig for b in _hexdig}
#8: <string>:5: 62.0 KiB
#9: Lib/_weakrefset.py:37: 60.0 KiB
    self.data = set()
#10: Lib/base64.py:142: 59.8 KiB
    _b32tab2 = [a + b for a in _b32tab for b in _b32tab]
6220 other: 3602.8 KiB
Total allocated size: 5303.1 KiB
```

Mira **`Snapshot.statistics()`** para más opciones.

Graba los tamaños actual y máximo de todos los bloques de memoria rastreados

El siguiente código calcula dos sumas como `0 + 1 + 2 + ...` de forma ineficiente, creando una lista con estos números. Esta lista consume mucha memoria de forma temporal. Podemos usar **`get_traced_memory()`** y **`reset_peak()`** para observar el pequeño uso de memoria después de que la suma es calculada, así como también el uso máximo de memoria durante el cálculo:

```
import tracemalloc

tracemalloc.start()

# Example code: compute a sum with a large temporary lis
large_sum = sum(list(range(100000)))
```

```
first_size, first_peak = tracemalloc.get_traced_memory()

tracemalloc.reset_peak()

# Example code: compute a sum with a small temporary lis
small_sum = sum(list(range(1000)))

second_size, second_peak = tracemalloc.get_traced_memory

print(f"{first_size=}, {first_peak=}")
print(f"{second_size=}, {second_peak=}")
```

Salida:

```
first_size=664, first_peak=3592984
second_size=804, second_peak=29704
```

El uso de `reset_peak()` asegura que podemos registrar precisamente el uso máximo de memoria durante el cálculo de `small_sum`, incluso si es mucho menor al máximo total desde la llamada a `start()`. Sin la llamada a **`reset_peak()`**, `second_peak` seguiría siendo el uso máximo de memoria del cálculo de `large_sum` (es decir, igual a `first_peak`). En este caso ambos máximos son mucho mayores al uso final de memoria, lo que sugiere que podemos optimizar (removiendo la llamada innecesaria a `list`, y escribiendo `sum(range(...))`).

API

Funciones

tracemalloc.clear_traces()

Limpia los rastros de los bloques de memoria asignados por Python.

Mira también la función `stop()`.

tracemalloc.get_object_traceback(*obj*)

Get the traceback where the Python object *obj* was allocated. Return a `Traceback` instance, or `None` if the **`tracemalloc`** module is not tracing memory allocations or did not trace the allocation of the object.

Mira también las funciones `gc.get_referrers()` y `sys.getsizeof()`.

tracemalloc.get_traceback_limit()

Obtiene el número máximo de cuadros guardados en el seguimiento de un rastro.

The **`tracemalloc`** module must be tracing memory allocations to get the limit, otherwise an exception is raised.

El limite es establecido por la función `start()`.

tracemalloc.get_traced_memory()

Get the current size and peak size of memory blocks traced by the **`tracemalloc`** module as a tuple: `(current: int, peak: int)`.

tracemalloc.reset_peak()

Set the peak size of memory blocks traced by the **`tracemalloc`** module to the current size.

Do nothing if the **`tracemalloc`** module is not tracing memory allocations.

Esta función sólo modifica el valor guardado para el uso máximo de memoria, y no modifica o borra ningún rastreo, no como `clear_traces()`. Capturas tomadas con `take_snapshot()` antes de una llamada a `reset_peak()` pueden ser comparadas significativamente con capturas hechas después de la llamada.

Mira también la función `get_traced_memory()`.

Added in version 3.9.

tracemalloc.get_tracemalloc_memory()

Get the memory usage in bytes of the **`tracemalloc`** module used to store traces of memory blocks. Return an **`int`**.

tracemalloc.is_tracing()

`True` if the **`tracemalloc`** module is tracing Python memory allocations, `False` otherwise.

También mira las funciones **`start()`** y **`stop()`**.

tracemalloc.start(*nframe*: *int* = *1*)

Empieza a rastrear las asignaciones de memoria de Python: instala *hooks* en las asignaciones de memoria de Python. Los rastreos coleccionados van a estar limitados a *nframe*. Por defecto, un rastro de un bloque de memoria solo guarda el cuadro mas reciente: el limite es `1`. *nframe* debe ser mayor o igual a *1*.

El número original de cuadros totales que componen el seguimiento observando el atributo **`Traceback.total_nframe`**.

Guardar mas de `1` cuadro es solo útil para calcular estadísticas agrupadas por seguimiento o para calcular estadísticas acumulativa: mira las funciones **`Snapshot.compare_to()`** y **`Snapshot.statistics()`**.

Storing more frames increases the memory and CPU overhead of the **`tracemalloc`** module. Use the **`get_tracemalloc_memory()`** function to measure how much memory is used by the **`tracemalloc`** module.

La variable del entorno **`PYTHONTRACEMALLOC`** (`PYTHONTRACEMALLOC=NFRAME`) y la opción de comando de linea **`-X`** `tracemalloc=NFRAME` se puede usar para empezar a rastrear desde el inicio.

También mira las funciones **`stop()`**, **`is_tracing()`** y **`get_traceback_limit()`**.

tracemalloc.stop()

Detiene el rastreo de las asignaciones de memoria de Python: desinstala los *hooks*. También limpia todos los rastros de memoria recolectados acerca de los bloques de memoria asignados por Python.

Llama a la función `take_snapshot()` para tomar una captura instantánea de los rastreos, antes de limpiarlos.

También mira las funciones `start()`, `is_tracing()` y `clear_traces()`.

tracemalloc.take_snapshot()

Toma una captura instantánea de los bloques de memoria asignados por Python. Retorna una nueva instancia de la clase `Snapshot`.

The snapshot does not include memory blocks allocated before the **`tracemalloc`** module started to trace memory allocations.

Los seguimientos de los rastros son limitados por la función `get_traceback_limit()`. Usa el parámetro *nframe* de la función `start()` para guardar mas cuadros.

The **`tracemalloc`** module must be tracing memory allocations to take a snapshot, see the `start()` function.

También mira la función `get_object_traceback()`.

Filtro de dominio

class tracemalloc.DomainFilter(*inclusive: bool*, *domain: int*)

Filtra los rastros de los bloques de memoria por su espacio de dirección (dominio).

Added in version 3.6.

inclusive

Si *inclusive* es `True` (incluye), relaciona los bloques de memoria asignados en el espacio de dirección **`domain`**.

Si *inclusive* es `False` (excluye), relaciona los bloques de memoria no asignados en el espacio de dirección **`domain`**.

domain

Espacio de dirección de un bloque de memoria (`int`). Propiedad solo-lectura.

Filtro

class tracemalloc.Filter(*inclusive: bool*, *filename_pattern: str*, *lineno: int = None*, *all_frames: bool = False*, *domain: int = None*)

Filtra los rastros de los bloques de memoria.

También mira la función **`fnmatch.fnmatch()`** para la sintaxis de *filename_pattern*. La extensión `'.pyc'` es remplazada por `'.py'`.

Ejemplos:

- `Filter(True, subprocess.__file__)` solo incluye los rastros de el módulo **`subprocess`**
- `Filter(False, tracemalloc.__file__)` excludes traces of the **`tracemalloc`** module
- `Filter(False, "<unknown>")` excluye los seguimientos vacíos

Distinto en la versión 3.5: La extensión `'.pyo'` ya no se remplaza con `'.py'`.

Distinto en la versión 3.6: Agregado el atributo **`domain`** .

domain

El espacio de dirección de un bloque de memoria (`int` o `None`).

tracemalloc usa el dominio `0` para rastrear las asignaciones de memoria hechas por Python. Las extensiones C pueden usar otros dominios para rastrear otros recursos.

inclusive

Si *inclusive* es `True` (incluye), solo relaciona los bloques de memoria asignados en un archivo con el nombre coincidiendo con el atributo **`filename_pattern`** en el número de línea del atributo **`lineno`**.

Si *inclusive* es `False` (excluye), ignora los bloques de memoria asignados en un archivo con el nombre coincidiendo con el atributo **`filename_pattern`** en el número de línea del atributo **`lineno`**.

lineno

El número de linea (`int`) del filtro. Si *lineno* es `None`, el filtro se relaciona con cualquier número de linea.

filename_pattern

El patrón del nombre de archivo del filtro (`str`). Propiedad solo-lectura.

all_frames

Si *all_frames* es `True`, todos los cuadros de los rastreos son chequeados. Si *all_frames* es `False`, solo el cuadro mas reciente es chequeado.

El atributo no tiene efecto si el limite de rastreo es `1`. Mira la función **`get_traceback_limit()`** y el atributo **`Snapshot.traceback_limit`**.

Cuadro

class tracemalloc.Frame

Cuadro de un rastreo.

La clase **`Traceback`** es una secuencia de las instancias de la clase **`Frame`**.

filename

Nombre de archivo (`str`).

lineno

Número de línea (`int`).

Captura instantánea

class tracemalloc.Snapshot

Captura instantánea de los rastros de los bloques de memoria asignados por Python.

La función **`take_snapshot()`** crea una instancia de captura instantánea.

compare_to(*old_snapshot: Snapshot*, *key_type: str*, *cumulative: bool = False*)

Calcula las diferencias con una vieja captura instantánea. Obtiene las estadísticas en una lista ordenada de instancias de la clase **`StatisticDiff`** agrupadas por *key_type*.

Mira el método **`Snapshot.statistics()`** para los parámetros *key_type* y *cumulative*.

El resultado esta guardado desde el más grande al más pequeño por los valores absolutos de **`StatisticDiff.size_diff()`**, **`StatisticDiff.size()`**, el valor absoluto de **`StatisticDiff.count_diff()`**, **`Statistic.count()`** y después por el atributo **`StatisticDiff.traceback()`**.

dump(*filename*)

Escribe la captura instantánea en un archivo.

Usa el método `load()` para recargar la captura instantánea.

filter_traces(*filters*)

Crea una nueva instancia de clase `Snapshot` con una secuencia de `traces` filtrados, *filters* es una lista de las instancias de `DomainFilter` y `Filter`. Si *filters* es una lista vacía, retorna una nueva instancia de clase **`Snapshot`** con una copia de los rastreos.

Los filtros `todo incluido` se aplican de a uno, si los filtros no incluidos coinciden. Si al menos un filtro exclusivo coincide, se ignora un rastro.

Distinto en la versión 3.6: Las instancias de clase `DomainFilter` ahora también son aceptadas en *filters*.

classmethod load(*filename*)

Carga la captura instantánea desde un archivo.

También mira el método `dump()`.

statistics(*key_type: str*, *cumulative: bool = False*)

Obtiene estadísticas como una lista ordenada, de instancias de `Statistic` agrupadas por *key_type*:

key_type	**descripción**
`'filename'`	nombre del archivo
`'lineno'`	nombre del archivo y número de línea
`'traceback'`	seguimiento

If *cumulative* is `True`, cumulate size and count of memory blocks of all frames of the traceback of a trace, not only the most recent frame. The cumulative mode can only be used with *key_type* equal to `'filename'` and `'lineno'`.

El resultado se organiza desde el más grande hasta el más pequeño por el atributo `Statistic.size`, `Statistic.count` y después por `Statistic.traceback`.

traceback_limit

El número máximo de cuadros organizados en el rastreo del atributo `traces`: resulta de la función `get_traceback_limit` cuando la captura instantánea fue tomada.

traces

Rastros de todos los bloques de memoria asignados por Python: secuencia de instancias de `Trace`.

La secuencia tiene un orden que no esta definido. Usa el método `Snapshot.statistics()` para obtener una lista ordenada de estadísticas.

Estadística

class tracemalloc.Statistic

Estadística de las asignaciones de memoria.

`Snapshot.statistics()` retorna una lista de instancias de la clase `Statistic`.

También mira la clase `StatisticDiff`.

count

Número de bloques de memoria (`int`).

size

El tamaño total de los bloques de memoria en bytes (`int`).

traceback

Rastrea donde un bloque de memoria fue asignado, la instancia de la clase `Traceback`.

StatisticDiff

class tracemalloc.StatisticDiff

La diferencia de estadística en las asignaciones de memoria entre una vieja y una nueva instancia de clase `Snapshot`.

`Snapshot.compare_to()` retorna una lista de instancias de la clase `StatisticDiff`. Mira también la clase `Statistic`.

count

El número de bloques de memoria en la nueva captura de pantalla (`int`): `0` si los bloques de memoria han sido liberados en la nueva captura instantánea.

count_diff

La diferencia de los números de los bloques de memoria entre las capturas viejas y nuevas (`int`): `0` si el bloque de memoria ha sido asignado en la nueva captura instantánea.

size

El tamaño total de los bloques de memoria en bytes en la nueva captura instantánea (`int`): `0` si el bloque de memoria ha sido liberado en la nueva captura instantánea.

size_diff

La diferencia de el tamaño total de un bloque de memoria en bytes entre las capturas instantáneas viejas y nuevas (`int`): `0` si el bloque de memoria ha sido asignado en la nueva captura instantánea.

traceback

Rastrea donde los bloques de memoria han sido asignados, instancia de la clase `Traceback`.

Rastro

class tracemalloc.Trace

Rastro de un bloque de memoria.

El atributo **`Snapshot.traces`** es una secuencia de las instancias de la clase **`Trace`**.

Distinto en la versión 3.6: Agregado el atributo **`domain`** .

domain

Espacio de dirección de un bloque de memoria (`int`). Propiedad solo-lectura.

tracemalloc usa el dominio `0` para rastrear las asignaciones de memoria hechas por Python. Las extensiones C pueden usar otros dominios para rastrear otros recursos.

size

Tamaño de un bloque de memoria en bytes (`int`).

traceback

Rastrea donde un bloque de memoria fue asignado, la instancia de la clase **`Traceback`**.

Seguimiento

class tracemalloc.Traceback

La secuencia de las instancias de la clase **`Frame`** organizadas desde el cuadro mas antiguo al más reciente.

Un seguimiento contiene por lo menos `1` cuadro. Si el módulo `tracemalloc` falla al traer un cuadro, se usa el nombre del archivo `"<unknown>"` en el número de linea `0`.

When a snapshot is taken, tracebacks of traces are limited to **`get_traceback_limit()`** frames. See the **`take_snapshot()`** function. The original number of frames of the traceback is stored in the **`Traceback.total_nframe`** attribute. That allows one to know if a traceback has been truncated by the traceback limit.

The **`Trace.traceback`** attribute is a **`Traceback`** instance.

Distinto en la versión 3.7: Los cuadros están organizados desde el mas antiguo hasta el más reciente, en vez de el más reciente al más antiguo.

total_nframe

Número total de cuadros que componen el seguimiento antes de ser truncado. Este atributo puede ser `None` si no hay información disponible.

Distinto en la versión 3.9: El atributo **`Traceback.total_nframe`** fue añadido.

format(*limit=None*, *most_recent_first=False*)

Formatea el seguimiento como una lista de líneas. Usa el módulo **`linecache`** para recuperar líneas del código fuente. Si se configura el límite, formatea los cuadros más recientes de los límites, si límite es positivo. De otra manera, formatea los cuadros más antiguos de `abs(limit)`. Si *most_recent_first* es `True`, se invierte el orden de los cuadros formateados, devolviendo el cuadro más reciente en lugar del último.

Similar a la función **`traceback.format_tb()`**, excepto por el método **`format()`** que no incluye nuevas líneas.

Ejemplo:

```
print("Traceback (most recent call first):")
for line in traceback:
    print(line)
```

Salida:

```
Traceback (most recent call first):
```

```
File "test.py", line 9
  obj = Object()
File "test.py", line 12
  tb = tracemalloc.get_object_traceback(f())
```

Empaquetado y distribución de software

Estas bibliotecas te ayudan a publicar e instalar software de Python. Aunque estos módulos están diseñados para funcionar junto con Python Package Index [https://pypi.org], también pueden ser utilizados con un servidor de indexado local, o sin servidor de indexado.

- **`ensurepip`** — Bootstrapping the **`pip`** installer
    - Command-line interface
    - API del módulo
- **`venv`** — Creation of virtual environments
    - Creación de entornos virtuales
    - Cómo funcionan los venvs
    - API
    - Un ejemplo de la extensión de **`EnvBuilder`**
- **`zipapp`** — Manage executable Python zip archives
    - Ejemplo básico
    - Interfaz de línea de comando
    - API de Python
    - Ejemplos
    - Especificar el intérprete
    - Creando aplicaciones independientes con zipapp
        - Advertencias
    - El formato de archivado Zip de aplicaciones Python

`ensurepip` — Bootstrapping the `pip` installer

Added in version 3.4.

Source code: Lib/ensurepip [https://github.com/python/cpython/tree/3.14/Lib/ensurepip]

The **`ensurepip`** package provides support for bootstrapping the `pip` installer into an existing Python installation or virtual environment. This bootstrapping approach reflects the fact that `pip` is an independent project with its own release cycle, and the latest available stable version is bundled with maintenance and feature releases of the CPython reference interpreter.

En la mayoría de los casos, los usuarios finales de Python no deberían tener que invocar este módulo directamente (como `pip` deben arrancarse de forma predeterminada), pero puede ser necesario si se omitió la instalación de `pip` al instalar Python (o al crear un entorno virtual) o después de desinstalar explícitamente `pip`.

Nota

Este módulo *no* accede a Internet. Todos los componentes necesarios para ejecutar `pip` se incluyen como partes internas del paquete.

This is an optional module. If it is missing from your copy of CPython, look for documentation from your distributor (that is, whoever provided Python to you). If you are the distributor, see Requirements for optional modules.

Ver también

Installing Python modules
La guía del usuario final para instalar paquetes python

PEP 453 [https://peps.python.org/pep-0453/] : Arranque explícito de pip en instalaciones de Python
La justificación original y la especificación de este módulo.

Availability: not Android, not iOS, not WASI.

This module is not supported on mobile platforms or WebAssembly platforms.

Command-line interface

La interfaz de línea de comandos se invoca mediante el modificador `-m` del intérprete.

La invocación más simple posible es:

```
python -m ensurepip
```

Esta invocación instalará `pip` si aún no está instalado, pero de lo contrario no hace nada. Para asegurarse de que la versión instalada de `pip` sea al menos tan reciente como la disponible en `ensurepip`, pase la opción `--upgrade`:

```
python -m ensurepip --upgrade
```

De forma predeterminada, `pip` se instala en el entorno virtual actual (si uno está activo) o en los paquetes de sitio del sistema (si no hay ningún entorno virtual activo). La ubicación de instalación se puede controlar a través de dos opciones de línea de comandos adicionales:

--root <dir>
Installs `pip` relative to the given root directory rather than the root of the currently active virtual environment (if any) or the default root for the current Python installation.

--user

Installs `pip` into the user site packages directory rather than globally for the current Python installation (this option is not permitted inside an active virtual environment).

De forma predeterminada, se instalarán los scripts `pipX` y `pipX.Y` (donde X.Y representa la versión de Python utilizada para invocar `ensurepip`). Los scripts instalados se pueden controlar a través de dos opciones de línea de comandos adicionales:

--altinstall

If an alternate installation is requested, the `pipX` script will *not* be installed.

--default-pip

If a «default pip» installation is requested, the `pip` script will be installed in addition to the two regular scripts.

Proporcionar ambas opciones de selección de script desencadenará una excepción.

API del módulo

`ensurepip` exposes two functions for programmatic use:

ensurepip.version()

Retorna una cadena que especifica la versión disponible de pip que se instalará al arrancar un entorno.

ensurepip.bootstrap(*root=None*, *upgrade=False*, *user=False*, *altinstall=False*, *default_pip=False*, *verbosity=0*)

Ejecuta `pip` en el entorno actual o designado.

root especifica un directorio raíz alternativo para instalar en relación con. Si *root* es `None`, la instalación utiliza la ubicación de instalación predeterminada para el entorno actual.

upgrade indica si se debe actualizar o no una instalación existente de una versión anterior de `pip` a la versión disponible.

user indica si se debe utilizar el esquema de usuario en lugar de instalar globalmente.

De forma predeterminada, se instalarán los scripts `pipX` y `pipX.Y` (donde X.Y representa la versión actual de Python).

Si se establece *altinstall*, *no* se instalará `pipX`.

Si se establece *default_pip*, se instalará `pip` además de los dos scripts normales.

Establecer tanto *altinstall* como *default_pip* desencadenará **`ValueError`**.

verbosity controla el nivel de salida a **`sys.stdout`** de la operación de ejecución.

Genera un evento auditing `ensurepip.bootstrap` con el argumento `root`.

Nota

El proceso de ejecución tiene efectos secundarios tanto en `sys.path` como `os.environ`. Invocar la interfaz de línea de comandos en un subproceso en su lugar permite evitar estos efectos secundarios.

Nota

El proceso de ejecución puede instalar módulos adicionales requeridos por `pip`, pero otro software no debe asumir que esas dependencias siempre estarán presentes de forma predeterminada (ya que las dependencias se pueden eliminar en una versión futura de `pip`).

venv — Creation of virtual environments

Added in version 3.3.

Código fuente: Lib/venv/ [https://github.com/python/cpython/tree/3.14/Lib/venv/]

The **venv** module supports creating lightweight «virtual environments», each with their own independent set of Python packages installed in their **site** directories. A virtual environment is created on top of an existing Python installation, known as the virtual environment's «base» Python, and by default is isolated from the packages in the base environment, so that only those explicitly installed in the virtual environment are available. See Virtual Environments and **site**'s virtual environments documentation for more information.

When used from within a virtual environment, common installation tools such as pip [https://pypi.org/project/pip/] will install Python packages into a virtual environment without needing to be told to do so explicitly.

A virtual environment is (amongst other things):

- Used to contain a specific Python interpreter and software libraries and binaries which are needed to support a project (library or application). These are by default isolated from software in other virtual environments and Python interpreters and libraries installed in the operating system.
- Contained in a directory, conventionally named `.venv` or `venv` in the project directory, or under a container directory for lots of virtual environments, such as `~/.virtualenvs`.
- Not checked into source control systems such as Git.
- Considered as disposable – it should be simple to delete and

recreate it from scratch. You don’t place any project code in the environment.
- Not considered as movable or copyable – you just recreate the same environment in the target location.

Ver **PEP 405** [https://peps.python.org/pep-0405/] para más información sobre los entornos virtuales de Python.

Ver también

Python Packaging User Guide: Creating and using virtual environments [https://packaging.python.org/guides/installing-using-pip-and-virtual-environments/#create-and-use-virtual-environments]

Availability: not Android, not iOS, not WASI.

This module is not supported on mobile platforms or WebAssembly platforms.

Creación de entornos virtuales

Virtual environments are created by executing the `venv` module:

```
python -m venv /path/to/new/virtual/environment
```

This creates the target directory (including parent directories as needed) and places a `pyvenv.cfg` file in it with a `home` key pointing to the Python installation from which the command was run. It also creates a `bin` (or `Scripts` on Windows) subdirectory containing a copy or symlink of the Python executable (as appropriate for the platform or arguments used at environment creation time). It also creates a `lib/pythonX.Y/site-packages` subdirectory (on Windows, this is `Lib\site-packages`). If an existing directory is specified, it will be re-used.

Distinto en la versión 3.5: The use of `venv` is now recommended for creating virtual environments.

Deprecated since version 3.6, removed in version 3.8: **pyvenv** was the recommended tool for creating virtual environments for Python 3.3

and 3.4, and replaced in 3.5 by executing `venv` directly.

On Windows, invoke the `venv` command as follows:

```
PS> python -m venv C:\path\to\new\virtual\environment
```

The command, if run with `-h`, will show the available options:

```
usage: venv [-h] [--system-site-packages] [--symlinks |
            [--upgrade] [--without-pip] [--prompt PROMPT
            [--without-scm-ignore-files]
            ENV_DIR [ENV_DIR ...]

Creates virtual Python environments in one or more targe

Once an environment has been created, you may wish to ac
sourcing an activate script in its bin directory.
```

ENV_DIR

A required argument specifying the directory to create the environment in.

--system-site-packages

Give the virtual environment access to the system site-packages directory.

--symlinks

Try to use symlinks rather than copies, when symlinks are not the default for the platform.

--copies

Try to use copies rather than symlinks, even when symlinks are the default for the platform.

--clear

Delete the contents of the environment directory if it already exists, before environment creation.

--upgrade

Upgrade the environment directory to use this version of Python, assuming Python has been upgraded in-place.

--without-pip

Skips installing or upgrading pip in the virtual environment (pip is bootstrapped by default).

--prompt <PROMPT>

Provides an alternative prompt prefix for this environment.

--upgrade-deps

Upgrade core dependencies (pip) to the latest version in PyPI.

--without-scm-ignore-files

Skips adding SCM ignore files to the environment directory (Git is supported by default).

Distinto en la versión 3.4: Installs pip by default, added the `--without-pip` and `--copies` options.

Distinto en la versión 3.4: In earlier versions, if the target directory already existed, an error was raised, unless the `--clear` or `--upgrade` option was provided.

Distinto en la versión 3.9: Add `--upgrade-deps` option to upgrade pip + setuptools to the latest on PyPI.

Distinto en la versión 3.12: `setuptools` is no longer a core venv dependency.

Distinto en la versión 3.13: Added the `--without-scm-ignore-files` option.

Distinto en la versión 3.13: `venv` now creates a `.gitignore` file for Git by default.

Nota

While symlinks are supported on Windows, they are not recommended. Of particular note is that double-clicking `python.exe` in File Explorer will resolve the symlink eagerly and ignore the virtual environment.

Nota

On Microsoft Windows, it may be required to enable the `Activate.ps1` script by setting the execution policy for the user. You can do this by issuing the following PowerShell command:

```
PS C:\> Set-ExecutionPolicy -ExecutionPolicy RemoteSign
```

See About Execution Policies [https://go.microsoft.com/fwlink/?LinkID=135170] for more information.

The created `pyvenv.cfg` file also includes the `include-system-site-packages` key, set to `true` if `venv` is run with the `--system-site-packages` option, `false` otherwise.

Unless the `--without-pip` option is given, **`ensurepip`** will be invoked to bootstrap `pip` into the virtual environment.

Multiple paths can be given to `venv`, in which case an identical virtual environment will be created, according to the given options, at each provided path.

Cómo funcionan los venvs

When a Python interpreter is running from a virtual environment, **`sys.prefix`** and **`sys.exec_prefix`** point to the directories of the virtual environment, whereas **`sys.base_prefix`** and **`sys.base_exec_prefix`** point to those of the base Python used to create the environment. It is sufficient to check `sys.prefix != sys.base_prefix` to determine if the current interpreter is running from a virtual environment.

A virtual environment may be «activated» using a script in its binary directory (`bin` on POSIX; `Scripts` on Windows). This will

prepend that directory to your `PATH`, so that running **python** will invoke the environment's Python interpreter and you can run installed scripts without having to use their full path. The invocation of the activation script is platform-specific (`<venv>` must be replaced by the path to the directory containing the virtual environment):

Plataforma	Shell	Comando para activar el entorno virtual
POSIX	bash/zsh	`$ source <venv>/bin/activate`
	fish	`$ source <venv>/bin/activate.fish`
	csh/tcsh	`$ source <venv>/bin/activate.csh`
	pwsh	`$ <venv>/bin/Activate.ps1`
Windows	cmd.exe	`C:\> <venv>\Scripts\activate.bat`
	PowerShell	`PS C:\> <venv>\Scripts\Activate.ps1`

Added in version 3.4: **fish** and **csh** activation scripts.

Added in version 3.8: Scripts de activación de PowerShell instalados en POSIX para compatibilidad con PowerShell Core.

No *necesita* específicamente activar un entorno virtual, ya que puede solo especificar la ruta completa al intérprete de Python de ese entorno al invocar Python. Además, todos los scripts instalados en el entorno deben poder ejecutarse sin activarlo.

In order to achieve this, scripts installed into virtual environments have a «shebang» line which points to the environment's Python interpreter, `#!/<path-to-venv>/bin/python`. This means that the script will run with that interpreter regardless of the value of `PATH`. On Windows, «shebang» line processing is supported if you have the Python install manager installed. Thus, double-clicking an installed script in a Windows Explorer window should run it with the correct interpreter without the environment needing to be activated or on the `PATH`.

Cuando se ha activado un entorno virtual, la variable de entorno `VIRTUAL_ENV` se establece en la ruta del entorno. Dado que no se requiere activar explícitamente una entorno virtual para usarla, no se puede confiar en `VIRTUAL_ENV` para determinar si se está usando un entorno virtual.

Advertencia

Debido a que los scripts instalados en entornos no deben esperar a que el entorno se active, sus líneas shebang contienen las rutas absolutas a los intérpretes de su entorno. Debido a esto, los entornos son inherentemente no portables, en el caso general. Siempre debe tener un medio simple para recrear un entorno (por ejemplo, si tiene un archivo de requisitos `requirements.txt`, puede invocar `pip install -r requirements.txt` al usar `pip` del entorno para instalar todos los paquetes que necesite el entorno). Si por alguna razón necesita mover el entorno a una nueva ubicación, debe recrearlo en la ubicación deseada y eliminar el de la ubicación anterior. Si mueve un entorno porque movió un directorio principal del mismo, debe recrear el entorno en su nueva ubicación. De lo contrario, el software instalado en el entorno puede no funcionar como se espera.

Puede desactivar un entorno virtual al escribir `deactivate` en el shell. El mecanismo exacto es específico de la plataforma y es un detalle de implementación interno (se usará normalmente una función de script o shell).

API

El método de alto nivel descrito anteriormente utiliza una sencilla API que proporciona mecanismos para que les creadores de entornos virtuales de terceras/os puedan personalizar la creación de entornos según sus necesidades, la clase **`EnvBuilder`**.

class venv.EnvBuilder(*system_site_packages=False*, *clear=False*, *symlinks=False*, *upgrade=False*, *with_pip=False*, *prompt=None*, *upgrade_deps=False*, ***, *scm_ignore_files=frozenset()*)

La clase **`EnvBuilder`** acepta los siguientes argumentos de palabras clave en la instanciación:

- *system_site_packages* – a boolean value indicating that the system Python site-packages should be available to the environment (defaults to `False`).

- *clear* – a boolean value which, if true, will delete the contents of any existing target directory, before creating the environment.
- *symlinks* – a boolean value indicating whether to attempt to symlink the Python binary rather than copying.
- *upgrade* – a boolean value which, if true, will upgrade an existing environment with the running Python - for use when that Python has been upgraded in-place (defaults to `False`).
- *with_pip* – a boolean value which, if true, ensures pip is installed in the virtual environment. This uses **`ensurepip`** with the `--default-pip` option.
- *prompt* – a string to be used after virtual environment is activated (defaults to `None` which means directory name of the environment would be used). If the special string `"."` is provided, the basename of the current directory is used as the prompt.
- *upgrade_deps* – Update the base venv modules to the latest on PyPI
- *scm_ignore_files* – Create ignore files based for the specified source control managers (SCM) in the iterable. Support is defined by having a method named `create_{scm}_ignore_file`. The only value supported by default is `"git"` via **`create_git_ignore_file()`**.

Distinto en la versión 3.4: Añadido el parámetro `with_pip`

Distinto en la versión 3.6: Añadido el parámetro `prompt`

Distinto en la versión 3.9: Se agregó el parámetro `upgrade_deps`

Distinto en la versión 3.13: Added the `scm_ignore_files` parameter

`EnvBuilder` may be used as a base class.

create(*env_dir*)

Crear un entorno virtual especificando el directorio de destino (de forma absoluta o relativa al directorio actual) que ha de contener el entorno virtual. El método `create` creará el entorno en el directorio especificado, o lanzará la correspondiente excepción.

El método `create` de la clase **`EnvBuilder`** ilustra los enlaces disponibles para la personalización de la subclase:

```
def create(self, env_dir):
    """
    Create a virtualized Python environment in
    env_dir is the target directory to create a
    """
    env_dir = os.path.abspath(env_dir)
    context = self.ensure_directories(env_dir)
    self.create_configuration(context)
    self.setup_python(context)
    self.setup_scripts(context)
    self.post_setup(context)
```

Cada uno de los métodos **`ensure_directories()`**, **`create_configuration()`**, **`setup_python()`**, **`setup_scripts()`** y **`post_setup()`** pueden ser anulados.

ensure_directories(*env_dir*)

Crea el directorio del entorno y todos los subdirectorios necesarios que aún no existen, y retorna un objeto de contexto. Este objeto de contexto es solo un contenedor de atributos (como rutas) para que lo usen otros métodos. Si se crea **`EnvBuilder`** con el argumento `clear=True`, se borrará el contenido del directorio del entorno y luego se recrearán todos los subdirectorios necesarios.

El objeto de contexto que retorna es un **`types.SimpleNamespace`** con los siguientes atributos:

- `env_dir` - La ubicación del entorno virtual. Se usa para `__VENV_DIR__` en scripts de activación (ver **`install_scripts()`**).
- `env_name` - El nombre del entorno virtual. Se usa para `__VENV_NAME__` en scripts de activación (ver **`install_scripts()`**).
- `prompt` - El prompt que utilizarán los scripts de activación. Se usa para `__VENV_PROMPT__` en scripts de activación (ver **`install_scripts()`**).
- `executable` - El ejecutable de Python subyacente que se utiliza por el entorno virtual. Esto tiene en cuenta el caso donde se crea un entorno virtual a partir de otro entorno virtual.
- `inc_path` - La ruta de include para el entorno virtual.
- `lib_path` - La ruta de purelib para el entorno virtual.
- `bin_path` - La ruta del script para el entorno virtual.
- `bin_name` - El nombre de la ruta del script en relación con la ubicación del entorno virtual. Se usa para `__VENV_BIN_NAME__` en scripts de activación (ver **`install_scripts()`**).
- `env_exe` - El nombre del intérprete de Python en el entorno virtual. Se usa para `__VENV_PYTHON__` en scripts de activación (ver **`install_scripts()`**).
- `env_exec_cmd` - El nombre del intérprete de Python, teniendo en cuenta las redirecciones del sistema de archivos. Se puede utilizar para ejecutar Python en el entorno virtual.

Distinto en la versión 3.11: El esquema de instalación de sysconfig de *venv* se utiliza para construir las rutas de los directorios creados.

Distinto en la versión 3.12: Se agregó el atributo `lib_path` al contexto y se documentó el objeto de contexto.

create_configuration(*context*)

Crea el archivo de configuración `pyvenv.cfg` en el entorno.

setup_python(*context*)

Crea una copia o enlace simbólico al ejecutable Python en el entorno. En los sistemas POSIX, si se usó un ejecutable específico `python3.x`, se crearán enlaces simbólicos a `python` y `python3` apuntando a ese ejecutable, a menos que ya existan archivos con esos nombres.

setup_scripts(*context*)

Instala los scripts de activación apropiados para la plataforma en el entorno virtual.

upgrade_dependencies(*context*)

Upgrades the core venv dependency packages (currently pip [https://pypi.org/project/pip/]) in the environment. This is done by shelling out to the `pip` executable in the environment.

Added in version 3.9.

Distinto en la versión 3.12: setuptools [https://pypi.org/project/setuptools/] is no longer a core venv dependency.

post_setup(*context*)

Un método de marcador de posición que puede ser anulado en implementaciones de terceros/as para previo instalar paquetes en el entorno virtual o realizar otros pasos posteriores a la creación.

install_scripts(*context*, *path*)

This method can be called from **`setup_scripts()`** or **`post_setup()`** in subclasses to assist in installing custom scripts into the virtual environment.

path is the path to a directory that should contain subdirectories `common`, `posix`, `nt`; each containing scripts destined for the `bin` directory in the environment. The contents of `common` and the directory corresponding to **`os.name`** are copied after some text replacement of placeholders:

- `__VENV_DIR__` se sustituye por la ruta absoluta del directorio del entorno.
- `__VENV_NAME__` se sustituye por el nombre del entorno (parte final de la ruta del directorio del entorno).
- `__VENV_PROMPT__` se sustituye por el *prompt* (el nombre del entorno entre paréntesis y con un espacio posterior)
- `__VENV_BIN_NAME__` se sustituye con el nombre del directorio *bin* (ya sea `bin` o `Scripts`).
- `__VENV_PYTHON__` se sustituye con la ruta absoluta del ejecutable del entorno.

Se permite la existencia de los directorios (para cuando se está actualizando un entorno existente).

create_git_ignore_file(*context*)

Creates a `.gitignore` file within the virtual environment that causes the entire directory to be ignored by the Git source control manager.

Added in version 3.13.

Distinto en la versión 3.7.2: Windows ahora usa scripts de redireccionamiento para `python[w].exe` en lugar de copiar los propios binarios. Para 3.7.2 solamente **`setup_python()`** no hace nada a menos que se ejecute desde una compilación en el árbol de directorios fuente.

Distinto en la versión 3.7.3: Windows copia los scripts de redireccionamiento como parte de **`setup_python()`** en lugar de **`setup_scripts()`**. Este no era el caso para 3.7.2.

Cuando se usan enlaces simbólicos, los ejecutables originales se enlazan.

También hay una función de conveniencia a nivel de módulo:

venv.create(*env_dir*, *system_site_packages=False*, *clear=False*, *symlinks=False*, *with_pip=False*, *prompt=None*, *upgrade_deps=False*, ***, *scm_ignore_files=frozenset()*)

Crea un **`EnvBuilder`** con los argumentos de la palabra clave dada, y llama a su método **`create()`** con el argumento *env_dir*.

Added in version 3.3.

Distinto en la versión 3.4: Added the *with_pip* parameter

Distinto en la versión 3.6: Added the *prompt* parameter

Distinto en la versión 3.9: Added the *upgrade_deps* parameter

Distinto en la versión 3.13: Added the *scm_ignore_files* parameter

Un ejemplo de la extensión de `EnvBuilder`

El siguiente script muestra como extender **`EnvBuilder`** implementando una subclase que instala setuptools y pip en un entorno virtual creado:

```
import os
import os.path
from subprocess import Popen, PIPE
import sys
from threading import Thread
from urllib.parse import urlsplit
from urllib.request import urlretrieve
import venv
```

```
class ExtendedEnvBuilder(venv.EnvBuilder):
    """
    This builder installs setuptools and pip so that you
    easy_install other packages into the created virtual

    :param nodist: If true, setuptools and pip are not i
                   created virtual environment.
    :param nopip: If true, pip is not installed into the
                  virtual environment.
    :param progress: If setuptools or pip are installed,
                     installation can be monitored by pa
                     callable. If specified, it is calle
                     arguments: a string indicating some
                     context indicating where the string
                     The context argument can have one o
                     'main', indicating that it is calle
                     itself, and 'stdout' and 'stderr',
                     by reading lines from the output st
                     which is used to install the app.

                     If a callable is not specified, def
                     information is output to sys.stderr
    """

    def __init__(self, *args, **kwargs):
        self.nodist = kwargs.pop('nodist', False)
        self.nopip = kwargs.pop('nopip', False)
        self.progress = kwargs.pop('progress', None)
        self.verbose = kwargs.pop('verbose', False)
        super().__init__(*args, **kwargs)

    def post_setup(self, context):
        """
        Set up any packages which need to be pre-install
        virtual environment being created.

        :param context: The information for the virtual
                        creation request being processed
        """
        os.environ['VIRTUAL_ENV'] = context.env_dir
        if not self.nodist:
            self.install_setuptools(context)
        # Can't install pip without setuptools
        if not self.nopip and not self.nodist:
            self.install_pip(context)

    def reader(self, stream, context):
        """
        Read lines from a subprocess' output stream and
        callable (if specified) or write progress inform
        """
        progress = self.progress
        while True:
            s = stream.readline()
            if not s:
                break
            if progress is not None:
                progress(s, context)
            else:
                if not self.verbose:
                    sys.stderr.write('.')
                else:
                    sys.stderr.write(s.decode('utf-8'))
                sys.stderr.flush()
        stream.close()

    def install_script(self, context, name, url):
        _, _, path, _, _ = urlsplit(url)
        fn = os.path.split(path)[-1]
        binpath = context.bin_path
        distpath = os.path.join(binpath, fn)
        # Download script into the virtual environment's
        urlretrieve(url, distpath)
        progress = self.progress
        if self.verbose:
            term = '\n'
        else:
            term = ''
        if progress is not None:
            progress('Installing %s ...%s' % (name, term
        else:
            sys.stderr.write('Installing %s ...%s' % (na
            sys.stderr.flush()
        # Install in the virtual environment
        args = [context.env_exe, fn]
        p = Popen(args, stdout=PIPE, stderr=PIPE, cwd=bi
        t1 = Thread(target=self.reader, args=(p.stdout,
        t1.start()
        t2 = Thread(target=self.reader, args=(p.stderr,
        t2.start()
        p.wait()
        t1.join()
        t2.join()
        if progress is not None:
            progress('done.', 'main')
        else:
            sys.stderr.write('done.\n')
        # Clean up - no longer needed
        os.unlink(distpath)

    def install_setuptools(self, context):
        """
        Install setuptools in the virtual environment.

        :param context: The information for the virtual
                        creation request being processed
        """
        url = "https://bootstrap.pypa.io/ez_setup.py"
        self.install_script(context, 'setuptools', url)
        # clear up the setuptools archive which gets dow
        pred = lambda o: o.startswith('setuptools-') and
        files = filter(pred, os.listdir(context.bin_path
        for f in files:
            f = os.path.join(context.bin_path, f)
            os.unlink(f)

    def install_pip(self, context):
        """
        Install pip in the virtual environment.

        :param context: The information for the virtual
                        creation request being processed
        """
        url = 'https://bootstrap.pypa.io/get-pip.py'
        self.install_script(context, 'pip', url)


def main(args=None):
    import argparse

    parser = argparse.ArgumentParser(prog=__name__,
                                     description='Create
                                                 'enviro
                                                 'more t
                                                 'direct
    parser.add_argument('dirs', metavar='ENV_DIR', nargs
                        help='A directory in which to cr
                             'virtual environment.')
    parser.add_argument('--no-setuptools', default=False
                        action='store_true', dest='nodis
                        help="Don't install setuptools o
                             "virtual environment.")
    parser.add_argument('--no-pip', default=False,
                        action='store_true', dest='nopip
                        help="Don't install pip in the v
                             "environment.")
    parser.add_argument('--system-site-packages', defaul
                        action='store_true', dest='syste
                        help='Give the virtual environme
                             'system site-packages dir.'
    if os.name == 'nt':
        use_symlinks = False
    else:
        use_symlinks = True
    parser.add_argument('--symlinks', default=use_symlin
                        action='store_true', dest='symli
                        help='Try to use symlinks rather
                             'when symlinks are not the
                             'the platform.')
    parser.add_argument('--clear', default=False, actio
                        dest='clear', help='Delete the c
                                           'virtual envi
                                           'directory if
                                           'exists, befo
                                           'environment
    parser.add_argument('--upgrade', default=False, acti
                        dest='upgrade', help='Upgrade th
                                             'environmen
                                             'use this v
                                             'Python, as
                                             'has been u
                                             'in-place.'
    parser.add_argument('--verbose', default=False, acti
                        dest='verbose', help='Display th
                                             'from the s
                                             'install se
    options = parser.parse_args(args)
    if options.upgrade and options.clear:
        raise ValueError('you cannot supply --upgrade an
    builder = ExtendedEnvBuilder(system_site_packages=op
                                 clear=options.clear,
                                 symlinks=options.syml
                                 upgrade=options.upgra
                                 nodist=options.nodist
                                 nopip=options.nopip,
                                 verbose=options.verbo
    for d in options.dirs:
        builder.create(d)

if __name__ == '__main__':
```

```
rc = 1
try:
    main()
    rc = 0
except Exception as e:
    print('Error: %s' % e, file=sys.stderr)
sys.exit(rc)
```

Este script está también disponible para su descarga online [https://gist.github.com/vsajip/4673395].

`zipapp` — Manage executable Python zip archives

Added in version 3.5.

Código fuente: Lib/zipapp.py [https://github.com/python/cpython/tree/3.14/Lib/zipapp.py]

Este módulo provee herramientas para administrar la creación de archivos zip que contengan código Python, los que pueden ser ejecutados directamente por el intérprete de Python. El módulo provee tanto una Interfaz de línea de comando y una API de Python.

Ejemplo básico

El siguiente ejemplo muestra cómo la Interfaz de línea de comando puede utilizarse para crear un archivador ejecutable de un directorio que contenga código en Python. Al ponerse en funcionamiento, el archivador ejecutará la función `main` del módulo `myapp` en el archivador.

```
$ python -m zipapp myapp -m "myapp:main"
$ python myapp.pyz
<output from myapp>
```

Interfaz de línea de comando

En la ejecución como programa desde la línea de comandos, se utiliza el siguiente formato:

```
$ python -m zipapp source [options]
```

Si *source* es un directorio, se creará un archivador zip para los contenidos de *source*. Si *source* es un archivo, debería ser un archivador, y se copiará al archivador de destino (o los contenidos de su línea *shebang* se mostrarán si se especifica la opción –info).

Se aceptan las siguientes opciones:

-o <output>, --output=<output>

Escribe la salida a un archivo llamado *output*. Si esta opción no está especificada, el nombre del archivo de salida será el mismo que la entrada *source*, con la extensión `.pyz` agregada. Si se provee de un nombre de archivo explícito, se usa tal como está (por lo que una extensión `.pyz` debería incluirse si esto se requiere).

Un nombre de archivo de salida debe especificarse si *source* es un archivador (y en ese caso, *output* no debería ser igual que *source*).

-p <interpreter>, --python=<interpreter>

Agrega una línea `#!` al archivador especificando *interpreter* como comando a ejecutar. También en POSIX, convierte al archivador en ejecutable. La opción por defecto es no escribir una línea `#!`, y no hacer que el archivo sea ejecutable.

-m <mainfn>, --main=<mainfn>

Escribe un archivo `__main__.py` en el archivador que ejecuta *mainfn*. El argumento *mainfn* debería tener la forma «pkg.mod:fn», donde «pkg.mod» is un paquete/módulo en el archivador, y «fn» es un invocable (*callable*) en ese módulo. El archivo `__main__.py` ejecutará ese invocable.

`--main` no puede especificarse al copiar un archivador.

-c, --compress

Comprime los archivos con el método *deflate*, lo que reduce el tamaño del archivo de salida. Por defecto, los archivos se guardan sin comprimir en el archivador.

`--compress` no surte efecto al copiar un archivador.

Added in version 3.7.

--info

Muestra el intérprete incrustado en el archivador, para diagnósticos. En este caso cualquier otra opción se ignora, y SOURCE debe ser un archivador, no un directorio.

-h, --help

Muestra un breve mensaje sobre el modo de uso, y sale.

API de Python

El módulo define dos funciones convenientes:

zipapp.create_archive(*source*, *target=None*, *interpreter=None*, *main=None*, *filter=None*, *compressed=False*)

Crea un archivador de aplicación a partir de *source*. Este *source* (origen), puede ser cualquiera de los siguientes:

- El nombre de un directorio, o un path-like object que se refiera a un directorio, en cuyo caso un nuevo archivador de aplicación se creará a partir del contenido de dicho directorio.
- El nombre de un archivador de aplicación existente o un path-like object que refiera a dicho archivo, en cuyo caso el archivo se copiará al destino (modificándolo para reflejar el valor dado por el argumento *interpreter*. El nombre de archivo debería incluir la extensión `.pyz`, si se requiere.
- Un objeto archivo abierto para lectura en modo bytes. El contenido del archivo debería ser un archivador de aplicación. Se infiere que el objeto archivo está posicionado al comienzo del archivador.

El argumento *target* determina dónde quedará escrito el archivador resultante:

- Si es el nombre de un archivo, o un path-like object, el archivador será escrito a ese archivo.

- Si es un objeto archivo abierto, el archivador se escribirá en ese objeto archivo, el cual debe estar abierto para escritura en modo bytes.
- Si el destino (*target*) se omite (o es `None`), el origen (*source*) debe ser un directorio, y el destino será un archivo con el mismo nombre que el origen, con la extensión `.pyz` añadida.

El argumento *interpreter* especifica el nombre del intérprete Python con el que el archivador será ejecutado. Se escribe como una línea *«shebang»* al comienzo del archivador. En POSIX, el Sistema Operativo será quien lo interprete, y en Windows será gestionado por el lanzador Python. Omitir el *interpreter* tendrá como consecuencia que no se escribirá ninguna línea *shebang*. Si se especifica un intérprete y el destino (*target*) es un nombre de archivo, el bit de ejecución del archivo destino será activado.

El argumento *main* especifica el nombre de un invocable (*callable*) que se utilizará como programa principal para el archivador. Solamente se puede especificar si el origen (*source*) es un directorio que no contiene un archivo `__main__.py`. El argumento *main* debería tener la forma «pkg.module:callable», y el archivador será ejecutado importando «pkg.module» y ejecutando el *callable* sin argumentos. Es un error omitir *main* si el origen es un directorio que no contiene un archivo `__main__.py`, ya que esto resultaría en un archivador no ejecutable.

El argumento opcional *filter* especifica una función *callback* que se pasa como objeto Path, para representar el *path* (ruta) al archivo que se está añadiendo (ruta relativa al directorio origen, *source*). Debería retornar `True` si el archivo será añadido.

El argumento opcional *compressed* determina si los archivos están comprimidos. Si se define como `True`, los archivos en el archivador serán comprimidos con el método deflate.

Si se especifica un objeto archivo para *source* o *target*, es responsabilidad de quien invoca cerrarlo luego de invocar a create_archive.

Al copiar un archivador existente, los objetos archivo provistos, solamente necesitan los métodos `read` y `readline`, o bien `write`. Al crear un archivador a partir de un directorio, si el destino (*target*) es un objeto archivo, éste se pasará a la clase `zipfile.ZipFile`, y debe proveer los métodos que esa clase necesita.

Distinto en la versión 3.7: Added the *filter* and *compressed* parameters.

zipapp.get_interpreter(*archive*)

Retorna el intérprete especificado en la línea `#!` al comienzo del archivador. Si no hay línea `#!`, retorna **`None`**. El argumento *archive* (archivador), puede ser un nombre de archivo o un objeto tipo archivo abierto para lectura en modo bytes. Se supone que está al principio del archivador.

Ejemplos

Empaqueta un directorio en un archivador, y lo ejecuta.

```
$ python -m zipapp myapp
$ python myapp.pyz
<output from myapp>
```

Lo mismo puede lograrse utilizando la función **`create_archive()`**:

```
>>> import zipapp
>>> zipapp.create_archive('myapp', 'myapp.pyz')
```

Para que la aplicación sea ejecutable directamente en POSIX, especifica un intérprete.

```
$ python -m zipapp myapp -p "/usr/bin/env python"
$ ./myapp.pyz
<output from myapp>
```

Para reemplazar la línea *shebang* en un archivador existente, cree un archivador modificado, utilizando la función **`create_archive()`**:

```
>>> import zipapp
>>> zipapp.create_archive('old_archive.pyz', 'new_archiv
```

Para actualizar el archivo en el lugar, reemplaza en memoria utilizando un objeto **`BytesIO`**, y luego sobreescribe el origen (*source*). Nótese que hay un riesgo al sobreescribir un archivo en el lugar, ya que un error resultará en la pérdida del archivo original. Este código no ofrece protección contra este tipo de errores, pero los códigos de producción deberían hacerlo. Además, este método solamente funcionará si el archivador cabe en la memoria:

```
>>> import zipapp
>>> import io
>>> temp = io.BytesIO()
>>> zipapp.create_archive('myapp.pyz', temp, '/usr/bin/p
>>> with open('myapp.pyz', 'wb') as f:
>>>     f.write(temp.getvalue())
```

Especificar el intérprete

Nótese que si se especifica el intérprete y luego se distribuye el archivador de aplicación, es necesario asegurarse de que el intérprete utilizado es portable. El lanzador Python para Windows soporta las formas más comunes de líneas `#!` POSIX, pero hay otras cuestiones a considerar:

- Si se utiliza «/usr/bin/env python» (u otras formas del comando «python», como «/usr/bin/python»), es necesario considerar que los usuarios quizá tengan tanto Python2 como Python3 como versión por defecto, y el código debe escribirse bajo ambas versiones.
- Si se utiliza una versión específica, por ejemplo «/usr/bin/env python3», la aplicación no funcionará para los usuarios que no tengan esa versión. (Esta puede ser la opción deseada si no se hecho el código compatible con Python 2).
- No hay manera de decir «python X.Y o posterior», así que se debe ser cuidadoso al utilizar una versión exacta, tal como «/

usr/bin/env python3.4», ya que será necesario cambiar la línea *shebang* para usuarios de Python 3.5, por ejemplo.

Normalmente, se debería utilizar «/usr/bin/env python2» o «/usr/bin/env python3», según si el código está escrito para Python 2 ó 3.

Creando aplicaciones independientes con zipapp

Using the **`zipapp`** module, it is possible to create self-contained Python programs, which can be distributed to end users who only need to have a suitable version of Python installed on their system. The key to doing this is to bundle all of the application's dependencies into the archive, along with the application code.

Los pasos para crear un archivador independiente son los siguientes:

1. Crea tu aplicación en un directorio normalmente, tal que tengas una directorio `myapp` que contenga un archivo `__main__.py`, y cualquier código extra de la aplicación.

2. Instala todas las dependencias de la aplicación en el directorio `myapp`, usando pip:

   ```
   $ python -m pip install -r requirements.txt --targe
   ```

   (se supone que tienes los requisitos de tu proyecto en un archivo `requirements.txt`, de lo contrario, puedes listar manualmente las dependencias en la línea de comandos de pip).

3. Empaqueta la aplicación utilizando:

   ```
   $ python -m zipapp -p "interpreter" myapp
   ```

Esto producirá un ejecutable independiente, que puede ser ejecutado en cualquier máquina que disponga del intérprete apropiado. Véase Especificar el intérprete para más detalles. Puede enviarse a los usuarios como un solo archivo.

En Unix, el archivo `myapp.pyz` será ejecutable tal como está. Puede ser renombrado, para quitar la extensión `.pyz` si se prefiere un nombre de comando «simple». En Windows, el archivo `myapp.pyz[w]` es ejecutable, ya que el intérprete Python registra las extensiones `.pyz` y `pyzw` al ser instalado.

Advertencias

Si la aplicación depende de un paquete que incluye una extensión C, ese paquete no puede ser ejecutado desde un archivo zip (esta es una limitación del sistema operativo, dado que el código ejecutable debe estar presente en el sistema de archivos para que el *loader* del SO lo pueda cargar). En este caso, se puede excluir la dependencia del archivo zip, y requerir que los usuarios la tengan instalada, o bien distribuirla conjuntamente con el archivo zip, y agregar código al `__main__.py` para que incluya el directorio que contiene el módulo descomprimido en `sys.path`. En este caso, será necesario asegurarse de distribuir los binarios adecuados para la/s arquitectura/s a las que esté destinada la aplicación (y potencialmente elegir la versión correcta para agregar a `sys.path` en tiempo de ejecución, basándose en la máquina del usuario).

El formato de archivado Zip de aplicaciones Python

Python puede ejecutar archivadores zip que contengan un archivo `__main__.py` desde la versión 2.6. Para que sea ejecutada por Python, basta con que una aplicación de archivador sea un archivo zip estándar que contenga un archivo `__main__.py`, el cual será ejecutado como punto de entrada para la aplicación. Como es usual para cualquier script Python, el elemento padre del script (en este caso, el archivo zip), será ubicado en `sys.path`, por lo que pueden importarse otros módulos desde el archivo zip.

El formato de archivo zip, permite que se antepongan al archivo datos arbitrarios. El formato de aplicación zip utiliza esta capacidad para anteponer una línea «*shebang*» POSIX estándar al archivo (`#!/ruta/al/interprete`).

Formalmente, el Formato de archivado Zip de aplicaciones Python es:

1. An optional shebang line, containing the characters `b'#!'` followed by an interpreter name, and then a newline (`b'\n'`) character. The interpreter name can be anything acceptable to the OS «shebang» processing, or the Python launcher on Windows. The interpreter should be encoded in UTF-8 on Windows, and in **`sys.getfilesystemencoding()`** on POSIX.
2. Los datos estándares de archivo zip, tal como se generan en el módulo **`zipfile`**. El contenido del archivo zip *debe* incluir un archivo llamado `__main__.py` (que debe estar en la «raíz» del archivo zip, es decir, no puede estar en un subdirectorio). El los datos del archivo zip pueden estar comprimidos o no.

Si un archivador de aplicación tiene una línea de shebang, puede tener el bit de ejecución activado en los sistemas POSIX, para permitir que sea ejecutado directamente.

No se requiere que las herramientas de este módulo sean las que se utilicen para crear archivadores de aplicación. El módulo es útil, pero cualquier archivo que esté en el formato descripto anteriormente es aceptable para Python, no importa cómo haya sido creado.

Servicios en tiempo de ejecución de Python

Los módulos descritos en este capítulo proporcionan una amplia gama de servicios relacionados con el intérprete de Python y su interacción con su entorno. Esta es una descripción general:

- **`sys`** — System-specific parameters and functions
- **`sys.monitoring`** — Execution event monitoring
    - Identificadores de herramientas
        - Registro y uso de herramientas
    - Eventos
        - Eventos locales
        - Deprecated event
        - Eventos auxiliares
        - Otros eventos
        - El evento STOP_ITERATION
    - Activar y desactivar eventos
        - Configuración de eventos globalmente
        - Eventos por objeto de código
        - Deshabilitando eventos
    - Registrando funciones de retrollamada
        - Argumentos de la función de retrollamada
- **`sysconfig`** — Provide access to Python's configuration information
    - Variables de configuración
    - Rutas de instalación

    - User scheme
        - **`posix_user`**
        - **`nt_user`**
        - **`osx_framework_user`**
    - Home scheme
        - **`posix_home`**
    - Prefix scheme
        - **`posix_prefix`**
        - **`nt`**
    - Installation path functions
    - Otras funciones
    - Command-line usage
- **`builtins`** — Built-in objects
- **`__main__`** — Top-level code environment
    - **`__name__ == '__main__'`**
        - ¿Qué es el «entorno de código de máximo nivel»?
        - Uso idiomático
        - Consideraciones de empaquetado
    - **`__main__.py`** en paquetes de Python
        - Uso idiomático
    - **`import __main__`**
- **`warnings`** — Warning control
    - Categorías de advertencia
    - El filtro de advertencias
        - Repeated Warning Suppression Criteria
        - Descripción de los filtros de advertencia
        - Filtro de advertencia predeterminado

        - Anulando el filtro por defecto

    - Eliminación temporal de las advertencias
    - Advertencias de prueba
    - Actualización del código para las nuevas versiones de las dependencias
    - Funciones disponibles
    - Gestores de contexto disponibles
    - Concurrent safety of Context Managers

- `dataclasses` — Data Classes

    - Contenidos del módulo
    - Procesamiento posterior a la inicialización
    - Variables de clase
    - Variable de solo inicialización
    - Instancias congeladas
    - Herencia
    - Re-ordering of keyword-only parameters in `__init__()`
    - Funciones fábrica por defecto
    - Valores por defecto mutables
    - Campos tipo descriptor

- `contextlib` — Utilidades para declaraciones de contexto `with`

    - Utilidades
    - Ejemplos y recetas

        - Apoyando un número variable de gestores de contexto
        - Capturando excepciones de los métodos `__enter__`
        - Limpieza en una implementación `__enter__`
        - Reemplazar cualquier uso de `try-finally` y marcar variables
        - Usar un gestor de contexto como decorador de funciones

    - Gestores de contexto de uso único, reutilizables y

    reentrantes
        - Gestores contextuales reentrantes
        - Gestores contextuales reutilizables

- `abc` — Abstract Base Classes
- `atexit` — Exit handlers
    - `atexit` Example
- `traceback` — Print or retrieve a stack traceback
    - Module-Level Functions
    - `TracebackException` Objects
    - `StackSummary` Objects
    - `FrameSummary` Objects
    - Examples of Using the Module-Level Functions
    - Examples of Using `TracebackException`
- `__future__` — Future statement definitions
    - Module Contents
- `gc` — Garbage Collector interface
- `inspect` — Inspect live objects
    - Tipos y miembros
    - Recuperar el código fuente
    - Introspección de los invocables con el objeto Signature
    - Clases y funciones
    - La pila del interprete
    - Obteniendo atributos estáticamente
    - Current State of Generators, Coroutines, and Asynchronous Generators
    - Objetos de código Bit Flags
    - Buffer flags
    - Command-line interface
- `annotationlib` — Functionality for introspecting annotations
    - Annotation semantics

    - Classes
    - Functions
    - Recipes
        - Using annotations in a metaclass
    - Limitations of the **`STRING`** format
    - Limitations of the **`FORWARDREF`** format
    - Security implications of introspecting annotations
- **`site`** — Site-specific configuration hook
    - **`sitecustomize`**
    - **`usercustomize`**
    - Configuración de *Readline*
    - Contenido del módulo
    - Command-line interface

Ver también

- Consulte el módulo **`concurrent.interpreters`**, que de manera similar expone funcionalidades centrales del entorno de ejecución.

`sys` — System-specific parameters and functions

This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available. Unless explicitly noted otherwise, all variables are read-only.

sys.abiflags

En los sistemas POSIX donde Python se construyó con el script estándar `configure`, este contiene los indicadores ABI según lo especificado por **PEP 3149** [https://peps.python.org/pep-3149/].

Added in version 3.2.

Distinto en la versión 3.8: Los flags por defecto se convirtieron en una cadena de caracteres vacía (el flag `m` para pymalloc ha sido eliminado).

Availability: Unix.

sys.addaudithook(*hook*)

Agrega el *hook* invocable a la lista de enlaces de inspección activos para el (sub)intérprete actual.

Cuando se genera un evento de auditoría a través de la función **`sys.audit()`**, cada hook será llamado en el orden en que se agregó con el nombre del evento y la tupla de argumentos. Los hooks nativos agregados por **`PySys_AddAuditHook()`** se llaman primero, seguidos de los hooks agregados en el (sub)intérprete actual. Los hooks pueden registrar un evento, lanzar una excepción para abortar la operación, o terminar el proceso completamente.

Tenga en cuenta que los ganchos de auditoría son principalmente para recopilar información sobre acciones internas o no observables, ya sea por Python o bibliotecas escritas en Python. No son adecuados para implementar una «caja de arena» (sandbox). En particular, el código malicioso puede desactivar o eludir trivialmente los hooks añadidos mediante esta función. Como mínimo, cualquier hook sensible a la seguridad debe añadirse utilizando la API de C **`PySys_AddAuditHook()`** antes de inicializar el tiempo de ejecución, y cualquier módulo que permita la modificación arbitraria de la memoria (como **`ctypes`**) debe ser completamente eliminado o supervisado de cerca.

Llamar **`sys.addaudithook()`** lanzará por si mismo un evento de auditoria llamado `sys.addaudithook` sin argumentos. Si uno de los hooks existentes lanza un excepción derivada de **`RuntimeError`**, el nuevo hook no será agregado y la excepción será suprimida. Como resultado, los que llaman no podrán asumir que el hook a sido agregado a menos que ellos controlen todos los hooks existentes.

Consulte la tabla de eventos de auditoría para todos los eventos lanzados por CPython, y **PEP 578** [https://peps.python.org/pep-0578/] para discusión del diseño original.

Added in version 3.8.

Distinto en la versión 3.8.1: Las excepciones derivadas de **`Exception`** pero no **`RuntimeError`** ya no se suprimen.

Cuando el rastreo está habilitado (ver **`settrace()`**), los ganchos de Python solo se rastrean si el invocable tiene un miembro `__cantrace__` que se establece en un valor verdadero. De lo contrario, las funciones de seguimiento omitirán el enlace.

sys.argv

La lista de argumentos de la línea de comandos pasados a un script de Python. `argv[0]` es el nombre del script (depende del sistema operativo si se trata de una ruta completa o no).

Si el comando se ejecutó usando la opción `-c` de la línea de comandos del intérprete, `argv[0]` se establece en la cadena de caracteres `'-c'`. Si no se pasó ningún nombre de secuencia de comandos al intérprete de Python, `argv[0]` es la cadena de caracteres vacía.

Para recorrer la entrada estándar, o la lista de archivos dada en la línea de comando, vea el módulo `fileinput`.

También ver `sys.orig_argv`.

Nota

En Unix, los argumentos de la línea de comandos se pasan por bytes desde el sistema operativo. Python los decodifica con la codificación del sistema de archivos y el controlador de errores «surrogateescape». Cuando necesite bytes originales, puede obtenerlos mediante `[os.fsencode(arg) for arg in sys.argv]`.

sys.audit(*event*, **args*)

Activar un evento de auditoría y activar cualquier hook de auditoría activo. *event* es una cadena que identifica el evento, y *args* puede contener argumentos opcionales con más información sobre el evento. El número y los tipos de argumentos para un evento determinado se consideran una API pública y estable y no deberían modificarse entre versiones.

Por ejemplo, un evento de auditoría se llama `os.chdir`. Este evento tiene un argumento llamado *path* que contendrá el nuevo directorio de trabajo solicitado.

`sys.audit()` llamará a los ganchos de auditoría existentes, pasando el nombre del evento y los argumentos, y volverá a lanzar la primera excepción de cualquier hook. En general, si se lanza una excepción, no debería ser manejada y el proceso debería terminar lo más rápidamente posible. Esto permite que las implementaciones de los hook decidan cómo responder a determinados eventos: pueden limitarse a registrar el evento o abortar la operación lanzando una excepción.

Los ganchos se agregan usando las funciones `sys.addaudithook()` o `PySys_AddAuditHook()`.

El equivalente nativo de esta función es `PySys_Audit()`. Se prefiere usar la función nativa cuando sea posible.

Consulte la tabla de eventos de auditoría para todos los eventos lanzados por CPython.

Added in version 3.8.

sys.base_exec_prefix

Equivalent to `exec_prefix`, but referring to the base Python installation.

When running under Virtual Environments, `exec_prefix` gets overwritten to the virtual environment prefix. **`base_exec_prefix`**, conversely, does not change, and always points to the base Python installation. Refer to Virtual Environments for more information.

Added in version 3.3.

sys.base_prefix

Equivalent to `prefix`, but referring to the base Python installation.

When running under virtual environment, `prefix` gets overwritten to the virtual environment prefix. **`base_prefix`**, conversely, does not change, and always points to the base Python installation. Refer to Virtual Environments for more information.

Added in version 3.3.

sys.byteorder

Un indicador del orden de bytes nativo. Esto tendrá el valor `'big'` en las plataformas big-endian (el byte más significativo primero) y `'little'` en las plataformas little-endian (el byte menos significativo primero).

sys.builtin_module_names

Una tupla de cadenas de caracteres que contiene los nombres de todos los módulos que se compilan en este intérprete de Python. (Esta información no está disponible de ninguna otra manera — `modules.keys()` solo enumera los módulos importados.)

Consulte también la lista **`sys.stdlib_module_names`**.

sys.call_tracing(*func*, *args*)

Call `func(*args)`, while tracing is enabled. The tracing state is saved, and restored afterwards. This is intended to be called from a debugger from a checkpoint, to recursively debug or profile some other code.

Tracing is suspended while calling a tracing function set by **`settrace()`** or **`setprofile()`** to avoid infinite recursion. **`call_tracing()`** enables explicit recursion of the tracing function.

sys.copyright

Una cadena de caracteres que contiene los derechos de autor pertenecientes al intérprete de Python.

sys._clear_type_cache()

Borre la caché de tipo interno. La caché de tipos se utiliza para acelerar las búsquedas de métodos y atributos. Utilice la función *solo* para eliminar referencias innecesarias durante la depuración de fugas de referencia.

Esta función debe usarse solo para fines internos y especializados.

Obsoleto desde la versión 3.13: Use the more general `_clear_internal_caches()` function instead.

sys._clear_internal_caches()

Clear all internal performance-related caches. Use this function *only* to release unnecessary references and memory blocks when hunting for leaks.

Added in version 3.13.

sys._current_frames()

Retorna un diccionario que asigna el identificador de cada subproceso al marco de pila superior actualmente activo en ese subproceso en el momento en que se llama a la función. Tenga en cuenta que las funciones en el módulo `traceback` pueden construir la pila de llamadas dado tal marco.

Esto es más útil para depurar *deadlock*: esta función no requiere la cooperación de los subprocesos con *deadlock*, y las pilas de llamadas de dichos subprocesos se congelan mientras permanezcan en *deadlock*. El marco retornado para un subproceso no en *deadlock* puede no tener relación con la actividad actual de ese subproceso en el momento en que el código de llamada examina el marco.

Esta función debe usarse solo para fines internos y especializados.

Lanza un auditing event `sys._current_frames` sin argumentos.

sys._current_exceptions()

Retorna un diccionario que mapea el identificador de cada hilo a la excepción mas alta actualmente activa en ese hilo en el tiempo que la función fue llamada. Si un hilo no esta manejando una excepción en el momento, no se incluye en el resultado del diccionario.

Esto es más útil para la elaboración de perfiles estadísticos.

Esta función debe usarse solo para fines internos y especializados.

Lanza un auditing event `sys._current_exceptions` sin argumentos.

Distinto en la versión 3.12: Cada valor del diccionario es ahora una única instancia de excepción, en lugar de una tripleta como la devuelta por `sys.exc_info()`.

sys.breakpointhook()

Esta función de gancho es llamada por built-in **`breakpoint()`**. De forma predeterminada, lo coloca en el depurador **`pdb`**, pero se puede configurar en cualquier otra función para que pueda elegir qué depurador se utiliza.

La firma de esta función depende de lo que llame. Por ejemplo, el enlace predeterminado (por ejemplo, `pdb.set_trace()`) no espera argumentos, pero puede vincularlo a una función que espera argumentos adicionales (posicional o palabra clave). La función incorporada `breakpoint()` pasa sus `*args` y `**kws` directamente. Lo que sea que retorne `breakpointhooks()` se retorna desde `breakpoint()`.

La implementación predeterminada consulta primero la variable de entorno **`PYTHONBREAKPOINT`**. Si se establece en `"0"`, esta función vuelve inmediatamente; es decir, no es una operación. Si la variable de entorno no se establece, o se establece en la cadena vacía, se llama a `pdb.set_trace()`. De lo contrario, esta variable debería nombrar una función para ejecutar, utilizando la nomenclatura de importación con puntos de Python, por ejemplo `package.subpackage.module.function`. En este caso, se importaría `package.subpackage.module` y el módulo resultante debe tener un invocable llamado `function()`. Esto se ejecuta, pasando `*args` y `**kws`, y lo que sea que `function()` retorna, `sys.breakpointhook()` retorna a la función **`breakpoint()`**.

Tenga en cuenta que si algo sale mal al importar el invocable nombrado por **`PYTHONBREAKPOINT`**, se informa un **`RuntimeWarning`** y se ignora el punto de interrupción (*breakpoint*).

También tenga en cuenta que si `sys.breakpointhook()` se anula mediante programación, **`PYTHONBREAKPOINT`** *no* se consulta.

Added in version 3.7.

sys._debugmallocstats()

Imprime información de bajo nivel para stderr sobre el estado del asignador de memoria de CPython.

Si Python es built in debug mode (**`configure --with-pydebug option`**), también realiza algunas costosas comprobaciones de coherencia interna.

Added in version 3.3.

Esta función es específica de CPython. El formato exacto de salida no se define aquí y puede cambiar.

sys.dllhandle

Número entero que especifica el identificador de la DLL de Python.

Availability: Windows.

sys.displayhook(*value*)

Si *value* no es `None`, esta función imprime `repr(value)` en `sys.stdout` y guarda *value* en `builtins._`. Si `repr(value)` no se puede codificar en `sys.stdout.encoding` con el controlador de errores `sys.stdout.errors` (que probablemente sea `'strict'`), codifíquelo para `sys.stdout.encoding` con el controlador de errores `'backslashreplace'`.

Se llama a `sys.displayhook` como resultado de evaluar un expression ingresado en una sesión interactiva de Python. La visualización de estos valores se puede personalizar asignando otra función de un argumento a `sys.displayhook`.

Pseudo-código:

```
def displayhook(value):
    if value is None:
        return
    # Set '_' to None to avoid recursion
    builtins._ = None
    text = repr(value)
    try:
        sys.stdout.write(text)
    except UnicodeEncodeError:
        bytes = text.encode(sys.stdout.encoding, 'b
        if hasattr(sys.stdout, 'buffer'):
            sys.stdout.buffer.write(bytes)
        else:
            text = bytes.decode(sys.stdout.encoding
            sys.stdout.write(text)
    sys.stdout.write("\n")
    builtins._ = value
```

Distinto en la versión 3.2: Usa el manejador de error `'backslashreplace'` en **`UnicodeEncodeError`**.

sys.dont_write_bytecode

Si esto es cierto, Python no intentará escribir archivos `.pyc` en la importación de módulos fuente. Este valor se establece inicialmente en `True` o `False` según la opción **`-B`** de la línea de comando y la variable de entorno **`PYTHONDONTWRITEBYTECODE`**, pero puede configurarlo usted mismo para controlar el código de bytes generación de archivos.

sys._emscripten_info

Un named tuple que contiene información sobre el medio ambiente en la plataforma *wasm32-emscripten*. La tupla nombrada es provisional y puede cambiar en el futuro.

_emscripten_info.emscripten_version

Versión de Emscripten como tupla de enteros (mayor, menor, micro), por ejemplo `(3, 1, 8)`.

_emscripten_info.runtime

Cadena de tiempo de ejecución, por ejemplo, agente de usuario del navegador, `'Node.js v14.18.2'`, o `'UNKNOWN'`.

_emscripten_info.pthreads

`True` si Python está compilado con soporte para pthreads de Emscripten.

_emscripten_info.shared_memory

`True` si Python está compilado con soporte de memoria compartida.

Availability: Emscripten.

Added in version 3.11.

sys.pycache_prefix

Si se establece esto (no `None`), Python escribirá archivos `.pyc` de caché de código de bytes en (y los leerá desde) un árbol de directorios paralelo con raíz en este directorio, en lugar de los directorios `__pycache__` en el árbol de código fuente. Se ignorarán todos los directorios `__pycache__` en el árbol del código fuente y se escribirán nuevos archivos `.pyc` dentro del prefijo de pycache. Por lo tanto, si usa **`compileall`** como paso previo a la compilación, debe asegurarse de ejecutarlo con el mismo prefijo de pycache (si corresponde) que usará en tiempo de ejecución.

Una ruta relativa se interpreta en relación con el directorio de directorio actual.

Este valor se establece inicialmente en función del valor de la opción de línea de comandos **`-X`** `pycache_prefix=PATH` o la variable de entorno **`PYTHONPYCACHEPREFIX`** (la línea de comandos tiene prioridad). Si no se establece ninguno, es `None`.

Added in version 3.8.

sys.excepthook(*type*, *value*, *traceback*)

Esta función imprime un rastreo y una excepción dados a `sys.stderr`.

Cuando se genera y no se detecta una excepción distinta de **`SystemExit`**, el intérprete llama a `sys.excepthook` con tres argumentos: la clase de excepción, la instancia de excepción y un objeto de rastreo. En una sesión interactiva, esto sucede justo antes de que se devuelva el control al indicador; en un programa Python esto sucede justo antes de que salga el programa. El manejo de dichas excepciones de nivel superior se puede personalizar asignando otra función de tres argumentos a `sys.excepthook`.

Genere un evento de auditoría `sys.excepthook` con argumentos `hook`, `type`, `value`, `traceback` cuando se produzca una excepción no detectada. Si no se ha colocado ningún gancho, `hook` puede ser `None`. Si algún gancho (*hook*) lanza una excepción derivada de **`RuntimeError`**, la llamada al gancho será suprimida. De lo contrario, se informará que la excepción del gancho de auditoría no se puede evaluar y se llamará a `sys.excepthook`.

Ver también

La función **`sys.unraisablehook()`** maneja las excepciones que no se pueden evaluar y la función **`threading.excepthook()`** maneja la excepción lanzada por **`threading.Thread.run()`**.

sys.__breakpointhook__

sys.__displayhook__
sys.__excepthook__
sys.__unraisablehook__

Estos objetos contienen los valores originales de `breakpointhook`, `displayhook`, `excepthook` y `unraisablehook` al inicio del programa. Se guardan para que `breakpointhook`, `displayhook` y `excepthook`, `unraisablehook` se puedan restaurar en caso de que sean reemplazados por objetos rotos o alternativos.

Added in version 3.7: __breakpointhook__

Added in version 3.8: __unraisablehook__

sys.exception()

Esta función, cuando se llama mientras se ejecuta un manejador de excepciones (como una cláusula `except` o `except*`), retorna la instancia de la excepción que fue capturada por este manejador. Cuando los manejadores de excepciones están anidados unos dentro de otros, sólo la excepción manejada por el manejador más interno es accesible.

Si no se está ejecutando ningún manejador de excepciones, esta función retorna `None`.

Added in version 3.11.

sys.exc_info()

Esta función retorna la representación de estilo antiguo de la excepción manejada. Si se maneja una excepción `e` (por lo que **`exception()`** retornaría `e`), **`exc_info()`** retorna la tupla `(type(e), e, e.__traceback__)`. Es decir, una tupla que contiene el tipo de la excepción (una subclase de **`BaseException`**), la propia excepción, y un objeto objeto traceback que suele encapsular la pila de llamadas en el punto en el que se produjo la última excepción.

Si no se está manejando ninguna excepción en ninguna parte de la pila, esta función retorna una tupla que contiene tres valores `None`.

Distinto en la versión 3.11: Los campos `type` y `traceback` ahora se derivan del `value` (la instancia de excepción), de modo que cuando se modifica una excepción mientras se maneja, los cambios se reflejan en los resultados de las subsiguientes llamadas a **`exc_info()`**.

sys.exec_prefix

Una cadena de caracteres que proporciona el prefijo de directorio específico del sitio donde están instalados los archivos Python dependientes de la plataforma; de forma predeterminada, también es `'/usr/local'`. Esto se puede configurar en el momento de la compilación con el argumento `--exec-prefix` del script **configure**. Específicamente, todos los archivos de configuración (por ejemplo, el archivo de encabezado `pyconfig.h`) se instalan en el directorio `exec_prefix/lib/pythonXY/config`, y los módulos de la biblioteca compartida se instalan en `exec_prefix/lib/pythonXY/lib-dynload`, donde *XY* es el número de versión de Python, por ejemplo,``3.2``.

Nota

If a virtual environment is in effect, this **`exec_prefix`** will point to the virtual environment. The value for the Python installation will still be available, via **`base_exec_prefix`**. Refer to Virtual Environments for more information.

Distinto en la versión 3.14: When running under a virtual environment, **`prefix`** and **`exec_prefix`** are now set to the virtual environment prefix by the path initialization, instead of **`site`**. This means that **`prefix`** and **`exec_prefix`** always point to the virtual environment, even when **`site`** is disabled (**`-S`**).

sys.executable

Una cadena de caracteres que proporciona la ruta absoluta del binario ejecutable para el intérprete de Python, en sistemas donde esto tiene sentido. Si Python no puede recuperar la ruta real a su ejecutable, **`sys.executable`** será una cadena de caracteres vacía o `None`.

sys.exit([*arg*])

Genera una excepción **`SystemExit`**, indicando la intención de salir del intérprete.

The optional argument *arg* can be an integer giving the exit status (defaulting to zero), or another type of object. If it is an integer, zero is considered «successful termination» and any nonzero value is considered «abnormal termination» by shells and the like. Most systems require it to be in the range 0–127, and produce undefined results otherwise. Some systems have a convention for assigning specific meanings to specific exit codes, but these are generally underdeveloped; Unix programs generally use 2 for command line syntax errors and 1 for all other kinds of errors. If another type of object is passed, `None` is equivalent to passing zero, and any other object is printed to **`stderr`** and results in an exit code of 1. In particular, `sys.exit("some error message")` is a quick way to exit a program when an error occurs.

Dado que **`exit()`** en última instancia «sólo» genera una excepción, sólo saldrá del proceso cuando se le llame desde el hilo principal y la excepción no se interceptará. Se respetan las acciones de limpieza especificadas por las cláusulas finalmente de las declaraciones **`try`** y es posible interceptar el intento de salida en un nivel externo.

Distinto en la versión 3.6: Si se produce un error en la limpieza después de que el intérprete de Python haya detectado **`SystemSalir`** (como un error al vaciar los datos almacenados en el búfer en los flujos estándar), el estado de salida cambia a 120.

sys.flags

The named tuple *flags* exposes the status of command line flags. Flags should only be accessed only by name and not by index. The attributes are read only.

flags.debug	**-d**
flags.inspect	**-i**
flags.interactive	**-i**
flags.isolated	**-I**
flags.optimize	**-O** o **-OO**
flags.dont_write_bytecode	**-B**
flags.no_user_site	**-s**
flags.no_site	**-S**
flags.ignore_environment	**-E**
flags.verbose	**-v**
flags.bytes_warning	**-b**
flags.quiet	**-q**
flags.hash_randomization	**-R**
flags.dev_mode	**-X dev** (Modo de desarrollo de Python)
flags.utf8_mode	**-X utf8**
flags.safe_path	**-P**
flags.int_max_str_digits	**-X int_max_str_digits** (integer string conversion length limitation)
flags.warn_default_encoding	**-X warn_default_encoding**
flags.gil	**-X gil** and **PYTHON_GIL**
flags.thread_inherit_context	**-X thread_inherit_context** and **PYTHON_THREAD_INHERIT_CONTEXT**
flags.context_aware_warnings	**-X context_aware_warnings** and **PYTHON_CONTEXT_AWARE_WARNINGS**

Distinto en la versión 3.2: Agregado el atributo `quiet` para el nuevo flag **-q**.

Added in version 3.2.3: El atributo `hash_randomization`.

Distinto en la versión 3.3: Eliminado el atributo obsoleto `division_warning`.

Distinto en la versión 3.4: Agregado el atributo `isolated` para el flag **-I** `isolated`.

Distinto en la versión 3.7: Se agregó el atributo `dev_mode` para el nuevo Modo de Desarrollo de Python y el atributo `utf8_mode` para la bandera **`-X`** `utf8`.

Distinto en la versión 3.10: Se agregó el atributo `warn_default_encoding` para el indicador **`-X`** `warn_default_encoding`.

Distinto en la versión 3.11: Se agregó el atributo `safe_path` para la opción **`-P`**.

Distinto en la versión 3.11: Se agregó el atributo `int_max_str_digits`.

Distinto en la versión 3.13: Added the `gil` attribute.

Distinto en la versión 3.14: Added the `thread_inherit_context` attribute.

Distinto en la versión 3.14: Added the `context_aware_warnings` attribute.

sys.float_info

Un named tuple que contiene información sobre el tipo de flotante. Contiene información de bajo nivel sobre la precisión y la representación interna. Los valores corresponden a las diversas constantes de coma flotante definidas en el archivo de encabezado estándar `float.h` para el lenguaje de programación "C"; consulte la sección 5.2.4.2.2 de la norma *ISO/IEC* C de 1999 [C99], "Características de los tipos flotantes", para obtener más detalles.

atributo	macro float.h	explicación
float_info.epsilon	**DBL_EPSILON**	diferencia entre 1,0 y el valor mínimo mayor que 1,0 que se puede representar como un flotante. Vea también **`math.ulp()`**.
float_info.dig	**DBL_DIG**	El número máximo de dígitos decimales que se pueden representar fielmente en un flotante; vea abajo.
float_info.mant_dig	**DBL_MANT_DIG**	Precisión del flotador: el número de dígitos en base `radix` en el significado de un flotador.
float_info.max	**DBL_MAX**	El flotante finito positivo máximo representable.

float_info.max_exp	**DBL_MAX_EXP**	máximo entero *e* tal que `radix**(e-1)` es un flotante finito representable.
float_info.max_10_exp	**DBL_MAX_10_EXP**	máximo entero *e* tal que `10**e` esté en el rango de flotadores finitos representables.
float_info.min	**DBL_MIN**	el flotante *normalized* positivo mínimo representable. Usa **`math.ulp(0.0)`** para obtener el menor flotante positivo *denormalizado* representable.
float_info.min_exp	**DBL_MIN_EXP**	mínimo entero *e* tal que `radix**(e-1)` sea un flotante normalizado.
float_info.min_10_exp	**DBL_MIN_10_EXP**	mínimo entero *e* tal que `10**e` sea un flotante normalizado.
float_info.radix	**FLT_RADIX**	base de la representación del exponente.
float_info.rounds	**FLT_ROUNDS**	Un número entero que representa el modo de redondeo para la aritmética de punto flotante. Esto refleja el valor de la macro **`FLT_ROUNDS`** del sistema en el momento de inicio del intérprete: • `-1`: indeterminable • `0`: toward zero • `1`: to nearest • `2`: toward positive infinity • `3`: toward negative infinity Todos los demás valores de **`FLT_ROUNDS`** caracterizan el comportamiento de redondeo definido por la implementación.

Atributos de **`float_info`** named tuple

El atributo **`sys.float_info.dig`** necesita más explicación. Si `s` es cualquier cadena que representa un número decimal con como máximo **`sys.float_info.dig`** dígitos significativos, entonces al convertir `s` a un flotante y viceversa se recuperará una cadena que representa el mismo valor decimal:

```
>>> import sys
>>> sys.float_info.dig
15
>>> s = '3.14159265358979'    # decimal string with
>>> format(float(s), '.15g')  # convert to float an
'3.14159265358979'
```

Pero para cadenas con más de `sys.float_info.dig` dígitos significativos, esto no siempre es cierto:

```
>>> s = '9876543211234567'    # 16 significant digi
>>> format(float(s), '.16g')  # conversion changes
'9876543211234568'
```

sys.float_repr_style

Una cadena que indica cómo se comporta la función `repr()` para los flotantes. Si la cadena tiene el valor `'short'`, entonces para un flotante finito `x`, `repr(x)` tiene como objetivo producir una cadena corta con la propiedad de que `float(repr(x)) == x`. Este es el comportamiento habitual en Python 3.1 y posteriores. De lo contrario, `float_repr_style` tiene el valor `'legacy'` y `repr(x)` se comporta de la misma manera que en las versiones de Python anteriores a la 3.1.

Added in version 3.1.

sys.getallocatedblocks()

Return the number of memory blocks currently allocated by the interpreter, regardless of their size. This function is mainly useful for tracking and debugging memory leaks. Because of the interpreter's internal caches, the result can vary from call to call; you may have to call `_clear_internal_caches()` and `gc.collect()` to get more predictable results.

If a Python build or implementation cannot reasonably compute this information, **`getallocatedblocks()`** is allowed to return 0 instead.

Added in version 3.4.

sys.getunicodeinternedsize()

Devuelve el número de objetos unicode que han sido internados.

Added in version 3.12.

sys.getandroidapilevel()

Return the build-time API level of Android as an integer. This represents the minimum version of Android this build of Python can run on. For runtime version information, see **`platform.android_ver()`**.

Availability: Android.

Added in version 3.7.

sys.getdefaultencoding()

Return `'utf-8'`. This is the name of the default string encoding, used in methods like **`str.encode()`**.

sys.getdlopenflags()

Devuelve el valor actual de los indicadores que se utilizan para las llamadas **`dlopen()`**. Los nombres simbólicos para los valores de las banderas se pueden encontrar en el módulo **`os`** (constantes `RTLD_xxx`, por ejemplo, **`os.RTLD_LAZY`**).

Availability: Unix.

sys.getfilesystemencoding()

Obtenga filesystem encoding: la codificación utilizada con filesystem error handler para convertir entre nombres de archivos Unicode y nombres de archivos en bytes. El controlador de errores del sistema de archivos se devuelve desde **`getfilesystemencodeerrors()`**.

Para la mejor compatibilidad, str debe ser usado por nombres de archivos en todos los casos, aunque representar nombres de archivos en bytes también es soportado. Funciones que acepta o retornan nombres de archivos deben soportar tanto str como bytes e internamente convertir el valor a representación preferida por el sistema.

`os.fsencode()` y **`os.fsdecode()`** deben usarse para garantizar que se utilizan la codificación correcta y el modo de errores.

El codificación del sistema de archivos y manejo de errores está configurado en el inicio de Python por la función **`PyConfig_Read()`**: ver **`filesystem_encoding`** y **`filesystem_errors`** que son miembros de **`PyConfig`**.

Distinto en la versión 3.2: el resultado de **`getfilesystemencoding()`** ya no puede ser `None`.

Distinto en la versión 3.6: Ya no se garantiza que Windows retorne `'mbcs'`. Consulte **PEP 529** [https://peps.python.org/pep-0529/] y **`_enablelegacywindowsfsencoding()`** para obtener más información.

Distinto en la versión 3.7: Retorna `'utf-8'` si Modo UTF-8 de Python esta habilitado.

sys.getfilesystemencodeerrors()

Obtener la codificación de sistema de archivos: La codificación usada con el gestor de errores de sistema de archivos para convertir nombres de archivos en Unicode a nombres de archivos en bytes. El gestor de errores del sistema de archivos es retornado desde **`getfilesystemencoding()`**.

`os.fsencode()` y **`os.fsdecode()`** deben usarse para garantizar que se utilizan la codificación correcta y el modo de errores.

El codificación del sistema de archivos y manejo de errores está configurado en el inicio de Python por la función **`PyConfig_Read()`**: ver **`filesystem_encoding`** y **`filesystem_errors`** que son miembros de **`PyConfig`**.

Added in version 3.6.

sys.get_int_max_str_digits()

Devuelve el valor actual de integer string conversion length limitation. Véase también `set_int_max_str_digits()`.

Added in version 3.11.

sys.getrefcount(*object*)

Retorna el recuento de referencias del *object*. El recuento retornado es generalmente uno más alto de lo que cabría esperar, porque incluye la referencia (temporal) como argumento para `getrefcount()`.

Note that the returned value may not actually reflect how many references to the object are actually held. For example, some objects are immortal and have a very high refcount that does not reflect the actual number of references. Consequently, do not rely on the returned value to be accurate, other than a value of 0 or 1.

Detalles de implementación de CPython: Immortal objects with a large reference count can be identified via `_is_immortal()`.

Distinto en la versión 3.12: Los objetos inmortales tienen refcounts muy grandes que no coinciden con el número real de referencias al objeto.

sys.getrecursionlimit()

Retorna el valor actual del límite de recursividad, la profundidad máxima de la pila de intérpretes de Python. Este límite evita que la recursividad infinita cause un desbordamiento de la pila C y bloquee Python. Se puede configurar mediante `setrecursionlimit()`.

sys.getsizeof(*object*[, *default*])

Retorna el tamaño de un objeto en bytes. El objeto puede ser cualquier tipo de objeto. Todos los objetos integrados retornarán resultados correctos, pero esto no tiene por qué ser cierto para las extensiones de terceros, ya que es una implementación específica.

Solo se tiene en cuenta el consumo de memoria atribuido directamente al objeto, no el consumo de memoria de los objetos a los que se refiere.

Si se proporciona, se retornará *predeterminado* si el objeto no proporciona los medios para recuperar el tamaño. De lo contrario, se lanzará un **`TypeError`**.

`getsizeof()` llama al método `__sizeof__` del objeto y agrega una sobrecarga adicional del recolector de basura si el objeto es administrado por el recolector de basura.

See recursive sizeof recipe [https://code.activestate.com/recipes/577504-compute-memory-footprint-of-an-object-and-its-cont/] for an example of using **`getsizeof()`** recursively to find the size of containers and all their contents.

sys.getswitchinterval()

Return the interpreter's «thread switch interval» in seconds; see **`setswitchinterval()`**.

Added in version 3.2.

sys._getframe([*depth*])

Retorna un objeto de marco de la pila de llamadas. Si se proporciona un entero opcional *depth*, retorna el objeto de marco que muchas llamadas debajo de la parte superior de la pila. Si eso es más profundo que la pila de llamadas, se lanza **`ValueError`**. El valor predeterminado de *depth* es cero, lo que retorna el marco en la parte superior de la pila de llamadas.

Genera un auditing event `sys._getframe` con el argumento `frame`.

Esta función debe utilizarse únicamente para fines internos y especializados. No se garantiza que exista en todas las implementaciones de Python.

sys._getframemodulename([*depth*])

Devuelve el nombre de un módulo de la pila de llamadas. Si se proporciona el entero opcional *depth*, devuelve el módulo que muchas llamadas se encuentran debajo de la parte superior de la pila. Si es más profundo que la pila de llamadas, o si el módulo no es identificable, se devuelve `None`. El valor predeterminado para *depth* es cero, lo que devuelve el módulo a la parte superior de la pila de llamadas.

Genera un auditing event `sys._getframemodulename` con el argumento `depth`.

Esta función debe utilizarse únicamente para fines internos y especializados. No se garantiza que exista en todas las implementaciones de Python.

Added in version 3.12.

sys.getobjects(*limit*[, *type*])

This function only exists if CPython was built using the specialized configure option **`--with-trace-refs`**. It is intended only for debugging garbage-collection issues.

Return a list of up to *limit* dynamically allocated Python objects. If *type* is given, only objects of that exact type (not subtypes) are included.

Objects from the list are not safe to use. Specifically, the result will include objects from all interpreters that share their object allocator state (that is, ones created with **`PyInterpreterConfig.use_main_obmalloc`** set to 1 or using **`Py_NewInterpreter()`**, and the main interpreter). Mixing objects from different interpreters may lead to crashes or other unexpected behavior.

Detalles de implementación de CPython: This function should be used for specialized purposes only. It is not guaranteed to exist in all implementations of Python.

Distinto en la versión 3.14: The result may include objects from other interpreters.

sys.getprofile()

Obtiene la función de generador de perfiles establecida por `setprofile()`.

sys.gettrace()

Obtiene la función de seguimiento (*trace*) establecida por `settrace()`.

La función `gettrace()` está pensada solo para implementar depuradores, perfiladores, herramientas de cobertura y similares. Su comportamiento es parte de la plataforma de implementación, en lugar de parte de la definición del lenguaje y, por lo tanto, es posible que no esté disponible en todas las implementaciones de Python.

sys.getwindowsversion()

Retorna una tupla con nombre que describe la versión de Windows que se está ejecutando actualmente. Los elementos nombrados son *major*, *minor*, *build*, *platform*, *service_pack*, *service_pack_minor*, *service_pack_major*, *suite_mask*, *product_type* y *platform_version*. *service_pack* contiene una cadena de caracteres, *platform_version* una tupla de 3 y todos los demás valores son números enteros. También se puede acceder a los componentes por su nombre, por lo que `sys.getwindowsversion()[0]` es equivalente a `sys.getwindowsversion().major`. Para compatibilidad con versiones anteriores, solo los primeros 5 elementos se pueden recuperar mediante la indexación.

platform será `2` (VER_PLATFORM_WIN32_NT).

product_type puede ser uno de los siguientes valores:

Constante	Significado
1 (VER_NT_WORKSTATION)	El sistema es una estación de trabajo.
2 (VER_NT_DOMAIN_CONTROLLER)	El sistema es un controlador de dominio.
3 (VER_NT_SERVER)	El sistema es un servidor, pero no un controlador de

dominio.

Esta función envuelve la función Win32 **`GetVersionEx()`**; consulte la documentación de Microsoft sobre **`OSVERSIONINFOEX()`** para obtener más información sobre estos campos.

platform_version retorna la versión principal, la versión secundaria y el número de compilación del sistema operativo actual, en lugar de la versión que se está emulando para el proceso. Está diseñado para su uso en el registro en lugar de para la detección de características.

Nota

platform_version deriva la versión desde kernel32.dll que puede ser de una versión diferente a la versión de SO. Por favor usar el módulo **`platform`** para obtener una versión de SO precisa.

Availability: Windows.

Distinto en la versión 3.2: Cambiada a una tupla con nombre y agregado *service_pack_minor*, *service_pack_major*, *suite_mask*, y *product_type*.

Distinto en la versión 3.6: Agregado *platform_version*

sys.get_asyncgen_hooks()

Devuelve un objeto *asyncgen_hooks*, que es similar a un **`namedtuple`** con el formato `(firstiter, finalizer)`, donde se espera que *firstiter* y *finalizer* sean `None` o funciones que toman un asynchronous generator iterator como argumento y se utilizan para programar la finalización de un generador asíncrono mediante un bucle de eventos. .

Added in version 3.6: Ver **PEP 525** [https://peps.python.org/pep-0525/] para más detalles.

Nota

Esta función se ha añadido de forma provisional (consulte **PEP 411** [https://peps.python.org/pep-0411/] para obtener más detalles).

sys.get_coroutine_origin_tracking_depth()

Obtiene la profundidad de seguimiento del origen de la corrutina actual, según lo establecido por `set_coroutine_origin_tracking_depth()`.

Added in version 3.7.

Nota

Esta función se ha añadido de forma provisional (consulte **PEP 411** [https://peps.python.org/pep-0411/] para obtener más detalles). Úsela sólo para fines de depuración.

sys.hash_info

A named tuple dando parámetros de la implementación de hash numérico. Para obtener más detalles sobre el hash de tipos numéricos, consulte Calculo del hash de tipos numéricos.

hash_info.width

El ancho en bits utilizado para los valores hash.

hash_info.modulus

El módulo principal P utilizado para el esquema hash numérico

hash_info.inf

El valor hash devuelto para un infinito positivo

hash_info.nan

(Este atributo ya no se utiliza)

hash_info.imag

El multiplicador utilizado para la parte imaginaria de un número complejo.

hash_info.algorithm

El nombre del algoritmo para el hash de cadena, bytes y vista de memoria.

hash_info.hash_bits

El tamaño de salida interno del algoritmo hash.

hash_info.seed_bits

El tamaño de la clave semilla del algoritmo hash.

hash_info.cutoff

Cutoff for small string DJBX33A optimization in range `[1, cutoff)`.

Added in version 3.2.

Distinto en la versión 3.4: Added *algorithm*, *hash_bits*, *seed_bits*, and *cutoff*.

sys.hexversion

El número de versión codificado como un solo entero. Se garantiza que esto aumentará con cada versión, incluido el soporte adecuado para versiones que no son de producción. Por ejemplo, para probar que el intérprete de Python es al menos la versión 1.5.2, use:

```
if sys.hexversion >= 0x010502F0:
    # use some advanced feature
    ...
else:
    # use an alternative implementation or warn the
    ...
```

Esto se llama `hexversion` ya que solo parece realmente significativo cuando se ve como el resultado de pasarlo a la función incorporada **`hex()`**. El named tuple **`sys.version_info`** puede usarse para una codificación más amigable para los humanos de la misma información.

Se pueden encontrar más detalles de `hexversion` en Versiones de API y ABI.

sys.implementation

Un objeto que contiene información sobre la implementación del intérprete de Python en ejecución. Los siguientes atributos deben existir en todas las implementaciones de Python.

name es el identificador de la implementación, por ejemplo `'cpython'`. La cadena de caracteres real está definida por la implementación de Python, pero se garantiza que estará en minúsculas.

version es una tupla con nombre, en el mismo formato que **`sys.version_info`**. Representa la versión de la *implementación* de Python. Esto tiene un significado distinto de la versión específica del *lenguaje* de Python al que se ajusta el intérprete que se está ejecutando actualmente, que representa `sys.version_info`. Por ejemplo, para PyPy 1.8 `sys.implementation.version` podría ser `sys.version_info(1, 8, 0, 'final', 0)`, mientras que `sys.version_info` sería `sys.version_info(2, 7, 2, 'final', 0)`. Para CPython tienen el mismo valor, ya que es la implementación de referencia.

hexversion es la versión de implementación en formato hexadecimal, como **`sys.hexversion`**.

cache_tag es la etiqueta utilizada por la maquinaria de importación en los nombres de archivo de los módulos almacenados en caché. Por convención, sería una combinación del nombre y la versión de la implementación, como `'cpython-33'`. Sin embargo, una implementación de Python puede usar algún otro valor si corresponde. Si `cache_tag` está configurado como `None`, indica que el almacenamiento en caché del módulo debe estar deshabilitado.

supports_isolated_interpreters is a boolean value, whether this implementation supports multiple isolated interpreters. It is `True` for CPython on most platforms. Platforms with this support implement the low-level **`_interpreters`** module.

Ver también

PEP 684 [https://peps.python.org/pep-0684/], **PEP 734** [https://peps.python.org/pep-0734/], and **`concurrent.interpreters`**.

`sys.implementation` puede contener atributos adicionales específicos de la implementación de Python. Estos atributos no estándar deben comenzar con un guion bajo y no se describen aquí. Independientemente de su contenido, **`sys.implementation`** no cambiará durante la ejecución del intérprete, ni entre las versiones de implementación. (Sin embargo, puede cambiar entre las versiones del lenguaje Python). Consulte **PEP 421** [https://peps.python.org/pep-0421/] para obtener más información.

Added in version 3.3.

Distinto en la versión 3.14: Added `supports_isolated_interpreters` field.

Nota

La adición de nuevos atributos obligatorios debe pasar por el proceso normal de PEP. Consulte **PEP 421** [https://peps.python.org/pep-0421/] para obtener más información.

sys.int_info

Un named tuple que contiene información sobre la representación interna de Python de los enteros. Los atributos son de solo lectura.

int_info.bits_per_digit

El número de bits contenidos en cada dígito. Los enteros de Python se almacenan internamente en la base `2**int_info.bits_per_digit`.

int_info.sizeof_digit

El tamaño en bytes del tipo C utilizado para representar un dígito.

int_info.default_max_str_digits

El valor predeterminado para **`sys.get_int_max_str_digits()`** cuando no está configurado explícitamente de otra manera.

int_info.str_digits_check_threshold

El valor mínimo distinto de cero para **`sys.set_int_max_str_digits()`**, **`PYTHONINTMAXSTRDIGITS`** o **`-X int_max_str_digits`**.

Added in version 3.1.

Distinto en la versión 3.11: Se agregaron **`default_max_str_digits`** y **`str_digits_check_threshold`**.

sys._interactivehook_

Cuando existe este atributo, su valor se llama automáticamente (sin argumentos) cuando se lanza el intérprete en modo interactivo. Esto se hace después de leer el archivo **`PYTHONSTARTUP`**, para que pueda establecer este enlace allí. El módulo **`site`** establece este.

Lanza un auditing event `cpython.run_interactivehook` con el objeto gancho (*hook*) como argumento cuando se llama al gancho en el inicio.

Added in version 3.4.

sys.intern(*string*)

Ingresa *string* en la tabla de cadenas de caracteres «internadas» y retorna la cadena de caracteres interna, que es *string* en sí misma o una copia. Internar cadenas de caracteres es útil para obtener un poco de rendimiento en la búsqueda de diccionario: si las claves en un diccionario están internadas y la clave de búsqueda está interna, las comparaciones de claves (después del hash) se pueden realizar mediante una comparación de punteros en lugar de una comparación de cadenas. Normalmente, los nombres utilizados en los programas de Python se internan automáticamente, y los diccionarios utilizados para contener los atributos de módulo, clase o instancia tienen claves internas.

Interned strings are not immortal; you must keep a reference to the return value of `intern()` around to benefit from it.

sys._is_gil_enabled()

Return `True` if the GIL is enabled and `False` if it is disabled.

Added in version 3.13.

Detalles de implementación de CPython: It is not guaranteed to exist in all implementations of Python.

sys.is_finalizing()

Return `True` if the main Python interpreter is shutting down. Return `False` otherwise.

See also the `PythonFinalizationError` exception.

Added in version 3.5.

sys._jit

Utilities for observing just-in-time compilation.

Detalles de implementación de CPython: JIT compilation is an *experimental implementation detail* of CPython. `sys._jit`

is not guaranteed to exist or behave the same way in all Python implementations, versions, or build configurations.

Added in version 3.14.

_jit.is_available()

Return `True` if the current Python executable supports JIT compilation, and `False` otherwise. This can be controlled by building CPython with the `--experimental-jit` option on Windows, and the **`--enable-experimental-jit`** option on all other platforms.

_jit.is_enabled()

Return `True` if JIT compilation is enabled for the current Python process (implies **`sys._jit.is_available()`**), and `False` otherwise. If JIT compilation is available, this can be controlled by setting the **`PYTHON_JIT`** environment variable to `0` (disabled) or `1` (enabled) at interpreter startup.

_jit.is_active()

Return `True` if the topmost Python frame is currently executing JIT code (implies **`sys._jit.is_enabled()`**), and `False` otherwise.

Nota

This function is intended for testing and debugging the JIT itself. It should be avoided for any other purpose.

Nota

Due to the nature of tracing JIT compilers, repeated calls to this function may give surprising results. For example, branching on its return value will likely lead to unexpected behavior (if doing so causes JIT code to be entered or exited):

```
>>> for warmup in range(BIG_NUMBER):
...     # This line is "hot", and is eventuall
...     if sys._jit.is_active():
...         # This line is "cold", and is run
...         assert sys._jit.is_active()
...
Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
    assert sys._jit.is_active()
              ~~~~~~~~~~~~~~~~~~^^
AssertionError
```

sys.last_exc

Esta variable no siempre está definida; se establece en la instancia de excepción cuando no se maneja una excepción y el intérprete imprime un mensaje de error y un seguimiento de la pila. Su uso previsto es permitir que un usuario interactivo importe un módulo depurador y realice una depuración post-mortem sin tener que volver a ejecutar el comando que causó el error. (El uso típico es `import pdb; pdb.pm()` para ingresar al depurador post-mortem; consulte el módulo **`pdb`** para obtener más información).

Added in version 3.12.

sys._is_immortal(*op*)

Return **`True`** if the given object is immortal, **`False`** otherwise.

Nota

Objects that are immortal (and thus return `True` upon being passed to this function) are not guaranteed to be immortal in future versions, and vice versa for mortal objects.

Added in version 3.14.

Detalles de implementación de CPython: This function should be used for specialized purposes only. It is not guaranteed to exist in all implementations of Python.

sys._is_interned(*string*)

Return **`True`** if the given string is «interned», **`False`** otherwise.

Added in version 3.13.

Detalles de implementación de CPython: It is not guaranteed to exist in all implementations of Python.

sys.last_type
sys.last_value
sys.last_traceback

Estas tres variables están obsoletas; utilice **`sys.last_exc`** en su lugar. Contienen la representación heredada de `sys.last_exc`, devuelta por **`exc_info()`**.

sys.maxsize

Un entero que da el valor máximo que puede tomar una variable de tipo **`Py_ssize_t`**. Suele ser `2**31 - 1` en una plataforma de 32 bits y `2** 63 - 1` en una plataforma de 64 bits.

sys.maxunicode

Un número entero que da el valor del punto de código Unicode más grande, es decir, `1114111` (`0x10FFFF` en hexadecimal).

Distinto en la versión 3.3: Antes **PEP 393** [https://peps.python.org/pep-0393/], `sys.maxunicode` solía ser `0xFFFF` o `0x10FFFF`, dependiendo de la opción de configuración que especificaba si los caracteres Unicode se almacenaban como UCS-2 o UCS-4.

sys.meta_path

A list of meta path finder objects that have their **`find_spec()`** methods called to see if one of the objects can find the module to be imported. By default, it holds entries that implement Python's default import semantics. The **`find_spec()`** method is called with at least the absolute name of the module being imported. If the module to be imported is contained in a package, then the parent package's **`__path__`** attribute is passed in as a second argument. The method returns a module spec, or `None` if the module cannot be found.

Ver también

`importlib.abc.MetaPathFinder`
La clase base abstracta que define la interfaz de los objetos del buscador en **`meta_path`**.

`importlib.machinery.ModuleSpec`
La clase concreta que **`find_spec()`** debería retornar instancias de.

Distinto en la versión 3.4: Module specs were introduced in Python 3.4, by **PEP 451** [https://peps.python.org/pep-0451/].

Distinto en la versión 3.12: Removed the fallback that looked for a **`find_module()`** method if a **`meta_path`** entry didn't have a **`find_spec()`** method.

sys.modules

Este es un diccionario que asigna nombres de módulos a módulos que ya se han cargado. Esto se puede manipular para forzar la recarga de módulos y otros trucos. Sin embargo, reemplazar el diccionario no necesariamente funcionará como se esperaba y eliminar elementos esenciales del diccionario puede provocar que Python falle. Si desea iterar sobre este diccionario global, utilice siempre `sys.modules.copy()` o `tuple(sys.modules)` para evitar excepciones, ya que su tamaño puede cambiar durante la iteración como efecto secundario del código o la actividad en otros subprocesos.

sys.orig_argv

La lista de los argumentos originales de la línea de comando que son pasados al ejecutable de Python.

The elements of `sys.orig_argv` are the arguments to the Python interpreter, while the elements of `sys.argv` are the arguments to the user's program. Arguments consumed by the interpreter itself will be present in `sys.orig_argv` and missing from `sys.argv`.

Added in version 3.10.

sys.path

Una lista de cadenas de caracteres que especifica la ruta de búsqueda de módulos. Inicializado desde la variable de entorno `PYTHONPATH`, más un valor predeterminado que depende de la instalación.

De forma predeterminada, tal como se inicializa al iniciar el programa, se antepone una ruta potencialmente insegura a `sys.path` (*before* son las entradas insertadas como resultado de `PYTHONPATH`):

- Línea de comando `python -m module`: anteponga el directorio de trabajo actual.
- `python script.py` línea de comandos: anteponer el directorio del script. Si es un enlace simbólico, resuelve los enlaces simbólicos.
- Líneas de comando `python -c code` y `python` (REPL): anteponen una cadena vacía, que significa el directorio de trabajo actual.

Para no anteponer esta ruta potencialmente insegura, utilice la opción de línea de comando `-P` o la variable de entorno `PYTHONSAFEPATH`.

Un programa es libre de modificar esta lista para sus propios fines. Sólo se deben agregar cadenas a `sys.path`; todos los demás tipos de datos se ignoran durante la importación.

Ver también

- Módulo **`site`** Esto describe cómo usar archivos .pth para extender **`sys.path`**.

sys.path_hooks

Una lista de invocables que toman un argumento de ruta para intentar crear un finder para la ruta. Si se puede crear un buscador, el invocable debe retornar; de lo contrario, lanza **`ImportError`**.

Especificado originalmente en **PEP 302** [https://peps.python.org/pep-0302/].

sys.path_importer_cache

Un diccionario que actúa como caché para objetos finder. Las claves son rutas que se han pasado a **`sys.path_hooks`** y los valores son los buscadores que se encuentran. Si una ruta es una ruta de sistema de archivos válida pero no se encuentra ningún buscador en **`sys.path_hooks`**, entonces se almacena `None`.

Especificado originalmente en **PEP 302** [https://peps.python.org/pep-0302/].

sys.platform

A string containing a platform identifier. Known values are:

Sistema	**Valor `platform`**
AIX	`'aix'`
Android	`'android'`
Emscripten	`'emscripten'`
FreeBSD	`'freebsd'`
iOS	`'ios'`
Linux	`'linux'`
macOS	`'darwin'`
Windows	`'win32'`
Windows/Cygwin	`'cygwin'`

WASI	`'wasi'`

On Unix systems not listed in the table, the value is the lowercased OS name as returned by `uname -s`, with the first part of the version as returned by `uname -r` appended, e.g. `'sunos5'`, *at the time when Python was built*. Unless you want to test for a specific system version, it is therefore recommended to use the following idiom:

```
if sys.platform.startswith('sunos'):
    # SunOS-specific code here...
```

Distinto en la versión 3.3: On Linux, **`sys.platform`** doesn't contain the major version anymore. It is always `'linux'`, instead of `'linux2'` or `'linux3'`.

Distinto en la versión 3.8: On AIX, **`sys.platform`** doesn't contain the major version anymore. It is always `'aix'`, instead of `'aix5'` or `'aix7'`.

Distinto en la versión 3.13: On Android, **`sys.platform`** now returns `'android'` rather than `'linux'`.

Distinto en la versión 3.14: On FreeBSD, **`sys.platform`** doesn't contain the major version anymore. It is always `'freebsd'`, instead of `'freebsd13'` or `'freebsd14'`.

Ver también

`os.name` tiene una granularidad más gruesa. **`os.uname()`** proporciona información de versión dependiente del sistema.

El módulo **`platform`** proporciona comprobaciones detalladas de la identidad del sistema.

sys.platlibdir

Nombre del directorio de la biblioteca específica de la plataforma. Se utiliza para construir la ruta de la biblioteca estándar y las rutas de los módulos de extensión instalados.

Es igual a `"lib"` en la mayoría de las plataformas. En Fedora y SuSE, es igual a `"lib64"` en plataformas de 64 bits, lo que da las siguientes rutas `sys.path` (donde `X.Y` es la versión `mayor.menor` de Python):

- `/usr/lib64/pythonX.Y/`: Biblioteca estándar (como `os.py` del módulo **`os`**)
- `/usr/lib64/pythonX.Y/lib-dynload/`: Módulos de extensión C de la biblioteca estándar (como el módulo **`errno`**, el nombre exacto del archivo es específico de la plataforma)
- `/usr/lib/pythonX.Y/site-packages/` (utilice siempre `lib`, no **`sys.platlibdir`**): Módulos de terceros
- `/usr/lib64/pythonX.Y/site-packages/`: Módulos de extensión C de paquetes de terceros

Added in version 3.9.

sys.prefix

Una cadena que proporciona el prefijo del directorio específico del sitio donde se instalan los archivos Python independientes de la plataforma; en Unix, el valor predeterminado es `/usr/local`. Esto se puede configurar en el momento de la compilación con el argumento **`--prefix`** del script **configure**. Consulte Rutas de instalación para conocer las rutas derivadas.

Nota

If a virtual environment is in effect, this **`prefix`** will point to the virtual environment. The value for the Python installation will still be available, via **`base_prefix`**. Refer to Virtual Environments for more information.

Distinto en la versión 3.14: When running under a virtual environment, **`prefix`** and **`exec_prefix`** are now set to the virtual environment prefix by the path initialization, instead of **`site`**. This means that **`prefix`** and **`exec_prefix`** always point to the virtual environment, even when **`site`** is disabled (**`-S`**).

sys.ps1
sys.ps2

Cadenas de caracteres que especifican el indicador principal y secundario del intérprete. Estos solo se definen si el intérprete está en modo interactivo. Sus valores iniciales en este caso son `'>>>'` y `'...'`. Si se asigna un objeto que no es una cadena a cualquiera de las variables, su **`str()`** se vuelve a evaluar cada vez que el intérprete se prepara para leer un nuevo comando interactivo; esto se puede utilizar para implementar un mensaje dinámico.

sys.setdlopenflags(*n*)

Configure los indicadores utilizados por el intérprete para las llamadas **`dlopen()`**, como cuando el intérprete carga módulos de extensión. Entre otras cosas, esto permitirá una resolución diferida de símbolos al importar un módulo, si se llama como `sys.setdlopenflags(0)`. Para compartir símbolos entre módulos de extensión, llame como `sys.setdlopenflags(os.RTLD_GLOBAL)`. Los nombres simbólicos para los valores de las banderas se pueden encontrar en el módulo **`os`** (constantes `RTLD_`*`xxx`*, por ejemplo, **`os.RTLD_LAZY`**).

Availability: Unix.

sys.set_int_max_str_digits(*maxdigits*)

Establece la limitación de la longitud de conversión de cadenas enteras utilizada por este intérprete. Véase también **`get_int_max_str_digits()`**.

Added in version 3.11.

sys.setprofile(*profilefunc*)

Configura la función de perfil del sistema, que le permite implementar un generador de perfiles de código fuente de Python en Python. Consulte el capítulo Los perfiladores de

Python para obtener más información sobre el generador de perfiles de Python. La función de perfil del sistema se llama de manera similar a la función de seguimiento del sistema (ver `settrace()`), pero se llama con diferentes eventos, por ejemplo, no se llama para cada línea de código ejecutada (solo al llamar y regresar, pero el evento de retorno se informa incluso cuando se ha establecido una excepción). La función es específica de subprocesos, pero el generador de perfiles no tiene forma de conocer los cambios de contexto entre subprocesos, por lo que no tiene sentido usar esto en presencia de varios subprocesos. Además, su valor de retorno no se usa, por lo que simplemente puede retornar `None`. Un error en la función del perfil provocará su desarmado.

Nota

The same tracing mechanism is used for **`setprofile()`** as `settrace()`. To trace calls with **`setprofile()`** inside a tracing function (e.g. in a debugger breakpoint), see `call_tracing()`.

Las funciones de perfil deben tener tres argumentos: *frame*, *event* y *arg*. *frame* es el marco de pila actual. *event* es una cadena de caracteres: `'call'`, `'return'`, `'c_call'`, `'c_return'`, o `'c_exception'`. *arg* depende del tipo de evento.

Los eventos tienen el siguiente significado:

`'call'`

Se llama a una función (o se ingresa algún otro bloque de código). Se llama a la función de perfil; *arg* es `None`.

`'return'`

Una función (u otro bloque de código) está a punto de regresar. Se llama a la función de perfil; *arg* es el valor que se retornará, o `None` si el evento es causado por una excepción.

`'c_call'`

Una función C está a punto de ser llamada. Esta puede ser una función de extensión o una incorporada. *arg* es el objeto de función de C.

`'c_return'`

Ha vuelto una función C. *arg* es el objeto de función de C.

`'c_exception'`

Una función C ha generado una excepción. *arg* es el objeto de función de C.

Lanza un auditing event `sys.setprofile` sin argumentos.

sys.setrecursionlimit(*limit*)

Establece la profundidad máxima de la pila de intérpretes de Python en *limit*. Este límite evita que la recursividad infinita cause un desbordamiento de la pila C y bloquee Python.

El límite más alto posible depende de la plataforma. Un usuario puede necesitar establecer un límite más alto cuando tiene un programa que requiere una recursividad profunda y una plataforma que admite un límite más alto. Esto debe hacerse con cuidado, ya que un límite demasiado alto puede provocar un accidente.

Si el nuevo límite es demasiado bajo en la profundidad de recursividad actual, se genera una excepción **`RecursionError`**.

Distinto en la versión 3.5.1: A la excepción **`RecursionError`** ahora se lanza si el nuevo límite es demasiado bajo en la profundidad de recursión actual.

sys.setswitchinterval(*interval*)

Establece el intervalo de cambio de hilo del intérprete (en segundos). Este valor de punto flotante determina la duración ideal de los «segmentos de tiempo» asignados a los subprocesos de Python que se ejecutan simultáneamente. Tenga en cuenta que el valor real puede ser mayor, especialmente si se utilizan funciones o métodos internos de larga duración. Además, qué hilo se programa al final del intervalo es decisión del sistema operativo. El intérprete no tiene su propio programador.

Added in version 3.2.

sys.settrace(*tracefunc*)

Configura la función de seguimiento del sistema, que le permite implementar un depurador de código fuente de Python en Python. La función es específica del hilo; para que un depurador admita múltiples subprocesos, debe registrar una función de seguimiento usando `settrace()` para cada subproceso que se depura o use `threading.settrace()`.

Trace functions should have three arguments: *frame*, *event*, and *arg*. *frame* is the current stack frame. *event* is a string: `'call'`, `'line'`, `'return'`, `'exception'` or `'opcode'`. *arg* depends on the event type.

La función de rastreo se invoca (con *event* establecido en `'call'`) cada vez que se ingresa un nuevo ámbito local; debe retornar una referencia a una función de rastreo local que se usará para el nuevo alcance, o `None` si no se debe rastrear el alcance.

The local trace function should return a reference to itself, or to another function which would then be used as the local trace function for the scope.

Si se produce algún error en la función de seguimiento, se desarmará, al igual que se llama a `settrace(None)`.

Nota

Tracing is disabled while calling the trace function (e.g. a function set by `settrace()`). For recursive tracing see `call_tracing()`.

Los eventos tienen el siguiente significado:

`'call'`

Se llama a una función (o se ingresa algún otro bloque de código). Se llama a la función de rastreo global; *arg* es `None`; el valor de retorno especifica la función de rastreo local.

`'line'`

The interpreter is about to execute a new line of code or re-execute the condition of a loop. The local trace function is called; *arg* is `None`; the return value specifies the new local trace function. See `Objects/lnotab_notes.txt` for a detailed explanation of how this works. Per-line events may be disabled for a frame by setting **`f_trace_lines`** to **`False`** on that frame.

`'return'`

Una función (u otro bloque de código) está a punto de regresar. Se llama a la función de rastreo local; *arg* es el valor que se retornará, o `None` si el evento es causado por una excepción. Se ignora el valor de retorno de la función de seguimiento.

`'exception'`

Ha ocurrido una excepción. Se llama a la función de rastreo local; *arg* es una tupla `(exception, value, traceback)`; el valor de retorno especifica la nueva función de rastreo local.

`'opcode'`

The interpreter is about to execute a new opcode (see **`dis`** for opcode details). The local trace function is called; *arg* is `None`; the return value specifies the new local trace function. Per-opcode events are not emitted by default: they must be explicitly requested by setting **`f_trace_opcodes`** to **`True`** on the frame.

Tenga en cuenta que como una excepción se propaga a lo largo de la cadena de llamadas de funciones, se lanza un evento de `'excepción'` en cada nivel.

Para un uso más detallado, es posible establecer una función de seguimiento asignando `frame.f_trace = tracefunc` explícitamente, en lugar de confiar en que se establezca indirectamente a través del valor de retorno de una función de seguimiento ya instalada. Esto también es necesario para activar la función de seguimiento en el marco actual, lo cual **`settrace()`** no funciona. Tenga en cuenta que para que esto funcione, se debe haber instalado una función de rastreo global con **`settrace()`** para habilitar la maquinaria de rastreo en tiempo de ejecución, pero no necesita ser la misma función de rastreo (por ejemplo, podría ser una función de rastreo de sobrecarga baja que simplemente retorna `None` para deshabilitarse inmediatamente en cada cuadro).

Para obtener más información sobre los objetos de código y marco, consulte Jerarquía de tipos estándar.

Lanza un auditing event `sys.settrace` sin argumentos.

La función **`settrace()`** está pensada únicamente para implementar depuradores, perfiladores, herramientas de cobertura y similares. Su comportamiento es parte de la plataforma de implementación, en lugar de parte de la definición del lenguaje y, por lo tanto, es posible que no esté disponible en todas las implementaciones de Python.

Distinto en la versión 3.7: `'opcode'` event type added; **`f_trace_lines`** and **`f_trace_opcodes`** attributes added to frames

sys.set_asyncgen_hooks(*[firstiter] [, finalizer]*)

Acepta dos argumentos de palabras clave opcionales que son invocables que aceptan un asynchronous generator iterator como argumento. Se llamará al *firstiter* invocable cuando se repita un generador asincrónico por primera vez. Se llamará al *finalizer* cuando un generador asincrónico esté a punto de ser recolectado como basura.

Lanza un auditing event `sys.set_asyncgen_hooks_firstiter` sin argumentos.

Lanza un auditing event `sys.set_asyncgen_hooks_finalizer` sin argumentos.

Se lanzan dos eventos de auditoría porque la API subyacente consta de dos llamadas, cada una de las cuales debe generar su propio evento.

Added in version 3.6: Ver **PEP 525** [https://peps.python.org/pep-0525/] para más detalles, y para un ejemplo de referencia de un método *finalizer* ver la implementación de `asyncio.Loop.shutdown_asyncgens` en Lib/asyncio/base_events.py [https://github.com/python/cpython/tree/3.14/Lib/asyncio/base_events.py]

Nota

Esta función se ha añadido de forma provisional (consulte **PEP 411** [https://peps.python.org/pep-0411/] para obtener más detalles).

sys.set_coroutine_origin_tracking_depth(*depth*)

Allows enabling or disabling coroutine origin tracking. When enabled, the `cr_origin` attribute on coroutine objects will contain a tuple of (filename, line number, function name) tuples describing the traceback where the coroutine object was created, with the most recent call first. When disabled, `cr_origin` will be `None`.

To enable, pass a *depth* value greater than zero; this sets the number of frames whose information will be captured. To disable, set *depth* to zero.

Esta configuración es específica de cada hilo.

Added in version 3.7.

Nota

Esta función se ha añadido de forma provisional (consulte **PEP 411** [https://peps.python.org/pep-0411/] para obtener más detalles). Úsela sólo para fines de depuración.

sys.activate_stack_trampoline(*backend*, */*)

Activa el trampolín *backend* del perfilador de pila. El único backend soportado es `"perf"`.

Stack trampolines cannot be activated if the JIT is active.

Availability: Linux.

Added in version 3.12.

Ver también

- Soporte de Python para el perfilador perf de Linux
- https://perf.wiki.kernel.org

sys.deactivate_stack_trampoline()

Desactive el backend del trampolín del perfilador de pila actual.

Si no se está ejecutando ningún perfilador, esta función no tiene efecto.

Availability: Linux.

Added in version 3.12.

sys.is_stack_trampoline_active()

Retorna `True` si un trampolín de perfil de pila está activo.

Availability: Linux.

Added in version 3.12.

sys.remote_exec(*pid*, *script*)

Executes *script*, a file containing Python code in the remote process with the given *pid*.

This function returns immediately, and the code will be executed by the target process's main thread at the next available opportunity, similarly to how signals are handled. There is no interface to determine when the code has been executed. The caller is responsible for making sure that the file still exists whenever the remote process tries to read it and that it hasn't been overwritten.

The remote process must be running a CPython interpreter of the same major and minor version as the local process. If either the local or remote interpreter is pre-release (alpha, beta, or release candidate) then the local and remote interpreters must be the same exact version.

See Remote debugging attachment protocol for more information about the remote debugging mechanism.

When the code is executed in the remote process, an auditing event `sys.remote_exec` is raised with the *pid* and the path to the script file. This event is raised in the process that called **`sys.remote_exec()`**.

When the script is executed in the remote process, an auditing event `cpython.remote_debugger_script` is raised with the path in the remote process. This event is raised in the remote process, not the one that called **`sys.remote_exec()`**.

Availability: Unix, Windows.

Added in version 3.14: See **PEP 768** [https://peps.python.org/pep-0768/] for more details.

sys._enablelegacywindowsfsencoding()

Cambia la codificación del sistema de archivos y manejo de errores a "mbcs" y "replace" respectivamente, para mantener la coherencia con las versiones de Python anteriores a la 3.6.

Esto es equivalente a definir la variable de entorno `PYTHONLEGACYWINDOWSFSENCODING` antes de iniciar Python.

También ver `sys.getfilesystemencoding()` y `sys.getfilesystemencodeerrors()`.

Availability: Windows.

Nota

Changing the filesystem encoding after Python startup is risky because the old fsencoding or paths encoded by the old fsencoding may be cached somewhere. Use `PYTHONLEGACYWINDOWSFSENCODING` instead.

Added in version 3.6: Consulte **PEP 529** [https://peps.python.org/pep-0529/] para obtener más detalles.

Deprecated since version 3.13, will be removed in version 3.16: Use `PYTHONLEGACYWINDOWSFSENCODING` instead.

sys.stdin
sys.stdout
sys.stderr

Objetos de archivo utilizado por el intérprete para entradas, salidas y errores estándar:

- `stdin` se usa para todas las entradas interactivas (incluidas las llamadas a `input()`);
- `stdout` se usa para la salida de `print()` y expression y para las solicitudes de `input()`;
- Las propias indicaciones del intérprete y sus mensajes de error van a `stderr`.

Estos flujos son regulares archivos de texto como los retornados por la función `open()`. Sus parámetros se eligen de la siguiente manera:

- La codificación y manejo de error son inicializado desde `PyConfig.stdio_encoding` y

`PyConfig.stdio_errors`.

En Windows, se usa UTF-8 para el dispositivo de consola. Los dispositivos que no son caracteres, como archivos de disco y tuberías, utilizan la codificación de la configuración regional del sistema (por ejemplo, la página de códigos ANSI). Los dispositivos de caracteres que no son de consola, como NUL (es decir, donde `isatty()` retorna `True`) utilizan el valor de las páginas de códigos de entrada y salida de la consola al inicio, respectivamente para stdin y stdout/stderr. Esto pasa por defecto al sistema de codificación local si el proceso no esta inicialmente acoplado a una consola.

El comportamiento especial de la consola se puede anular configurando la variable de entorno PYTHONLEGACYWINDOWSSTDIO antes de iniciar Python. En ese caso, las páginas de códigos de la consola se utilizan como para cualquier otro dispositivo de caracteres.

En todas las plataformas, puede anular la codificación de caracteres configurando la variable de entorno **`PYTHONIOENCODING`** antes de iniciar Python o usando la nueva **`-X`** `utf8` opción de línea de comando y **`PYTHONUTF8`** variable de entorno. Sin embargo, para la consola de Windows, esto solo se aplica cuando **`PYTHONLEGACYWINDOWSSTDIO`** también está configurado.

- Cuando es interactivo, el flujo de `stdout` se almacena en búfer de línea. En caso contrario, se almacena en búfer de bloque como los archivos de texto normales. El flujo `stderr` tiene un búfer de línea en ambos casos. Puede hacer que ambos flujos no tengan búfer pasando la opción de línea de comandos **`-u`** o estableciendo la variable de entorno **`PYTHONUNBUFFERED`**.

Distinto en la versión 3.9: Ahora, `stderr` no interactivo tiene búfer de línea en lugar de búfer completo.

Nota

Para escribir o leer datos binarios desde/hacia los flujos estándar, use el objeto binario subyacente **`buffer`**. Por ejemplo, para escribir bytes en **`stdout`**, use `sys.stdout.buffer.write(b'abc')`.

However, if you are writing a library (and do not control in which context its code will be executed), be aware that the standard streams may be replaced with file-like objects like **`io.StringIO`** which do not support the **`buffer`** attribute.

sys.__stdin__
sys.__stdout__
sys.__stderr__

Estos objetos contienen los valores originales de `stdin`, `stderr` y `stdout` al inicio del programa. Se utilizan durante la finalización y podrían ser útiles para imprimir en el flujo estándar real sin importar si el objeto `sys.std*` ha sido redirigido.

También se puede utilizar para restaurar los archivos reales a objetos de archivo de trabajo conocidos en caso de que se hayan sobrescrito con un objeto roto. Sin embargo, la forma preferida de hacer esto es guardar explícitamente la secuencia anterior antes de reemplazarla y restaurar el objeto guardado.

Nota

En algunas condiciones, `stdin`, `stdout` y `stderr`, así como los valores originales `__stdin__`, `__stdout__` y `__stderr__` pueden ser `None`. Suele ser el caso de las aplicaciones GUI de Windows que no están conectadas a una consola y las aplicaciones Python que comienzan con **pythonw**.

sys.stdlib_module_names

Un fronzenset de strings que contiene los nombres de los módulos de la librería estándar.

Es lo mismo para todas las plataformas. Módulos que no están disponibles en algunas plataformas y deshabilitados durante la construcción de Python también son listados. Todos los tipos de modulo son listados: Python puro, incorporados, congelados y módulos de extensión. Módulos de prueba son excluidos.

Para paquete, solo el principal es listado; sub-paquetes y sub-módulos no son listados. Por ejemplo, el paquete `email` es listado, pero el sub-paquete `email.mime` y el sub-módulo `email.message` no es listado.

Consulte también la lista **`sys.builtin_module_names`**.

Added in version 3.10.

sys.thread_info

Un named tuple que contiene información sobre la implementación del hilo.

thread_info.name

El nombre de la implementación del hilo:

- `"nt"`: hilos de Windows
- `"pthread"`: hilos de POSIX
- `"pthread-stubs"`: stub de hilos POSIX (en plataformas WebAssembly sin soporte de hilos)
- `"solaris"`: hilos de Solaris

thread_info.lock

El nombre de la implementación del bloqueo:

- `"semaphore"`: un bloqueo que utiliza un semáforo
- `"mutex+cond"`: un bloqueo que utiliza un mutex y una variable de condición
- `None` si esta información es desconocida

thread_info.version

El nombre y la versión de la biblioteca de subprocesos. Es una cadena, o `None` si se desconoce esta información.

Added in version 3.3.

sys.tracebacklimit

Cuando esta variable se establece en un valor entero, determina el número máximo de niveles de información de rastreo que se imprime cuando ocurre una excepción no controlada. El valor predeterminado es `1000`. Cuando se establece en `0` o menos, toda la información de rastreo se suprime y solo se imprimen el tipo y el valor de excepción.

sys.unraisablehook(*unraisable*, */*)

Maneja una excepción que no se puede lanzar.

Se llama cuando se ha producido una excepción pero no hay forma de que Python la maneje. Por ejemplo, cuando un destructor lanza una excepción o durante la recolección de basura (**`gc.collect()`**).

El argumento *unraisable* tiene los siguientes atributos:

- **`exc_type`**: tipo de excepción.
- **`exc_value`**: Valor de excepción, puede ser `None`.
- **`exc_traceback`**: rastreo de excepción, puede ser `None`.
- **`err_msg`**: Mensaje de error, puede ser `None`.
- **`object`**: Objeto que causa la excepción, puede ser `None`.

El enlace predeterminado da formato a **`err_msg`** y **`object`** como: `f'{err_msg}: {object!r}'`; utilice el mensaje de error «Excepción ignorada en» si **`err_msg`** es `None`.

`sys.unraisablehook()` se puede anular para controlar cómo se manejan las excepciones que no se pueden evaluar.

Ver también

`excepthook()` que maneja excepciones no detectadas.

Advertencia

Almacenar `exc_value` usando un gancho personalizado puede crear un ciclo de referencia. Se debe borrar explícitamente la interrupción del ciclo de referencia cuando la excepción ya no sea necesaria.

Almacenar `object` usando un gancho personalizado puede resucitarlo si está configurado en un objeto que se está finalizando. Evite almacenar `object` después de que se complete el gancho personalizado para evitar resucitar objetos.

Genera un evento de auditoría `sys.unraisablehook` con los argumentos *hook*, *unraisable* cuando ocurre una excepción que no se puede manejar. El objeto *unraisable* es el mismo que se pasará al gancho. Si no se ha establecido ningún gancho, *hook* puede ser `None`.

Added in version 3.8.

sys.version

Una cadena de caracteres que contiene el número de versión del intérprete de Python más información adicional sobre el número de compilación y el compilador utilizado. Esta cadena se muestra cuando se inicia el intérprete interactivo. No extraiga información de la versión de él, en su lugar, use `version_info` y las funciones proporcionadas por el módulo `platform`.

sys.api_version

The C API version, equivalent to the C macro `PYTHON_API_VERSION`. Defined for backwards compatibility.

Currently, this constant is not updated in new Python versions, and is not useful for versioning. This may change in the future.

sys.version_info

Una tupla que contiene los cinco componentes del número de versión: *major*, *minor*, *micro*, *releaselevel*, y *serial*. Todos los valores excepto *releaselevel* son números enteros; el nivel de lanzamiento es `'alpha'`, `'beta'`, `'candidate'`, o `'final'`. El valor de `version_info` correspondiente a la versión 2.0 de Python es `(2, 0, 0, 'final', 0)`. También se puede acceder a los componentes por su nombre, por lo que `sys.version_info[0]` es equivalente a `sys.version_info.major` y así sucesivamente.

Distinto en la versión 3.1: Se agregaron atributos de componentes con nombre.

sys.warnoptions

Este es un detalle de implementación del marco de advertencias; No modifique este valor. Consulte el módulo **`warnings`** para obtener más información sobre el marco de advertencias.

sys.winver

The version number used to form registry keys on Windows platforms. This is stored as string resource 1000 in the Python DLL. The value is normally the major and minor versions of the running Python interpreter. It is provided in the **`sys`** module for informational purposes; modifying this value has no effect on the registry keys used by Python.

Availability: Windows.

sys.monitoring

Espacio de nombres(Namespace) que contiene funciones y constantes para registrar retrollamadas(callbacks) y controlar eventos de monitorización. Véase **`sys.monitoring`** para más detalles.

sys._xoptions

Un diccionario de las diversas flags específicas de la implementación pasa a través de la opción de línea de comandos `-X`. Los nombres de las opciones se asignan a sus valores, si se dan explícitamente, o a `True`. Ejemplo:

```
$ ./python -Xa=b -Xc
Python 3.2a3+ (py3k, Oct 16 2010, 20:14:50)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" fo
>>> import sys
>>> sys._xoptions
{'a': 'b', 'c': True}
```

Esta es una forma específica de CPython de acceder a las opciones que se pasan a través de `-X`. Otras implementaciones pueden exportarlos a través de otros medios, o no exportarlos.

Added in version 3.2.

Citas

[C99]

ISO/CEI 9899:1999. «Lenguajes de programación - C.» Un borrador público de este estándar está disponible en https://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf.

`sys.monitoring` — Execution event monitoring

Added in version 3.12.

Nota

`sys.monitoring` is a namespace within the **`sys`** module, not an independent module, and `import sys.monitoring` would fail with a **`ModuleNotFoundError`**. Instead, simply `import sys` and then use `sys.monitoring`.

Este espacio de nombres proporciona acceso a las funciones y constantes necesarias para activar y controlar el monitoreo de eventos.

As programs execute, events occur that might be of interest to tools that monitor execution. The **`sys.monitoring`** namespace provides means to receive callbacks when events of interest occur.

La API de monitoreo consta de tres componentes:

- Tool identifiers
- Events
- Callbacks

Identificadores de herramientas

A tool identifier is an integer and the associated name. Tool identifiers are used to discourage tools from interfering with each other and to allow multiple tools to operate at the same time. Currently tools are completely independent and cannot be used to monitor each other. This restriction may be lifted in the future.

Before registering or activating events, a tool should choose an identifier. Identifiers are integers in the range 0 to 5 inclusive.

Registro y uso de herramientas

sys.monitoring.use_tool_id(*tool_id: int*, *name: str*, */*) → None

Must be called before *tool_id* can be used. *tool_id* must be in the range 0 to 5 inclusive. Raises a **`ValueError`** if *tool_id* is in use.

sys.monitoring.clear_tool_id(*tool_id: int*, */*) → None

Unregister all events and callback functions associated with *tool_id*.

sys.monitoring.free_tool_id(*tool_id: int*, */*) → None

Should be called once a tool no longer requires *tool_id*. Will call **`clear_tool_id()`** before releasing *tool_id*.

sys.monitoring.get_tool(*tool_id: int*, */*) → str | None

Returns the name of the tool if *tool_id* is in use, otherwise it returns `None`. *tool_id* must be in the range 0 to 5 inclusive.

La máquina virtual trata todos los ID de la misma manera con respecto a los eventos, pero los siguientes ID están predefinidos para facilitar la cooperación de las herramientas:

```
sys.monitoring.DEBUGGER_ID = 0
sys.monitoring.COVERAGE_ID = 1
sys.monitoring.PROFILER_ID = 2
sys.monitoring.OPTIMIZER_ID = 5
```

Eventos

Son aceptados los siguientes eventos:

sys.monitoring.events.BRANCH_LEFT

A conditional branch goes left.

It is up to the tool to determine how to present «left» and «right» branches. There is no guarantee which branch is «left» and which is «right», except that it will be consistent for the duration of the program.

sys.monitoring.events.BRANCH_RIGHT

A conditional branch goes right.

sys.monitoring.events.CALL

Una llamada en código Python (el evento ocurre antes de la llamada).

sys.monitoring.events.C_RAISE

An exception raised from any callable, except for Python functions (event occurs after the exit).

sys.monitoring.events.C_RETURN

Return from any callable, except for Python functions (event occurs after the return).

sys.monitoring.events.EXCEPTION_HANDLED

Se maneja una excepción.

sys.monitoring.events.INSTRUCTION

Está a punto de ejecutarse una instrucción de VM.

sys.monitoring.events.JUMP

Se realiza un salto incondicional en el gráfico de flujo de control.

sys.monitoring.events.LINE

Está a punto de ejecutarse una instrucción que tiene un número de línea diferente al de la instrucción anterior.

sys.monitoring.events.PY_RESUME

Resumption of a Python function (for generator and coroutine functions), except for `throw()` calls.

sys.monitoring.events.PY_RETURN

Retorna de una función Python (ocurre inmediatamente antes del retorno, el marco del destinatario estará en la pila).

sys.monitoring.events.PY_START

Inicio de una función Python (ocurre inmediatamente después de la llamada, el marco del destinatario estará en la pila)

sys.monitoring.events.PY_THROW

A Python function is resumed by a `throw()` call.

sys.monitoring.events.PY_UNWIND

Exit from a Python function during exception unwinding. This includes exceptions raised directly within the function and that are allowed to continue to propagate.

sys.monitoring.events.PY_YIELD

Rinde (*yield*) desde una función Python (ocurre inmediatamente antes del rendimiento, el marco del destinatario estará en la pila).

sys.monitoring.events.RAISE

An exception is raised, except those that cause a **`STOP_ITERATION`** event.

sys.monitoring.events.RERAISE

An exception is re-raised, for example at the end of a **`finally`** block.

sys.monitoring.events.STOP_ITERATION

An artificial **`StopIteration`** is raised; see the STOP_ITERATION event.

Es posible que se agreguen más eventos en el futuro.

These events are attributes of the **`sys.monitoring.events`** namespace. Each event is represented as a power-of-2 integer constant. To define a set of events, simply bitwise OR the individual events together. For example, to specify both **`PY_RETURN`** and **`PY_START`** events, use the expression `PY_RETURN | PY_START`.

sys.monitoring.events.NO_EVENTS

An alias for `0` so users can do explicit comparisons like:

```
if get_events(DEBUGGER_ID) == NO_EVENTS:
    ...
```

Setting this event deactivates all events.

Eventos locales

Los eventos locales están asociados con la ejecución normal del programa y ocurren en lugares claramente definidos. Todos los eventos locales se pueden desactivar. Los eventos locales son:

- **`PY_START`**
- **`PY_RESUME`**
- **`PY_RETURN`**
- **`PY_YIELD`**
- **`CALL`**
- **`LINE`**
- **`INSTRUCTION`**
- **`JUMP`**
- **`BRANCH_LEFT`**
- **`BRANCH_RIGHT`**
- **`STOP_ITERATION`**

Deprecated event

- `BRANCH`

The `BRANCH` event is deprecated in 3.14. Using **`BRANCH_LEFT`** and **`BRANCH_RIGHT`** events will give much better performance as they can be disabled independently.

Eventos auxiliares

Los eventos auxiliares se pueden monitorear como otros eventos, pero están controlados por otros eventos:

- `C_RAISE`
- `C_RETURN`

The `C_RETURN` and `C_RAISE` events are controlled by the `CALL` event. `C_RETURN` and `C_RAISE` events will only be seen if the corresponding `CALL` event is being monitored.

Otros eventos

Other events are not necessarily tied to a specific location in the program and cannot be individually disabled via `DISABLE`.

Los otros eventos que se pueden monitorear son:

- `PY_THROW`
- `PY_UNWIND`
- `RAISE`
- `EXCEPTION_HANDLED`

El evento STOP_ITERATION

PEP 380 [https://peps.python.org/pep-0380/#use-of-stopiteration-to-return-values] specifies that a `StopIteration` exception is raised when returning a value from a generator or coroutine. However, this is a very inefficient way to return a value, so some Python implementations, notably CPython 3.12+, do not raise an exception unless it would be visible to other code.

To allow tools to monitor for real exceptions without slowing down generators and coroutines, the `STOP_ITERATION` event is provided. `STOP_ITERATION` can be locally disabled, unlike `RAISE`.

Note that the `STOP_ITERATION` event and the `RAISE` event for a `StopIteration` exception are equivalent, and are treated as interchangeable when generating events. Implementations will favor `STOP_ITERATION` for performance reasons, but may generate a `RAISE` event with a `StopIteration`.

Activar y desactivar eventos

In order to monitor an event, it must be turned on and a corresponding callback must be registered. Events can be turned on or off by setting the events either globally and/or for a particular code object. An event will trigger only once, even if it is turned on both globally and locally.

Configuración de eventos globalmente

Los eventos se pueden controlar globalmente modificando el conjunto de eventos que están siendo monitoreados.

sys.monitoring.get_events(*tool_id: int*, /) → int

Retorna el `int` que representa todos los eventos activos.

sys.monitoring.set_events(*tool_id: int*, *event_set: int*, /) → None

Activates all events which are set in *event_set*. Raises a `ValueError` if *tool_id* is not in use.

No hay eventos activos de forma predeterminada.

Eventos por objeto de código

Events can also be controlled on a per code object basis. The functions defined below which accept a `types.CodeType` should be prepared to accept a look-alike object from functions which are not defined in Python (see Monitoring C API).

sys.monitoring.get_local_events(*tool_id: int*, *code: CodeType*, /) → int

Returns all the local events for *code*

sys.monitoring.set_local_events(*tool_id: int*, *code: CodeType*, *event_set: int*, /) → None

Activates all the local events for *code* which are set in *event_set*. Raises a `ValueError` if *tool_id* is not in use.

Deshabilitando eventos

sys.monitoring.DISABLE

> A special value that can be returned from a callback function to disable events for the current code location.

Local events can be disabled for a specific code location by returning **`sys.monitoring.DISABLE`** from a callback function. This does not change which events are set, or any other code locations for the same event.

Deshabilitar eventos para ubicaciones específicas es muy importante para el monitoreo de alto rendimiento. Por ejemplo, un programa se puede ejecutar con un depurador sin gastos adicionales si el depurador desactiva toda la supervisión excepto algunos puntos de interrupción.

If **`DISABLE`** is returned by a callback for a global event, **`ValueError`** will be raised by the interpreter in a non-specific location (that is, no traceback will be provided).

sys.monitoring.restart_events() → None

> Enable all the events that were disabled by **`sys.monitoring.DISABLE`** for all tools.

Registrando funciones de retrollamada

sys.monitoring.register_callback(*tool_id: int*, *event: int*, *func: Callable | None*, /) → Callable | None

> Registers the callable *func* for the *event* with the given *tool_id*
>
> If another callback was registered for the given *tool_id* and *event*, it is unregistered and returned. Otherwise **`register_callback()`** returns `None`.
>
> Raises an auditing event `sys.monitoring.register_callback` with argument `func`.

Funciones pueden ser canceladas llamando `sys.monitoring.register_callback(tool_id, event, None)`.

Las funciones de retrollamada se pueden registrar y cancelar en cualquier momento.

Callbacks are called only once regardless if the event is turned on both globally and locally. As such, if an event could be turned on for both global and local events by your code then the callback needs to be written to handle either trigger.

Argumentos de la función de retrollamada

sys.monitoring.MISSING

> A special value that is passed to a callback function to indicate that there are no arguments to the call.

When an active event occurs, the registered callback function is called. Callback functions returning an object other than **`DISABLE`** will have no effect. Different events will provide the callback function with different arguments, as follows:

- **`PY_START`** and **`PY_RESUME`**:

  ```
  func(code: CodeType, instruction_offset: int) -> ob
  ```

- **`PY_RETURN`** and **`PY_YIELD`**:

  ```
  func(code: CodeType, instruction_offset: int, retva
  ```

- **`CALL`**, **`C_RAISE`** and **`C_RETURN`** (*arg0* can be **`MISSING`** specifically):

  ```
  func(code: CodeType, instruction_offset: int, calla
  ```

  code represents the code object where the call is being made, while *callable* is the object that is about to be called (and thus triggered the event). If there are no arguments, *arg0* is set to **`sys.monitoring.MISSING`**.

For instance methods, *callable* will be the function object as found on the class with *arg0* set to the instance (i.e. the `self` argument to the method).

- **`RAISE`**, **`RERAISE`**, **`EXCEPTION_HANDLED`**, **`PY_UNWIND`**, **`PY_THROW`** and **`STOP_ITERATION`**:

  ```
  func(code: CodeType, instruction_offset: int, excep
  ```

- **`LINE`**:

  ```
  func(code: CodeType, line_number: int) -> object
  ```

- **`BRANCH_LEFT`**, **`BRANCH_RIGHT`** and **`JUMP`**:

  ```
  func(code: CodeType, instruction_offset: int, desti
  ```

  Note that the *destination_offset* is where the code will next execute.

- **`INSTRUCTION`**:

  ```
  func(code: CodeType, instruction_offset: int) -> ob
  ```

`sysconfig` — Provide access to Python's configuration information

Added in version 3.2.

Source code: Lib/sysconfig [https://github.com/python/cpython/tree/3.14/Lib/sysconfig]

The **`sysconfig`** module provides access to Python's configuration information like the list of installation paths and the configuration variables relevant for the current platform.

Variables de configuración

A Python distribution contains a `Makefile` and a `pyconfig.h` header file that are necessary to build both the Python binary itself and third-party C extensions compiled using `setuptools`.

`sysconfig` puts all variables found in these files in a dictionary that can be accessed using **`get_config_vars()`** or **`get_config_var()`**.

Tenga en cuenta que en Windows, es un conjunto mucho más pequeño.

sysconfig.get_config_vars(**args*)

> Sin argumentos, retorna un diccionario de todas las variables de configuración relevantes para la plataforma actual.
>
> Con argumentos, retorna un lista de valores que resultan de buscar cada argumento en el diccionario de variables de configuración.

Por cada argumento, si no se encuentra el valor, retorna `None`.

sysconfig.get_config_var(*name*)

Retorna el valor de un solo nombre de variable (*name*). Equivalente a `get_config_vars().get(name)`.

Si no se encuentra *name*, retorna `None`.

Ejemplos de uso:

```
>>> import sysconfig
>>> sysconfig.get_config_var('Py_ENABLE_SHARED')
0
>>> sysconfig.get_config_var('LIBDIR')
'/usr/local/lib'
>>> sysconfig.get_config_vars('AR', 'CXX')
['ar', 'g++']
```

Rutas de instalación

Python uses an installation scheme that differs depending on the platform and on the installation options. These schemes are stored in **`sysconfig`** under unique identifiers based on the value returned by **`os.name`**. The schemes are used by package installers to determine where to copy files to.

Python actualmente admite nueve esquemas:

- *posix_prefix*: esquema para plataformas POSIX como Linux o macOS. Este es el esquema predeterminado que se usa cuando se instala Python o un componente.
- *posix_home*: scheme for POSIX platforms, when the *home* option is used. This scheme defines paths located under a specific home prefix.
- *posix_user*: scheme for POSIX platforms, when the *user* option is used. This scheme defines paths located under the user's home directory (**`site.USER_BASE`**).
- *posix_venv*: scheme for **`Python virtual environments`**

on POSIX platforms; by default it is the same as *posix_prefix*.

- *nt*: scheme for Windows. This is the default scheme used when Python or a component is installed.
- *nt_user*: scheme for Windows, when the *user* option is used.
- *nt_venv*: scheme for **`Python virtual environments`** on Windows; by default it is the same as *nt*.
- *venv*: a scheme with values from either *posix_venv* or *nt_venv* depending on the platform Python runs on.
- *nt_user*: esquema para macOS, cuando se usa la opción *user*.

Cada esquema está compuesto por una serie de rutas y cada ruta tiene un identificador único. Python actualmente usa ocho rutas:

- *stdlib*: directorio que contiene los archivos de la biblioteca estándar de Python que no son específicos de la plataforma.
- *platstdlib*: directorio que contiene los archivos de la biblioteca estándar de Python que son específicos de la plataforma.
- *platlib*: directorio para archivos específicos del sitio, específicos de la plataforma.
- *purelib*: directory for site-specific, non-platform-specific files ("pure" Python).
- *include*: directorio para plataformas no especificadas para archivos de encabezado de Python C-API.
- *include*: directorio para plataformas especificadas para archivos de encabezado de Python C-API.
- *scripts*: directorio para archivos de script.
- *data*: directorio para archivos de datos.

User scheme

This scheme is designed to be the most convenient solution for users that don't have write permission to the global site-packages directory or don't want to install into it.

Files will be installed into subdirectories of **`site.USER_BASE`** (written as `userbase` hereafter). This scheme installs pure Python modules and extension modules in the same location (also known as **`site.USER_SITE`**).

`posix_user`

Path	Installation directory
stdlib	`userbase/lib/pythonX.Y`
platstdlib	`userbase/lib/pythonX.Y`
platlib	`userbase/lib/pythonX.Y/site-packages`
purelib	`userbase/lib/pythonX.Y/site-packages`
include	`userbase/include/pythonX.Y`
scripts	`userbase/bin`
data	`userbase`

`nt_user`

Path	Installation directory
stdlib	`userbase\PythonXY`
platstdlib	`userbase\PythonXY`
platlib	`userbase\PythonXY\site-packages`
purelib	`userbase\PythonXY\site-packages`
include	`userbase\PythonXY\Include`
scripts	`userbase\PythonXY\Scripts`
data	`userbase`

`osx_framework_user`

Path	Installation directory
stdlib	`userbase/lib/python`
platstdlib	`userbase/lib/python`
platlib	`userbase/lib/python/site-packages`
purelib	`userbase/lib/python/site-packages`
include	`userbase/include/pythonX.Y`
scripts	`userbase/bin`
data	`userbase`

Home scheme

The idea behind the «home scheme» is that you build and maintain a personal stash of Python modules. This scheme's name is derived from the idea of a «home» directory on Unix, since it's not unusual for a Unix user to make their home directory have a layout similar to `/usr/` or `/usr/local/`. This scheme can be used by anyone, regardless of the operating system they are installing for.

`posix_home`

Path	Installation directory
stdlib	`home/lib/python`
platstdlib	`home/lib/python`
platlib	`home/lib/python`
purelib	`home/lib/python`
include	`home/include/python`
platinclude	`home/include/python`
scripts	`home/bin`
data	`home`

Prefix scheme

The «prefix scheme» is useful when you wish to use one Python installation to perform the build/install (i.e., to run the setup script), but install modules into the third-party module directory of a different Python installation (or something that looks like a different Python installation). If this sounds a trifle unusual, it is—that's why the user and home schemes come before. However, there are at least two known cases where the prefix scheme will be useful.

First, consider that many Linux distributions put Python in `/usr`, rather than the more traditional `/usr/local`. This is entirely appropriate, since in those cases Python is part of «the system» rather than a local add-on. However, if you are installing Python modules from source, you probably want them to go in `/usr/local/lib/python2.X` rather than `/usr/lib/python2.X`.

Another possibility is a network filesystem where the name used to write to a remote directory is different from the name used to read it: for example, the Python interpreter accessed as `/usr/local/bin/python` might search for modules in `/usr/local/lib/python2.X`, but those modules would have to be installed to, say, `/mnt/@server/export/lib/python2.X`.

`posix_prefix`

Path	Installation directory

stdlib	`prefix/lib/pythonX.Y`
platstdlib	`prefix/lib/pythonX.Y`
platlib	`prefix/lib/pythonX.Y/site-packages`
purelib	`prefix/lib/pythonX.Y/site-packages`
include	`prefix/include/pythonX.Y`
platinclude	`prefix/include/pythonX.Y`
scripts	`prefix/bin`
data	`prefix`

nt

Path	**Installation directory**
stdlib	`prefix\Lib`
platstdlib	`prefix\Lib`
platlib	`prefix\Lib\site-packages`
purelib	`prefix\Lib\site-packages`
include	`prefix\Include`
platinclude	`prefix\Include`
scripts	`prefix\Scripts`
data	`prefix`

Installation path functions

`sysconfig` provides some functions to determine these installation paths.

sysconfig.get_scheme_names()

Return a tuple containing all schemes currently supported in **`sysconfig`**.

sysconfig.get_default_scheme()

Retorna el nombre del esquema predeterminado para la plataforma actual.

Added in version 3.10: Esta función se llamaba anteriormente `_get_default_scheme()` y se consideraba un detalle de implementación.

Distinto en la versión 3.11: Cuando Python se ejecuta desde un entorno virtual, este retorna el esquema *venv*.

sysconfig.get_preferred_scheme(*key*)

Retorna un nombre de esquema preferido para un diseño de instalación especificado por *key*.

key debe ser `"prefix"`, `"home"` o `"user"`.

The return value is a scheme name listed in **`get_scheme_names()`**. It can be passed to **`sysconfig`** functions that take a *scheme* argument, such as **`get_paths()`**.

Added in version 3.10.

Distinto en la versión 3.11: Cuando Python se ejecuta desde un entorno virtual con `key="prefix"`, este retorna el esquema *venv*.

sysconfig._get_preferred_schemes()

Retorna un diccionario que contiene los nombres de los esquemas preferidos en la plataforma actual. Los implementadores y redistribuidores de Python pueden agregar sus esquemas preferidos al valor global de nivel de módulo `_INSTALL_SCHEMES` y modificar esta función para devolver esos nombres de esquema, p. Ej. proporcionan diferentes esquemas para que los utilicen los administradores de paquetes de idioma y sistema, de modo que los paquetes instalados por uno no se mezclen con los del otro.

End users should not use this function, but **`get_default_scheme()`** and **`get_preferred_scheme()`** instead.

Added in version 3.10.

sysconfig.get_path_names()

Return a tuple containing all path names currently supported in **`sysconfig`**.

sysconfig.get_path(*name*[, *scheme*[, *vars*[, *expand*]]])

Retorna una ruta de instalación correspondiente a la ruta *name*, del esquema de instalación denominado *scheme*.

name tiene que ser un valor de la lista retornado por **`get_path_names()`**.

`sysconfig` stores installation paths corresponding to each path name, for each platform, with variables to be expanded. For instance the *stdlib* path for the *nt* scheme is: `{base}/Lib`.

`get_path()` utilizará las variables retornadas por **`get_config_vars()`** para expandir la ruta. Todas las variables tienen valores predeterminados para cada plataforma, por lo que se puede llamar a esta función y obtener el valor predeterminado.

Si se proporciona el esquema (*scheme*), debe ser un valor de la lista retornada por **`get_scheme_names()`**. De lo contrario, se utiliza el esquema predeterminado para la plataforma actual.

If *vars* is provided, it must be a dictionary of variables that will update the dictionary returned by **`get_config_vars()`**.

Si *expand* se establece en `False`, la ruta no se expandirá usando las variables.

Si no se encuentra *name*, lanza un **`KeyError`**.

sysconfig.get_paths([*scheme*[, *vars*[, *expand*]]])

Retorna un diccionario de contiene todas las rutas de instalación correspondientes a un esquema de instalación. Consulte **`get_path()`** para obtener más información.

Si no se proporciona el esquema (*scheme*), utilizará el esquema predeterminado para la plataforma actual.

Si se proporciona *vars*, debe ser un diccionario de variables que actualizará el diccionario utilizado para expandir las rutas.

Si *expand* se establece en falso, las rutas no se expandirán.

Si *scheme* no es un esquema existente, **`get_paths()`** lanzará un **`KeyError`**.

Otras funciones

sysconfig.get_python_version()

Retorna el número de versión versión `MAJOR.MINOR` de Python como una cadena. Similar a `'%d.%d' % sys.version_info[:2]`.

sysconfig.get_platform()

Retorna una cadena que identifica la plataforma actual.

This is used mainly to distinguish platform-specific build directories and platform-specific built distributions. Typically includes the OS name and version and the architecture (as supplied by **`os.uname()`**), although the exact information included depends on the OS; e.g., on Linux, the kernel version isn't particularly important.

Ejemplo de valores retornados:

Windows:

- win-amd64 (64-bit Windows on AMD64, aka x86_64, Intel64, and EM64T)
- win-arm64 (64-bit Windows on ARM64, aka AArch64)
- win32 (todos los demás - específicamente se retorna sys.platform)

POSIX based OS:

- linux-x86_64
- macosx-15.5-arm64

- macosx-26.0-universal2 (macOS on Apple Silicon or Intel)
- android-24-arm64_v8a

Para otras plataformas que no son POSIX, actualmente solo retorna **`sys.platform`**.

sysconfig.is_python_build()

Retorna `True` si el intérprete de Python en ejecución se compiló a partir de la fuente y se está ejecutando desde su ubicación compilada, y no desde una ubicación resultante de por ejemplo ejecutando `make install` o instalando a través de un instalador binario.

sysconfig.parse_config_h(*fp*[, *vars*])

Analiza un archivo de estilo `config.h`.

fp es un objeto similar a un archivo que apunta al archivo similar a `config.h`.

Se retorna un diccionario que contiene pares de nombre/valor. Si se pasa un diccionario opcional como un segundo argumento, se utiliza en lugar de un nuevo diccionario y se actualiza con los valores leídos en el archivo.

sysconfig.get_config_h_filename()

Retorna la ruta de `pyconfig.h`.

sysconfig.get_makefile_filename()

Retorna la ruta de `Makefile`.

Command-line usage

You can use **`sysconfig`** as a script with Python's *-m* option:

```
$ python -m sysconfig
Platform: "macosx-10.4-i386"
Python version: "3.2"
```

```
Current installation scheme: "posix_prefix"

Paths:
        data = "/usr/local"
        include = "/Users/tarek/Dev/svn.python.org/py3k/
        platinclude = "."
        platlib = "/usr/local/lib/python3.2/site-package
        platstdlib = "/usr/local/lib/python3.2"
        purelib = "/usr/local/lib/python3.2/site-package
        scripts = "/usr/local/bin"
        stdlib = "/usr/local/lib/python3.2"

Variables:
        AC_APPLE_UNIVERSAL_BUILD = "0"
        AIX_GENUINE_CPLUSPLUS = "0"
        AR = "ar"
        ARFLAGS = "rc"
        ...
```

Esta llamada imprimirá en la salida estándar la información retornada por **`get_platform()`**, **`get_python_version()`**, **`get_path()`** y **`get_config_vars()`**.

`builtins` — Built-in objects

This module provides direct access to all “built-in” identifiers of Python; for example, `builtins.open` is the full name for the built-in function **`open()`**.

Este módulo normalmente no es accedido explícitamente por la mayoría de las aplicaciones, pero puede ser útil en módulos que provean objetos con el mismo nombre que un valor incorporado, pero que sea necesario también el incorporado con ese nombre. Por ejemplo, en un módulo que quiera implementar una función **`open()`** que envuelva la integrada **`open()`**, este módulo puede se usado directamente:

```
import builtins

def open(path):
    f = builtins.open(path, 'r')
    return UpperCaser(f)

class UpperCaser:
    '''Wrapper around a file that converts output to upp

    def __init__(self, f):
        self._f = f

    def read(self, count=-1):
        return self._f.read(count).upper()

    # ...
```

Como un detalle de implementación, la mayoría de los módulos tienen el nombre `__builtins__` disponible como parte de sus globales. El valor de `__builtins__` es normalmente o este módulo o el valor del atributo **`__dict__`** de este módulo. Como este es un detalle de implementación, puede que no sea usado por implementaciones alternativas de Python.

Ver también

- Constantes incorporadas
- Excepciones incorporadas
- Funciones incorporadas
- Tipos integrados

`__main__` — Top-level code environment

En Python, el nombre especial `__main__` es utilizado para dos constructos importantes:

1. el nombre del entorno de máximo nivel del programa, que puede ser verificado usando la expresión `__name__=='__main__'`; y
2. el archivo `__main__.py` en paquetes de Python.

Ambos de estos mecanismos están relacionados con módulos de Python; como los usuarios interactúan con ellos y como ellos interactúan entre sí. Están explicados en detalle más abajo. Si estás empezando con los módulos de Python, mira la sección tutorial Módulos para una introducción.

`__name__ == '__main__'`

Cuando un módulo o paquete de Python es importado, `__name__` es asignado al nombre del módulo. Normalmente, este es el nombre del archivo de Python sin la extensión `.py`:

```
>>> import configparser
>>> configparser.__name__
'configparser'
```

Si el archivo es parte de un paquete, `__name__` también incluirá la ruta del paquete padre:

```
>>> from concurrent.futures import process
>>> process.__name__
'concurrent.futures.process'
```

Sin embargo, si el módulo es ejecutado en el entorno de código de máximo nivel, su `__name__` se le asigna el valor del string `'__main__'`.

¿Qué es el «entorno de código de máximo nivel»?

`__main__` es el nombre del entorno en el cual se ejecuta el código de máximo nivel. «Código de máximo nivel» es el primer módulo de Python especificado por el usuario que empieza a ejecutarse. Es «de máximo nivel» porque importa todos los demás módulos que necesita el programa. A veces «código de máximo nivel» es llamado un *punto de entrada* a la aplicación.

El entorno de código de máximo nivel puede ser:

- el ámbito de un intérprete interactivo:

  ```
  >>> __name__
  '__main__'
  ```

- el módulo de Python pasado al intérprete de Python como un argumento de archivo:

  ```
  $ python helloworld.py
  Hello, world!
  ```

- el módulo o paquete de Python pasado al interprete de Python con el argumento **`-m`**:

  ```
  $ python -m tarfile
  usage: tarfile.py [-h] [-v] (...)
  ```

- Código de Python leído por el interprete desde input estándar:

  ```
  $ echo "import this" | python
  The Zen of Python, by Tim Peters

  Beautiful is better than ugly.
  Explicit is better than implicit.
  ...
  ```

- Código de Python pasado al intérprete Python con el argumento **`-c`**:

  ```
  $ python -c "import this"
  The Zen of Python, by Tim Peters

  Beautiful is better than ugly.
  Explicit is better than implicit.
  ...
  ```

En cada una de estas situaciones, al `__name__` del módulo de máximo nivel se le asigna el valor `'__main__'`.

En consecuencia, un módulo puede descubrir si se está ejecutando o no en el ámbito principal al verificar su propio `__name__`, lo cual permite un vocablo común para ejecutar código condicionalmente cuando el módulo no es inicializado desde una declaración de importado:

```
if __name__ == '__main__':
    # Execute when the module is not initialized from ar
    ...
```

Ver también

Para una vista más detallada de como `__name__` es asignado en todas las situaciones, ver la sección tutorial Módulos.

Uso idiomático

Algunos módulos contienen código que está pensado para uso de script solamente, como para interpretar argumentos de línea de comando u obtener datos de la entrada estándar. Cuando un módulo como este fuese importado desde un módulo distinto, por ejemplo para realizar una prueba unitaria, el código del script se ejecutaría involuntariamente.

Aquí es donde es útil usar el código de bloque `if __name__=='__main__'`. El código dentro desde bloque no se ejecutará a menos que el módulo se ejecute en el entorno de máximo nivel.

Poner cuantas menos declaraciones posible en el bloque debajo de `if __name__=='__main__'` puede mejorar la claridad y validez del código. Mas a menudo, la función llamada `main` encapsula el comportamiento principal del programa:

```
# echo.py

import shlex
import sys

def echo(phrase: str) -> None:
   """A dummy wrapper around print."""
   # for demonstration purposes, you can imagine that th
   # valuable and reusable logic inside this function
   print(phrase)

def main() -> int:
    """Echo the input arguments to standard output"""
    phrase = shlex.join(sys.argv)
    echo(phrase)
    return 0

if __name__ == '__main__':
    sys.exit(main())  # next section explains the use of
```

Note que si el módulo no encapsulase el código dentro de la función `main` pero en vez lo pusiese directo dentro del bloque `if __name__=='__main__'`, la variable `phrase` sería global al módulo entero. Esto está propenso a generar errores ya que otras funciones dentro del módulo pudiesen estar usando involuntariamente la variable global en lugar de un nombre local. Una función `main` resuelve este problema.

Usar una función `main` tiene el beneficio añadido de que la función `echo` está aislada y es importable en otros sitios. Cuando `echo.py` es importado, las funciones `echo` y `main` serán definidas, pero ninguna de ellas será llamada porque `__name!='__main__'`.

Consideraciones de empaquetado

Las funciones `main` se utilizan a menudo para crear líneas de comando al especificarlas como puntos de entrada para scripts de terminal. Cuando esto se hace, pip [https://pip.pypa.io/] inserta la llamada de la función a un script plantilla, donde el valor retornado de `main` se pasa a **`sys.exit()`**. Por ejemplo:

```
sys.exit(main())
```

Dado que la llamada a `main` está dentro de **`sys.exit()`**, la expectativa es que tu función devolverá un valor aceptable como una entrada a **`sys.exit()`**; típicamente, un int o `None` (que se retorna implícitamente si tu función no tiene una declaración de retorno).

Al seguir pro-activamente esta convención nosotros mismo, nuestro módulo tendrá el mismo comportamiento cuando se ejecuta directamente (es decir, `python echo.py`) que si luego lo empaquetamos como un punto de entrada de script de terminal en un paquete instalable mediante pip.

En particular, ten cuidado al devolver cadenas de texto con tu función `main`. **`sys.exit()`** interpretará un argumento de cadena de texto como un mensaje de fallo, entonces tu programa tendrá un código de salida de `1`, indicando fallo, y la cadena de texto será escrita a **`sys.stderr`**. El ejemplo `echo.py` de antes muestra como usar la convención `sys.exit(main())`.

Ver también

Python Packaging User Guide [https://packaging.python.org/] contiene una colección de tutoriales y referencias sobre como distribuir e instalar paquetes de Python con herramientas modernas.

`__main__.py` en paquetes de Python

Si no estás familiarizado con paquetes de Python, ver la sección Paquetes del tutorial. Comúnmente, el archivo `__main__.py` es utilizado para proveer una interfaz de línea de comando para un paquete. Considera el siguiente paquete hipotético, «*bandclass*»:

```
bandclass
      __init__.py
      __main__.py
      student.py
```

`__main__.py` será ejecutado cuando el paquete sea invocado directamente desde la línea de comandos usando el indicador **-m**. Por ejemplo:

```
$ python -m bandclass
```

Este comando causará que `__main__.py` se ejecute. El como se use este mecanismo dependerá de la naturaleza del paquete que estás escribiendo, pero en este caso hipotético, puede tener sentido permitir que el profesor busque estudiantes:

```
# bandclass/__main__.py

import sys
from .student import search_students

student_name = sys.argv[1] if len(sys.argv) >= 2 else ''
print(f'Found student: {search_students(student_name)}')
```

Note que `from .student import search_students` es un ejemplo de una importación relativa. Este estilo de importación debe ser utilizado cuando se hace referencia a módulos dentro de un paquete. Para más detalles, ver Referencias internas en paquetes en la sección Módulos del tutorial.

Uso idiomático

Los contenidos de `__main__.py` no están típicamente acotados dentro de bloques `if __name__=='__main__'`. En cambio, esos archivos se mantienen cortos e importan funciones para ejecutar desde otros módulos. A esos otros módulos se les puede fácilmente realizar pruebas unitarias y son apropiadamente re-utilizables.

Si se usa, un bloque `if __name__=='__main__'` seguirá funcionando como se espera para un archivo `__main__.py` dentro de un paquete, porque su atributo `__name__` incluirá la ruta del paquete si es importado:

```
>>> import asyncio.__main__
>>> asyncio.__main__.__name__
'asyncio.__main__'
```

This won't work for `__main__.py` files in the root directory of a `.zip` file though. Hence, for consistency, a minimal `__main__.py` without a `__name__` check is preferred.

Ver también

En **`venv`** puedes conseguir un ejemplo de un paquete con un `__main__.py` minimalista en la librería estándar. No contiene un bloque `if __name__=='__main__'`. Lo puedes invocar con `python -m venv [directorio]`.

Ver **`runpy`** para más detalles sobre el indicador **`-m`** para el interprete ejecutable.

Ver **`zipapp`** para más información sobre como ejecutar aplicaciones empaquetadas como archivos *.zip*. En este caso Python busca un archivo `__main__.py` en el directorio raíz del archivo comprimido.

`import __main__`

Independientemente de con cual módulo se ha iniciado un programa de Python, otros módulos que están siendo ejecutados dentro del mismo programa pueden importar el ámbito del entorno de máximo nivel (namespace) al importar el módulo `__main__`. Esto no importa un archivo `__main__.py` pero en su lugar cualquier módulo que recibió el nombre especial `'__main__'`.

Acá hay un módulo ejemplo que consume el nombre de espacio `__main__`:

```
# namely.py

import __main__

def did_user_define_their_name():
    return 'my_name' in dir(__main__)

def print_user_name():
    if not did_user_define_their_name():
        raise ValueError('Define the variable `my_name`!

    print(__main__.my_name)
```

Ejemplo del uso de este módulo puede ser:

```
# start.py

import sys

from namely import print_user_name

# my_name = "Dinsdale"

def main():
    try:
        print_user_name()
    except ValueError as ve:
        return str(ve)

if __name__ == "__main__":
    sys.exit(main())
```

Si ahora iniciamos nuestro programa el resultado sería así:

```
$ python start.py
Define the variable `my_name`!
```

El código de salida del programa sería 1, indicando un error. Descomentando la línea con `my_name = "Dinsdale"` arregla el programa y ahora sale con un código de estado 0, indicando éxito:

```
$ python start.py
Dinsdale
```

Note que importar `__main__` no causa ningún problema de involuntariamente ejecutar código de máximo nivel que ha sido pensado para uso por scripts que es puesto en el bloque `if __name__=="__main__"` del módulo `start`. ¿Por qué funciona esto?

Python inserta un módulo `__main__` vacío en **`sys.modules`** al inicio del intérprete, y lo puebla ejecutando código de máximo nivel. En nuestro ejemplo este es el módulo `start` que corre línea a línea e importa `namely`. A su vez, `namely` importa `__main__` (que es en verdad `start`). ¡Es un ciclo de importado! Afortunadamente, como el módulo parcialmente poblado `__main__` está presente en **`sys.modules`**, Python pasa eso a `namely`. Ver Special considerations for __main__ en la referencia del sistema para información detallada de como funciona.

El REPL Python es otro ejemplo de un «entorno de máximo nivel», por lo tanto, cualquier cosa definida en el REPL se hace parte del ámbito `__main__`:

```
>>> import namely
>>> namely.did_user_define_their_name()
False
>>> namely.print_user_name()
Traceback (most recent call last):
...
ValueError: Define the variable `my_name`!
>>> my_name = 'Jabberwocky'
>>> namely.did_user_define_their_name()
True
>>> namely.print_user_name()
Jabberwocky
```

El ámbito `__main__` es utilizado en la implementación de **`pdb`** y **`rlcompleter`**.

`warnings` — Warning control

Código fuente: Lib/warnings.py [https://github.com/python/cpython/tree/3.14/Lib/warnings.py]

Los mensajes de advertencia suelen emitirse en situaciones en las que es útil alertar al usuario de alguna condición en un programa, cuando esa condición (normalmente) no justifica que se haga una excepción y se termine el programa. Por ejemplo, se puede emitir una advertencia cuando un programa utiliza un módulo obsoleto.

Los programadores de Python emiten advertencias llamando a la función `warn()` definida en este módulo. (Los programadores de C usan `PyErr_WarnEx()`; ver Manejo de excepciones para más detalles)

Los mensajes de advertencia se escriben normalmente en `sys.stderr`, pero su disposición puede cambiarse de manera flexible, desde ignorar todas las advertencias hasta convertirlas en excepciones. La disposición de las advertencias puede variar en función de la warning category, el texto del mensaje de advertencia y la ubicación de la fuente donde se emite. Las repeticiones de una advertencia particular para la misma ubicación de la fuente son típicamente suprimidas.

El control de las advertencias consta de dos etapas: en primer lugar, cada vez que se emite una advertencia, se determina si se debe emitir un mensaje o no; en segundo lugar, si se debe emitir un mensaje, se le da formato y se imprime utilizando un gancho establecido por el usuario.

La determinación de si se debe emitir un mensaje de advertencia está controlada por el warning filter, que es una secuencia de reglas y acciones que coinciden. Se pueden añadir reglas al filtro llamando a `filterwarnings()` y restablecer su estado por defecto llamando a `resetwarnings()`.

La impresión de los mensajes de advertencia se realiza llamando a **`showwarning()`**, que puede ser anulado; la implementación por defecto de esta función da formato al mensaje llamando a **`formatwarning()`**, que también está disponible para su uso en implementaciones personalizadas.

Ver también

`logging.captureWarnings()` permite manejar todas las advertencias con la infraestructura de registro estándar.

Categorías de advertencia

Hay una serie de excepciones incorporadas que representan categorías de advertencia. Esta categorización es útil para poder filtrar grupos de advertencias.

Aunque técnicamente se trata de built-in exceptions, están documentadas aquí, porque pertenecen al mecanismo de advertencias.

El código de usuario puede definir categorías de advertencia adicionales mediante la subclasificación de una de las categorías de advertencia estándar. Una categoría de advertencia siempre debe ser una subclase de la clase **`Warning`** class.

Actualmente están definidas las siguientes clases de categorías de advertencias:

Clase	Descripción
`Warning`	Esta es la clase principal para todas las clases que pertenecen a la categoría de advertencia. Es una subclase de **`Exception`**.
`UserWarning`	La categoría por defecto para **`warn()`**.
`DeprecationWarning`	Categoría principal para advertencias sobre características obsoletas cuando esas advertencias están destinadas a otros desarrolladores de Python (ignoradas por defecto, a menos que sean activadas por el código en `__main__`).
`SyntaxWarning`	Base category for warnings about dubious syntactic features (typically emitted when compiling Python source code, and

	hence may not be suppressed by runtime filters)
`RuntimeWarning`	Categoría principal para las advertencias sobre características dudosas de tiempo de ejecución.
`FutureWarning`	Categoría principal para las advertencias sobre características obsoletas cuando esas advertencias están destinadas a los usuarios finales de aplicaciones escritas en Python.
`PendingDeprecationWarning`	Categoría principal para las advertencias sobre las características que serán desaprobadas en el futuro (ignoradas por defecto).
`ImportWarning`	Categoría principal para las advertencias que se activan durante el proceso de importación de un módulo (se ignoran por defecto).
`UnicodeWarning`	Categoría principal para las advertencias relacionadas con Unicode.
`BytesWarning`	Categoría principal para las advertencias relacionadas con `bytes` y `bytearray`.
`ResourceWarning`	Categoría base para las advertencias relacionadas con el uso de los recursos (ignorada por defecto).

Distinto en la versión 3.7: Anteriormente `DeprecationWarning` y `FutureWarning` se distinguían en función de si una característica se eliminaba por completo o se cambiaba su comportamiento. Ahora se distinguen en función de su público objetivo y de la forma en que son manejados por los filtros de advertencia predeterminados.

El filtro de advertencias

El filtro de advertencias controla si las advertencias se ignoran, se muestran o se convierten en errores (planteando una excepción).

Conceptualmente, el filtro de advertencias mantiene una lista ordenada de especificaciones de filtros; cualquier advertencia específica se compara con cada especificación de filtro de la lista por turno hasta que se encuentra una coincidencia; el filtro determina la disposición de la coincidencia. Cada entrada es una tupla de la forma (*action*, *message*, *category*, *module*, *lineno*), donde:

- *action* es una de las siguientes cadenas:

Valor	**Disposición**
`"default"`	Imprime la primera ocurrencia de advertencias

	coincidentes para cada lugar (módulo + número de línea) donde se emite la advertencia
`"error"`	convertir las advertencias de coincidencia en excepciones
`"ignore"`	nunca imprime advertencias que coincidan
`"always"`	siempre imprime advertencias que coinciden
`"all"`	alias to «always»
`"module"`	imprime la primera ocurrencia de advertencias coincidentes para cada módulo en el que se emite la advertencia (independientemente del número de línea)
`"once"`	imprime sólo la primera aparición de advertencias coincidentes, independientemente de la ubicación

- *message* esa cadena de caracteres que contiene una expresión regular que debería coincidir con el inicio del mensaje de advertencia, sin discriminar mayúsculas ni minúsculas. En **`-W`** y **`PYTHONWARNINGS`**, *message* es una cadena de caracteres literal que debería coincidir con el inicio del mensaje de advertencia discriminando mayúsculas y/o minúsculas, pero ignorando cualquier espacio en blanco al inicio de *message*.

- *category* es una clase (una subclase de **`Warning`**) de la cual la categoría de advertencia debe ser una subclase para poder coincidir.

- *module* es un cadena de caracteres que contiene una expresión regular, la cual debe coincidir exactamente (con mayúsculas y minúsculas) al nombre completo del módulo. En **`-W`** y **`PYTHONWARNINGS`**, *module* es string literal que debe ser igual (con mayúsculas y minúsculas) al nombre completo del módulo, ignorando cualquier espacio en blanco al inicio o final de *module*.

- *lineno* es un número entero que el número de línea donde ocurrió la advertencia debe coincidir, o `0` para coincidir con todos los números de línea.

Como la clase **`Warning`** se deriva de la clase **`Excepción`** incorporada, para convertir una advertencia en un error simplemente lanzamos `category(message)`.

Si se informa de una advertencia y no coincide con ningún filtro registrado, se aplica la acción «por defecto» (de ahí su nombre).

Repeated Warning Suppression Criteria

The filters that suppress repeated warnings apply the following criteria to determine if a warning is considered a repeat:

- `"default"`: A warning is considered a repeat only if the (*message*, *category*, *module*, *lineno*) are all the same.
- `"module"`: A warning is considered a repeat if the (*message*, *category*, *module*) are the same, ignoring the line number.
- `"once"`: A warning is considered a repeat if the (*message*, *category*) are the same, ignoring the module and line number.

Descripción de los filtros de advertencia

The warnings filter is initialized by **`-W`** options passed to the Python interpreter command line and the **`PYTHONWARNINGS`** environment variable. The interpreter saves the arguments for all supplied entries without interpretation in **`sys.warnoptions`**; the **`warnings`** module parses these when it is first imported (invalid options are ignored, after printing a message to **`sys.stderr`**).

Los filtros de advertencia individuales se especifican como una secuencia de campos separados por dos puntos:

```
action:message:category:module:line
```

El significado de cada uno de estos campos es como se describe en El filtro de advertencias. Cuando se enumeran varios filtros en una sola línea (como en **`PYTHONWARNINGS`**), los filtros individuales se separan por comas y los filtros que se enumeran más tarde tienen prioridad sobre los que se enumeran antes de ellos (ya que se aplican de izquierda a derecha, y los filtros aplicados más recientemente tienen prioridad sobre los anteriores).

Los filtros de advertencia comúnmente utilizados se aplican a todas las advertencias, a las advertencias de una categoría particular o a las advertencias planteadas por módulos o paquetes particulares. Algunos ejemplos:

```
default                      # Show all warnings (even t
ignore                       # Ignore all warnings
error                        # Convert all warnings to e
error::ResourceWarning       # Treat ResourceWarning mes
default::DeprecationWarning  # Show DeprecationWarning m
ignore,default:::mymodule    # Only report warnings trig
error:::mymodule             # Convert warnings to error
```

Filtro de advertencia predeterminado

Por defecto, Python instala varios filtros de advertencia, que pueden ser anulados por la opción de línea de comandos `-W`, la variable de entorno `PYTHONWARNINGS` y llamadas a `filterwarnings()`.

En versiones regulares, el filtro de advertencia por defecto tiene las siguientes entradas (en orden de precedencia):

```
default::DeprecationWarning:__main__
ignore::DeprecationWarning
ignore::PendingDeprecationWarning
ignore::ImportWarning
ignore::ResourceWarning
```

En una compilación de depuración, la lista de filtros de advertencia predeterminados está vacía.

Distinto en la versión 3.2: `DeprecationWarning` es ahora ignorado por defecto además de `PendingDeprecationWarning`.

Distinto en la versión 3.7: `DeprecationWarning` se muestra de nuevo por defecto cuando se activa directamente por código en `__main__`.

Distinto en la versión 3.7: `BytesWarning` ya no aparece en la lista de filtros por defecto y en su lugar se configura a través de `sys.warnoptions` cuando `-b` se especifica dos veces.

Anulando el filtro por defecto

Los desarrolladores de aplicaciones escritas en Python pueden desear ocultar *todas* las advertencias de nivel de Python a sus usuarios por defecto, y sólo mostrarlas cuando se realizan pruebas o se trabaja de otra manera en la aplicación. El atributo **`sys.warnoptions`** utilizado para pasar las configuraciones de los filtros al intérprete puede utilizarse como marcador para indicar si las advertencias deben ser desactivadas o no:

```
import sys

if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
```

Se aconseja a los desarrolladores de pruebas de código Python que se aseguren de que *todas* las advertencias se muestren por defecto para el código que se está probando, usando un código como:

```
import sys

if not sys.warnoptions:
    import os, warnings
    warnings.simplefilter("default") # Change the filter
    os.environ["PYTHONWARNINGS"] = "default" # Also affe
```

Por último, se aconseja a los desarrolladores de shells interactivos que ejecuten el código de usuario en un espacio de nombres distinto de `__main__` que se aseguren de que los mensajes **`DeprecationWarning`** se hagan visibles por defecto, utilizando un código como el siguiente (donde `user_ns` es el módulo utilizado para ejecutar el código introducido interactivamente):

```
import warnings
warnings.filterwarnings("default", category=DeprecationW
                                   module=user_ns.get("_
```

Eliminación temporal de las advertencias

Si está utilizando un código que sabe que va a provocar una advertencia, como una función obsoleta, pero no quiere ver la advertencia (incluso cuando las advertencias se han configurado explícitamente a través de la línea de comandos), entonces es posible suprimir la advertencia utilizando el gestor de contexto **`catch_warnings`**:

```
import warnings

def fxn():
    warnings.warn("deprecated", DeprecationWarning)

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    fxn()
```

While within the context manager all warnings will simply be ignored. This allows you to use known-deprecated code without having to see the warning while not suppressing the warning for other code that might not be aware of its use of deprecated code.

> **Nota**
>
> See Concurrent safety of Context Managers for details on the concurrency-safety of the **`catch_warnings`** context manager when used in programs using multiple threads or async functions.

Advertencias de prueba

Para probar las advertencias planteadas por el código, use el administrador de contexto **`catch_warnings`**. Con él puedes mutar temporalmente el filtro de advertencias para facilitar tus pruebas. Por ejemplo, haz lo siguiente para capturar todas las advertencias levantadas para comprobar:

```
import warnings

def fxn():
    warnings.warn("deprecated", DeprecationWarning)

with warnings.catch_warnings(record=True) as w:
    # Cause all warnings to always be triggered.
    warnings.simplefilter("always")
    # Trigger a warning.
    fxn()
    # Verify some things
    assert len(w) == 1
    assert issubclass(w[-1].category, DeprecationWarning
    assert "deprecated" in str(w[-1].message)
```

También se puede hacer que todas las advertencias sean excepciones usando `error` en lugar de `always`. Una cosa que hay que tener en cuenta es que si una advertencia ya se ha planteado debido a una regla de `once` o `default`, entonces no importa qué filtros estén establecidos, la advertencia no se verá de nuevo a menos que el registro de advertencias relacionadas con la advertencia se haya borrado.

Once the context manager exits, the warnings filter is restored to its state when the context was entered. This prevents tests from changing the warnings filter in unexpected ways between tests and leading to indeterminate test results.

Nota

See Concurrent safety of Context Managers for details on the concurrency-safety of the **`catch_warnings`** context manager when used in programs using multiple threads or async functions.

Cuando se prueban múltiples operaciones que plantean el mismo tipo de advertencia, es importante probarlas de manera que se confirme que cada operación plantea una nueva advertencia (por ejemplo, establecer advertencias que se planteen como excepciones y comprobar que las operaciones plantean excepciones, comprobar que la longitud de la lista de advertencias siga aumentando después de cada operación, o bien suprimir las entradas anteriores de la lista de advertencias antes de cada nueva operación).

Actualización del código para las nuevas versiones de las dependencias

Las categorías de advertencia que interesan principalmente a los desarrolladores de Python (más que a los usuarios finales de aplicaciones escritas en Python) se ignoran por defecto.

En particular, esta lista de «ignorados por defecto» incluye `DeprecationWarning` (para cada módulo excepto `__main__`), lo que significa que los desarrolladores deben asegurarse de probar su código con advertencias típicamente ignoradas hechas visibles para recibir notificaciones oportunas de futuros cambios del API a última hora(ya sea en la biblioteca estándar o en paquetes de terceros).

En el caso ideal, el código tendrá un conjunto de pruebas adecuado, y el corredor de pruebas se encargará de habilitar implícitamente todas las advertencias cuando se ejecuten las pruebas (el corredor de pruebas proporcionado por el módulo `unittest` hace esto).

En casos menos ideales, las aplicaciones pueden ser revisadas por el uso de interfaces desaprobadas pasando `-Wd` al intérprete de Python (esto es la abreviatura de `-W default`) o ajustando `PYTHONWARNINGS=default` en el entorno. Esto permite el manejo por defecto de todas las advertencias, incluyendo aquellas que son ignoradas por defecto. Para cambiar la acción que se lleva a cabo para las advertencias encontradas puede cambiar el argumento que se pasa a `-W` (por ejemplo `-W error`). Consulte el indicador `-W` para obtener más detalles sobre lo que es posible.

Funciones disponibles

warnings.warn(*message*, *category*=*None*, *stacklevel*=*1*, *source*=*None*, ***, *skip_file_prefixes*=*()*)

Emite una advertencia, o tal vez la ignorar o lanza una excepción. El argumento *category*, si se da, debe ser un warning category class; por defecto es **`UserWarning`**. Alternativamente, *message* puede ser una instancia **`Warning`**, en cuyo caso *category* será ignorada y se usará `message.__class__`. En este caso, el texto del mensaje será `str(message)`. Esta función hace una excepción si la advertencia particular emitida es convertida en un error por el warnings filter. El argumento *stacklevel* puede ser usado por funciones de envoltura escritas en Python, como esta:

```
def deprecated_api(message):
    warnings.warn(message, DeprecationWarning, stac
```

Esto hace que la advertencia se refiera al invocador de `deprecated_api`, en lugar de a la fuente de `deprecated_api` en sí (ya que esta última perdería el propósito del mensaje de advertencia).

El parámetro de palabra clave *skip_file_prefixes* se puede utilizar para indicar qué *frames* de pila se ignoran al contar los niveles de pila. Esto puede ser útil cuando desea que la advertencia aparezca siempre en los sitios de llamadas fuera de un paquete cuando una constante *stacklevel* no se ajusta a todas las rutas de llamadas o en otro caso es difícil de mantener. Si se proporciona, debe ser una tupla de cadenas de caracteres. Cuando se proporcionan prefijos, el *stacklevel* se sobreescribe implícitamente con `max(2, stacklevel)`. Para hacer que se atribuya una advertencia al invocador desde afuera del paquete actual, puede escribir:

```
# example/lower.py
_warn_skips = (os.path.dirname(__file__),)

def one_way(r_luxury_yacht=None, t_wobbler_mangrove
    if r_luxury_yacht:
        warnings.warn("Please migrate to t_wobbler_
                      skip_file_prefixes=_warn_skip

# example/higher.py
from . import lower
```

```
def another_way(**kw):
    lower.one_way(**kw)
```

This makes the warning refer to both the `example.lower.one_way()` and `example.higher.another_way()` call sites only from calling code living outside of `example` package.

source, si se suministra, es el objeto destruido que emitió un **`ResourceWarning`**.

Distinto en la versión 3.6: Añadido parámetro *source*.

Distinto en la versión 3.12: Añadido *skip_file_prefixes*.

warnings.warn_explicit(*message*, *category*, *filename*, *lineno*, *module=None*, *registry=None*, *module_globals=None*, *source=None*)

This is a low-level interface to the functionality of **`warn()`**, passing in explicitly the message, category, filename and line number, and optionally other arguments. *message* must be a string and *category* a subclass of **`Warning`** or *message* may be a **`Warning`** instance, in which case *category* will be ignored.

module, if supplied, should be the module name. If no module is passed, the filename with `.py` stripped is used.

registry, if supplied, should be the `__warningregistry__` dictionary of the module. If no registry is passed, each warning is treated as the first occurrence, that is, filter actions `"default"`, `"module"` and `"once"` are handled as `"always"`.

module_globals, si se suministra, debe ser el espacio de nombres global en uso por el código para el que se emite la advertencia. (Este argumento se utiliza para apoyar la visualización de la fuente de los módulos que se encuentran en los archivos zip o en otras fuentes de importación que no son del sistema de archivos).

source, si se suministra, es el objeto destruido que emitió un **`ResourceWarning`**.

Distinto en la versión 3.6: Añade el parámetro *source*.

warnings.showwarning(*message*, *category*, *filename*, *lineno*, *file=None*, *line=None*)

Escriba una advertencia en un archivo. La implementación por defecto llama `formatwarning(message, category, filename, lineno, line)` y escribe la cadena resultante a *file*, que por defecto es **`sys.stderr`**. Puede reemplazar esta función con cualquier interlocutor asignando a `warnings.showwarning`. *line* es una línea de código fuente que se incluirá en el mensaje de advertencia; si no se proporciona *line*, **`showwarning()`** intentará leer la línea especificada por *nombre de archivo* y *lineno*.

warnings.formatwarning(*message*, *category*, *filename*, *lineno*, *line=None*)

Formatea una advertencia de la manera estándar. Esto retorna una cadena que puede contener nuevas líneas incrustadas y termina en una nueva línea. *line* es una línea de código fuente a incluir en el mensaje de advertencia; si *line* no se suministra, **`formatwarning()`** intentará leer la línea especificada por el nombre de fichero *filename* y *lineno*.

warnings.filterwarnings(*action*, *message=''*, *category=Warning*, *module=''*, *lineno=0*, *append=False*)

Inserta una entrada en la lista de warnings filter specifications. La entrada se inserta por defecto en el frente; si *append* es verdadero, se inserta al final. Esto comprueba los tipos de los argumentos, compila las expresiones regulares *message* y *module*, y las inserta como una tupla en la lista de filtros de aviso. Las entradas más cercanas al principio de la lista anulan las entradas posteriores de la lista, si ambas coinciden con una advertencia en particular. Los argumentos omitidos predeterminan un valor que coincide con todo.

warnings.simplefilter(*action*, *category=Warning*, *lineno=0*, *append=False*)

Inserte una simple entrada en la lista de warnings filter specifications. El significado de los parámetros de la función es el mismo que el de **`filterwarnings()`**, pero no se necesitan expresiones regulares ya que el filtro insertado siempre coincide con cualquier mensaje de cualquier módulo siempre que la categoría y el número de línea coincidan.

warnings.resetwarnings()

Reajusta el filtro de advertencias. Esto descarta el efecto de todas las llamadas previas a **`filterwarnings()`**, incluyendo la de las opciones de línea de comandos de **`-W`** y las llamadas a **`simplefilter()`**.

@warnings.deprecated(*msg*, ***, *category=DeprecationWarning*, *stacklevel=1*)

Decorator to indicate that a class, function or overload is deprecated.

When this decorator is applied to an object, deprecation warnings may be emitted at runtime when the object is used. static type checkers will also generate a diagnostic on usage of the deprecated object.

Usage:

```
from warnings import deprecated
from typing import overload

@deprecated("Use B instead")
class A:
    pass

@deprecated("Use g instead")
def f():
    pass
```

```
@overload
@deprecated("int support is deprecated")
def g(x: int) -> int: ...
@overload
def g(x: str) -> int: ...
```

The warning specified by *category* will be emitted at runtime on use of deprecated objects. For functions, that happens on calls; for classes, on instantiation and on creation of subclasses. If the *category* is `None`, no warning is emitted at runtime. The *stacklevel* determines where the warning is emitted. If it is `1` (the default), the warning is emitted at the direct caller of the deprecated object; if it is higher, it is emitted further up the stack. Static type checker behavior is not affected by the *category* and *stacklevel* arguments.

The deprecation message passed to the decorator is saved in the `__deprecated__` attribute on the decorated object. If applied to an overload, the decorator must be after the **`@~typing.overload`** decorator for the attribute to exist on the overload as returned by **`typing.get_overloads()`**.

Added in version 3.13: See PEP 702 [https://peps.python.org/pep-0702/].

Gestores de contexto disponibles

class warnings.catch_warnings(**, record=False, module=None, action=None, category=Warning, lineno=0, append=False*)

Un gestor de contexto que copia y, al salir, restaura el filtro de advertencias y la función **`showwarning()`**. Si el argumento *record* es **`False`** (por defecto) el gestor de contextos retorna **`None`** al entrar. Si *record* es **`True`**, se retorna una lista que se va poblando progresivamente de objetos como se ve por una función personalizada de **`showwarning()`** (que también suprime la salida a `sys.stdout`). Cada objeto de la lista tiene atributos con los mismos nombres que los argumentos de **`showwarning()`**.

The *module* argument takes a module that will be used instead of the module returned when you import **`warnings`** whose filter will be protected. This argument exists primarily for testing the **`warnings`** module itself.

Si el argumento *action* es diferente a `None`, los demás argumentos serán enviados a la función **`simplefilter()`** como si fueran invocados inmediatamente al entrar al contexto.

See El filtro de advertencias for the meaning of the *category* and *lineno* parameters.

Nota

See Concurrent safety of Context Managers for details on the concurrency-safety of the **`catch_warnings`** context manager when used in programs using multiple threads or async functions.

Distinto en la versión 3.11: Agrega los parámetros *action*, *category*, *lineno* y *append*.

Concurrent safety of Context Managers

The behavior of **`catch_warnings`** context manager depends on the **`sys.flags.context_aware_warnings`** flag. If the flag is true, the context manager behaves in a concurrent-safe fashion and otherwise not. Concurrent-safe means that it is both thread-safe and safe to use within asyncio coroutines and tasks. Being thread-safe means that behavior is predictable in a multi-threaded program. The flag defaults to true for free-threaded builds and false otherwise.

If the **`context_aware_warnings`** flag is false, then **`catch_warnings`** will modify the global attributes of the **`warnings`** module. This is not safe if used within a concurrent program (using multiple threads or using asyncio coroutines). For example, if two or more threads use the **`catch_warnings`** class at the same time, the behavior is undefined.

If the flag is true, `catch_warnings` will not modify global attributes and will instead use a `ContextVar` to store the newly established warning filtering state. A context variable provides thread-local storage and it makes the use of `catch_warnings` thread-safe.

The *record* parameter of the context handler also behaves differently depending on the value of the flag. When *record* is true and the flag is false, the context manager works by replacing and then later restoring the module's `showwarning()` function. That is not concurrent-safe.

When *record* is true and the flag is true, the `showwarning()` function is not replaced. Instead, the recording status is indicated by an internal property in the context variable. In this case, the `showwarning()` function will not be restored when exiting the context handler.

The `context_aware_warnings` flag can be set the `-X context_aware_warnings` command-line option or by the `PYTHON_CONTEXT_AWARE_WARNINGS` environment variable.

Nota

It is likely that most programs that desire thread-safe behaviour of the warnings module will also want to set the `thread_inherit_context` flag to true. That flag causes threads created by `threading.Thread` to start with a copy of the context variables from the thread starting it. When true, the context established by `catch_warnings` in one thread will also apply to new threads started by it. If false, new threads will start with an empty warnings context variable, meaning that any filtering that was established by a `catch_warnings` context manager will no longer be active.

Distinto en la versión 3.14: Added the `sys.flags.context_aware_warnings` flag and the use of a context variable for `catch_warnings` if the flag is true. Previous versions of Python acted as if the flag was always set to false.

`dataclasses` — Data Classes

Código fuente: Lib/dataclasses.py [https://github.com/python/cpython/tree/3.14/Lib/dataclasses.py]

This module provides a decorator and functions for automatically adding generated special methods such as **`__init__()`** and **`__repr__()`** to user-defined classes. It was originally described in **PEP 557** [https://peps.python.org/pep-0557/].

Las variables miembro a utilizar en estos métodos generados son definidas teniendo en cuenta anotaciones de tipo **PEP 526** [https://peps.python.org/pep-0526/]. Por ejemplo, en este código:

```
from dataclasses import dataclass

@dataclass
class InventoryItem:
    """Class for keeping track of an item in inventory."
    name: str
    unit_price: float
    quantity_on_hand: int = 0

    def total_cost(self) -> float:
        return self.unit_price * self.quantity_on_hand
```

will add, among other things, a **`__init__()`** that looks like:

```
def __init__(self, name: str, unit_price: float, quantit
    self.name = name
    self.unit_price = unit_price
    self.quantity_on_hand = quantity_on_hand
```

Note that this method is automatically added to the class: it is not directly specified in the **`InventoryItem`** definition shown above.

Added in version 3.7.

Contenidos del módulo

@dataclasses.dataclass(**, init=True, repr=True, eq=True, order=False, unsafe_hash=False, frozen=False, match_args=True, kw_only=False, slots=False, weakref_slot=False*)

This function is a decorator that is used to add generated special methods to classes, as described below.

The `@dataclass` decorator examines the class to find `field`s. A `field` is defined as a class variable that has a type annotation. With two exceptions described below, nothing in `@dataclass` examines the type specified in the variable annotation.

El orden de los campos en los métodos generados es el mismo en el que se encuentran en la definición de la clase.

The `@dataclass` decorator will add various «dunder» methods to the class, described below. If any of the added methods already exist in the class, the behavior depends on the parameter, as documented below. The decorator returns the same class that it is called on; no new class is created.

If `@dataclass` is used just as a simple decorator with no parameters, it acts as if it has the default values documented in this signature. That is, these three uses of `@dataclass` are equivalent:

```
@dataclass
class C:
    ...

@dataclass()
class C:
    ...

@dataclass(init=True, repr=True, eq=True, order=Fal
           match_args=True, kw_only=False, slots=Fa
class C:
    ...
```

The parameters to `@dataclass` are:

- *init*: If true (the default), a **`__init__()`** method will be generated.

  If the class already defines **`__init__()`**, this parameter is ignored.

- *repr*: If true (the default), a **`__repr__()`** method will be generated. The generated repr string will have the class name and the name and repr of each field, in the order they are defined in the class. Fields that are marked as being excluded from the repr are not included. For example: `InventoryItem(name='widget', unit_price=3.0, quantity_on_hand=10)`.

  If the class already defines **`__repr__()`**, this parameter is ignored.

- *eq*: If true (the default), an **`__eq__()`** method will be generated. This method compares the class as if it were a tuple of its fields, in order. Both instances in the comparison must be of the identical type.

  If the class already defines **`__eq__()`**, this parameter is ignored.

- *order*: If true (the default is `False`), **`__lt__()`**, **`__le__()`**, **`__gt__()`**, and **`__ge__()`** methods will be generated. These compare the class as if it were a tuple of its fields, in order. Both instances in the comparison must be of the identical type. If *order* is true and *eq* is false, a **`ValueError`** is raised.

  If the class already defines any of **`__lt__()`**, **`__le__()`**, **`__gt__()`**, or **`__ge__()`**, then

`TypeError` is raised.

- *unsafe_hash*: If true, force `dataclasses` to create a **`__hash__()`** method, even though it may not be safe to do so. Otherwise, generate a **`__hash__()`** method according to how *eq* and *frozen* are set. The default value is `False`.

  `__hash__()` is used by built-in **`hash()`**, and when objects are added to hashed collections such as dictionaries and sets. Having a **`__hash__()`** implies that instances of the class are immutable. Mutability is a complicated property that depends on the programmer's intent, the existence and behavior of **`__eq__()`**, and the values of the *eq* and *frozen* flags in the `@dataclass` decorator.

  By default, `@dataclass` will not implicitly add a **`__hash__()`** method unless it is safe to do so. Neither will it add or change an existing explicitly defined **`__hash__()`** method. Setting the class attribute `__hash__ = None` has a specific meaning to Python, as described in the **`__hash__()`** documentation.

  If **`__hash__()`** is not explicitly defined, or if it is set to `None`, then `@dataclass` *may* add an implicit **`__hash__()`** method. Although not recommended, you can force `@dataclass` to create a **`__hash__()`** method with `unsafe_hash=True`. This might be the case if your class is logically immutable but can still be mutated. This is a specialized use case and should be considered carefully.

  Here are the rules governing implicit creation of a **`__hash__()`** method. Note that you cannot both have an explicit **`__hash__()`** method in your dataclass and set `unsafe_hash=True`; this will result in a **`TypeError`**.

  If *eq* and *frozen* are both true, by default `@dataclass` will generate a **`__hash__()`** method for you. If *eq* is true and *frozen* is false, `__hash__()` will be set to `None`, marking it unhashable (which it is, since it is mutable). If *eq* is false, `__hash__()` will be left untouched meaning the `__hash__()` method of the superclass will be used (if the superclass is `object`, this means it will fall back to id-based hashing).

- *frozen*: If true (the default is `False`), assigning to fields will generate an exception. This emulates read-only frozen instances. See the discussion below.

  If `__setattr__()` or `__delattr__()` is defined in the class and *frozen* is true, then `TypeError` is raised.

- *match_args*: If true (the default is `True`), the `__match_args__` tuple will be created from the list of non keyword-only parameters to the generated `__init__()` method (even if `__init__()` is not generated, see above). If false, or if `__match_args__` is already defined in the class, then `__match_args__` will not be generated.

  Added in version 3.10.

- *kw_only*: If true (the default value is `False`), then all fields will be marked as keyword-only. If a field is marked as keyword-only, then the only effect is that the `__init__()` parameter generated from a keyword-only field must be specified with a keyword when `__init__()` is called. See the parameter glossary entry for details. Also see the `KW_ONLY` section.

  Keyword-only fields are not included in `__match_args__`.

  Added in version 3.10.

- *slots*: If true (the default is `False`), `__slots__` attribute will be generated and new class will be returned instead of the original one. If `__slots__` is already defined in the class, then `TypeError` is raised.

Advertencia

Passing parameters to a base class **`__init_subclass__()`** when using `slots=True` will result in a **`TypeError`**. Either use `__init_subclass__` with no parameters or use default values as a workaround. See gh-91126 [https://github.com/python/cpython/issues/91126] for full details.

Added in version 3.10.

Distinto en la versión 3.11: If a field name is already included in the **`__slots__`** of a base class, it will not be included in the generated **`__slots__`** to prevent overriding them. Therefore, do not use **`__slots__`** to retrieve the field names of a dataclass. Use **`fields()`** instead. To be able to determine inherited slots, base class **`__slots__`** may be any iterable, but *not* an iterator.

- *weakref_slot*: If true (the default is `False`), add a slot named «_weakref_», which is required to make an instance **`weakref-able`**. It is an error to specify `weakref_slot=True` without also specifying `slots=True`.

  Added in version 3.11.

Los `fields` pueden especificar un valor por defecto opcionalmente, simplemente usando la sintaxis normal de Python:

```
@dataclass
class C:
    a: int       # 'a' has no default value
    b: int = 0   # assign a default value for 'b'
```

In this example, both `a` and `b` will be included in the added `__init__()` method, which will be defined as:

```
def __init__(self, a: int, b: int = 0):
```

Si, en la definición de una clase, a un campo con valor por defecto le sigue un campo sin valor por defecto será lanzada una excepción `TypeError`. Esto se aplica también a la implementación de una clase única o como resultado de herencia de clases.

dataclasses.field(*, *default=MISSING*, *default_factory=MISSING*, *init=True*, *repr=True*, *hash=None*, *compare=True*, *metadata=None*, *kw_only=MISSING*, *doc=None*)

For common and simple use cases, no other functionality is required. There are, however, some dataclass features that require additional per-field information. To satisfy this need for additional information, you can replace the default field value with a call to the provided `field()` function. For example:

```
@dataclass
class C:
    mylist: list[int] = field(default_factory=list)

c = C()
c.mylist += [1, 2, 3]
```

Como se muestra arriba, el valor `MISSING` es un objeto centinela que se usa para detectar si el usuario proporciona algunos parámetros. Este centinela se utiliza porque `None` es un valor válido para algunos parámetros con un significado distinto. Ningún código debe utilizar directamente el valor `MISSING`.

The parameters to `field()` are:

- *default*: If provided, this will be the default value for this field. This is needed because the `field()` call itself replaces the normal position of the default value.

- *default_factory*: If provided, it must be a zero-argument callable that will be called when a default value is needed for this field. Among other purposes, this can be used to specify fields with mutable default values, as discussed below. It is an error to specify both *default* and *default_factory*.

- *init*: If true (the default), this field is included as a parameter to the generated `__init__()` method.

- *repr*: If true (the default), this field is included in the string returned by the generated `__repr__()` method.

- *hash*: This can be a bool or `None`. If true, this field is included in the generated `__hash__()` method. If false, this field is excluded from the generated `__hash__()`. If `None` (the default), use the value of *compare*: this would normally be the expected behavior, since a field should be included in the hash if it's used for comparisons. Setting this value to anything other than `None` is discouraged.

  Una posible razón para definir `hash=False` y `compare=True` podría ser el caso en el que computar el valor *hash* para dicho campo es costoso pero el campo es necesario para los métodos de comparación, siempre que existan otros campos que contribuyen al valor hash del tipo. Incluso si un campo se excluye del hash, se seguirá utilizando a la hora de comparar.

- *compare*: If true (the default), this field is included in the generated equality and comparison methods (`__eq__()`, `__gt__()`, et al.).

- *metadata*: This can be a mapping or `None`. `None` is treated as an empty dict. This value is wrapped in `MappingProxyType()` to make it read-only, and exposed on the `Field` object. It is not used at all by Data Classes, and is provided as a third-party extension mechanism. Multiple third-parties can each have their own key, to use as a namespace in the metadata.

- *kw_only*: If true, this field will be marked as keyword-only. This is used when the generated `__init__()` method's parameters are computed.

  Keyword-only fields are also not included in **`__match_args__`**.

  Added in version 3.10.

- *doc*: optional docstring for this field.

  Added in version 3.14.

If the default value of a field is specified by a call to **`field()`**, then the class attribute for this field will be replaced by the specified *default* value. If *default* is not provided, then the class attribute will be deleted. The intent is that after the `@dataclass` decorator runs, the class attributes will all contain the default values for the fields, just as if the default value itself were specified. For example, after:

```
@dataclass
class C:
    x: int
    y: int = field(repr=False)
    z: int = field(repr=False, default=10)
    t: int = 20
```

The class attribute **`C.z`** will be `10`, the class attribute **`C.t`** will be `20`, and the class attributes **`C.x`** and **`C.y`** will not be set.

class dataclasses.Field

`Field` objects describe each defined field. These objects are created internally, and are returned by the `fields()` module-level method (see below). Users should never instantiate a **`Field`** object directly. Its documented attributes are:

- **`name`**: The name of the field.

- **`type`**: The type of the field.
- **`default`**, **`default_factory`**, **`init`**, **`repr`**, **`hash`**, **`compare`**, **`metadata`**, and **`kw_only`** have the identical meaning and values as they do in the **`field()`** function.

Pueden existir otros atributos, pero son privados y no deberían ser considerados ni depender de ellos.

class dataclasses.InitVar

`InitVar[T]` type annotations describe variables that are init-only. Fields annotated with **`InitVar`** are considered pseudo-fields, and thus are neither returned by the **`fields()`** function nor used in any way except adding them as parameters to **`__init__()`** and an optional **`__post_init__()`**.

dataclasses.fields(*class_or_instance*)

Retorna una tupla de objetos **`Field`** que definen los campos para esta clase de datos. Acepta tanto una clase de datos como una instancia de esta. Lanza una excepción **`TypeError`** si se le pasa cualquier otro objeto. No retorna pseudocampos, que son `ClassVar` o `InitVar`.

dataclasses.asdict(*obj*, ***, *dict_factory=dict*)

Converts the dataclass *obj* to a dict (by using the factory function *dict_factory*). Each dataclass is converted to a dict of its fields, as `name: value` pairs. dataclasses, dicts, lists, and tuples are recursed into. Other objects are copied with **`copy.deepcopy()`**.

Example of using **`asdict()`** on nested dataclasses:

```
@dataclass
class Point:
     x: int
     y: int

@dataclass
```

```
class C:
     mylist: list[Point]

p = Point(10, 20)
assert asdict(p) == {'x': 10, 'y': 20}

c = C([Point(0, 0), Point(10, 4)])
assert asdict(c) == {'mylist': [{'x': 0, 'y': 0}, {
```

To create a shallow copy, the following workaround may be used:

```
{field.name: getattr(obj, field.name) for field in
```

`asdict()` raises **`TypeError`** if *obj* is not a dataclass instance.

dataclasses.astuple(*obj*, ***, *tuple_factory=tuple*)

Converts the dataclass *obj* to a tuple (by using the factory function *tuple_factory*). Each dataclass is converted to a tuple of its field values. dataclasses, dicts, lists, and tuples are recursed into. Other objects are copied with **`copy.deepcopy()`**.

Continuando con el ejemplo anterior:

```
assert astuple(p) == (10, 20)
assert astuple(c) == ([(0, 0), (10, 4)],)
```

To create a shallow copy, the following workaround may be used:

```
tuple(getattr(obj, field.name) for field in datacla
```

`astuple()` raises **`TypeError`** if *obj* is not a dataclass instance.

dataclasses.make_dataclass(*cls_name*, *fields*, ***, *bases=()*, *namespace=None*, *init=True*, *repr=True*, *eq=True*, *order=False*, *unsafe_hash=False*, *frozen=False*, *match_args=True*, *kw_only=False*, *slots=False, weakref_slot=False, module=None, decorator=dataclass*)

Creates a new dataclass with name *cls_name*, fields as defined in *fields*, base classes as given in *bases*, and initialized with a namespace as given in *namespace*. *fields* is an iterable whose elements are each either `name`, `(name, type)`, or `(name, type, Field)`. If just `name` is supplied, `typing.Any` is used for `type`. The values of *init*, *repr*, *eq*, *order*, *unsafe_hash*, *frozen*, *match_args*, *kw_only*, *slots*, and *weakref_slot* have the same meaning as they do in `@dataclass`.

If *module* is defined, the `__module__` attribute of the dataclass is set to that value. By default, it is set to the module name of the caller.

The *decorator* parameter is a callable that will be used to create the dataclass. It should take the class object as a first argument and the same keyword arguments as `@dataclass`. By default, the `@dataclass` function is used.

This function is not strictly required, because any Python mechanism for creating a new class with `__annotations__` can then apply the `@dataclass` function to convert that class to a dataclass. This function is provided as a convenience. For example:

```
C = make_dataclass('C',
                   [('x', int),
                     'y',
                    ('z', int, field(default=5))],
                   namespace={'add_one': lambda sel
```

Es equivalente a:

```
@dataclass
class C:
    x: int
    y: 'typing.Any'
    z: int = 5

    def add_one(self):
        return self.x + 1
```

Added in version 3.14: Added the *decorator* parameter.

dataclasses.replace(*obj*, /, ***changes*)

Creates a new object of the same type as *obj*, replacing fields with values from *changes*. If *obj* is not a Data Class, raises `TypeError`. If keys in *changes* are not field names of the given dataclass, raises **`TypeError`**.

The newly returned object is created by calling the `__init__()` method of the dataclass. This ensures that `__post_init__()`, if present, is also called.

Init-only variables without default values, if any exist, must be specified on the call to **`replace()`** so that they can be passed to **`__init__()`** and `__post_init__()`.

It is an error for *changes* to contain any fields that are defined as having `init=False`. A `ValueError` will be raised in this case.

Be forewarned about how `init=False` fields work during a call to **`replace()`**. They are not copied from the source object, but rather are initialized in `__post_init__()`, if they're initialized at all. It is expected that `init=False` fields will be rarely and judiciously used. If they are used, it might be wise to have alternate class constructors, or perhaps a custom **`replace()`** (or similarly named) method which handles instance copying.

Dataclass instances are also supported by generic function `copy.replace()`.

dataclasses.is_dataclass(*obj*)

Return `True` if its parameter is a dataclass (including subclasses of a dataclass) or an instance of one, otherwise return `False`.

Si se necesita conocer si una clase es una instancia de

dataclass (y no una clase de datos en si misma), se debe agregar una verificación adicional para `not isinstance(obj, type)`:

```
def is_dataclass_instance(obj):
    return is_dataclass(obj) and not isinstance(obj
```

dataclasses.MISSING

Un valor centinela que significa que falta un default o default_factory.

dataclasses.KW_ONLY

A sentinel value used as a type annotation. Any fields after a pseudo-field with the type of **`KW_ONLY`** are marked as keyword-only fields. Note that a pseudo-field of type **`KW_ONLY`** is otherwise completely ignored. This includes the name of such a field. By convention, a name of `_` is used for a **`KW_ONLY`** field. Keyword-only fields signify **`__init__()`** parameters that must be specified as keywords when the class is instantiated.

En este ejemplo, los campos `y` y `z` se marcarán como campos de solo palabras clave:

```
@dataclass
class Point:
    x: float
    _: KW_ONLY
    y: float
    z: float

p = Point(0, y=1.5, z=2.0)
```

In a single dataclass, it is an error to specify more than one field whose type is **`KW_ONLY`**.

Added in version 3.10.

exception dataclasses.FrozenInstanceError

Raised when an implicitly defined **`__setattr__()`** or **`__delattr__()`** is called on a dataclass which was defined with `frozen=True`. It is a subclass of **`AttributeError`**.

Procesamiento posterior a la inicialización

dataclasses.__post_init__()

When defined on the class, it will be called by the generated **`__init__()`**, normally as **`self.__post_init__()`**. However, if any `InitVar` fields are defined, they will also be passed to **`__post_init__()`** in the order they were defined in the class. If no **`__init__()`** method is generated, then **`__post_init__()`** will not automatically be called.

Entre otros usos, esto permite inicializar valores de campo que dependen de uno o más campos. Por ejemplo:

```
@dataclass
class C:
    a: float
    b: float
    c: float = field(init=False)

    def __post_init__(self):
        self.c = self.a + self.b
```

The **`__init__()`** method generated by **`@dataclass`** does not call base class **`__init__()`** methods. If the base class has an **`__init__()`** method that has to be called, it is common to call this method in a **`__post_init__()`** method:

```
class Rectangle:
    def __init__(self, height, width):
        self.height = height
        self.width = width

@dataclass
```

```
class Square(Rectangle):
    side: float

    def __post_init__(self):
        super().__init__(self.side, self.side)
```

Note, however, that in general the dataclass-generated **`__init__()`** methods don't need to be called, since the derived dataclass will take care of initializing all fields of any base class that is a dataclass itself.

See the section below on init-only variables for ways to pass parameters to **`__post_init__()`**. Also see the warning about how **`replace()`** handles `init=False` fields.

Variables de clase

One of the few places where **`@dataclass`** actually inspects the type of a field is to determine if a field is a class variable as defined in **PEP 526** [https://peps.python.org/pep-0526/]. It does this by checking if the type of the field is **`typing.ClassVar`**. If a field is a `ClassVar`, it is excluded from consideration as a field and is ignored by the dataclass mechanisms. Such `ClassVar` pseudo-fields are not returned by the module-level **`fields()`** function.

Variable de solo inicialización

Another place where **`@dataclass`** inspects a type annotation is to determine if a field is an init-only variable. It does this by seeing if the type of a field is of type **`InitVar`**. If a field is an **`InitVar`**, it is considered a pseudo-field called an init-only field. As it is not a true field, it is not returned by the module-level **`fields()`** function. Init-only fields are added as parameters to the generated **`__init__()`** method, and are passed to the optional **`__post_init__()`** method. They are not otherwise used by dataclasses.

Por ejemplo, supongamos que se va a inicializar un campo desde una base de datos, de no proporcionarse un valor al crear la clase:

```
@dataclass
class C:
    i: int
    j: int | None = None
    database: InitVar[DatabaseType | None] = None

    def __post_init__(self, database):
        if self.j is None and database is not None:
            self.j = database.lookup('j')

c = C(10, database=my_database)
```

In this case, `fields()` will return `Field` objects for **`i`** and **`j`**, but not for **`database`**.

Instancias congeladas

It is not possible to create truly immutable Python objects. However, by passing `frozen=True` to the `@dataclass` decorator you can emulate immutability. In that case, dataclasses will add `__setattr__()` and `__delattr__()` methods to the class. These methods will raise a `FrozenInstanceError` when invoked.

There is a tiny performance penalty when using `frozen=True`: `__init__()` cannot use simple assignment to initialize fields, and must use **`object.__setattr__()`**.

Herencia

When the dataclass is being created by the `@dataclass` decorator, it looks through all of the class's base classes in reverse MRO (that is, starting at `object`) and, for each dataclass that it finds, adds the fields from that base class to an ordered mapping of fields. After all of the base class fields are added, it adds its own fields to the ordered mapping. All of the generated methods will use this combined, calculated ordered mapping of fields. Because the fields are in insertion order, derived classes override base classes. An example:

```
@dataclass
class Base:
    x: Any = 15.0
    y: int = 0

@dataclass
class C(Base):
    z: int = 10
    x: int = 15
```

The final list of fields is, in order, `x`, `y`, `z`. The final type of `x` is `int`, as specified in class `C`.

The generated `__init__()` method for `C` will look like:

```
def __init__(self, x: int = 15, y: int = 0, z: int = 10)
```

Re-ordering of keyword-only parameters in `__init__()`

After the parameters needed for `__init__()` are computed, any keyword-only parameters are moved to come after all regular (non-keyword-only) parameters. This is a requirement of how keyword-only parameters are implemented in Python: they must come after non-keyword-only parameters.

In this example, `Base.y`, `Base.w`, and `D.t` are keyword-only fields, and `Base.x` and `D.z` are regular fields:

```
@dataclass
class Base:
    x: Any = 15.0
    _: KW_ONLY
    y: int = 0
    w: int = 1

@dataclass
class D(Base):
    z: int = 10
    t: int = field(kw_only=True, default=0)
```

The generated `__init__()` method for `D` will look like:

```
def __init__(self, x: Any = 15.0, z: int = 10, *, y: int
```

Tenga en cuenta que los parámetros se han reordenado a partir de cómo aparecen en la lista de campos: los parámetros derivados de los campos regulares son seguidos por los parámetros derivados de los campos de solo palabras clave.

The relative ordering of keyword-only parameters is maintained in the re-ordered `__init__()` parameter list.

Funciones fábrica por defecto

If a `field()` specifies a *default_factory*, it is called with zero arguments when a default value for the field is needed. For example, to create a new instance of a list, use:

```
mylist: list = field(default_factory=list)
```

If a field is excluded from `__init__()` (using `init=False`) and the field also specifies *default_factory*, then the default factory function will always be called from the generated `__init__()` function. This happens because there is no other way to give the field an initial value.

Valores por defecto mutables

Python almacena los valores miembros por defecto en atributos de clase. Considera este ejemplo, sin usar clases de datos:

```
class C:
    x = []
    def add(self, element):
        self.x.append(element)

o1 = C()
o2 = C()
```

```
o1.add(1)
o2.add(2)
assert o1.x == [1, 2]
assert o1.x is o2.x
```

Note that the two instances of class `C` share the same class variable `x`, as expected.

Usando clases de datos, *si* este código fuera válido:

```
@dataclass
class D:
    x: list = []      # This code raises ValueError
    def add(self, element):
        self.x.append(element)
```

generaría un código similar a:

```
class D:
    x = []
    def __init__(self, x=x):
        self.x = x
    def add(self, element):
        self.x.append(element)

assert D().x is D().x
```

This has the same issue as the original example using class `C`. That is, two instances of class `D` that do not specify a value for `x` when creating a class instance will share the same copy of `x`. Because dataclasses just use normal Python class creation they also share this behavior. There is no general way for Data Classes to detect this condition. Instead, the `@dataclass` decorator will raise a `ValueError` if it detects an unhashable default parameter. The assumption is that if a value is unhashable, it is mutable. This is a partial solution, but it does protect against many common errors.

Usar las funciones de fábrica por defecto es una forma de crear nuevas instancias de tipos mutables como valores por defecto para campos:

```
@dataclass
class D:
    x: list = field(default_factory=list)

assert D().x is not D().x
```

Distinto en la versión 3.11: Instead of looking for and disallowing objects of type **`list`**, **`dict`**, or **`set`**, unhashable objects are now not allowed as default values. Unhashability is used to approximate mutability.

Campos tipo descriptor

Los campos a los que se asigna objetos descriptor como valor por defecto tienen los siguientes comportamientos especiales:

- The value for the field passed to the dataclass's **`__init__()`** method is passed to the descriptor's **`__set__()`** method rather than overwriting the descriptor object.
- Similarly, when getting or setting the field, the descriptor's **`__get__()`** or **`__set__()`** method is called rather than returning or overwriting the descriptor object.
- To determine whether a field contains a default value, **`@dataclass`** will call the descriptor's **`__get__()`** method using its class access form: `descriptor.__get__(obj=None, type=cls)`. If the descriptor returns a value in this case, it will be used as the field's default. On the other hand, if the descriptor raises **`AttributeError`** in this situation, no default value will be provided for the field.

```
class IntConversionDescriptor:
    def __init__(self, *, default):
        self._default = default

    def __set_name__(self, owner, name):
        self._name = "_" + name

    def __get__(self, obj, type):
        if obj is None:
            return self._default

        return getattr(obj, self._name, self._default)

    def __set__(self, obj, value):
        setattr(obj, self._name, int(value))

@dataclass
class InventoryItem:
    quantity_on_hand: IntConversionDescriptor = IntConve

i = InventoryItem()
print(i.quantity_on_hand)   # 100
i.quantity_on_hand = 2.5    # calls __set__ with 2.5
print(i.quantity_on_hand)   # 2
```

Tenga en cuenta que si un campo está anotado con un tipo de descriptor, pero no se le asigna un objeto descriptor como valor por defecto, el campo actuará como un campo normal.

`contextlib` — Utilidades para declaraciones de contexto `with`

Código fuente: Lib/contextlib.py [https://github.com/python/cpython/tree/3.14/Lib/contextlib.py]

Este módulo proporciona utilidades para tareas comunes que involucran la declaración **`with`**. Para obtener más información, consulte también Tipos gestores de contexto y Gestores de Contexto en la Declaración with.

Utilidades

Funciones y clases proporcionadas:

class contextlib.AbstractContextManager

An abstract base class for classes that implement **`__enter__()`** and **`__exit__()`**. A default implementation for **`__enter__()`** is provided which returns `self` while **`__exit__()`** is an abstract method which by default returns `None`. See also the definition of Tipos gestores de contexto.

Added in version 3.6.

class contextlib.AbstractAsyncContextManager

An abstract base class for classes that implement **`__aenter__()`** and **`__aexit__()`**. A default implementation for **`__aenter__()`** is provided which returns `self` while **`__aexit__()`** is an abstract method which by default returns `None`. See also the definition of Gestores de contexto asíncronos.

Added in version 3.7.

@contextlib.contextmanager

This function is a decorator that can be used to define a factory function for `with` statement context managers, without needing to create a class or separate `__enter__()` and `__exit__()` methods.

While many objects natively support use in with statements, sometimes a resource needs to be managed that isn't a context manager in its own right, and doesn't implement a `close()` method for use with `contextlib.closing`.

Un ejemplo abstracto sería el siguiente para garantizar la gestión correcta de los recursos:

```
from contextlib import contextmanager

@contextmanager
def managed_resource(*args, **kwds):
    # Code to acquire resource, e.g.:
    resource = acquire_resource(*args, **kwds)
    try:
        yield resource
    finally:
        # Code to release resource, e.g.:
        release_resource(resource)
```

La función se puede usar así:

```
>>> with managed_resource(timeout=3600) as resource
...     # Resource is released at the end of this b
...     # even if code in the block raises an excep
```

La función que se está decorando debe retornar un iterador generador cuando se llama. Este iterador debe producir exactamente un valor, que estará vinculado a los objetivos en la `with` declaración de la cláusula `as`, si existe.

En el punto donde el generador cede, se ejecuta el bloque anidado en la palabra clave `with`. El generador se reanuda luego de salir del bloque. Si se produce una excepción no controlada en el bloque, se vuelve a plantear dentro del generador en el punto donde se produjo el rendimiento. Por lo tanto, puede usar una declaración `try`…`except`…`finally` para atrapar el error (si lo hay), o asegurarse de que se realice una limpieza. Si una excepción queda atrapada simplemente para registrarla o realizar alguna acción (en lugar de suprimirla por completo), el generador debe volver a generar esa excepción. De lo contrario, el administrador de contexto del generador indicará a la palabra clave **`with`** que se ha manejado la excepción, y la ejecución se reanudará con la declaración inmediatamente siguiente a la palabra clave **`with`**.

`contextmanager()` usa `ContextDecorator` para que los gestores de contexto que crea se puedan usar como decoradores, así como en declaraciones `with`. Cuando se usa como decorador, se crea implícitamente una nueva instancia de generador en cada llamada de función (esto permite que los gestores de contexto «de-un-tiro» creados por **`contextmanager()`** cumplan el requisito de que los gestores de contexto admitan múltiples invocaciones para ser utilizado como decoradores).

Distinto en la versión 3.2: Uso de `ContextDecorator`.

@contextlib.asynccontextmanager

Similar a `contextmanager()`, pero crea un administrador de contexto asíncrono.

This function is a decorator that can be used to define a factory function for `async with` statement asynchronous context managers, without needing to create a class or separate `__aenter__()` and `__aexit__()` methods. It must be applied to an asynchronous generator function.

Un ejemplo simple:

```
from contextlib import asynccontextmanager
```

```
@asynccontextmanager
async def get_connection():
    conn = await acquire_db_connection()
    try:
        yield conn
    finally:
        await release_db_connection(conn)

async def get_all_users():
    async with get_connection() as conn:
        return conn.query('SELECT ...')
```

Added in version 3.7.

Los administradores de contexto definidos con **`asynccontextmanager()`** se pueden utilizar como decoradores o con declaraciones **`async with`**:

```
import time
from contextlib import asynccontextmanager

@asynccontextmanager
async def timeit():
    now = time.monotonic()
    try:
        yield
    finally:
        print(f'it took {time.monotonic() - now}s t

@timeit()
async def main():
    # ... async code ...
```

Cuando se utiliza como decorador, se crea implícitamente una nueva instancia de generador en cada llamada de función. Esto permite que los administradores de contexto de otro modo «únicos» creados por **`asynccontextmanager()`** cumplan con el requisito de que los administradores de contexto admitan múltiples invocaciones para ser utilizados como decoradores.

Distinto en la versión 3.10: Los administradores de contexto asíncronos creados con `asynccontextmanager()` se pueden utilizar como decoradores.

contextlib.closing(*thing*)

retorna un gestor de contexto que cierra *thing* al completar el bloque. Esto es básicamente equivalente a:

```
from contextlib import contextmanager

@contextmanager
def closing(thing):
    try:
        yield thing
    finally:
        thing.close()
```

Y te permite escribir código como este:

```
from contextlib import closing
from urllib.request import urlopen

with closing(urlopen('https://www.python.org')) as
    for line in page:
        print(line)
```

sin necesidad de cerrar explícitamente la `page`. Incluso si se produce un error, se llamará a `page.close()` cuando salga el bloque `with`.

Nota

Most types managing resources support the context manager protocol, which closes *thing* on leaving the `with` statement. As such, `closing()` is most useful for third party types that don't support context managers. This example is purely for illustration purposes, as `urlopen()` would normally be used in a context manager.

contextlib.aclosing(*thing*)

Retorna un administrador de contexto asíncrono que llama al método `aclose()` de *thing* una vez completado el bloque. Esto es básicamente equivalente a:

```
from contextlib import asynccontextmanager

@asynccontextmanager
async def aclosing(thing):
    try:
        yield thing
    finally:
        await thing.aclose()
```

Significativamente, `aclosing()` admite la limpieza determinista de generadores asincrónicos cuando salen temprano por **`break`** o una excepción. Por ejemplo:

```
from contextlib import aclosing

async with aclosing(my_generator()) as values:
    async for value in values:
        if value == 42:
            break
```

Este patrón garantiza que el código de salida asíncrono del generador se ejecute en el mismo contexto que sus iteraciones (de modo que las excepciones y las variables de contexto funcionen como se esperaba, y el código de salida no se ejecute después de la vida útil de alguna tarea de la que depende).

Added in version 3.10.

contextlib.nullcontext(*enter_result=None*)

Return a context manager that returns *enter_result* from **`__enter__()`**, but otherwise does nothing. It is intended to be used as a stand-in for an optional context manager, for

example:

```
def myfunction(arg, ignore_exceptions=False):
    if ignore_exceptions:
        # Use suppress to ignore all exceptions.
        cm = contextlib.suppress(Exception)
    else:
        # Do not ignore any exceptions, cm has no e
        cm = contextlib.nullcontext()
    with cm:
        # Do something
```

Un ejemplo usando *enter_result*:

```
def process_file(file_or_path):
    if isinstance(file_or_path, str):
        # If string, open file
        cm = open(file_or_path)
    else:
        # Caller is responsible for closing file
        cm = nullcontext(file_or_path)

    with cm as file:
        # Perform processing on the file
```

También se puede utilizar como sustituto de asynchronous context managers:

```
async def send_http(session=None):
    if not session:
        # If no http session, create it with aiohtt
        cm = aiohttp.ClientSession()
    else:
        # Caller is responsible for closing the ses
        cm = nullcontext(session)

    async with cm as session:
        # Send http requests with session
```

Added in version 3.7.

Distinto en la versión 3.10: Se agregó compatibilidad con asynchronous context manager.

contextlib.suppress(**exceptions*)

Retorna un administrador de contexto que suprime cualquiera de las excepciones especificadas si ocurren en el cuerpo de una instrucción **`with`** y luego reanuda la ejecución con la primera instrucción que sigue al final de la instrucción **`with`**.

Al igual que con cualquier otro mecanismo que suprima completamente las excepciones, este administrador de contexto debe usarse solo para cubrir errores muy específicos en los que se sabe que continuar silenciosamente con la ejecución del programa es lo correcto.

Por ejemplo:

```
from contextlib import suppress

with suppress(FileNotFoundError):
    os.remove('somefile.tmp')

with suppress(FileNotFoundError):
    os.remove('someotherfile.tmp')
```

Este código es equivalente a:

```
try:
    os.remove('somefile.tmp')
except FileNotFoundError:
    pass

try:
    os.remove('someotherfile.tmp')
except FileNotFoundError:
    pass
```

Este gestor de contexto es reentrant.

If the code within the **`with`** block raises a

`BaseExceptionGroup`, suppressed exceptions are removed from the group. Any exceptions of the group which are not suppressed are re-raised in a new group which is created using the original group's **`derive()`** method.

Added in version 3.4.

Distinto en la versión 3.12: `suppress` now supports suppressing exceptions raised as part of a **`BaseExceptionGroup`**.

contextlib.redirect_stdout(*new_target*)

Administrador de contexto para redirigir temporalmente **`sys.stdout`** a otro archivo u objeto similar a un archivo.

Esta herramienta agrega flexibilidad a las funciones o clases existentes cuya salida está programada para stdout.

For example, the output of **`help()`** normally is sent to *sys.stdout*. You can capture that output in a string by redirecting the output to an **`io.StringIO`** object. The replacement stream is returned from the **`__enter__()`** method and so is available as the target of the **`with`** statement:

```
with redirect_stdout(io.StringIO()) as f:
    help(pow)
s = f.getvalue()
```

Para enviar la salida de **`help()`** a un archivo en el disco, redirija la salida a un archivo normal:

```
with open('help.txt', 'w') as f:
    with redirect_stdout(f):
        help(pow)
```

Para enviar la salida de **`help()`** a *sys.stderr*:

```
with redirect_stdout(sys.stderr):
    help(pow)
```

Tenga en cuenta que el efecto secundario global en `sys.stdout` significa que este administrador de contexto no es adecuado para su uso en el código de la biblioteca y en la mayoría de las aplicaciones con subprocesos. Tampoco tiene efecto en la salida de subprocesos. Sin embargo, sigue siendo un enfoque útil para muchos scripts de utilidad.

Este gestor de contexto es reentrant.

Added in version 3.4.

contextlib.redirect_stderr(*new_target*)

Similar a `redirect_stdout()` pero redirigiendo `sys.stderr` a otro archivo u objeto similar a un archivo.

Este gestor de contexto es reentrant.

Added in version 3.5.

contextlib.chdir(*path*)

Administrador de contexto no seguro en paralelo para cambiar el directorio de trabajo actual. Como esto cambia un estado global, el directorio de trabajo, no es adecuado para su uso en la mayoría de los contextos asincrónicos o de subprocesos. Tampoco es adecuado para la mayoría de las ejecuciones de código no lineal, como los generadores, donde la ejecución del programa se abandona temporalmente; a menos que se desee explícitamente, no debe ceder el paso cuando este administrador de contexto está activo.

Este es un contenedor simple alrededor de `chdir()`, cambia el directorio de trabajo actual al ingresar y restaura el anterior al salir.

Este gestor de contexto es reentrant.

Added in version 3.11.

class contextlib.ContextDecorator

Una clase base que permite que un administrador de contexto también se use como decorador.

Context managers inheriting from `ContextDecorator` have to implement **`__enter__()`** and **`__exit__()`** as normal. `__exit__` retains its optional exception handling even when used as a decorator.

`ContextDecorator` es utilizado por **`contextmanager()`**, por lo que obtiene esta funcionalidad automáticamente.

Ejemplo de `ContextDecorator`:

```
from contextlib import ContextDecorator

class mycontext(ContextDecorator):
    def __enter__(self):
        print('Starting')
        return self

    def __exit__(self, *exc):
        print('Finishing')
        return False
```

La clase se puede usar así:

```
>>> @mycontext()
... def function():
...     print('The bit in the middle')
...
>>> function()
Starting
The bit in the middle
Finishing

>>> with mycontext():
...     print('The bit in the middle')
...
Starting
The bit in the middle
Finishing
```

Este cambio es solo azúcar sintáctico para cualquier construcción de la siguiente forma:

```
def f():
    with cm():
        # Do stuff
```

`ContextDecorator` le permite escribir en su lugar:

```
@cm()
def f():
    # Do stuff
```

Deja en claro que el `cm` se aplica a toda la función, en lugar de solo una parte de ella (y guardar un nivel de sangría también es bueno).

Los gestores de contexto existentes que ya tienen una clase base pueden ampliarse utilizando `ContextDecorator` como una clase mezcla (*mixin*):

```
from contextlib import ContextDecorator

class mycontext(ContextBaseClass, ContextDecorator)
    def __enter__(self):
        return self

    def __exit__(self, *exc):
        return False
```

Nota

Como la función decorada debe poder llamarse varias veces, el gestor de contexto subyacente debe admitir el uso en múltiples declaraciones **`with`**. Si este no es el caso, se debe utilizar la construcción original con la declaración explícita **`with`** dentro de la función.

Added in version 3.2.

class contextlib.AsyncContextDecorator

Similar a `ContextDecorator` pero solo para funciones asincrónicas.

Ejemplo de `AsyncContextDecorator`:

```
from asyncio import run
from contextlib import AsyncContextDecorator

class mycontext(AsyncContextDecorator):
    async def __aenter__(self):
        print('Starting')
        return self

    async def __aexit__(self, *exc):
        print('Finishing')
        return False
```

La clase se puede usar así:

```
>>> @mycontext()
... async def function():
...     print('The bit in the middle')
...
>>> run(function())
Starting
The bit in the middle
Finishing

>>> async def function():
...    async with mycontext():
...         print('The bit in the middle')
...
>>> run(function())
Starting
The bit in the middle
Finishing
```

Added in version 3.10.

class contextlib.ExitStack

Un gestor de contexto que está diseñado para facilitar la combinación programática de otros gestores de contexto y funciones de limpieza, especialmente aquellas que son opcionales o que de otro modo son impulsadas por los datos de entrada.

Por ejemplo, un conjunto de archivos puede manejarse fácilmente en una sola declaración de la siguiente manera:

```
with ExitStack() as stack:
    files = [stack.enter_context(open(fname)) for f
    # All opened files will automatically be closed
    # the with statement, even if attempts to open
    # in the list raise an exception
```

The `__enter__()` method returns the **`ExitStack`** instance, and performs no additional operations.

Cada instancia mantiene una pila de retrollamadas registradas que se llaman en orden inverso cuando la instancia se cierra (ya sea explícita o implícitamente al final de una instrucción **`with`**). Tenga en cuenta que las retrollamadas *no* se invocan implícitamente cuando la instancia de la pila de contexto se recolecta basura.

Este modelo de pila se utiliza para que los administradores de contexto que adquieren sus recursos en su método `__init__` (como los objetos de archivo) se puedan manejar correctamente.

Dado que las retrollamadas registradas se invocan en el orden inverso del registro, esto termina comportándose como si se hubieran utilizado múltiples instrucciones anidadas **`with`** con el conjunto registrado de retrollamadas. Esto incluso se extiende al manejo de excepciones: si una retrollamada interna suprime o reemplaza una excepción, las retrollamadas externas se pasarán argumentos basados en ese estado actualizado.

Esta es una API de nivel relativamente bajo que se ocupa de los detalles de desenrollar correctamente la pila de retrollamadas de salida. Proporciona una base adecuada para administradores de contexto de nivel superior que manipulan la pila de salida en formas específicas de la aplicación.

Added in version 3.3.

enter_context(*cm*)

Enters a new context manager and adds its **`__exit__()`** method to the callback stack. The return value is the result of the context manager's own **`__enter__()`** method.

Estos administradores de contexto pueden suprimir excepciones tal como lo harían normalmente si se usaran directamente como parte de una declaración **`with`**.

Distinto en la versión 3.11: Lanza **`TypeError`** en lugar de **`AttributeError`** si *cm* no es un administrador de contexto.

push(*exit*)

Adds a context manager's **`__exit__()`** method to the callback stack.

As `__enter__` is *not* invoked, this method can be used to cover part of an **`__enter__()`** implementation with a context manager's own **`__exit__()`** method.

If passed an object that is not a context manager, this method assumes it is a callback with the same signature as a context manager's **`__exit__()`** method and adds it directly to the callback stack.

By returning true values, these callbacks can suppress exceptions the same way context manager **`__exit__()`** methods can.

El objeto pasado se retorna desde la función, lo que permite que este método se use como decorador de funciones.

callback(*callback*, */*, **args*, ***kwds*)

Acepta una función de retrollamada arbitraria y argumentos y la agrega a la pila de retrollamada.

A diferencia de los otros métodos, las retrollamadas agregadas de esta manera no pueden suprimir excepciones (ya que nunca se pasan los detalles de excepción).

La retrollamada pasada se retorna desde la función, lo que permite que este método se use como decorador de funciones.

pop_all()

Transfiere la pila de retrollamada a una instancia fresca **`ExitStack`** y la retorna. Esta operación no invoca retrollamadas; en cambio, ahora se invocarán cuando se cierre la nueva pila (ya sea explícita o implícitamente al final de una instrucción **`with`**).

Por ejemplo, un grupo de archivos se puede abrir como una operación de «todo o nada» de la siguiente manera:

```
with ExitStack() as stack:
    files = [stack.enter_context(open(fname)) f
    # Hold onto the close method, but don't cal
    close_files = stack.pop_all().close
    # If opening any file fails, all previously
    # closed automatically. If all files are op
    # they will remain open even after the with
    # close_files() can then be invoked explici
```

close()

Inmediatamente desenrolla la pila de retrollamada, invocando retrollamadas en el orden inverso de registro. Para los administradores de contexto y las retrollamadas de salida registradas, los argumentos pasados indicarán que no se produjo ninguna excepción.

class contextlib.AsyncExitStack

Un gestor de contexto asíncrono, similar a `ExitStack`, que admite la combinación de gestores de contexto síncrono y asíncrono, además de tener rutinas para la lógica de limpieza.

The `close()` method is not implemented; `aclose()` must be used instead.

async enter_async_context(*cm*)

Similar to `ExitStack.enter_context()` but expects an asynchronous context manager.

Distinto en la versión 3.11: Lanza `TypeError` en lugar de `AttributeError` si *cm* no es un administrador de contexto asíncrono.

push_async_exit(*exit*)

Similar to `ExitStack.push()` but expects either an asynchronous context manager or a coroutine function.

push_async_callback(*callback*, */*, **args*, ***kwds*)

Similar to `ExitStack.callback()` but expects a coroutine function.

async aclose()

Similar to `ExitStack.close()` but properly handles awaitables.

Continuando con el ejemplo para `asynccontextmanager()`:

```
async with AsyncExitStack() as stack:
    connections = [await stack.enter_async_context(
        for i in range(5)]
        # All opened connections will automatically be
        # the async with statement, even if attempts to
        # later in the list raise an exception.
```

Added in version 3.7.

Ejemplos y recetas

This section describes some examples and recipes for making effective use of the tools provided by **`contextlib`**.

Apoyando un número variable de gestores de contexto

El caso de uso principal para **`ExitStack`** es el que se proporciona en la documentación de la clase: admite un número variable de gestores de contexto y otras operaciones de limpieza en una sola **`with`**. La variabilidad puede provenir de la cantidad de gestores de contexto que necesitan ser impulsados por la entrada del usuario (como abrir una colección de archivos especificada por el usuario), o de que algunos de los gestores de contexto sean opcionales:

```
with ExitStack() as stack:
    for resource in resources:
        stack.enter_context(resource)
    if need_special_resource():
        special = acquire_special_resource()
        stack.callback(release_special_resource, special
    # Perform operations that use the acquired resources
```

Como se muestra, **`ExitStack`** también hace que sea bastante fácil de usar **`with`** para administrar recursos arbitrarios que no admiten de forma nativa el protocolo de gestión de contexto.

Capturando excepciones de los métodos `__enter__`

It is occasionally desirable to catch exceptions from an **`__enter__()`** method implementation, *without* inadvertently catching exceptions from the **`with`** statement body or the context manager’s `__exit__()` method. By using `ExitStack` the steps in the context management protocol can be separated slightly in order to allow this:

```
stack = ExitStack()
try:
    x = stack.enter_context(cm)
except Exception:
    # handle __enter__ exception
else:
    with stack:
        # Handle normal case
```

Es probable que la necesidad de hacer esto indique que la API subyacente debería proporcionar una interfaz de administración de recursos directa para usar con `try`/`except`/`finally`, pero no todas las API están bien diseñados en ese sentido. Cuando un administrador de contexto es la única API de administración de recursos proporcionada, entonces `ExitStack` puede facilitar el manejo de diversas situaciones que no se pueden manejar directamente en una declaración `with`.

Limpieza en una implementación `__enter__`

As noted in the documentation of `ExitStack.push()`, this method can be useful in cleaning up an already allocated resource if later steps in the `__enter__()` implementation fail.

Aquí hay un ejemplo de cómo hacer esto para un administrador de contexto que acepta funciones de adquisición y liberación de recursos, junto con una función de validación opcional, y las asigna al protocolo de administración de contexto:

```
from contextlib import contextmanager, AbstractContextMa

class ResourceManager(AbstractContextManager):

    def __init__(self, acquire_resource, release_resourc
        self.acquire_resource = acquire_resource
        self.release_resource = release_resource
        if check_resource_ok is None:
            def check_resource_ok(resource):
                return True
        self.check_resource_ok = check_resource_ok

    @contextmanager
    def _cleanup_on_error(self):
        with ExitStack() as stack:
            stack.push(self)
            yield
            # The validation check passed and didn't rai
            # Accordingly, we want to keep the resource,
            # back to our caller
            stack.pop_all()

    def __enter__(self):
        resource = self.acquire_resource()
        with self._cleanup_on_error():
            if not self.check_resource_ok(resource):
                msg = "Failed validation for {!r}"
                raise RuntimeError(msg.format(resource))
        return resource

    def __exit__(self, *exc_details):
        # We don't need to duplicate any of our resource
        self.release_resource()
```

Reemplazar cualquier uso de `try-finally` y marcar variables

Un patrón que a veces verá es una declaración de `try-finally` con una variable de indicador para indicar si el cuerpo de la cláusula `finally` debe ejecutarse o no. En su forma más simple (que ya no puede manejarse simplemente usando una cláusula `except` en su lugar), se parece a esto:

```
cleanup_needed = True
try:
    result = perform_operation()
    if result:
        cleanup_needed = False
finally:
    if cleanup_needed:
        cleanup_resources()
```

Al igual que con cualquier código basado en la declaración `try`, esto puede causar problemas de desarrollo y revisión, porque el código de configuración y el código de limpieza pueden terminar separados por secciones de código arbitrariamente largas.

`ExitStack` hace posible registrar una retrollamada para su ejecución al final de una instrucción `with`, y luego decide omitir la ejecución de esa retrollamada:

```
from contextlib import ExitStack

with ExitStack() as stack:
    stack.callback(cleanup_resources)
    result = perform_operation()
    if result:
        stack.pop_all()
```

This allows the intended cleanup behaviour to be made explicit up front, rather than requiring a separate flag variable.

Si una aplicación particular usa mucho este patrón, puede simplificarse aún más por medio de una pequeña clase auxiliar:

```
from contextlib import ExitStack

class Callback(ExitStack):
    def __init__(self, callback, /, *args, **kwds):
        super().__init__()
        self.callback(callback, *args, **kwds)

    def cancel(self):
        self.pop_all()

with Callback(cleanup_resources) as cb:
    result = perform_operation()
    if result:
        cb.cancel()
```

Si la limpieza del recurso no está bien agrupada en una función independiente, entonces todavía es posible usar la forma decoradora de **`ExitStack.callback()`** para declarar la limpieza del recurso por adelantado:

```
from contextlib import ExitStack

with ExitStack() as stack:
    @stack.callback
    def cleanup_resources():
        ...
    result = perform_operation()
    if result:
        stack.pop_all()
```

Debido a la forma en que funciona el protocolo decorador, una función de retrollamada declarada de esta manera no puede tomar ningún parámetro. En cambio, se debe acceder a los recursos que se liberarán como variables de cierre.

Usar un gestor de contexto como decorador de funciones

`ContextDecorator` hace posible usar un gestor de contexto tanto en una instrucción ordinaria `with` como también como decorador de funciones.

Por ejemplo, a veces es útil envolver funciones o grupos de declaraciones con un registrador que puede rastrear la hora de entrada y la hora de salida. En lugar de escribir tanto un decorador de funciones como un administrador de contexto para la tarea, heredar de **`ContextDecorator`** proporciona ambas capacidades en una sola definición:

```
from contextlib import ContextDecorator
import logging
```

```
logging.basicConfig(level=logging.INFO)

class track_entry_and_exit(ContextDecorator):
    def __init__(self, name):
        self.name = name

    def __enter__(self):
        logging.info('Entering: %s', self.name)

    def __exit__(self, exc_type, exc, exc_tb):
        logging.info('Exiting: %s', self.name)
```

Las instancias de esta clase se pueden usar como un gestor de contexto:

```
with track_entry_and_exit('widget loader'):
    print('Some time consuming activity goes here')
    load_widget()
```

Y también como decorador de funciones:

```
@track_entry_and_exit('widget loader')
def activity():
    print('Some time consuming activity goes here')
    load_widget()
```

Note that there is one additional limitation when using context managers as function decorators: there's no way to access the return value of `__enter__()`. If that value is needed, then it is still necessary to use an explicit `with` statement.

Ver también

PEP 343 [https://peps.python.org/pep-0343/] - La declaración «with»

La especificación, antecedentes y ejemplos de la declaración de Python `with`.

Gestores de contexto de uso único, reutilizables y reentrantes

La mayoría de los gestores de contexto están escritos de una manera que significa que solo se pueden usar de manera efectiva en una declaración **`with`** una vez. Estos administradores de contexto de un solo uso deben crearse de nuevo cada vez que se usan; si intenta usarlos por segunda vez, se activará una excepción o, de lo contrario, no funcionará correctamente.

Esta limitación común significa que generalmente es aconsejable crear gestores de contexto directamente en el encabezado de la palabra clave **`with`** donde se usan (como se muestra en todos los ejemplos de uso anteriores).

Los archivos son un ejemplo de gestores de contexto de un solo uso, ya que la primera **`with`** cerrará el archivo, evitando cualquier otra operación de E/S que use ese objeto de archivo.

Los gestores de contexto creados usando **`contextmanager()`** también son gestores de contexto de un solo uso, y se quejarán de la falla del generador subyacente si se intenta usarlos por segunda vez:

```
>>> from contextlib import contextmanager
>>> @contextmanager
... def singleuse():
...     print("Before")
...     yield
...     print("After")
...
>>> cm = singleuse()
>>> with cm:
...     pass
...
Before
After
>>> with cm:
...     pass
...
```

```
Traceback (most recent call last):
    ...
RuntimeError: generator didn't yield
```

Gestores contextuales reentrantes

Los gestores de contexto más sofisticados pueden ser «reentrantes». Estos administradores de contexto no solo se pueden usar en múltiples declaraciones `with`, sino que también se pueden usar *inside* a `with` que ya está usando el mismo gestor de contexto.

`threading.RLock` es un ejemplo de administrador de contexto reentrante, al igual que `suppress()`, `redirect_stdout()` y `chdir()`. Aquí hay un ejemplo muy simple de uso de reentrada:

```
>>> from contextlib import redirect_stdout
>>> from io import StringIO
>>> stream = StringIO()
>>> write_to_stream = redirect_stdout(stream)
>>> with write_to_stream:
...     print("This is written to the stream rather than
...     with write_to_stream:
...         print("This is also written to the stream")
...
>>> print("This is written directly to stdout")
This is written directly to stdout
>>> print(stream.getvalue())
This is written to the stream rather than stdout
This is also written to the stream
```

Es más probable que los ejemplos del mundo real de reentrada impliquen múltiples funciones que se llaman entre sí y, por lo tanto, sean mucho más complicadas que este ejemplo.

Tenga en cuenta también que ser reentrante *no* es lo mismo que ser seguro para subprocesos. `redirect_stdout()`, por ejemplo, definitivamente no es seguro para subprocesos, ya que realiza una modificación global al estado del sistema al vincular `sys.stdout` a una secuencia diferente.

Gestores contextuales reutilizables

Distintos de los administradores de contexto de uso único y reentrante son los administradores de contexto «reutilizables» (o, para ser completamente explícitos, los administradores de contexto «reutilizables, pero no reentrantes», ya que los administradores de contexto reentrantes también son reutilizables). Estos administradores de contexto admiten que se usen varias veces, pero fallarán (o de lo contrario no funcionarán correctamente) si la instancia específica del administrador de contexto ya se ha utilizado en una declaración que contiene.

`threading.Lock` es un ejemplo de un gestor de contexto reutilizable, pero no reentrante (para un bloqueo reentrante, es necesario usar **`threading.RLock`** en su lugar).

Otro ejemplo de un administrador de contexto reutilizable, pero no reentrante es **`ExitStack`**, ya que invoca *all* las retrollamadas registradas actualmente al dejar cualquier con declaración, independientemente de dónde se agregaron esas retrollamadas:

```
>>> from contextlib import ExitStack
>>> stack = ExitStack()
>>> with stack:
...     stack.callback(print, "Callback: from first cont
...     print("Leaving first context")
...
Leaving first context
Callback: from first context
>>> with stack:
...     stack.callback(print, "Callback: from second con
...     print("Leaving second context")
...
Leaving second context
Callback: from second context
>>> with stack:
...     stack.callback(print, "Callback: from outer cont
...     with stack:
...         stack.callback(print, "Callback: from inner 
...         print("Leaving inner context")
...     print("Leaving outer context")
...
Leaving inner context
Callback: from inner context
Callback: from outer context
Leaving outer context
```

Como muestra el resultado del ejemplo, la reutilización de un solo objeto de pila en múltiples con declaraciones funciona correctamente, pero intentar anidarlos hará que la pila se borre al final de la declaración más interna, lo que es poco probable que sea un comportamiento deseable.

El uso de instancias separadas **`ExitStack`** en lugar de reutilizar una sola instancia evita ese problema:

```
>>> from contextlib import ExitStack
>>> with ExitStack() as outer_stack:
...     outer_stack.callback(print, "Callback: from oute
...     with ExitStack() as inner_stack:
...         inner_stack.callback(print, "Callback: from 
...         print("Leaving inner context")
...     print("Leaving outer context")
...
Leaving inner context
Callback: from inner context
Leaving outer context
Callback: from outer context
```

[1]

`abc` — Abstract Base Classes

Código fuente: Lib/abc.py [https://github.com/python/cpython/tree/3.14/Lib/abc.py]

Este módulo proporciona la infraestructura para definir clases de base abstracta (CBAs) en Python, como se describe en **PEP 3119** [https://peps.python.org/pep-3119/]; consulte en el PEP el porqué fue agregado a Python. (Véase también **PEP 3141** [https://peps.python.org/pep-3141/] y el módulo **`numbers`** con respecto a una jerarquía de tipos para números basados en CBAs.)

The **`collections`** module has some concrete classes that derive from ABCs; these can, of course, be further derived. In addition, the **`collections.abc`** submodule has some ABCs that can be used to test whether a class or instance provides a particular interface, for example, if it is hashable or if it is a mapping.

Este módulo provee la metaclase **`ABCMeta`** para definir CBAs y una clase auxiliar **`ABC`** para definir CBAs alternativamente a través de herencia:

class abc.ABC

A helper class that has **`ABCMeta`** as its metaclass. With this class, an abstract base class can be created by simply deriving from **`ABC`** avoiding sometimes confusing metaclass usage, for example:

```
from abc import ABC

class MyABC(ABC):
    pass
```

Note that the type of **`ABC`** is still **`ABCMeta`**, therefore inheriting from **`ABC`** requires the usual precautions regarding metaclass usage, as multiple inheritance may lead to metaclass conflicts. One may also define an abstract base class by passing the metaclass keyword and using **`ABCMeta`** directly, for example:

```
from abc import ABCMeta

class MyABC(metaclass=ABCMeta):
    pass
```

Added in version 3.4.

class abc.ABCMeta

Metaclases para definir Clases de Base Abstracta (CBAs).

Utilice esta metaclase para crear una CBA. Una CBA puede ser heredada directamente y así, actuar como una clase mixta. También se puede registrar clases concretas no relacionadas (incluso clases integradas) y CBAs no relacionadas como «subclases virtuales» – estas y sus descendientes serán consideradas subclases del CBA registrado por la función integrada **`issubclass()`**, pero la CBA registrada no aparecerá en su *MRO* (Orden de Resolución de Métodos) ni las implementaciones de método definidas por la CBA registrada serán invocables (ni siquiera a través de **`super()`**). [1]

Classes created with a metaclass of **`ABCMeta`** have the following method:

register(*subclass*)

Registre la *subclase* como una «subclase virtual» de esta CBA. Por ejemplo:

```
from abc import ABC

class MyABC(ABC):
    pass

MyABC.register(tuple)
```

```
assert issubclass(tuple, MyABC)
assert isinstance((), MyABC)
```

Distinto en la versión 3.3: Retorna la subclase registrada, para permitir su uso como decorador de clase.

Distinto en la versión 3.4: To detect calls to **`register()`**, you can use the **`get_cache_token()`** function.

También se puede redefinir este método en una clase de base abstracta:

subclasshook(*subclass*)

(Debe ser definido como un método de clase.)

Check whether *subclass* is considered a subclass of this ABC. This means that you can customize the behavior of **`issubclass()`** further without the need to call **`register()`** on every class you want to consider a subclass of the ABC. (This class method is called from the **`__subclasscheck__()`** method of the ABC.)

This method should return `True`, `False` or **`NotImplemented`**. If it returns `True`, the *subclass* is considered a subclass of this ABC. If it returns `False`, the *subclass* is not considered a subclass of this ABC, even if it would normally be one. If it returns **`NotImplemented`**, the subclass check is continued with the usual mechanism.

Para una demostración de estos conceptos, vea este ejemplo de la definición CBA:

```
class Foo:
    def __getitem__(self, index):
        ...
    def __len__(self):
        ...
    def get_iterator(self):
        return iter(self)

class MyIterable(ABC):

    @abstractmethod
    def __iter__(self):
        while False:
            yield None

    def get_iterator(self):
        return self.__iter__()

    @classmethod
    def __subclasshook__(cls, C):
        if cls is MyIterable:
            if any("__iter__" in B.__dict__ for B i
                return True
        return NotImplemented

MyIterable.register(Foo)
```

The ABC `MyIterable` defines the standard iterable method, `__iter__()`, as an abstract method. The implementation given here can still be called from subclasses. The **`get_iterator()`** method is also part of the `MyIterable` abstract base class, but it does not have to be overridden in non-abstract derived classes.

The `__subclasshook__()` class method defined here says that any class that has an `__iter__()` method in its `__dict__` (or in that of one of its base classes, accessed via the `__mro__` list) is considered a `MyIterable` too.

Finally, the last line makes `Foo` a virtual subclass of `MyIterable`, even though it does not define an `__iter__()` method (it uses the old-style iterable protocol, defined in terms of `__len__()` and `__getitem__()`). Note that this will not make `get_iterator` available as a method of `Foo`, so it is provided separately.

The **abc** module also provides the following decorator:

@abc.abstractmethod

Un decorador que indica métodos abstractos.

Using this decorator requires that the class's metaclass is `ABCMeta` or is derived from it. A class that has a metaclass derived from **ABCMeta** cannot be instantiated unless all of its abstract methods and properties are overridden. The abstract methods can be called using any of the normal "super" call mechanisms. **abstractmethod()** may be used to declare abstract methods for properties and descriptors.

Dynamically adding abstract methods to a class, or attempting to modify the abstraction status of a method or class once it is created, are only supported using the `update_abstractmethods()` function. The **abstractmethod()** only affects subclasses derived using regular inheritance; «virtual subclasses» registered with the ABC's `register()` method are not affected.

When **abstractmethod()** is applied in combination with other method descriptors, it should be applied as the innermost decorator, as shown in the following usage examples:

```
class C(ABC):
    @abstractmethod
    def my_abstract_method(self, arg1):
        ...
    @classmethod
    @abstractmethod
    def my_abstract_classmethod(cls, arg2):
        ...
    @staticmethod
    @abstractmethod
    def my_abstract_staticmethod(arg3):
        ...

    @property
    @abstractmethod
    def my_abstract_property(self):
        ...
    @my_abstract_property.setter
    @abstractmethod
    def my_abstract_property(self, val):
        ...

    @abstractmethod
    def _get_x(self):
        ...
    @abstractmethod
    def _set_x(self, val):
        ...
    x = property(_get_x, _set_x)
```

In order to correctly interoperate with the abstract base class machinery, the descriptor must identify itself as abstract using **`__isabstractmethod__`**. In general, this attribute should be `True` if any of the methods used to compose the descriptor are abstract. For example, Python's built-in **`property`** does the equivalent of:

```
class Descriptor:
    ...
    @property
    def __isabstractmethod__(self):
        return any(getattr(f, '__isabstractmethod__
                   f in (self._fget, self._fset, se
```

Nota

A diferencia de los métodos abstractos de Java, estos métodos abstractos pueden tener una implementación. Esta implementación se puede llamar a través del mecanismo **`super()`** de la clase que lo invalida. Esto podría ser útil como un *end-point* para una super llamada en un *framework* que use herencia múltiple cooperativa.

The **`abc`** module also supports the following legacy decorators:

@abc.abstractclassmethod

Added in version 3.2.

Obsoleto desde la versión 3.3: Ahora es posible utilizar **`classmethod`** con **`abstractmethod()`**, lo cual hace que este decorador sea redundante.

Una subclase de la **`classmethod()`** incorporada, indicando un método de clase abstracto. De otra forma, es similar a **`abstractmethod()`**.

Este caso especial está obsoleto, ya que el decorador **`classmethod()`** ahora es identificado correctamente como abstracto cuando se aplica a un método abstracto:

```
class C(ABC):
    @classmethod
    @abstractmethod
    def my_abstract_classmethod(cls, arg):
        ...
```

@abc.abstractstaticmethod

Added in version 3.2.

Obsoleto desde la versión 3.3: Ahora es posible utilizar **`staticmethod`** con **`abstractmethod()`**, haciendo que este decorador sea redundante.

Una subclase de la **`staticmethod()`** incorporada, indicando un método estático abstracto. De otra forma, es similar a **`abstractmethod()`**.

Este caso especial está obsoleto, ya que el decorador **`staticmethod()`** ahora es identificado correctamente como abstracto cuando se aplica a un método abstracto:

```
class C(ABC):
    @staticmethod
    @abstractmethod
    def my_abstract_staticmethod(arg):
        ...
```

@abc.abstractproperty

Obsoleto desde la versión 3.3: Ahora es posible utilizar `property`, `property.getter()`, `property.setter()` y `property.deleter()` con `abstractmethod()`, lo cual hace que este decorador sea redundante.

Una subclase de la `property()` integrada, que indica una propiedad abstracta.

Este caso especial está obsoleto, ya que el decorador `property()` ahora es identificado correctamente como abstracto cuando es aplicado a un método abstracto:

```
class C(ABC):
    @property
    @abstractmethod
    def my_abstract_property(self):
        ...
```

En el ejemplo anterior se define una propiedad de solo lectura; también se puede definir una propiedad abstracta de lectura y escritura marcando adecuadamente uno o varios de los métodos subyacentes como abstractos:

```
class C(ABC):
    @property
    def x(self):
        ...

    @x.setter
    @abstractmethod
    def x(self, val):
        ...
```

Si solo algunos componentes son abstractos, solo estos componentes necesitan ser actualizados para crear una propiedad concreta en una subclase:

```
class D(C):
    @C.x.setter
    def x(self, val):
        ...
```

The `abc` module also provides the following functions:

abc.get_cache_token()

Retorna el token de caché de la clase base abstracta actual.

El token es un objeto opaco (que admite pruebas de igualdad) que identifica la versión actual de la caché de clases de base abstractas para subclases virtuales. El token cambia con cada llamada a `ABCMeta.register()` en cualquier CBA.

Added in version 3.4.

abc.update_abstractmethods(*cls*)

Una función para recalcular el estado de abstracción de una clase abstracta. Se debe llamar a esta función si los métodos abstractos de una clase se han implementado o cambiado después de su creación. Por lo general, esta función debe llamarse desde dentro de un decorador de clases.

Retorna *cls*, para permitir su uso como decorador de clases.

Si *cls* no es una instancia de `ABCMeta`, no hace nada.

Nota

Esta función asume que las superclases de *cls* ya están actualizadas. No actualiza subclases.

Added in version 3.10.

Notas al pie

Los desarrolladores de C++ pueden notar que el concepto de clase base virtual de Python no es el mismo que en C++.

`atexit` — Exit handlers

The **`atexit`** module defines functions to register and unregister cleanup functions. Functions thus registered are automatically executed upon normal interpreter termination. **`atexit`** runs these functions in the *reverse* order in which they were registered; if you register `A`, `B`, and `C`, at interpreter termination time they will be run in the order `C`, `B`, `A`.

Nota: Las funciones registradas a través de este módulo no son llamadas cuando el programa es eliminado por una señal no manejada por Python, cuando se detecta un error fatal interno en Python o cuando se llama a la función **`os._exit()`**.

Nota: El efecto de registrar o cancelar el registro de funciones dentro de una función de limpieza no esta definido.

Distinto en la versión 3.7: Cuando se usan con sub-intérpretes API C, las funciones registradas son locales para el intérprete en el que se registraron.

atexit.register(*func*, **args*, ***kwargs*)

Registra *func* como una función que se ejecutará cuando el intérprete se termine. Cualquier argumento opcional que deba pasarse a *func* debe pasarse como un argumento para la función **`register()`**. Es posible registrar la misma función y argumentos más de una vez.

En la finalización normal del programa (por ejemplo, si se llama a la función **`sys.exit()`** o finaliza la ejecución del módulo principal), todas las funciones registradas son llamadas en el orden último en entrar, primero en salir. Se supone que los módulos de menor nivel normalmente se importarán antes que los módulos de mayor nivel, y por lo tanto, se limpiarán al final.

Si se lanza una excepción durante la ejecución de los manejadores de salida, se muestra un rastreo (a menos que se haya lanzado `SystemExit`) y se guarda la información de la excepción. Después de que todos los manejadores de salida hayan tenido la oportunidad de ejecutarse, la última excepción lanzada se vuelve a lanzar.

Esta función retorna *func*, lo que hace posible usarlo como decorador.

Advertencia

Iniciar nuevos hilos o llamar a `os.fork()` desde una función registrada puede llevar a una condición de carrera entre el hilo principal de Python liberando estados de hilos mientras las rutinas internas de `threading` o el nuevo proceso intentan usar ese estado. Esto puede provocar fallos en lugar de un cierre limpio.

Distinto en la versión 3.12: Los intentos de iniciar un nuevo hilo o `bifurcar` un nuevo proceso en una función registrada conducen ahora a un `RuntimeError`.

atexit.unregister(*func*)

Remove *func* from the list of functions to be run at interpreter shutdown. `unregister()` silently does nothing if *func* was not previously registered. If *func* has been registered more than once, every occurrence of that function in the `atexit` call stack will be removed. Equality comparisons (==) are used internally during unregistration, so function references do not need to have matching identities.

Ver también

Módulo `readline`
Useful example of `atexit` to read and write `readline` history files.

`atexit` Example

El siguiente ejemplo muestra cómo un módulo puede inicializar un contador desde un archivo cuando se importa, y guardar el valor del contador actualizado automáticamente cuando finaliza el programa, sin necesidad de que la aplicación realice una llamada explícita en este módulo cuando el intérprete se detiene.

```
try:
    with open('counterfile') as infile:
        _count = int(infile.read())
except FileNotFoundError:
    _count = 0

def incrcounter(n):
    global _count
    _count = _count + n

def savecounter():
    with open('counterfile', 'w') as outfile:
        outfile.write('%d' % _count)

import atexit

atexit.register(savecounter)
```

Los argumentos posicionales y de palabras clave también se pueden pasar a la función **`register()`** para volver a pasar a la función registrada cuando se llama:

```
def goodbye(name, adjective):
    print('Goodbye %s, it was %s to meet you.' % (name,

import atexit

atexit.register(goodbye, 'Donny', 'nice')
# or:
atexit.register(goodbye, adjective='nice', name='Donny')
```

Usar como un decorator:

```
import atexit

@atexit.register
def goodbye():
    print('You are now leaving the Python sector.')
```

Esto solo funciona con funciones que se puedan llamar sin argumentos.

`traceback` — Print or retrieve a stack traceback

Código fuente: Lib/traceback.py [https://github.com/python/cpython/tree/3.14/Lib/traceback.py]

This module provides a standard interface to extract, format and print stack traces of Python programs. It is more flexible than the interpreter's default traceback display, and therefore makes it possible to configure certain aspects of the output. Finally, it contains a utility for capturing enough information about an exception to print it later, without the need to save a reference to the actual exception. Since exceptions can be the roots of large objects graph, this utility can significantly improve memory management.

The module uses traceback objects — these are objects of type **`types.TracebackType`**, which are assigned to the **`__traceback__`** field of **`BaseException`** instances.

Ver también

Módulo `faulthandler`
Usado para volcar explícitamente los rastreos Python, en un error, después de un tiempo de espera (*timeout*), o en una señal de usuario.

Módulo `pdb`
Depurador interactivo de código fuente para programas Python.

The module's API can be divided into two parts:

- Module-level functions offering basic functionality, which are useful for interactive inspection of exceptions and tracebacks.

- `TracebackException` class and its helper classes `StackSummary` and `FrameSummary`. These offer both more flexibility in the output generated and the ability to store the information necessary for later formatting without holding references to actual exception and traceback objects.

Added in version 3.13: Output is colorized by default and can be controlled using environment variables.

Module-Level Functions

traceback.print_tb(*tb*, *limit=None*, *file=None*)

Print up to *limit* stack trace entries from traceback object *tb* (starting from the caller's frame) if *limit* is positive. Otherwise, print the last `abs(limit)` entries. If *limit* is omitted or `None`, all entries are printed. If *file* is omitted or `None`, the output goes to `sys.stderr`; otherwise it should be an open file or file-like object to receive the output.

Nota

The meaning of the *limit* parameter is different than the meaning of `sys.tracebacklimit`. A negative *limit* value corresponds to a positive value of `sys.tracebacklimit`, whereas the behaviour of a positive *limit* value cannot be achieved with `sys.tracebacklimit`.

Distinto en la versión 3.5: Soporte para *limit* negativo añadido.

traceback.print_exception(*exc*, */*, [*value*, *tb*,]*limit=None*, *file=None*, *chain=True*)

Print exception information and stack trace entries from traceback object *tb* to *file*. This differs from `print_tb()` in the following ways:

- si *tb* no es `None`, muestra una cabecera `Traceback`

`(most recent call last):`

- imprime el tipo de excepción y *value* después del seguimiento de la pila

- si *type(value)* es **`SyntaxError`** y *value* tiene el formato apropiado, muestra la línea donde el error sintáctico ha ocurrido con un cursor indicando la posición aproximada del error.

Desde Python 3.10, en lugar de pasar *value* y *tb*, se puede pasar un objeto de excepción como primer argumento. Si se proporcionan *value* y *tb*, el primer argumento se ignora para proporcionar compatibilidad con versiones anteriores.

The optional *limit* argument has the same meaning as for **`print_tb()`**. If *chain* is true (the default), then chained exceptions (the **`__cause__`** or **`__context__`** attributes of the exception) will be printed as well, like the interpreter itself does when printing an unhandled exception.

Distinto en la versión 3.5: El argumento *etype* es ignorado e infiere desde el tipo de *value*.

Distinto en la versión 3.10: El parámetro *etype* ha cambiado de nombre a *exc* y ahora es solo posicional.

traceback.print_exc(*limit=None, file=None, chain=True*)

This is a shorthand for `print_exception(sys.exception(), limit=limit, file=file, chain=chain)`.

traceback.print_last(*limit=None, file=None, chain=True*)

This is a shorthand for `print_exception(sys.last_exc, limit=limit, file=file, chain=chain)`. In general it will work only after an exception has reached an interactive prompt (see **`sys.last_exc`**).

traceback.print_stack(*f=None, limit=None, file=None*)

Print up to *limit* stack trace entries (starting from the invocation point) if *limit* is positive. Otherwise, print the last `abs(limit)` entries. If *limit* is omitted or `None`, all entries are printed. The optional *f* argument can be used to specify an alternate stack frame to start. The optional *file* argument has the same meaning as for **`print_tb()`**.

Distinto en la versión 3.5: Soporte para *limit* negativo añadido.

traceback.extract_tb(*tb*, *limit=None*)

Return a **`StackSummary`** object representing a list of «pre-processed» stack trace entries extracted from the traceback object *tb*. It is useful for alternate formatting of stack traces. The optional *limit* argument has the same meaning as for **`print_tb()`**. A «pre-processed» stack trace entry is a **`FrameSummary`** object containing attributes **`filename`**, **`lineno`**, **`name`**, and **`line`** representing the information that is usually printed for a stack trace.

traceback.extract_stack(*f=None*, *limit=None*)

Extract the raw traceback from the current stack frame. The return value has the same format as for **`extract_tb()`**. The optional *f* and *limit* arguments have the same meaning as for **`print_stack()`**.

traceback.print_list(*extracted_list*, *file=None*)

Print the list of tuples as returned by **`extract_tb()`** or **`extract_stack()`** as a formatted stack trace to the given file. If *file* is `None`, the output is written to **`sys.stderr`**.

traceback.format_list(*extracted_list*)

Dada una lista de tuplas u objetos **`FrameSummary`** según lo retornado por **`extract_tb()`** o **`extract_stack()`**, retorna una lista de cadenas preparadas para ser mostradas. Cada cadena en la lista resultante corresponde con el elemento con el mismo índice en la lista de argumentos. Cada cadena finaliza en una nueva línea; las cadenas pueden contener nuevas líneas internas también, para aquellos

elementos cuya línea de texto de origen no es `None`.

traceback.format_exception_only(*exc*, /, [*value*,]*, *show_group=False*)

Format the exception part of a traceback using an exception value such as given by **`sys.last_value`**. The return value is a list of strings, each ending in a newline. The list contains the exception's message, which is normally a single string; however, for **`SyntaxError`** exceptions, it contains several lines that (when printed) display detailed information about where the syntax error occurred. Following the message, the list contains the exception's **`notes`**.

Desde Python 3,10, en lugar de pasar *value*, un objeto de excepción se puede pasar como primer argumento. Si se proporciona *value*, el primer argumento se ignora el fin de proporcionar compatibilidad hacia atrás.

When *show_group* is `True`, and the exception is an instance of **`BaseExceptionGroup`**, the nested exceptions are included as well, recursively, with indentation relative to their nesting depth.

Distinto en la versión 3.10: El parámetro *etype* ha cambiado de nombre a *exc* y ahora es solo posicional.

Distinto en la versión 3.11: The returned list now includes any **`notes`** attached to the exception.

Distinto en la versión 3.13: show_group parameter was added.

traceback.format_exception(*exc*, /, [*value*, *tb*,]*limit=None*, *chain=True*)

Formatea una traza de pila y la información de la excepción. Los argumentos tienen el mismo significado que los argumentos correspondientes a **`print_exception()`**. El valor retornado es una lista de cadenas, acabando cada una en una nueva línea y algunas contienen nuevas líneas internas. Cuando estas líneas son concatenadas y mostradas, exactamente el mismo texto es mostrado como hace **`print_exception()`**.

Distinto en la versión 3.5: El argumento *etype* es ignorado e infiere desde el tipo de *value*.

Distinto en la versión 3.10: El comportamiento y la firma de esta función se modificaron para coincidir con **`print_exception()`**.

traceback.format_exc(*limit=None*, *chain=True*)

Esto es como `print_exc(limit)` pero retorna una cadena en lugar de imprimirlo en un archivo.

traceback.format_tb(*tb*, *limit=None*)

Un atajo para `format_list(extract_tb(tb, limit))`.

traceback.format_stack(*f=None*, *limit=None*)

Un atajo para `format_list(extract_stack(f, limit))`.

traceback.clear_frames(*tb*)

Clears the local variables of all the stack frames in a traceback *tb* by calling the **`clear()`** method of each frame object.

Added in version 3.4.

traceback.walk_stack(*f*)

Walk a stack following **`f.f_back`** from the given frame, yielding the frame and line number for each frame. If *f* is `None`, the current stack is used. This helper is used with **`StackSummary.extract()`**.

Added in version 3.5.

Distinto en la versión 3.14: This function previously returned a generator that would walk the stack when first iterated over.

The generator returned now is the state of the stack when `walk_stack` is called.

traceback.walk_tb(*tb*)

Walk a traceback following **`tb_next`** yielding the frame and line number for each frame. This helper is used with **`StackSummary.extract()`**.

Added in version 3.5.

`TracebackException` Objects

Added in version 3.5.

`TracebackException` objects are created from actual exceptions to capture data for later printing. They offer a more lightweight method of storing this information by avoiding holding references to traceback and frame objects. In addition, they expose more options to configure the output compared to the module-level functions described above.

class traceback.TracebackException(*exc_type*, *exc_value*, *exc_traceback*, ***, *limit=None*, *lookup_lines=True*, *capture_locals=False*, *compact=False*, *max_group_width=15*, *max_group_depth=10*)

Capture an exception for later rendering. The meaning of *limit*, *lookup_lines* and *capture_locals* are as for the **`StackSummary`** class.

If *compact* is true, only data that is required by **`TracebackException`**'s **`format()`** method is saved in the class attributes. In particular, the **`__context__`** field is calculated only if **`__cause__`** is `None` and **`__suppress_context__`** is false.

Tenga en cuenta que cuando se capturan locales, también se muestran en el rastreo.

max_group_width (*anchura máxima del grupo*) y *max_group_depth*

(*profundidad máxima del grupo*) controlan el formato del grupo de excepciones (ver `BaseExceptionGroup`). La profundidad (*depth*) se refiere al nivel de anidamiento del grupo, y la anchura (*width*) se refiere al tamaño de una excepción simple perteneciente al arreglo del grupo de excepciones. El formato de salida se trunca cuando se excede el límite.

Distinto en la versión 3.10: Se agregó el parámetro *compact*.

Distinto en la versión 3.11: Se agregaron los parámetros *max_group_width* y *max_group_depth*

__cause__

A **`TracebackException`** of the original `__cause__`.

__context__

A **`TracebackException`** of the original `__context__`.

exceptions

If `self` represents an `ExceptionGroup`, this field holds a list of **`TracebackException`** instances representing the nested exceptions. Otherwise it is `None`.

Added in version 3.11.

__suppress_context__

The `__suppress_context__` value from the original exception.

__notes__

The `__notes__` value from the original exception, or `None` if the exception does not have any notes. If it is not `None` is it formatted in the traceback after the exception string.

Added in version 3.11.

stack

Una clase `StackSummary` representando el seguimiento de pila.

exc_type

La clase del seguimiento de pila original.

Obsoleto desde la versión 3.13.

exc_type_str

String display of the class of the original exception.

Added in version 3.13.

filename

Para errores sintácticos - el nombre del archivo donde el error ha ocurrido.

lineno

Para errores sintácticos - el número de línea donde el error ha ocurrido.

end_lineno

Para errores sintácticos - el número de línea donde el error ha ocurrido. Puede ser `None` si no está presente.

Added in version 3.10.

text

Para errores sintácticos - el texto donde el error ha ocurrido.

offset

Para errores sintácticos - el *offset* en el texto donde el error ha ocurrido.

end_offset

Para errores sintácticos - el *offset* en el texto donde el error ha ocurrido. Puede ser `None` si no está presente.

Added in version 3.10.

msg

Para errores sintácticos - el mensaje de error del compilador.

classmethod from_exception(*exc*, ***, *limit=None*, *lookup_lines=True*, *capture_locals=False*)

Captura una excepción para su posterior procesado. *limit*, *lookup_lines* y *capture_locals* son como para la clase **`StackSummary`**.

Tenga en cuenta que cuando se capturan locales, también se muestran en el rastreo.

print(***, *file=None*, *chain=True*)

Imprime en *file* (`sys.stderr` por defecto) la información de la excepción retornada por **`format()`**.

Added in version 3.11.

format(***, *chain=True*)

Formatea la excepción.

If *chain* is not `True`, **`__cause__`** and **`__context__`** will not be formatted.

El valor retornado es un generador de cadenas, donde cada una acaba en una nueva línea y algunas contienen nuevas líneas internas. **`print_exception()`** es un contenedor alrededor de este método que simplemente muestra las líneas de un archivo.

format_exception_only(***, *show_group=False*)

Formatea la parte de la excepción de un seguimiento de pila.

El valor retornado es un generador de cadenas, donde cada una acaba en una nueva línea.

When *show_group* is `False`, the generator emits the exception's message followed by its notes (if it has any). The exception message is normally a single string; however, for **`SyntaxError`** exceptions, it consists of several lines that (when printed) display detailed information about where the syntax error occurred.

When *show_group* is `True`, and the exception is an instance of **`BaseExceptionGroup`**, the nested exceptions are included as well, recursively, with indentation relative to their nesting depth.

Distinto en la versión 3.11: The exception's **`notes`** are now included in the output.

Distinto en la versión 3.13: Added the *show_group* parameter.

`StackSummary` Objects

Added in version 3.5.

`StackSummary` objects represent a call stack ready for formatting.

class traceback.StackSummary

classmethod extract(*frame_gen*, ***, *limit=None*, *lookup_lines=True*, *capture_locals=False*)

Construct a **`StackSummary`** object from a frame generator (such as is returned by **`walk_stack()`** or **`walk_tb()`**).

If *limit* is supplied, only this many frames are taken from *frame_gen*. If *lookup_lines* is `False`, the returned

`FrameSummary` objects will not have read their lines in yet, making the cost of creating the **`StackSummary`** cheaper (which may be valuable if it may not actually get formatted). If *capture_locals* is `True` the local variables in each **`FrameSummary`** are captured as object representations.

Distinto en la versión 3.12: Las excepciones lanzadas desde **`repr()`** en una variable local (cuando *capture_locals* es `True`) no son propagadas al invocador.

classmethod from_list(*a_list*)

Construct a **`StackSummary`** object from a supplied list of **`FrameSummary`** objects or old-style list of tuples. Each tuple should be a 4-tuple with *filename*, *lineno*, *name*, *line* as the elements.

format()

Returns a list of strings ready for printing. Each string in the resulting list corresponds to a single frame from the stack. Each string ends in a newline; the strings may contain internal newlines as well, for those items with source text lines.

Para secuencias largas del mismo marco y línea, se muestran las primeras repeticiones, seguidas de una línea de resumen que indica el número exacto de repeticiones adicionales.

Distinto en la versión 3.6: Las secuencias largas de cuadros repetidos ahora se abrevian.

format_frame_summary(*frame_summary*)

Returns a string for printing one of the frames involved in the stack. This method is called for each **`FrameSummary`** object to be printed by **`StackSummary.format()`**. If it returns `None`, the frame is omitted from the output.

Added in version 3.11.

`FrameSummary` Objects

Added in version 3.5.

A **`FrameSummary`** object represents a single frame in a traceback.

class traceback.FrameSummary(*filename*, *lineno*, *name*, ***, *lookup_line=True*, *locals=None*, *line=None*, *end_lineno=None*, *colno=None*, *end_colno=None*)

Represents a single frame in the traceback or stack that is being formatted or printed. It may optionally have a stringified version of the frame's locals included in it. If *lookup_line* is `False`, the source code is not looked up until the **`FrameSummary`** has the **`line`** attribute accessed (which also happens when casting it to a **`tuple`**). **`line`** may be directly provided, and will prevent line lookups happening at all. *locals* is an optional local variable mapping, and if supplied the variable representations are stored in the summary for later display.

`FrameSummary` instances have the following attributes:

filename

The filename of the source code for this frame. Equivalent to accessing **`f.f_code.co_filename`** on a frame object *f*.

lineno

The line number of the source code for this frame.

name

Equivalent to accessing **`f.f_code.co_name`** on a frame object *f*.

line

A string representing the source code for this frame,

with leading and trailing whitespace stripped. If the source is not available, it is `None`.

end_lineno

The last line number of the source code for this frame. By default, it is set to `lineno` and indexation starts from 1.

Distinto en la versión 3.13: The default value changed from `None` to `lineno`.

colno

The column number of the source code for this frame. By default, it is `None` and indexation starts from 0.

end_colno

The last column number of the source code for this frame. By default, it is `None` and indexation starts from 0.

Examples of Using the Module-Level Functions

Este ejemplo sencillo implementa un bucle de lectura, evaluación e impresión básico, similar a (pero menos útil) el bucle del intérprete interactivo estándar de Python. Para una implementación más completa del bucle del intérprete, ir al módulo **`code`**

```
import sys, traceback

def run_user_code(envdir):
    source = input(">>> ")
    try:
        exec(source, envdir)
    except Exception:
        print("Exception in user code:")
        print("-"*60)
        traceback.print_exc(file=sys.stdout)
        print("-"*60)

envdir = {}
while True:
    run_user_code(envdir)
```

El siguiente ejemplo demuestra las diferentes manera para mostrar y formatear la excepción y el seguimiento de pila:

```
import sys, traceback

def lumberjack():
    bright_side_of_life()

def bright_side_of_life():
    return tuple()[0]

try:
    lumberjack()
except IndexError as exc:
    print("*** print_tb:")
    traceback.print_tb(exc.__traceback__, limit=1, file=
    print("*** print_exception:")
    traceback.print_exception(exc, limit=2, file=sys.std
    print("*** print_exc:")
    traceback.print_exc(limit=2, file=sys.stdout)
    print("*** format_exc, first and last line:")
    formatted_lines = traceback.format_exc().splitlines(
    print(formatted_lines[0])
    print(formatted_lines[-1])
    print("*** format_exception:")
    print(repr(traceback.format_exception(exc)))
    print("*** extract_tb:")
    print(repr(traceback.extract_tb(exc.__traceback__)))
    print("*** format_tb:")
    print(repr(traceback.format_tb(exc.__traceback__)))
    print("*** tb_lineno:", exc.__traceback__.tb_lineno)
```

La salida para el ejemplo podría ser similar a esto:

```
*** print_tb:
  File "<doctest...>", line 10, in <module>
    lumberjack()
    ~~~~~~~~~~^^
*** print_exception:
Traceback (most recent call last):
  File "<doctest...>", line 10, in <module>
    lumberjack()
    ~~~~~~~~~~^^
  File "<doctest...>", line 4, in lumberjack
    bright_side_of_life()
    ~~~~~~~~~~~~~~~~~~~^^
IndexError: tuple index out of range
*** print_exc:
Traceback (most recent call last):
  File "<doctest...>", line 10, in <module>
    lumberjack()
    ~~~~~~~~~~^^
  File "<doctest...>", line 4, in lumberjack
    bright_side_of_life()
    ~~~~~~~~~~~~~~~~~~~^^
IndexError: tuple index out of range
*** format_exc, first and last line:
Traceback (most recent call last):
IndexError: tuple index out of range
*** format_exception:
['Traceback (most recent call last):\n',
 '  File "<doctest default[0]>", line 10, in <module>\n
 '  File "<doctest default[0]>", line 4, in lumberjack\
 '  File "<doctest default[0]>", line 7, in bright_side_
 'IndexError: tuple index out of range\n']
*** extract_tb:
[<FrameSummary file <doctest...>, line 10 in <module>>,
 <FrameSummary file <doctest...>, line 4 in lumberjack>,
 <FrameSummary file <doctest...>, line 7 in bright_side_
*** format_tb:
['  File "<doctest default[0]>", line 10, in <module>\n
 '  File "<doctest default[0]>", line 4, in lumberjack\
```

```
 '  File "<doctest default[0]>", line 7, in bright_side_
*** tb_lineno: 10
```

El siguiente ejemplo muestra las diferentes maneras de imprimir y formatear la pila:

```
>>> import traceback
>>> def another_function():
...     lumberstack()
...
>>> def lumberstack():
...     traceback.print_stack()
...     print(repr(traceback.extract_stack()))
...     print(repr(traceback.format_stack()))
...
>>> another_function()
  File "<doctest>", line 10, in <module>
    another_function()
  File "<doctest>", line 3, in another_function
    lumberstack()
  File "<doctest>", line 6, in lumberstack
    traceback.print_stack()
[('<doctest>', 10, '<module>', 'another_function()'),
 ('<doctest>', 3, 'another_function', 'lumberstack()'),
 ('<doctest>', 7, 'lumberstack', 'print(repr(traceback.e
['  File "<doctest>", line 10, in <module>\n    another_
 '  File "<doctest>", line 3, in another_function\n    l
 '  File "<doctest>", line 8, in lumberstack\n    print(
```

Este último ejemplo demuestra las últimas funciones de formateo:

```
>>> import traceback
>>> traceback.format_list([('spam.py', 3, '<module>', 's
...                        ('eggs.py', 42, 'eggs', 'retu
['  File "spam.py", line 3, in <module>\n    spam.eggs()
 '  File "eggs.py", line 42, in eggs\n    return "bacon"
>>> an_error = IndexError('tuple index out of range')
>>> traceback.format_exception_only(an_error)
['IndexError: tuple index out of range\n']
```

Examples of Using `TracebackException`

With the helper class, we have more options:

```
>>> import sys
>>> from traceback import TracebackException
>>>
>>> def lumberjack():
...     bright_side_of_life()
...
>>> def bright_side_of_life():
...     t = "bright", "side", "of", "life"
...     return t[5]
...
>>> try:
...     lumberjack()
... except IndexError as e:
...     exc = e
...
>>> try:
...     try:
...         lumberjack()
...     except:
...         1/0
... except Exception as e:
...     chained_exc = e
...
>>> # limit works as with the module-level functions
>>> TracebackException.from_exception(exc, limit=-2).pri
Traceback (most recent call last):
  File "<python-input-1>", line 6, in lumberjack
    bright_side_of_life()
    ~~~~~~~~~~~~~~~~~~~^^
  File "<python-input-1>", line 10, in bright_side_of_li
    return t[5]
           ~^^^
IndexError: tuple index out of range
```

```
>>> # capture_locals adds local variables in frames
>>> TracebackException.from_exception(exc, limit=-2, cap
Traceback (most recent call last):
  File "<python-input-1>", line 6, in lumberjack
    bright_side_of_life()
    ~~~~~~~~~~~~~~~~~~~~^^
  File "<python-input-1>", line 10, in bright_side_of_li
    return t[5]
           ~^^^
    t = ("bright", "side", "of", "life")
IndexError: tuple index out of range

>>> # The *chain* kwarg to print() controls whether chai
>>> # exceptions are displayed
>>> TracebackException.from_exception(chained_exc).print
Traceback (most recent call last):
  File "<python-input-19>", line 4, in <module>
    lumberjack()
    ~~~~~~~~~~^^
  File "<python-input-8>", line 7, in lumberjack
    bright_side_of_life()
    ~~~~~~~~~~~~~~~~~~~~^^
  File "<python-input-8>", line 11, in bright_side_of_li
    return t[5]
           ~^^^
IndexError: tuple index out of range

During handling of the above exception, another exceptic

Traceback (most recent call last):
  File "<python-input-19>", line 6, in <module>
    1/0
    ~^~
ZeroDivisionError: division by zero

>>> TracebackException.from_exception(chained_exc).print
Traceback (most recent call last):
  File "<python-input-19>", line 6, in <module>
```

```
    1/0
    ~^~
ZeroDivisionError: division by zero
```

[1]

`__future__` — Future statement definitions

Código fuente: Lib/__future__.py [https://github.com/python/cpython/tree/3.14/Lib/__future__.py]

Imports of the form `from __future__ import feature` are called future statements. These are special-cased by the Python compiler to allow the use of new Python features in modules containing the future statement before the release in which the feature becomes standard.

While these future statements are given additional special meaning by the Python compiler, they are still executed like any other import statement and the `__future__` exists and is handled by the import system the same way any other Python module would be. This design serves three purposes:

- Para evitar confundir las herramientas existentes que analizan las declaraciones de importación y esperan encontrar los módulos que están importando.
- To document when incompatible changes were introduced, and when they will be — or were — made mandatory. This is a form of executable documentation, and can be inspected programmatically via importing `__future__` and examining its contents.
- To ensure that future statements run under releases prior to Python 2.1 at least yield runtime exceptions (the import of `__future__` will fail, because there was no module of that name prior to 2.1).

Module Contents

No feature description will ever be deleted from `__future__`.

Since its introduction in Python 2.1 the following features have found their way into the language using this mechanism:

característica	opcional en	obligatorio en	efecto
nested_scopes	2.1.0b1	2.2	**PEP 227**: *Ámbitos anidados estáticamente*
generators	2.2.0a1	2.3	**PEP 255**: *Generadores simples*
division	2.2.0a2	3.0	**PEP 238**: *Cambio de operador de división*
absolute_import	2.5.0a1	3.0	**PEP 328**: *Importaciones: Multilínea y Absoluto/Relativo*
with_statement	2.5.0a1	2.6	**PEP 343**: *The "with" Statement*
print_function	2.6.0a2	3.0	**PEP 3105**: *Hacer de print una función*
unicode_literals	2.6.0a2	3.0	**PEP 3112**: *Bytes literales en Python 3000*
generator_stop	3.5.0b1	3.7	**PEP 479**: *Manejo de StopIteration dentro de generadores*
annotations	3.7.0b1	Never [1]	**PEP 563**: *Postponed evaluation of annotations*, **PEP 649**: *Deferred evaluation of annotations using descriptors*

class _future_._Feature

Cada declaración en `__future__.py` tiene la forma:

```
FeatureName = _Feature(OptionalRelease, MandatoryRe
                       CompilerFlag)
```

donde, normalmente, *OptionalRelease* es menor que *MandatoryRelease* y ambos son 5-tuplas de la misma forma que **`sys.version_info`**:

```
(PY_MAJOR_VERSION, # the 2 in 2.1.0a3; an int
 PY_MINOR_VERSION, # the 1; an int
 PY_MICRO_VERSION, # the 0; an int
 PY_RELEASE_LEVEL, # "alpha", "beta", "candidate" o
 PY_RELEASE_SERIAL # the 3; an int
)
```

_Feature.getOptionalRelease()

OptionalRelease registra la primera versión en la que se aceptó la característica.

_Feature.getMandatoryRelease()

En el caso de un *MandatoryRelease* que aún no se ha producido, *MandatoryRelease* predice el lanzamiento en el que la característica pasará a formar parte del lenguaje.

De otro modo, *MandatoryRelease* registra cuándo la característica se convirtió en parte del lenguaje; en versiones en o después de este, los módulos ya no necesitan una declaración futura para usar la característica en cuestión, pero pueden continuar usando dichas importaciones.

MandatoryRelease may also be `None`, meaning that a planned feature got dropped or that it is not yet decided.

_Feature.compiler_flag

CompilerFlag is the (bitfield) flag that should be passed in the fourth argument to the built-in function **`compile()`** to enable the feature in dynamically compiled code. This flag is stored in the **`_Feature.compiler_flag`** attribute on **`_Feature`** instances.

`from __future__ import annotations` was previously scheduled to become mandatory in Python 3.10, but the change was delayed and ultimately canceled. This feature will eventually be deprecated and removed. See **PEP 649** [https://peps.python.org/pep-0649/] and **PEP 749** [https://peps.python.org/pep-0749/].

Ver también

Declaraciones Futuras

Cómo trata el compilador las importaciones futuras.

PEP 236 [https://peps.python.org/pep-0236/] - Back to the _future_

The original proposal for the _future_ mechanism.

`gc` — Garbage Collector interface

Este módulo proporciona una interfaz para el recolector de basura opcional (recolector de basura cíclico generacional). Proporciona la capacidad de deshabilitar el recolector, ajustar la frecuencia de recolección y establecer opciones de depuración. También proporciona acceso a objetos inaccesibles (*unreachables*) que el recolector encontró pero no pudo liberar. Dado que el recolector de basura complementa el conteo de referencias, que ya se utiliza en Python, es posible desactivarlo siempre que se esté seguro de que el programa no crea referencias cíclicas. La recolección automática se puede desactivar llamando a `gc.disable()`. Para depurar un programa con fugas de memoria, se debe llamar a `gc.set_debug(gc.DEBUG_LEAK)`. Se debe tener en cuenta que la llamada anterior incluye `gc.DEBUG_SAVEALL`, lo que hace que los objetos recolectados se guarden en *gc.garbage* para su inspección.

The **`gc`** module provides the following functions:

gc.enable()

Habilita la recolección automática de basura.

gc.disable()

Deshabilita la recolección automática de basura.

gc.isenabled()

Retorna `True` si la recolección automática está habilitada.

gc.collect(*generation=2*)

With no arguments, run a full collection. The optional argument *generation* may be an integer specifying which generation to collect (from 0 to 2). A **`ValueError`** is raised if the generation number is invalid. The sum of collected objects and uncollectable objects is returned.

Las listas libres mantenidas para varios tipos incorporados son borradas cada vez que se ejecuta una recolección completa o una recolección de la generación más alta (2). No obstante, no todos los elementos de algunas listas libres pueden ser liberados, particularmente **`float`**, debido a su implementación particular.

El efecto de llamar a `gc.collect()` mientras el intérprete ya está realizando una recolección es indefinido.

Distinto en la versión 3.14: `generation=1` performs an increment of collection.

Distinto en la versión 3.14.5: `generation=1` performs collection of the middle generation.

gc.set_debug(*flags*)

Establece las flags de depuración para la recolección de basura. La información de depuración se escribirá en `sys.stderr`. A continuación se puede consultar una lista con las flags de depuración disponibles, que se pueden combinar mediante operaciones de bits para controlar la depuración.

gc.get_debug()

Retorna las flags de depuración actualmente establecidas.

gc.get_objects(*generation=None*)

Returns a list of all objects tracked by the collector, excluding the list returned. If *generation* is not `None`, return only the objects tracked by the collector that are in that generation.

Distinto en la versión 3.8: Se agregó el parámetro *generation*.

Distinto en la versión 3.14: Generation 1 is removed

Distinto en la versión 3.14.5: Generation 1 is reintroduced to maintain GC behavior from 3.13.

Lanza un evento de auditoría `gc.get_objects` con el argumento `generation`.

gc.get_stats()

Retorna una lista de tres diccionarios, uno por cada generación, que contienen estadísticas de recolección desde el inicio del intérprete. El número de claves puede cambiar en el futuro, pero actualmente cada diccionario contendrá los siguientes elementos:

- `collections` es el número de veces que se recolectó esta generación;
- `collected` es el número total de objetos recolectados dentro de esta generación;
- `uncollectable` es el número total de objetos que se consideraron no recolectables (y por lo tanto, se movieron a la lista **`garbage`**) dentro de esta generación.

Added in version 3.4.

gc.set_threshold(*threshold0*[, *threshold1*[, *threshold2*]])

Establece los umbrales de recolección (la frecuencia de recolección). Si se establece *threshold0* en cero se deshabilita la recolección.

The GC classifies objects into three generations depending on how many collection sweeps they have survived. New objects are placed in the youngest generation (generation `0`). If an object survives a collection it is moved into the next older generation. Since generation `2` is the oldest generation, objects in that generation remain there after a collection. In order to decide when to run, the collector keeps track of the number object allocations and deallocations since the last collection. When the number of allocations minus the number of deallocations exceeds *threshold0*, collection starts. Initially only generation `0` is examined. If generation `0` has been examined more than *threshold1* times since generation `1` has been examined, then generation `1` is examined as well. With the third generation, things are a bit more complicated, see Collecting the oldest generation [https://github.com/python/cpython/blob/ff0ef0a54bef26fc507fbf9b7a6009eb7d3f17f5/InternalDocs/garbage_collector.md#collecting-the-oldest-generation] for more information.

In the free-threaded build, the increase in process memory usage is also checked before running the collector. If the memory usage has not increased by 10% since the last collection and the net number of object allocations has not exceeded 40 times *threshold0*, the collection is not run.

See Garbage collector design [https://github.com/python/cpython/blob/3.14/InternalDocs/garbage_collector.md] for more information.

Distinto en la versión 3.14: threshold2 is ignored

Distinto en la versión 3.14.5: threshold2 is restored to match Python 3.13 behavior.

gc.get_count()

Retorna los recuentos de la recolección actual como una tupla de la forma `(count0, count1, count2)`.

gc.get_threshold()

Retorna los umbrales de la recolección actual como una tupla de la forma `(threshold0, threshold1, threshold2)`.

gc.get_referrers(**objs*)

Retorna la lista de objetos que referencian de forma directa a cualquiera de *objs*. Esta función solo localizará aquellos contenedores que admitan la recolección de basura; no se localizarán los tipos de extensión que referencian a otros objetos pero que no admiten la recolección de basura.

Téngase en cuenta que los objetos que ya se han desreferenciado, pero que tienen referencias cíclicas y aún no han sido recolectados por el recolector de basura pueden enumerarse entre las referencias resultantes. Para obtener solo los objetos activos actualmente, llame a `collect()` antes de llamar a `get_referrers()`.

Advertencia

Se debe tener un especial cuidado cuando se usan objetos retornados por `get_referrers()` dado que algunos de ellos aún podrían estar en construcción y por tanto en un estado temporalmente inválido. Debe evitarse el uso de `get_referrers()` para cualquier propósito que no sea la depuración.

Lanza un evento de auditoría `gc.get_referrers` con el argumento `objs`.

gc.get_referents(**objs*)

Retorna una lista de objetos a los que hace referencia directamente cualquiera de los argumentos. Las referencias retornadas son aquellos objetos visitados por los métodos `tp_traverse` a nivel de C de los argumentos (si los hay) y es posible que no todos los objetos sean directamente accesibles realmente. Los métodos `tp_traverse` solo son compatibles con los objetos que admiten recolección de basura y solo se requieren para visitar objetos que pueden estar involucrados en un ciclo de referencias. Lo que quiere decir que, por ejemplo, si se puede acceder directamente a un número entero desde uno de los argumento, ese objeto entero puede aparecer o no en la lista de resultados.

Lanza un evento de auditoría `gc.get_referents` con el argumento `objs`.

gc.is_tracked(*obj*)

Retorna `True` si el recolector de basura rastrea actualmente el objeto y `False` en caso contrario. Como regla general, las instancias de tipos atómicos no se rastrean y las instancias de tipos no atómicos (contenedores, objetos definidos por el usuario …) sí. Sin embargo, algunas optimizaciones específicas de tipo pueden estar presentes para suprimir el impacto del recolector de basura en instancias simples (por ejemplo, diccionarios que contienen solo claves y valores atómicos):

```
>>> gc.is_tracked(0)
False
>>> gc.is_tracked("a")
False
>>> gc.is_tracked([])
True
>>> gc.is_tracked({})
False
>>> gc.is_tracked({"a": 1})
True
```

Added in version 3.1.

gc.is_finalized(*obj*)

Retorna `True` si el objeto dado ha sido finalizado por el recolector de basura, `False` en caso contrario.

```
>>> x = None
>>> class Lazarus:
...     def __del__(self):
...         global x
...         x = self
...
>>> lazarus = Lazarus()
>>> gc.is_finalized(lazarus)
False
>>> del lazarus
>>> gc.is_finalized(x)
True
```

Added in version 3.9.

gc.freeze()

Congela todos los objetos rastreados por el recolector de basura; los mueve a una generación permanente y los ignora en todas las recolecciones futuras.

Si un proceso va a llamar a `fork()` sin llamar a `exec()`, evitar la copia en escritura innecesaria en procesos secundarios maximizará el intercambio de memoria y reducirá el uso general de la memoria. Esto requiere tanto evitar la creación de «agujeros» liberados en las páginas de memoria del proceso principal y garantizar que las colecciones de GC en los procesos secundarios no tocarán el contador `gc_refs` de objetos de larga duración que se originan en el proceso principal. Para lograr ambas cosas, llame a `gc.disable()` al principio del proceso principal, a `gc.freeze()` justo antes de `fork()` y a `gc.enable()` al principio. en procesos secundarios.

Added in version 3.7.

gc.unfreeze()

Descongela los objetos de la generación permanente y vuelve a colocarlos en la generación más antigua.

Added in version 3.7.

gc.get_freeze_count()

Retorna el número de objetos de la generación permanente.

Added in version 3.7.

Las siguientes variables se proporcionan para acceso de solo lectura (se pueden mutar los valores pero no se deben vincular de nuevo):

gc.garbage

Una lista de objetos que el recolector consideró inaccesibles pero que no pudieron ser liberados (objetos que no se pueden recolectar). A partir de Python 3.4, esta lista debe estar vacía la mayor parte del tiempo, excepto cuando se utilizan instancias de tipos de extensión C en los que la ranura (*slot*) `tp_del` no sea `NULL`.

Si se establece **`DEBUG_SAVEALL`**, todos los objetos inaccesibles se agregarán a esta lista en lugar de ser liberados.

Distinto en la versión 3.2: Si la lista no está vacía en el momento del interpreter shutdown, se emite un **`ResourceWarning`**, que es silencioso de forma predeterminada. Si se establece **`DEBUG_UNCOLLECTABLE`**, además se imprimen todos los objetos no recolectables.

Distinto en la versión 3.4: Siguiendo con **PEP 442** [https://peps.python.org/pep-0442/], los objetos con un método **`__del__()`** ya no terminan en **`gc.garbage`**.

gc.callbacks

Una lista de retrollamadas que el recolector de basura invocará antes y después de la recolección. Las retrollamadas serán llamadas con dos argumentos, *phase* e *info*.

phase puede tomar uno de estos valores:

> «start»: la recolección de basura está a punto de comenzar.
>
> «stop»: la recolección de basura ha terminado.

info es un diccionario que proporciona información adicional para la retrollamada. Las siguientes claves están definidas actualmente:

> «generation»: la generación más antigua que ha sido recolectada.
>
> «collected»: cuando *phase* es «stop», el número de objetos recolectados satisfactoriamente.
>
> «uncollectable»: cuando *phase* es «stop», el

número de objetos que no pudieron ser recolectados y fueron ubicados en **`garbage`**.

Las aplicaciones pueden agregar sus propias retrollamadas a esta lista. Los principales casos de uso son:

Recopilar estadísticas sobre la recolección de basura, como la frecuencia con la que se recolectan varias generaciones y cuánto tiempo lleva la recolección.

Permitir que las aplicaciones identifiquen y borren sus propios tipos no recolectables cuando aparecen en **`garbage`**.

Added in version 3.3.

Las siguientes constantes son proporcionadas para ser usadas con **`set_debug()`**:

gc.DEBUG_STATS

Imprime estadísticas durante la recolección. Esta información puede resultar útil para ajustar la frecuencia de recolección.

gc.DEBUG_COLLECTABLE

Imprime información sobre los objetos recolectables encontrados.

gc.DEBUG_UNCOLLECTABLE

Imprime información sobre los objetos no recolectables encontrados (objetos inaccesibles, pero que el recolector no puede liberar). Estos objetos se agregarán a la lista `garbage`.

Distinto en la versión 3.2: También imprime el contenido de la lista **`garbage`** durante interpreter shutdown, si no está vacía.

gc.DEBUG_SAVEALL

Cuando se establece, todos los objetos inaccesibles encontrados se agregarán a *garbage* en lugar de ser liberados. Esto puede resultar útil para depurar un programa con fugas de memoria.

gc.DEBUG_LEAK

Las flags de depuración necesarias para que el recolector imprima información sobre un programa con fugas de memoria (igual a `DEBUG_COLLECTABLE | DEBUG_UNCOLLECTABLE | DEBUG_SAVEALL`).

`inspect` — Inspect live objects

Código fuente: Lib/inspect.py [https://github.com/python/cpython/tree/3.14/Lib/inspect.py]

The `inspect` module provides several useful functions to help get information about live objects such as modules, classes, methods, functions, tracebacks, frame objects, and code objects. For example, it can help you examine the contents of a class, retrieve the source code of a method, extract and format the argument list for a function, or get all the information you need to display a detailed traceback.

Hay cuatro tipos principales de servicios que ofrece este módulo: comprobación de tipos, obtención del código fuente, inspección de clases y funciones, y examinar la pila del intérprete.

Tipos y miembros

The `getmembers()` function retrieves the members of an object such as a class or module. The functions whose names begin with «is» are mainly provided as convenient choices for the second argument to `getmembers()`. They also help you determine when you can expect to find the following special attributes (see Import-related attributes on module objects for module attributes):

Tipo	Atributo	Descripción
clase	__doc__	cadena de caracteres de documentación
	__name__	nombre con el que se definió esta clase
	__qualname__	nombre calificado
	__module__	nombre del módulo en el que se definió esta clase
	__type_params__	A tuple containing the type parameters of a generic class
método	__doc__	cadena de caracteres de documentación
	__name__	nombre con el que se definió este método
	__qualname__	nombre calificado

	__func__	objeto función que contiene la implementación del método
	__self__	instancia a la que este método está ligado, o `None`
	__module__	nombre del módulo en el cual este método fue definido
función	__doc__	cadena de caracteres de documentación
	__name__	nombre con el que se definió esta función
	__qualname__	nombre calificado
	__code__	objeto de código que contiene la función compilada bytecode
	__defaults__	tupla de cualquier valor por defecto para los parámetros de posición o de palabras clave
	__kwdefaults__	mapeo de cualquier valor predeterminado para parámetros de sólo palabras clave
	__globals__	namespace global en el que se definió esta función
	__builtins__	Espacio de nombres builtins
	__annotations__	mapeo de los nombres de parámetros a las anotaciones; la tecla `"return"` está reservada para las anotaciones de retorno.
	__type_params__	A tuple containing the type parameters of a generic function
	__module__	nombre del módulo en el cual esta función fue definida
traceback	tb_frame	enmarca el objeto a este nivel
	tb_lasti	índice del último intento de instrucción en código de bytes
	tb_lineno	número de línea actual en el código fuente de Python
	tb_next	el siguiente objeto de traceback interno (llamado por este nivel)
marco	f_back	el siguiente objeto exterior del marco (el que llama a este marco)
	f_builtins	construye el namespace visto por este marco
	f_code	objeto de código que se ejecuta en este marco
	f_globals	el namespace global visto por este marco
	f_lasti	índice del último intento de instrucción en código de bytes
	f_lineno	número de línea actual en el código fuente de Python
	f_locals	el namespace local visto por este marco
	f_generator	returns the generator or coroutine object that owns this frame, or `None` if the frame is of a regular function
	f_trace	función de rastreo para este marco, o `None`
	f_trace_lines	indicate whether a tracing event is triggered for each source source line
	f_trace_opcodes	indicate whether per-opcode events are requested
	clear()	used to clear all references to local variables
código	co_argcount	número de argumentos (sin incluir los argumentos de palabras clave, * o ** args)
	co_code	cadena de bytecode compilados en bruto
	co_cellvars	tupla de nombres de variables de celda (referenciados por contener alcances)

	co_consts	tupla de constantes utilizadas en el bytecode
	co_filename	nombre del archivo en el que este objeto código fue creado
	co_firstlineno	número de la primera línea del código fuente de Python
	co_flags	mapa de bits de los flags `CO_*`, leer más aquí
	co_lnotab	mapeo codificado de los números de línea a los índices de bytecode
	co_freevars	tupla de nombres de variables libres (referenciados a través del cierre de una función)
	co_posonlyargcount	número de argumentos solo posicionales
	co_kwonlyargcount	número de argumentos de sólo palabras clave (sin incluir el ** arg)
	co_name	nombre con el que se definió este objeto de código
	co_qualname	nombre completo con el que se definió este objeto de código
	co_names	tupla de nombres que no sean argumentos y funciones locales
	co_nlocals	número de variables locales
	co_stacksize	se requiere espacio en la pila de máquina virtual
	co_varnames	tupla de nombres de argumentos y variables locales
	co_lines()	returns an iterator that yields successive bytecode ranges
	co_positions()	returns an iterator of source code positions for each bytecode instruction
	replace()	returns a copy of the code object with new values
generador	__name__	nombre
	__qualname__	nombre calificado
	gi_frame	marco
	gi_running	¿Está el generador en ejecución?
	gi_suspended	is the generator suspended?
	gi_code	código
	gi_yieldfrom	el objeto siendo iterado por `yield from`, o `None`
asyncgen	__name__	nombre
	__qualname__	nombre calificado
	ag_await	objeto al que se espera, o `None`
	ag_frame	marco
	ag_running	¿Está el generador en ejecución?
	ag_suspended	is the generator suspended?
	ag_code	código
corrutina	__name__	nombre
	__qualname__	nombre calificado
	cr_await	objeto al que se espera, o `None`
	cr_frame	marco
	cr_running	¿Está la corrutina en ejecución?

	cr_suspended	is the coroutine suspended?
	cr_code	code
	cr_origin	donde se creó la corutina, o `None`. Ver `sys.set_coroutine_origin_tracking_depth()`
incorporado	__doc__	cadena de caracteres de documentación
	__name__	nombre original de esta función o método
	__qualname__	nombre calificado
	__self__	instancia a la que está ligada un método, o `None`

Distinto en la versión 3.5: Agrega atributos `__qualname__` y `gi_yieldfrom` a los generadores.

El atributo `__name__` de los generadores se establece ahora a partir del nombre de la función, en lugar del nombre del código, y ahora puede ser modificado.

Distinto en la versión 3.7: Agrega el atributo `cr_origin` a las corutinas.

Distinto en la versión 3.10: Agrega el atributo `__builtins__` a funciones.

Distinto en la versión 3.11: Add `gi_suspended` attribute to generators.

Distinto en la versión 3.11: Add `cr_suspended` attribute to coroutines.

Distinto en la versión 3.12: Add `ag_suspended` attribute to async generators.

Distinto en la versión 3.14: Add `f_generator` attribute to frames.

inspect.getmembers(*object*[, *predicate*])

Retorna todos los miembros de un objeto en una lista de pares `(name, value)` ordenados por nombre. Si se proporciona el argumento *predicate* opcional, que se llamará con el objeto `value` de cada miembro, solo se incluyen los miembros para los que el predicado retorna un valor verdadero.

Nota

`getmembers()` will only return class attributes defined in the metaclass when the argument is a class and those attributes have been listed in the metaclass” custom **`__dir__()`**.

inspect.getmembers_static(*object*[, *predicate*])

Retorna todos los miembros de un objeto en una lista de pares `(name, value)` ordenados por nombre sin activar la búsqueda dinámica a través del protocolo descriptor, _getattr_ o _getattribute_. Opcionalmente, solo retorna miembros que satisfagan un predicado determinado.

Nota

Es posible que **`getmembers_static()`** no pueda recuperar todos los miembros que getmembers puede obtener (como atributos creados dinámicamente) y puede encontrar miembros que getmembers no puede (como descriptores que generan AttributeError). También puede retornar objetos descriptores en lugar de miembros de instancia en algunos casos.

Added in version 3.11.

inspect.getmodulename(*path*)

Retorna el nombre del módulo nombrado por el *ruta* de archivo, sin incluir los nombres de los paquetes adjuntos. La extensión del archivo se comprueba con todas las entradas en **`importlib.machinery.all_suffixes()`**. Si coincide, el componente final de la ruta se retorna con la extensión eliminada. En caso contrario, se retorna `None`.

Ten en cuenta que esta función *sólo* retorna un nombre significativo para los módulos reales de Python - las rutas que potencialmente se refieren a los paquetes de Python seguirán retornando `None`.

Distinto en la versión 3.3: La función se basa directamente en

`importlib`.

inspect.ismodule(*object*)

Retorna `True` si el objeto es un módulo.

inspect.isclass(*object*)

Retorna `True` si el objeto es una clase, ya sea incorporada o creada en código Python.

inspect.ismethod(*object*)

Retorna `True` si el objeto es un método ligado escrito en Python.

inspect.ispackage(*object*)

Return `True` if the object is a package.

Added in version 3.14.

inspect.isfunction(*object*)

Retorna `True` si el objeto es una función de Python, que incluye funciones creadas por una expresión lambda.

inspect.isgeneratorfunction(*object*)

Retorna `True` si el objeto es una función generadora de Python.

Distinto en la versión 3.8: Funciones envueltas en **`functools.partial()`** ahora retornan `True` si la función envuelta es una función Python generadora.

Distinto en la versión 3.13: Functions wrapped in **`functools.partialmethod()`** now return `True` if the wrapped function is a Python generator function.

inspect.isgenerator(*object*)

Retorna `True` si el objeto es un generador.

inspect.iscoroutinefunction(*object*)

Return `True` if the object is a coroutine function (a function defined with an **`async def`** syntax), a **`functools.partial()`** wrapping a coroutine function, or a sync function marked with **`markcoroutinefunction()`**.

Added in version 3.5.

Distinto en la versión 3.8: Funciones envueltas en **`functools.partial()`** ahora retornan `True` si la función envuelta es un coroutine function.

Distinto en la versión 3.12: Sync functions marked with **`markcoroutinefunction()`** now return `True`.

Distinto en la versión 3.13: Functions wrapped in **`functools.partialmethod()`** now return `True` if the wrapped function is a coroutine function.

inspect.markcoroutinefunction(*func*)

Decorator to mark a callable as a coroutine function if it would not otherwise be detected by **`iscoroutinefunction()`**.

This may be of use for sync functions that return a coroutine, if the function is passed to an API that requires **`iscoroutinefunction()`**.

When possible, using an **`async def`** function is preferred. Also acceptable is calling the function and testing the return with **`iscoroutine()`**.

Added in version 3.12.

inspect.iscoroutine(*object*)

Retorna verdadero si el objeto es un coroutine creado por una función **`async def`**.

Added in version 3.5.

inspect.isawaitable(*object*)

Retorna `True` si el objeto puede ser usado en la expresión **`await`**.

Can also be used to distinguish generator-based coroutines from regular generators:

```
import types

def gen():
    yield
@types.coroutine
def gen_coro():
    yield

assert not isawaitable(gen())
assert isawaitable(gen_coro())
```

Added in version 3.5.

inspect.isasyncgenfunction(*object*)

Return `True` if the object is an asynchronous generator function, for example:

```
>>> async def agen():
...     yield 1
...
>>> inspect.isasyncgenfunction(agen)
True
```

Added in version 3.6.

Distinto en la versión 3.8: Functions wrapped in **`functools.partial()`** now return `True` if the wrapped function is an asynchronous generator function.

Distinto en la versión 3.13: Functions wrapped in **`functools.partialmethod()`** now return `True` if the wrapped function is a asynchronous generator function.

inspect.isasyncgen(*object*)

Retorna verdadero si el objeto es un asynchronous generator iterator creado por una función asynchronous generator.

Added in version 3.6.

inspect.istraceback(*object*)

Retorna `True` si el objeto es un traceback.

inspect.isframe(*object*)

Retorna `True` si el objeto es un marco.

inspect.iscode(*object*)

Retorna `True` si el objeto es un código.

inspect.isbuiltin(*object*)

Retorna `True` si el objeto es una función incorporada o un método ligado incorporado.

inspect.ismethodwrapper(*object*)

Retorna `True` si el tipo de objeto es **`MethodWrapperType`**.

Estas son instancias de **`MethodWrapperType`**, como **`__str__()`**, **`__eq__()`** y **`__repr__()`**.

Added in version 3.11.

inspect.isroutine(*object*)

Retorna `True` si el objeto es una función o método definido por el usuario o incorporado.

inspect.isabstract(*object*)

Retorna `True` si el objeto es una clase base abstracta.

inspect.ismethoddescriptor(*object*)

Retorna `True` si el objeto es un descriptor de método, pero no si `ismethod()`, `isclass()`, `isfunction()` o `isbuiltin()` son verdaderos.

This, for example, is true of `int.__add__`. An object passing this test has a `__get__()` method, but not a `__set__()` method or a `__delete__()` method. Beyond that, the set of attributes varies. A `__name__` attribute is usually sensible, and `__doc__` often is.

Methods implemented via descriptors that also pass one of the other tests return `False` from the **`ismethoddescriptor()`** test, simply because the other tests promise more – you can, e.g., count on having the `__func__` attribute (etc) when an object passes `ismethod()`.

Distinto en la versión 3.13: This function no longer incorrectly reports objects with `__get__()` and `__delete__()`, but not `__set__()`, as being method descriptors (such objects are data descriptors, not method descriptors).

inspect.isdatadescriptor(*object*)

Retorna `True` si el objeto es un descriptor de datos.

Data descriptors have a `__set__` or a `__delete__` method. Examples are properties (defined in Python), getsets, and members. The latter two are defined in C and there are more specific tests available for those types, which is robust across Python implementations. Typically, data descriptors will also have `__name__` and `__doc__` attributes (properties, getsets, and members have both of these attributes), but this is not guaranteed.

inspect.isgetsetdescriptor(*object*)

Retorna `True` si el objeto es un descriptor de conjunto.

conjuntos son atributos definidos en módulos de extensión a través de estructuras `PyGetSetDef`. Para implementaciones de Python sin tales tipos, este método siempre retornará `False`.

inspect.ismemberdescriptor(*object*)

Retorna `True` si el objeto es un descriptor de miembro.

Los descriptores de miembros son atributos definidos en los módulos de extensión a través de las estructuras **`PyMemberDef`**. Para implementaciones de Python sin tales tipos, este método siempre retornará `False`.

Recuperar el código fuente

inspect.getdoc(*object*)

Obtiene la cadena de documentación de un objeto, limpiada con **`cleandoc()`**. Si no se proporciona la cadena de documentación para un objeto y el objeto es una clase, un método, una propiedad o un descriptor, recupera la cadena de documentación de la jerarquía de herencia. Retorna `None` si la cadena de documentación no es válida o falta.

Distinto en la versión 3.5: Las cadenas de documentación son ahora heredadas, si no anuladas.

inspect.getcomments(*object*)

Retorna en una sola cadena las líneas de comentarios que preceden inmediatamente al código fuente del objeto (para una clase, función o método), o en la parte superior del archivo fuente de Python (si el objeto es un módulo). Si el código fuente del objeto no está disponible, retorna `None`. Esto podría suceder si el objeto ha sido definido en C o en el shell interactivo.

inspect.getfile(*object*)

Retorna el nombre del archivo (de texto o binario) en el que se definió un objeto. Esto fallará con un **`TypeError`** si el objeto es un módulo, clase o función incorporada.

inspect.getmodule(*object*)

Intenta adivinar en qué módulo se definió un objeto. Retorna

`None` si no se puede determinar el módulo.

inspect.getsourcefile(*object*)

Retorna el nombre del archivo fuente de Python en el que se definió un objeto o `None` si no se puede identificar ninguna forma de obtener la fuente. Esto fallará con un **`TypeError`** si el objeto es un módulo, una clase o una función incorporados.

inspect.getsourcelines(*object*)

Return a list of source lines and starting line number for an object. The argument may be a module, class, method, function, traceback, frame, or code object. The source code is returned as a list of the lines corresponding to the object and the line number indicates where in the original source file the first line of code was found. An **`OSError`** is raised if the source code cannot be retrieved. A **`TypeError`** is raised if the object is a built-in module, class, or function.

Distinto en la versión 3.3: **`OSError`** se lanza en lugar de **`IOError`**, ahora un alias del primero.

inspect.getsource(*object*)

Return the text of the source code for an object. The argument may be a module, class, method, function, traceback, frame, or code object. The source code is returned as a single string. An **`OSError`** is raised if the source code cannot be retrieved. A **`TypeError`** is raised if the object is a built-in module, class, or function.

Distinto en la versión 3.3: **`OSError`** se lanza en lugar de **`IOError`**, ahora un alias del primero.

inspect.cleandoc(*doc*)

Limpiar la indentación de los docstrings que están indentados para alinearse con los bloques de código.

Todos los espacios blancos principales se eliminan de la primera línea. Cualquier espacio blanco principal que pueda ser uniformemente removido de la segunda línea en adelante es removido. Las líneas vacías al principio y al final se eliminan posteriormente. Además, todas las pestañas se expanden a los espacios.

Introspección de los invocables con el objeto Signature

Added in version 3.3.

The **`Signature`** object represents the call signature of a callable object and its return annotation. To retrieve a **`Signature`** object, use the **`signature()`** function.

inspect.signature(*callable*, **, *follow_wrapped=True*, *globals=None*, *locals=None*, *eval_str=False*, *annotation_format=Format.VALUE*)

Return a **`Signature`** object for the given *callable*:

```
>>> from inspect import signature
>>> def foo(a, *, b:int, **kwargs):
...     pass

>>> sig = signature(foo)

>>> str(sig)
'(a, *, b: int, **kwargs)'

>>> str(sig.parameters['b'])
'b: int'

>>> sig.parameters['b'].annotation
<class 'int'>
```

Acepta un amplio rango de invocables de Python, desde funciones y clases simples hasta objetos **`functools.partial()`**.

If some of the annotations are strings (e.g., because `from __future__ import annotations` was used), **`signature()`** will attempt to automatically un-stringize the annotations using **`annotationlib.get_annotations()`**. The *globals*, *locals*, and *eval_str* parameters are passed into **`annotationlib.get_annotations()`** when resolving the annotations; see the documentation for **`annotationlib.get_annotations()`** for instructions on how to use these parameters. A member of the **`annotationlib.Format`** enum can be passed to the *annotation_format* parameter to control the format of the returned annotations. For example, use `annotation_format=annotationlib.Format.STRING` to return annotations in string format.

Raises **`ValueError`** if no signature can be provided, and **`TypeError`** if that type of object is not supported. Also, if the annotations are stringized, and *eval_str* is not false, the `eval()` call(s) to un-stringize the annotations in **`annotationlib.get_annotations()`** could potentially raise any kind of exception.

A slash (/) in the signature of a function denotes that the parameters prior to it are positional-only. For more info, see the FAQ entry on positional-only parameters.

Distinto en la versión 3.5: The *follow_wrapped* parameter was added. Pass `False` to get a signature of *callable* specifically (`callable.__wrapped__` will not be used to unwrap decorated callables.)

Distinto en la versión 3.10: The *globals*, *locals*, and *eval_str* parameters were added.

Distinto en la versión 3.14: The *annotation_format* parameter was added.

Nota

Algunos invocables pueden no ser introspeccionables en ciertas implementaciones de Python. Por ejemplo, en

CPython, algunas funciones incorporadas definidas en C no proporcionan metadatos sobre sus argumentos.

Detalles de implementación de CPython: If the passed object has a `__signature__` attribute, we may use it to create the signature. The exact semantics are an implementation detail and are subject to unannounced changes. Consult the source code for current semantics.

class inspect.Signature(*parameters=None*, ***, *return_annotation=Signature.empty*)

A `Signature` object represents the call signature of a function and its return annotation. For each parameter accepted by the function it stores a `Parameter` object in its `parameters` collection.

El argumento opcional *parámetros* es una secuencia de objetos `Parameter`, que se valida para comprobar que no hay parámetros con nombres duplicados, y que los parámetros están en el orden correcto, es decir, primero sólo de posición, luego de posición o palabra clave, y que los parámetros con valores por defecto siguen a los parámetros sin valores por defecto.

The optional *return_annotation* argument can be an arbitrary Python object. It represents the «return» annotation of the callable.

`Signature` objects are *immutable*. Use `Signature.replace()` or `copy.replace()` to make a modified copy.

Distinto en la versión 3.5: `Signature` objects are now picklable and hashable.

empty

Un marcador especial de clase para especificar la ausencia de una anotación de retorno.

parameters

Un mapeo ordenado de los nombres de los parámetros a los correspondientes objetos `Parameter`. Los parámetros aparecen en estricto orden de definición, incluyendo parámetros de sólo palabras clave.

Distinto en la versión 3.7: Python sólo garantizó explícitamente que conservaba el orden de declaración de los parámetros de sólo palabras clave a partir de la versión 3.7, aunque en la práctica este orden siempre se había conservado en Python 3.

return_annotation

La anotación de «retorno» para el invocable. Si el invocable no tiene ninguna anotación de «return», este atributo se establece en `Signature.empty`.

bind(**args, **kwargs*)

Crear un mapeo de argumentos posicionales y de palabras clave a los parámetros. Retorna `BoundArguments` si `*args` y `**kwargs` coinciden con el signature, o lanza un `TypeError`.

bind_partial(**args, **kwargs*)

Funciona de la misma manera que `Signature.bind()`, pero permite la omisión de algunos argumentos requeridos (imita el comportamiento de `functools.partial()`.) Retorna `BoundArguments`, o lanza un `TypeError` si los argumentos pasados no coinciden con la firma.

replace(**[, parameters][, return_annotation]*)

Create a new **`Signature`** instance based on the instance **`replace()`** was invoked on. It is possible to pass different *parameters* and/or *return_annotation* to override the corresponding properties of the base signature. To remove `return_annotation` from the copied **`Signature`**, pass in `Signature.empty`.

```
>>> def test(a, b):
...     pass
...
>>> sig = signature(test)
>>> new_sig = sig.replace(return_annotation="ne
>>> str(new_sig)
"(a, b) -> 'new return anno'"
```

`Signature` objects are also supported by the generic function **`copy.replace()`**.

format(***, *max_width=None*, *quote_annotation_strings=True*)

Create a string representation of the **`Signature`** object.

If *max_width* is passed, the method will attempt to fit the signature into lines of at most *max_width* characters. If the signature is longer than *max_width*, all parameters will be on separate lines.

If *quote_annotation_strings* is False, annotations in the signature are displayed without opening and closing quotation marks if they are strings. This is useful if the signature was created with the **`STRING`** format or if `from __future__ import annotations` was used.

Added in version 3.13.

Distinto en la versión 3.14: The *unquote_annotations* parameter was added.

classmethod from_callable(*obj*, ***, *follow_wrapped=True*, *globals=None*, *locals=None*, *eval_str=False*)

Return a **`Signature`** (or its subclass) object for a given callable *obj*.

This method simplifies subclassing of **`Signature`**:

```
class MySignature(Signature):
    pass
sig = MySignature.from_callable(sum)
assert isinstance(sig, MySignature)
```

Its behavior is otherwise identical to that of **`signature()`**.

Added in version 3.5.

Distinto en la versión 3.10: The *globals*, *locals*, and *eval_str* parameters were added.

class inspect.Parameter(*name*, *kind*, ***, *default=Parameter.empty*, *annotation=Parameter.empty*)

`Parameter` objects are *immutable*. Instead of modifying a **`Parameter`** object, you can use **`Parameter.replace()`** or **`copy.replace()`** to create a modified copy.

Distinto en la versión 3.5: Parameter objects are now picklable and hashable.

empty

Un marcador especial de clase para especificar la ausencia de valores predeterminados y anotaciones.

name

El nombre del parámetro como una cadena. El nombre debe ser un identificador Python válido.

CPython genera nombres de parámetros implícitos de la forma `.0` en los objetos de código utilizados para implementar expresiones de comprensiones y generadores.

Distinto en la versión 3.6: These parameter names are now exposed by this module as names like `implicit0`.

default

El valor por defecto del parámetro. Si el parámetro no tiene un valor por defecto, este atributo se establece en `Parameter.empty`.

annotation

La anotación para el parámetro. Si el parámetro no tiene ninguna anotación, este atributo se establece como `Parameter.empty`.

kind

Describes how argument values are bound to the parameter. The possible values are accessible via **`Parameter`** (like `Parameter.KEYWORD_ONLY`), and support comparison and ordering, in the following order:

Nombre	**Significado**
POSITIONAL_ONLY	El valor debe proporcionarse como un argumento posicional. Los parámetros solo posicionales son aquellos que aparecen antes de una entrada `/` (si está presente) en una definición de función de Python. aceptan sólo uno o dos parámetros) los aceptan.
POSITIONAL_OR_KEYWORD	El valor puede ser suministrado como una palabra clave o como un argumento posicional (este es el comportamiento estándar de unión para las funciones implementadas en Python)
VAR_POSITIONAL	Una tupla de argumentos posicionales que no están ligados a ningún otro parámetro. Esto corresponde a un parámetro `*args` en una definición de función Python.
KEYWORD_ONLY	El valor debe ser suministrado como argumento de la palabra clave. Los parámetros de sólo palabras clave son los que aparecen después de una entrada `*` o `*args` en una definición de función Python.
VAR_KEYWORD	Un dict de argumentos de palabras clave que no están ligadas a ningún otro parámetro. Esto corresponde a un parámetro `**kwargs` en una

definición de función Python.

Example: print all keyword-only arguments without default values:

```
>>> def foo(a, b, *, c, d=10):
...     pass

>>> sig = signature(foo)
>>> for param in sig.parameters.values():
...     if (param.kind == param.KEYWORD_ONLY ar
...                        param.default is par
...         print('Parameter:', param)
Parameter: c
```

kind.description

Describes an enum value of `Parameter.kind`.

Added in version 3.8.

Example: print all descriptions of arguments:

```
>>> def foo(a, b, *, c, d=10):
...     pass

>>> sig = signature(foo)
>>> for param in sig.parameters.values():
...     print(param.kind.description)
positional or keyword
positional or keyword
keyword-only
keyword-only
```

replace(**[, name][, kind][, default][, annotation]*)

Create a new **`Parameter`** instance based on the instance replaced was invoked on. To override a **`Parameter`** attribute, pass the corresponding argument. To remove a default value or/and an annotation from a **`Parameter`**, pass

`Parameter.empty`.

```
>>> from inspect import Parameter
>>> param = Parameter('foo', Parameter.KEYWORD_
>>> str(param)
'foo=42'

>>> str(param.replace()) # Will create a shallo
'foo=42'

>>> str(param.replace(default=Parameter.empty,
"foo: 'spam'"
```

`Parameter` objects are also supported by the generic function `copy.replace()`.

Distinto en la versión 3.4: In Python 3.3 **`Parameter`** objects were allowed to have `name` set to `None` if their `kind` was set to `POSITIONAL_ONLY`. This is no longer permitted.

class inspect.BoundArguments

Resultado de una llamada `Signature.bind()` o `Signature.bind_partial()`. Mantiene el mapeo de los argumentos a los parámetros de la función.

arguments

Un mapeo mutable de los nombres de los parámetros a los valores de los argumentos. Contiene solo argumentos vinculados explícitamente. Los cambios en `arguments` se reflejarán en `args` y `kwargs`.

Debe ser usado en conjunto con `Signature.parameters` para cualquier propósito de procesamiento de argumentos.

Nota

Los argumentos para los cuales `Signature.bind()` o `Signature.bind_partial()` se basaban en un valor por defecto se saltan. Sin embargo, si es necesario, use `BoundArguments.apply_defaults()` para añadirlos.

Distinto en la versión 3.9: `arguments` ahora es de tipo `dict`. Anteriormente, era de tipo `collections.OrderedDict`.

args

Una tupla de valores de argumentos posicionales. Calculados dinámicamente a partir del atributo `arguments`.

kwargs

A dict of keyword arguments values. Dynamically computed from the `arguments` attribute. Arguments that can be passed positionally are included in `args` instead.

signature

Una referencia al objeto padre `Signature`.

apply_defaults()

Establece valores por defecto para los argumentos que faltan.

Para los argumentos de posición variable (`*args`) el valor por defecto es una tupla vacía.

Para los argumentos de palabras clave variables (`**kwargs`) el valor por defecto es un diccionario vacío.

```
>>> def foo(a, b='ham', *args): pass
>>> ba = inspect.signature(foo).bind('spam')
>>> ba.apply_defaults()
>>> ba.arguments
```

```
{'a': 'spam', 'b': 'ham', 'args': ()}
```

Added in version 3.5.

The **`args`** and **`kwargs`** properties can be used to invoke functions:

```
def test(a, *, b):
    ...

sig = signature(test)
ba = sig.bind(10, b=20)
test(*ba.args, **ba.kwargs)
```

Ver también

PEP 362 [https://peps.python.org/pep-0362/] - Función Objeto Signature.
La especificación detallada, los detalles de implementación y los ejemplos.

Clases y funciones

inspect.getclasstree(*classes*, *unique=False*)

Organizar la lista de clases dada en una jerarquía de listas anidadas. Cuando aparece una lista anidada, contiene clases derivadas de la clase cuya entrada precede inmediatamente a la lista. Cada entrada es una tupla de 2 valores que contienen una clase y una tupla de sus clases base. Si el argumento *unique* es cierto, aparece exactamente una entrada en la estructura retornada para cada clase de la lista dada. De lo contrario, las clases que utilizan la herencia múltiple y sus descendientes aparecerán varias veces.

inspect.getfullargspec(*func*)

Obtener los nombres y valores por defecto de los parámetros de una función de Python. Se retorna un named tuple:

`FullArgSpec(args, varargs, varkw, defaults, kwonlyargs, kwonlydefaults, annotations)`

args es una lista de los nombres de los parámetros posicionales. *varargs* es el nombre del parámetro `*` o `None` si no se aceptan argumentos posicionales arbitrarios. *varkw* es el nombre del parámetro `**` o `None` si no se aceptan argumentos de palabras clave arbitrarias. *defaults* es una *n*-tupla de valores de argumentos por defecto que corresponden a los últimos parámetros de posición *n*, o `None` si no hay tales valores por defecto definidos. *kwonlyargs* es una lista de nombres de parámetros de sólo palabras clave en orden de declaración. *kwonlydefaults* es un diccionario que asigna los nombres de los parámetros de *kwonlyargs* a los valores por defecto utilizados si no se suministra ningún argumento. *annotations* es un diccionario que asigna los nombres de los parámetros a las anotaciones. La tecla especial `"return"` se utiliza para informar de la anotación del valor de retorno de la función (si existe).

Observe que **`signature()`** y Objeto Signature proporcionan la API recomendada para la introspección invocable, y soportan comportamientos adicionales (como los argumentos de sólo posición) que a veces se encuentran en las API de los módulos de extensión. Esta función se conserva principalmente para su uso en el código que necesita mantener la compatibilidad con la API de módulos de `inspect` de Python 2.

Distinto en la versión 3.4: Esta función se basa ahora en **`signature()`**, pero sigue ignorando los atributos `__wrapped__` e incluye el primer parámetro ya ligado en la salida del signature para los métodos ligados.

Distinto en la versión 3.6: This method was previously documented as deprecated in favour of **`signature()`** in Python 3.5, but that decision has been reversed in order to restore a clearly supported standard interface for single-source Python 2/3 code migrating away from the legacy **`getargspec()`** API.

Distinto en la versión 3.7: Python sólo garantizó explícitamente que conservaba el orden de declaración de los parámetros de sólo palabras clave a partir de la versión 3.7, aunque en la práctica este orden siempre se había conservado en Python 3.

inspect.getargvalues(*frame*)

Obtener información sobre los argumentos pasados en un marco particular. Un named tuple `ArgInfo(args, varargs, keywords, locals)` es retornado. *args* es una lista de los nombres de los argumentos. *varargs* y *keywords* son los nombres de los argumentos `*` y `**` o `None`. *locals* es el diccionario local del marco dado.

Nota

Esta función fue inadvertidamente marcada como obsoleta en Python 3.5.

inspect.formatargvalues(*args*[, *varargs*, *varkw*, *locals*, *formatarg*, *formatvarargs*, *formatvarkw*, *formatvalue*])

Formatea una bonita especificación de argumentos de los cuatro valores retornados por **`getargvalues()`**. Los argumentos de formato* son las correspondientes funciones de formato opcionales que se llaman para convertir nombres y valores en cadenas.

Nota

Esta función fue inadvertidamente marcada como obsoleta en Python 3.5.

inspect.getmro(*cls*)

Retorna una tupla de clases base de cls, incluyendo cls, en orden de resolución de métodos. Ninguna clase aparece más de una vez en esta tupla. Obsérvese que el orden de resolución de los métodos depende del tipo de cls. A menos que se utilice un meta tipo muy peculiar definido por el usuario, cls será el primer elemento de la tupla.

inspect.getcallargs(*func*, /, **args*, ***kwds*)

Bind the *args* and *kwds* to the argument names of the Python function or method *func*, as if it was called with them. For bound methods, bind also the first argument (typically named `self`) to the associated instance. A dict is returned, mapping the argument names (including the names of the `*` and `**` arguments, if any) to their values from *args* and *kwds*. In case of invoking *func* incorrectly, i.e. whenever `func(*args, **kwds)` would raise an exception because of incompatible signature, an exception of the same type and the same or similar message is raised. For example:

```
>>> from inspect import getcallargs
>>> def f(a, b=1, *pos, **named):
...     pass
...
>>> getcallargs(f, 1, 2, 3) == {'a': 1, 'named': {}
True
>>> getcallargs(f, a=2, x=4) == {'a': 2, 'named': {
True
>>> getcallargs(f)
Traceback (most recent call last):
...
TypeError: f() missing 1 required positional argume
```

Added in version 3.2.

Obsoleto desde la versión 3.5: Usa **`Signature.bind()`** y **`Signature.bind_partial()`** en su lugar.

inspect.getclosurevars(*func*)

Obtener el mapeo de referencias de nombres externos en una función o método *func* de Python a sus valores actuales. Un named tuple `ClosureVars(nonlocals, globals, builtins, unbound)` es retornado. *nonlocals* asigna los nombres referidos a las variables de cierre léxicas, *globals* a los globals de los módulos de la función y *builtins* a los builtins visibles desde el cuerpo de la función. *unbound* es el conjunto de nombres referenciados en la función que no pudieron ser resueltos en absoluto dados los actuales globals y builtins del módulo.

`TypeError` es lanzado si *func* no es una función o método de Python.

Added in version 3.3.

inspect.unwrap(*func*, ***, *stop=None*)

Obtiene el objeto envuelto por *func*. Sigue la cadena de atributos **`__wrapped__`** retornando el último objeto de la cadena.

stop es una retrollamada opcional que acepta un objeto de la cadena envuelta como único argumento que permite terminar el desenvolvimiento antes de tiempo si la retrollamada retorna un valor real. Si la retrollamada nunca retorna un valor verdadero, el último objeto de la cadena se retorna como de costumbre. Por ejemplo, **`signature()`** utiliza esto para detener el desenvolvimiento si algún objeto de la cadena tiene definido el atributo `__signature__`.

`ValueError` es lanzado si se encuentra un ciclo.

Added in version 3.4.

inspect.get_annotations(*obj*, ***, *globals=None*, *locals=None*, *eval_str=False*, *format=annotationlib.Format.VALUE*)

Calcular el diccionario de anotaciones de un objeto.

This is an alias for **`annotationlib.get_annotations()`**; see the documentation of that function for more information.

Prudencia

This function may execute arbitrary code contained in annotations. See Security implications of introspecting annotations for more information.

Added in version 3.10.

Distinto en la versión 3.14: This function is now an alias for **`annotationlib.get_annotations()`**. Calling it as `inspect.get_annotations` will continue to work.

La pila del interprete

Algunas de las siguientes funciones retornan objetos **`FrameInfo`**. Por compatibilidad con versiones anteriores, estos objetos permiten operaciones similares a tuplas en todos los atributos excepto `positions`. Este comportamiento se considera obsoleto y puede eliminarse en el futuro.

class inspect.FrameInfo

frame

El frame object al que corresponde el registro.

filename

El nombre del archivo asociado con el código que está ejecutando el marco al que corresponde este registro.

lineno

El número de línea de la línea actual asociada con el código que está ejecutando el marco al que corresponde este registro.

function

El nombre de la función que está ejecutando el marco al que corresponde este registro.

code_context

Una lista de líneas de contexto del código fuente que ejecuta el marco al que corresponde este registro.

index

El índice de la línea actual que se está ejecutando en la lista `code_context`.

positions

Un objeto `dis.Positions` que contiene el número de línea inicial, el número de línea final, el desplazamiento de columna inicial y el desplazamiento de columna final asociado con la instrucción que ejecuta el marco al que corresponde este registro.

Distinto en la versión 3.5: Retorna un named tuple en lugar de un `tuple`.

Distinto en la versión 3.11: `FrameInfo` ahora es una instancia de clase (que es retrocompatible con el anterior named tuple).

class inspect.Traceback

filename

El nombre de archivo asociado con el código que ejecuta el marco al que corresponde este rastreo.

lineno

El número de línea de la línea actual asociada con el código que está ejecutando el marco al que corresponde este rastreo.

function

El nombre de la función que está ejecutando el marco al que corresponde este rastreo.

code_context

Una lista de líneas de contexto del código fuente que ejecuta el marco al que corresponde este rastreo.

index

El índice de la línea actual que se está ejecutando en la lista `code_context`.

positions

Un objeto `dis.Positions` que contiene el número de línea inicial, el número de línea final, el desplazamiento de columna inicial y el desplazamiento de columna final asociado con la instrucción que ejecuta el marco al que corresponde este rastreo.

Distinto en la versión 3.11: **`Traceback`** ahora es una instancia de clase (que es retrocompatible con el anterior named tuple).

Nota

Mantener referencias a los objetos marco, como se encuentra en el primer elemento de los registros marco que estas funciones retornan, puede hacer que su programa cree ciclos de referencia. Una vez creado un ciclo de referencia, la vida útil de todos los objetos a los que se puede acceder desde los objetos que forman el ciclo puede ser mucho mayor, incluso si el detector de ciclos opcional de Python está activado. Si es necesario crear tales ciclos, es importante asegurarse de que se rompen explícitamente para evitar la destrucción retardada de los objetos y el aumento del consumo de memoria que se produce.

Aunque el detector de ciclos los captará, la destrucción de los marcos (y las variables locales) puede hacerse determinísticamente eliminando el ciclo en una cláusula de `finally`. Esto también es importante si el detector de ciclos fue desactivado cuando se compiló Python o usando `gc.disable()`. Por ejemplo:

```
def handle_stackframe_without_leak():
    frame = inspect.currentframe()
    try:
        # do something with the frame
    finally:
        del frame
```

Si quieres mantener el marco alrededor (por ejemplo para imprimir una traceback más tarde), también puedes romper los ciclos de referencia utilizando el método `frame.clear()`.

El argumento opcional de *context*, apoyado por la mayoría de estas funciones, especifica el número de líneas de contexto a retornar, que se centran en la línea actual.

inspect.getframeinfo(*frame*, *context=1*)

Obtiene información sobre un marco o un objeto de rastreo. Se retorna un objeto `Traceback`.

Distinto en la versión 3.11: Se retorna un objeto `Traceback` en lugar de una tupla con nombre.

inspect.getouterframes(*frame*, *context=1*)

Obtiene una lista de objetos `FrameInfo` para un marco y todos los marcos externos. Estos marcos representan las llamadas que conducen a la creación de *frame*. La primera entrada en la lista retornada representa *frame*; la última entrada representa la llamada más externa en la pila de *frame*.

Distinto en la versión 3.5: Una lista de named tuples `FrameInfo(frame, filename, lineno, function, code_context, index)` es retornada.

Distinto en la versión 3.11: Se retorna una lista de objetos `FrameInfo`.

inspect.getinnerframes(*traceback*, *context=1*)

Obtiene una lista de objetos `FrameInfo` para el marco de un rastreo y todos los marcos internos. Estos marcos representan llamadas realizadas como consecuencia de *frame*. La primera entrada de la lista representa *traceback*; la última entrada representa dónde se generó la excepción.

Distinto en la versión 3.5: Una lista de named tuples `FrameInfo(frame, filename, lineno, function, code_context, index)` es retornada.

Distinto en la versión 3.11: Se retorna una lista de objetos **`FrameInfo`**.

inspect.currentframe()

Retorna el objeto marco para el marco de la pila del que llama.

Esta función se basa en el soporte del marco de la pila de Python en el intérprete, que no está garantizado que exista en todas las implementaciones de Python. Si se ejecuta en una implementación sin soporte de marcos de pila de Python, esta función retorna `None`.

inspect.stack(*context=1*)

Retorna una lista de objetos **`FrameInfo`** para la pila del que llama. La primera entrada en la lista retornada representa al que llama; la última entrada representa la llamada más externa de la pila.

Distinto en la versión 3.5: Una lista de named tuples `FrameInfo(frame, filename, lineno, function, code_context, index)` es retornada.

Distinto en la versión 3.11: Se retorna una lista de objetos **`FrameInfo`**.

inspect.trace(*context=1*)

Retorna una lista de objetos **`FrameInfo`** para la pila entre el marco actual y el marco en el que se generó una excepción que se está manejando actualmente. La primera entrada en la lista representa al que llama; la última entrada representa dónde se generó la excepción.

Distinto en la versión 3.5: Una lista de named tuples `FrameInfo(frame, filename, lineno, function, code_context, index)` es retornada.

Distinto en la versión 3.11: Se retorna una lista de objetos **`FrameInfo`**.

Obteniendo atributos estáticamente

Both `getattr()` and `hasattr()` can trigger code execution when fetching or checking for the existence of attributes. Descriptors, like properties, will be invoked and `__getattr__()` and `__getattribute__()` may be called.

Para los casos en los que se quiera una introspección pasiva, como las herramientas de documentación, esto puede ser un inconveniente. `getattr_static()` tiene la misma firma que `getattr()` pero evita la ejecución de código cuando obtiene atributos.

inspect.getattr_static(*obj*, *attr*, *default*=*None*)

> Retrieve attributes without triggering dynamic lookup via the descriptor protocol, `__getattr__()` or `__getattribute__()`.
>
> Nota: es posible que esta función no pueda recuperar todos los atributos que getattr puede recuperar (como los atributos creados dinámicamente) y puede encontrar atributos que getattr no puede (como los descriptores que lanzan AttributeError). También puede retornar objetos descriptores en lugar de miembros de la instancia.
>
> Si la instancia `__dict__` es ensombrecida por otro miembro (por ejemplo una propiedad) entonces esta función no podrá encontrar miembros de la instancia.
>
> *Added in version 3.2.*

`getattr_static()` no resuelve los descriptores, por ejemplo los descriptores de ranura o los descriptores de getset en los objetos implementados en C. El objeto descriptor se retorna en lugar del atributo subyacente.

Puedes manejar esto con un código como el siguiente. Tenga en cuenta que la invocación de los descriptores de getset arbitrarios pueden desencadenar la ejecución del código:

```
# example code for resolving the builtin descriptor type
class _foo:
    __slots__ = ['foo']

slot_descriptor = type(_foo.foo)
getset_descriptor = type(type(open(__file__)).name)
wrapper_descriptor = type(str.__dict__['__add__'])
descriptor_types = (slot_descriptor, getset_descriptor,

result = getattr_static(some_object, 'foo')
if type(result) in descriptor_types:
    try:
        result = result.__get__()
    except AttributeError:
        # descriptors can raise AttributeError to
        # indicate there is no underlying value
        # in which case the descriptor itself will
        # have to do
        pass
```

Current State of Generators, Coroutines, and Asynchronous Generators

Al implementar los programadores de corutinas y para otros usos avanzados de los generadores, es útil determinar si un generador se está ejecutando actualmente, si está esperando para iniciarse o reanudarse o si ya ha terminado. **`getgeneratorstate()`** permite determinar fácilmente el estado actual de un generador.

inspect.getgeneratorstate(*generator*)

Obtener el estado actual de un generador-iterador.

Los posibles estados son:

- GEN_CREATED: Esperando para iniciar la ejecución.
- GEN_RUNNING: Actualmente está siendo ejecutado por el intérprete.
- GEN_SUSPENDED: Actualmente suspendido en una

expresión yield.
- GEN_CLOSED: La ejecución se ha completado.

Added in version 3.2.

inspect.getcoroutinestate(*coroutine*)

Obtener el estado actual de un objeto de corutina. La función está pensada para ser usada con objetos de corutina creados por las funciones **`async def`**, pero aceptará cualquier objeto de corutina que tenga los atributos `cr_running` y `cr_frame`.

Los posibles estados son:

- CORO_CREATED: Esperando para iniciar la ejecución.
- CORO_RUNNING: Actualmente está siendo ejecutado por el intérprete.
- CORO_SUSPENDED: Actualmente suspendido en una expresión de espera.
- CORO_CLOSED: La ejecución se ha completado.

Added in version 3.5.

inspect.getasyncgenstate(*agen*)

Get current state of an asynchronous generator object. The function is intended to be used with asynchronous iterator objects created by **`async def`** functions which use the **`yield`** statement, but will accept any asynchronous generator-like object that has `ag_running` and `ag_frame` attributes.

Los posibles estados son:

- AGEN_CREATED: Waiting to start execution.
- AGEN_RUNNING: Currently being executed by the interpreter.
- AGEN_SUSPENDED: Currently suspended at a yield expression.
- AGEN_CLOSED: Execution has completed.

Added in version 3.12.

También se puede consultar el estado interno actual del generador. Esto es mayormente útil para fines de prueba, para asegurar que el estado interno se actualiza como se espera:

inspect.getgeneratorlocals(*generator*)

Consigue el mapeo de las variables vivas locales en *generator* a sus valores actuales. Se retorna un diccionario que mapea de los nombres de las variables a los valores. Esto es el equivalente a llamar `locals()` en el cuerpo del generador, y se aplican todas las mismas advertencias.

Si *generator* es un generator sin marco asociado actualmente, entonces se retorna un diccionario vacío. `TypeError` es lanzado si *generator* no es un objeto generador de Python.

Esta función se basa en que el generador exponga un marco de pila de Python para la introspección, lo cual no está garantizado en todas las implementaciones de Python. En tales casos, esta función siempre retornará un diccionario vacío.

Added in version 3.3.

inspect.getcoroutinelocals(*coroutine*)

Esta función es análoga a `getgeneratorlocals()`, pero funciona para los objetos de corutina creados por funciones `async def`.

Added in version 3.5.

inspect.getasyncgenlocals(*agen*)

This function is analogous to `getgeneratorlocals()`, but works for asynchronous generator objects created by `async def` functions which use the `yield` statement.

Added in version 3.12.

Objetos de código Bit Flags

Python code objects have a `co_flags` attribute, which is a bitmap of the following flags:

inspect.CO_OPTIMIZED

El objeto del código está optimizado, usando locales rápidas (*fast locals*).

inspect.CO_NEWLOCALS

If set, a new dict will be created for the frame's `f_locals` when the code object is executed.

inspect.CO_VARARGS

El objeto del código tiene un parámetro posicional variable (similar a `*args`).

inspect.CO_VARKEYWORDS

El objeto del código tiene un parámetro de palabra clave variable (similar a `**kwargs`).

inspect.CO_NESTED

El flag se fija cuando el objeto del código es una función anidada.

inspect.CO_GENERATOR

El flag se fija cuando el objeto del código es una función generadora, es decir, un objeto generador es retornado cuando el objeto del código se ejecuta.

inspect.CO_COROUTINE

El flag se configura cuando el objeto del código es una función de corutina. Cuando el objeto código se ejecuta, retorna un objeto de corutina. Ver **PEP 492** [https://peps.python.org/pep-0492/] para más detalles.

Added in version 3.5.

inspect.CO_ITERABLE_COROUTINE

El flag se utiliza para transformar generadores en corutinas basadas en generadores. Los objetos generadores con este flag pueden ser usados en la expresión `await`, y objetos de corutina `yield from`. Ver **PEP 492** [https://peps.python.org/pep-0492/] para más detalles.

Added in version 3.5.

inspect.CO_ASYNC_GENERATOR

El flag se configura cuando el objeto del código es una función generadora asíncrona. Cuando el objeto código se ejecuta, retorna un objeto generador asíncrono. Ver **PEP 525** [https://peps.python.org/pep-0525/] para más detalles.

Added in version 3.6.

inspect.CO_HAS_DOCSTRING

The flag is set when there is a docstring for the code object in the source code. If set, it will be the first item in **`co_consts`**.

Added in version 3.14.

inspect.CO_METHOD

The flag is set when the code object is a function defined in class scope.

Added in version 3.14.

Nota

The flags are specific to CPython, and may not be defined in other Python implementations. Furthermore, the flags are an implementation detail, and can be removed or deprecated in future Python releases. It’s recommended to use public APIs from the **`inspect`** module for any introspection needs.

Buffer flags

class inspect.BufferFlags

This is an `enum.IntFlag` that represents the flags that can be passed to the `__buffer__()` method of objects implementing the buffer protocol.

The meaning of the flags is explained at Tipos de solicitud búfer.

SIMPLE

WRITABLE

FORMAT

ND

STRIDES

C_CONTIGUOUS

F_CONTIGUOUS

ANY_CONTIGUOUS

INDIRECT

CONTIG

CONTIG_RO

STRIDED

STRIDED_RO

RECORDS

RECORDS_RO

FULL

FULL_RO

READ

WRITE

Added in version 3.12.

Command-line interface

The `inspect` module also provides a basic introspection capability from the command line.

Por defecto, acepta el nombre de un módulo e imprime la fuente de ese módulo. Una clase o función dentro del módulo puede imprimirse en su lugar añadiendo dos puntos y el nombre calificado del objeto objetivo.

--details

Imprimir información sobre el objeto especificado en lugar del código fuente

`annotationlib` — Functionality for introspecting annotations

Added in version 3.14.

Source code: Lib/annotationlib.py [https://github.com/python/cpython/tree/3.14/Lib/annotationlib.py]

The **`annotationlib`** module provides tools for introspecting annotations on modules, classes, and functions.

Annotations are lazily evaluated and often contain forward references to objects that are not yet defined when the annotation is created. This module provides a set of low-level tools that can be used to retrieve annotations in a reliable way, even in the presence of forward references and other edge cases.

This module supports retrieving annotations in three main formats (see **`Format`**), each of which works best for different use cases:

- **`VALUE`** evaluates the annotations and returns their value. This is most straightforward to work with, but it may raise errors, for example if the annotations contain references to undefined names.
- **`FORWARDREF`** returns **`ForwardRef`** objects for annotations that cannot be resolved, allowing you to inspect the annotations without evaluating them. This is useful when you need to work with annotations that may contain unresolved forward references.
- **`STRING`** returns the annotations as a string, similar to how it would appear in the source file. This is useful for documentation generators that want to display annotations in a readable way.

The `get_annotations()` function is the main entry point for retrieving annotations. Given a function, class, or module, it returns an annotations dictionary in the requested format. This module also provides functionality for working directly with the annotate function that is used to evaluate annotations, such as `get_annotate_from_class_namespace()` and `call_annotate_function()`, as well as the `call_evaluate_function()` function for working with evaluate functions.

Prudencia

Most functionality in this module can execute arbitrary code; see the security section for more information.

Ver también

PEP 649 [https://peps.python.org/pep-0649/] proposed the current model for how annotations work in Python.

PEP 749 [https://peps.python.org/pep-0749/] expanded on various aspects of **PEP 649** [https://peps.python.org/pep-0649/] and introduced the `annotationlib` module.

Prácticas recomendadas para las anotaciones provides best practices for working with annotations.

typing-extensions [https://pypi.org/project/typing-extensions/] provides a backport of `get_annotations()` that works on earlier versions of Python.

Annotation semantics

The way annotations are evaluated has changed over the history of Python 3, and currently still depends on a future import. There have been execution models for annotations:

- *Stock semantics* (default in Python 3.0 through 3.13; see **PEP 3107** [https://peps.python.org/pep-3107/] and **PEP 526** [https://

peps.python.org/pep-0526/]): Annotations are evaluated eagerly, as they are encountered in the source code.

- *Stringified annotations* (used with `from __future__ import annotations` in Python 3.7 and newer; see **PEP 563** [https://peps.python.org/pep-0563/]): Annotations are stored as strings only.
- *Deferred evaluation* (default in Python 3.14 and newer; see **PEP 649** [https://peps.python.org/pep-0649/] and **PEP 749** [https://peps.python.org/pep-0749/]): Annotations are evaluated lazily, only when they are accessed.

As an example, consider the following program:

```
def func(a: Cls) -> None:
    print(a)

class Cls: pass

print(func.__annotations__)
```

This will behave as follows:

- Under stock semantics (Python 3.13 and earlier), it will throw a **`NameError`** at the line where `func` is defined, because `Cls` is an undefined name at that point.
- Under stringified annotations (if `from __future__ import annotations` is used), it will print `{'a': 'Cls', 'return': 'None'}`.
- Under deferred evaluation (Python 3.14 and later), it will print `{'a': <class 'Cls'>, 'return': None}`.

Stock semantics were used when function annotations were first introduced in Python 3.0 (by **PEP 3107** [https://peps.python.org/pep-3107/]) because this was the simplest, most obvious way to implement annotations. The same execution model was used when variable annotations were introduced in Python 3.6 (by **PEP 526** [https://peps.python.org/pep-0526/]). However, stock semantics caused problems when using annotations as type hints, such as a need to refer to names that are not yet defined when the annotation is encountered. In addition, there were performance problems with executing annotations at module import time. Therefore, in Python

3.7, **PEP 563** [https://peps.python.org/pep-0563/] introduced the ability to store annotations as strings using the `from __future__ import annotations` syntax. The plan at the time was to eventually make this behavior the default, but a problem appeared: stringified annotations are more difficult to process for those who introspect annotations at runtime. An alternative proposal, **PEP 649** [https://peps.python.org/pep-0649/], introduced the third execution model, deferred evaluation, and was implemented in Python 3.14. Stringified annotations are still used if `from __future__ import annotations` is present, but this behavior will eventually be removed.

Classes

class annotationlib.Format

An `IntEnum` describing the formats in which annotations can be returned. Members of the enum, or their equivalent integer values, can be passed to `get_annotations()` and other functions in this module, as well as to `__annotate__` functions.

VALUE = *1*

Values are the result of evaluating the annotation expressions.

VALUE_WITH_FAKE_GLOBALS = *2*

Special value used to signal that an annotate function is being evaluated in a special environment with fake globals. When passed this value, annotate functions should either return the same value as for the `Format.VALUE` format, or raise `NotImplementedError` to signal that they do not support execution in this environment. This format is only used internally and should not be passed to the functions in this module.

FORWARDREF = *3*

Values are real annotation values (as per

`Format.VALUE` format) for defined values, and `ForwardRef` proxies for undefined values. Real objects may contain references to `ForwardRef` proxy objects.

STRING = *4*

Values are the text string of the annotation as it appears in the source code, up to modifications including, but not restricted to, whitespace normalizations and constant values optimizations.

The exact values of these strings may change in future versions of Python.

Added in version 3.14.

class annotationlib.ForwardRef

A proxy object for forward references in annotations.

Instances of this class are returned when the `FORWARDREF` format is used and annotations contain a name that cannot be resolved. This can happen when a forward reference is used in an annotation, such as when a class is referenced before it is defined.

__forward_arg__

A string containing the code that was evaluated to produce the `ForwardRef`. The string may not be exactly equivalent to the original source.

evaluate(**, owner=None, globals=None, locals=None, type_params=None, format=Format.VALUE*)

Evaluate the forward reference, returning its value.

If the *format* argument is `VALUE` (the default), this method may throw an exception, such as `NameError`, if the forward reference refers to a name that cannot be resolved. The arguments to this method can be used to provide bindings for names that would otherwise be undefined. If the *format* argument is `FORWARDREF`, the method will never throw an exception, but may return a **`ForwardRef`** instance. For example, if the forward reference object contains the code `list[undefined]`, where `undefined` is a name that is not defined, evaluating it with the **`FORWARDREF`** format will return `list[ForwardRef('undefined')]`. If the *format* argument is **`STRING`**, the method will return **`__forward_arg__`**.

The *owner* parameter provides the preferred mechanism for passing scope information to this method. The owner of a **`ForwardRef`** is the object that contains the annotation from which the **`ForwardRef`** derives, such as a module object, type object, or function object.

The *globals*, *locals*, and *type_params* parameters provide a more precise mechanism for influencing the names that are available when the **`ForwardRef`** is evaluated. *globals* and *locals* are passed to **`eval()`**, representing the global and local namespaces in which the name is evaluated. The *type_params* parameter is relevant for objects created using the native syntax for generic classes and functions. It is a tuple of type parameters that are in scope while the forward reference is being evaluated. For example, if evaluating a **`ForwardRef`** retrieved from an annotation found in the class namespace of a generic class `C`, *type_params* should be set to `C.__type_params__`.

`ForwardRef` instances returned by **`get_annotations()`** retain references to information about the scope they originated from, so calling this method with no further arguments may be sufficient to evaluate such objects. **`ForwardRef`** instances created by other means may not have any information about their scope, so passing arguments to this method may be necessary to evaluate them successfully.

If no *owner*, *globals*, *locals*, or *type_params* are provided and the **`ForwardRef`** does not contain information about its origin, empty globals and locals dictionaries are used.

Added in version 3.14.

Functions

annotationlib.annotations_to_string(*annotations*)

Convert an annotations dict containing runtime values to a dict containing only strings. If the values are not already strings, they are converted using `type_repr()`. This is meant as a helper for user-provided annotate functions that support the `STRING` format but do not have access to the code creating the annotations.

For example, this is used to implement the `STRING` for `typing.TypedDict` classes created through the functional syntax:

```
>>> from typing import TypedDict
>>> Movie = TypedDict("movie", {"name": str, "year"
>>> get_annotations(Movie, format=Format.STRING)
{'name': 'str', 'year': 'int'}
```

Added in version 3.14.

annotationlib.call_annotate_function(*annotate*, *format*, ***, *owner=None*)

Call the annotate function *annotate* with the given *format*, a member of the `Format` enum, and return the annotations dictionary produced by the function.

This helper function is required because annotate functions generated by the compiler for functions, classes, and modules only support the `VALUE` format when called directly. To support other formats, this function calls the annotate function in a special environment that allows it to produce annotations in the other formats. This is a useful building block when implementing functionality that needs to partially evaluate annotations while a class is being constructed.

owner is the object that owns the annotation function, usually a function, class, or module. If provided, it is used in the `FORWARDREF` format to produce a `ForwardRef` object that carries more information.

Ver también

PEP 649 [https://peps.python.org/pep-0649/#the-stringizer-and-the-fake-globals-environment] contains an explanation of the implementation technique used by this function.

Added in version 3.14.

annotationlib.call_evaluate_function(*evaluate*, *format*, *, *owner=None*)

Call the evaluate function *evaluate* with the given *format*, a member of the `Format` enum, and return the value produced by the function. This is similar to `call_annotate_function()`, but the latter always returns a dictionary mapping strings to annotations, while this function returns a single value.

This is intended for use with the evaluate functions generated for lazily evaluated elements related to type aliases and type parameters:

- `typing.TypeAliasType.evaluate_value()`, the value of type aliases
- `typing.TypeVar.evaluate_bound()`, the bound of type variables
- `typing.TypeVar.evaluate_constraints()`, the constraints of type variables
- `typing.TypeVar.evaluate_default()`, the default value of type variables
- `typing.ParamSpec.evaluate_default()`, the default value of parameter specifications
- `typing.TypeVarTuple.evaluate_default()`, the

default value of type variable tuples

owner is the object that owns the evaluate function, such as the type alias or type variable object.

format can be used to control the format in which the value is returned:

```
>>> type Alias = undefined
>>> call_evaluate_function(Alias.evaluate_value, Fo
Traceback (most recent call last):
...
NameError: name 'undefined' is not defined
>>> call_evaluate_function(Alias.evaluate_value, Fo
ForwardRef('undefined')
>>> call_evaluate_function(Alias.evaluate_value, Fo
'undefined'
```

Added in version 3.14.

annotationlib.get_annotate_from_class_namespace(*namespace*)

Retrieve the annotate function from a class namespace dictionary *namespace*. Return `None` if the namespace does not contain an annotate function. This is primarily useful before the class has been fully created (e.g., in a metaclass); after the class exists, the annotate function can be retrieved with `cls.__annotate__`. See below for an example using this function in a metaclass.

Added in version 3.14.

annotationlib.get_annotations(*obj*, ***, *globals=None*, *locals=None*, *eval_str=False*, *format=Format.VALUE*)

Compute the annotations dict for an object.

obj may be a callable, class, module, or other object with `__annotate__` or `__annotations__` attributes. Passing any other object raises `TypeError`.

The *format* parameter controls the format in which annotations are returned, and must be a member of the `Format` enum or its integer equivalent. The different formats work as follows:

- VALUE: `object.__annotations__` is tried first; if that does not exist, the `object.__annotate__` function is called if it exists.
- FORWARDREF: If `object.__annotations__` exists and can be evaluated successfully, it is used; otherwise, the `object.__annotate__` function is called. If it does not exist either, `object.__annotations__` is tried again and any error from accessing it is re-raised.
  - When calling `object.__annotate__` it is first called with `FORWARDREF`. If this is not implemented, it will then check if `VALUE_WITH_FAKE_GLOBALS` is supported and use that in the fake globals environment. If neither of these formats are supported, it will fall back to using `VALUE`. If `VALUE` fails, the error from this call will be raised.
- STRING: If `object.__annotate__` exists, it is called first; otherwise, `object.__annotations__` is used and stringified using `annotations_to_string()`.
  - When calling `object.__annotate__` it is first called with `STRING`. If this is not implemented, it will then check if `VALUE_WITH_FAKE_GLOBALS` is supported and use that in the fake globals environment. If neither of these formats are supported, it will fall back to using `VALUE` with the result converted using `annotations_to_string()`. If `VALUE` fails, the error from this call will be raised.

Returns a dict. `get_annotations()` returns a new dict every time it's called; calling it twice on the same object will return two different but equivalent dicts.

This function handles several details for you:

- If *eval_str* is true, values of type `str` will be un-

stringized using `eval()`. This is intended for use with stringized annotations (`from __future__ import annotations`). It is an error to set *eval_str* to true with formats other than `Format.VALUE`.
- If *obj* doesn't have an annotations dict, returns an empty dict. (Functions and methods always have an annotations dict; classes, modules, and other types of callables may not.)
- Ignores inherited annotations on classes, as well as annotations on metaclasses. If a class doesn't have its own annotations dict, returns an empty dict.
- All accesses to object members and dict values are done using `getattr()` and `dict.get()` for safety.

eval_str controls whether or not values of type `str` are replaced with the result of calling `eval()` on those values:

- If eval_str is true, `eval()` is called on values of type `str`. (Note that `get_annotations()` doesn't catch exceptions; if `eval()` raises an exception, it will unwind the stack past the `get_annotations()` call.)
- If *eval_str* is false (the default), values of type `str` are unchanged.

globals and *locals* are passed in to `eval()`; see the documentation for `eval()` for more information. If *globals* or *locals* is `None`, this function may replace that value with a context-specific default, contingent on `type(obj)`:

- If *obj* is a module, *globals* defaults to `obj.__dict__`.
- If *obj* is a class, *globals* defaults to `sys.modules[obj.__module__].__dict__` and *locals* defaults to the *obj* class namespace.
- If *obj* is a callable, *globals* defaults to `obj.__globals__`, although if *obj* is a wrapped function (using `functools.update_wrapper()`) or a `functools.partial` object, it is unwrapped until a non-wrapped function is found.

Calling `get_annotations()` is best practice for accessing the annotations dict of any object. See Prácticas recomendadas para las anotaciones for more information on annotations best practices.

```
>>> def f(a: int, b: str) -> float:
...     pass
>>> get_annotations(f)
{'a': <class 'int'>, 'b': <class 'str'>, 'return':
```

Added in version 3.14.

annotationlib.type_repr(*value*)

Convert an arbitrary Python value to a format suitable for use by the **`STRING`** format. This calls **`repr()`** for most objects, but has special handling for some objects, such as type objects.

This is meant as a helper for user-provided annotate functions that support the **`STRING`** format but do not have access to the code creating the annotations. It can also be used to provide a user-friendly string representation for other objects that contain values that are commonly encountered in annotations.

Added in version 3.14.

Recipes

Using annotations in a metaclass

A metaclass may want to inspect or even modify the annotations in a class body during class creation. Doing so requires retrieving annotations from the class namespace dictionary. For classes created with `from __future__ import annotations`, the annotations will be in the `__annotations__` key of the dictionary. For other classes with annotations, **`get_annotate_from_class_namespace()`** can be used to get the annotate function, and **`call_annotate_function()`** can be used to call it and retrieve the annotations. Using the **`FORWARDREF`** format will usually be best, because this allows the annotations to refer to names that cannot yet be resolved when the class is created.

To modify the annotations, it is best to create a wrapper annotate function that calls the original annotate function, makes any necessary adjustments, and returns the result.

Below is an example of a metaclass that filters out all **`typing.ClassVar`** annotations from the class and puts them in a separate attribute:

```
import annotationlib
import typing

class ClassVarSeparator(type):
   def __new__(mcls, name, bases, ns):
      if "__annotations__" in ns:  # from __future__ imp
         annotations = ns["__annotations__"]
         classvar_keys = {
            key for key, value in annotations.items()
            # Use string comparison for simplicity; a mo
            # could use annotationlib.ForwardRef.evaluat
            if value.startswith("ClassVar")
         }
         classvars = {key: annotations[key] for key in c
         ns["__annotations__"] = {
            key: value for key, value in annotations.ite
            if key not in classvar_keys
         }
         wrapped_annotate = None
      elif annotate := annotationlib.get_annotate_from_c
         annotations = annotationlib.call_annotate_funct
            annotate, format=annotationlib.Format.FORWAR
         )
         classvar_keys = {
            key for key, value in annotations.items()
            if typing.get_origin(value) is typing.ClassV
         }
         classvars = {key: annotations[key] for key in c
```

```
        def wrapped_annotate(format):
            annos = annotationlib.call_annotate_functior
            return {key: value for key, value in annos.i

    else:  # no annotations
        classvars = {}
        wrapped_annotate = None
    typ = super().__new__(mcls, name, bases, ns)

    if wrapped_annotate is not None:
        # Wrap the original __annotate__ with a wrapper
        typ.__annotate__ = wrapped_annotate
    typ.classvars = classvars  # Store the ClassVars i
    return typ
```

Limitations of the `STRING` format

The `STRING` format is meant to approximate the source code of the annotation, but the implementation strategy used means that it is not always possible to recover the exact source code.

First, the stringifier of course cannot recover any information that is not present in the compiled code, including comments, whitespace, parenthesization, and operations that get simplified by the compiler.

Second, the stringifier can intercept almost all operations that involve names looked up in some scope, but it cannot intercept operations that operate fully on constants. As a corollary, this also means it is not safe to request the `STRING` format on untrusted code: Python is powerful enough that it is possible to achieve arbitrary code execution even with no access to any globals or builtins. For example:

```
>>> def f(x: (1).__class__.__base__.__subclasses__()[-1]
...
>>> annotationlib.get_annotations(f, format=annotationli
Hello world
{'x': 'None'}
```

Nota

This particular example works as of the time of writing, but it relies on implementation details and is not guaranteed to work in the future.

Among the different kinds of expressions that exist in Python, as represented by the `ast` module, some expressions are supported, meaning that the `STRING` format can generally recover the original source code; others are unsupported, meaning that they may result in incorrect output or an error.

The following are supported (sometimes with caveats):

- `ast.BinOp`
- `ast.UnaryOp`
    - `ast.Invert` (`~`), `ast.UAdd` (`+`), and `ast.USub` (`-`) are supported
    - `ast.Not` (`not`) is not supported
- `ast.Dict` (except when using `**` unpacking)
- `ast.Set`
- `ast.Compare`
    - `ast.Eq` and `ast.NotEq` are supported
    - `ast.Lt`, `ast.LtE`, `ast.Gt`, and `ast.GtE` are supported, but the operand may be flipped
    - `ast.Is`, `ast.IsNot`, `ast.In`, and `ast.NotIn` are not supported
- `ast.Call` (except when using `**` unpacking)
- `ast.Constant` (though not the exact representation of the constant; for example, escape sequences in strings are lost; hexadecimal numbers are converted to decimal)
- `ast.Attribute` (assuming the value is not a constant)
- `ast.Subscript` (assuming the value is not a constant)
- `ast.Starred` (`*` unpacking)
- `ast.Name`
- `ast.List`
- `ast.Tuple`
- `ast.Slice`

The following are unsupported, but throw an informative error when encountered by the stringifier:

- **`ast.FormattedValue`** (f-strings; error is not detected if conversion specifiers like `!r` are used)
- **`ast.JoinedStr`** (f-strings)

The following are unsupported and result in incorrect output:

- **`ast.BoolOp`** (`and` and `or`)
- **`ast.IfExp`**
- **`ast.Lambda`**
- **`ast.ListComp`**
- **`ast.SetComp`**
- **`ast.DictComp`**
- **`ast.GeneratorExp`**

The following are disallowed in annotation scopes and therefore not relevant:

- **`ast.NamedExpr`** (`:=`)
- **`ast.Await`**
- **`ast.Yield`**
- **`ast.YieldFrom`**

Limitations of the `FORWARDREF` format

The **`FORWARDREF`** format aims to produce real values as much as possible, with anything that cannot be resolved replaced with **`ForwardRef`** objects. It is affected by broadly the same Limitations as the **`STRING`** format: annotations that perform operations on literals or that use unsupported expression types may raise exceptions when evaluated using the **`FORWARDREF`** format.

Below are a few examples of the behavior with unsupported expressions:

```
>>> from annotationlib import get_annotations, Format
>>> def zerodiv(x: 1 / 0): ...
>>> get_annotations(zerodiv, format=Format.STRING)
Traceback (most recent call last):
  ...
ZeroDivisionError: division by zero
>>> get_annotations(zerodiv, format=Format.FORWARDREF)
Traceback (most recent call last):
  ...
ZeroDivisionError: division by zero
>>> def ifexp(x: 1 if y else 0): ...
>>> get_annotations(ifexp, format=Format.STRING)
{'x': '1'}
```

Security implications of introspecting annotations

Much of the functionality in this module involves executing code related to annotations, which can then do arbitrary things. For example, **`get_annotations()`** may call an arbitrary annotate function, and **`ForwardRef.evaluate()`** may call **`eval()`** on an arbitrary string. Code contained in an annotation might make arbitrary system calls, enter an infinite loop, or perform any other operation. This is also true for any access of the **`__annotations__`** attribute, and for various functions in the **`typing`** module that work with annotations, such as **`typing.get_type_hints()`**.

Any security issue arising from this also applies immediately after importing code that may contain untrusted annotations: importing code can always cause arbitrary operations to be performed. However, it is unsafe to accept strings or other input from an untrusted source and pass them to any of the APIs for introspecting annotations, for example by editing an `__annotations__` dictionary or directly creating a **`ForwardRef`** object.

`site` — Site-specific configuration hook

Código Fuente: Lib/site.py [https://github.com/python/cpython/tree/3.14/Lib/site.py]

Este módulo se importa automáticamente durante la inicialización. La importación automática se puede suprimir utilizando la opción del intérprete opción **`-S`** .

Importing this module normally appends site-specific paths to the module search path and adds callables, including **`help()`** to the built-in namespace. However, Python startup option **`-S`** blocks this and this module can be safely imported with no automatic modifications to the module search path or additions to the builtins. To explicitly trigger the usual site-specific additions, call the **`main()`** function.

Distinto en la versión 3.3: Importar el módulo utilizado para activar la manipulación de rutas incluso cuando se utiliza **`-S`**.

It starts by constructing up to four directories from a head and a tail part. For the head part, it uses `sys.prefix` and `sys.exec_prefix`; empty heads are skipped. For the tail part, it uses the empty string and then `lib/site-packages` (on Windows) or `lib/pythonX.Y[t]/site-packages` (on Unix and macOS). (The optional suffix «t» indicates the free-threaded build, and is appended if `"t"` is present in the **`sys.abiflags`** constant.) For each of the distinct head-tail combinations, it sees if it refers to an existing directory, and if so, adds it to `sys.path` and also inspects the newly added path for configuration files.

Distinto en la versión 3.5: Se ha eliminado la compatibilidad con el directorio «site-python».

Distinto en la versión 3.13: On Unix, Free threading Python installations are identified by the «t» suffix in the version-specific directory name, such as `lib/python3.13t/`.

Distinto en la versión 3.14: **`site`** is no longer responsible for updating **`sys.prefix`** and **`sys.exec_prefix`** on Virtual Environments. This is now done during the path initialization. As a result, under Virtual Environments, **`sys.prefix`** and **`sys.exec_prefix`** no longer depend on the **`site`** initialization, and are therefore unaffected by **`-S`**.

When running under a virtual environment, the `pyvenv.cfg` file in **`sys.prefix`** is checked for site-specific configurations. If the `include-system-site-packages` key exists and is set to `true` (case-insensitive), the system-level prefixes will be searched for site-packages, otherwise they won't.

Un archivo de configuración de ruta es un archivo cuyo nombre tiene la forma *`name`*`.pth` y existe en uno de los cuatro directorios mencionados anteriormente; su contenido son elementos adicionales (uno por línea) que se agregarán a `sys.path`. Los elementos que no existen nunca se agregan a `sys.path` y no se comprueba que el elemento se refiera a un directorio en lugar de a un archivo. No se agrega ningún elemento a `sys.path` más de una vez. Se omiten las líneas en blanco y las que comienzan con `#`. Se ejecutan las líneas que comienzan con `import` (seguidas de un espacio o tabulación).

Nota

Una línea ejecutable en un archivo `.pth` se ejecuta en cada inicio de Python, independientemente de si se va a utilizar un módulo en particular. Por tanto, su impacto debería reducirse al mínimo. El propósito principal de las líneas ejecutables es hacer que los módulos correspondientes se puedan importar (cargar ganchos de importación de terceros, ajustar **`PATH`**, etc.). Se supone que cualquier otra inicialización se debe realizar en la importación real de un módulo, siempre y cuando ocurra. Limitar un fragmento de código a una sola línea es una medida deliberada para desalentar la inclusión de algo más complejo aquí.

Distinto en la versión 3.13: The `.pth` files are now decoded by UTF-8 at first and then by the locale encoding if it fails.

Por ejemplo, supongamos que `sys.prefix` y `sys.exec_prefix` están configurados en `/usr/local`. La biblioteca Python X.Y se instala en `/usr/local/lib/pythonX.Y`. Supongamos que tiene un subdirectorio `/usr/local/lib/pythonX.Y/site-packages` con tres subsubdirectorios, `foo`, `bar` y `spam`, y dos archivos de configuración de ruta, `foo.pth` y `bar.pth`. Suponga `foo.pth` contiene lo siguiente:

```
# foo package configuration

foo
bar
bletch
```

y `bar.pth` contiene:

```
# bar package configuration

bar
```

Luego, los siguientes directorios específicos de la versión se agregan a `sys.path`, en este orden:

```
/usr/local/lib/pythonX.Y/site-packages/bar
/usr/local/lib/pythonX.Y/site-packages/foo
```

Tenga en cuenta que `bletch` se omite porque no existe; el directorio `bar` precede al directorio `foo` porque `bar.pth` viene alfabéticamente antes de `foo.pth`; y `spam` se omite porque no se menciona en ninguno de los archivos de configuración de ruta.

sitecustomize

After these path manipulations, an attempt is made to import a module named **`sitecustomize`**, which can perform arbitrary site-

specific customizations. It is typically created by a system administrator in the site-packages directory. If this import fails with an **`ImportError`** or its subclass exception, and the exception's **`name`** attribute equals `'sitecustomize'`, it is silently ignored. If Python is started without output streams available, as with `pythonw.exe` on Windows (which is used by default to start IDLE), attempted output from **`sitecustomize`** is ignored. Any other exception causes a silent and perhaps mysterious failure of the process.

usercustomize

After this, an attempt is made to import a module named **`usercustomize`**, which can perform arbitrary user-specific customizations, if **`ENABLE_USER_SITE`** is true. This file is intended to be created in the user site-packages directory (see below), which is part of `sys.path` unless disabled by **`-s`**. If this import fails with an **`ImportError`** or its subclass exception, and the exception's **`name`** attribute equals `'usercustomize'`, it is silently ignored.

Note that for some non-Unix systems, `sys.prefix` and `sys.exec_prefix` are empty, and the path manipulations are skipped; however the import of **`sitecustomize`** and **`usercustomize`** is still attempted.

Configuración de *Readline*

On systems that support **`readline`**, this module will also import and configure the **`rlcompleter`** module, if Python is started in interactive mode and without the **`-S`** option. The default behavior is to enable tab completion and to use `~/.python_history` as the history save file. To disable it, delete (or override) the **`sys.__interactivehook__`** attribute in your **`sitecustomize`** or **`usercustomize`** module or your **`PYTHONSTARTUP`** file.

Distinto en la versión 3.4: La activación de *rlcompleter* y el historial se hizo automática.

Contenido del módulo

site.PREFIXES

Una lista de prefijos para directorios de site-packages.

site.ENABLE_USER_SITE

Flag que muestra el estado del directorio site-packages del usuario. `True` significa que está habilitado y se agregó a `sys.path`. `False` significa que fue deshabilitado por solicitud del usuario (con **`-s`** o **`PYTHONNOUSERSITE`**). `None` significa que fue deshabilitado por razones de seguridad (falta de coincidencia entre la identificación de usuario o grupo y la identificación efectiva) o por un administrador.

site.USER_SITE

Path to the user site-packages for the running Python. Can be `None` if **`getusersitepackages()`** hasn't been called yet. Default value is `~/.local/lib/pythonX.Y[t]/site-packages` for UNIX and non-framework macOS builds, `~/Library/Python/X.Y/lib/python/site-packages` for macOS framework builds, and `%APPDATA%\Python\PythonXY\site-packages` on Windows. The optional «t» indicates the free-threaded build. This directory is a site directory, which means that `.pth` files in it will be processed.

site.USER_BASE

Path to the base directory for the user site-packages. Can be `None` if **`getuserbase()`** hasn't been called yet. Default value is `~/.local` for UNIX and macOS non-framework builds, `~/Library/Python/X.Y` for macOS framework builds, and `%APPDATA%\Python` for Windows. This value is used to compute the installation directories for scripts, data files, Python modules, etc. for the user installation scheme. See also **`PYTHONUSERBASE`**.

site.main()

Agrega todos los directorios estándar específicos del sitio a la ruta de búsqueda del módulo. Esta función se llama automáticamente cuando se importa este módulo, a menos que el intérprete de Python se haya iniciado con la opción **`-S`**.

Distinto en la versión 3.3: Esta función solía llamarse incondicionalmente.

site.addsitedir(*sitedir*, *known_paths=None*)

Agrega un directorio a sys.path y procesa sus archivos `.pth`. Típicamente utilizado en **`sitecustomize`** o **`usercustomize`** (ver arriba).

site.getsitepackages()

Retorna una lista que contiene todos los directorios *site-packages* globales.

Added in version 3.2.

site.getuserbase()

Retorna la ruta del directorio base del usuario **`USER_BASE`**. Si este aún no fue inicializado, esta función también lo configurará, respetando **`PYTHONUSERBASE`**.

Added in version 3.2.

site.getusersitepackages()

Retorna la ruta del directorio base del usuario **`USER_SITE`**. Si este aún no fue inicializado, esta función también lo configurará, respetando **`USER_BASE`**. Para determinar si los paquetes específicos del sitio de usuario fueron añadidos a `sys.path` debe usarse **`ENABLE_USER_SITE`**.

Added in version 3.2.

Command-line interface

The **`site`** module also provides a way to get the user directories from the command line:

```
$ python -m site --user-site
/home/user/.local/lib/python3.11/site-packages
```

Si se llama sin argumentos, imprimirá el contenido de `sys.path` en la salida estándar, seguido del valor de `USER_BASE` y si el directorio existe, entonces lo mismo para `USER_SITE`, y finalmente el valor de `ENABLE_USER_SITE`.

--user-base

Imprime la ruta al directorio base del usuario.

--user-site

Imprime la ruta al directorio *site-packages* del usuario.

Si se dan ambas opciones, la ruta del directorio base y la del directorio *site-packages* del usuario se imprimirán (siempre en este orden), separados por `os.pathsep`.

Si se da alguna opción, el script saldrá con uno de estos valores: `0` si el directorio *site-packages* del usuario está habilitado, `1` si fue deshabilitado por el usuario, `2` si está deshabilitado por razones de seguridad o por un administrador, y un valor mayor que 2 si hay un error.

Ver también

- **PEP 370** [https://peps.python.org/pep-0370/] - Directorio *site-packages* de cada usuario
- La inicialización de la ruta de búsqueda de módulo de sys.path – Inicialización de `sys.path`.

Intérpretes de Python personalizados

Los módulos descritos en este capítulo permiten escribir interfaces similares al intérprete interactivo de Python. Si desea un intérprete de Python que admita alguna característica especial además del lenguaje Python, debe mirar el módulo `code`. (El módulo `codeop` es de más bajo nivel y se utiliza para soportar la compilación de un fragmento posiblemente incompleto de código Python).

La lista completa de módulos descritos en este capítulo es:

- `code` — Interpreter base classes
    - Objetos de intérprete interactivo
    - Objetos de consola interactiva
- `codeop` — Compile Python code

`code` — Interpreter base classes

Código fuente:: Lib/code.py [https://github.com/python/cpython/tree/3.14/Lib/code.py]

El módulo de `code` proporciona facilidades para implementar bucles de lectura-evaluación-impresión en Python. Se incluyen dos clases y funciones de conveniencia que se pueden usar para crear aplicaciones que brinden un *prompt* interactivo del intérprete.

class code.InteractiveInterpreter(*locals=None*)

This class deals with parsing and interpreter state (the user's namespace); it does not deal with input buffering or prompting or input file naming (the filename is always passed in explicitly). The optional *locals* argument specifies a mapping to use as the namespace in which code will be executed; it defaults to a newly created dictionary with key `'__name__'` set to `'__console__'` and key `'__doc__'` set to `None`.

Note that functions and classes objects created under an **`InteractiveInterpreter`** instance will belong to the namespace specified by *locals*. They are only pickleable if *locals* is the namespace of an existing module.

class code.InteractiveConsole(*locals=None*, *filename='<console>'*, *local_exit=False*)

Closely emulate the behavior of the interactive Python interpreter. This class builds on **`InteractiveInterpreter`** and adds prompting using the familiar `sys.ps1` and `sys.ps2`, and input buffering. If *local_exit* is true, `exit()` and `quit()` in the console will not raise **`SystemExit`**, but instead return to the calling code.

Distinto en la versión 3.13: Added *local_exit* parameter.

code.interact(*banner=None, readfunc=None, local=None, exitmsg=None, local_exit=False*)

Convenience function to run a read-eval-print loop. This creates a new instance of `InteractiveConsole` and sets *readfunc* to be used as the `InteractiveConsole.raw_input()` method, if provided. If *local* is provided, it is passed to the **`InteractiveConsole`** constructor for use as the default namespace for the interpreter loop. If *local_exit* is provided, it is passed to the **`InteractiveConsole`** constructor. The `interact()` method of the instance is then run with *banner* and *exitmsg* passed as the banner and exit message to use, if provided. The console object is discarded after use.

Distinto en la versión 3.6: Se agregó el parámetro *exitmsg*.

Distinto en la versión 3.13: Added *local_exit* parameter.

code.compile_command(*source, filename='<input>', symbol='single'*)

Esta función es útil para programas que desean emular el bucle principal del intérprete de Python (también conocido como bucle *read-eval-print*). La parte complicada es determinar cuándo el usuario ha ingresado un comando incompleto que se puede completar ingresando más texto (en lugar de un comando completo o un error de sintaxis). Esta función *casi* siempre toma la misma decisión que el bucle principal del intérprete real.

source es la cadena de caracteres fuente; *filename* es el nombre de archivo opcional desde el que se leyó la fuente, por defecto es `'<input>'`; y *symbol* es el símbolo de inicio gramatical opcional, que debería ser `'single'` (el predeterminado), `'eval'` o `'exec'`.

Retorna un objeto de código (lo mismo que `compile(source, filename, symbol)`) si el comando está completo y es válido; `None` si el comando está incompleto; lanza **`SyntaxError`** si el comando está completo y contiene un error de sintaxis, o lanza **`OverflowError`** o **`ValueError`** si el comando contiene un literal no válido.

Objetos de intérprete interactivo

InteractiveInterpreter.runsource(*source*, *filename='<input>'*, *symbol='single'*)

Compila y ejecuta alguna fuente en el intérprete. Los argumentos son los mismos que para **`compile_command()`**; el valor predeterminado para *filename* es `'<input>'`, y para *symbol* es `'single'`. Puede suceder una de varias cosas:

- La entrada es incorrecta; **`compile_command()`** generó una excepción (**`SyntaxError`** o **`OverflowError`**). Se imprime un seguimiento de sintaxis llamando al método **`showsyntaxerror()`**. **`runsource()`** retorna `False`.
- La entrada está incompleta y se requiere más información; **`compile_command()`** retorna `None`. **`runsource()`** retorna `True`.
- La entrada está completa; **`compile_command()`** retornó un objeto de código. El código se ejecuta llamando a **`runcode()`** (que también maneja excepciones en tiempo de ejecución, excepto **`SystemExit`**). **`runsource()`** retorna `False`.

El valor de retorno se puede usar para decidir si usar `sys.ps1` o `sys.ps2` para solicitar la siguiente línea.

InteractiveInterpreter.runcode(*code*)

Ejecuta un objeto de código. Cuando ocurre una excepción, se llama a **`showtraceback()`** para mostrar un seguimiento del código. Se capturan todas las excepciones excepto **`SystemExit`**, que puede propagarse.

Una nota sobre **`KeyboardInterrupt`**: esta excepción puede ocurrir en otra parte de este código y no siempre se detecta. Quien llama la función debe estar preparado para manejar esto.

InteractiveInterpreter.showsyntaxerror(*filename=None*)

Muestra el error de sintaxis que acaba de ocurrir. Esto no muestra la traza de seguimiento porque no la hay para errores de sintaxis. Si se proporciona *filename*, se incluirá en la excepción en lugar del nombre de archivo predeterminado proporcionado por el analizador de Python, ya que siempre usa `'<string>'` cuando lee de una cadena de caracteres. La salida se escribe mediante el método **`write()`**.

InteractiveInterpreter.showtraceback()

Muestra la excepción que acaba de ocurrir. Eliminamos el primer elemento de la pila porque está dentro de la implementación del intérprete. La salida se escribe mediante el método **`write()`**.

Distinto en la versión 3.5: Se muestra la traza de seguimiento encadenada completa en lugar de solo la traza primaria de pila.

InteractiveInterpreter.write(*data*)

Escribe una cadena de caracteres en el flujo de error estándar (`sys.stderr`). Las clases derivadas deben reemplazar esto para proporcionar el manejo de salida apropiado según sea necesario.

Objetos de consola interactiva

La clase **`InteractiveConsole`** es una subclase de **`InteractiveInterpreter`**, por lo que ofrece todos los métodos de los objetos del intérprete, así como las siguientes adiciones.

InteractiveConsole.interact(*banner=None, exitmsg=None*)

Emula estrictamente la consola interactiva de Python. El argumento opcional *banner* especifica la bandera a imprimir antes de la primera interacción; de forma predeterminada, imprime una bandera similar al impreso por el intérprete estándar de Python, seguido del nombre de clase del objeto de la consola entre paréntesis (para no confundir esto con el intérprete real, ¡ya que está muy cerca!)

El argumento opcional *exitmsg* especifica un mensaje de salida que se imprime al salir. Pase la cadena de caracteres vacía para suprimir el mensaje de salida. Si no se proporciona *exitmsg* o es `None`, se imprime el mensaje predeterminado.

Distinto en la versión 3.4: Para suprimir la impresión de cualquier bandera, pase una cadena de caracteres vacía.

Distinto en la versión 3.6: Imprime un mensaje de salida al salir.

InteractiveConsole.push(*line*)

Envía una línea de texto fuente al intérprete. La línea no debe tener un salto de línea al final; puede tener nuevas líneas internas. La línea se agrega a un búfer y se llama al método **`runsource()`** del intérprete con el contenido concatenado del búfer y la nueva fuente. Si indica que el comando se ejecutó o no es válido, el búfer se restablece; de lo contrario, el comando está incompleto y el búfer se deja como estaba después de agregar la línea. Si se requieren más entradas el valor de retorno es `True`, `False` si la línea se procesó de alguna manera (esto es lo mismo que **`runsource()`**).

InteractiveConsole.resetbuffer()

Elimina cualquier texto fuente no gestionado del búfer de entrada.

InteractiveConsole.raw_input(*prompt=''*)

Escribe un *prompt* y lee una línea. La línea retornada no incluye el salto de línea final. Cuando el usuario ingresa la secuencia de teclas EOF, se lanza **`OFError`**. La implementación base lee de `sys.stdin`; una subclase puede reemplazar esto con una implementación diferente.

`codeop` — Compile Python code

Código fuente: Lib/codeop.py [https://github.com/python/cpython/tree/3.14/Lib/codeop.py]

The **`codeop`** module provides utilities upon which the Python read-eval-print loop can be emulated, as is done in the **`code`** module. As a result, you probably don't want to use the module directly; if you want to include such a loop in your program you probably want to use the **`code`** module instead.

Esta actividad consta de dos partes:

1. Ser capaz de identificar si una línea de entrada completa una sentencia de Python: en resumen, decir si se debe imprimir a continuación "`>>>`" o "`...`".
2. Recordar qué declaraciones futuras ha ingresado el usuario, para que la entrada posterior puedan ser compiladas con estas en efecto.

The **`codeop`** module provides a way of doing each of these things, and a way of doing them both.

Para hacer lo anterior:

codeop.compile_command(*source*, *filename='<input>'*, *symbol='single'*)

> Intenta compilar *source*, que debe ser una cadena de caracteres de código Python y retorna un objeto si *source* es código Python válido. En este caso, el atributo del nombre del archivo del objeto va a ser *filename*, el cuál por defecto es `'<input>'`. Retorna `None` si *source* no es código Python válido, pero es un prefijo de código Python válido.

Si hay un problema con *source*, se lanzará una excepción. **`SyntaxError`** se lanza si hay una sintaxis de Python no válida, y **`OverflowError`** o **`ValueError`** si hay un literal no válido.

El argumento *symbol* determina si *source* se compila como una declaración (`'single'`, el valor predeterminado), como una secuencia de statement (`'exec'`) o como un expression (`'eval'`). Cualquier otro valor hará que se lance **`ValueError`**.

Nota

Es posible (pero no probable) que el analizador deje de analizar con un resultado exitoso antes de llegar al final de la fuente; en este caso, los símbolos finales pueden ignorarse en lugar de provocar un error. Por ejemplo, una barra invertida seguida de dos nuevas líneas puede ir seguida de basura arbitraria. Esto se solucionará una vez que la API para el analizador sea mejor.

class codeop.Compile

Las instancias de esta clase tienen métodos **`__call__()`** idénticos en firma a la función incorporada **`compile()`**, pero con la diferencia de que si la instancia compila el texto del programa que contiene una instrucción **`__future__`**, la instancia "recuerda" y compila todos los textos de programa posteriores con la declaración en vigor.

class codeop.CommandCompiler

Las instancias de esta clase tienen métodos **`__call__()`** idénticos en firma a **`compile_command()`**; la diferencia es que si la instancia compila un texto de programa que contiene una declaración **`__future__`**, la instancia "recuerda" y compila todos los textos de programa posteriores con la declaración en vigor.

Importando módulos

Los módulos descritos en este capítulo proporcionan nuevas formas de importar otros módulos de Python y ganchos para personalizar los procesos de importación.

La lista completa de módulos descritos en este capítulo es:

- **`zipimport`** — Import modules from Zip archives
    - Objetos zipimporter
    - Ejemplos
- **`pkgutil`** — Package extension utility
- **`modulefinder`** — Find modules used by a script
    - Ejemplo de uso de **`ModuleFinder`**
- **`runpy`** — Localización y ejecución de módulos Python
- **`importlib`** — La implementación de **`import`**
    - Introducción
    - Funciones
    - **`importlib.abc`** – Abstract base classes related to import
    - **`importlib.machinery`** – Importers and path hooks
    - **`importlib.util`** – Utility code for importers
    - Ejemplos
        - Importar programáticamente
        - Comprobando si se puede importar un módulo
        - Importar un archivo fuente directamente
        - Implementar importaciones diferidas
        - Configurar un importador
        - Aproximando **`importlib.import_module()`**
- **`importlib.resources`** – Package resource reading,

opening and access

    - Functional API

- **`importlib.resources.abc`** – Abstract base classes for resources
- **`importlib.metadata`** – Acceso a los metadatos de los paquetes

    - Descripción general
    - API funcional

        - Puntos de entrada
        - Metadatos de distribución
        - Versiones de distribución
        - Archivos de distribución
        - Requerimientos de la distribución
        - Mapeo de paquetes de importación a distribución

    - Distribuciones
    - Distribution Discovery
    - Implementing Custom Providers

        - Example

- La inicialización de la ruta de búsqueda de módulo de **`sys.path`**

    - Virtual Environments
    - Archivos _pth
    - Python embebido

`zipimport` — Import modules from Zip archives

Código fuente: Lib/zipimport.py [https://github.com/python/cpython/tree/3.14/Lib/zipimport.py]

This module adds the ability to import Python modules (`*.py`, `*.pyc`) and packages from ZIP-format archives. It is usually not needed to use the **`zipimport`** module explicitly; it is automatically used by the built-in **`import`** mechanism for **`sys.path`** items that are paths to ZIP archives.

Típicamente **`sys.path`** es una lista de cadenas con nombres de directorios. Este módulo también permite a un elemento de **`sys.path`** ser una cadena con la que se nombre a un archivo ZIP. El archivo ZIP puede contener una estructura de subdirectorios para soportar la importación de paquetes, y una ruta dentro del archivo puede ser especificada para únicamente importar desde un subdirectorio. Por ejemplo, la ruta `example.zip/lib/` sólo importaría desde el subdirectorio `lib/` dentro del archivo.

Cualquier archivo puede estar presente en el archivo ZIP, pero los importadores solo son invocados para archivos `.py` y `.pyc`. La importación ZIP de módulos dinámicos (`.pyd`, `.so`) no está permitida. Cabe señalar que si un archivo ZIP contiene solamente archivos `.py`, Python no intentará modificar el archivo agregando los correspondientes archivos `.pyc`, esto quiere decir que si un archivo ZIP no contiene archivos `.pyc` la importación puede ser algo lenta.

Distinto en la versión 3.13: ZIP64 is supported

Distinto en la versión 3.8: Anteriormente, los archivos ZIP con un comentario de archivo no eran compatibles.

Ver también

PKZIP Nota de aplicación [https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT]
Documentación sobre el formato de archivo ZIP por Phil Katz, el creador del formato y algoritmos utilizados.

PEP 273 [https://peps.python.org/pep-0273/] - Importar módulos de archivos Zip
Escrito por James C. Ahlstrom, quien también proporcionó una implementación. Python 2.3 sigue la especificación en PEP 273, pero utiliza una implementación escrita por Just van Rossum que utiliza los ganchos importados descritos en PEP 302.

`importlib` - La implementación de la maquinaria de importación
El paquete proporciona los protocolos relevantes para que los implementen todos los importadores.

Este módulo define una excepción:

exception zipimport.ZipImportError

Excepción lanzada por objetos zipimporter. Es una subclase de `ImportError`, por lo que también puede ser capturada como `ImportError`.

Objetos zipimporter

`zipimporter` es la clase para importar archivos ZIP.

class zipimport.zipimporter(*archivepath*)

Crea una nueva instancia zipimporter. *archivepath* debe ser una ruta a un archivo ZIP, o a una ruta específica dentro de un archivo ZIP. Por ejemplo, un *archivepath* de `foo/bar.zip/lib` buscará módulos en el directorio `lib` dentro del archivo ZIP `foo/bar.zip` (siempre que exista).

`ZipImportError` es lanzada si *archivepath* no apunta a un archivo ZIP válido.

Distinto en la versión 3.12: Los métodos `find_loader()` y `find_module()`, deprecados en la versión 3.10, han sido eliminados. Use en su lugar **`find_spec()`**.

create_module(*spec*)

Implementación de **`importlib.abc.Loader.create_module()`** que retorna **`None`** para solicitar explícitamente la semántica predeterminada.

Added in version 3.10.

exec_module(*module*)

Implementación de **`importlib.abc.Loader.exec_module()`**.

Added in version 3.10.

find_spec(*fullname*, *target=None*)

Una implementación de **`importlib.abc.PathEntryFinder.find_spec()`**.

Added in version 3.10.

get_code(*fullname*)

Retorna el objeto de código para el módulo especificado. Lanza **`ZipImportError`** si el módulo no se pudo importar.

get_data(*pathname*)

Retorna los datos asociados con *pathname*. Lanza **`OSError`** si el archivo no fue encontrado.

Distinto en la versión 3.3: **`IOError`** used to be raised, it is now an alias of **`OSError`**.

get_filename(*fullname*)

Retorna el valor que se le habría asignado a `__file__` si el módulo especificado fue importado. Lanza **`ZipImportError`** si el módulo no se pudo importar.

Added in version 3.1.

get_source(*fullname*)

Retorna el código fuente para el módulo especificado. Lanza **`ZipImportError`** si el módulo no pudo ser encontrado, retorna **`None`** si el archivo contiene al módulo, pero no tiene fuente para ello.

is_package(*fullname*)

Retorna `True` si el módulo especificado por *fullname* es un paquete. Lanza **`ZipImportError`** si el módulo no pudo ser encontrado.

load_module(*fullname*)

Carga el módulo especificado por *fullname*. *fullname* debe ser el nombre completo de módulo (punteado). Retorna el módulo importado, o lanza **`ZipImportError`** si no fue encontrado.

Deprecated since version 3.10, will be removed in version 3.15: Utilizar en su lugar **`exec_module()`**.

invalidate_caches()

Limpia la caché interna de información sobre los archivos que se encuentran dentro del archivo ZIP.

Added in version 3.10.

archive

El nombre de archivo del archivo ZIP asociado del importador, sin una posible sub-ruta.

prefix

La sub-ruta dentro del archivo ZIP donde se buscan los módulos. Esta es la cadena vacía para objetos zipimporter la cual apunta a la raíz del archivo ZIP.

Los atributos `archive` y `prefix`, cuando son combinados con una barra diagonal, son iguales al argumento original *archivepath* dado al constructor `zipimporter`.

Ejemplos

Here is an example that imports a module from a ZIP archive - note that the **`zipimport`** module is not explicitly used.

```
$ unzip -l example_archive.zip
Archive:  example_archive.zip
  Length     Date    Time    Name
 --------    ----    ----    ----
     8467  01-01-00 12:30   example.py
 --------                   -------
     8467                   1 file
```

```
>>> import sys
>>> # Add the archive to the front of the module search
>>> sys.path.insert(0, 'example_archive.zip')
>>> import example
>>> example.__file__
'example_archive.zip/example.py'
```

`pkgutil` — Package extension utility

Código fuente: Lib/pkgutil.py [https://github.com/python/cpython/tree/3.14/Lib/pkgutil.py]

Este módulo proporciona utilidades para el sistema de importación, en particular soporte para paquetes.

class pkgutil.ModuleInfo(*module_finder*, *name*, *ispkg*)

Una tupla nombrada que contiene un breve resumen de la información del módulo.

Added in version 3.6.

pkgutil.extend_path(*path*, *name*)

Extiende la ruta de búsqueda de los módulos que componen un paquete. El uso previsto es colocar el siguiente código en `__init__.py`: de un paquete:

```
from pkgutil import extend_path
__path__ = extend_path(__path__, __name__)
```

For each directory on **`sys.path`** that has a subdirectory that matches the package name, add the subdirectory to the package's **`__path__`**. This is useful if one wants to distribute different parts of a single logical package as multiple directories.

It also looks for `*.pkg` files beginning where `*` matches the *name* argument. This feature is similar to `*.pth` files (see the **`site`** module for more information), except that it doesn't special-case lines starting with `import`. A `*.pkg` file is trusted at face value: apart from skipping blank lines and ignoring comments, all entries found in a `*.pkg` file are added to the path, regardless of whether they exist on the filesystem (this is a feature).

Si la ruta de entrada no es una lista (como es el caso de los paquetes congelados), se retorna sin cambios. La ruta de entrada no se modifica; se retorna una copia ampliada. Los elementos solo se adjuntan a la copia al final.

Se supone que **`sys.path`** es una secuencia. Los elementos de **`sys.path`** que no sean cadenas de caracteres que se refieran a directorios existentes se ignoran. Los elementos Unicode en **`sys.path`** que causan errores cuando se utilizan como nombres de archivo pueden hacer que esta función lance una excepción (en línea con el comportamiento de **`os.path.isdir()`**).

pkgutil.get_importer(*path_item*)

Recupera un finder para el *path_item*.

El buscador retornado se almacena en caché en **`sys.path_importer_cache`** si fue creado recientemente por un enlace de ruta.

La caché (o parte de ella) puede ser borrada manualmente si el escaneo de **`sys.path_hooks`** es necesario.

Distinto en la versión 3.3: Actualizado para basarse directamente en **`importlib`** en lugar de depender del paquete interno **PEP 302** [https://peps.python.org/pep-0302/] emulación de importación.

pkgutil.iter_importers(*fullname=''*)

Cede (*yield*) objetos finder para el nombre de módulo dado.

If *fullname* contains a `'.'`, the finders will be for the package containing *fullname*, otherwise they will be all registered top level finders (i.e. those on both **`sys.meta_path`** and **`sys.path_hooks`**).

Si el módulo nombrado está en un paquete, ese paquete se importa como un efecto secundario de invocar esta función.

Si no se especifica ningún nombre de módulo, se generan todos los buscadores de nivel superior.

Distinto en la versión 3.3: Actualizado para basarse directamente en **`importlib`** en lugar de depender del paquete interno **PEP 302** [https://peps.python.org/pep-0302/] emulación de importación.

pkgutil.iter_modules(*path=None*, *prefix=''*)

Yields **`ModuleInfo`** for all submodules on *path*, or, if *path* is `None`, all top-level modules on **`sys.path`**.

path tendría que ser `None` o una list de rutas en las que buscar módulos.

prefix es una cadena para mostrar delante de cada nombre de módulo en la salida.

Nota

Sólo funciona para un finder que define un método `iter_modules()`. Esta interfaz no es estándar, por lo que el módulo también proporciona implementaciones para **`importlib.machinery.FileFinder`** y **`zipimport.zipimporter`**.

Distinto en la versión 3.3: Actualizado para basarse directamente en **`importlib`** en lugar de depender del paquete interno **PEP 302** [https://peps.python.org/pep-0302/] emulación de importación.

pkgutil.walk_packages(*path=None*, *prefix=''*, *onerror=None*)

Cede (*yield*) **`ModuleInfo`** para todos los módulos de forma recursiva en *path*, o, si *path* es `None`, todos los módulos accesibles.

path tendría que ser `None` o una list de rutas en las que buscar módulos.

prefix es una cadena para mostrar delante de cada nombre de módulo en la salida.

Note que esta función debe importar todos los *packages* (¡*no* todos los módulos!) en el *path* especificado, para acceder al atributo `__path__` para encontrar submódulos.

onerror es una función que se llama con un argumento (el nombre del paquete que se estaba importando) si se produce alguna excepción al intentar importar un paquete. Si no se proporciona ninguna función *onerror*, los **`ImportError`** se capturan e ignoran, mientras que todas las demás excepciones se propagan, terminando la búsqueda.

Ejemplos:

```
# list all modules python can access
walk_packages()

# list all submodules of ctypes
walk_packages(ctypes.__path__, ctypes.__name__ + '.
```

Nota

Sólo funciona para un finder que define un método `iter_modules()`. Esta interfaz no es estándar, por lo que el módulo también proporciona implementaciones para **`importlib.machinery.FileFinder`** y **`zipimport.zipimporter`**.

Distinto en la versión 3.3: Actualizado para basarse directamente en **`importlib`** en lugar de depender del paquete interno **PEP 302** [https://peps.python.org/pep-0302/] emulación de importación.

pkgutil.get_data(*package*, *resource*)

Obtiene un recurso de un paquete.

This is a wrapper for the loader **`get_data`** API. The *package* argument should be the name of a package, in standard module format (`foo.bar`). The *resource* argument should be in the form of a relative filename, using `/` as the path separator.

La función retorna una cadena de caracteres binaria que es el contenido del recurso especificado.

This function uses the loader method **`get_data()`** to support modules installed in the filesystem, but also in zip files, databases, or elsewhere.

Para los paquetes ubicados en el sistema de archivos, que ya se han importado, este es el equivalente aproximado de:

```
d = os.path.dirname(sys.modules[package].__file__)
data = open(os.path.join(d, resource), 'rb').read()
```

Like the **`open()`** function, **`get_data()`** can follow parent directories (`../`) and absolute paths (starting with `/` or `C:/`, for example). It can open compilation/installation artifacts like `.py` and `.pyc` files or files with **`reserved filenames`**. To be compatible with non-filesystem loaders, avoid using these features.

Advertencia

This function is intended for trusted input. It does not verify that *resource* «belongs» to *package*.

If you use a user-provided *resource* path, consider verifying it. For example, require an alphanumeric filename with a known extension, or install and check a list of known resources.

Si el paquete no puede ser localizado o cargado, o usa un loader el cuál no soporta **`get_data`**, entonces retorna `None`. En particular, el: término loader para namespace packages no soporta **`get_data`**.

Ver también

The `importlib.resources` module provides structured access to module resources.

pkgutil.resolve_name(*name*)

Resuelve un nombre a un objeto.

Esta funcionalidad es usada en numerosos lugares en la librería estándar (ver bpo-12915 [https://bugs.python.org/issue?@action=redirect&bpo=12915]) - y existe funcionalidad equivalente también en librerías de terceros ampliamente usadas, tales como setuptools, Django y Pyramid.

Se espera que *name* sea una cadena de caracteres en uno de los siguientes formatos, donde W representa un identificador de Python válido, y un punto representa literalmente un punto en las siguientes pseudo-expresiones regulares:

- `W(.W)*`
- `W(.W)*:(W(.W)*)?`

La primera forma existe sólo para mantener compatibilidad con versiones anteriores. Asume que parte del nombre con puntos es un paquete, y el resto es un objeto en algún lugar dentro de ese paquete, posiblemente anidado dentro de otros objetos. Dado que el lugar donde el paquete termina y la jerarquía de objetos comienza no puede ser inferida por inspección, con esta forma se debe realizar repetidos intentos de importación.

En la segunda forma, el usuario hace claro el punto de división al proveer un signo de dos puntos: el nombre con puntos a la izquierda de los dos puntos es el paquete a ser importado, y el nombre con puntos a la derecha es la jerarquía de nombres dentro de ese paquete. Sólo una importación se necesaria con esta forma. Si termina con dos puntos, entonces se retorna un objeto módulo.

La función retornará un objeto (el cual puede ser un módulo), o lanzará una de las siguientes excepciones:

`ValueError` – si *name* no tiene un formato reconocido.

`ImportError` – si una importación falló cuando no lo debería haber hecho.

`AttributeError` – Si un fallo ocurrió mientras se atravesaba la jerarquía de objetos dentro del paquete importado para obtener el objeto deseado.

Added in version 3.9.

`modulefinder` — Find modules used by a script

Código fuente: Lib/modulefinder.py [https://github.com/python/cpython/tree/3.14/Lib/modulefinder.py]

Este módulo provee una clase **`ModuleFinder`** que puede ser usada para determinar el conjunto de módulos importados en un script. `modulefinder.py` puede también ejecutarse como un script, dando el nombre de un archivo de Python como argumento, tras lo cual se imprimirá un reporte de los módulos importados.

modulefinder.AddPackagePath(*pkg_name*, *path*)

> Registra que el paquete llamado *pkg_name* pueda ser encontrado en el *path* especificado.

modulefinder.ReplacePackage(*oldname*, *newname*)

> Permite especificar que el módulo llamado *oldname* es de hecho el paquete llamado *newname*.

class modulefinder.ModuleFinder(*path=None*, *debug=0*, *excludes=[]*, *replace_paths=[]*)

> Esta clase provee los métodos **`run_script()`** y **`report()`** que determinan el conjunto de módulos importados por un script. *path* puede ser un listado de directorios a buscar por módulos; si no es especificado, se usará `sys.path`. *debug* define el nivel de depuración; valores más altos hacen que la clase imprima mensajes de depuración acerca de lo que está haciendo. *excludes* es una lista de nombres de módulos que serán excluidos del análisis. *replace_paths* es una lista de tuplas `(oldpath, newpath)` que serán remplazadas en las rutas de los módulos.

report()

Imprime un reporte a la salida estándar que lista los módulos importados por el script y sus rutas, así como los módulos que faltan o parecieran faltar.

run_script(*pathname*)

Analiza los contenidos del archivo *pathname*, que debe contener código Python.

modules

Un diccionario que mapea los nombres de los módulos a los módulos. Vea Ejemplo de uso de ModuleFinder.

Ejemplo de uso de `ModuleFinder`

El script que será analizado más adelante (bacon.py):

```
import re, itertools

try:
    import baconhameggs
except ImportError:
    pass

try:
    import guido.python.ham
except ImportError:
    pass
```

El script que generará el reporte de bacon.py:

```
from modulefinder import ModuleFinder

finder = ModuleFinder()
finder.run_script('bacon.py')

print('Loaded modules:')
for name, mod in finder.modules.items():
    print('%s: ' % name, end='')
    print(','.join(list(mod.globalnames.keys())[:3]))

print('-'*50)
print('Modules not imported:')
print('\n'.join(finder.badmodules.keys()))
```

Resultado de ejemplo (puede variar dependiendo de la arquitectura):

```
Loaded modules:
_types:
copyreg:  _inverted_registry,_slotnames,__all__
re._compiler:  isstring,_sre,_optimize_unicode
_sre:
re._constants:  REPEAT_ONE,makedict,AT_END_LINE
sys:
re:  __module__,finditer,_expand
itertools:
__main__:  re,itertools,baconhameggs
re._parser:  _PATTERNENDERS,SRE_FLAG_UNICODE
array:
types:  __module__,IntType,TypeType
--------------------------------------------------
Modules not imported:
guido.python.ham
baconhameggs
```

`runpy` — Localización y ejecución de módulos Python

Código Fuente: Lib/runpy.py [https://github.com/python/cpython/tree/3.14/Lib/runpy.py]

The `runpy` module is used to locate and run Python modules without importing them first. Its main use is to implement the `-m` command line switch that allows scripts to be located using the Python module namespace rather than the filesystem.

Tenga en cuenta que este *no* es un módulo de espacio aislado - todo el código es ejecutado en el proceso actual, y cualquier efecto secundario (como las importaciones en cache de otros módulos) permanecerán en su lugar después de que las funciones hayan retornado.

Furthermore, any functions and classes defined by the executed code are not guaranteed to work correctly after a `runpy` function has returned. If that limitation is not acceptable for a given use case, `importlib` is likely to be a more suitable choice than this module.

The `runpy` module provides two functions:

runpy.run_module(*mod_name*, *init_globals=None*, *run_name=None*, *alter_sys=False*)

> Ejecuta el código del módulo especificado y devuelve el diccionario de globales del módulo resultante. El código del módulo se encuentra primero mediante el mecanismo de importación estándar (consulte **PEP 302** [https://peps.python.org/pep-0302/] para más detalles) y luego se ejecuta en un espacio de nombres de módulo nuevo.

El argumento *mod_name* debe ser un nombre de módulo absoluto. Si el nombre del módulo se refiere a un paquete en lugar de a un módulo normal, entonces ese paquete se importa y el submódulo **`__main__`** dentro de ese paquete se ejecuta y se devuelve el diccionario de globales del módulo resultante.

El argumento de diccionario opcional *init_globals* se puede utilizar para rellenar previamente el diccionario global del módulo antes de ejecutar el código. *init_globals* no se modificará. Si alguna de las variables globales especiales siguientes se define en *init_globals*, esas definiciones se reemplazan por **`run_module()`**.

Las variables globales especiales `__name__`, `__spec__`, `__file__`, `__cached__`, `__loader__` y `__package__` se establecen en el diccionario global antes del que el código del módulo sea ejecutado (tenga en cuenta que esto es un conjunto mínimo de variables - otras variables pueden establecerse implícitamente como un detalle de implementación del intérprete).

`__name__` se establece en *run_name* si el argumento opcional no es **`None`**, para `mod_name + '.__main__'` si módulo nombrado es un paquete y al argumento *mod_name* en caso contrario.

`__spec__` se configura apropiadamente para el módulo *realmente* importado (es decir, `__spec__.name` siempre será un *mod_name* o `mod_name + '.__main__'`, nunca *run_name*).

`__file__`, `__cached__`, `__loader__` y `__package__` son establecidos de forma normal basados en la especificación del módulo.

Si el argumento *alter_sys* es proporcionado y evaluado a **`True`**, entonces `sys.argv[0]` es actualizado y el valor de `__file__` y `sys.modules[__name__]` es actualizado con un objeto de módulo temporal para el módulo que se esta ejecutado. Ambas `sys.argv[0]` y `sys.modules[__name__]` son restauradas a sus valores originales antes del retorno de la función.

Tenga en cuenta que esta manipulación de **`sys`** no es segura para los hilos. Otros hilos pueden ver el módulo parcialmente inicializado, así como la lista de argumentos alterada. Se recomienda no utilizar el módulo `sys` cuando se invoque a esta función desde código con hilos.

Ver también

La opción **`-m`** ofrece una funcionalidad equivalente desde la línea de comandos.

Distinto en la versión 3.1: Se agrego la capacidad de ejecutar paquetes buscando un submódulo **`__main__`**.

Distinto en la versión 3.2: Se agrego la variable global `__cached__` (consultar **PEP 3147** [https://peps.python.org/pep-3147/]).

Distinto en la versión 3.4: Se ha actualizado para aprovechar la función de especificación de módulos añadida por **PEP 451** [https://peps.python.org/pep-0451/]. Esto permite que `__cached__` se establezca correctamente para que los módulos se ejecuten de esta manera, así como asegurar que el nombre real del módulo siempre sea accesible como `__spec__.name`.

Distinto en la versión 3.12: El establecimiento de `__cached__`, `__loader__`, y `__package__` están obsoletos. Véase **`ModuleSpec`** para alternativas.

runpy.run_path(*path_name*, *init_globals=None*, *run_name=None*)

Ejecuta el código en la ubicación del sistema de ficheros indicada y devuelve el diccionario global del módulo resultante. Al igual que con un nombre de script suministrado a la línea de comandos de CPython, *file_path* puede referirse a un archivo fuente de Python, un archivo de código de bytes compilado o una entrada **`sys.path`** válida que contenga un módulo **`__main__`** (por ejemplo, un archivo zip que contenga un archivo de nivel superior `__main__.py`).

Para un script simple, el código especificado simplemente se ejecuta en un espacio de nombres de módulo nuevo. Para una entrada **`sys.path`** válida (normalmente un archivo zip o directorio), la entrada se añade primero al principio de `sys.path`. A continuación, la función busca y ejecuta un módulo **`__main__`** utilizando la ruta actualizada. Tenga en cuenta que no hay ninguna protección especial contra la invocación de una entrada `__main__` existente ubicada en otro lugar de `sys.path` si no existe tal módulo en la ubicación especificada.

El argumento de diccionario opcional *init_globals* se puede utilizar para rellenar previamente el diccionario global del módulo antes de ejecutar el código. *init_globals* no se modificará. Si alguna de las variables globales especiales siguientes se define en *init_globals*, esas definiciones se reemplazan por **`run_path()`**.

Las variables globales especiales `__name__`, `__spec__`, `__file__`, `__cached__`, `__loader__` y `__package__` se establecen en el diccionario global antes del que el código del módulo sea ejecutado (tenga en cuenta que esto es un conjunto mínimo de variables - otras variables pueden establecerse implícitamente como un detalle de implementación del intérprete).

`__name__` se establece para *run_name* si el argumento opcional no es **`None`** y a `'<run_path>'` de lo contrario.

Si *file_path* hace referencia directamente a un archivo script (ya sea como fuente o un código de bytes precompilado), entonces `__file__` se establecerá en *file_path*, y `__spec__`, `__cached__`, `__loader__` y `__package__` se establecerán todos en **`None`**.

Si *file_path* es una referencia a una entrada válida de **`sys.path`**, entonces `__spec__` se establecerá apropiadamente para el módulo importado **`__main__`** (es decir, `__spec__.name` siempre será `__main__`). `__file__`, `__cached__`, `__loader__` y `__package__` serán set as normal basándose en la especificación del módulo.

También se realizan una serie de modificaciones en el módulo **`sys`**. En primer lugar, **`sys.path`** puede alterarse como se ha descrito anteriormente. Se actualiza `sys.argv[0]` con el valor de *file_path* y se actualiza `sys.modules[__name__]` con un objeto módulo temporal para el módulo que se está ejecutando. Todas las modificaciones a los elementos en **`sys`** son revertidas antes de que la función retorne.

Tenga en cuenta que, a diferencia de **`run_module()`**, las alteraciones realizadas en **`sys`** no son opcionales en esta función, ya que estos ajustes son esenciales para permitir la ejecución de las entradas de **`sys.path`**. Como las limitaciones de seguridad de hilos aún se aplican, el uso de esta función en código con hilos debe ser serializado con el bloqueo de importación o delegado a un proceso separado.

Ver también

Opciones de interfaz para una funcionalidad equivalente en la línea de comandos (`python path/to/script`).

Added in version 3.2.

Distinto en la versión 3.4: Se ha actualizado para aprovechar la función de especificación de módulos añadida por **PEP 451** [https://peps.python.org/pep-0451/]. Esto permite que `__cached__` se establezca correctamente en el caso de que `__main__` se importe desde una entrada **`sys.path`** válida en lugar de ejecutarse directamente.

Distinto en la versión 3.12: El establecimiento de `__cached__`, `__loader__` y `__package__` están en desuso.

Ver también

PEP 338 [https://peps.python.org/pep-0338/] – Ejecutando módulos como scripts

PEP escrito e implementado por Nick Coghlan.

PEP 366 [https://peps.python.org/pep-0366/] – Importaciones relativas explícitas del módulo principal

PEP escrito e implementado por Nick Coghlan.

PEP 451 [https://peps.python.org/pep-0451/] — Un tipo ModuleSpec para el sistema de Importación

PEP escrito e implementado por Eric Snow

Línea de comandos y entorno - Detalles de la línea de comandos CPython

La función `importlib.import_module()`

`importlib` — La implementación de `import`

Added in version 3.1.

Código fuente: Lib/importlib/__init__.py [https://github.com/python/cpython/tree/3.14/Lib/importlib/__init__.py]

Introducción

The purpose of the `importlib` package is three-fold.

Uno es proveer la implementación de la declaración de `import` (y así, por extensión, el método `__import__()`) en el código fuente de Python. Esto provee una implementación de la `import` la cual es compatible con cualquier intérprete de Python. También provee una implementación que es más fácil de comprender que una implementada en un lenguajes que no es Python.

Dos, los componentes incluidos para implementar `import` están expuestos en este paquete para que sea más fácil para los usuarios crear sus propios objetos (conocidos de forma genérica como importer) para participar en el proceso de importación.

Tres, el paquete contiene módulos exponiendo funcionalidad adicional para administrar aspectos de paquetes de Python:

- `importlib.metadata` presenta acceso a metadatos de distribuciones de terceros.
- `importlib.resources` provee rutinas para acceder a *recursos* que no son código de paquetes de Python.

Ver también

La declaración import

La referencia en el lenguaje para la declaración de `import`.

Especificaciones de paquetes [https://www.python.org/doc/essays/packages/]

Especificaciones originales de los paquetes. Algunas semánticas han cambiado desde que este documento fue escrito (ejemplo, redirección de acuerdo a `None` en `sys.modules`).

La función `__import__()`

La declaración de `import` es una decoración sintáctica para esta función.

La inicialización de la ruta de búsqueda de módulo de sys.path

La inicialización de `sys.path`.

PEP 235 [https://peps.python.org/pep-0235/]

Importar en sistemas que no distinguen entre mayúsculas y minúsculas

PEP 263 [https://peps.python.org/pep-0263/]

Definiendo las codificaciones del código fuente de Python

PEP 302 [https://peps.python.org/pep-0302/]

Nuevos ganchos de importación

PEP 328 [https://peps.python.org/pep-0328/]

Importaciones: Multilíneas, y absolutos/relativos

PEP 366 [https://peps.python.org/pep-0366/]

Importaciones relativas, explicitas, del módulo principal

PEP 420 [https://peps.python.org/pep-0420/]

Paquetes implícitos en el espacio de nombres

PEP 451 [https://peps.python.org/pep-0451/]

Un tipo de ModuleSpec para el sistema de importación

PEP 488 [https://peps.python.org/pep-0488/]

Eliminación de archivos PYO

PEP 489 **[https://peps.python.org/pep-0489/]**
Inicialización de extensión de módulo en múltiples fases

PEP 552 **[https://peps.python.org/pep-0552/]**
Pycs determinísticos

PEP 3120 **[https://peps.python.org/pep-3120/]**
Usando UTF-8 como la codificación fuente por defecto

PEP 3147 **[https://peps.python.org/pep-3147/]**
Repositorio de directorios PYC

Funciones

importlib.__import__(*name*, *globals=None*, *locals=None*, *fromlist=()*, *level=0*)

Una implementación de la función `__import__()` incorporada.

Nota

La importación programática los módulos debe usar `import_module()` en lugar de esta función.

importlib.import_module(*name*, *package=None*)

Importar un módulo. El argumento llamado *name* especifica qué módulo importar en términos absolutos o relativos (ejemplo, puede ser `pkg.mod` o `..mod`). Si el nombre fuera especificado en términos relativos, entonces el argumento llamado *package* debe ser igual al nombre del paquete que será el ancla para resolver el nombre del paquete (ejemplo `import_module('..mod', 'pkg.subpkg')` importará `pkg.mod`).

La función `import_module()` actúa como un envoltorio simplificador alrededor de `importlib.__import__()`. Esto quiere decir que las semánticas de la función son derivadas de `importlib.__import__()`. La diferencia más importante entre las dos funciones es que `import_module()` retorna el paquete especificado o el módulo (ejemplo `pkg.mod`), mientras que `__import__()` retorna el paquete o módulo del nivel superior (ejemplo `pkg`).

Si está importando dinámicamente un módulo que se creó desde que el intérprete comenzó la ejecución (por ejemplo, creó un archivo fuente de Python), es posible que deba llamar a `invalidate_caches()` para que el nuevo módulo sea detectado por el sistema de importación.

Distinto en la versión 3.3: Paquetes padres son importados automáticamente.

importlib.invalidate_caches()

Invalide los cache internos de ubicadores encontrados en `sys.meta_path`.Si un buscador implementa `invalidate_caches()` entonces será llamado para realizar la invalidación.Esta función debe ser llamada si cualquier módulo ha sido creado/instalado mientras tu programa esta siendo ejecutado para garantizar que todos los buscadores noten la existencia del nuevo módulo.

Added in version 3.3.

Distinto en la versión 3.10: Paquetes de espacio de nombres creados/instalados en una ubicación `sys.path` distinta después de que el mismo espacio de nombres fue importado son notados.

importlib.reload(*module*)

Recarga un *modulo* previamente importado. El argumento debe ser un objeto módulo, por lo que debe haber sido importado exitosamente. Esto es útil cuando has editado el código fuente de un archivo usando un editor externo y deseas probar la nueva versión sin abandonar el interprete de Python. El valor retornado es el objeto módulo (que puede ser diferente si la reimportación crea un nuevo objeto en `sys.modules`).

Cuando `reload()` es ejecutada:

- El código de un módulo de Python es recompilado y el código del módulo reejecutado, definiendo un nuevo conjunto de objetos que son asignados a los nombres de los módulos en el diccionario, reusando el loader que originalmente carga los módulos. El método `init` de los módulos de extension no es llamado de nuevo.
- Al igual que con todos los demás objetos en Python, los objetos antiguos solo se recuperan después de que sus recuentos de referencias caen a cero.
- Los nombres en el espacio de nombres del módulo se actualizan para señalar cualquier objeto nuevo o modificado.
- Otras referencias a los objetos antiguos (como los nombres externos al módulo) no se vuelven a vincular para hacer referencia a los nuevos objetos y deben actualizarse en cada espacio de nombres donde se produzcan si se desea.

Hay una serie de otras advertencias:

Cuando se vuelve a cargar un módulo, se conserva su diccionario (que contiene las variables globales del módulo). Las redefiniciones de nombres anularán las antiguas definiciones, por lo que generalmente esto no es un problema. Si la nueva versión de un módulo no define un nombre que fue definido por la versión anterior, la definición anterior permanece. Esta característica se puede utilizar en beneficio del módulo si mantiene una tabla global o caché de objetos — con una declaración `try` puede probar la presencia de la tabla y omitir su inicialización si lo desea:

```
try:
    cache
except NameError:
    cache = {}
```

Por lo general, no es muy útil recargar módulos integrados o cargados dinámicamente. No se recomienda recargar `sys`, `__main__`, `builtins` y otros módulos clave. En muchos casos, los módulos de extensión no están diseñados para inicializarse más de una vez y pueden fallar de manera arbitraria cuando se vuelven a cargar.

Si un módulo importa objetos de otro módulo usando `from` … `import` …, al llamar a `reload()` para el otro módulo no redefine los objetos importados de él — una forma de evitar esto es volver a ejecutar la instrucción `from`, otra es usar `import` y nombres calificados (*module.name*) en su lugar.

Si un módulo crea instancias de una clase, volver a cargar el módulo que define la clase no afecta las definiciones de método de las instancias — continúan usando la definición de clase anterior. Lo mismo ocurre con las clases derivadas.

Added in version 3.4.

Distinto en la versión 3.7: `ModuleNotFoundError` se lanza cuando el módulo que se está recargando carece de `ModuleSpec`.

Advertencia

This function is not thread-safe. Calling it from multiple threads can result in unexpected behavior. It’s recommended to use the `threading.Lock` or other synchronization primitives for thread-safe module reloading.

`importlib.abc` – Abstract base classes related to import

Código fuente: Lib/importlib/abc.py [https://github.com/python/cpython/tree/3.14/Lib/importlib/abc.py]

The **`importlib.abc`** module contains all of the core abstract base classes used by **`import`**. Some subclasses of the core abstract base classes are also provided to help in implementing the core ABCs.

Jerarquía ABC:

```
object
 +-- MetaPathFinder
 +-- PathEntryFinder
 +-- Loader
      +-- ResourceLoader --------+
      +-- InspectLoader          |
           +-- ExecutionLoader --+
                                  +-- FileLoader
                                  +-- SourceLoader
```

class importlib.abc.MetaPathFinder

Una clase base abstracta que representa meta path finder.

Added in version 3.3.

Distinto en la versión 3.10: Ya no hereda de **`Finder`**.

find_spec(*fullname*, *path*, *target=None*)

An abstract method for finding a spec for the specified module. If this is a top-level import, *path* will be `None`. Otherwise, this is a search for a subpackage or module and *path* will be the value of **`__path__`** from the parent package. If a spec cannot be found, `None` is returned. When passed in, `target` is a module object that the finder may use to make a more educated guess about what spec to return. **`importlib.util.spec_from_loader()`** may be useful for implementing concrete `MetaPathFinders`.

Added in version 3.4.

invalidate_caches()

Un método opcional que, cuando se llama, debería invalidar cualquier caché interno utilizado por el buscador. Utilizado por **`importlib.invalidate_caches()`** al invalidar los cachés de todos los buscadores en **`sys.meta_path`**.

Distinto en la versión 3.4: Retorna `None` cuando se llama en lugar de **`NotImplemented`**.

class importlib.abc.PathEntryFinder

Una clase base abstracta que representa un buscador de entradas de ruta. Aunque tiene algunas similitudes con **`MetaPathFinder`**, `PathEntryFinder` está diseñado para usarse solo dentro del subsistema de importación basado en rutas proporcionado por **`importlib.machinery.PathFinder`**.

Added in version 3.3.

Distinto en la versión 3.10: Ya no hereda de **`Finder`**.

find_spec(*fullname*, *target=None*)

Un método abstracto para encontrar un spec para el módulo especificado. El buscador buscará el módulo solo dentro del path entry a la que está asignado. Si no se puede encontrar una especificación, se retorna `None`. Cuando se pasa, `target` es un objeto de módulo que el buscador puede usar para hacer una suposición más informada sobre qué especificación retornar. **`importlib.util.spec_from_loader()`** puede ser útil para implementar `PathEntryFinders` concretos.

Added in version 3.4.

invalidate_caches()

Un método opcional que, cuando se llama, debería invalidar cualquier caché interno utilizado por el buscador. Usado por `importlib.machinery.PathFinder.invalidate_cache` al invalidar las cachés de todos los buscadores en caché.

class importlib.abc.Loader

Una clase base abstracta para un loader. Consulte **PEP 302** [https://peps.python.org/pep-0302/] para obtener la definición exacta de cargador.

Los cargadores que deseen admitir la lectura de recursos deben implementar un método `get_resource_reader()` según lo especificado por `importlib.abc.ResourceReader`.

Distinto en la versión 3.7: Introducido el método opcional `get_resource_reader()`.

create_module(*spec*)

Un método que retorna el objeto de módulo que se utilizará al importar un módulo. Este método puede retornar `None`, lo que indica que se debe llevar a cabo la semántica de creación de módulos predeterminada.

Added in version 3.4.

Distinto en la versión 3.6: Este método ya no es opcional cuando se defina `exec_module()`.

exec_module(*module*)

Un método abstracto que ejecuta el módulo en su propio espacio de nombres cuando se importa o se vuelve a cargar un módulo. El módulo ya debería estar inicializado cuando se llama a `exec_module()`. Cuando existe este método, se debe definir `create_module()`.

Added in version 3.4.

Distinto en la versión 3.6: `create_module()` también debe definirse.

load_module(*fullname*)

Un método heredado para cargar un módulo. Si el módulo no se puede cargar, se lanza `ImportError`; de lo contrario, se retorna el módulo cargado.

Si el módulo solicitado ya existe en `sys.modules`, ese módulo debe usarse y recargarse. De lo contrario, el cargador debe crear un nuevo módulo e insertarlo en `sys.modules` antes de que comience la carga, para evitar la recursividad de la importación. Si el cargador insertó un módulo y la carga falla, el cargador debe eliminarlo de `sys.modules`; los módulos que ya estaban en `sys.modules` antes de que el cargador comenzara a ejecutarse deben dejarse intactos.

El cargador debe establecer varios atributos en el módulo (tenga en cuenta que algunos de estos atributos pueden cambiar cuando se recarga un módulo):

- `module.__name__`
- `module.__file__`
- `module.__cached__` *(deprecated)*
- `module.__path__`
- `module.__package__` *(deprecated)*
- `module.__loader__` *(deprecated)*

Cuando `exec_module()` está disponible, se proporciona una funcionalidad compatible con versiones anteriores.

Distinto en la versión 3.4: Lanza `ImportError` cuando se llama en lugar de `NotImplementedError`. Funcionalidad proporcionada cuando `exec_module()` está disponible.

Deprecated since version 3.4, will be removed in version 3.15: La API recomendada para cargar un módulo es `exec_module()` (y `create_module()`). Los cargadores deberían implementarlo en lugar de `load_module()`. La maquinaria de importación se encarga de todas las demás responsabilidades de **`load_module()`** cuando se implementa **`exec_module()`**.

class importlib.abc.ResourceLoader

Reemplazado por TraversableResources

Una clase base abstracta para un loader que implementa el protocolo opcional **PEP 302** [https://peps.python.org/pep-0302/] para cargar recursos arbitrarios desde el back-end de almacenamiento.

Obsoleto desde la versión 3.7: This ABC is deprecated in favour of supporting resource loading through **`importlib.resources.abc.TraversableResources`**. This class exists for backwards compatibility only with other ABCs in this module.

abstractmethod get_data(*path*)

An abstract method to return the bytes for the data located at *path*. Loaders that have a file-like storage back-end that allows storing arbitrary data can implement this abstract method to give direct access to the data stored. `OSError` is to be raised if the *path* cannot be found. The *path* is expected to be constructed using a module's `__file__` attribute or an item from a package's `__path__`.

Distinto en la versión 3.4: Lanza `OSError` en vez de `NotImplementedError`.

class importlib.abc.InspectLoader

Una clase base abstracta para un loader que implementa el protocolo opcional **PEP 302** [https://peps.python.org/pep-0302/] para cargadores que inspeccionan módulos.

get_code(*fullname*)

Retorna el objeto código para un módulo, o `None` si el módulo no tiene un objeto código (como sería el caso, por ejemplo, para un módulo integrado). Lanza un `ImportError` si el cargador no puede encontrar el módulo solicitado.

Nota

Si bien el método tiene una implementación predeterminada, se sugiere que se anule si es posible para mejorar el rendimiento.

Distinto en la versión 3.4: Ya no es un método abstracto y se proporciona una implementación concreta.

abstractmethod get_source(*fullname*)

Un método abstracto para retornar la fuente de un módulo. Se retorna como una cadena de caracteres de texto usando universal newlines, traduciendo todos los separadores de línea reconocidos en caracteres `'\n'`.

Retorna `None` si no hay una fuente disponible (por ejemplo, un módulo integrado). Lanza la excepción **`ImportError`** si el cargador no puede encontrar el módulo especificado.

Distinto en la versión 3.4: Lanza **`ImportError`** en vez de **`NotImplementedError`**.

is_package(*fullname*)

Un método opcional para retornar un valor verdadero si el módulo es un paquete, un valor falso en caso contrario. Se lanza **`ImportError`** si el cargador no puede encontrar el módulo.

Distinto en la versión 3.4: Lanza **`ImportError`** en vez de **`NotImplementedError`**.

static source_to_code(*data*, *path='<string>'*)

Cree un objeto de código a partir de la fuente de Python.

El argumento *data* puede ser cualquier cosa que admita la función **`compile()`** (es decir, cadena de caracteres o bytes). El argumento *path* debe ser la «ruta» de donde se originó el código fuente, que puede ser un concepto abstracto (por ejemplo, ubicación en un archivo zip).

Con el objeto de código subsiguiente, uno puede ejecutarlo en un módulo ejecutando `exec(code, module.__dict__)`.

Added in version 3.4.

Distinto en la versión 3.5: Hace el método estático.

exec_module(*module*)

Implementación de **`Loader.exec_module()`**.

Added in version 3.4.

load_module(*fullname*)

Implementación de `Loader.load_module()`.

Deprecated since version 3.4, will be removed in version 3.15: use `exec_module()` en su lugar.

class importlib.abc.ExecutionLoader

Una clase base abstracta que hereda de `InspectLoader` que, cuando se implementa, ayuda a que un módulo se ejecute como un script. El ABC representa un protocolo opcional **PEP 302** [https://peps.python.org/pep-0302/].

abstractmethod get_filename(*fullname*)

An abstract method that is to return the value of `__file__` for the specified module. If no path is available, `ImportError` is raised.

Si el código fuente está disponible, entonces el método debe devolver la ruta al archivo fuente, independientemente de si se utilizó un código de bytes para cargar el módulo.

Distinto en la versión 3.4: Lanza `ImportError` en vez de `NotImplementedError`.

class importlib.abc.FileLoader(*fullname*, *path*)

Una clase base abstracta que hereda de `ResourceLoader` y `ExecutionLoader`, proporcionando implementaciones concretas de `ResourceLoader.get_data()` y `ExecutionLoader.get_filename()`.

El argumento *fullname* es un nombre completamente resuelto del módulo que el cargador debe manejar. El argumento *path* es la ruta al archivo del módulo.

Added in version 3.3.

name

El nombre del módulo que puede manejar el cargador.

path

Ruta al archivo del módulo.

load_module(*fullname*)

Llama a super's `load_module()`.

Deprecated since version 3.4, will be removed in version 3.15: Utilice **`Loader.exec_module()`** en su lugar.

abstractmethod get_filename(*fullname*)

Retorna **`path`**.

abstractmethod get_data(*path*)

Lee *path* como un archivo binario y retorna los bytes de él.

class importlib.abc.SourceLoader

Una clase base abstracta para implementar la carga de archivos fuente (y opcionalmente bytecode). La clase hereda tanto de **`ResourceLoader`** como de **`ExecutionLoader`**, lo que requiere la implementación de:

- **`ResourceLoader.get_data()`**
- **`ExecutionLoader.get_filename()`**
  Solo debe devolver la ruta al archivo de origen; la carga sin fuente no es compatible.

Los métodos abstractos definidos por esta clase son para agregar soporte de archivo de código de bytes opcional. No implementar estos métodos opcionales (o hacer que se lance

`NotImplementedError`) hace que el cargador solo funcione con el código fuente. La implementación de los métodos permite que el cargador trabaje con archivos fuente *y* código de bytes; no permite la carga *sin fuente* donde solo se proporciona un código de bytes. Los archivos de código de bytes son una optimización para acelerar la carga al eliminar el paso de análisis del compilador de Python, por lo que no se expone ninguna API específica de código de bytes.

path_stats(*path*)

Método abstracto opcional que retorna un `dict` que contiene metadatos sobre la ruta especificada. Las claves de diccionario admitidas son:

- `'mtime'` (obligatorio): un número entero o de punto flotante que representa la hora de modificación del código fuente;
- `'size'` (opcional): el tamaño en bytes del código fuente.

Cualquier otra clave del diccionario se ignora para permitir futuras extensiones. Si no se puede manejar la ruta, se genera `OSError`.

Added in version 3.3.

Distinto en la versión 3.4: Lanza `OSError` en vez de `NotImplementedError`.

path_mtime(*path*)

Método abstracto opcional que retorna la hora de modificación de la ruta especificada.

Obsoleto desde la versión 3.3: Este método está obsoleto en favor de `path_stats()`. No tiene que implementarlo, pero aún está disponible para fines de compatibilidad. Lanza `OSError` si la ruta no se puede manejar.

Distinto en la versión 3.4: Lanza `OSError` en vez de

`NotImplementedError`.

set_data(*path*, *data*)

Método abstracto opcional que escribe los bytes especificados en una ruta de archivo. Los directorios intermedios que no existan se crearán automáticamente.

Cuando la escritura en la ruta falla porque la ruta es de solo lectura (`errno.EACCES`/`PermissionError`), no propague la excepción.

Distinto en la versión 3.4: Ya no lanza `NotImplementedError` cuando se llama.

get_code(*fullname*)

Implementación concreta de `InspectLoader.get_code()`.

exec_module(*module*)

Implementación concreta de `Loader.exec_module()`.

Added in version 3.4.

load_module(*fullname*)

Implementación concreta de `Loader.load_module()`.

Deprecated since version 3.4, will be removed in version 3.15: Utilice `exec_module()` en su lugar.

get_source(*fullname*)

Implementación concreta de `InspectLoader.get_source()`.

is_package(*fullname*)

Implementación concreta de **`InspectLoader.is_package()`**. Se determina que un módulo es un paquete si su ruta de archivo (proporcionada por **`ExecutionLoader.get_filename()`**) es un archivo llamado `__init__` cuando se elimina la extensión del archivo **y** el nombre del módulo sí lo hace no termina en `__init__`.

`importlib.machinery` – Importers and path hooks

Código fuente: Lib/importlib/machinery.py [https://github.com/python/cpython/tree/3.14/Lib/importlib/machinery.py]

Este módulo contiene varios objetos que ayudan **`import`** buscar y cargar módulos.

importlib.machinery.SOURCE_SUFFIXES

Una lista de cadenas de caracteres que representan los sufijos de archivo reconocidos para los módulos de origen.

Added in version 3.3.

importlib.machinery.DEBUG_BYTECODE_SUFFIXES

Una lista de cadenas que representan los sufijos de archivo para módulos de código de bytes no optimizados.

Added in version 3.3.

Obsoleto desde la versión 3.5: Use **`BYTECODE_SUFFIXES`** instead.

importlib.machinery.OPTIMIZED_BYTECODE_SUFFIXES

Una lista de cadenas de caracteres que representan los sufijos de archivo para módulos de código de bytes optimizados.

Added in version 3.3.

Obsoleto desde la versión 3.5: Use **`BYTECODE_SUFFIXES`** instead.

importlib.machinery.BYTECODE_SUFFIXES

Una lista de cadenas de caracteres que representan los sufijos de archivo reconocidos para los módulos de código de bytes (incluido el punto inicial).

Added in version 3.3.

Distinto en la versión 3.5: El valor ya no depende de `__debug__`.

importlib.machinery.EXTENSION_SUFFIXES

Una lista de cadenas de caracteres que representan los sufijos de archivo reconocidos para los módulos de extensión.

Added in version 3.3.

importlib.machinery.all_suffixes()

Retorna una lista combinada de cadenas de caracteres que representan todos los sufijos de archivo para módulos reconocidos por la maquinaria de importación estándar. Este es un ayudante para el código que simplemente necesita saber si una ruta del sistema de archivos potencialmente se refiere a un módulo sin necesidad de detalles sobre el tipo de módulo (por ejemplo, **`inspect.getmodulename()`**).

Added in version 3.3.

class importlib.machinery.BuiltinImporter

Un importer para módulos integrados. Todos los módulos integrados conocidos se enumeran en **`sys.builtin_module_names`**. Esta clase implementa los ABC **`importlib.abc.MetaPathFinder`** y **`importlib.abc.InspectLoader`**.

Esta clase solo define los métodos de clase para aliviar la necesidad de instanciación.

Distinto en la versión 3.5: Como parte de **PEP 489** [https://peps.python.org/pep-0489/], el importador integrado ahora implementa **`Loader.create_module()`** y **`Loader.exec_module()`**

class importlib.machinery.FrozenImporter

Un importer para módulos congelados. Esta clase implementa los ABC **`importlib.abc.MetaPathFinder`** y **`importlib.abc.InspectLoader`**.

Esta clase solo define los métodos de clase para aliviar la necesidad de instanciación.

Distinto en la versión 3.4: Métodos obtenidos **`create_module()`** y **`exec_module()`**.

class importlib.machinery.WindowsRegistryFinder

Finder para los módulos declarados en el registro de Windows. Esta clase implementa el **`importlib.abc.MetaPathFinder`** ABC.

Esta clase solo define los métodos de clase para aliviar la necesidad de instanciación.

Added in version 3.3.

Obsoleto desde la versión 3.6: Utilice la configuración de **`site`** en su lugar. Es posible que las versiones futuras de Python no habiliten este buscador de forma predeterminada.

class importlib.machinery.PathFinder

Un Finder para **`sys.path`** y atributos del paquete `__path__`. Esta clase implementa el **`importlib.abc.MetaPathFinder`** ABC.

Esta clase solo define los métodos de clase para aliviar la necesidad de instanciación.

classmethod find_spec(*fullname*, *path=None*, *target=None*)

Método de clase que intenta encontrar un spec para el módulo especificado por *fullname* en `sys.path` o, si está definido, en *path*. Para cada entrada de ruta que se busca, se comprueba `sys.path_importer_cache`. Si se encuentra un objeto que no es falso, se utiliza como path entry finder para buscar el módulo que se está buscando. Si no se encuentra ninguna entrada en `sys.path_importer_cache`, entonces `sys.path_hooks` se busca un buscador para la entrada de ruta y, si se encuentra, se almacena en `sys.path_importer_cache` junto con ser consultado sobre el módulo. Si nunca se encuentra ningún buscador, entonces `None` se almacena en el caché y se retorna.

Added in version 3.4.

Distinto en la versión 3.5: Si el directorio de trabajo actual, representado por una cadena de caracteres vacía, ya no es válido, se retorna `None` pero no se almacena ningún valor en `sys.path_importer_cache`.

classmethod invalidate_caches()

Llama `importlib.abc.PathEntryFinder.invalidate_caches` en todos los buscadores almacenados en `sys.path_importer_cache` que definen el método. De lo contrario, las entradas en `sys.path_importer_cache` establecidas en `None` se eliminan.

Distinto en la versión 3.7: Se eliminan las entradas de `None` en `sys.path_importer_cache`.

Distinto en la versión 3.4: Llama a objetos en `sys.path_hooks` con el directorio de trabajo actual para `''` (es decir, la cadena de caracteres vacía).

class importlib.machinery.FileFinder(*path*, **loader_details*)

Una implementación concreta de **`importlib.abc.PathEntryFinder`** que almacena en caché los resultados del sistema de archivos.

El argumento *path* es el directorio que el buscador se encarga de buscar.

El argumento *loader_details* es un número variable de tuplas de 2 elementos, cada una de las cuales contiene un cargador y una secuencia de sufijos de archivo que el cargador reconoce. Se espera que los cargadores sean invocables que acepten dos argumentos del nombre del módulo y la ruta al archivo encontrado.

El buscador almacenará en caché el contenido del directorio según sea necesario, haciendo llamadas estadísticas para cada búsqueda de módulo para verificar que la caché no esté desactualizada. Debido a que la obsolescencia de la caché se basa en la granularidad de la información de estado del sistema operativo del sistema de archivos, existe una condición de carrera potencial de buscar un módulo, crear un nuevo archivo y luego buscar el módulo que representa el nuevo archivo. Si las operaciones ocurren lo suficientemente rápido como para ajustarse a la granularidad de las llamadas estadísticas, la búsqueda del módulo fallará. Para evitar que esto suceda, cuando cree un módulo dinámicamente, asegúrese de llamar a **`importlib.invalidate_caches()`**.

Added in version 3.3.

path

La ruta en la que buscará el buscador.

find_spec(*fullname*, *target=None*)

Intente encontrar la especificación para manejar *fullname* dentro de **`path`**.

Added in version 3.4.

invalidate_caches()

Borrar el caché interno.

classmethod path_hook(**loader_details*)

Un método de clase que retorna un cierre para su uso en `sys.path_hooks`. El cierre retorna una instancia de `FileFinder` utilizando el argumento de ruta proporcionado al cierre directamente y *loader_details* indirectamente.

Si el argumento del cierre no es un directorio existente, se lanza `ImportError`.

class importlib.machinery.SourceFileLoader(*fullname*, *path*)

Una implementación concreta de `importlib.abc.SourceLoader` subclasificando `importlib.abc.FileLoader` y proporcionando algunas implementaciones concretas de otros métodos.

Added in version 3.3.

name

El nombre del módulo que manejará este cargador.

path

La ruta al archivo de origen.

is_package(*fullname*)

Retorna `True` si `path` parece ser para un paquete.

path_stats(*path*)

Implementación concreta de `importlib.abc.SourceLoader.path_stats()`.

set_data(*path*, *data*)

Implementación concreta de `importlib.abc.SourceLoader.set_data()`.

load_module(*name=None*)

Implementación concreta de `importlib.abc.Loader.load_module()` donde especificar el nombre del módulo a cargar es opcional.

Deprecated since version 3.6, will be removed in version 3.15: Utilice `importlib.abc.Loader.exec_module()` en su lugar.

class importlib.machinery.SourcelessFileLoader(*fullname*, *path*)

Una implementación concreta de `importlib.abc.FileLoader` que puede importar archivos de código de bytes (es decir, no existen archivos de código fuente).

Tenga en cuenta que el uso directo de archivos de código de bytes (y, por lo tanto, no de archivos de código fuente) impide que sus módulos sean utilizables por todas las implementaciones de Python o las nuevas versiones de Python que cambian el formato de código de bytes.

Added in version 3.3.

name

El nombre del módulo que manejará el cargador.

path

La ruta al archivo de código de bytes.

is_package(*fullname*)

Determina si el módulo es un paquete basado en `path`.

get_code(*fullname*)

Retorna el objeto de código para `name` creado a partir de `path`.

get_source(*fullname*)

Retorna `None` ya que los archivos de código de bytes no tienen fuente cuando se usa este cargador.

load_module(*name=None*)

Implementación concreta de **`importlib.abc.Loader.load_module()`** donde especificar el nombre del módulo a cargar es opcional.

Deprecated since version 3.6, will be removed in version 3.15: Utilice **`importlib.abc.Loader.exec_module()`** en su lugar.

class importlib.machinery.ExtensionFileLoader(*fullname*, *path*)

Una implementación concreta de **`importlib.abc.ExecutionLoader`** para módulos de extensión.

El argumento *fullname* especifica el nombre del módulo que el cargador debe admitir. El argumento *path* es la ruta al archivo del módulo de extensión.

Tenga en cuenta que, de manera predeterminada, la importación de un módulo de extensión fallará en los subintérpretes si no implementa la inicialización multifase (ver **PEP 489** [https://peps.python.org/pep-0489/]), incluso si de lo contrario se importaría correctamente.

Added in version 3.3.

Distinto en la versión 3.12: Ahora se requiere inicialización multifase para el uso en subintérpretes.

name

Nombre del módulo que admite el cargador.

path

Ruta al módulo de extensión.

create_module(*spec*)

Crea el objeto de módulo a partir de la especificación dada de acuerdo con **PEP 489** [https://peps.python.org/pep-0489/].

Added in version 3.5.

exec_module(*module*)

Inicializa el objeto de módulo dado de acuerdo con **PEP 489** [https://peps.python.org/pep-0489/].

Added in version 3.5.

is_package(*fullname*)

Returns `True` if the file path points to a package's `__init__` module based on **`EXTENSION_SUFFIXES`**.

get_code(*fullname*)

Retorna `None` ya que los módulos de extensión carecen de un objeto de código.

get_source(*fullname*)

Retorna `None` ya que los módulos de extensión no tienen código fuente.

get_filename(*fullname*)

Retorna **`path`**.

Added in version 3.4.

class importlib.machinery.NamespaceLoader(*name*, *path*, *path_finder*)

Una implementación concreta de **`importlib.abc.InspectLoader`** para paquetes de espacio de nombres. Ésta es un alias para una clase privada y sólo se hace pública para introspeccionar el atributo `__loader__`

en paquetes de espacio de nombres:

```
>>> from importlib.machinery import NamespaceLoader
>>> import my_namespace
>>> isinstance(my_namespace.__loader__, NamespaceLo
True
>>> import importlib.abc
>>> isinstance(my_namespace.__loader__, importlib.a
True
```

Added in version 3.11.

class importlib.machinery.ModuleSpec(*name*, *loader*, ***, *origin=None*, *loader_state=None*, *is_package=None*)

A specification for a module's import-system-related state. This is typically exposed as the module's **`__spec__`** attribute. Many of these attributes are also available directly on a module: for example, `module.__spec__.origin == module.__file__`. Note, however, that while the *values* are usually equivalent, they can differ since there is no synchronization between the two objects. For example, it is possible to update the module's **`__file__`** at runtime and this will not be automatically reflected in the module's **`__spec__.origin`**, and vice versa.

Added in version 3.4.

name

The module's fully qualified name (see **`module.__name__`**). The finder should always set this attribute to a non-empty string.

loader

The loader used to load the module (see **`module.__loader__`**). The finder should always set this attribute.

origin

The location the loader should use to load the module (see **`module.__file__`**). For example, for modules loaded from a `.py` file this is the filename. The finder should always set this attribute to a meaningful value for the loader to use. In the uncommon case that there is not one (like for namespace packages), it should be set to `None`.

submodule_search_locations

A (possibly empty) sequence of strings enumerating the locations in which a package's submodules will be found (see **`module.__path__`**). Most of the time there will only be a single directory in this list.

The finder should set this attribute to a sequence, even an empty one, to indicate to the import system that the module is a package. It should be set to `None` for non-package modules. It is set automatically later to a special object for namespace packages.

loader_state

El buscador podría establecer este atributo a un objeto conteniendo datos adicionales y específicos al módulo para usar cuando se carga el módulo.. De lo contrario, debe establecerse en `None`.

cached

The filename of a compiled version of the module's code (see **`module.__cached__`**). The finder should always set this attribute but it may be `None` for modules that do not need compiled code stored.

parent

(Read-only) The fully qualified name of the package the module is in (or the empty string for a top-level module). See **`module.__package__`**. If the module is a package then this is the same as **`name`**.

has_location

`True` if the spec's **`origin`** refers to a loadable location, `False` otherwise. This value impacts how **`origin`** is interpreted and how the module's **`__file__`** is populated.

class importlib.machinery.AppleFrameworkLoader(*name*, *path*)

Una especialización de **`importlib.machinery.ExtensionFileLoader`** que puede cargar módulos de extensión en formato Framework.

Para que sea compatible con la App Store de iOS, *todos* los módulos binarios de una aplicación de iOS deben ser bibliotecas dinámicas, contenidas en un marco con metadatos apropiados, almacenados en la carpeta `Frameworks` de la aplicación empaquetada. Solo puede haber un único binario por marco y no puede haber material binario ejecutable fuera de la carpeta Frameworks.

To accommodate this requirement, when running on iOS, extension module binaries are *not* packaged as `.so` files on `sys.path`, but as individual standalone frameworks. To discover those frameworks, this loader is registered against the `.fwork` file extension, with a `.fwork` file acting as a placeholder in the original location of the binary on `sys.path`. The `.fwork` file contains the path of the actual binary in the `Frameworks` folder, relative to the app bundle. To allow for resolving a framework-packaged binary back to the original location, the framework is expected to contain a `.origin` file that contains the location of the `.fwork` file, relative to the app bundle.

Por ejemplo, considere el caso de una importación `from foo.bar import _whiz`, donde `_whiz` se implementa con el módulo binario `sources/foo/bar/_whiz.abi3.so`, donde `sources` es la ubicación registrada en `sys.path`, relativa al paquete de la aplicación. Este módulo *debe* distribuirse como `Frameworks/foo.bar._whiz.framework/foo.bar._whiz` (creando el nombre del marco a partir de la ruta de importación completa del módulo), con un archivo `Info.plist` en el directorio

`.framework` que identifica el binario como un marco. El módulo `foo.bar._whiz` se representaría en la ubicación original con un archivo marcador `sources/foo/bar/_whiz.abi3.fwork`, que contiene la ruta `Frameworks/foo.bar._whiz/foo.bar._whiz`. El framework también contendría `Frameworks/foo.bar._whiz.framework/foo.bar._whiz.origin`, que contiene la ruta al archivo `.fwork`.

When a module is loaded with this loader, the `__file__` for the module will report as the location of the `.fwork` file. This allows code to use the `__file__` of a module as an anchor for file system traversal. However, the spec origin will reference the location of the *actual* binary in the `.framework` folder.

El proyecto Xcode que crea la aplicación es responsable de convertir los archivos `.so` desde donde se encuentren en `PYTHONPATH` en marcos en la carpeta `Frameworks` (lo que incluye quitar las extensiones del archivo del módulo, agregar metadatos del marco y firmar el marco resultante), y crear los archivos `.fwork` y `.origin`. Esto generalmente se hará con un paso de compilación en el proyecto Xcode; consulte la documentación de iOS para obtener detalles sobre cómo construir este paso de compilación.

Added in version 3.13.

Availability: iOS.

name

Nombre del módulo que admite el cargador.

path

Ruta al archivo `.fwork` para el módulo de extensión.

`importlib.util` – Utility code for importers

Código fuente: Lib/importlib/util.py [https://github.com/python/cpython/tree/3.14/Lib/importlib/util.py]

Este módulo contiene los diversos objetos que ayudan en la construcción de un importer.

importlib.util.MAGIC_NUMBER

Los bytes que representan el número de versión del código de bytes. Si necesita ayuda para cargar/escribir código de bytes, considere **`importlib.abc.SourceLoader`**.

Added in version 3.4.

importlib.util.cache_from_source(*path*, *debug_override*=*None*, *, *optimization*=*None*)

Retorna la ruta **PEP 3147** [https://peps.python.org/pep-3147/]/**PEP 488** [https://peps.python.org/pep-0488/] al archivo compilado por bytes asociado con la *path* de origen. Por ejemplo, si *path* es `/foo/bar/baz.py`, el valor de retorno sería `/foo/bar/__pycache__/baz.cpython-32.pyc` para Python 3.2. La cadena de caracteres `cpython-32` proviene de la etiqueta mágica actual (ver **`get_tag()`**; si **`sys.implementation.cache_tag`** no está definido, se lanzará **`NotImplementedError`**).

El parámetro *optimization* se utiliza para especificar el nivel de optimización del archivo de código de bytes. Una cadena de caracteres vacía no representa optimización, por lo que `/foo/bar/baz.py` con una *optimization* de `''` dará como resultado una ruta de código de bytes de `/foo/bar/__pycache__/baz.cpython-32.pyc`. `None` hace que se utilice el nivel de optimización del intérprete. Se usa la representación de cadena de caracteres de cualquier otro valor, por lo que `/foo/bar/baz.py` con una *optimization* de `2` conducirá a la ruta del código de bytes de `/foo/bar/__pycache__/baz.cpython-32.opt-2.pyc`. La representación de cadena de caracteres *optimization* solo puede ser alfanumérica, de lo contrario se lanza **`ValueError`**.

El parámetro *debug_override* está obsoleto y se puede usar para anular el valor del sistema para `__debug__`. Un valor `True` es el equivalente a establecer *optimization* en la cadena de caracteres vacía. Un valor `False` es lo mismo que establecer *optimization* en `1`. Si tanto *debug_override* como *optimization* no es `None`, entonces se lanza **`TypeError`**.

Added in version 3.4.

Distinto en la versión 3.5: Se agregó el parámetro *optimization* y el parámetro *debug_override* quedó obsoleto.

Distinto en la versión 3.6: Acepta un path-like object.

importlib.util.source_from_cache(*path*)

Dado el *path* a un nombre de archivo **PEP 3147** [https://peps.python.org/pep-3147/], retorna la ruta del archivo del código fuente asociado. Por ejemplo, si *path* es `/foo/bar/__pycache__/baz.cpython-32.pyc`, la ruta retornada sería `/foo/bar/baz.py`. *path* no necesita existir, sin embargo, si no se ajusta al formato **PEP 3147** [https://peps.python.org/pep-3147/] o **PEP 488** [https://peps.python.org/pep-0488/], se lanza un **`ValueError`**. Si **`sys.implementation.cache_tag`** no está definido, se lanza **`NotImplementedError`**.

Added in version 3.4.

Distinto en la versión 3.6: Acepta un path-like object.

importlib.util.decode_source(*source_bytes*)

Decodifica los bytes dados que representan el código fuente y los retorna como una cadena de caracteres con nuevas líneas universales (como lo requiere **`importlib.abc.InspectLoader.get_source()`**).

Added in version 3.4.

importlib.util.resolve_name(*name*, *package*)

Resuelve un nombre de módulo relativo a uno absoluto.

Si **name** no tiene puntos iniciales, entonces **name** simplemente se retorna. Esto permite el uso como `importlib.util.resolve_name('sys', __spec__.parent)` sin hacer una verificación para ver si se necesita el argumento **package**.

`ImportError` se lanza si **name** es un nombre de módulo relativo pero **package** es un valor falso (por ejemplo, `None` o la cadena de caracteres vacía). También se lanza **`ImportError`** si un nombre relativo escaparía del paquete que lo contiene (por ejemplo, solicitando `..bacon` desde el paquete `spam`).

Added in version 3.3.

Distinto en la versión 3.9: Para mejorar la coherencia con las declaraciones de importación, aumente **`ImportError`** en lugar de **`ValueError`** para intentos de importación relativa no válidos.

importlib.util.find_spec(*name*, *package=None*)

Busque el spec de un módulo, opcionalmente relativa al nombre del **paquete** especificado. Si el módulo está en **`sys.modules`**, se retorna `sys.modules[name].__spec__` (a menos que la especificación sea `None` o no esté configurada, en cuyo caso se lanza una excepción **`ValueError`**). De lo contrario, se realiza una búsqueda utilizando **`sys.meta_path`**. Se retorna `None` si no se encuentra ninguna especificación.

Si **name** es para un submódulo (contiene un punto), el módulo principal se importa automáticamente.

name y **package** funcionan igual que para **`import_module()`**.

Added in version 3.4.

Distinto en la versión 3.7: Raises **`ModuleNotFoundError`** instead of `AttributeError` if **package** is in fact not a package (i.e. lacks a `__path__` attribute).

importlib.util.module_from_spec(*spec*)

Cree un nuevo módulo basado en **spec** y `spec.loader.create_module`.

Si `spec.loader.create_module` no retorna `None`, no se restablecerán los atributos preexistentes. Además, no se lanzará `AttributeError` si se activa mientras se accede a **spec** o se establece un atributo en el módulo.

Esta función es preferible a usar `types.ModuleType` para crear un nuevo módulo ya que **spec** se usa para establecer tantos atributos de importación controlados en el módulo como sea posible.

Added in version 3.5.

importlib.util.spec_from_loader(*name*, *loader*, *, *origin=None*, *is_package=None*)

Una función de fábrica para crear una instancia de `ModuleSpec` basada en un cargador. Los parámetros tienen el mismo significado que para ModuleSpec. La función utiliza APIs disponibles de loader, tal como `InspectLoader.is_package()`, para completar cualquier información que falte en la especificación.

Added in version 3.4.

importlib.util.spec_from_file_location(*name*, *location*, *, *loader=None*, *submodule_search_locations=None*)

Una función de fábrica para crear una instancia de `ModuleSpec` basada en la ruta a un archivo. La información que falte se completará en la especificación mediante el uso de las API de loader y la implicación de que el módulo estará basado en archivos.

Added in version 3.4.

Distinto en la versión 3.6: Acepta un path-like object.

importlib.util.source_hash(*source_bytes*)

Retorna el hash de *source_bytes* como bytes. Un archivo `.pyc` basado en hash incrusta **`source_hash()`** del contenido del archivo fuente correspondiente en su encabezado.

Added in version 3.7.

importlib.util._incompatible_extension_module_restrictions(*, *disable_check*)

Un administrador de contexto que puede omitir temporalmente la comprobación de compatibilidad para módulos de extensión. De manera predeterminada, la comprobación está habilitada y fallará cuando se importe un módulo de inicialización de una sola fase en un subintérprete. También fallará para un módulo de inicialización de varias fases que no admita explícitamente una GIL por intérprete, cuando se importe en un intérprete con su propia GIL.

Tenga en cuenta que esta función está pensada para dar cabida a un caso inusual, que probablemente desaparecerá en algún momento. Es muy probable que esto no sea lo que estaba buscando.

Puede obtener el mismo efecto que esta función implementando la interfaz básica de init multifase (**PEP 489** [https://peps.python.org/pep-0489/]) y mintiendo acerca del soporte para múltiples intérpretes (o por intérprete GIL).

Advertencia

El uso de esta función para desactivar la comprobación puede provocar un comportamiento inesperado e incluso fallos. Solo debe utilizarse durante el desarrollo de módulos de extensión.

Added in version 3.12.

class importlib.util.LazyLoader(*loader*)

Una clase que pospone la ejecución del cargador de un módulo hasta que el módulo tiene acceso a un atributo.

Esta clase **solo** funciona con cargadores que definen `exec_module()`, ya que se requiere control sobre qué tipo de módulo se usa para el módulo. Por esas mismas razones, el método `create_module()` del cargador debe retornar `None` o un tipo para el cual se pueda mutar su atributo `__class__` junto con no usar slots. Finalmente, los módulos que sustituyen el objeto colocado en `sys.modules` no funcionarán ya que no hay forma de reemplazar correctamente las referencias del módulo en todo el intérprete de manera segura; `ValueError` se lanza si se detecta dicha sustitución.

Nota

Para proyectos donde el tiempo de inicio es crítico, esta clase permite minimizar potencialmente el costo de cargar un módulo si nunca se usa. Para proyectos en los que el tiempo de inicio no es esencial, el uso de esta clase se desaconseja **en gran medida** debido a que los mensajes de error creados durante la carga se posponen y, por lo tanto, ocurren fuera de contexto.

Added in version 3.5.

Distinto en la versión 3.6: Comenzó a llamar `create_module()`, eliminando la advertencia de compatibilidad para `importlib.machinery.BuiltinImporter` y `importlib.machinery.ExtensionFileLoader`.

classmethod factory(*loader*)

Un método de clase que retorna un elemento invocable que crea un cargador diferido. Está pensado para usarse en situaciones en las que el cargador se pasa por clase en lugar de por instancia.

```
        suffixes = importlib.machinery.SOURCE_SUFFIXES
        loader = importlib.machinery.SourceFileLoader
        lazy_loader = importlib.util.LazyLoader.factory
        finder = importlib.machinery.FileFinder(path,
```

Ejemplos

Importar programáticamente

Para importar un módulo mediante programación, use **`importlib.import_module()`**.

```
import importlib

itertools = importlib.import_module('itertools')
```

Comprobando si se puede importar un módulo

Si necesita averiguar si un módulo se puede importar sin realmente realizar la importación, entonces debe usar **`importlib.util.find_spec()`**.

Note que si `name` es un sub-módulo (contiene sólo un punto), **`importlib.util.find_spec()`** importará el módulo padre.

```
import importlib.util
import sys

# Con fines ilustrativos.
name = 'itertools'

if name in sys.modules:
    print(f"{name!r} already in sys.modules")
elif (spec := importlib.util.find_spec(name)) is not Nor
# Si eligió realizar la importación real ...
module = importlib.util.module_from_spec(spec)
sys.modules[name] = module
spec.loader.exec_module(module)
print(f"{name!r} ha sido importado")
```

```
else:
    print(f"no puede ser encontrado el módulo {name!r}")
```

Importar un archivo fuente directamente

Esta receta debe usarse con precaución: es una aproximación de una declaración de importación donde se especifica directamente la ruta del archivo, en lugar de buscar `sys.path`. Primero se deben considerar alternativas, como modificar **`sys.path`** cuando se requiere un módulo adecuado o usar `runpy.run_path()` cuando el espacio de nombres global resultante de la ejecución de un archivo Python es apropiado.

Para importar un archivo fuente de Python directamente desde una ruta, use la siguiente receta:

```
import importlib.util
import sys


def import_from_path(module_name, file_path):
    spec = importlib.util.spec_from_file_location(module
    module = importlib.util.module_from_spec(spec)
    sys.modules[module_name] = module
    spec.loader.exec_module(module)
    return module


# Sólo con fines ilustrativos. (el uso de `json` es arbi
import json
file_path = json.__file__
module_name = json.__name__

# Resultado similar a `import json`.
json = import_from_path(module_name, file_path)
```

Implementar importaciones diferidas

El ejemplo de abajo muestra cómo implementar importaciones diferidas:

```
>>> import importlib.util
>>> import sys
>>> def lazy_import(name):
...     spec = importlib.util.find_spec(name)
...     loader = importlib.util.LazyLoader(spec.loader)
...     spec.loader = loader
...     module = importlib.util.module_from_spec(spec)
...     sys.modules[name] = module
...     loader.exec_module(module)
...     return module
...
>>> lazy_typing = lazy_import("typing")
>>> #lazy_typing es un objeto de módulo real,
>>> #pero aún no está cargado en la memoria.
>>> lazy_typing.TYPE_CHECKING
False
```

Configurar un importador

Para personalizaciones profundas de importación, normalmente querrá implementar un importador. Esto significa gestionar tanto el buscador como el cargador. Para los buscadores hay dos opciones para elegir según sus necesidades: un meta path finder o un path entry finder. El primero es lo que pondría en **`sys.meta_path`** mientras que el segundo es lo que crea utilizando un path entry hook en **`sys.path_hooks`** que funciona con entradas **`sys.path`** para crear potencialmente un buscador. Este ejemplo le mostrará cómo registrar sus propios importadores para que la importación los utilice (para crear un importador para usted, lea la documentación de las clases apropiadas definidas dentro de este paquete):

```
import importlib.machinery
import sys

# Sólo con fines ilustrativos.
SpamMetaPathFinder = importlib.machinery.PathFinder
SpamPathEntryFinder = importlib.machinery.FileFinder
```

```
detalles_del_cargador = (importlib.machinery.SourceFileL
                                              importlib.ma

# Configuración de un meta buscador de rutas.
# Asegúrese de colocar el buscador en la ubicación adecu
# prioridad.
sys.meta_path.append(SpamMetaPathFinder)

# Configuración de un buscador de entradas de ruta.
# Asegúrese de colocar el gancho de ruta en la ubicación
# de prioridad.
sys.path_hooks.append(SpamPathEntryFinder.path_hook(deta
```

Aproximando `importlib.import_module()`

La importación en sí está implementada en código Python, lo que permite exponer la mayor parte de la maquinaria de importación a través de importlib. Lo siguiente ayuda a ilustrar las diversas API que importlib expone al proporcionar una implementación aproximada de `importlib.import_module()`:

```
import importlib.util
import sys

def import_module(name, package=None):
    """An approximate implementation of import."""
    absolute_name = importlib.util.resolve_name(name, pa
    try:
        return sys.modules[absolute_name]
    except KeyError:
        pass

    path = None
    if '.' in absolute_name:
        parent_name, _, child_name = absolute_name.rpart
        parent_module = import_module(parent_name)
        path = parent_module.__spec__.submodule_search_l
    for finder in sys.meta_path:
        spec = finder.find_spec(absolute_name, path)
    if spec is not None:
        break
else:
    msg = f'No module named {absolute_name!r}'
    raise ModuleNotFoundError(msg, name=absolute_nam
module = importlib.util.module_from_spec(spec)
sys.modules[absolute_name] = module
spec.loader.exec_module(module)
if path is not None:
    setattr(parent_module, child_name, module)
return module
```

`importlib.resources` – Package resource reading, opening and access

Source code: Lib/importlib/resources/_init_.py [https://github.com/python/cpython/tree/3.14/Lib/importlib/resources/_init_.py]

Added in version 3.7.

This module leverages Python's import system to provide access to *resources* within *packages*.

«Resources» are file-like resources associated with a module or package in Python. The resources may be contained directly in a package, within a subdirectory contained in that package, or adjacent to modules outside a package. Resources may be text or binary. As a result, a package's Python module sources (.py), compilation artifacts (pycache), and installation artifacts (like `reserved filenames` in directories) are technically de-facto resources of that package. In practice, however, resources are primarily those non-Python artifacts exposed specifically by the package author.

Resources can be opened or read in either binary or text mode.

Los recursos son más o menos similares a los archivos dentro de los directorios, aunque es importante tener en cuenta que esto es solo una metáfora. Los recursos y paquetes **no** deben existir como archivos y directorios físicos en el sistema de archivos: por ejemplo, un paquete y sus recursos se pueden importar desde un archivo zip usando `zipimport`.

Advertencia

`importlib.resources` follows the same security model as the built-in `open()` function. Passing untrusted inputs to the functions in this module is unsafe.

Nota

El soporte independiente de este módulo proporciona más información sobre el uso de importlib.resources [https://importlib-resources.readthedocs.io/en/latest/using.html] y migración de pkg_resources a importlib.resources [https://importlib-resources.readthedocs.io/en/latest/migration.html].

`Los loaders` que deseen soportar la lectura de recursos deben implementar un método `get_resource_reader(fullname)` según lo especificado por **`importlib.resources.abc.ResourceReader`**.

class importlib.resources.Anchor

Represents an anchor for resources, either a **`module object`** or a module name as a string. Defined as `Union[str, ModuleType]`.

importlib.resources.files(*anchor: Anchor | None = None*)

Returns a **`Traversable`** object representing the resource container (think directory) and its resources (think files). A Traversable may contain other containers (think subdirectories).

anchor is an optional **`Anchor`**. If the anchor is a package, resources are resolved from that package. If a module, resources are resolved adjacent to that module (in the same package or the package root). If the anchor is omitted, the caller's module is used.

Added in version 3.9.

Distinto en la versión 3.12: package parameter was renamed to *anchor*. *anchor* can now be a non-package module and if omitted will default to the caller's module. *package* is still

accepted for compatibility but will raise a `DeprecationWarning`. Consider passing the anchor positionally or using `importlib_resources >= 5.10` for a compatible interface on older Pythons.

importlib.resources.as_file(*traversable*)

Given a `Traversable` object representing a file or directory, typically from `importlib.resources.files()`, return a context manager for use in a `with` statement. The context manager provides a `pathlib.Path` object.

Exiting the context manager cleans up any temporary file or directory created when the resource was extracted from e.g. a zip file.

Use `as_file` when the Traversable methods (`read_text`, etc) are insufficient and an actual file or directory on the file system is required.

Added in version 3.9.

Distinto en la versión 3.12: Added support for *traversable* representing a directory.

Functional API

A set of simplified, backwards-compatible helpers is available. These allow common operations in a single function call.

For all the following functions:

- *anchor* is an `Anchor`, as in `files()`. Unlike in `files`, it may not be omitted.
- *path_names* are components of a resource's path name, relative to the anchor. For example, to get the text of resource named `info.txt`, use:

  ```
  importlib.resources.read_text(my_module, "info.txt"
  ```

Like **`Traversable.joinpath`**, The individual components should use forward slashes (/) as path separators. For example, the following are equivalent:

```
importlib.resources.read_binary(my_module, "pics/pa
importlib.resources.read_binary(my_module, "pics",
```

For backward compatibility reasons, functions that read text require an explicit *encoding* argument if multiple *path_names* are given. For example, to get the text of `info/chapter1.txt`, use:

```
importlib.resources.read_text(my_module, "info", "c
                              encoding='utf-8')
```

importlib.resources.open_binary(*anchor*, **path_names*)

Open the named resource for binary reading.

See the introduction for details on *anchor* and *path_names*.

This function returns a **`BinaryIO`** object, that is, a binary stream open for reading.

This function is roughly equivalent to:

```
files(anchor).joinpath(*path_names).open('rb')
```

Distinto en la versión 3.13: Multiple *path_names* are accepted.

importlib.resources.open_text(*anchor*, **path_names*, *encoding='utf-8'*, *errors='strict'*)

Open the named resource for text reading. By default, the contents are read as strict UTF-8.

See the introduction for details on *anchor* and *path_names*. *encoding* and *errors* have the same meaning as in built-in **`open()`**.

For backward compatibility reasons, the *encoding* argument must be given explicitly if there are multiple *path_names*. This limitation is scheduled to be removed in Python 3.15.

This function returns a **`TextIO`** object, that is, a text stream open for reading.

This function is roughly equivalent to:

```
files(anchor).joinpath(*path_names).open('r', encod
```

Distinto en la versión 3.13: Multiple *path_names* are accepted. *encoding* and *errors* must be given as keyword arguments.

importlib.resources.read_binary(*anchor*, **path_names*)

Read and return the contents of the named resource as **`bytes`**.

See the introduction for details on *anchor* and *path_names*.

This function is roughly equivalent to:

```
files(anchor).joinpath(*path_names).read_bytes()
```

Distinto en la versión 3.13: Multiple *path_names* are accepted.

importlib.resources.read_text(*anchor*, **path_names*, *encoding='utf-8'*, *errors='strict'*)

Read and return the contents of the named resource as **`str`**. By default, the contents are read as strict UTF-8.

See the introduction for details on *anchor* and *path_names*. *encoding* and *errors* have the same meaning as in built-in **`open()`**.

For backward compatibility reasons, the *encoding* argument must be given explicitly if there are multiple *path_names*. This limitation is scheduled to be removed in Python 3.15.

This function is roughly equivalent to:

```
files(anchor).joinpath(*path_names).read_text(encod
```

Distinto en la versión 3.13: Multiple *path_names* are accepted. *encoding* and *errors* must be given as keyword arguments.

importlib.resources.path(*anchor*, **path_names*)

Provides the path to the *resource* as an actual file system path. This function returns a context manager for use in a `with` statement. The context manager provides a `pathlib.Path` object.

Exiting the context manager cleans up any temporary files created, e.g. when the resource needs to be extracted from a zip file.

For example, the `stat()` method requires an actual file system path; it can be used like this:

```
with importlib.resources.path(anchor, "resource.txt
    result = fspath.stat()
```

See the introduction for details on *anchor* and *path_names*.

This function is roughly equivalent to:

```
as_file(files(anchor).joinpath(*path_names))
```

Distinto en la versión 3.13: Multiple *path_names* are accepted. *encoding* and *errors* must be given as keyword arguments.

importlib.resources.is_resource(*anchor*, **path_names*)

Return `True` if the named resource exists, otherwise `False`. This function does not consider directories to be resources.

See the introduction for details on *anchor* and *path_names*.

This function is roughly equivalent to:

```
files(anchor).joinpath(*path_names).is_file()
```

Distinto en la versión 3.13: Multiple *path_names* are accepted.

importlib.resources.contents(*anchor*, **path_names*)

Return an iterable over the named items within the package or path. The iterable returns names of resources (e.g. files) and non-resources (e.g. directories) as **`str`**. The iterable does not recurse into subdirectories.

See the introduction for details on *anchor* and *path_names*.

This function is roughly equivalent to:

```
for resource in files(anchor).joinpath(*path_names)
    yield resource.name
```

Obsoleto desde la versión 3.11: Prefer `iterdir()` as above, which offers more control over the results and richer functionality.

`importlib.resources.abc` – Abstract base classes for resources

Source code: Lib/importlib/resources/abc.py [https://github.com/python/cpython/tree/3.14/Lib/importlib/resources/abc.py]

Added in version 3.11.

class importlib.resources.abc.ResourceReader

Superseded by TraversableResources

Un abstract base class para proporcionar la capacidad de leer *resources*.

From the perspective of this ABC, a *resource* is a binary artifact that is shipped within a package. Typically this is something like a data file that lives next to the `__init__.py` file of the package. The purpose of this class is to help abstract out the accessing of such data files so that it does not matter if the package and its data file(s) are stored e.g. in a zip file versus on the file system.

Para cualquiera de los métodos de esta clase, se espera que un argumento *resource* sea un path-like object que represente conceptualmente solo un nombre de archivo. Esto significa que no se deben incluir rutas de subdirectorios en el argumento *resource*. Esto se debe a que la ubicación del paquete para el que se encuentra el lector actúa como el «directorio». Por lo tanto, la metáfora de directorios y nombres de archivos es paquetes y recursos, respectivamente. Esta es también la razón por la que se espera que las instancias de esta clase se correlacionen directamente con un paquete específico (en lugar de representar potencialmente varios paquetes o un módulo).

Se espera que los cargadores que deseen admitir la lectura de recursos proporcionen un método llamado `get_resource_reader(fullname)` que retorna un objeto que implementa la interfaz de este ABC. Si el módulo especificado por nombre completo no es un paquete, este método debería retornar **`None`**. Un objeto compatible con este ABC solo debe retornarse cuando el módulo especificado es un paquete.

Obsoleto desde la versión 3.12: Use **`importlib.resources.abc.TraversableResources`** instead.

abstractmethod open_resource(*resource*)

Retorna un file-like object abierto para la lectura binaria del *resource*.

Si no se puede encontrar el recurso, se genera **`FileNotFoundError`**.

abstractmethod resource_path(*resource*)

Retorna la ruta del sistema de archivos al *resource*.

Si el recurso no existe concretamente en el sistema de archivos, lanza **`FileNotFoundError`**.

abstractmethod is_resource(*path*)

Returns `True` if the named *path* is considered a resource. **`FileNotFoundError`** is raised if *path* does not exist.

Distinto en la versión 3.10: The argument *name* was renamed to *path*.

abstractmethod contents()

Retorna un iterable de cadenas sobre el contenido del paquete. Tenga en cuenta que no se requiere que todos los nombres devueltos por el iterador sean recursos reales, p. es aceptable devolver nombres para los que `is_resource()` sería falso.

Permitir que se retornen nombres que no son de recursos es permitir situaciones en las que se conoce a priori cómo se almacenan un paquete y sus recursos y los nombres que no son de recursos serían útiles. Por ejemplo, se permite devolver nombres de subdirectorios para que cuando se sepa que el paquete y los recursos están almacenados en el sistema de archivos, esos nombres de subdirectorios se puedan usar directamente.

El método abstracto retorna un iterable sin elementos.

class importlib.resources.abc.Traversable

An object with a subset of `pathlib.Path` methods suitable for traversing directories and opening files.

For a representation of the object on the file-system, use `importlib.resources.as_file()`.

name

Resumen. El nombre base de este objeto sin ninguna referencia principal.

abstractmethod iterdir()

Produce generador iterador de objetos Traversable en self.

abstractmethod is_dir()

Return `True` if self is a directory.

abstractmethod is_file()

Return `True` if self is a file.

abstractmethod joinpath(**pathsegments*)

Traverse directories according to *pathsegments* and return the result as **`Traversable`**.

Each *pathsegments* argument may contain multiple names separated by forward slashes (`/`, `posixpath.sep`). For example, the following are equivalent:

```
files.joinpath('subdir', 'subsuddir', 'file.txt
files.joinpath('subdir/subsuddir/file.txt')
```

Note that some **`Traversable`** implementations might not be updated to the latest version of the protocol. For compatibility with such implementations, provide a single argument without path separators to each call to `joinpath`. For example:

```
files.joinpath('subdir').joinpath('subsubdir').
```

Distinto en la versión 3.11: `joinpath` accepts multiple *pathsegments*, and these segments may contain forward slashes as path separators. Previously, only a single *child* argument was accepted.

abstractmethod __truediv__(*child*)

Return Traversable child in self. Equivalent to `joinpath(child)`.

abstractmethod open(*mode='r'*, **args*, ***kwargs*)

mode puede ser "r" o "rb" para abrir como texto o binario. Retorna un identificador adecuado para la lectura (igual que **`pathlib.Path.open`**).

When opening as text, accepts encoding parameters such as those accepted by **`io.TextIOWrapper`**.

read_bytes()

Lee el contenido de self como bytes.

read_text(*encoding=None*)

Lee el contenido de self como texto.

class importlib.resources.abc.TraversableResources

An abstract base class for resource readers capable of serving the `importlib.resources.files()` interface. Subclasses `ResourceReader` and provides concrete implementations of the `ResourceReader`'s abstract methods. Therefore, any loader supplying `TraversableResources` also supplies `ResourceReader`.

Se espera que los cargadores que deseen admitir la lectura de recursos implementen esta interfaz.

abstractmethod files()

Retorna un objeto `importlib.resources.abc.Traversable` para el paquete cargado.

`importlib.metadata` – Acceso a los metadatos de los paquetes

Added in version 3.8.

Distinto en la versión 3.10: `importlib.metadata` ya no es provisional.

Source code: Lib/importlib/metadata/__init__.py [https://github.com/python/cpython/tree/3.14/Lib/importlib/metadata/__init__.py]

`importlib.metadata` is a library that provides access to the metadata of an installed Distribution Package [https://packaging.python.org/en/latest/glossary/#term-Distribution-Package], such as its entry points or its top-level names (Import Package [https://packaging.python.org/en/latest/glossary/#term-Import-Package]s, modules, if any). Built in part on Python's import system, this library provides the entry point and metadata APIs that were previously exposed by the now-removed `pkg_resources` package. Along with **`importlib.resources`**, it supersedes `pkg_resources`.

`importlib.metadata` operates on third-party *distribution packages* installed into Python's `site-packages` directory via tools such as pip [https://pypi.org/project/pip/]. Specifically, it works with distributions with discoverable `dist-info` or `egg-info` directories, and metadata defined by the Core metadata specifications [https://packaging.python.org/en/latest/specifications/core-metadata/#core-metadata].

Importante

These are *not* necessarily equivalent to or correspond 1:1 with the top-level *import package* names that can be imported inside Python code. One *distribution package* can contain multiple *import packages* (and single modules), and one top-level *import package* may map to multiple *distribution packages* if it is a namespace package. You can use packages_distributions() to get a mapping between them.

By default, distribution metadata can live on the file system or in zip archives on `sys.path`. Through an extension mechanism, the metadata can live almost anywhere.

Ver también

https://importlib-metadata.readthedocs.io/
La documentación de `importlib_metadata`, que proporciona un backport de `importlib.metadata`. Esto incluye una Referencia API [https://importlib-metadata.readthedocs.io/en/latest/api.html] para las clases y funciones de este módulo, así como una Guía de migración [https://importlib-metadata.readthedocs.io/en/latest/migration.html] para los usuarios existentes de `pkg_resources`.

Descripción general

Let's say you wanted to get the version string for a Distribution Package [https://packaging.python.org/en/latest/glossary/#term-Distribution-Package] you've installed using `pip`. We start by creating a virtual environment and installing something into it:

```
$ python -m venv example
$ source example/bin/activate
(example) $ python -m pip install wheel
```

Se puede obtener la cadena de versión para `wheel` ejecutando lo siguiente:

```
(example) $ python
>>> from importlib.metadata import version
>>> version('wheel')
'0.32.3'
```

You can also get a collection of entry points selectable by properties of the EntryPoint (typically "group" or "name"), such as `console_scripts`, `distutils.commands` and others. Each group contains a collection of EntryPoint objects.

Se pueden obtener los metadatos para una distribución:

```
>>> list(metadata('wheel'))
['Metadata-Version', 'Name', 'Version', 'Summary', 'Home
```

También se puede obtener el número de versión de una distribución, enumerar sus archivos constituyentes y obtener una lista de los Requerimientos de la distribución de la distribución.

exception importlib.metadata.PackageNotFoundError

Subclass of **`ModuleNotFoundError`** raised by several functions in this module when queried for a distribution package which is not installed in the current Python environment.

API funcional

Este paquete provee la siguiente funcionalidad a través de su API pública.

Puntos de entrada

importlib.metadata.entry_points(***select_params*)

Returns a **`EntryPoints`** instance describing entry points for the current environment. Any given keyword parameters are passed to the **`select()`** method for comparison to the attributes of the individual entry point definitions.

Note: it is not currently possible to query for entry points based on their **`EntryPoint.dist`** attribute (as different **`Distribution`** instances do not currently compare equal, even if they have the same attributes)

class importlib.metadata.EntryPoints

Details of a collection of installed entry points.

Also provides a `.groups` attribute that reports all identified entry point groups, and a `.names` attribute that reports all identified entry point names.

class importlib.metadata.EntryPoint

Details of an installed entry point.

Each **`EntryPoint`** instance has `.name`, `.group`, and `.value` attributes and a `.load()` method to resolve the value. There are also `.module`, `.attr`, and `.extras` attributes for getting the components of the `.value` attribute, and `.dist` for obtaining information regarding the distribution package that provides the entry point.

Consultar todos los puntos de entrada:

```
>>> eps = entry_points()
```

The **`entry_points()`** function returns a **`EntryPoints`** object, a collection of all **`EntryPoint`** objects with `names` and `groups` attributes for convenience:

```
>>> sorted(eps.groups)
['console_scripts', 'distutils.commands', 'distutils.set
```

`EntryPoints` has a **`select()`** method to select entry points matching specific properties. Select entry points in the `console_scripts` group:

```
>>> scripts = eps.select(group='console_scripts')
```

Equivalently, since **`entry_points()`** passes keyword arguments through to select:

```
>>> scripts = entry_points(group='console_scripts')
```

Elige un script específico llamado «wheel» (que se encuentra en el proyecto *wheel*):

```
>>> 'wheel' in scripts.names
True
```

```
>>> wheel = scripts['wheel']
```

De manera equivalente, consulta por ese punto de entrada durante la selección:

```
>>> (wheel,) = entry_points(group='console_scripts', nam
>>> (wheel,) = entry_points().select(group='console_scri
```

Inspeccionar el punto de entrada resuelto:

```
>>> wheel
EntryPoint(name='wheel', value='wheel.cli:main', group='
>>> wheel.module
'wheel.cli'
>>> wheel.attr
'main'
>>> wheel.extras
[]
>>> main = wheel.load()
>>> main
<function main at 0x103528488>
```

The `group` and `name` are arbitrary values defined by the package author and usually a client will wish to resolve all entry points for a particular group. Read the setuptools docs [https://setuptools.pypa.io/en/latest/userguide/entry_point.html] for more information on entry points, their definition, and usage.

Distinto en la versión 3.12: The «selectable» entry points were introduced in `importlib_metadata` 3.6 and Python 3.10. Prior to those changes, `entry_points` accepted no parameters and always returned a dictionary of entry points, keyed by group. With `importlib_metadata` 5.0 and Python 3.12, `entry_points` always returns an `EntryPoints` object. See backports.entry_points_selectable [https://pypi.org/project/backports.entry_points_selectable/] for compatibility options.

Distinto en la versión 3.13: `EntryPoint` objects no longer present a tuple-like interface (`__getitem__()`).

Metadatos de distribución

importlib.metadata.metadata(*distribution_name*)

Return the distribution metadata corresponding to the named distribution package as a **`PackageMetadata`** instance.

Raises **`PackageNotFoundError`** if the named distribution package is not installed in the current Python environment.

class importlib.metadata.PackageMetadata

A concrete implementation of the PackageMetadata protocol [https://importlib-metadata.readthedocs.io/en/latest/api.html#importlib_metadata.PackageMetadata].

In addition to providing the defined protocol methods and attributes, subscripting the instance is equivalent to calling the **`get()`** method.

Every Distribution Package [https://packaging.python.org/en/latest/glossary/#term-Distribution-Package] includes some metadata, which you can extract using the **`metadata()`** function:

```
>>> wheel_metadata = metadata('wheel')
```

The keys of the returned data structure name the metadata keywords, and the values are returned unparsed from the distribution metadata:

```
>>> wheel_metadata['Requires-Python']
'>=2.7, !=3.0.*, !=3.1.*, !=3.2.*, !=3.3.*'
```

`PackageMetadata` also presents a **`json`** attribute that returns all the metadata in a JSON-compatible form per **PEP 566** [https://peps.python.org/pep-0566/]:

```
>>> wheel_metadata.json['requires_python']
'>=2.7, !=3.0.*, !=3.1.*, !=3.2.*, !=3.3.*'
```

The full set of available metadata is not described here. See the PyPA Core metadata specification [https://packaging.python.org/en/latest/

specifications/core-metadata/#core-metadata] for additional details.

Distinto en la versión 3.10: La `Descripción` ahora se incluye en los metadatos cuando se presenta a través de la carga útil. Se han eliminado los caracteres de continuación de línea.

El atributo `json` fue añadido.

Versiones de distribución

importlib.metadata.version(*distribution_name*)

Return the installed distribution package version [https://packaging.python.org/en/latest/specifications/core-metadata/#version] for the named distribution package.

Raises **`PackageNotFoundError`** if the named distribution package is not installed in the current Python environment.

The **`version()`** function is the quickest way to get a Distribution Package [https://packaging.python.org/en/latest/glossary/#term-Distribution-Package]'s version number, as a string:

```
>>> version('wheel')
'0.32.3'
```

Archivos de distribución

importlib.metadata.files(*distribution_name*)

Return the full set of files contained within the named distribution package.

Raises **`PackageNotFoundError`** if the named distribution package is not installed in the current Python environment.

Returns **`None`** if the distribution is found but the installation database records reporting the files associated with the distribution package are missing.

class importlib.metadata.PackagePath

A **`pathlib.PurePath`** derived object with additional `dist`, `size`, and `hash` properties corresponding to the distribution package's installation metadata for that file.

The **`files()`** function takes a Distribution Package [https://packaging.python.org/en/latest/glossary/#term-Distribution-Package] name and returns all of the files installed by this distribution. Each file is reported as a **`PackagePath`** instance. For example:

```
>>> util = [p for p in files('wheel') if 'util.py' in st
>>> util
PackagePath('wheel/util.py')
>>> util.size
859
>>> util.dist
<importlib.metadata._hooks.PathDistribution object at 0x
>>> util.hash
<FileHash mode: sha256 value: bYkw5oMccfazVCoYQwKkkemoVy
```

Una vez que se tiene el archivo, también se puede leer su contenido:

```
>>> print(util.read_text())
import base64
import sys
...
def as_bytes(s):
    if isinstance(s, text_type):
        return s.encode('utf-8')
    return s
```

You can also use the **`locate()`** method to get the absolute path to the file:

```
>>> util.locate()
PosixPath('/home/gustav/example/lib/site-packages/wheel/
```

In the case where the metadata file listing files (`RECORD` or `SOURCES.txt`) is missing, **`files()`** will return **`None`**. The caller may wish to wrap calls to **`files()`** in always_iterable [https://more-itertools.readthedocs.io/en/stable/api.html#more_itertools.always_iterable] or

otherwise guard against this condition if the target distribution is not known to have the metadata present.

Requerimientos de la distribución

importlib.metadata.requires(*distribution_name*)

> Return the declared dependency specifiers for the named distribution package.
>
> Raises `PackageNotFoundError` if the named distribution package is not installed in the current Python environment.

To get the full set of requirements for a Distribution Package [https://packaging.python.org/en/latest/glossary/#term-Distribution-Package], use the **`requires()`** function:

```
>>> requires('wheel')
["pytest (>=3.0.0) ; extra == 'test'", "pytest-cov ; ext
```

Mapeo de paquetes de importación a distribución

importlib.metadata.packages_distributions()

> Return a mapping from the top level module and import package names found via `sys.meta_path` to the names of the distribution packages (if any) that provide the corresponding files.
>
> To allow for namespace packages (which may have members provided by multiple distribution packages), each top level import name maps to a list of distribution names rather than mapping directly to a single name.

Un método práctico para resolver el nombre del Paquete de distribución [https://packaging.python.org/en/latest/glossary/#term-Distribution-Package] (o nombres, en el caso de un paquete de espacio de nombres) que proporciona cada módulo Python de nivel superior importable o Paquete de importación [https://packaging.python.org/en/latest/glossary/#term-Import-Package]:

```
>>> packages_distributions()
{'importlib_metadata': ['importlib-metadata'], 'yaml': [
```

Algunas instalaciones editables, no suministran nombres de nivel superior [https://github.com/pypa/packaging-problems/issues/609], por lo que esta función no es fiable con dichas instalaciones.

Added in version 3.10.

Distribuciones

importlib.metadata.distribution(*distribution_name*)

Return a `Distribution` instance describing the named distribution package.

Raises `PackageNotFoundError` if the named distribution package is not installed in the current Python environment.

class importlib.metadata.Distribution

Details of an installed distribution package.

Note: different `Distribution` instances do not currently compare equal, even if they relate to the same installed distribution and accordingly have the same attributes.

While the module level API described above is the most common and convenient usage, you can get all of that information from the `Distribution` class. `Distribution` is an abstract object that represents the metadata for a Python Distribution Package [https://packaging.python.org/en/latest/glossary/#term-Distribution-Package]. You can get the concrete `Distribution` subclass instance for an installed distribution package by calling the `distribution()` function:

```
>>> from importlib.metadata import distribution
>>> dist = distribution('wheel')
>>> type(dist)
<class 'importlib.metadata.PathDistribution'>
```

Thus, an alternative way to get the version number is through the

`Distribution` instance:

```
>>> dist.version
'0.32.3'
```

There are all kinds of additional metadata available on **`Distribution`** instances:

```
>>> dist.metadata['Requires-Python']
'>=2.7, !=3.0.*, !=3.1.*, !=3.2.*, !=3.3.*'
>>> dist.metadata['License']
'MIT'
```

For editable packages, an `origin` property may present **PEP 610** [https://peps.python.org/pep-0610/] metadata:

```
>>> dist.origin.url
'file:///path/to/wheel-0.32.3.editable-py3-none-any.whl'
```

The full set of available metadata is not described here. See the PyPA Core metadata specification [https://packaging.python.org/en/latest/specifications/core-metadata/#core-metadata] for additional details.

Added in version 3.13: The `.origin` property was added.

Distribution Discovery

Por defecto, este paquete proporciona soporte incorporado para el descubrimiento de metadatos para el sistema de archivos y archivos zip Distribution Package [https://packaging.python.org/en/latest/glossary/#term-Distribution-Package]s. Este buscador de metadatos busca por defecto `sys.path`, pero varía ligeramente en cómo interpreta esos valores respecto a cómo lo hace otra maquinaria de importación. En particular:

- `importlib.metadata` does not honor **`bytes`** objects on `sys.path`.
- `importlib.metadata` respetará incidentalmente los objetos **`pathlib.Path`**` en `sys.path` aunque tales valores serán ignorados para las importaciones.

Implementing Custom Providers

`importlib.metadata` address two API surfaces, one for *consumers* and another for *providers*. Most users are consumers, consuming metadata provided by the packages. There are other use-cases, however, where users wish to expose metadata through some other mechanism, such as alongside a custom importer. Such a use case calls for a *custom provider*.

Because Distribution Package [https://packaging.python.org/en/latest/glossary/#term-Distribution-Package] metadata is not available through **`sys.path`** searches, or package loaders directly, the metadata for a distribution is found through import system finders. To find a distribution package’s metadata, `importlib.metadata` queries the list of meta path finders on **`sys.meta_path`**.

The implementation has hooks integrated into the `PathFinder`, serving metadata for distribution packages found on the file system.

La clase abstracta **`importlib.abc.MetaPathFinder`** define la interfaz que se espera de los buscadores por el sistema de importación de Python. `importlib.metadata` amplía este protocolo buscando una `find_distributions` opcional invocable en los buscadores desde **`sys.meta_path`** y presenta esta interfaz extendida como la clase base abstracta `DistributionFinder`, que define este método abstracto:

```
@abc.abstractmethod
def find_distributions(context=DistributionFinder.Contex
    """Return an iterable of all Distribution instances
    loading the metadata for packages for the indicated
    """
```

The `DistributionFinder.Context` object provides `.path` and `.name` properties indicating the path to search and name to match and may supply other relevant context sought by the consumer.

In practice, to support finding distribution package metadata in locations other than the file system, subclass `Distribution` and implement the abstract methods. Then from a custom finder, return instances of this derived `Distribution` in the `find_distributions()` method.

Example

Imagine a custom finder that loads Python modules from a database:

```
class DatabaseImporter(importlib.abc.MetaPathFinder):
    def __init__(self, db):
        self.db = db

    def find_spec(self, fullname, target=None) -> Module
        return self.db.spec_from_name(fullname)

sys.meta_path.append(DatabaseImporter(connect_db(...)))
```

That importer now presumably provides importable modules from a database, but it provides no metadata or entry points. For this custom importer to provide metadata, it would also need to implement `DistributionFinder`:

```
from importlib.metadata import DistributionFinder

class DatabaseImporter(DistributionFinder):
    ...

    def find_distributions(self, context=DistributionFi
        query = dict(name=context.name) if context.name
        for dist_record in self.db.query_distributions(
            yield DatabaseDistribution(dist_record)
```

In this way, `query_distributions` would return records for each distribution served by the database matching the query. For example, if `requests-1.0` is in the database, `find_distributions` would yield a `DatabaseDistribution` for `Context(name='requests')` or `Context(name=None)`.

For the sake of simplicity, this example ignores `context.path`.

The `path` attribute defaults to `sys.path` and is the set of import paths to be considered in the search. A `DatabaseImporter` could potentially function without any concern for a search path. Assuming the importer does no partitioning, the «path» would be irrelevant. In order to illustrate the purpose of `path`, the example would need to illustrate a more complex `DatabaseImporter` whose behavior varied depending on `sys.path/PYTHONPATH`. In that case, the `find_distributions` should honor the `context.path` and only yield `Distribution`s pertinent to that path.

`DatabaseDistribution`, then, would look something like:

```
class DatabaseDistribution(importlib.metadata.Distributi
    def __init__(self, record):
        self.record = record

    def read_text(self, filename):
        """
        Read a file like "METADATA" for the current dist
        """
        if filename == "METADATA":
            return f"""Name: {self.record.name}
Version: {self.record.version}
"""
        if filename == "entry_points.txt":
            return "\n".join(
              f"""[{ep.group}]\n{ep.name}={ep.value}"""
              for ep in self.record.entry_points)

    def locate_file(self, path):
        raise RuntimeError("This distribution has no fil
```

This basic implementation should provide metadata and entry points for packages served by the `DatabaseImporter`, assuming that the `record` supplies suitable `.name`, `.version`, and `.entry_points` attributes.

The `DatabaseDistribution` may also provide other metadata files, like `RECORD` (required for `Distribution.files`) or override the implementation of `Distribution.files`. See the source for more inspiration.

La inicialización de la ruta de búsqueda de módulo de `sys.path`

Una ruta de búsqueda de módulo se inicializa cuando se inicia Python. Se puede acceder a esta ruta de búsqueda de módulo en `sys.path`.

La primera entrada en la ruta de búsqueda de módulo es el directorio que contiene el script de entrada, si lo hay. De lo contrario, la primera entrada es el directorio actual, que es el caso cuando se ejecuta el shell interactivo, un comando `-c` o un módulo `-m`.

La variable de entorno `PYTHONPATH` se utiliza a menudo para añadir directorios a la ruta de búsqueda. Si se encuentra esta variable de entorno, su contenido se añade a la ruta de búsqueda del módulo.

Nota

`PYTHONPATH` afectará a todas las versiones/entornos de Python instalados. Tenga cuidado al establecer esto en su perfil de shell o en las variables de entorno globales. El módulo `site` ofrece técnicas más suavizadas como se menciona a continuación.

Los siguientes elementos que se añaden son los directorios que contienen los módulos estándar de Python, así como los extension module de los que dependen estos módulos. Los módulos de extensión son archivos `.pyd` en Windows y archivos `.so` en otras plataformas. El directorio con los módulos Python independientes de la plataforma se llama `prefix`. El directorio con los módulos de extensión se llama `exec_prefix`.

La variable de entorno **`PYTHONHOME`** puede utilizarse para establecer las ubicaciones `prefix` y `exec_prefix`. De lo contrario, estos directorios se encuentran utilizando el ejecutable de Python como punto de partida y luego buscando varios archivos y directorios «de referencia». Tenga en cuenta que cualquier enlace simbólico se sigue, por lo que la ubicación real del ejecutable de Python se utiliza como punto de partida de la búsqueda. La ubicación del ejecutable de Python se llama `home`.

Una vez determinado `home`, el directorio `prefix` se encuentra buscando primero `python`*`majorversionminorversion`*`.zip` (`python311.zip`). En Windows el archivo zip se busca en `home` y en Unix se espera que el archivo esté en `lib`. Tenga en cuenta que la ubicación esperada del archivo zip se añade a la ruta de búsqueda del módulo incluso si el archivo no existe. Si no se encuentra ningún archivo, Python en Windows continuará la búsqueda de `prefix` buscando en `Lib\os.py`. Python en Unix buscará `lib/python`*`majorversion`*`.`*`minorversion`*`/os.py` (`lib/python3.11/os.py`). En Windows `prefix` y `prefix_exec` son los mismos, sin embargo en otras plataformas se busca `lib/python`*`majorversion`*`.`*`minorversion`*`/lib-dynload` (`lib/python3.11/lib-dynload`) y se utiliza como ancla para `prefix_exec`. En algunas plataformas `lib` puede ser `lib64` u otro valor, ver **`sys.platlibdir`** y **`PYTHONPLATLIBDIR`**.

Once found, `prefix` and `exec_prefix` are available at **`sys.base_prefix`** and **`sys.base_exec_prefix`** respectively.

If **`PYTHONHOME`** is not set, and a `pyvenv.cfg` file is found alongside the main executable, or in its parent directory, **`sys.prefix`** and **`sys.exec_prefix`** get set to the directory containing `pyvenv.cfg`, otherwise they are set to the same value as **`sys.base_prefix`** and **`sys.base_exec_prefix`**, respectively. This is used by Virtual Environments.

Finalmente, el módulo **`site`** se procesa y los directorios `site-packages` se añaden a la ruta de búsqueda de módulo. Una forma común de personalizar la ruta de búsqueda es crear módulos **`sitecustomize`** o **`usercustomize`** como se describe en la documentación del módulo **`site`**.

Nota

Ciertas opciones de la línea de comandos pueden afectar aún más los cálculos de ruta. Vea `-E`, `-I`, `-s` y `-S` para más detalles.

Distinto en la versión 3.14: `sys.prefix` and `sys.exec_prefix` are now set to the `pyvenv.cfg` directory during the path initialization. This was previously done by `site`, therefore affected by `-S`.

Virtual Environments

Virtual environments place a `pyvenv.cfg` file in their prefix, which causes `sys.prefix` and `sys.exec_prefix` to point to them, instead of the base installation.

The `prefix` and `exec_prefix` values of the base installation are available at `sys.base_prefix` and `sys.base_exec_prefix`.

As well as being used as a marker to identify virtual environments, `pyvenv.cfg` may also be used to configure the `site` initialization. Please refer to `site`'s virtual environments documentation.

Nota

`PYTHONHOME` overrides the `pyvenv.cfg` detection.

Nota

There are other ways how «virtual environments» could be implemented, this documentation refers implementations based on the `pyvenv.cfg` mechanism, such as `venv`. Most virtual environment implementations follow the model set by `venv`, but there may be exotic implementations that diverge from it.

Archivos _pth

Para anular completamente `sys.path` cree un archivo `._pth` con el mismo nombre que la biblioteca compartida o el ejecutable (`python._pth` o `python311._pth`). La ruta de la biblioteca compartida se conoce siempre en Windows, pero puede no estar disponible en otras plataformas. En el archivo `._pth` especifique una línea por cada ruta a añadir a **`sys.path`**`. El archivo basado en el nombre de la biblioteca compartida anula el basado en el ejecutable, lo que permite restringir las rutas para cualquier programa en tiempo de ejecución si se desea.

Cuando el archivo existe, se ignoran todas las variables de registro y de entorno, se habilita el modo aislado y no se importa `site` a menos que una línea del archivo especifique `import site`. Las rutas en blanco y las líneas que comienzan con `#` son ignoradas. Cada ruta puede ser absoluta o relativa a la ubicación del fichero. No se permiten declaraciones de importación distintas de `site`, y no se puede especificar código arbitrario.

Tenga en cuenta que los archivos `.pth` (sin guión bajo inicial) serán procesados normalmente por el módulo `site` cuando se haya especificado `import site`.

Python embebido

If Python is embedded within another application `Py_InitializeFromConfig()` and the `PyConfig` structure can be used to initialize Python. The path specific details are described at Configuración de la ruta de Python.

Ver también

- Encontrar módulos para las notas detalladas de Windows.
- Uso de Python en plataformas Unix para los detalles de Unix.

Servicios del lenguaje Python

Python proporciona una serie de módulos para ayudar a trabajar con el lenguaje Python. Estos módulos admiten tokenización, análisis, análisis sintáctico, desensamblado de código de bytes, entre otras funciones.

Estos módulos incluyen:

- **`ast`** — Abstract syntax trees
  - Abstract grammar
  - Clases nodo
    - Nodos raíz
    - Literales
    - Variables
    - Expresiones
      - Subindexado
      - Comprensiones
    - Declaraciones
      - Importaciones
    - Control de flujo
    - La coincidencia de patrones
    - Type annotations
    - Tipos de parámetro
    - Definiciones de función y clase
    - Async y await
  - **`ast`** helpers
  - Compiler flags
  - Command-line usage
- **`symtable`** — Access to the compiler’s symbol tables

    - Generando tablas de símbolos
    - Examinando la tabla de símbolos
    - Command-Line Usage

- **`token`** — Constants used with Python parse trees
- **`keyword`** — Testing for Python keywords
- **`tokenize`** — Tokenizer for Python source
    - Convirtiendo la entrada en *tokens*
    - Uso como línea de comandos
    - Ejemplos
- **`tabnanny`** — Detection of ambiguous indentation
- **`pyclbr`** — Python module browser support
    - Objetos Function
    - Objetos Class
- **`py_compile`** — Compile Python source files
    - Interfaz de línea de comandos
- **`compileall`** — Byte-compile Python libraries
    - Uso de la línea de comandos
    - Funciones públicas
- **`dis`** — Desensamblador para bytecode de Python
    - Interfaz de línea de comandos
    - Análisis de bytecode
    - Funciones de análisis
    - Instrucciones bytecode de Python
    - Colecciones opcode
- **`pickletools`** — Tools for pickle developers
    - Command-line usage
        - Command-line options
    - Programmatic interface

`ast` — Abstract syntax trees

Código fuente: Lib/ast.py [https://github.com/python/cpython/tree/3.14/Lib/ast.py]

The **`ast`** module helps Python applications to process trees of the Python abstract syntax grammar. The abstract syntax itself might change with each Python release; this module helps to find out programmatically what the current grammar looks like.

Se puede generar un árbol de sintaxis abstracta pasando **`ast.PyCF_ONLY_AST`** como un indicador de la función incorporada **`compile()`**, o usando el ayudante **`parse()`** provisto en este módulo. El resultado será un árbol de objetos cuyas clases todas heredan de **`ast.AST`**. Se puede compilar un árbol de sintaxis abstracta en un objeto de código Python utilizando la función incorporada **`compile()`**.

Abstract grammar

La gramática abstracta se define actualmente de la siguiente manera:

```
-- ASDL's 4 builtin types are:
-- identifier, int, string, constant

module Python
{
    mod = Module(stmt* body, type_ignore* type_ignores)
        | Interactive(stmt* body)
        | Expression(expr body)
        | FunctionType(expr* argtypes, expr returns)

    stmt = FunctionDef(identifier name, arguments args,
                       stmt* body, expr* decorator_list,
             string? type_comment, type_param*
| AsyncFunctionDef(identifier name, arguments
                   stmt* body, expr* decorator
                   string? type_comment, type_

| ClassDef(identifier name,
   expr* bases,
   keyword* keywords,
   stmt* body,
   expr* decorator_list,
   type_param* type_params)
| Return(expr? value)

| Delete(expr* targets)
| Assign(expr* targets, expr value, string? ty
| TypeAlias(expr name, type_param* type_params
| AugAssign(expr target, operator op, expr val
-- 'simple' indicates that we annotate simple
| AnnAssign(expr target, expr annotation, expr

-- use 'orelse' because else is a keyword in t
| For(expr target, expr iter, stmt* body, stmt
| AsyncFor(expr target, expr iter, stmt* body,
| While(expr test, stmt* body, stmt* orelse)
| If(expr test, stmt* body, stmt* orelse)
| With(withitem* items, stmt* body, string? ty
| AsyncWith(withitem* items, stmt* body, strin

| Match(expr subject, match_case* cases)

| Raise(expr? exc, expr? cause)
| Try(stmt* body, excepthandler* handlers, stm
| TryStar(stmt* body, excepthandler* handlers,
| Assert(expr test, expr? msg)

| Import(alias* names)
| ImportFrom(identifier? module, alias* names,
               | Global(identifier* names)
               | Nonlocal(identifier* names)
               | Expr(expr value)
               | Pass | Break | Continue

               -- col_offset is the byte offset in the utf8 s
               attributes (int lineno, int col_offset, int? e

               -- BoolOp() can use left & right?
         expr = BoolOp(boolop op, expr* values)
              | NamedExpr(expr target, expr value)
              | BinOp(expr left, operator op, expr right)
              | UnaryOp(unaryop op, expr operand)
              | Lambda(arguments args, expr body)
              | IfExp(expr test, expr body, expr orelse)
              | Dict(expr?* keys, expr* values)
              | Set(expr* elts)
              | ListComp(expr elt, comprehension* generators)
              | SetComp(expr elt, comprehension* generators)
              | DictComp(expr key, expr value, comprehension*
              | GeneratorExp(expr elt, comprehension* generat
              -- the grammar constrains where yield expressio
              | Await(expr value)
              | Yield(expr? value)
              | YieldFrom(expr value)
              -- need sequences for compare to distinguish be
              -- x < 4 < 3 and (x < 4) < 3
              | Compare(expr left, cmpop* ops, expr* comparat
              | Call(expr func, expr* args, keyword* keywords
              | FormattedValue(expr value, int conversion, ex
              | Interpolation(expr value, constant str, int c
              | JoinedStr(expr* values)
              | TemplateStr(expr* values)
              | Constant(constant value, string? kind)

              -- the following expression can appear in assig
              | Attribute(expr value, identifier attr, expr_c
              | Subscript(expr value, expr slice, expr_contex
     | Starred(expr value, expr_context ctx)
     | Name(identifier id, expr_context ctx)
     | List(expr* elts, expr_context ctx)
     | Tuple(expr* elts, expr_context ctx)

     -- can appear only in Subscript
     | Slice(expr? lower, expr? upper, expr? step)

      -- col_offset is the byte offset in the utf8 s
      attributes (int lineno, int col_offset, int? e

expr_context = Load | Store | Del

boolop = And | Or

operator = Add | Sub | Mult | MatMult | Div | Mod |
             | RShift | BitOr | BitXor | BitAnd | Fl

unaryop = Invert | Not | UAdd | USub

cmpop = Eq | NotEq | Lt | LtE | Gt | GtE | Is | IsNo

comprehension = (expr target, expr iter, expr* ifs,

excepthandler = ExceptHandler(expr? type, identifier
                attributes (int lineno, int col_offs

arguments = (arg* posonlyargs, arg* args, arg? varar
             expr?* kw_defaults, arg? kwarg, expr* d

arg = (identifier arg, expr? annotation, string? typ
       attributes (int lineno, int col_offset, int?

-- keyword arguments supplied to call (NULL identifi
keyword = (identifier? arg, expr value)
           attributes (int lineno, int col_offset, i

-- import name with optional 'as' alias.
```

```
    alias = (identifier name, identifier? asname)
             attributes (int lineno, int col_offset, int

    withitem = (expr context_expr, expr? optional_vars)

    match_case = (pattern pattern, expr? guard, stmt* bo

    pattern = MatchValue(expr value)
            | MatchSingleton(constant value)
            | MatchSequence(pattern* patterns)
            | MatchMapping(expr* keys, pattern* patterns
            | MatchClass(expr cls, pattern* patterns, id

            | MatchStar(identifier? name)
            -- The optional "rest" MatchMapping paramete

            | MatchAs(pattern? pattern, identifier? name
            | MatchOr(pattern* patterns)

             attributes (int lineno, int col_offset, int

    type_ignore = TypeIgnore(int lineno, string tag)

    type_param = TypeVar(identifier name, expr? bound, e
               | ParamSpec(identifier name, expr? defaul
               | TypeVarTuple(identifier name, expr? def
               attributes (int lineno, int col_offset, i
}
```

Clases nodo

class ast.AST

This is the base of all AST node classes. The actual node classes are derived from the `Parser/Python.asdl` file, which is reproduced above. They are defined in the **`_ast`** C module and re-exported in **`ast`**.

Hay una clase definida para cada símbolo del lado izquierdo en la gramática abstracta (por ejemplo, **`ast.stmt`** o **`ast.expr`**). Además, hay una clase definida para cada constructor en el lado derecho; estas clases heredan de las clases para los árboles del lado izquierdo. Por ejemplo, **`ast.BinOp`** hereda de **`ast.expr`**. Para las reglas de producción con alternativas (también conocidas como «sumas»), la clase del lado izquierdo es abstracta: solo se crean instancias de nodos de constructor específicos.

_fields

Each concrete class has an attribute **`_fields`** which gives the names of all child nodes.

Cada instancia de una clase concreta tiene un atributo para cada nodo secundario, del tipo definido en la gramática. Por ejemplo, las instancias **`ast.BinOp`** tienen un atributo **`left`** de tipo **`ast.expr`**.

Si estos atributos están marcados como opcionales en la gramática (usando un signo de interrogación), el valor podría ser `None`. Si los atributos pueden tener cero o más valores (marcados con un asterisco), los valores se representan como listas de Python. Todos los atributos posibles deben estar presentes y tener valores válidos al compilar un AST con **`compile()`**.

_field_types

The **`_field_types`** attribute on each concrete class is a dictionary mapping field names (as also listed in **`_fields`**) to their types.

```
>>> ast.TypeVar._field_types
{'name': <class 'str'>, 'bound': ast.expr | Nor
```

Added in version 3.13.

lineno
col_offset
end_lineno

end_col_offset

Las instancias de las subclases `ast.expr` y `ast.stmt` tienen atributos `lineno`, `col_offset`, `lineno`, y `col_offset`. `lineno` y `end_lineno` son los números de la primera y última línea del intervalo de texto de origen (1 indexado, por lo que la primera línea es la línea 1), y `col_offset` y `end_col_offset` son las correspondientes compensaciones de bytes UTF-8 del primer y último token que generó el nodo. El desplazamiento UTF-8 se registra porque el analizador utiliza UTF-8 internamente.

Tenga en cuenta que el compilador no requiere las posiciones finales y, por lo tanto, son opcionales. El desplazamiento final es *después* del último símbolo, por ejemplo, uno puede obtener el segmento fuente de un nodo de expresión de una línea usando `source_line[node.col_offset: node.end_col_offset]`.

El constructor de una clase `ast.T` analiza sus argumentos de la siguiente manera:

- Si hay argumentos posicionales, debe haber tantos como elementos en `T._fields`; serán asignados como atributos de estos nombres.
- Si hay argumentos de palabras clave, establecerán los atributos de los mismos nombres a los valores dados.

Por ejemplo, para crear y completar un nodo `ast.UnaryOp`, puede usar

```
node = ast.UnaryOp(ast.USub(), ast.Constant(5, line
                   lineno=0, col_offset=0)
```

If a field that is optional in the grammar is omitted from the constructor, it defaults to `None`. If a list field is omitted, it defaults to the empty list. If a field of type `ast.expr_context` is omitted, it defaults to `Load()`. If any other field is omitted, a `DeprecationWarning` is raised and the AST node will not have this field. In Python 3.15, this condition will raise an error.

Distinto en la versión 3.8: La clase `ast.Constant` ahora se usa para todas las constantes.

Distinto en la versión 3.9: Los índices simples se representan por su valor, los segmentos extendidos se representan como tuplas.

Distinto en la versión 3.13: AST node constructors were changed to provide sensible defaults for omitted fields: optional fields now default to `None`, list fields default to an empty list, and fields of type **`ast.expr_context`** default to `Load()`. Previously, omitted attributes would not exist on constructed nodes (accessing them raised `AttributeError`).

Distinto en la versión 3.14: The `__repr__()` output of `AST` nodes includes the values of the node fields.

Deprecated since version 3.8, removed in version 3.14: Previous versions of Python provided the AST classes **`ast.Num`**, **`ast.Str`**, **`ast.Bytes`**, **`ast.NameConstant`** and **`ast.Ellipsis`**, which were deprecated in Python 3.8. These classes were removed in Python 3.14, and their functionality has been replaced with `ast.Constant`.

Obsoleto desde la versión 3.9: Old classes **`ast.Index`** and **`ast.ExtSlice`** are still available, but they will be removed in future Python releases. In the meantime, instantiating them will return an instance of a different class.

Deprecated since version 3.13, will be removed in version 3.15: Previous versions of Python allowed the creation of AST nodes that were missing required fields. Similarly, AST node constructors allowed arbitrary keyword arguments that were set as attributes of the AST node, even if they did not match any of the fields of the AST node. This behavior is deprecated and will be removed in Python 3.15.

Nota

Las descripciones de las clases de nodo específicas mostradas aquí fueron adaptadas inicialmente del fantástico proyecto Green Tree Snakes [https://greentreesnakes.readthedocs.io/en/latest/] y todos sus contribuidores.

Nodos raíz

class ast.Module(*body*, *type_ignores*)

Un módulo de Python, como con archivo-entrada. Tipo de nodo generado por **`ast.parse()`** en el modo por defecto `"exec"` *mode*.*mode*.

`body` is a **`list`** of the module's Declaraciones.

`type_ignores` is a **`list`** of the module's type ignore comments; see **`ast.parse()`** for more details.

```
>>> print(ast.dump(ast.parse('x = 1'), indent=4))
Module(
    body=[
        Assign(
            targets=[
                Name(id='x', ctx=Store())],
            value=Constant(value=1))])
```

class ast.Expression(*body*)

Una única expression input de Python. Tipo de nodo generado por **`ast.parse()`** cuando *mode* es `"eval"`.

`body` is a single node, one of the expression types.

```
>>> print(ast.dump(ast.parse('123', mode='eval'), i
Expression(
    body=Constant(value=123))
```

class ast.Interactive(*body*)

Una única entrada interactiva, como en Modo interactivo. Tipo de nodo generado por `ast.parse()` cuando *mode* es `"single"`.

`body` is a `list` of statement nodes.

```
>>> print(ast.dump(ast.parse('x = 1; y = 2', mode='
Interactive(
    body=[
        Assign(
            targets=[
                Name(id='x', ctx=Store())],
            value=Constant(value=1)),
        Assign(
            targets=[
                Name(id='y', ctx=Store())],
            value=Constant(value=2))])
```

class ast.FunctionType(*argtypes*, *returns*)

Una interpretación de un comentario de tipo de estilo antiguo para funciones, ya que las versiones de Python anteriores a la 3.5 no soportaban anotaciones **PEP 484** [https://peps.python.org/pep-0484/]. Tipo de nodo generado por `ast.parse()` cuando *mode* es `"func_type"`.

Los comentarios de este tipo tendrían el siguiente aspecto:

```
def sum_two_number(a, b):
    # type: (int, int) -> int
    return a + b
```

`argtypes` is a `list` of expression nodes.

`returns` is a single expression node.

```
>>> print(ast.dump(ast.parse('(int, str) -> List[in
FunctionType(
    argtypes=[
        Name(id='int', ctx=Load()),
        Name(id='str', ctx=Load())],
    returns=Subscript(
        value=Name(id='List', ctx=Load()),
        slice=Name(id='int', ctx=Load()),
        ctx=Load()))
```

Added in version 3.8.

Literales

class ast.Constant(*value*)

A constant value. The `value` attribute of the `Constant` literal contains the Python object it represents. The values represented can be instances of **`str`**, **`bytes`**, **`int`**, **`float`**, **`complex`**, and **`bool`**, and the constants **`None`** and **`Ellipsis`**.

```
>>> print(ast.dump(ast.parse('123', mode='eval'), i
Expression(
    body=Constant(value=123))
```

class ast.FormattedValue(*value*, *conversion*, *format_spec*)

Nodo que representa un único campo de formato en una `f-string`. Si la cadena de caracteres contiene un único campo de formato y nada más, el nodo puede estar aislado de otra manera aparece en **`JoinedStr`**.

- `value` es cualquier nodo de expresión (como un literal, una variable o una llamada a función).
- `conversion` es un entero:
    - ○ -1: sin formato
    - ○ 97 (`ord('a')`): `!a` **`ASCII`** formatting
    - ○ 114 (`ord('r')`): `!r` **`repr()`** formatting
    - ○ 115 (`ord('s')`): `!s` **`string`** formatting
- `format_spec` es un nodo **`JoinedStr`** que representa el formato del valor, o `None` si no se ha especificado un formato. Ambos, `conversion` y `format_spec`, pueden estar especificados al mismo tiempo.

class ast.JoinedStr(*values*)

Un f-string que comprende una serie de nodos **`FormattedValue`** y **`Constant`**.

```
>>> print(ast.dump(ast.parse('f"sin({a}) is {sin(a)
Expression(
    body=JoinedStr(
        values=[
            Constant(value='sin('),
            FormattedValue(
                value=Name(id='a', ctx=Load()),
                conversion=-1),
            Constant(value=') is '),
            FormattedValue(
                value=Call(
                    func=Name(id='sin', ctx=Load())
                    args=[
                        Name(id='a', ctx=Load())]),
                conversion=-1,
                format_spec=JoinedStr(
                    values=[
                        Constant(value='.3')]))]))
```

class ast.TemplateStr(*values*, */*)

Added in version 3.14.

Node representing a template string literal, comprising a series of **`Interpolation`** and **`Constant`** nodes. These nodes may be any order, and do not need to be interleaved.

```
>>> expr = ast.parse('t"{name} finished {place:ordi
>>> print(ast.dump(expr, indent=4))
Expression(
    body=TemplateStr(
        values=[
            Interpolation(
                value=Name(id='name', ctx=Load()),
                str='name',
                conversion=-1),
            Constant(value=' finished '),
            Interpolation(
                value=Name(id='place', ctx=Load()),
                str='place',
                conversion=-1,
                format_spec=JoinedStr(
                    values=[
                        Constant(value='ordinal')])
```

class ast.Interpolation(*value*, *str*, *conversion*, *format_spec=None*)

Added in version 3.14.

Node representing a single interpolation field in a template string literal.

- `value` is any expression node (such as a literal, a variable, or a function call). This has the same meaning as `FormattedValue.value`.
- `str` is a constant containing the text of the interpolation expression.

  If `str` is set to `None`, then `value` is used to generate code when calling **`ast.unparse()`**. This no longer guarantees that the generated code is identical to the original and is intended for code generation.
- `conversion` es un entero:
  - -1: no conversion
  - 97 (`ord('a')`): `!a` **`ASCII`** conversion
  - 114 (`ord('r')`): `!r` **`repr()`** conversion
  - 115 (`ord('s')`): `!s` **`string`** conversion

  This has the same meaning as `FormattedValue.conversion`.
- `format_spec` is a **`JoinedStr`** node representing the formatting of the value, or `None` if no format was specified. Both `conversion` and `format_spec` can be set at the same time. This has the same meaning as `FormattedValue.format_spec`.

class ast.List(*elts*, *ctx*)

class ast.Tuple(*elts*, *ctx*)

Una lista o tupla. `elts` contiene una lista de nodos que representa a los elementos. `ctx` es **`Store`** si el contenedor es un objetivo de asignación (por ejemplo `(x,y)=something`), y **`Load`** en cualquier otro caso.

```
>>> print(ast.dump(ast.parse('[1, 2, 3]', mode='eva
Expression(
    body=List(
        elts=[
            Constant(value=1),
            Constant(value=2),
            Constant(value=3)],
        ctx=Load()))
>>> print(ast.dump(ast.parse('(1, 2, 3)', mode='eva
Expression(
    body=Tuple(
        elts=[
            Constant(value=1),
            Constant(value=2),
            Constant(value=3)],
        ctx=Load()))
```

class ast.Set(*elts*)

Un set. `elts` contiene una lista de nodos que representa a un set de elementos.

```
>>> print(ast.dump(ast.parse('{1, 2, 3}', mode='eva
Expression(
    body=Set(
        elts=[
            Constant(value=1),
            Constant(value=2),
            Constant(value=3)]))
```

class ast.Dict(*keys*, *values*)

Un diccionario. `keys` y `values` contienen listas de nodos que representan las claves y los valores respectivamente en el orden correspondiente (el orden que retornaría `dictionary.keys()` y `dictionary.values()`).

Cuando se desempaqueta un diccionario utilizando literales de diccionario, la expresión a ser expandida va en la lista `values`, con `None` en la posición correspondiente en `keys`.

```
>>> print(ast.dump(ast.parse('{"a":1, **d}', mode='
Expression(
    body=Dict(
        keys=[
            Constant(value='a'),
            None],
        values=[
            Constant(value=1),
            Name(id='d', ctx=Load())]))
```

Variables

class ast.Name(*id*, *ctx*)

Un nombre de variable. `id` contiene el nombre de una cadena de caracteres y `ctx` es uno de los siguientes tipos.

class ast.Load
class ast.Store
class ast.Del

Referencias a variables que pueden ser usadas para cargar el valor de una variable, asignar un nuevo valor o borrarlo. Las referencias a variables reciben un contexto para distinguir entre estos casos.

```
>>> print(ast.dump(ast.parse('a'), indent=4))
Module(
    body=[
        Expr(
            value=Name(id='a', ctx=Load()))])

>>> print(ast.dump(ast.parse('a = 1'), indent=4))
```

```
Module(
    body=[
        Assign(
            targets=[
                Name(id='a', ctx=Store())],
            value=Constant(value=1))])

>>> print(ast.dump(ast.parse('del a'), indent=4))
Module(
    body=[
        Delete(
            targets=[
                Name(id='a', ctx=Del())])])
```

class ast.Starred(*value*, *ctx*)

Una referencia a variable `*var`. `value` contiene la variable, típicamente un nodo **`Name`**. Este tipo puede ser usado cuando se construye un nodo **`Call`** con `*args`.

```
>>> print(ast.dump(ast.parse('a, *b = it'), indent=
Module(
    body=[
        Assign(
            targets=[
                Tuple(
                    elts=[
                        Name(id='a', ctx=Store()),
                        Starred(
                            value=Name(id='b', ctx=
                            ctx=Store())],
                    ctx=Store())],
            value=Name(id='it', ctx=Load()))])
```

Expresiones

class ast.Expr(*value*)

Cuando una expresión, como un llamado a función, aparece como una declaración por sí misma sin que su valor de retorno se use o se almacene, está dentro de este contenedor. `value` contiene uno de los otros nodos en esta sección, un nodo **`Constant`**, **`Name`**, **`Lambda`**, **`Yield`** o **`YieldFrom`**.

```
>>> print(ast.dump(ast.parse('-a'), indent=4))
Module(
    body=[
        Expr(
            value=UnaryOp(
                op=USub(),
                operand=Name(id='a', ctx=Load())))]
```

class ast.UnaryOp(*op*, *operand*)

Una operación unaria. `op` es el operador y `operand` es cualquier nodo de expresión.

class ast.UAdd
class ast.USub
class ast.Not
class ast.Invert

Tokens de operador unario. **`Not`** es la palabra clave `not`, **`Invert`** es el operador `~`.

```
>>> print(ast.dump(ast.parse('not x', mode='eval'),
Expression(
    body=UnaryOp(
        op=Not(),
        operand=Name(id='x', ctx=Load())))
```

class ast.BinOp(*left*, *op*, *right*)

Una operación binaria (como la suma o división(. `op` es el operador, y `left` y `right` son cualquier nodo de expresión.

```
>>> print(ast.dump(ast.parse('x + y', mode='eval'),
Expression(
    body=BinOp(
        left=Name(id='x', ctx=Load()),
        op=Add(),
        right=Name(id='y', ctx=Load())))
```

class ast.Add
class ast.Sub
class ast.Mult
class ast.Div
class ast.FloorDiv
class ast.Mod
class ast.Pow
class ast.LShift
class ast.RShift
class ast.BitOr
class ast.BitXor
class ast.BitAnd
class ast.MatMult

Tokens de operador binario.

class ast.BoolOp(*op*, *values*)

Una operación booleana, “or” y “and”. `op` es **`Or`** o **`And`**. `values` son los valores involucrados. Operaciones consecutivas con el mismo operador, como `a or b or c`, colapsan en un nodo con varios valores.

Esto no incluye `not`, el cual es un **`UnaryOp`**.

```
>>> print(ast.dump(ast.parse('x or y', mode='eval')
Expression(
    body=BoolOp(
        op=Or(),
        values=[
            Name(id='x', ctx=Load()),
            Name(id='y', ctx=Load())]))
```

class ast.And
class ast.Or

Tokens de operador booleano.

class ast.Compare(*left*, *ops*, *comparators*)

Una comparación de dos o más valores. `left` es el primer valor en la comparación, `ops` es la lista de operadores, y `comparators` es la lista de valores después de el primer elemento en la comparación.

```
>>> print(ast.dump(ast.parse('1 <= a < 10', mode='e
Expression(
    body=Compare(
        left=Constant(value=1),
        ops=[
            LtE(),
            Lt()],
        comparators=[
            Name(id='a', ctx=Load()),
            Constant(value=10)]))
```

class ast.Eq
class ast.NotEq
class ast.Lt
class ast.LtE
class ast.Gt
class ast.GtE
class ast.Is
class ast.IsNot
class ast.In
class ast.NotIn

Tokens de operador de comparación.

class ast.Call(*func*, *args*, *keywords*)

Un llamado a función. `func` is la función, la cual suele ser un objeto **`Name`** o **`Attribute`**. De los argumentos:

- `args` contiene una lista de argumentos pasados por posición.
- `keywords` contiene una lista de objetos **`keyword`** que representan argumentos pasados por nombre clave.

The `args` and `keywords` arguments are optional and default to empty lists.

```
>>> print(ast.dump(ast.parse('func(a, b=c, *d, **e)
Expression(
    body=Call(
        func=Name(id='func', ctx=Load()),
        args=[
            Name(id='a', ctx=Load()),
            Starred(
                value=Name(id='d', ctx=Load()),
                ctx=Load())],
        keywords=[
            keyword(
                arg='b',
                value=Name(id='c', ctx=Load())),
            keyword(
                value=Name(id='e', ctx=Load()))]))
```

class ast.keyword(*arg*, *value*)

Un argumento de palabra clave para una llamada de función o definición de clase. `arg` es una cadena de caracteres sin formato del nombre del parámetro, `valor` es un nodo para pasar.

class ast.IfExp(*test*, *body*, *orelse*)

Una expresión como `a if b else c`. Cada campo contiene un único nodo, por lo que en el siguiente ejemplo, todos son nodos **`Name`**.

```
>>> print(ast.dump(ast.parse('a if b else c', mode=
Expression(
    body=IfExp(
        test=Name(id='b', ctx=Load()),
        body=Name(id='a', ctx=Load()),
        orelse=Name(id='c', ctx=Load())))
```

class ast.Attribute(*value*, *attr*, *ctx*)

Acceso a atributos, por ejemplo `d.keys`. `value` es un nodo, típicamente un **`Name`**. `attr` es una simple cadena de caracteres que da el nombre del atributo, y `ctx` es **`Load`**,

`Store` o **`Del`** de acuerdo a cómo se actúe sobre el atributo.

```
>>> print(ast.dump(ast.parse('snake.colour', mode='
Expression(
    body=Attribute(
        value=Name(id='snake', ctx=Load()),
        attr='colour',
        ctx=Load()))
```

class ast.NamedExpr(*target*, *value*)

Una expresión con nombre. Este nodo AST es producido por el operador de expresiones de asignación (también conocido como el operador walrus). A diferencia del nodo **`Assign`** en el cual el primer argumento puede ser varios nodos, en este caso `target` y `value` deben ser nodos únicos.

```
>>> print(ast.dump(ast.parse('(x := 4)', mode='eval
Expression(
    body=NamedExpr(
        target=Name(id='x', ctx=Store()),
        value=Constant(value=4)))
```

Added in version 3.8.

Subindexado

class ast.Subscript(*value*, *slice*, *ctx*)

Un subíndice, como `l[1]`. `value` es el objeto subindicado (usualmente una secuencia o mapeo). `slice` es un índice, un segmento o una clave. Este puede ser una **`Tuple`** y contener un **`Slice`**. `ctx` es **`Load`**, **`Store`** or **`Del`** de acuerdo a la acción tomada con el subíndice.

```
>>> print(ast.dump(ast.parse('l[1:2, 3]', mode='eva
Expression(
    body=Subscript(
        value=Name(id='l', ctx=Load()),
        slice=Tuple(
            elts=[
                Slice(
                    lower=Constant(value=1),
                    upper=Constant(value=2)),
                Constant(value=3)],
            ctx=Load()),
        ctx=Load()))
```

class ast.Slice(*lower*, *upper*, *step*)

Una segmentación regular (en la forma `lower:upper` o `lower:upper:step`). Puede ocurrir solamente dentro del campo *slice* de **`Subscript`**, ya sea directamente o como un elemento de **`Tuple`**.

```
>>> print(ast.dump(ast.parse('l[1:2]', mode='eval')
Expression(
    body=Subscript(
        value=Name(id='l', ctx=Load()),
        slice=Slice(
            lower=Constant(value=1),
            upper=Constant(value=2)),
        ctx=Load()))
```

Comprensiones

class ast.ListComp(*elt*, *generators*)
class ast.SetComp(*elt*, *generators*)
class ast.GeneratorExp(*elt*, *generators*)
class ast.DictComp(*key*, *value*, *generators*)

Listas y sets por comprensión, expresiones de generadores, y diccionarios por comprensión. `elt` (o `key` y `value`) es un único nodo que representa la parte que va a ser evaluada por cada item.

`generators` es una lista de nodos **`comprehension`**.

```
>>> print(ast.dump(
...     ast.parse('[x for x in numbers]', mode='eva
...     indent=4,
... ))
Expression(
    body=ListComp(
        elt=Name(id='x', ctx=Load()),
        generators=[
            comprehension(
                target=Name(id='x', ctx=Store()),
                iter=Name(id='numbers', ctx=Load())
                is_async=0)]))
>>> print(ast.dump(
...     ast.parse('{x: x**2 for x in numbers}', mod
...     indent=4,
... ))
Expression(
    body=DictComp(
        key=Name(id='x', ctx=Load()),
        value=BinOp(
            left=Name(id='x', ctx=Load()),
            op=Pow(),
            right=Constant(value=2)),
        generators=[
            comprehension(
                target=Name(id='x', ctx=Store()),
                iter=Name(id='numbers', ctx=Load())
                is_async=0)]))
>>> print(ast.dump(
...     ast.parse('{x for x in numbers}', mode='eva
...     indent=4,
... ))
Expression(
    body=SetComp(
        elt=Name(id='x', ctx=Load()),
        generators=[
            comprehension(
                target=Name(id='x', ctx=Store()),
                iter=Name(id='numbers', ctx=Load())
                is_async=0)]))
```

class ast.comprehension(*target*, *iter*, *ifs*, *is_async*)

Una cláusula `for` en una comprensión. `target` es la referencia a usarse por cada elemento - típicamente un nodo **`Name`** o **`Tuple`**. `iter` es el objeto por el cual se itera. `ifs` es una lista de expresiones de prueba: cada cláusula `for` puede tener múltiples `ifs`.

`is_async` indica que una compresión es asíncrona (usando `async for` en lugar de `for`). El valor es un entero (0 o 1).

```
>>> print(ast.dump(ast.parse('[ord(c) for line in f
...                        indent=4)) # Multiple comprehens
Expression(
    body=ListComp(
        elt=Call(
            func=Name(id='ord', ctx=Load()),
            args=[
                Name(id='c', ctx=Load())]),
        generators=[
            comprehension(
                target=Name(id='line', ctx=Store())
                iter=Name(id='file', ctx=Load()),
                is_async=0),
            comprehension(
                target=Name(id='c', ctx=Store()),
                iter=Name(id='line', ctx=Load()),
                is_async=0)]))

>>> print(ast.dump(ast.parse('(n**2 for n in it if
...                        indent=4)) # generator comprehen
Expression(
    body=GeneratorExp(
        elt=BinOp(
            left=Name(id='n', ctx=Load()),
            op=Pow(),
            right=Constant(value=2)),
        generators=[
            comprehension(
                target=Name(id='n', ctx=Store()),
                    iter=Name(id='it', ctx=Load()),
                    ifs=[
                        Compare(
                            left=Name(id='n', ctx=Load(
                            ops=[
                                Gt()],
                            comparators=[
                                Constant(value=5)]),
                        Compare(
                            left=Name(id='n', ctx=Load(
                            ops=[
                                Lt()],
                            comparators=[
                                Constant(value=10)])],
                    is_async=0)]))

>>> print(ast.dump(ast.parse('[i async for i in soc
...                    indent=4)) # Async comprehension
Expression(
    body=ListComp(
        elt=Name(id='i', ctx=Load()),
        generators=[
            comprehension(
                target=Name(id='i', ctx=Store()),
                iter=Name(id='soc', ctx=Load()),
                is_async=1)]))
```

Declaraciones

class ast.Assign(*targets*, *value*, *type_comment*)

Una asignación. `targets` es una lista de nodos, y `value` es un nodo único.

Nodos múltiples en `targets` representa asignar el mismo valor a cada uno. El desempaquetado se representa poniendo una **`Tuple`** o **`List`** en `targets`.

type_comment

`type_comment` es una cadena de caracteres opcional con la anotación de tipos como comentario.

```
>>> print(ast.dump(ast.parse('a = b = 1'), indent=4
Module(
    body=[
        Assign(
            targets=[
                Name(id='a', ctx=Store()),
                Name(id='b', ctx=Store())],
            value=Constant(value=1))])
```

```
>>> print(ast.dump(ast.parse('a,b = c'), indent=4))
Module(
    body=[
        Assign(
            targets=[
                Tuple(
                    elts=[
                        Name(id='a', ctx=Store()),
                        Name(id='b', ctx=Store())],
                    ctx=Store())],
            value=Name(id='c', ctx=Load()))])
```

class ast.AnnAssign(*target, annotation, value, simple*)

An assignment with a type annotation. `target` is a single node and can be a **`Name`**, an **`Attribute`** or a **`Subscript`**. `annotation` is the annotation, such as a **`Constant`** or **`Name`** node. `value` is a single optional node.

`simple` is always either 0 (indicating a «complex» target) or 1 (indicating a «simple» target). A «simple» target consists solely of a **`Name`** node that does not appear between parentheses; all other targets are considered complex. Only simple targets appear in the **`__annotations__`** dictionary of modules and classes.

```
>>> print(ast.dump(ast.parse('c: int'), indent=4))
Module(
    body=[
        AnnAssign(
            target=Name(id='c', ctx=Store()),
            annotation=Name(id='int', ctx=Load()),
            simple=1)])

>>> print(ast.dump(ast.parse('(a): int = 1'), inden
Module(
    body=[
        AnnAssign(
            target=Name(id='a', ctx=Store()),
            annotation=Name(id='int', ctx=Load()),
            value=Constant(value=1),
            simple=0)])

>>> print(ast.dump(ast.parse('a.b: int'), indent=4)
Module(
    body=[
        AnnAssign(
            target=Attribute(
                value=Name(id='a', ctx=Load()),
                attr='b',
                ctx=Store()),
            annotation=Name(id='int', ctx=Load()),
            simple=0)])

>>> print(ast.dump(ast.parse('a[1]: int'), indent=4
Module(
    body=[
        AnnAssign(
            target=Subscript(
                value=Name(id='a', ctx=Load()),
                slice=Constant(value=1),
                ctx=Store()),
            annotation=Name(id='int', ctx=Load()),
            simple=0)])
```

class ast.AugAssign(*target*, *op*, *value*)

Asignación aumentada, como `a+=1`. En el siguiente ejemplo, `target` es un nodo **`Name`** para `x` (con el contexto **`Store`**), `op` es **`Add`** y `value` es un **`Constant`** con valor 1.

El atributo `target` no puede ser de clase **`Tuple`** o **`List`**, a diferencia de los objetivos de **`Assign`**.

```
>>> print(ast.dump(ast.parse('x += 2'), indent=4))
Module(
    body=[
        AugAssign(
            target=Name(id='x', ctx=Store()),
            op=Add(),
            value=Constant(value=2))])
```

class ast.Raise(*exc*, *cause*)

Una declaración `raise`. `exc` es el objeto de excepción a ser lanzado, normalmente un **`Call`** or **`Name`**, o `None` para un `raise` solo. `cause` es la parte opcional para `y` en `raise x from y`.

```
>>> print(ast.dump(ast.parse('raise x from y'), ind
Module(
    body=[
        Raise(
            exc=Name(id='x', ctx=Load()),
            cause=Name(id='y', ctx=Load()))])
```

class ast.Assert(*test*, *msg*)

Una aserción. `test` contiene la condición, como un nodo **`Compare`**. `msg` contiene el mensaje de fallo.

```
>>> print(ast.dump(ast.parse('assert x,y'), indent=
Module(
    body=[
        Assert(
            test=Name(id='x', ctx=Load()),
            msg=Name(id='y', ctx=Load()))])
```

class ast.Delete(*targets*)

Contiene una declaración `del`. `targets` es una lista de nodos, como nodos **`Name`**, **`Attribute`** o **`Subscript`**.

```
>>> print(ast.dump(ast.parse('del x,y,z'), indent=4
Module(
    body=[
        Delete(
            targets=[
                Name(id='x', ctx=Del()),
                Name(id='y', ctx=Del()),
                Name(id='z', ctx=Del())])])
```

class ast.Pass

Una declaración `pass`.

```
>>> print(ast.dump(ast.parse('pass'), indent=4))
Module(
    body=[
        Pass()])
```

class ast.TypeAlias(*name*, *type_params*, *value*)

Un alias type alias creado mediante la sentencia **`type`**. `name` es el nombre del alias, `type_params` es una lista de parámetros type, y `value` es el valor del alias de tipo.

```
>>> print(ast.dump(ast.parse('type Alias = int'), i
Module(
    body=[
        TypeAlias(
            name=Name(id='Alias', ctx=Store()),
            value=Name(id='int', ctx=Load()))])
```

Added in version 3.12.

Otras declaraciones que solo son aplicables dentro de funciones o bucles descritos en otras secciones.

Importaciones

class ast.Import(*names*)

Una declaración de importación. `names` es una lista de nodos **`alias`**.

```
>>> print(ast.dump(ast.parse('import x,y,z'), inden
Module(
    body=[
        Import(
            names=[
                alias(name='x'),
                alias(name='y'),
                alias(name='z')])])
```

class ast.ImportFrom(*module*, *names*, *level*)

Representa `form x import y`. `module` es una cadena de caracteres sin formato del nombre "from", sin puntos, o `None` para declaraciones como `from . import foo`. `level` es un entero que contiene el nivel relativo de la importación (0 significa una importación absoluta).

```
>>> print(ast.dump(ast.parse('from y import x,y,z')
Module(
    body=[
        ImportFrom(
            module='y',
            names=[
                alias(name='x'),
                alias(name='y'),
                alias(name='z')],
            level=0)])
```

class ast.alias(*name*, *asname*)

Ambos parámetros son cadenas de caracteres sin formato para los nombres. `asname` puede ser `None` si se va a usar el nombre regular.

```
>>> print(ast.dump(ast.parse('from ..foo.bar import
Module(
    body=[
        ImportFrom(
            module='foo.bar',
            names=[
                alias(name='a', asname='b'),
                alias(name='c')],
            level=2)])
```

Control de flujo

Nota

Cláusulas opcionales como `else` se guardan como una lista vacía si no están presentes.

class ast.If(*test*, *body*, *orelse*)

Una declaración `if`. `test` contiene un único nodo, como un nodo **`Compare`**. `body` y `orelse` contiene cada uno una lista de nodos.

Cláusulas `elif` no tienen una representación especial en AST, pero pueden aparecer como nodos extra **`If`** dentro de la sección `orelse` del nodo anterior.

```
>>> print(ast.dump(ast.parse("""
... if x:
...     ...
... elif y:
...     ...
... else:
...     ...
... """), indent=4))
Module(
    body=[
        If(
            test=Name(id='x', ctx=Load()),
            body=[
                Expr(
                    value=Constant(value=Ellipsis))
            orelse=[
                If(
                    test=Name(id='y', ctx=Load()),
                    body=[
                        Expr(
                            value=Constant(value=El
                    orelse=[
                        Expr(
                            value=Constant(value=El
```

class ast.For(*target*, *iter*, *body*, *orelse*, *type_comment*)

Un bucle `for`. `target` contiene las variables a las que asigna el bucle, como un único nodo **`Name`**, **`Tuple`**, **`List`**, **`Attribute`** o **`Subscript`**. `iter` contiene el elemento sobre el que se realizará el bucle, nuevamente como un solo nodo. `body` y `orelse` contienen listas de nodos para ejecutar. Los de `orelse` se ejecutan si el ciclo finaliza normalmente, en lugar de mediante una instrucción `break`.

type_comment

`type_comment` es una cadena de caracteres opcional con la anotación de tipos como comentario.

```
>>> print(ast.dump(ast.parse("""
... for x in y:
...     ...
... else:
...     ...
... """), indent=4))
Module(
    body=[
        For(
            target=Name(id='x', ctx=Store()),
            iter=Name(id='y', ctx=Load()),
            body=[
                    Expr(
                        value=Constant(value=Ellipsis))
                orelse=[
                    Expr(
                        value=Constant(value=Ellipsis))
```

class ast.While(*test*, *body*, *orelse*)

Un bucle `while`. `test` contiene la condición, como un nodo **`Compare`**.

```
>>> print(ast.dump(ast.parse("""
... while x:
...    ...
... else:
...    ...
... """), indent=4))
Module(
    body=[
        While(
            test=Name(id='x', ctx=Load()),
            body=[
                Expr(
                    value=Constant(value=Ellipsis))
            orelse=[
                Expr(
                    value=Constant(value=Ellipsis))
```

class ast.Break

class ast.Continue

Las declaraciones `break` y `continue`.

```
>>> print(ast.dump(ast.parse("""\
... for a in b:
...     if a > 5:
...         break
...     else:
...         continue
...
... """), indent=4))
Module(
    body=[
        For(
            target=Name(id='a', ctx=Store()),
            iter=Name(id='b', ctx=Load()),
            body=[
                If(
                    test=Compare(
                        left=Name(id='a', ctx=Load(
                        ops=[
                            Gt()],
                        comparators=[
                            Constant(value=5)]),
                    body=[
                        Break()],
                    orelse=[
                        Continue()])])])
```

class ast.Try(*body*, *handlers*, *orelse*, *finalbody*)

Bloques `try`. Todos los atributos son listas de nodos a ejecutar, excepto para `handlers`, el cual es una lista de nodos **`ExceptHandler`**.

```
>>> print(ast.dump(ast.parse("""
... try:
...     ...
... except Exception:
...     ...
... except OtherException as e:
...     ...
... else:
...     ...
... finally:
...     ...
... """), indent=4))
Module(
    body=[
        Try(
            body=[
                Expr(
                    value=Constant(value=Ellipsis))
            handlers=[
                ExceptHandler(
                    type=Name(id='Exception', ctx=L
                    body=[
                        Expr(
                            value=Constant(value=El
                ExceptHandler(
                    type=Name(id='OtherException',
                    name='e',
                    body=[
                        Expr(
                            value=Constant(value=El
            orelse=[
                Expr(
                    value=Constant(value=Ellipsis))
            finalbody=[
                Expr(
                    value=Constant(value=Ellipsis))
```

class ast.TryStar(*body*, *handlers*, *orelse*, *finalbody*)

Bloques `try` que van seguidos de cláusulas `except*`. Los atributos son los mismos que para **`Try`** pero los nodos **`ExceptHandler`** en `handlers` se interpretan como bloques `except*` en lugar de `except`.

```
>>> print(ast.dump(ast.parse("""
... try:
...     ...
... except* Exception:
...     ...
... """), indent=4))
Module(
    body=[
        TryStar(
            body=[
                Expr(
                    value=Constant(value=Ellipsis))
            handlers=[
                ExceptHandler(
                    type=Name(id='Exception', ctx=L
                    body=[
                        Expr(
                            value=Constant(value=El
```

Added in version 3.11.

class ast.ExceptHandler(*type*, *name*, *body*)

Una sola cláusula `except`. `type` es el tipo de excepción con el que coincidirá, normalmente un nodo **Name** (o `None` para una cláusula `except:` generalizada). `name` es una cadena sin formato para que el nombre contenga la excepción, o `None` si la cláusula no tiene `as foo`. `body` es una lista de nodos.

```
>>> print(ast.dump(ast.parse("""\
... try:
...     a + 1
... except TypeError:
...     pass
... """), indent=4))
Module(
    body=[
        Try(
            body=[
                Expr(
                    value=BinOp(
                        left=Name(id='a', ctx=Load(
                        op=Add(),
                        right=Constant(value=1)))],
            handlers=[
                ExceptHandler(
                    type=Name(id='TypeError', ctx=L
                    body=[
                                                Pass()])])])
```

class ast.With(*items*, *body*, *type_comment*)

Un bloque `with`. `items` es una lista de nodos **`withitem`** que representan los administradores de contexto, y `body` es el bloque con sangría dentro del contexto.

type_comment

`type_comment` es una cadena de caracteres opcional con la anotación de tipos como comentario.

class ast.withitem(*context_expr*, *optional_vars*)

Un administrador de contexto único en un bloque `with`. `context_expr` es el administrador de contexto, a menudo un nodo **`Call`**. `optional_vars` es un **`Name`**, **`Tuple`** o **`List`** para la parte `as foo`, o `None` si no se usa.

```
>>> print(ast.dump(ast.parse("""\
... with a as b, c as d:
...    something(b, d)
... """), indent=4))
Module(
    body=[
        With(
            items=[
                withitem(
                    context_expr=Name(id='a', ctx=L
                    optional_vars=Name(id='b', ctx=
                withitem(
                    context_expr=Name(id='c', ctx=L
                    optional_vars=Name(id='d', ctx=
            body=[
                Expr(
                    value=Call(
                        func=Name(id='something', c
                        args=[
                            Name(id='b', ctx=Load()
                            Name(id='d', ctx=Load()
```

La coincidencia de patrones

class ast.Match(*subject*, *cases*)

Una declaración `match`. `subject` contiene el sujeto de la coincidencia (el objeto que se compara con los casos) y `cases` contiene un iterable de nodos **`match_case`** con los diferentes casos.

Added in version 3.10.

class ast.match_case(*pattern*, *guard*, *body*)

Un patrón de caso único en una declaración `match`. `pattern` contiene el patrón de coincidencia con el que se comparará el sujeto. Tenga en cuenta que los nodos **`AST`** producidos para patrones difieren de los producidos para expresiones, incluso cuando comparten la misma sintaxis.

El atributo `guard` contiene una expresión que se evaluará si el patrón coincide con el sujeto.

`body` contiene una lista de nodos para ejecutar si el patrón coincide y el resultado de evaluar la expresión de protección es verdadero.

```
>>> print(ast.dump(ast.parse("""
... match x:
...     case [x] if x>0:
...         ...
...     case tuple():
...         ...
... """), indent=4))
Module(
    body=[
        Match(
            subject=Name(id='x', ctx=Load()),
            cases=[
                match_case(
                    pattern=MatchSequence(
                        patterns=[
                                MatchAs(name='x')]),
                        guard=Compare(
                            left=Name(id='x', ctx=Load(
                            ops=[
                                Gt()],
                            comparators=[
                                Constant(value=0)]),
                        body=[
                            Expr(
                                value=Constant(value=El
                    match_case(
                        pattern=MatchClass(
                            cls=Name(id='tuple', ctx=Lo
                        body=[
                            Expr(
                                value=Constant(value=El
```

Added in version 3.10.

class ast.MatchValue(*value*)

Un patrón de valor o literal de coincidencia que se compara por igualdad. `value` es un nodo de expresión. Los nodos de valores permitidos están restringidos como se describe en la documentación de la declaración de coincidencia. Este patrón tiene éxito si el sujeto de la coincidencia es igual al valor evaluado.

```
>>> print(ast.dump(ast.parse("""
... match x:
...     case "Relevant":
...         ...
... """), indent=4))
Module(
    body=[
        Match(
            subject=Name(id='x', ctx=Load()),
            cases=[
                match_case(
                    pattern=MatchValue(
                        value=Constant(value='Relev
                    body=[
                        Expr(
                            value=Constant(value=El
```

Added in version 3.10.

class ast.MatchSingleton(*value*)

Un patrón literal de coincidencia que se compara por identidad. `value` es el singleton que se va a comparar con: `None`, `True` o `False`. Este patrón tiene éxito si el sujeto de la coincidencia es la constante dada.

```
>>> print(ast.dump(ast.parse("""
... match x:
...     case None:
...         ...
... """), indent=4))
Module(
    body=[
        Match(
            subject=Name(id='x', ctx=Load()),
            cases=[
                match_case(
                    pattern=MatchSingleton(value=No
                    body=[
                        Expr(
                            value=Constant(value=El
```

Added in version 3.10.

class ast.MatchSequence(*patterns*)

Un patrón de secuencia de coincidencia. `patterns` contiene los patrones que se compararán con los elementos del sujeto si el sujeto es una secuencia. Coincide con una secuencia de longitud variable si uno de los subpatrones es un nodo `MatchStar`; de lo contrario, coincide con una secuencia de longitud fija.

```
>>> print(ast.dump(ast.parse("""
... match x:
...     case [1, 2]:
...         ...
... """), indent=4))
Module(
    body=[
        Match(
            subject=Name(id='x', ctx=Load()),
            cases=[
                match_case(
                    pattern=MatchSequence(
                        patterns=[
                            MatchValue(
                                value=Constant(valu
                            MatchValue(
                                value=Constant(valu
                    body=[
                        Expr(
                            value=Constant(value=El
```

Added in version 3.10.

class ast.MatchStar(*name*)

Coincide con el resto de la secuencia en un patrón de secuencia de coincidencia de longitud variable. Si `name` no es `None`, una lista que contiene los elementos de secuencia restantes está vinculada a ese nombre si el patrón de secuencia general es exitoso.

```
>>> print(ast.dump(ast.parse("""
... match x:
...     case [1, 2, *rest]:
...         ...
...     case [*_]:
...         ...
... """), indent=4))
Module(
    body=[
Match(
    subject=Name(id='x', ctx=Load()),
    cases=[
        match_case(
            pattern=MatchSequence(
                patterns=[
                    MatchValue(
                        value=Constant(valu
                    MatchValue(
                        value=Constant(valu
                    MatchStar(name='rest')]
            body=[
                Expr(
                    value=Constant(value=El
        match_case(
            pattern=MatchSequence(
                patterns=[
                    MatchStar()]),
            body=[
                Expr(
                    value=Constant(value=El
```

Added in version 3.10.

class ast.MatchMapping(*keys*, *patterns*, *rest*)

Un patrón de mapeo de coincidencias. `keys` es una secuencia de nodos de expresión. `patterns` es una secuencia correspondiente de nodos de patrón. `rest` es un nombre opcional que se puede especificar para capturar los elementos de mapeo restantes. Las expresiones clave permitidas están restringidas como se describe en la documentación de la declaración de coincidencia.

Este patrón tiene éxito si el sujeto es un mapeo, todas las expresiones clave evaluadas están presentes en el mapeo y el valor correspondiente a cada clave coincide con el subpatrón correspondiente. Si `rest` no es `None`, un dict que contiene los elementos de mapeo restantes se vincula a ese nombre si el patrón de mapeo general es exitoso.

```
>>> print(ast.dump(ast.parse("""
... match x:
...     case {1: _, 2: _}:
...         ...
...     case {**rest}:
...         ...
... """), indent=4))
Module(
    body=[
        Match(
            subject=Name(id='x', ctx=Load()),
            cases=[
                match_case(
                    pattern=MatchMapping(
                        keys=[
                            Constant(value=1),
                            Constant(value=2)],
                        patterns=[
                            MatchAs(),
                            MatchAs()]),
                    body=[
                        Expr(
                            value=Constant(value=El
                match_case(
                    pattern=MatchMapping(rest='rest
                    body=[
                        Expr(
                            value=Constant(value=El
```

Added in version 3.10.

class ast.MatchClass(*cls*, *patterns*, *kwd_attrs*, *kwd_patterns*)

Un patrón de clase coincidente. `cls` es una expresión que da la clase nominal que se va a emparejar. `patterns` es una secuencia de nodos de patrón que se compararán con la secuencia definida por la clase de atributos de coincidencia de patrones. `kwd_attrs` es una secuencia de atributos adicionales que deben coincidir (especificados como argumentos de palabra clave en el patrón de clase), `kwd_patterns` son los patrones correspondientes (especificados como valores de palabras clave en el patrón de clase).

Este patrón tiene éxito si el sujeto es una instancia de la clase nominada, todos los patrones posicionales coinciden con los atributos definidos por la clase correspondientes y cualquier atributo de palabra clave especificado coincide con su patrón correspondiente.

Nota: las clases pueden definir una propiedad que retorna self para hacer coincidir un nodo de patrón con la instancia que se está comparando. Varios tipos incorporados también se combinan de esa manera, como se describe en la documentación de la declaración de coincidencia.

```
>>> print(ast.dump(ast.parse("""
... match x:
...     case Point2D(0, 0):
...         ...
...     case Point3D(x=0, y=0, z=0):
...         ...
... """), indent=4))
Module(
    body=[
        Match(
            subject=Name(id='x', ctx=Load()),
            cases=[
                match_case(
                    pattern=MatchClass(
                        cls=Name(id='Point2D', ctx=
                        patterns=[
                            MatchValue(
                                value=Constant(valu
                            MatchValue(
                                value=Constant(valu
                    body=[
                        Expr(
                            value=Constant(value=El
                match_case(
                    pattern=MatchClass(
                        cls=Name(id='Point3D', ctx=
                        kwd_attrs=[
                            'x',
                            'y',
                            'z'],
                        kwd_patterns=[
                            MatchValue(
                                value=Constant(valu
                            MatchValue(
                                value=Constant(valu
                            MatchValue(
                                value=Constant(valu
                    body=[
                        Expr(
                            value=Constant(value=El
```

Added in version 3.10.

class ast.MatchAs(*pattern*, *name*)

Una coincidencia «como patrón», patrón de captura o patrón comodín. `pattern` contiene el patrón de coincidencia con el que se comparará el sujeto. Si el patrón es `None`, el nodo representa un patrón de captura (es decir, un nombre simple) y siempre tendrá éxito.

El atributo `name` contiene el nombre que se vinculará si el patrón tiene éxito. Si `name` es `None`, `pattern` también debe ser `None` y el nodo representa el patrón comodín.

```
>>> print(ast.dump(ast.parse("""
... match x:
...     case [x] as y:
...         ...
...     case _:
...         ...
... """), indent=4))
```

```
Module(
    body=[
        Match(
            subject=Name(id='x', ctx=Load()),
            cases=[
                match_case(
                    pattern=MatchAs(
                        pattern=MatchSequence(
                            patterns=[
                                MatchAs(name='x')])
                        name='y'),
                    body=[
                        Expr(
                            value=Constant(value=El
                match_case(
                    pattern=MatchAs(),
                    body=[
                        Expr(
                            value=Constant(value=El
```

Added in version 3.10.

class ast.MatchOr(*patterns*)

Una coincidencia «o patrón». Un patrón-o hace coincidir cada uno de sus subpatrones con el sujeto, hasta que uno tiene éxito. Entonces se considera que el patrón-o tiene éxito. Si ninguno de los subpatrones tiene éxito, el patrón o falla. El atributo `patterns` contiene una lista de nodos de patrones de coincidencia que se compararán con el sujeto.

```
>>> print(ast.dump(ast.parse("""
... match x:
...     case [x] | (y):
...         ...
... """), indent=4))
Module(
    body=[
        Match(
            subject=Name(id='x', ctx=Load()),
                cases=[
                    match_case(
                        pattern=MatchOr(
                            patterns=[
                                MatchSequence(
                                    patterns=[
                                        MatchAs(name='x
                                MatchAs(name='y')]),
                        body=[
                            Expr(
                                value=Constant(value=El
```

Added in version 3.10.

Type annotations

class ast.TypeIgnore(*lineno*, *tag*)

A `# type: ignore` comment located at *lineno*. *tag* is the optional tag specified by the form `# type: ignore <tag>`.

```
>>> print(ast.dump(ast.parse('x = 1 # type: ignore'
Module(
    body=[
        Assign(
            targets=[
                Name(id='x', ctx=Store())],
            value=Constant(value=1))],
    type_ignores=[
        TypeIgnore(lineno=1, tag='')])
>>> print(ast.dump(ast.parse('x: bool = 1 # type: i
Module(
    body=[
        AnnAssign(
            target=Name(id='x', ctx=Store()),
            annotation=Name(id='bool', ctx=Load()),
            value=Constant(value=1),
            simple=1)],
    type_ignores=[
        TypeIgnore(lineno=1, tag='[assignment]')])
```

Nota

`TypeIgnore` nodes are not generated when the *type_comments* parameter is set to `False` (default). See **`ast.parse()`** for more details.

Added in version 3.8.

Tipos de parámetro

«Parámetros de tipo pueden existir en clases, funciones y tipos de alias»

class ast.TypeVar(*name*, *bound*, *default_value*)

A **`typing.TypeVar`**. `name` is the name of the type variable. `bound` is the bound or constraints, if any. If `bound` is a **`Tuple`**, it represents constraints; otherwise it represents the bound. `default_value` is the default value; if the **`TypeVar`** has no default, this attribute will be set to `None`.

```
>>> print(ast.dump(ast.parse("type Alias[T: int = b
Module(
    body=[
        TypeAlias(
            name=Name(id='Alias', ctx=Store()),
            type_params=[
                TypeVar(
                    name='T',
                    bound=Name(id='int', ctx=Load()
                    default_value=Name(id='bool', c
            value=Subscript(
                value=Name(id='list', ctx=Load()),
                slice=Name(id='T', ctx=Load()),
                ctx=Load()))])
```

Added in version 3.12.

Distinto en la versión 3.13: Added the *default_value* parameter.

class ast.ParamSpec(*name*, *default_value*)

A **`typing.ParamSpec`**. `name` is the name of the parameter specification. `default_value` is the default value; if the **`ParamSpec`** has no default, this attribute will be set to `None`.

```
>>> print(ast.dump(ast.parse("type Alias[**P = [int
Module(
    body=[
        TypeAlias(
            name=Name(id='Alias', ctx=Store()),
            type_params=[
                ParamSpec(
                    name='P',
                    default_value=List(
                        elts=[
                            Name(id='int', ctx=Load
                            Name(id='str', ctx=Load
                        ctx=Load()))],
            value=Subscript(
                value=Name(id='Callable', ctx=Load(
                slice=Tuple(
                    elts=[
                        Name(id='P', ctx=Load()),
                        Name(id='int', ctx=Load())]
                    ctx=Load()),
                ctx=Load()))])
```

Added in version 3.12.

Distinto en la versión 3.13: Added the *default_value* parameter.

class ast.TypeVarTuple(*name*, *default_value*)

A **`typing.TypeVarTuple`**. `name` is the name of the type variable tuple. `default_value` is the default value; if the **`TypeVarTuple`** has no default, this attribute will be set to `None`.

```
>>> print(ast.dump(ast.parse("type Alias[*Ts = ()]
Module(
    body=[
        TypeAlias(
            name=Name(id='Alias', ctx=Store()),
            type_params=[
                TypeVarTuple(
                    name='Ts',
                    default_value=Tuple(ctx=Load()))
            value=Subscript(
                value=Name(id='tuple', ctx=Load()),
                slice=Tuple(
                    elts=[
                        Starred(
                            value=Name(id='Ts', ctx
                            ctx=Load())],
                    ctx=Load()),
                ctx=Load()))])
```

Added in version 3.12.

Distinto en la versión 3.13: Added the *default_value* parameter.

Definiciones de función y clase

class ast.FunctionDef(*name*, *args*, *body*, *decorator_list*, *returns*, *type_comment*, *type_params*)

Una definición de función.

- `name` es una cadena de caracteres sin formato del nombre de la función.
- `args` es un nodo **`arguments`**.
- `body` es la lista de nodos dentro de la función.
- `decorator_list` es la lista de decoradores que se aplicarán, almacenados en el exterior primero (es decir, el primero de la lista se aplicará al final).
- `returns` es la anotación de retorno.
- `type_params` es una lista de parametros de tipo.

type_comment

`type_comment` es una cadena de caracteres opcional con la anotación de tipos como comentario.

Distinto en la versión 3.12: Se ha añadido `type_params`.

class ast.Lambda(*args*, *body*)

`lambda` es una definición de función mínima que se puede utilizar dentro de una expresión. A diferencia de **`FunctionDef`**, `body` tiene un solo nodo.

```
>>> print(ast.dump(ast.parse('lambda x,y: ...'), in
Module(
    body=[
        Expr(
            value=Lambda(
                args=arguments(
                    args=[
                        arg(arg='x'),
                        arg(arg='y')]),
                body=Constant(value=Ellipsis)))])
```

class ast.arguments(*posonlyargs*, *args*, *vararg*, *kwonlyargs*, *kw_defaults*, *kwarg*, *defaults*)

Los argumentos para una función.

- `posonlyargs`, `args` y `kwonlyargs` son listas de nodos **`arg`**.
- `vararg` y `kwarg` son nodos **`arg`** únicos, en referencia a los parámetros `*args, **kwargs`.
- `kw_defaults` es una lista de valores predeterminados para argumentos de solo palabras clave. Si uno es `None`, se requiere el argumento correspondiente.
- `defaults` es una lista de valores predeterminados para argumentos que se pueden pasar posicionalmente. Si hay menos valores predeterminados, corresponden a los últimos n argumentos.

class ast.arg(*arg*, *annotation*, *type_comment*)

A single argument in a list. `arg` is a raw string of the argument name; `annotation` is its annotation, such as a **`Name`** node.

type_comment

`type_comment` es una cadena opcional con la anotación de tipo como comentario

```
>>> print(ast.dump(ast.parse("""\
... @decorator1
... @decorator2
... def f(a: 'annotation', b=1, c=2, *d, e, f=3, **
...     pass
... """), indent=4))
Module(
    body=[
        FunctionDef(
            name='f',
            args=arguments(
                args=[
                    arg(
                        arg='a',
                        annotation=Constant(value='
                    arg(arg='b'),
                    arg(arg='c')],
                vararg=arg(arg='d'),
                kwonlyargs=[
                    arg(arg='e'),
                    arg(arg='f')],
                kw_defaults=[
                    None,
                    Constant(value=3)],
                kwarg=arg(arg='g'),
                defaults=[
                    Constant(value=1),
                    Constant(value=2)]),
            body=[
                    Pass()],
                decorator_list=[
                    Name(id='decorator1', ctx=Load()),
                    Name(id='decorator2', ctx=Load())],
                returns=Constant(value='return annotati
```

class ast.Return(*value*)

Una declaración `return`.

```
>>> print(ast.dump(ast.parse('return 4'), indent=4)
Module(
    body=[
        Return(
            value=Constant(value=4))])
```

class ast.Yield(*value*)
class ast.YieldFrom(*value*)

A `yield` or `yield from` expression. Because these are expressions, they must be wrapped in an **`Expr`** node if the value sent back is not used.

```
>>> print(ast.dump(ast.parse('yield x'), indent=4))
Module(
    body=[
        Expr(
            value=Yield(
                value=Name(id='x', ctx=Load())))])

>>> print(ast.dump(ast.parse('yield from x'), inden
Module(
    body=[
        Expr(
            value=YieldFrom(
                value=Name(id='x', ctx=Load())))])
```

class ast.Global(*names*)
class ast.Nonlocal(*names*)

Declaraciones `global` y `nonlocal`. `names` es una lista de cadenas sin formato.

```
>>> print(ast.dump(ast.parse('global x,y,z'), inden
Module(
    body=[
        Global(
            names=[
                'x',
                'y',
                'z'])])

>>> print(ast.dump(ast.parse('nonlocal x,y,z'), ind
Module(
    body=[
        Nonlocal(
            names=[
                'x',
                'y',
                'z'])])
```

class ast.ClassDef(*name*, *bases*, *keywords*, *body*, *decorator_list*, *type_params*)

Una definición de clase.

- `name` es una cadena sin formato para el nombre de la clase
- `bases` es una lista de nodos para clases base especificadas explícitamente.
- `keywords` is a list of **`keyword`** nodes, principally for "metaclass". Other keywords will be passed to the metaclass, as per **PEP 3115** [https://peps.python.org/pep-3115/].
- `body` es una lista de nodos que representan el código dentro de la definición de clase.
- `decorator_list` es una lista de nodos, como en **`FunctionDef`**.
- `type_params` es una lista de parametros de tipo.

```
>>> print(ast.dump(ast.parse("""\
... @decorator1
... @decorator2
... class Foo(base1, base2, metaclass=meta):
...     pass
... """), indent=4))
Module(
    body=[
        ClassDef(
            name='Foo',
            bases=[
                Name(id='base1', ctx=Load()),
                Name(id='base2', ctx=Load())],
            keywords=[
                keyword(
                    arg='metaclass',
                    value=Name(id='meta', ctx=Load(
            body=[
                Pass()],
            decorator_list=[
                Name(id='decorator1', ctx=Load()),
                Name(id='decorator2', ctx=Load())])
```

Distinto en la versión 3.12: Se ha añadido `type_params`.

Async y await

class ast.AsyncFunctionDef(*name, args, body, decorator_list, returns, type_comment, type_params*)

Una definición de función `async def`. Tiene los mismos campos que **`FunctionDef`**.

Distinto en la versión 3.12: Se ha añadido `type_params`.

class ast.Await(*value*)

Una expresión `await`. `value` es lo que espera. Solo válido en el cuerpo de un **`AsyncFunctionDef`**.

```
>>> print(ast.dump(ast.parse("""\
... async def f():
...     await other_func()
... """), indent=4))
Module(
    body=[
        AsyncFunctionDef(
            name='f',
            args=arguments(),
            body=[
                Expr(
                    value=Await(
                        value=Call(
                            func=Name(id='other_func', c
```

class ast.AsyncFor(*target*, *iter*, *body*, *orelse*, *type_comment*)

class ast.AsyncWith(*items*, *body*, *type_comment*)

Bucles `async for` y administradores de contexto `async with`. Tienen los mismos campos que **`For`** y **`With`**, respectivamente. Solo válido en el cuerpo de un **`AsyncFunctionDef`**.

Nota

When a string is parsed by **`ast.parse()`**, operator nodes (subclasses of **`ast.operator`**, **`ast.unaryop`**, **`ast.cmpop`**, **`ast.boolop`** and **`ast.expr_context`**) on the returned tree will be singletons. Changes to one will be reflected in all other occurrences of the same value (for example, **`ast.Add`**).

ast helpers

Apart from the node classes, the **`ast`** module defines these utility functions and classes for traversing abstract syntax trees:

ast.parse(*source*, *filename='<unknown>'*, *mode='exec'*, *, *type_comments=False*, *feature_version=None*, *optimize=-1*)

Parse the source into an AST node. Equivalent to `compile(source, filename, mode, flags=FLAGS_VALUE, optimize=optimize)`, where `FLAGS_VALUE` is `ast.PyCF_ONLY_AST` if `optimize <= 0` and `ast.PyCF_OPTIMIZED_AST` otherwise.

If `type_comments=True` is given, the parser is modified to check and return type comments as specified by **PEP 484** [https://peps.python.org/pep-0484/] and **PEP 526** [https://peps.python.org/pep-0526/]. This is equivalent to adding **`ast.PyCF_TYPE_COMMENTS`** to the flags passed to **`compile()`**. This will report syntax errors for misplaced type comments. Without this flag, type comments will be ignored, and the `type_comment` field on selected AST nodes will always be `None`. In addition, the locations of `# type: ignore` comments will be returned as the `type_ignores` attribute of **`Module`** (otherwise it is always an empty list).

Además, si `modo` es `'func_type'`, la sintaxis de entrada se modifica para corresponder a **PEP 484** [https://peps.python.org/pep-0484/] «comentarios de tipo de firma», por ejemplo `(str, int) -> List[str]`.

Setting `feature_version` to a tuple `(major, minor)` will result in a «best-effort» attempt to parse using that Python version's grammar. For example, setting `feature_version=(3, 9)` will attempt to disallow parsing of **`match`** statements. Currently `major` must equal to `3`. The lowest supported version is `(3, 7)` (and this may increase in future Python versions); the highest is `sys.version_info[0:2]`. «Best-effort» attempt means there is no guarantee that the parse (or success of the parse) is the same as when run on the Python version corresponding to `feature_version`.

If source contains a null character (`\0`), **`ValueError`** is raised.

Advertencia

Tenga en cuenta que analizar correctamente el código fuente en un objeto AST no garantiza que el código fuente proporcionado sea un código Python válido que se pueda ejecutar, ya que el paso de compilación puede lanzar más excepciones `SyntaxError`. Por ejemplo, la fuente `return 42` genera un nodo AST válido para una declaración de retorno, pero no se puede compilar solo (debe estar dentro de un nodo de función).

En particular, `ast.parse()` no realizará ninguna verificación de alcance, lo que hace el paso de compilación.

Advertencia

Es posible bloquear el intérprete de Python con una cadena de caracteres suficientemente grande/compleja debido a las limitaciones de profundidad de pila en el compilador AST de Python.

Distinto en la versión 3.8: Se agregaron `type_comments`, `mode='func_type'` y `feature_version`.

Distinto en la versión 3.13: The minimum supported version for `feature_version` is now `(3, 7)`. The `optimize` argument was added.

ast.unparse(*ast_obj*)

Analice un objeto `ast.AST` y genere una cadena con código que produciría un objeto `ast.AST` equivalente si se analiza con `ast.parse()`.

Advertencia

La cadena de código producida no será necesariamente igual al código original que generó el objeto `ast.AST` (sin ninguna optimización del compilador, como tuplas constantes / frozensets).

Advertencia

Intentar descomprimir una expresión muy compleja daría como resultado `RecursionError`.

Added in version 3.9.

ast.literal_eval(*node_or_string*)

Evalúa un nodo de expresión o una cadena de caracteres que contenga únicamente un literal de Python o visualizador de contenedor. La cadena o nodo proporcionado sólo puede consistir en las siguientes estructuras literales de Python: cadenas de caracteres, bytes, números, tuplas, listas, diccionarios, conjuntos, booleanos, `None` y `Ellipsis`.

Esto se puede usar para evaluar de forma segura las cadenas de caracteres que contienen valores de Python sin la necesidad de analizar los valores uno mismo. No es capaz de evaluar expresiones complejas arbitrariamente, por ejemplo, que involucran operadores o indexación.

Esta función había sido documentada como «segura» en el pasado sin definir lo que eso significaba. Eso era engañoso. Está diseñada específicamente no para ejecutar código Python, a diferencia de la más general `eval()`. No hay espacio de nombres, ni búsqueda de nombres, ni capacidad de llamada. Pero no está libre de ataques: Una entrada relativamente pequeña puede llevar a un agotamiento de la memoria o de la pila de C, haciendo colapsar el proceso. También existe la posibilidad de denegación de servicio por consumo excesivo de CPU en algunas entradas. Por lo tanto, no se recomienda llamarlo con datos no confiables.

Advertencia

Es posible bloquear el intérprete de Python debido a las limitaciones de profundidad de pila en el compilador AST de Python.

Puede generar `ValueError`, `TypeError`, `SyntaxError`, `MemoryError` y `RecursionError` dependiendo de la entrada mal formada.

Distinto en la versión 3.2: Ahora permite bytes y establece literales.

Distinto en la versión 3.9: Ahora admite la creación de conjuntos vacíos con `'set()'`.

Distinto en la versión 3.10: Para las entradas de cadena, los espacios iniciales y las tabulaciones ahora se eliminan.

ast.get_docstring(*node*, *clean=True*)

Retorna la cadena de caracteres de documentación del *node* dado (que debe ser un nodo `FunctionDef`, `AsyncFunctionDef`, `ClassDef`, o `Module`), o `None` si no tiene docstring. Si *clean* es verdadero, limpia la sangría del docstring con `inspect.cleandoc()`.

Distinto en la versión 3.5: `AsyncFunctionDef` ahora está soportada.

ast.get_source_segment(*source*, *node*, ***, *padded=False*)

Get source code segment of the *source* that generated *node*. If some location information (`lineno`, `end_lineno`, `col_offset`, or `end_col_offset`) is missing, return `None`.

Si *padded* es `True`, la primera línea de una declaración de varias líneas se rellenará con espacios para que coincidan con su posición original.

Added in version 3.8.

ast.fix_missing_locations(*node*)

When you compile a node tree with `compile()`, the compiler expects `lineno` and `col_offset` attributes for every node that supports them. This is rather tedious to fill in for generated nodes, so this helper adds these attributes recursively where not already set, by setting them to the values of the parent node. It works recursively starting at *node*.

ast.increment_lineno(*node*, *n=1*)

Incremente el número de línea y el número de línea final de cada nodo en el árbol comenzando en *node* por *n*. Esto es útil para «mover código» a una ubicación diferente en un archivo.

ast.copy_location(*new_node*, *old_node*)

Copy source location (**`lineno`**, **`col_offset`**, **`end_lineno`**, and **`end_col_offset`**) from *old_node* to *new_node* if possible, and return *new_node*.

ast.iter_fields(*node*)

Produce (*yield*) una tupla de `(fieldname, value)` para cada campo en `node._fields` que está presente en *node*.

ast.iter_child_nodes(*node*)

Cede todos los nodos secundarios directos de *node*, es decir, todos los campos que son nodos y todos los elementos de campos que son listas de nodos.

ast.walk(*node*)

Recursivamente produce todos los nodos descendientes en el árbol comenzando en *node* (incluido *node* en sí mismo), en ningún orden especificado. Esto es útil si solo desea modificar los nodos en su lugar y no le importa el contexto.

class ast.NodeVisitor

Una clase base de visitante de nodo que recorre el árbol de sintaxis abstracta y llama a una función de visitante para cada nodo encontrado. Esta función puede retornar un valor que se reenvía mediante el método **`visit()`**.

Esta clase está destinada a ser subclase, con la subclase agregando métodos de visitante.

visit(*node*)

Visita un nodo. La implementación predeterminada llama al método llamado `self.visit_classname` donde *classname* es el nombre de la clase de nodo, o **`generic_visit()`** si ese método no existe.

generic_visit(*node*)

Este visitante llama **`visit()`** en todos los hijos del nodo.

Tenga en cuenta que los nodos secundarios de los nodos que tienen un método de visitante personalizado no se visitarán a menos que el visitante llame **`generic_visit()`** o los visite a sí mismo.

visit_Constant(*node*)

Handles all constant nodes.

No use **`NodeVisitor`** si desea aplicar cambios a los nodos durante el recorrido. Para esto existe un visitante especial (**`NodeTransformer`**) que permite modificaciones.

Deprecated since version 3.8, removed in version 3.14: Methods **`visit_Num()`**, **`visit_Str()`**, **`visit_Bytes()`**, **`visit_NameConstant()`** and **`visit_Ellipsis()`** will not be called in Python 3.14+. Add the **`visit_Constant()`** method instead to handle all constant nodes.

class ast.NodeTransformer

Una subclase de **`NodeVisitor`** que recorre el árbol de sintaxis abstracta y permite la modificación de nodos.

La clase **`NodeTransformer`** recorrerá el AST y usará el valor de retorno de los métodos del visitante para reemplazar o eliminar el nodo anterior. Si el valor de retorno del método de visitante es `None`, el nodo se eliminará de su ubicación; de lo contrario, se reemplazará con el valor de retorno. El valor de retorno puede ser el nodo original, en cuyo caso no se realiza ningún reemplazo.

Aquí hay un transformador de ejemplo que reescribe todas las apariciones de búsquedas de nombres (`foo`) en `data['foo']`:

```
class RewriteName(NodeTransformer):

    def visit_Name(self, node):
        return Subscript(
            value=Name(id='data', ctx=Load()),
            slice=Constant(value=node.id),
            ctx=node.ctx
        )
```

Keep in mind that if the node you're operating on has child nodes you must either transform the child nodes yourself or call the `generic_visit()` method for the node first.

Para los nodos que formaban parte de una colección de declaraciones (que se aplica a todos los nodos de declaración), el visitante también puede retornar una lista de nodos en lugar de solo un nodo.

If **`NodeTransformer`** introduces new nodes (that weren't part of original tree) without giving them location information (such as `lineno`), `fix_missing_locations()` should be called with the new sub-tree to recalculate the location information:

```
tree = ast.parse('foo', mode='eval')
new_tree = fix_missing_locations(RewriteName().visi
```

Usualmente usas el transformador así:

```
node = YourTransformer().visit(node)
```

ast.dump(*node*, *annotate_fields=True*, *include_attributes=False*, *, *indent=None*, *show_empty=False*)

Retorna un volcado formateado del árbol en *node*. Esto es principalmente útil para propósitos de depuración. Si *annotate_fields* es verdadero (por defecto), la cadena de caracteres retornada mostrará los nombres y los valores de los campos. Si *annotate_fields* es falso, la cadena de resultados será más compacta omitiendo nombres de campo no ambiguos. Los atributos como los números de línea y las compensaciones de columna no se vuelcan de forma predeterminada. Si esto se desea, *include_attributes* se puede establecer en verdadero.

Si *indent* es un entero no negativo o una cadena de caracteres, entonces el árbol será impreso de forma linda con ese nivel de sangría. Un nivel de sangría de 0, negativo, o `""` solo insertará nuevas líneas. `None` (el valor por defecto) selecciona la representación de línea simple. Al usar un entero positivo se sangrará esa cantidad de espacios como sangría. Si *indent* es una cadena de caracteres (como `"\t"`), esa cadena se usa para sangrar cada nivel.

If *show_empty* is false (the default), optional empty lists will be omitted from the output. Optional `None` values are always omitted.

Distinto en la versión 3.9: Añadida la opción *indent*.

Distinto en la versión 3.13: Added the *show_empty* option.

```
>>> print(ast.dump(ast.parse("""\
... async def f():
...     await other_func()
... """), indent=4, show_empty=True))
Module(
    body=[
        AsyncFunctionDef(
            name='f',
            args=arguments(
                posonlyargs=[],
                args=[],
                kwonlyargs=[],
                    kw_defaults=[],
                    defaults=[]),
                body=[
                    Expr(
                        value=Await(
                            value=Call(
                                func=Name(id='other_fun
                                args=[],
                                keywords=[])))],
                decorator_list=[],
                type_params=[])],
        type_ignores=[])
```

Compiler flags

Los siguientes indicadores pueden pasarse a `compile()` para cambiar los efectos en la compilación de un programa:

ast.PyCF_ALLOW_TOP_LEVEL_AWAIT

> Habilita el soporte para `await`, `async for`, `async with` y comprensiones asíncronas de nivel superior.
>
> *Added in version 3.8.*

ast.PyCF_ONLY_AST

> Genera y retorna un árbol de sintaxis abstracto en lugar de retornar un objeto de código compilado.

ast.PyCF_OPTIMIZED_AST

> The returned AST is optimized according to the *optimize* argument in `compile()` or `ast.parse()`.
>
> *Added in version 3.13.*

ast.PyCF_TYPE_COMMENTS

> Habilita el soporte para comentarios de tipo de estilo **PEP 484** [https://peps.python.org/pep-0484/] y **PEP 526** [https://peps.python.org/pep-0526/] (`# type: <type>`, `# type:`

`ignore <stuff>`).

Added in version 3.8.

ast.compare(*a*, *b*, /, *, *compare_attributes=False*)

Recursively compares two ASTs.

compare_attributes affects whether AST attributes are considered in the comparison. If *compare_attributes* is `False` (default), then attributes are ignored. Otherwise they must all be equal. This option is useful to check whether the ASTs are structurally equal but differ in whitespace or similar details. Attributes include line numbers and column offsets.

Added in version 3.14.

Command-line usage

Added in version 3.9.

The **`ast`** module can be executed as a script from the command line. It is as simple as:

```
python -m ast [-m <mode>] [-a] [infile]
```

Las siguientes opciones son aceptadas:

-h, --help

Muestra el mensaje de ayuda y sale.

-m <mode>
--mode <mode>

Especifica qué tipo de código debe ser compilado, como el argumento *mode* en **`parse()`**.

--no-type-comments

No analizar los comentarios de tipo.

-a, --include-attributes

Incluye atributos como números de línea y sangrías.

-i <indent>
--indent <indent>

Sangría de nodos en AST (número de espacios).

--feature-version <version>

Python version in the format 3.x (for example, 3.10). Defaults to the current version of the interpreter.

Added in version 3.14.

-O <level>
--optimize <level>

Optimization level for parser. Defaults to no optimization.

Added in version 3.14.

--show-empty

Show empty lists and fields that are `None`. Defaults to not showing empty objects.

Added in version 3.14.

Si `infile` es especificado, su contenido es analizado a AST y mostrado en stdout. De otra forma, el contenido es leído desde stdin.

Ver también

Green Tree Snakes [https://greentreesnakes.readthedocs.io/], un recurso de documentación externo, tiene buenos detalles sobre cómo trabajar con Python AST.

ASTTokens [https://asttokens.readthedocs.io/en/latest/user-guide.html] anota ASTs de Python con la posición de tokens y texto en el código fuente que los genera. Esto es de ayuda para herramientas que hacen transformaciones de código fuente.

leoAst.py [https://leo-editor.github.io/leo-editor/appendices.html#leoast-py] unifies the token-based and parse-tree-based views of python programs by inserting two-way links between tokens and ast nodes.

LibCST [https://libcst.readthedocs.io/] analiza código como Árboles de Sintaxis Concreta que se ven como ASTs y mantienen todos los detalles de formato. Es útil para construir herramientas de refactor automáticas y linters.

Parso [https://parso.readthedocs.io] is a Python parser that supports error recovery and round-trip parsing for different Python versions (in multiple Python versions). Parso is also able to list multiple syntax errors in your Python file.

`symtable` — Access to the compiler's symbol tables

Código fuente: Lib/symtable.py [https://github.com/python/cpython/tree/3.14/Lib/symtable.py]

Symbol tables are generated by the compiler from AST just before bytecode is generated. The symbol table is responsible for calculating the scope of every identifier in the code. **`symtable`** provides an interface to examine these tables.

Generando tablas de símbolos

symtable.symtable(*code*, *filename*, *compile_type*)

Retorna la **`SymbolTable`** del nivel más alto para el código Python *code*. *filename* es el nombre del archivo conteniendo el código. *compile_type* es como el argumento *mode* de la función **`compile()`**.

Examinando la tabla de símbolos

class symtable.SymbolTableType

An enumeration indicating the type of a **`SymbolTable`** object.

MODULE = *"module"*

Used for the symbol table of a module.

FUNCTION = *"function"*

Used for the symbol table of a function.

CLASS = *"class"*

Used for the symbol table of a class.

The following members refer to different flavors of annotation scopes.

ANNOTATION = *"annotation"*

Used for annotations if `from __future__ import annotations` is active.

TYPE_ALIAS = *"type alias"*

Used for the symbol table of **`type`** constructions.

TYPE_PARAMETERS = *"type parameters"*

Used for the symbol table of generic functions or generic classes.

TYPE_VARIABLE = *"type variable"*

Used for the symbol table of the bound, the constraint tuple or the default value of a single type variable in the formal sense, i.e., a TypeVar, a TypeVarTuple or a ParamSpec object (the latter two do not support a bound or a constraint tuple).

Added in version 3.13.

class symtable.SymbolTable

Un espacio de nombres para el bloque. El constructor no es público.

get_type()

Return the type of the symbol table. Possible values are members of the **`SymbolTableType`** enumeration.

Distinto en la versión 3.12: Se agregaron `'annotation'`, `'TypeVar bound'`, `'type alias'` y `'type parameter'` como posibles valores de retorno.

Distinto en la versión 3.13: Return values are members of the **`SymbolTableType`** enumeration.

The exact values of the returned string may change in the future, and thus, it is recommended to use **`SymbolTableType`** members instead of hard-coded strings.

get_id()

Retorna el identificador de la tabla.

get_name()

Retorna el nombre de la tabla. Es el nombre de la clase si la tabla es para una clase, el nombre de la función si la tabla es para una función o `'top` si la tabla es global (**`get_type()`** retorna `'module`). Para ámbitos de parámetros de tipo (que se utilizan para clases genéricas, funciones y alias de tipo), es el nombre de la clase, función o alias de tipo subyacente. Para ámbitos de alias de tipo, es el nombre del alias de tipo. Para ámbitos enlazados **`TypeVar`**, es el nombre del `TypeVar`.

get_lineno()

Retorna el número de la primera línea en el bloque que esta tabla representa.

is_optimized()

Retorna `True` si los locales en esta tabla pueden ser optimizados.

is_nested()

Retorna `True` si el bloque es una clase o función anidadas.

has_children()

Retorna `True` si el bloque contiene espacios de nombres anidados en él. Estos pueden ser obtenidos con **`get_children()`**.

get_identifiers()

Retorna una vista de objeto que contiene los nombres de los símbolos en la tabla. Lee la documentación de vistas de objetos.

lookup(*name*)

Busca *name* en la tabla y retorna una instancia de **`Symbol`**.

get_symbols()

Retorna una lista de instancias de **`Symbol`** de los nombres en la tabla.

get_children()

Retorna una lista de las tablas de símbolos anidadas.

class symtable.Function

A namespace for a function or method. This class inherits from **`SymbolTable`**.

get_parameters()

Retorna una tupla conteniendo los nombres de los parámetros de esta función.

get_locals()

Retorna una tupla conteniendo los nombres de los locales en esta función.

get_globals()

Retorna una tupla conteniendo los nombres de los globales en esta función.

get_nonlocals()

Return a tuple containing names of explicitly declared nonlocals in this function.

get_frees()

Return a tuple containing names of free (closure) variables in this function.

class symtable.Class

A namespace of a class. This class inherits from **`SymbolTable`**.

get_methods()

Return a tuple containing the names of method-like functions declared in the class.

Here, the term "method" designates *any* function defined in the class body via **`def`** or **`async def`**.

Functions defined in a deeper scope (e.g., in an inner class) are not picked up by **`get_methods()`**.

For example:

```
>>> import symtable
>>> st = symtable.symtable('''
... def outer(): pass
...
... class A:
...    def f():
...        def w(): pass
...
...    def g(self): pass
...
...    @classmethod
...    async def h(cls): pass
...
...    global outer
...    def outer(self): pass
... ''', 'test', 'exec')
```

```
>>> class_A = st.get_children()[2]
>>> class_A.get_methods()
('f', 'g', 'h')
```

Although `A().f()` raises **`TypeError`** at runtime, `A.f` is still considered as a method-like function.

Deprecated since version 3.14, will be removed in version 3.16.

class symtable.Symbol

Una entrada en una **`SymbolTable`** correspondiente a un identificador en el código. El constructor no es público.

get_name()

Retorna el nombre del símbolo.

is_referenced()

Retorna `True` si el símbolo es usado en su bloque.

is_imported()

Retorna `True` si el símbolo es creado desde una instrucción *import*.

is_parameter()

Retorna `True` si el símbolo es un parámetro.

is_type_parameter()

Return `True` if the symbol is a type parameter.

Added in version 3.14.

is_global()

Retorna `True` si el símbolo es global.

is_nonlocal()

Retorna `True` si el símbolo es no local.

is_declared_global()

Retorna `True` si el símbolo es declarado global con una instrucción *global*.

is_local()

Retorna `True` si el símbolo es local a su bloque.

is_annotated()

Retorna `True` si el símbolo está anotado.

Added in version 3.6.

is_free()

Retorna `True` si el símbolo es referenciado en su bloque pero no asignado.

is_free_class()

Return *True* if a class-scoped symbol is free from the perspective of a method.

Consider the following example:

```
def f():
    x = 1  # function-scoped
    class C:
        x = 2  # class-scoped
        def method(self):
            return x
```

In this example, the class-scoped symbol `x` is considered to be free from the perspective of `C.method`, thereby allowing the latter to return *1* at runtime and not *2*.

Added in version 3.14.

is_assigned()

Retorna `True` si el símbolo es asignado en su bloque.

is_comp_iter()

Return `True` if the symbol is a comprehension iteration variable.

Added in version 3.14.

is_comp_cell()

Return `True` if the symbol is a cell in an inlined comprehension.

Added in version 3.14.

is_namespace()

Retorna `True` si la vinculación de nombres introduce un nuevo espacio de nombres.

Si el nombre es usado como objetivo de una instrucción *function* o *class* retornará verdadero.

Por ejemplo:

```
>>> table = symtable.symtable("def some_func():
>>> table.lookup("some_func").is_namespace()
True
```

Note que un solo nombre puede estar vinculado a varios objetos. Si el resultado es `True`, el nombre puede estar vinculado también a otros objetos, como un entero o una lista, esto no introduce un nuevo espacio de nombres.

get_namespaces()

Retorna una lista de espacios de nombres vinculados a este nombre.

get_namespace()

Retorna el espacio de nombre vinculado a este nombre. Si hay más de un espacio de nombre vinculado o ninguno a ese nombre se levanta un `ValueError`.

Command-Line Usage

Added in version 3.13.

The **`symtable`** module can be executed as a script from the command line.

```
python -m symtable [infile...]
```

Symbol tables are generated for the specified Python source files and dumped to stdout. If no input file is specified, the content is read from stdin.

`token` — Constants used with Python parse trees

Código fuente: Lib/token.py [https://github.com/python/cpython/tree/3.14/Lib/token.py]

Este módulo proporciona constantes que representan los valores numéricos de los nodos hoja del árbol de análisis (tokens terminales). Consulte el archivo `Grammar/Tokens` en la distribución de Python para conocer las definiciones de los nombres en el contexto de la gramática del lenguaje. Los valores numéricos específicos a los que se asignan los nombres pueden cambiar entre las versiones de Python.

El módulo también proporciona un mapeo de códigos numéricos a nombres y algunas funciones. Las funciones asemejan definiciones en los archivos Python C encabezados.

Note that a token's value may depend on tokenizer options. For example, a `"+"` token may be reported as either **`PLUS`** or **`OP`**, or a `"match"` token may be either **`NAME`** or **`SOFT_KEYWORD`**.

token.tok_name

Diccionario que mapea los valores numéricos de las constantes definidas en este módulo a cadenas de nombres, permitiendo una representación de árboles de sintaxis a ser generados más legible para humanos.

token.ISTERMINAL(*x*)

Retorna `True` para valores token terminales.

token.ISNONTERMINAL(*x*)

Retorna `True` para valores token no terminales.

token.ISEOF(*x*)

Retorna `True` si *x* es el marcador indicando el final del input.

Las constantes de token son:

token.NAME

Token value that indicates an identifier or keyword.

token.NUMBER

Token value that indicates a numeric literal

token.STRING

Token value that indicates a string or byte literal, excluding formatted string literals. The token string is not interpreted: it includes the surrounding quotation marks and the prefix (if given); backslashes are included literally, without processing escape sequences.

token.OP

A generic token value that indicates an operator or delimiter.

Detalles de implementación de CPython: This value is only reported by the `tokenize` module. Internally, the tokenizer uses exact token types instead.

token.COMMENT

Token value used to indicate a comment. The parser ignores `COMMENT` tokens.

token.NEWLINE

Token value that indicates the end of a logical line.

token.NL

Token value used to indicate a non-terminating newline. `NL` tokens are generated when a logical line of code is continued over multiple physical lines. The parser ignores `NL` tokens.

token.INDENT

Token value used at the beginning of a logical line to indicate the start of an indented block.

token.DEDENT

Token value used at the beginning of a logical line to indicate the end of an indented block.

token.FSTRING_START

Token value used to indicate the beginning of an f-string literal.

Detalles de implementación de CPython: The token string includes the prefix and the opening quote(s), but none of the contents of the literal.

token.FSTRING_MIDDLE

Token value used for literal text inside an f-string literal, including format specifications.

Detalles de implementación de CPython: Replacement fields (that is, the non-literal parts of f-strings) use the same tokens as other expressions, and are delimited by `LBRACE`, `RBRACE`, `EXCLAMATION` and `COLON` tokens.

token.FSTRING_END

Token value used to indicate the end of a f-string.

Detalles de implementación de CPython: The token string contains the closing quote(s).

token.TSTRING_START

Token value used to indicate the beginning of a template string literal.

Detalles de implementación de CPython: The token string includes the prefix and the opening quote(s), but none of the contents of the literal.

Added in version 3.14.

token.TSTRING_MIDDLE

Token value used for literal text inside a template string literal including format specifications.

Detalles de implementación de CPython: Replacement fields (that is, the non-literal parts of t-strings) use the same tokens as other expressions, and are delimited by `LBRACE`, `RBRACE`, `EXCLAMATION` and `COLON` tokens.

Added in version 3.14.

token.TSTRING_END

Token value used to indicate the end of a template string literal.

Detalles de implementación de CPython: The token string contains the closing quote(s).

Added in version 3.14.

token.ENDMARKER

Token value that indicates the end of input. Used in top-level grammar rules.

token.ENCODING

Valor de token que indica la codificación usada para decodificar los bytes de origen en texto. El primer token retornado por `tokenize.tokenize()` siempre será un token `ENCODING`.

Detalles de implementación de CPython: This token type isn't used by the C tokenizer but is needed for the `tokenize` module.

The following token types are not produced by the `tokenize` module, and are defined for special uses in the tokenizer or parser:

token.TYPE_IGNORE

Token value indicating that a `type: ignore` comment was recognized. Such tokens are produced instead of regular `COMMENT` tokens only with the `PyCF_TYPE_COMMENTS` flag.

token.TYPE_COMMENT

Token value indicating that a type comment was recognized. Such tokens are produced instead of regular `COMMENT` tokens only with the `PyCF_TYPE_COMMENTS` flag.

token.SOFT_KEYWORD

Token value indicating a soft keyword.

The tokenizer never produces this value. To check for a soft keyword, pass a `NAME` token's string to `keyword.issoftkeyword()`.

token.ERRORTOKEN

Token value used to indicate wrong input.

The `tokenize` module generally indicates errors by raising exceptions instead of emitting this token. It can also emit tokens such as `OP` or `NAME` with strings that are later rejected by the parser.

The remaining tokens represent specific operators and delimiters. (The `tokenize` module reports these as `OP`; see `exact_type` in the `tokenize` documentation for details.)

Token	**Value**
token.LPAR	"("
token.RPAR	")"
token.LSQB	"["
token.RSQB	"]"
token.COLON	":"
token.COMMA	","
token.SEMI	";"
token.PLUS	"+"
token.MINUS	"-"

'*'	token.STAR
'/'	token.SLASH
'|'	token.VBAR
'&'	token.AMPER
'<'	token.LESS
'>'	token.GREATER
'='	token.EQUAL
'.'	token.DOT
'%'	token.PERCENT
'{'	token.LBRACE
'}'	token.RBRACE
'=='	token.EQEQUAL
'!='	token.NOTEQUAL
'<='	token.LESSEQUAL
'>='	token.GREATEREQUAL
'~'	token.TILDE
'^'	token.CIRCUMFLEX
'<<'	token.LEFTSHIFT
'>>'	token.RIGHTSHIFT
'**'	token.DOUBLESTAR
'+='	token.PLUSEQUAL
'-='	token.MINEQUAL
'*='	token.STAREQUAL
'/='	token.SLASHEQUAL
'%='	token.PERCENTEQUAL
'&='	token.AMPEREQUAL
'|='	token.VBAREQUAL
'^='	token.CIRCUMFLEXEQUAL
'<<='	token.LEFTSHIFTEQUAL
'>>='	token.RIGHTSHIFTEQUAL
'**='	token.DOUBLESTAREQUAL
'//'	token.DOUBLESLASH
'//='	token.DOUBLESLASHEQUAL
'@'	token.AT
'@='	token.ATEQUAL
'->'	token.RARROW
'...'	token.ELLIPSIS
':='	token.COLONEQUAL
'!'	token.EXCLAMATION

The following non-token constants are provided:

token.N_TOKENS

The number of token types defined in this module.

token.EXACT_TOKEN_TYPES

A dictionary mapping the string representation of a token to its numeric code.

Added in version 3.8.

Distinto en la versión 3.5: Added **`AWAIT`** and **`ASYNC`** tokens.

Distinto en la versión 3.7: Agregados los tokens **`COMMENT`**, **`NL`** y **`ENCODING`**.

Distinto en la versión 3.7: Removed **`AWAIT`** and **`ASYNC`** tokens. «async» and «await» are now tokenized as **`NAME`** tokens.

Distinto en la versión 3.8: Added **`TYPE_COMMENT`**, **`TYPE_IGNORE`**, **`COLONEQUAL`**. Added **`AWAIT`** and **`ASYNC`** tokens back (they're needed to support parsing older Python versions for **`ast.parse()`** with `feature_version` set to 6 or lower).

Distinto en la versión 3.12: Added **`EXCLAMATION`**.

Distinto en la versión 3.13: Removed **`AWAIT`** and **`ASYNC`** tokens again.

`keyword` — Testing for Python keywords

Código fuente: Lib/keyword.py [https://github.com/python/cpython/tree/3.14/Lib/keyword.py]

Este módulo permite a un programa Python determinar si una cadena de caracteres es una palabra clave o palabra clave suave.

keyword.iskeyword(*s*)

Retorna `True` si *s* es una palabra clave Python.

keyword.kwlist

Secuencia que contiene todos las palabras clave definidos para el intérprete. Si cualquier palabra clave es definida para estar activa sólo cuando las declaraciones particulares **`__future__`** están vigentes, estas se incluirán también.

keyword.issoftkeyword(*s*)

Retorna `True` si *s* es una palabra clave suave de Python.

Added in version 3.9.

keyword.softkwlist

Secuencia que contiene todos las palabras clave suaves definidas para el intérprete. Si cualquier palabra clave suave es definida para estar activa sólo cuando las declaraciones particulares **`__future__`** están vigentes, estas se incluirán también.

Added in version 3.9.

`tokenize` — Tokenizer for Python source

Código fuente: Lib/tokenize.py [https://github.com/python/cpython/tree/3.14/Lib/tokenize.py]

The `tokenize` module provides a lexical scanner for Python source code, implemented in Python. The scanner in this module returns comments as tokens as well, making it useful for implementing «pretty-printers», including colorizers for on-screen displays.

Para simplificar el manejo de flujos de tokens, todos los tokens operator y delimiter y `Ellipsis` se retorna usando el tipo genérico `OP`. El tipo exacto se puede determinar usando la propiedad `exact_type` en la named tuple retornada por `tokenize.tokenize()`.

Advertencia

Ten en cuenta que las funciones en este módulo están diseñadas únicamente para analizar código Python sintácticamente válido (código que no genera errores al ser analizado usando `ast.parse()`). El comportamiento de las funciones en este módulo es **indefinido** cuando se proporciona código Python no válido y puede cambiar en cualquier momento.

Convirtiendo la entrada en *tokens*

El punto de entrada principal es un generador:

tokenize.tokenize(*readline*)

El generador `tokenize()` requiere un argumento, *readline*, que debe ser un objeto invocable que provee la misma interfaz que el método `io.IOBase.readline()` de los objetos archivos. Cada llamada a la función debe retornar una línea de entrada como bytes.

El generador produce una tupla con los siguientes 5 miembros: El tipo de token, la cadena del token, una tupla `(srow, scol)` de enteros especificando la fila y columna donde el token empieza en el código, una `(erow, ecol)` de enteros especificando la fila y columna donde el token acaba en el código, y la línea en la que se encontró el token. La línea pasada (el último elemento de la tupla) es la línea *física*. La tupla se retorna como una named tuple con los campos: `type string start end line`.

La named tuple retorna tiene una propiedad adicional llamada `exact_type` que contiene el tipo de operador exacto para tokens `OP`. Para todos los otros tipos de token, `exact_type` es el valor del campo `type` de la tupla con su respectivo nombre.

Distinto en la versión 3.1: Añadido soporte para tuplas con nombre.

Distinto en la versión 3.3: Añadido soporte para `exact_type`.

`tokenize()` determina la codificación del fichero buscando una marca BOM UTF-8 o una *cookie* de codificación, de acuerdo con **PEP 263** [https://peps.python.org/pep-0263/].

tokenize.generate_tokens(*readline*)

Convertir a tokens una fuente leyendo cadenas unicode en lugar de bytes.

Como `tokenize()`, el argumento *readline* es un invocable que retorna una sola línea de entrada. Sin embargo, `generate_tokens()` espera que *readline* retorne un objeto *str* en lugar de *bytes*.

El resultado es un iterador que produce tuplas con nombre, exactamente como `tokenize()`. No produce un token

`ENCODING`.

All constants from the `token` module are also exported from **`tokenize`**.

Otra función se encarga de invertir el proceso de conversión. Esto es útil para crear herramientas que convierten a tokens un script, modificar el flujo de token, y escribir el script modificado.

tokenize.untokenize(*iterable*)

Convierte los *tokens* de vuelta en código fuente Python. El *iterable* debe retornar secuencias con al menos dos elementos, el tipo de *token* y la cadena del *token*. Cualquier otro elemento de la secuencia es ignorado.

The result is guaranteed to tokenize back to match the input so that the conversion is lossless and round-trips are assured. The guarantee applies only to the token type and token string as the spacing between tokens (column positions) may change.

Retorna bytes, codificados usando el token `ENCODING`, que es el primer elemento de la secuencia retornada por `tokenize()`. Si no hay un token de codificación en la entrada, retorna una cadena.

`tokenize()` necesita detectar la codificación de los ficheros fuente que convierte a tokens. La función que utiliza para hacer esto está disponible como:

tokenize.detect_encoding(*readline*)

La función `detect_encoding()` se usa para detectar la codificación que debería usarse al leer un fichero fuente Python. Requiere un argumento, *readline*, del mismo modo que el generador `tokenize()`.

Llamará a *readline* un máximo de dos veces, retornando la codificación usada, como cadena, y una lista del resto de líneas leídas, no descodificadas de *bytes*.

Detecta la codificación a partir de la presencia de una marca BOM UTF-8 o una *cookie* de codificación, como se especifica en **PEP 263** [https://peps.python.org/pep-0263/]. Si ambas están presentes pero en desacuerdo, se lanzará un `SyntaxError`. Resaltar que si se encuentra la marca BOM, se retornará `'utf-8-sig'` como codificación.

Si no se especifica una codificación, por defecto se retornará `'utf-8'`.

Usa `open()` para abrir ficheros fuente Python: Ésta utiliza `detect_encoding()` para detectar la codificación del fichero.

tokenize.open(*filename*)

Abrir un fichero en modo sólo lectura usando la codificación detectada por `detect_encoding()`.

Added in version 3.2.

exception tokenize.TokenError

Lanzada cuando una expresión o docstring que puede separarse en más líneas no está completa en el fichero, por ejemplo:

```
"""Beginning of
docstring
```

o:

```
[1,
 2,
 3
```

Uso como línea de comandos

Added in version 3.3.

The **`tokenize`** module can be executed as a script from the command line. It is as simple as:

```
python -m tokenize [-e] [filename.py]
```

Se aceptan las siguientes opciones:

-h, --help
    muestra el mensaje de ayuda y sale

-e, --exact
    muestra los nombres de token usando el tipo exacto

Si se especifica `filename.py`, se convierte su contenido a tokens por la salida estándar. En otro caso, se convierte la entrada estándar.

Ejemplos

Ejemplo de un script que transforma literales float en objetos Decimal:

```
from tokenize import tokenize, untokenize, NUMBER, STRIN
from io import BytesIO

def decistmt(s):
    """Substitute Decimals for floats in a string of sta

    >>> from decimal import Decimal
    >>> s = 'print(+21.3e-5*-.1234/81.7)'
    >>> decistmt(s)
    "print (+Decimal ('21.3e-5')*-Decimal ('.1234')/Deci

    The format of the exponent is inherited from the pla
    Known cases are "e-007" (Windows) and "e-07" (not Wi
    we're only showing 12 digits, and the 13th isn't clo
    rest of the output should be platform-independent.

    >>> exec(s)  #doctest: +ELLIPSIS
    -3.21716034272e-0...7
    Output from calculations with Decimal should be ider
    platforms.

    >>> exec(decistmt(s))
    -3.217160342717258261933904529E-7
    """
    result = []
    g = tokenize(BytesIO(s.encode('utf-8')).readline)  #
    for toknum, tokval, _, _, _ in g:
        if toknum == NUMBER and '.' in tokval:  # replac
            result.extend([
                (NAME, 'Decimal'),
                (OP, '('),
                (STRING, repr(tokval)),
                (OP, ')')
            ])
        else:
            result.append((toknum, tokval))
    return untokenize(result).decode('utf-8')
```

Ejemplo de conversión desde la línea de comandos. El *script*:

```
def say_hello():
    print("Hello, World!")

say_hello()
```

se convertirá en la salida siguiente, donde la primera columna es el rango de coordenadas línea/columna donde se encuentra el token, la segunda columna es el nombre del token, y la última columna es el valor del token, si lo hay

```
$ python -m tokenize hello.py
0,0-0,0:            ENCODING       'utf-8'
1,0-1,3:            NAME           'def'
1,4-1,13:           NAME           'say_hello'
1,13-1,14:          OP             '('
1,14-1,15:          OP             ')'
```

```
1,15-1,16:          OP             ':'
1,16-1,17:          NEWLINE        '\n'
2,0-2,4:            INDENT         '    '
2,4-2,9:            NAME           'print'
2,9-2,10:           OP             '('
2,10-2,25:          STRING         '"Hello, World!"'
2,25-2,26:          OP             ')'
2,26-2,27:          NEWLINE        '\n'
3,0-3,1:            NL             '\n'
4,0-4,0:            DEDENT         ''
4,0-4,9:            NAME           'say_hello'
4,9-4,10:           OP             '('
4,10-4,11:          OP             ')'
4,11-4,12:          NEWLINE        '\n'
5,0-5,0:            ENDMARKER      ''
```

Los nombres de tipos de token exactos se pueden mostrar con la opción **`-e`**:

```
$ python -m tokenize -e hello.py
0,0-0,0:            ENCODING       'utf-8'
1,0-1,3:            NAME           'def'
1,4-1,13:           NAME           'say_hello'
1,13-1,14:          LPAR           '('
1,14-1,15:          RPAR           ')'
1,15-1,16:          COLON          ':'
1,16-1,17:          NEWLINE        '\n'
2,0-2,4:            INDENT         '    '
2,4-2,9:            NAME           'print'
2,9-2,10:           LPAR           '('
2,10-2,25:          STRING         '"Hello, World!"'
2,25-2,26:          RPAR           ')'
2,26-2,27:          NEWLINE        '\n'
3,0-3,1:            NL             '\n'
4,0-4,0:            DEDENT         ''
4,0-4,9:            NAME           'say_hello'
4,9-4,10:           LPAR           '('
4,10-4,11:          RPAR           ')'
4,11-4,12:          NEWLINE        '\n'
```

```
5,0-5,0:            ENDMARKER      ''
```

Ejemplo de tokenización de un fichero programáticamente, leyendo cadenas unicode en lugar de *bytes* con **`generate_tokens()`**:

```
import tokenize

with tokenize.open('hello.py') as f:
    tokens = tokenize.generate_tokens(f.readline)
    for token in tokens:
        print(token)
```

O leyendo *bytes* directamente con **`tokenize()`**:

```
import tokenize

with open('hello.py', 'rb') as f:
    tokens = tokenize.tokenize(f.readline)
    for token in tokens:
        print(token)
```

`tabnanny` — Detection of ambiguous indentation

Código fuente: Lib/tabnanny.py [https://github.com/python/cpython/tree/3.14/Lib/tabnanny.py]

Por el momento, este módulo está pensado para ser llamado como un script. Sin embargo, es posible importarlo en un IDE y usar la función `check()` que se describe a continuación.

Nota

Es probable que la API proporcionada por este módulo cambie en versiones futuras; dichos cambios pueden no ser compatibles con versiones anteriores.

tabnanny.check(*file_or_dir*)

Si *file_or_dir* es un directorio y no un enlace simbólico, desciende recursivamente en el árbol de directorios nombrado por *file_or_dir*, verificando todos los archivos `.py` al mismo tiempo. Si *file_or_dir* es un archivo fuente normal de Python, se comprueba si hay problemas relacionados con los espacios en blanco. Los mensajes de diagnóstico se escriben en la salida estándar mediante la función `print()`.

tabnanny.verbose

Marcador que indica si se deben imprimir mensajes detallados. Esto se incrementa con la opción `-v` si se llama como un script.

tabnanny.filename_only

Marcador que indica si se deben imprimir solo los nombres de archivo de los archivos que contienen problemas relacionados con los espacios en blanco. Esto se establece como verdadero con la opción `-q` si se llama como un script.

exception tabnanny.NannyNag

Invocada por `process_tokens()` sí detecta una indentación ambigua. Capturada y gestionada en `check()`.

tabnanny.process_tokens(*tokens*)

Esta función es utilizada por `check()` para procesar los tokens generados por el módulo `tokenize`.

Ver también

Módulo `tokenize`

Escáner léxico para código fuente Python.

`pyclbr` — Python module browser support

Código fuente: Lib/pyclbr.py [https://github.com/python/cpython/tree/3.14/Lib/pyclbr.py]

The **`pyclbr`** module provides limited information about the functions, classes, and methods defined in a Python-coded module. The information is sufficient to implement a module browser. The information is extracted from the Python source code rather than by importing the module, so this module is safe to use with untrusted code. This restriction makes it impossible to use this module with modules not implemented in Python, including all standard and optional extension modules.

pyclbr.readmodule(*module*, *path=None*)

> Retorna un diccionario que asigna nombres de clase a nivel de módulo con descriptores de clase. Si es posible, se incluyen descriptores para las clases base importadas. El parámetro *module* es una cadena con el nombre del módulo que se va a leer; puede ser el nombre de un módulo dentro de un paquete. Si se indica, *path* es una secuencia de rutas de directorios antepuesto a `sys.path`, que se utiliza para localizar el código fuente del módulo.
>
> Esta función es la interfaz original y sólo se mantiene por compatibilidad. Retorna una versión filtrada de lo siguiente.

pyclbr.readmodule_ex(*module*, *path=None*)

> Retorna un árbol basado en diccionarios que contiene un descriptor de función o clase para cada función y clase definida en el módulo con una instrucción `def` o `class`. El diccionario retornado asigna nombres de clase y función a nivel de módulo con sus descriptores. Los objetos anidados se

introducen en el diccionario hijo de su elemento padre. Al igual que con *readmodule*, *module* nombra el módulo que se va a leer y *path* se antepone a sys.path. Si el módulo que se lee es un paquete, el diccionario retornado tiene una clave `'__path__'` cuyo valor es una lista que contiene la ruta del paquete.

Added in version 3.7: Descriptores para definiciones anidadas. Se accede a ellos a través del nuevo atributo *children*. Cada uno tiene un nuevo atributo *parent*.

Los descriptores retornados por estas funciones son instancias de las clases Function y Class. No se espera que los usuarios creen instancias de estas clases.

Objetos Function

class pyclbr.Function

Class **`Function`** instances describe functions defined by def statements. They have the following attributes:

file

Nombre del archivo en el cual la función está definida.

module

El nombre del módulo que define la función descrita.

name

El nombre de la función.

lineno

El número de línea el en archivo donde inicia la definición.

parent

For top-level functions, `None`. For nested functions, the parent.

Added in version 3.7.

children

A `dictionary` mapping names to descriptors for nested functions and classes.

Added in version 3.7.

is_async

`True` for functions that are defined with the `async` prefix, `False` otherwise.

Added in version 3.10.

Objetos Class

class pyclbr.Class

Class **`Class`** instances describe classes defined by class statements. They have the same attributes as `Functions` and two more.

file

Nombre del archivo en el que la clase está definida.

module

Nombre del módulo que define la clase descrita.

name

El nombre de la clase.

lineno

El número de línea el en archivo donde inicia la definición.

parent

For top-level classes, `None`. For nested classes, the parent.

Added in version 3.7.

children

Un diccionario asignando nombres con descriptores para las clases y funciones anidadas.

Added in version 3.7.

super

A list of `Class` objects which describe the immediate base classes of the class being described. Classes which are named as superclasses but which are not discoverable by `readmodule_ex()` are listed as a string with the class name instead of as `Class` objects.

methods

A `dictionary` mapping method names to line numbers. This can be derived from the newer `children` dictionary, but remains for back-compatibility.

`py_compile` — Compile Python source files

Código fuente Lib/py_compile.py [https://github.com/python/cpython/tree/3.14/Lib/py_compile.py]

The **`py_compile`** module provides a function to generate a byte-code file from a source file, and another function used when the module source file is invoked as a script.

Aunque no es necesario seguidamente, esta función puede ser útil cuando se instalan módulos para uso compartido, especialmente si algunos de los usuarios pueden no tener permisos para escribir el archivo caché de bytes en el directorio conteniendo el código fuente.

exception py_compile.PyCompileError

Cuando un error ocurre mientras se intenta compilar el archivo, se lanza una excepción.

py_compile.compile(*file*, *cfile=None*, *dfile=None*, *doraise=False*, *optimize=-1*, *invalidation_mode=PycInvalidationMode.TIMESTAMP*, *quiet=0*)

Compile a source file to byte-code and write out the byte-code cache file. The source code is loaded from the file named *file*. The byte-code is written to *cfile*, which defaults to the **PEP 3147** [https://peps.python.org/pep-3147/]/**PEP 488** [https://peps.python.org/pep-0488/] path, ending in `.pyc`. For example, if *file* is `/foo/bar/baz.py` *cfile* will default to `/foo/bar/__pycache__/baz.cpython-32.pyc` for Python 3.2. If *dfile* is specified, it is used instead of *file* as the name of the source file from which source lines are obtained for display in exception tracebacks. If *doraise* is true, a **`PyCompileError`** is raised when an error is encountered while compiling *file*. If *doraise* is false (the default), an error string is written to `sys.stderr`, but no exception is raised. This function returns the path to byte-compiled file, i.e. whatever *cfile* value was used.

Los argumentos *doraise* y *quiet* determinan cómo los errores son gestionados mientras se compila el archivo. Si *quiet* es 0 o 1, y *doraise* es falso, la conducta por defecto es habilitada: un error cadena de caracteres es escrito a `sys.stderr`, y la función retorna `None` en vez de una ruta. Si *doraise* es verdadero, se lanzará un **`PyCompileError`**. Sin embargo si *quiet* es 2, ningún mensaje es escrito y *doraise* no tiene efecto.

Si la ruta que *cfile* se convierte (sea especificada explícitamente o computada) es un enlace simbólico o archivo no regular, se lanzará **`FileExistsError`**. Esto es para actuar como una advertencia que la importación convertirá esas rutas en archivos regulares si ésta tiene permitido escribir archivos compilados en bytes a esas rutas. Este es un efecto secundario de importar usando el renombramiento de archivos para colocar el archivo de bytes compilado final en el lugar para prevenir problemas de escritura en archivos simultáneos.

optimize controla el nivel de optimización y si es pasado a la función construida **`compile()`**. El predeterminado de `-1` selecciona el nivel de optimización del intérprete actual.

invalidation_mode debería ser un miembro del enum **`PycInvalidationMode`** y controla cómo el caché de código de bytes generado es invalidado en el tiempo de ejecución. El predeterminado es **`PycInvalidationMode.CHECKED_HASH`** si la variable de entorno **`SOURCE_DATE_EPOCH`** está establecida, de otra manera el predeterminado es **`PycInvalidationMode.TIMESTAMP`**.

Distinto en la versión 3.2: Cambiado el valor por defecto de *cfile* para que cumpla **PEP 3147** [https://peps.python.org/pep-3147/]. El por defecto anterior era *file* + `'c'` (`'o'` si la optimización estaba habilitada). También agregado el parámetro *optimize*.

Distinto en la versión 3.4: Se cambió el código para usar `importlib` para la escritura del archivo almacenado de código de bytes. Esto significa que la semántica de la creación/escritura del archivo ahora coincide con lo que `importlib` hace, por ejemplo permisos, semántica de escribir-y-mover, etc. Además se agregó la consideración de que `FileExistsError` es lanzado si *cfile* es un enlace simbólico o un archivo no regular.

Distinto en la versión 3.7: El parámetro *invalidation_mode* fue agregado como se especificó en **PEP 552** [https://peps.python.org/pep-0552/]. Si la variable de entorno `SOURCE_DATE_EPOCH` se establece, *invalidation_mode* será forzada a `PycInvalidationMode.CHECKED_HASH`.

Distinto en la versión 3.7.2: La variable de entorno `SOURCE_DATE_EPOCH` ya no sobreescribe el valor del argumento *invalidation_mode*, y en vez de eso determina su valor por defecto.

Distinto en la versión 3.8: El parámetro *quiet* fue agregado.

class py_compile.PycInvalidationMode

An enumeration of possible methods the interpreter can use to determine whether a bytecode file is up to date with a source file. The `.pyc` file indicates the desired invalidation mode in its header. See Invalidación del código de bytes en caché for more information on how Python invalidates `.pyc` files at runtime.

Added in version 3.7.

TIMESTAMP

El archivo `.pyc` incluye una marca de tiempo y tamaño del archivo fuente, el cual Python comparará contra los metadatos del archivo fuente durante el tiempo de ejecución para determinar si el archivo `.pyc` necesita ser regenerado.

CHECKED_HASH

El archivo `.pyc` incluye un hash del contenido del archivo fuente, el cual Python comparará contra la fuente durante el tiempo de ejecución para determinar si el archivo `.pyc` necesita ser regenerado.

UNCHECKED_HASH

Como **`CHECKED_HASH`**, el archivo `.pyc` incluye un has del contenido del archivo fuente. Sin embargo, Python asumirá durante el tiempo de ejecución que el archivo `.pyc` está actualizado y no validará el `.pyc` contra el archivo fuente en lo absoluto.

Esta opción es útil cuando los `.pucs` se mantienen actualizados al día por algún sistema externo a Python como un sistema de compilación.

Interfaz de línea de comandos

Este módulo puede invocarse como un script para compilar varios archivos fuente. Los archivos nombrados en *filenames* se compilan y el código de bytes resultante se almacena en caché de la manera normal. Este programa no busca en una estructura de directorio para localizar archivos fuente; solo compila archivos nombrados explícitamente. El estado de salida es distinto de cero si uno de los archivos no se pudo compilar.

<file> ... <fileN>
-

Los argumentos posicionales son archivos para compilar. Si `-` es el único parámetro, la lista de archivos se toma de la entrada estándar.

-q, --quiet

Suprime la salida de errores.

Distinto en la versión 3.2: Agregado soporte para `-`.

Distinto en la versión 3.10: Agregado soporte para `-q`.

Ver también

Módulo `compileall`
Utilidades para compilar todos los archivos fuente Python en el árbol del directorio.

`compileall` — Byte-compile Python libraries

Código fuente: Lib/compileall.py [https://github.com/python/cpython/tree/3.14/Lib/compileall.py]

Este módulo proporciona algunas funciones de utilidad para admitir la instalación de bibliotecas Python. Estas funciones compilan archivos fuente de Python en un árbol de directorios. Este módulo se puede usar para crear los archivos de código de bytes almacenados en caché en el momento de la instalación de la biblioteca, que los hace disponibles para su uso incluso por usuarios que no tienen permiso de escritura en los directorios de la biblioteca.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See Plataformas WebAssembly for more information.

Uso de la línea de comandos

Este módulo puede funcionar como un *script* (usando **python -m compileall**) para compilar fuentes de Python.

directory ...
file ...

> Los argumentos posicionales son archivos para compilar o directorios que contienen archivos fuente, recorridos recursivamente. Si no se proporciona ningún argumento, se comporta como si la línea de comando fuera `-l` *`<directories from sys.path>`*.

-l

No se recurre en subdirectorios, solo compila archivos de código fuente contenidos en directorios nombrados o implícitos.

-f

Forzar la reconstrucción incluso si las marcas de tiempo están actualizadas.

-q

No imprimir la lista de archivos compilados. Si se pasa una vez, los mensajes de error se imprimirán. Si se pasa dos veces, (`-qq`), se suprime toda la salida.

-d destdir

Directorio antepuesto a la ruta de cada archivo que está siendo compilado. Esto aparecerá en las devoluciones de tiempo de compilación, y también se compila en el archivo de código de bytes, donde se usará en las devoluciones de seguimiento y otros mensajes en casos donde el archivo fuente no existe al momento en que el archivo de código de bytes se ejecuta.

-s strip_prefix

Remove the given prefix from paths recorded in the `.pyc` files. Paths are made relative to the prefix.

This option can be used with `-p` but not with `-d`.

-p prepend_prefix

Prepend the given prefix to paths recorded in the `.pyc` files. Use `-p /` to make the paths absolute.

This option can be used with `-s` but not with `-d`.

-x regex

regex se usa para buscar la ruta completa a cada archivo considerado para compilación, y si la regex produce una coincidencia, se omite el archivo.

-i list

Leer el archivo `list` y cada línea que contiene la lista de archivos y directorios a compilar. Si `list` es `-`, leer líneas desde `stdin`.

-b

Escribir los archivos de código de byte en las locaciones y nombres de herencia, que pueden sobreescribir los archivos de código de bytes creado para otra versión de Python. El comportamiento por defecto es escribir archivos en sus locaciones y nombres **PEP 3147** [https://peps.python.org/pep-3147/], lo cual permite que archivos de código de byte de versiones múltiples de Python coexistan.

-r

Controlar el nivel máximo de recurrencia para subdirectorios. Si esto está dado, la opción `-l` no se tendrá en cuenta. **python -m compileall <directory> -r 0** es equivalente a **python -m compileall <directory> -l**.

-j N

Use *N* workers to compile the files within the given directory. If `0` is used, then the result of **`os.process_cpu_count()`** will be used.

--invalidation-mode [timestamp|checked-hash|unchecked-hash]

Controlar cómo los archivos de código de byte generados se invalidan al momento de ejecución. El valor `timestamp` significa que se generarán los archivos `.pyc` con la marca de tiempo fuente y el tamaño insertados. Los valores `checked-hash` y `unchecked-hash` generan *pycs* basados en hash. Los *pycs* basados en hash del archivo insertan un hash de los contenidos del archivo fuente, en lugar de una marca de tiempo. Véase Invalidación del código de bytes en caché para mayor información sobre cómo Python valida archivos de cache de código de bytes. El valor por defecto es `timestamp` si la variable de entorno **`SOURCE_DATE_EPOCH`** no está definida, y `checked-hash` si la variable de entorno

`SOURCE_DATE_EPOCH` está definida.

-o level

Compila con el nivel de optimización dado. Puede usarse varias veces para compilar varios niveles a la vez (por ejemplo, `compileall -o 1 -o 2`).

-e dir

Ignora los enlaces simbólicos que apuntan fuera del directorio dado.

--hardlink-dupes

Si dos archivos `.pyc` con diferente nivel de optimización tienen el mismo contenido, use enlaces físicos para consolidar los archivos duplicados.

Distinto en la versión 3.2: Se agregaron las opciones `-i`, `-b` y `-h`.

Distinto en la versión 3.5: Se agregaron las opciones `-j`, `-r`, and `-qq`. La opción `-q` se cambió a un valor multinivel. `-b` siempre producirá un archivo de código de byte que termina en `.pyc`, nunca `.pyo`.

Distinto en la versión 3.7: Se agregó la opción `--invalidation-mode`.

Distinto en la versión 3.9: Se agregaron las opciones `-s`, `-p`, `-e` y `--hardlink-dupes`. Se elevó el límite de recursividad predeterminado de 10 a `sys.getrecursionlimit()`. Se agregó la posibilidad de especificar la opción `-o` varias veces.

No hay opción de línea de comando para controlar el nivel de optimización que usa la función `compile()` porque el intérprete de Python en sí mismo ya proporciona la opción: **python -O -m compileall**.

De manera similar, la función `compile()` respeta la configuración `sys.pycache_prefix`. El cache de código de byte generado sólo será útil si `compile()` se ejecuta con el mismo `sys.pycache_prefix` (si es que existe alguno) que se utilizará en el momento de ejecución.

Funciones públicas

compileall.compile_dir(*dir*, *maxlevels=sys.getrecursionlimit()*, *ddir=None*, *force=False*, *rx=None*, *quiet=0*, *legacy=False*, *optimize=-1*, *workers=1*, *invalidation_mode=None*, ***, *stripdir=None*, *prependdir=None*, *limit_sl_dest=None*, *hardlink_dupes=False*)

Descender recursivamente el árbol de directorio invocado por *dir*, compilando todos los archivos `.py` que encuentra en el camino. Devolver un valor verdadero si todos los archivos se compilan exitosamente, y un valor falso en el caso contrario.

El parámetro *maxlevels* se usar para limitar la profundidad de la recursión; toma como valor predeterminado `sys.getrecursionlimit()`.

Si *ddir* está dado, se antepone a la ruta de cada archivo que se compila para usar en los rastreos de tiempo de compilación, y también se compilar en el archivo código de byte, donde se usarán en trazas y otros mensajes en casos donde el archivo fuente no existe en el momento cuando el archivo de código de byte se ejecuta.

Si *force* es verdadero, los módulos se re-compilan aun cuando las marcas de tiempo están actualizadas.

Si se proporciona *rx*, se llama a su método `search` en la ruta completa a cada archivo considerado para compilación, y si retorna un valor verdadero, se omite el archivo. Esto se puede usar para excluir archivos que coincidan con una expresión regular, dada como un objeto re.Pattern.

Si *quiet* es `False` o `0` (el valor predeterminado), los nombres de archivo y otra información se imprimen en salida estándar. Si se configura en `1`, solo se imprimen los errores. Si se configura en `2`, se suprime toda la salida.

Si *legacy* es verdadero, los archivos de código de byte se escriben en sus locaciones y nombres de herencia, que pueden sobreescribir los archivos de código de byte creado por otra versión de Python. El comportamiento por defecto es escribir archivos en sus locaciones y nombres **PEP 3147** [https://peps.python.org/pep-3147/], lo cual permite que archivos de código de byte de múltiples versiones de Python coexistan.

optimize especifica el nivel de optimización para el compilador. Se pasa a una función incorporada **`compile()`**. Acepta también una secuencia de niveles de optimización que conducen a múltiples compilaciones de un archivo `.py` en una llamada.

El argumento *workers* especifica cuántos workers se usan para compilar archivos en paralelo. El comportamiento por defecto es no usar múltiples workers. Si la plataforma no puede usar workers múltiples y el argumento *workers* está dado, la compilación secuencial se usará como *fallback*. Si *workers* es 0, el número de núcleos se usa en el sistema. Si *workers* es menor que `0`, se genera un **`ValueError`**.

invalidation_mode debería ser un miembro de la enumeración **`py_compile.PycInvalidationMode`** y controla cómo se invalidan los pycs generados en el momento de ejecución.

Los argumentos *stripdir*, *prependdir* y *limit_sl_dest* corresponden a las opciones `-s`, `-p` y `-e` descritas anteriormente. Pueden especificarse como `str`, o **`os.PathLike`**.

Si *hardlink_dupes* es verdadero y dos archivos `.pyc` con diferente nivel de optimización tienen el mismo contenido, use enlaces físicos para consolidar los archivos duplicados.

Distinto en la versión 3.2: Se agregó el parámetro *legacy* y *optimize*.

Distinto en la versión 3.5: Se agregó el parámetro *workers*.

Distinto en la versión 3.5: El parámetro *quiet* se cambió a un valor multinivel.

Distinto en la versión 3.5: El parámetro *legacy* solo escribe archivos `.pyc`, no archivos `.pyo`, no import cuál sea el valor de *optimize*.

Distinto en la versión 3.6: Acepta un path-like object.

Distinto en la versión 3.7: Se agregó el parámetro *invalidation_mode*.

Distinto en la versión 3.7.2: The *invalidation_mode* parameter's default value is updated to `None`.

Distinto en la versión 3.8: Configurar *workers* a 0 ahora elige el número óptimo de núcleos.

Distinto en la versión 3.9: Se agregaron los argumentos *stripdir*, *prependdir*, *limit_sl_dest* y *hardlink_dupes*. El valor predeterminado de *maxlevels* se cambió de `10` a `sys.getrecursionlimit()`

compileall.compile_file(*fullname, ddir=None, force=False, rx=None, quiet=0, legacy=False, optimize=-1, invalidation_mode=None, *, stripdir=None, prependdir=None, limit_sl_dest=None, hardlink_dupes=False*)

Compilar el archivo con ruta *fullname*. Retorna un valor verdadero si el archivo se compila exitosamente, y uno falso en el caso contrario.

Si *ddir* está dado, se antepone a la ruta del archivo que está siendo compilado para su uso en las trazas de tiempo de compilación, y también se compilar en el archivo de código de bytes, donde será utilizado en trazas y otros mensajes en casos donde el archivo fuente no existe en el momento en que el archivo de código de bytes es ejecutado.

Si se proporciona *rx*, a su método `search` se le pasa el nombre de ruta completo al archivo que se está compilando, y si retorna un valor verdadero, el archivo no se compila y se retorna `True`. Esto se puede usar para excluir archivos que coincidan con una expresión regular, dada como un objeto re.Pattern.

Si *quiet* es `False` o `0` (el valor predeterminado), los nombres de archivo y otra información se imprimen en salida estándar. Si se configura en `1`, solo se imprimen los errores. Si se configura en `2`, se suprime toda la salida.

Si *legacy* es verdadero, los archivos de código de byte se escriben en sus locaciones y nombres de herencia, que pueden sobreescribir los archivos de código de byte creado por otra versión de Python. El comportamiento por defecto es escribir archivos en sus locaciones y nombres **PEP 3147** [https://peps.python.org/pep-3147/], lo cual permite que archivos de código de byte de múltiples versiones de Python coexistan.

optimize especifica el nivel de optimización para el compilador. Se pasa a una función incorporada `compile()`. Acepta también una secuencia de niveles de optimización que conducen a múltiples compilaciones de un archivo `.py` en una llamada.

invalidation_mode debería ser un miembro de la enumeración `py_compile.PycInvalidationMode` y controla cómo se invalidan los pycs generados en el momento de ejecución.

Los argumentos *stripdir*, *prependdir* y *limit_sl_dest* corresponden a las opciones `-s`, `-p` y `-e` descritas anteriormente. Pueden especificarse como `str`, o `os.PathLike`.

Si *hardlink_dupes* es verdadero y dos archivos `.pyc` con diferente nivel de optimización tienen el mismo contenido, use enlaces físicos para consolidar los archivos duplicados.

Added in version 3.2.

Distinto en la versión 3.5: El parámetro *quiet* se cambió a un valor multinivel.

Distinto en la versión 3.5: El parámetro *legacy* solo escribe archivos `.pyc`, no archivos `.pyo`, no import cuál sea el valor de *optimize*.

Distinto en la versión 3.7: Se agregó el parámetro *invalidation_mode*.

Distinto en la versión 3.7.2: The *invalidation_mode* parameter's default value is updated to `None`.

Distinto en la versión 3.9: Se agregaron los argumentos *stripdir*, *prependdir*, *limit_sl_dest* y *hardlink_dupes*.

compileall.compile_path(*skip_curdir=True, maxlevels=0, force=False, quiet=0, legacy=False, optimize=-1, invalidation_mode=None*)

Compila en bytes todos los archivos `.py` a lo largo de `sys.path`. Retorna un valor verdadero si todos los archivos se compilan exitosamente, y uno falso en el caso contrario.

Si *skip_curdir* es verdadero (el valor predeterminado), el directorio actual no está incluido en la búsqueda. Todos los otros parámetros se pasan a la función **`compile_dir()`**. Nótese que, al contrario de las otras funciones de compilación, `maxlevels` tomar `0` como valor predeterminado.

Distinto en la versión 3.2: Se agregó el parámetro *legacy* y *optimize*.

Distinto en la versión 3.5: El parámetro *quiet* se cambió a un valor multinivel.

Distinto en la versión 3.5: El parámetro *legacy* solo escribe archivos `.pyc`, no archivos `.pyo`, no import cuál sea el valor de *optimize*.

Distinto en la versión 3.7: Se agregó el parámetro *invalidation_mode*.

Distinto en la versión 3.7.2: The *invalidation_mode* parameter's default value is updated to `None`.

Para forzar la re-compilación del los archivos `.py` en el subdirectorio `Lib/` y todos sus subdirectorios:

```
import compileall

compileall.compile_dir('Lib/', force=True)

# Perform same compilation, excluding files in .svn dire
import re
compileall.compile_dir('Lib/', rx=re.compile(r'[/\\][.]s

# pathlib.Path objects can also be used.
import pathlib
compileall.compile_dir(pathlib.Path('Lib/'), force=True)
```

Ver también

Módulo **`py_compile`**
Compilación byte de un solo archivo fuente.

`dis` — Desensamblador para bytecode de Python

Código fuente: Lib/dis.py [https://github.com/python/cpython/tree/3.14/Lib/dis.py]

The **`dis`** module supports the analysis of CPython bytecode by disassembling it. The CPython bytecode which this module takes as an input is defined in the file `Include/opcode.h` and used by the compiler and the interpreter.

Bytecode es un detalle de implementación del intérprete CPython. No se garantiza que el bytecode no se agregará, eliminará ni cambiará entre las versiones de Python. El uso de este módulo no debe considerarse para trabajar en diferentes máquinas virtuales Python o versiones de Python.

Distinto en la versión 3.6: Use 2 bytes para cada instrucción. Anteriormente, el número de bytes variaba según la instrucción.

Distinto en la versión 3.10: El argumento de las instrucciones de salto, manejo de excepciones y bucle ahora es el desplazamiento de instrucción en lugar del desplazamiento de byte.

Distinto en la versión 3.11: Some instructions are accompanied by one or more inline cache entries, which take the form of **`CACHE`** instructions. These instructions are hidden by default, but can be shown by passing `show_caches=True` to any **`dis`** utility. Furthermore, the interpreter now adapts the bytecode to specialize it for different runtime conditions. The adaptive bytecode can be shown by passing `adaptive=True`.

Distinto en la versión 3.12: El argumento de un salto es el desplazamiento de la instrucción de destino en relación con la instrucción que aparece inmediatamente después de las entradas

`CACHE` de la instrucción de salto.

Como consecuencia, la presencia de las instrucciones `CACHE` es transparente para los saltos hacia adelante, pero debe tenerse en cuenta al razonar sobre los saltos hacia atrás.

Distinto en la versión 3.13: La salida muestra etiquetas lógicas en lugar de desplazamientos de instrucciones para destinos de salto y controladores de excepciones. Se agregaron la opción de línea de comandos `-O` y el argumento `show_offsets`.

Distinto en la versión 3.14: The `-P` command-line option and the `show_positions` argument were added.

The `-S` command-line option is added.

Ejemplo: Dada la función **`myfunc()`**:

```
def myfunc(alist):
    return len(alist)
```

El siguiente comando se puede utilizar para mostrar el desmontaje de **`myfunc()`**:

```
>>> dis.dis(myfunc)
  2           RESUME                   0

  3           LOAD_GLOBAL              1 (len + NULL)
              LOAD_FAST_BORROW         0 (alist)
              CALL                     1
              RETURN_VALUE
```

(El «2» es un número de línea).

Interfaz de línea de comandos

The **`dis`** module can be invoked as a script from the command line:

```
python -m dis [-h] [-C] [-O] [-P] [-S] [infile]
```

Se aceptan las siguientes opciones:

-h, --help
Mostrar uso y salir.

-C, --show-caches
Mostrar cachés en línea.

Added in version 3.13.

-O, --show-offsets
Mostrar desplazamientos de instrucciones.

Added in version 3.13.

-P, --show-positions
Show positions of instructions in the source code.

Added in version 3.14.

-S, --specialized
Show specialized bytecode.

Added in version 3.14.

Si se especifica `infile`, su código desensamblado se escribirá en la salida estándar. De lo contrario, el desensamblado se realiza en el código fuente compilado recibido de la entrada estándar.

Análisis de bytecode

Added in version 3.4.

La API de análisis de bytecode permite que partes del código Python se envuelvan en un objeto `Bytecode` que proporciona un fácil acceso a los detalles del código compilado.

class dis.Bytecode(*x*, ***, *first_line=None*, *current_offset=None*, *show_caches=False*, *adaptive=False*, *show_offsets=False*, *show_positions=False*)

Analiza el bytecode correspondiente a una función, generador, generador asíncrono, corutina, método, cadena de código fuente o un objeto de código (como lo retorna `compile()`).

Este es un contenedor conveniente para muchas de las funciones enumeradas a continuación, en particular `get_instructions()`, ya que iterar sobre una instancia de `Bytecode` produce las operaciones de bytecode como instancias de `Instruction`.

Si *first_line* no es `None`, indica el número de línea que se debe informar para la primera línea de origen en el código desmontado. De lo contrario, la información de la línea de origen (si la hay) se toma directamente del objeto de código desmontado.

Si *current_offset* no es `None`, se refiere a un desplazamiento de instrucción en el código desmontado. Establecer esto significa `dis()` mostrará un marcador de «instrucción actual» contra el código de operación especificado.

Si *show_caches* es `True`, `dis()` mostrará las entradas de caché en línea utilizadas por el intérprete para especializar el código de bytes.

Si *adaptive* es `True`, `dis()` mostrará un código de bytes especializado que puede ser diferente del código de bytes original.

Si *show_offsets* es `True`, `dis()` incluirá desplazamientos de instrucciones en la salida.

If *show_positions* is `True`, `dis()` will include instruction source code positions in the output.

classmethod from_traceback(*tb*, *, *show_caches=False*)

Construye una instancia de `Bytecode` a partir del *traceback* dado, estableciendo *current_offset* en la instrucción responsable de la excepción.

codeobj

El objeto de código compilado.

first_line

La primera línea de origen del objeto de código (si está disponible)

dis()

Retorna una vista formateada de las operaciones de bytecode (lo mismo que impreso por **`dis.dis()`**, pero retornado como una cadena de caracteres multilínea).

info()

Retorna una cadena de caracteres multilínea formateada con información detallada sobre el objeto de código, como **`code_info()`**.

Distinto en la versión 3.7: Esto ahora puede manejar objetos generadores asíncronos y de corutinas.

Distinto en la versión 3.11: Se agregaron los parámetros *show_caches* y *adaptive*.

Distinto en la versión 3.13: Added the *show_offsets* parameter

Distinto en la versión 3.14: Added the *show_positions* parameter.

Ejemplo:

```
>>> bytecode = dis.Bytecode(myfunc)
>>> for instr in bytecode:
...     print(instr.opname)
...
RESUME
LOAD_GLOBAL
LOAD_FAST_BORROW
```

```
CALL
RETURN_VALUE
```

Funciones de análisis

The **`dis`** module also defines the following analysis functions that convert the input directly to the desired output. They can be useful if only a single operation is being performed, so the intermediate analysis object isn't useful:

dis.code_info(*x*)

> Retorna una cadena de caracteres multilínea formateada con información detallada del objeto de código para la función, generador, generador asíncrono, corutina, método, cadena de código fuente u objeto de código suministrados.
>
> Tenga en cuenta que el contenido exacto de las cadenas de información de código depende en gran medida de la implementación y puede cambiar arbitrariamente en las diferentes máquinas virtuales Python o las versiones de Python.
>
> *Added in version 3.2.*
>
> *Distinto en la versión 3.7:* Esto ahora puede manejar objetos generadores asíncronos y de corutinas.

dis.show_code(*x*, ***, *file=None*)

> Imprime información detallada del objeto de código para la función, método, cadena de código fuente u objeto de código suministrado en *file* (o `sys.stdout` si *file* no está especificado).
>
> Esta es una abreviatura conveniente para `print(code_info(x), file=file)`, destinado a la exploración interactiva en el indicador del intérprete (*prompt*).
>
> *Added in version 3.2.*

Distinto en la versión 3.4: Agrega un parámetro *file*.

dis.dis(*x=None*, ***, *file=None*, *depth=None*, *show_caches=False*, *adaptive=False*, *show_offsets=False*, *show_positions=False*)

Desensamblar el objeto *x*. *x* puede denotar un módulo, una clase, un método, una función, un generador, un generador asincrónico, una corrutina, un objeto de código, una cadena de código fuente o una secuencia de bytes de código de bytes sin formato. Para un módulo, desensamblar todas las funciones. Para una clase, desensamblar todos los métodos (incluidos los métodos de clase y estáticos). Para un objeto de código o una secuencia de código de bytes sin formato, imprime una línea por cada instrucción de código de bytes. También desensamblar de forma recursiva objetos de código anidados. Estos pueden incluir expresiones de generador, funciones anidadas, los cuerpos de clases anidadas y los objetos de código utilizados para annotation scopes. Las cadenas se compilan primero en objetos de código con la función incorporada **`compile()`** antes de desensamblarse. Si no se proporciona ningún objeto, esta función desensamblará el último seguimiento.

El desensamblaje se escribe como texto en el argumento *file* proporcionado si se proporciona y, de lo contrario, `sys.stdout`.

La profundidad máxima de recursión está limitada por *depth* a menos que sea `None`. `depth=0` significa que no hay recursión.

Si *show_caches* es `True`, esta función mostrará las entradas de caché en línea utilizadas por el intérprete para especializar el código de bytes.

Si *adaptive* es `True`, esta función mostrará un código de bytes especializado que puede ser diferente del código de bytes original.

Distinto en la versión 3.4: Agrega un parámetro *file*.

Distinto en la versión 3.7: Desensamblaje recursivo implementado y parámetro agregado *depth*.

Distinto en la versión 3.7: Esto ahora puede manejar objetos generadores asíncronos y de corutinas.

Distinto en la versión 3.11: Se agregaron los parámetros *show_caches* y *adaptive*.

Distinto en la versión 3.13: Se agregó el parámetro *show_offsets*.

Distinto en la versión 3.14: Added the *show_positions* parameter.

dis.distb(*tb=None, *, file=None, show_caches=False, adaptive=False, show_offset=False, show_positions=False*)

Desmonta la función de inicio de pila de un rastreo, utilizando el último rastreo si no se pasó ninguno. Se indica la instrucción que causa la excepción.

El desensamblaje se escribe como texto en el argumento *file* proporcionado si se proporciona y, de lo contrario, `sys.stdout`.

Distinto en la versión 3.4: Agrega un parámetro *file*.

Distinto en la versión 3.11: Se agregaron los parámetros *show_caches* y *adaptive*.

Distinto en la versión 3.13: Se agregó el parámetro *show_offsets*.

Distinto en la versión 3.14: Added the *show_positions* parameter.

dis.disassemble(*code, lasti=-1, *, file=None, show_caches=False, adaptive=False, show_offsets=False, show_positions=False*)
dis.disco(*code, lasti=-1, *, file=None, show_caches=False, adaptive=False, show_offsets=False, show_positions=False*)

Desmonta un objeto de código, que indica la última instrucción si se proporcionó *lasti*. La salida se divide en las siguientes columnas:

1. the source code location of the instruction. Complete location information is shown if *show_positions* is true. Otherwise (the default) only the line number is displayed.
2. la instrucción actual, indicada como `-->`,
3. una instrucción etiquetada, indicada con `>>`,
4. la dirección de la instrucción,
5. el nombre del código de operación,
6. parámetros de operación, y
7. interpretación de los parámetros entre paréntesis.

La interpretación de parámetros reconoce nombres de variables locales y globales, valores constantes, objetivos de ramificación y operadores de comparación.

El desensamblaje se escribe como texto en el argumento *file* proporcionado si se proporciona y, de lo contrario, `sys.stdout`.

Distinto en la versión 3.4: Agrega un parámetro *file*.

Distinto en la versión 3.11: Se agregaron los parámetros *show_caches* y *adaptive*.

Distinto en la versión 3.13: Se agregó el parámetro *show_offsets*.

Distinto en la versión 3.14: Added the *show_positions* parameter.

dis.get_instructions(*x*, ***, *first_line=None*, *show_caches=False*, *adaptive=False*)

Retorna un iterador sobre las instrucciones en la función, método, cadena de código fuente u objeto de código suministrado.

El iterador genera una serie de tuplas con nombre **`Instruction`** que dan los detalles de cada operación en el código suministrado.

Si *first_line* no es `None`, indica el número de línea que se debe informar para la primera línea de origen en el código desmontado. De lo contrario, la información de la línea de origen (si la hay) se toma directamente del objeto de código desmontado.

El parámetro *adaptive* funciona como lo hace en `dis()`.

Added in version 3.4.

Distinto en la versión 3.11: Se agregaron los parámetros *show_caches* y *adaptive*.

Distinto en la versión 3.13: El parámetro *show_caches* está obsoleto y no tiene efecto. El iterador genera las instancias `Instruction` con el campo *cache_info* completado (independientemente del valor de *show_caches*) y ya no genera elementos separados para las entradas de caché.

dis.findlinestarts(*code*)

Esta función generadora utiliza el método `co_lines()` de code object y *code* para encontrar los desplazamientos que son inicios de líneas en el código fuente. Se generan como pares `(offset, lineno)`.

Distinto en la versión 3.6: Los números de línea pueden estar disminuyendo. Antes, siempre estaban aumentando.

Distinto en la versión 3.10: Se utiliza el método **PEP 626** [https://peps.python.org/pep-0626/] `co_lines()` en lugar de los atributos `co_firstlineno` y `co_lnotab` del code object.

Distinto en la versión 3.13: Los números de línea pueden ser `None` para el código de bytes que no se asigna a las líneas de origen.

dis.findlabels(*code*)

Detecta todos los desplazamientos en la cadena de caracteres de código de byte compilada *code* que son objetivos de salto y retorna una lista de estos desplazamientos.

dis.stack_effect(*opcode*, *oparg=None*, *, *jump=None*)

Calcula el efecto de pila de *opcode* con el argumento *oparg*.

Si el código tiene un objetivo de salto y *jump* es `True`, **`stack_effect()`** retornará el efecto de pila del salto. Si *jump* es `False`, retornará el efecto de acumulación de no saltar. Y si *jump* es `None` (predeterminado), retornará el efecto de acumulación máxima de ambos casos.

Added in version 3.4.

Distinto en la versión 3.8: Agrega un parámetro *jump*.

Distinto en la versión 3.13: Si se omite `oparg` (o `None`), ahora se devuelve el efecto de pila para `oparg=0`. Anteriormente, esto era un error para los códigos de operación que usan su argumento. Tampoco es un error pasar un entero `oparg` cuando `opcode` no lo usa; en este caso, se ignora `oparg`.

Instrucciones bytecode de Python

La función **`get_instructions()`** y clase **`Bytecode`** proporcionan detalles de las instrucciones bytecode como instancias **`Instruction`**:

class dis.Instruction

Detalles para una operación de bytecode

opcode

código numérico para la operación, correspondiente a los valores del opcode listados a continuación y los valores de bytecode en Colecciones opcode.

opname

nombre legible por humanos para la operación

baseopcode

código numérico para la operación base si la operación es especializada; en caso contrario, igual a **`opcode`** baseopname

Nombre legible para humanos para la operación base si la operación es especializada; de lo contrario, es igual a **`opname`**

arg

argumento numérico para la operación (si existe), de lo contrario `None`

oparg

alias para **`arg`**

argval

valor de argumento resuelto (si lo hay); de lo contrario, `None`

argrepr

descripción legible por humanos del argumento de la operación (si lo hay); de lo contrario, una cadena vacía.

offset

índice de inicio de operación dentro de la secuencia de bytecode

start_offset

índice de inicio de la operación dentro de la secuencia de bytecode, incluidas las operaciones con prefijo `EXTENDED_ARG` si están presentes; de lo contrario, igual a **`offset`**

cache_offset

índice de inicio de las entradas de caché después de la operación

end_offset

índice final de las entradas de caché después de la operación starts_line

`True` si este código de operación inicia una línea de origen, de lo contrario `False`

line_number

número de línea de origen asociado con este código de operación (si lo hay), de lo contrario `None`

is_jump_target

`True` si otro código salta aquí, de lo contrario, `False`

jump_target

índice de código de bytes del objetivo del salto si se trata de una operación de salto, de lo contrario `None`

positions

Objeto **`dis.Positions`** que contiene las ubicaciones de inicio y finalización cubiertas por esta instrucción.

cache_info

Information about the cache entries of this instruction, as triplets of the form `(name, size, data)`, where the `name` and `size` describe the cache format and data is the contents of the cache. `cache_info` is `None` if the instruction does not have caches.

Added in version 3.4.

Distinto en la versión 3.11: Se agrega el campo `positions`.

Distinto en la versión 3.13: Se modificó el campo `starts_line`.

Se agregaron los campos `start_offset`, `cache_offset`, `end_offset`, `baseopname`, `baseopcode`, `jump_target`, `oparg`, `line_number` y `cache_info`.

class dis.Positions

En caso de que la información no esté disponible, algunos campos pueden ser `None`.

lineno

end_lineno

col_offset

end_col_offset

Added in version 3.11.

El compilador de Python actualmente genera las siguientes instrucciones de bytecode.

Instrucciones generales

A continuación, nos referiremos a la pila del intérprete como `STACK` y describiremos las operaciones en ella como si fuera una lista de Python. La parte superior de la pila corresponde a `STACK[-1]` en este lenguaje.

NOP

Código que no hace nada. Utilizado como marcador de posición por el optimizador de bytecode y para generar eventos de seguimiento de línea.

NOT_TAKEN

Do nothing code. Used by the interpreter to record **`BRANCH_LEFT`** and **`BRANCH_RIGHT`** events for **`sys.monitoring`**.

Added in version 3.14.

POP_ITER

Removes the iterator from the top of the stack.

Added in version 3.14.

POP_TOP

Elimina el elemento de la parte superior de la pila:

```
STACK.pop()
```

END_FOR

Elimina el elemento superior de la pila. Equivale a `POP_TOP`. Se utiliza para limpiar al final de los bucles, de ahí el nombre.

Added in version 3.12.

END_SEND

Implementa `del STACK[-2]`. Se utiliza para limpiar cuando finaliza un generador.

Added in version 3.12.

COPY(*i*)

Empuja el elemento i-ésimo hacia la parte superior de la pila sin quitarlo de su ubicación original:

```
assert i > 0
STACK.append(STACK[-i])
```

Added in version 3.11.

SWAP(*i*)

Intercambia la parte superior de la pila con el elemento i-ésimo:

```
STACK[-i], STACK[-1] = STACK[-1], STACK[-i]
```

Added in version 3.11.

CACHE

En lugar de ser una instrucción real, este código de operación se usa para marcar espacio adicional para que el intérprete almacene en caché datos útiles directamente en el bytecode. Todas las utilidades `dis` lo ocultan automáticamente, pero se puede ver con `show_caches=True`.

Lógicamente, este espacio forma parte de la instrucción anterior. Muchos códigos de operación esperan ser seguidos por un número exacto de cachés y le indicarán al intérprete que los omita en tiempo de ejecución.

Los cachés poblados pueden parecer instrucciones arbitrarias, por lo que se debe tener mucho cuidado al leer o modificar el bytecode adaptativo sin procesar que contiene datos acelerados.

Added in version 3.11.

Operaciones unarias

Las operaciones unarias toman la parte superior de la pila, aplican la operación y retornan el resultado a la pila.

UNARY_NEGATIVE

Implementa `STACK[-1] = -STACK[-1]`.

UNARY_NOT

Implementa `STACK[-1] = not STACK[-1]`.

Distinto en la versión 3.13: Esta instrucción ahora requiere un operando `bool` exacto.

UNARY_INVERT

Implementa `STACK[-1] = ~STACK[-1]`.

GET_ITER

Implementa `STACK[-1] = iter(STACK[-1])`.

GET_YIELD_FROM_ITER

Si `STACK[-1]` es un objeto generator iterator o coroutine, se deja como está. De lo contrario, se implementa `STACK[-1] = iter(STACK[-1])`.

Added in version 3.5.

TO_BOOL

Implementa `STACK[-1] = bool(STACK[-1])`.

Added in version 3.13.

Operaciones binarias e in situ

Las operaciones binarias eliminan los dos elementos superiores de la pila (`STACK[-1]` y `STACK[-2]`). Realizan la operación y luego vuelven a colocar el resultado en la pila.

Las operaciones en el lugar son como operaciones binarias, pero la operación se realiza en el lugar cuando `STACK[-2]` la admite, y el `STACK[-1]` resultante puede ser (pero no tiene que ser) el `STACK[-2]` original.

BINARY_OP(*op*)

Implementa los operadores binarios y locales (según el valor de *op*):

```
rhs = STACK.pop()
lhs = STACK.pop()
STACK.append(lhs op rhs)
```

Added in version 3.11.

Distinto en la versión 3.14: With oparg :`NB_SUBSCR`, implements binary subscript (replaces opcode `BINARY_SUBSCR`)

STORE_SUBSCR

Implementos:

```
key = STACK.pop()
container = STACK.pop()
value = STACK.pop()
container[key] = value
```

DELETE_SUBSCR

Implementos:

```
key = STACK.pop()
container = STACK.pop()
del container[key]
```

BINARY_SLICE

Implementos:

```
end = STACK.pop()
start = STACK.pop()
container = STACK.pop()
STACK.append(container[start:end])
```

Added in version 3.12.

STORE_SLICE

Implementos:

```
end = STACK.pop()
start = STACK.pop()
container = STACK.pop()
value = STACK.pop()
container[start:end] = value
```

Added in version 3.12.

Opcodes de corutinas

GET_AWAITABLE(*where*)

Implementa `STACK[-1] = get_awaitable(STACK[-1])`, donde `get_awaitable(o)` devuelve `o` si `o` es un objeto de corrutina o un objeto generador con el indicador **`CO_ITERABLE_COROUTINE`**, o resuelve `o.__await__`.

> Si el operando `where` es distinto de cero, indica dónde ocurre la instrucción:

- `1`: Después de una llamada a `__aenter__`
- `2`: Después de una llamada a `__aexit__`

Added in version 3.5.

Distinto en la versión 3.11: Anteriormente, esta instrucción no tenía un oparg.

GET_AITER

Implementa `STACK[-1] = STACK[-1].__aiter__()`.

Added in version 3.5.

Distinto en la versión 3.7: Ya no se admite el retorno de objetos *awaitable* de `__aiter__`.

GET_ANEXT

Implemente `STACK.append(get_awaitable(STACK[-1].__anext__()))` en la pila. Consulte `GET_AWAITABLE` para obtener más detalles sobre `get_awaitable`.

Added in version 3.5.

END_ASYNC_FOR

Finaliza un bucle **`async for`**. Controla una excepción generada al esperar un elemento siguiente. La pila contiene el iterable asincrónico en `STACK[-2]` y la excepción generada en `STACK[-1]`. Ambos se eliminan. Si la excepción no es **`StopAsyncIteration`**, se vuelve a generar.

Added in version 3.8.

Distinto en la versión 3.11: La representación de excepciones en la pila ahora consta de uno, no de tres elementos.

CLEANUP_THROW

Maneja una excepción generada durante una llamada

`throw()` o **`close()`** a través del marco actual. Si `STACK[-1]` es una instancia de **`StopIteration`**, extrae tres valores de la pila y envía su miembro `value`. De lo contrario, vuelve a generar `STACK[-1]`.

Added in version 3.12.

Opcodes misceláneos

SET_ADD(*i*)

Implementos:

```
item = STACK.pop()
set.add(STACK[-i], item)
```

Se utiliza para implementar comprensiones de conjuntos.

LIST_APPEND(*i*)

Implementos:

```
item = STACK.pop()
list.append(STACK[-i], item)
```

Se utiliza para implementar listas por comprensión.

MAP_ADD(*i*)

Implementos:

```
value = STACK.pop()
key = STACK.pop()
dict.__setitem__(STACK[-i], key, value)
```

Se utiliza para implementar comprensiones de diccionarios.

Added in version 3.1.

Distinto en la versión 3.8: El valor del mapa es `STACK[-1]` y la clave del mapa es `STACK[-2]`. Antes, estaban invertidos.

Para todas las instrucciones **`SET_ADD`**, **`LIST_APPEND`** y

`MAP_ADD`, mientras el valor agregado o el par clave/valor aparece, el objeto contenedor permanece en la pila para que quede disponible para futuras iteraciones del bucle.

RETURN_VALUE

Regresa con `STACK[-1]` al llamador de la función.

YIELD_VALUE

Produce `STACK.pop()` a partir de un generator.

Distinto en la versión 3.11: oparg establecido para ser la profundidad de la pila.

Distinto en la versión 3.12: oparg está configurado para ser la profundidad del bloque de excepción, para el cierre eficiente de los generadores.

Distinto en la versión 3.13: oparg es `1` si esta instrucción es parte de un yield-from o await, y `0` en caso contrario.

SETUP_ANNOTATIONS

Comprueba si `__anotaciones__` está definido en `locals()`, si no está configurado como un `dict` vacío. Este código de operación solo se emite si el cuerpo de una clase o módulo contiene anotaciones de variables estáticamente.

Added in version 3.6.

POP_EXCEPT

Extrae un valor de la pila, que se utiliza para restaurar el estado de excepción.

Distinto en la versión 3.11: La representación de excepciones en la pila ahora consta de uno, no de tres elementos.

RERAISE

Vuelve a generar la excepción que se encuentra actualmente en la parte superior de la pila. Si oparg no es cero, extrae un valor adicional de la pila que se utiliza para establecer `f_lasti` del marco actual.

Added in version 3.9.

Distinto en la versión 3.11: La representación de excepciones en la pila ahora consta de uno, no de tres elementos.

PUSH_EXC_INFO

Extrae un valor de la pila. Agrega la excepción actual a la parte superior de la pila. Agrega el valor que apareció originalmente de vuelta a la pila. Se utiliza en los controladores de excepciones.

Added in version 3.11.

CHECK_EXC_MATCH

Realiza una comparación de excepciones para `except`. Comprueba si `STACK[-2]` es una comparación de excepciones con `STACK[-1]`. Extrae `STACK[-1]` y envía el resultado booleano de la prueba.

Added in version 3.11.

CHECK_EG_MATCH

Realiza la coincidencia de excepciones para `except*`. Aplica `split(STACK[-1])` en el grupo de excepciones que representa a `STACK[-2]`.

En caso de coincidencia, extrae dos elementos de la pila y agrega el subgrupo que no coincide (`None` en caso de coincidencia total) seguido del subgrupo coincidente. Cuando no hay ninguna coincidencia, muestra un elemento (el tipo de coincidencia) y presiona `None`.

Added in version 3.11.

WITH_EXCEPT_START

Llama a la función en la posición 4 de la pila con argumentos (tipo, val, tb) que representan la excepción en la parte superior de la pila. Se utiliza para implementar la llamada `context_manager.__exit__(*exc_info())` cuando se ha producido una excepción en una sentencia **`with`**.

Added in version 3.9.

Distinto en la versión 3.11: La función `__exit__` está en la posición 4 de la pila en lugar de la 7. La representación de excepciones en la pila ahora consta de uno, no de tres, elementos.

LOAD_COMMON_CONSTANT

Pushes a common constant onto the stack. The interpreter contains a hardcoded list of constants supported by this instruction. Used by the **`assert`** statement to load **`AssertionError`**.

Added in version 3.14.

LOAD_BUILD_CLASS

Coloca **`builtins.__build_class__()`** en la pila. Luego se lo llama para construir una clase.

GET_LEN

Perform `STACK.append(len(STACK[-1]))`. Used in **`match`** statements where comparison with structure of pattern is needed.

Added in version 3.10.

MATCH_MAPPING

Si `STACK[-1]` es una instancia de **`collections.abc.Mapping`** (o, más técnicamente: si tiene el indicador **`Py_TPFLAGS_MAPPING`** establecido en su **`tp_flags`**), inserte `True` en la pila. De lo contrario, inserte `False`.

Added in version 3.10.

MATCH_SEQUENCE

Si `STACK[-1]` es una instancia de **`collections.abc.Sequence`** y *not* es una instancia de **`str`**/**`bytes`**/**`bytearray`** (o, más técnicamente: si tiene el indicador **`Py_TPFLAGS_SEQUENCE`** establecido en su **`tp_flags`**), inserte `True` en la pila. De lo contrario, inserte `False`.

Added in version 3.10.

MATCH_KEYS

`STACK[-1]` es una tupla de claves de mapeo y `STACK[-2]` es el sujeto de la coincidencia. Si `STACK[-2]` contiene todas las claves de `STACK[-1]`, inserte un **`tuple`** que contenga los valores correspondientes. De lo contrario, inserte `None`.

Added in version 3.10.

Distinto en la versión 3.11: Anteriormente, esta instrucción también generaba un valor booleano que indicaba éxito (`True`) o falla (`False`).

STORE_NAME(*namei*)

Implementa `name = STACK.pop()`. *namei* es el índice de *name* en el atributo **`co_names`** de code object. El compilador intenta utilizar **`STORE_FAST`** o **`STORE_GLOBAL`** si es posible.

DELETE_NAME(*namei*)

Implementa `del name`, donde *namei* es el índice en el atributo **`co_names`** del code object.

UNPACK_SEQUENCE(*count*)

Descomprime `STACK[-1]` en valores individuales *count*, que se colocan en la pila de derecha a izquierda. Requiere que haya exactamente valores *count*.:

```
assert(len(STACK[-1]) == count)
```

```
STACK.extend(STACK.pop()[:-count-1:-1])
```

UNPACK_EX(*counts*)

Implementa la asignación con un objetivo destacado: descomprime un iterable en `STACK[-1]` en valores individuales, donde la cantidad total de valores puede ser menor que la cantidad de elementos en el iterable: uno de los nuevos valores será una lista de todos los elementos restantes.

El número de valores antes y después del valor de la lista está limitado a 255.

La cantidad de valores antes del valor de la lista se codifica en el argumento del código de operación. La cantidad de valores después de la lista, si los hay, se codifica utilizando un `EXTENDED_ARG`. Como consecuencia, el argumento puede verse como valores de dos bytes, donde el byte bajo de *counts* es la cantidad de valores antes del valor de la lista y el byte alto de *counts*, la cantidad de valores después de este.

Los valores extraídos se colocan en la pila de derecha a izquierda, es decir, `a, *b, c = d` se almacenará después de la ejecución como `STACK.extend((a, b, c))`.

STORE_ATTR(*namei*)

Implementos:

```
obj = STACK.pop()
value = STACK.pop()
obj.name = value
```

donde *namei* es el índice del nombre en **`co_names`** del code object.

DELETE_ATTR(*namei*)

Implementos:

```
obj = STACK.pop()
del obj.name
```

donde *namei* es el índice del nombre en **`co_names`** del code object.

STORE_GLOBAL(*namei*)

Funciona como **`STORE_NAME`**, pero almacena el nombre como global.

DELETE_GLOBAL(*namei*)

Funciona como **`DELETE_NAME`**, pero elimina un nombre global.

LOAD_CONST(*consti*)

Apila `co_consts[consti]` en la pila.

LOAD_SMALL_INT(*i*)

Pushes the integer `i` onto the stack. `i` must be in `range(256)`

Added in version 3.14.

LOAD_NAME(*namei*)

Introduce el valor asociado con `co_names[namei]` en la pila. El nombre se busca en las variables locales, luego en las variables globales y luego en las integradas.

LOAD_LOCALS

Introduce una referencia al diccionario de variables locales en la pila. Esto se utiliza para preparar diccionarios de espacios de nombres para **`LOAD_FROM_DICT_OR_DEREF`** y **`LOAD_FROM_DICT_OR_GLOBALS`**.

Added in version 3.12.

LOAD_FROM_DICT_OR_GLOBALS(*i*)

Extrae una asignación de la pila y busca el valor de `co_names[namei]`. Si no se encuentra el nombre allí, lo busca en las variables globales y luego en las integradas, de forma similar a `LOAD_GLOBAL`. Esto se utiliza para cargar variables globales en annotation scopes dentro de los cuerpos de clase.

Added in version 3.12.

BUILD_TEMPLATE

Constructs a new `Template` instance from a tuple of strings and a tuple of interpolations and pushes the resulting object onto the stack:

```
interpolations = STACK.pop()
strings = STACK.pop()
STACK.append(_build_template(strings, interpolation
```

Added in version 3.14.

BUILD_INTERPOLATION(*format*)

Constructs a new `Interpolation` instance from a value and its source expression and pushes the resulting object onto the stack.

If no conversion or format specification is present, `format` is set to `2`.

If the low bit of `format` is set, it indicates that the interpolation contains a format specification.

If `format >> 2` is non-zero, it indicates that the interpolation contains a conversion. The value of `format >> 2` is the conversion type (`0` for no conversion, `1` for `!s`, `2` for `!r`, and `3` for `!a`):

```
conversion = format >> 2
if format & 1:
    format_spec = STACK.pop()
else:
    format_spec = None
expression = STACK.pop()
value = STACK.pop()
```

```
STACK.append(_build_interpolation(value, expression
```

Added in version 3.14.

BUILD_TUPLE(*count*)

Crea una tupla que consume elementos *count* de la pila y envía la tupla resultante a la pila:

```
if count == 0:
    value = ()
else:
    value = tuple(STACK[-count:])
    STACK = STACK[:-count]

STACK.append(value)
```

BUILD_LIST(*count*)

Funciona como **`BUILD_TUPLE`**, pero crea una lista.

BUILD_SET(*count*)

Funciona como **`BUILD_TUPLE`**, pero crea un conjunto.

BUILD_MAP(*count*)

Inserta un nuevo objeto de diccionario en la pila. Extrae elementos `2 * count` para que el diccionario contenga entradas *count*: `{..., STACK[-4]: STACK[-3], STACK[-2]: STACK[-1]}`.

Distinto en la versión 3.5: El diccionario se crea a partir de elementos de la pila en lugar de crear un diccionario vacío dimensionado previamente para contener *count* elementos.

BUILD_STRING(*count*)

Concatena *count* cadenas de caracteres de la pila y empuja la cadena de caracteres resultante en la pila.

Added in version 3.6.

LIST_EXTEND(*i*)

Implementos:

```
seq = STACK.pop()
list.extend(STACK[-i], seq)
```

Se utiliza para crear listas.

Added in version 3.9.

SET_UPDATE(*i*)

Implementos:

```
seq = STACK.pop()
set.update(STACK[-i], seq)
```

Se utiliza para construir decorados.

Added in version 3.9.

DICT_UPDATE(*i*)

Implementos:

```
map = STACK.pop()
dict.update(STACK[-i], map)
```

Se utiliza para crear diccionarios.

Added in version 3.9.

DICT_MERGE(*i*)

Como **`DICT_UPDATE`** pero lanza una excepción para claves duplicadas.

Added in version 3.9.

LOAD_ATTR(*namei*)

Si el bit bajo de `namei` no está configurado, esto reemplaza `STACK[-1]` con `getattr(STACK[-1],` `co_names[namei>>1]`).

Si se establece el bit bajo de `namei`, se intentará cargar un método llamado `co_names[namei>>1]` desde el objeto `STACK[-1]`. Se extrae `STACK[-1]`. Este código de bytes distingue dos casos: si `STACK[-1]` tiene un método con el nombre correcto, el código de bytes inserta el método no vinculado y `STACK[-1]`. **`CALL`** o **`CALL_KW`** utilizarán `STACK[-1]` como primer argumento (`self`) al llamar al método no vinculado. De lo contrario, se insertan `NULL` y el objeto devuelto por la búsqueda de atributos.

Distinto en la versión 3.12: Si se establece el bit bajo de `namei`, entonces se inserta un `NULL` o `self` en la pila antes del atributo o el método no vinculado respectivamente.

LOAD_SUPER_ATTR(*namei*)

Este código de operación implementa **`super()`**, tanto en su forma de cero argumentos como de dos argumentos (por ejemplo, `super().method()`, `super().attr` y `super(cls, self).method()`, `super(cls, self).attr`).

Extrae tres valores de la pila (desde la parte superior de la pila hacia abajo):

- `self`: el primer argumento del método actual
- `cls`: la clase dentro de la cual se definió el método actual
- El `super` global

Con respecto a su argumento, funciona de manera similar a **`LOAD_ATTR`**, excepto que `namei` se desplaza hacia la izquierda 2 bits en lugar de 1.

El bit bajo de `namei` indica que se debe intentar cargar un método, como en el caso de **`LOAD_ATTR`**, lo que da como resultado la inserción de `NULL` y el método cargado. Cuando no está configurado, se inserta un solo valor en la pila.

El segundo bit bajo de `namei`, si está configurado, significa que esta fue una llamada de dos argumentos a **`super()`** (si no está configurado significa cero argumentos).

Added in version 3.12.

COMPARE_OP(*opname*)

Realiza una operación booleana. El nombre de la operación se puede encontrar en `cmp_op[opname >> 5]`. Si se establece el quinto bit más bajo de `opname` (`opname & 16`), el resultado se debe convertir a `bool`.

Distinto en la versión 3.13: El quinto bit más bajo del oparg ahora indica una conversión forzada a **`bool`**.

IS_OP(*invert*)

Realiza una comparación `is` o `is not` si `invert` es 1.

Added in version 3.9.

CONTAINS_OP(*invert*)

Realiza una comparación `in` o `not in` si `invert` es 1.

Added in version 3.9.

IMPORT_NAME(*namei*)

Importa el módulo `co_names[namei]`. Se extraen `STACK[-1]` y `STACK[-2]` y proporcionan los argumentos *fromlist* y *level* de **`__import__()`**. El objeto del módulo se coloca en la pila. El espacio de nombres actual no se ve afectado: para una declaración de importación adecuada, una instrucción **`STORE_FAST`** posterior modifica el espacio de nombres.

IMPORT_FROM(*namei*)

Carga el atributo `co_names[namei]` del módulo que se encuentra en `STACK[-1]`. El objeto resultante se coloca en la pila para ser almacenado posteriormente mediante una instrucción **`STORE_FAST`**.

JUMP_FORWARD(*delta*)

Incrementa el contador de bytecode en *delta*.

JUMP_BACKWARD(*delta*)

Decrementa el contador de bytecode en *delta*. Comprueba si hay interrupciones.

Added in version 3.11.

JUMP_BACKWARD_NO_INTERRUPT(*delta*)

Decrementa el contador de bytecode en *delta*. No busca interrupciones.

Added in version 3.11.

POP_JUMP_IF_TRUE(*delta*)

Si `STACK[-1]` es verdadero, incrementa el contador de bytecode en *delta*. Se elimina `STACK[-1]`.

Distinto en la versión 3.11: El oparg ahora es un delta relativo en lugar de un objetivo absoluto. Este código de operación es una pseudoinstrucción, reemplazada en el bytecode final por las versiones dirigidas (hacia adelante/hacia atrás).

Distinto en la versión 3.12: Esto ya no es una pseudoinstrucción.

Distinto en la versión 3.13: Esta instrucción ahora requiere un operando **`bool`** exacto.

POP_JUMP_IF_FALSE(*delta*)

Si `STACK[-1]` es falso, incrementa el contador de bytecode en *delta*. Se elimina `STACK[-1]`.

Distinto en la versión 3.11: El oparg ahora es un delta relativo en lugar de un objetivo absoluto. Este código de operación es una pseudoinstrucción, reemplazada en el bytecode final por las versiones dirigidas (hacia adelante/hacia atrás).

Distinto en la versión 3.12: Esto ya no es una pseudoinstrucción.

Distinto en la versión 3.13: Esta instrucción ahora requiere un operando **`bool`** exacto.

POP_JUMP_IF_NOT_NONE(*delta*)

Si `STACK[-1]` no es `None`, incrementa el contador de bytecode en *delta*. Se elimina `STACK[-1]`.

Added in version 3.11.

Distinto en la versión 3.12: Esto ya no es una pseudoinstrucción.

POP_JUMP_IF_NONE(*delta*)

Si `STACK[-1]` es `None`, incrementa el contador de bytecode en *delta*. Se elimina `STACK[-1]`.

Added in version 3.11.

Distinto en la versión 3.12: Esto ya no es una pseudoinstrucción.

FOR_ITER(*delta*)

`STACK[-1]` es un iterator. Llama a su método **`__next__()`**. Si esto produce un nuevo valor, colócalo en la pila (dejando el iterador debajo de él). Si el iterador indica que está agotado, el contador de código de bytes se incrementa en *delta*.

Distinto en la versión 3.12: Hasta la versión 3.11, el iterador se activaba cuando se agotaba.

LOAD_GLOBAL(*namei*)

Carga el `co_names[namei>>1]` global llamado en la pila.

Distinto en la versión 3.11: Si se establece el bit bajo de `namei`, se agrega un `NULL` a la pila antes de la variable global.

LOAD_FAST(*var_num*)

Apila una referencia al local `co_varnames[var_num]` sobre la pila.

Distinto en la versión 3.12: Este código de operación ahora solo se utiliza en situaciones en las que se garantiza que la variable local se inicializará. No puede generar **`UnboundLocalError`**.

LOAD_FAST_BORROW(*var_num*)

Pushes a borrowed reference to the local `co_varnames[var_num]` onto the stack.

Added in version 3.14.

LOAD_FAST_LOAD_FAST(*var_nums*)

Pushes references to `co_varnames[var_nums >> 4]` and `co_varnames[var_nums & 15]` onto the stack.

Added in version 3.13.

LOAD_FAST_BORROW_LOAD_FAST_BORROW(*var_nums*)

Pushes borrowed references to `co_varnames[var_nums >> 4]` and `co_varnames[var_nums & 15]` onto the stack.

Added in version 3.14.

LOAD_FAST_CHECK(*var_num*)

Inserta una referencia al `co_varnames[var_num]` local en la pila y genera un **`UnboundLocalError`** si la variable local no se ha inicializado.

Added in version 3.12.

LOAD_FAST_AND_CLEAR(*var_num*)

Inserta una referencia al `co_varnames[var_num]` local en la pila (o inserta `NULL` en la pila si la variable local no se ha inicializado) y establece `co_varnames[var_num]` en `NULL`.

Added in version 3.12.

STORE_FAST(*var_num*)

Almacena `STACK.pop()` en el `co_varnames[var_num]` local.

STORE_FAST_STORE_FAST(*var_nums*)

Stores `STACK[-1]` into `co_varnames[var_nums >> 4]` and `STACK[-2]` into `co_varnames[var_nums & 15]`.

Added in version 3.13.

STORE_FAST_LOAD_FAST(*var_nums*)

Stores `STACK.pop()` into the local `co_varnames[var_nums >> 4]` and pushes a reference to the local `co_varnames[var_nums & 15]` onto the stack.

Added in version 3.13.

DELETE_FAST(*var_num*)

Elimina la `co_varnames[var_num]` local.

MAKE_CELL(*i*)

Crea una nueva celda en la ranura `i`. Si esa ranura no está vacía, ese valor se almacena en la nueva celda.

Added in version 3.11.

LOAD_DEREF(*i*)

Carga la celda contenida en el slot `i` del almacenamiento «fast locals». Agrega una referencia al objeto que contiene la celda en la pila.

Distinto en la versión 3.11: `i` ya no está compensado por la longitud de **`co_varnames`**.

LOAD_FROM_DICT_OR_DEREF(*i*)

Pops a mapping off the stack and looks up the name associated with slot `i` of the «fast locals» storage in this mapping. If the name is not found there, loads it from the cell contained in slot `i`, similar to **`LOAD_DEREF`**. This is used for loading closure variables in class bodies (which previously used **`LOAD_CLASSDEREF`**) and in annotation scopes within class bodies.

Added in version 3.12.

STORE_DEREF(*i*)

Almacena `STACK.pop()` en la celda contenida en la ranura `i` del almacenamiento de «locales rápidos».

Distinto en la versión 3.11: `i` ya no está compensado por la longitud de **`co_varnames`**.

DELETE_DEREF(*i*)

Vacía la celda contenida en la ranura `i` del almacenamiento de «locales rápidos». Utilizado por la instrucción **`del`**.

Added in version 3.2.

Distinto en la versión 3.11: `i` ya no está compensado por la longitud de **`co_varnames`**.

COPY_FREE_VARS(*n*)

Copies the `n` free (closure) variables from the closure into the frame. Removes the need for special code on the caller's side when calling closures.

Added in version 3.11.

RAISE_VARARGS(*argc*)

Provoca una excepción utilizando una de las 3 formas de la declaración `raise`, dependiendo del valor de *argc*:

- 0: `raise` (vuelve a lanzar la excepción anterior)
- 1: `raise STACK[-1]` (generar instancia o tipo de excepción en `STACK[-1]`)
- 2: `raise STACK[-2] from STACK[-1]` (generar una instancia o tipo de excepción en `STACK[-2]` con `__cause__` establecido en `STACK[-1]`)

CALL(*argc*)

Llama a un objeto invocable con la cantidad de argumentos especificados por `argc`. En la pila se encuentran (en orden ascendente):

- El llamable
- `self` o `NULL`
- Los argumentos posicionales restantes

`argc` es el total de los argumentos posicionales, excluyendo `self`.

`CALL` extrae todos los argumentos y el objeto invocable de la pila, llama al objeto invocable con esos argumentos y agrega el valor retornado por el objeto invocable.

Added in version 3.11.

Distinto en la versión 3.13: El objeto invocable ahora siempre aparece en la misma posición en la pila.

Distinto en la versión 3.13: Las llamadas con argumentos de palabras clave ahora son manejadas por **`CALL_KW`**.

CALL_KW(*argc*)

Llama a un objeto invocable con la cantidad de argumentos especificados por `argc`, incluidos uno o más argumentos nombrados. En la pila se encuentran (en orden ascendente):

- El llamable

- `self` o `NULL`
- Los argumentos posicionales restantes
- Los argumentos nombrados
- Un **`tuple`** de nombres de argumentos de palabras clave

`argc` es el total de argumentos posicionales y nombrados, excluido `self`. La longitud de la tupla de nombres de argumentos de palabras clave es la cantidad de argumentos nombrados.

`CALL_KW` extrae todos los argumentos, los nombres de palabras clave y el objeto invocable de la pila, llama al objeto invocable con esos argumentos y envía el valor de retorno devuelto por el objeto invocable.

Added in version 3.13.

CALL_FUNCTION_EX(*flags*)

Llama a un objeto invocable con un conjunto variable de argumentos posicionales y de palabras clave. Si se establece el bit más bajo de *flags*, la parte superior de la pila contiene un objeto de mapeo que contiene argumentos de palabras clave adicionales. Antes de que se llame al invocable, el objeto de mapeo y el objeto iterable se «desempaquetan» y su contenido se pasa como palabra clave y argumentos posicionales, respectivamente. `CALL_FUNCTION_EX` saca todos los argumentos y el objeto invocable de la pila, llama al objeto invocable con esos argumentos y empuja el valor de retorno retornado por el objeto invocable.

Added in version 3.6.

PUSH_NULL

Pushes a `NULL` to the stack. Used in the call sequence to match the `NULL` pushed by **`LOAD_METHOD`** for non-method calls.

Added in version 3.11.

MAKE_FUNCTION

Pushes a new function object on the stack built from the code object at `STACK[-1]`.

Distinto en la versión 3.10: El valor indicador `0x04` es una tupla de cadena de caracteres en vez de un diccionario

Distinto en la versión 3.11: Se eliminó el nombre calificado en `STACK[-1]`.

Distinto en la versión 3.13: Se eliminaron los atributos de funciones adicionales en la pila, señalados por indicadores oparg. Ahora utilizan **`SET_FUNCTION_ATTRIBUTE`**.

SET_FUNCTION_ATTRIBUTE(*flag*)

Establece un atributo en un objeto de función. Espera que la función esté en `STACK[-1]` y que el valor del atributo esté establecido en `STACK[-2]`; consume ambos y deja la función en `STACK[-1]`. El indicador determina qué atributo se debe establecer:

- `0x01`, una tupla de valores predeterminados para solo parámetros posicionales y posicionales o de palabras clave en orden posicional
- `0x02` un diccionario de valores predeterminados de solo palabras clave
- `0x04` una tupla de cadenas de caracteres que contiene anotaciones de parámetros
- `0x08` una tupla que contiene celdas para variables libres, haciendo un cierre (*closure*)
- `0x10` the annotate function for the function object

Added in version 3.13.

Distinto en la versión 3.14: Added `0x10` to indicate the annotate function for the function object.

BUILD_SLICE(*argc*)

Coloca un objeto de segmento en la pila. *argc* debe ser 2 o 3. Si es 2, implementa:

```
end = STACK.pop()
start = STACK.pop()
STACK.append(slice(start, end))
```

Si es 3, implementa:

```
step = STACK.pop()
end = STACK.pop()
start = STACK.pop()
STACK.append(slice(start, end, step))
```

Consulte la función incorporada **`slice()`** para obtener más información.

EXTENDED_ARG(*ext*)

Prefija cualquier código de operación que tenga un argumento demasiado grande para caber en el byte predeterminado. *ext* contiene un byte adicional que actúa como bits más altos en el argumento. Para cada opcode, como máximo se permiten tres prefijos `EXTENDED_ARG`, formando un argumento de dos bytes a cuatro bytes.

CONVERT_VALUE(*oparg*)

Convertir valor a cadena, dependiendo de `oparg`:

```
value = STACK.pop()
result = func(value)
STACK.append(result)
```

- `oparg == 1`: llama **`str()`** con *value*
- `oparg == 2`: llama **`repr()`** con *value*
- `oparg == 3`: llama **`ascii()`** con *value*

Used for implementing formatted string literals (f-strings).

Added in version 3.13.

FORMAT_SIMPLE

Formatea el valor en la parte superior de la pila:

```
value = STACK.pop()
result = value.__format__("")
STACK.append(result)
```

Used for implementing formatted string literals (f-strings).

Added in version 3.13.

FORMAT_WITH_SPEC

Formatea el valor dado con la especificación de formato dada:

```
spec = STACK.pop()
value = STACK.pop()
result = value.__format__(spec)
STACK.append(result)
```

Used for implementing formatted string literals (f-strings).

Added in version 3.13.

MATCH_CLASS(*count*)

`STACK[-1]` es una tupla de nombres de atributos de palabras clave, `STACK[-2]` es la clase con la que se realiza la comparación y `STACK[-3]` es el sujeto de la comparación. *count* es la cantidad de subpatrones posicionales.

Inserte `STACK[-1]`, `STACK[-2]` y `STACK[-3]`. Si `STACK[-3]` es una instancia de `STACK[-2]` y tiene los atributos posicionales y de palabras clave requeridos por *count* y `STACK[-1]`, inserte una tupla de atributos extraídos. De lo contrario, inserte `None`.

Added in version 3.10.

Distinto en la versión 3.11: Anteriormente, esta instrucción también generaba un valor booleano que indicaba éxito (`True`) o falla (`False`).

RESUME(*context*)

Un no-op. Realiza comprobaciones internas de seguimiento, depuración y optimización.

The `context` operand consists of two parts. The lowest two bits indicate where the `RESUME` occurs:

- `0` El inicio de una función, que no es un generador, ni una corrutina ni un generador asíncrono
- `1` Después de una expresión `yield`
- `2` Después de una expresión `yield from`
- `3` Después de una expresión `await`

El siguiente bit es `1` si RESUME está en la profundidad de excepción `1`, y `0` en caso contrario.

Added in version 3.11.

Distinto en la versión 3.13: El valor de oparg cambió para incluir información sobre except-depth

RETURN_GENERATOR

Crea un generador, una corrutina o un generador asincrónico a partir del marco actual. Se utiliza como primer código de operación del objeto de código para los elementos invocables mencionados anteriormente. Borra el marco actual y devuelve el generador recién creado.

Added in version 3.11.

SEND(*delta*)

Equivalente a `STACK[-1] = STACK[-2].send(STACK[-1])`. Se utiliza en las instrucciones `yield from` y `await`.

Si la llamada genera **`StopIteration`**, extrae el valor superior de la pila, inserta el atributo `value` de la excepción e incrementa el contador de bytecode en *delta*.

Added in version 3.11.

HAVE_ARGUMENT

En realidad, no se trata de un código de operación. Identifica la línea divisoria entre los códigos de operación en el rango [0,255] que no utilizan su argumento y los que sí lo hacen (`< HAVE_ARGUMENT` y `>= HAVE_ARGUMENT`, respectivamente).

Si su aplicación utiliza pseudoinstrucciones o instrucciones especializadas, utilice la colección **`hasarg`** en su lugar.

Distinto en la versión 3.6: Ahora cada instrucción tiene un argumento, pero los códigos de operación `<HAVE_ARGUMENT` la ignoran. Antes, solo los códigos de operación `> = HAVE_ARGUMENT` tenían un argumento.

Distinto en la versión 3.12: Pseudo instructions were added to the **`dis`** module, and for them it is not true that comparison with `HAVE_ARGUMENT` indicates whether they use their arg.

Obsoleto desde la versión 3.13: Utilice **`hasarg`** en su lugar.

CALL_INTRINSIC_1

Llama a una función intrínseca con un argumento. Pasa `STACK[-1]` como argumento y establece `STACK[-1]` como resultado. Se utiliza para implementar una funcionalidad que no es crítica para el rendimiento.

El operando determina qué función intrínseca se llama:

Operando	Descripción
`INTRINSIC_1_INVALID`	No válido
`INTRINSIC_PRINT`	Imprime el argumento en la salida estándar. Se utiliza en REPL.
`INTRINSIC_IMPORT_STAR`	Ejecuta `import *` para el módulo nombrado.
`INTRINSIC_STOPITERATION_ERROR`	Extrae el valor de retorno de una excepción `StopIteration`.
`INTRINSIC_ASYNC_GEN_WRAP`	Envuelve un valor de generador asincrónico
`INTRINSIC_UNARY_POSITIVE`	Realiza la operación unaria `+`
`INTRINSIC_LIST_TO_TUPLE`	Convierte una lista en una tupla
`INTRINSIC_TYPEVAR`	Crea un **`typing.TypeVar`**
`INTRINSIC_PARAMSPEC`	Crea un **`typing.ParamSpec`**
`INTRINSIC_TYPEVARTUPLE`	Crea un **`typing.TypeVarTuple`**
`INTRINSIC_SUBSCRIPT_GENERIC`	Devuelve **`typing.Generic`** subíndice con el argumento

`INTRINSIC_TYPEALIAS`	Crea un `typing.TypeAliasType`; se utiliza en la declaración `type`. El argumento es una tupla con el nombre del alias de tipo, los parámetros de tipo y el valor.

Added in version 3.12.

CALL_INTRINSIC_2

Llama a una función intrínseca con dos argumentos. Se utiliza para implementar una funcionalidad que no es crítica para el rendimiento:

```
arg2 = STACK.pop()
arg1 = STACK.pop()
result = intrinsic2(arg1, arg2)
STACK.append(result)
```

El operando determina qué función intrínseca se llama:

Operando	**Descripción**
`INTRINSIC_2_INVALID`	No válido
`INTRINSIC_PREP_RERAISE_STAR`	Calcula el `ExceptionGroup` que se debe aumentar desde un `try-except*`.
`INTRINSIC_TYPEVAR_WITH_BOUND`	Crea un `typing.TypeVar` con un límite.
`INTRINSIC_TYPEVAR_WITH_CONSTRAINTS`	Crea un `typing.TypeVar` con restricciones.
`INTRINSIC_SET_FUNCTION_TYPE_PARAMS`	Establece el atributo `__type_params__` de una función.

Added in version 3.12.

LOAD_SPECIAL

Performs special method lookup on `STACK[-1]`. If `type(STACK[-1]).__xxx__` is a method, leave `type(STACK[-1]).__xxx__; STACK[-1]` on the stack. If `type(STACK[-1]).__xxx__` is not a method, leave `STACK[-1].__xxx__; NULL` on the stack.

Added in version 3.14.

Pseudo-instrucciones

Estos códigos de operación no aparecen en el código de bytes de

Python. El compilador los utiliza, pero se reemplazan por códigos de operación reales o se eliminan antes de que se genere el código de bytes.

SETUP_FINALLY(*target*)

Configure un controlador de excepciones para el siguiente bloque de código. Si se produce una excepción, el nivel de pila de valores se restaura a su estado actual y el control se transfiere al controlador de excepciones en `target`.

SETUP_CLEANUP(*target*)

Como `SETUP_FINALLY`, pero en caso de excepción también envía la última instrucción (`lasti`) a la pila para que `RERAISE` pueda restaurarla. Si se produce una excepción, el nivel de pila de valores y la última instrucción del marco se restauran a su estado actual y el control se transfiere al controlador de excepciones en `target`.

SETUP_WITH(*target*)

Como `SETUP_CLEANUP`, pero en caso de una excepción, se extrae un elemento más de la pila antes de que el control se transfiera al controlador de excepciones en `target`.

Esta variante se utiliza en las construcciones **`with`** y **`async with`**, que envían el valor de retorno de **`__enter__()`** o **`__aenter__()`** del administrador de contexto a la pila.

POP_BLOCK

Marca el final del bloque de código asociado con el último `SETUP_FINALLY`, `SETUP_CLEANUP` o `SETUP_WITH`.

LOAD_CONST_IMMORTAL(*consti*)

Works as **`LOAD_CONST`**, but is more efficient for immortal objects.

JUMP
JUMP_NO_INTERRUPT

Instrucciones de salto relativo no dirigido que son reemplazadas por sus contrapartes dirigidas (hacia adelante/hacia atrás) por el ensamblador.

JUMP_IF_TRUE
JUMP_IF_FALSE

Conditional jumps which do not impact the stack. Replaced by the sequence `COPY 1`, `TO_BOOL`, `POP_JUMP_IF_TRUE/FALSE`.

LOAD_CLOSURE(*i*)

Envía una referencia a la celda contenida en la ranura `i` del almacenamiento de «valores locales rápidos».

Tenga en cuenta que `LOAD_CLOSURE` se reemplaza con `LOAD_FAST` en el ensamblador.

Distinto en la versión 3.13: Este código de operación ahora es una pseudoinstrucción.

Colecciones opcode

Estas colecciones se proporcionan para la introspección automática de instrucciones de bytecode:

Distinto en la versión 3.12: Las colecciones ahora también contienen pseudoinstrucciones e instrucciones instrumentadas. Se trata de códigos de operación con valores `>= MIN_PSEUDO_OPCODE` y `>= MIN_INSTRUMENTED_OPCODE`.

dis.opname

Secuencia de nombres de operaciones, indexable utilizando el bytecode.

dis.opmap

Nombres de operaciones de mapeo de diccionario a bytecodes.

dis.cmp_op

Secuencia de todos los nombres de operaciones de comparación.

dis.hasarg

Secuencia de códigos de bytes que utilizan su argumento.

Added in version 3.12.

dis.hasconst

Secuencia de bytecodes que acceden a una constante.

dis.hasfree

Sequence of bytecodes that access a free (closure) variable. “free” in this context refers to names in the current scope that are referenced by inner scopes or names in outer scopes that are referenced from this scope. It does *not* include references to global or builtin scopes.

dis.hasname

Secuencia de bytecodes que acceden a un atributo por nombre.

dis.hasjump

Secuencia de códigos de bytes que tienen un destino de salto. Todos los saltos son relativos.

Added in version 3.13.

dis.haslocal

Secuencia de códigos de bytes que acceden a una variable local.

dis.hascompare

Secuencia de bytecodes de operaciones booleanas.

dis.hasexc

Secuencia de códigos de bytes que establecen un controlador de excepciones.

Added in version 3.12.

dis.hasjrel

Secuencia de bytecodes que tienen un objetivo de salto relativo.

Obsoleto desde la versión 3.13: Ahora todos los saltos son relativos. Utilice `hasjump`.

dis.hasjabs

Secuencia de bytecodes que tienen un objetivo de salto absoluto.

Obsoleto desde la versión 3.13: Ahora todos los saltos son relativos. Esta lista está vacía.

`pickletools` — Tools for pickle developers

Código fuente: Lib/pickletools.py [https://github.com/python/cpython/tree/3.14/Lib/pickletools.py]

This module contains various constants relating to the intimate details of the **`pickle`** module, some lengthy comments about the implementation, and a few useful functions for analyzing pickled data. The contents of this module are useful for Python core developers who are working on the **`pickle`**; ordinary users of the **`pickle`** module probably won't find the **`pickletools`** module relevant.

Command-line usage

Added in version 3.2.

Cuando se invoca desde la línea de comandos, `python -m pickletools` desensamblará el contenido de uno o más archivos pickle. Tenga en cuenta que si desea ver el objeto Python almacenado en el pickle en lugar de los detalles del formato de pickle, es posible que desee utilizar `-m pickle` en su lugar. Sin embargo, cuando el archivo de pickle que desea examinar proviene de una fuente que no es de confianza, `-m pickletools` es una opción más segura porque no ejecuta el código de bytes de pickle.

Por ejemplo, con una tupla `(1, 2)` pickled en el archivo `x.pickle`:

```
$ python -m pickle x.pickle
(1, 2)

$ python -m pickletools x.pickle
```

```
    0: \x80 PROTO      3
    2: K    BININT1    1
    4: K    BININT1    2
    6: \x86 TUPLE2
    7: q    BINPUT     0
    9: .    STOP
highest protocol among opcodes = 2
```

Command-line options

-a, --annotate

Anote cada línea con una breve descripción del código de operación.

-o, --output=<file>

Nombre de un archivo donde se debe escribir la salida.

-l, --indentlevel=<num>

Número de espacios en blanco por los que se aplica una sangría a un nuevo nivel de MARK.

-m, --memo

Cuando se desensamblan varios objetos, conserve la nota entre los ensamblajes.

-p, --preamble=<preamble>

When more than one pickle file is specified, print given preamble before each disassembly.

pickle_file

A pickle file to read, or - to indicate reading from standard input.

Programmatic interface

pickletools.dis(*pickle*, *out=None*, *memo=None*, *indentlevel=4*, *annotate=0*)

Produce un desensamblado simbólico del pickle en el objeto similar a un archivo *salida*, de forma predeterminada en `sys.stdout`. *pickle* puede ser una cadena o un objeto similar a un archivo. *memo* puede ser un diccionario Python que se utilizará como nota del pickle; se puede utilizar para realizar ensamblajes de desuso en varios pickles creados por el mismo selector. Los niveles sucesivos, indicados por los códigos de operación `MARK` en la secuencia, son indentados por espacios *indentlevel*. Si se da un valor distinto de cero a *anotar*, cada código de operación de la salida se anota con una breve descripción. El valor de *anotar* se utiliza como sugerencia para la columna donde debe comenzar la anotación.

Distinto en la versión 3.2: Added the *annotate* parameter.

pickletools.genops(*pickle*)

Proporciona un iterator sobre todos los códigos de operación en un pickle, retornando una secuencia de triples `(opcode, arg, pos)`. *opcode* es una instancia de una clase **`OpcodeInfo`**; *arg* es el valor descodificado, como un objeto Python, del argumento del código de operación; *pos* es la posición en la que se encuentra este código de operación. *pickle* puede ser una cadena o un objeto similar a un archivo.

pickletools.optimize(*picklestring*)

Retorna una nueva cadena pickle equivalente después de eliminar los códigos de operación `PUT` no utilizados. El pickle optimizado es más corto, toma menos tiempo de transmisión, requiere menos espacio de almacenamiento y se restaura de manera más eficiente.

Servicios Específicos para MS Windows

Este capítulo describe los módulos que sólo están disponibles en plataformas MS Windows.

- **`msvcrt`** — Useful routines from the MS VC++ runtime
    - Operaciones con archivos
    - Consola E/S
    - Otras funciones
- **`winreg`** — Windows registry access
    - Funciones
    - Constantes
        - HKEY_* Constantes
        - Access Rights
            - Específico de 64 bits
        - Tipos de valor
    - Objetos de control del registro
- **`winsound`** — Sound-playing interface for Windows

`msvcrt` — Useful routines from the MS VC+ + runtime

These functions provide access to some useful capabilities on Windows platforms. Some higher-level modules use these functions to build the Windows implementations of their services. For example, the `getpass` module uses this in the implementation of the `getpass()` function.

Más información sobre estas funciones se pueden encontrar en la documentación de la API de la plataforma.

El módulo implementa las variantes tanto de caracteres normales como amplios de la API E/S de la consola (se codifican en más de 8 bits, pudiendo llegar hasta 32). La API normal se ocupa solamente de caracteres ASCII y es de uso limitado a aplicaciones internacionales. La API para caracteres amplios se recomienda usar siempre que sea posible.

Availability: Windows.

Distinto en la versión 3.3: Las operaciones en este módulo lanzan ahora `OSError` donde antes se lanzaba `IOError`.

Operaciones con archivos

msvcrt.locking(*fd*, *mode*, *nbytes*)

Lock part of a file based on file descriptor *fd* from the C runtime. Raises `OSError` on failure. The locked region of the file extends from the current file position for *nbytes* bytes, and may continue beyond the end of the file. *mode* must be one of the **`LK_*`** constants listed below. Multiple regions in a file may be locked at the same time, but may not overlap. Adjacent regions are not merged; they must be unlocked individually.

Lanza un evento de auditoría `msvcrt.locking` con los argumentos `fd`, `mode`, `nbytes`.

msvcrt.LK_LOCK
msvcrt.LK_RLCK

Locks the specified bytes. If the bytes cannot be locked, the program immediately tries again after 1 second. If, after 10 attempts, the bytes cannot be locked, **`OSError`** is raised.

msvcrt.LK_NBLCK
msvcrt.LK_NBRLCK

Bloquea los bytes especificados. Si no se pueden bloquear, lanza una excepción **`OSError`**.

msvcrt.LK_UNLCK

Desbloquea los bytes especificados que han sido previamente bloqueados.

msvcrt.setmode(*fd*, *flags*)

Establece el modo traducción del final de línea del descriptor de un archivo *fd*. Si se establece como modo texto, *flags* debería ser **`os.O_TEXT`**; para establecerlo como modo binario, debería ser **`os.O_BINARY`**.

msvcrt.open_osfhandle(*handle*, *flags*)

Create a C runtime file descriptor from the file handle *handle*. The *flags* parameter should be a bitwise OR of **`os.O_APPEND`**, **`os.O_RDONLY`**, **`os.O_TEXT`** and **`os.O_NOINHERIT`**. The returned file descriptor may be used as a parameter to **`os.fdopen()`** to create a file object.

The file descriptor is inheritable by default. Pass **`os.O_NOINHERIT`** flag to make it non inheritable.

Lanza un evento de auditoría `msvcrt.open_osfhandle` con los argumentos `handle`, `flags`.

msvcrt.get_osfhandle(*fd*)

Return the file handle for the file descriptor *fd*. Raises **`OSError`** if *fd* is not recognized.

Lanza un evento de auditoría `msvcrt.get_osfhandle` con el argumento `fd`.

Consola E/S

msvcrt.kbhit()

Returns a nonzero value if a keypress is waiting to be read. Otherwise, return 0.

msvcrt.getch()

Read a keypress and return the resulting character as a byte string. Nothing is echoed to the console. This call will block if a keypress is not already available, but will not wait for `Enter` to be pressed. If the pressed key was a special function key, this will return `'\000'` or `'\xe0'`; the next call will return the keycode. The `Control-C` keypress cannot be read with this function.

msvcrt.getwch()

Variante de carácter amplio de **`getch()`**, retornando un valor Unicode.

msvcrt.getche()

Similar to **`getch()`**, but the keypress will be echoed if it represents a printable character.

msvcrt.getwche()

Variante de carácter amplio de **`getche()`**, retornando un valor Unicode.

msvcrt.putch(*char*)

Imprime la cadena de caracteres de bytes *char* a la consola sin almacenamiento en buffer.

msvcrt.putwch(*unicode_char*)

Variante de carácter amplio de `putch()`, admitiendo un valor Unicode.

msvcrt.ungetch(*char*)

Provoca que la cadena de caracteres de bytes *char* sea «colocada de nuevo» en el buffer de la consola, será el siguiente carácter que lea la función `getch()` o `getche()`.

msvcrt.ungetwch(*unicode_char*)

Variante de carácter amplio de `ungetch()`, admitiendo un valor Unicode.

Otras funciones

msvcrt.heapmin()

Force the `malloc()` heap to clean itself up and return unused blocks to the operating system. On failure, this raises `OSError`.

msvcrt.set_error_mode(*mode*)

Changes the location where the C runtime writes an error message for an error that might end the program. *mode* must be one of the `OUT_*` constants listed below or `REPORT_ERRMODE`. Returns the old setting or -1 if an error occurs. Only available in debug build of Python.

msvcrt.OUT_TO_DEFAULT

Error sink is determined by the app's type. Only available in debug build of Python.

msvcrt.OUT_TO_STDERR

Error sink is a standard error. Only available in debug build of Python.

msvcrt.OUT_TO_MSGBOX

Error sink is a message box. Only available in debug build of Python.

msvcrt.REPORT_ERRMODE

Report the current error mode value. Only available in debug build of Python.

msvcrt.CrtSetReportMode(*type*, *mode*)

Specifies the destination or destinations for a specific report type generated by `_CrtDbgReport()` in the MS VC++ runtime. *type* must be one of the `CRT_*` constants listed below. *mode* must be one of the `CRTDBG_*` constants listed below. Only available in debug build of Python.

msvcrt.CrtSetReportFile(*type*, *file*)

After you use `CrtSetReportMode()` to specify `CRTDBG_MODE_FILE`, you can specify the file handle to receive the message text. *type* must be one of the `CRT_*` constants listed below. *file* should be the file handle your want specified. Only available in debug build of Python.

msvcrt.CRT_WARN

Warnings, messages, and information that doesn't need immediate attention.

msvcrt.CRT_ERROR

Errors, unrecoverable problems, and issues that require immediate attention.

msvcrt.CRT_ASSERT

Assertion failures.

msvcrt.CRTDBG_MODE_DEBUG

Writes the message to the debugger's output window.

msvcrt.CRTDBG_MODE_FILE

Writes the message to a user-supplied file handle. **`CrtSetReportFile()`** should be called to define the specific file or stream to use as the destination.

msvcrt.CRTDBG_MODE_WNDW

Creates a message box to display the message along with the `Abort`, `Retry`, and `Ignore` buttons.

msvcrt.CRTDBG_REPORT_MODE

Returns current *mode* for the specified *type*.

msvcrt.CRT_ASSEMBLY_VERSION

The CRT Assembly version, from the `crtassem.h` header file.

msvcrt.VC_ASSEMBLY_PUBLICKEYTOKEN

The VC Assembly public key token, from the `crtassem.h` header file.

msvcrt.LIBRARIES_ASSEMBLY_NAME_PREFIX

The Libraries Assembly name prefix, from the `crtassem.h` header file.

`winreg` — Windows registry access

Estas funciones exponen la API de registro de Windows a Python. En lugar de utilizar un número entero como identificador de registro, se utiliza un handle object para garantizar que los identificadores se cierren correctamente, incluso si el programador se niega a cerrarlos explícitamente.

Availability: Windows.

Distinto en la versión 3.3: Varias funciones de este módulo solían lanzar un `WindowsError`, que ahora es un alias de `OSError`.

Funciones

Este módulo ofrece las siguientes funciones:

winreg.CloseKey(*hkey*)

Cierra una clave de registro abierta previamente. El argumento *hkey* especifica una clave abierta previamente.

Nota

Si *hkey* no se cierra con este método (o mediante `hkey.Close()`), se cierra cuando Python destruye el objeto *hkey*.

winreg.ConnectRegistry(*computer_name*, *key*)

Establece una conexión con un identificador de registro predefinido en otra computadora y retorna un handle object.

computer_name es el nombre de la computadora remota, de la forma `r"\\computername"`. Si es `None`, se utiliza la computadora local.

key es el identificador predefinido al que conectarse.

El valor de retorno es el identificador de la llave abierta. Si la función falla, se lanza una excepción **`OSError`**.

Lanza un auditing event `winreg.ConnectRegistry` con argumentos `computer_name`, `key`.

Distinto en la versión 3.3: Ver above.

winreg.CreateKey(*key*, *sub_key*)

Crea o abre la clave especificada, retornando un handle object.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

sub_key es una cadena de caracteres que nombra la clave que este método abre o crea.

Si *key* es una de las claves predefinidas, *sub_key* puede ser `None`. En ese caso, el identificador retornado es el mismo identificador de clave que se pasó a la función.

Si la clave ya existe, esta función abre la clave existente.

El valor de retorno es el identificador de la llave abierta. Si la función falla, se lanza una excepción **`OSError`**.

Lanza un auditing event `winreg.CreateKey` con los argumentos `key`, `sub_key`, `access`.

Lanza un auditing event `winreg.OpenKey/result` con el argumento `key`.

Distinto en la versión 3.3: Ver above.

winreg.CreateKeyEx(*key*, *sub_key*, *reserved=0*, *access=KEY_WRITE*)

Crea o abre la clave especificada, retornando un handle object.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

sub_key es una cadena de caracteres que nombra la clave que este método abre o crea.

reserved es un número entero reservado y debe ser cero. El valor predeterminado es cero.

access es un número entero que especifica una máscara de acceso que describe el acceso de seguridad deseado para la clave. El valor predeterminado es **`KEY_WRITE`**. Ver Access Rights para otros valores permitidos.

Si *key* es una de las claves predefinidas, *sub_key* puede ser `None`. En ese caso, el identificador retornado es el mismo identificador de clave que se pasó a la función.

Si la clave ya existe, esta función abre la clave existente.

El valor de retorno es el identificador de la llave abierta. Si la función falla, se lanza una excepción **`OSError`**.

Lanza un auditing event `winreg.CreateKey` con los argumentos `key`, `sub_key`, `access`.

Lanza un auditing event `winreg.OpenKey/result` con el argumento `key`.

Added in version 3.2.

Distinto en la versión 3.3: Ver above.

winreg.DeleteKey(*key*, *sub_key*)

Elimina la clave especificada.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

sub_key es una cadena de caracteres que debe ser una subclave de la clave identificada por el parámetro *key*. Este valor no debe ser `None`, y es posible que la clave no tenga subclaves.

Este método no puede eliminar claves con subclaves.

Si el método tiene éxito, se elimina toda la clave, incluidos todos sus valores. Si el método falla, se lanza una excepción **`OSError`**.

Lanza un auditing event `winreg.DeleteKey` con los argumentos `key`, `sub_key`, `access`.

Distinto en la versión 3.3: Ver above.

winreg.DeleteKeyEx(*key*, *sub_key*, *access=KEY_WOW64_64KEY*, *reserved=0*)

Elimina la clave especificada.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

sub_key es una cadena de caracteres que debe ser una subclave de la clave identificada por el parámetro *key*. Este valor no debe ser `None`, y es posible que la clave no tenga subclaves.

reserved es un número entero reservado y debe ser cero. El valor predeterminado es cero.

access es un número entero que especifica una máscara de acceso que describe el acceso de seguridad deseado para la clave. El valor predeterminado es **`KEY_WOW64_64KEY`**. En Windows de 32-bit, las constantes WOW64 son ignoradas. Ver Access Rights para otros valores permitidos.

Este método no puede eliminar claves con subclaves.

Si el método tiene éxito, se elimina toda la clave, incluidos todos sus valores. Si el método falla, se lanza una excepción **`OSError`**.

En versiones de Windows no compatibles, se lanza **`NotImplementedError`**.

Lanza un auditing event `winreg.DeleteKey` con los argumentos `key`, `sub_key`, `access`.

Added in version 3.2.

Distinto en la versión 3.3: Ver above.

winreg.DeleteValue(*key*, *value*)

Elimina un valor con nombre de una clave de registro.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

value es una cadena que identifica el valor a eliminar.

Lanza un auditing event `winreg.DeleteValue` con argumentos `key`, `value`.

winreg.EnumKey(*key*, *index*)

Enumera las subclaves de una clave de registro abierta y retorna una cadena de caracteres.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

index es un número entero que identifica el índice de la clave a recuperar.

La función recupera el nombre de una subclave cada vez que se llama. Normalmente se llama repetidamente hasta que se lanza una excepción **`OSError`**, lo que indica que no hay más valores disponibles.

Lanza un auditing event `winreg.EnumKey` con argumentos `key`, `index`.

Distinto en la versión 3.3: Ver above.

winreg.EnumValue(*key*, *index*)

Enumera los valores de una clave de registro abierta y retorna una tupla.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

index es un número entero que identifica el índice del valor a recuperar.

La función recupera el nombre de una subclave cada vez que se llama. Normalmente se llama repetidamente, hasta que se lanza una excepción **`OSError`**, lo que indica que no hay más valores.

El resultado es una tupla de 3 elementos:

Índice	Significado
0	Una cadena de caracteres que identifica el nombre del valor
1	Un objeto que contiene los datos del valor y cuyo tipo depende del tipo de registro subyacente
2	Un número entero que identifica el tipo de datos de valor (consulte la tabla en los documentos para **`SetValueEx()`**)

Lanza un auditing event `winreg.EnumValue` con argumentos `key`, `index`.

Distinto en la versión 3.3: Ver above.

winreg.ExpandEnvironmentStrings(*str*)

Expande los marcadores de posición de la variable de entorno `%NAME%` en cadenas como **`REG_EXPAND_SZ`**:

```
>>> ExpandEnvironmentStrings('%windir%')
```

```
'C:\\Windows'
```

Lanza un auditing event `winreg.ExpandEnvironmentStrings` con el argumento `str`.

winreg.FlushKey(*key*)

Escribe todos los atributos de una clave en el registro.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

No es necesario llamar a **`FlushKey()`** para cambiar una clave. Los cambios en el registro se descargan en el disco mediante el registro mediante su vaciador diferido. Los cambios en el registro también se vacían en el disco cuando se apaga el sistema. A diferencia de **`CloseKey()`**, el método **`FlushKey()`** retorna solo cuando todos los datos se han escrito en el registro. Una aplicación solo debe llamar a **`FlushKey()`** si requiere absoluta certeza de que los cambios de registro están en el disco.

Nota

Si no sabe si se requiere una llamada **`FlushKey()`**, probablemente no lo sea.

winreg.LoadKey(*key*, *sub_key*, *file_name*)

Crea una subclave bajo la clave especificada y almacena la información de registro de un archivo especificado en esa subclave.

key es un identificador retornado por **`ConnectRegistry()`** o una de las constantes **`HKEY_USERS`** o **`HKEY_LOCAL_MACHINE`**.

sub_key es una cadena de caracteres que identifica la subclave a cargar.

file_name es el nombre del archivo desde el que cargar los datos de registro. Este archivo debe haber sido creado con la función `SaveKey()`. En el sistema de archivos de la tabla de asignación de archivos (FAT), es posible que el nombre del archivo no tenga extensión.

Una llamada a `LoadKey()` falla si el proceso de llamada no tiene el privilegio **`SE_RESTORE_PRIVILEGE`**. Tenga en cuenta que los privilegios son diferentes de los permisos; consulte la RegLoadKey documentation [https://msdn.microsoft.com/en-us/library/ms724889%28v=VS.85%29.aspx] para obtener más detalles.

Si *key* es un identificador retornado por `ConnectRegistry()`, entonces la ruta especificada en *file_name* es relativa a la computadora remota.

Lanza un auditing event `winreg.LoadKey` con los argumentos `key`, `sub_key`, `file_name`.

winreg.OpenKey(*key*, *sub_key*, *reserved=0*, *access=KEY_READ*)
winreg.OpenKeyEx(*key*, *sub_key*, *reserved=0*, *access=KEY_READ*)

Abre la clave especificada, retornando a handle object.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

sub_key es una cadena de caracteres que identifica la sub_key para abrir.

reserved es un número entero reservado y debe ser cero. El valor predeterminado es cero.

access es un número entero que especifica una máscara de acceso que describe el acceso de seguridad deseado para la clave. El valor predeterminado es `KEY_READ`. Ver Access Rights para otros valores permitidos.

El resultado es un nuevo identificador para la clave especificada.

Si la función falla, se lanza **`OSError`**.

Lanza un auditing event `winreg.OpenKey` con los argumentos `key`, `sub_key`, `access`.

Lanza un auditing event `winreg.OpenKey/result` con el argumento `key`.

Distinto en la versión 3.2: Permite el uso de argumentos con nombre.

Distinto en la versión 3.3: Ver above.

winreg.QueryInfoKey(*key*)

Retorna información sobre una clave, como una tupla.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

El resultado es una tupla de 3 elementos:

Índice	**Significado**
0	Un número entero que indica el número de subclaves que tiene esta clave.
1	Un número entero que da el número de valores que tiene esta clave.
2	Un número entero que indica la última vez que se modificó la clave (si está disponible) como cientos de nanosegundos desde el 1 de enero de 1601.

Lanza un auditing event `winreg.QueryInfoKey` con el argumento `key`.

winreg.QueryValue(*key*, *sub_key*)

Recupera el valor sin nombre de una clave, como una cadena de caracteres.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

sub_key es una cadena de caracteres que contiene el nombre de la subclave con la que está asociado el valor. Si este parámetro es `None` o está vacío, la función recupera el valor establecido por el método **`SetValue()`** para la clave identificada por *key*.

Los valores del registro tienen componentes de nombre, tipo y datos. Este método recupera los datos del primer valor de una clave que tiene un nombre `NULL`. Pero la llamada a la API subyacente no retorna el tipo, así que siempre use **`QueryValueEx()`** si es posible.

Lanza un auditing event `winreg.QueryValue` con los argumentos `key`, `sub_key`, `value_name`.

winreg.QueryValueEx(*key*, *value_name*)

Recupera el tipo y los datos de un nombre de valor especificado asociado con una clave de registro abierta.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

value_name es una cadena de caracteres que indica el valor a consultar.

El resultado es una tupla de 2 elementos:

Índice	Significado
0	El valor del elemento de registro.
1	Un número entero que proporciona el tipo de registro para este valor (consulte la tabla en docs para **`SetValueEx()`**)

Lanza un auditing event `winreg.QueryValue` con los argumentos `key`, `sub_key`, `value_name`.

winreg.SaveKey(*key*, *file_name*)

Guarda la clave especificada y todas sus subclaves en el archivo especificado.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

file_name es el nombre del archivo en el que se guardarán los datos del registro. Este archivo no puede existir ya. Si este nombre de archivo incluye una extensión, no se puede usar en sistemas de archivos de tabla de asignación de archivos (FAT) mediante el método `LoadKey()`.

Si *key* representa una clave en una computadora remota, la ruta descrita por *file_name* es relativa a la computadora remota. El invocador de este método debe poseer el privilegio de seguridad **SeBackupPrivilege**. Tenga en cuenta que los privilegios son diferentes a los permisos – consulte la documentación sobre conflictos entre derechos de usuario y permisos [https://msdn.microsoft.com/en-us/library/ms724878%28v=VS.85%29.aspx] para más detalles.

Esta función pasa `NULL` para *security_attributes* a la API.

Lanza un auditing event `winreg.SaveKey` con los argumentos `key`, `file_name`.

winreg.SetValue(*key*, *sub_key*, *type*, *value*)

Asocia un valor con una clave especificada.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

sub_key es una cadena de caracteres que nombra la subclave con la que está asociado el valor.

type es un número entero que especifica el tipo de datos. Actualmente debe ser `REG_SZ`, lo que significa que solo se admiten cadenas de caracteres. Utilice la función: `SetValueEx()` para admitir otros tipos de datos.

value es una cadena de caracteres que especifica el nuevo valor.

Si la clave especificada por el parámetro *sub_key* no existe, la función SetValue la crea.

Las longitudes de los valores están limitadas por la memoria disponible. Los valores largos (más de 2048 bytes) deben almacenarse como archivos con los nombres de archivo almacenados en el registro de configuración. Esto ayuda a que el registro funcione de manera eficiente.

La clave identificada por el parámetro *key* debe haber sido abierta con acceso **`KEY_SET_VALUE`**.

Lanza un auditing event `winreg.SetValue` con argumentos `key`, `sub_key`, `type`, `value`.

winreg.SetValueEx(*key*, *value_name*, *reserved*, *type*, *value*)

Almacena datos en el campo de valor de una clave de registro abierta.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

value_name es una cadena de caracteres que nombra la subclave con la que está asociado el valor.

reserved puede ser cualquier cosa — cero siempre se pasa a la API.

type es un número entero que especifica el tipo de datos. Consulte Value Types para los tipos disponibles.

value es una cadena de caracteres que especifica el nuevo valor.

Este método también puede establecer un valor adicional e información de tipo para la clave especificada. La clave identificada por el parámetro clave debe haber sido abierta con acceso **`KEY_SET_VALUE`**.

Para abrir la clave, use los métodos **`CreateKey()`** o **`OpenKey()`**.

Las longitudes de los valores están limitadas por la memoria disponible. Los valores largos (más de 2048 bytes) deben almacenarse como archivos con los nombres de archivo almacenados en el registro de configuración. Esto ayuda a que el registro funcione de manera eficiente.

Lanza un auditing event `winreg.SetValue` con argumentos `key`, `sub_key`, `type`, `value`.

winreg.DisableReflectionKey(*key*)

Desactiva la reflexión del registro para los procesos de 32 bits que se ejecutan en un sistema operativo de 64 bits.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

Generalmente lanzará **`NotImplementedError`** si se ejecuta en un sistema operativo de 32 bits.

Si la clave no está en la lista de reflexión, la función tiene éxito pero no tiene ningún efecto. La desactivación de la reflexión de una clave no afecta la reflexión de ninguna subclave.

Lanza un auditing event `winreg.DisableReflectionKey` con el argumento `key`.

winreg.EnableReflectionKey(*key*)

Restaura la reflexión del registro para la clave deshabilitada especificada.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

Generalmente lanzará **`NotImplementedError`** si se ejecuta en un sistema operativo de 32 bits.

La restauración de la reflexión de una clave no afecta la reflexión de ninguna subclave.

Lanza un auditing event `winreg.EnableReflectionKey` con el argumento `key`.

winreg.QueryReflectionKey(*key*)

Determina el estado de reflexión para la clave especificada.

key es una clave ya abierta, o una de las predefinidas HKEY_* constants.

Retorna `True` si la reflexión está deshabilitada.

Generalmente lanzará **`NotImplementedError`** si se ejecuta en un sistema operativo de 32 bits.

Lanza un auditing event `winreg.QueryReflectionKey` con el argumento `key`.

Constantes

The following constants are defined for use in many **`winreg`** functions.

HKEY_* Constantes

winreg.HKEY_CLASSES_ROOT

Las entradas de registro subordinadas a esta clave definen tipos (o clases) de documentos y las propiedades asociadas con esos tipos. Las aplicaciones Shell y COM utilizan la información almacenada en esta clave.

winreg.HKEY_CURRENT_USER

Las entradas de registro subordinadas a esta clave definen las preferencias del usuario actual. Estas preferencias incluyen la configuración de variables de entorno, datos sobre grupos de programas, colores, impresoras, conexiones de red y preferencias de la aplicación.

winreg.HKEY_LOCAL_MACHINE

Las entradas de registro subordinadas a esta clave definen el estado físico de la computadora, incluidos los datos sobre el tipo de bus, la memoria del sistema y el hardware y software instalados.

winreg.HKEY_USERS

Las entradas de registro subordinadas a esta clave definen la configuración de usuario predeterminada para nuevos usuarios en la computadora local y la configuración de usuario para el usuario actual.

winreg.HKEY_PERFORMANCE_DATA

Las entradas de registro subordinadas a esta clave le permiten acceder a los datos de rendimiento. Los datos no se almacenan realmente en el registro; las funciones de registro hacen que el sistema recopile los datos de su fuente.

winreg.HKEY_CURRENT_CONFIG

Contiene información sobre el perfil de hardware actual del sistema informático local.

winreg.HKEY_DYN_DATA

Esta clave no se usa en versiones de Windows posteriores a la 98.

Access Rights

Para más información, ver Registry Key Security and Access [https://msdn.microsoft.com/en-us/library/ms724878%28v=VS.85%29.aspx].

winreg.KEY_ALL_ACCESS

Combina los derechos de acceso STANDARD_RIGHTS_REQUIRED, `KEY_QUERY_VALUE`, `KEY_SET_VALUE`, `KEY_CREATE_SUB_KEY`, `KEY_ENUMERATE_SUB_KEYS`, `KEY_NOTIFY`, y `KEY_CREATE_LINK`.

winreg.KEY_WRITE

Combina los derechos de acceso STANDARD_RIGHTS_WRITE, `KEY_SET_VALUE`, y `KEY_CREATE_SUB_KEY`.

winreg.KEY_READ

Combina los valores STANDARD_RIGHTS_READ, **`KEY_QUERY_VALUE`**, **`KEY_ENUMERATE_SUB_KEYS`**, y **`KEY_NOTIFY`**.

winreg.KEY_EXECUTE

Equivalente a **`KEY_READ`**.

winreg.KEY_QUERY_VALUE

Requerido para consultar los valores de una clave de registro.

winreg.KEY_SET_VALUE

Requerido para crear, eliminar o establecer un valor de registro.

winreg.KEY_CREATE_SUB_KEY

Necesario para crear una subclave de una clave de registro.

winreg.KEY_ENUMERATE_SUB_KEYS

Requerido para enumerar las subclaves de una clave de registro.

winreg.KEY_NOTIFY

Requerido para solicitar notificaciones de cambio para una clave de registro o para subclaves de una clave de registro.

winreg.KEY_CREATE_LINK

Reservado para uso del sistema.

Específico de 64 bits

Para más información, ver Accessing an Alternate Registry View [https://msdn.microsoft.com/en-us/library/aa384129(v=VS.85).aspx].

winreg.KEY_WOW64_64KEY

Indica que una aplicación en Windows de 64 bits debería funcionar en la vista de registro de 64 bits. En Windows de 32 bits, esta constante se ignora.

winreg.KEY_WOW64_32KEY

Indica que una aplicación en Windows de 64 bits debería funcionar en la vista de registro de 32 bits. En Windows de 32 bits, esta constante se ignora.

Tipos de valor

Para más información, ver Registry Value Types [https://msdn.microsoft.com/en-us/library/ms724884%28v=VS.85%29.aspx].

winreg.REG_BINARY

Datos binarios en cualquier forma.

winreg.REG_DWORD

Número de 32 bits.

winreg.REG_DWORD_LITTLE_ENDIAN

Un número de 32 bits en formato little-endian. Equivalente a **`REG_DWORD`**.

winreg.REG_DWORD_BIG_ENDIAN

Un número de 32 bits en formato big-endian.

winreg.REG_EXPAND_SZ

Cadena de caracteres terminada en nulo que contiene referencias a variables de entorno (`%PATH%`).

winreg.REG_LINK

Un enlace simbólico Unicode.

winreg.REG_MULTI_SZ

Una secuencia de cadenas de caracteres terminadas en nulo, terminadas por dos caracteres nulos. (Python maneja esta terminación automáticamente).

winreg.REG_NONE

Sin tipo de valor definido.

winreg.REG_QWORD

Un número de 64 bits.

Added in version 3.6.

winreg.REG_QWORD_LITTLE_ENDIAN

Un número de 64 bits en formato little-endian. Equivalente a `REG_QWORD`.

Added in version 3.6.

winreg.REG_RESOURCE_LIST

Una lista de recursos de controladores de dispositivo.

winreg.REG_FULL_RESOURCE_DESCRIPTOR

Una configuración de hardware.

winreg.REG_RESOURCE_REQUIREMENTS_LIST

Una lista de recursos de hardware.

winreg.REG_SZ

Una cadena de caracteres terminada en nulo.

Objetos de control del registro

Este objeto envuelve un objeto HKEY de Windows y lo cierra automáticamente cuando se destruye. Para garantizar la limpieza, puede llamar al método `Close()` en el objeto, o a la función `CloseKey()`.

Todas las funciones de registro de este módulo retornan uno de estos objetos.

Todas las funciones de registro de este módulo que aceptan un objeto identificador también aceptan un número entero, sin embargo, se recomienda el uso del objeto identificador.

Los objetos de control proporcionan semántica para `__bool__()` – así

```
if handle:
    print("Yes")
```

imprimirá `Yes` si el controlador es válido actualmente (no se ha cerrado o desprendido).

Los objetos de identificador se pueden convertir a un número entero (por ejemplo, usando la función incorporada `int()` function), en cuyo caso se retorna el valor de identificador de Windows subyacente. También puede usar el método `Detach()` para retornar el identificador de enteros y también desconectar el identificador de Windows del objeto identificador.

PyHKEY.Close()

> Cierra el identificador de Windows subyacente.
>
> Si el controlador ya está cerrado, no se lanza ningún error.

PyHKEY.Detach()

> Separa el identificador de Windows del objeto identificador.
>
> El resultado es un número entero que contiene el valor del identificador antes de que se separe. Si el controlador ya está separado o cerrado, esto retornará cero.
>
> Después de llamar a esta función, el identificador se invalida efectivamente, pero el identificador no se cierra. Llamaría a esta función cuando necesite que el identificador Win32 subyacente exista más allá de la vida útil del objeto identificador.
>
> Lanza un auditing event `winreg.PyHKEY.Detach` con el argumento `key`.

PyHKEY.__enter__()

PyHKEY.__exit__(*exc_info)

El objeto HKEY implementa `__enter__()` y `__exit__()` y, por lo tanto, admite el protocolo de contexto para la declaración `with`:

```
with OpenKey(HKEY_LOCAL_MACHINE, "foo") as key:
    ...  # work with key
```

cerrará automáticamente *key* cuando el control abandone el bloque `with`.

`winsound` — Sound-playing interface for Windows

The **`winsound`** module provides access to the basic sound-playing machinery provided by Windows platforms. It includes functions and several constants.

Availability: Windows.

winsound.Beep(*frequency*, *duration*)

> Hace sonar el altavoz del PC. El parámetro *frequency* especifica la frecuencia, en hercio (hz), de la señal de audio y debe estar en el rango de 37 a 32.767 hz. El parámetro *duration* especifica el numero de milisegundo de duración de la señal de audio. Si el sistema no puede hacer sonar el altavoz, se lanza **`RuntimeError`**.

winsound.PlaySound(*sound*, *flags*)

> Llama a la función responsable de **`PlaySound()`** desde el API de la plataforma. El parámetro *sound* puede ser un nombre de archivo, un alias de sonido del sistema, datos de audio como un bytes-like object , o `None`. Su interpretación depende del valor de *flags*, que puede ser una combinación ORed de las constantes descritas a continuación. Si el parámetro de *sound* es `None`, cualquier sonido de forma de onda que se esté reproduciendo en ese momento se detiene. Si el sistema indica un error, se lanza **`RuntimeError``**.

winsound.MessageBeep(*type*=*MB_OK*)

> Llama a la función responsable de **`MensajeBeep()`** de la API de la plataforma. Esto reproduce un sonido como se especifica en el registro. El argumento *type* especifica qué sonido se reproduce; los valores posibles son `-1`,

`MB_ICONASTERISK`, `MB_ICONEXCLAMATION`, `MB_ICONHAND`, `MB_ICONQUESTION`, y `MB_OK`, todos descritos a continuación. El valor `-1` produce un «simple pitido»; este es el último recurso si un sonido no puede ser reproducido de otra manera. Si el sistema indica un error, se lanza **`RuntimeError`**.

winsound.SND_FILENAME

El parámetro *sound* es el nombre de un archivo WAV. No lo uses con **`SND_ALIAS`**.

winsound.SND_ALIAS

El parámetro *sound* es un nombre de asociación de sonido del registro. Si el registro no contiene tal nombre, reproduce el sonido por defecto del sistema a menos que **`SND_NODEFAULT`** también se especifique. Si no se registra ningún sonido por defecto, se lanza **`RuntimeError`**. No lo uses con **`SND_FILENAME`**.

Todos los sistemas Win32 soportan al menos lo siguiente; la mayoría de los sistemas soportan muchos más:

`PlaySound()` ***name***	**Panel de control correspondiente nombre (*name*) del sonido**
`'SystemAsterisk'`	Asterisco
`'SystemExclamation'`	Exclamación
`'SystemExit'`	Salir de Windows
`'SystemHand'`	Parada crítica
`'SystemQuestion'`	Pregunta

Por ejemplo:

```
import winsound
# Play Windows exit sound.
winsound.PlaySound("SystemExit", winsound.SND_ALIAS

# Probably play Windows default sound, if any is re
# "*" probably isn't the registered name of any sou
winsound.PlaySound("*", winsound.SND_ALIAS)
```

winsound.SND_LOOP

Reproducir el sonido repetidamente. El flag `SND_ASYNC` también debe ser usada para evitar el bloqueo. No puede ser usada con `SND_MEMORY`.

winsound.SND_MEMORY

El parámetro *sound* de `PlaySound()` es una imagen de memoria de un archivo WAV, como un bytes-like object.

Nota

Este módulo no admite la reproducción desde una imagen de memoria de forma sincrónica, por lo que una combinación de este indicador y `SND_ASYNC` se lanza `RuntimeError`.

winsound.SND_PURGE

Detiene la reproducción de todas las instancias de un sonido específico.

Nota

Esta flag no esta soportado en las plataformas modernas de Windows.

winsound.SND_ASYNC

Retorna inmediatamente, permitiendo que los sonidos se reproduzcan asincrónicamente.

winsound.SND_NODEFAULT

Si no se puede encontrar el audio especificado, no reproduce el sonido predeterminado del sistema.

winsound.SND_NOSTOP

No interrumpe los sonidos que se están reproduciendo.

winsound.SND_NOWAIT

Retorna inmediatamente en caso de que el controlador de sonido está ocupado.

Nota

Esta flag no esta soportado en las plataformas modernas de Windows.

winsound.SND_APPLICATION

The *sound* parameter is an application-specific alias in the registry. This flag can be combined with the `SND_ALIAS` flag to specify an application-defined sound alias.

winsound.SND_SENTRY

Triggers a SoundSentry event when the sound is played.

Added in version 3.14.

winsound.SND_SYNC

The sound is played synchronously. This is the default behavior.

Added in version 3.14.

winsound.SND_SYSTEM

Assign the sound to the audio session for system notification sounds.

Added in version 3.14.

winsound.MB_ICONASTERISK

Reproduce el sonido `SystemDefault`.

winsound.MB_ICONEXCLAMATION

Reproduce el sonido `SystemExclamation`.

winsound.MB_ICONHAND

Reproduce el sonido `SystemHand`.

winsound.MB_ICONQUESTION

Reproduce el sonido `SystemQuestion`.

winsound.MB_OK

Reproduce el sonido `SystemDefault`.

winsound.MB_ICONERROR

Reproduce el sonido `SystemHand`.

Added in version 3.14.

winsound.MB_ICONINFORMATION

Reproduce el sonido `SystemDefault`.

Added in version 3.14.

winsound.MB_ICONSTOP

Reproduce el sonido `SystemHand`.

Added in version 3.14.

winsound.MB_ICONWARNING

Reproduce el sonido `SystemExclamation`.

Added in version 3.14.

Unix-specific services

Los módulos descritos en este capítulo proporcionan interfaces para las características exclusivas del sistema operativo Unix o, en algunos casos, para algunas o muchas variantes del mismo. He aquí una visión general:

- **`shlex`** — Simple lexical analysis
    - objetos **`shlex`**
    - Reglas de análisis
    - Compatibilidad mejorada con intérprete de comandos
- **`posix`** — The most common POSIX system calls
    - Soporte de archivos grandes
    - Contenido notable del módulo
- **`pwd`** — The password database
- **`grp`** — The group database
- **`termios`** — POSIX style tty control
    - Ejemplo
- **`tty`** — Terminal control functions
- **`pty`** — Pseudo-terminal utilities
    - Ejemplo
- **`fcntl`** — The **`fcntl`** and **`ioctl`** system calls
- **`resource`** — Resource usage information
    - Límites de recursos
    - Utilización de recursos
- **`syslog`** — Unix syslog library routines
    - Ejemplos

■ Ejemplo sencillo

`shlex` — Simple lexical analysis

Código fuente: Lib/shlex.py [https://github.com/python/cpython/tree/3.14/Lib/shlex.py]

La clase `shlex` facilita la escritura de analizadores léxicos para sintaxis simples que se parecen al intérprete de comandos de Unix. Esto será a menudo útil para escribir pequeños lenguajes, (por ejemplo, en archivos de control para aplicaciones Python) o para analizar cadenas de texto citadas.

The `shlex` module defines the following functions:

shlex.split(*s*, *comments=False*, *posix=True*)

> Divide la cadena de caracteres *s* usando una sintaxis similar a la de un intérprete de comandos. Si *comments* es `False` (por defecto), el análisis de los comentarios en la cadena de caracteres dada sera deshabilitada (estableciendo el atributo `commenters` de la instancia `shlex` a la cadena de caracteres vacía). Esta función trabaja en modo POSIX por defecto, pero utiliza el modo non-POSIX si el argumento *posix* es falso.
>
> *Distinto en la versión 3.12:* Pasar `None` para el argumento *s* ahora lanza una excepción, en lugar de leer `sys.stdin`.

shlex.join(*split_command*)

> Concatena los tokens de la lista *split_command* y retorna una cadena. Esta función es la inversa de `split()`.

```
>>> from shlex import join
>>> print(join(['echo', '-n', 'Multiple words']))
echo -n 'Multiple words'
```

El valor retornado se escapa del intérprete de comandos para protegerlo contra vulnerabilidades de inyección (consulte `quote()`).

Added in version 3.8.

shlex.quote(*s*)

Retorna una versión con escape del intérprete de comandos de la cadena *s*. El valor retornado es una cadena que se puede usar de forma segura como un token en un intérprete de línea de comandos, para los casos en los que no se puede usar una lista.

Advertencia

El módulo `shlex` **solo está diseñado para los intérpretes de comandos UNIX**.

No se garantiza que la función `quote()` sea correcta en shells no compatibles con POSIX o shells de otros sistemas operativos como Windows. La ejecución de comandos citados por este módulo en tales shells puede abrir la posibilidad de una vulnerabilidad de inyección de comandos.

Considere el uso de funciones que pasen argumentos mediante listas tal como `subprocess.run()` con `shell=False`.

Este idioma sería inseguro:

```
>>> filename = 'somefile; rm -rf ~'
>>> command = 'ls -l {}'.format(filename)
>>> print(command)  # executed by a shell: boom!
ls -l somefile; rm -rf ~
```

`quote()` te permite tapar el agujero de seguridad:

```
>>> from shlex import quote
>>> command = 'ls -l {}'.format(quote(filename))
>>> print(command)
ls -l 'somefile; rm -rf ~'
>>> remote_command = 'ssh home {}'.format(quote(com
>>> print(remote_command)
ssh home 'ls -l '"'"'somefile; rm -rf ~'"'"''
```

La cita es compatible con los intérpretes de comandos UNIX y con **`split()`**:

```
>>> from shlex import split
>>> remote_command = split(remote_command)
>>> remote_command
['ssh', 'home', "ls -l 'somefile; rm -rf ~'"]
>>> command = split(remote_command[-1])
>>> command
['ls', '-l', 'somefile; rm -rf ~']
```

Added in version 3.3.

The **`shlex`** module defines the following class:

class shlex.shlex(*instream*=*None*, *infile*=*None*, *posix*=*False*, *punctuation_chars*=*False*)

Una instancia o instancia de la subclase **`shlex`** es un objeto de analizador léxico. El argumento de inicialización, si está presente, especifica de dónde leer caracteres. Debe ser un objeto similar a un archivo/stream con los métodos **`read()`** y **`readline()`** o una cadena de caracteres. Si no se da ningún argumento, la entrada se tomará de `sys.stdin`. El segundo argumento opcional es una cadena de nombre de archivo, que establece el valor inicial del atributo **`infile`**. Si el argumento *instream* es omitido o es igual a `sys.stdin`, este segundo argumento tiene como valor predeterminado «stdin». El argumento *posix* define el modo operativo: cuando *posix* no es true (predeterminado), la instancia **`shlex`** funcionará en modo de compatibilidad. Cuando se opera en modo POSIX, **`shlex`** intentará estar lo más cerca posible de las reglas de análisis de el intérprete de comandos POSIX. El argumento *punctuation_chars* proporciona una manera de hacer que el comportamiento sea aún más cercano a la forma en que se analizan los intérpretes de comandos reales. Esto puede tomar una serie de valores: el valor predeterminado, `False`, conserva el comportamiento visto en Python 3.5 y versiones anteriores. Si se establece en `True`, se cambia el análisis de los caracteres `();<>|&`: cualquier ejecución de estos caracteres (caracteres considerados de puntuación) se retorna como un único token. Si se establece en una cadena de caracteres no vacía, esos caracteres se utilizarán como caracteres de puntuación. Los caracteres del atributo **`wordchars`** que aparecen en *punctuation_chars* se eliminarán de **`wordchars`**. Consulte Compatibilidad mejorada con intérprete de comandos para obtener más información. *punctuation_chars* solo se puede establecer en la creación de la instancia **`shlex`** y no se puede modificar más adelante.

Distinto en la versión 3.6: Se añadió el parámetro *punctuation_chars*.

Ver también

Módulo **`configparser`**
Analizador de archivos de configuración similares a los archivos `.ini` de Windows.

objetos `shlex`

Una instancia **`shlex`** tiene los siguientes métodos:

shlex.get_token()

Retorna un token. Si los tokens han sido apiladas usando **`push_token()`**, saca una ficha de la pila. De lo contrario, lee uno de la secuencia de entrada. Si la lectura encuentra un fin de archivo inmediato, **`eof`** se retorna (la cadena de caracteres vacía (`''`) en modo no-POSIX, y `None` en modo POSIX).

shlex.push_token(*str*)

Coloca el argumento en la pila de tokens.

shlex.read_token()

Lee un token sin procesar. Ignora la pila de retroceso y no interpreta las peticiones de la fuente. (Este no es normalmente un punto de entrada útil, y está documentado aquí sólo para completarlo.)

shlex.sourcehook(*filename*)

Cuando `shlex` detecta una petición de fuente (ver `source` abajo) este método recibe el siguiente token como argumento, y se espera que retorne una tupla que consista en un nombre de archivo y un archivo abierto como objeto.

Normalmente, este método primero elimina las comillas del argumento. Si el resultado es un nombre de ruta absoluto, o no había ninguna solicitud de origen anterior en vigor, o el origen anterior era una secuencia (como `sys.stdin`), el resultado se deja solo. De lo contrario, si el resultado es un nombre de ruta relativo, la parte del directorio del nombre del archivo se pone inmediatamente antes en la pila de inclusión de origen (este comportamiento es similar a la forma en que el preprocesador de C controla `#include "file.h"`).

El resultado de las manipulaciones se trata como un nombre de archivo y se retorna como el primer componente de la tupla, con `open()` llamado en él para producir el segundo componente. (Nota: esto es lo contrario del orden de los argumentos en la inicialización de instancia!)

Este enlace se expone para que pueda usarlo para implementar rutas de búsqueda de directorios, adición de extensiones de archivo y otros trucos de espacios de nombres. No hay ningún enlace "close" correspondiente, pero una instancia de `shlex` llamará el método `close()` de la secuencia de entrada de origen cuando retorna EOF.

Para un control más explícito del apilamiento de código fuente, utilice los métodos `push_source()` y `pop_source()`.

shlex.push_source(*newstream*, *newfile=None*)

Inserta una secuencia de fuente de entrada en la pila de entrada. Si se especifica el argumento de nombre de archivo, más adelante estará disponible para su uso en mensajes de error. Este es el mismo método utilizado internamente por el método `sourcehook()`.

shlex.pop_source()

Saca la última fuente de entrada de la pila de entrada. Este es el mismo método que el analizador léxico utiliza cuando llega al EOF en una secuencia de entrada apilada.

shlex.error_leader(*infile=None*, *lineno=None*)

Este método genera un mensaje de error líder en el formato de una etiqueta de error del compilador de Unix C; el formato es `'"%s", line %d: '`, donde el `%s` es reemplazado con el nombre del archivo fuente actual y el `%d` con el número de línea de entrada actual (los argumentos opcionales pueden ser usado para sobrescribir estos).

This convenience is provided to encourage **`shlex`** users to generate error messages in the standard, parseable format understood by Emacs and other Unix tools.

Las instancias de las subclases `shlex` tienen algunas variables de instancia pública que controlan el análisis léxico o pueden ser usadas para la depuración:

shlex.commenters

La cadena de caracteres que son reconocidos como comentarios de principiantes. Todos los caracteres desde el comentario principiante hasta el final de la línea son ignorados. Incluye sólo `'#'` por defecto.

shlex.wordchars

La cadena de caracteres que se acumulará en tokens de varios caracteres. Por defecto, incluye todos los alfanuméricos ASCII y el subrayado. En el modo POSIX, los caracteres acentuados del conjunto Latin-1 también están incluidos. Si **`punctuation_chars`** no está vacío, los caracteres `~-./*?=`, que pueden aparecer en las especificaciones del nombre de archivo y en los parámetros de la línea de comandos, también se incluirán en este atributo, y cualquier carácter que aparezca en `punctuation_chars` será eliminado de `wordchars` si están presentes allí. Si **`whitespace_split`** se establece en `True`, esto no tendrá ningún efecto.

shlex.whitespace

Caracteres que serán considerados como espacio en blanco y salteados. El espacio blanco limita los tokens. Por defecto, incluye espacio, tabulación, salto de línea y retorno de carro.

shlex.escape

Caracteres que serán considerados como de escape. Esto sólo se usará en el modo POSIX, e incluye sólo `'\'` por defecto.

shlex.quotes

Los caracteres que serán considerados como citas de la cadena. El token se acumula hasta que la misma cita se encuentra de nuevo (así, los diferentes tipos de citas se protegen entre sí como en el intérprete de comandos.) Por defecto, incluye ASCII comillas simples y dobles.

shlex.escapedquotes

Caracteres en **`quotes`** que interpretarán los caracteres de escape definidos en **`escape`**. Esto sólo se usa en el modo POSIX, e incluye sólo `'"'` por defecto.

shlex.whitespace_split

Si es `True`, los tokens sólo se dividirán en espacios en blanco. Esto es útil, por ejemplo, para analizar las líneas de comando con **`shlex`**, obteniendo los tokens de forma similar a los argumentos de el intérprete de comandos. Cuando se utiliza en combinación con **`puntuation_chars`**, los tokens se dividirán en espacios en blanco además de esos caracteres.

Distinto en la versión 3.8: El atributo **`puntuation_chars`** se hizo compatible con el atributo **`whitespace_split`**.

shlex.infile

El nombre del archivo de entrada actual, como se estableció inicialmente al momento de la instanciación de la clase o apilado por solicitudes de fuente posteriores. Puede ser útil examinar esto cuando se construyan mensajes de error.

shlex.instream

El flujo de entrada del cual esta instancia **`shlex`** está leyendo caracteres.

shlex.source

Este atributo es `None` por defecto. Si le asignas una cadena, esa cadena será reconocida como una solicitud de inclusión a nivel léxico similar a la palabra clave `source` en varios intérpretes de comandos. Es decir, el token inmediatamente siguiente se abrirá como un nombre de archivo y la entrada se tomará de ese flujo hasta EOF, en cuyo momento se llamará al método **`close()`** de ese flujo y la fuente de entrada se convertirá de nuevo en el flujo de entrada original. Las peticiones de origen pueden ser apiladas a cualquier número de niveles de profundidad.

shlex.debug

Si este atributo es numérico y `1` o más, una instancia **`shlex`** imprimirá una salida de progreso verboso en su comportamiento. Si necesitas usar esto, puedes leer el código fuente del módulo para conocer los detalles.

shlex.lineno

Numero de linea de fuente (conteo de las nuevas lineas vistas hasta ahora mas uno).

shlex.token

El buffer de tokens. Puede ser útil examinarlo cuando se capturan excepciones.

shlex.eof

Token usado para determinar el final del archivo. Esto se ajustará a la cadena de caracteres vacía (`''`), en el modo no-POSIX, y a `None` en el modo POSIX.

shlex.punctuation_chars

Una propiedad de sólo lectura. Caracteres que serán considerados como puntuación. Las series de caracteres de puntuación se retornarán como un único token. Sin embargo, tenga en cuenta que no se realizará ninguna comprobación de validez semántica: por ejemplo, "> > >" podría ser retornado como un token, aunque no sea reconocido como tal por los intérpretes de comandos.

Added in version 3.6.

Reglas de análisis

When operating in non-POSIX mode, **`shlex`** will try to obey the following rules.

- Los caracteres entre comillas no son reconocidos dentro de las palabras (`Do"Not"Separate` es analizado como la única palabra `Do"Not"Separate`);
- Los caracteres de escape no son reconocidos;
- El encerrar los caracteres entre comillas preserva el valor literal de todos los caracteres dentro de las comillas;
- Las comillas finales separan las palabras (`"Do"Separate` es analizado como `"Do"` y `Separate`);
- Si **`whitepace_split`** es `False`, cualquier carácter que no sea declarado como un carácter de palabra, espacio en blanco, o una cita será retornado como un token de un solo carácter. Si es `True`, **`shlex`** sólo dividirá las palabras en espacios en blanco;
- EOF es señalado con una cadena de caracteres vacía (`''`);

- No es posible analizar cadenas de caracteres vacías, incluso si se citan.

When operating in POSIX mode, **`shlex`** will try to obey the following parsing rules.

- Las comillas se eliminan y no separan las palabras (`"Do"Not"Separate"` se analiza como la sola palabra `DoNotSeparate`);
- Los caracteres de escape no citados (por ejemplo `'\'`) conservan el valor literal del siguiente carácter que continua;
- Encerrar caracteres entre comillas que no forman parte de **`escapedquotes`** (por ejemplo, `"'"`) conservan el valor literal de todos los caracteres dentro de las comillas;
- Enclosing characters in quotes which are part of **`escapedquotes`** (e.g. `'"'`) preserves the literal value of all characters within the quotes, with the exception of the characters mentioned in **`escape`**. The escape characters retain their special meaning only when followed by the quote in use, or the escape character itself. Otherwise the escape character will be considered a normal character.
- EOF es señalado con un valor **`None`**;
- Se permiten cadenas de caracteres vacías entrecomilladas (`''`).

Compatibilidad mejorada con intérprete de comandos

Added in version 3.6.

La clase **`shlex`** provee compatibilidad con el análisis realizado por los intérprete de comandos comunes de Unix como `bash`, `dash` y `sh`. Para aprovechar esta compatibilidad, especifica el argumento `punctuation_chars` en el constructor. Esto por defecto es `False`, el cual conserva el comportamiento pre-3.6. Sin embargo, si se establece como `True`, entonces se cambia el análisis de los caracteres `();<>|&`: cualquier ejecución de estos caracteres se retorna como un único token. Mientras que esto se queda corto en un analizador completo para los intérprete de comandos (que estaría fuera del alcance de la biblioteca estándar, dada la multiplicidad de intérpretes de comandos que hay), te permite realizar el procesamiento de las líneas de comandos más fácilmente de lo que podrías hacerlo de otra manera. Para ilustrarlo, puede ver la diferencia en el siguiente fragmento:

```
>>> import shlex
>>> text = "a && b; c && d || e; f >'abc'; (def \"ghi\")
>>> s = shlex.shlex(text, posix=True)
>>> s.whitespace_split = True
>>> list(s)
['a', '&&', 'b;', 'c', '&&', 'd', '||', 'e;', 'f', '>abc
>>> s = shlex.shlex(text, posix=True, punctuation_chars=
>>> s.whitespace_split = True
>>> list(s)
['a', '&&', 'b', ';', 'c', '&&', 'd', '||', 'e', ';', 'f
'(', 'def', 'ghi', ')']
```

Por supuesto, se retornarán tokens que no son válidos para los intérpretes de comandos y deberá implementar sus propias comprobaciones de errores en los tokens retornados.

En lugar de pasar `True` como valor para el parámetro punctuation_chars, puede pasar una cadena con caracteres específicos, que se usará para determinar qué caracteres constituyen puntuación. Por ejemplo:

```
>>> import shlex
>>> s = shlex.shlex("a && b || c", punctuation_chars="|"
>>> list(s)
['a', '&', '&', 'b', '||', 'c']
```

Nota

Cuando es especificado `punctuation_chars`, el atributo **`wordchars`** se aumenta con los caracteres `~-./*?=`. Esto se debe a que estos caracteres pueden aparecer en los nombres de archivo (incluidos los comodines) y en los argumentos de la línea de comandos (por ejemplo, `--color=auto`). Por lo tanto:

```
>>> import shlex
>>> s = shlex.shlex('~/a && b-c --color=auto || d *.py?
...                     punctuation_chars=True)
>>> list(s)
['~/a', '&&', 'b-c', '--color=auto', '||', 'd', '*.py?'
```

Sin embargo, para que coincida con el intérprete de comandos lo más cerca posible, se recomienda utilizar siempre `posix` y **`whitespace_split`** cuando se utiliza **`punctuation_chars`**, el cual negará por completo **`wordchars`**.

Para obtener el mejor efecto, `punctuation_chars` debe establecerse junto con `posix=True`. (Tenga en cuenta que `posix=False` es el valor predeterminado para **`shlex`**.)

`posix` — The most common POSIX system calls

Este módulo proporciona acceso a la funcionalidad del sistema operativo que está estandarizada por el estándar C y el estándar POSIX (una interfaz Unix finamente disfrazada).

Availability: Unix.

Do not import this module directly. Instead, import the module `os`, which provides a *portable* version of this interface. On Unix, the `os` module provides a superset of the `posix` interface. On non-Unix operating systems the `posix` module is not available, but a subset is always available through the `os` interface. Once `os` is imported, there is *no* performance penalty in using it instead of `posix`. In addition, `os` provides some additional functionality, such as automatically calling `putenv()` when an entry in `os.environ` is changed.

Los errores se notifican como excepciones; las excepciones habituales se proporcionan para los errores de tipo, mientras que los errores notificados por las llamadas del sistema lanzan `OSError`.

Soporte de archivos grandes

Several operating systems (including AIX and Solaris) provide support for files that are larger than 2 GiB from a C programming model where int and long are 32-bit values. This is typically accomplished by defining the relevant size and offset types as 64-bit values. Such files are sometimes referred to as *large files*.

Large file support is enabled in Python when the size of an `off_t` is larger than a long and the long long is at least as large as an

`off_t`. It may be necessary to configure and compile Python with certain compiler flags to enable this mode. For example, with Solaris 2.6 and 2.7 you need to do something like:

```
CFLAGS="`getconf LFS_CFLAGS`" OPT="-g -O2 $CFLAGS" \
        ./configure
```

En sistemas Linux con capacidad para archivos grandes, esto podría funcionar:

```
CFLAGS='-D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64' OP
        ./configure
```

Contenido notable del módulo

In addition to many functions described in the `os` module documentation, **`posix`** defines the following data item:

posix.environ

Diccionario que representa el entorno de cadena en el momento en que se inició el intérprete. Las claves y los valores son bytes en Unix y str en Windows. Por ejemplo, `environ[b'HOME']` (`environ['HOME']` en Windows) es el nombre de ruta de acceso de su directorio principal, equivalente a `getenv("HOME")` en C.

La modificación de este diccionario no afecta al entorno de cadena que transmite `execv()`, `popen()` o `system()`; si necesita cambiar el entorno, pase `environ` a `execve()` o agregue asignaciones variables y declaraciones de exportación a la cadena de comandos para **`system()`** o **`popen()`**.

Distinto en la versión 3.2: En Unix, las claves y los valores son bytes.

Nota

The `os` module provides an alternate implementation of `environ` which updates the environment on modification. Note also that updating `os.environ` will render this

dictionary obsolete. Use of the **`os`** module version of this is recommended over direct access to the **`posix`** module.

`pwd` — The password database

Este módulo proporciona acceso a la base de datos de cuentas de usuario y contraseñas de Unix. Está disponible en todas las versiones de Unix.

Availability: Unix, not WASI, not iOS.

Las entradas de la base de datos de contraseñas se reportan como un objeto de tipo tupla, cuyos atributos corresponden a los miembros de la estructura `passwd` (campo Atributo abajo, ver `<pwd.h>`):

Índice	**Atributo**	**Significado**
0	`pw_name`	Nombre de usuario
1	`pw_passwd`	Contraseña encriptada opcional
2	`pw_uid`	Identificación numérica de usuario
3	`pw_gid`	Identificación del grupo numérico
4	`pw_gecos`	Nombre de usuario o campo de comentarios
5	`pw_dir`	El directorio *home* del usuario
6	`pw_shell`	Intérprete de comandos de usuario

Los elementos uid y gid son enteros, todos los demás son cadenas. **`KeyError`** se lanza si la entrada pedida no se encuentra.

Nota

In traditional Unix the field `pw_passwd` usually contains a password encrypted with a DES derived algorithm. However most modern unices use a so-called *shadow password* system. On those unices the *pw_passwd* field only contains an asterisk (`'*'`) or the letter `'x'` where the encrypted password is stored in a file `/etc/shadow` which is not world readable. Whether the *pw_passwd* field contains anything useful is system-dependent.

Define los siguientes elementos:

pwd.getpwuid(*uid*)

Retorna la entrada de la base de datos de contraseñas para el ID de usuario numérico dado.

pwd.getpwnam(*name*)

Retorna la entrada de la base de datos de contraseñas para el nombre de usuario dado.

pwd.getpwall()

Retorna una lista de todas las entradas de la base de datos de contraseñas disponibles, en orden arbitrario.

Ver también

Módulo `grp`

Una interfaz para la base de datos de grupos, similar a esta.

`grp` — The group database

Este módulo proporciona acceso a la base de datos del grupo Unix. Está disponible en todas las versiones de Unix.

Availability: Unix, not WASI, not Android, not iOS.

Las entradas de base de datos de grupo se notifican como un objeto similar a una tupla, cuyos atributos corresponden a los miembros de la estructura `group` (campo atributo a continuación, véase `<grp.h>`):

Índice	Atributo	Significado
0	gr_name	el nombre del grupo
1	gr_passwd	la contraseña (encriptada) del grupo; usualmente vacío
2	gr_gid	el grupo ID numérico
3	gr_mem	todos los nombres de usuario de los miembros del grupo

El *gid* es un número entero, el nombre y la contraseña son cadenas y la lista de miembros es una lista de cadenas. (Tenga en cuenta que la mayoría de los usuarios no se enumeran explícitamente como miembros del grupo en el que se encuentran de acuerdo con la base de datos de contraseñas. Consulte ambas bases de datos para obtener información completa sobre la membresía. También tenga en cuenta que un ''gr_name'' que comienza con un `+` o `-` es probable que sea una referencia de YP/NIS y puede que no sea accesible vía **`getgrnam()`** o **`getgrgid()`**.)

Define los siguientes elementos:

grp.getgrgid(*id*)

Retorna la entrada de la base de datos del grupo para el ID de grupo numérico dado. Se genera **`KeyError`** si no se puede encontrar la entrada solicitada.

Distinto en la versión 3.10: **`TypeError`** se genera para argumentos no enteros como flotantes o cadenas de caracteres.

grp.getgrnam(*name*)

Retorna la entrada de la base de datos del grupo para el nombre de grupo dado. Se genera `KeyError` si no se puede encontrar la entrada solicitada.

grp.getgrall()

Retorna una lista de todas las entradas de grupo disponibles, en orden arbitrario.

Ver también

Módulo `pwd`
Una interfaz para la base de datos de usuarios, similar a esta.

`termios` — POSIX style tty control

Este módulo proporciona una interfaz para las llamadas POSIX para el control de E/S (entrada/salida) tty. Para obtener una descripción completa de estas llamadas, consulte *termios (3)* *[https://manpages.debian.org/termios (3)]* Página de manual de Unix. Solo está disponible para aquellas versiones de Unix que admitan el control de E/S tty estilo POSIX *termios* configurado durante la instalación.

Availability: Unix.

Todas las funciones de este módulo toman un descriptor de archivo *fd* como primer argumento. Puede ser un descriptor de archivo entero, como el que retorna `sys.stdin.fileno()`, o un file object, como el propio `sys.stdin`.

Este módulo también define todas las constantes necesarias para trabajar con las funciones proporcionadas aquí; tienen el mismo nombre que sus contrapartes en C. Consulte la documentación de su sistema para obtener más información sobre el uso de estas interfaces de control de terminal.

El módulo define las siguientes funciones:

termios.tcgetattr(*fd*)

Return a list containing the tty attributes for file descriptor *fd*, as follows: `[iflag, oflag, cflag, lflag, ispeed, ospeed, cc]` where *cc* is a list of the tty special characters (each a string of length 1, except the items with indices **`VMIN`** and **`VTIME`**, which are integers when these fields are defined). The interpretation of the flags and the speeds as well as the indexing in the *cc* array must be done using the symbolic constants defined in the **`termios`** module.

termios.tcsetattr(*fd*, *when*, *attributes*)

Set the tty attributes for file descriptor *fd* from the *attributes*, which is a list like the one returned by `tcgetattr()`. The *when* argument determines when the attributes are changed:

termios.TCSANOW

Change attributes immediately.

termios.TCSADRAIN

Change attributes after transmitting all queued output.

termios.TCSAFLUSH

Change attributes after transmitting all queued output and discarding all queued input.

termios.tcsendbreak(*fd*, *duration*)

Envíe una pausa en el descriptor de archivo *fd*. Una *duración* cero envía una pausa de 0.25 a 0.5 segundos; una *duración* distinta de cero tiene un significado dependiente del sistema.

termios.tcdrain(*fd*)

Espere hasta que se haya transmitido toda la salida escrita en el descriptor de archivo *fd*.

termios.tcflush(*fd*, *queue*)

Descartar datos en cola en el descriptor de archivo *fd*. El selector *queue* especifica qué cola: `TCIFLUSH` para la cola de entrada, `TCOFLUSH` para la cola de salida, o `TCIOFLUSH` para ambas colas.

termios.tcflow(*fd*, *action*)

Suspender o reanudar la entrada o salida en el descriptor de archivo *fd*. El argumento *action* puede ser `TCOOFF` para suspender la salida, `TCOON` para reiniciar la salida, `TCIOFF` para suspender la entrada, o `TCION` para reiniciar la entrada.

termios.tcgetwinsize(*fd*)

Devuelve una tupla `(ws_row, ws_col)` que contiene el tamaño de la ventana tty para el descriptor de archivo *fd*. Requiere **`termios.TIOCGWINSZ`** o **`termios.TIOCGSIZE`**.

Added in version 3.11.

termios.tcsetwinsize(*fd*, *winsize*)

Establezca el tamaño de la ventana tty para el descriptor de archivo *fd* de *winsize*, que es un tupla de dos elementos `(ws_row, ws_col)` como la devuelta por **`tcgetwinsize()`**. Requiere que al menos uno de los pares (**`termios.TIOCGWINSZ`**, **`termios.TIOCSWINSZ`**); (**`termios.TIOCGSIZE`**, **`termios.TIOCSSIZE`**) por definir.

Added in version 3.11.

Ver también

Módulo `tty`

Funciones de conveniencia para operaciones comunes de control de terminal.

Ejemplo

Aquí hay una función que solicita una contraseña con el eco desactivado. Tenga en cuenta la técnica utilizando una llamada separada **`tcgetattr()`** y una sentencia **`try`** … **`finally`** para asegurarse de que los antiguos atributos tty se restauran exactamente sin importar lo que suceda:

```
def getpass(prompt="Password: "):
    import termios, sys
    fd = sys.stdin.fileno()
    old = termios.tcgetattr(fd)
    new = termios.tcgetattr(fd)
    new[3] = new[3] & ~termios.ECHO          # lflags
    try:
        termios.tcsetattr(fd, termios.TCSADRAIN, new)
        passwd = input(prompt)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old)
    return passwd
```

`tty` — Terminal control functions

Código fuente: Lib/tty.py [https://github.com/python/cpython/tree/3.14/Lib/tty.py]

The **`tty`** module defines functions for putting the tty into cbreak and raw modes.

Availability: Unix.

Dado que requiere el módulo **`termios`**, solamente funciona en Unix.

The **`tty`** module defines the following functions:

tty.cfmakeraw(*mode*)

Convert the tty attribute list *mode*, which is a list like the one returned by **`termios.tcgetattr()`**, to that of a tty in raw mode.

Added in version 3.12.

tty.cfmakecbreak(*mode*)

Convert the tty attribute list *mode*, which is a list like the one returned by **`termios.tcgetattr()`**, to that of a tty in cbreak mode.

This clears the `ECHO` and `ICANON` local mode flags in *mode* as well as setting the minimum input to 1 byte with no delay.

Added in version 3.12.

Distinto en la versión 3.12.2: The `ICRNL` flag is no longer cleared. This matches Linux and macOS `stty cbreak`

behavior and what `setcbreak()` historically did.

tty.setraw(*fd*, *when=termios.TCSAFLUSH*)

Change the mode of the file descriptor *fd* to raw. If *when* is omitted, it defaults to `termios.TCSAFLUSH`, and is passed to `termios.tcsetattr()`. The return value of `termios.tcgetattr()` is saved before setting *fd* to raw mode; this value is returned.

Distinto en la versión 3.12: The return value is now the original tty attributes, instead of `None`.

tty.setcbreak(*fd*, *when=termios.TCSAFLUSH*)

Change the mode of file descriptor *fd* to cbreak. If *when* is omitted, it defaults to `termios.TCSAFLUSH`, and is passed to `termios.tcsetattr()`. The return value of `termios.tcgetattr()` is saved before setting *fd* to cbreak mode; this value is returned.

This clears the `ECHO` and `ICANON` local mode flags as well as setting the minimum input to 1 byte with no delay.

Distinto en la versión 3.12: The return value is now the original tty attributes, instead of `None`.

Distinto en la versión 3.12.2: The `ICRNL` flag is no longer cleared. This restores the behavior of Python 3.11 and earlier as well as matching what Linux, macOS, & BSDs describe in their `stty(1)` man pages regarding cbreak mode.

Ver también

Módulo `termios`
Interfaz de control de la terminal de bajo nivel.

`pty` — Pseudo-terminal utilities

Código Fuente: Lib/pty.py [https://github.com/python/cpython/tree/3.14/Lib/pty.py]

The `pty` module defines operations for handling the pseudo-terminal concept: starting another process and being able to write to and read from its controlling terminal programmatically.

Availability: Unix.

El manejo de pseudo-terminales depende en gran medida de la plataforma. Este código se prueba principalmente en Linux, FreeBSD y macOS (se supone que funciona en otras plataformas POSIX, pero no se ha probado a fondo).

The `pty` module defines the following functions:

pty.fork()

Bifurcación. Conectar en su propia terminal (terminal hijo) una pseudo-terminal. El valor de retorno es `(pid, fd)`. Tener en cuenta que la terminal hijo tiene como valor *pid* 0 y *fd* es *invalid*. El valor de retorno del padre es el *pid* del hijo, y *fd* es un descriptor de archivo conectado a la terminal hijo (también a la salida y entrada estándar de la terminal hijo)

Advertencia

On macOS the use of this function is unsafe when mixed with using higher-level system APIs, and that includes using `urllib.request`.

pty.openpty()

Abre un nuevo par de pseudo-terminales, usando

`os.openpty()`, o código de emulación para sistemas genéricos de Unix. Retorna un par de descriptores de archivo `(master, slave)`, para el *master* y el *slave* respectivamente.

pty.spawn(*argv*[, *master_read*[, *stdin_read*]])

Genera un proceso conectado a su terminal con el io estándar del proceso actual. Esto se usa a frecuentemente para confundir programas que insisten en leer desde la terminal de control. Se espera que el proceso generado detrás de pty sea finalizado y cuando lo haga *spawn* se retornará.

Un bucle copia STDIN del proceso actual al hijo y los datos recibidos del niño a STDOUT del proceso actual. No se le indica al hijo si STDIN del proceso actual se cierra.

A las funciones *master_read* y *stdin_read* se les pasa un descriptor de archivo del cual deben leer, y siempre deben retornar una cadena de bytes. Para forzar el retorno de spawn antes de que el proceso hijo salga, se debe retornar un arreglo de bytes vacía para señalar el final del archivo.

La implementación predeterminada para ambas funciones retornará hasta 1024 bytes cada vez que se llamen. El dato retornado de *master_read* se pasa al descriptor de archivo maestro para leer la salida del proceso hijo, y *stdin_read* pasa el descriptor de archivo 0, para leer desde la entrada del proceso padre.

Retornando una cadena de bytes vacía de cualquier llamado es interpretado como una condición de fin de archivo (EOF), y el llamado no se realizará después de eso. Si *stdin_read* retorna EOF la terminal de control ya no puede comunicarse con el proceso padre o el proceso hijo. A menos que el proceso hijo se cierre sin ninguna entrada *spawn* se repetirá para siempre. Si *master_read* retorna EOF se produce el mismo comportamiento (al menos en Linux)

Retorna el valor del estado de salida de **`os.waitpid()`** en el proceso hijo.

`os.waitstatus_to_exitcode()` se puede utilizar para convertir el estado de salida en un código de salida.

Lanza un evento de auditoria `pty.spawn` con el argumento `argv`.

Distinto en la versión 3.4: **`spawn()`** ahora retorna el valor de estado de **`os.waitpid()`** para los procesos hijos.

Ejemplo

El siguiente programa actúa como el comando de Unix *script(1)* *[https://manpages.debian.org/script(1)]*, usando una pseudo-terminal para registrar todas las entradas y salidas de una sesión en «typescript».

```
import argparse
import os
import pty
import sys
import time

parser = argparse.ArgumentParser()
parser.add_argument('-a', dest='append', action='store_t
parser.add_argument('-p', dest='use_python', action='sto
parser.add_argument('filename', nargs='?', default='type
options = parser.parse_args()

shell = sys.executable if options.use_python else os.env
filename = options.filename
mode = 'ab' if options.append else 'wb'

with open(filename, mode) as script:
    def read(fd):
        data = os.read(fd, 1024)
        script.write(data)
        return data

    print('Script started, file is', filename)
    script.write(('Script started on %s\n' % time.asctim
pty.spawn(shell, read)

script.write(('Script done on %s\n' % time.asctime()
print('Script done, file is', filename)
```

`fcntl` — The `fcntl` and `ioctl` system calls

This module performs file and I/O control on file descriptors. It is an interface to the **`fcntl()`** and **`ioctl()`** Unix routines. See the *fcntl(2)* *[https://manpages.debian.org/fcntl(2)]* and *ioctl(2)* *[https://manpages.debian.org/ioctl(2)]* Unix manual pages for full details.

Availability: Unix, not WASI.

Todas las funciones de este módulo toman un descriptor de fichero *fd* como su primer argumento. Puede ser un descriptor de fichero entero, como el retornado por `sys.stdin.fileno()`, o un objeto **`io.IOBase`**, como `sys.stdin`, que proporciona un **`fileno()`** que retorna un descriptor de fichero original.

Distinto en la versión 3.3: Las operaciones en este módulo solían lanzar un **`IOError`** donde ahora lanzan un **`OSError`**.

Distinto en la versión 3.8: The **`fcntl`** module now contains `F_ADD_SEALS`, `F_GET_SEALS`, and `F_SEAL_*` constants for sealing of **`os.memfd_create()`** file descriptors.

Distinto en la versión 3.9: On macOS, the **`fcntl`** module exposes the `F_GETPATH` constant, which obtains the path of a file from a file descriptor. On Linux($>=$3.15), the **`fcntl`** module exposes the `F_OFD_GETLK`, `F_OFD_SETLK` and `F_OFD_SETLKW` constants, which are used when working with open file description locks.

Distinto en la versión 3.10: On Linux $>=$ 2.6.11, the **`fcntl`** module exposes the `F_GETPIPE_SZ` and `F_SETPIPE_SZ` constants, which allow to check and modify a pipe's size respectively.

Distinto en la versión 3.11: On FreeBSD, the **`fcntl`** module exposes the `F_DUP2FD` and `F_DUP2FD_CLOEXEC` constants, which allow to duplicate a file descriptor, the latter setting `FD_CLOEXEC` flag in addition.

Distinto en la versión 3.12: On Linux > = 4.5, the **fcntl** module exposes the `FICLONE` and `FICLONERANGE` constants, which allow to share some data of one file with another file by reflinking on some filesystems (e.g., btrfs, OCFS2, and XFS). This behavior is commonly referred to as «copy-on-write».

Distinto en la versión 3.13: On Linux > = 2.6.32, the **fcntl** module exposes the `F_GETOWN_EX`, `F_SETOWN_EX`, `F_OWNER_TID`, `F_OWNER_PID`, `F_OWNER_PGRP` constants, which allow to direct I/O availability signals to a specific thread, process, or process group. On Linux > = 4.13, the **fcntl** module exposes the `F_GET_RW_HINT`, `F_SET_RW_HINT`, `F_GET_FILE_RW_HINT`, `F_SET_FILE_RW_HINT`, and `RWH_WRITE_LIFE_*` constants, which allow to inform the kernel about the relative expected lifetime of writes on a given inode or via a particular open file description. On Linux > = 5.1 and NetBSD, the **fcntl** module exposes the `F_SEAL_FUTURE_WRITE` constant for use with `F_ADD_SEALS` and `F_GET_SEALS` operations. On FreeBSD, the **fcntl** module exposes the `F_READAHEAD`, `F_ISUNIONSTACK`, and `F_KINFO` constants. On macOS and FreeBSD, the **fcntl** module exposes the `F_RDAHEAD` constant. On NetBSD and AIX, the **fcntl** module exposes the `F_CLOSEM` constant. On NetBSD, the **fcntl** module exposes the `F_MAXFD` constant. On macOS and NetBSD, the **fcntl** module exposes the `F_GETNOSIGPIPE` and `F_SETNOSIGPIPE` constant.

Distinto en la versión 3.14: On Linux > = 6.1, the **fcntl** module exposes the `F_DUPFD_QUERY` to query a file descriptor pointing to the same file.

El módulo define las siguientes funciones:

fcntl.fcntl(*fd*, *cmd*, *arg=0*, */*)

Perform the operation *cmd* on file descriptor *fd* (file objects providing a **`fileno()`** method are accepted as well). The values used for *cmd* are operating system dependent, and are available as constants in the **fcntl** module, using the same

names as used in the relevant C header files. The argument *arg* can either be an integer value, a bytes-like object, or a string. The type and size of *arg* must match the type and size of the argument of the operation as specified in the relevant C documentation.

When *arg* is an integer, the function returns the integer return value of the C `fcntl()` call.

When the argument is bytes-like object, it represents a binary structure, for example, created by `struct.pack()`. A string value is encoded to binary using the UTF-8 encoding. The binary data is copied to a buffer whose address is passed to the C `fcntl()` call. The return value after a successful call is the contents of the buffer, converted to a `bytes` object. The length of the returned object will be the same as the length of the *arg* argument. This is limited to 1024 bytes.

If the `fcntl()` call fails, an `OSError` is raised.

Nota

If the type or the size of *arg* does not match the type or size of the argument of the operation (for example, if an integer is passed when a pointer is expected, or the information returned in the buffer by the operating system is larger than 1024 bytes), this is most likely to result in a segmentation violation or a more subtle data corruption.

Lanza un evento de auditoria `fcntl.fcntl` con argumentos `fd`, `cmd`, `arg`.

Distinto en la versión 3.14: Add support of arbitrary bytes-like objects, not only `bytes`.

fcntl.ioctl(*fd*, *request*, *arg=0*, *mutate_flag=True*, */*)

Esta función es idéntica a la función `fcntl()`, excepto por el manejo de los argumentos que es aún más complicado.

The *request* parameter is limited to values that can fit in 32-

bits or 64-bits, depending on the platform. Additional constants of interest for use as the *request* argument can be found in the `termios` module, under the same names as used in the relevant C header files.

The parameter *arg* can be an integer, a bytes-like object, or a string. The type and size of *arg* must match the type and size of the argument of the operation as specified in the relevant C documentation.

If *arg* does not support the read-write buffer interface or the *mutate_flag* is false, behavior is as for the `fcntl()` function.

If *arg* supports the read-write buffer interface (like `bytearray`) and *mutate_flag* is true (the default), then the buffer is (in effect) passed to the underlying `ioctl()` system call, the latter's return code is passed back to the calling Python, and the buffer's new contents reflect the action of the `ioctl()`. This is a slight simplification, because if the supplied buffer is less than 1024 bytes long it is first copied into a static buffer 1024 bytes long which is then passed to `ioctl()` and copied back into the supplied buffer.

If the `ioctl()` call fails, an `OSError` exception is raised.

Nota

If the type or size of *arg* does not match the type or size of the operation's argument (for example, if an integer is passed when a pointer is expected, or the information returned in the buffer by the operating system is larger than 1024 bytes, or the size of the mutable bytes-like object is too small), this is most likely to result in a segmentation violation or a more subtle data corruption.

Un ejemplo:

```
>>> import array, fcntl, struct, termios, os
>>> os.getpgrp()
13341
>>> struct.unpack('h', fcntl.ioctl(0, termios.TIOCG
13341
>>> buf = array.array('h', [0])
>>> fcntl.ioctl(0, termios.TIOCGPGRP, buf, 1)
0
>>> buf
array('h', [13341])
```

Lanza un evento de auditoria `fcntl.ioctl` con argumentos `fd`, `request`, `arg`.

Distinto en la versión 3.14: The GIL is always released during a system call. System calls failing with EINTR are automatically retried.

fcntl.flock(*fd*, *operation*, */*)

Realiza la operación de bloqueo *operation* sobre el descriptor de fichero *fd* (también se aceptan objetos de fichero que proporcionen un método **`fileno()`**). Ver el manual de Unix *flock(2)* *[https://manpages.debian.org/flock(2)]* para más detalles. (En algunos sistemas, esta función es emulada usando **`fcntl()`**.)

If the **`flock()`** call fails, an **`OSError`** exception is raised.

Lanza un evento de auditoria `fcntl.flock` con argumentos `fd`, `operation`.

fcntl.lockf(*fd*, *cmd*, *len=0*, *start=0*, *whence=0*, */*)

Esto es esencialmente un «wrapper» de las llamadas de bloqueo **`fcntl()`**. *fd* es el descriptor de fichero (también se aceptan objetos de fichero que proporcionen un método **`fileno()`**) del archivo para bloquear o desbloquear, y *cmd* es uno de los siguientes valores:

fcntl.LOCK_UN

Release an existing lock.

fcntl.LOCK_SH

Acquire a shared lock.

fcntl.LOCK_EX

Acquire an exclusive lock.

fcntl.LOCK_NB

Bitwise OR with any of the other three `LOCK_*` constants to make the request non-blocking.

If **`LOCK_NB`** is used and the lock cannot be acquired, an `OSError` will be raised and the exception will have an *errno* attribute set to `EACCES` or `EAGAIN` (depending on the operating system; for portability, check for both values). On at least some systems, **`LOCK_EX`** can only be used if the file descriptor refers to a file opened for writing.

len es el número de bytes a bloquear, *start* es el desplazamiento de bytes en el que comienza el bloqueo, relativo a *whence*, y *whence* es como con `io.IOBase.seek()`, específicamente:

- `0` – relativo al inicio del archivo (`os.SEEK_SET`)
- `1` – relativo a la posición actual del buffer (`os.SEEK_CUR`)
- `2` – relativo al final del archivo (`os.SEEK_END`)

El valor por defecto de*start* es 0, lo que significa que comienza al inicio del archivo. El valor por defecto para *len* es 0 lo que significa bloquear hasta el final del archivo. El valor por defecto para *whence* también es 0.

Lanza un evento de auditoria `fcntl.lockf` con argumentos `fd`, `cmd`, `len`, `start`, `whence`.

Ejemplos (todos en un sistema compatible con SVR4):

```
import struct, fcntl, os

f = open(...)
rv = fcntl.fcntl(f, fcntl.F_SETFL, os.O_NDELAY)

lockdata = struct.pack('hhllhh', fcntl.F_WRLCK, 0, 0, 0,
rv = fcntl.fcntl(f, fcntl.F_SETLKW, lockdata)
```

Tenga en cuenta que en el primer ejemplo, el valor de la variable retornada *rv* contendrá un valor entero; en el segundo ejemplo contendrá un objeto `bytes`. La estructura para la variable *lockdata* depende del sistema — por lo tanto, usar la llamada `flock()` puede ser mejor.

Ver también

Módulo `os`
Si los indicadores de bloqueo `O_SHLOCK` y `O_EXLOCK` están presentes en el módulo `os` (sólo en BSD), la función `os.open()` proporciona una alternativa a las funciones `lockf()` y `flock()`.

`resource` — Resource usage information

Este módulo proporciona mecanismos básicos para medir y controlar los recursos del sistema utilizados por un programa.

Availability: Unix, not WASI.

Las constantes simbólicas se utilizan para especificar recursos concretos del sistema y para solicitar información de uso sobre el proceso actual o sus elementos secundarios.

Se genera un `OSError` cuando la llamada al sistema (*syscall*) falla.

exception resource.error

Un alias en desuso de `OSError`.

Distinto en la versión 3.3: Tras **PEP 3151** [https://peps.python.org/pep-3151/] esta clase se convirtió en un alias de `OSError`.

Límites de recursos

El uso de recursos se puede limitar usando la función `setrlimit()` que se describe a continuación. Cada recurso está controlado por un par de límites: un límite flexible y un límite duro. El límite flexible es el límite actual, y puede ser reducido o elevado con el tiempo mediante un proceso. El límite flexible nunca puede exceder el límite duro. El límite duro se puede reducir a cualquier valor mayor que el del límite flexible, pero no se puede elevar. (Solo los procesos con el UID efectivo del superusuario pueden aumentar un límite duro.)

Los recursos específicos que se pueden limitar dependen del sistema. Se describen en la página de manual *getrlimit(2)* *[https://*

manpages.debian.org/getrlimit(2)]. Los recursos enumerados a continuación se admiten cuando el sistema operativo subyacente los admite; los recursos que no pueden ser verificados o controlados por el sistema operativo no se definen en este módulo para esas plataformas.

resource.RLIM_INFINITY

Constante utilizada para representar el límite de un recurso ilimitado.

resource.getrlimit(*resource*)

Retorna una tupla `(soft, hard)` con los límites flexible y duro actuales de *resource*. Genera **`ValueError`** si se especifica un recurso no válido o **`error`** si la llamada al sistema subyacente falla inesperadamente.

resource.setrlimit(*resource*, *limits*)

Sets new limits of consumption of *resource*. The *limits* argument must be a tuple `(soft, hard)` of two integers describing the new limits. A value of **`RLIM_INFINITY`** can be used to request a limit that is unlimited.

Raises **`ValueError`** if an invalid resource is specified, if the new soft limit exceeds the hard limit, or if a process tries to raise its hard limit. Specifying a limit of **`RLIM_INFINITY`** when the hard or system limit for that resource is not unlimited will result in a **`ValueError`**. A process with the effective UID of super-user can request any valid limit value, including unlimited, but **`ValueError`** will still be raised if the requested limit exceeds the system imposed limit.

`setrlimit` también puede generar un **`error`** si falla la llamada al sistema subyacente.

VxWorks only supports setting **`RLIMIT_NOFILE`**.

Genera un auditing event `resource.setrlimit` con los argumentos `resource`, `limits`.

resource.prlimit(*pid*, *resource*[, *limits*])

Combina **`setrlimit()`** y **`getrlimit()`** en una sola función y admite obtener y establecer los límites de recursos de un proceso arbitrario. Si *pid* es 0, entonces la llamada se aplica al proceso actual. *resource* y *limits* tienen el mismo significado que en **`setrlimit()`**, excepto por que *limits* es opcional.

Cuando no se proporciona *limits* la función retorna el límite de *resource* del proceso *pid*. Cuando se proporciona *limits*, se establece el límite de *resource* del proceso y se retorna el límite de recursos anterior.

Genera **`ProcessLookupError`** cuando no se encuentra *pid* y **`PermissionError`** cuando el usuario no tiene `CAP_SYS_RESOURCE` para el proceso.

Genera un evento de auditoría `resource.prlimit` con los argumentos `pid`, `resource`, `limits`.

Availability: Linux >= 2.6.36 with glibc >= 2.13.

Added in version 3.4.

Estos símbolos definen los recursos cuyo consumo se puede controlar usando las funciones **`setrlimit()`** y **`getrlimit()`** que se describen más abajo. Los valores de estos símbolos son exactamente las constantes utilizadas por programas en C.

La página de manual de Unix para *getrlimit(2)* *[https://manpages.debian.org/getrlimit(2)]* detalla los recursos disponibles. Tenga en cuenta que no todos los sistemas usan el mismo símbolo o el mismo valor para referirse al mismo recurso. Este módulo no pretende enmascarar las diferencias entre plataformas — los símbolos no definidos para una plataforma no estarán disponibles en este módulo en esa plataforma.

resource.RLIMIT_CORE

El tamaño máximo (en bytes) de un archivo central que puede crear el proceso actual. Esto podría resultar en la creación de un archivo central parcial si se requiriera uno más grande para contener la imagen del proceso entera.

resource.RLIMIT_CPU

The maximum amount of processor time (in seconds) that a process can use. If this limit is exceeded, a `SIGXCPU` signal is sent to the process. (See the `signal` module documentation for information about how to catch this signal and do something useful, e.g. flush open files to disk.)

resource.RLIMIT_FSIZE

El tamaño máximo de un archivo que pueda crear el proceso.

resource.RLIMIT_DATA

El tamaño máximo (en bytes) de la memoria *heap* del proceso.

resource.RLIMIT_STACK

El tamaño máximo (en bytes) de la pila de llamadas para el proceso actual. Esto afecta únicamente a la pila del hilo principal en un proceso multi-hilo.

resource.RLIMIT_RSS

El tamaño máximo del conjunto residente (*RSS*) del que puede disponer el proceso.

resource.RLIMIT_NPROC

El número máximo de procesos que puede crear el proceso actual.

resource.RLIMIT_NOFILE

El número máximo de descriptores de archivo abierto para el proceso actual.

resource.RLIMIT_OFILE

El nombre BDS para `RLIMIT_NOFILE`.

resource.RLIMIT_MEMLOCK

El espacio de direcciones máximo que se puede bloquear en la memoria.

resource.RLIMIT_VMEM

The largest area of mapped memory which the process may occupy. Usually an alias of **`RLIMIT_AS`**.

Availability: Solaris, FreeBSD, NetBSD.

resource.RLIMIT_AS

El área máxima (en bytes) de espacio de direcciones que puede tomar el proceso.

resource.RLIMIT_MSGQUEUE

El número de bytes que se pueden asignar a las colas de mensajes POSIX.

Availability: Linux $>=$ 2.6.8.

Added in version 3.4.

resource.RLIMIT_NICE

El techo del nivel del proceso *nice* (calculado como 20 - rlim_cur).

Availability: Linux $>=$ 2.6.12.

Added in version 3.4.

resource.RLIMIT_RTPRIO

El techo de la prioridad en tiempo real.

Availability: Linux $>=$ 2.6.12.

Added in version 3.4.

resource.RLIMIT_RTTIME

El límite de tiempo (en microsegundos) en tiempo de CPU que puede dedicar un proceso de programación en tiempo real sin hacer una llamada al sistema de bloqueo.

Availability: Linux $>=$ 2.6.25.

Added in version 3.4.

resource.RLIMIT_SIGPENDING

El número de señales que el proceso puede poner en cola.

Availability: Linux > = 2.6.8.

Added in version 3.4.

resource.RLIMIT_SBSIZE

El tamaño máximo (en bytes) de uso del búfer del socket para este usuario. Esto limita la cantidad de memoria de red, y por lo tanto la cantidad de mbufs, que este usuario puede retener en todo momento.

Availability: FreeBSD, NetBSD.

Added in version 3.4.

resource.RLIMIT_SWAP

El tamaño máximo (en bytes) del espacio de intercambio que puede ser reservado o utilizado por todos los procesos de este ID de usuario. Este límite se aplica solo si se establece el bit 1 de vm.overcommit sysctl. Consulte tuning(7) [https://www.freebsd.org/cgi/man.cgi?query=tuning&sektion=7] para obtener una descripción completa de este sysctl.

Availability: FreeBSD > = 8.

Added in version 3.4.

resource.RLIMIT_NPTS

El número máximo de pseudo-terminales que puede crear esta ID de usuario.

Availability: FreeBSD > = 8.

Added in version 3.4.

resource.RLIMIT_KQUEUES

El número máximo de kqueues que este ID de usuario puede crear.

Availability: FreeBSD >= 11.

Added in version 3.10.

Utilización de recursos

Estas funciones se usan para recuperar la información de utilización de recursos:

resource.getrusage(*who*)

This function returns an object that describes the resources consumed by either the current process or its children, as specified by the *who* parameter. The *who* parameter should be specified using one of the **`RUSAGE_*`** constants described below.

Un ejemplo sencillo:

```
from resource import *
import time

# a non CPU-bound task
time.sleep(3)
print(getrusage(RUSAGE_SELF))

# a CPU-bound task
for i in range(10 ** 8):
   _ = 1 + 1
print(getrusage(RUSAGE_SELF))
```

The fields of the return value each describe how a particular system resource has been used, e.g. amount of time spent running in user mode or number of times the process was swapped out of main memory. Some values are dependent on the clock tick interval, e.g. the amount of memory the process is using.

Por compatibilidad con versiones anteriores, el valor retornado es accesible también como una tupla de 16 elementos.

The fields **`ru_utime`** and **`ru_stime`** of the return value are floating-point values representing the amount of time spent executing in user mode and the amount of time spent executing in system mode, respectively. The remaining values are integers. Consult the *getrusage(2)* *[https://manpages.debian.org/getrusage(2)]* man page for detailed information about these values. A brief summary is presented here:

Índice	Campo	Recurso
0	**`ru_utime`**	tiempo en modo usuario (flotante en segundos)
1	**`ru_stime`**	tiempo en modo sistema (flotante en segundos)
2	**`ru_maxrss`**	tamaño máximo del conjunto residente
3	**`ru_ixrss`**	tamaño de memoria compartida
4	**`ru_idrss`**	tamaño de memoria no compartida
5	**`ru_isrss`**	tamaño de la pila no compartida
6	**`ru_minflt`**	fallos de página que no requieran E/S
7	**`ru_majflt`**	fallos de página que requieran E/S
8	**`ru_nswap`**	número de intercambios
9	**`ru_inblock`**	bloque de operaciones de entrada
10	**`ru_oublock`**	bloque de operaciones de salida
11	**`ru_msgsnd`**	mensajes enviados
12	**`ru_msgrcv`**	mensajes recibidos
13	**`ru_nsignals`**	señales recibidas
14	**`ru_nvcsw`**	intercambios de contexto voluntarios
15	**`ru_nivcsw`**	intercambios de contexto involuntarios

Esta función lanzará un **`ValueError`** si el parámetro *who* especificado no es válido. También puede generar una excepción **`error`** en circunstancias inusuales.

resource.getpagesize()

Retorna el número de bytes en una página de sistema. (Esta no es necesariamente del mismo tamaño que la página de hardware).

The following **`RUSAGE_*`** symbols are passed to the `getrusage()` function to specify which processes information should be provided for.

resource.RUSAGE_SELF

> Pasar a `getrusage()` para solicitar recursos consumidos por el proceso de llamada, que es la suma de recursos utilizados por todos los hilos en el proceso.

resource.RUSAGE_CHILDREN

> Pasar a `getrusage()` para solicitar recursos consumidos por procesos secundarios del proceso de llamada que se han terminado o a los que se les está esperando.

resource.RUSAGE_BOTH

> Pasar a `getrusage()` para solicitar recursos consumidos por el proceso actual y sus procesos secundarios. Puede que no esté disponible en todos los sistemas.

resource.RUSAGE_THREAD

> Pasa a `getrusage()` para solicitar recursos consumidos por el hilo actual. Puede que no esté disponible en todos los sistemas.
>
> *Added in version 3.2.*

`syslog` — Unix syslog library routines

Este módulo ofrece una interfaz a las rutinas de la biblioteca `syslog` de Unix. Consulte las páginas del manual de Unix para una descripción detallada de la *facility* (es así como se llama en la documentación a un subsistema de aplicaciones) `syslog`.

Availability: Unix, not WASI, not iOS.

This module wraps the system `syslog` family of routines. A pure Python library that can speak to a syslog server is available in the **`logging.handlers`** module as **`SysLogHandler`**.

El módulo define las siguientes funciones:

syslog.syslog(*message*)
syslog.syslog(*priority*, *message*)

Envía la cadena de caracteres *message* al registrador del sistema. Se añade un final de línea si es necesario. Cada mensaje se etiqueta con una prioridad compuesta por una *facility* y un *nivel*. El argumento opcional *priority*, que por defecto es **`LOG_INFO`**, determina la prioridad del mensaje. Si la *facility* no está codificada en *priority* usando un OR lógico (`LOG_INFO | LOG_USER`), se usará el valor con el que se llama a **`openlog()`**.

Si **`openlog()`** no se ha llamado antes de la llamada a **`syslog()`**, entonces **`openlog()`** se llamará sin argumentos.

Lanza un evento de auditoría `syslog.syslog` con los argumentos `priority`, `message`.

Distinto en la versión 3.2: En versiones anteriores, **`openlog()`**

no sería llamado automáticamente si no fuera un llamado previo de la llamada a la función **`syslog()`**, postergando la implementación de syslog para llamar a `openlog()`.

Distinto en la versión 3.12: Esta función está restringida en subintérpretes. (Solo el código que se ejecuta en múltiples intérpretes se ve afectado y la restricción no es relevante para la mayoría de los usuarios.) **`openlog()`** debe ser llamada en el intérprete principal antes de que **`syslog()`** pueda ser utilizada en un subintérprete. De lo contrario, lanzará **`RuntimeError`**.

syslog.openlog([*ident*[, *logoption*[, *facility*]]])

Las opciones de registro de las llamadas a **`syslog()`** pueden establecerse llamando a la función **`openlog()`**. **`syslog()`** llamará a **`openlog()`** sin argumentos si el registro no está abierto actualmente.

El argumento por palabra clave opcional *ident* es una cadena de caracteres que precede a cada mensaje, y por defecto es `sys.argv[0]` retirando los componentes delanteros de la ruta. El argumento nombrado opcional *logoption* (por defecto es 0) es un campo de tipo bit – véanse a continuación los posibles valores que pueden combinarse. El argumento nombrado opcional *facility* (por defecto es **`LOG_USER`**) establece una *facility* por defecto para mensajes que no tienen una *facility* explícitamente codificada.

Lanza un evento de auditoría `syslog.openlog` con los argumentos `ident`, `logoption`, `facility`.

Distinto en la versión 3.2: En versiones anteriores, argumentos por palabras llave no eran permitidos, y la *ident* era obligatoria.

Distinto en la versión 3.12: Esta función está restringida en subintérpretes. (Solo el código que se ejecuta en múltiples intérpretes se ve afectado y la restricción no es relevante para la mayoría de los usuarios.) Esta solo puede ser llamada en el intérprete principal. Lanzará **`RuntimeError`** si se llama en un subintérprete.

syslog.closelog()

Reinicia los valores del módulo syslog y llama a la librería del sistema `closelog()`.

Esto provoca que el módulo actúe como lo hacía cuando fue importado inicialmente. Por ejemplo, se llamará a **`openlog()`** en la primera llamada a **`syslog()`** (si no se ha llamado a **`openlog()`** ya), e *ident* y otros parámetros de **`openlog()`** se reiniciarán a sus valores por defecto.

Lanza un evento de auditoría `syslog.closelog` sin argumentos.

Distinto en la versión 3.12: Esta función está restringida en subintérpretes. (Solo el código que se ejecuta en múltiples intérpretes se ve afectado y la restricción no es relevante para la mayoría de los usuarios.) Esta solo puede ser llamada en el intérprete principal. Lanzará **`RuntimeError`** si se llama en un subintérprete.

syslog.setlogmask(*maskpri*)

Asigna la máscara de prioridad a *maskpri* y retorna el valor anterior de la máscara. Las llamadas a **`syslog()`** con un nivel de prioridad no asignado en *maskpri* se ignoran. El comportamiento por defecto es registrar todas las prioridades. La función `LOG_MASK(pri)` calcula la máscara para la prioridad individual *pri*. La función `LOG_UPTO(pri)` calcula la máscara para todas las prioridades mayores o iguales que *pri*.

Lanza un evento de auditoría `syslog.setlogmask` con argumento `maskpri`.

El módulo define las siguientes constantes:

syslog.LOG_EMERG
syslog.LOG_ALERT
syslog.LOG_CRIT

syslog.LOG_ERR
syslog.LOG_WARNING
syslog.LOG_NOTICE
syslog.LOG_INFO
syslog.LOG_DEBUG

Priority levels (high to low).

syslog.LOG_AUTH
syslog.LOG_AUTHPRIV
syslog.LOG_CRON
syslog.LOG_DAEMON
syslog.LOG_FTP
syslog.LOG_INSTALL
syslog.LOG_KERN
syslog.LOG_LAUNCHD
syslog.LOG_LPR
syslog.LOG_MAIL
syslog.LOG_NETINFO
syslog.LOG_NEWS
syslog.LOG_RAS
syslog.LOG_REMOTEAUTH
syslog.LOG_SYSLOG
syslog.LOG_USER
syslog.LOG_UUCP
syslog.LOG_LOCAL0
syslog.LOG_LOCAL1
syslog.LOG_LOCAL2
syslog.LOG_LOCAL3
syslog.LOG_LOCAL4
syslog.LOG_LOCAL5
syslog.LOG_LOCAL6
syslog.LOG_LOCAL7

Facilities, depending on availability in `<syslog.h>` for **`LOG_AUTHPRIV`**, **`LOG_FTP`**, **`LOG_NETINFO`**, **`LOG_REMOTEAUTH`**, **`LOG_INSTALL`** and **`LOG_RAS`**.

Distinto en la versión 3.13: Added **`LOG_FTP`**, **`LOG_NETINFO`**, **`LOG_REMOTEAUTH`**, **`LOG_INSTALL`**, **`LOG_RAS`**, and **`LOG_LAUNCHD`**.

syslog.LOG_PID
syslog.LOG_CONS
syslog.LOG_NDELAY
syslog.LOG_ODELAY
syslog.LOG_NOWAIT
syslog.LOG_PERROR

Log options, depending on availability in `<syslog.h>` for **`LOG_ODELAY`**, **`LOG_NOWAIT`** and **`LOG_PERROR`**.

Ejemplos

Ejemplo sencillo

Un conjunto sencillo de ejemplos:

```
import syslog

syslog.syslog('Processing started')
if error:
    syslog.syslog(syslog.LOG_ERR, 'Processing started')
```

Un ejemplo de configuración de varias opciones de registro, que incluirán el identificador del proceso en los mensajes registrados, y escribirán los mensajes a la facility de destino usada para el registro de correo:

```
syslog.openlog(logoption=syslog.LOG_PID, facility=syslog
syslog.syslog('E-mail processing initiated...')
```

Modules command-line interface (CLI)

The following modules have a command-line interface.

- ast
- asyncio
- `base64`
- calendar
- `code`
- compileall
- `cProfile`: see profile
- dis
- doctest
- `encodings.rot_13`
- ensurepip
- `filecmp`
- `fileinput`
- `ftplib`
- gzip
- http.server
- idlelib
- inspect
- json
- mimetypes
- pdb
- pickle
- pickletools
- platform
- `poplib`
- profile
- `pstats`
- py_compile
- `pyclbr`
- `pydoc`

- **`quopri`**
- random
- **`runpy`**
- site
- sqlite3
- symtable
- sysconfig
- **`tabnanny`**
- tarfile
- **`this`**
- timeit
- tokenize
- trace
- **`turtledemo`**
- unittest
- uuid
- venv
- webbrowser
- zipapp
- zipfile

See also the Python command-line interface.

Módulos reemplazados

The modules described in this chapter have been superseded by other modules for most use cases, and are retained primarily to preserve backwards compatibility.

Modules may appear in this chapter because they only cover a limited subset of a problem space, and a more generally applicable solution is available elsewhere in the standard library (for example, **`getopt`** covers the very specific task of «mimic the C **`getopt()`** API in Python», rather than the broader command line option parsing and argument parsing capabilities offered by **`optparse`** and **`argparse`**).

Alternatively, modules may appear in this chapter because they are deprecated outright, and awaiting removal in a future release, or they are soft deprecated and their use is actively discouraged in new projects. With the removal of various obsolete modules through **PEP 594** [https://peps.python.org/pep-0594/], there are currently no modules in this latter category.

- **`getopt`** — C-style parser for command line options

`getopt` — C-style parser for command line options

Código fuente: Lib/getopt.py [https://github.com/python/cpython/tree/3.14/Lib/getopt.py]

Nota

This module is considered feature complete. A more declarative and extensible alternative to this API is provided in the **`optparse`** module. Further functional enhancements for command line parameter processing are provided either as third party modules on PyPI, or else as features in the **`argparse`** module.

This module helps scripts to parse the command line arguments in `sys.argv`. It supports the same conventions as the Unix **`getopt()`** function (including the special meanings of arguments of the form "-" and "--"). Long options similar to those supported by GNU software may be used as well via an optional third argument.

Users who are unfamiliar with the Unix **`getopt()`** function should consider using the **`argparse`** module instead. Users who are familiar with the Unix **`getopt()`** function, but would like to get equivalent behavior while writing less code and getting better help and error messages should consider using the **`optparse`** module. See Choosing an argument parsing library for additional details.

Este módulo proporciona dos funciones y una excepción:

getopt.getopt(*args*, *shortopts*, *longopts=[]*)

Parses command line options and parameter list. *args* is the argument list to be parsed, without the leading reference to the running program. Typically, this means `sys.argv[1:]`. *shortopts* is the string of option letters that the script wants to recognize, with options that require an argument followed by a colon (`':'`) and options that accept an optional argument followed by two colons (`'::'`); i.e., the same format that Unix **`getopt()`** uses.

Nota

Unlike GNU **`getopt()`**, after a non-option argument, all further arguments are considered also non-options. This is similar to the way non-GNU Unix systems work.

longopts, if specified, must be a list of strings with the names of the long options which should be supported. The leading `'--'` characters should not be included in the option name. Long options which require an argument should be followed by an equal sign (`'='`). Long options which accept an optional argument should be followed by an equal sign and question mark (`'=?'`). To accept only long options, *shortopts* should be an empty string. Long options on the command line can be recognized so long as they provide a prefix of the option name that matches exactly one of the accepted options. For example, if *longopts* is `['foo', 'frob']`, the option `--fo` will match as `--foo`, but `--f` will not match uniquely, so **`GetoptError`** will be raised.

El valor de retorno consta de dos elementos: el primero es una lista de pares `(option, value)`; el segundo es la lista de argumentos del programa que quedan después de que se eliminó la lista de opciones (esta es una porción final de *args*). Cada par de opción y valor retornado tiene la opción como su primer elemento, con un guión para las opciones cortas (por ejemplo, `'-x'`) o dos guiones para las opciones largas (por ejemplo, `'--long-option'`), y el argumento de la opción como su segundo elemento, o una cadena vacía si la opción no tiene argumento. Las opciones aparecen en la lista en el mismo orden en que se encontraron, lo que permite múltiples ocurrencias. Las opciones largas y cortas pueden ser mixtas.

Distinto en la versión 3.14: Optional arguments are supported.

getopt.gnu_getopt(*args*, *shortopts*, *longopts=[]*)

Esta función funciona como `getopt()`, excepto que el modo de escaneo estilo GNU se usa por defecto. Esto significa que los argumentos opcionales y no opcionales pueden estar mezclados. La función **`getopt()`** detiene el procesamiento de opciones tan pronto como se encuentra un argumento no-opcionales.

If the first character of the option string is `'+'`, or if the environment variable **`POSIXLY_CORRECT`** is set, then option processing stops as soon as a non-option argument is encountered.

If the first character of the option string is `'-'`, non-option arguments that are followed by options are added to the list of option-and-value pairs as a pair that has `None` as its first element and the list of non-option arguments as its second element. The second element of the **`gnu_getopt()`** result is a list of program arguments after the last option.

Distinto en la versión 3.14: Support for returning intermixed options and non-option arguments in order.

exception getopt.GetoptError

This is raised when an unrecognized option is found in the argument list or when an option requiring an argument is given none. The argument to the exception is a string indicating the cause of the error. For long options, an argument given to an option which does not require one will also cause this exception to be raised. The attributes **`msg`** and **`opt`** give the error message and related option; if there is no specific option to which the exception relates, **`opt`** is an empty string.

exception getopt.error

Alias para `GetoptError`; para compatibilidad con versiones anteriores.

Un ejemplo que usa solo opciones de estilo Unix:

```
>>> import getopt
>>> args = '-a -b -cfoo -d bar a1 a2'.split()
>>> args
['-a', '-b', '-cfoo', '-d', 'bar', 'a1', 'a2']
>>> optlist, args = getopt.getopt(args, 'abc:d:')
>>> optlist
[('-a', ''), ('-b', ''), ('-c', 'foo'), ('-d', 'bar')]
>>> args
['a1', 'a2']
```

Usar nombres largos de opciones es igualmente fácil:

```
>>> s = '--condition=foo --testing --output-file abc.def
>>> args = s.split()
>>> args
['--condition=foo', '--testing', '--output-file', 'abc.d
>>> optlist, args = getopt.getopt(args, 'x', [
...     'condition=', 'output-file=', 'testing'])
>>> optlist
[('--condition', 'foo'), ('--testing', ''), ('--output-f
>>> args
['a1', 'a2']
```

Optional arguments should be specified explicitly:

```
>>> s = '-Con -C --color=off --color a1 a2'
>>> args = s.split()
>>> args
['-Con', '-C', '--color=off', '--color', 'a1', 'a2']
>>> optlist, args = getopt.getopt(args, 'C::', ['color=?
>>> optlist
[('-C', 'on'), ('-C', ''), ('--color', 'off'), ('--color
>>> args
['a1', 'a2']
```

The order of options and non-option arguments can be preserved:

```
>>> s = 'a1 -x a2 a3 a4 --long a5 a6'
>>> args = s.split()
>>> args
['a1', '-x', 'a2', 'a3', 'a4', '--long', 'a5', 'a6']
>>> optlist, args = getopt.gnu_getopt(args, '-x:', ['lor
>>> optlist
[(None, ['a1']), ('-x', 'a2'), (None, ['a3', 'a4']), ('-
>>> args
['a6']
```

In a script, typical usage is something like this:

```
import getopt, sys

def main():
    try:
        opts, args = getopt.getopt(sys.argv[1:], "ho:v",
    except getopt.GetoptError as err:
        # print help information and exit:
        print(err)  # will print something like "option
        usage()
        sys.exit(2)
    output = None
    verbose = False
    for o, a in opts:
        if o == "-v":
            verbose = True
        elif o in ("-h", "--help"):
            usage()
            sys.exit()
        elif o in ("-o", "--output"):
            output = a
        else:
            assert False, "unhandled option"
    process(args, output=output, verbose=verbose)

if __name__ == "__main__":
    main()
```

Note that an equivalent command line interface could be produced with less code and more informative help and error messages by using the **`optparse`** module:

```
import optparse

if __name__ == '__main__':
    parser = optparse.OptionParser()
    parser.add_option('-o', '--output')
    parser.add_option('-v', dest='verbose', action='stor
    opts, args = parser.parse_args()
    process(args, output=opts.output, verbose=opts.verbo
```

A roughly equivalent command line interface for this case can also be produced by using the **`argparse`** module:

```
import argparse

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-o', '--output')
    parser.add_argument('-v', dest='verbose', action='st
    parser.add_argument('rest', nargs='*')
    args = parser.parse_args()
    process(args.rest, output=args.output, verbose=args.
```

See Choosing an argument parsing library for details on how the `argparse` version of this code differs in behaviour from the `optparse` (and `getopt`) version.

Ver también

Module `optparse`
Declarative command line option parsing.

Módulo `argparse`
More opinionated command line option and argument parsing library.

Removed Modules

The modules described in this chapter have been removed from the Python standard library. They are documented here to help people find replacements.

- **`aifc`** — Read and write AIFF and AIFC files
- **`asynchat`** — Asynchronous socket command/response handler
- **`asyncore`** — Asynchronous socket handler
- **`audioop`** — Manipulate raw audio data
- **`cgi`** — Common Gateway Interface support
- **`cgitb`** — Traceback manager for CGI scripts
- **`chunk`** — Read IFF chunked data
- **`crypt`** — Function to check Unix passwords
- **`distutils`** — Building and installing Python modules
- **`imghdr`** — Determine the type of an image
- **`imp`** — Access the import internals
- **`mailcap`** — Mailcap file handling
- **`msilib`** — Read and write Microsoft Installer files
- **`nis`** — Interface to Sun's NIS (Yellow Pages)
- **`nntplib`** — NNTP protocol client
- **`ossaudiodev`** — Access to OSS-compatible audio devices
- **`pipes`** — Interface to shell pipelines
- **`smtpd`** — SMTP Server
- **`sndhdr`** — Determine type of sound file
- **`spwd`** — The shadow password database
- **`sunau`** — Read and write Sun AU files
- **`telnetlib`** — Telnet client
- **`uu`** — Encode and decode uuencode files
- **`xdrlib`** — Encode and decode XDR data

Consideraciones de seguridad

Los siguientes módulos tienen consideraciones de seguridad específicas:

- **`base64`**: consideraciones de seguridad de base64 en **RFC 4648** [https://datatracker.ietf.org/doc/html/rfc4648.html]
- **`hashlib`**: todos los constructores toman un argumento de solo palabra clave «usedforsecurity» que deshabilita algoritmos inseguros y bloqueados conocidos
- **`http.server`** no se recomienda para producción. Sólo implementa controles de seguridad básicos. Consulte las consideraciones de seguridad.
- **`logging`**: La configuración de registro usa eval()
- **`multiprocessing`**: Connection.recv() usa pickle
- **`pickle`**: Restricción de globals en pickle
- **`random`** no debe usarse por motivos de seguridad, usa **`secrets`** en su lugar
- **`shelve`**: el estante se basa en pickle y, por lo tanto, no es adecuado para tratar con fuentes no confiables
- **`ssl`**: consideraciones de seguridad SSL/TLS
- **`subprocess`**: Consideraciones sobre seguridad de subprocesos
- **`tempfile`**: mktemp está en desuso debido a la vulnerabilidad a condiciones de urgencia
- **`xml`**: XML security
- **`zipfile`**: los archivos de .zip preparados maliciosamente pueden causar agotamiento del volumen del disco

La opción de línea de comando **`-I`** puede usarse para correr Python en modo aislado. Cuando no puede ser usado, la opción **`-P`** o la variable de entorno **`PYTHONSAFEPATH`** pueden usarse para no anteponer un camino potencialmente inseguro al **`sys.path`** como el directorio actual, el directorio del script o un string vacío.

Ampliación e incrustación del intérprete de Python

Este documento describe cómo escribir módulos en C o C++ para extender el intérprete de Python con nuevos módulos. Esos módulos no solo pueden definir nuevas funciones sino también nuevos tipos de objetos y sus métodos. El documento también describe cómo incrustar el intérprete de Python en otra aplicación, para usarlo como un lenguaje de extensión. Finalmente, muestra cómo compilar y vincular módulos de extensión para que puedan cargarse dinámicamente (en tiempo de ejecución) en el intérprete, si el sistema operativo subyacente admite esta característica.

Este documento asume conocimientos básicos sobre Python. Para una introducción informal al lenguaje, consulte El tutorial de Python. Referencia del Lenguaje Python da una definición más formal del lenguaje. La biblioteca estándar de Python documenta los tipos de objetos, funciones y módulos existentes (tanto incorporados como escritos en Python) que le dan al lenguaje su amplio rango de aplicaciones.

Para obtener una descripción detallada de toda la API de Python/C, consulte el apartado separado Python/C API reference manual.

Herramientas de terceros recomendadas

This guide only covers the basic tools for creating extensions provided as part of this version of CPython. Some third party tools offer both simpler and more sophisticated approaches to creating C and C++ extensions for Python.

Crear extensiones sin herramientas de terceros

Esta sección de la guía cubre la creación de extensiones C y C++ sin la ayuda de herramientas de terceros. Está destinado principalmente a los creadores de esas herramientas, en lugar de ser una forma recomendada de crear sus propias extensiones C.

Ver también

PEP 489 [https://peps.python.org/pep-0489/] – Multi-phase extension module initialization

- 1. Extendiendo Python con C o C++
    - 1.1. Un ejemplo simple
    - 1.2. Intermezzo: errores y excepciones
    - 1.3. De vuelta al ejemplo
    - 1.4. La tabla de métodos del módulo y la función de inicialización
    - 1.5. Compilación y enlazamiento
    - 1.6. Llamando funciones Python desde C
    - 1.7. Extracción de parámetros en funciones de extensión
    - 1.8. Parámetros de palabras clave para funciones de extensión
    - 1.9. Construyendo valores arbitrarios
    - 1.10. Conteo de referencias
    - 1.11. Escribiendo extensiones en C++
    - 1.12. Proporcionar una API C para un módulo de extensión
- 2. Definición de tipos de extensión: Tutorial
    - 2.1. Lo básico
    - 2.2. Agregar datos y métodos al ejemplo básico
    - 2.3. Proporcionar un control más preciso sobre los atributos de datos
    - 2.4. Apoyo a la recolección de basura cíclica
    - 2.5. Subclases de otros tipos
- 3. Definición de tipos de extensión: temas variados

    - 3.1. Finalización y desasignación
    - 3.2. Presentación de objetos
    - 3.3. Gestión de atributos
    - 3.4. Comparación de Objetos
    - 3.5. Soporte de protocolo abstracto
    - 3.6. Soporte de referencia débil
    - 3.7. Más Sugerencias
- 4. Construyendo extensiones C y C++
    - 4.1. Building C and C++ Extensions with setuptools
- 5. Creación de extensiones C y C++ en Windows
    - 5.1. Un enfoque de libro de cocina
    - 5.2. Diferencias entre Unix y Windows
    - 5.3. Usar DLL en la práctica

Incrustar el tiempo de ejecución de CPython en una aplicación más grande

A veces, en lugar de crear una extensión que se ejecute dentro del intérprete de Python como la aplicación principal, es conveniente incorporar el tiempo de ejecución de CPython dentro de una aplicación más grande. Esta sección cubre algunos de los detalles involucrados en hacerlo con éxito.

- 1. Incrustando Python en otra aplicación
    - 1.1. Incrustación de muy alto nivel
    - 1.2. Más allá de la incrustación de muy alto nivel: una visión general
    - 1.3. Incrustación pura
    - 1.4. Extendiendo Python incrustado
    - 1.5. Incrustando Python en C++
    - 1.6. Compilar y enlazar bajo sistemas tipo Unix

[1] [2] [3] [4]

1. Extendiendo Python con C o C++

Es muy fácil agregar nuevos módulos incorporados a Python, si sabe cómo programar en C. Tales como *módulos de extensión* pueden hacer dos cosas que no se pueden hacer directamente en Python: pueden implementar nuevos tipos objetos incorporados, y pueden llamar a funciones de biblioteca C y llamadas de sistema.

Para admitir extensiones, la API de Python (interfaz de programadores de aplicaciones) define un conjunto de funciones, macros y variables que proporcionan acceso a la mayoría de los aspectos del sistema de tiempo de ejecución de Python. La API de Python se incorpora en un archivo fuente C incluyendo el encabezado `"Python.h"`.

La compilación de un módulo de extensión depende de su uso previsto, así como de la configuración de su sistema; los detalles se dan en capítulos posteriores.

Nota

La interfaz de extensión C es específica de CPython, y los módulos de extensión no funcionan en otras implementaciones de Python. En muchos casos, es posible evitar escribir extensiones C y preservar la portabilidad a otras implementaciones. Por ejemplo, si su caso de uso es llamar a funciones de biblioteca C o llamadas de sistema, debería considerar usar el módulo **`ctypes`** o la biblioteca cffi [https://cffi.readthedocs.io/] en lugar de escribir código personalizado C. Estos módulos le permiten escribir código Python para interactuar con el código C y son más portátiles entre las implementaciones de Python que escribir y compilar un módulo de extensión C.

1.1. Un ejemplo simple

Creemos un módulo de extensión llamado `spam` (la comida favorita de los fanáticos de Monty Python …) y digamos que queremos crear una interfaz de Python para la función de biblioteca C **`system()`** [1] . Esta función toma una cadena de caracteres con terminación nula como argumento y retorna un entero. Queremos que esta función se pueda llamar desde Python de la siguiente manera:

```
>>> import spam
>>> status = spam.system("ls -l")
```

Comience creando un archivo `spammodule.c`. (Históricamente, si un módulo se llama `spam`, el archivo C que contiene su implementación se llama `spammodule.c`; si el nombre del módulo es muy largo, como `spammify`, el nombre del módulo puede sea solo `spammify.c`.)

Las dos primeras líneas de nuestro archivo pueden ser:

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>
```

que extrae la API de Python (puede agregar un comentario que describa el propósito del módulo y un aviso de copyright si lo desea).

Nota

Dado que Python puede definir algunas definiciones de preprocesador que afectan los encabezados estándar en algunos sistemas, *debe* incluir `Python.h` antes de incluir encabezados estándar.

`#define PY_SSIZE_T_CLEAN` was used to indicate that `Py_ssize_t` should be used in some APIs instead of `int`. It is not necessary since Python 3.13, but we keep it here for backward compatibility. See Cadena de caracteres y búferes for a description of this macro.

All user-visible symbols defined by `Python.h` have a prefix of `Py` or `PY`, except those defined in standard header files.

Truco

For backward compatibility, `Python.h` includes several standard header files. C extensions should include the standard headers that they use, and should not rely on these implicit includes. If using the limited C API version 3.13 or newer, the implicit includes are:

- `<assert.h>`
- `<intrin.h>` (on Windows)
- `<inttypes.h>`
- `<limits.h>`
- `<math.h>`
- `<stdarg.h>`
- `<wchar.h>`
- `<sys/types.h>` (if present)

If **`Py_LIMITED_API`** is not defined, or is set to version 3.12 or older, the headers below are also included:

- `<ctype.h>`
- `<unistd.h>` (on POSIX)

If **`Py_LIMITED_API`** is not defined, or is set to version 3.10 or older, the headers below are also included:

- `<errno.h>`
- `<stdio.h>`
- `<stdlib.h>`
- `<string.h>`

Lo siguiente que agregamos a nuestro archivo de módulo es la función C que se llamará cuando se evalúe la expresión Python `spam.system(string)` (veremos en breve cómo termina siendo llamado):

```
static PyObject *
```

```
spam_system(PyObject *self, PyObject *args)
{
    const char *command;
    int sts;

    if (!PyArg_ParseTuple(args, "s", &command))
        return NULL;
    sts = system(command);
    return PyLong_FromLong(sts);
}
```

Hay una traducción directa de la lista de argumentos en Python (por ejemplo, la única expresión `"ls -l"`) a los argumentos pasados a la función C. La función C siempre tiene dos argumentos, llamados convencionalmente *self* y *args*.

El argumento *self* apunta al objeto del módulo para funciones a nivel de módulo; para un método apuntaría a la instancia del objeto.

El argumento *args* será un puntero a un objeto de tupla de Python que contiene los argumentos. Cada elemento de la tupla corresponde a un argumento en la lista de argumentos de la llamada. Los argumentos son objetos de Python — para hacer algo con ellos en nuestra función C tenemos que convertirlos a valores C. La función **`PyArg_ParseTuple()`** en la API de Python verifica los tipos de argumento y los convierte a valores C. Utiliza una cadena de plantilla para determinar los tipos requeridos de los argumentos, así como los tipos de las variables C en las que almacenar los valores convertidos. Más sobre esto más tarde.

`PyArg_ParseTuple()` retorna verdadero (distinto de cero) si todos los argumentos tienen el tipo correcto y sus componentes se han almacenado en las variables cuyas direcciones se pasan. Retorna falso (cero) si se pasó una lista de argumentos no válidos. En el último caso, también genera una excepción apropiada para que la función de llamada pueda retornar `NULL` inmediatamente (como vimos en el ejemplo).

1.2. Intermezzo: errores y excepciones

Una convención importante en todo el intérprete de Python es la siguiente: cuando una función falla, debe establecer una condición de excepción y retornar un valor de error (generalmente `-1` o un puntero `NULL`). La información de excepción se almacena en tres miembros del estado del subproceso del intérprete. Estos son `NULL` si no hay excepción. De lo contrario, son los equivalentes en C de los miembros de la tupla Python retornada por **`sys.exc_info()`**. Estos son el tipo de excepción, la instancia de excepción y un objeto de rastreo. Es importante conocerlos para comprender cómo se transmiten los errores.

La API de Python define una serie de funciones para establecer varios tipos de excepciones.

El más común es **`PyErr_SetString()`**. Sus argumentos son un objeto de excepción y una cadena C. El objeto de excepción suele ser un objeto predefinido como **`PyExc_ZeroDivisionError`**. La cadena C indica la causa del error y se convierte en un objeto de cadena Python y se almacena como el «valor asociado» de la excepción.

Otra función útil es **`PyErr_SetFromErrno()`**, que solo toma un argumento de excepción y construye el valor asociado mediante la inspección de la variable global **`errno`**. La función más general es **`PyErr_SetObject()`**, que toma dos argumentos de objeto, la excepción y su valor asociado. No necesita **`Py_INCREF()`** los objetos pasados a cualquiera de estas funciones.

Puede probar de forma no destructiva si se ha establecido una excepción con **`PyErr_Occurred()`**. Esto retorna el objeto de excepción actual o `NULL` si no se ha producido ninguna excepción. Normalmente no necesita llamar a **`PyErr_Occurred()`** para ver si se produjo un error en una llamada a la función, ya que debería poder distinguir el valor de retorno.

Cuando una función *f* que llama a otra función *g* detecta que esta última falla, *f* debería retornar un valor de error (normalmente `NULL` o `-1`). Debería *no* llamar a una de las funciones `PyErr_*` — *g* ya ha llamado a una. Se supone que la persona que llama a *f* también retornará una indicación de error a *su* persona que la llama, nuevamente *sin* llama a `PyErr_*`, y así sucesivamente — la función que lo detectó primero ya informó la causa más detallada del error. Una vez que el error llega al bucle principal del intérprete de Python, este aborta el código de Python que se está ejecutando actualmente e intenta encontrar un controlador de excepciones especificado por el programador de Python.

(Hay situaciones en las que un módulo puede dar un mensaje de error más detallado llamando a otra función `PyErr_*`, y en tales casos está bien hacerlo. Sin embargo, como regla general, esto no es necesario y puede causar que se pierda información sobre la causa del error: la mayoría de las operaciones pueden fallar por una variedad de razones).

Para ignorar una excepción establecida por una llamada de función que falló, la condición de excepción debe borrarse explícitamente llamando a **`PyErr_Clear()`**. La única vez que el código C debe llamar **`PyErr_Clear()`** es si no quiere pasar el error al intérprete pero quiere manejarlo completamente por sí mismo (posiblemente probando algo más o pretendiendo que nada salió mal))

Cada llamada fallida a **`malloc()`** debe convertirse en una excepción — la persona que llama directamente de **`malloc()`** (o **`realloc()`**) debe llamar **`PyErr_NoMemory()`** y retorna un indicador de falla en sí mismo. Todas las funciones de creación de objetos (por ejemplo, **`PyLong_FromLong()`**) ya hacen esto, por lo que esta nota solo es relevante para aquellos que llaman **`malloc()`** directamente.

También tenga en cuenta que, con la importante excepción de **`PyArg_ParseTuple()`** y sus amigos, las funciones que retornan un estado entero generalmente retornan un valor positivo o cero para el éxito y `-1` para el fracaso, como las llamadas al sistema Unix.

Finalmente, tenga cuidado de limpiar la basura (haciendo **`Py_XDECREF()`** o **`Py_DECREF()`** requiere objetos que ya ha creado) cuando retorna un indicador de error!

The choice of which exception to raise is entirely yours. There are predeclared C objects corresponding to all built-in Python exceptions, such as **`PyExc_ZeroDivisionError`**, which you can

use directly. Of course, you should choose exceptions wisely — don't use `PyExc_TypeError` to mean that a file couldn't be opened (that should probably be `PyExc_OSError`). If something's wrong with the argument list, the `PyArg_ParseTuple()` function usually raises **`PyExc_TypeError`**. If you have an argument whose value must be in a particular range or must satisfy other conditions, `PyExc_ValueError` is appropriate.

You can also define a new exception that is unique to your module. The simplest way to do this is to declare a static global object variable at the beginning of the file:

```
static PyObject *SpamError = NULL;
```

and initialize it by calling `PyErr_NewException()` in the module's `Py_mod_exec` function (**`spam_module_exec()`**):

```
SpamError = PyErr_NewException("spam.error", NULL, NULL)
```

Since **`SpamError`** is a global variable, it will be overwritten every time the module is reinitialized, when the `Py_mod_exec` function is called.

For now, let's avoid the issue: we will block repeated initialization by raising an `ImportError`:

```
static PyObject *SpamError = NULL;

static int
spam_module_exec(PyObject *m)
{
    if (SpamError != NULL) {
        PyErr_SetString(PyExc_ImportError,
                        "cannot initialize spam module m
        return -1;
    }
    SpamError = PyErr_NewException("spam.error", NULL, N
    if (PyModule_AddObjectRef(m, "SpamError", SpamError)
        return -1;
    }
    return 0;
}

static PyModuleDef_Slot spam_module_slots[] = {
    {Py_mod_exec, spam_module_exec},
    {0, NULL}
};

static struct PyModuleDef spam_module = {
    .m_base = PyModuleDef_HEAD_INIT,
    .m_name = "spam",
    .m_size = 0,  // non-negative
    .m_slots = spam_module_slots,
};

PyMODINIT_FUNC
PyInit_spam(void)
{
    return PyModuleDef_Init(&spam_module);
}
```

Note that the Python name for the exception object is **`spam.error`**. The **`PyErr_NewException()`** function may create a class with the base class being **`Exception`** (unless another class is passed in instead of `NULL`), described in Excepciones incorporadas.

Note also that the **`SpamError`** variable retains a reference to the newly created exception class; this is intentional! Since the exception could be removed from the module by external code, an owned reference to the class is needed to ensure that it will not be discarded, causing **`SpamError`** to become a dangling pointer. Should it become a dangling pointer, C code which raises the exception could cause a core dump or other unintended side effects.

For now, the **`Py_DECREF()`** call to remove this reference is missing. Even when the Python interpreter shuts down, the global **`SpamError`** variable will not be garbage-collected. It will «leak». We did, however, ensure that this will happen at most once per process.

We discuss the use of `PyMODINIT_FUNC` as a function return type later in this sample.

The **`spam.error`** exception can be raised in your extension module using a call to `PyErr_SetString()` as shown below:

```
static PyObject *
spam_system(PyObject *self, PyObject *args)
{
    const char *command;
    int sts;

    if (!PyArg_ParseTuple(args, "s", &command))
        return NULL;
    sts = system(command);
    if (sts < 0) {
        PyErr_SetString(SpamError, "System command faile
        return NULL;
    }
    return PyLong_FromLong(sts);
}
```

1.3. De vuelta al ejemplo

Volviendo a nuestra función de ejemplo, ahora debería poder comprender esta declaración:

```
if (!PyArg_ParseTuple(args, "s", &command))
    return NULL;
```

It returns `NULL` (the error indicator for functions returning object pointers) if an error is detected in the argument list, relying on the exception set by `PyArg_ParseTuple()`. Otherwise the string value of the argument has been copied to the local variable **`command`**. This is a pointer assignment and you are not supposed to modify the string to which it points (so in Standard C, the variable **`command`** should properly be declared as `const char *command`).

La siguiente declaración es una llamada a la función Unix **`system()`**, pasándole la cadena que acabamos de obtener de **`PyArg_ParseTuple()`**:

```
sts = system(command);
```

Our **`spam.system()`** function must return the value of **`sts`** as a Python object. This is done using the function **`PyLong_FromLong()`**.

```
return PyLong_FromLong(sts);
```

En este caso, retornará un objeto entero. (Sí, ¡incluso los enteros son objetos en el montículo (*heap*) en Python!)

Si tiene una función C que no retorna ningún argumento útil (una función que retorna void), la función de Python correspondiente debe retornar `None`. Necesita esta expresión para hacerlo (que se implementa mediante la macro **`Py_RETURN_NONE`**):

```
Py_INCREF(Py_None);
return Py_None;
```

`Py_None` es el nombre C para el objeto especial de Python `None`. Es un objeto genuino de Python en lugar de un puntero `NULL`, que significa «error» en la mayoría de los contextos, como hemos visto.

1.4. La tabla de métodos del módulo y la función de inicialización

I promised to show how **`spam_system()`** is called from Python programs. First, we need to list its name and address in a «method table»:

```
static PyMethodDef spam_methods[] = {
    ...
    {"system",  spam_system, METH_VARARGS,
     "Execute a shell command."},
    ...
    {NULL, NULL, 0, NULL}        /* Sentinel */
};
```

Tenga en cuenta la tercera entrada (`METH_VARARGS`). Esta es una bandera que le dice al intérprete la convención de llamada que se utilizará para la función C. Normalmente debería ser siempre `METH_VARARGS` o `METH_VARARGS | METH_KEYWORDS`; un valor de `0` significa que se usa una variante obsoleta de **`PyArg_ParseTuple()`**.

Cuando se usa solo `METH_VARARGS`, la función debe esperar que los parámetros a nivel de Python se pasen como una tupla aceptable para el análisis mediante **`PyArg_ParseTuple()`**; A continuación se proporciona más información sobre esta función.

The **`METH_KEYWORDS`** bit may be set in the third field if keyword arguments should be passed to the function. In this case, the C function should accept a third `PyObject *` parameter which will be a dictionary of keywords. Use **`PyArg_ParseTupleAndKeywords()`** to parse the arguments to such a function.

La tabla de métodos debe ser referenciada en la estructura de definición del módulo:

```
static struct PyModuleDef spam_module = {
    ...
    .m_methods = spam_methods,
    ...
};
```

This structure, in turn, must be passed to the interpreter in the module’s initialization function. The initialization function must be named **`PyInit_name()`**, where *name* is the name of the module, and should be the only non-`static` item defined in the module file:

```
PyMODINIT_FUNC
PyInit_spam(void)
{
    return PyModuleDef_Init(&spam_module);
}
```

Note that `PyMODINIT_FUNC` declares the function as `PyObject *` return type, declares any special linkage declarations required by the platform, and for C++ declares the function as `extern "C"`.

`PyInit_spam()` is called when each interpreter imports its module **`spam`** for the first time. (See below for comments about embedding Python.) A pointer to the module definition must be returned via `PyModuleDef_Init()`, so that the import machinery can create the module and store it in `sys.modules`.

When embedding Python, the **`PyInit_spam()`** function is not called automatically unless there's an entry in the `PyImport_Inittab` table. To add the module to the initialization table, use `PyImport_AppendInittab()`, optionally followed by an import of the module:

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>

int
main(int argc, char *argv[])
{
    PyStatus status;
    PyConfig config;
    PyConfig_InitPythonConfig(&config);

    /* Add a built-in module, before Py_Initialize */
    if (PyImport_AppendInittab("spam", PyInit_spam) == -
        fprintf(stderr, "Error: could not extend in-buil
        exit(1);
    }

    /* Pass argv[0] to the Python interpreter */
    status = PyConfig_SetBytesString(&config, &config.pr
    if (PyStatus_Exception(status)) {
        goto exception;
    }

    /* Initialize the Python interpreter.  Required.
```

```
       If this step fails, it will be a fatal error. */
    status = Py_InitializeFromConfig(&config);
    if (PyStatus_Exception(status)) {
        goto exception;
    }
    PyConfig_Clear(&config);

    /* Optionally import the module; alternatively,
       import can be deferred until the embedded script
       imports it. */
    PyObject *pmodule = PyImport_ImportModule("spam");
    if (!pmodule) {
        PyErr_Print();
        fprintf(stderr, "Error: could not import module
    }

    // ... use Python C API here ...

    return 0;

  exception:
     PyConfig_Clear(&config);
     Py_ExitStatusException(status);
}
```

Nota

If you declare a global variable or a local static one, the module may experience unintended side-effects on re-initialisation, for example when removing entries from `sys.modules` or importing compiled modules into multiple interpreters within a process (or following a **`fork()`** without an intervening **`exec()`**). If module state is not yet fully isolated, authors should consider marking the module as having no support for subinterpreters (via **`Py_MOD_MULTIPLE_INTERPRETERS_NOT_SUPPORTED`**).

A more substantial example module is included in the Python

source distribution as `Modules/xxlimited.c`. This file may be used as a template or simply read as an example.

1.5. Compilación y enlazamiento

Hay dos cosas más que hacer antes de que pueda usar su nueva extensión: compilarla y vincularla con el sistema Python. Si usa carga dinámica, los detalles pueden depender del estilo de carga dinámica que usa su sistema; Para obtener más información al respecto, consulte los capítulos sobre la creación de módulos de extensión (capítulo Construyendo extensiones C y C++) e información adicional que se refiere solo a la construcción en Windows (capítulo Creación de extensiones C y C++ en Windows).

Si no puede utilizar la carga dinámica, o si desea que su módulo sea una parte permanente del intérprete de Python, tendrá que cambiar la configuración (*setup*) y reconstruir el intérprete. Afortunadamente, esto es muy simple en Unix: simplemente coloque su archivo (`spammodule.c` por ejemplo) en el directorio `` Modules/ ` de una distribución fuente desempaquetada, agregue una línea al archivo :file:`Modules/Setup.local `` que describe su archivo:

```
spam spammodule.o
```

y reconstruya el intérprete ejecutando **make** en el directorio de nivel superior. También puede ejecutar **make** en el subdirectorio `Modules/`, pero primero debe reconstruir `Makefile` ejecutando "**make** Makefile". (Esto es necesario cada vez que cambia el archivo `Configuración`).

Si su módulo requiere bibliotecas adicionales para vincular, también se pueden enumerar en la línea del archivo de configuración, por ejemplo:

```
spam spammodule.o -lX11
```

1.6. Llamando funciones Python desde C

Hasta ahora nos hemos concentrado en hacer que las funciones de C puedan llamarse desde Python. Lo contrario también es útil: llamar a las funciones de Python desde C. Este es especialmente el caso de las bibliotecas que admiten las llamadas funciones de «retrollamada». Si una interfaz C utiliza retrollamadas, el Python equivalente a menudo necesita proporcionar un mecanismo de retrollamada al programador de Python; la implementación requerirá llamar a las funciones de retrollamada de Python desde una retrollamada en C. Otros usos también son imaginables.

Afortunadamente, el intérprete de Python se llama fácilmente de forma recursiva, y hay una interfaz estándar para llamar a una función de Python. (No me detendré en cómo llamar al analizador Python con una cadena particular como entrada — si está interesado, eche un vistazo a la implementación de la opción de línea de comando **`-c`** en `Modules/main.c` del código fuente de Python.)

Llamar a una función de Python es fácil. Primero, el programa Python debe de alguna manera pasar el objeto de función Python. Debe proporcionar una función (o alguna otra interfaz) para hacer esto. Cuando se llama a esta función, guarde un puntero en el objeto de la función Python (tenga cuidado de usar **`Py_INCREF()`**) En una variable global — o donde mejor le parezca. Por ejemplo, la siguiente función podría ser parte de una definición de módulo:

```
static PyObject *my_callback = NULL;

static PyObject *
my_set_callback(PyObject *dummy, PyObject *args)
{
    PyObject *result = NULL;
    PyObject *temp;

    if (PyArg_ParseTuple(args, "O:set_callback", &temp))
        if (!PyCallable_Check(temp)) {
            PyErr_SetString(PyExc_TypeError, "parameter
            return NULL;
        }
        Py_XINCREF(temp);         /* Add a reference to
        Py_XDECREF(my_callback);  /* Dispose of previous
        my_callback = temp;       /* Remember new callba
        /* Boilerplate to return "None" */
        Py_INCREF(Py_None);
        result = Py_None;
    }
    return result;
}
```

This function must be registered with the interpreter using the **`METH_VARARGS`** flag; this is described in section La tabla de métodos del módulo y la función de inicialización. The **`PyArg_ParseTuple()`** function and its arguments are documented in section Extracción de parámetros en funciones de extensión.

Las macros **`Py_XINCREF()`** y **`Py_XDECREF()`** incrementan/disminuyen el recuento de referencia de un objeto y son seguros en presencia de punteros `NULL` (pero tenga en cuenta que *temp* no lo hará ser `NULL` en este contexto). Más información sobre ellos en la sección Conteo de referencias.

Más tarde, cuando es hora de llamar a la función, llama a la función C **`PyObject_CallObject()`**. Esta función tiene dos argumentos, ambos punteros a objetos arbitrarios de Python: la función Python y la lista de argumentos. La lista de argumentos siempre debe ser un objeto de tupla, cuya longitud es el número de argumentos. Para llamar a la función Python sin argumentos, pase `NULL` o una tupla vacía; para llamarlo con un argumento, pasa una tupla singleton. **`Py_BuildValue()`** retorna una tupla cuando su cadena de formato consta de cero o más códigos de formato entre paréntesis. Por ejemplo:

```
int arg;
PyObject *arglist;
PyObject *result;
...
arg = 123;
...
/* Time to call the callback */
arglist = Py_BuildValue("(i)", arg);
```

```
result = PyObject_CallObject(my_callback, arglist);
Py_DECREF(arglist);
```

`PyObject_CallObject()` retorna un puntero de objeto Python: este es el valor de retorno de la función Python. **`PyObject_CallObject()`** es «recuento-referencia-neutral» con respecto a sus argumentos. En el ejemplo, se creó una nueva tupla para servir como lista de argumentos, a la cual se le llama **`Py_DECREF()`** inmediatamente después de la llamada **`PyObject_CallObject()`**.

El valor de retorno de **`PyObject_CallObject()`** es «nuevo»: o bien es un objeto nuevo o es un objeto existente cuyo recuento de referencias se ha incrementado. Por lo tanto, a menos que desee guardarlo en una variable global, debería de alguna manera **`Py_DECREF()`** el resultado, incluso (¡especialmente!) Si no está interesado en su valor.

Sin embargo, antes de hacer esto, es importante verificar que el valor de retorno no sea `NULL`. Si es así, la función de Python terminó generando una excepción. Si el código C que llamó **`PyObject_CallObject()`** se llama desde Python, ahora debería retornar una indicación de error a su llamador de Python, para que el intérprete pueda imprimir un seguimiento de la pila, o el código de Python que llama puede manejar la excepción. Si esto no es posible o deseable, la excepción se debe eliminar llamando a **`PyErr_Clear()`**. Por ejemplo:

```
if (result == NULL)
    return NULL; /* Pass error back */
...use result...
Py_DECREF(result);
```

Dependiendo de la interfaz deseada para la función de retrollamada de Python, es posible que también deba proporcionar una lista de argumentos para **`PyObject_CallObject()`**. En algunos casos, el programa Python también proporciona la lista de argumentos, a través de la misma interfaz que especificó la función de retrollamada. Luego se puede guardar y usar de la misma manera que el objeto de función. En otros casos, puede que tenga que construir una nueva tupla para pasarla como lista de argumentos.

La forma más sencilla de hacer esto es llamar a **`Py_BuildValue()`**. Por ejemplo, si desea pasar un código de evento integral, puede usar el siguiente código:

```
PyObject *arglist;
...
arglist = Py_BuildValue("(l)", eventcode);
result = PyObject_CallObject(my_callback, arglist);
Py_DECREF(arglist);
if (result == NULL)
    return NULL; /* Pass error back */
/* Here maybe use the result */
Py_DECREF(result);
```

¡Observe la ubicación de `Py_DECREF(arglist)` inmediatamente después de la llamada, antes de la verificación de errores! También tenga en cuenta que, estrictamente hablando, este código no está completo: **`Py_BuildValue()`** puede quedarse sin memoria, y esto debe verificarse.

También puede llamar a una función con argumentos de palabras clave utilizando **`PyObject_Call()`**, que admite argumentos y argumentos de palabras clave. Como en el ejemplo anterior, usamos **`Py_BuildValue()`** para construir el diccionario.

```
PyObject *dict;
...
dict = Py_BuildValue("{s:i}", "name", val);
result = PyObject_Call(my_callback, NULL, dict);
Py_DECREF(dict);
if (result == NULL)
    return NULL; /* Pass error back */
/* Here maybe use the result */
Py_DECREF(result);
```

1.7. Extracción de parámetros en funciones de extensión

La función **`PyArg_ParseTuple()`** se declara de la siguiente manera:

```
int PyArg_ParseTuple(PyObject *arg, const char *format,
```

El argumento *arg* debe ser un objeto de tupla que contenga una lista de argumentos pasada de Python a una función C. El argumento *format* debe ser una cadena de formato, cuya sintaxis se explica en Analizando argumentos y construyendo valores en el Manual de referencia de la API de Python/C. Los argumentos restantes deben ser direcciones de variables cuyo tipo está determinado por la cadena de formato.

Tenga en cuenta que si bien **`PyArg_ParseTuple()`** verifica que los argumentos de Python tengan los tipos requeridos, no puede verificar la validez de las direcciones de las variables C pasadas a la llamada: si comete errores allí, su código probablemente se bloqueará o al menos sobrescribir bits aleatorios en la memoria. ¡Así que ten cuidado!

Tenga en cuenta que las referencias de objetos de Python que se proporcionan a quien llama son referencias prestadas (*borrowed*); ¡no disminuya su recuento de referencias!

Algunas llamadas de ejemplo:

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>

int ok;
int i, j;
long k, l;
const char *s;
Py_ssize_t size;

ok = PyArg_ParseTuple(args, ""); /* No arguments */
    /* Python call: f() */

ok = PyArg_ParseTuple(args, "s", &s); /* A string */
    /* Possible Python call: f('whoops!') */

ok = PyArg_ParseTuple(args, "lls", &k, &l, &s); /* Two 
```

```
    /* Possible Python call: f(1, 2, 'three') */

ok = PyArg_ParseTuple(args, "(ii)s#", &i, &j, &s, &size)
    /* A pair of ints and a string, whose size is also r
    /* Possible Python call: f((1, 2), 'three') */

{
    const char *file;
    const char *mode = "r";
    int bufsize = 0;
    ok = PyArg_ParseTuple(args, "s|si", &file, &mode, &b
    /* A string, and optionally another string and an in
    /* Possible Python calls:
       f('spam')
       f('spam', 'w')
       f('spam', 'wb', 100000) */
}

{
    int left, top, right, bottom, h, v;
    ok = PyArg_ParseTuple(args, "((ii)(ii))(ii)",
             &left, &top, &right, &bottom, &h, &v);
    /* A rectangle and a point */
    /* Possible Python call:
       f(((0, 0), (400, 300)), (10, 10)) */
}

{
    Py_complex c;
    ok = PyArg_ParseTuple(args, "D:myfunction", &c);
    /* a complex, also providing a function name for err
    /* Possible Python call: myfunction(1+2j) */
}
```

1.8. Parámetros de palabras clave para funciones de extensión

La función `PyArg_ParseTupleAndKeywords()` se declara de la siguiente manera:

```
int PyArg_ParseTupleAndKeywords(PyObject *arg, PyObject
                                const char *format, char
```

Los parámetros *arg* y *format* son idénticos a los de la función **`PyArg_ParseTuple()`**. El parámetro *kwdict* es el diccionario de palabras clave recibidas como tercer parámetro del tiempo de ejecución de Python. El parámetro *kwlist* es una lista de cadenas terminadas en `NULL` que identifican los parámetros; los nombres se corresponden con la información de tipo de *format* de izquierda a derecha. En caso de éxito, **`PyArg_ParseTupleAndKeywords()`** retorna verdadero; de lo contrario, retorna falso y genera una excepción apropiada.

Nota

¡Las tuplas anidadas no se pueden analizar al usar argumentos de palabras clave! Los parámetros de palabras clave pasados que no están presentes en la *kwlist* provocarán que se genere **`TypeError`**.

Aquí hay un módulo de ejemplo que usa palabras clave, basado en un ejemplo de *Geoff Philbrick (philbrick@hks.com)*:

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>

static PyObject *
keywdarg_parrot(PyObject *self, PyObject *args, PyObject
{
    int voltage;
    const char *state = "a stiff";
    const char *action = "voom";
    const char *type = "Norwegian Blue";

    static char *kwlist[] = {"voltage", "state", "action

    if (!PyArg_ParseTupleAndKeywords(args, keywds, "i|ss
                                                &voltage, &state, &
        return NULL;

    printf("-- This parrot wouldn't %s if you put %i Vol
           action, voltage);
    printf("-- Lovely plumage, the %s -- It's %s!\n", ty

    Py_RETURN_NONE;
}

static PyMethodDef keywdarg_methods[] = {
    /* The cast of the function is necessary since PyCFu
     * only take two PyObject* parameters, and keywdarg_
     * three.
     */
    {"parrot", (PyCFunction)(void(*)(void))keywdarg_parr
     "Print a lovely skit to standard output."},
    {NULL, NULL, 0, NULL}   /* sentinel */
};

static struct PyModuleDef keywdarg_module = {
    .m_base = PyModuleDef_HEAD_INIT,
    .m_name = "keywdarg",
    .m_size = 0,
    .m_methods = keywdarg_methods,
};

PyMODINIT_FUNC
PyInit_keywdarg(void)
{
    return PyModuleDef_Init(&keywdarg_module);
}
```

1.9. Construyendo valores arbitrarios

Esta función es la contraparte de `PyArg_ParseTuple()`. Se declara de la siguiente manera:

```
PyObject *Py_BuildValue(const char *format, ...);
```

Reconoce un conjunto de unidades de formato similares a las reconocidas por **`PyArg_ParseTuple()`**, pero los argumentos (que son de entrada a la función, no de salida) no deben ser punteros, solo valores. Retorna un nuevo objeto Python, adecuado para regresar de una función C llamada desde Python.

Una diferencia con **`PyArg_ParseTuple()`**: mientras que este último requiere que su primer argumento sea una tupla (ya que las listas de argumentos de Python siempre se representan como tuplas internamente), **`Py_BuildValue()`** no siempre construye una tupla . Construye una tupla solo si su cadena de formato contiene dos o más unidades de formato. Si la cadena de formato está vacía, retorna `None`; si contiene exactamente una unidad de formato, retorna el objeto que describa esa unidad de formato. Para forzarlo a retornar una tupla de tamaño 0 o uno, agregar paréntesis a la cadena de formato.

Ejemplos (a la izquierda la llamada, a la derecha el valor de Python resultante):

```
Py_BuildValue("")                              None
Py_BuildValue("i", 123)                        123
Py_BuildValue("iii", 123, 456, 789)            (123, 456, 789)
Py_BuildValue("s", "hello")                    'hello'
Py_BuildValue("y", "hello")                    b'hello'
Py_BuildValue("ss", "hello", "world")          ('hello', 'worl
Py_BuildValue("s#", "hello", 4)                'hell'
Py_BuildValue("y#", "hello", 4)                b'hell'
Py_BuildValue("()")                            ()
Py_BuildValue("(i)", 123)                      (123,)
Py_BuildValue("(ii)", 123, 456)                (123, 456)
Py_BuildValue("(i,i)", 123, 456)               (123, 456)
Py_BuildValue("[i,i]", 123, 456)               [123, 456]
Py_BuildValue("{s:i,s:i}",
              "abc", 123, "def", 456)          {'abc': 123, 'c
Py_BuildValue("((ii)(ii)) (ii)",
              1, 2, 3, 4, 5, 6)                (((1, 2), (3, 4
```

1.10. Conteo de referencias

En lenguajes como C o C++, el programador es responsable de la asignación dinámica y la desasignación de memoria en el montón. En C, esto se hace usando las funciones **`malloc()`** y **`free()`**. En C++, los operadores `new` y `delete` se usan esencialmente con el mismo significado y restringiremos la siguiente discusión al caso C.

Every block of memory allocated with **`malloc()`** should eventually be returned to the pool of available memory by exactly one call to **`free()`**. It is important to call **`free()`** at the right time. If a block's address is forgotten but **`free()`** is not called for it, the memory it occupies cannot be reused until the program terminates. This is called a *memory leak*. On the other hand, if a program calls **`free()`** for a block and then continues to use the block, it creates a conflict with reuse of the block through another **`malloc()`** call. This is called *using freed memory*. It has the same bad consequences as referencing uninitialized data — core dumps, wrong results, mysterious crashes.

Las causas comunes de pérdidas de memoria son rutas inusuales a través del código. Por ejemplo, una función puede asignar un bloque de memoria, hacer algunos cálculos y luego liberar el bloque nuevamente. Ahora, un cambio en los requisitos de la función puede agregar una prueba al cálculo que detecta una condición de error y puede regresar prematuramente de la función. Es fácil olvidar liberar el bloque de memoria asignado al tomar esta salida prematura, especialmente cuando se agrega más tarde al código. Tales filtraciones, una vez introducidas, a menudo pasan desapercibidas durante mucho tiempo: la salida del error se toma solo en una pequeña fracción de todas las llamadas, y la mayoría de las máquinas modernas tienen mucha memoria virtual, por lo que la filtración solo se hace evidente en un proceso de larga ejecución que usa la función de fugas con frecuencia. Por lo tanto, es importante evitar que se produzcan fugas mediante una convención o estrategia de codificación que minimice este tipo de errores.

Dado que Python hace un uso intensivo de **`malloc()`** y **`free()`**, necesita una estrategia para evitar pérdidas de memoria, así como el uso de memoria liberada. El método elegido se llama *recuento de* *referencias*. El principio es simple: cada objeto contiene un contador, que se incrementa cuando se almacena una referencia al objeto en algún lugar, y que se reduce cuando se elimina una referencia al mismo. Cuando el contador llega a cero, la última referencia al objeto se ha eliminado y el objeto se libera.

Una estrategia alternativa se llama *recolección automática de basura*. (A veces, el recuento de referencias también se conoce como una estrategia de recolección de basura, de ahí mi uso de «automático» para distinguir los dos). La gran ventaja de la recolección automática de basura es que el usuario no necesita llamar a **`free()`** explícitamente. (Otra ventaja afirmada es una mejora en la velocidad o el uso de la memoria; sin embargo, esto no es un hecho difícil). La desventaja es que para C, no hay un recolector de basura automático verdaderamente portátil, mientras que el conteo de referencias se puede implementar de forma portátil (siempre que las funciones **`malloc()`** y **`free()`** están disponibles — que garantiza el estándar C). Tal vez algún día un recolector de basura automático lo suficientemente portátil estará disponible para C. Hasta entonces, tendremos que vivir con recuentos de referencia.

Si bien Python utiliza la implementación tradicional de conteo de referencias, también ofrece un detector de ciclos que funciona para detectar ciclos de referencia. Esto permite que las aplicaciones no se preocupen por crear referencias circulares directas o indirectas; Estas son las debilidades de la recolección de basura implementada utilizando solo el conteo de referencias. Los ciclos de referencia consisten en objetos que contienen referencias (posiblemente indirectas) a sí mismos, de modo que cada objeto en el ciclo tiene un recuento de referencias que no es cero. Las implementaciones típicas de recuento de referencias no pueden recuperar la memoria que pertenece a algún objeto en un ciclo de referencia, o referenciada a partir de los objetos en el ciclo, a pesar de que no hay más referencias al ciclo en sí.

El detector de ciclos es capaz de detectar ciclos basura y puede recuperarlos. El módulo `gc` expone una forma de ejecutar el detector (la función `collect()`), así como interfaces de configuración y la posibilidad de desactivar el detector en tiempo de ejecución.

1.10.1. Conteo de referencias en Python

Hay dos macros, `Py_INCREF(x)` y `Py_DECREF(x)`, que manejan el incremento y la disminución del recuento de referencias. **`Py_DECREF()`** también libera el objeto cuando el recuento llega a cero. Por flexibilidad, no llama a **`free()`** directamente — más bien, realiza una llamada a través de un puntero de función en el objeto *type object*. Para este propósito (y otros), cada objeto también contiene un puntero a su objeto de tipo.

La gran pregunta ahora permanece: ¿cuándo usar `Py_INCREF(x)` y `Py_DECREF(x)`? Primero introduzcamos algunos términos. Nadie «posee» un objeto; sin embargo, puede *poseer una referencia* a un objeto. El recuento de referencias de un objeto ahora se define como el número de referencias que posee. El propietario de una referencia es responsable de llamar a **`Py_DECREF()`** cuando la referencia ya no es necesaria. La propiedad de una referencia puede ser transferida. Hay tres formas de deshacerse de una referencia de propiedad: pasarla, almacenarla o llamar a **`Py_DECREF()`**. Olvidar deshacerse de una referencia de propiedad crea una pérdida de memoria.

También es posible *tomar prestada* [2] una referencia a un objeto. El prestatario de una referencia no debe llamar a **`Py_DECREF()`**. El prestatario no debe retener el objeto por más tiempo que el propietario del cual fue prestado. El uso de una referencia prestada después de que el propietario la haya eliminado corre el riesgo de usar memoria liberada y debe evitarse por completo [3].

La ventaja de pedir prestado sobre tener una referencia es que no necesita ocuparse de disponer de la referencia en todas las rutas posibles a través del código — en otras palabras, con una referencia prestada no corre el riesgo de fugas cuando se toma una salida prematura. La desventaja de pedir prestado sobre la posesión es que hay algunas situaciones sutiles en las que, en un código aparentemente correcto, una referencia prestada se puede usar después de que el propietario del que se tomó prestado la haya eliminado.

Una referencia prestada se puede cambiar en una referencia de propiedad llamando a `Py_INCREF()`. Esto no afecta el estado del propietario del cual se tomó prestada la referencia: crea una nueva referencia de propiedad y otorga responsabilidades completas al propietario (el nuevo propietario debe disponer de la referencia correctamente, así como el propietario anterior).

1.10.2. Reglas de propiedad

Cuando una referencia de objeto se pasa dentro o fuera de una función, es parte de la especificación de la interfaz de la función si la propiedad se transfiere con la referencia o no.

La mayoría de las funciones que retornan una referencia a un objeto pasan de propiedad con la referencia. En particular, todas las funciones cuya función es crear un nuevo objeto, como `PyLong_FromLong()` y `Py_BuildValue()`, pasan la propiedad al receptor. Incluso si el objeto no es realmente nuevo, aún recibirá la propiedad de una nueva referencia a ese objeto. Por ejemplo, `PyLong_FromLong()` mantiene un caché de valores populares y puede retornar una referencia a un elemento en caché.

Muchas funciones que extraen objetos de otros objetos también transfieren la propiedad con la referencia, por ejemplo `PyObject_GetAttrString()`. Sin embargo, la imagen es menos clara aquí, ya que algunas rutinas comunes son excepciones: `PyTuple_GetItem()`, `PyList_GetItem()`, `PyDict_GetItem()`, y `PyDict_GetItemString()` todas las referencias retornadas que tomaste prestadas de la tupla, lista o diccionario.

La función `PyImport_AddModule()` también retorna una referencia prestada, aunque en realidad puede crear el objeto que retorna: esto es posible porque una referencia de propiedad del objeto se almacena en `sys.modules`.

Cuando pasa una referencia de objeto a otra función, en general, la función toma prestada la referencia de usted — si necesita almacenarla, usará `Py_INCREF()` para convertirse en un propietario independiente. Hay exactamente dos excepciones importantes a esta regla: `PyTuple_SetItem()` y `PyList_SetItem()`. Estas funciones se hacen cargo de la propiedad del artículo que se les pasa, ¡incluso si fallan! (Tenga en cuenta que **`PyDict_SetItem()`** y sus amigos no se hacen cargo de la propiedad — son «normales»)

Cuando se llama a una función C desde Python, toma de la persona que llama referencias a sus argumentos. Quien llama posee una referencia al objeto, por lo que la vida útil de la referencia prestada está garantizada hasta que la función regrese. Solo cuando dicha referencia prestada debe almacenarse o transmitirse, debe convertirse en una referencia propia llamando a **`Py_INCREF()`**.

La referencia de objeto retornada desde una función C que se llama desde Python debe ser una referencia de propiedad: la propiedad se transfiere de la función a su llamador.

1.10.3. Hielo delgado

Hay algunas situaciones en las que el uso aparentemente inofensivo de una referencia prestada puede generar problemas. Todo esto tiene que ver con invocaciones implícitas del intérprete, lo que puede hacer que el propietario de una referencia se deshaga de él.

El primer y más importante caso que debe conocer es el uso de **`Py_DECREF()`** en un objeto no relacionado mientras toma prestada una referencia a un elemento de la lista. Por ejemplo:

```
void
bug(PyObject *list)
{
    PyObject *item = PyList_GetItem(list, 0);

    PyList_SetItem(list, 1, PyLong_FromLong(0L));
    PyObject_Print(item, stdout, 0); /* BUG! */
}
```

Esta función primero toma prestada una referencia a `list[0]`, luego reemplaza `list[1]` con el valor `0`, y finalmente imprime la referencia prestada. Parece inofensivo, ¿verdad? ¡Pero no lo es!

Let’s follow the control flow into **`PyList_SetItem()`**. The list owns references to all its items, so when item 1 is replaced, it has to

dispose of the original item 1. Now let's suppose the original item 1 was an instance of a user-defined class, and let's further suppose that the class defined a `__del__()` method. If this class instance has a reference count of 1, disposing of it will call its `__del__()` method. Internally, `PyList_SetItem()` calls `Py_DECREF()` on the replaced item, which invokes replaced item's corresponding `tp_dealloc` function. During deallocation, `tp_dealloc` calls `tp_finalize`, which is mapped to the `__del__()` method for class instances (see PEP 442 [https://peps.python.org/pep-0442/]). This entire sequence happens synchronously within the `PyList_SetItem()` call.

Since it is written in Python, the `__del__()` method can execute arbitrary Python code. Could it perhaps do something to invalidate the reference to `item` in `bug()`? You bet! Assuming that the list passed into `bug()` is accessible to the `__del__()` method, it could execute a statement to the effect of `del list[0]`, and assuming this was the last reference to that object, it would free the memory associated with it, thereby invalidating `item`.

La solución, una vez que conoce el origen del problema, es fácil: incremente temporalmente el recuento de referencia. La versión correcta de la función dice:

```
void
no_bug(PyObject *list)
{
    PyObject *item = PyList_GetItem(list, 0);

    Py_INCREF(item);
    PyList_SetItem(list, 1, PyLong_FromLong(0L));
    PyObject_Print(item, stdout, 0);
    Py_DECREF(item);
}
```

This is a true story. An older version of Python contained variants of this bug and someone spent a considerable amount of time in a C debugger to figure out why his `__del__()` methods would fail…

The second case of problems with a borrowed reference is a variant

involving threads. Normally, multiple threads in the Python interpreter can't get in each other's way, because there is a global lock protecting Python's entire object space. However, it is possible to temporarily release this lock using the macro **`Py_BEGIN_ALLOW_THREADS`**, and to re-acquire it using **`Py_END_ALLOW_THREADS`**. This is common around blocking I/O calls, to let other threads use the processor while waiting for the I/O to complete. Obviously, the following function has the same problem as the previous one:

```
void
bug(PyObject *list)
{
    PyObject *item = PyList_GetItem(list, 0);
    Py_BEGIN_ALLOW_THREADS
    ...some blocking I/O call...
    Py_END_ALLOW_THREADS
    PyObject_Print(item, stdout, 0); /* BUG! */
}
```

1.10.4. Punteros NULL

En general, las funciones que toman referencias de objetos como argumentos no esperan que les pase los punteros `NULL`, y volcará el núcleo (o causará volcados de núcleo posteriores) si lo hace. Las funciones que retornan referencias a objetos generalmente retornan `NULL` solo para indicar que ocurrió una excepción. La razón para no probar los argumentos `NULL` es que las funciones a menudo pasan los objetos que reciben a otra función — si cada función probara `NULL`, habría muchas pruebas redundantes y el código correría más lentamente.

Es mejor probar `NULL` solo en «source:» cuando se recibe un puntero que puede ser `NULL`, por ejemplo, de **`malloc()`** o de una función que puede plantear una excepción.

Las macros **`Py_INCREF()`** y **`Py_DECREF()`** no comprueban los punteros `NULL` — sin embargo, sus variantes **`Py_XINCREF()`** y **`Py_XDECREF()`** lo hacen.

Las macros para verificar un tipo de objeto en particular (`Pytype_Check()`) no verifican los punteros `NULL` — nuevamente, hay mucho código que llama a varios de estos en una fila para probar un objeto contra varios tipos esperados diferentes, y esto generaría pruebas redundantes. No hay variantes con comprobación `NULL`.

El mecanismo de llamada a la función C garantiza que la lista de argumentos pasada a las funciones C (`args` en los ejemplos) nunca sea `NULL` — de hecho, garantiza que siempre sea una tupla [4].

Es un error grave dejar que un puntero `NULL` «escape» al usuario de Python.

1.11. Escribiendo extensiones en C++

Es posible escribir módulos de extensión en C++. Se aplican algunas restricciones. Si el compilador de C compila y vincula el programa principal (el intérprete de Python), no se pueden usar objetos globales o estáticos con constructores. Esto no es un problema si el programa principal está vinculado por el compilador de C++. Las funciones que serán llamadas por el intérprete de Python (en particular, las funciones de inicialización del módulo) deben declararse usando `extern "C"`. No es necesario encerrar los archivos de encabezado de Python en `extern "C" {...}` — ya usan este formulario si el símbolo `__cplusplus` está definido (todos los compiladores recientes de C++ definen este símbolo) .

1.12. Proporcionar una API C para un módulo de extensión

Muchos módulos de extensión solo proporcionan nuevas funciones y tipos para ser utilizados desde Python, pero a veces el código en un módulo de extensión puede ser útil para otros módulos de extensión. Por ejemplo, un módulo de extensión podría implementar un tipo de «colección» que funciona como listas sin orden. Al igual que el tipo de lista Python estándar tiene una API C que permite a los módulos de extensión crear y manipular listas, este nuevo tipo de colección debe tener un conjunto de funciones C para la manipulación directa desde otros módulos de extensión.

A primera vista, esto parece fácil: simplemente escriba las funciones (sin declararlas `static`, por supuesto), proporcione un archivo de encabezado apropiado y documente la API de C. Y, de hecho, esto funcionaría si todos los módulos de extensión siempre estuvieran vinculados estáticamente con el intérprete de Python. Sin embargo, cuando los módulos se usan como bibliotecas compartidas, los símbolos definidos en un módulo pueden no ser visibles para otro módulo. Los detalles de visibilidad dependen del sistema operativo; algunos sistemas usan un espacio de nombres global para el intérprete de Python y todos los módulos de extensión (Windows, por ejemplo), mientras que otros requieren una lista explícita de símbolos importados en el momento del enlace del módulo (AIX es un ejemplo) u ofrecen una variedad de estrategias diferentes (la mayoría Unices). E incluso si los símbolos son visibles a nivel mundial, ¡el módulo cuyas funciones uno desea llamar podría no haberse cargado todavía!

Por lo tanto, la portabilidad requiere no hacer suposiciones sobre la visibilidad del símbolo. Esto significa que todos los símbolos en los módulos de extensión deben declararse `static`, excepto la función de inicialización del módulo, para evitar conflictos de nombres con otros módulos de extensión (como se discutió en la sección La tabla de métodos del módulo y la función de inicialización). Y significa que los símbolos que *deberían* ser accesibles desde otros módulos de extensión deben exportarse de una manera diferente.

Python proporciona un mecanismo especial para pasar información de nivel C (punteros) de un módulo de extensión a otro: cápsulas. Una cápsula es un tipo de datos de Python que almacena un puntero (void*). Solo se puede crear y acceder a las cápsulas a través de su API C, pero se pueden pasar como cualquier otro objeto de Python. En particular, se pueden asignar a un nombre en el espacio de nombres de un módulo de extensión. Otros módulos de extensión pueden importar este módulo, recuperar el valor de este nombre y luego recuperar el puntero de la Cápsula.

Hay muchas formas en que las Cápsulas se pueden usar para exportar la API de C de un módulo de extensión. Cada función podría tener su propia cápsula, o todos los punteros de API C podrían almacenarse en una matriz cuya dirección se publica en una cápsula. Y las diversas tareas de almacenamiento y recuperación de los punteros se pueden distribuir de diferentes maneras entre el módulo que proporciona el código y los módulos del cliente.

Cualquiera que sea el método que elija, es importante nombrar correctamente sus Cápsulas. La función `PyCapsule_New()` toma un parámetro de nombre (const char*); se le permite pasar un nombre `NULL`, pero le recomendamos encarecidamente que especifique un nombre. Las Cápsulas correctamente nombradas brindan un grado de seguridad de tipo de tiempo de ejecución; no hay una forma factible de distinguir una Cápsula sin nombre de otra.

En particular, las cápsulas utilizadas para exponer las API de C deben recibir un nombre siguiendo esta convención:

```
modulename.attributename
```

La función de conveniencia `PyCapsule_Import()` facilita la carga de una API C proporcionada a través de una cápsula, pero solo si el nombre de la cápsula coincide con esta convención. Este comportamiento brinda a los usuarios de C API un alto grado de certeza de que la Cápsula que cargan contiene la API de C correcta.

El siguiente ejemplo demuestra un enfoque que pone la mayor parte de la carga sobre el escritor del módulo de exportación, que es apropiado para los módulos de biblioteca de uso común. Almacena todos los punteros de la API de C (¡solo uno en el ejemplo!) en un arreglo de punteros void que se convierte en el valor de una Cápsula. El archivo de encabezado correspondiente al módulo proporciona una macro que se encarga de importar el módulo y recuperar sus punteros C API; los módulos de cliente solo tienen que llamar a esta macro antes de acceder a la API de C.

The exporting module is a modification of the **`spam`** module from section Un ejemplo simple. The function **`spam.system()`** does not call the C library function **`system()`** directly, but a function **`PySpam_System()`**, which would of course do something more complicated in reality (such as adding «spam» to every command).

This function **`PySpam_System()`** is also exported to other extension modules.

The function **`PySpam_System()`** is a plain C function, declared `static` like everything else:

```
static int
PySpam_System(const char *command)
{
    return system(command);
}
```

The function **`spam_system()`** is modified in a trivial way:

```
static PyObject *
spam_system(PyObject *self, PyObject *args)
{
    const char *command;
    int sts;

    if (!PyArg_ParseTuple(args, "s", &command))
        return NULL;
    sts = PySpam_System(command);
    return PyLong_FromLong(sts);
}
```

Al comienzo del módulo, justo después de la línea:

```
#include <Python.h>
```

se deben agregar dos líneas más:

```
#define SPAM_MODULE
#include "spammodule.h"
```

The `#define` is used to tell the header file that it is being included in the exporting module, not a client module. Finally, the module's **`mod_exec`** function must take care of initializing the C API pointer array:

```
static int
```

```
spam_module_exec(PyObject *m)
{
    static void *PySpam_API[PySpam_API_pointers];
    PyObject *c_api_object;

    /* Initialize the C API pointer array */
    PySpam_API[PySpam_System_NUM] = (void *)PySpam_Syste

    /* Create a Capsule containing the API pointer array
    c_api_object = PyCapsule_New((void *)PySpam_API, "sp

    if (PyModule_Add(m, "_C_API", c_api_object) < 0) {
        return -1;
    }

    return 0;
}
```

Note that `PySpam_API` is declared `static`; otherwise the pointer array would disappear when **`PyInit_spam()`** terminates!

La mayor parte del trabajo está en el archivo de encabezado `spammodule.h`, que se ve así:

```
#ifndef Py_SPAMMODULE_H
#define Py_SPAMMODULE_H
#ifdef __cplusplus
extern "C" {
#endif

/* Header file for spammodule */

/* C API functions */
#define PySpam_System_NUM 0
#define PySpam_System_RETURN int
#define PySpam_System_PROTO (const char *command)

/* Total number of C API pointers */
#define PySpam_API_pointers 1
```

```
#ifdef SPAM_MODULE
/* This section is used when compiling spammodule.c */

static PySpam_System_RETURN PySpam_System PySpam_System_

#else
/* This section is used in modules that use spammodule'

static void **PySpam_API;

#define PySpam_System \
 (*(PySpam_System_RETURN (*)PySpam_System_PROTO) PySpam_

/* Return -1 on error, 0 on success.
 * PyCapsule_Import will set an exception if there's an
 */
static int
import_spam(void)
{
    PySpam_API = (void **)PyCapsule_Import("spam._C_API"
    return (PySpam_API != NULL) ? 0 : -1;
}

#endif

#ifdef __cplusplus
}
#endif

#endif /* !defined(Py_SPAMMODULE_H) */
```

All that a client module must do in order to have access to the function **`PySpam_System()`** is to call the function (or rather macro) **`import_spam()`** in its **`mod_exec`** function:

```
static int
client_module_exec(PyObject *m)
```

```
{
    if (import_spam() < 0) {
        return -1;
    }
    /* additional initialization can happen here */
    return 0;
}
```

La principal desventaja de este enfoque es que el archivo `spammodule.h` es bastante complicado. Sin embargo, la estructura básica es la misma para cada función que se exporta, por lo que solo se debe aprender una vez.

Finalmente, debe mencionarse que las cápsulas ofrecen una funcionalidad adicional, que es especialmente útil para la asignación de memoria y la desasignación del puntero almacenado en una cápsula. Los detalles se describen en el Manual de referencia de Python/C API en la sección Cápsulas y en la implementación de Capsules (archivos `Include/pycapsule.h` y `Objects/pycapsule.c` en la distribución del código fuente de Python).

Notas al pie de página

Ya existe una interfaz para esta función en el módulo estándar **`os`** — se eligió como un ejemplo simple y directo.

La metáfora de «pedir prestado» una referencia no es completamente correcta: el propietario todavía tiene una copia de la referencia.

¡Comprobar que el recuento de referencia es al menos 1 **no funciona** — el recuento de referencia en sí podría estar en la memoria liberada y, por lo tanto, puede reutilizarse para otro objeto!

Estas garantías no se cumplen cuando utiliza la convención de llamadas de estilo «antiguo», que todavía se encuentra en muchos códigos existentes.

[1] [2] [3] [4]

2. Definición de tipos de extensión: Tutorial

Python le permite al escritor de un módulo de extensión C definir nuevos tipos que pueden ser manipulados desde el código Python, al igual que los tipos incorporados `str` y `list`. El código para todos los tipos de extensión sigue un patrón, pero hay algunos detalles que debe comprender antes de comenzar. Este documento es una introducción suave al tema.

2.1. Lo básico

The CPython runtime sees all Python objects as variables of type PyObject*, which serves as a «base type» for all Python objects. The `PyObject` structure itself only contains the object's reference count and a pointer to the object's «type object». This is where the action is; the type object determines which (C) functions get called by the interpreter when, for instance, an attribute gets looked up on an object, a method called, or it is multiplied by another object. These C functions are called «type methods».

Por lo tanto, si desea definir un nuevo tipo de extensión, debe crear un nuevo objeto de tipo.

This sort of thing can only be explained by example, so here's a minimal, but complete, module that defines a new type named `Custom` inside a C extension module `custom`:

Nota

Lo que estamos mostrando aquí es la forma tradicional de definir tipos de extensión *estáticos*. Debe ser adecuado para la mayoría de los usos. La API de C también permite definir tipos de extensiones asignadas en el montón utilizando la función

`PyType_FromSpec()`, que no se trata en este tutorial.

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>

typedef struct {
    PyObject_HEAD
    /* Type-specific fields go here. */
} CustomObject;

static PyTypeObject CustomType = {
    .ob_base = PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "custom.Custom",
    .tp_doc = PyDoc_STR("Custom objects"),
    .tp_basicsize = sizeof(CustomObject),
    .tp_itemsize = 0,
    .tp_flags = Py_TPFLAGS_DEFAULT,
    .tp_new = PyType_GenericNew,
};

static int
custom_module_exec(PyObject *m)
{
    if (PyType_Ready(&CustomType) < 0) {
        return -1;
    }

    if (PyModule_AddObjectRef(m, "Custom", (PyObject *)
        return -1;
    }

    return 0;
}

static PyModuleDef_Slot custom_module_slots[] = {
    {Py_mod_exec, custom_module_exec},
    // Just use this while using static types
    {Py_mod_multiple_interpreters, Py_MOD_MULTIPLE_INTER
    {0, NULL}
};

static PyModuleDef custom_module = {
    .m_base = PyModuleDef_HEAD_INIT,
    .m_name = "custom",
    .m_doc = "Example module that creates an extension t
    .m_size = 0,
    .m_slots = custom_module_slots,
};

PyMODINIT_FUNC
PyInit_custom(void)
{
    return PyModuleDef_Init(&custom_module);
}
```

Ahora, eso es bastante para asimilar a la vez, pero espero que los fragmentos le resulten familiares del capítulo anterior. Este archivo define tres cosas:

1. What a **`Custom` object** contains: this is the `CustomObject` struct, which is allocated once for each **`Custom`** instance.
2. How the **`Custom` type** behaves: this is the `CustomType` struct, which defines a set of flags and function pointers that the interpreter inspects when specific operations are requested.
3. How to define and execute the **`custom`** module: this is the `PyInit_custom` function and the associated `custom_module` struct for defining the module, and the `custom_module_exec` function to set up a fresh module object.

La primera parte es:

```
typedef struct {
    PyObject_HEAD
} CustomObject;
```

This is what a Custom object will contain. `PyObject_HEAD` is mandatory at the start of each object struct and defines a field called `ob_base` of type **`PyObject`**, containing a pointer to a type object and a reference count (these can be accessed using the macros **`Py_TYPE`** and **`Py_REFCNT`** respectively). The reason for the macro is to abstract away the layout and to enable additional fields in debug builds.

Nota

No hay punto y coma (;) arriba después de la macro **`PyObject_HEAD`**. Tenga cuidado de agregar uno por accidente: algunos compiladores se quejarán.

Por supuesto, los objetos generalmente almacenan datos adicionales además del estándar `PyObject_HEAD` repetitivo; por ejemplo, aquí está la definición de puntos flotantes del estándar de Python:

```
typedef struct {
    PyObject_HEAD
    double ob_fval;
} PyFloatObject;
```

La segunda parte es la definición del tipo de objeto.

```
static PyTypeObject CustomType = {
    .ob_base = PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "custom.Custom",
    .tp_doc = PyDoc_STR("Custom objects"),
    .tp_basicsize = sizeof(CustomObject),
    .tp_itemsize = 0,
    .tp_flags = Py_TPFLAGS_DEFAULT,
    .tp_new = PyType_GenericNew,
};
```

Nota

Recomendamos utilizar los inicializadores designados al estilo C99 como se indica arriba, para evitar enumerar todos los campos **`PyTypeObject`** que no le interesan y también para evitar preocuparse por el orden de declaración de los campos.

La definición real de **`PyTypeObject`** en `object.h` tiene muchos más campos que la definición anterior. El compilador de C rellenará los campos restantes con ceros, y es una práctica común no especificarlos explícitamente a menos que los necesite.

Lo vamos a separar, un campo a la vez:

```
.ob_base = PyVarObject_HEAD_INIT(NULL, 0)
```

Esta línea es obligatoria para inicializar el campo `ob_base` mencionado anteriormente.

```
.tp_name = "custom.Custom",
```

El nombre de nuestro tipo. Esto aparecerá en la representación textual predeterminada de nuestros objetos y en algunos mensajes de error, por ejemplo:

```
>>> "" + custom.Custom()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "custom.Custom"
```

Note that the name is a dotted name that includes both the module name and the name of the type within the module. The module in this case is **`custom`** and the type is **`Custom`**, so we set the type name to **`custom.Custom`**. Using the real dotted import path is important to make your type compatible with the **`pydoc`** and **`pickle`** modules.

```
.tp_basicsize = sizeof(CustomObject),
.tp_itemsize = 0,
```

This is so that Python knows how much memory to allocate when creating new **`Custom`** instances. **`tp_itemsize`** is only used for variable-sized objects and should otherwise be zero.

Nota

> If you want your type to be subclassable from Python, and your type has the same `tp_basicsize` as its base type, you may have problems with multiple inheritance. A Python subclass of your type will have to list your type first in its `__bases__`, or else it will not be able to call your type's `__new__()` method without getting an error. You can avoid this problem by ensuring that your type has a larger value for `tp_basicsize` than its base type does. Most of the time, this will be true anyway, because either your base type will be `object`, or else you will be adding data members to your base type, and therefore increasing its size.

We set the class flags to `Py_TPFLAGS_DEFAULT`.

```
.tp_flags = Py_TPFLAGS_DEFAULT,
```

Todos los tipos deben incluir esta constante en sus banderas. Habilita todos los miembros definidos hasta al menos Python 3.3. Si necesita más miembros, necesitará O (*OR*) las banderas correspondientes.

Proporcionamos una cadena de documentos para el tipo en `tp_doc`.

```
.tp_doc = PyDoc_STR("Custom objects"),
```

To enable object creation, we have to provide a `tp_new` handler. This is the equivalent of the Python method `__new__()`, but has to be specified explicitly. In this case, we can just use the default implementation provided by the API function `PyType_GenericNew()`.

```
.tp_new = PyType_GenericNew,
```

Everything else in the file should be familiar, except for some code in `custom_module_exec()`:

```
if (PyType_Ready(&CustomType) < 0) {
    return -1;
}
```

This initializes the **`Custom`** type, filling in a number of members to the appropriate default values, including **`ob_type`** that we initially set to `NULL`.

```
if (PyModule_AddObjectRef(m, "Custom", (PyObject *) &Cus
    return -1;
}
```

This adds the type to the module dictionary. This allows us to create **`Custom`** instances by calling the **`Custom`** class:

```
>>> import custom
>>> mycustom = custom.Custom()
```

That's it! All that remains is to build it; put the above code in a file called `custom.c`,

```
[build-system]
requires = ["setuptools"]
build-backend = "setuptools.build_meta"

[project]
name = "custom"
version = "1"
```

in a file called `pyproject.toml`, and

```
from setuptools import Extension, setup
setup(ext_modules=[Extension("custom", ["custom.c"])])
```

en un archivo llamado `setup.py`; luego escribiendo

```
$ python -m pip install .
```

in a shell should produce a file `custom.so` in a subdirectory and install it; now fire up Python — you should be able to `import custom` and play around with `Custom` objects.

Eso no fue tan difícil, ¿verdad?

Por supuesto, el tipo personalizado actual es bastante poco

interesante. No tiene datos y no hace nada. Ni siquiera se puede subclasificar.

2.2. Agregar datos y métodos al ejemplo básico

Let’s extend the basic example to add some data and methods. Let’s also make the type usable as a base class. We’ll create a new module, **`custom2`** that adds these capabilities:

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>
#include <stddef.h> /* for offsetof() */

typedef struct {
    PyObject_HEAD
    PyObject *first; /* first name */
    PyObject *last;  /* last name */
    int number;
} CustomObject;

static void
Custom_dealloc(PyObject *op)
{
    CustomObject *self = (CustomObject *) op;
    Py_XDECREF(self->first);
    Py_XDECREF(self->last);
    Py_TYPE(self)->tp_free(self);
}

static PyObject *
Custom_new(PyTypeObject *type, PyObject *args, PyObject
{
    CustomObject *self;
    self = (CustomObject *) type->tp_alloc(type, 0);
    if (self != NULL) {
        self->first = Py_GetConstant(Py_CONSTANT_EMPTY_S
        if (self->first == NULL) {
            Py_DECREF(self);
            return NULL;
        }
        self->last = Py_GetConstant(Py_CONSTANT_EMPTY_ST
        if (self->last == NULL) {
            Py_DECREF(self);
            return NULL;
        }
        self->number = 0;
    }
    return (PyObject *) self;
}

static int
Custom_init(PyObject *op, PyObject *args, PyObject *kwds
{
    CustomObject *self = (CustomObject *) op;
    static char *kwlist[] = {"first", "last", "number",
    PyObject *first = NULL, *last = NULL;

    if (!PyArg_ParseTupleAndKeywords(args, kwds, "|OOi",
                                     &first, &last,
                                     &self->number))
        return -1;

    if (first) {
        Py_XSETREF(self->first, Py_NewRef(first));
    }
    if (last) {
        Py_XSETREF(self->last, Py_NewRef(last));
    }
    return 0;
}

static PyMemberDef Custom_members[] = {
    {"first", Py_T_OBJECT_EX, offsetof(CustomObject, fir
     "first name"},
    {"last", Py_T_OBJECT_EX, offsetof(CustomObject, last
     "last name"},
    {"number", Py_T_INT, offsetof(CustomObject, number),
     "custom number"},
    {NULL}  /* Sentinel */
};

static PyObject *
Custom_name(PyObject *op, PyObject *Py_UNUSED(dummy))
{
    CustomObject *self = (CustomObject *) op;
    if (self->first == NULL) {
        PyErr_SetString(PyExc_AttributeError, "first");
        return NULL;
    }
    if (self->last == NULL) {
        PyErr_SetString(PyExc_AttributeError, "last");
        return NULL;
    }
    return PyUnicode_FromFormat("%S %S", self->first, se
}

static PyMethodDef Custom_methods[] = {
    {"name", Custom_name, METH_NOARGS,
     "Return the name, combining the first and last name
    },
    {NULL}  /* Sentinel */
};

static PyTypeObject CustomType = {
    .ob_base = PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "custom2.Custom",
    .tp_doc = PyDoc_STR("Custom objects"),
    .tp_basicsize = sizeof(CustomObject),
    .tp_itemsize = 0,
    .tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE
    .tp_new = Custom_new,
    .tp_init = Custom_init,
    .tp_dealloc = Custom_dealloc,
    .tp_members = Custom_members,
    .tp_methods = Custom_methods,
};

static int
custom_module_exec(PyObject *m)
{
    if (PyType_Ready(&CustomType) < 0) {
        return -1;
    }

    if (PyModule_AddObjectRef(m, "Custom", (PyObject *)
        return -1;
    }

    return 0;
}

static PyModuleDef_Slot custom_module_slots[] = {
    {Py_mod_exec, custom_module_exec},
    {Py_mod_multiple_interpreters, Py_MOD_MULTIPLE_INTER
    {0, NULL}
};

static PyModuleDef custom_module = {
    .m_base = PyModuleDef_HEAD_INIT,
    .m_name = "custom2",
    .m_doc = "Example module that creates an extension t
    .m_size = 0,
    .m_slots = custom_module_slots,
};

PyMODINIT_FUNC
PyInit_custom2(void)
{
    return PyModuleDef_Init(&custom_module);
}
```

Esta versión del módulo tiene una serie de cambios.

The **`Custom`** type now has three data attributes in its C struct, *first*, *last*, and *number*. The *first* and *last* variables are Python strings containing first and last names. The *number* attribute is a C integer.

La estructura del objeto se actualiza en consecuencia:

```
typedef struct {
    PyObject_HEAD
    PyObject *first; /* first name */
    PyObject *last;  /* last name */
    int number;
} CustomObject;
```

Debido a que ahora tenemos datos para administrar, debemos ser más cuidadosos con la asignación de objetos y la desasignación. Como mínimo, necesitamos un método de desasignación:

```
static void
Custom_dealloc(PyObject *op)
{
    CustomObject *self = (CustomObject *) op;
    Py_XDECREF(self->first);
    Py_XDECREF(self->last);
    Py_TYPE(self)->tp_free(self);
}
```

que se asigna al miembro **`tp_dealloc`**:

```
.tp_dealloc = Custom_dealloc,
```

This method first clears the reference counts of the two Python attributes. **`Py_XDECREF()`** correctly handles the case where its argument is `NULL` (which might happen here if `tp_new` failed midway). It then calls the **`tp_free`** member of the object's type (computed by `Py_TYPE(self)`) to free the object's memory. Note that the object's type might not be **`CustomType`**, because the object may be an instance of a subclass.

Nota

> The explicit cast to `CustomObject *` above is needed because we defined `Custom_dealloc` to take a `PyObject *` argument, as the `tp_dealloc` function pointer expects to receive a `PyObject *` argument. By assigning to the `tp_dealloc` slot of a type, we declare that it can only be called with instances of our `CustomObject` class, so the cast to `(CustomObject *)` is safe. This is object-oriented polymorphism, in C!
>
> In existing code, or in previous versions of this tutorial, you might see similar functions take a pointer to the subtype object structure (`CustomObject*`) directly, like this:
>
> ```
> Custom_dealloc(CustomObject *self)
> {
>     Py_XDECREF(self->first);
>     Py_XDECREF(self->last);
>     Py_TYPE(self)->tp_free((PyObject *) self);
> }
> ...
> .tp_dealloc = (destructor) Custom_dealloc,
> ```
>
> This does the same thing on all architectures that CPython supports, but according to the C standard, it invokes undefined behavior.

Queremos asegurarnos de que el nombre y el apellido se inicialicen en cadenas de caracteres vacías, por lo que proporcionamos una implementación `tp_new`:

```
static PyObject *
Custom_new(PyTypeObject *type, PyObject *args, PyObject
{
    CustomObject *self;
    self = (CustomObject *) type->tp_alloc(type, 0);
    if (self != NULL) {
        self->first = PyUnicode_FromString("");
        if (self->first == NULL) {
            Py_DECREF(self);
            return NULL;
```

```
        }
        self->last = PyUnicode_FromString("");
        if (self->last == NULL) {
            Py_DECREF(self);
            return NULL;
        }
        self->number = 0;
    }
    return (PyObject *) self;
}
```

e instalarlo en el miembro `tp_new`:

```
.tp_new = Custom_new,
```

The `tp_new` handler is responsible for creating (as opposed to initializing) objects of the type. It is exposed in Python as the `__new__()` method. It is not required to define a `tp_new` member, and indeed many extension types will simply reuse `PyType_GenericNew()` as done in the first version of the **`Custom`** type above. In this case, we use the `tp_new` handler to initialize the `first` and `last` attributes to non-`NULL` default values.

`tp_new` se pasa el tipo que se instancia (no necesariamente `CustomType`, si se instancia una subclase) y cualquier argumento pasado cuando se llamó al tipo, y se espera que retorna la instancia creada. Los manejadores `tp_new` siempre aceptan argumentos posicionales y de palabras clave, pero a menudo ignoran los argumentos, dejando el manejo de argumentos al inicializador (también conocido como, `tp_init` en C o `__init__` en Python).

Nota

`tp_new` no debería llamar explícitamente a `tp_init`, ya que el intérprete lo hará por sí mismo.

La implementación `tp_new` llama al `tp_alloc` para asignar memoria:

```
self = (CustomObject *) type->tp_alloc(type, 0);
```

Como la asignación de memoria puede fallar, debemos verificar el resultado `tp_alloc` contra `NULL` antes de continuar.

Nota

No llenamos la ranura `tp_alloc` nosotros mismos. Más bien `PyType_Ready()` lo llena para nosotros al heredarlo de nuestra clase base, que es `object` por defecto. La mayoría de los tipos utilizan la estrategia de asignación predeterminada.

Nota

If you are creating a co-operative `tp_new` (one that calls a base type's `tp_new` or `__new__()`), you must *not* try to determine what method to call using method resolution order at runtime. Always statically determine what type you are going to call, and call its `tp_new` directly, or via `type->tp_base->tp_new`. If you do not do this, Python subclasses of your type that also inherit from other Python-defined classes may not work correctly. (Specifically, you may not be able to create instances of such subclasses without getting a `TypeError`.)

También definimos una función de inicialización que acepta argumentos para proporcionar valores iniciales para nuestra instancia:

```
static int
Custom_init(PyObject *op, PyObject *args, PyObject *kwds
{
    CustomObject *self = (CustomObject *) op;
    static char *kwlist[] = {"first", "last", "number",
    PyObject *first = NULL, *last = NULL, *tmp;

    if (!PyArg_ParseTupleAndKeywords(args, kwds, "|OOi",
                                     &first, &last,
                                     &self->number))
```

```
        return -1;

    if (first) {
        tmp = self->first;
        Py_INCREF(first);
        self->first = first;
        Py_XDECREF(tmp);
    }
    if (last) {
        tmp = self->last;
        Py_INCREF(last);
        self->last = last;
        Py_XDECREF(tmp);
    }
    return 0;
}
```

rellenando la ranura **`tp_init`**.

```
.tp_init = Custom_init,
```

The **`tp_init`** slot is exposed in Python as the **`__init__()`** method. It is used to initialize an object after it's created. Initializers always accept positional and keyword arguments, and they should return either `0` on success or `-1` on error.

Unlike the `tp_new` handler, there is no guarantee that `tp_init` is called at all (for example, the **`pickle`** module by default doesn't call **`__init__()`** on unpickled instances). It can also be called multiple times. Anyone can call the **`__init__()`** method on our objects. For this reason, we have to be extra careful when assigning the new attribute values. We might be tempted, for example to assign the `first` member like this:

```
if (first) {
    Py_XDECREF(self->first);
    Py_INCREF(first);
    self->first = first;
}
```

But this would be risky. Our type doesn't restrict the type of the `first` member, so it could be any kind of object. It could have a destructor that causes code to be executed that tries to access the `first` member; or that destructor could detach the thread state and let arbitrary code run in other threads that accesses and modifies our object.

Para ser paranoicos y protegernos de esta posibilidad, casi siempre reasignamos miembros antes de disminuir sus recuentos de referencias. ¿Cuándo no tenemos que hacer esto?

- cuando sabemos absolutamente que el recuento de referencia es mayor que 1;
- when we know that deallocation of the object [1] will neither detach the thread state nor cause any calls back into our type's code;
- al disminuir un recuento de referencias en un manejador **`tp_dealloc`** en un tipo que no admite la recolección de basura cíclica [2].

Queremos exponer nuestras variables de instancia como atributos. Hay varias formas de hacerlo. La forma más simple es definir definiciones de miembros:

```
static PyMemberDef Custom_members[] = {
    {"first", Py_T_OBJECT_EX, offsetof(CustomObject, fir
     "first name"},
    {"last", Py_T_OBJECT_EX, offsetof(CustomObject, last
     "last name"},
    {"number", Py_T_INT, offsetof(CustomObject, number),
     "custom number"},
    {NULL}  /* Sentinel */
};
```

y poner las definiciones en la ranura **`tp_members`**:

```
.tp_members = Custom_members,
```

Cada definición de miembro tiene un nombre de miembro, tipo, desplazamiento, banderas de acceso y cadena de caracteres de documentación. Consulte la sección Gestión de atributos genéricos a continuación para obtener más detalles.

Una desventaja de este enfoque es que no proporciona una forma de restringir los tipos de objetos que se pueden asignar a los atributos de Python. Esperamos que el nombre y el apellido sean cadenas, pero se pueden asignar objetos de Python. Además, los atributos se pueden eliminar, configurando los punteros C en `NULL`. Si bien podemos asegurarnos de que los miembros se inicialicen en valores que no sean `NULL`, los miembros se pueden establecer en `NULL` si se eliminan los atributos.

We define a single method, **`Custom.name()`**, that outputs the objects name as the concatenation of the first and last names.

```
static PyObject *
Custom_name(PyObject *op, PyObject *Py_UNUSED(dummy))
{
    CustomObject *self = (CustomObject *) op;
    if (self->first == NULL) {
        PyErr_SetString(PyExc_AttributeError, "first");
        return NULL;
    }
    if (self->last == NULL) {
        PyErr_SetString(PyExc_AttributeError, "last");
        return NULL;
    }
    return PyUnicode_FromFormat("%S %S", self->first, se
}
```

The method is implemented as a C function that takes a **`Custom`** (or **`Custom`** subclass) instance as the first argument. Methods always take an instance as the first argument. Methods often take positional and keyword arguments as well, but in this case we don’t take any and don’t need to accept a positional argument tuple or keyword argument dictionary. This method is equivalent to the Python method:

```
def name(self):
    return "%s %s" % (self.first, self.last)
```

Note that we have to check for the possibility that our `first` and `last` members are `NULL`. This is because they can be deleted, in which case they are set to `NULL`. It would be better to prevent deletion of these attributes and to restrict the attribute values to be strings. We'll see how to do that in the next section.

Ahora que hemos definido el método, necesitamos crear un arreglo de definiciones de métodos:

```
static PyMethodDef Custom_methods[] = {
    {"name", Custom_name, METH_NOARGS,
     "Return the name, combining the first and last name
    },
    {NULL}  /* Sentinel */
};
```

(note that we used the `METH_NOARGS` flag to indicate that the method is expecting no arguments other than *self*)

y asignarlo a la ranura `tp_methods`:

```
.tp_methods = Custom_methods,
```

Finally, we'll make our type usable as a base class for subclassing. We've written our methods carefully so far so that they don't make any assumptions about the type of the object being created or used, so all we need to do is to add the `Py_TPFLAGS_BASETYPE` to our class flag definition:

```
.tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE,
```

We rename `PyInit_custom()` to `PyInit_custom2()`, update the module name in the `PyModuleDef` struct, and update the full class name in the `PyTypeObject` struct.

Finally, we update our `setup.py` file to include the new module,

```
from setuptools import Extension, setup
setup(ext_modules=[
    Extension("custom", ["custom.c"]),
    Extension("custom2", ["custom2.c"]),
])
```

and then we re-install so that we can `import custom2`:

```
$ python -m pip install .
```

2.3. Proporcionar un control más preciso sobre los atributos de datos

In this section, we'll provide finer control over how the **`first`** and **`last`** attributes are set in the **`Custom`** example. In the previous version of our module, the instance variables **`first`** and **`last`** could be set to non-string values or even deleted. We want to make sure that these attributes always contain strings.

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>
#include <stddef.h> /* for offsetof() */

typedef struct {
    PyObject_HEAD
    PyObject *first; /* first name */
    PyObject *last;  /* last name */
    int number;
} CustomObject;

static void
Custom_dealloc(PyObject *op)
{
    CustomObject *self = (CustomObject *) op;
    Py_XDECREF(self->first);
    Py_XDECREF(self->last);
    Py_TYPE(self)->tp_free(self);
}

static PyObject *
Custom_new(PyTypeObject *type, PyObject *args, PyObject
{
    CustomObject *self;
    self = (CustomObject *) type->tp_alloc(type, 0);
    if (self != NULL) {
        self->first = Py_GetConstant(Py_CONSTANT_EMPTY_S
        if (self->first == NULL) {
            Py_DECREF(self);
            return NULL;
        }
        self->last = Py_GetConstant(Py_CONSTANT_EMPTY_ST
        if (self->last == NULL) {
            Py_DECREF(self);
            return NULL;
        }
        self->number = 0;
    }
    return (PyObject *) self;
}

static int
Custom_init(PyObject *op, PyObject *args, PyObject *kwds
{
    CustomObject *self = (CustomObject *) op;
    static char *kwlist[] = {"first", "last", "number",
    PyObject *first = NULL, *last = NULL;

    if (!PyArg_ParseTupleAndKeywords(args, kwds, "|UUi",
                                     &first, &last,
                                     &self->number))
        return -1;

    if (first) {
        Py_SETREF(self->first, Py_NewRef(first));
    }
    if (last) {
        Py_SETREF(self->last, Py_NewRef(last));
    }
    return 0;
}
```

```
static PyMemberDef Custom_members[] = {
    {"number", Py_T_INT, offsetof(CustomObject, number),
     "custom number"},
    {NULL}  /* Sentinel */
};

static PyObject *
Custom_getfirst(PyObject *op, void *closure)
{
    CustomObject *self = (CustomObject *) op;
    return Py_NewRef(self->first);
}

static int
Custom_setfirst(PyObject *op, PyObject *value, void *clo
{
    CustomObject *self = (CustomObject *) op;
    if (value == NULL) {
        PyErr_SetString(PyExc_TypeError, "Cannot delete
        return -1;
    }
    if (!PyUnicode_Check(value)) {
        PyErr_SetString(PyExc_TypeError,
                        "The first attribute value must
        return -1;
    }
    Py_SETREF(self->first, Py_NewRef(value));
    return 0;
}

static PyObject *
Custom_getlast(PyObject *op, void *closure)
{
    CustomObject *self = (CustomObject *) op;
    return Py_NewRef(self->last);
}
```

```
static int
Custom_setlast(PyObject *op, PyObject *value, void *clos
{
    CustomObject *self = (CustomObject *) op;
    if (value == NULL) {
        PyErr_SetString(PyExc_TypeError, "Cannot delete 
        return -1;
    }
    if (!PyUnicode_Check(value)) {
        PyErr_SetString(PyExc_TypeError,
                        "The last attribute value must b
        return -1;
    }
    Py_SETREF(self->last, Py_NewRef(value));
    return 0;
}

static PyGetSetDef Custom_getsetters[] = {
    {"first", Custom_getfirst, Custom_setfirst,
     "first name", NULL},
    {"last", Custom_getlast, Custom_setlast,
     "last name", NULL},
    {NULL}  /* Sentinel */
};

static PyObject *
Custom_name(PyObject *op, PyObject *Py_UNUSED(dummy))
{
    CustomObject *self = (CustomObject *) op;
    return PyUnicode_FromFormat("%S %S", self->first, se
}

static PyMethodDef Custom_methods[] = {
    {"name", Custom_name, METH_NOARGS,
     "Return the name, combining the first and last name
    },
    {NULL}  /* Sentinel */
};
```

```
static PyTypeObject CustomType = {
    .ob_base = PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "custom3.Custom",
    .tp_doc = PyDoc_STR("Custom objects"),
    .tp_basicsize = sizeof(CustomObject),
    .tp_itemsize = 0,
    .tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE
    .tp_new = Custom_new,
    .tp_init = Custom_init,
    .tp_dealloc = Custom_dealloc,
    .tp_members = Custom_members,
    .tp_methods = Custom_methods,
    .tp_getset = Custom_getsetters,
};

static int
custom_module_exec(PyObject *m)
{
    if (PyType_Ready(&CustomType) < 0) {
        return -1;
    }

    if (PyModule_AddObjectRef(m, "Custom", (PyObject *)
        return -1;
    }

    return 0;
}

static PyModuleDef_Slot custom_module_slots[] = {
    {Py_mod_exec, custom_module_exec},
    {Py_mod_multiple_interpreters, Py_MOD_MULTIPLE_INTER
    {0, NULL}
};

static PyModuleDef custom_module = {
    .m_base = PyModuleDef_HEAD_INIT,
    .m_name = "custom3",
    .m_doc = "Example module that creates an extension t
    .m_size = 0,
    .m_slots = custom_module_slots,
};

PyMODINIT_FUNC
PyInit_custom3(void)
{
    return PyModuleDef_Init(&custom_module);
}
```

To provide greater control, over the **`first`** and **`last`** attributes, we'll use custom getter and setter functions. Here are the functions for getting and setting the **`first`** attribute:

```
static PyObject *
Custom_getfirst(PyObject *op, void *closure)
{
    CustomObject *self = (CustomObject *) op;
    Py_INCREF(self->first);
    return self->first;
}

static int
Custom_setfirst(PyObject *op, PyObject *value, void *clo
{
    CustomObject *self = (CustomObject *) op;
    PyObject *tmp;
    if (value == NULL) {
        PyErr_SetString(PyExc_TypeError, "Cannot delete
        return -1;
    }
    if (!PyUnicode_Check(value)) {
        PyErr_SetString(PyExc_TypeError,
                        "The first attribute value must
        return -1;
    }
    tmp = self->first;
    Py_INCREF(value);
    self->first = value;
    Py_DECREF(tmp);
    return 0;
}
```

The getter function is passed a **`Custom`** object and a «closure», which is a void pointer. In this case, the closure is ignored. (The closure supports an advanced usage in which definition data is passed to the getter and setter. This could, for example, be used to allow a single set of getter and setter functions that decide the attribute to get or set based on data in the closure.)

The setter function is passed the **`Custom`** object, the new value, and the closure. The new value may be `NULL`, in which case the attribute is being deleted. In our setter, we raise an error if the attribute is deleted or if its new value is not a string.

Creamos un arreglo de estructuras **`PyGetSetDef`**:

```
static PyGetSetDef Custom_getsetters[] = {
    {"first", Custom_getfirst, Custom_setfirst,
     "first name", NULL},
    {"last", Custom_getlast, Custom_setlast,
     "last name", NULL},
    {NULL}  /* Sentinel */
};
```

y lo registra en la ranura **`tp_getset`**:

```
.tp_getset = Custom_getsetters,
```

El último elemento en la estructura **`PyGetSetDef`** es el «cierre» (*closure*) mencionado anteriormente. En este caso, no estamos usando un cierre, por lo que simplemente pasamos `NULL`.

También eliminamos las definiciones de miembro para estos atributos:

```
static PyMemberDef Custom_members[] = {
    {"number", Py_T_INT, offsetof(CustomObject, number),
     "custom number"},
    {NULL}  /* Sentinel */
};
```

También necesitamos actualizar el manejador **`tp_init`** para permitir que solo se pasen las cadenas [3]:

```
static int
Custom_init(PyObject *op, PyObject *args, PyObject *kwds
{
    CustomObject *self = (CustomObject *) op;
    static char *kwlist[] = {"first", "last", "number",
    PyObject *first = NULL, *last = NULL, *tmp;

    if (!PyArg_ParseTupleAndKeywords(args, kwds, "|UUi",
                                     &first, &last,
                                     &self->number))
        return -1;

    if (first) {
        tmp = self->first;
        Py_INCREF(first);
        self->first = first;
        Py_DECREF(tmp);
    }
    if (last) {
        tmp = self->last;
        Py_INCREF(last);
        self->last = last;
        Py_DECREF(tmp);
    }
    return 0;
}
```

Con estos cambios, podemos asegurar que los miembros `primero` y `último` nunca sean `NULL`, por lo que podemos eliminar las comprobaciones de los valores `NULL` en casi todos los casos. Esto significa que la mayoría de las llamadas **`Py_XDECREF()`** se pueden convertir en llamadas **`Py_DECREF()`**. El único lugar donde no podemos cambiar estas llamadas es en la implementación `tp_dealloc`, donde existe la posibilidad de que la inicialización de estos miembros falle en `tp_new`.

También cambiamos el nombre de la función de inicialización del módulo y el nombre del módulo en la función de inicialización, como lo hicimos antes, y agregamos una definición adicional al archivo `setup.py`.

2.4. Apoyo a la recolección de basura cíclica

Python tiene un recolector de basura cíclico (GC) que puede identificar objetos innecesarios incluso cuando sus recuentos de referencia no son cero. Esto puede suceder cuando los objetos están involucrados en ciclos. Por ejemplo, considere:

```
>>> l = []
>>> l.append(l)
>>> del l
```

En este ejemplo, creamos una lista que se contiene a sí misma. Cuando lo eliminamos, todavía tiene una referencia de sí mismo. Su recuento de referencia no cae a cero. Afortunadamente, el recolector cíclico de basura de Python finalmente descubrirá que la lista es basura y la liberará.

In the second version of the **`Custom`** example, we allowed any kind of object to be stored in the **`first`** or **`last`** attributes [4]. Besides, in the second and third versions, we allowed subclassing **`Custom`**, and subclasses may add arbitrary attributes. For any of those two reasons, **`Custom`** objects can participate in cycles:

```
>>> import custom3
>>> class Derived(custom3.Custom): pass
...
>>> n = Derived()
>>> n.some_attribute = n
```

To allow a **`Custom`** instance participating in a reference cycle to be

properly detected and collected by the cyclic GC, our **`Custom`** type needs to fill two additional slots and to enable a flag that enables these slots:

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>
#include <stddef.h> /* for offsetof() */

typedef struct {
    PyObject_HEAD
    PyObject *first; /* first name */
    PyObject *last;  /* last name */
    int number;
} CustomObject;

static int
Custom_traverse(PyObject *op, visitproc visit, void *arg
{
    CustomObject *self = (CustomObject *) op;
    Py_VISIT(self->first);
    Py_VISIT(self->last);
    return 0;
}

static int
Custom_clear(PyObject *op)
{
    CustomObject *self = (CustomObject *) op;
    Py_CLEAR(self->first);
    Py_CLEAR(self->last);
    return 0;
}

static void
Custom_dealloc(PyObject *op)
{
    PyObject_GC_UnTrack(op);
    (void)Custom_clear(op);
    Py_TYPE(op)->tp_free(op);
```

```
}

static PyObject *
Custom_new(PyTypeObject *type, PyObject *args, PyObject
{
    CustomObject *self;
    self = (CustomObject *) type->tp_alloc(type, 0);
    if (self != NULL) {
        self->first = Py_GetConstant(Py_CONSTANT_EMPTY_S
        if (self->first == NULL) {
            Py_DECREF(self);
            return NULL;
        }
        self->last = Py_GetConstant(Py_CONSTANT_EMPTY_ST
        if (self->last == NULL) {
            Py_DECREF(self);
            return NULL;
        }
        self->number = 0;
    }
    return (PyObject *) self;
}

static int
Custom_init(PyObject *op, PyObject *args, PyObject *kwds
{
    CustomObject *self = (CustomObject *) op;
    static char *kwlist[] = {"first", "last", "number",
    PyObject *first = NULL, *last = NULL;

    if (!PyArg_ParseTupleAndKeywords(args, kwds, "|UUi",
                                     &first, &last,
                                     &self->number))
        return -1;

    if (first) {
        Py_SETREF(self->first, Py_NewRef(first));
    }
    if (last) {
        Py_SETREF(self->last, Py_NewRef(last));
    }
    return 0;
}

static PyMemberDef Custom_members[] = {
    {"number", Py_T_INT, offsetof(CustomObject, number),
     "custom number"},
    {NULL}  /* Sentinel */
};

static PyObject *
Custom_getfirst(PyObject *op, void *closure)
{
    CustomObject *self = (CustomObject *) op;
    return Py_NewRef(self->first);
}

static int
Custom_setfirst(PyObject *op, PyObject *value, void *clo
{
    CustomObject *self = (CustomObject *) op;
    if (value == NULL) {
        PyErr_SetString(PyExc_TypeError, "Cannot delete
        return -1;
    }
    if (!PyUnicode_Check(value)) {
        PyErr_SetString(PyExc_TypeError,
                        "The first attribute value must
        return -1;
    }
    Py_XSETREF(self->first, Py_NewRef(value));
    return 0;
}

static PyObject *
Custom_getlast(PyObject *op, void *closure)
```

```
{
    CustomObject *self = (CustomObject *) op;
    return Py_NewRef(self->last);
}

static int
Custom_setlast(PyObject *op, PyObject *value, void *clos
{
    CustomObject *self = (CustomObject *) op;
    if (value == NULL) {
        PyErr_SetString(PyExc_TypeError, "Cannot delete
        return -1;
    }
    if (!PyUnicode_Check(value)) {
        PyErr_SetString(PyExc_TypeError,
                        "The last attribute value must b
        return -1;
    }
    Py_XSETREF(self->last, Py_NewRef(value));
    return 0;
}

static PyGetSetDef Custom_getsetters[] = {
    {"first", Custom_getfirst, Custom_setfirst,
     "first name", NULL},
    {"last", Custom_getlast, Custom_setlast,
     "last name", NULL},
    {NULL}  /* Sentinel */
};

static PyObject *
Custom_name(PyObject *op, PyObject *Py_UNUSED(dummy))
{
    CustomObject *self = (CustomObject *) op;
    return PyUnicode_FromFormat("%S %S", self->first, se
}

static PyMethodDef Custom_methods[] = {
    {"name", Custom_name, METH_NOARGS,
     "Return the name, combining the first and last name
    },
    {NULL}  /* Sentinel */
};

static PyTypeObject CustomType = {
    .ob_base = PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "custom4.Custom",
    .tp_doc = PyDoc_STR("Custom objects"),
    .tp_basicsize = sizeof(CustomObject),
    .tp_itemsize = 0,
    .tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE
    .tp_new = Custom_new,
    .tp_init = Custom_init,
    .tp_dealloc = Custom_dealloc,
    .tp_traverse = Custom_traverse,
    .tp_clear = Custom_clear,
    .tp_members = Custom_members,
    .tp_methods = Custom_methods,
    .tp_getset = Custom_getsetters,
};

static int
custom_module_exec(PyObject *m)
{
    if (PyType_Ready(&CustomType) < 0) {
        return -1;
    }

    if (PyModule_AddObjectRef(m, "Custom", (PyObject *)
        return -1;
    }

    return 0;
}

static PyModuleDef_Slot custom_module_slots[] = {
    {Py_mod_exec, custom_module_exec},
    {Py_mod_multiple_interpreters, Py_MOD_MULTIPLE_INTER
    {0, NULL}
};

static PyModuleDef custom_module = {
    .m_base = PyModuleDef_HEAD_INIT,
    .m_name = "custom4",
    .m_doc = "Example module that creates an extension t
    .m_size = 0,
    .m_slots = custom_module_slots,
};

PyMODINIT_FUNC
PyInit_custom4(void)
{
    return PyModuleDef_Init(&custom_module);
}
```

Primero, el método transversal permite que el GC cíclico conozca los subobjetos que podrían participar en los ciclos:

```
static int
Custom_traverse(PyObject *op, visitproc visit, void *arg
{
    CustomObject *self = (CustomObject *) op;
    int vret;
    if (self->first) {
        vret = visit(self->first, arg);
        if (vret != 0)
            return vret;
    }
    if (self->last) {
        vret = visit(self->last, arg);
        if (vret != 0)
            return vret;
    }
    return 0;
}
```

For each subobject that can participate in cycles, we need to call the **`visit()`** function, which is passed to the traversal method. The **`visit()`** function takes as arguments the subobject and the extra argument *arg* passed to the traversal method. It returns an integer value that must be returned if it is non-zero.

Python proporciona una macro `Py_VISIT()` que automatiza las funciones de visita de llamada. Con **`Py_VISIT()`**, podemos minimizar la cantidad de repeticiones en `Custom_traverse`:

```
static int
Custom_traverse(PyObject *op, visitproc visit, void *arg
{
    CustomObject *self = (CustomObject *) op;
    Py_VISIT(self->first);
    Py_VISIT(self->last);
    return 0;
}
```

Nota

La implementación `tp_traverse` debe nombrar sus argumentos exactamente *visit* y *arg* para usar `Py_VISIT()`.

En segundo lugar, debemos proporcionar un método para borrar cualquier subobjeto que pueda participar en los ciclos:

```
static int
Custom_clear(PyObject *op)
{
    CustomObject *self = (CustomObject *) op;
    Py_CLEAR(self->first);
    Py_CLEAR(self->last);
    return 0;
}
```

Observe el uso de la macro `Py_CLEAR()`. Es la forma recomendada y segura de borrar los atributos de datos de tipos arbitrarios al tiempo que disminuye sus recuentos de referencia. Si tuviera que llamar a **`Py_XDECREF()`** en lugar del atributo antes de establecerlo en `NULL`, existe la posibilidad de que el destructor del atributo vuelva a llamar al código que lee el atributo nuevamente (*especialmente* si hay un ciclo de referencia).

Nota

Puede emular **`Py_CLEAR()`** escribiendo:

```
PyObject *tmp;
tmp = self->first;
self->first = NULL;
Py_XDECREF(tmp);
```

Sin embargo, es mucho más fácil y menos propenso a errores usar siempre **`Py_CLEAR()`** al eliminar un atributo. ¡No intentes micro-optimizar a expensas de la robustez!

El desasignador `Custom_dealloc` puede llamar a un código arbitrario al borrar los atributos. Significa que el GC circular se puede activar dentro de la función. Dado que el GC asume que el recuento de referencias no es cero, debemos destrabar el objeto del GC llamando a **`PyObject_GC_UnTrack()`** antes de borrar los miembros. Aquí está nuestro reubicador reimplementado usando **`PyObject_GC_UnTrack()`** y `Custom_clear`:

```
static void
Custom_dealloc(PyObject *op)
{
    PyObject_GC_UnTrack(op);
    (void)Custom_clear(op);
    Py_TYPE(op)->tp_free(op);
}
```

Finally, we add the **`Py_TPFLAGS_HAVE_GC`** flag to the class flags:

```
.tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE | 
```

Eso es prácticamente todo. Si hubiéramos escrito controladores personalizados **`tp_alloc`** o **`tp_free`**, tendríamos que modificarlos para la recolección de basura cíclica. La mayoría de las extensiones usarán las versiones proporcionadas automáticamente.

2.5. Subclases de otros tipos

Es posible crear nuevos tipos de extensión que se derivan de los tipos existentes. Es más fácil heredar de los tipos incorporados, ya que una extensión puede usar fácilmente `PyTypeObject` que necesita. Puede ser difícil compartir estas estructuras `PyTypeObject` entre módulos de extensión.

In this example we will create a `SubList` type that inherits from the built-in `list` type. The new type will be completely compatible with regular lists, but will have an additional `increment()` method that increases an internal counter:

```
>>> import sublist
>>> s = sublist.SubList(range(3))
>>> s.extend(s)
>>> print(len(s))
6
>>> print(s.increment())
1
>>> print(s.increment())
2
```

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>

typedef struct {
    PyListObject list;
    int state;
} SubListObject;

static PyObject *
SubList_increment(PyObject *op, PyObject *Py_UNUSED(dumm
{
    SubListObject *self = (SubListObject *) op;
    self->state++;
    return PyLong_FromLong(self->state);
}

static PyMethodDef SubList_methods[] = {
    {"increment", SubList_increment, METH_NOARGS,
     PyDoc_STR("increment state counter")},
    {NULL},
};

static int
SubList_init(PyObject *op, PyObject *args, PyObject *kwd
{
    SubListObject *self = (SubListObject *) op;
    if (PyList_Type.tp_init(op, args, kwds) < 0)
        return -1;
    self->state = 0;
    return 0;
}

static PyTypeObject SubListType = {
    .ob_base = PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "sublist.SubList",
    .tp_doc = PyDoc_STR("SubList objects"),
    .tp_basicsize = sizeof(SubListObject),
    .tp_itemsize = 0,
    .tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE
    .tp_init = SubList_init,
    .tp_methods = SubList_methods,
};

static int
sublist_module_exec(PyObject *m)
{
    SubListType.tp_base = &PyList_Type;
    if (PyType_Ready(&SubListType) < 0) {
        return -1;
    }
    if (PyModule_AddObjectRef(m, "SubList", (PyObject *)
        return -1;
    }

    return 0;
}

static PyModuleDef_Slot sublist_module_slots[] = {
    {Py_mod_exec, sublist_module_exec},
    {Py_mod_multiple_interpreters, Py_MOD_MULTIPLE_INTER
    {0, NULL}
};

static PyModuleDef sublist_module = {
    .m_base = PyModuleDef_HEAD_INIT,
    .m_name = "sublist",
    .m_doc = "Example module that creates an extension t
    .m_size = 0,
    .m_slots = sublist_module_slots,
};

PyMODINIT_FUNC
PyInit_sublist(void)
{
    return PyModuleDef_Init(&sublist_module);
}
```

As you can see, the source code closely resembles the **`Custom`** examples in previous sections. We will break down the main differences between them.

```
typedef struct {
    PyListObject list;
    int state;
} SubListObject;
```

La diferencia principal para los objetos de tipo derivado es que la estructura de objeto del tipo base debe ser el primer valor. El tipo base ya incluirá **`PyObject_HEAD()`** al comienzo de su estructura.

When a Python object is a **`SubList`** instance, its `PyObject *` pointer can be safely cast to both `PyListObject *` and `SubListObject *`:

```
static int
SubList_init(PyObject *op, PyObject *args, PyObject *kwd
{
    SubListObject *self = (SubListObject *) op;
    if (PyList_Type.tp_init(op, args, kwds) < 0)
        return -1;
    self->state = 0;
    return 0;
}
```

We see above how to call through to the `__init__()` method of the base type.

Este patrón es importante cuando se escribe un tipo con miembros personalizados `tp_new` y `tp_dealloc`. El manejador **`tp_new`** no debería crear realmente la memoria para el objeto con su `tp_alloc`, pero deja que la clase base lo maneje llamando a su propio **`tp_new`**.

The `PyTypeObject` struct supports a `tp_base` specifying the type's concrete base class. Due to cross-platform compiler issues, you can't fill that field directly with a reference to `PyList_Type`; it should be done in the `Py_mod_exec` function:

```
static int
sublist_module_exec(PyObject *m)
{
    SubListType.tp_base = &PyList_Type;
    if (PyType_Ready(&SubListType) < 0) {
        return -1;
    }

    if (PyModule_AddObjectRef(m, "SubList", (PyObject *)
        return -1;
    }
    return 0;
}
```

Antes de llamar a `PyType_Ready()`, la estructura de tipo debe tener el espacio `tp_base` rellenado. Cuando derivamos un tipo existente, no es necesario completar el `tp_alloc` ranura con `PyType_GenericNew()` – la función de asignación del tipo base será heredada.

After that, calling `PyType_Ready()` and adding the type object to the module is the same as with the basic **`Custom`** examples.

Notas al pie

Esto es cierto cuando sabemos que el objeto es un tipo básico, como una cadena o un flotador.

Nos basamos en esto en el manejador `tp_dealloc` en este ejemplo, porque nuestro tipo no admite la recolección de basura.

Ahora sabemos que el primer y el último miembro son cadenas de caracteres, por lo que quizás podríamos ser menos cuidadosos al disminuir sus recuentos de referencia, sin embargo, aceptamos instancias de subclases de cadenas. A pesar de que la desasignación de cadenas normales no volverá a llamar a nuestros objetos, no podemos garantizar que la desasignación de una instancia de una subclase de cadena de caracteres no vuelva a llamar a nuestros objetos.

Además, incluso con nuestros atributos restringidos a instancias de cadenas, el usuario podría pasar subclases arbitrarias `str` y, por lo tanto, seguir creando ciclos de referencia.

3. Definición de tipos de extensión: temas variados

Esta sección tiene como objetivo dar un vistazo rápido a los diversos métodos de tipo que puede implementar y lo que hacen.

Aquí está la definición de `PyTypeObject`, con algunos campos que solo se usan en las versiones de depuración omitidas:

```
typedef struct _typeobject {
    PyObject_VAR_HEAD
    const char *tp_name; /* For printing, in format "<mo
    Py_ssize_t tp_basicsize, tp_itemsize; /* For allocat

    /* Methods to implement standard operations */

    destructor tp_dealloc;
    Py_ssize_t tp_vectorcall_offset;
    getattrfunc tp_getattr;
    setattrfunc tp_setattr;
    PyAsyncMethods *tp_as_async; /* formerly known as tp
                                    or tp_reserved (Pyth
    reprfunc tp_repr;

    /* Method suites for standard classes */

    PyNumberMethods *tp_as_number;
    PySequenceMethods *tp_as_sequence;
    PyMappingMethods *tp_as_mapping;

    /* More standard operations (here for binary compati

    hashfunc tp_hash;
    ternaryfunc tp_call;
    reprfunc tp_str;
    getattrofunc tp_getattro;
    setattrofunc tp_setattro;

    /* Functions to access object as input/output buffer
    PyBufferProcs *tp_as_buffer;

    /* Flags to define presence of optional/expanded fea
    unsigned long tp_flags;

    const char *tp_doc; /* Documentation string */

    /* Assigned meaning in release 2.0 */
    /* call function for all accessible objects */
    traverseproc tp_traverse;

    /* delete references to contained objects */
    inquiry tp_clear;

    /* Assigned meaning in release 2.1 */
    /* rich comparisons */
    richcmpfunc tp_richcompare;

    /* weak reference enabler */
    Py_ssize_t tp_weaklistoffset;

    /* Iterators */
    getiterfunc tp_iter;
    iternextfunc tp_iternext;

    /* Attribute descriptor and subclassing stuff */
    PyMethodDef *tp_methods;
    PyMemberDef *tp_members;
    PyGetSetDef *tp_getset;
    // Strong reference on a heap type, borrowed referen
    PyTypeObject *tp_base;
    PyObject *tp_dict;
    descrgetfunc tp_descr_get;
    descrsetfunc tp_descr_set;
    Py_ssize_t tp_dictoffset;
    initproc tp_init;
    allocfunc tp_alloc;
    newfunc tp_new;
    freefunc tp_free; /* Low-level free-memory routine *
    inquiry tp_is_gc; /* For PyObject_IS_GC */
    PyObject *tp_bases;
    PyObject *tp_mro; /* method resolution order */
    PyObject *tp_cache; /* no longer used */
    void *tp_subclasses;  /* for static builtin types th
    PyObject *tp_weaklist; /* not used for static builti
    destructor tp_del;

    /* Type attribute cache version tag. Added in versio
     * If zero, the cache is invalid and must be initial
     */
    unsigned int tp_version_tag;

    destructor tp_finalize;
    vectorcallfunc tp_vectorcall;

    /* bitset of which type-watchers care about this typ
    unsigned char tp_watched;

    /* Number of tp_version_tag values used.
     * Set to _Py_ATTR_CACHE_UNUSED if the attribute cac
     * disabled for this type (e.g. due to custom MRO en
     * Otherwise, limited to MAX_VERSIONS_PER_CLASS (def
     */
    uint16_t tp_versions_used;
} PyTypeObject;
```

Esos son *muchos* métodos. Sin embargo, no se preocupe demasiado: si tiene un tipo que desea definir, es muy probable que solo implemente un puñado de estos.

Como probablemente espera ahora, vamos a repasar esto y daremos más información sobre los diversos controladores. No iremos en el orden en que se definen en la estructura, porque hay mucho equipaje histórico que afecta el orden de los campos. A menudo es más fácil encontrar un ejemplo que incluya los campos que necesita y luego cambiar los valores para adaptarlos a su nuevo tipo.

```
const char *tp_name; /* For printing */
```

El nombre del tipo – como se mencionó en el capítulo anterior, aparecerá en varios lugares, casi por completo para fines de diagnóstico. ¡Intente elegir algo que sea útil en tal situación!

```
Py_ssize_t tp_basicsize, tp_itemsize; /* For allocation
```

Estos campos le dicen al tiempo de ejecución cuánta memoria asignar cuando se crean nuevos objetos de este tipo. Python tiene algún soporte incorporado para estructuras de longitud variable (piense: cadenas, tuplas) que es donde entra el campo **`tp_itemsize`**. Esto se tratará más adelante.

```
const char *tp_doc;
```

Aquí puede poner una cadena de caracteres (o su dirección) que desea que se retorne cuando el script de Python haga referencia a `obj.__doc__` para recuperar el docstring.

Ahora llegamos a los métodos de tipo básicos: los que implementarán la mayoría de los tipos de extensión.

3.1. Finalización y desasignación

```
destructor tp_dealloc;
```

Se llama a esta función cuando el recuento de referencia de la instancia de su tipo se reduce a cero y el intérprete de Python quiere reclamarlo. Si su tipo tiene memoria para liberar u otra limpieza para realizar, puede ponerla aquí. El objeto en sí mismo necesita ser liberado aquí también. Aquí hay un ejemplo de esta función:

```
static void
newdatatype_dealloc(PyObject *op)
```

```
{
    newdatatypeobject *self = (newdatatypeobject *) op;
    free(self->obj_UnderlyingDatatypePtr);
    Py_TYPE(self)->tp_free(self);
}
```

Si su tipo admite la recolección de basura, el destructor debe llamar a **`PyObject_GC_UnTrack()`** antes de borrar cualquier campo miembro:

```
static void
newdatatype_dealloc(PyObject *op)
{
    newdatatypeobject *self = (newdatatypeobject *) op;
    PyObject_GC_UnTrack(op);
    Py_CLEAR(self->other_obj);
    ...
    Py_TYPE(self)->tp_free(self);
}
```

Un requisito importante de la función desasignador es que deja solo las excepciones pendientes. Esto es importante ya que los desasignadores se llaman con frecuencia cuando el intérprete desenrolla la pila de Python; cuando la pila se desenrolla debido a una excepción (en lugar de retornos normales), no se hace nada para proteger a los desasignadores de memoria (*deallocator*) de ver que ya se ha establecido una excepción. Cualquier acción que realice un desasignador que pueda hacer que se ejecute código Python adicional puede detectar que se ha establecido una excepción. Esto puede conducir a errores engañosos del intérprete. La forma correcta de protegerse contra esto es guardar una excepción pendiente antes de realizar la acción insegura y restaurarla cuando haya terminado. Esto se puede hacer usando las funciones **`PyErr_Fetch()`** y **`PyErr_Restore()`**:

```
static void
my_dealloc(PyObject *obj)
{
    MyObject *self = (MyObject *) obj;
    PyObject *cbresult;
    if (self->my_callback != NULL) {
        PyObject *err_type, *err_value, *err_traceback;

        /* This saves the current exception state */
        PyErr_Fetch(&err_type, &err_value, &err_tracebac

        cbresult = PyObject_CallNoArgs(self->my_callback
        if (cbresult == NULL) {
           PyErr_WriteUnraisable(self->my_callback);
        }
        else {
            Py_DECREF(cbresult);
        }

        /* This restores the saved exception state */
        PyErr_Restore(err_type, err_value, err_traceback

        Py_DECREF(self->my_callback);
    }
    Py_TYPE(self)->tp_free(self);
}
```

Nota

Existen limitaciones para lo que puede hacer de manera segura en una función de desasignación. Primero, si su tipo admite la recolección de basura (usando `tp_traverse` o `tp_clear`), algunos de los miembros del objeto pueden haber sido borrados o finalizados por el time `tp_dealloc` es llamado. Segundo, en `tp_dealloc`, su objeto está en un estado inestable: su recuento de referencia es igual a cero. Cualquier llamada a un objeto o API no trivial (como en el ejemplo anterior) podría terminar llamando `tp_dealloc` nuevamente, causando una doble liberación y un bloqueo.

Comenzando con Python 3.4, se recomienda no poner ningún código de finalización complejo en `tp_dealloc`, y en su lugar use el nuevo método de tipo `tp_finalize`.

Ver también

PEP 442 [https://peps.python.org/pep-0442/] explica el nuevo esquema de finalización.

3.2. Presentación de objetos

En Python, hay dos formas de generar una representación textual de un objeto: la función `repr()`, y la función `str()`. (La función `print()` solo llama a `str()`.) Estos controladores son opcionales.

```
reprfunc tp_repr;
reprfunc tp_str;
```

El manejador `tp_repr` debe retornar un objeto de cadena que contenga una representación de la instancia para la que se llama. Aquí hay un ejemplo simple:

```
static PyObject *
newdatatype_repr(PyObject *op)
{
    newdatatypeobject *self = (newdatatypeobject *) op;
    return PyUnicode_FromFormat("Repr-ified_newdatatype{
                                self->obj_UnderlyingData
}
```

Si no se especifica un controlador `tp_repr`, el intérprete proporcionará una representación que utiliza el `tp_name` del tipo y un valor de identificación exclusivo para el objeto.

El manejador `tp_str` es para `str()` lo que el manejador `tp_repr` descrito arriba es para `repr()`; es decir, se llama cuando el código Python llama `str()` en una instancia de su objeto. Su implementación es muy similar a la función `tp_repr`, pero la cadena resultante está destinada al consumo humano. Si `tp_str` no se especifica, en su lugar se utiliza el controlador `tp_repr`.

Aquí hay un ejemplo simple:

```
static PyObject *
newdatatype_str(PyObject *op)
{
    newdatatypeobject *self = (newdatatypeobject *) op;
    return PyUnicode_FromFormat("Stringified_newdatatype
                                self->obj_UnderlyingData
}
```

3.3. Gestión de atributos

Para cada objeto que puede soportar atributos, el tipo correspondiente debe proporcionar las funciones que controlan cómo se resuelven los atributos. Es necesario que haya una función que pueda recuperar atributos (si hay alguna definida), y otra para establecer atributos (si se permite establecer atributos). La eliminación de un atributo es un caso especial, para el cual el nuevo valor pasado al controlador es `NULL`.

Python admite dos pares de controladores de atributos; un tipo que admite atributos solo necesita implementar las funciones para un par. La diferencia es que un par toma el nombre del atributo como char*, mientras que el otro acepta un PyObject*. Cada tipo puede usar cualquier par que tenga más sentido para la conveniencia de la implementación.

```
getattrfunc  tp_getattr;        /* char * version */
setattrfunc  tp_setattr;
/* ... */
getattrofunc tp_getattro;       /* PyObject * version */
setattrofunc tp_setattro;
```

Si acceder a los atributos de un objeto siempre es una operación simple (esto se explicará en breve), existen implementaciones genéricas que se pueden usar para proporcionar la versión PyObject* de las funciones de administración de atributos. La necesidad real de controladores de atributos específicos de tipo desapareció casi por completo a partir de Python 2.2, aunque hay muchos ejemplos que no se han actualizado para usar algunos de los nuevos mecanismos genéricos disponibles.

3.3.1. Gestión de atributos genéricos

La mayoría de los tipos de extensión solo usan atributos *simple*. Entonces, ¿qué hace que los atributos sean simples? Solo hay un par de condiciones que se deben cumplir:

1. El nombre de los atributos debe ser conocido cuando **`PyType_Ready()`** es llamado.
2. No se necesita un procesamiento especial para registrar que un atributo se buscó o se configuró, ni se deben tomar acciones basadas en el valor.

Tenga en cuenta que esta lista no impone restricciones a los valores de los atributos, cuándo se calculan los valores o cómo se almacenan los datos relevantes.

Cuando se llama a **`PyType_Ready()`**, utiliza tres tablas a las que hace referencia el objeto de tipo para crear descriptor que se colocan en el diccionario del objeto de tipo. Cada descriptor controla el acceso a un atributo del objeto de instancia. Cada una de las tablas es opcional; si los tres son `NULL`, las instancias del tipo solo tendrán atributos que se heredan de su tipo base, y deberían dejar **`tp_getattro`** y los campos **`tp_setattro`** `NULL` también, permitiendo que el tipo base maneje los atributos.

Las tablas se declaran como tres campos del tipo objeto:

```
struct PyMethodDef *tp_methods;
struct PyMemberDef *tp_members;
struct PyGetSetDef *tp_getset;
```

Si **`tp_methods`** no es `NULL`, debe referirse a un arreglo de estructuras **`PyMethodDef`**. Cada entrada en la tabla es una instancia de esta estructura:

```
typedef struct PyMethodDef {
    const char  *ml_name;       /* method name */
    PyCFunction  ml_meth;       /* implementation functi
    int          ml_flags;      /* flags */
    const char  *ml_doc;        /* docstring */
} PyMethodDef;
```

One entry should be defined for each method provided by the type; no entries are needed for methods inherited from a base type. One additional entry is needed at the end; it is a sentinel that marks the end of the array. The **`ml_name`** field of the sentinel must be `NULL`.

La segunda tabla se utiliza para definir atributos que se asignan directamente a los datos almacenados en la instancia. Se admite una variedad de tipos C primitivos, y el acceso puede ser de solo lectura o lectura-escritura. Las estructuras en la tabla se definen como:

```
typedef struct PyMemberDef {
    const char *name;
    int         type;
    int         offset;
    int         flags;
    const char *doc;
} PyMemberDef;
```

For each entry in the table, a descriptor will be constructed and added to the type which will be able to extract a value from the instance structure. The **`type`** field should contain a type code like **`Py_T_INT`** or **`Py_T_DOUBLE`**; the value will be used to determine how to convert Python values to and from C values. The **`flags`** field is used to store flags which control how the attribute can be accessed: you can set it to **`Py_READONLY`** to prevent Python code from setting it.

An interesting advantage of using the **`tp_members`** table to build descriptors that are used at runtime is that any attribute defined this way can have an associated doc string simply by providing the text in the table. An application can use the introspection API to retrieve the descriptor from the class object, and get the doc string using its **`__doc__`** attribute.

As with the **`tp_methods`** table, a sentinel entry with a **`ml_name`** value of `NULL` is required.

3.3.2. Gestión de atributos específicos de tipo

Para simplificar, solo la versión char* será demostrada aquí; el tipo del parámetro con nombre es la única diferencia entre char* y PyObject* de la interfaz. Este ejemplo efectivamente hace lo mismo que el ejemplo genérico anterior, pero no usa el soporte genérico agregado en Python 2.2. Explica cómo se llaman las funciones del controlador, de modo que si necesita ampliar su funcionalidad, comprenderá lo que debe hacerse.

The **`tp_getattr`** handler is called when the object requires an attribute look-up. It is called in the same situations where the **`__getattr__()`** method of a class would be called.

Aquí hay un ejemplo:

```
static PyObject *
newdatatype_getattr(PyObject *op, char *name)
{
    newdatatypeobject *self = (newdatatypeobject *) op;
    if (strcmp(name, "data") == 0) {
        return PyLong_FromLong(self->data);
    }

    PyErr_Format(PyExc_AttributeError,
                 "'%.100s' object has no attribute '%.40
                 Py_TYPE(self)->tp_name, name);
    return NULL;
}
```

The **`tp_setattr`** handler is called when the **`__setattr__()`** or **`__delattr__()`** method of a class instance would be called. When an attribute should be deleted, the third parameter will be `NULL`. Here is an example that simply raises an exception; if this were really all you wanted, the **`tp_setattr`** handler should be set to `NULL`.

```
static int
newdatatype_setattr(PyObject *op, char *name, PyObject *
{
    PyErr_Format(PyExc_RuntimeError, "Read-only attribut
    return -1;
}
```

3.4. Comparación de Objetos

```
richcmpfunc tp_richcompare;
```

The **`tp_richcompare`** handler is called when comparisons are needed. It is analogous to the rich comparison methods, like **`__lt__()`**, and also called by **`PyObject_RichCompare()`** and **`PyObject_RichCompareBool()`**.

Esta función se llama con dos objetos de Python y el operador como argumentos, donde el operador es uno de `Py_EQ`, `Py_NE`, `Py_LE`, `Py_GE`, `Py_LT` o `Py_GT`. Debe comparar los dos objetos con respecto al operador especificado y retornar `Py_True` o `Py_False` si la comparación es exitosa, `Py_NotImplemented` para indicar que la comparación no está implementada y se debe probar el método de comparación del otro objeto, o `NULL` si se estableció una excepción.

Aquí hay una implementación de muestra, para un tipo de datos que se considera igual si el tamaño de un puntero interno es igual:

```
static PyObject *
newdatatype_richcmp(PyObject *lhs, PyObject *rhs, int op
{
    newdatatypeobject *obj1 = (newdatatypeobject *) lhs;
    newdatatypeobject *obj2 = (newdatatypeobject *) rhs;
    PyObject *result;
    int c, size1, size2;

    /* code to make sure that both arguments are of type
       newdatatype omitted */

    size1 = obj1->obj_UnderlyingDatatypePtr->size;
    size2 = obj2->obj_UnderlyingDatatypePtr->size;

    switch (op) {
    case Py_LT: c = size1 <  size2; break;
    case Py_LE: c = size1 <= size2; break;
    case Py_EQ: c = size1 == size2; break;
    case Py_NE: c = size1 != size2; break;
    case Py_GT: c = size1 >  size2; break;
    case Py_GE: c = size1 >= size2; break;
    }
    result = c ? Py_True : Py_False;
    return Py_NewRef(result);
}
```

3.5. Soporte de protocolo abstracto

Python admite una variedad de protocolos *abstractos*; las interfaces específicas proporcionadas para usar estas interfaces están documentadas en Capa de objetos abstractos.

Varias de estas interfaces abstractas se definieron temprano en el desarrollo de la implementación de Python. En particular, los protocolos de número, mapeo y secuencia han sido parte de Python desde el principio. Se han agregado otros protocolos con el tiempo. Para los protocolos que dependen de varias rutinas de controlador de la implementación de tipo, los protocolos más antiguos se han definido como bloques opcionales de controladores a los que hace referencia el objeto de tipo. Para los protocolos más nuevos, hay espacios adicionales en el objeto de tipo principal, con un bit de marca que se establece para indicar que los espacios están presentes y el intérprete debe verificarlos. (El bit de indicador no indica que los valores de intervalo no son `NULL`. El indicador puede establecerse para indicar la presencia de un intervalo, pero un intervalo aún puede estar vacío.):

```
PyNumberMethods   *tp_as_number;
PySequenceMethods *tp_as_sequence;
PyMappingMethods  *tp_as_mapping;
```

Si desea que su objeto pueda actuar como un número, una secuencia o un objeto de mapeo, entonces coloca la dirección de una estructura que implementa el tipo C **`PyNumberMethods`**, **`PySequenceMethods`**, o **`PyMappingMethods`**, respectivamente. Depende de usted completar esta estructura con los valores apropiados. Puede encontrar ejemplos del uso de cada uno de estos en el directorio `Objects` de la distribución fuente de Python.

```
hashfunc tp_hash;
```

Esta función, si elige proporcionarla, debería retornar un número hash para una instancia de su tipo de datos. Aquí hay un ejemplo simple:

```
static Py_hash_t
newdatatype_hash(PyObject *op)
{
    newdatatypeobject *self = (newdatatypeobject *) op;
    Py_hash_t result;
    result = self->some_size + 32767 * self->some_number
    if (result == -1) {
        result = -2;
    }
    return result;
}
```

`Py_hash_t` es un tipo entero con signo con un ancho que varia dependiendo de la plataforma.retornar `-1` de **`tp_hash`** indica un error, por lo que debe tener cuidado de evitar retornarlo cuando el cálculo de hash sea exitoso, como se vio anteriormente.

```
ternaryfunc tp_call;
```

Esta función se llama cuando una instancia de su tipo de datos se «llama», por ejemplo, si `obj1` es una instancia de su tipo de datos y el script de Python contiene `obj1('hello')`, el controlador **`tp_call`** se invoca.

Esta función toma tres argumentos:

1. *self* es la instancia del tipo de datos que es el sujeto de la llamada. Si la llamada es `obj1('hola')`, entonces *self* es `obj1`.
2. *args* es una tupla que contiene los argumentos de la llamada. Puede usar **`PyArg_ParseTuple()`** para extraer los argumentos.
3. *kwds* es un diccionario de argumentos de palabras clave que

se pasaron. Si no es `NULL` y admite argumentos de palabras clave, use **`PyArg_ParseTupleAndKeywords()`** para extraer los argumentos. Si no desea admitir argumentos de palabras clave y esto no es `NULL`, genere un **`TypeError`** con un mensaje que indique que los argumentos de palabras clave no son compatibles.

Aquí hay una implementación de juguete `tp_call`:

```
static PyObject *
newdatatype_call(PyObject *op, PyObject *args, PyObject
{
    newdatatypeobject *self = (newdatatypeobject *) op;
    PyObject *result;
    const char *arg1;
    const char *arg2;
    const char *arg3;

    if (!PyArg_ParseTuple(args, "sss:call", &arg1, &arg2
        return NULL;
    }
    result = PyUnicode_FromFormat(
        "Returning -- value: [%d] arg1: [%s] arg2: [%s]
        self->obj_UnderlyingDatatypePtr->size,
        arg1, arg2, arg3);
    return result;
}
```

```
/* Iterators */
getiterfunc tp_iter;
iternextfunc tp_iternext;
```

These functions provide support for the iterator protocol. Both handlers take exactly one parameter, the instance for which they are being called, and return a new reference. In the case of an error, they should set an exception and return `NULL`. **`tp_iter`** corresponds to the Python **`__iter__()`** method, while **`tp_iternext`** corresponds to the Python **`__next__()`** method.

Cualquier objeto iterable debe implementar el manejador `tp_iter`, que debe retornar un objeto iterator. Aquí se aplican las mismas pautas que para las clases de Python:

- Para colecciones (como listas y tuplas) que pueden admitir múltiples iteradores independientes, cada llamada debe crear y retornar un nuevo iterador a `tp_iter`.
- Los objetos que solo se pueden iterar una vez (generalmente debido a los efectos secundarios de la iteración, como los objetos de archivo) pueden implementar `tp_iter` retornando una nueva referencia a ellos mismos y, por lo tanto, también deben implementar el manejador `tp_iternext`.

Cualquier objeto iterator debe implementar tanto `tp_iter` como `tp_iternext`. El manejador de un iterador **`tp_iter`** debería retornar una nueva referencia al iterador. Su controlador **`tp_iternext`** debería retornar una nueva referencia al siguiente objeto en la iteración, si hay uno. Si la iteración ha llegado al final, **`tp_iternext`** puede retornar `NULL` sin establecer una excepción, o puede establecer `StopIteration` *además* para retornar `NULL`; evitar la excepción puede producir un rendimiento ligeramente mejor. Si se produce un error real, **`tp_iternext`** siempre debe establecer una excepción y retornar `NULL`.

3.6. Soporte de referencia débil

Uno de los objetivos de la implementación de referencia débil de Python es permitir que cualquier tipo participe en el mecanismo de referencia débil sin incurrir en la sobrecarga de objetos críticos para el rendimiento (como los números).

Ver también

Documentación para el módulo `weakref`.

For an object to be weakly referenceable, the extension type must set the `Py_TPFLAGS_MANAGED_WEAKREF` bit of the `tp_flags` field. The legacy `tp_weaklistoffset` field should be left as zero.

Concretely, here is how the statically declared type object would look:

```
static PyTypeObject TrivialType = {
    PyVarObject_HEAD_INIT(NULL, 0)
    /* ... other members omitted for brevity ... */
    .tp_flags = Py_TPFLAGS_MANAGED_WEAKREF | ...,
};
```

The only further addition is that `tp_dealloc` needs to clear any weak references (by calling **`PyObject_ClearWeakRefs()`**):

```
static void
Trivial_dealloc(PyObject *op)
{
    /* Clear weakrefs first before calling any destructo
    PyObject_ClearWeakRefs(op);
    /* ... remainder of destruction code omitted for bre
    Py_TYPE(op)->tp_free(op);
}
```

3.7. Más Sugerencias

Para aprender a implementar cualquier método específico para su nuevo tipo de datos, obtenga el código fuente CPython. Vaya al directorio `Objects`, luego busque en los archivos fuente C `tp_` más la función que desee (por ejemplo, `tp_richcompare`). Encontrará ejemplos de la función que desea implementar.

Cuando necesite verificar que un objeto es una instancia concreta del tipo que está implementando, use la función **`PyObject_TypeCheck()`**. Una muestra de su uso podría ser algo como lo siguiente:

```
if (!PyObject_TypeCheck(some_object, &MyType)) {
    PyErr_SetString(PyExc_TypeError, "arg #1 not a mythi
    return NULL;
}
```

Ver también

Descargue las versiones de origen de CPython.
https://www.python.org/downloads/source/

El proyecto CPython en GitHub, donde se desarrolla el código fuente de CPython.
https://github.com/python/cpython

4. Construyendo extensiones C y C++

A C extension for CPython is a shared library (for example, a `.so` file on Linux, `.pyd` on Windows), which exports an *initialization function*.

See Defining extension modules for details.

4.1. Building C and C++ Extensions with setuptools

Building, packaging and distributing extension modules is best done with third-party tools, and is out of scope of this document. One suitable tool is Setuptools, whose documentation can be found at https://setuptools.pypa.io/en/latest/setuptools.html.

The **`distutils`** module, which was included in the standard library until Python 3.12, is now maintained as part of Setuptools.

5. Creación de extensiones C y C++ en Windows

Este capítulo explica brevemente cómo crear un módulo de extensión de Windows para Python usando Microsoft Visual C++, y sigue con información de fondo más detallada sobre cómo funciona. El material explicativo es útil tanto para el programador de Windows que está aprendiendo a construir extensiones de Python como para el programador de Unix interesado en producir software que se pueda construir con éxito tanto en Unix como en Windows.

Se alienta a los autores de módulos a utilizar el enfoque distutils para construir módulos de extensión, en lugar del descrito en esta sección. Aún necesitará el compilador de C que se utilizó para construir Python; típicamente Microsoft Visual C++.

Nota

Este capítulo menciona varios nombres de archivo que incluyen un número de versión codificado de Python. Estos nombres de archivo se representan con el número de versión que se muestra como `XY`; en la práctica, `'X'` será el número de versión principal y `'Y'` será el número de versión menor de la versión de Python con la que está trabajando. Por ejemplo, si está utilizando Python 2.2.1, `XY` en realidad será `22`.

5.1. Un enfoque de libro de cocina

There are two approaches to building extension modules on Windows, just as there are on Unix: use the `setuptools` package to control the build process, or do things manually. The setuptools approach works well for most extensions; documentation on using `setuptools` to build and package extension modules is available in Building C and C++ Extensions with setuptools. If you find you really need to do things manually, it may be instructive to study the project file for the winsound [https://github.com/python/cpython/tree/3.14/PCbuild/winsound.vcxproj] standard library module.

5.2. Diferencias entre Unix y Windows

Unix y Windows usan paradigmas completamente diferentes para la carga de código en tiempo de ejecución. Antes de intentar construir un módulo que se pueda cargar dinámicamente, tenga en cuenta cómo funciona su sistema.

En Unix, un archivo de objeto compartido (`.so`) contiene código para ser utilizado por el programa, y también los nombres de funciones y datos que espera encontrar en el programa. Cuando el archivo se une al programa, todas las referencias a esas funciones y datos en el código del archivo se cambian para apuntar a las ubicaciones reales en el programa donde las funciones y los datos se colocan en la memoria. Esto es básicamente una operación de enlace.

En Windows, un archivo de biblioteca de enlace dinámico (`.dll`) no tiene referencias colgantes. En cambio, un acceso a funciones o datos pasa por una tabla de búsqueda. Por lo tanto, el código DLL no tiene que repararse en tiempo de ejecución para referirse a la memoria del programa; en cambio, el código ya usa la tabla de búsqueda de la DLL, y la tabla de búsqueda se modifica en tiempo de ejecución para apuntar a las funciones y los datos.

En Unix, solo hay un tipo de archivo de biblioteca (`.a`) que contiene código de varios archivos de objeto (`.o`). Durante el paso de enlace para crear un archivo de objeto compartido (`.so`), el enlazador puede encontrar que no sabe dónde se define un identificador. El enlazador lo buscará en los archivos de objetos en las bibliotecas; si lo encuentra, incluirá todo el código de ese archivo de objeto.

En Windows, hay dos tipos de biblioteca, una biblioteca estática y una biblioteca de importación (ambas llamadas `.lib`). Una biblioteca estática es como un archivo Unix `.a`; Contiene código para ser incluido según sea necesario. Una biblioteca de importación se usa básicamente solo para asegurarle al enlazador que cierto identificador es legal y estará presente en el programa cuando se cargue la DLL. Por lo tanto, el enlazador utiliza la información de la biblioteca de importación para crear la tabla de búsqueda para usar identificadores que no están incluidos en la DLL. Cuando se vincula una aplicación o una DLL, se puede generar una biblioteca de importación, que deberá usarse para todas las DLL futuras que dependan de los símbolos en la aplicación o DLL.

Suponga que está creando dos módulos de carga dinámica, B y C, que deberían compartir otro bloque de código A. En Unix, *no* pasaría `A.a` al enlazador para `B.so` y `C.so`; eso haría que se incluyera dos veces, de modo que B y C tengan cada uno su propia copia. En Windows, compilar `A.dll` también compilará `A.lib`. Usted *si* pasa `A.lib` al enlazador para B y C. `A.lib` no contiene código; solo contiene información que se usará en tiempo de ejecución para acceder al código de A.

En Windows, usar una biblioteca de importación es como usar `importar spam`; le da acceso a los nombres de spam, pero no crea una copia separada. En Unix, vincular con una biblioteca es más como `from spam import*`; crea una copia separada.

Py_NO_LINK_LIB

> Turn off the implicit, `#pragma`-based linkage with the Python library, performed inside CPython header files.
>
> *Added in version 3.14.*

5.3. Usar DLL en la práctica

Windows Python is built in Microsoft Visual C++; using other compilers may or may not work. The rest of this section is MSVC++ specific.

When creating DLLs in Windows, you can use the CPython library in two ways:

1. By default, inclusion of `PC/pyconfig.h` directly or via

`Python.h` triggers an implicit, configure-aware link with the library. The header file chooses `pythonXY_d.lib` for Debug, `pythonXY.lib` for Release, and `pythonX.lib` for Release with the Limited API enabled.

To build two DLLs, spam and ni (which uses C functions found in spam), you could use these commands:

```
cl /LD /I/python/include spam.c
cl /LD /I/python/include ni.c spam.lib
```

The first command created three files: `spam.obj`, `spam.dll` and `spam.lib`. `Spam.dll` does not contain any Python functions (such as **`PyArg_ParseTuple()`**), but it does know how to find the Python code thanks to the implicitly linked `pythonXY.lib`.

El segundo comando creó `ni.dll` (y `.obj` y `.lib`), que sabe cómo encontrar las funciones necesarias del spam, y también del ejecutable de Python.

2. Manually by defining **`Py_NO_LINK_LIB`** macro before including `Python.h`. You must pass `pythonXY.lib` to the linker.

   To build two DLLs, spam and ni (which uses C functions found in spam), you could use these commands:

   ```
   cl /LD /DPy_NO_LINK_LIB /I/python/include spam.c ..
   cl /LD /DPy_NO_LINK_LIB /I/python/include ni.c spam
   ```

   El primer comando creó tres archivos: `spam.obj`, `spam.dll` y `spam.lib`. `Spam.dll` no contiene ninguna función de Python (como **`PyArg_ParseTuple()`**), pero sabe cómo encontrar el código de Python gracias a `pythonXY.lib`.

   El segundo comando creó `ni.dll` (y `.obj` y `.lib`), que sabe cómo encontrar las funciones necesarias del spam, y también del ejecutable de Python.

No todos los identificadores se exportan a la tabla de búsqueda. Si desea que cualquier otro módulo (incluido Python) pueda ver sus identificadores, debe decir `_declspec(dllexport)`, como en `void _declspec(dllexport) initspam(void)` o `PyObject_declspec(dllexport) *NiGetSpamData(void)`.

Developer Studio will throw in a lot of import libraries that you do not really need, adding about 100K to your executable. To get rid of them, use the Project Settings dialog, Link tab, to specify *ignore default libraries*. Add the correct `msvcrtxx.lib` to the list of libraries.

1. Incrustando Python en otra aplicación

Los capítulos anteriores discutieron cómo extender Python, es decir, cómo extender la funcionalidad de Python al adjuntarle una biblioteca de funciones C. También es posible hacerlo al revés: enriquezca su aplicación C/C++ incrustando Python en ella. La incrustación proporciona a su aplicación la capacidad de implementar parte de la funcionalidad de su aplicación en Python en lugar de C o C++. Esto se puede usar para muchos propósitos; Un ejemplo sería permitir a los usuarios adaptar la aplicación a sus necesidades escribiendo algunos scripts en Python. También puede usarlo usted mismo si parte de la funcionalidad se puede escribir en Python más fácilmente.

Incrustar Python es similar a extenderlo, pero no del todo. La diferencia es que cuando extiende Python, el programa principal de la aplicación sigue siendo el intérprete de Python, mientras que si incrusta Python, el programa principal puede no tener nada que ver con Python — en cambio, algunas partes de la aplicación ocasionalmente llaman al Intérprete de Python para ejecutar algún código de Python.

Entonces, si está incrustando Python, está proporcionando su propio programa principal. Una de las cosas que tiene que hacer este programa principal es inicializar el intérprete de Python. Como mínimo, debe llamar a la función `Py_Initialize()`. Hay llamadas opcionales para pasar argumentos de línea de comandos a Python. Luego, puede llamar al intérprete desde cualquier parte de la aplicación.

Hay varias formas diferentes de llamar al intérprete: puede pasar una cadena que contiene declaraciones de Python a `PyRun_SimpleString()`, o puede pasar un puntero de archivo estándar y un nombre de archivo (solo para identificación en mensajes de error) a `PyRun_SimpleFile()`. También puede llamar a las operaciones de nivel inferior descritas en los capítulos anteriores para construir y usar objetos de Python.

Ver también

Python/C API reference manual
Los detalles de la interfaz C de Python se dan en este manual. Una gran cantidad de información necesaria se puede encontrar aquí.

1.1. Incrustación de muy alto nivel

La forma más simple de incrustar Python es el uso de la interfaz de muy alto nivel. Esta interfaz está diseñada para ejecutar un script de Python sin necesidad de interactuar directamente con la aplicación. Esto puede usarse, por ejemplo, para realizar alguna operación en un archivo.

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>

int
main(int argc, char *argv[])
{
    PyStatus status;
    PyConfig config;
    PyConfig_InitPythonConfig(&config);

    /* optional but recommended */
    status = PyConfig_SetBytesString(&config, &config.pr
    if (PyStatus_Exception(status)) {
        goto exception;
    }

    status = Py_InitializeFromConfig(&config);
    if (PyStatus_Exception(status)) {
        goto exception;
    }
    PyConfig_Clear(&config);

    PyRun_SimpleString("from time import time,ctime\n"
                       "print('Today is', ctime(time()))
    if (Py_FinalizeEx() < 0) {
        exit(120);
    }
    return 0;

  exception:
     PyConfig_Clear(&config);
     Py_ExitStatusException(status);
}
```

Nota

`#define PY_SSIZE_T_CLEAN` se usaba para indicar que `Py_ssize_t` debería usarse en algunas APIs en lugar de `int`. No es necesario desde Python 3.13, pero lo mantenemos aquí por compatibilidad. Ver Cadena de caracteres y búferes para una descripción de esta macro.

La configuración **`PyConfig.program_name`** debe llamarse antes de **`Py_InitializeFromConfig()`** para informar al intérprete sobre las rutas a las bibliotecas de tiempo de ejecución de Python. A continuación, el intérprete de Python se inicializa con **`Py_Initialize()`**, seguido de la ejecución de un script Python codificado que imprime la fecha y la hora. Luego, la llamada **`Py_FinalizeEx()`** cierra el intérprete, seguido por el final del programa. En un programa real, es posible que desee obtener el script de Python de otra fuente, tal vez una rutina de editor de texto, un archivo o una base de datos. Obtener el código Python de un archivo se puede hacer mejor usando la función **`PyRun_SimpleFile()`**, que le ahorra la molestia de asignar espacio de memoria y cargar el contenido del archivo.

1.2. Más allá de la incrustación de muy

alto nivel: una visión general

La interfaz de alto nivel le permite ejecutar piezas arbitrarias de código Python desde su aplicación, pero el intercambio de valores de datos es bastante engorroso, por decir lo menos. Si lo desea, debe usar llamadas de nivel inferior. A costa de tener que escribir más código C, puede lograr casi cualquier cosa.

Cabe señalar que extender Python e incrustar Python es la misma actividad, a pesar de la intención diferente. La mayoría de los temas tratados en los capítulos anteriores siguen siendo válidos. Para mostrar esto, considere lo que realmente hace el código de extensión de Python a C:

1. Convierte valores de datos de Python a C,
2. Realice una llamada de función a una rutina C usando los valores convertidos, y
3. Convierte los valores de datos de la llamada de C a Python.

Al incrustar Python, el código de interfaz hace:

1. Convierte valores de datos de C a Python,
2. Realice una llamada de función a una rutina de interfaz de Python utilizando los valores convertidos, y
3. Convierte los valores de datos de la llamada de Python a C.

Como puede ver, los pasos de conversión de datos simplemente se intercambian para acomodar la dirección diferente de la transferencia de idiomas cruzados. La única diferencia es la rutina que llama entre ambas conversiones de datos. Al extender, llama a una rutina C, al incrustar, llama a una rutina Python.

Este capítulo no discutirá cómo convertir datos de Python a C y viceversa. Además, se supone que se entiende el uso adecuado de las referencias y el tratamiento de errores. Dado que estos aspectos no difieren de extender el intérprete, puede consultar los capítulos anteriores para obtener la información requerida.

1.3. Incrustación pura

El primer programa tiene como objetivo ejecutar una función en un script Python. Al igual que en la sección sobre la interfaz de muy alto nivel, el intérprete de Python no interactúa directamente con la aplicación (pero eso cambiará en la siguiente sección).

El código para ejecutar una función definida en un script de Python es:

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>

int
main(int argc, char *argv[])
{
    PyObject *pName, *pModule, *pFunc;
    PyObject *pArgs, *pValue;
    int i;

    if (argc < 3) {
        fprintf(stderr,"Usage: call pythonfile funcname
        return 1;
    }

    Py_Initialize();
    pName = PyUnicode_DecodeFSDefault(argv[1]);
    /* Error checking of pName left out */

    pModule = PyImport_Import(pName);
    Py_DECREF(pName);

    if (pModule != NULL) {
        pFunc = PyObject_GetAttrString(pModule, argv[2])
        /* pFunc is a new reference */

        if (pFunc && PyCallable_Check(pFunc)) {
            pArgs = PyTuple_New(argc - 3);
            for (i = 0; i < argc - 3; ++i) {
                pValue = PyLong_FromLong(atoi(argv[i + 
                if (!pValue) {
                    Py_DECREF(pArgs);
                    Py_DECREF(pModule);
                    fprintf(stderr, "Cannot convert argu
                    return 1;
                }
                /* pValue reference stolen here: */
                PyTuple_SetItem(pArgs, i, pValue);
            }
            pValue = PyObject_CallObject(pFunc, pArgs);
            Py_DECREF(pArgs);
            if (pValue != NULL) {
                printf("Result of call: %ld\n", PyLong_A
                Py_DECREF(pValue);
            }
            else {
                Py_DECREF(pFunc);
                Py_DECREF(pModule);
                PyErr_Print();
                fprintf(stderr,"Call failed\n");
                return 1;
            }
        }
        else {
            if (PyErr_Occurred())
                PyErr_Print();
            fprintf(stderr, "Cannot find function \"%s\"
        }
        Py_XDECREF(pFunc);
        Py_DECREF(pModule);
    }
    else {
        PyErr_Print();
        fprintf(stderr, "Failed to load \"%s\"\n", argv[
        return 1;
    }
    if (Py_FinalizeEx() < 0) {
        return 120;
    }
    return 0;
}
```

Este código carga un script de Python usando `argv[1]` y llama a la función nombrada en `argv[2]`. Sus argumentos enteros son los otros valores del arreglo `argv`. Si usted compila y enlaza este programa (llamemos al ejecutable terminado **call**), y úselo para ejecutar un script Python, como:

```
def multiply(a,b):
    print("Will compute", a, "times", b)
    c = 0
    for i in range(0, a):
        c = c + b
    return c
```

entonces el resultado debería ser:

```
$ call multiply multiply 3 2
Will compute 3 times 2
Result of call: 6
```

Aunque el programa es bastante grande por su funcionalidad, la mayor parte del código es para la conversión de datos entre Python y C, y para informes de errores. La parte interesante con respecto a incrustar Python comienza con:

```
Py_Initialize();
pName = PyUnicode_DecodeFSDefault(argv[1]);
/* Error checking of pName left out */
pModule = PyImport_Import(pName);
```

After initializing the interpreter, the script is loaded using **`PyImport_Import()`**. This routine needs a Python string as its argument, which is constructed using the **`PyUnicode_DecodeFSDefault()`** data conversion routine.

```
pFunc = PyObject_GetAttrString(pModule, argv[2]);
/* pFunc is a new reference */
```

```
if (pFunc && PyCallable_Check(pFunc)) {
    ...
}
Py_XDECREF(pFunc);
```

Una vez que se carga el script, el nombre que estamos buscando se recupera usando `PyObject_GetAttrString()`. Si el nombre existe y el objeto retornado es invocable, puede asumir con seguridad que es una función. Luego, el programa continúa construyendo una tupla de argumentos como de costumbre. La llamada a la función Python se realiza con:

```
pValue = PyObject_CallObject(pFunc, pArgs);
```

Al regresar la función, `pValue` es `NULL` o contiene una referencia al valor de retorno de la función. Asegúrese de liberar la referencia después de examinar el valor.

1.4. Extendiendo Python incrustado

Hasta ahora, el intérprete de Python incorporado no tenía acceso a la funcionalidad de la aplicación misma. La API de Python lo permite al extender el intérprete incorporado. Es decir, el intérprete incorporado se amplía con las rutinas proporcionadas por la aplicación. Si bien suena complejo, no es tan malo. Simplemente olvide por un momento que la aplicación inicia el intérprete de Python. En cambio, considere que la aplicación es un conjunto de subrutinas y escriba un código de pegamento que le otorgue a Python acceso a esas rutinas, al igual que escribiría una extensión normal de Python. Por ejemplo:

```
static int numargs=0;

/* Return the number of arguments of the application com
static PyObject*
emb_numargs(PyObject *self, PyObject *args)
{
    if(!PyArg_ParseTuple(args, ":numargs"))
        return NULL;
    return PyLong_FromLong(numargs);
}

static PyMethodDef emb_module_methods[] = {
    {"numargs", emb_numargs, METH_VARARGS,
     "Return the number of arguments received by the pro
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef emb_module = {
    .m_base = PyModuleDef_HEAD_INIT,
    .m_name = "emb",
    .m_size = 0,
    .m_methods = emb_module_methods,
};

static PyObject*
PyInit_emb(void)
{
    return PyModuleDef_Init(&emb_module);
}
```

Inserte el código anterior justo encima de la función **`main()`**. Además, inserte las siguientes dos declaraciones antes de la llamada a **`Py_Initialize()`**:

```
numargs = argc;
PyImport_AppendInittab("emb", &PyInit_emb);
```

Estas dos líneas inicializan la variable `numargs` y hacen que la función **`emb.numargs()`** sea accesible para el intérprete de Python incorporado. Con estas extensiones, el script de Python puede hacer cosas como

```
import emb
print("Number of arguments", emb.numargs())
```

En una aplicación real, los métodos expondrán una API de la aplicación a Python.

1.5. Incrustando Python en C+ +

También es posible incrustar Python en un programa C+ +; precisamente cómo se hace esto dependerá de los detalles del sistema C+ + utilizado; en general, necesitará escribir el programa principal en C+ + y usar el compilador de C+ + para compilar y vincular su programa. No es necesario volver a compilar Python usando C+ +.

1.6. Compilar y enlazar bajo sistemas tipo Unix

No es necesariamente trivial encontrar los indicadores correctos para pasar a su compilador (y enlazador) para incrustar el intérprete de Python en su aplicación, particularmente porque Python necesita cargar módulos de biblioteca implementados como extensiones dinámicas en C (archivos `.so`) enlazados en su contra.

Para conocer los indicadores necesarios del compilador y el enlazador, puede ejecutar el script `python`*`X.Y`*`-config` que se genera como parte del proceso de instalación (también puede estar disponible un script `python3-config`) Este script tiene varias opciones, de las cuales las siguientes serán directamente útiles para usted:

- `pythonX.Y-config --cflags` le dará las banderas recomendadas al compilar:

  ```
  $ /opt/bin/python3.11-config --cflags
  -I/opt/include/python3.11 -I/opt/include/python3.11
  ```

- `pythonX.Y-config --ldflags --embed` le dará las banderas recomendadas al vincular:

  ```
  $ /opt/bin/python3.11-config --ldflags --embed
  -L/opt/lib/python3.11/config-3.11-x86_64-linux-gnu
  ```

Nota

> Para evitar confusiones entre varias instalaciones de Python (y especialmente entre el sistema Python y su propio Python compilado), se recomienda que use la ruta absoluta a `pythonX.Y-config`, como en el ejemplo anterior.

Si este procedimiento no funciona para usted (no se garantiza que funcione para todas las plataformas tipo Unix; sin embargo, le damos la bienvenida informes de errores) deberá leer la documentación de su sistema sobre dinámica vincular o examinar Python `Makefile` (use **`sysconfig.get_makefile_filename()`** para encontrar su ubicación) y las opciones de compilación. En este caso, el módulo **`sysconfig`** es una herramienta útil para extraer mediante programación los valores de configuración que querrá combinar. Por ejemplo:

```
>>> import sysconfig
>>> sysconfig.get_config_var('LIBS')
'-lpthread -ldl  -lutil'
>>> sysconfig.get_config_var('LINKFORSHARED')
'-Xlinker -export-dynamic'
```

Python/C API reference manual

Este manual documenta la API utilizada por los programadores de C y C + + que desean escribir módulos de extensión o incorporar Python. Es un complemento de Ampliación e incrustación del intérprete de Python, que describe los principios generales de la escritura de extensión pero no documenta las funciones API en detalle.

- Introducción
    - Language version compatibility
    - Estándares de codificación
    - Archivos de cabecera (*Include*)
    - Macros útiles
    - Objetos, tipos y conteos de referencias
    - Excepciones
    - Integración de Python
    - Depuración de compilaciones
    - Recommended third party tools
- Estabilidad de la API en C
    - Unstable C API
    - Interfaz binaria de aplicación estable
    - Consideraciones de la plataforma
    - Contenido de la API limitada
- La capa de muy alto nivel
    - Available start symbols
    - Stack Effects
- Conteo de referencias
- Manejo de excepciones
    - Impresión y limpieza

  - Lanzando excepciones
  - Emitir advertencias
  - Consultando el indicador de error
  - Manejo de señal
  - Clases de excepción
  - Objetos excepción
  - Objetos unicode de excepción
  - Control de recursión
  - Exception and warning types
  - Tracebacks

- Defining extension modules

  - Multiple module instances
  - Initialization function
  - Multi-phase initialization
  - Legacy single-phase initialization

- Utilidades

  - Utilidades del sistema operativo
  - Funciones del Sistema
  - Control de procesos
  - Importando módulos
  - Soporte de empaquetado (*marshalling*) de datos
  - Analizando argumentos y construyendo valores
  - Conversión y formato de cadenas de caracteres
  - Character classification and conversion
  - PyHash API
  - Reflexión
  - Registro de códec y funciones de soporte
  - PyTime C API
  - Soporte para Mapeo Perf

- Capa de objetos abstractos

  - Protocolo de objeto
  - Protocolo de llamada
  - Protocolo de números
  - Protocolo de secuencia
  - Protocolo de mapeo

  - Protocolo iterador
  - Protocolo búfer

- Capa de objetos concretos

  - Objetos fundamentales
  - Objetos numéricos
  - Objetos de secuencia
  - Objetos contenedor
  - Objetos de función
  - Otros objetos
  - C API for extension modules

- Interpreter initialization and finalization

  - Before Python initialization
  - Global configuration variables
  - Initializing and finalizing the interpreter
  - Cautions regarding runtime finalization
  - Process-wide parameters

- Thread states and the global interpreter lock

  - Detaching the thread state from extension code
  - Non-Python created threads
  - Legacy API
  - Cautions about fork()
  - High-level APIs
  - GIL-state APIs
  - Low-level APIs

- Asynchronous notifications
- Operating system thread APIs
- Synchronization primitives

  - Python critical section API
  - Legacy locking APIs

- Thread-local storage support

  - Thread-specific storage API
  - Dynamic allocation

  - Methods
  - Legacy APIs
- Multiple interpreters in a Python process
  - A per-interpreter GIL
  - Bugs and caveats
  - High-level APIs
  - Low-level APIs
  - Advanced debugger support
- Profiling and tracing
- Reference tracing
- Configuración de inicialización de Python
  - PyInitConfig C API
  - Configuration Options
  - Runtime Python configuration API
  - PyConfig C API
  - Py_GetArgcArgv()
  - Delaying main module execution
- Gestión de la memoria
  - Visión general
  - Dominios del asignador
  - Interfaz de memoria sin procesar
  - Interfaz de memoria
  - Asignadores de objetos
  - Asignadores de memoria predeterminados
  - Personalizar asignadores de memoria
  - Configurar enlaces para detectar errores en las funciones del asignador de memoria de Python
  - El asignador pymalloc
  - El asignador mimalloc
  - tracemalloc C API
  - Ejemplos
- Soporte de implementación de objetos
  - Allocating objects on the heap

    - Object Life Cycle
    - Estructuras de objetos comunes
    - Type Object Structures
    - Apoyo a la recolección de basura cíclica

- Versiones de API y ABI

    - Build-time version constants
    - Run-time version
    - Bit-packing macros

- Monitoring C API
- Generating Execution Events

    - Managing the Monitoring State

Introducción

La interfaz del programador de aplicaciones (API) con Python brinda a los programadores de C y C++ acceso al intérprete de Python en una variedad de niveles. La API es igualmente utilizable desde C++, pero por brevedad generalmente se conoce como la API Python/C. Hay dos razones fundamentalmente diferentes para usar la API Python/C. La primera razón es escribir *módulos de extensión* para propósitos específicos; Estos son módulos C que extienden el intérprete de Python. Este es probablemente el uso más común. La segunda razón es usar Python como componente en una aplicación más grande; Esta técnica se conoce generalmente como integración (*embedding*) Python en una aplicación.

Escribir un módulo de extensión es un proceso relativamente bien entendido, donde un enfoque de «libro de cocina» (*cookbook*) funciona bien. Hay varias herramientas que automatizan el proceso hasta cierto punto. Si bien las personas han integrado Python en otras aplicaciones desde su existencia temprana, el proceso de integrar Python es menos sencillo que escribir una extensión.

Muchas funciones API son útiles independientemente de si está integrando o extendiendo Python; Además, la mayoría de las aplicaciones que integran Python también necesitarán proporcionar una extensión personalizada, por lo que probablemente sea una buena idea familiarizarse con la escritura de una extensión antes de intentar integrar Python en una aplicación real.

Language version compatibility

Python's C API is compatible with C11 and C++11 versions of C and C++.

This is a lower limit: the C API does not require features from later C/C++ versions. You do *not* need to enable your compiler's «c11 mode».

Estándares de codificación

Si está escribiendo código C para su inclusión en CPython, **debe** seguir las pautas y estándares definidos en **PEP 7** [https://peps.python.org/pep-0007/]. Estas pautas se aplican independientemente de la versión de Python a la que esté contribuyendo. Seguir estas convenciones no es necesario para sus propios módulos de extensión de terceros, a menos que eventualmente espere contribuir con ellos a Python.

Archivos de cabecera (*Include*)

Todas las definiciones de función, tipo y macro necesarias para usar la API Python/C se incluyen en su código mediante la siguiente línea:

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>
```

Esto implica la inclusión de los siguientes archivos de encabezado estándar: `<stdio.h>`, `<string.h>`, `<errno.h>`, `<limits.h>`, `<assert.h>` y `<stdlib.h>` (si está disponible).

Nota

Dado que Python puede definir algunas definiciones de preprocesador que afectan los encabezados estándar en algunos sistemas, *debe* incluir `Python.h` antes de incluir encabezados estándar.

Se recomienda definir siempre `PY_SSIZE_T_CLEAN` antes de incluir `Python.h`. Consulte Analizando argumentos y construyendo valores para obtener una descripción de este macro.

Todos los nombres visibles del usuario definidos por `Python.h` (excepto los definidos por los encabezados estándar incluidos) tienen uno de los prefijos `Py` o `_Py`. Los nombres que comienzan con `_Py` son para uso interno de la implementación de Python y no deben ser utilizados por escritores de extensiones. Los nombres de miembros de estructura no tienen un prefijo reservado.

Nota

El código de usuario nunca debe definir nombres que comiencen con `Py` o `_Py`. Esto confunde al lector y pone en peligro la portabilidad del código de usuario para futuras versiones de Python, que pueden definir nombres adicionales que comienzan con uno de estos prefijos.

The header files are typically installed with Python. On Unix, these are located in the directories `prefix/include/pythonversion/` and `exec_prefix/include/pythonversion/`, where **`prefix`** and **`exec_prefix`** are defined by the corresponding parameters to Python's **configure** script and *version* is `'%d.%d' % sys.version_info[:2]`. On Windows, the headers are installed in `prefix/include`, where `prefix` is the installation directory specified to the installer.

To include the headers, place both directories (if different) on your compiler's search path for includes. Do *not* place the parent directories on the search path and then use `#include <pythonX.Y/Python.h>`; this will break on multi-platform builds since the platform independent headers under **`prefix`** include the platform specific headers from **`exec_prefix`**.

Los usuarios de C++ deben tener en cuenta que aunque la API se define completamente usando C, los archivos de encabezado declaran correctamente que los puntos de entrada son `extern "C"`. Como resultado, no es necesario hacer nada especial para usar la API desde C++.

Macros útiles

Several useful macros are defined in the Python header files. Many are defined closer to where they are useful (for example, **`Py_RETURN_NONE`**, **`PyMODINIT_FUNC`**). Others of a more general utility are defined here. This is not necessarily a complete listing.

Py_CAN_START_THREADS

If this macro is defined, then the current system is able to start threads.

Currently, all systems supported by CPython (per **PEP 11** [https://peps.python.org/pep-0011/]), with the exception of some WebAssembly platforms, support starting threads.

Added in version 3.13.

Py_GETENV(s)

Like `getenv(s)`, but returns `NULL` if **`-E`** was passed on the command line (see **`PyConfig.use_environment`**).

Docstring macros

PyDoc_STRVAR(name, str)

Creates a variable with name *name* that can be used in docstrings. If Python is built without docstrings (**`--without-doc-strings`**), the value will be an empty string.

Ejemplo:

```
PyDoc_STRVAR(pop_doc, "Remove and return the rightm

static PyMethodDef deque_methods[] = {
    // ...
    {"pop", (PyCFunction)deque_pop, METH_NOARGS, po
    // ...
}
```

Expands to `PyDoc_VAR(name) = PyDoc_STR(str)`.

PyDoc_STR(str)

Expands to the given input string, or an empty string if docstrings are disabled (**`--without-doc-strings`**).

Ejemplo:

```
static PyMethodDef pysqlite_row_methods[] = {
    {"keys", (PyCFunction)pysqlite_row_keys, METH_N
        PyDoc_STR("Returns the keys of the row.")},
    {NULL, NULL}
};
```

PyDoc_VAR(name)

Declares a static character array variable with the given *name*. Expands to `static const char name[]`

For example:

```
PyDoc_VAR(python_doc) = PyDoc_STR(
   "A genus of constricting snakes in the Pythonida
   "to the tropics and subtropics of the Eastern He
```

General utility macros

The following macros are for common tasks not specific to Python.

Py_UNUSED(arg)

Use esto para argumentos no utilizados en una definición de función para silenciar las advertencias del compilador. Ejemplo: `int func(int a, int Py_UNUSED(b)) {return a; }`.

Added in version 3.4.

Py_GCC_ATTRIBUTE(name)

Use a GCC attribute *name*, hiding it from compilers that don't support GCC attributes (such as MSVC).

This expands to `__attribute__((name))` on a GCC compiler, and expands to nothing on compilers that don't support GCC attributes.

Numeric utilities

Py_ABS(x)

Retorna el valor absoluto de `x`.

The argument may be evaluated more than once. Consequently, do not pass an expression with side-effects directly to this macro.

If the result cannot be represented (for example, if `x` has **INT_MIN** value for int type), the behavior is undefined.

Corresponds roughly to `((x) < 0 ? -(x) : (x))`

Added in version 3.3.

Py_MAX(x, y)
Py_MIN(x, y)

Return the larger or smaller of the arguments, respectively.

Any arguments may be evaluated more than once. Consequently, do not pass an expression with side-effects directly to this macro.

Py_MAX corresponds roughly to `(((x) > (y)) ? (x) : (y))`.

Added in version 3.3.

Py_ARITHMETIC_RIGHT_SHIFT(type, integer, positions)

Similar to `integer >> positions`, but forces sign extension, as the C standard does not define whether a right-shift of a signed integer will perform sign extension or a zero-fill.

integer should be any signed integer type. *positions* is the number of positions to shift to the right.

Both *integer* and *positions* can be evaluated more than once; consequently, avoid directly passing a function call or some other operation with side-effects to this macro. Instead, store the result as a variable and then pass it.

type is unused and only kept for backwards compatibility. Historically, *type* was used to cast *integer*.

Distinto en la versión 3.1: This macro is now valid for all signed integer types, not just those for which `unsigned type` is legal. As a result, *type* is no longer used.

Py_CHARMASK(c)

El argumento debe ser un carácter o un número entero en el rango [-128, 127] o [0, 255]. Este macro retorna la conversión `c` a un `unsigned char`.

Assertion utilities

Py_UNREACHABLE()

Use esto cuando tenga una ruta de código a la que no se pueda acceder por diseño. Por ejemplo, en la cláusula `default:` en una declaración `switch` para la cual todos los valores posibles están cubiertos en declaraciones `case`. Use esto en lugares donde podría tener la tentación de poner una llamada `assert(0)` o `abort()`.

En el modo de lanzamiento, la macro ayuda al compilador a optimizar el código y evita una advertencia sobre el código inalcanzable. Por ejemplo, la macro se implementa con `__builtin_unreachable()` en GCC en modo de lanzamiento.

In debug mode, and on unsupported compilers, the macro expands to a call to **`Py_FatalError()`**.

A use for `Py_UNREACHABLE()` is following a call to a function that never returns but that is not declared `_Noreturn`.

Si una ruta de código es un código muy poco probable pero se puede acceder en casos excepcionales, esta macro no debe utilizarse. Por ejemplo, en condiciones de poca memoria o si una llamada al sistema retorna un valor fuera del rango esperado. En este caso, es mejor informar el error a la persona

que llama. Si no se puede informar del error a la persona que llama, se puede utilizar **`Py_FatalError()`**.

Added in version 3.7.

Py_SAFE_DOWNCAST(value, larger, smaller)

Cast *value* to type *smaller* from type *larger*, validating that no information was lost.

On release builds of Python, this is roughly equivalent to `((smaller) value)` (in C++, `static_cast<smaller>(value)` will be used instead).

On debug builds (implying that **`Py_DEBUG`** is defined), this asserts that no information was lost with the cast from *larger* to *smaller*.

value, *larger*, and *smaller* may all be evaluated more than once in the expression; consequently, do not pass an expression with side-effects directly to this macro.

Py_BUILD_ASSERT(cond)

Asserts a compile-time condition *cond*, as a statement. The build will fail if the condition is false or cannot be evaluated at compile time.

Corresponds roughly to `static_assert(cond)` on C23 and above.

For example:

```
Py_BUILD_ASSERT(sizeof(PyTime_t) == sizeof(int64_t)
```

Added in version 3.3.

Py_BUILD_ASSERT_EXPR(cond)

Asserts a compile-time condition *cond*, as an expression that evaluates to `0`. The build will fail if the condition is false or cannot be evaluated at compile time.

For example:

```
#define foo_to_char(foo) \
    ((char *)(foo) + Py_BUILD_ASSERT_EXPR(offsetof(
```

Added in version 3.3.

Type size utilities

Py_ARRAY_LENGTH(array)

Compute the length of a statically allocated C array at compile time.

The *array* argument must be a C array with a size known at compile time. Passing an array with an unknown size, such as a heap-allocated array, will result in a compilation error on some compilers, or otherwise produce incorrect results.

This is roughly equivalent to:

```
sizeof(array) / sizeof((array)[0])
```

Py_MEMBER_SIZE(type, member)

Return the size of a structure (*type*) *member* in bytes.

Corresponds roughly to `sizeof(((type *)NULL)->member)`.

Added in version 3.6.

Macro definition utilities

Py_FORCE_EXPANSION(X)

This is equivalent to `X`, which is useful for token-pasting in macros, as macro expansions in *X* are forcefully evaluated by the preprocessor.

Py_STRINGIFY(x)

Convert `x` to a C string. For example, `Py_STRINGIFY(123)`

returns `"123"`.

Added in version 3.4.

Declaration utilities

The following macros can be used in declarations. They are most useful for defining the C API itself, and have limited use for extension authors. Most of them expand to compiler-specific spellings of common extensions to the C language.

Py_ALWAYS_INLINE

Ask the compiler to always inline a static inline function. The compiler can ignore it and decide to not inline the function.

Corresponds to `always_inline` attribute in GCC and `__forceinline` in MSVC.

Puede ser usado para usar inline en funciones estáticas inline de rendimiento crítico cuando se corre Python en modo de depuración con inline de funciones deshabilitado. Por ejemplo, MSC deshabilita el inline de funciones cuando se configura en modo de depuración.

Marcar ciegamente una función estática inline con Py_ALWAYS_INLINE puede resultar en peor rendimientos (debido a un aumento del tamaño del código, por ejemplo). El compilador es generalmente más inteligente que el desarrollador para el análisis costo/beneficio.

If Python is built in debug mode (if the **`Py_DEBUG`** macro is defined), the **`Py_ALWAYS_INLINE`** macro does nothing.

Debe ser especificado antes del tipo de retorno de la función. Uso:

```
static inline Py_ALWAYS_INLINE int random(void) { r
```

Added in version 3.11.

Py_NO_INLINE

Deshabilita el uso de inline en una función. Por ejemplo, reduce el consumo de la pila C: útil en compilaciones LTO +PGO que usan mucho inline (ver bpo-33720 [https://bugs.python.org/issue?@action=redirect&bpo=33720]).

Corresponds to the `noinline` attribute/specification on GCC and MSVC.

Uso:

```
Py_NO_INLINE static int random(void) { return 4; }
```

Added in version 3.11.

Py_DEPRECATED(version)

Use this to declare APIs that were deprecated in a specific CPython version. The macro must be placed before the symbol name.

Ejemplo:

```
Py_DEPRECATED(3.8) PyAPI_FUNC(int) Py_OldFunction(v
```

Distinto en la versión 3.8: Soporte para MSVC fue agregado.

Py_LOCAL(type)

Declare a function returning the specified *type* using a fast-calling qualifier for functions that are local to the current file. Semantically, this is equivalent to `static type`.

Py_LOCAL_INLINE(type)

Equivalent to **`Py_LOCAL`** but additionally requests the function be inlined.

Py_LOCAL_SYMBOL

Macro used to declare a symbol as local to the shared library (hidden). On supported platforms, it ensures the symbol is not exported.

On compatible versions of GCC/Clang, it expands to `__attribute__((visibility("hidden")))`.

Py_EXPORTED_SYMBOL

Macro used to declare a symbol (function or data) as exported. On Windows, this expands to `__declspec(dllexport)`. On compatible versions of GCC/ Clang, it expands to `__attribute__((visibility("default")))`. This macro is for defining the C API itself; extension modules should not use it.

Py_IMPORTED_SYMBOL

Macro used to declare a symbol as imported. On Windows, this expands to `__declspec(dllimport)`. This macro is for defining the C API itself; extension modules should not use it.

PyAPI_FUNC(type)

Macro used by CPython to declare a function as part of the C API. Its expansion depends on the platform and build configuration. This macro is intended for defining CPython's C API itself; extension modules should not use it for their own symbols.

PyAPI_DATA(type)

Macro used by CPython to declare a public global variable as part of the C API. Its expansion depends on the platform and build configuration. This macro is intended for defining CPython's C API itself; extension modules should not use it for their own symbols.

Outdated macros

The following macros have been used to features that have been standardized in C11.

Py_ALIGNED(num)

Specify alignment to *num* bytes on compilers that support it.

Consider using the C11 standard `_Alignas` specifier over this macro.

Py_LL(number)
Py_ULL(number)

Use *number* as a `long long` or `unsigned long long` integer literal, respectively.

Expands to *number* followed by `LL` or `LLU`, respectively, but will expand to some compiler-specific suffixes on some older compilers.

Consider using the C99 standard suffixes `LL` and `LLU` directly.

Py_MEMCPY(dest, src, n)

This is a soft deprecated alias to **`memcpy()`**. Use **`memcpy()`** directly instead.

Obsoleto desde la versión 3.14: The macro is soft deprecated.

Py_VA_COPY

This is a soft deprecated alias to the C99-standard `va_copy` function.

Historically, this would use a compiler-specific method to copy a `va_list`.

Distinto en la versión 3.6: This is now an alias to `va_copy`.

Objetos, tipos y conteos de referencias

Most Python/C API functions have one or more arguments as well as a return value of type PyObject*. This type is a pointer to an opaque data type representing an arbitrary Python object. Since all Python object types are treated the same way by the Python language in most situations (e.g., assignments, scope rules, and

argument passing), it is only fitting that they should be represented by a single C type. Almost all Python objects live on the heap: you never declare an automatic or static variable of type **`PyObject`**, only pointer variables of type PyObject* can be declared. The sole exception are the type objects; since these must never be deallocated, they are typically static **`PyTypeObject`** objects.

Todos los objetos de Python (incluso los enteros de Python) tienen un tipo (*type*) y un conteo de referencia (*reference count*). El tipo de un objeto determina qué tipo de objeto es (por ejemplo, un número entero, una lista o una función definida por el usuario; hay muchos más como se explica en Jerarquía de tipos estándar). Para cada uno de los tipos conocidos hay un macro para verificar si un objeto es de ese tipo; por ejemplo, `PyList_Check(a)` es verdadero si (y solo si) el objeto al que apunta *a* es una lista de Python.

Conteo de Referencias

The reference count is important because today's computers have a finite (and often severely limited) memory size; it counts how many different places there are that have a strong reference to an object. Such a place could be another object, or a global (or static) C variable, or a local variable in some C function. When the last strong reference to an object is released (i.e. its reference count becomes zero), the object is deallocated. If it contains references to other objects, those references are released. Those other objects may be deallocated in turn, if there are no more references to them, and so on. (There's an obvious problem with objects that reference each other here; for now, the solution is «don't do that.»)

Reference counts are always manipulated explicitly. The normal way is to use the macro **`Py_INCREF()`** to take a new reference to an object (i.e. increment its reference count by one), and **`Py_DECREF()`** to release that reference (i.e. decrement the reference count by one). The **`Py_DECREF()`** macro is considerably more complex than the incref one, since it must check whether the reference count becomes zero and then cause the object's deallocator to be called. The deallocator is a function pointer contained in the object's type structure. The type-specific deallocator takes care of releasing references for other objects

contained in the object if this is a compound object type, such as a list, as well as performing any additional finalization that's needed. There's no chance that the reference count can overflow; at least as many bits are used to hold the reference count as there are distinct memory locations in virtual memory (assuming `sizeof(Py_ssize_t) >= sizeof(void*)`). Thus, the reference count increment is a simple operation.

It is not necessary to hold a strong reference (i.e. increment the reference count) for every local variable that contains a pointer to an object. In theory, the object's reference count goes up by one when the variable is made to point to it and it goes down by one when the variable goes out of scope. However, these two cancel each other out, so at the end the reference count hasn't changed. The only real reason to use the reference count is to prevent the object from being deallocated as long as our variable is pointing to it. If we know that there is at least one other reference to the object that lives at least as long as our variable, there is no need to take a new strong reference (i.e. increment the reference count) temporarily. An important situation where this arises is in objects that are passed as arguments to C functions in an extension module that are called from Python; the call mechanism guarantees to hold a reference to every argument for the duration of the call.

However, a common pitfall is to extract an object from a list and hold on to it for a while without taking a new reference. Some other operation might conceivably remove the object from the list, releasing that reference, and possibly deallocating it. The real danger is that innocent-looking operations may invoke arbitrary Python code which could do this; there is a code path which allows control to flow back to the user from a **`Py_DECREF()`**, so almost any operation is potentially dangerous.

A safe approach is to always use the generic operations (functions whose name begins with `PyObject_`, `PyNumber_`, `PySequence_` or `PyMapping_`). These operations always create a new strong reference (i.e. increment the reference count) of the object they return. This leaves the caller with the responsibility to call **`Py_DECREF()`** when they are done with the result; this soon becomes second nature.

Detalles del conteo de referencia

The reference count behavior of functions in the Python/C API is best explained in terms of *ownership of references*. Ownership pertains to references, never to objects (objects are not owned: they are always shared). «Owning a reference» means being responsible for calling Py_DECREF on it when the reference is no longer needed. Ownership can also be transferred, meaning that the code that receives ownership of the reference then becomes responsible for eventually releasing it by calling **`Py_DECREF()`** or **`Py_XDECREF()`** when it's no longer needed—or passing on this responsibility (usually to its caller). When a function passes ownership of a reference on to its caller, the caller is said to receive a *new* reference. When no ownership is transferred, the caller is said to *borrow* the reference. Nothing needs to be done for a borrowed reference.

Por el contrario, cuando una función de llamada pasa una referencia a un objeto, hay dos posibilidades: la función *roba* una referencia al objeto, o no lo hace. *Robar una referencia* significa que cuando pasa una referencia a una función, esa función asume que ahora posee esa referencia, y usted ya no es responsable de ella.

Pocas funciones roban referencias; las dos excepciones notables son **`PyList_SetItem()`** y **`PyTuple_SetItem()`**, que roban una referencia al elemento (¡pero no a la tupla o lista en la que se coloca el elemento!). Estas funciones fueron diseñadas para robar una referencia debido a un idioma común para poblar una tupla o lista con objetos recién creados; por ejemplo, el código para crear la tupla `(1, 2, "tres")` podría verse así (olvidando el manejo de errores por el momento; una mejor manera de codificar esto se muestra a continuación):

```
PyObject *t;

t = PyTuple_New(3);
PyTuple_SetItem(t, 0, PyLong_FromLong(1L));
PyTuple_SetItem(t, 1, PyLong_FromLong(2L));
PyTuple_SetItem(t, 2, PyUnicode_FromString("three"));
```

Aquí `PyLong_FromLong()` retorna una nueva referencia que es inmediatamente robada por `PyTuple_SetItem()`. Cuando quiera seguir usando un objeto aunque se le robe la referencia, use `Py_INCREF()` para tomar otra referencia antes de llamar a la función de robo de referencias.

Por cierto, `PyTuple_SetItem()` es la *única* forma de establecer elementos de tupla; `PySequence_SetItem()` y `PyObject_SetItem()` se niegan a hacer esto ya que las tuplas son un tipo de datos inmutable. Solo debe usar **`PyTuple_SetItem()`** para las tuplas que está creando usted mismo.

El código equivalente para llenar una lista se puede escribir usando `PyList_New()` y `PyList_SetItem()`.

Sin embargo, en la práctica, rara vez utilizará estas formas de crear y completar una tupla o lista. Hay una función genérica, `Py_BuildValue()`, que puede crear los objetos más comunes a partir de valores C, dirigidos por un una cadena de caracteres de formato (*format string*). Por ejemplo, los dos bloques de código anteriores podrían reemplazarse por lo siguiente (que también se ocupa de la comprobación de errores):

```
PyObject *tuple, *list;

tuple = Py_BuildValue("(iis)", 1, 2, "three");
list = Py_BuildValue("[iis]", 1, 2, "three");
```

It is much more common to use `PyObject_SetItem()` and friends with items whose references you are only borrowing, like arguments that were passed in to the function you are writing. In that case, their behaviour regarding references is much saner, since you don’t have to take a new reference just so you can give that reference away («have it be stolen»). For example, this function sets all items of a list (actually, any mutable sequence) to a given item:

```
int
set_all(PyObject *target, PyObject *item)
{
    Py_ssize_t i, n;
    n = PyObject_Length(target);
    if (n < 0)
        return -1;
    for (i = 0; i < n; i++) {
        PyObject *index = PyLong_FromSsize_t(i);
        if (!index)
            return -1;
        if (PyObject_SetItem(target, index, item) < 0) {
            Py_DECREF(index);
            return -1;
        }
        Py_DECREF(index);
    }
    return 0;
}
```

La situación es ligeramente diferente para los valores de retorno de la función. Si bien pasar una referencia a la mayoría de las funciones no cambia sus responsabilidades de propiedad para esa referencia, muchas funciones que retornan una referencia a un objeto le otorgan la propiedad de la referencia. La razón es simple: en muchos casos, el objeto retornado se crea sobre la marcha, y la referencia que obtiene es la única referencia al objeto. Por lo tanto, las funciones genéricas que retornan referencias de objeto, como `PyObject_GetItem()` y `PySequence_GetItem()`, siempre retornan una nueva referencia (la entidad que llama se convierte en el propietario de la referencia).

Es importante darse cuenta de que si posee una referencia retornada por una función depende de a qué función llame únicamente — *el plumaje* (el tipo del objeto pasado como argumento a la función) *no entra en él!* Por lo tanto, si extrae un elemento de una lista usando `PyList_GetItem()`, no posee la referencia — pero si obtiene el mismo elemento de la misma lista usando `PySequence_GetItem()` (que toma exactamente los mismos argumentos), usted posee una referencia al objeto retornado.

Aquí hay un ejemplo de cómo podría escribir una función que calcule la suma de los elementos en una lista de enteros; una vez usando `PyList_GetItem()`, y una vez usando

`PySequence_GetItem()`.

```
long
sum_list(PyObject *list)
{
    Py_ssize_t i, n;
    long total = 0, value;
    PyObject *item;

    n = PyList_Size(list);
    if (n < 0)
        return -1; /* Not a list */
    for (i = 0; i < n; i++) {
        item = PyList_GetItem(list, i); /* Can't fail */
        if (!PyLong_Check(item)) continue; /* Skip non-i
        value = PyLong_AsLong(item);
        if (value == -1 && PyErr_Occurred())
            /* Integer too big to fit in a C long, bail
            return -1;
        total += value;
    }
    return total;
}

long
sum_sequence(PyObject *sequence)
{
    Py_ssize_t i, n;
    long total = 0, value;
    PyObject *item;
    n = PySequence_Length(sequence);
    if (n < 0)
        return -1; /* Has no length */
    for (i = 0; i < n; i++) {
        item = PySequence_GetItem(sequence, i);
        if (item == NULL)
            return -1; /* Not a sequence, or other failu
        if (PyLong_Check(item)) {
            value = PyLong_AsLong(item);
            Py_DECREF(item);
            if (value == -1 && PyErr_Occurred())
                /* Integer too big to fit in a C long, b
                return -1;
            total += value;
        }
        else {
            Py_DECREF(item); /* Discard reference owners
        }
    }
    return total;
}
```

Tipos

There are few other data types that play a significant role in the Python/C API; most are simple C types such as int, long, double and char*. A few structure types are used to describe static tables used to list the functions exported by a module or the data attributes of a new object type, and another is used to describe the value of a complex number. These will be discussed together with the functions that use them.

type Py_ssize_t

Part of the Stable ABI.
Un tipo integral con signo tal que `sizeof(Py_ssize_t) == sizeof(size_t)`. C99 no define directamente tal cosa (size_t es un tipo integral sin signo). Vea **PEP 353** [https://peps.python.org/pep-0353/] para más detalles. `PY_SSIZE_T_MAX` es el valor positivo más grande del tipo **`Py_ssize_t`**.

Excepciones

El programador de Python solo necesita lidiar con excepciones si se requiere un manejo específico de errores; las excepciones no manejadas se propagan automáticamente a la persona que llama, luego a la persona que llama, y así sucesivamente, hasta que llegan al intérprete de nivel superior, donde se informan al usuario acompañado de un seguimiento de pila (*stack traceback*).

Para los programadores de C, sin embargo, la comprobación de errores siempre tiene que ser explícita. Todas las funciones en la API Python/C pueden generar excepciones, a menos que se señale explícitamente en la documentación de una función. En general, cuando una función encuentra un error, establece una excepción, descarta cualquier referencia de objeto que posea y retorna un indicador de error. Si no se documenta lo contrario, este indicador es `NULL` o `-1`, dependiendo del tipo de retorno de la función. Algunas funciones retornan un resultado booleano verdadero/falso, con falso que indica un error. Muy pocas funciones no retornan ningún indicador de error explícito o tienen un valor de retorno ambiguo, y requieren pruebas explícitas de errores con **`PyErr_Occurred()`**. Estas excepciones siempre se documentan explícitamente.

El estado de excepción se mantiene en el almacenamiento por subproceso (esto es equivalente a usar el almacenamiento global en una aplicación sin subprocesos). Un subproceso puede estar en uno de dos estados: se ha producido una excepción o no. La función **`PyErr_Occurred()`** puede usarse para verificar esto: retorna una referencia prestada al objeto de tipo de excepción cuando se produce una excepción, y `NULL` de lo contrario. Hay una serie de funciones para establecer el estado de excepción: **`PyErr_SetString()`** es la función más común (aunque no la más general) para establecer el estado de excepción, y **`PyErr_Clear()`** borra la excepción estado.

El estado de excepción completo consta de tres objetos (todos los cuales pueden ser `NULL`): el tipo de excepción, el valor de excepción correspondiente y el rastreo. Estos tienen los mismos significados que el resultado de Python de `sys.exc_info()`; sin embargo, no son lo mismo: los objetos Python representan la última excepción manejada por una declaración de Python **`try`** … **`except`**, mientras que el estado de excepción de nivel C solo existe mientras se está pasando una excepción entre las funciones de C hasta que llega al bucle principal del intérprete de código de bytes (*bytecode*) de Python, que se encarga de transferirlo a `sys.exc_info()` y amigos.

Tenga en cuenta que a partir de Python 1.5, la forma preferida y segura de subprocesos para acceder al estado de excepción desde el código de Python es llamar a la función `sys.exc_info()`, que retorna el estado de excepción por subproceso para el código de Python. Además, la semántica de ambas formas de acceder al estado de excepción ha cambiado de modo que una función que detecta una excepción guardará y restaurará el estado de excepción de su hilo para preservar el estado de excepción de su llamador. Esto evita errores comunes en el código de manejo de excepciones causado por una función de aspecto inocente que sobrescribe la excepción que se maneja; También reduce la extensión de vida útil a menudo no deseada para los objetos a los que hacen referencia los marcos de pila en el rastreo.

Como principio general, una función que llama a otra función para realizar alguna tarea debe verificar si la función llamada generó una excepción y, de ser así, pasar el estado de excepción a quien la llama (*caller*). Debe descartar cualquier referencia de objeto que posea y retornar un indicador de error, pero *no* debe establecer otra excepción — que sobrescribirá la excepción que se acaba de generar y perderá información importante sobre la causa exacta del error.

A simple example of detecting exceptions and passing them on is shown in the **`sum_sequence()`** example above. It so happens that this example doesn't need to clean up any owned references when it detects an error. The following example function shows some error cleanup. First, to remind you why you like Python, we show the equivalent Python code:

```
def incr_item(dict, key):
    try:
        item = dict[key]
    except KeyError:
        item = 0
    dict[key] = item + 1
```

Aquí está el código C correspondiente, en todo su esplendor:

```
int
incr_item(PyObject *dict, PyObject *key)
{
    /* Objects all initialized to NULL for Py_XDECREF */
    PyObject *item = NULL, *const_one = NULL, *increment
    int rv = -1; /* Return value initialized to -1 (fail

    item = PyObject_GetItem(dict, key);
    if (item == NULL) {
        /* Handle KeyError only: */
        if (!PyErr_ExceptionMatches(PyExc_KeyError))
            goto error;

        /* Clear the error and use zero: */
        PyErr_Clear();
        item = PyLong_FromLong(0L);
        if (item == NULL)
            goto error;
    }
    const_one = PyLong_FromLong(1L);
    if (const_one == NULL)
        goto error;

    incremented_item = PyNumber_Add(item, const_one);
    if (incremented_item == NULL)
        goto error;

    if (PyObject_SetItem(dict, key, incremented_item) <
        goto error;
    rv = 0; /* Success */
    /* Continue with cleanup code */

 error:
    /* Cleanup code, shared by success and failure path

    /* Use Py_XDECREF() to ignore NULL references */
    Py_XDECREF(item);
    Py_XDECREF(const_one);
    Py_XDECREF(incremented_item);

    return rv; /* -1 for error, 0 for success */
```

```
}
```

Este ejemplo representa un uso aprobado de la declaración `goto` en C! Ilustra el uso de **`PyErr_ExceptionMatches()`** y **`PyErr_Clear()`** para manejar excepciones específicas, y el uso de **`Py_XDECREF()`** para eliminar referencias propias que pueden ser `NULL` (tenga en cuenta la `'X'`” en el nombre; **`Py_DECREF()`** se bloqueará cuando se enfrente con una referencia `NULL`). Es importante que las variables utilizadas para contener referencias propias se inicialicen en `NULL` para que esto funcione; Del mismo modo, el valor de retorno propuesto se inicializa a `-1` (falla) y solo se establece en éxito después de que la última llamada realizada sea exitosa.

Integración de Python

La única tarea importante de la que solo tienen que preocuparse los integradores (a diferencia de los escritores de extensión) del intérprete de Python es la inicialización, y posiblemente la finalización, del intérprete de Python. La mayor parte de la funcionalidad del intérprete solo se puede usar después de que el intérprete se haya inicializado.

La función básica de inicialización es **`Py_Initialize()`**. Esto inicializa la tabla de módulos cargados y crea los módulos fundamentales **`builtins`**, **`__main__`**, y **`sys`**. También inicializa la ruta de búsqueda del módulo (`sys.path`).

`Py_Initialize()` no establece la «lista de argumentos de script» (`sys.argv`). Si esta variable es necesaria por el código Python que se ejecutará más tarde, debe establecerse **`PyConfig.argv`** y **`PyConfig.parse_argv`**: consulte Python Initialization Configuration.

En la mayoría de los sistemas (en particular, en Unix y Windows, aunque los detalles son ligeramente diferentes), **`Py_Initialize()`** calcula la ruta de búsqueda del módulo basándose en su mejor estimación de la ubicación del ejecutable del intérprete de Python estándar, suponiendo que la biblioteca de Python se encuentra en una ubicación fija en relación con el ejecutable del intérprete de Python. En particular, busca un directorio llamado `lib/pythonX.Y` relativo al directorio padre donde se encuentra el ejecutable llamado `python` en la ruta de búsqueda del comando *shell* (la variable de entorno **PATH**).

Por ejemplo, si el ejecutable de Python se encuentra en `/usr/local/bin/python`, se supondrá que las bibliotecas están en `/usr/local/lib/pythonX.Y`. (De hecho, esta ruta particular también es la ubicación «alternativa», utilizada cuando no se encuentra un archivo ejecutable llamado `python` junto con **PATH**.) El usuario puede anular este comportamiento configurando la variable de entorno **PYTHONHOME**, o inserte directorios adicionales delante de la ruta estándar estableciendo **PYTHONPATH**.

The embedding application can steer the search by setting `PyConfig.program_name` *before* calling `Py_InitializeFromConfig()`. Note that **PYTHONHOME** still overrides this and **PYTHONPATH** is still inserted in front of the standard path. An application that requires total control has to provide its own implementation of `Py_GetPath()`, `Py_GetPrefix()`, `Py_GetExecPrefix()`, and `Py_GetProgramFullPath()` (all defined in `Modules/getpath.c`).

A veces, es deseable «no inicializar» Python. Por ejemplo, la aplicación puede querer comenzar de nuevo (hacer otra llamada a `Py_Initialize()`) o la aplicación simplemente se hace con el uso de Python y quiere liberar memoria asignada por Python. Esto se puede lograr llamando a `Py_FinalizeEx()`. La función `Py_IsInitialized()` retorna verdadero si Python se encuentra actualmente en el estado inicializado. Se proporciona más información sobre estas funciones en un capítulo posterior. Tenga en cuenta que `Py_FinalizeEx()` *no* libera toda la memoria asignada por el intérprete de Python, por ejemplo, la memoria asignada por los módulos de extensión actualmente no se puede liberar.

Depuración de compilaciones

Python se puede construir con varios macros para permitir verificaciones adicionales del intérprete y los módulos de extensión. Estas comprobaciones tienden a agregar una gran cantidad de sobrecarga al tiempo de ejecución, por lo que no están habilitadas de forma predeterminada.

A full list of the various types of debugging builds is in the file `Misc/SpecialBuilds.txt` in the Python source distribution. Builds are available that support tracing of reference counts, debugging the memory allocator, or low-level profiling of the main interpreter loop. Only the most frequently used builds will be described in the remainder of this section.

Py_DEBUG

Compiling the interpreter with the **`Py_DEBUG`** macro defined produces what is generally meant by a debug build of Python. **`Py_DEBUG`** is enabled in the Unix build by adding **`--with-pydebug`** to the `./configure` command. It is also implied by the presence of the not-Python-specific **`_DEBUG`** macro. When **`Py_DEBUG`** is enabled in the Unix build, compiler optimization is disabled.

Además de la depuración del recuento de referencia que se describe a continuación, se realizan verificaciones adicionales, véase compilaciones de depuración.

Definiendo **`Py_TRACE_REFS`** habilita el rastreo de referencias (véase la opción **`configure --with-trace-refs`**). Cuando se define, se mantiene una lista circular doblemente vinculada de objetos activos al agregar dos campos adicionales a cada **`PyObject`**. También se realiza un seguimiento de las asignaciones totales. Al salir, se imprimen todas las referencias existentes. (En modo interactivo, esto sucede después de cada declaración ejecutada por el intérprete).

Consulte `Misc/SpecialBuilds.txt` en la distribución fuente de Python para obtener información más detallada.

Recommended third party tools

The following third party tools offer both simpler and more sophisticated approaches to creating C, C++ and Rust extensions for Python:

- Cython [https://cython.org/]
- cffi [https://cffi.readthedocs.io]
- HPy [https://hpyproject.org/]
- nanobind [https://github.com/wjakob/nanobind] (C++)
- Numba [https://numba.pydata.org/]
- pybind11 [https://pybind11.readthedocs.io/] (C++)
- PyO3 [https://pyo3.rs/] (Rust)
- SWIG [https://www.swig.org]

Using tools such as these can help avoid writing code that is tightly bound to a particular version of CPython, avoid reference counting errors, and focus more on your own code than on using the CPython API. In general, new versions of Python can be supported by updating the tool, and your code will often use newer and more efficient APIs automatically. Some tools also support compiling for other implementations of Python from a single set of sources.

These projects are not supported by the same people who maintain Python, and issues need to be raised with the projects directly. Remember to check that the project is still maintained and supported, as the list above may become outdated.

Ver también

Python Packaging User Guide: Binary Extensions [https://packaging.python.org/guides/packaging-binary-extensions/]

The Python Packaging User Guide not only covers several available tools that simplify the creation of binary extensions, but also discusses the various reasons why creating an extension module may be desirable in the first place.

Estabilidad de la API en C

Unless documented otherwise, Python's C API is covered by the Backwards Compatibility Policy, **PEP 387** [https://peps.python.org/pep-0387/]. Most changes to it are source-compatible (typically by only adding new API). Changing existing API or removing API is only done after a deprecation period or to fix serious issues.

CPython's Application Binary Interface (ABI) is forward- and backwards-compatible across a minor release (if these are compiled the same way; see Consideraciones de la plataforma below). So, code compiled for Python 3.10.0 will work on 3.10.8 and vice versa, but will need to be compiled separately for 3.9.x and 3.11.x.

There are two tiers of C API with different stability expectations:

- Unstable API, may change in minor versions without a deprecation period. It is marked by the `PyUnstable` prefix in names.
- Limited API, is compatible across several minor releases. When **`Py_LIMITED_API`** is defined, only this subset is exposed from `Python.h`.

These are discussed in more detail below.

Names prefixed by an underscore, such as `_Py_InternalState`, are private API that can change without notice even in patch releases. If you need to use this API, consider reaching out to CPython developers [https://discuss.python.org/c/core-dev/c-api/30] to discuss adding public API for your use case.

Unstable C API

Any API named with the `PyUnstable` prefix exposes CPython implementation details, and may change in every minor release (e.g. from 3.9 to 3.10) without any deprecation warnings. However,

it will not change in a bugfix release (e.g. from 3.10.0 to 3.10.1).

It is generally intended for specialized, low-level tools like debuggers.

Projects that use this API are expected to follow CPython development and spend extra effort adjusting to changes.

Interfaz binaria de aplicación estable

For simplicity, this document talks about *extensions*, but the Limited API and Stable ABI work the same way for all uses of the API – for example, embedding Python.

Limited C API

Python 3.2 introduced the *Limited API*, a subset of Python's C API. Extensions that only use the Limited API can be compiled once and be loaded on multiple versions of Python. Contents of the Limited API are listed below.

Py_LIMITED_API

> Se define esta macro antes de incluir `Python.h` para optar por usar sólo la API limitada y para seleccionar la versión de la API limitada.
>
> Define `Py_LIMITED_API` to the value of **`PY_VERSION_HEX`** corresponding to the lowest Python version your extension supports. The extension will be ABI-compatible with all Python 3 releases from the specified one onward, and can use Limited API introduced up to that version.
>
> En lugar de utilizar directamente la macro `PY_VERSION_HEX`, se codifica una versión menor mínima (por ejemplo, `0x030A0000` para Python 3.10) para tener estabilidad cuando se compila con futuras versiones de Python.
>
> También se puede definir `Py_LIMITED_API` con `3`. Esto funciona igual que `0x03020000` (Python 3.2, la función que

introdujo la API limitada).

Stable ABI

To enable this, Python provides a *Stable ABI*: a set of symbols that will remain ABI-compatible across Python 3.x versions.

Nota

The Stable ABI prevents ABI issues, like linker errors due to missing symbols or data corruption due to changes in structure layouts or function signatures. However, other changes in Python can change the *behavior* of extensions. See Python's Backwards Compatibility Policy (**PEP 387** [https://peps.python.org/pep-0387/]) for details.

The Stable ABI contains symbols exposed in the Limited API, but also other ones – for example, functions necessary to support older versions of the Limited API.

En Windows, las extensiones que usan la ABI estable deben estar vinculadas con `python3.dll` en lugar de una biblioteca específica de la versión como `python39.dll`.

En algunas plataformas, Python buscará y cargará archivos de bibliotecas compartidas con el nombre de la etiqueta `abi3` (por ejemplo, `mymodule.abi3.so`). No comprueba si tales extensiones se ajustan a una ABI estable. El usuario (o sus herramientas de empaquetado) necesitan asegurarse que, por ejemplo, las extensiones que se crean con la API limitada 3.10+ no estén instaladas para versiones inferiores de Python.

Todas las funciones de la ABI estable se presentan como funciones en la biblioteca compartida de Python, no sólo como macros. Esto las hace utilizables desde lenguajes que no usan el preprocesador de C.

Alcance y rendimiento de la API limitada

El objetivo de la API limitada es permitir todo lo que es posible con la API completa en C, pero posiblemente con una penalización de rendimiento.

Por ejemplo, mientras **`PyList_GetItem()`** está disponible, su variante macro “insegura” **`PyList_GET_ITEM()`** no lo está. La macro puede ser más rápida porque puede confiar en los detalles de implementación específicos de la versión del objeto de lista.

Sin definirse `Py_LIMITED_API`, algunas funciones de la API en C están integradas o reemplazadas por macros. Definir `Py_LIMITED_API` desactiva esta integración, permitiendo estabilidad mientras que se mejoren las estructuras de datos de Python, pero posiblemente reduzca el rendimiento.

Al dejar fuera la definición de `Py_LIMITED_API`, es posible compilar una extensión de la API limitada con una ABI específica de la versión. Esto puede mejorar el rendimiento para esa versión de Python, pero limitará la compatibilidad. Compilar con `Py_LIMITED_API` producirá una extensión que se puede distribuir donde una versión específica no esté disponible - por ejemplo, para los prelanzamientos de una versión próxima de Python.

Advertencias de la API limitada

Note that compiling with `Py_LIMITED_API` is *not* a complete guarantee that code conforms to the Limited API or the Stable ABI. `Py_LIMITED_API` only covers definitions, but an API also includes other issues, such as expected semantics.

Un problema contra el que `Py_LIMITED_API` no protege es llamar una función con argumentos que son inválidos en una versión inferior de Python. Por ejemplo, se considera una función que empieza a aceptar `NULL` como un argumento. Ahora en Python 3.9, `NULL` selecciona un comportamiento predeterminado, pero en Python 3.8, el argumento se usará directamente, causando una desreferencia `NULL` y se detendrá. Un argumento similar funciona para campos de estructuras.

Otro problema es que algunos campos de estructura no se ocultan actualmente cuando se define `Py_LIMITED_API`, aunque son parte de la API limitada.

Por estas razones, recomendamos probar una extensión con *todas* las versiones menores de Python que soporte, y preferiblemente compilar con la versión *más baja*.

También recomendamos revisar la documentación de todas las API usadas para verificar si es parte explícitamente de la API limitada. Aunque se defina `Py_LIMITED_API`, algunas declaraciones privadas se exponen por razones técnicas (o incluso involuntariamente, como errores).

También tome en cuenta que la API limitada no necesariamente es estable: compilar con `Py_LIMITED_API` con Python 3.8 significa que la extensión se ejecutará con Python 3.12, pero no necesariamente *compilará* con Python 3.12. En particular, las partes de la API limitada se pueden quedar obsoletas y eliminarse, siempre que la ABI estable permanezca estable.

Consideraciones de la plataforma

ABI stability depends not only on Python, but also on the compiler used, lower-level libraries and compiler options. For the purposes of the Stable ABI, these details define a “platform”. They usually depend on the OS type and processor architecture

Es la responsabilidad de cada distribuidor particular de Python de asegurarse de que todas las versiones de Python en una plataforma particular se compilen de una forma que no rompa la ABI estable. Este es el caso de las versiones de Windows y macOS de `python.org` y muchos distribuidores de terceros.

Contenido de la API limitada

Currently, the Limited API includes the following items:

- **`METH_CLASS`**
- **`METH_COEXIST`**
- **`METH_FASTCALL`**
- **`METH_METHOD`**

- **METH_NOARGS**
- **METH_O**
- **METH_STATIC**
- **METH_VARARGS**
- **PY_VECTORCALL_ARGUMENTS_OFFSET**
- **PyAIter_Check()**
- **PyArg_Parse()**
- **PyArg_ParseTuple()**
- **PyArg_ParseTupleAndKeywords()**
- **PyArg_UnpackTuple()**
- **PyArg_VaParse()**
- **PyArg_VaParseTupleAndKeywords()**
- **PyArg_ValidateKeywordArguments()**
- **PyBUF_ANY_CONTIGUOUS**
- **PyBUF_CONTIG**
- **PyBUF_CONTIG_RO**
- **PyBUF_C_CONTIGUOUS**
- **PyBUF_FORMAT**
- **PyBUF_FULL**
- **PyBUF_FULL_RO**
- **PyBUF_F_CONTIGUOUS**
- **PyBUF_INDIRECT**
- **PyBUF_MAX_NDIM**
- **PyBUF_ND**
- **PyBUF_READ**
- **PyBUF_RECORDS**
- **PyBUF_RECORDS_RO**
- **PyBUF_SIMPLE**
- **PyBUF_STRIDED**
- **PyBUF_STRIDED_RO**
- **PyBUF_STRIDES**
- **PyBUF_WRITABLE**
- **PyBUF_WRITE**
- **PyBaseObject_Type**
- **PyBool_FromLong()**
- **PyBool_Type**
- **PyBuffer_FillContiguousStrides()**
- **PyBuffer_FillInfo()**
- **PyBuffer_FromContiguous()**
- **PyBuffer_GetPointer()**

- `PyBuffer_IsContiguous()`
- `PyBuffer_Release()`
- `PyBuffer_SizeFromFormat()`
- `PyBuffer_ToContiguous()`
- `PyByteArrayIter_Type`
- `PyByteArray_AsString()`
- `PyByteArray_Concat()`
- `PyByteArray_FromObject()`
- `PyByteArray_FromStringAndSize()`
- `PyByteArray_Resize()`
- `PyByteArray_Size()`
- `PyByteArray_Type`
- `PyBytesIter_Type`
- `PyBytes_AsString()`
- `PyBytes_AsStringAndSize()`
- `PyBytes_Concat()`
- `PyBytes_ConcatAndDel()`
- `PyBytes_DecodeEscape()`
- `PyBytes_FromFormat()`
- `PyBytes_FromFormatV()`
- `PyBytes_FromObject()`
- `PyBytes_FromString()`
- `PyBytes_FromStringAndSize()`
- `PyBytes_Repr()`
- `PyBytes_Size()`
- `PyBytes_Type`
- `PyCFunction`
- `PyCFunctionFast`
- `PyCFunctionFastWithKeywords`
- `PyCFunctionWithKeywords`
- `PyCFunction_GetFlags()`
- `PyCFunction_GetFunction()`
- `PyCFunction_GetSelf()`
- `PyCFunction_New()`
- `PyCFunction_NewEx()`
- `PyCFunction_Type`
- `PyCMethod_New()`
- `PyCallIter_New()`
- `PyCallIter_Type`
- `PyCallable_Check()`

- `PyCapsule_Destructor`
- `PyCapsule_GetContext()`
- `PyCapsule_GetDestructor()`
- `PyCapsule_GetName()`
- `PyCapsule_GetPointer()`
- `PyCapsule_Import()`
- `PyCapsule_IsValid()`
- `PyCapsule_New()`
- `PyCapsule_SetContext()`
- `PyCapsule_SetDestructor()`
- `PyCapsule_SetName()`
- `PyCapsule_SetPointer()`
- `PyCapsule_Type`
- `PyClassMethodDescr_Type`
- `PyCodec_BackslashReplaceErrors()`
- `PyCodec_Decode()`
- `PyCodec_Decoder()`
- `PyCodec_Encode()`
- `PyCodec_Encoder()`
- `PyCodec_IgnoreErrors()`
- `PyCodec_IncrementalDecoder()`
- `PyCodec_IncrementalEncoder()`
- `PyCodec_KnownEncoding()`
- `PyCodec_LookupError()`
- `PyCodec_NameReplaceErrors()`
- `PyCodec_Register()`
- `PyCodec_RegisterError()`
- `PyCodec_ReplaceErrors()`
- `PyCodec_StreamReader()`
- `PyCodec_StreamWriter()`
- `PyCodec_StrictErrors()`
- `PyCodec_Unregister()`
- `PyCodec_XMLCharRefReplaceErrors()`
- `PyComplex_FromDoubles()`
- `PyComplex_ImagAsDouble()`
- `PyComplex_RealAsDouble()`
- `PyComplex_Type`
- `PyDescr_NewClassMethod()`
- `PyDescr_NewGetSet()`
- `PyDescr_NewMember()`

- `PyDescr_NewMethod()`
- `PyDictItems_Type`
- `PyDictIterItem_Type`
- `PyDictIterKey_Type`
- `PyDictIterValue_Type`
- `PyDictKeys_Type`
- `PyDictProxy_New()`
- `PyDictProxy_Type`
- `PyDictRevIterItem_Type`
- `PyDictRevIterKey_Type`
- `PyDictRevIterValue_Type`
- `PyDictValues_Type`
- `PyDict_Clear()`
- `PyDict_Contains()`
- `PyDict_Copy()`
- `PyDict_DelItem()`
- `PyDict_DelItemString()`
- `PyDict_GetItem()`
- `PyDict_GetItemRef()`
- `PyDict_GetItemString()`
- `PyDict_GetItemStringRef()`
- `PyDict_GetItemWithError()`
- `PyDict_Items()`
- `PyDict_Keys()`
- `PyDict_Merge()`
- `PyDict_MergeFromSeq2()`
- `PyDict_New()`
- `PyDict_Next()`
- `PyDict_SetItem()`
- `PyDict_SetItemString()`
- `PyDict_Size()`
- `PyDict_Type`
- `PyDict_Update()`
- `PyDict_Values()`
- `PyEllipsis_Type`
- `PyEnum_Type`
- `PyErr_BadArgument()`
- `PyErr_BadInternalCall()`
- `PyErr_CheckSignals()`
- `PyErr_Clear()`

- `PyErr_Display()`
- `PyErr_DisplayException()`
- `PyErr_ExceptionMatches()`
- `PyErr_Fetch()`
- `PyErr_Format()`
- `PyErr_FormatV()`
- `PyErr_GetExcInfo()`
- `PyErr_GetHandledException()`
- `PyErr_GetRaisedException()`
- `PyErr_GivenExceptionMatches()`
- `PyErr_NewException()`
- `PyErr_NewExceptionWithDoc()`
- `PyErr_NoMemory()`
- `PyErr_NormalizeException()`
- `PyErr_Occurred()`
- `PyErr_Print()`
- `PyErr_PrintEx()`
- `PyErr_ProgramText()`
- `PyErr_ResourceWarning()`
- `PyErr_Restore()`
- `PyErr_SetExcFromWindowsErr()`
- `PyErr_SetExcFromWindowsErrWithFilename()`
- `PyErr_SetExcFromWindowsErrWithFilenameObject()`
- `PyErr_SetExcFromWindowsErrWithFilenameObjects()`
- `PyErr_SetExcInfo()`
- `PyErr_SetFromErrno()`
- `PyErr_SetFromErrnoWithFilename()`
- `PyErr_SetFromErrnoWithFilenameObject()`
- `PyErr_SetFromErrnoWithFilenameObjects()`
- `PyErr_SetFromWindowsErr()`
- `PyErr_SetFromWindowsErrWithFilename()`
- `PyErr_SetHandledException()`
- `PyErr_SetImportError()`
- `PyErr_SetImportErrorSubclass()`
- `PyErr_SetInterrupt()`
- `PyErr_SetInterruptEx()`
- `PyErr_SetNone()`
- `PyErr_SetObject()`
- `PyErr_SetRaisedException()`
- `PyErr_SetString()`

- `PyErr_SyntaxLocation()`
- `PyErr_SyntaxLocationEx()`
- `PyErr_WarnEx()`
- `PyErr_WarnExplicit()`
- `PyErr_WarnFormat()`
- `PyErr_WriteUnraisable()`
- `PyEval_AcquireThread()`
- `PyEval_EvalCode()`
- `PyEval_EvalCodeEx()`
- `PyEval_EvalFrame()`
- `PyEval_EvalFrameEx()`
- `PyEval_GetBuiltins()`
- `PyEval_GetFrame()`
- `PyEval_GetFrameBuiltins()`
- `PyEval_GetFrameGlobals()`
- `PyEval_GetFrameLocals()`
- `PyEval_GetFuncDesc()`
- `PyEval_GetFuncName()`
- `PyEval_GetGlobals()`
- `PyEval_GetLocals()`
- `PyEval_InitThreads()`
- `PyEval_ReleaseThread()`
- `PyEval_RestoreThread()`
- `PyEval_SaveThread()`
- `PyExc_ArithmeticError`
- `PyExc_AssertionError`
- `PyExc_AttributeError`
- `PyExc_BaseException`
- `PyExc_BaseExceptionGroup`
- `PyExc_BlockingIOError`
- `PyExc_BrokenPipeError`
- `PyExc_BufferError`
- `PyExc_BytesWarning`
- `PyExc_ChildProcessError`
- `PyExc_ConnectionAbortedError`
- `PyExc_ConnectionError`
- `PyExc_ConnectionRefusedError`
- `PyExc_ConnectionResetError`
- `PyExc_DeprecationWarning`
- `PyExc_EOFError`

- `PyExc_EncodingWarning`
- `PyExc_EnvironmentError`
- `PyExc_Exception`
- `PyExc_FileExistsError`
- `PyExc_FileNotFoundError`
- `PyExc_FloatingPointError`
- `PyExc_FutureWarning`
- `PyExc_GeneratorExit`
- `PyExc_IOError`
- `PyExc_ImportError`
- `PyExc_ImportWarning`
- `PyExc_IndentationError`
- `PyExc_IndexError`
- `PyExc_InterruptedError`
- `PyExc_IsADirectoryError`
- `PyExc_KeyError`
- `PyExc_KeyboardInterrupt`
- `PyExc_LookupError`
- `PyExc_MemoryError`
- `PyExc_ModuleNotFoundError`
- `PyExc_NameError`
- `PyExc_NotADirectoryError`
- `PyExc_NotImplementedError`
- `PyExc_OSError`
- `PyExc_OverflowError`
- `PyExc_PendingDeprecationWarning`
- `PyExc_PermissionError`
- `PyExc_ProcessLookupError`
- `PyExc_RecursionError`
- `PyExc_ReferenceError`
- `PyExc_ResourceWarning`
- `PyExc_RuntimeError`
- `PyExc_RuntimeWarning`
- `PyExc_StopAsyncIteration`
- `PyExc_StopIteration`
- `PyExc_SyntaxError`
- `PyExc_SyntaxWarning`
- `PyExc_SystemError`
- `PyExc_SystemExit`
- `PyExc_TabError`

- `PyExc_TimeoutError`
- `PyExc_TypeError`
- `PyExc_UnboundLocalError`
- `PyExc_UnicodeDecodeError`
- `PyExc_UnicodeEncodeError`
- `PyExc_UnicodeError`
- `PyExc_UnicodeTranslateError`
- `PyExc_UnicodeWarning`
- `PyExc_UserWarning`
- `PyExc_ValueError`
- `PyExc_Warning`
- `PyExc_WindowsError`
- `PyExc_ZeroDivisionError`
- `PyExceptionClass_Name()`
- `PyException_GetArgs()`
- `PyException_GetCause()`
- `PyException_GetContext()`
- `PyException_GetTraceback()`
- `PyException_SetArgs()`
- `PyException_SetCause()`
- `PyException_SetContext()`
- `PyException_SetTraceback()`
- `PyFile_FromFd()`
- `PyFile_GetLine()`
- `PyFile_WriteObject()`
- `PyFile_WriteString()`
- `PyFilter_Type`
- `PyFloat_AsDouble()`
- `PyFloat_FromDouble()`
- `PyFloat_FromString()`
- `PyFloat_GetInfo()`
- `PyFloat_GetMax()`
- `PyFloat_GetMin()`
- `PyFloat_Type`
- `PyFrameObject`
- `PyFrame_GetCode()`
- `PyFrame_GetLineNumber()`
- `PyFrozenSet_New()`
- `PyFrozenSet_Type`
- `PyGC_Collect()`

- `PyGC_Disable()`
- `PyGC_Enable()`
- `PyGC_IsEnabled()`
- `PyGILState_Ensure()`
- `PyGILState_GetThisThreadState()`
- `PyGILState_Release()`
- `PyGILState_STATE`
- `PyGetSetDef`
- `PyGetSetDescr_Type`
- `PyImport_AddModule()`
- `PyImport_AddModuleObject()`
- `PyImport_AddModuleRef()`
- `PyImport_AppendInittab()`
- `PyImport_ExecCodeModule()`
- `PyImport_ExecCodeModuleEx()`
- `PyImport_ExecCodeModuleObject()`
- `PyImport_ExecCodeModuleWithPathnames()`
- `PyImport_GetImporter()`
- `PyImport_GetMagicNumber()`
- `PyImport_GetMagicTag()`
- `PyImport_GetModule()`
- `PyImport_GetModuleDict()`
- `PyImport_Import()`
- `PyImport_ImportFrozenModule()`
- `PyImport_ImportFrozenModuleObject()`
- `PyImport_ImportModule()`
- `PyImport_ImportModuleLevel()`
- `PyImport_ImportModuleLevelObject()`
- `PyImport_ImportModuleNoBlock()`
- `PyImport_ReloadModule()`
- `PyIndex_Check()`
- `PyInterpreterState`
- `PyInterpreterState_Clear()`
- `PyInterpreterState_Delete()`
- `PyInterpreterState_Get()`
- `PyInterpreterState_GetDict()`
- `PyInterpreterState_GetID()`
- `PyInterpreterState_New()`
- `PyIter_Check()`
- `PyIter_Next()`

- `PyIter_NextItem()`
- `PyIter_Send()`
- `PyListIter_Type`
- `PyListRevIter_Type`
- `PyList_Append()`
- `PyList_AsTuple()`
- `PyList_GetItem()`
- `PyList_GetItemRef()`
- `PyList_GetSlice()`
- `PyList_Insert()`
- `PyList_New()`
- `PyList_Reverse()`
- `PyList_SetItem()`
- `PyList_SetSlice()`
- `PyList_Size()`
- `PyList_Sort()`
- `PyList_Type`
- `PyLongObject`
- `PyLongRangeIter_Type`
- `PyLong_AsDouble()`
- `PyLong_AsInt()`
- `PyLong_AsInt32()`
- `PyLong_AsInt64()`
- `PyLong_AsLong()`
- `PyLong_AsLongAndOverflow()`
- `PyLong_AsLongLong()`
- `PyLong_AsLongLongAndOverflow()`
- `PyLong_AsNativeBytes()`
- `PyLong_AsSize_t()`
- `PyLong_AsSsize_t()`
- `PyLong_AsUInt32()`
- `PyLong_AsUInt64()`
- `PyLong_AsUnsignedLong()`
- `PyLong_AsUnsignedLongLong()`
- `PyLong_AsUnsignedLongLongMask()`
- `PyLong_AsUnsignedLongMask()`
- `PyLong_AsVoidPtr()`
- `PyLong_FromDouble()`
- `PyLong_FromInt32()`
- `PyLong_FromInt64()`

- `PyLong_FromLong()`
- `PyLong_FromLongLong()`
- `PyLong_FromNativeBytes()`
- `PyLong_FromSize_t()`
- `PyLong_FromSsize_t()`
- `PyLong_FromString()`
- `PyLong_FromUInt32()`
- `PyLong_FromUInt64()`
- `PyLong_FromUnsignedLong()`
- `PyLong_FromUnsignedLongLong()`
- `PyLong_FromUnsignedNativeBytes()`
- `PyLong_FromVoidPtr()`
- `PyLong_GetInfo()`
- `PyLong_Type`
- `PyMap_Type`
- `PyMapping_Check()`
- `PyMapping_GetItemString()`
- `PyMapping_GetOptionalItem()`
- `PyMapping_GetOptionalItemString()`
- `PyMapping_HasKey()`
- `PyMapping_HasKeyString()`
- `PyMapping_HasKeyStringWithError()`
- `PyMapping_HasKeyWithError()`
- `PyMapping_Items()`
- `PyMapping_Keys()`
- `PyMapping_Length()`
- `PyMapping_SetItemString()`
- `PyMapping_Size()`
- `PyMapping_Values()`
- `PyMem_Calloc()`
- `PyMem_Free()`
- `PyMem_Malloc()`
- `PyMem_RawCalloc()`
- `PyMem_RawFree()`
- `PyMem_RawMalloc()`
- `PyMem_RawRealloc()`
- `PyMem_Realloc()`
- `PyMemberDef`
- `PyMemberDescr_Type`
- `PyMember_GetOne()`

- `PyMember_SetOne()`
- `PyMemoryView_FromBuffer()`
- `PyMemoryView_FromMemory()`
- `PyMemoryView_FromObject()`
- `PyMemoryView_GetContiguous()`
- `PyMemoryView_Type`
- `PyMethodDef`
- `PyMethodDescr_Type`
- `PyModuleDef`
- `PyModuleDef_Base`
- `PyModuleDef_Init()`
- `PyModuleDef_Slot`
- `PyModuleDef_Type`
- `PyModule_Add()`
- `PyModule_AddFunctions()`
- `PyModule_AddIntConstant()`
- `PyModule_AddObject()`
- `PyModule_AddObjectRef()`
- `PyModule_AddStringConstant()`
- `PyModule_AddType()`
- `PyModule_Create2()`
- `PyModule_ExecDef()`
- `PyModule_FromDefAndSpec2()`
- `PyModule_GetDef()`
- `PyModule_GetDict()`
- `PyModule_GetFilename()`
- `PyModule_GetFilenameObject()`
- `PyModule_GetName()`
- `PyModule_GetNameObject()`
- `PyModule_GetState()`
- `PyModule_New()`
- `PyModule_NewObject()`
- `PyModule_SetDocString()`
- `PyModule_Type`
- `PyNumber_Absolute()`
- `PyNumber_Add()`
- `PyNumber_And()`
- `PyNumber_AsSsize_t()`
- `PyNumber_Check()`
- `PyNumber_Divmod()`

- `PyNumber_Float()`
- `PyNumber_FloorDivide()`
- `PyNumber_InPlaceAdd()`
- `PyNumber_InPlaceAnd()`
- `PyNumber_InPlaceFloorDivide()`
- `PyNumber_InPlaceLshift()`
- `PyNumber_InPlaceMatrixMultiply()`
- `PyNumber_InPlaceMultiply()`
- `PyNumber_InPlaceOr()`
- `PyNumber_InPlacePower()`
- `PyNumber_InPlaceRemainder()`
- `PyNumber_InPlaceRshift()`
- `PyNumber_InPlaceSubtract()`
- `PyNumber_InPlaceTrueDivide()`
- `PyNumber_InPlaceXor()`
- `PyNumber_Index()`
- `PyNumber_Invert()`
- `PyNumber_Long()`
- `PyNumber_Lshift()`
- `PyNumber_MatrixMultiply()`
- `PyNumber_Multiply()`
- `PyNumber_Negative()`
- `PyNumber_Or()`
- `PyNumber_Positive()`
- `PyNumber_Power()`
- `PyNumber_Remainder()`
- `PyNumber_Rshift()`
- `PyNumber_Subtract()`
- `PyNumber_ToBase()`
- `PyNumber_TrueDivide()`
- `PyNumber_Xor()`
- `PyOS_AfterFork()`
- `PyOS_AfterFork_Child()`
- `PyOS_AfterFork_Parent()`
- `PyOS_BeforeFork()`
- `PyOS_CheckStack()`
- `PyOS_FSPath()`
- `PyOS_InputHook`
- `PyOS_InterruptOccurred()`
- `PyOS_double_to_string()`

- `PyOS_getsig()`
- `PyOS_mystricmp()`
- `PyOS_mystrnicmp()`
- `PyOS_setsig()`
- `PyOS_sighandler_t`
- `PyOS_snprintf()`
- `PyOS_string_to_double()`
- `PyOS_strtol()`
- `PyOS_strtoul()`
- `PyOS_vsnprintf()`
- `PyObject`
- `PyObject.ob_refcnt`
- `PyObject.ob_type`
- `PyObject_ASCII()`
- `PyObject_AsFileDescriptor()`
- `PyObject_Bytes()`
- `PyObject_Call()`
- `PyObject_CallFunction()`
- `PyObject_CallFunctionObjArgs()`
- `PyObject_CallMethod()`
- `PyObject_CallMethodObjArgs()`
- `PyObject_CallNoArgs()`
- `PyObject_CallObject()`
- `PyObject_Calloc()`
- `PyObject_CheckBuffer()`
- `PyObject_ClearWeakRefs()`
- `PyObject_CopyData()`
- `PyObject_DelAttr()`
- `PyObject_DelAttrString()`
- `PyObject_DelItem()`
- `PyObject_DelItemString()`
- `PyObject_Dir()`
- `PyObject_Format()`
- `PyObject_Free()`
- `PyObject_GC_Del()`
- `PyObject_GC_IsFinalized()`
- `PyObject_GC_IsTracked()`
- `PyObject_GC_Track()`
- `PyObject_GC_UnTrack()`
- `PyObject_GenericGetAttr()`

- `PyObject_GenericGetDict()`
- `PyObject_GenericSetAttr()`
- `PyObject_GenericSetDict()`
- `PyObject_GetAIter()`
- `PyObject_GetAttr()`
- `PyObject_GetAttrString()`
- `PyObject_GetBuffer()`
- `PyObject_GetItem()`
- `PyObject_GetIter()`
- `PyObject_GetOptionalAttr()`
- `PyObject_GetOptionalAttrString()`
- `PyObject_GetTypeData()`
- `PyObject_HasAttr()`
- `PyObject_HasAttrString()`
- `PyObject_HasAttrStringWithError()`
- `PyObject_HasAttrWithError()`
- `PyObject_Hash()`
- `PyObject_HashNotImplemented()`
- `PyObject_Init()`
- `PyObject_InitVar()`
- `PyObject_IsInstance()`
- `PyObject_IsSubclass()`
- `PyObject_IsTrue()`
- `PyObject_Length()`
- `PyObject_Malloc()`
- `PyObject_Not()`
- `PyObject_Realloc()`
- `PyObject_Repr()`
- `PyObject_RichCompare()`
- `PyObject_RichCompareBool()`
- `PyObject_SelfIter()`
- `PyObject_SetAttr()`
- `PyObject_SetAttrString()`
- `PyObject_SetItem()`
- `PyObject_Size()`
- `PyObject_Str()`
- `PyObject_Type()`
- `PyObject_Vectorcall()`
- `PyObject_VectorcallMethod()`
- `PyProperty_Type`

- **PyRangeIter_Type**
- **PyRange_Type**
- **PyReversed_Type**
- **PySeqIter_New()**
- **PySeqIter_Type**
- **PySequence_Check()**
- **PySequence_Concat()**
- **PySequence_Contains()**
- **PySequence_Count()**
- **PySequence_DelItem()**
- **PySequence_DelSlice()**
- **PySequence_Fast()**
- **PySequence_GetItem()**
- **PySequence_GetSlice()**
- **PySequence_In()**
- **PySequence_InPlaceConcat()**
- **PySequence_InPlaceRepeat()**
- **PySequence_Index()**
- **PySequence_Length()**
- **PySequence_List()**
- **PySequence_Repeat()**
- **PySequence_SetItem()**
- **PySequence_SetSlice()**
- **PySequence_Size()**
- **PySequence_Tuple()**
- **PySetIter_Type**
- **PySet_Add()**
- **PySet_Clear()**
- **PySet_Contains()**
- **PySet_Discard()**
- **PySet_New()**
- **PySet_Pop()**
- **PySet_Size()**
- **PySet_Type**
- **PySlice_AdjustIndices()**
- **PySlice_GetIndices()**
- **PySlice_GetIndicesEx()**
- **PySlice_New()**
- **PySlice_Type**
- **PySlice_Unpack()**

- `PyState_AddModule()`
- `PyState_FindModule()`
- `PyState_RemoveModule()`
- `PyStructSequence_Desc`
- `PyStructSequence_Field`
- `PyStructSequence_GetItem()`
- `PyStructSequence_New()`
- `PyStructSequence_NewType()`
- `PyStructSequence_SetItem()`
- `PyStructSequence_UnnamedField`
- `PySuper_Type`
- `PySys_Audit()`
- `PySys_AuditTuple()`
- `PySys_FormatStderr()`
- `PySys_FormatStdout()`
- `PySys_GetObject()`
- `PySys_GetXOptions()`
- `PySys_ResetWarnOptions()`
- `PySys_SetArgv()`
- `PySys_SetArgvEx()`
- `PySys_SetObject()`
- `PySys_WriteStderr()`
- `PySys_WriteStdout()`
- `PyThreadState`
- `PyThreadState_Clear()`
- `PyThreadState_Delete()`
- `PyThreadState_Get()`
- `PyThreadState_GetDict()`
- `PyThreadState_GetFrame()`
- `PyThreadState_GetID()`
- `PyThreadState_GetInterpreter()`
- `PyThreadState_New()`
- `PyThreadState_SetAsyncExc()`
- `PyThreadState_Swap()`
- `PyThread_GetInfo()`
- `PyThread_ReInitTLS()`
- `PyThread_acquire_lock()`
- `PyThread_acquire_lock_timed()`
- `PyThread_allocate_lock()`
- `PyThread_create_key()`

- `PyThread_delete_key()`
- `PyThread_delete_key_value()`
- `PyThread_exit_thread()`
- `PyThread_free_lock()`
- `PyThread_get_key_value()`
- `PyThread_get_stacksize()`
- `PyThread_get_thread_ident()`
- `PyThread_get_thread_native_id()`
- `PyThread_init_thread()`
- `PyThread_release_lock()`
- `PyThread_set_key_value()`
- `PyThread_set_stacksize()`
- `PyThread_start_new_thread()`
- `PyThread_tss_alloc()`
- `PyThread_tss_create()`
- `PyThread_tss_delete()`
- `PyThread_tss_free()`
- `PyThread_tss_get()`
- `PyThread_tss_is_created()`
- `PyThread_tss_set()`
- `PyTraceBack_Here()`
- `PyTraceBack_Print()`
- `PyTraceBack_Type`
- `PyTupleIter_Type`
- `PyTuple_GetItem()`
- `PyTuple_GetSlice()`
- `PyTuple_New()`
- `PyTuple_Pack()`
- `PyTuple_SetItem()`
- `PyTuple_Size()`
- `PyTuple_Type`
- `PyTypeObject`
- `PyType_ClearCache()`
- `PyType_Freeze()`
- `PyType_FromMetaclass()`
- `PyType_FromModuleAndSpec()`
- `PyType_FromSpec()`
- `PyType_FromSpecWithBases()`
- `PyType_GenericAlloc()`
- `PyType_GenericNew()`

- `PyType_GetBaseByToken()`
- `PyType_GetFlags()`
- `PyType_GetFullyQualifiedName()`
- `PyType_GetModule()`
- `PyType_GetModuleByDef()`
- `PyType_GetModuleName()`
- `PyType_GetModuleState()`
- `PyType_GetName()`
- `PyType_GetQualName()`
- `PyType_GetSlot()`
- `PyType_GetTypeDataSize()`
- `PyType_IsSubtype()`
- `PyType_Modified()`
- `PyType_Ready()`
- `PyType_Slot`
- `PyType_Spec`
- `PyType_Type`
- `PyUnicodeDecodeError_Create()`
- `PyUnicodeDecodeError_GetEncoding()`
- `PyUnicodeDecodeError_GetEnd()`
- `PyUnicodeDecodeError_GetObject()`
- `PyUnicodeDecodeError_GetReason()`
- `PyUnicodeDecodeError_GetStart()`
- `PyUnicodeDecodeError_SetEnd()`
- `PyUnicodeDecodeError_SetReason()`
- `PyUnicodeDecodeError_SetStart()`
- `PyUnicodeEncodeError_GetEncoding()`
- `PyUnicodeEncodeError_GetEnd()`
- `PyUnicodeEncodeError_GetObject()`
- `PyUnicodeEncodeError_GetReason()`
- `PyUnicodeEncodeError_GetStart()`
- `PyUnicodeEncodeError_SetEnd()`
- `PyUnicodeEncodeError_SetReason()`
- `PyUnicodeEncodeError_SetStart()`
- `PyUnicodeIter_Type`
- `PyUnicodeTranslateError_GetEnd()`
- `PyUnicodeTranslateError_GetObject()`
- `PyUnicodeTranslateError_GetReason()`
- `PyUnicodeTranslateError_GetStart()`
- `PyUnicodeTranslateError_SetEnd()`

- `PyUnicodeTranslateError_SetReason()`
- `PyUnicodeTranslateError_SetStart()`
- `PyUnicode_Append()`
- `PyUnicode_AppendAndDel()`
- `PyUnicode_AsASCIIString()`
- `PyUnicode_AsCharmapString()`
- `PyUnicode_AsDecodedObject()`
- `PyUnicode_AsDecodedUnicode()`
- `PyUnicode_AsEncodedObject()`
- `PyUnicode_AsEncodedString()`
- `PyUnicode_AsEncodedUnicode()`
- `PyUnicode_AsLatin1String()`
- `PyUnicode_AsMBCSString()`
- `PyUnicode_AsRawUnicodeEscapeString()`
- `PyUnicode_AsUCS4()`
- `PyUnicode_AsUCS4Copy()`
- `PyUnicode_AsUTF16String()`
- `PyUnicode_AsUTF32String()`
- `PyUnicode_AsUTF8AndSize()`
- `PyUnicode_AsUTF8String()`
- `PyUnicode_AsUnicodeEscapeString()`
- `PyUnicode_AsWideChar()`
- `PyUnicode_AsWideCharString()`
- `PyUnicode_BuildEncodingMap()`
- `PyUnicode_Compare()`
- `PyUnicode_CompareWithASCIIString()`
- `PyUnicode_Concat()`
- `PyUnicode_Contains()`
- `PyUnicode_Count()`
- `PyUnicode_Decode()`
- `PyUnicode_DecodeASCII()`
- `PyUnicode_DecodeCharmap()`
- `PyUnicode_DecodeCodePageStateful()`
- `PyUnicode_DecodeFSDefault()`
- `PyUnicode_DecodeFSDefaultAndSize()`
- `PyUnicode_DecodeLatin1()`
- `PyUnicode_DecodeLocale()`
- `PyUnicode_DecodeLocaleAndSize()`
- `PyUnicode_DecodeMBCS()`
- `PyUnicode_DecodeMBCSStateful()`

- `PyUnicode_DecodeRawUnicodeEscape()`
- `PyUnicode_DecodeUTF16()`
- `PyUnicode_DecodeUTF16Stateful()`
- `PyUnicode_DecodeUTF32()`
- `PyUnicode_DecodeUTF32Stateful()`
- `PyUnicode_DecodeUTF7()`
- `PyUnicode_DecodeUTF7Stateful()`
- `PyUnicode_DecodeUTF8()`
- `PyUnicode_DecodeUTF8Stateful()`
- `PyUnicode_DecodeUnicodeEscape()`
- `PyUnicode_EncodeCodePage()`
- `PyUnicode_EncodeFSDefault()`
- `PyUnicode_EncodeLocale()`
- `PyUnicode_Equal()`
- `PyUnicode_EqualToUTF8()`
- `PyUnicode_EqualToUTF8AndSize()`
- `PyUnicode_FSConverter()`
- `PyUnicode_FSDecoder()`
- `PyUnicode_Find()`
- `PyUnicode_FindChar()`
- `PyUnicode_Format()`
- `PyUnicode_FromEncodedObject()`
- `PyUnicode_FromFormat()`
- `PyUnicode_FromFormatV()`
- `PyUnicode_FromObject()`
- `PyUnicode_FromOrdinal()`
- `PyUnicode_FromString()`
- `PyUnicode_FromStringAndSize()`
- `PyUnicode_FromWideChar()`
- `PyUnicode_GetDefaultEncoding()`
- `PyUnicode_GetLength()`
- `PyUnicode_InternFromString()`
- `PyUnicode_InternInPlace()`
- `PyUnicode_IsIdentifier()`
- `PyUnicode_Join()`
- `PyUnicode_Partition()`
- `PyUnicode_RPartition()`
- `PyUnicode_RSplit()`
- `PyUnicode_ReadChar()`
- `PyUnicode_Replace()`

- PyUnicode_Resize()
- PyUnicode_RichCompare()
- PyUnicode_Split()
- PyUnicode_Splitlines()
- PyUnicode_Substring()
- PyUnicode_Tailmatch()
- PyUnicode_Translate()
- PyUnicode_Type
- PyUnicode_WriteChar()
- PyVarObject
- PyVarObject.ob_base
- PyVarObject.ob_size
- PyVectorcall_Call()
- PyVectorcall_NARGS()
- PyWeakReference
- PyWeakref_GetObject()
- PyWeakref_GetRef()
- PyWeakref_NewProxy()
- PyWeakref_NewRef()
- PyWrapperDescr_Type
- PyWrapper_New()
- PyZip_Type
- Py_ASNATIVEBYTES_ALLOW_INDEX
- Py_ASNATIVEBYTES_BIG_ENDIAN
- Py_ASNATIVEBYTES_DEFAULTS
- Py_ASNATIVEBYTES_LITTLE_ENDIAN
- Py_ASNATIVEBYTES_NATIVE_ENDIAN
- Py_ASNATIVEBYTES_REJECT_NEGATIVE
- Py_ASNATIVEBYTES_UNSIGNED_BUFFER
- Py_AUDIT_READ
- Py_AddPendingCall()
- Py_AtExit()
- Py_BEGIN_ALLOW_THREADS
- Py_BLOCK_THREADS
- Py_BuildValue()
- Py_BytesMain()
- Py_CompileString()
- Py_DecRef()
- Py_DecodeLocale()
- Py_END_ALLOW_THREADS

- `Py_EncodeLocale()`
- `Py_EndInterpreter()`
- `Py_EnterRecursiveCall()`
- `Py_Exit()`
- `Py_FatalError()`
- `Py_FileSystemDefaultEncodeErrors`
- `Py_FileSystemDefaultEncoding`
- `Py_Finalize()`
- `Py_FinalizeEx()`
- `Py_GenericAlias()`
- `Py_GenericAliasType`
- `Py_GetBuildInfo()`
- `Py_GetCompiler()`
- `Py_GetConstant()`
- `Py_GetConstantBorrowed()`
- `Py_GetCopyright()`
- `Py_GetExecPrefix()`
- `Py_GetPath()`
- `Py_GetPlatform()`
- `Py_GetPrefix()`
- `Py_GetProgramFullPath()`
- `Py_GetProgramName()`
- `Py_GetPythonHome()`
- `Py_GetRecursionLimit()`
- `Py_GetVersion()`
- `Py_HasFileSystemDefaultEncoding`
- `Py_IncRef()`
- `Py_Initialize()`
- `Py_InitializeEx()`
- `Py_Is()`
- `Py_IsFalse()`
- `Py_IsFinalizing()`
- `Py_IsInitialized()`
- `Py_IsNone()`
- `Py_IsTrue()`
- `Py_LeaveRecursiveCall()`
- `Py_Main()`
- `Py_MakePendingCalls()`
- `Py_NewInterpreter()`
- `Py_NewRef()`

- `Py_PACK_FULL_VERSION()`
- `Py_PACK_VERSION()`
- `Py_READONLY`
- `Py_REFCNT()`
- `Py_RELATIVE_OFFSET`
- `Py_ReprEnter()`
- `Py_ReprLeave()`
- `Py_SetProgramName()`
- `Py_SetPythonHome()`
- `Py_SetRecursionLimit()`
- `Py_TPFLAGS_BASETYPE`
- `Py_TPFLAGS_DEFAULT`
- `Py_TPFLAGS_HAVE_GC`
- `Py_TPFLAGS_HAVE_VECTORCALL`
- `Py_TPFLAGS_ITEMS_AT_END`
- `Py_TPFLAGS_METHOD_DESCRIPTOR`
- `Py_TP_USE_SPEC`
- `Py_TYPE()`
- `Py_T_BOOL`
- `Py_T_BYTE`
- `Py_T_CHAR`
- `Py_T_DOUBLE`
- `Py_T_FLOAT`
- `Py_T_INT`
- `Py_T_LONG`
- `Py_T_LONGLONG`
- `Py_T_OBJECT_EX`
- `Py_T_PYSSIZET`
- `Py_T_SHORT`
- `Py_T_STRING`
- `Py_T_STRING_INPLACE`
- `Py_T_UBYTE`
- `Py_T_UINT`
- `Py_T_ULONG`
- `Py_T_ULONGLONG`
- `Py_T_USHORT`
- `Py_UCS4`
- `Py_UNBLOCK_THREADS`
- `Py_UTF8Mode`
- `Py_VaBuildValue()`

- `Py_Version`
- `Py_XNewRef()`
- `Py_am_aiter`
- `Py_am_anext`
- `Py_am_await`
- `Py_am_send`
- `Py_bf_getbuffer`
- `Py_bf_releasebuffer`
- `Py_buffer`
- `Py_intptr_t`
- `Py_mod_create`
- `Py_mod_exec`
- `Py_mod_gil`
- `Py_mod_multiple_interpreters`
- `Py_mp_ass_subscript`
- `Py_mp_length`
- `Py_mp_subscript`
- `Py_nb_absolute`
- `Py_nb_add`
- `Py_nb_and`
- `Py_nb_bool`
- `Py_nb_divmod`
- `Py_nb_float`
- `Py_nb_floor_divide`
- `Py_nb_index`
- `Py_nb_inplace_add`
- `Py_nb_inplace_and`
- `Py_nb_inplace_floor_divide`
- `Py_nb_inplace_lshift`
- `Py_nb_inplace_matrix_multiply`
- `Py_nb_inplace_multiply`
- `Py_nb_inplace_or`
- `Py_nb_inplace_power`
- `Py_nb_inplace_remainder`
- `Py_nb_inplace_rshift`
- `Py_nb_inplace_subtract`
- `Py_nb_inplace_true_divide`
- `Py_nb_inplace_xor`
- `Py_nb_int`
- `Py_nb_invert`

- `Py_nb_lshift`
- `Py_nb_matrix_multiply`
- `Py_nb_multiply`
- `Py_nb_negative`
- `Py_nb_or`
- `Py_nb_positive`
- `Py_nb_power`
- `Py_nb_remainder`
- `Py_nb_rshift`
- `Py_nb_subtract`
- `Py_nb_true_divide`
- `Py_nb_xor`
- `Py_sq_ass_item`
- `Py_sq_concat`
- `Py_sq_contains`
- `Py_sq_inplace_concat`
- `Py_sq_inplace_repeat`
- `Py_sq_item`
- `Py_sq_length`
- `Py_sq_repeat`
- `Py_ssize_t`
- `Py_tp_alloc`
- `Py_tp_base`
- `Py_tp_bases`
- `Py_tp_call`
- `Py_tp_clear`
- `Py_tp_dealloc`
- `Py_tp_del`
- `Py_tp_descr_get`
- `Py_tp_descr_set`
- `Py_tp_doc`
- `Py_tp_finalize`
- `Py_tp_free`
- `Py_tp_getattr`
- `Py_tp_getattro`
- `Py_tp_getset`
- `Py_tp_hash`
- `Py_tp_init`
- `Py_tp_is_gc`
- `Py_tp_iter`

- `Py_tp_iternext`
- `Py_tp_members`
- `Py_tp_methods`
- `Py_tp_new`
- `Py_tp_repr`
- `Py_tp_richcompare`
- `Py_tp_setattr`
- `Py_tp_setattro`
- `Py_tp_str`
- `Py_tp_token`
- `Py_tp_traverse`
- `Py_tp_vectorcall`
- `Py_uintptr_t`
- `allocfunc`
- `binaryfunc`
- `descrgetfunc`
- `descrsetfunc`
- `destructor`
- `getattrfunc`
- `getattrofunc`
- `getbufferproc`
- `getiterfunc`
- `getter`
- `hashfunc`
- `initproc`
- `inquiry`
- `iternextfunc`
- `lenfunc`
- `newfunc`
- `objobjargproc`
- `objobjproc`
- `releasebufferproc`
- `reprfunc`
- `richcmpfunc`
- `setattrfunc`
- `setattrofunc`
- `setter`
- `ssizeargfunc`
- `ssizeobjargproc`
- `ssizessizeargfunc`

- **`ssizessizeobjargproc`**
- **`symtable`**
- **`ternaryfunc`**
- **`traverseproc`**
- **`unaryfunc`**
- **`vectorcallfunc`**
- **`visitproc`**

La capa de muy alto nivel

Las funciones en este capítulo te permitirán ejecutar código fuente de Python desde un archivo o un búfer, pero no te permitirán interactuar de una manera detallada con el intérprete.

Several of these functions accept a start symbol from the grammar as a parameter. The available start symbols are **`Py_eval_input`**, **`Py_file_input`**, **`Py_single_input`**, and **`Py_func_type_input`**. These are described following the functions which accept them as parameters.

Note also that several of these functions take FILE* parameters. One particular issue which needs to be handled carefully is that the **`FILE`** structure for different C libraries can be different and incompatible. Under Windows (at least), it is possible for dynamically linked extensions to actually use different libraries, so care should be taken that FILE* parameters are only passed to these functions if it is certain that they were created by the same library that the Python runtime is using.

int PyRun_AnyFile(FILE *fp, const char *filename)

> Esta es una interfaz simplificada para **`PyRun_AnyFileExFlags()`** más abajo, dejando *closeit* establecido a `0` y *flags* establecido a `NULL`.

int PyRun_AnyFileFlags(FILE *fp, const char *filename, PyCompilerFlags *flags)

> Esta es una interfaz simplificada para **`PyRun_AnyFileExFlags()`** más abajo, dejando *closeit* establecido a `0`.

int PyRun_AnyFileEx(FILE *fp, const char *filename, int closeit)

> Esta es una interfaz simplificada para **`PyRun_AnyFileExFlags()`** más abajo, dejando *flags*

establecido a `NULL`.

int PyRun_AnyFileExFlags(FILE *fp, const char *filename, int closeit, PyCompilerFlags *flags)

If *fp* refers to a file associated with an interactive device (console or terminal input or Unix pseudo-terminal), return the value of **`PyRun_InteractiveLoop()`**, otherwise return the result of **`PyRun_SimpleFile()`**. *filename* is decoded from the filesystem encoding (**`sys.getfilesystemencoding()`**). If *filename* is `NULL`, this function uses `"???"` as the filename. If *closeit* is true, the file is closed before `PyRun_SimpleFileExFlags()` returns.

int PyRun_SimpleString(const char *command)

This is a simplified interface to **`PyRun_SimpleStringFlags()`** below, leaving the **`PyCompilerFlags`*** argument set to `NULL`.

int PyRun_SimpleStringFlags(const char *command, PyCompilerFlags *flags)

Ejecuta el código fuente de Python desde *command* en el módulo **`__main__`** de acuerdo con el argumento *flags*. Si **`__main__`** aún no existe, se crea. Retorna `0` en caso de éxito o `-1` si se produjo una excepción. Si hubo un error, no hay forma de obtener la información de excepción. Para el significado de *flags*, ver abajo.

Note that if an otherwise unhandled **`SystemExit`** is raised, this function will not return `-1`, but exit the process, as long as **`PyConfig.inspect`** is zero.

int PyRun_SimpleFile(FILE *fp, const char *filename)

Esta es una interfaz simplificada para **`PyRun_SimpleStringFlags()`** más abajo, dejando *closeit* establecido a `0` y *flags* establecido a `NULL`.

int PyRun_SimpleFileEx(FILE *fp, const char *filename, int closeit)

Esta es una interfaz simplificada para **`PyRun_SimpleStringFlags()`** más abajo, dejando *flags* establecido a `NULL`.

int PyRun_SimpleFileExFlags(FILE *fp, const char *filename, int closeit, PyCompilerFlags *flags)

Similar a **`PyRun_SimpleStringFlags()`**, pero el código fuente de Python se lee desde *fp* en lugar de una cadena de caracteres en memoria. *filename* debe ser el nombre del fichero, se decodifica desde filesystem encoding and error handler. Si *closeit* es verdadero, el fichero se cierra antes de que `PyRun_SimpleFileExFlags()` retorne.

Nota

En Windows, *fp* debe abrirse en modo binario (por ejemplo, `fopen(filename, "rb")`. De lo contrario, Python puede no manejar correctamente el archivo de script con la terminación de línea LF.

int PyRun_InteractiveOneObject(FILE *fp, PyObject *filename, PyCompilerFlags *flags)

Read and execute a single statement from a file associated with an interactive device according to the *flags* argument. The user will be prompted using `sys.ps1` and `sys.ps2`. *filename* must be a Python **`str`** object.

Retorna `0` cuando la entrada se ejecuta con éxito, `-1` si hubo una excepción, o un código de error del archivo `errcode.h` distribuido como parte de Python si hubo un error de análisis gramatical. (Tenga en cuenta que `errcode.h` no está incluido en `Python.h`, por lo que debe incluirse específicamente si es necesario).

int PyRun_InteractiveOne(FILE *fp, const char *filename)

Esta es una interfaz simplificada para **`PyRun_InteractiveOneFlags()`** más abajo, dejando *flags* establecido a `NULL`.

int PyRun_InteractiveOneFlags(FILE *fp, const char *filename, PyCompilerFlags *flags)

Similar to **`PyRun_InteractiveOneObject()`**, but *filename* is a const char*, which is decoded from the filesystem encoding and error handler.

int PyRun_InteractiveLoop(FILE *fp, const char *filename)

Esta es una interfaz simplificada para **`PyRun_InteractiveLoopFlags()`** más abajo, dejando *flags* establecido a `NULL`.

int PyRun_InteractiveLoopFlags(FILE *fp, const char *filename, PyCompilerFlags *flags)

Lee y ejecuta declaraciones de un archivo asociado con un dispositivo interactivo hasta llegar al EOF. Se le solicitará al usuario usando `sys.ps1` y `sys.ps2`.*filename* se decodifica a partir del manejador de codificación y errores del sistema de archivos. Retorna `0` en EOF o un número negativo en caso de falla.

int (*PyOS_InputHook)(void)

Part of the Stable ABI.
Can be set to point to a function with the prototype `int func(void)`. The function will be called when Python's interpreter prompt is about to become idle and wait for user input from the terminal. The return value is ignored. Overriding this hook can be used to integrate the interpreter's prompt with other event loops, as done in `Modules/_tkinter.c` in the Python source code.

Distinto en la versión 3.12: This function is only called from the main interpreter.

char *(*PyOS_ReadlineFunctionPointer)(FILE*, FILE*, const char*)

Se puede configurar para que apunte a una función con el prototipo `char *func (FILE *stdin, FILE *stdout, char *prompt)`, sobrescribiendo la función predeterminada utilizada para leer una sola línea de entrada desde el intérprete. Se espera que la función genere la cadena de caracteres *prompt* si no es `NULL`, y luego lea una línea de entrada del archivo de entrada estándar proporcionado, retornando la cadena de caracteres resultante. Por ejemplo, el módulo **`readline`** establece este enlace para proporcionar funciones de edición de línea y finalización de tabulación.

El resultado debe ser una cadena de caracteres alocado por **`PyMem_RawMalloc()`** o **`PyMem_RawRealloc()`**, o `NULL` si ocurre un error.

Distinto en la versión 3.4: El resultado debe ser alocado por **`PyMem_RawMalloc()`** o **`PyMem_RawRealloc()`**, en vez de ser alocado por **`PyMem_Malloc()`** o **`PyMem_Realloc()`**.

Distinto en la versión 3.12: This function is only called from the main interpreter.

PyObject *PyRun_String(const char *str, int start, PyObject *globals, PyObject *locals)

Return value: New reference.
Esta es una interfaz simplificada para **`PyRun_StringFlags()`** más abajo, dejando *flags* establecido a `NULL`.

PyObject *PyRun_StringFlags(const char *str, int start, PyObject *globals, PyObject *locals, PyCompilerFlags *flags)

Return value: New reference.
Execute Python source code from *str* in the context specified by the objects *globals* and *locals* with the compiler flags specified by *flags*. *globals* must be a dictionary; *locals* can be any object that implements the mapping protocol. The parameter *start* specifies the start symbol and must be one of the available start symbols.

Retorna el resultado de ejecutar el código como un objeto Python, o `NULL`” si se produjo una excepción.

PyObject *PyRun_File(FILE *fp, const char *filename, int start, PyObject *globals, PyObject *locals)

Return value: New reference.
Esta es una interfaz simplificada para **`PyRun_FileExFlags()`** más abajo, dejando *closeit* establecido a `0` y *flags* establecido a `NULL`.

PyObject *PyRun_FileEx(FILE *fp, const char *filename, int start, PyObject *globals, PyObject *locals, int closeit)

Return value: New reference.
Esta es una interfaz simplificada para **`PyRun_FileExFlags()`** más abajo, dejando *flags* establecido a `NULL`.

PyObject *PyRun_FileFlags(FILE *fp, const char *filename, int start, PyObject *globals, PyObject *locals, PyCompilerFlags *flags)

Return value: New reference.
Esta es una interfaz simplificada para **`PyRun_FileExFlags()`** más abajo, dejando *closeit* establecido a `0`.

PyObject *PyRun_FileExFlags(FILE *fp, const char *filename, int start, PyObject *globals, PyObject *locals, int closeit, PyCompilerFlags *flags)

Return value: New reference.
Similar a **`PyRun_StringFlags()`**, pero el código fuente de Python se lee de *fp* en lugar de una cadena de caracteres en memoria. *filename* debe ser el nombre del fichero, es decodificado desde el filesystem encoding and error handler. Si *closeit* es verdadero, el fichero se cierra antes de que **`PyRun_FileExFlags()`** retorne.

PyObject *Py_CompileString(const char *str, const char *filename, int start)

Return value: New reference. Part of the Stable ABI.
Esta es una interfaz simplificada para **`Py_CompileStringFlags()`** más abajo, dejando *flags* establecido a `NULL`.

PyObject *Py_CompileStringFlags(const char *str, const char *filename, int start, PyCompilerFlags *flags)

Return value: New reference.
Esta es una interfaz simplificada para **`Py_CompileStringExFlags()`** más abajo, con *optimize* establecido a `-1`.

PyObject *Py_CompileStringObject(const char *str, PyObject *filename, int start, PyCompilerFlags *flags, int optimize)

Return value: New reference.
Parse and compile the Python source code in *str*, returning the resulting code object. The start symbol is given by *start*; this can be used to constrain the code which can be compiled and should be available start symbols. The filename specified by *filename* is used to construct the code object and may appear in tracebacks or **`SyntaxError`** exception messages. This returns `NULL` if the code cannot be parsed or compiled.

El número entero *optimize* especifica el nivel de optimización del compilador; un valor de `-1` selecciona el nivel de optimización del intérprete como se indica en las opciones **`-O`**. Los niveles explícitos son `0` (sin optimización; `__debug__` es verdadero), `1` (los *asserts* se eliminan, `__debug__` es falso) o `2` (los docstrings también se eliminan))

Added in version 3.4.

PyObject *Py_CompileStringExFlags(const char *str, const char *filename, int start, PyCompilerFlags *flags, int optimize)

Return value: New reference.
Como **`Py_CompileStringObject()`**, pero *filename* es una

cadena de bytes decodificada a partir del manejador de codificación y errores del sistema de archivos.

Added in version 3.2.

PyObject *PyEval_EvalCode(PyObject *co, PyObject *globals, PyObject *locals)

Return value: New reference. Part of the Stable ABI.
Esta es una interfaz simplificada para **`PyEval_EvalCodeEx()`**, con solo el objeto de código y las variables globales y locales. Los otros argumentos están establecidos en `NULL`.

PyObject *PyEval_EvalCodeEx(PyObject *co, PyObject *globals, PyObject *locals, PyObject *const *args, int argcount, PyObject *const *kws, int kwcount, PyObject *const *defs, int defcount, PyObject *kwdefs, PyObject *closure)

Return value: New reference. Part of the Stable ABI.
Evaluar un objeto de código precompilado, dado un entorno particular para su evaluación. Este entorno consta de un diccionario de variables globales, un objeto de mapeo de variables locales, arreglos de argumentos, palabras clave y valores predeterminados, un diccionario de valores predeterminados para argumentos keyword-only y una tupla de cierre de células.

PyObject *PyEval_EvalFrame(PyFrameObject *f)

Return value: New reference. Part of the Stable ABI.
Evaluar un marco de ejecución. Esta es una interfaz simplificada para **`PyEval_EvalFrameEx()`**, para compatibilidad con versiones anteriores.

PyObject *PyEval_EvalFrameEx(PyFrameObject *f, int throwflag)

Return value: New reference. Part of the Stable ABI.
Esta es la función principal sin barnizar de la interpretación de Python. El objeto de código asociado con el marco de ejecución del marco *f* se ejecuta, interpretando el código de

bytes y ejecutando llamadas según sea necesario. El parámetro adicional *throwflag* se puede ignorar por lo general; si es verdadero, entonces se lanza una excepción de inmediato; esto se usa para los métodos **`throw()`** de objetos generadores.

Distinto en la versión 3.4: Esta función ahora incluye una afirmación de depuración para ayudar a garantizar que no descarte silenciosamente una excepción activa.

int PyEval_MergeCompilerFlags(PyCompilerFlags *cf)

Esta función cambia los flags del marco de evaluación actual, y retorna verdad (*true*) en caso de éxito, falso (*false*) en caso de fallo.

struct PyCompilerFlags

Esta es la estructura usada para contener los flags del compilador. En casos donde el código es sólo compilado, es pasado como `int flags`, y en casos donde el código es ejecutado, es pasado como `PyCompilerFlags *flags`. En este caso, `from __future__ import` puede modificar los *flags*.

Whenever `PyCompilerFlags *flags` is `NULL`, **`cf_flags`** is treated as equal to `0`, and any modification due to `from __future__ import` is discarded.

int cf_flags

Flags del compilador.

int cf_feature_version

cf_feature_version es la versión menor de Python. Debe ser inicializado a `PY_MINOR_VERSION`.

The field is ignored by default, it is used if and only if `PyCF_ONLY_AST` flag is set in **`cf_flags`**.

Distinto en la versión 3.8: Agregado el campo *cf_feature_version*.

The available compiler flags are accessible as macros:

PyCF_ALLOW_TOP_LEVEL_AWAIT
PyCF_ONLY_AST
PyCF_OPTIMIZED_AST
PyCF_TYPE_COMMENTS

See compiler flags in documentation of the **`ast`** Python module, which exports these constants under the same names.

The «`PyCF`» flags above can be combined with «`CO_FUTURE`» flags such as **`CO_FUTURE_ANNOTATIONS`** to enable features normally selectable using future statements. See Code Object Flags for a complete list.

Available start symbols

int Py_eval_input

El símbolo de inicio de la gramática de Python para expresiones aisladas; para usar con **`Py_CompileString()`**.

int Py_file_input

El símbolo de inicio de la gramática de Python para secuencias de declaración leídos desde un archivo u otra fuente; para usar con **`Py_CompileString()`**. Este es el símbolo usado cuando se compile un código fuente en Python arbitrariamente largo.

int Py_single_input

El símbolo de inicio de la gramática de Python para una declaración única; para usar con **`Py_CompileString()`**. Este es el símbolo usado para el bucle interactivo del intérprete.

int Py_func_type_input

The start symbol from the Python grammar for a function type; for use with **`Py_CompileString()`**. This is used to parse «signature type comments» from **PEP 484** [https://

peps.python.org/pep-0484/].

This requires the `PyCF_ONLY_AST` flag to be set.

Ver también

- `ast.FunctionType`
- PEP 484 [https://peps.python.org/pep-0484/]

Added in version 3.8.

Stack Effects

Ver también

`dis.stack_effect()`

PY_INVALID_STACK_EFFECT

Sentinel value representing an invalid stack effect.

This is currently equivalent to `INT_MAX`.

Added in version 3.8.

int PyCompile_OpcodeStackEffect(int opcode, int oparg)

Compute the stack effect of *opcode* with argument *oparg*.

On success, this function returns the stack effect; on failure, this returns `PY_INVALID_STACK_EFFECT`.

Added in version 3.4.

int PyCompile_OpcodeStackEffectWithJump(int opcode, int oparg, int jump)

Similar to `PyCompile_OpcodeStackEffect()`, but don't include the stack effect of jumping if *jump* is zero.

If *jump* is `0`, this will not include the stack effect of jumping, but if *jump* is `1` or `-1`, this will include it.

On success, this function returns the stack effect; on failure, this returns **`PY_INVALID_STACK_EFFECT`**.

Added in version 3.8.

Conteo de referencias

Las funciones y macros de esta sección se utilizan para administrar conteos de referencia de objetos en Python.

Py_ssize_t Py_REFCNT(PyObject *o)

Part of the Stable ABI since version 3.14.
Obtiene el recuento de referencias para el objeto de Python *o*.

Ten cuenta que el valor devuelto puede que no reflejar cuantas referencias al objecto existen realmente. Por ejemplo, algunos objetos son immortal y tienen un refcount muy alto que no refleja el número real de referencias. Por lo tanto, no confíes en que el valor devuelto sea preciso, salvo cuando sea 0 o 1.

Usa la función `Py_SET_REFCNT()` para establecer la cuenta de referencias de un objeto.

Nota

On free-threaded builds of Python, returning 1 isn't sufficient to determine if it's safe to treat *o* as having no access by other threads. Use `PyUnstable_Object_IsUniquelyReferenced()` for that instead.

See also the function `PyUnstable_Object_IsUniqueReferencedTemporary()`.

Distinto en la versión 3.10: `Py_REFCNT()` se convierte en una función estática en línea.

Distinto en la versión 3.11: El tipo de parámetro ya no es const PyObject*.

void Py_SET_REFCNT(PyObject *o, Py_ssize_t refcnt)

Establece la cuenta de referencias del objeto *o* al valor *refcnt*.

En compilación de Python con Free Threading, si *refcnt* es mas mayor que `UINT32_MAX`, el objeto se convierte en immortal.

Esta función no afecta a los objetos immortal.

Added in version 3.9.

Distinto en la versión 3.12: Los objetos inmortales no se modifican.

void Py_INCREF(PyObject *o)

Indica tomar una nueva strong reference al objeto *o*, lo que indica que está en uso y no debe ser destruido.

Esta función no afecta a los objetos immortal.

Esta función se usa generalmente para convertir un borrowed reference en un strong reference en su lugar. La función **`Py_NewRef()`** se puede utilizar para crear un nuevo strong reference.

Cuando se termine de usar el objeto, se libera llamando a **`Py_DECREF()`**.

El objeto no debe ser `NULL`; si no está seguro de que no sea `NULL`, use **`Py_XINCREF()`**.

No esperes que esta función modifique realmente *o* de ninguna manera. Al menos para **algunos objetos** [https://peps.python.org/pep-0683/], esta función no tiene ningún efecto.

Distinto en la versión 3.12: Los objetos inmortales no se modifican.

void Py_XINCREF(PyObject *o)

Similar a **`Py_INCREF()`**, pero el objeto *o* puede ser `NULL`, en cuyo caso esto no tiene efecto.

Ver también **`Py_XNewRef()`**.

PyObject *Py_NewRef(PyObject *o)

Part of the Stable ABI since version 3.10.
Crea una nueva strong reference a un objeto: llama a **`Py_INCREF()`** sobre *o* y devuelve el objeto *o*.

Cuando la strong reference ya no sea necesaria, se debe llamar a **`Py_DECREF()`** para disminuir el recuento de referencias del objeto.

El objeto *o* no debe ser `NULL`; use **`Py_XNewRef()`** si *o* puede ser `NULL`.

Por ejemplo:

```
Py_INCREF(obj);
self->attr = obj;
```

puede ser escrito como:

```
self->attr = Py_NewRef(obj);
```

Ver también **`Py_INCREF()`**.

Added in version 3.10.

PyObject *Py_XNewRef(PyObject *o)

Part of the Stable ABI since version 3.10.
Similar a **`Py_NewRef()`**, pero el objeto *o* puede ser NULL.

Si el objeto *o* es `NULL`, la función solo retorna `NULL`.

Added in version 3.10.

void Py_DECREF(PyObject *o)

Libera una strong reference al objeto *o*, indicando que la referencia ya no se usa.

Esta función no afecta a los objetos immortal.

Una vez que la última strong reference sea liberada (por ejemplo, cuando la cuenta de referencias del objeto llegue a 0), se invoca la función de desasignación del tipo de objeto (la cual no debe ser `NULL`).

Esta función se usa generalmente para eliminar un strong reference antes de salir de su alcance.

El objeto no debe ser `NULL`; si no está seguro de que no sea `NULL`, use **`Py_XINCREF()`**.

No esperes que esta función modifique realmente *o* de ninguna manera. Al menos para **algunos objetos** [https://peps.python.org/pep-0683/], esta función no tiene ningún efecto.

Advertencia

La función de desasignación puede hacer que se invoque un código arbitrario de Python (por ejemplo, cuando se desasigna una instancia de clase con el método **`__del__()`**). Mientras las excepciones en dicho código no sean propagadas, el código ejecutado tendrá acceso libre a todas las variables globales de Python. Esto significa que cualquier objeto al que se pueda acceder desde una variable global debería estar en un estado coherente antes de invocar a **`Py_DECREF()`**. Por ejemplo, el código para eliminar un objeto de una lista debe copiar una referencia al objeto eliminado en una variable temporal, actualizar la estructura de datos de la lista y luego llamar a **`Py_DECREF()`** para la variable temporal.

Distinto en la versión 3.12: Los objetos inmortales no se modifican.

void Py_XDECREF(PyObject *o)

Similar a **`Py_DECREF()`**, pero el objeto *o* puede ser `NULL`, en cuyo caso esto no tendría efecto alguno. El mismo aviso de **`Py_DECREF()`** aplica aquí también.

void Py_CLEAR(PyObject *o)

Libera una strong reference del objeto *o*. El objeto puede ser `NULL`, en cuyo caso el macro no tiene efecto; de lo contrario, el efecto es el mismo que el de **`Py_DECREF()`**, excepto que el argumento también se establece en `NULL`. La advertencia de **`Py_DECREF()`** no se aplica en este caso, ya que el macro usa cuidadosamente una variable temporal y asigna `NULL` al argumento antes de liberar la referencia.

Es buena idea usar este macro al liberar una referencia de un objeto que podría ser recorrido durante la recolección de basura.

Distinto en la versión 3.12: Ahora, el macro argumento solo se evalúa una vez. Si el argumento tiene efectos secundarios, estos ya no se duplican.

void Py_IncRef(PyObject *o)

Part of the Stable ABI.
Indica la toma de una nueva strong reference al objeto *o*. Es una versión en forma de función de **`Py_XINCREF()`**. Puede utilizarse para la integración dinámica de Python en tiempo de ejecución.

void Py_DecRef(PyObject *o)

Part of the Stable ABI.
Libera una strong reference al objeto *o*. Una versión en forma de función de **`Py_XDECREF()`**. Puede utilizarse para la integración dinámica de Python en tiempo de ejecución.

Py_SETREF(dst, src)

Un macro que libera de forma segura un strong reference al objeto *dst* y establece *dst* al valor *src*.

Como en el caso de **`Py_CLEAR()`**, el código «obvio» puede ser mortal:

```
Py_DECREF(dst);
```

```
dst = src;
```

La forma segura es:

```
Py_SETREF(dst, src);
```

That arranges to set *dst* to *src* *before* releasing the reference to the old value of *dst*, so that any code triggered as a side-effect of *dst* getting torn down no longer believes *dst* points to a valid object.

Added in version 3.6.

Distinto en la versión 3.12: Los macro argumentos ahora solo se evalúan una vez. Si algún argumento tiene efectos secundarios, estos ya no se duplican.

Py_XSETREF(dst, src)

Un variante del macro **`Py_SETREF`** que usa **`Py_XDECREF()`** en lugar de **`Py_DECREF()`**.

Added in version 3.6.

Distinto en la versión 3.12: Los macro argumentos ahora solo se evalúan una vez. Si algún argumento tiene efectos secundarios, estos ya no se duplican.

Manejo de excepciones

Las funciones descritas en este capítulo le permitirán manejar y lanzar excepciones de Python. Es importante comprender algunos de los conceptos básicos del manejo de excepciones de Python. Funciona de manera similar a la variable POSIX **`errno`**: hay un indicador global (por hilo) del último error que ocurrió. La mayoría de las funciones de C API no borran esto en caso de éxito, pero lo configurarán para indicar la causa del error en caso de falla. La mayoría de las funciones de C API también retornan un indicador de error, generalmente `NULL` si se supone que retornan un puntero, o `-1` si retornan un número entero (excepción: las funciones `PyArg_*` retornan `1` para el éxito y `0` para el fracaso).

Concretamente, el indicador de error consta de tres punteros de objeto: el tipo de excepción, el valor de la excepción y el objeto de rastreo. Cualquiera de esos punteros puede ser `NULL` si no está configurado (aunque algunas combinaciones están prohibidas, por ejemplo, no puede tener un rastreo no `NULL` si el tipo de excepción es `NULL`).

Cuando una función debe fallar porque alguna función que llamó falló, generalmente no establece el indicador de error; la función que llamó ya lo configuró. Es responsable de manejar el error y borrar la excepción o regresar después de limpiar cualquier recurso que tenga (como referencias de objetos o asignaciones de memoria); debería *no* continuar normalmente si no está preparado para manejar el error. Si regresa debido a un error, es importante indicarle a la persona que llama que se ha establecido un error. Si el error no se maneja o se propaga cuidadosamente, es posible que las llamadas adicionales a la API de Python/C no se comporten como se espera y pueden fallar de manera misteriosa.

Nota

The error indicator is **not** the result of **`sys.exc_info()`**. The former corresponds to an exception that is not yet caught (and is therefore still propagating), while the latter returns an exception after it is caught (and has therefore stopped propagating).

Impresión y limpieza

void PyErr_Clear()

Part of the Stable ABI.
Borra el indicador de error. Si el indicador de error no está configurado, no hay efecto.

void PyErr_PrintEx(int set_sys_last_vars)

Part of the Stable ABI.
Imprime un rastreo estándar en `sys.stderr` y borra el indicador de error. **A menos que** el error sea un `Salida del sistema`, en ese caso no se imprime ningún rastreo y el proceso de Python se cerrará con el código de error especificado por la instancia de `Salida del sistema`.

Llame a esta función **solo** cuando el indicador de error está configurado. De lo contrario, provocará un error fatal!

If *set_sys_last_vars* is nonzero, the variable **`sys.last_exc`** is set to the printed exception. For backwards compatibility, the deprecated variables **`sys.last_type`**, **`sys.last_value`** and **`sys.last_traceback`** are also set to the type, value and traceback of this exception, respectively.

Distinto en la versión 3.12: The setting of **`sys.last_exc`** was added.

void PyErr_Print()

Part of the Stable ABI.
Alias para `PyErr_PrintEx(1)`.

void PyErr_WriteUnraisable(PyObject *obj)

Part of the Stable ABI.

Llama **`sys.unraisablehook()`** utilizando la excepción actual y el argumento *obj*.

This utility function prints a warning message to `sys.stderr` when an exception has been set but it is impossible for the interpreter to actually raise the exception. It is used, for example, when an exception occurs in an **`__del__()`** method.

The function is called with a single argument *obj* that identifies the context in which the unraisable exception occurred. If possible, the repr of *obj* will be printed in the warning message. If *obj* is `NULL`, only the traceback is printed.

Se debe establecer una excepción al llamar a esta función.

Distinto en la versión 3.4: Print a traceback. Print only traceback if *obj* is `NULL`.

Distinto en la versión 3.8: Use **`sys.unraisablehook()`**.

void PyErr_FormatUnraisable(const char *format, ...)

Similar to **`PyErr_WriteUnraisable()`**, but the *format* and subsequent parameters help format the warning message; they have the same meaning and values as in **`PyUnicode_FromFormat()`**. `PyErr_WriteUnraisable(obj)` is roughly equivalent to `PyErr_FormatUnraisable("Exception ignored in: %R", obj)`. If *format* is `NULL`, only the traceback is printed.

Added in version 3.13.

void PyErr_DisplayException(PyObject *exc)

Part of the Stable ABI since version 3.12.
Print the standard traceback display of `exc` to `sys.stderr`, including chained exceptions and notes.

Added in version 3.12.

Lanzando excepciones

Estas funciones lo ayudan a configurar el indicador de error del hilo actual. Por conveniencia, algunas de estas funciones siempre retornarán un puntero `NULL` para usar en una declaración `return`.

void PyErr_SetString(PyObject *type, const char *message)

Part of the Stable ABI.
This is the most common way to set the error indicator. The first argument specifies the exception type; it is normally one of the standard exceptions, e.g. **`PyExc_RuntimeError`**. You need not create a new strong reference to it (e.g. with **`Py_INCREF()`**). The second argument is an error message; it is decoded from `'utf-8'`.

void PyErr_SetObject(PyObject *type, PyObject *value)

Part of the Stable ABI.
Esta función es similar a **`PyErr_SetString()`** pero le permite especificar un objeto Python arbitrario para el «valor» de la excepción.

PyObject *PyErr_Format(PyObject *exception, const char *format, ...)

Return value: Always NULL. Part of the Stable ABI.
Esta función establece el indicador de error y retorna `NULL`. *exception* debe ser una clase de excepción Python. El *format* y los parámetros posteriores ayudan a formatear el mensaje de error; tienen el mismo significado y valores que en **`PyUnicode_FromFormat()`**. *format* es una cadena de caracteres codificada en ASCII.

PyObject *PyErr_FormatV(PyObject *exception, const char *format, va_list vargs)

Return value: Always NULL. Part of the Stable ABI since version 3.5.
Igual que **`PyErr_Format()`**, pero tomando un argumento

`va_list` en lugar de un número variable de argumentos.

Added in version 3.5.

void PyErr_SetNone(PyObject *type)

Part of the Stable ABI.
Esta es una abreviatura de `PyErr_SetObject(type, Py_None)`.

int PyErr_BadArgument()

Part of the Stable ABI.
Esta es una abreviatura de `PyErr_SetString(PyExc_TypeError, message)`, donde *message* indica que se invocó una operación incorporada con un argumento ilegal. Es principalmente para uso interno.

PyObject *PyErr_NoMemory()

Return value: Always NULL. Part of the Stable ABI.
Esta es una abreviatura de `PyErr_SetNone(PyExc_MemoryError)`; retorna `NULL` para que una función de asignación de objetos pueda escribir `return PyErr_NoMemory();` cuando se queda sin memoria.

PyObject *PyErr_SetFromErrno(PyObject *type)

Return value: Always NULL. Part of the Stable ABI.
This is a convenience function to raise an exception when a C library function has returned an error and set the C variable **`errno`**. It constructs a tuple object whose first item is the integer **`errno`** value and whose second item is the corresponding error message (gotten from **`strerror()`**), and then calls `PyErr_SetObject(type, object)`. On Unix, when the **`errno`** value is **`EINTR`**, indicating an interrupted system call, this calls **`PyErr_CheckSignals()`**, and if that set the error indicator, leaves it set to that. The function always returns `NULL`, so a wrapper function around a system call can write `return`

`PyErr_SetFromErrno(type);` when the system call returns an error.

PyObject *PyErr_SetFromErrnoWithFilenameObject(PyObject *type, PyObject *filenameObject)

Return value: Always NULL. Part of the Stable ABI.
Similar to **`PyErr_SetFromErrno()`**, with the additional behavior that if *filenameObject* is not `NULL`, it is passed to the constructor of *type* as a third parameter. In the case of **`OSError`** exception, this is used to define the **`filename`** attribute of the exception instance.

PyObject *PyErr_SetFromErrnoWithFilenameObjects(PyObject *type, PyObject *filenameObject, PyObject *filenameObject2)

Return value: Always NULL. Part of the Stable ABI since version 3.7.
Similar a **`PyErr_SetFromErrnoWithFilenameObject()`**, pero toma un segundo objeto de nombre de archivo, para lanzar errores cuando falla una función que toma dos nombres de archivo.

Added in version 3.4.

PyObject *PyErr_SetFromErrnoWithFilename(PyObject *type, const char *filename)

Return value: Always NULL. Part of the Stable ABI.
Similar a **`PyErr_SetFromErrnoWithFilenameObject()`**, pero el nombre del archivo se da como una cadena de caracteres de C. *filename* se decodifica a partir de la codificación de filesystem encoding and error handler.

PyObject *PyErr_SetFromWindowsErr(int ierr)

Return value: Always NULL. Part of the Stable ABI on Windows since version 3.7.
This is a convenience function to raise **`OSError`**. If called

with *ierr* of `0`, the error code returned by a call to **`GetLastError()`** is used instead. It calls the Win32 function **`FormatMessage()`** to retrieve the Windows description of error code given by *ierr* or **`GetLastError()`**, then it constructs a **`OSError`** object with the **`winerror`** attribute set to the error code, the **`strerror`** attribute set to the corresponding error message (gotten from **`FormatMessage()`**), and then calls `PyErr_SetObject(PyExc_OSError, object)`. This function always returns `NULL`.

Availability: Windows.

PyObject *PyErr_SetExcFromWindowsErr(PyObject *type, int ierr)

Return value: Always NULL. Part of the Stable ABI on Windows since version 3.7.
Similar a **`PyErr_SetFromWindowsErr()`**, con un parámetro adicional que especifica el tipo de excepción que se lanzará.

Availability: Windows.

PyObject *PyErr_SetFromWindowsErrWithFilename(int ierr, const char *filename)

Return value: Always NULL. Part of the Stable ABI on Windows since version 3.7.
Similar to **`PyErr_SetFromWindowsErr()`**, with the additional behavior that if *filename* is not `NULL`, it is decoded from the filesystem encoding (**`os.fsdecode()`**) and passed to the constructor of **`OSError`** as a third parameter to be used to define the **`filename`** attribute of the exception instance.

Availability: Windows.

PyObject *PyErr_SetExcFromWindowsErrWithFilenameObject(PyObject *type, int ierr, PyObject *filename)

Return value: Always NULL. Part of the Stable ABI on Windows since version 3.7.
Similar to **`PyErr_SetExcFromWindowsErr()`**, with the additional behavior that if *filename* is not `NULL`, it is passed to the constructor of **`OSError`** as a third parameter to be used to define the **`filename`** attribute of the exception instance.

Availability: Windows.

PyObject *PyErr_SetExcFromWindowsErrWithFilenameObjects(PyObject *type, int ierr, PyObject *filename, PyObject *filename2)

Return value: Always NULL. Part of the Stable ABI on Windows since version 3.7.
Similar a **`PyErr_SetExcFromWindowsErrWithFilenameObject()`**, pero acepta un segundo objeto de nombre de archivo.

Availability: Windows.

Added in version 3.4.

PyObject *PyErr_SetExcFromWindowsErrWithFilename(PyObject *type, int ierr, const char *filename)

Return value: Always NULL. Part of the Stable ABI on Windows since version 3.7.
Similar a **`PyErr_SetFromWindowsErrWithFilename()`**, con un parámetro adicional que especifica el tipo de excepción que se lanzará.

Availability: Windows.

PyObject *PyErr_SetImportError(PyObject *msg, PyObject *name, PyObject *path)

Return value: Always NULL. Part of the Stable ABI since version 3.7.
Esta es una función conveniente para subir **`ImportError`**.

msg se establecerá como la cadena de mensaje de la excepción. *name* y *path*, que pueden ser `NULL`, se establecerán como atributos respectivos `name` y `path` de **`ImportError`**.

Added in version 3.3.

PyObject *PyErr_SetImportErrorSubclass(PyObject *exception, PyObject *msg, PyObject *name, PyObject *path)

Return value: Always NULL. Part of the Stable ABI since version 3.6.
Al igual que **`PyErr_SetImportError()`** pero esta función permite especificar una subclase de **`ImportError`** para aumentar.

Added in version 3.6.

void PyErr_SyntaxLocationObject(PyObject *filename, int lineno, int col_offset)

Establece información de archivo, línea y desplazamiento para la excepción actual. Si la excepción actual no es un **`SyntaxError`**, establece atributos adicionales, lo que hace que el sub sistema de impresión de excepciones piense que la excepción es **`SyntaxError`**.

Added in version 3.4.

void PyErr_RangedSyntaxLocationObject(PyObject *filename, int lineno, int col_offset, int end_lineno, int end_col_offset)

Similar to **`PyErr_SyntaxLocationObject()`**, but also sets the *end_lineno* and *end_col_offset* information for the current exception.

Added in version 3.10.

void PyErr_SyntaxLocationEx(const char *filename, int lineno, int col_offset)

Part of the Stable ABI since version 3.7.

Como `PyErr_SyntaxLocationObject()`, pero *filename* es una cadena de bytes decodificada a partir de filesystem encoding and error handler.

Added in version 3.2.

void PyErr_SyntaxLocation(const char *filename, int lineno)

Part of the Stable ABI.
Como `PyErr_SyntaxLocationEx()`, pero se omite el parámetro *col_offset*.

void PyErr_BadInternalCall()

Part of the Stable ABI.
Esta es una abreviatura de `PyErr_SetString(PyExc_SystemError, message)`, donde *message* indica que se invocó una operación interna (por ejemplo, una función de Python/C API) con un argumento ilegal. Es principalmente para uso interno.

PyObject *PyErr_ProgramTextObject(PyObject *filename, int lineno)

Get the source line in *filename* at line *lineno*. *filename* should be a Python `str` object.

On success, this function returns a Python string object with the found line. On failure, this function returns `NULL` without an exception set.

PyObject *PyErr_ProgramText(const char *filename, int lineno)

Part of the Stable ABI.
Similar to `PyErr_ProgramTextObject()`, but *filename* is a const char*, which is decoded with the filesystem encoding and error handler, instead of a Python object reference.

Emitir advertencias

Use estas funciones para emitir advertencias desde el código C.

Reflejan funciones similares exportadas por el módulo Python **`warnings`**. Normalmente imprimen un mensaje de advertencia a *sys.stderr*; sin embargo, también es posible que el usuario haya especificado que las advertencias se conviertan en errores, y en ese caso lanzarán una excepción. También es posible que las funciones generen una excepción debido a un problema con la maquinaria de advertencia. El valor de retorno es `0` si no se lanza una excepción, o `-1` si se lanza una excepción. (No es posible determinar si realmente se imprime un mensaje de advertencia, ni cuál es el motivo de la excepción; esto es intencional). Si se produce una excepción, la persona que llama debe hacer su manejo normal de excepciones (por ejemplo, referencias propiedad de **`Py_DECREF()`** y retornan un valor de error).

int PyErr_WarnEx(PyObject *category, const char *message, Py_ssize_t stack_level)

Part of the Stable ABI.
Emite un mensaje de advertencia. El argumento *category* es una categoría de advertencia (ver más abajo) o `NULL`; el argumento *message* es una cadena de caracteres codificada en UTF-8. *stack_level* es un número positivo que proporciona una cantidad de marcos de pila; la advertencia se emitirá desde la línea de código que se está ejecutando actualmente en ese marco de pila. Un *stack_level* de 1 es la función que llama **`PyErr_WarnEx()`**, 2 es la función por encima de eso, y así sucesivamente.

Las categorías de advertencia deben ser subclases de **`PyExc_Warning`**; **`PyExc_Warning`** es una subclase de **`PyExc_Exception`**; la categoría de advertencia predeterminada es **`PyExc_RuntimeWarning`**. Las categorías de advertencia estándar de Python están disponibles como variables globales cuyos nombres se enumeran en Warning types.

Para obtener información sobre el control de advertencia, consulte la documentación del módulo **`warnings`** y la opción **`-W`** en la documentación de la línea de comandos. No hay API de C para el control de advertencia.

int PyErr_WarnExplicitObject(PyObject *category, PyObject *message, PyObject *filename, int lineno, PyObject *module, PyObject *registry)

Emite un mensaje de advertencia con control explícito sobre todos los atributos de advertencia. Este es un contenedor sencillo alrededor de la función Python **`warnings.warn_explicit()`**; consulte allí para obtener más información. Los argumentos *module* y *registry* pueden establecerse en `NULL` para obtener el efecto predeterminado que se describe allí.

Added in version 3.4.

int PyErr_WarnExplicit(PyObject *category, const char *message, const char *filename, int lineno, const char *module, PyObject *registry)

Part of the Stable ABI.
Similar a **`PyErr_WarnExplicitObject()`** excepto que *message* y *module* son cadenas codificadas UTF-8, y *filename* se decodifica de filesystem encoding and error handler.

int PyErr_WarnFormat(PyObject *category, Py_ssize_t stack_level, const char *format, ...)

Part of the Stable ABI.
Función similar a **`PyErr_WarnEx()`**, pero usa **`PyUnicode_FromFormat()`** para formatear el mensaje de advertencia. *format* es una cadena de caracteres codificada en ASCII.

Added in version 3.2.

int PyErr_WarnExplicitFormat(PyObject *category, const char *filename, int lineno, const char *module, PyObject *registry, const char *format, ...)

Similar to **`PyErr_WarnExplicit()`**, but uses **`PyUnicode_FromFormat()`** to format the warning message. *format* is an ASCII-encoded string.

Added in version 3.2.

int PyErr_ResourceWarning(PyObject *source, Py_ssize_t stack_level, const char *format, ...)

Part of the Stable ABI since version 3.6.
Function similar to **`PyErr_WarnFormat()`**, but *category* is **`ResourceWarning`** and it passes *source* to **`warnings.WarningMessage`**.

Added in version 3.6.

Consultando el indicador de error

PyObject *PyErr_Occurred()

Return value: Borrowed reference. Part of the Stable ABI.
Prueba si el indicador de error está configurado. Si se establece, retorna la excepción *type* (el primer argumento de la última llamada a una de las funciones `PyErr_Set*` o **`PyErr_Restore()`**). Si no está configurado, retorna `NULL`. No posee una referencia al valor de retorno, por lo que no necesita usar **`Py_DECREF()`**.

The caller must have an attached thread state.

Nota

No compare el valor de retorno con una excepción específica; use **`PyErr_ExceptionMatches()`** en su lugar, como se muestra a continuación. (La comparación podría fallar fácilmente ya que la excepción puede ser una instancia en lugar de una clase, en el caso de una excepción de clase, o puede ser una subclase de la excepción esperada).

int PyErr_ExceptionMatches(PyObject *exc)

Part of the Stable ABI.
Equivalente a

`PyErr_GivenExceptionMatches(PyErr_Occurred(), exc)`. Esto solo debería llamarse cuando se establece una excepción; se producirá una infracción de acceso a la memoria si no se ha producido ninguna excepción.

int PyErr_GivenExceptionMatches(PyObject *given, PyObject *exc)

Part of the Stable ABI.
Retorna verdadero si la excepción *dada* coincide con el tipo de excepción en *exc*. Si *exc* es un objeto de clase, esto también retorna verdadero cuando *dado* es una instancia de una subclase. Si *exc* es una tupla, se busca una coincidencia en todos los tipos de excepción en la tupla (y recursivamente en sub tuplas).

PyObject *PyErr_GetRaisedException(void)

Return value: New reference. Part of the Stable ABI since version 3.12.
Return the exception currently being raised, clearing the error indicator at the same time. Return `NULL` if the error indicator is not set.

This function is used by code that needs to catch exceptions, or code that needs to save and restore the error indicator temporarily.

For example:

```
{
   PyObject *exc = PyErr_GetRaisedException();

   /* ... code that might produce other errors ...

   PyErr_SetRaisedException(exc);
}
```

Ver también

`PyErr_GetHandledException()`, to save the exception currently being handled.

Added in version 3.12.

void PyErr_SetRaisedException(PyObject *exc)

Part of the Stable ABI since version 3.12.
Set *exc* as the exception currently being raised, clearing the existing exception if one is set.

Advertencia

This call steals a reference to *exc*, which must be a valid exception.

Added in version 3.12.

void PyErr_Fetch(PyObject **ptype, PyObject **pvalue, PyObject **ptraceback)

Part of the Stable ABI.
Obsoleto desde la versión 3.12: Use **`PyErr_GetRaisedException()`** instead.

Recupera el indicador de error en tres variables cuyas direcciones se pasan. Si el indicador de error no está configurado, configure las tres variables en `NULL`. Si está configurado, se borrará y usted tendrá una referencia a cada objeto recuperado. El objeto de valor y rastreo puede ser `NULL` incluso cuando el objeto de tipo no lo es.

Nota

This function is normally only used by legacy code that needs to catch exceptions or save and restore the error indicator temporarily.

For example:

```
{
   PyObject *type, *value, *traceback;
   PyErr_Fetch(&type, &value, &traceback);
```

```
        /* ... code that might produce other errors ...

        PyErr_Restore(type, value, traceback);
    }
```

void PyErr_Restore(PyObject *type, PyObject *value, PyObject *traceback)

Part of the Stable ABI.
Obsoleto desde la versión 3.12: Use **`PyErr_SetRaisedException()`** instead.

Set the error indicator from the three objects, *type*, *value*, and *traceback*, clearing the existing exception if one is set. If the objects are `NULL`, the error indicator is cleared. Do not pass a `NULL` type and non-`NULL` value or traceback. The exception type should be a class. Do not pass an invalid exception type or value. (Violating these rules will cause subtle problems later.) This call takes away a reference to each object: you must own a reference to each object before the call and after the call you no longer own these references. (If you don't understand this, don't use this function. I warned you.)

Nota

This function is normally only used by legacy code that needs to save and restore the error indicator temporarily. Use **`PyErr_Fetch()`** to save the current error indicator.

void PyErr_NormalizeException(PyObject **exc, PyObject **val, PyObject **tb)

Part of the Stable ABI.
Obsoleto desde la versión 3.12: Use **`PyErr_GetRaisedException()`** instead, to avoid any possible de-normalization.

Bajo ciertas circunstancias, los valores retornados por

`PyErr_Fetch()` a continuación pueden ser «no normalizados», lo que significa que `*exc` es un objeto de clase pero `*val` no es una instancia de la misma clase . Esta función se puede utilizar para crear instancias de la clase en ese caso. Si los valores ya están normalizados, no pasa nada. La normalización retrasada se implementa para mejorar el rendimiento.

Nota

This function *does not* implicitly set the `__traceback__` attribute on the exception value. If setting the traceback appropriately is desired, the following additional snippet is needed:

```
if (tb != NULL) {
  PyException_SetTraceback(val, tb);
}
```

PyObject *PyErr_GetHandledException(void)

Part of the Stable ABI since version 3.11.
Recupera la instancia de excepción activa, como la que devolvería `sys.exception()`. Esto se refiere a una excepción que *ya fue capturada*, no a una excepción recién lanzada. Retorna una nueva referencia a la excepción o `NULL`. No modifica el estado de excepción del intérprete.

Nota

Esta función normalmente no es utilizada por el código que quiere manejar excepciones. En cambio, se puede usar cuando el código necesita guardar y restaurar el estado de excepción temporalmente. Use `PyErr_SetHandledException()` para restaurar o borrar el estado de excepción.

Added in version 3.11.

void PyErr_SetHandledException(PyObject *exc)

Part of the Stable ABI since version 3.11.
Establece la excepción activa, como se conoce de `sys.exception()`. Esto se refiere a la excepción que **ya fue capturada**, no a una excepción que fue lanzada recientemente. Para borrar el estado de la excepción, pasa `NULL`.

Nota

Esta función normalmente no es utilizada por el código que quiere manejar excepciones. En cambio, se puede usar cuando el código necesita guardar y restaurar el estado de excepción temporalmente. Use **`PyErr_GetHandledException()`** para leer el estado de excepción.

Added in version 3.11.

void PyErr_GetExcInfo(PyObject **ptype, PyObject **pvalue, PyObject **ptraceback)

Part of the Stable ABI since version 3.7.
Recupera la información de excepción, como se conoce de **`sys.exc_info()`**. Esto se refiere a una excepción que *ya fue capturada*, no a una excepción que fue lanzada recientemente. Retorna nuevas referencias para los tres objetos, cualquiera de los cuales puede ser `NULL`. No modifica el estado de información de excepción. Esta función se mantiene por retro-compatibilidad. Es preferible usar **`PyErr_GetHandledException()`**.

Nota

Esta función normalmente no es utilizada por el código que quiere manejar excepciones. En cambio, se puede usar cuando el código necesita guardar y restaurar el estado de excepción temporalmente. Use **`PyErr_SetExcInfo()`** para restaurar o borrar el estado de excepción.

Added in version 3.3.

void PyErr_SetExcInfo(PyObject *type, PyObject *value, PyObject *traceback)

Part of the Stable ABI since version 3.7.
Establece la información de excepción, como se conoce de `sys.exc_info()`. Esto se refiere a una excepción que *ya fue capturada*, no a una excepción que fue lanzada recientemente. Esta función roba las referencias de los argumentos. Para borrar el estado de excepción, pase `NULL` para los tres argumentos. Para ver las reglas generales sobre los tres argumentos, consulte **`PyErr_SetHandledException()`**.

Nota

Esta función normalmente no es utilizada por el código que quiere manejar excepciones. En cambio, se puede usar cuando el código necesita guardar y restaurar el estado de excepción temporalmente. Use **`PyErr_GetExcInfo()`** para leer el estado de excepción.

Added in version 3.3.

Distinto en la versión 3.11: Los argumentos `type` y `traceback` ya no se utilizan y pueden ser NULL. El intérprete los deriva ahora de la instancia de la excepción (el argumento `value`). La función sigue robando referencias de los tres argumentos.

Manejo de señal

int PyErr_CheckSignals()

Part of the Stable ABI.
Handle external interruptions, such as signals or activating a debugger, whose processing has been delayed until it is safe to run Python code and/or raise exceptions.

For example, pressing `Ctrl-C` causes a terminal to send the

`signal.SIGINT` signal. This function executes the corresponding Python signal handler, which, by default, raises the `KeyboardInterrupt` exception.

`PyErr_CheckSignals()` should be called by long-running C code frequently enough so that the response appears immediate to humans.

Handlers invoked by this function currently include:

- Signal handlers, including Python functions registered using the `signal` module.

  Signal handlers are only run in the main thread of the main interpreter.

  (This is where the function got the name: originally, signals were the only way to interrupt the interpreter.)
- Running the garbage collector, if necessary.
- Executing a pending remote debugger script.

If any handler raises an exception, immediately return `-1` with that exception set. Any remaining interruptions are left to be processed on the next `PyErr_CheckSignals()` invocation, if appropriate.

If all handlers finish successfully, or there are no handlers to run, return `0`.

Distinto en la versión 3.12: This function may now invoke the garbage collector.

Distinto en la versión 3.14: This function may now execute a remote debugger script, if remote debugging is enabled.

void PyErr_SetInterrupt()

Part of the Stable ABI.
Simulate the effect of a **`SIGINT`** signal arriving. This is equivalent to `PyErr_SetInterruptEx(SIGINT)`.

Nota

This function is async-signal-safe. It can be called without an attached thread state and from a C signal handler.

int PyErr_SetInterruptEx(int signum)

Part of the Stable ABI since version 3.10.
Simula el efecto de la llegada de una señal. La próxima vez que sea llamado **`PyErr_CheckSignals()`**, se llamará al manejador de señal de Python para el número de señal dado.

Esta función puede ser llamada por código C que configura su propio manejo de señales y quiere que los manejadores de señales de Python sean invocados como se espera cuando se solicita una interrupción (por ejemplo, cuando el usuario presiona Ctrl-C para interrumpir una operación).

If the given signal isn't handled by Python (it was set to **`signal.SIG_DFL`** or **`signal.SIG_IGN`**), it will be ignored.

Si *signum* está fuera del rango permitido de números de señal, se devuelve `-1`. De lo contrario, se devuelve `0`. Esta función nunca cambia el indicador de error.

Nota

This function is async-signal-safe. It can be called without an attached thread state and from a C signal handler.

Added in version 3.10.

int PySignal_SetWakeupFd(int fd)

Esta función de utilidad especifica un descriptor de archivo en el que el número de señal se escribe como un solo byte cada vez que se recibe una señal. *fd* debe ser sin bloqueo. retorna el descriptor de archivo anterior.

El valor `-1` desactiva la función; Este es el estado inicial.

Esto es equivalente a **`signal.set_wakeup_fd()`** en Python, pero sin verificación de errores. *fd* debe ser un descriptor de archivo válido. La función solo debe llamarse desde el hilo principal.

Distinto en la versión 3.5: En Windows, la función ahora también admite controladores de socket.

Clases de excepción

PyObject *PyErr_NewException(const char *name, PyObject *base, PyObject *dict)

Return value: New reference. Part of the Stable ABI.
Esta función de utilidad crea y retorna una nueva clase de excepción. El argumento *name* debe ser el nombre de la nueva excepción, una cadena de caracteres en C de la forma `module.classname`. Los argumentos *base* y *dict* son normalmente `NULL`. Esto crea un objeto de clase derivado de **`Exception`** (accesible en C como **`PyExc_Exception`**).

The **`__module__`** attribute of the new class is set to the first part (up to the last dot) of the *name* argument, and the class name is set to the last part (after the last dot). The *base* argument can be used to specify alternate base classes; it can either be only one class or a tuple of classes. The *dict* argument can be used to specify a dictionary of class variables and methods.

PyObject *PyErr_NewExceptionWithDoc(const char *name, const char *doc, PyObject *base, PyObject *dict)

Return value: New reference. Part of the Stable ABI.
Igual que **`PyErr_NewException()`**, excepto que la nueva clase de excepción puede recibir fácilmente una cadena de documentación: si *doc* no es `NULL`, se utilizará como la cadena de documentación para la clase de excepción.

Added in version 3.2.

int PyExceptionClass_Check(PyObject *ob)

Return non-zero if *ob* is an exception class, zero otherwise. This function always succeeds.

const char *PyExceptionClass_Name(PyObject *ob)

Part of the Stable ABI since version 3.8.
Return **`tp_name`** of the exception class *ob*.

Objetos excepción

int PyExceptionInstance_Check(PyObject *op)

Return true if *op* is an instance of **`BaseException`**, false otherwise. This function always succeeds.

PyExceptionInstance_Class(op)

Equivalent to **`Py_TYPE(op)`**.

PyObject *PyException_GetTraceback(PyObject *ex)

Return value: New reference. Part of the Stable ABI.
Return the traceback associated with the exception as a new reference, as accessible from Python through the **`__traceback__`** attribute. If there is no traceback associated, this returns `NULL`.

int PyException_SetTraceback(PyObject *ex, PyObject *tb)

Part of the Stable ABI.
Establezca el rastreo asociado con la excepción a *tb*. Use `Py_None` para borrarlo.

PyObject *PyException_GetContext(PyObject *ex)

Return value: New reference. Part of the Stable ABI.
Return the context (another exception instance during whose handling *ex* was raised) associated with the exception as a new reference, as accessible from Python through the **`__context__`** attribute. If there is no context associated, this returns `NULL`.

void PyException_SetContext(PyObject *ex, PyObject *ctx)

Part of the Stable ABI.
Establece el contexto asociado con la excepción a *ctx*. Use `NULL` para borrarlo. No hay verificación de tipo para asegurarse de que *ctx* es una instancia de excepción. Esto roba una referencia a *ctx*.

PyObject *PyException_GetCause(PyObject *ex)

Return value: New reference. Part of the Stable ABI.
Return the cause (either an exception instance, or `None`, set by `raise ... from ...`) associated with the exception as a new reference, as accessible from Python through the **`__cause__`** attribute.

void PyException_SetCause(PyObject *ex, PyObject *cause)

Part of the Stable ABI.
Set the cause associated with the exception to *cause*. Use `NULL` to clear it. There is no type check to make sure that *cause* is either an exception instance or `None`. This steals a reference to *cause*.

The **`__suppress_context__`** attribute is implicitly set to `True` by this function.

PyObject *PyException_GetArgs(PyObject *ex)

Return value: New reference. Part of the Stable ABI since version 3.12.
Return **`args`** of exception *ex*.

void PyException_SetArgs(PyObject *ex, PyObject *args)

Part of the Stable ABI since version 3.12.
Set **`args`** of exception *ex* to *args*.

PyObject *PyUnstable_Exc_PrepReraiseStar(PyObject *orig, PyObject *excs)

This is Unstable API. It may change without warning in minor releases.

Implement part of the interpreter's implementation of **`except*`**. *orig* is the original exception that was caught, and *excs* is the list of the exceptions that need to be raised. This list contains the unhandled part of *orig*, if any, as well as the exceptions that were raised from the **`except*`** clauses (so they have a different traceback from *orig*) and those that were reraised (and have the same traceback as *orig*). Return the **`ExceptionGroup`** that needs to be reraised in the end, or `None` if there is nothing to reraise.

Added in version 3.12.

Objetos unicode de excepción

Las siguientes funciones se utilizan para crear y modificar excepciones Unicode de C.

PyObject *PyUnicodeDecodeError_Create(const char *encoding, const char *object, Py_ssize_t length, Py_ssize_t start, Py_ssize_t end, const char *reason)

Return value: New reference. Part of the Stable ABI.
Crea un objeto **`UnicodeDecodeError`** con los atributos *encoding*, *object*, *length*, *start*, *end* y *reason*. *encoding* y *reason* son cadenas codificadas UTF-8.

PyObject *PyUnicodeDecodeError_GetEncoding(PyObject *exc)
PyObject *PyUnicodeEncodeError_GetEncoding(PyObject *exc)

Return value: New reference. Part of the Stable ABI.
Retorna el atributo *encoding* del objeto de excepción dado.

PyObject *PyUnicodeDecodeError_GetObject(PyObject *exc)
PyObject *PyUnicodeEncodeError_GetObject(PyObject *exc)
PyObject *PyUnicodeTranslateError_GetObject(PyObject *exc)

Return value: New reference. Part of the Stable ABI.

Retorna el atributo *object* del objeto de excepción dado.

int PyUnicodeDecodeError_GetStart(PyObject *exc, Py_ssize_t *start)
int PyUnicodeEncodeError_GetStart(PyObject *exc, Py_ssize_t *start)
int PyUnicodeTranslateError_GetStart(PyObject *exc, Py_ssize_t *start)

Part of the Stable ABI.
Obtiene el atributo *start* del objeto de excepción dado y lo coloca en **start*. *start* no debe ser `NULL`. retorna `0` en caso de éxito, `-1` en caso de error.

If the **`UnicodeError.object`** is an empty sequence, the resulting *start* is `0`. Otherwise, it is clipped to `[0, len(object) - 1]`.

Ver también

`UnicodeError.start`

int PyUnicodeDecodeError_SetStart(PyObject *exc, Py_ssize_t start)
int PyUnicodeEncodeError_SetStart(PyObject *exc, Py_ssize_t start)
int PyUnicodeTranslateError_SetStart(PyObject *exc, Py_ssize_t start)

Part of the Stable ABI.
Set the *start* attribute of the given exception object to *start*. Return `0` on success, `-1` on failure.

Nota

While passing a negative *start* does not raise an exception, the corresponding getters will not consider it as a relative offset.

int PyUnicodeDecodeError_GetEnd(PyObject *exc, Py_ssize_t *end)
int PyUnicodeEncodeError_GetEnd(PyObject *exc, Py_ssize_t *end)
int PyUnicodeTranslateError_GetEnd(PyObject *exc, Py_ssize_t *end)

Part of the Stable ABI.
Obtiene el atributo *end* del objeto de excepción dado y lo coloca en **end*. *end* no debe ser `NULL`. retorna `0` en caso de éxito, `-1` en caso de error.

If the **`UnicodeError.object`** is an empty sequence, the resulting *end* is `0`. Otherwise, it is clipped to `[1, len(object)]`.

int PyUnicodeDecodeError_SetEnd(PyObject *exc, Py_ssize_t end)
int PyUnicodeEncodeError_SetEnd(PyObject *exc, Py_ssize_t end)
int PyUnicodeTranslateError_SetEnd(PyObject *exc, Py_ssize_t end)

Part of the Stable ABI.
Establece el atributo *end* del objeto de excepción dado en *end*. Retorna `0` en caso de éxito, `-1` en caso de error.

Ver también

`UnicodeError.end`

PyObject *PyUnicodeDecodeError_GetReason(PyObject *exc)
PyObject *PyUnicodeEncodeError_GetReason(PyObject *exc)
PyObject *PyUnicodeTranslateError_GetReason(PyObject *exc)

Return value: New reference. Part of the Stable ABI.
Retorna el atributo *reason* del objeto de excepción dado.

int PyUnicodeDecodeError_SetReason(PyObject *exc, const char *reason)
int PyUnicodeEncodeError_SetReason(PyObject *exc, const char *reason)
int PyUnicodeTranslateError_SetReason(PyObject *exc, const char

*reason)

Part of the Stable ABI.
Establece el atributo *reason* del objeto de excepción dado en *reason*. Retorna `0` en caso de éxito, `-1` en caso de error.

Control de recursión

Estas dos funciones proporcionan una forma de realizar llamadas recursivas seguras en el nivel C, tanto en el núcleo como en los módulos de extensión. Son necesarios si el código recursivo no invoca necesariamente el código Python (que rastrea su profundidad de recursión automáticamente). Tampoco son necesarios para las implementaciones de *tp_call* porque call protocol se encarga del manejo de la recursividad.

int Py_EnterRecursiveCall(const char *where)

Part of the Stable ABI since version 3.9.
Marca un punto donde una llamada recursiva de nivel C está a punto de realizarse.

The function then checks if the stack limit is reached. If this is the case, a **`RecursionError`** is set and a nonzero value is returned. Otherwise, zero is returned.

where debería ser una cadena de caracteres codificada en UTF-8 como `"en la comprobación de instancia"` para concatenarse con el mensaje **`RecursionError`** causado por el límite de profundidad de recursión.

Ver también

The **`PyUnstable_ThreadState_SetStackProtection()`** function.

Distinto en la versión 3.9: This function is now also available in the limited API.

void Py_LeaveRecursiveCall(void)

Part of the Stable ABI since version 3.9.
Termina una `Py_EnterRecursiveCall()`. Se debe llamar una vez por cada invocación *exitosa* de `Py_EnterRecursiveCall()`.

Distinto en la versión 3.9: This function is now also available in the limited API.

La implementación adecuada de `tp_repr` para los tipos de contenedor requiere un manejo de recursión especial. Además de proteger la pila, `tp_repr` también necesita rastrear objetos para evitar ciclos. Las siguientes dos funciones facilitan esta funcionalidad. Efectivamente, estos son los C equivalentes a `reprlib.recursive_repr()`.

int Py_ReprEnter(PyObject *object)

Part of the Stable ABI.
Llamado al comienzo de la implementación `tp_repr` para detectar ciclos.

Si el objeto ya ha sido procesado, la función retorna un entero positivo. En ese caso, la implementación `tp_repr` debería retornar un objeto de cadena que indique un ciclo. Como ejemplos, los objetos `dict` retornan `{...}` y los objetos `list` retornan `[...]`.

La función retornará un entero negativo si se alcanza el límite de recursión. En ese caso, la implementación `tp_repr` normalmente debería retornar `NULL`.

De lo contrario, la función retorna cero y la implementación `tp_repr` puede continuar normalmente.

void Py_ReprLeave(PyObject *object)

Part of the Stable ABI.
Termina a `Py_ReprEnter()`. Se debe llamar una vez por cada invocación de `Py_ReprEnter()` que retorna cero.

int Py_GetRecursionLimit(void)

Part of the Stable ABI.
Get the recursion limit for the current interpreter. It can be set with `Py_SetRecursionLimit()`. The recursion limit prevents the Python interpreter stack from growing infinitely.

This function cannot fail, and the caller must hold an attached thread state.

Ver también

`sys.getrecursionlimit()`

void Py_SetRecursionLimit(int new_limit)

Part of the Stable ABI.
Set the recursion limit for the current interpreter.

This function cannot fail, and the caller must hold an attached thread state.

Ver también

`sys.setrecursionlimit()`

Exception and warning types

All standard Python exceptions and warning categories are available as global variables whose names are `PyExc_` followed by the Python exception name. These have the type PyObject*; they are all class objects.

For completeness, here are all the variables:

Exception types

C name	Python name

PyObject *PyExc_BaseException

Part of the Stable ABI.

PyObject *PyExc_BaseExceptionGroup

Part of the Stable ABI since version 3.11.

PyObject *PyExc_Exception

Part of the Stable ABI.

PyObject *PyExc_ArithmeticError

Part of the Stable ABI.

PyObject *PyExc_AssertionError

Part of the Stable ABI.

PyObject *PyExc_AttributeError

Part of the Stable ABI.

PyObject *PyExc_BlockingIOError

Part of the Stable ABI since version 3.7.

PyObject *PyExc_BrokenPipeError

Part of the Stable ABI since version 3.7.

PyObject *PyExc_BufferError

Part of the Stable ABI.

PyObject *PyExc_ChildProcessError

Part of the Stable ABI since version 3.7.

PyObject *PyExc_ConnectionAbortedError

Part of the Stable ABI since version 3.7.

PyObject *PyExc_ConnectionError

Part of the Stable ABI since version 3.7.

PyObject *PyExc_ConnectionRefusedError

Part of the Stable ABI since version 3.7.

PyObject *PyExc_ConnectionResetError

Part of the Stable ABI since version 3.7.

PyObject *PyExc_EOFError

Part of the Stable ABI.

PyObject *PyExc_FileExistsError

Part of the Stable ABI since version 3.7.

PyObject *PyExc_FileNotFoundError

Part of the Stable ABI since version 3.7.

PyObject *PyExc_FloatingPointError

Part of the Stable ABI.

PyObject *PyExc_GeneratorExit

Part of the Stable ABI.

PyObject *PyExc_ImportError

Part of the Stable ABI.

PyObject *PyExc_IndentationError

Part of the Stable ABI.

PyObject *PyExc_IndexError

Part of the Stable ABI.

PyObject *PyExc_InterruptedError

Part of the Stable ABI since version 3.7.

PyObject *PyExc_IsADirectoryError

Part of the Stable ABI since version 3.7.

PyObject *PyExc_KeyError

Part of the Stable ABI.

PyObject *PyExc_KeyboardInterrupt

Part of the Stable ABI.

PyObject *PyExc_LookupError

Part of the Stable ABI.

PyObject *PyExc_MemoryError

Part of the Stable ABI.

PyObject *PyExc_ModuleNotFoundError

Part of the Stable ABI since version 3.6.

PyObject *PyExc_NameError

Part of the Stable ABI.

PyObject *PyExc_NotADirectoryError

Part of the Stable ABI since version 3.7.

PyObject *PyExc_NotImplementedError

Part of the Stable ABI.

PyObject *PyExc_OSError

Part of the Stable ABI.

PyObject *PyExc_OverflowError

Part of the Stable ABI.

PyObject *PyExc_PermissionError

Part of the Stable ABI since version 3.7.

PyObject *PyExc_ProcessLookupError

Part of the Stable ABI since version 3.7.

PyObject *PyExc_PythonFinalizationError	**PythonFinalizationError**
PyObject *PyExc_RecursionError *Part of the Stable ABI since version 3.7.*	**RecursionError**
PyObject *PyExc_ReferenceError *Part of the Stable ABI.*	**ReferenceError**
PyObject *PyExc_RuntimeError *Part of the Stable ABI.*	**RuntimeError**
PyObject *PyExc_StopAsyncIteration *Part of the Stable ABI since version 3.7.*	**StopAsyncIteration**
PyObject *PyExc_StopIteration *Part of the Stable ABI.*	**StopIteration**
PyObject *PyExc_SyntaxError *Part of the Stable ABI.*	**SyntaxError**
PyObject *PyExc_SystemError *Part of the Stable ABI.*	**SystemError**
PyObject *PyExc_SystemExit *Part of the Stable ABI.*	**SystemExit**
PyObject *PyExc_TabError *Part of the Stable ABI.*	**TabError**
PyObject *PyExc_TimeoutError *Part of the Stable ABI since version 3.7.*	**TimeoutError**
PyObject *PyExc_TypeError *Part of the Stable ABI.*	**TypeError**
PyObject *PyExc_UnboundLocalError *Part of the Stable ABI.*	**UnboundLocalError**
PyObject *PyExc_UnicodeDecodeError *Part of the Stable ABI.*	**UnicodeDecodeError**
PyObject *PyExc_UnicodeEncodeError *Part of the Stable ABI.*	**UnicodeEncodeError**
PyObject *PyExc_UnicodeError *Part of the Stable ABI.*	**UnicodeError**
PyObject *PyExc_UnicodeTranslateError *Part of the Stable ABI.*	**UnicodeTranslateError**
PyObject *PyExc_ValueError *Part of the Stable ABI.*	**ValueError**
PyObject *PyExc_ZeroDivisionError	**ZeroDivisionError**

Part of the Stable ABI.

Added in version 3.3: **`PyExc_BlockingIOError`**, **`PyExc_BrokenPipeError`**, **`PyExc_ChildProcessError`**, **`PyExc_ConnectionError`**, **`PyExc_ConnectionAbortedError`**, **`PyExc_ConnectionRefusedError`**, **`PyExc_ConnectionResetError`**, **`PyExc_FileExistsError`**, **`PyExc_FileNotFoundError`**, **`PyExc_InterruptedError`**, **`PyExc_IsADirectoryError`**, **`PyExc_NotADirectoryError`**, **`PyExc_PermissionError`**, **`PyExc_ProcessLookupError`** y **`PyExc_TimeoutError`** fueron introducidos luego de **PEP 3151** [https://peps.python.org/pep-3151/].

Added in version 3.5: **`PyExc_StopAsyncIteration`** y **`PyExc_RecursionError`**.

Added in version 3.6: **`PyExc_ModuleNotFoundError`**.

Added in version 3.11: **`PyExc_BaseExceptionGroup`**.

OSError aliases

The following are a compatibility aliases to **`PyExc_OSError`**.

Distinto en la versión 3.3: Estos alias solían ser tipos de excepción separados.

C name	Python name	Notes
PyObject *PyExc_EnvironmentError *Part of the Stable ABI.*	OSError	
PyObject *PyExc_IOError *Part of the Stable ABI.*	OSError	
PyObject *PyExc_WindowsError *Part of the Stable ABI on Windows since version 3.7.*	OSError	[win]

Notas:

[win]

`PyExc_WindowsError` is only defined on Windows; protect code

that uses this by testing that the preprocessor macro `MS_WINDOWS` is defined.

Warning types

C name	Python name
PyObject *PyExc_Warning *Part of the Stable ABI.*	`Warning`
PyObject *PyExc_BytesWarning *Part of the Stable ABI.*	`BytesWarning`
PyObject *PyExc_DeprecationWarning *Part of the Stable ABI.*	`DeprecationWarning`
PyObject *PyExc_EncodingWarning *Part of the Stable ABI since version 3.10.*	`EncodingWarning`
PyObject *PyExc_FutureWarning *Part of the Stable ABI.*	`FutureWarning`
PyObject *PyExc_ImportWarning *Part of the Stable ABI.*	`ImportWarning`
PyObject *PyExc_PendingDeprecationWarning *Part of the Stable ABI.*	`PendingDeprecationWarning`
PyObject *PyExc_ResourceWarning *Part of the Stable ABI since version 3.7.*	`ResourceWarning`
PyObject *PyExc_RuntimeWarning *Part of the Stable ABI.*	`RuntimeWarning`
PyObject *PyExc_SyntaxWarning *Part of the Stable ABI.*	`SyntaxWarning`
PyObject *PyExc_UnicodeWarning *Part of the Stable ABI.*	`UnicodeWarning`
PyObject *PyExc_UserWarning *Part of the Stable ABI.*	`UserWarning`

Added in version 3.2: `PyExc_ResourceWarning`.

Added in version 3.10: `PyExc_EncodingWarning`.

Tracebacks

PyTypeObject PyTraceBack_Type

Part of the Stable ABI.
Type object for traceback objects. This is available as **`types.TracebackType`** in the Python layer.

int PyTraceBack_Check(PyObject *op)

Return true if *op* is a traceback object, false otherwise. This function does not account for subtypes.

int PyTraceBack_Here(PyFrameObject *f)

Part of the Stable ABI.
Replace the **`__traceback__`** attribute on the current exception with a new traceback prepending *f* to the existing chain.

Calling this function without an exception set is undefined behavior.

This function returns `0` on success, and returns `-1` with an exception set on failure.

int PyTraceBack_Print(PyObject *tb, PyObject *f)

Part of the Stable ABI.
Write the traceback *tb* into the file *f*.

This function returns `0` on success, and returns `-1` with an exception set on failure.

[1]

Defining extension modules

A C extension for CPython is a shared library (for example, a `.so` file on Linux, `.pyd` DLL on Windows), which is loadable into the Python process (for example, it is compiled with compatible compiler settings), and which exports an initialization function.

To be importable by default (that is, by **`importlib.machinery.ExtensionFileLoader`**), the shared library must be available on **`sys.path`**, and must be named after the module name plus an extension listed in **`importlib.machinery.EXTENSION_SUFFIXES`**.

Nota

Building, packaging and distributing extension modules is best done with third-party tools, and is out of scope of this document. One suitable tool is Setuptools, whose documentation can be found at https://setuptools.pypa.io/en/latest/setuptools.html.

Normally, the initialization function returns a module definition initialized using **`PyModuleDef_Init()`**. This allows splitting the creation process into several phases:

- Before any substantial code is executed, Python can determine which capabilities the module supports, and it can adjust the environment or refuse loading an incompatible extension.
- By default, Python itself creates the module object – that is, it does the equivalent of **`object.__new__()`** for classes. It also sets initial attributes like **`__package__`** and **`__loader__`**.
- Afterwards, the module object is initialized using extension-specific code – the equivalent of **`__init__()`** on classes.

This is called *multi-phase initialization* to distinguish it from the legacy (but still supported) *single-phase initialization* scheme, where the initialization function returns a fully constructed module. See the single-phase-initialization section below for details.

Distinto en la versión 3.5: Added support for multi-phase initialization (**PEP 489** [https://peps.python.org/pep-0489/]).

Multiple module instances

By default, extension modules are not singletons. For example, if the `sys.modules` entry is removed and the module is re-imported, a new module object is created, and typically populated with fresh method and type objects. The old module is subject to normal garbage collection. This mirrors the behavior of pure-Python modules.

Additional module instances may be created in sub-interpreters or after Python runtime reinitialization (`Py_Finalize()` and `Py_Initialize()`). In these cases, sharing Python objects between module instances would likely cause crashes or undefined behavior.

To avoid such issues, each instance of an extension module should be *isolated*: changes to one instance should not implicitly affect the others, and all state owned by the module, including references to Python objects, should be specific to a particular module instance. See Aislamiento de módulos de extensión for more details and a practical guide.

A simpler way to avoid these issues is raising an error on repeated initialization.

All modules are expected to support sub-interpreters, or otherwise explicitly signal a lack of support. This is usually achieved by isolation or blocking repeated initialization, as above. A module may also be limited to the main interpreter using the `Py_mod_multiple_interpreters` slot.

Initialization function

The initialization function defined by an extension module has the following signature:

PyObject *PyInit_modulename(void)

Its name should be `PyInit_<name>`, with `<name>` replaced by the name of the module.

For modules with ASCII-only names, the function must instead be named `PyInit_<name>`, with `<name>` replaced by the name of the module. When using Multi-phase initialization, non-ASCII module names are allowed. In this case, the initialization function name is `PyInitU_<name>`, with `<name>` encoded using Python's *punycode* encoding with hyphens replaced by underscores. In Python:

```
def initfunc_name(name):
    try:
        suffix = b'_' + name.encode('ascii')
    except UnicodeEncodeError:
        suffix = b'U_' + name.encode('punycode').replace
    return b'PyInit' + suffix
```

It is recommended to define the initialization function using a helper macro:

PyMODINIT_FUNC

Declare an extension module initialization function. This macro:

- specifies the PyObject* return type,
- adds any special linkage declarations required by the platform, and
- for C++, declares the function as `extern "C"`.

For example, a module called `spam` would be defined like this:

```
static struct PyModuleDef spam_module = {
    .m_base = PyModuleDef_HEAD_INIT,
    .m_name = "spam",
    ...
};

PyMODINIT_FUNC
PyInit_spam(void)
{
    return PyModuleDef_Init(&spam_module);
}
```

It is possible to export multiple modules from a single shared library by defining multiple initialization functions. However, importing them requires using symbolic links or a custom importer, because by default only the function corresponding to the filename is found. See the Multiple modules in one library [https://peps.python.org/pep-0489/#multiple-modules-in-one-library] section in **PEP 489** [https://peps.python.org/pep-0489/] for details.

The initialization function is typically the only non-`static` item defined in the module's C source.

Multi-phase initialization

Normally, the initialization function (`PyInit_modulename`) returns a **`PyModuleDef`** instance with non-`NULL` **`m_slots`**. Before it is returned, the `PyModuleDef` instance must be initialized using the following function:

PyObject *PyModuleDef_Init(PyModuleDef *def)

Part of the Stable ABI since version 3.5.
Ensure a module definition is a properly initialized Python object that correctly reports its type and a reference count.

Return *def* cast to `PyObject*`, or `NULL` if an error occurred.

Calling this function is required for Multi-phase initialization. It should not be used in other contexts.

Note that Python assumes that `PyModuleDef` structures are statically allocated. This function may return either a new reference or a borrowed one; this reference must not be released.

Added in version 3.5.

Legacy single-phase initialization

Atención

Single-phase initialization is a legacy mechanism to initialize extension modules, with known drawbacks and design flaws. Extension module authors are encouraged to use multi-phase initialization instead.

In single-phase initialization, the initialization function (`PyInit_modulename`) should create, populate and return a module object. This is typically done using **`PyModule_Create()`** and functions like **`PyModule_AddObjectRef()`**.

Single-phase initialization differs from the default in the following ways:

- Single-phase modules are, or rather *contain*, "singletons".

  When the module is first initialized, Python saves the contents of the module's `__dict__` (that is, typically, the module's functions and types).

  For subsequent imports, Python does not call the initialization function again. Instead, it creates a new module object with a new `__dict__`, and copies the saved contents to it. For example, given a single-phase module `_testsinglephase` [1] that defines a function `sum` and an exception class `error`:

  ```
  >>> import sys
  >>> import _testsinglephase as one
  >>> del sys.modules['_testsinglephase']
  >>> import _testsinglephase as two
  >>> one is two
  ```

```
False
>>> one.__dict__ is two.__dict__
False
>>> one.sum is two.sum
True
>>> one.error is two.error
True
```

  The exact behavior should be considered a CPython implementation detail.

- To work around the fact that `PyInit_modulename` does not take a *spec* argument, some state of the import machinery is saved and applied to the first suitable module created during the `PyInit_modulename` call. Specifically, when a sub-module is imported, this mechanism prepends the parent package name to the name of the module.

  A single-phase `PyInit_modulename` function should create “its” module object as soon as possible, before any other module objects can be created.

- Non-ASCII module names (`PyInitU_modulename`) are not supported.

- Single-phase modules support module lookup functions like **`PyState_FindModule()`**.

`_testsinglephase` is an internal module used in CPython’s self-test suite; your installation may or may not include it.

Utilidades

Las funciones de este capítulo realizan varias tareas de utilidad, que van desde ayudar a que el código C sea más portátil en todas las plataformas, usar módulos Python desde C y analizar argumentos de funciones y construir valores Python a partir de valores C.

- Utilidades del sistema operativo
- Funciones del Sistema
- Control de procesos
- Importando módulos
- Soporte de empaquetado (*marshalling*) de datos
- Analizando argumentos y construyendo valores
  - Analizando argumentos
    - Cadena de caracteres y búferes
    - Números
    - Otros objetos
    - Funciones API
  - Construyendo valores
- Conversión y formato de cadenas de caracteres
- Character classification and conversion
- PyHash API
- Reflexión
- Registro de códec y funciones de soporte
  - API de búsqueda de códec
  - API de registro para controladores de errores de codificación Unicode
  - Codec utility variables
- PyTime C API
  - Types

  - Clock Functions
  - Raw Clock Functions
  - Conversion functions

- Soporte para Mapeo Perf

Utilidades del sistema operativo

PyObject *PyOS_FSPath(PyObject *path)

Return value: New reference. Part of the Stable ABI since version 3.6.
Retorna la representación del sistema de archivos para *path*. Si el objeto es **`str`** o **`bytes`**, entonces se retorna una nueva referencia fuerte. Si el objeto implementa la interfaz **`os.PathLike`**, entonces **`__fspath__()`** se retorna siempre que sea un objeto **`str`** o **`bytes`**. De lo contrario **`TypeError`** se lanza y se retorna `NULL`.

Added in version 3.6.

int Py_FdIsInteractive(FILE *fp, const char *filename)

Retorna verdadero (distinto de cero) si el archivo de E/S (*I/O*) estándar *fp* con nombre *filename* se considera interactivo. Este es el caso de los archivos para los que `isatty(fileno(fp))` es verdadero. Si **`PyConfig.interactive`** es distinto de cero, esta función también retorna verdadero si el puntero *filename* es `NULL` o si el nombre es igual a una de las cadenas de caracteres `'<stdin>'` o `'???'`.

Esta función no debe ser llamada antes de que Python sea inicializado.

void PyOS_BeforeFork()

Part of the Stable ABI on platforms with fork() since version 3.7.
Función para preparar algún estado interno antes de una bifurcación de proceso (*process fork*). Esto debería llamarse antes de llamar a **`fork()`** o cualquier función similar que clone el proceso actual. Solo disponible en sistemas donde

`fork()` está definido.

Advertencia

La llamada C **`fork()`** solo debe hacerse desde hilo «principal» (del intérprete «principal»). Lo mismo es cierto para `PyOS_BeforeFork()`.

Added in version 3.7.

void PyOS_AfterFork_Parent()

Part of the Stable ABI on platforms with fork() since version 3.7. Función para actualizar algún estado interno después de una bifurcación de proceso. Se debe invocar desde el proceso principal después de llamar a **`fork()`** o cualquier función similar que clone el proceso actual, independientemente de si la clonación del proceso fue exitosa. Solo disponible en sistemas donde **`fork()`** está definido.

Advertencia

La llamada C **`fork()`** solo debe hacerse desde hilo «principal» (del intérprete «principal»). Lo mismo es cierto para `PyOS_AfterFork_Parent()`.

Added in version 3.7.

void PyOS_AfterFork_Child()

Part of the Stable ABI on platforms with fork() since version 3.7. Función para actualizar el estado del intérprete interno después de una bifurcación de proceso (*process fork*). Debe llamarse desde el proceso secundario después de llamar a **`fork()`**, o cualquier función similar que clone el proceso actual, si existe alguna posibilidad de que el proceso vuelva a llamar al intérprete de Python. Solo disponible en sistemas donde **`fork()`** está definido.

Advertencia

La llamada C `fork()` solo debe hacerse desde hilo «principal» (del intérprete «principal»). Lo mismo es cierto para `PyOS_AfterFork_Child()`.

Added in version 3.7.

Ver también

`os.register_at_fork()` permite registrar funciones personalizadas de Python a las que puede llamar `PyOS_BeforeFork()`, `PyOS_AfterFork_Parent()` y `PyOS_AfterFork_Child()`.

void PyOS_AfterFork()

Part of the Stable ABI on platforms with fork().
Función para actualizar algún estado interno después de una bifurcación de proceso (*process fork*); Esto debería llamarse en el nuevo proceso si el intérprete de Python continuará siendo utilizado. Si se carga un nuevo ejecutable en el nuevo proceso, no es necesario llamar a esta función.

Obsoleto desde la versión 3.7: Esta función es reemplazada por `PyOS_AfterFork_Child()`.

int PyOS_CheckStack()

Part of the Stable ABI on platforms with USE_STACKCHECK since version 3.7.
Retorna verdadero cuando el intérprete se queda sin espacio de pila (*stack space*). Esta es una verificación confiable, pero solo está disponible cuando `USE_STACKCHECK` está definido (actualmente en algunas versiones de Windows usando el compilador de *Microsoft Visual C++*). `USE_STACKCHECK` se definirá automáticamente; nunca debe cambiar la definición en su propio código.

typedef void (*PyOS_sighandler_t)(int)

Part of the Stable ABI.

PyOS_sighandler_t PyOS_getsig(int i)

Part of the Stable ABI.

Retorna el controlador de señal actual para la señal *i*. Esta es una pequeña envoltura alrededor de `sigaction()` o `signal()`. ¡No llame a esas funciones directamente!

PyOS_sighandler_t PyOS_setsig(int i, PyOS_sighandler_t h)

Part of the Stable ABI.

Configura el controlador de señal para la señal *i* como *h*; retorna el antiguo controlador de señal. Esta es una pequeña envoltura alrededor de `sigaction()` o `signal()`. ¡No llame a esas funciones directamente!

int PyOS_InterruptOccurred(void)

Part of the Stable ABI.

Check if a `SIGINT` signal has been received.

Returns `1` if a `SIGINT` has occurred and clears the signal flag, or `0` otherwise.

In most cases, you should prefer `PyErr_CheckSignals()` over this function. `PyErr_CheckSignals()` invokes the appropriate signal handlers for all pending signals, allowing Python code to handle the signal properly. This function only detects `SIGINT` and does not invoke any Python signal handlers.

This function is async-signal-safe and this function cannot fail. The caller must hold an attached thread state.

wchar_t *Py_DecodeLocale(const char *arg, size_t *size)

Part of the Stable ABI since version 3.7.

Advertencia

Esta función no debe llamarse directamente: utilice la API `PyConfig` con la función `PyConfig_SetBytesString()` que asegura que Python está preinicializado.

Esta función no debe llamarse antes de que Python esté preinicializado y para que la configuración local LC_CTYPE esté correctamente configurada: véase la función **`Py_PreInitialize()`**.

Decodifica una cadena de bytes a partir del manejador de codificación y errores del sistema de archivos. Si el controlador de error es el controlador de error surrogateescape, los bytes no codificables se decodifican como caracteres en el rango U+DC80..U+DCFF; y si una secuencia de bytes se puede decodificar como un carácter sustituto, escape los bytes usando el controlador de error surrogateescape en lugar de decodificarlos.

Retorna un puntero a una cadena de caracteres anchos recientemente asignada, use **`PyMem_RawFree()`** para liberar la memoria. Si el tamaño no es `NULL`, escribe el número de caracteres anchos excluyendo el carácter nulo en `*size`

Retorna `NULL` en caso de error de decodificación o error de asignación de memoria. Si *size* no es `NULL`, `*size` se establece en `(size_t) -1` en caso de error de memoria o en `(size_t) -2` en caso de error de decodificación.

El filesystem encoding and error handler son seleccionados por **`PyConfig_Read()`**: ver **`filesystem_encoding`** y **`filesystem_errors`** que pertenecen a **`PyConfig`**.

Los errores de decodificación nunca deberían ocurrir, a menos que haya un error en la biblioteca C.

Utilice la función **`Py_EncodeLocale()`** para codificar la cadena de caracteres en una cadena de bytes.

Ver también

Las funciones **`PyUnicode_DecodeFSDefaultAndSize()`** y **`PyUnicode_DecodeLocaleAndSize()`**.

Added in version 3.5.

Distinto en la versión 3.7: La función ahora utiliza la codificación UTF-8 en el Modo Python UTF-8.

Distinto en la versión 3.8: La función ahora usa la codificación UTF-8 en Windows si **`PyPreConfig.legacy_windows_fs_encoding`** es cero;

char *Py_EncodeLocale(const wchar_t *text, size_t *error_pos)

Part of the Stable ABI since version 3.7.
Codifica una cadena de caracteres amplios según el término filesystem encoding and error handler. Si el gestor de errores es surrogateescape error handler, los caracteres sustituidos en el rango U+DC80..U+DCFF se convierten en bytes 0x80..0xFF.

Retorna un puntero a una cadena de bytes recién asignada, usa **`PyMem_Free()`** para liberar la memoria. Retorna `NULL` si se genera un error de codificación o error de asignación de memoria.

Si *error_pos* no es `NULL`, `*error_pos` se establece en `(size_t)-1` en caso de éxito, o se establece en el índice del carácter no válido en el error de codificación.

El filesystem encoding and error handler son seleccionados por **`PyConfig_Read()`**: ver **`filesystem_encoding`** y **`filesystem_errors`** que pertenecen a **`PyConfig`**.

Use la función **`Py_DecodeLocale()`** para decodificar la cadena de bytes en una cadena de caracteres anchos.

Advertencia

Esta función no debe llamarse antes de que Python esté preinicializado y para que la configuración local LC_CTYPE esté correctamente configurada: véase la función **`Py_PreInitialize()`**.

Ver también

Las funciones `PyUnicode_EncodeFSDefault()` y `PyUnicode_EncodeLocale()`.

Added in version 3.5.

Distinto en la versión 3.7: La función ahora utiliza la codificación UTF-8 en el Modo Python UTF-8.

Distinto en la versión 3.8: La función ahora usa la codificación UTF-8 en Windows si `PyPreConfig.legacy_windows_fs_encoding` es cero.

FILE *Py_fopen(PyObject *path, const char *mode)

Similar to `fopen()`, but *path* is a Python object and an exception is set on error.

path must be a `str` object, a `bytes` object, or a path-like object.

On success, return the new file pointer. On error, set an exception and return `NULL`.

The file must be closed by `Py_fclose()` rather than calling directly `fclose()`.

The file descriptor is created non-inheritable (**PEP 446** [https://peps.python.org/pep-0446/]).

The caller must have an attached thread state.

Added in version 3.14.

int Py_fclose(FILE *file)

Close a file that was opened by `Py_fopen()`.

On success, return `0`. On error, return `EOF` and `errno` is set to indicate the error. In either case, any further access (including another call to `Py_fclose()`) to the stream results in undefined behavior.

Added in version 3.14.

Funciones del Sistema

Estas son funciones de utilidad que hacen que la funcionalidad del módulo `sys` sea accesible para el código C. Todos funcionan con el diccionario del módulo `sys` del subproceso actual del intérprete, que está contenido en la estructura interna del estado del subproceso.

PyObject *PySys_GetObject(const char *name)

Return value: Borrowed reference. Part of the Stable ABI.
Retorna el objeto *name* del módulo `sys` o `NULL` si no existe, sin establecer una excepción.

int PySys_SetObject(const char *name, PyObject *v)

Part of the Stable ABI.
Establece *name* en el módulo `sys` en *v* a menos que *v* sea `NULL`, en cuyo caso *name* se elimina del módulo *sys*. Retorna `0` en caso de éxito, `-1` en caso de error.

void PySys_ResetWarnOptions()

Part of the Stable ABI.
Restablece `sys.warnoptions` a una lista vacía. Esta función puede llamarse antes de `Py_Initialize()`.

Deprecated since version 3.13, will be removed in version 3.15: En su lugar, borra `sys.warnoptions` y `warnings.filters`.

void PySys_WriteStdout(const char *format, ...)

Part of the Stable ABI.
Escribe la cadena de caracteres de salida descrita por *format* en `sys.stdout`. No se lanzan excepciones, incluso si se produce el truncamiento (ver más abajo).

format debe limitar el tamaño total de la cadena de caracteres de salida formateada a 1000 bytes o menos; después de 1000 bytes, la cadena de caracteres de salida se trunca. En particular, esto significa que no deben existir formatos «%s» sin restricciones; estos deben limitarse usando «%.<N>s» donde <N> es un número decimal calculado de modo que <N> más el tamaño máximo de otro texto formateado no exceda los 1000 bytes. También tenga cuidado con «%f», que puede imprimir cientos de dígitos para números muy grandes.

Si ocurre un problema, o **`sys.stdout`** no está configurado, el mensaje formateado se escribe en el real (nivel C) *stdout*.

void PySys_WriteStderr(const char *format, ...)

Part of the Stable ABI.
Como **`PySys_WriteStdout()`**, pero escribe a **`sys.stderr`** o *stderr* en su lugar.

void PySys_FormatStdout(const char *format, ...)

Part of the Stable ABI.
Función similar a `PySys_WriteStdout()` pero formatea el mensaje usando **`PyUnicode_FromFormatV()`** y no trunca el mensaje a una longitud arbitraria.

Added in version 3.2.

void PySys_FormatStderr(const char *format, ...)

Part of the Stable ABI.
Como **`PySys_FormatStdout()`**, pero escribe a **`sys.stderr`** o *stderr* en su lugar.

Added in version 3.2.

PyObject *PySys_GetXOptions()

Return value: Borrowed reference. Part of the Stable ABI since version 3.7.
Retorna el diccionario actual de opciones **`-X`**, de manera similar a **`sys._xoptions`**. En caso de error, se retorna

`NULL` y se establece una excepción.

Added in version 3.2.

int PySys_Audit(const char *event, const char *format, ...)

Part of the Stable ABI since version 3.13.
Lanza un evento de auditoría con cualquier gancho activo. Retorna cero para el éxito y no cero con una excepción establecida en caso de error.

El argumento de cadena *event* no debe ser *NULL*.

Si se han agregado ganchos, *format* y otros argumentos se utilizarán para construir una tupla para pasar. Además de `N`, están disponibles los mismos caracteres de formato que los utilizados en **`Py_BuildValue()`**. Si el valor generado no es una tupla, se agregará a una tupla de un solo elemento.

La opción de formato `N` no debe usarse. Consume una referencia, pero dado que no hay forma de saber si los argumentos de esta función serán consumidos, usarla puede causar fugas de referencia.

Tenga en cuenta que los caracteres de formato `#` deben tratarse como **`Py_ssize_t`**, independientemente de si se definió `PY_SSIZE_T_CLEAN`.

`sys.audit()` realiza la misma función del código Python.

Ver también **`PySys_AuditTuple()`**.

Added in version 3.8.

Distinto en la versión 3.8.2: Requiere **`Py_ssize_t`** para los caracteres de formato `#`. Anteriormente, se lanzaba una advertencia de deprecación inevitable.

int PySys_AuditTuple(const char *event, PyObject *args)

Part of the Stable ABI since version 3.13.
Similar a **`PySys_Audit()`**, pero pasa argumentos como un

objeto Python. *args* debe ser una `tuple`. Para no pasar argumentos, *args* puede ser *NULL*.

Added in version 3.13.

int PySys_AddAuditHook(Py_AuditHookFunction hook, void *userData)

Agrega el *hook* invocable a la lista de hooks de auditoría activos. Retorna cero para el éxito y no cero en caso de error. Si el tiempo de ejecución se ha inicializado, también configura un error en caso de fallo. Los hooks agregados a través de esta API se llaman para todos los intérpretes creados por el tiempo de ejecución.

El puntero *userData* se pasa a la función gancho. Dado que las funciones de enlace pueden llamarse desde diferentes tiempos de ejecución, este puntero no debe referirse directamente al estado de Python.

Es seguro llamar a esta función antes de `Py_Initialize()`. Cuando se llama después de la inicialización del tiempo de ejecución, se notifican los enlaces de auditoría existentes y pueden anular silenciosamente la operación al generar un error subclasificado de `Excepción` (otros errores no se silenciarán).

The hook function is always called with an attached thread state by the Python interpreter that raised the event.

Ver **PEP 578** [https://peps.python.org/pep-0578/] para una descripción detallada de la auditoría. Las funciones en el tiempo de ejecución y la biblioteca estándar que generan eventos se enumeran en table de eventos de auditoria. Los detalles se encuentran en la documentación de cada función.

Si el intérprete se inicializa, esta función lanza un evento de auditoría `sys.addaudithook` sin argumentos. Si algún gancho existente lanza una excepción derivada de `Exception`, el nuevo gancho no se agregará y la excepción se borrará. Como resultado, las personas que llaman no pueden asumir que su gancho ha sido agregado a menos que controlen todos los ganchos existentes.

typedef int (*Py_AuditHookFunction)(const char *event, PyObject *args, void *userData)

El tipo de la función gancho. *event* es el argumento de cadena C de evento pasado a **`PySys_Audit()`** o **`PySys_AuditTuple()`**. Se garantiza que *args* es un **`PyTupleObject`**. *userData* es el argumento pasado a PySys_AddAuditHook().

Added in version 3.8.

Control de procesos

void Py_FatalError(const char *message)

Part of the Stable ABI.
Imprime un mensaje de error fatal y elimina el proceso. No se realiza limpieza. Esta función solo debe invocarse cuando se detecta una condición que haría peligroso continuar usando el intérprete de Python; por ejemplo, cuando la administración del objeto parece estar dañada. En Unix, se llama a la función de biblioteca C estándar **`abort()`** que intentará producir un archivo `core`.

La función `Py_FatalError()` se reemplaza con una macro que registra automáticamente el nombre de la función actual, a menos que se defina la macro `Py_LIMITED_API`.

Distinto en la versión 3.9: Registra el nombre de la función automáticamente.

void Py_Exit(int status)

Part of the Stable ABI.
Sale del proceso actual. Esto llama **`Py_FinalizeEx()`** y luego llama a la función estándar de la biblioteca C

`exit(status)`. Si **`Py_FinalizeEx()`** indica un error, el estado de salida se establece en 120.

Distinto en la versión 3.6: Los errores de finalización ya no se ignoran.

int Py_AtExit(void (*func)())

Part of the Stable ABI.
Registra una función de limpieza a la que llamará **`Py_FinalizeEx()`**. Se llamará a la función de limpieza sin argumentos y no debería retornar ningún valor. Como máximo se pueden registrar 32 funciones de limpieza. Cuando el registro es exitoso, **`Py_AtExit()`** retorna `0`; en caso de error, retorna `-1`. La última función de limpieza registrada se llama primero. Cada función de limpieza se llamará como máximo una vez. Dado que la finalización interna de Python se habrá completado antes de la función de limpieza, *func* no debería llamar a las API de Python.

Ver también

`PyUnstable_AtExit()` for passing a `void *data` argument.

Importando módulos

PyObject *PyImport_ImportModule(const char *name)

Return value: New reference. Part of the Stable ABI.
Esta es una envoltura alrededor de **`PyImport_Import()`** que toma un const char* como argumento en lugar de un PyObject*.

PyObject *PyImport_ImportModuleNoBlock(const char *name)

Return value: New reference. Part of the Stable ABI.
Esta función es un alias obsoleto de **`PyImport_ImportModule()`**.

Distinto en la versión 3.3: Esta función solía fallar inmediatamente cuando el bloqueo de importación era retenido por otro hilo. Sin embargo, en Python 3.3, el esquema de bloqueo cambió a bloqueos por módulo para la mayoría de los propósitos, por lo que el comportamiento especial de esta función ya no es necesario.

Deprecated since version 3.13, will be removed in version 3.15: Usa **`PyImport_ImportModule()`** en su lugar.

PyObject *PyImport_ImportModuleEx(const char *name, PyObject *globals, PyObject *locals, PyObject *fromlist)

Return value: New reference.
Importa un módulo. Esto se describe mejor refiriéndose a la función incorporada de Python **`__import__()`**.

El valor de retorno es una nueva referencia al módulo importado o paquete de nivel superior, o `NULL` con una excepción establecida en caso de error. Como para **`__import__()`**, el valor de retorno cuando se solicita un submódulo de un paquete es normalmente el paquete de nivel superior, a menos que se proporcione una *fromlist* no vacía.

Las importaciones que fallan eliminan objetos de módulo incompletos, como con `PyImport_ImportModule()`.

PyObject *PyImport_ImportModuleLevelObject(PyObject *name, PyObject *globals, PyObject *locals, PyObject *fromlist, int level)

Return value: New reference. Part of the Stable ABI since version 3.7.
Importa un módulo. Esto se describe mejor haciendo referencia a la función Python incorporada `__import__()`, ya que la función estándar `__import__()` llama a esta función directamente.

El valor de retorno es una nueva referencia al módulo importado o paquete de nivel superior, o `NULL` con una excepción establecida en caso de error. Como para `__import__()`, el valor de retorno cuando se solicita un submódulo de un paquete es normalmente el paquete de nivel superior, a menos que se proporcione una *fromlist* no vacía.

Added in version 3.3.

PyObject *PyImport_ImportModuleLevel(const char *name, PyObject *globals, PyObject *locals, PyObject *fromlist, int level)

Return value: New reference. Part of the Stable ABI.
Similar a `PyImport_ImportModuleLevelObject()`, pero el nombre es una cadena de caracteres codificada UTF-8 en lugar de un objeto Unicode.

Distinto en la versión 3.3: Los valores negativos para *level* ya no se aceptan.

PyObject *PyImport_Import(PyObject *name)

Return value: New reference. Part of the Stable ABI.
Esta es una interfaz de nivel superior que llama a la «función de enlace de importación» actual (con un nivel explícito de 0, que significa importación absoluta). Invoca la función `__import__()` de las `__builtins__` de los globales (*globals*) actuales. Esto significa que la importación se realiza

utilizando los ganchos de importación instalados en el entorno actual.

Esta función siempre usa importaciones absolutas.

PyObject *PyImport_ReloadModule(PyObject *m)

Return value: New reference. Part of the Stable ABI.
Recarga un módulo. Retorna una nueva referencia al módulo recargado, o `NULL` con una excepción establecida en caso de error (el módulo todavía existe en este caso).

PyObject *PyImport_AddModuleRef(const char *name)

Return value: New reference. Part of the Stable ABI since version 3.13.
Retorna el objeto módulo correspondiente a un nombre de módulo.

El argumento *name* puede tener la forma `package.module`. Primero verifica el diccionario de módulos si hay uno allí, y si no, crea uno nuevo y lo inserta en el diccionario de módulos.

Retorna una referencia fuerte al módulo en caso de éxito. Retorna `NULL` con una excepción establecida en caso de error.

El nombre del módulo *name* se decodifica desde UTF-8.

Esta función no carga ni importa el módulo; si el módulo no estaba cargado, obtendrás un objeto de módulo vacío. Utiliza **`PyImport_ImportModule()`** o una de sus variantes para importar un módulo. Las estructuras de paquete implicadas por un nombre punteado para *name* no se crean si aún no están presentes.

Added in version 3.13.

PyObject *PyImport_AddModuleObject(PyObject *name)

Return value: Borrowed reference. Part of the Stable ABI since version 3.7.

Similar a `PyImport_AddModuleRef()`, pero retorna una referencia prestada y *name* es un objeto Python `str`.

Added in version 3.3.

PyObject *PyImport_AddModule(const char *name)

Return value: Borrowed reference. Part of the Stable ABI.
Similar a `PyImport_AddModuleRef()`, pero retorna una referencia prestada.

PyObject *PyImport_ExecCodeModule(const char *name, PyObject *co)

Return value: New reference. Part of the Stable ABI.
Dado un nombre de módulo (posiblemente de la forma `package.module`) y un objeto código leído desde un archivo de *bytecode* de Python u obtenido de la función incorporada `compile()`, carga el módulo. Retorna una nueva referencia al objeto módulo, o `NULL` con una excepción establecida si se produjo un error. *name* se elimina de `sys.modules` en casos de error, incluso si *name* ya estaba en `sys.modules` en la entrada a `PyImport_ExecCodeModule()`. Dejar módulos inicializados de forma incompleta en `sys.modules` es peligroso, ya que las importaciones de dichos módulos no tienen forma de saber que el objeto del módulo es un estado desconocido (y probablemente dañado con respecto a las intenciones del autor del módulo).

The module's `__spec__` and `__loader__` will be set, if not set already, with the appropriate values. The spec's loader will be set to the module's `__loader__` (if set) and to an instance of `SourceFileLoader` otherwise.

The module's `__file__` attribute will be set to the code object's `co_filename`. If applicable, `__cached__` will also be set.

Esta función volverá a cargar el módulo si ya se importó. Consulte `PyImport_ReloadModule()` para conocer la forma prevista de volver a cargar un módulo.

Si *name* apunta a un nombre punteado de la forma `package.module`, cualquier estructura de paquete que no se haya creado aún no se creará.

Ver también **`PyImport_ExecCodeModuleEx()`** y **`PyImport_ExecCodeModuleWithPathnames()`**.

Distinto en la versión 3.12: The setting of **`__cached__`** and **`__loader__`** is deprecated. See **`ModuleSpec`** for alternatives.

PyObject *PyImport_ExecCodeModuleEx(const char *name, PyObject *co, const char *pathname)

Return value: New reference. Part of the Stable ABI.
Like **`PyImport_ExecCodeModule()`**, but the **`__file__`** attribute of the module object is set to *pathname* if it is non-`NULL`.

Ver también **`PyImport_ExecCodeModuleWithPathnames()`**.

PyObject *PyImport_ExecCodeModuleObject(PyObject *name, PyObject *co, PyObject *pathname, PyObject *cpathname)

Return value: New reference. Part of the Stable ABI since version 3.7.
Like **`PyImport_ExecCodeModuleEx()`**, but the **`__cached__`** attribute of the module object is set to *cpathname* if it is non-`NULL`. Of the three functions, this is the preferred one to use.

Added in version 3.3.

Distinto en la versión 3.12: Setting **`__cached__`** is deprecated. See **`ModuleSpec`** for alternatives.

PyObject *PyImport_ExecCodeModuleWithPathnames(const char *name, PyObject *co, const char *pathname, const char *cpathname)

Return value: New reference. Part of the Stable ABI.
Como **`PyImport_ExecCodeModuleObject()`**, pero *name*, *pathname* y *cpathname* son cadenas codificadas UTF-8. También se hacen intentos de averiguar cuál debería ser el valor para *pathname* desde *cpathname* si el primero se establece en `NULL`.

Added in version 3.2.

Distinto en la versión 3.3: Utiliza **`imp.source_from_cache()`** para calcular la ruta de origen si solo se proporciona la ruta del *bytecode*.

Distinto en la versión 3.12: Ya no usa el módulo **`imp`** eliminado.

long PyImport_GetMagicNumber()

Part of the Stable ABI.
Retorna el número mágico para archivos de bytecode de Python (también conocidos como archivos `.pyc`). El número mágico debería estar presente en los primeros cuatro bytes del archivo bytecode, en orden de bytes little-endian. Retorna `-1` en caso de error.

Distinto en la versión 3.3: Retorna un valor de `-1` en caso de error.

const char *PyImport_GetMagicTag()

Part of the Stable ABI.
Retorna la cadena de caracteres de etiqueta mágica para nombres de archivo de código de bytes Python en formato **PEP 3147** [https://peps.python.org/pep-3147/]. Tenga en cuenta que el valor en `sys.implementation.cache_tag` es autoritario y debe usarse en lugar de esta función.

Added in version 3.2.

PyObject *PyImport_GetModuleDict()

Return value: Borrowed reference. Part of the Stable ABI.
Retorna el diccionario utilizado para la administración del módulo (también conocido como `sys.modules`). Tenga en cuenta que esta es una variable por intérprete.

PyObject *PyImport_GetModule(PyObject *name)

Return value: New reference. Part of the Stable ABI since version 3.8.
Retorna el módulo ya importado con el nombre dado. Si el módulo aún no se ha importado, retorna `NULL` pero no establece un error. Retorna `NULL` y establece un error si falla la búsqueda.

Added in version 3.7.

PyObject *PyImport_GetImporter(PyObject *path)

Return value: New reference. Part of the Stable ABI.
Retorna un objeto buscador para un elemento *path* **`sys.path`**/**`pkg.__path__`**, posiblemente obteniéndolo del diccionario **`sys.path_importer_cache`**. Si aún no estaba en caché, atraviesa **`sys.path_hooks`** hasta que se encuentre un gancho (*hook*) que pueda manejar el elemento de ruta. Retorna `None` si ningún gancho (*hook*) podría; esto le dice a la persona que llama que path based finder no pudo encontrar un buscador para este elemento de ruta. Guarda en el resultado (caché) en **`sys.path_importer_cache`**. Retorna una nueva referencia al objeto del buscador.

int PyImport_ImportFrozenModuleObject(PyObject *name)

Part of the Stable ABI since version 3.7.
Carga un módulo congelado llamado *name*. Retorna `1` para el éxito, `0` si no se encuentra el módulo y `-1` con una excepción establecida si falla la inicialización. Para acceder al módulo importado con una carga exitosa, use **`PyImport_ImportModule()`**. (Tenga en cuenta el nombre inapropiado — esta función volvería a cargar el módulo si ya se importó).

Added in version 3.3.

Distinto en la versión 3.4: El atributo `__file__` ya no está establecido en el módulo.

int PyImport_ImportFrozenModule(const char *name)

Part of the Stable ABI.

Similar a **`PyImport_ImportFrozenModuleObject()`**, pero el nombre es una cadena de caracteres codificada UTF-8 en lugar de un objeto Unicode.

struct _frozen

Esta es la definición del tipo de estructura para los descriptores de módulos congelados, según lo generado con la herramienta **freeze** (ver `Tools/freeze` en la distribución de código fuente de Python). Su definición, que se encuentra en `Include/import.h`, es:

```
struct _frozen {
    const char *name;
    const unsigned char *code;
    int size;
    bool is_package;
};
```

Distinto en la versión 3.11: El nuevo campo `is_package` indica si el módulo es un paquete o no. Esto sustituye a la configuración del campo `size` con un valor negativo.

const struct _frozen *PyImport_FrozenModules

Este puntero se inicializa para apuntar a un arreglo de registros **`_frozen`**, terminado por uno cuyos registros son todos `NULL` o cero. Cuando se importa un módulo congelado, se busca en esta tabla. El código de terceros podría jugar con esto para proporcionar una colección de módulos congelados creada dinámicamente.

int PyImport_AppendInittab(const char *name, PyObject *(*initfunc)(void))

Part of the Stable ABI.

Agrega un solo módulo a la tabla existente de módulos incorporados. Este es un contenedor conveniente **`PyImport_ExtendInittab()`**, que retorna `-1` si la tabla no se puede extender. El nuevo módulo se puede importar con el nombre *name*, y utiliza la función *initfunc* como la función de inicialización llamada en el primer intento de importación. Esto debería llamarse antes de **`Py_Initialize()`**.

struct _inittab

Estructura que describe una sola entrada en la lista de módulos incorporados. Cada una de estas estructuras proporciona el nombre y la función de inicialización de un módulo incorporado en el intérprete. El nombre es una cadena de caracteres codificada ASCII. Los programas que incorporan Python pueden usar una matriz de estas estructuras junto con **`PyImport_ExtendInittab()`** para proporcionar módulos integrados adicionales. La estructura se define en `Include/import.h` como:

const char *name

El nombre del módulo, como una cadena codificada ASCII.

PyObject *(*initfunc)(void)

Función de inicialización para un módulo incorporado en el intérprete.

int PyImport_ExtendInittab(struct _inittab *newtab)

Agrega una colección de módulos a la tabla de módulos integrados. El arreglo *newtab* debe terminar con una entrada centinela que contiene `NULL` para el campo **`name`**; Si no se proporciona el valor centinela, se puede producir un error de memoria. Retorna `0` en caso de éxito o `-1` si se puede asignar memoria insuficiente para ampliar la tabla interna. En caso de error, no se agregan módulos a la tabla interna. Esta función debe ser llamada antes de **`Py_Initialize()`**.

Si Python es inicializado múltiples veces, se debe llamar

`PyImport_AppendInittab()` o `PyImport_ExtendInittab()` antes de cada inicialización de Python.

struct _inittab *PyImport_Inittab

The table of built-in modules used by Python initialization. Do not use this directly; use `PyImport_AppendInittab()` and `PyImport_ExtendInittab()` instead.

PyObject *PyImport_ImportModuleAttr(PyObject *mod_name, PyObject *attr_name)

Return value: New reference.
Import the module *mod_name* and get its attribute *attr_name*.

Names must be Python `str` objects.

Helper function combining `PyImport_Import()` and `PyObject_GetAttr()`. For example, it can raise `ImportError` if the module is not found, and `AttributeError` if the attribute doesn't exist.

Added in version 3.14.

PyObject *PyImport_ImportModuleAttrString(const char *mod_name, const char *attr_name)

Return value: New reference.
Similar to `PyImport_ImportModuleAttr()`, but names are UTF-8 encoded strings instead of Python `str` objects.

Added in version 3.14.

Soporte de empaquetado (*marshalling*) de datos

Estas rutinas permiten que el código C funcione con objetos serializados utilizando el mismo formato de datos que el módulo **`marshal`**. Hay funciones para escribir datos en el formato de serialización y funciones adicionales que se pueden usar para volver a leer los datos. Los archivos utilizados para almacenar datos ordenados deben abrirse en modo binario.

Los valores numéricos se almacenan con el byte menos significativo primero.

The module supports several versions of the data format; see the **`Python module documentation`** for details.

Py_MARSHAL_VERSION

The current format version. See **`marshal.version`**.

void PyMarshal_WriteLongToFile(long value, FILE *file, int version)

Empaqueta (*marshal*) un entero *value* long a un archivo *file*. Esto solo escribirá los 32 bits menos significativos de *value*; sin importar el tamaño del tipo long nativo. *version* indica el formato del archivo.

Esta función puede fallar, en cuyo caso establece el indicador de error. Utiliza **`PyErr_Occurred()`** para comprobarlo.

void PyMarshal_WriteObjectToFile(PyObject *value, FILE *file, int version)

Empaqueta (*marshal*) un objeto Python, *value*, a un archivo *file*. *version* indica el formato del archivo.

Esta función puede fallar, en cuyo caso establece el indicador de error. Utiliza `PyErr_Occurred()` para comprobarlo.

PyObject *PyMarshal_WriteObjectToString(PyObject *value, int version)

Return value: New reference.
Retorna un objeto de bytes que contiene la representación empaquetada (*marshalled*) de *value*. *version* indica el formato del archivo.

Las siguientes funciones permiten volver a leer los valores empaquetados (*marshalled*).

long PyMarshal_ReadLongFromFile(FILE *file)

Retorna un entero long de C desde el flujo de datos FILE* abierto para lectura. Solo se puede leer un valor de 32 bits con esta función, independientemente del tamaño nativo del tipo long.

En caso de error, establece la excepción apropiada (**`EOFError`**) y retorna `-1`.

int PyMarshal_ReadShortFromFile(FILE *file)

Retorna un entero short de C desde el flujo de datos FILE* abierto para lectura. Solo se puede leer un valor de 16 bits con esta función, independientemente del tamaño nativo del tipo short.

En caso de error, establece la excepción apropiada (**`EOFError`**) y retorna `-1`.

PyObject *PyMarshal_ReadObjectFromFile(FILE *file)

Return value: New reference.
Retorna un objeto Python del flujo de datos FILE* abierto para lectura.

En caso de error, establece la excepción apropiada (**`EOFError`**, **`ValueError`** o **`TypeError`**) y retorna `NULL`.

PyObject *PyMarshal_ReadLastObjectFromFile(FILE *file)

Return value: New reference.
Return a Python object from the data stream in a FILE* opened for reading. Unlike **`PyMarshal_ReadObjectFromFile()`**, this function assumes that no further objects will be read from the file, allowing it to aggressively load file data into memory so that the de-serialization can operate from data in memory rather than reading a byte at a time from the file. Only use this variant if you are certain that you won't be reading anything else from the file.

En caso de error, establece la excepción apropiada (**`EOFError`**, **`ValueError`** o **`TypeError`**) y retorna `NULL`.

PyObject *PyMarshal_ReadObjectFromString(const char *data, Py_ssize_t len)

Return value: New reference.
Retorna un objeto Python del flujo de datos en un búfer de bytes que contiene *len* bytes a los que apunta *data*.

En caso de error, establece la excepción apropiada (**`EOFError`**, **`ValueError`** o **`TypeError`**) y retorna `NULL`.

Analizando argumentos y construyendo valores

These functions are useful when creating your own extension functions and methods. Additional information and examples are available in Ampliación e incrustación del intérprete de Python.

Las tres primeras de estas funciones descritas, **`PyArg_ParseTuple()`**, **`PyArg_ParseTupleAndKeywords()`**, y **`PyArg_Parse()`**, todas usan *cadenas de caracteres de formato* que se utilizan para contarle a la función sobre los argumentos esperados. Las cadenas de caracteres de formato utilizan la misma sintaxis para cada una de estas funciones.

Analizando argumentos

Una cadena de formato consta de cero o más «unidades de formato.» Una unidad de formato describe un objeto Python; por lo general es un solo carácter o una secuencia de unidades formato entre paréntesis. Con unas pocas excepciones, una unidad de formato que no es una secuencia entre paréntesis normalmente corresponde a un único argumento de dirección de estas funciones. En la siguiente descripción, la forma citada es la unidad de formato; la entrada en paréntesis (redondos) es el tipo de objeto Python que coincida con la unidad de formato; y la entrada entre corchetes [cuadrados] es el tipo de la variable(s) C cuya dirección debe ser pasada.

Cadena de caracteres y búferes

Nota

En Python 3.12 y versiones anteriores, la macro **`PY_SSIZE_T_CLEAN`** debe estar definida antes de incluir

`Python.h` para usar todas las variantes `#` de formatos (`s#`, `y#`, etc.) explicadas a continuación. Esto no es necesario en Python 3.13 y versiones posteriores.

Estos formatos permiten acceder a un objeto como un bloque contiguo de memoria. Usted no tiene que proporcionar almacenamiento en bruto para el Unicode o área de bytes retornada.

A menos que se indique lo contrario, los búferes no son terminados en NULL (*NUL-terminated*).

Hay tres formas en que las cadenas de caracteres y los búferes pueden convertirse a C:

- Formats such as `y*` and `s*` fill a `Py_buffer` structure. This locks the underlying buffer so that the caller can subsequently use the buffer even inside a `Py_BEGIN_ALLOW_THREADS` block without the risk of mutable data being resized or destroyed. As a result, **you have to call** `PyBuffer_Release()` after you have finished processing the data (or in any early abort case).

- Los formatos `es`, `es#`, `et` y `et#` asignan el búfer de resultado. **Debe llamar** `PyMem_Free()` después de haber terminado de procesar los datos (o en cualquier caso de aborto temprano).

- Otros formatos toman un `str` o un objeto de tipo bytes de solo lectura, como `bytes`, y proporcionan un puntero `const char *` a su búfer. En este caso el búfer es «prestado» (*borrowed*): es gestionado por el objeto Python correspondiente y comparte el tiempo de vida de este objeto. No tendrá que liberar ninguna memoria usted mismo.

  Para asegurar que el búfer subyacente pueda ser prestado de forma segura, el campo `PyBufferProcs.bf_releasebuffer` del objeto debe ser `NULL`. Esto desautoriza objetos mutables comunes como `bytearray`, pero también algunos objetos de solo lectura como `memoryview` de `bytes`.

Además de este requisito de `bf_releasebuffer`, no hay verificación para comprobar si el objeto de entrada es inmutable (por ejemplo, si respetaría una solicitud de un búfer escribible, o si otro hilo puede mutar los datos).

`s` (**`str`**) [const char *]

Convierte un objeto Unicode a un puntero C a una cadena de caracteres. Un puntero a una cadena de caracteres existente se almacena en la variable puntero del carácter cuya dirección se pasa. La cadena de caracteres en C es terminada en NULL. La cadena de caracteres de Python no debe contener puntos de código incrustado nulos; si lo hace, se lanza una excepción **`ValueError`**. Los objetos Unicode se convierten en cadenas de caracteres de C utilizando codificación `'utf-8'`. Si esta conversión fallase lanza un **`UnicodeError`**.

Nota

Este formato no acepta objetos de tipo bytes. Si desea aceptar los caminos del sistema de archivos y convertirlos en cadenas de caracteres C, es preferible utilizar el formato `O&` con **`PyUnicode_FSConverter()`** como convertidor.

Distinto en la versión 3.5: Anteriormente, **`TypeError`** se lanzó cuando se encontraron puntos de código nulos incrustados en la cadena de caracteres de Python.

`s*` (**`str`** o bytes-like object) [Py_buffer]

Este formato acepta objetos Unicode, así como objetos de tipo bytes. Llena una estructura **`Py_buffer`** proporcionada por la persona que llama. En este caso la cadena de caracteres de C resultante puede contener bytes NUL embebidos. Los objetos Unicode se convierten en cadenas de caracteres C utilizando codificación `'utf-8'`.

`s#` (**`str`**, bytes-like object de sólo lectura) [const char *, **`Py_ssize_t`**]

Like `s*`, except that it provides a borrowed buffer. The result is stored into two C variables, the first one a pointer to a C string, the second one its length. The string may contain embedded null bytes. Unicode objects are converted to C strings using `'utf-8'` encoding.

`z` (**`str`** o `None`) [const char *]

Como `s`, pero el objeto Python también puede ser `None`, en cuyo caso el puntero C se establece en `NULL`.

`z*` (**`str`**, bytes-like object o `None`) [Py_buffer]

Como `s*`, pero el objeto Python también puede ser `None`, en cuyo caso el miembro de `buf` de la estructura **`Py_buffer`** se establece en `NULL`.

`z#` (**`str`**, bytes-like object de sólo lectura o `None`) [const char *, **`Py_ssize_t`**]

Como `s#`, pero el objeto Python también puede ser `None`, en cuyo caso el puntero C se establece en `NULL`.

`y` (bytes-like object de sólo lectura) [const char *]

This format converts a bytes-like object to a C pointer to a borrowed character string; it does not accept Unicode objects. The bytes buffer must not contain embedded null bytes; if it does, a **`ValueError`** exception is raised.

Distinto en la versión 3.5: Anteriormente, **`TypeError`** se lanzó cuando bytes nulos incrustados se encontraron en el buffer de bytes.

`y*` (bytes-like object) [Py_buffer]

Esta variante de `s*` no acepta objetos Unicode, solamente los objetos de tipo bytes. **Esta es la forma recomendada para aceptar datos binarios.**

`y#` (bytes-like object de sólo lectura) [const char *, **`Py_ssize_t`**]

Esta variante en `s#` no acepta objetos Unicode, solo objetos similares a bytes.

`S` (**`bytes`**) [PyBytesObject *]

Requires that the Python object is a **`bytes`** object, without attempting any conversion. Raises **`TypeError`** if the object is not a bytes object. The C variable may also be declared as PyObject*.

`Y` (**`bytearray`**) [PyByteArrayObject *]

Requires that the Python object is a **`bytearray`** object, without attempting any conversion. Raises **`TypeError`** if the object is not a **`bytearray`** object. The C variable may also be declared as PyObject*.

`U` (**`str`**) [PyObject *]

Requires that the Python object is a Unicode object, without attempting any conversion. Raises **`TypeError`** if the object is not a Unicode object. The C variable may also be declared as PyObject*.

`w*` (bytes-like object de lectura y escritura) [Py_buffer]

This format accepts any object which implements the read-write buffer interface. It fills a **`Py_buffer`** structure provided by the caller. The buffer may contain embedded null bytes. The caller has to call **`PyBuffer_Release()`** when it is done with the buffer.

`es` (**`str`**) [const char *encoding, char **buffer]

Esta variante en `s` se usa para codificar Unicode en un búfer de caracteres. Solo funciona para datos codificados sin bytes NUL integrados.

This format requires two arguments. The first is only used as input, and must be a const char* which points to the name of an encoding as a NUL-terminated string, or `NULL`, in which case `'utf-8'` encoding is used. An exception is raised if the named encoding is not known to Python. The second argument must be a char**; the value of the pointer it references will be set to a buffer with the contents of the argument text. The text will be encoded in the encoding specified by the first argument.

`PyArg_ParseTuple()` asignará un búfer del tamaño necesitado, copiará los datos codificados en este búfer y ajustará **buffer* para referenciar el nuevo almacenamiento asignado. Quien llama es responsable para llamar **`PyMem_Free()`** para liberar el búfer asignado después de su uso.

`et` (**`str`**, **`bytes`** o **`bytearray`**) [const char *encoding, char **buffer]

Igual que `es`, excepto que los objetos de cadena de caracteres de bytes se pasan sin recodificarlos. En cambio, la implementación supone que el objeto de cadena de caracteres de bytes utiliza la codificación que se pasa como parámetro.

`es#` (**`str`**) [const char *encoding, char **buffer, **`Py_ssize_t`** *buffer_length]

Esta variante en `s#` se usa para codificar Unicode en un búfer de caracteres. A diferencia del formato `es`, esta variante permite datos de entrada que contienen caracteres NUL.

It requires three arguments. The first is only used as input, and must be a const char* which points to the name of an encoding as a NUL-terminated string, or `NULL`, in which case `'utf-8'` encoding is used. An exception is raised if the named encoding is not known to Python. The second argument must be a char**; the value of the pointer it references will be set to a buffer with the contents of the argument text. The text will be encoded in the encoding specified by the first argument. The third argument must be a pointer to an integer; the referenced integer will be set to the number of bytes in the output buffer.

Hay dos modos de operación:

Si **buffer* señala un puntero `NULL`, la función asignará un búfer del tamaño necesario, copiará los datos codificados en este búfer y configurará **buffer* para hacer referencia al almacenamiento recién asignado. Quien llama es responsable de llamar a **`PyMem_Free()`** para liberar el búfer asignado después del uso.

Si **buffer* apunta a un puntero no `NULL` (un búfer ya asignado), **`PyArg_ParseTuple()`** usará esta ubicación como el búfer e interpretará el valor inicial de **buffer_length* como el tamaño del búfer. Luego copiará los datos codificados en el búfer y los terminará en NUL. Si el búfer no es lo suficientemente grande, se establecerá a **`ValueError`**.

En ambos casos, **buffer_length* se establece a la longitud de los datos codificados sin el byte NUL final.

`et#` (**`str`**, **`bytes`** o **`bytearray`**) [const char *encoding, char **buffer, **`Py_ssize_t`** *buffer_length]

Igual que `es#`, excepto que los objetos de cadena de caracteres de bytes se pasan sin recodificarlos. En cambio, la implementación supone que el objeto de cadena de caracteres de bytes utiliza la codificación que se pasa como parámetro.

Distinto en la versión 3.12: `u`, `u#`, `Z` y `Z#` fueron eliminados porque usaban una representación heredada `Py_UNICODE*`.

Números

These formats allow representing Python numbers or single characters as C numbers. Formats that require **`int`**, **`float`** or **`complex`** can also use the corresponding special methods **`__index__()`**, **`__float__()`** or **`__complex__()`** to convert the Python object to the required type.

For signed integer formats, **`OverflowError`** is raised if the value is out of range for the C type. For unsigned integer formats, no range checking is done — the most significant bits are silently truncated when the receiving field is too small to receive the value.

`b` (**`int`**) [unsigned char]

Convert a nonnegative Python integer to an unsigned tiny integer, stored in a C unsigned char.

`B` (**`int`**) [unsigned char]

Convert a Python integer to a tiny integer without overflow checking, stored in a C unsigned char.

`h` (**`int`**) [short int]

Convert a Python integer to a C short int.

`H` (**`int`**) [unsigned short int]

Convert a Python integer to a C unsigned short int, without overflow checking.

`i` (**`int`**) [int]

Convert a Python integer to a plain C int.

`I` (**`int`**) [unsigned int]

Convert a Python integer to a C unsigned int, without overflow checking.

`l` (**`int`**) [long int]

Convert a Python integer to a C long int.

`k` (**`int`**) [unsigned long]

Convert a Python integer to a C unsigned long without overflow checking.

Distinto en la versión 3.14: Use **`__index__()`** if available.

`L` (**`int`**) [long long]

Convert a Python integer to a C long long.

`K` (**`int`**) [unsigned long long]

Convert a Python integer to a C unsigned long long without overflow checking.

Distinto en la versión 3.14: Use **`__index__()`** if available.

`n` (**`int`**) [**`Py_ssize_t`**]

Convierte un entero de Python a un **`Py_ssize_t`** de C.

`c` (**`bytes`** o **`bytearray`** de largo 1) [char]

Convert a Python byte, represented as a **`bytes`** or **`bytearray`** object of length 1, to a C char.

Distinto en la versión 3.3: Permite objetos **`bytearray`**.

`C` (**`str`** de largo 1) [int]

Convert a Python character, represented as a **`str`** object of

length 1, to a C int.

`f` (**`float`**) [float]

Convert a Python floating-point number to a C float.

`d` (**`float`**) [double]

Convert a Python floating-point number to a C double.

`D` (**`complex`**) [Py_complex]

Convierte un número complejo de Python en una estructura **`Py_complex`** de C.

Otros objetos

`O` (object) [PyObject *]

Store a Python object (without any conversion) in a C object pointer. The C program thus receives the actual object that was passed. A new strong reference to the object is not created (i.e. its reference count is not increased). The pointer stored is not `NULL`.

`O!` (object) [*typeobject*, PyObject *]

Store a Python object in a C object pointer. This is similar to `O`, but takes two C arguments: the first is the address of a Python type object, the second is the address of the C variable (of type PyObject*) into which the object pointer is stored. If the Python object does not have the required type, **`TypeError`** is raised.

`O&` (object) [*converter*, *address*]

Convert a Python object to a C variable through a *converter* function. This takes two arguments: the first is a function, the second is the address of a C variable (of arbitrary type), converted to void*. The *converter* function in turn is called as follows:

```
status = converter(object, address);
```

where *object* is the Python object to be converted and *address* is the void* argument that was passed to the `PyArg_Parse*` function. The returned *status* should be `1` for a successful conversion and `0` if the conversion has failed. When the conversion fails, the *converter* function should raise an exception and leave the content of *address* unmodified.

If the *converter* returns **`Py_CLEANUP_SUPPORTED`**, it may get called a second time if the argument parsing eventually fails, giving the converter a chance to release any memory that it had already allocated. In this second call, the *object* parameter will be `NULL`; *address* will have the same value as in the original call.

Examples of converters: **`PyUnicode_FSConverter()`** and **`PyUnicode_FSDecoder()`**.

Distinto en la versión 3.1: **`Py_CLEANUP_SUPPORTED`** was added.

`p` (**`bool`**) [int]

Prueba el valor pasado por verdad (un booleano predicado **p**) y convierte el resultado a su valor entero C verdadero/falso entero equivalente. Establece int en `1` si la expresión era verdadera y `0` si era falsa. Esto acepta cualquier valor válido de Python. Consulte Evaluar como valor verdadero/falso para obtener más información sobre cómo Python prueba los valores por verdad.

Added in version 3.3.

`(items)` (sequence) [*matching-items*]

The object must be a Python sequence (except **`str`**, **`bytes`** or **`bytearray`**) whose length is the number of format units in *items*. The C arguments must correspond to the individual format units in *items*. Format units for sequences may be nested.

If *items* contains format units which store a borrowed buffer (`s`, `s#`, `z`, `z#`, `y`, or `y#`) or a borrowed reference (`S`, `Y`, `U`, `O`, or `O!`), the object must be a Python tuple. The *converter* for the `O&` format unit in *items* must not store a borrowed buffer or a borrowed reference.

Distinto en la versión 3.14: `str` and `bytearray` objects no longer accepted as a sequence.

Obsoleto desde la versión 3.14: Non-tuple sequences are deprecated if *items* contains format units which store a borrowed buffer or a borrowed reference.

Algunos otros caracteres tienen un significado en una cadena de formato. Esto puede no ocurrir dentro de paréntesis anidados. Son:

`|`

Indica que los argumentos restantes en la lista de argumentos de Python son opcionales. Las variables C correspondientes a argumentos opcionales deben inicializarse a su valor predeterminado — cuando no se especifica un argumento opcional, `PyArg_ParseTuple()` no toca el contenido de las variables C correspondientes.

`$`

`PyArg_ParseTupleAndKeywords()` solamente: indica que los argumentos restantes en la lista de argumentos de Python son solo palabras clave. Actualmente, todos los argumentos de solo palabras clave también deben ser argumentos opcionales, por lo que `|` siempre debe especificarse antes de `$` en la cadena de formato.

Added in version 3.3.

`:`

La lista de unidades de formato termina aquí; la cadena después de los dos puntos se usa como el nombre de la función en los mensajes de error (el «valor asociado» de la excepción que `PyArg_ParseTuple()` lanza).

`;`

La lista de unidades de formato termina aquí; la cadena después del punto y coma se usa como mensaje de error *en lugar de* del mensaje de error predeterminado. `:` y `;` se excluyen mutuamente.

Note that any Python object references which are provided to the caller are *borrowed* references; do not release them (i.e. do not decrement their reference count)!

Los argumentos adicionales pasados a estas funciones deben ser direcciones de variables cuyo tipo está determinado por la cadena de formato; Estos se utilizan para almacenar valores de la tupla de entrada. Hay algunos casos, como se describe en la lista de unidades de formato anterior, donde estos parámetros se utilizan como valores de entrada; deben coincidir con lo especificado para la unidad de formato correspondiente en ese caso.

For the conversion to succeed, the *arg* object must match the format and the format must be exhausted. On success, the `PyArg_Parse*` functions return true, otherwise they return false and raise an appropriate exception. When the `PyArg_Parse*` functions fail due to conversion failure in one of the format units, the variables at the addresses corresponding to that and the following format units are left untouched.

Funciones API

int PyArg_ParseTuple(PyObject *args, const char *format, ...)

> *Part of the Stable ABI.*
> Analiza los parámetros de una función que solo toma parámetros posicionales en variables locales. Retorna verdadero en el éxito; en caso de fallo, retorna falso y lanza la excepción apropiada.

int PyArg_VaParse(PyObject *args, const char *format, va_list vargs)

> *Part of the Stable ABI.*
> Idéntico a **`PyArg_ParseTuple()`**, excepto que acepta una *va_list* en lugar de un número variable de argumentos .

int PyArg_ParseTupleAndKeywords(PyObject *args, PyObject *kw, const char *format, char *const *keywords, ...)

> *Part of the Stable ABI.*

Parse the parameters of a function that takes both positional and keyword parameters into local variables. The *keywords* argument is a `NULL`-terminated array of keyword parameter names specified as null-terminated ASCII or UTF-8 encoded C strings. Empty names denote positional-only parameters. Returns true on success; on failure, it returns false and raises the appropriate exception.

Nota

The *keywords* parameter declaration is char *const* in C and const char *const* in C++. This can be overridden with the **`PY_CXX_CONST`** macro.

Distinto en la versión 3.6: Soporte agregado para sólo parámetros posicionales.

Distinto en la versión 3.13: The *keywords* parameter has now type char *const* in C and const char *const* in C++, instead of char**. Added support for non-ASCII keyword parameter names.

int PyArg_VaParseTupleAndKeywords(PyObject *args, PyObject *kw, const char *format, char *const *keywords, va_list vargs)

Part of the Stable ABI.
Idéntico a **`PyArg_ParseTupleAndKeywords()`**, excepto que acepta una *va_list* en lugar de un número variable de argumentos.

int PyArg_ValidateKeywordArguments(PyObject*)

Part of the Stable ABI.
Asegúrese de que las claves en el diccionario de argumentos de palabras clave son cadenas. Esto solo es necesario si **`PyArg_ParseTupleAndKeywords()`** no se utiliza, ya que este último ya hace esta comprobación.

Added in version 3.2.

int PyArg_Parse(PyObject *args, const char *format, ...)

Part of the Stable ABI.
Parse the parameter of a function that takes a single positional parameter into a local variable. Returns true on success; on failure, it returns false and raises the appropriate exception.

Example:

```
// Function using METH_O calling convention
static PyObject*
my_function(PyObject *module, PyObject *arg)
{
    int value;
    if (!PyArg_Parse(arg, "i:my_function", &value))
        return NULL;
    }
    // ... use value ...
}
```

int PyArg_UnpackTuple(PyObject *args, const char *name, Py_ssize_t min, Py_ssize_t max, ...)

Part of the Stable ABI.
A simpler form of parameter retrieval which does not use a format string to specify the types of the arguments. Functions which use this method to retrieve their parameters should be declared as **`METH_VARARGS`** in function or method tables. The tuple containing the actual parameters should be passed as *args*; it must actually be a tuple. The length of the tuple must be at least *min* and no more than *max*; *min* and *max* may be equal. Additional arguments must be passed to the function, each of which should be a pointer to a PyObject* variable; these will be filled in with the values from *args*; they will contain borrowed references. The variables which correspond to optional parameters not given by *args* will not be filled in; these should be initialized by the caller. This function returns true on success and false if *args* is not a tuple or contains the wrong number of elements; an exception will be set if there was a failure.

This is an example of the use of this function, taken from the sources for the **`_weakref`** helper module for weak references:

```
static PyObject *
weakref_ref(PyObject *self, PyObject *args)
{
    PyObject *object;
    PyObject *callback = NULL;
    PyObject *result = NULL;

    if (PyArg_UnpackTuple(args, "ref", 1, 2, &objec
        result = PyWeakref_NewRef(object, callback)
    }
    return result;
}
```

La llamada a **`PyArg_UnpackTuple()`** en este ejemplo es completamente equivalente a esta llamada a **`PyArg_ParseTuple()`**:

```
PyArg_ParseTuple(args, "O|O:ref", &object, &callbac
```

PY_CXX_CONST

The value to be inserted, if any, before char *const* in the *keywords* parameter declaration of **`PyArg_ParseTupleAndKeywords()`** and **`PyArg_VaParseTupleAndKeywords()`**. Default empty for C and `const` for C++ (const char *const*). To override, define it to the desired value before including `Python.h`.

Added in version 3.13.

Construyendo valores

PyObject *Py_BuildValue(const char *format, ...)

Return value: New reference. Part of the Stable ABI.
Create a new value based on a format string similar to those accepted by the `PyArg_Parse*` family of functions and a

sequence of values. Returns the value or `NULL` in the case of an error; an exception will be raised if `NULL` is returned.

`Py_BuildValue()` no siempre genera una tupla. Construye una tupla solo si su cadena de formato contiene dos o más unidades de formato. Si la cadena de formato está vacía, retorna `None`; si contiene exactamente una unidad de formato, retorna el objeto que describa esa unidad de formato. Para forzarlo a retornar una tupla de tamaño 0 o uno, paréntesis la cadena de formato.

Cuando los búfer de memoria se pasan como parámetros para suministrar datos para construir objetos, como para los formatos `s` y `s#`, los datos requeridos se copian. Las memorias intermedias proporcionadas por quien llama nunca son referenciadas por los objetos creados por **`Py_BuildValue()`**. En otras palabras, si su código invoca **`malloc()`** y pasa la memoria asignada a **`Py_BuildValue()`**, su código es responsable de llamar a **`free()`** para esa memoria una vez retorna **`Py_BuildValue()`**.

En la siguiente descripción, la cadena de caracteres entre comillas, *así*, es la unidad de formato; la entrada entre paréntesis (redondos) es el tipo de objeto Python que retornará la unidad de formato; y la entrada entre corchetes [cuadrados] es el tipo de los valores C que se pasarán.

Los caracteres espacio, tabulación, dos puntos y coma se ignoran en las cadenas de formato (pero no dentro de las unidades de formato como `s#`). Esto se puede usar para hacer que las cadenas de formato largo sean un poco más legibles.

`s` (**`str`** o `None`) [const char *]

Convierte una cadena de caracteres C terminada en nulo en un objeto Python **`str`** usando la codificación `'utf-8'`. Si el puntero de la cadena de caracteres C es `NULL`, se usa `None`.

`s#` (**`str`** o `None`) [const char *, **`Py_ssize_t`**]

Convierte una cadena de caracteres de C y su longitud en un objeto Python `str` utilizando la codificación `'utf-8'`. Si el puntero de la cadena de caracteres de C es `NULL`, la longitud se ignora y se retorna `None`.

`y` (**`bytes`**) [const char *]
Esto convierte una cadena de caracteres de C en un objeto Python **`bytes`**. Si el puntero de la cadena de caracteres de C es `NULL`, se retorna `None`.

`y#` (**`bytes`**) [const char *, **`Py_ssize_t`**]
Esto convierte una cadena de caracteres de C y sus longitudes en un objeto Python. Si el puntero de la cadena de caracteres de C es `NULL`, se retorna `None`.

`z` (**`str`** o `None`) [const char *]
Igual que `s`.

`z#` (**`str`** o `None`) [const char *, **`Py_ssize_t`**]
Igual que `s#`.

`u` (**`str`**) [const wchar_t *]
Convert a null-terminated **`wchar_t`** buffer of Unicode (UTF-16 or UCS-4) data to a Python Unicode object. If the Unicode buffer pointer is `NULL`, `None` is returned.

`u#` (**`str`**) [const wchar_t *, **`Py_ssize_t`**]
Convierte un búfer de datos Unicode (UTF-16 o UCS-4) y su longitud en un objeto Python Unicode. Si el puntero del búfer Unicode es `NULL`, la longitud se ignora y se retorna `None`.

`U` (**`str`** o `None`) [const char *]
Igual que `s`.

`z#` (**`str`** o `None`) [const char *, **`Py_ssize_t`**]
Igual que `s#`.

`i` (**`int`**) [int]
Convert a plain C int to a Python integer object.

`b` (**`int`**) [char]

Convert a plain C char to a Python integer object.

`h` (**`int`**) [short int]
Convert a plain C short int to a Python integer object.

`l` (**`int`**) [long int]
Convert a C long int to a Python integer object.

`B` (**`int`**) [unsigned char]
Convert a C unsigned char to a Python integer object.

`H` (**`int`**) [unsigned short int]
Convert a C unsigned short int to a Python integer object.

`I` (**`int`**) [unsigned int]
Convert a C unsigned int to a Python integer object.

`k` (**`int`**) [unsigned long]
Convert a C unsigned long to a Python integer object.

`L` (**`int`**) [long long]
Convert a C long long to a Python integer object.

`K` (**`int`**) [unsigned long long]
Convert a C unsigned long long to a Python integer object.

`n` (**`int`**) [**`Py_ssize_t`**]
Convierte un **`Py_ssize_t`** de C a un entero de Python.

`p` (**`bool`**) [int]
Convert a C int to a Python **`bool`** object.

Be aware that this format requires an `int` argument. Unlike most other contexts in C, variadic arguments are not coerced to a suitable type automatically. You can convert another type (for example, a pointer or a float) to a suitable `int` value using `(x) ? 1 : 0` or `!!x`.

Added in version 3.14.

`c` (**`bytes`** de largo 1) [char]

Convert a C int representing a byte to a Python **`bytes`** object of length 1.

`C` (**`str`** de largo 1) [int]

Convert a C int representing a character to Python **`str`** object of length 1.

`d` (**`float`**) [double]

Convert a C double to a Python floating-point number.

`f` (**`float`**) [float]

Convert a C float to a Python floating-point number.

`D` (**`complex`**) [Py_complex *]

Convierte una estructura **`Py_complex`** de C en un número complejo de Python.

`O` (object) [PyObject *]

Pass a Python object untouched but create a new strong reference to it (i.e. its reference count is incremented by one). If the object passed in is a `NULL` pointer, it is assumed that this was caused because the call producing the argument found an error and set an exception. Therefore, **`Py_BuildValue()`** will return `NULL` but won't raise an exception. If no exception has been raised yet, **`SystemError`** is set.

`S` (object) [PyObject *]

Igual que `O`.

`N` (object) [PyObject *]

Same as `O`, except it doesn't create a new strong reference. Useful when the object is created by a call to an object constructor in the argument list.

`O&` (object) [*converter*, *anything*]

Convert *anything* to a Python object through a *converter* function. The function is called with *anything* (which should be compatible with void*) as its argument and should return a «new» Python object, or `NULL` if an error occurred.

`(items)` (**`tuple`**) [*matching-items*]

Convierta una secuencia de valores C en una tupla de Python con el mismo número de elementos.

`[items]` (**`list`**) [*matching-items*]

Convierte una secuencia de valores C en una lista de Python con el mismo número de elementos.

`{items}` (**`dict`**) [*matching-items*]

Convierte una secuencia de valores C en un diccionario Python. Cada par de valores C consecutivos agrega un elemento al diccionario, que sirve como clave y valor, respectivamente.

Si hay un error en la cadena de formato, se establece la excepción **`SystemError`** y se retorna `NULL`.

PyObject *Py_VaBuildValue(const char *format, va_list vargs)

Return value: New reference. Part of the Stable ABI.
Idéntico a **`Py_BuildValue()`**, excepto que acepta una *va_list* en lugar de un número variable de argumentos.

Conversión y formato de cadenas de caracteres

Funciones para conversión de números y salida de cadena de caracteres formateadas.

int PyOS_snprintf(char *str, size_t size, const char *format, ...)

> *Part of the Stable ABI.*
> Salida de no más de *size* bytes a *str* según la cadena de caracteres de formato *format* y los argumentos adicionales. Consulte la página de manual de Unix *snprintf(3)* *[https://manpages.debian.org/snprintf(3)]*.

int PyOS_vsnprintf(char *str, size_t size, const char *format, va_list va)

> *Part of the Stable ABI.*
> Salida de no más de *size* bytes a *str* según la cadena de caracteres de formato *format* y la lista de argumentos variables *va*. Página de manual de Unix *vsnprintf(3)* *[https://manpages.debian.org/vsnprintf(3)]*.

`PyOS_snprintf()` y **`PyOS_vsnprintf()`** envuelven las funciones estándar de la biblioteca C **`snprintf()`** y **`vsnprintf()`**. Su propósito es garantizar un comportamiento consistente en casos de esquina (*corner cases*), que las funciones del Estándar C no hacen.

Las envolturas aseguran que `str[size-1]` sea siempre `'\0'` al retornar. Nunca se escriben más de *size* bytes (incluido el `'\0'` del final) en *str*. Ambas funciones requieren que `str != NULL`, `size > 0`, `format != NULL` y `size < INT_MAX`. Tenga en cuenta que esto significa que no hay equivalente a la expresión `n = snprintf(NULL, 0, ...)` de C99, la cual determinaría el tamaño del búfer necesario.

El valor de retorno (*rv*) para estas funciones debe interpretarse de la siguiente manera:

- Cuando `0 <= rv < size`, la conversión de salida fue exitosa y los caracteres *rv* se escribieron en *str* (excluyendo el byte `'\0'` final en `str[rv]`).
- Cuando `rv >= size`, la conversión de salida se truncó y se habría necesitado un búfer con `rv + 1` bytes para tener éxito. `str[size-1]` es `'\0'` en este caso.
- When `rv < 0`, the output conversion failed and `str[size-1]` is `'\0'` in this case too, but the rest of *str* is undefined. The exact cause of the error depends on the underlying platform.

Las siguientes funciones proporcionan cadenas de caracteres independientes de la configuración regional para numerar las conversiones.

unsigned long PyOS_strtoul(const char *str, char **ptr, int base)

Part of the Stable ABI.
Convierte la parte inicial de la cadena de caracteres en `str` a un valor unsigned long según la `base` dada, que debe estar entre `2` y `36` inclusive, o ser el valor especial `0`.

Se ignoran los espacios en blanco y las mayúsculas y minúsculas. Si `base` es cero se busca un prefijo `0b`, `0o` o `0x` para indicar la base. Si no están, por defecto es `10`. La base debe ser 0 o entre 2 y 36 (ambos inclusive). Si `ptr` no es `NULL`, contendrá un puntero al final del escaneo.

Si el valor convertido queda fuera del rango del tipo de retorno correspondiente, se produce un error de rango (**`errno`** se establece en **`ERANGE`**) y se devuelve **`ULONG_MAX`**. Si no se puede realizar la conversión, se devuelve `0`.

Vea también el manual Unix de *strtoul(3)* *[https://manpages.debian.org/strtoul(3)]*.

Added in version 3.2.

long PyOS_strtol(const char *str, char **ptr, int base)

Part of the Stable ABI.

Convierte la parte inicial de la cadena de caracteres en `str` a un valor long según la `base` dada, que debe estar entre `2` y `36` inclusive, o ser el valor especial `0`.

Igual que **`PyOS_strtoul()`**, pero devuelve un valor long en su lugar y **`LONG_MAX`** en desbordamientos.

Vea también el manual Unix de *strtol(3)* *[https://manpages.debian.org/strtol(3)]*.

Added in version 3.2.

double PyOS_string_to_double(const char *s, char **endptr, PyObject *overflow_exception)

Part of the Stable ABI.

Convierte una cadena de caracteres `s` a un valor double, generando una excepción de Python en caso de fallo. El conjunto de cadenas de caracteres aceptadas corresponde al conjunto de cadenas aceptadas por el constructor de Python **`float()`**, excepto que `s` no debe tener espacios en blanco iniciales o finales. La conversión es independiente de la configuración regional actual.

Si `endptr` es `NULL`, convierte toda la cadena de caracteres. Lanza **`ValueError`** y retorna `-1.0` si la cadena de caracteres no es una representación válida de un número de punto flotante.

Si *endptr* no es `NULL`, convierte la mayor cantidad posible de la cadena de caracteres y configura `*endptr` para que apunte al primer carácter no convertido. Si ningún segmento inicial de la cadena de caracteres es la representación válida de un número de punto flotante, configura `*endptr` para que apunte al comienzo de la cadena de caracteres, lanza ValueError y retorna `-1.0`.

If `s` represents a value that is too large to store in a float (for example, `"1e500"` is such a string on many platforms) then if `overflow_exception` is `NULL` return `Py_INFINITY` (with an appropriate sign) and don't set any exception. Otherwise, `overflow_exception` must point to a Python exception object; raise that exception and return `-1.0`. In both cases, set `*endptr` to point to the first character after the converted value.

Si se produce algún otro error durante la conversión (por ejemplo, un error de falta de memoria), establece la excepción Python adecuada y retorna `-1.0`.

Added in version 3.1.

char *PyOS_double_to_string(double val, char format_code, int precision, int flags, int *ptype)

Part of the Stable ABI.
Convierte un double *val* a una cadena de caracteres usando *format_code*, *precision* y *flags* suministrados.

format_code debe ser uno de `'e'`, `'E'`, `'f'`, `'F'`, `'g'`, `'G'` or `'r'`. Para `'r'`, la *precision* suministrada debe ser 0 y se ignora. El código de formato `'r'` especifica el formato estándar **`repr()`**.

flags can be zero or more of the following values or-ed together:

Py_DTSF_SIGN

Always precede the returned string with a sign character, even if *val* is non-negative.

Py_DTSF_ADD_DOT_0

Ensure that the returned string will not look like an integer.

Py_DTSF_ALT

Apply «alternate» formatting rules. See the documentation for the **`PyOS_snprintf()`** `'#'` specifier for details.

Py_DTSF_NO_NEG_0

Negative zero is converted to positive zero.

Added in version 3.11.

If *ptype* is non-`NULL`, then the value it points to will be set to one of the following constants depending on the type of *val*:

***ptype**	**type of *val***
Py_DTST_FINITE	finite number
Py_DTST_INFINITE	infinite number
Py_DTST_NAN	not a number

El valor de retorno es un puntero a *buffer* con la cadena de caracteres convertida o `NULL` si la conversión falla. La persona que llama es responsable de liberar la cadena de caracteres retornada llamando a **`PyMem_Free()`**.

Added in version 3.1.

int PyOS_mystricmp(const char *str1, const char *str2)

int PyOS_mystrnicmp(const char *str1, const char *str2, Py_ssize_t size)

Part of the Stable ABI.

Case insensitive comparison of strings. These functions work almost identically to **`strcmp()`** and **`strncmp()`** (respectively), except that they ignore the case of ASCII characters.

Return `0` if the strings are equal, a negative value if *str1* sorts lexicographically before *str2*, or a positive value if it sorts after.

In the *str1* or *str2* arguments, a NUL byte marks the end of the string. For **`PyOS_mystrnicmp()`**, the *size* argument gives the maximum size of the string, as if NUL was present at the index given by *size*.

These functions do not use the locale.

int PyOS_stricmp(const char *str1, const char *str2)

int PyOS_strnicmp(const char *str1, const char *str2, Py_ssize_t size)

Case insensitive comparison of strings.

On Windows, these are aliases of **`stricmp()`** and **`strnicmp()`**, respectively.

On other platforms, they are aliases of **`PyOS_mystricmp()`** and **`PyOS_mystrnicmp()`**, respectively.

Character classification and conversion

The following macros provide locale-independent (unlike the C standard library `ctype.h`) character classification and conversion. The argument must be a signed or unsigned char.

Py_ISALNUM(c)

Return true if the character *c* is an alphanumeric character.

Py_ISALPHA(c)

Return true if the character *c* is an alphabetic character (`a-z` and `A-Z`).

Py_ISDIGIT(c)

Return true if the character *c* is a decimal digit (`0-9`).

Py_ISLOWER(c)

Return true if the character *c* is a lowercase ASCII letter (`a-z`).

Py_ISUPPER(c)

Return true if the character *c* is an uppercase ASCII letter (`A-

`Z`).

Py_ISSPACE(c)

Return true if the character *c* is a whitespace character (space, tab, carriage return, newline, vertical tab, or form feed).

Py_ISXDIGIT(c)

Return true if the character *c* is a hexadecimal digit (`0-9`, `a-f`, and `A-F`).

Py_TOLOWER(c)

Return the lowercase equivalent of the character *c*.

Py_TOUPPER(c)

Return the uppercase equivalent of the character *c*.

PyHash API

See also the `PyTypeObject.tp_hash` member and Calculo del hash de tipos numéricos.

type Py_hash_t

Hash value type: signed integer.

Added in version 3.2.

type Py_uhash_t

Hash value type: unsigned integer.

Added in version 3.2.

Py_HASH_ALGORITHM

A numerical value indicating the algorithm for hashing of `str`, `bytes`, and `memoryview`.

The algorithm name is exposed by `sys.hash_info.algorithm`.

Added in version 3.4.

Py_HASH_FNV
Py_HASH_SIPHASH24
Py_HASH_SIPHASH13

Numerical values to compare to `Py_HASH_ALGORITHM` to determine which algorithm is used for hashing. The hash algorithm can be configured via the configure `--with-hash-algorithm` option.

Added in version 3.4: Add `Py_HASH_FNV` and `Py_HASH_SIPHASH24`.

Added in version 3.11: Add `Py_HASH_SIPHASH13`.

Py_HASH_CUTOFF

Buffers of length in range `[1, Py_HASH_CUTOFF)` are hashed using DJBX33A instead of the algorithm described by **`Py_HASH_ALGORITHM`**.

- A **`Py_HASH_CUTOFF`** of 0 disables the optimization.
- **`Py_HASH_CUTOFF`** must be non-negative and less or equal than 7.

32-bit platforms should use a cutoff smaller than 64-bit platforms because it is easier to create colliding strings. A cutoff of 7 on 64-bit platforms and 5 on 32-bit platforms should provide a decent safety margin.

This corresponds to the **`sys.hash_info.cutoff`** constant.

Added in version 3.4.

PyHASH_MODULUS

The Mersenne prime [https://en.wikipedia.org/wiki/Mersenne_prime] `P = 2**n -1`, used for numeric hash scheme.

This corresponds to the **`sys.hash_info.modulus`** constant.

Added in version 3.13.

PyHASH_BITS

The exponent `n` of `P` in **`PyHASH_MODULUS`**.

Added in version 3.13.

PyHASH_MULTIPLIER

Prime multiplier used in string and various other hashes.

Added in version 3.13.

PyHASH_INF

The hash value returned for a positive infinity.

This corresponds to the `sys.hash_info.inf` constant.

Added in version 3.13.

PyHASH_IMAG

The multiplier used for the imaginary part of a complex number.

This corresponds to the `sys.hash_info.imag` constant.

Added in version 3.13.

type PyHash_FuncDef

Hash function definition used by `PyHash_GetFuncDef()`.

Py_hash_t (*const hash)(const void*, Py_ssize_t)

Hash function.

const char *name

Hash function name (UTF-8 encoded string).

This corresponds to the `sys.hash_info.algorithm` constant.

const int hash_bits

Internal size of the hash value in bits.

This corresponds to the `sys.hash_info.hash_bits` constant.

const int seed_bits

Size of seed input in bits.

This corresponds to the `sys.hash_info.seed_bits` constant.

Added in version 3.4.

PyHash_FuncDef *PyHash_GetFuncDef(void)

Get the hash function definition.

Ver también

PEP 456 [https://peps.python.org/pep-0456/] «Secure and interchangeable hash algorithm».

Added in version 3.4.

Py_hash_t Py_HashPointer(const void *ptr)

Hash a pointer value: process the pointer value as an integer (cast it to `uintptr_t` internally). The pointer is not dereferenced.

The function cannot fail: it cannot return `-1`.

Added in version 3.13.

Py_hash_t Py_HashBuffer(const void *ptr, Py_ssize_t len)

Compute and return the hash value of a buffer of *len* bytes starting at address *ptr*. The hash is guaranteed to match that of **`bytes`**, **`memoryview`**, and other built-in objects that implement the buffer protocol.

Use this function to implement hashing for immutable objects whose **`tp_richcompare`** function compares to another object's buffer.

len must be greater than or equal to `0`.

This function always succeeds.

Added in version 3.14.

Py_hash_t PyObject_GenericHash(PyObject *obj)

Generic hashing function that is meant to be put into a type object's `tp_hash` slot. Its result only depends on the object's identity.

Detalles de implementación de CPython: In CPython, it is equivalent to `Py_HashPointer()`.

Added in version 3.13.

Reflexión

PyObject *PyEval_GetBuiltins(void)

Return value: Borrowed reference. Part of the Stable ABI.
Obsoleto desde la versión 3.13: Utilice **`PyEval_GetFrameBuiltins()`** en su lugar.

Retorna un diccionario de los builtins en el marco de ejecución actual, o el intérprete del estado del hilo si no hay ningún marco en ejecución actualmente.

PyObject *PyEval_GetLocals(void)

Return value: Borrowed reference. Part of the Stable ABI.
Obsoleto desde la versión 3.13: Utilice **`PyEval_GetFrameLocals()`** para obtener el mismo comportamiento que llamando a **`locals()`** en código Python, o bien llama a **`PyFrame_GetLocals()`** sobre el resultado de **`PyEval_GetFrame()`** para acceder al atributo **`f_locals`** del frame actualmente en ejecución.

Retorna un diccionario de las variables locales en el marco de ejecución actual, o `NULL` si actualmente no se está ejecutando ningún marco.

Consulte **`locals()`** para obtener más información sobre el mapeo retornado en diferentes ámbitos.

Como esta función retorna una referencia prestada, el diccionario retornado para ámbitos optimizados se almacena en caché en el objeto marco y permanecerá vivo mientras lo haga el marco del objeto. A diferencia de **`PyEval_GetFrameLocals()`** y **`locals()`**, las llamadas posteriores a esta función en el mismo marco actualizarán el contenido del diccionario en caché para reflejar los cambios en el estado de las variables locales en lugar de retornar una nueva instantánea.

Distinto en la versión 3.13: Como parte de **PEP 667** [https://peps.python.org/pep-0667/], **`PyFrame_GetLocals()`**, **`locals()`**, y **`FrameType.f_locals`** ya no utilizan el diccionario de caché compartido. Consulte la entrada What's New para más detalles.

PyObject *PyEval_GetGlobals(void)

Return value: Borrowed reference. Part of the Stable ABI.
Obsoleto desde la versión 3.13: Utilice **`PyEval_GetFrameGlobals()`** en su lugar.

Retorna un diccionario de las variables globales en el marco de ejecución actual, o `NULL` si actualmente no se está ejecutando ningún marco.

PyFrameObject *PyEval_GetFrame(void)

Return value: Borrowed reference. Part of the Stable ABI.
Return the attached thread state's frame, which is `NULL` if no frame is currently executing.

Vea también **`PyThreadState_GetFrame()`**.

PyObject *PyEval_GetFrameBuiltins(void)

Return value: New reference. Part of the Stable ABI since version 3.13.
Retorna un diccionario de los builtins en el marco de ejecución actual, o el intérprete del estado del hilo si no hay ningún marco en ejecución actualmente.

Added in version 3.13.

PyObject *PyEval_GetFrameLocals(void)

Return value: New reference. Part of the Stable ABI since version 3.13.
Retorna un diccionario de las variables locales en el marco de ejecución actual, o `NULL` si no se está ejecutando ningún marco. Equivale a llamar a **`locals()`** en código Python.

Para acceder a `f_locals` en el marco actual sin hacer una captura independiente en ámbitos optimizados, llame a `PyFrame_GetLocals()` sobre el resultado de `PyEval_GetFrame()`.

Added in version 3.13.

PyObject *PyEval_GetFrameGlobals(void)

Return value: New reference. Part of the Stable ABI since version 3.13.
Retorna un diccionario de las variables locales en el marco de ejecución actual, o `NULL` si no se está ejecutando ningún marco. Equivale a llamar a `globals()` en código Python.

Added in version 3.13.

const char *PyEval_GetFuncName(PyObject *func)

Part of the Stable ABI.
Retorna el nombre de *func* si es una función, clase u objeto de instancia; de lo contrario, el nombre del tipo *func*s.

const char *PyEval_GetFuncDesc(PyObject *func)

Part of the Stable ABI.
Retorna una cadena de caracteres de descripción, según el tipo de *func*. Los valores de retorno incluyen «()» para funciones y métodos, «constructor», «instancia» y «objeto». Concatenado con el resultado de `PyEval_GetFuncName()`, el resultado será una descripción de *func*.

Registro de códec y funciones de soporte

int PyCodec_Register(PyObject *search_function)

Part of the Stable ABI.
Registra una nueva función de búsqueda de códec.

As a side effect, this tries to load the **`encodings`** package, if not yet done, to make sure that it is always first in the list of search functions.

int PyCodec_Unregister(PyObject *search_function)

Part of the Stable ABI since version 3.10.
Anula el registro de una función de búsqueda de códecs y borra el caché del registro. Si la función de búsqueda no está registrada, no hace nada. Retorna 0 en caso de éxito. Lanza una excepción y devuelva -1 en caso de error.

Added in version 3.10.

int PyCodec_KnownEncoding(const char *encoding)

Part of the Stable ABI.
Retorna `1` o `0` dependiendo de si hay un códec registrado para el *encoding* dado. Esta función siempre finaliza con éxito.

PyObject *PyCodec_Encode(PyObject *object, const char *encoding, const char *errors)

Return value: New reference. Part of the Stable ABI.
API de codificación genérica basada en códec.

object se pasa a través de la función de codificador encontrada por el *encoding* dado usando el método de manejo de errores definido por *errors*. *errors* pueden ser `NULL` para usar el método predeterminado definido para el códec. Lanza un **`LookupError`** si no se puede encontrar el codificador.

PyObject *PyCodec_Decode(PyObject *object, const char *encoding, const char *errors)

Return value: New reference. Part of the Stable ABI.
API de decodificación basada en códec genérico.

object is passed through the decoder function found for the given *encoding* using the error handling method defined by *errors*. *errors* may be `NULL` to use the default method defined for the codec. Raises a **`LookupError`** if no decoder can be found.

API de búsqueda de códec

En las siguientes funciones, la cadena de caracteres *encoding* se busca convertida a todos los caracteres en minúscula, lo que hace que las codificaciones se busquen a través de este mecanismo sin distinción entre mayúsculas y minúsculas. Si no se encuentra ningún códec, se establece un **`KeyError`** y se retorna `NULL`.

PyObject *PyCodec_Encoder(const char *encoding)

Return value: New reference. Part of the Stable ABI.
Obtiene una función de codificador para el *encoding* dado.

PyObject *PyCodec_Decoder(const char *encoding)

Return value: New reference. Part of the Stable ABI.
Obtiene una función de decodificador para el *encoding* dado.

PyObject *PyCodec_IncrementalEncoder(const char *encoding, const char *errors)

Return value: New reference. Part of the Stable ABI.
Obtiene un objeto **`IncrementalEncoder`** para el *encoding* dada.

PyObject *PyCodec_IncrementalDecoder(const char *encoding, const char *errors)

Return value: New reference. Part of the Stable ABI.
Obtiene un objeto **`IncrementalDecoder`** para el *encoding* dado.

PyObject *PyCodec_StreamReader(const char *encoding, PyObject *stream, const char *errors)

Return value: New reference. Part of the Stable ABI.
Obtiene una función de fábrica **`StreamReader`** para el *encoding* dado.

PyObject *PyCodec_StreamWriter(const char *encoding, PyObject *stream, const char *errors)

Return value: New reference. Part of the Stable ABI.
Obtiene una función de fábrica **`StreamWriter`** por el *encoding* dado.

API de registro para controladores de errores de codificación Unicode

int PyCodec_RegisterError(const char *name, PyObject *error)

Part of the Stable ABI.
Registra la función de devolución de llamada de manejo de errores *error* bajo el nombre *name* dado. Esta función de devolución de llamada será llamada por un códec cuando encuentre caracteres no codificables / bytes no codificables y *name* se especifica como parámetro de error en la llamada a la función de codificación / decodificación.

La devolución de llamada obtiene un único argumento, una instancia de **`UnicodeEncodeError`**, **`UnicodeDecodeError`** o **`UnicodeTranslateError`** que contiene información sobre la secuencia problemática de caracteres o bytes y su desplazamiento en la cadena original (consulte Objetos unicode de excepción para funciones para extraer esta información). La devolución de llamada debe lanzar la excepción dada o retornar una tupla de dos elementos que contiene el reemplazo de la secuencia problemática, y un número entero que proporciona el desplazamiento en la cadena original en la que se debe reanudar la codificación / decodificación.

Retorna `0` en caso de éxito, `-1` en caso de error.

PyObject *PyCodec_LookupError(const char *name)

Return value: New reference. Part of the Stable ABI.
Busca la función de devolución de llamada de manejo de errores registrada con *name*. Como caso especial se puede pasar `NULL`, en cuyo caso se retornará la devolución de llamada de manejo de errores para «estricto».

PyObject *PyCodec_StrictErrors(PyObject *exc)

Return value: Always NULL. Part of the Stable ABI.
Lanza *exc* como una excepción.

PyObject *PyCodec_IgnoreErrors(PyObject *exc)

Return value: New reference. Part of the Stable ABI.
Ignora el error Unicode, omitiendo la entrada defectuosa.

PyObject *PyCodec_ReplaceErrors(PyObject *exc)

Return value: New reference. Part of the Stable ABI.
Reemplaza el error de codificación Unicode con `?` o `U+FFFD`.

PyObject *PyCodec_XMLCharRefReplaceErrors(PyObject *exc)

Return value: New reference. Part of the Stable ABI.
Reemplaza el error de codificación Unicode con referencias de caracteres XML.

PyObject *PyCodec_BackslashReplaceErrors(PyObject *exc)

Return value: New reference. Part of the Stable ABI.
Reemplaza el error de codificación Unicode con escapes de barra invertida (`\x`, `\u` y `\U`).

PyObject *PyCodec_NameReplaceErrors(PyObject *exc)

Return value: New reference. Part of the Stable ABI since version 3.7.
Reemplaza el error de codificación Unicode con escapes `\N{...}`.

Added in version 3.5.

Codec utility variables

const char *Py_hexdigits

A string constant containing the lowercase hexadecimal digits: `"0123456789abcdef"`.

Added in version 3.3.

PyTime C API

Added in version 3.13.

The clock C API provides access to system clocks. It is similar to the Python `time` module.

For C API related to the `datetime` module, see Objetos DateTime.

Types

type PyTime_t

A timestamp or duration in nanoseconds, represented as a signed 64-bit integer.

The reference point for timestamps depends on the clock used. For example, `PyTime_Time()` returns timestamps relative to the UNIX epoch.

The supported range is around [-292.3 years; +292.3 years]. Using the Unix epoch (January 1st, 1970) as reference, the supported date range is around [1677-09-21; 2262-04-11]. The exact limits are exposed as constants:

PyTime_t PyTime_MIN

Minimum value of `PyTime_t`.

PyTime_t PyTime_MAX

Maximum value of `PyTime_t`.

Clock Functions

The following functions take a pointer to a PyTime_t that they set to the value of a particular clock. Details of each clock are given in the documentation of the corresponding Python function.

The functions return `0` on success, or `-1` (with an exception set) on failure.

On integer overflow, they set the **`PyExc_OverflowError`** exception and set `*result` to the value clamped to the `[PyTime_MIN; PyTime_MAX]` range. (On current systems, integer overflows are likely caused by misconfigured system time.)

As any other C API (unless otherwise specified), the functions must be called with an attached thread state.

int PyTime_Monotonic(PyTime_t *result)

Read the monotonic clock. See **`time.monotonic()`** for important details on this clock.

int PyTime_PerfCounter(PyTime_t *result)

Read the performance counter. See **`time.perf_counter()`** for important details on this clock.

int PyTime_Time(PyTime_t *result)

Read the “wall clock” time. See **`time.time()`** for details important on this clock.

Raw Clock Functions

Similar to clock functions, but don’t set an exception on error and don’t require the caller to have an attached thread state.

On success, the functions return `0`.

On failure, they set `*result` to `0` and return `-1`, *without* setting an exception. To get the cause of the error, attach a thread state, and call the regular (non-`Raw`) function. Note that the regular function may succeed after the `Raw` one failed.

int PyTime_MonotonicRaw(PyTime_t *result)

Similar to **`PyTime_Monotonic()`**, but don't set an exception on error and don't require an attached thread state.

int PyTime_PerfCounterRaw(PyTime_t *result)

Similar to **`PyTime_PerfCounter()`**, but don't set an exception on error and don't require an attached thread state.

int PyTime_TimeRaw(PyTime_t *result)

Similar to **`PyTime_Time()`**, but don't set an exception on error and don't require an attached thread state.

Conversion functions

double PyTime_AsSecondsDouble(PyTime_t t)

Convert a timestamp to a number of seconds as a C double.

The function cannot fail, but note that double has limited accuracy for large values.

Soporte para Mapeo Perf

En las plataformas soportadas (en el momento de escribir esto, sólo Linux), el tiempo de ejecución puede aprovechar *perf map files* para hacer que las funciones de Python sean visibles para una herramienta externa de perfiles (como perf [https://perf.wiki.kernel.org/index.php/Main_Page]). Un proceso en ejecución puede crear un fichero en el directorio `/tmp`, que contiene entradas que pueden asignar una sección de código ejecutable a un nombre. Esta interfaz se describe en la documentación de la herramienta Perf de Linux [https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/tools/perf/Documentation/jit-interface.txt].

En Python, estas API auxiliares pueden ser utilizadas por bibliotecas y funciones que dependen de la generación de código de máquina sobre la marcha.

Note that holding an attached thread state is not required for these APIs.

int PyUnstable_PerfMapState_Init(void)

> *This is Unstable API. It may change without warning in minor releases.*

Abre el archivo `/tmp/perf-$pid.map`, a menos que ya esté abierto, y crea un bloqueo para garantizar escrituras seguras para hilos en el archivo (siempre que las escrituras se realicen mediante **`PyUnstable_WritePerfMapEntry()`**). Normalmente, no es necesario llamar a esto explícitamente; solo use **`PyUnstable_WritePerfMapEntry()`** e inicializará el estado en la primera llamada.

Retorna `0` en caso de éxito, `-1` en caso de fallo al crear/abrir el fichero perf map, o `-2` en caso de fallo al crear un bloqueo. Comprueba `errno` para más información sobre la causa de un fallo.

int PyUnstable_WritePerfMapEntry(const void *code_addr, unsigned int code_size, const char *entry_name)

This is Unstable API. It may change without warning in minor releases.

Escribe una única entrada en el fichero `/tmp/perf-$pid.map`. Esta función es segura para hilos. Aquí hay un ejemplo de entrada:

```
# address      size  name
7f3529fcf759 b     py::bar:/run/t.py
```

Llamará a **`PyUnstable_PerfMapState_Init()`** antes de escribir la entrada, si el fichero perf map no está ya abierto. Retorna `0` en caso de éxito, o los mismos códigos de error que **`PyUnstable_PerfMapState_Init()`** en caso de fallo.

void PyUnstable_PerfMapState_Fini(void)

This is Unstable API. It may change without warning in minor releases.

Cierra el fichero perf map abierto por **`PyUnstable_PerfMapState_Init()`**. Esto es llamado por el propio tiempo de ejecución durante el cierre del intérprete. En general, no debería haber una razón para llamar explícitamente a esto, excepto para manejar escenarios específicos como la bifurcación.

Capa de objetos abstractos

Las funciones de este capítulo interactúan con los objetos de Python independientemente de su tipo, o con amplias clases de tipos de objetos (por ejemplo, todos los tipos numéricos o todos los tipos de secuencia). Cuando se usan en tipos de objetos para los que no se aplican, lanzarán una excepción de Python.

No es posible utilizar estas funciones en objetos que no se inicializan correctamente, como un objeto de lista que ha sido creado por `PyList_New()`, pero cuyos elementos no se han establecido en algunos valores no-`NULL` aún.

- Protocolo de objeto
- Protocolo de llamada
    - El protocolo *tp_call*
    - El protocolo vectorcall
        - Control de recursión
        - API de soporte para vectorcall
    - API para invocar objetos
    - API de soporte de llamadas
- Protocolo de números
- Protocolo de secuencia
- Protocolo de mapeo
- Protocolo iterador
- Protocolo búfer
    - Estructura de búfer
    - Tipos de solicitud búfer
        - campos independientes de solicitud
        - formato de sólo lectura
        - formas, *strides*, *suboffsets*

    - solicitudes de contigüidad
    - solicitudes compuestas
- Arreglos complejos
    - Estilo NumPy: forma y *strides*
    - Estilo PIL: forma, *strides* y *suboffsets*
- Funciones relacionadas a búfer

Protocolo de objeto

PyObject *Py_GetConstant(unsigned int constant_id)

Part of the Stable ABI since version 3.13.
Obtiene una referencia fuerte a una constante.

Establece una excepción y retorna `NULL` si *constant_id* es inválido.

constant_id debe ser uno de estos identificadores constantes:

Identificador de constante	Valor	Objeto retornado
Py_CONSTANT_NONE	0	**`None`**
Py_CONSTANT_FALSE	1	**`False`**
Py_CONSTANT_TRUE	2	**`True`**
Py_CONSTANT_ELLIPSIS	3	**`Ellipsis`**
Py_CONSTANT_NOT_IMPLEMENTED	4	**`NotImplemented`**
Py_CONSTANT_ZERO	5	0
Py_CONSTANT_ONE	6	1
Py_CONSTANT_EMPTY_STR	7	''
Py_CONSTANT_EMPTY_BYTES	8	b''
Py_CONSTANT_EMPTY_TUPLE	9	()

Los valores numéricos solo se proporcionan para proyectos que no pueden usar los identificadores constantes.

Added in version 3.13.

En CPython, todas estas constantes son inmortales.

PyObject *Py_GetConstantBorrowed(unsigned int constant_id)

Part of the Stable ABI since version 3.13.
Similar a **`Py_GetConstant()`**, pero retorna una referencia prestada.

Esta función está destinada principalmente para compatibilidad hacia atrás: se recomienda usar **`Py_GetConstant()`** para código nuevo.

La referencia es prestada del intérprete, y es válida hasta la finalización del intérprete.

Added in version 3.13.

PyObject *Py_NotImplemented

El singleton `NotImplemented`, se usa para indicar que una operación no está implementada para la combinación de tipos dada.

Py_RETURN_NOTIMPLEMENTED

Maneja adecuadamente el retorno de **`Py_NotImplemented`** desde una función C (es decir, crea una nueva referencia fuerte a **`NotImplemented`** y lo retorna).

Py_PRINT_RAW

Flag to be used with multiple functions that print the object (like **`PyObject_Print()`** and **`PyFile_WriteObject()`**). If passed, these functions use the **`str()`** of the object instead of the **`repr()`**.

int PyObject_Print(PyObject *o, FILE *fp, int flags)

Imprime un objeto *o*, en el archivo *fp*. Retorna `-1` en caso de error. El argumento de las banderas se usa para habilitar ciertas opciones de impresión. La única opción actualmente admitida es **`Py_PRINT_RAW`**; si se proporciona, se escribe **`str()`** del objeto en lugar de **`repr()`**.

int PyObject_HasAttrWithError(PyObject *o, PyObject *attr_name)

Part of the Stable ABI since version 3.13.
Retorna `1` si *o* tiene el atributo *attr_name*, y `0` en caso contrario. Esto es equivalente a la expresión de Python `hasattr(o, attr_name)`. En caso de error, retorna `-1`.

Added in version 3.13.

int PyObject_HasAttrStringWithError(PyObject *o, const char *attr_name)

Part of the Stable ABI since version 3.13.
Esto es lo mismo que `PyObject_HasAttrWithError()`, pero *attr_name* se especifica como una cadena de bytes codificada en UTF-8 const char*, en lugar de un PyObject*.

Added in version 3.13.

int PyObject_HasAttr(PyObject *o, PyObject *attr_name)

Part of the Stable ABI.
Retorna `1` si *o* tiene el atributo *attr_name*, y `0` en caso contrario. Esta función siempre tiene éxito.

Nota

Exceptions that occur when this calls `__getattr__()` and `__getattribute__()` methods aren't propagated, but instead given to `sys.unraisablehook()`. For proper error handling, use `PyObject_HasAttrWithError()`, `PyObject_GetOptionalAttr()` or `PyObject_GetAttr()` instead.

int PyObject_HasAttrString(PyObject *o, const char *attr_name)

Part of the Stable ABI.
Esto es lo mismo que `PyObject_HasAttr()`, pero *attr_name* se especifica como una cadena de bytes codificada en UTF-8 const char*, en lugar de un PyObject*.

Nota

Las excepciones que ocurren cuando esto llama a los métodos `__getattr__()` y `__getattribute__()` o mientras se crea el objeto `str` temporal se ignoran silenciosamente. Para un manejo adecuado de errores, usa `PyObject_HasAttrStringWithError()`, `PyObject_GetOptionalAttrString()` o

`PyObject_GetAttrString()` en su lugar.

PyObject *PyObject_GetAttr(PyObject *o, PyObject *attr_name)

Return value: New reference. Part of the Stable ABI.
Recupera un atributo llamado *attr_name* del objeto *o*. Retorna el valor del atributo en caso de éxito o `NULL` en caso de error. Este es el equivalente de la expresión de Python `o.attr_name`.

Si el atributo faltante no debe tratarse como un error, puedes usar **`PyObject_GetOptionalAttr()`** en su lugar.

PyObject *PyObject_GetAttrString(PyObject *o, const char *attr_name)

Return value: New reference. Part of the Stable ABI.
Esto es lo mismo que **`PyObject_GetAttr()`**, pero *attr_name* se especifica como una cadena de bytes codificada en UTF-8 const char*, en lugar de un PyObject*.

Si el atributo faltante no debe tratarse como un error, puedes usar **`PyObject_GetOptionalAttrString()`** en su lugar.

int PyObject_GetOptionalAttr(PyObject *obj, PyObject *attr_name, PyObject **result);

Part of the Stable ABI since version 3.13.
Variante de **`PyObject_GetAttr()`** que no lanza **`AttributeError`** si el atributo no se encuentra.

Si se encuentra el atributo, retorna `1` y establece **result* a una nueva referencia fuerte al atributo. Si no se encuentra el atributo, retorna `0` y establece **result* a `NULL`; el **`AttributeError`** se silencia. Si se lanza un error distinto de **`AttributeError`**, retorna `-1` y establece **result* a `NULL`.

Added in version 3.13.

int PyObject_GetOptionalAttrString(PyObject *obj, const char *attr_name, PyObject **result);

Part of the Stable ABI since version 3.13.
Esto es lo mismo que **`PyObject_GetOptionalAttr()`**, pero *attr_name* se especifica como una cadena de bytes codificada en UTF-8 const char*, en lugar de un PyObject*.

Added in version 3.13.

PyObject *PyObject_GenericGetAttr(PyObject *o, PyObject *name)

Return value: New reference. Part of the Stable ABI.
Función *getter* de atributo genérico que debe colocarse en la ranura `tp_getattro` de un objeto tipo. Busca un descriptor en el diccionario de clases en el MRO del objeto, así como un atributo en el objeto **`__dict__`** (si está presente). Como se describe en Implementando descriptores, los descriptores de datos tienen preferencia sobre los atributos de instancia, mientras que los descriptores que no son de datos no lo hacen. De lo contrario, se lanza un **`AttributeError`**.

int PyObject_SetAttr(PyObject *o, PyObject *attr_name, PyObject *v)

Part of the Stable ABI.
Establece el valor del atributo llamado *attr_name*, para el objeto *o*, en el valor *v*. Lanza una excepción y retorna `-1` en caso de falla; retorna `0` en caso de éxito. Este es el equivalente de la declaración de Python `o.attr_name = v`.

Si *v* es `NULL`, el atributo se elimina. Este comportamiento está deprecado en favor de usar **`PyObject_DelAttr()`**, pero por el momento no hay planes de quitarlo.

int PyObject_SetAttrString(PyObject *o, const char *attr_name, PyObject *v)

Part of the Stable ABI.
Esto es lo mismo que **`PyObject_SetAttr()`**, pero *attr_name* se especifica como una cadena de bytes codificada en UTF-8 const char*, en lugar de un PyObject*.

Si *v* es `NULL`, el atributo se elimina, sin embargo, esta característica está deprecada en favor de usar **`PyObject_DelAttrString()`**.

El número de diferentes nombres de atributos pasados a esta función debe mantenerse pequeño, generalmente usando una cadena asignada estáticamente como *attr_name*. Para nombres de atributos que no se conocen en tiempo de compilación, prefiere llamar directamente a **`PyUnicode_FromString()`** y **`PyObject_SetAttr()`**. Para más detalles, consulta **`PyUnicode_InternFromString()`**, que puede usarse internamente para crear un objeto clave.

int PyObject_GenericSetAttr(PyObject *o, PyObject *name, PyObject *value)

Part of the Stable ABI.
Establecimiento de atributo genérico y función de eliminación que está destinada a colocarse en la ranura de un objeto tipo **`tp_setattro`**. Busca un descriptor de datos en el diccionario de clases en el MRO del objeto y, si se encuentra, tiene preferencia sobre la configuración o eliminación del atributo en el diccionario de instancias. De lo contrario, el atributo se establece o elimina en el objeto **`__dict__`** (si está presente). En caso de éxito, se retorna `0`; de lo contrario, se lanza un **`AttributeError`** y se retorna `-1`.

int PyObject_DelAttr(PyObject *o, PyObject *attr_name)

Part of the Stable ABI since version 3.13.
Elimina el atributo llamado *attr_name*, para el objeto *o*. Retorna `-1` en caso de falla. Este es el equivalente de la declaración de Python `del o.attr_name`.

int PyObject_DelAttrString(PyObject *o, const char *attr_name)

Part of the Stable ABI since version 3.13.
Esto es lo mismo que **`PyObject_DelAttr()`**, pero *attr_name* se especifica como una cadena de bytes codificada en UTF-8 const char*, en lugar de un PyObject*.

El número de diferentes nombres de atributos pasados a esta función debe mantenerse pequeño, generalmente usando una cadena asignada estáticamente como *attr_name*. Para nombres de atributos que no se conocen en tiempo de compilación, prefiere llamar directamente a **`PyUnicode_FromString()`** y **`PyObject_DelAttr()`**. Para más detalles, consulta **`PyUnicode_InternFromString()`**, que puede usarse internamente para crear un objeto clave para la búsqueda.

PyObject *PyObject_GenericGetDict(PyObject *o, void *context)

Return value: New reference. Part of the Stable ABI since version 3.10.
Una implementación genérica para obtener un descriptor `__dict__`. Crea el diccionario si es necesario.

Esta función también puede ser llamada para obtener el **`__dict__`** del objeto *o*. Se pasa *context* igual a `NULL` cuando se lo llama. Dado que esta función puede necesitar asignar memoria para el diccionario, puede ser más eficiente llamar a **`PyObject_GetAttr()`** para acceder a un atributo del objeto.

En caso de fallo, retorna `NULL` con una excepción establecida.

Added in version 3.3.

int PyObject_GenericSetDict(PyObject *o, PyObject *value, void *context)

Part of the Stable ABI since version 3.7.
Una implementación genérica para el creador de un descriptor `__dict__`. Esta implementación no permite que se elimine el diccionario.

Added in version 3.3.

PyObject **_PyObject_GetDictPtr(PyObject *obj)

Retorna un puntero al **`__dict__`** del objeto *obj*. Si no hay `__dict__`, retorna `NULL` sin establecer una excepción.

Esta función puede necesitar asignar memoria para el diccionario, por lo que puede ser más eficiente llamar a **`PyObject_GetAttr()`** para acceder a un atributo del objeto.

PyObject *PyObject_RichCompare(PyObject *o1, PyObject *o2, int opid)

Return value: New reference. Part of the Stable ABI.
Compara los valores de *o1* y *o2* utilizando la operación especificada por *opid*, que debe ser uno de **`Py_LT`**, **`Py_LE`**, **`Py_EQ`**, **`Py_NE`**, **`Py_GT`**, o **`Py_GE`**, correspondiente a <, <=, ==, !=, > o >= respectivamente. Este es el equivalente de la expresión de Python `o1 op o2`, donde `op` es el operador correspondiente a *opid*. Retorna el valor de la comparación en caso de éxito o `NULL` en caso de error.

int PyObject_RichCompareBool(PyObject *o1, PyObject *o2, int opid)

Part of the Stable ABI.
Compara los valores de *o1* y *o2* usando la operación especificada por *opid*, como **`PyObject_RichCompare()`**, pero retorna `-1` en caso de error, `0` si el resultado es falso, `1` en caso contrario.

Nota

Si *o1* y *o2* son el mismo objeto, **`PyObject_RichCompareBool()`** siempre retornará `1` para **`Py_EQ`** y `0` para **`Py_NE`**.

PyObject *PyObject_Format(PyObject *obj, PyObject *format_spec)

Part of the Stable ABI.
Formatea *obj* usando *format_spec*. Esto es equivalente a la expresión de Python `format(obj, format_spec)`.

format_spec puede ser `NULL`. En este caso, la llamada es equivalente a `format(obj)`. Retorna la cadena formateada en caso de éxito, `NULL` en caso de fallo.

PyObject *PyObject_Repr(PyObject *o)

Return value: New reference. Part of the Stable ABI.
Calcula una representación de cadena de caracteres del objeto *o*. Retorna la representación de cadena de caracteres en caso de éxito, `NULL` en caso de error. Este es el equivalente de la expresión de Python `repr(o)`. Llamado por la función incorporada **`repr()`**.

If argument is `NULL`, return the string `'<NULL>'`.

Distinto en la versión 3.4: Esta función ahora incluye una afirmación de depuración para ayudar a garantizar que no descarte silenciosamente una excepción activa.

PyObject *PyObject_ASCII(PyObject *o)

Return value: New reference. Part of the Stable ABI.
Como **`PyObject_Repr()`**, calcula una representación de cadena de caracteres del objeto *o*, pero escapa los caracteres no ASCII en la cadena de caracteres retornada por **`PyObject_Repr()`** con `\x`, `\u` o `\U` escapa. Esto genera una cadena de caracteres similar a la que retorna **`PyObject_Repr()`** en Python 2. Llamado por la función incorporada **`ascii()`**.

If argument is `NULL`, return the string `'<NULL>'`.

PyObject *PyObject_Str(PyObject *o)

Return value: New reference. Part of the Stable ABI.
Calcula una representación de cadena de caracteres del objeto *o*. Retorna la representación de cadena de caracteres en caso de éxito, `NULL` en caso de error. Llamado por la función incorporada **`str()`** y, por lo tanto, por la función **`print()`**.

If argument is `NULL`, return the string `'<NULL>'`.

Distinto en la versión 3.4: Esta función ahora incluye una afirmación de depuración para ayudar a garantizar que no descarte silenciosamente una excepción activa.

PyObject *PyObject_Bytes(PyObject *o)

Return value: New reference. Part of the Stable ABI.
Calcula una representación de bytes del objeto *o*. `NULL` se retorna en caso de error y un objeto de bytes en caso de éxito. Esto es equivalente a la expresión de Python `bytes(o)`, cuando *o* no es un número entero. A diferencia de `bytes(o)`, se lanza un TypeError cuando *o* es un entero en lugar de un objeto de bytes con inicialización cero.

If argument is `NULL`, return the **`bytes`** object `b'<NULL>'`.

int PyObject_IsSubclass(PyObject *derived, PyObject *cls)

Part of the Stable ABI.
Retorna `1` si la clase *derived* es idéntica o derivada de la clase *cls*; de lo contrario, retorna `0`. En caso de error, retorna `-1`.

Si *cls* es una tupla, la verificación se realizará con cada entrada en *cls*. El resultado será `1` cuando al menos una de las verificaciones retorne `1`, de lo contrario será `0`.

Si *cls* tiene un método **`__subclasscheck__()`**, se llamará para determinar el estado de la subclase como se describe en **PEP 3119** [https://peps.python.org/pep-3119/]. De lo contrario, *derived* es una subclase de *cls* si es una subclase directa o indirecta, es decir, contenida en **`cls.__mro__`**.

Normalmente, solo los objetos de clase, es decir, instancias de **`type`** o una clase derivada, se consideran clases. Sin embargo, los objetos pueden anular esto al tener un atributo **`__bases__`** (que debe ser una tupla de clases base).

int PyObject_IsInstance(PyObject *inst, PyObject *cls)

Part of the Stable ABI.
Retorna `1` si *inst* es una instancia de la clase *cls* o una subclase de *cls*, o `0` si no. En caso de error, retorna `-1` y

establece una excepción.

Si *cls* es una tupla, la verificación se realizará con cada entrada en *cls*. El resultado será `1` cuando al menos una de las verificaciones retorne `1`, de lo contrario será `0`.

Si *cls* tiene un método **`__instancecheck__()`**, se llamará para determinar el estado de la subclase como se describe en **PEP 3119** [https://peps.python.org/pep-3119/]. De lo contrario, *inst* es una instancia de *cls* si su clase es una subclase de *cls*.

Una instancia *inst* puede anular lo que se considera su clase al tener un atributo **`__class__`**.

Un objeto *cls* puede anular si se considera una clase, y cuáles son sus clases base, al tener un atributo **`__bases__`** (que debe ser una tupla de clases base).

Py_hash_t PyObject_Hash(PyObject *o)

Part of the Stable ABI.
Calcula y retorna el valor hash de un objeto *o*. En caso de fallo, retorna `-1`. Este es el equivalente de la expresión de Python `hash(o)`.

Distinto en la versión 3.2: El tipo de retorno ahora es *Py_hash_t*. Este es un entero con signo del mismo tamaño que **`Py_ssize_t`**.

Py_hash_t PyObject_HashNotImplemented(PyObject *o)

Part of the Stable ABI.
Establece un **`TypeError`** indicando que `type(o)` no es hashable y retorna `-1`. Esta función recibe un tratamiento especial cuando se almacena en una ranura `tp_hash`, lo que permite que un tipo indique explícitamente al intérprete que no es hashable.

int PyObject_IsTrue(PyObject *o)

Part of the Stable ABI.
Retorna `1` si el objeto *o* se considera verdadero y `0` en caso

contrario. Esto es equivalente a la expresión de Python `not not o`. En caso de error, retorna `-1`.

int PyObject_Not(PyObject *o)

Part of the Stable ABI.

Retorna `0` si el objeto *o* se considera verdadero, y `1` de lo contrario. Esto es equivalente a la expresión de Python `not o`. En caso de error, retorna `-1`.

PyObject *PyObject_Type(PyObject *o)

Return value: New reference. Part of the Stable ABI.

Cuando *o* es non-`NULL`, retorna un objeto de tipo correspondiente al tipo de objeto del objeto *o*. En caso de fallo, lanza **`SystemError`** y retorna `NULL`. Esto es equivalente a la expresión de Python `type(o)`. Esta función crea una nueva referencia fuerte al valor de retorno. Realmente no hay razón para usar esta función en lugar de la función **`Py_TYPE()`**, que retorna un puntero de tipo PyTypeObject*, excepto cuando se necesita una nueva referencia fuerte.

int PyObject_TypeCheck(PyObject *o, PyTypeObject *type)

Retorna un valor no-nulo si el objeto *o* es de tipo *type* o un subtipo de *type*, y `0` en cualquier otro caso. Ninguno de los dos parámetros debe ser `NULL`.

Py_ssize_t PyObject_Size(PyObject *o)

Py_ssize_t PyObject_Length(PyObject *o)

Part of the Stable ABI.

Retorna la longitud del objeto *o*. Si el objeto *o* proporciona los protocolos de secuencia y mapeo, se retorna la longitud de la secuencia. En caso de error, se retorna `-1`. Este es el equivalente a la expresión de Python `len(o)`.

Py_ssize_t PyObject_LengthHint(PyObject *o, Py_ssize_t defaultvalue)

Retorna una longitud estimada para el objeto *o*. Primero intenta retornar su longitud real, luego una estimación usando **`__length_hint__()`**, y finalmente retorna el valor predeterminado. En caso de error, retorna `-1`. Este es el equivalente a la expresión de Python `operator.length_hint(o, defaultvalue)`.

Added in version 3.4.

PyObject *PyObject_GetItem(PyObject *o, PyObject *key)

Return value: New reference. Part of the Stable ABI.
Retorna el elemento de *o* correspondiente a la clave *key* del objeto o `NULL` en caso de error. Este es el equivalente de la expresión de Python `o[key]`.

int PyObject_SetItem(PyObject *o, PyObject *key, PyObject *v)

Part of the Stable ABI.
Asigna el objeto *key* al valor *v*. Lanza una excepción y retorna `-1` en caso de error; retorna `0` en caso de éxito. Este es el equivalente de la declaración de Python `o[key] = v`. Esta función *no* roba una referencia a *v*.

int PyObject_DelItem(PyObject *o, PyObject *key)

Part of the Stable ABI.
Elimina la asignación para el objeto *key* del objeto *o*. Retorna `-1` en caso de falla. Esto es equivalente a la declaración de Python `del o[key]`.

int PyObject_DelItemString(PyObject *o, const char *key)

Part of the Stable ABI.
This is the same as **`PyObject_DelItem()`**, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*.

PyObject *PyObject_Dir(PyObject *o)

Return value: New reference. Part of the Stable ABI.
This is equivalent to the Python expression `dir(o)`, returning a (possibly empty) list of strings appropriate for the

object argument, or `NULL` if there was an error. If the argument is `NULL`, this is like the Python `dir()`, returning the names of the current locals; in this case, if no execution frame is active then `NULL` is returned but **`PyErr_Occurred()`** will return false.

PyObject *PyObject_GetIter(PyObject *o)

Return value: New reference. Part of the Stable ABI.
This is equivalent to the Python expression `iter(o)`. It returns a new iterator for the object argument, or the object itself if the object is already an iterator. Raises **`TypeError`** and returns `NULL` if the object cannot be iterated.

PyObject *PyObject_SelfIter(PyObject *obj)

Return value: New reference. Part of the Stable ABI.
This is equivalent to the Python `__iter__(self): return self` method. It is intended for iterator types, to be used in the **`PyTypeObject.tp_iter`** slot.

PyObject *PyObject_GetAIter(PyObject *o)

Return value: New reference. Part of the Stable ABI since version 3.10.
Esto es equivalente a la expresión de Python `aiter(o)`. Toma un objeto **`AsyncIterable`** y retorna un **`AsyncIterator`**. Este es típicamente un nuevo iterador, pero si el argumento es un **`AsyncIterator`**, se retornará a sí mismo. Lanza **`TypeError`** y retorna `NULL` si el objeto no puede ser iterado.

Added in version 3.10.

void *PyObject_GetTypeData(PyObject *o, PyTypeObject *cls)

Part of the Stable ABI since version 3.12.
Obtiene un puntero a datos específicos de subclase reservados para *cls*.

El objeto *o* debe ser una instancia de *cls*, y *cls* debe haberse creado usando **`PyType_Spec.basicsize`** negativo. Python no verifica esto.

En caso de error, establece una excepción y retorna `NULL`.

Added in version 3.12.

Py_ssize_t PyType_GetTypeDataSize(PyTypeObject *cls)

Part of the Stable ABI since version 3.12.
Retorna el tamaño del espacio de memoria de instancia reservado para *cls*, es decir, el tamaño de la memoria que retorna **`PyObject_GetTypeData()`**.

Esto puede ser mayor de lo solicitado usando **`-PyType_Spec.basicsize`**; es seguro usar este tamaño mayor (por ejemplo, con **`memset()`**).

El tipo *cls* **debe** haberse creado usando **`PyType_Spec.basicsize`** negativo. Python no verifica esto.

En caso de error, establece una excepción y retorna un valor negativo.

Added in version 3.12.

void *PyObject_GetItemData(PyObject *o)

Obtiene un puntero a datos por elemento para una clase con **`Py_TPFLAGS_ITEMS_AT_END`**.

En caso de error, establece una excepción y retorna `NULL`. Se lanza **`TypeError`** si *o* no tiene establecido **`Py_TPFLAGS_ITEMS_AT_END`**.

Added in version 3.12.

int PyObject_VisitManagedDict(PyObject *obj, visitproc visit, void *arg)

Visita el diccionario gestionado de *obj*.

Esta función solo debe llamarse en una función de recorrido del tipo que tiene la bandera **`Py_TPFLAGS_MANAGED_DICT`** establecida.

Added in version 3.13.

void PyObject_ClearManagedDict(PyObject *obj)

Limpia el diccionario gestionado de *obj*.

This function must only be called in a clear function of the type which has the **`Py_TPFLAGS_MANAGED_DICT`** flag set.

Added in version 3.13.

int PyUnstable_Object_EnableDeferredRefcount(PyObject *obj)

> *This is Unstable API. It may change without warning in minor releases.*

Enable deferred reference counting [https://peps.python.org/pep-0703/#deferred-reference-counting] on *obj*, if supported by the runtime. In the free-threaded build, this allows the interpreter to avoid reference count adjustments to *obj*, which may improve multi-threaded performance. The tradeoff is that *obj* will only be deallocated by the tracing garbage collector, and not when the interpreter no longer has any references to it.

This function returns `1` if deferred reference counting is enabled on *obj*, and `0` if deferred reference counting is not supported or if the hint was ignored by the interpreter, such as when deferred reference counting is already enabled on *obj*. This function is thread-safe, and cannot fail.

This function does nothing on builds with the GIL enabled, which do not support deferred reference counting. This also does nothing if *obj* is not an object tracked by the garbage collector (see **`gc.is_tracked()`** and **`PyObject_GC_IsTracked()`**).

This function is intended to be used soon after *obj* is created,

by the code that creates it, such as in the object's `tp_new` slot.

Added in version 3.14.

int PyUnstable_Object_IsUniqueReferencedTemporary(PyObject *obj)

> *This is Unstable API. It may change without warning in minor releases.*

Check if *obj* is a unique temporary object. Returns `1` if *obj* is known to be a unique temporary object, and `0` otherwise. This function cannot fail, but the check is conservative, and may return `0` in some cases even if *obj* is a unique temporary object.

If an object is a unique temporary, it is guaranteed that the current code has the only reference to the object. For arguments to C functions, this should be used instead of checking if the reference count is `1`. Starting with Python 3.14, the interpreter internally avoids some reference count modifications when loading objects onto the operands stack by borrowing references when possible, which means that a reference count of `1` by itself does not guarantee that a function argument uniquely referenced.

In the example below, `my_func` is called with a unique temporary object as its argument:

```
my_func([1, 2, 3])
```

In the example below, `my_func` is **not** called with a unique temporary object as its argument, even if its refcount is `1`:

```
my_list = [1, 2, 3]
my_func(my_list)
```

See also the function `Py_REFCNT()`.

Added in version 3.14.

int PyUnstable_IsImmortal(PyObject *obj)

This is Unstable API. It may change without warning in minor releases.

This function returns non-zero if *obj* is immortal, and zero otherwise. This function cannot fail.

Nota

Objects that are immortal in one CPython version are not guaranteed to be immortal in another.

Added in version 3.14.

int PyUnstable_TryIncRef(PyObject *obj)

This is Unstable API. It may change without warning in minor releases.

Increments the reference count of *obj* if it is not zero. Returns `1` if the object's reference count was successfully incremented. Otherwise, this function returns `0`.

`PyUnstable_EnableTryIncRef()` must have been called earlier on *obj* or this function may spuriously return `0` in the free-threaded build.

This function is logically equivalent to the following C code, except that it behaves atomically in the free-threaded build:

```
if (Py_REFCNT(op) > 0) {
   Py_INCREF(op);
   return 1;
}
return 0;
```

This is intended as a building block for managing weak references without the overhead of a Python weak reference object.

Typically, correct use of this function requires support from *obj*'s deallocator (`tp_dealloc`). For example, the following sketch could be adapted to implement a «weakmap» that works like a `WeakValueDictionary` for a specific type:

```
PyMutex mutex;

PyObject *
add_entry(weakmap_key_type *key, PyObject *value)
{
    PyUnstable_EnableTryIncRef(value);
    weakmap_type weakmap = ...;
    PyMutex_Lock(&mutex);
    weakmap_add_entry(weakmap, key, value);
    PyMutex_Unlock(&mutex);
    Py_RETURN_NONE;
}

PyObject *
get_value(weakmap_key_type *key)
{
    weakmap_type weakmap = ...;
    PyMutex_Lock(&mutex);
    PyObject *result = weakmap_find(weakmap, key);
    if (PyUnstable_TryIncRef(result)) {
        // `result` is safe to use
        PyMutex_Unlock(&mutex);
        return result;
    }
    // if we get here, `result` is starting to be g
    // but has not been removed from the weakmap ye
    PyMutex_Unlock(&mutex);
    return NULL;
}
```

```
// tp_dealloc function for weakmap values
void
value_dealloc(PyObject *value)
{
    weakmap_type weakmap = ...;
    PyMutex_Lock(&mutex);
    weakmap_remove_value(weakmap, value);

    ...
    PyMutex_Unlock(&mutex);
}
```

Added in version 3.14.

void PyUnstable_EnableTryIncRef(PyObject *obj)

> *This is Unstable API. It may change without warning in minor releases.*

Enables subsequent uses of `PyUnstable_TryIncRef()` on *obj*. The caller must hold a strong reference to *obj* when calling this.

Added in version 3.14.

int PyUnstable_Object_IsUniquelyReferenced(PyObject *op)

> *This is Unstable API. It may change without warning in minor releases.*

Determine if *op* only has one reference.

On GIL-enabled builds, this function is equivalent to Py_REFCNT(op) == 1.

On a free-threaded build, this checks if *op*'s reference count is equal to one and additionally checks if *op* is only used by this thread. Py_REFCNT(op) == 1 is **not** thread-safe on free-threaded builds; prefer this function.

The caller must hold an attached thread state, despite the fact that this function doesn't call into the Python interpreter. This function cannot fail.

Added in version 3.14.

Protocolo de llamada

CPython admite dos protocolos de llamada diferentes: *tp_call* y vectorcall.

El protocolo *tp_call*

Las instancias de clases que establecen **`tp_call`** son invocables. La firma del slot es:

```
PyObject *tp_call(PyObject *callable, PyObject *args, Py
```

Se realiza una llamada usando una tupla para los argumentos posicionales y un dict para los argumentos de palabras clave, de manera similar a `callable(*args, **kwargs)` en el código Python. *args* debe ser no NULL (use una tupla vacía si no hay argumentos) pero *kwargs* puede ser *NULL* si no hay argumentos de palabra clave.

Esta convención no solo es utilizada por *tp_call*: **`tp_new`** y **`tp_init`** también pasan argumentos de esta manera.

Para llamar a un objeto, use **`PyObject_Call()`** u otra llamada a la API.

El protocolo vectorcall

Added in version 3.9.

El protocolo vectorcall se introdujo en **PEP 590** [https://peps.python.org/pep-0590/] como un protocolo adicional para hacer que las llamadas sean más eficientes.

Como regla general, CPython preferirá el vectorcall para llamadas internas si el invocable lo admite. Sin embargo, esta no es una regla estricta. Además, algunas extensiones de terceros usan *tp_call* directamente (en lugar de usar `PyObject_Call()`). Por lo tanto, una clase que admita vectorcall también debe implementar `tp_call`. Además, el invocable debe comportarse de la misma manera independientemente del protocolo que se utilice. La forma recomendada de lograr esto es configurando `tp_call` en `PyVectorcall_Call()`. Vale la pena repetirlo:

Advertencia

Una clase que admita vectorcall **debe** también implementar `tp_call` con la misma semántica.

Distinto en la versión 3.12: Ahora el indicador `Py_TPFLAGS_HAVE_VECTORCALL` se elimina de una clase cuando se reasigna el método de la clase `__call__()`. (Esto configura internamente solo `tp_call` y, por lo tanto, puede hacer que se comporte de forma diferente a la función vectorcall.) En versiones anteriores de Python, vectorcall solo debería usarse con tipos `immutables` o estáticos.

Una clase no debería implementar vectorcall si eso fuera más lento que *tp_call*. Por ejemplo, si el destinatario de la llamada necesita convertir los argumentos a una tupla args y un dict kwargs de todos modos, entonces no tiene sentido implementar vectorcall.

Las clases pueden implementar el protocolo vectorcall habilitando el indicador `Py_TPFLAGS_HAVE_VECTORCALL` y la configuración `tp_vectorcall_offset` al desplazamiento dentro de la estructura del objeto donde aparece un *vectorcallfunc*. Este es un puntero a una función con la siguiente firma:

typedef PyObject *(*vectorcallfunc)(PyObject *callable, PyObject *const *args, size_t nargsf, PyObject *kwnames)

Part of the Stable ABI since version 3.12.

- *callable* es el objeto siendo invocado.
- *args* es un arreglo en C que consta de los argumentos posicionales seguidos por el
  valores de los argumentos de la palabra clave. Puede

ser *NULL* si no hay argumentos.

- *nargsf* es el número de argumentos posicionales más posiblemente el
  indicador **`PY_VECTORCALL_ARGUMENTS_OFFSET`**. Para obtener el número real de argumentos posicionales de *nargsf*, use **`PyVectorcall_NARGS()`**.

- *kwnames* es una tupla que contiene los nombres de los argumentos de la palabra clave;
  en otras palabras, las claves del diccionario kwargs. Estos nombres deben ser cadenas (instancias de `str` o una subclase) y deben ser únicos. Si no hay argumentos de palabras clave, entonces *kwnames* puede ser *NULL*.

PY_VECTORCALL_ARGUMENTS_OFFSET

Part of the Stable ABI since version 3.12.
Si este indicador se establece en un argumento vectorcall *nargsf*, el destinatario de la llamada puede cambiar temporalmente `args[-1]`. En otras palabras, *args* apunta al argumento 1 (no 0) en el vector asignado. El destinatario de la llamada debe restaurar el valor de `args[-1]` antes de regresar.

Para **`PyObject_VectorcallMethod()`**, este indicador significa en cambio que `args[0]` puede cambiarse.

Siempre que puedan hacerlo de forma económica (sin asignación adicional), se anima a las personas que llaman a utilizar **`PY_VECTORCALL_ARGUMENTS_OFFSET`**. Si lo hace, permitirá que las personas que llaman, como los métodos enlazados, realicen sus llamadas posteriores (que incluyen un argumento *self* antepuesto) de manera muy eficiente.

Added in version 3.8.

Para llamar a un objeto que implementa vectorcall, use una función call API como con cualquier otro invocable. **`PyObject_Vectorcall()`** normalmente será más eficiente.

Control de recursión

Cuando se usa *tp_call*, los destinatarios no necesitan preocuparse por recursividad: CPython usa **`Py_EnterRecursiveCall()`** y **`Py_LeaveRecursiveCall()`** para llamadas realizadas usando *tp_call*.

Por eficiencia, este no es el caso de las llamadas realizadas mediante vectorcall: el destinatario de la llamada debe utilizar *Py_EnterRecursiveCall* y *Py_LeaveRecursiveCall* si es necesario.

API de soporte para vectorcall

Py_ssize_t PyVectorcall_NARGS(size_t nargsf)

Part of the Stable ABI since version 3.12.
Dado un argumento vectorcall *nargsf*, retorna el número real de argumentos. Actualmente equivalente a:

```
(Py_ssize_t)(nargsf & ~PY_VECTORCALL_ARGUMENTS_OFFS
```

Sin embargo, la función `PyVectorcall_NARGS` debe usarse para permitir futuras extensiones.

Added in version 3.8.

vectorcallfunc PyVectorcall_Function(PyObject *op)

Si *op* no admite el protocolo vectorcall (ya sea porque el tipo no lo hace o porque la instancia específica no lo hace), retorna *NULL*. De lo contrario, retorna el puntero de la función vectorcall almacenado en *op*. Esta función nunca lanza una excepción.

Esto es principalmente útil para verificar si *op* admite vectorcall, lo cual se puede hacer marcando `PyVectorcall_Function(op) != NULL`.

Added in version 3.9.

PyObject *PyVectorcall_Call(PyObject *callable, PyObject *tuple, PyObject *dict)

Part of the Stable ABI since version 3.12.

Llama a la `vectorcallfunc` de *callable* con argumentos posicionales y de palabras clave dados en una tupla y dict, respectivamente.

Esta es una función especializada, destinada a colocarse en el slot `tp_call` o usarse en una implementación de `tp_call`. No comprueba el indicador `Py_TPFLAGS_HAVE_VECTORCALL` y no vuelve a `tp_call`.

Added in version 3.8.

API para invocar objetos

Hay varias funciones disponibles para llamar a un objeto Python. Cada una convierte sus argumentos a una convención respaldada por el objeto llamado, ya sea *tp_call* o vectorcall. Para realizar la menor conversión posible, elija la que mejor se adapte al formato de datos que tiene disponible.

La siguiente tabla resume las funciones disponibles; consulte la documentación individual para obtener más detalles.

Función	**invocable**	**args**	**kwargs**
`PyObject_Call()`	`PyObject *`	tupla	dict/`NULL`
`PyObject_CallNoArgs()`	`PyObject *`	---	---
`PyObject_CallOneArg()`	`PyObject *`	1 objeto	---
`PyObject_CallObject()`	`PyObject *`	tupla/`NULL`	---
`PyObject_CallFunction()`	`PyObject *`	formato	---
`PyObject_CallMethod()`	obj + `char*`	formato	---
`PyObject_CallFunctionObjArgs()`	`PyObject *`	variadica	---
`PyObject_CallMethodObjArgs()`	obj + nombre	variadica	---
`PyObject_CallMethodNoArgs()`	obj + nombre	---	---
`PyObject_CallMethodOneArg()`	obj + nombre	1 objeto	---
`PyObject_Vectorcall()`	`PyObject *`	vectorcall	vectorcall
`PyObject_VectorcallDict()`	`PyObject *`	vectorcall	dict/`NULL`
`PyObject_VectorcallMethod()`	arg + nombre	vectorcall	vectorcall

PyObject *PyObject_Call(PyObject *callable, PyObject *args, PyObject *kwargs)

Return value: New reference. Part of the Stable ABI.
Llama a un objeto de Python invocable *callable*, con argumentos dados por la tupla *args*, y argumentos con nombre dados por el diccionario *kwargs*.

args no debe ser *NULL*; use una tupla vacía si no se necesitan argumentos. Si no se necesitan argumentos con nombre, *kwargs* puede ser *NULL*.

Retorna el resultado de la llamada en caso de éxito o lanza una excepción y retorna *NULL* en caso de error.

Este es el equivalente de la expresión de Python: `callable(*args, **kwargs)`.

PyObject *PyObject_CallNoArgs(PyObject *callable)

Return value: New reference. Part of the Stable ABI since version 3.10.
Llama a un objeto de Python invocable *callable* sin ningún argumento. Es la forma más eficiente de llamar a un objeto Python invocable sin ningún argumento.

Retorna el resultado de la llamada en caso de éxito o lanza una excepción y retorna *NULL* en caso de error.

Added in version 3.9.

PyObject *PyObject_CallOneArg(PyObject *callable, PyObject *arg)

Return value: New reference.
Llama a un objeto de Python invocable *callable* con exactamente 1 argumento posicional *arg* y sin argumentos de palabra clave.

Retorna el resultado de la llamada en caso de éxito o lanza una excepción y retorna *NULL* en caso de error.

Added in version 3.9.

PyObject *PyObject_CallObject(PyObject *callable, PyObject *args)

Return value: New reference. Part of the Stable ABI.
Llama a un objeto de Python invocable *callable*, con argumentos dados por la tupla *args*. Si no se necesitan argumentos, entonces *args* puede ser *NULL*.

Retorna el resultado de la llamada en caso de éxito o lanza una excepción y retorna *NULL* en caso de error.

Este es el equivalente de la expresión de Python: `callable(*args)`.

PyObject *PyObject_CallFunction(PyObject *callable, const char *format, ...)

Return value: New reference. Part of the Stable ABI.
Llama a un objeto de Python invocable *callable*, con un número variable de argumentos C. Los argumentos de C se describen usando una cadena de caracteres de formato de estilo **`Py_BuildValue()`**. El formato puede ser *NULL*, lo que indica que no se proporcionan argumentos.

Retorna el resultado de la llamada en caso de éxito o lanza una excepción y retorna *NULL* en caso de error.

Este es el equivalente de la expresión de Python: `callable(*args)`.

Tenga en cuenta que si solo pasa PyObject* args, **`PyObject_CallFunctionObjArgs()`** es una alternativa más rápida.

Distinto en la versión 3.4: El tipo de *format* se cambió desde `char *`.

PyObject *PyObject_CallMethod(PyObject *obj, const char *name, const char *format, ...)

Return value: New reference. Part of the Stable ABI.
Llama al método llamado *name* del objeto *obj* con un número variable de argumentos en C. Los argumentos de C se

describen mediante una cadena de formato **`Py_BuildValue()`** que debería producir una tupla.

El formato puede ser *NULL*, lo que indica que no se proporcionan argumentos.

Retorna el resultado de la llamada en caso de éxito o lanza una excepción y retorna *NULL* en caso de error.

Este es el equivalente de la expresión de Python: `obj.name(arg1, arg2, ...)`.

Tenga en cuenta que si solo pasa PyObject* args, **`PyObject_CallMethodObjArgs()`** es una alternativa más rápida.

Distinto en la versión 3.4: Los tipos de *name* y *format* se cambiaron desde `char *`.

PyObject *PyObject_CallFunctionObjArgs(PyObject *callable, ...)

Return value: New reference. Part of the Stable ABI.
Llama a un objeto de Python invocable *callable*, con un número variable de argumentos PyObject*. Los argumentos se proporcionan como un número variable de parámetros seguidos de *NULL*.

Retorna el resultado de la llamada en caso de éxito o lanza una excepción y retorna *NULL* en caso de error.

Este es el equivalente de la expresión de Python: `callable(arg1, arg2, ...)`.

PyObject *PyObject_CallMethodObjArgs(PyObject *obj, PyObject *name, ...)

Return value: New reference. Part of the Stable ABI.
Llama a un método del objeto de Python *obj*, donde el nombre del método se proporciona como un objeto de cadena de caracteres de Python en *name*. Se llama con un número variable de argumentos PyObject*. Los argumentos se proporcionan como un número variable de parámetros

seguidos de *NULL*.

Retorna el resultado de la llamada en caso de éxito o lanza una excepción y retorna *NULL* en caso de error.

PyObject *PyObject_CallMethodNoArgs(PyObject *obj, PyObject *name)

Llama a un método del objeto de Python *obj* sin argumentos, donde el nombre del método se da como un objeto de cadena de caracteres de Python en *name*.

Retorna el resultado de la llamada en caso de éxito o lanza una excepción y retorna *NULL* en caso de error.

Added in version 3.9.

PyObject *PyObject_CallMethodOneArg(PyObject *obj, PyObject *name, PyObject *arg)

Llama a un método del objeto de Python *obj* con un único argumento posicional *arg*, donde el nombre del método se proporciona como un objeto de cadena de caracteres de Python en *name*.

Retorna el resultado de la llamada en caso de éxito o lanza una excepción y retorna *NULL* en caso de error.

Added in version 3.9.

PyObject *PyObject_Vectorcall(PyObject *callable, PyObject *const *args, size_t nargsf, PyObject *kwnames)

Part of the Stable ABI since version 3.12.
Llama a un objeto de Python invocable *callable*. Los argumentos son los mismos que para **`vectorcallfunc`**. Si *callable* admite vectorcall, esto llama directamente a la función vectorcall almacenada en *callable*.

Retorna el resultado de la llamada en caso de éxito o lanza una excepción y retorna *NULL* en caso de error.

Added in version 3.8: as `_PyObject_Vectorcall`

Distinto en la versión 3.9: Renamed to the current name, without the leading underscore. The old provisional name is soft deprecated.

PyObject *PyObject_VectorcallDict(PyObject *callable, PyObject *const *args, size_t nargsf, PyObject *kwdict)

Llamada *invocable* con argumentos posicionales pasados exactamente como en el protocolo vectorcall, pero con argumentos de palabras clave pasados como un diccionario *kwdict*. El arreglo *args* contiene solo los argumentos posicionales.

Independientemente del protocolo que se utilice internamente, es necesario realizar una conversión de argumentos. Por lo tanto, esta función solo debe usarse si la persona que llama ya tiene un diccionario listo para usar para los argumentos de palabras clave, pero no una tupla para los argumentos posicionales.

Added in version 3.9.

PyObject *PyObject_VectorcallMethod(PyObject *name, PyObject *const *args, size_t nargsf, PyObject *kwnames)

Part of the Stable ABI since version 3.12.
Llama a un método usando la convención de llamada vectorcall. El nombre del método se proporciona como una cadena de caracteres de Python *name*. El objeto cuyo método se llama es *args[0]*, y el arreglo *args* que comienza en *args [1]* representa los argumentos de la llamada. Debe haber al menos un argumento posicional. *nargsf* es el número de argumentos posicionales que incluyen *args [0]*, más **`PY_VECTORCALL_ARGUMENTS_OFFSET`** si el valor de `args[0]` puede cambiarse temporalmente. Los argumentos de palabras clave se pueden pasar como en **`PyObject_Vectorcall()`**.

Si el objeto tiene la característica

`Py_TPFLAGS_METHOD_DESCRIPTOR`, esto llamará al objeto de método independiente con el vector *args* completo como argumentos.

Retorna el resultado de la llamada en caso de éxito o lanza una excepción y retorna *NULL* en caso de error.

Added in version 3.9.

API de soporte de llamadas

int PyCallable_Check(PyObject *o)

Part of the Stable ABI.
Determina si el objeto *o* es invocable. Retorna `1` si el objeto es invocable y `0` en caso contrario. Esta función siempre finaliza con éxito.

Protocolo de números

int PyNumber_Check(PyObject *o)

Part of the Stable ABI.
Retorna `1` si el objeto *o* proporciona protocolos numéricos, y falso en caso contrario. Esta función siempre finaliza con éxito.

Distinto en la versión 3.8: Retorna `1` si *o* es un índice entero.

PyObject *PyNumber_Add(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el resultado de agregar *o1* y *o2*, o `NULL` en caso de falla. Este es el equivalente de la expresión de Python `o1 + o2`.

PyObject *PyNumber_Subtract(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el resultado de restar *o2* de *o1*, o `NULL` en caso de falla. Este es el equivalente de la expresión de Python `o1 - o2`.

PyObject *PyNumber_Multiply(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el resultado de multiplicar *o1* y *o2*, o `NULL` en caso de error. Este es el equivalente de la expresión de Python `o1 * o2`.

PyObject *PyNumber_MatrixMultiply(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI since version 3.7.
Retorna el resultado de la multiplicación de matrices en *o1* y *o2*, o `NULL` en caso de falla. Este es el equivalente de la

expresión de Python `o1 @ o2`.

Added in version 3.5.

PyObject *PyNumber_FloorDivide(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el cociente de la división entera a la baja entre *o1* y *o2*, o `NULL` en caso de error. Este es el equivalente de la expresión de Python `o1 // o2`.

PyObject *PyNumber_TrueDivide(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna una aproximación razonable para el valor matemático de *o1* dividido por *o2* o `NULL` en caso de falla. El valor de retorno es «aproximado» porque los números binarios de punto flotante son aproximados; No es posible representar todos los números reales en base dos. Esta función puede retornar un valor de punto flotante cuando se pasan dos enteros. Es equivalente a la expresión de Python `o1 / o2`.

PyObject *PyNumber_Remainder(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el resto de la división entera a la baja entre *o1* y *o2*, o `NULL` en caso de error. Este es el equivalente de la expresión de Python `o1 % o2`.

PyObject *PyNumber_Divmod(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Vea la función incorporada **`divmod()`**. Retorna `NULL` en caso de falla. Este es el equivalente de la expresión de Python `divmod (o1, o2)`.

PyObject *PyNumber_Power(PyObject *o1, PyObject *o2, PyObject *o3)

Return value: New reference. Part of the Stable ABI.
Consulte la función incorporada **`pow()`**. Retorna `NULL` en

caso de falla. Este es el equivalente de la expresión de Python `pow(o1, o2, o3)`, donde *o3* es opcional. Si se ignora *o3*, pase **`Py_None`** en su lugar (pasar `NULL` por *o3* provocaría un acceso ilegal a la memoria).

PyObject *PyNumber_Negative(PyObject *o)

Return value: New reference. Part of the Stable ABI.
Retorna la negación de *o* en caso de éxito o `NULL` en caso de error. Este es el equivalente de la expresión de Python `-o`.

PyObject *PyNumber_Positive(PyObject *o)

Return value: New reference. Part of the Stable ABI.
Retorna *o* en caso de éxito o `NULL` en caso de error. Este es el equivalente de la expresión de Python `+o`.

PyObject *PyNumber_Absolute(PyObject *o)

Return value: New reference. Part of the Stable ABI.
Retorna el valor absoluto de *o* o `NULL` en caso de error. Este es el equivalente de la expresión de Python `abs(o)`.

PyObject *PyNumber_Invert(PyObject *o)

Return value: New reference. Part of the Stable ABI.
Retorna la negación bit a bit de *o* en caso de éxito o `NULL` en caso de error. Este es el equivalente de la expresión de Python `~o`.

PyObject *PyNumber_Lshift(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el resultado del desplazamiento a la izquierda *o1* por *o2* en caso de éxito o `NULL` en caso de error. Este es el equivalente de la expresión de Python `o1 << o2`.

PyObject *PyNumber_Rshift(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el resultado del desplazamiento a la derecha *o1* por *o2* en caso de éxito o `NULL` en caso de error. Este es el equivalente de la expresión de Python `o1 >> o2`.

PyObject *PyNumber_And(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el «bit a bit y» (*bitwise and*) de *o1* y *o2* en caso de éxito y `NULL` en caso de error. Este es el equivalente de la expresión de Python `o1 & o2`.

PyObject *PyNumber_Xor(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el «bit a bit o exclusivo» (*bitwise exclusive or*) de *o1* por *o2* en caso de éxito, o `NULL` en caso de error. Este es el equivalente de la expresión de Python `o1 ^ o2`.

PyObject *PyNumber_Or(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el «bit a bit o» (*bitwise or*) de *o1* y *o2* en caso de éxito, o `NULL` en caso de error. Este es el equivalente de la expresión de Python `o1 | o2`.

PyObject *PyNumber_InPlaceAdd(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el resultado de agregar *o1* y *o2*, o `NULL` en caso de falla. La operación se realiza en su lugar (*in-place*) cuando *o1* lo admite. Este es el equivalente de la declaración de Python `o1 += o2`.

PyObject *PyNumber_InPlaceSubtract(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el resultado de restar *o2* de *o1*, o `NULL` en caso de falla. La operación se realiza en su lugar (*in-place*) cuando *o1* lo admite. Este es el equivalente de la declaración de Python `o1 -= o2`.

PyObject *PyNumber_InPlaceMultiply(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el resultado de multiplicar *o1* y *o2*, o `NULL` en caso de error. La operación se realiza en su lugar (*in-place*) cuando *o1* lo admite. Este es el equivalente de la declaración de

Python `o1 *= o2`.

PyObject *PyNumber_InPlaceMatrixMultiply(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI since version 3.7.
Retorna el resultado de la multiplicación de matrices en *o1* y *o2*, o `NULL` en caso de falla. La operación se realiza en su lugar (*in-place*) cuando *o1* lo admite. Este es el equivalente de la declaración de Python `o1 @= o2`.

Added in version 3.5.

PyObject *PyNumber_InPlaceFloorDivide(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el cociente de la división entera a la baja entre *o1* y *o2*, o `NULL` en caso de error. La operación se realiza in situ (*in-place*) cuando *o1* lo admite. Es el equivalente de la sentencia de Python `o1 //= o2`.

PyObject *PyNumber_InPlaceTrueDivide(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna una aproximación razonable para el valor matemático de *o1* dividido por *o2* o `NULL` en caso de falla. El valor de retorno es «aproximado» porque los números binarios de coma flotante son aproximados; No es posible representar todos los números reales en base dos. Esta función puede retornar un valor de punto flotante cuando se pasan dos enteros. La operación se realiza en su lugar (*in-place*) cuando *o1* lo admite.

PyObject *PyNumber_InPlaceRemainder(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el resto de dividir *o1* entre *o2* o `NULL` en caso de

error. La operación se realiza en su lugar (*in-place*) cuando *o1* lo admite. Este es el equivalente de la declaración de Python `o1%= o2`.

PyObject *PyNumber_InPlacePower(PyObject *o1, PyObject *o2, PyObject *o3)

Return value: New reference. Part of the Stable ABI.
Consulte la función incorporada **`pow()`**. Retorna `NULL` en caso de falla. La operación se realiza en su lugar (*in-place*) cuando *o1* lo admite. Este es el equivalente de la declaración de Python `o1 **= o2` cuando *o3* es **`Py_None`**, o una variante en su lugar (*in-place*) de `pow (o1, o2, o3)` de lo contrario. Si se ignora *o3*, pase **`Py_None`** en su lugar (pasar `NULL` para *o3* provocaría un acceso ilegal a la memoria).

PyObject *PyNumber_InPlaceLshift(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el resultado del desplazamiento a la izquierda *o1* por *o2* en caso de éxito o `NULL` en caso de error. La operación se realiza en su sitio (*in-place*) cuando *o1* lo admite. Este es el equivalente de la declaración de Python `o1 <<= o2`.

PyObject *PyNumber_InPlaceRshift(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el resultado del desplazamiento a la derecha *o1* por *o2* en caso de éxito o `NULL` en caso de error. La operación se realiza en su lugar (*in-place*) cuando *o1* lo admite. Este es el equivalente de la declaración de Python `o1 >>= o2`.

PyObject *PyNumber_InPlaceAnd(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el «bit a bit y» (*bitwise and*) de *o1* y *o2* en caso de éxito y `NULL` en caso de error. La operación se realiza en su lugar (*in-place*) cuando *o1* lo admite. Este es el equivalente de la declaración de Python `o1 &= o2`.

PyObject *PyNumber_InPlaceXor(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el «bit a bit o exclusivo» (*bitwise exclusive or*) de *o1* por *o2* en caso de éxito, o `NULL` en caso de error. La operación se realiza en su lugar (*in-place*) cuando *o1* lo admite. Este es el equivalente de la declaración de Python `o1 ^= o2`.

PyObject *PyNumber_InPlaceOr(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna el «bit a bit o» (*bitwise or*) de *o1* y *o2* en caso de éxito, o `NULL` en caso de error. La operación se realiza en su lugar *in-place* cuando *o1* lo admite. Este es el equivalente de la declaración de Python `o1 |= o2`.

PyObject *PyNumber_Long(PyObject *o)

Return value: New reference. Part of the Stable ABI.
Retorna el *o* convertido a un objeto entero en caso de éxito, o `NULL` en caso de error. Este es el equivalente de la expresión de Python `int(o)`.

PyObject *PyNumber_Float(PyObject *o)

Return value: New reference. Part of the Stable ABI.
Retorna el *o* convertido a un objeto flotante en caso de éxito o `NULL` en caso de error. Este es el equivalente de la expresión de Python `float(o)`.

PyObject *PyNumber_Index(PyObject *o)

Return value: New reference. Part of the Stable ABI.
Retorna el *o* convertido aun entero de Python (*int*) en caso de éxito o `NULL` con una excepción **`TypeError`** lanzada en caso de error.

Distinto en la versión 3.10: El resultado siempre tiene el tipo exacto **`int`**. Anteriormente, el resultado podía ser una instancia de una subclase de `int`.

PyObject *PyNumber_ToBase(PyObject *n, int base)

Return value: New reference. Part of the Stable ABI.
Retorna el entero *n* convertido a base *base* como una cadena de caracteres. El argumento *base* debe ser uno de 2, 8, 10 o 16. Para la base 2, 8 o 16, la cadena retornada está prefijada con un marcador base de `'0b'`", `'0o'` o `'0x'`, respectivamente. Si *n* no es un entero (*int*) Python, primero se convierte con **`PyNumber_Index()`**.

Py_ssize_t PyNumber_AsSsize_t(PyObject *o, PyObject *exc)

Part of the Stable ABI.
Retorna *o* convertido a un valor **`Py_ssize_t`** si *o* puede interpretarse como un entero. Si la llamada falla, se lanza una excepción y se retorna `-1`.

Si *o* puede convertirse a un entero de Python pero el intento de convertirlo a un valor **`Py_ssize_t`** lanzaría un **`OverflowError`**, entonces el argumento *exc* es el tipo de excepción que se lanzará (normalmente **`IndexError`** o **`OverflowError`**). Si *exc* es `NULL`, la excepción se borra y el valor se recorta a `PY_SSIZE_T_MIN` para un entero negativo o a `PY_SSIZE_T_MAX` para un entero positivo.

int PyIndex_Check(PyObject *o)

Part of the Stable ABI since version 3.8.
Retorna `1` si *o* es un entero índice (tiene la ranura `nb_index` de la estructura `tp_as_number` rellenada), y `0` en caso contrario. Esta función siempre finaliza con éxito.

Protocolo de secuencia

int PySequence_Check(PyObject *o)

Part of the Stable ABI.
Return `1` if the object provides the sequence protocol, and `0` otherwise. Note that it returns `1` for Python classes with a **`__getitem__()`** method, unless they are **`dict`** subclasses, since in general it is impossible to determine what type of keys the class supports. This function always succeeds.

Py_ssize_t PySequence_Size(PyObject *o)
Py_ssize_t PySequence_Length(PyObject *o)

Part of the Stable ABI.
Retorna el número de objetos en secuencia *o* en caso de éxito y `-1` en caso de error. Esto es equivalente a la expresión de Python `len(o)`.

PyObject *PySequence_Concat(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna la concatenación de *o1* y *o2* en caso de éxito, y `NULL` en caso de error. Este es el equivalente de la expresión de Python `o1+o2`.

PyObject *PySequence_Repeat(PyObject *o, Py_ssize_t count)

Return value: New reference. Part of the Stable ABI.
Retorna el resultado de repetir el objeto de secuencia *o count* veces, o `NULL` en caso de falla. Este es el equivalente de la expresión de Python `o*count`.

PyObject *PySequence_InPlaceConcat(PyObject *o1, PyObject *o2)

Return value: New reference. Part of the Stable ABI.
Retorna la concatenación de *o1* y *o2* en caso de éxito, y `NULL` en caso de error. La operación se realiza en su lugar *in-place*

cuando *o1* lo admite. Este es el equivalente de la expresión de Python `o1+=o2`.

PyObject *PySequence_InPlaceRepeat(PyObject *o, Py_ssize_t count)

Return value: New reference. Part of the Stable ABI.
Retorna el resultado de repetir el objeto de secuencia *o* *count* veces, o `NULL` en caso de falla. La operación se realiza en su lugar (*in-place*) cuando *o* lo admite. Este es el equivalente de la expresión de Python `o*=count`.

PyObject *PySequence_GetItem(PyObject *o, Py_ssize_t i)

Return value: New reference. Part of the Stable ABI.
Retorna el elemento *i*-ésimo de *o* o `NULL` en caso de error. Este es el equivalente de la expresión de Python `o[i]`.

PyObject *PySequence_GetSlice(PyObject *o, Py_ssize_t i1, Py_ssize_t i2)

Return value: New reference. Part of the Stable ABI.
Retorna la rebanada del objeto secuencia *o* entre *i1* y *i2*, o `NULL` en caso de error. Este es el equivalente de la expresión de Python `o[i1:i2]`.

int PySequence_SetItem(PyObject *o, Py_ssize_t i, PyObject *v)

Part of the Stable ABI.
Asigna el objeto *v* al elemento *i*-ésimo de *o*. Lanza una excepción y retorna `-1` en caso de falla; retorna `0` en caso de éxito. Este es el equivalente de la declaración de Python `o[i]=v`. Esta función *no* roba una referencia a *v*.

If *v* is `NULL`, the element is deleted, but this feature is deprecated in favour of using **`PySequence_DelItem()`**.

int PySequence_DelItem(PyObject *o, Py_ssize_t i)

Part of the Stable ABI.
Elimina el elemento *i*-ésimo del objeto *o*. Retorna `-1` en caso de falla. Este es el equivalente de la declaración de Python

`del o[i]`.

int PySequence_SetSlice(PyObject *o, Py_ssize_t i1, Py_ssize_t i2, PyObject *v)

Part of the Stable ABI.
Asigna el objeto secuencia *v* al segmento en el objeto secuencia *o* de *i1* a *i2*. Este es el equivalente de la declaración de Python `o[i1:i2]=v`.

int PySequence_DelSlice(PyObject *o, Py_ssize_t i1, Py_ssize_t i2)

Part of the Stable ABI.
Elimina el segmento en el objeto secuencia *o* de *i1* a *i2*. Retorna `-1` en caso de falla. Este es el equivalente de la declaración de Python `del o[i1:i2]`.

Py_ssize_t PySequence_Count(PyObject *o, PyObject *value)

Part of the Stable ABI.
Retorna el número de apariciones de *value* en *o*, es decir, retorna el número de claves para las que `o[clave]==value`. En caso de fallo, retorna `-1`. Esto es equivalente a la expresión de Python `o.count(value)`.

int PySequence_Contains(PyObject *o, PyObject *value)

Part of the Stable ABI.
Determine si *o* contiene *valor*. Si un elemento en *o* es igual a *value*, retorna `1`; de lo contrario, retorna `0`. En caso de error, retorna `-1`. Esto es equivalente a la expresión de Python `value in o`.

int PySequence_In(PyObject *o, PyObject *value)

Part of the Stable ABI.
Alias for **`PySequence_Contains()`**.

Soft deprecated since version 3.14: The function should no longer be used to write new code.

Py_ssize_t PySequence_Index(PyObject *o, PyObject *value)

Part of the Stable ABI.
Retorna el primer índice *i* para el que `o[i]==value`. En caso de error, retorna `-1`. Esto es equivalente a la expresión de Python `o.index(value)`.

PyObject *PySequence_List(PyObject *o)

Return value: New reference. Part of the Stable ABI.
Retorna un objeto lista con el mismo contenido que la secuencia o iterable *o*, o `NULL` en caso de error. La lista retornada está garantizada como nueva. Esto es equivalente a la expresión de Python `list(o)`.

PyObject *PySequence_Tuple(PyObject *o)

Return value: New reference. Part of the Stable ABI.
Retorna un objeto tupla con el mismo contenido que la secuencia o iterable *o*, o `NULL` en caso de error. Si *o* es una tupla, se retornará una nueva referencia; de lo contrario, se construirá una tupla con el contenido apropiado. Esto es equivalente a la expresión de Python `tupla(o)`.

PyObject *PySequence_Fast(PyObject *o, const char *m)

Return value: New reference. Part of the Stable ABI.
Retorna la secuencia o iterable *o* como un objeto utilizable por la otra familia de funciones `PySequence_Fast*`. Si el objeto no es una secuencia o no es iterable, lanza **`TypeError`** con *m* como texto del mensaje. Retorna `NULL` en caso de falla.

Las funciones `PySequence_Fast*` se denominan así porque suponen que *o* es un **`PyTupleObject`** o un **`PyListObject`** y acceden a los campos de datos de *o* directamente.

Como detalle de implementación de CPython, si *o* ya es una secuencia o lista, se retornará.

Py_ssize_t PySequence_Fast_GET_SIZE(PyObject *o)

Returns the length of *o*, assuming that *o* was returned by **`PySequence_Fast()`** and that *o* is not `NULL`. The size can also be retrieved by calling `PySequence_Size()` on *o*, but `PySequence_Fast_GET_SIZE()` is faster because it can assume *o* is a list or tuple.

PyObject *PySequence_Fast_GET_ITEM(PyObject *o, Py_ssize_t i)

Return value: Borrowed reference.
Retorna el elemento *i*-ésimo de *o*, suponiendo que *o* haya sido retornado por `PySequence_Fast()`, *o* no es `NULL` y que *i* está dentro de los límites.

PyObject **PySequence_Fast_ITEMS(PyObject *o)

Retorna el arreglo subyacente de punteros *PyObject*. Asume que *o* fue retornado por `PySequence_Fast()` y *o* no es `NULL`.

Tenga en cuenta que si una lista cambia de tamaño, la reasignación puede reubicar el arreglo de elementos. Por lo tanto, solo use el puntero de arreglo subyacente en contextos donde la secuencia no puede cambiar.

PyObject *PySequence_ITEM(PyObject *o, Py_ssize_t i)

Return value: New reference.
Retorna el elemento *i*-ésimo de *o* o `NULL` en caso de error. Es la forma más rápida de `PySequence_GetItem()` pero sin verificar que `PySequence_Check()` en *o* es verdadero y sin ajuste para índices negativos.

Protocolo de mapeo

Consulte también **`PyObject_GetItem()`**, **`PyObject_SetItem()`** y **`PyObject_DelItem()`**.

int PyMapping_Check(PyObject *o)

Part of the Stable ABI.
Return `1` if the object provides the mapping protocol or supports slicing, and `0` otherwise. Note that it returns `1` for Python classes with a **`__getitem__()`** method, since in general it is impossible to determine what type of keys the class supports. This function always succeeds.

Py_ssize_t PyMapping_Size(PyObject *o)
Py_ssize_t PyMapping_Length(PyObject *o)

Part of the Stable ABI.
Retorna el número de claves en el objeto *o* en caso de éxito, y `-1` en caso de error. Esto es equivalente a la expresión de Python `len(o)`.

PyObject *PyMapping_GetItemString(PyObject *o, const char *key)

Return value: New reference. Part of the Stable ABI.
This is the same as **`PyObject_GetItem()`**, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*.

int PyMapping_GetOptionalItem(PyObject *obj, PyObject *key, PyObject **result)

Part of the Stable ABI since version 3.13.
Variant of **`PyObject_GetItem()`** which doesn't raise **`KeyError`** if the key is not found.

If the key is found, return `1` and set **result* to a new strong reference to the corresponding value. If the key is not found,

return `0` and set **result* to `NULL`; the **`KeyError`** is silenced. If an error other than **`KeyError`** is raised, return `-1` and set **result* to `NULL`.

Added in version 3.13.

int PyMapping_GetOptionalItemString(PyObject *obj, const char *key, PyObject **result)

Part of the Stable ABI since version 3.13.
This is the same as **`PyMapping_GetOptionalItem()`**, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*.

Added in version 3.13.

int PyMapping_SetItemString(PyObject *o, const char *key, PyObject *v)

Part of the Stable ABI.
This is the same as **`PyObject_SetItem()`**, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*.

int PyMapping_DelItem(PyObject *o, PyObject *key)

This is an alias of **`PyObject_DelItem()`**.

int PyMapping_DelItemString(PyObject *o, const char *key)

This is the same as **`PyObject_DelItem()`**, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*.

int PyMapping_HasKeyWithError(PyObject *o, PyObject *key)

Part of the Stable ABI since version 3.13.
Return `1` if the mapping object has the key *key* and `0` otherwise. This is equivalent to the Python expression `key in o`. On failure, return `-1`.

Added in version 3.13.

int PyMapping_HasKeyStringWithError(PyObject *o, const char *key)

Part of the Stable ABI since version 3.13.
This is the same as **`PyMapping_HasKeyWithError()`**, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*.

Added in version 3.13.

int PyMapping_HasKey(PyObject *o, PyObject *key)

Part of the Stable ABI.
Retorna `1` si el objeto de mapeo tiene la clave *key* y `0` de lo contrario. Esto es equivalente a la expresión de Python `key in o`. Esta función siempre finaliza con éxito.

Nota

Exceptions which occur when this calls the **`__getitem__()`** method are silently ignored. For proper error handling, use **`PyMapping_HasKeyWithError()`**, **`PyMapping_GetOptionalItem()`** or **`PyObject_GetItem()`** instead.

int PyMapping_HasKeyString(PyObject *o, const char *key)

Part of the Stable ABI.
This is the same as **`PyMapping_HasKey()`**, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*.

Nota

Exceptions that occur when this calls the **`__getitem__()`** method or while creating the temporary **`str`** object are silently ignored. For proper error handling, use **`PyMapping_HasKeyStringWithError()`**, **`PyMapping_GetOptionalItemString()`** or **`PyMapping_GetItemString()`** instead.

PyObject *PyMapping_Keys(PyObject *o)

Return value: New reference. Part of the Stable ABI.
En caso de éxito, retorna una lista de las claves en el objeto *o*. En caso de fallo, retorna `NULL`.

Distinto en la versión 3.7: Anteriormente, la función retornaba una lista o una tupla.

PyObject *PyMapping_Values(PyObject *o)

Return value: New reference. Part of the Stable ABI.
En caso de éxito, retorna una lista de los valores en el objeto *o*. En caso de fallo, retorna `NULL`.

Distinto en la versión 3.7: Anteriormente, la función retornaba una lista o una tupla.

PyObject *PyMapping_Items(PyObject *o)

Return value: New reference. Part of the Stable ABI.
En caso de éxito, retorna una lista de los elementos en el objeto *o*, donde cada elemento es una tupla que contiene un par clave-valor (*key-value*). En caso de fallo, retorna `NULL`.

Distinto en la versión 3.7: Anteriormente, la función retornaba una lista o una tupla.

Protocolo iterador

Hay dos funciones específicas para trabajar con iteradores.

int PyIter_Check(PyObject *o)

Part of the Stable ABI since version 3.8.
Return non-zero if the object *o* can be safely passed to **`PyIter_NextItem()`** and `0` otherwise. This function always succeeds.

int PyAIter_Check(PyObject *o)

Part of the Stable ABI since version 3.10.
Retorna un valor distinto de cero si el objeto *o* proporciona el protocolo **`AsyncIterator`**, y `0` en caso contrario. Esta función siempre tiene éxito.

Added in version 3.10.

int PyIter_NextItem(PyObject *iter, PyObject **item)

Part of the Stable ABI since version 3.14.
Return `1` and set *item* to a strong reference of the next value of the iterator *iter* on success. Return `0` and set *item* to `NULL` if there are no remaining values. Return `-1`, set *item* to `NULL` and set an exception on error.

Added in version 3.14.

PyObject *PyIter_Next(PyObject *o)

Return value: New reference. Part of the Stable ABI.
This is an older version of **`PyIter_NextItem()`**, which is retained for backwards compatibility. Prefer **`PyIter_NextItem()`**.

Retorna el siguiente valor del iterador *o*. El objeto debe ser un iterador según **`PyIter_Check()`** (depende del llamador verificar esto). Si no hay valores restantes, retorna `NULL` sin establecer una excepción. Si ocurre un error al recuperar el elemento, devuelve `NULL` y envía la excepción.

type PySendResult

El valor de enumeración utilizado para representar diferentes resultados de **`PyIter_Send()`**.

Added in version 3.10.

PySendResult PyIter_Send(PyObject *iter, PyObject *arg, PyObject **presult)

Part of the Stable ABI since version 3.10.
Envía el valor *arg* al iterador *iter*. Retorna:

- `PYGEN_RETURN` si el iterador regresa. El valor de retorno se retorna a través de *presult*.
- `PYGEN_NEXT` si el iterador cede. El valor cedido se retorna a través de *presult*.
- `PYGEN_ERROR` si el iterador ha lanzado una excepción. *presult* se establece en `NULL`.

Added in version 3.10.

Protocolo búfer

Ciertos objetos disponibles en Python ajustan el acceso a un arreglo de memoria subyacente o *buffer*. Dichos objetos incluyen el incorporado `bytes` y `bytearray`, y algunos tipos de extensión como `array.array`. Las bibliotecas de terceros pueden definir sus propios tipos para fines especiales, como el procesamiento de imágenes o el análisis numérico.

Si bien cada uno de estos tipos tiene su propia semántica, comparten la característica común de estar respaldados por un búfer de memoria posiblemente grande. Es deseable, en algunas situaciones, acceder a ese búfer directamente y sin copia intermedia.

Python provides such a facility at the C and Python level in the form of the buffer protocol. This protocol has two sides:

- on the producer side, a type can export a «buffer interface» which allows objects of that type to expose information about their underlying buffer. This interface is described in the section Estructuras de objetos búfer; for Python see Emulando tipos de búfer.
- on the consumer side, several means are available to obtain a pointer to the raw underlying data of an object (for example a method parameter). For Python see `memoryview`.

Los objetos simples como `bytes` y `bytearray` exponen su búfer subyacente en forma orientada a bytes. Otras formas son posibles; por ejemplo, los elementos expuestos por un `array.array` pueden ser valores de varios bytes.

An example consumer of the buffer interface is the `write()` method of file objects: any object that can export a series of bytes through the buffer interface can be written to a file. While `write()` only needs read-only access to the internal contents of the object passed to it, other methods such as `readinto()` need write access to the contents of their argument. The buffer interface allows objects to selectively allow or reject exporting of read-write and read-only buffers.

Hay dos formas para que un consumidor de la interfaz del búfer adquiera un búfer sobre un objeto de destino:

- llamar **`PyObject_GetBuffer()`** con los parámetros correctos;
- llamar **`PyArg_ParseTuple()`** (o uno de sus hermanos) con uno de los `y*`, `w*` o `s*` códigos de formato.

En ambos casos, se debe llamar a **`PyBuffer_Release()`** cuando ya no se necesita el búfer. De lo contrario, podrían surgir varios problemas, como pérdidas de recursos.

Added in version 3.12: The buffer protocol is now accessible in Python, see Emulando tipos de búfer and **`memoryview`**.

Estructura de búfer

Las estructuras de búfer (o simplemente «búferes») son útiles como una forma de exponer los datos binarios de otro objeto al programador de Python. También se pueden usar como un mecanismo de corte de copia cero. Usando su capacidad para hacer referencia a un bloque de memoria, es posible exponer cualquier información al programador Python con bastante facilidad. La memoria podría ser una matriz grande y constante en una extensión C, podría ser un bloque de memoria sin procesar para su manipulación antes de pasar a una biblioteca del sistema operativo, o podría usarse para pasar datos estructurados en su formato nativo en memoria .

Contrariamente a la mayoría de los tipos de datos expuestos por el intérprete de Python, los búferes no son punteros **`PyObject`** sino estructuras C simples. Esto les permite ser creados y copiados de manera muy simple. Cuando se necesita un contenedor genérico alrededor de un búfer, un objeto memoryview puede ser creado.

Para obtener instrucciones breves sobre cómo escribir un objeto de exportación, consulte Estructuras de objetos búfer. Para obtener un búfer, consulte **`PyObject_GetBuffer()`**.

type Py_buffer

Part of the Stable ABI (including all members) since version 3.11.

void *buf

Un puntero al inicio de la estructura lógica descrita por los campos del búfer. Puede ser cualquier ubicación dentro del bloque de memoria física subyacente del exportador. Por ejemplo, con negativo **`strides`** el valor puede apuntar al final del bloque de memoria.

Para arreglos contiguous, el valor apunta al comienzo del bloque de memoria.

PyObject *obj

A new reference to the exporting object. The reference is owned by the consumer and automatically released (i.e. reference count decremented) and set to `NULL` by **`PyBuffer_Release()`**. The field is the equivalent of the return value of any standard C-API function.

Como un caso especial, para los búferes *temporary* que están envueltos por **`PyMemoryView_FromBuffer()`** o **`PyBuffer_FillInfo()`** este campo es `NULL`. En general, los objetos de exportación NO DEBEN usar este esquema.

Py_ssize_t len

`product(shape) * itemize`. Para arreglos contiguos, esta es la longitud del bloque de memoria subyacente. Para arreglos no contiguos, es la longitud que tendría la estructura lógica si se copiara en una representación contigua.

Accede a `((char *)buf)[0] hasta ((char *)buf)[len-1]` solo es válido si el búfer se ha obtenido mediante una solicitud que garantiza la contigüidad. En la mayoría de los casos, dicha solicitud será **PyBUF_SIMPLE** o **PyBUF_WRITABLE**.

int readonly

Un indicador de si el búfer es de solo lectura. Este campo está controlado por el indicador **PyBUF_WRITABLE**.

Py_ssize_t itemsize

Tamaño del elemento en bytes de un solo elemento. Igual que el valor de **struct.calcsize()** invocado en valores no `NULL` **format**.

Excepción importante: si un consumidor solicita un búfer sin el indicador **PyBUF_FORMAT**, **format** se establecerá en `NULL`, pero **itemsize** todavía tiene el valor para el formato original.

Si **shape** está presente, la igualdad `product(shape) * itemsize == len` aún se mantiene y el consumidor puede usar **itemsize** para navegar el búfer.

Si **shape** es `NULL` como resultado de un **PyBUF_SIMPLE** o un **PyBUF_WRITABLE**, el consumidor debe ignorar **itemsize** y asume `itemsize == 1`.

char *format

A *NULL* terminated string in **struct** module style syntax describing the contents of a single item. If this is `NULL`, `"B"` (unsigned bytes) is assumed.

Este campo está controlado por el indicador **PyBUF_FORMAT**.

int ndim

The number of dimensions the memory represents as an n-dimensional array. If it is `0`, **buf** points to a single item representing a scalar. In this case, **shape**,

`strides` and **`suboffsets`** MUST be `NULL`. The maximum number of dimensions is given by **`PyBUF_MAX_NDIM`**.

Py_ssize_t *shape

Un arreglo de **`Py_ssize_t`** de longitud **`ndim`** que indica la forma de la memoria como un arreglo n-dimensional. Tenga en cuenta que `shape[0] * ... * shape[ndim-1] * itemsize` DEBE ser igual a **`len`**.

Los valores de forma están restringidos a `shape[n] >= 0`. El caso `shape[n] == 0` requiere atención especial. Vea arreglos complejos (complex arrays) para más información.

El arreglo de formas es de sólo lectura para el consumidor.

Py_ssize_t *strides

Un arreglo de **`Py_ssize_t`** de longitud **`ndim`** que proporciona el número de bytes que se omiten para llegar a un nuevo elemento en cada dimensión.

Los valores de *stride* pueden ser cualquier número entero. Para los arreglos regulares, los pasos son generalmente positivos, pero un consumidor DEBE ser capaz de manejar el caso `strides[n] <= 0`. Ver complex arrays para más información.

El arreglo *strides* es de sólo lectura para el consumidor.

Py_ssize_t *suboffsets

Un arreglo de **`Py_ssize_t`** de longitud **`ndim`**. Si `suboffsets[n] >= 0`, los valores almacenados a lo largo de la enésima dimensión son punteros y el valor del *suboffsets* dicta cuántos bytes agregar a cada puntero después de desreferenciarlos. Un valor de *suboffsets* negativo indica que no debe producirse una desreferenciación (*striding* en un bloque de memoria contiguo).

Si todos los *suboffsets* son negativos (es decir, no se necesita desreferenciar), entonces este campo debe ser `NULL` (el valor predeterminado).

Python Imaging Library (PIL) utiliza este tipo de representación de arreglos. Consulte complex arrays para obtener más información sobre cómo acceder a los elementos de dicho arreglo.

El arreglo de *suboffsets* es de sólo lectura para el consumidor.

void *internal

Esto es para uso interno del objeto exportador. Por ejemplo, el exportador podría volver a emitirlo como un número entero y utilizarlo para almacenar indicadores sobre si las matrices de forma, *strides* y *suboffsets* deben liberarse cuando se libera el búfer. El consumidor NO DEBE alterar este valor.

Constants:

PyBUF_MAX_NDIM

Part of the Stable ABI since version 3.11.
The maximum number of dimensions the memory represents. Exporters MUST respect this limit, consumers of multi-dimensional buffers SHOULD be able to handle up to **`PyBUF_MAX_NDIM`** dimensions. Currently set to 64.

Tipos de solicitud búfer

Los búferes obtienen generalmente enviando una solicitud de búfer a un objeto de exportación a través de **`PyObject_GetBuffer()`**. Dado que la complejidad de la estructura lógica de la memoria puede variar drásticamente, el consumidor usa el argumento *flags* para especificar el tipo de búfer exacto que puede manejar.

All **`Py_buffer`** fields are unambiguously defined by the request type.

campos independientes de solicitud

Los siguientes campos no están influenciados por *flags* y siempre deben completarse con los valores correctos: **`obj`**, **`buf`**, **`len`**, **`itemsize`**, **`ndim`**.

formato de sólo lectura

PyBUF_WRITABLE

Part of the Stable ABI since version 3.11. Controls the **`readonly`** field. If set, the exporter MUST provide a writable buffer or else report failure. Otherwise, the exporter MAY provide either a read-only or writable buffer, but the choice MUST be consistent for all consumers. For example, PyBUF_SIMPLE | PyBUF_WRITABLE can be used to request a simple writable buffer.

PyBUF_WRITEABLE

This is a soft deprecated alias to **`PyBUF_WRITABLE`**.

PyBUF_FORMAT

Part of the Stable ABI since version 3.11. Controla el campo **`format`**. Si se establece, este campo DEBE completarse correctamente. De lo contrario, este campo DEBE ser `NULL`.

`PyBUF_WRITABLE` puede ser |”d a cualquiera de las banderas en la siguiente sección. Dado que **`PyBUF_SIMPLE`** se define como 0, **`PyBUF_WRITABLE`** puede usarse como un indicador independiente para solicitar un búfer de escritura simple.

`PyBUF_FORMAT` must be |”d to any of the flags except **`PyBUF_SIMPLE`**, because the latter already implies format `B` (unsigned bytes). **`PyBUF_FORMAT`** cannot be used on its own.

formas, *strides*, *suboffsets*

Las banderas que controlan la estructura lógica de la memoria se enumeran en orden decreciente de complejidad. Tenga en cuenta que cada bandera contiene todos los bits de las banderas debajo de ella.

Solicitud	**forma**	***strides***	***suboffsets***
PyBUF_INDIRECT *Part of the Stable ABI since version 3.11.*	sí	yes	si es necesario
PyBUF_STRIDES *Part of the Stable ABI since version 3.11.*	sí	yes	NULL
PyBUF_ND *Part of the Stable ABI since version 3.11.*	sí	NULL	NULL
PyBUF_SIMPLE *Part of the Stable ABI since version 3.11.*	NULL	NULL	NULL

solicitudes de contigüidad

La contigüidad C o Fortran se puede solicitar explícitamente, con y sin información de paso. Sin información de paso, el búfer debe ser C-contiguo.

Solicitud	**forma**	***strides***	***suboffsets***	**contig**
PyBUF_C_CONTIGUOUS *Part of the Stable ABI since version 3.11.*	sí	yes	NULL	C
PyBUF_F_CONTIGUOUS *Part of the Stable ABI since version 3.11.*	sí	yes	NULL	F
PyBUF_ANY_CONTIGUOUS *Part of the Stable ABI since version 3.11.*	sí	yes	NULL	C o F
`PyBUF_ND`	sí	NULL	NULL	C

solicitudes compuestas

Todas las solicitudes posibles están completamente definidas por alguna combinación de las banderas en la sección anterior. Por conveniencia, el protocolo de memoria intermedia proporciona combinaciones de uso frecuente como indicadores únicos.

En la siguiente tabla *U* significa contigüidad indefinida. El consumidor tendría que llamar a **`PyBuffer_IsContiguous()`** para determinar la contigüidad.

PyBUF_FULL

Part of the Stable ABI since version 3.11.

PyBUF_FULL_RO

Part of the Stable ABI since version 3.11.

PyBUF_RECORDS

Part of the Stable ABI since version 3.11.

PyBUF_RECORDS_RO

Part of the Stable ABI since version 3.11.

PyBUF_STRIDED

Part of the Stable ABI since version 3.11.

PyBUF_STRIDED_RO

Part of the Stable ABI since version 3.11.

PyBUF_CONTIG

Part of the Stable ABI since version 3.11.

PyBUF_CONTIG_RO

Part of the Stable ABI since version 3.11.

Arreglos complejos

Estilo NumPy: forma y *strides*

La estructura lógica de las matrices de estilo NumPy está definida por **`itemsize`**, **`ndim`**, **`shape`** y **`strides`**.

Si `ndim == 0`, la ubicación de memoria señalada por **`buf`** se interpreta como un escalar de tamaño **`itemsize`**. En ese caso, tanto **`shape`** como **`strides`** son `NULL`.

Si **`strides`** es `NULL`, el arreglo se interpreta como un arreglo C n-dimensional estándar. De lo contrario, el consumidor debe acceder a un arreglo n-dimensional de la siguiente manera:

```
ptr = (char *)buf + indices[0] * strides[0] + ... + indi
item = *((typeof(item) *)ptr);
```

Como se señaló anteriormente, `buf` puede apuntar a cualquier ubicación dentro del bloque de memoria real. Un exportador puede verificar la validez de un búfer con esta función:

```
def verify_structure(memlen, itemsize, ndim, shape, stri
    """Verify that the parameters represent a valid arra
       the bounds of the allocated memory:
           char *mem: start of the physical memory block
           memlen: length of the physical memory block
           offset: (char *)buf - mem
    """
    if offset % itemsize:
        return False
    if offset < 0 or offset+itemsize > memlen:
        return False
    if any(v % itemsize for v in strides):
        return False

    if ndim <= 0:
        return ndim == 0 and not shape and not strides
    if 0 in shape:
        return True

    imin = sum(strides[j]*(shape[j]-1) for j in range(nd
               if strides[j] <= 0)
    imax = sum(strides[j]*(shape[j]-1) for j in range(nd
               if strides[j] > 0)

    return 0 <= offset+imin and offset+imax+itemsize <=
```

Estilo PIL: forma, *strides* y *suboffsets*

Además de los elementos normales, los arreglos de estilo PIL pueden contener punteros que deben seguirse para llegar al siguiente elemento en una dimensión. Por ejemplo, el arreglo C tridimensional regular `char v[2][2][3]` también se puede ver como un arreglo de 2 punteros a 2 arreglos bidimensionales: `char (*v[2])[2][3]`. En la representación de *suboffsets*, esos dos punteros pueden incrustarse al comienzo de **`buf`**, apuntando a dos matrices `char x[2][3]` que pueden ubicarse en cualquier lugar de la memoria.

Aquí hay una función que retorna un puntero al elemento en un arreglo N-D a la que apunta un índice N-dimensional cuando hay *strides* y *suboffsets* no `NULL`:

```
void *get_item_pointer(int ndim, void *buf, Py_ssize_t *
                       Py_ssize_t *suboffsets, Py_ssize_
    char *pointer = (char*)buf;
    int i;
    for (i = 0; i < ndim; i++) {
        pointer += strides[i] * indices[i];
        if (suboffsets[i] >=0 ) {
            pointer = *((char**)pointer) + suboffsets[i]
        }
    }
    return (void*)pointer;
}
```

Funciones relacionadas a búfer

int PyObject_CheckBuffer(PyObject *obj)

Part of the Stable ABI since version 3.11.
Retorna `1` si *obj* admite la interfaz de búfer; de lo contrario, `0` cuando se retorna `1`, no garantiza que **`PyObject_GetBuffer()`** tenga éxito. Esta función siempre finaliza con éxito.

int PyObject_GetBuffer(PyObject *exporter, Py_buffer *view, int flags)

Part of the Stable ABI since version 3.11.
Send a request to *exporter* to fill in *view* as specified by *flags*. If

the exporter cannot provide a buffer of the exact type, it MUST raise **`BufferError`**, set `view->obj` to `NULL` and return `-1`.

Si tiene éxito, completa *view*, establece `view->obj` en una nueva referencia a *exporter* y retorna 0. En el caso de proveedores de búfer encadenados que redirigen las solicitudes a un solo objeto, `view->obj` PUEDE referirse a este objeto en lugar de *exporter* (Ver Estructuras de objetos de búfer).

Las llamadas exitosas a **`PyObject_GetBuffer()`** deben combinarse con las llamadas a **`PyBuffer_Release()`**, similar a **`malloc()`** y **`free()`**. Por lo tanto, después de que el consumidor haya terminado con el búfer, **`PyBuffer_Release()`** debe llamarse exactamente una vez.

void PyBuffer_Release(Py_buffer *view)

Part of the Stable ABI since version 3.11.
Release the buffer *view* and release the strong reference (i.e. decrement the reference count) to the view's supporting object, `view->obj`. This function MUST be called when the buffer is no longer being used, otherwise reference leaks may occur.

Es un error llamar a esta función en un búfer que no se obtuvo a través de **`PyObject_GetBuffer()`**.

Py_ssize_t PyBuffer_SizeFromFormat(const char *format)

Part of the Stable ABI since version 3.11.
Return the implied **`itemsize`** from **`format`**. On error, raise an exception and return -1.

Added in version 3.9.

int PyBuffer_IsContiguous(const Py_buffer *view, char order)

Part of the Stable ABI since version 3.11.
Retorna `1` si la memoria definida por *view* es de estilo C (*order* es `'C'`) o de estilo Fortran (*order* es `'F'`) contiguous

o uno cualquiera (*order* es `'A'`). Retorna `0` de lo contrario. Esta función siempre finaliza con éxito.

void *PyBuffer_GetPointer(const Py_buffer *view, const Py_ssize_t *indices)

Part of the Stable ABI since version 3.11.
Obtiene el área de memoria señalada por los *indices* dentro del *view* dado. *indices* deben apuntar a un arreglo de índices `view->ndim`.

int PyBuffer_FromContiguous(const Py_buffer *view, const void *buf, Py_ssize_t len, char fort)

Part of the Stable ABI since version 3.11.
Copia *len* bytes contiguos de *buf* a *view*. *fort* puede ser `'C'` o `'F'` (para pedidos al estilo C o al estilo Fortran). `0` se retorna en caso de éxito, `-1` en caso de error.

int PyBuffer_ToContiguous(void *buf, const Py_buffer *src, Py_ssize_t len, char order)

Part of the Stable ABI since version 3.11.
Copia *len* bytes de *src* a su representación contigua en *buf*. *order* puede ser `'C'` o `'F'` o `''A'` (para pedidos al estilo C o al estilo Fortran o cualquiera) `0` se retorna en caso de éxito, `-1` en caso de error.

Esta función falla si *len* != *src->len*.

int PyObject_CopyData(PyObject *dest, PyObject *src)

Part of the Stable ABI since version 3.11.
Copiar datos del búfer *src* al *dest*. Puede convertir entre búferes de estilo C o Fortran.

Se retorna `0` en caso de éxito, `-1` en caso de error.

void PyBuffer_FillContiguousStrides(int ndims, Py_ssize_t *shape, Py_ssize_t *strides, int itemsize, char order)

Part of the Stable ABI since version 3.11.

Rellena el arreglo *strides* con bytes de paso de un contiguous (estilo C si *order* es `'C'` o estilo Fortran si *order* es `'F'`) arreglo de la forma dada con el número dado de bytes por elemento.

int PyBuffer_FillInfo(Py_buffer *view, PyObject *exporter, void *buf, Py_ssize_t len, int readonly, int flags)

Part of the Stable ABI since version 3.11.
Maneje las solicitudes de búfer para un exportador que quiera exponer *buf* de tamaño *len* con capacidad de escritura establecida de acuerdo con *readonly*. *buf* se interpreta como una secuencia de bytes sin signo.

El argumento *flags* indica el tipo de solicitud. Esta función siempre llena *view* según lo especificado por *flags*, a menos que *buf* haya sido designado como solo lectura y **`PyBUF_WRITABLE`** esté configurado en *flags*.

On success, set `view->obj` to a new reference to *exporter* and return 0. Otherwise, raise **`BufferError`**, set `view->obj` to `NULL` and return `-1`;

Si esta función se usa como parte de a getbufferproc, *exporter* DEBE establecerse en el objeto exportador y *flags* deben pasarse sin modificaciones. De lo contrario, *exporter* DEBE ser `NULL`.

Capa de objetos concretos

Las funciones de este capítulo son específicas de ciertos tipos de objetos de Python. Pasarles un objeto del tipo incorrecto no es una buena idea; si recibe un objeto de un programa Python y no está seguro de que tenga el tipo correcto, primero debe realizar una verificación de tipo; por ejemplo, para verificar que un objeto es un diccionario, utilice `PyDict_Check()`. El capítulo está estructurado como el «árbol genealógico» de los tipos de objetos Python.

Advertencia

Si bien las funciones descritas en este capítulo verifican cuidadosamente el tipo de objetos que se pasan, muchos de ellos no verifican si se pasa `NULL` en lugar de un objeto válido. Permitir que se pase `NULL` puede causar violaciones de acceso a la memoria y la terminación inmediata del intérprete.

Objetos fundamentales

Esta sección describe los objetos de tipo Python y el objeto singleton `None`.

- Objetos tipo
  - Crear tipos asignados en montículo (*heap*)
- El objeto `None`

Objetos numéricos

- Objetos enteros
  - Export API

    - PyLongWriter API
    - Deprecated API
- Objetos booleanos
- Floating-Point Objects
    - Funciones de empaquetado y desempaquetado
        - Funciones de Empaquetado
        - Funciones de Desempaquetado
- Objetos de números complejos
    - Números complejos como estructuras C
    - Números complejos como objetos de Python

Objetos de secuencia

Las operaciones genéricas en los objetos de secuencia se discutieron en el capítulo anterior; Esta sección trata sobre los tipos específicos de objetos de secuencia que son intrínsecos al lenguaje Python.

- Objetos bytes
- Objetos de arreglos de bytes (*bytearrays*)
    - Macros de verificación de tipos
    - Funciones API directas
    - Macros
- Objetos y códecs unicode
    - Objetos unicode
        - Tipo unicode
        - Propiedades de caracteres Unicode
        - Creando y accediendo a cadenas de caracteres Unicode
        - Codificación regional
        - Codificación del sistema de archivos
        - soporte wchar_t

    - Códecs incorporados
        - Códecs genéricos
        - Códecs UTF-8
        - Códecs UTF-32
        - Códecs UTF-16
        - Códecs UTF-7
        - Códecs Unicode escapado
        - Códecs para Unicode escapado en bruto
        - Códecs Latin-1
        - Códecs ASCII
        - Códecs de mapa de caracteres
        - Códecs MBCS para Windows
    - Métodos y funciones de ranura (*Slot*)
    - PyUnicodeWriter
    - Deprecated API
- Objetos tupla
- Objetos de secuencia de estructura
- Objetos lista

Objetos contenedor

- Objetos diccionario
    - Dictionary View Objects
    - Ordered Dictionaries
- Objetos conjunto
    - Deprecated API

Objetos de función

- Objetos función
- Objetos de método de instancia
- Objetos método
- Objetos celda
- Objetos código

- Code Object Flags
- Extra information

Otros objetos

- File objects
    - Soft-deprecated API
- Objetos módulo
- Module definitions
    - Module slots
- Creating extension modules dynamically
- Funciones de soporte
    - Module lookup (single-phase initialization)
- Objetos iteradores
    - Range Objects
    - Builtin Iterator Types
    - Other Iterator Objects
- Objetos descriptores
    - Built-in descriptors
- Objetos rebanada (*slice*)
    - Objeto elipsis
- Objetos de vista de memoria (*MemoryView*)
- Pickle buffer objects
- Objetos de referencia débil
- Cápsulas
- Frame objects
    - Frame locals proxies
    - Legacy local variable APIs
    - Internal frames

- Objetos generadores
    - Asynchronous Generator Objects
    - Deprecated API
- Objetos corrutina
- Objetos de variables de contexto
- Objetos para indicaciones de tipado

C API for extension modules

- Curses C API
- Internal data
- Objetos *DateTime*
- Internal data

Objetos tipo

type PyTypeObject

Part of the Limited API (as an opaque struct).

La estructura C de los objetos utilizados para describir los tipos incorporados.

PyTypeObject PyType_Type

Part of the Stable ABI.

Este es el objeto tipo para objetos tipo; es el mismo objeto que `type` en la capa Python.

int PyType_Check(PyObject *o)

Retorna un valor distinto de cero si el objeto *o* es un objeto tipo, incluidas las instancias de tipos derivados del objeto de tipo estándar. Retorna 0 en todos los demás casos. Esta función siempre finaliza con éxito.

int PyType_CheckExact(PyObject *o)

Retorna un valor distinto de cero si el objeto *o* es un objeto tipo, pero no un subtipo del objeto tipo estándar. Retorna 0 en todos los demás casos. Esta función siempre finaliza con éxito.

unsigned int PyType_ClearCache()

Part of the Stable ABI.

Borra la caché de búsqueda interna. Retorna la etiqueta (*tag*) de la versión actual.

unsigned long PyType_GetFlags(PyTypeObject *type)

Part of the Stable ABI.

Retorna el miembro `tp_flags` de *type*. Esta función está destinada principalmente para su uso con

`Py_LIMITED_API`; se garantiza que los bits de bandera individuales serán estables en las versiones de Python, pero el acceso a **`tp_flags`** en sí mismo no forma parte de la API limitada.

Added in version 3.2.

Distinto en la versión 3.4: El tipo de retorno es ahora `unsigned long` en vez de `long`.

PyObject *PyType_GetDict(PyTypeObject *type)

Retorna el espacio de nombres interno del objeto tipo, que de otra manera solo está expuesto a través de un *proxy* de solo lectura (**`cls.__dict__`**). Esto es un reemplazo para acceder a **`tp_dict`** directamente. El diccionario retornado debe tratarse como de solo lectura.

Esta función está destinada a casos específicos de incrustación y vinculación de lenguajes, donde el acceso directo al diccionario es necesario y el acceso indirecto (por ejemplo, a través del *proxy* o **`PyObject_GetAttr()`**) no es adecuado.

Los módulos de extensión deben continuar usando `tp_dict`, directa o indirectamente, al configurar sus propios tipos.

Added in version 3.12.

void PyType_Modified(PyTypeObject *type)

Part of the Stable ABI.
Invalida la memoria caché de búsqueda interna para el tipo y todos sus subtipos. Esta función debe llamarse después de cualquier modificación manual de los atributos o clases base del tipo.

int PyType_AddWatcher(PyType_WatchCallback callback)

Registra *callback* como un observador de tipo. Retorna un ID entero no negativo que debe pasarse a llamadas futuras a **`PyType_Watch()`**. En caso de error (por ejemplo, no hay más IDs de observador disponibles), retorna `-1` y establece una excepción.

In free-threaded builds, **`PyType_AddWatcher()`** is not thread-safe, so it must be called at start up (before spawning the first thread).

Added in version 3.12.

int PyType_ClearWatcher(int watcher_id)

Borra el observador identificado por *watcher_id* (previamente retornado de **`PyType_AddWatcher()`**). Retorna `0` en caso de éxito, `-1` en caso de error (por ejemplo, si *watcher_id* nunca fue registrado).

Una extensión nunca debe llamar a `PyType_ClearWatcher` con un *watcher_id* que no le fue retornado por una llamada previa a **`PyType_AddWatcher()`**.

Added in version 3.12.

int PyType_Watch(int watcher_id, PyObject *type)

Marca *type* como observado. El retrollamada (*callback*) otorgado *watcher_id* por **`PyType_AddWatcher()`** será llamado cada vez que **`PyType_Modified()`** reporte un cambio a *type*. (El *callback* puede ser llamado solo una vez para una serie de modificaciones consecutivas a *type*, si **`_PyType_Lookup()`** no es llamado en *type* entre las modificaciones; esto es un detalle de implementación y sujeto a cambios).

Una extensión nunca debe llamar a `PyType_Watch` con un *watcher_id* que no le fue retornado por una llamada previa a **`PyType_AddWatcher()`**.

Added in version 3.12.

int PyType_Unwatch(int watcher_id, PyObject *type)

Mark *type* as not watched. This undoes a previous call to **`PyType_Watch()`**. *type* must not be `NULL`.

An extension should never call this function with a *watcher_id* that was not returned to it by a previous call to **`PyType_AddWatcher()`**.

On success, this function returns `0`. On failure, this function returns `-1` with an exception set.

Added in version 3.12.

typedef int (*PyType_WatchCallback)(PyObject *type)

Tipo de una función de retrollamada (*callback*) de observador de tipo.

El retrollamada (*callback*) no debe modificar *type* o causar que **`PyType_Modified()`** sea llamado en *type* o cualquier tipo en su MRO; violar esta regla podría causar recursión infinita.

Added in version 3.12.

int PyType_HasFeature(PyTypeObject *o, int feature)

Retorna un valor distinto de cero si el tipo objeto *o* establece la característica *feature*. Las características de tipo se indican mediante flags de un solo bit.

int PyType_FastSubclass(PyTypeObject *type, int flag)

Return non-zero if the type object *type* sets the subclass flag *flag*. Subclass flags are denoted by **`Py_TPFLAGS_*_SUBCLASS`**. This function is used by many `_Check` functions for common types.

Ver también

`PyObject_TypeCheck()`, which is used as a slower alternative in `_Check` functions for types that don't come with subclass flags.

int PyType_IS_GC(PyTypeObject *o)

Retorna verdadero si el objeto tipo incluye soporte para el detector de ciclos; esto prueba el indicador de tipo **`Py_TPFLAGS_HAVE_GC`**.

int PyType_IsSubtype(PyTypeObject *a, PyTypeObject *b)

Part of the Stable ABI.
Retorna verdadero si *a* es un subtipo de *b*.

Esta función solo busca subtipos reales, lo que significa que **`__subclasscheck__()`** no se llama en *b*. Llama **`PyObject_IsSubclass()`** para hacer la misma verificación que **`issubclass()`** haría.

PyObject *PyType_GenericAlloc(PyTypeObject *type, Py_ssize_t nitems)

Return value: New reference. Part of the Stable ABI.
Generic handler for the **`tp_alloc`** slot of a type object. Uses Python's default memory allocation mechanism to allocate memory for a new instance, zeros the memory, then initializes the memory as if by calling **`PyObject_Init()`** or **`PyObject_InitVar()`**.

Do not call this directly to allocate memory for an object; call the type's **`tp_alloc`** slot instead.

For types that support garbage collection (i.e., the **`Py_TPFLAGS_HAVE_GC`** flag is set), this function behaves like **`PyObject_GC_New`** or **`PyObject_GC_NewVar`** (except the memory is guaranteed to be zeroed before initialization), and should be paired with **`PyObject_GC_Del()`** in **`tp_free`**. Otherwise, it behaves like **`PyObject_New`** or **`PyObject_NewVar`** (except the memory is guaranteed to be zeroed before initialization) and should be paired with **`PyObject_Free()`** in **`tp_free`**.

PyObject *PyType_GenericNew(PyTypeObject *type, PyObject *args, PyObject *kwds)

Return value: New reference. Part of the Stable ABI.

Generic handler for the `tp_new` slot of a type object. Creates a new instance using the type's `tp_alloc` slot and returns the resulting object.

int PyType_Ready(PyTypeObject *type)

Part of the Stable ABI.
Finalizar un objeto tipo. Se debe llamar a todos los objetos tipo para finalizar su inicialización. Esta función es responsable de agregar ranuras heredadas de la clase base de un tipo. Retorna `0` en caso de éxito o retorna `-1` y establece una excepción en caso de error.

Nota

Si algunas de las clases base implementan el protocolo GC y el tipo proporcionado no incluye `Py_TPFLAGS_HAVE_GC` en sus banderas, entonces el protocolo GC se implementará automáticamente desde sus padres. Por el contrario, si el tipo que se está creando incluye `Py_TPFLAGS_HAVE_GC` en sus banderas, entonces **debe** implementar el protocolo GC por sí mismo implementando al menos el manejador `tp_traverse`.

PyObject *PyType_GetName(PyTypeObject *type)

Return value: New reference. Part of the Stable ABI since version 3.11.
Retorna el nombre del tipo. Equivalente a obtener el atributo `__name__` del tipo.

Added in version 3.11.

PyObject *PyType_GetQualName(PyTypeObject *type)

Return value: New reference. Part of the Stable ABI since version 3.11.
Retorna el nombre calificado del tipo. Equivalente a obtener el atributo `__qualname__` del tipo.

Added in version 3.11.

PyObject *PyType_GetFullyQualifiedName(PyTypeObject *type)

Part of the Stable ABI since version 3.13.
Retorna el nombre completamente calificado del tipo. Equivalente a `f"{type.__module__}.{type.__qualname__}"`, o **`type.__qualname__`** si **`type.__module__`** no es una cadena de caracteres o es igual a `"builtins"`.

Added in version 3.13.

PyObject *PyType_GetModuleName(PyTypeObject *type)

Part of the Stable ABI since version 3.13.
Retorna el nombre del módulo del tipo. Equivalente a obtener el atributo **`type.__module__`**.

Added in version 3.13.

void *PyType_GetSlot(PyTypeObject *type, int slot)

Part of the Stable ABI since version 3.4.
Retorna el puntero de función almacenado en la ranura dada. Si el resultado es `NULL`, esto indica que la ranura es `NULL` o que la función se llamó con parámetros no válidos. Las personas que llaman suelen convertir el puntero de resultado en el tipo de función apropiado.

Consulte **`PyType_Slot.slot`** para conocer los posibles valores del argumento *slot*.

Added in version 3.4.

Distinto en la versión 3.10: **`PyType_GetSlot()`** ahora puede aceptar todos los tipos. Anteriormente, estaba limitado a heap types.

PyObject *PyType_GetModule(PyTypeObject *type)

Return value: Borrowed reference. Part of the Stable ABI since version 3.10.

Retorna el objeto módulo asociado con el tipo dado cuando se creó el tipo usando **`PyType_FromModuleAndSpec()`**.

The returned reference is borrowed from *type*, and will be valid as long as you hold a reference to *type*. Do not release it with **`Py_DECREF()`** or similar.

Si no hay ningún módulo asociado con el tipo dado, establece **`TypeError`** y retorna `NULL`.

Esta función se suele utilizar para obtener el módulo en el que se define un método. Tenga en cuenta que en un método de este tipo, es posible que `PyType_GetModule(Py_TYPE(self))` no retorne el resultado deseado. `Py_TYPE(self)` puede ser una *subclass* de la clase deseada, y las subclases no están necesariamente definidas en el mismo módulo que su superclase. Consulte **`PyCMethod`** para obtener la clase que define el método. Consulte **`PyType_GetModuleByDef()`** para los casos en los que no se puede usar **`PyCMethod`**.

Added in version 3.9.

void *PyType_GetModuleState(PyTypeObject *type)

Part of the Stable ABI since version 3.10.
Retorna el estado del objeto de módulo asociado con el tipo dado. Este es un atajo para llamar **`PyModule_GetState()`** en el resultado de **`PyType_GetModule()`**.

Si no hay ningún módulo asociado con el tipo dado, establece **`TypeError`** y retorna `NULL`.

Si el tipo *type* tiene un módulo asociado pero su estado es `NULL`, retorna `NULL` sin establecer una excepción.

Added in version 3.9.

PyObject *PyType_GetModuleByDef(PyTypeObject *type, struct PyModuleDef *def)

Return value: Borrowed reference. Part of the Stable ABI since

version 3.13.
Encuentra la primer superclase cuyo módulo fue creado a partir del **`PyModuleDef`** *def* dado, y retorna ese módulo.

Si no se encuentra ningún módulo, lanza **`TypeError`** y retorna `NULL`.

Esta función está pensada para ser utilizada junto con **`PyModule_GetState()`** para obtener el estado del módulo de los métodos de ranura (como **`tp_init`** o **`nb_add`**) y en otros lugares donde la clase que define a un método no se puede pasar utilizando la convención de llamada **`PyCMethod`**.

The returned reference is borrowed from *type*, and will be valid as long as you hold a reference to *type*. Do not release it with **`Py_DECREF()`** or similar.

Added in version 3.11.

int PyType_GetBaseByToken(PyTypeObject *type, void *token, PyTypeObject **result)

Part of the Stable ABI since version 3.14.
Find the first superclass in *type*'s method resolution order whose **`Py_tp_token`** token is equal to the given one.

- If found, set **result* to a new strong reference to it and return `1`.
- If not found, set **result* to `NULL` and return `0`.
- On error, set **result* to `NULL` and return `-1` with an exception set.

The *result* argument may be `NULL`, in which case **result* is not set. Use this if you need only the return value.

The *token* argument may not be `NULL`.

Added in version 3.14.

int PyUnstable_Type_AssignVersionTag(PyTypeObject *type)

This is Unstable API. It may change without warning in minor releases.

Intenta asignar una etiqueta (*tag*) de versión al tipo dado.

Retorna 1 si el tipo ya tenía una etiqueta (*tag*) de versión válida o se asignó una nueva, o 0 si no se pudo asignar una nueva etiqueta.

Added in version 3.12.

int PyType_SUPPORTS_WEAKREFS(PyTypeObject *type)

Return true if instances of *type* support creating weak references, false otherwise. This function always succeeds. *type* must not be `NULL`.

Ver también

- Objetos de referencia débil
- **`weakref`**

Crear tipos asignados en montículo (*heap*)

Las siguientes funciones y estructuras se utilizan para crear heap types.

PyObject *PyType_FromMetaclass(PyTypeObject *metaclass, PyObject *module, PyType_Spec *spec, PyObject *bases)

Part of the Stable ABI since version 3.12.
Crea y retorna un tipo heap a partir del *spec* (ver **`Py_TPFLAGS_HEAPTYPE`**).

La metaclase *metaclass* se usa para construir el objeto tipo resultante. Cuando *metaclass* es `NULL`, la metaclase se deriva de *bases* (o ranuras *Py_tp_base[s]* si *bases* es `NULL`, ver abajo).

Metaclasses that override **`tp_new`** are not supported, except if `tp_new` is `NULL`.

The *bases* argument can be used to specify base classes; it can either be only one class or a tuple of classes. If *bases* is `NULL`, the **`Py_tp_bases`** slot is used instead. If that also is `NULL`, the **`Py_tp_base`** slot is used instead. If that also is `NULL`, the new type derives from **`object`**.

El argumento *module* se puede utilizar para registrar el módulo en el que se define la nueva clase. Debe ser un objeto de módulo o `NULL`. Si no es `NULL`, el módulo se asocia con el nuevo tipo y luego se puede recuperar con **`PyType_GetModule()`**. El módulo asociado no es heredado por subclases; debe especificarse para cada clase individualmente.

Esta función llama **`PyType_Ready()`** en el tipo nuevo.

Nota que esta función *no* coincide completamente con el comportamiento de llamar **`type()`** o usar la sentencia **`class`**. Con tipos base o metaclases proporcionados por el usuario, prefiera llamar **`type`** (o la metaclase) sobre las funciones `PyType_From*`. Específicamente:

- **`__new__()`** no es llamado en la nueva clase (y debe ser establecido a `type.__new__`).
- **`__init__()`** no es llamado en la nueva clase.
- **`__init_subclass__()`** no es llamado en ninguna base.
- **`__set_name__()`** no es llamado en nuevos descriptores.

Added in version 3.12.

PyObject *PyType_FromModuleAndSpec(PyObject *module, PyType_Spec *spec, PyObject *bases)

Return value: New reference. Part of the Stable ABI since version 3.10.

Equivalente a `PyType_FromMetaclass(NULL, module, spec, bases)`.

Added in version 3.9.

Distinto en la versión 3.10: La función ahora acepta una sola clase como argumento *bases* y `NULL` como ranura `tp_doc`.

Distinto en la versión 3.12: La función ahora encuentra y usa una metaclase correspondiente a las clases base proporcionadas. Anteriormente, sólo se retornaban instancias de **`type`**.

The **`tp_new`** of the metaclass is *ignored*. which may result in incomplete initialization. Creating classes whose metaclass overrides **`tp_new`** is deprecated.

Distinto en la versión 3.14: Creating classes whose metaclass overrides **`tp_new`** is no longer allowed.

PyObject *PyType_FromSpecWithBases(PyType_Spec *spec, PyObject *bases)

Return value: New reference. Part of the Stable ABI since version 3.3.
Equivalente a `PyType_FromMetaclass(NULL, NULL, spec, bases)`.

Added in version 3.3.

Distinto en la versión 3.12: La función ahora encuentra y usa una metaclase correspondiente a las clases base proporcionadas. Anteriormente, sólo se retornaban instancias de **`type`**.

The **`tp_new`** of the metaclass is *ignored*. which may result in incomplete initialization. Creating classes whose metaclass overrides **`tp_new`** is deprecated.

Distinto en la versión 3.14: Creating classes whose metaclass overrides **`tp_new`** is no longer allowed.

PyObject *PyType_FromSpec(PyType_Spec *spec)

Return value: New reference. Part of the Stable ABI.

Equivalente a `PyType_FromMetaclass(NULL, NULL, spec, NULL)`.

Distinto en la versión 3.12: La función ahora encuentra y usa una metaclase correspondiente a las clases base proporcionadas en las ranuras *Py_tp_base[s]*. Anteriormente, sólo se retornaban instancias de **`type`**.

The **`tp_new`** of the metaclass is *ignored*. which may result in incomplete initialization. Creating classes whose metaclass overrides **`tp_new`** is deprecated.

Distinto en la versión 3.14: Creating classes whose metaclass overrides **`tp_new`** is no longer allowed.

int PyType_Freeze(PyTypeObject *type)

Part of the Stable ABI since version 3.14.
Make a type immutable: set the **`Py_TPFLAGS_IMMUTABLETYPE`** flag.

All base classes of *type* must be immutable.

On success, return `0`. On error, set an exception and return `-1`.

The type must not be used before it's made immutable. For example, type instances must not be created before the type is made immutable.

Added in version 3.14.

type PyType_Spec

Part of the Stable ABI (including all members).
Estructura que define el comportamiento de un tipo.

const char *name

Nombre del tipo, utilizado para establecer **`PyTypeObject.tp_name`**.

int basicsize

Si es positivo, especifica el tamaño de la instancia en bytes. Se utiliza para establecer **`PyTypeObject.tp_basicsize`**.

Si es cero, especifica que **`tp_basicsize`** debe ser heredado.

If negative, the absolute value specifies how much space instances of the class need *in addition* to the superclass. Use **`PyObject_GetTypeData()`** to get a pointer to subclass-specific memory reserved this way. For negative **`basicsize`**, Python will insert padding when needed to meet **`tp_basicsize`**'s alignment requirements.

Distinto en la versión 3.12: Anteriormente, este campo no podía ser negativo.

int itemsize

Tamaño de un elemento de un tipo de tamaño variable, en bytes. Utilizado para establecer **`PyTypeObject.tp_itemsize`**. Consulte la documentación de `tp_itemsize` para conocer las advertencias.

Si es cero, **`tp_itemsize`** se hereda. Extender clases de tamaño variable arbitrario es peligroso, ya que algunos tipos usan un desplazamiento fijo para memoria de tamaño variable, que puede entonces superponerse con memoria de tamaño fijo usada por una subclase. Para ayudar a prevenir errores, heredar `itemsize` sólo es posible en las siguientes situaciones:

- La base no es de tamaño variable (su **`tp_itemsize`**).
- El **`PyType_Spec.basicsize`** solicitado es positivo, sugiriendo que la distribución de memoria de la clase base es conocida.
- El **`PyType_Spec.basicsize`** solicitado es cero, sugiriendo que la subclase no accede a la

memoria de la instancia directamente.

- Con la bandera **`Py_TPFLAGS_ITEMS_AT_END`**.

unsigned int flags

Banderas (*flags*) del tipo, que se usan para establecer **`PyTypeObject.tp_flags`**.

Si el indicador `Py_TPFLAGS_HEAPTYPE` no está establecido, **`PyType_FromSpecWithBases()`** lo establece automáticamente.

PyType_Slot *slots

Arreglo de estructuras **`PyType_Slot`**. Terminado por el valor de ranura especial `{0, NULL}`.

Cada ID de ranura debe especificarse como máximo una vez.

type PyType_Slot

Part of the Stable ABI (including all members).
Estructura que define la funcionalidad opcional de un tipo, que contiene una ranura ID y un puntero de valor.

int slot

Una ranura ID.

Las ranuras IDs se nombran como los nombres de campo de las estructuras **`PyTypeObject`**, **`PyNumberMethods`**, **`PySequenceMethods`**, **`PyMappingMethods`** y **`PyAsyncMethods`** con un prefijo `Py_` agregado. Por ejemplo, use:

- **`Py_tp_dealloc`** to set **`PyTypeObject.tp_dealloc`**
- **`Py_nb_add`** to set **`PyNumberMethods.nb_add`**
- **`Py_sq_length`** to set **`PySequenceMethods.sq_length`**

An additional slot is supported that does not correspond to a **`PyTypeObject`** struct field:

- **`Py_tp_token`**

The following “offset” fields cannot be set using **`PyType_Slot`**:

- **`tp_weaklistoffset`** (use **`Py_TPFLAGS_MANAGED_WEAKREF`** en su lugar si es posible)
- **`tp_dictoffset`** (use **`Py_TPFLAGS_MANAGED_DICT`** en su lugar si es posible)
- **`tp_vectorcall_offset`** (use `"__vectorcalloffset__"` en PyMemberDef)

If it is not possible to switch to a `MANAGED` flag (for example, for vectorcall or to support Python older than 3.12), specify the offset in **`Py_tp_members`**. See PyMemberDef documentation for details.

The following internal fields cannot be set at all when creating a heap type:

- **`tp_dict`**, **`tp_mro`**, **`tp_cache`**, **`tp_subclasses`**, and **`tp_weaklist`**.

Establecer **`Py_tp_bases`** o **`Py_tp_base`** puede ser problemático en algunas plataformas. Para evitar problemas, use el argumento *bases* de **`PyType_FromSpecWithBases()`** en su lugar.

Distinto en la versión 3.9: Las ranuras en **`PyBufferProcs`** se pueden configurar en la API ilimitada.

Distinto en la versión 3.11: **`bf_getbuffer`** y **`bf_releasebuffer`** ahora están disponibles bajo la API limitada.

Distinto en la versión 3.14: The field **`tp_vectorcall`** can now be set using **`Py_tp_vectorcall`**. See the field’s documentation for details.

void *pfunc

El valor deseado de la ranura. En la mayoría de los casos, este es un puntero a una función.

pfunc values may not be `NULL`, except for the following slots:

- **`Py_tp_doc`**
- **`Py_tp_token`** (for clarity, prefer **`Py_TP_USE_SPEC`** rather than `NULL`)

Py_tp_token

Part of the Stable ABI since version 3.14.
A **`slot`** that records a static memory layout ID for a class.

If the **`PyType_Spec`** of the class is statically allocated, the token can be set to the spec using the special value **`Py_TP_USE_SPEC`**:

```
static PyType_Slot foo_slots[] = {
   {Py_tp_token, Py_TP_USE_SPEC},
```

It can also be set to an arbitrary pointer, but you must ensure that:

- The pointer outlives the class, so it's not reused for something else while the class exists.
- It «belongs» to the extension module where the class lives, so it will not clash with other extensions.

Use **`PyType_GetBaseByToken()`** to check if a class's superclass has a given token – that is, check whether the memory layout is compatible.

To get the token for a given class (without considering superclasses), use **`PyType_GetSlot()`** with `Py_tp_token`.

Added in version 3.14.

Py_TP_USE_SPEC

Part of the Stable ABI since version 3.14.
Used as a value with **`Py_tp_token`** to set the token to the class's **`PyType_Spec`**. Expands to `NULL`.

Added in version 3.14.

El objeto `None`

Ten en cuenta que el **`PyTypeObject`** para `None` no está expuesto directamente en la API de Python/C. Dado que `None` es un singleton, basta con comprobar la identidad del objeto (usando `==` en C). No existe una función **`PyNone_Check()`** por la misma razón.

PyObject *Py_None

El objeto Python `None`, que denota falta de valor. Este objeto no tiene métodos y es immortal.

Distinto en la versión 3.12: **`Py_None`** es immortal.

Py_RETURN_NONE

Retorna **`Py_None`** desde una función.

Objetos enteros

Todos los enteros se implementan como objetos enteros «largos» (*long*) de tamaño arbitrario.

En caso de error, la mayoría de las API `PyLong_As*` retornan `(tipo de retorno) -1` que no se puede distinguir de un número. Use **`PyErr_Occurred()`** para desambiguar.

type PyLongObject

Part of the Limited API (as an opaque struct).
Este subtipo de **`PyObject`** representa un objeto entero de Python.

PyTypeObject PyLong_Type

Part of the Stable ABI.
Esta instancia de **`PyTypeObject`** representa el tipo entero de Python. Este es el mismo objeto que **`int`** en la capa de Python.

int PyLong_Check(PyObject *p)

Retorna verdadero si su argumento es un **`PyLongObject`** o un subtipo de **`PyLongObject`**. Esta función siempre finaliza con éxito.

int PyLong_CheckExact(PyObject *p)

Retorna verdadero si su argumento es un **`PyLongObject`**, pero no un subtipo de **`PyLongObject`**. Esta función siempre finaliza con éxito.

PyObject *PyLong_FromLong(long v)

Return value: New reference. Part of the Stable ABI.
Retorna un objeto **`PyLongObject`** nuevo desde *v*, o `NULL` en caso de error.

Detalles de implementación de CPython: CPython keeps an array of integer objects for all integers between `-5` and `256`. When you create an int in that range you actually just get back a reference to the existing object.

PyObject *PyLong_FromUnsignedLong(unsigned long v)

Return value: New reference. Part of the Stable ABI.
Retorna un objeto **`PyLongObject`** nuevo desde un C unsigned long, o `NULL` en caso de error.

PyObject *PyLong_FromSsize_t(Py_ssize_t v)

Return value: New reference. Part of the Stable ABI.
Retorna un objeto **`PyLongObject`** nuevo desde un C **`Py_ssize_t`**, o `NULL` en caso de error.

PyObject *PyLong_FromSize_t(size_t v)

Return value: New reference. Part of the Stable ABI.
Retorna un objeto **`PyLongObject`** nuevo desde un C **`size_t`**, o `NULL` en caso de error.

PyObject *PyLong_FromLongLong(long long v)

Return value: New reference. Part of the Stable ABI.
Retorna un objeto **`PyLongObject`** nuevo desde un C long long, o `NULL` en caso de error.

PyObject *PyLong_FromInt32(int32_t value)
PyObject *PyLong_FromInt64(int64_t value)

Part of the Stable ABI since version 3.14.
Return a new **`PyLongObject`** object from a signed C int32_t or int64_t, or `NULL` with an exception set on failure.

Added in version 3.14.

PyObject *PyLong_FromUnsignedLongLong(unsigned long long v)

Return value: New reference. Part of the Stable ABI.
Retorna un objeto **`PyLongObject`** nuevo desde un C unsigned long long, o `NULL` en caso de error.

PyObject *PyLong_FromUInt32(uint32_t value)

PyObject *PyLong_FromUInt64(uint64_t value)

Part of the Stable ABI since version 3.14.
Return a new **`PyLongObject`** object from an unsigned C uint32_t or uint64_t, or `NULL` with an exception set on failure.

Added in version 3.14.

PyObject *PyLong_FromDouble(double v)

Return value: New reference. Part of the Stable ABI.
Retorna un nuevo objeto **`PyLongObject`** de la parte entera de *v*, o `NULL` en caso de error.

PyObject *PyLong_FromString(const char *str, char **pend, int base)

Return value: New reference. Part of the Stable ABI.
Retorna un nuevo **`PyLongObject`** basado en el valor de cadena de caracteres en *str*, que se interpreta de acuerdo con la raíz en *base*, o `NULL` en caso de error. Si *pend* no es `NULL`, **pend* apuntará al final de *str* en caso de éxito o al primer carácter que no se pudo procesar en caso de error. Si *base* es `0`, *str* se interpreta utilizando la definición Literales enteros; en este caso, los ceros a la izquierda en un número decimal distinto de cero lanzan un **`ValueError`**. Si *base* no es `0`, debe estar entre `2` y `36`, inclusive. Se ignoran los espacios iniciales y finales y los guiones bajos individuales después de un especificador base y entre dígitos. Si no hay dígitos o *str* no termina en NULL después de los dígitos y los espacios finales, se lanzará **`ValueError`**.

Ver también

`PyLong_AsNativeBytes()` and **`PyLong_FromNativeBytes()`** functions can be used to convert a **`PyLongObject`** to/from an array of bytes in base `256`.

PyObject *PyLong_FromUnicodeObject(PyObject *u, int base)

Return value: New reference.
Convierte una secuencia de dígitos Unicode en la cadena de caracteres *u* en un valor entero de Python.

Added in version 3.3.

PyObject *PyLong_FromVoidPtr(void *p)

Return value: New reference. Part of the Stable ABI.
Crea un entero de Python desde el puntero *p*. El valor del puntero se puede recuperar del valor resultante usando **`PyLong_AsVoidPtr()`**.

PyObject *PyLong_FromNativeBytes(const void *buffer, size_t n_bytes, int flags)

Part of the Stable ABI since version 3.14.
Crea un entero de Python desde el valor contenido en los primeros *n_bytes* de *buffer*, interpretado como un número con signo del complemento a dos.

flags son los mismos que para **`PyLong_AsNativeBytes()`**. Al pasar `-1` seleccionará el endian nativo con el que CPython fue compilado y asumirá que el bit más significativo es un bit de signo. Al pasar `Py_ASNATIVEBYTES_UNSIGNED_BUFFER` producirá el mismo resultado que llamar a **`PyLong_FromUnsignedNativeBytes()`**. Otros indicadores se ignoran.

Added in version 3.13.

PyObject *PyLong_FromUnsignedNativeBytes(const void *buffer, size_t n_bytes, int flags)

Part of the Stable ABI since version 3.14.
Crea un entero de Python desde el valor contenido en los primeros *n_bytes* de *buffer*, interpretado como un número sin signo.

flags son los mismos que para `PyLong_AsNativeBytes()`. Al pasar `-1` seleccionará el endian nativo con el que CPython fue compilado y asumirá que el bit más significativo no es un bit con signo. Los indicadores que no sean endian se ignoran.

Added in version 3.13.

PyLong_FromPid(pid)

Macro for creating a Python integer from a process identifier.

This can be defined as an alias to `PyLong_FromLong()` or `PyLong_FromLongLong()`, depending on the size of the system's PID type.

Added in version 3.2.

long PyLong_AsLong(PyObject *obj)

Part of the Stable ABI.
Retorna una representación C long de *obj*. Si *obj* no es una instancia de `PyLongObject`, primero llama a su método `__index__()` (si está presente) para convertirlo en un **`PyLongObject`**.

Lanza `OverflowError` si el valor de *obj* está fuera de rango para un long.

Retorna `-1` en caso de error. Use `PyErr_Occurred()` para desambiguar.

Distinto en la versión 3.8: Use `__index__()` si está disponible.

Distinto en la versión 3.10: Esta función no usará más `__int__()`.

long PyLong_AS_LONG(PyObject *obj)

Exactly equivalent to the preferred `PyLong_AsLong`. In particular, it can fail with `OverflowError` or another exception.

Soft deprecated since version 3.14.

int PyLong_AsInt(PyObject *obj)

Part of the Stable ABI since version 3.13.

Similar a `PyLong_AsLong()`, pero almacena el resultado en un C int en lugar de un C long.

Added in version 3.13.

long PyLong_AsLongAndOverflow(PyObject *obj, int *overflow)

Part of the Stable ABI.

Retorna una representación C long de *obj*. Si *obj* no es una instancia de `PyLongObject`, primero llama a su método `__index__()` (si está presente) para convertirlo en un **`PyLongObject`**.

Si el valor de *obj* es mayor que **`LONG_MAX`** o menor que **`LONG_MIN`**, establece **overflow* en `1` o `-1`, respectivamente, y retorna `-1`; de lo contrario, establece **overflow* en `0`. Si se produce alguna otra excepción, configura **overflow* en `0` y retorna `-1` como de costumbre.

Retorna `-1` en caso de error. Use `PyErr_Occurred()` para desambiguar.

Distinto en la versión 3.8: Use `__index__()` si está disponible.

Distinto en la versión 3.10: Esta función no usará más `__int__()`.

long long PyLong_AsLongLong(PyObject *obj)

Part of the Stable ABI.

Retorna una representación C long long de *obj*. Si *obj* no es una instancia de `PyLongObject`, primero llame a su método `__index__()` (si está presente) para convertirlo en un **`PyLongObject`**.

Lanza `OverflowError` si el valor de *obj* está fuera de rango para un long long.

Retorna `-1` en caso de error. Use **`PyErr_Occurred()`** para desambiguar.

Distinto en la versión 3.8: Use **`__index__()`** si está disponible.

Distinto en la versión 3.10: Esta función no usará más **`__int__()`**.

long long PyLong_AsLongLongAndOverflow(PyObject *obj, int *overflow)

Part of the Stable ABI.
Retorna una representación C long long de *obj*. Si *obj* no es una instancia de **`PyLongObject`**, primero llame a su método **`__index__()`** (si está presente) para convertirlo en un **`PyLongObject`**.

Si el valor de *obj* es mayor que **`LLONG_MAX`** o menor que **`LLONG_MIN`**, establece **overflow* en `1` o `-1`, respectivamente, y retorna `-1`; de lo contrario, establece **overflow* en `0`. Si se produce alguna otra excepción, configura **overflow* en `0` y retorna `-1` como de costumbre.

Retorna `-1` en caso de error. Use **`PyErr_Occurred()`** para desambiguar.

Added in version 3.2.

Distinto en la versión 3.8: Use **`__index__()`** si está disponible.

Distinto en la versión 3.10: Esta función no usará más **`__int__()`**.

Py_ssize_t PyLong_AsSsize_t(PyObject *pylong)

Part of the Stable ABI.
Retorna una representación de C **`Py_ssize_t`** de *pylong*. *pylong* debe ser una instancia de **`PyLongObject`**.

Lanza `OverflowError` si el valor de *pylong* está fuera de rango para un `Py_ssize_t`.

Retorna `-1` en caso de error. Use `PyErr_Occurred()` para desambiguar.

unsigned long PyLong_AsUnsignedLong(PyObject *pylong)

Part of the Stable ABI.
Retorna una representación de C unsigned long de *pylong*. *pylong* debe ser una instancia de `PyLongObject`.

Lanza `OverflowError` si el valor de *pylong* está fuera de rango para un unsigned long.

Retorna `(unsigned long)-1` en caso de error. Use `PyErr_Occurred()` para desambiguar.

size_t PyLong_AsSize_t(PyObject *pylong)

Part of the Stable ABI.
Retorna una representación de C **`size_t`** de *pylong*. *pylong* debe ser una instancia de `PyLongObject`.

Lanza `OverflowError` si el valor de *pylong* está fuera de rango para un **`size_t`**.

Retorna `(size_t) -1` en caso de error. Use `PyErr_Occurred()` para desambiguar.

unsigned long long PyLong_AsUnsignedLongLong(PyObject *pylong)

Part of the Stable ABI.
Retorna una representación de C unsigned long long de *pylong*. *pylong* debe ser una instancia de `PyLongObject`.

Lanza `OverflowError` si el valor de *pylong* está fuera de rango para un unsigned long long.

Retorna `(unsigned long long) -1` en caso de error. Use `PyErr_Occurred()` para desambiguar.

Distinto en la versión 3.1: Ahora un *pylong* negativo lanza un **`OverflowError`**, no **`TypeError`**.

unsigned long PyLong_AsUnsignedLongMask(PyObject *obj)

Part of the Stable ABI.
Retorna una representación C unsigned long de *obj*. Si *obj* no es una instancia de **`PyLongObject`**, primero llame a su método **`__index__()`** (si está presente) para convertirlo en un **`PyLongObject`**.

Si el valor de *obj* está fuera del rango para unsigned long, retorna la reducción de ese valor módulo `ULONG_MAX + 1`.

Retorna `(unsigned long)-1` en caso de error. Use **`PyErr_Occurred()`** para desambiguar.

Distinto en la versión 3.8: Use **`__index__()`** si está disponible.

Distinto en la versión 3.10: Esta función no usará más **`__int__()`**.

unsigned long long PyLong_AsUnsignedLongLongMask(PyObject *obj)

Part of the Stable ABI.
Retorna una representación C unsigned long long de *obj*. Si *obj* no es una instancia de **`PyLongObject`**, primero llame a su método **`__index__()`** (si está presente) para convertirlo en un **`PyLongObject`**.

Si el valor de *obj* está fuera del rango para unsigned long long, retorna la reducción de ese valor módulo `ULLONG_MAX + 1`.

Retorna `(unsigned long long) -1` por error. Use **`PyErr_Occurred()`** para desambiguar.

Distinto en la versión 3.8: Use **`__index__()`** si está disponible.

Distinto en la versión 3.10: Esta función no usará más `__int__()`.

int PyLong_AsInt32(PyObject *obj, int32_t *value)
int PyLong_AsInt64(PyObject *obj, int64_t *value)

Part of the Stable ABI since version 3.14.
Set **value* to a signed C int32_t or int64_t representation of *obj*.

If *obj* is not an instance of **`PyLongObject`**, first call its **`__index__()`** method (if present) to convert it to a **`PyLongObject`**.

If the *obj* value is out of range, raise an **`OverflowError`**.

Set **value* and return `0` on success. Set an exception and return `-1` on error.

value must not be `NULL`.

Added in version 3.14.

int PyLong_AsUInt32(PyObject *obj, uint32_t *value)
int PyLong_AsUInt64(PyObject *obj, uint64_t *value)

Part of the Stable ABI since version 3.14.
Set **value* to an unsigned C uint32_t or uint64_t representation of *obj*.

If *obj* is not an instance of **`PyLongObject`**, first call its **`__index__()`** method (if present) to convert it to a **`PyLongObject`**.

- If *obj* is negative, raise a **`ValueError`**.
- If the *obj* value is out of range, raise an **`OverflowError`**.

Set **value* and return `0` on success. Set an exception and return `-1` on error.

value must not be `NULL`.

Added in version 3.14.

double PyLong_AsDouble(PyObject *pylong)

Part of the Stable ABI.
Retorna una representación de C double de *pylong*. *pylong* debe ser una instancia de **`PyLongObject`**.

Lanza **`OverflowError`** si el valor de *pylong* está fuera de rango para un double.

Retorna `-1.0` en caso de error. Use **`PyErr_Occurred()`** para desambiguar.

void *PyLong_AsVoidPtr(PyObject *pylong)

Part of the Stable ABI.
Convierte un entero Python *pylong* en un puntero C void. Si *pylong* no se puede convertir, se lanzará un **`OverflowError`**. Esto solo se garantiza para producir un puntero utilizable void para valores creados con **`PyLong_FromVoidPtr()`**.

Retorna `NULL` en caso de error. Use **`PyErr_Occurred()`** para desambiguar.

Py_ssize_t PyLong_AsNativeBytes(PyObject *pylong, void *buffer, Py_ssize_t n_bytes, int flags)

Part of the Stable ABI since version 3.14.
Copia el valor entero de Python *pylong* a un *buffer* nativo de tamaño *n_bytes*. Los *flags* pueden establecerse en `-1` para un comportamiento similar a una conversión de C, o en los valores documentados a continuación para controlar el comportamiento.

Retorna `-1` con una excepción lanzada en caso de error. Esto puede ocurrir si *pylong* no se puede interpretar como un entero, o si *pylong* era negativo y se configuró el indicador `Py_ASNATIVEBYTES_REJECT_NEGATIVE`.

Otherwise, returns the number of bytes required to store the value. If this is equal to or less than *n_bytes*, the entire value was copied. All *n_bytes* of the buffer are written: remaining bytes filled by copies of the sign bit.

If the returned value is greater than *n_bytes*, the value was truncated: as many of the lowest bits of the value as could fit are written, and the higher bits are ignored. This matches the typical behavior of a C-style downcast.

Nota

El desbordamiento no se considera un error. Si el valor que se retorna es mayor que *n_bytes*, se descartan los bits más significativos.

`0` nunca será retornado.

Los valores siempre se copian como complemento a dos.

Ejemplo de uso:

```
int32_t value;
Py_ssize_t bytes = PyLong_AsNativeBytes(pylong, &va
if (bytes < 0) {
    // Error. Se estableció una excepción de Python
    return NULL;
}
else if (bytes <= (Py_ssize_t)sizeof(value)) {
    // ¡Éxito!
}
else {
    // Se produjo un desbordamiento, pero 'value' c
    // los bits más bajos truncados de pylong.
}
```

Al pasar cero a *n_bytes* retornará el tamaño de un búfer lo suficientemente grande como para contener el valor. Este tamaño puede ser mayor de lo técnicamente necesario, pero no excesivamente grande. Si *n_bytes=0*, *buffer* puede ser `NULL`.

Nota

Al pasar *n_bytes=0* a esta función no es una forma precisa de determinar la longitud en bits del valor.

Para obtener el valor completo de Python de un tamaño desconocido, la función se puede llamar dos veces: primero para determinar el tamaño del búfer y luego para llenarlo:

```
// Pide cuánto espacio necesitamos.
Py_ssize_t expected = PyLong_AsNativeBytes(pylong, 
if (expected < 0) {
    // Error. Se estableció una excepción de Python
    return NULL;
}
assert(expected != 0);  // Imposible según la defin
uint8_t *bignum = malloc(expected);
if (!bignum) {
    PyErr_SetString(PyExc_MemoryError, "bignum mall
    return NULL;
}
// Obtiene el valor completo de forma segura.
Py_ssize_t bytes = PyLong_AsNativeBytes(pylong, big
if (bytes < 0) {  // Se configuró una excepción.
    free(bignum);
    return NULL;
}
else if (bytes > expected) {  // Esto no debería se
    PyErr_SetString(PyExc_RuntimeError,
        "Unexpected bignum truncation after a size 
    free(bignum);
    return NULL;
}
// El éxito esperado dada la comprobación previa.
// ... use bignum ...
free(bignum);
```

flags is either `-1` (`Py_ASNATIVEBYTES_DEFAULTS`) to select defaults that behave most like a C cast, or a combination of the other flags in the table below. Note that `-1` cannot be combined with other flags.

Actualmente, `-1` corresponde a `Py_ASNATIVEBYTES_NATIVE_ENDIAN | Py_ASNATIVEBYTES_UNSIGNED_BUFFER`.

Indicador	**Valor**
Py_ASNATIVEBYTES_DEFAULTS *Part of the Stable ABI since version 3.14.*	-1
Py_ASNATIVEBYTES_BIG_ENDIAN *Part of the Stable ABI since version 3.14.*	0
Py_ASNATIVEBYTES_LITTLE_ENDIAN *Part of the Stable ABI since version 3.14.*	1
Py_ASNATIVEBYTES_NATIVE_ENDIAN *Part of the Stable ABI since version 3.14.*	3
Py_ASNATIVEBYTES_UNSIGNED_BUFFER *Part of the Stable ABI since version 3.14.*	4
Py_ASNATIVEBYTES_REJECT_NEGATIVE *Part of the Stable ABI since version 3.14.*	8
Py_ASNATIVEBYTES_ALLOW_INDEX *Part of the Stable ABI since version 3.14.*	16

Especificar `Py_ASNATIVEBYTES_NATIVE_ENDIAN` redefinirá cualquier otro indicador endian. Pasar `2` está reservado.

Por defecto, se solicitará búfer suficiente para incluir un bit de signo. Por ejemplo, al convertir 128 con $n\_bytes=1$, la función retornará 2 (o más) para almacenar un bit de signo cero.

Si se especifica `Py_ASNATIVEBYTES_UNSIGNED_BUFFER`, se omitirá el bit de signo cero en los cálculos de tamaño. Esto permite, por ejemplo, que quepa 128 en un búfer de un solo byte. Si el búfer de destino se trata posteriormente como con signo, un valor de entrada positivo puede volverse negativo. Tenga en cuenta que este indicador no afecta el manejo de valores negativos: para ellos, siempre se solicita espacio para un bit de signo.

Especificar `Py_ASNATIVEBYTES_REJECT_NEGATIVE`, se establece una excepción si *pylong* es negativo. Sin este indicador, se copiarán los valores negativos siempre que haya suficiente espacio para al menos un bit de signo, independientemente de si se especificó `Py_ASNATIVEBYTES_UNSIGNED_BUFFER`.

Si se especifica `Py_ASNATIVEBYTES_ALLOW_INDEX` y se pasa un valor distinto de un entero, se llamará primero a su método **`__index__()`**. Esto puede resultar la ejecución de código Python y la ejecución de otros subprocesos, lo que podría causar cambios en otros objetos o valores en uso. Cuando *flags* es `-1`, esta opción no se establece y los valores distintos de un entero lanzarán **`TypeError`**.

Nota

Con los *flags* predeterminados (`-1` o *UNSIGNED_BUFFER* sin *REJECT_NEGATIVE*), varios enteros de Python pueden asignarse a un único valor sin desbordamiento. Por ejemplo, tanto `255` como `-1` se ajustan a un búfer de un solo byte y configuran todos sus bits. Esto coincide con el comportamiento típico de conversión de C.

Added in version 3.13.

PyLong_AsPid(pid)

Macro for converting a Python integer into a process identifier.

This can be defined as an alias to **`PyLong_AsLong()`**, **`PyLong_FromLongLong()`**, or **`PyLong_AsInt()`**, depending on the size of the system's PID type.

Added in version 3.2.

int PyLong_GetSign(PyObject *obj, int *sign)

Get the sign of the integer object *obj*.

On success, set **sign* to the integer sign (0, -1 or +1 for zero, negative or positive integer, respectively) and return 0.

On failure, return -1 with an exception set. This function always succeeds if *obj* is a **`PyLongObject`** or its subtype.

Added in version 3.14.

int PyLong_IsPositive(PyObject *obj)

Check if the integer object *obj* is positive (`obj > 0`).

If *obj* is an instance of **`PyLongObject`** or its subtype, return `1` when it's positive and `0` otherwise. Else set an exception and return `-1`.

Added in version 3.14.

int PyLong_IsNegative(PyObject *obj)

Check if the integer object *obj* is negative (`obj < 0`).

If *obj* is an instance of **`PyLongObject`** or its subtype, return `1` when it's negative and `0` otherwise. Else set an exception and return `-1`.

Added in version 3.14.

int PyLong_IsZero(PyObject *obj)

Check if the integer object *obj* is zero.

If *obj* is an instance of **`PyLongObject`** or its subtype, return `1` when it's zero and `0` otherwise. Else set an exception and return `-1`.

Added in version 3.14.

PyObject *PyLong_GetInfo(void)

Part of the Stable ABI.

En caso de éxito, retorna una named tuple de solo lectura, que contiene información sobre la representación interna de enteros en Python. Consulte `sys.int_info` para obtener una descripción de cada campo.

En caso de error, retorna `NULL` con una excepción configurada.

Added in version 3.1.

int PyUnstable_Long_IsCompact(const PyLongObject *op)

This is Unstable API. It may change without warning in minor releases.

Retorna 1 si *op* es compacto, 0 en caso contrario.

Esta función permite que el código de rendimiento crítico implemente una "ruta rápida" para enteros pequeños. Para valores compactos, utilice `PyUnstable_Long_CompactValue()`; para otros, utilice una función `PyLong_As*` o `PyLong_AsNativeBytes()`.

Se espera que la aceleración sea insignificante para la mayoría de los usuarios.

Exactamente qué valores se consideran compactos es un detalle de implementación y está sujeto a cambios.

Added in version 3.12.

Py_ssize_t PyUnstable_Long_CompactValue(const PyLongObject *op)

This is Unstable API. It may change without warning in minor releases.

Si *op* es compacto, se determina por `PyUnstable_Long_IsCompact()`, retorna su valor.

De lo contrario, el valor que retorna está indefinido.

Added in version 3.12.

Export API

Added in version 3.14.

struct PyLongLayout

Layout of an array of «digits» («limbs» in the GMP terminology), used to represent absolute value for arbitrary precision integers.

Use `PyLong_GetNativeLayout()` to get the native layout of Python `int` objects, used internally for integers with «big enough» absolute value.

See also `sys.int_info` which exposes similar information in Python.

uint8_t bits_per_digit

Bits per digit. For example, a 15 bit digit means that bits 0-14 contain meaningful information.

uint8_t digit_size

Digit size in bytes. For example, a 15 bit digit will require at least 2 bytes.

int8_t digits_order

Digits order:

- `1` for most significant digit first
- `-1` for least significant digit first

int8_t digit_endianness

Digit endianness:

- `1` for most significant byte first (big endian)
- `-1` for least significant byte first (little endian)

const PyLongLayout *PyLong_GetNativeLayout(void)

Get the native layout of Python `int` objects.

See the `PyLongLayout` structure.

The function must not be called before Python initialization nor after Python finalization. The returned layout is valid until Python is finalized. The layout is the same for all Python sub-interpreters in a process, and so it can be cached.

struct PyLongExport

Export of a Python `int` object.

There are two cases:

- If `digits` is `NULL`, only use the `value` member.
- If `digits` is not `NULL`, use `negative`, `ndigits` and **`digits`** members.

int64_t value

The native integer value of the exported `int` object. Only valid if `digits` is `NULL`.

uint8_t negative

`1` if the number is negative, `0` otherwise. Only valid if `digits` is not `NULL`.

Py_ssize_t ndigits

Number of digits in `digits` array. Only valid if **`digits`** is not `NULL`.

const void *digits

Read-only array of unsigned digits. Can be `NULL`.

int PyLong_Export(PyObject *obj, PyLongExport *export_long)

Export a Python `int` object.

export_long must point to a `PyLongExport` structure allocated by the caller. It must not be `NULL`.

On success, fill in **export_long* and return `0`. On error, set an exception and return `-1`.

`PyLong_FreeExport()` must be called when the export is no longer needed.

> **Detalles de implementación de CPython:** This function always succeeds if *obj* is a Python **`int`** object or a subclass.

void PyLong_FreeExport(PyLongExport *export_long)

Release the export *export_long* created by **`PyLong_Export()`**.

Detalles de implementación de CPython: Calling **`PyLong_FreeExport()`** is optional if *export_long->digits* is `NULL`.

PyLongWriter API

The **`PyLongWriter`** API can be used to import an integer.

Added in version 3.14.

struct PyLongWriter

A Python **`int`** writer instance.

The instance must be destroyed by **`PyLongWriter_Finish()`** or **`PyLongWriter_Discard()`**.

PyLongWriter *PyLongWriter_Create(int negative, Py_ssize_t ndigits, void **digits)

Create a **`PyLongWriter`**.

On success, allocate **digits* and return a writer. On error, set an exception and return `NULL`.

negative is `1` if the number is negative, or `0` otherwise.

ndigits is the number of digits in the *digits* array. It must be greater than 0.

digits must not be NULL.

After a successful call to this function, the caller should fill in the array of digits *digits* and then call **`PyLongWriter_Finish()`** to get a Python **`int`**. The layout of *digits* is described by **`PyLong_GetNativeLayout()`**.

Digits must be in the range [`0`; `(1 << bits_per_digit) - 1`] (where the **`bits_per_digit`** is the number of bits per digit). Any unused most significant digits must be set to `0`.

Alternately, call **`PyLongWriter_Discard()`** to destroy the writer instance without creating an **`int`** object.

PyObject *PyLongWriter_Finish(PyLongWriter *writer)

Return value: New reference.
Finish a **`PyLongWriter`** created by **`PyLongWriter_Create()`**.

On success, return a Python **`int`** object. On error, set an exception and return `NULL`.

The function takes care of normalizing the digits and converts the object to a compact integer if needed.

The writer instance and the *digits* array are invalid after the call.

void PyLongWriter_Discard(PyLongWriter *writer)

Discard a **`PyLongWriter`** created by **`PyLongWriter_Create()`**.

If *writer* is `NULL`, no operation is performed.

The writer instance and the *digits* array are invalid after the call.

Deprecated API

These macros are soft deprecated. They describe parameters of the internal representation of `PyLongObject` instances.

Use `PyLong_GetNativeLayout()` instead, along with `PyLong_Export()` to read integer data or `PyLongWriter` to write it. These currently use the same layout, but are designed to continue working correctly even if CPython's internal integer representation changes.

PyLong_SHIFT

This is equivalent to `bits_per_digit` in the output of `PyLong_GetNativeLayout()`.

PyLong_BASE

This is currently equivalent to 1 << PyLong_SHIFT.

PyLong_MASK

This is currently equivalent to (1 << PyLong_SHIFT) - 1

Objetos booleanos

Los booleanos en Python se implementan como una subclase de enteros. Solo hay dos booleanos **`Py_False`** y **`Py_True`**. Como tal, las funciones normales de creación y eliminación no se aplican a los booleanos. Sin embargo, los siguientes macros están disponibles.

PyTypeObject PyBool_Type

Part of the Stable ABI.

Esta instancia de **`PyTypeObject`** representa el tipo booleano de Python; es el mismo objeto que **`bool`** en la capa de Python.

int PyBool_Check(PyObject *o)

Retorna verdadero si *o* es de tipo **`PyBool_Type`**. Esta función siempre finaliza con éxito.

PyObject *Py_False

The Python `False` object. This object has no methods and is immortal.

Distinto en la versión 3.12: **`Py_False`** is immortal.

PyObject *Py_True

The Python `True` object. This object has no methods and is immortal.

Distinto en la versión 3.12: **`Py_True`** is immortal.

Py_RETURN_FALSE

Retorna **`Py_False`** desde una función.

Py_RETURN_TRUE

Retorna **`Py_True`** desde una función..

PyObject *PyBool_FromLong(long v)

Return value: New reference. Part of the Stable ABI.
Retorna `Py_True` o `Py_False`, dependiendo del valor verdadero de *v*.

Floating-Point Objects

type PyFloatObject

This subtype of `PyObject` represents a Python floating-point object.

PyTypeObject PyFloat_Type

Part of the Stable ABI.
This instance of `PyTypeObject` represents the Python floating-point type. This is the same object as `float` in the Python layer.

int PyFloat_Check(PyObject *p)

Retorna verdadero si su argumento es un `PyFloatObject` o un subtipo de `PyFloatObject`. Esta función siempre finaliza con éxito.

int PyFloat_CheckExact(PyObject *p)

Retorna verdadero si su argumento es un `PyFloatObject`, pero no un subtipo de `PyFloatObject`. Esta función siempre finaliza con éxito.

PyObject *PyFloat_FromString(PyObject *str)

Return value: New reference. Part of the Stable ABI.
Crea un objeto `PyFloatObject` basado en la cadena de caracteres en *str*, o `NULL` en caso de error.

PyObject *PyFloat_FromDouble(double v)

Return value: New reference. Part of the Stable ABI.
Crea un objeto `PyFloatObject` a partir de *v*, o `NULL` en caso de error.

double PyFloat_AsDouble(PyObject *pyfloat)

Part of the Stable ABI.
Return a C double representation of the contents of *pyfloat*. If *pyfloat* is not a Python floating-point object but has a **`__float__()`** method, this method will first be called to convert *pyfloat* into a float. If **`__float__()`** is not defined then it falls back to **`__index__()`**. This method returns `-1.0` upon failure, so one should call **`PyErr_Occurred()`** to check for errors.

Distinto en la versión 3.8: Utilice **`__index__()`** si está disponible.

double PyFloat_AS_DOUBLE(PyObject *pyfloat)

Retorna una representación C double de los contenidos de *pyfloat*, pero sin verificación de errores.

PyObject *PyFloat_GetInfo(void)

Return value: New reference. Part of the Stable ABI.
Retorna una instancia de *structseq* que contiene información sobre la precisión, los valores mínimos y máximos de un flotante. Es un contenedor reducido alrededor del archivo de encabezado `float.h`.

double PyFloat_GetMax()

Part of the Stable ABI.
Retorna el máximo flotante finito representable *DBL_MAX* como C double.

double PyFloat_GetMin()

Part of the Stable ABI.
Retorna el flotante positivo normalizado mínimo *DBL_MIN* como C double.

Py_INFINITY

This macro expands to a constant expression of type double, that represents the positive infinity.

On most platforms, this is equivalent to the **`INFINITY`**

macro from the C11 standard `<math.h>` header.

Py_NAN

This macro expands to a constant expression of type double, that represents a quiet not-a-number (qNaN) value.

On most platforms, this is equivalent to the **`NAN`** macro from the C11 standard `<math.h>` header.

Py_HUGE_VAL

Equivalent to **`INFINITY`**.

Obsoleto desde la versión 3.14: The macro is soft deprecated.

Py_MATH_E

The definition (accurate for a double type) of the **`math.e`** constant.

Py_MATH_El

High precision (long double) definition of **`e`** constant.

Py_MATH_PI

The definition (accurate for a double type) of the **`math.pi`** constant.

Py_MATH_PIl

High precision (long double) definition of **`pi`** constant.

Py_MATH_TAU

The definition (accurate for a double type) of the **`math.tau`** constant.

Added in version 3.6.

Py_RETURN_NAN

Return **`math.nan`** from a function.

On most platforms, this is equivalent to `return`

```
PyFloat_FromDouble(NAN).
```

Py_RETURN_INF(sign)

Return **`math.inf`** or **`-math.inf`** from a function, depending on the sign of *sign*.

On most platforms, this is equivalent to the following:

```
return PyFloat_FromDouble(copysign(INFINITY, sign))
```

Py_IS_FINITE(X)

Return `1` if the given floating-point number *X* is finite, that is, it is normal, subnormal or zero, but not infinite or NaN. Return `0` otherwise.

Obsoleto desde la versión 3.14: The macro is soft deprecated. Use **`isfinite`** instead.

Py_IS_INFINITY(X)

Return `1` if the given floating-point number *X* is positive or negative infinity. Return `0` otherwise.

Obsoleto desde la versión 3.14: The macro is soft deprecated. Use **`isinf`** instead.

Py_IS_NAN(X)

Return `1` if the given floating-point number *X* is a not-a-number (NaN) value. Return `0` otherwise.

Obsoleto desde la versión 3.14: The macro is soft deprecated. Use **`isnan`** instead.

Funciones de empaquetado y desempaquetado

Las funciones de empaquetar y desempaquetar proporcionan una manera eficiente e independiente de la plataforma para almacenar valores de coma flotante como cadenas de bytes. Las rutinas Pack producen una cadena de bytes a partir de un C double, y las rutinas Desempaquetar producen un C double a partir de dicha cadena de bytes. El sufijo (2, 4 u 8) especifica el número de bytes en la cadena de bytes.

En plataformas que parecen usar formatos IEEE 754, estas funciones actúan copiando los bits. En otras plataformas, el formato 2-byte es idéntico al formato de media precision IEEE 754 binary16, el formato de 4-byte (32 bits) es idéntico al formato de precisión simple binario IEEE 754 binary32, y el formato de 8-byte al formato de doble precisión binario IEEE 754 binary64, aunque el empaquetado de INFs y NaNs (si existen en la plataforma) no se maneja correctamente, mientras que intentar desempaquetar una cadena de bytes que contenga un IEEE INF o NaN generará una excepción.

Note that NaN type may not be preserved on IEEE platforms (signaling NaNs become quiet NaNs), for example on x86 systems in 32-bit mode.

En plataformas que no son IEEE con más precisión, o mayor rango dinámico, que el IEEE 754 admite, no se pueden empaquetar todos los valores; en plataformas que no son IEEE con menos precisión o con un rango dinámico más pequeño, no se pueden desempaquetar todos los valores. Lo que sucede en tales casos es en parte accidental (desafortunadamente).

Added in version 3.11.

Funciones de Empaquetado

The pack routines write 2, 4 or 8 bytes, starting at *p*. *le* is an int argument, non-zero if you want the bytes string in little-endian format (exponent last, at `p+1`, `p+3`, or `p+6` and `p+7`), zero if you want big-endian format (exponent first, at *p*). The **`PY_BIG_ENDIAN`** constant can be used to use the native endian: it is equal to `1` on big endian processor, or `0` on little endian processor.

Valor retornado: `0` si todo está bien, `-1` si hay error (y se establece una excepción, probablemente `OverflowError`).

Hay dos problemas en plataformas que no son IEEE:

- Lo que esto hace es indefinido si *x* es un NaN o infinito.
- `-0.0` and `+0.0` produce la misma cadena de bytes.

int PyFloat_Pack2(double x, char *p, int le)

Empaquete un C doble como el formato de media precisión IEEE 754 binary16.

int PyFloat_Pack4(double x, char *p, int le)

Empaque un C doble como el formato de precisión simple IEEE 754 binary32.

int PyFloat_Pack8(double x, char *p, int le)

Empaque un C doble como el formato de doble precisión IEEE 754 binary64.

Funciones de Desempaquetado

Las rutinas de desempaquetado leen 2, 4 u 8 bytes, comenzando en *p*. *le* es un argumento int , distinto de cero si la cadena bytes está en formato little-endian (exponente al final, en `p+1`, `p+3` o `p+6` y `p+7`), cero si está en formato big-endian (exponente primero, en *p*). La constante **`PY_BIG_ENDIAN`** se puede usar para utilizar el endian nativo: es igual a `1` en un procesador big endian, o `0` en un procesador little-endian.

Valor retornado: Doble desempaquetado. Si hay error, `-1.0` y `PyErr_Occurred()` es verdadero (y se establece una excepción, probablemente `OverflowError`).

Hay que tener en cuenta que en una plataforma que no sea IEEE, esto se negará a desempaquetar una cadena de bytes que representa un NaN o infinito.

double PyFloat_Unpack2(const char *p, int le)

Descomprima el formato de media precisión IEEE 754 binary16 como un doble C.

double PyFloat_Unpack4(const char *p, int le)

Descomprima el formato de precisión simple IEEE 754 binary32 como un doble C.

double PyFloat_Unpack8(const char *p, int le)

Descomprima el formato de doble precisión IEEE 754 binary64 como un doble C.

Objetos de números complejos

Los objetos de números complejos de Python se implementan como dos tipos distintos cuando se ven desde la API de C: uno es el objeto de Python expuesto a los programas de Python, y el otro es una estructura en C que representa el valor de número complejo real. La API proporciona funciones para trabajar con ambos.

Números complejos como estructuras C

Tenga en cuenta que las funciones que aceptan estas estructuras como parámetros y las retornan como resultados lo hacen *por valor* en lugar de desreferenciarlas a través de punteros. Esto es consistente en toda la API.

type Py_complex

La estructura C que corresponde a la porción de valor de un objeto de número complejo de Python. La mayoría de las funciones para tratar con objetos de números complejos utilizan estructuras de este tipo como valores de entrada o salida, según corresponda.

double real
double imag

La estructura se define como:

```
typedef struct {
    double real;
    double imag;
} Py_complex;
```

Py_complex _Py_c_sum(Py_complex left, Py_complex right)

Retorna la suma de dos números complejos, utilizando la representación C **`Py_complex`**.

Py_complex _Py_c_diff(Py_complex left, Py_complex right)

Retorna la diferencia entre dos números complejos, usando la representación C `Py_complex`.

Py_complex _Py_c_neg(Py_complex num)

Retorna la negación del número complejo *num*, utilizando la representación C `Py_complex`.

Py_complex _Py_c_prod(Py_complex left, Py_complex right)

Retorna el producto de dos números complejos, usando la representación C `Py_complex`.

Py_complex _Py_c_quot(Py_complex dividend, Py_complex divisor)

Retorna el cociente de dos números complejos, utilizando la representación C `Py_complex`.

Si *divisor* es nulo, este método retorna cero y establece `errno` en `EDOM`.

Py_complex _Py_c_pow(Py_complex num, Py_complex exp)

Retorna la exponenciación de *num* por *exp*, utilizando la representación C `Py_complex`.

Si *num* es nulo y *exp* no es un número real positivo, este método retorna cero y establece `errno` a `EDOM`.

Set `errno` to `ERANGE` on overflows.

Números complejos como objetos de Python

type PyComplexObject

Este subtipo de `PyObject` representa un objeto de número complejo de Python.

PyTypeObject PyComplex_Type

Part of the Stable ABI.
Esta instancia de **`PyTypeObject`** representa el tipo de número complejo de Python. Es el mismo objeto que **`complex`** en la capa de Python.

int PyComplex_Check(PyObject *p)

Retorna verdadero si su argumento es un **`PyComplexObject`** o un subtipo de **`PyComplexObject`**. Esta función siempre finaliza con éxito.

int PyComplex_CheckExact(PyObject *p)

Retorna verdadero si su argumento es un **`PyComplexObject`**, pero no un subtipo de **`PyComplexObject`**. Esta función siempre finaliza con éxito.

PyObject *PyComplex_FromCComplex(Py_complex v)

Return value: New reference.
Crea un nuevo objeto de número complejo de Python a partir de un valor C **`Py_complex`**. Retorna `NULL` con una excepción establecida en caso de error.

PyObject *PyComplex_FromDoubles(double real, double imag)

Return value: New reference. Part of the Stable ABI.
Retorna un nuevo objeto **`PyComplexObject`** de *real* e *imag*. Retorna `NULL` con una excepción establecida en caso de error.

double PyComplex_RealAsDouble(PyObject *op)

Part of the Stable ABI.
Retorna la parte real de *op* como double en C.

Si *op* no es un objeto de número complejo de Python pero tiene un método **`__complex__()`**, primero se llamará a este método para convertir *op* en un objeto de número complejo de Python. Si **`__complex__()`** no está definido, entonces recurre a **`PyFloat_AsDouble()`** y retorna su resultado.

En caso de falla, este método retorna `-1.0` con una excepción establecida, por lo que se debe llamar `PyErr_Occurred()` para verificar si hay errores.

Distinto en la versión 3.13: Use `__complex__()` si está disponible.

double PyComplex_ImagAsDouble(PyObject *op)

Part of the Stable ABI.

Retorna la parte imaginaria de *op* como un double de C.

Si *op* no es un objeto de número complejo de Python pero tiene un método `__complex__()`, primero se llamará a este método para convertir *op* en un objeto de número complejo de Python. Si `__complex__()` no está definido, entonces recurre a `PyFloat_AsDouble()` y retorna `0.0` en caso de éxito.

En caso de falla, este método retorna `-1.0` con una excepción establecida, por lo que se debe llamar `PyErr_Occurred()` para verificar si hay errores.

Distinto en la versión 3.13: Use `__complex__()` si está disponible.

Py_complex PyComplex_AsCComplex(PyObject *op)

Retorna el valor `Py_complex` del número complejo *op*.

Si *op* no es un objeto de número complejo de Python pero tiene un método `__complex__()`, primero se llamará a este método para convertir *op* en un objeto de número complejo de Python. Si `__complex__()` no está definido, entonces recurre a `__float__()`. Si `__float__()` no está definido, entonces recurre a `__index__()`.

En caso de falla, este método retorna `Py_complex` con `real` establecido en `-1.0` y con una excepción establecida, por lo que se debe llamar a `PyErr_Occurred()` para verificar si hay errores.

Distinto en la versión 3.8: Use `__index__()` si está disponible.

Objetos bytes

Estas funciones lanzan **`TypeError`** cuando se espera un parámetro de bytes y se llama con un parámetro que no es bytes.

type PyBytesObject

Este subtipo de **`PyObject`** representa un objeto bytes de Python.

PyTypeObject PyBytes_Type

Part of the Stable ABI.
Esta instancia de **`PyTypeObject`** representa el tipo bytes de Python; es el mismo objeto que **`bytes`** en la capa de Python.

int PyBytes_Check(PyObject *o)

Retorna verdadero si el objeto *o* es un objeto bytes o una instancia de un subtipo del tipo bytes. Esta función siempre finaliza con éxito.

int PyBytes_CheckExact(PyObject *o)

Retorna verdadero si el objeto *o* es un objeto bytes, pero no una instancia de un subtipo del tipo bytes. Esta función siempre finaliza con éxito.

PyObject *PyBytes_FromString(const char *v)

Return value: New reference. Part of the Stable ABI. Thread safety: Atomic.
Retorna un nuevo objeto bytes con una copia de la cadena de caracteres *v* como valor en caso de éxito y `NULL` en caso de error. El parámetro *v* no debe ser `NULL`; no se comprobará.

PyObject *PyBytes_FromStringAndSize(const char *v, Py_ssize_t len)

Return value: New reference. Part of the Stable ABI. Thread *safety: Atomic.*

Retorna un nuevo objeto bytes con una copia de la cadena de caracteres *v* como valor y longitud *len* en caso de éxito y `NULL` en caso de error. Si *v* es `NULL`, el contenido del objeto bytes no se inicializa.

PyObject *PyBytes_FromFormat(const char *format, ...)

Return value: New reference. Part of the Stable ABI. Thread safety: Atomic.

Toma una cadena de caracteres *format* del estilo C **`printf()`** y un número variable de argumentos, calcula el tamaño del objeto bytes Python resultante y retorna un objeto bytes con los valores formateados. Los argumentos variables deben ser tipos C y deben corresponder exactamente a los caracteres de formato en la cadena de caracteres *format*. Se permiten los siguientes caracteres de formato:

Caracteres de formato	Tipo	Comentario
`%%`	*n/a*	El carácter literal %.
`%c`	int	Un solo byte, representado como un C int.
`%d`	int	Equivalente a `printf("%d")`. [1]
`%u`	unsigned int	Equivalente a `printf("%u")`. [1]
`%ld`	long	Equivalente a `printf("%ld")`. [1]
`%lu`	unsigned long	Equivalente a `printf("%lu")`. [1]
`%zd`	`Py_ssize_t`	Equivalente a `printf("%zd")`. [1]
`%zu`	size_t	Equivalente a `printf("%zu")`. [1]
`%i`	int	Equivalente a `printf("%i")`. [1]
`%x`	int	Equivalente a `printf("%x")`. [1]
`%s`	const char*	Un arreglo de caracteres C terminados en nulo.
`%p`	const void*	La representación hexadecimal de un puntero en C. Principalmente equivalente a `printf("%p")` excepto que se garantiza que comience con el literal `0x`, independientemente de lo que produzca el `printf` de la plataforma.

Un carácter de formato no reconocido hace que todo el resto de la cadena de caracteres de formato se copie como está en el objeto de resultado y se descartan los argumentos adicionales.

[1] (1,2,3,4,5,6,7,8)
Para especificadores de enteros *(d, u, ld, lu, zd, zu, i, x)*: el indicador de conversión 0 tiene efecto incluso cuando se proporciona una precisión.

PyObject *PyBytes_FromFormatV(const char *format, va_list vargs)

Return value: New reference. Part of the Stable ABI. Thread safety: Atomic.
Idéntica a **`PyBytes_FromFormat()`** excepto que toma exactamente dos argumentos.

PyObject *PyBytes_FromObject(PyObject *o)

Return value: New reference. Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Retorna la representación en bytes del objeto *o* que implementa el protocolo de búfer.

Nota

If the object implements the buffer protocol, then the buffer must not be mutated while the bytes object is being created.

Py_ssize_t PyBytes_Size(PyObject *o)

Part of the Stable ABI. Thread safety: Atomic.
Retorna la longitud de los bytes en el objeto bytes *o*.

Py_ssize_t PyBytes_GET_SIZE(PyObject *o)

Thread safety: Atomic.
Forma macro de **`PyBytes_Size()`** pero sin verificación de errores.

char *PyBytes_AsString(PyObject *o)

Part of the Stable ABI. Thread safety: Safe to call from multiple threads with external synchronization only.
Retorna un puntero al contenido de *o*. El puntero se refiere al

búfer interno de *o*, que consiste en `len(o) + 1` bytes. El último byte en el búfer siempre es nulo, independientemente de si hay otros bytes nulos. Los datos no deben modificarse de ninguna manera, a menos que el objeto se haya creado usando `PyBytes_FromStringAndSize(NULL, size)`. No debe ser desasignado. Si *o* no es un objeto de bytes en absoluto, **`PyBytes_AsString()`** retorna `NULL` y lanza un **`TypeError`**.

char *PyBytes_AS_STRING(PyObject *string)

Thread safety: Safe to call from multiple threads with external synchronization only.
Forma macro de **`PyBytes_AsString()`** pero sin verificación de errores.

int PyBytes_AsStringAndSize(PyObject *obj, char **buffer, Py_ssize_t *length)

Part of the Stable ABI. Thread safety: Safe to call from multiple threads with external synchronization only.
Retorna los contenidos terminados en nulo del objeto *obj* a través de las variables de salida *buffer* y *length*. Retorna `0` en caso de éxito.

Si *length* es `NULL`, el objeto bytes no puede contener bytes nulos incrustados; en caso contrario, la función retorna `-1` y se lanza un **`ValueError`**.

El búfer se refiere a un búfer interno de *obj*, que incluye un byte nulo adicional al final (no está considerado en *length*). Los datos no deben modificarse de ninguna manera, a menos que el objeto se haya creado usando `PyBytes_FromStringAndSize(NULL, size)`. No debe ser desasignado. Si *obj* no es un objeto bytes en absoluto, **`PyBytes_AsStringAndSize()`** retorna `-1` y lanza **`TypeError`**.

Distinto en la versión 3.5: Anteriormente, **`TypeError`** se lanzaba cuando se encontraban bytes nulos incrustados en el objeto bytes.

void PyBytes_Concat(PyObject **bytes, PyObject *newpart)

Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Crea un nuevo objeto de bytes en **bytes* que contiene el contenido de *newpart* agregado a *bytes*; la persona que llama poseerá la nueva referencia. La referencia al valor anterior de *bytes* será robada. Si no se puede crear el nuevo objeto, la referencia anterior a *bytes* se seguirá descartando y el valor de **bytes* se establecerá en `NULL`; se establecerá la excepción apropiada.

Nota

If *newpart* implements the buffer protocol, then the buffer must not be mutated while the new bytes object is being created.

void PyBytes_ConcatAndDel(PyObject **bytes, PyObject *newpart)

Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Crea un nuevo objeto de bytes en **bytes* que contenga el contenido de *newpart* agregado a *bytes*. Esta versión libera la strong reference a *newpart* (es decir, disminuye su recuento de referencias).

Nota

If *newpart* implements the buffer protocol, then the buffer must not be mutated while the new bytes object is being created.

PyObject *PyBytes_Join(PyObject *sep, PyObject *iterable)

Thread safety: Safe for concurrent use on the same object.
Similar to `sep.join(iterable)` in Python.

sep must be Python **`bytes`** object. (Note that

`PyUnicode_Join()` accepts `NULL` separator and treats it as a space, whereas **`PyBytes_Join()`** doesn't accept `NULL` separator.)

iterable must be an iterable object yielding objects that implement the buffer protocol.

On success, return a new **`bytes`** object. On error, set an exception and return `NULL`.

Added in version 3.14.

Nota

If *iterable* objects implement the buffer protocol, then the buffers must not be mutated while the new bytes object is being created.

int _PyBytes_Resize(PyObject **bytes, Py_ssize_t newsize)

Thread safety: Safe to call without external synchronization on distinct objects.
Redimensiona un objeto bytes. *newsize* será la nueva longitud del objeto bytes. Se puede considerar como crear un nuevo objeto bytes y destruir el anterior, solo que de forma más eficiente. Pasa la dirección de un objeto bytes existente como lvalue (se puede escribir en él) y el nuevo tamaño deseado. En caso de éxito, **bytes* contiene el objeto bytes redimensionado y se retorna `0`; la dirección en **bytes* puede diferir de su valor de entrada. Si la reasignación falla, el objeto bytes original en **bytes* se desasigna, **bytes* se establece en `NULL`, se establece **`MemoryError`** y se retorna `-1`.

PyObject *PyBytes_Repr(PyObject *bytes, int smartquotes)

Part of the Stable ABI. Thread safety: Atomic.
Get the string representation of *bytes*. This function is currently used to implement **`bytes.__repr__()`** in Python.

This function does not do type checking; it is undefined behavior to pass *bytes* as a non-bytes object or `NULL`.

If *smartquotes* is true, the representation will use a double-quoted string instead of single-quoted string when single-quotes are present in *bytes*. For example, the byte string `'Python'` would be represented as `b"'Python'"` when *smartquotes* is true, or `b'\'Python\''` when it is false.

On success, this function returns a strong reference to a **`str`** object containing the representation. On failure, this returns `NULL` with an exception set.

PyObject *PyBytes_DecodeEscape(const char *s, Py_ssize_t len, const char *errors, Py_ssize_t unicode, const char *recode_encoding)

Part of the Stable ABI. Thread safety: Atomic.
Unescape a backslash-escaped string *s*. *s* must not be `NULL`. *len* must be the size of *s*.

errors must be one of `"strict"`, `"replace"`, or `"ignore"`. If *errors* is `NULL`, then `"strict"` is used by default.

On success, this function returns a strong reference to a Python **`bytes`** object containing the unescaped string. On failure, this function returns `NULL` with an exception set.

Distinto en la versión 3.9: unicode and *recode_encoding* are now unused.

Objetos de arreglos de bytes (*bytearrays*)

type PyByteArrayObject

Este subtipo de `PyObject` representa un objeto arreglo de bytes de Python.

PyTypeObject PyByteArray_Type

Part of the Stable ABI.
Esta instancia de `PyTypeObject` representa el tipo arreglo de bytes de Python; es el mismo objeto que `bytearray` en la capa de Python.

Macros de verificación de tipos

int PyByteArray_Check(PyObject *o)

Retorna verdadero si el objeto *o* es un objeto de arreglo de bytes o una instancia de un subtipo del tipo arreglo de bytes. Esta función siempre finaliza con éxito.

int PyByteArray_CheckExact(PyObject *o)

Retorna verdadero si el objeto *o* es un objeto de arreglo de bytes, pero no una instancia de un subtipo del tipo arreglo de bytes. Esta función siempre finaliza con éxito.

Funciones API directas

PyObject *PyByteArray_FromObject(PyObject *o)

Return value: New reference. Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Retorna un nuevo objeto de arreglo de bytes de cualquier objeto, *o*, que implementa el buffer protocol.

Al fallar, retorna `NULL` con una excepción asignada.

Nota

If the object implements the buffer protocol, then the buffer must not be mutated while the bytearray object is being created.

PyObject *PyByteArray_FromStringAndSize(const char *string, Py_ssize_t len)

Return value: New reference. Part of the Stable ABI. Thread safety: Atomic.
Crea un nuevo objeto de arreglo de bytes a partir de *string* y su longitud, *len*.

Al fallar, retorna `NULL` con una excepción asignada.

PyObject *PyByteArray_Concat(PyObject *a, PyObject *b)

Return value: New reference. Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Une los arreglos de bytes (*bytearrays*) *a* y *b* y retorna un nuevo arreglo de bytes (*bytearray*) con el resultado.

Al fallar, retorna `NULL` con una excepción asignada.

Nota

If the object implements the buffer protocol, then the buffer must not be mutated while the bytearray object is being created.

Py_ssize_t PyByteArray_Size(PyObject *bytearray)

Part of the Stable ABI. Thread safety: Atomic.
Retorna el tamaño de *bytearray* después de buscar un puntero `NULL`.

char *PyByteArray_AsString(PyObject *bytearray)

Part of the Stable ABI. Thread safety: Safe to call from multiple threads with external synchronization only.
Retorna el contenido de *bytearray* como un arreglo de caracteres después de verificar un puntero `NULL`. El arreglo retornado siempre tiene un byte nulo adicional agregado.

Nota

It is not thread-safe to mutate the bytearray object while using the returned char array.

int PyByteArray_Resize(PyObject *bytearray, Py_ssize_t len)

Part of the Stable ABI.
Resize the internal buffer of *bytearray* to *len*. Failure is a `-1` return with an exception set.

Distinto en la versión 3.14: A negative *len* will now result in an exception being set and -1 returned.

Macros

Estos macros intercambian seguridad por velocidad y no comprueban punteros.

char *PyByteArray_AS_STRING(PyObject *bytearray)

Thread safety: Safe to call from multiple threads with external synchronization only.
Similar a **`PyByteArray_AsString()`**, pero sin comprobación de errores.

Nota

It is not thread-safe to mutate the bytearray object while using the returned char array.

Py_ssize_t PyByteArray_GET_SIZE(PyObject *bytearray)

Thread safety: *Atomic*.

Similar a `PyByteArray_Size()`, pero sin comprobación de errores.

Objetos y códecs unicode

Objetos unicode

Desde la implementación del **PEP 393** [https://peps.python.org/pep-0393/] en Python 3.3, los objetos Unicode utilizan internamente una variedad de representaciones, para permitir el manejo del rango completo de caracteres Unicode mientras se mantiene la eficiencia de memoria. Hay casos especiales para cadenas de caracteres donde todos los puntos de código están por debajo de 128, 256 o 65536; de lo contrario, los puntos de código deben estar por debajo de 1114112 (que es el rango completo de Unicode).

La representación UTF-8 se crea bajo demanda y se almacena en caché en el objeto Unicode.

Nota

La representación `Py_UNICODE` ha sido eliminada desde Python 3.12 con APIs obsoletas. Consulte **PEP 623** [https://peps.python.org/pep-0623/] para obtener más información.

Tipo unicode

Estos son los tipos básicos de objetos Unicode utilizados para la implementación de Unicode en Python:

PyTypeObject PyUnicode_Type

Part of the Stable ABI.
This instance of `PyTypeObject` represents the Python Unicode type. It is exposed to Python code as `str`.

PyTypeObject PyUnicodeIter_Type

Part of the Stable ABI.

This instance of `PyTypeObject` represents the Python Unicode iterator type. It is used to iterate over Unicode string objects.

type Py_UCS4
type Py_UCS2
type Py_UCS1

Part of the Stable ABI.
Estos tipos son definiciones de tipo (*typedefs*) para los tipos "enteros sin signo" (*unsigned int*) lo suficientemente anchos como para contener caracteres de 32 bits, 16 bits y 8 bits, respectivamente. Cuando se trate con caracteres Unicode individuales, use `Py_UCS4`.

Added in version 3.3.

type PyASCIIObject
type PyCompactUnicodeObject
type PyUnicodeObject

Estos subtipos de `PyObject` representan un objeto Python Unicode. En casi todos los casos, no deben usarse directamente, ya que todas las funciones API que se ocupan de objetos Unicode toman y retornan punteros `PyObject`.

Added in version 3.3.

The structure of a particular object can be determined using the following macros. The macros cannot fail; their behavior is undefined if their argument is not a Python Unicode object.

PyUnicode_IS_COMPACT(o)

True if *o* uses the `PyCompactUnicodeObject` structure.

Added in version 3.3.

PyUnicode_IS_COMPACT_ASCII(o)

True if *o* uses the `PyASCIIObject` structure.

Added in version 3.3.

Las siguientes API son realmente macros de C y se pueden utilizar para realizar comprobaciones rápidas y acceder a datos internos de solo lectura de objetos Unicode:

int PyUnicode_Check(PyObject *obj)

Retorna verdadero si el objeto *obj* es un objeto Unicode o una instancia de un subtipo Unicode. Esta función siempre tiene éxito.

int PyUnicode_CheckExact(PyObject *obj)

Retorna verdadero si el objeto *obj* es un objeto Unicode, pero no una instancia de un subtipo. Esta función siempre tiene éxito.

Py_ssize_t PyUnicode_GET_LENGTH(PyObject *unicode)

Retorna la longitud de la cadena de caracteres Unicode, en puntos de código. *unicode* tiene que ser un objeto Unicode en la representación «canónica» (no marcada).

Added in version 3.3.

Py_UCS1 *PyUnicode_1BYTE_DATA(PyObject *unicode)
Py_UCS2 *PyUnicode_2BYTE_DATA(PyObject *unicode)
Py_UCS4 *PyUnicode_4BYTE_DATA(PyObject *unicode)

Retorna un puntero a la representación canónica emitida a los tipos enteros UCS1, UCS2 o UCS4 para el acceso directo a los caracteres. No se realizan verificaciones si la representación canónica tiene el tamaño de carácter correcto; use **`PyUnicode_KIND()`** para seleccionar la función correcta.

Added in version 3.3.

PyUnicode_1BYTE_KIND
PyUnicode_2BYTE_KIND
PyUnicode_4BYTE_KIND

Retorna los valores de la macro **`PyUnicode_KIND()`**.

Added in version 3.3.

Distinto en la versión 3.12: `PyUnicode_WCHAR_KIND` ha sido eliminada.

int PyUnicode_KIND(PyObject *unicode)

Retorna una de las constantes de tipo `PyUnicode` (ver arriba) que indican cuántos bytes por carácter utiliza este objeto Unicode para almacenar sus datos. *unicode* tiene que ser un objeto Unicode en la representación «canónica» (no verificada).

Added in version 3.3.

void *PyUnicode_DATA(PyObject *unicode)

Retorna un puntero vacío al búfer Unicode sin formato. *unicode* tiene que ser un objeto Unicode en la representación «canónica» (no marcada).

Added in version 3.3.

void PyUnicode_WRITE(int kind, void *data, Py_ssize_t index, Py_UCS4 value)

Write the code point *value* to the given zero-based *index* in a string.

The *kind* value and *data* pointer must have been obtained from a string using **`PyUnicode_KIND()`** and **`PyUnicode_DATA()`** respectively. You must hold a reference to that string while calling **`PyUnicode_WRITE()`**. All requirements of **`PyUnicode_WriteChar()`** also apply.

The function performs no checks for any of its requirements, and is intended for usage in loops.

Added in version 3.3.

Py_UCS4 PyUnicode_READ(int kind, void *data, Py_ssize_t index)

Lee un punto de código de una representación canónica *data* (obtenido con **`PyUnicode_DATA()`**). No se realizan verificaciones ni llamadas preparadas.

Added in version 3.3.

Py_UCS4 PyUnicode_READ_CHAR(PyObject *unicode, Py_ssize_t index)

Lee un carácter de un objeto Unicode *unicode*, que debe estar en la representación «canónica». Esto es menos eficiente que **`PyUnicode_READ()`** si realiza varias lecturas consecutivas.

Added in version 3.3.

Py_UCS4 PyUnicode_MAX_CHAR_VALUE(PyObject *unicode)

Retorna el punto de código máximo adecuado para crear otra cadena basada en *unicode*, que debe estar en la representación «canónica». Esto siempre es una aproximación pero más eficiente que iterar sobre la cadena.

Added in version 3.3.

int PyUnicode_IsIdentifier(PyObject *unicode)

Part of the Stable ABI.
Retorna `1` si la cadena de caracteres es un identificador válido de acuerdo con la definición del lenguaje, sección Names (identifiers and keywords). Retorna `0` de lo contrario.

Distinto en la versión 3.9: La función ya no llama a **`Py_FatalError()`** si la cadena de caracteres no está lista.

unsigned int PyUnicode_IS_ASCII(PyObject *unicode)

Return true if the string only contains ASCII characters. Equivalent to **`str.isascii()`**.

Added in version 3.2.

Propiedades de caracteres Unicode

Unicode proporciona muchas propiedades de caracteres diferentes. Los que se necesitan con mayor frecuencia están disponibles a través de estas macros que se asignan a las funciones de C según la configuración de Python.

int Py_UNICODE_ISSPACE(Py_UCS4 ch)

Retorna `1` o `0` dependiendo de si *ch* es un carácter de espacio en blanco.

int Py_UNICODE_ISLOWER(Py_UCS4 ch)

Retorna `1` o `0` dependiendo de si *ch* es un carácter en minúscula.

int Py_UNICODE_ISUPPER(Py_UCS4 ch)

Retorna `1` o `0` dependiendo de si *ch* es un carácter en mayúscula.

int Py_UNICODE_ISTITLE(Py_UCS4 ch)

Retorna `1` o `0` dependiendo de si *ch* es un carácter en caso de título (*titlecase*).

int Py_UNICODE_ISLINEBREAK(Py_UCS4 ch)

Retorna `1` o `0` dependiendo de si *ch* es un carácter de salto de línea.

int Py_UNICODE_ISDECIMAL(Py_UCS4 ch)

Retorna `1` o `0` dependiendo de si *ch* es un carácter decimal o no.

int Py_UNICODE_ISDIGIT(Py_UCS4 ch)

Retorna `1` o `0` dependiendo de si *ch* es un carácter de dígitos.

int Py_UNICODE_ISNUMERIC(Py_UCS4 ch)

Retorna `1` o `0` dependiendo de si *ch* es un carácter numérico.

int Py_UNICODE_ISALPHA(Py_UCS4 ch)

Retorna `1` o `0` dependiendo de si *ch* es un carácter alfabético.

int Py_UNICODE_ISALNUM(Py_UCS4 ch)

Retorna `1` o `0` dependiendo de si *ch* es un carácter alfanumérico.

int Py_UNICODE_ISPRINTABLE(Py_UCS4 ch)

Return `1` or `0` depending on whether *ch* is a printable character, in the sense of **`str.isprintable()`**.

Estas API se pueden usar para conversiones caracteres rápidas y directos:

Py_UCS4 Py_UNICODE_TOLOWER(Py_UCS4 ch)

Retorna el carácter *ch* convertido a minúsculas.

Py_UCS4 Py_UNICODE_TOUPPER(Py_UCS4 ch)

Retorna el carácter *ch* convertido a mayúsculas.

Py_UCS4 Py_UNICODE_TOTITLE(Py_UCS4 ch)

Retorna el carácter *ch* convertido a formato de título (*titlecase*).

int Py_UNICODE_TODECIMAL(Py_UCS4 ch)

Retorna el carácter *ch* convertido a un entero positivo decimal. Retorna `-1` si esto no es posible. Esta función no lanza excepciones.

int Py_UNICODE_TODIGIT(Py_UCS4 ch)

Retorna el carácter *ch* convertido a un entero de un solo dígito. Retorna `-1` si esto no es posible. Esta función no lanza excepciones.

double Py_UNICODE_TONUMERIC(Py_UCS4 ch)

Retorna el carácter *ch* convertido a doble. Retorna `-1.0` si esto no es posible. Esta función no lanza excepciones.

Estas API se pueden usar para trabajar con sustitutos:

int Py_UNICODE_IS_SURROGATE(Py_UCS4 ch)

Comprueba si *ch* es un sustituto (`0xD800 <= ch <= 0xDFFF`).

int Py_UNICODE_IS_HIGH_SURROGATE(Py_UCS4 ch)

Comprueba si *ch* es un sustituto alto (`0xD800 <= ch <= 0xDFFF`).

int Py_UNICODE_IS_LOW_SURROGATE(Py_UCS4 ch)

Comprueba si *ch* es un sustituto bajo (`0xD800 <= ch <= 0xDFFF`).

Py_UCS4 Py_UNICODE_HIGH_SURROGATE(Py_UCS4 ch)

Return the high UTF-16 surrogate (`0xD800` to `0xDBFF`) for a Unicode code point in the range `[0x10000; 0x10FFFF]`.

Py_UCS4 Py_UNICODE_LOW_SURROGATE(Py_UCS4 ch)

Return the low UTF-16 surrogate (`0xDC00` to `0xDFFF`) for a Unicode code point in the range `[0x10000; 0x10FFFF]`.

Py_UCS4 Py_UNICODE_JOIN_SURROGATES(Py_UCS4 high, Py_UCS4 low)

Join two surrogate code points and return a single **`Py_UCS4`** value. *high* and *low* are respectively the leading and trailing surrogates in a surrogate pair. *high* must be in the range `[0xD800; 0xDBFF]` and *low* must be in the range `[0xDC00; 0xDFFF]`.

Creando y accediendo a cadenas de caracteres Unicode

Para crear objetos Unicode y acceder a sus propiedades de secuencia básicas, use estas API:

PyObject *PyUnicode_New(Py_ssize_t size, Py_UCS4 maxchar)

Return value: New reference.
Crea un nuevo objeto Unicode. *maxchar* debe ser el punto de código máximo que se colocará en la cadena de caracteres. Como una aproximación, se puede redondear al valor más cercano en la secuencia 127, 255, 65535, 1114111.

En caso de error, establecer una excepción y devolver `NULL`.

After creation, the string can be filled by **`PyUnicode_WriteChar()`**, **`PyUnicode_CopyCharacters()`**, **`PyUnicode_Fill()`**, **`PyUnicode_WRITE()`** or similar. Since strings are supposed to be immutable, take care to not "use" the result while it is being modified. In particular, before it's filled with its final contents, a string:

- must not be hashed,
- must not be **`converted to UTF-8`**, or another non-«canonical» representation,
- must not have its reference count changed,
- must not be shared with code that might do one of the above.

This list is not exhaustive. Avoiding these uses is your responsibility; Python does not always check these requirements.

To avoid accidentally exposing a partially-written string object, prefer using the **`PyUnicodeWriter`** API, or one of the `PyUnicode_From*` functions below.

Added in version 3.3.

PyObject *PyUnicode_FromKindAndData(int kind, const void *buffer, Py_ssize_t size)

Return value: New reference.

Crea un nuevo objeto Unicode con el tipo *kind* dado (los valores posibles son `PyUnicode_1BYTE_KIND` etc., según lo retornado por `PyUnicode_KIND()`). El *búfer* debe apuntar a un vector (*array*) de *tamaño* unidades de 1, 2 o 4 bytes por carácter, según el tipo.

Si es necesario, la entrada *buffer* se copia y se transforma en la representación canónica. Por ejemplo, si el *buffer* es una cadena de caracteres UCS4 (`PyUnicode_4BYTE_KIND`) y consta solo de puntos de código en el rango UCS1, se transformará en UCS1 (`PyUnicode_1BYTE_KIND`).

Added in version 3.3.

PyObject *PyUnicode_FromStringAndSize(const char *str, Py_ssize_t size)

Return value: New reference. Part of the Stable ABI.
Crea un objeto Unicode desde el búfer de caracteres *str*. Los bytes se interpretarán como codificados en UTF-8. El búfer se copia en el nuevo objeto. El valor de retorno podría ser un objeto compartido, es decir, no se permite la modificación de los datos.

Esta función lanza `SystemError` cuando:

- *size* < 0,
- *str* es `NULL` y *size* > 0

Distinto en la versión 3.12: str == `NULL` con *size* > 0 ya no está permitido.

PyObject *PyUnicode_FromString(const char *str)

Return value: New reference. Part of the Stable ABI.
Crea un objeto Unicode a partir de un búfer *str* de caracteres terminado en nulo y codificado en UTF-8.

PyObject *PyUnicode_FromFormat(const char *format, ...)

Return value: New reference. Part of the Stable ABI.
Toma una cadena de caracteres *format* con el estilo de

`printf()` en C y un número variable de argumentos, calcula el tamaño de la cadena Python Unicode resultante y retorna una cadena de caracteres con los valores formateados. Los argumentos variables deben ser tipos de C y deben corresponder exactamente a los caracteres de formato en la cadena de caracteres *format* codificada en ASCII.

Un especificador de conversión contiene dos o más caracteres y tiene los siguientes componentes, que deben aparecer en este orden:

1. El carácter `'%'`, que marca el inicio del especificador.
2. Indicadores de conversión (opcional), que afectan el resultado de algunos tipos de conversión.
3. Ancho de campo mínimo (opcional). Si se especifica como un `'*'` (asterisco), el ancho real se proporciona en el siguiente argumento, que debe ser de tipo int, y el objeto a convertir viene después del ancho de campo mínimo y la precisión opcional.
4. Precisión (opcional), dada como un `'.'` (punto) seguido de la precisión. Si se especifica como `'*'` (un asterisco), la precisión real se proporciona en el siguiente argumento, que debe ser de tipo int, y el valor a convertir viene después de la precisión.
5. Modificador de longitud (opcional).
6. Tipo de conversión.

Los caracteres de los indicadores de conversión son:

Indicador	**Significado**
0	La conversión se rellenará con ceros para valores numéricos.
-	El valor convertido se ajusta a la izquierda (anula el indicador `0` si ambos están dados).

Los modificadores de longitud para las siguientes conversiones de enteros (`d`, `i`, `o`, `u`, `x` o `X`) especifican el tipo del argumento (int por defecto):

Modificador	**Tipos**
l	long o unsigned long

`ll`	long long o unsigned long long
`j`	**`intmax_t`** o **`uintmax_t`**
`z`	**`size_t`** o **`ssize_t`**
`t`	**`ptrdiff_t`**

El modificador de longitud `l` para las siguientes conversiones `s` o `V` especifica que el tipo del argumento es const wchar_t*.

Los especificadores de conversión son:

Especificador	**Tipo**	**Comentario**
`%`	*n/a*	El carácter literal `%`.
`d`, `i`	Especificado por el modificador de longitud	La representación decimal de un entero C con signo.
`u`	Especificado por el modificador de longitud	La representación decimal de un entero C sin signo.
`o`	Especificado por el modificador de longitud	La representación octal de un entero C sin signo.
`x`	Especificado por el modificador de longitud	La representación hexadecimal de un entero C sin signo (minúsculas).
`X`	Especificado por el modificador de longitud	La representación hexadecimal de un entero C sin signo (mayúsculas).
`c`	int	Un solo carácter.
`s`	const char* o const wchar_t*	Un arreglo de caracteres de C terminada en nulo.
`p`	const void*	La representación hexadecimal de un puntero en C. Principalmente equivalente a `printf("%p")` excepto que se garantiza que comience con el literal `0x`, independientemente de lo que produzca el `printf` de la plataforma.
`A`	PyObject*	El resultado de llamar **`ascii()`**.
`U`	PyObject*	Un objeto unicode.
`V`	PyObject*, const char* o const wchar_t*	Un objeto Unicode (que puede ser `NULL`) y un arreglo de caracteres C terminado en nulo como segundo parámetro (que se utilizará, si el primer parámetro es `NULL`).Un objeto Unicode (que puede ser `NULL`) y un arreglo de caracteres de C terminada en nulo como segundo parámetro (que se utilizará, si el primer parámetro es `NULL`).
`S`	PyObject*	El resultado de llamar **`PyObject_Str()`**.
`R`	PyObject*	El resultado de llamar **`PyObject_Repr()`**.
`T`	PyObject*	Obtener el nombre completamente cualificado de un tipo de objeto; llamar a **`PyType_GetFullyQualifiedName()`**.
`#T`	PyObject*	Similar al formato `T`, pero usa dos puntos (`:`) como

		separador entre el nombre del módulo y el nombre cualificado.
`N`	`PyTypeObject*`	Obtener el nombre completamente cualificado de un tipo; llamar a `PyType_GetFullyQualifiedName()`.
`N#`	`PyTypeObject*`	Similar al formato `N`, pero usa dos puntos (`:`) como separador entre el nombre del módulo y el nombre cualificado.

Nota

La unidad del formateador de ancho es el número de caracteres en lugar de bytes. La unidad del formateador de precisión es la cantidad de bytes o elementos **`wchar_t`** (si se usa el modificador de longitud `l`) para `"%s"` y `"%V"` (si el argumento `PyObject*` es `NULL`), y una cantidad de caracteres para `"%A"`, `"%U"`, `"%S"`, `"%R"` y `"%V"` (si el argumento `PyObject*` no es `NULL`).

Nota

A diferencia de **`printf()`** de C, el indicador `0` tiene efecto incluso cuando se proporciona una precisión para conversiones de enteros (`d`, `i`, `u`, `o`, `x` o `X`).

Distinto en la versión 3.2: Soporte agregado para `"%lld"` y `"%llu"`.

Distinto en la versión 3.3: Soporte agregado para `"%li"`, `"%lli"` y `"%zi"`.

Distinto en la versión 3.4: Soporte agregado para formateadores de anchura y precisión para `"%s"`, `"%A"`, `"%U"`, `"%V"`, `"%S"`, `"%R"`.

Distinto en la versión 3.12: Soporte para los especificadores de conversión `o` y `X`. Soporte para los modificadores de longitud `j` y `t`. Los modificadores de longitud ahora se aplican a todas las conversiones de enteros. El modificador de longitud `l` ahora se aplica a los especificadores de conversión `s` y `V`. Soporte para ancho y precisión variables `*`. Soporte para el indicador `-`.

Un carácter de formato no reconocido ahora establece un **`SystemError`**. En versiones anteriores, hacía que el resto de la cadena de formato se copiara tal cual a la cadena resultante, y cualquier argumento extra se descartaba.

Distinto en la versión 3.13: Soporte agregado para los formatos `%T`, `%#T`, `%N` y `%#N`.

PyObject *PyUnicode_FromFormatV(const char *format, va_list vargs)

Return value: New reference. Part of the Stable ABI.
Idéntico a **`PyUnicode_FromFormat()`** excepto que toma exactamente dos argumentos.

PyObject *PyUnicode_FromObject(PyObject *obj)

Return value: New reference. Part of the Stable ABI.
Copia una instancia de un subtipo Unicode a un nuevo objeto Unicode verdadero si es necesario. Si *obj* ya es un verdadero objeto Unicode (no un subtipo), retorna una nueva referencia fuerte al objeto.

Los objetos que no sean Unicode o sus subtipos causarán un **`TypeError`**.

PyObject *PyUnicode_FromOrdinal(int ordinal)

Return value: New reference. Part of the Stable ABI.
Create a Unicode Object from the given Unicode code point *ordinal*.

The ordinal must be in `range(0x110000)`. A **`ValueError`** is raised in the case it is not.

PyObject *PyUnicode_FromEncodedObject(PyObject *obj, const char *encoding, const char *errors)

Return value: New reference. Part of the Stable ABI.
Decodifica un objeto codificado *obj* en un objeto Unicode.

`bytes`, **`bytearray`** y otros los objetos similares a bytes se decodifican de acuerdo con el *encoding* dado y utilizan el manejo de errores definido por *errors*. Ambos pueden ser `NULL` para que la interfaz use los valores predeterminados (ver Códecs incorporados para más detalles).

Todos los demás objetos, incluidos los objetos Unicode, hacen que se establezca un **`TypeError`**.

La API retorna `NULL` si hubo un error. La entidad que hace la llamadas es la responsable de desreferenciar los objetos retornados.

void PyUnicode_Append(PyObject **p_left, PyObject *right)

Part of the Stable ABI.
Append the string *right* to the end of *p_left*. *p_left* must point to a strong reference to a Unicode object; **`PyUnicode_Append()`** releases («steals») this reference.

On error, set **p_left* to `NULL` and set an exception.

On success, set **p_left* to a new strong reference to the result.

void PyUnicode_AppendAndDel(PyObject **p_left, PyObject *right)

Part of the Stable ABI.
The function is similar to **`PyUnicode_Append()`**, with the only difference being that it decrements the reference count of *right* by one.

PyObject *PyUnicode_BuildEncodingMap(PyObject *string)

Return value: New reference. Part of the Stable ABI.
Return a mapping suitable for decoding a custom single-byte encoding. Given a Unicode string *string* of up to 256 characters representing an encoding table, returns either a compact internal mapping object or a dictionary mapping character ordinals to byte values. Raises a **`TypeError`** and return `NULL` on invalid input.

Added in version 3.2.

const char *PyUnicode_GetDefaultEncoding(void)

Part of the Stable ABI.
Return the name of the default string encoding, `"utf-8"`. See **`sys.getdefaultencoding()`**.

The returned string does not need to be freed, and is valid until interpreter shutdown.

Py_ssize_t PyUnicode_GetLength(PyObject *unicode)

Part of the Stable ABI since version 3.7.
Retorna la longitud del objeto Unicode, en puntos de código.

En caso de error, establece una excepción y retorna `-1`.

Added in version 3.3.

Py_ssize_t PyUnicode_CopyCharacters(PyObject *to, Py_ssize_t to_start, PyObject *from, Py_ssize_t from_start, Py_ssize_t how_many)

Copia caracteres de un objeto Unicode en otro. Esta función realiza la conversión de caracteres cuando es necesario y recurre a **`memcpy()`** si es posible. Retorna `-1` y establece una excepción en caso de error; de lo contrario, retorna el número de caracteres copiados.

The string must not have been “used” yet. See **`PyUnicode_New()`** for details.

Added in version 3.3.

int PyUnicode_Resize(PyObject **unicode, Py_ssize_t length);

Part of the Stable ABI.
Resize a Unicode object **unicode* to the new *length* in code points.

Try to resize the string in place (which is usually faster than allocating a new string and copying characters), or create a new string.

**unicode* is modified to point to the new (resized) object and

`0` is returned on success. Otherwise, `-1` is returned and an exception is set, and **unicode* is left untouched.

The function doesn't check string content, the result may not be a string in canonical representation.

Py_ssize_t PyUnicode_Fill(PyObject *unicode, Py_ssize_t start, Py_ssize_t length, Py_UCS4 fill_char)

Rellena una cadena con un carácter: escriba *fill_char* en `unicode[inicio:inicio+longitud]`.

Falla si *fill_char* es más grande que el carácter máximo de la cadena, o si la cadena tiene más de 1 referencia.

The string must not have been "used" yet. See **`PyUnicode_New()`** for details.

Retorna el número de caracteres escritos, o retorna `-1` y lanza una excepción en caso de error.Retorna el número de caracteres escritos o retorna `-1` y lanza una excepción en caso de error.

Added in version 3.3.

int PyUnicode_WriteChar(PyObject *unicode, Py_ssize_t index, Py_UCS4 character)

Part of the Stable ABI since version 3.7.
Write a *character* to the string *unicode* at the zero-based *index*. Return `0` on success, `-1` on error with an exception set.

This function checks that *unicode* is a Unicode object, that the index is not out of bounds, and that the object's reference count is one. See **`PyUnicode_WRITE()`** for a version that skips these checks, making them your responsibility.

The string must not have been "used" yet. See **`PyUnicode_New()`** for details.

Added in version 3.3.

Py_UCS4 PyUnicode_ReadChar(PyObject *unicode, Py_ssize_t index)

Part of the Stable ABI since version 3.7.
Lee un carácter de una cadena. Esta función verifica que *unicode* es un objeto Unicode y que el índice no está fuera de límites, en contraste con la macro **`PyUnicode_READ_CHAR()`**, que no realiza ninguna comprobación de errores.

Retorna el carácter en caso de éxito, `-1` en caso de error con una excepción establecida.

Added in version 3.3.

PyObject *PyUnicode_Substring(PyObject *unicode, Py_ssize_t start, Py_ssize_t end)

Return value: New reference. Part of the Stable ABI since version 3.7.
Retorna una subcadena de *unicode*, desde el índice de caracteres *start* (incluido) hasta el índice de caracteres *end* (excluido). Los índices negativos no son compatibles. En caso de error, establece una excepción y retorna `NULL`.

Added in version 3.3.

Py_UCS4 *PyUnicode_AsUCS4(PyObject *unicode, Py_UCS4 *buffer, Py_ssize_t buflen, int copy_null)

Part of the Stable ABI since version 3.7.
Copia la cadena *unicode* en un búfer UCS4, incluyendo un carácter nulo, si *copy_null* está configurado. Retorna `NULL` y establece una excepción en caso de error (en particular, un **`SystemError`** si *buflen* es menor que la longitud de *unicode*). Se retorna *buffer* en caso de éxito.

Added in version 3.3.

Py_UCS4 *PyUnicode_AsUCS4Copy(PyObject *unicode)

Part of the Stable ABI since version 3.7.
Copia la cadena *unicode* en un nuevo búfer UCS4 que se

asigna usando **`PyMem_Malloc()`**. Si esto falla, retorna `NULL` con un **`MemoryError`** establecido. El búfer retornado siempre tiene un punto de código nulo adicional agregado.

Added in version 3.3.

Codificación regional

La codificación local actual se puede utilizar para decodificar texto del sistema operativo.

PyObject *PyUnicode_DecodeLocaleAndSize(const char *str, Py_ssize_t length, const char *errors)

Return value: New reference. Part of the Stable ABI since version 3.7.
Decodifica una cadena de caracteres UTF-8 en Android y VxWorks, o de la codificación de configuración regional actual en otras plataformas. Los manejadores de errores admitidos son `"estricto"` y `"subrogateescape"` (**PEP 383** [https://peps.python.org/pep-0383/]). El decodificador usa el controlador de errores `"estricto"` si *errors* es `NULL`. *str* debe terminar con un carácter nulo pero no puede contener caracteres nulos incrustados.

Use **`PyUnicode_DecodeFSDefaultAndSize()`** para decodificar una cadena desde el codificador de sistema de archivos y gestor de errores.

Esta función ignora el modo Python UTF-8.

Ver también

La función **`Py_DecodeLocale()`**.

Added in version 3.3.

Distinto en la versión 3.7: La función ahora también usa la codificación de configuración regional actual para el controlador de errores `subrogateescape`, excepto en Android. Anteriormente, **`Py_DecodeLocale()`** se usaba para el `subrogateescape`, y la codificación local actual se usaba para `estricto`.

PyObject *PyUnicode_DecodeLocale(const char *str, const char *errors)

Return value: New reference. Part of the Stable ABI since version 3.7.
Similar a **`PyUnicode_DecodeLocaleAndSize()`**, pero calcula la longitud de la cadena usando **`strlen()`**.

Added in version 3.3.

PyObject *PyUnicode_EncodeLocale(PyObject *unicode, const char *errors)

Return value: New reference. Part of the Stable ABI since version 3.7.
Codifica un objeto Unicode UTF-8 en Android y VxWorks, o en la codificación local actual en otras plataformas. Los manejadores de errores admitidos son `"estricto"` y `"subrogateescape"` (**PEP 383** [https://peps.python.org/pep-0383/]). El codificador utiliza el controlador de errores `"estricto"` si *errors* es `NULL`. Retorna un objeto **`bytes`**. *unicode* no puede contener caracteres nulos incrustados.

Use **`PyUnicode_EncodeFSDefault()`** para codificar una cadena al codificador de sistema de archivos y gestor de errores.

Esta función ignora el modo Python UTF-8.

Ver también

La función **`Py_EncodeLocale()`**.

Added in version 3.3.

Distinto en la versión 3.7: La función ahora también usa la codificación de configuración regional actual para el controlador de errores `subrogateescape`, excepto en

Android. Anteriormente, **`Py_EncodeLocale()`** se usaba para el `subrogateescape`, y la codificación local actual se usaba para `estricto`.

Codificación del sistema de archivos

Funciones que codifican y decodifican desde el codificador de sistema de archivos y gestor de errores (**PEP 383** [https://peps.python.org/pep-0383/] y **PEP 529** [https://peps.python.org/pep-0529/]).

To encode file names to **`bytes`** during argument parsing, the `"O&"` converter should be used, passing **`PyUnicode_FSConverter()`** as the conversion function:

int PyUnicode_FSConverter(PyObject *obj, void *result)

Part of the Stable ABI.
PyArg_Parse* converter: encode **`str`** objects – obtained directly or through the **`os.PathLike`** interface – to **`bytes`** using **`PyUnicode_EncodeFSDefault()`**; **`bytes`** objects are output as-is. *result* must be an address of a C variable of type PyObject* (or PyBytesObject*). On success, set the variable to a new strong reference to a bytes object which must be released when it is no longer used and return a non-zero value (**`Py_CLEANUP_SUPPORTED`**). Embedded null bytes are not allowed in the result. On failure, return `0` with an exception set.

If *obj* is `NULL`, the function releases a strong reference stored in the variable referred by *result* and returns `1`.

Added in version 3.1.

Distinto en la versión 3.6: Acepta un objeto similar a una ruta (path-like object).

To decode file names to **`str`** during argument parsing, the `"O&"` converter should be used, passing **`PyUnicode_FSDecoder()`** as the conversion function:

int PyUnicode_FSDecoder(PyObject *obj, void *result)

Part of the Stable ABI.
PyArg_Parse* converter: decode `bytes` objects – obtained either directly or indirectly through the `os.PathLike` interface – to `str` using `PyUnicode_DecodeFSDefaultAndSize()`; `str` objects are output as-is. *result* must be an address of a C variable of type PyObject* (or PyUnicodeObject*). On success, set the variable to a new strong reference to a Unicode object which must be released when it is no longer used and return a non-zero value (`Py_CLEANUP_SUPPORTED`). Embedded null characters are not allowed in the result. On failure, return `0` with an exception set.

If *obj* is `NULL`, release the strong reference to the object referred to by *result* and return `1`.

Added in version 3.2.

Distinto en la versión 3.6: Acepta un objeto similar a una ruta (path-like object).

PyObject *PyUnicode_DecodeFSDefaultAndSize(const char *str, Py_ssize_t size)

Return value: New reference. Part of the Stable ABI.
Decodifica una cadena desde el codificador de sistema de archivos y gestor de errores.

Si necesita decodificar una cadena desde la codificación de configuración regional actual, use `PyUnicode_DecodeLocaleAndSize()`.

Ver también

La función `Py_DecodeLocale()`.

Distinto en la versión 3.6: Ahora se usa el manejador de errores del sistema de archivos.

PyObject *PyUnicode_DecodeFSDefault(const char *str)

Return value: New reference. Part of the Stable ABI.
Decodifica una cadena terminada en nulo desde el codificador de sistema de archivos y gestor de errores.

Si se conoce la longitud de la cadena, utilice **`PyUnicode_DecodeFSDefaultAndSize()`**.

Distinto en la versión 3.6: Ahora se usa el manejador de errores del sistema de archivos.

PyObject *PyUnicode_EncodeFSDefault(PyObject *unicode)

Return value: New reference. Part of the Stable ABI.
Codifica un objeto Unicode al codificador de sistema de archivos y gestor de errores, y retorna **`bytes`**. Tenga en cuenta que el objeto resultante **`bytes`** puede contener bytes nulos.

Si necesitas codificar una cadena en la codificación de la configuración regional actual, utiliza **`PyUnicode_EncodeLocale()`**.

Ver también

La función **`Py_EncodeLocale()`**.

Added in version 3.2.

Distinto en la versión 3.6: Ahora se usa el manejador de errores del sistema de archivos.

soporte wchar_t

soporte **`wchar_t`** para plataformas que lo soportan:

PyObject *PyUnicode_FromWideChar(const wchar_t *wstr, Py_ssize_t size)

Return value: New reference. Part of the Stable ABI.
Crea un objeto Unicode a partir del búfer **`wchar_t`** *wstr* del tamaño *size* dado. Pasar `-1` como *size* indica que la función

debe calcular la longitud, usando **`wcslen()`**. Retorna `NULL` en caso de falla.

Py_ssize_t PyUnicode_AsWideChar(PyObject *unicode, wchar_t *wstr, Py_ssize_t size)

Part of the Stable ABI.
Copiar el contenido del objeto Unicode en el búfer **`wchar_t`** *wstr*. Como máximo se copian *size* caracteres **`wchar_t`** (excluyendo un posible carácter de terminación nulo al final). Devolver el número de caracteres **`wchar_t`** copiados o `-1` en caso de error.

Cuando *wstr* es `NULL`, retornar en su lugar el *size* que sería necesario para almacenar todo *unicode*, incluyendo un nulo de terminación.

Tenga en cuenta que la cadena wchar_t* resultante puede o no terminar en nulo. Es responsabilidad del llamador asegurarse de que la cadena wchar_t* termine en nulo en caso de que la aplicación lo requiera. Además, tenga en cuenta que la cadena wchar_t* podría contener caracteres nulos, lo que provocaría que la cadena se truncara cuando se utiliza con la mayoría de las funciones de C.

wchar_t *PyUnicode_AsWideCharString(PyObject *unicode, Py_ssize_t *size)

Part of the Stable ABI since version 3.7.
Convierte el objeto Unicode en una cadena de caracteres ancha. La cadena de salida siempre termina con un carácter nulo. Si *size* no es `NULL`, escribe el número de caracteres anchos (excluyendo el carácter de terminación nulo final) en **size*. Tenga en cuenta que la cadena resultante **`wchar_t`** podría contener caracteres nulos, lo que provocaría que la cadena se truncara cuando se usara con la mayoría de las funciones de C. Si *size* es `NULL` y la cadena wchar_t* contiene caracteres nulos un **`ValueError`** aparece.

Retorna un búfer asignado por **`PyMem_New`** (utilice **`PyMem_Free()`** para liberarlo) en caso de éxito. En caso de error, retorna `NULL` y **size* no está definido. Lanza un **`MemoryError`** si falla la asignación de memoria.

Added in version 3.2.

Distinto en la versión 3.7: Lanza un **`ValueError`** si *size* es `NULL` y la cadena wchar_t* contiene caracteres nulos.

Códecs incorporados

Python proporciona un conjunto de códecs integrados que están escritos en C para mayor velocidad. Todos estos códecs se pueden usar directamente a través de las siguientes funciones.

Muchas de las siguientes API toman dos argumentos de *encoding* y *errors*, y tienen la misma semántica que las del constructor de objetos de cadena incorporado **`str()`**.

Establecer la codificación en `NULL` hace que se use la codificación predeterminada, que es UTF-8. Las llamadas al sistema de archivos deben usar **`PyUnicode_FSConverter()`** para codificar nombres de archivos. Esto utiliza el codificador de sistema de archivos y gestor de errores internamente.

El manejo de errores se establece mediante *errors* que también pueden establecerse en `NULL`, lo que significa usar el manejo predeterminado definido para el códec. El manejo de errores predeterminado para todos los códecs integrados es «estricto» (se lanza **`ValueError`**).

Todos los códecs usan una interfaz similar. Solo las desviaciones de los siguientes genéricos se documentan por simplicidad.

Códecs genéricos

The following macro is provided:

Py_UNICODE_REPLACEMENT_CHARACTER

The Unicode code point `U+FFFD` (replacement character).

This Unicode character is used as the replacement character during decoding if the *errors* argument is set to «replace».

Estas son las APIs de códecs genéricos:

PyObject *PyUnicode_Decode(const char *str, Py_ssize_t size, const char *encoding, const char *errors)

Return value: New reference. Part of the Stable ABI.
Crea un objeto Unicode decodificando *size* bytes de la cadena codificada *str*. *encoding* y *errors* tienen el mismo significado que los parámetros del mismo nombre en la función incorporada **`str()`**. El códec que se utilizará se busca utilizando el registro de códec Python. Retorna `NULL` si el códec provocó una excepción.

PyObject *PyUnicode_AsEncodedString(PyObject *unicode, const char *encoding, const char *errors)

Return value: New reference. Part of the Stable ABI.
Codifica un objeto Unicode y retorna el resultado como un objeto de bytes de Python. *encoding* y *errors* tienen el mismo significado que los parámetros del mismo nombre en el método Unicode **`encode()`**. El códec que se utilizará se busca utilizando el registro de códec Python. Retorna `NULL` si el códec provocó una excepción.

Códecs UTF-8

Estas son las APIs del códec UTF-8:

PyObject *PyUnicode_DecodeUTF8(const char *str, Py_ssize_t size, const char *errors)

Return value: New reference. Part of the Stable ABI.
Crea un objeto Unicode decodificando *size* bytes de la cadena codificada UTF-8 *str*. Retorna `NULL` si el códec provocó una excepción.

PyObject *PyUnicode_DecodeUTF8Stateful(const char *str, Py_ssize_t size, const char *errors, Py_ssize_t *consumed)

Return value: New reference. Part of the Stable ABI.
Si *consumed* es `NULL`, se comporta como **`PyUnicode_DecodeUTF8()`**. Si *consumed* no es `NULL`, las secuencias de bytes UTF-8 incompletas no se tratarán como un error. Esos bytes no serán decodificados y la cantidad de bytes que han sido decodificados se almacenará en *consumed*.

PyObject *PyUnicode_AsUTF8String(PyObject *unicode)

Return value: New reference. Part of the Stable ABI.
Codifica un objeto Unicode usando UTF-8 y retorna el resultado como un objeto de bytes de Python. El manejo de errores es «estricto». Retorna `NULL` si el códec provocó una excepción.

La función falla si la cadena contiene puntos de código sustitutos (`U+D800 - U+DFFF`).

const char *PyUnicode_AsUTF8AndSize(PyObject *unicode, Py_ssize_t *size)

Part of the Stable ABI since version 3.10.
Retorna un puntero a la codificación UTF-8 del objeto Unicode y almacena el tamaño de la representación codificada (en bytes) en *size*. El argumento *size* puede ser `NULL`; en este caso no se almacenará el tamaño. El búfer retornado siempre tiene un byte nulo adicional agregado (no incluido en *size*), independientemente de si hay otros puntos de código nulo.

En caso de error, establecer una excepción, establecer *size* en `-1` (si no es NULL) y devolver `NULL`.

La función falla si la cadena contiene puntos de código sustitutos (`U+D800 - U+DFFF`).

Esto almacena en caché la representación UTF-8 de la cadena en el objeto Unicode, y las llamadas posteriores retornarán un puntero al mismo búfer. El llamador no es responsable de desasignar el búfer. El búfer se desasigna y los punteros al mismo se invalidan cuando el objeto Unicode es recolectado por el recolector de basura.

Added in version 3.3.

Distinto en la versión 3.7: El tipo de retorno ahora es `const char *` en lugar de `char *`.

Distinto en la versión 3.10: Esta función es parte de la API limitada.

const char *PyUnicode_AsUTF8(PyObject *unicode)

Como **`PyUnicode_AsUTF8AndSize()`**, pero no almacena el tamaño.

Advertencia

This function does not have any special behavior for null characters [https://en.wikipedia.org/wiki/Null_character] embedded within *unicode*. As a result, strings containing null characters will remain in the returned string, which some C functions might interpret as the end of the string, leading to truncation. If truncation is an issue, it is recommended to use **`PyUnicode_AsUTF8AndSize()`** instead.

Added in version 3.3.

Distinto en la versión 3.7: El tipo de retorno ahora es `const char *` en lugar de `char *`.

Códecs UTF-32

Estas son las APIs de códecs para UTF-32:

PyObject *PyUnicode_DecodeUTF32(const char *str, Py_ssize_t size, const char *errors, int *byteorder)

Return value: New reference. Part of the Stable ABI.
Decodifica *size* bytes de una cadena de búfer codificada

UTF-32 y retorna el objeto Unicode correspondiente. *errors* (si no es `NULL`) define el manejo de errores. Su valor predeterminado es «estricto».

Si *byteorder* no es `NULL`, el decodificador comienza a decodificar utilizando el orden de bytes dado:

```
*byteorder == -1: little endian
*byteorder == 0:  native order
*byteorder == 1:  big endian
```

Si `*byteorder` es cero, y los primeros cuatro bytes de los datos de entrada son una marca de orden de bytes (BOM), el decodificador cambia a este orden de bytes y la BOM no se copia en la cadena de caracteres Unicode resultante. Si `*byteorder` es `-1` o `1`, cualquier marca de orden de bytes se copia en la salida.

Una vez completado, **byteorder* se establece en el orden de bytes actual al final de los datos de entrada.

Si *byteorder* es `NULL`, el códec se inicia en modo de orden nativo.

Retorna `NULL` si el códec provocó una excepción.

PyObject *PyUnicode_DecodeUTF32Stateful(const char *str, Py_ssize_t size, const char *errors, int *byteorder, Py_ssize_t *consumed)

Return value: New reference. Part of the Stable ABI.
Si *consumed* es `NULL`, se comporta como **`PyUnicode_DecodeUTF32()`**. Si *consumed* no es `NULL`, **`PyUnicode_DecodeUTF32Stateful()`** no tratará las secuencias de bytes UTF-32 incompletas finales (como un número de bytes no divisible por cuatro) como un error. Esos bytes no serán decodificados y la cantidad de bytes que han sido decodificados se almacenará en *consumed*.

PyObject *PyUnicode_AsUTF32String(PyObject *unicode)

Return value: New reference. Part of the Stable ABI.

Retorna una cadena de bytes de Python usando la codificación UTF-32 en orden de bytes nativo. La cadena siempre comienza con una marca BOM. El manejo de errores es «estricto». Retorna `NULL` si el códec provocó una excepción.

Códecs UTF-16

Estas son las APIs de códecs para UTF-16:

PyObject *PyUnicode_DecodeUTF16(const char *str, Py_ssize_t size, const char *errors, int *byteorder)

Return value: New reference. Part of the Stable ABI.
Decodifica *size* bytes de una cadena de caracteres de búfer codificada UTF-16 y retorna el objeto Unicode correspondiente. *errors* (si no es `NULL`) define el manejo de errores. Su valor predeterminado es «estricto».

Si *byteorder* no es `NULL`, el decodificador comienza a decodificar utilizando el orden de bytes dado:

```
*byteorder == -1: little endian
*byteorder == 0:  native order
*byteorder == 1:  big endian
```

Si `*byteorder` es cero, y los primeros dos bytes de los datos de entrada son una marca de orden de bytes (BOM), el decodificador cambia a este orden de bytes y la BOM no se copia en la cadena de caracteres Unicode resultante. Si `*byteorder` es `-1` o `1`, cualquier marca de orden de bytes se copia en la salida (donde dará como resultado un `\ufeff` o un carácter `\ufffe`).

Una vez completado, **byteorder* se establece en el orden de bytes actual al final de los datos de entrada.

Si *byteorder* es `NULL`, el códec se inicia en modo de orden nativo.

Retorna `NULL` si el códec provocó una excepción.

PyObject *PyUnicode_DecodeUTF16Stateful(const char *str, Py_ssize_t size, const char *errors, int *byteorder, Py_ssize_t *consumed)

Return value: New reference. Part of the Stable ABI.
Si *consumed* es `NULL`, se comporta como **`PyUnicode_DecodeUTF16()`**. Si *consumed* no es `NULL`, **`PyUnicode_DecodeUTF16Stateful()`** no tratará las secuencias de bytes UTF-16 incompletas finales (como un número impar de bytes o un par sustituto dividido) como un error. Esos bytes no serán decodificados y la cantidad de bytes que han sido decodificados se almacenará en *consumed*.

PyObject *PyUnicode_AsUTF16String(PyObject *unicode)

Return value: New reference. Part of the Stable ABI.
Retorna una cadena de bytes de Python usando la codificación UTF-16 en orden de bytes nativo. La cadena siempre comienza con una marca BOM. El manejo de errores es «estricto». Retorna `NULL` si el códec provocó una excepción.

Códecs UTF-7

Estas son las APIs del códec UTF-7:

PyObject *PyUnicode_DecodeUTF7(const char *str, Py_ssize_t size, const char *errors)

Return value: New reference. Part of the Stable ABI.
Crea un objeto Unicode decodificando *size* bytes de la cadena de caracteres codificada UTF-7 *str*. Retorna `NULL` si el códec provocó una excepción.

PyObject *PyUnicode_DecodeUTF7Stateful(const char *str, Py_ssize_t size, const char *errors, Py_ssize_t *consumed)

Return value: New reference. Part of the Stable ABI.
Si *consumed* es `NULL`, se comporta como **`PyUnicode_DecodeUTF7()`**. Si *consumed* no es `NULL`, las secciones UTF-7 base-64 incompletas no se tratarán como un

error. Esos bytes no serán decodificados y la cantidad de bytes que han sido decodificados se almacenará en *consumed*.

Códecs Unicode escapado

Estas son las APIs de códecs para Unicode escapado:

PyObject *PyUnicode_DecodeUnicodeEscape(const char *str, Py_ssize_t size, const char *errors)

Return value: New reference. Part of the Stable ABI.
Crea un objeto Unicode decodificando *size* bytes de la cadena codificada Unicode escapada (*Unicode-Escape*) *str*. Retorna `NULL` si el códec provocó una excepción.

PyObject *PyUnicode_AsUnicodeEscapeString(PyObject *unicode)

Return value: New reference. Part of the Stable ABI.
Codifica un objeto Unicode usando Unicode escapado (*Unicode-Escape*) y retorna el resultado como un objeto de bytes. El manejo de errores es «estricto». Retorna `NULL` si el códec provocó una excepción.

Códecs para Unicode escapado en bruto

Estas son las API del códec Unicode escapado en bruto (*Raw Unicode Escape*):

PyObject *PyUnicode_DecodeRawUnicodeEscape(const char *str, Py_ssize_t size, const char *errors)

Return value: New reference. Part of the Stable ABI.
Crea un objeto Unicode decodificando *size* bytes de la cadena de caracteres codificada Unicode escapada en bruto (*Raw-Unicode-Escape*) *str*. Retorna `NULL` si el códec provocó una excepción.

PyObject *PyUnicode_AsRawUnicodeEscapeString(PyObject *unicode)

Return value: New reference. Part of the Stable ABI.

Codifica un objeto Unicode usando Unicode escapado en bruto (*Raw-Unicode-Escape*) y retorna el resultado como un objeto de bytes. El manejo de errores es «estricto». Retorna `NULL` si el códec provocó una excepción.

Códecs Latin-1

Estas son las API del códec Latin-1: Latin-1 corresponde a los primeros 256 ordinales Unicode y solo estos son aceptados por los códecs durante la codificación.

PyObject *PyUnicode_DecodeLatin1(const char *str, Py_ssize_t size, const char *errors)

Return value: New reference. Part of the Stable ABI.
Crea un objeto Unicode decodificando *size* bytes de la cadena de caracteres codificada en latin-1 *str*. Retorna `NULL` si el códec provocó una excepción.

PyObject *PyUnicode_AsLatin1String(PyObject *unicode)

Return value: New reference. Part of the Stable ABI.
Codifica un objeto Unicode usando Latin-1 y retorna el resultado como un objeto de bytes Python. El manejo de errores es «estricto». Retorna `NULL` si el códec provocó una excepción.

Códecs ASCII

Estas son las API del códec ASCII. Solo se aceptan datos ASCII de 7 bits. Todos los demás códigos generan errores.

PyObject *PyUnicode_DecodeASCII(const char *str, Py_ssize_t size, const char *errors)

Return value: New reference. Part of the Stable ABI.
Crea un objeto Unicode decodificando *size* bytes de la cadena de caracteres codificada ASCII *str*. Retorna `NULL` si el códec provocó una excepción.

PyObject *PyUnicode_AsASCIIString(PyObject *unicode)

Return value: New reference. Part of the Stable ABI.
Codifica un objeto Unicode usando ASCII y retorna el resultado como un objeto de bytes de Python. El manejo de errores es «estricto». Retorna `NULL` si el códec provocó una excepción.

Códecs de mapa de caracteres

Este códec es especial en el sentido de que puede usarse para implementar muchos códecs diferentes (y de hecho así se hizo para obtener la mayoría de los códecs estándar incluidos en el paquete **`encodings`**). El códec utiliza mapeos para codificar y decodificar caracteres. Los objetos de mapeo proporcionados deben admitir la interfaz de mapeo **`__getitem__()`**; los diccionarios y las secuencias funcionan bien.

Estos son las API de códec de mapeo:

PyObject *PyUnicode_DecodeCharmap(const char *str, Py_ssize_t length, PyObject *mapping, const char *errors)

Return value: New reference. Part of the Stable ABI.
Crea un objeto Unicode decodificando *size* bytes de la cadena de caracteres codificada *str* usando el objeto *mapping* dado. Retorna `NULL` si el códec provocó una excepción.

Si *mapping* es `NULL`, se aplicará la decodificación Latin-1. De lo contrario, *mapping* debe asignar bytes ordinales (enteros en el rango de 0 a 255) a cadenas de caracteres Unicode, enteros (que luego se interpretan como ordinales Unicode) o `None`. Los bytes de datos sin asignar - los que causan un **`LookupError`**, así como los que se asignan a `None`, `0xFFFE` o `'\ uffe'`, se tratan como asignaciones indefinidas y causan un error.

PyObject *PyUnicode_AsCharmapString(PyObject *unicode, PyObject *mapping)

Return value: New reference. Part of the Stable ABI.
Codifica un objeto Unicode usando el objeto *mapping* dado y retorna el resultado como un objeto de bytes. El manejo de

errores es «estricto». Retorna `NULL` si el códec provocó una excepción.

El objeto *mapping* debe asignar enteros ordinales Unicode a objetos de bytes, enteros en el rango de 0 a 255 o `None`. Los ordinales de caracteres no asignados (los que causan un **`LookupError`**), así como los asignados a `Ninguno`, se tratan como «mapeo indefinido» y causan un error.

La siguiente API de códec es especial en que asigna Unicode a Unicode.

PyObject *PyUnicode_Translate(PyObject *unicode, PyObject *table, const char *errors)

Return value: New reference. Part of the Stable ABI.
Traduce una cadena de caracteres aplicando una tabla de mapeo y retornando el objeto Unicode resultante. Retorna `NULL` cuando el códec provocó una excepción.

La tabla de mapeo debe mapear enteros ordinales Unicode a enteros ordinales Unicode o `None` (causando la eliminación del carácter).

Las tablas de mapeo solo necesitan proporcionar la interfaz **`__getitem__()`**; los diccionarios y las secuencias funcionan bien. Los ordinales de caracteres no asignados (los que causan un **`LookupError`**) se dejan intactos y se copian tal cual.

errors tiene el significado habitual para los códecs. Puede ser `NULL`, lo que indica que debe usar el manejo de errores predeterminado.

Códecs MBCS para Windows

Estas son las API de códec MBCS. Actualmente solo están disponibles en Windows y utilizan los convertidores Win32 MBCS para implementar las conversiones. Tenga en cuenta que MBCS (o DBCS) es una clase de codificaciones, no solo una. La codificación de destino está definida por la configuración del usuario en la máquina que ejecuta el códec.

PyObject *PyUnicode_DecodeMBCS(const char *str, Py_ssize_t size, const char *errors)

Return value: New reference. Part of the Stable ABI on Windows since version 3.7.

Crea un objeto Unicode decodificando *size* bytes de la cadena de caracteres codificada con MBCS *str*. Retorna `NULL` si el códec provocó una excepción.

PyObject *PyUnicode_DecodeMBCSStateful(const char *str, Py_ssize_t size, const char *errors, Py_ssize_t *consumed)

Return value: New reference. Part of the Stable ABI on Windows since version 3.7.

Si *consumed* es `NULL`, se comporta como **`PyUnicode_DecodeMBCS()`**. Si *consumed* no es `NULL`, **`PyUnicode_DecodeMBCSStateful()`** no decodificará el byte inicial y el número de bytes que se han decodificado se almacenará en *consumed*.

PyObject *PyUnicode_DecodeCodePageStateful(int code_page, const char *str, Py_ssize_t size, const char *errors, Py_ssize_t *consumed)

Return value: New reference. Part of the Stable ABI on Windows since version 3.7.

Similar to **`PyUnicode_DecodeMBCSStateful()`**, except uses the code page specified by *code_page*.

PyObject *PyUnicode_AsMBCSString(PyObject *unicode)

Return value: New reference. Part of the Stable ABI on Windows since version 3.7.

Codifica un objeto Unicode usando MBCS y retorna el resultado como un objeto de bytes de Python. El manejo de errores es «estricto». Retorna `NULL` si el códec provocó una excepción.

PyObject *PyUnicode_EncodeCodePage(int code_page, PyObject *unicode, const char *errors)

Return value: New reference. Part of the Stable ABI on Windows

since version 3.7.
Codifica el objeto Unicode utilizando la página de códigos especificada y retorna un objeto de bytes de Python. Retorna `NULL` si el códec provocó una excepción. Use la página de códigos **`CP_ACP`** para obtener el codificador MBCS.

Added in version 3.3.

Métodos y funciones de ranura (*Slot*)

Las siguientes API son capaces de manejar objetos Unicode y cadenas de caracteres en la entrada (nos referimos a ellos como cadenas de caracteres en las descripciones) y retorna objetos Unicode o enteros según corresponda.

Todos retornan `NULL` o `-1` si ocurre una excepción.

PyObject *PyUnicode_Concat(PyObject *left, PyObject *right)

Return value: New reference. Part of the Stable ABI.
Une dos cadenas de caracteres que dan una nueva cadena de caracteres Unicode.

PyObject *PyUnicode_Split(PyObject *unicode, PyObject *sep, Py_ssize_t maxsplit)

Return value: New reference. Part of the Stable ABI.
Divide una cadena de caracteres dando una lista de cadenas de caracteres Unicode. Si *sep* es `NULL`, la división se realizará en todas las subcadenas de espacios en blanco. De lo contrario, las divisiones ocurren en el separador dado. A lo sumo se realizarán *maxsplit* divisiones. Si es negativo, no se establece ningún límite. Los separadores no están incluidos en la lista resultante.

On error, return `NULL` with an exception set.

Equivalent to **`str.split()`**.

PyObject *PyUnicode_RSplit(PyObject *unicode, PyObject *sep,

Py_ssize_t maxsplit)

Return value: New reference. Part of the Stable ABI.
Similar to **`PyUnicode_Split()`**, but splitting will be done beginning at the end of the string.

On error, return `NULL` with an exception set.

Equivalent to **`str.rsplit()`**.

PyObject *PyUnicode_Splitlines(PyObject *unicode, int keepends)

Return value: New reference. Part of the Stable ABI.
Divide una cadena de caracteres Unicode en los saltos de línea, retornando una lista de cadenas de caracteres Unicode. CRLF se considera un salto de línea. Si *keepends* es `0`, los caracteres de salto de línea no se incluyen en las cadenas de caracteres resultantes.

PyObject *PyUnicode_Partition(PyObject *unicode, PyObject *sep)

Return value: New reference. Part of the Stable ABI.
Split a Unicode string at the first occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing the string itself, followed by two empty strings.

sep must not be empty.

On error, return `NULL` with an exception set.

Equivalent to **`str.partition()`**.

PyObject *PyUnicode_RPartition(PyObject *unicode, PyObject *sep)

Return value: New reference. Part of the Stable ABI.
Similar to **`PyUnicode_Partition()`**, but split a Unicode string at the last occurrence of *sep*. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself.

sep must not be empty.

On error, return `NULL` with an exception set.

Equivalent to **`str.rpartition()`**.

PyObject *PyUnicode_Join(PyObject *separator, PyObject *seq)

Return value: New reference. Part of the Stable ABI.
Une una secuencia de cadenas de caracteres usando el *separator* dado y retorna la cadena de caracteres Unicode resultante.

Py_ssize_t PyUnicode_Tailmatch(PyObject *unicode, PyObject *substr, Py_ssize_t start, Py_ssize_t end, int direction)

Part of the Stable ABI.
Retorna `1` si *substr* coincide con `unicode[start:end]` en el final de cola dado (*direction* == `-1` significa hacer una coincidencia de prefijo, *direction* == `1` una coincidencia de sufijo), `0` de lo contrario. Retorna `-1` si ocurrió un error.

Py_ssize_t PyUnicode_Find(PyObject *unicode, PyObject *substr, Py_ssize_t start, Py_ssize_t end, int direction)

Part of the Stable ABI.
Retorna la primera posición de *substr* en `unicode[start:end]` usando la *direction* dada (*direction* == `1` significa hacer una búsqueda hacia adelante, *direction* == `-1` una búsqueda hacia atrás). El valor de retorno es el índice de la primera coincidencia; un valor de `-1` indica que no se encontró ninguna coincidencia, y `-2` indica que se produjo un error y se ha establecido una excepción.

Py_ssize_t PyUnicode_FindChar(PyObject *unicode, Py_UCS4 ch, Py_ssize_t start, Py_ssize_t end, int direction)

Part of the Stable ABI since version 3.7.
Retorna la primera posición del carácter *ch* en `unicode[start:end]` usando la *direction* dada (*direction* == `1` significa hacer una búsqueda hacia adelante, *direction*

`==` `-1` una búsqueda hacia atrás). El valor de retorno es el índice de la primera coincidencia; un valor de `-1` indica que no se encontró ninguna coincidencia, y `-2` indica que se produjo un error y se ha establecido una excepción.

Added in version 3.3.

Distinto en la versión 3.7: start y *end* ahora están ajustados para comportarse como `unicode[start:end]`.

Py_ssize_t PyUnicode_Count(PyObject *unicode, PyObject *substr, Py_ssize_t start, Py_ssize_t end)

Part of the Stable ABI.
Retorna el número de ocurrencias no superpuestas de *substr* en `unicode[start:end]`. Retorna `-1` si ocurrió un error.

PyObject *PyUnicode_Replace(PyObject *unicode, PyObject *substr, PyObject *replstr, Py_ssize_t maxcount)

Return value: New reference. Part of the Stable ABI.
Reemplaza como máximo *maxcount* ocurrencias de *substr* en *unicode* con *replstr* y retorna el objeto Unicode resultante. *maxcount* `==` `-1` significa reemplazar todas las ocurrencias.

int PyUnicode_Compare(PyObject *left, PyObject *right)

Part of the Stable ABI.
Compara dos cadenas de caracteres y retorna `-1`, `0`, `1` para menor que, igual y mayor que, respectivamente.

Esta función retorna `-1` en caso de falla, por lo que se debe llamar a **`PyErr_Occurred()`** para verificar si hay errores.

Ver también

The **`PyUnicode_Equal()`** function.

int PyUnicode_Equal(PyObject *a, PyObject *b)

Part of the Stable ABI since version 3.14.

Test if two strings are equal:

- Return `1` if *a* is equal to *b*.
- Return `0` if *a* is not equal to *b*.
- Set a **`TypeError`** exception and return `-1` if *a* or *b* is not a **`str`** object.

The function always succeeds if *a* and *b* are **`str`** objects.

The function works for **`str`** subclasses, but does not honor custom `__eq__()` method.

Ver también

The **`PyUnicode_Compare()`** function.

Added in version 3.14.

int PyUnicode_EqualToUTF8AndSize(PyObject *unicode, const char *string, Py_ssize_t size)

Part of the Stable ABI since version 3.13.
Compara un objeto Unicode con un búfer de caracteres que se interpreta como codificado en UTF-8 o ASCII y retorna verdadero (`1`) si son iguales, o falso (`0`) en caso contrario. Si el objeto Unicode contiene puntos de código subrogados (`U+D800` - `U+DFFF`) o la cadena C no es UTF-8 válida, se retorna falso (`0`).

Esta función no lanza excepciones.

Added in version 3.13.

int PyUnicode_EqualToUTF8(PyObject *unicode, const char *string)

Part of the Stable ABI since version 3.13.
Similar a **`PyUnicode_EqualToUTF8AndSize()`**, pero calcula la longitud de *string* usando **`strlen()`**. Si el objeto Unicode contiene caracteres nulos, se retorna falso (`0`).

Added in version 3.13.

int PyUnicode_CompareWithASCIIString(PyObject *unicode, const char *string)

Part of the Stable ABI.
Compara un objeto Unicode, *unicode*, con *string* y retorna `-1`, `0`, `1` para menor que, igual y mayor que, respectivamente. Es mejor pasar solo cadenas de caracteres codificadas en ASCII, pero la función interpreta la cadena de entrada como ISO-8859-1 si contiene caracteres no ASCII.

Esta función no lanza excepciones.

PyObject *PyUnicode_RichCompare(PyObject *left, PyObject *right, int op)

Return value: New reference. Part of the Stable ABI.
Comparación enriquecida de dos cadenas de caracteres Unicode y retorna uno de los siguientes:

- `NULL` en caso de que se produzca una excepción
- **`Py_True`** o **`Py_False`** para comparaciones exitosas
- **`Py_NotImplemented`** en caso que se desconozca la combinación de tipos

Los posibles valores para *op* son **`Py_GT`**, **`Py_GE`**, **`Py_EQ`**, **`Py_NE`**, **`Py_LT`**, y **`Py_LE`**.

PyObject *PyUnicode_Format(PyObject *format, PyObject *args)

Return value: New reference. Part of the Stable ABI.
Retorna un nuevo objeto de cadena de caracteres desde *format* y *args*; esto es análogo al `format % args`.

int PyUnicode_Contains(PyObject *unicode, PyObject *substr)

Part of the Stable ABI.
Comprueba si *substr* está contenido en *unicode* y retorna verdadero o falso en consecuencia.

substr tiene que convertirse a una cadena Unicode de un solo elemento. Se retorna `-1` si hubo un error.

void PyUnicode_InternInPlace(PyObject **p_unicode)

Part of the Stable ABI.

Interna el argumento *p_unicode en su lugar. El argumento debe ser la dirección de una variable de puntero que apunta a un objeto de cadena Unicode de Python. Si existe una cadena internada que es igual a *p_unicode, la establece en *p_unicode (liberando la referencia al objeto de cadena anterior y creando una nueva referencia fuerte al objeto de cadena internada), de lo contrario, deja *p_unicode tal cual y lo interna.

(Aclaración: aunque se habla mucho de referencias, piense en esta función como de referencia neutral. Debe poseer el objeto que pasa; después de la llamada, ya no posee la referencia pasada, pero ahora posee el resultado).

Esta función nunca lanza una excepción. En caso de error, deja su argumento inalterado sin internarlo.

Las instancias de subclases de `str` pueden no ser internadas, es decir, PyUnicode_CheckExact(*p_unicode) debe ser verdadero. Si no lo es, entonces, como con cualquier otro error, el argumento se deja sin cambios.

Tenga en cuenta que las cadenas internadas no son "inmortales". Debe mantener una referencia al resultado para beneficiarse de la internación.

PyObject *PyUnicode_InternFromString(const char *str)

Return value: New reference. Part of the Stable ABI.

Una combinación de `PyUnicode_FromString()` y `PyUnicode_InternInPlace()`, destinada a cadenas asignadas estáticamente.

Retorna una nueva referencia («owned») a un nuevo objeto de cadena Unicode que ha sido internado, o a un objeto de cadena previamente internado con el mismo valor.

Python puede conservar una referencia al resultado o hacerlo immortal, evitando que sea recolectado por el recolector de basura de forma inmediata. Para internar un número no acotado de cadenas diferentes, como las que provienen de la entrada del usuario, se recomienda llamar directamente a `PyUnicode_FromString()` y `PyUnicode_InternInPlace()`.

unsigned int PyUnicode_CHECK_INTERNED(PyObject *str)

Return a non-zero value if *str* is interned, zero if not. The *str* argument must be a string; this is not checked. This function always succeeds.

Detalles de implementación de CPython: A non-zero return value may carry additional information about *how* the string is interned. The meaning of such non-zero values, as well as each specific string's intern-related details, may change between CPython versions.

PyUnicodeWriter

The `PyUnicodeWriter` API can be used to create a Python `str` object.

Added in version 3.14.

type PyUnicodeWriter

A Unicode writer instance.

The instance must be destroyed by `PyUnicodeWriter_Finish()` on success, or `PyUnicodeWriter_Discard()` on error.

PyUnicodeWriter *PyUnicodeWriter_Create(Py_ssize_t length)

Create a Unicode writer instance.

length must be greater than or equal to `0`.

If *length* is greater than `0`, preallocate an internal buffer of *length* characters.

Set an exception and return `NULL` on error.

PyObject *PyUnicodeWriter_Finish(PyUnicodeWriter *writer)

Return the final Python **`str`** object and destroy the writer instance.

Set an exception and return `NULL` on error.

The writer instance is invalid after this call.

void PyUnicodeWriter_Discard(PyUnicodeWriter *writer)

Discard the internal Unicode buffer and destroy the writer instance.

If *writer* is `NULL`, no operation is performed.

The writer instance is invalid after this call.

int PyUnicodeWriter_WriteChar(PyUnicodeWriter *writer, Py_UCS4 ch)

Write the single Unicode character *ch* into *writer*.

On success, return `0`. On error, set an exception, leave the writer unchanged, and return `-1`.

int PyUnicodeWriter_WriteUTF8(PyUnicodeWriter *writer, const char *str, Py_ssize_t size)

Decode the string *str* from UTF-8 in strict mode and write the output into *writer*.

size is the string length in bytes. If *size* is equal to `-1`, call `strlen(str)` to get the string length.

On success, return `0`. On error, set an exception, leave the writer unchanged, and return `-1`.

See also **`PyUnicodeWriter_DecodeUTF8Stateful()`**.

int PyUnicodeWriter_WriteASCII(PyUnicodeWriter *writer, const char *str, Py_ssize_t size)

Write the ASCII string *str* into *writer*.

size is the string length in bytes. If *size* is equal to `-1`, call `strlen(str)` to get the string length.

str must only contain ASCII characters. The behavior is undefined if *str* contains non-ASCII characters.

On success, return `0`. On error, set an exception, leave the writer unchanged, and return `-1`.

int PyUnicodeWriter_WriteWideChar(PyUnicodeWriter *writer, const wchar_t *str, Py_ssize_t size)

Write the wide string *str* into *writer*.

size is a number of wide characters. If *size* is equal to `-1`, call `wcslen(str)` to get the string length.

On success, return `0`. On error, set an exception, leave the writer unchanged, and return `-1`.

int PyUnicodeWriter_WriteUCS4(PyUnicodeWriter *writer, Py_UCS4 *str, Py_ssize_t size)

Writer the UCS4 string *str* into *writer*.

size is a number of UCS4 characters.

On success, return `0`. On error, set an exception, leave the writer unchanged, and return `-1`.

int PyUnicodeWriter_WriteStr(PyUnicodeWriter *writer, PyObject *obj)

Call **`PyObject_Str()`** on *obj* and write the output into *writer*.

On success, return `0`. On error, set an exception, leave the writer unchanged, and return `-1`.

To write a **`str`** subclass which overrides the **`__str__()`** method, **`PyUnicode_FromObject()`** can be used to get the original string.

int PyUnicodeWriter_WriteRepr(PyUnicodeWriter *writer, PyObject *obj)

Call **`PyObject_Repr()`** on *obj* and write the output into *writer*.

If *obj* is `NULL`, write the string `"<NULL>"` into *writer*.

On success, return `0`. On error, set an exception, leave the writer unchanged, and return `-1`.

Distinto en la versión 3.14.4: Added support for `NULL`.

int PyUnicodeWriter_WriteSubstring(PyUnicodeWriter *writer, PyObject *str, Py_ssize_t start, Py_ssize_t end)

Write the substring `str[start:end]` into *writer*.

str must be Python **`str`** object. *start* must be greater than or equal to 0, and less than or equal to *end*. *end* must be less than or equal to *str* length.

On success, return `0`. On error, set an exception, leave the writer unchanged, and return `-1`.

int PyUnicodeWriter_Format(PyUnicodeWriter *writer, const char *format, ...)

Similar to **`PyUnicode_FromFormat()`**, but write the output directly into *writer*.

On success, return `0`. On error, set an exception, leave the writer unchanged, and return `-1`.

int PyUnicodeWriter_DecodeUTF8Stateful(PyUnicodeWriter *writer, const char *string, Py_ssize_t length, const char *errors, Py_ssize_t *consumed)

Decode the string *str* from UTF-8 with *errors* error handler and write the output into *writer*.

size is the string length in bytes. If *size* is equal to `-1`, call `strlen(str)` to get the string length.

errors is an error handler name, such as `"replace"`. If *errors* is `NULL`, use the strict error handler.

If *consumed* is not `NULL`, set **consumed* to the number of decoded bytes on success. If *consumed* is `NULL`, treat trailing incomplete UTF-8 byte sequences as an error.

On success, return `0`. On error, set an exception, leave the writer unchanged, and return `-1`.

See also **`PyUnicodeWriter_WriteUTF8()`**.

Deprecated API

The following API is deprecated.

type Py_UNICODE

This is a typedef of **`wchar_t`**, which is a 16-bit type or 32-bit type depending on the platform. Please use **`wchar_t`** directly instead.

Distinto en la versión 3.3: En versiones anteriores, este era un tipo de 16 bits o de 32 bits, dependiendo de si seleccionó una versión Unicode «estrecha» o «amplia» de Python en el momento de la compilación.

Deprecated since version 3.13, will be removed in version 3.15.

int PyUnicode_READY(PyObject *unicode)

Do nothing and return `0`. This API is kept only for backward compatibility, but there are no plans to remove it.

Added in version 3.3.

Obsoleto desde la versión 3.10: This API does nothing since Python 3.12. Previously, this needed to be called for each string created using the old API (**`PyUnicode_FromUnicode()`** or similar).

unsigned int PyUnicode_IS_READY(PyObject *unicode)

Do nothing and return `1`. This API is kept only for backward compatibility, but there are no plans to remove it.

Added in version 3.3.

Obsoleto desde la versión 3.14: This API does nothing since Python 3.12. Previously, this could be called to check if **`PyUnicode_READY()`** is necessary.

Objetos tupla

type PyTupleObject

Este subtipo de **`PyObject`** representa un objeto tupla de Python.

PyTypeObject PyTuple_Type

Part of the Stable ABI.
Esta instancia de **`PyTypeObject`** representa el tipo tupla de Python; es el mismo objeto que **`tuple`** en la capa de Python.

int PyTuple_Check(PyObject *p)

Retorna verdadero si *p* es un objeto tupla o una instancia de un subtipo del tipo tupla. Esta función siempre finaliza con éxito.

int PyTuple_CheckExact(PyObject *p)

Retorna verdadero si *p* es un objeto tupla pero no una instancia de un subtipo del tipo tupla. Esta función siempre finaliza con éxito.

PyObject *PyTuple_New(Py_ssize_t len)

Return value: New reference. Part of the Stable ABI.
Retorna un nuevo objeto tupla de tamaño *len*, o `NULL` con una excepción establecida en caso de falla.

PyObject *PyTuple_Pack(Py_ssize_t n, ...)

Return value: New reference. Part of the Stable ABI.
Retorna un nuevo objeto tupla de tamaño *n*, o `NULL` con una excepción establecida en caso de falla. Los valores de tupla se inicializan con los siguientes *n* argumentos C que apuntan a objetos de Python. `PyTuple_Pack(2, a, b)` es equivalente a `Py_BuildValue("(OO)", a, b)`.

Py_ssize_t PyTuple_Size(PyObject *p)

Part of the Stable ABI.
Take a pointer to a tuple object, and return the size of that tuple. On error, return `-1` with an exception set.

Py_ssize_t PyTuple_GET_SIZE(PyObject *p)

Como **`PyTuple_Size()`**, pero sin verificación de errores.

PyObject *PyTuple_GetItem(PyObject *p, Py_ssize_t pos)

Return value: Borrowed reference. Part of the Stable ABI.
Retorna el objeto en la posición *pos* en la tupla señalada por *p*. Si *pos* es negativo o está fuera de los límites, retorna `NULL` y establece una excepción **`IndexError`**.

La referencia retornada es prestada de la tupla *p* (es decir: solo es válida mientras mantengas una referencia a *p*). Para obtener una referencia fuerte, usa **`Py_NewRef(PyTuple_GetItem(...))`** o **`PySequence_GetItem()`**.

PyObject *PyTuple_GET_ITEM(PyObject *p, Py_ssize_t pos)

Return value: Borrowed reference.
Como **`PyTuple_GetItem()`**, pero no verifica sus argumentos.

PyObject *PyTuple_GetSlice(PyObject *p, Py_ssize_t low, Py_ssize_t high)

Return value: New reference. Part of the Stable ABI.
Retorna la rebanada de la tupla apuntada por *p* entre *low* y *high*, o `NULL` con una excepción establecida en caso de falla.

Esto es equivalente a la expresión de Python `p[low:high]`. La indexación desde el final de la tupla no es compatible.

int PyTuple_SetItem(PyObject *p, Py_ssize_t pos, PyObject *o)

Part of the Stable ABI.
Inserta una referencia al objeto *o* en la posición *pos* de la

tupla señalada por *p*. Retorna `0` en caso de éxito. Si *pos* está fuera de límites, retorna `-1` y establece una excepción **`IndexError`**.

Nota

Esta función «roba» una referencia a *o* y descarta una referencia a un elemento que ya está en la tupla en la posición afectada.

void PyTuple_SET_ITEM(PyObject *p, Py_ssize_t pos, PyObject *o)

Al igual que **`PyTuple_SetItem()`**, pero no realiza ninguna comprobación de errores, y debe *solo* usarse para completar tuplas nuevas.

La comprobación de límites se realiza como una aserción si Python se construye en modo de depuración o **`con aserciones`**.

Nota

Esta función «roba» una referencia a *o* y, a diferencia de **`PyTuple_SetItem()`**, *no* descarta una referencia a ningún elemento que se está reemplazando; cualquier referencia en la tupla en la posición *pos* se filtrará.

Advertencia

This macro should *only* be used on tuples that are newly created. Using this macro on a tuple that is already in use (or in other words, has a refcount > 1) could lead to undefined behavior.

int _PyTuple_Resize(PyObject **p, Py_ssize_t newsize)

Se puede usar para cambiar el tamaño de una tupla. *newsize* será el nuevo tamaño de la tupla. Debido a que se *supone* que las tuplas son inmutables, esto solo debe usarse si solo hay una referencia al objeto. *No* use esto si la tupla ya puede ser conocida por alguna otra parte del código. La tupla siempre crecerá o disminuirá al final. Piense en esto como destruir la antigua tupla y crear una nueva, solo que de manera más eficiente. Retorna `0` en caso de éxito. El código del cliente nunca debe suponer que el valor resultante de `*p` será el mismo que antes de llamar a esta función. Si se reemplaza el objeto referenciado por `*p`, se destruye el original `*p`. En caso de fallo, retorna `-1` y establece `*p` en `NULL`, y lanza **`MemoryError`** o **`SystemError`**.

Objetos de secuencia de estructura

Los objetos de secuencia de estructura son el equivalente en C de los objetos **`namedtuple()`**, es decir, una secuencia a cuyos elementos también se puede acceder a través de atributos. Para crear una secuencia de estructura, primero debe crear un tipo de secuencia de estructura específico.

PyTypeObject *PyStructSequence_NewType(PyStructSequence_Desc *desc)

Return value: New reference. Part of the Stable ABI.
Crea un nuevo tipo de secuencia de estructura a partir de los datos en *desc*, que se describen a continuación. Las instancias del tipo resultante se pueden crear con **`PyStructSequence_New()`**.

Retorna `NULL` con una excepción establecida en caso de falla.

void PyStructSequence_InitType(PyTypeObject *type, PyStructSequence_Desc *desc)

Inicializa una secuencia de estructura tipo *type* desde *desc* en su lugar.

int PyStructSequence_InitType2(PyTypeObject *type, PyStructSequence_Desc *desc)

Como **`PyStructSequence_InitType()`**, pero retorna `0` en caso de éxito y `-1` con una excepción establecida en caso de falla.

Added in version 3.4.

type PyStructSequence_Desc

Part of the Stable ABI (including all members).
Contiene la meta información de un tipo de secuencia de estructura para crear.

const char *name

Fully qualified name of the type; null-terminated UTF-8 encoded. The name must contain the module name.

const char *doc

Puntero al *docstring* para el tipo o `NULL` para omitir

PyStructSequence_Field *fields

Puntero al arreglo terminado en `NULL` con nombres de campo del nuevo tipo

int n_in_sequence

Cantidad de campos visibles para el lado de Python (si se usa como tupla)

type PyStructSequence_Field

Part of the Stable ABI (including all members).
Describe un campo de una secuencia de estructura. Como una secuencia de estructura se modela como una tupla, todos los campos se escriben como PyObject*. El índice en el arreglo **`fields`** de **`PyStructSequence_Desc`** determina qué campo de la secuencia de estructura se describe.

const char *name

Nombre para el campo o `NULL` para finalizar la lista de campos con nombre, establece en **`PyStructSequence_UnnamedField`** para dejar sin nombre

const char *doc

Campo *docstring* o `NULL` para omitir

const char *const PyStructSequence_UnnamedField

Part of the Stable ABI since version 3.11.
Valor especial para un nombre de campo para dejarlo sin nombre.

Distinto en la versión 3.9: El tipo se cambió de `char *`.

PyObject *PyStructSequence_New(PyTypeObject *type)

Return value: New reference. Part of the Stable ABI.
Crea una instancia de *type*, que debe haberse creado con **`PyStructSequence_NewType()`**.

Retorna `NULL` con una excepción establecida en caso de falla.

PyObject *PyStructSequence_GetItem(PyObject *p, Py_ssize_t pos)

Return value: Borrowed reference. Part of the Stable ABI.
Retorna el objeto en la posición *pos* en la secuencia de estructura apuntada por *p*.

La comprobación de límites se realiza como una aserción si Python se construye en modo de depuración o **`con aserciones`**.

PyObject *PyStructSequence_GET_ITEM(PyObject *p, Py_ssize_t pos)

Return value: Borrowed reference.
Alias de **`PyStructSequence_GetItem()`**.

Distinto en la versión 3.13: Ahora implementado como un alias de **`PyStructSequence_GetItem()`**.

void PyStructSequence_SetItem(PyObject *p, Py_ssize_t pos, PyObject *o)

Part of the Stable ABI.
Establece el campo en el índice *pos* de la secuencia de estructura *p* en el valor *o*. Como `PyTuple_SET_ITEM()`, esto solo debe usarse para completar instancias nuevas.

La comprobación de límites se realiza como una aserción si Python se construye en modo de depuración o `con aserciones`.

Nota

Esta función «roba» una referencia a *o*.

void PyStructSequence_SET_ITEM(PyObject *p, Py_ssize_t *pos, PyObject *o)

Alias de `PyStructSequence_SetItem()`.

Distinto en la versión 3.13: Ahora implementado como un alias de `PyStructSequence_SetItem()`.

Objetos lista

type PyListObject

Este subtipo de `PyObject` representa un objeto lista de Python.

PyTypeObject PyList_Type

Part of the Stable ABI.

Esta instancia de `PyTypeObject` representa el tipo lista de Python. Este es el mismo objeto que `list` en la capa de Python.

int PyList_Check(PyObject *p)

Thread safety: Atomic.

Retorna verdadero si *p* es un objeto lista o una instancia de un subtipo del tipo lista. Esta función siempre finaliza con éxito.

int PyList_CheckExact(PyObject *p)

Thread safety: Atomic.

Retorna verdadero si *p* es un objeto lista, pero no una instancia de un subtipo del tipo lista. Esta función siempre finaliza con éxito.

PyObject *PyList_New(Py_ssize_t len)

Return value: New reference. Part of the Stable ABI. Thread safety: Atomic.

Retorna una nueva lista de longitud *len* en caso de éxito o `NULL` en caso de error.

Nota

Si *len* es mayor que cero, los elementos del objeto de la lista retornada se establecen en `NULL`. Por lo tanto, no puede utilizar funciones API abstractas como `PySequence_SetItem()` o exponer el objeto al código Python antes de configurar todos los elementos en un objeto real con `PyList_SetItem()` o `PyList_SET_ITEM()`. Las siguientes APIs son APIs seguras antes de que la lista esté completamente inicializada: **`PyList_SetItem()`** y **`PyList_SET_ITEM()`**.

Py_ssize_t PyList_Size(PyObject *list)

Part of the Stable ABI. Thread safety: Atomic.
Retorna la longitud del objeto lista en *list*; esto es equivalente a `len(list)` en un objeto lista.

Py_ssize_t PyList_GET_SIZE(PyObject *list)

Thread safety: Atomic.
Similar a `PyList_Size()`, pero sin comprobación de errores.

PyObject *PyList_GetItemRef(PyObject *list, Py_ssize_t index)

Return value: New reference. Part of the Stable ABI since version 3.13. Thread safety: Atomic.
Retorna el objeto en la posición *index* en la lista a la que apunta *list*. La posición no debe ser negativa; La indexación desde el final de la lista no es compatible. Si *index* está fuera de los límites (*<0 o >= len(list)*), retorna `NULL` y establece una excepción `IndexError`.

Added in version 3.13.

PyObject *PyList_GetItem(PyObject *list, Py_ssize_t index)

Return value: Borrowed reference. Part of the Stable ABI. Thread safety: Safe to call from multiple threads with external synchronization only.
Como `PyList_GetItemRef()`, pero retorna una referencia prestada en lugar de una referencia fuerte.

Nota

In the free-threaded build, the returned borrowed reference may become invalid if another thread modifies the list concurrently. Prefer `PyList_GetItemRef()`, which returns a strong reference.

PyObject *PyList_GET_ITEM(PyObject *list, Py_ssize_t i)

Return value: Borrowed reference. Thread safety: Safe to call from multiple threads with external synchronization only.
Similar a `PyList_GetItem()`, pero sin comprobación de errores.

Nota

In the free-threaded build, the returned borrowed reference may become invalid if another thread modifies the list concurrently. Prefer `PyList_GetItemRef()`, which returns a strong reference.

int PyList_SetItem(PyObject *list, Py_ssize_t index, PyObject *item)

Part of the Stable ABI. Thread safety: Atomic.
Establece el elemento en el índice *index* en la lista a *item*. Retorna `0` en caso de éxito. Si *index* está fuera de límites, retorna `-1` y establece una excepción `IndexError`.

Nota

Esta función «roba» una referencia a *item* y descarta una referencia a un elemento que ya está en la lista en la posición afectada.

void PyList_SET_ITEM(PyObject *list, Py_ssize_t i, PyObject *o)

Thread safety: Safe to call from multiple threads with external synchronization only.

Forma macro de **`PyList_SetItem()`** sin comprobación de errores. Esto normalmente solo se usa para completar nuevas listas donde no hay contenido anterior.

La verificación de límites se realiza como una aserción si Python se compila en modo de depuración o **`con aserciones`**.

Nota

Este macro «roba» una referencia a *item* y, a diferencia de **`PyList_SetItem()`**, *no descarta* una referencia a ningún elemento que se está reemplazando; cualquier referencia en *list* en la posición *i* se filtrará.

Nota

In the free-threaded build, this macro has no internal synchronization. It is normally only used to fill in new lists where no other thread has a reference to the list. If the list may be shared, use **`PyList_SetItem()`** instead, which uses a per-object lock.

int PyList_Insert(PyObject *list, Py_ssize_t index, PyObject *item)

Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Inserta el elemento *item* en la lista *list* delante del índice *index*. Retorna `0` si tiene éxito; retorna `-1` y establece una excepción si no tiene éxito. Análogo a `list.insert(index, item)`.

int PyList_Append(PyObject *list, PyObject *item)

Part of the Stable ABI. Thread safety: Atomic.
Agrega el objeto *item* al final de la lista *list*. Retorna `0` si tiene éxito; retorna `-1` y establece una excepción si no tiene éxito. Análogo a `list.append(item)`.

PyObject *PyList_GetSlice(PyObject *list, Py_ssize_t low, Py_ssize_t high)

Return value: New reference. Part of the Stable ABI. Thread safety: Atomic.
Retorna una lista de los objetos en *list* que contiene los objetos *between*, *low* y *high*. Retorna `NULL` y establece una excepción si no tiene éxito. Análogo a `list[low:high]`. La indexación desde el final de la lista no es compatible.

int PyList_SetSlice(PyObject *list, Py_ssize_t low, Py_ssize_t high, PyObject *itemlist)

Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Establece el segmento de *list* entre *low* y *high* para el contenido de *itemlist*. Análogo a `list[low:high] = itemlist`. La lista *itemlist* puede ser `NULL`, lo que indica la asignación de una lista vacía (eliminación de segmentos). Retorna `0` en caso de éxito, `-1` en caso de error. La indexación desde el final de la lista no es compatible.

Nota

In the free-threaded build, when *itemlist* is a **`list`**, both *list* and *itemlist* are locked for the duration of the operation. For other iterables (or `NULL`), only *list* is locked.

int PyList_Extend(PyObject *list, PyObject *iterable)

Thread safety: Safe for concurrent use on the same object.
Extiende *list* con los contenidos de *iterable*. Esto es lo mismo que `PyList_SetSlice(list, PY_SSIZE_T_MAX, PY_SSIZE_T_MAX, iterable)` y análogo a `list.extend(iterable)` o `list += iterable`.

Lanza una excepción y retorna `-1` si *list* no es un objeto **`list`**. Retorna 0 en caso de éxito.

Added in version 3.13.

Nota

In the free-threaded build, when *iterable* is a **`list`**, **`set`**, **`dict`**, or dict view, both *list* and *iterable* (or its underlying dict) are locked for the duration of the operation. For other iterables, only *list* is locked; *iterable* may be concurrently modified by another thread.

int PyList_Clear(PyObject *list)

Thread safety: Atomic.
Elimina todos los elementos de *list*. Esto es lo mismo que `PyList_SetSlice(list, 0, PY_SSIZE_T_MAX, NULL)` y análogo a `list.clear()` o `del list[:]`.

Lanza una excepción y retorna `-1` si *list* no es un objeto **`list`**. Retorna 0 en caso de éxito.

Added in version 3.13.

int PyList_Sort(PyObject *list)

Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Ordena los elementos de *list* en su lugar. Retorna `0` en caso de éxito, `-1` en caso de error. Esto es equivalente a `list.sort()`.

Nota

In the free-threaded build, element comparison via **`__lt__()`** can execute arbitrary Python code, during which the per-object lock may be temporarily released. For built-in types (**`str`**, **`int`**, **`float`**), the lock is not released during comparison.

int PyList_Reverse(PyObject *list)

Part of the Stable ABI. Thread safety: Safe for concurrent use on *the same object*.

Invierte los elementos de la lista *list* en su lugar. Retorna `0` en caso de éxito, `-1` en caso de error. Este es el equivalente de `list.reverse()`.

PyObject *PyList_AsTuple(PyObject *list)

Return value: New reference. Part of the Stable ABI. Thread safety: Atomic.

Retorna un nuevo objeto tupla que contiene el contenido de *list*; equivalente a `tuple(list)`.

Objetos diccionario

type PyDictObject

Este subtipo de **`PyObject`** representa un objeto diccionario de Python.

PyTypeObject PyDict_Type

Part of the Stable ABI.
Esta instancia de **`PyTypeObject`** representa el tipo diccionario de Python. Este es el mismo objeto que **`dict`** en la capa de Python.

int PyDict_Check(PyObject *p)

Thread safety: Atomic.
Retorna verdadero si *p* es un objeto `dict` o una instancia de un subtipo del tipo `dict`. Esta función siempre finaliza con éxito.

int PyDict_CheckExact(PyObject *p)

Thread safety: Atomic.
Retorna verdadero si *p* es un objeto `dict`, pero no una instancia de un subtipo del tipo `dict`. Esta función siempre finaliza con éxito.

PyObject *PyDict_New()

Return value: New reference. Part of the Stable ABI. Thread safety: Atomic.
Retorna un nuevo diccionario vacío, o `NULL` en caso de falla.

PyObject *PyDictProxy_New(PyObject *mapping)

Return value: New reference. Part of the Stable ABI.
Retorna un objeto a **`types.MappingProxyType`** para una asignación que imponga un comportamiento de solo lectura.

Esto normalmente se usa para crear una vista para evitar la modificación del diccionario para los tipos de clase no dinámicos.

PyTypeObject PyDictProxy_Type

Part of the Stable ABI.
The type object for mapping proxy objects created by **`PyDictProxy_New()`** and for the read-only `__dict__` attribute of many built-in types. A **`PyDictProxy_Type`** instance provides a dynamic, read-only view of an underlying dictionary: changes to the underlying dictionary are reflected in the proxy, but the proxy itself does not support mutation operations. This corresponds to **`types.MappingProxyType`** in Python.

void PyDict_Clear(PyObject *p)

Part of the Stable ABI. Thread safety: Atomic.
Vacía un diccionario existente de todos los pares clave-valor (*key-value*).

int PyDict_Contains(PyObject *p, PyObject *key)

Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Determine if dictionary *p* contains *key*. If an item in *p* matches *key*, return `1`, otherwise return `0`. On error, return `-1`. This is equivalent to the Python expression `key in p`.

Nota

The operation is atomic on free threading when *key* is **`str`**, **`int`**, **`float`**, **`bool`** or **`bytes`**.

int PyDict_ContainsString(PyObject *p, const char *key)

Thread safety: Atomic.
This is the same as **`PyDict_Contains()`**, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*.

Added in version 3.13.

PyObject *PyDict_Copy(PyObject *p)

Return value: New reference. Part of the Stable ABI. Thread safety: Atomic.
Retorna un nuevo diccionario que contiene los mismos pares clave-valor (*key-value*) que *p*.

int PyDict_SetItem(PyObject *p, PyObject *key, PyObject *val)

Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Inserta *val* en el diccionario *p* con una clave *key*. *key* debe ser hashable; si no lo es, se lanzará **`TypeError`**. Retorna `0` en caso de éxito o `-1` en caso de error. Esta función *no* roba una referencia a *val*.

Nota

The operation is atomic on free threading when *key* is **`str`**, **`int`**, **`float`**, **`bool`** or **`bytes`**.

int PyDict_SetItemString(PyObject *p, const char *key, PyObject *val)

Part of the Stable ABI. Thread safety: Atomic.
This is the same as **`PyDict_SetItem()`**, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*.

int PyDict_DelItem(PyObject *p, PyObject *key)

Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Remove the entry in dictionary *p* with key *key*. *key* must be hashable; if it isn't, **`TypeError`** is raised. If *key* is not in the dictionary, **`KeyError`** is raised. Return `0` on success or `-1` on failure.

Nota

The operation is atomic on free threading when *key* is `str`, `int`, `float`, `bool` or `bytes`.

int PyDict_DelItemString(PyObject *p, const char *key)

Part of the Stable ABI. Thread safety: Atomic.
This is the same as `PyDict_DelItem()`, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*.

int PyDict_GetItemRef(PyObject *p, PyObject *key, PyObject **result)

Part of the Stable ABI since version 3.13. Thread safety: Safe for concurrent use on the same object.
Return a new strong reference to the object from dictionary *p* which has a key *key*:

- If the key is present, set **result* to a new strong reference to the value and return `1`.
- If the key is missing, set **result* to `NULL` and return `0`.
- On error, raise an exception and return `-1`.

Nota

The operation is atomic on free threading when *key* is `str`, `int`, `float`, `bool` or `bytes`.

Added in version 3.13.

See also the `PyObject_GetItem()` function.

PyObject *PyDict_GetItem(PyObject *p, PyObject *key)

Return value: Borrowed reference. Part of the Stable ABI. Thread safety: Safe to call from multiple threads with external synchronization only.
Return a borrowed reference to the object from dictionary *p* which has a key *key*. Return `NULL` if the key *key* is missing

without setting an exception.

Nota

Exceptions that occur while this calls `__hash__()` and `__eq__()` methods are silently ignored. Prefer the `PyDict_GetItemWithError()` function instead.

Nota

In the free-threaded build, the returned borrowed reference may become invalid if another thread modifies the dictionary concurrently. Prefer `PyDict_GetItemRef()`, which returns a strong reference.

Distinto en la versión 3.10: Calling this API without an attached thread state had been allowed for historical reason. It is no longer allowed.

PyObject *PyDict_GetItemWithError(PyObject *p, PyObject *key)

Return value: Borrowed reference. Part of the Stable ABI. Thread safety: Safe to call from multiple threads with external synchronization only.
Variante de `PyDict_GetItem()` que no suprime las excepciones. Retorna `NULL` **con** una excepción establecida si se produjo una excepción. Retorna `NULL` **sin** una excepción establecida si la clave no estaba presente.

Nota

In the free-threaded build, the returned borrowed reference may become invalid if another thread modifies the dictionary concurrently. Prefer `PyDict_GetItemRef()`, which returns a strong reference.

PyObject *PyDict_GetItemString(PyObject *p, const char *key)

Return value: Borrowed reference. Part of the Stable ABI. Thread safety: Safe to call from multiple threads with external synchronization only.
This is the same as `PyDict_GetItem()`, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*.

Nota

Exceptions that occur while this calls `__hash__()` and `__eq__()` methods or while creating the temporary `str` object are silently ignored. Prefer using the `PyDict_GetItemWithError()` function with your own `PyUnicode_FromString()` *key* instead.

Nota

In the free-threaded build, the returned borrowed reference may become invalid if another thread modifies the dictionary concurrently. Prefer `PyDict_GetItemStringRef()`, which returns a strong reference.

int PyDict_GetItemStringRef(PyObject *p, const char *key, PyObject **result)

Part of the Stable ABI since version 3.13. Thread safety: Atomic.
Similar to `PyDict_GetItemRef()`, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*.

Added in version 3.13.

PyObject *PyDict_SetDefault(PyObject *p, PyObject *key, PyObject *defaultobj)

Return value: Borrowed reference. Thread safety: Safe to call from multiple threads with external synchronization only.
Esto es lo mismo al nivel de Python `dict.setdefault()`.

Si está presente, retorna el valor correspondiente a *key* del diccionario *p*. Si la clave no está en el `dict`, se inserta con el valor *defaultobj* y se retorna *defaultobj*. Esta función evalúa la función *hash* de *key* solo una vez, en lugar de evaluarla independientemente para la búsqueda y la inserción.

Added in version 3.4.

Nota

In the free-threaded build, the returned borrowed reference may become invalid if another thread modifies the dictionary concurrently. Prefer **`PyDict_SetDefaultRef()`**, which returns a strong reference.

int PyDict_SetDefaultRef(PyObject *p, PyObject *key, PyObject *default_value, PyObject **result)

Thread safety: Safe for concurrent use on the same object.
Inserts *default_value* into the dictionary *p* with a key of *key* if the key is not already present in the dictionary. If *result* is not `NULL`, then **result* is set to a strong reference to either *default_value*, if the key was not present, or the existing value, if *key* was already present in the dictionary. Returns `1` if the key was present and *default_value* was not inserted, or `0` if the key was not present and *default_value* was inserted. On failure, returns `-1`, sets an exception, and sets `*result` to `NULL`.

For clarity: if you have a strong reference to *default_value* before calling this function, then after it returns, you hold a strong reference to both *default_value* and **result* (if it's not `NULL`). These may refer to the same object: in that case you hold two separate references to it.

Nota

The operation is atomic on free threading when *key* is **`str`**, **`int`**, **`float`**, **`bool`** or **`bytes`**.

Added in version 3.13.

int PyDict_Pop(PyObject *p, PyObject *key, PyObject **result)

Thread safety: Safe for concurrent use on the same object.
Remove *key* from dictionary *p* and optionally return the removed value. Do not raise `KeyError` if the key is missing.

- If the key is present, set **result* to a new reference to the removed value if *result* is not `NULL`, and return `1`.
- If the key is missing, set **result* to `NULL` if *result* is not `NULL`, and return `0`.
- On error, raise an exception and return `-1`.

Similar to `dict.pop()`, but without the default value and not raising `KeyError` if the key is missing.

Nota

The operation is atomic on free threading when *key* is `str`, `int`, `float`, `bool` or `bytes`.

Added in version 3.13.

int PyDict_PopString(PyObject *p, const char *key, PyObject **result)

Thread safety: Atomic.
Similar to `PyDict_Pop()`, but *key* is specified as a const char* UTF-8 encoded bytes string, rather than a PyObject*.

Added in version 3.13.

PyObject *PyDict_Items(PyObject *p)

Return value: New reference. Part of the Stable ABI. Thread safety: Atomic.
Retorna un `PyListObject` que contiene todos los elementos del diccionario.

PyObject *PyDict_Keys(PyObject *p)

Return value: New reference. Part of the Stable ABI. Thread safety: Atomic.
Retorna un **`PyListObject`** que contiene todas las claves del diccionario.

PyObject *PyDict_Values(PyObject *p)

Return value: New reference. Part of the Stable ABI. Thread safety: Atomic.
Retorna un **`PyListObject`** que contiene todos los valores del diccionario *p*.

Py_ssize_t PyDict_Size(PyObject *p)

Part of the Stable ABI. Thread safety: Atomic.
Retorna el número de elementos en el diccionario. Esto es equivalente a `len(p)` en un diccionario.

Py_ssize_t PyDict_GET_SIZE(PyObject *p)

Thread safety: Atomic.
Similar to **`PyDict_Size()`**, but without error checking.

int PyDict_Next(PyObject *p, Py_ssize_t *ppos, PyObject **pkey, PyObject **pvalue)

Part of the Stable ABI. Thread safety: Safe to call from multiple threads with external synchronization only.
Iterate over all key-value pairs in the dictionary *p*. The **`Py_ssize_t`** referred to by *ppos* must be initialized to `0` prior to the first call to this function to start the iteration; the function returns true for each pair in the dictionary, and false once all pairs have been reported. The parameters *pkey* and *pvalue* should either point to PyObject* variables that will be filled in with each key and value, respectively, or may be `NULL`. Any references returned through them are borrowed. *ppos* should not be altered during iteration. Its value represents offsets within the internal dictionary structure, and since the structure is sparse, the offsets are not consecutive.

Por ejemplo:

```
PyObject *key, *value;
Py_ssize_t pos = 0;

while (PyDict_Next(self->dict, &pos, &key, &value))
    /* do something interesting with the values...
    ...
}
```

El diccionario *p* no debe mutarse durante la iteración. Es seguro modificar los valores de las claves a medida que recorre el diccionario, pero solo mientras el conjunto de claves no cambie. Por ejemplo:

```
PyObject *key, *value;
Py_ssize_t pos = 0;

while (PyDict_Next(self->dict, &pos, &key, &value))
    long i = PyLong_AsLong(value);
    if (i == -1 && PyErr_Occurred()) {
        return -1;
    }
    PyObject *o = PyLong_FromLong(i + 1);
    if (o == NULL)
        return -1;
    if (PyDict_SetItem(self->dict, key, o) < 0) {
        Py_DECREF(o);
        return -1;
    }
    Py_DECREF(o);
}
```

The function is not thread-safe in the free-threaded build without external synchronization. You can use **`Py_BEGIN_CRITICAL_SECTION`** to lock the dictionary while iterating over it:

```
Py_BEGIN_CRITICAL_SECTION(self->dict);
while (PyDict_Next(self->dict, &pos, &key, &value))
    ...
```

```
}
Py_END_CRITICAL_SECTION();
```

Nota

On the free-threaded build, this function can be used safely inside a critical section. However, the references returned for *pkey* and *pvalue* are borrowed and are only valid while the critical section is held. If you need to use these objects outside the critical section or when the critical section can be suspended, create a strong reference (for example, using **`Py_NewRef()`**).

int PyDict_Merge(PyObject *a, PyObject *b, int override)

Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Itera sobre el objeto de mapeo *b* agregando pares clave-valor al diccionario *a*. *b* puede ser un diccionario o cualquier objeto que soporte **`PyMapping_Keys()`** y **`PyObject_GetItem()`**. Si *override* es verdadero, los pares existentes en *a* se reemplazarán si se encuentra una clave coincidente en *b*, de lo contrario, los pares solo se agregarán si no hay una clave coincidente en *a*. Retorna `0` en caso de éxito o `-1` si se lanza una excepción.

Nota

In the free-threaded build, when *b* is a **`dict`** (with the standard iterator), both *a* and *b* are locked for the duration of the operation. When *b* is a non-dict mapping, only *a* is locked; *b* may be concurrently modified by another thread.

int PyDict_Update(PyObject *a, PyObject *b)

Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Esto es lo mismo que `PyDict_Merge(a, b, 1)` en C, y es similar a `a.update(b)` en Python excepto que

`PyDict_Update()` no vuelve a la iteración sobre una secuencia de pares de valores clave si el segundo argumento no tiene el atributo «claves». Retorna `0` en caso de éxito o `-1` si se produjo una excepción.

Nota

In the free-threaded build, when *b* is a `dict` (with the standard iterator), both *a* and *b* are locked for the duration of the operation. When *b* is a non-dict mapping, only *a* is locked; *b* may be concurrently modified by another thread.

int PyDict_MergeFromSeq2(PyObject *a, PyObject *seq2, int override)

Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.

Actualiza o combina en el diccionario *a*, desde los pares clave-valor en *seq2*. *seq2* debe ser un objeto iterable que produzca objetos iterables de longitud 2, vistos como pares clave-valor. En el caso de claves duplicadas, el último gana si *override* es verdadero, de lo contrario, el primero gana. Retorna `0` en caso de éxito o `-1` si se produjo una excepción. El equivalente en Python (excepto el valor de retorno)

```
def PyDict_MergeFromSeq2(a, seq2, override):
    for key, value in seq2:
        if override or key not in a:
            a[key] = value
```

Nota

In the free-threaded build, only *a* is locked. The iteration over *seq2* is not synchronized; *seq2* may be concurrently modified by another thread.

int PyDict_AddWatcher(PyDict_WatchCallback callback)

Thread safety: *Safe to call from multiple threads with external synchronization only*.
Register *callback* as a dictionary watcher. Return a non-negative integer id which must be passed to future calls to **`PyDict_Watch()`**. In case of error (e.g. no more watcher IDs available), return `-1` and set an exception.

Nota

This function is not internally synchronized. In the free-threaded build, callers should ensure no concurrent calls to **`PyDict_AddWatcher()`** or **`PyDict_ClearWatcher()`** are in progress.

Added in version 3.12.

int PyDict_ClearWatcher(int watcher_id)

Thread safety: *Safe to call from multiple threads with external synchronization only*.
Clear watcher identified by *watcher_id* previously returned from **`PyDict_AddWatcher()`**. Return `0` on success, `-1` on error (e.g. if the given *watcher_id* was never registered.)

Nota

This function is not internally synchronized. In the free-threaded build, callers should ensure no concurrent calls to **`PyDict_AddWatcher()`** or **`PyDict_ClearWatcher()`** are in progress.

Added in version 3.12.

int PyDict_Watch(int watcher_id, PyObject *dict)

Thread safety: *Safe to call without external synchronization on distinct objects*.
Mark dictionary *dict* as watched. The callback granted *watcher_id* by **`PyDict_AddWatcher()`** will be called when *dict* is modified or deallocated. Return `0` on success or `-1` on error.

Added in version 3.12.

int PyDict_Unwatch(int watcher_id, PyObject *dict)

Thread safety: Safe to call without external synchronization on distinct objects.

Mark dictionary *dict* as no longer watched. The callback granted *watcher_id* by **`PyDict_AddWatcher()`** will no longer be called when *dict* is modified or deallocated. The dict must previously have been watched by this watcher. Return `0` on success or `-1` on error.

Added in version 3.12.

type PyDict_WatchEvent

Enumeration of possible dictionary watcher events: `PyDict_EVENT_ADDED`, `PyDict_EVENT_MODIFIED`, `PyDict_EVENT_DELETED`, `PyDict_EVENT_CLONED`, `PyDict_EVENT_CLEARED`, or `PyDict_EVENT_DEALLOCATED`.

Added in version 3.12.

typedef int (*PyDict_WatchCallback)(PyDict_WatchEvent event, PyObject *dict, PyObject *key, PyObject *new_value)

Type of a dict watcher callback function.

If *event* is `PyDict_EVENT_CLEARED` or `PyDict_EVENT_DEALLOCATED`, both *key* and *new_value* will be `NULL`. If *event* is `PyDict_EVENT_ADDED` or `PyDict_EVENT_MODIFIED`, *new_value* will be the new value for *key*. If *event* is `PyDict_EVENT_DELETED`, *key* is being deleted from the dictionary and *new_value* will be `NULL`.

`PyDict_EVENT_CLONED` occurs when *dict* was previously empty and another dict is merged into it. To maintain efficiency of this operation, per-key `PyDict_EVENT_ADDED` events are not issued in this case; instead a single `PyDict_EVENT_CLONED` is issued, and *key* will be the source dictionary.

The callback may inspect but must not modify *dict*; doing so could have unpredictable effects, including infinite recursion. Do not trigger Python code execution in the callback, as it could modify the dict as a side effect.

If *event* is `PyDict_EVENT_DEALLOCATED`, taking a new reference in the callback to the about-to-be-destroyed dictionary will resurrect it and prevent it from being freed at this time. When the resurrected object is destroyed later, any watcher callbacks active at that time will be called again.

Callbacks occur before the notified modification to *dict* takes place, so the prior state of *dict* can be inspected.

If the callback sets an exception, it must return `-1`; this exception will be printed as an unraisable exception using **`PyErr_WriteUnraisable()`**. Otherwise it should return `0`.

There may already be a pending exception set on entry to the callback. In this case, the callback should return `0` with the same exception still set. This means the callback may not call any other API that can set an exception unless it saves and clears the exception state first, and restores it before returning.

Added in version 3.12.

Dictionary View Objects

int PyDictViewSet_Check(PyObject *op)

Return true if *op* is a view of a set inside a dictionary. This is currently equivalent to PyDictKeys_Check(op) || PyDictItems_Check(op). This function always succeeds.

PyTypeObject PyDictKeys_Type

Part of the Stable ABI.
Type object for a view of dictionary keys. In Python, this is the type of the object returned by **`dict.keys()`**.

int PyDictKeys_Check(PyObject *op)

Return true if *op* is an instance of a dictionary keys view. This function always succeeds.

PyTypeObject PyDictValues_Type

Part of the Stable ABI.
Type object for a view of dictionary values. In Python, this is the type of the object returned by **`dict.values()`**.

int PyDictValues_Check(PyObject *op)

Return true if *op* is an instance of a dictionary values view. This function always succeeds.

PyTypeObject PyDictItems_Type

Part of the Stable ABI.
Type object for a view of dictionary items. In Python, this is the type of the object returned by **`dict.items()`**.

int PyDictItems_Check(PyObject *op)

Return true if *op* is an instance of a dictionary items view. This function always succeeds.

Ordered Dictionaries

Python's C API provides interface for **`collections.OrderedDict`** from C. Since Python 3.7, dictionaries are ordered by default, so there is usually little need for these functions; prefer `PyDict*` where possible.

PyTypeObject PyODict_Type

Type object for ordered dictionaries. This is the same object as **`collections.OrderedDict`** in the Python layer.

int PyODict_Check(PyObject *od)

Return true if *od* is an ordered dictionary object or an instance of a subtype of the **`OrderedDict`** type. This function always succeeds.

int PyODict_CheckExact(PyObject *od)

Return true if *od* is an ordered dictionary object, but not an instance of a subtype of the `OrderedDict` type. This function always succeeds.

PyTypeObject PyODictKeys_Type

Analogous to `PyDictKeys_Type` for ordered dictionaries.

PyTypeObject PyODictValues_Type

Analogous to `PyDictValues_Type` for ordered dictionaries.

PyTypeObject PyODictItems_Type

Analogous to `PyDictItems_Type` for ordered dictionaries.

PyObject *PyODict_New(void)

Return a new empty ordered dictionary, or `NULL` on failure.

This is analogous to `PyDict_New()`.

int PyODict_SetItem(PyObject *od, PyObject *key, PyObject *value)

Insert *value* into the ordered dictionary *od* with a key of *key*. Return `0` on success or `-1` with an exception set on failure.

This is analogous to `PyDict_SetItem()`.

int PyODict_DelItem(PyObject *od, PyObject *key)

Remove the entry in the ordered dictionary *od* with key *key*. Return `0` on success or `-1` with an exception set on failure.

This is analogous to `PyDict_DelItem()`.

These are soft deprecated aliases to `PyDict` APIs:

PyODict	**PyDict**
PyODict_GetItem(od, key)	`PyDict_GetItem()`

PyODict_GetItemWithError(od, key)	`PyDict_GetItemWithError()`
PyODict_GetItemString(od, key)	`PyDict_GetItemString()`
PyODict_Contains(od, key)	`PyDict_Contains()`
PyODict_Size(od)	`PyDict_Size()`
PyODict_SIZE(od)	`PyDict_GET_SIZE()`

Objetos conjunto

Esta sección detalla la API pública de los objetos `set` y `frozenset`. Cualquier funcionalidad que no esté listada a continuación se accede mejor utilizando el protocolo abstracto de objetos (incluyendo `PyObject_CallMethod()`, `PyObject_RichCompareBool()`, `PyObject_Hash()`, `PyObject_Repr()`, `PyObject_IsTrue()`, `PyObject_Print()`, y `PyObject_GetIter()`) o el protocolo numérico abstracto (incluyendo `PyNumber_And()`, `PyNumber_Subtract()`, `PyNumber_Or()`, `PyNumber_Xor()`, `PyNumber_InPlaceAnd()`, `PyNumber_InPlaceSubtract()`, `PyNumber_InPlaceOr()`, y `PyNumber_InPlaceXor()`).

type PySetObject

Este subtipo de `PyObject` se utiliza para mantener los datos internos de los objetos `set` y `frozenset`. Es como un `PyDictObject` en el sentido de que tiene un tamaño fijo para los conjuntos pequeños (muy parecido al almacenamiento de tuplas) y apuntará a un bloque de memoria separado y de tamaño variable para los conjuntos de tamaño medio y grande (muy parecido al almacenamiento de listas). Ninguno de los campos de esta estructura debe considerarse público y todos están sujetos a cambios. Todo el acceso debe hacerse a través de la API documentada en lugar de manipular los valores de la estructura.

PyTypeObject PySet_Type

Part of the Stable ABI.

Esta es una instancia de `PyTypeObject` que representa el tipo Python `set`.

PyTypeObject PyFrozenSet_Type

Part of the Stable ABI.

Esta es una instancia de `PyTypeObject` que representa el tipo Python `frozenset`.

Los siguientes macros de comprobación de tipos funcionan en punteros a cualquier objeto de Python. Del mismo modo, las funciones del constructor funcionan con cualquier objeto Python iterable.

int PySet_Check(PyObject *p)

Retorna verdadero si *p* es un objeto `set` o una instancia de un subtipo. Esta función siempre finaliza con éxito.

int PyFrozenSet_Check(PyObject *p)

Retorna verdadero si *p* es un objeto `frozenset` o una instancia de un subtipo. Esta función siempre finaliza con éxito.

int PyAnySet_Check(PyObject *p)

Retorna verdadero si *p* es un objeto `set`, un objeto `frozenset`, o una instancia de un subtipo. Esta función siempre finaliza con éxito.

int PySet_CheckExact(PyObject *p)

Retorna verdadero si *p* es un objeto `set` pero no una instancia de un subtipo. Esta función siempre finaliza con éxito.

Added in version 3.10.

int PyAnySet_CheckExact(PyObject *p)

Retorna verdadero si *p* es un objeto `set` o un objeto `frozenset` pero no una instancia de un subtipo. Esta función siempre finaliza con éxito.

int PyFrozenSet_CheckExact(PyObject *p)

Retorna verdadero si *p* es un objeto `frozenset` pero no una instancia de un subtipo. Esta función siempre finaliza con éxito.

PyObject *PySet_New(PyObject *iterable)

Return value: New reference. Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Retorna un nuevo **`set`** que contiene objetos retornados por *iterable*. El *iterable* puede ser `NULL` para crear un nuevo conjunto vacío. Retorna el nuevo conjunto en caso de éxito o `NULL` en caso de error. Lanza **`TypeError`** si *iterable* no es realmente iterable. El constructor también es útil para copiar un conjunto (`c=set(s)`).

Nota

The operation is atomic on free threading when *iterable* is a **`set`**, **`frozenset`** or **`dict`**.

PyObject *PyFrozenSet_New(PyObject *iterable)

Return value: New reference. Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Retorna un nuevo **`frozenset`** que contiene objetos retornados por *iterable*. El *iterable* puede ser `NULL` para crear un nuevo frozenset vacío. Retorna el nuevo conjunto en caso de éxito o `NULL` en caso de error. Lanza **`TypeError`** si *iterable* no es realmente iterable.

Nota

The operation is atomic on free threading when *iterable* is a **`set`**, **`frozenset`** or **`dict`**.

Las siguientes funciones y macros están disponibles para instancias de **`set`** o **`frozenset`** o instancias de sus subtipos.

Py_ssize_t PySet_Size(PyObject *anyset)

Part of the Stable ABI. Thread safety: Atomic.
Retorna la longitud de un objeto **`set`** o **`frozenset`**. Equivalente a `len(anyset)`. Lanza un **`SystemError`** si

anyset no es `set`, `frozenset`, o una instancia de un subtipo.

Py_ssize_t PySet_GET_SIZE(PyObject *anyset)

Thread safety: Atomic.

Forma macro de `PySet_Size()` sin comprobación de errores.

int PySet_Contains(PyObject *anyset, PyObject *key)

Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.

Retorna `1` si se encuentra, `0` si no se encuentra y `-1` si se encuentra un error. A diferencia del método Python `__contains__()`, esta función no convierte automáticamente conjuntos que no son hashable en frozensets temporales. Lanza un `TypeError` si la *key* no es hashable. Lanza `SystemError` si *anyset* no es un `set`, `frozenset`, o una instancia de un subtipo.

Nota

The operation is atomic on free threading when *key* is `str`, `int`, `float`, `bool` or `bytes`.

int PySet_Add(PyObject *set, PyObject *key)

Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.

Añade *key* a una instancia `set`. También funciona con instancias de `frozenset` (al igual que `PyTuple_SetItem()` puede usarse para rellenar los valores de nuevos frozensets antes de que sean expuestos a otro código). Retorna `0` en caso de éxito o `-1` en caso de fallo. Lanza un error `TypeError` si la *key* no es hashable. Lanza un `MemoryError` si no hay espacio para crecer. Lanza un `SystemError` si *set* no es una instancia de `set` o su subtipo.

Nota

The operation is atomic on free threading when *key* is **`str`**, **`int`**, **`float`**, **`bool`** or **`bytes`**.

Las siguientes funciones están disponibles para instancias de **`set`** o sus subtipos, pero no para instancias de **`frozenset`** o sus subtipos.

int PySet_Discard(PyObject *set, PyObject *key)

Part of the Stable ABI. Thread safety: Safe for concurrent use on the same object.
Return `1` if found and removed, `0` if not found (no action taken), and `-1` if an error is encountered. Does not raise **`KeyError`** for missing keys. Raise a **`TypeError`** if the *key* is unhashable. Unlike the Python **`discard()`** method, this function does not automatically convert unhashable sets into temporary frozensets. Raise **`SystemError`** if *set* is not an instance of **`set`** or its subtype.

Nota

The operation is atomic on free threading when *key* is **`str`**, **`int`**, **`float`**, **`bool`** or **`bytes`**.

PyObject *PySet_Pop(PyObject *set)

Return value: New reference. Part of the Stable ABI. Thread safety: Atomic.
Retorna una nueva referencia a un objeto arbitrario en el *set* y elimina el objeto del *set*. Retorna `NULL` en caso de falla. Lanza **`KeyError`** si el conjunto está vacío. Lanza a **`SystemError`** si *set* no es una instancia de **`set`** o su subtipo.

int PySet_Clear(PyObject *set)

Part of the Stable ABI. Thread safety: Atomic.
Vacía todos los elementos de un conjunto existente. Retorna `0` en caso de éxito. Retorna `-1` y lanza **`SystemError`** si *set* no es una instancia de **`set`** o su subtipo.

Nota

In the free-threaded build, the set is emptied before its entries are cleared, so other threads will observe an empty set rather than intermediate states.

Deprecated API

PySet_MINSIZE

A soft deprecated constant representing the size of an internal preallocated table inside `PySetObject` instances.

This is documented solely for completeness, as there are no guarantees that a given version of CPython uses preallocated tables with a fixed size. In code that does not deal with unstable set internals, **`PySet_MINSIZE`** can be replaced with a small constant like `8`.

If looking for the size of a set, use `PySet_Size()` instead.

Objetos función

Hay algunas funciones específicas para las funciones de Python.

type PyFunctionObject

La estructura C utilizada para las funciones.

PyTypeObject PyFunction_Type

Esta es una instancia de **`PyTypeObject`** y representa el tipo función de Python. Está expuesto a los programadores de Python como `types.FunctionType`.

int PyFunction_Check(PyObject *o)

Retorna verdadero si *o* es un objeto función (tiene tipo **`PyFunction_Type`**). El parámetro no debe ser `NULL`. Esta función siempre finaliza con éxito.

PyObject *PyFunction_New(PyObject *code, PyObject *globals)

Return value: New reference.
Retorna un nuevo objeto función asociado con el objeto código *code*. *globals* debe ser un diccionario con las variables globales accesibles para la función.

El docstring y el nombre de la función se obtienen del objeto código. **`__module__`** se obtiene de *globals*. Los argumentos por defecto, anotaciones y clausura se establecen en `NULL`. **`__qualname__`** se establece en el mismo valor que el campo **`co_qualname`** del objeto código.

PyObject *PyFunction_NewWithQualName(PyObject *code, PyObject *globals, PyObject *qualname)

Return value: New reference.
Como **`PyFunction_New()`**, pero también permite configurar el atributo **`__qualname__`** del objeto función.

qualname debe ser un objeto unicode o `NULL`; si es `NULL`, el atributo **`__qualname__`** se establece en el mismo valor que el campo **`co_qualname`** del objeto código.

Added in version 3.3.

PyObject *PyFunction_GetCode(PyObject *op)

Return value: Borrowed reference.
Retorna el objeto código asociado con el objeto función *op*.

PyObject *PyFunction_GetGlobals(PyObject *op)

Return value: Borrowed reference.
Retorna el diccionario global asociado con el objeto función *op*.

PyObject *PyFunction_GetModule(PyObject *op)

Return value: Borrowed reference.
Retorna una referencia prestada al atributo **`__module__`** del objeto función *op*. Puede ser *NULL*.

Esto es normalmente una **`cadena de caracteres`** que contiene el nombre del módulo, pero se puede establecer en cualquier otro objeto mediante código Python.

PyObject *PyFunction_GetDefaults(PyObject *op)

Return value: Borrowed reference.
Retorna los valores predeterminados del argumento del objeto función *op*. Esto puede ser una tupla de argumentos o `NULL`.

int PyFunction_SetDefaults(PyObject *op, PyObject *defaults)

Establece los valores predeterminados del argumento para el objeto función *op*. *defaults* deben ser `Py_None` o una tupla.

Lanza **`SystemError`** y retorna `-1` en caso de error.

void PyFunction_SetVectorcall(PyFunctionObject *func, vectorcallfunc vectorcall)

Establece el campo vectorcall de un objeto función dado *func*.

Advertencia: ¡las extensiones que usan esta API deben preservar el comportamiento de la función vectorcall inalterada (por defecto)!

Added in version 3.12.

PyObject *PyFunction_GetKwDefaults(PyObject *op)

Return value: Borrowed reference.
Return the keyword-only argument default values of the function object *op*. This can be a dictionary of arguments or `NULL`.

int PyFunction_SetKwDefaults(PyObject *op, PyObject *defaults)

Set the keyword-only argument default values of the function object *op*. *defaults* must be a dictionary of keyword-only arguments or `Py_None`.

This function returns `0` on success, and returns `-1` with an exception set on failure.

PyObject *PyFunction_GetClosure(PyObject *op)

Return value: Borrowed reference.
Retorna el cierre asociado con el objeto función *op*. Esto puede ser `NULL` o una tupla de objetos celda.

int PyFunction_SetClosure(PyObject *op, PyObject *closure)

Establece el cierre asociado con el objeto función *op*. *cierre* debe ser `Py_None` o una tupla de objetos celda.

Lanza **`SystemError`** y retorna `-1` en caso de error.

PyObject *PyFunction_GetAnnotations(PyObject *op)

Return value: Borrowed reference.
Retorna las anotaciones del objeto función *op*. Este puede ser

un diccionario mutable o `NULL`.

int PyFunction_SetAnnotations(PyObject *op, PyObject *annotations)

Establece las anotaciones para el objeto función *op*. *annotations* debe ser un diccionario o `Py_None`.

Lanza **`SystemError`** y retorna `-1` en caso de error.

PyObject *PyFunction_GET_CODE(PyObject *op)
PyObject *PyFunction_GET_GLOBALS(PyObject *op)
PyObject *PyFunction_GET_MODULE(PyObject *op)
PyObject *PyFunction_GET_DEFAULTS(PyObject *op)
PyObject *PyFunction_GET_KW_DEFAULTS(PyObject *op)
PyObject *PyFunction_GET_CLOSURE(PyObject *op)
PyObject *PyFunction_GET_ANNOTATIONS(PyObject *op)

Return value: Borrowed reference.
These functions are similar to their `PyFunction_Get*` counterparts, but do not do type checking. Passing anything other than an instance of **`PyFunction_Type`** is undefined behavior.

int PyFunction_AddWatcher(PyFunction_WatchCallback callback)

Registra *callback* como un observador de función para el intérprete actual. Retorna un ID que puede pasarse a **`PyFunction_ClearWatcher()`**. En caso de error (por ejemplo, no hay más IDs de observador disponibles), retorna `-1` y establece una excepción.

Added in version 3.12.

int PyFunction_ClearWatcher(int watcher_id)

Limpia el observador identificado por *watcher_id* previamente retornado de **`PyFunction_AddWatcher()`** para el intérprete actual. Retorna `0` en caso de éxito, o `-1` y establece una excepción en caso de error (por ejemplo, si el *watcher_id* dado nunca fue registrado).

Added in version 3.12.

type PyFunction_WatchEvent

Enumeration of possible function watcher events:

- `PyFunction_EVENT_CREATE`
- `PyFunction_EVENT_DESTROY`
- `PyFunction_EVENT_MODIFY_CODE`
- `PyFunction_EVENT_MODIFY_DEFAULTS`
- `PyFunction_EVENT_MODIFY_KWDEFAULTS`

Added in version 3.12.

typedef int (*PyFunction_WatchCallback)(PyFunction_WatchEvent event, PyFunctionObject *func, PyObject *new_value)

Tipo de una función callback de observador de función.

Si *event* es `PyFunction_EVENT_CREATE` o `PyFunction_EVENT_DESTROY` entonces *new_value* será `NULL`. De lo contrario, *new_value* mantendrá una referencia prestada al nuevo valor que está a punto de almacenarse en *func* para el atributo que se está modificando.

El callback puede inspeccionar pero no debe modificar *func*; hacerlo podría tener efectos impredecibles, incluyendo recursión infinita.

If *event* is `PyFunction_EVENT_CREATE`, then the callback is invoked after *func* has been fully initialized. Otherwise, the callback is invoked before the modification to *func* takes place, so the prior state of *func* can be inspected. The runtime is permitted to optimize away the creation of function objects when possible. In such cases no event will be emitted. Although this creates the possibility of an observable difference of runtime behavior depending on optimization decisions, it does not change the semantics of the Python code being executed.

Si *event* es `PyFunction_EVENT_DESTROY`, tomar una referencia en el callback a la función a punto de ser destruida la resucitará, evitando que sea liberada en este momento. Cuando el objeto resucitado sea destruido más tarde, cualquier callback observador activo en ese momento será llamado de nuevo.

Si el callback establece una excepción, debe retornar `-1`; esta excepción se imprimirá como una excepción no lanzable usando **`PyErr_WriteUnraisable()`**. De lo contrario debe retornar `0`.

Puede haber ya una excepción pendiente establecida al entrar al callback. En este caso, el callback debe retornar `0` con la misma excepción aún establecida. Esto significa que el callback no puede llamar a ninguna otra API que pueda establecer una excepción a menos que guarde y borre el estado de la excepción primero, y lo restaure antes de retornar.

Added in version 3.12.

Objetos de método de instancia

An instance method is a wrapper for a **`PyCFunction`** and the new way to bind a **`PyCFunction`** to a class object. It replaces the former call `PyMethod_New(func, NULL, class)`.

PyTypeObject PyInstanceMethod_Type

Esta instancia de **`PyTypeObject`** representa el tipo de método de instancia de Python. No está expuesto a los programas de Python.

int PyInstanceMethod_Check(PyObject *o)

Retorna verdadero si *o* es un objeto de método de instancia (tiene tipo **`PyInstanceMethod_Type`**). El parámetro no debe ser `NULL`. Esta función siempre finaliza con éxito.

PyObject *PyInstanceMethod_New(PyObject *func)

Return value: New reference.
Return a new instance method object, with *func* being any callable object. *func* is the function that will be called when the instance method is called.

PyObject *PyInstanceMethod_Function(PyObject *im)

Return value: Borrowed reference.
Retorna el objeto de función asociado con el método de instancia *im*.

PyObject *PyInstanceMethod_GET_FUNCTION(PyObject *im)

Return value: Borrowed reference.
Versión macro de **`PyInstanceMethod_Function()`** que evita la comprobación de errores.

Objetos método

Los métodos son objetos de función enlazados. Los métodos siempre están vinculados a una instancia de una clase definida por el usuario. Los métodos no vinculados (métodos vinculados a un objeto de clase) ya no están disponibles.

PyTypeObject PyMethod_Type

Esta instancia de **`PyTypeObject`** representa el tipo de método Python. Esto está expuesto a los programas de Python como `types.MethodType`.

int PyMethod_Check(PyObject *o)

Retorna verdadero si *o* es un objeto de método (tiene tipo **`PyMethod_Type`**). El parámetro no debe ser `NULL`. Esta función siempre finaliza con éxito.

PyObject *PyMethod_New(PyObject *func, PyObject *self)

Return value: New reference.
Retorna un nuevo objeto de método, con *func* como cualquier objeto invocable y *self* la instancia en la que se debe vincular el método. *func* es la función que se llamará cuando se llame al método. *self* no debe ser `NULL`.

PyObject *PyMethod_Function(PyObject *meth)

Return value: Borrowed reference.
Retorna el objeto de función asociado con el método *meth*.

PyObject *PyMethod_GET_FUNCTION(PyObject *meth)

Return value: Borrowed reference.
Versión macro de **`PyMethod_Function()`** que evita la comprobación de errores.

PyObject *PyMethod_Self(PyObject *meth)

Return value: Borrowed reference.
Retorna la instancia asociada con el método *meth*.

PyObject *PyMethod_GET_SELF(PyObject *meth)

Return value: Borrowed reference.
Versión macro de `PyMethod_Self()` que evita la comprobación de errores.

Objetos celda

«Cell» objects are used to implement variables referenced by multiple scopes. For each such variable, a cell object is created to store the value; the local variables of each stack frame that references the value contain a reference to the cells from outer scopes which also use that variable. When the value is accessed, the value contained in the cell is used instead of the cell object itself. This de-referencing of the cell object requires support from the generated byte-code; these are not automatically de-referenced when accessed. Cell objects are not likely to be useful elsewhere.

type PyCellObject

La estructura C utilizada para objetos celda.

PyTypeObject PyCell_Type

El objeto tipo correspondiente a los objetos celda.

int PyCell_Check(PyObject *ob)

Retorna verdadero si *ob* es un objeto de celda; *ob* no debe ser `NULL`. Esta función siempre finaliza con éxito.

PyObject *PyCell_New(PyObject *ob)

Return value: New reference.
Crea y retorna un nuevo objeto de celda que contiene el valor *ob*. El parámetro puede ser `NULL`.

PyObject *PyCell_Get(PyObject *cell)

Return value: New reference.
Return the contents of the cell *cell*, which can be `NULL`. If *cell* is not a cell object, returns `NULL` with an exception set.

PyObject *PyCell_GET(PyObject *cell)

Return value: Borrowed reference.
Retorna el contenido de la celda *cell*, pero sin verificar que *cell* no sea `NULL` y que sea un objeto de celda.

int PyCell_Set(PyObject *cell, PyObject *value)

Set the contents of the cell object *cell* to *value*. This releases the reference to any current content of the cell. *value* may be `NULL`. *cell* must be non-`NULL`.

On success, return `0`. If *cell* is not a cell object, set an exception and return `-1`.

void PyCell_SET(PyObject *cell, PyObject *value)

Establece el valor del objeto de celda *cell* en el valor *value*. No se ajustan los recuentos de referencia y no se realizan verificaciones de seguridad; *cell* no debe ser `NULL` y debe ser un objeto de celda.

Objetos código

Los objetos código son un detalle de bajo nivel de la implementación de CPython. Cada uno representa un fragmento de código ejecutable que aún no se ha vinculado a una función.

type PyCodeObject

La estructura en C de los objetos utilizados para describir objetos código. Los campos de este tipo están sujetos a cambios en cualquier momento.

PyTypeObject PyCode_Type

This is an instance of **`PyTypeObject`** representing the Python code object.

int PyCode_Check(PyObject *co)

Return true if *co* is a code object. This function always succeeds.

Py_ssize_t PyCode_GetNumFree(PyCodeObject *co)

Return the number of free (closure) variables in a code object.

int PyUnstable_Code_GetFirstFree(PyCodeObject *co)

> *This is Unstable API. It may change without warning in minor releases.*

Return the position of the first free (closure) variable in a code object.

Distinto en la versión 3.13: Renamed from `PyCode_GetFirstFree` as part of Unstable C API. The old name is deprecated, but will remain available until the

signature changes again.

PyCodeObject *PyUnstable_Code_New(int argcount, int kwonlyargcount, int nlocals, int stacksize, int flags, PyObject *code, PyObject *consts, PyObject *names, PyObject *varnames, PyObject *freevars, PyObject *cellvars, PyObject *filename, PyObject *name, PyObject *qualname, int firstlineno, PyObject *linetable, PyObject *exceptiontable)

> *This is Unstable API. It may change without warning in minor releases.*

Return a new code object. If you need a dummy code object to create a frame, use **`PyCode_NewEmpty()`** instead.

Since the definition of the bytecode changes often, calling **`PyUnstable_Code_New()`** directly can bind you to a precise Python version.

The many arguments of this function are inter-dependent in complex ways, meaning that subtle changes to values are likely to result in incorrect execution or VM crashes. Use this function only with extreme care.

Distinto en la versión 3.11: Added `qualname` and `exceptiontable` parameters.

Distinto en la versión 3.12: Renamed from `PyCode_New` as part of Unstable C API. The old name is deprecated, but will remain available until the signature changes again.

PyCodeObject *PyUnstable_Code_NewWithPosOnlyArgs(int argcount, int posonlyargcount, int kwonlyargcount, int nlocals, int stacksize, int flags, PyObject *code, PyObject *consts, PyObject *names, PyObject *varnames, PyObject *freevars, PyObject *cellvars, PyObject *filename, PyObject *name, PyObject *qualname, int firstlineno, PyObject *linetable, PyObject *exceptiontable)

This is Unstable API. It may change without warning in minor releases.

Similar to **`PyUnstable_Code_New()`**, but with an extra «posonlyargcount» for positional-only arguments. The same caveats that apply to `PyUnstable_Code_New` also apply to this function.

Added in version 3.8: as `PyCode_NewWithPosOnlyArgs`

Distinto en la versión 3.11: Added `qualname` and `exceptiontable` parameters.

Distinto en la versión 3.12: Renamed to `PyUnstable_Code_NewWithPosOnlyArgs`. The old name is deprecated, but will remain available until the signature changes again.

PyCodeObject *PyCode_NewEmpty(const char *filename, const char *funcname, int firstlineno)

Return value: New reference.
Retorna un nuevo objeto de código vacío con el nombre de archivo especificado, el nombre de la función y el número de la primera línea. Si el objeto código resultante es ejecutado, lanzará una `Exception`.

int PyCode_Addr2Line(PyCodeObject *co, int byte_offset)

Retorna el número de línea de la instrucción que se produce en o antes de `byte_offset` y finaliza después. Si solo necesita el número de línea de un marco, use **`PyFrame_GetLineNumber()`** en su lugar.

For efficiently iterating over the line numbers in a code object, use **the API described in PEP 626** [https://peps.python.org/pep-0626/#out-of-process-debuggers-and-profilers].

int PyCode_Addr2Location(PyObject *co, int byte_offset, int *start_line, int *start_column, int *end_line, int *end_column)

Establece los punteros `int` pasados en los números de línea y columna del código fuente para las instrucciones en `byte_offset`. Establece el valor en `0` cuando la información no está disponible para algún elemento en particular.

Retorna `1` si la función fue exitosa y `0` de lo contrario.

Added in version 3.11.

PyObject *PyCode_GetCode(PyCodeObject *co)

Equivalente al código Python `getattr(co, 'co_code')`. Retorna una referencia fuerte a un **`PyBytesObject`** representando el bytecode en un objecto código. En caso de error se retorna `NULL` y se lanza una excepción.

Este `PyBytesObject` puede ser creado a pedido del intérprete y no necesariamente representa el bytecode que es realmente ejecutado por CPython. Los casos de uso principales para esta función son depuradores y perfiladores.

Added in version 3.11.

PyObject *PyCode_GetVarnames(PyCodeObject *co)

Equivalente al código Python `getattr(co, 'co_varnames')`. Retorna una nueva referencia a un **`PyTupleObject`** que contiene los nombres de las variables locales. En caso de error, retorna `NULL` y lanza una excepción.

Added in version 3.11.

PyObject *PyCode_GetCellvars(PyCodeObject *co)

Equivalente al código Python `getattr(co, 'co_cellvars')`. Retorna una nueva referencia a un **`PyTupleObject`** que contiene los nombres de las variables locales referenciadas por funciones anidadas. En caso de error, retorna `NULL` y lanza una excepción.

Added in version 3.11.

PyObject *PyCode_GetFreevars(PyCodeObject *co)

Equivalent to the Python code `getattr(co, 'co_freevars')`. Returns a new reference to a **`PyTupleObject`** containing the names of the free (closure) variables. On error, `NULL` is returned and an exception is raised.

Added in version 3.11.

int PyCode_AddWatcher(PyCode_WatchCallback callback)

Register *callback* as a code object watcher for the current interpreter. Return an ID which may be passed to **`PyCode_ClearWatcher()`**. In case of error (e.g. no more watcher IDs available), return `-1` and set an exception.

Added in version 3.12.

int PyCode_ClearWatcher(int watcher_id)

Clear watcher identified by *watcher_id* previously returned from **`PyCode_AddWatcher()`** for the current interpreter. Return `0` on success, or `-1` and set an exception on error (e.g. if the given *watcher_id* was never registered.)

Added in version 3.12.

type PyCodeEvent

Enumeration of possible code object watcher events: - `PY_CODE_EVENT_CREATE` - `PY_CODE_EVENT_DESTROY`

Added in version 3.12.

typedef int (*PyCode_WatchCallback)(PyCodeEvent event, PyCodeObject *co)

Type of a code object watcher callback function.

If *event* is `PY_CODE_EVENT_CREATE`, then the callback is invoked after *co* has been fully initialized. Otherwise, the callback is invoked before the destruction of *co* takes place, so the prior state of *co* can be inspected.

If *event* is `PY_CODE_EVENT_DESTROY`, taking a reference in the callback to the about-to-be-destroyed code object will resurrect it and prevent it from being freed at this time. When the resurrected object is destroyed later, any watcher callbacks active at that time will be called again.

Users of this API should not rely on internal runtime implementation details. Such details may include, but are not limited to, the exact order and timing of creation and destruction of code objects. While changes in these details may result in differences observable by watchers (including whether a callback is invoked or not), it does not change the semantics of the Python code being executed.

If the callback sets an exception, it must return `-1`; this exception will be printed as an unraisable exception using **`PyErr_WriteUnraisable()`**. Otherwise it should return `0`.

There may already be a pending exception set on entry to the callback. In this case, the callback should return `0` with the same exception still set. This means the callback may not call any other API that can set an exception unless it saves and clears the exception state first, and restores it before returning.

Added in version 3.12.

PyObject *PyCode_Optimize(PyObject *code, PyObject *consts, PyObject *names, PyObject *lnotab_obj)

This is a soft deprecated function that does nothing.

Prior to Python 3.10, this function would perform basic optimizations to a code object.

Distinto en la versión 3.10: This function now does nothing.

Code Object Flags

Code objects contain a bit-field of flags, which can be retrieved as the `co_flags` Python attribute (for example using `PyObject_GetAttrString()`), and set using a *flags* argument to `PyUnstable_Code_New()` and similar functions.

Flags whose names start with `CO_FUTURE_` correspond to features normally selectable by future statements. These flags can be used in `PyCompilerFlags.cf_flags`. Note that many `CO_FUTURE_` flags are mandatory in current versions of Python, and setting them has no effect.

The following flags are available. For their meaning, see the linked documentation of their Python equivalents.

Flag	Meaning
CO_OPTIMIZED	`inspect.CO_OPTIMIZED`
CO_NEWLOCALS	`inspect.CO_NEWLOCALS`
CO_VARARGS	`inspect.CO_VARARGS`
CO_VARKEYWORDS	`inspect.CO_VARKEYWORDS`
CO_NESTED	`inspect.CO_NESTED`
CO_GENERATOR	`inspect.CO_GENERATOR`
CO_COROUTINE	`inspect.CO_COROUTINE`
CO_ITERABLE_COROUTINE	`inspect.CO_ITERABLE_COROUTINE`
CO_ASYNC_GENERATOR	`inspect.CO_ASYNC_GENERATOR`
CO_HAS_DOCSTRING	`inspect.CO_HAS_DOCSTRING`
CO_METHOD	`inspect.CO_METHOD`
CO_FUTURE_DIVISION	no effect (`__future__.division`)
CO_FUTURE_ABSOLUTE_IMPORT	no effect (`__future__.absolute_import`)
CO_FUTURE_WITH_STATEMENT	no effect (`__future__.with_statement`)
CO_FUTURE_PRINT_FUNCTION	no effect (`__future__.print_function`)
CO_FUTURE_UNICODE_LITERALS	no effect (`__future__.unicode_literals`)
CO_FUTURE_GENERATOR_STOP	no effect (`__future__.generator_stop`)
CO_FUTURE_ANNOTATIONS	`__future__.annotations`

Extra information

To support low-level extensions to frame evaluation, such as external just-in-time compilers, it is possible to attach arbitrary extra data to code objects.

These functions are part of the unstable C API tier: this functionality is a CPython implementation detail, and the API may change without deprecation warnings.

Py_ssize_t PyUnstable_Eval_RequestCodeExtraIndex(freefunc free)

> *This is Unstable API. It may change without warning in minor releases.*

Return a new opaque index value used to adding data to code objects.

You generally call this function once (per interpreter) and use the result with `PyCode_GetExtra` and `PyCode_SetExtra` to manipulate data on individual code objects.

If *free* is not `NULL`: when a code object is deallocated, *free* will be called on non-`NULL` data stored under the new index. Use **`Py_DecRef()`** when storing **`PyObject`**.

Added in version 3.6: as `_PyEval_RequestCodeExtraIndex`

Distinto en la versión 3.12: Renamed to `PyUnstable_Eval_RequestCodeExtraIndex`. The old private name is deprecated, but will be available until the API changes.

int PyUnstable_Code_GetExtra(PyObject *code, Py_ssize_t index, void **extra)

> *This is Unstable API. It may change without warning in minor*

releases.

Set *extra* to the extra data stored under the given index. Return 0 on success. Set an exception and return -1 on failure.

If no data was set under the index, set *extra* to `NULL` and return 0 without setting an exception.

Added in version 3.6: as `_PyCode_GetExtra`

Distinto en la versión 3.12: Renamed to `PyUnstable_Code_GetExtra`. The old private name is deprecated, but will be available until the API changes.

int PyUnstable_Code_SetExtra(PyObject *code, Py_ssize_t index, void *extra)

This is Unstable API. It may change without warning in minor releases.

Set the extra data stored under the given index to *extra*. Return 0 on success. Set an exception and return -1 on failure.

Added in version 3.6: as `_PyCode_SetExtra`

Distinto en la versión 3.12: Renamed to `PyUnstable_Code_SetExtra`. The old private name is deprecated, but will be available until the API changes.

File objects

These APIs are a minimal emulation of the Python 2 C API for built-in file objects, which used to rely on the buffered I/O (FILE*) support from the C standard library. In Python 3, files and streams use the new `io` module, which defines several layers over the low-level unbuffered I/O of the operating system. The functions described below are convenience C wrappers over these new APIs, and meant mostly for internal error reporting in the interpreter; third-party code is advised to access the `io` APIs instead.

PyObject *PyFile_FromFd(int fd, const char *name, const char *mode, int buffering, const char *encoding, const char *errors, const char *newline, int closefd)

> *Return value: New reference. Part of the Stable ABI.*
> Crea un objeto archivo Python a partir del descriptor de archivo de un archivo ya abierto *fd*. Los argumentos *name*, *encoding*, *errors* y *newline* pueden ser `NULL` para usar los valores predeterminados; *buffering* puede ser *-1* para usar el valor predeterminado. *name* se ignora y se mantiene por compatibilidad con versiones anteriores. Retorna `NULL` en caso de error. Para obtener una descripción más completa de los argumentos, consulte la documentación de la función `io.open()`.
>
> **Advertencia**
>
> Dado que las transmisiones (*streams*) de Python tienen su propia capa de almacenamiento en búfer, combinarlas con descriptores de archivos a nivel del sistema operativo puede producir varios problemas (como un pedido inesperado de datos).
>
> *Distinto en la versión 3.2:* Ignora el atributo *name*.

int PyObject_AsFileDescriptor(PyObject *p)

Part of the Stable ABI.
Return the file descriptor associated with *p* as an int. If the object is an integer, its value is returned. If not, the object's **`fileno()`** method is called if it exists; the method must return an integer, which is returned as the file descriptor value. Sets an exception and returns `-1` on failure.

PyObject *PyFile_GetLine(PyObject *p, int n)

Return value: New reference. Part of the Stable ABI.
Equivalente a `p.readline([n])`, esta función lee una línea del objeto *p*. *p* puede ser un objeto archivo o cualquier objeto con un método **`readline()`**. Si *n* es `0`, se lee exactamente una línea, independientemente de la longitud de la línea. Si *n* es mayor que `0`, no se leerán más de *n* bytes del archivo; se puede retornar una línea parcial. En ambos casos, se retorna una cadena de caracteres vacía si se llega al final del archivo de inmediato. Si *n* es menor que `0`, sin embargo, se lee una línea independientemente de la longitud, pero **`EOFError`** se lanza si se llega al final del archivo de inmediato.

int PyFile_SetOpenCodeHook(Py_OpenCodeHookFunction handler)

Sobrescribe el comportamiento normal de **`io.open_code()`** para pasar su parámetro a través del controlador proporcionado.

The *handler* is a function of type:

typedef PyObject *(*Py_OpenCodeHookFunction)(PyObject*, void*)

Equivalent of PyObject *(*)(PyObject *path, void *userData), where *path* is guaranteed to be **`PyUnicodeObject`**.

El puntero *userData* se pasa a la función de enlace. Dado que las funciones de enlace pueden llamarse desde diferentes tiempos de ejecución, este puntero no debe referirse directamente al estado de Python.

Como este *hook* se usa intencionalmente durante la importación, evite importar nuevos módulos durante su ejecución a menos que se sepa que están congelados o disponibles en `sys.modules`.

Una vez que se ha establecido un *hook*, no se puede quitar ni reemplazar, y luego llamadas a **`PyFile_SetOpenCodeHook()`** fallarán. En caso de error, la función retorna -1 y establece una excepción si el intérprete se ha inicializado.

Es seguro llamar a esta función antes de **`Py_Initialize()`**.

Genera un evento de auditoría `setopencodehook` sin argumentos.

Added in version 3.8.

PyObject *PyFile_OpenCodeObject(PyObject *path)

Open *path* with the mode `'rb'`. *path* must be a Python **`str`** object. The behavior of this function may be overridden by **`PyFile_SetOpenCodeHook()`** to allow for some preprocessing of the text.

This is analogous to **`io.open_code()`** in Python.

On success, this function returns a strong reference to a Python file object. On failure, this function returns `NULL` with an exception set.

Added in version 3.8.

PyObject *PyFile_OpenCode(const char *path)

Similar to **`PyFile_OpenCodeObject()`**, but *path* is a UTF-8 encoded const char*.

Added in version 3.8.

int PyFile_WriteObject(PyObject *obj, PyObject *p, int flags)

Part of the Stable ABI.

Write object *obj* to file object *p*. The only supported flag for *flags* is **`Py_PRINT_RAW`**; if given, the **`str()`** of the object is written instead of the **`repr()`**.

If *obj* is `NULL`, write the string `"<NULL>"`.

Return `0` on success or `-1` on failure; the appropriate exception will be set.

int PyFile_WriteString(const char *s, PyObject *p)

Part of the Stable ABI.

Escribe la cadena *s* en el objeto archivo *p*. Retorna `0` en caso de éxito o `-1` en caso de error; se establecerá la excepción apropiada.

Soft-deprecated API

Soft deprecated since version 3.15.

These are APIs that were included in Python's C API by mistake. They are documented solely for completeness; use other `PyFile*` APIs instead.

PyObject *PyFile_NewStdPrinter(int fd)

Use **`PyFile_FromFd()`** with defaults (`fd, NULL, "w", -1, NULL, NULL, NULL, 0`) instead.

PyTypeObject PyStdPrinter_Type

Type of file-like objects used internally at Python startup when **`io`** is not yet available. Use Python **`open()`** or **`PyFile_FromFd()`** to create file objects instead.

Objetos módulo

PyTypeObject PyModule_Type

Part of the Stable ABI.
This instance of `PyTypeObject` represents the Python module type. This is exposed to Python programs as `types.ModuleType`.

int PyModule_Check(PyObject *p)

Retorna verdadero si *p* es un objeto de módulo o un subtipo de un objeto de módulo. Esta función siempre finaliza con éxito.

int PyModule_CheckExact(PyObject *p)

Retorna verdadero si *p* es un objeto módulo, pero no un subtipo de `PyModule_Type`. Esta función siempre finaliza con éxito.

PyObject *PyModule_NewObject(PyObject *name)

Return value: New reference. Part of the Stable ABI since version 3.7.
Return a new module object with `module.__name__` set to *name*. The module's `__name__`, `__doc__`, `__package__` and `__loader__` attributes are filled in (all but `__name__` are set to `None`). The caller is responsible for setting a `__file__` attribute.

Return `NULL` with an exception set on error.

Added in version 3.3.

Distinto en la versión 3.4: `__package__` and `__loader__` are now set to `None`.

PyObject *PyModule_New(const char *name)

Return value: New reference. Part of the Stable ABI.
Similar a **`PyModule_NewObject()`**, pero el nombre es una cadena de caracteres codificada UTF-8 en lugar de un objeto Unicode.

PyObject *PyModule_GetDict(PyObject *module)

Return value: Borrowed reference. Part of the Stable ABI.
Retorna el objeto del diccionario que implementa el espacio de nombres de *module*; este objeto es el mismo que el atributo **`__dict__`** del objeto módulo. Si *module* no es un objeto módulo (o un subtipo de un objeto de módulo), se lanza **`SystemError`** y se retorna `NULL`.

It is recommended extensions use other `PyModule_*` and `PyObject_*` functions rather than directly manipulate a module's **`__dict__`**.

The returned reference is borrowed from the module; it is valid until the module is destroyed.

PyObject *PyModule_GetNameObject(PyObject *module)

Return value: New reference. Part of the Stable ABI since version 3.7.
Return *module*'s **`__name__`** value. If the module does not provide one, or if it is not a string, **`SystemError`** is raised and `NULL` is returned.

Added in version 3.3.

const char *PyModule_GetName(PyObject *module)

Part of the Stable ABI.
Similar a **`PyModule_GetNameObject()`** pero retorna el nombre codificado a `'utf-8'`.

The returned buffer is only valid until the module is renamed or destroyed. Note that Python code may rename a module by setting its **`__name__`** attribute.

void *PyModule_GetState(PyObject *module)

Part of the Stable ABI.
Retorna el «estado» del módulo, es decir, un puntero al bloque de memoria asignado en el momento de la creación del módulo, o `NULL`. Ver **`PyModuleDef.m_size`**.

PyModuleDef *PyModule_GetDef(PyObject *module)

Part of the Stable ABI.
Retorna un puntero a la estructura **`PyModuleDef`** a partir de la cual se creó el módulo, o `NULL` si el módulo no se creó a partir de una definición.

On error, return `NULL` with an exception set. Use **`PyErr_Occurred()`** to tell this case apart from a missing **`PyModuleDef`**.

PyObject *PyModule_GetFilenameObject(PyObject *module)

Return value: New reference. Part of the Stable ABI.
Return the name of the file from which *module* was loaded using *module*'s **`__file__`** attribute. If this is not defined, or if it is not a string, raise **`SystemError`** and return `NULL`; otherwise return a reference to a Unicode object.

Added in version 3.2.

const char *PyModule_GetFilename(PyObject *module)

Part of the Stable ABI.
Similar a **`PyModule_GetFilenameObject()`** pero retorna el nombre de archivo codificado a "utf-8".

The returned buffer is only valid until the module's **`__file__`** attribute is reassigned or the module is destroyed.

Obsoleto desde la versión 3.2: **`PyModule_GetFilename()`** raises **`UnicodeEncodeError`** on unencodable filenames, use **`PyModule_GetFilenameObject()`** instead.

Module definitions

The functions in the previous section work on any module object, including modules imported from Python code.

Modules defined using the C API typically use a *module definition*, **`PyModuleDef`** – a statically allocated, constant “description» of how a module should be created.

The definition is usually used to define an extension’s “main” module object (see Defining extension modules for details). It is also used to create extension modules dynamically.

Unlike **`PyModule_New()`**, the definition allows management of *module state* – a piece of memory that is allocated and cleared together with the module object. Unlike the module’s Python attributes, Python code cannot replace or delete data stored in module state.

type PyModuleDef

> *Part of the Stable ABI (including all members).*
> The module definition struct, which holds all information needed to create a module object. This structure must be statically allocated (or be otherwise guaranteed to be valid while any modules created from it exist). Usually, there is only one variable of this type for each extension module.
>
> PyModuleDef_Base m_base
>
> > Always initialize this member to **`PyModuleDef_HEAD_INIT`**.
>
> const char *m_name
>
> > Nombre para el nuevo módulo.
>
> const char *m_doc
>
> > Docstring para el módulo; por lo general, se usa una variable docstring creada con **`PyDoc_STRVAR`**.

Py_ssize_t m_size

El estado del módulo se puede mantener en un área de memoria por módulo que se puede recuperar con **`PyModule_GetState()`**, en lugar de en globales estáticos. Esto hace que los módulos sean seguros para su uso en múltiples sub-interpretadores.

This memory area is allocated based on *m_size* on module creation, and freed when the module object is deallocated, after the **`m_free`** function has been called, if present.

Setting it to a non-negative value means that the module can be re-initialized and specifies the additional amount of memory it requires for its state.

Setting `m_size` to `-1` means that the module does not support sub-interpreters, because it has global state. Negative `m_size` is only allowed when using legacy single-phase initialization or when creating modules dynamically.

Ver **PEP 3121** [https://peps.python.org/pep-3121/] para más detalles.

PyMethodDef *m_methods

Un puntero a una tabla de funciones de nivel de módulo, descrito por valores **`PyMethodDef`**. Puede ser `NULL` si no hay funciones presentes.

PyModuleDef_Slot *m_slots

An array of slot definitions for multi-phase initialization, terminated by a `{0, NULL}` entry. When using legacy single-phase initialization, *m_slots* must be `NULL`.

Distinto en la versión 3.5: Antes de la versión 3.5, este miembro siempre estaba configurado en `NULL` y se definía como:

inquiry m_reload

traverseproc m_traverse

Una función transversal para llamar durante el recorrido GC del objeto del módulo, o `NULL` si no es necesario.

This function is not called if the module state was requested but is not allocated yet. This is the case immediately after the module is created and before the module is executed (**`Py_mod_exec`** function). More precisely, this function is not called if **`m_size`** is greater than 0 and the module state (as returned by **`PyModule_GetState()`**) is `NULL`.

Distinto en la versión 3.9: Ya no se llama antes de que se asigne el estado del módulo.

inquiry m_clear

Una función clara para llamar durante la limpieza GC del objeto del módulo, o `NULL` si no es necesario.

This function is not called if the module state was requested but is not allocated yet. This is the case immediately after the module is created and before the module is executed (**`Py_mod_exec`** function). More precisely, this function is not called if **`m_size`** is greater than 0 and the module state (as returned by **`PyModule_GetState()`**) is `NULL`.

Tal como **`PyTypeObject.tp_clear`**, esta función no *siempre* es llamada antes de la designación de un módulo. Por ejemplo, cuando el recuento de referencias está listo para determinar que un objeto no se usa más, la recolección de basura cíclica no se involucra y se llama a **`m_free`** directamente.

Distinto en la versión 3.9: Ya no se llama antes de que se asigne el estado del módulo.

freefunc m_free

Una función para llamar durante la desasignación del objeto del módulo, o `NULL` si no es necesario.

This function is not called if the module state was requested but is not allocated yet. This is the case immediately after the module is created and before the module is executed (**`Py_mod_exec`** function). More precisely, this function is not called if **`m_size`** is greater than 0 and the module state (as returned by **`PyModule_GetState()`**) is `NULL`.

Distinto en la versión 3.9: Ya no se llama antes de que se asigne el estado del módulo.

PyTypeObject PyModuleDef_Type

Part of the Stable ABI since version 3.5.
The type of `PyModuleDef` objects.

Module slots

type PyModuleDef_Slot

Part of the Stable ABI (including all members) since version 3.5.

int slot

Una ranura ID, elegida entre los valores disponibles que se explican a continuación.

void *value

Valor de la ranura, cuyo significado depende de la ID de la ranura.

Added in version 3.5.

Los tipos de ranura disponibles son:

Py_mod_create

Part of the Stable ABI since version 3.5.

Especifica una función que se llama para crear el objeto del módulo en sí. El puntero *value* de este espacio debe apuntar a una función de la firma:

PyObject *create_module(PyObject *spec, PyModuleDef *def)

La función recibe una instancia de **`ModuleSpec`**, como se define en **PEP 451** [https://peps.python.org/pep-0451/], y la definición del módulo. Debería retornar un nuevo objeto de módulo, o establecer un error y retornar `NULL`.

Esta función debe mantenerse mínima. En particular, no debería llamar a código arbitrario de Python, ya que intentar importar el mismo módulo nuevamente puede dar como resultado un bucle infinito.

Múltiples ranuras `Py_mod_create` no pueden especificarse en una definición de módulo.

Si no se especifica `Py_mod_create`, la maquinaria de importación creará un objeto de módulo normal usando **`PyModule_New()`**. El nombre se toma de *spec*, no de la definición, para permitir que los módulos de extensión se ajusten dinámicamente a su lugar en la jerarquía de módulos y se importen bajo diferentes nombres a través de enlaces simbólicos, todo mientras se comparte una definición de módulo único.

No es necesario que el objeto retornado sea una instancia de **`PyModule_Type`**. Se puede usar cualquier tipo, siempre que admita la configuración y la obtención de atributos relacionados con la importación. Sin embargo, solo se pueden retornar instancias `PyModule_Type` si el `PyModuleDef` no tiene `NULL` `m_traverse`, `m_clear`, `m_free`; `m_size` distinto de cero; o ranuras que no sean `Py_mod_create`.

Added in version 3.5.

Py_mod_exec

Part of the Stable ABI since version 3.5.

Especifica una función que se llama para ejecutar (*execute*) el módulo. Esto es equivalente a ejecutar el código de un módulo Python: por lo general, esta función agrega clases y constantes al módulo. La firma de la función es:

int exec_module(PyObject *module)

Si se especifican varias ranuras `Py_mod_exec`, se procesan en el orden en que aparecen en el arreglo *m_slots*.

Added in version 3.5.

Py_mod_multiple_interpreters

Part of the Stable ABI since version 3.12.
Specifies one of the following values:

Py_MOD_MULTIPLE_INTERPRETERS_NOT_SUPPORTED

The module does not support being imported in subinterpreters.

Py_MOD_MULTIPLE_INTERPRETERS_SUPPORTED

The module supports being imported in subinterpreters, but only when they share the main interpreter's GIL. (See Aislamiento de módulos de extensión.)

Py_MOD_PER_INTERPRETER_GIL_SUPPORTED

The module supports being imported in subinterpreters, even when they have their own GIL. (See Aislamiento de módulos de extensión.)

This slot determines whether or not importing this module in a subinterpreter will fail.

Multiple `Py_mod_multiple_interpreters` slots may not be specified in one module definition.

If `Py_mod_multiple_interpreters` is not specified, the import machinery defaults to `Py_MOD_MULTIPLE_INTERPRETERS_SUPPORTED`.

Added in version 3.12.

Py_mod_gil

Part of the Stable ABI since version 3.13.
Specifies one of the following values:

Py_MOD_GIL_USED

The module depends on the presence of the global interpreter lock (GIL), and may access global state without synchronization.

Py_MOD_GIL_NOT_USED

The module is safe to run without an active GIL.

This slot is ignored by Python builds not configured with **`--disable-gil`**. Otherwise, it determines whether or not importing this module will cause the GIL to be automatically enabled. See Free-threaded CPython for more detail.

Multiple `Py_mod_gil` slots may not be specified in one module definition.

If `Py_mod_gil` is not specified, the import machinery defaults to `Py_MOD_GIL_USED`.

Added in version 3.13.

Creating extension modules dynamically

The following functions may be used to create a module outside of an extension's initialization function. They are also used in single-phase initialization.

PyObject *PyModule_Create(PyModuleDef *def)

Return value: New reference.
Create a new module object, given the definition in *def*. This is a macro that calls **`PyModule_Create2()`** with *module_api_version* set to **`PYTHON_API_VERSION`**, or to **`PYTHON_ABI_VERSION`** if using the limited API.

PyObject *PyModule_Create2(PyModuleDef *def, int module_api_version)

Return value: New reference. Part of the Stable ABI.
Crea un nuevo objeto de módulo, dada la definición en *def*, asumiendo la versión de API *module_api_version*. Si esa versión no coincide con la versión del intérprete en ejecución, se emite un **`RuntimeWarning`**.

Return `NULL` with an exception set on error.

This function does not support slots. The **`m_slots`** member of *def* must be `NULL`.

Nota

La mayoría de los usos de esta función deberían usar **`PyModule_Create()`** en su lugar; solo use esto si está seguro de que lo necesita.

PyObject *PyModule_FromDefAndSpec(PyModuleDef *def, PyObject *spec)

Return value: New reference.
This macro calls **`PyModule_FromDefAndSpec2()`** with *module_api_version* set to **`PYTHON_API_VERSION`**, or to **`PYTHON_ABI_VERSION`** if using the limited API.

Added in version 3.5.

PyObject *PyModule_FromDefAndSpec2(PyModuleDef *def, PyObject *spec, int module_api_version)

Return value: New reference. Part of the Stable ABI since version

3.7.
Create a new module object, given the definition in *def* and the ModuleSpec *spec*, assuming the API version *module_api_version*. If that version does not match the version of the running interpreter, a **`RuntimeWarning`** is emitted.

Return `NULL` with an exception set on error.

Note that this does not process execution slots (**`Py_mod_exec`**). Both `PyModule_FromDefAndSpec` and `PyModule_ExecDef` must be called to fully initialize a module.

Nota

La mayoría de los usos de esta función deberían usar **`PyModule_FromDefAndSpec()`** en su lugar; solo use esto si está seguro de que lo necesita.

Added in version 3.5.

int PyModule_ExecDef(PyObject *module, PyModuleDef *def)

Part of the Stable ABI since version 3.7.
Procesa cualquier ranura de ejecución (**`Py_mod_exec`**) dado en *def*.

Added in version 3.5.

PYTHON_API_VERSION

The C API version. Defined for backwards compatibility.

Currently, this constant is not updated in new Python versions, and is not useful for versioning. This may change in the future.

PYTHON_ABI_VERSION

Defined as `3` for backwards compatibility.

Currently, this constant is not updated in new Python versions, and is not useful for versioning. This may change in the future.

Funciones de soporte

The following functions are provided to help initialize a module state. They are intended for a module's execution slots (**`Py_mod_exec`**), the initialization function for legacy single-phase initialization, or code that creates modules dynamically.

int PyModule_AddObjectRef(PyObject *module, const char *name, PyObject *value)

Part of the Stable ABI since version 3.10.
Agrega un objeto a *module* como *name*. Esta es una función de conveniencia que se puede usar desde la función de inicialización del módulo.

En caso de éxito, retorna `0`. En caso de error, lanza una excepción y retorna `-1`.

Ejemplo de uso

```
static int
add_spam(PyObject *module, int value)
{
    PyObject *obj = PyLong_FromLong(value);
    if (obj == NULL) {
        return -1;
    }
    int res = PyModule_AddObjectRef(module, "spam",
    Py_DECREF(obj);
    return res;
 }
```

To be convenient, the function accepts `NULL` *value* with an exception set. In this case, return `-1` and just leave the raised exception unchanged.

El ejemplo puede también ser escrito sin verificar explicitamente si *obj* es `NULL`:

```
static int
add_spam(PyObject *module, int value)
{
    PyObject *obj = PyLong_FromLong(value);
    int res = PyModule_AddObjectRef(module, "spam",
    Py_XDECREF(obj);
    return res;
 }
```

Note que `Py_XDECREF()` debería ser usado en vez de `Py_DECREF()` en este caso, ya que *obj* puede ser `NULL`.

The number of different *name* strings passed to this function should be kept small, usually by only using statically allocated strings as *name*. For names that aren't known at compile time, prefer calling **`PyUnicode_FromString()`** and **`PyObject_SetAttr()`** directly. For more details, see **`PyUnicode_InternFromString()`**, which may be used internally to create a key object.

Added in version 3.10.

int PyModule_Add(PyObject *module, const char *name, PyObject *value)

Part of the Stable ABI since version 3.13.
Similar to **`PyModule_AddObjectRef()`**, but «steals» a reference to *value*. It can be called with a result of function that returns a new reference without bothering to check its result or even saving it to a variable.

Ejemplo de uso

```
if (PyModule_Add(module, "spam", PyBytes_FromString
    goto error;
}
```

Added in version 3.13.

int PyModule_AddObject(PyObject *module, const char *name, PyObject *value)

Part of the Stable ABI.
Similar a **`PyModule_AddObjectRef()`**, pero roba una referencia a *value* en caso de éxito (en este caso retorna `0`).

The new **`PyModule_Add()`** or **`PyModule_AddObjectRef()`** functions are recommended, since it is easy to introduce reference leaks by misusing the **`PyModule_AddObject()`** function.

> **Nota**
>
> Unlike other functions that steal references, `PyModule_AddObject()` only releases the reference to *value* **on success.**
>
> This means that its return value must be checked, and calling code must **`Py_XDECREF()`** *value* manually on error.

Ejemplo de uso

```
PyObject *obj = PyBytes_FromString(value);
if (PyModule_AddObject(module, "spam", obj) < 0) {
    // If 'obj' is not NULL and PyModule_AddObject(
    // 'obj' strong reference must be deleted with
    // If 'obj' is NULL, Py_XDECREF() does nothing.
    Py_XDECREF(obj);
    goto error;
}
// PyModule_AddObject() stole a reference to obj:
// Py_XDECREF(obj) is not needed here.
```

Soft deprecated since version 3.13.

int PyModule_AddIntConstant(PyObject *module, const char

*name, long value)

Part of the Stable ABI.
Add an integer constant to *module* as *name*. This convenience function can be used from the module's initialization function. Return `-1` with an exception set on error, `0` on success.

This is a convenience function that calls **`PyLong_FromLong()`** and **`PyModule_AddObjectRef()`**; see their documentation for details.

int PyModule_AddStringConstant(PyObject *module, const char *name, const char *value)

Part of the Stable ABI.
Add a string constant to *module* as *name*. This convenience function can be used from the module's initialization function. The string *value* must be `NULL`-terminated. Return `-1` with an exception set on error, `0` on success.

This is a convenience function that calls **`PyUnicode_InternFromString()`** and **`PyModule_AddObjectRef()`**; see their documentation for details.

PyModule_AddIntMacro(module, macro)

Add an int constant to *module*. The name and the value are taken from *macro*. For example `PyModule_AddIntMacro(module, AF_INET)` adds the int constant *AF_INET* with the value of *AF_INET* to *module*. Return `-1` with an exception set on error, `0` on success.

PyModule_AddStringMacro(module, macro)

Agrega una constante de cadena de caracteres a *module*.

int PyModule_AddType(PyObject *module, PyTypeObject *type)

Part of the Stable ABI since version 3.10.
Add a type object to *module*. The type object is finalized by

calling internally **`PyType_Ready()`**. The name of the type object is taken from the last component of **`tp_name`** after dot. Return `-1` with an exception set on error, `0` on success.

Added in version 3.9.

int PyModule_AddFunctions(PyObject *module, PyMethodDef *functions)

Part of the Stable ABI since version 3.7.
Add the functions from the `NULL` terminated *functions* array to *module*. Refer to the **`PyMethodDef`** documentation for details on individual entries (due to the lack of a shared module namespace, module level «functions» implemented in C typically receive the module as their first parameter, making them similar to instance methods on Python classes).

This function is called automatically when creating a module from `PyModuleDef` (such as when using Multi-phase initialization, `PyModule_Create`, or `PyModule_FromDefAndSpec`). Some module authors may prefer defining functions in multiple **`PyMethodDef`** arrays; in that case they should call this function directly.

The *functions* array must be statically allocated (or otherwise guaranteed to outlive the module object).

Added in version 3.5.

int PyModule_SetDocString(PyObject *module, const char *docstring)

Part of the Stable ABI since version 3.7.
Set the docstring for *module* to *docstring*. This function is called automatically when creating a module from `PyModuleDef` (such as when using Multi-phase initialization, `PyModule_Create`, or `PyModule_FromDefAndSpec`).

Return `0` on success. Return `-1` with an exception set on error.

Added in version 3.5.

int PyUnstable_Module_SetGIL(PyObject *module, void *gil)

This is Unstable API. It may change without warning in minor releases.

Indicate that *module* does or does not support running without the global interpreter lock (GIL), using one of the values from **`Py_mod_gil`**. It must be called during *module*'s initialization function when using Legacy single-phase initialization. If this function is not called during module initialization, the import machinery assumes the module does not support running without the GIL. This function is only available in Python builds configured with **`--disable-gil`**. Return `-1` with an exception set on error, `0` on success.

Added in version 3.13.

Module lookup (single-phase initialization)

The legacy single-phase initialization initialization scheme creates singleton modules that can be looked up in the context of the current interpreter. This allows the module object to be retrieved later with only a reference to the module definition.

Estas funciones no funcionarán en módulos creados mediante la inicialización de múltiples fases, ya que se pueden crear múltiples módulos de este tipo desde una sola definición.

PyObject *PyState_FindModule(PyModuleDef *def)

Return value: Borrowed reference. Part of the Stable ABI.
Retorna el objeto módulo que se creó a partir de *def* para el intérprete actual. Este método requiere que el objeto módulo se haya adjuntado al estado del intérprete con **`PyState_AddModule()`** de antemano. En caso de que el objeto módulo correspondiente no se encuentre o no se haya

adjuntado al estado del intérprete, retornará `NULL`.

int PyState_AddModule(PyObject *module, PyModuleDef *def)

Part of the Stable ABI since version 3.3.
Adjunta el objeto del módulo pasado a la función al estado del intérprete. Esto permite que se pueda acceder al objeto del módulo a través de **`PyState_FindModule()`**.

Solo es efectivo en módulos creados con la inicialización monofásica.

Python calls `PyState_AddModule` automatically after importing a module that uses single-phase initialization, so it is unnecessary (but harmless) to call it from module initialization code. An explicit call is needed only if the module's own init code subsequently calls `PyState_FindModule`. The function is mainly intended for implementing alternative import mechanisms (either by calling it directly, or by referring to its implementation for details of the required state updates).

If a module was attached previously using the same *def*, it is replaced by the new *module*.

The caller must have an attached thread state.

Return `-1` with an exception set on error, `0` on success.

Added in version 3.3.

int PyState_RemoveModule(PyModuleDef *def)

Part of the Stable ABI since version 3.3.
Removes the module object created from *def* from the interpreter state. Return `-1` with an exception set on error, `0` on success.

The caller must have an attached thread state.

Added in version 3.3.

Objetos iteradores

Python proporciona dos objetos iteradores de propósito general. El primero, un iterador de secuencia, funciona con una secuencia arbitraria que admite el método `__getitem__()`. El segundo funciona con un objeto invocable y un valor centinela, llamando al invocable para cada elemento en la secuencia y finalizando la iteración cuando se retorna el valor centinela.

PyTypeObject PySeqIter_Type

Part of the Stable ABI.
Objeto tipo para objetos iteradores retornados por `PySeqIter_New()` y la forma de un argumento de la función incorporada `iter()` para los tipos de secuencia incorporados.

int PySeqIter_Check(PyObject *op)

Retorna verdadero si el tipo de *op* es `PySeqIter_Type`. Esta función siempre finaliza con éxito.

PyObject *PySeqIter_New(PyObject *seq)

Return value: New reference. Part of the Stable ABI.
Retorna un iterador que funciona con un objeto de secuencia general, *seq*. La iteración termina cuando la secuencia lanza `IndexError` para la operación de suscripción.

PyTypeObject PyCallIter_Type

Part of the Stable ABI.
Objeto tipo para los objetos iteradores retornados por `PyCallIter_New()` y la forma de dos argumentos de la función incorporada `iter()`.

int PyCallIter_Check(PyObject *op)

Retorna verdadero si el tipo de *op* es `PyCallIter_Type`.

Esta función siempre finaliza con éxito.

PyObject *PyCallIter_New(PyObject *callable, PyObject *sentinel)

Return value: New reference. Part of the Stable ABI.
Retorna un nuevo iterador. El primer parámetro, *callable*, puede ser cualquier objeto invocable de Python que se pueda invocar sin parámetros; cada llamada debe retornar el siguiente elemento en la iteración. Cuando *callable* retorna un valor igual a *sentinel*, la iteración finalizará.

Range Objects

PyTypeObject PyRange_Type

Part of the Stable ABI.
The type object for **`range`** objects.

int PyRange_Check(PyObject *o)

Return true if the object *o* is an instance of a **`range`** object. This function always succeeds.

Builtin Iterator Types

These are built-in iteration types that are included in Python's C API, but provide no additional functions. They are here for completeness.

Python type	C type
`enumerate`	PyTypeObject PyEnum_Type *Part of the Stable ABI.*
`filter`	PyTypeObject PyFilter_Type *Part of the Stable ABI.*
`map`	PyTypeObject PyMap_Type *Part of the Stable ABI.*
`reversed`	PyTypeObject PyReversed_Type *Part of the Stable ABI.*
`zip`	PyTypeObject PyZip_Type

Part of the Stable ABI.

Other Iterator Objects

PyTypeObject PyByteArrayIter_Type

Part of the Stable ABI.

PyTypeObject PyBytesIter_Type

Part of the Stable ABI.

PyTypeObject PyListIter_Type

Part of the Stable ABI.

PyTypeObject PyListRevIter_Type

Part of the Stable ABI.

PyTypeObject PySetIter_Type

Part of the Stable ABI.

PyTypeObject PyTupleIter_Type

Part of the Stable ABI.

PyTypeObject PyRangeIter_Type

Part of the Stable ABI.

PyTypeObject PyLongRangeIter_Type

Part of the Stable ABI.

PyTypeObject PyDictIterKey_Type

Part of the Stable ABI.

PyTypeObject PyDictRevIterKey_Type

Part of the Stable ABI since version 3.8.

PyTypeObject PyDictIterValue_Type

Part of the Stable ABI.

PyTypeObject PyDictRevIterValue_Type

Part of the Stable ABI since version 3.8.

PyTypeObject PyDictIterItem_Type

Part of the Stable ABI.

PyTypeObject PyDictRevIterItem_Type

Part of the Stable ABI since version 3.8.

PyTypeObject PyODictIter_Type

Type objects for iterators of various built-in objects.

Do not create instances of these directly; prefer calling **`PyObject_GetIter()`** instead.

Note that there is no guarantee that a given built-in type uses a given iterator type. For example, iterating over **`range`** will use one of two iterator types depending on the size of the range. Other types may start using a similar scheme in the future, without warning.

Objetos descriptores

Los «descriptores» son objetos que describen algún atributo de un objeto. Se encuentran en el diccionario de objetos tipo.

PyTypeObject PyProperty_Type

Part of the Stable ABI.
El objeto de tipo para los tipos de descriptor incorporado.

PyObject *PyDescr_NewGetSet(PyTypeObject *type, struct PyGetSetDef *getset)

Return value: New reference. Part of the Stable ABI.

PyObject *PyDescr_NewMember(PyTypeObject *type, struct PyMemberDef *meth)

Return value: New reference. Part of the Stable ABI.

PyTypeObject PyMemberDescr_Type

Part of the Stable ABI.
The type object for member descriptor objects created from **`PyMemberDef`** structures. These descriptors expose fields of a C struct as attributes on a type, and correspond to **`types.MemberDescriptorType`** objects in Python.

PyTypeObject PyGetSetDescr_Type

Part of the Stable ABI.
The type object for get/set descriptor objects created from **`PyGetSetDef`** structures. These descriptors implement attributes whose value is computed by C getter and setter functions, and are used for many built-in type attributes.

PyObject *PyDescr_NewMethod(PyTypeObject *type, struct PyMethodDef *meth)

Return value: New reference. Part of the Stable ABI.

PyTypeObject PyMethodDescr_Type

Part of the Stable ABI.
The type object for method descriptor objects created from `PyMethodDef` structures. These descriptors expose C functions as methods on a type, and correspond to `types.MemberDescriptorType` objects in Python.

PyObject *PyDescr_NewWrapper(PyTypeObject *type, struct wrapperbase *wrapper, void *wrapped)

Return value: New reference.

PyTypeObject PyWrapperDescr_Type

Part of the Stable ABI.
The type object for wrapper descriptor objects created by `PyDescr_NewWrapper()` and `PyWrapper_New()`. Wrapper descriptors are used internally to expose special methods implemented via wrapper structures, and appear in Python as `types.WrapperDescriptorType` objects.

PyObject *PyDescr_NewClassMethod(PyTypeObject *type, PyMethodDef *method)

Return value: New reference. Part of the Stable ABI.

int PyDescr_IsData(PyObject *descr)

Return non-zero if the descriptor object *descr* describes a data attribute, or `0` if it describes a method. *descr* must be a descriptor object; there is no error checking.

PyObject *PyWrapper_New(PyObject*, PyObject*)

Return value: New reference. Part of the Stable ABI.

Built-in descriptors

PyTypeObject PySuper_Type

Part of the Stable ABI.
The type object for super objects. This is the same object as **`super`** in the Python layer.

PyTypeObject PyClassMethod_Type

The type of class method objects. This is the same object as **`classmethod`** in the Python layer.

PyTypeObject PyClassMethodDescr_Type

Part of the Stable ABI.
The type object for C-level class method descriptor objects. This is the type of the descriptors created for **`classmethod()`** defined in C extension types, and is the same object as **`classmethod`** in Python.

PyObject *PyClassMethod_New(PyObject *callable)

Create a new **`classmethod`** object wrapping *callable*. *callable* must be a callable object and must not be `NULL`.

On success, this function returns a strong reference to a new class method descriptor. On failure, this function returns `NULL` with an exception set.

PyTypeObject PyStaticMethod_Type

The type of static method objects. This is the same object as **`staticmethod`** in the Python layer.

PyObject *PyStaticMethod_New(PyObject *callable)

Create a new **`staticmethod`** object wrapping *callable*. *callable* must be a callable object and must not be `NULL`.

On success, this function returns a strong reference to a new static method descriptor. On failure, this function returns `NULL` with an exception set.

Objetos rebanada (*slice*)

PyTypeObject PySlice_Type

Part of the Stable ABI.

El objeto tipo para objetos rebanadas. Esto es lo mismo que **`slice`** en la capa de Python.

int PySlice_Check(PyObject *ob)

Retorna verdadero si *ob* es un objeto rebanada; *ob* no debe ser `NULL`. Esta función siempre funciona correctamente.

PyObject *PySlice_New(PyObject *start, PyObject *stop, PyObject *step)

Return value: New reference. Part of the Stable ABI.

Retorna un nuevo objeto rebanada con los valores dados. Los parámetros *start*, *stop* y *step* se utilizan como los valores de los atributos del objeto rebanada de los mismos nombres. Cualquiera de los valores puede ser `NULL`, en cuyo caso se usará `None` para el atributo correspondiente.

Retorna `NULL` con una excepción establecida si no se ha podido asignar el nuevo objeto.

int PySlice_GetIndices(PyObject *slice, Py_ssize_t length, Py_ssize_t *start, Py_ssize_t *stop, Py_ssize_t *step)

Part of the Stable ABI.

Recupera los índices *start*, *stop* y *step* del objeto rebanada *slice*, suponiendo una secuencia de longitud *length*. Trata los índices mayores que *length* como errores.

Retorna `0` en caso de éxito y `-1` en caso de error sin una excepción establecida (a menos que uno de los índices no sea `None` y no se haya convertido a un entero, en cuyo caso `- 1` se retorna con una excepción establecida).

Probablemente no quiera usar esta función.

Distinto en la versión 3.2: El tipo de parámetro para el parámetro *slice* era `PySliceObject*` antes.

int PySlice_GetIndicesEx(PyObject *slice, Py_ssize_t length, Py_ssize_t *start, Py_ssize_t *stop, Py_ssize_t *step, Py_ssize_t *slicelength)

Part of the Stable ABI.

Reemplazo utilizable para **`PySlice_GetIndices()`**. Recupera los índices de *start*, *stop*, y *step* del objeto rebanada *slice* asumiendo una secuencia de longitud *length*, y almacena la longitud de la rebanada en *slicelength*. Los índices fuera de los límites se recortan de manera coherente con el manejo de sectores normales.

Retorna `0` en caso de éxito y `-1` en caso de error con una excepción establecida.

Nota

Esta función se considera no segura para secuencias redimensionables. Su invocación debe ser reemplazada por una combinación de **`PySlice_Unpack()`** y **`PySlice_AdjustIndices()`** donde:

```
if (PySlice_GetIndicesEx(slice, length, &start, &s
    // retorna un error
}
```

es reemplazado por:

```
if (PySlice_Unpack(slice, &start, &stop, &step) <
    // retorna un error
}
slicelength = PySlice_AdjustIndices(length, &start
```

Distinto en la versión 3.2: El tipo de parámetro para el parámetro *slice* era `PySliceObject*` antes.

Distinto en la versión 3.6.1: Si `Py_LIMITED_API` no se establece o se establece el valor entre `0x03050400` y `0x03060000` (sin incluir) o `0x03060100` o un superior **`PySlice_GetIndicesEx()`** se implementa como un macro usando **`PySlice_Unpack()`** y **`PySlice_AdjustIndices()`**. Los argumentos *start*, *stop* y *step* se evalúan más de una vez.

Obsoleto desde la versión 3.6.1: Si `Py_LIMITED_API` se establece en un valor menor que `0x03050400` o entre `0x03060000` y `0x03060100` (sin incluir) **`PySlice_GetIndicesEx()`** es una función obsoleta.

int PySlice_Unpack(PyObject *slice, Py_ssize_t *start, Py_ssize_t *stop, Py_ssize_t *step)

Part of the Stable ABI since version 3.7.
Extrae los miembros de datos *start*, *stop*, y *step* de un objeto rebanada como enteros en C. Reduce silenciosamente los valores mayores que `PY_SSIZE_T_MAX` a `PY_SSIZE_T_MAX`, aumenta silenciosamente los valores *start* y *stop* inferiores a `PY_SSIZE_T_MIN` a `PY_SSIZE_T_MIN`, y silenciosamente aumenta los valores de *step* a menos de `-PY_SSE` a `-PY_SSIZE_T_MAX`.

Retorna `-1` con una excepción establecida en caso de error, `0` en caso de éxito.

Added in version 3.6.1.

Py_ssize_t PySlice_AdjustIndices(Py_ssize_t length, Py_ssize_t *start, Py_ssize_t *stop, Py_ssize_t step)

Part of the Stable ABI since version 3.7.
Ajusta los índices de corte de inicio/fin asumiendo una secuencia de la longitud especificada. Los índices fuera de los límites se recortan de manera coherente con el manejo de sectores normales.

Retorna la longitud de la rebanada. Siempre exitoso. No llama al código de Python.

Added in version 3.6.1.

Objeto elipsis

PyTypeObject PyEllipsis_Type

Part of the Stable ABI.
The type of Python **`Ellipsis`** object. Same as **`types.EllipsisType`** in the Python layer.

PyObject *Py_Ellipsis

El objeto `Ellipsis` de Python. Este objeto no tiene métodos. Al igual que **`Py_None`**, es un objeto singleton immortal.

Distinto en la versión 3.12: **`Py_Ellipsis`** es inmortal.

Objetos de vista de memoria (*MemoryView*)

Un objeto `memoryview` expone la interfaz de búfer a nivel de C como un objeto Python que luego puede pasarse como cualquier otro objeto.

PyTypeObject PyMemoryView_Type

Part of the Stable ABI.
This instance of `PyTypeObject` represents the Python memoryview type. This is the same object as `memoryview` in the Python layer.

PyObject *PyMemoryView_FromObject(PyObject *obj)

Return value: New reference. Part of the Stable ABI.
Crea un objeto de vista de memoria *memoryview* a partir de un objeto que proporciona la interfaz del búfer. Si *obj* admite exportaciones de búfer de escritura, el objeto de vista de memoria será de lectura/escritura, de lo contrario puede ser de solo lectura o de lectura/escritura a discreción del exportador.

PyBUF_READ

Part of the Stable ABI since version 3.11.
Indicador para solicitar un búfer de solo lectura.

PyBUF_WRITE

Part of the Stable ABI since version 3.11.
Indicador para solicitar un búfer escribible.

PyObject *PyMemoryView_FromMemory(char *mem, Py_ssize_t size, int flags)

Return value: New reference. Part of the Stable ABI since version

3.7.
Crea un objeto de vista de memoria usando *mem* como el búfer subyacente. *flags* pueden ser uno de `PyBUF_READ` o `PyBUF_WRITE`.

Added in version 3.3.

PyObject *PyMemoryView_FromBuffer(const Py_buffer *view)

Return value: New reference. Part of the Stable ABI since version 3.11.
Crea un objeto de vista de memoria que ajuste la estructura de búfer dada *view*. Para memorias intermedias de bytes simples, `PyMemoryView_FromMemory()` es la función preferida.

PyObject *PyMemoryView_GetContiguous(PyObject *obj, int buffertype, char order)

Return value: New reference. Part of the Stable ABI.
Crea un objeto de vista de memoria *memoryview* para un fragmento de memoria contiguo (contiguous, en *order* "C" o "F" de Fortran) desde un objeto que define la interfaz del búfer. Si la memoria es contigua, el objeto de vista de memoria apunta a la memoria original. De lo contrario, se realiza una copia y la vista de memoria apunta a un nuevo objeto de bytes.

buffertype puede ser uno de `PyBUF_READ` o `PyBUF_WRITE`.

int PyMemoryView_Check(PyObject *obj)

Retorna verdadero si el objeto *obj* es un objeto de vista de memoria. Actualmente no está permitido crear subclases de `memoryview`. Esta función siempre finaliza con éxito.

Py_buffer *PyMemoryView_GET_BUFFER(PyObject *mview)

Retorna un puntero a la copia privada de la vista de memoria del búfer del exportador. *mview* **debe** ser una instancia de *memoryview*; este macro no verifica su tipo, debe hacerlo usted mismo o correrá el riesgo de fallas.

PyObject *PyMemoryView_GET_BASE(PyObject *mview)

Retorna un puntero al objeto de exportación en el que se basa la vista de memoria o `NULL` si la vista de memoria ha sido creada por una de las funciones **`PyMemoryView_FromMemory()`** o **`PyMemoryView_FromBuffer()`**. *mview* **debe** ser una instancia de *memoryview*.

Pickle buffer objects

Added in version 3.8.

A **`pickle.PickleBuffer`** object wraps a buffer-providing object for out-of-band data transfer with the **`pickle`** module.

PyTypeObject PyPickleBuffer_Type

> This instance of **`PyTypeObject`** represents the Python pickle buffer type. This is the same object as **`pickle.PickleBuffer`** in the Python layer.

int PyPickleBuffer_Check(PyObject *op)

> Return true if *op* is a pickle buffer instance. This function always succeeds.

PyObject *PyPickleBuffer_FromObject(PyObject *obj)

> Create a pickle buffer from the object *obj*.
>
> This function will fail if *obj* doesn't support the buffer protocol.
>
> On success, return a new pickle buffer instance. On failure, set an exception and return `NULL`.
>
> Analogous to calling **`pickle.PickleBuffer`** with *obj* in Python.

const Py_buffer *PyPickleBuffer_GetBuffer(PyObject *picklebuf)

> Get a pointer to the underlying **`Py_buffer`** that the pickle buffer wraps.
>
> The returned pointer is valid as long as *picklebuf* is alive and has not been released. The caller must not modify or free the returned **`Py_buffer`**. If the pickle buffer has been released,

raise `ValueError`.

On success, return a pointer to the buffer view. On failure, set an exception and return `NULL`.

int PyPickleBuffer_Release(PyObject *picklebuf)

Release the underlying buffer held by the pickle buffer.

Return `0` on success. On failure, set an exception and return `-1`.

Analogous to calling `pickle.PickleBuffer.release()` in Python.

Objetos de referencia débil

Python soporta *referencias débiles* como objetos de primera clase. Hay dos tipos de objetos específicos que implementan directamente referencias débiles. El primero es un objeto con referencia simple, y el segundo actúa como un proxy del objeto original tanto como pueda.

int PyWeakref_Check(PyObject *ob)

> Retorna un valor distinto de cero si *ob* es una referencia o un objeto proxy. Esta función siempre finaliza con éxito.

int PyWeakref_CheckRef(PyObject *ob)

> Return non-zero if *ob* is a reference object or a subclass of the reference type. This function always succeeds.

int PyWeakref_CheckRefExact(PyObject *ob)

> Return non-zero if *ob* is a reference object, but not a subclass of the reference type. This function always succeeds.

int PyWeakref_CheckProxy(PyObject *ob)

> Retorna un valor distinto de cero si *ob* es un objeto proxy. Esta función siempre finaliza con éxito.

PyObject *PyWeakref_NewRef(PyObject *ob, PyObject *callback)

> *Return value: New reference. Part of the Stable ABI.*
> Retorna un objeto de referencia débil para el objeto *ob*. Esto siempre retornará una nueva referencia, pero no garantiza la creación de un objeto nuevo; un objeto de referencia ya existente puede ser retornado. El segundo parámetro, *callback*, puede ser un objeto invocable que recibe una notificación cuando *ob* es recolectado como basura; debe aceptar un solo parámetro, el cual será el mismo objeto de

referencia débil. *callback* también puede ser `None` o `NULL`. Si *ob* no es un objeto que puede ser referido de forma débil, o si *callback* no es invocable, `None`, o `NULL`, esto retornará `NULL` y lanzará un **`TypeError`**.

Ver también

`PyType_SUPPORTS_WEAKREFS()` for checking if *ob* is weakly referenceable.

PyObject *PyWeakref_NewProxy(PyObject *ob, PyObject *callback)

Return value: New reference. Part of the Stable ABI.
Retorna un objeto proxy de referencia débil para el objeto *ob*. Esto siempre retornará una nueva referencia, pero no garantiza la creación de un objeto nuevo; un objeto proxy de referencia ya existente puede ser retornado. El segundo parámetro, *callback*, puede ser un objeto invocable que recibe una notificación cuando *ob* es recolectado como basura; debe aceptar un solo parámetro, el cual será el mismo objeto de referencia débil. *callback* también puede ser `None` o `NULL`. Si *ob* no es un objeto que puede ser referido de forma débil, o si *callback* no es invocable, `None`, o `NULL`, esto retornará `NULL` y lanzará un **`TypeError`**.

Ver también

`PyType_SUPPORTS_WEAKREFS()` for checking if *ob* is weakly referenceable.

int PyWeakref_GetRef(PyObject *ref, PyObject **pobj)

Part of the Stable ABI since version 3.13.
Obtiene un strong reference al objeto referenciado desde una referencia débil, *ref*, en **pobj*.

- En caso de éxito, asigna **pobj* en un nuevo strong reference al objeto referenciado y retorna 1.
- Si la referencia está inactiva, asigna **pobj* a `NULL` y

retorna 0.
- En caso de error, lanza una excepción y devuelve -1.

Added in version 3.13.

PyObject *PyWeakref_GetObject(PyObject *ref)

Return value: Borrowed reference. Part of the Stable ABI.
Retorna un borrowed reference del objeto referenciado desde una referencia débil, *ref*. Si el referente ya no está activo, retorna `Py_None`.

Nota

Esta función retorna una referencia borrowed reference al objeto referenciado. Esto significa que siempre debe llamar a **`Py_INCREF()`** sobre el objeto excepto cuando no pueda ser destruido antes del último uso de la referencia prestada.

Deprecated since version 3.13, will be removed in version 3.15: Utiliza **`PyWeakref_GetRef()`** en su lugar.

PyObject *PyWeakref_GET_OBJECT(PyObject *ref)

Return value: Borrowed reference.
Similar a **`PyWeakref_GetObject()`**, pero no realiza ninguna comprobación de errores.

Deprecated since version 3.13, will be removed in version 3.15: Utiliza **`PyWeakref_GetRef()`** en su lugar.

int PyWeakref_IsDead(PyObject *ref)

Test if the weak reference *ref* is dead. Returns 1 if the reference is dead, 0 if it is alive, and -1 with an error set if *ref* is not a weak reference object.

Added in version 3.14.

void PyObject_ClearWeakRefs(PyObject *object)

Part of the Stable ABI.
Esta función es invocada por el gestor **`tp_dealloc`** para limpiar referencias débiles.

Esto recorre las referencias débiles de *object* e invoca retrollamadas para aquellas referencias que tengan una. Retorna cuando se han intentado todas las retrollamadas.

void PyUnstable_Object_ClearWeakRefsNoCallbacks(PyObject *object)

> *This is Unstable API. It may change without warning in minor releases.*

Borra las referencias débiles para *object* sin llamar a las retrollamadas.

Esta función es invocada por el gestor **`tp_dealloc`** para tipos con finalizadores (i.e., **`__del__()`**). El gestor de esos objetos primero llama a **`PyObject_ClearWeakRefs()`** para borrar las referencias débiles y llamar a sus callbacks, luego al finalizador, y finalmente a esta función para borrar cualquier referencia débil que pueda haber sido creada por el finalizador.

En la mayoría de los casos, es más apropiado utilizar **`PyObject_ClearWeakRefs()`** para borrar las referencias débiles en lugar de esta función.

Added in version 3.13.

Cápsulas

Consulta Proporcionar una API C para un módulo de extensión para obtener más información sobre el uso de estos objetos.

Added in version 3.1.

type PyCapsule

This subtype of `PyObject` represents an opaque value, useful for C extension modules which need to pass an opaque value (as a void* pointer) through Python code to other C code. It is often used to make a C function pointer defined in one module available to other modules, so the regular import mechanism can be used to access C APIs defined in dynamically loaded modules.

PyTypeObject PyCapsule_Type

Part of the Stable ABI.
The type object corresponding to capsule objects. This is the same object as `types.CapsuleType` in the Python layer.

type PyCapsule_Destructor

Part of the Stable ABI.
El tipo de devolución de llamada de un destructor para una cápsula. Definido como:

```
typedef void (*PyCapsule_Destructor)(PyObject *);
```

Consulte `PyCapsule_New()` para conocer la semántica de las devoluciones de llamada de *PyCapsule_Destructor*.

int PyCapsule_CheckExact(PyObject *p)

Thread safety: Atomic.
Retorna verdadero si su argumento es a `PyCapsule`. Esta función siempre finaliza con éxito.

PyObject *PyCapsule_New(void *pointer, const char *name, PyCapsule_Destructor destructor)

Return value: New reference. Part of the Stable ABI. Thread safety: Atomic.
Crea un **`PyCapsule`** encapsulando el *pointer*. El argumento *pointer* puede no ser `NULL`.

En caso de falla, establece una excepción y retorna `NULL`.

La cadena de caracteres *name* puede ser `NULL` o un puntero a una cadena C válida. Si no es `NULL`, esta cadena de caracteres debe sobrevivir a la cápsula. (Aunque está permitido liberarlo dentro del *destructor*).

Si el argumento *destructor* no es `NULL`, se llamará con la cápsula como argumento cuando se destruya.

Si esta cápsula se almacenará como un atributo de un módulo, el nombre *name* debe especificarse como `modulename.attributename`. Esto permitirá que otros módulos importen la cápsula usando **`PyCapsule_Import()`**.

void *PyCapsule_GetPointer(PyObject *capsule, const char *name)

Part of the Stable ABI. Thread safety: Safe to call without external synchronization on distinct objects.
Recupera el *pointer* almacenado en la cápsula. En caso de falla, establece una excepción y retorna `NULL`.

El parámetro *name* debe coincidir exactamente con el nombre almacenado en la cápsula. Si el nombre almacenado en la cápsula es `NULL`, el *name* también debe ser `NULL`. Python utiliza la función C **`strcmp()`** para comparar los nombres de las cápsulas.

PyCapsule_Destructor PyCapsule_GetDestructor(PyObject *capsule)

Part of the Stable ABI. Thread safety: Safe to call without external synchronization on distinct objects.
Retorna el destructor actual almacenado en la cápsula. En

caso de falla, establece una excepción y retorna `NULL`.

Es legal que una cápsula tenga un destructor `NULL`. Esto hace que un código de retorno `NULL` sea algo ambiguo; use **`PyCapsule_IsValid()`** o **`PyErr_Occurred()`** para desambiguar.

void *PyCapsule_GetContext(PyObject *capsule)

Part of the Stable ABI. Thread safety: Safe to call without external synchronization on distinct objects.
Retorna el contexto actual almacenado en la cápsula. En caso de falla, establece una excepción y retorna `NULL`.

Es legal que una cápsula tenga un contexto `NULL`. Esto hace que un código de retorno `NULL` sea algo ambiguo; use **`PyCapsule_IsValid()`** o **`PyErr_Occurred()`** para desambiguar.

const char *PyCapsule_GetName(PyObject *capsule)

Part of the Stable ABI. Thread safety: Safe to call without external synchronization on distinct objects.
Retorna el nombre actual almacenado en la cápsula. En caso de falla, establece una excepción y retorna `NULL`.

Es legal que una cápsula tenga un nombre `NULL`. Esto hace que un código de retorno `NULL` sea algo ambiguo; use **`PyCapsule_IsValid()`** o **`PyErr_Occurred()`** para desambiguar.

void *PyCapsule_Import(const char *name, int no_block)

Part of the Stable ABI. Thread safety: Safe to call from multiple threads with external synchronization only.
Importa un puntero a un objeto C desde un atributo cápsula en un módulo. El parámetro *name* debe especificar el nombre completo del atributo, como en `module.attribute`. El nombre *name* almacenado en la cápsula debe coincidir exactamente con esta cadena de caracteres.

This function splits *name* on the `.` character, and imports the first element. It then processes further elements using attribute lookups.

Retorna el puntero *pointer* interno de la cápsula en caso de éxito. En caso de falla, establece una excepción y retorna `NULL`.

Nota

If *name* points to an attribute of some submodule or subpackage, this submodule or subpackage must be previously imported using other means (for example, by using **`PyImport_ImportModule()`**) for the attribute lookups to succeed.

Distinto en la versión 3.3: no_block ya no tiene efecto.

int PyCapsule_IsValid(PyObject *capsule, const char *name)

Part of the Stable ABI. Thread safety: Safe to call without external synchronization on distinct objects.
Determina si *capsule* es o no una cápsula válida. Una cápsula válida no es `NULL`, pasa **`PyCapsule_CheckExact()`**, tiene un puntero no `NULL` almacenado y su nombre interno coincide con el parámetro *name*. (Consulte **`PyCapsule_GetPointer()`** para obtener información sobre cómo se comparan los nombres de las cápsulas).

En otras palabras, si **`PyCapsule_IsValid()`** retorna un valor verdadero, se garantiza que las llamadas a cualquiera de las funciones de acceso (cualquier función que comience con `PyCapsule_Get`) tendrán éxito.

Retorna un valor distinto de cero si el objeto es válido y coincide con el nombre pasado. Retorna `0` de lo contrario. Esta función no fallará.

int PyCapsule_SetContext(PyObject *capsule, void *context)

Part of the Stable ABI. Thread safety: Safe to call without external synchronization on distinct objects.

Establece el puntero de contexto dentro de *capsule* a *context*.

Retorna `0` en caso de éxito. Retorna distinto de cero y establece una excepción en caso de error.

int PyCapsule_SetDestructor(PyObject *capsule, PyCapsule_Destructor destructor)

Part of the Stable ABI. Thread safety: Safe to call without external synchronization on distinct objects.
Establece el destructor dentro de *capsule* en *destructor*.

Retorna `0` en caso de éxito. Retorna distinto de cero y establece una excepción en caso de error.

int PyCapsule_SetName(PyObject *capsule, const char *name)

Part of the Stable ABI. Thread safety: Safe to call without external synchronization on distinct objects.
Establece el nombre dentro de *capsule* a *name*. Si no es `NULL`, el nombre debe sobrevivir a la cápsula. Si el *name* anterior almacenado en la cápsula no era `NULL`, no se intenta liberarlo.

Retorna `0` en caso de éxito. Retorna distinto de cero y establece una excepción en caso de error.

int PyCapsule_SetPointer(PyObject *capsule, void *pointer)

Part of the Stable ABI. Thread safety: Safe to call without external synchronization on distinct objects.
Establece el puntero vacío dentro de *capsule* a *pointer*. El puntero puede no ser `NULL`.

Retorna `0` en caso de éxito. Retorna distinto de cero y establece una excepción en caso de error.

Frame objects

type PyFrameObject

Part of the Limited API (as an opaque struct).
La estructura C de los objetos utilizados para describir los objetos del frame.

No hay miembros públicos en esta estructura.

Distinto en la versión 3.11: Los miembros de esta estructura se han eliminado de la API pública de C. Consulte la entrada Novedades para más detalles.

Las funciones **`PyEval_GetFrame()`** y **`PyThreadState_GetFrame()`** pueden utilizarse para obtener un objeto frame.

Véase también Reflexión.

PyTypeObject PyFrame_Type

The type of frame objects. It is the same object as **`types.FrameType`** in the Python layer.

Distinto en la versión 3.11: Previously, this type was only available after including `<frameobject.h>`.

PyFrameObject *PyFrame_New(PyThreadState *tstate, PyCodeObject *code, PyObject *globals, PyObject *locals)

Create a new frame object. This function returns a strong reference to the new frame object on success, and returns `NULL` with an exception set on failure.

int PyFrame_Check(PyObject *obj)

Return non-zero if *obj* is a frame object.

Distinto en la versión 3.11: Previously, this function was only available after including `<frameobject.h>`.

PyFrameObject *PyFrame_GetBack(PyFrameObject *frame)

Return value: New reference.
Obtiene el *frame* exterior siguiente.

Return a strong reference, or `NULL` if *frame* has no outer frame. This raises no exceptions.

Added in version 3.9.

PyObject *PyFrame_GetBuiltins(PyFrameObject *frame)

Return value: New reference.
Get the *frame*'s **`f_builtins`** attribute.

Retorna una strong reference, o `NULL` si *frame* no tiene frame exterior.

Added in version 3.11.

PyCodeObject *PyFrame_GetCode(PyFrameObject *frame)

Return value: New reference. Part of the Stable ABI since version 3.10.
Obtenga el código *frame*.

Retorna un strong reference.

El resultado (frame code) no puede ser `NULL`.

Added in version 3.9.

PyObject *PyFrame_GetGenerator(PyFrameObject *frame)

Return value: New reference.
Obtiene el generador, rutina o generador asíncrono al que pertenece este frame, o `NULL` si este frame no es propiedad de un generador. No lanza una excepción, incluso si el valor de retorno es `NULL`.

Retorna un strong reference, o `NULL`.

Added in version 3.11.

PyObject *PyFrame_GetGlobals(PyFrameObject *frame)

Return value: New reference.
Get the *frame*'s **`f_globals`** attribute.

Retorna una strong reference, o `NULL` si *frame* no tiene frame exterior.

Added in version 3.11.

int PyFrame_GetLasti(PyFrameObject *frame)

Get the *frame*'s **`f_lasti`** attribute.

Retorna -1 si `frame.f_lasti` es `None`.

Added in version 3.11.

PyObject *PyFrame_GetVar(PyFrameObject *frame, PyObject *name)

Return value: New reference.
Get the variable *name* of *frame*.

- Return a strong reference to the variable value on success.
- Raise **`NameError`** and return `NULL` if the variable does not exist.
- Raise an exception and return `NULL` on error.

name type must be a **`str`**.

Added in version 3.12.

PyObject *PyFrame_GetVarString(PyFrameObject *frame, const char *name)

Return value: New reference.
Similar to **`PyFrame_GetVar()`**, but the variable name is a C

string encoded in UTF-8.

Added in version 3.12.

PyObject *PyFrame_GetLocals(PyFrameObject *frame)

Return value: New reference.
Get the *frame*'s **`f_locals`** attribute. If the frame refers to an optimized scope, this returns a write-through proxy object that allows modifying the locals. In all other cases (classes, modules, **`exec()`**, **`eval()`**) it returns the mapping representing the frame locals directly (as described for **`locals()`**).

Retorna un strong reference.

Added in version 3.11.

Distinto en la versión 3.13: As part of **PEP 667** [https://peps.python.org/pep-0667/], return an instance of **`PyFrameLocalsProxy_Type`**.

int PyFrame_GetLineNumber(PyFrameObject *frame)

Part of the Stable ABI since version 3.10.
Retorna el número de línea en la que se está ejecutando el *frame*.

Frame locals proxies

Added in version 3.13.

The **`f_locals`** attribute on a frame object is an instance of a «frame-locals proxy». The proxy object exposes a write-through view of the underlying locals dictionary for the frame. This ensures that the variables exposed by `f_locals` are always up to date with the live local variables in the frame itself.

See **PEP 667** [https://peps.python.org/pep-0667/] for more information.

PyTypeObject PyFrameLocalsProxy_Type

The type of frame `locals()` proxy objects.

int PyFrameLocalsProxy_Check(PyObject *obj)

Return non-zero if *obj* is a frame `locals()` proxy.

Legacy local variable APIs

These APIs are soft deprecated. As of Python 3.13, they do nothing. They exist solely for backwards compatibility.

void PyFrame_LocalsToFast(PyFrameObject *f, int clear)

Prior to Python 3.13, this function would copy the `f_locals` attribute of *f* to the internal «fast» array of local variables, allowing changes in frame objects to be visible to the interpreter. If *clear* was true, this function would process variables that were unset in the locals dictionary.

Soft deprecated since version 3.13: This function now does nothing.

void PyFrame_FastToLocals(PyFrameObject *f)

Prior to Python 3.13, this function would copy the internal «fast» array of local variables (which is used by the interpreter) to the `f_locals` attribute of *f*, allowing changes in local variables to be visible to frame objects.

Soft deprecated since version 3.13: This function now does nothing.

int PyFrame_FastToLocalsWithError(PyFrameObject *f)

Prior to Python 3.13, this function was similar to `PyFrame_FastToLocals()`, but would return `0` on success, and `-1` with an exception set on failure.

Soft deprecated since version 3.13: This function now does nothing.

Ver también

PEP 667 [https://peps.python.org/pep-0667/]

Internal frames

Unless using **PEP 523** [https://peps.python.org/pep-0523/], you will not need this.

struct _PyInterpreterFrame

The interpreter's internal frame representation.

Added in version 3.11.

PyObject *PyUnstable_InterpreterFrame_GetCode(struct _PyInterpreterFrame *frame);

This is Unstable API. It may change without warning in minor releases.

Return a strong reference to the code object for the frame.

Added in version 3.12.

int PyUnstable_InterpreterFrame_GetLasti(struct _PyInterpreterFrame *frame);

This is Unstable API. It may change without warning in minor releases.

Return the byte offset into the last executed instruction.

Added in version 3.12.

int PyUnstable_InterpreterFrame_GetLine(struct

_PyInterpreterFrame *frame);

> *This is Unstable API. It may change without warning in minor releases.*

Return the currently executing line number, or -1 if there is no line number.

Added in version 3.12.

Objetos generadores

Los objetos generadores son lo que Python usa para implementar iteradores generadores. Normalmente se crean iterando sobre una función que produce valores, en lugar de llamar explícitamente **`PyGen_New()`** o **`PyGen_NewWithQualName()`**.

type PyGenObject

La estructura en C utilizada para los objetos generadores.

PyTypeObject PyGen_Type

El objeto tipo correspondiente a los objetos generadores.

int PyGen_Check(PyObject *ob)

Retorna verdadero si *ob* es un objeto generador; *ob* no debe ser `NULL`. Esta función siempre finaliza con éxito.

int PyGen_CheckExact(PyObject *ob)

Retorna verdadero si el tipo de *ob* es **`PyGen_Type`**; *ob* no debe ser `NULL`. Esta función siempre finaliza con éxito.

PyObject *PyGen_New(PyFrameObject *frame)

Return value: New reference.
Crea y retorna un nuevo objeto generador basado en el objeto *frame*. Una referencia a *frame* es robada por esta función. El argumento no debe ser `NULL`.

PyObject *PyGen_NewWithQualName(PyFrameObject *frame, PyObject *name, PyObject *qualname)

Return value: New reference.
Crea y retorna un nuevo objeto generador basado en el objeto *frame*, con `__name__` y `__qualname__` establecido en *name* y *qualname*. Una referencia a *frame* es robada por esta

función. El argumento *frame* no debe ser `NULL`.

PyCodeObject *PyGen_GetCode(PyGenObject *gen)

Return a new strong reference to the code object wrapped by *gen*. This function always succeeds.

Asynchronous Generator Objects

Ver también

PEP 525 [https://peps.python.org/pep-0525/]

PyTypeObject PyAsyncGen_Type

The type object corresponding to asynchronous generator objects. This is available as **`types.AsyncGeneratorType`** in the Python layer.

Added in version 3.6.

PyObject *PyAsyncGen_New(PyFrameObject *frame, PyObject *name, PyObject *qualname)

Create a new asynchronous generator wrapping *frame*, with `__name__` and `__qualname__` set to *name* and *qualname*. *frame* is stolen by this function and must not be `NULL`.

On success, this function returns a strong reference to the new asynchronous generator. On failure, this function returns `NULL` with an exception set.

Added in version 3.6.

int PyAsyncGen_CheckExact(PyObject *op)

Return true if *op* is an asynchronous generator object, false otherwise. This function always succeeds.

Added in version 3.6.

Deprecated API

PyAsyncGenASend_CheckExact(op)

This is a soft deprecated API that was included in Python's C API by mistake.

It is solely here for completeness; do not use this API.

Objetos corrutina

Added in version 3.5.

Los objetos de corrutina son las funciones declaradas con un retorno de palabra clave `async`.

type PyCoroObject

La estructura en C utilizada para objeto corrutina.

PyTypeObject PyCoro_Type

El tipo de objeto correspondiente a los objetos corrutina.

int PyCoro_CheckExact(PyObject *ob)

Retorna verdadero si el tipo de *ob* es **`PyCoro_Type`**; *ob* no debe ser `NULL`. Esta función siempre finaliza con éxito.

PyObject *PyCoro_New(PyFrameObject *frame, PyObject *name, PyObject *qualname)

Return value: New reference.
Crea y retorna un nuevo objeto corrutina basado en el objeto *frame*, con `__name__` y `__qualname__` establecido en *name* y *qualname*. Una referencia a *frame* es robada por esta función. El argumento *frame* no debe ser `NULL`.

Objetos de variables de contexto

Added in version 3.7.

Distinto en la versión 3.7.1:

Nota

En Python 3.7.1, las firmas de todas las variables de contexto C APIs fueron **cambiadas** para usar punteros `PyObject` en lugar de `PyContext`, `PyContextVar`, y `PyContextToken`, por ejemplo:

```
// en 3.7.0:
PyContext *PyContext_New(void);

// en 3.7.1+:
PyObject *PyContext_New(void);
```

Ver bpo-34762 [https://bugs.python.org/issue?@action=redirect&bpo=34762] para más detalles.

Esta sección detalla la API pública de C para el módulo `contextvars`.

type PyContext

La estructura C utilizada para representar un objeto `contextvars.Context`.

type PyContextVar

La estructura C utilizada para representar un objeto `contextvars.ContextVar`.

type PyContextToken

La estructura C solía representar un objeto **`contextvars.Token`**.

PyTypeObject PyContext_Type

El objeto de tipo que representa el tipo *context*.

PyTypeObject PyContextVar_Type

El objeto tipo que representa el tipo *variable de contexto*.

PyTypeObject PyContextToken_Type

El tipo objeto que representa el tipo *token de variable de contexto*.

Macros de verificación de tipo:

int PyContext_CheckExact(PyObject *o)

Retorna verdadero si *o* es de tipo **`PyContext_Type`**. *o* no debe ser `NULL`. Esta función siempre finaliza con éxito.

int PyContextVar_CheckExact(PyObject *o)

Retorna verdadero si *o* es de tipo **`PyContextVar_Type`**. *o* no debe ser `NULL`. Esta función siempre finaliza con éxito.

int PyContextToken_CheckExact(PyObject *o)

Retorna verdadero si *o* es de tipo **`PyContextToken_Type`**. *o* no debe ser `NULL`. Esta función siempre finaliza con éxito.

Funciones de gestión de objetos de contexto:

PyObject *PyContext_New(void)

Return value: New reference.
Crea un nuevo objeto de contexto vacío. Retorna `NULL` si se ha producido un error.

PyObject *PyContext_Copy(PyObject *ctx)

Return value: New reference.
Crea una copia superficial del objeto de contexto *ctx* pasado. Retorna `NULL` si se ha producido un error.

PyObject *PyContext_CopyCurrent(void)

Return value: New reference.
Crea una copia superficial del contexto actual del hilo. Retorna `NULL` si se ha producido un error.

int PyContext_Enter(PyObject *ctx)

Establece *ctx* como el contexto actual para el hilo actual. Retorna `0` en caso de éxito y `-1` en caso de error.

int PyContext_Exit(PyObject *ctx)

Desactiva el contexto *ctx* y restaura el contexto anterior como el contexto actual para el hilo actual. Retorna `0` en caso de éxito y `-1` en caso de error.

int PyContext_AddWatcher(PyContext_WatchCallback callback)

Register *callback* as a context object watcher for the current interpreter. Return an ID which may be passed to **`PyContext_ClearWatcher()`**. In case of error (e.g. no more watcher IDs available), return `-1` and set an exception.

Added in version 3.14.

int PyContext_ClearWatcher(int watcher_id)

Clear watcher identified by *watcher_id* previously returned from **`PyContext_AddWatcher()`** for the current interpreter. Return `0` on success, or `-1` and set an exception on error (e.g. if the given *watcher_id* was never registered.)

Added in version 3.14.

type PyContextEvent

Enumeration of possible context object watcher events:

- `Py_CONTEXT_SWITCHED`: The current context has switched to a different context. The object passed to the watch callback is the now-current **`contextvars.Context`** object, or None if no context is current.

Added in version 3.14.

typedef int (*PyContext_WatchCallback)(PyContextEvent event, PyObject *obj)

Context object watcher callback function. The object passed to the callback is event-specific; see **`PyContextEvent`** for details.

If the callback returns with an exception set, it must return `-1`; this exception will be printed as an unraisable exception using **`PyErr_FormatUnraisable()`**. Otherwise it should return `0`.

There may already be a pending exception set on entry to the callback. In this case, the callback should return `0` with the same exception still set. This means the callback may not call any other API that can set an exception unless it saves and clears the exception state first, and restores it before returning.

Added in version 3.14.

Funciones variables de contexto:

PyObject *PyContextVar_New(const char *name, PyObject *def)

Return value: New reference.
Crea un nuevo objeto `ContextVar`. El parámetro *name* se usa para propósitos de introspección y depuración. El parámetro *def* especifica el valor predeterminado para la variable de contexto, o `NULL` para no especificar un valor predeterminado. Si se ha producido un error, esta función retorna `NULL`.

int PyContextVar_Get(PyObject *var, PyObject *default_value, PyObject **value)

Obtiene el valor de una variable de contexto. Retorna `-1` si se produjo un error durante la búsqueda y `0` si no se produjo ningún error, se haya encontrado o no un valor.

Si se encontró la variable de contexto, *value* será un puntero a ella. Si la variable de contexto *not* se encontró, *value* apuntará a:

- *default_value*, si no es `NULL`;
- el valor predeterminado de *var*, si no es `NULL`;
- `NULL`

A excepción de `NULL`, la función retorna una nueva referencia.

PyObject *PyContextVar_Set(PyObject *var, PyObject *value)

Return value: New reference.
Establece el valor de *var* en *value* en el contexto actual. Retorna un nuevo objeto token para este cambio, o `NULL` si se ha producido un error.

int PyContextVar_Reset(PyObject *var, PyObject *token)

Restablece el estado de la variable de contexto *var* a la que estaba antes **`PyContextVar_Set()`** que retornó el *token* fue llamado. Esta función retorna `0` en caso de éxito y `-1` en caso de error.

Objetos para indicaciones de tipado

Se proporcionan varios tipos incorporados para indicaciones de tipado. Actualmente existen dos tipos – GenericAlias y Union. Solo `GenericAlias` es expuesto a C.

PyObject *Py_GenericAlias(PyObject *origin, PyObject *args)

Part of the Stable ABI since version 3.9.
Crea un objeto GenericAlias. Equivalente a llamar la clase de Python **`types.GenericAlias`**. Los argumentos *origin* y *args* establecen los atributos `__origin__` y `__args__` de `GenericAlias` respectivamente. *origin* debe ser un PyTypeObject*, y *args* puede ser un PyTupleObject* o cualquier `PyObject*`. Si *args* no es una tupla, se construye automáticamente una tupla de un elemento y `__args__` se establece como `(args,)`. Se realiza una verificación mínima para los argumentos, por lo que la función tendrá éxito incluso si *origin* no es un tipo. El atributo `__parameters__` de `GenericAlias` se construye de manera perezosa a partir de `__args__`. En caso de falla, se levantará una excepción y retorna `NULL`.

Aquí hay un ejemplo sobre cómo hacer un tipo de extensión genérica:

```
static PyMethodDef my_obj_methods[] = {
    // Otros Métodos.
    ...
    {"__class_getitem__", Py_GenericAlias, METH_O|M
    ...

}
```

Ver también

El método del modelo de datos `__class_getitem__()`.

Added in version 3.9.

PyTypeObject Py_GenericAliasType

Part of the Stable ABI since version 3.9.
El tipo en C del objeto retornado por `Py_GenericAlias()`. Equivalente a `types.GenericAlias` en Python.

Added in version 3.9.

Curses C API

`curses` exposes a small C interface for extension modules. Consumers must include the header file `py_curses.h` (which is not included by default by `Python.h`) and **`import_curses()`** must be invoked, usually as part of the module initialisation function, to populate **`PyCurses_API`**.

Advertencia

Neither the C API nor the pure Python **`curses`** module are compatible with subinterpreters.

import_curses()

Import the curses C API. The macro does not need a semi-colon to be called.

On success, populate the **`PyCurses_API`** pointer.

On failure, set **`PyCurses_API`** to NULL and set an exception. The caller must check if an error occurred via **`PyErr_Occurred()`**:

```
import_curses();  // semi-colon is optional but rec
if (PyErr_Occurred()) { /* cleanup */ }
```

void **PyCurses_API

Dynamically allocated object containing the curses C API. This variable is only available once **`import_curses`** succeeds.

`PyCurses_API[0]` corresponds to **`PyCursesWindow_Type`**.

`PyCurses_API[1]`, `PyCurses_API[2]`, and

`PyCurses_API[3]` are pointers to predicate functions of type `int (*)(void)`.

When called, these predicates return whether **`curses.setupterm()`**, **`curses.initscr()`**, and **`curses.start_color()`** have been called respectively.

See also the convenience macros **`PyCursesSetupTermCalled`**, **`PyCursesInitialised`**, and **`PyCursesInitialisedColor`**.

> **Nota**
>
> The number of entries in this structure is subject to changes. Consider using **`PyCurses_API_pointers`** to check if new fields are available or not.

PyCurses_API_pointers

The number of accessible fields (`4`) in **`PyCurses_API`**. This number is incremented whenever new fields are added.

PyTypeObject PyCursesWindow_Type

The heap type corresponding to **`curses.window`**.

int PyCursesWindow_Check(PyObject *op)

Return true if *op* is a **`curses.window`** instance, false otherwise.

The following macros are convenience macros expanding into C statements. In particular, they can only be used as `macro;` or `macro`, but not `macro()` or `macro();`.

PyCursesSetupTermCalled

Macro checking if **`curses.setupterm()`** has been called.

The macro expansion is roughly equivalent to:

```
{
    typedef int (*predicate_t)(void);
    predicate_t was_setupterm_called = (predicate_t
    if (!was_setupterm_called()) {
        return NULL;
    }
}
```

PyCursesInitialised

Macro checking if `curses.initscr()` has been called.

The macro expansion is roughly equivalent to:

```
{
    typedef int (*predicate_t)(void);
    predicate_t was_initscr_called = (predicate_t)P
    if (!was_initscr_called()) {
        return NULL;
    }
}
```

PyCursesInitialisedColor

Macro checking if `curses.start_color()` has been called.

The macro expansion is roughly equivalent to:

```
{
    typedef int (*predicate_t)(void);
    predicate_t was_start_color_called = (predicate_
    if (!was_start_color_called()) {
        return NULL;
    }
}
```

Internal data

The following objects are exposed by the C API but should be considered internal-only.

PyCurses_CAPSULE_NAME

Name of the curses capsule to pass to `PyCapsule_Import()`.

Internal usage only. Use `import_curses` instead.

Objetos *DateTime*

Various date and time objects are supplied by the `datetime` module. Before using any of these functions, the header file `datetime.h` must be included in your source (note that this is not included by `Python.h`), and the macro `PyDateTime_IMPORT` must be invoked, usually as part of the module initialisation function. The macro puts a pointer to a C structure into a static variable, `PyDateTimeAPI`, that is used by the following macros.

PyDateTime_IMPORT()

Import the datetime C API.

On success, populate the `PyDateTimeAPI` pointer. On failure, set **`PyDateTimeAPI`** to `NULL` and set an exception. The caller must check if an error occurred via `PyErr_Occurred()`:

```
PyDateTime_IMPORT;
if (PyErr_Occurred()) { /* cleanup */ }
```

Advertencia

This is not compatible with subinterpreters.

type PyDateTime_CAPI

Structure containing the fields for the datetime C API.

The fields of this structure are private and subject to change.

Do not use this directly; prefer `PyDateTime_*` APIs instead.

PyDateTime_CAPI *PyDateTimeAPI

Dynamically allocated object containing the datetime C API.

This variable is only available once `PyDateTime_IMPORT` succeeds.

type PyDateTime_Date

Este subtipo de `PyObject` representa un *date object* de Python.

type PyDateTime_DateTime

Este subtipo de `PyObject` representa un *datetime object* de Python.

type PyDateTime_Time

Este subtipo de `PyObject` representa un *time object* de Python.

type PyDateTime_Delta

Este subtipo de `PyObject` representa la diferencia entre dos valores *datetime*.

PyTypeObject PyDateTime_DateType

Esta instancia de `PyTypeObject` representa el *date type* de Python; es el mismo objeto que `datetime.date` en la capa de Python.

PyTypeObject PyDateTime_DateTimeType

Esta instancia de `PyTypeObject` representa el *datetime type* de Python; es el mismo objeto que `datetime.datetime` en la capa de Python.

PyTypeObject PyDateTime_TimeType

Esta instancia de `PyObject` representa el *time type* de Python; es el mismo objeto que `datetime.time` en la capa de Python.

PyTypeObject PyDateTime_DeltaType

This instance of `PyTypeObject` represents the Python type for the difference between two datetime values; it is the same object as `datetime.timedelta` in the Python layer.

PyTypeObject PyDateTime_TZInfoType

Esta instancia de `PyTypeObject` representa el tipo de Python de *time zone info*; es el mismo objeto que `datetime.tzinfo` de la capa de Python.

Macro para acceder al singleton UTC:

PyObject *PyDateTime_TimeZone_UTC

Retorna la zona horaria singleton que representa UTC, el mismo objeto que `datetime.timezone.utc`.

Added in version 3.7.

Macros de verificación de tipo:

int PyDate_Check(PyObject *ob)

Retorna verdadero si *ob* es de tipo `PyDateTime_DateType` o un subtipo de **`PyDateTime_DateType`**. *ob* no debe ser `NULL`. Esta función siempre finaliza con éxito.

int PyDate_CheckExact(PyObject *ob)

Retorna verdadero si *ob* es de tipo `PyDateTime_DateType`. *ob* no debe ser `NULL`. Esta función siempre finaliza con éxito.

int PyDateTime_Check(PyObject *ob)

Retorna verdadero si *ob* es de tipo `PyDateTime_DateTimeType` o un subtipo de **`PyDateTime_DateTimeType`**. *ob* no debe ser `NULL`. Esta función siempre finaliza con éxito.

int PyDateTime_CheckExact(PyObject *ob)

Retorna verdadero si *ob* es de tipo `PyDateTime_DateTimeType`. *ob* no debe ser `NULL`. Esta función siempre finaliza con éxito.

int PyTime_Check(PyObject *ob)

Retorna verdadero si *ob* es de tipo `PyDateTime_TimeType` o un subtipo de **`PyDateTime_TimeType`**. *ob* no debe ser `NULL`. Esta función siempre finaliza con éxito.

int PyTime_CheckExact(PyObject *ob)

Retorna verdadero si *ob* es de tipo `PyDateTime_TimeType`. *ob* no debe ser `NULL`. Esta función siempre finaliza con éxito.

int PyDelta_Check(PyObject *ob)

Retorna verdadero si *ob* es de tipo `PyDateTime_DeltaType` o un subtipo de **`PyDateTime_DeltaType`**. *ob* no debe ser `NULL`. Esta función siempre finaliza con éxito.

int PyDelta_CheckExact(PyObject *ob)

Retorna verdadero si *ob* es de tipo `PyDateTime_DeltaType`. *ob* no debe ser `NULL`. Esta función siempre finaliza con éxito.

int PyTZInfo_Check(PyObject *ob)

Retorna verdadero si *ob* es de tipo `PyDateTime_TZInfoType` o un subtipo de **`PyDateTime_TZInfoType`**. *ob* no debe ser `NULL`. Esta función siempre finaliza con éxito.

int PyTZInfo_CheckExact(PyObject *ob)

Retorna verdadero si *ob* es de tipo `PyDateTime_TZInfoType`. *ob* no debe ser `NULL`. Esta función siempre finaliza con éxito.

Macros para crear objetos:

PyObject *PyDate_FromDate(int year, int month, int day)

Return value: New reference.
Retorna un objeto `datetime.date` con el año, mes y día

especificados.

PyObject *PyDateTime_FromDateAndTime(int year, int month, int day, int hour, int minute, int second, int usecond)

Return value: New reference.
Retorna un objeto `datetime.datetime` con el año, mes, día, hora, minuto, segundo y micro segundo especificados.

PyObject *PyDateTime_FromDateAndTimeAndFold(int year, int month, int day, int hour, int minute, int second, int usecond, int fold)

Return value: New reference.
Retorna un objeto `datetime.datetime` con el año, mes, día, hora, minuto, segundo, micro segundo y doblez especificados.

Added in version 3.6.

PyObject *PyTime_FromTime(int hour, int minute, int second, int usecond)

Return value: New reference.
Retorna un objeto `datetime.time` con la hora, minuto, segundo y micro segundo especificados.

PyObject *PyTime_FromTimeAndFold(int hour, int minute, int second, int usecond, int fold)

Return value: New reference.
Retorna un objeto `datetime.time` con la hora, minuto, segundo, micro segundo y doblez especificados.

Added in version 3.6.

PyObject *PyDelta_FromDSU(int days, int seconds, int useconds)

Return value: New reference.
Retorna un objeto `datetime.timedelta` que representa el número dado de días, segundos y micro segundos. La normalización se realiza de modo que el número resultante de

micro segundos y segundos se encuentre en los rangos documentados para los objetos **`datetime.timedelta`**.

PyObject *PyTimeZone_FromOffset(PyObject *offset)

Return value: New reference.
Retorna un objeto **`datetime.timezone`** con un desplazamiento fijo sin nombre representado por el argumento *offset*.

Added in version 3.7.

PyObject *PyTimeZone_FromOffsetAndName(PyObject *offset, PyObject *name)

Return value: New reference.
Retorna un objeto **`datetime.timezone`** con un desplazamiento fijo representado por el argumento *offset* y con tzname *name*.

Added in version 3.7.

Macros para extraer campos de objetos de fecha. El argumento debe ser una instancia de **`PyDateTime_Date`**, incluidas las subclases (como **`PyDateTime_DateTime`**). El argumento no debe ser `NULL`, y el tipo no se comprueba.

int PyDateTime_GET_YEAR(PyDateTime_Date *o)

Regrese el año, como un int positivo.

int PyDateTime_GET_MONTH(PyDateTime_Date *o)

Regresa el mes, como int del 1 al 12.

int PyDateTime_GET_DAY(PyDateTime_Date *o)

Retorna el día, como int del 1 al 31.

Macros para extraer campos de objetos de fecha y hora. El argumento debe ser una instancia de **`PyDateTime_DateTime`**, incluidas las subclases. El argumento no debe ser `NULL` y el tipo no es comprobado:

int PyDateTime_DATE_GET_HOUR(PyDateTime_DateTime *o)

Retorna la hora, como un int de 0 hasta 23.

int PyDateTime_DATE_GET_MINUTE(PyDateTime_DateTime *o)

Retorna el minuto, como un int de 0 hasta 59.

int PyDateTime_DATE_GET_SECOND(PyDateTime_DateTime *o)

Retorna el segundo, como un int de 0 hasta 59.

int PyDateTime_DATE_GET_MICROSECOND(PyDateTime_DateTime *o)

Retorna el micro segundo, como un int de 0 hasta 999999.

int PyDateTime_DATE_GET_FOLD(PyDateTime_DateTime *o)

Retorna el *fold*, como int de 0 a 1.

Added in version 3.6.

PyObject *PyDateTime_DATE_GET_TZINFO(PyDateTime_DateTime *o)

Retorna el tzinfo (que puede ser `None`).

Added in version 3.10.

Macros para extraer campos de objetos de tiempo. El argumento debe ser una instancia de **`PyDateTime_Time`**, incluidas las subclases. El argumento no debe ser `NULL` y el tipo no es comprobado:

int PyDateTime_TIME_GET_HOUR(PyDateTime_Time *o)

Retorna la hora, como un int de 0 hasta 23.

int PyDateTime_TIME_GET_MINUTE(PyDateTime_Time *o)

Retorna el minuto, como un int de 0 hasta 59.

int PyDateTime_TIME_GET_SECOND(PyDateTime_Time *o)

Retorna el segundo, como un int de 0 hasta 59.

int PyDateTime_TIME_GET_MICROSECOND(PyDateTime_Time *o)

Retorna el micro segundo, como un int de 0 hasta 999999.

int PyDateTime_TIME_GET_FOLD(PyDateTime_Time *o)

Retorna el *fold*, como int de 0 a 1.

Added in version 3.6.

PyObject *PyDateTime_TIME_GET_TZINFO(PyDateTime_Time *o)

Retorna el tzinfo (que puede ser `None`).

Added in version 3.10.

Macros para extraer campos de objetos delta de tiempo. El argumento debe ser una instancia de **`PyDateTime_Delta`**, incluidas las subclases. El argumento no debe ser `NULL` y el tipo no es comprobado:

int PyDateTime_DELTA_GET_DAYS(PyDateTime_Delta *o)

Retorna el número de días, como un int desde -999999999 a 999999999.

Added in version 3.3.

int PyDateTime_DELTA_GET_SECONDS(PyDateTime_Delta *o)

Retorna el número de segundos, como un int de 0 a 86399.

Added in version 3.3.

int PyDateTime_DELTA_GET_MICROSECONDS(PyDateTime_Delta *o)

Retorna el número de micro segundos, como un int de 0 a 999999.

Added in version 3.3.

Macros para la conveniencia de módulos que implementan la API DB:

PyObject *PyDateTime_FromTimestamp(PyObject *args)

Return value: New reference.
Crea y retorna un nuevo objeto `datetime.datetime` dado una tupla de argumentos adecuada para pasar a `datetime.datetime.fromtimestamp()`.

PyObject *PyDate_FromTimestamp(PyObject *args)

Return value: New reference.
Crea y retorna un nuevo objeto `datetime.date` dado una tupla de argumentos adecuada para pasar a `datetime.date.fromtimestamp()`.

Internal data

The following symbols are exposed by the C API but should be considered internal-only.

PyDateTime_CAPSULE_NAME

Name of the datetime capsule to pass to `PyCapsule_Import()`.

Internal usage only. Use `PyDateTime_IMPORT` instead.

Interpreter initialization and finalization

See Python Initialization Configuration for details on how to configure the interpreter prior to initialization.

Before Python initialization

In an application embedding Python, the `Py_Initialize()` function must be called before using any other Python/C API functions; with the exception of a few functions and the global configuration variables.

The following functions can be safely called before Python is initialized:

- Functions that initialize the interpreter:
    - `Py_Initialize()`
    - `Py_InitializeEx()`
    - `Py_InitializeFromConfig()`
    - `Py_BytesMain()`
    - `Py_Main()`
    - the runtime pre-initialization functions covered in Configuración de inicialización de Python
- Configuration functions:
    - `PyImport_AppendInittab()`
    - `PyImport_ExtendInittab()`
    - `PyInitFrozenExtensions()`
    - `PyMem_SetAllocator()`
    - `PyMem_SetupDebugHooks()`
    - `PyObject_SetArenaAllocator()`
    - `Py_SetProgramName()`
    - `Py_SetPythonHome()`
    - the configuration functions covered in Configuración de inicialización de Python

- Informative functions:
    - `Py_IsInitialized()`
    - `PyMem_GetAllocator()`
    - `PyObject_GetArenaAllocator()`
    - `Py_GetBuildInfo()`
    - `Py_GetCompiler()`
    - `Py_GetCopyright()`
    - `Py_GetPlatform()`
    - `Py_GetVersion()`
    - `Py_IsInitialized()`
- Utilities:
    - `Py_DecodeLocale()`
    - the status reporting and utility functions covered in Configuración de inicialización de Python
- Memory allocators:
    - `PyMem_RawMalloc()`
    - `PyMem_RawRealloc()`
    - `PyMem_RawCalloc()`
    - `PyMem_RawFree()`
- Synchronization:
    - `PyMutex_Lock()`
    - `PyMutex_Unlock()`

Nota

Despite their apparent similarity to some of the functions listed above, the following functions **should not be called** before the interpreter has been initialized: `Py_EncodeLocale()`, `PyEval_InitThreads()`, and `Py_RunMain()`.

Global configuration variables

Python has variables for the global configuration to control different features and options. By default, these flags are controlled by command line options.

When a flag is set by an option, the value of the flag is the number of times that the option was set. For example, `-b` sets `Py_BytesWarningFlag` to 1 and `-bb` sets

`Py_BytesWarningFlag` to 2.

int Py_BytesWarningFlag

This API is kept for backward compatibility: setting **`PyConfig.bytes_warning`** should be used instead, see Python Initialization Configuration.

Issue a warning when comparing **`bytes`** or **`bytearray`** with **`str`** or **`bytes`** with **`int`**. Issue an error if greater or equal to `2`.

Set by the **`-b`** option.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_DebugFlag

This API is kept for backward compatibility: setting **`PyConfig.parser_debug`** should be used instead, see Python Initialization Configuration.

Turn on parser debugging output (for expert only, depending on compilation options).

Set by the **`-d`** option and the **`PYTHONDEBUG`** environment variable.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_DontWriteBytecodeFlag

This API is kept for backward compatibility: setting **`PyConfig.write_bytecode`** should be used instead, see Python Initialization Configuration.

If set to non-zero, Python won’t try to write `.pyc` files on the import of source modules.

Set by the **`-B`** option and the **`PYTHONDONTWRITEBYTECODE`** environment variable.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_FrozenFlag

This API is kept for backward compatibility: setting **`PyConfig.pathconfig_warnings`** should be used instead, see Python Initialization Configuration.

Private flag used by `_freeze_module` and `frozenmain` programs.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_HashRandomizationFlag

This API is kept for backward compatibility: setting **`PyConfig.hash_seed`** and **`PyConfig.use_hash_seed`** should be used instead, see Python Initialization Configuration.

Set to `1` if the **`PYTHONHASHSEED`** environment variable is set to a non-empty string.

If the flag is non-zero, read the **`PYTHONHASHSEED`** environment variable to initialize the secret hash seed.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_IgnoreEnvironmentFlag

This API is kept for backward compatibility: setting **`PyConfig.use_environment`** should be used instead, see Python Initialization Configuration.

Ignore all **`PYTHON*`** environment variables, e.g. **`PYTHONPATH`** and **`PYTHONHOME`**, that might be set.

Set by the **`-E`** and **`-I`** options.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_InspectFlag

This API is kept for backward compatibility: setting **`PyConfig.inspect`** should be used instead, see Python Initialization Configuration.

When a script is passed as first argument or the `-c` option is used, enter interactive mode after executing the script or the command, even when `sys.stdin` does not appear to be a terminal.

Set by the `-i` option and the `PYTHONINSPECT` environment variable.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_InteractiveFlag

This API is kept for backward compatibility: setting `PyConfig.interactive` should be used instead, see Python Initialization Configuration.

Set by the `-i` option.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_IsolatedFlag

This API is kept for backward compatibility: setting `PyConfig.isolated` should be used instead, see Python Initialization Configuration.

Run Python in isolated mode. In isolated mode `sys.path` contains neither the script's directory nor the user's site-packages directory.

Set by the `-I` option.

Added in version 3.4.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_LegacyWindowsFSEncodingFlag

This API is kept for backward compatibility: setting `PyPreConfig.legacy_windows_fs_encoding` should be used instead, see Python Initialization Configuration.

If the flag is non-zero, use the `mbcs` encoding with `replace` error handler, instead of the UTF-8 encoding with `surrogatepass` error handler, for the filesystem encoding and error handler.

Set to `1` if the **`PYTHONLEGACYWINDOWSFSENCODING`** environment variable is set to a non-empty string.

See **PEP 529** [https://peps.python.org/pep-0529/] for more details.

Availability: Windows.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_LegacyWindowsStdioFlag

This API is kept for backward compatibility: setting **`PyConfig.legacy_windows_stdio`** should be used instead, see Python Initialization Configuration.

If the flag is non-zero, use **`io.FileIO`** instead of **`io._WindowsConsoleIO`** for **`sys`** standard streams.

Set to `1` if the **`PYTHONLEGACYWINDOWSSTDIO`** environment variable is set to a non-empty string.

See **PEP 528** [https://peps.python.org/pep-0528/] for more details.

Availability: Windows.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_NoSiteFlag

This API is kept for backward compatibility: setting **`PyConfig.site_import`** should be used instead, see Python Initialization Configuration.

Disable the import of the module **`site`** and the site-dependent manipulations of **`sys.path`** that it entails. Also disable these manipulations if **`site`** is explicitly imported later (call **`site.main()`** if you want them to be triggered).

Set by the **`-S`** option.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_NoUserSiteDirectory

This API is kept for backward compatibility: setting **`PyConfig.user_site_directory`** should be used instead, see Python Initialization Configuration.

Don't add the **`user site-packages directory`** to **`sys.path`**.

Set by the **`-s`** and **`-I`** options, and the **`PYTHONNOUSERSITE`** environment variable.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_OptimizeFlag

This API is kept for backward compatibility: setting **`PyConfig.optimization_level`** should be used instead, see Python Initialization Configuration.

Set by the **`-O`** option and the **`PYTHONOPTIMIZE`** environment variable.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_QuietFlag

This API is kept for backward compatibility: setting **`PyConfig.quiet`** should be used instead, see Python Initialization Configuration.

Don't display the copyright and version messages even in interactive mode.

Set by the **`-q`** option.

Added in version 3.2.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_UnbufferedStdioFlag

This API is kept for backward compatibility: setting **`PyConfig.buffered_stdio`** should be used instead, see Python Initialization Configuration.

Force the stdout and stderr streams to be unbuffered.

Set by the **`-u`** option and the **`PYTHONUNBUFFERED`** environment variable.

Deprecated since version 3.12, will be removed in version 3.15.

int Py_VerboseFlag

This API is kept for backward compatibility: setting **`PyConfig.verbose`** should be used instead, see Python Initialization Configuration.

Print a message each time a module is initialized, showing the place (filename or built-in module) from which it is loaded. If greater or equal to `2`, print a message for each file that is checked for when searching for a module. Also provides information on module cleanup at exit.

Set by the **`-v`** option and the **`PYTHONVERBOSE`** environment variable.

Deprecated since version 3.12, will be removed in version 3.15.

Initializing and finalizing the interpreter

void Py_Initialize()

Part of the Stable ABI.
Initialize the Python interpreter. In an application embedding Python, this should be called before using any other Python/C API functions; see Before Python Initialization for the few exceptions.

This initializes the table of loaded modules (`sys.modules`), and creates the fundamental modules **`builtins`**, **`__main__`** and **`sys`**. It also initializes the module search path (`sys.path`). It does not set `sys.argv`; use the Python

Initialization Configuration API for that. This is a no-op when called for a second time (without calling `Py_FinalizeEx()` first). There is no return value; it is a fatal error if the initialization fails.

Use `Py_InitializeFromConfig()` to customize the Python Initialization Configuration.

> **Nota**
>
> On Windows, changes the console mode from `O_TEXT` to `O_BINARY`, which will also affect non-Python uses of the console using the C Runtime.

void Py_InitializeEx(int initsigs)

Part of the Stable ABI.
This function works like `Py_Initialize()` if *initsigs* is `1`. If *initsigs* is `0`, it skips initialization registration of signal handlers, which may be useful when CPython is embedded as part of a larger application.

Use `Py_InitializeFromConfig()` to customize the Python Initialization Configuration.

PyStatus Py_InitializeFromConfig(const PyConfig *config)

Initialize Python from *config* configuration, as described in Inicialización con PyConfig.

See the Configuración de inicialización de Python section for details on pre-initializing the interpreter, populating the runtime configuration structure, and querying the returned status structure.

int Py_IsInitialized()

Part of the Stable ABI.
Return true (nonzero) when the Python interpreter has been initialized, false (zero) if not. After `Py_FinalizeEx()` is called, this returns false until `Py_Initialize()` is called

again.

int Py_IsFinalizing()

Part of the Stable ABI since version 3.13.
Return true (non-zero) if the main Python interpreter is shutting down. Return false (zero) otherwise.

Added in version 3.13.

int Py_FinalizeEx()

Part of the Stable ABI since version 3.6.
Undo all initializations made by `Py_Initialize()` and subsequent use of Python/C API functions, and destroy all sub-interpreters (see `Py_NewInterpreter()` below) that were created and not yet destroyed since the last call to `Py_Initialize()`. This is a no-op when called for a second time (without calling `Py_Initialize()` again first).

Since this is the reverse of `Py_Initialize()`, it should be called in the same thread with the same interpreter active. That means the main thread and the main interpreter. This should never be called while `Py_RunMain()` is running.

Normally the return value is `0`. If there were errors during finalization (flushing buffered data), `-1` is returned.

Note that Python will do a best effort at freeing all memory allocated by the Python interpreter. Therefore, any C-Extension should make sure to correctly clean up all of the previously allocated PyObjects before using them in subsequent calls to `Py_Initialize()`. Otherwise it could introduce vulnerabilities and incorrect behavior.

This function is provided for a number of reasons. An embedding application might want to restart Python without having to restart the application itself. An application that has loaded the Python interpreter from a dynamically loadable library (or DLL) might want to free all memory allocated by Python before unloading the DLL. During a hunt for memory leaks in an application a developer might want to free all memory allocated by Python before exiting from the application.

Bugs and caveats: The destruction of modules and objects in modules is done in random order; this may cause destructors (`__del__()` methods) to fail when they depend on other objects (even functions) or modules. Dynamically loaded extension modules loaded by Python are not unloaded. Small amounts of memory allocated by the Python interpreter may not be freed (if you find a leak, please report it). Memory tied up in circular references between objects is not freed. Interned strings will all be deallocated regardless of their reference count. Some memory allocated by extension modules may not be freed. Some extensions may not work properly if their initialization routine is called more than once; this can happen if an application calls `Py_Initialize()` and `Py_FinalizeEx()` more than once. `Py_FinalizeEx()` must not be called recursively from within itself. Therefore, it must not be called by any code that may be run as part of the interpreter shutdown process, such as `atexit` handlers, object finalizers, or any code that may be run while flushing the stdout and stderr files.

Raises an auditing event `cpython._PySys_ClearAuditHooks` with no arguments.

Added in version 3.6.

void Py_Finalize()

Part of the Stable ABI.
This is a backwards-compatible version of `Py_FinalizeEx()` that disregards the return value.

int Py_BytesMain(int argc, char **argv)

Part of the Stable ABI since version 3.8.
Similar to `Py_Main()` but *argv* is an array of bytes strings, allowing the calling application to delegate the text decoding step to the CPython runtime.

Added in version 3.8.

int Py_Main(int argc, wchar_t **argv)

Part of the Stable ABI.
The main program for the standard interpreter, encapsulating a full initialization/finalization cycle, as well as additional behaviour to implement reading configurations settings from the environment and command line, and then executing `__main__` in accordance with Línea de comando.

This is made available for programs which wish to support the full CPython command line interface, rather than just embedding a Python runtime in a larger application.

The *argc* and *argv* parameters are similar to those which are passed to a C program's **`main()`** function, except that the *argv* entries are first converted to `wchar_t` using **`Py_DecodeLocale()`**. It is also important to note that the argument list entries may be modified to point to strings other than those passed in (however, the contents of the strings pointed to by the argument list are not modified).

The return value is `2` if the argument list does not represent a valid Python command line, and otherwise the same as **`Py_RunMain()`**.

In terms of the CPython runtime configuration APIs documented in the runtime configuration section (and without accounting for error handling), `Py_Main` is approximately equivalent to:

```
PyConfig config;
PyConfig_InitPythonConfig(&config);
PyConfig_SetArgv(&config, argc, argv);
Py_InitializeFromConfig(&config);
PyConfig_Clear(&config);

Py_RunMain();
```

In normal usage, an embedding application will call this function *instead* of calling **`Py_Initialize()`**, **`Py_InitializeEx()`** or **`Py_InitializeFromConfig()`** directly, and all settings will be applied as described elsewhere in this documentation. If this function is instead called *after* a preceding runtime initialization API call, then exactly which environmental and command line configuration settings will be updated is version dependent (as it depends on which settings correctly support being modified after they have already been set once when the runtime was first initialized).

int Py_RunMain(void)

Executes the main module in a fully configured CPython runtime.

Executes the command (**`PyConfig.run_command`**), the script (**`PyConfig.run_filename`**) or the module (**`PyConfig.run_module`**) specified on the command line or in the configuration. If none of these values are set, runs the interactive Python prompt (REPL) using the `__main__` module's global namespace.

If **`PyConfig.inspect`** is not set (the default), the return value will be `0` if the interpreter exits normally (that is, without raising an exception), the exit status of an unhandled **`SystemExit`**, or `1` for any other unhandled exception.

If **`PyConfig.inspect`** is set (such as when the **`-i`** option is used), rather than returning when the interpreter exits, execution will instead resume in an interactive Python prompt (REPL) using the `__main__` module's global namespace. If the interpreter exited with an exception, it is immediately raised in the REPL session. The function return value is then determined by the way the *REPL session* terminates: `0`, `1`, or the status of a **`SystemExit`**, as specified above.

This function always finalizes the Python interpreter before it returns.

See Python Configuration for an example of a customized Python that always runs in isolated mode using `Py_RunMain()`.

int PyUnstable_AtExit(PyInterpreterState *interp, void (*func)(void*), void *data)

This is Unstable API. It may change without warning in minor releases.

Register an `atexit` callback for the target interpreter *interp*. This is similar to `Py_AtExit()`, but takes an explicit interpreter and data pointer for the callback.

There must be an attached thread state for *interp*.

Added in version 3.13.

Cautions regarding runtime finalization

In the late stage of interpreter shutdown, after attempting to wait for non-daemon threads to exit (though this can be interrupted by `KeyboardInterrupt`) and running the `atexit` functions, the runtime is marked as *finalizing*: `Py_IsFinalizing()` and `sys.is_finalizing()` return true. At this point, only the *finalization thread* that initiated finalization (typically the main thread) is allowed to acquire the GIL.

If any thread, other than the finalization thread, attempts to attach a thread state during finalization, either explicitly or implicitly, the thread enters **a permanently blocked state** where it remains until the program exits. In most cases this is harmless, but this can result in deadlock if a later stage of finalization attempts to acquire a lock owned by the blocked thread, or otherwise waits on the blocked thread.

Gross? Yes. This prevents random crashes and/or unexpectedly skipped C++ finalizations further up the call stack when such threads were forcibly exited here in CPython 3.13 and earlier. The

CPython runtime thread state C APIs have never had any error reporting or handling expectations at thread state attachment time that would've allowed for graceful exit from this situation. Changing that would require new stable C APIs and rewriting the majority of C code in the CPython ecosystem to use those with error handling.

Process-wide parameters

void Py_SetProgramName(const wchar_t *name)

Part of the Stable ABI.
This API is kept for backward compatibility: setting **`PyConfig.program_name`** should be used instead, see Python Initialization Configuration.

This function should be called before **`Py_Initialize()`** is called for the first time, if it is called at all. It tells the interpreter the value of the `argv[0]` argument to the **`main()`** function of the program (converted to wide characters). This is used by **`Py_GetPath()`** and some other functions below to find the Python run-time libraries relative to the interpreter executable. The default value is `'python'`. The argument should point to a zero-terminated wide character string in static storage whose contents will not change for the duration of the program's execution. No code in the Python interpreter will change the contents of this storage.

Use **`Py_DecodeLocale()`** to decode a bytes string to get a wchar_t* string.

Deprecated since version 3.11, will be removed in version 3.15.

wchar_t *Py_GetProgramName()

Part of the Stable ABI.
Return the program name set with **`PyConfig.program_name`**, or the default. The returned string points into static storage; the caller should not modify its value.

This function should not be called before `Py_Initialize()`, otherwise it returns `NULL`.

Distinto en la versión 3.10: It now returns `NULL` if called before `Py_Initialize()`.

Deprecated since version 3.13, will be removed in version 3.15: Use `PyConfig_Get("executable")` (`sys.executable`) instead.

wchar_t *Py_GetPrefix()

Part of the Stable ABI.
Return the *prefix* for installed platform-independent files. This is derived through a number of complicated rules from the program name set with `PyConfig.program_name` and some environment variables; for example, if the program name is `'/usr/local/bin/python'`, the prefix is `'/usr/local'`. The returned string points into static storage; the caller should not modify its value. This corresponds to the **prefix** variable in the top-level `Makefile` and the `--prefix` argument to the **configure** script at build time. The value is available to Python code as `sys.base_prefix`. It is only useful on Unix. See also the next function.

This function should not be called before `Py_Initialize()`, otherwise it returns `NULL`.

Distinto en la versión 3.10: It now returns `NULL` if called before `Py_Initialize()`.

Deprecated since version 3.13, will be removed in version 3.15: Use `PyConfig_Get("base_prefix")` (`sys.base_prefix`) instead. Use **`PyConfig_Get("prefix")`** (`sys.prefix`) if virtual environments need to be handled.

wchar_t *Py_GetExecPrefix()

Part of the Stable ABI.
Return the *exec-prefix* for installed platform-*dependent* files. This is derived through a number of complicated rules from

the program name set with **`PyConfig.program_name`** and some environment variables; for example, if the program name is `'/usr/local/bin/python'`, the exec-prefix is `'/usr/local'`. The returned string points into static storage; the caller should not modify its value. This corresponds to the **exec_prefix** variable in the top-level `Makefile` and the `--exec-prefix` argument to the **configure** script at build time. The value is available to Python code as `sys.base_exec_prefix`. It is only useful on Unix.

Background: The exec-prefix differs from the prefix when platform dependent files (such as executables and shared libraries) are installed in a different directory tree. In a typical installation, platform dependent files may be installed in the `/usr/local/plat` subtree while platform independent may be installed in `/usr/local`.

Generally speaking, a platform is a combination of hardware and software families, e.g. Sparc machines running the Solaris 2.x operating system are considered the same platform, but Intel machines running Solaris 2.x are another platform, and Intel machines running Linux are yet another platform. Different major revisions of the same operating system generally also form different platforms. Non-Unix operating systems are a different story; the installation strategies on those systems are so different that the prefix and exec-prefix are meaningless, and set to the empty string. Note that compiled Python bytecode files are platform independent (but not independent from the Python version by which they were compiled!).

System administrators will know how to configure the **mount** or **automount** programs to share `/usr/local` between platforms while having `/usr/local/plat` be a different filesystem for each platform.

This function should not be called before **`Py_Initialize()`**, otherwise it returns `NULL`.

Distinto en la versión 3.10: It now returns `NULL` if called before `Py_Initialize()`.

Deprecated since version 3.13, will be removed in version 3.15: Use `PyConfig_Get("base_exec_prefix")` (`sys.base_exec_prefix`) instead. Use **`PyConfig_Get("exec_prefix")`** (`sys.exec_prefix`) if virtual environments need to be handled.

wchar_t *Py_GetProgramFullPath()

Part of the Stable ABI.
Return the full program name of the Python executable; this is computed as a side-effect of deriving the default module search path from the program name (set by `PyConfig.program_name`). The returned string points into static storage; the caller should not modify its value. The value is available to Python code as `sys.executable`.

This function should not be called before `Py_Initialize()`, otherwise it returns `NULL`.

Distinto en la versión 3.10: It now returns `NULL` if called before `Py_Initialize()`.

Deprecated since version 3.13, will be removed in version 3.15: Use `PyConfig_Get("executable")` (`sys.executable`) instead.

wchar_t *Py_GetPath()

Part of the Stable ABI.
Return the default module search path; this is computed from the program name (set by `PyConfig.program_name`) and some environment variables. The returned string consists of a series of directory names separated by a platform dependent delimiter character. The delimiter character is `':'` on Unix and macOS, `';'` on Windows. The returned string points into static storage; the caller should not modify its value. The list `sys.path` is initialized with this value on interpreter startup; it can be (and usually is) modified later to change the search path for loading modules.

This function should not be called before **`Py_Initialize()`**, otherwise it returns `NULL`.

Distinto en la versión 3.10: It now returns `NULL` if called before **`Py_Initialize()`**.

Deprecated since version 3.13, will be removed in version 3.15: Use **`PyConfig_Get("module_search_paths")`** (**`sys.path`**) instead.

const char *Py_GetVersion()

Part of the Stable ABI.
Return the version of this Python interpreter. This is a string that looks something like

```
"3.0a5+ (py3k:63103M, May 12 2008, 00:53:55) \n[GCC
```

The first word (up to the first space character) is the current Python version; the first characters are the major and minor version separated by a period. The returned string points into static storage; the caller should not modify its value. The value is available to Python code as **`sys.version`**.

See also the **`Py_Version`** constant.

const char *Py_GetPlatform()

Part of the Stable ABI.
Return the platform identifier for the current platform. On Unix, this is formed from the «official» name of the operating system, converted to lower case, followed by the major revision number; e.g., for Solaris 2.x, which is also known as SunOS 5.x, the value is `'sunos5'`. On macOS, it is `'darwin'`. On Windows, it is `'win'`. The returned string points into static storage; the caller should not modify its value. The value is available to Python code as `sys.platform`.

const char *Py_GetCopyright()

Part of the Stable ABI.

Return the official copyright string for the current Python version, for example

```
'Copyright 1991-1995 Stichting Mathematisch
Centrum, Amsterdam'
```

The returned string points into static storage; the caller should not modify its value. The value is available to Python code as `sys.copyright`.

const char *Py_GetCompiler()

Part of the Stable ABI.
Return an indication of the compiler used to build the current Python version, in square brackets, for example:

```
"[GCC 2.7.2.2]"
```

The returned string points into static storage; the caller should not modify its value. The value is available to Python code as part of the variable `sys.version`.

const char *Py_GetBuildInfo()

Part of the Stable ABI.
Return information about the sequence number and build date and time of the current Python interpreter instance, for example

```
"#67, Aug  1 1997, 22:34:28"
```

The returned string points into static storage; the caller should not modify its value. The value is available to Python code as part of the variable `sys.version`.

void PySys_SetArgvEx(int argc, wchar_t **argv, int updatepath)

Part of the Stable ABI.
This API is kept for backward compatibility: setting **`PyConfig.argv`**, **`PyConfig.parse_argv`** and **`PyConfig.safe_path`** should be used instead, see Python Initialization Configuration.

Set `sys.argv` based on *argc* and *argv*. These parameters are similar to those passed to the program's `main()` function with the difference that the first entry should refer to the script file to be executed rather than the executable hosting the Python interpreter. If there isn't a script that will be run, the first entry in *argv* can be an empty string. If this function fails to initialize `sys.argv`, a fatal condition is signalled using `Py_FatalError()`.

If *updatepath* is zero, this is all the function does. If *updatepath* is non-zero, the function also modifies `sys.path` according to the following algorithm:

- If the name of an existing script is passed in `argv[0]`, the absolute path of the directory where the script is located is prepended to `sys.path`.
- Otherwise (that is, if *argc* is `0` or `argv[0]` doesn't point to an existing file name), an empty string is prepended to `sys.path`, which is the same as prepending the current working directory (`"."`).

Use `Py_DecodeLocale()` to decode a bytes string to get a wchar_t* string.

See also `PyConfig.orig_argv` and `PyConfig.argv` members of the Python Initialization Configuration.

Nota

It is recommended that applications embedding the Python interpreter for purposes other than executing a single script pass `0` as *updatepath*, and update `sys.path` themselves if desired. See **CVE 2008-5983** [https://www.cve.org/CVERecord?id=CVE-2008-5983].

On versions before 3.1.3, you can achieve the same effect by manually popping the first `sys.path` element after having called `PySys_SetArgv()`, for example using:

```
PyRun_SimpleString("import sys; sys.path.pop(0)\n"
```

Added in version 3.1.3.

Deprecated since version 3.11, will be removed in version 3.15.

void PySys_SetArgv(int argc, wchar_t **argv)

Part of the Stable ABI.
This API is kept for backward compatibility: setting `PyConfig.argv` and `PyConfig.parse_argv` should be used instead, see Python Initialization Configuration.

This function works like `PySys_SetArgvEx()` with *updatepath* set to `1` unless the **python** interpreter was started with the `-I`.

Use `Py_DecodeLocale()` to decode a bytes string to get a wchar_t* string.

See also `PyConfig.orig_argv` and `PyConfig.argv` members of the Python Initialization Configuration.

Distinto en la versión 3.4: The *updatepath* value depends on `-I`.

Deprecated since version 3.11, will be removed in version 3.15.

void Py_SetPythonHome(const wchar_t *home)

Part of the Stable ABI.
This API is kept for backward compatibility: setting `PyConfig.home` should be used instead, see Python Initialization Configuration.

Set the default «home» directory, that is, the location of the standard Python libraries. See `PYTHONHOME` for the meaning of the argument string.

The argument should point to a zero-terminated character string in static storage whose contents will not change for the duration of the program's execution. No code in the Python interpreter will change the contents of this storage.

Use `Py_DecodeLocale()` to decode a bytes string to get a wchar_t* string.

Deprecated since version 3.11, will be removed in version 3.15.

wchar_t *Py_GetPythonHome()

Part of the Stable ABI.
Return the default «home», that is, the value set by `PyConfig.home`, or the value of the `PYTHONHOME` environment variable if it is set.

This function should not be called before `Py_Initialize()`, otherwise it returns `NULL`.

Distinto en la versión 3.10: It now returns `NULL` if called before `Py_Initialize()`.

Deprecated since version 3.13, will be removed in version 3.15: Use `PyConfig_Get("home")` or the `PYTHONHOME` environment variable instead.

Thread states and the global interpreter lock

Unless on a free-threaded build of CPython, the Python interpreter is generally not thread-safe. In order to support multi-threaded Python programs, there's a global lock, called the global interpreter lock or GIL, that must be held by a thread before accessing Python objects. Without the lock, even the simplest operations could cause problems in a multi-threaded program: for example, when two threads simultaneously increment the reference count of the same object, the reference count could end up being incremented only once instead of twice.

As such, only a thread that holds the GIL may operate on Python objects or invoke Python's C API.

In order to emulate concurrency, the interpreter regularly tries to switch threads between bytecode instructions (see `sys.setswitchinterval()`). This is why locks are also necessary for thread-safety in pure-Python code.

Additionally, the global interpreter lock is released around blocking I/O operations, such as reading or writing to a file. From the C API, this is done by detaching the thread state.

The Python interpreter keeps some thread-local information inside a data structure called `PyThreadState`, known as a thread state. Each thread has a thread-local pointer to a `PyThreadState`; a thread state referenced by this pointer is considered to be attached.

A thread can only have one attached thread state at a time. An attached thread state is typically analogous with holding the GIL, except on free-threaded builds. On builds with the GIL enabled, attaching a thread state will block until the GIL can be acquired. However, even on builds with the GIL disabled, it is still required to have an attached thread state, as the interpreter needs to keep track of which threads may access Python objects.

Nota

Even on the free-threaded build, attaching a thread state may block, as the GIL can be re-enabled or threads might be temporarily suspended (such as during a garbage collection).

Generally, there will always be an attached thread state when using Python's C API, including during embedding and when implementing methods, so it's uncommon to need to set up a thread state on your own. Only in some specific cases, such as in a **`Py_BEGIN_ALLOW_THREADS`** block or in a fresh thread, will the thread not have an attached thread state. If uncertain, check if **`PyThreadState_GetUnchecked()`** returns `NULL`.

If it turns out that you do need to create a thread state, call **`PyThreadState_New()`** followed by **`PyThreadState_Swap()`**, or use the dangerous **`PyGILState_Ensure()`** function.

Detaching the thread state from extension code

Most extension code manipulating the thread state has the following simple structure:

```
Save the thread state in a local variable.
... Do some blocking I/O operation ...
Restore the thread state from the local variable.
```

This is so common that a pair of macros exists to simplify it:

```
Py_BEGIN_ALLOW_THREADS
... Do some blocking I/O operation ...
Py_END_ALLOW_THREADS
```

The **`Py_BEGIN_ALLOW_THREADS`** macro opens a new block and declares a hidden local variable; the **`Py_END_ALLOW_THREADS`** macro closes the block.

The block above expands to the following code:

```
PyThreadState *_save;

_save = PyEval_SaveThread();
... Do some blocking I/O operation ...
PyEval_RestoreThread(_save);
```

Here is how these functions work:

The attached thread state implies that the GIL is held for the interpreter. To detach it, **`PyEval_SaveThread()`** is called and the result is stored in a local variable.

By detaching the thread state, the GIL is released, which allows other threads to attach to the interpreter and execute while the current thread performs blocking I/O. When the I/O operation is complete, the old thread state is reattached by calling **`PyEval_RestoreThread()`**, which will wait until the GIL can be acquired.

Nota

Performing blocking I/O is the most common use case for detaching the thread state, but it is also useful to call it over long-running native code that doesn't need access to Python objects or Python's C API. For example, the standard **`zlib`** and **`hashlib`** modules detach the thread state when compressing or hashing data.

On a free-threaded build, the GIL is usually out of the question, but **detaching the thread state is still required**, because the interpreter periodically needs to block all threads to get a consistent view of Python objects without the risk of race conditions. For example, CPython currently suspends all threads for a short period of time while running the garbage collector.

Advertencia

Detaching the thread state can lead to unexpected behavior during interpreter finalization. See Cautions regarding runtime finalization for more details.

APIs

The following macros are normally used without a trailing semicolon; look for example usage in the Python source distribution.

Nota

These macros are still necessary on the free-threaded build to prevent deadlocks.

Py_BEGIN_ALLOW_THREADS

Part of the Stable ABI.
This macro expands to `{ PyThreadState *_save; _save = PyEval_SaveThread();`. Note that it contains an opening brace; it must be matched with a following **`Py_END_ALLOW_THREADS`** macro. See above for further discussion of this macro.

Py_END_ALLOW_THREADS

Part of the Stable ABI.
This macro expands to `PyEval_RestoreThread(_save); }`. Note that it contains a closing brace; it must be matched with an earlier **`Py_BEGIN_ALLOW_THREADS`** macro. See above for further discussion of this macro.

Py_BLOCK_THREADS

Part of the Stable ABI.
This macro expands to `PyEval_RestoreThread(_save);`: it is equivalent to **`Py_END_ALLOW_THREADS`** without the closing brace.

Py_UNBLOCK_THREADS

Part of the Stable ABI.

This macro expands to `_save = PyEval_SaveThread();`: it is equivalent to **`Py_BEGIN_ALLOW_THREADS`** without the opening brace and variable declaration.

Non-Python created threads

When threads are created using the dedicated Python APIs (such as the **`threading`** module), a thread state is automatically associated with them, However, when a thread is created from native code (for example, by a third-party library with its own thread management), it doesn't hold an attached thread state.

If you need to call Python code from these threads (often this will be part of a callback API provided by the aforementioned third-party library), you must first register these threads with the interpreter by creating a new thread state and attaching it.

The most robust way to do this is through **`PyThreadState_New()`** followed by **`PyThreadState_Swap()`**.

Nota

`PyThreadState_New` requires an argument pointing to the desired interpreter; such a pointer can be acquired via a call to **`PyInterpreterState_Get()`** from the code where the thread was created.

For example:

```
/* The return value of PyInterpreterState_Get() from the
   function that created this thread. */
PyInterpreterState *interp = thread_data->interp;

/* Create a new thread state for the interpreter. It doe
   attached. */
PyThreadState *tstate = PyThreadState_New(interp);

/* Attach the thread state, which will acquire the GIL.
```

```
PyThreadState_Swap(tstate);

/* Perform Python actions here. */
result = CallSomeFunction();
/* evaluate result or handle exception */

/* Destroy the thread state. No Python API allowed beyor
PyThreadState_Clear(tstate);
PyThreadState_DeleteCurrent();
```

Advertencia

If the interpreter finalized before `PyThreadState_Swap` was called, then `interp` will be a dangling pointer!

Legacy API

Another common pattern to call Python code from a non-Python thread is to use **`PyGILState_Ensure()`** followed by a call to **`PyGILState_Release()`**.

These functions do not work well when multiple interpreters exist in the Python process. If no Python interpreter has ever been used in the current thread (which is common for threads created outside Python), `PyGILState_Ensure` will create and attach a thread state for the «main» interpreter (the first interpreter in the Python process).

Additionally, these functions have thread-safety issues during interpreter finalization. Using `PyGILState_Ensure` during finalization will likely crash the process.

Usage of these functions look like such:

```
PyGILState_STATE gstate;
gstate = PyGILState_Ensure();

/* Perform Python actions here. */
result = CallSomeFunction();
```

```
/* evaluate result or handle exception */

/* Release the thread. No Python API allowed beyond this
PyGILState_Release(gstate);
```

Cautions about fork()

Another important thing to note about threads is their behaviour in the face of the C `fork()` call. On most systems with `fork()`, after a process forks only the thread that issued the fork will exist. This has a concrete impact both on how locks must be handled and on all stored state in CPython’s runtime.

The fact that only the «current» thread remains means any locks held by other threads will never be released. Python solves this for `os.fork()` by acquiring the locks it uses internally before the fork, and releasing them afterwards. In addition, it resets any Lock objects in the child. When extending or embedding Python, there is no way to inform Python of additional (non-Python) locks that need to be acquired before or reset after a fork. OS facilities such as `pthread_atfork()` would need to be used to accomplish the same thing. Additionally, when extending or embedding Python, calling `fork()` directly rather than through `os.fork()` (and returning to or calling into Python) may result in a deadlock by one of Python’s internal locks being held by a thread that is defunct after the fork. `PyOS_AfterFork_Child()` tries to reset the necessary locks, but is not always able to.

The fact that all other threads go away also means that CPython’s runtime state there must be cleaned up properly, which `os.fork()` does. This means finalizing all other `PyThreadState` objects belonging to the current interpreter and all other `PyInterpreterState` objects. Due to this and the special nature of the «main» interpreter, `fork()` should only be called in that interpreter’s «main» thread, where the CPython global runtime was originally initialized. The only exception is if `exec()` will be called immediately after.

High-level APIs

These are the most commonly used types and functions when writing multi-threaded C extensions.

type PyThreadState

Part of the Limited API (as an opaque struct).
This data structure represents the state of a single thread. The only public data member is:

PyInterpreterState *interp

This thread's interpreter state.

void PyEval_InitThreads()

Part of the Stable ABI.
Deprecated function which does nothing.

In Python 3.6 and older, this function created the GIL if it didn't exist.

Distinto en la versión 3.9: The function now does nothing.

Distinto en la versión 3.7: This function is now called by **`Py_Initialize()`**, so you don't have to call it yourself anymore.

Distinto en la versión 3.2: This function cannot be called before **`Py_Initialize()`** anymore.

Obsoleto desde la versión 3.9.

PyThreadState *PyEval_SaveThread()

Part of the Stable ABI.
Detach the attached thread state and return it. The thread will have no thread state upon returning.

void PyEval_RestoreThread(PyThreadState *tstate)

Part of the Stable ABI.
Set the attached thread state to *tstate*. The passed thread state

should not be attached, otherwise deadlock ensues. *tstate* will be attached upon returning.

Nota

Calling this function from a thread when the runtime is finalizing will hang the thread until the program exits, even if the thread was not created by Python. Refer to Cautions regarding runtime finalization for more details.

Distinto en la versión 3.14: Hangs the current thread, rather than terminating it, if called while the interpreter is finalizing.

PyThreadState *PyThreadState_Get()

Part of the Stable ABI.
Return the attached thread state. If the thread has no attached thread state, (such as when inside of **`Py_BEGIN_ALLOW_THREADS`** block), then this issues a fatal error (so that the caller needn't check for `NULL`).

See also **`PyThreadState_GetUnchecked()`**.

PyThreadState *PyThreadState_GetUnchecked()

Similar to **`PyThreadState_Get()`**, but don't kill the process with a fatal error if it is NULL. The caller is responsible to check if the result is NULL.

Added in version 3.13: In Python 3.5 to 3.12, the function was private and known as `_PyThreadState_UncheckedGet()`.

PyThreadState *PyThreadState_Swap(PyThreadState *tstate)

Part of the Stable ABI.
Set the attached thread state to *tstate*, and return the thread state that was attached prior to calling.

This function is safe to call without an attached thread state; it will simply return `NULL` indicating that there was no prior thread state.

Ver también

`PyEval_ReleaseThread()`

Nota

Similar to `PyGILState_Ensure()`, this function will hang the thread if the runtime is finalizing.

GIL-state APIs

The following functions use thread-local storage, and are not compatible with sub-interpreters:

type PyGILState_STATE

Part of the Stable ABI.
The type of the value returned by `PyGILState_Ensure()` and passed to `PyGILState_Release()`.

enumerator PyGILState_LOCKED

The GIL was already held when `PyGILState_Ensure()` was called.

enumerator PyGILState_UNLOCKED

The GIL was not held when `PyGILState_Ensure()` was called.

PyGILState_STATE PyGILState_Ensure()

Part of the Stable ABI.
Ensure that the current thread is ready to call the Python C API regardless of the current state of Python, or of the attached thread state. This may be called as many times as desired by a thread as long as each call is matched with a call to `PyGILState_Release()`. In general, other thread-related APIs may be used between `PyGILState_Ensure()`

and `PyGILState_Release()` calls as long as the thread state is restored to its previous state before the Release(). For example, normal usage of the `Py_BEGIN_ALLOW_THREADS` and `Py_END_ALLOW_THREADS` macros is acceptable.

The return value is an opaque «handle» to the attached thread state when `PyGILState_Ensure()` was called, and must be passed to `PyGILState_Release()` to ensure Python is left in the same state. Even though recursive calls are allowed, these handles *cannot* be shared - each unique call to `PyGILState_Ensure()` must save the handle for its call to `PyGILState_Release()`.

When the function returns, there will be an attached thread state and the thread will be able to call arbitrary Python code. Failure is a fatal error.

Advertencia

Calling this function when the runtime is finalizing is unsafe. Doing so will either hang the thread until the program ends, or fully crash the interpreter in rare cases. Refer to Cautions regarding runtime finalization for more details.

Distinto en la versión 3.14: Hangs the current thread, rather than terminating it, if called while the interpreter is finalizing.

void PyGILState_Release(PyGILState_STATE)

Part of the Stable ABI.
Release any resources previously acquired. After this call, Python's state will be the same as it was prior to the corresponding `PyGILState_Ensure()` call (but generally this state will be unknown to the caller, hence the use of the GILState API).

Every call to `PyGILState_Ensure()` must be matched by a call to `PyGILState_Release()` on the same thread.

PyThreadState *PyGILState_GetThisThreadState()

Part of the Stable ABI.

Get the attached thread state for this thread. May return `NULL` if no GILState API has been used on the current thread. Note that the main thread always has such a thread-state, even if no auto-thread-state call has been made on the main thread. This is mainly a helper/diagnostic function.

Nota

This function may return non-`NULL` even when the thread state is detached. Prefer **`PyThreadState_Get()`** or **`PyThreadState_GetUnchecked()`** for most cases.

Ver también

`PyThreadState_Get()`

int PyGILState_Check()

Return `1` if the current thread is holding the GIL and `0` otherwise. This function can be called from any thread at any time. Only if it has had its thread state initialized via **`PyGILState_Ensure()`** will it return `1`. This is mainly a helper/diagnostic function. It can be useful for example in callback contexts or memory allocation functions when knowing that the GIL is locked can allow the caller to perform sensitive actions or otherwise behave differently.

Nota

If the current Python process has ever created a subinterpreter, this function will *always* return `1`. Prefer **`PyThreadState_GetUnchecked()`** for most cases.

Added in version 3.4.

Low-level APIs

PyThreadState *PyThreadState_New(PyInterpreterState *interp)

Part of the Stable ABI.
Create a new thread state object belonging to the given interpreter object. An attached thread state is not needed.

void PyThreadState_Clear(PyThreadState *tstate)

Part of the Stable ABI.
Reset all information in a thread state object. *tstate* must be attached

Distinto en la versión 3.9: This function now calls the **`PyThreadState.on_delete`** callback. Previously, that happened in **`PyThreadState_Delete()`**.

Distinto en la versión 3.13: The **`PyThreadState.on_delete`** callback was removed.

void PyThreadState_Delete(PyThreadState *tstate)

Part of the Stable ABI.
Destroy a thread state object. *tstate* should not be attached to any thread. *tstate* must have been reset with a previous call to **`PyThreadState_Clear()`**.

void PyThreadState_DeleteCurrent(void)

Detach the attached thread state (which must have been reset with a previous call to **`PyThreadState_Clear()`**) and then destroy it.

No thread state will be attached upon returning.

PyFrameObject *PyThreadState_GetFrame(PyThreadState *tstate)

Part of the Stable ABI since version 3.10.
Get the current frame of the Python thread state *tstate*.

Return a strong reference. Return `NULL` if no frame is currently executing.

See also **`PyEval_GetFrame()`**.

tstate must not be `NULL`, and must be attached.

Added in version 3.9.

uint64_t PyThreadState_GetID(PyThreadState *tstate)

Part of the Stable ABI since version 3.10.
Get the unique thread state identifier of the Python thread state *tstate*.

tstate must not be `NULL`, and must be attached.

Added in version 3.9.

PyInterpreterState *PyThreadState_GetInterpreter(PyThreadState *tstate)

Part of the Stable ABI since version 3.10.
Get the interpreter of the Python thread state *tstate*.

tstate must not be `NULL`, and must be attached.

Added in version 3.9.

void PyThreadState_EnterTracing(PyThreadState *tstate)

Suspend tracing and profiling in the Python thread state *tstate*.

Resume them using the **`PyThreadState_LeaveTracing()`** function.

Added in version 3.11.

void PyThreadState_LeaveTracing(PyThreadState *tstate)

Resume tracing and profiling in the Python thread state *tstate* suspended by the **`PyThreadState_EnterTracing()`** function.

See also **`PyEval_SetTrace()`** and **`PyEval_SetProfile()`** functions.

Added in version 3.11.

int PyUnstable_ThreadState_SetStackProtection(PyThreadState *tstate, void *stack_start_addr, size_t stack_size)

> *This is Unstable API. It may change without warning in minor releases.*

Set the stack protection start address and stack protection size of a Python thread state.

On success, return `0`. On failure, set an exception and return `-1`.

CPython implements recursion control for C code by raising **`RecursionError`** when it notices that the machine execution stack is close to overflow. See for example the **`Py_EnterRecursiveCall()`** function. For this, it needs to know the location of the current thread's stack, which it normally gets from the operating system. When the stack is changed, for example using context switching techniques like the Boost library's `boost::context`, you must call **`PyUnstable_ThreadState_SetStackProtection()`** to inform CPython of the change.

Call **`PyUnstable_ThreadState_SetStackProtection()`** either before or after changing the stack. Do not call any other Python C API between the call and the stack change.

See **`PyUnstable_ThreadState_ResetStackProtection()`** for undoing this operation.

Added in version 3.15.

void PyUnstable_ThreadState_ResetStackProtection(PyThreadState *tstate)

This is Unstable API. It may change without warning in minor releases.

Reset the stack protection start address and stack protection size of a Python thread state to the operating system defaults.

See **`PyUnstable_ThreadState_SetStackProtection()`** for an explanation.

Added in version 3.15.

PyObject *PyThreadState_GetDict()

Return value: Borrowed reference. Part of the Stable ABI.
Return a dictionary in which extensions can store thread-specific state information. Each extension should use a unique key to use to store state in the dictionary. It is okay to call this function when no thread state is attached. If this function returns `NULL`, no exception has been raised and the caller should assume no thread state is attached.

void PyEval_AcquireThread(PyThreadState *tstate)

Part of the Stable ABI.
Attach *tstate* to the current thread, which must not be `NULL` or already attached.

The calling thread must not already have an attached thread state.

Nota

Calling this function from a thread when the runtime is finalizing will hang the thread until the program exits, even if the thread was not created by Python. Refer to Cautions regarding runtime finalization for more details.

Distinto en la versión 3.8: Updated to be consistent with **`PyEval_RestoreThread()`**, **`Py_END_ALLOW_THREADS()`**, and **`PyGILState_Ensure()`**, and terminate the current thread if called while the interpreter is finalizing.

Distinto en la versión 3.14: Hangs the current thread, rather than terminating it, if called while the interpreter is finalizing.

`PyEval_RestoreThread()` is a higher-level function which is always available (even when threads have not been initialized).

void PyEval_ReleaseThread(PyThreadState *tstate)

Part of the Stable ABI.
Detach the attached thread state. The *tstate* argument, which must not be `NULL`, is only used to check that it represents the attached thread state — if it isn't, a fatal error is reported.

`PyEval_SaveThread()` is a higher-level function which is always available (even when threads have not been initialized).

Asynchronous notifications

A mechanism is provided to make asynchronous notifications to the main interpreter thread. These notifications take the form of a function pointer and a void pointer argument.

int Py_AddPendingCall(int (*func)(void*), void *arg)

Part of the Stable ABI.
Schedule a function to be called from the main interpreter thread. On success, `0` is returned and *func* is queued for being called in the main thread. On failure, `-1` is returned without setting any exception.

When successfully queued, *func* will be *eventually* called from the main interpreter thread with the argument *arg*. It will be called asynchronously with respect to normally running

Python code, but with both these conditions met:

- on a bytecode boundary;
- with the main thread holding an attached thread state (*func* can therefore use the full C API).

func must return `0` on success, or `-1` on failure with an exception set. *func* won't be interrupted to perform another asynchronous notification recursively, but it can still be interrupted to switch threads if the thread state is detached.

This function doesn't need an attached thread state. However, to call this function in a subinterpreter, the caller must have an attached thread state. Otherwise, the function *func* can be scheduled to be called from the wrong interpreter.

Advertencia

This is a low-level function, only useful for very special cases. There is no guarantee that *func* will be called as quick as possible. If the main thread is busy executing a system call, *func* won't be called before the system call returns. This function is generally **not** suitable for calling Python code from arbitrary C threads. Instead, use the PyGILState API.

Added in version 3.1.

Distinto en la versión 3.9: If this function is called in a subinterpreter, the function *func* is now scheduled to be called from the subinterpreter, rather than being called from the main interpreter. Each subinterpreter now has its own list of scheduled calls.

Distinto en la versión 3.12: This function now always schedules *func* to be run in the main interpreter.

int Py_MakePendingCalls(void)

Part of the Stable ABI.
Execute all pending calls. This is usually executed

automatically by the interpreter.

This function returns `0` on success, and returns `-1` with an exception set on failure.

If this is not called in the main thread of the main interpreter, this function does nothing and returns `0`. The caller must hold an attached thread state.

Added in version 3.1.

Distinto en la versión 3.12: This function only runs pending calls in the main interpreter.

int PyThreadState_SetAsyncExc(unsigned long id, PyObject *exc)

Part of the Stable ABI.
Asynchronously raise an exception in a thread. The *id* argument is the thread id of the target thread; *exc* is the exception object to be raised. This function does not steal any references to *exc*. To prevent naive misuse, you must write your own C extension to call this. Must be called with an attached thread state. Returns the number of thread states modified; this is normally one, but will be zero if the thread id isn't found. If *exc* is `NULL`, the pending exception (if any) for the thread is cleared. This raises no exceptions.

Distinto en la versión 3.7: The type of the *id* parameter changed from long to unsigned long.

Operating system thread APIs

PYTHREAD_INVALID_THREAD_ID

Sentinel value for an invalid thread ID.

This is currently equivalent to `(unsigned long)-1`.

unsigned long PyThread_start_new_thread(void (*func)(void*), void *arg)

Part of the Stable ABI.
Start function *func* in a new thread with argument *arg*. The resulting thread is not intended to be joined.

func must not be `NULL`, but *arg* may be `NULL`.

On success, this function returns the identifier of the new thread; on failure, this returns **`PYTHREAD_INVALID_THREAD_ID`**.

The caller does not need to hold an attached thread state.

unsigned long PyThread_get_thread_ident(void)

Part of the Stable ABI.
Return the identifier of the current thread, which will never be zero.

This function cannot fail, and the caller does not need to hold an attached thread state.

Ver también

`threading.get_ident()`

PyObject *PyThread_GetInfo(void)

Part of the Stable ABI since version 3.3.
Get general information about the current thread in the form of a struct sequence object. This information is accessible as **`sys.thread_info`** in Python.

On success, this returns a new strong reference to the thread information; on failure, this returns `NULL` with an exception set.

The caller must hold an attached thread state.

PY_HAVE_THREAD_NATIVE_ID

This macro is defined when the system supports native thread IDs.

unsigned long PyThread_get_thread_native_id(void)

Part of the Stable ABI on platforms with native thread IDs.
Get the native identifier of the current thread as it was assigned by the operating system's kernel, which will never be less than zero.

This function is only available when `PY_HAVE_THREAD_NATIVE_ID` is defined.

This function cannot fail, and the caller does not need to hold an attached thread state.

Ver también

`threading.get_native_id()`

void PyThread_exit_thread(void)

Part of the Stable ABI.
Terminate the current thread. This function is generally considered unsafe and should be avoided. It is kept solely for backwards compatibility.

This function is only safe to call if all functions in the full call stack are written to safely allow it.

Advertencia

If the current system uses POSIX threads (also known as «pthreads»), this calls *pthread_exit(3)* *[https://manpages.debian.org/pthread_exit(3)]*, which attempts to unwind the stack and call C++ destructors on some libc implementations. However, if a `noexcept` function is reached, it may terminate the process. Other systems, such as macOS, do unwinding.

On Windows, this function calls `_endthreadex()`, which kills the thread without calling C++ destructors.

In any case, there is a risk of corruption on the thread's stack.

Obsoleto desde la versión 3.14.

void PyThread_init_thread(void)

Part of the Stable ABI.
Initialize `PyThread*` APIs. Python executes this function automatically, so there's little need to call it from an extension module.

int PyThread_set_stacksize(size_t size)

Part of the Stable ABI.
Set the stack size of the current thread to *size* bytes.

This function returns `0` on success, `-1` if *size* is invalid, or `-2` if the system does not support changing the stack size. This function does not set exceptions.

The caller does not need to hold an attached thread state.

size_t PyThread_get_stacksize(void)

Part of the Stable ABI.
Return the stack size of the current thread in bytes, or `0` if the system's default stack size is in use.

The caller does not need to hold an attached thread state.

Synchronization primitives

The C-API provides a basic mutual exclusion lock.

type PyMutex

A mutual exclusion lock. The **`PyMutex`** should be initialized to zero to represent the unlocked state. For example:

```
PyMutex mutex = {0};
```

Instances of **`PyMutex`** should not be copied or moved. Both the contents and address of a **`PyMutex`** are meaningful, and it must remain at a fixed, writable location in memory.

Nota

A **`PyMutex`** currently occupies one byte, but the size should be considered unstable. The size may change in future Python releases without a deprecation period.

Added in version 3.13.

void PyMutex_Lock(PyMutex *m)

Thread safety: Atomic.
Lock mutex *m*. If another thread has already locked it, the calling thread will block until the mutex is unlocked. While blocked, the thread will temporarily detach the thread state if one exists.

Added in version 3.13.

void PyMutex_Unlock(PyMutex *m)

Thread safety: Atomic.
Unlock mutex *m*. The mutex must be locked — otherwise, the function will issue a fatal error.

Added in version 3.13.

int PyMutex_IsLocked(PyMutex *m)

Thread safety: Atomic.

Returns non-zero if the mutex *m* is currently locked, zero otherwise.

Nota

This function is intended for use in assertions and debugging only and should not be used to make concurrency control decisions, as the lock state may change immediately after the check.

Added in version 3.14.

Python critical section API

The critical section API provides a deadlock avoidance layer on top of per-object locks for free-threaded CPython. They are intended to replace reliance on the global interpreter lock, and are no-ops in versions of Python with the global interpreter lock.

Critical sections are intended to be used for custom types implemented in C-API extensions. They should generally not be used with built-in types like `list` and `dict` because their public C-APIs already use critical sections internally, with the notable exception of `PyDict_Next()`, which requires critical section to be acquired externally.

Critical sections avoid deadlocks by implicitly suspending active critical sections, hence, they do not provide exclusive access such as provided by traditional locks like `PyMutex`. When a critical section is started, the per-object lock for the object is acquired. If the code executed inside the critical section calls C-API functions then it can suspend the critical section thereby releasing the per-object lock, so other threads can acquire the per-object lock for the same object.

Variants that accept `PyMutex` pointers rather than Python objects are also available. Use these variants to start a critical section in a situation where there is no `PyObject` – for example, when working with a C type that does not extend or wrap `PyObject` but still needs to call into the C API in a manner that might lead to deadlocks.

The functions and structs used by the macros are exposed for cases where C macros are not available. They should only be used as in the given macro expansions. Note that the sizes and contents of the structures may change in future Python versions.

Nota

Operations that need to lock two objects at once must use `Py_BEGIN_CRITICAL_SECTION2`. You *cannot* use nested critical sections to lock more than one object at once, because the inner critical section may suspend the outer critical sections. This API does not provide a way to lock more than two objects at once.

Example usage:

```
static PyObject *
set_field(MyObject *self, PyObject *value)
{
   Py_BEGIN_CRITICAL_SECTION(self);
   Py_SETREF(self->field, Py_XNewRef(value));
   Py_END_CRITICAL_SECTION();
   Py_RETURN_NONE;
}
```

In the above example, `Py_SETREF` calls `Py_DECREF`, which can call arbitrary code through an object's deallocation function. The critical section API avoids potential deadlocks due to reentrancy and lock ordering by allowing the runtime to temporarily suspend the critical section if the code triggered by the finalizer blocks and calls `PyEval_SaveThread()`.

Py_BEGIN_CRITICAL_SECTION(op)

Acquires the per-object lock for the object *op* and begins a critical section.

In the free-threaded build, this macro expands to:

```
{
    PyCriticalSection _py_cs;
    PyCriticalSection_Begin(&_py_cs, (PyObject*)(op
```

In the default build, this macro expands to `{`.

Added in version 3.13.

Py_BEGIN_CRITICAL_SECTION_MUTEX(m)

Locks the mutex *m* and begins a critical section.

In the free-threaded build, this macro expands to:

```
{
     PyCriticalSection _py_cs;
     PyCriticalSection_BeginMutex(&_py_cs, m)
```

Note that unlike **`Py_BEGIN_CRITICAL_SECTION`**, there is no cast for the argument of the macro - it must be a **`PyMutex`** pointer.

On the default build, this macro expands to `{`.

Added in version 3.14.

Py_END_CRITICAL_SECTION()

Ends the critical section and releases the per-object lock.

In the free-threaded build, this macro expands to:

```
    PyCriticalSection_End(&_py_cs);
}
```

In the default build, this macro expands to `}`.

Added in version 3.13.

Py_BEGIN_CRITICAL_SECTION2(a, b)

Acquires the per-object locks for the objects *a* and *b* and begins a critical section. The locks are acquired in a consistent order (lowest address first) to avoid lock ordering deadlocks.

In the free-threaded build, this macro expands to:

```
{
    PyCriticalSection2 _py_cs2;
    PyCriticalSection2_Begin(&_py_cs2, (PyObject*)(
```

In the default build, this macro expands to `{`.

Added in version 3.13.

Py_BEGIN_CRITICAL_SECTION2_MUTEX(m1, m2)

Locks the mutexes *m1* and *m2* and begins a critical section.

In the free-threaded build, this macro expands to:

```
{
     PyCriticalSection2 _py_cs2;
     PyCriticalSection2_BeginMutex(&_py_cs2, m1, m2
```

Note that unlike **`Py_BEGIN_CRITICAL_SECTION2`**, there is no cast for the arguments of the macro - they must be **`PyMutex`** pointers.

On the default build, this macro expands to `{`.

Added in version 3.14.

Py_END_CRITICAL_SECTION2()

Ends the critical section and releases the per-object locks.

In the free-threaded build, this macro expands to:

```
    PyCriticalSection2_End(&_py_cs2);
}
```

In the default build, this macro expands to `}`.

Added in version 3.13.

Legacy locking APIs

These APIs are obsolete since Python 3.13 with the introduction of **`PyMutex`**.

type PyThread_type_lock

A pointer to a mutual exclusion lock.

type PyLockStatus

The result of acquiring a lock with a timeout.

enumerator PY_LOCK_FAILURE

Failed to acquire the lock.

enumerator PY_LOCK_ACQUIRED

The lock was successfully acquired.

enumerator PY_LOCK_INTR

The lock was interrupted by a signal.

PyThread_type_lock PyThread_allocate_lock(void)

Part of the Stable ABI.
Allocate a new lock.

On success, this function returns a lock; on failure, this function returns `0` without an exception set.

The caller does not need to hold an attached thread state.

void PyThread_free_lock(PyThread_type_lock lock)

Part of the Stable ABI.
Destroy *lock*. The lock should not be held by any thread when calling this.

The caller does not need to hold an attached thread state.

PyLockStatus PyThread_acquire_lock_timed(PyThread_type_lock lock, long long microseconds, int intr_flag)

Part of the Stable ABI.
Acquire *lock* with a timeout.

This will wait for *microseconds* microseconds to acquire the lock. If the timeout expires, this function returns **`PY_LOCK_FAILURE`**. If *microseconds* is `-1`, this will wait indefinitely until the lock has been released.

If *intr_flag* is `1`, acquiring the lock may be interrupted by a signal, in which case this function returns **`PY_LOCK_INTR`**. Upon interruption, it's generally expected that the caller makes a call to **`Py_MakePendingCalls()`** to propagate an exception to Python code.

If the lock is successfully acquired, this function returns **`PY_LOCK_ACQUIRED`**.

The caller does not need to hold an attached thread state.

int PyThread_acquire_lock(PyThread_type_lock lock, int waitflag)

Part of the Stable ABI.
Acquire *lock*.

If *waitflag* is `1` and another thread currently holds the lock, this function will wait until the lock can be acquired and will always return `1`.

If *waitflag* is `0` and another thread holds the lock, this function will not wait and instead return `0`. If the lock is not held by any other thread, then this function will acquire it and return `1`.

Unlike **`PyThread_acquire_lock_timed()`**, acquiring the lock cannot be interrupted by a signal.

The caller does not need to hold an attached thread state.

int PyThread_release_lock(PyThread_type_lock lock)

Part of the Stable ABI.

Release *lock*. If *lock* is not held, then this function issues a fatal error.

The caller does not need to hold an attached thread state.

Thread-local storage support

The Python interpreter provides low-level support for thread-local storage (TLS) which wraps the underlying native TLS implementation to support the Python-level thread-local storage API (**`threading.local`**). The CPython C level APIs are similar to those offered by pthreads and Windows: use a thread key and functions to associate a void* value per thread.

A thread state does *not* need to be attached when calling these functions; they supply their own locking.

Note that `Python.h` does not include the declaration of the TLS APIs, you need to include `pythread.h` to use thread-local storage.

Nota

None of these API functions handle memory management on behalf of the void* values. You need to allocate and deallocate them yourself. If the void* values happen to be PyObject*, these functions don't do refcount operations on them either.

Thread-specific storage API

The thread-specific storage (TSS) API was introduced to supersede the use of the existing TLS API within the CPython interpreter. This API uses a new type **`Py_tss_t`** instead of int to represent thread keys.

Added in version 3.7.

Ver también

«A New C-API for Thread-Local Storage in CPython» (**PEP 539** [https://peps.python.org/pep-0539/])

type Py_tss_t

This data structure represents the state of a thread key, the definition of which may depend on the underlying TLS implementation, and it has an internal field representing the key's initialization state. There are no public members in this structure.

When Py_LIMITED_API is not defined, static allocation of this type by `Py_tss_NEEDS_INIT` is allowed.

Py_tss_NEEDS_INIT

This macro expands to the initializer for `Py_tss_t` variables. Note that this macro won't be defined with Py_LIMITED_API.

Dynamic allocation

Dynamic allocation of the `Py_tss_t`, required in extension modules built with Py_LIMITED_API, where static allocation of this type is not possible due to its implementation being opaque at build time.

Py_tss_t *PyThread_tss_alloc()

Part of the Stable ABI since version 3.7.
Return a value which is the same state as a value initialized with `Py_tss_NEEDS_INIT`, or `NULL` in the case of dynamic allocation failure.

void PyThread_tss_free(Py_tss_t *key)

Part of the Stable ABI since version 3.7.
Free the given *key* allocated by `PyThread_tss_alloc()`, after first calling `PyThread_tss_delete()` to ensure any associated thread locals have been unassigned. This is a no-op if the *key* argument is `NULL`.

Nota

A freed key becomes a dangling pointer. You should reset the key to `NULL`.

Methods

The parameter *key* of these functions must not be `NULL`. Moreover, the behaviors of **`PyThread_tss_set()`** and **`PyThread_tss_get()`** are undefined if the given **`Py_tss_t`** has not been initialized by **`PyThread_tss_create()`**.

int PyThread_tss_is_created(Py_tss_t *key)

Part of the Stable ABI since version 3.7.
Return a non-zero value if the given **`Py_tss_t`** has been initialized by **`PyThread_tss_create()`**.

int PyThread_tss_create(Py_tss_t *key)

Part of the Stable ABI since version 3.7.
Return a zero value on successful initialization of a TSS key. The behavior is undefined if the value pointed to by the *key* argument is not initialized by **`Py_tss_NEEDS_INIT`**. This function can be called repeatedly on the same key – calling it on an already initialized key is a no-op and immediately returns success.

void PyThread_tss_delete(Py_tss_t *key)

Part of the Stable ABI since version 3.7.
Destroy a TSS key to forget the values associated with the key across all threads, and change the key's initialization state to uninitialized. A destroyed key is able to be initialized again by **`PyThread_tss_create()`**. This function can be called repeatedly on the same key – calling it on an already destroyed key is a no-op.

int PyThread_tss_set(Py_tss_t *key, void *value)

Part of the Stable ABI since version 3.7.
Return a zero value to indicate successfully associating a void* value with a TSS key in the current thread. Each thread

has a distinct mapping of the key to a void* value.

void *PyThread_tss_get(Py_tss_t *key)

Part of the Stable ABI since version 3.7.
Return the void* value associated with a TSS key in the current thread. This returns `NULL` if no value is associated with the key in the current thread.

Legacy APIs

Obsoleto desde la versión 3.7: This API is superseded by the thread-specific storage (TSS) API.

Nota

This version of the API does not support platforms where the native TLS key is defined in a way that cannot be safely cast to `int`. On such platforms, **`PyThread_create_key()`** will return immediately with a failure status, and the other TLS functions will all be no-ops on such platforms.

Due to the compatibility problem noted above, this version of the API should not be used in new code.

int PyThread_create_key()

Part of the Stable ABI.

void PyThread_delete_key(int key)

Part of the Stable ABI.

int PyThread_set_key_value(int key, void *value)

Part of the Stable ABI.

void *PyThread_get_key_value(int key)

Part of the Stable ABI.

void PyThread_delete_key_value(int key)

Part of the Stable ABI.

void PyThread_ReInitTLS()

Part of the Stable ABI.

Multiple interpreters in a Python process

While in most uses, you will only embed a single Python interpreter, there are cases where you need to create several independent interpreters in the same process and perhaps even in the same thread. Sub-interpreters allow you to do that.

The «main» interpreter is the first one created when the runtime initializes. It is usually the only Python interpreter in a process. Unlike sub-interpreters, the main interpreter has unique process-global responsibilities like signal handling. It is also responsible for execution during runtime initialization and is usually the active interpreter during runtime finalization. The **`PyInterpreterState_Main()`** function returns a pointer to its state.

You can switch between sub-interpreters using the **`PyThreadState_Swap()`** function. You can create and destroy them using the following functions:

type PyInterpreterConfig

Structure containing most parameters to configure a sub-interpreter. Its values are used only in **`Py_NewInterpreterFromConfig()`** and never modified by the runtime.

Added in version 3.12.

Structure fields:

int use_main_obmalloc

If this is `0` then the sub-interpreter will use its own «object» allocator state. Otherwise it will use (share) the main interpreter's.

If this is `0` then `check_multi_interp_extensions` must be `1` (non-zero). If this is `1` then `gil` must not be `PyInterpreterConfig_OWN_GIL`.

int allow_fork

If this is `0` then the runtime will not support forking the process in any thread where the sub-interpreter is currently active. Otherwise fork is unrestricted.

Note that the `subprocess` module still works when fork is disallowed.

int allow_exec

If this is `0` then the runtime will not support replacing the current process via exec (e.g. `os.execv()`) in any thread where the sub-interpreter is currently active. Otherwise exec is unrestricted.

Note that the `subprocess` module still works when exec is disallowed.

int allow_threads

If this is `0` then the sub-interpreter's `threading` module won't create threads. Otherwise threads are allowed.

int allow_daemon_threads

If this is `0` then the sub-interpreter's `threading` module won't create daemon threads. Otherwise daemon threads are allowed (as long as `allow_threads` is non-zero).

int check_multi_interp_extensions

If this is `0` then all extension modules may be imported, including legacy (single-phase init) modules, in any thread where the sub-interpreter is currently active. Otherwise only multi-phase init extension modules (see **PEP 489** [https://peps.python.org/pep-0489/]) may be imported. (Also see **`Py_mod_multiple_interpreters`**.)

This must be `1` (non-zero) if **`use_main_obmalloc`** is `0`.

int gil

This determines the operation of the GIL for the sub-interpreter. It may be one of the following:

PyInterpreterConfig_DEFAULT_GIL

Use the default selection (**`PyInterpreterConfig_SHARED_GIL`**).

PyInterpreterConfig_SHARED_GIL

Use (share) the main interpreter's GIL.

PyInterpreterConfig_OWN_GIL

Use the sub-interpreter's own GIL.

If this is **`PyInterpreterConfig_OWN_GIL`** then **`PyInterpreterConfig.use_main_obmalloc`** must be `0`.

PyStatus Py_NewInterpreterFromConfig(PyThreadState **tstate_p, const PyInterpreterConfig *config)

Create a new sub-interpreter. This is an (almost) totally separate environment for the execution of Python code. In particular, the new interpreter has separate, independent versions of all imported modules, including the fundamental modules **`builtins`**, **`__main__`** and **`sys`**. The table of loaded modules (`sys.modules`) and the module search path (`sys.path`) are also separate. The new environment has no `sys.argv` variable. It has new standard I/O stream file objects `sys.stdin`, `sys.stdout` and `sys.stderr` (however these refer to the same underlying file descriptors).

The given *config* controls the options with which the interpreter is initialized.

Upon success, *tstate_p* will be set to the first thread state created in the new sub-interpreter. This thread state is attached. Note that no actual thread is created; see the discussion of thread states below. If creation of the new interpreter is unsuccessful, *tstate_p* is set to `NULL`; no exception is set since the exception state is stored in the attached thread state, which might not exist.

Like all other Python/C API functions, an attached thread state must be present before calling this function, but it might be detached upon returning. On success, the returned thread state will be attached. If the sub-interpreter is created with its own GIL then the attached thread state of the calling interpreter will be detached. When the function returns, the new interpreter's thread state will be attached to the current thread and the previous interpreter's attached thread state will remain detached.

Added in version 3.12.

Sub-interpreters are most effective when isolated from each other, with certain functionality restricted:

```
PyInterpreterConfig config = {
    .use_main_obmalloc = 0,
    .allow_fork = 0,
    .allow_exec = 0,
    .allow_threads = 1,
    .allow_daemon_threads = 0,
    .check_multi_interp_extensions = 1,
    .gil = PyInterpreterConfig_OWN_GIL,
};
PyThreadState *tstate = NULL;
PyStatus status = Py_NewInterpreterFromConfig(&tsta
if (PyStatus_Exception(status)) {
    Py_ExitStatusException(status);
}
```

Note that the config is used only briefly and does not get modified. During initialization the config's values are converted into various **`PyInterpreterState`** values. A read-only copy of the config may be stored internally on the **`PyInterpreterState`**.

Extension modules are shared between (sub-)interpreters as follows:

- For modules using multi-phase initialization, e.g. **`PyModule_FromDefAndSpec()`**, a separate module object is created and initialized for each interpreter. Only C-level static and global variables are shared between these module objects.

- For modules using legacy single-phase initialization, e.g. **`PyModule_Create()`**, the first time a particular extension is imported, it is initialized normally, and a (shallow) copy of its module's dictionary is squirreled away. When the same extension is imported by another (sub-)interpreter, a new module is initialized and filled with the contents of this copy; the extension's `init` function is not called. Objects in the module's dictionary thus end up shared across (sub-)interpreters, which might cause unwanted behavior (see Bugs and caveats below).

  Note that this is different from what happens when an extension is imported after the interpreter has been completely re-initialized by calling **`Py_FinalizeEx()`** and **`Py_Initialize()`**; in that case, the extension's `initmodule` function *is* called again. As with multi-phase initialization, this means that only C-level static and global variables are shared between these modules.

PyThreadState *Py_NewInterpreter(void)

Part of the Stable ABI.

Create a new sub-interpreter. This is essentially just a wrapper around `Py_NewInterpreterFromConfig()` with a config that preserves the existing behavior. The result is an unisolated sub-interpreter that shares the main interpreter's GIL, allows fork/exec, allows daemon threads, and allows single-phase init modules.

void Py_EndInterpreter(PyThreadState *tstate)

Part of the Stable ABI.
Destroy the (sub-)interpreter represented by the given thread state. The given thread state must be attached. When the call returns, there will be no attached thread state. All thread states associated with this interpreter are destroyed.

`Py_FinalizeEx()` will destroy all sub-interpreters that haven't been explicitly destroyed at that point.

A per-interpreter GIL

Added in version 3.12.

Using `Py_NewInterpreterFromConfig()` you can create a sub-interpreter that is completely isolated from other interpreters, including having its own GIL. The most important benefit of this isolation is that such an interpreter can execute Python code without being blocked by other interpreters or blocking any others. Thus a single Python process can truly take advantage of multiple CPU cores when running Python code. The isolation also encourages a different approach to concurrency than that of just using threads. (See **PEP 554** [https://peps.python.org/pep-0554/] and **PEP 684** [https://peps.python.org/pep-0684/].)

Using an isolated interpreter requires vigilance in preserving that isolation. That especially means not sharing any objects or mutable state without guarantees about thread-safety. Even objects that are otherwise immutable (e.g. `None`, `(1, 5)`) can't normally be shared because of the refcount. One simple but less-efficient approach around this is to use a global lock around all use of some state (or object). Alternately, effectively immutable objects (like integers or strings) can be made safe in spite of their refcounts by making them immortal. In fact, this has been done for the builtin singletons, small integers, and a number of other builtin objects.

If you preserve isolation then you will have access to proper multi-core computing without the complications that come with free-threading. Failure to preserve isolation will expose you to the full consequences of free-threading, including races and hard-to-debug crashes.

Aside from that, one of the main challenges of using multiple isolated interpreters is how to communicate between them safely (not break isolation) and efficiently. The runtime and stdlib do not provide any standard approach to this yet. A future stdlib module would help mitigate the effort of preserving isolation and expose effective tools for communicating (and sharing) data between interpreters.

Bugs and caveats

Because sub-interpreters (and the main interpreter) are part of the same process, the insulation between them isn't perfect — for example, using low-level file operations like `os.close()` they can (accidentally or maliciously) affect each other's open files. Because of the way extensions are shared between (sub-)interpreters, some extensions may not work properly; this is especially likely when using single-phase initialization or (static) global variables. It is possible to insert objects created in one sub-interpreter into a namespace of another (sub-)interpreter; this should be avoided if possible.

Special care should be taken to avoid sharing user-defined functions, methods, instances or classes between sub-interpreters, since import operations executed by such objects may affect the wrong (sub-)interpreter's dictionary of loaded modules. It is equally important to avoid sharing objects from which the above are reachable.

Also note that combining this functionality with `PyGILState_*` APIs is delicate, because these APIs assume a bijection between

Python thread states and OS-level threads, an assumption broken by the presence of sub-interpreters. It is highly recommended that you don't switch sub-interpreters between a pair of matching **`PyGILState_Ensure()`** and **`PyGILState_Release()`** calls. Furthermore, extensions (such as **`ctypes`**) using these APIs to allow calling of Python code from non-Python created threads will probably be broken when using sub-interpreters.

High-level APIs

type PyInterpreterState

Part of the Limited API (as an opaque struct).
This data structure represents the state shared by a number of cooperating threads. Threads belonging to the same interpreter share their module administration and a few other internal items. There are no public members in this structure.

Threads belonging to different interpreters initially share nothing, except process state like available memory, open file descriptors and such. The global interpreter lock is also shared by all threads, regardless of to which interpreter they belong.

Distinto en la versión 3.12: **PEP 684** [https://peps.python.org/pep-0684/] introduced the possibility of a per-interpreter GIL. See **`Py_NewInterpreterFromConfig()`**.

PyInterpreterState *PyInterpreterState_Get(void)

Part of the Stable ABI since version 3.9.
Get the current interpreter.

Issue a fatal error if there is no attached thread state. It cannot return NULL.

Added in version 3.9.

int64_t PyInterpreterState_GetID(PyInterpreterState *interp)

Part of the Stable ABI since version 3.7.
Return the interpreter's unique ID. If there was any error in

doing so then `-1` is returned and an error is set.

The caller must have an attached thread state.

Added in version 3.7.

PyObject *PyInterpreterState_GetDict(PyInterpreterState *interp)

Return value: Borrowed reference. Part of the Stable ABI since version 3.8.
Return a dictionary in which interpreter-specific data may be stored. If this function returns `NULL` then no exception has been raised and the caller should assume no interpreter-specific dict is available.

This is not a replacement for **`PyModule_GetState()`**, which extensions should use to store interpreter-specific state information.

The returned dictionary is borrowed from the interpreter and is valid until interpreter shutdown.

Added in version 3.8.

typedef PyObject *(*_PyFrameEvalFunction)(PyThreadState *tstate, _PyInterpreterFrame *frame, int throwflag)

Type of a frame evaluation function.

The *throwflag* parameter is used by the `throw()` method of generators: if non-zero, handle the current exception.

Distinto en la versión 3.9: The function now takes a *tstate* parameter.

Distinto en la versión 3.11: The *frame* parameter changed from `PyFrameObject*` to `_PyInterpreterFrame*`.

_PyFrameEvalFunction
_PyInterpreterState_GetEvalFrameFunc(PyInterpreterState *interp)

Get the frame evaluation function.

See the **PEP 523** [https://peps.python.org/pep-0523/] «Adding a frame evaluation API to CPython».

Added in version 3.9.

void _PyInterpreterState_SetEvalFrameFunc(PyInterpreterState *interp, _PyFrameEvalFunction eval_frame)

Set the frame evaluation function.

See the **PEP 523** [https://peps.python.org/pep-0523/] «Adding a frame evaluation API to CPython».

Added in version 3.9.

Low-level APIs

All of the following functions must be called after **`Py_Initialize()`**.

Distinto en la versión 3.7: **`Py_Initialize()`** now initializes the GIL and sets an attached thread state.

PyInterpreterState *PyInterpreterState_New()

Part of the Stable ABI.
Create a new interpreter state object. An attached thread state is not needed, but may optionally exist if it is necessary to serialize calls to this function.

Raises an auditing event `cpython.PyInterpreterState_New` with no arguments.

void PyInterpreterState_Clear(PyInterpreterState *interp)

Part of the Stable ABI.
Reset all information in an interpreter state object. There must be an attached thread state for the interpreter.

Raises an auditing event `cpython.PyInterpreterState_Clear` with no arguments.

void PyInterpreterState_Delete(PyInterpreterState *interp)

Part of the Stable ABI.

Destroy an interpreter state object. There **should not** be an attached thread state for the target interpreter. The interpreter state must have been reset with a previous call to **`PyInterpreterState_Clear()`**.

Advanced debugger support

These functions are only intended to be used by advanced debugging tools.

PyInterpreterState *PyInterpreterState_Head()

Return the interpreter state object at the head of the list of all such objects.

PyInterpreterState *PyInterpreterState_Main()

Return the main interpreter state object.

PyInterpreterState *PyInterpreterState_Next(PyInterpreterState *interp)

Return the next interpreter state object after *interp* from the list of all such objects.

PyThreadState *PyInterpreterState_ThreadHead(PyInterpreterState *interp)

Return the pointer to the first **`PyThreadState`** object in the list of threads associated with the interpreter *interp*.

PyThreadState *PyThreadState_Next(PyThreadState *tstate)

Return the next thread state object after *tstate* from the list of all such objects belonging to the same **`PyInterpreterState`** object.

Profiling and tracing

The Python interpreter provides some low-level support for attaching profiling and execution tracing facilities. These are used for profiling, debugging, and coverage analysis tools.

This C interface allows the profiling or tracing code to avoid the overhead of calling through Python-level callable objects, making a direct C function call instead. The essential attributes of the facility have not changed; the interface allows trace functions to be installed per-thread, and the basic events reported to the trace function are the same as had been reported to the Python-level trace functions in previous versions.

typedef int (*Py_tracefunc)(PyObject *obj, PyFrameObject *frame, int what, PyObject *arg)

The type of the trace function registered using `PyEval_SetProfile()` and `PyEval_SetTrace()`. The first parameter is the object passed to the registration function as *obj*, *frame* is the frame object to which the event pertains, *what* is one of the constants `PyTrace_CALL`, `PyTrace_EXCEPTION`, `PyTrace_LINE`, `PyTrace_RETURN`, `PyTrace_C_CALL`, `PyTrace_C_EXCEPTION`, `PyTrace_C_RETURN`, or `PyTrace_OPCODE`, and *arg* depends on the value of *what*:

Value of *what*	Meaning of *arg*
`PyTrace_CALL`	Always `Py_None`.
`PyTrace_EXCEPTION`	Exception information as returned by `sys.exc_info()`.
`PyTrace_LINE`	Always `Py_None`.
`PyTrace_RETURN`	Value being returned to the caller, or `NULL` if caused by an exception.
`PyTrace_C_CALL`	Function object being called.
`PyTrace_C_EXCEPTION`	Function object being called.
`PyTrace_C_RETURN`	Function object being called.
`PyTrace_OPCODE`	Always `Py_None`.

int PyTrace_CALL

The value of the *what* parameter to a `Py_tracefunc` function when a new call to a function or method is being reported, or a new entry into a generator. Note that the creation of the iterator for a generator function is not reported as there is no control transfer to the Python bytecode in the corresponding frame.

int PyTrace_EXCEPTION

The value of the *what* parameter to a `Py_tracefunc` function when an exception has been raised. The callback function is called with this value for *what* when after any bytecode is processed after which the exception becomes set within the frame being executed. The effect of this is that as exception propagation causes the Python stack to unwind, the callback is called upon return to each frame as the exception propagates. Only trace functions receive these events; they are not needed by the profiler.

int PyTrace_LINE

The value passed as the *what* parameter to a `Py_tracefunc` function (but not a profiling function) when a line-number event is being reported. It may be disabled for a frame by setting `f_trace_lines` to *0* on that frame.

int PyTrace_RETURN

The value for the *what* parameter to `Py_tracefunc` functions when a call is about to return.

int PyTrace_C_CALL

The value for the *what* parameter to `Py_tracefunc` functions when a C function is about to be called.

int PyTrace_C_EXCEPTION

The value for the *what* parameter to `Py_tracefunc` functions when a C function has raised an exception.

int PyTrace_C_RETURN

The value for the *what* parameter to **`Py_tracefunc`** functions when a C function has returned.

int PyTrace_OPCODE

The value for the *what* parameter to **`Py_tracefunc`** functions (but not profiling functions) when a new opcode is about to be executed. This event is not emitted by default: it must be explicitly requested by setting **`f_trace_opcodes`** to *1* on the frame.

void PyEval_SetProfile(Py_tracefunc func, PyObject *obj)

Set the profiler function to *func*. The *obj* parameter is passed to the function as its first parameter, and may be any Python object, or `NULL`. If the profile function needs to maintain state, using a different value for *obj* for each thread provides a convenient and thread-safe place to store it. The profile function is called for all monitored events except **`PyTrace_LINE`** **`PyTrace_OPCODE`** and **`PyTrace_EXCEPTION`**.

See also the **`sys.setprofile()`** function.

The caller must have an attached thread state.

void PyEval_SetProfileAllThreads(Py_tracefunc func, PyObject *obj)

Like **`PyEval_SetProfile()`** but sets the profile function in all running threads belonging to the current interpreter instead of the setting it only on the current thread.

The caller must have an attached thread state.

As **`PyEval_SetProfile()`**, this function ignores any exceptions raised while setting the profile functions in all threads.

Added in version 3.12.

void PyEval_SetTrace(Py_tracefunc func, PyObject *obj)

Set the tracing function to *func*. This is similar to

`PyEval_SetProfile()`, except the tracing function does receive line-number events and per-opcode events, but does not receive any event related to C function objects being called. Any trace function registered using `PyEval_SetTrace()` will not receive `PyTrace_C_CALL`, `PyTrace_C_EXCEPTION` or `PyTrace_C_RETURN` as a value for the *what* parameter.

See also the `sys.settrace()` function.

The caller must have an attached thread state.

void PyEval_SetTraceAllThreads(Py_tracefunc func, PyObject *obj)

Like `PyEval_SetTrace()` but sets the tracing function in all running threads belonging to the current interpreter instead of the setting it only on the current thread.

The caller must have an attached thread state.

As `PyEval_SetTrace()`, this function ignores any exceptions raised while setting the trace functions in all threads.

Added in version 3.12.

Reference tracing

Added in version 3.13.

typedef int (*PyRefTracer)(PyObject*, int event, void *data)

The type of the trace function registered using `PyRefTracer_SetTracer()`. The first parameter is a Python object that has been just created (when **event** is set to `PyRefTracer_CREATE`) or about to be destroyed (when **event** is set to `PyRefTracer_DESTROY`). The **data** argument is the opaque pointer that was provided when `PyRefTracer_SetTracer()` was called.

If a new tracing function is registered replacing the current one, a call to the trace function will be made with the object set to **NULL** and **event** set to `PyRefTracer_TRACKER_REMOVED`. This will happen just before the new function is registered.

Added in version 3.13.

int PyRefTracer_CREATE

The value for the *event* parameter to `PyRefTracer` functions when a Python object has been created.

int PyRefTracer_DESTROY

The value for the *event* parameter to `PyRefTracer` functions when a Python object has been destroyed.

int PyRefTracer_TRACKER_REMOVED

The value for the *event* parameter to `PyRefTracer` functions when the current tracer is about to be replaced by a new one.

Added in version 3.14.

int PyRefTracer_SetTracer(PyRefTracer tracer, void *data)

Register a reference tracer function. The function will be called when a new Python object has been created or when an object is going to be destroyed. If **data** is provided it must be an opaque pointer that will be provided when the tracer function is called. Return `0` on success. Set an exception and return `-1` on error.

Note that tracer functions **must not** create Python objects inside or otherwise the call will be re-entrant. The tracer also **must not** clear any existing exception or set an exception. A thread state will be active every time the tracer function is called.

There must be an attached thread state when calling this function.

If another tracer function was already registered, the old function will be called with **event** set to `PyRefTracer_TRACKER_REMOVED` just before the new function is registered.

Added in version 3.13.

PyRefTracer PyRefTracer_GetTracer(void **data)

Get the registered reference tracer function and the value of the opaque data pointer that was registered when `PyRefTracer_SetTracer()` was called. If no tracer was registered this function will return NULL and will set the **data** pointer to NULL.

There must be an attached thread state when calling this function.

Added in version 3.13.

Configuración de inicialización de Python

PyInitConfig C API

Added in version 3.14.

Python can be initialized with `Py_InitializeFromInitConfig()`.

La función `Py_RunMain()` se puede utilizar para escribir un programa Python personalizado.

Consulte también Inicialización, finalización y subprocesos.

Ver también

PEP 741 [https://peps.python.org/pep-0741/] «Python Configuration C API».

Ejemplo

Example of customized Python always running with the Python Development Mode enabled; return `-1` on error:

```
int init_python(void)
{
    PyInitConfig *config = PyInitConfig_Create();
    if (config == NULL) {
        printf("PYTHON INIT ERROR: memory allocation fai
        return -1;
    }

    // Enable the Python Development Mode
    if (PyInitConfig_SetInt(config, "dev_mode", 1) < 0)
        goto error;
    }

    // Initialize Python with the configuration
    if (Py_InitializeFromInitConfig(config) < 0) {
        goto error;
    }
    PyInitConfig_Free(config);
    return 0;

error:
    {
        // Display the error message.
        //
        // This uncommon braces style is used, because y
        // goto targets point to variable declarations.
        const char *err_msg;
        (void)PyInitConfig_GetError(config, &err_msg);
        printf("PYTHON INIT ERROR: %s\n", err_msg);
        PyInitConfig_Free(config);
        return -1;
    }
}
```

Create Config

struct PyInitConfig

Opaque structure to configure the Python initialization.

PyInitConfig *PyInitConfig_Create(void)

Create a new initialization configuration using Isolated Configuration default values.

It must be freed by **`PyInitConfig_Free()`**.

Return `NULL` on memory allocation failure.

void PyInitConfig_Free(PyInitConfig *config)

Free memory of the initialization configuration *config*.

If *config* is `NULL`, no operation is performed.

Error Handling

int PyInitConfig_GetError(PyInitConfig *config, const char **err_msg)

Get the *config* error message.

- Set **err_msg* and return `1` if an error is set.
- Set **err_msg* to `NULL` and return `0` otherwise.

An error message is a UTF-8 encoded string.

If *config* has an exit code, format the exit code as an error message.

The error message remains valid until another `PyInitConfig` function is called with *config*. The caller doesn’t have to free the error message.

int PyInitConfig_GetExitCode(PyInitConfig *config, int *exitcode)

Get the *config* exit code.

- Set **exitcode* and return `1` if *config* has an exit code set.
- Return `0` if *config* has no exit code set.

Only the `Py_InitializeFromInitConfig()` function can set an exit code if the `parse_argv` option is non-zero.

An exit code can be set when parsing the command line failed (exit code `2`) or when a command line option asks to display the command line help (exit code `0`).

Get Options

The configuration option *name* parameter must be a non-NULL null-terminated UTF-8 encoded string. See Configuration Options.

int PyInitConfig_HasOption(PyInitConfig *config, const char *name)

Test if the configuration has an option called *name*.

Return `1` if the option exists, or return `0` otherwise.

int PyInitConfig_GetInt(PyInitConfig *config, const char *name, int64_t *value)

Get an integer configuration option.

- Set **value*, and return `0` on success.
- Set an error in *config* and return `-1` on error.

int PyInitConfig_GetStr(PyInitConfig *config, const char *name, char **value)

Get a string configuration option as a null-terminated UTF-8 encoded string.

- Set **value*, and return `0` on success.
- Set an error in *config* and return `-1` on error.

**value* can be set to `NULL` if the option is an optional string and the option is unset.

On success, the string must be released with `free(value)` if it's not `NULL`.

int PyInitConfig_GetStrList(PyInitConfig *config, const char *name, size_t *length, char ***items)

Get a string list configuration option as an array of null-terminated UTF-8 encoded strings.

- Set **length* and **value*, and return `0` on success.
- Set an error in *config* and return `-1` on error.

On success, the string list must be released with `PyInitConfig_FreeStrList(length, items)`.

void PyInitConfig_FreeStrList(size_t length, char **items)

Free memory of a string list created by `PyInitConfig_GetStrList()`.

Set Options

The configuration option *name* parameter must be a non-NULL null-terminated UTF-8 encoded string. See Configuration Options.

Some configuration options have side effects on other options. This logic is only implemented when `Py_InitializeFromInitConfig()` is called, not by the «Set» functions below. For example, setting `dev_mode` to `1` does not set `faulthandler` to `1`.

int PyInitConfig_SetInt(PyInitConfig *config, const char *name, int64_t value)

Set an integer configuration option.

- Return `0` on success.
- Set an error in *config* and return `-1` on error.

int PyInitConfig_SetStr(PyInitConfig *config, const char *name, const char *value)

Set a string configuration option from a null-terminated UTF-8 encoded string. The string is copied.

- Return `0` on success.
- Set an error in *config* and return `-1` on error.

int PyInitConfig_SetStrList(PyInitConfig *config, const char *name, size_t length, char *const *items)

Set a string list configuration option from an array of null-terminated UTF-8 encoded strings. The string list is copied.

- Return `0` on success.
- Set an error in *config* and return `-1` on error.

Module

int PyInitConfig_AddModule(PyInitConfig *config, const char *name, PyObject *(*initfunc)(void))

Add a built-in extension module to the table of built-in modules.

The new module can be imported by the name *name*, and uses the function *initfunc* as the initialization function called on the first attempted import.

- Return `0` on success.
- Set an error in *config* and return `-1` on error.

If Python is initialized multiple times, `PyInitConfig_AddModule()` must be called at each Python initialization.

Similar to the **`PyImport_AppendInittab()`** function.

Initialize Python

int Py_InitializeFromInitConfig(PyInitConfig *config)

Initialize Python from the initialization configuration.

- Return `0` on success.
- Set an error in *config* and return `-1` on error.
- Set an exit code in *config* and return `-1` if Python wants to exit.

See `PyInitConfig_GetExitcode()` for the exit code case.

Configuration Options

Option	PyConfig/PyPreConfig member	Type	Visibility
`"allocator"`	**`allocator`**	`int`	Read-only
`"argv"`	**`argv`**	`list[str]`	Public
`"base_exec_prefix"`	**`base_exec_prefix`**	`str`	Public

base_executable	base_executable"	Public
base_prefix	base_prefix"	Public
buffered_stdio	buffered_stdio"	Read-only
bytes_warning	bytes_warning"	Public
check_hash_pycs_mode	check_hash_pycs_mode"	Read-only
code_debug_ranges	code_debug_ranges"	Read-only
coerce_c_locale	coerce_c_locale"	Read-only
coerce_c_locale_warn	coerce_c_locale_warn"	Read-only
configure_c_stdio	configure_c_stdio"	Read-only
configure_locale	configure_locale"	Read-only
cpu_count	cpu_count"	Public
dev_mode	dev_mode"	Read-only
dump_refs	dump_refs"	Read-only
dump_refs_file	dump_refs_file"	Read-only
exec_prefix	exec_prefix"	Public
executable	executable"	Public
faulthandler	faulthandler"	Read-only
filesystem_encoding	filesystem_encoding"	Read-only
filesystem_errors	filesystem_errors"	Read-only
hash_seed	hash_seed"	Read-only
home	home"	Read-only
import_time	import_time"	Read-only
inspect	inspect"	Public
install_signal_handlers	install_signal_handlers"	Read-only
int_max_str_digits	int_max_str_digits"	Public
interactive	interactive"	Public
isolated	isolated"	Read-only
legacy_windows_fs_encoding	legacy_windows_fs_encoding"	Read-only
legacy_windows_stdio	legacy_windows_stdio"	Read-only
malloc_stats	malloc_stats"	Read-only
module_search_paths	module_search_paths"	Public
optimization_level	optimization_level"	Public
orig_argv	orig_argv"	Read-only
parse_argv	parse_argv"	Read-only
parser_debug	parser_debug"	Public
pathconfig_warnings	pathconfig_warnings"	Read-only
perf_profiling	perf_profiling"	Read-only
platlibdir	platlibdir"	Public
prefix	prefix"	Public
program_name	program_name"	Read-only

`"pycache_prefix"`	**`pycache_prefix`**	`str`	Public
`"quiet"`	**`quiet`**	`bool`	Public
`"run_command"`	**`run_command`**	`str`	Read-only
`"run_filename"`	**`run_filename`**	`str`	Read-only
`"run_module"`	**`run_module`**	`str`	Read-only
`"run_presite"`	**`run_presite`**	`str`	Read-only
`"safe_path"`	**`safe_path`**	`bool`	Read-only
`"show_ref_count"`	**`show_ref_count`**	`bool`	Read-only
`"site_import"`	**`site_import`**	`bool`	Read-only
`"skip_source_first_line"`	**`skip_source_first_line`**	`bool`	Read-only
`"stdio_encoding"`	**`stdio_encoding`**	`str`	Read-only
`"stdio_errors"`	**`stdio_errors`**	`str`	Read-only
`"stdlib_dir"`	**`stdlib_dir`**	`str`	Public
`"tracemalloc"`	**`tracemalloc`**	`int`	Read-only
`"use_environment"`	**`use_environment`**	`bool`	Public
`"use_frozen_modules"`	**`use_frozen_modules`**	`bool`	Read-only
`"use_hash_seed"`	**`use_hash_seed`**	`bool`	Read-only
`"use_system_logger"`	**`use_system_logger`**	`bool`	Read-only
`"user_site_directory"`	**`user_site_directory`**	`bool`	Read-only
`"utf8_mode"`	**`utf8_mode`**	`bool`	Read-only
`"verbose"`	**`verbose`**	`int`	Public
`"warn_default_encoding"`	**`warn_default_encoding`**	`bool`	Read-only
`"warnoptions"`	**`warnoptions`**	`list[str]`	Public
`"write_bytecode"`	**`write_bytecode`**	`bool`	Public
`"xoptions"`	**`xoptions`**	`dict[str, str]`	Public
`"_pystats"`	**`_pystats`**	`bool`	Read-only

Visibility:

- Public: Can be retrieved by **`PyConfig_Get()`** and set by **`PyConfig_Set()`**.
- Read-only: Can be retrieved by **`PyConfig_Get()`**, but cannot be set by **`PyConfig_Set()`**.

Runtime Python configuration API

At runtime, it's possible to get and set configuration options using **`PyConfig_Get()`** and **`PyConfig_Set()`** functions.

The configuration option *name* parameter must be a non-NULL null-terminated UTF-8 encoded string. See Configuration Options.

Some options are read from the **`sys`** attributes. For example, the option `"argv"` is read from **`sys.argv`**.

PyObject *PyConfig_Get(const char *name)

Get the current runtime value of a configuration option as a Python object.

- Return a new reference on success.
- Set an exception and return `NULL` on error.

The object type depends on the configuration option. It can be:

- `bool`
- `int`
- `str`
- `list[str]`
- `dict[str, str]`

The caller must have an attached thread state. The function cannot be called before Python initialization nor after Python finalization.

Added in version 3.14.

int PyConfig_GetInt(const char *name, int *value)

Similar to **`PyConfig_Get()`**, but get the value as a C int.

- Return `0` on success.
- Set an exception and return `-1` on error.

Added in version 3.14.

PyObject *PyConfig_Names(void)

Get all configuration option names as a `frozenset`.

- Return a new reference on success.
- Set an exception and return `NULL` on error.

The caller must have an attached thread state. The function cannot be called before Python initialization nor after Python finalization.

Added in version 3.14.

int PyConfig_Set(const char *name, PyObject *value)

Set the current runtime value of a configuration option.

- Raise a **`ValueError`** if there is no option *name*.
- Raise a **`ValueError`** if *value* is an invalid value.
- Raise a **`ValueError`** if the option is read-only (cannot be set).
- Raise a **`TypeError`** if *value* has not the proper type.

The caller must have an attached thread state. The function cannot be called before Python initialization nor after Python finalization.

Raises an auditing event `cpython.PyConfig_Set` with arguments `name`, `value`.

Added in version 3.14.

PyConfig C API

Added in version 3.8.

Python se puede inicializar con **`Py_InitializeFromConfig()`** y la estructura **`PyConfig`**. Se puede preinicializar con **`Py_PreInitialize()`** y la estructura **`PyPreConfig`**.

Hay dos tipos de configuración:

- La Configuración de Python se puede utilizar para crear un Python personalizado que se comporte como el Python normal. Por ejemplo, las variables de entorno y los argumentos de la línea de comandos se utilizan para configurar Python.
- La Configuración Aislada se puede utilizar para incrustar

Python en una aplicación. Aísla a Python del sistema. Por ejemplo, las variables de entorno se ignoran, la configuración regional LC_CTYPE se deja sin cambios y no se registra ningún manejador de señales.

La función `Py_RunMain()` se puede utilizar para escribir un programa Python personalizado.

Consulte también Inicialización, finalización y subprocesos.

Ver también

PEP 587 [https://peps.python.org/pep-0587/] «Configuración de inicialización de Python».

Ejemplo

Ejemplo de Python personalizado que siempre se ejecuta en modo aislado:

```
int main(int argc, char **argv)
{
    PyStatus status;

    PyConfig config;
    PyConfig_InitPythonConfig(&config);
    config.isolated = 1;

    /* Decode command line arguments.
       Implicitly preinitialize Python (in isolated mode
    status = PyConfig_SetBytesArgv(&config, argc, argv);
    if (PyStatus_Exception(status)) {
        goto exception;
    }

    status = Py_InitializeFromConfig(&config);
    if (PyStatus_Exception(status)) {
        goto exception;
    }
    PyConfig_Clear(&config);

    return Py_RunMain();

exception:
    PyConfig_Clear(&config);
    if (PyStatus_IsExit(status)) {
        return status.exitcode;
    }
    /* Display the error message and exit the process wi
       non-zero exit code */
    Py_ExitStatusException(status);
}
```

PyWideStringList

type PyWideStringList

Lista de cadenas de caracteres `wchar_t*`.

Si *length* no es cero, *items* no deben ser `NULL` y todas las cadenas de caracteres deben ser no `NULL`.

Métodos:

PyStatus PyWideStringList_Append(PyWideStringList *list, const wchar_t *item)

Agregar *item* a *list*.

Python debe estar preinicializado para llamar a esta función.

PyStatus PyWideStringList_Insert(PyWideStringList *list, Py_ssize_t index, const wchar_t *item)

Inserta *item* en *list* en *index*.

Si *index* es mayor o igual que el largo de *list*, agrega *item* a *list*.

index debe ser mayor o igual que `0`.

Python debe estar preinicializado para llamar a esta función.

Campos de estructura:

Py_ssize_t length

Longitud de la lista.

wchar_t **items

Elementos de la lista.

PyStatus

type PyStatus

Estructura para almacenar el estado de una función de inicialización: éxito, error o salida.

Para un error, puede almacenar el nombre de la función C que creó el error.

Campos de estructura:

int exitcode

Código de salida El argumento pasó a `exit()`.

const char *err_msg

Mensaje de error.

const char *func

El nombre de la función que creó un error puede ser `NULL`.

Funciones para crear un estado:

PyStatus PyStatus_Ok(void)

Éxito.

PyStatus PyStatus_Error(const char *err_msg)

Error de inicialización con un mensaje.

err_msg no debe ser `NULL`.

PyStatus PyStatus_NoMemory(void)

Error de asignación de memoria (sin memoria).

PyStatus PyStatus_Exit(int exitcode)

Sale de Python con el código de salida especificado.

Funciones para manejar un estado:

int PyStatus_Exception(PyStatus status)

¿Es el estado un error o una salida? Si es verdadero, la excepción debe ser manejada; por ejemplo llamando a **`Py_ExitStatusException()`**.

int PyStatus_IsError(PyStatus status)

¿Es el resultado un error?

int PyStatus_IsExit(PyStatus status)

¿El resultado es una salida?

void Py_ExitStatusException(PyStatus status)

Llama a `exit(exitcode)` si *status* es una salida. Imprime el mensaje de error y sale con un código de salida distinto de cero si *status* es un error. Solo se debe llamar si `PyStatus_Exception(status)` no es cero.

Nota

Internamente, Python usa macros que establecen `PyStatus.func`, mientras que las funciones para crear un estado establecen `func` en `NULL`.

Ejemplo:

```
PyStatus alloc(void **ptr, size_t size)
{
    *ptr = PyMem_RawMalloc(size);
    if (*ptr == NULL) {
        return PyStatus_NoMemory();
    }
    return PyStatus_Ok();
}

int main(int argc, char **argv)
{
    void *ptr;
    PyStatus status = alloc(&ptr, 16);
    if (PyStatus_Exception(status)) {
        Py_ExitStatusException(status);
    }
    PyMem_Free(ptr);
    return 0;
}
```

PyPreConfig

type PyPreConfig

Estructura utilizada para preinicializar Python.

Función para inicializar una preconfiguración:

void PyPreConfig_InitPythonConfig(PyPreConfig *preconfig)

Inicializa la preconfiguración con Configuración de Python.

void PyPreConfig_InitIsolatedConfig(PyPreConfig *preconfig)

Inicializa la preconfiguración con Configuración aislada.

Campos de estructura:

int allocator

Nombre de los asignadores de memoria de Python:

- `PYMEM_ALLOCATOR_NOT_SET` (`0`): no cambia asignadores de memoria (usa los valores predeterminados)
- `PYMEM_ALLOCATOR_DEFAULT` (`1`): asignadores de memoria predeterminados.
- `PYMEM_ALLOCATOR_DEBUG` (`2`): asignadores de memoria predeterminados con ganchos de depuración.
- `PYMEM_ALLOCATOR_MALLOC` (`3`): usa `malloc()` de la biblioteca C.
- `PYMEM_ALLOCATOR_MALLOC_DEBUG` (`4`): fuerza el uso de `malloc()` con ganchos de depuración.
- `PYMEM_ALLOCATOR_PYMALLOC` (`5`): asignador de memoria pymalloc de Python.
- `PYMEM_ALLOCATOR_PYMALLOC_DEBUG` (`6`): asignador de memoria pymalloc de Python con ganchos de depuración.
- `PYMEM_ALLOCATOR_MIMALLOC` (`6`): usa `mimalloc`, un reemplazo rápido de malloc.
- `PYMEM_ALLOCATOR_MIMALLOC_DEBUG` (`7`): usa `mimalloc`, un reemplazo rápido de malloc con ganchos de depuración.

`PYMEM_ALLOCATOR_PYMALLOC` y `PYMEM_ALLOCATOR_PYMALLOC_DEBUG` no son compatibles si Python es **`configurado usando --without-pymalloc`**.

`PYMEM_ALLOCATOR_MIMALLOC` y `PYMEM_ALLOCATOR_MIMALLOC_DEBUG` no son compatibles si Python es **`configurado usando --without-mimalloc`** o si el soporte atómico subyacente no está disponible.

Ver Administración de memorias.

Predeterminado: `PYMEM_ALLOCATOR_NOT_SET`.

int configure_locale

Establece la configuración regional LC_CTYPE a la configuración regional preferida del usuario.

Si es igual a `0`, establece los miembros **`coerce_c_locale`** y **`coerce_c_locale_warn`** a `0`.

Vea el locale encoding.

Predeterminado: `1` en la configuración de Python, `0` en la configuración aislada.

int coerce_c_locale

Si es igual a `2`, imponga la configuración regional C.

Si es igual a `1`, lee la configuración regional LC_CTYPE para decidir si debe ser forzada.

Vea el locale encoding.

Predeterminado: `-1` en la configuración de Python, `0` en la configuración aislada.

int coerce_c_locale_warn

Si no es cero, emita una advertencia si la configuración regional C está coaccionada.

Predeterminado: `-1` en la configuración de Python, `0` en la configuración aislada.

int dev_mode

Modo de desarrollo de Python: consulte **`PyConfig.dev_mode`**.

Por defecto: `-1` en modo Python, `0` en modo aislado.

int isolated

Modo aislado: consulte **`PyConfig.isolated`**.

Por defecto: `0` en modo Python, `1` en modo aislado.

int legacy_windows_fs_encoding

Si no es cero:

- Establezca **`PyPreConfig.utf8_mode`** en `0`,
- Establezca **`PyConfig.filesystem_encoding`** en `"mbcs"`,
- Establezca **`PyConfig.filesystem_errors`** en `"replace"`.

Inicializado desde el valor de la variable de entorno **`PYTHONLEGACYWINDOWSFSENCODING`**.

Solo disponible en Windows. La macro `#ifdef MS_WINDOWS` se puede usar para el código específico de Windows.

Predeterminado: `0`.

int parse_argv

Si no es cero, **`Py_PreInitializeFromArgs()`** y **`Py_PreInitializeFromBytesArgs()`** analizan su argumento `argv` de la misma manera que Python analiza los argumentos de la línea de comandos: ver Argumentos de línea de comandos.

Predeterminado: `1` en la configuración de Python, `0` en la configuración aislada.

int use_environment

¿Utiliza variables de entorno? Consulte **`PyConfig.use_environment`**.

Predeterminado: `1` en la configuración de Python y `0` en la configuración aislada.

int utf8_mode

Si es distinto de cero, habilite Python UTF-8 Mode.

Establecido a `0` o `1` por la opción de línea de comandos **`-X utf8`** y la variable de entorno

`PYTHONUTF8`.

También se pone a `1` si la configuración regional `LC_CTYPE` es `C` o `POSIX`.

Predeterminado: `-1` en la configuración de Python y `0` en la configuración aislada.

Preinicialización de Python con PyPreConfig

La preinicialización de Python:

- Establecer los asignadores de memoria de Python (**`PyPreConfig.allocator`**)
- Configurar la configuración regional LC_CTYPE (locale encoding)
- Establecer el Python UTF-8 Mode (**`PyPreConfig.utf8_mode`**)

La preconfiguración actual (tipo `PyPreConfig`) se almacena en `_PyRuntime.preconfig`.

Funciones para preinicializar Python:

PyStatus Py_PreInitialize(const PyPreConfig *preconfig)

Preinicializa Python desde la preconfiguración *preconfig*.

preconfig no debe ser `NULL`.

PyStatus Py_PreInitializeFromBytesArgs(const PyPreConfig *preconfig, int argc, char *const *argv)

Preinicializa Python desde la preconfiguración *preconfig*.

Analice los argumentos de la línea de comando *argv* (cadenas de bytes) si **`parse_argv`** de *preconfig* no es cero.

preconfig no debe ser `NULL`.

PyStatus Py_PreInitializeFromArgs(const PyPreConfig *preconfig, int argc, wchar_t *const *argv)

Preinicializa Python desde la preconfiguración *preconfig*.

Analice los argumentos de la línea de comando *argv* (cadenas anchas) si **`parse_argv`** de *preconfig* no es cero.

preconfig no debe ser `NULL`.

La persona que llama es responsable de manejar las excepciones (error o salida) usando **`PyStatus_Exception()`** y **`Py_ExitStatusException()`**.

Para Configuración de Python (**`PyPreConfig_InitPythonConfig()`**), si Python se inicializa con argumentos de línea de comando, los argumentos de la línea de comando también deben pasarse para preinicializar Python, ya que tienen un efecto en la preconfiguración como codificaciones. Por ejemplo, la opción de línea de comando **`-X utf8`** habilita el Python UTF-8 Mode.

`PyMem_SetAllocator()` se puede llamar después de **`Py_PreInitialize()`** y antes **`Py_InitializeFromConfig()`** para instalar un asignador de memoria personalizado. Se puede llamar antes **`Py_PreInitialize()`** si **`PyPreConfig.allocator`** está configurado en `PYMEM_ALLOCATOR_NOT_SET`.

Las funciones de asignación de memoria de Python como **`PyMem_RawMalloc()`** no deben usarse antes de la preinicialización de Python, mientras que llamar directamente a `malloc()` y `free()` siempre es seguro. No se debe llamar a **`Py_DecodeLocale()`** antes de la preinicialización de Python.

Ejemplo usando la preinicialización para habilitar el Python UTF-8 Mode

```
PyStatus status;
PyPreConfig preconfig;
PyPreConfig_InitPythonConfig(&preconfig);
```

```
preconfig.utf8_mode = 1;

status = Py_PreInitialize(&preconfig);
if (PyStatus_Exception(status)) {
    Py_ExitStatusException(status);
}

/* at this point, Python speaks UTF-8 */

Py_Initialize();
/* ... use Python API here ... */
Py_Finalize();
```

PyConfig

type PyConfig

Estructura que contiene la mayoría de los parámetros para configurar Python.

Cuando termine, se debe utilizar la función **`PyConfig_Clear()`** para liberar la memoria de configuración.

Métodos de estructura:

void PyConfig_InitPythonConfig(PyConfig *config)

Inicialice la configuración con la Configuración de Python.

void PyConfig_InitIsolatedConfig(PyConfig *config)

Inicialice la configuración con la Configuración Aislada.

PyStatus PyConfig_SetString(PyConfig *config, wchar_t *const *config_str, const wchar_t *str)

Copia la cadena de caracteres anchos *str* en `*config_str`.

Preinicializa Python si es necesario.

PyStatus PyConfig_SetBytesString(PyConfig *config, wchar_t *const *config_str, const char *str)

Decodifique *str* usando **`Py_DecodeLocale()`** y establezca el resultado en `*config_str`.

Preinicializa Python si es necesario.

PyStatus PyConfig_SetArgv(PyConfig *config, int argc, wchar_t *const *argv)

Configure los argumentos de la línea de comando (miembro **`argv`** de *config*) de la lista *argv* de cadenas de caracteres anchas.

Preinicializa Python si es necesario.

PyStatus PyConfig_SetBytesArgv(PyConfig *config, int argc, char *const *argv)

Establezca argumentos de línea de comando (miembro **`argv`** de *config*) de la lista *argv* de cadenas de bytes. Decodifica bytes usando **`Py_DecodeLocale()`**.

Preinicializa Python si es necesario.

PyStatus PyConfig_SetWideStringList(PyConfig *config, PyWideStringList *list, Py_ssize_t length, wchar_t **items)

Establece la lista de cadenas de caracteres anchas *list* a *length* y *items*.

Preinicializa Python si es necesario.

PyStatus PyConfig_Read(PyConfig *config)

Lee toda la configuración de Python.

Los campos que ya están inicializados no se modifican.

Los campos para la configuración de ruta ya no se calculan ni modifican al llamar a esta función, a partir de Python 3.11.

La función **`PyConfig_Read()`** solo analiza los argumentos **`PyConfig.argv`** una vez: **`PyConfig.parse_argv`** se establece en `2` después de que se analizan los argumentos. Como los argumentos de Python se eliminan de **`PyConfig.argv`**, analizar los argumentos dos veces analizaría las opciones de la aplicación como opciones de Python.

Preinicializa Python si es necesario.

Distinto en la versión 3.10: Los argumentos **`PyConfig.argv`** ahora solo se analizan una vez, **`PyConfig.parse_argv`** se establece en `2` después de analizar los argumentos y los argumentos solo se analizan si **`PyConfig.parse_argv`** es igual a `1`.

Distinto en la versión 3.11: **`PyConfig_Read()`** ya no calcula todas las rutas, por lo que los campos listados en Python Path Configuration ya no pueden ser actualizados hasta que se llame a **`Py_InitializeFromConfig()`**.

void PyConfig_Clear(PyConfig *config)

Libera memoria de configuración.

Most `PyConfig` methods preinitialize Python if needed. In that case, the Python preinitialization configuration (**`PyPreConfig`**) is based on the **`PyConfig`**. If configuration fields which are in common with **`PyPreConfig`** are tuned, they must be set before calling a **`PyConfig`** method:

- **`PyConfig.dev_mode`**
- **`PyConfig.isolated`**
- **`PyConfig.parse_argv`**
- **`PyConfig.use_environment`**

Además, si se usa **`PyConfig_SetArgv()`** o **`PyConfig_SetBytesArgv()`**, este método debe llamarse antes que otros métodos, ya que la configuración de preinicialización depende de los argumentos de la línea de comando (si `parse_argv` no es cero).

Quien llama de estos métodos es responsable de manejar las excepciones (error o salida) usando `PyStatus_Exception()` y `Py_ExitStatusException()`.

Campos de estructura:

PyWideStringList argv

Establece los argumentos de la línea de comandos `sys.argv` basados en `argv`. Estos parámetros son similares a los pasados a la función `main()` del programa con la diferencia de que la primera entrada debe referirse al archivo de script a ejecutar en lugar del ejecutable que aloja al intérprete de Python. Si no hay un script que se vaya a ejecutar, la primera entrada en `argv` puede ser una cadena vacía.

Configure `parse_argv` en `1` para analizar `argv` de la misma manera que Python normal analiza los argumentos de la línea de comandos de Python y luego quita los argumentos de Python de `argv`.

Si `argv` está vacío, se agrega una cadena vacía para garantizar que `sys.argv` siempre exista y nunca esté vacío.

Valor predeterminado: `NULL`.

Consulte también el miembro `orig_argv`.

int safe_path

Si es igual a cero, `Py_RunMain()` agrega una ruta potencialmente insegura a `sys.path` al inicio:

- Si `argv[0]` es igual a `L"-m"` (`python -m module`), añade el directorio de trabajo actual.
- Si se ejecuta un script (`python script.py`),

anteponer el directorio del script. Si es un enlace simbólico, resuelve los enlaces simbólicos.
- En caso contrario (`python -c code` and `python`), añade una cadena vacía, que significa el directorio de trabajo actual.

Establecido en `1` por la opción de la línea de comandos **`-P`** y la variable de entorno **`PYTHONSAFEPATH`**.

Predeterminado: `0` en la configuración de Python, `1` en la configuración aislada.

Added in version 3.11.

wchar_t *base_exec_prefix

`sys.base_exec_prefix`.

Valor predeterminado: `NULL`.

Parte de la salida Python Path Configuration.

Consulte también **`PyConfig.exec_prefix`**.

wchar_t *base_executable

Ejecutable base de Python: **`sys._base_executable`**.

Establecido por la variable de entorno **`__PYVENV_LAUNCHER__`**.

Establecido desde **`PyConfig.executable`** si `NULL`.

Valor predeterminado: `NULL`.

Parte de la salida Python Path Configuration.

Consulte también **`PyConfig.executable`**.

wchar_t *base_prefix

`sys.base_prefix`.

Valor predeterminado: `NULL`.

Parte de la salida Python Path Configuration.

Consulte también `PyConfig.prefix`.

int buffered_stdio

Si es igual a `0` y `configure_c_stdio` no es cero, deshabilite el almacenamiento en búfer en las secuencias C stdout y stderr.

Establecido en `0` por la opción de línea de comando `-u` y la variable de entorno `PYTHONUNBUFFERED`.

stdin siempre se abre en modo de búfer.

Predeterminado: `1`.

int bytes_warning

Si es igual a `1`, emite una advertencia al comparar `bytes` o `bytearray` con `str`, o al comparar `bytes` con `int`.

Si es igual o mayor a `2`, lanza una excepción `BytesWarning` en estos casos.

Incrementado por la opción de línea de comando `-b`.

Predeterminado: `0`.

int warn_default_encoding

Si no es cero, emite una advertencia `EncodingWarning` cuando `io.TextIOWrapper` usa su codificación predeterminada. Consulte EncodingWarning opcional para obtener más detalles.

Predeterminado: `0`.

Added in version 3.10.

int code_debug_ranges

Si es igual a `0`, desactiva la inclusión de la línea final y las asignaciones de columna en los objetos de código. También desactiva la impresión de indicadores de rastreo a ubicaciones de error específicas.

Establecido en `0` por la variable de entorno **`PYTHONNODEBUGRANGES`** y por la opción de línea de comando **`-X no_debug_ranges`**.

Predeterminado: `1`.

Added in version 3.11.

wchar_t *check_hash_pycs_mode

Controla el comportamiento de validación de archivos `.pyc` basados en hash: valor de la opción de línea de comando **`--check-hash-based-pycs`**.

Valores válidos:

- `L"always"`: Calcula el hash el archivo fuente para invalidación independientemente del valor de la marca “check_source”.
- `L"never"`: suponga que los pycs basados en hash siempre son válidos.
- `L"default"`: El indicador “check_source” en pycs basados en hash determina la invalidación.

Predeterminado: `L"default"`.

Consulte también **PEP 552** [https://peps.python.org/pep-0552/] «Pycs deterministas».

int configure_c_stdio

Si es distinto de cero, configure los flujos estándar de C:

- En Windows, configure el modo binario (`O_BINARY`) en stdin, stdout y stderr.
- Si **`buffered_stdio`** es igual a cero, deshabilite el almacenamiento en búfer de los flujos stdin, stdout y stderr.
- Si **`interactive`** no es cero, habilite el

almacenamiento en búfer de flujo en stdin y stdout (solo stdout en Windows).

Predeterminado: `1` en la configuración de Python, `0` en la configuración aislada.

int dev_mode

Si es distinto de cero, habilita Modo de desarrollo de Python.

Establecido en `1` por la opción **`-X dev`** y la variable de entorno **`PYTHONDEVMODE`**.

Por defecto: `-1` en modo Python, `0` en modo aislado.

int dump_refs

¿Volcar referencias de Python?

Si no es cero, volcar todos los objetos que aún están vivos en la salida.

Establecido en `1` por la variable de entorno **`PYTHONDUMPREFS`**.

Necesita una compilación especial de Python con la macro `Py_TRACE_REFS` definida: consulte la opción **`configure --with-trace-refs`**.

Predeterminado: `0`.

wchar_t *dump_refs_file

Filename where to dump Python references.

Set by the **`PYTHONDUMPREFSFILE`** environment variable.

Valor predeterminado: `NULL`.

Added in version 3.11.

wchar_t *exec_prefix

El prefijo de directorio específico del sitio donde se instalan los archivos Python dependientes de la plataforma: **`sys.exec_prefix`**.

Valor predeterminado: `NULL`.

Parte de la salida Python Path Configuration.

Consulte también **`PyConfig.base_exec_prefix`**.

wchar_t *executable

La ruta absoluta del binario ejecutable para el intérprete de Python: **`sys.executable`**.

Valor predeterminado: `NULL`.

Parte de la salida Python Path Configuration.

Consulte también **`PyConfig.base_executable`**.

int faulthandler

¿Habilitar administrador de fallas?

Si no es cero, llama a **`faulthandler.enable()`** al inicio.

Establecido en `1` por **`-X faulthandler`** y la variable de entorno **`PYTHONFAULTHANDLER`**.

Por defecto: `-1` en modo Python, `0` en modo aislado.

wchar_t *filesystem_encoding

Codificación del sistema de archvios: **`sys.getfilesystemencoding()`**.

En macOS, Android y VxWorks: use `"utf-8"` de forma predeterminada.

En Windows: utilice `"utf-8"` de forma

predeterminada o `"mbcs"` si **`legacy_windows_fs_encoding`** de **`PyPreConfig`** no es cero.

Codificación predeterminada en otras plataformas:

- `"utf-8"` si **`PyPreConfig.utf8_mode`** es distinto de cero.
- `"ascii"` si Python detecta que `nl_langinfo(CODESET)` anuncia la codificación ASCII, mientras que la función `mbstowcs()` decodifica desde una codificación diferente (usualmente Latin1).
- `"utf-8"` si `nl_langinfo(CODESET)` retorna una cadena vacía.
- De lo contrario, utilice el resultado locale encoding: `nl_langinfo(CODESET)`.

Al inicio de Python, el nombre de codificación se normaliza al nombre del códec de Python. Por ejemplo, `"ANSI_X3.4-1968"` se reemplaza por `"ascii"`.

Consulte también el miembro **`filesystem_errors`**.

wchar_t *filesystem_errors

Manejador de errores del sistema de archivos: **`sys.getfilesystemencodeerrors()`**.

En Windows: utilice `"surrogatepass"` de forma predeterminada o `"replace"` si **`legacy_windows_fs_encoding`** de **`PyPreConfig`** no es cero.

En otras plataformas: utilice `"surrogateescape"` de forma predeterminada.

Controladores de errores admitidos:

- `"strict"`
- `"surrogateescape"`
- `"surrogatepass"` (solo compatible con la

codificación UTF-8)

Consulte también el miembro **`filesystem_encoding`**.

int use_frozen_modules

If non-zero, use frozen modules.

Set by the **`PYTHON_FROZEN_MODULES`** environment variable.

Default: `1` in a release build, or `0` in a debug build.

unsigned long hash_seed

int use_hash_seed

Funciones de semillas aleatorias hash.

Si **`use_hash_seed`** es cero, se elige una semilla al azar en el inicio de Python y se ignora **`hash_seed`**.

Establecido por la variable de entorno **`PYTHONHASHSEED`**.

Valor predeterminado de *use_hash_seed*: `-1` en modo Python, `0` en modo aislado.

wchar_t *home

Establece el directorio «home» de Python por defecto, es decir, la ubicación de las bibliotecas estándar de Python (ver **`PYTHONHOME`**).

Establecido por la variable de entorno **`PYTHONHOME`**.

Valor predeterminado: `NULL`.

Parte de la entrada Python Path Configuration.

int import_time

If `1`, profile import time. If `2`, include additional output that indicates when an imported module has already been loaded.

Set by the **`-X importtime`** option and the **`PYTHONPROFILEIMPORTTIME`** environment variable.

Predeterminado: `0`.

Distinto en la versión 3.14: Added support for `import_time = 2`

int inspect

Ingresa al modo interactivo después de ejecutar un script o un comando.

Si es mayor que `0`, habilite la inspección: cuando se pasa un script como primer argumento o se usa la opción -c, ingrese al modo interactivo después de ejecutar el script o el comando, incluso cuando **`sys.stdin`** no parezca ser una terminal.

Incrementado por la opción de línea de comando **`-i`**. Establecido en `1` si la variable de entorno **`PYTHONINSPECT`** no está vacía.

Predeterminado: `0`.

int install_signal_handlers

¿Instalar controladores de señales de Python?

Por defecto: `1` en modo Python, `0` en modo aislado.

int interactive

Si es mayor que `0`, habilita el modo interactivo (REPL).

Incrementado por la opción de línea de comando **`-i`**.

Predeterminado: `0`.

int int_max_str_digits

Configura la limitación de la longitud de conversión de entero a cadena. Un valor inicial de `-1` significa que el valor se tomará de la línea de comandos o del entorno o, de lo contrario, será por defecto 4300 (**`sys.int_info.default_max_str_digits`**). Un valor de `0` deshabilita la limitación. Los valores mayores que cero pero menores que 640 (**`sys.int_info.str_digits_check_threshold`**) no son compatibles y producirán un error.

Configurado por la bandera de línea de comandos **`-X int_max_str_digits`** o la variable de entorno **`PYTHONINTMAXSTRDIGITS`**.

Predeterminado: `-1` en modo Python. 4300 (**`sys.int_info.default_max_str_digits`**) en modo aislado.

Added in version 3.12.

int cpu_count

Si el valor de **`cpu_count`** no es `-1`, entonces anulará los valores de retorno de **`os.cpu_count()`**, **`os.process_cpu_count()`** y **`multiprocessing.cpu_count()`**.

Configurado por la bandera de línea de comandos `-X cpu_count=n|default` o la variable de entorno **`PYTHON_CPU_COUNT`**.

Predeterminado: `-1`.

Added in version 3.13.

int isolated

Si es mayor que `0`, habilite el modo aislado:

- Establezca **`safe_path`** en `1`: no anteponga una ruta potencialmente insegura a **`sys.path`** al inicio de Python, como el directorio actual, el directorio del script o una cadena vacía.
- Establezca **`use_environment`** en `0`: ignore las variables de entorno `PYTHON`.
- Establezca **`user_site_directory`** en `0`: no agregue el directorio del sitio del usuario a **`sys.path`**.
- Python REPL no importa **`readline`** ni habilita la configuración predeterminada de readline en mensajes interactivos.

Establecido en `1` por la opción de línea de comando **`-I`**.

Por defecto: `0` en modo Python, `1` en modo aislado.

Consulte también la Configuración Aislada y **`PyPreConfig.isolated`**.

int legacy_windows_stdio

Si no es cero, use **`io.FileIO`** en lugar de **`io._WindowsConsoleIO`** para **`sys.stdin`**, **`sys.stdout`** y **`sys.stderr`**.

Establecido en `1` si la variable de entorno **`PYTHONLEGACYWINDOWSSTDIO`** está establecida en una cadena no vacía.

Solo disponible en Windows. La macro `#ifdef MS_WINDOWS` se puede usar para el código específico de Windows.

Predeterminado: `0`.

Consulte también **PEP 528** [https://peps.python.org/pep-0528/] (Cambiar la codificación de la consola de Windows a UTF-8).

int malloc_stats

Si no es cero, volcar las estadísticas en Asignador de memoria Python pymalloc en la salida.

Establecido en `1` por la variable de entorno **`PYTHONMALLOCSTATS`**.

La opción se ignora si Python es **`configurado usando la opción --without-pymalloc`**.

Predeterminado: `0`.

wchar_t *platlibdir

Nombre del directorio de la biblioteca de la plataforma: **`sys.platlibdir`**.

Establecido por la variable de entorno **`PYTHONPLATLIBDIR`**.

Predeterminado: valor de la macro `PLATLIBDIR` que se establece mediante la opción **`configure --with-platlibdir`** (predeterminado: `"lib"`, o `"DLLs"` en Windows).

Parte de la entrada Python Path Configuration.

Added in version 3.9.

Distinto en la versión 3.11: Esta macro se utiliza ahora en Windows para localizar los módulos de extensión de la biblioteca estándar, normalmente en `DLLs`. Sin embargo, por compatibilidad, tenga en cuenta que este valor se ignora para cualquier disposición no estándar, incluyendo las construcciones dentro del árbol y los entornos virtuales.

wchar_t *pythonpath_env

Rutas de búsqueda de módulos (**`sys.path`**) como una cadena separada por `DELIM` (**`os.pathsep`**).

Establecido por la variable de entorno **`PYTHONPATH`**.

Valor predeterminado: `NULL`.

Parte de la entrada Python Path Configuration.

PyWideStringList module_search_paths

int module_search_paths_set

Rutas de búsqueda del módulo: **`sys.path`**.

Si **`module_search_paths_set`** es igual a `0`, **`Py_InitializeFromConfig()`** reemplazará **`module_search_paths`** y establecerá **`module_search_paths_set`** en `1`.

Por defecto: lista vacía (`module_search_paths`) y `0` (`module_search_paths_set`).

Parte de la salida Python Path Configuration.

int optimization_level

Nivel de optimización de compilación:

- `0`: Optimizador de mirilla, configure `__debug__` en `True`.
- `1`: Nivel 0, elimina las aserciones, establece `__debug__` en `False`.
- `2`: Nivel 1, elimina docstrings.

Incrementado por la opción de línea de comando **`-O`**. Establecido en el valor de la variable de entorno **`PYTHONOPTIMIZE`**.

Predeterminado: `0`.

PyWideStringList orig_argv

La lista de los argumentos originales de la línea de comandos pasados al ejecutable de Python: **`sys.orig_argv`**.

Si la lista **`orig_argv`** está vacía y **`argv`** no es una lista que solo contiene una cadena vacía, `PyConfig_Read()` copia **`argv`** en **`orig_argv`** antes de modificar **`argv`** (si `parse_argv` no es cero).

Consulte también el miembro `argv` y la función `Py_GetArgcArgv()`.

Predeterminado: lista vacía.

Added in version 3.10.

int parse_argv

¿Analizar los argumentos de la línea de comando?

Si es igual a `1`, analiza `argv` de la misma forma que Python normal analiza argumentos de línea de comando y elimina los argumentos de Python de **`argv`**.

La función `PyConfig_Read()` solo analiza los argumentos `PyConfig.argv` una vez: `PyConfig.parse_argv` se establece en `2` después de que se analizan los argumentos. Como los argumentos de Python se eliminan de **`PyConfig.argv`**, analizar los argumentos dos veces analizaría las opciones de la aplicación como opciones de Python.

Por defecto: `1` en modo Python, `0` en modo aislado.

Distinto en la versión 3.10: Los argumentos `PyConfig.argv` ahora solo se analizan si `PyConfig.parse_argv` es igual a `1`.

int parser_debug

Modo de depuración del analizador. Si es mayor que `0`, active la salida de depuración del analizador (solo para expertos, dependiendo de las opciones de compilación).

Incrementado por la opción de línea de comando `-d`. Establecido en el valor de la variable de entorno `PYTHONDEBUG`.

Necesita una compilación de depuración de Python (se debe definir la macro `Py_DEBUG`).

Predeterminado: `0`.

int pathconfig_warnings

Si no es cero, se permite que el cálculo de la configuración de ruta registre advertencias en `stderr`. Si es igual a `0`, suprima estas advertencias.

Por defecto: `1` en modo Python, `0` en modo aislado.

Parte de la entrada Python Path Configuration.

Distinto en la versión 3.11: Ahora también se aplica en Windows.

wchar_t *prefix

El prefijo de directorio específico del sitio donde se instalan los archivos Python independientes de la plataforma: **`sys.prefix`**.

Valor predeterminado: `NULL`.

Parte de la salida Python Path Configuration.

Consulte también **`PyConfig.base_prefix`**.

wchar_t *program_name

Nombre del programa utilizado para inicializar **`executable`** y en los primeros mensajes de error durante la inicialización de Python.

- En macOS, usa la variable de entorno **`PYTHONEXECUTABLE`** si está configurada.
- Si se define la macro `WITH_NEXT_FRAMEWORK`, utiliza la variable de entorno **`__PYVENV_LAUNCHER__`** si está configurada.
- Utiliza `argv[0]` de **`argv`** si está disponible y no está vacío.

- De lo contrario, utiliza `L"python"` en Windows o `L"python3"` en otras plataformas.

Valor predeterminado: `NULL`.

Parte de la entrada Python Path Configuration.

wchar_t *pycache_prefix

Directorio donde se escriben los archivos `.pyc` almacenados en caché: **`sys.pycache_prefix`**.

Establecido por la opción de línea de comando **`-X pycache_prefix=PATH`** y la variable de entorno **`PYTHONPYCACHEPREFIX`**. La opción de línea de comandos tiene prioridad.

Si `NULL`, **`sys.pycache_prefix`** es establecido a `None`.

Valor predeterminado: `NULL`.

int quiet

Modo silencioso. Si es mayor que `0`, no muestre los derechos de autor y la versión al inicio de Python en modo interactivo.

Incrementado por la opción de línea de comando **`-q`**.

Predeterminado: `0`.

wchar_t *run_command

Valor de la opción de línea de comando **`-c`**.

Usado por **`Py_RunMain()`**.

Valor predeterminado: `NULL`.

wchar_t *run_filename

Nombre de archivo pasado en la línea de comandos: argumento final de la línea de comandos sin **`-c`** o **`-m`**.

Es utilizado por la función **`Py_RunMain()`**.

Por ejemplo, se establece en `script.py` mediante la línea de comandos `python3 script.py arg`.

Consulte también la opción **`PyConfig.skip_source_first_line`**.

Valor predeterminado: `NULL`.

wchar_t *run_module

Valor de la opción de línea de comando **`-m`**.

Usado por **`Py_RunMain()`**.

Valor predeterminado: `NULL`.

wchar_t *run_presite

Ruta `package.module` al módulo que debe ser importado antes de que se ejecute `site.py`.

Establecido por la opción de línea de comandos **`-X presite=package.module`** y la variable de entorno **`PYTHON_PRESITE`**. La opción de línea de comandos tiene precedencia.

Necesita una compilación de depuración de Python (se debe definir la macro `Py_DEBUG`).

Valor predeterminado: `NULL`.

int show_ref_count

¿Mostrar el recuento total de referencias al salir (excluyendo objetos immortal)?

Establecido en `1` por la opción de línea de comando **`-X showrefcount`**.

Necesita una compilación de depuración de Python (la macro `Py_REF_DEBUG` debe estar definida).

Predeterminado: `0`.

int site_import

¿Importar el módulo `site` al inicio?

Si es igual a cero, desactive la importación del sitio del módulo y las manipulaciones dependientes del sitio de `sys.path` que conlleva.

También deshabilite estas manipulaciones si el módulo `site` se importa explícitamente más tarde (llame a `site.main()` si desea que se activen).

Establecido en `0` mediante la opción de línea de comando `-S`.

`sys.flags.no_site` se establece en el valor invertido de `site_import`.

Predeterminado: `1`.

int skip_source_first_line

Si no es cero, omita la primera línea de la fuente `PyConfig.run_filename`.

Permite el uso de formas de `#!cmd` que no son Unix. Esto está destinado únicamente a un truco específico de DOS.

Establecido en `1` mediante la opción de línea de comando `-x`.

Predeterminado: `0`.

wchar_t *stdio_encoding

wchar_t *stdio_errors

Codificación y errores de codificación de `sys.stdin`, `sys.stdout` y `sys.stderr` (pero **`sys.stderr`** siempre usa el controlador de errores `"backslashreplace"`).

Utilice la variable de entorno **`PYTHONIOENCODING`** si no está vacía.

Codificación predeterminada:

- `"UTF-8"` si **`PyPreConfig.utf8_mode`** es distinto de cero.
- De lo contrario, usa el locale encoding.

Manejador de errores predeterminado:

- En Windows: use `"surrogateescape"`.
- `"surrogateescape"` si **`PyPreConfig.utf8_mode`** no es cero o si la configuración regional LC_CTYPE es «C» o «POSIX».
- `"strict"` de lo contrario.

Consulte también **`PyConfig.legacy_windows_stdio`**.

int tracemalloc

¿Habilitar tracemalloc?

Si no es cero, llama a **`tracemalloc.start()`** al inicio.

Establecido por la opción de línea de comando **`-X tracemalloc=N`** y por la variable de entorno **`PYTHONTRACEMALLOC`**.

Por defecto: `-1` en modo Python, `0` en modo aislado.

int perf_profiling

Enable the Linux `perf` profiler support?

If equals to `1`, enable support for the Linux `perf` profiler.

If equals to `2`, enable support for the Linux `perf` profiler with DWARF JIT support.

Set to `1` by **`-X perf`** command-line option and the **`PYTHONPERFSUPPORT`** environment variable.

Set to `2` by the **`-X perf_jit`** command-line option and the **`PYTHON_PERF_JIT_SUPPORT`** environment variable.

Predeterminado: `-1`.

Ver también

See Soporte de Python para el perfilador perf de Linux for more information.

Added in version 3.12.

wchar_t *stdlib_dir

Directory of the Python standard library.

Valor predeterminado: `NULL`.

Added in version 3.11.

int use_environment

¿Utiliza variables de entorno?

Si es igual a cero, ignora las variables de entorno.

Establecido en `0` por la variable de entorno **`-E`**.

Predeterminado: `1` en la configuración de Python y `0` en la configuración aislada.

int use_system_logger

If non-zero, `stdout` and `stderr` will be redirected to the system log.

Only available on macOS 10.12 and later, and on iOS.

Default: `0` (don't use the system log) on macOS; `1` on iOS (use the system log).

Added in version 3.14.

int user_site_directory

Si es distinto de cero, agregue el directorio del sitio del usuario a **`sys.path`**.

Establecido en `0` por las opciones de línea de comando **`-s`** y **`-I`**.

Establecido en `0` por la variable de entorno **`PYTHONNOUSERSITE`**.

Por defecto: `1` en modo Python, `0` en modo aislado.

int verbose

Modo detallado. Si es mayor que `0`, imprime un mensaje cada vez que se importa un módulo, mostrando el lugar (nombre de archivo o módulo incorporado) desde el que se carga.

Si es mayor o igual a `2`, imprime un mensaje para cada archivo que se verifica al buscar un módulo. También proporciona información sobre la limpieza del módulo al salir.

Incrementado por la opción de línea de comando **`-v`**.

Establecido por el valor de la variable de entorno **`PYTHONVERBOSE`**.

Predeterminado: `0`.

PyWideStringList warnoptions

Opciones del módulo **`warnings`** para crear filtros de advertencias, de menor a mayor prioridad:

`sys.warnoptions`.

El módulo `warnings` agrega `sys.warnoptions` en el orden inverso: el último elemento `PyConfig.warnoptions` se convierte en el primer elemento de **`warnings.filters`** que es verificado primero (máxima prioridad).

Las opciones de la línea de comando `-W` agregan su valor a `warnoptions`, se puede usar varias veces.

La variable de entorno `PYTHONWARNINGS` también se puede utilizar para agregar opciones de advertencia. Se pueden especificar varias opciones, separadas por comas (`,`).

Predeterminado: lista vacía.

int write_bytecode

Si es igual a `0`, Python no intentará escribir archivos `.pyc` al importar módulos fuente.

Establecido en `0` por la opción de línea de comando `-B` y la variable de entorno `PYTHONDONTWRITEBYTECODE`.

`sys.dont_write_bytecode` se inicializa al valor invertido de `write_bytecode`.

Predeterminado: `1`.

PyWideStringList xoptions

Valores de las opciones de la línea de comando `-X`: `sys._xoptions`.

Predeterminado: lista vacía.

int _pystats

If non-zero, write performance statistics at Python exit.

Need a special build with the `Py_STATS` macro: see **`--enable-pystats`**.

Predeterminado: `0`.

Si **`parse_argv`** no es cero, los argumentos de **`argv`** se analizan de la misma forma que Python normal analiza argumentos de línea de comando, y los argumentos de Python se eliminan de **`argv`**.

Las opciones de **`xoptions`** se analizan para establecer otras opciones: consulte la opción de línea de comando **`-X`**.

Distinto en la versión 3.9: El campo `show_alloc_count` fue removido.

Inicialización con PyConfig

Initializing the interpreter from a populated configuration struct is handled by calling **`Py_InitializeFromConfig()`**.

La persona que llama es responsable de manejar las excepciones (error o salida) usando **`PyStatus_Exception()`** y **`Py_ExitStatusException()`**.

Si se utilizan **`PyImport_FrozenModules()`**, **`PyImport_AppendInittab()`** o **`PyImport_ExtendInittab()`**, deben establecerse o llamarse después de la preinicialización de Python y antes de la inicialización de Python. Si Python se inicializa varias veces, se debe llamar a **`PyImport_AppendInittab()`** o **`PyImport_ExtendInittab()`** antes de cada inicialización de Python.

La configuración actual (tipo `PyConfig`) se almacena en `PyInterpreterState.config`.

Ejemplo de configuración del nombre del programa:

```
void init_python(void)
{
    PyStatus status;
    PyConfig config;
    PyConfig_InitPythonConfig(&config);

    /* Set the program name. Implicitly preinitialize Py
    status = PyConfig_SetString(&config, &config.program
                                L"/path/to/my_program");
    if (PyStatus_Exception(status)) {
        goto exception;
    }

    status = Py_InitializeFromConfig(&config);
    if (PyStatus_Exception(status)) {
        goto exception;
    }
    PyConfig_Clear(&config);
    return;

exception:
    PyConfig_Clear(&config);
    Py_ExitStatusException(status);
}
```

Ejemplo más completo modificando la configuración predeterminada, lee la configuración y luego anula algunos parámetros. Ten en cuenta que, desde la versión 3.11, muchos parámetros no se calculan hasta la inicialización, por lo que los valores no se pueden leer desde la estructura de configuración. Cualquier valor establecido antes de llamar a la inicialización se dejará sin cambios por la inicialización:

```
PyStatus init_python(const char *program_name)
{
    PyStatus status;

    PyConfig config;
    PyConfig_InitPythonConfig(&config);

    /* Set the program name before reading the configura
```

```
   (decode byte string from the locale encoding).

   Implicitly preinitialize Python. */
status = PyConfig_SetBytesString(&config, &config.pr
                                 program_name);
if (PyStatus_Exception(status)) {
    goto done;
}

/* Read all configuration at once */
status = PyConfig_Read(&config);
if (PyStatus_Exception(status)) {
    goto done;
}

/* Specify sys.path explicitly */
/* If you want to modify the default set of paths, f
   initialization first and then use PySys_GetObject
config.module_search_paths_set = 1;
status = PyWideStringList_Append(&config.module_sear
                                 L"/path/to/stdlib")
if (PyStatus_Exception(status)) {
    goto done;
}
status = PyWideStringList_Append(&config.module_sear
                                 L"/path/to/more/moc
if (PyStatus_Exception(status)) {
    goto done;
}

/* Override executable computed by PyConfig_Read() *
status = PyConfig_SetString(&config, &config.executa
                            L"/path/to/my_executable
if (PyStatus_Exception(status)) {
    goto done;
}

status = Py_InitializeFromConfig(&config);
done:
    PyConfig_Clear(&config);
    return status;
}
```

Configuración aislada

`PyPreConfig_InitIsolatedConfig()` y las funciones **`PyConfig_InitIsolatedConfig()`** crean una configuración para aislar Python del sistema. Por ejemplo, para incrustar Python en una aplicación.

Esta configuración ignora las variables de configuración global, las variables de entorno, los argumentos de la línea de comandos (**`PyConfig.argv`** no se analizan) y el directorio del sitio del usuario. Los flujos estándar de C (por ejemplo, `stdout`) y la configuración regional LC_CTYPE no se modifican. Los manejadores de señales no están instalados.

Los archivos de configuración se siguen utilizando con esta configuración para determinar las rutas que no se especifican. Asegúrese de que se especifica **`PyConfig.home`** para evitar que se calcule la configuración de la ruta por defecto.

Configuración de Python

`PyPreConfig_InitPythonConfig()` y las funciones **`PyConfig_InitPythonConfig()`** crean una configuración para construir un Python personalizado que se comporta como el Python normal.

Las variables de entorno y los argumentos de la línea de comandos se utilizan para configurar Python, mientras que las variables de configuración global se ignoran.

Esta función habilita la coerción de configuración regional C (**PEP 538** [https://peps.python.org/pep-0538/]) y Python UTF-8 Mode (**PEP 540** [https://peps.python.org/pep-0540/]) según la configuración regional LC_CTYPE, las variables de entorno **`PYTHONUTF8`** y

`PYTHONCOERCECLOCALE`.

Configuración de la ruta de Python

`PyConfig` contiene múltiples campos para la configuración de ruta:

- Entradas de configuración de ruta:
  - ○ `PyConfig.home`
  - ○ `PyConfig.platlibdir`
  - ○ `PyConfig.pathconfig_warnings`
  - ○ `PyConfig.program_name`
  - ○ `PyConfig.pythonpath_env`
  - ○ directorio de trabajo actual: para obtener rutas absolutas
  - ○ Variable de entorno `PATH` para obtener la ruta completa del programa (de `PyConfig.program_name`)
  - ○ Variable de entorno `__PYVENV_LAUNCHER__`
  - ○ (Solo Windows) Rutas de aplicación en el registro en «SoftwarePythonPythonCoreX.YPythonPath» de HKEY_CURRENT_USER y HKEY_LOCAL_MACHINE (donde X.Y es la versión de Python).
- Campos de salida de configuración de ruta:
  - ○ `PyConfig.base_exec_prefix`
  - ○ `PyConfig.base_executable`
  - ○ `PyConfig.base_prefix`
  - ○ `PyConfig.exec_prefix`
  - ○ `PyConfig.executable`
  - ○ `PyConfig.module_search_paths_set`, `PyConfig.module_search_paths`
  - ○ `PyConfig.prefix`

Si al menos un «campo de salida» no está establecido, Python calcula la configuración de la ruta para rellenar los campos no establecidos. Si `module_search_paths_set` es igual a `0`, `module_search_paths` se anula y **`module_search_paths_set`** se establece en `1`.

Es posible ignorar completamente la función que calcula la configuración de ruta predeterminada estableciendo explícitamente todos los campos de salida de configuración de ruta mencionados anteriormente. Una cadena se considera establecida incluso si no está vacía. `module_search_paths` se considera establecido si `module_search_paths_set` se establece en `1`. En este caso, `module_search_paths` se utilizará sin modificaciones.

Establezca **`pathconfig_warnings`** en `0` para suprimir las advertencias al calcular la configuración de la ruta (solo Unix, Windows no registra ninguna advertencia).

Si **`base_prefix`** o los campos **`base_exec_prefix`** no están establecidos, heredan su valor de **`prefix`** y **`exec_prefix`** respectivamente.

`Py_RunMain()` y **`Py_Main()`** modifican **`sys.path`**:

- Si **`run_filename`** está configurado y es un directorio que contiene un script `__main__.py`, anteponga **`run_filename`** a **`sys.path`**.
- Si **`isolated`** es cero:
  - ○ Si **`run_module`** está configurado, anteponga el directorio actual a **`sys.path`**. No haga nada si el directorio actual no se puede leer.
  - ○ Si **`run_filename`** está configurado, anteponga el directorio del nombre del archivo a **`sys.path`**.
  - ○ De lo contrario, anteponga una cadena de caracteres vacía a **`sys.path`**.

Si **`site_import`** no es cero, **`sys.path`** puede ser modificado por el módulo **`site`**. Si **`user_site_directory`** no es cero y el directorio del paquete del sitio del usuario existe, el módulo **`site`** agrega el directorio del paquete del sitio del usuario a **`sys.path`**.

La configuración de ruta utiliza los siguientes archivos de configuración:

- `pyvenv.cfg`
- archivo `._pth` (ej: `python._pth`)
- `pybuilddir.txt` (sólo Unix)

Si un archivo `._pth` está presente:

- Establezca **`isolated`** en `1`.
- Establezca **`use_environment`** en `0`.
- Establezca **`site_import`** en `0`.
- Establezca **`safe_path`** en `1`.

If **`home`** is not set and a `pyvenv.cfg` file is present in the same directory as **`executable`**, or its parent, **`prefix`** and **`exec_prefix`** are set that location. When this happens, **`base_prefix`** and **`base_exec_prefix`** still keep their value, pointing to the base installation. See Virtual Environments for more information.

La variable de entorno `__PYVENV_LAUNCHER__` se usa para establecer **`PyConfig.base_executable`**.

Distinto en la versión 3.14: **`prefix`**, and **`exec_prefix`**, are now set to the `pyvenv.cfg` directory. This was previously done by **`site`**, therefore affected by **`-S`**.

Py_GetArgcArgv()

void Py_GetArgcArgv(int *argc, wchar_t ***argv)

Obtiene los argumentos originales de la línea de comandos, antes de que Python los modificara.

Ver también el miembro **`PyConfig.orig_argv`**.

Delaying main module execution

In some embedding use cases, it may be desirable to separate interpreter initialization from the execution of the main module.

This separation can be achieved by setting `PyConfig.run_command` to the empty string during initialization (to prevent the interpreter from dropping into the interactive prompt), and then subsequently executing the desired main module code using `__main__.__dict__` as the global namespace.

Gestión de la memoria

Visión general

La gestión de memoria en Python implica un montón privado que contiene todos los objetos de Python y estructuras de datos. El *administrador de memoria de Python* garantiza internamente la gestión de este montón privado. El administrador de memoria de Python tiene diferentes componentes que se ocupan de varios aspectos de la gestión dinámica del almacenamiento, como compartir, segmentación, asignación previa o almacenamiento en caché.

En el nivel más bajo, un asignador de memoria sin procesar asegura que haya suficiente espacio en el montón privado para almacenar todos los datos relacionados con Python al interactuar con el administrador de memoria del sistema operativo. Además del asignador de memoria sin procesar, varios asignadores específicos de objeto operan en el mismo montón e implementan políticas de administración de memoria distintas adaptadas a las peculiaridades de cada tipo de objeto. Por ejemplo, los objetos enteros se administran de manera diferente dentro del montón que las cadenas, tuplas o diccionarios porque los enteros implican diferentes requisitos de almacenamiento y compensaciones de velocidad / espacio. El administrador de memoria de Python delega parte del trabajo a los asignadores específicos de objeto, pero asegura que este último opere dentro de los límites del montón privado.

Es importante comprender que la gestión del montón de Python la realiza el propio intérprete y que el usuario no tiene control sobre él, incluso si manipulan regularmente punteros de objetos a bloques de memoria dentro de ese montón. El administrador de memoria de Python realiza la asignación de espacio de almacenamiento dinámico para los objetos de Python y otros búferes internos a pedido a través de las funciones de API de Python/C enumeradas en este documento.

Para evitar daños en la memoria, los escritores de extensiones nunca deberían intentar operar en objetos Python con las funciones exportadas por la biblioteca C: **`malloc()`**, **`calloc()`**, **`realloc()`** y **`free()`**. Esto dará como resultado llamadas mixtas entre el asignador de C y el administrador de memoria de Python con consecuencias fatales, ya que implementan diferentes algoritmos y operan en diferentes montones. Sin embargo, uno puede asignar y liberar de forma segura bloques de memoria con el asignador de la biblioteca C para fines individuales, como se muestra en el siguiente ejemplo:

```
PyObject *res;
char *buf = (char *) malloc(BUFSIZ); /* para I/O */

if (buf == NULL)
    return PyErr_NoMemory();
...Hacer algunas operaciones I/O incluyendo buf...
res = PyBytes_FromString(buf);
free(buf); /* malloc'ed */
return res;
```

En este ejemplo, la solicitud de memoria para el búfer de E/S es manejada por el asignador de la biblioteca C. El administrador de memoria de Python solo participa en la asignación del objeto de bytes retornado como resultado.

Sin embargo, en la mayoría de las situaciones, se recomienda asignar memoria del montón de Python específicamente porque este último está bajo el control del administrador de memoria de Python. Por ejemplo, esto es necesario cuando el intérprete se amplía con nuevos tipos de objetos escritos en C. Otra razón para usar el montón de Python es el deseo de *informar* al administrador de memoria de Python sobre las necesidades de memoria del módulo de extensión. Incluso cuando la memoria solicitada se usa exclusivamente para fines internos y altamente específicos, delegar todas las solicitudes de memoria al administrador de memoria de Python hace que el intérprete tenga una imagen más precisa de su huella de memoria en su conjunto. En consecuencia, bajo ciertas circunstancias, el administrador de memoria de Python puede o no desencadenar acciones apropiadas, como recolección de basura, compactación de memoria u otros procedimientos preventivos. Tenga en cuenta que al usar el asignador de la biblioteca C como se muestra en el ejemplo anterior, la memoria asignada para el búfer de E/S escapa completamente al administrador de memoria Python.

Ver también

La variable de entorno `PYTHONMALLOC` puede usarse para configurar los asignadores de memoria utilizados por Python.

La variable de entorno `PYTHONMALLOCSTATS` se puede utilizar para imprimir estadísticas de asignador de memoria pymalloc cada vez que se crea un nuevo escenario de objetos pymalloc, y en el apagado.

Dominios del asignador

All allocating functions belong to one of three different «domains» (see also `PyMemAllocatorDomain`). These domains represent different allocation strategies and are optimized for different purposes. The specific details on how every domain allocates memory or what internal functions each domain calls is considered an implementation detail, but for debugging purposes a simplified table can be found at Asignadores de memoria predeterminados. The APIs used to allocate and free a block of memory must be from the same domain. For example, `PyMem_Free()` must be used to free memory allocated using `PyMem_Malloc()`.

Los tres dominios de asignación son:

- Raw domain: intended for allocating memory for general-purpose memory buffers where the allocation *must* go to the system allocator or where the allocator can operate without an attached thread state. The memory is requested directly from the system. See Raw Memory Interface.
- «Mem» domain: intended for allocating memory for Python buffers and general-purpose memory buffers where the

allocation must be performed with an attached thread state. The memory is taken from the Python private heap. See Memory Interface.

- Dominio de objeto: destinado a asignar memoria perteneciente a objetos de Python. La memoria se toma del montículo privado de Python. Consulte Asignadores de objetos.

Nota

La compilación free-threaded requiere que solamente objetos de Python se asignen usando el dominio de «object» y que todos los objetos de Python se asignan usando ese dominio. Esto es diferente que los versiones anteriores de Python, donde esto fue una práctica buena y no un requisito.

Por ejemplo, *buffers* (objetos que no son de Python) deben asignarse usando **`PyMem_Malloc()`**, **`PyMem_RawMalloc()`**, o **`malloc()`**, pero no **`PyObject_Malloc()`**.

Consulte APIs de Asignación de Memoria.

Interfaz de memoria sin procesar

The following function sets are wrappers to the system allocator. These functions are thread-safe, so a thread state does not need to be attached.

El asignador de memoria sin formato usa las siguientes funciones: **`malloc()`**, **`calloc()`**, **`realloc()`** y **`free()`**; llame a `malloc(1)` (o `calloc(1, 1)`) cuando solicita cero bytes.

Added in version 3.4.

void *PyMem_RawMalloc(size_t n)

Part of the Stable ABI since version 3.13.
Asigna *n* bytes y retorna un puntero de tipo void* a la memoria asignada, o `NULL` si la solicitud falla.

Solicitar cero bytes retorna un puntero distinto que no sea `NULL` si es posible, como si en su lugar se hubiera llamado a `PyMem_RawMalloc(1)`. La memoria no se habrá inicializado de ninguna manera.

void *PyMem_RawCalloc(size_t nelem, size_t elsize)

Part of the Stable ABI since version 3.13.
Asigna *nelem* elementos cada uno cuyo tamaño en bytes es *elsize* y retorna un puntero de tipo void* a la memoria asignada, o `NULL` si la solicitud falla. La memoria se inicializa a ceros.

Solicitar elementos cero o elementos de tamaño cero bytes retorna un puntero distinto `NULL` si es posible, como si en su lugar se hubiera llamado `PyMem_RawCalloc(1, 1)`.

Added in version 3.5.

void *PyMem_RawRealloc(void *p, size_t n)

Part of the Stable ABI since version 3.13.
Cambia el tamaño del bloque de memoria señalado por *p* a *n* bytes. Los contenidos no se modificarán al mínimo de los tamaños antiguo y nuevo.

Si *p* es `NULL`, la llamada es equivalente a `PyMem_RawMalloc(n)`; de lo contrario, si *n* es igual a cero, el bloque de memoria cambia de tamaño pero no se libera, y el puntero retornado no es `NULL`.

A menos que *p* sea `NULL`, debe haber sido retornado por una llamada previa a **`PyMem_RawMalloc()`**, **`PyMem_RawRealloc()`** o **`PyMem_RawCalloc()`**.

Si la solicitud falla, **`PyMem_RawRealloc()`** retorna `NULL` y *p* sigue siendo un puntero válido al área de memoria anterior.

void PyMem_RawFree(void *p)

Part of the Stable ABI since version 3.13.
Libera el bloque de memoria al que apunta *p*, que debe haber

sido retornado por una llamada anterior a **`PyMem_RawMalloc()`**, **`PyMem_RawRealloc()`** o **`PyMem_RawCalloc()`**. De lo contrario, o si se ha llamado antes a `PyMem_RawFree(p)`, se produce un comportamiento indefinido.

Si *p* es `NULL`, no se realiza ninguna operación.

Interfaz de memoria

Los siguientes conjuntos de funciones, modelados según el estándar ANSI C, pero que especifican el comportamiento cuando se solicitan cero bytes, están disponibles para asignar y liberar memoria del montón de Python.

In the GIL-enabled build (default build) the default memory allocator uses the pymalloc memory allocator, whereas in the free-threaded build, the default is the mimalloc memory allocator instead.

Advertencia

There must be an attached thread state when using these functions.

Distinto en la versión 3.6: El asignador predeterminado ahora es pymalloc en lugar del **`malloc()`** del sistema.

Distinto en la versión 3.13: In the free-threaded build, the default allocator is now mimalloc.

void *PyMem_Malloc(size_t n)

Part of the Stable ABI.
Asigna *n* bytes y retorna un puntero de tipo void* a la memoria asignada, o `NULL` si la solicitud falla.

Solicitar cero bytes retorna un puntero distinto que no sea `NULL` si es posible, como si en su lugar se hubiera llamado a `PyMem_Malloc(1)`. La memoria no se habrá inicializado de ninguna manera.

void *PyMem_Calloc(size_t nelem, size_t elsize)

Part of the Stable ABI since version 3.7.
Asigna *nelem* elementos cada uno cuyo tamaño en bytes es *elsize* y retorna un puntero de tipo void* a la memoria asignada, o `NULL` si la solicitud falla. La memoria se inicializa a ceros.

Solicitar elementos cero o elementos de tamaño cero bytes retorna un puntero distinto `NULL` si es posible, como si en su lugar se hubiera llamado `PyMem_Calloc(1, 1)`.

Added in version 3.5.

void *PyMem_Realloc(void *p, size_t n)

Part of the Stable ABI.
Cambia el tamaño del bloque de memoria señalado por *p* a *n* bytes. Los contenidos no se modificarán al mínimo de los tamaños antiguo y nuevo.

Si *p* es `NULL`, la llamada es equivalente a `PyMem_Malloc(n)`; de lo contrario, si *n* es igual a cero, el bloque de memoria cambia de tamaño pero no se libera, y el puntero retornado no es `NULL`.

A menos que *p* sea `NULL`, debe haber sido retornado por una llamada previa a **`PyMem_Malloc()`**, **`PyMem_Realloc()`** o **`PyMem_Calloc()`**.

Si la solicitud falla, **`PyMem_Realloc()`** retorna `NULL` y *p* sigue siendo un puntero válido al área de memoria anterior.

void PyMem_Free(void *p)

Part of the Stable ABI.
Libera el bloque de memoria señalado por *p*, que debe haber sido retornado por una llamada anterior a **`PyMem_Malloc()`**, **`PyMem_Realloc()`** o **`PyMem_Calloc()`**. De lo contrario, o si se ha llamado antes a

`PyMem_Free(p)`, se produce un comportamiento indefinido.

Si *p* es `NULL`, no se realiza ninguna operación.

Las siguientes macros orientadas a tipos se proporcionan por conveniencia. Tenga en cuenta que *TYPE* se refiere a cualquier tipo de C.

PyMem_New(TYPE, n)

Igual que **`PyMem_Malloc()`**, pero asigna `(n * sizeof(TYPE))` bytes de memoria. Retorna una conversión de puntero a `TYPE*`. La memoria no se habrá inicializado de ninguna manera.

PyMem_Resize(p, TYPE, n)

Igual que **`PyMem_Realloc()`**, pero el bloque de memoria cambia de tamaño a `(n * sizeof(TYPE))` bytes. Retorna una conversión de puntero a `TYPE*`. Al retornar, *p* será un puntero a la nueva área de memoria, o `NULL` en caso de falla.

Esta es una macro de preprocesador C; *p* siempre se reasigna. Guarde el valor original de *p* para evitar perder memoria al manejar errores.

void PyMem_Del(void *p)

La misma que **`PyMem_Free()`**.

Deprecated aliases

These are soft deprecated aliases to existing functions and macros. They exist solely for backwards compatibility.

Deprecated alias	Corresponding function or macro
PyMem_MALLOC(size)	**`PyMem_Malloc()`**
PyMem_NEW(type, size)	**`PyMem_New`**
PyMem_REALLOC(ptr, size)	**`PyMem_Realloc()`**
PyMem_RESIZE(ptr, type, size)	**`PyMem_Resize`**
PyMem_FREE(ptr)	**`PyMem_Free()`**

PyMem_DEL(ptr)

PyMem_Free()

Distinto en la versión 3.4: The macros are now aliases of the corresponding functions and macros. Previously, their behavior was the same, but their use did not necessarily preserve binary compatibility across Python versions.

Obsoleto desde la versión 2.0.

Asignadores de objetos

Los siguientes conjuntos de funciones, modelados según el estándar ANSI C, pero que especifican el comportamiento cuando se solicitan cero bytes, están disponibles para asignar y liberar memoria del montón de Python.

Nota

No hay garantía de que la memoria retornada por estos asignadores se pueda convertir con éxito en un objeto Python al interceptar las funciones de asignación en este dominio mediante los métodos descritos en la sección Personalizar Asignadores de Memoria.

The default object allocator uses the pymalloc memory allocator. In the free-threaded build, the default is the mimalloc memory allocator instead.

Advertencia

There must be an attached thread state when using these functions.

void *PyObject_Malloc(size_t n)

Part of the Stable ABI.
Asigna *n* bytes y retorna un puntero de tipo void* a la memoria asignada, o `NULL` si la solicitud falla.

Solicitar cero bytes retorna un puntero distinto que no sea `NULL` si es posible, como si en su lugar se hubiera llamado a `PyObject_Malloc(1)`. La memoria no se habrá inicializado de ninguna manera.

void *PyObject_Calloc(size_t nelem, size_t elsize)

Part of the Stable ABI since version 3.7.
Asigna *nelem* elementos cada uno cuyo tamaño en bytes es *elsize* y retorna un puntero de tipo void* a la memoria asignada, o `NULL` si la solicitud falla. La memoria se inicializa a ceros.

Solicitar elementos cero o elementos de tamaño cero bytes retorna un puntero distinto `NULL` si es posible, como si en su lugar se hubiera llamado `PyObject_Calloc(1, 1)`.

Added in version 3.5.

void *PyObject_Realloc(void *p, size_t n)

Part of the Stable ABI.
Cambia el tamaño del bloque de memoria señalado por *p* a *n* bytes. Los contenidos no se modificarán al mínimo de los tamaños antiguo y nuevo.

Si *p* es `NULL`, la llamada es equivalente a `PyObject_Malloc(n)`; de lo contrario, si *n* es igual a cero, el bloque de memoria cambia de tamaño pero no se libera, y el puntero retornado no es `NULL`.

A menos que *p* sea `NULL`, debe haber sido retornado por una llamada previa a **`PyObject_Malloc()`**, **`PyObject_Realloc()`** o **`PyObject_Calloc()`**.

Si la solicitud falla, **`PyObject_Realloc()`** retorna `NULL` y *p* sigue siendo un puntero válido al área de memoria anterior.

void PyObject_Free(void *p)

Part of the Stable ABI.
Libera el bloque de memoria al que apunta *p*, que debe haber

sido retornado por una llamada anterior a **`PyObject_Malloc()`**, **`PyObject_Realloc()`** o **`PyObject_Calloc()`**. De lo contrario, o si se ha llamado antes a `PyObject_Free(p)`, se produce un comportamiento indefinido.

Si *p* es `NULL`, no se realiza ninguna operación.

Do not call this directly to free an object's memory; call the type's **`tp_free`** slot instead.

Do not use this for memory allocated by **`PyObject_GC_New`** or **`PyObject_GC_NewVar`**; use **`PyObject_GC_Del()`** instead.

Ver también

- **`PyObject_GC_Del()`** is the equivalent of this function for memory allocated by types that support garbage collection.
- **`PyObject_Malloc()`**
- **`PyObject_Realloc()`**
- **`PyObject_Calloc()`**
- **`PyObject_New`**
- **`PyObject_NewVar`**
- **`PyType_GenericAlloc()`**
- **`tp_free`**

Asignadores de memoria predeterminados

Asignadores de memoria predeterminados:

Configuración	Nombre	PyMem_RawMalloc	PyMem_Malloc	PyObject_Malloc
Lanzamiento de compilación	`"pymalloc"`	`malloc`	`pymalloc`	`pymalloc`
Compilación de depuración	`"pymalloc_debug"`	`malloc` + debug	`pymalloc` + debug	`pymalloc` + debug
Lanzamiento de compilación, sin pymalloc	`"malloc"`	`malloc`	`malloc`	`malloc`
Compilación de depuración, sin pymalloc	`"malloc_debug"`	`malloc` + debug	`malloc` + debug	`malloc` + debug
Free-threaded build	`"mimalloc"`	`mimalloc`	`mimalloc`	`mimalloc`

`mimalloc_debug`	Freethreaded **"debug"** build

Leyenda:

- Nombre: valor para variable de entorno **`PYTHONMALLOC`**.
- `malloc`: asignadores del sistema de la biblioteca C estándar, funciones C: **`malloc()`**, **`calloc()`**, **`realloc()`** y **`free()`**.
- `pymalloc`: asignador de memoria pymalloc.
- `mimalloc`: mimalloc memory allocator.
- «+ debug»: con enlaces de depuración en los asignadores de memoria de Python.
- «Debug build»: Compilación de Python en modo de depuración.

Personalizar asignadores de memoria

Added in version 3.4.

type PyMemAllocatorEx

Estructura utilizada para describir un asignador de bloque de memoria. La estructura tiene cuatro campos:

Campo	Significado
`void *ctx`	contexto de usuario pasado como primer argumento
`void* malloc(void *ctx, size_t size)`	asignar un bloque de memoria
`void* calloc(void *ctx, size_t nelem, size_t elsize)`	asignar un bloque de memoria, inicializado con ceros
`void* realloc(void *ctx, void *ptr, size_t new_size)`	asignar o cambiar el tamaño de un bloque de memoria
`void free(void *ctx, void *ptr)`	liberar un bloque de memoria

Distinto en la versión 3.5: La estructura **`PyMemAllocator`** se renombró a **`PyMemAllocatorEx`** y se agregó un nuevo campo `calloc`.

type PyMemAllocatorDomain

Enum se utiliza para identificar un dominio asignador. Dominios:

PYMEM_DOMAIN_RAW

Funciones:

- **`PyMem_RawMalloc()`**
- **`PyMem_RawRealloc()`**
- **`PyMem_RawCalloc()`**
- **`PyMem_RawFree()`**

PYMEM_DOMAIN_MEM

Funciones:

- **`PyMem_Malloc()`**,
- **`PyMem_Realloc()`**
- **`PyMem_Calloc()`**
- **`PyMem_Free()`**

PYMEM_DOMAIN_OBJ

Funciones:

- **`PyObject_Malloc()`**
- **`PyObject_Realloc()`**
- **`PyObject_Calloc()`**
- **`PyObject_Free()`**

void PyMem_GetAllocator(PyMemAllocatorDomain domain, PyMemAllocatorEx *allocator)

Obtenga el asignador de bloque de memoria del dominio especificado.

void PyMem_SetAllocator(PyMemAllocatorDomain domain, PyMemAllocatorEx *allocator)

Establece el asignador de bloque de memoria del dominio especificado.

El nuevo asignador debe retornar un puntero distinto `NULL` al solicitar cero bytes.

For the **`PYMEM_DOMAIN_RAW`** domain, the allocator must be thread-safe: a thread state is not attached when the allocator is called.

For the remaining domains, the allocator must also be thread-safe: the allocator may be called in different interpreters that do not share a GIL.

Si el nuevo asignador no es un enlace (no llama al asignador anterior), se debe llamar a la función `PyMem_SetupDebugHooks()` para reinstalar los enlaces de depuración en la parte superior del nuevo asignador.

Vea también `PyPreConfig.allocator` y Preinicialización de Python con PyPreConfig.

Advertencia

`PyMem_SetAllocator()` tiene el contrato siguiente:

- It can be called after `Py_PreInitialize()` and before `Py_InitializeFromConfig()` to install a custom memory allocator. There are no restrictions over the installed allocator other than the ones imposed by the domain (for instance, the Raw Domain allows the allocator to be called without an attached thread state). See the section on allocator domains for more information.
- Si se llama después de que Python haya terminado inicializándose (después de llamar `Py_InitializeFromConfig()`) el asignador **debe** envolver el asignador existente. Substituyendo el asignador actual con otro arbitrario **no es compatible**.

Distinto en la versión 3.12: Todos los asignadores deben ser seguro para los hilos.

void PyMem_SetupDebugHooks(void)

Configurar enlaces de depuración en los asignadores de memoria de Python para detectar errores de memoria.

Configurar enlaces para detectar errores en las funciones del asignador de memoria de Python

Cuando Python está construido en modo de depuración, la función **`PyMem_SetupDebugHooks()`** se llama en Preinicialización de Python para configurar los enlaces de depuración en Python asignadores de memoria para detectar errores de memoria.

La variable de entorno **`PYTHONMALLOC`** se puede utilizar para instalar enlaces de depuración en un Python compilado en modo de lanzamiento (por ejemplo: `PYTHONMALLOC=debug`).

La función **`PyMem_SetupDebugHooks()`** se puede utilizar para establecer enlaces de depuración después de llamar a **`PyMem_SetAllocator()`**.

Estos enlaces de depuración llenan bloques de memoria asignados dinámicamente con patrones de bits especiales y reconocibles. La memoria recién asignada se llena con el byte `0xCD` (`PYMEM_CLEANBYTE`), la memoria liberada se llena con el byte `0xDD` (`PYMEM_DEADBYTE`). Los bloques de memoria están rodeados por «bytes prohibidos» rellenos con el byte `0xFD` (`PYMEM_FORBIDDENBYTE`). Es poco probable que las cadenas de estos bytes sean direcciones válidas, flotantes o cadenas ASCII.

Verificaciones de tiempo de ejecución:

- Detecte violaciones de API, por ejemplo: **`PyObject_Free()`** llamado en un búfer asignado por **`PyMem_Malloc()`**.
- Detectar escritura antes del inicio del búfer (desbordamiento del búfer)
- Detectar escritura después del final del búfer (desbordamiento del búfer)
- Check that there is an attached thread state when allocator functions of **`PYMEM_DOMAIN_OBJ`** (ex: **`PyObject_Malloc()`**) and **`PYMEM_DOMAIN_MEM`** (ex: **`PyMem_Malloc()`**) domains are called.

En caso de error, los enlaces de depuración usan el módulo **`tracemalloc`** para obtener el rastreo donde se asignó un bloque de memoria. El rastreo solo se muestra si **`tracemalloc`** rastrea las asignaciones de memoria de Python y se rastrea el bloque de memoria.

Sea S = `sizeof(size_t)`. Se agregan `2*S` bytes en cada extremo de cada bloque de N bytes solicitados. El diseño de la memoria es así, donde p representa la dirección retornada por una función similar a malloc o realloc (`p[i:j]` significa el segmento de bytes de `*(p+i)` inclusive hasta `*(p+j)` exclusivo; tenga en cuenta que el tratamiento de los índices negativos difiere de un segmento de Python):

`p[-2*S:-S]`

Número de bytes solicitados originalmente. Este es un size_t, big-endian (más fácil de leer en un volcado de memoria).

`p[-S]`

Identificador de API (carácter ASCII):

- `'r'` para **`PYMEM_DOMAIN_RAW`**.
- `'m'` para **`PYMEM_DOMAIN_MEM`**.
- `'o'` para **`PYMEM_DOMAIN_OBJ`**.

`p[-S+1:0]`

Copias de PYMEM_FORBIDDENBYTE. Se utiliza para detectar suscripciones y lecturas.

`p[0:N]`

La memoria solicitada, llena de copias de PYMEM_CLEANBYTE, utilizada para capturar la referencia a la memoria no inicializada. Cuando se llama a una función similar a realloc solicitando un bloque de memoria más grande, los nuevos bytes en exceso también se llenan con PYMEM_CLEANBYTE. Cuando se llama a una función de tipo free, se sobrescriben con PYMEM_DEADBYTE, para captar la referencia a la memoria liberada. Cuando se llama a una función similar a la realloc solicitando un bloque de memoria más pequeño, los bytes antiguos sobrantes también se llenan con PYMEM_DEADBYTE.

`p[N:N+S]`

Copias de PYMEM_FORBIDDENBYTE. Se utiliza para detectar sobrescrituras y lecturas.

`p[N+S:N+2*S]`

Solo se utiliza si la macro `PYMEM_DEBUG_SERIALNO` está definida (no definida por defecto).

Un número de serie, incrementado en 1 en cada llamada a una función similar a malloc o realloc. Big-endian **`size_t`**. Si se detecta «mala memoria» más tarde, el número de serie ofrece una excelente manera de establecer un punto de interrupción en la siguiente ejecución, para capturar el instante en el que se pasó este bloque. La función estática bumpserialno() en obmalloc.c es el único lugar donde se incrementa el número de serie, y existe para que pueda establecer un punto de interrupción fácilmente.

Una función de tipo realloc o de tipo free primero verifica que los bytes PYMEM_FORBIDDENBYTE en cada extremo estén intactos. Si se han modificado, la salida de diagnóstico se escribe en stderr y el programa se aborta mediante Py_FatalError(). El otro modo de falla principal es provocar un error de memoria cuando un programa lee uno de los patrones de bits especiales e intenta usarlo como una dirección. Si ingresa a un depurador y observa el objeto, es probable que vea que está completamente lleno de PYMEM_DEADBYTE (lo que significa que se está usando la memoria liberada) o PYMEM_CLEANBYTE (que significa que se está usando la memoria no inicializada).

Distinto en la versión 3.6: The **`PyMem_SetupDebugHooks()`** function now also works on Python compiled in release mode. On error, the debug hooks now use **`tracemalloc`** to get the traceback where a memory block was allocated. The debug hooks now also check if there is an attached thread state when functions of **`PYMEM_DOMAIN_OBJ`** and **`PYMEM_DOMAIN_MEM`** domains are called.

Distinto en la versión 3.8: Los patrones de bytes `0xCB` (`PYMEM_CLEANBYTE`), `0xDB` (`PYMEM_DEADBYTE`) y `0xFB` (`PYMEM_FORBIDDENBYTE`) se han reemplazado por `0xCD`, `0xDD` y `0xFD` para usar los mismos valores que la depuración de Windows CRT `malloc()` y `free()`.

El asignador pymalloc

Python tiene un asignador *pymalloc* optimizado para objetos pequeños (más pequeños o iguales a 512 bytes) con una vida útil corta. Utiliza asignaciones de memoria llamadas «arenas» con un tamaño fijo de 256 KiB en plataformas de 32 bits o 1 MiB en plataformas de 64 bits. Vuelve a **`PyMem_RawMalloc()`** y **`PyMem_RawRealloc()`** para asignaciones de más de 512 bytes.

pymalloc es el asignador por defecto de **`PYMEM_DOMAIN_MEM`** (por ejemplo: **`PyMem_Malloc()`**) y **`PYMEM_DOMAIN_OBJ`** (por ejemplo: **`PyObject_Malloc()`**) dominios.

El asignador de arena utiliza las siguientes funciones:

- **`VirtualAlloc()`** y **`VirtualFree()`** en Windows,
- **`mmap()`** y **`munmap()`** si está disponible,
- **`malloc()`** y **`free()`** en caso contrario.

Este asignador está deshabilitado si Python está configurado con la opción **`--without-pymalloc`**. También se puede deshabilitar en tiempo de ejecución usando la variable de entorno **`PYTHONMALLOC`** (por ejemplo: `PYTHONMALLOC=malloc`).

Typically, it makes sense to disable the pymalloc allocator when building Python with AddressSanitizer (**`--with-address-sanitizer`**) which helps uncover low level bugs within the C code.

Personalizar asignador de arena de pymalloc

Added in version 3.4.

type PyObjectArenaAllocator

Estructura utilizada para describir un asignador de arena. La estructura tiene tres campos:

Campo	Significado
`void *ctx`	contexto de usuario pasado como primer argumento
`void* alloc(void *ctx, size_t size)`	asignar una arena de bytes de tamaño
`void free(void *ctx, void *ptr, size_t size)`	liberar la arena

void PyObject_GetArenaAllocator(PyObjectArenaAllocator *allocator)

Consigue el asignador de arena.

void PyObject_SetArenaAllocator(PyObjectArenaAllocator *allocator)

Establecer el asignador de arena.

El asignador mimalloc

Added in version 3.13.

Python supports the mimalloc [https://github.com/microsoft/mimalloc/] allocator when the underlying platform support is available. mimalloc is a general purpose allocator with excellent performance characteristics, initially developed by Daan Leijen for the runtime systems of the Koka and Lean languages.

Unlike pymalloc, which is optimized for small objects (512 bytes or fewer), mimalloc handles allocations of any size.

In the free-threaded build, mimalloc is the default and **required** allocator for the **`PYMEM_DOMAIN_MEM`** and **`PYMEM_DOMAIN_OBJ`** domains. It cannot be disabled in free-threaded builds. The free-threaded build uses per-thread mimalloc heaps, which allows allocation and deallocation to proceed without locking in most cases.

In the default (non-free-threaded) build, mimalloc is available but not the default allocator. It can be selected at runtime using **`PYTHONMALLOC`**`=mimalloc` (or `mimalloc_debug` to include debug hooks). It can be disabled at build time using the **`--`**

`without-mimalloc` configure option, but this option cannot be combined with `--disable-gil`.

tracemalloc C API

Added in version 3.7.

int PyTraceMalloc_Track(unsigned int domain, uintptr_t ptr, size_t size)

> Rastree un bloque de memoria asignado en el módulo `tracemalloc`.
>
> Retorna `0` en caso de éxito, retorna `-1` en caso de error (no se pudo asignar memoria para almacenar la traza). Retorna `-2` si tracemalloc está deshabilitado.
>
> Si el bloque de memoria ya está rastreado, actualice el rastreo existente.

int PyTraceMalloc_Untrack(unsigned int domain, uintptr_t ptr)

> Descomprima un bloque de memoria asignado en el módulo `tracemalloc`. No haga nada si el bloque no fue rastreado.
>
> Retorna `-2` si tracemalloc está deshabilitado; de lo contrario, retorna `0`.

Ejemplos

Aquí está el ejemplo de la sección Visión general, reescrito para que el búfer de E/S se asigne desde el montón de Python utilizando el primer conjunto de funciones:

```
PyObject *res;
char *buf = (char *) PyMem_Malloc(BUFSIZ); /* para I/O *

if (buf == NULL)
    return PyErr_NoMemory();
/* ...Hacer algunas operaciones I/O incluyendo buf... */
```

```
res = PyBytes_FromString(buf);
PyMem_Free(buf); /* destinado con PyMem_Malloc */
return res;
```

El mismo código que utiliza el conjunto de funciones orientado a tipos:

```
PyObject *res;
char *buf = PyMem_New(char, BUFSIZ); /* for I/O */

if (buf == NULL)
    return PyErr_NoMemory();
/* ...Do some I/O operation involving buf... */
res = PyBytes_FromString(buf);
PyMem_Free(buf); /* allocated with PyMem_New */
return res;
```

Tenga en cuenta que en los dos ejemplos anteriores, el búfer siempre se manipula a través de funciones que pertenecen al mismo conjunto. De hecho, es necesario usar la misma familia de API de memoria para un bloque de memoria dado, de modo que el riesgo de mezclar diferentes asignadores se reduzca al mínimo. La siguiente secuencia de código contiene dos errores, uno de los cuales está etiquetado como *fatal* porque mezcla dos asignadores diferentes que operan en montones diferentes.:

```
char *buf1 = PyMem_New(char, BUFSIZ);
char *buf2 = (char *) malloc(BUFSIZ);
char *buf3 = (char *) PyMem_Malloc(BUFSIZ);
...
PyMem_Del(buf3);  /* Wrong -- should be PyMem_Free()  */
free(buf2);       /* Right -- allocated via malloc()  */
free(buf1);       /* Fatal -- should be PyMem_Free()   */
```

In addition to the functions aimed at handling raw memory blocks from the Python heap, objects in Python are allocated and released with **`PyObject_New`**, **`PyObject_NewVar`** and **`PyObject_Free()`**.

Esto se explicará en el próximo capítulo sobre cómo definir e implementar nuevos tipos de objetos en C.

Soporte de implementación de objetos

Este capítulo describe las funciones, los tipos y las macros utilizados al definir nuevos tipos de objetos.

- Allocating objects on the heap
    - Soft-deprecated aliases
- Object Life Cycle
    - Life Events
    - Cyclic Isolate Destruction
    - Functions
- Estructuras de objetos comunes
    - Tipos objeto base y macros
    - Implementando funciones y métodos
    - Acceder a atributos de tipos de extensión
        - Member flags
        - Member types
        - Defining Getters and Setters
- Type Object Structures
    - Referencia rápida
        - «ranuras *tp*» (*tp slots*)
        - sub-ranuras (*sub-slots*)
        - ranura de *typedefs*
    - Definición de **`PyTypeObject`**
    - Ranuras (*Slots*) **`PyObject`**
    - Ranuras **`PyVarObject`**

  - Ranuras `PyTypeObject`
  - Tipos estáticos
  - Tipos Heap
  - Estructuras de objetos de números
  - Estructuras de objetos mapeo
  - Estructuras de objetos secuencia
  - Estructuras de objetos búfer
  - Estructuras de objetos asíncronos
  - Tipo Ranura *typedefs*
  - Ejemplos
- Apoyo a la recolección de basura cíclica
  - Controlar el estado del recolector de basura
  - Consultar el estado del recolector de basura

Allocating objects on the heap

PyObject *_PyObject_New(PyTypeObject *type)

Return value: New reference.

PyVarObject *_PyObject_NewVar(PyTypeObject *type, Py_ssize_t size)

Return value: New reference.

PyObject *PyObject_Init(PyObject *op, PyTypeObject *type)

Return value: Borrowed reference. Part of the Stable ABI. Initialize a newly allocated object *op* with its type and initial reference. Returns the initialized object. Other fields of the object are not initialized. Despite its name, this function is unrelated to the object's `__init__()` method (`tp_init` slot). Specifically, this function does **not** call the object's `__init__()` method.

In general, consider this function to be a low-level routine. Use `tp_alloc` where possible. For implementing `tp_alloc` for your type, prefer `PyType_GenericAlloc()` or `PyObject_New()`.

Nota

This function only initializes the object's memory corresponding to the initial `PyObject` structure. It does not zero the rest.

PyVarObject *PyObject_InitVar(PyVarObject *op, PyTypeObject *type, Py_ssize_t size)

Return value: Borrowed reference. Part of the Stable ABI.

Esto hace todo lo que `PyObject_Init()` hace, y también inicializa la información de longitud para un objeto de tamaño variable.

Nota

This function only initializes some of the object's memory. It does not zero the rest.

PyObject_New(TYPE, typeobj)

Allocates a new Python object using the C structure type *TYPE* and the Python type object *typeobj* (`PyTypeObject*`) by calling `PyObject_Malloc()` to allocate memory and initializing it like `PyObject_Init()`. The caller will own the only reference to the object (i.e. its reference count will be one).

Avoid calling this directly to allocate memory for an object; call the type's `tp_alloc` slot instead.

When populating a type's `tp_alloc` slot, `PyType_GenericAlloc()` is preferred over a custom function that simply calls this macro.

This macro does not call `tp_alloc`, `tp_new` (`__new__()`), or `tp_init` (`__init__()`).

This cannot be used for objects with `Py_TPFLAGS_HAVE_GC` set in `tp_flags`; use `PyObject_GC_New` instead.

Memory allocated by this macro must be freed with `PyObject_Free()` (usually called via the object's `tp_free` slot).

Nota

The returned memory is not guaranteed to have been completely zeroed before it was initialized.

Nota

This macro does not construct a fully initialized object of the given type; it merely allocates memory and prepares it for further initialization by `tp_init`. To construct a fully initialized object, call *typeobj* instead. For example:

```
PyObject *foo = PyObject_CallNoArgs((PyObject *)&P
```

Ver también

- `PyObject_Free()`
- `PyObject_GC_New`
- `PyType_GenericAlloc()`
- `tp_alloc`

PyObject_NewVar(TYPE, typeobj, size)

Like `PyObject_New` except:

- It allocates enough memory for the *TYPE* structure plus *size* (`Py_ssize_t`) fields of the size given by the `tp_itemsize` field of *typeobj*.
- The memory is initialized like `PyObject_InitVar()`.

This is useful for implementing objects like tuples, which are able to determine their size at construction time. Embedding the array of fields into the same allocation decreases the number of allocations, improving the memory management efficiency.

Avoid calling this directly to allocate memory for an object; call the type's `tp_alloc` slot instead.

When populating a type's `tp_alloc` slot, `PyType_GenericAlloc()` is preferred over a custom function that simply calls this macro.

This cannot be used for objects with `Py_TPFLAGS_HAVE_GC`

set in `tp_flags`; use `PyObject_GC_NewVar` instead.

Memory allocated by this function must be freed with `PyObject_Free()` (usually called via the object's `tp_free` slot).

Nota

The returned memory is not guaranteed to have been completely zeroed before it was initialized.

Nota

This macro does not construct a fully initialized object of the given type; it merely allocates memory and prepares it for further initialization by `tp_init`. To construct a fully initialized object, call *typeobj* instead. For example:

```
PyObject *list_instance = PyObject_CallNoArgs((PyO
```

Ver también

- `PyObject_Free()`
- `PyObject_GC_NewVar`
- `PyType_GenericAlloc()`
- `tp_alloc`

PyObject _Py_NoneStruct

Objeto que es visible en Python como `None`. Esto solo se debe acceder utilizando el macro `Py_None`, que se evalúa como un puntero a este objeto.

Ver también

Objetos módulo

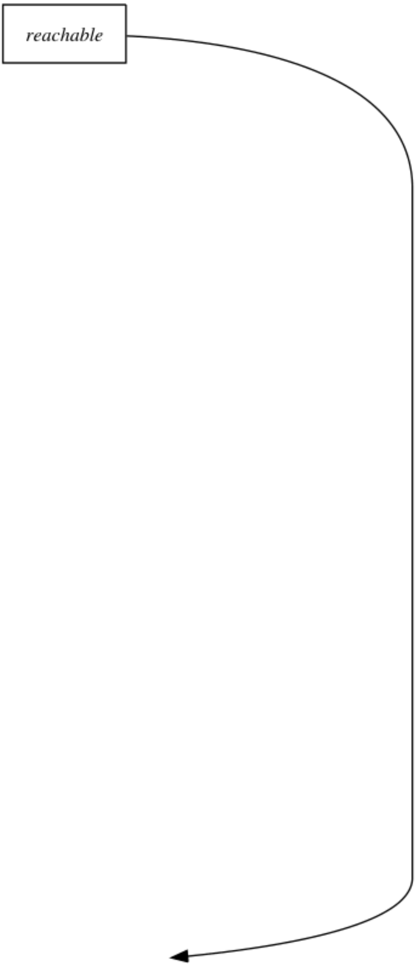
Para asignar y crear módulos de extensión.

Soft-deprecated aliases

Soft deprecated since version 3.10.

These are aliases to existing functions and macros. They exist solely for backwards compatibility.

Soft deprecated alias	Function
PyObject_NEW(type, typeobj)	`PyObject_New`
PyObject_NEW_VAR(type, typeobj, n)	`PyObject_NewVar`
PyObject_INIT(op, typeobj)	`PyObject_Init()`
PyObject_INIT_VAR(op, typeobj, n)	`PyObject_InitVar()`
PyObject_MALLOC(n)	`PyObject_Malloc()`
PyObject_REALLOC(p, n)	`PyObject_Realloc()`
PyObject_FREE(p)	`PyObject_Free()`
PyObject_DEL(p)	`PyObject_Free()`
PyObject_Del(p)	`PyObject_Free()`


reachable

Object Life Cycle

This section explains how a type's slots relate to each other throughout the life of an object. It is not intended to be a complete canonical reference for the slots; instead, refer to the slot-specific documentation in Type Object Structures for details about a particular slot.

Life Events

The figure below illustrates the order of events that can occur throughout an object's life. An arrow from *A* to *B* indicates that event *B* can occur after event *A* has occurred, with the arrow's label indicating the condition that must be true for *B* to occur after *A*.

Explanation:

- When a new object is constructed by calling its type:
    1. `tp_new` is called to create a new object.
    2. `tp_alloc` is directly called by `tp_new` to allocate the memory for the new object.
    3. `tp_init` initializes the newly created object. `tp_init` can be called again to re-initialize an object, if desired. The `tp_init` call can also be skipped entirely, for example by Python code calling `__new__()`.
- After `tp_init` completes, the object is ready to use.
- Some time after the last reference to an object is removed:
    1. If an object is not marked as *finalized*, it might be finalized by marking it as *finalized* and calling its `tp_finalize` function. Python does *not* finalize an object when the last reference to it is deleted; use `PyObject_CallFinalizerFromDealloc()` to ensure that `tp_finalize` is always called.
    2. If the object is marked as finalized, `tp_clear` might be called by the garbage collector to clear references held by the object. It is *not* called when the object's reference count reaches zero.
    3. `tp_dealloc` is called to destroy the object. To avoid

   code duplication, `tp_dealloc` typically calls into `tp_clear` to free up the object's references.
4. When `tp_dealloc` finishes object destruction, it directly calls `tp_free` (usually set to `PyObject_Free()` or `PyObject_GC_Del()` automatically as appropriate for the type) to deallocate the memory.

- The `tp_finalize` function is permitted to add a reference to the object if desired. If it does, the object is *resurrected*, preventing its pending destruction. (Only `tp_finalize` is allowed to resurrect an object; `tp_clear` and `tp_dealloc` cannot without calling into `tp_finalize`.) Resurrecting an object may or may not cause the object's *finalized* mark to be removed. Currently, Python does not remove the *finalized* mark from a resurrected object if it supports garbage collection (i.e., the `Py_TPFLAGS_HAVE_GC` flag is set) but does remove the mark if the object does not support garbage collection; either or both of these behaviors may change in the future.
- `tp_dealloc` can optionally call `tp_finalize` via `PyObject_CallFinalizerFromDealloc()` if it wishes to reuse that code to help with object destruction. This is recommended because it guarantees that `tp_finalize` is always called before destruction. See the `tp_dealloc` documentation for example code.
- If the object is a member of a cyclic isolate and either `tp_clear` fails to break the reference cycle or the cyclic isolate is not detected (perhaps `gc.disable()` was called, or the `Py_TPFLAGS_HAVE_GC` flag was erroneously omitted in one of the involved types), the objects remain indefinitely uncollectable (they «leak»). See `gc.garbage`.

If the object is marked as supporting garbage collection (the `Py_TPFLAGS_HAVE_GC` flag is set in `tp_flags`), the following events are also possible:

- The garbage collector occasionally calls `tp_traverse` to identify cyclic isolates.
- When the garbage collector discovers a cyclic isolate, it finalizes one of the objects in the group by marking it as

finalized and calling its `tp_finalize` function, if it has one. This repeats until the cyclic isolate doesn't exist or all of the objects have been finalized.

- `tp_finalize` is permitted to resurrect the object by adding a reference from outside the cyclic isolate. The new reference causes the group of objects to no longer form a cyclic isolate (the reference cycle may still exist, but if it does the objects are no longer isolated).
- When the garbage collector discovers a cyclic isolate and all of the objects in the group have already been marked as *finalized*, the garbage collector clears one or more of the uncleared objects in the group (possibly concurrently) by calling each's `tp_clear` function. This repeats as long as the cyclic isolate still exists and not all of the objects have been cleared.

Cyclic Isolate Destruction

Listed below are the stages of life of a hypothetical cyclic isolate that continues to exist after each member object is finalized or cleared. It is a memory leak if a cyclic isolate progresses through all of these stages; it should vanish once all objects are cleared, if not sooner. A cyclic isolate can vanish either because the reference cycle is broken or because the objects are no longer isolated due to finalizer resurrection (see `tp_finalize`).

1. **Reachable** (not yet a cyclic isolate): All objects are in their normal, reachable state. A reference cycle could exist, but an external reference means the objects are not yet isolated.
2. **Unreachable but consistent:** The final reference from outside the cyclic group of objects has been removed, causing the objects to become isolated (thus a cyclic isolate is born). None of the group's objects have been finalized or cleared yet. The cyclic isolate remains at this stage until some future run of the garbage collector (not necessarily the next run because the next run might not scan every object).
3. **Mix of finalized and not finalized:** Objects in a cyclic isolate are finalized one at a time, which means that there is a period of time when the cyclic isolate is composed of a mix of finalized and non-finalized objects. Finalization order is

unspecified, so it can appear random. A finalized object must behave in a sane manner when non-finalized objects interact with it, and a non-finalized object must be able to tolerate the finalization of an arbitrary subset of its referents.
4. **All finalized:** All objects in a cyclic isolate are finalized before any of them are cleared.
5. **Mix of finalized and cleared:** The objects can be cleared serially or concurrently (but with the GIL held); either way, some will finish before others. A finalized object must be able to tolerate the clearing of a subset of its referents. **PEP 442** [https://peps.python.org/pep-0442/] calls this stage «cyclic trash».
6. **Leaked:** If a cyclic isolate still exists after all objects in the group have been finalized and cleared, then the objects remain indefinitely uncollectable (see `gc.garbage`). It is a bug if a cyclic isolate reaches this stage—it means the `tp_clear` methods of the participating objects have failed to break the reference cycle as required.

If `tp_clear` did not exist, then Python would have no way to safely break a reference cycle. Simply destroying an object in a cyclic isolate would result in a dangling pointer, triggering undefined behavior when an object referencing the destroyed object is itself destroyed. The clearing step makes object destruction a two-phase process: first `tp_clear` is called to partially destroy the objects enough to detangle them from each other, then `tp_dealloc` is called to complete the destruction.

Unlike clearing, finalization is not a phase of destruction. A finalized object must still behave properly by continuing to fulfill its design contracts. An object’s finalizer is allowed to execute arbitrary Python code, and is even allowed to prevent the impending destruction by adding a reference. The finalizer is only related to destruction by call order—if it runs, it runs before destruction, which starts with `tp_clear` (if called) and concludes with `tp_dealloc`.

The finalization step is not necessary to safely reclaim the objects in a cyclic isolate, but its existence makes it easier to design types that behave in a sane manner when objects are cleared. Clearing an object might necessarily leave it in a broken, partially destroyed state—it might be unsafe to call any of the cleared object's methods or access any of its attributes. With finalization, only finalized objects can possibly interact with cleared objects; non-finalized objects are guaranteed to interact with only non-cleared (but potentially finalized) objects.

To summarize the possible interactions:

- A non-finalized object might have references to or from non-finalized and finalized objects, but not to or from cleared objects.
- A finalized object might have references to or from non-finalized, finalized, and cleared objects.
- A cleared object might have references to or from finalized and cleared objects, but not to or from non-finalized objects.

Without any reference cycles, an object can be simply destroyed once its last reference is deleted; the finalization and clearing steps are not necessary to safely reclaim unused objects. However, it can be useful to automatically call `tp_finalize` and `tp_clear` before destruction anyway because type design is simplified when all objects always experience the same series of events regardless of whether they participated in a cyclic isolate. Python currently only calls `tp_finalize` and `tp_clear` as needed to destroy a cyclic isolate; this may change in a future version.

Functions

To allocate and free memory, see Allocating objects on the heap.

void PyObject_CallFinalizer(PyObject *op)

> Finalizes the object as described in `tp_finalize`. Call this function (or `PyObject_CallFinalizerFromDealloc()`) instead of calling `tp_finalize` directly because this function may deduplicate multiple calls to `tp_finalize`. Currently, calls are only deduplicated if the type supports garbage collection (i.e., the `Py_TPFLAGS_HAVE_GC` flag is set); this may change in the future.
>
> *Added in version 3.4.*

int PyObject_CallFinalizerFromDealloc(PyObject *op)

Same as `PyObject_CallFinalizer()` but meant to be called at the beginning of the object's destructor (`tp_dealloc`). There must not be any references to the object. If the object's finalizer resurrects the object, this function returns -1; no further destruction should happen. Otherwise, this function returns 0 and destruction can continue normally.

Added in version 3.4.

Ver también

`tp_dealloc` for example code.

Estructuras de objetos comunes

Hay un gran número de estructuras que se utilizan en la definición de los tipos de objetos de Python. Esta sección describe estas estructuras y la forma en que se utilizan.

Tipos objeto base y macros

All Python objects ultimately share a small number of fields at the beginning of the object's representation in memory. These are represented by the `PyObject` and `PyVarObject` types, which are defined, in turn, by the expansions of some macros also used, whether directly or indirectly, in the definition of all other Python objects. Additional macros can be found under reference counting.

type PyObject

Part of the Limited API. (Only some members are part of the stable ABI.)
All object types are extensions of this type. This is a type which contains the information Python needs to treat a pointer to an object as an object. In a normal «release» build, it contains only the object's reference count and a pointer to the corresponding type object. Nothing is actually declared to be a `PyObject`, but every pointer to a Python object can be cast to a PyObject*.

The members must not be accessed directly; instead use macros such as `Py_REFCNT` and `Py_TYPE`.

Py_ssize_t ob_refcnt

Part of the Stable ABI.
The object's reference count, as returned by `Py_REFCNT`. Do not use this field directly; instead use functions and macros such as `Py_REFCNT`, `Py_INCREF()` and `Py_DecRef()`.

The field type may be different from `Py_ssize_t`, depending on build configuration and platform.

PyTypeObject *ob_type

Part of the Stable ABI.
The object's type. Do not use this field directly; use **`Py_TYPE`** and **`Py_SET_TYPE()`** instead.

type PyVarObject

Part of the Limited API. (Only some members are part of the stable ABI.)
An extension of **`PyObject`** that adds the **`ob_size`** field. This is intended for objects that have some notion of *length*.

As with **`PyObject`**, the members must not be accessed directly; instead use macros such as **`Py_SIZE`**, **`Py_REFCNT`** and **`Py_TYPE`**.

Py_ssize_t ob_size

Part of the Stable ABI.
A size field, whose contents should be considered an object's internal implementation detail.

Do not use this field directly; use **`Py_SIZE`** instead.

Object creation functions such as **`PyObject_NewVar()`** will generally set this field to the requested size (number of items). After creation, arbitrary values can be stored in **`ob_size`** using **`Py_SET_SIZE`**.

To get an object's publicly exposed length, as returned by the Python function **`len()`**, use **`PyObject_Length()`** instead.

PyObject_HEAD

Esta es una macro utilizado cuando se declara nuevos tipos que representan objetos sin una longitud variable. La macro PyObject_HEAD se expande a:

```
PyObject ob_base;
```

Consulte la documentación de **`PyObject`** en secciones anteriores.

PyObject_VAR_HEAD

Esta es una macro utilizado cuando se declara nuevos tipos que representan objetos con una longitud que varía de una instancia a otra instancia. La macro PyObject_VAR_HEAD se expande a:

```
PyVarObject ob_base;
```

Consulte la documentación de **`PyVarObject`** anteriormente.

PyTypeObject PyBaseObject_Type

Part of the Stable ABI.
The base class of all other objects, the same as **`object`** in Python.

int Py_Is(PyObject *x, PyObject *y)

Part of the Stable ABI since version 3.10.
Prueba si el objeto *x* es el objeto *y*, lo mismo que `x is y` en Python.

Added in version 3.10.

int Py_IsNone(PyObject *x)

Part of the Stable ABI since version 3.10.
Prueba si un objeto es la instancia única `None`, lo mismo que `x is None` en Python.

Added in version 3.10.

int Py_IsTrue(PyObject *x)

Part of the Stable ABI since version 3.10.
Prueba si un objeto es la instancia única `True`, lo mismo que `x is True` en Python.

Added in version 3.10.

int Py_IsFalse(PyObject *x)

Part of the Stable ABI since version 3.10.
Prueba si un objeto es la instancia única `False`, lo mismo que `x is False` en Python.

Added in version 3.10.

PyTypeObject *Py_TYPE(PyObject *o)

Return value: Borrowed reference. Part of the Stable ABI since version 3.14.
Obtiene el tipo de objeto Python *o*.

The returned reference is borrowed from *o*. Do not release it with **`Py_DECREF()`** or similar.

Distinto en la versión 3.11: **`Py_TYPE()`** se cambia a una función estática inline. El tipo de parámetro ya no es const PyObject*.

int Py_IS_TYPE(PyObject *o, PyTypeObject *type)

Retorna un valor distinto de cero si el objeto *o* tipo es *type*. Retorna cero en caso contrario. Equivalente a: `Py_TYPE(o) == type`.

Added in version 3.9.

void Py_SET_TYPE(PyObject *o, PyTypeObject *type)

Set the type of object *o* to *type*, without any checking or reference counting.

This is a very low-level operation. Consider instead setting the Python attribute **`__class__`** using **`PyObject_SetAttrString()`** or similar.

Note that assigning an incompatible type can lead to undefined behavior.

If *type* is a heap type, the caller must create a new reference to it. Similarly, if the old type of *o* is a heap type, the caller must release a reference to that type.

Added in version 3.9.

Py_ssize_t Py_SIZE(PyVarObject *o)

Get the **`ob_size`** field of *o*.

Distinto en la versión 3.11: **`Py_SIZE()`** se cambia a una función estática inline. El tipo de parámetro ya no es const PyVarObject*.

void Py_SET_SIZE(PyVarObject *o, Py_ssize_t size)

Set the **`ob_size`** field of *o* to *size*.

Added in version 3.9.

PyObject_HEAD_INIT(type)

Esta es una macro que se expande para valores de inicialización para un nuevo tipo **`PyObject`**. Esta macro expande:

```
_PyObject_EXTRA_INIT
1, type,
```

PyVarObject_HEAD_INIT(type, size)

This is a macro which expands to initialization values for a new **`PyVarObject`** type, including the **`ob_size`** field. This macro expands to:

```
_PyObject_EXTRA_INIT
1, type, size,
```

Implementando funciones y métodos

type PyCFunction

Part of the Stable ABI.
Type of the functions used to implement most Python callables in C. Functions of this type take two PyObject* parameters and return one such value. If the return value is `NULL`, an exception shall have been set. If not `NULL`, the return value is interpreted as the return value of the function as exposed in Python. The function must return a new reference.

La firma de la función es:

```
PyObject *PyCFunction(PyObject *self,
                      PyObject *args);
```

type PyCFunctionWithKeywords

Part of the Stable ABI.
Type of the functions used to implement Python callables in C with signature METH_VARARGS | METH_KEYWORDS. The function signature is:

```
PyObject *PyCFunctionWithKeywords(PyObject *self,
                                  PyObject *args,
                                  PyObject *kwargs)
```

type PyCFunctionFast

Part of the Stable ABI since version 3.13.
Type of the functions used to implement Python callables in C with signature **`METH_FASTCALL`**. The function signature is:

```
PyObject *PyCFunctionFast(PyObject *self,
                          PyObject *const *args,
                          Py_ssize_t nargs);
```

type PyCFunctionFastWithKeywords

Part of the Stable ABI since version 3.13.
Type of the functions used to implement Python callables in C with signature METH_FASTCALL | METH_KEYWORDS. The function signature is:

```
PyObject *PyCFunctionFastWithKeywords(PyObject *sel
                                      PyObject *con
                                      Py_ssize_t na
                                      PyObject *kwn
```

type PyCMethod

Type of the functions used to implement Python callables in C with signature METH_METHOD | METH_FASTCALL | METH_KEYWORDS. The function signature is:

```
PyObject *PyCMethod(PyObject *self,
                    PyTypeObject *defining_class,
                    PyObject *const *args,
                    Py_ssize_t nargs,
                    PyObject *kwnames)
```

Added in version 3.9.

type PyMethodDef

Part of the Stable ABI (including all members).
Estructura utiliza para describir un método de un tipo de extensión. Esta estructura tiene cuatro campos:

const char *ml_name

Name of the method.

PyCFunction ml_meth

Pointer to the C implementation.

int ml_flags

Flags bits indicating how the call should be constructed.

const char *ml_doc

Points to the contents of the docstring.

The **`ml_meth`** is a C function pointer. The functions may be of different types, but they always return PyObject*. If the function is not of the `PyCFunction`, the compiler will require a cast in the method table. Even though **`PyCFunction`** defines the first parameter as PyObject*, it is common that the method implementation uses the specific C type of the *self* object.

The `ml_flags` field is a bitfield which can include the following flags. The individual flags indicate either a calling convention or a binding convention.

Existen estas convenciones de llamada:

METH_VARARGS

> *Part of the Stable ABI.*
> This is the typical calling convention, where the methods have the type `PyCFunction`. The function expects two PyObject* values. The first one is the *self* object for methods; for module functions, it is the module object. The second parameter (often called *args*) is a tuple object representing all arguments. This parameter is typically processed using `PyArg_ParseTuple()` or `PyArg_UnpackTuple()`.

METH_KEYWORDS

> Can only be used in certain combinations with other flags: METH_VARARGS | METH_KEYWORDS, METH_FASTCALL | METH_KEYWORDS and METH_METHOD | METH_FASTCALL | METH_KEYWORDS.

METH_VARARGS | METH_KEYWORDS

> Los métodos con estas *flags* deben ser del tipo `PyCFunctionWithKeywords`. La función espera tres parámetros: *self*, *args*, *kwargs* donde *kwargs* es un diccionario de todos los argumentos de palabras clave o, posiblemente, `NULL` si no hay argumentos de palabra clave. Los parámetros se procesan típicamente usando `PyArg_ParseTupleAndKeywords()`.

METH_FASTCALL

> *Part of the Stable ABI since version 3.7.*
> Fast calling convention supporting only positional arguments.

The methods have the type **`PyCFunctionFast`**. The first parameter is *self*, the second parameter is a C array of PyObject* values indicating the arguments and the third parameter is the number of arguments (the length of the array).

Added in version 3.7.

Distinto en la versión 3.10: `METH_FASTCALL` is now part of the stable ABI.

METH_FASTCALL | METH_KEYWORDS

Extension of **`METH_FASTCALL`** supporting also keyword arguments, with methods of type **`PyCFunctionFastWithKeywords`**. Keyword arguments are passed the same way as in the vectorcall protocol: there is an additional fourth PyObject* parameter which is a tuple representing the names of the keyword arguments (which are guaranteed to be strings) or possibly `NULL` if there are no keywords. The values of the keyword arguments are stored in the *args* array, after the positional arguments.

Added in version 3.7.

METH_METHOD

Part of the Stable ABI since version 3.7.
Can only be used in the combination with other flags: METH_METHOD | METH_FASTCALL | METH_KEYWORDS.

METH_METHOD | METH_FASTCALL | METH_KEYWORDS

Extension of METH_FASTCALL | METH_KEYWORDS supporting the *defining class*, that is, the class that contains the method in question. The defining class might be a superclass of `Py_TYPE(self)`.

El método debe ser de tipo **`PyCMethod`**, lo mismo que para `METH_FASTCALL | METH_KEYWORDS` con el argumento `defining_clase` añadido después de `self`.

Added in version 3.9.

METH_NOARGS

Part of the Stable ABI.
Methods without parameters don't need to check whether arguments are given if they are listed with the **`METH_NOARGS`** flag. They need to be of type **`PyCFunction`**. The first parameter is typically named *self* and will hold a reference to the module or object instance. In all cases the second parameter will be `NULL`.

La función debe tener 2 parámetros. Dado que el segundo parámetro no se usa, **`Py_UNUSED`** se puede usar para evitar una advertencia del compilador.

METH_O

Part of the Stable ABI.
Methods with a single object argument can be listed with the **`METH_O`** flag, instead of invoking **`PyArg_ParseTuple()`** with a `"O"` argument. They have the type **`PyCFunction`**, with the *self* parameter, and a PyObject* parameter representing the single argument.

These two constants are not used to indicate the calling convention but the binding when used with methods of classes. These may not be used for functions defined for modules. At most one of these flags may be set for any given method.

METH_CLASS

Part of the Stable ABI.
Al método se le pasará el objeto tipo como primer parámetro, en lugar de una instancia del tipo. Esto se utiliza para crear métodos de clase (*class methods*), similar a lo que se crea cuando se utiliza la función **`classmethod()`** incorporada.

METH_STATIC

Part of the Stable ABI.
El método pasará `NULL` como el primer parámetro en lugar de una instancia del tipo. Esto se utiliza para crear métodos estáticos (*static methods*), similar a lo que se crea cuando se utiliza la función **`staticmethod()`** incorporada.

En otros controles constantes dependiendo si se carga un método en su lugar (*in place*) de otra definición con el mismo nombre del método.

METH_COEXIST

Part of the Stable ABI.
The method will be loaded in place of existing definitions. Without *METH_COEXIST*, the default is to skip repeated definitions. Since slot wrappers are loaded before the method table, the existence of a *sq_contains* slot, for example, would generate a wrapped method named **`__contains__()`** and preclude the loading of a corresponding PyCFunction with the same name. With the flag defined, the PyCFunction will be loaded in place of the wrapper object and will co-exist with the slot. This is helpful because calls to PyCFunctions are optimized more than wrapper object calls.

PyTypeObject PyCMethod_Type

The type object corresponding to Python C method objects. This is available as **`types.BuiltinMethodType`** in the Python layer.

int PyCMethod_Check(PyObject *op)

Return true if *op* is an instance of the **`PyCMethod_Type`** type or a subtype of it. This function always succeeds.

int PyCMethod_CheckExact(PyObject *op)

This is the same as **`PyCMethod_Check()`**, but does not account for subtypes.

PyObject *PyCMethod_New(PyMethodDef *ml, PyObject *self, PyObject *module, PyTypeObject *cls)

Return value: New reference. Part of the Stable ABI since version 3.9.
Turn *ml* into a Python callable object. The caller must ensure that *ml* outlives the callable. Typically, *ml* is defined as a static variable.

The *self* parameter will be passed as the *self* argument to the C function in `ml->ml_meth` when invoked. *self* can be `NULL`.

The callable object's `__module__` attribute can be set from the given *module* argument. *module* should be a Python string, which will be used as name of the module the function is defined in. If unavailable, it can be set to **`None`** or `NULL`.

Ver también

`function.__module__`

The *cls* parameter will be passed as the *defining_class* argument to the C function. Must be set if **`METH_METHOD`** is set on `ml->ml_flags`.

Added in version 3.9.

PyTypeObject PyCFunction_Type

Part of the Stable ABI.
The type object corresponding to Python C function objects. This is available as **`types.BuiltinFunctionType`** in the Python layer.

int PyCFunction_Check(PyObject *op)

Return true if *op* is an instance of the **`PyCFunction_Type`** type or a subtype of it. This function always succeeds.

int PyCFunction_CheckExact(PyObject *op)

This is the same as **`PyCFunction_Check()`**, but does not account for subtypes.

PyObject *PyCFunction_NewEx(PyMethodDef *ml, PyObject *self, PyObject *module)

Return value: New reference. Part of the Stable ABI.
Equivalent to `PyCMethod_New(ml, self, module, NULL)`.

PyObject *PyCFunction_New(PyMethodDef *ml, PyObject *self)

Return value: New reference. Part of the Stable ABI since version 3.4.

Equivalent to `PyCMethod_New(ml, self, NULL, NULL)`.

int PyCFunction_GetFlags(PyObject *func)

Part of the Stable ABI.

Get the function's flags on *func* as they were passed to **`ml_flags`**.

If *func* is not a C function object, this fails with an exception. *func* must not be `NULL`.

This function returns the function's flags on success, and `-1` with an exception set on failure.

int PyCFunction_GET_FLAGS(PyObject *func)

This is the same as **`PyCFunction_GetFlags()`**, but without error or type checking.

PyCFunction PyCFunction_GetFunction(PyObject *func)

Part of the Stable ABI.

Get the function pointer on *func* as it was passed to **`ml_meth`**.

If *func* is not a C function object, this fails with an exception. *func* must not be `NULL`.

This function returns the function pointer on success, and `NULL` with an exception set on failure.

int PyCFunction_GET_FUNCTION(PyObject *func)

This is the same as **`PyCFunction_GetFunction()`**, but without error or type checking.

PyObject *PyCFunction_GetSelf(PyObject *func)

Part of the Stable ABI.
Get the «self» object on *func*. This is the object that would be passed to the first argument of a **`PyCFunction`**. For C function objects created through a **`PyMethodDef`** on a **`PyModuleDef`**, this is the resulting module object.

If *func* is not a C function object, this fails with an exception. *func* must not be `NULL`.

This function returns a borrowed reference to the «self» object on success, and `NULL` with an exception set on failure.

PyObject *PyCFunction_GET_SELF(PyObject *func)

This is the same as **`PyCFunction_GetSelf()`**, but without error or type checking.

Acceder a atributos de tipos de extensión

type PyMemberDef

Part of the Stable ABI (including all members).
Structure which describes an attribute of a type which corresponds to a C struct member. When defining a class, put a NULL-terminated array of these structures in the **`tp_members`** slot.

Its fields are, in order:

const char *name

Name of the member. A NULL value marks the end of a `PyMemberDef[]` array.

The string should be static, no copy is made of it.

int type

The type of the member in the C struct. See Member types for the possible values.

Py_ssize_t offset

The offset in bytes that the member is located on the type's object struct.

int flags

Zero or more of the Member flags, combined using bitwise OR.

const char *doc

The docstring, or NULL. The string should be static, no copy is made of it. Typically, it is defined using `PyDoc_STR`.

By default (when `flags` is `0`), members allow both read and write access. Use the `Py_READONLY` flag for read-only access. Certain types, like `Py_T_STRING`, imply **`Py_READONLY`**. Only `Py_T_OBJECT_EX` (and legacy `T_OBJECT`) members can be deleted.

For heap-allocated types (created using `PyType_FromSpec()` or similar), `PyMemberDef` may contain a definition for the special member `"__vectorcalloffset__"`, corresponding to `tp_vectorcall_offset` in type objects. This member must be defined with `Py_T_PYSSIZET`, and either `Py_READONLY` or `Py_READONLY | Py_RELATIVE_OFFSET`. For example:

```
static PyMemberDef spam_type_members[] = {
    {"__vectorcalloffset__", Py_T_PYSSIZET,
     offsetof(Spam_object, vectorcall), Py_READONLY
    {NULL}  /* Sentinel */
};
```

(You may need to `#include <stddef.h>` for **`offsetof()`**.)

The legacy offsets `tp_dictoffset` and `tp_weaklistoffset` can be defined similarly using `"__dictoffset__"` and `"__weaklistoffset__"` members, but extensions are strongly encouraged to use `Py_TPFLAGS_MANAGED_DICT` and

`Py_TPFLAGS_MANAGED_WEAKREF` instead.

Distinto en la versión 3.12: `PyMemberDef` is always available. Previously, it required including `"structmember.h"`.

Distinto en la versión 3.14: **`Py_RELATIVE_OFFSET`** is now allowed for `"__vectorcalloffset__"`, `"__dictoffset__"` and `"__weaklistoffset__"`.

PyObject *PyMember_GetOne(const char *obj_addr, struct PyMemberDef *m)

Part of the Stable ABI.
Obtiene un atributo que pertenece al objeto en la dirección *obj_addr*. El atributo se describe por `PyMemberDef` *m*. Retorna `NULL` en caso de error.

Distinto en la versión 3.12: `PyMember_GetOne` is always available. Previously, it required including `"structmember.h"`.

int PyMember_SetOne(char *obj_addr, struct PyMemberDef *m, PyObject *o)

Part of the Stable ABI.
Establece un atributo que pertenece al objeto en la dirección *obj_addr* al objeto *o*. El atributo a establecer se describe por `PyMemberDef` *m*. Retorna `0` si tiene éxito y un valor negativo si falla.

Distinto en la versión 3.12: `PyMember_SetOne` is always available. Previously, it required including `"structmember.h"`.

Member flags

The following flags can be used with **`PyMemberDef.flags`**:

Py_READONLY

Part of the Stable ABI since version 3.12.
Not writable.

Py_AUDIT_READ

Part of the Stable ABI since version 3.12.
Emit an `object.__getattr__` audit event before reading.

Py_RELATIVE_OFFSET

Part of the Stable ABI since version 3.12.
Indicates that the **`offset`** of this `PyMemberDef` entry indicates an offset from the subclass-specific data, rather than from `PyObject`.

Can only be used as part of the **`Py_tp_members slot`** when creating a class using negative **`basicsize`**. It is mandatory in that case. When setting **`tp_members`** from the slot during class creation, Python clears the flag and sets **`PyMemberDef.offset`** to the offset from the `PyObject` struct.

Distinto en la versión 3.10: The **`RESTRICTED`**, **`READ_RESTRICTED`** and **`WRITE_RESTRICTED`** macros available with `#include "structmember.h"` are deprecated. **`READ_RESTRICTED`** and **`RESTRICTED`** are equivalent to **`Py_AUDIT_READ`**; **`WRITE_RESTRICTED`** does nothing.

Distinto en la versión 3.12: The **`READONLY`** macro was renamed to **`Py_READONLY`**. The **`PY_AUDIT_READ`** macro was renamed with the `Py_` prefix. The new names are now always available. Previously, these required `#include "structmember.h"`. The header is still available and it provides the old names.

Member types

`PyMemberDef.type` can be one of the following macros corresponding to various C types. When the member is accessed in Python, it will be converted to the equivalent Python type. When it is set from Python, it will be converted back to the C type. If that is not possible, an exception such as **`TypeError`** or **`ValueError`** is raised.

Unless marked (D), attributes defined this way cannot be deleted using e.g. **`del`** or **`delattr()`**.

Macro name	C type	Python type
Py_T_BYTE *Part of the Stable ABI since version 3.12.*	char	int
Py_T_SHORT *Part of the Stable ABI since version 3.12.*	short	int
Py_T_INT *Part of the Stable ABI since version 3.12.*	int	int
Py_T_LONG *Part of the Stable ABI since version 3.12.*	long	int
Py_T_LONGLONG *Part of the Stable ABI since version 3.12.*	long long	int
Py_T_UBYTE *Part of the Stable ABI since version 3.12.*	unsigned char	int
Py_T_UINT *Part of the Stable ABI since version 3.12.*	unsigned int	int
Py_T_USHORT *Part of the Stable ABI since version 3.12.*	unsigned short	int
Py_T_ULONG *Part of the Stable ABI since version 3.12.*	unsigned long	int
Py_T_ULONGLONG *Part of the Stable ABI since version 3.12.*	unsigned long long	int
Py_T_PYSSIZET *Part of the Stable ABI since version 3.12.*	Py_ssize_t	int
Py_T_FLOAT *Part of the Stable ABI since version 3.12.*	float	float
Py_T_DOUBLE *Part of the Stable ABI since version 3.12.*	double	float
Py_T_BOOL *Part of the Stable ABI since version 3.12.*	char (written as 0 or 1)	bool
Py_T_STRING *Part of the Stable ABI since version 3.12.*	const char* (*)	str (RO)
Py_T_STRING_INPLACE *Part of the Stable ABI since version 3.12.*	const char[] (*)	str (RO)
Py_T_CHAR *Part of the Stable ABI since version 3.12.*	char (0-127)	str (**)
Py_T_OBJECT_EX	PyObject*	object (D)

Part of the Stable ABI since version 3.12.

(*): Zero-terminated, UTF8-encoded C string. With **`Py_T_STRING`** the C representation is a pointer; with **`Py_T_STRING_INPLACE`** the string is stored directly in the structure.

(**): String of length 1. Only ASCII is accepted.

(RO): Implies **`Py_READONLY`**.

(D): Can be deleted, in which case the pointer is set to `NULL`. Reading a `NULL` pointer raises **`AttributeError`**.

Added in version 3.12: In previous versions, the macros were only available with `#include "structmember.h"` and were named without the `Py_` prefix (e.g. as `T_INT`). The header is still available and contains the old names, along with the following deprecated types:

T_OBJECT

Like `Py_T_OBJECT_EX`, but `NULL` is converted to `None`. This results in surprising behavior in Python: deleting the attribute effectively sets it to `None`.

T_NONE

Always `None`. Must be used with **`Py_READONLY`**.

Defining Getters and Setters

type PyGetSetDef

Part of the Stable ABI (including all members).
Estructura para definir el acceso para un tipo como el de una propiedad. Véase también la descripción de la ranura **`PyTypeObject.tp_getset`**.

const char *name

nombre del atributo

getter get

C function to get the attribute.

setter set

Optional C function to set or delete the attribute. If `NULL`, the attribute is read-only.

const char *doc

docstring opcional

void *closure

Optional user data pointer, providing additional data for getter and setter.

typedef PyObject *(*getter)(PyObject*, void*)

Part of the Stable ABI.
The `get` function takes one PyObject* parameter (the instance) and a user data pointer (the associated `closure`):

Debe retornar una nueva referencia en caso de éxito o `NULL` con una excepción establecida en caso de error.

typedef int (*setter)(PyObject*, PyObject*, void*)

Part of the Stable ABI.
`set` functions take two PyObject* parameters (the instance and the value to be set) and a user data pointer (the associated `closure`):

En caso de que el atributo deba suprimirse el segundo parámetro es `NULL`. Debe retornar `0` en caso de éxito o `-1` con una excepción explícita en caso de fallo.

[1] [2]

Type Object Structures

Perhaps one of the most important structures of the Python object system is the structure that defines a new type: the **`PyTypeObject`** structure. Type objects can be handled using any of the `PyObject_*` or `PyType_*` functions, but do not offer much that's interesting to most Python applications. These objects are fundamental to how objects behave, so they are very important to the interpreter itself and to any extension module that implements new types.

Los objetos de tipo son bastante grandes en comparación con la mayoría de los tipos estándar. La razón del tamaño es que cada objeto de tipo almacena una gran cantidad de valores, principalmente punteros de función C, cada uno de los cuales implementa una pequeña parte de la funcionalidad del tipo. Los campos del objeto tipo se examinan en detalle en esta sección. Los campos se describirán en el orden en que aparecen en la estructura.

Además de la siguiente referencia rápida, la sección Ejemplos proporciona una visión rápida del significado y uso de **`PyTypeObject`**.

Referencia rápida

«ranuras *tp*» (*tp slots*)

Ranura PyTypeObject [1]	Tipo	métodos/atributos especiales	Información [2]
			O T D I
`tp_name`	const char *	__name__	X X
`tp_basicsize`	Py_ssize_t		X X X
`tp_itemsize`	Py_ssize_t		X X
`tp_dealloc`	destructor		X X X
`tp_vectorcall_offset`	Py_ssize_t		X
(`tp_getattr`)	getattrfunc	__getattribute__, __getattr__	G
(`tp_setattr`)	setattrfunc	__setattr__, __delattr__	G

%	tp_as_async	PyAsyncMethods *	(sub-slots)
X	tp_repr	reprfunc	__repr__
%	tp_as_number	PyNumberMethods *	(sub-slots)
%	tp_as_sequence	PySequenceMethods *	(sub-slots)
%	tp_as_mapping	PyMappingMethods *	(sub-slots)
G	tp_hash	hashfunc	__hash__
X	tp_call	ternaryfunc	__call__
X	tp_str	reprfunc	__str__
G	tp_getattro	getattrofunc	__getattribute__, __getattr__
G	tp_setattro	setattrofunc	__setattr__, __delattr__
%	tp_as_buffer	PyBufferProcs *	(sub-slots)
X	tp_flags	unsigned long	
X	tp_doc	const char *	__doc__
G	tp_traverse	traverseproc	
G	tp_clear	inquiry	
G	tp_richcompare	richcmpfunc	__lt__, __le__, __eq__, __ne__, __gt__, __ge__
X	(tp_weaklistoffset)	Py_ssize_t	
X	tp_iter	getiterfunc	__iter__
X	tp_iternext	iternextfunc	__next__
X	tp_methods	PyMethodDef []	
X	tp_members	PyMemberDef []	
X	tp_getset	PyGetSetDef []	
X	tp_base	PyTypeObject *	__base__
?	tp_dict	PyObject *	__dict__
X	tp_descr_get	descrgetfunc	__get__
X	tp_descr_set	descrsetfunc	__set__, __delete__
X	(tp_dictoffset)	Py_ssize_t	
X	tp_init	initproc	__init__
X	tp_alloc	allocfunc	
X	tp_new	newfunc	__new__
X	tp_free	freefunc	
X	tp_is_gc	inquiry	
	<tp_bases>	PyObject *	__bases__
	<tp_mro>	PyObject *	__mro__
	[tp_cache]	PyObject *	
	[tp_subclasses]	void *	__subclasses__
	[tp_weaklist]	PyObject *	
	(tp_del)	destructor	
	[tp_version_tag]	unsigned int	
X	tp_finalize	destructor	__del__

tp_vectorcall	**vectorcallfunc**
[**tp_watched**]	unsigned char

(): A slot name in parentheses indicates it is (effectively) deprecated.

< >: Names in angle brackets should be initially set to `NULL` and treated as read-only.

[] : Names in square brackets are for internal use only.

<R> (as a prefix) means the field is required (must be non-`NULL`).

Columnas:

«O»: set on **PyBaseObject_Type**

«T»: set on **PyType_Type**

«D»: por defecto (si la ranura está establecida como `NULL`)

```
X - PyType_Ready sets this value if it is NULL
~ - PyType_Ready always sets this value (it should be NU
? - PyType_Ready may set this value depending on other s

Also see the inheritance column ("I").
```

«I»: herencia

```
X - type slot is inherited via *PyType_Ready* if define
% - the slots of the sub-struct are inherited individual
G - inherited, but only in combination with other slots;
? - it's complicated; see the slot's description
```

Tenga en cuenta que algunos espacios se heredan efectivamente a través de la cadena de búsqueda de atributos normal.

sub-ranuras (*sub-slots*)

Ranuras (*Slot*)	**Type**	métodos especiales
am_await	**unaryfunc**	__await__

`am_aiter`	`unaryfunc`	__aiter__
`am_anext`	`unaryfunc`	__anext__
`am_send`	`sendfunc`	
`nb_add`	`binaryfunc`	__add__ __radd__
`nb_inplace_add`	`binaryfunc`	__iadd__
`nb_subtract`	`binaryfunc`	__sub__ __rsub__
`nb_inplace_subtract`	`binaryfunc`	__isub__
`nb_multiply`	`binaryfunc`	__mul__ __rmul__
`nb_inplace_multiply`	`binaryfunc`	__imul__
`nb_remainder`	`binaryfunc`	__mod__ __rmod__
`nb_inplace_remainder`	`binaryfunc`	__imod__
`nb_divmod`	`binaryfunc`	__divmod__ __rdivmod__
`nb_power`	`ternaryfunc`	__pow__ __rpow__
`nb_inplace_power`	`ternaryfunc`	__ipow__
`nb_negative`	`unaryfunc`	__neg__
`nb_positive`	`unaryfunc`	__pos__
`nb_absolute`	`unaryfunc`	__abs__
`nb_bool`	`inquiry`	__bool__
`nb_invert`	`unaryfunc`	__invert__
`nb_lshift`	`binaryfunc`	__lshift__ __rlshift__
`nb_inplace_lshift`	`binaryfunc`	__ilshift__
`nb_rshift`	`binaryfunc`	__rshift__ __rrshift__
`nb_inplace_rshift`	`binaryfunc`	__irshift__
`nb_and`	`binaryfunc`	__and__ __rand__
`nb_inplace_and`	`binaryfunc`	__iand__
`nb_xor`	`binaryfunc`	__xor__ __rxor__
`nb_inplace_xor`	`binaryfunc`	__ixor__
`nb_or`	`binaryfunc`	__or__ __ror__
`nb_inplace_or`	`binaryfunc`	__ior__
`nb_int`	`unaryfunc`	__int__
`nb_reserved`	void*	
`nb_float`	`unaryfunc`	__float__
`nb_floor_divide`	`binaryfunc`	__floordiv__
`nb_inplace_floor_divide`	`binaryfunc`	__ifloordiv__
`nb_true_divide`	`binaryfunc`	__truediv__
`nb_inplace_true_divide`	`binaryfunc`	__itruediv__
`nb_index`	`unaryfunc`	__index__
`nb_matrix_multiply`	`binaryfunc`	__matmul__ __rmatmul__
`nb_inplace_matrix_multiply`	`binaryfunc`	__imatmul__
`mp_length`	`lenfunc`	__len__

`mp_subscript`	`binaryfunc`	__getitem__
`mp_ass_subscript`	`objobjargproc`	__setitem__, __delitem__
`sq_length`	`lenfunc`	__len__
`sq_concat`	`binaryfunc`	__add__
`sq_repeat`	`ssizeargfunc`	__mul__
`sq_item`	`ssizeargfunc`	__getitem__
`sq_ass_item`	`ssizeobjargproc`	__setitem__ __delitem__
`sq_contains`	`objobjproc`	__contains__
`sq_inplace_concat`	`binaryfunc`	__iadd__
`sq_inplace_repeat`	`ssizeargfunc`	__imul__
`bf_getbuffer`	`getbufferproc()`	__buffer__
`bf_releasebuffer`	`releasebufferproc()`	__release_buffer__

ranura de *typedefs*

typedef	Tipos de parámetros	Tipo de retorno
`allocfunc`	`PyTypeObject` * `Py_ssize_t`	`PyObject` *
`destructor`	`PyObject` *	void
`freefunc`	void *	void
`traverseproc`	`PyObject` * `visitproc` void *	int
`newfunc`	`PyTypeObject` * `PyObject` * `PyObject` *	`PyObject` *
`initproc`	`PyObject` * `PyObject` * `PyObject` *	int
`reprfunc`	`PyObject` *	`PyObject` *
`getattrfunc`	`PyObject` * const char *	`PyObject` *
`setattrfunc`	`PyObject` * const char * `PyObject` *	int
`getattrofunc`	`PyObject` * `PyObject` *	`PyObject` *
`setattrofunc`	`PyObject` * `PyObject` * `PyObject` *	int

`descrgetfunc`	`PyObject` * `PyObject` * `PyObject` *	`PyObject` *
`descrsetfunc`	`PyObject` * `PyObject` * `PyObject` *	int
`hashfunc`	`PyObject` *	Py_hash_t
`richcmpfunc`	`PyObject` * `PyObject` * int	`PyObject` *
`getiterfunc`	`PyObject` *	`PyObject` *
`iternextfunc`	`PyObject` *	`PyObject` *
`lenfunc`	`PyObject` *	`Py_ssize_t`
`getbufferproc`	`PyObject` * `Py_buffer` * int	int
`releasebufferproc`	`PyObject` * `Py_buffer` *	void
`inquiry`	`PyObject` *	int
`unaryfunc`	`PyObject` *	`PyObject` *
`binaryfunc`	`PyObject` * `PyObject` *	`PyObject` *
`ternaryfunc`	`PyObject` * `PyObject` * `PyObject` *	`PyObject` *
`ssizeargfunc`	`PyObject` * `Py_ssize_t`	`PyObject` *
`ssizeobjargproc`	`PyObject` * `Py_ssize_t` `PyObject` *	int
`objobjproc`	`PyObject` * `PyObject` *	int
`objobjargproc`	`PyObject` * `PyObject` * `PyObject` *	int

Vea Tipo Ranura typedefs abajo para más detalles.

Definición de `PyTypeObject`

The structure definition for **`PyTypeObject`** can be found in `Include/cpython/object.h`. For convenience of reference, this repeats the definition found there:

```
typedef struct _typeobject {
    PyObject_VAR_HEAD
    const char *tp_name; /* For printing, in format "<mo
    Py_ssize_t tp_basicsize, tp_itemsize; /* For allocat

    /* Methods to implement standard operations */

    destructor tp_dealloc;
    Py_ssize_t tp_vectorcall_offset;
    getattrfunc tp_getattr;
    setattrfunc tp_setattr;
    PyAsyncMethods *tp_as_async; /* formerly known as tp
                                    or tp_reserved (Pyth
    reprfunc tp_repr;

    /* Method suites for standard classes */

    PyNumberMethods *tp_as_number;
    PySequenceMethods *tp_as_sequence;
    PyMappingMethods *tp_as_mapping;

    /* More standard operations (here for binary compati

    hashfunc tp_hash;
    ternaryfunc tp_call;
    reprfunc tp_str;
    getattrofunc tp_getattro;
    setattrofunc tp_setattro;

    /* Functions to access object as input/output buffer
    PyBufferProcs *tp_as_buffer;

    /* Flags to define presence of optional/expanded fea
    unsigned long tp_flags;
```

```
const char *tp_doc; /* Documentation string */

/* Assigned meaning in release 2.0 */
/* call function for all accessible objects */
traverseproc tp_traverse;

/* delete references to contained objects */
inquiry tp_clear;

/* Assigned meaning in release 2.1 */
/* rich comparisons */
richcmpfunc tp_richcompare;

/* weak reference enabler */
Py_ssize_t tp_weaklistoffset;

/* Iterators */
getiterfunc tp_iter;
iternextfunc tp_iternext;

/* Attribute descriptor and subclassing stuff */
PyMethodDef *tp_methods;
PyMemberDef *tp_members;
PyGetSetDef *tp_getset;
// Strong reference on a heap type, borrowed referen
PyTypeObject *tp_base;
PyObject *tp_dict;
descrgetfunc tp_descr_get;
descrsetfunc tp_descr_set;
Py_ssize_t tp_dictoffset;
initproc tp_init;
allocfunc tp_alloc;
newfunc tp_new;
freefunc tp_free; /* Low-level free-memory routine *
inquiry tp_is_gc; /* For PyObject_IS_GC */
PyObject *tp_bases;
PyObject *tp_mro; /* method resolution order */
PyObject *tp_cache; /* no longer used */
```

```
    void *tp_subclasses;  /* for static builtin types th
    PyObject *tp_weaklist; /* not used for static builti
    destructor tp_del;

    /* Type attribute cache version tag. Added in versio
     * If zero, the cache is invalid and must be initial
     */
    unsigned int tp_version_tag;

    destructor tp_finalize;
    vectorcallfunc tp_vectorcall;

    /* bitset of which type-watchers care about this typ
    unsigned char tp_watched;

    /* Number of tp_version_tag values used.
     * Set to _Py_ATTR_CACHE_UNUSED if the attribute cac
     * disabled for this type (e.g. due to custom MRO en
     * Otherwise, limited to MAX_VERSIONS_PER_CLASS (def
     */
    uint16_t tp_versions_used;
} PyTypeObject;
```

Ranuras (*Slots*) `PyObject`

The type object structure extends the **`PyVarObject`** structure. The **`ob_size`** field is used for dynamic types (created by **`type_new()`**, usually called from a class statement). Note that **`PyType_Type`** (the metatype) initializes **`tp_itemsize`**, which means that its instances (i.e. type objects) *must* have the **`ob_size`** field.

`PyObject.ob_refcnt`

The type object's reference count is initialized to `1` by the `PyObject_HEAD_INIT` macro. Note that for statically allocated type objects, the type's instances (objects whose **`ob_type`** points back to the type) do *not* count as references. But for dynamically allocated type objects, the instances *do*

count as references.

Herencia:

Este campo no es heredado por los subtipos.

PyObject.ob_type

Este es el tipo del tipo, en otras palabras, su metatipo. Se inicializa mediante el argumento de la macro `PyObject_HEAD_INIT`, y su valor normalmente debería ser `&PyType_Type`. Sin embargo, para los módulos de extensión cargables dinámicamente que deben ser utilizables en Windows (al menos), el compilador se queja de que este no es un inicializador válido. Por lo tanto, la convención es pasar `NULL` al macro `PyObject_HEAD_INIT` e inicializar este campo explícitamente al comienzo de la función de inicialización del módulo, antes de hacer cualquier otra cosa. Esto normalmente se hace así:

```
Foo_Type.ob_type = &PyType_Type;
```

This should be done before any instances of the type are created. **`PyType_Ready()`** checks if **`ob_type`** is `NULL`, and if so, initializes it to the **`ob_type`** field of the base class. **`PyType_Ready()`** will not change this field if it is non-zero.

Herencia:

Este campo es heredado por subtipos.

Ranuras `PyVarObject`

PyVarObject.ob_size

Para objetos de tipo estáticamente asignados, debe inicializarse a cero. Para objetos de tipo dinámicamente asignados, este campo tiene un significado interno especial.

This field should be accessed using the `Py_SIZE()` macro.

Herencia:

Este campo no es heredado por los subtipos.

Ranuras `PyTypeObject`

Each slot has a section describing inheritance. If `PyType_Ready()` may set a value when the field is set to `NULL` then there will also be a «Default» section. (Note that many fields set on `PyBaseObject_Type` and `PyType_Type` effectively act as defaults.)

const char *PyTypeObject.tp_name

Pointer to a NUL-terminated string containing the name of the type. For types that are accessible as module globals, the string should be the full module name, followed by a dot, followed by the type name; for built-in types, it should be just the type name. If the module is a submodule of a package, the full package name is part of the full module name. For example, a type named `T` defined in module `M` in subpackage `Q` in package `P` should have the `tp_name` initializer `"P.Q.M.T"`.

Para objetos de tipo dinámicamente asignados, éste debe ser sólo el nombre del tipo, y el nombre del módulo almacenado explícitamente en el dict tipo que el valor de `'__module__'` clave.

For statically allocated type objects, the *tp_name* field should contain a dot. Everything before the last dot is made accessible as the `__module__` attribute, and everything after the last dot is made accessible as the `__name__` attribute.

If no dot is present, the entire `tp_name` field is made accessible as the `__name__` attribute, and the `__module__` attribute is undefined (unless explicitly set in the dictionary, as explained above). This means your type will be impossible to pickle. Additionally, it will not be listed in module documentations created with pydoc.

Este campo no debe ser `NULL`. Es el único campo obligatorio en `PyTypeObject()` (que no sea potencialmente `tp_itemsize`).

Herencia:

Este campo no es heredado por los subtipos.

Py_ssize_t PyTypeObject.tp_basicsize
Py_ssize_t PyTypeObject.tp_itemsize

Estos campos permiten calcular el tamaño en bytes de instancias del tipo.

There are two kinds of types: types with fixed-length instances have a zero `tp_itemsize` field, types with variable-length instances have a non-zero `tp_itemsize` field. For a type with fixed-length instances, all instances have the same size, given in `tp_basicsize`. (Exceptions to this rule can be made using `PyUnstable_Object_GC_NewWithExtraData()`.)

For a type with variable-length instances, the instances must have an `ob_size` field, and the instance size is `tp_basicsize` plus N times `tp_itemsize`, where N is the «length» of the object.

Functions like `PyObject_NewVar()` will take the value of N as an argument, and store in the instance's `ob_size` field. Note that the `ob_size` field may later be used for other purposes. For example, `int` instances use the bits of `ob_size` in an implementation-defined way; the underlying storage and its size should be accessed using `PyLong_Export()`.

Nota

The `ob_size` field should be accessed using the `Py_SIZE()` and `Py_SET_SIZE()` macros.

Also, the presence of an `ob_size` field in the instance layout doesn't mean that the instance structure is variable-length. For example, the `list` type has fixed-length instances, yet those instances have a `ob_size` field. (As with `int`, avoid reading lists" `ob_size` directly. Call `PyList_Size()` instead.)

The `tp_basicsize` includes size needed for data of the type's `tp_base`, plus any extra data needed by each instance.

The correct way to set `tp_basicsize` is to use the `sizeof` operator on the struct used to declare the instance layout. This struct must include the struct used to declare the base type. In other words, `tp_basicsize` must be greater than or equal to the base's `tp_basicsize`.

Since every type is a subtype of `object`, this struct must include `PyObject` or `PyVarObject` (depending on whether `ob_size` should be included). These are usually defined by the macro `PyObject_HEAD` or `PyObject_VAR_HEAD`, respectively.

The basic size does not include the GC header size, as that header is not part of `PyObject_HEAD`.

For cases where struct used to declare the base type is unknown, see `PyType_Spec.basicsize` and `PyType_FromMetaclass()`.

Notes about alignment:

- `tp_basicsize` must be a multiple of `_Alignof(PyObject)`. When using `sizeof` on a `struct` that includes `PyObject_HEAD`, as recommended, the compiler ensures this. When not

using a C `struct`, or when using compiler extensions like `__attribute__((packed))`, it is up to you.
- If the variable items require a particular alignment, **`tp_basicsize`** and **`tp_itemsize`** must each be a multiple of that alignment. For example, if a type's variable part stores a `double`, it is your responsibility that both fields are a multiple of `_Alignof(double)`.

Herencia:

These fields are inherited separately by subtypes. (That is, if the field is set to zero, **`PyType_Ready()`** will copy the value from the base type, indicating that the instances do not need additional storage.)

If the base type has a non-zero **`tp_itemsize`**, it is generally not safe to set **`tp_itemsize`** to a different non-zero value in a subtype (though this depends on the implementation of the base type).

destructor PyTypeObject.tp_dealloc

The corresponding slot ID `Py_tp_dealloc` *is part of the Stable ABI.*

A pointer to the instance destructor function. The function signature is:

```
void tp_dealloc(PyObject *self);
```

The destructor function should remove all references which the instance owns (e.g., call **`Py_CLEAR()`**), free all memory buffers owned by the instance, and call the type's **`tp_free`** function to free the object itself.

If you may call functions that may set the error indicator, you must use **`PyErr_GetRaisedException()`** and **`PyErr_SetRaisedException()`** to ensure you don't clobber a preexisting error indicator (the deallocation could have occurred while processing a different error):

```
static void
```

```
foo_dealloc(foo_object *self)
{
    PyObject *et, *ev, *etb;
    PyObject *exc = PyErr_GetRaisedException();
    ...
    PyErr_SetRaisedException(exc);
}
```

The dealloc handler itself must not raise an exception; if it hits an error case it should call `PyErr_FormatUnraisable()` to log (and clear) an unraisable exception.

No guarantees are made about when an object is destroyed, except:

- Python will destroy an object immediately or some time after the final reference to the object is deleted, unless its finalizer (`tp_finalize`) subsequently resurrects the object.
- An object will not be destroyed while it is being automatically finalized (`tp_finalize`) or automatically cleared (`tp_clear`).

CPython currently destroys an object immediately from `Py_DECREF()` when the new reference count is zero, but this may change in a future version.

It is recommended to call `PyObject_CallFinalizerFromDealloc()` at the beginning of **`tp_dealloc`** to guarantee that the object is always finalized before destruction.

If the type supports garbage collection (the `Py_TPFLAGS_HAVE_GC` flag is set), the destructor should call `PyObject_GC_UnTrack()` before clearing any member fields.

It is permissible to call `tp_clear` from **`tp_dealloc`** to reduce code duplication and to guarantee that the object is always cleared before destruction. Beware that **`tp_clear`** might have already been called.

If the type is heap allocated (`Py_TPFLAGS_HEAPTYPE`), the deallocator should release the owned reference to its type object (via `Py_DECREF()`) after calling the type deallocator. See the example code below.:

```
static void
foo_dealloc(PyObject *op)
{
   foo_object *self = (foo_object *) op;
   PyObject_GC_UnTrack(self);
   Py_CLEAR(self->ref);
   Py_TYPE(self)->tp_free(self);
}
```

`tp_dealloc` must leave the exception status unchanged. If it needs to call something that might raise an exception, the exception state must be backed up first and restored later (after logging any exceptions with `PyErr_WriteUnraisable()`).

Example:

```
static void
foo_dealloc(PyObject *self)
{
    PyObject *exc = PyErr_GetRaisedException();

    if (PyObject_CallFinalizerFromDealloc(self) < 0
        // self was resurrected.
        goto done;
    }

    PyTypeObject *tp = Py_TYPE(self);

    if (tp->tp_flags & Py_TPFLAGS_HAVE_GC) {
        PyObject_GC_UnTrack(self);
    }
    // Optional, but convenient to avoid code dupli
    if (tp->tp_clear && tp->tp_clear(self) < 0) {
        PyErr_WriteUnraisable(self);
    }

    // Any additional destruction goes here.

    tp->tp_free(self);
    self = NULL;  // In case PyErr_WriteUnraisable(

    if (tp->tp_flags & Py_TPFLAGS_HEAPTYPE) {
        Py_CLEAR(tp);
    }

done:
    // Optional, if something was called that might
    // exception.
    if (PyErr_Occurred()) {
        PyErr_WriteUnraisable(self);
    }
    PyErr_SetRaisedException(exc);
}
```

`tp_dealloc` may be called from any Python thread, not just the thread which created the object (if the object becomes part of a refcount cycle, that cycle might be collected by a garbage collection on any thread). This is not a problem for Python API calls, since the thread on which **`tp_dealloc`** is called with an attached thread state. However, if the object being destroyed in turn destroys objects from some other C library, care should be taken to ensure that destroying those objects on the thread which called **`tp_dealloc`** will not violate any assumptions of the library.

Herencia:

Este campo es heredado por subtipos.

Ver también

Object Life Cycle for details about how this slot relates to other slots.

Py_ssize_t PyTypeObject.tp_vectorcall_offset

Un desplazamiento opcional a una función por instancia que implementa la llamada al objeto usando vectorcall protocol, una alternativa más eficiente del simple **`tp_call`**.

This field is only used if the flag **`Py_TPFLAGS_HAVE_VECTORCALL`** is set. If so, this must be a positive integer containing the offset in the instance of a **`vectorcallfunc`** pointer.

The *vectorcallfunc* pointer may be `NULL`, in which case the instance behaves as if **`Py_TPFLAGS_HAVE_VECTORCALL`** was not set: calling the instance falls back to **`tp_call`**.

Cualquier clase que establezca `_Py_TPFLAGS_HAVE_VECTORCALL` también debe establecer **`tp_call`** y asegurarse de que su comportamiento sea coherente con la función *vectorcallfunc*. Esto se puede hacer configurando *tp_call* en **`PyVectorcall_Call()`**.

Distinto en la versión 3.8: Antes de la versión 3.8, este slot se llamaba `tp_print`. En Python 2.x, se usó para imprimir en un archivo. En Python 3.0 a 3.7, no se usó.

Distinto en la versión 3.12: Before version 3.12, it was not recommended for mutable heap types to implement the vectorcall protocol. When a user sets **`__call__`** in Python code, only *tp_call* is updated, likely making it inconsistent with the vectorcall function. Since 3.12, setting `__call__` will disable vectorcall optimization by clearing the **`Py_TPFLAGS_HAVE_VECTORCALL`** flag.

Herencia:

This field is always inherited. However, the **`Py_TPFLAGS_HAVE_VECTORCALL`** flag is not always inherited. If it's not set, then the subclass won't use vectorcall, except when **`PyVectorcall_Call()`** is explicitly called.

getattrfunc PyTypeObject.tp_getattr

The corresponding slot ID `Py_tp_getattr` *is part of the Stable ABI.*

Un puntero opcional a la función «obtener atributo cadena de caracteres» (*get-attribute-string*).

Este campo está en desuso. Cuando se define, debe apuntar a una función que actúe igual que la función **`tp_getattro`**, pero tomando una cadena de caracteres C en lugar de un objeto de cadena Python para dar el nombre del atributo.

Herencia:

Group: **`tp_getattr`**, **`tp_getattro`**

Este campo es heredado por los subtipos junto con **`tp_getattro`**: un subtipo hereda ambos **`tp_getattr`** y **`tp_getattro`** de su base escriba cuando los subtipos **`tp_getattr`** y **`tp_getattro`** son ambos `NULL`.

setattrfunc PyTypeObject.tp_setattr

The corresponding slot ID `Py_tp_setattr` *is part of the Stable ABI.*

Un puntero opcional a la función para configurar y eliminar atributos.

Este campo está en desuso. Cuando se define, debe apuntar a una función que actúe igual que la función **`tp_setattro`**, pero tomando una cadena de caracteres C en lugar de un objeto de cadena Python para dar el nombre del atributo.

Herencia:

Group: **`tp_setattr`**, **`tp_setattro`**

Este campo es heredado por los subtipos junto con **`tp_setattro`**: un subtipo hereda ambos **`tp_setattr`** y **`tp_setattro`** de su base escriba cuando los subtipos **`tp_setattr`** y **`tp_setattro`** son ambos `NULL`.

PyAsyncMethods *PyTypeObject.tp_as_async

Puntero a una estructura adicional que contiene campos relevantes solo para los objetos que implementan los protocolos «esperable» (awaitable) y «iterador asíncrono» (asynchronous iterator) en el nivel C. Ver Estructuras de objetos asíncronos para más detalles.

Added in version 3.5: Anteriormente conocidos como `tp_compare` y `tp_reserved`.

Herencia:

El campo **`tp_as_async`** no se hereda, pero los campos contenidos se heredan individualmente.

reprfunc PyTypeObject.tp_repr

The corresponding slot ID `Py_tp_repr` is part of the Stable ABI.

Un puntero opcional a una función que implementa la función incorporada **`repr()`**.

La firma es la misma que para **`PyObject_Repr()`**:

```
PyObject *tp_repr(PyObject *self);
```

La función debe retornar una cadena de caracteres o un objeto Unicode. Idealmente, esta función debería retornar una cadena que, cuando se pasa a **`eval()`**, dado un entorno adecuado, retorna un objeto con el mismo valor. Si esto no es factible, debe retornar una cadena que comience con `'<'` y termine con `'>'` desde la cual se puede deducir tanto el tipo como el valor del objeto.

Herencia:

Este campo es heredado por subtipos.

Por defecto:

Cuando este campo no está configurado, se retorna una cadena de caracteres de la forma `<%s object at %p>`, donde `%s` se reemplaza por el nombre del tipo y `%p` por dirección de memoria del objeto.

PyNumberMethods *PyTypeObject.tp_as_number

Puntero a una estructura adicional que contiene campos relevantes solo para objetos que implementan el protocolo numérico. Estos campos están documentados en Estructuras de objetos de números.

Herencia:

El campo **`tp_as_number`** no se hereda, pero los campos contenidos se heredan individualmente.

PySequenceMethods *PyTypeObject.tp_as_sequence

Puntero a una estructura adicional que contiene campos relevantes solo para objetos que implementan el protocolo de secuencia. Estos campos están documentados en estructuras de secuencia.

Herencia:

El campo **`tp_as_sequence`** no se hereda, pero los campos contenidos se heredan individualmente.

PyMappingMethods *PyTypeObject.tp_as_mapping

Puntero a una estructura adicional que contiene campos relevantes solo para objetos que implementan el protocolo de mapeo. Estos campos están documentados en Estructuras de objetos mapeo.

Herencia:

El campo **`tp_as_mapping`** no se hereda, pero los campos contenidos se heredan individualmente.

hashfunc PyTypeObject.tp_hash

The corresponding slot ID `Py_tp_hash` *is part of the Stable ABI.*

Un puntero opcional a una función que implementa la función incorporada **`hash()`**.

La firma es la misma que para **`PyObject_Hash()`**:

```
Py_hash_t tp_hash(PyObject *);
```

El valor `-1` no debe retornarse como un valor de retorno normal; Cuando se produce un error durante el cálculo del valor *hash*, la función debe establecer una excepción y retornar `-1`.

When this field is not set (*and* **`tp_richcompare`** is not set), an attempt to take the hash of the object raises **`TypeError`**. This is the same as setting it to **`PyObject_HashNotImplemented()`**.

Este campo se puede establecer explícitamente en **`PyObject_HashNotImplemented()`** para bloquear la herencia del método *hash* de un tipo primario. Esto se interpreta como el equivalente de `__hash__ = None` en el nivel de Python, lo que hace que `isinstance(o, collections.Hashable)` retorne correctamente `False`. Tenga en cuenta que lo contrario también es cierto: establecer `__hash__ = None` en una clase en el nivel de Python dará como resultado que la ranura `tp_hash` se establezca en **`PyObject_HashNotImplemented()`**.

Herencia:

Group: **`tp_hash`**, **`tp_richcompare`**

Este campo es heredado por subtipos junto con **`tp_richcompare`**: un subtipo hereda ambos **`tp_richcompare`** y **`tp_hash`**, cuando los subtipos

`tp_richcompare` y **`tp_hash`** son ambos `NULL`.

Por defecto:

`PyBaseObject_Type` uses **`PyObject_GenericHash()`**.

ternaryfunc PyTypeObject.tp_call

The corresponding slot ID `Py_tp_call` *is part of the Stable ABI.*

Un puntero opcional a una función que implementa la llamada al objeto. Esto debería ser `NULL` si el objeto no es invocable. La firma es la misma que para **`PyObject_Call()`**:

```
PyObject *tp_call(PyObject *self, PyObject *args, P
```

Herencia:

Este campo es heredado por subtipos.

reprfunc PyTypeObject.tp_str

The corresponding slot ID `Py_tp_str` *is part of the Stable ABI.*

Un puntero opcional a una función que implementa la operación integrada **`str()`**. (Tenga en cuenta que **`str`** es un tipo ahora, y **`str()`** llama al constructor para ese tipo. Este constructor llama a **`PyObject_Str()`** para hacer el trabajo real, y **`PyObject_Str()`** llamará a este controlador.)

La firma es la misma que para **`PyObject_Str()`**:

```
PyObject *tp_str(PyObject *self);
```

La función debe retornar una cadena de caracteres o un objeto Unicode. Debe ser una representación de cadena «amigable» del objeto, ya que esta es la representación que será utilizada, entre otras cosas, por la función **`print()`**.

Herencia:

Este campo es heredado por subtipos.

Por defecto:

Cuando este campo no está configurado, se llama a **`PyObject_Repr()`** para retornar una representación de cadena de caracteres.

getattrofunc PyTypeObject.tp_getattro

The corresponding slot ID `Py_tp_getattro` *is part of the Stable ABI.*

Un puntero opcional a la función «obtener atributo» (*getattribute*).

La firma es la misma que para **`PyObject_GetAttr()`**:

```
PyObject *tp_getattro(PyObject *self, PyObject *att
```

Por lo general, es conveniente establecer este campo en **`PyObject_GenericGetAttr()`**, que implementa la forma normal de buscar atributos de objeto.

Herencia:

Group: **`tp_getattr`**, **`tp_getattro`**

Este campo es heredado por los subtipos junto con **`tp_getattr`**: un subtipo hereda ambos **`tp_getattr`** y **`tp_getattro`** de su base escriba cuando los subtipos **`tp_getattr`** y **`tp_getattro`** son ambos `NULL`.

Por defecto:

`PyBaseObject_Type` uses **`PyObject_GenericGetAttr()`**.

setattrofunc PyTypeObject.tp_setattro

The corresponding slot ID `Py_tp_setattro` *is part of the Stable ABI.*

Un puntero opcional a la función para configurar y eliminar atributos.

La firma es la misma que para `PyObject_SetAttr()`:

```
int tp_setattro(PyObject *self, PyObject *attr, PyO
```

Además, se debe admitir la configuración de *value* en `NULL` para eliminar un atributo. Por lo general, es conveniente establecer este campo en `PyObject_GenericSetAttr()`, que implementa la forma normal de establecer los atributos del objeto.

Herencia:

Group: `tp_setattr`, `tp_setattro`

Los subtipos heredan este campo junto con `tp_setattr`: un subtipo hereda ambos `tp_setattr` y `tp_setattro` de su base escriba cuando los subtipos `tp_setattr` y `tp_setattro` son ambos `NULL`.

Por defecto:

`PyBaseObject_Type` uses `PyObject_GenericSetAttr()`.

PyBufferProcs *PyTypeObject.tp_as_buffer

Puntero a una estructura adicional que contiene campos relevantes solo para objetos que implementan la interfaz del búfer. Estos campos están documentados en Estructuras de objetos búfer.

Herencia:

El campo `tp_as_buffer` no se hereda, pero los campos contenidos se heredan individualmente.

unsigned long PyTypeObject.tp_flags

Este campo es una máscara de bits de varias banderas. Algunas banderas indican semántica variante para ciertas situaciones; otros se utilizan para indicar que ciertos campos en el tipo de objeto (o en las estructuras de extensión a las que se hace referencia a través de **`tp_as_number`**, **`tp_as_sequence`**, **`tp_as_mapping`**, y **`tp_as_buffer`**) que históricamente no siempre estuvieron presentes son válidos; si dicho bit de bandera está claro, no se debe acceder a los campos de tipo que protege y se debe considerar que tienen un valor cero o `NULL`.

Herencia:

Inheritance of this field is complicated. Most flag bits are inherited individually, i.e. if the base type has a flag bit set, the subtype inherits this flag bit. The flag bits that pertain to extension structures are strictly inherited if the extension structure is inherited, i.e. the base type's value of the flag bit is copied into the subtype together with a pointer to the extension structure. The **`Py_TPFLAGS_HAVE_GC`** flag bit is inherited together with the **`tp_traverse`** and **`tp_clear`** fields, i.e. if the **`Py_TPFLAGS_HAVE_GC`** flag bit is clear in the subtype and the **`tp_traverse`** and **`tp_clear`** fields in the subtype exist and have `NULL` values.

Por defecto:

`PyBaseObject_Type` uses `Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE`.

Máscaras de bits:

Las siguientes máscaras de bits están definidas actualmente; estos se pueden unidos por *OR* usando el operador `|` para formar el valor del campo **`tp_flags`**. El macro **`PyType_HasFeature()`** toma un tipo y un valor de banderas, *tp* y *f*, y comprueba si `tp->tp_flags & f` no es cero.

Py_TPFLAGS_HEAPTYPE

This bit is set when the type object itself is allocated on the heap, for example, types created dynamically using **`PyType_FromSpec()`**. In this case, the **`ob_type`** field of its instances is considered a reference to the type, and the type object is INCREF'ed when a new instance is created, and DECREF'ed when an instance is destroyed (this does not apply to instances of subtypes; only the type referenced by the instance's ob_type gets INCREF'ed or DECREF'ed). Heap types should also support garbage collection as they can form a reference cycle with their own module object.

Herencia:

???

Py_TPFLAGS_BASETYPE

Part of the Stable ABI.
Este bit se establece cuando el tipo se puede usar como el tipo base de otro tipo. Si este bit es claro, el tipo no puede subtiparse (similar a una clase «final» en Java).

Herencia:

???

Py_TPFLAGS_READY

Este bit se establece cuando el objeto tipo ha sido completamente inicializado por `PyType_Ready()`.

Herencia:

???

Py_TPFLAGS_READYING

Este bit se establece mientras `PyType_Ready()` está en el proceso de inicialización del objeto tipo.

Herencia:

???

Py_TPFLAGS_HAVE_GC

Part of the Stable ABI.
This bit is set when the object supports garbage collection. If this bit is set, memory for new instances (see `tp_alloc`) must be allocated using `PyObject_GC_New` or `PyType_GenericAlloc()` and deallocated (see `tp_free`) using `PyObject_GC_Del()`. More information in section Apoyo a la recolección de basura cíclica.

Herencia:

Group: `Py_TPFLAGS_HAVE_GC`, `tp_traverse`, `tp_clear`

The `Py_TPFLAGS_HAVE_GC` flag bit is inherited together with the `tp_traverse` and `tp_clear` fields, i.e. if the **`Py_TPFLAGS_HAVE_GC`** flag bit is clear in the subtype and the **`tp_traverse`** and **`tp_clear`** fields in the subtype exist and have `NULL` values.

Py_TPFLAGS_DEFAULT

Part of the Stable ABI.
This is a bitmask of all the bits that pertain to the existence of certain fields in the type object and its extension structures. Currently, it includes the following bits: `Py_TPFLAGS_HAVE_STACKLESS_EXTENSION`.

Herencia:

???

Py_TPFLAGS_METHOD_DESCRIPTOR

Part of the Stable ABI since version 3.8.
Este bit indica que los objetos se comportan como métodos independientes.

Si este indicador está configurado para `type(meth)`, entonces:

- `meth.__get__(obj, cls)(*args, **kwds)` (con `obj` no `None`) debe ser equivalente a `meth(obj, *args, **kwds)`.
- `meth.__get__(None, cls)(*args, **kwds)` debe ser equivalente a `meth(*args, **kwds)`.

Este indicador (*flag*) permite una optimización para llamadas a métodos típicos como `obj.meth()`: evita crear un objeto temporal de «método vinculado» para `obj.meth`.

Added in version 3.8.

Herencia:

This flag is never inherited by types without the **`Py_TPFLAGS_IMMUTABLETYPE`** flag set. For extension types, it is inherited whenever **`tp_descr_get`** is inherited.

Py_TPFLAGS_MANAGED_DICT

This bit indicates that instances of the class have a **`__dict__`** attribute, and that the space for the dictionary is managed by the VM.

If this flag is set, **`Py_TPFLAGS_HAVE_GC`** should also be set.

The type traverse function must call **`PyObject_VisitManagedDict()`** and its clear function must call **`PyObject_ClearManagedDict()`**.

Added in version 3.12.

Herencia:

This flag is inherited unless the **`tp_dictoffset`** field is set in a superclass.

Py_TPFLAGS_MANAGED_WEAKREF

This bit indicates that instances of the class should be weakly referenceable.

Added in version 3.12.

Herencia:

This flag is inherited unless the **`tp_weaklistoffset`** field is set in a superclass.

Py_TPFLAGS_ITEMS_AT_END

Part of the Stable ABI since version 3.12.
Only usable with variable-size types, i.e. ones with non-zero **`tp_itemsize`**.

Indicates that the variable-sized portion of an instance of this type is at the end of the instance's memory area, at an offset of `Py_TYPE(obj)->tp_basicsize` (which may be different in each subclass).

When setting this flag, be sure that all superclasses either use this memory layout, or are not variable-sized. Python does not check this.

Added in version 3.12.

Herencia:

This flag is inherited.

Py_TPFLAGS_LONG_SUBCLASS

Py_TPFLAGS_LIST_SUBCLASS

Py_TPFLAGS_TUPLE_SUBCLASS

Py_TPFLAGS_BYTES_SUBCLASS

Py_TPFLAGS_UNICODE_SUBCLASS

Py_TPFLAGS_DICT_SUBCLASS

Py_TPFLAGS_BASE_EXC_SUBCLASS

Py_TPFLAGS_TYPE_SUBCLASS

Functions such as `PyLong_Check()` will call `PyType_FastSubclass()` with one of these flags to quickly determine if a type is a subclass of a built-in type; such specific checks are faster than a generic check, like `PyObject_IsInstance()`. Custom types that inherit from built-ins should have their `tp_flags` set appropriately, or the code that interacts with such types will behave differently depending on what kind of check is used.

Py_TPFLAGS_HAVE_FINALIZE

Este bit se establece cuando la ranura `tp_finalize` está presente en la estructura de tipo.

Added in version 3.4.

Obsoleto desde la versión 3.8: Este indicador ya no es necesario, ya que el intérprete asume que: el espacio `tp_finalize` siempre está presente en la estructura de tipos.

Py_TPFLAGS_HAVE_VECTORCALL

Part of the Stable ABI since version 3.12.
Este bit se establece cuando la clase implementa protocolo vectorcall. Consulte `tp_vectorcall_offset` para obtener más detalles.

Herencia:

This bit is inherited if `tp_call` is also inherited.

Added in version 3.8: as `_Py_TPFLAGS_HAVE_VECTORCALL`

Distinto en la versión 3.9.

Renamed to the current name, without the leading underscore. The old provisional name is soft deprecated.

Distinto en la versión 3.12: This flag is now removed from a class when the class's `__call__()` method is reassigned.

This flag can now be inherited by mutable classes.

Py_TPFLAGS_IMMUTABLETYPE

Este bit se establece para objetos de tipo que son inmutables: los atributos de tipo no se pueden establecer ni eliminar.

`PyType_Ready()` aplica automáticamente este indicador a static types.

Herencia:

Este flag no se hereda.

Added in version 3.10.

Py_TPFLAGS_DISALLOW_INSTANTIATION

No permita la creación de instancias del tipo: establezca `tp_new` en NULL y no cree la clave `__new__` en el diccionario de tipos.

La bandera debe establecerse antes de crear el tipo, no después. Por ejemplo, debe establecerse antes de que se llame a `PyType_Ready()` en el tipo.

La bandera se establece automáticamente en static types si `tp_base` es NULL o `&PyBaseObject_Type` y `tp_new` es NULL.

Herencia:

This flag is not inherited. However, subclasses will not be instantiable unless they provide a non-NULL `tp_new` (which is only possible via the C API).

Nota

To disallow instantiating a class directly but allow instantiating its subclasses (e.g. for an abstract base class), do not use this flag. Instead, make `tp_new` only succeed for subclasses.

Added in version 3.10.

Py_TPFLAGS_MAPPING

Este bit indica que las instancias de la clase pueden coincidir con los patrones de correspondencia cuando se utilizan como sujeto de un bloque `match`. Se configura automáticamente al registrar o subclasificar `collections.abc.Mapping`, y se desarma al registrar `collections.abc.Sequence`.

Nota

`Py_TPFLAGS_MAPPING` and `Py_TPFLAGS_SEQUENCE` are mutually exclusive; it is an error to enable both flags simultaneously.

Herencia:

This flag is inherited by types that do not already set `Py_TPFLAGS_SEQUENCE`.

Ver también

PEP 634 [https://peps.python.org/pep-0634/] - Coincidencia de patrones estructurales: especificación

Added in version 3.10.

Py_TPFLAGS_SEQUENCE

Este bit indica que las instancias de la clase pueden coincidir con los patrones de secuencia cuando se utilizan como sujeto de un bloque `match`. Se configura automáticamente al registrar o subclasificar `collections.abc.Sequence`, y se desarma al registrar `collections.abc.Mapping`.

Nota

`Py_TPFLAGS_MAPPING` and `Py_TPFLAGS_SEQUENCE` are mutually exclusive; it is an error to enable both flags simultaneously.

Herencia:

This flag is inherited by types that do not already set `Py_TPFLAGS_MAPPING`.

Ver también

PEP 634 [https://peps.python.org/pep-0634/] - Coincidencia de patrones estructurales: especificación

Added in version 3.10.

Py_TPFLAGS_VALID_VERSION_TAG

Internal. Do not set or unset this flag. To indicate that a class has changed call `PyType_Modified()`

Advertencia

This flag is present in header files, but is not be used. It will be removed in a future version of CPython

Py_TPFLAGS_HAVE_VERSION_TAG

This is a soft deprecated macro that does nothing. Historically, this would indicate that the `tp_version_tag` field was available and initialized.

Py_TPFLAGS_INLINE_VALUES

This bit indicates that instances of this type will have an «inline values» array (containing the object's attributes) placed directly after the end of the object.

This requires that `Py_TPFLAGS_HAVE_GC` is set.

Herencia:

Este flag no se hereda.

Added in version 3.13.

Py_TPFLAGS_IS_ABSTRACT

This bit indicates that this is an abstract type and therefore cannot be instantiated.

Herencia:

Este flag no se hereda.

Ver también

`abc`

Py_TPFLAGS_HAVE_STACKLESS_EXTENSION

Internal. Do not set or unset this flag. Historically, this was a reserved flag for use in Stackless Python.

Advertencia

This flag is present in header files, but is not be used. This may be removed in a future version of CPython.

const char *PyTypeObject.tp_doc

The corresponding slot ID `Py_tp_doc` *is part of the Stable ABI.*

An optional pointer to a NUL-terminated C string giving the docstring for this type object. This is exposed as the **`__doc__`** attribute on the type and instances of the type.

Herencia:

Este campo es *no* heredado por los subtipos.

traverseproc PyTypeObject.tp_traverse

The corresponding slot ID `Py_tp_traverse` *is part of the Stable ABI.*

An optional pointer to a traversal function for the garbage collector. This is only used if the **`Py_TPFLAGS_HAVE_GC`** flag bit is set. The signature is:

```
int tp_traverse(PyObject *self, visitproc visit, vo
```

Se puede encontrar más información sobre el esquema de recolección de basura de Python en la sección Apoyo a la recolección de basura cíclica.

The **`tp_traverse`** pointer is used by the garbage collector to detect reference cycles. A typical implementation of a **`tp_traverse`** function simply calls **`Py_VISIT()`** on each of the instance's members that are Python objects that the instance owns. For example, this is function **`local_traverse()`** from the **`_thread`** extension module:

```
static int
local_traverse(PyObject *op, visitproc visit, void
{
    localobject *self = (localobject *) op;
    Py_VISIT(self->args);
    Py_VISIT(self->kw);
    Py_VISIT(self->dict);
    return 0;
}
```

Tenga en cuenta que **`Py_VISIT()`** solo se llama a aquellos miembros que pueden participar en los ciclos de referencia. Aunque también hay un miembro `self->key`, solo puede ser `NULL` o una cadena de caracteres de Python y, por lo tanto, no puede ser parte de un ciclo de referencia.

Por otro lado, incluso si sabe que un miembro nunca puede ser parte de un ciclo, como ayuda para la depuración puede visitarlo de todos modos solo para que la función **`get_referents()`** del módulo **`gc`** lo incluirá.

Heap types (**`Py_TPFLAGS_HEAPTYPE`**) must visit their type with:

```
Py_VISIT(Py_TYPE(self));
```

It is only needed since Python 3.9. To support Python 3.8 and older, this line must be conditional:

```
#if PY_VERSION_HEX >= 0x03090000
    Py_VISIT(Py_TYPE(self));
#endif
```

If the **`Py_TPFLAGS_MANAGED_DICT`** bit is set in the **`tp_flags`** field, the traverse function must call **`PyObject_VisitManagedDict()`** like this:

```
PyObject_VisitManagedDict((PyObject*)self, visit, a
```

Advertencia

Al implementar **`tp_traverse`**, solo se deben visitar los miembros que tienen la instancia *owns* (por tener referencias fuertes). Por ejemplo, si un objeto admite referencias débiles a través de la ranura **`tp_weaklist`**, el puntero que respalda la lista vinculada (a lo que apunta *tp_weaklist*) **no** debe visitarse ya que la instancia no posee directamente las referencias débiles a sí misma (la lista de referencias débiles está ahí para respaldar la maquinaria de referencia débil, pero la instancia no tiene una fuerte referencia a los elementos dentro de ella, ya que pueden eliminarse incluso si la instancia todavía está viva).

Advertencia

The traversal function must not have any side effects. It must not modify the reference counts of any Python objects nor create or destroy any Python objects.

Note that `Py_VISIT()` requires the *visit* and *arg* parameters to `local_traverse()` to have these specific names; don't name them just anything.

Las instancias de heap-allocated types contienen una referencia a su tipo. Por lo tanto, su función transversal debe visitar `Py_TYPE(self)` o delegar esta responsabilidad llamando a `tp_traverse` de otro tipo asignado al heap (como una superclase asignada al heap). Si no es así, es posible que el objeto de tipo no se recolecte como basura.

Nota

The `tp_traverse` function can be called from any thread.

Distinto en la versión 3.9: Se espera que los tipos asignados al heap visiten `Py_TYPE(self)` en `tp_traverse`. En versiones anteriores de Python, debido al bug 40217 [https://bugs.python.org/issue40217], hacer esto puede provocar fallas en las subclases.

Herencia:

Group: `Py_TPFLAGS_HAVE_GC`, `tp_traverse`, `tp_clear`

This field is inherited by subtypes together with `tp_clear` and the `Py_TPFLAGS_HAVE_GC` flag bit: the flag bit, `tp_traverse`, and `tp_clear` are all inherited from the base type if they are all zero in the subtype.

inquiry PyTypeObject.tp_clear

The corresponding slot ID `Py_tp_clear` *is part of the Stable ABI.*

An optional pointer to a clear function. The signature is:

```
int tp_clear(PyObject *);
```

The purpose of this function is to break reference cycles that are causing a cyclic isolate so that the objects can be safely destroyed. A cleared object is a partially destroyed object; the object is not obligated to satisfy design invariants held during normal use.

`tp_clear` does not need to delete references to objects that can't participate in reference cycles, such as Python strings or Python integers. However, it may be convenient to clear all references, and write the type's **`tp_dealloc`** function to invoke **`tp_clear`** to avoid code duplication. (Beware that **`tp_clear`** might have already been called. Prefer calling idempotent functions like **`Py_CLEAR()`**.)

Any non-trivial cleanup should be performed in **`tp_finalize`** instead of **`tp_clear`**.

Nota

If **`tp_clear`** fails to break a reference cycle then the objects in the cyclic isolate may remain indefinitely uncollectable («leak»). See **`gc.garbage`**.

Nota

Referents (direct and indirect) might have already been cleared; they are not guaranteed to be in a consistent state.

Nota

The **`tp_clear`** function can be called from any thread.

Nota

An object is not guaranteed to be automatically cleared before its destructor (`tp_dealloc`) is called.

This function differs from the destructor (`tp_dealloc`) in the following ways:

- The purpose of clearing an object is to remove references to other objects that might participate in a reference cycle. The purpose of the destructor, on the other hand, is a superset: it must release *all* resources it owns, including references to objects that cannot participate in a reference cycle (e.g., integers) as well as the object's own memory (by calling `tp_free`).
- When **`tp_clear`** is called, other objects might still hold references to the object being cleared. Because of this, **`tp_clear`** must not deallocate the object's own memory (`tp_free`). The destructor, on the other hand, is only called when no (strong) references exist, and as such, must safely destroy the object itself by deallocating it.
- **`tp_clear`** might never be automatically called. An object's destructor, on the other hand, will be automatically called some time after the object becomes unreachable (i.e., either there are no references to the object or the object is a member of a cyclic isolate).

No guarantees are made about when, if, or how often Python automatically clears an object, except:

- Python will not automatically clear an object if it is reachable, i.e., there is a reference to it and it is not a member of a cyclic isolate.
- Python will not automatically clear an object if it has not been automatically finalized (see `tp_finalize`). (If the finalizer resurrected the object, the object may or may not be automatically finalized again before it is cleared.)
- If an object is a member of a cyclic isolate, Python will not automatically clear it if any member of the cyclic

isolate has not yet been automatically finalized (`tp_finalize`).
- Python will not destroy an object until after any automatic calls to its `tp_clear` function have returned. This ensures that the act of breaking a reference cycle does not invalidate the `self` pointer while `tp_clear` is still executing.
- Python will not automatically call `tp_clear` multiple times concurrently.

CPython currently only automatically clears objects as needed to break reference cycles in a cyclic isolate, but future versions might clear objects regularly before their destruction.

Taken together, all `tp_clear` functions in the system must combine to break all reference cycles. This is subtle, and if in any doubt supply a `tp_clear` function. For example, the tuple type does not implement a `tp_clear` function, because it's possible to prove that no reference cycle can be composed entirely of tuples. Therefore the `tp_clear` functions of other types are responsible for breaking any cycle containing a tuple. This isn't immediately obvious, and there's rarely a good reason to avoid implementing `tp_clear`.

Las implementaciones de `tp_clear` deberían descartar las referencias de la instancia a las de sus miembros que pueden ser objetos de Python, y establecer sus punteros a esos miembros en `NULL`, como en el siguiente ejemplo:

```
static int
local_clear(PyObject *op)
{
    localobject *self = (localobject *) op;
    Py_CLEAR(self->key);
    Py_CLEAR(self->args);
    Py_CLEAR(self->kw);
    Py_CLEAR(self->dict);
    return 0;
}
```

The `Py_CLEAR()` macro should be used, because clearing references is delicate: the reference to the contained object must not be released (via `Py_DECREF()`) until after the pointer to the contained object is set to `NULL`. This is because releasing the reference may cause the contained object to become trash, triggering a chain of reclamation activity that may include invoking arbitrary Python code (due to finalizers, or weakref callbacks, associated with the contained object). If it's possible for such code to reference *self* again, it's important that the pointer to the contained object be `NULL` at that time, so that *self* knows the contained object can no longer be used. The `Py_CLEAR()` macro performs the operations in a safe order.

If the `Py_TPFLAGS_MANAGED_DICT` bit is set in the `tp_flags` field, the clear function must call `PyObject_ClearManagedDict()` like this:

```
PyObject_ClearManagedDict((PyObject*)self);
```

Se puede encontrar más información sobre el esquema de recolección de basura de Python en la sección Apoyo a la recolección de basura cíclica.

Herencia:

Group: `Py_TPFLAGS_HAVE_GC`, `tp_traverse`, `tp_clear`

This field is inherited by subtypes together with `tp_traverse` and the `Py_TPFLAGS_HAVE_GC` flag bit: the flag bit, `tp_traverse`, and `tp_clear` are all inherited from the base type if they are all zero in the subtype.

Ver también

Object Life Cycle for details about how this slot relates to other slots.

richcmpfunc PyTypeObject.tp_richcompare

The corresponding [slot ID](#) `Py_tp_richcompare` is part of the [Stable ABI](#).

Un puntero opcional a la función de comparación enriquecida, cuya firma es:

```
PyObject *tp_richcompare(PyObject *self, PyObject *
```

Se garantiza que el primer parámetro será una instancia del tipo definido por **`PyTypeObject`**.

La función debería retornar el resultado de la comparación (generalmente `Py_True` o `Py_False`). Si la comparación no está definida, debe retornar `Py_NotImplemented`, si se produce otro error, debe retornar `NULL` y establecer una condición de excepción.

Las siguientes constantes se definen para ser utilizadas como el tercer argumento para **`tp_richcompare`** y para **`PyObject_RichCompare()`**:

Constante	Comparación
Py_LT	<
Py_LE	<=
Py_EQ	==
Py_NE	!=
Py_GT	>
Py_GE	>=

El siguiente macro está definido para facilitar la escritura de funciones de comparación enriquecidas:

Py_RETURN_RICHCOMPARE(VAL_A, VAL_B, op)

> Retorna `Py_True` o `Py_False` de la función, según el resultado de una comparación. `VAL_A` y `VAL_B` deben ser ordenados por operadores de comparación C (por ejemplo, pueden ser enteros o punto flotantes de C). El tercer argumento especifica la operación solicitada, como por ejemplo **`PyObject_RichCompare()`**.
>
> The returned value is a new strong reference.

En caso de error, establece una excepción y retorna `NULL` de la función.

Added in version 3.7.

Herencia:

Group: **`tp_hash`**, **`tp_richcompare`**

Este campo es heredado por subtipos junto con **`tp_hash`**: un subtipo hereda **`tp_richcompare`** y **`tp_hash`** cuando el subtipo **`tp_richcompare`** y **`tp_hash`** son ambos `NULL`.

Por defecto:

`PyBaseObject_Type` provides a **`tp_richcompare`** implementation, which may be inherited. However, if only **`tp_hash`** is defined, not even the inherited function is used and instances of the type will not be able to participate in any comparisons.

Py_ssize_t PyTypeObject.tp_weaklistoffset

While this field is still supported, **`Py_TPFLAGS_MANAGED_WEAKREF`** should be used instead, if at all possible.

If the instances of this type are weakly referenceable, this field is greater than zero and contains the offset in the instance structure of the weak reference list head (ignoring the GC header, if present); this offset is used by **`PyObject_ClearWeakRefs()`** and the `PyWeakref_*` functions. The instance structure needs to include a field of type PyObject* which is initialized to `NULL`.

No confunda este campo con **`tp_weaklist`**; ese es el encabezado de la lista para referencias débiles al objeto de tipo en sí.

It is an error to set both the **`Py_TPFLAGS_MANAGED_WEAKREF`** bit and **`tp_weaklistoffset`**.

Herencia:

Este campo es heredado por subtipos, pero consulte las reglas que se enumeran a continuación. Un subtipo puede anular este desplazamiento; Esto significa que el subtipo utiliza un encabezado de lista de referencia débil diferente que el tipo base. Dado que el encabezado de la lista siempre se encuentra a través de **`tp_weaklistoffset`**, esto no debería ser un problema.

Por defecto:

If the **`Py_TPFLAGS_MANAGED_WEAKREF`** bit is set in the **`tp_flags`** field, then **`tp_weaklistoffset`** will be set to a negative value, to indicate that it is unsafe to use this field.

getiterfunc PyTypeObject.tp_iter

The corresponding slot ID `Py_tp_iter` *is part of the Stable ABI.*

An optional pointer to a function that returns an iterator for the object. Its presence normally signals that the instances of this type are iterable (although sequences may be iterable without this function).

Esta función tiene la misma firma que **`PyObject_GetIter()`**:

```
PyObject *tp_iter(PyObject *self);
```

Herencia:

Este campo es heredado por subtipos.

iternextfunc PyTypeObject.tp_iternext

The corresponding slot ID `Py_tp_iternext` *is part of the Stable ABI.*

An optional pointer to a function that returns the next item in an iterator. The signature is:

```
PyObject *tp_iternext(PyObject *self);
```

Cuando el iterador está agotado, debe retornar `NULL`; a la excepción **`StopIteration`** puede o no establecerse. Cuando se produce otro error, también debe retornar `NULL`. Su presencia indica que las instancias de este tipo son iteradores.

Los tipos de iterador también deberían definir la función **`tp_iter`**, y esa función debería retornar la instancia de iterador en sí (no una nueva instancia de iterador).

Esta función tiene la misma firma que **`PyIter_Next()`**.

Herencia:

Este campo es heredado por subtipos.

struct PyMethodDef *PyTypeObject.tp_methods

The corresponding slot ID `Py_tp_methods` *is part of the Stable ABI.*

Un puntero opcional a un arreglo estático terminado en `NULL` de estructuras **`PyMethodDef`**, declarando métodos regulares de este tipo.

Para cada entrada en el arreglo, se agrega una entrada al diccionario del tipo (ver **`tp_dict`** a continuación) que contiene un descriptor *method*.

Herencia:

Los subtipos no heredan este campo (los métodos se heredan mediante un mecanismo diferente).

struct PyMemberDef *PyTypeObject.tp_members

The corresponding slot ID `Py_tp_members` *is part of the Stable ABI.*

Un puntero opcional a un arreglo estático terminado en `NULL` de estructuras **`PyMemberDef`**, declarando miembros de datos regulares (campos o ranuras) de instancias de este tipo.

Para cada entrada en el arreglo, se agrega una entrada al diccionario del tipo (ver **`tp_dict`** a continuación) que contiene un descriptor *member*.

Herencia:

Los subtipos no heredan este campo (los miembros se heredan mediante un mecanismo diferente).

struct PyGetSetDef *PyTypeObject.tp_getset

The corresponding slot ID `Py_tp_getset` *is part of the Stable ABI.*

Un puntero opcional a un arreglo estático terminado en `NULL` de estructuras **`PyGetSetDef`**, declarando atributos calculados de instancias de este tipo.

Para cada entrada en el arreglo, se agrega una entrada al diccionario del tipo (ver **`tp_dict`** a continuación) que contiene un descriptor *getset*.

Herencia:

Este campo no es heredado por los subtipos (los atributos computados se heredan a través de un mecanismo diferente).

PyTypeObject *PyTypeObject.tp_base

The corresponding slot ID `Py_tp_base` *is part of the Stable ABI.*

Un puntero opcional a un tipo base del que se heredan las propiedades de tipo. En este nivel, solo se admite una herencia única; La herencia múltiple requiere la creación dinámica de un objeto tipo llamando al metatipo.

Nota

La inicialización de ranuras está sujeta a las reglas de inicialización de globales. C99 requiere que los inicializadores sean «constantes de dirección». Los designadores de funciones como

`PyType_GenericNew()`, con conversión implícita a un puntero, son constantes de dirección C99 válidas.

However, the unary "&" operator applied to a non-static variable like `PyBaseObject_Type` is not required to produce an address constant. Compilers may support this (gcc does), MSVC does not. Both compilers are strictly standard conforming in this particular behavior.

En consecuencia, `tp_base` debe establecerse en la función *init* del módulo de extensión.

Herencia:

Este campo no es heredado por los subtipos (obviamente).

Por defecto:

Este campo predeterminado es `&PyBaseObject_Type` (que para los programadores de Python se conoce como el tipo **`objeto`**).

PyObject *PyTypeObject.tp_dict

El diccionario del tipo se almacena aquí por `PyType_Ready()`.

This field should normally be initialized to `NULL` before PyType_Ready is called; it may also be initialized to a dictionary containing initial attributes for the type. Once `PyType_Ready()` has initialized the type, extra attributes for the type may be added to this dictionary only if they don't correspond to overloaded operations (like `__add__()`). Once initialization for the type has finished, this field should be treated as read-only.

Some types may not store their dictionary in this slot. Use `PyType_GetDict()` to retrieve the dictionary for an arbitrary type.

Distinto en la versión 3.12: Internals detail: For static builtin types, this is always `NULL`. Instead, the dict for such types is stored on `PyInterpreterState`. Use **`PyType_GetDict()`** to get the dict for an arbitrary type.

Herencia:

Este campo no es heredado por los subtipos (aunque los atributos definidos aquí se heredan a través de un mecanismo diferente).

Por defecto:

Si este campo es `NULL`, **`PyType_Ready()`** le asignará un nuevo diccionario.

> **Advertencia**
>
> No es seguro usar **`PyDict_SetItem()`** en o modificar de otra manera a **`tp_dict`** con el diccionario C-API.

descrgetfunc PyTypeObject.tp_descr_get

The corresponding slot ID `Py_tp_descr_get` *is part of the Stable ABI.*

Un puntero opcional a una función «obtener descriptor» (*descriptor ger*).

La firma de la función es:

```
PyObject * tp_descr_get(PyObject *self, PyObject *o
```

Herencia:

Este campo es heredado por subtipos.

descrsetfunc PyTypeObject.tp_descr_set

The corresponding slot ID `Py_tp_descr_set` *is part of the Stable ABI.*

Un puntero opcional a una función para configurar y eliminar el valor de un descriptor.

La firma de la función es:

```
int tp_descr_set(PyObject *self, PyObject *obj, PyO
```

El argumento *value* se establece a `NULL` para borrar el valor.

Herencia:

Este campo es heredado por subtipos.

Py_ssize_t PyTypeObject.tp_dictoffset

While this field is still supported, `Py_TPFLAGS_MANAGED_DICT` should be used instead, if at all possible.

Si las instancias de este tipo tienen un diccionario que contiene variables de instancia, este campo no es cero y contiene el desplazamiento en las instancias del tipo del diccionario de variables de instancia; este desplazamiento es utilizado por `PyObject_GenericGetAttr()`.

No confunda este campo con `tp_dict`; ese es el diccionario para los atributos del tipo de objeto en sí.

The value specifies the offset of the dictionary from the start of the instance structure.

El `tp_dictoffset` debe considerarse como de solo escritura. Para obtener el puntero al diccionario, llame a `PyObject_GenericGetDict()`. Llamar a **`PyObject_GenericGetDict()`** puede necesitar asignar memoria para el diccionario, por lo que puede ser más eficiente llamar a `PyObject_GetAttr()` cuando se accede a un atributo en el objeto.

It is an error to set both the `Py_TPFLAGS_MANAGED_DICT` bit and `tp_dictoffset`.

Herencia:

This field is inherited by subtypes. A subtype should not override this offset; doing so could be unsafe, if C code tries to access the dictionary at the previous offset. To properly support inheritance, use **`Py_TPFLAGS_MANAGED_DICT`**.

Por defecto:

This slot has no default. For static types, if the field is `NULL` then no **`__dict__`** gets created for instances.

If the **`Py_TPFLAGS_MANAGED_DICT`** bit is set in the **`tp_flags`** field, then **`tp_dictoffset`** will be set to `-1`, to indicate that it is unsafe to use this field.

initproc PyTypeObject.tp_init

The corresponding slot ID `Py_tp_init` *is part of the Stable ABI.*

Un puntero opcional a una función de inicialización de instancia.

This function corresponds to the **`__init__()`** method of classes. Like **`__init__()`**, it is possible to create an instance without calling **`__init__()`**, and it is possible to reinitialize an instance by calling its **`__init__()`** method again.

La firma de la función es:

```
int tp_init(PyObject *self, PyObject *args, PyObjec
```

The self argument is the instance to be initialized; the *args* and *kwds* arguments represent positional and keyword arguments of the call to **`__init__()`**.

La función **`tp_init`**, si no es `NULL`, se llama cuando una instancia se crea normalmente llamando a su tipo, después de la función **`tp_new`** del tipo ha retornado una instancia del tipo. Si la función **`tp_new`** retorna una instancia de otro tipo que no es un subtipo del tipo original, no se llama la función **`tp_init`**; if **`tp_new`** retorna una instancia de un subtipo del tipo original, se llama al subtipo **`tp_init`**.

Retorna `0` en caso de éxito, `-1` y establece una excepción en caso de error.

Herencia:

Este campo es heredado por subtipos.

Por defecto:

Para tipos estáticos, este campo no tiene un valor predeterminado.

allocfunc PyTypeObject.tp_alloc

The corresponding slot ID `Py_tp_alloc` *is part of the Stable ABI.*

Un puntero opcional a una función de asignación de instancia.

La firma de la función es:

```
PyObject *tp_alloc(PyTypeObject *self, Py_ssize_t n
```

Herencia:

Static subtypes inherit this slot, which will be **`PyType_GenericAlloc()`** if inherited from **`object`**.

Heap subtypes do not inherit this slot.

Por defecto:

For heap subtypes, this field is always set to **`PyType_GenericAlloc()`**.

For static subtypes, this slot is inherited (see above).

newfunc PyTypeObject.tp_new

The corresponding slot ID `Py_tp_new` *is part of the Stable ABI.*

Un puntero opcional a una función de creación de instancias.

La firma de la función es:

```
PyObject *tp_new(PyTypeObject *subtype, PyObject *a
```

El argumento *subtype* es el tipo de objeto que se está creando; los argumentos *args* y *kwds* representan argumentos posicionales y de palabras clave de la llamada al tipo. Tenga en cuenta que *subtype* no tiene que ser igual al tipo cuya función **`tp_new`** es llamada; puede ser un subtipo de ese tipo (pero no un tipo no relacionado).

La función **`tp_new`** debería llamar a `subtype->tp_alloc(subtype, nitems)` para asignar espacio para el objeto, y luego hacer solo la inicialización adicional que sea absolutamente necesaria. La inicialización que se puede ignorar o repetir de forma segura debe colocarse en el controlador **`tp_init`**. Una buena regla general es que para los tipos inmutables, toda la inicialización debe tener lugar en **`tp_new`**, mientras que para los tipos mutables, la mayoría de las inicializaciones se deben diferir a **`tp_init`**.

Set the **`Py_TPFLAGS_DISALLOW_INSTANTIATION`** flag to disallow creating instances of the type in Python.

Herencia:

Este campo se hereda por subtipos, excepto que no lo heredan tipos estáticos cuyo **`tp_base`** es `NULL` o `&PyBaseObject_Type`.

Por defecto:

Para tipos estáticos, este campo no tiene ningún valor predeterminado. Esto significa que si la ranura se define como `NULL`, no se puede llamar al tipo para crear nuevas instancias; presumiblemente hay alguna otra forma de crear instancias, como una función de fábrica.

freefunc PyTypeObject.tp_free

The corresponding slot ID `Py_tp_free` is part of the Stable ABI.

Un puntero opcional a una función de desasignación de instancia. Su firma es:

```
void tp_free(void *self);
```

This function must free the memory allocated by `tp_alloc`.

Herencia:

Static subtypes inherit this slot, which will be `PyObject_Free()` if inherited from `object`. Exception: If the type supports garbage collection (i.e., the `Py_TPFLAGS_HAVE_GC` flag is set in `tp_flags`) and it would inherit `PyObject_Free()`, then this slot is not inherited but instead defaults to `PyObject_GC_Del()`.

Heap subtypes do not inherit this slot.

Por defecto:

For heap subtypes, this slot defaults to a deallocator suitable to match `PyType_GenericAlloc()` and the value of the `Py_TPFLAGS_HAVE_GC` flag.

For static subtypes, this slot is inherited (see above).

inquiry PyTypeObject.tp_is_gc

The corresponding slot ID `Py_tp_is_gc` *is part of the Stable ABI.*

Un puntero opcional a una función llamada por el recolector de basura.

The garbage collector needs to know whether a particular object is collectible or not. Normally, it is sufficient to look at the object's type's `tp_flags` field, and check the `Py_TPFLAGS_HAVE_GC` flag bit. But some types have a mixture of statically and dynamically allocated instances, and the statically allocated instances are not collectible. Such types should define this function; it should return `1` for a collectible instance, and `0` for a non-collectible instance. The signature is:

```
int tp_is_gc(PyObject *self);
```

(El único ejemplo de esto son los tipos en sí. El metatipo, **`PyType_Type`**, define esta función para distinguir entre tipos estática y dinámicamente asignados.)

Herencia:

Este campo es heredado por subtipos.

Por defecto:

This slot has no default. If this field is `NULL`, **`Py_TPFLAGS_HAVE_GC`** is used as the functional equivalent.

PyObject *PyTypeObject.tp_bases

The corresponding slot ID `Py_tp_bases` *is part of the Stable ABI.*

Tupla de tipos base.

This field should be set to `NULL` and treated as read-only. Python will fill it in when the type is **`initialized`**.

For dynamically created classes, the **`Py_tp_bases slot`** can be used instead of the *bases* argument of **`PyType_FromSpecWithBases()`**. The argument form is preferred.

Advertencia

Multiple inheritance does not work well for statically defined types. If you set `tp_bases` to a tuple, Python will not raise an error, but some slots will only be inherited from the first base.

Herencia:

Este campo no se hereda.

PyObject *PyTypeObject.tp_mro

Tupla que contiene el conjunto expandido de tipos base, comenzando con el tipo en sí y terminando con `object`, en orden de resolución de método.

This field should be set to `NULL` and treated as read-only. Python will fill it in when the type is `initialized`.

Herencia:

Este campo no se hereda; se calcula fresco por `PyType_Ready()`.

PyObject *PyTypeObject.tp_cache

No usado. Solo para uso interno.

Herencia:

Este campo no se hereda.

void *PyTypeObject.tp_subclasses

A collection of subclasses. Internal use only. May be an invalid pointer.

To get a list of subclasses, call the Python method `__subclasses__()`.

Distinto en la versión 3.12: For some types, this field does not hold a valid PyObject*. The type was changed to void* to indicate this.

Herencia:

Este campo no se hereda.

PyObject *PyTypeObject.tp_weaklist

Cabecera de lista de referencia débil, para referencias débiles a este tipo de objeto. No heredado Solo para uso interno.

Distinto en la versión 3.12: Internals detail: For the static builtin types this is always `NULL`, even if weakrefs are added. Instead, the weakrefs for each are stored on `PyInterpreterState`. Use the public C-API or the internal `_PyObject_GET_WEAKREFS_LISTPTR()` macro to avoid the distinction.

Herencia:

Este campo no se hereda.

destructor PyTypeObject.tp_del

The corresponding slot ID `Py_tp_del` *is part of the Stable ABI.*

Este campo está en desuso. Use **`tp_finalize`** en su lugar.

unsigned int PyTypeObject.tp_version_tag

Se usa para indexar en el caché de métodos. Solo para uso interno.

Herencia:

Este campo no se hereda.

destructor PyTypeObject.tp_finalize

The corresponding slot ID `Py_tp_finalize` *is part of the Stable ABI since version 3.5.*

An optional pointer to an instance finalization function. This is the C implementation of the **`__del__()`** special method. Its signature is:

```
void tp_finalize(PyObject *self);
```

The primary purpose of finalization is to perform any non-trivial cleanup that must be performed before the object is destroyed, while the object and any other objects it directly or indirectly references are still in a consistent state. The finalizer is allowed to execute arbitrary Python code.

Before Python automatically finalizes an object, some of the

object's direct or indirect referents might have themselves been automatically finalized. However, none of the referents will have been automatically cleared (`tp_clear`) yet.

Other non-finalized objects might still be using a finalized object, so the finalizer must leave the object in a sane state (e.g., invariants are still met).

Nota

After Python automatically finalizes an object, Python might start automatically clearing (`tp_clear`) the object and its referents (direct and indirect). Cleared objects are not guaranteed to be in a consistent state; a finalized object must be able to tolerate cleared referents.

Nota

An object is not guaranteed to be automatically finalized before its destructor (`tp_dealloc`) is called. It is recommended to call `PyObject_CallFinalizerFromDealloc()` at the beginning of `tp_dealloc` to guarantee that the object is always finalized before destruction.

Nota

The `tp_finalize` function can be called from any thread, although the GIL will be held.

Nota

The `tp_finalize` function can be called during shutdown, after some global variables have been deleted. See the documentation of the `__del__()` method for details.

When Python finalizes an object, it behaves like the following algorithm:

1. Python might mark the object as *finalized*. Currently, Python always marks objects whose type supports garbage collection (i.e., the `Py_TPFLAGS_HAVE_GC` flag is set in `tp_flags`) and never marks other types of objects; this might change in a future version.
2. If the object is not marked as *finalized* and its **`tp_finalize`** finalizer function is non-`NULL`, the finalizer function is called.
3. If the finalizer function was called and the finalizer made the object reachable (i.e., there is a reference to the object and it is not a member of a cyclic isolate), then the finalizer is said to have *resurrected* the object. It is unspecified whether the finalizer can also resurrect the object by adding a new reference to the object that does not make it reachable, i.e., the object is (still) a member of a cyclic isolate.
4. If the finalizer resurrected the object, the object's pending destruction is canceled and the object's *finalized* mark might be removed if present. Currently, Python never removes the *finalized* mark; this might change in a future version.

Automatic finalization refers to any finalization performed by Python except via calls to `PyObject_CallFinalizer()` or `PyObject_CallFinalizerFromDealloc()`. No guarantees are made about when, if, or how often an object is automatically finalized, except:

- Python will not automatically finalize an object if it is reachable, i.e., there is a reference to it and it is not a member of a cyclic isolate.
- Python will not automatically finalize an object if finalizing it would not mark the object as *finalized*. Currently, this applies to objects whose type does not support garbage collection, i.e., the `Py_TPFLAGS_HAVE_GC` flag is not set. Such objects can still be manually finalized by calling `PyObject_CallFinalizer()` or `PyObject_CallFinalizerFromDealloc()`.
- Python will not automatically finalize any two

members of a cyclic isolate concurrently.

- Python will not automatically finalize an object after it has automatically cleared (`tp_clear`) the object.
- If an object is a member of a cyclic isolate, Python will not automatically finalize it after automatically clearing (see `tp_clear`) any other member.
- Python will automatically finalize every member of a cyclic isolate before it automatically clears (see `tp_clear`) any of them.
- If Python is going to automatically clear an object (`tp_clear`), it will automatically finalize the object first.

Python currently only automatically finalizes objects that are members of a cyclic isolate, but future versions might finalize objects regularly before their destruction.

To manually finalize an object, do not call this function directly; call `PyObject_CallFinalizer()` or `PyObject_CallFinalizerFromDealloc()` instead.

`tp_finalize` should leave the current exception status unchanged. The recommended way to write a non-trivial finalizer is to back up the exception at the beginning by calling `PyErr_GetRaisedException()` and restore the exception at the end by calling `PyErr_SetRaisedException()`. If an exception is encountered in the middle of the finalizer, log and clear it with `PyErr_WriteUnraisable()` or `PyErr_FormatUnraisable()`. For example:

```
static void
foo_finalize(PyObject *self)
{
    // Save the current exception, if any.
    PyObject *exc = PyErr_GetRaisedException();

    // ...

    if (do_something_that_might_raise() != success_
        PyErr_WriteUnraisable(self);
        goto done;
    }

done:
    // Restore the saved exception.  This silently
    // raised above, so be sure to call PyErr_Write
    // necessary.
    PyErr_SetRaisedException(exc);
}
```

Herencia:

Este campo es heredado por subtipos.

Added in version 3.4.

Distinto en la versión 3.8: Before version 3.8 it was necessary to set the **`Py_TPFLAGS_HAVE_FINALIZE`** flags bit in order for this field to be used. This is no longer required.

Ver también

- **PEP 442** [https://peps.python.org/pep-0442/]: «Safe object finalization»
- Object Life Cycle for details about how this slot relates to other slots.
- **`PyObject_CallFinalizer()`**
- **`PyObject_CallFinalizerFromDealloc()`**

vectorcallfunc PyTypeObject.tp_vectorcall

The corresponding slot ID `Py_tp_vectorcall` *is part of the Stable ABI since version 3.14.*

A vectorcall function to use for calls of this type object (rather than instances). In other words, `tp_vectorcall` can be used to optimize `type.__call__`, which typically returns a new instance of *type*.

As with any vectorcall function, if `tp_vectorcall` is `NULL`, the *tp_call* protocol (`Py_TYPE(type)->tp_call`) is used instead.

Nota

The vectorcall protocol requires that the vectorcall function has the same behavior as the corresponding `tp_call`. This means that `type->tp_vectorcall` must match the behavior of `Py_TYPE(type)->tp_call`.

Specifically, if *type* uses the default metaclass, `type->tp_vectorcall` must behave the same as PyType_Type->tp_call, which:

- calls `type->tp_new`,
- if the result is a subclass of *type*, calls `type->tp_init` on the result of `tp_new`, and
- returns the result of `tp_new`.

Typically, `tp_vectorcall` is overridden to optimize this process for specific **`tp_new`** and **`tp_init`**. When doing this for user-subclassable types, note that both can be overridden (using **`__new__()`** and **`__init__()`**, respectively).

Herencia:

Este campo nunca se hereda.

Added in version 3.9: (el campo existe desde 3.8 pero solo se usa desde 3.9)

unsigned char PyTypeObject.tp_watched

Internal. Do not use.

Added in version 3.12.

Tipos estáticos

Tradicionalmente, los tipos definidos en el código C son *static*, es decir, una estructura estática **`PyTypeObject`** se define directamente en el código y se inicializa usando **`PyType_Ready()`**.

Esto da como resultado tipos que están limitados en relación con los tipos definidos en Python:

- Los tipos estáticos están limitados a una base, es decir, no pueden usar herencia múltiple.
- Los objetos de tipo estático (pero no necesariamente sus instancias) son inmutables. No es posible agregar o modificar los atributos del objeto tipo desde Python.
- Los objetos de tipo estático se comparten en sub intérpretes, por lo que no deben incluir ningún estado específico del sub interpretador.

Also, since **`PyTypeObject`** is only part of the Limited API as an opaque struct, any extension modules using static types must be compiled for a specific Python minor version.

Tipos Heap

An alternative to static types is *heap-allocated types*, or *heap types* for short, which correspond closely to classes created by Python's `class` statement. Heap types have the **`Py_TPFLAGS_HEAPTYPE`** flag set.

This is done by filling a **`PyType_Spec`** structure and calling **`PyType_FromSpec()`**, **`PyType_FromSpecWithBases()`**, **`PyType_FromModuleAndSpec()`**, or **`PyType_FromMetaclass()`**.

Estructuras de objetos de números

type PyNumberMethods

Esta estructura contiene punteros a las funciones que utiliza un objeto para implementar el protocolo numérico. Cada función es utilizada por la función de un nombre similar documentado en la sección Protocolo de números.

Aquí está la definición de la estructura:

```
typedef struct {
     binaryfunc nb_add;
     binaryfunc nb_subtract;
     binaryfunc nb_multiply;
     binaryfunc nb_remainder;
     binaryfunc nb_divmod;
     ternaryfunc nb_power;
     unaryfunc nb_negative;
     unaryfunc nb_positive;
     unaryfunc nb_absolute;
     inquiry nb_bool;
     unaryfunc nb_invert;
     binaryfunc nb_lshift;
     binaryfunc nb_rshift;
     binaryfunc nb_and;
     binaryfunc nb_xor;
     binaryfunc nb_or;
     unaryfunc nb_int;
     void *nb_reserved;
     unaryfunc nb_float;

     binaryfunc nb_inplace_add;
     binaryfunc nb_inplace_subtract;
     binaryfunc nb_inplace_multiply;
     binaryfunc nb_inplace_remainder;
     ternaryfunc nb_inplace_power;
     binaryfunc nb_inplace_lshift;
     binaryfunc nb_inplace_rshift;
     binaryfunc nb_inplace_and;
     binaryfunc nb_inplace_xor;
     binaryfunc nb_inplace_or;

     binaryfunc nb_floor_divide;
     binaryfunc nb_true_divide;
     binaryfunc nb_inplace_floor_divide;
```

```
    binaryfunc nb_inplace_true_divide;

    unaryfunc nb_index;

    binaryfunc nb_matrix_multiply;
    binaryfunc nb_inplace_matrix_multiply;
} PyNumberMethods;
```

Nota

Las funciones binarias y ternarias deben verificar el tipo de todos sus operandos e implementar las conversiones necesarias (al menos uno de los operandos es una instancia del tipo definido). Si la operación no está definida para los operandos dados, las funciones binarias y ternarias deben retornar `Py_NotImplemented`, si se produce otro error, deben retornar `NULL` y establecer una excepción.

Nota

The **`nb_reserved`** field should always be `NULL`. It was previously called **`nb_long`**, and was renamed in Python 3.0.1.

binaryfunc PyNumberMethods.nb_add

The corresponding slot ID `Py_nb_add` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_subtract

The corresponding slot ID `Py_nb_subtract` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_multiply

The corresponding slot ID `Py_nb_multiply` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_remainder

The corresponding slot ID `Py_nb_remainder` *is part of the*

Stable ABI.

binaryfunc PyNumberMethods.nb_divmod

The corresponding slot ID `Py_nb_divmod` *is part of the Stable ABI*.

ternaryfunc PyNumberMethods.nb_power

The corresponding slot ID `Py_nb_power` *is part of the Stable ABI*.

unaryfunc PyNumberMethods.nb_negative

The corresponding slot ID `Py_nb_negative` *is part of the Stable ABI*.

unaryfunc PyNumberMethods.nb_positive

The corresponding slot ID `Py_nb_positive` *is part of the Stable ABI*.

unaryfunc PyNumberMethods.nb_absolute

The corresponding slot ID `Py_nb_absolute` *is part of the Stable ABI*.

inquiry PyNumberMethods.nb_bool

The corresponding slot ID `Py_nb_bool` *is part of the Stable ABI*.

unaryfunc PyNumberMethods.nb_invert

The corresponding slot ID `Py_nb_invert` *is part of the Stable ABI*.

binaryfunc PyNumberMethods.nb_lshift

The corresponding slot ID `Py_nb_lshift` *is part of the Stable ABI*.

binaryfunc PyNumberMethods.nb_rshift

The corresponding slot ID `Py_nb_rshift` *is part of the Stable ABI*.

binaryfunc PyNumberMethods.nb_and

The corresponding slot ID `Py_nb_and` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_xor

The corresponding slot ID `Py_nb_xor` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_or

The corresponding slot ID `Py_nb_or` *is part of the Stable ABI.*

unaryfunc PyNumberMethods.nb_int

The corresponding slot ID `Py_nb_int` *is part of the Stable ABI.*

void *PyNumberMethods.nb_reserved

unaryfunc PyNumberMethods.nb_float

The corresponding slot ID `Py_nb_float` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_inplace_add

The corresponding slot ID `Py_nb_inplace_add` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_inplace_subtract

The corresponding slot ID `Py_nb_inplace_subtract` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_inplace_multiply

The corresponding slot ID `Py_nb_inplace_multiply` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_inplace_remainder

The corresponding slot ID `Py_nb_inplace_remainder` *is part of the Stable ABI.*

ternaryfunc PyNumberMethods.nb_inplace_power

The corresponding slot ID `Py_nb_inplace_power` *is part of* *the Stable ABI.*

binaryfunc PyNumberMethods.nb_inplace_lshift

The corresponding slot ID `Py_nb_inplace_lshift` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_inplace_rshift

The corresponding slot ID `Py_nb_inplace_rshift` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_inplace_and

The corresponding slot ID `Py_nb_inplace_and` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_inplace_xor

The corresponding slot ID `Py_nb_inplace_xor` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_inplace_or

The corresponding slot ID `Py_nb_inplace_or` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_floor_divide

The corresponding slot ID `Py_nb_floor_divide` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_true_divide

The corresponding slot ID `Py_nb_true_divide` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_inplace_floor_divide

The corresponding slot ID `Py_nb_inplace_floor_divide` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_inplace_true_divide

The corresponding slot ID `Py_nb_inplace_true_divide` *is part of the Stable ABI.*

unaryfunc PyNumberMethods.nb_index

The corresponding slot ID `Py_nb_index` *is part of the Stable ABI.*

binaryfunc PyNumberMethods.nb_matrix_multiply

The corresponding slot ID `Py_nb_matrix_multiply` *is part of the Stable ABI since version 3.5.*

binaryfunc PyNumberMethods.nb_inplace_matrix_multiply

The corresponding slot ID `Py_nb_inplace_matrix_multiply` *is part of the Stable ABI since version 3.5.*

Estructuras de objetos mapeo

type PyMappingMethods

Esta estructura contiene punteros a las funciones que utiliza un objeto para implementar el protocolo de mapeo. Tiene tres miembros:

lenfunc PyMappingMethods.mp_length

The corresponding slot ID `Py_mp_length` *is part of the Stable ABI.*

Esta función es utilizada por **`PyMapping_Size()`** y **`PyObject_Size()`**, y tiene la misma firma. Esta ranura puede establecerse en `NULL` si el objeto no tiene una longitud definida.

binaryfunc PyMappingMethods.mp_subscript

The corresponding slot ID `Py_mp_subscript` *is part of the Stable ABI.*

Esta función es utilizada por **`PyObject_GetItem()`** y **`PySequence_GetSlice()`**, y tiene la misma firma que **`PyObject_GetItem()`**. Este espacio debe llenarse para que la función **`PyMapping_Check()`** retorna `1`, de lo contrario puede ser `NULL`.

objobjargproc PyMappingMethods.mp_ass_subscript

The corresponding slot ID `Py_mp_ass_subscript` *is part of the Stable ABI.*

This function is used by **`PyObject_SetItem()`**, **`PyObject_DelItem()`**, **`PySequence_SetSlice()`** and **`PySequence_DelSlice()`**. It has the same signature as **`PyObject_SetItem()`**, but *v* can also be set to `NULL` to delete an item. If this slot is `NULL`, the object does not support item assignment and deletion.

Estructuras de objetos secuencia

type PySequenceMethods

Esta estructura contiene punteros a las funciones que utiliza un objeto para implementar el protocolo de secuencia.

lenfunc PySequenceMethods.sq_length

The corresponding slot ID `Py_sq_length` *is part of the Stable ABI.*

Esta función es utilizada por **`PySequence_Size()`** y **`PyObject_Size()`**, y tiene la misma firma. También se usa para manejar índices negativos a través de los espacios **`sq_item`** y **`sq_ass_item`**.

binaryfunc PySequenceMethods.sq_concat

The corresponding slot ID `Py_sq_concat` *is part of the Stable ABI.*

Esta función es utilizada por **`PySequence_Concat()`** y tiene la misma firma. También es utilizado por el operador `+`, después de intentar la suma numérica a través de la ranura **`nb_add`**.

ssizeargfunc PySequenceMethods.sq_repeat

The corresponding slot ID `Py_sq_repeat` *is part of the Stable ABI.*

Esta función es utilizada por **`PySequence_Repeat()`** y tiene la misma firma. También es utilizado por el operador `*`, después de intentar la multiplicación numérica a través de la ranura **`nb_multiply`**.

ssizeargfunc PySequenceMethods.sq_item

The corresponding slot ID `Py_sq_item` *is part of the Stable ABI.*

Esta función es utilizada por **`PySequence_GetItem()`** y tiene la misma firma. También es utilizado por **`PyObject_GetItem()`**, después de intentar la suscripción a través de la ranura **`mp_subscript`**. Este espacio debe llenarse para que la función **`PySequence_Check()`** retorna `1`, de lo contrario puede ser `NULL`.

Negative indexes are handled as follows: if the **`sq_length`** slot is filled, it is called and the sequence length is used to compute a positive index which is passed to **`sq_item`**. If **`sq_length`** is `NULL`, the index is passed as is to the function.

ssizeobjargproc PySequenceMethods.sq_ass_item

The corresponding slot ID `Py_sq_ass_item` *is part of the Stable ABI.*

Esta función es utilizada por **`PySequence_SetItem()`** y tiene la misma firma. También lo usan **`PyObject_SetItem()`** y **`PyObject_DelItem()`**, después de intentar la asignación y eliminación del elemento a través de la ranura **`mp_ass_subscript`**. Este espacio puede dejarse en `NULL` si el objeto no admite la asignación y eliminación de elementos.

objobjproc PySequenceMethods.sq_contains

The corresponding slot ID `Py_sq_contains` *is part of the Stable ABI.*

Esta función puede ser utilizada por **`PySequence_Contains()`** y tiene la misma firma. Este espacio puede dejarse en `NULL`, en este caso

`PySequence_Contains()` simplemente atraviesa la secuencia hasta que encuentra una coincidencia.

binaryfunc PySequenceMethods.sq_inplace_concat

The corresponding slot ID `Py_sq_inplace_concat` is part of the Stable ABI.

Esta función es utilizada por **`PySequence_InPlaceConcat()`** y tiene la misma firma. Debería modificar su primer operando y retornarlo. Este espacio puede dejarse en `NULL`, en este caso **`PySequence_InPlaceConcat()`** volverá a **`PySequence_Concat()`**. También es utilizado por la asignación aumentada `+=`, después de intentar la suma numérica en el lugar a través de la ranura **`nb_inplace_add`**.

ssizeargfunc PySequenceMethods.sq_inplace_repeat

The corresponding slot ID `Py_sq_inplace_repeat` is part of the Stable ABI.

Esta función es utilizada por **`PySequence_InPlaceRepeat()`** y tiene la misma firma. Debería modificar su primer operando y retornarlo. Este espacio puede dejarse en `NULL`, en este caso **`PySequence_InPlaceRepeat()`** volverá a **`PySequence_Repeat()`**. También es utilizado por la asignación aumentada `*=`, después de intentar la multiplicación numérica en el lugar a través de la ranura **`nb_inplace_multiply`**.

Estructuras de objetos búfer

type PyBufferProcs

Esta estructura contiene punteros a las funciones requeridas por Buffer protocol. El protocolo define cómo un objeto exportador puede exponer sus datos internos a objetos de consumo.

getbufferproc PyBufferProcs.bf_getbuffer

The corresponding slot ID `Py_bf_getbuffer` *is part of the Stable ABI since version 3.11.*

La firma de esta función es:

```
int (PyObject *exporter, Py_buffer *view, int flags
```

Maneja una solicitud a *exporter* para completar *view* según lo especificado por *flags*. Excepto por el punto (3), una implementación de esta función DEBE seguir estos pasos:

1. Check if the request can be met. If not, raise **`BufferError`**, set view->obj to `NULL` and return `-1`.
2. Rellene los campos solicitados.
3. Incrementa un contador interno para el número de exportaciones (*exports*).
4. Set view->obj to *exporter* and increment view->obj.
5. Retorna `0`.

Thread safety:

In the free-threaded build, implementations must ensure:

- The export counter increment in step (3) is atomic.
- The underlying buffer data remains valid and at a stable memory location for the lifetime of all exports.
- For objects that support resizing or reallocation (such as **`bytearray`**), the export counter is checked atomically before such operations, and **`BufferError`** is raised if exports exist.
- The function is safe to call concurrently from multiple threads.

See also Thread safety for memoryview objects for the Python-level thread safety guarantees of **`memoryview`** objects.

Si *exporter* es parte de una cadena o árbol de proveedores de búfer, se pueden usar dos esquemas principales:

- Re-export: Each member of the tree acts as the exporting object and sets view->obj to a new reference to itself.
- Redirect: The buffer request is redirected to the root object of the tree. Here, view->obj will be a new reference to the root object.

Los campos individuales de *view* se describen en la sección Estructura de búfer, las reglas sobre cómo debe reaccionar un exportador a solicitudes específicas se encuentran en la sección Tipos de solicitud de búfer.

Toda la memoria señalada en la estructura **`Py_buffer`** pertenece al exportador y debe permanecer válida hasta que no queden consumidores. **`format`**, **`shape`**, **`strides`**, **`suboffsets`** y **`internal`** son de solo lectura para el consumidor.

`PyBuffer_FillInfo()` proporciona una manera fácil de exponer un búfer de bytes simple mientras se trata correctamente con todos los tipos de solicitud.

`PyObject_GetBuffer()` es la interfaz para el consumidor que envuelve esta función.

releasebufferproc PyBufferProcs.bf_releasebuffer

The corresponding slot ID `Py_bf_releasebuffer` *is part of the Stable ABI since version 3.11.*

La firma de esta función es:

```
void (PyObject *exporter, Py_buffer *view);
```

Maneja una solicitud para liberar los recursos del búfer. Si no es necesario liberar recursos, **`PyBufferProcs.bf_releasebuffer`** puede ser `NULL`. De lo contrario, una implementación estándar de esta función tomará estos pasos opcionales:

1. Disminuir un contador interno para el número de exportaciones.

2. Si el contador es `0`, libera toda la memoria asociada con *view*.

Thread safety:

In the free-threaded build:

- The export counter decrement in step (1) must be atomic.
- Resource cleanup when the counter reaches zero must be done atomically, as the final release may race with concurrent releases from other threads and dellocation must only happen once.

El exportador DEBE utilizar el campo **`internal`** para realizar un seguimiento de los recursos específicos del búfer. Se garantiza que este campo permanecerá constante, mientras que un consumidor PUEDE pasar una copia del búfer original como argumento *view*.

This function MUST NOT decrement view->obj, since that is done automatically in **`PyBuffer_Release()`** (this scheme is useful for breaking reference cycles).

`PyBuffer_Release()` es la interfaz para el consumidor que envuelve esta función.

Estructuras de objetos asíncronos

Added in version 3.5.

type PyAsyncMethods

Esta estructura contiene punteros a las funciones requeridas para implementar objetos «esperable» (awaitable) y «iterador asincrónico» (asynchronous iterator).

Aquí está la definición de la estructura:

```
typedef struct {
    unaryfunc am_await;
    unaryfunc am_aiter;
    unaryfunc am_anext;
    sendfunc am_send;
} PyAsyncMethods;
```

unaryfunc PyAsyncMethods.am_await

The corresponding slot ID `Py_am_await` *is part of the Stable ABI since version 3.5.*

La firma de esta función es:

```
PyObject *am_await(PyObject *self);
```

The returned object must be an iterator, i.e. **`PyIter_Check()`** must return `1` for it.

Este espacio puede establecerse en `NULL` si un objeto no es awaitable.

unaryfunc PyAsyncMethods.am_aiter

The corresponding slot ID `Py_am_aiter` *is part of the Stable ABI since version 3.5.*

La firma de esta función es:

```
PyObject *am_aiter(PyObject *self);
```

Must return an asynchronous iterator object. See **`__anext__()`** for details.

Este espacio puede establecerse en `NULL` si un objeto no implementa el protocolo de iteración asincrónica.

unaryfunc PyAsyncMethods.am_anext

The corresponding slot ID `Py_am_anext` *is part of the Stable ABI since version 3.5.*

La firma de esta función es:

```
PyObject *am_anext(PyObject *self);
```

Must return an awaitable object. See **`__anext__()`** for details. This slot may be set to `NULL`.

sendfunc PyAsyncMethods.am_send

The corresponding slot ID `Py_am_send` *is part of the Stable ABI since version 3.10.*

La firma de esta función es:

```
PySendResult am_send(PyObject *self, PyObject *arg,
```

Consulte **`PyIter_Send()`** para obtener más detalles. Esta ranura se puede establecer en `NULL`.

Added in version 3.10.

Tipo Ranura *typedefs*

typedef PyObject *(*allocfunc)(PyTypeObject *cls, Py_ssize_t nitems)

Part of the Stable ABI.
The purpose of this function is to separate memory allocation from memory initialization. It should return a pointer to a block of memory of adequate length for the instance, suitably aligned, and initialized to zeros, but with **`ob_refcnt`** set to `1` and **`ob_type`** set to the type argument. If the type's **`tp_itemsize`** is non-zero, the object's **`ob_size`** field should be initialized to *nitems* and the length of the allocated memory block should be `tp_basicsize + nitems*tp_itemsize`, rounded up to a multiple of `sizeof(void*)`; otherwise, *nitems* is not used and the length of the block should be **`tp_basicsize`**.

Esta función no debe hacer ninguna otra instancia de inicialización, ni siquiera para asignar memoria adicional; eso debe ser realizado por **`tp_new`**.

typedef void (*destructor)(PyObject*)

Part of the Stable ABI.

typedef void (*freefunc)(void*)

Consulte **`tp_free`**.

typedef PyObject *(*newfunc)(PyTypeObject*, PyObject*, PyObject*)

Part of the Stable ABI.
Consulte **`tp_new`**.

typedef int (*initproc)(PyObject*, PyObject*, PyObject*)

Part of the Stable ABI.
Consulte **`tp_init`**.

typedef PyObject *(*reprfunc)(PyObject*)

Part of the Stable ABI.
Consulte **`tp_repr`**.

typedef PyObject *(*getattrfunc)(PyObject *self, char *attr)

Part of the Stable ABI.
Retorna el valor del atributo nombrado para el objeto.

typedef int (*setattrfunc)(PyObject *self, char *attr, PyObject *value)

Part of the Stable ABI.
Establece el valor del atributo nombrado para el objeto. El argumento del valor se establece en `NULL` para eliminar el atributo.

typedef PyObject *(*getattrofunc)(PyObject *self, PyObject *attr)

Part of the Stable ABI.
Retorna el valor del atributo nombrado para el objeto.

Consulte **`tp_getattro`**.

typedef int (*setattrofunc)(PyObject *self, PyObject *attr, PyObject *value)

Part of the Stable ABI.
Establece el valor del atributo nombrado para el objeto. El

argumento del valor se establece en `NULL` para eliminar el atributo.

Consulte **`tp_setattro`**.

typedef PyObject *(*descrgetfunc)(PyObject*, PyObject*, PyObject*)

Part of the Stable ABI.
See **`tp_descr_get`**.

typedef int (*descrsetfunc)(PyObject*, PyObject*, PyObject*)

Part of the Stable ABI.
See **`tp_descr_set`**.

typedef Py_hash_t (*hashfunc)(PyObject*)

Part of the Stable ABI.
Consulte **`tp_hash`**.

typedef PyObject *(*richcmpfunc)(PyObject*, PyObject*, int)

Part of the Stable ABI.
Consulte **`tp_richcompare`**.

typedef PyObject *(*getiterfunc)(PyObject*)

Part of the Stable ABI.
Consulte **`tp_iter`**.

typedef PyObject *(*iternextfunc)(PyObject*)

Part of the Stable ABI.
Consulte **`tp_iternext`**.

typedef Py_ssize_t (*lenfunc)(PyObject*)

Part of the Stable ABI.

typedef int (*getbufferproc)(PyObject*, Py_buffer*, int)

Part of the Stable ABI since version 3.12.

typedef void (*releasebufferproc)(PyObject*, Py_buffer*)

Part of the Stable ABI since version 3.12.

typedef PyObject *(*unaryfunc)(PyObject*)

Part of the Stable ABI.

typedef PyObject *(*binaryfunc)(PyObject*, PyObject*)

Part of the Stable ABI.

typedef PySendResult (*sendfunc)(PyObject*, PyObject*, PyObject**)

Consulte `am_send`.

typedef PyObject *(*ternaryfunc)(PyObject*, PyObject*, PyObject*)

Part of the Stable ABI.

typedef PyObject *(*ssizeargfunc)(PyObject*, Py_ssize_t)

Part of the Stable ABI.

typedef int (*ssizeobjargproc)(PyObject*, Py_ssize_t, PyObject*)

Part of the Stable ABI.

typedef int (*objobjproc)(PyObject*, PyObject*)

Part of the Stable ABI.

typedef int (*objobjargproc)(PyObject*, PyObject*, PyObject*)

Part of the Stable ABI.

Ejemplos

Los siguientes son ejemplos simples de definiciones de tipo Python. Incluyen el uso común que puede encontrar. Algunos demuestran casos difíciles de esquina (*corner cases*). Para obtener más ejemplos, información práctica y un tutorial, consulte «definiendo nuevos tipos» (Definición de tipos de extensión: Tutorial) y «tópicos de nuevos tipos (Definición de tipos de extensión: temas variados).

Un tipo estático básico:

```
typedef struct {
    PyObject_HEAD
    const char *data;
} MyObject;

static PyTypeObject MyObject_Type = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "mymod.MyObject",
    .tp_basicsize = sizeof(MyObject),
    .tp_doc = PyDoc_STR("My objects"),
    .tp_new = myobj_new,
    .tp_dealloc = (destructor)myobj_dealloc,
    .tp_repr = (reprfunc)myobj_repr,
};
```

También puede encontrar código más antiguo (especialmente en la base de código CPython) con un inicializador más detallado:

```
static PyTypeObject MyObject_Type = {
    PyVarObject_HEAD_INIT(NULL, 0)
    "mymod.MyObject",               /* tp_name */
    sizeof(MyObject),               /* tp_basicsize */
    0,                              /* tp_itemsize */
    (destructor)myobj_dealloc,      /* tp_dealloc */
    0,                              /* tp_vectorcall_off
    0,                              /* tp_getattr */
    0,                              /* tp_setattr */
    0,                              /* tp_as_async */
    (reprfunc)myobj_repr,           /* tp_repr */
    0,                              /* tp_as_number */
    0,                              /* tp_as_sequence */
    0,                              /* tp_as_mapping */
    0,                              /* tp_hash */
    0,                              /* tp_call */
    0,                              /* tp_str */
    0,                              /* tp_getattro */
    0,                              /* tp_setattro */
    0,                              /* tp_as_buffer */
    0,                              /* tp_flags */
    PyDoc_STR("My objects"),        /* tp_doc */
    0,                              /* tp_traverse */
    0,                              /* tp_clear */
    0,                              /* tp_richcompare */
    0,                              /* tp_weaklistoffset
    0,                              /* tp_iter */
    0,                              /* tp_iternext */
    0,                              /* tp_methods */
    0,                              /* tp_members */
    0,                              /* tp_getset */
    0,                              /* tp_base */
    0,                              /* tp_dict */
    0,                              /* tp_descr_get */
    0,                              /* tp_descr_set */
    0,                              /* tp_dictoffset */
    0,                              /* tp_init */
    0,                              /* tp_alloc */
    myobj_new,                      /* tp_new */
};
```

Un tipo que admite referencias débiles, instancias de diccionarios (*dicts*) y *hashing*:

```
typedef struct {
    PyObject_HEAD
    const char *data;
} MyObject;

static PyTypeObject MyObject_Type = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "mymod.MyObject",
    .tp_basicsize = sizeof(MyObject),
    .tp_doc = PyDoc_STR("My objects"),
    .tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE
         Py_TPFLAGS_HAVE_GC | Py_TPFLAGS_MANAGED_DICT |
         Py_TPFLAGS_MANAGED_WEAKREF,
    .tp_new = myobj_new,
    .tp_traverse = (traverseproc)myobj_traverse,
    .tp_clear = (inquiry)myobj_clear,
    .tp_alloc = PyType_GenericNew,
    .tp_dealloc = (destructor)myobj_dealloc,
    .tp_repr = (reprfunc)myobj_repr,
    .tp_hash = (hashfunc)myobj_hash,
    .tp_richcompare = PyBaseObject_Type.tp_richcompare,
};
```

A str subclass that cannot be subclassed and cannot be called to create instances (e.g. uses a separate factory func) using **`Py_TPFLAGS_DISALLOW_INSTANTIATION`** flag:

```
typedef struct {
    PyUnicodeObject raw;
    char *extra;
} MyStr;

static PyTypeObject MyStr_Type = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "mymod.MyStr",
    .tp_basicsize = sizeof(MyStr),
    .tp_base = NULL,  // set to &PyUnicode_Type in modul
    .tp_doc = PyDoc_STR("my custom str"),
    .tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_DISALLOW
    .tp_repr = (reprfunc)myobj_repr,
};
```

El tipo estático más simple con instancias de longitud fija:

```
typedef struct {
    PyObject_HEAD
} MyObject;

static PyTypeObject MyObject_Type = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "mymod.MyObject",
};
```

El tipo estático más simple con instancias de longitud variable:

```
typedef struct {
    PyObject_VAR_HEAD
    const char *data[1];
} MyObject;

static PyTypeObject MyObject_Type = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "mymod.MyObject",
    .tp_basicsize = sizeof(MyObject) - sizeof(char *),
    .tp_itemsize = sizeof(char *),
};
```

Apoyo a la recolección de basura cíclica

El soporte de Python para detectar y recolectar basura que involucra referencias circulares requiere el soporte de tipos de objetos que son «contenedores» para otros objetos que también pueden ser contenedores. Los tipos que no almacenan referencias a otros objetos, o que solo almacenan referencias a tipos atómicos (como números o cadenas), no necesitan proporcionar ningún soporte explícito para la recolección de basura.

Para crear un tipo de contenedor, el campo `tp_flags` del objeto tipo debe incluir `Py_TPFLAGS_HAVE_GC` y proporcionar una implementación del manejador `tp_traverse`. Si las instancias del tipo son mutables, también se debe proporcionar una implementación a `tp_clear`.

`Py_TPFLAGS_HAVE_GC`

Los objetos con un tipo con este indicador establecido deben cumplir con las reglas documentadas aquí. Por conveniencia, estos objetos se denominarán objetos contenedor.

Los constructores para tipos de contenedores deben cumplir con dos reglas:

1. La memoria para el objeto debe asignarse usando `PyObject_GC_New` o `PyObject_GC_NewVar`.
2. Una vez que se inicializan todos los campos que pueden contener referencias a otros contenedores, debe llamar a `PyObject_GC_Track()`.

Del mismo modo, el desasignador (*deallocator*) para el objeto debe cumplir con un par similar de reglas:

1. Antes de invalidar los campos que se refieren a otros contenedores, debe llamarse `PyObject_GC_UnTrack()`.

2. La memoria del objeto debe ser desasignada (*deallocated*) usando `PyObject_GC_Del()`.

Advertencia

Si un tipo añade el Py_TPFLAGS_HAVE_GC, entonces *must* implementar al menos un manejado `tp_traverse` o usar explícitamente uno de su subclase o subclases.

Al llamar a `PyType_Ready()` o alguna de las APIs que indirectamente lo llaman como `PyType_FromSpecWithBases()` o `PyType_FromSpec()` el intérprete automáticamente llenara los campos `tp_flags`, `tp_traverse` y `tp_clear` si el tipo hereda de una clase que implementa el protocolo del recolector de basura y la clase secundaria *no* incluye el *flag* `Py_TPFLAGS_HAVE_GC`.

PyObject_GC_New(TYPE, typeobj)

Análogo a `PyObject_New` pero para objetos de contenedor con el *flag* `Py_TPFLAGS_HAVE_GC` establecido.

Do not call this directly to allocate memory for an object; call the type's `tp_alloc` slot instead.

When populating a type's `tp_alloc` slot, `PyType_GenericAlloc()` is preferred over a custom function that simply calls this macro.

Memory allocated by this macro must be freed with `PyObject_GC_Del()` (usually called via the object's `tp_free` slot).

Ver también

- `PyObject_GC_Del()`
- `PyObject_New`
- `PyType_GenericAlloc()`
- `tp_alloc`

PyObject_GC_NewVar(TYPE, typeobj, size)

Análogo a `PyObject_NewVar` pero para objetos de contenedor con el *flag* `Py_TPFLAGS_HAVE_GC` establecido.

Do not call this directly to allocate memory for an object; call the type's `tp_alloc` slot instead.

When populating a type's `tp_alloc` slot, `PyType_GenericAlloc()` is preferred over a custom function that simply calls this macro.

Memory allocated by this macro must be freed with `PyObject_GC_Del()` (usually called via the object's `tp_free` slot).

Ver también

- `PyObject_GC_Del()`
- `PyObject_NewVar`
- `PyType_GenericAlloc()`
- `tp_alloc`

PyObject *PyUnstable_Object_GC_NewWithExtraData(PyTypeObject *type, size_t extra_size)

This is Unstable API. It may change without warning in minor releases.

Análogo a `PyObject_GC_New` pero asigna *extra_size* bytes al final del objeto (en el desplazamiento `tp_basicsize`). La memoria asignada se inicializa a ceros, excepto para el `encabezado del objeto Python`.

Los datos extras se desasignarán con el objeto, pero por lo demás no se gestionan por Python.

Memory allocated by this function must be freed with **`PyObject_GC_Del()`** (usually called via the object's **`tp_free`** slot).

Advertencia

La función está marcada como inestable porque aún no se ha decidido el mecanismo final para reservar datos extra después de una instancia. Para asignar un número variable de campos, se recomienda usar en su lugar **`PyVarObject`** y **`tp_itemsize`**.

Added in version 3.12.

PyObject_GC_Resize(TYPE, op, newsize)

Cambia el tamaño de un objeto asignado por **`PyObject_NewVar`**. Retorna el objeto redimensionado de tipo `TYPE*` (se refiere a cualquier tipo de C) o `NULL` en caso de falla.

op debe ser de tipo PyVarObject* y aún no debe ser rastreado por el recolector. *newsize* debe ser de tipo **`Py_ssize_t`**.

void PyObject_GC_Track(PyObject *op)

Part of the Stable ABI.
Agrega el objeto *op* al conjunto de objetos contenedor seguidos por el recolector de basura. El recolector puede ejecutarse en momentos inesperados, por lo que los objetos deben ser válidos durante el seguimiento. Esto debería llamarse una vez que todos los campos seguidos por **`tp_traverse`** se vuelven válidos, generalmente cerca del final del constructor.

int PyObject_IS_GC(PyObject *obj)

Retorna un valor distinto de cero si el objeto implementa el protocolo del recolector de basura; de lo contrario, retorna 0.

El recolector de basura no puede rastrear el objeto si esta función retorna 0.

int PyObject_GC_IsTracked(PyObject *op)

Part of the Stable ABI since version 3.9.
Retorna 1 si el tipo de objeto de *op* implementa el protocolo GC y el recolector de basura está rastreando *op* y 0 en caso contrario.

Esto es análogo a la función de Python `gc.is_tracked()`.

Added in version 3.9.

int PyObject_GC_IsFinalized(PyObject *op)

Part of the Stable ABI since version 3.9.
Retorna 1 si el tipo de objeto de *op* implementa el protocolo GC y *op* ya ha sido finalizado por el recolector de basura y 0 en caso contrario.

Esto es análogo a la función de Python `gc.is_finalized()`.

Added in version 3.9.

void PyObject_GC_Del(void *op)

Part of the Stable ABI.
Libera memoria asignada a un objeto usando `PyObject_GC_New` o `PyObject_GC_NewVar`.

Do not call this directly to free an object's memory; call the type's `tp_free` slot instead.

Do not use this for memory allocated by `PyObject_New`, `PyObject_NewVar`, or related allocation functions; use `PyObject_Free()` instead.

Ver también

- `PyObject_Free()` is the non-GC equivalent of this function.

- `PyObject_GC_New`
- `PyObject_GC_NewVar`
- `PyType_GenericAlloc()`
- `tp_free`

void PyObject_GC_UnTrack(void *op)

Part of the Stable ABI.
Elimina el objeto *op* del conjunto de objetos contenedor rastreados por el recolector de basura. Tenga en cuenta que `PyObject_GC_Track()` puede ser llamado nuevamente en este objeto para agregarlo nuevamente al conjunto de objetos rastreados. El desasignador (el manejador `tp_dealloc`) debería llamarlo para el objeto antes de que cualquiera de los campos utilizados por el manejador `tp_traverse` no sea válido.

Distinto en la versión 3.8: Los macros `_PyObject_GC_TRACK()` y `_PyObject_GC_UNTRACK()` se han eliminado de la API pública de C.

El manejador `tp_traverse` acepta un parámetro de función de este tipo:

typedef int (*visitproc)(PyObject *object, void *arg)

Part of the Stable ABI.
Tipo de la función visitante que se pasa al manejador `tp_traverse`. La función debe llamarse con un objeto para atravesar como *object* y el tercer parámetro para el manejador `tp_traverse` como *arg*. El núcleo de Python utiliza varias funciones visitantes para implementar la detección de basura cíclica; No se espera que los usuarios necesiten escribir sus propias funciones visitante.

El manejador `tp_traverse` debe tener el siguiente tipo:

typedef int (*traverseproc)(PyObject *self, visitproc visit, void *arg)

Part of the Stable ABI.
Función transversal para un objeto contenedor. Las

implementaciones deben llamar a la función *visit* para cada objeto directamente contenido por *self*, siendo los parámetros a *visit* el objeto contenido y el valor *arg* pasado al controlador. La función *visit* no debe llamarse con un argumento de objeto `NULL`. Si *visit* retorna un valor distinto de cero, ese valor debe retornarse inmediatamente.

The traversal function must not have any side effects. Implementations may not modify the reference counts of any Python objects nor create or destroy any Python objects.

Para simplificar la escritura de los manejadores **`tp_traverse`**, se proporciona un macro a **`Py_VISIT()`**. Para usar este macro, la implementación **`tp_traverse`** debe nombrar sus argumentos exactamente *visit* y *arg*:

Py_VISIT(o)

If the PyObject* *o* is not `NULL`, call the *visit* callback, with arguments *o* and *arg*. If *visit* returns a non-zero value, then return it. Using this macro, **`tp_traverse`** handlers look like:

```
static int
my_traverse(Noddy *self, visitproc visit, void *arg
{
    Py_VISIT(self->foo);
    Py_VISIT(self->bar);
    return 0;
}
```

El manejador **`tp_clear`** debe ser del tipo **`query`**, o `NULL` si el objeto es inmutable.

typedef int (*inquiry)(PyObject *self)

Part of the Stable ABI.
Descarta referencias que pueden haber creado ciclos de referencia. Los objetos inmutables no tienen que definir este método ya que nunca pueden crear directamente ciclos de referencia. Tenga en cuenta que el objeto aún debe ser válido

después de llamar a este método (no solo llame a `Py_DECREF()` en una referencia). El recolector de basura llamará a este método si detecta que este objeto está involucrado en un ciclo de referencia.

Controlar el estado del recolector de basura

La C-API proporciona las siguientes funciones para controlar las ejecuciones de recolección de basura.

Py_ssize_t PyGC_Collect(void)

Part of the Stable ABI.
Realiza una recolección de basura completa, si el recolector de basura está habilitado. (Tenga en cuenta que `gc.collect()` lo ejecuta incondicionalmente).

Retorna el número de objetos recolectados e inalcanzables que no se pueden recolectar. Si el recolector de basura está deshabilitado o ya está recolectando, retorna `0` inmediatamente. Los errores durante la recolección de basura se pasan a `sys.unraisablehook`. Esta función no genera excepciones.

int PyGC_Enable(void)

Part of the Stable ABI since version 3.10.
Habilita el recolector de basura: similar a `gc.enable()`. Retorna el estado anterior, 0 para deshabilitado y 1 para habilitado.

Added in version 3.10.

int PyGC_Disable(void)

Part of the Stable ABI since version 3.10.
Deshabilita el recolector de basura: similar a `gc.disable()`. Retorna el estado anterior, 0 para deshabilitado y 1 para habilitado.

Added in version 3.10.

int PyGC_IsEnabled(void)

Part of the Stable ABI since version 3.10.
Consulta el estado del recolector de basura: similar a `gc.isenabled()`. Retorna el estado actual, 0 para deshabilitado y 1 para habilitado.

Added in version 3.10.

Consultar el estado del recolector de basura

La C-API proporciona la siguiente interfaz para consultar información sobre el recolector de basura.

void PyUnstable_GC_VisitObjects(gcvisitobjects_t callback, void *arg)

This is Unstable API. It may change without warning in minor releases.

Ejecuta la *callback* suministrada en todos los objetos activos con capacidad para GC. *arg* se pasa a todas las invocaciones de *callback*.

Advertencia

Si la devolución de llamada (des)asigna nuevos objetos, no está definido si serán visitados.

La recolección de basura está deshabilitada durante la operación. Ejecutar una recolección explícitamente en la devolución de llamada puede provocar un comportamiento indefinido, por ejemplo, visitar los mismos objetos varias veces o no visitarlos en absoluto.

Added in version 3.12.

typedef int (*gcvisitobjects_t)(PyObject *object, void *arg)

Type of the visitor function to be passed to **`PyUnstable_GC_VisitObjects()`**. *arg* is the same as the *arg* passed to `PyUnstable_GC_VisitObjects`. Return `1` to continue iteration, return `0` to stop iteration. Other return values are reserved for now so behavior on returning anything else is undefined.

Added in version 3.12.

Versiones de API y ABI

Build-time version constants

CPython exposes its version number in the following macros. Note that these correspond to the version code is **built** with. See **`Py_Version`** for the version used at **run time**.

Consulte Estabilidad de la API en C para obtener una discusión sobre la estabilidad de API y ABI en todas las versiones.

PY_MAJOR_VERSION

El `3` en `3.4.1a2`.

PY_MINOR_VERSION

El `4` en `3.4.1a2`.

PY_MICRO_VERSION

El `1` en `3.4.1a2`.

PY_RELEASE_LEVEL

La `a` en `3.4.1a2`. Puede ser `0xA` para la versión alfa, `0xB` para la versión beta, `0xC` para la versión candidata o `0xF` para la versión final.

PY_RELEASE_SERIAL

El `2` en `3.4.1a2`, cero para lanzamientos finales.

PY_VERSION_HEX

The Python version number encoded in a single integer. See **`Py_PACK_FULL_VERSION()`** for the encoding details.

Use this for numeric comparisons, for example, `#if PY_VERSION_HEX >= ...`.

These macros are defined in Include/patchlevel.h [https://github.com/python/cpython/tree/3.14/Include/patchlevel.h].

Run-time version

const unsigned long Py_Version

Part of the Stable ABI since version 3.11.
The Python runtime version number encoded in a single constant integer. See `Py_PACK_FULL_VERSION()` for the encoding details. This contains the Python version used at run time.

Use this for numeric comparisons, for example, `if (Py_Version >= ...)`.

Added in version 3.11.

Bit-packing macros

uint32_t Py_PACK_FULL_VERSION(int major, int minor, int micro, int release_level, int release_serial)

Part of the Stable ABI since version 3.14.
Return the given version, encoded as a single 32-bit integer with the following structure:

Bits	Bit mask	Field	Example values	
			`3.4.1a2`	**`3.10.0`**
24-31	`0xFF000000`	*major*	3	3
16-23	`0x00FF0000`	*minor*	4	10
8-15	`0x0000FF00`	*micro*	1	0
4-7	`0x000000F0`	*release_level*	`0xA`	`0xF`
0-3	`0x0000000F`	*release_serial*	2	0

For example:

Version	`Py_PACK_FULL_VERSION` arguments	Encoded version
3.4.1a2	`(3, 4, 1, 0xA, 2)`	`0x030401a2`
3.10.0	`(3, 10, 0, 0xF, 0)`	`0x030a00f0`

Out-of range bits in the arguments are ignored. That is, the macro can be defined as:

```
#ifndef Py_PACK_FULL_VERSION
#define Py_PACK_FULL_VERSION(X, Y, Z, LEVEL, SERIAL
   (((X) & 0xff) << 24) |
   (((Y) & 0xff) << 16) |
   (((Z) & 0xff) << 8) |
   (((LEVEL) & 0xf) << 4) |
   (((SERIAL) & 0xf) << 0))
#endif
```

`Py_PACK_FULL_VERSION` is primarily a macro, intended for use in `#if` directives, but it is also available as an exported function.

Added in version 3.14.

uint32_t Py_PACK_VERSION(int major, int minor)

Part of the Stable ABI since version 3.14.
Equivalent to `Py_PACK_FULL_VERSION(major, minor, 0, 0, 0)`. The result does not correspond to any Python release, but is useful in numeric comparisons.

Added in version 3.14.

Monitoring C API

Added in version 3.13.

An extension may need to interact with the event monitoring system. Subscribing to events and registering callbacks can be done via the Python API exposed in **`sys.monitoring`**.

Generating Execution Events

The functions below make it possible for an extension to fire monitoring events as it emulates the execution of Python code. Each of these functions accepts a `PyMonitoringState` struct which contains concise information about the activation state of events, as well as the event arguments, which include a `PyObject*` representing the code object, the instruction offset and sometimes additional, event-specific arguments (see **`sys.monitoring`** for details about the signatures of the different event callbacks). The `codelike` argument should be an instance of **`types.CodeType`** or of a type that emulates it.

The VM disables tracing when firing an event, so there is no need for user code to do that.

Monitoring functions should not be called with an exception set, except those listed below as working with the current exception.

type PyMonitoringState

> Representation of the state of an event type. It is allocated by the user while its contents are maintained by the monitoring API functions described below.

All of the functions below return 0 on success and -1 (with an exception set) on error.

See `sys.monitoring` for descriptions of the events.

int PyMonitoring_FirePyStartEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset)

Fire a `PY_START` event.

int PyMonitoring_FirePyResumeEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset)

Fire a `PY_RESUME` event.

int PyMonitoring_FirePyReturnEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *retval)

Fire a `PY_RETURN` event.

int PyMonitoring_FirePyYieldEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *retval)

Fire a `PY_YIELD` event.

int PyMonitoring_FireCallEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *callable, PyObject *arg0)

Fire a `CALL` event.

int PyMonitoring_FireLineEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset, int lineno)

Fire a `LINE` event.

int PyMonitoring_FireJumpEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *target_offset)

Fire a `JUMP` event.

int PyMonitoring_FireBranchLeftEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *target_offset)

Fire a `BRANCH_LEFT` event.

int PyMonitoring_FireBranchRightEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *target_offset)

Fire a `BRANCH_RIGHT` event.

int PyMonitoring_FireCReturnEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *retval)

Fire a `C_RETURN` event.

int PyMonitoring_FirePyThrowEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset)

Fire a `PY_THROW` event with the current exception (as returned by **`PyErr_GetRaisedException()`**).

int PyMonitoring_FireRaiseEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset)

Fire a `RAISE` event with the current exception (as returned by **`PyErr_GetRaisedException()`**).

int PyMonitoring_FireCRaiseEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset)

Fire a `C_RAISE` event with the current exception (as returned by **`PyErr_GetRaisedException()`**).

int PyMonitoring_FireReraiseEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset)

Fire a `RERAISE` event with the current exception (as returned by **`PyErr_GetRaisedException()`**).

int PyMonitoring_FireExceptionHandledEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset)

Fire an `EXCEPTION_HANDLED` event with the current exception (as returned by **`PyErr_GetRaisedException()`**).

int PyMonitoring_FirePyUnwindEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset)

Fire a `PY_UNWIND` event with the current exception (as returned by **`PyErr_GetRaisedException()`**).

int PyMonitoring_FireStopIterationEvent(PyMonitoringState *state, PyObject *codelike, int32_t offset, PyObject *value)

Fire a `STOP_ITERATION` event. If `value` is an instance of **`StopIteration`**, it is used. Otherwise, a new **`StopIteration`** instance is created with `value` as its argument.

Managing the Monitoring State

Monitoring states can be managed with the help of monitoring scopes. A scope would typically correspond to a Python function.

int PyMonitoring_EnterScope(PyMonitoringState *state_array, uint64_t *version, const uint8_t *event_types, Py_ssize_t length)

Enter a monitored scope. `event_types` is an array of the event IDs for events that may be fired from the scope. For example, the ID of a `PY_START` event is the value `PY_MONITORING_EVENT_PY_START`, which is numerically equal to the base-2 logarithm of `sys.monitoring.events.PY_START`. `state_array` is an array with a monitoring state entry for each event in `event_types`, it is allocated by the user but populated by **`PyMonitoring_EnterScope()`** with information about the activation state of the event. The size of `event_types` (and hence also of `state_array`) is given in `length`.

The `version` argument is a pointer to a value which should be allocated by the user together with `state_array` and initialized to 0, and then set only by **`PyMonitoring_EnterScope()`** itself. It allows this function to determine whether event states have changed since the previous call, and to return quickly if they have not.

The scopes referred to here are lexical scopes: a function, class or method. **`PyMonitoring_EnterScope()`** should be called whenever the lexical scope is entered. Scopes can be reentered, reusing the same *state_array* and *version*, in situations like when emulating a recursive Python function. When a code-like's execution is paused, such as when emulating a generator, the scope needs to be exited and re-entered.

The macros for *event_types* are:

Macro	Event
PY_MONITORING_EVENT_BRANCH_LEFT	**`BRANCH_LEFT`**
PY_MONITORING_EVENT_BRANCH_RIGHT	**`BRANCH_RIGHT`**
PY_MONITORING_EVENT_CALL	**`CALL`**
PY_MONITORING_EVENT_C_RAISE	**`C_RAISE`**
PY_MONITORING_EVENT_C_RETURN	**`C_RETURN`**
PY_MONITORING_EVENT_EXCEPTION_HANDLED	**`EXCEPTION_HANDLED`**
PY_MONITORING_EVENT_INSTRUCTION	**`INSTRUCTION`**
PY_MONITORING_EVENT_JUMP	**`JUMP`**
PY_MONITORING_EVENT_LINE	**`LINE`**
PY_MONITORING_EVENT_PY_RESUME	**`PY_RESUME`**
PY_MONITORING_EVENT_PY_RETURN	**`PY_RETURN`**
PY_MONITORING_EVENT_PY_START	**`PY_START`**
PY_MONITORING_EVENT_PY_THROW	**`PY_THROW`**
PY_MONITORING_EVENT_PY_UNWIND	**`PY_UNWIND`**
PY_MONITORING_EVENT_PY_YIELD	**`PY_YIELD`**
PY_MONITORING_EVENT_RAISE	**`RAISE`**
PY_MONITORING_EVENT_RERAISE	**`RERAISE`**
PY_MONITORING_EVENT_STOP_ITERATION	**`STOP_ITERATION`**

int PyMonitoring_ExitScope(void)

Exit the last scope that was entered with **`PyMonitoring_EnterScope()`**.

int PY_MONITORING_IS_INSTRUMENTED_EVENT(uint8_t ev)

Return true if the event corresponding to the event ID *ev* is a local event.

Added in version 3.13.

Soft deprecated since version 3.14.

Installing Python modules

As a popular open source development project, Python has an active supporting community of contributors and users that also make their software available for other Python developers to use under open-source license terms.

Esto permite a los usuarios de Python compartir y colaborar de manera efectiva, beneficiándose de las soluciones que otros ya han creado para problemas comunes (¡y a veces incluso raros!), además de contribuir potencialmente con sus propias soluciones al grupo común.

This guide covers the installation part of the process. For a guide to creating and sharing your own Python projects, refer to the Python packaging user guide [https://packaging.python.org/en/latest/tutorials/packaging-projects/].

Nota

Para los usuarios corporativos y otros usuarios institucionales, se debe tener en cuenta que muchas organizaciones tienen sus propias políticas sobre el uso y la contribución al software de código abierto. Se deben tener en cuenta dichas políticas al utilizar las herramientas de distribución e instalación proporcionadas con Python.

Palabras clave

- **pip** is the preferred installer program. It is included by default with the Python binary installers.
- Un *entorno virtual* es un entorno de Python parcialmente aislado que permite instalar paquetes para que los use una aplicación en particular, en lugar de instalarlos en todo el sistema.

- `venv` is the standard tool for creating virtual environments. It defaults to installing **pip** into all created virtual environments.
- `virtualenv` is a third-party alternative (and predecessor) to `venv`.
- The Python Package Index (PyPI) [https://pypi.org] is a public repository of open source licensed packages made available for use by other Python users.
- The Python Packaging Authority [https://www.pypa.io/] is the group of developers and documentation authors responsible for the maintenance and evolution of the standard packaging tools and the associated metadata and file format standards. They maintain a variety of tools, documentation, and issue trackers on GitHub [https://github.com/pypa].

Distinto en la versión 3.5: Ahora se recomienda el uso de `venv` para crear entornos virtuales.

Ver también

Guía de usuario de empaquetado de Python: Crear y usar entornos virtuales [https://packaging.python.org/installing/#creating-virtual-environments]

Uso básico

Las herramientas estándar de empaquetado están diseñadas para que se usen desde la línea de comandos.

The following command will install the latest version of a module and its dependencies from PyPI:

```
python -m pip install SomePackage
```

Nota

Para usuarios POSIX (incluyendo los usuarios de macOS y Linux), los ejemplos en esta guía asumen que se está usando un virtual environment.

Para los usuarios de Windows, los ejemplos en esta guía asumen que se seleccionó la opción de ajustar la variable de entorno PATH del sistema al instalar Python.

Es posible especificar una versión exacta o mínima directamente en la linea de comandos. Cuando se use un operando comparador como `>`, `<` o cualquier otro carácter especial que puede ser interpretado por el intérprete de comandos, el nombre del paquete y la versión deben ir entre comillas dobles:

```
python -m pip install SomePackage==1.0.4    # specific v
python -m pip install "SomePackage>=1.0.4"  # minimum ve
```

Normalmente, si ya hay instalado un módulo adecuado, intentar instalarlo otra vez no tendrá efecto alguno. Actualizar módulos existentes requiere que se solicite explícitamente:

```
python -m pip install --upgrade SomePackage
```

More information and resources regarding **pip** and its capabilities can be found in the Python Packaging User Guide [https://packaging.python.org].

La creación de entornos virtuales se realiza a través de el módulo **`venv`**. Instalar paquetes en un entorno virtual activo usa los comandos mostrados arriba.

Ver también

Guía de usuario de empaquetado de Python: Instalando paquetes de distribución de Python [https://packaging.python.org/installing/]

¿Cómo…

Respuestas rápidas o enlaces para algunas tareas comunes.

… instalo paquetes solamente para el usuario actual?

Pasando la opción `--user` a `python -m pip install` instalará el paquete únicamente para el usuario actual, en lugar de hacerlo para todos los usuarios del sistema.

… instalo paquetes científicos de Python?

A number of scientific Python packages have complex binary dependencies, and aren't currently easy to install using **pip** directly. It will often be easier for users to install these packages by other means [https://packaging.python.org/science/] rather than attempting to install them with **pip**.

Ver también

Guía de usuario de empaquetado de Python: Instalando paquetes científicos [https://packaging.python.org/science/]

… trabajo con múltiples versiones de Python instaladas en paralelo?

On Linux, macOS, and other POSIX systems, use the versioned Python commands in combination with the `-m` switch to run the appropriate copy of **pip**:

```
python3    -m pip install SomePackage  # default Python
python3.14 -m pip install SomePackage  # specifically Py
```

Appropriately versioned **pip** commands may also be available.

On Windows, use the **py** Python launcher in combination with the `-m` switch:

```
py -3    -m pip install SomePackage  # default Python 3
py -3.14 -m pip install SomePackage  # specifically Pyth
```

Problemas de instalación comunes

Instalando en el Python del sistema bajo Linux

On Linux systems, a Python installation will typically be included as

part of the distribution. Installing into this Python installation requires root access to the system, and may interfere with the operation of the system package manager and other components of the system if a component is unexpectedly upgraded using **pip**.

On such systems, it is often better to use a virtual environment or a per-user installation when installing packages with **pip**.

Pip no está instalado

It is possible that **pip** does not get installed by default. One potential fix is:

```
python -m ensurepip --default-pip
```

There are also additional resources for installing pip [https://packaging.python.org/en/latest/tutorials/installing-packages/#ensure-pip-setuptools-and-wheel-are-up-to-date].

Instalando extensiones binarias

Python once relied heavily on source-based distribution, with end users being expected to compile extension modules from source as part of the installation process.

With the introduction of the binary wheel format, and the ability to publish wheels through PyPI, this problem is diminishing, as users are more regularly able to install pre-built extensions rather than needing to build them themselves.

Some of the solutions for installing scientific software [https://packaging.python.org/science/] that are not yet available as pre-built wheel files may also help with obtaining other binary extensions without needing to build them locally.

Ver también

Guía de usuario de empaquetado de Python: Extensiones binarias [https://packaging.python.org/extensions/]

Comos (*HOWTOs*) de Python

Los Comos (*HOWTOs*) de Python son documentos que cubren un sólo tema específico e intentan cubrirlo de manera bastante completa. Siguiendo el modelo de la colección HOWTO del Proyecto de documentación de Linux, esta colección es un esfuerzo para fomentar la documentación que es más detallada que la Referencia de la biblioteca de Python.

General:

- A Conceptual Overview of asyncio
- Prácticas recomendadas para las anotaciones
- Tutorial de argparse
- Descriptor Guide
- HOWTO - Enum
- HOWTO - Programación funcional
- Introducción al modulo ipaddress
- HOWTO hacer registros (Logging)
- Libro de recetas de Logging
- Regular expression HOWTO
- Sorting Techniques
- CÓMO (HOWTO) Unicode
- HOWTO - Cómo obtener recursos de Internet con el paquete urllib

Desarrollo avanzado:

- Programación de curses con Python
- Python support for free threading
- C API Extension Support for Free Threading
- Aislamiento de módulos de extensión
- The Python 2.3 Method Resolution Order
- HOW TO - Programación con sockets
- timer file descriptor HOWTO
- Portar módulos de extensión a Python 3

Depuración y Perfilado:

- Debugging C API extensions and CPython Internals with GDB
- Instrumentación de CPython con DTrace y SystemTap
- Soporte de Python para el perfilador perf de Linux
- Remote debugging attachment protocol

A Conceptual Overview of `asyncio`

This HOWTO article seeks to help you build a sturdy mental model of how `asyncio` fundamentally works, helping you understand the how and why behind the recommended patterns.

You might be curious about some key `asyncio` concepts. By the end of this article, you'll be able to comfortably answer these questions:

- What's happening behind the scenes when an object is awaited?
- How does `asyncio` differentiate between a task which doesn't need CPU time (such as a network request or file read) as opposed to a task that does (such as computing n-factorial)?
- How to write an asynchronous variant of an operation, such as an async sleep or database request.

Ver también

- The guide [https://github.com/anordin95/a-conceptual-overview-of-asyncio/tree/main] that inspired this HOWTO article, by Alexander Nordin.
- This in-depth YouTube tutorial series [https://www.youtube.com/watch?v=Xbl7XjFYsN4&list=PLhNSoGM2ik6SIkVGXWBwerucXjgP1rHmB] on `asyncio` created by Python core team member, Łukasz Langa.
- 500 Lines or Less: A Web Crawler With asyncio Coroutines [https://aosabook.org/en/500L/a-web-crawler-with-asyncio-coroutines.html] by A. Jesse Jiryu Davis and Guido van Rossum.

A conceptual overview part 1: the high-level

In part 1, we'll cover the main, high-level building blocks of **`asyncio`**: the event loop, coroutine functions, coroutine objects, tasks, and `await`.

Event Loop

Everything in **`asyncio`** happens relative to the event loop. It's the star of the show. It's like an orchestra conductor. It's behind the scenes managing resources. Some power is explicitly granted to it, but a lot of its ability to get things done comes from the respect and cooperation of its worker bees.

In more technical terms, the event loop contains a collection of jobs to be run. Some jobs are added directly by you, and some indirectly by **`asyncio`**. The event loop takes a job from its backlog of work and invokes it (or «gives it control»), similar to calling a function, and then that job runs. Once it pauses or completes, it returns control to the event loop. The event loop will then select another job from its pool and invoke it. You can *roughly* think of the collection of jobs as a queue: jobs are added and then processed one at a time, generally (but not always) in order. This process repeats indefinitely, with the event loop cycling endlessly onwards. If there are no more jobs pending execution, the event loop is smart enough to rest and avoid needlessly wasting CPU cycles, and will come back when there's more work to be done.

Effective execution relies on jobs sharing well and cooperating; a greedy job could hog control and leave the other jobs to starve, rendering the overall event loop approach rather useless.

```
import asyncio

# This creates an event loop and indefinitely cycles thr
# its collection of jobs.
event_loop = asyncio.new_event_loop()
event_loop.run_forever()
```

Asynchronous functions and coroutines

This is a basic, boring Python function:

```
def hello_printer():
    print(
        "Hi, I am a lowly, simple printer, though I have
        "need in life -- \nfresh paper and my dearly bel
        "partner in crime."
    )
```

Calling a regular function invokes its logic or body:

```
>>> hello_printer()
Hi, I am a lowly, simple printer, though I have all I ne
fresh paper and my dearly beloved octopus partner in cri
```

The async def, as opposed to just a plain `def`, makes this an asynchronous function (or «coroutine function»). Calling it creates and returns a coroutine object.

```
async def loudmouth_penguin(magic_number: int):
    print(
     "I am a super special talking penguin. Far cooler t
     f"By the way, my lucky number is: {magic_number}."
    )
```

Calling the async function, `loudmouth_penguin`, does not execute the print statement; instead, it creates a coroutine object:

```
>>> loudmouth_penguin(magic_number=3)
<coroutine object loudmouth_penguin at 0x104ed2740>
```

The terms «coroutine function» and «coroutine object» are often conflated as coroutine. That can be confusing! In this article, coroutine specifically refers to a coroutine object, or more precisely, an instance of `types.CoroutineType` (native coroutine). Note that coroutines can also exist as instances of `collections.abc.Coroutine` – a distinction that matters for type checking.

A coroutine represents the function's body or logic. A coroutine has to be explicitly started; again, merely creating the coroutine does not start it. Notably, the coroutine can be paused and resumed at various points within the function's body. That pausing and resuming ability is what allows for asynchronous behavior!

Coroutines and coroutine functions were built by leveraging the functionality of generators and generator functions. Recall, a generator function is a function that **`yield`**s, like this one:

```
def get_random_number():
    # This would be a bad random number generator!
    print("Hi")
    yield 1
    print("Hello")
    yield 7
    print("Howdy")
    yield 4
    ...
```

Similar to a coroutine function, calling a generator function does not run it. Instead, it creates a generator object:

```
>>> get_random_number()
<generator object get_random_number at 0x1048671c0>
```

You can proceed to the next `yield` of a generator by using the built-in function **`next()`**. In other words, the generator runs, then pauses. For example:

```
>>> generator = get_random_number()
>>> next(generator)
Hi
1
>>> next(generator)
Hello
7
```

Tasks

Roughly speaking, tasks are coroutines (not coroutine functions) tied to an event loop. A task also maintains a list of callback functions whose importance will become clear in a moment when we discuss **`await`**. The recommended way to create tasks is via **`asyncio.create_task()`**.

Creating a task automatically schedules it for execution (by adding a callback to run it in the event loop's to-do list, that is, collection of jobs).

`asyncio` automatically associates tasks with the event loop for you. This automatic association was purposely designed into **`asyncio`** for the sake of simplicity. Without it, you'd have to keep track of the event loop object and pass it to any coroutine function that wants to create tasks, adding redundant clutter to your code.

```
coroutine = loudmouth_penguin(magic_number=5)
# This creates a Task object and schedules its executio
task = asyncio.create_task(coroutine)
```

Earlier, we manually created the event loop and set it to run forever. In practice, it's recommended to use (and common to see) **`asyncio.run()`**, which takes care of managing the event loop and ensuring the provided coroutine finishes before advancing. For example, many async programs follow this setup:

```
import asyncio

async def main():
    # Perform all sorts of wacky, wild asynchronous thi
    ...

if __name__ == "__main__":
    asyncio.run(main())
    # The program will not reach the following print sta
    # coroutine main() finishes.
    print("coroutine main() is done!")
```

It's important to be aware that the task itself is not added to the event loop, only a callback to the task is. This matters if the task object you created is garbage collected before it’s called by the event loop. For example, consider this program:

```
 1async def hello():
 2    print("hello!")
 3
 4async def main():
 5    asyncio.create_task(hello())
 6    # Other asynchronous instructions which run for a
 7    # and cede control to the event loop...
 8    ...
 9
10asyncio.run(main())
```

Because there’s no reference to the task object created on line 5, it *might* be garbage collected before the event loop invokes it. Later instructions in the coroutine `main()` hand control back to the event loop so it can invoke other jobs. When the event loop eventually tries to run the task, it might fail and discover the task object does not exist! This can also happen even if a coroutine keeps a reference to a task but completes before that task finishes. When the coroutine exits, local variables go out of scope and may be subject to garbage collection. In practice, `asyncio` and Python’s garbage collector work pretty hard to ensure this sort of thing doesn’t happen. But that’s no reason to be reckless!

await

`await` is a Python keyword that’s commonly used in one of two different ways:

```
await task
await coroutine
```

In a crucial way, the behavior of `await` depends on the type of object being awaited.

Awaiting a task will cede control from the current task or coroutine to the event loop. In the process of relinquishing control, a few important things happen. We’ll use the following code example to illustrate:

```
async def plant_a_tree():
    dig_the_hole_task = asyncio.create_task(dig_the_hole
    await dig_the_hole_task

    # Other instructions associated with planting a tree
    ...
```

In this example, imagine the event loop has passed control to the start of the coroutine `plant_a_tree()`. As seen above, the coroutine creates a task and then awaits it. The `await dig_the_hole_task` instruction adds a callback (which will resume `plant_a_tree()`) to the `dig_the_hole_task` object's list of callbacks. And then, the instruction cedes control to the event loop. Some time later, the event loop will pass control to `dig_the_hole_task` and the task will finish whatever it needs to do. Once the task finishes, it will add its various callbacks to the event loop, in this case, a call to resume `plant_a_tree()`.

Generally speaking, when the awaited task finishes (`dig_the_hole_task`), the original task or coroutine (`plant_a_tree()`) is added back to the event loop's to-do list to be resumed.

This is a basic, yet reliable mental model. In practice, the control handoffs are slightly more complex, but not by much. In part 2, we'll walk through the details that make this possible.

Unlike tasks, awaiting a coroutine does not hand control back to the event loop! Wrapping a coroutine in a task first, then awaiting that would cede control. The behavior of `await coroutine` is effectively the same as invoking a regular, synchronous Python function. Consider this program:

```
import asyncio

async def coro_a():
   print("I am coro_a(). Hi!")
```

```
async def coro_b():
    print("I am coro_b(). I sure hope no one hogs the eve

async def main():
    task_b = asyncio.create_task(coro_b())
    num_repeats = 3
    for _ in range(num_repeats):
        await coro_a()
    await task_b

asyncio.run(main())
```

The first statement in the coroutine `main()` creates `task_b` and schedules it for execution via the event loop. Then, `coro_a()` is repeatedly awaited. Control never cedes to the event loop, which is why we see the output of all three `coro_a()` invocations before `coro_b()`"s output:

```
I am coro_a(). Hi!
I am coro_a(). Hi!
I am coro_a(). Hi!
I am coro_b(). I sure hope no one hogs the event loop...
```

If we change `await coro_a()` to `await asyncio.create_task(coro_a())`, the behavior changes. The coroutine `main()` cedes control to the event loop with that statement. The event loop then proceeds through its backlog of work, calling `task_b` and then the task which wraps `coro_a()` before resuming the coroutine `main()`.

```
I am coro_b(). I sure hope no one hogs the event loop...
I am coro_a(). Hi!
I am coro_a(). Hi!
I am coro_a(). Hi!
```

This behavior of `await coroutine` can trip a lot of people up! That example highlights how using only `await coroutine` could unintentionally hog control from other tasks and effectively stall the event loop. **`asyncio.run()`** can help you detect such occurrences via the `debug=True` flag, which enables debug mode. Among other things, it will log any coroutines that monopolize execution for 100ms or longer.

The design intentionally trades off some conceptual clarity around usage of `await` for improved performance. Each time a task is awaited, control needs to be passed all the way up the call stack to the event loop. That might sound minor, but in a large program with many `await` statements and a deep call stack, that overhead can add up to a meaningful performance drag.

A conceptual overview part 2: the nuts and bolts

Part 2 goes into detail on the mechanisms **`asyncio`** uses to manage control flow. This is where the magic happens. You’ll come away from this section knowing what `await` does behind the scenes and how to make your own asynchronous operators.

The inner workings of coroutines

`asyncio` leverages four components to pass around control.

`coroutine.send(arg)` is the method used to start or resume a coroutine. If the coroutine was paused and is now being resumed, the argument `arg` will be sent in as the return value of the `yield` statement which originally paused it. If the coroutine is being used for the first time (as opposed to being resumed), `arg` must be `None`.

```
class Rock:
    def __await__(self):
        value_sent_in = yield 7
        print(f"Rock.__await__ resuming with value: {v
        return value_sent_in

async def main():
    print("Beginning coroutine main().")
    rock = Rock()
    print("Awaiting rock...")
```

```
11    value_from_rock = await rock
12    print(f"Coroutine received value: {value_from_rock
13    return 23
14
15coroutine = main()
16intermediate_result = coroutine.send(None)
17print(f"Coroutine paused and returned intermediate val
18
19print(f"Resuming coroutine and sending in value: 42.")
20try:
21    coroutine.send(42)
22except StopIteration as e:
23    returned_value = e.value
24print(f"Coroutine main() finished and provided value:
```

yield, as usual, pauses execution and returns control to the caller. In the example above, the `yield`, on line 3, is called by `... = await rock` on line 11. More broadly speaking, `await` calls the **`__await__()`** method of the given object. `await` also does one more very special thing: it propagates (or «passes along») any `yield`s it receives up the call chain. In this case, that's back to `... = coroutine.send(None)` on line 16.

The coroutine is resumed via the `coroutine.send(42)` call on line 21. The coroutine picks back up from where it `yield`ed (or paused) on line 3 and executes the remaining statements in its body. When a coroutine finishes, it raises a **`StopIteration`** exception with the return value attached in the **`value`** attribute.

That snippet produces this output:

```
Beginning coroutine main().
Awaiting rock...
Coroutine paused and returned intermediate value: 7.
Resuming coroutine and sending in value: 42.
Rock.__await__ resuming with value: 42.
Coroutine received value: 42 from rock.
Coroutine main() finished and provided value: 23.
```

It's worth pausing for a moment here and making sure you followed the various ways that control flow and values were passed. A lot of important ideas were covered and it’s worth ensuring your understanding is firm.

The only way to yield (or effectively cede control) from a coroutine is to `await` an object that `yield`s in its `__await__` method. That might sound odd to you. You might be thinking:

1. What about a `yield` directly within the coroutine function? The coroutine function becomes an async generator function, a different beast entirely.

2. What about a yield from within the coroutine function to a (plain) generator? That causes the error: `SyntaxError: yield from not allowed in a coroutine`. This was intentionally designed for the sake of simplicity – mandating only one way of using coroutines. Initially `yield` was barred as well, but was re-accepted to allow for async generators. Despite that, `yield from` and `await` effectively do the same thing.

Futures

A future is an object meant to represent a computation’s status and result. The term is a nod to the idea of something still to come or not yet happened, and the object is a way to keep an eye on that something.

A future has a few important attributes. One is its state, which can be either «pending», «cancelled», or «done». Another is its result, which is set when the state transitions to done. Unlike a coroutine, a future does not represent the actual computation to be done; instead, it represents the status and result of that computation, kind of like a status light (red, yellow, or green) or indicator.

`asyncio.Task` subclasses **`asyncio.Future`** in order to gain these various capabilities. The prior section said tasks store a list of callbacks, which wasn’t entirely accurate. It’s actually the `Future` class that implements this logic, which `Task` inherits.

Futures may also be used directly (not via tasks). Tasks mark themselves as done when their coroutine is complete. Futures are much more versatile and will be marked as done when you say so. In this way, they're the flexible interface for you to make your own conditions for waiting and resuming.

A homemade asyncio.sleep

We'll go through an example of how you could leverage a future to create your own variant of asynchronous sleep (`async_sleep`) which mimics **`asyncio.sleep()`**.

This snippet registers a few tasks with the event loop and then awaits the task created by `asyncio.create_task`, which wraps the `async_sleep(3)` coroutine. We want that task to finish only after three seconds have elapsed, but without preventing other tasks from running.

```
async def other_work():
    print("I like work. Work work.")

async def main():
    # Add a few other tasks to the event loop, so there'
    # to do while asynchronously sleeping.
    work_tasks = [
        asyncio.create_task(other_work()),
        asyncio.create_task(other_work()),
        asyncio.create_task(other_work())
    ]
    print(
        "Beginning asynchronous sleep at time: "
        f"{datetime.datetime.now().strftime("%H:%M:%S")}
    )
    await asyncio.create_task(async_sleep(3))
    print(
        "Done asynchronous sleep at time: "
        f"{datetime.datetime.now().strftime("%H:%M:%S")}
    )
    # asyncio.gather effectively awaits each task in the
    await asyncio.gather(*work_tasks)
```

Below, we use a future to enable custom control over when that task will be marked as done. If **`future.set_result()`** (the method responsible for marking that future as done) is never called, then this task will never finish. We've also enlisted the help of another task, which we'll see in a moment, that will monitor how much time has elapsed and, accordingly, call `future.set_result()`.

```
async def async_sleep(seconds: float):
    future = asyncio.Future()
    time_to_wake = time.time() + seconds
    # Add the watcher-task to the event loop.
    watcher_task = asyncio.create_task(_sleep_watcher(fu
    # Block until the future is marked as done.
    await future
```

Below, we use a rather bare `YieldToEventLoop()` object to `yield` from its `__await__` method, ceding control to the event loop. This is effectively the same as calling `asyncio.sleep(0)`, but this approach offers more clarity, not to mention it's somewhat cheating to use `asyncio.sleep` when showcasing how to implement it!

As usual, the event loop cycles through its tasks, giving them control and receiving control back when they pause or finish. The `watcher_task`, which runs the coroutine `_sleep_watcher(...)`, will be invoked once per full cycle of the event loop. On each resumption, it'll check the time and if not enough has elapsed, then it'll pause once again and hand control back to the event loop. Once enough time has elapsed, `_sleep_watcher(...)` marks the future as done and completes by exiting its infinite `while` loop. Given this helper task is only invoked once per cycle of the event loop, you'd be correct to note that this asynchronous sleep will sleep *at least* three seconds, rather than exactly three seconds. Note this is also true of `asyncio.sleep`.

```
class YieldToEventLoop:
    def __await__(self):
        yield

async def _sleep_watcher(future, time_to_wake):
    while True:
        if time.time() >= time_to_wake:
            # This marks the future as done.
            future.set_result(None)
            break
        else:
            await YieldToEventLoop()
```

Here is the full program's output:

```
$ python custom-async-sleep.py
Beginning asynchronous sleep at time: 14:52:22.
I like work. Work work.
I like work. Work work.
I like work. Work work.
Done asynchronous sleep at time: 14:52:25.
```

You might feel this implementation of asynchronous sleep was unnecessarily convoluted. And, well, it was. The example was meant to showcase the versatility of futures with a simple example that could be mimicked for more complex needs. For reference, you could implement it without futures, like so:

```
async def simpler_async_sleep(seconds):
    time_to_wake = time.time() + seconds
    while True:
        if time.time() >= time_to_wake:
            return
        else:
            await YieldToEventLoop()
```

But that's all for now. Hopefully you're ready to more confidently dive into some async programming or check out advanced topics in the **rest of the documentation**.

Portar módulos de extensión a Python 3

Recomendamos los siguientes recursos para portar módulos de extensiones a Python 3:

- El capítulo Migrating C extensions [http://python3porting.com/cextensions.html] de *Support Python 3: An in-depth guide*, un libro sobre migrar de Python 2 a Python 3 en general, guía al lector a través cómo portar un módulo de extensión.
- La Porting guide [https://py3c.readthedocs.io/en/latest/guide.html] de el proyecto *py3c* provee sugerencias dogmáticas con código de ejemplo.
- Recommended third party tools offer abstractions over the Python's C API. Extensions generally need to be re-written to use one of them, but the library then handles differences between various Python versions and implementations.

Programación de curses con Python

Autor:
A.M. Kuchling, Eric S. Raymond
Versión:
2.04

Resumen

Este documento describe cómo usar el módulo de extensión `curses` para controlar las pantallas en modo texto.

¿Qué es curses?

La biblioteca curses proporciona una instalación independiente del terminal para manejo de teclado e impresión de pantalla en terminales basados en texto; tales terminales incluyen VT100, la consola de Linux y el terminal simulado proporcionado por varios programas. Los terminales de pantalla admiten varios códigos de control para realizar operaciones comunes, como mover el cursor, desplazar la pantalla y borrar áreas. Diferentes terminales utilizan códigos muy diferentes y, a menudo, tienen sus propias peculiaridades menores.

En un mundo de pantallas gráficas, uno podría preguntarse « ¿por qué molestarse »? Es cierto que los terminales de pantalla de celdas de caracteres son una tecnología obsoleta, pero hay nichos en los que poder hacer cosas elegantes con ellos sigue siendo valioso. Un nicho está en pequeños *Unix* incrustados o pequeñas marcas que no ejecutan un servidor X. Por otro lado herramientas como los instaladores del sistema operativo y los configuradores de kernel que pueden tener que ejecutarse antes de que haya soporte gráfico disponible.

The curses library provides fairly basic functionality, providing the programmer with an abstraction of a display containing multiple non-overlapping windows of text. The contents of a window can be changed in various ways—adding text, erasing it, changing its appearance—and the curses library will figure out what control codes need to be sent to the terminal to produce the right output. curses doesn't provide many user-interface concepts such as buttons, checkboxes, or dialogs; if you need such features, consider a user interface library such as Urwid [https://pypi.org/project/Urwid/].

La biblioteca curses se escribió originalmente para BSD Unix; Las versiones posteriores de Unix de *System V* de AT&T agregaron muchas mejoras y nuevas funciones. BSD curses ya no se mantiene, ya que ha sido reemplazado por ncurses, que es una implementación de código abierto de la interfaz AT&T. Si está utilizando un Unix de código abierto como Linux o FreeBSD, su sistema casi seguro usa ncurses. Como la mayoría de las versiones comerciales actuales de Unix se basan en el código del Sistema V, todas las funciones descritas aquí probablemente estarán disponibles. Sin embargo, las versiones anteriores de curses llevadas por algunos Unix propietarios no pueden admitir todo.

The Windows version of Python doesn't include the **`curses`** module. A ported version called UniCurses [https://pypi.org/project/UniCurses/] is available.

El módulo curses de Python

The Python module is a fairly simple wrapper over the C functions provided by curses; if you're already familiar with curses programming in C, it's really easy to transfer that knowledge to Python. The biggest difference is that the Python interface makes things simpler by merging different C functions such as **`addstr()`**, **`mvaddstr()`**, and **`mvwaddstr()`** into a single **`addstr()`** method. You'll see this covered in more detail later.

Este HOWTO es una introducción a la escritura de programas en modo texto con curses y Python. No intenta ser una guía completa de la API de curses; para eso, vea la sección de la guía de la biblioteca de Python sobre ncurses, y las páginas del manual de C para ncurses. Sin embargo, le dará las ideas básicas.

Iniciar y finalizar una aplicación de curses

Before doing anything, curses must be initialized. This is done by calling the **`initscr()`** function, which will determine the terminal type, send any required setup codes to the terminal, and create various internal data structures. If successful, **`initscr()`** returns a window object representing the entire screen; this is usually called `stdscr` after the name of the corresponding C variable.

```
import curses
stdscr = curses.initscr()
```

Por lo general, las aplicaciones de curses desactivan el eco automático de las teclas en la pantalla, para poder leer las teclas y solo mostrarlas bajo ciertas circunstancias. Esto requiere llamar a la función **`noecho()`** function.

```
curses.noecho()
```

Las aplicaciones también tendrán que reaccionar a las teclas al instante, sin necesidad de presionar la tecla Intro; Esto se llama modo cbreak, en oposición al modo de entrada almacenado en memoria intermedia habitual.

```
curses.cbreak()
```

Los terminales generalmente retornan teclas especiales, como las teclas del cursor o las teclas de navegación, como Re Pág e Inicio, como una secuencia de escape multibyte. Si bien puede escribir su aplicación para esperar tales secuencias y procesarlas en consecuencia, curses pueden hacerlo por usted, retornando un valor especial como **`curses.KEY_LEFT`**. Para obtener que curses haga el trabajo, deberá habilitar el modo de teclado.

```
stdscr.keypad(True)
```

Terminar una aplicación de curses es mucho más fácil que iniciar una. Tendrás que llamar a

```
curses.nocbreak()
stdscr.keypad(False)
curses.echo()
```

para revertir la configuración de terminal amigable de curses. Llame luego a la función `endwin()` para restaurar el terminal a su modo operativo original.

```
curses.endwin()
```

Un problema común al depurar una aplicación curses es hacer que su terminal se estropee cuando la aplicación muere sin restaurar el terminal a su estado anterior. En Python, esto ocurre comúnmente cuando su código tiene errores y genera una excepción no detectada. Las teclas ya no se repiten en la pantalla cuando las escribe, por ejemplo, lo que dificulta el uso del shell.

En Python puede evitar estas complicaciones y facilitar la depuración importando la función `curses.wrapper()` y usándola así:

```
from curses import wrapper

def main(stdscr):
    # Clear screen
    stdscr.clear()

    # This raises ZeroDivisionError when i == 10.
    for i in range(0, 11):
        v = i-10
        stdscr.addstr(i, 0, '10 divided by {} is {}'.for

        stdscr.refresh()
        stdscr.getkey()

wrapper(main)
```

The `wrapper()` function takes a callable object and does the initializations described above, also initializing colors if color support is present. **`wrapper()`** then runs your provided callable. Once the callable returns, **`wrapper()`** will restore the original state of the terminal. The callable is called inside a `try`…`except` that catches exceptions, restores the state of the terminal, and then re-raises the exception. Therefore your terminal won't be left in a funny state on exception and you'll be able to read the exception's message and traceback.

Ventanas y pads

Las ventanas son la abstracción básica en curses. Un objeto de ventana representa un área rectangular de la pantalla y admite métodos para mostrar texto, borrarlo, permitir al usuario ingresar cadenas, etc.

El objeto `stdscr` retornado por la función `initscr()` es un objeto de ventana que cubre toda la pantalla. Es posible que muchos programas solo necesiten esta única ventana, pero es posible que desee dividir la pantalla en ventanas más pequeñas para volver a dibujarlas o borrarlas por separado. La función `newwin()` crea una nueva ventana de un tamaño dado, retornando el nuevo objeto de ventana.

```
begin_x = 20; begin_y = 7
height = 5; width = 40
win = curses.newwin(height, width, begin_y, begin_x)
```

Tenga en cuenta que el sistema de coordenadas utilizado en curses es inusual. Las coordenadas siempre se pasan en el orden *y,x*, y la esquina superior izquierda de una ventana es la coordenada (0,0). Esto rompe la convención normal para manejar coordenadas donde la coordenada *x* es lo primero. Esta es una desafortunada diferencia de la mayoría de las otras aplicaciones informáticas, pero ha sido parte de curses desde que se escribió por primera vez, y ahora es demasiado tarde para cambiar las cosas.

Su aplicación puede determinar el tamaño de la pantalla utilizando las variables `curses.LINES` y `curses.COLS` para obtener los tamaños *y* y *x*. Las coordenadas legales se extenderán de `(0,0)` a `(curses.LINES - 1, curses.COLS - 1)`.

Cuando llama a un método para mostrar o borrar texto, el efecto no aparece inmediatamente en la pantalla. En su lugar, debe llamar al método **`refresh()`** de los objetos de ventana para actualizar la pantalla.

This is because curses was originally written with slow 300-baud terminal connections in mind; with these terminals, minimizing the time required to redraw the screen was very important. Instead curses accumulates changes to the screen and displays them in the most efficient manner when you call **`refresh()`**. For example, if your program displays some text in a window and then clears the window, there's no need to send the original text because they're never visible.

In practice, explicitly telling curses to redraw a window doesn't really complicate programming with curses much. Most programs go into a flurry of activity, and then pause waiting for a keypress or some other action on the part of the user. All you have to do is to be sure that the screen has been redrawn before pausing to wait for user input, by first calling **`stdscr.refresh()`** or the **`refresh()`** method of some other relevant window.

Un pad es un caso especial de una ventana; puede ser más grande que la pantalla de visualización real, y solo se muestra una parte del pad a la vez. La creación de un pad requiere la altura y el ancho del pad, mientras que la actualización de un pad requiere dar las coordenadas del área en pantalla donde se mostrará una subsección del pad.

```
pad = curses.newpad(100, 100)
# These loops fill the pad with letters; addch() is
# explained in the next section
for y in range(0, 99):
    for x in range(0, 99):
        pad.addch(y,x, ord('a') + (x*x+y*y) % 26)

# Displays a section of the pad in the middle of the scr
# (0,0) : coordinate of upper-left corner of pad area to
# (5,5) : coordinate of upper-left corner of window area
#         with pad content.
# (20, 75) : coordinate of lower-right corner of window
#          : filled with pad content.
pad.refresh( 0,0, 5,5, 20,75)
```

The **`refresh()`** call displays a section of the pad in the rectangle extending from coordinate (5,5) to coordinate (20,75) on the screen; the upper left corner of the displayed section is coordinate (0,0) on the pad. Beyond that difference, pads are exactly like ordinary windows and support the same methods.

If you have multiple windows and pads on screen there is a more efficient way to update the screen and prevent annoying screen flicker as each part of the screen gets updated. **`refresh()`** actually does two things:

1. Llama al método **`noutrefresh()`** de cada ventana para actualizar una estructura de datos subyacente que representa el estado deseado de la pantalla.
2. Llama a la función **`doupdate()`** para cambiar la pantalla física para que coincida con el estado deseado registrado en la estructura de datos.

Instead you can call **`noutrefresh()`** on a number of windows to update the data structure, and then call **`doupdate()`** to update the screen.

Mostrando el texto

From a C programmer's point of view, curses may sometimes look like a twisty maze of functions, all subtly different. For example, **`addstr()`** displays a string at the current cursor location in the `stdscr` window, while **`mvaddstr()`** moves to a given y,x coordinate first before displaying the string. **`waddstr()`** is just like **`addstr()`**, but allows specifying a window to use instead of using `stdscr` by default. **`mvwaddstr()`** allows specifying both a window and a coordinate.

Afortunadamente, la interfaz de Python oculta todos estos detalles. `stdscr` es un objeto de ventana como cualquier otro, y métodos como `addstr()` aceptan múltiples formas de argumento. Por lo general, hay cuatro formas diferentes.

Forma	**Descripción**
str o *ch*	Mostrar la cadena *str* o el carácter *ch* en la posición actual
str o *ch*, *attr*	Muestra la cadena *str* o el carácter *ch*, utilizando el atributo *attr* en la posición actual
y, *x*, *str* o *ch*	Moverse a la posición *y,x* dentro de la ventana, y mostrar *st* o *ch*
y, *x*, *str* o *ch*, *attr*	Muévase a la posición *y, x* dentro de la ventana y muestre *str* o *ch*, usando el atributo *attr*

Los atributos permiten mostrar texto en formas resaltadas como negrita, subrayado, código inverso o en color. Se explicarán con más detalle en la siguiente subsección.

The `addstr()` method takes a Python string or bytestring as the value to be displayed. The contents of bytestrings are sent to the terminal as-is. Strings are encoded to bytes using the value of the window’s `encoding` attribute; this defaults to the default system encoding as returned by `locale.getencoding()`.

Los métodos `addch()` toman un carácter, que puede ser una cadena de longitud 1, una cadena de bytes de longitud 1 o un entero.

Se proporcionan constantes para los caracteres de extensión; estas constantes son enteros mayores que 255. Por ejemplo `ACS_PLMINUS` es un símbolo +/-, y `ACS_ULCORNER` es la esquina superior izquierda de un cuadro (útil para dibujar bordes). También puede usar el carácter Unicode apropiado.

Windows recuerda dónde se dejó el cursor después de la última operación, por lo que si deja de lado las coordenadas *y, x*, la cadena o el carácter se mostrarán donde se haya quedado la última operación. También puede mover el cursor con el método `move(y, x)`. Debido a que algunos terminales siempre muestran un cursor parpadeante, es posible que desee asegurarse de que el cursor esté ubicado en algún lugar donde no distraiga; Puede ser confuso tener el cursor parpadeando en alguna ubicación aparentemente aleatoria.

Si su aplicación no necesita un cursor parpadeante, puede llamar a `curs_set (False)` para hacerla invisible. Para compatibilidad con versiones anteriores de curses, hay una función `jeaveok (bool)` que es sinónimo de **`curs_set()`**. Cuando *bool* es verdadero, la biblioteca curses intentará suprimir el cursor parpadeante, y no tendrá que preocuparse por dejarlo en ubicaciones extrañas.

Atributos y color

Los caracteres se pueden mostrar de diferentes maneras. Las líneas de estado en una aplicación basada en texto se muestran comúnmente en video inverso, o un visor de texto puede necesitar resaltar ciertas palabras. curses admite esto al permitirle especificar un atributo para cada celda en la pantalla.

Un atributo es un entero, cada bit representa un atributo diferente. Puede intentar mostrar texto con múltiples conjuntos de bits de atributos, pero curses no garantizan que todas las combinaciones posibles estén disponibles, o que todas sean visualmente distintas. Eso depende de la capacidad del terminal que se utilice, por lo que es más seguro apegarse a los atributos más comúnmente disponibles, enumerados aquí.

Atributo	Descripción
A_BLINK	Texto parpadeante
A_BOLD	Texto extra brillante o en negrita
A_DIM	Texto medio brillante
A_REVERSE	Texto de video inverso
A_STANDOUT	El mejor modo de resaltado disponible
A_UNDERLINE	Texto subrayado

Entonces, para mostrar una línea de estado de video inverso en la línea superior de la pantalla, puede codificar:

```
stdscr.addstr(0, 0, "Current mode: Typing mode",
              curses.A_REVERSE)
stdscr.refresh()
```

La biblioteca curses también admite color en los terminales que lo proporcionen. El terminal más común es probablemente la consola de Linux, seguido de *xterms* en color.

Para usar el color, debe llamar a la función `start_color()` poco después de llamar `initscr()`, para inicializar el conjunto de colores predeterminado (la función `curses.wrapper()` hace esto automáticamente). Una vez hecho esto, la función `has_colors()` retorna TRUE si el terminal en uso realmente puede mostrar color. (Nota: curses usa el 'color' de la ortografía estadounidense, en lugar del 'color' de la ortografía canadiense/británica. Si está acostumbrado a la ortografía británica, tendrá que resignarse a escribir mal por el bien de estas funciones.)

La biblioteca curses mantiene un número finito de pares de colores, que contienen un color de primer plano (o texto) y un color de fondo. Puede obtener el valor del atributo correspondiente a un par de colores con la función `color_pair()`; esto puede ser operado bit a bit con otros atributos como `A_REVERSE`, pero nuevamente, no se garantiza que tales combinaciones funcionen en todos los terminales.

Un ejemplo, que muestra una línea de texto usando el par de colores 1

```
stdscr.addstr("Pretty text", curses.color_pair(1))
stdscr.refresh()
```

Como dije antes, un par de colores consiste en un color de primer plano y de fondo. La función `init_pair (n, f, b)` cambia la definición del par de colores *n*, a color de primer plano f y color de fondo b. El par de colores 0 está cableado a blanco sobre negro, y no se puede cambiar.

Los colores están numerados y `start_color()` inicializa 8 colores básicos cuando activa el modo de color. Son: 0:negro, 1:rojo, 2:verde, 3:amarillo, 4:azul, 5:magenta, 6:cian y 7:blanco. El módulo `curses` define constantes con nombre para cada uno de estos colores: `curses.COLOR_BLACK`, `curses.COLOR_RED`, y así sucesivamente.

Pongamos todo esto juntos. Para cambiar el color 1 al texto rojo sobre un fondo blanco, debe llamar a:

```
curses.init_pair(1, curses.COLOR_RED, curses.COLOR_WHITE
```

Cuando cambia un par de colores, cualquier texto que ya se muestre usando ese par de colores cambiará a los nuevos colores. También puede mostrar texto nuevo en este color con:

```
stdscr.addstr(0,0, "RED ALERT!", curses.color_pair(1))
```

Los terminales muy elegantes pueden cambiar las definiciones de los colores reales a un valor RGB dado. Esto le permite cambiar el color 1, que generalmente es rojo, a púrpura o azul o cualquier otro color que desee. Desafortunadamente, la consola de Linux no admite esto, por lo que no puedo probarlo y no puedo proporcionar ningún ejemplo. Puede verificar si su terminal puede hacer esto llamando a **`can_change_color()`**, que retorna `Verdadero` si la capacidad está ahí. Si tiene la suerte de tener un terminal tan talentoso, consulte las páginas de manual de su sistema para obtener más información.

Input del usuario

The C curses library offers only very simple input mechanisms. Python's **`curses`** module adds a basic text-input widget. (Other libraries such as Urwid [https://pypi.org/project/Urwid/] have more extensive collections of widgets.)

Hay dos métodos para obtener información desde una ventana:

- **`getch()`** actualiza la pantalla y luego espera a que el usuario presione una tecla, mostrando la tecla si **`echo()`** ha sido llamado anteriormente. Opcionalmente, puede especificar una coordenada a la que se debe mover el cursor antes de pausar.
- **`getkey()`** hace lo mismo pero convierte el entero en una cadena. Los caracteres individuales se retornan como cadenas de 1 carácter, y las teclas especiales como las teclas de función retornan cadenas más largas que contienen un nombre de tecla como `KEY_UP` o `^ G`.

It’s possible to not wait for the user using the **`nodelay()`** window method. After `nodelay(True)`, **`getch()`** and **`getkey()`** for the window become non-blocking. To signal that no input is ready, **`getch()`** returns `curses.ERR` (a value of -1) and **`getkey()`** raises an exception. There’s also a **`halfdelay()`** function, which can be used to (in effect) set a timer on each **`getch()`**; if no input becomes available within a specified delay (measured in tenths of a second), curses raises an exception.

The **`getch()`** method returns an integer; if it’s between 0 and 255, it represents the ASCII code of the key pressed. Values greater than 255 are special keys such as Page Up, Home, or the cursor keys. You can compare the value returned to constants such as **`curses.KEY_PPAGE`**, **`curses.KEY_HOME`**, or **`curses.KEY_LEFT`**. The main loop of your program may look something like this:

```
while True:
    c = stdscr.getch()
    if c == ord('p'):
        PrintDocument()
    elif c == ord('q'):
        break  # Exit the while loop
    elif c == curses.KEY_HOME:
        x = y = 0
```

El módulo **`curses.ascii`** proporciona funciones de membresía de clase ASCII que toman argumentos de cadena de entero o de 1 carácter; Estos pueden ser útiles para escribir pruebas más legibles para dichos bucles. También proporciona funciones de conversión que toman argumentos enteros o de cadena de 1 carácter y retornen el mismo tipo. Por ejemplo, **`curses.ascii.ctrl()`** retorna el carácter de control correspondiente a su argumento.

También hay un método para recuperar una cadena completa, **`getstr()`**. No se usa con mucha frecuencia, porque su funcionalidad es bastante limitada; Las únicas teclas de edición disponibles son la tecla de retroceso y la tecla Intro, que termina la cadena. Opcionalmente, puede limitarse a un número fijo de caracteres.

```
curses.echo()            # Enable echoing of characters

# Get a 15-character string, with the cursor on the top
s = stdscr.getstr(0,0, 15)
```

El módulo `curses.textpad` proporciona un cuadro de texto que admite un conjunto de teclas tipo Emacs. Varios métodos de la clase `Textbox` admiten la edición con validación de entrada y recopilan los resultados de edición con o sin espacios finales. Aquí hay un ejemplo:

```
import curses
from curses.textpad import Textbox, rectangle

def main(stdscr):
    stdscr.addstr(0, 0, "Enter IM message: (hit Ctrl-G t

    editwin = curses.newwin(5,30, 2,1)
    rectangle(stdscr, 1,0, 1+5+1, 1+30+1)
    stdscr.refresh()

    box = Textbox(editwin)

    # Let the user edit until Ctrl-G is struck.
    box.edit()

    # Get resulting contents
    message = box.gather()
```

Consulte la documentación de la biblioteca sobre `curses.textpad` para más detalles.

Para más información

Este CÓMO no cubre algunos temas avanzados, como leer el contenido de la pantalla o capturar eventos del mouse desde una instancia de xterm, pero la página de la biblioteca de Python para el módulo `curses` ahora está razonablemente completa. Deberías buscarlo posteriormente.

If you're in doubt about the detailed behavior of the curses functions, consult the manual pages for your curses implementation, whether it's ncurses or a proprietary Unix vendor's. The manual pages will document any quirks, and provide complete lists of all the functions, attributes, and ACS_* characters available to you.

Debido a que la API de curses es tan grande, algunas funciones no son compatibles con la interfaz de Python. A menudo, esto no se debe a que sean difíciles de implementar, sino a que nadie los ha necesitado todavía. Además, Python aún no admite la biblioteca de menús asociada con ncurses. Los parches que agregan soporte para estos serían bienvenidos; consulte la Guía del desarrollador de Python [https://devguide.python.org/] para obtener más información sobre cómo enviar parches a Python.

- Escribir programas con NCURSES [https://invisible-island.net/ncurses/ncurses-intro.html]: un tutorial extenso para programadores en C.
- La página web con el manual de ncurses [https://linux.die.net/man/3/ncurses]
- Preguntas y respuestas frecuentes de ncurses [http://invisible-island.net/ncurses/ncurses.faq.html]
- «Use curses… don't swear» [https://www.youtube.com/watch?v=eN1eZtjLEnU]: video de una charla de PyCon 2013 sobre el control de terminales usando curses o Urwid.
- «Console Applications with Urwid» [https://www.pyvideo.org/video/1568/console-applications-with-urwid]: video de una charla en PyCon CA 2012 que muestra algunas aplicaciones escritas usando Urwid.

Descriptor Guide

Autor:
Raymond Hettinger
Contacto:
<python at rcn dot com>

Contenido

- Descriptor Guide
    - Guía introductoria
        - Ejemplo simple: un descriptor que retorna una constante
        - Búsquedas dinámicas
        - Atributos gestionados
        - Nombres personalizados
        - Pensamientos finales
    - Ejemplo completo práctico
        - Clase validadora
        - Validadores personalizados
        - Aplicación práctica
    - Tutorial técnico
        - Resumen
        - Definición e introducción
        - Protocolo de descriptores
        - Visión general de invocación de descriptores
        - Invocación desde una instancia
        - Invocación desde una clase
        - Invocación desde super
        - Resumen de la lógica de invocación
        - Notificación automática de nombre
        - Ejemplo de mapeos objeto-relacional (*ORM*)
    - Equivalentes en Python puro
        - Propiedades
        - Funciones y métodos
        - Tipos de métodos

- Métodos estáticos
- Métodos de clase
- Objetos miembros y _slots_

Los descriptores permiten a objetos personalizar la búsqueda, almacenamiento y eliminación de atributos.

Esta guía tiene cuatro secciones principales:

1. La guía introductoria da una visión general básica, moviéndose gentilmente por ejemplos simples, añadiendo una funcionalidad a la vez. Comienza acá si eres nuevo con los descriptores.
2. La segunda sección muestra un ejemplo completo y práctico de un descriptor. Si ya sabes lo básico comienza acá.
3. La tercera sección provee un tutorial más técnico que adentra en la mecánica detallada de cómo funcionan los descriptores. La mayoría de la gente no necesita este nivel de detalle.
4. La última sección tiene equivalentes en Python puro para descriptores incorporados que están escritos en C. Lee esta sección si tienes curiosidad de cómo las funciones se convierten en métodos vinculados, o sobre la implementación de herramientas comunes como **`classmethod()`**, **`staticmethod()`**, **`property()`**, y _slots_.

Guía introductoria

En esta guía introductoria comenzamos con el ejemplo más básico posible y luego vamos añadiendo nuevas funcionalidades una a una.

Ejemplo simple: un descriptor que retorna una constante

The **`Ten`** class is a descriptor whose **`__get__()`** method always returns the constant `10`:

```
class Ten:
    def __get__(self, obj, objtype=None):
        return 10
```

Para usar el descriptor, éste se debe almacenar como una variable de clase en otra clase:

```
class A:
    x = 5                       # Regular class attribut
    y = Ten()                   # Descriptor instance
```

Una sesión interactiva muestra la diferencia entre un la búsqueda de atributo normal y la búsqueda a través del descriptor:

```
>>> a = A()                     # Make an instance of cl
>>> a.x                         # Normal attribute looku
5
>>> a.y                         # Descriptor lookup
10
```

En la búsqueda de atributo `a.x`, el operador punto encuentra `'x': 5` en el diccionario de la clase. En la búsqueda `a.y`, el operador punto encuentra una instancia de un descriptor, reconocible por su método `__get__`. Llamar a ese método retorna `10`.

Nota que el valor `10` no es almacenado ni en el diccionario de la clase ni en el diccionario de la instancia. En cambio, el valor `10` es calculado bajo demanda.

Este ejemplo muestra cómo funciona un descriptor simple, pero no es muy útil. Para recuperar constantes una búsqueda de atributos normal sería mejor.

En la próxima sección crearemos algo más útil, una búsqueda dinámica.

Búsquedas dinámicas

Descriptores interesantes típicamente ejecutan cálculos en vez de retornar constantes:

```
import os

class DirectorySize:
    def __get__(self, obj, objtype=None):
        return len(os.listdir(obj.dirname))

class Directory:

    size = DirectorySize()              # Descriptor ins

    def __init__(self, dirname):
        self.dirname = dirname          # Regular instar
```

Una sesión interactiva muestra que la búsqueda es dinámica — calcula respuestas diferentes y actualizadas en cada ocasión:

```
>>> s = Directory('songs')
>>> g = Directory('games')
>>> s.size                              # The songs dire
20
>>> g.size                              # The games dire
3
>>> os.remove('games/chess')            # Delete a game
>>> g.size                              # File count is
2
```

Besides showing how descriptors can run computations, this example also reveals the purpose of the parameters to `__get__()`. The *self* parameter is *size*, an instance of *DirectorySize*. The *obj* parameter is either *g* or *s*, an instance of *Directory*. It is the *obj* parameter that lets the `__get__()` method learn the target directory. The *objtype* parameter is the class *Directory*.

Atributos gestionados

A popular use for descriptors is managing access to instance data. The descriptor is assigned to a public attribute in the class dictionary while the actual data is stored as a private attribute in the instance dictionary. The descriptor’s `__get__()` and `__set__()` methods are triggered when the public attribute is accessed.

En el siguiente ejemplo, *age* es el atributo público y *_age* es el atributo privado. Cuando el atributo público es accedido, el descriptor registra la búsqueda o actualización:

```
import logging

logging.basicConfig(level=logging.INFO)

class LoggedAgeAccess:

    def __get__(self, obj, objtype=None):
        value = obj._age
        logging.info('Accessing %r giving %r', 'age', va
        return value

    def __set__(self, obj, value):
        logging.info('Updating %r to %r', 'age', value)
        obj._age = value

class Person:

    age = LoggedAgeAccess()             # Descriptor ins

    def __init__(self, name, age):
        self.name = name                # Regular insta
        self.age = age                  # Calls __set__

    def birthday(self):
        self.age += 1                   # Calls both __g
```

Una sesión interactiva muestra que todos los accesos al atributo gestionado *age* son registrados, pero que el atributo normal *name* no es registrado:

```
>>> mary = Person('Mary M', 30)         # The initial ag
INFO:root:Updating 'age' to 30
>>> dave = Person('David D', 40)
INFO:root:Updating 'age' to 40
```

```
>>> vars(mary)                              # The actual dat
{'name': 'Mary M', '_age': 30}
>>> vars(dave)
{'name': 'David D', '_age': 40}

>>> mary.age                                # Access the dat
INFO:root:Accessing 'age' giving 30
30
>>> mary.birthday()                         # Updates are lo
INFO:root:Accessing 'age' giving 30
INFO:root:Updating 'age' to 31

>>> dave.name                               # Regular attrib
'David D'
>>> dave.age                                # Only the manag
INFO:root:Accessing 'age' giving 40
40
```

Un gran problema con este ejemplo es que el nombre privado *_age* está fijado en la clase *LoggedAgeAccess*. Esto significa que cada instancia puede sólo puede registrar un atributo, y que su nombre no se puede cambiar. En el siguiente ejemplo solucionaremos ese problema.

Nombres personalizados

Cuando una clase usa descriptores, puede informar a cada descriptor el nombre se usó para la variable.

In this example, the **`Person`** class has two descriptor instances, *name* and *age*. When the **`Person`** class is defined, it makes a callback to **`__set_name__()`** in *LoggedAccess* so that the field names can be recorded, giving each descriptor its own *public_name* and *private_name*:

```
import logging

logging.basicConfig(level=logging.INFO)
```

```
class LoggedAccess:

    def __set_name__(self, owner, name):
        self.public_name = name
        self.private_name = '_' + name

    def __get__(self, obj, objtype=None):
        value = getattr(obj, self.private_name)
        logging.info('Accessing %r giving %r', self.publ
        return value

    def __set__(self, obj, value):
        logging.info('Updating %r to %r', self.public_na
        setattr(obj, self.private_name, value)

class Person:

    name = LoggedAccess()                # First descrip
    age = LoggedAccess()                 # Second descri

    def __init__(self, name, age):
        self.name = name                 # Calls the fir
        self.age = age                   # Calls the sec

    def birthday(self):
        self.age += 1
```

An interactive session shows that the **`Person`** class has called **`__set_name__()`** so that the field names would be recorded. Here we call **`vars()`** to look up the descriptor without triggering it:

```
>>> vars(vars(Person)['name'])
{'public_name': 'name', 'private_name': '_name'}
>>> vars(vars(Person)['age'])
{'public_name': 'age', 'private_name': '_age'}
```

La nueva clase ahora registrar accesos tanto a *name* como a *age*:

```
>>> pete = Person('Peter P', 10)
```

```
INFO:root:Updating 'name' to 'Peter P'
INFO:root:Updating 'age' to 10
>>> kate = Person('Catherine C', 20)
INFO:root:Updating 'name' to 'Catherine C'
INFO:root:Updating 'age' to 20
```

Las dos instancias de *Person* contienen sólo los nombres privados:

```
>>> vars(pete)
{'_name': 'Peter P', '_age': 10}
>>> vars(kate)
{'_name': 'Catherine C', '_age': 20}
```

Pensamientos finales

A descriptor is what we call any object that defines **`__get__()`**, **`__set__()`**, or **`__delete__()`**.

Optionally, descriptors can have a **`__set_name__()`** method. This is only used in cases where a descriptor needs to know either the class where it was created or the name of class variable it was assigned to. (This method, if present, is called even if the class is not a descriptor.)

Los descriptores son invocados por el operador punto durante la búsqueda de atributos. Si un descriptor es accedido indirectamente con `vars(una_clase)[nombre_del_descriptor]`, la instancia del descriptor es retornada sin ser invocada.

Los descriptores sólo funcionan cuando se usan como variables de clase. Cuando son puestos en una instancia no tienen efecto.

La mayor motivación detrás de los descriptores es el proveer un gancho que permita a los objetos guardados en variables de clase controlar lo que ocurre al buscar un atributo.

Tradicionalmente, la clase que llama controla qué ocurre durante la búsqueda. Los descriptores invierten esta relación y permiten que los datos que están siendo buscados tengan algo qué decir al respecto.

Los descriptores se usan a través de todo el lenguaje. Es cómo funciones se convierten en métodos vinculados. Herramientas comunes como `classmethod()`, `staticmethod()`, `property()`, y `functools.cached_property()` se implementan todas como descriptores.

Ejemplo completo práctico

En este ejemplo creamos una herramienta práctica y poderosa para encontrar errores de corrupción de datos que son notoriamente difíciles de encontrar.

Clase validadora

Un validador es un descriptor que da acceso a un atributo gestionado. Antes de almacenar cualquier dato, verifica que el nuevo valor cumple con varias restricciones de tipo y rango. Si esas restricciones no se cumplen, lanza una excepción para así prevenir corrupción de datos en su origen.

This **`Validator`** class is both an abstract base class and a managed attribute descriptor:

```
from abc import ABC, abstractmethod

class Validator(ABC):

    def __set_name__(self, owner, name):
        self.private_name = '_' + name

    def __get__(self, obj, objtype=None):
        return getattr(obj, self.private_name)

    def __set__(self, obj, value):
        self.validate(value)
        setattr(obj, self.private_name, value)

    @abstractmethod
    def validate(self, value):
        pass
```

Custom validators need to inherit from **`Validator`** and must supply a **`validate()`** method to test various restrictions as needed.

Validadores personalizados

Acá hay tres utilidades de validación de datos prácticas:

1. **`OneOf`** verifies that a value is one of a restricted set of options.
2. **`Number`** verifies that a value is either an **`int`** or **`float`**. Optionally, it verifies that a value is between a given minimum or maximum.
3. **`String`** verifies that a value is a **`str`**. Optionally, it validates a given minimum or maximum length. It can validate a user-defined predicate [https://en.wikipedia.org/wiki/Predicate_(mathematical_logic)] as well.

```
class OneOf(Validator):

    def __init__(self, *options):
        self.options = set(options)

    def validate(self, value):
        if value not in self.options:
            raise ValueError(
                f'Expected {value!r} to be one of {self.
            )

class Number(Validator):

    def __init__(self, minvalue=None, maxvalue=None):
        self.minvalue = minvalue
        self.maxvalue = maxvalue

    def validate(self, value):
        if not isinstance(value, (int, float)):
            raise TypeError(f'Expected {value!r} to be a
        if self.minvalue is not None and value < self.mi
            raise ValueError(
                f'Expected {value!r} to be at least {sel
            )
        if self.maxvalue is not None and value > self.ma
            raise ValueError(
                f'Expected {value!r} to be no more than 
            )

class String(Validator):

    def __init__(self, minsize=None, maxsize=None, predi
        self.minsize = minsize
        self.maxsize = maxsize
        self.predicate = predicate

    def validate(self, value):
        if not isinstance(value, str):
            raise TypeError(f'Expected {value!r} to be a
        if self.minsize is not None and len(value) < sel
            raise ValueError(
                f'Expected {value!r} to be no smaller th
            )
        if self.maxsize is not None and len(value) > sel
            raise ValueError(
                f'Expected {value!r} to be no bigger tha
            )
        if self.predicate is not None and not self.predi
            raise ValueError(
                f'Expected {self.predicate} to be true f
            )
```

Aplicación práctica

Acá se muestra cómo se puede usar los validadores de datos en una clase real:

```
class Component:
    name = String(minsize=3, maxsize=10, predicate=str.i
    kind = OneOf('wood', 'metal', 'plastic')
    quantity = Number(minvalue=0)

    def __init__(self, name, kind, quantity):
        self.name = name
        self.kind = kind
        self.quantity = quantity
```

Los descriptores previenen que se creen instancias inválidas:

```
>>> Component('Widget', 'metal', 5)      # Blocked: 'Wic
Traceback (most recent call last):
    ...
ValueError: Expected <method 'isupper' of 'str' objects>

>>> Component('WIDGET', 'metle', 5)      # Blocked: 'met
Traceback (most recent call last):
    ...
ValueError: Expected 'metle' to be one of {'metal', 'pla

>>> Component('WIDGET', 'metal', -5)     # Blocked: -5 i
Traceback (most recent call last):
    ...
ValueError: Expected -5 to be at least 0

>>> Component('WIDGET', 'metal', 'V')    # Blocked: 'V'
Traceback (most recent call last):
    ...
TypeError: Expected 'V' to be an int or float

>>> c = Component('WIDGET', 'metal', 5)  # Allowed:  The
```

Tutorial técnico

Lo que sigue es un tutorial más práctico sobre las mecánicas y detalles de cómo funcionan los descriptores.

Resumen

Define los descriptores, resume el protocolo, y muestra cómo los descriptores son llamados. Provee ejemplos mostrando cómo funcionan los mapeos objeto-relacional (*ORM*).

Aprender acerca de los descriptores no sólo brinda acceso a un conjunto de herramientas mayor, sino que genera una comprensión más profunda de cómo funciona Python.

Definición e introducción

In general, a descriptor is an attribute value that has one of the methods in the descriptor protocol. Those methods are **`__get__()`**, **`__set__()`**, and **`__delete__()`**. If any of those methods are defined for an attribute, it is said to be a descriptor.

El comportamiento predeterminado para el acceso a los atributos es obtener, establecer o eliminar el atributo del diccionario de un objeto. Por ejemplo, `a.x` tiene una cadena de búsqueda que comienza con `a.__dict__['x']`, luego `type(a).__dict__['x']` y continúa a través del orden de resolución de métodos de `type(a)`. Si el valor buscado es un objeto que define uno de los métodos de descriptores, entonces Python puede anular el comportamiento predeterminado e invocar el método del descriptor en su lugar. El lugar donde esto ocurre en la cadena de precedencia depende de qué métodos de descriptores fueron definidos.

Descriptors are a powerful, general purpose protocol. They are the mechanism behind properties, methods, static methods, class methods, and **`super()`**. They are used throughout Python itself. Descriptors simplify the underlying C code and offer a flexible set of new tools for everyday Python programs.

Protocolo de descriptores

`descr.__get__(self, obj, type=None)`

`descr.__set__(self, obj, value)`

```
descr.__delete__(self, obj)
```

Eso es todo lo que hay que hacer. Si se define cualquiera de estos métodos, el objeto se considera un descriptor y puede anular el comportamiento predeterminado al ser buscado como un atributo.

If an object defines `__set__()` or `__delete__()`, it is considered a data descriptor. Descriptors that only define `__get__()` are called non-data descriptors (they are often used for methods but other uses are possible).

Los descriptores de datos y de no-datos difieren en cómo se calculan las anulaciones con respecto a las entradas en el diccionario de una instancia. Si el diccionario de una instancia tiene una entrada con el mismo nombre que un descriptor de datos, el descriptor de datos tiene prioridad. Si el diccionario de una instancia tiene una entrada con el mismo nombre que un descriptor de no-datos, la entrada del diccionario tiene prioridad.

To make a read-only data descriptor, define both `__get__()` and `__set__()` with the `__set__()` raising an `AttributeError` when called. Defining the `__set__()` method with an exception raising placeholder is enough to make it a data descriptor.

Visión general de invocación de descriptores

Un descriptor puede ser llamado directamente con `desc.__get__(obj)` o `desc.__get__(None, cls)`.

Pero es más común que un descriptor sea invocado automáticamente por la búsqueda de atributos.

The expression `obj.x` looks up the attribute `x` in the chain of namespaces for `obj`. If the search finds a descriptor outside of the instance `__dict__`, its `__get__()` method is invoked according to the precedence rules listed below.

Los detalles de la invocación dependen de si `obj` es un objeto una clase, o una instancia de super.

Invocación desde una instancia

Instance lookup scans through a chain of namespaces giving data descriptors the highest priority, followed by instance variables, then non-data descriptors, then class variables, and lastly `__getattr__()` if it is provided.

Si se encuentra un descriptor para `a.x` entonces se invoca con `desc.__get__(a, type(a))`.

La lógica para una búsqueda con puntos se encuentra en `object.__getattribute__()`. Acá hay un equivalente en Python puro:

```
def find_name_in_mro(cls, name, default):
    "Emulate _PyType_Lookup() in Objects/typeobject.c"
    for base in cls.__mro__:
        if name in vars(base):
            return vars(base)[name]
    return default

def object_getattribute(obj, name):
    "Emulate PyObject_GenericGetAttr() in Objects/object
    null = object()
    objtype = type(obj)
    cls_var = find_name_in_mro(objtype, name, null)
    descr_get = getattr(type(cls_var), '__get__', null)
    if descr_get is not null:
        if (hasattr(type(cls_var), '__set__')
            or hasattr(type(cls_var), '__delete__')):
            return descr_get(cls_var, obj, objtype)
    if hasattr(obj, '__dict__') and name in vars(obj):
        return vars(obj)[name]
    if descr_get is not null:
        return descr_get(cls_var, obj, objtype)
    if cls_var is not null:
        return cls_var
    raise AttributeError(name)
```

Note, there is no `__getattr__()` hook in the `__getattribute__()` code. That is why calling **`__getattribute__()`** directly or with `super().__getattribute__` will bypass **`__getattr__()`** entirely.

Instead, it is the dot operator and the **`getattr()`** function that are responsible for invoking **`__getattr__()`** whenever **`__getattribute__()`** raises an **`AttributeError`**. Their logic is encapsulated in a helper function:

```
def getattr_hook(obj, name):
    "Emulate slot_tp_getattr_hook() in Objects/typeobjec
    try:
        return obj.__getattribute__(name)
    except AttributeError:
        if not hasattr(type(obj), '__getattr__'):
            raise
    return type(obj).__getattr__(obj, name)
```

Invocación desde una clase

The logic for a dotted lookup such as `A.x` is in **`type.__getattribute__()`**. The steps are similar to those for **`object.__getattribute__()`** but the instance dictionary lookup is replaced by a search through the class's method resolution order.

Si se encuentra un descriptor, se invoca con `desc.__get__(None, A)`.

The full C implementation can be found in **`type_getattro()`** and **`_PyType_Lookup()`** in Objects/typeobject.c [https://github.com/python/cpython/tree/3.14/Objects/typeobject.c].

Invocación desde super

The logic for super's dotted lookup is in the **`__getattribute__()`** method for object returned by **`super()`**.

Una búsqueda con puntos tal como `super(A, obj).m` busca `obj.__class__.__mro__` para la clase base `B` que sigue inmediatamente a `A` y luego retorna

`B.__dict__['m'].__get__(obj, A)`. Si no es un descriptor, `m` se retorna sin cambiar.

The full C implementation can be found in **`super_getattro()`** in Objects/typeobject.c [https://github.com/python/cpython/tree/3.14/Objects/typeobject.c]. A pure Python equivalent can be found in Guido's Tutorial [https://www.python.org/download/releases/2.2.3/descrintro/#cooperation].

Resumen de la lógica de invocación

The mechanism for descriptors is embedded in the **`__getattribute__()`** methods for **`object`**, **`type`**, and **`super()`**.

Los puntos importantes a recordar son:

- Descriptors are invoked by the **`__getattribute__()`** method.
- Las clases heredan esta maquinaria desde **`object`**, **`type`**, o **`super()`**.
- Overriding **`__getattribute__()`** prevents automatic descriptor calls because all the descriptor logic is in that method.
- **`object.__getattribute__()`** and **`type.__getattribute__()`** make different calls to **`__get__()`**. The first includes the instance and may include the class. The second puts in `None` for the instance and always includes the class.
- Los descriptores de datos siempre anulan los diccionarios de instancia.
- Los descriptores de no-datos pueden ser reemplazados por los diccionarios de instancia.

Notificación automática de nombre

Sometimes it is desirable for a descriptor to know what class variable name it was assigned to. When a new class is created, the **`type`** metaclass scans the dictionary of the new class. If any of the entries are descriptors and if they define **`__set_name__()`**, that

method is called with two arguments. The *owner* is the class where the descriptor is used, and the *name* is the class variable the descriptor was assigned to.

The implementation details are in `type_new()` and `set_names()` in Objects/typeobject.c [https://github.com/python/cpython/tree/3.14/Objects/typeobject.c].

Since the update logic is in `type.__new__()`, notifications only take place at the time of class creation. If descriptors are added to the class afterwards, `__set_name__()` will need to be called manually.

Ejemplo de mapeos objeto-relacional (*ORM*)

El siguiente código es un esqueleto simplificado que muestra cómo descriptores de datos pueden ser usados para implementar un mapeo objeto-relacional [https://es.wikipedia.org/wiki/Asignaci%C3%B3n_objeto-relacional].

La idea esencial es que los datos se almacenan en una base de datos externa. Las instancias de Python sólo mantienen llaves a las tablas de la base de datos. Los descriptores se hacen cargo de las búsquedas o actualizaciones:

```
class Field:

    def __set_name__(self, owner, name):
        self.fetch = f'SELECT {name} FROM {owner.table}
        self.store = f'UPDATE {owner.table} SET {name}=

    def __get__(self, obj, objtype=None):
        return conn.execute(self.fetch, [obj.key]).fetch

    def __set__(self, obj, value):
        conn.execute(self.store, [value, obj.key])
        conn.commit()
```

We can use the `Field` class to define models [https://en.wikipedia.org/wiki/Database_model] that describe the schema for each table in a

database:

```
class Movie:
    table = 'Movies'                    # Table name
    key = 'title'                       # Primary key
    director = Field()
    year = Field()

    def __init__(self, key):
        self.key = key

class Song:
    table = 'Music'
    key = 'title'
    artist = Field()
    year = Field()
    genre = Field()

    def __init__(self, key):
        self.key = key
```

Para usar los modelos, primera conéctate a la base de datos:

```
>>> import sqlite3
>>> conn = sqlite3.connect('entertainment.db')
```

Una sesión interactiva muestra cómo los datos son obtenidos desde la base de datos y cómo se pueden actualizar:

```
>>> Movie('Star Wars').director
'George Lucas'
>>> jaws = Movie('Jaws')
>>> f'Released in {jaws.year} by {jaws.director}'
'Released in 1975 by Steven Spielberg'

>>> Song('Country Roads').artist
'John Denver'

>>> Movie('Star Wars').director = 'J.J. Abrams'
>>> Movie('Star Wars').director
'J.J. Abrams'
```

Equivalentes en Python puro

El protocolo de descriptores es simple y ofrece posibilidades estimulantes. Varios casos de uso son tan comunes que han sido pre-empaquetados en herramientas incorporadas. Propiedades, métodos vinculados, métodos estáticos, métodos de clase y _slots_ están todos basados en el protocolo de descriptores.

Propiedades

Llamar a **`property()`** es una forma sucinta de construir un descriptor de datos que desencadena llamadas a funciones al acceder a un atributo. Su firma es:

```
property(fget=None, fset=None, fdel=None, doc=None) -> p
```

La documentación muestra un uso típico para definir un atributo gestionado `x`:

```
class C:
    def getx(self): return self.__x
    def setx(self, value): self.__x = value
    def delx(self): del self.__x
    x = property(getx, setx, delx, "I'm the 'x' property
```

To see how **`property()`** is implemented in terms of the descriptor protocol, here is a pure Python equivalent that implements most of the core functionality:

```
class Property:
    "Emulate PyProperty_Type() in Objects/descrobject.c"

    def __init__(self, fget=None, fset=None, fdel=None,
        self.fget = fget
        self.fset = fset
        self.fdel = fdel
        if doc is None and fget is not None:
            doc = fget.__doc__
        self.__doc__ = doc

    def __set_name__(self, owner, name):
        self.__name__ = name

    def __get__(self, obj, objtype=None):
        if obj is None:
            return self
        if self.fget is None:
            raise AttributeError
        return self.fget(obj)

    def __set__(self, obj, value):
        if self.fset is None:
            raise AttributeError
        self.fset(obj, value)

    def __delete__(self, obj):
        if self.fdel is None:
            raise AttributeError
        self.fdel(obj)

    def getter(self, fget):
        return type(self)(fget, self.fset, self.fdel, se

    def setter(self, fset):
        return type(self)(self.fget, fset, self.fdel, se

    def deleter(self, fdel):
        return type(self)(self.fget, self.fset, fdel, se
```

La función incorporada **`property()`** es de ayuda cuando una interfaz de usuario ha otorgado acceso a atributos y luego los cambios posteriores requieren la intervención de un método.

Por ejemplo, una clase de hoja de cálculo puede otorgar acceso al valor de una celda a través de `Cell('b10').value`. Las mejoras posteriores del programa requieren que la celda se vuelva a calcular en cada acceso; sin embargo, la programadora no quiere afectar al código de cliente existente que accede al atributo directamente. La solución es envolver el acceso al valor del atributo en un descriptor de datos propiedad:

```
class Cell:
    ...

    @property
    def value(self):
        "Recalculate the cell before returning value"
        self.recalc()
        return self._value
```

Either the built-in `property()` or our **`Property()`** equivalent would work in this example.

Funciones y métodos

Las características orientadas a objetos de Python se basan en un entorno basado en funciones. Usando descriptores de no-datos, ambas se combinan perfectamente.

Las funciones almacenadas en diccionarios de clase son convertidas en métodos cuando son invocadas. Los métodos sólo difieren de funciones regulares en que la instancia del objeto es antepuesta a los otros argumentos. Por convención, la instancia se llama *self*, pero podría ser llamada *this* o cualquier otro nombre de variable.

Los métodos se pueden crear manualmente con `types.MethodType`, lo que es aproximadamente equivalente a:

```
class MethodType:
    "Emulate PyMethod_Type in Objects/classobject.c"

    def __init__(self, func, obj):
        self.__func__ = func
        self.__self__ = obj
    def __call__(self, *args, **kwargs):
        func = self.__func__
        obj = self.__self__
        return func(obj, *args, **kwargs)

    def __getattribute__(self, name):
        "Emulate method_getset() in Objects/classobject.
        if name == '__doc__':
            return self.__func__.__doc__
        return object.__getattribute__(self, name)

    def __getattr__(self, name):
        "Emulate method_getattro() in Objects/classobjec
        return getattr(self.__func__, name)

    def __get__(self, obj, objtype=None):
        "Emulate method_descr_get() in Objects/classobje
        return self
```

To support automatic creation of methods, functions include the `__get__()` method for binding methods during attribute access. This means that functions are non-data descriptors that return bound methods during dotted lookup from an instance. Here’s how it works:

```
class Function:
    ...

    def __get__(self, obj, objtype=None):
        "Simulate func_descr_get() in Objects/funcobject
        if obj is None:
            return self
        return MethodType(self, obj)
```

Ejecutar la siguiente clase en el intérprete muestra cómo funciona el descriptor de función en la práctica:

```
class D:
    def f(self):
        return self

class D2:
    pass
```

La función tiene un atributo de nombre calificado para soportar introspección:

```
>>> D.f.__qualname__
'D.f'
```

Accessing the function through the class dictionary does not invoke `__get__()`. Instead, it just returns the underlying function object:

```
>>> D.__dict__['f']
<function D.f at 0x00C45070>
```

Dotted access from a class calls `__get__()` which just returns the underlying function unchanged:

```
>>> D.f
<function D.f at 0x00C45070>
```

The interesting behavior occurs during dotted access from an instance. The dotted lookup calls `__get__()` which returns a bound method object:

```
>>> d = D()
>>> d.f
<bound method D.f of <__main__.D object at 0x00B18C90>>
```

Internamente, el método vinculado guarda la función subyacente y la instancia vinculada:

```
>>> d.f.__func__
<function D.f at 0x00C45070>

>>> d.f.__self__
<__main__.D object at 0x00B18C90>
```

Si alguna vez te preguntaste de dónde viene *self* en métodos regulares, o de dónde viene *cls* en métodos de clase, ¡es acá!

Tipos de métodos

Los descriptores de no-datos proporcionan un mecanismo simple para variaciones de los patrones habituales para vincular funciones en métodos.

To recap, functions have a `__get__()` method so that they can be converted to a method when accessed as attributes. The non-data descriptor transforms an `obj.f(*args)` call into `f(obj, *args)`. Calling `cls.f(*args)` becomes `f(*args)`.

Este cuadro resume el enlace (*binding*) y sus dos variantes más útiles:

Transformación	**Llamado desde un objeto**	**Llamado desde una clase**
función	f(obj, *args)	f(*args)
método estático	f(*args)	f(*args)
método de clase	f(type(obj), *args)	f(cls, *args)

Métodos estáticos

Los métodos estáticos retornan la función subyacente sin cambios. Llamar a `c.f` o `C.f` es equivalente a una búsqueda directa en `object.__getattribute__(c, "f")` o en `object.__getattribute__(C, "f")`. Como resultado, la función se vuelve idénticamente accesible desde un objeto o una clase.

Buenos candidatos para ser métodos estáticos son los métodos que no hacen referencia a la variable `self`.

For instance, a statistics package may include a container class for experimental data. The class provides normal methods for computing the average, mean, median, and other descriptive statistics that depend on the data. However, there may be useful functions which are conceptually related but do not depend on the data. For instance, `erf(x)` is handy conversion routine that comes up in statistical work but does not directly depend on a particular dataset. It can be called either from an object or the class:

`s.erf(1.5) --> 0.9332` or `Sample.erf(1.5) --> 0.9332`.

Dado que los métodos estáticos retornan la función subyacente sin cambios, las llamadas de ejemplo carecen de interés:

```
class E:
    @staticmethod
    def f(x):
        return x * 10

>>> E.f(3)
30
>>> E().f(3)
30
```

Usando el protocolo de descriptores de no-datos, una versión pura de Python de **`staticmethod()`** se vería así:

```
import functools

class StaticMethod:
    "Emulate PyStaticMethod_Type() in Objects/funcobject

    def __init__(self, f):
        self.f = f
        functools.update_wrapper(self, f)

    def __get__(self, obj, objtype=None):
        return self.f

    def __call__(self, *args, **kwds):
        return self.f(*args, **kwds)

    @property
    def __annotations__(self):
        return self.f.__annotations__
```

The **`functools.update_wrapper()`** call adds a `__wrapped__` attribute that refers to the underlying function. Also it carries forward the attributes necessary to make the wrapper look like the wrapped function, including `__name__`, `__qualname__`, and `__doc__`.

Métodos de clase

A diferencia de los métodos estáticos, los métodos de clase anteponen la referencia de clase a la lista de argumentos antes de llamar a la función. Este formato es el mismo si quien llama es un objeto o una clase:

```
class F:
    @classmethod
    def f(cls, x):
        return cls.__name__, x

>>> F.f(3)
('F', 3)
>>> F().f(3)
('F', 3)
```

Este comportamiento es útil siempre que la función solo necesite tener una referencia de clase y no necesita contar con los datos almacenados en una instancia específica. Un uso de los métodos de clase es crear constructores de clase alternativos. Por ejemplo, el método de clase `dict.fromkeys()` crea un nuevo diccionario a partir de una lista de claves. El equivalente puro de Python es:

```
class Dict(dict):
    @classmethod
    def fromkeys(cls, iterable, value=None):
        "Emulate dict_fromkeys() in Objects/dictobject.c
        d = cls()
        for key in iterable:
            d[key] = value
        return d
```

Ahora se puede construir un nuevo diccionario de claves únicas así:

```
>>> d = Dict.fromkeys('abracadabra')
>>> type(d) is Dict
```

```
True
>>> d
{'a': None, 'b': None, 'r': None, 'c': None, 'd': None}
```

Usando el protocolo de descriptores de no-datos, una implementación pura en Python de **`classmethod()`** se vería así:

```
import functools

class ClassMethod:
    "Emulate PyClassMethod_Type() in Objects/funcobject.

    def __init__(self, f):
        self.f = f
        functools.update_wrapper(self, f)

    def __get__(self, obj, cls=None):
        if cls is None:
            cls = type(obj)
        return MethodType(self.f, cls)
```

The **`functools.update_wrapper()`** call in `ClassMethod` adds a `__wrapped__` attribute that refers to the underlying function. Also it carries forward the attributes necessary to make the wrapper look like the wrapped function: **`__name__`**, **`__qualname__`**, **`__doc__`**, and **`__annotations__`**.

Objetos miembros y _slots_

Cuando una clase define `__slots__`, reemplaza los diccionarios de instancia por un arreglo de valores de ranura de largo fijo. Desde el punto de vista del usuario esto tiene varios efectos:

1. Provides immediate detection of bugs due to misspelled attribute assignments. Only attribute names specified in `__slots__` are allowed:

```
class Vehicle:
    __slots__ = ('id_number', 'make', 'model')
```

```
>>> auto = Vehicle()
>>> auto.id_nubmer = 'VYE483814LQEX'
Traceback (most recent call last):
    ...
AttributeError: 'Vehicle' object has no attribute 'id_nu
```

2. Helps create immutable objects where descriptors manage access to private attributes stored in `__slots__`:

```
class Immutable:

    __slots__ = ('_dept', '_name')          # Replace th

    def __init__(self, dept, name):
        self._dept = dept                   # Store to p
        self._name = name                   # Store to p

    @property                               # Read-only
    def dept(self):
        return self._dept

    @property
    def name(self):                         # Read-only
        return self._name

>>> mark = Immutable('Botany', 'Mark Watney')
>>> mark.dept
'Botany'
>>> mark.dept = 'Space Pirate'
Traceback (most recent call last):
    ...
AttributeError: property 'dept' of 'Immutable' object ha
>>> mark.location = 'Mars'
Traceback (most recent call last):
    ...
AttributeError: 'Immutable' object has no attribute 'loc
```

3. Saves memory. On a 64-bit Linux build, an instance with two attributes takes 48 bytes with `__slots__` and 152 bytes without.

This flyweight design pattern [https://en.wikipedia.org/wiki/Flyweight_pattern] likely only matters when a large number of instances are going to be created.

4. Improves speed. Reading instance variables is 35% faster with `__slots__` (as measured with Python 3.10 on an Apple M1 processor).

5. Blocks tools like `functools.cached_property()` which require an instance dictionary to function correctly:

```
from functools import cached_property

class CP:
    __slots__ = ()                          # Eliminates

    @cached_property                        # Requires a
    def pi(self):
        return 4 * sum((-1.0)**n / (2.0*n + 1.0)
                       for n in reversed(range(100_000))

>>> CP().pi
Traceback (most recent call last):
  ...
TypeError: No '__dict__' attribute on 'CP' instance to c
```

No es posible crear una versión exacta de `__slots__` en Python puro porque requiere acceso directo a estructuras en C y control sobre asignación de memoria de objetos. Sin embargo podemos construir una simulación casi totalmente fiel donde la estructura real en C para las ranuras es emulada con una lista privada `_slotvalues`. Las lecturas y escrituras de esta estructura privada se manejan con descriptores miembros:

```
null = object()

class Member:

    def __init__(self, name, clsname, offset):
        'Emulate PyMemberDef in Include/structmember.h'
        # Also see descr_new() in Objects/descrobject.c
        self.name = name
        self.clsname = clsname
        self.offset = offset

    def __get__(self, obj, objtype=None):
        'Emulate member_get() in Objects/descrobject.c'
        # Also see PyMember_GetOne() in Python/structmem
        if obj is None:
            return self
        value = obj._slotvalues[self.offset]
        if value is null:
            raise AttributeError(self.name)
        return value

    def __set__(self, obj, value):
        'Emulate member_set() in Objects/descrobject.c'
        obj._slotvalues[self.offset] = value

    def __delete__(self, obj):
        'Emulate member_delete() in Objects/descrobject.
        value = obj._slotvalues[self.offset]
        if value is null:
            raise AttributeError(self.name)
        obj._slotvalues[self.offset] = null

    def __repr__(self):
        'Emulate member_repr() in Objects/descrobject.c'
        return f'<Member {self.name!r} of {self.clsname!
```

The **`type.__new__()`** method takes care of adding member objects to class variables:

```
class Type(type):
    'Simulate how the type metaclass adds member objects

    def __new__(mcls, clsname, bases, mapping, **kwargs)
        'Emulate type_new() in Objects/typeobject.c'
        # type_new() calls PyTypeReady() which calls ad
        slot_names = mapping.get('slot_names', [])
        for offset, name in enumerate(slot_names):
            mapping[name] = Member(name, clsname, offset
        return type.__new__(mcls, clsname, bases, mappin
```

El método **`object.__new__()`** se hace cargo de crear instancias que tienen ranuras en vez un diccionario de instancia. Acá hay una simulación aproximada en Python puro:

```
class Object:
    'Simulate how object.__new__() allocates memory for

    def __new__(cls, *args, **kwargs):
        'Emulate object_new() in Objects/typeobject.c'
        inst = super().__new__(cls)
        if hasattr(cls, 'slot_names'):
            empty_slots = [null] * len(cls.slot_names)
            object.__setattr__(inst, '_slotvalues', empt
        return inst

    def __setattr__(self, name, value):
        'Emulate _PyObject_GenericSetAttrWithDict() Obje
        cls = type(self)
        if hasattr(cls, 'slot_names') and name not in cl
            raise AttributeError(
                f'{cls.__name__!r} object has no attribu
            )
        super().__setattr__(name, value)

    def __delattr__(self, name):
        'Emulate _PyObject_GenericSetAttrWithDict() Obje
        cls = type(self)
        if hasattr(cls, 'slot_names') and name not in cl
            raise AttributeError(
                f'{cls.__name__!r} object has no attribu
            )
        super().__delattr__(name)
```

To use the simulation in a real class, just inherit from **`Object`** and set the metaclass to **`Type`**:

```
class H(Object, metaclass=Type):
    'Instance variables stored in slots'

    slot_names = ['x', 'y']

    def __init__(self, x, y):
        self.x = x
        self.y = y
```

En este punto, la metaclase ha cargado los objetos miembros para *x* e *y*:

```
>>> from pprint import pp
>>> pp(dict(vars(H)))
{'__module__': '__main__',
 '__doc__': 'Instance variables stored in slots',
 'slot_names': ['x', 'y'],
 '__init__': <function H.__init__ at 0x7fb5d302f9d0>,
 'x': <Member 'x' of 'H'>,
 'y': <Member 'y' of 'H'>}
```

Cuando se crean instancias, éstas tienen una lista `slot_values` donde se almacenan los atributos:

```
>>> h = H(10, 20)
>>> vars(h)
{'_slotvalues': [10, 20]}
>>> h.x = 55
>>> vars(h)
{'_slotvalues': [55, 20]}
```

Atributos mal deletreados o no asignados lazarán una excepción:

```
>>> h.xz
Traceback (most recent call last):
    ...
AttributeError: 'H' object has no attribute 'xz'
```

Debugging C API extensions and CPython Internals with GDB

This document explains how the Python GDB extension, `python-gdb.py`, can be used with the GDB debugger to debug CPython extensions and the CPython interpreter itself.

When debugging low-level problems such as crashes or deadlocks, a low-level debugger, such as GDB, is useful to diagnose and correct the issue. By default, GDB (or any of its front-ends) doesn't support high-level information specific to the CPython interpreter.

The `python-gdb.py` extension adds CPython interpreter information to GDB. The extension helps introspect the stack of currently executing Python functions. Given a Python object represented by a PyObject* pointer, the extension surfaces the type and value of the object.

Developers who are working on CPython extensions or tinkering with parts of CPython that are written in C can use this document to learn how to use the `python-gdb.py` extension with GDB.

Nota

This document assumes that you are familiar with the basics of GDB and the CPython C API. It consolidates guidance from the devguide [https://devguide.python.org] and the Python wiki [https://wiki.python.org/moin/DebuggingWithGdb].

Prerequisites

You need to have:

- GDB 7 or later. (For earlier versions of GDB, see `Misc/gdbinit` in the sources of Python 3.11 or earlier.)
- GDB-compatible debugging information for Python and any extension you are debugging.
- The `python-gdb.py` extension.

The extension is built with Python, but might be distributed separately or not at all. Below, we include tips for a few common systems as examples. Note that even if the instructions match your system, they might be outdated.

Setup with Python built from source

When you build CPython from source, debugging information should be available, and the build should add a `python-gdb.py` file to the root directory of your repository.

To activate support, you must add the directory containing `python-gdb.py` to GDB's «auto-load-safe-path». If you haven't done this, recent versions of GDB will print out a warning with instructions on how to do this.

Nota

If you do not see instructions for your version of GDB, put this in your configuration file (`~/.gdbinit` or `~/.config/gdb/gdbinit`):

```
add-auto-load-safe-path /path/to/cpython
```

You can also add multiple paths, separated by `:`.

Setup for Python from a Linux distro

Most Linux systems provide debug information for the system Python in a package called `python-debuginfo`, `python-dbg` or similar. For example:

- Fedora:

```
sudo dnf install gdb
sudo dnf debuginfo-install python3
```

- Ubuntu:

```
sudo apt install gdb python3-dbg
```

On several recent Linux systems, GDB can download debugging symbols automatically using *debuginfod*. However, this will not install the `python-gdb.py` extension; you generally do need to install the debug info package separately.

Using the Debug build and Development mode

For easier debugging, you might want to:

- Use a debug build of Python. (When building from source, use `configure --with-pydebug`. On Linux distros, install and run a package like `python-debug` or `python-dbg`, if available.)
- Use the runtime development mode (`-X dev`).

Both enable extra assertions and disable some optimizations. Sometimes this hides the bug you are trying to find, but in most cases they make the process easier.

Using the `python-gdb` extension

When the extension is loaded, it provides two main features: pretty printers for Python values, and additional commands.

Pretty-printers

This is what a GDB backtrace looks like (truncated) when this extension is enabled:

```
#0  0x000000000041a6b1 in PyObject_Malloc (nbytes=Cannot
) at Objects/obmalloc.c:748
```

```
#1  0x000000000041b7c0 in _PyObject_DebugMallocApi (id=1
#2  0x000000000041b717 in _PyObject_DebugMalloc (nbytes=
#3  0x000000000044060a in _PyUnicode_New (length=11) at 
#4  0x00000000004466aa in PyUnicodeUCS2_DecodeUTF8Statef
    0x0) at Objects/unicodeobject.c:2531
#5  0x0000000000446647 in PyUnicodeUCS2_DecodeUTF8 (s=0x
    at Objects/unicodeobject.c:2495
#6  0x0000000000440d1b in PyUnicodeUCS2_FromStringAndSiz
    at Objects/unicodeobject.c:551
#7  0x0000000000440d94 in PyUnicodeUCS2_FromString (u=0x
#8  0x0000000000584abd in PyDict_GetItemString (v=
    {'Yuck': <type at remote 0xad4730>, '__builtins__': 
    0x5c2b8d "__lltrace__") at Objects/dictobject.c:2171
```

Notice how the dictionary argument to `PyDict_GetItemString` is displayed as its `repr()`, rather than an opaque `PyObject *` pointer.

The extension works by supplying a custom printing routine for values of type `PyObject *`. If you need to access lower-level details of an object, then cast the value to a pointer of the appropriate type. For example:

```
(gdb) p globals
$1 = {'__builtins__': <module at remote 0x7ffff7fb1868>,
'__main__', 'ctypes': <module at remote 0x7ffff7f14360>,
'__package__': None}

(gdb) p *(PyDictObject*)globals
$2 = {ob_refcnt = 3, ob_type = 0x3dbdf85820, ma_fill = 5
ma_mask = 7, ma_table = 0x63d0f8, ma_lookup = 0x3dbdc7ea
<lookdict_string>, ma_smalltable = {{me_hash = 706518619
me_key = '__builtins__', me_value = <module at remote 0x
{me_hash = -368181376027291943, me_key = '__name__',
me_value ='__main__'}, {me_hash = 0, me_key = 0x0, me_va
{me_hash = 0, me_key = 0x0, me_value = 0x0},
{me_hash = -9177857982131165996, me_key = 'ctypes',
me_value = <module at remote 0x7ffff7f14360>},
{me_hash = -8518757509529533123, me_key = '__doc__', me_
```

```
{me_hash = 0, me_key = 0x0, me_value = 0x0}, {
  me_hash = 6614918939584953775, me_key = '__package__',
```

Note that the pretty-printers do not actually call `repr()`. For basic types, they try to match its result closely.

An area that can be confusing is that the custom printer for some types look a lot like GDB's built-in printer for standard types. For example, the pretty-printer for a Python `int` (PyLongObject*) gives a representation that is not distinguishable from one of a regular machine-level integer:

```
(gdb) p some_machine_integer
$3 = 42

(gdb) p some_python_integer
$4 = 42
```

The internal structure can be revealed with a cast to PyLongObject*:

```
(gdb) p *(PyLongObject*)some_python_integer
$5 = {ob_base = {ob_base = {ob_refcnt = 8, ob_type = 0x3
ob_digit = {42}}
```

A similar confusion can arise with the `str` type, where the output looks a lot like gdb's built-in printer for `char *`:

```
(gdb) p ptr_to_python_str
$6 = '__builtins__'
```

The pretty-printer for `str` instances defaults to using single-quotes (as does Python's `repr` for strings) whereas the standard printer for `char *` values uses double-quotes and contains a hexadecimal address:

```
(gdb) p ptr_to_char_star
$7 = 0x6d72c0 "hello world"
```

Again, the implementation details can be revealed with a cast to PyUnicodeObject*:

```
(gdb) p *(PyUnicodeObject*)$6
$8 = {ob_base = {ob_refcnt = 33, ob_type = 0x3dad3a95a0}
str = 0x7ffff2128500, hash = 7065186196740147912, state
```

py-list

The extension adds a `py-list` command, which lists the Python source code (if any) for the current frame in the selected thread. The current line is marked with a «>»:

```
(gdb) py-list
 901        if options.profile:
 902            options.profile = False
 903            profile_me()
 904            return
 905
>906        u = UI()
 907        if not u.quit:
 908            try:
 909                gtk.main()
 910            except KeyboardInterrupt:
 911                # properly quit on a keyboard
```

Use `py-list START` to list at a different line number within the Python source, and `py-list START,END` to list a specific range of lines within the Python source.

`py-up` and `py-down`

The `py-up` and `py-down` commands are analogous to GDB's regular `up` and `down` commands, but try to move at the level of CPython frames, rather than C frames.

GDB is not always able to read the relevant frame information, depending on the optimization level with which CPython was compiled. Internally, the commands look for C frames that are executing the default frame evaluation function (that is, the core bytecode interpreter loop within CPython) and look up the value of the related `PyFrameObject *`.

They emit the frame number (at the C level) within the thread.

For example:

```
(gdb) py-up
#37 Frame 0x9420b04, for file /usr/lib/python2.6/s
gnome_sudoku/main.py, line 906, in start_game ()
    u = UI()
(gdb) py-up
#40 Frame 0x948e82c, for file /usr/lib/python2.6/s
gnome_sudoku/gnome_sudoku.py, line 22, in start_ga
    main.start_game()
(gdb) py-up
Unable to find an older python frame
```

so we’re at the top of the Python stack.

The frame numbers correspond to those displayed by GDB’s standard `backtrace` command. The command skips C frames which are not executing Python code.

Going back down:

```
(gdb) py-down
#37 Frame 0x9420b04, for file /usr/lib/python2.6/s
    u = UI()
(gdb) py-down
#34 (unable to read python frame information)
(gdb) py-down
#23 (unable to read python frame information)
(gdb) py-down
#19 (unable to read python frame information)
(gdb) py-down
#14 Frame 0x99262ac, for file /usr/lib/python2.6/s
```

```
            swallower.run_dialog(self.dialog)
(gdb) py-down
#11 Frame 0x9aead74, for file /usr/lib/python2.6/s
            gtk.main()
(gdb) py-down
#8 (unable to read python frame information)
(gdb) py-down
Unable to find a newer python frame
```

and we're at the bottom of the Python stack.

Note that in Python 3.12 and newer, the same C stack frame can be used for multiple Python stack frames. This means that `py-up` and `py-down` may move multiple Python frames at once. For example:

```
(gdb) py-up
#6 Frame 0x7ffff7fb62b0, for file /tmp/rec.py, lin
   time.sleep(5)
#6 Frame 0x7ffff7fb6240, for file /tmp/rec.py, lin
   recursive_function(n-1)
#6 Frame 0x7ffff7fb61d0, for file /tmp/rec.py, lin
   recursive_function(n-1)
#6 Frame 0x7ffff7fb6160, for file /tmp/rec.py, lin
   recursive_function(n-1)
#6 Frame 0x7ffff7fb60f0, for file /tmp/rec.py, lin
   recursive_function(n-1)
#6 Frame 0x7ffff7fb6080, for file /tmp/rec.py, lin
   recursive_function(n-1)
#6 Frame 0x7ffff7fb6020, for file /tmp/rec.py, lin
   recursive_function(5)
(gdb) py-up
Unable to find an older python frame
```

py-bt

The `py-bt` command attempts to display a Python-level backtrace of the current thread.

For example:

```
(gdb) py-bt
#8 (unable to read python frame information)
#11 Frame 0x9aead74, for file /usr/lib/python2.6/s
            gtk.main()
#14 Frame 0x99262ac, for file /usr/lib/python2.6/s
            swallower.run_dialog(self.dialog)
#19 (unable to read python frame information)
#23 (unable to read python frame information)
#34 (unable to read python frame information)
#37 Frame 0x9420b04, for file /usr/lib/python2.6/s
    u = UI()
#40 Frame 0x948e82c, for file /usr/lib/python2.6/s
    main.start_game()
```

The frame numbers correspond to those displayed by GDB's standard `backtrace` command.

py-print

The `py-print` command looks up a Python name and tries to print it. It looks in locals within the current thread, then globals, then finally builtins:

```
(gdb) py-print self
local 'self' = <SwappableArea(running=<gtk.Dialog 
main_page=0) at remote 0x98fa6e4>
(gdb) py-print __name__
global '__name__' = 'gnome_sudoku.dialog_swallower
(gdb) py-print len
builtin 'len' = <built-in function len>
(gdb) py-print scarlet_pimpernel
'scarlet_pimpernel' not found
```

If the current C frame corresponds to multiple Python frames, `py-print` only considers the first one.

py-locals

The `py-locals` command looks up all Python locals within the current Python frame in the selected thread, and prints their representations:

```
(gdb) py-locals
self = <SwappableArea(running=<gtk.Dialog at remot
main_page=0) at remote 0x98fa6e4>
d = <gtk.Dialog at remote 0x98faaa4>
```

If the current C frame corresponds to multiple Python frames, locals from all of them will be shown:

```
(gdb) py-locals
Locals for recursive_function
n = 0
Locals for recursive_function
n = 1
Locals for recursive_function
n = 2
Locals for recursive_function
n = 3
Locals for recursive_function
n = 4
Locals for recursive_function
n = 5
Locals for <module>
```

Use with GDB commands

The extension commands complement GDB’s built-in commands. For example, you can use a frame numbers shown by `py-bt` with the `frame` command to go a specific frame within the selected thread, like this:

```
(gdb) py-bt
(output snipped)
```

```
#68 Frame 0xaa4560, for file Lib/test/regrtest.py, line
        main()
(gdb) frame 68
#68 0x00000000004cd1e6 in PyEval_EvalFrameEx (f=Frame 0x
2665                            x = call_function(&sp, c
(gdb) py-list
1543        # Run the tests in a context manager that te
1544        # temporary and writable directory. If it's
1545        # change the CWD, the original CWD will be u
1546        # available from test_support.SAVEDCWD.
1547        with test_support.temp_cwd(TESTCWD, quiet=Tr
>1548            main()
```

The `info threads` command will give you a list of the threads within the process, and you can use the `thread` command to select a different one:

```
(gdb) info threads
  105 Thread 0x7fffefa18710 (LWP 10260)  sem_wait () at
  104 Thread 0x7fffdf5fe710 (LWP 10259)  sem_wait () at
* 1 Thread 0x7ffff7fe2700 (LWP 10145)  0x00000038e46d73e
```

You can use `thread apply all COMMAND` or (`t a a COMMAND` for short) to run a command on all threads. With `py-bt`, this lets you see what every thread is doing at the Python level:

```
(gdb) t a a py-bt

Thread 105 (Thread 0x7fffefa18710 (LWP 10260)):
#5 Frame 0x7fffd00019d0, for file /home/david/coding/pyt
        self.__block.acquire()
#8 Frame 0x7fffac001640, for file /home/david/coding/pyt
            self._acquire_restore(saved_state)
#12 Frame 0x7fffb8001a10, for file /home/david/coding/py
            cond.wait()
#16 Frame 0x7fffb8001c40, for file /home/david/coding/py
                f()

Thread 104 (Thread 0x7fffdf5fe710 (LWP 10259)):
```

```
#5 Frame 0x7fffe4001580, for file /home/david/coding/pyt
        self.__block.acquire()
#8 Frame 0x7fffc8002090, for file /home/david/coding/pyt
            self._acquire_restore(saved_state)
#12 Frame 0x7fffac001c90, for file /home/david/coding/py
            cond.wait()
#16 Frame 0x7fffac0011c0, for file /home/david/coding/py
                f()

Thread 1 (Thread 0x7ffff7fe2700 (LWP 10145)):
#5 Frame 0xcb5380, for file /home/david/coding/python-sv
    time.sleep(0.01)
#8 Frame 0x7fffd00024a0, for file /home/david/coding/pyt
        _wait()
```

HOWTO - Enum

Un **`Enum`** es un conjunto de nombres simbólicos vinculados a valores únicos. Son similares a las variables globales, pero ofrecen un **`repr()`** más útil, agrupación, seguridad de tipos y algunas otras características.

Son más útiles cuando tiene una variable que puede tomar uno de una selección limitada de valores. Por ejemplo, los días de la semana:

```
>>> from enum import Enum
>>> class Weekday(Enum):
...     MONDAY = 1
...     TUESDAY = 2
...     WEDNESDAY = 3
...     THURSDAY = 4
...     FRIDAY = 5
...     SATURDAY = 6
...     SUNDAY = 7
```

O quizás los colores primarios RGB:

```
>>> from enum import Enum
>>> class Color(Enum):
...     RED = 1
...     GREEN = 2
...     BLUE = 3
```

Como puede ver, crear un **`Enum`** es tan simple como escribir una clase que herede del propio **`Enum`**.

Nota

Caso de miembros de Enum

> Dado que las enumeraciones se utilizan para representar constantes y para evitar problemas con conflictos de nombres entre los métodos/atributos de las clases mixin y los nombres de las enumeraciones, recomendamos encarecidamente utilizar nombres en MAYÚSCULAS para los miembros, y utilizaremos ese estilo en nuestros ejemplos.

Dependiendo de la naturaleza de la enumeración, el valor de un miembro puede o no ser importante, pero de cualquier manera ese valor puede usarse para obtener el miembro correspondiente:

```
>>> Weekday(3)
<Weekday.WEDNESDAY: 3>
```

Como puede ver, el `repr()` de un miembro muestra el nombre de enumeración, el nombre del miembro y el valor. El `str()` de un miembro muestra solo el nombre de enumeración y el nombre del miembro:

```
>>> print(Weekday.THURSDAY)
Weekday.THURSDAY
```

El *type* de un miembro de la enumeración es la enumeración a la que pertenece:

```
>>> type(Weekday.MONDAY)
<enum 'Weekday'>
>>> isinstance(Weekday.FRIDAY, Weekday)
True
```

Enum members have an attribute that contains just their **`name`**:

```
>>> print(Weekday.TUESDAY.name)
TUESDAY
```

Likewise, they have an attribute for their **`value`**:

```
>>> Weekday.WEDNESDAY.value
3
```

Unlike many languages that treat enumerations solely as name/

value pairs, Python Enums can have behavior added. For example, `datetime.date` has two methods for returning the weekday: `weekday()` and `isoweekday()`. The difference is that one of them counts from 0-6 and the other from 1-7. Rather than keep track of that ourselves we can add a method to the **`Weekday`** enum to extract the day from the **`date`** instance and return the matching enum member:

```
@classmethod
def from_date(cls, date):
    return cls(date.isoweekday())
```

The complete **`Weekday`** enum now looks like this:

```
>>> class Weekday(Enum):
...     MONDAY = 1
...     TUESDAY = 2
...     WEDNESDAY = 3
...     THURSDAY = 4
...     FRIDAY = 5
...     SATURDAY = 6
...     SUNDAY = 7
...     #
...     @classmethod
...     def from_date(cls, date):
...         return cls(date.isoweekday())
```

¡Ahora podemos averiguar qué día de la semana es hoy! Observe:

```
>>> import datetime as dt
>>> Weekday.from_date(dt.date.today())
<Weekday.TUESDAY: 2>
```

Por supuesto, si estás leyendo esto en otro día, verás ese día en su lugar.

This **`Weekday`** enum is great if our variable only needs one day, but what if we need several? Maybe we're writing a function to plot chores during a week, and don't want to use a `list` – we could use a different type of `Enum`:

```
>>> from enum import Flag
>>> class Weekday(Flag):
...     MONDAY = 1
...     TUESDAY = 2
...     WEDNESDAY = 4
...     THURSDAY = 8
...     FRIDAY = 16
...     SATURDAY = 32
...     SUNDAY = 64
```

Hemos cambiado dos cosas: somos heredados de `Flag` y los valores son todos potencia de 2.

Just like the original `Weekday` enum above, we can have a single selection:

```
>>> first_week_day = Weekday.MONDAY
>>> first_week_day
<Weekday.MONDAY: 1>
```

Pero `Flag` también nos permite combinar varios miembros en una sola variable:

```
>>> weekend = Weekday.SATURDAY | Weekday.SUNDAY
>>> weekend
<Weekday.SATURDAY|SUNDAY: 96>
```

Incluso puede iterar sobre una variable `Flag`:

```
>>> for day in weekend:
...     print(day)
Weekday.SATURDAY
Weekday.SUNDAY
```

Bien, preparemos algunas tareas:

```
>>> chores_for_ethan = {
...     'feed the cat': Weekday.MONDAY | Weekday.WEDNESD
...     'do the dishes': Weekday.TUESDAY | Weekday.THURS
...     'answer SO questions': Weekday.SATURDAY,
...     }
```

Y una función para mostrar las tareas de un día determinado:

```
>>> def show_chores(chores, day):
...     for chore, days in chores.items():
...         if day in days:
...             print(chore)
...
>>> show_chores(chores_for_ethan, Weekday.SATURDAY)
answer SO questions
```

En los casos en que los valores reales de los miembros no importen, puede ahorrarse algo de trabajo y usar **`auto()`** para los valores:

```
>>> from enum import auto
>>> class Weekday(Flag):
...     MONDAY = auto()
...     TUESDAY = auto()
...     WEDNESDAY = auto()
...     THURSDAY = auto()
...     FRIDAY = auto()
...     SATURDAY = auto()
...     SUNDAY = auto()
...     WEEKEND = SATURDAY | SUNDAY
```

Acceso programático a los miembros de la enumeración y sus atributos

A veces es útil acceder a los miembros en las enumeraciones programáticamente (es decir, situaciones en las que `Color.RED` no funcionará porque no se conoce el color exacto en el momento de escribir el programa). `Enum` permite dicho acceso:

```
>>> Color(1)
<Color.RED: 1>
>>> Color(3)
<Color.BLUE: 3>
```

Si desea acceder a los miembros de la enumeración por *name*, use el acceso a elementos:

```
>>> Color['RED']
<Color.RED: 1>
>>> Color['GREEN']
<Color.GREEN: 2>
```

If you have an enum member and need its **`name`** or **`value`**:

```
>>> member = Color.RED
>>> member.name
'RED'
>>> member.value
1
```

Duplicar miembros y valores de enumeración

Tener dos miembros de enumeración con el mismo nombre no es válido:

```
>>> class Shape(Enum):
...     SQUARE = 2
...     SQUARE = 3
...
Traceback (most recent call last):
...
TypeError: 'SQUARE' already defined as 2
```

Sin embargo, un miembro de enumeración puede tener otros nombres asociados. Dadas dos entradas `A` y `B` con el mismo valor (y `A` definido primero), `B` es un alias para el miembro `A`. La búsqueda por valor del valor de `A` retornará el miembro `A`. La búsqueda por nombre de `A` retornará el miembro `A`. La búsqueda por nombre de `B` también retornará el miembro `A`:

```
>>> class Shape(Enum):
...     SQUARE = 2
...     DIAMOND = 1
...     CIRCLE = 3
...     ALIAS_FOR_SQUARE = 2
...
>>> Shape.SQUARE
<Shape.SQUARE: 2>
>>> Shape.ALIAS_FOR_SQUARE
<Shape.SQUARE: 2>
>>> Shape(2)
<Shape.SQUARE: 2>
```

Nota

No está permitido intentar crear un miembro con el mismo nombre que un atributo ya definido (otro miembro, un método, etc.) o intentar crear un atributo con el mismo nombre que un miembro.

Garantizar valores de enumeración únicos

De forma predeterminada, las enumeraciones permiten múltiples nombres como alias para el mismo valor. Cuando no se desea este comportamiento, puede usar el decorador `unique()`:

```
>>> from enum import Enum, unique
>>> @unique
... class Mistake(Enum):
...     ONE = 1
...     TWO = 2
...     THREE = 3
...     FOUR = 3
...
Traceback (most recent call last):
...
ValueError: duplicate values found in <enum 'Mistake'>:
```

Uso de valores automáticos

Si el valor exacto no es importante, puede usar `auto`:

```
>>> from enum import Enum, auto
>>> class Color(Enum):
...     RED = auto()
...     BLUE = auto()
...     GREEN = auto()
...
>>> [member.value for member in Color]
[1, 2, 3]
```

The values are chosen by `_generate_next_value_()`, which can be overridden:

```
>>> class AutoName(Enum):
...     @staticmethod
...     def _generate_next_value_(name, start, count, la
...         return name
...
>>> class Ordinal(AutoName):
...     NORTH = auto()
...     SOUTH = auto()
...     EAST = auto()
...     WEST = auto()
...
>>> [member.value for member in Ordinal]
['NORTH', 'SOUTH', 'EAST', 'WEST']
```

Nota

The `_generate_next_value_()` method must be defined before any members.

Iteración

Iterar sobre los miembros de una enumeración no proporciona los alias:

```
>>> list(Shape)
[<Shape.SQUARE: 2>, <Shape.DIAMOND: 1>, <Shape.CIRCLE: 3
>>> list(Weekday)
[<Weekday.MONDAY: 1>, <Weekday.TUESDAY: 2>, <Weekday.WED
```

Note que los alias `Shape.ALIAS_FOR_SQUARE` y `Weekday.WEEKEND` no se muestran.

El atributo especial `__members__` es una asignación ordenada de solo lectura de nombres a miembros. Incluye todos los nombres definidos en la enumeración, incluidos los alias:

```
>>> for name, member in Shape.__members__.items():
...     name, member
...
('SQUARE', <Shape.SQUARE: 2>)
('DIAMOND', <Shape.DIAMOND: 1>)
('CIRCLE', <Shape.CIRCLE: 3>)
('ALIAS_FOR_SQUARE', <Shape.SQUARE: 2>)
```

El atributo `__members__` se puede utilizar para el acceso programático detallado a los miembros de la enumeración. Por ejemplo, encontrar todos los alias:

```
>>> [name for name, member in Shape.__members__.items()
['ALIAS_FOR_SQUARE']
```

Nota

Los alias para las banderas incluyen valores con múltiples banderas establecidas, como `3`, y ningún conjunto de banderas, es decir, `0`.

Comparaciones

Los miembros de la enumeración se comparan por identidad:

```
>>> Color.RED is Color.RED
True
```

```
>>> Color.RED is Color.BLUE
False
>>> Color.RED is not Color.BLUE
True
```

Las comparaciones ordenadas entre valores de enumeración son compatibles con *not*. Los miembros de la enumeración no son números enteros (pero consulte IntEnum a continuación):

```
>>> Color.RED < Color.BLUE
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'Color
```

Las comparaciones de igualdad se definen aunque:

```
>>> Color.BLUE == Color.RED
False
>>> Color.BLUE != Color.RED
True
>>> Color.BLUE == Color.BLUE
True
```

Las comparaciones con valores que no son de enumeración siempre comparan no iguales (nuevamente, **`IntEnum`** se diseñó explícitamente para comportarse de manera diferente, consulte a continuación):

```
>>> Color.BLUE == 2
False
```

Advertencia

Es posible recargar módulos; si un módulo recargado contiene enumeraciones, estas se crearán de nuevo y los nuevos miembros pueden no compararse como idénticos/iguales a los miembros originales.

Miembros permitidos y atributos de

enumeraciones

La mayoría de los ejemplos anteriores usan números enteros para los valores de enumeración. El uso de números enteros es corto y práctico (y proporcionado por defecto por el Functional API), pero no se aplica estrictamente. En la gran mayoría de los casos de uso, a uno no le importa cuál es el valor real de una enumeración. Pero si el valor *is* es importante, las enumeraciones pueden tener valores arbitrarios.

Las enumeraciones son clases de Python y pueden tener métodos y métodos especiales como de costumbre. Si tenemos esta enumeración:

```
>>> class Mood(Enum):
...     FUNKY = 1
...     HAPPY = 3
...
...     def describe(self):
...         # self es el miembro aquí
...         return self.name, self.value
...
...     def __str__(self):
...         return 'my custom str! {0}'.format(self.valu
...
...     @classmethod
...     def favorite_mood(cls):
...         # cls es la enumeración aquí
...         return cls.HAPPY
...
```

Después:

```
>>> Mood.favorite_mood()
<Mood.HAPPY: 3>
>>> Mood.HAPPY.describe()
('HAPPY', 3)
>>> str(Mood.FUNKY)
'my custom str! 1'
```

The rules for what is allowed are as follows: names that start and end with a single underscore are reserved by enum and cannot be used; all other attributes defined within an enumeration will become members of this enumeration, with the exception of special methods (`__str__()`, `__add__()`, etc.), descriptors (methods are also descriptors), and variable names listed in `_ignore_`.

Note: if your enumeration defines `__new__()` and/or `__init__()`, any value(s) given to the enum member will be passed into those methods. See Planet for an example.

Nota

The `__new__()` method, if defined, is used during creation of the Enum members; it is then replaced by Enum's `__new__()` which is used after class creation for lookup of existing members. See When to use _new_() vs. _init_() for more details.

Subclases de Enum restringidas

Una nueva clase `Enum` debe tener una clase de enumeración base, hasta un tipo de datos concreto y tantas clases mixtas basadas en `object` como sea necesario. El orden de estas clases base es:

```
class EnumName([mix-in, ...,] [data-type,] base-enum):
    pass
```

Además, la subclasificación de una enumeración solo se permite si la enumeración no define ningún miembro. Así que esto está prohibido:

```
>>> class MoreColor(Color):
...     PINK = 17
...
Traceback (most recent call last):
...
TypeError: <enum 'MoreColor'> cannot extend <enum 'Color
```

Pero esto está permitido:

```
>>> class Foo(Enum):
...     def some_behavior(self):
...         pass
...
>>> class Bar(Foo):
...     HAPPY = 1
...     SAD = 2
...
```

Permitir la subclasificación de enumeraciones que definen miembros conduciría a una violación de algunas invariantes importantes de tipos e instancias. Por otro lado, tiene sentido permitir compartir algún comportamiento común entre un grupo de enumeraciones. (Consulte OrderedEnum para ver un ejemplo).

Soporte de Dataclass

Cuando se hereda de una **`dataclass`**, el **`__repr__()`** omite el nombre de la clase heredada. Por ejemplo:

```
>>> from dataclasses import dataclass, field
>>> @dataclass
... class CreatureDataMixin:
...     size: str
...     legs: int
...     tail: bool = field(repr=False, default=True)
...
>>> class Creature(CreatureDataMixin, Enum):
...     BEETLE = 'small', 6
...     DOG = 'medium', 4
...
>>> Creature.DOG
<Creature.DOG: size='medium', legs=4>
```

Utilice el argumento `repr=False` de **`dataclass()`** para utilizar el **`repr()`** estándar.

Distinto en la versión 3.12: Solo se muestran los campos de la dataclass en el área de valores, no el nombre de la dataclass.

Nota

Añadir el decorador `dataclass()` a `Enum` y sus subclases no está soportado. No lanzará ningún error, pero producirá resultados bastante extraños en tiempo de ejecución, como miembros siendo iguales a otros:

```
>>> @dataclass               # no haga esto: no tiene r
... class Color(Enum):
...     RED = 1
...     BLUE = 2
...
>>> Color.RED is Color.BLUE
False
>>> Color.RED == Color.BLUE  # problema aquí:: no deber
True
```

Serialización (Pickling)

Las enumeraciones se pueden serializar y deserializar:

```
>>> from test.test_enum import Fruit
>>> from pickle import dumps, loads
>>> Fruit.TOMATO is loads(dumps(Fruit.TOMATO))
True
```

Se aplican las restricciones habituales para el *pickling*: las enumeraciones serializables deben definirse en el nivel superior de un módulo, ya que el decapado requiere que se puedan importar desde ese módulo.

Nota

Con la versión 4 del protocolo pickle es posible deserializar fácilmente enumeraciones anidadas en otras clases.

It is possible to modify how enum members are pickled/unpickled by defining `__reduce_ex__()` in the enumeration class. The default method is by-value, but enums with complicated values may want to use by-name:

```
>>> import enum
>>> class MyEnum(enum.Enum):
...     __reduce_ex__ = enum.pickle_by_enum_name
```

Nota

No se recomienda usar banderas por nombre , ya que los alias sin nombre no se desempaquetarán.

API funcional

Se puede llamar a la clase **Enum**, que proporciona la siguiente API funcional:

```
>>> Animal = Enum('Animal', 'ANT BEE CAT DOG')
>>> Animal
<enum 'Animal'>
>>> Animal.ANT
<Animal.ANT: 1>
>>> list(Animal)
[<Animal.ANT: 1>, <Animal.BEE: 2>, <Animal.CAT: 3>, <Ani
```

La semántica de esta API se asemeja a **namedtuple**. El primer argumento de la llamada a **Enum** es el nombre de la enumeración.

The second argument is the *source* of enumeration member names. It can be a whitespace-separated string of names, a sequence of names, a sequence of 2-tuples with key/value pairs, or a mapping (e.g. dictionary) of names to values. The last two options enable assigning arbitrary values to enumerations; the others auto-assign increasing integers starting with 1 (use the `start` parameter to specify a different starting value). A new class derived from **Enum** is returned. In other words, the above assignment to **Animal** is equivalent to:

```
>>> class Animal(Enum):
```

```
...     ANT = 1
...     BEE = 2
...     CAT = 3
...     DOG = 4
...
```

La razón por la que se toma por defecto `1` como el número inicial y no `0` es que `0` es `False` en un sentido booleano, pero por defecto todos los miembros de la enumeración se evalúan como `True`.

Deserializar las enumeraciones creadas con la API funcional puede ser complicado, ya que los detalles de implementación de la pila de marcos se usan para tratar de averiguar en qué módulo se está creando la enumeración (por ejemplo, fallará si usa una función de utilidad en un módulo separado, y también puede no trabajar en IronPython o Jython). La solución es especificar el nombre del módulo explícitamente de la siguiente manera:

```
>>> Animal = Enum('Animal', 'ANT BEE CAT DOG', module=__
```

Advertencia

Si no se proporciona `module` y Enum no puede determinar de qué se trata, los nuevos miembros de Enum no serán seleccionables; para mantener los errores más cerca de la fuente, se desactivará el decapado.

El nuevo protocolo pickle 4 también, en algunas circunstancias, depende de que **`__qualname__`** se establezca en la ubicación donde pickle podrá encontrar la clase. Por ejemplo, si la clase estuvo disponible en la clase SomeData en el ámbito global:

```
>>> Animal = Enum('Animal', 'ANT BEE CAT DOG', qualname=
```

La firma completa es:

```
Enum(
    value='NewEnumName',
    names=<...>,
    *,
module='...',
qualname='...',
type=<mixed-in class>,
start=1,
)
```

- *value*: Lo que la nueva clase de enumeración registrará como su nombre.
- *names*: Los miembros de la enumeración. Puede ser una cadena separada por comas o espacios en blanco (los valores comenzarán en 1 a menos que se especifique lo contrario):

  ```
  'RED GREEN BLUE' | 'RED,GREEN,BLUE' | 'RED, GREEN,
  ```

  o un iterador de nombres:

  ```
  ['RED', 'GREEN', 'BLUE']
  ```

  o un iterador de (nombre, valor) pares:

  ```
  [('CYAN', 4), ('MAGENTA', 5), ('YELLOW', 6)]
  ```

  o un mapeo:

  ```
  {'CHARTREUSE': 7, 'SEA_GREEN': 11, 'ROSEMARY': 42}
  ```

- *module*: nombre del módulo donde se puede encontrar la nueva clase de enumeración.
- *qualname*: donde en el módulo se puede encontrar la nueva clase de enumeración.
- *type*: tipo para mezclar en la nueva clase de enumeración.
- *start*: número para comenzar a contar si solo se pasan nombres.

Distinto en la versión 3.5: Se agregó el parámetro *start*.

Enumeraciones derivadas

IntEnum

La primera variación de `Enum` que se proporciona también es una subclase de `int`. Los miembros de un `IntEnum` se pueden comparar con números enteros; por extensión, las enumeraciones enteras de diferentes tipos también se pueden comparar entre sí:

```
>>> from enum import IntEnum
>>> class Shape(IntEnum):
...     CIRCLE = 1
...     SQUARE = 2
...
>>> class Request(IntEnum):
...     POST = 1
...     GET = 2
...
>>> Shape == 1
False
>>> Shape.CIRCLE == 1
True
>>> Shape.CIRCLE == Request.POST
True
```

Sin embargo, aún no se pueden comparar con las enumeraciones `Enum` estándar:

```
>>> class Shape(IntEnum):
...     CIRCLE = 1
...     SQUARE = 2
...
>>> class Color(Enum):
...     RED = 1
...     GREEN = 2
...
>>> Shape.CIRCLE == Color.RED
False
```

Los valores `IntEnum` se comportan como números enteros en otras formas que esperaría:

```
>>> int(Shape.CIRCLE)
1
>>> ['a', 'b', 'c'][Shape.CIRCLE]
'b'
>>> [i for i in range(Shape.SQUARE)]
[0, 1]
```

StrEnum

La segunda variación de `Enum` que se proporciona también es una subclase de `str`. Los miembros de un `StrEnum` se pueden comparar con cadenas; por extensión, las enumeraciones de cadenas de diferentes tipos también se pueden comparar entre sí.

Added in version 3.11.

IntFlag

La siguiente variación de `Enum` proporcionada, `IntFlag`, también se basa en `int`. La diferencia es que los miembros `IntFlag` se pueden combinar usando los operadores bit a bit (&, |, ^, ~) y el resultado sigue siendo un miembro `IntFlag`, si es posible. Al igual que `IntEnum`, los miembros `IntFlag` también son números enteros y se pueden utilizar siempre que se utilice un `int`.

Nota

Cualquier operación en un miembro `IntFlag` además de las operaciones bit a bit perderá la pertenencia a `IntFlag`.

Las operaciones bit a bit que den como resultado valores `IntFlag` no válidos perderán la pertenencia a `IntFlag`. Ver `FlagBoundary` para más detalles.

Added in version 3.6.

Distinto en la versión 3.11.

Ejemplo de clase `IntFlag`:

```
>>> from enum import IntFlag
>>> class Perm(IntFlag):
...     R = 4
...     W = 2
...     X = 1
...
>>> Perm.R | Perm.W
<Perm.R|W: 6>
>>> Perm.R + Perm.W
6
>>> RW = Perm.R | Perm.W
>>> Perm.R in RW
True
```

También es posible nombrar las combinaciones:

```
>>> class Perm(IntFlag):
...     R = 4
...     W = 2
...     X = 1
...     RWX = 7
...
>>> Perm.RWX
<Perm.RWX: 7>
>>> ~Perm.RWX
<Perm: 0>
>>> Perm(7)
<Perm.RWX: 7>
```

Nota

Las combinaciones con nombre se consideran alias. Los alias no aparecen durante la iteración, pero se pueden devolver a partir de búsquedas por valor.

Distinto en la versión 3.11.

Otra diferencia importante entre `IntFlag` y `Enum` es que si no se establecen banderas (el valor es 0), su evaluación booleana es

`False`:

```
>>> Perm.R & Perm.X
<Perm: 0>
>>> bool(Perm.R & Perm.X)
False
```

Debido a que los miembros `IntFlag` también son subclases de `int`, se pueden combinar con ellos (pero pueden perder la membresía **`IntFlag`**:

```
>>> Perm.X | 4
<Perm.R|X: 5>

>>> Perm.X + 8
9
```

Nota

El operador de negación, `~`, siempre retorna un miembro `IntFlag` con un valor positivo:

```
>>> (~Perm.X).value == (Perm.R|Perm.W).value == 6
True
```

Los miembros `IntFlag` también se pueden iterar sobre:

```
>>> list(RW)
[<Perm.R: 4>, <Perm.W: 2>]
```

Added in version 3.11.

Bandera

La última variación es `Flag`. Al igual que `IntFlag`, los miembros de **`Flag`** se pueden combinar mediante los operadores bit a bit (&, |, ^, ~). A diferencia de **`IntFlag`**, no se pueden combinar ni comparar con ninguna otra enumeración **`Flag`** ni con `int`. Si bien es posible especificar los valores directamente, se recomienda usar `auto` como valor y dejar que **`Flag`** seleccione un valor apropiado.

Added in version 3.6.

Al igual que `IntFlag`, si una combinación de miembros `Flag` da como resultado que no se establezcan indicadores, la evaluación booleana es `False`:

```
>>> from enum import Flag, auto
>>> class Color(Flag):
...     RED = auto()
...     BLUE = auto()
...     GREEN = auto()
...
>>> Color.RED & Color.GREEN
<Color: 0>
>>> bool(Color.RED & Color.GREEN)
False
```

Las banderas individuales deben tener valores que sean potencias de dos (1, 2, 4, 8, …), mientras que las combinaciones de banderas no:

```
>>> class Color(Flag):
...     RED = auto()
...     BLUE = auto()
...     GREEN = auto()
...     WHITE = RED | BLUE | GREEN
...
>>> Color.WHITE
<Color.WHITE: 7>
```

Dar un nombre a la condición «sin banderas establecidas» no cambia su valor booleano:

```
>>> class Color(Flag):
...     BLACK = 0
...     RED = auto()
...     BLUE = auto()
...     GREEN = auto()
...
>>> Color.BLACK
<Color.BLACK: 0>
```

```
>>> bool(Color.BLACK)
False
```

Los miembros `Flag` también se pueden iterar sobre:

```
>>> purple = Color.RED | Color.BLUE
>>> list(purple)
[<Color.RED: 1>, <Color.BLUE: 2>]
```

Added in version 3.11.

Nota

Para la mayoría del código nuevo, se recomienda encarecidamente `Enum` y `Flag`, ya que `IntEnum` y `IntFlag` rompen algunas promesas semánticas de una enumeración (al ser comparables con los números enteros y, por lo tanto, por la transitividad a otras enumeraciones no relacionadas). `IntEnum` y `IntFlag` deben usarse solo en los casos en que `Enum` y `Flag` no sirvan; por ejemplo, cuando las constantes enteras se reemplazan con enumeraciones, o para la interoperabilidad con otros sistemas.

Otros

Si bien `IntEnum` es parte del módulo `enum`, sería muy simple de implementar de forma independiente:

```
class IntEnum(int, ReprEnum):   # o Enum en lugar de Rep
    pass
```

This demonstrates how similar derived enumerations can be defined; for example a `FloatEnum` that mixes in `float` instead of `int`.

Algunas reglas:

1. Al subclasificar `Enum`, los tipos de combinación deben aparecer antes que `Enum` en la secuencia de bases, como en el ejemplo anterior de `IntEnum`.

2. Los tipos mixtos deben ser subclasificables. Por ejemplo, `bool` y `range` no son subclasificables y generarán un error durante la creación de Enum si se usan como tipo de combinación.
3. Si bien `Enum` puede tener miembros de cualquier tipo, una vez que mezcle un tipo adicional, todos los miembros deben tener valores de ese tipo, p. `int` anterior. Esta restricción no se aplica a los complementos que solo agregan métodos y no especifican otro tipo.
4. When another data type is mixed in, the `value` attribute is *not the same* as the enum member itself, although it is equivalent and will compare equal.
5. A `data type` is a mixin that defines `__new__()`, or a `dataclass`
6. %-style formatting: `%s` and `%r` call the `Enum` class's `__str__()` and `__repr__()` respectively; other codes (such as `%i` or `%h` for IntEnum) treat the enum member as its mixed-in type.
7. Formatted string literals, `str.format()`, and `format()` will use the enum's `__str__()` method.

Nota

Because `IntEnum`, `IntFlag`, and `StrEnum` are designed to be drop-in replacements for existing constants, their `__str__()` method has been reset to their data types" `__str__()` method.

When to use `__new__()` vs. `__init__()`

`__new__()` must be used whenever you want to customize the actual value of the `Enum` member. Any other modifications may go in either `__new__()` or `__init__()`, with `__init__()` being preferred.

Por ejemplo, si desea pasar varios elementos al constructor, pero solo desea que uno de ellos sea el valor:

```
>>> class Coordinate(bytes, Enum):
...     """
...     Coordenada con códigos binarios que pueden ser i
...     """
...     def __new__(cls, value, label, unit):
...         obj = bytes.__new__(cls, [value])
...         obj._value_ = value
...         obj.label = label
...         obj.unit = unit
...         return obj
...     PX = (0, 'P.X', 'km')
...     PY = (1, 'P.Y', 'km')
...     VX = (2, 'V.X', 'km/s')
...     VY = (3, 'V.Y', 'km/s')
...

>>> print(Coordinate['PY'])
Coordinate.PY

>>> print(Coordinate(3))
Coordinate.VY
```

Advertencia

No llame a `super().__new__()`, ya que encontrará el `__new__` de solo búsqueda; en su lugar, utilice directamente el tipo de datos.

Puntos más finos

Supported `__dunder__` and `_sunder_` names

The supported `__dunder__` and `_sunder_` names can be found in the Enum API documentation.

_Private__names

Private names no se convierten en miembros de enumeración, sino que siguen siendo atributos normales.

Distinto en la versión 3.11.

Tipo de miembro `Enum`

Los miembros de una enumeración son instancias de su clase de enumeración y se acceden normalmente como `EnumClass.member`. En ciertas situaciones, como al escribir comportamiento personalizado para una enumeración, es útil poder acceder a un miembro directamente desde otro, y esto está soportado; sin embargo, para evitar conflictos de nombres entre los nombres de los miembros y los atributos/métodos de las clases mezcladas, se recomienda encarecidamente utilizar nombres en mayúsculas.

Distinto en la versión 3.5.

Creación de miembros que se mezclan con otros tipos de datos

Al crear subclases de otros tipos de datos, como `int` o `str`, con un `Enum`, todos los valores después de `=` se pasan al constructor de ese tipo de datos. Por ejemplo:

```
>>> class MyEnum(IntEnum):      # help(int) -> int(x, ba
...     example = '11', 16      # so x='11' and base=16
...
>>> MyEnum.example.value        # and hex(11) is...
17
```

Valor booleano de clases y miembros `Enum`

Las clases de enumeración que se mezclan con tipos que no son `Enum` (como `int`, `str`, etc.) se evalúan de acuerdo con las reglas del tipo combinado; de lo contrario, todos los miembros se evalúan como `True`. Para hacer que la evaluación booleana de su propia enumeración dependa del valor del miembro, agregue lo siguiente a su clase:

```
def __bool__(self):
    return bool(self.value)
```

Las clases simples `Enum` siempre se evalúan como `True`.

Clases `Enum` con métodos

Si le da a su subclase de enumeración métodos adicionales, como la clase Planet a continuación, esos métodos aparecerán en un `dir()` del miembro, pero no de la clase:

```
>>> dir(Planet)
['EARTH', 'JUPITER', 'MARS', 'MERCURY', 'NEPTUNE', 'SATU
>>> dir(Planet.EARTH)
['__class__', '__doc__', '__module__', 'mass', 'name', '
```

Combinación de miembros de `Flag`

La iteración sobre una combinación de miembros `Flag` solo devolverá los miembros que se componen de un solo bit:

```
>>> class Color(Flag):
...     RED = auto()
...     GREEN = auto()
...     BLUE = auto()
...     MAGENTA = RED | BLUE
...     YELLOW = RED | GREEN
...     CYAN = GREEN | BLUE
...
>>> Color(3)  #combinación con nombre
<Color.YELLOW: 3>
>>> Color(7)      # combinación sin nombre
<Color.RED|GREEN|BLUE: 7>
```

Minuciosidades `Flag` y `IntFlag`

Usando el siguiente fragmento para nuestros ejemplos:

```
>>> class Color(IntFlag):
...     BLACK = 0
...     RED = 1
...     GREEN = 2
...     BLUE = 4
...     PURPLE = RED | BLUE
...     WHITE = RED | GREEN | BLUE
...
```

lo siguiente es cierto:

- las banderas de un solo bit son canónicas
- las banderas multibit y zero-bit son alias
- solo se retornan banderas canónicas durante la iteración:

  ```
  >>> list(Color.WHITE)
  [<Color.RED: 1>, <Color.GREEN: 2>, <Color.BLUE: 4>]
  ```

- negar una bandera o un conjunto de banderas retorna una nueva bandera/conjunto de banderas con el valor entero positivo correspondiente:

  ```
  >>> Color.BLUE
  <Color.BLUE: 4>

  >>> ~Color.BLUE
  <Color.RED|GREEN: 3>
  ```

- los nombres de las pseudo-banderas se construyen a partir de los nombres de sus miembros:

  ```
  >>> (Color.RED | Color.GREEN).name
  'RED|GREEN'

  >>> class Perm(IntFlag):
  ...     R = 4
  ...     W = 2
  ...     X = 1
  ...
  >>> (Perm.R & Perm.W).name is None  # efectivamente
  True
  ```

- las banderas de varios bits, también conocidas como alias, se pueden devolver desde las operaciones:

```
>>> Color.RED | Color.BLUE
<Color.PURPLE: 5>

>>> Color(7)  # or Color(-1)
<Color.WHITE: 7>

>>> Color(0)
<Color.BLACK: 0>
```

- comprobación de pertenencia / contención: las banderas de valor cero siempre se consideran contenidas:

  ```
  >>> Color.BLACK in Color.WHITE
  True
  ```

  de lo contrario, solo si todos los bits de una bandera están en la otra bandera, se devolverá True:

  ```
  >>> Color.PURPLE in Color.WHITE
  True

  >>> Color.GREEN in Color.PURPLE
  False
  ```

Hay un nuevo mecanismo de límite que controla cómo se manejan los bits no válidos/fuera de rango: `STRICT`, `CONFORM`, `EJECT` y `KEEP`:

- STRICT –> lanza una excepción cuando se presentan valores no válidos
- CONFORM –> descarta cualquier bit inválido
- EJECT -> pierde el estado de la bandera y se convierte en un int normal con el valor dado
- KEEP –> mantener los bits adicionales
  - mantiene el estado de la bandera y bits adicionales
  - los bits adicionales no aparecen en la iteración
  - bits adicionales aparecen en repr() y str()

El valor predeterminado para Flag es `STRICT`, el valor predeterminado para `IntFlag` es `EJECT` y el valor predeterminado para `_convert_` es `KEEP` (consulte `ssl.Options` para ver un ejemplo de cuándo se necesita `KEEP`).

¿En qué se diferencian las Enumeraciones (Enums) y las Banderas (Flags)?

Las enumeraciones tienen una metaclase personalizada que afecta a muchos aspectos de las clases **`Enum`** derivadas y sus instancias (miembros).

Clases de enumeración

The **`EnumType`** metaclass is responsible for providing the **`__contains__()`**, **`__dir__()`**, **`__iter__()`** and other methods that allow one to do things with an **`Enum`** class that fail on a typical class, such as `list(Color)` or `some_enum_var in Color`. **`EnumType`** is responsible for ensuring that various other methods on the final **`Enum`** class are correct (such as **`__new__()`**, **`__getnewargs__()`**, **`__str__()`** and **`__repr__()`**).

Clases de Banderas

Las banderas tienen una vista ampliada de la creación de alias: para ser canónico, el valor de una bandera debe ser un valor de potencia de dos y no un nombre duplicado. Por lo tanto, además de la definición de alias de **`Enum`**, una bandera sin valor (también conocida como `0`) o con más de un valor de potencia de dos (por ejemplo, `3`) se considera un alias.

Miembros de enumeración (también conocidos como instancias)

The most interesting thing about enum members is that they are singletons. **`EnumType`** creates them all while it is creating the enum class itself, and then puts a custom **`__new__()`** in place to ensure that no new ones are ever instantiated by returning only the existing member instances.

Miembros de Banderas

Los miembros de las Banderas se pueden recorrer de la misma manera que la clase **`Flag`**, y solo se devolverán los miembros canónicos. Por ejemplo:

```
>>> list(Color)
[<Color.RED: 1>, <Color.GREEN: 2>, <Color.BLUE: 4>]
```

(Note que `BLACK`, `PURPLE`, y `WHITE` no se muestran.)

Invertir un miembro de la bandera devuelve el valor positivo correspondiente, en lugar de un valor negativo — por ejemplo:

```
>>> ~Color.RED
<Color.GREEN|BLUE: 6>
```

Los miembros de las Banderas tienen una longitud que corresponde al número de valores de potencia de dos que contienen. Por ejemplo:

```
>>> len(Color.PURPLE)
2
```

Recetario de Enumeraciones

Si bien se espera que **`Enum`**, **`IntEnum`**, **`StrEnum`**, **`Flag`** y **`IntFlag`** cubran la mayoría de los casos de uso, no pueden cubrirlos todos. Aquí hay recetas para algunos tipos diferentes de enumeraciones que se pueden usar directamente o como ejemplos para crear las propias.

Omitir valores

En muchos casos de uso, a uno no le importa cuál es el valor real de una enumeración. Hay varias formas de definir este tipo de enumeración simple:

- usar instancias de **`auto`** para el valor
- usar instancias de **`object`** como valor

- use una cadena descriptiva como el valor
- use a tuple as the value and a custom `__new__()` to replace the tuple with an `int` value

El uso de cualquiera de estos métodos significa para el usuario que estos valores no son importantes y también permite agregar, eliminar o reordenar miembros sin tener que volver a numerar los miembros restantes.

Usando `auto`

El uso de `auto` se vería así:

```
>>> class Color(Enum):
...     RED = auto()
...     BLUE = auto()
...     GREEN = auto()
...
>>> Color.GREEN
<Color.GREEN: 3>
```

Usando `object`

El uso de `object` se vería así:

```
>>> class Color(Enum):
...     RED = object()
...     GREEN = object()
...     BLUE = object()
...
>>> Color.GREEN
<Color.GREEN: <object object at 0x...>>
```

This is also a good example of why you might want to write your own `__repr__()`:

```
>>> class Color(Enum):
...     RED = object()
...     GREEN = object()
...     BLUE = object()
...     def __repr__(self):
...         return "<%s.%s>" % (self.__class__.__name__,
...
>>> Color.GREEN
<Color.GREEN>
```

Usar una cadena descriptiva

Usando una cadena como el valor se vería así:

```
>>> class Color(Enum):
...     RED = 'stop'
...     GREEN = 'go'
...     BLUE = 'too fast!'
...
>>> Color.GREEN
<Color.GREEN: 'go'>
```

Using a custom `__new__()`

Using an auto-numbering `__new__()` would look like:

```
>>> class AutoNumber(Enum):
...     def __new__(cls):
...         value = len(cls.__members__) + 1
...         obj = object.__new__(cls)
...         obj._value_ = value
...         return obj
...
>>> class Color(AutoNumber):
...     RED = ()
...     GREEN = ()
...     BLUE = ()
...
>>> Color.GREEN
<Color.GREEN: 2>
```

Para hacer un `AutoNumber` de uso más general, agregue `*args` a la firma:

```
>>> class AutoNumber(Enum):
...     def __new__(cls, *args):      # éste es el únic
...         value = len(cls.__members__) + 1
...         obj = object.__new__(cls)
...         obj._value_ = value
...         return obj
...
```

Luego, cuando hereda de `AutoNumber`, puede escribir su propio `__init__` para manejar cualquier argumento adicional:

```
>>> class Swatch(AutoNumber):
...     def __init__(self, pantone='unknown'):
...         self.pantone = pantone
...     AUBURN = '3497'
...     SEA_GREEN = '1246'
...     BLEACHED_CORAL = () # ¡Nuevo color, aún sin códi
...
>>> Swatch.SEA_GREEN
<Swatch.SEA_GREEN: 2>
>>> Swatch.SEA_GREEN.pantone
'1246'
>>> Swatch.BLEACHED_CORAL.pantone
'unknown'
```

Nota

The **`__new__()`** method, if defined, is used during creation of the Enum members; it is then replaced by Enum's **`__new__()`** which is used after class creation for lookup of existing members.

Advertencia

No llame a `super().__new__()`, ya que encontrará el `__new__` de solo búsqueda; en su lugar, utilice directamente el tipo de datos – por ejemplo:

```
obj = int.__new__(cls, value)
```

Enum ordenado

Una enumeración ordenada que no se basa en `IntEnum` y, por lo tanto, mantiene las invariantes normales de `Enum` (como no ser comparable con otras enumeraciones):

```
>>> class OrderedEnum(Enum):
...     def __ge__(self, other):
...         if self.__class__ is other.__class__:
...             return self.value >= other.value
...         return NotImplemented
...     def __gt__(self, other):
...         if self.__class__ is other.__class__:
...             return self.value > other.value
...         return NotImplemented
...     def __le__(self, other):
...         if self.__class__ is other.__class__:
...             return self.value <= other.value
...         return NotImplemented
...     def __lt__(self, other):
...         if self.__class__ is other.__class__:
...             return self.value < other.value
...         return NotImplemented
...
>>> class Grade(OrderedEnum):
...     A = 5
...     B = 4
...     C = 3
...     D = 2
...     F = 1
...
>>> Grade.C < Grade.A
True
```

DuplicateFreeEnum

Lanza un error si se encuentra un nombre de miembro duplicado en lugar de crear un alias:

```
>>> class DuplicateFreeEnum(Enum):
...     def __init__(self, *args):
...         cls = self.__class__
...         if any(self.value == e.value for e in cls):
...             a = self.name
...             e = cls(self.value).name
...             raise ValueError(
...                 "aliases not allowed in DuplicateFre
...                 % (a, e))
...
>>> class Color(DuplicateFreeEnum):
...     RED = 1
...     GREEN = 2
...     BLUE = 3
...     GRENE = 2
...
Traceback (most recent call last):
  ...
ValueError: aliases not allowed in DuplicateFreeEnum:  '
```

Nota

Este es un ejemplo útil para subclasificar Enum para agregar o cambiar otros comportamientos, así como para no permitir alias. Si el único cambio deseado es prohibir los alias, se puede usar el decorador `unique()` en su lugar.

MultiValueEnum

Soporta tener más de un valor por miembro:

```
>>> class MultiValueEnum(Enum):
...     def __new__(cls, value, *values):
...         self = object.__new__(cls)
...         self._value_ = value
...         for v in values:
...             self._add_value_alias_(v)
...         return self
```

```
...
>>> class DType(MultiValueEnum):
...     float32 = 'f', 8
...     double64 = 'd', 9
...
>>> DType('f')
<DType.float32: 'f'>
>>> DType(9)
<DType.double64: 'd'>
```

Planeta

If `__new__()` or `__init__()` is defined, the value of the enum member will be passed to those methods:

```
>>> class Planet(Enum):
...     MERCURY = (3.303e+23, 2.4397e6)
...     VENUS   = (4.869e+24, 6.0518e6)
...     EARTH   = (5.976e+24, 6.37814e6)
...     MARS    = (6.421e+23, 3.3972e6)
...     JUPITER = (1.9e+27,   7.1492e7)
...     SATURN  = (5.688e+26, 6.0268e7)
...     URANUS  = (8.686e+25, 2.5559e7)
...     NEPTUNE = (1.024e+26, 2.4746e7)
...     def __init__(self, mass, radius):
...         self.mass = mass       # in kilograms
...         self.radius = radius   # in meters
...     @property
...     def surface_gravity(self):
...         # universal gravitational constant  (m3 kg-1
...         G = 6.67300E-11
...         return G * self.mass / (self.radius * self.
...
>>> Planet.EARTH.value
(5.976e+24, 6378140.0)
>>> Planet.EARTH.surface_gravity
9.802652743337129
```

Periodo de tiempo

An example to show the `_ignore_` attribute in use:

```
>>> import datetime as dt
>>> class Period(dt.timedelta, Enum):
...     "different lengths of time"
...     _ignore_ = 'Period i'
...     Period = vars()
...     for i in range(367):
...         Period['day_%d' % i] = i
...
>>> list(Period)[:2]
[<Period.day_0: datetime.timedelta(0)>, <Period.day_1: c
>>> list(Period)[-2:]
[<Period.day_365: datetime.timedelta(days=365)>, <Perioc
```

Subclase EnumType

Si bien la mayoría de las necesidades de enumeración se pueden satisfacer mediante la personalización de las subclases **`Enum`**, ya sea con decoradores de clase o funciones personalizadas, **`EnumType`** se puede dividir en subclases para proporcionar una experiencia de enumeración diferente.

HOWTO - Programación funcional

Autor:

A. M. Kuchling

Publicación:

0.32

En este documento, haremos un recorrido de las características de Python adecuadas para implementar programas en un estilo funcional. Después de una introducción de los conceptos de programación funcional, veremos las características del lenguaje como iteradores y generadores y módulos de librería relevantes como `itertools` y `functools`.

Introducción

Esta sección explica el concepto básico de programación funcional; si solo está interesado en aprender acerca de las características del lenguaje Python, pase a la siguiente sección en iteradores.

Los lenguajes de programación soportan la descomposición de problemas en muchas formas diferentes:

- La mayoría de los lenguajes de programación son **procedimentales**: los programas son listas de instrucciones que le dicen a la computadora qué hacer con la entrada del programa. C, Pascal e incluso las terminales Unix son lenguajes procedimentales.
- En los lenguajes **declarativos**, se escribe una especificación que describe el problema que se resolverá, y la implementación del lenguaje averigua como realizar el cálculo de forma eficiente. SQL es el lenguaje declarativo con el que probablemente esté más familiarizado; una consulta SQL describe el conjunto de datos que quiere recuperar, y el

motor SQL decide si escanear tablas o usar índices, qué subcláusulas deben ejecutarse primero, etc.
- Los programas **orientados a objetos** manipulan colecciones de objetos. Los objetos tienen estado interno y soportan métodos que consultan o modifican su estado interno de alguna manera. Smalltalk y Java son lenguajes orientados a objetos. C++ y Python son lenguajes que soportan la programación orientada a objetos, pero no fuerzan el uso de las características orientadas a objetos.
- La programación **funcional** descompone un problema en un conjunto de funciones. Idealmente, las funciones solo reciben entradas y producen salidas, y no tienen ningún estado interno que afecte la salida producida para una entrada dada. Los lenguajes funcionales bien conocidos incluyen la familia ML (Standard ML, OCaml, y otras variantes) y Haskell.

Los diseñadores de algunos lenguajes de computadora eligen enfatizar en un enfoque particular para programar. Esto a menudo hace difícil escribir programas que usen un enfoque diferente. Otros lenguajes son lenguajes multiparadigma que soportan varios enfoques diferentes. Lisp, C++ y Python son multiparadigma; puede escribir programas o librerías que son en gran parte procedimentales, orientados a objetos o funcionales en todos estos lenguajes. En un programa grande, las diferentes secciones podrían escribirse usando diferentes enfoques; la GUI podría ser orientada a objetos mientras la lógica de procesamiento es procedimental o funcional, por ejemplo.

En un programa funcional, la entrada fluye a través de un conjunto de funciones. Cada función opera sobre su entrada y produce alguna salida. El estilo funcional desalienta las funciones con efectos secundarios que modifican el estado interno o hacen otros cambios que no son visibles en el valor de retorno de la función. Las funciones que no tienen ningún efecto secundario se llaman **puramente funcionales**. Evitar los efectos secundarios significa no usar estructuras de datos que se actualicen mientras se ejecuta un programa; cada salida de la función debe depender solo de su entrada.

Algunos lenguajes son demasiado estrictos y ni siquiera tienen declaraciones de asignación como `a=3` o `c = a + b`, pero es difícil evitar todos los efectos secundarios, como imprimir en pantalla o escribir en un archivo de disco. Otro ejemplo es llamar a las funciones `print()` o `time.sleep()`, ninguna de las dos retorna un valor útil, ambas son llamadas solo por sus efectos secundarios de enviar texto a la pantalla o pausar la ejecución por un segundo.

Los programas de Python escritos en estilo funcional usualmente no irán al extremo de evitar todas las E/S o todas las asignaciones; en cambio, proveerán una interfaz aparentemente funcional pero internamente usará características no funcionales. Por ejemplo, la implementación de una función todavía usará asignaciones a variables locales, pero no modificará variables globales ni tendrá otros efectos secundarios.

La programación funcional se puede considerar lo opuesto a la programación orientada a objetos. Los objetos son pequeñas capsulas que contienen algún estado interno junto con una colección de llamadas a métodos que le permiten modificar este estado, y los programas consisten en realizar el conjunto correcto de cambios de estado. La programación funcional quiere evitar cambios de estado tanto como sea posible y trabaja con flujos de datos entre funciones. En Python podría combinar los dos enfoques para escribir funciones que reciban y retornen instancias que representen objetos en su aplicación (mensajes de e-mail, transacciones, etc.).

El diseño funcional puede parecer una restricción extraña bajo la cuál trabajar. ¿Por qué evitaría objetos y efectos secundarios? Hay ventajas teóricas y prácticas para el estilo funcional:

- Demostrabilidad formal.
- Modularidad.
- Componibilidad.
- Facilidad de depurar y probar.

Demostrabilidad formal

Un beneficio teórico es que es más fácil construir una demostración matemática de que un programa funcional es correcto.

Por un largo tiempo los investigadores se han interesado en buscar formas de demostrar matemáticamente que los programas son correctos. Esto es diferente de probar un programa sobre numerosas entradas y concluir que su salida es usualmente correcta, o leer el código fuente de un programa y concluir que el código se ve bien; en lugar de eso el objetivo es una demostración rigurosa de que un programa produce el resultado correcto para todas las entradas posibles.

La técnica utilizada para demostrar que los programas son correctos es anotar **invariantes**, propiedades de los datos de entrada y de las variables del programa que siempre son verdaderas. Por cada línea de código, debe mostrar que si las invariantes X e Y son verdaderas **antes** que la línea sea ejecutada, las invariantes ligeramente diferentes X" e Y" son verdaderas **después** que la línea se ejecutó. Esto continúa hasta que alcance el fin del programa, punto en el cuál las invariantes deben coincidir con las condiciones deseadas en la salida del programa.

La evitación de las asignaciones de la programación funcional surge porque las asignaciones son difíciles de manejar con esta técnica; las asignaciones pueden romper invariantes que eran verdaderas antes de la asignación sin producir nuevas invariantes que se puedan propagar hacia adelante.

Desafortunadamente, demostrar que los programas son correctos es en gran parte impráctico y no relevante al software de Python. Incluso los programas triviales requieren demostraciones que tienen varias páginas; la demostración de correctitud para un programa moderadamente complicado sería enorme, y pocos o ninguno de los programas que usa diariamente (el interprete de Python, su analizador XML, su navegador web) se podrían demostrar que son correctos. Aún si anotó o generó una demostración, entonces estaría la cuestión de verificar la demostración; quizás hay un error en ella, y equivocadamente cree que demostró que el programa es correcto.

Modularidad

Un beneficio más práctico de la programación funcional es que fuerza a romper su problema en pequeñas piezas. Como resultado los programas son más modulares. Es más fácil especificar y escribir una función pequeña que hace una cosa que una función grande que realiza una transformación complicada. Las funciones pequeñas también son más fáciles de leer y comprobar si hay errores.

Facilidad de depurar y probar

Probar y depurar un programa en estilo funcional es más fácil.

La depuración se simplifica porque las funciones generalmente son pequeñas y claramente especificadas. Cuando un programa no funciona, cada función es un punto de interfaz donde puede comprobar si los datos son correctos. Puede ver las entradas y salidas intermedias para aislar rápidamente la función que es responsable de un error.

Las pruebas son más fáciles porque cada función es un sujeto potencial para una prueba unitaria. Las funciones no dependen de un estado del sistema que necesite ser replicado antes de correr una prueba; en lugar de eso solo tiene que sintetizar la entrada correcta y comprobar que la salida coincida con las expectativas.

Componibilidad

Mientras trabaja en un programa en estilo funcional, escribirá un número de funciones con diferentes entradas y salidas. Algunas de estas funciones inevitablemente estarán especializadas en una aplicación en particular, pero otras serán útiles en una amplia variedad de programas. Por ejemplo, una función que recibe la ruta de un directorio y retorna todos los archivos XML en el directorio, o una función que recibe el nombre de un archivo y retorna su contenido, se puede aplicar a muchas situaciones diferentes.

Con el tiempo formará una librería personal de utilidades. A menudo ensamblará nuevos programas organizando funciones existentes en una nueva configuración y escribiendo unas pocas funciones especializadas para la tarea actual.

Iteradores

Comenzaré viendo una característica del lenguaje Python que es una base importante para escribir programas en estilo funcional: iteradores.

Un iterador es un objeto que representa un flujo de datos; este objeto retorna los datos de a un elemento a la vez. Un iterador de Python debe soportar un método llamado `__next__()` que no recibe argumentos y siempre retorna el siguiente elemento en el flujo. Si no hay más elementos en el flujo, `__next__()` debe lanzar la excepción `StopIteration`. Los iteradores no tienen que ser finitos; es perfectamente razonable escribir un iterador que produce un flujo de datos infinito.

La función incorporada `iter()` recibe un objeto arbitrario e intenta retornar un iterador que retornará los contenidos o elementos del objeto, lanzando `TypeError` si el objeto no soporta iteración. Muchos tipos de datos integrados de Python soportan iteración, siendo los más comunes las listas y los diccionarios. Un objeto se llama iterable si puede obtener un iterador para él.

Puede experimentar con la interfaz de iteración manualmente:

```
>>> L = [1, 2, 3]
>>> it = iter(L)
>>> it
<...iterator object at ...>
>>> it.__next__()  # same as next(it)
1
>>> next(it)
2
>>> next(it)
3
>>> next(it)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
```

Python espera objetos iterables en muchos contextos diferentes, siendo el más importante la sentencia `for`. En la sentencia `for X` `in Y`, Y debe ser un iterador o algún objeto para el que **`iter()`** puede crear un iterador. Estas dos sentencias son equivalentes:

```
for i in iter(obj):
    print(i)

for i in obj:
    print(i)
```

Los iteradores se pueden materializar como listas o tuplas utilizando las funciones constructoras **`list()`** o **`tuple()`**:

```
>>> L = [1, 2, 3]
>>> iterator = iter(L)
>>> t = tuple(iterator)
>>> t
(1, 2, 3)
```

El desempaquetado de secuencias también soporta iteradores: si sabe que un iterador retornará N elementos, puede desempaquetarlos en una N-tupla:

```
>>> L = [1, 2, 3]
>>> iterator = iter(L)
>>> a, b, c = iterator
>>> a, b, c
(1, 2, 3)
```

Las funciones incorporadas como **`max()`** y **`min()`** pueden recibir un solo iterador como argumento y retornarán el elemento más grande o más pequeño. Los operadores `"in"` y `"not in"` también soportan iteradores: `X in iterator` es verdadero si X se encuentra en el flujo que retornó el iterador. Se encontrará con problemas obvios si el iterador es infinito; **`max()`**, **`min()`** nunca retornarán, y si el elemento X nunca aparece en el flujo, los operadores `"in"` y `"not in"` tampoco retornarán.

Note que solo puede ir hacia adelante en un iterador; no hay forma de obtener el elemento anterior, reiniciar el iterador o hacer una copia de él. Los objetos iteradores opcionalmente pueden proveer estas capacidades adicionales, pero el protocolo del iterador solo especifica el método `__next__()`. Por lo tanto las funciones pueden consumir toda la salida del iterador, y si necesita hacer algo diferente con el mismo flujo, tendrá que crear un nuevo iterador.

Tipos de datos que soportan iteradores

Ya hemos visto cómo las listas y tuplas soportan iteradores. De hecho, cualquier tipo de secuencia de Python, como cadenas de caracteres, automáticamente soportará la creación de un iterador.

Llamar a `iter()` en un diccionario retornará un iterador que recorrerá sobre las claves del diccionario:

```
>>> m = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May':
...       'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov':
>>> for key in m:
...     print(key, m[key])
Jan 1
Feb 2
Mar 3
Apr 4
May 5
Jun 6
Jul 7
Aug 8
Sep 9
Oct 10
Nov 11
Dec 12
```

Note que a partir de Python 3.7, se garantiza que el orden de iteración del diccionario es el mismo que el orden de inserción. En versiones anteriores, el comportamiento no estaba especificado y podía variar entre implementaciones.

Aplicar `iter()` a un diccionario siempre recorre sobre las claves, pero los diccionarios tienen métodos que retornan otros iteradores. Si quiere iterar sobre valores o pares clave/valor, puede explícitamente llamar a los métodos `values()` o `items()` para obtener un iterador apropiado.

El constructor **`dict()`** puede aceptar un iterador que retorna un flujo finito de tuplas `(key, value)`:

```
>>> L = [('Italy', 'Rome'), ('France', 'Paris'), ('US',
>>> dict(iter(L))
{'Italy': 'Rome', 'France': 'Paris', 'US': 'Washington D
```

Los archivos también soportan iteración llamando al método **`readline()`** hasta que no haya más líneas en el archivo. Esto significa que puede leer cada línea de un archivo de esta forma:

```
for line in file:
    # do something for each line
    ...
```

Los conjuntos pueden recibir sus contenidos de un iterable y le permiten iterar sobre los elementos del conjunto:

```
>>> S = {2, 3, 5, 7, 11, 13}
>>> for i in S:
...     print(i)
2
3
5
7
11
13
```

Expresiones generadoras y listas por comprensión

Dos operaciones comunes en la salida de un iterador son 1) realizar alguna operación para cada elemento, 2) elegir un subconjunto de elementos que reúnen alguna condición. Por ejemplo, dada una lista de cadena de caracteres, podría querer remover los espacios finales de cada línea o extraer todas las cadenas de caracteres que contienen una subcadena de caracteres dada.

Las listas por comprensión y las expresiones generadoras (forma abreviada: «listcomps» y «genexps») son una notación concisa para tales operaciones, prestadas del lenguaje de programación funcional Haskell (https://www.haskell.org/). Puede remover todos los espacios de un flujo de cadena de caracteres con el siguiente código:

```
>>> line_list = ['  line 1\n', 'line 2  \n', ' \n', '']

>>> # Generator expression -- returns iterator
>>> stripped_iter = (line.strip() for line in line_list)

>>> # List comprehension -- returns list
>>> stripped_list = [line.strip() for line in line_list]
```

Puede seleccionar solo ciertos elementos agregando una condición `"if"`:

```
>>> stripped_list = [line.strip() for line in line_list
...                  if line != ""]
```

Con una lista por comprensión, obtiene una lista de Python; `stripped_list` es una lista que contiene las líneas resultantes, no un iterador. Las expresiones generadoras retornan un iterador que calcula los valores cuando es necesario, sin necesidad de materializar todos los valores a la vez. Esto significa que las listas por comprensión son inútiles si está trabajando con iteradores que retornan un flujo infinito o una gran cantidad de datos. En estas situaciones son preferibles las expresiones generadoras.

Las expresiones generadoras están rodeadas por paréntesis («()») y las listas por comprensión están rodeadas por corchetes («[]»). Las expresiones generadoras tienen la forma:

```
( expression for expr in sequence1
             if condition1
             for expr2 in sequence2
             if condition2
             for expr3 in sequence3
             ...
             if condition3
```

```
            for exprN in sequenceN
            if conditionN )
```

Nuevamente, para una lista por comprensión solo los corchetes exteriores son diferentes (corchetes en lugar de paréntesis).

Los elementos de la salida generada serán valores sucesivos de `expression`. Las cláusulas `if` son todas opcionales; si está presente, `expression` es solo evaluado y añadido al resultado cuando `condition` es verdadero.

Las expresiones generadoras siempre se tienen que escribir dentro de paréntesis, pero los paréntesis que indican la llamada a una función también cuentan. Si quiere crear un iterador que se pase inmediatamente a una función puede escribir:

```
obj_total = sum(obj.count for obj in list_all_objects())
```

Las cláusulas `for...in` contienen las secuencias sobre las que se itera. Las secuencias no tienen que tener la misma longitud, porque son iteradas de izquierda a derecha, **no** en paralelo. Por cada elemento en `sequence1`, se recorre `sequence2` desde el inicio. Luego se recorre `sequence3` por cada par de elementos resultante de `sequence1` y `sequence2`.

Para ponerlo en otra forma, una lista por comprensión o expresión generadora es equivalente al siguiente código Python:

```
for expr1 in sequence1:
    if not (condition1):
        continue   # Skip this element
    for expr2 in sequence2:
        if not (condition2):
            continue   # Skip this element
        ...
        for exprN in sequenceN:
            if not (conditionN):
                continue   # Skip this element

            # Output the value of
```

```
             # the expression.
```

Esto significa que cuando hay múltiples cláusulas `for...in` pero no cláusulas `if`, la longitud de la salida resultante será igual al producto de las longitudes de todas las secuencias. Si tiene dos listas de longitud 3, la lista de salida tendrá 9 elementos:

```
>>> seq1 = 'abc'
>>> seq2 = (1, 2, 3)
>>> [(x, y) for x in seq1 for y in seq2]
[('a', 1), ('a', 2), ('a', 3),
 ('b', 1), ('b', 2), ('b', 3),
 ('c', 1), ('c', 2), ('c', 3)]
```

Para evitar introducir una ambigüedad en la gramática de Python, si `expression` está creando una tupla, debe estar rodeada de paréntesis. La primera lista por comprensión de abajo tiene un error de sintaxis, mientras que la segunda es correcta:

```
# Syntax error
[x, y for x in seq1 for y in seq2]
# Correct
[(x, y) for x in seq1 for y in seq2]
```

Generadores

Los generadores son una clase especial de funciones que simplifican la tarea de escribir iteradores. Las funciones regulares calculan un valor y lo retornan, pero los generadores retornan un iterador que retorna un flujo de valores.

Sin duda está relacionado con cómo funcionan las llamadas a funciones regulares en Python o C. Cuando llama a una función, esta obtiene su espacio de nombres privado donde se crean sus variables locales. Cuando la función alcanza una sentencia `return`, las variables locales son destruidas y el valor se retorna al llamador. Una llamada posterior a la misma función crea un nuevo espacio de nombres privado y un conjunto limpio de variables locales. Pero, ¿qué pasa si las variables locales no fueron desechadas en la salida de una función? ¿Qué pasa si más tarde podría reanudar la función desde donde quedó? Esto es lo que proveen los generadores; se pueden pensar como funciones que se reanudan.

Este es el ejemplo más simple de una función generadora:

```
>>> def generate_ints(N):
...     for i in range(N):
...         yield i
```

Cualquier función que contiene una palabra clave **`yield`** es una función generadora; esto es detectado por el compilador bytecode de Python que compila la función de forma especial como resultado.

Cuando llama a una función generadora, no retorna un solo valor; en lugar de eso retorna un objeto generador que soporta el protocolo iterador. Al ejecutar la expresión `yield`, el generador produce el valor de `i`, de forma similar a una sentencia `return`. La gran diferencia entre una sentencia `yield` y un `return` es que al alcanzar un `yield` se suspende el estado de ejecución del generador y se preservan las variables locales. En la próxima llamada al método **`__next__()`** del generador, la función reanudará la ejecución.

Este es un ejemplo de uso del generador `generate_ints()`:

```
>>> gen = generate_ints(3)
>>> gen
<generator object generate_ints at ...>
>>> next(gen)
0
>>> next(gen)
1
>>> next(gen)
2
>>> next(gen)
Traceback (most recent call last):
  File "stdin", line 1, in <module>
  File "stdin", line 2, in generate_ints
```

```
StopIteration
```

De igual forma podría escribir `for i in generate_ints(5)`, o `a, b, c = generate_ints(3)`.

Dentro de una función generadora, `return value` causa que se lance `StopIteration(value)` del método **`__next__()`**. Una vez que esto pase, o que se alcance el final de la función, termina la procesión de valores y el generador no puede producir más valores.

Podría lograr el efecto de los generadores manualmente escribiendo su propia clase y guardando todas las variables locales del generador como variables de instancia. Por ejemplo, retornar una lista de enteros se podría hacer estableciendo `self.count` a 0, y teniendo el método **`__next__()`** que incrementa `self.count` y lo retorna. Sin embargo, para un generador moderadamente complicado, escribir una clase correspondiente puede ser mucho más confuso.

El banco de pruebas incluido con la librería de Python, Lib/test/test_generators.py [https://github.com/python/cpython/tree/3.14/Lib/test/test_generators.py], contiene un número de ejemplos más interesantes. Este es un generador que implementa un recorrido inorden de un árbol usando generadores recursivamente.

```
# A recursive generator that generates Tree leaves in ir
def inorder(t):
    if t:
        for x in inorder(t.left):
            yield x

        yield t.label

        for x in inorder(t.right):
            yield x
```

Otros dos ejemplos en `test_generators.py` producen soluciones al problema de N-reinas (ubicar N reinas en un tablero de ajedrez de NxN de forma que ninguna reina amenace a otra) y el problema del caballo (encontrar una ruta que lleve a un caballo a cada cuadro de un tablero de ajedrez de NxN sin visitar ningún cuadro dos veces).

Pasar valores a un generador

En Python 2.4 y anteriores, los generadores solo producían salida. Una vez que el código de un generador era invocado para crear un iterador, no había forma de pasar ninguna información nueva a la función cuando se reanuda su ejecución. Podría lograr esta habilidad haciendo que el generador mire una variable global o pasando en algún objeto mutable que el llamador luego modifica, pero estos enfoques son confusos.

En Python 2.5 hay una forma más simple de pasar valores a un generador. **`yield`** se convirtió en una expresión, retornando un valor que se puede asignar a una variable o sobre el que se puede operar:

```
val = (yield i)
```

Recomiendo que **siempre** ponga paréntesis alrededor de una expresión `yield` cuando esté haciendo algo con el valor retornado, como en el ejemplo de arriba. Los paréntesis no siempre son necesarios, pero siempre es más fácil agregarlos en lugar de tener que recordar cuándo son necesarios.

(**PEP 342** [https://peps.python.org/pep-0342/] explica las reglas exactas, que son que una expresión `yield` siempre debe estar entre paréntesis excepto cuando se encuentra en la expresión de primer nivel en el lado derecho de una asignación. Esto significa que puede escribir `val = yield i` pero tiene que usar paréntesis cuando hay una operación, como en `val = (yield i) + 12`.)

Los valores son enviados a un generador llamando a su método **`send(value)`**. Este método reanuda el código del generador y la expresión `yield` retorna el valor especificado. Si se llama al método regular **`__next__()`**, `yield` retorna `None`.

Este es un simple contador que incrementa por 1 y permite cambiar los valores del contador interno.

```
def counter(maximum):
    i = 0
    while i < maximum:
        val = (yield i)
        # If value provided, change counter
        if val is not None:
            i = val
        else:
            i += 1
```

Y este es un ejemplo del cambio del contador:

```
>>> it = counter(10)
>>> next(it)
0
>>> next(it)
1
>>> it.send(8)
8
>>> next(it)
9
>>> next(it)
Traceback (most recent call last):
  File "t.py", line 15, in <module>
    it.next()
StopIteration
```

Como `yield` a menudo retornará `None`, siempre debería comprobar este caso. No use simplemente su valor en expresiones a menos que esté seguro que el método **`send()`** será el único método utilizado para reanudar su función generadora.

Además del método **`send()`**, hay otros dos métodos para generadores:

- **`throw(value)`** is used to raise an exception inside the generator; the exception is raised by the `yield` expression where the generator's execution is paused.
- **`close()`** sends a **`GeneratorExit`** exception to the

> generator to terminate the iteration. On receiving this exception, the generator's code must either raise **`GeneratorExit`** or **`StopIteration`**; catching the exception and doing anything else is illegal and will trigger a **`RuntimeError`**. **`close()`** will also be called by Python's garbage collector when the generator is garbage-collected.
>
> Si necesita ejecutar un código de limpieza cuando ocurre un **`GeneratorExit`**, sugiero usar un `try: ... finally:` en lugar de capturar **`GeneratorExit`**.

El efecto acumulativo de estos cambios es convertir a los generadores de productores de información unidireccionales en productores y consumidores.

Los generadores también se convierten en **corrutinas**, una forma más generalizada de subrutinas. Las subrutinas inician en un punto y salen en otro punto (el inicio de la función, y la sentencia `return`), pero las corrutinas pueden iniciar, salir, y reanudarse en muchos puntos diferentes (las sentencias `yield`).

Funciones incorporadas

Veamos con más detalle las funciones incorporadas usadas a menudo con iteradores.

Dos de las funciones incorporadas de Python, **`map()`** y **`filter()`** duplican las características de las expresiones generadoras:

`map(f, iterA, iterB, ...)` retorna un iterador sobre la secuencia

```
f(iterA[0], iterB[0]), f(iterA[1], iterB[1]),
f(iterA[2], iterB[2]), ....
```

```
>>> def upper(s):
...     return s.upper()

>>> list(map(upper, ['sentence', 'fragment']))
['SENTENCE', 'FRAGMENT']
>>> [upper(s) for s in ['sentence', 'fragment']]
```

```
    ['SENTENCE', 'FRAGMENT']
```

Por supuesto puede lograr el mismo efecto con una lista por comprensión.

`filter(predicate, iter)` retorna un iterador sobre todos los elementos de la secuencia que reúnen cierta condición, y de forma similar está duplicado por las listas por comprensión. Un **predicado** es una función que retorna un valor de verdad de alguna condición; para usarlo con **`filter()`**, el predicado debe recibir un solo valor.

```
>>> def is_even(x):
...     return (x % 2) == 0

>>> list(filter(is_even, range(10)))
[0, 2, 4, 6, 8]
```

Esto también se puede escribir como una lista por comprensión:

```
>>> list(x for x in range(10) if is_even(x))
[0, 2, 4, 6, 8]
```

`enumerate(iter, start=0)` enumera los elementos en el iterable retornando 2-tuplas que contienen la enumeración (desde *start*) y cada elemento.

```
>>> for item in enumerate(['subject', 'verb', 'object'])
...     print(item)
(0, 'subject')
(1, 'verb')
(2, 'object')
```

`enumerate()` a menudo se usa cuando se recorre una lista y se registran los índices en los que se reúnen una cierta condición:

```
f = open('data.txt', 'r')
for i, line in enumerate(f):
    if line.strip() == '':
        print('Blank line at line #%i' % i)
```

`sorted(iterable, key=None, reverse=False)` reúne todos los elementos de un iterable en una lista, ordena la lista, y retorna el resultado ordenado. Los argumentos *key* y *reverse* se pasan a través del método **`sort()`** de la lista construida.

```
>>> import random
>>> # Generate 8 random numbers between [0, 10000)
>>> rand_list = random.sample(range(10000), 8)
>>> rand_list
[769, 7953, 9828, 6431, 8442, 9878, 6213, 2207]
>>> sorted(rand_list)
[769, 2207, 6213, 6431, 7953, 8442, 9828, 9878]
>>> sorted(rand_list, reverse=True)
[9878, 9828, 8442, 7953, 6431, 6213, 2207, 769]
```

(Para un tratamiento más detallado del ordenamiento, ver Sorting Techniques.)

Las funciones incorporadas **`any(iter)`** y **`all(iter)`** ven los valores de verdad de los contenidos de un iterable. **`any()`** retorna `True` si algún elemento en el iterable es un valor verdadero, y **`all()`** retorna `True` si todos los elementos son valores verdaderos:

```
>>> any([0, 1, 0])
True
>>> any([0, 0, 0])
False
>>> any([1, 1, 1])
True
>>> all([0, 1, 0])
False
>>> all([0, 0, 0])
False
>>> all([1, 1, 1])
True
```

`zip(iterA, iterB, ...)` toma un elemento de cada iterable y los retorna en una tupla:

```
zip(['a', 'b', 'c'], (1, 2, 3)) =>
  ('a', 1), ('b', 2), ('c', 3)
```

Esto no construye una lista en memoria y consume todos los iteradores de entrada antes de ejecutar; en lugar de eso, las tuplas se construyen y retornan solo si son requeridas. (El término técnico para este comportamiento es evaluación perezosa [https://es.wikipedia.org/wiki/Evaluaci%C3%B3n_perezosa].)

Se pretende que el iterador se use con iterables que tengan todos la misma longitud. Si los iterables son de diferentes longitudes, el flujo resultante tendrá la misma longitud que el iterable más corto.

```
zip(['a', 'b'], (1, 2, 3)) =>
  ('a', 1), ('b', 2)
```

Sin embargo, debería evitar hacer esto, porque se puede tomar un elemento de los iteradores más largos y descartarlo. Esto significa que ya no puede seguir usando los iteradores porque corre el riesgo de saltarse un elemento descartado.

El módulo itertools

The `itertools` module contains a number of commonly used iterators as well as functions for combining several iterators. This section will introduce the module’s contents by showing small examples.

Las funciones del módulo caen en unas pocas clases generales:

- Funciones que crean un nuevo iterador basado en un iterador existente.
- Funciones para tratar los elementos de un iterador como argumentos de función.
- Funciones para seleccionar porciones de la salida de un iterador.
- Una función para agrupar la salida de un iterador.

Crear nuevos iteradores

`itertools.count(start, step)` retorna un flujo infinito de valores equiespaciados. Opcionalmente puede suministrar el número de inicio, que por defecto es 0, y el intervalo entre números, que por defecto es 1:

```
itertools.count() =>
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ...
itertools.count(10) =>
  10, 11, 12, 13, 14, 15, 16, 17, 18, 19, ...
itertools.count(10, 5) =>
  10, 15, 20, 25, 30, 35, 40, 45, 50, 55, ...
```

`itertools.cycle(iter)` guarda una copia de los contenidos de un iterable provisto y retorna un nuevo iterador que retorna sus elementos del primero al último. El nuevo iterador repetirá estos elementos infinitamente.

```
itertools.cycle([1, 2, 3, 4, 5]) =>
  1, 2, 3, 4, 5, 1, 2, 3, 4, 5, ...
```

`itertools.repeat(elem, [n])` retorna el elemento provisto *n* veces, o retorna el elemento indefinidamente si no se provee *n*.

```
itertools.repeat('abc') =>
  abc, abc, abc, abc, abc, abc, abc, abc, abc, abc, ...
itertools.repeat('abc', 5) =>
  abc, abc, abc, abc, abc
```

`itertools.chain(iterA, iterB, ...)` recibe un número arbitrario de iterables como entrada, y retorna todos los elementos del primer iterador, luego todos los elementos del segundo, y así sucesivamente, hasta que hayan sido agotados todos los iterables.

```
itertools.chain(['a', 'b', 'c'], (1, 2, 3)) =>
  a, b, c, 1, 2, 3
```

`itertools.islice(iter, [start], stop, [step])` retorna un flujo que es un segmento del iterador. Con solo un argumento *stop*, retornará los primeros *stop* elementos. Si suministra un índice de inicio, obtendrá los elementos *stop-start*, y si suministra un valor para *step*, los elementos se saltarán en consecuencia. A diferencia de la segmentación de listas y cadenas de caracteres de Python, no puede usar valores negativos para *start*, *stop* o *step*.

```
itertools.islice(range(10), 8) =>
  0, 1, 2, 3, 4, 5, 6, 7
itertools.islice(range(10), 2, 8) =>
  2, 3, 4, 5, 6, 7
itertools.islice(range(10), 2, 8, 2) =>
  2, 4, 6
```

`itertools.tee(iter, [n])` replica un iterador; retorna *n* iteradores independientes que retornarán los contenidos del iterador fuente. Si no suministra un valor para *n*, por defecto es 2. Replicar iteradores requiere guardar algunos de los contenidos del iterador fuente, esto puede consumir memoria significativa si el iterador es grande y uno de los nuevos iteradores se consume más que los otros.

```
itertools.tee( itertools.count() ) =>
   iterA, iterB

where iterA ->
   0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ...

and   iterB ->
   0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ...
```

Aplicar funciones a los elementos

El módulo **`operator`** contiene un conjunto de funciones correspondientes a los operadores de Python. Algunos ejemplos son **`operator.add(a, b)`** (suma dos valores), **`operator.ne(a, b)`** (lo mismo que `a != b`) y **`operator.attrgetter('id')`** (retorna un invocable que obtiene el atributo `.id`).

`itertools.starmap(func, iter)` asume que el iterable retornará un flujo de tuplas, y llama a *func* usando estas tuplas como los argumentos:

```
itertools.starmap(os.path.join,
                  [('/bin', 'python'), ('/usr', 'bin', '
                   ('/usr', 'bin', 'perl'), ('/usr', 'bi
=>
```

```
  /bin/python, /usr/bin/java, /usr/bin/perl, /usr/bin/ru
```

Seleccionar elementos

Otro grupo de funciones elige un subconjunto de elementos de un iterador basado en un predicado dado.

`itertools.filterfalse(predicate, iter)` es el opuesto de **`filter()`**, retornando todos los elementos para los que el predicado retorna falso:

```
itertools.filterfalse(is_even, itertools.count()) =>
  1, 3, 5, 7, 9, 11, 13, 15, ...
```

`itertools.takewhile(predicate, iter)` retorna los elementos mientras el predicado retorne verdadero. Una vez que el predicado retorna falso, el iterador indicará el final de sus resultados.

```
def less_than_10(x):
    return x < 10

itertools.takewhile(less_than_10, itertools.count()) =>
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9

itertools.takewhile(is_even, itertools.count()) =>
  0
```

`itertools.dropwhile(predicate, iter)` descarta los elementos mientras el predicado retorne verdadero, y luego retorna el resto de los resultados del iterable.

```
itertools.dropwhile(less_than_10, itertools.count()) =>
  10, 11, 12, 13, 14, 15, 16, 17, 18, 19, ...

itertools.dropwhile(is_even, itertools.count()) =>
  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ...
```

`itertools.compress(data, selectors)` recibe dos iteradores y retorna solo aquellos elementos de *data* para los cuales el elemento correspondiente de *selectors* es verdadero, deteniéndose cuando alguno se termine:

```
itertools.compress([1, 2, 3, 4, 5], [True, True, False,
    1, 2, 5
```

Funciones combinatorias

La función `itertools.combinations(iterable, r)` retorna un iterador proporcionando todas las combinaciones de *r*-tuplas de los elementos contenidos en *iterable*.

```
itertools.combinations([1, 2, 3, 4, 5], 2) =>
   (1, 2), (1, 3), (1, 4), (1, 5),
   (2, 3), (2, 4), (2, 5),
   (3, 4), (3, 5),
   (4, 5)

itertools.combinations([1, 2, 3, 4, 5], 3) =>
   (1, 2, 3), (1, 2, 4), (1, 2, 5), (1, 3, 4), (1, 3, 5),
   (2, 3, 4), (2, 3, 5), (2, 4, 5),
   (3, 4, 5)
```

Los elementos dentro de cada tupla permanecen en el mismo orden en el que *iterable* los retornó. Por ejemplo, el número 1 siempre está antes que 2, 3, 4 o 5 en los ejemplos de arriba. Una función similar, `itertools.permutations(iterable, r=None)`, remueve esta restricción en el orden, retornando todas las disposiciones posibles de longitud *r*:

```
itertools.permutations([1, 2, 3, 4, 5], 2) =>
   (1, 2), (1, 3), (1, 4), (1, 5),
   (2, 1), (2, 3), (2, 4), (2, 5),
   (3, 1), (3, 2), (3, 4), (3, 5),
   (4, 1), (4, 2), (4, 3), (4, 5),
   (5, 1), (5, 2), (5, 3), (5, 4)

itertools.permutations([1, 2, 3, 4, 5]) =>
   (1, 2, 3, 4, 5), (1, 2, 3, 5, 4), (1, 2, 4, 3, 5),
   ...
```

```
  (5, 4, 3, 2, 1)
```

Si no suministra un valor para *r* se usa la longitud del iterable, lo que significa que se permutan todos los elementos.

Note que estas funciones producen todas las combinaciones posibles por posición y no requieren que los contenidos de *iterable* sean únicos:

```
itertools.permutations('aba', 3) =>
  ('a', 'b', 'a'), ('a', 'a', 'b'), ('b', 'a', 'a'),
  ('b', 'a', 'a'), ('a', 'a', 'b'), ('a', 'b', 'a')
```

La tupla idéntica `('a', 'a', 'b')` aparece dos veces, pero las dos cadenas de caracteres “a” provienen de diferentes posiciones.

La función **`itertools.combinations_with_replacement(iterable, r)`** relaja una restricción diferente: los elementos se pueden repetir dentro de una misma tupla. Conceptualmente un elemento se elige para la primera posición de cada tupla y luego se reemplaza antes de que se elija el segundo elemento.

```
itertools.combinations_with_replacement([1, 2, 3, 4, 5],
  (1, 1), (1, 2), (1, 3), (1, 4), (1, 5),
  (2, 2), (2, 3), (2, 4), (2, 5),
  (3, 3), (3, 4), (3, 5),
  (4, 4), (4, 5),
  (5, 5)
```

Agrupar elementos

La última función que trataré, **`itertools.groupby(iter, key_func=None)`**, es la más complicada. `key_func(elem)` es una función que calcula un valor clave para cada elemento retornado por el iterable. Si no suministra una función de clave, la clave será simplemente el elemento mismo.

`groupby()` reúne todos los elementos consecutivos del iterable subyacente que tienen el mismo valor clave, y retorna un flujo de 2-tuplas que contienen un valor clave y un iterador para los elementos con esa clave.

```
city_list = [('Decatur', 'AL'), ('Huntsville', 'AL'), ('
             ('Anchorage', 'AK'), ('Nome', 'AK'),
             ('Flagstaff', 'AZ'), ('Phoenix', 'AZ'), ('T
             ...
            ]

def get_state(city_state):
    return city_state[1]

itertools.groupby(city_list, get_state) =>
  ('AL', iterator-1),
  ('AK', iterator-2),
  ('AZ', iterator-3), ...

where
iterator-1 =>
  ('Decatur', 'AL'), ('Huntsville', 'AL'), ('Selma', 'AL
iterator-2 =>
  ('Anchorage', 'AK'), ('Nome', 'AK')
iterator-3 =>
  ('Flagstaff', 'AZ'), ('Phoenix', 'AZ'), ('Tucson', 'AZ
```

`groupby()` asume que los contenidos del iterable subyacente ya se ordenó basado en la clave. Note que los iteradores retornados también usan el iterable subyacente, así que tiene que consumir los resultados de iterador-1 antes de solicitar iterador-2 y su clave correspondiente.

El módulo functools

The **`functools`** module contains some higher-order functions. A **higher-order function** takes one or more functions as input and returns a new function. The most useful tool in this module is the **`functools.partial()`** function.

Para programas escritos en un estilo funcional, a veces querrá construir variantes de funciones existentes que tienen algunos de los parámetros predefinidos. Considere una función de Python `f(a, b, c)`; puede querer crear una nueva función `g(b, c)` que sea equivalente a `f(1, b, c)`; está completando un valor para uno de los parámetros de `f()`. Esto se llama «aplicación parcial de funciones».

El constructor para **`partial()`** recibe los argumentos `(function, arg1, arg2, ..., kwarg1=value1, kwarg2=value2)`. El objeto resultante es invocable, por lo que puede invocar a `function` con los argumentos rellenados.

Aquí hay un ejemplo pequeño pero realista:

```
import functools

def log(message, subsystem):
    """Write the contents of 'message' to the specified
    print('%s: %s' % (subsystem, message))
    ...

server_log = functools.partial(log, subsystem='server')
server_log('Unable to open socket')
```

`functools.reduce(func, iter, [initial_value])` realiza acumulativamente una operación en todos los elementos del iterable y, por lo tanto, no se puede aplicar a infinitos iterables. *func* debe ser una función que recibe dos elementos y retorna un solo valor. **`functools.reduce()`** recibe los primeros dos elementos A y B retornados por el iterador y calcula `func(A, B)`. Luego pide el tercer elemento, C, calcula `func(func(A, B), C)`, combina este resultado con el cuarto elemento retornado, y continua hasta que se agote el iterable. Si el iterable no retorna ningún valor, se lanza una excepción **`TypeError`**. Si se suministra el valor inicial, se usa como punto inicial y `func(initial_value, A)` es el primer cálculo.

```
>>> import operator, functools
>>> functools.reduce(operator.concat, ['A', 'BB', 'C'])
'ABBC'
>>> functools.reduce(operator.concat, [])
```

```
Traceback (most recent call last):
  ...
TypeError: reduce() of empty sequence with no initial va
>>> functools.reduce(operator.mul, [1, 2, 3], 1)
6
>>> functools.reduce(operator.mul, [], 1)
1
```

Si usa `operator.add()` con `functools.reduce()`, sumará todos los elementos del iterable. Este caso es tan común que hay una función incorporada especial llamada `sum()` para calcularla:

```
>>> import functools, operator
>>> functools.reduce(operator.add, [1, 2, 3, 4], 0)
10
>>> sum([1, 2, 3, 4])
10
>>> sum([])
0
```

Sin embargo, para muchos usos de `functools.reduce()` puede ser mas claro simplemente escribir el ciclo `for`:

```
import functools
# Instead of:
product = functools.reduce(operator.mul, [1, 2, 3], 1)

# You can write:
product = 1
for i in [1, 2, 3]:
    product *= i
```

A related function is `itertools.accumulate(iterable, func=operator.add)`. It performs the same calculation, but instead of returning only the final result, `accumulate()` returns an iterator that also yields each partial result:

```
itertools.accumulate([1, 2, 3, 4, 5]) =>
  1, 3, 6, 10, 15
```

```
itertools.accumulate([1, 2, 3, 4, 5], operator.mul) =>
  1, 2, 6, 24, 120
```

El módulo operator

El módulo `operator` se mencionó anteriormente. Contiene un conjunto de funciones que corresponden a los operadores de Python. Estas funciones a menudo son útiles en código en estilo funcional porque le salvan de escribir funciones triviales que realizan una sola operación.

Algunas de las funciones en este módulo son:

- Operaciones matemáticas: `add()`, `sub()`, `mul()`, `floordiv()`, `abs()`, …
- Operaciones lógicas: `not_()`, `truth()`.
- Operaciones bit a bit: `and_()`, `or_()`, `invert()`.
- Comparaciones: `eq()`, `ne()`, `lt()`, `le()`, `gt()` y `ge()`.
- Identidad de objeto: `is_()`, `is_not()`.

Consulte la documentación del módulo operator para una lista completa.

Funciones pequeñas y la expresión lambda

Cuando se escriben programas en estilo funcional, a menudo necesitará pequeñas funciones que actúen como predicados o que combinen elementos de alguna manera.

Si hay una función incorporada o un módulo de Python, no necesita definir una nueva función en absoluto:

```
stripped_lines = [line.strip() for line in lines]
existing_files = filter(os.path.exists, file_list)
```

Si la función que necesita no existe, necesita escribirla. Una forma de escribir funciones pequeñas es usar la expresión **`lambda`**. `lambda` recibe un número de parámetros y una expresión que combina estos parámetros, y crea una función anónima que retorna el valor de la expresión:

```
adder = lambda x, y: x+y

print_assign = lambda name, value: name + '=' + str(valu
```

Una alternativa es simplemente usar la sentencia `def` y definir una función en la forma usual:

```
def adder(x, y):
    return x + y

def print_assign(name, value):
    return name + '=' + str(value)
```

¿Qué alternativa es preferible? Esa es una pregunta de estilo; mi rumbo usual es evitar usar `lambda`.

Una razón para mi preferencia es que `lambda` es muy limitado en las funciones que puede definir. El resultado tiene que ser calculable como una sola expresión, que significa que no tiene comparaciones multivía `if... elif... else` o sentencias `try... except`. Si intenta hacer mucho en una sentencia `lambda`, terminará con una expresión demasiado complicada que es difícil de leer. Rápido, ¿qué hace el siguiente código?

```
import functools
total = functools.reduce(lambda a, b: (0, a[1] + b[1]),
```

Puede averiguarlo, pero toma tiempo desenredar la expresión para averiguar que está pasando. Usar una corta sentencia `def` anidada hace las cosas un poco mejor:

```
import functools
def combine(a, b):
    return 0, a[1] + b[1]

total = functools.reduce(combine, items)[1]
```

Pero lo mejor de todo sería si simplemente hubiese usado un ciclo `for`:

```
total = 0
for a, b in items:
    total += b
```

O la función incorporada `sum()` y una expresión generadora:

```
total = sum(b for a, b in items)
```

Muchos usos de `functools.reduce()` son más claros cuando se escriben como ciclos `for`.

Fredrik Lundh una vez sugirió el siguiente conjunto de reglas para refactorizar los usos de `lambda`:

1. Escribir una función lambda.
2. Escribir un comentario explicando qué demonios hace esa lambda.
3. Estudiar el comentario por un momento, y pensar en un nombre que capture la esencia del comentario.
4. Convertir la lambda a una sentencia def, usando ese nombre.
5. Remover el comentario.

Me gustan mucho estas reglas, pero es libre de disentir acerca de si este estilo libre de lambda es mejor.

Historia de revisiones y reconocimientos

Al autor le gustaría agradecer a las siguientes personar por ofrecer sugerencias, correcciones y asistencia con varios borradores de este articulo: Ian Bicking, Nick Coghlan, Nick Efford, Raymond Hettinger, Jim Jewett, Mike Krell, Leandro Lameiro, Jussi Salmela, Collin Winter, Blake Winton.

Versión 0.1: publicada el 30 de junio de 2006.

Versión 0.11: publicada el 1 de julio de 2006. Errores tipográficos arreglados.

Versión 0.2: publicada el 10 de julio de 2006. Secciones genexp y listcomp unidas en una sola. Errores tipográficos arreglados.

Versión 0.21: Agregadas mas referencias sugeridas en la lista de correos tutor.

Versión 0.30: Agrega una sección sobre el módulo `functional` escrito por Collin Winter; agrega una sección corta sobre el módulo operator; y unas pocas otras ediciones.

Referencias

Generales

Structure and Interpretation of Computer Programs, by Harold Abelson and Gerald Jay Sussman with Julie Sussman. The book can be found at https://mitpress.mit.edu/sicp. In this classic textbook of computer science, chapters 2 and 3 discuss the use of sequences and streams to organize the data flow inside a program. The book uses Scheme for its examples, but many of the design approaches described in these chapters are applicable to functional-style Python code.

https://defmacro.org/2006/06/19/fp.html: A general introduction to functional programming that uses Java examples and has a lengthy historical introduction.

https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional: Entrada general de Wikipedia que describe la programación funcional.

https://es.wikipedia.org/wiki/Corrutina: Entrada para corrutinas.

https://en.wikipedia.org/wiki/Partial_application: Entry for the concept of partial function application.

https://es.wikipedia.org/wiki/Currificaci%C3%B3n: Entrada para el concepto de currificación.

Específicas de Python

https://gnosis.cx/TPiP/: The first chapter of David Mertz’s book *Text Processing in Python* discusses functional programming for text processing, in the section titled «Utilizing Higher-Order Functions in Text Processing».

Mertz también escribió una serie de artículos de 3 partes sobre programación funcional para el sitio DeveloperWorks de IBM; ver parte 1 [https://developer.ibm.com/articles/l-prog/], parte 2 [https://developer.ibm.com/tutorials/l-prog2/], y parte 3 [https://developer.ibm.com/tutorials/l-prog3/],

Documentación de Python

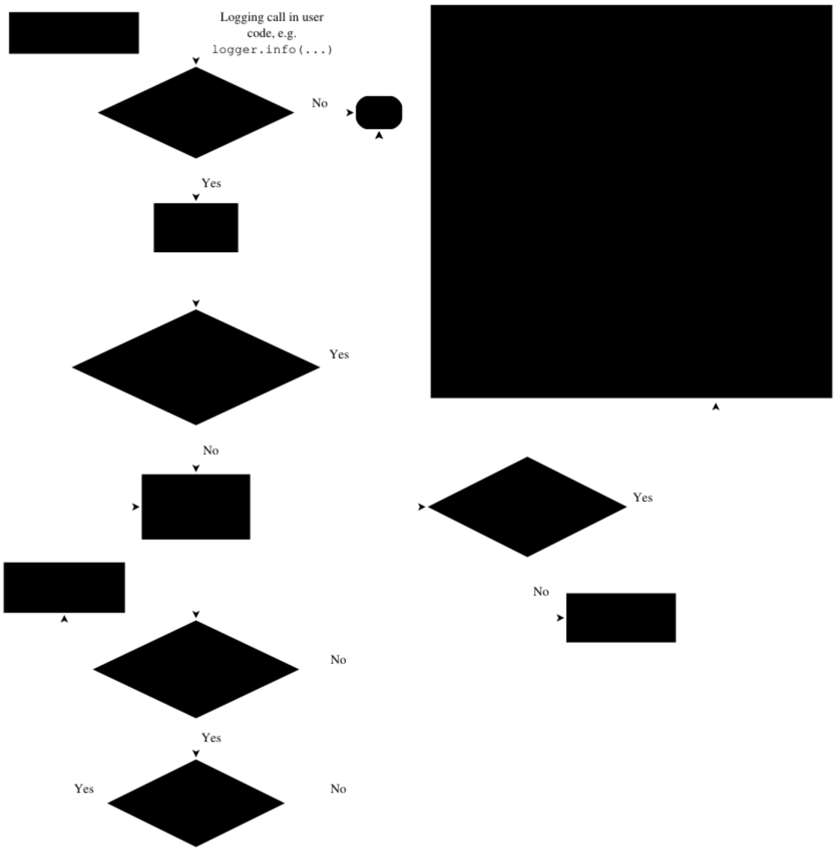
Documentación del módulo `itertools`.

Documentación del módulo `functools`.

Documentación del módulo `operator`.

PEP 289 [https://peps.python.org/pep-0289/]: «Expresiones generadoras»

PEP 342 [https://peps.python.org/pep-0342/]: «Corrutinas a través de generadores mejorados» describe las características del nuevo generador en Python 2.5.


Logging call in user code, e.g.
logger.info(...)
No
Yes
Yes
No
Yes
No
No
Yes
Yes
No

HOWTO hacer registros (*Logging*)

Autor:

Vinay Sajip <vinay_sajip at red-dove dot com>

Esta página contiene información de tutorial. Para enlaces a información de referencia y una guía de registro, consulte Otros recursos.

Tutorial básico de *logging*

Logging es un medio de rastrear los eventos que ocurren cuando se ejecuta algún software. El desarrollador del software agrega llamadas de registro a su código para indicar que ciertos eventos han ocurrido. Un evento se describe mediante un mensaje descriptivo que puede contener opcionalmente datos variables (es decir, datos que son potencialmente diferentes para cada ocurrencia del evento). Los eventos también tienen una importancia que el desarrollador atribuye al evento; la importancia también puede llamarse el *nivel* o la *severidad*.

Cuándo usar *logging*

You can access logging functionality by creating a logger via `logger = logging.getLogger(__name__)`, and then calling the logger's **`debug()`**, **`info()`**, **`warning()`**, **`error()`** and **`critical()`** methods. To determine when to use logging, and to see which logger methods to use when, see the table below. It states, for each of a set of common tasks, the best tool to use for that task.

La tarea que quieres realizar	La mejor herramienta para la tarea
Mostrar salidas de la consola para el uso ordinario de un programa o guión (script) de línea de comandos	**`print()`**
Reportar eventos que ocurren durante el funcionamiento normal de un programa (por ejemplo, para la supervisión del estado o la investigación de fallos)	A logger's **`info()`** (or **`debug()`** method for very detailed output for diagnostic purposes)
Emitir una advertencia con respecto a un evento de tiempo de ejecución en particular	**`warnings.warn()`** en el código de la biblioteca si el problema es evitable y la aplicación cliente debe ser modificada para eliminar la advertencia A logger's **`warning()`** method if there is nothing the client application can do about the situation, but the event should still be noted
Reportar un error con respecto a un evento particular al tiempo de ejecución	Lanza una excepción
Reportar la supresión de un error sin invocar una excepción (por ejemplo, el manejador de errores en un proceso de servidor de larga duración)	A logger's **`error()`**, **`exception()`** or **`critical()`** method as appropriate for the specific error and application domain

The logger methods are named after the level or severity of the events they are used to track. The standard levels and their applicability are described below (in increasing order of severity):

Nivel	Cuando es usado
`DEBUG`	Información detallada, típicamente de interés sólo durante el diagnóstico de problemas.
`INFO`	Confirmación de que las cosas están funcionando como se esperaba.
`WARNING`	Un indicio de que algo inesperado sucedió, o indicativo de algún problema en el futuro cercano (por ejemplo, «espacio de disco bajo»). El software sigue funcionando como se esperaba.
`ERROR`	Debido a un problema más grave, el software no ha sido capaz de realizar alguna función.
`CRITICAL`	Un grave error, que indica que el programa en sí mismo puede ser incapaz de seguir funcionando.

The default level is `WARNING`, which means that only events of this severity and higher will be tracked, unless the logging package is configured to do otherwise.

Los eventos que se rastrean pueden ser manejados en diferentes maneras. La forma más simple de manejar los eventos rastreados es imprimirlos en la consola o terminal. Otra forma común es escribirlos en un archivo de disco.

Un simple ejemplo

Un ejemplo muy simple es:

```
import logging
logging.warning('Watch out!')  # will print a message to
logging.info('I told you so')  # will not print anything
```

Si escribes estas líneas en un script y lo ejecutas, verás:

```
WARNING:root:Watch out!
```

printed out on the console. The `INFO` message doesn’t appear because the default level is `WARNING`. The printed message includes the indication of the level and the description of the event provided in the logging call, i.e. “Watch out!”. The actual output can be formatted quite flexibly if you need that; formatting options will also be explained later.

Notice that in this example, we use functions directly on the `logging` module, like `logging.debug`, rather than creating a logger and calling functions on it. These functions operate on the root logger, but can be useful as they will call **`basicConfig()`** for you if it has not been called yet, like in this example. In larger programs you’ll usually want to control the logging configuration explicitly however - so for that reason as well as others, it’s better to create loggers and call their methods.

Logging a un archivo

Una situación muy común es la de registrar archivos de log, así que veamos eso a continuación. Asegúrese de intentar lo siguiente en un intérprete de Python recién iniciado, y no sólo continúe de la sesión descrita anteriormente:

```
import logging
logger = logging.getLogger(__name__)
logging.basicConfig(filename='example.log', encoding='ut
logger.debug('This message should go to the log file')
logger.info('So should this')
logger.warning('And this, too')
logger.error('And non-ASCII stuff, too, like Øresund anc
```

Distinto en la versión 3.9: Se agregó el argumento *encoding*. En versiones anteriores de Python, o si no se especifica, la codificación utilizada es el valor predeterminado utilizado por **`open()`**. Aunque no se muestra en el ejemplo anterior, ahora también se puede pasar un argumento *errors*, que determina cómo se manejan los errores de codificación. Para conocer los valores disponibles y los predeterminados, consulte la documentación de **`open()`**.

Y ahora si abrimos el archivo y miramos lo que tenemos, deberíamos encontrar los mensajes de registro:

```
DEBUG:__main__:This message should go to the log file
```

```
INFO:__main__:So should this
WARNING:__main__:And this, too
ERROR:__main__:And non-ASCII stuff, too, like Øresund ar
```

Este ejemplo también muestra cómo se puede establecer el nivel de registro que actúa como umbral para el rastreo. En este caso, como establecimos el umbral en `DEBUG`, todos los mensajes fueron impresos.

Si quieres establecer el nivel de registro desde una opción de línea de comandos como:

```
--log=INFO
```

y tienes el valor del parámetro pasado por `--log` en alguna variable *loglevel*, puedes usar:

```
getattr(logging, loglevel.upper())
```

para obtener el valor que pasarás a **`basicConfig()`** mediante el argumento *level*. Puede que quieras comprobar un error por cualquier valor de entrada del usuario, quizás como en el siguiente ejemplo:

```
# assuming loglevel is bound to the string value obtaine
# command line argument. Convert to upper case to allow
# specify --log=DEBUG or --log=debug
numeric_level = getattr(logging, loglevel.upper(), None)
if not isinstance(numeric_level, int):
    raise ValueError('Invalid log level: %s' % loglevel)
logging.basicConfig(level=numeric_level, ...)
```

The call to **`basicConfig()`** should come *before* any calls to a logger’s methods such as **`debug()`**, **`info()`**, etc. Otherwise, that logging event may not be handled in the desired manner.

Si ejecutas el script anterior varias veces, los mensajes de las ejecuciones sucesivas se añaden al archivo *example.log*. Si quieres que cada ejecución se inicie de nuevo, sin recordar los mensajes de ejecuciones anteriores, puedes especificar el argumento *filemode*, cambiando la llamada en el ejemplo anterior a:

```
logging.basicConfig(filename='example.log', filemode='w'
```

La impresión será la misma que antes, pero el archivo de registro ya no se adjunta, por lo que los mensajes de las ejecuciones anteriores se pierden.

Registrar datos de variables

Para registrar los datos de variables, utilice una cadena de formato para el mensaje de descripción del evento y añada los datos variables como argumentos. Por ejemplo:

```
import logging
logging.warning('%s before you %s', 'Look', 'leap!')
```

se mostrará:

```
WARNING:root:Look before you leap!
```

Como puede ver, la combinación de datos de variables en el mensaje de descripción del evento utiliza el antiguo estilo de formato de cadena con el signo de porcentaje (%). Esto se hace por razones de compatibilidad con versiones anteriores: el paquete de registro existe desde antes de las opciones de formato más nuevas como **`str.format()`** y **`string.Template`**. Estas opciones de formato más nuevas *son* soportadas, pero explorarlas está fuera del alcance de este tutorial: consulte Usar estilos de formato particulares en toda su aplicación para obtener más información.

Cambiar el formato de los mensajes mostrados

Para cambiar el formato que se utiliza para visualizar los mensajes, es necesario especificar el formato que se desea utilizar:

```
import logging
logging.basicConfig(format='%(levelname)s:%(message)s',
logging.debug('This message should appear on the console
logging.info('So should this')
logging.warning('And this, too')
```

que se imprimirá:

```
DEBUG:This message should appear on the console
INFO:So should this
WARNING:And this, too
```

Noten que la ‘root’ que aparecía en los ejemplos anteriores ha desaparecido. Para un conjunto completo de cosas que pueden aparecer en formato de cadenas, puede consultar la documentación de Atributos LogRecord, pero para un uso sencillo, sólo necesita el *levelname* (gravedad), *message* (descripción del evento, incluyendo los datos variables) y tal vez mostrar cuándo ocurrió el evento. Esto se describe en la siguiente sección.

Visualización de la fecha/hora en los mensajes

Para mostrar la fecha y la hora de un evento, usted colocaría “%(asctime)s” en su cadena de formato:

```
import logging
logging.basicConfig(format='%(asctime)s %(message)s')
logging.warning('is when this event was logged.')
```

que debería imprimir algo como esto:

```
2010-12-12 11:41:42,612 is when this event was logged.
```

El formato por defecto para la visualización de la fecha/hora (mostrado arriba) es como ISO8601 o **RFC 3339** [https://datatracker.ietf.org/doc/html/rfc3339.html]. Si necesita más control sobre el formato de la fecha/hora, proporcione un argumento *datefmt* a `basicConfig`, como en este ejemplo:

```
import logging
logging.basicConfig(format='%(asctime)s %(message)s', da
logging.warning('is when this event was logged.')
```

que mostraría algo como esto:

```
12/12/2010 11:46:36 AM is when this event was logged.
```

El formato del argumento *datefmt* es el mismo que el soportado por **`time.strftime()`**.

Próximos pasos

Eso concluye el tutorial básico. Debería ser suficiente para ponerte en marcha con el registro. Hay mucho más que el paquete de registro ofrece, pero para obtener lo mejor de él, tendrá que invertir un poco más de su tiempo en la lectura de las siguientes secciones. Si estás listo para eso, toma un poco de tu bebida favorita y sigue adelante.

If your logging needs are simple, then use the above examples to incorporate logging into your own scripts, and if you run into problems or don't understand something, please post a question in the Help category of the Python discussion forum [https://discuss.python.org/c/help/7] and you should receive help before too long.

¿Todavía esta aquí? Puedes seguir leyendo las siguientes secciones, que proporcionan un tutorial un poco más avanzado y profundo que el básico de arriba. Después de eso, puedes echar un vistazo al Libro de recetas de Logging.

Tutorial de registro avanzado

La biblioteca de logging adopta un enfoque modular y ofrece varias categorías de componentes: registradores, gestores, filtros y formateadores.

- Los registradores exponen la interfaz que el código de la aplicación utiliza directamente.
- Los gestores envían los registros de log (creados por los registradores) al destino apropiado.
- Los filtros proporcionan una instalación de grano más fino para determinar qué registros de log se deben producir.
- Los formatos especifican la disposición de los archivos de log en el resultado final.

La información de los eventos de registro se pasa entre los registradores, gestores, filtros y formateadores en una instancia `LogRecord`.

El registro se realiza llamando a métodos en instancias de la clase **`Logger`** (de aquí en adelante llamada *loggers*). Cada instancia tiene un nombre, y se organizan conceptualmente en una jerarquía de espacios de nombres utilizando puntos (puntos) como separadores. Por ejemplo, un registrador llamado "scan" es el padre de los registradores "scan.text", "scan.html" y "scan.pdf". Los nombres de los registradores pueden ser cualquier cosa que se desee, e indican el área de una aplicación en la que se origina un mensaje registrado.

Una buena convención que se puede utilizar para nombrar a los registradores es utilizar un registrador a nivel de módulo, en cada módulo que utilice el registro, llamado de la siguiente manera:

```
logger = logging.getLogger(__name__)
```

Esto significa que los nombres de los registradores rastrean la jerarquía de paquetes/módulos, y es intuitivamente obvio donde se registran los eventos sólo a partir del nombre del registrador.

La raíz de la jerarquía de los registradores se llama root *logger*. Ese es el registrador usado por las funciones **`debug()`**, **`info()`**, **`warning()`**, **`error()`** y **`critical()`**, que sólo llaman al mismo método del registrador raíz. Las funciones y los métodos tienen las mismas firmas. El nombre del root *logger* se imprime como 'root' en la salida registrada.

Por supuesto, es posible registrar mensajes a diferentes destinos. El paquete incluye soporte para escribir mensajes de registro en archivos, ubicaciones HTTP GET/POST, correo electrónico a través de SMTP, sockets genéricos, colas o mecanismos de registro específicos del sistema operativo como syslog o el registro de eventos de Windows NT. Los destinos son servidos por clases *handler*. Puedes crear tu propia clase de destino de registro si tienes requisitos especiales que no se cumplen con ninguna de las clases de gestor incorporadas.

Por defecto, no se establece ningún destino para los mensajes de registro. Puede especificar un destino (como consola o archivo) usando **`basicConfig()`** como en los ejemplos del tutorial. Si llama a las funciones **`debug()`**, **`info()`**, **`warning()`**, **`error()`**

y **`critical()`**, ellas comprobarán si no hay ningún destino establecido; y si no hay ninguno establecido, establecerán un destino de la consola (`sys.stderr`) y un formato por defecto para el mensaje mostrado antes de delegar en el registrador root para hacer la salida real del mensaje.

El formato por defecto establecido por **`basicConfig()`** para los mensajes es:

```
severity:logger name:message
```

Puedes cambiar esto pasando una cadena de formato a **`basicConfig()`** con el argumento de la palabra clave *format*. Para todas las opciones relativas a cómo se construye una cadena de formato, ver Objetos formateadores.

Flujo de Registro

En el siguiente diagrama se ilustra el flujo de información de los eventos de registro en los registradores y gestores.

Registradores

Los objetos de **`Logger`** tienen un trabajo triple. Primero, exponen varios métodos al código de la aplicación para que las aplicaciones puedan registrar mensajes en tiempo de ejecución. Segundo, los objetos *logger* determinan sobre qué mensajes de registro actuar en base de la severidad (la facilidad de filtrado por defecto) o los objetos de filtro. Tercero, los objetos registradores pasan los mensajes de registro relevantes a todos los manejadores de registro interesados.

Los métodos más utilizados en los objetos de registro se dividen en dos categorías: configuración y envío de mensajes.

Estos son los métodos de configuración más comunes:

- **`Logger.setLevel()`** especifica el mensaje de registro de menor gravedad que un registrador manejará, donde debug es el nivel de gravedad incorporado más bajo y *critical* es el de mayor gravedad incorporado. Por ejemplo, si el nivel de severidad es INFO, el registrador sólo manejará los mensajes INFO, WARNING, ERROR y CRITICAL e ignorará los mensajes DEBUG.

- **`Logger.addHandler()`** y **`Logger.removeHandler()`** agregan y quitan los objetos *handler* del objeto *logger*. Los manejadores (*handlers*) se tratan con más detalle en Gestores.
- **`Logger.addFilter()`** y **`Logger.removeFilter()`** agregan y quitan los objetos de filtro del objeto logger. Los filtros se tratan con más detalle en Filtro de Objetos.

No es necesario que siempre llames a estos métodos en cada registrador que crees. Vea los dos últimos párrafos de esta sección.

Con el objeto *logger* configurado, los siguientes métodos crean mensajes de log:

- **`Logger.debug()`**, **`Logger.info()`**, **`Logger.warning()`**, **`Logger.error()`**, y **`Logger.critical()`** todos crean registros de registro con un mensaje y un nivel que corresponde a sus respectivos nombres de método. El mensaje es en realidad una cadena de formato, que puede contener la sintaxis estándar de sustitución de cadenas de `%s`, `%d`, `%f`, y así sucesivamente. El resto de sus argumentos es una lista de objetos que se corresponden con los campos de sustitución del mensaje. Con respecto a `**kwargs`, los métodos de registro sólo se preocupan por una palabra clave de `exc_info` y la usan para determinar si registran información de excepción.
- **`Logger.exception()`** crea un mensaje de registro similar a **`Logger.error()`**. La diferencia es que **`Logger.exception()`** vuelca un rastro de pila junto con él. Llama a este método sólo desde un manejador de excepciones.
- **`Logger.log()`** toma un nivel de log como argumento explícito. Esto es un poco más verboso para el registro de mensajes que usar los métodos de conveniencia de nivel de registro listados arriba, pero así es como se registra en niveles de registro personalizados.

`getLogger()` retorna una referencia a una instancia de *logger* con el nombre especificado si se proporciona, o `root` si no. Los nombres son estructuras jerárquicas separadas por períodos. Múltiples llamadas a **`getLogger()`** con el mismo nombre retornarán una referencia al mismo objeto *logger*. Los *loggers* que están más abajo en la lista jerárquica son hijos de los *loggers* que están más arriba en la lista. Por ejemplo, dado un *logger* con un nombre de `foo`, los *loggers* con nombres de `foo.bar`, `foo.bar.baz`, y `foo.bam` son todos descendientes de `foo`.

Los registradores tienen un concepto de *nivel efectivo*. Si un nivel no se establece explícitamente en un registrador, el nivel de su clase padre se utiliza en su lugar como su nivel efectivo. Si el padre no tiene un nivel explícito establecido, *su* padre es examinado, y así sucesivamente - se buscan todos los ancestros hasta que se encuentra un nivel explícitamente establecido. El registrador raíz siempre tiene un conjunto de niveles explícito (`Advertencia` por defecto). Cuando se decide si se procesa un evento, el nivel efectivo del registrador se utiliza para determinar si el evento se pasa a los manejadores del registrador.

Los *loggers* inferiores propagan mensajes hasta los gestores asociados con sus *loggers* ancestros. Debido a esto, no es necesario definir y configurar los manejadores para todos los registradores que utiliza una aplicación. Basta con configurar los manejadores para un registrador de nivel superior y crear registradores hijos según sea necesario. (Sin embargo, puedes desactivar la propagación estableciendo el atributo *propagate* de un *logger* en `False`.)

Gestores

Los objetos **`Handler`** son responsables de enviar los mensajes de registro apropiados (basados en la severidad de los mensajes de registro) al destino especificado por el handler. **`Logger`** los objetos pueden añadir cero o más objetos *handler* a sí mismos con un método **`addHandler()`**. Como escenario de ejemplo, una aplicación puede querer enviar todos los mensajes de registro a un archivo de registro, todos los mensajes de registro de error o superiores a stdout, y todos los mensajes de crítico a una dirección de correo electrónico. Este escenario requiere tres manejadores individuales donde cada manejador es responsable de enviar mensajes de una severidad específica a una ubicación específica.

La biblioteca estándar incluye bastantes tipos de *handler* (ver Gestores útiles); los tutoriales usan principalmente `StreamHandler` y `FileHandler` en sus ejemplos.

Hay muy pocos métodos en un manejador para que los desarrolladores de aplicaciones se preocupen. Los únicos métodos de manejador que parecen relevantes para los desarrolladores de aplicaciones que utilizan los objetos de manejador incorporados (es decir, que no crean manejadores personalizados) son los siguientes métodos de configuración:

- The `setLevel()` method, just as in logger objects, specifies the lowest severity that will be dispatched to the appropriate destination. Why are there two `setLevel()` methods? The level set in the logger determines which severity of messages it will pass to its handlers. The level set in each handler determines which messages that handler will send on.
- `setFormatter()` selecciona un objeto *Formatter* para que este *handler* lo use.
- `addFilter()` y `removeFilter()` respectivamente configuran y desconfiguran los objetos del filtro en los handlers.

El código de la aplicación no debe instanciar directamente y usar instancias de `Handler`. En su lugar, la clase `Handler` es una clase base que define la interfaz que todos los *handlers* deben tener y establece algún comportamiento por defecto que las clases hijas pueden usar (o anular).

Formateadores

Los objetos de formato configuran el orden final, la estructura y el contenido del mensaje de registro. A diferencia de la clase base `logging.Handler`, el código de la aplicación puede instanciar clases de formateo, aunque probablemente podría subclasificar el formateo si su aplicación necesita un comportamiento especial. El constructor toma tres argumentos opcionales – una cadena de formato de mensaje, una cadena de formato de fecha y un indicador de estilo.

logging.Formatter.__init__(*fmt=None*, *datefmt=None*, *style='%'*)

Si no hay una cadena de formato de mensaje, el valor predeterminado es utilizar el mensaje en bruto. Si no hay una cadena de formato de fecha, el formato de fecha por defecto es:

```
%Y-%m-%d %H:%M:%S
```

con los milisegundos insertados al final. El `style` es uno de `'%'`, `'{'` o `'$'`. Si uno de estos no se especifica, entonces se usará `'%'`.

Si el `style` es `'%'`, la cadena del formato de mensaje utiliza `%(<dictionary key>)s` estilo de sustitución de cadena; las posibles llaves están documentadas en Atributos LogRecord. Si el estilo es `'{'`, se asume que la cadena del formato del mensaje es compatible con **`str.format()`** (usando argumentos de palabras clave), mientras que si el estilo es `'$'` entonces la cadena del formato del mensaje debe ajustarse a lo que se espera de **`string.Template.substitute()`**.

Distinto en la versión 3.2: Añadió el parámetro `style`.

La siguiente cadena de formato de mensaje registrará la hora en un formato legible para los humanos, la gravedad del mensaje y el contenido del mensaje, en ese orden:

```
'%(asctime)s - %(levelname)s - %(message)s'
```

Los formateadores utilizan una función configurable por el usuario para convertir la hora de creación de un registro en una tupla. Por defecto, se utiliza **`time.localtime()`**; para cambiar esto para una instancia de formateador particular, establezca el atributo `converter` de la instancia a una función con la misma firma que **`time.localtime()`** o **`time.gmtime()`**. Para cambiarlo para todos los formateadores, por ejemplo si quieres que todas las horas de registro se muestren en GMT, establece el atributo `converter` en la clase *Formatter* (a `time.gmtime` para mostrar GMT).

Configuración del registro

Los programadores pueden configurar el registro en tres maneras:

1. Creando registradores, manejadores y formateadores explícitamente usando código Python que llama a los métodos de configuración listados arriba.
2. Creando un archivo de configuración de registro y leyéndolo usando la función `fileConfig()`.
3. Creando un diccionario de información de configuración y pasándolo a la función `dictConfig()`.

Para la documentación de referencia sobre las dos últimas opciones, vea Funciones de configuración. El siguiente ejemplo configura un *logger* muy simple, un manejador de consola, y un formateador simple usando código Python:

```
import logging

# create logger
logger = logging.getLogger('simple_example')
logger.setLevel(logging.DEBUG)

# create console handler and set level to debug
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG)

# create formatter
formatter = logging.Formatter('%(asctime)s - %(name)s -

# add formatter to ch
ch.setFormatter(formatter)

# add ch to logger
logger.addHandler(ch)

# 'application' code
logger.debug('debug message')
logger.info('info message')
logger.warning('warn message')
logger.error('error message')
logger.critical('critical message')
```

Ejecutar este módulo desde la línea de comandos produce la siguiente salida:

```
$ python simple_logging_module.py
2005-03-19 15:10:26,618 - simple_example - DEBUG - debu
2005-03-19 15:10:26,620 - simple_example - INFO - info m
2005-03-19 15:10:26,695 - simple_example - WARNING - war
2005-03-19 15:10:26,697 - simple_example - ERROR - error
2005-03-19 15:10:26,773 - simple_example - CRITICAL - cr
```

El siguiente módulo de Python crea un registrador, manejador y formateador casi idéntico a los del ejemplo anterior, con la única diferencia de los nombres de los objetos:

```
import logging
import logging.config

logging.config.fileConfig('logging.conf')

# create logger
logger = logging.getLogger('simpleExample')

# 'application' code
logger.debug('debug message')
logger.info('info message')
logger.warning('warn message')
logger.error('error message')
logger.critical('critical message')
```

Aquí está el archivo logging.conf:

```
[loggers]
keys=root,simpleExample

[handlers]
keys=consoleHandler

[formatters]
```

```
keys=simpleFormatter

[logger_root]
level=DEBUG
handlers=consoleHandler

[logger_simpleExample]
level=DEBUG
handlers=consoleHandler
qualname=simpleExample
propagate=0

[handler_consoleHandler]
class=StreamHandler
level=DEBUG
formatter=simpleFormatter
args=(sys.stdout,)

[formatter_simpleFormatter]
format=%(asctime)s - %(name)s - %(levelname)s - %(messag
```

La salida es casi idéntica a la del ejemplo basado en un archivo no configurado:

```
$ python simple_logging_config.py
2005-03-19 15:38:55,977 - simpleExample - DEBUG - debug
2005-03-19 15:38:55,979 - simpleExample - INFO - info me
2005-03-19 15:38:56,054 - simpleExample - WARNING - warr
2005-03-19 15:38:56,055 - simpleExample - ERROR - error
2005-03-19 15:38:56,130 - simpleExample - CRITICAL - cri
```

Se puede ver que el enfoque del archivo de configuración tiene algunas ventajas sobre el enfoque del código Python, principalmente la separación de la configuración y el código y la capacidad de los no codificadores de modificar fácilmente las propiedades de registro.

Advertencia

> La función ``fileConfig` ()`` toma un parámetro por defecto, `disable_existing_loggers`, que por defecto es `True` por razones de compatibilidad retroactiva. Esto puede ser o no lo que usted quiera, ya que causará que cualquier registrador no existente antes de la llamada `fileConfig()` sea desactivado a menos que ellos (o un ancestro) sean nombrados explícitamente en la configuración. Por favor, consulte la documentación de referencia para más información, y especifique `False` para este parámetro si lo desea.
>
> El diccionario pasado a `dictConfig()` también puede especificar un valor booleano con la tecla `disable_existing_loggers`, que si no se especifica explícitamente en el diccionario también se interpreta por defecto como `True`. Esto lleva al comportamiento de deshabilitación de los registradores descrito anteriormente, que puede no ser lo que usted desea - en cuyo caso, proporcione a la clave explícitamente un valor de `False`.

Obsérvese que los nombres de clase a los que se hace referencia en los archivos de configuración deben ser relativos al módulo de registro, o bien valores absolutos que puedan resolverse mediante mecanismos de importación normales. Por lo tanto, puedes usar `WatchedFileHandler` (relativo al módulo de registro) o `mypackage.mymodule.MyHandler` (para una clase definida en el paquete `mypackage` y el módulo `mymodule`, donde `mypackage` está disponible en la ruta de importación de Python).

En Python 3.2, se ha introducido un nuevo medio para configurar el registro, utilizando diccionarios para guardar la información de configuración. Esto proporciona un superconjunto de la funcionalidad del enfoque basado en archivos de configuración descrito anteriormente, y es el método de configuración recomendado para nuevas aplicaciones y despliegues. Dado que se utiliza un diccionario de Python para guardar información de configuración, y dado que se puede rellenar ese diccionario utilizando diferentes medios, se dispone de más opciones de configuración. Por ejemplo, puede utilizar un archivo de configuración en formato JSON o, si tiene acceso a la funcionalidad de procesamiento YAML, un archivo en formato YAML, para rellenar el diccionario de configuración. O, por supuesto, puedes construir el diccionario en código Python, recibirlo en forma encurtida sobre un zócalo, o usar cualquier enfoque que tenga sentido para tu aplicación.

Aquí hay un ejemplo de la misma configuración que arriba, en formato YAML para el nuevo enfoque basado en el diccionario:

```
version: 1
formatters:
  simple:
    format: '%(asctime)s - %(name)s - %(levelname)s - %
handlers:
  console:
    class: logging.StreamHandler
    level: DEBUG
    formatter: simple
    stream: ext://sys.stdout
loggers:
  simpleExample:
    level: DEBUG
    handlers: [console]
    propagate: no
root:
  level: DEBUG
  handlers: [console]
```

Para más información sobre el registro usando un diccionario, ver Funciones de configuración.

¿Qué pasa si no se proporciona ninguna configuración

If no logging configuration is provided, it is possible to have a situation where a logging event needs to be output, but no handlers can be found to output the event.

The event is output using a “handler of last resort”, stored in `lastResort`. This internal handler is not associated with any logger, and acts like a `StreamHandler` which writes the event description message to the current value of `sys.stderr` (therefore respecting any redirections which may be in effect). No formatting is done on the message - just the bare event description message is printed. The handler's level is set to `WARNING`, so all events at this and greater severities will be output.

Distinto en la versión 3.2: Para las versiones de Python anteriores a la 3.2, el comportamiento es el siguiente:

- If **`raiseExceptions`** is `False` (production mode), the event is silently dropped.
- If **`raiseExceptions`** is `True` (development mode), a message "No handlers could be found for logger X.Y.Z" is printed once.

To obtain the pre-3.2 behaviour, **`lastResort`** can be set to `None`.

Configurando Logging para una biblioteca

Cuando se desarrolla una biblioteca que utiliza el registro, se debe tener cuidado de documentar la forma en que la biblioteca utiliza el registro, por ejemplo, los nombres de los registradores utilizados. También hay que tener en cuenta su configuración de registro. Si la aplicación que lo utiliza no usa el registro, y el código de la biblioteca hace llamadas de registro, entonces (como se describe en la sección anterior) los eventos de gravedad `WARNING` y mayores se imprimirán en `sys.stderr`. Esto se considera el mejor comportamiento por defecto.

Si por alguna razón usted *no* quiere que estos mensajes se impriman en ausencia de cualquier configuración de registro, puede adjuntar un manejador de no hacer nada al registrador de nivel superior de su biblioteca. Esto evita que el mensaje se imprima, ya que siempre se encontrará un manejador para los eventos de la biblioteca: simplemente no produce ninguna salida. Si el usuario de la biblioteca configura el registro para el uso de la aplicación, presumiblemente esa configuración añadirá algunos manejadores, y si los niveles están configurados adecuadamente, entonces las llamadas de registro realizadas en el código de la biblioteca enviarán una salida a esos manejadores, como es normal.

Un manejador de no hacer nada está incluido en el paquete de registro: **`NullHandler`** (desde Python 3.1). Una instancia de este manejador podría ser añadida al *logger* de nivel superior del espacio de nombres de registro usado por la biblioteca (*si* quieres evitar que los eventos de registro de tu biblioteca se envíen a `sys.stderr` en ausencia de la configuración de registro). Si todo el registro de una biblioteca *foo* se hace usando registradores con nombres que coincidan con “foo.x”, “foo.x.y”, etc. entonces el código:

```
import logging
logging.getLogger('foo').addHandler(logging.NullHandler(
```

debería tener el efecto deseado. Si una organización produce varias bibliotecas, el nombre del registrador especificado puede ser ‘orgname.foo’ en lugar de sólo ‘foo’.

Nota

Se recomienda encarecidamente que *no registre en el registrador raíz* en su biblioteca. En su lugar, utilice un registrador con un nombre único y fácilmente identificable, como el `__name__` para el paquete o módulo de nivel superior de su biblioteca. Al registrar en el registrador raíz, dificultará o será imposible para el desarrollador de la aplicación configurar la verbosidad o los handlers de registro de su biblioteca según sus preferencias.

Nota

Se recomienda encarecidamente que *no añada ningún otro manejador que no sea* **`NullHandler`** *a los loggers de su biblioteca*. Esto se debe a que la configuración de los *handlers* es prerrogativa del desarrollador de aplicaciones que utiliza su biblioteca. El desarrollador de la aplicación conoce su público objetivo y qué manejadores son los más apropiados para su aplicación: si añades manejadores ‘bajo el capó’, podrías interferir en su capacidad de realizar pruebas unitarias y entregar registros que se ajusten a sus necesidades.

Niveles de registro

Los valores numéricos de los niveles de registro se indican en el siguiente cuadro. Éstos son de interés principalmente si se desea definir los propios niveles y se necesita que tengan valores específicos en relación con los niveles predefinidos. Si se define un nivel con el mismo valor numérico, éste sobrescribe el valor predefinido; el nombre predefinido se pierde.

Nivel	Valor numérico
`CRITICAL`	50
`ERROR`	40
`WARNING`	30
`INFO`	20
`DEBUG`	10
`NOTSET`	0

Los niveles también pueden asociarse con los registradores, siendo establecidos por el desarrollador o mediante la carga de una configuración de registro guardada. Cuando se llama a un método de registro en un registrador, éste compara su propio nivel con el nivel asociado a la llamada del método. Si el nivel del registrador es superior al de la llamada al método, no se genera ningún mensaje de registro. Este es el mecanismo básico que controla la verbosidad de la salida del registro.

Los mensajes de registro se codifican como instancias de la clase `logging.LogRecord`. Cuando un *logger* decide registrar realmente un evento, se crea una instancia `LogRecord` a partir del mensaje de registro.

Los mensajes de registro están sujetos a un mecanismo de envío mediante el uso de *handlers*, que son instancias de subclases de la clase `Handler`. Los gestores son responsables de asegurar que un mensaje registrado (en forma de `LogRecord`) termine en una ubicación particular (o conjunto de ubicaciones) que sea útil para el público al que va dirigido ese mensaje (como usuarios finales, personal de asistencia técnica, administradores de sistemas, desarrolladores). Los manejadores pasan instancias `LogRecord` destinadas a destinos particulares. Cada *logger* puede tener cero, uno o más manejadores asociados a él (a través del método `addHandler()` de `Logger`). Además de los *handlers* directamente asociados a un *logger*, todos los manejadores (*handlers*) asociados a todos los ancestros del *logger* son llamados a enviar el mensaje (a menos que el flag *propagate* de un *logger* se establezca en un valor falso, en cuyo caso el paso a los *handlers* ancestrales se detiene).

Al igual que para los *logger*, los gestores pueden tener niveles asociados a ellos. El nivel de un gestor actúa como un filtro de la misma manera que el nivel de un *logger*. Si un manejador (*handler*) decide realmente enviar un evento, el método `emit()` se utiliza para enviar el mensaje a su destino. La mayoría de las subclases definidas por el usuario de `Handler` necesitarán anular este `emit()`.

Niveles personalizados

Definir sus propios niveles es posible, pero no debería ser necesario, ya que los niveles existentes se han elegido sobre la base de la experiencia práctica. Sin embargo, si usted está convencido de que necesita niveles personalizados, debe tener mucho cuidado al hacer esto, y es posiblemente *una muy mala idea definir niveles personalizados si está desarrollando una biblioteca*. Esto se debe a que si los autores de múltiples bibliotecas definen sus propios niveles personalizados, existe la posibilidad de que el resultado del registro de tales bibliotecas múltiples utilizadas conjuntamente sea difícil de controlar y/o interpretar para el desarrollador usuario, porque un valor numérico dado podría significar cosas diferentes para diferentes bibliotecas.

Gestores útiles

Además de la base `Handler` class, se proporcionan muchas subclases útiles:

1. `StreamHandler` instancias envían mensajes a los *streams* (objetos como de tipo archivo).
2. `FileHandler` instancias enviar mensajes a los archivos del disco.
3. `BaseRotatingHandler` es la clase base para los

manejadores (*handlers*) que rotan los archivos de registro en un punto determinado. No está pensada para ser instanciada directamente. En su lugar, utilice `RotatingFileHandler` o `TimedRotatingFileHandler`.

4. Las instancias de `RotatingFileHandler` envían mensajes a los archivos de disco, con soporte para el tamaño máximo de los archivos de registro y la rotación de los mismos.
5. Las instancias de `TimedRotatingFileHandler` envían mensajes a los archivos de disco, rotando el archivo de registro a ciertos intervalos de tiempo.
6. Las instancias de `SocketHandler` envían mensajes a los sockets TCP/IP. Desde la versión 3.4, los sockets de dominio Unix también están soportados.
7. Instancias de `DatagramHandler` envían mensajes a los sockets UDP. Desde la versión 3.4, los sockets de dominio Unix también están soportados.
8. Las instancias de `SMTPHandler` envían mensajes a una dirección de correo electrónico designada.
9. Las instancias de `SysLogHandler` envían mensajes a un demonio del syslog de Unix, posiblemente en una máquina remota.
10. Las instancias de `NTEventLogHandler` envían mensajes a un registro de eventos de Windows NT/2000/XP.
11. Las instancias de `MemoryHandler` envían mensajes a un buffer en la memoria, que es limpiado cuando se cumplen ciertos criterios.
12. Las instancias de `HTTPHandler` envían mensajes a un servidor HTTP usando la semántica de «GET» o «POST».
13. Las instancias de `WatchedFileHandler` ven el archivo al que están accediendo. Si el archivo cambia, se cierra y se vuelve a abrir usando el nombre del archivo. Este manejador sólo es útil en sistemas tipo Unix; Windows no soporta el mecanismo subyacente utilizado.
14. Las instancias de `QueueHandler` envían mensajes a una cola, como los implementados en los módulos `queue` or `multiprocessing`.
15. Las instancias de `NullHandler` no hacen nada con los mensajes de error. Son utilizadas por los desarrolladores de bibliotecas que desean utilizar el registro, pero quieren evitar el mensaje "No se pudo encontrar ningún controlador para el

registrador *XXX*", que puede mostrarse si el usuario de la biblioteca no ha configurado el registro. Consulte Configurando Logging para una biblioteca para obtener más información.

Added in version 3.1: La clase **`NullHandler`**.

Added in version 3.2: La **`QueueHandler`** (La clase de gestores de Cola).

Las clases **`NullHandler`**, **`StreamHandler`** y **`FileHandler`** están definidas en el paquete de registro del núcleo. Los otros manejadores se definen en un sub-módulo, **`logging.handlers`**. (También hay otro submódulo, **`logging.config`**, para la funcionalidad de configuración)

Los mensajes registrados se formatean para su presentación a través de instancias de la clase **`Formatter`**. Se inicializan con una cadena de formato adecuada para su uso con el operador % y un diccionario.

For formatting multiple messages in a batch, instances of **`BufferingFormatter`** can be used. In addition to the format string (which is applied to each message in the batch), there is provision for header and trailer format strings.

Cuando el filtrado basado en el nivel de *logger* o el nivel de manejador (*handler*) no es suficiente, se pueden añadir instancias de **`Filter`** tanto a **`Logger`** como a **`Handler`** instancias (a través de su método **`addFilter()`**). Antes de decidir procesar un mensaje más adelante, tanto los *loggers* como los manejadores (*handlers*) consultan todos sus filtros para obtener permiso. Si algún filtro retorna un valor falso, el mensaje no se procesa más.

La funcionalidad básica **`Filtro`** permite filtrar por un nombre de registro específico. Si se utiliza esta función, los mensajes enviados al registrador nombrado y a sus hijos se permiten a través del filtro, y todos los demás se eliminan.

Excepciones lanzadas durante logging

El paquete de tala está diseñado para tragarse las excepciones que se producen durante la tala en la producción. Esto es así para que los errores que ocurren durante el manejo de los eventos de registro - como la mala configuración del registro, errores de red u otros errores similares - no causen que la aplicación que utiliza el registro termine prematuramente.

Las excepciones de `SystemExit` (Salida del sistema) y `KeyboardInterrupt` (Interrupción del teclado) nunca se tragan. Otras excepciones que ocurren durante el método `emit()` de una subclase `Handler` se pasan a su método `handleError()` .

La implementación por defecto de `handleError()` en `Handler` comprueba si una variable de nivel de módulo, `raiseExceptions`, está establecida. Si se establece, se imprime una traza en `sys.stderr`. Si no se establece, se traga la excepción.

Nota

El valor por defecto de `raiseExceptions` (lanzar excepciones) es `True`. Esto se debe a que durante el desarrollo, normalmente quieres ser notificado de cualquier excepción que ocurra. Se aconseja que establezca `raiseExceptions` a `False` para el uso en producción.

Usando objetos arbitrarios como mensajes

En las secciones y ejemplos anteriores, se ha supuesto que el mensaje pasado al registrar el suceso es una cadena. Sin embargo, esta no es la única posibilidad. Se puede pasar un objeto arbitrario como mensaje, y su método `__str__()` será llamado cuando el sistema de registro necesite convertirlo en una representación de cadena. De hecho, si quieres, puedes evitar computar una representación de cadena por completo - por ejemplo, el método `SocketHandler` emite un evento al *pickling* y enviarlo por el cable.

Optimización

El formato de los argumentos del mensaje se aplaza hasta que no se pueda evitar. Sin embargo, el cálculo de los argumentos pasados al método de registro también puede ser costoso, y puede que quieras evitar hacerlo si el registrador simplemente tirará tu evento. Para decidir qué hacer, puedes llamar al método **`isEnabledFor()`** que toma un argumento de nivel y retorna true si el evento sería creado por el *Logger* para ese nivel de llamada. Puedes escribir código como este:

```
if logger.isEnabledFor(logging.DEBUG):
    logger.debug('Message with %s, %s', expensive_func1(
                                         expensive_func2(
```

so that if the logger's threshold is set above `DEBUG`, the calls to `expensive_func1` and `expensive_func2` are never made.

Nota

En algunos casos, **`isEnabledFor()`** puede ser en sí mismo más caro de lo que te gustaría (por ejemplo, para los *loggers* profundamente anidados donde un nivel explícito sólo se establece en lo alto de la jerarquía de *loggers*). En estos casos (o si quieres evitar llamar a un método en bucles estrechos), puedes guardar en caché el resultado de una llamada a **`isEnabledFor()`** en una variable local o de instancia, y usarla en lugar de llamar al método cada vez. Tal valor en caché sólo necesitaría ser recalculado cuando la configuración de registro cambie dinámicamente mientras la aplicación se está ejecutando (lo cual no es tan común).

Hay otras optimizaciones que pueden hacerse para aplicaciones específicas que necesitan un control más preciso sobre la información de registro que se recoge. Aquí hay una lista de cosas que puede hacer para evitar el procesamiento durante el registro que no necesita:

Qué no quieres colectar	Cómo evitar coleccionarlo
Información sobre dónde se hicieron las llamadas.	Establezca `logging._srcfile` en `None`. Esto evita llamar a **`sys._getframe()`**, que puede ayudar a acelerar su código en entornos como PyPy (que no puede acelerar el código que usa

	`sys._getframe()`).
Información sobre código con hilos.	Establece `logging.logThreads` en `False`.
ID del proceso actual (`os.getpid()`)	Establece `logging.logProcesses` en `False`.
Nombre del proceso actual cuando se usa `multiprocessing` para administrar múltiples procesos.	Establece `logging.logMultiprocessing` en `False`.
Nombre actual de `asyncio.Task` cuando se utiliza `asyncio`.	Establece `logging.logAsyncioTasks` en `False`.

Observe también que el módulo de registro del núcleo sólo incluye los gestores básicos. Si no importas **`logging.handlers`** y **`logging.config`**, no ocuparán ninguna memoria.

Otros recursos

Ver también

Módulo `logging`
Referencia API para el módulo de registro.

Módulo `logging.config`
API de configuración para el módulo de registro.

Módulo `logging.handlers`
Gestores útiles incluidos en el módulo de registro.

Un libro de recetas

Libro de recetas de Logging

Autor:

Vinay Sajip <vinay_sajip at red-dove dot com>

Esta página contiene un número de recetas sobre *logging*, que han sido útiles en el pasado. Para obtener enlaces al tutorial e información de referencia, consulte Otros recursos.

Usar logging en múltiples módulos

Múltiples llamadas a `logging.getLogger('someLogger')` retornan una referencia al mismo objeto logger. Esto es cierto no solo dentro del mismo módulo, sino también en todos los módulos siempre que estén ejecutándose en el mismo proceso del intérprete de Python. Es válido para las referencias al mismo objeto. Además, el código de la aplicación puede definir y configurar un logger primario en un módulo y crear (pero no configurar) un logger secundario en un módulo separado, y todas las llamadas al secundario pasarán al principal. A continuación un módulo principal:

```
import logging
import auxiliary_module

# create logger with 'spam_application'
logger = logging.getLogger('spam_application')
logger.setLevel(logging.DEBUG)
# create file handler which logs even debug messages
fh = logging.FileHandler('spam.log')
fh.setLevel(logging.DEBUG)
# create console handler with a higher log level
ch = logging.StreamHandler()
ch.setLevel(logging.ERROR)
```

```
# create formatter and add it to the handlers
formatter = logging.Formatter('%(asctime)s - %(name)s -
fh.setFormatter(formatter)
ch.setFormatter(formatter)
# add the handlers to the logger
logger.addHandler(fh)
logger.addHandler(ch)

logger.info('creating an instance of auxiliary_module.Au
a = auxiliary_module.Auxiliary()
logger.info('created an instance of auxiliary_module.Aux
logger.info('calling auxiliary_module.Auxiliary.do_somet
a.do_something()
logger.info('finished auxiliary_module.Auxiliary.do_some
logger.info('calling auxiliary_module.some_function()')
auxiliary_module.some_function()
logger.info('done with auxiliary_module.some_function()'
```

Y aquí un módulo auxiliar:

```
import logging

# create logger
module_logger = logging.getLogger('spam_application.auxi

class Auxiliary:
    def __init__(self):
        self.logger = logging.getLogger('spam_applicatio
        self.logger.info('creating an instance of Auxili

    def do_something(self):
        self.logger.info('doing something')
        a = 1 + 1
        self.logger.info('done doing something')

def some_function():
    module_logger.info('received a call to "some_functio
```

El resultado se ve así:

```
2005-03-23 23:47:11,663 - spam_application - INFO -
   creating an instance of auxiliary_module.Auxiliary
2005-03-23 23:47:11,665 - spam_application.auxiliary.Aux
   creating an instance of Auxiliary
2005-03-23 23:47:11,665 - spam_application - INFO -
   created an instance of auxiliary_module.Auxiliary
2005-03-23 23:47:11,668 - spam_application - INFO -
   calling auxiliary_module.Auxiliary.do_something
2005-03-23 23:47:11,668 - spam_application.auxiliary.Aux
   doing something
2005-03-23 23:47:11,669 - spam_application.auxiliary.Aux
   done doing something
2005-03-23 23:47:11,670 - spam_application - INFO -
   finished auxiliary_module.Auxiliary.do_something
2005-03-23 23:47:11,671 - spam_application - INFO -
   calling auxiliary_module.some_function()
2005-03-23 23:47:11,672 - spam_application.auxiliary - I
   received a call to 'some_function'
2005-03-23 23:47:11,673 - spam_application - INFO -
   done with auxiliary_module.some_function()
```

Logging desde múltiples hilos

Realizar *logging* desde múltiples hilos (*threads*) no requiere ningún esfuerzo especial. El siguiente ejemplo muestra el logging desde el hilo principal (inicial) y otro hilo:

```
import logging
import threading
import time

def worker(arg):
    while not arg['stop']:
        logging.debug('Hi from myfunc')
        time.sleep(0.5)

def main():
    logging.basicConfig(level=logging.DEBUG, format='%(r
    info = {'stop': False}
    thread = threading.Thread(target=worker, args=(info,
    thread.start()
    while True:
        try:
            logging.debug('Hello from main')
            time.sleep(0.75)
        except KeyboardInterrupt:
            info['stop'] = True
            break
    thread.join()

if __name__ == '__main__':
    main()
```

Cuando se ejecuta, el script debe imprimir algo como lo siguiente:

```
   0 Thread-1 Hi from myfunc
   3 MainThread Hello from main
 505 Thread-1 Hi from myfunc
 755 MainThread Hello from main
1007 Thread-1 Hi from myfunc
1507 MainThread Hello from main
1508 Thread-1 Hi from myfunc
2010 Thread-1 Hi from myfunc
2258 MainThread Hello from main
2512 Thread-1 Hi from myfunc
3009 MainThread Hello from main
3013 Thread-1 Hi from myfunc
3515 Thread-1 Hi from myfunc
3761 MainThread Hello from main
4017 Thread-1 Hi from myfunc
4513 MainThread Hello from main
4518 Thread-1 Hi from myfunc
```

Esto muestra la salida de logging intercalada como cabría esperar. Por supuesto, este enfoque funciona para más hilos de lo que se muestran aquí.

Múltiples gestores y formateadores

Los *loggers* son simples objetos Python. El método `addHandler()` no tiene una cuota mínima o máxima para la cantidad de gestores (*handlers*) que puede agregar. A veces será beneficioso para una aplicación registrar todos los mensajes de todas las prioridades en un archivo de texto mientras se registran simultáneamente los errores o más en la consola. Para configurar esto, simplemente configure los gestores apropiados. Las llamadas de logging en el código de la aplicación permanecerán sin cambios. Aquí hay una ligera modificación al ejemplo de configuración simple anterior basado en módulo:

```
import logging

logger = logging.getLogger('simple_example')
logger.setLevel(logging.DEBUG)
# create file handler which logs even debug messages
fh = logging.FileHandler('spam.log')
fh.setLevel(logging.DEBUG)
# create console handler with a higher log level
ch = logging.StreamHandler()
ch.setLevel(logging.ERROR)
# create formatter and add it to the handlers
formatter = logging.Formatter('%(asctime)s - %(name)s -
ch.setFormatter(formatter)
fh.setFormatter(formatter)
# add the handlers to logger
logger.addHandler(ch)
logger.addHandler(fh)

# 'application' code
logger.debug('debug message')
logger.info('info message')
logger.warning('warn message')
logger.error('error message')
logger.critical('critical message')
```

Tenga en cuenta que el código de la “aplicación” no se preocupa por los gestores múltiples. Todo lo que cambió fue la adición y configuración de un nuevo gestor llamado *fh*.

La capacidad de crear nuevos gestores con filtros de mayor o menor prioridad puede ser muy útil al escribir y probar una aplicación. En lugar de usar muchas declaraciones `print` para la depuración, use `logger.debug`: a diferencia de las declaraciones de impresión, que tendrá que eliminar o comentar más tarde, las declaraciones de *logger.debug* pueden permanecer intactas en el código fuente y permanecen inactivas hasta que las necesite nuevamente. En ese momento, el único cambio que debe realizar es modificar el nivel de prioridad del *logger* y/o gestor para depurar.

Logging en múltiples destinos

Supongamos que desea que la consola y un archivo tengan diferentes formatos de mensaje y salida de log para diferentes situaciones. Por ejemplo, desea registrar mensajes con un nivel DEBUG y superiores en un archivo y enviar mensajes con nivel INFO y superior a la consola. Además, suponga que desea grabar una marca de tiempo en el archivo y no imprimirlo en la consola. Puede lograr este comportamiento haciendo lo siguiente:

```
import logging

# set up logging to file - see previous section for more
logging.basicConfig(level=logging.DEBUG,
                    format='%(asctime)s %(name)-12s %(le
                    datefmt='%m-%d %H:%M',
                    filename='/tmp/myapp.log',
                    filemode='w')
# define a Handler which writes INFO messages or higher
console = logging.StreamHandler()
console.setLevel(logging.INFO)
# set a format which is simpler for console use
formatter = logging.Formatter('%(name)-12s: %(levelname)
# tell the handler to use this format
console.setFormatter(formatter)
# add the handler to the root logger
```

```
logging.getLogger().addHandler(console)

# Now, we can log to the root logger, or any other logge
logging.info('Jackdaws love my big sphinx of quartz.')

# Now, define a couple of other loggers which might repr
# application:

logger1 = logging.getLogger('myapp.area1')
logger2 = logging.getLogger('myapp.area2')

logger1.debug('Quick zephyrs blow, vexing daft Jim.')
logger1.info('How quickly daft jumping zebras vex.')
logger2.warning('Jail zesty vixen who grabbed pay from q
logger2.error('The five boxing wizards jump quickly.')
```

Cuando ejecute esto, en la consola verá

```
root         : INFO     Jackdaws love my big sphinx of qu
myapp.area1 : INFO     How quickly daft jumping zebras v
myapp.area2 : WARNING  Jail zesty vixen who grabbed pay
myapp.area2 : ERROR    The five boxing wizards jump quic
```

y en el archivo verá algo como

```
10-22 22:19 root          INFO     Jackdaws love my big s
10-22 22:19 myapp.area1  DEBUG    Quick zephyrs blow, ve
10-22 22:19 myapp.area1  INFO     How quickly daft jumpi
10-22 22:19 myapp.area2  WARNING  Jail zesty vixen who g
10-22 22:19 myapp.area2  ERROR    The five boxing wizarc
```

Como se puede ver, el mensaje DEBUG sólo se muestra en el archivo. Los otros mensajes se envían a los dos destinos.

Este ejemplo usa gestores de consola y archivos, pero puede usar cualquier número y combinación de los gestores que elija.

Tenga en cuenta que la elección anterior del nombre del archivo de registro `/tmp/myapp.log` implica el uso de una ubicación estándar para los archivos temporales en los sistemas POSIX. En

Windows, es posible que tenga que elegir un nombre de directorio diferente para el registro - sólo asegúrese de que el directorio existe y que tiene los permisos para crear y actualizar archivos en él.

Gestión personalizada de niveles

A veces, es posible que desee hacer algo ligeramente diferente del manejo estándar de niveles, donde todos los niveles por encima de un umbral son procesados por un gestor. Para ello, es necesario utilizar filtros. Veamos un escenario en el que se desea organizar las cosas de la siguiente manera:

- Enviar mensajes de gravedad `INFO` y `WARNING` a `sys.stdout`
- Enviar mensajes de gravedad `ERROR` y superiores a `sys.stderr`
- Envía los mensajes de gravedad `DEBUG` y superiores al archivo `app.log`

Supongamos que se configura el registro con el siguiente JSON:

```
{
    "version": 1,
    "disable_existing_loggers": false,
    "formatters": {
        "simple": {
            "format": "%(levelname)-8s - %(message)s"
        }
    },
    "handlers": {
        "stdout": {
            "class": "logging.StreamHandler",
            "level": "INFO",
            "formatter": "simple",
            "stream": "ext://sys.stdout"
        },
        "stderr": {
            "class": "logging.StreamHandler",
            "level": "ERROR",
            "formatter": "simple",
            "stream": "ext://sys.stderr"
        },
        "file": {
            "class": "logging.FileHandler",
            "formatter": "simple",
            "filename": "app.log",
            "mode": "w"
        }
    },
    "root": {
        "level": "DEBUG",
        "handlers": [
            "stderr",
            "stdout",
            "file"
        ]
    }
}
```

This configuration does *almost* what we want, except that `sys.stdout` would show messages of severity `ERROR` and only events of this severity and higher will be tracked as well as `INFO` and `WARNING` messages. To prevent this, we can set up a filter which excludes those messages and add it to the relevant handler. This can be configured by adding a `filters` section parallel to `formatters` and `handlers`:

```
{
    "filters": {
        "warnings_and_below": {
            "()" : "__main__.filter_maker",
            "level": "WARNING"
        }
    }
}
```

y cambiando la sección del gestor `stdout` para añadirlo:

```
{
    "stdout": {
        "class": "logging.StreamHandler",
        "level": "INFO",
        "formatter": "simple",
        "stream": "ext://sys.stdout",
        "filters": ["warnings_and_below"]
    }
}
```

Un filtro no es más que una función, por lo que podemos definir el `filter_maker` (una función de fábrica) como sigue:

```
def filter_maker(level):
    level = getattr(logging, level)

    def filter(record):
        return record.levelno <= level

    return filter
```

Esto convierte el argumento de la cadena de caracteres pasada en un nivel numérico, y retorna una función que sólo retorna `True` si el nivel del registro pasado está en o por debajo del nivel especificado. Ten en cuenta que en este ejemplo se ha definido el `filter_maker` en un script de prueba `main.py` que se ejecuta desde la línea de comandos, por lo que su módulo será `__main__` - de ahí el `__main__.filter_maker` en la configuración del filtro. Tendrás que cambiar eso si lo defines en un módulo diferente.

Con el filtro añadido, podemos ejecutar `main.py`, que en su totalidad es:

```
import json
import logging
import logging.config

CONFIG = '''
{
    "version": 1,
    "disable_existing_loggers": false,
    "formatters": {
        "simple": {
            "format": "%(levelname)-8s - %(message)s"
        }
    },
    "filters": {
        "warnings_and_below": {
            "()" : "__main__.filter_maker",
            "level": "WARNING"
        }
    },
    "handlers": {
        "stdout": {
            "class": "logging.StreamHandler",
            "level": "INFO",
            "formatter": "simple",
            "stream": "ext://sys.stdout",
            "filters": ["warnings_and_below"]
        },
        "stderr": {
            "class": "logging.StreamHandler",
            "level": "ERROR",
            "formatter": "simple",
            "stream": "ext://sys.stderr"
        },
        "file": {
            "class": "logging.FileHandler",
            "formatter": "simple",
            "filename": "app.log",
            "mode": "w"
        }
    },
    "root": {
        "level": "DEBUG",
        "handlers": [
            "stderr",
            "stdout",
                "file"
            ]
        }
}
'''

def filter_maker(level):
    level = getattr(logging, level)

    def filter(record):
        return record.levelno <= level

    return filter

logging.config.dictConfig(json.loads(CONFIG))
logging.debug('A DEBUG message')
logging.info('An INFO message')
logging.warning('A WARNING message')
logging.error('An ERROR message')
logging.critical('A CRITICAL message')
```

Y después de ejecutarlo de esta manera:

```
python main.py 2>stderr.log >stdout.log
```

Podemos ver que los resultados son los esperados:

```
$ more *.log
::::::::::::::
app.log
::::::::::::::
DEBUG    - A DEBUG message
INFO     - An INFO message
WARNING  - A WARNING message
ERROR    - An ERROR message
CRITICAL - A CRITICAL message
::::::::::::::
stderr.log
::::::::::::::
```

```
ERROR    - An ERROR message
CRITICAL - A CRITICAL message
::::::::::::::
stdout.log
::::::::::::::
INFO     - An INFO message
WARNING  - A WARNING message
```

Ejemplo de servidor de configuración

Aquí hay un ejemplo de un módulo que usa el servidor de configuración logging:

```
import logging
import logging.config
import time
import os

# read initial config file
logging.config.fileConfig('logging.conf')

# create and start listener on port 9999
t = logging.config.listen(9999)
t.start()

logger = logging.getLogger('simpleExample')

try:
    # loop through logging calls to see the difference
    # new configurations make, until Ctrl+C is pressed
    while True:
        logger.debug('debug message')
        logger.info('info message')
        logger.warning('warn message')
        logger.error('error message')
        logger.critical('critical message')
        time.sleep(5)
except KeyboardInterrupt:
    # cleanup
    logging.config.stopListening()
    t.join()
```

Y aquí hay un script que toma un nombre de archivo y envía ese archivo al servidor, precedido adecuadamente con la longitud codificada en binario, como la nueva configuración de logging:

```
#!/usr/bin/env python
import socket, sys, struct

with open(sys.argv[1], 'rb') as f:
    data_to_send = f.read()

HOST = 'localhost'
PORT = 9999
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print('connecting...')
s.connect((HOST, PORT))
print('sending config...')
s.send(struct.pack('>L', len(data_to_send)))
s.send(data_to_send)
s.close()
print('complete')
```

Tratar con gestores que bloquean

A veces tiene que hacer que sus gestores de registro hagan su trabajo sin bloquear el hilo desde el que está iniciando sesión. Esto es común en las aplicaciones web, aunque, por supuesto, también ocurre en otros escenarios.

Un responsable habitual que ejemplifica un comportamiento lento es la **`SMTPHandler`**: el envío de correos electrónicos puede llevar mucho tiempo, por varias razones fuera del control del desarrollador (por ejemplo, una infraestructura de red o correo de bajo rendimiento). Pero casi cualquier controlador basado en red puede bloquear: incluso una operación **`SocketHandler`** puede estar haciendo a bajo nivel una consulta DNS que es demasiado lenta (y esta consulta puede estar en el código de la biblioteca de socket, debajo de la capa de Python, y fuera de su control).

Una solución es utilizar un enfoque de dos partes. Para la primera parte, adjunte solo una **`QueueHandler`** a los loggers que se acceden desde subprocesos críticos de rendimiento. Simplemente escriben en su cola, que puede dimensionarse a una capacidad lo suficientemente grande o inicializarse sin límite superior a su tamaño. La escritura en la cola generalmente se aceptará rápidamente, aunque es probable que deba atrapar la excepción **`queue.Full`** como precaución en su código. Si usted es un desarrollador de bibliotecas que tiene subprocesos críticos de rendimiento en su código, asegúrese de documentar esto (junto con una sugerencia de adjuntar solo `QueueHandlers` a sus loggers) para el beneficio de otros desarrolladores que usarán su código.

La segunda parte de la solución es **`QueueListener`**, que fue designado como la contraparte de **`QueueHandler`**. Un **`QueueListener`** es muy simple: ha pasado una cola y algunos gestores, y activa un hilo interno que escucha su cola para *LogRecords* enviados desde `QueueHandlers` (o cualquier otra fuente de `LogRecords`, para el caso). Los `LogRecords` se eliminan de la cola y se pasan a los gestores para su procesamiento.

La ventaja de tener una clase separada **`QueueListener`** es que puede usar la misma instancia para dar servicio a múltiples `QueueHandlers`. Esto es más amigable con los recursos que, por ejemplo, tener versiones enhebradas de las clases de gestores existentes, que consumirían un hilo por gestor sin ningún beneficio particular.

Un ejemplo del uso de estas dos clases a continuación (se omiten *imports*):

```
que = queue.Queue(-1)  # no limit on size
queue_handler = QueueHandler(que)
handler = logging.StreamHandler()
listener = QueueListener(que, handler)
root = logging.getLogger()
root.addHandler(queue_handler)
```

```
formatter = logging.Formatter('%(threadName)s: %(message
handler.setFormatter(formatter)
listener.start()
# The log output will display the thread which generatec
# the event (the main thread) rather than the internal
# thread which monitors the internal queue. This is what
# you want to happen.
root.warning('Look out!')
listener.stop()
```

que, cuando se ejecuta, producirá:

```
MainThread: Look out!
```

Nota

Aunque la discusión anterior no se refería específicamente al código asíncrono, sino más bien a los gestores de logging lentos, hay que tener en cuenta que cuando se realiza logging desde código asíncrono, los gestores de red e incluso de archivos podrían dar problemas (bloqueo del bucle de eventos) porque parte del logging se realiza desde los internos de **`asyncio`**. Podría ser mejor, si se utiliza cualquier código asíncrono en una aplicación, utilizar el enfoque anterior para el logging, de modo que cualquier código de bloqueo se ejecute sólo en el hilo `QueueListener`.

Distinto en la versión 3.5: Antes de Python 3.5, **`QueueListener`** siempre pasaba cada mensaje recibido de la cola a cada controlador con el que se inicializaba. (Esto se debió a que se asumió que el filtrado de nivel se realizó en el otro lado, donde se llena la cola). A partir de 3.5, este comportamiento se puede cambiar pasando un argumento de palabra clave `respect_handler_level=True` al constructor del oyente . Cuando se hace esto, el oyente compara el nivel de cada mensaje con el nivel del controlador y solo pasa un mensaje a un controlador si es apropiado hacerlo.

Distinto en la versión 3.14: The **`QueueListener`** can be started (and stopped) via the **`with`** statement. For example:

```
with QueueListener(que, handler) as listener:
    # The queue listener automatically starts
    # when the 'with' block is entered.
    pass
# The queue listener automatically stops once
# the 'with' block is exited.
```

Enviar y recibir eventos logging a través de una red

Supongamos que desea enviar eventos de registro a través de una red y gestionarlos en el extremo receptor. Una forma sencilla de hacer esto es adjuntar una instancia de `SocketHandler` al registrador raíz en el extremo de envío:

```
import logging, logging.handlers

rootLogger = logging.getLogger()
rootLogger.setLevel(logging.DEBUG)
socketHandler = logging.handlers.SocketHandler('localhos
                    logging.handlers.DEFAULT_TCP_LOGGING
# don't bother with a formatter, since a socket handler
# an unformatted pickle
rootLogger.addHandler(socketHandler)

# Now, we can log to the root logger, or any other logge
logging.info('Jackdaws love my big sphinx of quartz.')

# Now, define a couple of other loggers which might repr
# application:

logger1 = logging.getLogger('myapp.area1')
logger2 = logging.getLogger('myapp.area2')

logger1.debug('Quick zephyrs blow, vexing daft Jim.')
logger1.info('How quickly daft jumping zebras vex.')
logger2.warning('Jail zesty vixen who grabbed pay from q
logger2.error('The five boxing wizards jump quickly.')
```

En el extremo receptor, puede configurar un receptor usando el módulo **`socketserver`**. Aquí hay un ejemplo básico de trabajo:

```
import pickle
import logging
import logging.handlers
import socketserver
import struct


class LogRecordStreamHandler(socketserver.StreamRequestH
    """Handler for a streaming logging request.

    This basically logs the record using whatever loggin
    configured locally.
    """

    def handle(self):
        """
        Handle multiple requests - each expected to be a
        followed by the LogRecord in pickle format. Logs
        according to whatever policy is configured local
        """
        while True:
            chunk = self.connection.recv(4)
            if len(chunk) < 4:
                break
            slen = struct.unpack('>L', chunk)[0]
            chunk = self.connection.recv(slen)
            while len(chunk) < slen:
                chunk = chunk + self.connection.recv(sle
            obj = self.unPickle(chunk)
            record = logging.makeLogRecord(obj)
            self.handleLogRecord(record)

    def unPickle(self, data):
        return pickle.loads(data)

    def handleLogRecord(self, record):
        # if a name is specified, we use the named logge
        # implied by the record.
        if self.server.logname is not None:
            name = self.server.logname
        else:
            name = record.name
        logger = logging.getLogger(name)
        # N.B. EVERY record gets logged. This is because
        # is normally called AFTER logger-level filterin
        # to do filtering, do it at the client end to sa
        # cycles and network bandwidth!
        logger.handle(record)

class LogRecordSocketReceiver(socketserver.ThreadingTCPS
    """
    Simple TCP socket-based logging receiver suitable fo
    """

    allow_reuse_address = True

    def __init__(self, host='localhost',
                 port=logging.handlers.DEFAULT_TCP_LOGGI
                 handler=LogRecordStreamHandler):
        socketserver.ThreadingTCPServer.__init__(self, (
        self.abort = 0
        self.timeout = 1
        self.logname = None

    def serve_until_stopped(self):
        import select
        abort = 0
        while not abort:
            rd, wr, ex = select.select([self.socket.file
                                       [], [],
                                       self.timeout)
            if rd:
                self.handle_request()
            abort = self.abort
```

```
def main():
    logging.basicConfig(
        format='%(relativeCreated)5d %(name)-15s %(level
    tcpserver = LogRecordSocketReceiver()
    print('About to start TCP server...')
    tcpserver.serve_until_stopped()

if __name__ == '__main__':
    main()
```

Primero ejecuta el servidor, y luego el cliente. Del lado del cliente, nada se imprime en la consola; del lado del servidor, se debería ver algo como esto:

```
About to start TCP server...
   59 root            INFO     Jackdaws love my big sphi
   59 myapp.area1     DEBUG    Quick zephyrs blow, vexir
   69 myapp.area1     INFO     How quickly daft jumping 
   69 myapp.area2     WARNING  Jail zesty vixen who grab
   69 myapp.area2     ERROR    The five boxing wizards j
```

Note that there are some security issues with pickle in some scenarios. If these affect you, you can use an alternative serialization scheme by overriding the **`makePickle()`** method and implementing your alternative there, as well as adapting the above script to use your alternative serialization.

Ejecutando un logging de socket oyente en producción

To run a logging listener in production, you may need to use a process-management tool such as Supervisor [http://supervisord.org/]. Here is a Gist [https://gist.github.com/vsajip/4b227eeec43817465ca835ca66f75e2b] which provides the bare-bones files to run the above functionality using Supervisor. It consists of the following files:

File	Purpose
prepare.sh	A Bash script to prepare the environment for testing

`supervisor.conf`	The Supervisor configuration file, which has entries for the listener and a multi-process web application
`ensure_app.sh`	A Bash script to ensure that Supervisor is running with the above configuration
`log_listener.py`	The socket listener program which receives log events and records them to a file
`main.py`	A simple web application which performs logging via a socket connected to the listener
`webapp.json`	A JSON configuration file for the web application
`client.py`	A Python script to exercise the web application

The web application uses Gunicorn [https://gunicorn.org/], which is a popular web application server that starts multiple worker processes to handle requests. This example setup shows how the workers can write to the same log file without conflicting with one another — they all go through the socket listener.

To test these files, do the following in a POSIX environment:

1. Download the Gist [https://gist.github.com/vsajip/4b227eeec43817465ca835ca66f75e2b] as a ZIP archive using the *Download ZIP* button.
2. Unzip the above files from the archive into a scratch directory.
3. In the scratch directory, run `bash prepare.sh` to get things ready. This creates a `run` subdirectory to contain Supervisor-related and log files, and a `venv` subdirectory to contain a virtual environment into which `bottle`, `gunicorn` and `supervisor` are installed.
4. Run `bash ensure_app.sh` to ensure that Supervisor is running with the above configuration.
5. Run `venv/bin/python client.py` to exercise the web application, which will lead to records being written to the log.
6. Inspect the log files in the `run` subdirectory. You should see the most recent log lines in files matching the pattern `app.log*`. They won't be in any particular order, since they have been handled concurrently by different worker processes in a non-deterministic way.
7. You can shut down the listener and the web application by

running `venv/bin/supervisorctl -c supervisor.conf shutdown`.

You may need to tweak the configuration files in the unlikely event that the configured ports clash with something else in your test environment.

The default configuration uses a TCP socket on port 9020. You can use a Unix Domain socket instead of a TCP socket by doing the following:

1. In `listener.json`, add a `socket` key with the path to the domain socket you want to use. If this key is present, the listener listens on the corresponding domain socket and not on a TCP socket (the `port` key is ignored).
2. In `webapp.json`, change the socket handler configuration dictionary so that the `host` value is the path to the domain socket, and set the `port` value to `null`.

Agregar información contextual a su salida de logging

A veces, desea que la salida de logging contenga información contextual además de los parámetros pasados a la llamada del logging. Por ejemplo, en una aplicación en red, puede ser conveniente registrar información específica del cliente en el logging (por ejemplo, el nombre de usuario del cliente remoto o la dirección IP). Aunque puede usar el parámetro *extra* para lograr esto, no siempre es conveniente pasar la información de esta manera. Si bien puede resultar tentador crear instancias **`Logger`** por conexión, esta no es una buena idea porque estas instancias no se liberan de memoria vía el recolector de basura (*garbage collector*). Si bien esto no es un problema en la práctica, cuando el número de instancias de **`Logger`** depende del nivel de granularidad que desea usar para hacer el logging de una aplicación, podría ser difícil de administrar si el número de instancias **`Logger`** se vuelven efectivamente ilimitadas.

Uso de LoggerAdapters para impartir información

contextual

Una manera fácil de pasar información contextual para que se genere junto con la información de eventos logging es usar la clase `LoggerAdapter`. Esta clase está diseñada para parecerse a `Logger`, de modo que pueda llamar `debug()`, `info()`, `warning()`, `error()`, `excepción()`, `critical()` y `log()`. Estos métodos tienen las mismas signaturas que sus contrapartes en `Logger`, por lo que puede usar los dos tipos de instancias indistintamente.

Cuando creas una instancia de `LoggerAdapter`, le pasas una instancia de `Logger` y un objeto similar a un dict que contiene tu información contextual. Cuando llamas a uno de los métodos de registro en una instancia de `LoggerAdapter`, delega la llamada a la instancia subyacente de `Logger` pasada a su constructor, y se arregla para pasar la información contextual en la llamada delegada . Aquí hay un fragmento del código de `LoggerAdapter`:

```
def debug(self, msg, /, *args, **kwargs):
    """
    Delegate a debug call to the underlying logger, afte
    contextual information from this adapter instance.
    """
    msg, kwargs = self.process(msg, kwargs)
    self.logger.debug(msg, *args, **kwargs)
```

El método `process()` de `LoggerAdapter` es donde la información contextual se agrega a la salida del logging. Se pasa el mensaje y los argumentos de palabra clave de la llamada logging, y retorna versiones (potencialmente) modificadas de estos para usar en la llamada al logging subyacente. La implementación predeterminada de este método deja el mensaje solo, pero inserta una clave “extra” en el argumento de palabra clave cuyo valor es el objeto tipo dict pasado al constructor. Por supuesto, si ha pasado un argumento de palabra clave “extra” en la llamada al adaptador, se sobrescribirá silenciosamente.

La ventaja de usar “extra” es que los valores en el objeto dict se combinan en la instancia `LogRecord` _dict_, lo que le permite usar cadenas personalizadas con sus instancias **`Formatter`** que conocen las claves del objeto dict. Si necesita un método diferente, por ejemplo, si desea anteponer o agregar la información contextual a la cadena del mensaje, solo necesita la subclase **`LoggerAdapter`** y anular **`process()`** para hacer lo que necesita. Aquí hay un ejemplo simple:

```
class CustomAdapter(logging.LoggerAdapter):
    """
    This example adapter expects the passed in dict-like
    'connid' key, whose value in brackets is prepended t
    """
    def process(self, msg, kwargs):
        return '[%s] %s' % (self.extra['connid'], msg),
```

que puede usar así:

```
logger = logging.getLogger(__name__)
adapter = CustomAdapter(logger, {'connid': some_conn_id}
```

Luego, cualquier evento que registre en el adaptador tendrá el valor de `some_conn_id` antepuesto a los mensajes de logging.

Usar objetos distintos a los diccionarios para transmitir información contextual

No es necesario pasar un diccionario real a la **`LoggerAdapter`** - puedes pasar una instancia de una clase que implemente `__getitem__` y `__iter__` de modo que parezca un diccionario para el logging. Esto es útil si quieres generar valores dinámicamente (mientras que los valores en un diccionario son constantes).

Usar filtros para impartir información contextual

También puedes agregar información contextual a la salida del log utilizando un **`Filter`** definido por el usuario. Las instancias de `Filter` pueden modificar los `LogRecords` que se les pasan, incluido el agregado de atributos adicionales que luego se pueden generar utilizando cadena de caracteres con el formato adecuado, o si es necesario, un `Formatter` personalizado.

Por ejemplo, en una aplicación web, la solicitud que se está procesando (o al menos, las partes interesantes de la misma) se pueden almacenar en una variable *threadlocal* (`threading.local`) y luego se puede acceder a ella desde `Filter` para agregar información de la solicitud, - digamos, la dirección IP remota y el nombre de usuario-, al `LogRecord`, usando los nombres de atributo “ip” y “user” como en el ejemplo anterior de `LoggerAdapter`. En ese caso, se puede usar el mismo formato de cadena de caracteres para obtener un resultado similar al que se muestra arriba. Aquí hay un script de ejemplo:

```
import logging
from random import choice

class ContextFilter(logging.Filter):
    """
    This is a filter which injects contextual informatio

    Rather than use actual contextual information, we ju
    data in this demo.
    """

    USERS = ['jim', 'fred', 'sheila']
    IPS = ['123.231.231.123', '127.0.0.1', '192.168.0.1'

    def filter(self, record):

        record.ip = choice(ContextFilter.IPS)
        record.user = choice(ContextFilter.USERS)
        return True

if __name__ == '__main__':
    levels = (logging.DEBUG, logging.INFO, logging.WARNI
    logging.basicConfig(level=logging.DEBUG,
                        format='%(asctime)-15s %(name)-5
    a1 = logging.getLogger('a.b.c')
    a2 = logging.getLogger('d.e.f')
    f = ContextFilter()
    a1.addFilter(f)
    a2.addFilter(f)
    a1.debug('A debug message')
    a1.info('An info message with %s', 'some parameters'
    for x in range(10):
        lvl = choice(levels)
        lvlname = logging.getLevelName(lvl)
        a2.log(lvl, 'A message at %s level with %d %s',
```

que cuando se ejecuta, produce algo como:

```
2010-09-06 22:38:15,292 a.b.c DEBUG    IP: 123.231.231.1
2010-09-06 22:38:15,300 a.b.c INFO     IP: 192.168.0.1
2010-09-06 22:38:15,300 d.e.f CRITICAL IP: 127.0.0.1
2010-09-06 22:38:15,300 d.e.f ERROR    IP: 127.0.0.1
2010-09-06 22:38:15,300 d.e.f DEBUG    IP: 127.0.0.1
2010-09-06 22:38:15,300 d.e.f ERROR    IP: 123.231.231.1
2010-09-06 22:38:15,300 d.e.f CRITICAL IP: 192.168.0.1
2010-09-06 22:38:15,300 d.e.f CRITICAL IP: 127.0.0.1
2010-09-06 22:38:15,300 d.e.f DEBUG    IP: 192.168.0.1
2010-09-06 22:38:15,301 d.e.f ERROR    IP: 127.0.0.1
2010-09-06 22:38:15,301 d.e.f DEBUG    IP: 123.231.231.1
2010-09-06 22:38:15,301 d.e.f INFO     IP: 123.231.231.1
```

Uso de `contextvars`

Desde la versión 3.7 de Python, el módulo `contextvars` ha proporcionado almacenamiento local de contexto que funciona tanto para las necesidades de procesamiento de `threading` como de `asyncio`. Este tipo de almacenamiento puede ser, por tanto, preferible a los hilos locales. El siguiente ejemplo muestra cómo, en un entorno multihilo, los logs pueden rellenarse con información contextual como, por ejemplo, los atributos de las peticiones gestionadas por las aplicaciones web.

A modo de ilustración, digamos que tienes diferentes aplicaciones web, cada una de ellas independiente de la otra, pero que se ejecutan en el mismo proceso de Python y utilizan una biblioteca común a todas ellas. ¿Cómo puede cada una de estas aplicaciones tener su propio registro, donde todos los mensajes de logging de la biblioteca (y otro código de procesamiento de solicitudes) se dirigen al archivo de registro de la aplicación apropiada, mientras se incluye en el registro información contextual adicional como la IP del cliente, el método de solicitud HTTP y el nombre de usuario del cliente?

Supongamos que la biblioteca se puede simular con el siguiente código:

```
# webapplib.py
import logging
import time

logger = logging.getLogger(__name__)

def useful():
    # Just a representative event logged from the librar
    logger.debug('Hello from webapplib!')
    # Just sleep for a bit so other threads get to run
    time.sleep(0.01)
```

Podemos simular las aplicaciones web múltiples mediante dos clases simples, `Request` y `WebApp`. Éstas simulan cómo funcionan las aplicaciones web reales con hilos, cada petición es manejada por un hilo:

```
# main.py
import argparse
from contextvars import ContextVar
import logging
import os
from random import choice
import threading
import webapplib

logger = logging.getLogger(__name__)
root = logging.getLogger()
```

```
root.setLevel(logging.DEBUG)

class Request:
    """
    A simple dummy request class which just holds dummy
    client IP address and client username
    """
    def __init__(self, method, ip, user):
        self.method = method
        self.ip = ip
        self.user = user

# A dummy set of requests which will be used in the simu
# from this list randomly. Note that all GET requests ar
# addresses, whereas POST requests are from 192.16.3.XXX
# are represented in the sample requests.

REQUESTS = [
    Request('GET', '192.168.2.20', 'jim'),
    Request('POST', '192.168.3.20', 'fred'),
    Request('GET', '192.168.2.21', 'sheila'),
    Request('POST', '192.168.3.21', 'jim'),
    Request('GET', '192.168.2.22', 'fred'),
    Request('POST', '192.168.3.22', 'sheila'),
]

# Note that the format string includes references to req
# such as HTTP method, client IP and username

formatter = logging.Formatter('%(threadName)-11s %(appNa

# Create our context variables. These will be filled at
# processing, and used in the logging that happens durin

ctx_request = ContextVar('request')
ctx_appname = ContextVar('appname')

class InjectingFilter(logging.Filter):
    """
    A filter which injects context-specific information
    that only information for a specific webapp is inclu
    """
    def __init__(self, app):
        self.app = app

    def filter(self, record):
        request = ctx_request.get()
        record.method = request.method
        record.ip = request.ip
        record.user = request.user
        record.appName = appName = ctx_appname.get()
        return appName == self.app.name

class WebApp:
    """
    A dummy web application class which has its own hand
    webapp-specific log.
    """
    def __init__(self, name):
        self.name = name
        handler = logging.FileHandler(name + '.log', 'w'
        f = InjectingFilter(self)
        handler.setFormatter(formatter)
        handler.addFilter(f)
        root.addHandler(handler)
        self.num_requests = 0

    def process_request(self, request):
        """
        This is the dummy method for processing a reques
        different thread for every request. We store the
        the context vars before doing anything else.
        """
        ctx_request.set(request)
        ctx_appname.set(self.name)
        self.num_requests += 1
        logger.debug('Request processing started')
        webapplib.useful()
        logger.debug('Request processing finished')

def main():
    fn = os.path.splitext(os.path.basename(__file__))[0]
    adhf = argparse.ArgumentDefaultsHelpFormatter
    ap = argparse.ArgumentParser(formatter_class=adhf, p
                                 description='Simulate a
                                             'applicatio
                                             'requests, 
                                             'context ca
                                             'populate l
    aa = ap.add_argument
    aa('--count', '-c', type=int, default=100, help='How
    options = ap.parse_args()

    # Create the dummy webapps and put them in a list wh
    # from randomly
    app1 = WebApp('app1')
    app2 = WebApp('app2')
    apps = [app1, app2]
    threads = []
    # Add a common handler which will capture all events
    handler = logging.FileHandler('app.log', 'w')
    handler.setFormatter(formatter)
    root.addHandler(handler)

    # Generate calls to process requests
    for i in range(options.count):
        try:
            # Pick an app at random and a request for it
            app = choice(apps)
            request = choice(REQUESTS)
            # Process the request in its own thread
            t = threading.Thread(target=app.process_requ
            threads.append(t)
            t.start()
        except KeyboardInterrupt:
            break

    # Wait for the threads to terminate
    for t in threads:
        t.join()

    for app in apps:
        print('%s processed %s requests' % (app.name, ap

if __name__ == '__main__':
    main()
```

Si ejecuta lo anterior, debería encontrar que aproximadamente la mitad de las peticiones van a `app1.log` y el resto a `app2.log`, y las todas las peticiones se registran en `app.log`. Cada registro específico de la aplicación web contendrá únicamente entradas de log para esa aplicación web, y la información de las peticiones se mostrará de forma consistente en el registro (es decir, la información de cada petición ficticia aparecerá siempre junta en una línea de log). Esto se ilustra con la siguiente salida del shell:

```
~/logging-contextual-webapp$ python main.py
app1 processed 51 requests
app2 processed 49 requests
~/logging-contextual-webapp$ wc -l *.log
  153 app1.log
  147 app2.log
  300 app.log
  600 total
~/logging-contextual-webapp$ head -3 app1.log
Thread-3 (process_request) app1 __main__  jim    192.16
Thread-3 (process_request) app1 webapplib jim    192.16
Thread-5 (process_request) app1 __main__  jim    192.16
~/logging-contextual-webapp$ head -3 app2.log
Thread-1 (process_request) app2 __main__  sheila 192.16
Thread-1 (process_request) app2 webapplib sheila 192.16
Thread-2 (process_request) app2 __main__  jim    192.16
~/logging-contextual-webapp$ head app.log
```

```
Thread-1 (process_request) app2 __main__  sheila 192.168
Thread-1 (process_request) app2 webapplib sheila 192.168
Thread-2 (process_request) app2 __main__  jim    192.168
Thread-3 (process_request) app1 __main__  jim    192.168
Thread-2 (process_request) app2 webapplib jim    192.168
Thread-3 (process_request) app1 webapplib jim    192.168
Thread-4 (process_request) app2 __main__  fred   192.168
Thread-5 (process_request) app1 __main__  jim    192.168
Thread-4 (process_request) app2 webapplib fred   192.168
Thread-6 (process_request) app1 __main__  jim    192.168
~/logging-contextual-webapp$ grep app1 app1.log | wc -l
153
~/logging-contextual-webapp$ grep app2 app2.log | wc -l
147
~/logging-contextual-webapp$ grep app1 app.log | wc -l
153
~/logging-contextual-webapp$ grep app2 app.log | wc -l
147
```

Impartir información contextual en los gestores

Cada `Handler` tiene su propia cadena de filtros. Si quieres añadir información contextual a una `LogRecord` sin filtrarla a otros gestores, puedes utilizar un filtro que retorna una nueva **`LogRecord`** en lugar de modificarlo in situ, como se muestra en el siguiente script:

```
import copy
import logging

def filter(record: logging.LogRecord):
    record = copy.copy(record)
    record.user = 'jim'
    return record

if __name__ == '__main__':
    logger = logging.getLogger()
    logger.setLevel(logging.INFO)
    handler = logging.StreamHandler()
    formatter = logging.Formatter('%(message)s from %(us
    handler.setFormatter(formatter)
    handler.addFilter(filter)
    logger.addHandler(handler)

    logger.info('A log message')
```

Logging a un sólo archivo desde múltiples procesos

Aunque logging es seguro para hilos, y el logging a un solo archivo desde múltiples hilos en un solo proceso *es* compatible, el logging en un solo archivo desde *múltiples procesos no* es compatible, porque no existe una forma estándar de serializar el acceso a un solo archivo en múltiples procesos en Python. Si necesita hacer esto último, una forma de abordarlo es hacer que todos los procesos se registren en una **`SocketHandler`**, y tener un proceso separado que implemente un servidor de socket que lee del socket y los loggings para archivar. (Si lo prefiere, puede dedicar un hilo en uno de los procesos existentes para realizar esta función.) Esta sección documenta este enfoque con más detalle e incluye un receptor socket que funciona que se puede utilizar como punto de partida para que se adapte a sus propias aplicaciones.

You could also write your own handler which uses the **`Lock`** class from the **`multiprocessing`** module to serialize access to the file from your processes. The stdlib **`FileHandler`** and subclasses do not make use of **`multiprocessing`**.

Alternativamente, puede usar una `Queue` y **`QueueHandler`** para enviar todos los logging a uno de los procesos en su aplicación multi-proceso. El siguiente script de ejemplo demuestra cómo puede hacer esto; en el ejemplo, un proceso de escucha independiente escucha los eventos enviados por otros procesos y los registra de acuerdo con su propia configuración de logging. Aunque el ejemplo solo demuestra una forma de hacerlo (por ejemplo, es posible que desee utilizar un hilo de escucha en lugar de un proceso de escucha separado; la implementación sería análoga), permite configuraciones de logging completamente diferentes para el oyente y los otros procesos en su aplicación. Y se puede utilizar como base para el código que cumpla con sus propios requisitos específicos:

```
# You'll need these imports in your own code
import logging
import logging.handlers
import multiprocessing

# Next two import lines for this demo only
from random import choice, random
import time

#
# Because you'll want to define the logging configurati
# listener and worker process functions take a configure
# for configuring logging for that process. These functi
# which they use for communication.
#
# In practice, you can configure the listener however y
# simple example, the listener does not apply level or f
# In practice, you would probably want to do this logic
# sending events which would be filtered out between pr
#
# The size of the rotated files is made small so you ca
def listener_configurer():
    root = logging.getLogger()
    h = logging.handlers.RotatingFileHandler('mptest.lo
    f = logging.Formatter('%(asctime)s %(processName)-1
    h.setFormatter(f)
    root.addHandler(h)

# This is the listener process top-level loop: wait for
# (LogRecords)on the queue and handle them, quit when y
# LogRecord.
def listener_process(queue, configurer):
    configurer()
    while True:
        try:
            record = queue.get()
            if record is None:  # We send this as a sent
                break
            logger = logging.getLogger(record.name)
            logger.handle(record)  # No level or filter
        except Exception:
            import sys, traceback
            print('Whoops! Problem:', file=sys.stderr)
            traceback.print_exc(file=sys.stderr)

# Arrays used for random selections in this demo

LEVELS = [logging.DEBUG, logging.INFO, logging.WARNING,
          logging.ERROR, logging.CRITICAL]

LOGGERS = ['a.b.c', 'd.e.f']

MESSAGES = [
    'Random message #1',
    'Random message #2',
    'Random message #3',
]

# The worker configuration is done at the start of the w
# Note that on Windows you can't rely on fork semantics,
# will run the logging configuration code when it starts
def worker_configurer(queue):
    h = logging.handlers.QueueHandler(queue)  # Just the
    root = logging.getLogger()
    root.addHandler(h)
    # send all messages, for demo; no other level or fil
    root.setLevel(logging.DEBUG)

# This is the worker process top-level loop, which just
# random intervening delays before terminating.
# The print messages are just so you know it's doing som
def worker_process(queue, configurer):
    configurer(queue)
    name = multiprocessing.current_process().name
    print('Worker started: %s' % name)
    for i in range(10):
        time.sleep(random())
        logger = logging.getLogger(choice(LOGGERS))
        level = choice(LEVELS)
        message = choice(MESSAGES)
        logger.log(level, message)
    print('Worker finished: %s' % name)

# Here's where the demo gets orchestrated. Create the qu
# the listener, create ten workers and start them, wait
# then send a None to the queue to tell the listener to
def main():
    queue = multiprocessing.Queue(-1)
    listener = multiprocessing.Process(target=listener_p
                                       args=(queue, list
    listener.start()
    workers = []
    for i in range(10):
        worker = multiprocessing.Process(target=worker_p
                                         args=(queue, wo
        workers.append(worker)
        worker.start()
    for w in workers:
        w.join()
    queue.put_nowait(None)
    listener.join()

if __name__ == '__main__':
    main()
```

Una variante del script anterior mantiene el logging en el proceso principal, en un hilo separado:

```
import logging
import logging.config
import logging.handlers
```

```
from multiprocessing import Process, Queue
import random
import threading
import time

def logger_thread(q):
    while True:
        record = q.get()
        if record is None:
            break
        logger = logging.getLogger(record.name)
        logger.handle(record)


def worker_process(q):
    qh = logging.handlers.QueueHandler(q)
    root = logging.getLogger()
    root.setLevel(logging.DEBUG)
    root.addHandler(qh)
    levels = [logging.DEBUG, logging.INFO, logging.WARNI
              logging.CRITICAL]
    loggers = ['foo', 'foo.bar', 'foo.bar.baz',
               'spam', 'spam.ham', 'spam.ham.eggs']
    for i in range(100):
        lvl = random.choice(levels)
        logger = logging.getLogger(random.choice(loggers
        logger.log(lvl, 'Message no. %d', i)

if __name__ == '__main__':
    q = Queue()
    d = {
        'version': 1,
        'formatters': {
            'detailed': {
                'class': 'logging.Formatter',
                'format': '%(asctime)s %(name)-15s %(lev
            }
        },
        'handlers': {
            'console': {
                'class': 'logging.StreamHandler',
                'level': 'INFO',
            },
            'file': {
                'class': 'logging.FileHandler',
                'filename': 'mplog.log',
                'mode': 'w',
                'formatter': 'detailed',
            },
            'foofile': {
                'class': 'logging.FileHandler',
                'filename': 'mplog-foo.log',
                'mode': 'w',
                'formatter': 'detailed',
            },
            'errors': {
                'class': 'logging.FileHandler',
                'filename': 'mplog-errors.log',
                'mode': 'w',
                'level': 'ERROR',
                'formatter': 'detailed',
            },
        },
        'loggers': {
            'foo': {
                'handlers': ['foofile']
            }
        },
        'root': {
            'level': 'DEBUG',
            'handlers': ['console', 'file', 'errors']
        },
    }
    workers = []
    for i in range(5):
        wp = Process(target=worker_process, name='worker
        workers.append(wp)
        wp.start()
    logging.config.dictConfig(d)
    lp = threading.Thread(target=logger_thread, args=(q,
    lp.start()
    # At this point, the main process could do some usef
    # Once it's done that, it can wait for the workers t
    for wp in workers:
        wp.join()
    # And now tell the logging thread to finish up, too
    q.put(None)
    lp.join()
```

Esta variante muestra cómo puede, por ejemplo, aplicar la configuración para logging particulares: el registrador `foo` tiene un gestor especial que almacena todos los eventos en el subsistema `foo` en un archivo `mplog-foo.log`. Esto será utilizado por la maquinaria de logging en el proceso principal (aunque los eventos logging se generen en los procesos de trabajo) para dirigir los mensajes a los destinos apropiados.

Usando concurrent.futures.ProcessPoolExecutor

Si desea utilizar **`concurrent.futures.ProcessPoolExecutor`** para iniciar sus procesos de trabajo, debe crear la cola de manera ligeramente diferente. En vez de

```
queue = multiprocessing.Queue(-1)
```

debería usar

```
queue = multiprocessing.Manager().Queue(-1)  # also work
```

y luego puede reemplazar la creación del trabajador de esto:

```
workers = []
for i in range(10):
    worker = multiprocessing.Process(target=worker_proce
                                     args=(queue, worker
    workers.append(worker)
    worker.start()
for w in workers:
    w.join()
```

a esto (recuerda el primer *import* `concurrent.futures`):

```
with concurrent.futures.ProcessPoolExecutor(max_workers=
    for i in range(10):
        executor.submit(worker_process, queue, worker_co
```

Deploying web applications using Gunicorn and uWSGI

When deploying web applications using Gunicorn [https://gunicorn.org/] or uWSGI [https://uwsgi-docs.readthedocs.io/en/latest/] (or similar), multiple worker processes are created to handle client requests. In such environments, avoid creating file-based handlers directly in your web application. Instead, use a `SocketHandler` to log from the web application to a listener in a separate process. This can be set up using a process management tool such as Supervisor - see Running a logging socket listener in production for more details.

Usando rotación de archivos

Sometimes you want to let a log file grow to a certain size, then open a new file and log to that. You may want to keep a certain number of these files, and when that many files have been created, rotate the files so that the number of files and the size of the files both remain bounded. For this usage pattern, the logging package provides a `RotatingFileHandler`:

```
import glob
import logging
import logging.handlers

LOG_FILENAME = 'logging_rotatingfile_example.out'
```

```
# Set up a specific logger with our desired output level
my_logger = logging.getLogger('MyLogger')
my_logger.setLevel(logging.DEBUG)

# Add the log message handler to the logger
handler = logging.handlers.RotatingFileHandler(
              LOG_FILENAME, maxBytes=20, backupCount=5)

my_logger.addHandler(handler)

# Log some messages
for i in range(20):
    my_logger.debug('i = %d' % i)

# See what files are created
logfiles = glob.glob('%s*' % LOG_FILENAME)

for filename in logfiles:
    print(filename)
```

El resultado debe ser 6 archivos separados, cada uno con parte del historial de log de la aplicación:

```
logging_rotatingfile_example.out
logging_rotatingfile_example.out.1
logging_rotatingfile_example.out.2
logging_rotatingfile_example.out.3
logging_rotatingfile_example.out.4
logging_rotatingfile_example.out.5
```

El archivo más actual siempre es `logging_rotatingfile_example.out`, y cada vez que alcanza el límite de tamaño, se le cambia el nombre con el sufijo `.1`. Se cambia el nombre de cada uno de los archivos de respaldo existentes para incrementar el sufijo (`.1` se convierte en `.2`, etc.) y se borra el archivo `.6`.

Obviamente, este ejemplo establece la longitud del log demasiado pequeña como un ejemplo extremo. Se querrá establecer *maxBytes* en un valor apropiado.

Uso de estilos de formato alternativos

Cuando se agregó logging a la biblioteca estándar de Python, la única forma de formatear mensajes con contenido variable era usar el método de formateo %. Desde entonces, Python ha ganado dos nuevos enfoques de formato: **`string.Template`** (agregado en Python 2.4) y **`str.format()`** (agregado en Python 2.6).

Logging (a partir de la versión 3.2) proporciona un soporte mejorado para estos dos estilos de formato adicionales. La clase **`Formatter`** ha sido mejorada para tomar un parámetro de palabra clave adicional llamado `style`. El valor predeterminado es `'%'`, pero otros valores posibles son `'{'` y `'$'`, que corresponden a los otros dos estilos de formato. La compatibilidad con versiones anteriores se mantiene de forma predeterminada (como era de esperar), pero al especificar explícitamente un parámetro de estilo, tiene la capacidad de especificar cadenas de formato que funcionan con **`str.format()`** o **`string.Template`**. Aquí hay una sesión de consola de ejemplo para mostrar las posibilidades:

```
>>> import logging
>>> root = logging.getLogger()
>>> root.setLevel(logging.DEBUG)
>>> handler = logging.StreamHandler()
>>> bf = logging.Formatter('{asctime} {name} {levelname:
...                        style='{')
>>> handler.setFormatter(bf)
>>> root.addHandler(handler)
>>> logger = logging.getLogger('foo.bar')
>>> logger.debug('This is a DEBUG message')
2010-10-28 15:11:55,341 foo.bar DEBUG    This is a DEBUG
>>> logger.critical('This is a CRITICAL message')
2010-10-28 15:12:11,526 foo.bar CRITICAL This is a CRITI
>>> df = logging.Formatter('$asctime $name ${levelname}
...                        style='$')
>>> handler.setFormatter(df)
>>> logger.debug('This is a DEBUG message')
2010-10-28 15:13:06,924 foo.bar DEBUG This is a DEBUG me
>>> logger.critical('This is a CRITICAL message')
```

```
2010-10-28 15:13:11,494 foo.bar CRITICAL This is a CRITI
>>>
```

Tenga en cuenta que el formato de logging para la salida final a los logs es completamente independiente de cómo se construye un mensaje de logging individual. Para eso todavía puede usar el formateo %, como se muestra aquí:

```
>>> logger.error('This is an%s %s %s', 'other,', 'ERROR,
2010-10-28 15:19:29,833 foo.bar ERROR This is another, E
>>>
```

Las llamadas de Logging (`logger.debug()`, `logger.info()`, etc.) solo toman parámetros posicionales para el mensaje de logging real en sí, los parámetros de palabras clave se usan solo para determinar opciones sobre cómo gestionar la llamada propiamente a Logging (por ejemplo, el parámetro de palabra clave `exc_info` para indicar que la información de rastreo debe registrarse, o el parámetro de palabra clave `extra` para indicar información contextual adicional que se agregará al log). Por lo tanto, no puede realizar llamadas de logging directamente usando la sintaxis **`str.format()`** o **`string.Template`**, porque internamente el paquete de logging usa formato % para fusionar la cadena de formato y los argumentos de las variables. No habría ningún cambio en esto mientras se conserva la compatibilidad con versiones anteriores, ya que todas las llamadas de logging que están en el código existente usarán cadenas de formato %.

Sin embargo, existe una forma en la que puede usar el formato {} - y $ - para construir sus mensajes de log individuales. Recuerde que para un mensaje puede usar un objeto arbitrario como una cadena de caracteres de formato de mensaje, y que el paquete logging llamará a `str()` en ese objeto para obtener la cadena de caracteres de formato real. Considere las siguientes dos clases:

```
class BraceMessage:
    def __init__(self, fmt, /, *args, **kwargs):
        self.fmt = fmt
        self.args = args
        self.kwargs = kwargs
    def __str__(self):
        return self.fmt.format(*self.args, **self.kwargs

class DollarMessage:
    def __init__(self, fmt, /, **kwargs):
        self.fmt = fmt
        self.kwargs = kwargs

    def __str__(self):
        from string import Template
        return Template(self.fmt).substitute(**self.kwar
```

Cualquiera de estos puede usarse en lugar de una cadena de formato, para permitir que se use el formato {} - o $ - para construir la parte del «mensaje» real que aparece en la salida del log en lugar de «%(message)s» o «{message}» o «$message». Es un poco difícil de manejar usar los nombres de las clases siempre que quieras registrar algo, pero es bastante aceptable si usas un alias como __ (doble subrayado — no confundir con _, el subrayado simple usado como sinónimo/alias para **`gettext.gettext()`** o sus hermanos).

Las clases anteriores no están incluidas en Python, aunque son bastante fáciles de copiar y pegar en su propio código. Se pueden usar de la siguiente manera (asumiendo que están declaradas en un módulo llamado `wherever`):

```
>>> from wherever import BraceMessage as __
>>> print(__('Message with {0} {name}', 2, name='placeho
Message with 2 placeholders
>>> class Point: pass
...
>>> p = Point()
>>> p.x = 0.5
>>> p.y = 0.5
>>> print(__('Message with coordinates: ({point.x:.2f},
...          point=p))
Message with coordinates: (0.50, 0.50)
```

```
>>> from wherever import DollarMessage as __
>>> print(__('Message with $num $what', num=2, what='pla
Message with 2 placeholders
>>>
```

Si bien los ejemplos anteriores usan `print()` para mostrar cómo funciona el formateo, por supuesto usaría `logger.debug()` o similar para realmente registrar usando este enfoque.

One thing to note is that you pay no significant performance penalty with this approach: the actual formatting happens not when you make the logging call, but when (and if) the logged message is actually about to be output to a log by a handler. So the only slightly unusual thing which might trip you up is that the parentheses go around the format string and the arguments, not just the format string. That’s because the _ notation is just syntax sugar for a constructor call to one of the `XXXMessage` classes.

Si lo prefiere, puede usar **`LoggerAdapter`** para lograr un efecto similar al anterior, como en el siguiente ejemplo:

```
import logging

class Message:
    def __init__(self, fmt, args):
        self.fmt = fmt
        self.args = args

    def __str__(self):
        return self.fmt.format(*self.args)

class StyleAdapter(logging.LoggerAdapter):
    def log(self, level, msg, /, *args, stacklevel=1, **
        if self.isEnabledFor(level):
            msg, kwargs = self.process(msg, kwargs)
            self.logger.log(level, Message(msg, args), *
                            stacklevel=stacklevel+1)

logger = StyleAdapter(logging.getLogger(__name__))
```

```
def main():
    logger.debug('Hello, {}', 'world!')

if __name__ == '__main__':
    logging.basicConfig(level=logging.DEBUG)
    main()
```

The above script should log the message `Hello, world!` when run with Python 3.8 or later.

Personalización de `LogRecord`

Cada evento logging está representado por una instancia **`LogRecord`**. Cuando se registra un evento y no se filtra por el nivel de un registrador, se crea **`LogRecord`**, se llena con información sobre el evento y luego se pasa a los gestores de ese registrador (y sus antepasados, hasta (e incluyéndolo) el registrador donde se deshabilita una mayor propagación en la jerarquía). Antes de Python 3.2, solo había dos lugares donde se realizaba esta creación:

- **`Logger.makeRecord()`**, que se llama en el proceso normal de logging de un evento. Esto invoca **`LogRecord`** directamente para crear una instancia.
- **`makeLogRecord()`**, que se llama con un diccionario que contiene atributos que se agregarán al *LogRecord*. Esto se suele invocar cuando se ha recibido un diccionario adecuado a través de la red (por ejemplo, en forma de *pickle* a través de **`SocketHandler`**, o en formato JSON a través de **`HTTPHandler`**).

Por lo general, esto significa que si necesita hacer algo especial con **`LogRecord`**, debe hacer una de las siguientes cosas.

- Cree su propia subclase **`Logger`**, que anula **`Logger.makeRecord()`**, y configúrelo usando **`setLoggerClass()`** antes de que se creen instancias de los registradores que le interesan.
- Agrega un **`Filter`** a un registrador o gestor, que realiza la manipulación especial necesaria que necesita cuando se llama a su método **`filter()`**.

El primer enfoque sería un poco difícil de manejar en el escenario en el que (digamos) varias bibliotecas diferentes quisieran hacer cosas diferentes. Cada uno intentaría establecer su propia subclase `Logger`, y el que hiciera esto último ganaría.

El segundo enfoque funciona razonablemente bien en muchos casos, pero no le permite, por ejemplo, usar una subclase especializada de `LogRecord`. Los desarrolladores de bibliotecas pueden establecer un filtro adecuado en sus registradores, pero tendrían que recordar hacerlo cada vez que introduzcan un nuevo registrador (lo que harían simplemente agregando nuevos paquetes o módulos y haciendo

```
logger = logging.getLogger(__name__)
```

a nivel de módulo). Probablemente sean demasiadas cosas en las que pensar. Los desarrolladores también podrían agregar el filtro a `NullHandler` adjunto a su registrador de nivel superior, pero esto no se invocaría si un desarrollador de aplicaciones adjuntara un controlador a un registrador de biblioteca de nivel inferior — así que la salida de ese gestor no reflejaría las intenciones del desarrollador de la biblioteca.

En Python 3.2 y posteriores, la creación de `LogRecord` se realiza a través de una fábrica, que puede especificar. La fábrica es simplemente un invocable que puede configurar con `setLogRecordFactory()`, e interrogar con `getLogRecordFactory()`. La fábrica se invoca con la misma firma que el constructor **`LogRecord`**, ya que `LogRecord` es la configuración predeterminada de la fábrica.

Este enfoque permite que una fábrica personalizada controle todos los aspectos de la creación de *LogRecord*. Por ejemplo, podría devolver una subclase, o simplemente agregar algunos atributos adicionales al registro una vez creado, usando un patrón similar a este:

```
old_factory = logging.getLogRecordFactory()

def record_factory(*args, **kwargs):
    record = old_factory(*args, **kwargs)
    record.custom_attribute = 0xdecafbad
    return record

logging.setLogRecordFactory(record_factory)
```

Este patrón permite que diferentes bibliotecas encadenen fábricas juntas, y siempre que no sobrescriban los atributos de las demás o sobrescriban involuntariamente los atributos proporcionados como estándar, no debería haber sorpresas. Sin embargo, debe tenerse en cuenta que cada eslabón de la cadena agrega una sobrecarga de tiempo de ejecución a todas las operaciones de logging, y la técnica solo debe usarse cuando el uso de `Filter` no proporciona el resultado deseado.

Subclassing QueueHandler and QueueListener- a ZeroMQ example

Subclass `QueueHandler`

Puede usar una subclase `QueueHandler` para enviar mensajes a otros tipos de colas, por ejemplo, un socket de "publicación" ZeroMQ. En el siguiente ejemplo, el socket se crea por separado y se pasa al gestor (como su "cola"):

```
import zmq   # using pyzmq, the Python binding for ZeroM
import json  # for serializing records portably

ctx = zmq.Context()
sock = zmq.Socket(ctx, zmq.PUB)  # or zmq.PUSH, or other
sock.bind('tcp://*:5556')        # or wherever

class ZeroMQSocketHandler(QueueHandler):
    def enqueue(self, record):
        self.queue.send_json(record.__dict__)


handler = ZeroMQSocketHandler(sock)
```

Por supuesto, hay otras formas de organizar esto, por ejemplo, pasando los datos que necesita el gestor para crear el socket:

```
class ZeroMQSocketHandler(QueueHandler):
    def __init__(self, uri, socktype=zmq.PUB, ctx=None):
        self.ctx = ctx or zmq.Context()
        socket = zmq.Socket(self.ctx, socktype)
        socket.bind(uri)
        super().__init__(socket)

    def enqueue(self, record):
        self.queue.send_json(record.__dict__)

    def close(self):
        self.queue.close()
```

Subclass `QueueListener`

También puede subclasificar `QueueListener` para obtener mensajes de otros tipos de colas, por ejemplo, un socket de “suscripción” de ZeroMQ. Aquí tienes un ejemplo:

```
class ZeroMQSocketListener(QueueListener):
    def __init__(self, uri, /, *handlers, **kwargs):
        self.ctx = kwargs.get('ctx') or zmq.Context()
        socket = zmq.Socket(self.ctx, zmq.SUB)
        socket.setsockopt_string(zmq.SUBSCRIBE, '')  # s
        socket.connect(uri)
        super().__init__(socket, *handlers, **kwargs)

    def dequeue(self):
        msg = self.queue.recv_json()
        return logging.makeLogRecord(msg)
```

Subclassing QueueHandler and QueueListener- a `pynng` example

In a similar way to the above section, we can implement a listener and handler using pynng [https://pypi.org/project/pynng/], which is a

Python binding to NNG [https://nng.nanomsg.org/], billed as a spiritual successor to ZeroMQ. The following snippets illustrate – you can test them in an environment which has `pynng` installed. Just for variety, we present the listener first.

Subclass `QueueListener`

```
# listener.py
import json
import logging
import logging.handlers

import pynng

DEFAULT_ADDR = "tcp://localhost:13232"

interrupted = False

class NNGSocketListener(logging.handlers.QueueListener):

    def __init__(self, uri, /, *handlers, **kwargs):
        # Have a timeout for interruptability, and open
        # subscriber socket
        socket = pynng.Sub0(listen=uri, recv_timeout=500
        # The b'' subscription matches all topics
        topics = kwargs.pop('topics', None) or b''
        socket.subscribe(topics)
        # We treat the socket as a queue
        super().__init__(socket, *handlers, **kwargs)

    def dequeue(self, block):
        data = None
        # Keep looping while not interrupted and no data
        # socket
        while not interrupted:
            try:
                data = self.queue.recv(block=block)
                break
            except pynng.Timeout:
                pass
            except pynng.Closed:  # sometimes happens wh
                break
        if data is None:
            return None
        # Get the logging event sent from a publisher
        event = json.loads(data.decode('utf-8'))
        return logging.makeLogRecord(event)

    def enqueue_sentinel(self):
        # Not used in this implementation, as the socket
        # queue
        pass

logging.getLogger('pynng').propagate = False
listener = NNGSocketListener(DEFAULT_ADDR, logging.Strea
listener.start()
print('Press Ctrl-C to stop.')
try:
    while True:
        pass
except KeyboardInterrupt:
    interrupted = True
finally:
    listener.stop()
```

Subclass `QueueHandler`

```
# sender.py
import json
import logging
import logging.handlers
import time
import random

import pynng

DEFAULT_ADDR = "tcp://localhost:13232"
```

```
class NNGSocketHandler(logging.handlers.QueueHandler):

    def __init__(self, uri):
        socket = pynng.Pub0(dial=uri, send_timeout=500)
        super().__init__(socket)

    def enqueue(self, record):
        # Send the record as UTF-8 encoded JSON
        d = dict(record.__dict__)
        data = json.dumps(d)
        self.queue.send(data.encode('utf-8'))

    def close(self):
        self.queue.close()

logging.getLogger('pynng').propagate = False
handler = NNGSocketHandler(DEFAULT_ADDR)
# Make sure the process ID is in the output
logging.basicConfig(level=logging.DEBUG,
                    handlers=[logging.StreamHandler(), h
                    format='%(levelname)-8s %(name)10s %
levels = (logging.DEBUG, logging.INFO, logging.WARNING,
          logging.CRITICAL)
logger_names = ('myapp', 'myapp.lib1', 'myapp.lib2')
msgno = 1
while True:
    # Just randomly select some loggers and levels and l
    level = random.choice(levels)
    logger = logging.getLogger(random.choice(logger_name
    logger.log(level, 'Message no. %5d' % msgno)
    msgno += 1
    delay = random.random() * 2 + 0.5
    time.sleep(delay)
```

You can run the above two snippets in separate command shells. If we run the listener in one shell and run the sender in two separate shells, we should see something like the following. In the first sender shell:

```
$ python sender.py
DEBUG         myapp    613 Message no.     1
WARNING  myapp.lib2    613 Message no.     2
CRITICAL myapp.lib2    613 Message no.     3
WARNING  myapp.lib2    613 Message no.     4
CRITICAL myapp.lib1    613 Message no.     5
DEBUG         myapp    613 Message no.     6
CRITICAL myapp.lib1    613 Message no.     7
INFO     myapp.lib1    613 Message no.     8
(and so on)
```

In the second sender shell:

```
$ python sender.py
INFO     myapp.lib2    657 Message no.     1
CRITICAL myapp.lib2    657 Message no.     2
CRITICAL      myapp    657 Message no.     3
CRITICAL myapp.lib1    657 Message no.     4
INFO     myapp.lib1    657 Message no.     5
WARNING  myapp.lib2    657 Message no.     6
CRITICAL      myapp    657 Message no.     7
DEBUG    myapp.lib1    657 Message no.     8
(and so on)
```

In the listener shell:

```
$ python listener.py
Press Ctrl-C to stop.
DEBUG         myapp    613 Message no.     1
WARNING  myapp.lib2    613 Message no.     2
INFO     myapp.lib2    657 Message no.     1
CRITICAL myapp.lib2    613 Message no.     3
CRITICAL myapp.lib2    657 Message no.     2
CRITICAL      myapp    657 Message no.     3
WARNING  myapp.lib2    613 Message no.     4
CRITICAL myapp.lib1    613 Message no.     5
CRITICAL myapp.lib1    657 Message no.     4
INFO     myapp.lib1    657 Message no.     5
DEBUG         myapp    613 Message no.     6
```

```
WARNING  myapp.lib2    657 Message no.     6
CRITICAL      myapp    657 Message no.     7
CRITICAL myapp.lib1    613 Message no.     7
INFO     myapp.lib1    613 Message no.     8
DEBUG    myapp.lib1    657 Message no.     8
(and so on)
```

As you can see, the logging from the two sender processes is interleaved in the listener’s output.

Una configuración de ejemplo basada en diccionario

A continuación se muestra un ejemplo de un diccionario de configuración de logging, tomado de la documentación del proyecto Django [https://docs.djangoproject.com/en/1.9/topics/logging/#configuring-logging]. Este diccionario se pasa a `dictConfig()` para poner en efecto la configuración:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'verbose': {
            'format': '{levelname} {asctime} {module} {p
            'style': '{',
        },
        'simple': {
            'format': '{levelname} {message}',
            'style': '{',
        },
    },
    'filters': {
        'special': {
            '()': 'project.logging.SpecialFilter',
            'foo': 'bar',
        },
    },
    'handlers': {
        'console': {
            'level': 'INFO',
            'class': 'logging.StreamHandler',
            'formatter': 'simple',
        },
        'mail_admins': {
            'level': 'ERROR',
            'class': 'django.utils.log.AdminEmailHandler
            'filters': ['special']
        }
    },
    'loggers': {
        'django': {
            'handlers': ['console'],
            'propagate': True,
        },
        'django.request': {
            'handlers': ['mail_admins'],
            'level': 'ERROR',
            'propagate': False,
        },
        'myproject.custom': {
            'handlers': ['console', 'mail_admins'],
            'level': 'INFO',
            'filters': ['special']
        }
    }
}
```

Para obtener más información sobre esta configuración, puede ver la sección correspondiente [https://docs.djangoproject.com/en/1.9/topics/logging/#configuring-logging] de la documentación de Django.

Usar un rotador y un nombre para personalizar el procesamiento de rotación de log

An example of how you can define a namer and rotator is given in the following runnable script, which shows gzip compression of the log file:

```
import gzip
import logging
import logging.handlers
import os
import shutil

def namer(name):
    return name + ".gz"

def rotator(source, dest):
    with open(source, 'rb') as f_in:
        with gzip.open(dest, 'wb') as f_out:
            shutil.copyfileobj(f_in, f_out)
    os.remove(source)


rh = logging.handlers.RotatingFileHandler('rotated.log',
rh.rotator = rotator
rh.namer = namer

root = logging.getLogger()
root.setLevel(logging.INFO)
root.addHandler(rh)
f = logging.Formatter('%(asctime)s %(message)s')
rh.setFormatter(f)
for i in range(1000):
    root.info(f'Message no. {i + 1}')
```

After running this, you will see six new files, five of which are compressed:

```
$ ls rotated.log*
rotated.log       rotated.log.2.gz  rotated.log.4.gz
rotated.log.1.gz  rotated.log.3.gz  rotated.log.5.gz
$ zcat rotated.log.1.gz
```

```
2023-01-20 02:28:17,767 Message no. 996
2023-01-20 02:28:17,767 Message no. 997
2023-01-20 02:28:17,767 Message no. 998
```

Un ejemplo de multiprocesamiento más elaborado

El siguiente ejemplo de trabajo muestra cómo logging se puede usar con multiprocesamiento usando archivos de configuración. Las configuraciones son bastante simples, pero sirven para ilustrar cómo se podrían implementar las más complejas en un escenario real de multiprocesamiento.

En el ejemplo, el proceso principal genera un proceso de escucha y algunos procesos de trabajo. Cada uno de los procesos principales, el oyente y los trabajadores tienen tres configuraciones separadas (todos los trabajadores comparten la misma configuración). Podemos ver el registro en el proceso principal, cómo los trabajadores se registran en un QueueHandler y cómo el oyente implementa un QueueListener y una configuración de registro más compleja, y organiza el envío de eventos recibidos a través de la cola a los controladores especificados en la configuración. Tenga en cuenta que estas configuraciones son puramente ilustrativas, pero debería poder adaptar este ejemplo a su propio escenario.

Aquí está el script, el docstrings y los comentarios, esperemos, expliquen cómo funciona:

```
import logging
import logging.config
import logging.handlers
from multiprocessing import Process, Queue, Event, curre
import os
import random
import time

class MyHandler:
    """
    A simple handler for logging events. It runs in the
    dispatches events to loggers based on the name in th
    which then get dispatched, by the logging system, to
    configured for those loggers.
    """

    def handle(self, record):
        if record.name == "root":
            logger = logging.getLogger()
        else:
            logger = logging.getLogger(record.name)

        if logger.isEnabledFor(record.levelno):
            # The process name is transformed just to sh
            # doing the logging to files and console
            record.processName = '%s (for %s)' % (curren
            logger.handle(record)

def listener_process(q, stop_event, config):
    """
    This could be done in the main process, but is just
    process for illustrative purposes.

    This initialises logging according to the specified
    starts the listener and waits for the main process t
    via the event. The listener is then stopped, and the
    """
    logging.config.dictConfig(config)
    listener = logging.handlers.QueueListener(q, MyHandl
    listener.start()
    if os.name == 'posix':
        # On POSIX, the setup logger will have been conf
        # parent process, but should have been disabled
        # dictConfig call.
        # On Windows, since fork isn't used, the setup l
        # exist in the child, so it would be created and
        # would appear - hence the "if posix" clause.
        logger = logging.getLogger('setup')
        logger.critical('Should not appear, because of d
    stop_event.wait()
    listener.stop()

def worker_process(config):
    """
    A number of these are spawned for the purpose of ill
    practice, they could be a heterogeneous bunch of pro
    ones which are identical to each other.

    This initialises logging according to the specified
    and logs a hundred messages with random levels to ra
    loggers.

    A small sleep is added to allow other processes a ch
    is not strictly needed, but it mixes the output from
    processes a bit more than if it's left out.
    """
    logging.config.dictConfig(config)
    levels = [logging.DEBUG, logging.INFO, logging.WARNI
              logging.CRITICAL]
    loggers = ['foo', 'foo.bar', 'foo.bar.baz',
               'spam', 'spam.ham', 'spam.ham.eggs']
    if os.name == 'posix':
        # On POSIX, the setup logger will have been conf
        # parent process, but should have been disabled
        # dictConfig call.
        # On Windows, since fork isn't used, the setup l
        # exist in the child, so it would be created and
        # would appear - hence the "if posix" clause.
        logger = logging.getLogger('setup')
        logger.critical('Should not appear, because of d
    for i in range(100):
        lvl = random.choice(levels)
        logger = logging.getLogger(random.choice(loggers
        logger.log(lvl, 'Message no. %d', i)
        time.sleep(0.01)

def main():
```

```
q = Queue()
# The main process gets a simple configuration which
config_initial = {
    'version': 1,
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'level': 'INFO'
        }
    },
    'root': {
        'handlers': ['console'],
        'level': 'DEBUG'
    }
}
# The worker process configuration is just a QueueHa
# root logger, which allows all messages to be sent
# We disable existing loggers to disable the "setup"
# parent process. This is needed on POSIX because th
# be there in the child following a fork().
config_worker = {
    'version': 1,
    'disable_existing_loggers': True,
    'handlers': {
        'queue': {
            'class': 'logging.handlers.QueueHandler'
            'queue': q
        }
    },
    'root': {
        'handlers': ['queue'],
        'level': 'DEBUG'
    }
}
# The listener process configuration shows that the
# logging configuration is available to dispatch eve
# you want.
# We disable existing loggers to disable the "setup"
```

```
    # parent process. This is needed on POSIX because th
    # be there in the child following a fork().
    config_listener = {
        'version': 1,
        'disable_existing_loggers': True,
        'formatters': {
            'detailed': {
                'class': 'logging.Formatter',
                'format': '%(asctime)s %(name)-15s %(lev
            },
            'simple': {
                'class': 'logging.Formatter',
                'format': '%(name)-15s %(levelname)-8s %
            }
        },
        'handlers': {
            'console': {
                'class': 'logging.StreamHandler',
                'formatter': 'simple',
                'level': 'INFO'
            },
            'file': {
                'class': 'logging.FileHandler',
                'filename': 'mplog.log',
                'mode': 'w',
                'formatter': 'detailed'
            },
            'foofile': {
                'class': 'logging.FileHandler',
                'filename': 'mplog-foo.log',
                'mode': 'w',
                'formatter': 'detailed'
            },
            'errors': {
                'class': 'logging.FileHandler',
                'filename': 'mplog-errors.log',
                'mode': 'w',
                'formatter': 'detailed',
            'level': 'ERROR'
        }
    },
    'loggers': {
        'foo': {
            'handlers': ['foofile']
        }
    },
    'root': {
        'handlers': ['console', 'file', 'errors'],
        'level': 'DEBUG'
    }
}
# Log some initial events, just to show that logging
# normally.
logging.config.dictConfig(config_initial)
logger = logging.getLogger('setup')
logger.info('About to create workers ...')
workers = []
for i in range(5):
    wp = Process(target=worker_process, name='worker
                 args=(config_worker,))
    workers.append(wp)
    wp.start()
    logger.info('Started worker: %s', wp.name)
logger.info('About to create listener ...')
stop_event = Event()
lp = Process(target=listener_process, name='listener
             args=(q, stop_event, config_listener))
lp.start()
logger.info('Started listener')
# We now hang around for the workers to finish their
for wp in workers:
    wp.join()
# Workers all done, listening can now stop.
# Logging in the parent still works normally.
logger.info('Telling listener to stop ...')
stop_event.set()
```

```
    lp.join()
    logger.info('All done.')

if __name__ == '__main__':
    main()
```

Insertar BOM en mensajes enviados a SysLogHandler

RFC 5424 [https://datatracker.ietf.org/doc/html/rfc5424.html] requiere que se envíe un mensaje Unicode a un demonio syslog como un conjunto de bytes que tienen la siguiente estructura: un componente opcional ASCII puro, seguido de una marca de orden de bytes UTF-8 (BOM), seguida de Codificado en Unicode usando UTF-8. (Ver sección relevante de la especificación RFC 5424 Section 6 [https://datatracker.ietf.org/doc/html/rfc5424.html#section-6].)

En Python 3.1, se agregó código a `SysLogHandler` para insertar BOM en el mensaje, pero desafortunadamente, se implementó incorrectamente, BOM aparece al principio del mensaje y, por lo tanto, no permite ningún componente ASCII puro para que aparezca antes.

Como este comportamiento no funciona, el código de inserción BOM incorrecto se elimina de Python 3.2.4 y versiones posteriores. Sin embargo, no se está reemplazando, y si desea producir mensajes compatibles con RFC 5424 [https://datatracker.ietf.org/doc/html/rfc5424.html] que incluyan BOM, una secuencia opcional de ASCII puro antes y Unicode arbitrario después, codificados usando UTF-8; entonces necesita hacer lo siguiente:

1. Adjunte una instancia de `Formatter` a su instancia `SysLogHandler`, con una cadena de caracteres de formato como:

   ```
   'ASCII section\ufeffUnicode section'
   ```

   El punto de código Unicode U+ FEFF, cuando se codifica usando UTF-8, se codificará como una BOM UTF-8, la cadena de bytes `b'\xef\xbb\xbf'`.

2. Reemplace la sección ASCII con los marcadores de posición que desee, pero asegúrese de que los datos que aparecen allí después de la sustitución sean siempre ASCII (de esa manera, permanecerán sin cambios después de la codificación UTF-8).

3. Reemplace la sección Unicode con los marcadores de posición que desee; si los datos que aparecen allí después de la sustitución contienen caracteres fuera del rango ASCII, está bien: se codificarán usando UTF-8.

El mensaje formateado *se* codificará utilizando la codificación UTF-8 por `SysLogHandler`. Si sigue las reglas anteriores, debería poder producir mensajes compatibles con **RFC 5424** [https://datatracker.ietf.org/doc/html/rfc5424.html]. Si no lo hace, es posible que el logging no se queje, pero sus mensajes no serán compatibles con RFC 5424 y su demonio syslog puede quejarse.

Implementar logging estructurado

Aunque la mayoría de los mensajes de registro están destinados a ser leídos por humanos y, por lo tanto, no se pueden analizar fácilmente mediante una máquina, puede haber circunstancias en las que desee generar mensajes en un formato estructurado que *sea* capaz de ser analizado por un programa (sin necesidad de expresiones regulares complejas para analizar el mensaje de registro). Esto es sencillo de lograr utilizando el paquete de registro. Hay varias formas de lograr esto, pero el siguiente es un enfoque simple que usa JSON para serializar el evento de una manera analizable por máquina:

```
import json
import logging

class StructuredMessage:
    def __init__(self, message, /, **kwargs):
        self.message = message
        self.kwargs = kwargs
    def __str__(self):
        return '%s >>> %s' % (self.message, json.dumps(s

_ = StructuredMessage   # optional, to improve readabili

logging.basicConfig(level=logging.INFO, format='%(messag
logging.info(_('message 1', foo='bar', bar='baz', num=12
```

Si se ejecuta el script anterior, se imprime:

```
message 1 >>> {"fnum": 123.456, "num": 123, "bar": "baz"
```

Tenga en cuenta que el orden de los elementos puede ser diferente según la versión de Python utilizada.

Si necesita un procesamiento más especializado, puede utilizar un codificador JSON personalizado, como en el siguiente ejemplo completo:

```
import json
import logging


class Encoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, set):
            return tuple(o)
        elif isinstance(o, str):
            return o.encode('unicode_escape').decode('as
        return super().default(o)

class StructuredMessage:
    def __init__(self, message, /, **kwargs):
        self.message = message
        self.kwargs = kwargs

    def __str__(self):
        s = Encoder().encode(self.kwargs)
        return '%s >>> %s' % (self.message, s)
```

```
_ = StructuredMessage   # optional, to improve readabili

def main():
    logging.basicConfig(level=logging.INFO, format='%(me
    logging.info(_('message 1', set_value={1, 2, 3}, sno

if __name__ == '__main__':
    main()
```

Cuando se ejecuta el script anterior, se imprime:

```
message 1 >>> {"snowman": "\u2603", "set_value": [1, 2,
```

Tenga en cuenta que el orden de los elementos puede ser diferente según la versión de Python utilizada.

Personalización de gestores con `dictConfig()`

Hay ocasiones en las que desea personalizar los gestores de logging de formas particulares, y si usa `dictConfig()` puede hacerlo sin subclases. Como ejemplo, considere que es posible que desee establecer la propiedad de un archivo de log. En POSIX, esto se hace fácilmente usando `shutil.chown()`, pero los gestores de archivos en stdlib no ofrecen soporte integrado. Puede personalizar la creación de gestores usando una función simple como:

```
def owned_file_handler(filename, mode='a', encoding=None
    if owner:
        if not os.path.exists(filename):
            open(filename, 'a').close()
        shutil.chown(filename, *owner)
    return logging.FileHandler(filename, mode, encoding)
```

Luego puede especificar, en una configuración de logging pasada a `dictConfig()`, que se cree un gestor de logging llamando a esta función:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'default': {
            'format': '%(asctime)s %(levelname)s %(name)
        },
    },
    'handlers': {
        'file':{
            # The values below are popped from this dict
            # used to create the handler, set the handle
            # its formatter.
            '()': owned_file_handler,
            'level':'DEBUG',
            'formatter': 'default',
            # The values below are passed to the handler
            # as keyword arguments.
            'owner': ['pulse', 'pulse'],
            'filename': 'chowntest.log',
            'mode': 'w',
            'encoding': 'utf-8',
        },
    },
    'root': {
        'handlers': ['file'],
        'level': 'DEBUG',
    },
}
```

En este ejemplo, se establece la propiedad utilizando el usuario y el grupo `pulse`, solo con fines ilustrativos. Poniéndolo junto en un script de trabajo, `chowntest.py`:

```
import logging, logging.config, os, shutil

def owned_file_handler(filename, mode='a', encoding=None
    if owner:
        if not os.path.exists(filename):
            open(filename, 'a').close()
        shutil.chown(filename, *owner)
    return logging.FileHandler(filename, mode, encoding)

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'default': {
            'format': '%(asctime)s %(levelname)s %(name)
        },
    },
    'handlers': {
        'file':{
            # The values below are popped from this dict
            # used to create the handler, set the handle
            # its formatter.
            '()': owned_file_handler,
            'level':'DEBUG',
            'formatter': 'default',
            # The values below are passed to the handler
            # as keyword arguments.
            'owner': ['pulse', 'pulse'],
            'filename': 'chowntest.log',
            'mode': 'w',
            'encoding': 'utf-8',
        },
    },
    'root': {
        'handlers': ['file'],
        'level': 'DEBUG',
    },
}

logging.config.dictConfig(LOGGING)
logger = logging.getLogger('mylogger')
logger.debug('A debug message')
```

Para ejecutar esto, probablemente se necesite ejecutarlo como `root`:

```
$ sudo python3.3 chowntest.py
$ cat chowntest.log
2013-11-05 09:34:51,128 DEBUG mylogger A debug message
$ ls -l chowntest.log
-rw-r--r-- 1 pulse pulse 55 2013-11-05 09:34 chowntest.l
```

Tenga en cuenta que este ejemplo usa Python 3.3 porque ahí es donde **`shutil.chown()`** aparece. Este enfoque debería funcionar con cualquier versión de Python que admita **`dictConfig()`**, es decir, Python 2.7, 3.2 o posterior. Con las versiones anteriores a 3.3, necesitaría implementar el cambio de propiedad real usando, por ejemplo, **`os.chown()`**.

En la práctica, la función de creación de gestores puede estar en un módulo de utilidad en algún lugar de su proyecto. En lugar de la línea en la configuración:

```
'()': owned_file_handler,
```

podrías usar, por ejemplo,:

```
'()': 'ext://project.util.owned_file_handler',
```

donde `project.util` se puede reemplazar con el nombre real del paquete donde reside la función. En el script de trabajo anterior, el uso de `'ext://__main__.owned_file_handler'` debería funcionar. Aquí, el invocable real se resuelve mediante **`dictConfig()`** de la especificación `ext://`.

Por fortuna, este ejemplo también indica el camino hacia cómo podría implementar otros tipos de cambio de archivo, por ejemplo, configurando de la misma manera bits de permisos POSIX específicos, usando **`os.chmod()`**.

Por supuesto, el enfoque también podría extenderse a tipos de gestores distintos a **`FileHandler`** - por ejemplo, uno de los gestores de archivos rotativos, o un tipo diferente por completo.

Usar estilos de formato particulares en toda su aplicación

En Python 3.2, **`Formatter`** obtuvo un parámetro de palabra clave `estilo` que, aunque por defecto era `%` para compatibilidad con versiones anteriores, permitía la especificación de `{` o `$` para permitir los enfoques de formato admitidos por **`str.format()`** y **`string.Template`**. Tenga en cuenta que esto rige el formato de los mensajes de logging para la salida final a los logging y es completamente ortogonal a cómo se construye un mensaje de logging individual.

Logging calls (**`debug()`**, **`info()`** etc.) only take positional parameters for the actual logging message itself, with keyword parameters used only for determining options for how to handle the logging call (e.g. the `exc_info` keyword parameter to indicate that traceback information should be logged, or the `extra` keyword parameter to indicate additional contextual information to be added to the log). So you cannot directly make logging calls using **`str.format()`** or **`string.Template`** syntax, because internally the logging package uses %-formatting to merge the format string and the variable arguments. There would be no changing this while preserving backward compatibility, since all logging calls which are out there in existing code will be using %-format strings.

Ha habido sugerencias para asociar estilos de formato con *loggers* específicos, pero ese enfoque también tiene problemas de compatibilidad con versiones anteriores porque cualquier código existente podría estar usando un nombre de *logger* dado y usando formato %.

Para que logging funcione de manera interoperable entre cualquier biblioteca de terceros y su código, las decisiones sobre el formato deben tomarse a nivel de la llamada de logging individual. Esto abre un par de formas en las que se pueden acomodar estilos de formato alternativos.

Uso de fábricas de LogRecord

En Python 3.2, junto con los cambios de **`Formatter`** mencionados anteriormente, el paquete logging ganó la capacidad de permitir a los usuarios establecer sus propias subclases **`LogRecord`**, usando la función **`setLogRecordFactory()`**. Puede usar esto para configurar su propia subclase de **`LogRecord`**, que hace lo correcto al anular el método **`getMessage()`**. La implementación de la clase base de este método es donde ocurre el formato `msg % args` y donde puede sustituir su formato alternativo; sin embargo, debe tener cuidado de admitir todos los estilos de formato y permitir formato % como predeterminado, para garantizar la interoperabilidad con otro código. También se debe tener cuidado de llamar a `str(self.msg)`, tal como lo hace la implementación base.

Consulte la documentación de referencia en **`setLogRecordFactory()`** y **`LogRecord`** para obtener más información.

Usar objetos de mensaje personalizados

Existe otra forma, quizás más sencilla, de usar el formato {} - y $ - para construir sus mensajes de log individuales. Al iniciar sesión, recuerde que puede usar cualquier objeto como una cadena de caracteres de formato de mensaje (Usando objetos arbitrarios como mensajes) que al iniciar sesión puede usar un objeto arbitrario como una cadena de formato de mensaje, y que el paquete de logging llamará **`str()`** en ese objeto para obtener el cadena de formato real. Considere las siguientes dos clases:

```
class BraceMessage:
    def __init__(self, fmt, /, *args, **kwargs):
        self.fmt = fmt
        self.args = args
        self.kwargs = kwargs

    def __str__(self):
        return self.fmt.format(*self.args, **self.kwargs

class DollarMessage:
    def __init__(self, fmt, /, **kwargs):
        self.fmt = fmt
        self.kwargs = kwargs
    def __str__(self):
        from string import Template
        return Template(self.fmt).substitute(**self.kwar
```

Cualquiera de estos puede usarse en lugar de una cadena de formato, para permitir que se use el formato {} - o $ - para construir la parte del «mensaje» real que aparece en la salida del log formateado en lugar de “%(message)s” or “{message}” or “$message”. Si le resulta un poco difícil de manejar usar los nombres de las clases cada vez que desea registrar algo, puede hacerlo más tolerable si usa un alias como `M` o `_` para el mensaje (o quizás `__`, si está utilizando `_` para localización).

A continuación se dan ejemplos de este enfoque. En primer lugar, formatear con **`str.format()`**:

```
>>> __ = BraceMessage
>>> print(__('Message with {0} {1}', 2, 'placeholders'))
Message with 2 placeholders
>>> class Point: pass
...
>>> p = Point()
>>> p.x = 0.5
>>> p.y = 0.5
>>> print(__('Message with coordinates: ({point.x:.2f},
Message with coordinates: (0.50, 0.50)
```

En segundo lugar, formatear con **`string.Template`**:

```
>>> __ = DollarMessage
>>> print(__('Message with $num $what', num=2, what='pla
Message with 2 placeholders
>>>
```

One thing to note is that you pay no significant performance penalty with this approach: the actual formatting happens not when you make the logging call, but when (and if) the logged message is actually about to be output to a log by a handler. So the only slightly unusual thing which might trip you up is that the parentheses go around the format string and the arguments, not just the format string. That’s because the _ notation is just syntax sugar for a constructor call to one of the `XXXMessage` classes shown above.

Configurar filtros con `dictConfig()`

Puedes configurar filtros usando `dictConfig()`, aunque a primera vista es posible que no sea obvio cómo hacerlo (de ahí esta receta). Dado que `Filter` es la única clase de filtro incluida en la biblioteca estándar, y es poco probable que satisfaga muchos requisitos (solo está allí como clase base), normalmente necesitarás definir tu propia subclase **`Filter`** con un método `filter()` sobreescrito. Para hacer esto, especifique la clave `()` en el diccionario de configuración para el filtro, especificando un invocable que se usará para crear el filtro (una clase es la más obvia, pero puede proporcionar cualquier invocable que retorne una instancia **`Filter`**). Aquí hay un ejemplo completo:

```
import logging
import logging.config
import sys

class MyFilter(logging.Filter):
    def __init__(self, param=None):
        self.param = param

    def filter(self, record):
        if self.param is None:
            allow = True
        else:
            allow = self.param not in record.msg
        if allow:
            record.msg = 'changed: ' + record.msg
        return allow

LOGGING = {
    'version': 1,
    'filters': {
        'myfilter': {
                '()': MyFilter,
                'param': 'noshow',
            }
        },
        'handlers': {
            'console': {
                'class': 'logging.StreamHandler',
                'filters': ['myfilter']
            }
        },
        'root': {
            'level': 'DEBUG',
            'handlers': ['console']
        },
    }

if __name__ == '__main__':
    logging.config.dictConfig(LOGGING)
    logging.debug('hello')
    logging.debug('hello - noshow')
```

Este ejemplo muestra cómo puede pasar datos de configuración al invocable que construye la instancia, en forma de parámetros de palabras clave. Cuando se ejecuta, se imprimirá el script anterior:

```
changed: hello
```

que muestra que el filtro está funcionando según lo configurado.

Un par de puntos adicionales a tener en cuenta:

- Si no puede hacer referencia al invocable directamente en la configuración (por ejemplo, si vive en un módulo diferente y no puede importarlo directamente donde está el diccionario de configuración), puede usar el formulario `ext://...` como se describe en Acceso a objetos externos. Por ejemplo, podría haber usado el texto `'ext://__main__.MyFilter'` en lugar de `MyFilter` en el ejemplo anterior.
- Además de los filtros, esta técnica también se puede utilizar

para configurar gestores y formateadores personalizados. Consultar Objetos definidos por el usuario para obtener más información sobre cómo logging admite el uso de objetos definidos por el usuario en su configuración, y ver arriba la otra receta Personalización de gestores con dictConfig().

Formato de excepción personalizado

Puede haber ocasiones en las que desee personalizar un formato de excepción; por el bien del argumento, digamos que desea exactamente una línea por evento registrado, incluso cuando la información de la excepción está presente. Puede hacer esto con una clase de formateador personalizada, como se muestra en el siguiente ejemplo:

```
import logging

class OneLineExceptionFormatter(logging.Formatter):
    def formatException(self, exc_info):
        """
        Format an exception so that it prints on a singl
        """
        result = super().formatException(exc_info)
        return repr(result)  # or format into one line h

    def format(self, record):
        s = super().format(record)
        if record.exc_text:
            s = s.replace('\n', '') + '|'
        return s

def configure_logging():
    fh = logging.FileHandler('output.txt', 'w')
    f = OneLineExceptionFormatter('%(asctime)s|%(levelna
                                  '%d/%m/%Y %H:%M:%S')
    fh.setFormatter(f)
    root = logging.getLogger()
    root.setLevel(logging.DEBUG)
    root.addHandler(fh)
```

```
def main():
    configure_logging()
    logging.info('Sample message')
    try:
        x = 1 / 0
    except ZeroDivisionError as e:
        logging.exception('ZeroDivisionError: %s', e)

if __name__ == '__main__':
    main()
```

Cuando se ejecuta, esto produce un archivo con exactamente dos líneas:

```
28/01/2015 07:21:23|INFO|Sample message|
28/01/2015 07:21:23|ERROR|ZeroDivisionError: division by
```

Si bien el tratamiento anterior es simplista, señala el camino hacia cómo se puede formatear la información de excepción a su gusto. El módulo `traceback` puede resultar útil para necesidades más especializadas.

Mensajes de logging hablantes

Puede haber situaciones en las que sea deseable que los mensajes de logging se presenten en un formato audible en lugar de visible. Esto es fácil de hacer si tiene la funcionalidad de texto a voz (*TTS* por sus siglas en inglés) disponible en su sistema, incluso si no tiene un *binding* Python. La mayoría de los sistemas TTS tienen un programa de línea de comandos que puede ejecutar, y esto puede invocarse desde un gestor usando `subprocess`. Aquí se asume que los programas de línea de comando TTS no esperarán interactuar con los usuarios o tardarán mucho en completarse, y que la frecuencia de los mensajes registrados no será tan alta como para inundar al usuario con mensajes, y que es aceptable que los mensajes se reproducen uno a la vez en lugar de todos al mismo tiempo. La implementación de ejemplo a continuación espera a que se pronuncie un mensaje antes de que se procese el siguiente, y esto puede hacer que otros gestores se mantengan esperando. Aquí hay un breve ejemplo que muestra el enfoque, que asume que el paquete TTS `espeak` está disponible:

```
import logging
import subprocess
import sys

class TTSHandler(logging.Handler):
    def emit(self, record):
        msg = self.format(record)
        # Speak slowly in a female English voice
        cmd = ['espeak', '-s150', '-ven+f3', msg]
        p = subprocess.Popen(cmd, stdout=subprocess.PIPE
                             stderr=subprocess.STDOUT)
        # wait for the program to finish
        p.communicate()

def configure_logging():
    h = TTSHandler()
    root = logging.getLogger()
    root.addHandler(h)
    # the default formatter just returns the message
    root.setLevel(logging.DEBUG)

def main():
    logging.info('Hello')
    logging.debug('Goodbye')

if __name__ == '__main__':
    configure_logging()
    sys.exit(main())
```

Cuando se ejecute, este script debería decir «Hola» y luego «Adiós» con voz femenina.

El enfoque anterior puede, por supuesto, adaptarse a otros sistemas TTS e incluso a otros sistemas que pueden procesar mensajes a través de programas externos ejecutados desde una línea de comando.

Almacenamiento en búfer de mensajes de logging y su salida condicional

Puede haber situaciones en las que desee registrar mensajes en un área temporal y solo mostrarlos si se produce una determinada condición. Por ejemplo, es posible que desee comenzar a registrar eventos de depuración en una función, y si la función se completa sin errores, no desea saturar el log con la información de depuración recopilada; pero si hay un error, desea toda la información de depuración información así como el error.

Aquí hay un ejemplo que muestra cómo puede hacer esto usando un decorador para sus funciones donde desea que el logging se comporte de esta manera. Hace uso de **`logging.handlers.MemoryHandler`**, que permite el almacenamiento en búfer de eventos registrados hasta que se produzca alguna condición, momento en el que los eventos almacenados en búfer se `flushed` y se pasan a otro gestor (el gestor `target`) para su procesamiento. De forma predeterminada, el `MemoryHandler` se vacía cuando su búfer se llena o se ve un evento cuyo nivel es mayor o igual a un umbral especificado. Puede usar esta receta con una subclase más especializada de `MemoryHandler` si desea un comportamiento de descarga personalizado.

El script de ejemplo tiene una función simple, `foo`, que recorre todos los niveles de logging, escribiendo en `sys.stderr` para decir en qué nivel está a punto de *loguear* y luego registrar un mensaje en ese nivel. Puede pasar un parámetro a `foo` que, si es verdadero, se registrará en los niveles ERROR y CRITICAL; de lo contrario, solo registrará en los niveles DEBUG, INFO y WARNING.

El script simplemente dispone decorar `foo` con un decorador que hará el logging condicional que se requiere. El decorador toma un registrador como parámetro y adjunta un gestor de memoria durante la duración de la llamada a la función decorada. El decorador se puede parametrizar adicionalmente utilizando un gestor *target*, un nivel en el que debe producirse el vaciado y una capacidad para el búfer (número de registros almacenados en búfer). Estos están predeterminados a **`StreamHandler`** que escribe en `sys.stderr`, `logging.ERROR` y `100` respectivamente.

Aquí está el script:

```
import logging
from logging.handlers import MemoryHandler
import sys

logger = logging.getLogger(__name__)
logger.addHandler(logging.NullHandler())

def log_if_errors(logger, target_handler=None, flush_lev
    if target_handler is None:
        target_handler = logging.StreamHandler()
    if flush_level is None:
        flush_level = logging.ERROR
    if capacity is None:
        capacity = 100
    handler = MemoryHandler(capacity, flushLevel=flush_l

    def decorator(fn):
        def wrapper(*args, **kwargs):
            logger.addHandler(handler)
            try:
                return fn(*args, **kwargs)
            except Exception:
                logger.exception('call failed')
                raise
            finally:
                super(MemoryHandler, handler).flush()
                logger.removeHandler(handler)
        return wrapper

    return decorator
```

```
def write_line(s):
    sys.stderr.write('%s\n' % s)

def foo(fail=False):
    write_line('about to log at DEBUG ...')
    logger.debug('Actually logged at DEBUG')
    write_line('about to log at INFO ...')
    logger.info('Actually logged at INFO')
    write_line('about to log at WARNING ...')
    logger.warning('Actually logged at WARNING')
    if fail:
        write_line('about to log at ERROR ...')
        logger.error('Actually logged at ERROR')
        write_line('about to log at CRITICAL ...')
        logger.critical('Actually logged at CRITICAL')
    return fail

decorated_foo = log_if_errors(logger)(foo)

if __name__ == '__main__':
    logger.setLevel(logging.DEBUG)
    write_line('Calling undecorated foo with False')
    assert not foo(False)
    write_line('Calling undecorated foo with True')
    assert foo(True)
    write_line('Calling decorated foo with False')
    assert not decorated_foo(False)
    write_line('Calling decorated foo with True')
    assert decorated_foo(True)
```

Cuando se ejecuta este script, se debe observar el siguiente resultado:

```
Calling undecorated foo with False
about to log at DEBUG ...
about to log at INFO ...
about to log at WARNING ...
Calling undecorated foo with True
about to log at DEBUG ...
about to log at INFO ...
about to log at WARNING ...
about to log at ERROR ...
about to log at CRITICAL ...
Calling decorated foo with False
about to log at DEBUG ...
about to log at INFO ...
about to log at WARNING ...
Calling decorated foo with True
about to log at DEBUG ...
about to log at INFO ...
about to log at WARNING ...
about to log at ERROR ...
Actually logged at DEBUG
Actually logged at INFO
Actually logged at WARNING
Actually logged at ERROR
about to log at CRITICAL ...
Actually logged at CRITICAL
```

Como puede ver, la salida real de logging solo ocurre cuando se registra un evento cuya gravedad es ERROR o mayor, pero en ese caso, también se registra cualquier evento anterior con una gravedad menor.

Por supuesto, puede utilizar las formas de decoración convencionales:

```
@log_if_errors(logger)
def foo(fail=False):
    ...
```

Enviando mensajes de logging al correo electrónico, con almacenamiento en búfer

Para ilustrar cómo puede enviar mensajes de logging por correo electrónico, de modo que se envíe un número determinado de mensajes, puede usar la subclase **`BufferingHandler`**. En el siguiente ejemplo, que puedes adaptar a tus necesidades específicas, se proporciona un sencillo arnés de pruebas que permite ejecutar el script con argumentos de línea de comandos especificando lo que normalmente necesitas para enviar cosas por SMTP. (Ejecute el script descargado con el argumento `-h` para ver los argumentos necesarios y opcionales).

```
import logging
import logging.handlers
import smtplib

class BufferingSMTPHandler(logging.handlers.BufferingHar
    def __init__(self, mailhost, port, username, passwor
                 subject, capacity):
        logging.handlers.BufferingHandler.__init__(self,
        self.mailhost = mailhost
        self.mailport = port
        self.username = username
        self.password = password
        self.fromaddr = fromaddr
        if isinstance(toaddrs, str):
            toaddrs = [toaddrs]
        self.toaddrs = toaddrs
        self.subject = subject
        self.setFormatter(logging.Formatter("%(asctime)s

    def flush(self):
        if len(self.buffer) > 0:
            try:
                smtp = smtplib.SMTP(self.mailhost, self.
                smtp.starttls()
                smtp.login(self.username, self.password)
                msg = "From: %s\r\nTo: %s\r\nSubject: %s
                for record in self.buffer:
                    s = self.format(record)
                    msg = msg + s + "\r\n"
                smtp.sendmail(self.fromaddr, self.toaddr
                smtp.quit()
            except Exception:
                if logging.raiseExceptions:
                    raise
            self.buffer = []

if __name__ == '__main__':
    import argparse

    ap = argparse.ArgumentParser()
    aa = ap.add_argument
    aa('host', metavar='HOST', help='SMTP server')
    aa('--port', '-p', type=int, default=587, help='SMTP
    aa('user', metavar='USER', help='SMTP username')
    aa('password', metavar='PASSWORD', help='SMTP passwo
    aa('to', metavar='TO', help='Addressee for emails')
    aa('sender', metavar='SENDER', help='Sender email ad
    aa('--subject', '-s',
       default='Test Logging email from Python logging m
       help='Subject of email')
    options = ap.parse_args()
    logger = logging.getLogger()
    logger.setLevel(logging.DEBUG)
    h = BufferingSMTPHandler(options.host, options.port,
                             options.password, options.s
                             options.to, options.subject
    logger.addHandler(h)
    for i in range(102):
        logger.info("Info index = %d", i)
    h.flush()
    h.close()
```

Si ejecuta este script y su servidor SMTP está correctamente configurado, debería ver que envía once correos electrónicos al destinatario que usted especifique. Los primeros diez correos electrónicos tendrán cada uno diez mensajes de log, y el undécimo tendrá dos mensajes. Eso hace 102 mensajes como se especifica en el script.

Formateo de horas usando UTC (GMT) a

través de la configuración

A veces desea formatear las horas usando UTC, lo que se puede hacer usando una clase como `UTCFormatter`, como se muestra a continuación:

```
import logging
import time

class UTCFormatter(logging.Formatter):
    converter = time.gmtime
```

y luego puede usar el `UTCFormatter` en su código en lugar de **`Formatter`**. Si desea hacer eso a través de la configuración, puede usar la API **`dictConfig()`** con un enfoque ilustrado por el siguiente ejemplo completo:

```
import logging
import logging.config
import time

class UTCFormatter(logging.Formatter):
    converter = time.gmtime

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'utc': {
            '()': UTCFormatter,
            'format': '%(asctime)s %(message)s',
        },
        'local': {
            'format': '%(asctime)s %(message)s',
        }
    },
    'handlers': {
        'console1': {
            'class': 'logging.StreamHandler',
                'formatter': 'utc',
            },
            'console2': {
                'class': 'logging.StreamHandler',
                'formatter': 'local',
            },
        },
        'root': {
            'handlers': ['console1', 'console2'],
       }
}

if __name__ == '__main__':
    logging.config.dictConfig(LOGGING)
    logging.warning('The local time is %s', time.asctime
```

Cuando se ejecuta este script, debería imprimir algo como:

```
2015-10-17 12:53:29,501 The local time is Sat Oct 17 13:
2015-10-17 13:53:29,501 The local time is Sat Oct 17 13:
```

mostrando cómo se formatea la hora como hora local y UTC, una para cada gestor.

Usar un administrador de contexto para logging selectivo

Hay ocasiones en las que sería útil cambiar temporalmente la configuración de logging y revertir esto después de hacer algo. Para ello, un administrador de contexto es la forma más obvia de guardar y restaurar el contexto de logging. Aquí hay un ejemplo simple de un administrador de contexto de este tipo, que le permite cambiar opcionalmente el nivel de logging y agregar un gestor de logging exclusivamente en el alcance del administrador de contexto:

```
import logging
import sys

class LoggingContext:
    def __init__(self, logger, level=None, handler=None,
        self.logger = logger
        self.level = level
        self.handler = handler
        self.close = close

    def __enter__(self):
        if self.level is not None:
            self.old_level = self.logger.level
            self.logger.setLevel(self.level)
        if self.handler:
            self.logger.addHandler(self.handler)

    def __exit__(self, et, ev, tb):
        if self.level is not None:
            self.logger.setLevel(self.old_level)
        if self.handler:
            self.logger.removeHandler(self.handler)
        if self.handler and self.close:
            self.handler.close()
        # implicit return of None => don't swallow excep
```

Si especifica un valor de nivel, el nivel del registrador se establece en ese valor en el alcance del bloque *with* cubierto por el administrador de contexto. Si especifica un gestor, se agrega al registrador al entrar al bloque y se elimina al salir del bloque. También puede pedirle al administrador que cierre el gestor por usted al salir del bloque si ya no lo necesita.

Para ilustrar cómo funciona, podemos agregar el siguiente bloque de código al anterior:

```
if __name__ == '__main__':
    logger = logging.getLogger('foo')
    logger.addHandler(logging.StreamHandler())
    logger.setLevel(logging.INFO)
    logger.info('1. This should appear just once on stde
    logger.debug('2. This should not appear.')
    with LoggingContext(logger, level=logging.DEBUG):
        logger.debug('3. This should appear once on stde
    logger.debug('4. This should not appear.')
    h = logging.StreamHandler(sys.stdout)
    with LoggingContext(logger, level=logging.DEBUG, har
        logger.debug('5. This should appear twice - once
    logger.info('6. This should appear just once on stde
    logger.debug('7. This should not appear.')
```

Inicialmente configuramos el nivel del registrador en `INFO`, por lo que aparece el mensaje #1 y el mensaje #2 no. Luego cambiamos el nivel a `DEBUG` temporalmente en el siguiente bloque `with`, y aparece el mensaje #3. Una vez que se sale del bloque, el nivel del registrador se restaura a `INFO` y, por lo tanto, el mensaje #4 no aparece. En el siguiente bloque `with`, configuramos el nivel en `DEBUG` nuevamente, pero también agregamos un gestor que escribe en `sys.stdout`. Por lo tanto, el mensaje #5 aparece dos veces en la consola (una vez a través de `stderr` y una vez a través de `stdout`). Después de la finalización de la declaración `with`, se vuelve al estado anterior, por lo que aparece el mensaje #6 (como el mensaje #1) mientras que el mensaje #7 no (como el mensaje #2).

Si ejecutamos el script resultante, el resultado es el siguiente:

```
$ python logctx.py
1. This should appear just once on stderr.
3. This should appear once on stderr.
5. This should appear twice - once on stderr and once or
5. This should appear twice - once on stderr and once or
6. This should appear just once on stderr.
```

Si lo ejecutamos de nuevo, pero dirigimos `stderr` a `/dev/null`, vemos lo siguiente, que es el único mensaje escrito en `stdout`:

```
$ python logctx.py 2>/dev/null
5. This should appear twice - once on stderr and once or
```

Una vez más, pero canalizando `stdout` a `/dev/null`, obtenemos:

```
$ python logctx.py >/dev/null
1. This should appear just once on stderr.
3. This should appear once on stderr.
5. This should appear twice - once on stderr and once or
6. This should appear just once on stderr.
```

En este caso, el mensaje #5 impreso en `stdout` no aparece, como se esperaba.

Por supuesto, el enfoque descrito aquí puede generalizarse, por ejemplo, para adjuntar filtros de logging temporalmente. Tenga en cuenta que el código anterior funciona tanto en Python 2 como en Python 3.

Una plantilla de inicio de aplicación CLI

Aquí hay un ejemplo que muestra cómo puede:

- Utilizar un nivel de logging basado en argumentos de la línea de comandos
- Enviar a varios subcomandos en archivos separados, todos registrando en el mismo nivel de forma coherente
- Utilizar una configuración mínima y sencilla

Supongamos que tenemos una aplicación de línea de comandos cuyo trabajo es detener, iniciar o reiniciar algunos servicios. Esto podría organizarse con fines ilustrativos como un archivo `app.py` que es el script principal de la aplicación, con comandos individuales implementados en `start.py`, `stop.py` y `restart.py`. Supongamos además que queremos controlar la verbosidad de la aplicación a través de un argumento de línea de comandos, por defecto en `logging.INFO`. Aquí hay una forma en que se podría escribir `app.py`:

```
import argparse
import importlib
import logging
import os
import sys
```

```
def main(args=None):
    scriptname = os.path.basename(__file__)
    parser = argparse.ArgumentParser(scriptname)
    levels = ('DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRIT
    parser.add_argument('--log-level', default='INFO', c
    subparsers = parser.add_subparsers(dest='command',
                                       help='Available c
    start_cmd = subparsers.add_parser('start', help='Sta
    start_cmd.add_argument('name', metavar='NAME',
                           help='Name of service to star
    stop_cmd = subparsers.add_parser('stop',
                                     help='Stop one or m
    stop_cmd.add_argument('names', metavar='NAME', nargs
                          help='Name of service to stop'
    restart_cmd = subparsers.add_parser('restart',
                                        help='Restart or
    restart_cmd.add_argument('names', metavar='NAME', na
                             help='Name of service to re
    options = parser.parse_args()
    # the code to dispatch commands could all be in this
    # of illustration only, we implement each command in
    try:
        mod = importlib.import_module(options.command)
        cmd = getattr(mod, 'command')
    except (ImportError, AttributeError):
        print('Unable to find the code for command \'%s\
        return 1
    # Could get fancy here and load configuration from f
    logging.basicConfig(level=options.log_level,
                        format='%(levelname)s %(name)s %
    cmd(options)

if __name__ == '__main__':
    sys.exit(main())
```

Y los comandos `start`, `stop` y `reiniciar` se pueden implementar en módulos separados, como para iniciar:

```
# start.py
```

```
import logging

logger = logging.getLogger(__name__)

def command(options):
    logger.debug('About to start %s', options.name)
    # actually do the command processing here ...
    logger.info('Started the \'%s\' service.', options.n
```

y así para detener:

```
# stop.py
import logging

logger = logging.getLogger(__name__)

def command(options):
    n = len(options.names)
    if n == 1:
        plural = ''
        services = '\'%s\'' % options.names[0]
    else:
        plural = 's'
        services = ', '.join('\'%s\'' % name for name in
        i = services.rfind(', ')
        services = services[:i] + ' and ' + services[i +
    logger.debug('About to stop %s', services)
    # actually do the command processing here ...
    logger.info('Stopped the %s service%s.', services, p
```

y de manera similar para reiniciar:

```
# restart.py
import logging

logger = logging.getLogger(__name__)

def command(options):
    n = len(options.names)
    if n == 1:
        plural = ''
        services = '\'%s\'' % options.names[0]
    else:
        plural = 's'
        services = ', '.join('\'%s\'' % name for name ir
        i = services.rfind(', ')
        services = services[:i] + ' and ' + services[i +
    logger.debug('About to restart %s', services)
    # actually do the command processing here ...
    logger.info('Restarted the %s service%s.', services,
```

Si ejecutamos esta aplicación con el nivel de log predeterminado, obtenemos un resultado como este:

```
$ python app.py start foo
INFO start Started the 'foo' service.

$ python app.py stop foo bar
INFO stop Stopped the 'foo' and 'bar' services.

$ python app.py restart foo bar baz
INFO restart Restarted the 'foo', 'bar' and 'baz' servic
```

La primera palabra es el nivel de logging y la segunda palabra es el nombre del módulo o paquete del lugar donde se registró el evento.

Si cambiamos el nivel de logging, podemos cambiar la información enviada al log. Por ejemplo, si queremos más información:

```
$ python app.py --log-level DEBUG start foo
DEBUG start About to start foo
INFO start Started the 'foo' service.

$ python app.py --log-level DEBUG stop foo bar
DEBUG stop About to stop 'foo' and 'bar'
INFO stop Stopped the 'foo' and 'bar' services.

$ python app.py --log-level DEBUG restart foo bar baz
DEBUG restart About to restart 'foo', 'bar' and 'baz'
```

```
INFO restart Restarted the 'foo', 'bar' and 'baz' servic
```

Y si queremos menos:

```
$ python app.py --log-level WARNING start foo
$ python app.py --log-level WARNING stop foo bar
$ python app.py --log-level WARNING restart foo bar baz
```

En este caso, los comandos no imprimen nada en la consola, ya que no registran nada en el nivel de `WARNING` o superior.

Una GUI de Qt para logging

A question that comes up from time to time is about how to log to a GUI application. The Qt [https://www.qt.io/] framework is a popular cross-platform UI framework with Python bindings using PySide2 [https://pypi.org/project/PySide2/] or PyQt5 [https://pypi.org/project/PyQt5/] libraries.

El siguiente ejemplo muestra cómo iniciar sesión en una GUI de Qt. Esto introduce una clase simple `QtHandler` que toma un invocable, que debería ser un slot en el hilo principal que realiza actualizaciones de la GUI. También se crea un hilo de trabajo para mostrar cómo puede iniciar sesión en la GUI tanto desde la propia interfaz de usuario (a través de un botón para el logging manual) como desde un hilo de trabajo que trabaja en segundo plano (aquí, simplemente registrando mensajes en niveles aleatorios con aleatorio breves retrasos intermedios).

El hilo *worker* se implementa usando la clase `QThread` de Qt en lugar del módulo **`threading`**, ya que hay circunstancias en las que uno tiene que usar `QThread`, que ofrece una mejor integración con otros componentes `Qt`.

The code should work with recent releases of any of `PySide6`, `PyQt6`, `PySide2` or `PyQt5`. You should be able to adapt the approach to earlier versions of Qt. Please refer to the comments in the code snippet for more detailed information.

```
import logging
import random
import sys
import time

# Deal with minor differences between different Qt packa
try:
    from PySide6 import QtCore, QtGui, QtWidgets
    Signal = QtCore.Signal
    Slot = QtCore.Slot
except ImportError:
    try:
        from PyQt6 import QtCore, QtGui, QtWidgets
        Signal = QtCore.pyqtSignal
        Slot = QtCore.pyqtSlot
    except ImportError:
        try:
            from PySide2 import QtCore, QtGui, QtWidgets
            Signal = QtCore.Signal
            Slot = QtCore.Slot
        except ImportError:
            from PyQt5 import QtCore, QtGui, QtWidgets
            Signal = QtCore.pyqtSignal
            Slot = QtCore.pyqtSlot

logger = logging.getLogger(__name__)


#
# Signals need to be contained in a QObject or subclass
# initialized.
#
class Signaller(QtCore.QObject):
    signal = Signal(str, logging.LogRecord)

#
# Output to a Qt GUI is only supposed to happen on the m
# handler is designed to take a slot function which is s
# thread. In this example, the function takes a string a
# formatted log message, and the log record which genera
# string is just a convenience - you could format a stri
# you like in the slot function itself.
#
# You specify the slot function to do whatever GUI updat
# doesn't know or care about specific UI elements.
#
class QtHandler(logging.Handler):
    def __init__(self, slotfunc, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.signaller = Signaller()
        self.signaller.signal.connect(slotfunc)

    def emit(self, record):
        s = self.format(record)
        self.signaller.signal.emit(s, record)

#
# This example uses QThreads, which means that the threa
# are named something like "Dummy-1". The function below
# current thread.
#
def ctname():
    return QtCore.QThread.currentThread().objectName()


#
# Used to generate random levels for logging.
#
LEVELS = (logging.DEBUG, logging.INFO, logging.WARNING,
          logging.CRITICAL)

#
# This worker class represents work that is done in a th
# main thread. The way the thread is kicked off to do wo
# that connects to a slot in the worker.
#
# Because the default threadName value in the LogRecord
# a qThreadName which contains the QThread name as compu
# value in an "extra" dictionary which is used to update
# QThread name.
#
# This example worker just outputs messages sequentially
# random delays of the order of a few seconds.
#
class Worker(QtCore.QObject):
    @Slot()
    def start(self):
        extra = {'qThreadName': ctname() }
        logger.debug('Started work', extra=extra)
        i = 1
        # Let the thread run until interrupted. This all
        # thread termination.
        while not QtCore.QThread.currentThread().isInter
            delay = 0.5 + random.random() * 2
            time.sleep(delay)
            try:
                if random.random() < 0.1:
                    raise ValueError('Exception raised:
                else:
                    level = random.choice(LEVELS)
                    logger.log(level, 'Message after del
            except ValueError as e:
                logger.exception('Failed: %s', e, extra=
            i += 1

#
# Implement a simple UI for this cookbook example. This
#
# * A read-only text edit window which holds formatted l
# * A button to start work and log stuff in a separate t
# * A button to log something from the main thread
# * A button to clear the log window
#
class Window(QtWidgets.QWidget):
    COLORS = {
        logging.DEBUG: 'black',
        logging.INFO: 'blue',
        logging.WARNING: 'orange',
        logging.ERROR: 'red',
        logging.CRITICAL: 'purple',
    }

    def __init__(self, app):
        super().__init__()
        self.app = app
        self.textedit = te = QtWidgets.QPlainTextEdit(se
        # Set whatever the default monospace font is for
        f = QtGui.QFont('nosuchfont')
        if hasattr(f, 'Monospace'):
            f.setStyleHint(f.Monospace)
        else:
            f.setStyleHint(f.StyleHint.Monospace)  # for
        te.setFont(f)
        te.setReadOnly(True)
        PB = QtWidgets.QPushButton
        self.work_button = PB('Start background work', s
        self.log_button = PB('Log a message at a random
        self.clear_button = PB('Clear log window', self)
        self.handler = h = QtHandler(self.update_status)
        # Remember to use qThreadName rather than threa
        fs = '%(asctime)s %(qThreadName)-12s %(levelname
        formatter = logging.Formatter(fs)
        h.setFormatter(formatter)
        logger.addHandler(h)
        # Set up to terminate the QThread when we exit
        app.aboutToQuit.connect(self.force_quit)

        # Lay out all the widgets
        layout = QtWidgets.QVBoxLayout(self)
        layout.addWidget(te)
        layout.addWidget(self.work_button)
        layout.addWidget(self.log_button)
        layout.addWidget(self.clear_button)
        self.setFixedSize(900, 400)

        # Connect the non-worker slots and signals
        self.log_button.clicked.connect(self.manual_upda
        self.clear_button.clicked.connect(self.clear_dis

        # Start a new worker thread and connect the slot
        self.start_thread()
        self.work_button.clicked.connect(self.worker.sta
        # Once started, the button should be disabled
        self.work_button.clicked.connect(lambda : self.w

    def start_thread(self):
        self.worker = Worker()
        self.worker_thread = QtCore.QThread()
        self.worker.setObjectName('Worker')
        self.worker_thread.setObjectName('WorkerThread')
        self.worker.moveToThread(self.worker_thread)
        # This will start an event loop in the worker th
        self.worker_thread.start()

    def kill_thread(self):
        # Just tell the worker to stop, then tell it to
        # to happen
        self.worker_thread.requestInterruption()
        if self.worker_thread.isRunning():
            self.worker_thread.quit()
            self.worker_thread.wait()
        else:
            print('worker has already exited.')

    def force_quit(self):
        # For use when the window is closed
        if self.worker_thread.isRunning():
            self.kill_thread()

    # The functions below update the UI and run in the m
    # that's where the slots are set up

    @Slot(str, logging.LogRecord)
    def update_status(self, status, record):
        color = self.COLORS.get(record.levelno, 'black')
        s = '<pre><font color="%s">%s</font></pre>' % (c
        self.textedit.appendHtml(s)

    @Slot()
    def manual_update(self):
        # This function uses the formatted message passe
        # information from the record to format the mess
        # color according to its severity (level).
        level = random.choice(LEVELS)
        extra = {'qThreadName': ctname() }
        logger.log(level, 'Manually logged!', extra=extr

    @Slot()
    def clear_display(self):
        self.textedit.clear()


def main():
    QtCore.QThread.currentThread().setObjectName('MainTh
    logging.getLogger().setLevel(logging.DEBUG)
    app = QtWidgets.QApplication(sys.argv)
    example = Window(app)
    example.show()
    if hasattr(app, 'exec'):
        rc = app.exec()
    else:
        rc = app.exec_()
    sys.exit(rc)

if __name__=='__main__':
    main()
```

Logging en syslog con soporte RFC5424

Although **RFC 5424** [https://datatracker.ietf.org/doc/html/rfc5424.html] dates from 2009, most syslog servers are configured by default to use the older **RFC 3164** [https://datatracker.ietf.org/doc/html/rfc3164.html], which hails from 2001. When `logging` was added to Python in 2003, it supported the earlier (and only existing) protocol at the time. Since RFC 5424 came out, as there has not been widespread deployment of it in syslog servers, the **`SysLogHandler`** functionality has not been updated.

El RFC 5424 contiene algunas características útiles como el soporte para datos estructurados, y si necesitas poder registrar en un servidor syslog con soporte para ello, puedes hacerlo con una subclase *handler* que se parezca a esto:

```
import datetime as dt
import logging.handlers
import re
import socket
import time

class SysLogHandler5424(logging.handlers.SysLogHandler):

    tz_offset = re.compile(r'([+-]\d{2})(\d{2})$')
    escaped = re.compile(r'([\]"\\])')

    def __init__(self, *args, **kwargs):
        self.msgid = kwargs.pop('msgid', None)
        self.appname = kwargs.pop('appname', None)
        super().__init__(*args, **kwargs)

    def format(self, record):
        version = 1
        asctime = dt.datetime.fromtimestamp(record.creat
        m = self.tz_offset.match(time.strftime('%z'))
        has_offset = False
        if m and time.timezone:
            hrs, mins = m.groups()
            if int(hrs) or int(mins):
                has_offset = True
        if not has_offset:
            asctime += 'Z'
        else:
            asctime += f'{hrs}:{mins}'
        try:
            hostname = socket.gethostname()
        except Exception:
            hostname = '-'
        appname = self.appname or '-'
        procid = record.process
        msgid = '-'
        msg = super().format(record)
        sdata = '-'
        if hasattr(record, 'structured_data'):
            sd = record.structured_data
            # This should be a dict where the keys are S
            # dict mapping PARAM-NAME to PARAM-VALUE (re
            # mean)
            # There's no error checking here - it's pure
            # can adapt this code for use in production
            parts = []

            def replacer(m):
                g = m.groups()
                return '\\' + g[0]

            for sdid, dv in sd.items():
                part = f'[{sdid}'
                for k, v in dv.items():
                    s = str(v)
                    s = self.escaped.sub(replacer, s)
                    part += f' {k}="{s}"'
                part += ']'
                parts.append(part)
            sdata = ''.join(parts)
        return f'{version} {asctime} {hostname} {appname
```

Tendrá que estar familiarizado con el RFC 5424 para entender completamente el código anterior, y puede ser que tenga necesidades ligeramente diferentes (por ejemplo, para la forma de pasar los datos estructurales al log). No obstante, lo anterior debería poder adaptarse a sus necesidades específicas. Con el gestor anterior, usted pasaría datos estructurados usando algo como esto:

```
sd = {
    'foo@12345': {'bar': 'baz', 'baz': 'bozz', 'fizz': r
    'foo@54321': {'rab': 'baz', 'zab': 'bozz', 'zzif': r
}
extra = {'structured_data': sd}
i = 1
logger.debug('Message %d', i, extra=extra)
```

Cómo tratar un logger como una salida stream

A veces, necesitas interactuar con una API de terceros que espera un objeto tipo archivo para escribir, pero quieres dirigir la salida de la API a un logger. Puedes hacer esto usando una clase que envuelva un logger con una API tipo archivo. Aquí hay un pequeño script que ilustra tal clase:

```
import logging

class LoggerWriter:
    def __init__(self, logger, level):
        self.logger = logger
        self.level = level

    def write(self, message):
        if message != '\n':  # avoid printing bare newli
            self.logger.log(self.level, message)

    def flush(self):
        # doesn't actually do anything, but might be exp
        # object - so optional depending on your situati
        pass
    def close(self):
        # doesn't actually do anything, but might be exp
        # object - so optional depending on your situati
        # to set a flag so that later calls to write rai
        pass

def main():
    logging.basicConfig(level=logging.DEBUG)
    logger = logging.getLogger('demo')
    info_fp = LoggerWriter(logger, logging.INFO)
    debug_fp = LoggerWriter(logger, logging.DEBUG)
    print('An INFO message', file=info_fp)
    print('A DEBUG message', file=debug_fp)

if __name__ == "__main__":
    main()
```

Cuando se ejecuta este script, se imprime

```
INFO:demo:An INFO message
DEBUG:demo:A DEBUG message
```

También puedes usar `LoggerWriter` para redirigir `sys.stdout` y `sys.stderr` haciendo algo así:

```
import sys

sys.stdout = LoggerWriter(logger, logging.INFO)
sys.stderr = LoggerWriter(logger, logging.WARNING)
```

Debería hacer esto *después* de configurar logging para sus necesidades. En el ejemplo anterior, la llamada **`basicConfig()`** hace esto (utilizando el valor de `sys.stderr` *antes* de que sea sobrescrito por una instancia de `LoggerWriter`). Entonces, obtendrías este tipo de resultado:

```
>>> print('Foo')
INFO:demo:Foo
>>> print('Bar', file=sys.stderr)
WARNING:demo:Bar
```

```
>>>
```

Por supuesto, los ejemplos anteriores muestran la salida según el formato utilizado por **`basicConfig()`**, pero puede utilizar un formateador diferente cuando configure logging.

Ten en cuenta que con el esquema anterior, estás un poco a merced del buffering y de la secuencia de llamadas de escritura que estás interceptando. Por ejemplo, con la definición de `LoggerWriter` anterior, si tienes el fragmento

```
sys.stderr = LoggerWriter(logger, logging.WARNING)
1 / 0
```

entonces la ejecución del script da como resultado

```
WARNING:demo:Traceback (most recent call last):

WARNING:demo:  File "/home/runner/cookbook-loggerwriter/

WARNING:demo:
WARNING:demo:main()
WARNING:demo:  File "/home/runner/cookbook-loggerwriter/

WARNING:demo:
WARNING:demo:1 / 0
WARNING:demo:ZeroDivisionError
WARNING:demo::
WARNING:demo:division by zero
```

As you can see, this output isn't ideal. That's because the underlying code which writes to `sys.stderr` makes multiple writes, each of which results in a separate logged line (for example, the last three lines above). To get around this problem, you need to buffer things and only output log lines when newlines are seen. Let's use a slightly better implementation of `LoggerWriter`:

```
class BufferingLoggerWriter(LoggerWriter):
    def __init__(self, logger, level):
        super().__init__(logger, level)
        self.buffer = ''

    def write(self, message):
        if '\n' not in message:
            self.buffer += message
        else:
            parts = message.split('\n')
            if self.buffer:
                s = self.buffer + parts.pop(0)
                self.logger.log(self.level, s)
            self.buffer = parts.pop()
            for part in parts:
                self.logger.log(self.level, part)
```

Esto sólo almacena cosas hasta que se ve una nueva línea, y luego registra las líneas completas. Con este enfoque, se obtiene una mejor salida:

```
WARNING:demo:Traceback (most recent call last):
WARNING:demo:  File "/home/runner/cookbook-loggerwriter/
WARNING:demo:    main()
WARNING:demo:  File "/home/runner/cookbook-loggerwriter/
WARNING:demo:    1/0
WARNING:demo:ZeroDivisionError: division by zero
```

How to uniformly handle newlines in logging output

Usually, messages that are logged (say to console or file) consist of a single line of text. However, sometimes there is a need to handle messages with multiple lines - whether because a logging format string contains newlines, or logged data contains newlines. If you want to handle such messages uniformly, so that each line in the logged message appears uniformly formatted as if it was logged separately, you can do this using a handler mixin, as in the following snippet:

```
# Assume this is in a module mymixins.py
import copy
```

```
class MultilineMixin:
    def emit(self, record):
        s = record.getMessage()
        if '\n' not in s:
            super().emit(record)
        else:
            lines = s.splitlines()
            rec = copy.copy(record)
            rec.args = None
            for line in lines:
                rec.msg = line
                super().emit(rec)
```

You can use the mixin as in the following script:

```
import logging

from mymixins import MultilineMixin

logger = logging.getLogger(__name__)

class StreamHandler(MultilineMixin, logging.StreamHandle
    pass

if __name__ == '__main__':
    logging.basicConfig(level=logging.DEBUG, format='%(a
                        handlers = [StreamHandler()])
    logger.debug('Single line')
    logger.debug('Multiple lines:\nfool me once ...')
    logger.debug('Another single line')
    logger.debug('Multiple lines:\n%s', 'fool me ...\nca
```

The script, when run, prints something like:

```
2025-07-02 13:54:47,234 DEBUG     Single line
2025-07-02 13:54:47,234 DEBUG     Multiple lines:
2025-07-02 13:54:47,234 DEBUG     fool me once ...
2025-07-02 13:54:47,234 DEBUG     Another single line
```

```
2025-07-02 13:54:47,234 DEBUG     Multiple lines:
2025-07-02 13:54:47,234 DEBUG     fool me ...
2025-07-02 13:54:47,234 DEBUG     can't get fooled agair
```

If, on the other hand, you are concerned about log injection [https://owasp.org/www-community/attacks/Log_Injection], you can use a formatter which escapes newlines, as per the following example:

```
import logging

logger = logging.getLogger(__name__)

class EscapingFormatter(logging.Formatter):
    def format(self, record):
        s = super().format(record)
        return s.replace('\n', r'\n')

if __name__ == '__main__':
    h = logging.StreamHandler()
    h.setFormatter(EscapingFormatter('%(asctime)s %(leve
    logging.basicConfig(level=logging.DEBUG, handlers =
    logger.debug('Single line')
    logger.debug('Multiple lines:\nfool me once ...')
    logger.debug('Another single line')
    logger.debug('Multiple lines:\n%s', 'fool me ...\nca
```

You can, of course, use whatever escaping scheme makes the most sense for you. The script, when run, should produce output like this:

```
2025-07-09 06:47:33,783 DEBUG     Single line
2025-07-09 06:47:33,783 DEBUG     Multiple lines:\nfool
2025-07-09 06:47:33,783 DEBUG     Another single line
2025-07-09 06:47:33,783 DEBUG     Multiple lines:\nfool
```

Escaping behaviour can’t be the stdlib default , as it would break backwards compatibility.

Patrones para evitar

Aunque las secciones anteriores han descrito formas de hacer las cosas que podría necesitar hacer o con las que tratar, vale la pena mencionar algunos patrones de uso que son *inútiles* y que, por lo tanto, deben evitarse en la mayoría de los casos. Las siguientes secciones no están en ningún orden en particular.

Abrir el mismo archivo de registro varias veces

En Windows, por lo general, no podrá abrir el mismo archivo varias veces, ya que esto lanzará un error de «otro proceso está usando el archivo». Sin embargo, en las plataformas POSIX no obtendrá ningún error si abre el mismo archivo varias veces. Esto podría hacerse accidentalmente, por ejemplo:

- Agregar un controlador de archivos más de una vez que hace referencia al mismo archivo (por ejemplo, mediante un error de copiar/pegar/olvidar-cambiar).
- Abrir dos archivos que se ven diferentes, ya que tienen diferentes nombres, pero son iguales porque uno es un enlace simbólico al otro.
- Bifurcar un proceso, después del cual tanto el padre como el hijo tienen una referencia al mismo archivo. Esto podría ser mediante el uso del módulo `multiprocessing`, por ejemplo.

Abrir un archivo varias veces puede *parecer* que funciona la mayor parte del tiempo, pero puede dar lugar a una serie de problemas en la práctica:

- La salida del registro se puede distorsionar porque varios subprocesos o procesos intentan escribir en el mismo archivo. Aunque el registro se protege de el uso simultáneo de la misma instancia de controlador por varios subprocesos, no existe tal protección si dos subprocesos diferentes intentan escrituras simultáneas utilizando dos instancias de controlador diferentes que apunten al mismo archivo.
- Un intento de eliminar un archivo (por ejemplo, durante la rotación de archivos) falla silenciosamente, porque hay otra referencia que apunta a él. Esto puede generar confusión y una pérdida de tiempo de depuración: las entradas del

> registro terminan en lugares inesperados o se pierden por completo. O un archivo que debía ser movido permanece en su lugar, y crece en tamaño inesperadamente a pesar de que la rotación basada en el tamaño está supuestamente en su lugar.

Utilice las técnicas descritas en Logging a un sólo archivo desde múltiples procesos para evitar estos problemas.

Usar registradores como atributos en una clase o pasarlos como parámetros

Si bien puede haber casos inusuales en los que deba hacer esto, en general no tiene sentido porque los registradores son singletons. El código siempre puede acceder a una instancia de registrador dada por su nombre usando `logging.getLogger(name)`, por lo que pasar instancias y mantenerlas como atributos de instancia no tiene sentido. Tenga en cuenta que en otros lenguajes como Java y C#, los registradores suelen ser atributos de clase estáticos. Sin embargo, este patrón no tiene sentido en Python, donde el módulo (y no la clase) es la unidad de descomposición del software.

Adding handlers other than `NullHandler` to a logger in a library

La configuración del registro agregando controladores, formateadores y filtros es responsabilidad del desarrollador de la aplicación, no del desarrollador de la biblioteca. Si mantiene una biblioteca, asegúrese de no agregar controladores a ninguno de sus registradores que no sean una instancia **`NullHandler`**.

Crear muchos registradores (*loggers*)

Los registradores son singletons que nunca se liberan durante la ejecución de un script, por lo que la creación de muchos registradores consumirá memoria que luego no se puede liberar. En lugar de crear un registrador por ej. archivo procesado o conexión de red realizada, use los mecanismos existentes para pasar información contextual a sus registros y restringir los registradores creados a aquellos que describen áreas dentro de su aplicación (generalmente módulos, pero ocasionalmente un poco más detallado que eso).

Otros recursos

Ver también

Módulo `logging`
Referencia de API para el módulo logging.

Módulo `logging.config`
API de configuración para el módulo logging.

Módulo `logging.handlers`
Gestores útiles incluidos con el módulo logging.

Tutorial Básico

Tutorial Avanzado

Regular expression HOWTO

Autor:

A.M. Kuchling <amk@amk.ca>

Resumen

Este documento es un tutorial de introducción al uso de expresiones regulares en Python con el módulo `re`. Proporciona una introducción más apacible que la sección correspondiente en la Referencia de la Biblioteca.

Introducción

Las expresiones regulares (llamadas RE, o regex, o patrones de regex) son esencialmente en un lenguaje de programación diminuto y altamente especializado incrustado dentro de Python y disponible a través del módulo `re`. Usando este pequeño lenguaje, especificas las reglas para el conjunto de cadenas de caracteres posibles que deseas hacer coincidir; este conjunto puede contener frases en inglés, o direcciones de correo electrónico, o comandos TeX, o cualquier cosa que desee. A continuación, puede hacer preguntas como «¿Coincide esta cadena con el patrón?» o «¿Hay alguna coincidencia con el patrón en alguna parte de esta cadena?». También puede utilizar RE para modificar una cadena de caracteres o dividirla de varias formas.

Los patrones de expresiones regulares se compilan en una serie de códigos de bytes que luego son ejecutados por un motor de coincidencia escrito en C. Para un uso avanzado, puede ser necesario prestar mucha atención a cómo el motor ejecutará una RE dado y escribir la RE en un de cierta manera para producir un código de bytes que se ejecute más rápido. La optimización no se trata en este documento, porque requiere que tenga un buen conocimiento de los componentes internos del motor de coincidencia.

El lenguaje de expresiones regulares es relativamente pequeño y restringido, por lo que no todas las posibles tareas de procesamiento de cadenas de caracteres se pueden realizar utilizando expresiones regulares. También hay tareas que *se pueden* hacer con expresiones regulares, pero las expresiones resultan ser muy complicadas. En estos casos, es mejor que escriba código Python para realizar el procesamiento; Si bien el código Python será más lento que una expresión regular elaborada, probablemente también será más comprensible.

Simple patterns

Comenzaremos aprendiendo sobre las expresiones regulares más simples posibles. Dado que las expresiones regulares se utilizan para operar en cadenas de caracteres, comenzaremos con la tarea más común: hacer coincidir caracteres.

Para obtener una explicación detallada de la informática que subyace a las expresiones regulares (autómatas finitos deterministas y no deterministas), puede consultar casi cualquier libro de texto sobre la escritura de compiladores.

Matching characters

La mayoría de letras y caracteres simplemente coincidirán. Por ejemplo, la expresión regular `test` coincidirá exactamente con la cadena `test`. (Puede habilitar un modo que no distinga entre mayúsculas y minúsculas que permitiría que este RE coincida con `test` o `TEST` también; más sobre esto más adelante.)

Hay excepciones a esta regla; algunos caracteres son especiales *metacharacters*, y no coinciden. En cambio, señalan que debe coincidir con algo fuera de lo común, o afectan otras partes de la RE repitiéndolos o cambiando su significado. Gran parte de este documento está dedicado a discutir varios metacaracteres y lo que hacen.

Aquí hay una lista completa de los metacaracteres; sus significados se discutirán en el resto de este COMO (*HOWTO*).

```
. ^ $ * + ? { } [ ] \ | ( )
```

Los primeros metacaracteres que veremos son `[` and `]`. Se utilizan para especificar una clase de carácter, que es un conjunto de caracteres que desea hacer coincidir. Los caracteres se pueden enumerar individualmente, o se puede indicar un rango de caracteres dando dos caracteres y separándolos con un `'-'`. Por ejemplo, `[abc]` coincidirá con cualquiera de los caracteres `a`, `b` o `c`; esto es lo mismo que `[a-c]`, que usa un rango para expresar el mismo conjunto de caracteres. Si quisiera hacer coincidir solo letras minúsculas, su RE sería `[a-c]`.

Los metacaracteres (excepto `\`) no están activos dentro de las clases. Por ejemplo, `[akm$]` coincidirá con cualquiera de los caracteres `'a'`, `'k'`, `'m'`, o `'$'`; `'$'` suele ser un metacarácter, pero dentro de una clase de carácter se le quita su naturaleza especial.

Puede hacer coincidir los caracteres que no figuran en la clase mediante el conjunto *complementing*. Esto se indica mediante la inclusión de un `'^'` como primer carácter de la clase. Por ejemplo, `[^5]` coincidirá con cualquier carácter excepto con `'5'`. Si el símbolo de intercalación aparece en otra parte de una clase de caracter, no tiene un significado especial. Por ejemplo: `[5^]` coincidirá con un `'5'` o un `'^'`.

Quizás el metacarácter más importante es la barra invertida, `\`. Al igual que en los literales de cadena de Python, la barra invertida puede ir seguida de varios caracteres para señalar varias secuencias especiales. También se usa para escapar de todos los metacaracteres, de modo que aún pueda emparejarlos en patrones; por ejemplo, si necesita hacer coincidir un `[` o `\`, puede precederlos con una barra invertida para eliminar su significado especial: `\[` o `\\`.

Algunas de las secuencias especiales que comienzan con `'\'` representan conjuntos predefinidos de caracteres que a menudo son útiles, como el conjunto de dígitos, el conjunto de letras o el conjunto de cualquier cosa que no sea un espacio en blanco.

Tomemos un ejemplo: `\w` coincide con cualquier carácter alfanumérico. Si el patrón de expresiones regulares se expresa en bytes, esto es equivalente a la clase `[a-zA-Z0-9_]`. Si el patrón de expresiones regulares es una cadena de caracteres, `\w` coincidirá con todos los caracteres marcados como letras en la base de datos Unicode proporcionada por el módulo **`unicodedata`**. Puede usar la definición más restringida de `\w` en un patrón de cadena proporcionando el indicador **`re.ASCII`** al compilar la expresión regular.

La siguiente lista de secuencias especiales no está completa. Para obtener una lista completa de secuencias y definiciones de clases expandidas para patrones de cadenas Unicode, consulte la última parte de Regular Expression Syntax en la referencia de la biblioteca estándar. En general, las versiones Unicode coinciden con cualquier carácter que esté en la categoría apropiada en la base de datos Unicode.

`\d`

Coincide con cualquier dígito decimal; esto es equivalente a la clase `[0-9]`.

`\D`

Coincide con cualquier carácter que no sea un dígito; esto es equivalente a la clase `[^0-9]`.

`\s`

Coincide con cualquier carácter de espacio en blanco; esto es equivalente a la clase `[ \t\n\r\f\v]`.

`\S`

Coincide con cualquier carácter que no sea un espacio en blanco; esto es equivalente a la clase `[^ \t\n\r\f\v]`.

`\w`

Coincide con cualquier carácter alfanumérico; esto es equivalente a la clase `[a-zA-Z0-9_]`.

`\W`

Coincide con cualquier carácter no alfanumérico; esto es equivalente a la clase `[^a-zA-Z0-9_]`.

Estas secuencias se pueden incluir dentro de una clase de carácter. Por ejemplo, `[\s,.]` es una clase de carácter que coincidirá con cualquier carácter de espacio en blanco, o `','` o `'.'`.

El metacarácter final en esta sección es `.`. Coincide con cualquier cosa excepto un carácter de nueva línea, y hay un modo alternativo (**`re.DOTALL`**) donde coincidirá incluso con una nueva línea. `.` se usa a menudo cuando se desea hacer coincidir «cualquier carácter».

Repeating things

Ser capaz de hacer coincidir diferentes conjuntos de caracteres es lo primero que pueden hacer las expresiones regulares que ya no es posible con los métodos disponibles en cadenas de caracteres. Sin embargo, si esa fuera la única capacidad adicional de las expresiones regulares, no serían un gran avance. Otra capacidad es que puede especificar que partes de la RE deben repetirse un cierto número de veces.

El primer metacarácter para repetir cosas que veremos es `*`. `*` no coincide con el carácter literal `'*'`; en cambio, especifica que el carácter anterior puede coincidir cero o más veces, en lugar de exactamente una vez.

Por ejemplo, `ca*t` coincidirá con `'ct'` (0 `'a'` caracteres), `'cat'` (1 `'a'`), `'caaat'` (3 `'a'` caracteres), etc.

Las repeticiones como `*` son *greedy*; al repetir una RE, el motor de emparejamiento intentará repetirlo tantas veces como sea posible. Si las partes posteriores del patrón no coinciden, el motor de coincidencia hará una copia de seguridad y volverá a intentarlo con menos repeticiones.

Un ejemplo paso a paso hará que esto sea más obvio. Consideremos la expresión `a[bcd]*b`. Esto coincide con la letra `'a'`, cero o más letras de la clase `[bcd]`, y finalmente termina con una `'b'`. Ahora imagina hacer coincidir este RE con la cadena de caracteres `'abcbd'`.

Paso	Coincidencias	Explicación
1	`a`	La `a` en las RE coincide.

2	`abcbd`	El motor coincide con `[bcd]*`, yendo tan lejos como puede, que es hasta el final de la cadena de caracteres.
3	*Failure*	El motor intenta hacer coincidir `b`, pero la posición actual está al final de la cadena de caracteres, por lo que falla.
4	`abcb`	Hace una copia de seguridad para que `[bcd]*` coincida con un carácter menos.
5	*Failure*	Intente `b` de nuevo, pero la posición actual está en el último caracter, que es un `'d'`.
6	`abc`	Haga una copia de seguridad de nuevo, de modo que `[bcd]*` solo coincida con `bc`.
7	`abcb`	Intente `b` de nuevo. Esta vez, el carácter en la posición actual es `'b'`, por lo que tiene éxito.

Se ha alcanzado el final de la RE y ha coincidido con `'abcb'`. Esto demuestra cómo el motor de coincidencias llega tan lejos como puede al principio, y si no se encuentra ninguna coincidencia, retrocederá progresivamente y volverá a intentar de la RE una y otra vez. Hará una copia de seguridad hasta que haya probado cero coincidencias para `[bcd]*`, y si eso falla posteriormente, el motor concluirá que la cadena no coincide con la RE en absoluto.

Otro metacarácter que se repite es `+`, que coincide una o más veces. Preste especial atención a la diferencia entre `*` and `+`; coincide con *cero* o más veces, por lo que cualquier cosa que se repita puede no estar presente en absoluto, mientras que `+` requiere al menos *one* aparición. Para usar un ejemplo similar, `'cat'` (1 `'a'`), `'caaat'` (3 `'a'`s), pero no coincidirá con `'ct'`.

Hay otros dos operadores de repetición o cuantificadores. El carácter de interrogación, `?`, coincide una vez o cero veces; puede pensar en ello como marcar algo como opcional. Por ejemplo, `home-?brew` coincide con `'homebrew'` o `'home-brew'`.

El cuantificador más complicado es `{m,n}`, donde *m* y *n* son enteros decimales. Este cuantificador significa que debe haber al menos *m* repeticiones y como máximo *n*. Por ejemplo, `a/{1,3}b` coincidirá con `'a/b'`, `'a//b'`, y `'a///b'`. No coincidirá con `'ab'`, que no tiene barras diagonales, ni con `'a////b'`, que tiene cuatro.

Puede omitir *m* o *n*; en ese caso, se asume un valor razonable para el valor faltante. Omitir *m* se interpreta como un límite inferior de 0, mientras que omitir *n* da como resultado un límite superior de infinito.

The simplest case `{m}` matches the preceding item exactly *m* times. For example, `a/{2}b` will only match `'a//b'`.

Los lectores con una inclinación reduccionista pueden notar que los otros tres cuantificadores se pueden expresar todos utilizando esta notación. `{0,}` es lo mismo que `*`, `{1,}` es equivalente a `+`, y `{0,1}` es lo mismo que `?`. Es mejor usar `*`, `+`, o `?` cuando sea posible, simplemente porque son más cortos y más fáciles de leer.

Using regular expressions

Ahora que hemos visto algunas expresiones regulares simples, ¿cómo las usamos realmente en Python? El módulo **`re`** proporciona una interfaz para el motor de expresiones regulares, lo que le permite compilar RE en objetos y luego realizar coincidencias con ellos.

Compiling regular expressions

Las expresiones regulares se compilan en objetos de patrón, que tienen métodos para diversas operaciones, como buscar coincidencias de patrones o realizar sustituciones de cadenas de caracteres.

```
>>> import re
>>> p = re.compile('ab*')
>>> p
re.compile('ab*')
```

`re.compile()` también acepta un argumento opcional *flags*, usado para habilitar varias características especiales y variaciones de sintaxis. Repasaremos las configuraciones disponibles más adelante, pero por ahora un solo ejemplo servirá:

```
>>> p = re.compile('ab*', re.IGNORECASE)
```

La RE se pasa a `re.compile()` como una cadena de caracteres. Las RE se manejan como cadenas de caracteres porque las expresiones regulares no son parte del lenguaje central de Python y no se creó una sintaxis especial para expresarlas. (Hay aplicaciones que no necesitan RE en absoluto, por lo que no hay necesidad de aumentar la especificación del lenguaje incluyéndolas). En cambio, el módulo `re` es simplemente un módulo de extensión C incluido en Python, al igual que los módulos `socket` o `zlib`.

Poner RE en cadenas de caracteres mantiene el lenguaje Python más simple, pero tiene una desventaja que es el tema de la siguiente sección.

The backslash plague

Como se indicó anteriormente, las expresiones regulares usan el carácter de barra invertida (`'\'`) para indicar formas especiales o para permitir que se usen caracteres especiales sin invocar su significado especial. Esto entra en conflicto con el uso de Python del mismo carácter para el mismo propósito en cadenas literales.

Supongamos que desea escribir una RE que coincida con la cadena de caracteres `\section`, que podría encontrarse en un archivo LaTeX. Para averiguar qué escribir en el código del programa, comience con la cadena deseada para que coincida. A continuación, debe escapar de las barras invertidas y otros metacaracteres precediéndolos con una barra invertida, lo que da como resultado la cadena `\\section`. La cadena resultante que debe pasarse a `re.compile()` debe ser `\\section`. Sin embargo, para expresar esto como una cadena literal de Python, ambas barras invertidas deben escaparse *nuevamente*.

Caracteres	**Explicación**
`\section`	Cadena de texto que debe coincidir
`\\section`	Barra invertida de escape para `re.compile()`
`"\\\\section"`	Barra invertida de escape para un literal de cadena de caracteres

En resumen, para hacer coincidir una barra invertida literal, uno tiene que escribir `'\\\\'` como la cadena RE, porque la expresión regular debe ser `\\`, y cada barra invertida debe expresarse como `\\` dentro de un literal de cadena Python normal. En las RE que presentan barras invertidas repetidamente, esto genera muchas barras invertidas repetidas y dificulta la comprensión de las cadenas resultantes.

La solución es utilizar la notación de cadena de caracteres sin formato de Python para expresiones regulares; las barras invertidas no se manejan de ninguna manera especial en una cadena literal con el prefijo `'r'`, por lo que `r"\n"` es una cadena de dos caracteres que contiene `'\'` y `'n'`, mientras que `"\n"` es una cadena de un carácter que contiene una nueva línea. Las expresiones regulares a menudo se escribirán en código Python utilizando esta notación de cadena sin formato.

In addition, special escape sequences that are valid in regular expressions, but not valid as Python string literals, now result in a **`SyntaxWarning`** and will eventually become a **`SyntaxError`**, which means the sequences will be invalid if raw string notation or escaping the backslashes isn’t used.

Cadena de caracteres regular	Cadena de caracteres crudas (*Raw string*)
`"ab*"`	`r"ab*"`
`"\\\\section"`	`r"\\section"`
`"\\w+\\s+\\1"`	`r"\w+\s+\1"`

Performing matches

Una vez que tiene un objeto que representa una expresión regular compilada, ¿qué hace con él? Los objetos de patrón tienen varios métodos y atributos. Aquí solo se cubrirán los más importantes; consulte los documentos **`re`** para obtener una lista completa.

Método/atributo	Objetivo
`match()`	Determina si la RE coincide con el comienzo de la cadena de caracteres.
`search()`	Escanea una cadena, buscando cualquier ubicación donde coincida este RE.
`findall()`	Find all substrings where the RE matches, and return them as a list.
`finditer()`	Find all substrings where the RE matches, and return them as an iterator.

`match()` y **`search()`** retornan `None` si la coincidencia no puede ser encontrada. Si tienen éxito, se retorna una instancia match object, que contiene información sobre la coincidencia: dónde comienza y termina, la subcadena de caracteres con la que coincidió, y más.

Puedes aprender sobre esto experimentando de forma interactiva con el módulo **`re`**.

Este CÓMO (*HOWTO*) utiliza el intérprete estándar de Python para sus ejemplos. Primero, ejecute el intérprete de Python, importe el módulo **`re`** y compile en RE:

```
>>> import re
>>> p = re.compile('[a-z]+')
>>> p
re.compile('[a-z]+')
```

Ahora, puede intentar hacer coincidir varias cadenas con la RE `[a-z]+`. Una cadena de caracteres vacía no debería coincidir en absoluto, ya que `+` significa que "una o más repeticiones". **`match()`** debería retornar `None` en este caso, lo que hará que el intérprete no imprima ningún resultado. Puede imprimir explícitamente el resultado de **`match()`** para aclarar esto.

```
>>> p.match("")
>>> print(p.match(""))
None
```

Ahora, intentémoslo en una cadena de caracteres que debería coincidir, como `tempo`.En este caso , **`match()`** retornará un match object, por lo que debe almacenar el resultado en una variable para su posterior uso.

```
>>> m = p.match('tempo')
>>> m
<re.Match object; span=(0, 5), match='tempo'>
```

Ahora puede consultar match object para obtener información sobre la cadena coincidente. Las instancias de objetos coincidentes también tienen varios métodos y atributos; los más importantes son:

Método/atributo	Objetivo
`group()`	Retorna la cadena de caracteres que coincide con la RE
`start()`	Retorna la posición de inicio de la coincidencia
`end()`	Retorna la posición final de la coincidencia
`span()`	Retorna una tupla que contiene (inicio, final) las posiciones de coincidencia

Probando estos métodos pronto aclarará sus significados:

```
>>> m.group()
'tempo'
>>> m.start(), m.end()
(0, 5)
>>> m.span()
(0, 5)
```

`group()` retorna la subcadena de caracteres que coincide con la RE. **`start()`** y **`end()`** retornan el índice inicial y final de la coincidencia. **`span()`** retorna el índice inicial y final en una única tupla. Dado que el método **`match()`** solo verifica si la RE coincide al comienzo de una cadena de caracteres, **`start()`** siempre será cero. Sin embargo, el método de patrones **`search()`** escanea a través de la cadena de caracteres, por lo que es posible que la coincidencia no comience en cero en ese caso.

```
>>> print(p.match('::: message'))
None
>>> m = p.search('::: message'); print(m)
<re.Match object; span=(4, 11), match='message'>
>>> m.group()
'message'
>>> m.span()
(4, 11)
```

En programas reales, el estilo más común es almacenar match object en una variable, y luego verificar si era `None`. Esto generalmente se ve así:

```
p = re.compile( ... )
m = p.match( 'string goes here' )
```

```
if m:
    print('Match found: ', m.group())
else:
    print('No match')
```

Dos métodos de patrón retornan todas las coincidencias de un patrón. **`findall()`** retorna una lista de cadenas de caracteres coincidentes:

```
>>> p = re.compile(r'\d+')
>>> p.findall('12 drummers drumming, 11 pipers piping, 1
['12', '11', '10']
```

The `r` prefix, making the literal a raw string literal, is needed in this example because escape sequences in a normal «cooked» string literal that are not recognized by Python, as opposed to regular expressions, now result in a **`SyntaxWarning`** and will eventually become a **`SyntaxError`**. See The backslash plague.

`findall()` tiene que crear la lista completa antes de que pueda retornarse como resultado. El método **`finditer()`** retorna una secuencia de match object instancias como iterados iterator:

```
>>> iterator = p.finditer('12 drummers drumming, 11 ...
>>> iterator
<callable_iterator object at 0x...>
>>> for match in iterator:
...     print(match.span())
...
(0, 2)
(22, 24)
(29, 31)
```

Module-level functions

No es necesario crear un objeto patrón y llamar a sus métodos; el módulo **`re`** también proporciona funciones de nivel superior llamadas **`match()`** , **`search()`** , **`findall()`** , **`sub()`**, y así sucesivamente. Estas funciones toman los mismos argumentos que el método de patrón correspondiente con la cadena de RE agregada como primer argumento, y aún así retornan una instancia de `None` o match object.

```
>>> print(re.match(r'From\s+', 'Fromage amk'))
None
>>> re.match(r'From\s+', 'From amk Thu May 14 19:12:10 1
<re.Match object; span=(0, 5), match='From '>
```

Bajo el capó (*hood*), estas funciones simplemente crean un objeto patrón para usted y llaman al método apropiado en él. También almacenan el objeto compilado en un caché, por lo que las futuras llamadas que usen el mismo RE no necesitarán analizar el patrón una y otra vez.

¿Debería utilizar estas funciones a nivel de módulo o debería obtener el patrón y llamar a sus métodos usted mismo? Si está accediendo a una expresión regular dentro de un bucle, la compilación previa guardará algunas llamadas a funciones. Fuera de los bucles, no hay mucha diferencia gracias al caché interno.

Compilation flags

Las flags de compilación le permiten modificar algunos aspectos de cómo funcionan las expresiones regulares. Las flags están disponibles en el módulo **`re`** con dos nombres, un nombre largo como **`IGNORECASE`** y una forma corta de una letra como **`I`**. (Si está familiarizado con los modificadores de patrones de Perl, las formas de una letra usan las mismas letras; la forma corta de **`re.VERBOSE`** es **`re.X`**, por ejemplo).Se pueden especificar varios indicadores uniéndolos con *OR* bit a bit; `re.I | re.M` establece los flags **`I`** and **`M`**, por ejemplo.

Aquí hay una tabla de las flags disponibles, seguida de una explicación más detallada de cada una.

Flag	Significado
`ASCII`, **`A`**	Hace que varios escapes como `\w`, `\b`, `\s` y `\d` coincidan solo en caracteres ASCII con la propiedad respectiva.
`DOTALL`, **`S`**	Hace que `.` coincida con cualquier caracter, incluidas las nuevas líneas.
`IGNORECASE`, **`I`**	Hace coincidencias que no distingan entre mayúsculas y

	minúsculas.
LOCALE, **L**	Hace una coincidencia con reconocimiento de configuración regional.
MULTILINE, **M**	Coincidencia de varias líneas, que afecta a `^` y `$`.
VERBOSE, **X** (for “extended”)	Habilite REs detallados, que se pueden organizar de manera más limpia y comprensible.

re.I
re.IGNORECASE

Realiza una coincidencia que no distinga entre mayúsculas y minúsculas; la clase de caracteres y las cadenas de caracteres literales coincidirán con las letras ignorando las mayúsculas. Por ejemplo, `[A-Z]` también coincidirá con letras minúsculas. La coincidencia completa de Unicode también funciona a menos que se utilice la flag **ASCII** para deshabilitar las coincidencias que no sean ASCII. Cuando los patrones Unicode `[a-z]` o `[A-Z]` se utilizan en combinación con el indicador **IGNORECASE**, coincidirán con las 52 letras ASCII y 4 letras adicionales no ASCII “İ” (U+0130, letra mayúscula latina I con un punto arriba), “ı” (U+0131, letra minúscula latina sin punto i),”ſ” (U+017F, letra minúscula latina larga s) y “K” (U+212A, signo de Kelvin). `Spam` coincidirá `'Spam'`, `'spam'`, `'spAM'`, o `'ſpam'` (este último solo coincide en modo Unicode). Estas minúsculas no tiene en cuenta la configuración regional actual; lo hará si también establece la flag **LOCALE**.

re.L
re.LOCALE

Hace que `\w`, `\W`, `\b`, `\B` y la coincidencia que no distinga entre mayúsculas y minúsculas dependan de la configuración regional actual en lugar de la base de datos Unicode.

Las configuraciones regionales son una característica de la biblioteca C destinada a ayudar a escribir programas que tengan en cuenta las diferencias de idioma. Por ejemplo, si está procesando texto en francés codificado, querrá poder escribir `\w+` para que coincida con las palabras, pero `\w` solo coincide con la clase de caracteres `[A-Za-z]` en patrones de bytes; no coincidirá con los bytes correspondientes a `é` or `ç`. Si su sistema está configurado correctamente y se selecciona una configuración regional francesa, ciertas funciones de C le dirán al programa que el byte correspondiente a `é` también debe considerarse una letra. Establecer el indicador **`LOCALE`** al compilar una expresión regular hará que el objeto compilado resultante use estas funciones C para `\w`; esto es más lento, pero también permite que `\w+` coincida con las palabras en francés como era de esperar. Se desaconseja el uso de esta flag en Python 3 ya que el mecanismo de configuración regional es muy poco confiable, solo maneja una «cultura» a la vez y solo funciona con configuraciones regionales de 8 bits. La coincidencia Unicode ya está habilitada de forma predeterminada en Python 3 para patrones Unicode (str), y puede manejar diferentes configuraciones regionales/idiomas.

re.M
re.MULTILINE

(`^` y `$` aún no se han explicado; se presentarán en la sección More metacharacters.)

Por lo general, `^` coincide solo al principio de la cadena de caracteres, y `$` solo coincide con el final de la cadena de caracteres e inmediatamente antes del salto de línea (si existe) al final de la cadena de caracteres. Cuando se especifica esta bandera, `^` coincide al principio de la cadena y al comienzo de cada línea dentro de la cadena, inmediatamente después de cada nueva línea. De manera similar, el metacarácter `$` coincide al final de la cadena de caracteres y al final de cada línea (inmediatamente antes de cada nueva línea).

re.S
re.DOTALL

Hace que el carácter especial `'.'` coincida con cualquier carácter, incluida una nueva línea; sin esta bandera, `'.'` coincidirá con cualquier cosa *except* una nueva línea.

re.A

re.ASCII

Haga que `\w`, `\W`, `\b`, `\B`, `\s` y `\S` realicen una coincidencia solo en ASCII en lugar de una coincidencia Unicode completa. Esto solo es significativo para los patrones Unicode y se ignora para los patrones de bytes.

re.X
re.VERBOSE

This flag allows you to write regular expressions that are more readable by granting you more flexibility in how you can format them. When this flag has been specified, whitespace within the RE string is ignored, except when the whitespace is in a character class or preceded by an unescaped backslash; this lets you organize and indent the RE more clearly. This flag also lets you put comments within a RE that will be ignored by the engine; comments are marked by a `'#'` that's neither in a character class nor preceded by an unescaped backslash.

Por ejemplo, aquí hay una RE que usa **`re.VERBOSE`**; ¿Ves lo fácil que es leer?

```
charref = re.compile(r"""
 &[#]                # Start of a numeric entity re
 (
     0[0-7]+         # Octal form
   | [0-9]+          # Decimal form
   | x[0-9a-fA-F]+   # Hexadecimal form
 )
 ;                   # Trailing semicolon
""", re.VERBOSE)
```

Sin la configuración detallada, la RE se vería así:

```
charref = re.compile("&#(0[0-7]+"
                     "|[0-9]+"
                     "|x[0-9a-fA-F]+);")
```

En el ejemplo anterior, la concatenación automática de cadenas de caracteres literales de Python se ha utilizado para dividir la RE en partes más pequeñas, pero aún es más difícil de entender que la versión que usa **`re.VERBOSE`**.

More pattern power

Hasta ahora solo hemos cubierto una parte de las características de las expresiones regulares. En esta sección, cubriremos algunos metacaracteres nuevos y cómo usar grupos para recuperar partes del texto que coincidió.

More metacharacters

Hay algunos metacaracteres que aún no hemos cubierto. La mayoría de ellos se tratarán en esta sección.

Algunos de los metacaracteres restantes que se discutirán son *zero-width assertions*. No hacen que el motor avance a través de la cadena de caracteres; en cambio, no consumen caracteres en absoluto y simplemente tienen éxito o fracasan. Por ejemplo, `\b` es una flag de que la posición actual se encuentra en el límite de una palabra; la posición no cambia por la `\b` en absoluto. Esto significa que las aserciones de ancho cero nunca deben repetirse, porque si coinciden una vez en una ubicación determinada, obviamente pueden coincidir un número infinito de veces.

`|`

Alternancia, o el operador «or». Si *A* y *B* son expresiones regulares, `A|B` coincidirá con cualquier cadena de caracteres que coincida con *A* o *B*. `|` tiene una precedencia muy baja para que funcione razonablemente cuando está alternando cadenas de varios caracteres. `Crow|Servo` coincidirá con `'Crow'` o `'Servo'`, no `'Cro'`, un `'w'` o un `'S'`, y `'ervo'`.

Para hacer coincidir un literal `'|'`, use `\|`, o enciérrelo dentro de una clase de carácter, como en `[|]`.

`^`

Coincide con el comienzo de las líneas. A menos que se haya establecido la flag **`MULTILINE`** , esto solo coincidirá al principio de la cadena de caracteres. En modo **`MULTILINE`** , esto también coincide inmediatamente después de cada nueva línea dentro de la cadena.

Por ejemplo, si desea hacer coincidir la palabra `From` solo al principio de una línea, la RE que debe usar es `^From`.

```
>>> print(re.search('^From', 'From Here to Eternity
<re.Match object; span=(0, 4), match='From'>
>>> print(re.search('^From', 'Reciting From Memory'
None
```

Para una coincidencia literal `'^'`, usar `\^`.

`$`

Coincide con el final de una línea, que se define como el final de la cadena o cualquier ubicación seguida de un carácter de nueva línea.

```
>>> print(re.search('}$', '{block}'))
<re.Match object; span=(6, 7), match='}'>
>>> print(re.search('}$', '{block} '))
None
>>> print(re.search('}$', '{block}\n'))
<re.Match object; span=(6, 7), match='}'>
```

Para hacer coincidir un literal `'$'`, usar `\$` o enciérrelo dentro de una clase de carácter, como en `[$]`.

`\A`

Coincide solo al comienzo de la cadena de caracteres. Cuando no está en el modo **`MULTILINE`**,``A`` y `^` son efectivamente lo mismo. En el modo **`MULTILINE`**, son diferentes: `\A` todavía coincide solo al principio de la cadena, pero `^` puede coincidir en cualquier ubicación dentro de la cadena de caracteres que sigue a un carácter de nueva línea.

`\z`

Coincidencias solo al final de la cadena de caracteres.

`\Z`

The same as `\z`. For compatibility with old Python versions.

`\b`

Límite de palabra. Esta es una aserción de ancho cero que coincide solo al principio o al final de una palabra. Una palabra se define como una secuencia de caracteres alfanuméricos, por lo que el final de una palabra se indica mediante un espacio en blanco o un carácter no alfanumérico.

El siguiente ejemplo coincide con `class` solo cuando es una palabra completa; no coincidirá cuando esté contenido dentro de otra palabra.

```
>>> p = re.compile(r'\bclass\b')
>>> print(p.search('no class at all'))
<re.Match object; span=(3, 8), match='class'>
>>> print(p.search('the declassified algorithm'))
None
>>> print(p.search('one subclass is'))
None
```

Hay dos sutilezas que debe recordar al usar esta secuencia especial. Primero, esta es la peor colisión entre las cadenas literales de caracteres de Python y las secuencias de expresiones regulares. En las cadenas de caracteres literales de Python, `\b` es el carácter de retroceso (*backspace*), valor ASCII 8. Si no está utilizando cadenas de caracteres sin procesar, Python convertirá la `\b` en una linea de retroceso, y su RE no lo hará coincidir como lo espera. El siguiente ejemplo tiene el mismo aspecto que nuestra RE anterior, pero omite la `'r'` delante de la cadena de caracteres de RE.

```
>>> p = re.compile('\bclass\b')
>>> print(p.search('no class at all'))
None
>>> print(p.search('\b' + 'class' + '\b'))
<re.Match object; span=(0, 7), match='\x08class\x08
```

En segundo lugar, dentro de una clase de caracteres, donde no hay uso para esta aserción, `\b` representa el carácter de retroceso, por compatibilidad con las cadenas de caracteres literales de Python.

`\B`

Otra flag de ancho cero, esto es lo opuesto a `\b`, solo coincide cuando la posición actual no está en el límite de una palabra.

Agrupando

Con frecuencia, necesita obtener más información que solo si la RE coincide o no. Las expresiones regulares se utilizan a menudo para diseccionar cadenas de caracteres escribiendo una RE dividido en varios subgrupos que coinciden con diferentes componentes de interés. Por ejemplo, una línea de encabezado RFC-822 se divide en un nombre de encabezado y un valor, separados por un `':'`, así:

```
From: author@example.com
User-Agent: Thunderbird 1.5.0.9 (X11/20061227)
MIME-Version: 1.0
To: editor@example.com
```

Esto se puede manejar escribiendo una expresión regular que coincida con una línea de encabezado completa y que tenga un grupo que coincida con el nombre del encabezado y otro grupo que coincida con el valor del encabezado.

Los grupos se marcan con los metacaracteres `'('` y `')'`. `'('` y `')'` tienen prácticamente el mismo significado que en las expresiones matemáticas; agrupan las expresiones contenidas dentro de ellos, y puedes repetir el contenido de un grupo con un cuantificador, como `*`, `+`, `?`, o `{m,n}`. Por ejemplo, `(ab)*` coincidirá con cero o más repeticiones de `ab`.

```
>>> p = re.compile('(ab)*')
>>> print(p.match('ababababab').span())
(0, 10)
```

Los grupos indicados con `'('`, `')'` también capturan el índice inicial y final del texto que coinciden; esto se puede recuperar pasando un argumento a **`group()`**, **`start()`**, **`end()`**, y **`span()`**. Los grupos se numeran empezando por 0. El grupo 0 siempre está presente; es todo las RE, entonces todos los métodos match object tienen el grupo 0 como argumento predeterminado. Más adelante veremos cómo expresar grupos que no capturan el espacio de texto que coinciden.

```
>>> p = re.compile('(a)b')
>>> m = p.match('ab')
>>> m.group()
'ab'
>>> m.group(0)
'ab'
```

Los subgrupos están numerados de izquierda a derecha, de 1 en adelante. Los grupos se pueden anidar; para determinar el número, simplemente cuente los caracteres del paréntesis de apertura, de izquierda a derecha.

```
>>> p = re.compile('(a(b)c)d')
>>> m = p.match('abcd')
>>> m.group(0)
'abcd'
>>> m.group(1)
'abc'
>>> m.group(2)
'b'
```

`group()` se pueden pasar varios números de grupo a la vez, en cuyo caso retornará una tupla que contiene los valores correspondientes para esos grupos.

```
>>> m.group(2,1,2)
('b', 'abc', 'b')
```

El método **`groups()`** retorna una tupla que contiene las cadenas de caracteres de todos los subgrupos, desde 1 hasta la cantidad que haya.

```
>>> m.groups()
('abc', 'b')
```

Las referencias inversas en un patrón le permiten especificar que el contenido de un grupo de captura anterior también debe encontrarse en la ubicación actual en la cadena. Por ejemplo,``1`` tendrá éxito si el contenido exacto del grupo 1 se puede encontrar en la posición actual y falla en caso contrario. Recuerde que las cadenas de caracteres literales de Python también usan una barra invertida seguida de números para permitir la inclusión de caracteres arbitrarios en una cadena de caracteres, así que asegúrese de usar una cadena de caracteres sin procesar al incorporar referencias inversas en una RE.

Por ejemplo, la siguiente RE detecta palabras duplicadas en una cadena.

```
>>> p = re.compile(r'\b(\w+)\s+\1\b')
>>> p.search('Paris in the the spring').group()
'the the'
```

Las referencias inversas como esta no suelen ser útiles para buscar a través de una cadena — hay pocos formatos de texto que repiten datos de esta manera — pero pronto descubrirá que son *muy* útiles al realizar sustituciones de cadenas de caracteres.

Non-capturing and named groups

Las RE elaboradas pueden utilizar muchos grupos, tanto para capturar subcadenas de caracteres de interés como para agrupar y estructurar la propia RE. En las RE complejas, resulta difícil realizar un seguimiento de los números de los grupos. Hay dos funciones que ayudan con este problema. Ambos usan una sintaxis común para las extensiones de expresiones regulares, así que veremos eso primero.

Perl 5 es bien conocido por sus poderosas adiciones a las expresiones regulares estándar. Para estas nuevas características, los desarrolladores de Perl no podían elegir nuevos metacaracteres de una sola pulsación de tecla o nuevas secuencias especiales que comienzan con `\` sin hacer que las expresiones regulares de Perl sean confusamente diferentes de las RE estándar. Si eligieran `&` como un nuevo metacarácter, por ejemplo, las expresiones antiguas supondrían que `&` era un carácter regular y no se habría escapado escribiendo `\&` o `[&]`.

La solución elegida por los desarrolladores de Perl fue utilizar `(?...)` como sintaxis de extensión. `?` inmediatamente después de un paréntesis había un error de sintaxis porque el `?` no tendría nada que repetir, por lo que esto no introdujo ningún problema de compatibilidad. Los caracteres inmediatamente después de `?` Indican qué extensión se está utilizando, por lo que `(?=foo)` es una cosa (una flag de anticipación positiva) y `(?:foo)` es otra cosa (un grupo de no captura que contiene la subexpresión `foo`).

Python admite varias de las extensiones de Perl y agrega una sintaxis de extensión a la sintaxis de extensión de Perl. Si el primer carácter después del signo de interrogación es una `P`, sabrá que es una extensión específica de Python.

Ahora que hemos visto la sintaxis de la extensión general, podemos volver a las características que simplifican el trabajo con grupos en RE complejos.

A veces querrá usar un grupo para denotar una parte de una expresión regular, pero no está interesado en recuperar el contenido del grupo. Puede hacer que este hecho sea explícito utilizando un grupo de no captura: `(?:...)`, donde puede reemplazar el `...` con cualquier otra expresión regular.

```
>>> m = re.match("([abc])+", "abc")
>>> m.groups()
('c',)
>>> m = re.match("(?:[abc])+", "abc")
>>> m.groups()
()
```

Excepto por el hecho de que no puede recuperar el contenido de lo que coincide con el grupo, un grupo que no captura se comporta exactamente igual que un grupo que captura; puede poner cualquier cosa dentro de él, repetirlo con un metacarácter de repetición como

`*` y anidarlo dentro de otros grupos (capturando o no capturando). `(?:...)` es particularmente útil cuando se modifica un patrón existente, ya que puede agregar nuevos grupos sin cambiar cómo se numeran todos los demás grupos. Cabe mencionar que no hay diferencia de rendimiento en la búsqueda entre grupos de captura y no captura; ninguna forma es más rápida que la otra.

Una característica más significativa son los grupos nombrados: en lugar de referirse a ellos por números, los grupos pueden ser referenciados por un nombre.

La sintaxis de un grupo con nombre es una de las extensiones específicas de Python: `(?P<name>...)`. *name* es, obviamente, el nombre del grupo. Los grupos con nombre se comportan exactamente como los grupos de captura y, además, asocian un nombre con un grupo. Los métodos match object que tratan con la captura de grupos aceptan enteros que se refieren al grupo por número o cadenas de caracteres que contienen el nombre del grupo deseado. Los grupos con nombre todavía reciben números, por lo que puede recuperar información sobre un grupo de dos maneras:

```
>>> p = re.compile(r'(?P<word>\b\w+\b)')
>>> m = p.search( '(((( Lots of punctuation )))' )
>>> m.group('word')
'Lots'
>>> m.group(1)
'Lots'
```

Además, puede recuperar grupos nombrados como un diccionario con **`groupdict()`**:

```
>>> m = re.match(r'(?P<first>\w+) (?P<last>\w+)', 'Jane
>>> m.groupdict()
{'first': 'Jane', 'last': 'Doe'}
```

Los grupos nombrados son útiles porque le permiten usar nombres fáciles de recordar en lugar de tener que recordar números. Aquí hay un ejemplo de una RE del módulo **`imaplib`**:

```
InternalDate = re.compile(r'INTERNALDATE "'
        r'(?P<day>[ 123][0-9])-(?P<mon>[A-Z][a-z][a-z])-
        r'(?P<year>[0-9][0-9][0-9][0-9])'
        r' (?P<hour>[0-9][0-9]):(?P<min>[0-9][0-9]):(?P<
        r' (?P<zonen>[-+])(?P<zoneh>[0-9][0-9])(?P<zonem
        r'"')
```

Obviamente, es mucho más fácil recuperar `m.group('zonem')`, en lugar de tener que recordar recuperar el grupo 9.

La sintaxis de las referencias inversas en una expresión como `(...)\1` se refiere al número del grupo. Naturalmente, existe una variante que usa el nombre del grupo en lugar del número. Esta es otra extensión de Python: `(?P=name)` indica que el contenido del grupo llamado *name* debe coincidir nuevamente en el punto actual. La expresión regular para encontrar palabras duplicadas, `\b(\w+)\s+\1\b` también se puede escribir como `\b(?P<word>\w+)\s+(?P=word)\b`:

```
>>> p = re.compile(r'\b(?P<word>\w+)\s+(?P=word)\b')
>>> p.search('Paris in the the spring').group()
'the the'
```

Lookahead assertions

Otra aserción de ancho cero es la aserción de anticipación. Las afirmaciones anticipadas están disponibles tanto en forma positiva como negativa, y tienen este aspecto:

`(?=...)`
: Aserción de anticipación positiva. Esto tiene éxito si la expresión regular contenida, representada aquí por `...`, coincide con éxito en la ubicación actual y falla en caso contrario. Pero, una vez que se ha probado la expresión contenida, el motor de comparación no avanza en absoluto; el resto del patrón se intenta justo donde comenzó la aserción.

`(?!...)`
: Aserción de anticipación negativa. Esto es lo opuesto a la flag positiva; tiene éxito si la expresión contenida *no* coincide con la posición actual en la cadena.

Para que esto sea concreto, veamos un caso en el que una búsqueda anticipada es útil. Considere un patrón simple para hacer coincidir un nombre de archivo y dividirlo en un nombre base y una extensión, separados por un `.`. Por ejemplo, en `news.rc`, `news` es el nombre base y `rc` es la extensión del nombre del archivo.

El patrón para que coincida con esto es bastante simple:

```
.*[.].*$
```

Tenga en cuenta que el `.` Debe tratarse especialmente porque es un metacarácter, por lo que está dentro de una clase de carácter para coincidir solo con ese carácter específico. También observe el final `$`; esto se agrega para garantizar que todo el resto de la cadena deba incluirse en la extensión. Esta expresión regular coincide con `foo.bar` y `autoexec.bat` y `sendmail.cf` y `printers.conf`.

Ahora, considere complicar un poco el problema; ¿Qué sucede si desea hacer coincidir los nombres de archivo donde la extensión no es `bat`? Algunos intentos incorrectos:

```
.*[.][^b].*$
```

The first attempt above tries to exclude `bat` by requiring that the first character of the extension is not a `b`. This is wrong, because the pattern also doesn’t match `foo.bar`.

```
.*[.]([^b]..|.[^a].|..[^t])$
```

La expresión se vuelve más desordenada cuando intenta parchear la primera solución requiriendo que coincida uno de los siguientes casos: el primer carácter de la extensión no es `b`; el segundo carácter no es `a`; o el tercer carácter no es `t`. Esto acepta `foo.bar` y rechaza `autoexec.bat`, pero requiere una extensión de tres letras y no acepta un nombre de archivo con una extensión de dos letras como `sendmail.cf`. Complicaremos el patrón nuevamente en un esfuerzo por arreglarlo.

```
.*[.]([^b].?.?|.[^a]?.?|..?[^t]?)$
```

En el tercer intento, la segunda y tercera letras se hacen opcionales para permitir extensiones coincidentes de menos de tres caracteres, como `sendmail.cf`.

El patrón se está volviendo realmente complicado ahora, lo que dificulta su lectura y comprensión. Peor aún, si el problema cambia y desea excluir tanto `bat` y `exe` como extensiones, el patrón se volvería aún más complicado y confuso.

Una mirada anticipada negativa atraviesa toda esta confusión:

```
.*[.](?!bat$)[^.]*$
```

The negative lookahead means: if the expression `bat` doesn't match at this point, try the rest of the pattern; if `bat$` does match, the whole pattern will fail. The trailing `$` is required to ensure that something like `sample.batch`, where the extension only starts with `bat`, will be allowed. The `[^.]*` makes sure that the pattern works when there are multiple dots in the filename.

Ahora es fácil excluir otra extensión de nombre de archivo; simplemente agréguelo como una alternativa dentro de la aserción. El siguiente patrón excluye los nombres de archivo que terminan en `bat` o `exe`:

```
.*[.](?!bat$|exe$)[^.]*$
```

Modifying strings

Hasta este punto, simplemente hemos realizado búsquedas en una cadena de caracteres estática. Las expresiones regulares también se utilizan comúnmente para modificar cadenas de varias formas, utilizando los siguientes métodos de patrón:

Método/atributo	Objetivo
`split()`	Divida la cadena de caracteres en una lista, dividiéndola donde coincida la RE
`sub()`	Encuentra todas las subcadenas de caracteres donde coincida la RE y las reemplaza con una cadena de caracteres diferente
`subn()`	Hace lo mismo que **`sub()`**, pero retorna la nueva cadena de caracteres y el número de reemplazos

Splitting strings

El método `split()` de un patrón que divide una cadena de caracteres donde la RE coincide, retornando una lista de las piezas. Es similar al método de cadenas de caracteres `split()` pero proporciona mucha más generalidad en los delimitadores por los que puede dividir; cadena de caracteres **`split()`** solo admite la división por espacios en blanco o por una cadena fija. Como era de esperar, también hay una función a nivel de módulo `re.split()`.

.split(*string*[, *maxsplit*=*0*])

> Dividir *string* por las coincidencias de la expresión regular. Si se utilizan paréntesis de captura en la RE, su contenido también se retornará como parte de la lista resultante. Si *maxsplit* es distinto de cero, se realizan como máximo divisiones *maxsplit* .

Puede limitar el número de divisiones realizadas, pasando un valor para *maxsplit*. Cuando *maxsplit* es distinto de cero, se realizarán como máximo *maxsplit* divisiones, y el resto de la cadena de caracteres se retorna como el elemento final de la lista. En el siguiente ejemplo, el delimitador es cualquier secuencia de caracteres no alfanuméricos.

```
>>> p = re.compile(r'\W+')
>>> p.split('This is a test, short and sweet, of split()
['This', 'is', 'a', 'test', 'short', 'and', 'sweet', 'of
>>> p.split('This is a test, short and sweet, of split()
['This', 'is', 'a', 'test, short and sweet, of split().'
```

A veces, no solo le interesa cuál es el texto entre delimitadores, sino que también necesita saber cuál era el delimitador. Si se utilizan paréntesis de captura en la RE, sus valores también se retornan como parte de la lista. Compare las siguientes llamadas:

```
>>> p = re.compile(r'\W+')
>>> p2 = re.compile(r'(\W+)')
>>> p.split('This... is a test.')
['This', 'is', 'a', 'test', '']
```

```
>>> p2.split('This... is a test.')
['This', '... ', 'is', ' ', 'a', ' ', 'test', '.', '']
```

La función de nivel de módulo `re.split()` agrega la RE que se usará como primer argumento, pero por lo demás es el mismo.

```
>>> re.split(r'[\W]+', 'Words, words, words.')
['Words', 'words', 'words', '']
>>> re.split(r'([\W]+)', 'Words, words, words.')
['Words', ', ', 'words', ', ', 'words', '.', '']
>>> re.split(r'[\W]+', 'Words, words, words.', 1)
['Words', 'words, words.']
```

Search and replace

Otra tarea común es encontrar todas las coincidencias para un patrón y reemplazarlas con una cadena de caracteres diferente. El método `sub()` toma un valor de reemplazo, que puede ser una cadena de caracteres o una función, y la cadena de caracteres a procesar.

.sub(*replacement*, *string*[, *count*=*0*])

Retorna la cadena de caracteres obtenida al reemplazar las apariciones no superpuestas del extremo izquierdo de la RE en *string* por el reemplazo *replacement*. Si no se encuentra el patrón, el *string* se retorna sin cambios.

El argumento opcional *count* es el número máximo de ocurrencias de patrones que se reemplazarán; *count* debe ser un número entero no negativo. El valor predeterminado de 0 significa reemplazar todas las ocurrencias.

Aquí hay un ejemplo simple del uso del método `sub()`. Reemplaza los nombres de los colores con la palabra `colour`:

```
>>> p = re.compile('(blue|white|red)')
>>> p.sub('colour', 'blue socks and red shoes')
'colour socks and colour shoes'
>>> p.sub('colour', 'blue socks and red shoes', count=1)
```

```
'colour socks and red shoes'
```

El método `subn()` hace el mismo trabajo, pero retorna una tupla de 2 que contiene el nuevo valor de cadena de caracteres y el número de reemplazos que se realizaron:

```
>>> p = re.compile('(blue|white|red)')
>>> p.subn('colour', 'blue socks and red shoes')
('colour socks and colour shoes', 2)
>>> p.subn('colour', 'no colours at all')
('no colours at all', 0)
```

Las coincidencias vacías se reemplazan solo cuando no son adyacentes a una coincidencia vacía anterior.

```
>>> p = re.compile('x*')
>>> p.sub('-', 'abxd')
'-a-b--d-'
```

Si *replacement* es una cadena, se procesan los escapes de barra invertida que contenga. Es decir, `\n` se convierte en un solo carácter de nueva línea, `\r` se convierte en una REtorno de carro, y así sucesivamente. Los escapes desconocidos como `\&` se dejan en paz. Las referencias inversas, como `\6`, se reemplazan con la subcadena de caracteres que coincide con el grupo correspondiente a la RE. Esto le permite incorporar partes del texto original en la cadena de reemplazo resultante.

Este ejemplo hace coincidir la palabra `section` seguida de una cadena encerrada entre `{`, `}`, y cambia `section` a `subsection`:

```
>>> p = re.compile('section{ ( [^}]* ) }', re.VERBOSE)
>>> p.sub(r'subsection{\1}','section{First} section{seco
'subsection{First} subsection{second}'
```

También hay una sintaxis para hacer referencia a grupos con nombre según lo definido por la sintaxis `(?P<name>…)`. `\g<name>` usará la subcadena de caracteres que coincide con el grupo llamado `name`, y `\g<number>` usa el número de grupo correspondiente. `\g<2>` es equivalente a `\2`, pero no es ambiguo en una cadena de reemplazo como `\g<2>0`. (`\20` se interpretaría como una referencia al grupo 20, no como una referencia al grupo 2 seguido del carácter literal `'0'`.) Las siguientes sustituciones son todas equivalentes, pero use las tres variaciones de la cadena de reemplazo.

```
>>> p = re.compile('section{ (?P<name> [^}]* ) }', re.VE
>>> p.sub(r'subsection{\1}','section{First}')
'subsection{First}'
>>> p.sub(r'subsection{\g<1>}','section{First}')
'subsection{First}'
>>> p.sub(r'subsection{\g<name>}','section{First}')
'subsection{First}'
```

replacement también puede ser una función, lo que le brinda aún más control. Si *replacement* es una función, la función se llama para cada ocurrencia no superpuesta de *pattern*. En cada llamada, a la función se le pasa un argumento match object para la coincidencia y puede usar esta información para calcular la cadena de reemplazo deseada y retornarla.

En el siguiente ejemplo, la función de reemplazo traduce decimales a hexadecimales:

```
>>> def hexrepl(match):
...     "Return the hex string for a decimal number"
...     value = int(match.group())
...     return hex(value)
...
>>> p = re.compile(r'\d+')
>>> p.sub(hexrepl, 'Call 65490 for printing, 49152 for u
'Call 0xffd2 for printing, 0xc000 for user code.'
```

Cuando se usa la función *module-level* **`re.sub()`**, el patrón se pasa como primer argumento. El patrón puede proporcionarse como un objeto o como una cuerda; Si necesita especificar marcas de expresión regular, debe usar un objeto de patrón como primer parámetro o usar modificadores incrustados en la cadena de patrón, por ejemplo `sub("(?i)b+", "x", "bbbb BBBB")` retorna `'x x'`.

Common problems

Las expresiones regulares son una herramienta poderosa para algunas aplicaciones, pero de alguna manera su comportamiento no es intuitivo y, a veces, no se comportan de la forma esperada. Esta sección señalará algunos de los errores más comunes.

Use string methods

A veces, usar el módulo `re` es un error. Si está haciendo coincidir una cadena de caracteres fija, o una clase de un solo caracter, y no está usando ninguna característica `re` como la marca `IGNORECASE`, entonces todo el poder de las expresiones regulares puede que no sea necesario. Las cadenas tienen varios métodos para realizar operaciones con cadenas fijas y, por lo general, son mucho más rápidas, porque la implementación es un único bucle C pequeño que se ha optimizado para este propósito, en lugar del motor de expresión regular más grande y generalizado.

Un ejemplo podría ser reemplazar una sola cadena fija por otra; por ejemplo, puede reemplazar `word` por `deed`. `re.sub()` parece la función a utilizar para esto, pero considere el método `replace()`. Tenga en cuenta que `replace()` también reemplazará `word` dentro de las palabras, convirtiendo `swordfish` en `sdeedfish`, pero la RE naíf `word` también lo habría hecho. (Para evitar realizar la sustitución en partes de palabras, el patrón tendría que ser `\bword\b`, para requerir que `word` tenga un límite de palabra en cada lado. Esto lleva el trabajo más allá de las habilidades de `replace()`.)

Otra tarea común es eliminar cada aparición de un solo carácter de una cadena o reemplazarlo con otro solo carácter. Puede hacer esto con algo como `re.sub('\n', ' ', S)`, pero `translate()` es capaz de realizar ambas tareas y será más rápido que cualquier expresión regular la operación puede ser.

En resumen, antes de pasar al módulo `re`, considere si su problema puede resolverse con un método de cadena de caracteres más rápido y simple.

match() versus search()

La función **`match()`** solo verifica si la RE coincide con el comienzo de la cadena de caracteres, mientras que **`search()`** buscará una coincidencia en la cadena de caracteres. Es importante tener en cuenta esta distinción. Recuerde **`match()`** solo informará una coincidencia exitosa que comenzará en 0; si la coincidencia no comienza en cero, **`match()`** *no* lo informará.

```
>>> print(re.match('super', 'superstition').span())
(0, 5)
>>> print(re.match('super', 'insuperable'))
None
```

Por otro lado, **`search()`** escaneará hacia adelante a través de la cadena de caracteres, informando la primera coincidencia que encuentre.

```
>>> print(re.search('super', 'superstition').span())
(0, 5)
>>> print(re.search('super', 'insuperable').span())
(2, 7)
```

A veces, tendrá la tentación de seguir usando **`re.match()`**, y simplemente agregar `.*` al frente de su RE. Resista esta tentación y use **`re.search()`** en su lugar. El compilador de expresiones regulares realiza un análisis de las RE para acelerar el proceso de búsqueda de coincidencias. Uno de esos análisis determina cuál debe ser el primer carácter de una coincidencia; por ejemplo, un patrón que comienza con `Crow` debe coincidir con una `'C'`.El análisis permite que el motor escanee rápidamente a través de la cadena en busca del carácter inicial, solo probando la coincidencia completa si se encuentra una `'C'`.

Agregar `.*` anula esta optimización, lo que requiere escanear hasta el final de la cadena y luego retroceder para encontrar una coincidencia para el resto de la RE. Utilice **`re.search()`** en su lugar.

Greedy versus non-greedy

Al repetir una expresión regular, como en `a*`, la acción resultante es consumir la mayor cantidad posible del patrón. Este hecho suele molestarle cuando intenta hacer coincidir un par de delimitadores equilibrados, como los corchetes angulares que rodean una etiqueta HTML. El patrón ingenuo para hacer coincidir una sola etiqueta HTML no funciona debido a la naturaleza codiciosa de `.*`.

```
>>> s = '<html><head><title>Title</title>'
>>> len(s)
32
>>> print(re.match('<.*>', s).span())
(0, 32)
>>> print(re.match('<.*>', s).group())
<html><head><title>Title</title>
```

La RE coincide con el `'<'` en `'<html>'`, y el `.*` consume el resto de la cadena de caracteres. Sin embargo, aún queda más en la RE y el > no puede coincidir al final de la cadena de caracteres, por lo que el motor de la expresión regular tiene que retroceder carácter por carácter hasta que encuentre una coincidencia para el >. La coincidencia final se extiende desde el `'<'` en `'<html>'` al `'>'` en `'</title>'`, que no es lo que queremos.

En este caso, la solución es utilizar los cuantificadores no codiciosos `*?`, `+?`, `??`, o `{m,n}?`, que coinciden con la *menor* cantidad de texto posible. En el ejemplo anterior, el `'>'` se prueba inmediatamente después de que el primer `'<'` coincida, y cuando falla, el motor avanza un carácter a la vez, volviendo a intentar el `'>'` en cada paso. Esto produce el resultado correcto:

```
>>> print(re.match('<.*?>', s).group())
<html>
```

(Tenga en cuenta que analizar HTML o XML con expresiones regulares es doloroso. Los patrones rápidos y sucios manejarán casos comunes, pero HTML y XML tienen casos especiales que romperán la expresión regular obvia; para cuando haya escrito una expresión regular que maneja todos los casos posibles, los patrones serán *muy* complicados. Utilice un módulo analizador HTML o XML para tales tareas.)

Usando re.VERBOSE

Probablemente ya haya notado que las expresiones regulares son una notación muy compacta, pero no son muy legibles. Las RE de complejidad moderada pueden convertirse en largas colecciones de barras invertidas, paréntesis y metacarácteres, lo que dificulta su lectura y comprensión.

Para tales RE, especificar el flag **`re.VERBOSE`** al compilar la expresión regular puede ser útil, porque le permite formatear la expresión regular con mayor claridad.

La flag `re.VERBOSE` tiene varios efectos. Los espacios en blanco en la expresión regular que *no están* dentro de una clase de caracteres se ignoran. Esto significa que una expresión como `dog | cat` es equivalente al menos legible `dog|cat`, pero `[a b]` seguirá coincidiendo con los caracteres `'a'`, `'b'` o un espacio. Además, también puede poner comentarios dentro de una RE; los comentarios se extienden desde un carácter `#` hasta la siguiente línea nueva. Cuando se usa con cadenas entre comillas triples, esto permite que las REs sean formateados de manera más ordenada:

```
pat = re.compile(r"""
 \s*                 # Skip leading whitespace
 (?P<header>[^:]+)   # Header name
 \s* :               # Whitespace, and a colon
 (?P<value>.*?)      # The header's value -- *? used to
                     # lose the following trailing white
 \s*$                # Trailing whitespace to end-of-li
""", re.VERBOSE)
```

Esto es mas legible que:

```
pat = re.compile(r"\s*(?P<header>[^:]+)\s*:(?P<value>.*?
```

Feedback

Regular expressions are a complicated topic. Did this document help you understand them? Were there parts that were unclear, or problems you encountered that weren’t covered here? If so, please send suggestions for improvements to the issue tracker.

El libro más completo sobre expresiones regulares es casi con certeza *Mastering Regular Expressions de Jeffrey Friedl*, publicado por *O'Reilly*. Desafortunadamente, se concentra exclusivamente en los tipos de expresiones regulares de Perl y Java, y no contiene ningún material de Python, por lo que no será útil como referencia para la programación en Python. (La primera edición cubría el módulo **`regex`** de Python, ahora eliminado, que no le ayudará mucho.) Considere sacarlo de su biblioteca.

HOW TO - Programación con sockets

Autor:

Gordon McMillan

Resumen

Los sockets son usados casi en todas partes pero son una de las tecnologías más incomprendidas. Esta es una descripción general de los sockets. No es realmente un tutorial, todavía tendrás trabajo para hacer que las cosas funcionen. No cubre los pequeños detalles (y hay muchos de ellos) pero espero que pueda dar suficiente información para comenzar a usarlos decentemente.

Sockets

Solo voy a hablar de los sockets *INET* (como IPv4), pues solo ellos cubren el 99% del uso de los sockets. Y solo voy a hablar sobre los sockets *STREAM* (como TCP), a menos que realmente sepas lo que haces (y en ese caso esta guía no es para ti), tendrás mejor comportamiento y rendimiento con un socket *STREAM* que con cualquier otro. Voy a tratar de aclarar el misterio de que es un socket, además de algunas ideas sobre como trabajar con sockets bloqueantes y no bloqueantes. Pero voy a comenzar hablando de los sockets bloqueantes, necesitarás saber como funcionan antes de lidiar con los no bloqueantes.

Parte del problema para entenderlos es que «socket» puede significar un número de cosas ligeramente diferentes dependiendo del contexto. Entonces, primero vamos a hacer una distinción entre sockets «cliente» - un extremo de una conversación, y un socket «servidor», que es más como una central de teléfonos. La aplicación cliente (tu navegador, por ejemplo) usa sockets «cliente» exclusivamente; el servidor web con quien se está comunicando usa sockets «cliente» y «servidor».

Historia

De las varias formas de comunicación entre procesos (IPC) los sockets son, por mucho, la más popular. En cualquier plataforma es probable que existan otras formas de IPC más rápidas, pero en comunicación multiplataforma los sockets son los únicos competidores.

Fueron inventados en Berkeley como parte del sabor BSD de Unix. Se propagaron como la pólvora con Internet. Con razón: la combinación de sockets con INET hace que hablar con máquinas arbitrarias de todo el mundo sea increíblemente fácil (al menos en comparación con otros esquemas).

Creando un socket

De manera general, cuando hiciste click en el enlace que te trajo a esta página tu navegador hizo algo como lo siguiente:

```
# create an INET, STREAMing socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# now connect to the web server on port 80 - the normal
s.connect(("www.python.org", 80))
```

Cuando `connect` termina, el socket `s` puede ser usado en una petición para traer el texto de la página. El mismo socket leerá la respuesta y luego será destruido. Así es, destruido. Los sockets cliente son normalmente usados solo para un intercambio (o un pequeño numero se intercambios secuenciales).

Lo que sucede en el servidor web es un poco más complejo. Primero, el servidor web crea un «socket servidor»:

```
# create an INET, STREAMing socket
serversocket = socket.socket(socket.AF_INET, socket.SOCK
# bind the socket to a public host, and a well-known por
serversocket.bind((socket.gethostname(), 80))
```

```
# become a server socket
serversocket.listen(5)
```

Un par de cosas que señalar: usamos `socket.gethostname()` para que el socket fuera visible al mundo exterior. Si hubiésemos usado `s.bind(('localhost', 80))` o `s.bind(('127.0.0.1', 80))` habríamos tenido un socket servidor pero solo habría sido visible en la misma máquina. `s.bind(('', 80))` especifica que el socket es accesible desde cualquier dirección que tenga la máquina.

Algo más para señalar: los números de puerto bajos son normalmente reservados para servicios «conocidos» (HTTP, SNMP, etc.). Si estás probando los sockets usa un número grande (4 dígitos).

Finalmente, el argumento que se le pasa a `listen` le indica a la librería del socket que queremos poner en cola no más de 5 solicitudes de conexión (el máximo normal) antes de rechazar conexiones externas. Si el resto del código está escrito correctamente eso debería ser suficiente.

Ahora que tenemos un socket servidor escuchando en el puerto 80 ya podemos entrar al bucle principal del servidor web:

```
while True:
    # accept connections from outside
    (clientsocket, address) = serversocket.accept()
    # now do something with the clientsocket
    # in this case, we'll pretend this is a threaded ser
    ct = make_client_thread(clientsocket)
    ct.start()
```

Existen en realidad 3 maneras generales en las cuales este bucle puede funcionar - despachar un hilo para manejar `clientsocket`, crear un proceso nuevo para manejar `clientsocket` o reestructurar esta aplicación para usar sockets no bloqueantes y multiplexar entre nuestro «socket servidor» y cualquier `clientsocket` activo usando `select`. Más sobre esto después. Lo importante a entender ahora es: esto es *todo* lo que un «socket servidor hace». No manda ningún dato. No recibe ningún dato. Solo produce «sockets clientes». Cada `clientsocket` es creado en respuesta a algún otro «socket cliente» que hace `connect()` al host y al puerto al que estamos vinculados. Tan pronto como hemos credo ese `clientsocket` volvemos a escuchar por más conexiones. Los dos «clientes» son libres de «conversar» entre ellos - están usando algún puerto asignado dinámicamente que será reciclado cuando la conversación termine.

IPC

Si necesitas conexiones IPC rápidas entre dos procesos en una misma máquina puedes revisar los *pipes* o la memoria compartida. Si decides usar sockets `AF_INET`, vincula el servidor con `"localhost"`. En la mayoría de las plataformas, esto tomará un atajo alrededor de algunas capas del código de red y será un poco más rápido.

Ver también

El módulo **`multiprocessing`** integra IPC multiplataforma en un API de alto nivel.

Usando un socket

Lo primero a señalar es que el «socket cliente» del navegador y el «socket cliente» del servidor web son bestias idénticas. Es decir, esta es una conversación *peer to peer*. O para decirlo de otra manera, *como diseñador, tendrás que decidir cuáles son las reglas de etiqueta para una conversación*. Normalmente, el socket que se conecta inicia la conversación, enviando una solicitud o tal vez un inicio de sesión. Pero esa es una decisión de diseño: no es una regla de los sockets.

Hay dos conjuntos de verbos que se usan para la comunicación. Puedes usar `send` y `recv` o puedes transformar tu socket cliente en algo similar a un archivo y usar `read` y `write`. Esta última es la forma en la que Java presenta sus sockets. No voy a hablar acerca de eso aquí, excepto para advertirte que necesitas usar `flush` en los sockets. Estos son archivos en buffer, y un error común es usar `write` para escribir algo y luego usar `read` para leer la respuesta. Sin usar `flush` en este caso, puedes terminar esperando la respuesta por siempre porque la petición estaría aún en el buffer de salida.

Ahora llegamos al principal problema de los sockets - `send` y `recv` operan en los buffers de red. Ellos no manejan necesariamente todos los bytes que se les entrega (o espera de ellos), porque su enfoque principal es manejar los buffers de red. En general, ellos retornan cuando los buffers de red asociados se han llenado (`send`) o vaciado (`recv`). Luego ellos dicen cuántos bytes manejaron. Es *tu* responsabilidad llamarlos nuevamente hasta que su mensaje haya sido tratado por completo.

Cuando `recv` retorna 0 bytes significa que el otro lado ha cerrado (o está en el proceso de cerrar) la conexión. No recibirás más datos de esta conexión. Nunca. Es posible que puedas mandar datos exitosamente. De eso voy a hablar más tarde.

Un protocolo como HTTP usa un socket para una sola transferencia. El cliente manda una petición, luego lee la respuesta. Eso es todo. El socket es descartado. Esto significa que un cliente puede detectar el final de la respuesta al recibir 0 bytes.

Pero si planeas reusar el socket para más transferencias, tienes que darte cuenta que *no hay* EOT *en un socket.* Repito: si la llamada a `send` o `recv` de un socket retorna después de manejar 0 bytes, la conexión se ha interrumpido. Si la conexión *no* se ha interrumpido, puedes esperar un `recv` para siempre, porque el socket no te dirá cuando no hay más nada por leer (por ahora). Ahora, si piensas sobre eso un poco, te darás cuenta de una verdad fundamental de los sockets: *los mensajes deben ser de longitud fija* (ouch), *o ser delimitados* (ouch), *o indicar que tan largo son* (mucho mejor), *o terminar cerrando la conexión.* La elección es completamente tuya (pero hay algunas vías más correctas que otras).

Asumiendo que no quieres terminar la conexión, la solución más simple es un mensaje de longitud fija:

```
class MySocket:
    """demonstration class only
```

```
      - coded for clarity, not efficiency
    """

    def __init__(self, sock=None):
        if sock is None:
            self.sock = socket.socket(
                            socket.AF_INET, socket.SOCK_
        else:
            self.sock = sock

    def connect(self, host, port):
        self.sock.connect((host, port))

    def mysend(self, msg):
        totalsent = 0
        while totalsent < MSGLEN:
            sent = self.sock.send(msg[totalsent:])
            if sent == 0:
                raise RuntimeError("socket connection br
            totalsent = totalsent + sent

    def myreceive(self):
        chunks = []
        bytes_recd = 0
        while bytes_recd < MSGLEN:
            chunk = self.sock.recv(min(MSGLEN - bytes_re
            if chunk == b'':
                raise RuntimeError("socket connection br
            chunks.append(chunk)
            bytes_recd = bytes_recd + len(chunk)
        return b''.join(chunks)
```

El código de envío aquí es usable para prácticamente cualquier esquema de mensajería - en Python envías cadenas y usas `len()` para determinar su longitud (incluso si tiene caracteres `\0` incrustados). Es principalmente el código receptor el que se vuelve más complejo. (Y en C no es mucho peor, excepto que no puedes usar `strlen` si el mensaje tiene `\0` incrustados).

La mejora más fácil es hacer que el primer caracter del mensaje un indicador del tipo de mensaje y que el tipo determine la longitud. Ahora tienes dos `recv` - el primero para obtener (al menos) ese primer caracter para conocer la longitud, y el segundo en un bucle para obtener el resto. Si decides ir por el camino del delimitador, estarás recibiendo un fragmento de tamaño arbitrario (4096 o 8192 son a menudo buenas elecciones para tamaños de buffers de red) y escaneando lo que recibas en busca del delimitador.

Hay una complicación de la que estar consiente: si el protocolo conversacional permite mandar múltiples mensajes consecutivos (sin ningún tipo de respuesta), y pasas a `recv` un tamaño de fragmento arbitrario poder terminar leyendo el inicio de un próximo mensaje. Tendrás que dejarlo aparte y guardarlo hasta que sea necesario.

Prefijar el mensaje con su longitud (por ejemplo, 5 caracteres numéricos) se vuelve más complicado porque (créalo o no), puede que no recibas los 5 caracteres en una llamada a `recv`. Para proyectos pequeños te saldrás con la tuya; pero con altas cargas de red, tu código se romperá rápidamente a menos que uses dos `recv` en bucle - el primero para determinar la longitud, el segundo para obtener la parte del mensaje. Sucio. También será cuando descubras que `send` no siempre logra enviar todo de una sola vez. Y a pesar de haber leído esto eventualmente te va a morder!

Con interés de espacio, la construcción de tu carácter (y preservar mi posición competitiva), estas mejoras se dejan como un ejercicio para el lector. Pasemos a la limpieza.

Datos binarios

Es perfectamente posible enviar datos binarios a través de un socket. El principal problema es que no todas las máquinas utilizan los mismos formatos para datos binarios. Por ejemplo, orden de bytes de red [https://en.wikipedia.org/wiki/Endianness#Networking] es big-endian, con el byte más significativo primero, por lo que es un entero de 16 bits con el valor `1` serían los dos bytes hexadecimales `00 01`. Sin embargo, los procesadores más comunes (x86/AMD64, ARM, RISC-V) son little-endian, con el byte menos significativo primero; ese mismo `1` sería `01 00`.

Las bibliotecas de socket tienen llamadas para convertir enteros de 16 y 32 bits - `ntohl, htonl, ntohs, htons` donde «n» significa *network* (red) y «h» significa *host* (host), «s» significa *short* (corto) y «l» significa *long* (largo). Cuando el orden de la red es el orden del host, estos no hacen nada, pero cuando la máquina está invertida en bytes, intercambian los bytes de manera adecuada.

En estos días de máquinas de 64 bit, la representación ASCII de los datos binarios es con frecuencia más pequeña que la representación binaria. Esto es porque una sorprendente cantidad de veces, todos esos enteros tienen el valor 0, o tal vez 1. La cadena `"0"` tendría dos bytes, mientras un entero completo de 64 bit tendría 8. Por supuesto, esto no funciona bien con los mensajes de longitud fija. Decisiones, decisiones.

Desconectando

Estrictamente hablando, se supone que debes usar `shutdown` en un socket antes de cerrarlo con `close`. `shutdown` es un aviso para el socket en el otro lado. Dependiendo del argumento que se le pase, puede significar «No voy a mandar más datos, pero voy a escuchar» o «No estoy escuchando, adiós!». La mayoría de bibliotecas para sockets, sin embargo, están tan acostumbradas a que los programadores ignoren esta parte de la etiqueta que normalmente `close` es lo mismo que `shutdown(); close()`. Por tanto en la mayoría de las situaciones usar `shutdown` de manera explícita no es necesario.

Una forma de usar `shutdown` de manera efectiva es en un intercambio similar a *HTTP*. El cliente manda una petición y entonces hace un `shutdown(1)`. Esto le dice al servidor «El cliente terminó de enviar, pero todavía puede recibir». El servidor puede detectar «EOF» (Fin del Archivo) al recibir 0 bytes. Puede asumir que se completó la petición. El servidor envía una respuesta. Si el `send` termina satisfactoriamente entonces, en efecto, el cliente todavía estaba recibiendo.

Python lleva el apagado automático un paso más allá, y dice que cuando un socket es eliminado por el recolector de basura, automáticamente llama a `close` si es necesario. Pero confiar en esto es un mal hábito. Si tu socket simplemente desaparece sin llamar a `close`, el socket del otro lado puede colgarse indefinidamente, pensando que solo estas siendo lento. *Por favor* cierra los sockets cuando termines.

Cuando los sockets mueren

Probablemente lo peor de usar sockets bloqueantes es lo que pasa cuando el otro lado se apaga inesperadamente (sin llamar a `close`). Tu socket es probable que se cuelgue. TCP es un protocolo confiable, y va a esperar un largo, largo tiempo antes de rendirse con una conexión. Si estás usando hilos, todo el hilo está esencialmente muerto. No hay mucho que puedas hacer respecto a eso. A menos que no estés haciendo algo tonto, como mantener un bloqueo mientras se realiza una lectura bloqueante, el hilo realmente no estará consumiendo muchos recursos. *No* trates de matar el hilo - parte de la razón por la que los hilos son más eficientes que los procesos es que evitan la complicación asociada con el reciclaje automático de recursos. En otras palabras, si te las arreglas para matar el hilo, es muy probable que todo el proceso termine arruinado.

Sockets no bloqueantes

Si has entendido todo lo anterior, ya conoces la mayor parte de lo que necesitas saber sobre las mecánicas del uso de los sockets. Usarás las mismas llamadas, de la misma manera. Es solo eso, si lo haces correctamente, tu aplicación estará casi correcta.

En Python, usa `socket.setblocking(False)` para que no sea bloqueante. En C, es más complejo (por un lado, deberá elegir entre el sabor BSD `O_NONBLOCK` y el sabor POSIX casi indistinguible `O_NDELAY`, que es completamente diferente de `TCP_NODELAY`) , pero es exactamente la misma idea. Haz esto después de crear el socket, pero antes de usarlo. (En realidad, si estás loco, puedes cambiar de un lado a otro).

La principal diferencia mecánica es que `send`, `recv`, `connect` y `accept` pueden retornar sin haber hecho nada. Tu tienes (por supuesto) un número de elecciones. Puedes verificar el código de retorno y los códigos de error y en general volverte loco. Si no me crees pruébalo alguna vez. Tu aplicación crecerá grande, con errores y consumirá todo el CPU. Así que vamos a saltarnos las soluciones descerebradas y hacerlo correctamente.

Usando `select`.

En C, usar `select` es algo complejo. En Python es pan comido, pero está lo suficientemente cercano a la versión de C que si entiendes el `select` en Python tendrás pocos problemas con él el C:

```
ready_to_read, ready_to_write, in_error = \
               select.select(
                  potential_readers,
                  potential_writers,
                  potential_errs,
                  timeout)
```

A `select` se le pasan tres listas: la primera contiene todos los sockets que puedes intentar leer; la segunda con todos los sockets que puedes intentar escribir, y la tercera (normalmente se deja vacía) todos los que quieras verificar los errores. Debes tener en cuenta que un socket puede ir en más de una lista. La llamada a `select` es bloqueante, pero puedes darle un tiempo de espera. Esto generalmente es una cosa sensata de hacer - dale un tiempo de espera largo (un minuto por ejemplo) a menos que tengas una buena razón para no hacerlo.

En el retorno tendrás tres listas. Estas contienen los sockets que son realmente legibles, escribibles y con error. Cada una de estas lista es un subconjunto (posiblemente vacío) de la lista correspondiente que pasaste.

Si un socket está en la lista retornada legible, puedes estar tan-seguro-como-podrías-estarlo-en-este-negocio que una llamada a `recv` en este socket va a retornar *algo*. La misma idea se aplica a la lista de escribibles. Serás capaz de mandar *algo*. Tal vez no todo lo que quieras, pero *algo* es mejor que nada. (Realmente, cualquier socket socket razonablemente saludable va a retornar como escribible - eso solo significa que el espacio de salida del buffer de red está disponible)

Si tienes un socket *servidor*, ponlo en la lista de *potenciales legibles*. Se retorna en la lista de legibles, una llamada a `accept` va a funcionar (casi seguro). Se has creado un nuevo socket para llamar a `connect` para conectarte con otro, ponlo en la lista de *potenciales escribibles*. Si retorna en la lista de escribibles, tienes una buena oportunidad de que esté conectado.

Realmente, `select` puede ser útil incluso con sockets bloqueantes. Es una manera de determinar si vas a bloquear - el socket retorna como leíble cuando hay algo en el buffer. Sin embargo, esto aun no sirve de ayuda con el problema de determinar si el otro extremo terminó, o solo está ocupado con otra cosa.

Alerta de portabilidad: En Unix, `select` funciona tanto con sockets como con archivos. No intentes esto en Windows. En Windows `select` funciona solo con sockets. También ten en cuenta que en C, muchas de las opciones más avanzadas de los sockets se hacen diferentes en Windows. De hecho, en Windows normalmente uso hilos (que funciona muy, muy bien) con los sockets.

Sorting Techniques

Autor:

Andrew Dalke and Raymond Hettinger

Las listas de Python tienen un método incorporado `list.sort()` que modifica la lista in situ. También hay una función incorporada `sorted()` que crea una nueva lista ordenada a partir de un iterable.

En este documento exploramos las distintas técnicas para ordenar datos usando Python.

Conceptos básicos de ordenación

Una clasificación ascendente simple es muy fácil: simplemente llame a la función `sorted()`. Retorna una nueva lista ordenada:

```
>>> sorted([5, 2, 3, 1, 4])
[1, 2, 3, 4, 5]
```

También puede usar el método `list.sort()`. Modifica la lista in situ (y retorna `None` para evitar confusiones). Por lo general, es menos conveniente que `sorted()`, pero si no necesita la lista original, es un poco más eficiente.

```
>>> a = [5, 2, 3, 1, 4]
>>> a.sort()
>>> a
[1, 2, 3, 4, 5]
```

Otra diferencia es que el método `list.sort()` solo aplica para las listas. En contraste, la función `sorted()` acepta cualquier iterable.

```
>>> sorted({1: 'D', 2: 'B', 3: 'B', 4: 'E', 5: 'A'})
```

```
[1, 2, 3, 4, 5]
```

Funciones clave

The `list.sort()` method and the functions `sorted()`, `min()`, `max()`, `heapq.nsmallest()`, and `heapq.nlargest()` have a *key* parameter to specify a function (or other callable) to be called on each list element prior to making comparisons.

For example, here's a case-insensitive string comparison using `str.casefold()`:

```
>>> sorted("This is a test string from Andrew".split(),
['a', 'Andrew', 'from', 'is', 'string', 'test', 'This']
```

El valor del parámetro *key* debe ser una función (u otra invocable) que tome un solo argumento y retorne una clave para usar con fines de clasificación. Esta técnica es rápida porque la función de la tecla se llama exactamente una vez para cada registro de entrada.

Un uso frecuente es ordenar objetos complejos utilizando algunos de los índices del objeto como claves. Por ejemplo:

```
>>> student_tuples = [
...     ('john', 'A', 15),
...     ('jane', 'B', 12),
...     ('dave', 'B', 10),
... ]
>>> sorted(student_tuples, key=lambda student: student[2
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)
```

La misma técnica funciona para objetos con atributos nombrados. Por ejemplo:

```
>>> class Student:
...     def __init__(self, name, grade, age):
...         self.name = name
...         self.grade = grade
...         self.age = age
...     def __repr__(self):
```

```
...         return repr((self.name, self.grade, self.age

>>> student_objects = [
...     Student('john', 'A', 15),
...     Student('jane', 'B', 12),
...     Student('dave', 'B', 10),
... ]
>>> sorted(student_objects, key=lambda student: student.
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)
```

Objects with named attributes can be made by a regular class as shown above, or they can be instances of **`dataclass`** or a named tuple.

Operator Module Functions and Partial Function Evaluation

The key function patterns shown above are very common, so Python provides convenience functions to make accessor functions easier and faster. The **`operator`** module has **`itemgetter()`**, **`attrgetter()`**, and a **`methodcaller()`** function.

Usando esas funciones, los ejemplos anteriores se vuelven más simples y rápidos:

```
>>> from operator import itemgetter, attrgetter

>>> sorted(student_tuples, key=itemgetter(2))
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)

>>> sorted(student_objects, key=attrgetter('age'))
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)
```

Las funciones del módulo *operator* permiten múltiples niveles de clasificación. Por ejemplo, para ordenar por *grade* y luego por *age*:

```
>>> sorted(student_tuples, key=itemgetter(1,2))
[('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)
```

```
>>> sorted(student_objects, key=attrgetter('grade', 'age
[('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)
```

The **`functools`** module provides another helpful tool for making key-functions. The **`partial()`** function can reduce the arity [https://en.wikipedia.org/wiki/Arity] of a multi-argument function making it suitable for use as a key-function.

```
>>> from functools import partial
>>> from unicodedata import normalize

>>> names = 'Zoë Åbjørn Núñez Élana Zeke Abe Nubia Elois

>>> sorted(names, key=partial(normalize, 'NFD'))
['Abe', 'Åbjørn', 'Eloise', 'Élana', 'Nubia', 'Núñez', '

>>> sorted(names, key=partial(normalize, 'NFC'))
['Abe', 'Eloise', 'Nubia', 'Núñez', 'Zeke', 'Zoë', 'Åbj
```

Ascendente y descendente

Ambos **`list.sort()`** y **`sorted()`** aceptan un parámetro *reverse* con un valor booleano. Esto se usa para marcar tipos descendentes. Por ejemplo, para obtener los datos de los estudiantes en orden inverso de *edad*:

```
>>> sorted(student_tuples, key=itemgetter(2), reverse=Tr
[('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)

>>> sorted(student_objects, key=attrgetter('age'), rever
[('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)
```

Estabilidad de ordenamiento y ordenamientos complejos

Se garantiza que las clasificaciones serán estables [https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento#Estabilidad]. Eso significa que cuando varios registros tienen la misma clave, se conserva su orden original.

```
>>> data = [('red', 1), ('blue', 1), ('red', 2), ('blue'
>>> sorted(data, key=itemgetter(0))
[('blue', 1), ('blue', 2), ('red', 1), ('red', 2)]
```

Observe cómo los dos registros para *blue* conservan su orden original de modo que se garantice que `('blue', 1)` preceda a `('blue', 2)`.

Esta maravillosa propiedad le permite construir ordenamientos complejos en varias etapas. Por ejemplo, para ordenar los datos de estudiantes en orden descendente por *grade* y luego ascendente por *age*, ordene primero por *age* y luego por *grade*:

```
>>> s = sorted(student_objects, key=attrgetter('age'))
>>> sorted(s, key=attrgetter('grade'), reverse=True)
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)
```

Esto se puede encapsular en una función que tome una lista y tuplas (atributo, orden) para ordenarlas por múltiples pases.

```
>>> def multisort(xs, specs):
...     for key, reverse in reversed(specs):
...         xs.sort(key=attrgetter(key), reverse=reverse
...     return xs

>>> multisort(list(student_objects), (('grade', True), (
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)
```

El algoritmo Timsort [https://en.wikipedia.org/wiki/Timsort] utilizado en Python realiza múltiples ordenamientos de manera eficiente porque puede aprovechar cualquier orden ya presente en el conjunto de datos.

Decorate-Sort-Undecorate

Este patrón de implementación, llamado DSU (por sus siglas en inglés *Decorate-Sort-Undecorate*), se realiza en tres pasos:

- Primero, la lista inicial está «decorada» con nuevos valores que controlarán el orden en que se realizará el pedido.
- En segundo lugar, se ordena la lista decorada.
- Finalmente, los valores decorados se eliminan, creando una lista que contiene solo los valores iniciales en el nuevo orden.

Por ejemplo, para ordenar los datos de los estudiantes por *grade* utilizando el enfoque DSU:

```
>>> decorated = [(student.grade, i, student) for i, stuc
>>> decorated.sort()
>>> [student for grade, i, student in decorated]
[('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)
```

Esta técnica funciona porque las tuplas se comparan en orden lexicográfico; se comparan los primeros objetos; si hay objetos idénticos, se compara el siguiente objeto, y así sucesivamente.

No es estrictamente necesario en todos los casos incluir el índice *i* en la lista decorada, pero incluirlo ofrece dos ventajas:

- El orden es estable: si dos elementos tienen la misma clave, su orden se conservará en la lista ordenada.
- Los elementos originales no tienen que ser comparables porque el orden de las tuplas decoradas estará determinado por, como máximo, los dos primeros elementos. Entonces, por ejemplo, la lista original podría contener números complejos que no se pueden ordenar directamente.

Otro nombre para esta técnica es Transformación Schwartziana [https://en.wikipedia.org/wiki/Schwartzian_transform], después de que Randal L. Schwartz la popularizara entre los programadores de Perl.

Ahora que la clasificación de Python proporciona funciones clave, esta técnica ya no se usa con frecuencia.

Funciones de comparación

A diferencia de las funciones clave que devuelven un valor absoluto para la ordenación, una función de comparación calcula la ordenación relativa para dos entradas.

Por ejemplo, una escala de balance [https://upload.wikimedia.org/wikipedia/commons/1/17/Balance_à_tabac_1850.JPG] compara dos muestras dando un orden relativo: más ligero, igual o más pesado. Del mismo modo, una función de comparación como `cmp(a, b)` devolverá un valor negativo para menor que, cero si las entradas son iguales, o un valor positivo para mayor que.

Es habitual encontrar funciones de comparación al traducir algoritmos de otros lenguajes. Además, algunas bibliotecas proporcionan funciones de comparación como parte de su API. Por ejemplo, **`locale.strcoll()`** es una función de comparación.

Para adaptarse a estas situaciones, Python proporciona **`functools.cmp_to_key`** para envolver la función de comparación y hacerla utilizable como una función clave:

```
sorted(words, key=cmp_to_key(strcoll))  # locale-aware s
```

Strategies For Unorderable Types and Values

A number of type and value issues can arise when sorting. Here are some strategies that can help:

- Convert non-comparable input types to strings prior to sorting:

```
>>> data = ['twelve', '11', 10]
>>> sorted(map(str, data))
['10', '11', 'twelve']
```

This is needed because most cross-type comparisons raise a **`TypeError`**.

- Remove special values prior to sorting:

```
>>> from math import isnan
>>> from itertools import filterfalse
>>> data = [3.3, float('nan'), 1.1, 2.2]
>>> sorted(filterfalse(isnan, data))
[1.1, 2.2, 3.3]
```

This is needed because the IEEE-754 standard [https://en.wikipedia.org/wiki/IEEE_754] specifies that, «Every NaN shall compare unordered with everything, including itself.»

Likewise, `None` can be stripped from datasets as well:

```
>>> data = [3.3, None, 1.1, 2.2]
>>> sorted(x for x in data if x is not None)
[1.1, 2.2, 3.3]
```

This is needed because `None` is not comparable to other types.

- Convert mapping types into sorted item lists before sorting:

```
>>> data = [{'a': 1}, {'b': 2}]
>>> sorted(data, key=lambda d: sorted(d.items()))
[{'a': 1}, {'b': 2}]
```

This is needed because dict-to-dict comparisons raise a **`TypeError`**.

- Convert set types into sorted lists before sorting:

```
>>> data = [{'a', 'b', 'c'}, {'b', 'c', 'd'}]
>>> sorted(map(sorted, data))
[['a', 'b', 'c'], ['b', 'c', 'd']]
```

This is needed because the elements contained in set types do not have a deterministic order. For example, `list({'a', 'b'})` may produce either `['a', 'b']` or `['b', 'a']`.

Curiosidades

- Para ordenar teniendo en cuenta la localización, utilice **`locale.strxfrm()`** para una función clave o **`locale.strcoll()`** para una función de comparación. Esto es necesario porque la ordenación «alfabética» puede variar entre culturas aunque el alfabeto subyacente sea el mismo.

- El parámetro *reverse* aún mantiene estabilidad de ordenamiento (de modo que los registros con claves iguales conservan el orden original). Curiosamente, ese efecto se puede simular sin el parámetro utilizando la función incorporada `reversed()` dos veces:

```
>>> data = [('red', 1), ('blue', 1), ('red', 2), ('
>>> standard_way = sorted(data, key=itemgetter(0),
>>> double_reversed = list(reversed(sorted(reversed
>>> assert standard_way == double_reversed
>>> standard_way
[('red', 1), ('red', 2), ('blue', 1), ('blue', 2)]
```

- The sort routines use < when making comparisons between two objects. So, it is easy to add a standard sort order to a class by defining an `__lt__()` method:

```
>>> Student.__lt__ = lambda self, other: self.age <
>>> sorted(student_objects)
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A'
```

  However, note that < can fall back to using `__gt__()` if `__lt__()` is not implemented (see `object.__lt__()` for details on the mechanics). To avoid surprises, PEP 8 [https://peps.python.org/pep-0008/] recommends that all six comparison methods be implemented. The `total_ordering()` decorator is provided to make that task easier.

- Las funciones clave no necesitan depender directamente de los objetos que se ordenan. Una función clave también puede acceder a recursos externos. Por ejemplo, si las calificaciones de los estudiantes se almacenan en un diccionario, se pueden usar para ordenar una lista separada de nombres de estudiantes:

```
>>> students = ['dave', 'john', 'jane']
>>> newgrades = {'john': 'F', 'jane':'A', 'dave': '
>>> sorted(students, key=newgrades.__getitem__)
['jane', 'dave', 'john']
```

Partial Sorts

Some applications require only some of the data to be ordered. The standard library provides several tools that do less work than a full sort:

- **`min()`** and **`max()`** return the smallest and largest values, respectively. These functions make a single pass over the input data and require almost no auxiliary memory.
- **`heapq.nsmallest()`** and **`heapq.nlargest()`** return the *n* smallest and largest values, respectively. These functions make a single pass over the data keeping only *n* elements in memory at a time. For values of *n* that are small relative to the number of inputs, these functions make far fewer comparisons than a full sort.
- **`heapq.heappush()`** and **`heapq.heappop()`** create and maintain a partially sorted arrangement of data that keeps the smallest element at position `0`. These functions are suitable for implementing priority queues which are commonly used for task scheduling.

CÓMO (*HOWTO*) Unicode

Lanzamiento:

1.12

Este CÓMO (*HOWTO*) debate el soporte de Python para la especificación Unicode para representar datos textuales, y explica varios problemas que comúnmente encuentra la gente cuando tratan de trabajar con Unicode.

Introducción a Unicode

Definiciones

Los programas de hoy necesitan poder manejar una amplia variedad de caracteres. Las aplicaciones son a menudo internacionalizadas para mostrar mensajes y resultados en una variedad de idiomas seleccionables por el usuario; Es posible que el mismo programa necesite generar un mensaje de error en inglés, francés, japonés, hebreo o ruso. El contenido web se puede escribir en cualquiera de estos idiomas y también puede incluir una variedad de símbolos *emoji*. El tipo cadena de Python utiliza el estándar Unicode para representar caracteres, lo que permite a los programas de Python trabajar con todos estos caracteres posibles diferentes.

Unicode (https://www.unicode.org/) es una especificación que apunta a listar cada carácter usado por lenguajes humanos y darle a cada carácter su propio código único. La especificación Unicode es continuamente revisada y actualizada para añadir nuevos lenguajes y símbolos.

Un **carácter** es el componente mas pequeño posible de un texto. "A", "B", "C", etc., son todos diferentes caracteres. También lo son *"È"* e *"Í"*. Los caracteres varían dependiendo del lenguaje o del contexto en el que estás hablando. Por ejemplo, Existe un carácter para el «Número Uno Romano», “I”, que es distinto de la letra “I” mayúscula. Estos usualmente lucen igual, pero son dos caracteres diferentes que tienen distintos significados.

El estándar Unicode describe cómo se representan los caracteres mediante **puntos de código**. Un valor de punto de código es un número entero en el rango de 0 a 0x10FFFF (aproximadamente 1.1 millones de valores, el número real asignado [https://www.unicode.org/versions/latest/#Summary] es menor que eso). En el estándar y en este documento, un punto de código se escribe usando la notación `U+265E` para significar el carácter con valor `0x265e` (9,822 en decimal).

El estándar Unicode contiene muchas tablas que enumeran caracteres y sus puntos de código correspondientes:

```
0061    'a'; LATIN SMALL LETTER A
0062    'b'; LATIN SMALL LETTER B
0063    'c'; LATIN SMALL LETTER C
...
007B    '{'; LEFT CURLY BRACKET
...
2167    'Ⅷ'; ROMAN NUMERAL EIGHT
2168    'Ⅸ'; ROMAN NUMERAL NINE
...
265E    '♞'; BLACK CHESS KNIGHT
265F    '♟'; BLACK CHESS PAWN
...
1F600   '😀'; GRINNING FACE
1F609   '😉'; WINKING FACE
...
```

Estrictamente, estas definiciones implican que no tiene sentido decir “este es el carácter `U+265E`. `U+265E` es un punto de código, que representa algún carácter en particular; en este caso, representa el carácter “CABALLERO AJEDREZ NEGRO”, “♞”. En contextos informales, esta distinción entre puntos de código y caracteres a veces se olvidará.

Un carácter es representado en una pantalla o en papel por un conjunto de elementos gráficos llamado **glifo**. El glifo para una A mayúscula, por ejemplo, es dos trazos diagonales y uno horizontal, aunque los detalles exactos van a depender de la fuente utilizada. La mayoría del código de Python no necesita preocuparse por los glifos; averiguar el glifo correcto para mostrar es generalmente el trabajo de un kit de herramientas GUI o el renderizador de fuentes de una terminal.

Codificaciones

Para resumir la sección anterior: Una cadena Unicode es una secuencia de código de posiciones que son números desde 0 hasta `0x10FFFF` (1114111 decimal). Esta secuencia de código de posiciones necesita ser representada en memoria como un conjunto de **unidades de código**, y las **unidades de código** son mapeadas a bytes de 8 bits. Las reglas para traducir una cadena Unicode a una secuencia de bytes son llamadas **Codificación de carácter**, o sólo una **codificación**.

La primera codificación en que podrías pensar es usar enteros de 32 bits como unidad de código, y luego usar la representación de la CPU de enteros de 32 bits. En esta representación, la cadena «Python» podría verse así:

```
   P           y           t           h           o
0x50 00 00 00 79 00 00 00 74 00 00 00 68 00 00 00 6f 00
   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
```

Esta representación es sencilla pero utilizarla presenta una serie de problemas.

1. No es portable; diferentes procesadores ordenan los bytes de manera diferente.
2. Es un desperdicio de espacio. En la mayoría de los textos, la mayoría de los códigos de posición son menos de 127, o menos de 255, por lo que una gran cantidad de espacio está ocupado por bytes `0x00`. La cadena anterior toma 24 bytes en comparación con los 6 bytes necesarios para una representación ASCII. El aumento en el uso de RAM no importa demasiado (las computadoras de escritorio tienen

gigabytes de RAM, y las cadenas no suelen ser tan grandes), pero expandir nuestro uso del disco y el ancho de banda de la red en un factor de 4 es intolerable.
3. No es compatible con funciones existentes en C como `strlen()`, para eso se necesitaría una nueva familia de funciones de cadenas.

Por lo tanto esta codificación no es muy utilizada, y la gente prefiere elegir codificaciones que son mas eficientes y convenientes, como UTF-8.

UTF-8 es una de las codificaciones mas utilizadas, y Python generalmente la usa de forma predeterminada. UTF significa *«Unicode Transformation Format»*, y el «8» significa que se utilizan valores de 8 bits en la codificación. (También hay codificaciones UTF-16 y UTF-32, pero son menos frecuentes que UTF-8.) UTF-8 usa las siguientes reglas:

1. Si el código de posición is $<$ 128, es representado por el valor de byte correspondiente.
2. Si el código de posición es $>=$ 128, se transforma en una secuencia de dos, tres, o cuatro bytes, donde cada byte de la secuencia está entre 128 y 255.

UTF-8 tiene varias propiedades convenientes:

1. Puede manejar cualquier punto de código Unicode.
2. Una cadena Unicode se convierte en una secuencia de bytes que contiene cero bytes incrustados solo donde representan el carácter nulo (U+0000). Esto significa que las cadenas UTF-8 pueden ser procesadas por funciones C como `strcpy()` y enviadas a través de protocolos que no pueden manejar cero bytes para nada más que marcadores de fin de cadena de caracteres.
3. Una cadena de texto ASCII es también texto UTF-8.
4. UTF-8 es bastante compacto; La mayoría de los caracteres comúnmente usados pueden ser representados con uno o dos bytes.
5. Si los bytes están corruptos o perdidos, es posible determinar el comienzo del próximo código de posición y re-sincronizar. También es poco probable que datos aleatorios de 8 bit se

vean como UTF-8 válido.

6. UTF-8 es una codificación orientada a bytes. La codificación especifica que cada carácter está representado por una secuencia específica de uno o más bytes. Esto evita los problemas de ordenamiento de bytes que pueden ocurrir con codificaciones orientadas a números enteros y palabras, como UTF-16 y UTF-32, donde la secuencia de bytes varía según el hardware en el que se codificó la cadena.

Referencias

El sitio del Consorcio Unicode [https://www.unicode.org] tiene gráficos de caracteres, un glosario y versiones en PDF de la especificación Unicode. Esté preparado para una lectura difícil. Una cronología [https://www.unicode.org/history/] del origen y desarrollo de Unicode también está disponible en el sitio.

En el canal de *Youtube Computerphile*, *Tom Scott* discute brevemente la historia de Unicode y UTF-8 [https://www.youtube.com/watch?v=MijmeoH9LT4] (9 minutos 36 segundos).

To help understand the standard, Jukka Korpela has written an introductory guide [https://jkorpela.fi/unicode/guide.html] to reading the Unicode character tables.

Otro buen articulo introductorio [https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses/] fue escrito por *Joel Spolsky*. Si esta introducción no aclara las cosas para usted, debería tratar leyendo este articulo alternativo antes de continuar.

Artículos de *Wikipedia* son a menudo útiles. Mire los artículos para «codificación de caracteres [https://es.wikipedia.org/wiki/Codificaci%C3%B3n_de_caracteres]» y UTF-8 [https://es.wikipedia.org/wiki/UTF-8], por ejemplo.

Soporte Unicode de Python

Ahora que ya ha aprendido los rudimentos de Unicode, podemos mirar las características de Unicode de Python.

El tipo cadena

Desde Python 3.0, el tipo `str` del lenguaje contiene caracteres Unicode, lo que significa que cualquier cadena creada usando `"unicode rocks!"`, `'unicode rocks!'`, o la sintaxis de cadena entre comillas triples es almacenado como Unicode.

La codificación predeterminada para el código fuente de Python es UTF-8, por lo que simplemente puede incluir un carácter Unicode en un literal de cadena de caracteres:

```
try:
    with open('/tmp/input.txt', 'r') as f:
        ...
except OSError:
    # 'File not found' error message.
    print("Fichier non trouvé")
```

Nota al margen: Python 3 también soporta el uso de caracteres Unicode en identificadores:

```
répertoire = "/tmp/records.log"
with open(répertoire, "w") as f:
    f.write("test\n")
```

Si no puede ingresar un carácter en particular en su editor o desea mantener el código fuente solo ASCII por alguna razón, también puede usar secuencias de escape en cadenas de caracteres literales. (Dependiendo de su sistema, es posible que vea el glifo delta de mayúsculas en lugar de un escape u):

```
>>> "\N{GREEK CAPITAL LETTER DELTA}"  # Using the charac
'\u0394'
>>> "\u0394"                          # Using a 16-bit h
'\u0394'
>>> "\U00000394"                      # Using a 32-bit h
'\u0394'
```

Además, uno puede crear una cadena usando el método **`decode()`** de la clase **`bytes`**. Este método recibe una *codificación* como argumento, como `UTF-8`, y opcionalmente un argumento *errores*.

El argumento *errores* especifica la respuesta cuando la cadena ingresada no puede ser convertida de acuerdo a las reglas de codificación. Los posibles valores para este argumento son `'strict'` (levanta una excepción **`UnicodeDecodeError`**), `'replace'` (use `U+FFFD, CARACTER DE REEMPLAZO`), `'ignore'` (solo deje el carácter fuera del resultado Unicode), o `'backslahsreplace'` (inserta una secuencia de escape `\xNN`). Los siguientes ejemplos muestran las diferencias

```
>>> b'\x80abc'.decode("utf-8", "strict")
Traceback (most recent call last):
    ...
UnicodeDecodeError: 'utf-8' codec can't decode byte 0x80
  invalid start byte
>>> b'\x80abc'.decode("utf-8", "replace")
'\ufffdabc'
>>> b'\x80abc'.decode("utf-8", "backslashreplace")
'\\x80abc'
>>> b'\x80abc'.decode("utf-8", "ignore")
'abc'
```

Las codificaciones son especificadas como cadenas que contienen el nombre de la codificación. Python viene con cerca de 100 codificaciones diferentes; consulta la referencia de la biblioteca de Python en Codificaciones estándar para una lista. Algunas codificaciones tienen múltiples nombres; por ejemplo, `'latin-1'`, `'iso_8859_1'` y `'8859`” son sinónimos para la misma codificación.

Las cadenas de un solo carácter pueden ser creadas también con la función incorporada **`chr()`**, que toma un entero y retorna una cadena Unicode de longitud 1 que contiene el correspondiente código de posición. La operación inversa es la función incorporada **`ord()`** que toma una cadena Unicode de un carácter y retorna el código de posición:

```
>>> chr(57344)
'\ue000'
>>> ord('\ue000')
57344
```

Convirtiendo a Bytes

El método opuesto a `bytes.decode()` es `str.encode()`, que retorna una representación de `bytes` de la cadena Unicode, codificada en la codificación solicitada.

El parámetro *errores* es el mismo que el parámetro del método `decode()` pero soporta algunos manejadores mas.

El siguiente ejemplo muestra los diferentes resultados:

```
>>> u = chr(40960) + 'abcd' + chr(1972)
>>> u.encode('utf-8')
b'\xea\x80\x80abcd\xde\xb4'
>>> u.encode('ascii')
Traceback (most recent call last):
    ...
UnicodeEncodeError: 'ascii' codec can't encode character
  position 0: ordinal not in range(128)
>>> u.encode('ascii', 'ignore')
b'abcd'
>>> u.encode('ascii', 'replace')
b'?abcd?'
>>> u.encode('ascii', 'xmlcharrefreplace')
b'&#40960;abcd&#1972;'
>>> u.encode('ascii', 'backslashreplace')
b'\\ua000abcd\\u07b4'
>>> u.encode('ascii', 'namereplace')
b'\\N{YI SYLLABLE IT}abcd\\u07b4'
```

Las rutinas de bajo nivel para registrar y acceder a las codificaciones disponibles se encuentran en el módulo `codecs`. La implementación de nuevas codificaciones también requiere comprender el módulo **`codecs`**. Sin embargo, las funciones de codificación y decodificación retornadas por este módulo generalmente son de nivel más bajo de lo que es cómodo, y escribir nuevas codificaciones es una tarea especializada, por lo que el módulo no se cubrirá en este CÓMO.

Literales Unicode en código fuente Python

En el código fuente de Python, se pueden escribir puntos de código Unicode específicos utilizando la secuencia de escape `\u`, que es seguida por cuatro dígitos hexadecimales que dan el punto de código. La secuencia de escape `\U` es similar, pero espera ocho dígitos hexadecimales, no cuatro:

```
>>> s = "a\xac\u1234\u20ac\U00008000"
... #     ^^^^ two-digit hex escape
... #         ^^^^^^ four-digit Unicode escape
... #                     ^^^^^^^^^^ eight-digit Unicode
>>> [ord(c) for c in s]
[97, 172, 4660, 8364, 32768]
```

El uso de secuencias de escape para puntos de código superiores a 127 está bien en pequeñas dosis, pero se convierte en una molestia si está utilizando muchos caracteres acentuados, como lo haría en un programa con mensajes en francés o algún otro lenguaje que utilice acento. También puede ensamblar cadenas usando la función incorporada **`chr()`**, pero esto es aún más tedioso.

Idealmente, desearía poder escribir literales en la codificación natural de su idioma. Luego, puede editar el código fuente de Python con su editor favorito, que mostrará los caracteres acentuados de forma natural y tendrá los caracteres correctos utilizados en tiempo de ejecución.

Python soporta la escritura de código fuente en UTF-8 de forma predeterminada, pero puede usar casi cualquier codificación si declara la codificación que está utilizando. Esto se hace mediante la inclusión de un comentario especial en la primera o segunda línea del archivo fuente:

```
#!/usr/bin/env python
# -*- coding: latin-1 -*-
```

```
u = 'abcdé'
print(ord(u[-1]))
```

La sintaxis está inspirada en la notación de Emacs para especificar variables locales a un archivo. Emacs admite muchas variables diferentes, pero Python solo admite *“coding”*. Los símbolos `- * -` indican a Emacs que el comentario es especial; no tienen importancia para Python pero son una convención. Python busca `coding: name` o `coding=name` en el comentario.

Si no incluye dicho comentario, la codificación predeterminada utilizada será UTF-8 como ya se mencionó. Ver también **PEP 263** [https://peps.python.org/pep-0263/] para más información.

Propiedades Unicode

La especificación Unicode incluye una base de datos de información sobre puntos de código. Para cada punto de código definido, la información incluye el nombre del carácter, su categoría, el valor numérico si corresponde (para caracteres que representan conceptos numéricos como los números romanos, fracciones como un tercio y cuatro quintos, etc.). También hay propiedades relacionadas con la visualización, como cómo usar el punto de código en texto bidireccional.

El siguiente programa muestra información sobre varios caracteres e imprime el valor numérico de un carácter en particular:

```
import unicodedata

u = chr(233) + chr(0x0bf2) + chr(3972) + chr(6000) + chr

for i, c in enumerate(u):
    print(i, '%04x' % ord(c), unicodedata.category(c), e
    print(unicodedata.name(c))

# Get numeric value of second character
print(unicodedata.numeric(u[1]))
```

Cuando se ejecuta, este imprime:

```
0 00e9 Ll LATIN SMALL LETTER E WITH ACUTE
1 0bf2 No TAMIL NUMBER ONE THOUSAND
2 0f84 Mn TIBETAN MARK HALANTA
3 1770 Lo TAGBANWA LETTER SA
4 33af So SQUARE RAD OVER S SQUARED
1000.0
```

Los códigos de categoría son abreviaturas que describen la naturaleza del personaje. Estos se agrupan en categorías como «Letra», «Número», «Puntuación» o «Símbolo», que a su vez se dividen en subcategorías. Para tomar los códigos de la salida anterior, `'Ll'` significa "Letra, minúscula ", `'No'` significa «Número, otro», `'Mn'` es «Marca, sin espacios» , y `'So'` es «Símbolo, otro». Consulte la sección Valores de categoría generales de la documentación de la base de datos de caracteres Unicode [https://www.unicode.org/reports/tr44/#General_Category_Values] para obtener una lista de códigos de categoría.

Comparando cadenas

Unicode agrega algunas complicaciones a la comparación de cadenas, porque el mismo conjunto de caracteres puede representarse mediante diferentes secuencias de puntos de código. Por ejemplo, una letra como "ê" puede representarse como un único punto de código U+00EA, o como U+0065 U+0302, que es el punto de código para "e" seguido de un punto de código para "COMBINING CIRCUMFLEX ACCENT" . Estos producirán la misma salida cuando se impriman, pero uno es una cadena de longitud 1 y el otro es de longitud 2.

Una herramienta para una comparación que no distingue entre mayúsculas y minúsculas es el método `casefold()` que convierte una cadena en una forma que no distingue entre mayúsculas y minúsculas siguiendo un algoritmo descrito por el estándar Unicode. Este algoritmo tiene un manejo especial para caracteres como la letra Alemana "ß" (punto de código U+00DF), que se convierte en el par de letras minúsculas "ss".

```
>>> street = 'Gürzenichstraße'
>>> street.casefold()
'gürzenichstrasse'
```

A second tool is the `unicodedata` module's `normalize()` function that converts strings to one of several normal forms, where letters followed by a combining character are replaced with single characters. **`normalize()`** can be used to perform string comparisons that won't falsely report inequality if two strings use combining characters differently:

```
import unicodedata

def compare_strs(s1, s2):
    def NFD(s):
        return unicodedata.normalize('NFD', s)

    return NFD(s1) == NFD(s2)

single_char = 'ê'
multiple_chars = '\N{LATIN SMALL LETTER E}\N{COMBINING C
print('length of first string=', len(single_char))
print('length of second string=', len(multiple_chars))
print(compare_strs(single_char, multiple_chars))
```

Cuando se ejecuta, esto genera:

```
$ python compare-strs.py
length of first string= 1
length of second string= 2
True
```

El primer argumento para la función `normalize()` es una cadena que proporciona la forma de normalización deseada, que puede ser una de “NFC”, “NFKC”, “NFD” y “NFKD”.

El estándar Unicode también especifica cómo hacer comparaciones sin mayúsculas y minúsculas:

```
import unicodedata
```

```
def compare_caseless(s1, s2):
    def NFD(s):
        return unicodedata.normalize('NFD', s)

    return NFD(NFD(s1).casefold()) == NFD(NFD(s2).casefo

# Example usage
single_char = 'ê'
multiple_chars = '\N{LATIN CAPITAL LETTER E}\N{COMBINING

print(compare_caseless(single_char, multiple_chars))
```

This will print `True`. (Why is **`NFD()`** invoked twice? Because there are a few characters that make **`casefold()`** return a non-normalized string, so the result needs to be normalized again. See section 3.13 of the Unicode Standard for a discussion and an example.)

Expresiones regulares Unicode

Las expresiones regulares soportadas por el módulo **`re`** se pueden proporcionar como bytes o cadenas. Algunas de las secuencias de caracteres especiales como `\d` y `\w` tienen diferentes significados dependiendo de si el patrón se suministra como bytes o una cadena. Por ejemplo, `\d` coincidirá con los caracteres `[0-9]` en bytes, pero en las cadenas coincidirá con cualquier carácter que esté en la categoría `'Nd'`.

La cadena en este ejemplo tiene el número 57 escrito en números tailandeses y árabes:

```
import re
p = re.compile(r'\d+')

s = "Over \u0e55\u0e57 57 flavours"
m = p.search(s)
print(repr(m.group()))
```

Cuando se ejecuta, `\d+` coincidirá con los números tailandeses y los imprimirá. Si proporciona el indicador `re.ASCII` a `compile()`, `\d+` coincidirá con la subcadena «57» en su lugar.

Del mismo modo, `\w` coincide con una amplia variedad de caracteres Unicode pero solo `[a-zA-Z0-9_]` en bytes o si `re.ASCII` se suministra, y `\s` coincidirá con los caracteres de espacio en blanco Unicode o `[ \t\n\r\f\v]`.

Referencias

Algunas buenas discusiones alternativas sobre el soporte Unicode de Python son:

- Processing Text Files in Python 3 [https://python-notes.curiousefficiency.org/en/latest/python3/text_file_processing.html], by Nick Coghlan.
- Pragmatic Unicode [https://nedbatchelder.com/text/unipain.html], una presentación de *Ned Batchelder* en PyCon 2012.

El tipo `str` se describe en la referencia de la biblioteca de Python en Cadenas de caracteres — str.

La documentación para el módulo `unicodedata`.

La documentación para el módulo `codecs`.

Marc-André Lemburg hizo una presentación titulada «Python and Unicode» (diapositivas en PDF) [https://downloads.egenix.com/python/Unicode-EPC2002-Talk.pdf] en EuroPython 2002. Las diapositivas son una excelente descripción general del diseño de las características Unicode de Python 2 (donde el tipo de cadena Unicode se llama `unicode` y los literales comienzan con `u`).

Leyendo y escribiendo datos Unicode

Una vez que haya escrito un código que funcione con datos Unicode, el siguiente problema es la entrada/salida. ¿Cómo obtiene cadenas Unicode en su programa y cómo convierte Unicode en una forma adecuada para almacenamiento o transmisión?

Es posible que no necesite hacer nada dependiendo de sus fuentes de entrada y destinos de salida; debe verificar si las bibliotecas utilizadas en su aplicación son compatibles con Unicode de forma nativa. Los analizadores XML a menudo retornan datos Unicode, por ejemplo. Muchas bases de datos relacionales también admiten columnas con valores Unicode y pueden retornar valores Unicode de una consulta SQL.

Los datos Unicode generalmente se convierten a una codificación particular antes de escribirse en el disco o enviarse a través de un socket. Es posible hacer todo el trabajo usted mismo: abra un archivo, lea un objeto de bytes de 8 bits y convierta los bytes con `bytes.decode(codificación)`. Sin embargo, no se recomienda el enfoque manual.

Un problema es la naturaleza de múltiples bytes de las codificaciones; Un carácter Unicode puede ser representado por varios bytes. Si desea leer el archivo en fragmentos de tamaño arbitrario (por ejemplo, 1024 o 4096 bytes), debe escribir un código de manejo de errores para detectar el caso en el que solo una parte de los bytes que codifican un solo carácter Unicode se leen al final de Un trozo. Una solución sería leer todo el archivo en la memoria y luego realizar la decodificación, pero eso le impide trabajar con archivos que son extremadamente grandes; si necesita leer un archivo de 2 GB, necesita 2 GB de RAM. (Más, realmente, ya que por al menos un momento necesitarías tener tanto la cadena codificada como su versión Unicode en la memoria).

La solución sería utilizar la interfaz de decodificación de bajo nivel para detectar el caso de secuencias de codificación parcial. El trabajo de implementar esto ya se ha realizado para usted: la función incorporada **`open()`** puede retornar un objeto similar a un archivo que asume que el contenido del archivo está en una codificación especificada y acepta parámetros Unicode para métodos como **`read()`** y **`write()`**. Esto funciona a través de los parámetros *enconding* y *errors* de **`open()`** que se interpretan como los de **`str.encode()`** y **`bytes.decode()`**.

Por lo tanto, leer Unicode de un archivo es simple:

```
with open('unicode.txt', encoding='utf-8') as f:
    for line in f:
        print(repr(line))
```

También es posible abrir archivos en modo de actualización, lo que permite leer y escribir:

```
with open('test', encoding='utf-8', mode='w+') as f:
    f.write('\u4500 blah blah blah\n')
    f.seek(0)
    print(repr(f.readline()[:1]))
```

El carácter Unicode `U+FEFF` se usa como marca de orden de bytes (BOM), y a menudo se escribe como el primer carácter de un archivo para ayudar a la auto detección del orden de bytes del archivo. Algunas codificaciones, como UTF-16, esperan que haya una BOM al comienzo de un archivo; cuando se utiliza dicha codificación, la BOM se escribirá automáticamente como el primer carácter y se descartará en silencio cuando se lea el archivo. Existen variantes de estas codificaciones, como "utf-16-le" y "utf-16-be" para codificaciones "little-endian" y "big-endian", que especifican un orden de bytes particular y no omiten la BOM.

En algunas áreas, también es convencional usar una «BOM» al comienzo de los archivos codificados UTF-8; el nombre es engañoso ya que UTF-8 no depende del orden de bytes. La marca simplemente anuncia que el archivo está codificado en UTF-8. Para leer dichos archivos, use el códec "utf-8-sig" para omitir automáticamente la marca si está presente.

Nombres de archivos Unicode

La mayoría de los sistemas operativos de uso común en la actualidad admiten nombres de archivo que contienen caracteres Unicode arbitrarios. Por lo general, esto se implementa convirtiendo la cadena Unicode en una codificación que varía según el sistema. Hoy Python está convergiendo en el uso de UTF-8: Python en MacOS ha usado UTF-8 para varias versiones, y Python 3.6 también ha cambiado a usar UTF-8 en Windows. En los sistemas Unix, solo habrá un filesystem encoding. si ha configurado las variables de entorno `LANG` o `LC_CTYPE`; si no lo ha hecho, la codificación predeterminada es nuevamente UTF-8.

La función **`sys.getfilesystemencoding()`** retorna la codificación para usar en su sistema actual, en caso de que desee realizar la codificación manualmente, pero no hay muchas razones para molestarse. Al abrir un archivo para leer o escribir, generalmente puede proporcionar la cadena Unicode como nombre de archivo, y se convertirá automáticamente a la codificación correcta para usted:

```
filename = 'filename\u4500abc'
with open(filename, 'w') as f:
    f.write('blah\n')
```

Las funciones en el módulo **`os`** como **`os.stat()`** también aceptarán nombres de archivo Unicode.

La función **`os.listdir()`** retorna nombres de archivo, lo que plantea un problema: ¿debería devolver la versión Unicode de los nombres de archivo o debería devolver bytes que contienen las versiones codificadas? **`os.listdir()`** puede hacer ambas cosas, dependiendo de si proporcionó la ruta del directorio como bytes o una cadena Unicode. Si pasa una cadena Unicode como ruta, los nombres de archivo se decodificarán utilizando la codificación del sistema de archivos y se devolverá una lista de cadenas Unicode, mientras que pasar una ruta de bytes devolverá los nombres de archivo como bytes. Por ejemplo, suponiendo que el filesystem encoding predeterminado es UTF-8, al ejecutar el siguiente programa:

```
fn = 'filename\u4500abc'
f = open(fn, 'w')
f.close()

import os
print(os.listdir(b'.'))
print(os.listdir('.'))
```

producirá la siguiente salida:

```
$ python listdir-test.py
```

```
[b'filename\xe4\x94\x80abc', ...]
['filename\u4500abc', ...]
```

La primera lista contiene nombres de archivos codificados con UTF-8, y la segunda lista contiene las versiones Unicode.

Tenga en cuenta que en la mayoría de las ocasiones, debe seguir usando Unicode con estas API. Las API de bytes solo deben usarse en sistemas donde pueden estar presentes nombres de archivo no codificables; eso es prácticamente solo sistemas Unix ahora.

Consejos para escribir programas compatibles con Unicode

Esta sección proporciona algunas sugerencias sobre cómo escribir software que maneje Unicode.

El consejo más importante es:

> El software solo debería funcionar con cadenas Unicode internamente, decodificando los datos de entrada lo antes posible y codificando la salida solo al final.

Si intenta escribir funciones de procesamiento que acepten cadenas Unicode y de bytes, encontrará que su programa es vulnerable a errores dondequiera que combine los dos tipos diferentes de cadenas. No hay codificación o decodificación automática: si hace, por ejemplo: `str+bytes`, un **`TypeError`** se generará.

Cuando se usan datos que provienen de un navegador web u otra fuente no confiable, una técnica común es verificar si hay caracteres ilegales en una cadena antes de usar la cadena en una línea de comando generada o almacenarla en una base de datos. Si está haciendo esto, tenga cuidado de verificar la cadena decodificada, no los datos de bytes codificados; Algunas codificaciones pueden tener propiedades interesantes, como no ser biyectivo o no ser totalmente compatible con ASCII. Esto es especialmente cierto si los datos de entrada también especifican la codificación, ya que el atacante puede elegir una forma inteligente de ocultar el texto malicioso en el flujo de bytes codificado.

Conversión entre codificaciones de archivo

La clase **`StreamRecoder`** puede convertir de forma transparente entre codificaciones, tomar una secuencia que retorna datos en la codificación 1 y comportarse como una secuencia que retorna datos en la codificación 2.

Por ejemplo, si tiene un archivo de entrada *f* que está en Latin-1, puede envolverlo con **`StreamRecoder`** para retornar bytes codificados en UTF-8:

```
new_f = codecs.StreamRecoder(f,
    # en/decoder: used by read() to encode its results a
    # by write() to decode its input.
    codecs.getencoder('utf-8'), codecs.getdecoder('utf-8

    # reader/writer: used to read and write to the strea
    codecs.getreader('latin-1'), codecs.getwriter('latin
```

Archivos en una codificación desconocida

¿Qué puede hacer si necesita hacer un cambio en un archivo, pero no conoce la codificación del archivo? Si sabe que la codificación es compatible con ASCII y solo desea examinar o modificar las partes ASCII, puede abrir el archivo con el manejador de errores `surrogateescape`:

```
with open(fname, 'r', encoding="ascii", errors="surrogat
    data = f.read()

# make changes to the string 'data'

with open(fname + '.new', 'w',
          encoding="ascii", errors="surrogateescape") as
    f.write(data)
```

El manejador de errores `surrogateescape` decodificará los bytes que no sean ASCII como puntos de código en un rango especial que va desde U+DC80 a U+DCFF. Estos puntos de código volverán a convertirse en los mismos bytes cuando se use el controlador de error `subrogateescape` para codificar los datos y volver a escribirlos.

Referencias

One section of Mastering Python 3 Input/Output [https://pyvideo.org/video/289/pycon-2010--mastering-python-3-i-o], a PyCon 2010 talk by David Beazley, discusses text processing and binary data handling.

El PDF slides for Marc-André Lemburg's presentation «Writing Unicode-aware Applications in Python» [https://downloads.egenix.com/python/LSM2005-Developing-Unicode-aware-applications-in-Python.pdf] discute cuestiones de codificaciones de caracteres, así como también cómo internacionalizar y localizar una aplicación. Estas diapositivas cubren solo Python 2.x.

The Guts of Unicode in Python [https://pyvideo.org/video/1768/the-guts-of-unicode-in-python] is a PyCon 2013 talk by Benjamin Peterson that discusses the internal Unicode representation in Python 3.3.

Agradecimientos

El borrador inicial de este documento fue escrito por *Andrew Kuchling*. Desde entonces ha sido revisado por *Alexander Belopolsky, Georg Brandl, Andrew Kuchling y Ezio Melotti*.

293/5000 Gracias a las siguientes personas que notaron errores u ofrecieron sugerencias sobre este artículo: *Éric Araujo, Nicholas Bastin, Nick Coghlan, Marius Gedminas, Kent Johnson, Ken Krugler, Marc-André Lemburg, Martin von Löwis, Terry J. Reedy, Serhiy Storchaka , Eryk Sun, Chad Whitacre, Graham Wideman*.

[1] [2] [3] [4] [5] [6]

HOWTO - Cómo obtener recursos de Internet con el paquete urllib

Autor:

Michael Foord [https://agileabstractions.com/]

Introducción

Related Articles

También puedes encontrar útil el siguiente artículo sobre la obtención de recursos web con Python:

- Basic Authentication [https://web.archive.org/web/20201215133350/http://www.voidspace.org.uk/python/articles/authentication.shtml]

  Un tutorial sobre *Autenticación Básica*, con ejemplos en Python.

urllib.request es un módulo Python para acceder y utilizar recursos de internet identificados por URLs (*Uniform Resource Locators*). Ofrece una interfaz muy simple, a través de la función *urlopen*. Esta función es capaz de acceder a URLs usando una variedad de protocolos diferentes. También ofrece una interfaz un poco más compleja para manejar situaciones comunes - como la autenticación básica, cookies y proxies, entre otros. Estos son proporcionados por los llamados objetos de apertura y gestores.

urllib.request soporta la obtención de recursos identificados por URLs para muchos «esquemas de URL» (identificados por la cadena de texto ubicada antes del `":"` en el URL - por ejemplo `"ftp"` es el esquema de URL de `"ftp://python.org/"`) usando sus protocolos de red asociados (por ejemplo FTP, HTTP). Este tutorial se centra en el caso más común, HTTP.

Para situaciones sencillas *urlopen* es muy fácil de usar. Pero tan pronto como encuentres errores o casos no triviales al abrir URLs HTTP, necesitarás entender el Protocolo de Transferencia de Hipertexto. La referencia más completa y autorizada para HTTP es **RFC 2616** [https://datatracker.ietf.org/doc/html/rfc2616.html]. Este es un documento técnico y no pretende ser fácil de leer. Este HOWTO tiene como objetivo ilustrar el uso de la *urllib*, con suficientes detalles sobre HTTP para ayudarte a entenderlo. No pretende reemplazar los documentos **`urllib.request`**, pero es complementario a ellos.

Obtención de URLs

La forma más simple de usar urllib.request es la siguiente:

```
import urllib.request
with urllib.request.urlopen('http://python.org/') as res
   html = response.read()
```

Si deseas recuperar un recurso a partir de la URL y almacenarlo en una ubicación temporal, puede hacerlo a través de las funciones **`shutil.copyfileobj()`** y **`tempfile.NamedTemporaryFile()`**:

```
import shutil
import tempfile
import urllib.request

with urllib.request.urlopen('http://python.org/') as res
    with tempfile.NamedTemporaryFile(delete=False) as tm
        shutil.copyfileobj(response, tmp_file)

with open(tmp_file.name) as html:
    pass
```

Muchos usos de urllib serán así de sencillos (nótese que en lugar de una URL “http:” podríamos haber usado una URL que empezara por “ftp:”, “file:”, etc.). Sin embargo, el propósito de este tutorial es explicar los casos más complicados, concentrándose en el HTTP.

HTTP se basa en peticiones y respuestas - el cliente hace peticiones y los servidores envían respuestas. urllib.request refleja esto con un objeto `Request` que representa la petición HTTP que estás haciendo. En su forma más simple se crea un objeto Request que especifica la URL que se quiere obtener. Llamar a `urlopen` con este objeto Request retorna un objeto respuesta para la URL solicitada. Esta respuesta es un objeto tipo archivo, lo que significa que puedes por ejemplo llamar a `.read()` en la respuesta:

```
import urllib.request

req = urllib.request.Request('http://python.org/')
with urllib.request.urlopen(req) as response:
   the_page = response.read()
```

Tenga en cuenta que urllib.request utiliza la misma interfaz de Request para gestionar todos los esquemas de URL. Por ejemplo, puedes hacer una petición FTP de la siguiente manera:

```
req = urllib.request.Request('ftp://example.com/')
```

En el caso de HTTP, hay dos cosas adicionales que los objetos Request le permiten hacer: Primero, puede pasar datos para enviarlos al servidor. En segundo lugar, puede pasar información adicional («metadatos») *about* los datos o sobre la solicitud en sí, al servidor; esta información se envía como «encabezados» HTTP. Veamos cada uno de estos por turno.

Datos

A veces quieres enviar datos a una URL (a menudo la URL se referirá a un script CGI (Common Gateway Interface) u otra aplicación web). Con HTTP, esto se hace a menudo usando lo que se conoce como una petición **POST**. Esto es a menudo lo que su navegador hace cuando envías un formulario HTML que has rellenado en la web. No todos los POSTs tienen que provenir de formularios: puedes usar un POST para transmitir datos arbitrarios a tu propia aplicación. En el caso común de los formularios HTML, los datos tienen que ser codificados de forma estándar, y luego pasados al objeto Request como el argumento `data`. La codificación se hace usando una función de la biblioteca **`urllib.parse`**`.

```
import urllib.parse
import urllib.request

url = 'http://www.someserver.com/cgi-bin/register.cgi'
values = {'name' : 'Michael Foord',
          'location' : 'Northampton',
          'language' : 'Python' }

data = urllib.parse.urlencode(values)
data = data.encode('ascii') # data should be bytes
req = urllib.request.Request(url, data)
with urllib.request.urlopen(req) as response:
   the_page = response.read()
```

Ten en cuenta que a veces se requieren otras codificaciones (por ejemplo, para la carga de archivos desde formularios HTML - ver HTML Specification, Form Submission [https://www.w3.org/TR/REC-html40/interact/forms.html#h-17.13] para más detalles).

Si no pasas el argumento `data`, urllib usa una petición **GET**. Una de las formas en la que las peticiones GET y POST difieren entre sí es que las peticiones POST a menudo tienen «efectos secundarios»: cambian el estado del sistema de alguna manera (por ejemplo, haciendo una petición al sitio para que un centenar de spam chatarra sea entregado a tu dirección). Aunque el estándar HTTP deja claro que las solicitudes POST están *siempre* destinadas a causar efectos secundarios, y las solicitudes GET a *nunca* causar efectos secundarios, nada impide que una solicitud GET tenga efectos secundarios, ni que una solicitud POST no tenga efectos secundarios. Los datos también pueden ser pasados en una solicitud GET HTTP codificándolos en la propia URL.

Esto se hace de la siguiente manera:

```
>>> import urllib.request
>>> import urllib.parse
>>> data = {}
>>> data['name'] = 'Somebody Here'
>>> data['location'] = 'Northampton'
>>> data['language'] = 'Python'
>>> url_values = urllib.parse.urlencode(data)
>>> print(url_values)  # The order may differ from belo
name=Somebody+Here&language=Python&location=Northampton
>>> url = 'http://www.example.com/example.cgi'
>>> full_url = url + '?' + url_values
>>> data = urllib.request.urlopen(full_url)
```

Nota que la URL completa se crea añadiendo un `?` a la URL, seguido de los valores codificados.

Encabezados (Headers)

Discutiremos aquí un encabezado HTTP en particular, para ilustrar cómo agregar encabezados a su solicitud HTTP.

A algunos sitios web [1] no les gusta ser navegados por programas, o envían diferentes versiones a diferentes navegadores [2]. Por defecto urllib se identifica como `Python-urllib/x.y` (donde `x` y `y` son los números mayor y menor de la versión de Python, por ejemplo `Python-urllib/2.5`), lo que puede confundir el sitio, o simplemente no funcionar. La forma en que un navegador se identifica a sí mismo es a través del encabezado `User-Agent` [3]. Cuando creas un objeto Request puedes pasarle un diccionario de encabezados. El siguiente ejemplo hace la misma petición que arriba, pero se identifica como una versión de Internet Explorer [4].

```
import urllib.parse
import urllib.request

url = 'http://www.someserver.com/cgi-bin/register.cgi'
user_agent = 'Mozilla/5.0 (Windows NT 6.1; Win64; x64)'
values = {'name': 'Michael Foord',
          'location': 'Northampton',
          'language': 'Python' }
headers = {'User-Agent': user_agent}
```

```
data = urllib.parse.urlencode(values)
data = data.encode('ascii')
req = urllib.request.Request(url, data, headers)
with urllib.request.urlopen(req) as response:
   the_page = response.read()
```

La respuesta también tiene dos métodos útiles. Ver la sección de info y geturl que viene después de que echemos un vistazo a lo que pasa cuando las cosas van mal.

Gestión de excepciones

urlopen raises `URLError` when it cannot handle a response (though as usual with Python APIs, built-in exceptions such as `ValueError`, `TypeError` etc. may also be raised).

`HTTPError` is the subclass of `URLError` raised in the specific case of HTTP URLs.

Las clases de excepción se exportan desde el módulo `urllib.error`.

URLError

A menudo, URLError se genera porque no hay conexión de red (no se encuentra ruta al servidor especificado), o el servidor especificado no existe. En este caso, la excepción generada tendrá un atributo «reason», que es una tupla que contiene un código de error y un mensaje de error de texto.

por ejemplo

```
>>> req = urllib.request.Request('http://www.pretend_ser
>>> try: urllib.request.urlopen(req)
... except urllib.error.URLError as e:
...     print(e.reason)
...
(4, 'getaddrinfo failed')
```

HTTPError

Every HTTP response from the server contains a numeric «status code». Sometimes the status code indicates that the server is unable to fulfil the request. The default handlers will handle some of these responses for you (for example, if the response is a «redirection» that requests the client fetch the document from a different URL, urllib will handle that for you). For those it can't handle, urlopen will raise an `HTTPError`. Typical errors include "404" (page not found), "403" (request forbidden), and "401" (authentication required).

Vea la sección 10 de RFC 2616 [https://datatracker.ietf.org/doc/html/rfc2616.html] para una referencia sobre todos los códigos de error HTTP.

The `HTTPError` instance raised will have an integer "code" attribute, which corresponds to the error sent by the server.

Códigos de error

Debido a que los gestores por defecto gestionan redirecciones (códigos en el rango de 300), y que los códigos en el rango de 100–299 indican éxito, normalmente sólo verás códigos de error en el rango de 400–599.

`http.server.BaseHTTPRequestHandler.responses` is a useful dictionary of response codes that shows all the response codes used by RFC 2616 [https://datatracker.ietf.org/doc/html/rfc2616.html]. An excerpt from the dictionary is shown below

```
responses = {
    ...
    <HTTPStatus.OK: 200>: ('OK', 'Request fulfilled, doc
    ...
    <HTTPStatus.FORBIDDEN: 403>: ('Forbidden',
                                  'Request forbidden --
                                  'not help'),
    <HTTPStatus.NOT_FOUND: 404>: ('Not Found',
                                  'Nothing matches the g
    ...
    <HTTPStatus.IM_A_TEAPOT: 418>: ("I'm a Teapot",
                                    'Server refuses to b
                                    'it is a teapot'),
    ...
    <HTTPStatus.SERVICE_UNAVAILABLE: 503>: ('Service Una
                                            'The server
                                            'request due
    ...
    }
```

When an error is raised the server responds by returning an HTTP error code *and* an error page. You can use the **`HTTPError`** instance as a response on the page returned. This means that as well as the code attribute, it also has read, geturl, and info, methods as returned by the `urllib.response` module:

```
>>> req = urllib.request.Request('http://www.python.org/
>>> try:
...     urllib.request.urlopen(req)
... except urllib.error.HTTPError as e:
...     print(e.code)
...     print(e.read())
...
404
b'<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitio
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.d
  ...
  <title>Page Not Found</title>\n
  ...
```

Resumiéndolo

So if you want to be prepared for **`HTTPError`** *or* **`URLError`** there are two basic approaches. I prefer the second approach.

Número 1

```
from urllib.request import Request, urlopen
from urllib.error import URLError, HTTPError
req = Request(someurl)
try:
    response = urlopen(req)
except HTTPError as e:
    print('The server couldn\'t fulfill the request.')
    print('Error code: ', e.code)
except URLError as e:
    print('We failed to reach a server.')
    print('Reason: ', e.reason)
else:
    # everything is fine
```

Nota

The `except HTTPError` *must* come first, otherwise `except URLError` will *also* catch an **`HTTPError`**.

Número 2

```
from urllib.request import Request, urlopen
from urllib.error import URLError
req = Request(someurl)
try:
    response = urlopen(req)
except URLError as e:
    if hasattr(e, 'reason'):
        print('We failed to reach a server.')
        print('Reason: ', e.reason)
    elif hasattr(e, 'code'):
        print('The server couldn\'t fulfill the request.
        print('Error code: ', e.code)
else:
    # everything is fine
```

info y geturl

The response returned by urlopen (or the `HTTPError` instance) has two useful methods `info()` and `geturl()` and is defined in the module `urllib.response`.

- **geturl** - retorna la verdadera URL de la página obtenida. Esto es útil porque `urlopen` (o el objeto de apertura utilizado) puede haber seguido una redirección. El URL de la página obtenida puede no ser el mismo que el URL solicitado.
- **info** - retorna un objeto parecido a un diccionario que describe la página consultada, particularmente los encabezados enviados por el servidor. Actualmente es una instancia `http.client.HTTPMessage`.

Los encabezados típicos incluyen "Content-length", "Content-type" y así sucesivamente. Mira la Quick Reference to HTTP Headers [https://jkorpela.fi/http.html] para un listado útil de encabezados de HTTP con breves explicaciones de su significado y uso.

Objetos de Apertura (Openers) y Gestores (Handlers)

Cuando obtienes una URL utilizas una apertura (una instancia de la quizás confusamente llamada `urllib.request.OpenerDirector`). Normalmente hemos estado usando la apertura por defecto - a través de `urlopen` - pero puedes crear aperturas personalizadas. Las aperturas utilizan manejadores. Todo el «trabajo pesado» lo hacen los manejadores. Cada manejador sabe cómo abrir URLs para un esquema de URL particular (http, ftp, etc.), o cómo manejar un aspecto de la apertura de URLs, por ejemplo redirecciones HTTP o cookies HTTP.

Desearás crear objetos de apertura si deseas consultar URLs con gestores específicos instalados, por ejemplo para obtener un objeto de apertura que gestione cookies, o para obtener un objeto de apertura que no gestione redireccionamientos.

Para crear un objeto de apertura, debes instanciar un `OpenerDirector`, y luego llamar `.add_handler(some_handler_instance)` repetidamente.

Alternativamente, puedes usar `build_opener`, que es una función conveniente para crear objetos de apertura con una sola llamada a la función. `build_opener` añade varios gestores por defecto, pero proporciona una forma rápida de añadir más y/o sobrescribir los gestores por defecto.

Otros tipos de gestores que puedes querer permiten manejar proxies, autenticación, y otras situaciones comunes pero ligeramente especializadas.

`install_opener` puede ser usado para hacer que un objeto `opener` sea el objeto de apertura (global) por defecto. Esto significa que las llamadas a `urlopen` usarán el objeto de apertura que has instalado.

Los objetos de apertura tienen un método `open`, que puede ser llamado directamente para consultar urls de la misma manera que la función «urlopen»: no hay necesidad de llamar `install_opener`, excepto por conveniencia.

Autenticación básica

To illustrate creating and installing a handler we will use the `HTTPBasicAuthHandler`. For a more detailed discussion of this subject – including an explanation of how Basic Authentication works - see the Basic Authentication Tutorial [https://web.archive.org/web/20201215133350/http://www.voidspace.org.uk/python/articles/authentication.shtml].

Cuando se requiere la autenticación, el servidor envía un encabezado (así como el código de error 401) solicitando la autenticación. Esto especifica el esquema de autenticación y un “realm”. El encabezado tiene el siguiente aspecto: `WWW-Authenticate: SCHEME realm="REALM"`.

por ejemplo.

```
WWW-Authenticate: Basic realm="cPanel Users"
```

El cliente debe entonces volver a intentar la solicitud con el nombre y la contraseña apropiados para el realm incluido como encabezamiento en la solicitud. Esto es “autenticación básica”. Para simplificar este proceso podemos crear una instancia de `HTTPBasicAuthHandler` y un objeto de apertura para usar este manejador.

El `HTTPBasicAuthHandler` utiliza un objeto llamado administrador de contraseñas para gestionar el mapeo de URLs y realms con contraseñas y nombres de usuario. Si sabes cuál es el realm (por el encabezado de autenticación enviado por el servidor), entonces puedes usar un `HTTPPasswordMgr`. Frecuentemente a uno no le importa cuál es el realm. En ese caso, es conveniente usar «HTTPPasswordMgrWithDefaultRealm». Esto te permite especificar un nombre de usuario y una contraseña por defecto para una URL. Esto será suministrado en caso de que no proporciones una combinación alternativa para un realm específico. Lo indicamos proporcionando `None` como el argumento del realm al método `add_password`.

La URL de primer nivel es la primera URL que requiere autenticación. Las URLs «más profundas» que la URL que pasas a .add_password() también coincidirán.

```
# create a password manager
password_mgr = urllib.request.HTTPPasswordMgrWithDefault

# Add the username and password.
# If we knew the realm, we could use it instead of None.
top_level_url = "http://example.com/foo/"
password_mgr.add_password(None, top_level_url, username,

handler = urllib.request.HTTPBasicAuthHandler(password_m

# create "opener" (OpenerDirector instance)
opener = urllib.request.build_opener(handler)

# use the opener to fetch a URL
opener.open(a_url)

# Install the opener.
# Now all calls to urllib.request.urlopen use our opener
```

```
urllib.request.install_opener(opener)
```

Nota

In the above example we only supplied our `HTTPBasicAuthHandler` to `build_opener`. By default openers have the handlers for normal situations – `ProxyHandler` (if a proxy setting such as an **`http_proxy`** environment variable is set), `UnknownHandler`, `HTTPHandler`, `HTTPDefaultErrorHandler`, `HTTPRedirectHandler`, `FTPHandler`, `FileHandler`, `DataHandler`, `HTTPErrorProcessor`.

`top_level_url` es de hecho *o* una URL completa (incluyendo el componente del esquema "http:" y el nombre del host y opcionalmente el número de puerto) p.ej. `"http://example.com/"` *o* una «autoridad» (esto es, el nombre del host, incluyendo opcionalmente el número de puerto) por ejemplo `"example.com"` o `"example.com:8080"` (este último ejemplo incluye un número de puerto). La autoridad, si está presente, NO debe contener el componente «userinfo» - por ejemplo `"joe:password@example.com"` no es correcto.

Proxies

urllib detectará automáticamente tu configuración de proxy y la utilizará. Esto es a través de `ProxyHandler`, que es parte de la cadena de gestores normales cuando se detecta un ajuste de proxy. Normalmente esto es algo bueno, pero hay ocasiones en las que puede no ser útil [5]. Una forma de hacerlo es configurar nuestro propio `ProxyHandler`, sin proxies definidos. Esto se hace usando pasos similares a la configuración de un gestor Basic Authentication:

```
>>> proxy_support = urllib.request.ProxyHandler({})
>>> opener = urllib.request.build_opener(proxy_support)
>>> urllib.request.install_opener(opener)
```

Nota

Actualmente `urllib.request` *no* soporta la consulta de ubicaciones `https` a través de un proxy. Sin embargo, esto puede ser habilitado extendiendo urllib.request como se muestra en la receta [6].

Nota

`HTTP_PROXY` será ignorado si se establece una variable `REQUEST_METHOD`; ver la documentación en **``getproxies`()``**.

Sockets y capas

El soporte de Python para obtener recursos de la web funciona en capas. urllib utiliza la biblioteca **`http.client`**, que a su vez utiliza la biblioteca de sockets.

A partir de Python 2.3 se puede especificar cuánto tiempo debe esperar un socket para obtener una respuesta antes de que se agote el tiempo de espera. Esto puede ser útil en aplicaciones que tienen que consultar páginas web. Por defecto, el módulo socket no tiene *tiempo de espera* y puede colgarse. Actualmente, el tiempo de espera de la conexión no se expone en los niveles http.client o urllib.request. Sin embargo, puede establecerse el tiempo de espera por defecto de forma global para todas los sockets usando

```
import socket
import urllib.request

# timeout in seconds
timeout = 10
socket.setdefaulttimeout(timeout)

# this call to urllib.request.urlopen now uses the defau
# we have set in the socket module
req = urllib.request.Request('http://www.voidspace.org.u
response = urllib.request.urlopen(req)
```

Notas a pie de página

Este documento fue examinado y revisado por John Lee.

Google por ejemplo.

El rastreo de navegadores es una práctica muy mala para el diseño de sitios web - construir sitios usando estándares web es mucho más sensato. Desafortunadamente muchos sitios siguen enviando versiones diferentes a diferentes navegadores.

El agente de usuario para MSIE 6 es *“Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)”*

Para detalles de más encabezados de peticiones HTTP, ver Quick Reference to HTTP Headers [https://jkorpela.fi/http.html].

En mi caso tengo que usar un proxy para acceder a internet en el trabajo. Si intentas consultar URLs de *localhost* a través de este proxy, las bloquea. IE está configurado para usar el proxy, que urllib recoge. Para poder probar los scripts con un servidor localhost, tengo que evitar que urllib use el proxy.

urllib opener for SSL proxy (CONNECT method): ASPN Cookbook Recipe [https://code.activestate.com/recipes/456195-urrlib2-opener-for-ssl-proxy-connect-method/].

Tutorial de *argparse*

autor:

Tshepang Mbambo

Este tutorial pretende ser una leve introducción a `argparse`, el módulo de análisis (*parsing*) de línea de comandos recomendado en la biblioteca estándar de Python.

Nota

The standard library includes two other libraries directly related to command-line parameter processing: the lower level `optparse` module (which may require more code to configure for a given application, but also allows an application to request behaviors that `argparse` doesn't support), and the very low level `getopt` (which specifically serves as an equivalent to the **`getopt()`** family of functions available to C programmers). While neither of those modules is covered directly in this guide, many of the core concepts in `argparse` first originated in `optparse`, so some aspects of this tutorial will also be relevant to `optparse` users.

Conceptos

Vamos a mostrar el tipo de funcionalidad que vamos a explorar en este tutorial introductorio haciendo uso del comando **ls**:

```
$ ls
cpython  devguide  prog.py  pypy  rm-unused-function.pat
$ ls pypy
ctypes_configure  demo  dotviewer  include  lib_pypy  li
$ ls -l
total 20
```

```
drwxr-xr-x 19 wena wena 4096 Feb 18 18:51 cpython
drwxr-xr-x  4 wena wena 4096 Feb  8 12:04 devguide
-rwxr-xr-x  1 wena wena  535 Feb 19 00:05 prog.py
drwxr-xr-x 14 wena wena 4096 Feb  7 00:59 pypy
-rw-r--r--  1 wena wena  741 Feb 18 01:01 rm-unused-func
$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory 
Sort entries alphabetically if none of -cftuvSUX nor --s
...
```

Algunos conceptos que podemos aprender de los cuatro comandos:

- El comando **ls** es útil cuando se ejecuta sin ninguna opción en absoluto. Por defecto muestra el contenido del directorio actual.
- Si queremos hacer algo, mas allá de lo que provee por defecto, le contamos un poco mas. En este caso, queremos mostrar un directorio diferente, `pypy`. Lo que hicimos fue especificar lo que se conoce como argumento posicional. Se llama así porque el programa debe saber que hacer con ese valor, basado únicamente en función de donde aparece en la línea de comandos. Este concepto es mas relevante para un comando como **cp**, cuyo uso mas básico es `cp SRC DEST`. La primer posición es *lo que quieres copiar*, y la segunda posición es *a donde lo quieres copiar*.
- Ahora, digamos que queremos cambiar el comportamiento del programa. En nuestro ejemplo, mostramos mas información para cada archivo en lugar de solo mostrar los nombres de los archivos. El argumento `-l` en ese caso se conoce como argumento opcional.
- Este es un fragmento del texto de ayuda. Es muy útil porque puedes encontrar un programa que nunca has usado antes, y puedes darte cuenta de como funciona simplemente leyendo el texto de ayuda.

Las bases

Comencemos con un simple ejemplo, el cual no hace (casi) nada:

```
import argparse
parser = argparse.ArgumentParser()
parser.parse_args()
```

Lo siguiente es el resultado de ejecutar el código:

```
$ python prog.py
$ python prog.py --help
usage: prog.py [-h]

options:
  -h, --help  show this help message and exit
$ python prog.py --verbose
usage: prog.py [-h]
prog.py: error: unrecognized arguments: --verbose
$ python prog.py foo
usage: prog.py [-h]
prog.py: error: unrecognized arguments: foo
```

Esto es lo que está pasando:

- Ejecutar el script sin ninguna opción da como resultado que no se muestra nada en *stdout*. No es tan útil.
- El segundo comienza a mostrar la utilidad del módulo **`argparse`**. No hemos hecho casi nada, pero ya recibimos un buen mensaje de ayuda.
- La opción `--help`, que también puede ser abreviada como `-h`, es la única opción que tenemos gratis (es decir, no necesitamos especificarla). Especificar cualquier otra cosa da como resultado un error. Pero aún así, recibimos un mensaje útil, también gratis.

Introducción a los argumentos posicionales

Un ejemplo:

```
import argparse
parser = argparse.ArgumentParser()
```

```
parser.add_argument("echo")
args = parser.parse_args()
print(args.echo)
```

Y ejecutando el código:

```
$ python prog.py
usage: prog.py [-h] echo
prog.py: error: the following arguments are required: ec
$ python prog.py --help
usage: prog.py [-h] echo

positional arguments:
  echo

options:
  -h, --help  show this help message and exit
$ python prog.py foo
foo
```

Aquí está lo que está sucediendo:

- Hemos agregado el método **`add_argument()`**, el cual es el que usamos para especificar cuales opciones de la línea de comandos el programa está dispuesto a aceptar. En este caso, lo he llamado `echo` para que concuerde con su función.
- Llamar nuestro programa ahora requiere que especifiquemos una opción.
- El método **`parse_args()`** retorna de hecho algunos datos de las opciones especificadas, en este caso, `echo`.
- La variable es una forma de “magia” que **`argparse`** se realiza de forma gratuita (es decir, no es necesario especificar en qué variable se almacena ese valor). También notará que su nombre coincide con el argumento de cadena dado al método, `echo`.

Sin embargo, tenga en cuenta que, aunque la pantalla de ayuda luce bien y todo, en realidad no es tan útil como podría ser. Por ejemplo, vemos que tenemos `echo` como un argumento posicional, pero no sabemos lo que hace, de otra manera que no sea adivinar o leer el código fuente. Entonces, vamos a hacerlo un poco mas útil:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("echo", help="echo the string you us
args = parser.parse_args()
print(args.echo)
```

Y la salida:

```
$ python prog.py -h
usage: prog.py [-h] echo

positional arguments:
  echo        echo the string you use here

options:
  -h, --help  show this help message and exit
```

Ahora, que tal si hacemos algo más útil:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", help="display a square of
args = parser.parse_args()
print(args.square**2)
```

Lo siguiente es el resultado de ejecutar el código:

```
$ python prog.py 4
Traceback (most recent call last):
  File "prog.py", line 5, in <module>
    print(args.square**2)
TypeError: unsupported operand type(s) for ** or pow():
```

Eso no fue tan bien. Esto es porque **`argparse`** trata las opciones que le damos como cadenas, a menos que le digamos otra cosa. Entonces, vamos a llamar a **`argparse`** para tratar esa entrada como un entero:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", help="display a square of
                    type=int)
args = parser.parse_args()
print(args.square**2)
```

Lo siguiente es el resultado de ejecutar el código:

```
$ python prog.py 4
16
$ python prog.py four
usage: prog.py [-h] square
prog.py: error: argument square: invalid int value: 'fou
```

Eso fue bien. El programa ahora aún se cierra útilmente en caso de una entrada ilegal incorrecta antes de proceder.

Introducción a los argumentos opcionales

Hasta ahora hemos estado jugando con argumentos posicionales. Vamos a darle una mirada a como agregar los opcionales:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("--verbosity", help="increase output
args = parser.parse_args()
if args.verbosity:
    print("verbosity turned on")
```

Y la salida:

```
$ python prog.py --verbosity 1
verbosity turned on
$ python prog.py
$ python prog.py --help
usage: prog.py [-h] [--verbosity VERBOSITY]

options:
  -h, --help            show this help message and exit
  --verbosity VERBOSITY
                        increase output verbosity
$ python prog.py --verbosity
usage: prog.py [-h] [--verbosity VERBOSITY]
prog.py: error: argument --verbosity: expected one argum
```

Esto es lo que está pasando:

- El programa está escrito para mostrar algo cuando `--verbosity` sea especificado y no mostrar nada cuando no.
- Para mostrar que la opción es de hecho opcional, no hay ningún error al ejecutar el programa sin ella. Tenga en cuenta que por defecto, si un argumento opcional no es usado, la variable relevante, en este caso `args.verbosity`, se le da `None` como valor, razón por la cual falla la prueba de verdad del **`if`**.
- El mensaje de ayuda es un poco diferente.
- Cuando usamos la opción `--verbosity`, también se debe especificar un valor, cualquier valor.

El ejemplo anterior acepta arbitrariamente valores enteros para `--verbosity`, pero para nuestro simple programa, solo dos valores son realmente útiles, `True` o `False`. Modifiquemos el código de acuerdo a esto:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("--verbose", help="increase output v
                    action="store_true")
args = parser.parse_args()
if args.verbose:
    print("verbosity turned on")
```

Y la salida:

```
$ python prog.py --verbose
verbosity turned on
$ python prog.py --verbose 1
usage: prog.py [-h] [--verbose]
```

```
prog.py: error: unrecognized arguments: 1
$ python prog.py --help
usage: prog.py [-h] [--verbose]

options:
  -h, --help  show this help message and exit
  --verbose   increase output verbosity
```

Esto es lo que está pasando:

- La opción ahora es más una bandera que algo que requiere un valor. Incluso cambiamos el nombre de la opción para que coincida con esa idea. Tenga en cuenta que ahora especificamos una nueva palabra clave, `action`, y le dimos el valor `"store_true"`. Esto significa que, si la opción es especificada, se asigna el valor `True` a `args.verbose`. No especificarlo implica `False`.
- Se queja cuando se especifica un valor, en verdadero espíritu de lo que realmente son los flags.
- Observe los diferentes textos de ayuda.

Opciones cortas

Si estas familiarizado con el uso de la línea de comandos, podrás observar que aún no he tocado el tema de las versiones cortas de las opciones. Es bastante simple:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("-v", "--verbose", help="increase ou
                    action="store_true")
args = parser.parse_args()
if args.verbose:
    print("verbosity turned on")
```

Y aquí va:

```
$ python prog.py -v
verbosity turned on
$ python prog.py --help
```

```
usage: prog.py [-h] [-v]

options:
  -h, --help     show this help message and exit
  -v, --verbose  increase output verbosity
```

Tenga en cuenta que la nueva habilidad es también reflejada en el texto de ayuda.

Combinar argumentos opcionales y posicionales

Nuestro programa sigue creciendo en complejidad:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given nu
parser.add_argument("-v", "--verbose", action="store_tru
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbose:
    print(f"the square of {args.square} equals {answer}"
else:
    print(answer)
```

Y ahora la salida:

```
$ python prog.py
usage: prog.py [-h] [-v] square
prog.py: error: the following arguments are required: sq
$ python prog.py 4
16
$ python prog.py 4 --verbose
the square of 4 equals 16
$ python prog.py --verbose 4
the square of 4 equals 16
```

- Hemos traído de vuelta un argumento posicional, de ahí la queja.
- Tenga en cuenta que el orden no importa.

Que tal si le retornamos a nuestro programa la capacidad de tener múltiples valores de verbosidad, y realmente usarlos:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given nu
parser.add_argument("-v", "--verbosity", type=int,
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity == 2:
    print(f"the square of {args.square} equals {answer}"
elif args.verbosity == 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

Y la salida:

```
$ python prog.py 4
16
$ python prog.py 4 -v
usage: prog.py [-h] [-v VERBOSITY] square
prog.py: error: argument -v/--verbosity: expected one ar
$ python prog.py 4 -v 1
4^2 == 16
$ python prog.py 4 -v 2
the square of 4 equals 16
$ python prog.py 4 -v 3
16
```

Todos estos se ven bien, excepto el último, que expone un error en nuestro programa. Corrijamos esto restringiendo los valores que la opción `--verbosity` puede aceptar:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given nu
parser.add_argument("-v", "--verbosity", type=int, choic
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity == 2:
    print(f"the square of {args.square} equals {answer}"
elif args.verbosity == 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

Y la salida:

```
$ python prog.py 4 -v 3
usage: prog.py [-h] [-v {0,1,2}] square
prog.py: error: argument -v/--verbosity: invalid choice:
$ python prog.py 4 -h
usage: prog.py [-h] [-v {0,1,2}] square

positional arguments:
  square                display a square of a given numb

options:
  -h, --help            show this help message and exit
  -v, --verbosity {0,1,2}
                        increase output verbosity
```

Tenga en cuenta que el cambio se refleja tanto en el mensaje de error como en la cadena de ayuda.

Ahora, usemos un enfoque diferente para jugar con la verbosidad, lo cual es bastante común. También coincide con la forma en que el ejecutable de CPython maneja su propio argumento de verbosidad (verifique el resultado de `python --help`):

```
import argparse
```

```
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display the square of a given
parser.add_argument("-v", "--verbosity", action="count",
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity == 2:
    print(f"the square of {args.square} equals {answer}"
elif args.verbosity == 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

Hemos introducido otra acción, «count», para contar el número de apariciones de opciones específicas.

```
$ python prog.py 4
16
$ python prog.py 4 -v
4^2 == 16
$ python prog.py 4 -vv
the square of 4 equals 16
$ python prog.py 4 --verbosity --verbosity
the square of 4 equals 16
$ python prog.py 4 -v 1
usage: prog.py [-h] [-v] square
prog.py: error: unrecognized arguments: 1
$ python prog.py 4 -h
usage: prog.py [-h] [-v] square

positional arguments:
  square           display a square of a given number

options:
  -h, --help       show this help message and exit
  -v, --verbosity  increase output verbosity
$ python prog.py 4 -vvv
16
```

- Si, ahora es mas una bandera (similar a `action="store_true"`) en la versión anterior de nuestro script. Esto debería explicar la queja.
- También se comporta de manera similar a la acción `"store_true"`.
- Ahora aquí una demostración de lo que la acción `"count"` da. Probablemente haya visto esta clase de uso antes.
- Y si no especificas la bandera `-v`, se considera que esa bandera tiene el valor `None`.
- Como debería esperarse, especificando la forma larga de la bandera, obtendríamos el mismo resultado.
- Lamentablemente, nuestra salida de ayuda no es muy informativa sobre la nueva capacidad que ha adquirido nuestro script, pero eso siempre se puede solucionar mejorando la documentación de nuestro script (por ejemplo, a través del argumento de la palabra clave `help`).
- La última salida expone un error en nuestro programa.

Vamos a arreglarlo:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given nu
parser.add_argument("-v", "--verbosity", action="count",
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2

# bugfix: replace == with >=
if args.verbosity >= 2:
    print(f"the square of {args.square} equals {answer}"
elif args.verbosity >= 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

Y esto es lo que da:

```
$ python prog.py 4 -vvv
```

```
the square of 4 equals 16
$ python prog.py 4 -vvvv
the square of 4 equals 16
$ python prog.py 4
Traceback (most recent call last):
  File "prog.py", line 11, in <module>
    if args.verbosity >= 2:
TypeError: '>=' not supported between instances of 'None
```

- La primer salida fue correcta, y corrigió el error que teníamos antes. Es decir, queremos que cualquier valor > = 2 sea lo más detallado posible.
- Tercer salida no tan buena.

Vamos a arreglar ese error:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given nu
parser.add_argument("-v", "--verbosity", action="count",
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity >= 2:
    print(f"the square of {args.square} equals {answer}"
elif args.verbosity >= 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

Acabamos de introducir otra palabra clave, `default`. Lo hemos configurado en `0` para que sea comparable con otros valores int. Recuerde que por defecto, si un argumento opcional no es especificado, obtiene el valor `None`, y eso no puede ser comparado con un valor int (de ahí la excepción **`TypeError`**).

Y:

```
$ python prog.py 4
```

Tu puedes llegar bastante lejos con lo que hemos aprendido hasta ahora, y solo arañado la superficie. El módulo **`argparse`** es muy poderoso, y exploraremos un poco mas antes de finalizar este tutorial.

Un poco mas avanzado

Qué pasaría si quisiéramos expandir nuestro pequeño programa para que tenga otros poderes, no solo cuadrados:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
parser.add_argument("-v", "--verbosity", action="count",
args = parser.parse_args()
answer = args.x**args.y
if args.verbosity >= 2:
    print(f"{args.x} to the power {args.y} equals {answe
elif args.verbosity >= 1:
    print(f"{args.x}^{args.y} == {answer}")
else:
    print(answer)
```

Salida:

```
$ python prog.py
usage: prog.py [-h] [-v] x y
prog.py: error: the following arguments are required: x,
$ python prog.py -h
usage: prog.py [-h] [-v] x y

positional arguments:
  x                  the base
  y                  the exponent

options:
  -h, --help       show this help message and exit
  -v, --verbosity
$ python prog.py 4 2 -v
4^2 == 16
```

Tenga en cuenta que hasta ahora hemos estado usando el nivel de verbosidad para *cambiar* el texto que se muestra. El siguiente ejemplo en lugar de usar nivel de verbosidad para mostrar *mas* texto en su lugar:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
parser.add_argument("-v", "--verbosity", action="count",
args = parser.parse_args()
answer = args.x**args.y
if args.verbosity >= 2:
    print(f"Running '{__file__}'")
if args.verbosity >= 1:
    print(f"{args.x}^{args.y} == ", end="")
print(answer)
```

Salida:

```
$ python prog.py 4 2
16
$ python prog.py 4 2 -v
4^2 == 16
$ python prog.py 4 2 -vv
Running 'prog.py'
4^2 == 16
```

Especificando argumentos ambiguos

Cuando hay ambigüedad al decidir si un argumento es posicional o para un argumento, `--` se puede usar para indicar **`parse_args()`** que todo lo que sigue a eso es un argumento posicional:

```
>>> parser = argparse.ArgumentParser(prog='PROG')
>>> parser.add_argument('-n', nargs='+')
>>> parser.add_argument('args', nargs='*')

>>> # ambiguo, parse_args asume que es una  opción
>>> parser.parse_args(['-f'])
usage: PROG [-h] [-n N [N ...]] [args ...]
PROG: error: unrecognized arguments: -f

>>> parser.parse_args(['--', '-f'])
Namespace(args=['-f'], n=None)

>>> # ambiguo, la opción -n option acepta argumentos ávi
>>> parser.parse_args(['-n', '1', '2', '3'])
Namespace(args=[], n=['1', '2', '3'])

>>> parser.parse_args(['-n', '1', '--', '2', '3'])
Namespace(args=['2', '3'], n=['1'])
```

Opciones conflictivas

Hasta ahora, hemos estado trabajando con dos métodos de una instancia de **`argparse.ArgumentParser`**. Vamos a introducir un tercer método, **`add_mutually_exclusive_group()`**. Nos permite especificar opciones que entran en conflicto entre sí. Cambiemos también el resto del programa para que la nueva funcionalidad tenga mas sentido: presentamos la opción `--quiet`, la cual es lo opuesto a la opción `--verbose`:

```
import argparse

parser = argparse.ArgumentParser()
group = parser.add_mutually_exclusive_group()
group.add_argument("-v", "--verbose", action="store_true
group.add_argument("-q", "--quiet", action="store_true")
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
args = parser.parse_args()
answer = args.x**args.y
```

```
if args.quiet:
    print(answer)
elif args.verbose:
    print(f"{args.x} to the power {args.y} equals {answe
else:
    print(f"{args.x}^{args.y} == {answer}")
```

Nuestro programa ahora es mas simple, y perdimos algunas funcionalidades en aras de la demostración. De todos modos, aquí esta el resultado:

```
$ python prog.py 4 2
4^2 == 16
$ python prog.py 4 2 -q
16
$ python prog.py 4 2 -v
4 to the power 2 equals 16
$ python prog.py 4 2 -vq
usage: prog.py [-h] [-v | -q] x y
prog.py: error: argument -q/--quiet: not allowed with ar
$ python prog.py 4 2 -v --quiet
usage: prog.py [-h] [-v | -q] x y
prog.py: error: argument -q/--quiet: not allowed with ar
```

Esto debería ser sencillo de seguir. He agregado esa última salida para que se pueda ver el tipo de flexibilidad que obtiene, es decir, mezclar opciones de forma larga con opciones de forma corta.

Antes de concluir, probablemente quiera contarle a sus usuarios el propósito principal de su programa, solo en caso de que no lo supieran:

```
import argparse

parser = argparse.ArgumentParser(description="calculate
group = parser.add_mutually_exclusive_group()
group.add_argument("-v", "--verbose", action="store_true
group.add_argument("-q", "--quiet", action="store_true")
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
```

```
args = parser.parse_args()
answer = args.x**args.y

if args.quiet:
    print(answer)
elif args.verbose:
    print(f"{args.x} to the power {args.y} equals {answe
else:
    print(f"{args.x}^{args.y} == {answer}")
```

Tenga en cuenta la ligera diferencia en el uso del texto. Tenga en cuenta `[-v | -q]`, lo cual nos indica que podemos usar `-v` o `-q`, pero no ambos al mismo tiempo:

```
$ python prog.py --help
usage: prog.py [-h] [-v | -q] x y

calculate X to the power of Y

positional arguments:
  x              the base
  y              the exponent

options:
  -h, --help     show this help message and exit
  -v, --verbose
  -q, --quiet
```

Cómo traducir la salida de argparse

La salida del módulo **`argparse`**, como su texto de ayuda y mensajes de error, se pueden traducir usando el módulo **`gettext`**. Esto permite que las aplicaciones localicen fácilmente los mensajes producidos por **`argparse`**. Consulte Internacionalizando sus programas y módulos para más información.

Por ejemplo, en esta salida **`argparse`**:

```
$ python prog.py --help
```

```
usage: prog.py [-h] [-v | -q] x y

calculate X to the power of Y

positional arguments:
  x               the base
  y               the exponent

options:
  -h, --help      show this help message and exit
  -v, --verbose
  -q, --quiet
```

Las cadenas de caracteres `usage:`, `positional arguments:`, `options:` y `` show this help message and exit` `` son todas traducibles.

Para traducir estas cadenas de caracteres, primero se deben extraer en un archivo `.po`. Por ejemplo, usando Babel [https://babel.pocoo.org/], ejecute este comando:

```
$ pybabel extract -o messages.po /usr/lib/python3.12/arg
```

Este comando extraerá todas las cadenas de caracteres traducibles del módulo `argparse` y las generará en un archivo llamado `messages.po`. Este comando asume que su instalación de Python está en `/usr/lib`.

Puede encontrar la ubicación del módulo `argparse` en su sistema usando este script:

```
import argparse
print(argparse.__file__)
```

Una vez que los mensajes en el archivo `.po` estén traducidos y las traducciones estén instaladas usando `gettext`, `argparse` podrá mostrar los mensajes traducidos.

Para traducir sus propias cadenas de caracteres en la salida `argparse`, use `gettext`.

Custom type converters

The **`argparse`** module allows you to specify custom type converters for your command-line arguments. This allows you to modify user input before it's stored in the **`argparse.Namespace`**. This can be useful when you need to pre-process the input before it is used in your program.

When using a custom type converter, you can use any callable that takes a single string argument (the argument value) and returns the converted value. However, if you need to handle more complex scenarios, you can use a custom action class with the **action** parameter instead.

For example, let's say you want to handle arguments with different prefixes and process them accordingly:

```
import argparse

parser = argparse.ArgumentParser(prefix_chars='-+')

parser.add_argument('-a', metavar='<value>', action='app
                    type=lambda x: ('-', x))
parser.add_argument('+a', metavar='<value>', action='app
                    type=lambda x: ('+', x))

args = parser.parse_args()
print(args)
```

Salida:

```
$ python prog.py -a value1 +a value2
Namespace(a=[('-', 'value1'), ('+', 'value2')])
```

In this example, we:

- Created a parser with custom prefix characters using the `prefix_chars` parameter.
- Defined two arguments, `-a` and `+a`, which used the `type` parameter to create custom type converters to store the value

in a tuple with the prefix.

Without the custom type converters, the arguments would have treated the `-a` and `+a` as the same argument, which would have been undesirable. By using custom type converters, we were able to differentiate between the two arguments.

Conclusión

El módulo **`argparse`** ofrece mucho más que solo lo mostrado aquí. Su documentación es bastante detallada y completa, y está llena de ejemplos. Habiendo seguido este tutorial, debe digerirlos fácilmente sin sentirse abrumado.

Introducción al modulo *ipaddress*

autor:

Peter Moody

autor:

Nick Coghlan

Descripción

Este documento tiene como objetivo proporcionar una introducción apacible al módulo `ipaddress`. Está dirigido principalmente a los usuarios que no están familiarizados con la terminología IP de redes, pero también puede ser útil para los ingenieros de red que quieren una visión general de cómo `ipaddress` representa los conceptos de direccionamiento IP de red.

Creando objetos Dirección/Red/Interfaz (*Address/Network/Interface*)

Siendo `ipaddress` un módulo para inspeccionar y manipular direcciones IP, la primer cosa que usted querrá hacer es crear algunos objetos. Puede utilizar `ipaddress` para crear objetos a partir de cadenas de caracteres y enteros.

Nota sobre versiones IP

Para los lectores que no están particularmente familiarizados con el direccionamiento IP, es importante saber que el Protocolo de Internet (IP) se encuentra actualmente en el proceso de pasar de la versión 4 del protocolo a la versión 6. Esta transición se produce en gran parte porque la versión 4 del protocolo no proporciona suficientes direcciones para satisfacer las necesidades de todo el mundo, especialmente dado el creciente número de dispositivos con conexiones directas a Internet.

Explicando los detalles de las diferencias entre las dos versiones del protocolo está más allá del alcance de esta introducción, pero los lectores deben al menos ser conscientes de que estas dos versiones existen, y a veces será necesario forzar el uso de una versión u otra.

Direcciones IP del Host

Las direcciones, a menudo denominadas «direcciones de host» son la unidad más básica cuando se trabaja con direccionamiento IP. La forma más sencilla de crear direcciones es usar la función de fabrica **`ipaddress.ip_address()`**, que determina automáticamente si se crea una dirección IPv4 o IPv6 en función del valor pasado:

```
>>> ipaddress.ip_address('192.0.2.1')
IPv4Address('192.0.2.1')
>>> ipaddress.ip_address('2001:DB8::1')
IPv6Address('2001:db8::1')
```

Las direcciones también se pueden crear directamente a partir de enteros. Los valores que caben dentro de 32 bits se asume que son direcciones IPv4:

```
>>> ipaddress.ip_address(3221225985)
IPv4Address('192.0.2.1')
>>> ipaddress.ip_address(42540766411282592856903984951165
IPv6Address('2001:db8::1')
```

Para forzar el uso de direcciones IPv4 o IPv6, las clases relevantes se pueden invocar directamente. Esto es particularmente útil para forzar la creación de direcciones IPv6 para enteros pequeños:

```
>>> ipaddress.ip_address(1)
IPv4Address('0.0.0.1')
>>> ipaddress.IPv4Address(1)
IPv4Address('0.0.0.1')
>>> ipaddress.IPv6Address(1)
IPv6Address('::1')
```

Definiendo redes

Las direcciones de host generalmente se agrupan en redes IP, por lo que **`ipaddress`** proporciona una forma de crear, inspeccionar y manipular definiciones de red. Los objetos de red IP se construyen a partir de cadenas que definen el rango de direcciones de host que forman parte de esa red. La forma más simple para esa información es un par de «dirección de red/prefijo de red», donde el prefijo define el número de bits iniciales que se comparan para determinar si una dirección es parte de la red y la dirección de red define el valor esperado de esos bits.

En cuanto a las direcciones, se proporciona una función de fábrica que determina automáticamente la versión IP correcta:

```
>>> ipaddress.ip_network('192.0.2.0/24')
IPv4Network('192.0.2.0/24')
>>> ipaddress.ip_network('2001:db8::0/96')
IPv6Network('2001:db8::/96')
```

Los objetos de red no pueden tener ningún bit de host establecido. El efecto práctico de esto es que `192.0.2.1/24` no describe una red. Tales definiciones se conocen como objetos de interfaz ya que la notación *ip-on-a-network* se usa comúnmente para describir interfaces de red de un ordenador en una red determinada y se describen más adelante en la siguiente sección.

De forma predeterminada, al intentar crear un objeto de red con los bits de host establecidos, se lanzará un **`ValueError`**. Para solicitar que los bits adicionales se coaccionen a cero, el flag `strict=False` se puede pasar al constructor:

```
>>> ipaddress.ip_network('192.0.2.1/24')
Traceback (most recent call last):
   ...
ValueError: 192.0.2.1/24 has host bits set
>>> ipaddress.ip_network('192.0.2.1/24', strict=False)
IPv4Network('192.0.2.0/24')
```

Si bien la forma de cadena de caracteres ofrece mucha más flexibilidad, las redes también se pueden definir con enteros, al igual que las direcciones de host. En este caso, se considera que la red contiene solo la dirección única identificada por el entero, por lo que el prefijo de red incluye toda la dirección de red:

```
>>> ipaddress.ip_network(3221225984)
IPv4Network('192.0.2.0/32')
>>> ipaddress.ip_network(4254076641128259285690398495165
IPv6Network('2001:db8::/128')
```

Al igual que con las direcciones, la creación de un tipo particular de red se puede forzar llamando directamente al constructor de clase en lugar de usar la función de fábrica.

Interfaces de host

Como se mencionó anteriormente, si necesita describir una dirección en una red en particular, ni la dirección ni las clases de red son suficientes. La notación como `192.0.2.1/24` es comúnmente utilizada por los ingenieros de red y las personas que escriben herramientas para cortafuegos y enrutadores como abreviatura de «el host `192.0.2.1` en la red `192.0.2.0/24`», En consecuencia, **`ipaddress`** proporciona un conjunto de clases híbridas que asocian una dirección con una red en particular. La interfaz para la creación es idéntica a la de definir objetos de red, excepto que la parte de dirección no está restringida a ser una dirección de red.

```
>>> ipaddress.ip_interface('192.0.2.1/24')
IPv4Interface('192.0.2.1/24')
>>> ipaddress.ip_interface('2001:db8::1/96')
IPv6Interface('2001:db8::1/96')
```

Se aceptan entradas enteras (como con las redes), y el uso de una versión IP particular se puede forzar llamando directamente al constructor relevante.

Inspeccionando objetos Dirección/Red/Interfaz (*Address/Network/Interface*)

Se ha tomado la molestia de crear un objeto *IPv(4|6)(Address|Network|Interface)*, por lo que probablemente desee obtener información al respecto. **`ipaddress`** intenta hacer esto fácil e intuitivo.

Extrayendo la versión IP:

```
>>> addr4 = ipaddress.ip_address('192.0.2.1')
>>> addr6 = ipaddress.ip_address('2001:db8::1')
>>> addr6.version
6
>>> addr4.version
4
```

Obteniendo la red desde una interfaz:

```
>>> host4 = ipaddress.ip_interface('192.0.2.1/24')
>>> host4.network
IPv4Network('192.0.2.0/24')
>>> host6 = ipaddress.ip_interface('2001:db8::1/96')
>>> host6.network
IPv6Network('2001:db8::/96')
```

Averiguando cuántas direcciones individuales hay en una red:

```
>>> net4 = ipaddress.ip_network('192.0.2.0/24')
>>> net4.num_addresses
256
>>> net6 = ipaddress.ip_network('2001:db8::0/96')
>>> net6.num_addresses
4294967296
```

Iterando a través de las direcciones «utilizables» en una red:

```
>>> net4 = ipaddress.ip_network('192.0.2.0/24')
>>> for x in net4.hosts():
...     print(x)
192.0.2.1
192.0.2.2
192.0.2.3
```

```
192.0.2.4
...
192.0.2.252
192.0.2.253
192.0.2.254
```

Obteniendo la máscara de red (es decir, establecer bits correspondientes al prefijo de red) o la máscara de host (cualquier bits que no forme parte de la máscara de red):

```
>>> net4 = ipaddress.ip_network('192.0.2.0/24')
>>> net4.netmask
IPv4Address('255.255.255.0')
>>> net4.hostmask
IPv4Address('0.0.0.255')
>>> net6 = ipaddress.ip_network('2001:db8::0/96')
>>> net6.netmask
IPv6Address('ffff:ffff:ffff:ffff:ffff:ffff::')
>>> net6.hostmask
IPv6Address('::ffff:ffff')
```

Expandiendo o comprimiendo la dirección:

```
>>> addr6.exploded
'2001:0db8:0000:0000:0000:0000:0000:0001'
>>> addr6.compressed
'2001:db8::1'
>>> net6.exploded
'2001:0db8:0000:0000:0000:0000:0000:0000/96'
>>> net6.compressed
'2001:db8::/96'
```

Si bien IPv4 no admite expansión o compresión, los objetos asociados aún proporcionan las propiedades relevantes para que el código neutral de la versión pueda garantizar fácilmente que se use la forma más concisa o detallada para las direcciones IPv6 mientras se maneja correctamente las direcciones IPv4.

Redes como listas de direcciones

A veces es útil tratar las redes como listas. Esto significa que es posible indexarlas de esta manera:

```
>>> net4[1]
IPv4Address('192.0.2.1')
>>> net4[-1]
IPv4Address('192.0.2.255')
>>> net6[1]
IPv6Address('2001:db8::1')
>>> net6[-1]
IPv6Address('2001:db8::ffff:ffff')
```

También significa que los objetos de red se prestan a usar la sintaxis del test de lista de membresía como esta:

```
if address in network:
    # do something
```

Las pruebas de contención se realizan de manera eficiente según el prefijo de red:

```
>>> addr4 = ipaddress.ip_address('192.0.2.1')
>>> addr4 in ipaddress.ip_network('192.0.2.0/24')
True
>>> addr4 in ipaddress.ip_network('192.0.3.0/24')
False
```

Comparaciones

`ipaddress` proporciona algunas formas simples e intuitivas de comparar objetos, donde esto tiene sentido:

```
>>> ipaddress.ip_address('192.0.2.1') < ipaddress.ip_add
True
```

Se genera una excepción **`TypeError`** si intenta comparar objetos de diferentes versiones o tipos diferentes.

Uso de direcciones IP con otros módulos

Otros módulos que usan direcciones IP (como **`socket`**) generalmente no aceptarán directamente objetos de este módulo. En su lugar, deben ser forzados a un entero o una cadena que el otro módulo deberá aceptar:

```
>>> addr4 = ipaddress.ip_address('192.0.2.1')
>>> str(addr4)
'192.0.2.1'
>>> int(addr4)
3221225985
```

Obtener más detalles cuando se produce un error en la creación de instancias

Al crear objetos de dirección/red/interfaz utilizando las funciones de fábrica independientes de la versión, cualquier error se informará como **`ValueError`** con un mensaje de error genérico que simplemente dice que el valor pasado no se reconoció como un objeto de ese tipo. La falta de un error específico se debe a que es necesario saber si *se supone* que el valor es IPv4 o IPv6 para poder proporcionar más detalles sobre por qué se ha rechazado.

Para admitir casos de uso en los que es útil tener acceso a este detalle adicional, los constructores de clase individuales en realidad lanza las subclases **`ValueError`** **`ipaddress.AddressValueError`** y **`ipaddress.NetmaskValueError`** para indicar exactamente qué parte de la definición no se pudo analizar correctamente.

Los mensajes de error son significativamente más detallados cuando se usan los constructores de clase directamente. Por ejemplo:

```
>>> ipaddress.ip_address("192.168.0.256")
Traceback (most recent call last):
  ...
ValueError: '192.168.0.256' does not appear to be an IPv
>>> ipaddress.IPv4Address("192.168.0.256")
Traceback (most recent call last):
  ...
```

```
ipaddress.AddressValueError: Octet 256 (> 255) not permi

>>> ipaddress.ip_network("192.168.0.1/64")
Traceback (most recent call last):
  ...
ValueError: '192.168.0.1/64' does not appear to be an IP
>>> ipaddress.IPv4Network("192.168.0.1/64")
Traceback (most recent call last):
  ...
ipaddress.NetmaskValueError: '64' is not a valid netmask
```

Sin embargo, ambas excepciones específicas del módulo tienen **`ValueError`** como su clase principal, por lo que si no le preocupa el tipo particular de error, aún puede escribir código como el siguiente:

```
try:
    network = ipaddress.IPv4Network(address)
except ValueError:
    print('address/netmask is invalid for IPv4:', addres
```

Instrumentación de CPython con DTrace y SystemTap

autor:

David Malcolm

autor:

Łukasz Langa

DTrace y SystemTap son herramientas de monitoreo, cada una de las cuales proporciona una forma de inspeccionar lo que están haciendo los procesos en un sistema informático. Ambos usan lenguajes específicos de dominio que permiten al usuario escribir scripts que:

- filtrar qué procesos deben observarse
- recopilar datos de los procesos de interés
- generar reportes sobre los datos

A partir de Python 3.6, CPython se puede construir con «marcadores» incrustados, también conocidos como «sondas», que se pueden observar mediante un script de DTrace o SystemTap, lo que facilita la supervisión de lo que hacen los procesos de CPython en un sistema.

Los marcadores de DTrace son detalles de implementación del intérprete CPython. No se ofrecen garantías sobre la compatibilidad de la sonda entre versiones de CPython. Los scripts de DTrace pueden dejar de funcionar o funcionar incorrectamente sin previo aviso al cambiar las versiones de CPython.

Habilitando los marcadores estáticos

macOS viene con soporte integrado para DTrace. En Linux, para construir CPython con los marcadores incrustados para SystemTap, se deben instalar las herramientas de desarrollo de SystemTap.

En una máquina Linux, esto se puede hacer a través de:

```
$ yum install systemtap-sdt-devel
```

o:

```
$ sudo apt-get install systemtap-sdt-dev
```

CPython debe ser **`configurado con la opción --with-dtrace`**:

```
checking for --with-dtrace... yes
```

En macOS, puede enumerar las sondas disponibles de DTrace ejecutando un proceso de Python en segundo plano y listando todas las sondas disponibles por el proveedor de Python:

```
$ python3.6 -q &
$ sudo dtrace -l -P python$!  # or: dtrace -l -m python3

   ID   PROVIDER            MODULE
29564 python18035        python3.6          _PyEval_Eval
29565 python18035        python3.6             dtrace_fu
29566 python18035        python3.6          _PyEval_Eval
29567 python18035        python3.6            dtrace_fun
29568 python18035        python3.6
29569 python18035        python3.6
29570 python18035        python3.6          _PyEval_Eval
29571 python18035        python3.6                 maybe
```

En Linux, puede verificar si los marcadores estáticos SystemTap están presentes en el binario construido al ver si contiene una sección «.note.stapsdt».

```
$ readelf -S ./python | grep .note.stapsdt
[30] .note.stapsdt        NOTE         0000000000000000
```

Si ha creado Python como una biblioteca compartida (con la opción de configuración **`--enable-shared`**), debe buscar en la biblioteca compartida. Por ejemplo:

```
$ readelf -S libpython3.3dm.so.1.0 | grep .note.stapsdt
[29] .note.stapsdt        NOTE         0000000000000000
```

Un lector de formato ELF suficientemente moderno puede imprimir los metadatos:

```
$ readelf -n ./python

Displaying notes found at file offset 0x00000254 with le
    Owner                 Data size          Descriptio
    GNU                  0x00000010          NT_GNU_ABI_
        OS: Linux, ABI: 2.6.32

Displaying notes found at file offset 0x00000274 with le
    Owner                 Data size          Descriptio
    GNU                  0x00000014          NT_GNU_BUI
        Build ID: df924a2b08a7e89f6e11251d4602022977af2

Displaying notes found at file offset 0x002d6c30 with le
    Owner                 Data size          Descriptio
    stapsdt              0x00000031          NT_STAPSDT
        Provider: python
        Name: gc__start
        Location: 0x00000000004371c3, Base: 0x000000000
        Arguments: -4@%ebx
    stapsdt              0x00000030          NT_STAPSDT
        Provider: python
        Name: gc__done
        Location: 0x00000000004374e1, Base: 0x000000000
        Arguments: -8@%rax
    stapsdt              0x00000045          NT_STAPSDT
        Provider: python
        Name: function__entry
        Location: 0x000000000053db6c, Base: 0x000000000
        Arguments: 8@%rbp 8@%r12 -4@%eax
    stapsdt              0x00000046          NT_STAPSDT
        Provider: python
        Name: function__return
        Location: 0x000000000053dba8, Base: 0x000000000
        Arguments: 8@%rbp 8@%r12 -4@%eax
```

The above metadata contains information for SystemTap describing how it can patch strategically placed machine code instructions to enable the tracing hooks used by a SystemTap script.

Sondas estáticas DTrace

El siguiente ejemplo de script DTrace se puede utilizar para mostrar la jerarquía de llamada/retorno de un script de Python, solo rastreando dentro de la invocación de una función llamada «start». En otras palabras, las llamadas a funciones durante la importación aparecerán en la lista:

```
self int indent;

python$target:::function-entry
/copyinstr(arg1) == "start"/
{
        self->trace = 1;
}

python$target:::function-entry
/self->trace/
{
        printf("%d\t%*s:", timestamp, 15, probename);
        printf("%*s", self->indent, "");
        printf("%s:%s:%d\n", basename(copyinstr(arg0)),
        self->indent++;
}

python$target:::function-return
/self->trace/
{
        self->indent--;
        printf("%d\t%*s:", timestamp, 15, probename);
        printf("%*s", self->indent, "");
        printf("%s:%s:%d\n", basename(copyinstr(arg0)),
}
```

```
python$target:::function-return
/copyinstr(arg1) == "start"/
{
        self->trace = 0;
}
```

Se puede invocar así:

```
$ sudo dtrace -q -s call_stack.d -c "python3.6 script.py
```

La salida se verá así:

```
156641360502280  function-entry:call_stack.py:start:23
156641360518804  function-entry: call_stack.py:function_
156641360532797  function-entry:  call_stack.py:function
156641360546807 function-return:  call_stack.py:function
156641360563367 function-return: call_stack.py:function_
156641360578365  function-entry: call_stack.py:function_
156641360591757  function-entry:  call_stack.py:function
156641360605556  function-entry:   call_stack.py:functio
156641360617482 function-return:   call_stack.py:functio
156641360629814 function-return:  call_stack.py:function
156641360642285 function-return: call_stack.py:function_
156641360656770  function-entry: call_stack.py:function_
156641360669707 function-return: call_stack.py:function_
156641360687853  function-entry: call_stack.py:function_
156641360700719 function-return: call_stack.py:function_
156641360719640  function-entry: call_stack.py:function_
156641360732567 function-return: call_stack.py:function_
156641360747370 function-return:call_stack.py:start:28
```

Marcadores estáticos SystemTap

La forma de bajo nivel para utilizar la integración de SystemTap es utilizar los marcadores estáticos directamente. Esto requiere que indique explícitamente el archivo binario que los contiene.

Por ejemplo, este script SystemTap se puede utilizar para mostrar la jerarquía de llamada/retorno de un script de Python:

```
probe process("python").mark("function__entry") {
     filename = user_string($arg1);
     funcname = user_string($arg2);
     lineno = $arg3;

     printf("%s => %s in %s:%d\\n",
            thread_indent(1), funcname, filename, lineno
}

probe process("python").mark("function__return") {
    filename = user_string($arg1);
    funcname = user_string($arg2);
    lineno = $arg3;

    printf("%s <= %s in %s:%d\\n",
           thread_indent(-1), funcname, filename, lineno
}
```

Se puede invocar así:

```
$ stap \
  show-call-hierarchy.stp \
  -c "./python test.py"
```

La salida se verá así:

```
11408 python(8274):        => __contains__ in Lib/_abcol
11414 python(8274):         => __getitem__ in Lib/os.py:
11418 python(8274):          => encode in Lib/os.py:490
11424 python(8274):          <= encode in Lib/os.py:493
11428 python(8274):         <= __getitem__ in Lib/os.py:
11433 python(8274):        <= __contains__ in Lib/_abcol
```

donde las columnas son:

- tiempo en microsegundos desde el inicio del script
- nombre del ejecutable
- PID de proceso

y el resto indica la jerarquía de llamada/retorno a medida que se ejecuta el script.

Para una compilación **`--enable-shared`** de CPython, los marcadores están contenidos dentro de la biblioteca compartida libpython, y la ruta de puntos de la sonda debe reflejar esto. Por ejemplo, esta línea del ejemplo anterior:

```
probe process("python").mark("function__entry") {
```

en su lugar debería leer:

```
probe process("python").library("libpython3.6dm.so.1.0")
```

(asumiendo una compilación de depuración de CPython 3.6)

Marcadores estáticos disponibles

function_entry(str filename, str funcname, int lineno)

> Este marcador indica que ha comenzado la ejecución de una función de Python. Solo se activa para funciones de Python puro (código de bytes).
>
> El nombre del archivo, el nombre de la función y el número de línea se retornan al script de rastreo como argumentos posicionales, a los que se debe acceder usando `$arg1`, `$arg2`, `$arg3`:
>
> - `$arg1 : (const char *)` nombre del archivo, accesible usando `user_string($arg1)`
> - `$arg2 : (const char *)` nombre de la función, accesible usando `user_string($arg2)`
> - `$arg3 : int` número de linea

function_return(str filename, str funcname, int lineno)

> This marker is the converse of **`function__entry()`**, and indicates that execution of a Python function has ended (either via `return`, or via an exception). It is only triggered

for pure-Python (bytecode) functions.

The arguments are the same as for `function__entry()`

line(str filename, str funcname, int lineno)

Este marcador indica que una línea de Python está a punto de ejecutarse. Es el equivalente al rastreo línea por línea con un generador de perfiles de Python. No se activa con las funciones de C.

The arguments are the same as for `function__entry()`.

gc_start(int generation)

Fires when the Python interpreter starts a garbage collection cycle. `arg0` is the generation to scan, like `gc.collect()`.

gc_done(long collected)

Se activa cuando el intérprete de Python finaliza un ciclo de recolección de basura. `arg0` es el número de objetos recopilados.

import_find_load_start(str modulename)

Se activa antes `importlib` e intenta encontrar y cargar el módulo. `arg0` es el nombre del módulo.

Added in version 3.7.

import_find_load_done(str modulename, int found)

Se activa después de que la función *find_and_load* de `importlib` es llamada. `arg0` es el nombre del módulo, `arg1` indica si el módulo se cargó correctamente.

Added in version 3.7.

audit(str event, void *tuple)

Se activa cuando se llama `sys.audit()` o `PySys_Audit()`. `arg0` es el nombre del evento como cadena C, `arg1` es un puntero `PyObject` a un objeto tupla.

Added in version 3.8.

C Entry Points

To simplify triggering of DTrace markers, Python’s C API comes with a number of helper functions that mirror each static marker. On builds of Python without DTrace enabled, these do nothing.

In general, it is not necessary to call these yourself, as Python will do it for you.

C API Function	Static Marker	Notes
void PyDTrace_LINE(const char *arg0, const char *arg1, int arg2)	**`line()`**	
void PyDTrace_FUNCTION_ENTRY(const char *arg0, const char *arg1, int arg2)	**`function__entry()`**	
void PyDTrace_FUNCTION_RETURN(const char *arg0, const char *arg1, int arg2)	**`function__return()`**	
void PyDTrace_GC_START(int arg0)	**`gc__start()`**	
void PyDTrace_GC_DONE(Py_ssize_t arg0)	**`gc__done()`**	
void PyDTrace_INSTANCE_NEW_START(int arg0)	**`instance__new__start()`**	Not used by Python
void PyDTrace_INSTANCE_NEW_DONE(int arg0)	**`instance__new__done()`**	Not used by Python
void PyDTrace_INSTANCE_DELETE_START(int arg0)	**`instance__delete__start()`**	Not used by Python
void PyDTrace_INSTANCE_DELETE_DONE(int arg0)	**`instance__delete__done()`**	Not used by Python
void PyDTrace_IMPORT_FIND_LOAD_START(const char *arg0)	**`import__find__load__start()`**	
void PyDTrace_IMPORT_FIND_LOAD_DONE(const char *arg0, int arg1)	**`import__find__load__done()`**	
void PyDTrace_AUDIT(const char *arg0, void *arg1)	**`audit()`**	

C Probing Checks

int PyDTrace_LINE_ENABLED(void)

int PyDTrace_FUNCTION_ENTRY_ENABLED(void)

int PyDTrace_FUNCTION_RETURN_ENABLED(void)

int PyDTrace_GC_START_ENABLED(void)

int PyDTrace_GC_DONE_ENABLED(void)

int PyDTrace_INSTANCE_NEW_START_ENABLED(void)

int PyDTrace_INSTANCE_NEW_DONE_ENABLED(void)

int PyDTrace_INSTANCE_DELETE_START_ENABLED(void)

int PyDTrace_INSTANCE_DELETE_DONE_ENABLED(void)

int PyDTrace_IMPORT_FIND_LOAD_START_ENABLED(void)

int PyDTrace_IMPORT_FIND_LOAD_DONE_ENABLED(void)

int PyDTrace_AUDIT_ENABLED(void)

All calls to `PyDTrace` functions must be guarded by a call to one of these functions. This allows Python to minimize performance impact when probing is disabled.

On builds without DTrace enabled, these functions do nothing and return `0`.

SystemTap Tapsets

La forma de nivel superior de utilizar la integración de SystemTap es utilizar un «tapset»: el equivalente de SystemTap a una biblioteca, que oculta algunos de los detalles de bajo nivel de los marcadores estáticos.

A continuación un archivo de tapset, basado en una compilación no compartida de CPython:

```
/*
   Provide a higher-level wrapping around the function__
   function__return markers:
```

```
 \*/
probe python.function.entry = process("python").mark("fu
{
    filename = user_string($arg1);
    funcname = user_string($arg2);
    lineno = $arg3;
    frameptr = $arg4
}
probe python.function.return = process("python").mark("f
{
    filename = user_string($arg1);
    funcname = user_string($arg2);
    lineno = $arg3;
    frameptr = $arg4
}
```

Si este archivo está instalado en el directorio de tapset de SystemTap (por ejemplo, `/usr/share/systemtap/tapset`), estos puntos de sonda adicionales estarán disponibles:

python.function.entry(str filename, str funcname, int lineno, frameptr)

> Este punto de sonda indica que ha comenzado la ejecución de una función de Python. Solo se activa para funciones de Python puro (código de bytes).

python.function.return(str filename, str funcname, int lineno, frameptr)

> Este punto de prueba es el inverso de `python.function.return`, e indica que la ejecución de una función de Python ha finalizado (ya sea mediante `return` o mediante una excepción). Solo se activa para funciones de Python puro (código de bytes).

Ejemplos

Este script de SystemTap utiliza el tapset anterior para implementar de manera más limpia el ejemplo de rastrear la jerarquía de llamadas a funciones de Python, sin necesidad de nombrar directamente los marcadores estáticos:

```
probe python.function.entry
{
  printf("%s => %s in %s:%d\n",
         thread_indent(1), funcname, filename, lineno);
}

probe python.function.return
{
  printf("%s <= %s in %s:%d\n",
         thread_indent(-1), funcname, filename, lineno);
}
```

The following script uses the tapset above to provide a top-like view of all running CPython code, showing the top 20 most frequently entered bytecode frames, each second, across the whole system:

```
global fn_calls;

probe python.function.entry
{
    fn_calls[pid(), filename, funcname, lineno] += 1;
}

probe timer.ms(1000) {
    printf("\033[2J\033[1;1H") /* clear screen \*/
    printf("%6s %80s %6s %30s %6s\n",
           "PID", "FILENAME", "LINE", "FUNCTION", "CALLS
    foreach ([pid, filename, funcname, lineno] in fn_cal
        printf("%6d %80s %6d %30s %6d\n",
            pid, filename, lineno, funcname,
            fn_calls[pid, filename, funcname, lineno]);
    }
    delete fn_calls;
}
```

Soporte de Python para el perfilador `perf` de Linux

autor:

Pablo Galindo

El perfilador perf de Linux [https://perf.wiki.kernel.org] es una herramienta muy poderosa que le permite crear perfiles y obtener información sobre el rendimiento de su aplicación. `perf` también tiene un ecosistema muy vibrante de herramientas que ayudan con el análisis de los datos que produce.

El principal problema con el uso del perfilador `perf` con aplicaciones Python es que `perf` sólo obtiene información sobre símbolos nativos, es decir, los nombres de funciones y procedimientos escritos en C. Esto significa que los nombres y nombres de archivos de las funciones de Python en su código no aparecerán en la salida de `perf`.

Desde Python 3.12, el intérprete puede ejecutarse en un modo especial que permite que las funciones de Python aparezcan en la salida del perfilador `perf`. Cuando este modo está habilitado, el intérprete interpondrá un pequeño fragmento de código compilado sobre la marcha antes de la ejecución de cada función de Python y enseñará a `perf` la relación entre este fragmento de código y la función de Python asociada usando perf map files.

Nota

Actualmente, el soporte para el perfilador `perf` solo está disponible para Linux en arquitecturas seleccionadas. Verifique el resultado del paso de compilación `configure` o verifique el resultado de `python -m sysconfig | grep HAVE_PERF_TRAMPOLINE` para ver si su sistema es compatible.

Por ejemplo, considere el siguiente script:

```
def foo(n):
    result = 0
    for _ in range(n):
        result += 1
    return result

def bar(n):
    foo(n)

def baz(n):
    bar(n)

if __name__ == "__main__":
    baz(1000000)
```

Podemos ejecutar `perf` para obtener un registro de los seguimientos de la pila de CPU a 9999 hercios:

```
$ perf record -F 9999 -g -o perf.data python my_script.p
```

Luego podemos usar `perf report` para analizar los datos:

```
$ perf report --stdio -n -g

# Children      Self       Samples  Command     Shared O
# ........  ........  ............  ..........  ........
#
    91.08%     0.00%             0  python.exe  python.e
            |
            ---_start
            |
                --90.71%--__libc_start_main
                        Py_BytesMain
                        |
                        |--56.88%--pymain_run_python.cor
                        |          |
                        |          |--56.13%--_PyRun_Any
                        |          |          _PyRun_Sim
                        |          |          |
                        |          |          |--55.02%-
                        |          |          |
                        |          |          |
                        |          |          |
                        |          |          |
                        |          |          |
                        |          |          |
                        |          |          |
                        |          |          |
                        |          |          |
                        |          |          |
                        |          |          |
                        |          |          |
                        |          |          |
                        |          |          |
                        |          |          |
                        |          |          |
                        |          |          |
...
```

As you can see, the Python functions are not shown in the output, only `_PyEval_EvalFrameDefault` (the function that evaluates the Python bytecode) shows up. Unfortunately that's not very useful because all Python functions use the same C function to evaluate bytecode so we cannot know which Python function corresponds to which bytecode-evaluating function.

En cambio, si ejecutamos el mismo experimento con el soporte `perf` habilitado obtenemos:

```
$ perf report --stdio -n -g

# Children      Self       Samples  Command     Shared C
# ........  ........  ............  ..........  ........
#
    90.58%     0.36%             1  python.exe  python.e
            |
                 ---_start
                 |
                     --89.86%--__libc_start_main
                             Py_BytesMain
                             |
                             |--55.43%--pymain_run_python.cor
                             |           |
                             |           |--54.71%--_PyRun_Any
                             |           |          _PyRun_Sim
                             |           |          |
                             |           |          |--53.62%-
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
                             |           |          |
```

Cómo habilitar el soporte de creación de perfiles `perf`

El soporte de creación de perfiles `perf` se puede habilitar desde el principio usando la variable de entorno **`PYTHONPERFSUPPORT`** o la opción `-X perf`, o dinámicamente usando `sys.activate_stack_trampoline()` y `sys.deactivate_stack_trampoline()`.

Las funciones `sys` tienen prioridad sobre la opción `-X`, la opción `-X` tiene prioridad sobre la variable de entorno.

Ejemplo, usando la variable de entorno:

```
$ PYTHONPERFSUPPORT=1 perf record -F 9999 -g -o perf.dat
$ perf report -g -i perf.data
```

Ejemplo, usando la opción `-X`:

```
$ perf record -F 9999 -g -o perf.data python -X perf my_
$ perf report -g -i perf.data
```

Ejemplo, usando las API `sys` en el archivo `example.py`:

```
import sys

sys.activate_stack_trampoline("perf")
do_profiled_stuff()
sys.deactivate_stack_trampoline()

non_profiled_stuff()
```

…entonces:

```
$ perf record -F 9999 -g -o perf.data python ./example.p
$ perf report -g -i perf.data
```

Cómo obtener los mejores resultados

Para obtener mejores resultados, Python debe compilarse con `CFLAGS="-fno-omit-frame-pointer -mno-omit-leaf-frame-pointer"` ya que esto permite a los perfiladores desenrollarse usando solo el puntero del marco y no en la información de depuración de DWARF. Esto se debe a que como el código que se interpone para permitir el soporte `perf` se genera dinámicamente, no tiene ninguna información de depuración DWARF disponible.

Puede verificar si su sistema ha sido compilado con este indicador ejecutando:

```
$ python -m sysconfig | grep 'no-omit-frame-pointer'
```

Si no ve ningún resultado, significa que su intérprete no ha sido compilado con punteros de marco y, por lo tanto, es posible que no pueda mostrar funciones de Python en el resultado de `perf`.

How to work without frame pointers

If you are working with a Python interpreter that has been compiled without frame pointers, you can still use the `perf` profiler, but the overhead will be a bit higher because Python needs to generate unwinding information for every Python function call on the fly. Additionally, `perf` will take more time to process the data because it will need to use the DWARF debugging information to unwind the stack and this is a slow process.

To enable this mode, you can use the environment variable **`PYTHON_PERF_JIT_SUPPORT`** or the **`-X perf_jit`** option, which will enable the JIT mode for the `perf` profiler.

Nota

Due to a bug in the `perf` tool, only `perf` versions higher than v6.8 will work with the JIT mode. The fix was also backported to the v6.7.2 version of the tool.

Note that when checking the version of the `perf` tool (which can be done by running `perf version`) you must take into account that some distros add some custom version numbers including a `-` character. This means that `perf 6.7-3` is not necessarily `perf 6.7.3`.

When using the perf JIT mode, you need an extra step before you can run `perf report`. You need to call the `perf inject`

command to inject the JIT information into the `perf.data` file.:

```
$ perf record -F 9999 -g -k 1 --call-graph dwarf -o perf
$ perf inject -i perf.data --jit --output perf.jit.data
$ perf report -g -i perf.jit.data
```

or using the environment variable:

```
$ PYTHON_PERF_JIT_SUPPORT=1 perf record -F 9999 -g --cal
$ perf inject -i perf.data --jit --output perf.jit.data
$ perf report -g -i perf.jit.data
```

`perf inject --jit` command will read `perf.data`, automatically pick up the perf dump file that Python creates (in `/tmp/perf-$PID.dump`), and then create `perf.jit.data` which merges all the JIT information together. It should also create a lot of `jitted-XXXX-N.so` files in the current directory which are ELF images for all the JIT trampolines that were created by Python.

Advertencia

When using `--call-graph dwarf`, the `perf` tool will take snapshots of the stack of the process being profiled and save the information in the `perf.data` file. By default, the size of the stack dump is 8192 bytes, but you can change the size by passing it after a comma like `--call-graph dwarf,16384`.

The size of the stack dump is important because if the size is too small `perf` will not be able to unwind the stack and the output will be incomplete. On the other hand, if the size is too big, then `perf` won't be able to sample the process as frequently as it would like as the overhead will be higher.

The stack size is particularly important when profiling Python code compiled with low optimization levels (like `-O0`), as these builds tend to have larger stack frames. If you are compiling Python with `-O0` and not seeing Python functions in your profiling output, try increasing the stack dump size to 65528 bytes (the maximum):

```
$ perf record -F 9999 -g -k 1 --call-graph dwarf,65528
```

Different compilation flags can significantly impact stack sizes:

- Builds with `-O0` typically have much larger stack frames than those with `-O1` or higher
- Adding optimizations (`-O1`, `-O2`, etc.) typically reduces stack size
- Frame pointers (`-fno-omit-frame-pointer`) generally provide more reliable stack unwinding

Prácticas recomendadas para las anotaciones

autor:

Larry Hastings

Resumen

Este documento se designó para encapsular las prácticas recomendadas para trabajar con diccionarios de anotaciones. Si escribe código Python que examina `__annotations__` en objetos Python, recomendamos seguir las pautas que se describen a continuación.

El documento se organizó en 4 secciones: prácticas recomendadas para acceder a las anotaciones de un objeto en las versiones de Python 3.10 y posteriores, prácticas recomendadas para acceder a las anotaciones de un objeto en las versiones de Python 3.9 y anteriores, otras prácticas recomendadas para `__annotations__` que aplican a cualquier versión de Python y peculiaridades de `__annotations__`.

Tome en cuenta que este documento trabaja específicamente con `__annotations__`, no usarlo *para* anotaciones. Si está buscando información sobre cómo usar «anotaciones de tipado» en su código, consulte el módulo `typing`.

Acceder al diccionario de anotaciones de un objeto en las versiones de Python 3.10 y posteriores

Python 3.10 adds a new function to the standard library: `inspect.get_annotations()`. In Python versions 3.10 through

3.13, calling this function is the best practice for accessing the annotations dict of any object that supports annotations. This function can also «un-stringize» stringized annotations for you.

In Python 3.14, there is a new `annotationlib` module with functionality for working with annotations. This includes a `annotationlib.get_annotations()` function, which supersedes `inspect.get_annotations()`.

Si por alguna razón `inspect.get_annotations()` no es viable para su caso de uso, puede acceder manualmente al miembro de datos `__annotations__`. La práctica recomendada para esto también cambió en Python 3.10: a partir de Python 3.10, se garantiza que `o.__annotations__` *siempre* opere en funciones, clases y módulos de Python. Si está seguro de que el objeto que está examinando es uno de los tres objetos *específicos*, puede usar simplemente `o.__annotations__` para obtener el diccionario de anotaciones del objeto.

Sin embargo, otros tipos de llamadas – por ejemplo, invocables creados por `functools.partial()` – pueden no tener definido un atributo `__annotations__`. Al acceder a `__annotations__` de un objeto posiblemente desconocido, la práctica recomendada de las versiones de Python 3.10 y posteriores es llamar `getattr()` con tres argumentos, por ejemplo `getattr(o, '__annotations__', None)`.

Antes de Python 3.10, acceder a `__annotations__` en una clase que no define anotaciones pero que tiene una clase padre con anotaciones devolvería las anotaciones de la clase padre. En Python 3.10 y versiones posteriores, las anotaciones de la clase hija serán un diccionario vacío en su lugar.

Acceder al diccionario de anotaciones de un objeto en las versiones de Python 3.9 y anteriores

En versiones de Python 3.9 y anteriores, acceder al diccionario de anotaciones de un objeto es mucho más complicado que en versiones más recientes. El problema es un defecto de diseño en estas versiones anteriores de Python, específicamente relacionado con las anotaciones de clase.

La práctica recomendada para acceder al diccionario de anotaciones de otros objetos – funciones, otros invocables y módulos – es la misma que la de la 3.10, asumiendo que no está llamando **`inspect.get_annotations()`**: debe usar tres argumentos de **`getattr()`** para acceder al atributo `__annotations__` del objeto.

Desafortunadamente, esta no es la mejor práctica para las clases. El problema es que, dado que `__annotations__` es opcional en las clases, y debido a que las clases pueden heredar atributos desde sus clases base, acceder al atributo `__annotations__` de una clase puede retornar por inadvertencia el diccionario de anotaciones de una *clase base.* Como ejemplo:

```
class Base:
    a: int = 3
    b: str = 'abc'

class Derived(Base):
    pass

print(Derived.__annotations__)
```

Esto imprimirá el diccionario de anotaciones de `Base`, no de `Derived`.

Su código deberá tener una ruta de código separada si el objeto que está examinando es una clase (`isinstance(o, type)`). En este caso, la práctica recomendada se basa en un detalle de implementación de las versiones de Python 3.9 y anteriores: si una clase tiene anotaciones definidas, se almacenan en el diccionario **`__dict__`** de la clase. Dado que la clase puede o no tener anotaciones definidas, la mejor práctica es llamar al método **`get()`** en el diccionario de la clase.

Para ponerlo todo junto, aquí hay un código de muestra que accede de forma segura al atributo `__annotations__` en un objeto arbitrario en las versiones de Python 3.9 y anteriores:

```
if isinstance(o, type):
    ann = o.__dict__.get('__annotations__', None)
else:
    ann = getattr(o, '__annotations__', None)
```

Después de ejecutar este código, `ann` debería ser un diccionario o `None`. Recomendamos que vuelva a verificar el tipo de `ann` usando **`isinstance()`** antes de un examen más detenido.

Tome en cuenta que algunos objetos de tipo exótico o con formato incorrecto pueden no tener un atributo **`__dict__`** , así que para mayor seguridad, también puede usar **`getattr()`** para acceder a **`__dict__`**.

Desencadenamiento manual de anotaciones en cadena

En situaciones donde algunas anotaciones pueden estar «encadenadas», y desea evaluar esas cadenas de caracteres para producir los valores de Python que representan, lo mejor es llamar a **`inspect.get_annotations()`** para que haga este trabajo por usted.

Si está usando la versión de Python 3.9 o anterior, o si por alguna razón no puede usar **`inspect.get_annotations()`**, necesitará duplicar su lógica. Recomendamos que examine la implementación de **`inspect.get_annotations()`** en la versión actual de Python y siga un enfoque similar.

En pocas palabras, si desea evaluar una anotación en cadena en un objeto arbitrario `o`:

- Si `o` es un módulo, use `o.__dict__` como `globals` cuando llame a **`eval()`**.
- Si `o` es una clase, use `sys.modules[o.__module__].__dict__` como `globals` y `dict(vars(o))` como `locals` cuando llame a **`eval()`**.

- Si `o` es un invocable envuelto usando `functools.update_wrapper()`, `functools.wraps()` o `functools.partial()`, lo desenvuelve iterativamente accediendo a `o.__wrapped__` o `o.func` según corresponda, hasta que haya encontrado la función raíz sin envolver.
- Si `o` es un invocable (pero no una clase), use `o.__globals__` como globales cuando llame a `eval()`.

Sin embargo, no todos los valores de cadena de caracteres usados como anotaciones se pueden convertir correctamente en valores de Python mediante `eval()`. Los valores de cadena de caracteres pueden contener, teóricamente, cualquier cadena de caracteres válida, y en la práctica, hay varios casos de uso válidos para anotaciones de tipo que requieren anotaciones con valores de cadena de caracteres que *no pueden* evaluarse específicamente. Por ejemplo:

- **PEP 604** [https://peps.python.org/pep-0604/] introduce tipos de unión usando `|`, antes de que se agregara soporte para esto en Python 3.10.
- Las definiciones que no son necesarias en tiempo de ejecución, sólo se importan cuando `typing.TYPE_CHECKING` es verdadero.

Si `eval()` intenta evaluar tales valores, fallará y lanzará una excepción. Por lo tanto, al diseñar una API de biblioteca que funcione con anotaciones, se recomienda sólo intentar evaluar valores de cadena de caracteres cuando la llamada lo solicite explícitamente.

Prácticas recomendadas para `__annotations__` en cualquier versión de Python

- Debe evitar asignar directamente al miembro `__annotations__` de objetos. Deje que Python administre la configuración `__annotations__`.
- Si asigna directamente al miembro `__annotations__` de

un objeto, siempre debe establecerlo en un objeto `dict`.
- You should avoid accessing `__annotations__` directly on any object. Instead, use **`annotationlib.get_annotations()`** (Python 3.14+) or **`inspect.get_annotations()`** (Python 3.10+).
- If you do directly access the `__annotations__` member of an object, you should ensure that it's a dictionary before attempting to examine its contents.
- Debe evitar modificar los diccionarios `__annotations__`.
- Debe evitar eliminar el atributo `__annotations__` de un objeto.

Peculiaridades de `__annotations__`

En todas las versiones de Python 3, los objetos de función crean de forma diferida un diccionario de anotaciones si no hay anotaciones definidas en ese objeto. Puede eliminar el atributo `__annotations__` usando `del fn.__annotations__`, pero si luego accede a `fn.__annotations__`, el objeto creará un diccionario nuevo vacío que almacenará y retornará como sus anotaciones. Eliminar las anotaciones en una función antes de que haya creado su diccionario de anotaciones de forma diferida arrojará un `AttributeError`; el uso de dos veces seguidas de `del fn.__annotations__` garantiza que siempre arroje un `AttributeError`.

Todo en el párrafo anterior también se aplica a los objetos de clase y módulo en las versiones de Python 3.10 y posteriores.

En todas las versiones de Python 3, puede establecer `__annotations__` en un objeto de función en `None`. Sin embargo, al acceder después a las anotaciones en ese objeto usando `fn.__annotations__` se creará un diccionario vacío de forma diferida según el primer párrafo de esta sección. Esto *no* es cierto para módulos y clases, en cualquier versión de Python; esos objetos permiten establecer `__annotations__` en cualquier valor de Python, y conservarán cualquier valor que se establezca.

Si Python encadena sus anotaciones por usted (usando `from __future__ import annotations`), y especifica una cadena de caracteres como una anotación, la cadena de caracteres en sí se citará. En efecto, la anotación se cita *dos veces*. Por ejemplo:

```
from __future__ import annotations
def foo(a: "str"): pass

print(foo.__annotations__)
```

Esto imprime `{'a': "'str'"}`. En realidad, esto no debería considerarse una «peculiaridad»; aquí se menciona simplemente porque podría sorprenderle.

If you use a class with a custom metaclass and access `__annotations__` on the class, you may observe unexpected behavior; see **749** [https://peps.python.org/pep-0749/#pep749-metaclasses] for some examples. You can avoid these quirks by using **`annotationlib.get_annotations()`** on Python 3.14+ or **`inspect.get_annotations()`** on Python 3.10+. On earlier versions of Python, you can avoid these bugs by accessing the annotations from the class's **`__dict__`** (for example, `cls.__dict__.get('__annotations__', None)`).

In some versions of Python, instances of classes may have an `__annotations__` attribute. However, this is not supported functionality. If you need the annotations of an instance, you can use **`type()`** to access its class (for example, `annotationlib.get_annotations(type(myinstance))` on Python 3.14+).

Aislamiento de módulos de extensión

Resumen

Tradicionalmente, el estado perteneciente a los módulos de extensión de Python se mantuvo en las variables C `static`, que tienen un alcance de todo el proceso. Este documento describe los problemas de dicho estado por proceso y muestra una forma más segura: el estado por módulo.

El documento también describe cómo cambiar al estado por módulo cuando sea posible. Esta transición implica asignar espacio para ese estado, cambiar potencialmente de tipos estáticos a tipos de montón y, quizás lo más importante, acceder al estado por módulo desde el código.

¿Quién debería leer esto?

Esta guía está escrita para los mantenedores de extensiones C-API que deseen hacer que esa extensión sea más segura para usar en aplicaciones donde Python se usa como biblioteca.

Trasfondo

Un intérprete (*interpreter*) es el contexto en el que se ejecuta el código de Python. Contiene la configuración (p. ej., la ruta de importación) y el estado de tiempo de ejecución (p. ej., el conjunto de módulos importados).

Python admite la ejecución de varios intérpretes en un solo proceso. Hay dos casos en los que pensar, los usuarios pueden ejecutar intérpretes:

- en secuencia, con varios ciclos `Py_InitializeEx()`/`Py_FinalizeEx()`, y
- en paralelo, gestionando «subintérpretes» mediante `Py_NewInterpreter()`/`Py_EndInterpreter()`.

Ambos casos (y combinaciones de ellos) serían más útiles al incorporar Python dentro de una biblioteca. Las bibliotecas generalmente no deben hacer suposiciones sobre la aplicación que las usa, lo que incluye asumir un «intérprete principal de Python» en todo el proceso.

Históricamente, los módulos de extensión de Python no manejan bien este caso de uso. Muchos módulos de extensión (e incluso algunos módulos stdlib) usan el estado global *por-proceso*, porque las variables C `static` son extremadamente fáciles de usar. Así, los datos que deberían ser específicos de un intérprete acaban siendo compartidos entre intérpretes. A menos que el desarrollador de la extensión tenga cuidado, es muy fácil introducir casos extremos que provocan bloqueos cuando un módulo se carga en más de un intérprete en el mismo proceso.

Desafortunadamente, el estado *por-intérprete* no es fácil de lograr. Los autores de extensiones tienden a no tener en cuenta múltiples intérpretes cuando desarrollan, y actualmente es engorroso probar el comportamiento.

Ingrese al estado por módulo

Instead of focusing on per-interpreter state, Python's C API is evolving to better support the more granular *per-module* state. This means that C-level data should be attached to a *module object*. Each interpreter creates its own module object, keeping the data separate. For testing the isolation, multiple module objects corresponding to a single extension can even be loaded in a single interpreter.

El estado por módulo proporciona una manera fácil de pensar en la vida útil y la propiedad de los recursos: el módulo de extensión se inicializará cuando se cree un objeto de módulo y se limpiará cuando se libere. En este sentido, un módulo es como cualquier otro PyObject*; no hay ganchos de «apagado del intérprete» para pensar u olvidar.

Tenga en cuenta que hay casos de uso para diferentes tipos de «globales»: por proceso, por intérprete, por subproceso o por estado de tarea. Con el estado por módulo como predeterminado, estos aún son posibles, pero debe tratarlos como casos excepcionales: si los necesita, debe brindarles atención y pruebas adicionales. (Tenga en cuenta que esta guía no los cubre).

Objetos módulos aislados

El punto clave a tener en cuenta al desarrollar un módulo de extensión es que se pueden crear varios objetos de módulo a partir de una única biblioteca compartida. Por ejemplo:

```
>>> import sys
>>> import binascii
>>> old_binascii = binascii
>>> del sys.modules['binascii']
>>> import binascii  # create a new module object
>>> old_binascii == binascii
False
```

Como regla general, los dos módulos deben ser completamente independientes. Todos los objetos y el estado específico del módulo deben encapsularse dentro del objeto del módulo, no compartirse con otros objetos del módulo y limpiarse cuando se desasigna el objeto del módulo. Dado que esto es solo una regla general, las excepciones son posibles (consulte Managing Global State), pero necesitarán más reflexión y atención en los casos extremos.

Si bien algunos módulos podrían funcionar con restricciones menos estrictas, los módulos aislados facilitan el establecimiento de expectativas y pautas claras que funcionan en una variedad de casos de uso.

Casos extremos sorprendentes

Tenga en cuenta que los módulos aislados crean algunos casos extremos sorprendentes. En particular, cada objeto de módulo normalmente no compartirá sus clases y excepciones con otros módulos similares. Continuando con example above, tenga en cuenta que `old_binascii.Error` y `binascii.Error` son objetos separados. En el código siguiente, se detecta la excepción *not*:

```
>>> old_binascii.Error == binascii.Error
False
>>> try:
...     old_binascii.unhexlify(b'qwertyuiop')
... except binascii.Error:
...     print('boo')
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
binascii.Error: Non-hexadecimal digit found
```

Esto se espera. Tenga en cuenta que los módulos de Python puro se comportan de la misma manera: es una parte de cómo funciona Python.

El objetivo es hacer que los módulos de extensión sean seguros en el nivel C, no hacer que los piratas informáticos se comporten de manera intuitiva. Mutar `sys.modules` «manualmente» cuenta como un hack.

Cómo hacer que los módulos sean seguros con varios intérpretes

Administrar el estado global

A veces, el estado asociado con un módulo de Python no es específico de ese módulo, sino de todo el proceso (o algo más «más global» que un módulo). Por ejemplo:

- El módulo `readline` gestiona *el* terminal.
- Un módulo que se ejecuta en una placa de circuito quiere controlar *el* LED integrado.

En estos casos, el módulo Python debería proporcionar *acceso* al estado global, en lugar de *poseerlo*. Si es posible, escriba el módulo para que varias copias del mismo puedan acceder al estado de forma independiente (junto con otras bibliotecas, ya sea para Python u otros lenguajes). Si eso no es posible, considere el bloqueo explícito.

If it is necessary to use process-global state, the simplest way to avoid issues with multiple interpreters is to explicitly prevent a module from being loaded more than once per process—see Exclusión voluntaria: limitación a un objeto de módulo por proceso.

Administración del estado por módulo

Para usar el estado por módulo, use multi-phase extension module initialization. Esto indica que su módulo admite múltiples intérpretes correctamente.

Establezca `PyModuleDef.m_size` en un número positivo para solicitar tantos bytes de almacenamiento local para el módulo. Por lo general, esto se establecerá en el tamaño de algún `struct` específico del módulo, que puede almacenar todo el estado de nivel C del módulo. En particular, es donde debe colocar los punteros a las clases (incluidas las excepciones, pero excluyendo los tipos estáticos) y configuraciones (por ejemplo, **`field_size_limit`** de `csv`) que el código C necesita para funcionar.

Nota

Otra opción es almacenar el estado en el `__dict__` del módulo, pero debe evitar fallas cuando los usuarios modifican `__dict__` desde el código de Python. Esto generalmente significa verificación de errores y tipos en el nivel C, que es fácil equivocarse y difícil de probar lo suficiente.

Sin embargo, si el estado del módulo no es necesario en el código C, almacenarlo solo en `__dict__` es una buena idea.

Si el estado del módulo incluye punteros `PyObject`, el objeto del módulo debe contener referencias a esos objetos e implementar los enlaces de nivel de módulo `m_traverse`, `m_clear` y `m_free`. Estos funcionan como `tp_traverse`, `tp_clear` y `tp_free` de una clase. Agregarlos requerirá algo de trabajo y hará que el código sea más largo; este es el precio de los módulos que se pueden descargar limpiamente.

Un ejemplo de un módulo con estado por módulo está actualmente disponible como xxlimited [https://github.com/python/cpython/blob/master/Modules/xxlimited.c]; ejemplo de inicialización del módulo que se muestra en la parte inferior del archivo.

Exclusión voluntaria: limitación a un objeto de módulo por proceso

Un `PyModuleDef.m_size` no negativo indica que un módulo admite varios intérpretes correctamente. Si este aún no es el caso de su módulo, puede hacer que su módulo se pueda cargar explícitamente solo una vez por proceso. Por ejemplo:

```
// A process-wide flag
static int loaded = 0;

// Mutex to provide thread safety (only needed for free-
static PyMutex modinit_mutex = {0};

static int
exec_module(PyObject* module)
{
    PyMutex_Lock(&modinit_mutex);
    if (loaded) {
        PyMutex_Unlock(&modinit_mutex);
        PyErr_SetString(PyExc_ImportError,
                        "cannot load module more than o
        return -1;
    }
    loaded = 1;
    PyMutex_Unlock(&modinit_mutex);
    // ... rest of initialization
}
```

If your module's **`PyModuleDef.m_clear`** function is able to prepare for future re-initialization, it should clear the `loaded` flag. In this case, your module won't support multiple instances existing *concurrently*, but it will, for example, support being loaded after Python runtime shutdown (**`Py_FinalizeEx()`**) and re-initialization (**`Py_Initialize()`**).

Acceso al estado del módulo desde las funciones

Acceder al estado desde funciones a nivel de módulo es sencillo. Las funciones obtienen el objeto del módulo como su primer argumento; para extraer el estado, puede usar `PyModule_GetState`:

```
static PyObject *
func(PyObject *module, PyObject *args)
{
    my_struct *state = (my_struct*)PyModule_GetState(moc
    if (state == NULL) {
        return NULL;
    }
    // ... rest of logic
}
```

Nota

`PyModule_GetState` puede retornar `NULL` sin establecer una excepción si no hay un estado de módulo, es decir, `PyModuleDef.m_size` era cero. En su propio módulo, tiene el control de `m_size`, por lo que es fácil de evitar.

Tipos Heap

Tradicionalmente, los tipos definidos en código C son *estáticos*; es decir, estructuras `static PyTypeObject` definidas directamente en el código e inicializadas mediante `PyType_Ready()`.

Tales tipos son necesariamente compartidos a lo largo del proceso.

Compartirlos entre objetos de módulo requiere prestar atención a cualquier estado que posean o al que accedan. Para limitar los posibles problemas, los tipos estáticos son inmutables en el nivel de Python: por ejemplo, no puede configurar `str.myattribute = 123`.

Compartir objetos verdaderamente inmutables entre intérpretes está bien, siempre que no proporcionen acceso a objetos mutables. Sin embargo, en CPython, cada objeto de Python tiene un detalle de implementación mutable: el recuento de referencias. Los cambios en el refcount están protegidos por el GIL. Por lo tanto, el código que comparte cualquier objeto de Python entre intérpretes depende implícitamente del GIL actual de todo el proceso de CPython.

Debido a que son inmutables y globales de proceso, los tipos estáticos no pueden acceder a «su» estado de módulo. Si algún método de este tipo requiere acceso al estado del módulo, el tipo debe convertirse a *tipo almacenado en memoria dinámica (heap)* o *tipo heap* para abreviar. Estos se corresponden más estrechamente con las clases creadas por la instrucción `class` de Python.

Para los módulos nuevos, usar tipos heap de forma predeterminada es una buena regla general.

Cambio de tipos estáticos a tipos heap

Los tipos estáticos se pueden convertir en tipos heap, pero tenga en cuenta que la API de tipo heap no se diseñó para la conversión «sin pérdidas» de tipos estáticos, es decir, para crear un tipo que funcione exactamente como un tipo estático determinado. Por lo tanto, al reescribir la definición de clase en una nueva API, es probable que cambie sin querer algunos detalles (por ejemplo, capacidad de selección o espacios heredados). Siempre pruebe los detalles que son importantes para usted.

Tenga cuidado con los siguientes dos puntos en particular (pero tenga en cuenta que esta no es una lista completa):

- Unlike static types, heap type objects are mutable by default. Use the **`Py_TPFLAGS_IMMUTABLETYPE`** flag to prevent mutability.

- Heap types inherit `tp_new` by default, so it may become possible to instantiate them from Python code. You can prevent this with the `Py_TPFLAGS_DISALLOW_INSTANTIATION` flag.

Definición de tipos heap

Los tipos heap se pueden crear completando una estructura `PyType_Spec`, una descripción o «modelo» de una clase y llamando a `PyType_FromModuleAndSpec()` para construir un nuevo objeto de clase.

> **Nota**
>
> Otras funciones, como `PyType_FromSpec()`, también pueden crear tipos heap, pero `PyType_FromModuleAndSpec()` asocia el módulo con la clase, lo que permite el acceso al estado del módulo desde los métodos.

La clase generalmente debe almacenarse en *ambos*, el estado del módulo (para acceso seguro desde C) y el `__dict__` del módulo (para acceso desde código Python).

Protocolo de recolección de basura

Las instancias de tipos heap contienen una referencia a su tipo. Esto garantiza que el tipo no se destruya antes de que se destruyan todas sus instancias, pero puede generar ciclos de referencia que el recolector de elementos no utilizados debe interrumpir.

Para evitar pérdidas de memoria, las instancias de los tipos heap deben implementar el protocolo de recolección de elementos no utilizados. Es decir, los tipos heap deben:

- Have the `Py_TPFLAGS_HAVE_GC` flag.
- Define a traverse function using `Py_tp_traverse`, which visits the type (e.g. using `Py_VISIT(Py_TYPE(self))`).

Please refer to the documentation of `Py_TPFLAGS_HAVE_GC` and `tp_traverse` for additional considerations.

The API for defining heap types grew organically, leaving it somewhat awkward to use in its current state. The following sections will guide you through common issues.

`tp_traverse` in Python 3.8 and lower

The requirement to visit the type from `tp_traverse` was added in Python 3.9. If you support Python 3.8 and lower, the traverse function must *not* visit the type, so it must be more complicated:

```
static int my_traverse(PyObject *self, visitproc visit,
{
    if (Py_Version >= 0x03090000) {
        Py_VISIT(Py_TYPE(self));
    }
    return 0;
}
```

Unfortunately, **`Py_Version`** was only added in Python 3.11. As a replacement, use:

- **`PY_VERSION_HEX`**, if not using the stable ABI, or
- **`sys.version_info`** (via **`PySys_GetObject()`** and **`PyArg_ParseTuple()`**).

Delegating `tp_traverse`

If your traverse function delegates to the **`tp_traverse`** of its base class (or another type), ensure that `Py_TYPE(self)` is visited only once. Note that only heap type are expected to visit the type in `tp_traverse`.

Por ejemplo, si su función poligonal incluye:

```
base->tp_traverse(self, visit, arg)
```

…y `base` puede ser un tipo estático, entonces también debe incluir:

```
if (base->tp_flags & Py_TPFLAGS_HEAPTYPE) {
    // a heap type's tp_traverse already visited Py_TYPE
} else {
    if (Py_Version >= 0x03090000) {
        Py_VISIT(Py_TYPE(self));
    }
}
```

It is not necessary to handle the type’s reference count in `tp_new` and `tp_clear`.

Defining `tp_dealloc`

If your type has a custom `tp_dealloc` function, it needs to:

- call `PyObject_GC_UnTrack()` before any fields are invalidated, and
- decrement the reference count of the type.

To keep the type valid while `tp_free` is called, the type’s refcount needs to be decremented *after* the instance is deallocated. For example:

```
static void my_dealloc(PyObject *self)
{
    PyObject_GC_UnTrack(self);
    ...
    PyTypeObject *type = Py_TYPE(self);
    type->tp_free(self);
    Py_DECREF(type);
}
```

The default `tp_dealloc` function does this, so if your type does *not* override `tp_dealloc` you don’t need to add it.

Not overriding `tp_free`

The `tp_free` slot of a heap type must be set to `PyObject_GC_Del()`. This is the default; do not override it.

Avoiding `PyObject_New`

GC-tracked objects need to be allocated using GC-aware functions.

If you use `PyObject_New()` or `PyObject_NewVar()`:

- Get and call type's `tp_alloc` slot, if possible. That is, replace `TYPE *o = PyObject_New(TYPE, typeobj)` with:

  ```
  TYPE *o = typeobj->tp_alloc(typeobj, 0);
  ```

  Replace `o = PyObject_NewVar(TYPE, typeobj, size)` with the same, but use size instead of the 0.

- If the above is not possible (e.g. inside a custom `tp_alloc`), call `PyObject_GC_New()` or `PyObject_GC_NewVar()`:

  ```
  TYPE *o = PyObject_GC_New(TYPE, typeobj);
  ```

  ```
  TYPE *o = PyObject_GC_NewVar(TYPE, typeobj, size);
  ```

Acceso al estado del módulo desde las clases

Si tiene un objeto de tipo definido con `PyType_FromModuleAndSpec()`, puede llamar a `PyType_GetModule()` para obtener el módulo asociado y luego a `PyModule_GetState()` para obtener el estado del módulo.

Para ahorrar un tedioso código repetitivo de manejo de errores, puede combinar estos dos pasos con `PyType_GetModuleState()`, lo que da como resultado:

```
my_struct *state = (my_struct*)PyType_GetModuleState(typ
if (state == NULL) {
    return NULL;
}
```

Acceso al estado del módulo desde métodos regulares

Acceder al estado de nivel de módulo desde los métodos de una clase es algo más complicado, pero es posible gracias a la API introducida en Python 3.9. Para obtener el estado, primero debe obtener la *clase de definición* y luego obtener el estado del módulo.

El obstáculo más grande es obtener *la clase en la que se definió un método*, o la «clase de definición» de ese método para abreviar. La clase de definición puede tener una referencia al módulo del que forma parte.

Do not confuse the defining class with `Py_TYPE(self)`. If the method is called on a *subclass* of your type, `Py_TYPE(self)` will refer to that subclass, which may be defined in different module than yours.

Nota

El siguiente código de Python puede ilustrar el concepto. `Base.get_defining_class` retorna `Base` incluso si `type(self) == Sub`:

```
class Base:
    def get_type_of_self(self):
        return type(self)

    def get_defining_class(self):
        return __class__

class Sub(Base):
    pass
```

For a method to get its «defining class», it must use the METH_METHOD | METH_FASTCALL | METH_KEYWORDS **`calling convention`** and the corresponding **`PyCMethod`** signature:

```
PyObject *PyCMethod(
    PyObject *self,               // object the method w
    PyTypeObject *defining_class, // defining class
    PyObject *const *args,        // C array of argument
    Py_ssize_t nargs,             // length of "args"
    PyObject *kwnames)            // NULL, or dict of ke
```

Una vez que tenga la clase de definición, llame a **`PyType_GetModuleState()`** para obtener el estado de su módulo asociado.

Por ejemplo:

```
static PyObject *
example_method(PyObject *self,
        PyTypeObject *defining_class,
        PyObject *const *args,
        Py_ssize_t nargs,
        PyObject *kwnames)
{
    my_struct *state = (my_struct*)PyType_GetModuleState
    if (state == NULL) {
        return NULL;
    }
    ... // rest of logic
}

PyDoc_STRVAR(example_method_doc, "...");

static PyMethodDef my_methods[] = {
    {"example_method",
      (PyCFunction)(void(*)(void))example_method,
      METH_METHOD|METH_FASTCALL|METH_KEYWORDS,
      example_method_doc}
    {NULL},
}
```

Acceso al estado del módulo desde métodos de Slot, Getters y Setters

Nota

Esto es nuevo en Python 3.11.

Slot methods—the fast C equivalents for special methods, such as `nb_add` for `__add__` or `tp_new` for initialization—have a very simple API that doesn't allow passing in the defining class, unlike with `PyCMethod`. The same goes for getters and setters defined with `PyGetSetDef`.

Para acceder al estado del módulo en estos casos, utilice la función `PyType_GetModuleByDef()` y pase la definición del módulo. Una vez que tenga el módulo, llame a `PyModule_GetState()` para obtener el estado:

```
PyObject *module = PyType_GetModuleByDef(Py_TYPE(self),
my_struct *state = (my_struct*)PyModule_GetState(module)
if (state == NULL) {
    return NULL;
}
```

`PyType_GetModuleByDef()` works by searching the method resolution order (i.e. all superclasses) for the first superclass that has a corresponding module.

Nota

In very exotic cases (inheritance chains spanning multiple modules created from the same definition), **`PyType_GetModuleByDef()`** might not return the module of the true defining class. However, it will always return a module with the same definition, ensuring a compatible C memory layout.

Vida útil del estado del módulo

Cuando un objeto de módulo se recolecta como basura, se libera su estado de módulo. Para cada puntero a (una parte de) el estado del módulo, debe tener una referencia al objeto del módulo.

Por lo general, esto no es un problema, porque los tipos creados con `PyType_FromModuleAndSpec()` y sus instancias contienen una referencia al módulo. Sin embargo, debe tener cuidado en el recuento de referencias cuando hace referencia al estado del módulo desde otros lugares, como devoluciones de llamada para bibliotecas externas.

Problemas abiertos

Varios problemas relacionados con el estado por módulo y los tipos heap todavía están abiertos.

Discussions about improving the situation are best held on the discuss forum under c-api tag [https://discuss.python.org/c/core-dev/c-api/30].

Alcance por clase

Actualmente (a partir de Python 3.11) no es posible adjuntar estado a *tipos* individuales sin depender de los detalles de implementación de CPython (que pueden cambiar en el futuro, tal vez, irónicamente, para permitir una solución adecuada para el alcance por clase).

Conversión sin pérdidas a tipos heap

La API de tipo heap no se diseñó para la conversión «sin pérdidas» de tipos estáticos; es decir, crear un tipo que funcione exactamente como un tipo estático determinado.

timer file descriptor HOWTO

Release:

1.13

This HOWTO discusses Python's support for the linux timer file descriptor.

Examples

The following example shows how to use a timer file descriptor to execute a function twice a second:

```
# Practical scripts should use really use a non-blocking
# we use a blocking timer here for simplicity.
import os, time

# Create the timer file descriptor
fd = os.timerfd_create(time.CLOCK_REALTIME)

# Start the timer in 1 second, with an interval of half
os.timerfd_settime(fd, initial=1, interval=0.5)

try:
    # Process timer events four times.
    for _ in range(4):
        # read() will block until the timer expires
        _ = os.read(fd, 8)
        print("Timer expired")
finally:
    # Remember to close the timer file descriptor!
    os.close(fd)
```

To avoid the precision loss caused by the `float` type, timer file descriptors allow specifying initial expiration and interval in integer nanoseconds with `_ns` variants of the functions.

This example shows how `epoll()` can be used with timer file descriptors to wait until the file descriptor is ready for reading:

```
import os, time, select, socket, sys

# Create an epoll object
ep = select.epoll()

# In this example, use loopback address to send "stop" c
#
# $ telnet 127.0.0.1 1234
# Trying 127.0.0.1...
# Connected to 127.0.0.1.
# Escape character is '^]'.
# stop
# Connection closed by foreign host.
#
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(("127.0.0.1", 1234))
sock.setblocking(False)
sock.listen(1)
ep.register(sock, select.EPOLLIN)

# Create timer file descriptors in non-blocking mode.
num = 3
fds = []
for _ in range(num):
    fd = os.timerfd_create(time.CLOCK_REALTIME, flags=os
    fds.append(fd)
    # Register the timer file descriptor for read events
    ep.register(fd, select.EPOLLIN)

# Start the timer with os.timerfd_settime_ns() in nanose
# Timer 1 fires every 0.25 seconds; timer 2 every 0.5 se
for i, fd in enumerate(fds, start=1):
    one_sec_in_nsec = 10**9
    i = i * one_sec_in_nsec
    os.timerfd_settime_ns(fd, initial=i//4, interval=i//

timeout = 3
try:
    conn = None
    is_active = True
    while is_active:
        # Wait for the timer to expire for 3 seconds.
        # epoll.poll() returns a list of (fd, event) pai
        # fd is a file descriptor.
        # sock and conn[=returned value of socket.accept
        # So use sock.fileno() and conn.fileno() to get
        events = ep.poll(timeout)

        # If more than one timer file descriptors are re
        # epoll.poll() returns a list of (fd, event) pai
        #
        # In this example settings,
        #    1st timer fires every 0.25 seconds in 0.25
        #    2nd timer every 0.5 seconds in 0.5 seconds.
        #    3rd timer every 0.75 seconds in 0.75 seconc
        #
        #    In 0.25 seconds, only 1st timer fires.
        #    In 0.5 seconds, 1st timer and 2nd timer fir
        #    In 0.75 seconds, 1st timer and 3rd timer fi
        #    In 1.5 seconds, 1st timer, 2nd timer and 3r
        #
        # If a timer file descriptor is signaled more th
        # the last os.read() call, os.read() returns the
        # as host order of class bytes.
        print(f"Signaled events={events}")
        for fd, event in events:
            if event & select.EPOLLIN:
                if fd == sock.fileno():
                    # Check if there is a connection req
                    print(f"Accepting connection {fd}")
                    conn, addr = sock.accept()
                    conn.setblocking(False)
                    print(f"Accepted connection {conn} f
                    ep.register(conn, select.EPOLLIN)
                elif conn and fd == conn.fileno():
                    # Check if there is data to read.
                    print(f"Reading data {fd}")
                    data = conn.recv(1024)
                    if data:
                        # You should catch UnicodeDecode
                        cmd = data.decode()
                        if cmd.startswith("stop"):
                            print(f"Stopping server")
                            is_active = False
                        else:
                            print(f"Unknown command: {cm
                    else:
                        # No more data, close connectio
                        print(f"Closing connection {fd}"
                        ep.unregister(conn)
                        conn.close()
                        conn = None
                elif fd in fds:
                    print(f"Reading timer {fd}")
                    count = int.from_bytes(os.read(fd, 8
                    print(f"Timer {fds.index(fd) + 1} ex
                else:
                    print(f"Unknown file descriptor {fd}
finally:
    for fd in fds:
        ep.unregister(fd)
        os.close(fd)
    ep.close()
```

This example shows how `select()` can be used with timer file descriptors to wait until the file descriptor is ready for reading:

```
import os, time, select, socket, sys

# In this example, use loopback address to send "stop" c
#
```

```
# $ telnet 127.0.0.1 1234
# Trying 127.0.0.1...
# Connected to 127.0.0.1.
# Escape character is '^]'.
# stop
# Connection closed by foreign host.
#
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(("127.0.0.1", 1234))
sock.setblocking(False)
sock.listen(1)

# Create timer file descriptors in non-blocking mode.
num = 3
fds = [os.timerfd_create(time.CLOCK_REALTIME, flags=os.T
       for _ in range(num)]
select_fds = fds + [sock]

# Start the timers with os.timerfd_settime() in seconds.
# Timer 1 fires every 0.25 seconds; timer 2 every 0.5 se
for i, fd in enumerate(fds, start=1):
   os.timerfd_settime(fd, initial=i/4, interval=i/4)

timeout = 3
try:
    conn = None
    is_active = True
    while is_active:
       # Wait for the timer to expire for 3 seconds.
       # select.select() returns a list of file descript
       rfd, wfd, xfd = select.select(select_fds, select_
       for fd in rfd:
           if fd == sock:
               # Check if there is a connection request.
               print(f"Accepting connection {fd}")
               conn, addr = sock.accept()
               conn.setblocking(False)
               print(f"Accepted connection {conn} from {
```

```
               select_fds.append(conn)
           elif conn and fd == conn:
               # Check if there is data to read.
               print(f"Reading data {fd}")
               data = conn.recv(1024)
               if data:
                   # You should catch UnicodeDecodeError
                   cmd = data.decode()
                   if cmd.startswith("stop"):
                       print(f"Stopping server")
                       is_active = False
                   else:
                       print(f"Unknown command: {cmd}")
               else:
                   # No more data, close connection
                   print(f"Closing connection {fd}")
                   select_fds.remove(conn)
                   conn.close()
                   conn = None
           elif fd in fds:
               print(f"Reading timer {fd}")
               count = int.from_bytes(os.read(fd, 8), by
               print(f"Timer {fds.index(fd) + 1} expire
           else:
               print(f"Unknown file descriptor {fd}")
finally:
    for fd in fds:
       os.close(fd)
    sock.close()
    sock = None
```

[1] [2] [3]

The Python 2.3 Method Resolution Order

Nota

This is a historical document, provided as an appendix to the official documentation. The Method Resolution Order discussed here was *introduced* in Python 2.3, but it is still used in later versions – including Python 3.

By Michele Simionato [https://www.phyast.pitt.edu/~micheles/].

Abstract:

This document is intended for Python programmers who want to understand the C3 Method Resolution Order used in Python 2.3. Although it is not intended for newbies, it is quite pedagogical with many worked out examples. I am not aware of other publicly available documents with the same scope, therefore it should be useful.

Disclaimer:

> *I donate this document to the Python Software Foundation, under the Python 2.3 license. As usual in these circumstances, I warn the reader that what follows* should *be correct, but I don't give any warranty. Use it at your own risk and peril!*

Acknowledgments:

All the people of the Python mailing list who sent me their support. Paul Foley who pointed out various imprecisions and made me to add the part on local precedence ordering. David Goodger for help with the

> *formatting in reStructuredText. David Mertz for help with the editing. Finally, Guido van Rossum who enthusiastically added this document to the official Python 2.3 home-page.*

The beginning

> *Felix qui potuit rerum cognoscere causas* – Virgilius

Everything started with a post by Samuele Pedroni to the Python development mailing list [1]. In his post, Samuele showed that the Python 2.2 method resolution order is not monotonic and he proposed to replace it with the C3 method resolution order. Guido agreed with his arguments and therefore now Python 2.3 uses C3. The C3 method itself has nothing to do with Python, since it was invented by people working on Dylan and it is described in a paper intended for lispers [2]. The present paper gives a (hopefully) readable discussion of the C3 algorithm for Pythonistas who want to understand the reasons for the change.

First of all, let me point out that what I am going to say only applies to the *new style classes* introduced in Python 2.2: *classic classes* maintain their old method resolution order, depth first and then left to right. Therefore, there is no breaking of old code for classic classes; and even if in principle there could be breaking of code for Python 2.2 new style classes, in practice the cases in which the C3 resolution order differs from the Python 2.2 method resolution order are so rare that no real breaking of code is expected. Therefore:

> *Don't be scared!*

Moreover, unless you make strong use of multiple inheritance and you have non-trivial hierarchies, you don't need to understand the C3 algorithm, and you can easily skip this paper. On the other hand, if you really want to know how multiple inheritance works, then this paper is for you. The good news is that things are not as complicated as you might expect.

Let me begin with some basic definitions.

1. Given a class C in a complicated multiple inheritance hierarchy, it is a non-trivial task to specify the order in which methods are overridden, i.e. to specify the order of the ancestors of C.
2. The list of the ancestors of a class C, including the class itself, ordered from the nearest ancestor to the furthest, is called the class precedence list or the *linearization* of C.
3. The *Method Resolution Order* (MRO) is the set of rules that construct the linearization. In the Python literature, the idiom «the MRO of C» is also used as a synonymous for the linearization of the class C.
4. For instance, in the case of single inheritance hierarchy, if C is a subclass of C1, and C1 is a subclass of C2, then the linearization of C is simply the list [C, C1 , C2]. However, with multiple inheritance hierarchies, the construction of the linearization is more cumbersome, since it is more difficult to construct a linearization that respects *local precedence ordering* and *monotonicity*.
5. I will discuss the local precedence ordering later, but I can give the definition of monotonicity here. A MRO is monotonic when the following is true: *if C1 precedes C2 in the linearization of C, then C1 precedes C2 in the linearization of any subclass of C*. Otherwise, the innocuous operation of deriving a new class could change the resolution order of methods, potentially introducing very subtle bugs. Examples where this happens will be shown later.
6. Not all classes admit a linearization. There are cases, in complicated hierarchies, where it is not possible to derive a class such that its linearization respects all the desired properties.

Here I give an example of this situation. Consider the hierarchy

```
>>> O = object
>>> class X(O): pass
>>> class Y(O): pass
>>> class A(X,Y): pass
>>> class B(Y,X): pass
```

which can be represented with the following inheritance graph, where I have denoted with O the `object` class, which is the beginning of any hierarchy for new style classes:

```
 -----------
|           |
|    O      |
|  /   \    |
 - X    Y  /
   |  / | /
   | /  |/
   A    B
   \   /
     ?
```

In this case, it is not possible to derive a new class C from A and B, since X precedes Y in A, but Y precedes X in B, therefore the method resolution order would be ambiguous in C.

Python 2.3 raises an exception in this situation (TypeError: MRO conflict among bases Y, X) forbidding the naive programmer from creating ambiguous hierarchies. Python 2.2 instead does not raise an exception, but chooses an *ad hoc* ordering (CABXYO in this case).

The C3 Method Resolution Order

Let me introduce a few simple notations which will be useful for the following discussion. I will use the shortcut notation:

```
C1 C2 ... CN
```

to indicate the list of classes [C1, C2, ... , CN].

The *head* of the list is its first element:

```
head = C1
```

whereas the *tail* is the rest of the list:

```
tail = C2 ... CN.
```

I shall also use the notation:

```
C + (C1 C2 ... CN) = C C1 C2 ... CN
```

to denote the sum of the lists [C] + [C1, C2, ... ,CN].

Now I can explain how the MRO works in Python 2.3.

Consider a class C in a multiple inheritance hierarchy, with C inheriting from the base classes B1, B2, ... , BN. We want to compute the linearization L[C] of the class C. The rule is the following:

> *the linearization of C is the sum of C plus the merge of the linearizations of the parents and the list of the parents.*

In symbolic notation:

```
L[C(B1 ... BN)] = C + merge(L[B1] ... L[BN], B1 ... BN)
```

In particular, if C is the `object` class, which has no parents, the linearization is trivial:

```
L[object] = object.
```

However, in general one has to compute the merge according to the following prescription:

> *take the head of the first list, i.e L[B1][0]; if this head is not in the tail of any of the other lists, then add it to the linearization of C and remove it from the lists in the merge, otherwise look at the head of the next list and take it, if it is a good head. Then repeat the operation until all the class are removed or it is impossible to find good heads. In this case, it is impossible to construct the merge, Python 2.3 will refuse to create the class C and will raise an exception.*

This prescription ensures that the merge operation *preserves* the ordering, if the ordering can be preserved. On the other hand, if the order cannot be preserved (as in the example of serious order disagreement discussed above) then the merge cannot be computed.

The computation of the merge is trivial if C has only one parent (single inheritance); in this case:

```
L[C(B)] = C + merge(L[B],B) = C + L[B]
```

However, in the case of multiple inheritance things are more cumbersome and I don't expect you can understand the rule without a couple of examples ;-)

Examples

First example. Consider the following hierarchy:

```
>>> O = object
>>> class F(O): pass
>>> class E(O): pass
>>> class D(O): pass
>>> class C(D,F): pass
>>> class B(D,E): pass
>>> class A(B,C): pass
```

In this case the inheritance graph can be drawn as:

```
                                6
                               ---
      Level 3                 | O |                   (mo
                            /  ---  \
                           /    |    \
                          /     |     \
                         /      |      \
                        ---    ---    ---
      Level 2        3 | D | 4| E |  | F | 5
                        ---    ---    ---
                         \  \ _ /       |
                          \    / \ _    |
                           \  /      \  |
                            ---      ---
      Level 1            1 | B |    | C | 2
                            ---      ---
                              \      /
                               \    /
                                 ---
      Level 0                 0 | A |                (mo
                                 ---
```

The linearizations of O,D,E and F are trivial:

```
L[O] = O
L[D] = D O
L[E] = E O
L[F] = F O
```

The linearization of B can be computed as:

```
L[B] = B + merge(DO, EO, DE)
```

We see that D is a good head, therefore we take it and we are reduced to compute `merge(O,EO,E)`. Now O is not a good head, since it is in the tail of the sequence EO. In this case the rule says that we have to skip to the next sequence. Then we see that E is a good head; we take it and we are reduced to compute `merge(O,O)` which gives O. Therefore:

```
L[B] =  B D E O
```

Using the same procedure one finds:

```
L[C] = C + merge(DO,FO,DF)
     = C + D + merge(O,FO,F)
     = C + D + F + merge(O,O)
     = C D F O
```

Now we can compute:

```
L[A] = A + merge(BDEO,CDFO,BC)
     = A + B + merge(DEO,CDFO,C)
     = A + B + C + merge(DEO,DFO)
     = A + B + C + D + merge(EO,FO)
```

```
    = A + B + C + D + E + merge(O,FO)
    = A + B + C + D + E + F + merge(O,O)
    = A B C D E F O
```

In this example, the linearization is ordered in a pretty nice way according to the inheritance level, in the sense that lower levels (i.e. more specialized classes) have higher precedence (see the inheritance graph). However, this is not the general case.

I leave as an exercise for the reader to compute the linearization for my second example:

```
>>> O = object
>>> class F(O): pass
>>> class E(O): pass
>>> class D(O): pass
>>> class C(D,F): pass
>>> class B(E,D): pass
>>> class A(B,C): pass
```

The only difference with the previous example is the change B(D,E) –> B(E,D); however even such a little modification completely changes the ordering of the hierarchy:

```
                                 6
                                ---
      Level 3                  | O |
                             /  ---  \
                            /    |    \
                           /     |     \
                          /      |      \
                        ---     ---    ---
      Level 2        2 | E | 4 | D |  | F | 5
                        ---     ---    ---
                         \      / \     /
                          \    /   \   /
                           \  /     \ /
                            ---     ---
      Level 1            1 | B |   | C | 3
                            ---     ---
```

```
                             \       /
                              \     /
                                ---
      Level 0                0 | A |
                                ---
```

Notice that the class E, which is in the second level of the hierarchy, precedes the class C, which is in the first level of the hierarchy, i.e. E is more specialized than C, even if it is in a higher level.

A lazy programmer can obtain the MRO directly from Python 2.2, since in this case it coincides with the Python 2.3 linearization. It is enough to invoke the `mro()` method of class A:

```
>>> A.mro()
[<class 'A'>, <class 'B'>, <class 'E'>,
<class 'C'>, <class 'D'>, <class 'F'>,
<class 'object'>]
```

Finally, let me consider the example discussed in the first section, involving a serious order disagreement. In this case, it is straightforward to compute the linearizations of O, X, Y, A and B:

```
      L[O] = 0
      L[X] = X O
      L[Y] = Y O
      L[A] = A X Y O
      L[B] = B Y X O
```

However, it is impossible to compute the linearization for a class C that inherits from A and B:

```
L[C] = C + merge(AXYO, BYXO, AB)
     = C + A + merge(XYO, BYXO, B)
     = C + A + B + merge(XYO, YXO)
```

At this point we cannot merge the lists XYO and YXO, since X is in the tail of YXO whereas Y is in the tail of XYO: therefore there are no good heads and the C3 algorithm stops. Python 2.3 raises an error and refuses to create the class C.

Bad Method Resolution Orders

A MRO is *bad* when it breaks such fundamental properties as local precedence ordering and monotonicity. In this section, I will show that both the MRO for classic classes and the MRO for new style classes in Python 2.2 are bad.

It is easier to start with the local precedence ordering. Consider the following example:

```
>>> F=type('Food',(),{'remember2buy':'spam'})
>>> E=type('Eggs',(F,),{'remember2buy':'eggs'})
>>> G=type('GoodFood',(F,E),{}) # under Python 2.3 this
```

with inheritance diagram

```
                   O
                   |
      (buy spam)   F
                   | \
                   | E   (buy eggs)
                   | /
                   G

            (buy eggs or spam ?)
```

We see that class G inherits from F and E, with F *before* E: therefore we would expect the attribute *G.remember2buy* to be inherited by *F.remember2buy* and not by *E.remember2buy*: nevertheless Python 2.2 gives

```
>>> G.remember2buy
'eggs'
```

This is a breaking of local precedence ordering since the order in the local precedence list, i.e. the list of the parents of G, is not preserved in the Python 2.2 linearization of G:

```
L[G,P22]= G E F object   # F *follows* E
```

One could argue that the reason why F follows E in the Python 2.2 linearization is that F is less specialized than E, since F is the superclass of E; nevertheless the breaking of local precedence ordering is quite non-intuitive and error prone. This is particularly true since it is a different from old style classes:

```
>>> class F: remember2buy='spam'
>>> class E(F): remember2buy='eggs'
>>> class G(F,E): pass
>>> G.remember2buy
'spam'
```

In this case the MRO is GFEF and the local precedence ordering is preserved.

As a general rule, hierarchies such as the previous one should be avoided, since it is unclear if F should override E or vice-versa. Python 2.3 solves the ambiguity by raising an exception in the creation of class G, effectively stopping the programmer from generating ambiguous hierarchies. The reason for that is that the C3 algorithm fails when the merge:

```
merge(FO,EFO,FE)
```

cannot be computed, because F is in the tail of EFO and E is in the tail of FE.

The real solution is to design a non-ambiguous hierarchy, i.e. to derive G from E and F (the more specific first) and not from F and E; in this case the MRO is GEF without any doubt.

```
                 O
                 |
                 F  (spam)
               / |
      (eggs)   E |
               \ |
                 G
                   (eggs, no doubt)
```

Python 2.3 forces the programmer to write good hierarchies (or, at least, less error-prone ones).

On a related note, let me point out that the Python 2.3 algorithm is smart enough to recognize obvious mistakes, as the duplication of classes in the list of parents:

```
>>> class A(object): pass
>>> class C(A,A): pass # error
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: duplicate base class A
```

Python 2.2 (both for classic classes and new style classes) in this situation, would not raise any exception.

Finally, I would like to point out two lessons we have learned from this example:

1. despite the name, the MRO determines the resolution order of attributes, not only of methods;
2. the default food for Pythonistas is spam ! (but you already knew that ;-)

Having discussed the issue of local precedence ordering, let me now consider the issue of monotonicity. My goal is to show that neither the MRO for classic classes nor that for Python 2.2 new style classes is monotonic.

To prove that the MRO for classic classes is non-monotonic is rather trivial, it is enough to look at the diamond diagram:

```
         C
        / \
       /   \
      A     B
       \   /
        \ /
         D
```

One easily discerns the inconsistency:

```
L[B,P21] = B C        # B precedes C : B's methods win
L[D,P21] = D A C B C  # B follows C  : C's methods win!
```

On the other hand, there are no problems with the Python 2.2 and 2.3 MROs, they give both:

```
L[D] = D A B C
```

Guido points out in his essay [3] that the classic MRO is not so bad in practice, since one can typically avoids diamonds for classic classes. But all new style classes inherit from `object`, therefore diamonds are unavoidable and inconsistencies shows up in every multiple inheritance graph.

The MRO of Python 2.2 makes breaking monotonicity difficult, but not impossible. The following example, originally provided by Samuele Pedroni, shows that the MRO of Python 2.2 is non-monotonic:

```
>>> class A(object): pass
>>> class B(object): pass
>>> class C(object): pass
>>> class D(object): pass
>>> class E(object): pass
>>> class K1(A,B,C): pass
>>> class K2(D,B,E): pass
>>> class K3(D,A):   pass
>>> class Z(K1,K2,K3): pass
```

Here are the linearizations according to the C3 MRO (the reader should verify these linearizations as an exercise and draw the inheritance diagram ;-)

```
L[A] = A O
L[B] = B O
L[C] = C O
L[D] = D O
L[E] = E O
L[K1]= K1 A B C O
L[K2]= K2 D B E O
```

```
L[K3]= K3 D A O
L[Z] = Z K1 K2 K3 D A B C E O
```

Python 2.2 gives exactly the same linearizations for A, B, C, D, E, K1, K2 and K3, but a different linearization for Z:

```
L[Z,P22] = Z K1 K3 A K2 D B C E O
```

It is clear that this linearization is *wrong*, since A comes before D whereas in the linearization of K3 A comes *after* D. In other words, in K3 methods derived by D override methods derived by A, but in Z, which still is a subclass of K3, methods derived by A override methods derived by D! This is a violation of monotonicity. Moreover, the Python 2.2 linearization of Z is also inconsistent with local precedence ordering, since the local precedence list of the class Z is [K1, K2, K3] (K2 precedes K3), whereas in the linearization of Z K2 *follows* K3. These problems explain why the 2.2 rule has been dismissed in favor of the C3 rule.

The end

This section is for the impatient reader, who skipped all the previous sections and jumped immediately to the end. This section is for the lazy programmer too, who didn't want to exercise her/his brain. Finally, it is for the programmer with some hubris, otherwise s/he would not be reading a paper on the C3 method resolution order in multiple inheritance hierarchies ;-) These three virtues taken all together (and *not* separately) deserve a prize: the prize is a short Python 2.2 script that allows you to compute the 2.3 MRO without risk to your brain. Simply change the last line to play with the various examples I have discussed in this paper.:

```
#<mro.py>

"""C3 algorithm by Samuele Pedroni (with readability enh

class __metaclass__(type):
    "All classes are metamagically modified to be nicely
    __repr__ = lambda cls: cls.__name__
```

```
class ex_2:
    "Serious order disagreement" #From Guido
    class O: pass
    class X(O): pass
    class Y(O): pass
    class A(X,Y): pass
    class B(Y,X): pass
    try:
        class Z(A,B): pass #creates Z(A,B) in Python 2.2
    except TypeError:
        pass # Z(A,B) cannot be created in Python 2.3

class ex_5:
    "My first example"
    class O: pass
    class F(O): pass
    class E(O): pass
    class D(O): pass
    class C(D,F): pass
    class B(D,E): pass
    class A(B,C): pass

class ex_6:
    "My second example"
    class O: pass
    class F(O): pass
    class E(O): pass
    class D(O): pass
    class C(D,F): pass
    class B(E,D): pass
    class A(B,C): pass

class ex_9:
    "Difference between Python 2.2 MRO and C3" #From Sam
    class O: pass
    class A(O): pass
    class B(O): pass
    class C(O): pass
    class D(O): pass
    class E(O): pass
    class K1(A,B,C): pass
    class K2(D,B,E): pass
    class K3(D,A): pass
    class Z(K1,K2,K3): pass

def merge(seqs):
    print '\n\nCPL[%s]=%s' % (seqs[0][0],seqs),
    res = []; i=0
    while 1:
      nonemptyseqs=[seq for seq in seqs if seq]
      if not nonemptyseqs: return res
      i+=1; print '\n',i,'round: candidates...',
      for seq in nonemptyseqs: # find merge candidates a
          cand = seq[0]; print ' ',cand,
          nothead=[s for s in nonemptyseqs if cand in s[
          if nothead: cand=None #reject candidate
          else: break
      if not cand: raise "Inconsistent hierarchy"
      res.append(cand)
      for seq in nonemptyseqs: # remove cand
          if seq[0] == cand: del seq[0]

def mro(C):
    "Compute the class precedence list (mro) according t
    return merge([[C]]+map(mro,C.__bases__)+[list(C.__ba

def print_mro(C):
    print '\nMRO[%s]=%s' % (C,mro(C))
    print '\nP22 MRO[%s]=%s' % (C,C.mro())

print_mro(ex_9.Z)

#</mro.py>
```

That’s all folks,

enjoy !

Resources

The thread on python-dev started by Samuele Pedroni: https://mail.python.org/pipermail/python-dev/2002-October/029035.html

The paper *A Monotonic Superclass Linearization for Dylan*: https://doi.org/10.1145/236337.236343

Guido van Rossum's essay, *Unifying types and classes in Python 2.2*: https://web.archive.org/web/20140210194412/http://www.python.org/download/releases/2.2.2/descrintro

Python support for free threading

Starting with the 3.13 release, CPython has support for a build of Python called free threading where the global interpreter lock (GIL) is disabled. Free-threaded execution allows for full utilization of the available processing power by running threads in parallel on available CPU cores. While not all software will benefit from this automatically, programs designed with threading in mind will run faster on multi-core hardware.

Some third-party packages, in particular ones with an extension module, may not be ready for use in a free-threaded build, and will re-enable the GIL.

This document describes the implications of free threading for Python code. See C API Extension Support for Free Threading for information on how to write C extensions that support the free-threaded build.

Ver también

PEP 703 [https://peps.python.org/pep-0703/] – Making the Global Interpreter Lock Optional in CPython for an overall description of free-threaded Python.

Installation

Starting with Python 3.13, the official macOS and Windows installers optionally support installing free-threaded Python binaries. The installers are available at https://www.python.org/downloads/.

For information on other platforms, see the Installing a Free-

Threaded Python [https://py-free-threading.github.io/installing-cpython/], a community-maintained installation guide for installing free-threaded Python.

When building CPython from source, the `--disable-gil` configure option should be used to build a free-threaded Python interpreter.

Identifying free-threaded Python

To check if the current interpreter supports free-threading, `python -VV` and `sys.version` contain «free-threading build». The new `sys._is_gil_enabled()` function can be used to check whether the GIL is actually disabled in the running process.

The `sysconfig.get_config_var("Py_GIL_DISABLED")` configuration variable can be used to determine whether the build supports free threading. If the variable is set to `1`, then the build supports free threading. This is the recommended mechanism for decisions related to the build configuration.

The global interpreter lock in free-threaded Python

Free-threaded builds of CPython support optionally running with the GIL enabled at runtime using the environment variable `PYTHON_GIL` or the command-line option `-X gil`.

The GIL may also automatically be enabled when importing a C-API extension module that is not explicitly marked as supporting free threading. A warning will be printed in this case.

In addition to individual package documentation, the following websites track the status of popular packages support for free threading:

- https://py-free-threading.github.io/tracking/
- https://hugovk.github.io/free-threaded-wheels/

Thread safety

The free-threaded build of CPython aims to provide similar thread-safety behavior at the Python level to the default GIL-enabled build. Built-in types like `dict`, `list`, and `set` use internal locks to protect against concurrent modifications in ways that behave similarly to the GIL. However, Python has not historically guaranteed specific behavior for concurrent modifications to these built-in types, so this should be treated as a description of the current implementation, not a guarantee of current or future behavior.

Nota

It’s recommended to use the `threading.Lock` or other synchronization primitives instead of relying on the internal locks of built-in types, when possible.

Known limitations

This section describes known limitations of the free-threaded CPython build.

Immortalization

In the free-threaded build, some objects are immortal. Immortal objects are not deallocated and have reference counts that are never modified. This is done to avoid reference count contention that would prevent efficient multi-threaded scaling.

As of the 3.14 release, immortalization is limited to:

- Code constants: numeric literals, string literals, and tuple literals composed of other constants.
- Strings interned by `sys.intern()`.

Frame objects

It is not safe to access `frame.f_locals` from a frame object if that frame is currently executing in another thread, and doing so may crash the interpreter.

Iterators

It is generally not thread-safe to access the same iterator object from multiple threads concurrently, and threads may see duplicate or missing elements.

Single-threaded performance

The free-threaded build has additional overhead when executing Python code compared to the default GIL-enabled build. The amount of overhead depends on the workload and hardware. On the pyperformance benchmark suite, the average overhead ranges from about 1% on macOS aarch64 to 8% on x86-64 Linux systems.

Behavioral changes

This section describes CPython behavioural changes with the free-threaded build.

Context variables

In the free-threaded build, the flag `thread_inherit_context` is set to true by default which causes threads created with `threading.Thread` to start with a copy of the `Context()` of the caller of `start()`. In the default GIL-enabled build, the flag defaults to false so threads start with an empty `Context()`.

Warning filters

In the free-threaded build, the flag `context_aware_warnings` is set to true by default. In the default GIL-enabled build, the flag defaults to false. If the flag is true then the `warnings.catch_warnings` context manager uses a context variable for warning filters. If the flag is false then `catch_warnings` modifies the global filters list, which is not thread-safe. See the `warnings` module for more details.

C API Extension Support for Free Threading

Starting with the 3.13 release, CPython has support for running with the global interpreter lock (GIL) disabled in a configuration called free threading. This document describes how to adapt C API extensions to support free threading.

Identifying the Free-Threaded Build in C

The CPython C API exposes the `Py_GIL_DISABLED` macro: in the free-threaded build it's defined to `1`, and in the regular build it's not defined. You can use it to enable code that only runs under the free-threaded build:

```
#ifdef Py_GIL_DISABLED
/* code that only runs in the free-threaded build */
#endif
```

Nota

On Windows, this macro is not defined automatically, but must be specified to the compiler when building. The **`sysconfig.get_config_var()`** function can be used to determine whether the current running interpreter had the macro defined.

Module Initialization

Extension modules need to explicitly indicate that they support running with the GIL disabled; otherwise importing the extension will raise a warning and enable the GIL at runtime.

There are two ways to indicate that an extension module supports running with the GIL disabled depending on whether the extension uses multi-phase or single-phase initialization.

Multi-Phase Initialization

Extensions that use multi-phase initialization (i.e., **`PyModuleDef_Init()`**) should add a **`Py_mod_gil`** slot in the module definition. If your extension supports older versions of CPython, you should guard the slot with a **`PY_VERSION_HEX`** check.

```
static struct PyModuleDef_Slot module_slots[] = {
    ...
#if PY_VERSION_HEX >= 0x030D0000
    {Py_mod_gil, Py_MOD_GIL_NOT_USED},
#endif
    {0, NULL}
};

static struct PyModuleDef moduledef = {
    PyModuleDef_HEAD_INIT,
    .m_slots = module_slots,
    ...
};
```

Single-Phase Initialization

Extensions that use single-phase initialization (i.e., **`PyModule_Create()`**) should call **`PyUnstable_Module_SetGIL()`** to indicate that they support running with the GIL disabled. The function is only defined in the free-threaded build, so you should guard the call with `#ifdef Py_GIL_DISABLED` to avoid compilation errors in the regular build.

```
static struct PyModuleDef moduledef = {
    PyModuleDef_HEAD_INIT,
    ...
};

PyMODINIT_FUNC
PyInit_mymodule(void)
{
    PyObject *m = PyModule_Create(&moduledef);
    if (m == NULL) {
        return NULL;
    }
#ifdef Py_GIL_DISABLED
    PyUnstable_Module_SetGIL(m, Py_MOD_GIL_NOT_USED);
#endif
    return m;
}
```

General API Guidelines

Most of the C API is thread-safe, but there are some exceptions.

- **Struct Fields**: Accessing fields in Python C API objects or structs directly is not thread-safe if the field may be concurrently modified.
- **Macros**: Accessor macros like `PyList_GET_ITEM`, `PyList_SET_ITEM`, and macros like `PySequence_Fast_GET_SIZE` that use the object returned by `PySequence_Fast()` do not perform any error checking or locking. These macros are not thread-safe if the container object may be modified concurrently.
- **Borrowed References**: C API functions that return borrowed references may not be thread-safe if the containing object is modified concurrently. See the section on borrowed references for more information.

Container Thread Safety

Containers like `PyListObject`, `PyDictObject`, and `PySetObject` perform internal locking in the free-threaded build. For example, the `PyList_Append()` will lock the list before appending an item.

PyDict_Next

A notable exception is **`PyDict_Next()`**, which does not lock the dictionary. You should use **`Py_BEGIN_CRITICAL_SECTION`** to protect the dictionary while iterating over it if the dictionary may be concurrently modified:

```
Py_BEGIN_CRITICAL_SECTION(dict);
PyObject *key, *value;
Py_ssize_t pos = 0;
while (PyDict_Next(dict, &pos, &key, &value)) {
    ...
}
Py_END_CRITICAL_SECTION();
```

Borrowed References

Some C API functions return borrowed references. These APIs are not thread-safe if the containing object is modified concurrently. For example, it's not safe to use **`PyList_GetItem()`** if the list may be modified concurrently.

The following table lists some borrowed reference APIs and their replacements that return strong references.

Borrowed reference API	Strong reference API
`PyList_GetItem()`	**`PyList_GetItemRef()`**
`PyList_GET_ITEM()`	**`PyList_GetItemRef()`**
`PyDict_GetItem()`	**`PyDict_GetItemRef()`**
`PyDict_GetItemWithError()`	**`PyDict_GetItemRef()`**
`PyDict_GetItemString()`	**`PyDict_GetItemStringRef()`**
`PyDict_SetDefault()`	**`PyDict_SetDefaultRef()`**
`PyDict_Next()`	none (see PyDict_Next)
`PyWeakref_GetObject()`	**`PyWeakref_GetRef()`**
`PyWeakref_GET_OBJECT()`	**`PyWeakref_GetRef()`**
`PyImport_AddModule()`	**`PyImport_AddModuleRef()`**
`PyCell_GET()`	**`PyCell_Get()`**

Not all APIs that return borrowed references are problematic. For example, **`PyTuple_GetItem()`** is safe because tuples are immutable. Similarly, not all uses of the above APIs are problematic. For example, `PyDict_GetItem()` is often used for parsing keyword argument dictionaries in function calls; those keyword argument dictionaries are effectively private (not accessible by other threads), so using borrowed references in that context is safe.

Some of these functions were added in Python 3.13. You can use the pythoncapi-compat [https://github.com/python/pythoncapi-compat] package to provide implementations of these functions for older Python versions.

Memory Allocation APIs

Python's memory management C API provides functions in three different allocation domains: «raw», «mem», and «object». For thread-safety, the free-threaded build requires that only Python objects are allocated using the object domain, and that all Python objects are allocated using that domain. This differs from the prior Python versions, where this was only a best practice and not a hard requirement.

Nota

Search for uses of `PyObject_Malloc()` in your extension and check that the allocated memory is used for Python objects. Use `PyMem_Malloc()` to allocate buffers instead of `PyObject_Malloc()`.

Thread State and GIL APIs

Python provides a set of functions and macros to manage thread state and the GIL, such as:

- `PyGILState_Ensure()` and `PyGILState_Release()`
- `PyEval_SaveThread()` and `PyEval_RestoreThread()`
- `Py_BEGIN_ALLOW_THREADS` and `Py_END_ALLOW_THREADS`

These functions should still be used in the free-threaded build to manage thread state even when the GIL is disabled. For example, if you create a thread outside of Python, you must call `PyGILState_Ensure()` before calling into the Python API to ensure that the thread has a valid Python thread state.

You should continue to call `PyEval_SaveThread()` or `Py_BEGIN_ALLOW_THREADS` around blocking operations, such as I/O or lock acquisitions, to allow other threads to run the cyclic garbage collector.

Protecting Internal Extension State

Your extension may have internal state that was previously protected by the GIL. You may need to add locking to protect this state. The approach will depend on your extension, but some common patterns include:

- **Caches**: global caches are a common source of shared state. Consider using a lock to protect the cache or disabling it in the free-threaded build if the cache is not critical for performance.
- **Global State**: global state may need to be protected by a lock or moved to thread local storage. C11 and C++11 provide the `thread_local` or `_Thread_local` for thread-local storage [https://en.cppreference.com/w/c/language/storage_duration].

Critical Sections

In the free-threaded build, CPython provides a mechanism called «critical sections» to protect data that would otherwise be protected by the GIL. While extension authors may not interact with the internal critical section implementation directly, understanding their behavior is crucial when using certain C API functions or managing shared state in the free-threaded build.

What Are Critical Sections?

Conceptually, critical sections act as a deadlock avoidance layer built on top of simple mutexes. Each thread maintains a stack of active critical sections. When a thread needs to acquire a lock associated with a critical section (e.g., implicitly when calling a thread-safe C API function like `PyDict_SetItem()`, or explicitly using macros), it attempts to acquire the underlying mutex.

Using Critical Sections

The primary APIs for using critical sections are:

- `Py_BEGIN_CRITICAL_SECTION` and `Py_END_CRITICAL_SECTION` - For locking a single object
- `Py_BEGIN_CRITICAL_SECTION2` and `Py_END_CRITICAL_SECTION2` - For locking two objects simultaneously

These macros must be used in matching pairs and must appear in the same C scope, since they establish a new local scope. These macros are no-ops in non-free-threaded builds, so they can be safely added to code that needs to support both build types.

A common use of a critical section would be to lock an object while accessing an internal attribute of it. For example, if an extension type has an internal count field, you could use a critical section while reading or writing that field:

```
// read the count, returns new reference to internal cou
PyObject *result;
Py_BEGIN_CRITICAL_SECTION(obj);
result = Py_NewRef(obj->count);
Py_END_CRITICAL_SECTION();
return result;

// write the count, consumes reference from new_count
Py_BEGIN_CRITICAL_SECTION(obj);
obj->count = new_count;
Py_END_CRITICAL_SECTION();
```

How Critical Sections Work

Unlike traditional locks, critical sections do not guarantee exclusive access throughout their entire duration. If a thread would block while holding a critical section (e.g., by acquiring another lock or performing I/O), the critical section is temporarily suspended—all locks are released—and then resumed when the blocking operation completes.

This behavior is similar to what happens with the GIL when a thread makes a blocking call. The key differences are:

- Critical sections operate on a per-object basis rather than globally
- Critical sections follow a stack discipline within each thread (the «begin» and «end» macros enforce this since they must be paired and within the same scope)
- Critical sections automatically release and reacquire locks around potential blocking operations

Deadlock Avoidance

Critical sections help avoid deadlocks in two ways:

1. If a thread tries to acquire a lock that's already held by another thread, it first suspends all of its active critical sections, temporarily releasing their locks
2. When the blocking operation completes, only the top-most critical section is reacquired first

This means you cannot rely on nested critical sections to lock multiple objects at once, as the inner critical section may suspend the outer ones. Instead, use `Py_BEGIN_CRITICAL_SECTION2` to lock two objects simultaneously.

Note that the locks described above are only `PyMutex` based locks. The critical section implementation does not know about or affect other locking mechanisms that might be in use, like POSIX mutexes. Also note that while blocking on any `PyMutex` causes the critical sections to be suspended, only the mutexes that are part of the critical sections are released. If `PyMutex` is used without a critical section, it will not be released and therefore does not get the same deadlock avoidance.

Important Considerations

- Critical sections may temporarily release their locks, allowing other threads to modify the protected data. Be careful about making assumptions about the state of the data after operations that might block.
- Because locks can be temporarily released (suspended), entering a critical section does not guarantee exclusive access to the protected resource throughout the section's duration. If code within a critical section calls another function that blocks (e.g., acquires another lock, performs blocking I/O), all locks held by the thread via critical sections will be released. This is similar to how the GIL can be released during blocking calls.
- Only the lock(s) associated with the most recently entered (top-most) critical section are guaranteed to be held at any given time. Locks for outer, nested critical sections might have been suspended.
- You can lock at most two objects simultaneously with these APIs. If you need to lock more objects, you'll need to restructure your code.
- While critical sections will not deadlock if you attempt to lock the same object twice, they are less efficient than purpose-built reentrant locks for this use case.
- When using `Py_BEGIN_CRITICAL_SECTION2`, the order of the objects doesn't affect correctness (the implementation handles deadlock avoidance), but it's good practice to always lock objects in a consistent order.
- Remember that the critical section macros are primarily for protecting access to *Python objects* that might be involved in internal CPython operations susceptible to the deadlock scenarios described above. For protecting purely internal extension state, standard mutexes or other synchronization primitives might be more appropriate.

Building Extensions for the Free-Threaded Build

C API extensions need to be built specifically for the free-threaded build. The wheels, shared libraries, and binaries are indicated by a `t` suffix.

- pypa/manylinux [https://github.com/pypa/manylinux] supports the free-threaded build, with the `t` suffix, such as `python3.14t`.
- pypa/cibuildwheel [https://github.com/pypa/cibuildwheel] supports building wheels for the free-threaded build of Python 3.14 and newer.

Limited C API and Stable ABI

The free-threaded build does not currently support the Limited C API or the stable ABI. If you use setuptools [https://setuptools.pypa.io/en/latest/setuptools.html] to build your extension and currently set `py_limited_api=True` you can use `py_limited_api=not sysconfig.get_config_var("Py_GIL_DISABLED")` to opt out of the limited API when building with the free-threaded build.

Nota

You will need to build separate wheels specifically for the free-threaded build. If you currently use the stable ABI, you can continue to build a single wheel for multiple non-free-threaded Python versions.

Windows

Due to a limitation of the official Windows installer, you will need to manually define `Py_GIL_DISABLED=1` when building extensions from source.

Ver también

Porting Extension Modules to Support Free-Threading [https://py-free-threading.github.io/porting/]: A community-maintained porting guide for extension authors.

Remote debugging attachment protocol

This protocol enables external tools to attach to a running CPython process and execute Python code remotely.

Most platforms require elevated privileges to attach to another Python process.

Disabling remote debugging

To disable remote debugging support, use any of the following:

- Set the `PYTHON_DISABLE_REMOTE_DEBUG` environment variable to `1` before starting the interpreter.
- Use the `-X disable_remote_debug` command-line option.
- Compile Python with the `--without-remote-debug` build flag.

Permission requirements

Attaching to a running Python process for remote debugging requires elevated privileges on most platforms. The specific requirements and troubleshooting steps depend on your operating system:

Linux

The tracer process must have the `CAP_SYS_PTRACE` capability or equivalent privileges. You can only trace processes you own and can signal. Tracing may fail if the process is already being traced, or if it is running with set-user-ID or set-group-ID. Security modules like Yama may further restrict tracing.

To temporarily relax ptrace restrictions (until reboot), run:

```
echo 0 | sudo tee /proc/sys/kernel/
yama/ptrace_scope
```

Nota

Disabling `ptrace_scope` reduces system hardening and should only be done in trusted environments.

If running inside a container, use `--cap-add=SYS_PTRACE` or `--privileged`, and run as root if needed.

Try re-running the command with elevated privileges:

```
sudo -E !!
```

macOS

To attach to another process, you typically need to run your debugging tool with elevated privileges. This can be done by using `sudo` or running as root.

Even when attaching to processes you own, macOS may block debugging unless the debugger is run with root privileges due to system security restrictions.

Windows

To attach to another process, you usually need to run your debugging tool with administrative privileges. Start the command prompt or terminal as Administrator.

Some processes may still be inaccessible even with Administrator rights, unless you have the `SeDebugPrivilege` privilege enabled.

To resolve file or folder access issues, adjust the security permissions:

1. Right-click the file or folder and select **Properties**.
2. Go to the **Security** tab to view users and groups with access.
3. Click **Edit** to modify permissions.
4. Select your user account.
5. In **Permissions**, check **Read** or **Full control** as needed.
6. Click **Apply**, then **OK** to confirm.

Nota

Ensure you've satisfied all Permission requirements before proceeding.

This section describes the low-level protocol that enables external tools to inject and execute a Python script within a running CPython process.

This mechanism forms the basis of the **`sys.remote_exec()`** function, which instructs a remote Python process to execute a `.py` file. However, this section does not document the usage of that function. Instead, it provides a detailed explanation of the underlying protocol, which takes as input the `pid` of a target Python process and the path to a Python source file to be executed. This information supports independent reimplementation of the protocol, regardless of programming language.

Advertencia

The execution of the injected script depends on the interpreter reaching a safe evaluation point. As a result, execution may be delayed depending on the runtime state of the target process.

Once injected, the script is executed by the interpreter within the target process the next time a safe evaluation point is reached. This approach enables remote execution capabilities without modifying the behavior or structure of the running Python application.

Subsequent sections provide a step-by-step description of the protocol, including techniques for locating interpreter structures in memory, safely accessing internal fields, and triggering code execution. Platform-specific variations are noted where applicable, and example implementations are included to clarify each operation.

Locating the PyRuntime structure

CPython places the `PyRuntime` structure in a dedicated binary section to help external tools find it at runtime. The name and format of this section vary by platform. For example, `.PyRuntime` is used on ELF systems, and `__DATA,__PyRuntime` is used on macOS. Tools can find the offset of this structure by examining the binary on disk.

The `PyRuntime` structure contains CPython's global interpreter state and provides access to other internal data, including the list of interpreters, thread states, and debugger support fields.

To work with a remote Python process, a debugger must first find the memory address of the `PyRuntime` structure in the target process. This address can't be hardcoded or calculated from a symbol name, because it depends on where the operating system loaded the binary.

The method for finding `PyRuntime` depends on the platform, but the steps are the same in general:

1. Find the base address where the Python binary or shared library was loaded in the target process.

2. Use the on-disk binary to locate the offset of the `.PyRuntime` section.
3. Add the section offset to the base address to compute the address in memory.

The sections below explain how to do this on each supported platform and include example code.

Linux (ELF)

To find the `PyRuntime` structure on Linux:

1. Read the process's memory map (for example, `/proc/<pid>/maps`) to find the address where the Python executable or `libpython` was loaded.
2. Parse the ELF section headers in the binary to get the offset of the `.PyRuntime` section.
3. Add that offset to the base address from step 1 to get the memory address of `PyRuntime`.

The following is an example implementation:

```
def find_py_runtime_linux(pid: int) -> int:
    # Step 1: Try to find the Python executable in memor
    binary_path, base_address = find_mapped_binary(
        pid, name_contains="python"
    )

    # Step 2: Fallback to shared library if executable i
    if binary_path is None:
        binary_path, base_address = find_mapped_binary(
            pid, name_contains="libpython"
        )

    # Step 3: Parse ELF headers to get .PyRuntime sectic
    section_offset = parse_elf_section_offset(
        binary_path, ".PyRuntime"
    )
    # Step 4: Compute PyRuntime address in memory
    return base_address + section_offset
```

On Linux systems, there are two main approaches to read memory from another process. The first is through the `/proc` filesystem, specifically by reading from `/proc/[pid]/mem` which provides direct access to the process's memory. This requires appropriate permissions - either being the same user as the target process or having root access. The second approach is using the `process_vm_readv()` system call which provides a more efficient way to copy memory between processes. While ptrace's `PTRACE_PEEKTEXT` operation can also be used to read memory, it is significantly slower as it only reads one word at a time and requires multiple context switches between the tracer and tracee processes.

For parsing ELF sections, the process involves reading and interpreting the ELF file format structures from the binary file on disk. The ELF header contains a pointer to the section header table. Each section header contains metadata about a section including its name (stored in a separate string table), offset, and size. To find a specific section like .PyRuntime, you need to walk through these headers and match the section name. The section header then provides the offset where that section exists in the file, which can be used to calculate its runtime address when the binary is loaded into memory.

You can read more about the ELF file format in the ELF specification [https://en.wikipedia.org/wiki/Executable_and_Linkable_Format].

macOS (Mach-O)

To find the `PyRuntime` structure on macOS:

1. Call `task_for_pid()` to get the `mach_port_t` task port for the target process. This handle is needed to read memory using APIs like `mach_vm_read_overwrite` and `mach_vm_region`.
2. Scan the memory regions to find the one containing the

Python executable or `libpython`.

3. Load the binary file from disk and parse the Mach-O headers to find the section named `PyRuntime` in the `__DATA` segment. On macOS, symbol names are automatically prefixed with an underscore, so the `PyRuntime` symbol appears as `_PyRuntime` in the symbol table, but the section name is not affected.

The following is an example implementation:

```
def find_py_runtime_macos(pid: int) -> int:
    # Step 1: Get access to the process's memory
    handle = get_memory_access_handle(pid)

    # Step 2: Try to find the Python executable in memor
    binary_path, base_address = find_mapped_binary(
        handle, name_contains="python"
    )

    # Step 3: Fallback to libpython if the executable is
    if binary_path is None:
        binary_path, base_address = find_mapped_binary(
            handle, name_contains="libpython"
        )

    # Step 4: Parse Mach-O headers to get __DATA,__PyRun
    section_offset = parse_macho_section_offset(
        binary_path, "__DATA", "__PyRuntime"
    )

    # Step 5: Compute the PyRuntime address in memory
    return base_address + section_offset
```

On macOS, accessing another process's memory requires using Mach-O specific APIs and file formats. The first step is obtaining a `task_port` handle via `task_for_pid()`, which provides access to the target process's memory space. This handle enables memory operations through APIs like `mach_vm_read_overwrite()`.

The process memory can be examined using `mach_vm_region()`

to scan through the virtual memory space, while `proc_regionfilename()` helps identify which binary files are loaded at each memory region. When the Python binary or library is found, its Mach-O headers need to be parsed to locate the `PyRuntime` structure.

The Mach-O format organizes code and data into segments and sections. The `PyRuntime` structure lives in a section named `__PyRuntime` within the `__DATA` segment. The actual runtime address calculation involves finding the `__TEXT` segment which serves as the binary's base address, then locating the `__DATA` segment containing our target section. The final address is computed by combining the base address with the appropriate section offsets from the Mach-O headers.

Note that accessing another process's memory on macOS typically requires elevated privileges - either root access or special security entitlements granted to the debugging process.

Windows (PE)

To find the `PyRuntime` structure on Windows:

1. Use the ToolHelp API to enumerate all modules loaded in the target process. This is done using functions such as CreateToolhelp32Snapshot [https://learn.microsoft.com/en-us/windows/win32/api/tlhelp32/nf-tlhelp32-createtoolhelp32snapshot], Module32First [https://learn.microsoft.com/en-us/windows/win32/api/tlhelp32/nf-tlhelp32-module32first], and Module32Next [https://learn.microsoft.com/en-us/windows/win32/api/tlhelp32/nf-tlhelp32-module32next].
2. Identify the module corresponding to `python.exe` or `pythonXY.dll`, where `X` and `Y` are the major and minor version numbers of the Python version, and record its base address.
3. Locate the `PyRuntim` section. Due to the PE format's 8-character limit on section names (defined as `IMAGE_SIZEOF_SHORT_NAME`), the original name `PyRuntime` is truncated. This section contains the `PyRuntime` structure.

4. Retrieve the section's relative virtual address (RVA) and add it to the base address of the module.

The following is an example implementation:

```
def find_py_runtime_windows(pid: int) -> int:
    # Step 1: Try to find the Python executable in memor
    binary_path, base_address = find_loaded_module(
        pid, name_contains="python"
    )

    # Step 2: Fallback to shared pythonXY.dll if the exe
    # found
    if binary_path is None:
        binary_path, base_address = find_loaded_module(
            pid, name_contains="python3"
        )

    # Step 3: Parse PE section headers to get the RVA of
    # section. The section name appears as "PyRuntim" du
    # 8-character limit defined by the PE format (IMAGE_
    section_rva = parse_pe_section_offset(binary_path, "

    # Step 4: Compute PyRuntime address in memory
    return base_address + section_rva
```

On Windows, accessing another process's memory requires using the Windows API functions like `CreateToolhelp32Snapshot()` and `Module32First()/Module32Next()` to enumerate loaded modules. The `OpenProcess()` function provides a handle to access the target process's memory space, enabling memory operations through `ReadProcessMemory()`.

The process memory can be examined by enumerating loaded modules to find the Python binary or DLL. When found, its PE headers need to be parsed to locate the `PyRuntime` structure.

The PE format organizes code and data into sections. The `PyRuntime` structure lives in a section named «PyRuntim» (truncated from «PyRuntime» due to PE's 8-character name limit).

The actual runtime address calculation involves finding the module's base address from the module entry, then locating our target section in the PE headers. The final address is computed by combining the base address with the section's virtual address from the PE section headers.

Note that accessing another process's memory on Windows typically requires appropriate privileges - either administrative access or the `SeDebugPrivilege` privilege granted to the debugging process.

Reading _Py_DebugOffsets

Once the address of the `PyRuntime` structure has been determined, the next step is to read the `_Py_DebugOffsets` structure located at the beginning of the `PyRuntime` block.

This structure provides version-specific field offsets that are needed to safely read interpreter and thread state memory. These offsets vary between CPython versions and must be checked before use to ensure they are compatible.

To read and check the debug offsets, follow these steps:

1. Read memory from the target process starting at the `PyRuntime` address, covering the same number of bytes as the `_Py_DebugOffsets` structure. This structure is located at the very start of the `PyRuntime` memory block. Its layout is defined in CPython's internal headers and stays the same within a given minor version, but may change in major versions.
2. Check that the structure contains valid data:
    - The `cookie` field must match the expected debug marker.
    - The `version` field must match the version of the Python interpreter used by the debugger.
    - If either the debugger or the target process is using a pre-release version (for example, an alpha, beta, or release candidate), the versions must match exactly.

  - The `free_threaded` field must have the same value in both the debugger and the target process.

7. If the structure is valid, the offsets it contains can be used to locate fields in memory. If any check fails, the debugger should stop the operation to avoid reading memory in the wrong format.

The following is an example implementation that reads and checks `_Py_DebugOffsets`:

```
def read_debug_offsets(pid: int, py_runtime_addr: int) -
    # Step 1: Read memory from the target process at the
    data = read_process_memory(
        pid, address=py_runtime_addr, size=DEBUG_OFFSETS
    )

    # Step 2: Deserialize the raw bytes into a _Py_Debug
    debug_offsets = parse_debug_offsets(data)

    # Step 3: Validate the contents of the structure
    if debug_offsets.cookie != EXPECTED_COOKIE:
        raise RuntimeError("Invalid or missing debug co
    if debug_offsets.version != LOCAL_PYTHON_VERSION:
        raise RuntimeError(
            "Mismatch between caller and target Python
        )
    if debug_offsets.free_threaded != LOCAL_FREE_THREADE
        raise RuntimeError("Mismatch in free-threaded c

    return debug_offsets
```

Advertencia

Process suspension recommended

To avoid race conditions and ensure memory consistency, it is strongly recommended that the target process be suspended before performing any operations that read or write internal interpreter state. The Python runtime may concurrently mutate interpreter data structures—such as creating or destroying threads—during normal execution. This can result in invalid memory reads or writes.

A debugger may suspend execution by attaching to the process with `ptrace` or by sending a `SIGSTOP` signal. Execution should only be resumed after debugger-side memory operations are complete.

Nota

Some tools, such as profilers or sampling-based debuggers, may operate on a running process without suspension. In such cases, tools must be explicitly designed to handle partially updated or inconsistent memory. For most debugger implementations, suspending the process remains the safest and most robust approach.

Locating the interpreter and thread state

Before code can be injected and executed in a remote Python process, the debugger must choose a thread in which to schedule execution. This is necessary because the control fields used to perform remote code injection are located in the `_PyRemoteDebuggerSupport` structure, which is embedded in a `PyThreadState` object. These fields are modified by the debugger to request execution of injected scripts.

The `PyThreadState` structure represents a thread running inside a Python interpreter. It maintains the thread's evaluation context and contains the fields required for debugger coordination. Locating a valid `PyThreadState` is therefore a key prerequisite for triggering execution remotely.

A thread is typically selected based on its role or ID. In most cases, the main thread is used, but some tools may target a specific thread by its native thread ID. Once the target thread is chosen, the debugger must locate both the interpreter and the associated thread state structures in memory.

The relevant internal structures are defined as follows:

- `PyInterpreterState` represents an isolated Python interpreter instance. Each interpreter maintains its own set of imported modules, built-in state, and thread state list. Although most Python applications use a single interpreter, CPython supports multiple interpreters in the same process.
- `PyThreadState` represents a thread running within an interpreter. It contains execution state and the control fields used by the debugger.

To locate a thread:

1. Use the offset `runtime_state.interpreters_head` to obtain the address of the first interpreter in the `PyRuntime` structure. This is the entry point to the linked list of active interpreters.
2. Use the offset `interpreter_state.threads_main` to access the main thread state associated with the selected interpreter. This is typically the most reliable thread to target.
3. Optionally, use the offset `interpreter_state.threads_head` to iterate through the linked list of all thread states. Each `PyThreadState` structure contains a `native_thread_id` field, which may be compared to a target thread ID to find a specific thread.
4. Once a valid `PyThreadState` has been found, its address can be used in later steps of the protocol, such as writing debugger control fields and scheduling execution.

The following is an example implementation that locates the main thread state:

```
def find_main_thread_state(
    pid: int, py_runtime_addr: int, debug_offsets: Debug
) -> int:
    # Step 1: Read interpreters_head from PyRuntime
    interp_head_ptr = (
        py_runtime_addr + debug_offsets.runtime_state.in
    )
    interp_addr = read_pointer(pid, interp_head_ptr)
    if interp_addr == 0:
        raise RuntimeError("No interpreter found in the

    # Step 2: Read the threads_main pointer from the int
    threads_main_ptr = (
        interp_addr + debug_offsets.interpreter_state.th
    )
    thread_state_addr = read_pointer(pid, threads_main_p
    if thread_state_addr == 0:
        raise RuntimeError("Main thread state is not ava

    return thread_state_addr
```

The following example demonstrates how to locate a thread by its native thread ID:

```
def find_thread_by_id(
    pid: int,
    interp_addr: int,
    debug_offsets: DebugOffsets,
    target_tid: int,
) -> int:
    # Start at threads_head and walk the linked list
    thread_ptr = read_pointer(
        pid,
        interp_addr + debug_offsets.interpreter_state.th
    )

    while thread_ptr:
        native_tid_ptr = (
            thread_ptr + debug_offsets.thread_state.nati
        )
        native_tid = read_int(pid, native_tid_ptr)
        if native_tid == target_tid:
            return thread_ptr
        thread_ptr = read_pointer(
            pid,
            thread_ptr + debug_offsets.thread_state.next
        )

    raise RuntimeError("Thread with the given ID was not
```

Once a valid thread state has been located, the debugger can proceed with modifying its control fields and scheduling execution, as described in the next section.

Writing control information

Once a valid `PyThreadState` structure has been identified, the debugger may modify control fields within it to schedule the execution of a specified Python script. These control fields are checked periodically by the interpreter, and when set correctly, they trigger the execution of remote code at a safe point in the evaluation loop.

Each `PyThreadState` contains a `_PyRemoteDebuggerSupport` structure used for communication between the debugger and the interpreter. The locations of its fields are defined by the `_Py_DebugOffsets` structure and include the following:

- `debugger_script_path`: A fixed-size buffer that holds the full path to a Python source file (`.py`). This file must be accessible and readable by the target process when execution is triggered.
- `debugger_pending_call`: An integer flag. Setting this to `1` tells the interpreter that a script is ready to be executed.
- `eval_breaker`: A field checked by the interpreter during execution. Setting bit 5 (`_PY_EVAL_PLEASE_STOP_BIT`, value `1U << 5`) in this field causes the interpreter to pause and check for debugger activity.

To complete the injection, the debugger must perform the following steps:

1. Write the full script path into the `debugger_script_path` buffer.
2. Set `debugger_pending_call` to `1`.
3. Read the current value of `eval_breaker`, set bit 5 (`_PY_EVAL_PLEASE_STOP_BIT`), and write the updated value back. This signals the interpreter to check for debugger activity.

The following is an example implementation:

```
def inject_script(
    pid: int,
    thread_state_addr: int,
    debug_offsets: DebugOffsets,
    script_path: str
) -> None:
    # Compute the base offset of _PyRemoteDebuggerSuppor
    support_base = (
        thread_state_addr +
        debug_offsets.debugger_support.remote_debugger_s
    )

    # Step 1: Write the script path into debugger_script
    script_path_ptr = (
        support_base +
        debug_offsets.debugger_support.debugger_script_p
    )
    write_string(pid, script_path_ptr, script_path)

    # Step 2: Set debugger_pending_call to 1
    pending_ptr = (
        support_base +
        debug_offsets.debugger_support.debugger_pending_
    )
    write_int(pid, pending_ptr, 1)
    # Step 3: Set _PY_EVAL_PLEASE_STOP_BIT (bit 5, value
    # eval_breaker
    eval_breaker_ptr = (
        thread_state_addr +
        debug_offsets.debugger_support.eval_breaker
    )
    breaker = read_int(pid, eval_breaker_ptr)
    breaker |= (1 << 5)
    write_int(pid, eval_breaker_ptr, breaker)
```

Once these fields are set, the debugger may resume the process (if it was suspended). The interpreter will process the request at the next safe evaluation point, load the script from disk, and execute it.

It is the responsibility of the debugger to ensure that the script file remains present and accessible to the target process during execution.

Nota

Script execution is asynchronous. The script file cannot be deleted immediately after injection. The debugger should wait until the injected script has produced an observable effect before removing the file. This effect depends on what the script is designed to do. For example, a debugger might wait until the remote process connects back to a socket before removing the script. Once such an effect is observed, it is safe to assume the file is no longer needed.

Summary

To inject and execute a Python script in a remote process:

1. Locate the `PyRuntime` structure in the target process's memory.
2. Read and validate the `_Py_DebugOffsets` structure at the

beginning of `PyRuntime`.

3. Use the offsets to locate a valid `PyThreadState`.
4. Write the path to a Python script into `debugger_script_path`.
5. Set the `debugger_pending_call` flag to `1`.
6. Set `_PY_EVAL_PLEASE_STOP_BIT` in the `eval_breaker` field.
7. Resume the process (if suspended). The script will execute at the next safe evaluation point.

Security and threat model

The remote debugging protocol relies on the same operating system primitives used by native debuggers such as GDB and LLDB. Attaching to a process requires the **same privileges** that those debuggers require, for example `ptrace` / Yama LSM on Linux, `task_for_pid` on macOS, and `SeDebugPrivilege` on Windows. Python does not introduce any new privilege escalation path; if an attacker already possesses the permissions needed to attach to a process, they could equally use GDB to read memory or inject code.

The following principles define what is, and is not, considered a security vulnerability in this feature:

Attaching requires OS-level privileges

> On every supported platform the operating system gates cross-process memory access behind privilege checks (`CAP_SYS_PTRACE`, root, or administrator rights). A report that demonstrates an issue only after these privileges have already been obtained is **not** a vulnerability in CPython, since the OS security boundary was already crossed.

Crashes or memory errors when reading a compromised process are not vulnerabilities

> A tool that reads internal interpreter state from a target process must trust that memory to be well-formed. If the

target process has been corrupted or is controlled by an attacker, the debugger or profiler may crash, produce garbage output, or behave unpredictably. This is the same risk accepted by every `ptrace`-based debugger. Bugs in this category (buffer overflows, segmentation faults, or undefined behaviour triggered by reading corrupted state) are **not** treated as security issues, though fixes that improve robustness are welcome.

Vulnerabilities in the target process are not in scope

If the Python process being debugged has already been compromised, the attacker already controls execution in that process. Demonstrating further impact from that starting point does not constitute a vulnerability in the remote debugging protocol.

When to use `PYTHON_DISABLE_REMOTE_DEBUG`

The environment variable `PYTHON_DISABLE_REMOTE_DEBUG` (and the equivalent `-X disable_remote_debug` flag) allows operators to disable the in-process side of the protocol as a **defence-in-depth** measure. This may be useful in hardened or sandboxed deployment environments where no debugging or profiling of the process is expected and reducing attack surface is a priority, even though the OS-level privilege checks already prevent unprivileged access.

Setting this variable does **not** affect other OS-level debugging interfaces (`ptrace`, `/proc`, `task_for_pid`, etc.), which remain available according to their own permission models.

Preguntas más frecuentes de Python

- Preguntas frecuentes generales sobre Python
- Preguntas frecuentes de programación
- Preguntas frecuentes sobre diseño e historia
- Preguntas frecuentes sobre bibliotecas y extensiones
- Extendiendo/Embebiendo FAQ
- Preguntas frecuentes sobre Python en Windows
- Preguntas frecuentes sobre la Interfaz Gráfica de Usuario (*GUI*)
- «¿Por qué está Python instalado en mi ordenador?» FAQ

Preguntas frecuentes generales sobre Python

Contenido

- Preguntas frecuentes generales sobre Python
    - Información general
        - ¿Qué es Python?
        - ¿Que es la *Python Software Foundation*?
        - ¿Hay restricciones de *copyright* sobre el uso de Python?
        - ¿Por qué se creó Python en primer lugar?
        - ¿Para qué es bueno Python?
        - ¿Cómo funciona el esquema numérico de versiones de Python?
        - ¿Cómo obtengo una copia del código fuente de Python?
        - ¿Cómo consigo documentación sobre Python?
        - Nunca he programado antes. ¿Hay un tutorial de Python?
        - ¿Hay un *newsgroup* o una lista de correo dedicada a Python?
        - ¿Cómo obtengo una versión de prueba *beta* de Python?
        - ¿Cómo envío un reporte de *bug* y parches para Python?
        - ¿Hay algún artículo publicado sobre Python que pueda referir?
        - ¿Hay libros sobre Python?
        - ¿En qué parte del mundo está ubicado www.python.org?
        - ¿Por qué se llama Python?
        - ¿Debe gustarme «Monty Python's Flying Circus»?
    - Python en el mundo real
        - ¿Cuán estable es Python?

- ¿Cuánta gente usa Python?
- ¿Hay proyectos significativos hechos con Python?
- ¿Qué nuevos desarrollos se esperan para Python en el futuro?
- ¿Es razonable proponer cambios incompatibles a Python?
- ¿Python es un buen lenguaje para principiantes?

Información general

¿Qué es Python?

Python es un lenguaje interpretado, interactivo y orientado a objetos. Incorpora módulos, excepciones, tipado dinámico, tipos de datos de muy alto nivel y clases. Python combina un poder destacado con una sintaxis muy clara. Tiene interfaces a muchas llamadas de sistema y bibliotecas, así como a varios sistemas de ventana, y es extensible en C o C++. También es usable como un lenguaje de extensión para aplicaciones que necesitan una interfaz programable. Por último, Python es portable: corre en muchas variantes de Unix, en Mac y en Windows 2000 y posteriores.

Para saber más, comienza con El tutorial de Python. La Beginner's Guide to Python [https://wiki.python.org/moin/BeginnersGuide] vincula a otros recursos y tutoriales introductorios para aprender Python.

¿Que es la *Python Software Foundation*?

La *Python Software Foundation* es una organización independiente sin fines de lucro que posee los derechos sobre Python desde la versión 2.1 en adelante. La misión de la PSF es hacer avanzar la tecnología *open source* relacionada al lenguaje de programación Python y publicitar su uso. El sitio web de la PSF es https://www.python.org/psf/.

Las donaciones a la PSF están exentas de impuestos en Estados Unidos. Si usas Python y lo encuentras útil, por favor contribuye a través de la página de donaciones de la PSF [https://www.python.org/psf/donations/].

¿Hay restricciones de *copyright* sobre el uso de Python?

Puedes hacer cualquier cosa que quieras con el código fuente mientras mantengas y muestres los mensajes de derechos de autor en cualquier documentación sobre Python que produzcas. Si respetas las reglas de derechos de autor, está permitido usar Python para fines comerciales, vender copias de Python en forma de código fuente o binarios (modificados o no), o vender productos que incorporen Python de alguna manera. De cualquier manera nos gustaría saber de todos los usos comerciales de Python, por supuesto.

See the license page [https://docs.python.org/3/license.html] to find further explanations and the full text of the PSF License.

El logo de Python tiene derechos comerciales (*trademarked*) y en ciertos casos se requiere un permiso de uso. Consulta la Trademark Usage Policy [https://www.python.org/psf/trademarks/] para más información.

¿Por qué se creó Python en primer lugar?

Aquí hay un *muy* breve resumen sobre qué fue lo que comenzó todo, escrita por Guido van Rossum:

> Tenía vasta experiencia implementando un lenguaje interpretado en el grupo ABC en CWI y trabajando con este grupo había aprendido mucho sobre diseño de lenguajes. Este es el origen de muchas características de Python, incluyendo el uso de sangría para el agrupamiento de sentencias y la inclusión de tipos de datos de muy alto nivel (aunque los detalles son todos diferentes en Python).
>
> Tenía algunos resquemores sobre el lenguaje ABC pero también me gustaban muchas de sus características. Era imposible extenderlo (al lenguaje o sus implementaciones) para remediar

mis quejas – de hecho, la ausencia de extensibilidad fue uno de los mayores problemas. Contaba con alguna experiencia usando Modula-2+ y conversé con los diseñadores de Modula-3 y leí su reporte. Modula-3 es el origen de la sintaxis y semántica que usé para las excepciones y otras características de Python.

Estaba trabajando en Grupo del sistema operativo distribuido Amoeba en CWI. Necesitábamos una mejor manera de hacer administración de sistemas que escribir programas en C o *scripts* de *Bourne shell*, ya que Amoeba tenía sus propia interfaz de llamadas a sistema que no era fácilmente accesible desde *Bourne shell*. Mi experiencia con el manejo de errores de Amoeba me hizo muy consciente de la importancia de las excepciones como una característica de los lenguaje de programación.

Se me ocurrió que un lenguaje de scripts con una sintaxis como ABC pero con acceso a las llamadas al sistema Amoeba satisfaría la necesidad. Me di cuenta de que sería una tontería escribir un lenguaje específico para Amoeba, así que decidí que necesitaba un lenguaje que fuera generalmente extensible.

Durante las vacaciones de Navidad de 1989 tenía mucho tiempo libre, así que decidí hacer un intento. Durante el año siguiente, mientras seguía trabajando principalmente en él durante mi propio tiempo, Python se utilizó en el proyecto Amoeba con un éxito creciente, y los comentarios de mis colegas me hicieron agregar muchas mejoras iniciales.

En febrero de 1991, justo después de un año de desarrollo, decidí publicarlo en USENET. El resto está en el archivo `Misc/HISTORY`.

¿Para qué es bueno Python?

Python es un lenguaje de programación de propósito general de alto nivel que se puede aplicar a muchas clases diferentes de problemas.

El lenguaje viene con una gran biblioteca estándar que cubre áreas como procesamiento de cadenas de caracteres (expresiones regulares, Unicode, cálculo de diferencias entre archivos), protocolos de Internet (HTTP, FTP, SMTP, XML-RPC, POP, IMAP), ingeniería de software. (pruebas unitarias, registro, análisis de rendimiento (*profiling*), análisis de código Python) e interfaces del sistema operativo (llamadas al sistema, sistemas de archivos, sockets TCP / IP). Mira la tabla de contenido de La biblioteca estándar de Python para tener una idea de lo que está disponible. También hay disponible una amplia variedad de extensiones de terceros. Consulta the Python Package Index [https://pypi.org] para encontrar paquetes de interés.

¿Cómo funciona el esquema numérico de versiones de Python?

Las versions de Python son numeradas «A.B.C» o «A.B»:

- *A* es el número de version principal – sólo es incrementada en cambios realmente importantes en el lenguaje.
- *B* es el número de versión menor – se incrementa para cambios menos trascendentales.
- *C* es el número de versión micro – se incrementa para cada versión de corrección de errores.

Not all releases are bugfix releases. In the run-up to a new feature release, a series of development releases are made, denoted as alpha, beta, or release candidate. Alphas are early releases in which interfaces aren't yet finalized; it's not unexpected to see an interface change between two alpha releases. Betas are more stable, preserving existing interfaces but possibly adding new modules, and release candidates are frozen, making no changes except as needed to fix critical bugs.

Las versiones *alpha*, *beta* o *release candidate* tienen un sufijo adicional:

- El sufijo para una versión alfa es «aN» para un pequeño número *N*.
- El sufijo para una versión beta es «bN» para un pequeño número *N*.
- El sufijo para una versión *release candidate * es «rcN» para un pequeño número *N*.

En otras palabras, todas las versiones con la etiqueta *2.0aN* preceden a las versiones con la etiqueta *2.0bN*, que preceden a las versiones con la etiqueta *2.0rcN*, y *esas* preceden a la 2.0.

También puedes encontrar números de versión con un sufijo «+», por ejemplo «2.2+». Estas son versiones sin lanzar, construidas directamente desde el repositorio de desarrollo de CPython. En la práctica, luego de que un lanzamiento menor se realiza, la versión es incrementada a la siguiente versión menor, que se vuelve «a0», por ejemplo «2.4a0».

See the Developer's Guide [https://devguide.python.org/developer-workflow/development-cycle/] for more information about the development cycle, and **PEP 387** [https://peps.python.org/pep-0387/] to learn more about Python's backward compatibility policy. See also the documentation for `sys.version`, `sys.hexversion`, and `sys.version_info`.

¿Cómo obtengo una copia del código fuente de Python?

El código fuente de la versión más reciente de Python está siempre disponible desde python.org, en https://www.python.org/downloads/. El código fuente en desarrollo más reciente se puede obtener en https://github.com/python/cpython/.

La distribución de fuentes es un archivo tar comprimido con gzip que contiene el código C completo, documentación en formato Sphinx, los módulos de la biblioteca de Python, programas de ejemplo y varias piezas útiles de software libremente distribuibles. El código fuente compilará y se ejecutará sin problemas en la mayoría de las plataformas Unix.

Consulta Getting Started section of the Python Developer's Guide [https://devguide.python.org/setup/] para más información sobre cómo obtener el código fuente y compilarlo.

¿Cómo consigo documentación sobre Python?

The standard documentation for the current stable version of Python is available at https://docs.python.org/3/. EPUB, plain text, and downloadable HTML versions are also available at https://docs.python.org/3/download.html.

La documentación está escrita en reStructuredText y procesada con la herramienta de documentación Sphinx [https://www.sphinx-doc.org/]. Las fuentes reStructuredText de la documentación son parte de la distribución fuente de Python.

Nunca he programado antes. ¿Hay un tutorial de Python?

Hay numerosos tutoriales y libros disponibles. La documentación estándar incluye El tutorial de Python.

Consulta the Beginner's Guide [https://wiki.python.org/moin/BeginnersGuide] para encontrar información para principiantes en Python, incluyendo una lista de tutoriales.

¿Hay un *newsgroup* o una lista de correo dedicada a Python?

Hay un grupo de noticias, *comp.lang.python*, y una lista de correo, python-list [https://mail.python.org/mailman/listinfo/python-list]. Tanto el grupo de noticias como la lista de correo están interconectadas entre sí – si puedes leer las noticias no es necesario que te suscribas a la lista de correo. *comp.lang.python* tiene mucho tráfico, recibiendo cientos de publicaciones cada día. y los lectores de Usenet suelen ser más capaces de hacer frente a este volumen.

Announcements of new software releases and events can be found in comp.lang.python.announce, a low-traffic moderated list that receives about five postings per day. It's available as the python-announce mailing list [https://mail.python.org/mailman3/lists/python-announce-list.python.org/].

Más información sobre listas de correo o grupos de noticias puede hallarse en https://www.python.org/community/lists/.

¿Cómo obtengo una versión de prueba *beta* de Python?

Las versiones alpha y beta están disponibles desde https://www.python.org/downloads/. Todos los lanzamientos son anunciados en el grupo de noticias comp.lang.python y comp.lang.python.announce, así como también en la página principal de Python en https://www.python.org/; un *feed* RSS está disponible.

También puedes acceder a la versión en desarrollo de Python desde Git. Mira The Python Developer's Guide [https://devguide.python.org/] para los detalles.

¿Cómo envío un reporte de *bug* y parches para Python?

Para reportar un *bug* o enviar un parche, usa el rastreador de problemas en https://github.com/python/cpython/issues.

Para más información sobre cómo se desarrolla Python, consulta the Python Developer's Guide [https://devguide.python.org/].

¿Hay algún artículo publicado sobre Python que pueda referir?

Lo más probable es que lo mejor sea citar a tu libro preferido sobre Python.

The very first article [https://ir.cwi.nl/pub/18204] about Python was written in 1991 and is now quite outdated.

> Guido van Rossum y Jelke de Boer, «*Interactively Testing Remote Servers Using the Python Programming Language», CWI Quarterly, Volume 4, Issue 4 (Diembre de 1991), Amsterdam, pp 283–303*.

¿Hay libros sobre Python?

Sí, hay muchos, y hay más siendo publicados. Mira la wiki de python.org en https://wiki.python.org/moin/PythonBooks para ver una lista.

También puedes buscar «Python» en las librerías en línea y excluir las que refieran a los Monty Python; o quizás buscar «Python» y «lenguaje».

¿En qué parte del mundo está ubicado www.python.org?

La infraestructura del proyecto Python está ubicada alrededor de todo el mundo y es gestionada por el *Python Infraestructure Team*. Detalles aquí [https://infra.psf.io].

¿Por qué se llama Python?

Cuando comenzó a implementar Python, Guido van Rossum también estaba leyendo los guiones publicados de «Monty Python's Flying Circus» [https://es.wikipedia.org/wiki/Monty_Python], una serie de comedia producida por la BBC de los 70”. Van Rossum pensó que necesitaba un nombre que fuera corto, único y ligeramente misterioso, entonces decidió llamar al lenguaje Python.

¿Debe gustarme «Monty Python's Flying Circus»?

No, pero ayuda. :)

Python en el mundo real

¿Cuán estable es Python?

Very stable. New, stable releases have been coming out roughly every 6 to 18 months since 1991, and this seems likely to continue. As of version 3.9, Python will have a new feature release every 12 months (**PEP 602** [https://peps.python.org/pep-0602/]).

The developers issue bugfix releases of older versions, so the stability of existing releases gradually improves. Bugfix releases, indicated by a third component of the version number (e.g. 3.5.3, 3.6.2), are managed for stability; only fixes for known problems are included in a bugfix release, and it's guaranteed that interfaces will remain the same throughout a series of bugfix releases.

The latest stable releases can always be found on the Python download page [https://www.python.org/downloads/]. Python 3.x is the recommended version and supported by most widely used libraries. Python 2.x **is not maintained anymore** [https://peps.python.org/pep-0373/].

¿Cuánta gente usa Python?

Probablemente hay decenas de miles de usuarios y usuarias, aunque es difícil obtener una cuenta exacta.

Python está disponible gratuitamente para ser descargado por lo que no existen cifras de ventas, a su vez se incluye en muchos sitios diferentes y está empaquetado en muchas distribuciones de Linux, por lo que las estadísticas de descarga tampoco cuentan toda la historia.

El grupo de noticias comp.lang.python es muy activo, pero no todos los usuarios de Python publican allí o incluso lo leen.

¿Hay proyectos significativos hechos con Python?

Mira https://www.python.org/about/success para una lista de proyecto que usan Python. Consultar las actas de conferencias de Python pasadas [https://www.python.org/community/workshops/] revelará contribuciones de diferentes empresas y organizaciones.

Proyectos en Python de alto perfil incluyen el gestor de listas de correo Mailman [https://www.list.org] y el servidor de aplicaciones

Zope [https://www.zope.org]. Muchas distribuciones de Linux, más notoriamente Red Hat [https://www.redhat.com], han escrito partes de sus instaladores y software de administración en Python. Entre las empresas que usan Python internamente se encuentran Google, Yahoo y Lucasfilm Ltd.

¿Qué nuevos desarrollos se esperan para Python en el futuro?

Consulta https://peps.python.org/ para conocer las propuestas de mejora de Python (PEP). Los PEP son documentos de diseño que describen una nueva característica sugerida para Python, que proporciona una especificación técnica concisa y una justificación. Busca un PEP titulado «Python X.Y Release Schedule», donde X.Y es una versión que aún no se ha lanzado públicamente.

New development is discussed on the python-dev mailing list [https://mail.python.org/mailman3/lists/python-dev.python.org/].

¿Es razonable proponer cambios incompatibles a Python?

En general no. Ya existen millones de líneas de código Python alrededor del mundo, por lo que cualquier cambio en el lenguaje que invalide más que una fracción muy pequeña de los programas existentes tiene que ser mal visto. Incluso si puedes proporcionar un programa de conversión, todavía existe el problema de actualizar toda la documentación; se han escrito muchos libros sobre Python y no queremos invalidarlos a todos de un plumazo.

Si una funcionalidad se debe cambiar, es necesario proporcionar una ruta de actualización gradual. **PEP 5** [https://peps.python.org/pep-0005/] describe el procedimiento seguido para introducir cambios incompatibles con versiones anteriores para minimizar disrupciones a los usuarios y usuarias.

¿Python es un buen lenguaje para principiantes?

Sí.

Todavía es común hacer comenzar a estudiantes con lenguajes procedimentales de tipado estático como Pascal, C o un subconjunto de C++ o Java. Los y las estudiantes pueden verse favorecidos si aprenden Python como primer lenguaje. Python tiene una sintaxis simple y consistente y una gran biblioteca estándar. Y, más importante, usar Python en cursos introductorios de programación permite a los estudiantes concentrarse en lo importante de las habilidades de programación como la descomposición de problemas y el diseño de tipos de datos. Con Python los estudiantes pueden ser rápidamente introducidos a conceptos como bucles y procedimientos. Incluso puede trabajar con objetos definidos por el usuario en su primer curso.

Para estudiantes que nunca han programado antes, usar un lenguaje de tipado estático parece antinatural. Presenta complejidades adicionales que deben ser dominadas y ralentizan el ritmo del curso. Quienes están aprendiendo intentan pensar como la computadora, descomponer problemas, diseñar interfaces consistentes y encapsular datos. Si bien aprender a usar un lenguaje de tipado estático es importante en el largo plazo, no es necesariamente el mejor tema a tratar en un primer curso de programación.

Muchos otros aspectos de Python lo vuelven un buen primer lenguaje. Como Java, Python tiene una biblioteca estándar, de manera que los y las estudiantes pueden recibir, de manera temprana, consignas para realizar proyectos de programación que *hagan* algo. Estas consignas no están restringidas a las típicas calculadoras de cuatro operaciones o programas de balances contables. Al usar la biblioteca estándar, pueden ganar la satisfacción de trabajar en aplicaciones realistas mientras aprenden los fundamentos de la programación. Usar la biblioteca estándar también enseña a reusar código. Módulos de terceros, como PyGame, también ayudan a extender los alcances de los y las estudiantes.

El intérprete interactivo de Python les permite probar funcionalidades del lenguaje mientras programan. Pueden tener una ventana con el intérprete corriendo mientras escriben el código de su programa en otra. Si no recuerdan los métodos para una lista, pueden hacer algo así:

```
>>> L = []
>>> dir(L)
['__add__', '__class__', '__contains__', '__delattr__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__', '__hash__',
'__imul__', '__init__', '__iter__', '__le__', '__len__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_
'__repr__', '__reversed__', '__rmul__', '__setattr__', '
'__sizeof__', '__str__', '__subclasshook__', 'append', '
'copy', 'count', 'extend', 'index', 'insert', 'pop', 're
'reverse', 'sort']
>>> [d for d in dir(L) if '__' not in d]
['append', 'clear', 'copy', 'count', 'extend', 'index',

>>> help(L.append)
Help on built-in function append:

append(...)
    L.append(object) -> None -- append object to end

>>> L.append(1)
>>> L
[1]
```

Con el intérprete, la documentación nunca está lejos de los o las estudiantes mientras están programando.

También hay buenos IDE para Python. IDLE es un IDE multiplataforma para Python que está escrito en Python usando Tkinter. Los usuarios de Emacs estarán felices de saber que hay un muy buen modo de Python para Emacs. Todos estos entornos de programación proporcionan resaltado de sintaxis, indentación automática y acceso al intérprete interactivo durante la codificación. Consulta la wiki de Python [https://wiki.python.org/moin/PythonEditors] para obtener una lista completa de los entornos de edición de Python.

Si quieres discutir el uso de Python en la educación, quizás te interese unirte a la la lista de correo edu-sig [https://www.python.org/community/sigs/current/edu-sig].

Preguntas frecuentes de programación

Contenido

- Preguntas frecuentes de programación
  - General questions
    - Is there a source code-level debugger with breakpoints and single-stepping?
    - ¿Existe alguna herramienta que ayude a encontrar errores o realizar análisis estático?
    - ¿Cómo puedo crear un binario independiente a partir de un programa Python?
    - ¿Existen estándares de código o una guía de estilo para programas Python?
  - Core language
    - ¿Por qué obtengo un *UnboundLocalError* cuando la variable tiene un valor?
    - ¿Cuáles son las reglas para las variables locales y globales en Python?
    - ¿Por qué las funciones lambda definidas en un bucle con diferentes valores devuelven todas el mismo resultado?
    - ¿Cómo puedo compartir variables globales entre módulos?
    - ¿Cuáles son las «buenas prácticas» para usar import en un módulo?
    - ¿Por qué los valores por defecto se comparten entre objetos?
    - ¿Cómo puedo pasar parámetros por palabra clave u opcionales de una función a otra?
    - ¿Cuál es la diferencia entre argumentos y parámetros?
    - ¿Por qué cambiando la lista "y" cambia, también, la lista "x"?

    - ¿Cómo puedo escribir una función sin parámetros (invocación mediante referencia)?
    - ¿Cómo se puede hacer una función de orden superior en Python?
    - ¿Cómo copio un objeto en Python?
    - ¿Cómo puedo encontrar los métodos o atributos de un objeto?
    - ¿Cómo puede mi código descubrir el nombre de un objeto?
    - ¿Qué ocurre con la precedencia del operador coma?
    - ¿Existe un equivalente al operador ternario de C «?:»?
    - ¿Es posible escribir expresiones en una línea de forma ofuscada en Python?
    - ¿Qué hace la barra (/) en medio de la lista de parámetros de una función?
- Números y cadenas
    - ¿Cómo puedo especificar enteros hexadecimales y octales?
    - ¿Por qué -22 // 10 devuelve -3?
    - ¿Cómo puedo obtener un atributo int literal en lugar de SyntaxError?
    - ¿Cómo convierto una cadena a un número?
    - ¿Cómo puedo convertir un número a una cadena?
    - ¿Cómo puedo modificar una cadena in situ?
    - ¿Cómo puedo usar cadenas para invocar funciones/métodos?
    - Is there an equivalent to Perl's **`chomp()`** for removing trailing newlines from strings?
    - Is there a **`scanf()`** or **`sscanf()`** equivalent?
    - What does **`UnicodeDecodeError`** or **`UnicodeEncodeError`** error mean?
    - Can I end a raw string with an odd number of backslashes?
- Rendimiento
    - Mi programa es muy lento. ¿Cómo puedo acelerarlo?
    - ¿Cuál es la forma más eficiente de concatenar muchas cadenas conjuntamente?

- Sequences (tuples/lists)
  - ¿Cómo convertir entre tuplas y listas?
  - ¿Qué es un índice negativo?
  - ¿Cómo puedo iterar sobre una secuencia en orden inverso?
  - ¿Cómo eliminar duplicados de una lista?
  - How do you remove multiple items from a list?
  - ¿Cómo se puede hacer un array en Python?
  - ¿Cómo puedo crear una lista multidimensional?
  - How do I apply a method or function to a sequence of objects?
  - ¿Por qué hacer lo siguiente, `a_tuple[i] += ['item']`, lanza una excepción cuando la suma funciona?
  - Quiero hacer una ordenación compleja: ¿Puedes hacer una transformada Schwartziana (Schwartzian Transform) en Python?
  - ¿Cómo puedo ordenar una lista a partir de valores de otra lista?
- Objetos
  - ¿Qué es una clase?
  - ¿Qué es un método?
  - ¿Qué es self?
  - ¿Cómo puedo comprobar si un objeto es una instancia de una clase dada o de una subclase de la misma?
  - ¿Qué es la delegación?
  - ¿Cómo invoco a un método definido en una clase base desde una clase derivada que la extiende?
  - ¿Cómo puedo organizar mi código para hacer que sea más sencillo modificar la clase base?
  - ¿Cómo puedo crear datos estáticos de clase y métodos estáticos de clase?
  - ¿Como puedo sobrecargar constructores (o métodos) en Python?
  - Intento usar __spam y obtengo un error sobre _SomeClassName__spam.
  - Mi clase define __del__ pero no se le invoca cuando borro el objeto.
  - ¿Cómo puedo obtener una lista de todas las

    instancias de una clase dada?
    - ¿Por qué el resultado de `id()` no parece ser único?
    - ¿Cuándo puedo fiarme de pruebas de identidad con el operador *is*?
    - ¿Cómo puede una subclase controlar qué datos se almacenan en una instancia inmutable?
    - ¿Cómo cacheo llamadas de método?
- Módulos
    - ¿Cómo creo un fichero .pyc?
    - ¿Cómo puedo encontrar el nombre del módulo en uso?
    - ¿Cómo podría tener módulos que se importan mutuamente entre ellos?
    - _import_("x.y.z") devuelve <module "x">; ¿cómo puedo obtener z?
    - Cuando edito un módulo importado y lo reimporto los cambios no tienen efecto. ¿Por qué sucede esto?

General questions

Is there a source code-level debugger with breakpoints and single-stepping?

Sí.

Debajo se describen algunos depuradores para Python y la función integrada `breakpoint()` te permite ejecutar alguno de ellos.

El módulo pdb es en depurador en modo consola simple pero conveniente para Python. Es parte de la biblioteca estándar de Python y está `documentado en el manual de referencia de la biblioteca`. Puedes escribir tu propio depurador usando el código de pdb como ejemplo.

The IDLE interactive development environment, which is part of the standard Python distribution (normally available as `idlelib`), includes a graphical debugger.

PythonWin es un IDE Python que incluye un depurador con GUI basado en pdb. El depurador PythonWin colorea los puntos de interrupción y dispone de características geniales como la depuración de programas no modificados mediante PythonWin. PythonWin está disponible como parte del proyecto Las extensiones de Python para Windows [https://sourceforge.net/projects/pywin32/] y como parte de la distribución ActivePython [https://www.activestate.com/activepython].

Eric [https://eric-ide.python-projects.org/] is an IDE built on PyQt and the Scintilla editing component.

trepan3k [https://github.com/rocky/python3-trepan/] es un depurador similar a *gdb*.

Visual Studio Code [https://code.visualstudio.com/] es un IDE con herramientas de depuración que se integra con software de control de versiones.

Existen varios IDEs comerciales para Python que incluyen depuradores gráficos. Entre ellos tenemos:

- IDE Wing [https://wingware.com/]
- PyCharm [https://www.jetbrains.com/pycharm/]

¿Existe alguna herramienta que ayude a encontrar errores o realizar análisis estático?

Sí.

Ruff [https://docs.astral.sh/ruff/], Pylint [https://pylint.readthedocs.io/] and Pyflakes [https://github.com/PyCQA/pyflakes] do basic checking that will help you catch bugs sooner.

Static type checkers such as mypy [https://mypy-lang.org/], ty [https://docs.astral.sh/ty/], Pyrefly [https://pyrefly.org/], and pytype [https://github.com/google/pytype] can check type hints in Python source code.

¿Cómo puedo crear un binario independiente a partir de un programa Python?

No necesitas tener la habilidad de compilar Python a código C si lo único que necesitas es un programa independiente que los usuarios puedan descargar y ejecutar sin necesidad de instalar primero una distribución Python. Existe una serie de herramientas que determinan el conjunto de módulos que necesita un programa y une estos módulos conjuntamente con un binario Python para generar un único ejecutable.

One is to use the freeze tool, which is included in the Python source tree as Tools/freeze [https://github.com/python/cpython/tree/3.14/Tools/freeze]. It converts Python byte code to C arrays; with a C compiler you can embed all your modules into a new program, which is then linked with the standard Python modules.

Funciona escaneando su fuente de forma recursiva en busca de declaraciones de importación (en ambas formas) y buscando los módulos en la ruta estándar de Python, así como en el directorio de la fuente (para los módulos incorporados). Luego convierte el *bytecode* de los módulos escritos en Python en código C (inicializadores de arrays que pueden ser convertidos en objetos de código usando el módulo marshal) y crea un archivo de configuración a medida que sólo contiene aquellos módulos incorporados que se usan realmente en el programa. A continuación, compila el código C generado y lo enlaza con el resto del intérprete de Python para formar un binario autónomo que actúa exactamente igual que su script.

Los siguientes paquetes pueden ayudar con la creación de ejecutables de consola y GUI:

- Nuitka [https://nuitka.net/] (Multiplataforma)
- PyInstaller [https://pyinstaller.org/] (Cross-platform)
- PyOxidizer [https://pyoxidizer.readthedocs.io/en/stable/] (Multiplataforma)
- cx_Freeze [https://marcelotduarte.github.io/cx_Freeze/] (Multiplataforma)
- py2app [https://github.com/ronaldoussoren/py2app] (macOS solamente)
- py2exe [https://www.py2exe.org/] (Windows only)

¿Existen estándares de código o una guía de estilo para programas Python?

Sí. El estilo de código requerido para los módulos de la biblioteca estándar se encuentra documentado como **PEP 8** [https://peps.python.org/pep-0008/].

Core language

¿Por qué obtengo un *UnboundLocalError* cuando la variable tiene un valor?

It can be a surprise to get the `UnboundLocalError` in previously working code when it is modified by adding an assignment statement somewhere in the body of a function.

Este código:

```
>>> x = 10
>>> def bar():
...     print(x)
...
>>> bar()
10
```

funciona, pero este código:

```
>>> x = 10
>>> def foo():
...     print(x)
...     x += 1
```

results in an `UnboundLocalError`:

```
>>> foo()
Traceback (most recent call last):
  ...
UnboundLocalError: cannot access local variable 'x' wher
```

Esto es debido a que cuando realizas una asignación a una variable en un ámbito de aplicación, esa variable se convierte en local y enmascara cualquier variable llamada de forma similar en un ámbito de aplicación exterior. Desde la última declaración en foo asigna un nuevo valor a `x`, el compilador la reconoce como una variable local. Consecuentemente, cuando el `print(x)` más próximo intenta mostrar la variable local no inicializada se muestra un error.

En el ejemplo anterior puedes acceder al ámbito de aplicación exterior a la variable declarándola como global:

```
>>> x = 10
>>> def foobar():
...     global x
...     print(x)
...     x += 1
...
>>> foobar()
10
```

Esta declaración explícita es necesaria de cara a recordarte que (a diferencia de la situación superficialmente análoga con las variables de clase e instancia) estás modificando el valor de la variable en un ámbito de aplicación más externo:

```
>>> print(x)
11
```

Puedes hacer algo similar en un ámbito de aplicación anidado usando la palabra clave **`nonlocal`**:

```
>>> def foo():
...    x = 10
...    def bar():
...        nonlocal x
...        print(x)
...        x += 1
...    bar()
...    print(x)
```

```
...
>>> foo()
10
11
```

¿Cuáles son las reglas para las variables locales y globales en Python?

En Python, las variables que solo se encuentran referenciadas dentro de una función son globales implícitamente. Si a una variable se le asigna un valor en cualquier lugar dentro del cuerpo de una función, se asumirá que es local a no ser que explícitamente se la declare como global.

Aunque, inicialmente, puede parecer sorprendente, un momento de consideración permite explicar esto. Por una parte, requerir **`global`** para variables asignadas proporciona una barrera frente a efectos secundarios indeseados. Por otra parte, si `global` es requerido para todas las referencias globales, deberás usar `global` en todo momento. Deberías declarar como global cualquier referencia a una función integrada o a un componente de un módulo importado. Este embrollo arruinaría la utilidad de la declaración «global» para identificar los efectos secundarios.

¿Por qué las funciones lambda definidas en un bucle con diferentes valores devuelven todas el mismo resultado?

Assume you use a for loop to define a few different lambdas (or even plain functions), for example:

```
>>> squares = []
>>> for x in range(5):
...     squares.append(lambda: x**2)
```

Lo siguiente proporciona una lista que contiene 5 funciones lambda que calculan `x**2`. Esperarías que, cuando se les invoca, retornaran, respectivamente, `0`, `1`, `4`, `9` y `16`. Sin embargo, cuando lo ejecutes verás que todas devuelven `16`:

```
>>> squares[2]()
16
>>> squares[4]()
16
```

This happens because `x` is not local to the lambdas, but is defined in the outer scope, and it is accessed when the lambda is called — not when it is defined. At the end of the loop, the value of `x` is `4`, so all the functions now return `4**2`, that is `16`. You can also verify this by changing the value of `x` and see how the results of the lambdas change:

```
>>> x = 8
>>> squares[2]()
64
```

De cara a evitar esto necesitas guardar los valores en variables locales a las funciones lambda de tal forma que no dependan del valor de la `x` global:

```
>>> squares = []
>>> for x in range(5):
...     squares.append(lambda n=x: n**2)
```

Aquí, `n=x` crea una nueva variable `n` local a la función lambda y ejecutada cuando la función lambda se define de tal forma que tiene el mismo valor que tenía `x` en ese punto en el bucle. Esto significa que el valor de `n` será `0` en la primera función lambda, `1` en la segunda, `2` en la tercera y así sucesivamente. Por tanto, ahora cada lambda retornará el resultado correcto:

```
>>> squares[2]()
4
>>> squares[4]()
16
```

Es de destacar que este comportamiento no es peculiar de las funciones lambda sino que aplica también a las funciones regulares.

¿Cómo puedo compartir variables globales entre

módulos?

La forma canónica de compartir información entre módulos dentro de un mismo programa sería creando un módulo especial (a menudo llamado config o cfg). Simplemente importa el módulo config en todos los módulos de tu aplicación; el módulo estará disponible como un nombre global. Debido a que solo hay una instancia de cada módulo, cualquier cambio hecho en el objeto módulo se reflejará en todos los sitios. Por ejemplo:

config.py:

```
x = 0   # Default value of the 'x' configuration settin
```

mod.py:

```
import config
config.x = 1
```

main.py:

```
import config
import mod
print(config.x)
```

Note that using a module is also the basis for implementing the singleton design pattern, for the same reason.

¿Cuáles son las «buenas prácticas» para usar import en un módulo?

En general, no uses `from modulename import *`. Haciendo eso embarulla el espacio de nombres del importador y hace que sea más difícil para los *linters* el detectar los nombres sin definir.

Importar los módulos en la parte inicial del fichero. Haciéndolo así deja claro los módulos que son necesarios para tu código y evita preguntas sobre si el nombre del módulo se encuentra en el ámbito de la aplicación. Usar una importación por línea hace que sea sencillo añadir y eliminar módulos importados pero usar múltiples importaciones por línea usa menos espacio de pantalla.

Es una buena práctica si importas los módulos en el orden siguiente:

1. standard library modules – such as `sys`, `os`, `argparse`, `re`
2. third-party library modules (anything installed in Python's site-packages directory) – such as dateutil [https://pypi.org/project/dateutil/], requests [https://pypi.org/project/requests/], tzdata [https://pypi.org/project/tzdata/]
3. locally developed modules

Hay veces en que es necesario mover las importaciones a una función o clase para evitar problemas de importaciones circulares. Gordon McMillan dice:

> No hay problema con las importaciones circulares cuando ambos módulos usan la forma de importación «import <module>». Fallará cuando el segundo módulo quiera coger un nombre del primer módulo («from module import name») y la importación se encuentre en el nivel superior. Esto sucede porque los nombres en el primero todavía no se encuentran disponibles debido a que el primer módulo se encuentra ocupado importando al segundo.

En este caso, si el segundo módulo se usa solamente desde una función, la importación se puede mover de forma sencilla dentro de la función. En el momento en que se invoca a la importación el primer módulo habrá terminado de inicializarse y el segundo módulo podrá hacer la importación.

También podría ser necesario mover importaciones fuera del nivel superior del código si alguno de loa módulos son específicos a la plataforma. En ese caso podría, incluso, no ser posible importar todos los módulos en la parte superior del fichero. Para esos casos, la importación correcta de los módulos en el código correspondiente específico de la plataforma es una buena opción.

Solo debes mover importaciones a un ámbito de aplicación local, como dentro de la definición de una función, si es necesario resolver problemas como una importación circular o al intentar reducir el tiempo de inicialización de un módulo. Esta técnica es especialmente útil si muchas de las importaciones no son necesarias dependiendo de cómo se ejecute el programa. También podrías mover importaciones a una función si los módulos solo se usan dentro de esa función. Nótese que la primera carga de un módulo puede ser costosa debido al tiempo necesario para la inicialización del módulo,pero la carga de un módulo múltiples veces está prácticamente libre de coste ya que solo es necesario hacer búsquedas en un diccionario. Incluso si el nombre del módulo ha salido del ámbito de aplicación el módulo se encuentre, probablemente, en **`sys.modules`**.

¿Por qué los valores por defecto se comparten entre objetos?

Este tipo de error golpea a menudo a programadores novatos. Considera esta función:

```
def foo(mydict={}):  # Danger: shared reference to one
    ... compute something ...
    mydict[key] = value
    return mydict
```

La primera vez que llamas a esta función, `mydict` solamente contiene un único elemento. La segunda vez, `mydict` contiene dos elementos debido a que cuando comienza la ejecución de `foo()`, `mydict` comienza conteniendo un elemento de partida.

A menudo se esperaría que una invocación a una función cree nuevos objetos para valores por defecto. Eso no es lo que realmente sucede. Los valores por defecto se crean exactamente una sola vez, cuando se define la función. Se se cambia el objeto, como el diccionario en este ejemplo, posteriores invocaciones a la función estarán referidas al objeto cambiado.

Por definición, los objetos inmutables como números, cadenas, tuplas y `None` están asegurados frente al cambio. Cambios en objetos mutables como diccionarios, listas e instancias de clase pueden llevar a confusión.

Debido a esta característica es una buena práctica de programación el no usar valores mutables como valores por defecto. En su lugar usa `None` como valor por defecto dentro de la función, comprueba si el parámetro es `None` y crea una nueva lista/un nuevo diccionario/cualquier otras cosa que necesites. Por ejemplo, no escribas:

```
def foo(mydict={}):
    ...
```

pero:

```
def foo(mydict=None):
    if mydict is None:
        mydict = {}  # create a new dict for local names
```

Esta característica puede ser útil. Cuando tienes una función que es muy costosa de ejecutar, una técnica común es *cachear* sus parámetros y el valor resultante de cada invocación a la función y retornar el valor *cacheado* si se solicita nuevamente el mismo valor. A esto se le llama «memoizing» y se puede implementar de la siguiente forma:

```
# Callers can only provide two parameters and optionally
def expensive(arg1, arg2, *, _cache={}):
    if (arg1, arg2) in _cache:
        return _cache[(arg1, arg2)]

    # Calculate the value
    result = ... expensive computation ...
    _cache[(arg1, arg2)] = result           # Store resu
    return result
```

Podrías usar una variable global conteniendo un diccionario en lugar de un valor por defecto; es una cuestión de gustos.

¿Cómo puedo pasar parámetros por palabra clave u opcionales de una función a otra?

Recopila los argumentos usando los especificadores `*` y `**` en la lista de parámetros de la función; esto te proporciona los argumentos posicionales como una tupla y los argumentos con palabras clave como un diccionario. Puedes, entonces, pasar estos argumentos cuando invoques a otra función usando `*` y `**`:

```
def f(x, *args, **kwargs):
    ...
    kwargs['width'] = '14.3c'
    ...
    g(x, *args, **kwargs)
```

¿Cuál es la diferencia entre argumentos y parámetros?

Parameters are defined by the names that appear in a function definition, whereas arguments are the values actually passed to a function when calling it. Parameters define what kind of arguments a function can accept. For example, given the function definition:

```
def func(foo, bar=None, **kwargs):
    pass
```

foo, *bar* y *kwargs* son parámetros de `func`. Sin embargo, cuando invocamos a `func`, por ejemplo:

```
func(42, bar=314, extra=somevar)
```

los valores `42`, `314` y `somevar` son argumentos.

¿Por qué cambiando la lista "y" cambia, también, la lista "x"?

Si escribes código como:

```
>>> x = []
>>> y = x
>>> y.append(10)
>>> y
[10]
>>> x
```

```
[10]
```

te estarás preguntando porque añadir un elemento a `y` ha cambiado también a `x`.

Hay dos factores que provocan este resultado:

1. Las variables son simplemente nombres que referencian a objetos. Haciendo `y = x` no crea una copia de la lista – crea una nueva variable `y` que referencia al mismo objeto al que referencia `x` . Esto significa que solo existe un objeto (la lista) y tanto `x` como `y` hacen referencia al mismo.
2. Las listas son mutable, lo que significa que puedes cambiar su contenido.

After the call to **`append()`**, the content of the mutable object has changed from `[]` to `[10]`. Since both the variables refer to the same object, using either name accesses the modified value `[10]`.

Si, por otra parte, asignamos un objeto inmutable a `x`:

```
>>> x = 5  # ints are immutable
>>> y = x
>>> x = x + 1  # 5 can't be mutated, we are creating a r
>>> x
6
>>> y
5
```

podemos ver que `x` e `y` ya no son iguales. Esto es debido a que los enteros son immutable, y cuando hacemos `x = x + 1` no estamos mutando el entero `5` incrementando su valor; en su lugar, estamos creando un nuevo objeto (el entero `6`) y se lo asignamos a `x` (esto es, cambiando el objeto al cual referencia `x`). Después de esta asignación tenemos dos objetos (los enteros `6` y `5`) y dos variables que referencian a ellos (`x` ahora referencia a `6` pero `y` todavía referencia a `5`).

Some operations (for example `y.append(10)` and `y.sort()`) mutate the object, whereas superficially similar operations (for example `y = y + [10]` and **`sorted(y)`**) create a new object. In general in Python (and in all cases in the standard library) a method that mutates an object will return `None` to help avoid getting the two types of operations confused. So if you mistakenly write `y.sort()` thinking it will give you a sorted copy of `y`, you'll instead end up with `None`, which will likely cause your program to generate an easily diagnosed error.

Sin embargo, existe una clase de operaciones en las cuales la misma operación tiene, a veces, distintos comportamientos con diferentes tipos: los operadores de asignación aumentada. Por ejemplo, `+=` muta listas pero no tuplas o enteros (`a_list += [1, 2, 3]` es equivalente a `a_list.extend([1, 2, 3])` y muta `a_list`, mientras que `some_tuple += (1, 2, 3)` y `some_int += 1` crea nuevos objetos).

En otras palabras:

- If we have a mutable object (such as `list`, `dict`, `set`), we can use some specific operations to mutate it and all the variables that refer to it will see the change.
- If we have an immutable object (such as `str`, `int`, `tuple`), all the variables that refer to it will always see the same value, but operations that transform that value into a new value always return a new object.

Si deseas saber si dos variables referencian o no al mismo objeto puedes usar el operador `is` o la función incorporada `id()`.

¿Cómo puedo escribir una función sin parámetros (invocación mediante referencia)?

Remember that arguments are passed by assignment in Python. Since assignment just creates references to objects, there's no alias between an argument name in the caller and callee, and consequently no call-by-reference. You can achieve the desired effect in a number of ways.

1. Mediante el retorno de una tupla de resultados:

   ```
   >>> def func1(a, b):
   ...     a = 'new-value'        # a and b are local
   ```

```
...     b = b + 1                  # assigned to new ob
...     return a, b                # return new values
...
>>> x, y = 'old-value', 99
>>> func1(x, y)
('new-value', 100)
```

Esta es, casi siempre, la solución más clara.

2. Mediante el uso de variables globales. No es thread-safe y no se recomienda.

3. Pasando un objeto mutable (intercambiable en el mismo sitio):

```
>>> def func2(a):
...     a[0] = 'new-value'         # 'a' references a m
...     a[1] = a[1] + 1            # changes a shared o
...
>>> args = ['old-value', 99]
>>> func2(args)
>>> args
['new-value', 100]
```

4. Pasando un diccionario que muta:

```
>>> def func3(args):
...     args['a'] = 'new-value'     # args is a mut
...     args['b'] = args['b'] + 1   # change it in-
...
>>> args = {'a': 'old-value', 'b': 99}
>>> func3(args)
>>> args
{'a': 'new-value', 'b': 100}
```

5. O empaquetar valores en una instancia de clase:

```
>>> class Namespace:
...     def __init__(self, /, **args):
...         for key, value in args.items():
```

```
...                 setattr(self, key, value)
...
>>> def func4(args):
...     args.a = 'new-value'        # args is a mut
...     args.b = args.b + 1         # change object
...
>>> args = Namespace(a='old-value', b=99)
>>> func4(args)
>>> vars(args)
{'a': 'new-value', 'b': 100}
```

Casi nunca existe una buena razón para hacer esto tan complicado.

Tu mejor opción es retornar una tupla que contenga los múltiples resultados.

¿Cómo se puede hacer una función de orden superior en Python?

Tienes dos opciones: puedes usar ámbitos de aplicación anidados o puedes usar objetos invocables. Por ejemplo, supón que querías definir `linear(a,b)` que devuelve una función `f(x)` que calcula el valor `a*x+b`. Usar ámbitos de aplicación anidados:

```
def linear(a, b):
    def result(x):
        return a * x + b
    return result
```

O usar un objeto invocable:

```
class linear:

    def __init__(self, a, b):
        self.a, self.b = a, b

    def __call__(self, x):
        return self.a * x + self.b
```

En ambos casos,

```
taxes = linear(0.3, 2)
```

nos da un objeto invocable donde `taxes(10e6) == 0.3 * 10e6 + 2`.

El enfoque del objeto invocable tiene la desventaja que es un ligeramente más lento y el resultado es un código levemente más largo. Sin embargo, destacar que una colección de invocables pueden compartir su firma vía herencia:

```
class exponential(linear):
    # __init__ inherited
    def __call__(self, x):
        return self.a * (x ** self.b)
```

Los objetos pueden encapsular el estado de varios métodos:

```
class counter:

    value = 0

    def set(self, x):
        self.value = x

    def up(self):
        self.value = self.value + 1

    def down(self):
        self.value = self.value - 1

count = counter()
inc, dec, reset = count.up, count.down, count.set
```

Aquí `inc()`, `dec()` y `reset()` se comportan como funciones las cuales comparten la misma variable de conteo.

¿Cómo copio un objeto en Python?

En general, prueba **`copy.copy()`** o **`copy.deepcopy()`** para el caso general. No todos los objetos se pueden copiar pero la mayoría sí que pueden copiarse.

Algunas objetos se pueden copiar de forma más sencilla. Los diccionarios disponen de un método **`copy()`**:

```
newdict = olddict.copy()
```

Las secuencias se pueden copiar usando un rebanado:

```
new_l = l[:]
```

¿Cómo puedo encontrar los métodos o atributos de un objeto?

For an instance `x` of a user-defined class, **`dir(x)`** returns an alphabetized list of the names containing the instance attributes and methods and attributes defined by its class.

¿Cómo puede mi código descubrir el nombre de un objeto?

Hablando de forma general no podrían puesto que los objetos no disponen, realmente, de un nombre. Esencialmente, las asignaciones relacionan un nombre con su valor; Lo mismo se cumple con las declaraciones `def` y `class` pero, en este caso, el valor es un invocable. Considera el siguiente código:

```
>>> class A:
...     pass
...
>>> B = A
>>> a = B()
>>> b = a
>>> print(b)
<__main__.A object at 0x16D07CC>
>>> print(a)
<__main__.A object at 0x16D07CC>
```

Arguably the class has a name: even though it is bound to two names and invoked through the name `B` the created instance is still reported as an instance of class `A`. However, it is impossible to say whether the instance's name is `a` or `b`, since both names are bound to the same value.

En términos generales, no debería ser necesario que tu código «conozca los nombres» de determinados valores. A menos que estés escribiendo deliberadamente programas introspectivos, esto suele ser una indicación de que un cambio de enfoque podría ser beneficioso.

En comp.lang.python, Fredrik Lundh proporcionó una vez una excelente analogía en respuesta a esta pregunta:

> De la misma forma que obtienes el nombre de ese gato que te has encontrado en tu porche el propio gato (objeto) no te puede indicar su nombre y, realmente, no importa – por tanto, la única forma de encontrar cómo se llama sería preguntando a todos los vecinos (espacios de nombres) si es su gato (objeto)…
>
> …y no te sorprendas si encuentras que se le conoce mediante diferentes nombres o ¡nadie conoce su nombre!

¿Qué ocurre con la precedencia del operador coma?

La coma no es un operador en Python. Considera la sesión:

```
>>> "a" in "b", "a"
(False, 'a')
```

Debido a que la coma no es un operador sino un separador entre expresiones lo anterior se evalúe como se ha introducido:

```
("a" in "b"), "a"
```

no:

```
"a" in ("b", "a")
```

The same is true of the various assignment operators (`=`, `+=`, and so on). They are not truly operators but syntactic delimiters in assignment statements.

¿Existe un equivalente al operador ternario de C «?:»?

Sí, existe. La sintaxis es como sigue:

```
[on_true] if [expression] else [on_false]

x, y = 50, 25
small = x if x < y else y
```

Antes de que esta sintaxis se introdujera en Python 2.5 una expresión común fue el uso de operadores lógicos:

```
[expression] and [on_true] or [on_false]
```

Sin embargo, esa expresión no es segura ya que puede retornar valores erróneos cuando *on_true* tiene un valor booleano falso. Por tanto, siempre es mejor usar la forma `... if ... else ...`.

¿Es posible escribir expresiones en una línea de forma ofuscada en Python?

Yes. Usually this is done by nesting **`lambda`** within **`lambda`**. See the following three examples, slightly adapted from Ulf Bartelt:

```
from functools import reduce

# Primes < 1000
print(list(filter(None,map(lambda y:y*reduce(lambda x,y:
map(lambda x,y=y:y%x,range(2,int(pow(y,0.5)+1))),1),rang

# First 10 Fibonacci numbers
print(list(map(lambda x,f=lambda x,f:(f(x-1,f)+f(x-2,f))
f(x,f), range(10))))
```

```
# Mandelbrot set
print((lambda Ru,Ro,Iu,Io,IM,Sx,Sy:reduce(lambda x,y:x+'
Iu=Iu,Io=Io,Ru=Ru,Ro=Ro,Sy=Sy,L=lambda yc,Iu=Iu,Io=Io,Ru
Sx=Sx,Sy=Sy:reduce(lambda x,y:x+y,map(lambda x,xc=Ru,yc=
i=i,Sx=Sx,F=lambda xc,yc,x,y,k,f=lambda xc,yc,x,y,k,f:(k
>=4.0) or 1+f(xc,yc,x*x-y*y+xc,2.0*x*y+yc,k-1,f):f(xc,yc
64+F(Ru+x*(Ro-Ru)/Sx,yc,0,0,i)),range(Sx))):L(Iu+y*(Io-I
))))(-2.1, 0.7, -1.2, 1.2, 30, 80, 24))
#    \___ ___/  \___ ___/  |   |   |__ lines on screen
#        V          V      |   |______ columns on screen
#        |          |      |__________ maximum of "itera
#        |          |_________________ range on y axis
#        |____________________________ range on x axis
```

¡No probéis esto en casa, personitas!

¿Qué hace la barra (/) en medio de la lista de parámetros de una función?

A slash in the argument list of a function denotes that the parameters prior to it are positional-only. Positional-only parameters are the ones without an externally usable name. Upon calling a function that accepts positional-only parameters, arguments are mapped to parameters based solely on their position. For example, **`divmod()`** is a function that accepts positional-only parameters. Its documentation looks like this:

```
>>> help(divmod)
Help on built-in function divmod in module builtins:

divmod(x, y, /)
    Return the tuple (x//y, x%y).  Invariant: div*y + mo
```

El *slash* al final de la lista de parámetros indica que los tres parámetros son únicamente posicionales. Por tanto, invocar a **`pow()`** con argumentos con palabra clave podría derivar en un error:

```
>>> divmod(x=3, y=4)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: divmod() takes no keyword arguments
```

Números y cadenas

¿Cómo puedo especificar enteros hexadecimales y octales?

Para especificar un dígito octal, prefija el valor octal con un cero y una «o» en minúscula o mayúscula. Por ejemplo, para definir la variable «a» con el valor octal «10» (8 en decimal), escribe:

```
>>> a = 0o10
>>> a
8
```

Un hexadecimal es igual de simple. Simplemente añade un cero y una «x», en minúscula o mayúscula, antes del número hexadecimal . Los dígitos hexadecimales se pueden especificar en minúsculas o mayúsculas. Por ejemplo, en el intérprete de Python:

```
>>> a = 0xa5
>>> a
165
>>> b = 0XB2
>>> b
178
```

¿Por qué -22 // 10 devuelve -3?

Es debido, principalmente al deseo que `i % j` tenga el mismo signo que `j`. Si quieres eso y, además, quieres:

```
i == (i // j) * j + (i % j)
```

entonces la división entera a la baja debe retornar el valor base más bajo. C también requiere que esa identidad se mantenga de tal forma que cuando los compiladores truncan `i // j` necesitan que `i % j` tenga el mismo signo que `i`.

Existen unos pocos casos para `i % j` cuando `j` es negativo. Cuando `j` es positivo, existen muchos casos y, virtualmente, en todos ellos es más útil para `i % j` que sea `>= 0`. Si el reloj dice que ahora son las 10, ¿qué dijo hace 200 horas? `-190 % 12 == 2` es útil; `-190 % 12 == -10` es un error listo para morderte.

¿Cómo puedo obtener un atributo int literal en lugar de SyntaxError?

Trying to lookup an `int` literal attribute in the normal manner gives a **`SyntaxError`** because the period is seen as a decimal point:

```
>>> 1.__class__
  File "<stdin>", line 1
  1.__class__
   ^
SyntaxError: invalid decimal literal
```

La solución es separar el literal del punto con un espacio o un paréntesis.

```
>>> 1 .__class__
<class 'int'>
>>> (1).__class__
<class 'int'>
```

¿Cómo convierto una cadena a un número?

For integers, use the built-in **`int()`** type constructor, for example, `int('144') == 144`. Similarly, **`float()`** converts to a floating-point number, for example, `float('144') == 144.0`.

Por defecto, estas interpretan el número como decimal de tal forma que `int('0144') == 144` y `int('0x144')` lanzará **`ValueError`**. `int(string, base)` toma la base para convertirlo desde un segundo parámetro opcional, por tanto `int('0x144', 16) == 324`. Si la base se especifica como 0, el número se interpreta usando las reglas de Python’s rules: un prefijo “0o” indica octal y un prefijo “0x” indica un número hexadecimal.

No uses la función incorporada **`eval()`** si todo lo que necesitas es convertir cadenas a números. **`eval()`** será considerablemente más lento y presenta riesgos de seguridad: cualquiera podría introducir una expresión Python que presentara efectos indeseados. Por ejemplo, alguien podría pasar `__import__('os').system("rm -rf $HOME")` lo cual borraría el directorio home al completo.

`eval()` also has the effect of interpreting numbers as Python expressions, so that, for example, `eval('09')` gives a syntax error because Python does not allow leading “0” in a decimal number (except “0”).

¿Cómo puedo convertir un número a una cadena?

For example, to convert the number `144` to the string `'144'`, use the built-in type constructor **`str()`**. If you want a hexadecimal or octal representation, use the built-in functions **`hex()`** or **`oct()`**. For fancy formatting, see the f-strings and Format string syntax sections. For example, `"{:04d}".format(144)` yields `'0144'` and `"{:.3f}".format(1.0/3.0)` yields `'0.333'`.

¿Cómo puedo modificar una cadena in situ?

You can’t, because strings are immutable. In most situations, you should simply construct a new string from the various parts you want to assemble it from. However, if you need an object with the ability to modify in-place Unicode data, try using an **`io.StringIO`** object or the **`array`** module:

```
>>> import io
>>> s = "Hello, world"
>>> sio = io.StringIO(s)
>>> sio.getvalue()
'Hello, world'
>>> sio.seek(7)
7
>>> sio.write("there!")
```

```
6
>>> sio.getvalue()
'Hello, there!'

>>> import array
>>> a = array.array('w', s)
>>> print(a)
array('w', 'Hello, world')
>>> a[0] = 'y'
>>> print(a)
array('w', 'yello, world')
>>> a.tounicode()
'yello, world'
```

¿Cómo puedo usar cadenas para invocar funciones/métodos?

Existen varias técnicas.

- Lo mejor sería usar un diccionario que mapee cadenas a funciones. La principal ventaja de esta técnica es que las cadenas no necesitan ser iguales que los nombres de las funciones. Esta es también la principal técnica que se usa para emular un constructo *case*:

  ```
  def a():
      pass

  def b():
      pass

  dispatch = {'go': a, 'stop': b}  # Note lack of par

  dispatch[get_input()]()  # Note trailing parens to
  ```

- Usa la función incorporada **`getattr()`**:

  ```
  import foo
  getattr(foo, 'bar')()
  ```

Nótese que **`getattr()`** funciona en cualquier objeto, incluido clases, instancias de clases, módulos, etc.

Esto se usa en varios lugares de la biblioteca estándar, como esto:

```
class Foo:
    def do_foo(self):
        ...

    def do_bar(self):
        ...

f = getattr(foo_instance, 'do_' + opname)
f()
```

- Use **`locals()`** para resolver el nombre de la función:

```
def myFunc():
    print("hello")

fname = "myFunc"

f = locals()[fname]
f()
```

Is there an equivalent to Perl's **`chomp()`** for removing trailing newlines from strings?

Puedes usar `S.rstrip("\r\n")` para eliminar todas las ocurrencias de cualquier terminación de línea desde el final de la cadena `S` sin eliminar el resto de espacios en blanco que le siguen. Si la cadena `S` representa más de una línea con varias líneas vacías al final, las terminaciones de línea para todas las líneas vacías se eliminarán:

```
>>> lines = ("line 1 \r\n"
...          "\r\n"
...          "\r\n")
>>> lines.rstrip("\n\r")
```

```
'line 1 '
```

Ya que esto solo sería deseable, típicamente, cuando lees texto línea a línea, usar `S.rstrip()` de esta forma funcionaría bien.

Is there a `scanf()` or `sscanf()` equivalent?

No de la misma forma.

For simple input parsing, the easiest approach is usually to split the line into whitespace-delimited words using the `split()` method of string objects and then convert decimal strings to numeric values using `int()` or `float()`. **`split()`** supports an optional «sep» parameter which is useful if the line uses something other than whitespace as a separator.

For more complicated input parsing, regular expressions are more powerful than C's `sscanf` and better suited for the task.

What does `UnicodeDecodeError` or `UnicodeEncodeError` error mean?

Ver CÓMO (HOWTO) Unicode.

Can I end a raw string with an odd number of backslashes?

A raw string ending with an odd number of backslashes will escape the string's quote:

```
>>> r'C:\this\will\not\work\'
  File "<stdin>", line 1
    r'C:\this\will\not\work\'
    ^
SyntaxError: unterminated string literal (detected at li
```

There are several workarounds for this. One is to use regular strings and double the backslashes:

```
>>> 'C:\\this\\will\\work\\'
```

```
'C:\\this\\will\\work\\'
```

Another is to concatenate a regular string containing an escaped backslash to the raw string:

```
>>> r'C:\this\will\work' '\\'
'C:\\this\\will\\work\\'
```

It is also possible to use **`os.path.join()`** to append a backslash on Windows:

```
>>> os.path.join(r'C:\this\will\work', '')
'C:\\this\\will\\work\\'
```

Note that while a backslash will «escape» a quote for the purposes of determining where the raw string ends, no escaping occurs when interpreting the value of the raw string. That is, the backslash remains present in the value of the raw string:

```
>>> r'backslash\'preserved'
"backslash\\'preserved"
```

Also see the specification in the language reference.

Rendimiento

Mi programa es muy lento. ¿Cómo puedo acelerarlo?

That's a tough one, in general. First, here is a list of things to remember before diving further:

- Las características del rendimiento varían entre las distintas implementaciones de Python. Estas preguntas frecuentes se enfocan en CPython.
- El comportamiento puede variar entre distintos sistemas operativos, especialmente cuando se habla de tareas I/O o multi-tarea.
- Siempre deberías encontrar las partes importantes en tu programa *antes* de intentar optimizar el código (ver el módulo **`profile`**).

- Escribir programas de comparación del rendimiento te permitirá iterar rápidamente cuando te encuentres buscando mejoras (ver el módulo `timeit`).
- Es altamente recomendable disponer de una buena cobertura de código (a partir de pruebas unitarias o cualquier otra técnica) antes de introducir potenciales regresiones ocultas en sofisticadas optimizaciones.

Dicho lo anterior, existen muchos trucos para acelerar código Python. Aquí tienes algunos principios generales que te permitirán llegar a alcanzar niveles de rendimiento aceptables:

- El hacer más rápido tu algoritmo (o cambiarlo por alguno más rápido) puede provocar mayores beneficios que intentar unos pocos trucos de micro-optimización a través de todo tu código.
- Utiliza las estructuras de datos correctas. Estudia la documentación para los Tipos integrados y el módulo `collections`.
- Cuando la biblioteca estándar proporciona una primitiva de hacer algo, esta supuestamente será (aunque no se garantiza) más rápida que cualquier otra alternativa que se te ocurra. Esto es doblemente cierto si las primitivas han sido escritas en C, como los *builtins* y algunos tipos extendidos. Por ejemplo, asegúrate de usar el método integrado `list.sort()` o la función relacionada `sorted()` para ordenar (y ver Sorting Techniques para ver ejemplos de uso moderadamente avanzados).
- Las abstracciones tienden a crear rodeos y fuerzan al intérprete a trabajar más. Si el nivel de rodeos sobrepasa el trabajo útil realizado tu programa podría ser más lento. Deberías evitar abstracciones excesivas, especialmente, en forma de pequeñas funciones o métodos (que también va en detrimento de la legibilidad).

If you have reached the limit of what pure Python can allow, there are tools to take you further away. For example, Cython [https://cython.org] can compile a slightly modified version of Python code into a C extension, and can be used on many different platforms. Cython can take advantage of compilation (and optional type annotations) to make your code significantly faster than when interpreted. If you are confident in your C programming skills, you can also write a C extension module yourself.

Ver también

La página de la wiki dedicada a trucos de rendimiento [https://wiki.python.org/moin/PythonSpeed/PerformanceTips].

¿Cuál es la forma más eficiente de concatenar muchas cadenas conjuntamente?

Los objetos `str` y `bytes` son inmutables, por tanto, concatenar muchas cadenas en una sola es ineficiente debido a que cada concatenación crea un nuevo objeto. En el caso más general, el coste total en tiempo de ejecución es cuadrático en relación a la longitud de la cadena final.

Para acumular muchos objetos `str`, la forma recomendada sería colocarlos en una lista y llamar al método `str.join()` al final:

```
chunks = []
for s in my_strings:
    chunks.append(s)
result = ''.join(chunks)
```

(Another reasonably efficient idiom is to use `io.StringIO`.)

Para acumular muchos objetos `bytes`, la forma recomendada sería extender un objeto `bytearray` usando el operador de concatenación in situ (el operador `+=`):

```
result = bytearray()
for b in my_bytes_objects:
    result += b
```

Sequences (tuples/lists)

¿Cómo convertir entre tuplas y listas?

El constructor `tuple(seq)` convierte cualquier secuencia (en realidad, cualquier iterable) en una tupla con los mismos elementos y en el mismo orden.

Por ejemplo, `tuple([1, 2, 3])` lo convierte en `(1, 2, 3)` y `tuple('abc')` lo convierte en `('a', 'b', 'c')`. Si el argumento es una tupla no creará una nueva copia y retornará el mismo objeto, por tanto, llamar a **`tuple()`** no tendrá mucho coste si no estás seguro si un objeto ya es una tupla.

El constructor `list(seq)` convierte cualquier secuencia o iterable en una lista con los mismos elementos y en el mismo orden. Por ejemplo, `list((1, 2, 3))` lo convierte a `[1, 2, 3]` y `list('abc')` lo convierte a `['a', 'b', 'c']`. Si el argumento es una lista, hará una copia como lo haría `seq[:]`.

¿Qué es un índice negativo?

Las secuencias en Python están indexadas con números positivos y negativos. Para los números positivos el 0 será el primer índice, el 1 el segundo y así en adelante. Para los índices negativos el -1 el último índice, el -2 el penúltimo, etc. Piensa en `seq[-n]` como si fuera `seq[len(seq)-n]`.

El uso de índices negativos puede ser muy conveniente. Por ejemplo `S[:-1]` se usa para todo la cadena excepto para su último carácter, lo cual es útil para eliminar el salto de línea final de una cadena.

¿Cómo puedo iterar sobre una secuencia en orden inverso?

Usa la función incorporada **`reversed()`**:

```
for x in reversed(sequence):
    ...  # do something with x ...
```

Esto no transformará la secuencia original sino que creará una nueva copia en orden inverso por la que se puede iterar.

¿Cómo eliminar duplicados de una lista?

Puedes echar un vistazo al recetario de Python para ver una gran discusión mostrando muchas formas de hacer esto:

https://code.activestate.com/recipes/52560/

Si no te preocupa que la lista se reordene la puedes ordenar y, después, y después escanearla desde el final borrando duplicados a medida que avanzas:

```
if mylist:
    mylist.sort()
    last = mylist[-1]
    for i in range(len(mylist)-2, -1, -1):
        if last == mylist[i]:
            del mylist[i]
        else:
            last = mylist[i]
```

If all elements of the list may be used as set keys (that is, they are all hashable) this is often faster:

```
mylist = list(set(mylist))
```

Esto convierte la lista en un conjunto eliminando, por tanto, los duplicados y, posteriormente, puedes volver a una lista.

How do you remove multiple items from a list?

As with removing duplicates, explicitly iterating in reverse with a delete condition is one possibility. However, it is easier and faster to use slice replacement with an implicit or explicit forward iteration. Here are three variations:

```
mylist[:] = filter(keep_function, mylist)
mylist[:] = (x for x in mylist if keep_condition)
mylist[:] = [x for x in mylist if keep_condition]
```

Esta comprensión de lista puede ser la más rápida.

¿Cómo se puede hacer un array en Python?

Usa una lista:

```
["this", 1, "is", "an", "array"]
```

Las listas son equivalentes en complejidad temporal a arrays en C o Pascal; La principal diferencia es que una lista en Python puede contener objetos de diferentes tipos.

The `array` module also provides methods for creating arrays of fixed types with compact representations, but they are slower to index than lists. Also note that NumPy [https://numpy.org/] and other third-party packages define array-like structures with various characteristics as well.

To get Lisp-style linked lists, you can emulate *cons cells* using tuples:

```
lisp_list = ("like",  ("this",  ("example", None) ) )
```

If mutability is desired, you could use lists instead of tuples. Here the analogue of a Lisp *car* is `lisp_list[0]` and the analogue of *cdr* is `lisp_list[1]`. Only do this if you're sure you really need to, because it's usually a lot slower than using Python lists.

¿Cómo puedo crear una lista multidimensional?

Seguramente hayas intentado crear un array multidimensional de la siguiente forma:

```
>>> A = [[None] * 2] * 3
```

Esto parece correcto si lo muestras en pantalla:

```
>>> A
[[None, None], [None, None], [None, None]]
```

Pero cuando asignas un valor, se muestra en múltiples sitios:

```
>>> A[0][0] = 5
>>> A
[[5, None], [5, None], [5, None]]
```

La razón es que replicar una lista con `*` no crea copias, solo crea referencias a los objetos existentes. El `*3` crea una lista conteniendo 3 referencias a la misma lista de longitud dos. Cambios a una fila se mostrarán en todas las filas, lo cual, seguramente, no es lo que deseas.

El enfoque recomendado sería crear, primero, una lista de la longitud deseada y, después, rellenar cada elemento con una lista creada en ese momento:

```
A = [None] * 3
for i in range(3):
    A[i] = [None] * 2
```

Esto genera una lista conteniendo 3 listas distintas de longitud dos. También puedes usar una comprensión de lista:

```
w, h = 2, 3
A = [[None] * w for i in range(h)]
```

Or, you can use an extension that provides a matrix datatype; NumPy [https://numpy.org/] is the best known.

How do I apply a method or function to a sequence of objects?

To call a method or function and accumulate the return values in a list, a list comprehension is an elegant solution:

```
result = [obj.method() for obj in mylist]

result = [function(obj) for obj in mylist]
```

To just run the method or function without saving the return values, a plain **`for`** loop will suffice:

```
for obj in mylist:
    obj.method()

for obj in mylist:
    function(obj)
```

¿Por qué hacer lo siguiente, `a_tuple[i] += ['item']`, lanza una excepción cuando la suma funciona?

Esto es debido a la combinación del hecho de que un operador de asignación aumentada es un operador de *asignación* y a la diferencia entre objetos mutables e inmutable en Python.

Esta discusión aplica, en general, cuando los operadores de asignación aumentada se aplican a elementos de una tupla que apuntan a objetos mutables. Pero vamos a usar una `lista` y `+=` para el ejemplo.

Si escribes:

```
>>> a_tuple = (1, 2)
>>> a_tuple[0] += 1
Traceback (most recent call last):
   ...
TypeError: 'tuple' object does not support item assignme
```

La razón por la que se produce la excepción debería ser evidente: `1` se añade al objeto `a_tuple[0]` que apunta a (`1`), creando el objeto resultante, `2`, pero cuando intentamos asignar el resultado del cálculo, `2`, al elemento `0` de la tupla, obtenemos un error debido a que no podemos cambiar el elemento al que apunta la tupla.

En realidad, lo que esta declaración de asignación aumentada está haciendo es, aproximadamente, lo siguiente:

```
>>> result = a_tuple[0] + 1
>>> a_tuple[0] = result
Traceback (most recent call last):
  ...
TypeError: 'tuple' object does not support item assignme
```

Es la parte de asignación de la operación la que provoca el error, debido a que una tupla es inmutable.

Cuando escribes algo como lo siguiente:

```
>>> a_tuple = (['foo'], 'bar')
>>> a_tuple[0] += ['item']
Traceback (most recent call last):
  ...
TypeError: 'tuple' object does not support item assignme
```

La excepción es un poco más sorprendente e, incluso, más sorprendente es el hecho que aunque hubo un error, la agregación funcionó:

```
>>> a_tuple[0]
['foo', 'item']
```

To see why this happens, you need to know that (a) if an object implements an `__iadd__()` magic method, it gets called when the `+=` augmented assignment is executed, and its return value is what gets used in the assignment statement; and (b) for lists, `__iadd__()` is equivalent to calling `extend()` on the list and returning the list. That's why we say that for lists, `+=` is a «shorthand» for `list.extend()`:

```
>>> a_list = []
>>> a_list += [1]
>>> a_list
[1]
```

Esto es equivalente a

```
>>> result = a_list.__iadd__([1])
>>> a_list = result
```

El objeto al que apunta a_list ha mutado y el puntero al objeto mutado es asignado de vuelta a `a_list`. El resultado final de la asignación no es opción debido a que es un puntero al mismo objeto al que estaba apuntando `a_list` pero la asignación sí que ocurre.

Por tanto, en nuestro ejemplo con tupla lo que está pasando es equivalente a:

```
>>> result = a_tuple[0].__iadd__(['item'])
>>> a_tuple[0] = result
Traceback (most recent call last):
  ...
TypeError: 'tuple' object does not support item assignme
```

The **`__iadd__()`** succeeds, and thus the list is extended, but even though `result` points to the same object that `a_tuple[0]` already points to, that final assignment still results in an error, because tuples are immutable.

Quiero hacer una ordenación compleja: ¿Puedes hacer una transformada Schwartziana (Schwartzian Transform) en Python?

La técnica, atribuida a Randal Schwartz, miembro de la comunidad Perl, ordena los elementos de una lista mediante una métrica que mapea cada elemento a su «valor orden». En Python, usa el argumento `key` par el método **`list.sort()`**:

```
Isorted = L[:]
Isorted.sort(key=lambda s: int(s[10:15]))
```

¿Cómo puedo ordenar una lista a partir de valores de otra lista?

Merge them into an iterator of tuples, sort the resulting list, and then pick out the element you want.

```
>>> list1 = ["what", "I'm", "sorting", "by"]
>>> list2 = ["something", "else", "to", "sort"]
>>> pairs = zip(list1, list2)
>>> pairs = sorted(pairs)
>>> pairs
[("I'm", 'else'), ('by', 'sort'), ('sorting', 'to'), ('w
>>> result = [x[1] for x in pairs]
>>> result
['else', 'sort', 'to', 'something']
```

Objetos

¿Qué es una clase?

Una clase es un tipo de objeto particular creado mediante la ejecución de la declaración class. Los objetos class se usan como plantillas para crear instancias de objetos que son tanto los datos (atributos) como el código (métodos) específicos para un tipo de dato.

Una clase puede estar basada en una o más clases diferentes, llamadas su(s) clase(s). Hereda los atributos y métodos de sus clases base. Esto permite que se pueda refinar un objeto modelo de forma sucesiva mediante herencia. Puedes tener una clase genérica `Mailbox` que proporciona métodos de acceso básico para un buzón de correo y subclases como `MboxMailbox`, `MaildirMailbox`, `OutlookMailbox` que gestionan distintos formatos específicos de buzón de correos.

¿Qué es un método?

Un método es una función de un objeto `x` que puedes llamar, normalmente, de la forma `x.name(arguments...)`. Los métodos se definen como funciones dentro de la definición de la clase:

```
class C:
    def meth(self, arg):
        return arg * 2 + self.attribute
```

¿Qué es self?

Self es, básicamente, un nombre que se usa de forma convencional como primer argumento de un método. Un método definido como `meth(self, a, b, c)` se le llama como `x.meth(a, b, c)` para una instancia `x` de la clase es que se definió; el método invocado pensará que se le ha invocado como `meth(x, a, b, c)`.

Ver también ¿Por qué debe usarse "self" explícitamente en las

definiciones y llamadas de métodos?.

¿Cómo puedo comprobar si un objeto es una instancia de una clase dada o de una subclase de la misma?

Use the built-in function **`isinstance(obj, cls)`**. You can check if an object is an instance of any of a number of classes by providing a tuple instead of a single class, for example, `isinstance(obj, (class1, class2, ...))`, and can also check whether an object is one of Python's built-in types, for example, `isinstance(obj, str)` or `isinstance(obj, (int, float, complex))`.

Note that **`isinstance()`** also checks for virtual inheritance from an abstract base class. So, the test will return `True` for a registered class even if hasn't directly or indirectly inherited from it. To test for «true inheritance», scan the method resolution order (MRO) of the class:

```
from collections.abc import Mapping

class P:
     pass

class C(P):
    pass

Mapping.register(P)

>>> c = C()
>>> isinstance(c, C)        # direct
True
>>> isinstance(c, P)        # indirect
True
>>> isinstance(c, Mapping)  # virtual
True

# Actual inheritance chain
>>> type(c).__mro__
```

```
(<class 'C'>, <class 'P'>, <class 'object'>)

# Test for "true inheritance"
>>> Mapping in type(c).__mro__
False
```

Destacar que muchos programas no necesitan usar **`isinstance()`** de forma frecuente en clases definidas por el usuario. Si estás desarrollando clases un mejor estilo orientado a objetos sería el de definir los métodos en las clases que encapsulan un comportamiento en particular en lugar de ir comprobando la clase del objeto e ir haciendo cosas en base a la clase que es. Por ejemplo, si tienes una función que hace lo siguiente:

```
def search(obj):
    if isinstance(obj, Mailbox):
        ...  # code to search a mailbox
    elif isinstance(obj, Document):
        ...  # code to search a document
    elif ...
```

Un enfoque más adecuado sería definir un método `search()` en todas las clases e invocarlo:

```
class Mailbox:
    def search(self):
        ...  # code to search a mailbox

class Document:
    def search(self):
        ...  # code to search a document

obj.search()
```

¿Qué es la delegación?

Delegation is an object-oriented technique (also called a design pattern). Let’s say you have an object `x` and want to change the behaviour of just one of its methods. You can create a new class that provides a new implementation of the method you’re interested

in changing and delegates all other methods to the corresponding method of `x`.

Los programadores Python pueden implementar la delegación de forma muy sencilla. Por ejemplo, la siguiente clase implementa una clase que se comporta como un fichero pero convierte todos los datos escritos a mayúsculas:

```
class UpperOut:

    def __init__(self, outfile):
        self._outfile = outfile

    def write(self, s):
        self._outfile.write(s.upper())

    def __getattr__(self, name):
        return getattr(self._outfile, name)
```

Here the `UpperOut` class redefines the `write()` method to convert the argument string to uppercase before calling the underlying `self._outfile.write()` method. All other methods are delegated to the underlying `self._outfile` object. The delegation is accomplished via the **`__getattr__()`** method; consult the language reference for more information about controlling attribute access.

Note that for more general cases delegation can get trickier. When attributes must be set as well as retrieved, the class must define a **`__setattr__()`** method too, and it must do so carefully. The basic implementation of **`__setattr__()`** is roughly equivalent to the following:

```
class X:
    ...
    def __setattr__(self, name, value):
        self.__dict__[name] = value
    ...
```

Many **`__setattr__()`** implementations call

`object.__setattr__()` to set an attribute on self without causing infinite recursion:

```
class X:
    def __setattr__(self, name, value):
        # Custom logic here...
        object.__setattr__(self, name, value)
```

Alternatively, it is possible to set attributes by inserting entries into **`self.__dict__`** directly.

¿Cómo invoco a un método definido en una clase base desde una clase derivada que la extiende?

Usa la función incorporada **`super()`**:

```
class Derived(Base):
    def meth(self):
        super().meth()  # calls Base.meth
```

En el ejemplo, **`super()`** automáticamente determinará la instancia desde la cual ha sido llamada (el valor `` self` ``), busca el method resolution order (MRO) con `type(self).__mro__`, y devuelve el siguiente en línea después de `Derived` en el MRO: `Base`.

¿Cómo puedo organizar mi código para hacer que sea más sencillo modificar la clase base?

You could assign the base class to an alias and derive from the alias. Then all you have to change is the value assigned to the alias. Incidentally, this trick is also handy if you want to decide dynamically (such as depending on availability of resources) which base class to use. Example:

```
class Base:
    ...

BaseAlias = Base

class Derived(BaseAlias):
    ...
```

¿Cómo puedo crear datos estáticos de clase y métodos estáticos de clase?

Tanto los datos estáticos como los métodos estáticos (en el sentido de C++ o Java) están permitidos en Python.

Para datos estáticos simplemente define un atributo de clase. Para asignar un nuevo valor al atributo debes usar de forma explícita el nombre de la clase en la asignación:

```
class C:
    count = 0   # number of times C.__init__ called

    def __init__(self):
        C.count = C.count + 1

    def getcount(self):
        return C.count  # or return self.count
```

`c.count` también se refiere a `C.count` para cualquier `c` de tal forma que se cumpla `isinstance(c, C)`, a no ser que `c` sea sobreescrita por si misma o por alguna clase contenida en la búsqueda de clases base desde `c.__class__` hasta `C`.

Debes tener cuidado: dentro de un método de C, una asignación como `self.count = 42` creará una nueva instancia sin relación con la original que se llamará «count» en el propio diccionario de `self`. El reunificar el nombre de datos estáticos de una clase debería llevar, siempre, a especificar la clase tanto si se produce desde dentro de un método como si no:

```
C.count = 314
```

Los métodos estáticos son posibles:

```
class C:
    @staticmethod
    def static(arg1, arg2, arg3):
        # No 'self' parameter!
        ...
```

Sin embargo, una forma más directa de obtener el efecto de un método estático sería mediante una simple función a nivel de módulo:

```
def getcount():
    return C.count
```

Si has estructurado tu código para definir una clase única (o una jerarquía de clases altamente relacionadas) por módulo, esto proporcionará la encapsulación deseada.

¿Como puedo sobrecargar constructores (o métodos) en Python?

Esta respuesta es aplicable, en realidad, a todos los métodos pero la pregunta suele surgir primero en el contexto de los constructores.

In C++ you'd write:

```
class C {
    C() { cout << "No arguments\n"; }
    C(int i) { cout << "Argument is " << i << "\n"; }
}
```

En Python solo debes escribir un único constructor que tenga en cuenta todos los casos usando los argumentos por defecto. Por ejemplo:

```
class C:
    def __init__(self, i=None):
        if i is None:
            print("No arguments")
        else:
            print("Argument is", i)
```

Esto no es totalmente equivalente pero, en la práctica, es muy similar.

You could also try a variable-length argument list, for example:

```
def __init__(self, *args):
    ...
```

El mismo enfoque funciona para todas las definiciones de métodos.

Intento usar _spam y obtengo un error sobre _SomeClassName_spam.

Nombres de variable con doble guión prefijado se convierten, con una modificación de nombres, para proporcionar una forma simple pero efectiva de definir variables de clase privadas. Cualquier identificador de la forma `__spam` (como mínimo dos guiones bajos como prefijo, como máximo un guión bajo como sufijo) se reemplaza con `_classname__spam`, donde `classname` es el nombre de la clase eliminando cualquier guión bajo prefijado.

The identifier can be used unchanged within the class, but to access it outside the class, the mangled name must be used:

```
class A:
    def __one(self):
        return 1
    def two(self):
        return 2 * self.__one()

class B(A):
    def three(self):
        return 3 * self._A__one()

four = 4 * A()._A__one()
```

In particular, this does not guarantee privacy since an outside user can still deliberately access the private attribute; many Python programmers never bother to use private variable names at all.

Ver también

The private name mangling specifications for details and special cases.

Mi clase define _del_ pero no se le invoca cuando borro el objeto.

Existen varias razones posibles para que suceda así.

The `del` statement does not necessarily call `__del__()` – it simply decrements the object’s reference count, and if this reaches zero `__del__()` is called.

If your data structures contain circular links (for example, a tree where each child has a parent reference and each parent has a list of children) the reference counts will never go back to zero. Once in a while Python runs an algorithm to detect such cycles, but the garbage collector might run some time after the last reference to your data structure vanishes, so your `__del__()` method may be called at an inconvenient and random time. This is inconvenient if you’re trying to reproduce a problem. Worse, the order in which object’s `__del__()` methods are executed is arbitrary. You can run `gc.collect()` to force a collection, but there *are* pathological cases where objects will never be collected.

Despite the cycle collector, it’s still a good idea to define an explicit `close()` method on objects to be called whenever you’re done with them. The `close()` method can then remove attributes that refer to subobjects. Don’t call `__del__()` directly – `__del__()` should call `close()` and `close()` should make sure that it can be called more than once for the same object.

Otra forma de evitar referencias cíclicas sería usando el módulo `weakref`, que permite apuntar hacia objetos sin incrementar su conteo de referencias. Las estructuras de datos en árbol, por ejemplo, deberían usar referencias débiles para las referencias del padre y hermanos (¡si es que las necesitan!).

Finally, if your `__del__()` method raises an exception, a warning message is printed to `sys.stderr`.

¿Cómo puedo obtener una lista de todas las instancias

de una clase dada?

Python no hace seguimiento de todas las instancias de una clase (o de los tipos incorporados). Puedes programar el constructor de una clase para que haga seguimiento de todas sus instancias manteniendo una lista de referencias débiles a cada instancia.

¿Por qué el resultado de `id()` no parece ser único?

La función incorporada `id()` devuelve un entero que se garantiza que sea único durante la vida del objeto. Debido a que en CPython esta es la dirección en memoria del objeto, sucede que, frecuentemente, después de que un objeto se elimina de la memoria el siguiente objeto recién creado se localiza en la misma posición en memoria. Esto se puede ver ilustrado en este ejemplo:

```
>>> id(1000)
13901272
>>> id(2000)
13901272
```

Las dos ids pertenecen a dos objetos “entero” diferentes que se crean antes y se eliminan inmediatamente después de la ejecución de la invocación a `id()`. Para estar seguro que los objetos cuya id quieres examinar siguen vivos crea otra referencia al objeto:

```
>>> a = 1000; b = 2000
>>> id(a)
13901272
>>> id(b)
13891296
```

¿Cuándo puedo fiarme de pruebas de identidad con el operador *is*?

El operador `is` verifica la identidad de un objeto. La prueba `a is b` es equivalente a `id(a) == id(b)`.

La propiedad más importante de una prueba de identidad es que un objeto siempre es idéntico a si mismo, `a is a` siempre devuelve

`True`. Las pruebas de identidad suelen ser más rápidas que pruebas de igualdad. Y a diferencia de las pruebas de igualdad, las pruebas de identidad están garantizadas de devolver un booleano `True` o `False`.

Sin embargo, las pruebas de identidad *solo* pueden ser sustituidas por pruebas de igualdad cuando la identidad de objeto está asegurada. Generalmente hay tres circunstancias en las que la identidad está garantizada:

1. Assignments create new names but do not change object identity. After the assignment `new = old`, it is guaranteed that `new is old`.
2. Putting an object in a container that stores object references does not change object identity. After the list assignment `s[0] = x`, it is guaranteed that `s[0] is x`.
3. If an object is a singleton, it means that only one instance of that object can exist. After the assignments `a = None` and `b = None`, it is guaranteed that `a is b` because `None` is a singleton.

En la mayoría de las demás circunstancias, no se recomiendan las pruebas de identidad y las pruebas de igualdad son preferidas. En particular, las pruebas de identidad no deben ser usadas para verificar constantes como **`int`** y **`str`** que no están garantizadas a ser singletons:

```
>>> a = 10_000_000
>>> b = 5_000_000
>>> c = b + 5_000_000
>>> a is c
False

>>> a = 'Python'
>>> b = 'Py'
>>> c = b + 'thon'
>>> a is c
False
```

De la misma manera, nuevas instancias de contenedores mutables nunca son idénticas:

```
>>> a = []
>>> b = []
>>> a is b
False
```

En la librería estándar de código, verás varios patrones comunes para usar correctamente pruebas de identidad:

1. As recommended by **PEP 8** [https://peps.python.org/pep-0008/], an identity test is the preferred way to check for `None`. This reads like plain English in code and avoids confusion with other objects that may have boolean values that evaluate to false.

2. Detecting optional arguments can be tricky when `None` is a valid input value. In those situations, you can create a singleton sentinel object guaranteed to be distinct from other objects. For example, here is how to implement a method that behaves like **`dict.pop()`**:

   ```
   _sentinel = object()

   def pop(self, key, default=_sentinel):
       if key in self:
           value = self[key]
           del self[key]
           return value
       if default is _sentinel:
           raise KeyError(key)
       return default
   ```

3. Container implementations sometimes need to augment equality tests with identity tests. This prevents the code from being confused by objects such as `float('NaN')` that are not equal to themselves.

For example, here is the implementation of **`collections.abc.Sequence.__contains__()`**:

```
def __contains__(self, value):
    for v in self:
        if v is value or v == value:
            return True
    return False
```

¿Cómo puede una subclase controlar qué datos se almacenan en una instancia inmutable?

When subclassing an immutable type, override the `__new__()` method instead of the `__init__()` method. The latter only runs *after* an instance is created, which is too late to alter data in an immutable instance.

Todas estas clases inmutables tienen una firma distinta que su clase padre:

```
import datetime as dt

class FirstOfMonthDate(dt.date):
    "Always choose the first day of the month"
    def __new__(cls, year, month, day):
        return super().__new__(cls, year, month, 1)

class NamedInt(int):
    "Allow text names for some numbers"
    xlat = {'zero': 0, 'one': 1, 'ten': 10}
    def __new__(cls, value):
        value = cls.xlat.get(value, value)
        return super().__new__(cls, value)

class TitleStr(str):
    "Convert str to name suitable for a URL path"
    def __new__(cls, s):
        s = s.lower().replace(' ', '-')
        s = ''.join([c for c in s if c.isalnum() or c ==
        return super().__new__(cls, s)
```

Las clases pueden ser utilizadas así:

```
>>> FirstOfMonthDate(2012, 2, 14)
FirstOfMonthDate(2012, 2, 1)
>>> NamedInt('ten')
10
>>> NamedInt(20)
20
>>> TitleStr('Blog: Why Python Rocks')
'blog-why-python-rocks'
```

¿Cómo cacheo llamadas de método?

Las dos herramientas principales para cachear métodos son **`functools.cached_property()`** y **`functools.lru_cache()`**. El primero guarda resultados a nivel de instancia y el último a nivel de clase.

The `cached_property` approach only works with methods that do not take any arguments. It does not create a reference to the instance. The cached method result will be kept only as long as the instance is alive.

The advantage is that when an instance is no longer used, the cached method result will be released right away. The disadvantage is that if instances accumulate, so too will the accumulated method results. They can grow without bound.

The `lru_cache` approach works with methods that have hashable arguments. It creates a reference to the instance unless special efforts are made to pass in weak references.

La ventaja del algoritmo usado menos recientemente es que el cache está limitado por el *maxsize* especificado. La desventaja es que las instancias se mantienen activas hasta que sean eliminadas del cache por edad o que el cache sea borrado.

Este ejemplo muestra las diversas técnicas:

```
class Weather:
    "Lookup weather information on a government website"
    def __init__(self, station_id):
        self._station_id = station_id
        # The _station_id is private and immutable

    def current_temperature(self):
        "Latest hourly observation"
        # Do not cache this because old results
        # can be out of date.

    @cached_property
    def location(self):
        "Return the longitude/latitude coordinates of th
        # Result only depends on the station_id

    @lru_cache(maxsize=20)
    def historic_rainfall(self, date, units='mm'):
        "Rainfall on a given date"
        # Depends on the station_id, date, and units.
```

The above example assumes that the *station_id* never changes. If the relevant instance attributes are mutable, the `cached_property` approach can't be made to work because it cannot detect changes to the attributes.

To make the `lru_cache` approach work when the *station_id* is mutable, the class needs to define the **`__eq__()`** and **`__hash__()`** methods so that the cache can detect relevant attribute updates:

```
class Weather:
    "Example with a mutable station identifier"

    def __init__(self, station_id):
        self.station_id = station_id

    def change_station(self, station_id):
        self.station_id = station_id

    def __eq__(self, other):
        return self.station_id == other.station_id

    def __hash__(self):
        return hash(self.station_id)

    @lru_cache(maxsize=20)
    def historic_rainfall(self, date, units='cm'):
        'Rainfall on a given date'
        # Depends on the station_id, date, and units.
```

Módulos

¿Cómo creo un fichero .pyc?

Cuando se importa un módulo por primera vez (o cuando el código fuente ha cambiado desde que el fichero compilado se creó) un fichero `.pyc` que contiene el código compilado se debería crear en la subcarpeta `__pycache__` del directorio que contiene al fichero `.py`. El fichero `.pyc` tendrá un nombre que empezará con el mismo nombre que el del fichero `.py` y terminará con `.pyc`, con un componente intermedio que dependerá del binario `python` en particular que lo creó. (Ver **PEP 3147** [https://peps.python.org/pep-3147/] para detalles.)

Una razón por la que no se cree un fichero `.pyc` podría ser debido a un problema de permisos del directorio que contiene al fichero fuente, lo que significa que el subdirectorio `__pycache__` no se puede crear. Esto puede suceder, por ejemplo, si desarrollas como un usuario pero lo ejecutas como otro, como si estuvieras probando en un servidor web.

Unless the **`PYTHONDONTWRITEBYTECODE`** environment variable is set, creation of a .pyc file is automatic if you’re importing a module and Python has the ability (permissions, free space, and so on) to create a `__pycache__` subdirectory and write the compiled module to that subdirectory.

Running Python on a top-level script is not considered an import and no `.pyc` will be created. For example, if you have a top-level module `foo.py` that imports another module `xyz.py`, when you run `foo` (by typing `python foo.py` as a shell command), a `.pyc` will be created for `xyz` because `xyz` is imported, but no `.pyc` file will be created for `foo` since `foo.py` isn't being imported.

Si necesitas crear un fichero `.pyc` también para `foo` – es decir, crear un fichero `.pyc` para un módulo que no ha sido importado – puedes usar los módulos **`py_compile`** y **`compileall`**.

El módulo **`py_compile`** puede compilar manualmente cualquier módulo. Una forma sería usando la función `compile()` de ese módulo de forma interactiva:

```
>>> import py_compile
>>> py_compile.compile('foo.py')
```

This will write the `.pyc` to a `__pycache__` subdirectory in the same location as `foo.py` (or you can override that with the optional parameter *cfile*).

Puedes compilar automáticamente todos los ficheros en un directorio o directorios usando el módulo **`compileall`**. Lo puedes hacer desde la línea de comandos ejecutando `compileall.py` y proporcionando una ruta al directorio que contiene los ficheros Python a compilar:

```
python -m compileall .
```

¿Cómo puedo encontrar el nombre del módulo en uso?

Un módulo puede encontrar su propio nombre mirando en la variable global predeterminada `__name__`. Si tiene el valor `'__main__'`, el programa se está ejecutando como un script. Muchos módulos que se usan, generalmente, importados en otro script proporcionan, además, una interfaz para la línea de comandos o para probarse a si mismos y solo ejecutan código después de comprobar `__name__`:

```
def main():
    print('Running test...')
    ...

if __name__ == '__main__':
    main()
```

¿Cómo podría tener módulos que se importan mutuamente entre ellos?

Supón que tienes los siguientes módulos:

`foo.py`:

```
from bar import bar_var
foo_var = 1
```

`bar.py`:

```
from foo import foo_var
bar_var = 2
```

El problema es que el intérprete realizará los siguientes pasos:

- main importa a `foo`
- Se crean *globals* vacíos para foo
- `foo` se compila y se comienza a ejecutar
- `foo` importa a `bar`
- Se crean *globals* vacíos para `bar`
- `bar` se compila y se comienza a ejecutar
- `bar` importa a `foo` (lo cual es un no-op ya que ya hay un módulo que se llama `foo`)
- El mecanismo de importado intenta leer `foo_var` de globales de `foo`, para establecer `bar.foo_var = foo.foo_var`

El último paso falla debido a que Python todavía no ha terminado de interpretar a `foo` y el diccionario de símbolos global para `foo` todavía se encuentra vacío.

Lo mismo ocurre cuando usas `import foo` y luego tratas de acceder a `foo.foo_var` en un código global.

Existen (al menos) tres posibles soluciones para este problema.

Guido van Rossum recomienda evitar todos los usos de `from <module> import ...`, y colocar todo el código dentro de funciones. La inicialización de variables globales y variables de clase debería usar únicamente constantes o funciones incorporadas . Esto significa que todo se referenciará como `<module>.<name>` desde un módulo importado.

Jim Roskind sugiere realizar los siguientes pasos en el siguiente orden en cada módulo:

- exportar (*globals*, funciones y clases que no necesitan clases bases importadas)
- `import` declaraciones
- código activo (incluyendo *globals* que han sido inicializados desde valores importados).

Van Rossum doesn't like this approach much because the imports appear in a strange place, but it does work.

Matthias Urlichs recomienda reestructurar tu código de tal forma que un import recursivo no sea necesario.

Estas soluciones no son mutuamente excluyentes.

import("x.y.z") devuelve <module "x">; ¿cómo puedo obtener z?

Considera, en su lugar, usa la función de conveniencia **`import_module()`** de **`importlib`**:

```
z = importlib.import_module('x.y.z')
```

Cuando edito un módulo importado y lo reimporto los cambios no tienen efecto. ¿Por qué sucede esto?

Por razones de eficiencia además de por consistencia, Python solo lee el fichero del módulo la primera vez que el módulo se importa. Si no lo hiciera así, un programa escrito en muchos módulos donde cada módulo importa al mismo módulo básico estaría analizando sintácticamente el mismo módulo básico muchas veces. Para forzar una relectura de un módulo que ha sido modificado haz lo siguiente:

```
import importlib
import modname
importlib.reload(modname)
```

Warning: this technique is not 100% fool-proof. In particular, modules containing statements like:

```
from modname import some_objects
```

continuarán funcionando con la versión antigua de los objetos importados. Si el módulo contiene definiciones de clase, instancias de clase ya existentes *no* se actualizarán para usar la nueva definición de la clase. Esto podría resultar en el comportamiento paradójico siguiente:

```
>>> import importlib
>>> import cls
>>> c = cls.C()                # Create an instance of C
>>> importlib.reload(cls)
<module 'cls' from 'cls.py'>
>>> isinstance(c, cls.C)       # isinstance is false?!?
False
```

La naturaleza del problema se hace evidente si muestras la «identity» de los objetos clase:

```
>>> hex(id(c.__class__))
'0x7352a0'
>>> hex(id(cls.C))
'0x4198d0'
```

Preguntas frecuentes sobre diseño e historia

Contenido

- Preguntas frecuentes sobre diseño e historia
    - ¿Por qué Python usa indentación para agrupar declaraciones?
    - ¿Por qué obtengo resultados extraños con operaciones aritméticas simples?
    - ¿Por qué los cálculos de punto flotante son tan inexactos?
    - ¿Por qué las cadenas de caracteres de Python son inmutables?
    - ¿Por qué debe usarse “self” explícitamente en las definiciones y llamadas de métodos?
    - ¿Por qué no puedo usar una tarea en una expresión?
    - ¿Por qué Python usa métodos para alguna funcionalidad (por ejemplo, list.index()) pero funciones para otra (por ejemplo, len(list))?
    - ¿Por qué join() es un método de cadena de caracteres en lugar de un método de lista o tupla?
    - ¿Qué tan rápido van las excepciones?
    - ¿Por qué no hay un *switch* o una declaración *case* en Python?
    - ¿No puede emular hilos en el intérprete en lugar de confiar en una implementación de hilos específica del sistema operativo?
    - ¿Por qué las expresiones lambda no pueden contener sentencias?
    - ¿Se puede compilar Python en código máquina, C o algún otro lenguaje?
    - ¿Cómo gestiona Python la memoria?
    - ¿Por qué CPython no utiliza un esquema de recolección de basura más tradicional?

- ¿Por qué no se libera toda la memoria cuando sale CPython?
- ¿Por qué hay tipos de datos separados de tuplas y listas?
- ¿Cómo se implementan las listas en Python?
- ¿Cómo se implementan los diccionarios en CPython?
- ¿Por qué las claves del diccionario deben ser inmutables?
- ¿Por qué list.sort() no retorna la lista ordenada?
- ¿Cómo se especifica y aplica una especificación de interfaz en Python?
- ¿Por qué no hay goto?
- ¿Por qué las cadenas de caracteres sin formato (r-strings) no pueden terminar con una barra diagonal inversa?
- ¿Por qué Python no tiene una declaración «with» para las asignaciones de atributos?
- ¿Por qué los generadores no admiten la declaración with?
- ¿Por qué se requieren dos puntos para las declaraciones if/while/def/class?
- ¿Por qué Python permite comas al final de las listas y tuplas?

¿Por qué Python usa indentación para agrupar declaraciones?

Guido van Rossum cree que usar indentación para agrupar es extremadamente elegante y contribuye mucho a la claridad del programa Python promedio. La mayoría de las personas aprenden a amar esta característica después de un tiempo.

Como no hay corchetes de inicio/fin, no puede haber un desacuerdo entre la agrupación percibida por el analizador y el lector humano. Ocasionalmente, los programadores de C encontrarán un fragmento de código como este:

```
if (x <= y)
        x++;
        y--;
z++;
```

Solo se ejecuta la instrucción `x ++` si la condición es verdadera, pero la indentación lo lleva a creer lo contrario. Incluso los programadores experimentados de C a veces lo miran durante mucho tiempo preguntándose por qué `y` se está disminuyendo incluso para `x > y`.

Debido a que no hay corchetes de inicio/fin, Python es mucho menos propenso a conflictos de estilo de codificación. En C hay muchas formas diferentes de colocar llaves. Después de acostumbrarse a leer y escribir código usando un estilo en particular, es normal sentirse algo incómodo al leer (o tener que escribir) en uno diferente.

Muchos estilos de codificación colocan corchetes de inicio / fin en una línea por sí mismos. Esto hace que los programas sean considerablemente más largos y desperdicia un valioso espacio en la pantalla, lo que dificulta obtener una buena visión general de un programa. Idealmente, una función debería caber en una pantalla (por ejemplo, 20-30 líneas). 20 líneas de Python pueden hacer mucho más trabajo que 20 líneas de C. Esto no se debe únicamente a la falta de corchetes de inicio/fin – la falta de declaraciones y los tipos de datos de alto nivel también son responsables – sino también la indentación basada en la sintaxis ciertamente ayuda.

¿Por qué obtengo resultados extraños con operaciones aritméticas simples?

Ver la siguiente pregunta.

¿Por qué los cálculos de punto flotante son tan inexactos?

Los usuarios a menudo se sorprenden por resultados como este:

```
>>> 1.2 - 1.0
```

```
0.19999999999999996
```

y creo que es un error en Python. No es. Esto tiene poco que ver con Python, y mucho más con la forma en que la plataforma subyacente maneja los números de punto flotante.

El tipo `float` en CPython usa una C `double` para el almacenamiento. Un valor del objeto **`float`** se almacena en coma flotante binaria con una precisión fija (típicamente 53 bits) y Python usa operaciones C, que a su vez dependen de la implementación de hardware en el procesador, para realizar operaciones de coma flotante. Esto significa que, en lo que respecta a las operaciones de punto flotante, Python se comporta como muchos lenguajes populares, incluidos C y Java.

Many numbers that can be written easily in decimal notation cannot be expressed exactly in binary floating point. For example, after:

```
>>> x = 1.2
```

el valor almacenado para `x` es una aproximación (muy buena) al valor decimal `1.2`, pero no es exactamente igual a él. En una máquina típica, el valor real almacenado es:

```
1.0011001100110011001100110011001100110011001100110011
```

que es exactamente:

```
1.1999999999999999555910790149937383830547332763671875
```

La precisión típica de 53 bits proporciona flotantes Python con 15–16 dígitos decimales de precisión.

For a fuller explanation, please see the floating-point arithmetic chapter in the Python tutorial.

¿Por qué las cadenas de caracteres de Python son inmutables?

Hay varias ventajas.

Una es el rendimiento: saber que una cadena es inmutable significa que podemos asignarle espacio en el momento de la creación, y los requisitos de almacenamiento son fijos e inmutables. Esta es también una de las razones para la distinción entre tuplas y listas.

Otra ventaja es que las cadenas en Python se consideran tan «elementales» como los números. Ninguna cantidad de actividad cambiará el valor 8 a otra cosa, y en Python, ninguna cantidad de actividad cambiará la cadena «ocho» a otra cosa.

¿Por qué debe usarse “self” explícitamente en las definiciones y llamadas de métodos?

La idea fue tomada de Modula-3. Resulta ser muy útil, por una variedad de razones.

Primero, es más obvio que está utilizando un método o atributo de instancia en lugar de una variable local. Leer `self.x` o `self.meth()` deja absolutamente claro que se usa una variable de instancia o método incluso si no conoce la definición de clase de memoria. En C++, puede darse cuenta de la falta de una declaración de variable local (suponiendo que los globales son raros o fácilmente reconocibles) – pero en Python, no hay declaraciones de variables locales, por lo que debería buscar la definición de clase para estar seguro. Algunos estándares de codificación de C++ y Java requieren que los atributos de instancia tengan un prefijo `m_`, porque el ser explícito también es útil en esos lenguajes.

En segundo lugar, significa que no es necesaria una sintaxis especial si desea hacer referencia explícita o llamar al método desde una clase en particular. En C++, si desea usar un método de una clase base que se anula en una clase derivada, debe usar el operador `::` – en Python puede escribir `baseclass.methodname(self, <argument list>)`. Esto es particularmente útil para métodos **`__init__()`**, y en general en los casos en que un método de clase derivada quiere extender el método de clase base del mismo nombre y, por lo tanto, tiene que llamar al método de clase base de alguna manera.

Finalmente, para las variables de instancia se resuelve un problema sintáctico con la asignación: dado que las variables locales en Python son (¡por definición!) Aquellas variables a las que se asigna un valor en un cuerpo de función (y que no se declaran explícitamente como globales), tiene que haber una forma de decirle al intérprete que una asignación estaba destinada a asignar a una variable de instancia en lugar de a una variable local, y que preferiblemente debería ser sintáctica (por razones de eficiencia). C ++ hace esto a través de declaraciones, pero Python no tiene declaraciones y sería una pena tener que presentarlas solo para este propósito. Usar el `self.var` explícito resuelve esto muy bien. Del mismo modo, para usar variables de instancia, tener que escribir `self.var` significa que las referencias a nombres no calificados dentro de un método no tienen que buscar en los directorios de la instancia. Para decirlo de otra manera, las variables locales y las variables de instancia viven en dos espacios de nombres diferentes, y debe decirle a Python qué espacio de nombres usar.

¿Por qué no puedo usar una tarea en una expresión?

¡A partir de Python 3.8, se puede!

Asignación de expresiones usando el operador walrus `:=` asigna una variable en una expresión:

```
while chunk := fp.read(200):
   print(chunk)
```

Ver **PEP 572** [https://peps.python.org/pep-0572/] para más información.

¿Por qué Python usa métodos para alguna funcionalidad (por ejemplo, list.index()) pero funciones para otra (por ejemplo,

len(list))?

Como dijo Guido:

> (a) For some operations, prefix notation just reads better than postfix – prefix (and infix!) operations have a long tradition in mathematics which likes notations where the visuals help the mathematician thinking about a problem. Compare the easy with which we rewrite a formula like x*(a+b) into x*a + x*b to the clumsiness of doing the same thing using a raw OO notation.
>
> (b) When I read code that says len(x) I *know* that it is asking for the length of something. This tells me two things: the result is an integer, and the argument is some kind of container. To the contrary, when I read x.len(), I have to already know that x is some kind of container implementing an interface or inheriting from a class that has a standard len(). Witness the confusion we occasionally have when a class that is not implementing a mapping has a get() or keys() method, or something that isn't a file has a write() method.
>
> —https://mail.python.org/pipermail/python-3000/2006-November/004643.html

¿Por qué join() es un método de cadena de caracteres en lugar de un método de lista o tupla?

Las cadenas de caracteres se volvieron mucho más parecidas a otros tipos estándar a partir de Python 1.6, cuando se agregaron métodos que brindan la misma funcionalidad que siempre ha estado disponible utilizando las funciones del módulo de cadenas. La mayoría de estos nuevos métodos han sido ampliamente aceptados, pero el que parece hacer que algunos programadores se sientan incómodos es:

```
", ".join(['1', '2', '4', '8', '16'])
```

que da el resultado:

```
"1, 2, 4, 8, 16"
```

Hay dos argumentos comunes en contra de este uso.

El primero corre a lo largo de las líneas de: «Se ve realmente feo el uso de un método de un literal de cadena (constante de cadena)», a lo que la respuesta es que sí, pero un literal de cadena es solo un valor fijo. Si se permiten los métodos en nombres vinculados a cadenas, no hay razón lógica para que no estén disponibles en literales.

La segunda objeción generalmente se presenta como: «Realmente estoy diciéndole a una secuencia que una a sus miembros junto con una constante de cadena». Lamentablemente, no lo estas haciendo. Por alguna razón, parece ser mucho menos difícil tener `split()` como método de cadena, ya que en ese caso es fácil ver que:

```
"1, 2, 4, 8, 16".split(", ")
```

es una instrucción a un literal de cadena para retornar las subcadenas de caracteres delimitadas por el separador dado (o, por defecto, ejecuciones arbitrarias de espacio en blanco).

`join()` es un método de cadena de caracteres porque al usarlo le está diciendo a la cadena de separación que itere sobre una secuencia de cadenas y se inserte entre elementos adyacentes. Este método se puede usar con cualquier argumento que obedezca las reglas para los objetos de secuencia, incluidas las clases nuevas que pueda definir usted mismo. Existen métodos similares para bytes y objetos bytearray.

¿Qué tan rápido van las excepciones?

Un bloque `try`/`except` es extremadamente eficiente si no son lanzada excepciones. En realidad, capturar una excepción es costoso. En versiones de Python anteriores a la 2.0, era común usar este modismo:

```
try:
    value = mydict[key]
except KeyError:
    mydict[key] = getvalue(key)
    value = mydict[key]
```

Esto solo tenía sentido cuando esperaba que el dict tuviera la clave casi todo el tiempo. Si ese no fuera el caso, lo codificó así:

```
if key in mydict:
    value = mydict[key]
else:
    value = mydict[key] = getvalue(key)
```

Para este caso específico, también podría usar `value = dict.setdefault(key, getvalue(key))`, pero solo si la llamada `getvalue()` es lo suficientemente barata porque se evalúa en todos los casos.

¿Por qué no hay un *switch* o una declaración *case* en Python?

In general, structured switch statements execute one block of code when an expression has a particular value or set of values. Since Python 3.10 one can easily match literal values, or constants within a namespace, with a `match ... case` statement. An older alternative is a sequence of `if... elif... elif... else`.

Para los casos en los que necesita elegir entre una gran cantidad de posibilidades, puede crear un diccionario que asigne valores de casos a funciones para llamar. Por ejemplo:

```
functions = {'a': function_1,
             'b': function_2,
             'c': self.method_1}
```

```
func = functions[value]
func()
```

Para invocar métodos en objetos, puede simplificar aún más utilizando **`getattr()`** incorporado para recuperar métodos con un nombre particular:

```
class MyVisitor:
    def visit_a(self):
        ...

    def dispatch(self, value):
        method_name = 'visit_' + str(value)
        method = getattr(self, method_name)
        method()
```

Se sugiere que utilice un prefijo para los nombres de los métodos, como `visit_` en este ejemplo. Sin dicho prefijo, si los valores provienen de una fuente no confiable, un atacante podría invocar cualquier método en su objeto.

Imitating switch with fallthrough, as with C's switch-case-default, is possible, much harder, and less needed.

¿No puede emular hilos en el intérprete en lugar de confiar en una implementación de hilos específica del sistema operativo?

Respuesta 1: Desafortunadamente, el intérprete empuja al menos un marco de pila C para cada marco de pila de Python. Además, las extensiones pueden volver a llamar a Python en momentos casi aleatorios. Por lo tanto, una implementación completa de subprocesos requiere soporte de subprocesos para C.

Respuesta 2: Afortunadamente, existe Python sin pila [https://github.com/stackless-dev/stackless/wiki], que tiene un bucle de intérprete completamente rediseñado que evita la pila C.

¿Por qué las expresiones lambda no pueden contener sentencias?

Las expresiones lambda de Python no pueden contener declaraciones porque el marco sintáctico de Python no puede manejar declaraciones anidadas dentro de expresiones. Sin embargo, en Python, este no es un problema grave. A diferencia de las formas lambda en otros lenguajes, donde agregan funcionalidad, las lambdas de Python son solo una notación abreviada si eres demasiado vago para definir una función.

Las funciones ya son objetos de primera clase en Python, y pueden declararse en un ámbito local. Por lo tanto, la única ventaja de usar una lambda en lugar de una función definida localmente es que no es necesario inventar un nombre para la función – ¡pero eso es sólo una variable local a la que se asigna el objeto función (que es exactamente el mismo tipo de objeto que produce una expresión lambda)!

¿Se puede compilar Python en código máquina, C o algún otro lenguaje?

Cython [https://cython.org/] compiles a modified version of Python with optional annotations into C extensions. Nuitka [https://nuitka.net/] is an up-and-coming compiler of Python into C++ code, aiming to support the full Python language.

¿Cómo gestiona Python la memoria?

Los detalles de la administración de memoria de Python dependen de la implementación. La implementación estándar de Python, CPython, utiliza el recuento de referencias para detectar objetos inaccesibles, y otro mecanismo para recopilar ciclos de referencia, ejecutando periódicamente un algoritmo de detección de ciclos que busca ciclos inaccesibles y elimina los objetos involucrados. El módulo `gc` proporciona funciones para realizar una recolección de basura, obtener estadísticas de depuración y ajustar los parámetros del recolector.

Other implementations (such as Jython [https://www.jython.org] or PyPy [https://pypy.org]), however, can rely on a different mechanism such as a full-blown garbage collector. This difference can cause some subtle porting problems if your Python code depends on the behavior of the reference counting implementation.

En algunas implementaciones de Python, el siguiente código (que está bien en CPython) probablemente se quedará sin descriptores de archivo:

```
for file in very_long_list_of_files:
    f = open(file)
    c = f.read(1)
```

De hecho, utilizando el esquema de conteo de referencias y destructor de CPython, cada nueva asignación a `f` cierra el archivo anterior. Sin embargo, con un GC tradicional, esos objetos de archivo solo se recopilarán (y cerrarán) a intervalos variables y posiblemente largos.

Si desea escribir código que funcione con cualquier implementación de Python, debe cerrar explícitamente el archivo o utilizar una declaración `with`; esto funcionará independientemente del esquema de administración de memoria:

```
for file in very_long_list_of_files:
    with open(file) as f:
        c = f.read(1)
```

¿Por qué CPython no utiliza un esquema de recolección de basura más tradicional?

Por un lado, esta no es una característica estándar de C y, por lo tanto, no es portátil. (Sí, sabemos acerca de la biblioteca Boehm GC. Tiene fragmentos de código de ensamblador para *la mayoría* de las plataformas comunes, no para todas ellas, y aunque es principalmente transparente, no es completamente transparente; se requieren parches para obtener Python para trabajar con eso)

El GC tradicional también se convierte en un problema cuando Python está integrado en otras aplicaciones. Mientras que en un Python independiente está bien reemplazar el estándar `malloc()` y `free()` con versiones proporcionadas por la biblioteca GC, una aplicación que incruste Python puede querer tener su *propio* sustituto de `malloc()` y `free()`, y puede no querer el de Python. En este momento, CPython funciona con todo lo que implementa `malloc()` y `free()` correctamente.

¿Por qué no se libera toda la memoria cuando sale CPython?

Los objetos a los que se hace referencia desde los espacios de nombres globales de los módulos de Python no siempre se desasignan cuando Python sale. Esto puede suceder si hay referencias circulares. También hay ciertos bits de memoria asignados por la biblioteca de C que son imposibles de liberar (por ejemplo, una herramienta como Purify se quejará de estos). Python es, sin embargo, agresivo sobre la limpieza de la memoria al salir e intenta destruir cada objeto.

Si desea forzar a Python a eliminar ciertas cosas en la desasignación, use el módulo **`atexit`** para ejecutar una función que obligará a esas eliminaciones.

¿Por qué hay tipos de datos separados de tuplas y listas?

Las listas y las tuplas, si bien son similares en muchos aspectos, generalmente se usan de maneras fundamentalmente diferentes. Las tuplas pueden considerarse similares a los `records` de Pascal o a los `structs` de C; son pequeñas colecciones de datos relacionados que pueden ser de diferentes tipos que funcionan como un grupo. Por ejemplo, una coordenada cartesiana se representa adecuadamente como una tupla de dos o tres números.

Las listas, por otro lado, son más como matrices en otros lenguajes. Tienden a contener un número variable de objetos, todos los cuales tienen el mismo tipo y que se operan uno por uno. Por ejemplo, `os.listdir('.')` retorna una lista de cadenas de caracteres que representan los archivos en el directorio actual. Las funciones que operan en esta salida generalmente no se romperían si agregara otro archivo o dos al directorio.

Las tuplas son inmutables, lo que significa que una vez que se ha creado una tupla, no puede reemplazar ninguno de sus elementos con un nuevo valor. Las listas son mutables, lo que significa que siempre puede cambiar los elementos de una lista. Solo los elementos inmutables se pueden usar como claves de diccionario y, por lo tanto, solo las tuplas y no las listas se pueden usar como claves.

¿Cómo se implementan las listas en Python?

Las listas de CPython son realmente matrices de longitud variable, no listas enlazadas al estilo Lisp. La implementación utiliza una matriz contigua de referencias a otros objetos y mantiene un puntero a esta matriz y la longitud de la matriz en una estructura de encabezado de lista.

Esto hace que indexar una lista `a[i]` una operación cuyo costo es independiente del tamaño de la lista o del valor del índice.

Cuando se añaden o insertan elementos, la matriz de referencias cambia de tamaño. Se aplica cierta inteligencia para mejorar el rendimiento de la adición de elementos repetidamente; cuando la matriz debe crecer, se asigna un espacio extra para que las próximas veces no requieran un cambio de tamaño real.

¿Cómo se implementan los diccionarios en CPython?

Los diccionarios de CPython se implementan como tablas hash redimensionables. En comparación con los árboles B (*B-trees*), esto proporciona un mejor rendimiento para la búsqueda (la operación más común con diferencia) en la mayoría de las circunstancias, y la implementación es más simple.

Dictionaries work by computing a hash code for each key stored in the dictionary using the **`hash()`** built-in function. The hash code varies widely depending on the key and a per-process seed; for example, `'Python'` could hash to `-539294296` while `'python'`, a string that differs by a single bit, could hash to `1142331976`. The hash code is then used to calculate a location in an internal array where the value will be stored. Assuming that you're storing keys that all have different hash values, this means that dictionaries take constant time – $O(1)$, in Big-O notation – to retrieve a key.

¿Por qué las claves del diccionario deben ser inmutables?

La implementación de la tabla hash de los diccionarios utiliza un valor hash calculado a partir del valor clave para encontrar la clave. Si la clave fuera un objeto mutable, su valor podría cambiar y, por lo tanto, su hash también podría cambiar. Pero dado que quien cambie el objeto clave no puede decir que se estaba utilizando como clave de diccionario, no puede mover la entrada en el diccionario. Luego, cuando intente buscar el mismo objeto en el diccionario, no se encontrará porque su valor hash es diferente. Si trató de buscar el valor anterior, tampoco lo encontraría, porque el valor del objeto que se encuentra en ese hash bin sería diferente.

Si desea un diccionario indexado con una lista, simplemente convierta la lista a una tupla primero; La función `tuple(L)` crea una tupla con las mismas entradas que la lista `L`. Las tuplas son inmutables y, por lo tanto, pueden usarse como claves de diccionario.

Algunas soluciones inaceptables que se han propuesto:

- Listas de hash por su dirección (ID de objeto). Esto no funciona porque si construye una nueva lista con el mismo valor, no se encontrará; por ejemplo:

```
mydict = {[1, 2]: '12'}
print(mydict[[1, 2]])
```

generaría una excepción **`KeyError`** porque la identificación del `[1, 2]` usado en la segunda línea difiere de la de la primera línea. En otras palabras, las claves del diccionario deben compararse usando `==`, no usando **`is`**.

- Hacer una copia cuando use una lista como clave. Esto no funciona porque la lista, al ser un objeto mutable, podría contener una referencia a sí misma, y luego el código de copia se ejecutaría en un bucle infinito.

- Permitir listas como claves pero decirle al usuario que no las modifique. Esto permitiría una clase de errores difíciles de rastrear en los programas cuando olvidó o modificó una lista por accidente. También invalida una invariante importante de diccionarios: cada valor en `d.keys()` se puede usar como una clave del diccionario.

- Marcar las listas como de solo lectura una vez que se usan como clave de diccionario. El problema es que no solo el objeto de nivel superior puede cambiar su valor; podría usar una tupla que contiene una lista como clave. Ingresar cualquier cosa como clave en un diccionario requeriría marcar todos los objetos accesibles desde allí como de solo lectura – y nuevamente, los objetos autoreferenciados podrían causar un bucle infinito.

Hay un truco para evitar esto si lo necesita, pero úselo bajo su propio riesgo: Puede envolver una estructura mutable dentro de una instancia de clase que tenga tanto un método **`__eq__()`** y un **`__hash__()`**. Luego debe asegurarse de que el valor hash para todos los objetos de contenedor que residen en un diccionario (u otra estructura basada en hash) permanezca fijo mientras el objeto está en el diccionario (u otra estructura).

```
class ListWrapper:
    def __init__(self, the_list):
        self.the_list = the_list

    def __eq__(self, other):
        return self.the_list == other.the_list

    def __hash__(self):
        l = self.the_list
        result = 98767 - len(l)*555
        for i, el in enumerate(l):
            try:
                result = result + (hash(el) % 9999999) *
            except Exception:
                result = (result % 7777777) + i * 333
        return result
```

Tenga en cuenta que el cálculo de hash se complica por la posibilidad de que algunos miembros de la lista sean inquebrantables y también por la posibilidad de desbordamiento aritmético.

Además, siempre debe darse el caso de que si `o1 == o2` (es decir, `o1.__eq__(o2) is True`), entonces `hash(o1) == hash(o2)` (es decir, `o1.__hash__() == o2.__hash__()`), independientemente de si el objeto está en un diccionario o no. Si no cumple con estas restricciones, los diccionarios y otras estructuras basadas en hash se comportarán mal.

En el caso de **`ListWrapper`**, siempre que el objeto contenedor esté en un diccionario, la lista contenida no debe cambiar para evitar anomalías. No haga esto a menos que esté preparado para pensar detenidamente sobre los requisitos y las consecuencias de no cumplirlos correctamente. Considérese advertido.

¿Por qué list.sort() no retorna la lista ordenada?

En situaciones donde el rendimiento es importante, hacer una copia de la lista solo para ordenarlo sería un desperdicio. Por lo tanto, **`list.sort()`** ordena la lista en su lugar. Para recordarle ese hecho, no retorna la lista ordenada. De esta manera, no se dejará engañar por sobreescribir accidentalmente una lista cuando necesite una copia ordenada, pero también deberá mantener la versión sin ordenar.

Si desea retornar una nueva lista, use la función incorporada **`sorted()`** en su lugar. Esta función crea una nueva lista a partir de un iterativo proporcionado, la ordena y la retorna. Por ejemplo, a continuación se explica cómo iterar sobre las teclas de un diccionario en orden ordenado:

```
for key in sorted(mydict):
    ...  # do whatever with mydict[key]...
```

¿Cómo se especifica y aplica una especificación de interfaz en Python?

Una especificación de interfaz para un módulo proporcionada por lenguajes como C++ y Java describe los prototipos para los métodos y funciones del módulo. Muchos sienten que la aplicación en tiempo de compilación de las especificaciones de la interfaz ayuda en la construcción de grandes programas.

Python 2.6 agrega un módulo **`abc`** que le permite definir clases base abstractas (ABC). Luego puede usar **`isinstance()`** y **`issubclass()`** para verificar si una instancia o una clase implementa un ABC en particular. El módulo **`collections.abc`** define un conjunto de ABC útiles como **`Iterable`**, **`Container`** y **`MutableMapping`**.

Para Python, muchas de las ventajas de las especificaciones de interfaz se pueden obtener mediante una disciplina de prueba adecuada para los componentes.

Un buen conjunto de pruebas para un módulo puede proporcionar una prueba de regresión y servir como una especificación de interfaz de módulo y un conjunto de ejemplos. Muchos módulos de Python se pueden ejecutar como un script para proporcionar una simple «autocomprobación». Incluso los módulos que usan interfaces externas complejas a menudo se pueden probar de forma aislada utilizando emulaciones triviales de «stub» de la interfaz externa. Los módulos `doctest` y `unittest` o marcos de prueba de terceros se pueden utilizar para construir conjuntos de pruebas exhaustivas que ejercitan cada línea de código en un módulo.

An appropriate testing discipline can help build large complex applications in Python as well as having interface specifications would. In fact, it can be better because an interface specification cannot test certain properties of a program. For example, the `list.append()` method is expected to add new elements to the end of some internal list; an interface specification cannot test that your `list.append()` implementation will actually do this correctly, but it's trivial to check this property in a test suite.

Escribir conjuntos de pruebas es muy útil, y es posible que desee diseñar su código con miras a que sea fácilmente probado. Una técnica cada vez más popular, el desarrollo dirigido por pruebas, requiere escribir partes del conjunto de pruebas primero, antes de escribir el código real. Por supuesto, Python te permite ser descuidado y no escribir casos de prueba.

¿Por qué no hay goto?

En la década de 1970, la gente se dio cuenta de que el goto irrestricto podía generar un código «espagueti» desordenado que era difícil de entender y revisar. En un lenguaje de alto nivel, también es innecesario siempre que haya formas de bifurcar (en Python, con sentencias `if` y expresiones `or`, `and` e `if`/`else`) y bucle (con sentencia `while` y `for`, que posiblemente contengan `continue` y `break`).

También se pueden utilizar excepciones para proporcionar un «goto estructurado» que funcione incluso a través de llamadas a funciones. Muchos creen que las excepciones pueden emular convenientemente todos los usos razonables de las construcciones `go` o `goto` de C, Fortran y otros lenguajes. Por ejemplo:

```
class label(Exception): pass  # declare a label

try:
    ...
    if condition: raise label()  # goto label
    ...
except label:  # where to goto
    pass
...
```

Esto no le permite saltar a la mitad de un bucle, pero eso es generalmente considerado un abuso de `goto` de todos modos. Utilizar con moderación.

¿Por qué las cadenas de caracteres sin formato (r-strings) no pueden terminar con una barra diagonal inversa?

Más precisamente, no pueden terminar con un número impar de barras invertidas: la barra invertida no emparejada al final escapa el carácter de comillas de cierre, dejando una cadena sin terminar.

Las cadenas de caracteres sin formato se diseñaron para facilitar la creación de entradas para procesadores (principalmente motores de expresión regular) que desean realizar su propio procesamiento de escape de barra invertida. Tales procesadores consideran que una barra invertida sin par es un error de todos modos, por lo que las cadenas de caracteres sin procesar no lo permiten. A cambio, le permiten pasar el carácter de comillas de cadena escapándolo con una barra invertida. Estas reglas funcionan bien cuando las cadenas de caracteres r (*r-strings*) se usan para el propósito previsto.

Si está intentando construir nombres de ruta de Windows, tenga en cuenta que todas las llamadas al sistema de Windows también aceptan barras diagonales:

```
f = open("/mydir/file.txt")  # works fine!
```

Si está tratando de construir una ruta para un comando de DOS, intente por ejemplo uno de los siguientes:

```
dir = r"\this\is\my\dos\dir" "\\"
dir = r"\this\is\my\dos\dir\ "[:-1]
dir = "\\this\\is\\my\\dos\\dir\\"
```

¿Por qué Python no tiene una declaración «with» para las asignaciones de atributos?

Python tiene una sentencia **`with`** que envuelve la ejecución de un bloque, llamando al código en la entrada y salida del bloque. Algunos lenguajes tienen una construcción que se ve así:

```
with obj:
    a = 1               # equivalent to obj.a = 1
    total = total + 1   # obj.total = obj.total + 1
```

En Python, tal construcción sería ambigua.

Otros lenguajes, como Object Pascal, Delphi y C + +, utilizan tipos estáticos, por lo que es posible saber, de manera inequívoca, a qué miembro se le está asignando. Este es el punto principal de la escritura estática: el compilador *siempre* conoce el alcance de cada variable en tiempo de compilación.

Python usa tipos dinámicos. Es imposible saber de antemano a qué atributo se hará referencia en tiempo de ejecución. Los atributos de los miembros pueden agregarse o eliminarse de los objetos sobre la marcha. Esto hace que sea imposible saber, a partir de una simple lectura, a qué atributo se hace referencia: ¿uno local, uno global o un atributo miembro?

Por ejemplo, tome el siguiente fragmento incompleto:

```
def foo(a):
    with a:
        print(x)
```

El fragmento supone que `a` debe tener un atributo miembro llamado `x`. Sin embargo, no hay nada en Python que le diga esto al intérprete. ¿Qué debería suceder si `a` es, digamos, un número entero? Si hay una variable global llamada `x`, ¿se usará dentro del bloque **`with`**? Como puede ver, la naturaleza dinámica de Python hace que tales elecciones sean mucho más difíciles.

El principal beneficio de `with` y características de lenguaje similares (reducción del volumen del código) puede, sin embargo, lograr fácilmente en Python mediante la asignación. En vez de:

```
function(args).mydict[index][index].a = 21
function(args).mydict[index][index].b = 42
function(args).mydict[index][index].c = 63
```

escribe esto:

```
ref = function(args).mydict[index][index]
ref.a = 21
ref.b = 42
ref.c = 63
```

Esto también tiene el efecto secundario de aumentar la velocidad de ejecución porque los enlaces de nombres se resuelven en tiempo de ejecución en Python, y la segunda versión solo necesita realizar la resolución una vez.

Otras propuestas similares que introducían sintaxis para reducir aún más el volumen del código, como el uso de un “punto inicial”, han sido rechazadas en favor de la claridad (véase https://mail.python.org/pipermail/python-ideas/2016-May/040070.html).

¿Por qué los generadores no admiten la declaración with?

Por razones técnicas, un generador utilizado directamente como gestor de contexto no funcionaría correctamente. Cuando, como es más común, un generador se utiliza como un iterador ejecutado hasta su finalización, no es necesario cerrar. Cuando lo esté, envuélvalo como `contextlib.closing(generator)` en la sentencia `with`.

¿Por qué se requieren dos puntos para las declaraciones if/while/def/class?

Los dos puntos se requieren principalmente para mejorar la legibilidad (uno de los resultados del lenguaje ABC experimental). Considera esto:

```
if a == b
    print(a)
```

versus

```
if a == b:
    print(a)
```

Observe cómo el segundo es un poco más fácil de leer. Observe más a fondo cómo los dos puntos establecen el ejemplo en esta respuesta de preguntas frecuentes; Es un uso estándar en inglés.

Otra razón menor es que los dos puntos facilitan a los editores con resaltado de sintaxis; pueden buscar dos puntos para decidir cuándo se debe aumentar la indentación en lugar de tener que hacer un análisis más elaborado del texto del programa.

¿Por qué Python permite comas al final de las listas y tuplas?

Python le permite agregar una coma final al final de las listas, tuplas y diccionarios:

```
[1, 2, 3,]
('a', 'b', 'c',)
d = {
    "A": [1, 5],
    "B": [6, 7],  # last trailing comma is optional but
}
```

Hay varias razones para permitir esto.

Cuando tiene un valor literal para una lista, tupla o diccionario distribuido en varias líneas, es más fácil agregar más elementos porque no tiene que recordar agregar una coma a la línea anterior. Las líneas también se pueden reordenar sin crear un error de sintaxis.

La omisión accidental de la coma puede ocasionar errores difíciles de diagnosticar. Por ejemplo:

```
x = [
  "fee",
  "fie"
  "foo",
  "fum"
]
```

Parece que esta lista tiene cuatro elementos, pero en realidad contiene tres: «fee», «fiefoo» y «fum». Agregar siempre la coma evita esta fuente de error.

Permitir la coma final también puede facilitar la generación de código programático.

Preguntas frecuentes sobre bibliotecas y extensiones

Contenido

- Preguntas frecuentes sobre bibliotecas y extensiones
  - Cuestiones generales sobre bibliotecas
    - ¿Cómo encuentro un módulo o aplicación para ejecutar la tarea X?
    - ¿Dónde está el fichero fuente *math.py* (*socket.py*, *regex.py*, etc.)?
    - ¿Cómo hago ejecutable un script Python en Unix?
    - ¿Hay un paquete curses/termcap para Python?
    - ¿Hay un equivalente en Python al onexit() de C?
    - ¿Por qué no funcionan mis manejadores de señales?
  - Tareas comunes
    - ¿Cómo pruebo un programa o un componente Python?
    - ¿Cómo creo documentación a partir de los docstrings?
    - ¿Cómo consigo presionar una única tecla cada vez?
  - Hilos
    - ¿Cómo programo usando hilos?
    - Ninguno de mis hilos parece funcionar: ¿por qué?
    - ¿Cómo puedo dividir trabajo entre un grupo de hilos?
    - ¿Qué tipos de mutación de valores globales son *thread-safe*?
    - ¿Podemos deshacernos del *Global Interpreter Lock*?
  - Entrada y salida
    - ¿Cómo borro un fichero? (Y otras preguntas sobre ficheros…)

    - ¿Cómo copio un fichero?
    - ¿Cómo leo (o escribo) datos binarios?
    - No consigo usar os.read() en un *pipe* creado con os.popen(); ¿por qué?
    - ¿Cómo accedo al puerto serial (RS232)?
    - ¿Por qué al cerrar sys.stdout (stdin, stderr) realmente no se cierran?
  - Programación de Redes/Internet
    - ¿Qué herramientas de Python existen para WWW?
    - ¿Qué modulo debería usar para generación de HTML?
    - ¿Cómo envío correo desde un script Python?
    - ¿Cómo evito el bloqueo en el método *connect()* de un *socket*?
  - Bases de datos
    - ¿Hay paquetes para interfaces a bases de datos en Python?
    - ¿Cómo implementar objetos persistentes en Python?
  - Matemáticas y numérica
    - ¿Cómo genero números aleatorios en Python?

Cuestiones generales sobre bibliotecas

¿Cómo encuentro un módulo o aplicación para ejecutar la tarea X?

Vea la referencia de bibliotecas para comprobar si existe un módulo relevante en la biblioteca estándar. (Eventualmente aprenderá lo que hay en la biblioteca estándar y será capaz de saltarse este paso.)

Para paquetes de terceros, busque Python Package Index [https://pypi.org] o pruebe Google [https://www.google.com] u otro motor de búsqueda web. La búsqueda de «Python» más una palabra clave o dos para su tema de interés generalmente encontrará algo útil.

¿Dónde está el fichero fuente *math.py* (*socket.py*, *regex.py*, etc.)?

Si no puede encontrar un fichero fuente para un módulo, puede ser un módulo incorporado o cargado dinámicamente implementado en C, C++ u otro lenguaje compilado. En este caso puede no disponer del fichero fuente o puede ser algo como `mathmodule.c`, en algún lugar de un directorio fuente C (fuera del Python *Path*).

Hay (al menos) tres tipos de módulos en Python:

1. módulos escritos en Python (.py);
2. módulos escritos en C y cargados dinámicamente (*.dll, .pyd, .so, .sl*, etc.);
3. módulos escritos en C y enlazados con el intérprete; para obtener una lista de estos, escriba:

   ```
   import sys
   print(sys.builtin_module_names)
   ```

¿Cómo hago ejecutable un script Python en Unix?

Necesita hacer dos cosas: el modo del fichero del script debe ser ejecutable y la primera línea debe comenzar con `#!` seguido de la ruta al intérprete de Python.

Lo primero se hace ejecutando `chmod +x scriptfile` o bien `chmod 755 scriptfile`.

Lo segundo se puede hacer de distintas maneras. La manera más directa es escribir

```
#!/usr/local/bin/python
```

en la primera línea de su fichero, usando la ruta donde está instalado el intérprete de Python en su plataforma.

Si quiere que el script sea independiente de donde se ubique el intérprete de Python, puede usar el programa **env**. Casi todas las variantes de Unix soportan lo siguiente, asumiendo que el intérprete de Python está en un directorio del **`PATH`** de usuario:

```
#!/usr/bin/env python
```

No haga esto para scripts CGI. La variable **`PATH`** para scripts CGI es mínima, así que necesita usar la ruta real absoluta al intérprete.

Ocasionalmente, un entorno de usuario está tan lleno que el programa **/usr/bin/env** falla; o bien no existe el programa env. En ese caso, puede intentar el siguiente truco (gracias a Alex Rezinsky):

```
#! /bin/sh
""":"
exec python $0 ${1+"$@"}
"""
```

Una pequeña desventaja es que esto define el *_doc_* del script. Sin embargo, puede arreglarlo añadiendo

```
__doc__ = """...Whatever..."""
```

¿Hay un paquete curses/termcap para Python?

Para variantes Unix: La distribución estándar de Python viene con un módulo curses en el subdirectorio Modules [https://github.com/python/cpython/tree/3.14/Modules], aunque no está compilado por defecto. (Nótese que esto no está disponible en la distribución Windows — no hay módulo curses para Windows.)

El módulo **`curses`** soporta características básicas de cursores así como muchas funciones adicionales de ncurses y cursores SYSV como color, soporte para conjuntos de caracteres alternativos, pads, y soporte para ratón. Esto significa que el módulo no es compatible con sistemas operativos que sólo tienen cursores BSD, pero no parece que ningún sistema operativo actualmente mantenido caiga dentro de esta categoría.

¿Hay un equivalente en Python al onexit() de C?

The **`atexit`** module provides a register function that is similar to C's **`onexit()`**.

¿Por qué no funcionan mis manejadores de señales?

El problema más común es que el manejador de señales esté declarado con la lista incorrecta de argumentos. Se llama como

```
handler(signum, frame)
```

así que debería declararse con dos argumentos:

```
def handler(signum, frame):
    ...
```

Tareas comunes

¿Cómo pruebo un programa o un componente Python?

Python viene con dos *frameworks* de *testing*. El módulo **`doctest`** encuentra ejemplos en los docstrings para un módulo y los ejecuta, comparando la salida con la salida esperada especificada en la cadena de documentación.

El módulo **`unittest`** es un *framework* de *testing* más agradable y modelado sobre los *frameworks* de *testing* de Java y Smalltalk.

Para hacer más fácil el *testing*, debería usar un buen diseño modular en su programa. Su programa debería tener casi toda la funcionalidad encapsulada en funciones o en métodos de clases — y esto algunas veces tiene el efecto sorprendente y encantador de que su programa funcione más rápido (porque los accesos a las variables locales son más rápidas que los accesos a las variables globales). Además el programa debería evitar depender de la mutación de variables globales, ya que esto dificulta mucho más hacer el *testing*.

La «lógica global principal» de su programa puede ser tan simple como

```
if __name__ == "__main__":
    main_logic()
```

al final del módulo principal de su programa.

Una vez que su programa esté organizado en una colección manejable de funciones y comportamientos de clases, usted debería escribir funciones de comprobación que ejerciten los comportamientos. Se puede asociar un conjunto de pruebas a cada módulo. Esto suena a mucho trabajo, pero gracias a que Python es tan conciso y flexible, se hace sorprendentemente fácil. Puede codificar de manera mucho más agradable y divertida escribiendo funciones de comprobación en paralelo con el «código de producción», ya que esto facilita encontrar antes errores e incluso fallos de diseño.

Los «módulos de soporte» que no tienen la intención de estar en el módulo principal de un programa pueden incluir un auto *test* del módulo.

```
if __name__ == "__main__":
    self_test()
```

Incluso los programas que interactúan con interfaces externas complejas se pueden comprobar cuando las interfaces externas no están disponibles usando interfaces «simuladas» implementadas en Python.

¿Cómo creo documentación a partir de los docstrings?

El módulo `pydoc` puede crear HTML desde los docstrings existentes en su código fuente Python. Una alternativa para crear documentación API estrictamente desde docstrings es epydoc [http://epydoc.sourceforge.net/]. Sphinx [http://sphinx-doc.org] también puede incluir contenido docstring.

¿Cómo consigo presionar una única tecla cada vez?

Para variantes Unix hay varias soluciones. Lo más directo es hacerlo usando cursores, pero curses es un módulo bastante amplio para aprenderlo.

Hilos

¿Cómo programo usando hilos?

Asegúrese de usar el módulo `threading` y no el módulo `_thread`. El módulo `threading` construye abstracciones convenientes sobre las primitivas de bajo nivel proporcionadas por el módulo `_thread`.

Ninguno de mis hilos parece funcionar: ¿por qué?

Tan pronto como el hilo principal termine, se matan todos los hilos. Su hilo principal está corriendo demasiado rápido, sin dar tiempo a los hilos para hacer algún trabajo.

Una solución sencilla es añadir un *sleep* al final del programa que sea suficientemente largo para que todos los hilos terminen:

```
import threading, time

def thread_task(name, n):
    for i in range(n):
        print(name, i)

for i in range(10):
    T = threading.Thread(target=thread_task, args=(str(i
    T.start()

time.sleep(10)  # <----------------------------!
```

Por ahora (en muchas plataformas) los hilos no corren en paralelo, sino que parece que corren secuencialmente, ¡uno a la vez! La razón es que el planificador de hilos del sistema operativo no inicia un nuevo hilo hasta que el hilo anterior está bloqueado.

Una solución sencilla es añadir un pequeño *sleep* al comienzo de la función run:

```
def thread_task(name, n):
    time.sleep(0.001)  # <---------------------!
    for i in range(n):
        print(name, i)
```

```
for i in range(10):
    T = threading.Thread(target=thread_task, args=(str(i
    T.start()

time.sleep(10)
```

En vez de intentar adivinar un valor de retardo adecuado para **`time.sleep()`**, es mejor usar algún tipo de mecanismo de semáforo. Una idea es usar el módulo **`queue`** para crear un objeto cola, permitiendo que cada hilo añada un *token* a la cola cuando termine, y permitiendo al hilo principal leer tantos tokens de la cola como hilos haya.

¿Cómo puedo dividir trabajo entre un grupo de hilos?

La manera más fácil es usar el nuevo módulo **`concurrent.futures`**, especialmente el módulo **`ThreadPoolExecutor`**.

O, si quiere tener un control más preciso sobre el algoritmo de despacho, puede escribir su propia lógica manualmente. Use el módulo **`queue`** para crear una cola que contenga una lista de trabajos. La clase **`Queue`** mantiene una lista de objetos y tiene un método `.put(obj)` que añade elementos a la cola y un método `.get()` que los retorna. Esta clase se encargará de los bloqueos necesarios para asegurar que cada trabajo se reparte exactamente una vez.

Aquí hay un ejemplo trivial:

```
import threading, queue, time

# The worker thread gets jobs off the queue.  When the q
# assumes there will be no more work and exits.
# (Realistically workers will run until terminated.)
def worker():
    print('Running worker')
    time.sleep(0.1)
    while True:
        try:
            arg = q.get(block=False)
        except queue.Empty:
            print('Worker', threading.current_thread(),
            print('queue empty')
            break
        else:
            print('Worker', threading.current_thread(),
            print('running with argument', arg)
            time.sleep(0.5)

# Create queue
q = queue.Queue()

# Start a pool of 5 workers
for i in range(5):
    t = threading.Thread(target=worker, name='worker %i'
    t.start()

# Begin adding work to the queue
for i in range(50):
    q.put(i)

# Give threads time to run
print('Main thread sleeping')
time.sleep(5)
```

Cuando se ejecute, esto producirá la siguiente salida:

```
Running worker
Running worker
Running worker
Running worker
Running worker
Main thread sleeping
Worker <Thread(worker 1, started 130283832797456)> runni
Worker <Thread(worker 2, started 130283824404752)> runni
Worker <Thread(worker 3, started 130283816012048)> runni
Worker <Thread(worker 4, started 130283807619344)> runni
Worker <Thread(worker 5, started 130283799226640)> runni
Worker <Thread(worker 1, started 130283832797456)> runni
...
```

Consulte la documentación del módulo para más detalles; la clase `Queue` proporciona una interfaz llena de características.

¿Qué tipos de mutación de valores globales son *thread-safe*?

Un global interpreter lock (GIL) se usa internamente para asegurar que sólo un hilo corre a la vez en la VM de Python. En general, Python ofrece cambiar entre hilos sólo en instrucciones bytecode; la frecuencia con la que cambia se puede fijar vía `sys.setswitchinterval()`. Cada instrucción bytecode y por lo tanto, toda la implementación de código C alcanzada por cada instrucción, es atómica desde el punto de vista de un programa Python.

En teoría, esto significa que un informe exacto requiere de un conocimiento exacto de la implementación en bytecode de la PVM. En la práctica, esto significa que las operaciones entre variables compartidas de tipos de datos *built-in* (enteros, listas, diccionarios, etc.) que «parecen atómicas» realmente lo son.

Por ejemplo, las siguientes operaciones son todas atómicas (L, L1, L2 son listas, D, D1, D2 son diccionarios, *x*, *y* son objetos, *i*, *j* son enteros):

```
L.append(x)
L1.extend(L2)
x = L[i]
x = L.pop()
L1[i:j] = L2
L.sort()
x = y
x.field = y
D[x] = y
D1.update(D2)
D.keys()
```

Estas no lo son:

```
i = i+1
L.append(L[-1])
L[i] = L[j]
D[x] = D[x] + 1
```

Operations that replace other objects may invoke those other objects” `__del__()` method when their reference count reaches zero, and that can affect things. This is especially true for the mass updates to dictionaries and lists. When in doubt, use a mutex!

¿Podemos deshacernos del *Global Interpreter Lock*?

El global interpreter lock (GIL) se percibe a menudo como un obstáculo en el despliegue de Python sobre máquinas servidoras finales de múltiples procesadores, porque un programa Python multihilo efectivamente sólo usa una CPU, debido a la exigencia de que (casi) todo el código Python sólo puede correr mientras el GIL esté activado.

With the approval of **PEP 703** [https://peps.python.org/pep-0703/] work is now underway to remove the GIL from the CPython implementation of Python. Initially it will be implemented as an optional compiler flag when building the interpreter, and so separate builds will be available with and without the GIL. Long-term, the hope is to settle on a single build, once the performance implications of removing the GIL are fully understood. Python 3.13 is likely to be the first release containing this work, although it may not be completely functional in this release.

The current work to remove the GIL is based on a fork of Python 3.9 with the GIL removed [https://github.com/colesbury/nogil] by Sam Gross. Prior to that, in the days of Python 1.5, Greg Stein actually implemented a comprehensive patch set (the «free threading» patches) that removed the GIL and replaced it with fine-grained locking. Adam Olsen did a similar experiment in his python-safethread [https://code.google.com/archive/p/python-safethread] project. Unfortunately, both of these earlier experiments exhibited a sharp drop in single-thread performance (at least 30% slower), due to the amount of fine-grained locking necessary to compensate for the removal of the GIL. The Python 3.9 fork is the first attempt at removing the GIL with an acceptable performance impact.

The presence of the GIL in current Python releases doesn't mean that you can't make good use of Python on multi-CPU machines! You just have to be creative with dividing the work up between multiple *processes* rather than multiple *threads*. The **`ProcessPoolExecutor`** class in the new **`concurrent.futures`** module provides an easy way of doing so; the **`multiprocessing`** module provides a lower-level API in case you want more control over dispatching of tasks.

El uso sensato de extensiones C también ayudará; si usa una extensión C para ejecutar una tarea que consume mucho tiempo, la extensión puede liberar al GIL mientras el hilo de ejecución esté en el código C y permite a otros hilos hacer trabajo. Algunos módulos de la biblioteca estándar tales como **`zlib`** y **`hashlib`** ya lo hacen.

An alternative approach to reducing the impact of the GIL is to make the GIL a per-interpreter-state lock rather than truly global. This was first implemented in Python 3.12 and is available in the C API. A Python interface to it is expected in Python 3.13. The main limitation to it at the moment is likely to be 3rd party extension modules, since these must be written with multiple interpreters in mind in order to be usable, so many older extension modules will not be usable.

Entrada y salida

¿Cómo borro un fichero? (Y otras preguntas sobre ficheros…)

Use `os.remove(filename)` o `os.unlink(filename)`; para la documentación, vea el módulo **`os`**. Las dos funciones son idénticas; **`unlink()`** es simplemente el nombre de la llamada al sistema UNIX para esta función.

Para borrar un directorio, use **`os.rmdir()`**; use **`os.mkdir()`** para crear uno. `os.makedirs(path)` creará cualquier directorio intermedio que no exista en `path`. `os.removedirs(path)` borrará los directorios intermedios siempre y cuando estén vacíos; si quiere borrar un árbol de directorios completo y sus contenidos, use **`shutil.rmtree()`**.

Para renombrar un fichero, use `os.rename(old_path, new_path)`.

Para truncar un fichero, ábralo usando `f = open(filename, "rb+")`, y use `f.truncate(offset)`; el desplazamiento toma por defecto la posición actual de búsqueda. También existe `os.ftruncate(fd, offset)` para ficheros abiertos con **`os.open()`**, donde *fd* es el descriptor del fichero (un entero pequeño).

El módulo **`shutil`** también contiene distintas funciones para trabajar con ficheros incluyendo **`copyfile()`**, **`copytree()`** y **`rmtree()`**.

¿Cómo copio un fichero?

El módulo **`shutil`** contiene una función **`copyfile()`**. Nótese que en volúmenes de Windows NTFS, no copia los flujos alternativos de datos (ADS) [https://es.wikipedia.org/wiki/Alternate_Data_Streams] ni las bifurcaciones de recursos (resource forks) [https://en.wikipedia.org/wiki/Resource_fork] en los volúmenes macOS HFS +, aunque ahora ambos rara vez son utilizados. Además tampoco copia los permisos ni metadatos de los archivos, pero si se usa **`shutil.copy2()`** en su lugar, se preservará la mayor parte (aunque no todo).

¿Cómo leo (o escribo) datos binarios?

Para leer o escribir formatos binarios de datos complejos, es mejor usar el módulo **`struct`**. Esto le permite tomar una cadena de texto que contiene datos binarios (normalmente números) y convertirla a objetos de Python; y viceversa.

Por ejemplo, el siguiente código lee de un fichero dos enteros de 2-bytes y uno de 4-bytes en formato *big-endian*:

```
import struct

with open(filename, "rb") as f:
    s = f.read(8)
    x, y, z = struct.unpack(">hhl", s)
```

El “>” en la cadena de formato fuerza los datos a *big-endian*; la letra “h” lee un «entero corto» (2 bytes), y “l” lee un «entero largo» (4 bytes) desde la cadena de texto.

Para datos que son más regulares (por ejemplo una lista homogénea de enteros o flotantes), puede también usar el módulo `array`.

Nota

Para leer y escribir datos binarios, es obligatorio abrir el fichero en modo binario (aquí, pasando `"rb"` a `open()`). Si, en cambio, usa `"r"` (por defecto), el fichero se abrirá en modo texto y `f.read()` retornará objetos `str` en vez de objetos `bytes`.

No consigo usar os.read() en un *pipe* creado con os.popen(); ¿por qué?

`os.read()` es una función de bajo nivel que recibe un descriptor de fichero, un entero pequeño representando el fichero abierto. `os.popen()` crea un objeto fichero de alto nivel, el mismo tipo que retorna la función *built-in* `open()`. Así, para leer *n* bytes de un *pipe* *p* creado con `os.popen()`, necesita usar `p.read(n)`.

¿Cómo accedo al puerto serial (RS232)?

Para Win32, OSX, Linux, BSD, Jython, IronPython:

pyserial [https://pypi.org/project/pyserial/]

Para Unix, vea una publicación Usenet de Mitch Chapman:

https://groups.google.com/groups?

selm=34A04430.CF9@ohioee.com

¿Por qué al cerrar sys.stdout (stdin, stderr) realmente no se cierran?

Los objetos de tipo fichero en Python son una capa de abstracción de alto nivel sobre los descriptores de ficheros de bajo nivel de C.

Para la mayoría de objetos de tipo fichero que cree en Python vía la función *built-in* **`open()`**, `f.close()` marca el objeto de tipo fichero Python como ya cerrado desde el punto de vista de Python, y también ordena el cierre del descriptor de fichero subyacente en C. Esto además ocurre automáticamente en el destructor de `f`, cuando `f` se convierte en basura.

Pero *stdin*, *stdout* y *stderr* se tratan de manera especial en Python, debido a un estatus especial que también tienen en C. Ejecutando `sys.stdout.close()` marca el objeto fichero de nivel Python para ser cerrado, pero *no* cierra el descriptor de fichero asociado en C.

Para cerrar el descriptor de fichero subyacente en C para uno de estos tres casos, debería primero asegurarse de que eso es realmente lo que quiere hacer (por ejemplo, puede confundir módulos de extensión intentado hacer *I/O*). Si es así, use **`os.close()`**:

```
os.close(stdin.fileno())
os.close(stdout.fileno())
os.close(stderr.fileno())
```

O puede usar las constantes numéricas 0, 1 y 2, respectivamente.

Programación de Redes/Internet

¿Qué herramientas de Python existen para WWW?

Vea los capítulos titulados Protocolos y soporte de Internet y Manejo de datos de internet en el manual de referencia de bibliotecas. Python tiene muchos módulos que le ayudarán a construir sistemas web del lado del servidor y del lado del cliente.

Paul Boddie mantiene un resumen de los *frameworks* disponibles en https://wiki.python.org/moin/WebProgramming.

¿Qué modulo debería usar para generación de HTML?

Puede encontrar una colección de enlaces útiles en la página wiki de programación web [https://wiki.python.org/moin/WebProgramming].

¿Cómo envío correo desde un script Python?

Use el módulo **`smtplib`** de la biblioteca estándar.

Aquí hay un remitente simple interactivo que lo usa. Este método trabajará en cualquier máquina que soporte un *listener SMTP*.

```
import sys, smtplib

fromaddr = input("From: ")
toaddrs  = input("To: ").split(',')
print("Enter message, end with ^D:")
msg = ''
while True:
    line = sys.stdin.readline()
    if not line:
        break
    msg += line

# The actual mail send
server = smtplib.SMTP('localhost')
server.sendmail(fromaddr, toaddrs, msg)
server.quit()
```

Una alternativa sólo para UNIX es usar *sendmail*. La ubicación del programa *sendmail* varía entre sistemas; algunas veces está en `/usr/lib/sendmail`, otras veces en `/usr/sbin/sendmail`. El manual de *sendmail* le ayudará. Aquí hay un ejemplo de código:

```
import os

SENDMAIL = "/usr/sbin/sendmail"  # sendmail location
```

```
p = os.popen("%s -t -i" % SENDMAIL, "w")
p.write("To: receiver@example.com\n")
p.write("Subject: test\n")
p.write("\n")  # blank line separating headers from body
p.write("Some text\n")
p.write("some more text\n")
sts = p.close()
if sts != 0:
    print("Sendmail exit status", sts)
```

¿Cómo evito el bloqueo en el método *connect()* de un *socket*?

El módulo `select` es mayoritariamente usado para ayudar con entrada/salida de sockets.

To prevent the TCP connect from blocking, you can set the socket to non-blocking mode. Then when you do the `connect()`, you will either connect immediately (unlikely) or get an exception that contains the error number as `.errno`. `errno.EINPROGRESS` indicates that the connection is in progress, but hasn't finished yet. Different OSes will return different values, so you're going to have to check what's returned on your system.

You can use the `connect_ex()` method to avoid creating an exception. It will just return the errno value. To poll, you can call `connect_ex()` again later – `0` or `errno.EISCONN` indicate that you're connected – or you can pass this socket to `select.select()` to check if it's writable.

Nota

The `asyncio` module provides a general purpose single-threaded and concurrent asynchronous library, which can be used for writing non-blocking network code. The third-party Twisted [https://twisted.org/] library is a popular and feature-rich alternative.

Bases de datos

¿Hay paquetes para interfaces a bases de datos en Python?

Sí.

Interfaces a *hashes* basados en disco tales como **`DBM`** y **`GDBM`** están también incluidas en Python estándar. También hay un módulo **`sqlite3`**, que proporciona una base de datos relacional ligera basada en disco.

Está disponible el soporte para la mayoría de bases de datos relacionales. Vea la página wiki de Programación de Bases de datos [https://wiki.python.org/moin/DatabaseProgramming] para más detalles.

¿Cómo implementar objetos persistentes en Python?

El módulo de biblioteca **`pickle`** soluciona esto de una forma muy general (aunque todavía no puede almacenar cosas como ficheros abiertos, sockets o ventanas), y el módulo de biblioteca **`shelve`** usa *pickle* y *(g)dbm* para crear mapeos persistentes que contienen objetos arbitrarios Python.

Matemáticas y numérica

¿Cómo genero números aleatorios en Python?

El módulo estándar **`random`** implementa un generador de números aleatorios. El uso es simple:

```
import random
random.random()
```

This returns a random floating-point number in the range [0, 1).

Hay también muchos otros generadores especializados en este módulo, tales como:

- `randrange(a, b)` selecciona un entero en el rango [a, b).
- `uniform(a, b)` chooses a floating-point number in the range [a, b).
- `normalvariate(mean, sdev)` muestrea una distribución normal (*Gausiana*).

Algunas funciones de alto nivel operan directamente sobre secuencias, tales como:

- `choice(S)` selecciona un elemento aleatorio de una secuencia dada.
- `shuffle(L)` reorganiza una lista in-situ, es decir, la permuta aleatoriamente.

También hay una clase `Random` que usted puede instanciar para crear múltiples generadores independientes de valores aleatorios.

Extendiendo/Embebiendo FAQ

Contenido

- Extendiendo/Embebiendo FAQ
    - ¿Puedo crear mis propias funciones en C?
    - ¿Puedo crear mis propias funciones en C++?
    - Escribir en C es difícil; ¿no hay otra alternativa?
    - ¿Cómo puedo ejecutar declaraciones arbitrarias de Python desde C?
    - ¿Cómo puedo evaluar una expresión arbitraria de Python desde C?
    - ¿Cómo extraigo valores C de un objeto Python?
    - ¿Cómo utilizo Py_BuildValue() para crear una tupla de un tamaño arbitrario?
    - ¿Cómo puedo llamar un método de un objeto desde C?
    - ¿Cómo obtengo la salida de PyErr_Print() (o cualquier cosa que se imprime a stdout/stderr)?
    - ¿Cómo accedo al módulo escrito en Python desde C?
    - ¿Cómo hago una interface a objetos C++ desde Python?
    - He agregado un módulo usando el archivo de configuración y el *make* falla. ¿Porque?
    - ¿Cómo puedo depurar una extensión?
    - Quiero compilar un módulo Python en mi sistema Linux, pero me faltan algunos archivos . ¿Por qué?
    - ¿Cómo digo «entrada incompleta» desde «entrada inválida»?
    - ¿Cómo encuentro símbolos g++ __builtin_new o __pure_virtual?
    - ¿Puedo crear una clase objeto con algunos métodos implementado en C y otros en Python (por ejemplo a través de la herencia)?

¿Puedo crear mis propias funciones en C?

Si, puedes crear módulos incorporados que contengan funciones, variables, excepciones y incluso nuevos tipos en C. Esto esta explicado en el documento Ampliación e incrustación del intérprete de Python.

La mayoría de los libros intermedios o avanzados de Python también tratan este tema.

¿Puedo crear mis propias funciones en C ++?

Si, utilizando las características de compatibilidad encontradas en C ++. Coloca `extern "C" { ... }` alrededor los archivos incluidos Python y pon `extern "C"` antes de cada función que será llamada por el interprete Python. Objetos globales o estáticos C ++ con constructores no son una buena idea seguramente.

Escribir en C es difícil; ¿no hay otra alternativa?

There are a number of alternatives to writing your own C extensions, depending on what you're trying to do. Recommended third party tools offer both simpler and more sophisticated approaches to creating C and C++ extensions for Python.

¿Cómo puedo ejecutar declaraciones arbitrarias de Python desde C?

La función de más alto nivel para hacer esto es **`PyRun_SimpleString()`** que toma un solo argumento de cadena de caracteres para ser ejecutado en el contexto del módulo `__main__` y retorna `0` si tiene éxito y `-1` cuando ocurre una excepción (incluyendo **`SyntaxError`**). Si quieres mas control, usa **`PyRun_String()`**; mira la fuente para **`PyRun_SimpleString()`** en `Python/pythonrun.c`.

¿Cómo puedo evaluar una expresión arbitraria de Python desde C?

Llama a la función `PyRun_String()` de la pregunta anterior con el símbolo de comienzo `Py_eval_input`; analiza una expresión, evalúa y retorna su valor.

¿Cómo extraigo valores C de un objeto Python?

That depends on the object's type. If it's a tuple, `PyTuple_Size()` returns its length and `PyTuple_GetItem()` returns the item at a specified index. Lists have similar functions, `PyList_Size()` and `PyList_GetItem()`.

For bytes, `PyBytes_Size()` returns its length and `PyBytes_AsStringAndSize()` provides a pointer to its value and its length. Note that Python bytes objects may contain null bytes so C's `strlen()` should not be used.

Para testear el tipo de un objeto, primero debes estar seguro que no es `NULL`, y luego usa `PyBytes_Check()`, `PyTuple_Check()`, `PyList_Check()`, etc.

También hay una API de alto nivel para objetos Python que son provistos por la supuestamente llamada interfaz "abstracta" – lee `Include/abstract.h` para mas detalles. Permite realizar una interfaz con cualquier tipo de secuencia Python usando llamadas como `PySequence_Length()`, `PySequence_GetItem()`, etc. así como otros protocolos útiles como números (`PyNumber_Index()` et al.) y mapeos en las *PyMapping APIs*.

¿Cómo utilizo Py_BuildValue() para crear una tupla de un tamaño arbitrario?

No puedes hacerlo. Utiliza a cambio `PyTuple_Pack()`.

¿Cómo puedo llamar un método de un objeto desde C?

Se puede utilizar la función **`PyObject_CallMethod()`** para llamar a un método arbitrario de un objeto. Los parámetros son el objeto, el nombre del método a llamar, una cadena de caracteres de formato como las usadas con **`Py_BuildValue()`**, y los valores de argumento

```
PyObject *
PyObject_CallMethod(PyObject *object, const char *methoc
                    const char *arg_format, ...);
```

Esto funciona para cualquier objeto que tenga métodos – sean estos incorporados o definidos por el usuario. Eres responsable si eventualmente usas **`Py_DECREF()`** en el valor de retorno.

Para llamar, por ejemplo, un método «seek» de un objeto archivo con argumentos 10, 0 (considerando que puntero del objeto archivo es «f»):

```
res = PyObject_CallMethod(f, "seek", "(ii)", 10, 0);
if (res == NULL) {
        ... an exception occurred ...
}
else {
        Py_DECREF(res);
}
```

Note que debido a **`PyObject_CallObject()`** *siempre* necesita una tupla para la lista de argumento, para llamar una función sin argumentos, deberás pasar «()» para el formato, y para llamar a una función con un solo argumento, encierra el argumento entre paréntesis, por ejemplo «(i)».

¿Cómo obtengo la salida de PyErr_Print() (o cualquier cosa que se imprime a stdout/stderr)?

En código Python, define un objeto que soporta el método `write()`. Asigna este objeto a **`sys.stdout`** y **`sys.stderr`**. Llama a print_error, o solo permite que el mecanismo estándar de rastreo funcione. Luego, la salida se hará cuando invoques `write()`.

La manera mas fácil de hacer esto es usar la clase **`io.StringIO`**:

```
>>> import io, sys
>>> sys.stdout = io.StringIO()
>>> print('foo')
>>> print('hello world!')
>>> sys.stderr.write(sys.stdout.getvalue())
foo
hello world!
```

Un objeto personalizado para hacer lo mismo se vería así:

```
>>> import io, sys
>>> class StdoutCatcher(io.TextIOBase):
...     def __init__(self):
...         self.data = []
...     def write(self, stuff):
...         self.data.append(stuff)
...
>>> import sys
>>> sys.stdout = StdoutCatcher()
>>> print('foo')
>>> print('hello world!')
>>> sys.stderr.write(''.join(sys.stdout.data))
foo
hello world!
```

¿Cómo accedo al módulo escrito en Python desde C?

Puedes obtener un puntero al módulo objeto de esta manera:

```
module = PyImport_ImportModule("<modulename>");
```

Si el módulo todavía no se importó, (por ejemplo aún no esta presente en `sys.modules`), esto inicializa el módulo, de otra forma simplemente retorna el valor de `sys.modules["<modulename>"]`. Nota que no entra el módulo a ningún espacio de nombres (*namespace*) –solo asegura que fue inicializado y guardado en `sys.modules`.

Puedes acceder luego a los atributos del módulo (por ejemplo a cualquier nombre definido en el módulo) de esta forma:

```
attr = PyObject_GetAttrString(module, "<attrname>");
```

También funciona llamar a `PyObject_SetAttrString()` para asignar a variables en el módulo.

¿Cómo hago una interface a objetos C++ desde Python?

Dependiendo de lo que necesites, hay varias maneras de hacerlo. Para hacerlo manualmente empieza por leer the «Extending and Embedding» document.Fíjate que para el sistema de tiempo de ejecución Python, no hay una gran diferencia entre C y C++, por lo que la estrategia de construir un nuevo tipo Python alrededor de una estructura de C de tipo puntero, también funcionará para objetos C++.

Para bibliotecas C++, mira Escribir en C es difícil; ¿no hay otra alternativa?.

He agregado un módulo usando el archivo de configuración y el *make* falla. ¿Porque?

La configuración debe terminar en una nueva linea, si esta no está, entonces el proceso *build* falla. (reparar esto requiere algún truco de linea de comandos que puede ser no muy prolijo, y seguramente el error es tan pequeño que no valdrá el esfuerzo.)

¿Cómo puedo depurar una extensión?

Cuando se usa GDB con extensiones cargadas de forma dinámica, puedes configurar un punto de verificación (*breakpoint*) en tu extensión hasta que esta se cargue.

En tu archivo `.gdbinit` (o interactivamente), agrega el comando:

```
br _PyImport_LoadDynamicModule
```

Luego, cuando corras GDB:

```
$ gdb /local/bin/python
gdb) run myscript.py
gdb) continue # repeat until your extension is loaded
gdb) finish   # so that your extension is loaded
gdb) br myfunction.c:50
gdb) continue
```

Quiero compilar un módulo Python en mi sistema Linux, pero me faltan algunos archivos . ¿Por qué?

Most packaged versions of Python omit some files required for compiling Python extensions.

For Red Hat, install the python3-devel RPM to get the necessary files.

For Debian, run `apt-get install python3-dev`.

¿Cómo digo «entrada incompleta» desde «entrada inválida»?

A veces quieres emular el comportamiento del interprete interactivo de Python, que te da una continuación del *prompt* cuando la entrada esta incompleta (por ejemplo si comenzaste tu instrucción «if» o no cerraste un paréntesis o triples comillas), pero te da un mensaje de error de sintaxis inmediatamente cuando la entrada es invalida.

En Python puedes usar el módulo `codeop`, que aproxima el comportamiento del analizador gramatical (*parser*) . IDLE usa esto, por ejemplo.

La manera mas fácil de hacerlo en C es llamar a `PyRun_InteractiveLoop()` (quizá en un hilo separado) y dejar que el interprete Python gestione la entrada por ti. Puedes también configurar `PyOS_ReadlineFunctionPointer()` para apuntar a tu función de entrada personalizada.

¿Cómo encuentro símbolos g++ _builtin_new o _pure_virtual?

Para cargar dinámicamente módulos de extensión g++, debes recompilar Python, hacer un nuevo *link* usando g++ (cambia LINKCC en el Python Modules Makefile) y enlaza *link* tu extensión usando g++ (por ejemplo `g++ -shared -o mymodule.so mymodule.o`).

¿Puedo crear una clase objeto con algunos métodos implementado en C y otros en Python (por ejemplo a través de la herencia)?

Si, puedes heredar de clases integradas como `int`, `list`, `dict`, etc.

The Boost Python Library (BPL, https://www.boost.org/libs/python/doc/index.html) provides a way of doing this from C++ (i.e. you can inherit from an extension class written in C++ using the BPL).

Preguntas frecuentes sobre Python en Windows

Contenido

- Preguntas frecuentes sobre Python en Windows
    - ¿Cómo ejecutar un programa Python en Windows?
    - ¿Cómo hacer que los scripts de Python sean ejecutables?
    - ¿Por qué Python tarda en comenzar?
    - ¿Cómo hacer un ejecutable a partir de un script de Python?
    - ¿Es un archivo `*.pyd` lo mismo que una DLL?
    - ¿Cómo puedo integrar Python en una aplicación de Windows?
    - ¿Cómo puedo evitar que mi editor inserte pestañas en mi archivo fuente de Python?
    - ¿Cómo verifico una pulsación de tecla sin bloquearla?
    - ¿Cómo resuelvo el error de api-ms-win-crt-runtime-l1-1-0.dll no encontrado?

¿Cómo ejecutar un programa Python en Windows?

No es necesariamente una pregunta simple. Si ya está familiarizado con el lanzamiento de programas desde la línea de comandos de Windows, todo parecerá obvio; de lo contrario, es posible que necesite un poco más de orientación.

A menos que esté utilizando algún tipo de entorno de desarrollo, terminará escribiendo comandos de Windows en lo que se denomina «DOS» o «símbolo del sistema de Windows». En general, puede abrir dicha ventana desde su barra de búsqueda buscando *cmd*. Debería poder reconocer cuándo inició dicha ventana porque verá un símbolo del sistema de Windows, que en general se ve así:

```
C:\>
```

La letra puede ser diferente y puede haber otras cosas seguidas, por lo que también puede verse así:

```
D:\YourName\Projects\Python>
```

dependiendo de la configuración de su computadora y de lo que haya hecho recientemente con ella. Una vez que haya abierto esta ventana, está en camino de iniciar los programas de Python.

Tenga en cuenta que sus scripts de Python deben ser procesados por otro programa llamado «intérprete» de Python. El intérprete lee su script, lo compila en *bytecode* y ejecuta el *bytecode* para ejecutar su programa. Entonces, ¿cómo le das tu código Python al intérprete?

Primero, debe asegurarse de que la ventana del símbolo del sistema reconoce la palabra «python» como una instrucción para iniciar el intérprete. Si abrió un símbolo del sistema, escriba el comando `py` y luego presione la tecla Enter:

```
C:\Users\YourName> py
```

Debería ver algo como esto:

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MS
Type "help", "copyright", "credits" or "license" for mor
>>>
```

Has comenzado el intérprete en su «modo interactivo». Esto significa que puede ingresar declaraciones o expresiones de Python de forma interactiva para ejecutarlas. Esta es una de las características más poderosas de Python. Puede verificar esto ingresando algunos comandos de su elección y ver los resultado:

```
>>> print("Hello")
Hello
>>> "Hello" * 3
'HelloHelloHello'
```

Muchas personas usan el modo interactivo como una calculadora práctica pero altamente programable. Cuando desee finalizar su sesión interactiva de Python, llame a la función **`exit()`** o mantenga presionada la tecla `Ctrl` mientras ingresa una `Z`, luego presione la tecla «`Enter`» para regresar a su símbolo del sistema de Windows.

Es posible que haya notado una nueva entrada en su menú Inicio, como *Inicio* ‣ *Programas* ‣ *Python 3.x* ‣ *Python (línea de comando)* que hace que vea el mensaje `>>>` en una nueva ventana. Si es así, la ventana desaparecerá cuando llame a la función **`exit()`** o presione la combinación `Ctrl-Z`; Windows ejecuta un comando «python» en la ventana y lo cierra cuando cierra el intérprete.

Ahora que sabemos que se reconoce el comando `py`, puede darle su script Python. Debe proporcionar la ruta absoluta o relativa de la secuencia de comandos de Python. Digamos que su script Python se encuentra en su escritorio y se llama `hello.py`, y su símbolo del sistema está abierto en su directorio de inicio, por lo que verá algo como:

```
C:\Users\YourName>
```

Entonces, le pedirá al comando `py` que le dé su script a Python escribiendo `py` seguido de la ruta de su script:

```
C:\Users\YourName> py Desktop\hello.py
hello
```

¿Cómo hacer que los scripts de Python sean ejecutables?

En Windows, el instalador de Python asocia la extensión *.py* con un tipo de archivo *(Python.File)* y un comando que inicia el intérprete (`D:\Archivos de programa\Python\python.exe "%1" %*`). Esto es suficiente para poder ejecutar scripts de Python desde la línea de comandos ingresando “foo.py”. Si desea poder ejecutar los scripts simplemente escribiendo “foo” sin la extensión, debe agregar .py a la variable del entorno PATHEXT.

¿Por qué Python tarda en comenzar?

Normalmente en Windows, Python se inicia muy rápidamente, pero a veces los informes de error indican que Python de repente comienza a tardar mucho tiempo en iniciarse. Es aún más desconcertante porque Python funcionará correctamente con otros Windows configurados de manera idéntica.

El problema puede provenir de un antivirus mal configurado. Se sabe que algunos antivirus duplican el tiempo de arranque cuando se configuran para verificar todas las lecturas del sistema de archivos. Intente verificar si los antivirus de las dos máquinas están configurados correctamente de manera idéntica. *McAfee* es especialmente problemático cuando se configura para verificar todas las lecturas del sistema de archivos.

¿Cómo hacer un ejecutable a partir de un script de Python?

Vea ¿Cómo puedo crear un binario independiente a partir de un programa Python? para una lista de herramientas que pueden ser usadas para crear ejecutables.

¿Es un archivo `*.pyd` lo mismo que una DLL?

Sí, los archivos *.pyd* son archivos dll, pero hay algunas diferencias. Si tiene una DLL llamada `foo.pyd`, debe tener una función `PyInit_foo()`. Luego puede escribir en Python «import foo» y Python buscará el archivo *foo.pyd* (así como *foo.py* y *foo.pyc*); si lo encuentra, intentará llamar a `PyInit_foo()` para inicializarlo. No vincules tu .exe con *foo.lib* porque en este caso Windows necesitará la DLL.

Tenga en cuenta que la ruta de búsqueda para *foo.pyd* es PYTHONPATH, es diferente de la que usa Windows para buscar *foo.dll*. Además, *foo.pyd* no necesita estar presente para que su programa se ejecute, mientras que si ha vinculado su programa con una dll, esto es necesario. Por supuesto, *foo.pyd* es necesario si escribes `import foo`. En una DLL, el enlace se declara en el código fuente con `__declspec(dllexport)`. En un *.pyd*, la conexión se define en una lista de funciones disponibles.

¿Cómo puedo integrar Python en una aplicación de Windows?

La integración del intérprete de Python en una aplicación de Windows se puede resumir de la siguiente manera:

1. Do **not** build Python into your .exe file directly. On Windows, Python must be a DLL to handle importing modules that are themselves DLL's. (This is the first key undocumented fact.) Instead, link to `pythonNN.dll`; it is typically installed in `C:\Windows\System`. *NN* is the Python version, a number such as «33» for Python 3.3.

   Puede vincular a Python de dos maneras diferentes. Un enlace en tiempo de carga significa apuntar al archivo `pythonNN.lib`, mientras que un enlace en tiempo de ejecución significa apuntar al archivo `pythonNN.dll`. (Nota general: el archivo `pythonNN.lib` es la llamada «lib de importación» correspondiente para el archivo `pythonNN.dll`. Simplemente define enlaces simbólicos para el enlazador).

   El enlace en tiempo real simplifica enormemente las opciones de enlace; Todo sucede en el momento de la ejecución. Su código debe cargar el archivo `pythonNN.dll` utilizando la rutina de Windows `LoadLibraryEx()`. El código también debe usar rutinas de acceso y datos en el archivo `pythonNN.dll` (es decir, las API C de Python) usando punteros obtenidos por la rutina de Windows `GetProcAddress()`. Las macros pueden hacer que el uso de estos punteros sea transparente para cualquier código C que llame a rutinas en la API C de Python.

2. If you use SWIG, it is easy to create a Python «extension module» that will make the app's data and methods available

to Python. SWIG will handle just about all the grungy details for you. The result is C code that you link *into* your .exe file (!) You do **not** have to create a DLL file, and this also simplifies linking.

3. *SWIG* creará una función de inicialización (función en C) cuyo nombre depende del nombre del complemento. Por ejemplo, si el nombre del módulo es leo, la función *init* se llamará *initleo()*. Si está utilizando clases *shadow SWIG*, como debería, la función *init* se llamará *initleoc()*. Esto inicializa una clase auxiliar invisible utilizada por la clase *shadow*.

   ¡La razón por la que puede vincular el código C en el paso 2 en su archivo .exe es que llamar a la función de inicialización es equivalente a importar el módulo a Python! (Este es el segundo hecho clave indocumentado).

4. En resumen, puede usar el siguiente código para inicializar el intérprete de Python con su complemento.

   ```
   #include <Python.h>
   ...
   Py_Initialize();  // Initialize Python.
   initmyAppc();  // Initialize (import) the helper cl
   PyRun_SimpleString("import myApp");  // Import the
   ```

5. Hay dos problemas con la API de Python C que aparecerán si utiliza un compilador que no sea *MSVC*, el compilador utilizado para construir *pythonNN.dll*.

   Problem 1: The so-called «Very High Level» functions that take `FILE *` arguments will not work in a multi-compiler environment because each compiler's notion of a `struct FILE` will be different. From an implementation standpoint these are very low level functions.

   Problema 2: *SWIG* genera el siguiente código al generar contenedores para cancelar las funciones:

   ```
   Py_INCREF(Py_None);
   _resultobj = Py_None;
   ```

```
return _resultobj;
```

Por desgracia, *Py_None* es una macro que se desarrolla con referencia a una estructura de datos compleja llamada *_Py_NoneStruct* dentro de *pythonNN.dll*. Nuevamente, este código fallará en un entorno de múltiples compiladores. Reemplace este código con:

```
return Py_BuildValue("");
```

Es posible utilizar el comando *SWIG* `%typemap` para realizar el cambio automáticamente, aunque no he logrado que funcione (soy un principiante con *SWIG*).

6. Usar una secuencia de comandos de shell Python para crear una ventana de intérprete de Python desde su aplicación de Windows no es una buena idea; la ventana resultante será independiente del sistema de ventanas de su aplicación. Usted (o la clase *wxPythonWindow*) debería crear una ventana de intérprete «nativa». Es fácil conectar esta ventana al intérprete de Python. Puede redirigir la entrada/salida de Python a cualquier objeto que admita lectura y escritura, por lo que todo lo que necesita es un objeto de Python (definido en su complemento) que contenga los métodos de *read()* y *write()*.

¿Cómo puedo evitar que mi editor inserte pestañas en mi archivo fuente de Python?

Las preguntas frecuentes no recomiendan el uso de pestañas y la guía de estilo de Python, **PEP 8** [https://peps.python.org/pep-0008/], recomienda el uso de 4 espacios para distribuir el código de Python. Este es también el modo predeterminado de Emacs con Python.

Sea cual sea su editor, mezclar pestañas y espacios es una mala idea. *MSVC* no es diferente en este aspecto, y se puede configurar fácilmente para usar espacios: vaya a *Tools* ‣ *Options* ‣ *Tabs*, y para el tipo de archivo «Default», debe establecer «Tab size» and «Indent size» en 4, luego seleccione Insertar espacios.

Python lanzará los errores `IndentationError` o `TabError` si una combinación de tabulación y sangría es problemática al comienzo de la línea. También puede usar el módulo `tabnanny` para detectar estos errores.

¿Cómo verifico una pulsación de tecla sin bloquearla?

Use el módulo `msvcrt`. Es una extensión estándar específica de Windows, que define una función `kbhit()` que verifica si se ha presionado una tecla, y `getch()` que recupera el carácter sin mostrarlo.

¿Cómo resuelvo el error de api-ms-win-crt-runtime-l1-1-0.dll no encontrado?

Esto puede ocurrir en Python 3.5 y posteriores usando Windows 8.1 o anteriores sin que todas las actualizaciones hayan sido instaladas. Primero asegúrese que su sistema operativo sea compatible y esté actualizado, y si eso no resuelve el problema, visite la Página de soporte de Microsoft [https://support.microsoft.com/en-us/help/3118401/] para obtener orientación sobre como instalar manualmente la actualización de C Runtime.

Preguntas frecuentes sobre la Interfaz Gráfica de Usuario (*GUI*)

Contenido

- Preguntas frecuentes sobre la Interfaz Gráfica de Usuario (*GUI*)
  - Preguntas generales de la GUI
  - ¿Qué conjuntos de herramientas de GUI existen para Python?
  - Preguntas de Tkinter
    - ¿Cómo congelo las aplicaciones de Tkinter?
    - ¿Puedo tener eventos Tk manejados mientras espero por *I/O*?
    - No puedo hacer que los atajos de teclado funcionen en Tkinter: ¿por qué?

Preguntas generales de la GUI

¿Qué conjuntos de herramientas de GUI existen para Python?

Los empaquetados estándar de Python incluyen una interfaz orientada a objetos para el conjunto de widgets de Tcl/Tk, llamada tkinter. Esta es probablemente la más fácil de instalar (ya que viene incluida con la mayoría de distribuciones binarias [https://www.python.org/downloads/] de Python) y usar. Para obtener más información sobre Tk, incluyendo referencias a la fuente, ver la Página de inicio Tcl/Tk [https://www.tcl.tk]. Tcl/Tk es totalmente portable a macOS, Windows y plataformas Unix.

Dependiendo de a que plataforma(s) estés apuntando, hay también múltiples alternativas. Una lista de conjuntos de herramientas multiplataforma [https://wiki.python.org/moin/GuiProgramming#Cross-Platform_Frameworks] y de plataforma específica [https://wiki.python.org/moin/GuiProgramming#Platform-specific_Frameworks] puede ser encontrada en la wiki de Python.

Preguntas de Tkinter

¿Cómo congelo las aplicaciones de Tkinter?

Freeze es una herramienta para crear aplicaciones independientes. Al congelar aplicaciones Tkinter, las aplicaciones no serán realmente independientes, ya que la aplicación seguirá necesitando las bibliotecas Tcl y Tk.

Una solución es enviar la aplicación con las bibliotecas Tcl y Tk, y apuntarlas en tiempo de ejecución utilizando las variables de entorno `TCL_LIBRARY` y `TK_LIBRARY`.

Varias bibliotecas de terceros para congelar como py2exe y cx_Freeze tienen soporte integrado para aplicaciones Tkinter.

¿Puedo tener eventos Tk manejados mientras espero por *I/O*?

En plataformas que no sean Windows, sí, ¡y ni siquiera necesitas hilos! Pero tendrá que reestructurar un poco su código de entrada/salida. Tk tiene un equivalente de la llamada `XtAddInput()` de Xt, que permite registrar una función de retorno que será llamada desde el bucle principal de Tk cuando sea posible entrada/salida en un descriptor de archivo. Ver Gestor de archivos.

No puedo hacer que los atajos de teclado funcionen en Tkinter: ¿por qué?

Una queja frecuente que se escucha es que los controladores de eventos vinculados con el método `bind()` no se ejecutan incluso cuando se presiona la tecla correspondiente.

La causa más común es que el widget al que se aplica el atajo no tiene enfoque de teclado. Consulte la documentación de Tk para el comando de *focus*. Por lo general, un *widget* recibe el foco del teclado haciendo clic en él (pero no para las etiquetas; consulte la opción *takefocus*).

«¿Por qué está Python instalado en mi ordenador?» FAQ

¿Qué es Python?

Python es un lenguaje de programación. Se usa para muchas aplicaciones diferentes. Se utiliza en algunas escuelas secundarias y universidades como un lenguaje de programación introductorio porque Python es fácil de aprender, pero también es utilizado por desarrolladores de software profesionales en lugares como Google, NASA, y Lucasfilm Ltd.

Si desea aprender más sobre Python, comience con la Guía del principiante de Python [https://wiki.python.org/moin/BeginnersGuide].

¿Por qué Python está instalado en mi máquina?

Si encuentras Python instalado en tu sistema pero no recuerdas haberlo instalado, hay varias maneras posibles en las que podría haber llegado ahí.

- Tal vez otro usuario de la computadora quiso aprender a programar y la instaló.
- Una aplicación de terceros instalada en la máquina podría haber sido escrita en Python e incluir una instalación de Python. Hay muchas aplicaciones de este tipo, desde programas GUI hasta servidores de red y scripts administrativos.
- Algunas máquinas Windows también tienen Python instalado. Al momento de escribir este artículo, sabemos que las computadoras de Hewlett-Packard y Compaq incluyen

Python. Aparentemente algunas de las herramientas administrativas de HP/Compaq están escritas en Python.
- Muchos sistemas operativos compatibles con Unix, como macOS y algunas distribuciones de Linux, tienen Python instalado de forma predeterminada; está incluido en la instalación básica.

¿Puedo eliminar Python?

Eso depende de dónde vino Python.

Si alguien lo instaló deliberadamente, puede quitarlo sin dañar nada. En Windows, utilice el icono Agregar o quitar programas en el Panel de control.

Si Python fue instalado por una aplicación de terceros, también puede eliminarlo, pero esa aplicación ya no funcionará. Deberías usar el desinstalador de esa aplicación en lugar de eliminar Python directamente.

Si Python vino con su sistema operativo, no se recomienda quitarlo. Si lo eliminas, las herramientas escritas en Python ya no funcionarán, y algunas de ellas pueden ser importantes para ti. Reinstalar todo el sistema sería entonces necesario para arreglar las cosas de nuevo.

Desusos

Pending removal in Python 3.15

- The import system:
  - Setting `__cached__` on a module while failing to set `__spec__.cached` is deprecated. In Python 3.15, `__cached__` will cease to be set or take into consideration by the import system or standard library. (gh-97879 [https://github.com/python/cpython/issues/97879])
  - Setting `__package__` on a module while failing to set `__spec__.parent` is deprecated. In Python 3.15, `__package__` will cease to be set or take into consideration by the import system or standard library. (gh-97879 [https://github.com/python/cpython/issues/97879])
- `ctypes`:
  - La función `ctypes.SetPointerType()` no documentada ha quedado obsoleta desde Python 3.13.
- `http.server`:
  - El obsoleto y poco utilizado `CGIHTTPRequestHandler` ha quedado obsoleto desde Python 3.13. No existe un reemplazo directo. *Anything* es mejor que CGI para interconectar un servidor web con un controlador de solicitudes.
  - La bandera `--cgi` de la interfaz de línea de comandos **python -m http.server** ha quedado obsoleta desde Python 3.13.
- `importlib`:
  - Método `load_module()`: utilice `exec_module()` en su lugar.
- `locale`:
  - La función `getdefaultlocale()` ha quedado obsoleta desde Python 3.11. Su eliminación se había planeado originalmente para Python 3.13 (gh-90817 [https://github.com/python/cpython/issues/90817]), pero se

    pospuso hasta Python 3.15. Utilice `getlocale()`, `setlocale()` y `getencoding()` en su lugar. (Contribuido por Hugo van Kemenade en gh-111187 [https://github.com/python/cpython/issues/111187].)

- `pathlib`:
    - ○ `PurePath.is_reserved()` ha quedado obsoleto desde Python 3.13. Utilice `os.path.isreserved()` para detectar rutas reservadas en Windows.
- `platform`:
    - ○ `java_ver()` ha quedado obsoleto desde Python 3.13. Esta función solo es útil para la compatibilidad con Jython, tiene una API confusa y, en gran medida, no se ha probado.
- `sysconfig`:
    - ○ The *check_home* argument of `sysconfig.is_python_build()` has been deprecated since Python 3.12.
- `threading`:
    - ○ `RLock()` will take no arguments in Python 3.15. Passing any arguments has been deprecated since Python 3.14, as the Python version does not permit any arguments, but the C version allows any number of positional or keyword arguments, ignoring every argument.
- `types`:
    - ○ `types.CodeType`: Accessing `co_lnotab` was deprecated in **PEP 626** [https://peps.python.org/pep-0626/] since 3.10 and was planned to be removed in 3.12, but it only got a proper `DeprecationWarning` in 3.12. May be removed in 3.15. (Contributed by Nikita Sobolev in gh-101866 [https://github.com/python/cpython/issues/101866].)
- `typing`:
    - ○ The undocumented keyword argument syntax for creating `NamedTuple` classes (for example, `Point = NamedTuple("Point", x=int, y=int)`) has been deprecated since Python 3.13. Use the class-based syntax or the functional syntax instead.
    - ○ When using the functional syntax of `TypedDict`s, failing to pass a value to the *fields* parameter (`TD =

    `TypedDict("TD")`) or passing `None` (`TD = TypedDict("TD", None)`) has been deprecated since Python 3.13. Use `class TD(TypedDict): pass` or `TD = TypedDict("TD", {})` to create a TypedDict with zero field.
  - La función decoradora **`typing.no_type_check_decorator()`** ha quedado obsoleta desde Python 3.13. Después de ocho años en el módulo **`typing`**, todavía no ha sido compatible con ningún verificador de tipos importante.
- **`wave`**:
  - Los métodos **`getmark()`**, **`setmark()`** y **`getmarkers()`** de las clases **`Wave_read`** y **`Wave_write`** han quedado obsoletos desde Python 3.13.
- **`zipimport`**:
  - **`load_module()`** has been deprecated since Python 3.10. Use **`exec_module()`** instead. (Contributed by Jiahao Li in gh-125746 [https://github.com/python/cpython/issues/125746].)

Pending removal in Python 3.16

- The import system:

  - Setting **`__loader__`** on a module while failing to set **`__spec__.loader`** is deprecated. In Python 3.16, **`__loader__`** will cease to be set or taken into consideration by the import system or the standard library.

- **`array`**:

  - El código de formato `'u'` (**`wchar_t`**) ha quedado obsoleto en la documentación desde Python 3.3 y en el entorno de ejecución desde Python 3.13. Utilice el código de formato `'w'` (**`Py_UCS4`**) para caracteres Unicode.

- **`asyncio`**:

  - `asyncio.iscoroutinefunction()` is deprecated and will be removed in Python 3.16; use `inspect.iscoroutinefunction()` instead. (Contributed by Jiahao Li and Kumar Aditya in gh-122875 [https://github.com/python/cpython/issues/122875].)

  - `asyncio` policy system is deprecated and will be removed in Python 3.16. In particular, the following classes and functions are deprecated:

    - `asyncio.AbstractEventLoopPolicy`
    - `asyncio.DefaultEventLoopPolicy`
    - `asyncio.WindowsSelectorEventLoopPolicy`
    - `asyncio.WindowsProactorEventLoopPolicy`
    - `asyncio.get_event_loop_policy()`
    - `asyncio.set_event_loop_policy()`

    Users should use `asyncio.run()` or `asyncio.Runner` with *loop_factory* to use the desired event loop implementation.

    For example, to use `asyncio.SelectorEventLoop` on Windows:

    ```
    import asyncio

    async def main():
        ...

    asyncio.run(main(), loop_factory=asyncio.Select
    ```

    (Contributed by Kumar Aditya in gh-127949 [https://github.com/python/cpython/issues/127949].)

- `builtins`:

  - La inversión bit a bit en tipos booleanos, `~True` o `~False`, ha quedado obsoleta desde Python 3.12, ya que produce resultados sorprendentes y poco intuitivos (`-2` y `-1`). En su lugar, utilice `not x` para la negación lógica de un valor booleano. En el caso poco frecuente

  de que necesite la inversión bit a bit del entero subyacente, convierta a `int` explícitamente (`~int(x)`).

- **`functools`**:
  - Calling the Python implementation of **`functools.reduce()`** with *function* or *sequence* as keyword arguments has been deprecated since Python 3.14.
- **`logging`**:

  Support for custom logging handlers with the *strm* argument is deprecated and scheduled for removal in Python 3.16. Define handlers with the *stream* argument instead. (Contributed by Mariusz Felisiak in gh-115032 [https://github.com/python/cpython/issues/115032].)
- **`mimetypes`**:
  - Valid extensions start with a "." or are empty for **`mimetypes.MimeTypes.add_type()`**. Undotted extensions are deprecated and will raise a **`ValueError`** in Python 3.16. (Contributed by Hugo van Kemenade in gh-75223 [https://github.com/python/cpython/issues/75223].)
- **`shutil`**:
  - La excepción **`ExecError`** ha quedado obsoleta desde Python 3.14. No ha sido utilizada por ninguna función en **`shutil`** desde Python 3.4 y ahora es un alias de **`RuntimeError`**.
- **`symtable`**:
  - El método **`Class.get_methods`** ha quedado obsoleto desde Python 3.14.
- **`sys`**:

  - La función `_enablelegacywindowsfsencoding()` ha quedado obsoleta desde Python 3.13. En su lugar, utilice la variable de entorno `PYTHONLEGACYWINDOWSFSENCODING`.

- `sysconfig`:

  - The `sysconfig.expand_makefile_vars()` function has been deprecated since Python 3.14. Use the `vars` argument of `sysconfig.get_paths()` instead.

- `tarfile`:

  - El atributo `TarFile.tarfile`, no documentado ni utilizado, ha quedado obsoleto desde Python 3.13.

Pending removal in Python 3.17

- `collections.abc`:

  - `collections.abc.ByteString` is scheduled for removal in Python 3.17.

    Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use `Buffer` or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).

    `ByteString` was originally intended to be an abstract class that would serve as a supertype of both `bytes` and `bytearray`. However, since the ABC never had any methods, knowing that an object was an instance of `ByteString` never actually told you anything useful about the object. Other common buffer types such as `memoryview` were also never understood as subtypes of `ByteString` (either at runtime or by static type checkers).

See **PEP 688** [https://peps.python.org/pep-0688/#current-options] for more details. (Contributed by Shantanu Jain in gh-91896 [https://github.com/python/cpython/issues/91896].)

- **`typing`**:
    - Before Python 3.14, old-style unions were implemented using the private class `typing._UnionGenericAlias`. This class is no longer needed for the implementation, but it has been retained for backward compatibility, with removal scheduled for Python 3.17. Users should use documented introspection helpers like **`typing.get_origin()`** and **`typing.get_args()`** instead of relying on private implementation details.
    - **`typing.ByteString`**, deprecated since Python 3.9, is scheduled for removal in Python 3.17.

      Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use **`Buffer`** or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).

      `ByteString` was originally intended to be an abstract class that would serve as a supertype of both **`bytes`** and **`bytearray`**. However, since the ABC never had any methods, knowing that an object was an instance of **`ByteString`** never actually told you anything useful about the object. Other common buffer types such as **`memoryview`** were also never understood as subtypes of **`ByteString`** (either at runtime or by static type checkers).

      See **PEP 688** [https://peps.python.org/pep-0688/#current-options] for more details. (Contributed by Shantanu Jain in gh-91896 [https://github.com/python/cpython/issues/91896].)

Pending removal in Python 3.18

- **`decimal`**:
    - The non-standard and undocumented **`Decimal`** format specifier `'N'`, which is only supported in the **`decimal`** module's C implementation, has been deprecated since Python 3.13. (Contributed by Serhiy Storchaka in gh-89902 [https://github.com/python/cpython/issues/89902].)

Pending removal in Python 3.19

- **`ctypes`**:
    - Implicitly switching to the MSVC-compatible struct layout by setting **`_pack_`** but not **`_layout_`** on non-Windows platforms.

Pending removal in future versions

Las siguientes API se eliminarán en el futuro, aunque actualmente no hay una fecha programada para su eliminación.

- **`argparse`**:
    - Nesting argument groups and nesting mutually exclusive groups are deprecated.
    - Passing the undocumented keyword argument *prefix_chars* to **`add_argument_group()`** is now deprecated.
    - The **`argparse.FileType`** type converter is deprecated.
- **`builtins`**:
    - Generadores: las firmas `throw(type, exc, tb)` y `athrow(type, exc, tb)` están obsoletas: utilice `throw(exc)` y `athrow(exc)` en su lugar, la firma de argumento único.
    - Actualmente, Python acepta literales numéricos seguidos inmediatamente de palabras clave, por ejemplo, `0in x`, `1or x`, `0if 1else 2`. Permite

  expresiones confusas y ambiguas como `[0x1for x in y]` (que se puede interpretar como `[0x1 for x in y]` o `[0x1f or x in y]`). Se genera una advertencia de sintaxis si el literal numérico va seguido inmediatamente de una de las palabras clave **`and`**, **`else`**, **`for`**, **`if`**, **`in`**, **`is`** y **`or`**. En una versión futura, se cambiará a un error de sintaxis. (gh-87999 [https://github.com/python/cpython/issues/87999])
  - Compatibilidad con los métodos `__index__()` y `__int__()` que devuelven un tipo que no es int: estos métodos serán necesarios para devolver una instancia de una subclase estricta de **`int`**.
  - Compatibilidad con el método `__float__()` que devuelve una subclase estricta de **`float`**: estos métodos serán necesarios para devolver una instancia de **`float`**.
  - Compatibilidad con el método `__complex__()` que devuelve una subclase estricta de **`complex`**: estos métodos serán necesarios para devolver una instancia de **`complex`**.
  - Ahora está obsoleto el paso de un número complejo como argumento *real* o *imag* en el constructor **`complex()`**; solo debe pasarse como un único argumento posicional. (Contribuido por Serhiy Storchaka en gh-109218 [https://github.com/python/cpython/issues/109218].)
- **`calendar`**: Las constantes `calendar.January` y `calendar.February` han quedado obsoletas y han sido reemplazadas por **`calendar.JANUARY`** y **`calendar.FEBRUARY`**. (Contribuido por Prince Roshan en gh-103636 [https://github.com/python/cpython/issues/103636].)
- **`codecs`**: use **`open()`** instead of **`codecs.open()`**. (gh-133038 [https://github.com/python/cpython/issues/133038])
- **`codeobject.co_lnotab`**: utilice el método **`codeobject.co_lines()`** en su lugar.
- **`datetime`**:
  - **`utcnow()`**: utilice `datetime.datetime.now(tz=datetime.UTC)`.
  - **`utcfromtimestamp()`**: utilice `datetime.datetime.fromtimestamp(timestamp,`

`tz=datetime.UTC)`.

- **`gettext`**: El valor plural debe ser un número entero.
- **`importlib`**:
  - **`cache_from_source()`** El parámetro *debug_override* está obsoleto: utilice el parámetro *optimization* en su lugar.
- **`importlib.metadata`**:
  - Interfaz de tupla `EntryPoints`.
  - `None` implícito en los valores de retorno.
- **`logging`**: el método `warn()` ha quedado obsoleto desde Python 3.3, utilice **`warning()`** en su lugar.
- **`mailbox`**: El uso del modo de entrada y texto StringIO está obsoleto; en su lugar, utilice BytesIO y el modo binario.
- **`os`**: Llamada a **`os.register_at_fork()`** en un proceso multiproceso.
- **`pydoc.ErrorDuringImport`**: Un valor de tupla para el parámetro *exc_info* está obsoleto, utilice una instancia de excepción.
- **`re`**: Ahora se aplican reglas más estrictas para las referencias numéricas de grupos y los nombres de grupos en expresiones regulares. Ahora solo se aceptan secuencias de dígitos ASCII como referencia numérica. El nombre de grupo en patrones de bytes y cadenas de reemplazo ahora solo puede contener letras y dígitos ASCII y guiones bajos. (Contribuido por Serhiy Storchaka en gh-91760 [https://github.com/python/cpython/issues/91760].)
- Módulos **`sre_compile`**, **`sre_constants`** y **`sre_parse`**.
- **`shutil`**: El parámetro *onerror* de **`rmtree()`** está obsoleto en Python 3.12; utilice el parámetro *onexc* en su lugar.
- Opciones y protocolos **`ssl`**:
  - **`ssl.SSLContext`** sin argumento de protocolo está obsoleto.
  - **`ssl.SSLContext`**: **`set_npn_protocols()`** y **`selected_npn_protocol()`** están obsoletos: utilice ALPN en su lugar.
  - Opciones de `ssl.OP_NO_SSL*`
  - Opciones de `ssl.OP_NO_TLS*`
  - `ssl.PROTOCOL_SSLv3`
  - `ssl.PROTOCOL_TLS`
  - `ssl.PROTOCOL_TLSv1`

  - `ssl.PROTOCOL_TLSv1_1`
  - `ssl.PROTOCOL_TLSv1_2`
  - `ssl.TLSVersion.SSLv3`
  - `ssl.TLSVersion.TLSv1`
  - `ssl.TLSVersion.TLSv1_1`
- Métodos `threading`:
  - **`threading.Condition.notifyAll()`**: utilice `notify_all()`.
  - **`threading.Event.isSet()`**: utilice `is_set()`.
  - **`threading.Thread.isDaemon()`**, `threading.Thread.setDaemon()`: utilice el atributo `threading.Thread.daemon`.
  - **`threading.Thread.getName()`**, `threading.Thread.setName()`: utilice el atributo `threading.Thread.name`.
  - **`threading.currentThread()`**: utilice `threading.current_thread()`.
  - **`threading.activeCount()`**: utilice `threading.active_count()`.
- `typing.Text` (gh-92332 [https://github.com/python/cpython/issues/92332]).
- The internal class `typing._UnionGenericAlias` is no longer used to implement `typing.Union`. To preserve compatibility with users using this private class, a compatibility shim will be provided until at least Python 3.17. (Contributed by Jelle Zijlstra in gh-105499 [https://github.com/python/cpython/issues/105499].)
- `unittest.IsolatedAsyncioTestCase`: está obsoleto devolver un valor que no sea `None` de un caso de prueba.
- Funciones obsoletas de `urllib.parse`: `urlparse()` en su lugar
  - `splitattr()`
  - `splithost()`
  - `splitnport()`
  - `splitpasswd()`
  - `splitport()`
  - `splitquery()`
  - `splittag()`
  - `splittype()`
  - `splituser()`

    - `splitvalue()`
    - `to_bytes()`
- **`wsgiref`**: `SimpleHandler.stdout.write()` no debería realizar escrituras parciales.
- **`xml.etree.ElementTree`**: La prueba del valor de verdad de un **`Element`** está obsoleta. En una versión futura, siempre devolverá `True`. En su lugar, es preferible realizar pruebas explícitas `len(elem)` o `elem is not None`.
- **`sys._clear_type_cache()`** is deprecated: use **`sys._clear_internal_caches()`** instead.

C API deprecations

Pending removal in Python 3.15

- **`PyImport_ImportModuleNoBlock()`**: utilice **`PyImport_ImportModule()`** en su lugar.

- **`PyWeakref_GetObject()`** and **`PyWeakref_GET_OBJECT()`**: Use **`PyWeakref_GetRef()`** instead. The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get **`PyWeakref_GetRef()`** on Python 3.12 and older.

- Tipo **`Py_UNICODE`** y macro **`Py_UNICODE_WIDE`**: utilice **`wchar_t`** en su lugar.

- **`PyUnicode_AsDecodedObject()`**: Utilice **`PyCodec_Decode()`** en su lugar.

- **`PyUnicode_AsDecodedUnicode()`**: Use **`PyCodec_Decode()`** instead; Note that some codecs (for example, «base64») may return a type other than **`str`**, such as **`bytes`**.

- **`PyUnicode_AsEncodedObject()`**: Utilice **`PyCodec_Encode()`** en su lugar.

- **`PyUnicode_AsEncodedUnicode()`**: Use **`PyCodec_Encode()`** instead; Note that some codecs (for

example, «base64») may return a type other than `bytes`, such as `str`.

- Python initialization functions, deprecated in Python 3.13:
    - `Py_GetPath()`: Use `PyConfig_Get("module_search_paths")` (`sys.path`) instead.
    - `Py_GetPrefix()`: Use `PyConfig_Get("base_prefix")` (`sys.base_prefix`) instead. Use `PyConfig_Get("prefix")` (`sys.prefix`) if virtual environments need to be handled.
    - `Py_GetExecPrefix()`: Use `PyConfig_Get("base_exec_prefix")` (`sys.base_exec_prefix`) instead. Use `PyConfig_Get("exec_prefix")` (`sys.exec_prefix`) if virtual environments need to be handled.
    - `Py_GetProgramFullPath()`: Use `PyConfig_Get("executable")` (`sys.executable`) instead.
    - `Py_GetProgramName()`: Use `PyConfig_Get("executable")` (`sys.executable`) instead.
    - `Py_GetPythonHome()`: Use `PyConfig_Get("home")` or the `PYTHONHOME` environment variable instead.

  The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get `PyConfig_Get()` on Python 3.13 and older.

- Funciones para configurar la inicialización de Python, obsoletas en Python 3.11:
    - `PySys_SetArgvEx()`: Establezca `PyConfig.argv` en su lugar.
    - `PySys_SetArgv()`: Establezca `PyConfig.argv` en su lugar.
    - `Py_SetProgramName()`: Establezca

  `PyConfig.program_name` en su lugar.
  - `Py_SetPythonHome()`: Establezca `PyConfig.home` en su lugar.
  - `PySys_ResetWarnOptions()`: borre `sys.warnoptions` y `warnings.filters` en su lugar.

  La API `Py_InitializeFromConfig()` debe utilizarse con `PyConfig`.

- Variables de configuración global:
  - `Py_DebugFlag`: Use `PyConfig.parser_debug` or `PyConfig_Get("parser_debug")` instead.
  - `Py_VerboseFlag`: Use `PyConfig.verbose` or `PyConfig_Get("verbose")` instead.
  - `Py_QuietFlag`: Use `PyConfig.quiet` or `PyConfig_Get("quiet")` instead.
  - `Py_InteractiveFlag`: Use `PyConfig.interactive` or `PyConfig_Get("interactive")` instead.
  - `Py_InspectFlag`: Use `PyConfig.inspect` or `PyConfig_Get("inspect")` instead.
  - `Py_OptimizeFlag`: Use `PyConfig.optimization_level` or `PyConfig_Get("optimization_level")` instead.
  - `Py_NoSiteFlag`: Use `PyConfig.site_import` or `PyConfig_Get("site_import")` instead.
  - `Py_BytesWarningFlag`: Use `PyConfig.bytes_warning` or `PyConfig_Get("bytes_warning")` instead.
  - `Py_FrozenFlag`: Use `PyConfig.pathconfig_warnings` or `PyConfig_Get("pathconfig_warnings")` instead.
  - `Py_IgnoreEnvironmentFlag`: Use `PyConfig.use_environment` or `PyConfig_Get("use_environment")` instead.
  - `Py_DontWriteBytecodeFlag`: Use `PyConfig.write_bytecode` or `PyConfig_Get("write_bytecode")` instead.

  - `Py_NoUserSiteDirectory`: Use `PyConfig.user_site_directory` or `PyConfig_Get("user_site_directory")` instead.
  - `Py_UnbufferedStdioFlag`: Use `PyConfig.buffered_stdio` or `PyConfig_Get("buffered_stdio")` instead.
  - `Py_HashRandomizationFlag`: Use `PyConfig.use_hash_seed` and `PyConfig.hash_seed` or `PyConfig_Get("hash_seed")` instead.
  - `Py_IsolatedFlag`: Use `PyConfig.isolated` or `PyConfig_Get("isolated")` instead.
  - `Py_LegacyWindowsFSEncodingFlag`: Use `PyPreConfig.legacy_windows_fs_encoding` or `PyConfig_Get("legacy_windows_fs_encoding")` instead.
  - `Py_LegacyWindowsStdioFlag`: Use `PyConfig.legacy_windows_stdio` or `PyConfig_Get("legacy_windows_stdio")` instead.
  - `Py_FileSystemDefaultEncoding`, `Py_HasFileSystemDefaultEncoding`: Use `PyConfig.filesystem_encoding` or `PyConfig_Get("filesystem_encoding")` instead.
  - `Py_FileSystemDefaultEncodeErrors`: Use `PyConfig.filesystem_errors` or `PyConfig_Get("filesystem_errors")` instead.
  - `Py_UTF8Mode`: Use `PyPreConfig.utf8_mode` or `PyConfig_Get("utf8_mode")` instead. (see `Py_PreInitialize()`)

  The `Py_InitializeFromConfig()` API should be used with `PyConfig` to set these options. Or `PyConfig_Get()` can be used to get these options at runtime.

Pending removal in Python 3.18

- The following private functions are deprecated and planned for removal in Python 3.18:

- **`_PyBytes_Join()`**: use **`PyBytes_Join()`**.
- **`_PyDict_GetItemStringWithError()`**: use **`PyDict_GetItemStringRef()`**.
- **`_PyDict_Pop()`**: use **`PyDict_Pop()`**.
- **`_PyLong_Sign()`**: use **`PyLong_GetSign()`**.
- **`_PyLong_FromDigits()`** and **`_PyLong_New()`**: use **`PyLongWriter_Create()`**.
- **`_PyThreadState_UncheckedGet()`**: use **`PyThreadState_GetUnchecked()`**.
- **`_PyUnicode_AsString()`**: use **`PyUnicode_AsUTF8()`**.
- **`_PyUnicodeWriter_Init()`**: replace `_PyUnicodeWriter_Init(&writer)` with **`writer = PyUnicodeWriter_Create(0)`**.
- **`_PyUnicodeWriter_Finish()`**: replace `_PyUnicodeWriter_Finish(&writer)` with **`PyUnicodeWriter_Finish(writer)`**.
- **`_PyUnicodeWriter_Dealloc()`**: replace `_PyUnicodeWriter_Dealloc(&writer)` with **`PyUnicodeWriter_Discard(writer)`**.
- **`_PyUnicodeWriter_WriteChar()`**: replace `_PyUnicodeWriter_WriteChar(&writer, ch)` with **`PyUnicodeWriter_WriteChar(writer, ch)`**.
- **`_PyUnicodeWriter_WriteStr()`**: replace `_PyUnicodeWriter_WriteStr(&writer, str)` with **`PyUnicodeWriter_WriteStr(writer, str)`**.
- **`_PyUnicodeWriter_WriteSubstring()`**: replace `_PyUnicodeWriter_WriteSubstring(&writer, str, start, end)` with **`PyUnicodeWriter_WriteSubstring(writer, str, start, end)`**.
- **`_PyUnicodeWriter_WriteASCIIString()`**: replace `_PyUnicodeWriter_WriteASCIIString(&writer, str)` with **`PyUnicodeWriter_WriteASCII(writer, str)`**.
- **`_PyUnicodeWriter_WriteLatin1String()`**: replace `_PyUnicodeWriter_WriteLatin1String(&writer,

    `str)` with `PyUnicodeWriter_WriteUTF8(writer, str)`.
    - `_PyUnicodeWriter_Prepare()`: (no replacement).
    - `_PyUnicodeWriter_PrepareKind()`: (no replacement).
    - `_Py_HashPointer()`: use `Py_HashPointer()`.
    - `_Py_fopen_obj()`: use `Py_fopen()`.

  The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get these new public functions on Python 3.13 and older. (Contributed by Victor Stinner in gh-128863 [https://github.com/python/cpython/issues/128863].)

Pending removal in future versions

Las siguientes API están obsoletas y se eliminarán, aunque actualmente no hay una fecha programada para su eliminación.

- `Py_TPFLAGS_HAVE_FINALIZE`: Innecesario desde Python 3.8.
- `PyErr_Fetch()`: Utilice `PyErr_GetRaisedException()` en su lugar.
- `PyErr_NormalizeException()`: Utilice `PyErr_GetRaisedException()` en su lugar.
- `PyErr_Restore()`: Utilice `PyErr_SetRaisedException()` en su lugar.
- `PyModule_GetFilename()`: Utilice `PyModule_GetFilenameObject()` en su lugar.
- `PyOS_AfterFork()`: Utilice `PyOS_AfterFork_Child()` en su lugar.
- `PySlice_GetIndicesEx()`: Utilice `PySlice_Unpack()` y `PySlice_AdjustIndices()` en su lugar.
- `PyUnicode_READY()`: Innecesario desde Python 3.12
- `PyErr_Display()`: Utilice `PyErr_DisplayException()` en su lugar.
- `_PyErr_ChainExceptions()`: Utilice `_PyErr_ChainExceptions1()` en su lugar.
- Miembro de `PyBytesObject.ob_shash`: llame a `PyObject_Hash()` en su lugar.

- API de almacenamiento local de subprocesos (TLS):
    - `PyThread_create_key()`: Utilice `PyThread_tss_alloc()` en su lugar.
    - `PyThread_delete_key()`: Utilice `PyThread_tss_free()` en su lugar.
    - `PyThread_set_key_value()`: Utilice `PyThread_tss_set()` en su lugar.
    - `PyThread_get_key_value()`: Utilice `PyThread_tss_get()` en su lugar.
    - `PyThread_delete_key_value()`: Utilice `PyThread_tss_delete()` en su lugar.
    - `PyThread_ReInitTLS()`: Innecesario desde Python 3.7.

Glosario

>>>

The default Python prompt of the interactive shell. Often seen for code examples which can be executed interactively in the interpreter.

...

Puede referirse a:

- The default Python prompt of the interactive shell when entering the code for an indented code block, when within a pair of matching left and right delimiters (parentheses, square brackets, curly braces or triple quotes), or after specifying a decorator.

- The three dots form of the Ellipsis object.

clase base abstracta

Las clases base abstractas (ABC, por sus siglas en inglés *Abstract Base Class*) complementan al duck-typing brindando un forma de definir interfaces con técnicas como `hasattr()` que serían confusas o sutilmente erróneas (por ejemplo con magic methods). Las ABC introduce subclases virtuales, las cuales son clases que no heredan desde una clase pero aún así son reconocidas por `isinstance()` y `issubclass()`; vea la documentación del módulo `abc`. Python viene con muchas ABC incorporadas para las estructuras de datos(en el módulo `collections.abc`), números (en el módulo `numbers`) , flujos de datos (en el módulo `io`) , buscadores y cargadores de importaciones (en el módulo `importlib.abc`) . Puede crear sus propios ABCs con el módulo `abc`.

annotate function

A function that can be called to retrieve the annotations of an object. This function is accessible as the **`__annotate__`** attribute of functions, classes, and modules. Annotate functions are a subset of evaluate functions.

anotación

Una etiqueta asociada a una variable, atributo de clase, parámetro de función o valor de retorno, usado por convención como un type hint.

Annotations of local variables cannot be accessed at runtime, but annotations of global variables, class attributes, and functions can be retrieved by calling **`annotationlib.get_annotations()`** on modules, classes, and functions, respectively.

See variable annotation, function annotation, **PEP 484** [https://peps.python.org/pep-0484/], **PEP 526** [https://peps.python.org/pep-0526/], and **PEP 649** [https://peps.python.org/pep-0649/], which describe this functionality. Also see Prácticas recomendadas para las anotaciones for best practices on working with annotations.

argumento

Un valor pasado a una function (o method) cuando se llama a la función. Hay dos clases de argumentos:

- *argumento nombrado*: es un argumento precedido por un identificador (por ejemplo, `nombre=`) en una llamada a una función o pasado como valor en un diccionario precedido por `**`. Por ejemplo `3` y `5` son argumentos nombrados en las llamadas a **`complex()`**:

  ```
  complex(real=3, imag=5)
  complex(**{'real': 3, 'imag': 5})
  ```

- *argumento posicional* son aquellos que no son nombrados. Los argumentos posicionales deben aparecer al principio de una lista de argumentos o ser pasados como elementos de un iterable precedido por `*`. Por ejemplo, `3` y `5` son argumentos posicionales en las siguientes llamadas:

```
complex(3, 5)
complex(*(3, 5))
```

Los argumentos son asignados a las variables locales en el cuerpo de la función. Vea en la sección Invocaciones las reglas que rigen estas asignaciones. Sintácticamente, cualquier expresión puede ser usada para representar un argumento; el valor evaluado es asignado a la variable local.

Vea también el parameter en el glosario, la pregunta frecuente la diferencia entre argumentos y parámetros, y **PEP 362** [https://peps.python.org/pep-0362/].

administrador asincrónico de contexto

Un objeto que controla el entorno visible en un sentencia `async with` al definir los métodos `__aenter__()` y `__aexit__()`. Introducido por **PEP 492** [https://peps.python.org/pep-0492/].

generador asincrónico

Una función que retorna un asynchronous generator iterator. Es similar a una función corrutina definida con `async def` excepto que contiene expresiones `yield` para producir series de variables usadas en un ciclo `async for`.

Usualmente se refiere a una función generadora asincrónica, pero puede referirse a un *iterador generador asincrónico* en ciertos contextos. En aquellos casos en los que el significado no está claro, usar los términos completos evita la ambigüedad.

Una función generadora asincrónica puede contener expresiones `await` así como sentencias `async for`, y `async with`.

iterador generador asincrónico

An object created by an asynchronous generator function.

Este es un asynchronous iterator el cual cuando es llamado usa el método `__anext__()` retornando un objeto a la espera (*awaitable*) el cual ejecutará el cuerpo de la función generadora asincrónica hasta la siguiente expresión `yield`.

Each `yield` temporarily suspends processing, remembering the execution state (including local variables and pending try-statements). When the *asynchronous generator iterator* effectively resumes with another awaitable returned by `__anext__()`, it picks up where it left off. See **PEP 492** [https://peps.python.org/pep-0492/] and **PEP 525** [https://peps.python.org/pep-0525/].

iterable asincrónico

Un objeto, que puede ser usado en una sentencia `async for`. Debe retornar un asynchronous iterator de su método `__aiter__()`. Introducido por **PEP 492** [https://peps.python.org/pep-0492/].

iterador asincrónico

Un objeto que implementa los métodos `__aiter__()` y `__anext__()`. `__anext__()` debe retornar un objeto awaitable. `async for` resuelve los esperables retornados por un método de iterador asincrónico `__anext__()` hasta que lanza una excepción `StopAsyncIteration`. Introducido por **PEP 492** [https://peps.python.org/pep-0492/].

atomic operation

An operation that appears to execute as a single, indivisible step: no other thread can observe it half-done, and its effects become visible all at once. Python does not guarantee that high-level statements are atomic (for example, `x += 1` performs multiple bytecode operations and is not atomic). Atomicity is only guaranteed where explicitly documented. See also race condition and data race.

attached thread state

A thread state that is active for the current OS thread.

When a thread state is attached, the OS thread has access to the full Python C API and can safely invoke the bytecode interpreter.

Unless a function explicitly notes otherwise, attempting to call the C API without an attached thread state will result in a fatal error or undefined behavior. A thread state can be attached and detached explicitly by the user through the C API, or implicitly by the runtime, including during blocking C calls and by the bytecode interpreter in between calls.

On most builds of Python, having an attached thread state implies that the caller holds the GIL for the current interpreter, so only one OS thread can have an attached thread state at a given moment. In free-threaded builds of Python, threads can concurrently hold an attached thread state, allowing for true parallelism of the bytecode interpreter.

atributo

Un valor asociado a un objeto al que se suele hacer referencia por su nombre utilizando expresiones punteadas. Por ejemplo, si un objeto *o* tiene un atributo *a* se referenciaría como *o.a*.

Es posible dar a un objeto un atributo cuyo nombre no sea un identificador definido por Names (identifiers and keywords), por ejemplo usando `setattr()`, si el objeto lo permite. Dicho atributo no será accesible utilizando una expresión con puntos, y en su lugar deberá ser recuperado con `getattr()`.

a la espera

Un objeto que puede utilizarse en una expresión `await`. Puede ser una corutina o un objeto con un método `__await__()`. Véase también **PEP 492** [https://peps.python.org/pep-0492/].

BDFL

Sigla de *Benevolent Dictator For Life*, benevolente dictador vitalicio, es decir Guido van Rossum [https://gvanrossum.github.io/], el creador de Python.

archivo binario

A file object able to read and write bytes-like objects. Examples of binary files are files opened in binary mode (`'rb'`, `'wb'` or `'rb+'`), `sys.stdin.buffer`, `sys.stdout.buffer`, and instances of `io.BytesIO` and `gzip.GzipFile`.

Vea también text file para un objeto archivo capaz de leer y escribir objetos `str`.

referencia prestada

En la API C de Python, una referencia prestada es una referencia a un objeto, donde el código usando el objeto no posee la referencia. Se convierte en un puntero colgante si se destruye el objeto. Por ejemplo, una recolección de basura puede eliminar el último strong reference del objeto y así destruirlo.

Se recomienda llamar a `Py_INCREF()` en la referencia prestada para convertirla en una referencia fuerte in situ, excepto cuando el objeto no se puede destruir antes del último uso de la referencia prestada. La función `Py_NewRef()` se puede utilizar para crear una nueva referencia fuerte.

objetos tipo binarios

Un objeto que soporta Protocolo búfer y puede exportar un búfer C-contiguous. Esto incluye todas los objetos `bytes`, `bytearray`, y `array.array`, así como muchos objetos comunes `memoryview`. Los objetos tipo binarios pueden ser usados para varias operaciones que usan datos binarios; éstas incluyen compresión, salvar a archivos binarios, y enviarlos a través de un socket.

Algunas operaciones necesitan que los datos binarios sean mutables. La documentación frecuentemente se refiere a éstos como «objetos tipo binario de lectura y escritura». Ejemplos de objetos de búfer mutables incluyen a `bytearray` y `memoryview` de la `bytearray`. Otras operaciones que requieren datos binarios almacenados en objetos inmutables («objetos tipo binario de sólo lectura»); ejemplos de éstos incluyen `bytes` y `memoryview` del objeto `bytes`.

bytecode

El código fuente Python es compilado en *bytecode*, la representación interna de un programa python en el intérprete CPython. El *bytecode* también es guardado en caché en los archivos *.pyc* de tal forma que ejecutar el mismo archivo es más fácil la segunda vez (la recompilación desde el código fuente a *bytecode* puede ser evitada). Este «lenguaje intermedio» deberá corren en una virtual machine que ejecute el código de máquina correspondiente a cada *bytecode*. Note que los *bytecodes* no tienen como requisito trabajar en las diversas máquina virtuales de Python, ni de ser estable entre versiones Python.

Una lista de las instrucciones en *bytecode* está disponible en la documentación de el módulo dis.

callable

Un callable es un objeto que puede ser llamado, posiblemente con un conjunto de argumentos (véase argument), con la siguiente sintaxis:

```
callable(argument1, argument2, argumentN)
```

Una function, y por extensión un method, es un callable. Una instancia de una clase que implementa el método `__call__()` también es un callable.

retrollamada

Una función de subrutina que se pasa como un argumento para ejecutarse en algún momento en el futuro.

clase

Una plantilla para crear objetos definidos por el usuario. Las definiciones de clase normalmente contienen definiciones de métodos que operan una instancia de la clase.

variable de clase

Una variable definida en una clase y prevista para ser modificada sólo a nivel de clase (es decir, no en una instancia de la clase).

closure variable

A free variable referenced from a nested scope that is defined in an outer scope rather than being resolved at runtime from the globals or builtin namespaces. May be explicitly defined with the **`nonlocal`** keyword to allow write access, or implicitly defined if the variable is only being read.

For example, in the `inner` function in the following code, both `x` and `print` are free variables, but only `x` is a *closure variable*:

```
def outer():
    x = 0
    def inner():
        nonlocal x
        x += 1
        print(x)
    return inner
```

Due to the **`codeobject.co_freevars`** attribute (which, despite its name, only includes the names of closure variables rather than listing all referenced free variables), the more general free variable term is sometimes used even when the intended meaning is to refer specifically to closure variables.

número complejo

Una extensión del sistema familiar de número reales en el cual los números son expresados como la suma de una parte real y una parte imaginaria. Los números imaginarios son múltiplos de la unidad imaginaria (la raíz cuadrada de `-1`), usualmente escrita como `i` en matemáticas o `j` en ingeniería. Python tiene soporte incorporado para números complejos, los cuales son escritos con la notación mencionada al final.; la parte imaginaria es escrita con un sufijo `j`, por ejemplo, `3+1j`. Para tener acceso a los equivalentes complejos del módulo **`math`** module, use **`cmath`**. El uso de

números complejos es matemática bastante avanzada. Si no le parecen necesarios, puede ignorarlos sin inconvenientes.

concurrency

The ability of a computer program to perform multiple tasks at the same time. Python provides libraries for writing programs that make use of different forms of concurrency. `asyncio` is a library for dealing with asynchronous tasks and coroutines. `threading` provides access to operating system threads and `multiprocessing` to operating system processes. Multi-core processors can execute threads and processes on different CPU cores at the same time (see parallelism).

concurrent modification

When multiple threads modify shared data at the same time. Concurrent modification without proper synchronization can cause race conditions, and might also trigger a data race, data corruption, or both.

context

This term has different meanings depending on where and how it is used. Some common meanings:

- The temporary state or environment established by a context manager via a `with` statement.
- The collection of keyvalue bindings associated with a particular `contextvars.Context` object and accessed via `ContextVar` objects. Also see context variable.
- A `contextvars.Context` object. Also see current context.

context management protocol

The `__enter__()` and `__exit__()` methods called by the `with` statement. See PEP 343 [https://peps.python.org/pep-0343/].

administrador de contextos

An object which implements the context management

protocol and controls the environment seen in a `with` statement. See **PEP 343** [https://peps.python.org/pep-0343/].

variable de contexto

A variable whose value depends on which context is the current context. Values are accessed via `contextvars.ContextVar` objects. Context variables are primarily used to isolate state between concurrent asynchronous tasks.

contiguo

Un búfer es considerado contiguo con precisión si es *C-contiguo* o *Fortran contiguo*. Los búferes cero dimensionales con C y Fortran contiguos. En los arreglos unidimensionales, los ítems deben ser dispuestos en memoria uno siguiente al otro, ordenados por índices que comienzan en cero. En arreglos unidimensionales C-contiguos, el último índice varía más velozmente en el orden de las direcciones de memoria. Sin embargo, en arreglos Fortran contiguos, el primer índice vería más rápidamente.

corrutina

Las corrutinas son una forma más generalizadas de las subrutinas. A las subrutinas se ingresa por un punto y se sale por otro punto. Las corrutinas pueden se iniciadas, finalizadas y reanudadas en muchos puntos diferentes. Pueden ser implementadas con la sentencia `async def`. Vea además **PEP 492** [https://peps.python.org/pep-0492/].

función corrutina

Un función que retorna un objeto coroutine . Una función corrutina puede ser definida con la sentencia `async def`, y puede contener las palabras claves `await`, `async for`, y `async with`. Las mismas son introducidas en **PEP 492** [https://peps.python.org/pep-0492/].

CPython

La implementación canónica del lenguaje de programación Python, como se distribuye en python.org [https://

www.python.org]. El término «CPython» es usado cuando es necesario distinguir esta implementación de otras como *Jython* o *IronPython*.

current context

The context (`contextvars.Context` object) that is currently used by `ContextVar` objects to access (get or set) the values of context variables. Each thread has its own current context. Frameworks for executing asynchronous tasks (see `asyncio`) associate each task with a context which becomes the current context whenever the task starts or resumes execution.

cyclic isolate

A subgroup of one or more objects that reference each other in a reference cycle, but are not referenced by objects outside the group. The goal of the cyclic garbage collector is to identify these groups and break the reference cycles so that the memory can be reclaimed.

data race

A situation where multiple threads access the same memory location concurrently, at least one of the accesses is a write, and the threads do not use any synchronization to control their access. Data races lead to non-deterministic behavior and can cause data corruption. Proper use of locks and other synchronization primitives prevents data races. Note that data races can only happen in native code, but that native code might be exposed in a Python API. See also race condition and thread-safe.

deadlock

A situation in which two or more tasks (threads, processes, or coroutines) wait indefinitely for each other to release resources or complete actions, preventing any from making progress. For example, if thread A holds lock 1 and waits for lock 2, while thread B holds lock 2 and waits for lock 1, both threads will wait indefinitely. In Python this often arises from acquiring multiple locks in conflicting orders or from circular

join/await dependencies. Deadlocks can be avoided by always acquiring multiple locks in a consistent order. See also lock and reentrant.

decorador

Una función que retorna otra función, usualmente aplicada como una función de transformación empleando la sintaxis `@envoltorio`. Ejemplos comunes de decoradores son **`classmethod()`** y **`staticmethod()`**.

La sintaxis del decorador es meramente azúcar sintáctico, las definiciones de las siguientes dos funciones son semánticamente equivalentes:

```
def f(arg):
    ...
f = staticmethod(f)

@staticmethod
def f(arg):
    ...
```

El mismo concepto existe para clases, pero son menos usadas. Vea la documentación de function definitions y class definitions para mayor detalle sobre decoradores.

descriptor

Any object which defines the methods **`__get__()`**, **`__set__()`**, or **`__delete__()`**. When a class attribute is a descriptor, its special binding behavior is triggered upon attribute lookup. Normally, using *a.b* to get, set or delete an attribute looks up the object named *b* in the class dictionary for *a*, but if *b* is a descriptor, the respective descriptor method gets called. Understanding descriptors is a key to a deep understanding of Python because they are the basis for many features including functions, methods, properties, class methods, static methods, and reference to super classes.

Para obtener más información sobre los métodos de los descriptores, consulte Implementando descriptores o Guía práctica de uso de los descriptores.

diccionario

An associative array, where arbitrary keys are mapped to values. The keys can be any object with `__hash__()` and `__eq__()` methods. Called a hash in Perl.

comprensión de diccionarios

Una forma compacta de procesar todos o parte de los elementos en un iterable y retornar un diccionario con los resultados. `results = {n: n ** 2 for n in range(10)}` genera un diccionario que contiene la clave `n` asignada al valor `n ** 2`. Ver Despliegues para listas, conjuntos y diccionarios.

vista de diccionario

Los objetos retornados por los métodos `dict.keys()`, `dict.values()`, y `dict.items()` son llamados vistas de diccionarios. Proveen una vista dinámica de las entradas de un diccionario, lo que significa que cuando el diccionario cambia, la vista refleja éstos cambios. Para forzar a la vista de diccionario a convertirse en una lista completa, use `list(dictview)`. Vea Objetos tipos vista de diccionario.

docstring

A string literal which appears as the first expression in a class, function or module. While ignored when the suite is executed, it is recognized by the compiler and put into the `__doc__` attribute of the enclosing class, function or module. Since it is available via introspection, it is the canonical place for documentation of the object.

tipado de pato

Un estilo de programación que no revisa el tipo del objeto para determinar si tiene la interfaz correcta; en vez de ello, el método o atributo es simplemente llamado o usado («Si se ve como un pato y grazna como un pato, debe ser un pato»). Enfatizando las interfaces en vez de hacerlo con los tipos específicos, un código bien diseñado pues tener mayor flexibilidad permitiendo la sustitución polimórfica. El tipado de pato *duck-typing* evita usar pruebas llamando a `type()` o `isinstance()`. (Nota: si embargo, el tipado de pato puede ser complementado con abstract base classes. En su lugar, generalmente pregunta con `hasattr()` o EAFP.

dunder

An informal short-hand for «double underscore», used when talking about a special method. For example, `__init__` is often pronounced «dunder init».

EAFP

Del inglés *Easier to ask for forgiveness than permission*, es más fácil pedir perdón que pedir permiso. Este estilo de codificación común en Python asume la existencia de claves o atributos válidos y atrapa las excepciones si esta suposición resulta falsa. Este estilo rápido y limpio está caracterizado por muchas sentencias `try` y `except`. Esta técnica contrasta con estilo LBYL usual en otros lenguajes como C.

evaluate function

A function that can be called to evaluate a lazily evaluated attribute of an object, such as the value of type aliases created with the `type` statement.

expresión

Una construcción sintáctica que puede ser evaluada, hasta dar un valor. En otras palabras, una expresión es una acumulación de elementos de expresión tales como literales, nombres, accesos a atributos, operadores o llamadas a funciones, todos ellos retornando valor. A diferencia de otros lenguajes, no toda la sintaxis del lenguaje son expresiones. También hay statements que no pueden ser usadas como expresiones, como la `while`. Las asignaciones también son sentencias, no expresiones.

módulo de extensión

Un módulo escrito en C o C++, usando la API para C de Python para interactuar con el núcleo y el código del usuario.

f-string
f-strings

String literals prefixed with `f` or `F` are commonly called «f-strings» which is short for formatted string literals. See also **PEP 498** [https://peps.python.org/pep-0498/].

objeto archivo

An object exposing a file-oriented API (with methods such as **`read()`** or **`write()`**) to an underlying resource. Depending on the way it was created, a file object can mediate access to a real on-disk file or to another type of storage or communication device (for example standard input/output, in-memory buffers, sockets, pipes, etc.). File objects are also called *file-like objects* or *streams*.

Existen tres categorías de objetos archivo: crudos *raw* archivos binarios, con búfer archivos binarios y archivos de texto. Sus interfaces son definidas en el módulo **`io`**. La forma canónica de crear objetos archivo es usando la función **`open()`**.

objetos tipo archivo

Un sinónimo de file object.

codificación del sistema de archivos y manejador de errores

Controlador de errores y codificación utilizado por Python para decodificar bytes del sistema operativo y codificar Unicode en el sistema operativo.

La codificación del sistema de archivos debe garantizar la decodificación exitosa de todos los bytes por debajo de 128. Si la codificación del sistema de archivos no proporciona esta garantía, las funciones de API pueden lanzar **`UnicodeError`**.

Las funciones **`sys.getfilesystemencoding()`** y **`sys.getfilesystemencodeerrors()`** se pueden utilizar para obtener la codificación del sistema de archivos y el controlador de errores.

La codificación del sistema de archivos y el manejador de errores se configuran al inicio de Python mediante la función `PyConfig_Read()`: consulte los miembros `filesystem_encoding` y `filesystem_errors` de `PyConfig`.

Vea también locale encoding.

buscador

Un objeto que trata de encontrar el loader para el módulo que está siendo importado.

There are two types of finder: meta path finders for use with `sys.meta_path`, and path entry finders for use with `sys.path_hooks`.

See Buscadores y cargadores and `importlib` for much more detail.

división entera a la baja

Una división matemática que se redondea hacia el entero menor más cercano. El operador de la división entera a la baja es `//`. Por ejemplo, la expresión `11 // 4` evalúa `2` a diferencia del `2.75` retornado por la verdadera división de números flotantes. Note que `(-11) // 4` es `-3` porque es `-2.75` redondeado *para abajo*. Ver **PEP 238** [https://peps.python.org/pep-0238/].

free threading

A threading model where multiple threads can run Python bytecode simultaneously within the same interpreter. This is in contrast to the global interpreter lock which allows only one thread to execute Python bytecode at a time. See **PEP 703** [https://peps.python.org/pep-0703/].

free-threaded build

A build of CPython that supports free threading, configured using the `--disable-gil` option before compilation.

See Python support for free threading.

free variable

Formally, as defined in the language execution model, a free variable is any variable used in a namespace which is not a local variable in that namespace. See closure variable for an example. Pragmatically, due to the name of the `codeobject.co_freevars` attribute, the term is also sometimes used as a synonym for closure variable.

función

Una serie de sentencias que retornan un valor al que las llama. También se le puede pasar cero o más argumentos los cuales pueden ser usados en la ejecución de la misma. Vea también parameter, method, y la sección Definiciones de funciones.

anotación de función

Una annotation del parámetro de una función o un valor de retorno.

Las anotaciones de funciones son usadas frecuentemente para indicadores de tipo, por ejemplo, se espera que una función tome dos argumentos de clase `int` y también se espera que retorne dos valores `int`:

```
def sum_two_numbers(a: int, b: int) -> int:
   return a + b
```

La sintaxis de las anotaciones de funciones son explicadas en la sección Definiciones de funciones.

Consulte variable annotation y **PEP 484** [https://peps.python.org/pep-0484/], que describen esta funcionalidad. Consulte también Prácticas recomendadas para las anotaciones para conocer las mejores prácticas sobre cómo trabajar con anotaciones.

__future__

Un future statement, `from __future__ import <feature>`, indica al compilador que compile el módulo actual utilizando una sintaxis o semántica que se convertirá en estándar en una versión futura de Python. El módulo `__future__` documenta los posibles valores de *feature*. Al importar este módulo y evaluar sus variables, puede ver cuándo se agregó por primera vez una nueva característica al lenguaje y cuándo se convertirá (o se convirtió) en la predeterminada:

```
>>> import __future__
>>> __future__.division
_Feature((2, 2, 0, 'alpha', 2), (3, 0, 0, 'alpha',
```

recolección de basura

El proceso de liberar la memoria de lo que ya no está en uso. Python realiza recolección de basura (*garbage collection*) llevando la cuenta de las referencias, y el recogedor de basura cíclico es capaz de detectar y romper las referencias cíclicas. El recogedor de basura puede ser controlado mediante el módulo `gc` .

generador

Una función que retorna un generator iterator. Luce como una función normal excepto que contiene la expresión `yield` para producir series de valores utilizables en un bucle *for* o que pueden ser obtenidas una por una con la función `next()`.

Usualmente se refiere a una función generadora, pero puede referirse a un *iterador generador* en ciertos contextos. En aquellos casos en los que el significado no está claro, usar los términos completos evita la ambigüedad.

iterador generador

Un objeto creado por una función generator.

Each `yield` temporarily suspends processing, remembering the execution state (including local variables and pending try-statements). When the *generator iterator* resumes, it picks up where it left off (in contrast to functions which start fresh on every invocation).

expresión generadora

An expression that returns an iterator. It looks like a normal expression followed by a `for` clause defining a loop variable, range, and an optional `if` clause. The combined expression generates values for an enclosing function:

```
>>> sum(i*i for i in range(10))         # sum of sq
285
```

función genérica

Una función compuesta de muchas funciones que implementan la misma operación para diferentes tipos. Qué implementación deberá ser usada durante la llamada a la misma es determinado por el algoritmo de despacho.

Vea también la entrada de glosario single dispatch, el decorador `functools.singledispatch()`, y **PEP 443** [https://peps.python.org/pep-0443/].

tipos genéricos

Un type que se puede parametrizar; normalmente un container class como `list` o `dict`. Usado para type hints y annotations.

Para más detalles, véase generic alias types, **PEP 483** [https://peps.python.org/pep-0483/], **PEP 484** [https://peps.python.org/pep-0484/], **PEP 585** [https://peps.python.org/pep-0585/], y el módulo `typing`.

GIL

Vea global interpreter lock.

bloqueo global del intérprete

Mecanismo empleado por el intérprete CPython para asegurar que sólo un hilo ejecute el bytecode Python por vez. Esto simplifica la implementación de CPython haciendo que el modelo de objetos (incluyendo algunos críticos como `dict`) están implícitamente a salvo de acceso concurrente. Bloqueando el intérprete completo se simplifica hacerlo multi-hilos, a costa de mucho del paralelismo ofrecido por las máquinas con múltiples procesadores.

Sin embargo, algunos módulos de extensión, tanto estándar como de terceros, están diseñados para liberar el GIL cuando se realizan tareas computacionalmente intensivas como la compresión o el *hashing*. Además, el GIL siempre es liberado cuando se hace entrada/salida.

As of Python 3.13, the GIL can be disabled using the `--disable-gil` build configuration. After building Python with this option, code must be run with `-X gil=0` or after setting the `PYTHON_GIL=0` environment variable. This feature enables improved performance for multi-threaded applications and makes it easier to use multi-core CPUs efficiently. For more details, see **PEP 703** [https://peps.python.org/pep-0703/].

In prior versions of Python's C API, a function might declare that it requires the GIL to be held in order to use it. This refers to having an attached thread state.

global state

Data that is accessible throughout a program, such as module-level variables, class variables, or C static variables in extension modules. In multi-threaded programs, global state shared between threads typically requires synchronization to avoid race conditions and data races.

hash-based pyc

Un archivo cache de *bytecode* que usa el *hash* en vez de usar el tiempo de la última modificación del archivo fuente correspondiente para determinar su validez. Vea Invalidación del código de bytes en caché.

hashable

An object is *hashable* if it has a hash value which never changes during its lifetime (it needs a `__hash__()` method), and can be compared to other objects (it needs an `__eq__()` method). Hashable objects which compare equal must have the same hash value.

Ser *hashable* hace a un objeto utilizable como clave de un diccionario y miembro de un set, porque éstas estructuras de datos usan los valores de hash internamente.

La mayoría de los objetos inmutables incorporados en Python son *hashables*; los contenedores mutables (como las listas o los diccionarios) no lo son; los contenedores inmutables (como tuplas y conjuntos *frozensets*) son *hashables* si sus elementos son *hashables* . Los objetos que son instancias de clases definidas por el usuario son *hashables* por defecto. Todos se comparan como desiguales (excepto consigo mismos), y su valor de hash está derivado de su función `id()`.

IDLE

Un Entorno Integrado de Desarrollo y Aprendizaje para Python. IDLE — Python editor and shell es un editor básico y un entorno de intérprete que se incluye con la distribución estándar de Python.

immortal

Immortal objects are a CPython implementation detail introduced in **PEP 683** [https://peps.python.org/pep-0683/].

If an object is immortal, its reference count is never modified, and therefore it is never deallocated while the interpreter is running. For example, `True` and `None` are immortal in CPython.

Immortal objects can be identified via `sys._is_immortal()`, or via `PyUnstable_IsImmortal()` in the C API.

inmutable

An object with a fixed value. Immutable objects include numbers, strings and tuples. Such an object cannot be altered. A new object has to be created if a different value has to be stored. They play an important role in places where a constant hash value is needed, for example as a key in a dictionary. Immutable objects are inherently thread-safe because their state cannot be modified after creation, eliminating concerns about improperly synchronized concurrent modification.

ruta de importación

Una lista de las ubicaciones (o entradas de ruta) que son revisadas por path based finder al importar módulos. Durante la importación, ésta lista de localizaciones usualmente viene de **`sys.path`**, pero para los subpaquetes también puede incluir al atributo `__path__` del paquete padre.

importar

El proceso mediante el cual el código Python dentro de un módulo se hace alcanzable desde otro código Python en otro módulo.

importador

Un objeto que buscan y lee un módulo; un objeto que es tanto finder como loader.

index

A numeric value that represents the position of an element in a sequence.

In Python, indexing starts at zero. For example, `things[0]` names the *first* element of `things`; `things[1]` names the second one.

In some contexts, Python allows negative indexes for counting from the end of a sequence, and indexing using slices.

See also subscript.

interactivo

Python has an interactive interpreter which means you can enter statements and expressions at the interpreter prompt, immediately execute them and see their results. Just launch `python` with no arguments (possibly by selecting it from your computer's main menu). It is a very powerful way to test out new ideas or inspect modules and packages (remember `help(x)`). For more on interactive mode, see Modo interactivo.

interpretado

Python es un lenguaje interpretado, a diferencia de uno compilado, a pesar de que la distinción puede ser difusa debido al compilador a *bytecode*. Esto significa que los archivos fuente pueden ser corridos directamente, sin crear explícitamente un ejecutable que es corrido luego. Los lenguajes interpretados típicamente tienen ciclos de desarrollo y depuración más cortos que los compilados, sin embargo sus programas suelen correr más lentamente. Vea también interactive.

apagado del intérprete

Cuando se le solicita apagarse, el intérprete Python ingresa a un fase especial en la cual gradualmente libera todos los recursos reservados, como módulos y varias estructuras internas críticas. También hace varias llamadas al recolector de basura. Esto puede disparar la ejecución de código de destructores definidos por el usuario o *weakref callbacks*. El código ejecutado durante la fase de apagado puede encontrar varias excepciones debido a que los recursos que necesita pueden no funcionar más (ejemplos comunes son los módulos de bibliotecas o los artefactos de advertencias *warnings machinery*)

La principal razón para el apagado del intérpreter es que el módulo `__main__` o el script que estaba corriendo termine su ejecución.

iterable

An object capable of returning its members one at a time. Examples of iterables include all sequence types (such as **`list`**, **`str`**, and **`tuple`**) and some non-sequence types like **`dict`**, file objects, and objects of any classes you define with an **`__iter__()`** method or with a **`__getitem__()`** method that implements sequence semantics.

Iterables can be used in a `for` loop and in many other places where a sequence is needed (`zip()`, `map()`, ...). When an iterable object is passed as an argument to the built-in function `iter()`, it returns an iterator for the object. This iterator is good for one pass over the set of values. When using iterables, it is usually not necessary to call `iter()` or deal with iterator objects yourself. The `for` statement does that automatically for you, creating a temporary unnamed variable to hold the iterator for the duration of the loop. See also iterator, sequence, and generator.

iterador

An object representing a stream of data. Repeated calls to the iterator's `__next__()` method (or passing it to the built-in function `next()`) return successive items in the stream. When no more data are available a `StopIteration` exception is raised instead. At this point, the iterator object is exhausted and any further calls to its `__next__()` method just raise `StopIteration` again. Iterators are required to have an `__iter__()` method that returns the iterator object itself so every iterator is also iterable and may be used in most places where other iterables are accepted. One notable exception is code which attempts multiple iteration passes. A container object (such as a `list`) produces a fresh new iterator each time you pass it to the `iter()` function or use it in a `for` loop. Attempting this with an iterator will just return the same exhausted iterator object used in the previous iteration pass, making it appear like an empty container.

Puede encontrar más información en Tipos de iteradores.

Detalles de implementación de CPython: CPython does not consistently apply the requirement that an iterator define `__iter__()`. And also please note that free-threaded CPython does not guarantee thread-safe behavior of iterator operations.

key

A value that identifies an entry in a mapping. See also subscript.

función clave

Una función clave o una función de colación es un invocable que retorna un valor usado para el ordenamiento o clasificación. Por ejemplo, **`locale.strxfrm()`** es usada para producir claves de ordenamiento que se adaptan a las convenciones específicas de ordenamiento de un *locale*.

Cierta cantidad de herramientas de Python aceptan funciones clave para controlar como los elementos son ordenados o agrupados. Incluyendo a **`min()`**, **`max()`**, **`sorted()`**, **`list.sort()`**, **`heapq.merge()`**, **`heapq.nsmallest()`**, **`heapq.nlargest()`**, y **`itertools.groupby()`**.

There are several ways to create a key function. For example. the **`str.casefold()`** method can serve as a key function for case insensitive sorts. Alternatively, a key function can be built from a **`lambda`** expression such as `lambda r: (r[0], r[2])`. Also, **`operator.attrgetter()`**, **`operator.itemgetter()`**, and **`operator.methodcaller()`** are three key function constructors. See the Sorting HOW TO for examples of how to create and use key functions.

argumento nombrado

Vea argument.

lambda

Una función anónima de una línea consistente en un sola expression que es evaluada cuando la función es llamada. La sintaxis para crear una función lambda es `lambda [parameters]: expression`

LBYL

Del inglés *Look before you leap*, «mira antes de saltar». Es un estilo de codificación que prueba explícitamente las condiciones previas antes de hacer llamadas o búsquedas. Este estilo contrasta con la manera EAFP y está caracterizado por la presencia de muchas sentencias **`if`**.

In a multi-threaded environment, the LBYL approach can risk

introducing a race condition between «the looking» and «the leaping». For example, the code, `if key in mapping: return mapping[key]` can fail if another thread removes *key* from *mapping* after the test, but before the lookup. This issue can be solved with locks or by using the EAFP approach. See also thread-safe.

lexical analyzer

Formal name for the *tokenizer*; see token.

lista

A built-in Python sequence. Despite its name it is more akin to an array in other languages than to a linked list since access to elements is $O(1)$.

comprensión de listas

Una forma compacta de procesar todos o parte de los elementos en una secuencia y retornar una lista como resultado. `result = ['{:#04x}'.format(x) for x in range(256) if x % 2 == 0]` genera una lista de cadenas conteniendo números hexadecimales (0x..) entre 0 y 255. La cláusula **`if`** es opcional. Si es omitida, todos los elementos en `range(256)` son procesados.

lock

A synchronization primitive that allows only one thread at a time to access a shared resource. A thread must acquire a lock before accessing the protected resource and release it afterward. If a thread attempts to acquire a lock that is already held by another thread, it will block until the lock becomes available. Python's **`threading`** module provides **`Lock`** (a basic lock) and **`RLock`** (a reentrant lock). Locks are used to prevent race conditions and ensure thread-safe access to shared data. Alternative design patterns to locks exist such as queues, producer/consumer patterns, and thread-local state. See also deadlock, and reentrant.

lock-free

An operation that does not acquire any lock and uses atomic

CPU instructions to ensure correctness. Lock-free operations can execute concurrently without blocking each other and cannot be blocked by operations that hold locks. In free-threaded Python, built-in types like `dict` and `list` provide lock-free read operations, which means other threads may observe intermediate states during multi-step modifications even when those modifications hold the per-object lock.

cargador

An object that loads a module. It must define the **`exec_module()`** and **`create_module()`** methods to implement the `Loader` interface. A loader is typically returned by a finder. See also:

- Buscadores y cargadores
- `importlib.abc.Loader`
- **PEP 302** [https://peps.python.org/pep-0302/]

codificación de la configuración regional

En Unix, es la codificación de la configuración regional LC_CTYPE. Se puede configurar con `locale.setlocale(locale.LC_CTYPE, new_locale)`.

En Windows, es la página de códigos ANSI (por ejemplo, `"cp1252"`).

En Android y VxWorks, Python utiliza `"utf-8"` como codificación regional.

`locale.getencoding()` can be used to get the locale encoding.

Vea también filesystem encoding and error handler.

método mágico

Una manera informal de llamar a un special method.

mapeado

Un objeto contenedor que permite recupero de claves arbitrarias y que implementa los métodos especificados en la `collections.abc.Mapping` o `collections.abc.MutableMapping` abstract base classes. Por ejemplo, `dict`, `collections.defaultdict`, `collections.OrderedDict` y `collections.Counter`.

meta buscadores de ruta

Un finder retornado por una búsqueda de `sys.meta_path`. Los meta buscadores de ruta están relacionados a buscadores de entradas de rutas, pero son algo diferente.

Vea en `importlib.abc.MetaPathFinder` los métodos que los meta buscadores de ruta implementan.

metaclase

La clase de una clase. Las definiciones de clases crean nombres de clase, un diccionario de clase, y una lista de clases base. Las metaclases son responsables de tomar estos tres argumentos y crear la clase. La mayoría de los objetos de un lenguaje de programación orientado a objetos provienen de una implementación por defecto. Lo que hace a Python especial que es posible crear metaclases a medida. La mayoría de los usuario nunca necesitarán esta herramienta, pero cuando la necesidad surge, las metaclases pueden brindar soluciones poderosas y elegantes. Han sido usadas para *loggear* acceso de atributos, agregar seguridad a hilos, rastrear la creación de objetos, implementar *singletons*, y muchas otras tareas.

Más información hallará en Metaclases.

método

Una función que es definida dentro del cuerpo de una clase. Si es llamada como un atributo de una instancia de otra clase, el método tomará el objeto instanciado como su primer argument (el cual es usualmente denominado *self*). Vea function y nested scope.

orden de resolución de métodos

Method Resolution Order is the order in which base classes are searched for a member during lookup. See The Python 2.3

Method Resolution Order for details of the algorithm used by the Python interpreter since the 2.3 release.

módulo

Un objeto que sirve como unidad de organización del código Python. Los módulos tienen espacios de nombres conteniendo objetos Python arbitrarios. Los módulos son cargados en Python por el proceso de importing.

Vea también package.

especificador de módulo

Un espacio de nombres que contiene la información relacionada a la importación usada al leer un módulo. Una instancia de `importlib.machinery.ModuleSpec`.

See also Module specs.

MRO

Vea method resolution order.

mutable

An object with state that is allowed to change during the course of the program. In multi-threaded programs, mutable objects that are shared between threads require careful synchronization to avoid race conditions. See also immutable, thread-safe, and concurrent modification.

tupla nombrada

La denominación «tupla nombrada» se aplica a cualquier tipo o clase que hereda de una tupla y cuyos elementos indexables son también accesibles usando atributos nombrados. Este tipo o clase puede tener además otras capacidades.

Varios tipos incorporados son tuplas nombradas, incluyendo los valores retornados por `time.localtime()` y `os.stat()`. Otro ejemplo es `sys.float_info`:

```
>>> sys.float_info[1]                   # indexed a
1024
```

```
>>> sys.float_info.max_exp            # named fie
1024
>>> isinstance(sys.float_info, tuple)   # kind of t
True
```

Some named tuples are built-in types (such as the above examples). Alternatively, a named tuple can be created from a regular class definition that inherits from `tuple` and that defines named fields. Such a class can be written by hand, or it can be created by inheriting `typing.NamedTuple`, or with the factory function `collections.namedtuple()`. The latter techniques also add some extra methods that may not be found in hand-written or built-in named tuples.

espacio de nombres

El lugar donde la variable es almacenada. Los espacios de nombres son implementados como diccionarios. Hay espacio de nombre local, global, e incorporado así como espacios de nombres anidados en objetos (en métodos). Los espacios de nombres soportan modularidad previniendo conflictos de nombramiento. Por ejemplo, las funciones `builtins.open` y `os.open()` se distinguen por su espacio de nombres. Los espacios de nombres también ayuda a la legibilidad y mantenibilidad dejando claro qué módulo implementa una función. Por ejemplo, escribiendo `random.seed()` o `itertools.islice()` queda claro que éstas funciones están implementadas en los módulos `random` y `itertools`, respectivamente.

paquete de espacios de nombres

A package which serves only as a container for subpackages. Namespace packages may have no physical representation, and specifically are not like a regular package because they have no `__init__.py` file.

Namespace packages allow several individually installable packages to have a common parent package. Otherwise, it is recommended to use a regular package.

For more information, see **PEP 420** [https://peps.python.org/

pep-0420/] and Paquetes de espacio de nombres.

Vea también module.

native code

Code that is compiled to machine instructions and runs directly on the processor, as opposed to code that is interpreted or runs in a virtual machine. In the context of Python, native code typically refers to C, C++, Rust or Fortran code in extension modules that can be called from Python. See also extension module.

alcances anidados

La habilidad de referirse a una variable dentro de una definición encerrada. Por ejemplo, una función definida dentro de otra función puede referir a variables en la función externa. Note que los alcances anidados por defecto sólo funcionan para referencia y no para asignación. Las variables locales leen y escriben sólo en el alcance más interno. De manera semejante, las variables globales pueden leer y escribir en el espacio de nombres global. Con `nonlocal` se puede escribir en alcances exteriores.

clase de nuevo estilo

Old name for the flavor of classes now used for all class objects. In earlier Python versions, only new-style classes could use Python's newer, versatile features like `__slots__`, descriptors, properties, `__getattribute__()`, class methods, and static methods.

non-deterministic

Behavior where the outcome of a program can vary between executions with the same inputs. In multi-threaded programs, non-deterministic behavior often results from race conditions where the relative timing or interleaving of threads affects the result. Proper synchronization using locks and other synchronization primitives helps ensure deterministic behavior.

objeto

Cualquier dato con estado (atributo o valor) y comportamiento definido (métodos). También es la más básica clase base para cualquier new-style class.

optimized scope

A scope where target local variable names are reliably known to the compiler when the code is compiled, allowing optimization of read and write access to these names. The local namespaces for functions, generators, coroutines, comprehensions, and generator expressions are optimized in this fashion. Note: most interpreter optimizations are applied to all scopes, only those relying on a known set of local and nonlocal variable names are restricted to optimized scopes.

optional module

An extension module that is part of the standard library, but may be absent in some builds of CPython, usually due to missing third-party libraries or because the module is not available for a given platform.

See Requirements for optional modules for a list of optional modules that require third-party libraries.

paquete

Un module Python que puede contener submódulos o recursivamente, subpaquetes. Técnicamente, un paquete es un módulo Python con un atributo `__path__`.

Vea también regular package y namespace package.

parallelism

Executing multiple operations at the same time (e.g. on multiple CPU cores). In Python builds with the global interpreter lock (GIL), only one thread runs Python bytecode at a time, so taking advantage of multiple CPU cores typically involves multiple processes (e.g. `multiprocessing`) or native extensions that release the GIL. In free-threaded Python, multiple Python threads can run Python code simultaneously on different cores.

parámetro

Una entidad nombrada en una definición de una function (o método) que especifica un argument (o en algunos casos, varios argumentos) que la función puede aceptar. Existen cinco tipos de argumentos:

- *posicional o nombrado*: especifica un argumento que puede ser pasado tanto como posicional o como nombrado. Este es el tipo por defecto de parámetro, como *foo* y *bar* en el siguiente ejemplo:

  ```
  def func(foo, bar=None): ...
  ```

- *sólo posicional*: especifica un argumento que puede ser pasado sólo por posición. Los parámetros sólo posicionales pueden ser definidos incluyendo un carácter `/` en la lista de parámetros de la función después de ellos, como *posonly1* y *posonly2* en el ejemplo que sigue:

  ```
  def func(posonly1, posonly2, /, positional_or_k
  ```

- *sólo nombrado*: especifica un argumento que sólo puede ser pasado por nombre. Los parámetros sólo por nombre pueden ser definidos incluyendo un parámetro posicional de una sola variable o un simple `` *` `` antes de ellos en la lista de parámetros en la definición de la función, como *kw_only1* y *kw_only2* en el ejemplo siguiente:

  ```
  def func(arg, *, kw_only1, kw_only2): ...
  ```

- *variable posicional*: especifica una secuencia arbitraria de argumentos posicionales que pueden ser brindados (además de cualquier argumento posicional aceptado por otros parámetros). Este parámetro puede ser definido anteponiendo al nombre del parámetro `*`, como a *args* en el siguiente ejemplo:

  ```
  def func(*args, **kwargs): ...
  ```

- *variable nombrado*: especifica que arbitrariamente muchos argumentos nombrados pueden ser brindados (además de cualquier argumento nombrado ya aceptado por cualquier otro parámetro). Este parámetro puede ser definido anteponiendo al nombre del parámetro con `**`, como *kwargs* en el ejemplo precedente.

Los parámetros puede especificar tanto argumentos opcionales como requeridos, así como valores por defecto para algunos argumentos opcionales.

Vea también el glosario de argument, la pregunta respondida en la diferencia entre argumentos y parámetros, la clase `inspect.Parameter`, la sección Definiciones de funciones , y **PEP 362** [https://peps.python.org/pep-0362/].

per-object lock

A lock associated with an individual object instance rather than a global lock shared across all objects. In free-threaded Python, built-in types like `dict` and `list` use per-object locks to allow concurrent operations on different objects while serializing operations on the same object. Operations that hold the per-object lock prevent other locking operations on the same object from proceeding, but do not block lock-free operations.

entrada de ruta

Una ubicación única en el import path que el path based finder consulta para encontrar los módulos a importar.

buscador de entradas de ruta

Un finder retornado por un invocable en `sys.path_hooks` (esto es, un path entry hook) que sabe cómo localizar módulos dada una path entry.

Vea en `importlib.abc.PathEntryFinder` los métodos que los buscadores de entradas de ruta implementan.

gancho a entrada de ruta

A callable on the `sys.path_hooks` list which returns a path entry finder if it knows how to find modules on a specific path entry.

buscador basado en ruta

Uno de los meta buscadores de ruta por defecto que busca un import path para los módulos.

objeto tipo ruta

Un objeto que representa una ruta del sistema de archivos. Un objeto tipo ruta puede ser tanto una `str` como un `bytes` representando una ruta, o un objeto que implementa el protocolo `os.PathLike`. Un objeto que soporta el protocolo `os.PathLike` puede ser convertido a ruta del sistema de archivo de clase `str` o `bytes` usando la función `os.fspath()`; `os.fsdecode()` `os.fsencode()` pueden emplearse para garantizar que retorne respectivamente `str` o `bytes`. Introducido por **PEP 519** [https://peps.python.org/pep-0519/].

PEP

Propuesta de mejora de Python, del inglés *Python Enhancement Proposal*. Un PEP es un documento de diseño que brinda información a la comunidad Python, o describe una nueva capacidad para Python, sus procesos o entorno. Los PEPs deberían dar una especificación técnica concisa y una fundamentación para las capacidades propuestas.

Los PEPs tienen como propósito ser los mecanismos primarios para proponer nuevas y mayores capacidad, para recoger la opinión de la comunidad sobre un tema, y para documentar las decisiones de diseño que se han hecho en Python. El autor del PEP es el responsable de lograr consenso con la comunidad y documentar las opiniones disidentes.

Vea **PEP 1** [https://peps.python.org/pep-0001/].

porción

Un conjunto de archivos en un único directorio (posiblemente guardo en un archivo comprimido *zip*) que contribuye a un espacio de nombres de paquete, como está definido en **PEP 420** [https://peps.python.org/pep-0420/].

argumento posicional

Vea argument.

API provisional

Una API provisoria es aquella que deliberadamente fue excluida de las garantías de compatibilidad hacia atrás de la biblioteca estándar. Aunque no se esperan cambios fundamentales en dichas interfaces, como están marcadas como provisionales, los cambios incompatibles hacia atrás (incluso remover la misma interfaz) podrían ocurrir si los desarrolladores principales lo estiman. Estos cambios no se hacen gratuitamente – solo ocurrirán si fallas fundamentales y serias son descubiertas que no fueron vistas antes de la inclusión de la API.

Incluso para APIs provisorias, los cambios incompatibles hacia atrás son vistos como una «solución de último recurso» - se intentará todo para encontrar una solución compatible hacia atrás para los problemas identificados.

Este proceso permite que la biblioteca estándar continúe evolucionando con el tiempo, sin bloquearse por errores de diseño problemáticos por períodos extensos de tiempo. Vea **PEP 411** [https://peps.python.org/pep-0411/] para más detalles.

paquete provisorio

Vea provisional API.

Python 3000

Apodo para la fecha de lanzamiento de Python 3.x (acuñada en un tiempo cuando llegar a la versión 3 era algo distante en el futuro.) También se lo abrevió como *Py3k*.

Pythónico

Una idea o pieza de código que sigue ajustadamente la convenciones idiomáticas comunes del lenguaje Python, en vez de implementar código usando conceptos comunes a otros lenguajes. Por ejemplo, una convención común en Python es hacer bucles sobre todos los elementos de un iterable con la sentencia **`for`**. Muchos otros lenguajes no tienen este tipo de construcción, así que los que no están familiarizados con Python podrían usar contadores numéricos:

```
for i in range(len(food)):
    print(food[i])
```

En contraste, un método Pythónico más limpio:

```
for piece in food:
    print(piece)
```

nombre calificado

Un nombre con puntos mostrando la ruta desde el alcance global del módulo a la clase, función o método definido en dicho módulo, como se define en **PEP 3155** [https://peps.python.org/pep-3155/]. Para las funciones o clases de más alto nivel, el nombre calificado es el igual al nombre del objeto:

```
>>> class C:
...     class D:
...         def meth(self):
...             pass
...
>>> C.__qualname__
'C'
>>> C.D.__qualname__
'C.D'
>>> C.D.meth.__qualname__
'C.D.meth'
```

Cuando es usado para referirse a los módulos, *nombre completamente calificado* significa la ruta con puntos completo al módulo, incluyendo cualquier paquete padre, por ejemplo, `email.mime.text`:

```
>>> import email.mime.text
>>> email.mime.text.__name__
'email.mime.text'
```

race condition

A condition of a program where the behavior depends on the relative timing or ordering of events, particularly in multi-threaded programs. Race conditions can lead to non-deterministic behavior and bugs that are difficult to reproduce. A data race is a specific type of race condition involving unsynchronized access to shared memory. The LBYL coding style is particularly susceptible to race conditions in multi-threaded code. Using locks and other synchronization primitives helps prevent race conditions.

contador de referencias

The number of references to an object. When the reference count of an object drops to zero, it is deallocated. Some objects are immortal and have reference counts that are never modified, and therefore the objects are never deallocated. Reference counting is generally not visible to Python code, but it is a key element of the CPython implementation. Programmers can call the **`sys.getrefcount()`** function to return the reference count for a particular object.

In CPython, reference counts are not considered to be stable or well-defined values; the number of references to an object, and how that number is affected by Python code, may be different between versions.

paquete regular

Un package tradicional, como aquellos con un directorio conteniendo el archivo `__init__.py`.

Vea también namespace package.

reentrant

A property of a function or lock that allows it to be called or acquired multiple times by the same thread without causing errors or a deadlock.

For functions, reentrancy means the function can be safely called again before a previous invocation has completed, which is important when functions may be called recursively or from signal handlers. Thread-unsafe functions may be non-deterministic if they're called reentrantly in a multithreaded program.

For locks, Python's `threading.RLock` (reentrant lock) is reentrant, meaning a thread that already holds the lock can acquire it again without blocking. In contrast, `threading.Lock` is not reentrant - attempting to acquire it twice from the same thread will cause a deadlock.

See also lock and deadlock.

REPL

An acronym for the «read–eval–print loop», another name for the interactive interpreter shell.

__slots__

Es una declaración dentro de una clase que ahorra memoria predeclarando espacio para las atributos de la instancia y eliminando diccionarios de la instancia. Aunque es popular, esta técnica es algo dificultosa de lograr correctamente y es mejor reservarla para los casos raros en los que existen grandes cantidades de instancias en aplicaciones con uso crítico de memoria.

secuencia

An iterable which supports efficient element access using integer indices via the `__getitem__()` special method and defines a `__len__()` method that returns the length of the sequence. Some built-in sequence types are `list`, `str`, `tuple`, and `bytes`. Note that `dict` also supports `__getitem__()` and `__len__()`, but is considered a mapping rather than a sequence because the lookups use arbitrary hashable keys rather than integers.

The `collections.abc.Sequence` abstract base class defines a much richer interface that goes beyond just

`__getitem__()` and `__len__()`, adding `count()`, `index()`, `__contains__()`, and `__reversed__()`. Types that implement this expanded interface can be registered explicitly using `register()`. For more documentation on sequence methods generally, see Common Sequence Operations.

comprensión de conjuntos

Una forma compacta de procesar todos o parte de los elementos en un iterable y retornar un conjunto con los resultados. `results = {c for c in 'abracadabra' if c not in 'abc'}` genera el conjunto de cadenas `{'r', 'd'}`. Ver Despliegues para listas, conjuntos y diccionarios.

despacho único

Una forma de despacho de una generic function donde la implementación es elegida a partir del tipo de un sólo argumento.

rebanada

An object of type `slice`, used to describe a portion of a sequence. A slice object is created when using the slicing form of subscript notation, with colons inside square brackets, such as in `variable_name[1:3:5]`.

soft deprecated

A soft deprecated API should not be used in new code, but it is safe for already existing code to use it. The API remains documented and tested, but will not be enhanced further.

Soft deprecation, unlike normal deprecation, does not plan on removing the API and will not emit warnings.

See PEP 387: Soft Deprecation [https://peps.python.org/pep-0387/#soft-deprecation].

método especial

Un método que es llamado implícitamente por Python cuando ejecuta ciertas operaciones en un tipo, como la adición. Estos métodos tienen nombres que comienzan y terminan con doble barra baja. Los métodos especiales están documentados en Nombres especiales de método.

standard library

The collection of packages, modules and extension modules distributed as a part of the official Python interpreter package. The exact membership of the collection may vary based on platform, available system libraries, or other criteria. Documentation can be found at La biblioteca estándar de Python.

See also `sys.stdlib_module_names` for a list of all possible standard library module names.

sentencia

Una sentencia es parte de un conjunto (un «bloque» de código). Una sentencia tanto es una expression como alguna de las varias sintaxis usando una palabra clave, como `if`, `while` o `for`.

static type checker

An external tool that reads Python code and analyzes it, looking for issues such as incorrect types. See also type hints and the `typing` module.

stdlib

An abbreviation of standard library.

referencia fuerte

En la API de C de Python, una referencia fuerte es una referencia a un objeto que es propiedad del código que mantiene la referencia. La referencia fuerte se toma llamando a `Py_INCREF()` cuando se crea la referencia y se libera con `Py_DECREF()` cuando se elimina la referencia.

La función `Py_NewRef()` se puede utilizar para crear una referencia fuerte a un objeto. Por lo general, se debe llamar a la función `Py_DECREF()` en la referencia fuerte antes de salir del alcance de la referencia fuerte, para evitar filtrar una referencia.

Consulte también borrowed reference.

subscript

The expression in square brackets of a subscription expression, for example, the `3` in `items[3]`. Usually used to select an element of a container. Also called a key when subscripting a mapping, or an index when subscripting a sequence.

synchronization primitive

A basic building block for coordinating (synchronizing) the execution of multiple threads to ensure thread-safe access to shared resources. Python's **`threading`** module provides several synchronization primitives including **`Lock`**, **`RLock`**, **`Semaphore`**, **`Condition`**, **`Event`**, and **`Barrier`**. Additionally, the **`queue`** module provides multi-producer, multi-consumer queues that are especially useful in multithreaded programs. These primitives help prevent race conditions and coordinate thread execution. See also lock.

t-string
t-strings

String literals prefixed with `t` or `T` are commonly called «t-strings» which is short for template string literals.

codificación de texto

Una cadena de caracteres en Python es una secuencia de puntos de código Unicode (en el rango `U+0000`–`U+10FFFF`). Para almacenar o transferir una cadena de caracteres, es necesario serializarla como una secuencia de bytes.

La serialización de una cadena de caracteres en una secuencia de bytes se conoce como «codificación», y la recreación de la cadena de caracteres a partir de la secuencia de bytes se conoce como «decodificación».

Existe una gran variedad de serializaciones de texto codecs, que se denominan colectivamente «codificaciones de texto».

archivo de texto

Un file object capaz de leer y escribir objetos `str`. Frecuentemente, un archivo de texto también accede a un flujo de datos binario y maneja automáticamente el text encoding. Ejemplos de archivos de texto que son abiertos en modo texto (`'r'` o `'w'`), `sys.stdin`, `sys.stdout`, y las instancias de `io.StringIO`.

Vea también binary file por objeto de archivos capaces de leer y escribir objeto tipo binario.

thread state

The information used by the CPython runtime to run in an OS thread. For example, this includes the current exception, if any, and the state of the bytecode interpreter.

Each thread state is bound to a single OS thread, but threads may have many thread states available. At most, one of them may be attached at once.

An attached thread state is required to call most of Python's C API, unless a function explicitly documents otherwise. The bytecode interpreter only runs under an attached thread state.

Each thread state belongs to a single interpreter, but each interpreter may have many thread states, including multiple for the same OS thread. Thread states from multiple interpreters may be bound to the same thread, but only one can be attached in that thread at any given moment.

See Thread State and the Global Interpreter Lock for more information.

thread-safe

A module, function, or class that behaves correctly when used by multiple threads concurrently. Thread-safe code uses appropriate synchronization primitives like locks to protect shared mutable state, or is designed to avoid shared mutable state entirely. In the free-threaded build, built-in types like **`dict`**, **`list`**, and **`set`** use internal locking to make many operations thread-safe, although thread safety is not necessarily guaranteed. Code that is not thread-safe may experience race conditions and data races when used in multi-threaded programs.

token

A small unit of source code, generated by the lexical analyzer (also called the *tokenizer*). Names, numbers, strings, operators, newlines and similar are represented by tokens.

The **`tokenize`** module exposes Python's lexical analyzer. The **`token`** module contains information on the various types of tokens.

cadena con triple comilla

Una cadena que está enmarcada por tres instancias de comillas (») o apostrofes ("). Aunque no brindan ninguna funcionalidad que no está disponible usando cadenas con comillas simple, son útiles por varias razones. Permiten incluir comillas simples o dobles sin escapar dentro de las cadenas y pueden abarcar múltiples líneas sin el uso de caracteres de continuación, haciéndolas particularmente útiles para escribir docstrings.

tipo

The type of a Python object determines what kind of object it is; every object has a type. An object's type is accessible as its **`__class__`** attribute or can be retrieved with `type(obj)`.

alias de tipos

Un sinónimo para un tipo, creado al asignar un tipo a un identificador.

Los alias de tipos son útiles para simplificar los indicadores de tipo. Por ejemplo:

```
def remove_gray_shades(
        colors: list[tuple[int, int, int]]) -> list
    pass
```

podría ser más legible así:

```
Color = tuple[int, int, int]

def remove_gray_shades(colors: list[Color]) -> list
    pass
```

Vea `typing` y **PEP 484** [https://peps.python.org/pep-0484/], que describen esta funcionalidad.

indicador de tipo

Una annotation que especifica el tipo esperado para una variable, un atributo de clase, un parámetro para una función o un valor de retorno.

Type hints are optional and are not enforced by Python but they are useful to static type checkers. They can also aid IDEs with code completion and refactoring.

Los indicadores de tipo de las variables globales, atributos de clase, y funciones, no de variables locales, pueden ser accedidos usando `typing.get_type_hints()`.

Vea `typing` y **PEP 484** [https://peps.python.org/pep-0484/], que describen esta funcionalidad.

saltos de líneas universales

Una manera de interpretar flujos de texto en la cual son reconocidos como finales de línea todas siguientes formas: la convención de Unix para fin de línea `'\n'`, la convención de Windows `'\r\n'`, y la vieja convención de Macintosh `'\r'`. Vea **PEP 278** [https://peps.python.org/pep-0278/] y **PEP 3116** [https://peps.python.org/pep-3116/], además de `bytes.splitlines()` para usos adicionales.

anotación de variable

Una annotation de una variable o un atributo de clase.

Cuando se anota una variable o un atributo de clase, la asignación es opcional:

```
class C:
    field: 'annotation'
```

Las anotaciones de variables son frecuentemente usadas para type hints: por ejemplo, se espera que esta variable tenga valores de clase `int`:

```
count: int = 0
```

La sintaxis de la anotación de variables está explicada en la sección Declaraciones de asignación anotadas.

Consulte function annotation, **PEP 484** [https://peps.python.org/pep-0484/] y **PEP 526** [https://peps.python.org/pep-0526/], que describen esta funcionalidad. Consulte también Prácticas recomendadas para las anotaciones para conocer las mejores prácticas sobre cómo trabajar con anotaciones.

entorno virtual

Un entorno cooperativamente aislado de ejecución que permite a los usuarios de Python y a las aplicaciones instalar y actualizar paquetes de distribución de Python sin interferir con el comportamiento de otras aplicaciones de Python en el mismo sistema.

Vea también `venv`.

máquina virtual

Una computadora definida enteramente por software. La máquina virtual de Python ejecuta el bytecode generado por el compilador de *bytecode*.

walrus operator

A light-hearted way to refer to the assignment expression operator `:=` because it looks a bit like a walrus if you turn your head.

Zen de Python

Un listado de los principios de diseño y la filosofía de Python que son útiles para entender y usar el lenguaje. El listado puede encontrarse ingresando «`import this`» en la consola interactiva.

About this documentation

Python's documentation is generated from reStructuredText [https://docutils.sourceforge.io/rst.html] sources using Sphinx [https://www.sphinx-doc.org/], a documentation generator originally created for Python and now maintained as an independent project.

El desarrollo de la documentación y su cadena de herramientas es un esfuerzo enteramente voluntario, al igual que Python. Si tu quieres contribuir, por favor revisa la página Lidiar con errores para más información de cómo hacerlo. Los nuevos voluntarios son siempre bienvenidos!

Agradecemos a:

- Fred L. Drake, Jr., the creator of the original Python documentation toolset and author of much of the content;
- el proyecto Docutils [https://docutils.sourceforge.io/] para creación de reStructuredText y la suite Docutils;
- Fredrik Lundh por su proyecto Referencia Alternativa de Python del que Sphinx obtuvo muchas buenas ideas.

Contributors to the Python documentation

Muchas personas han contribuido para el lenguaje de Python, la librería estándar de Python, y la documentación de Python. Revisa Misc/ACKS [https://github.com/python/cpython/tree/3.14/Misc/ACKS] la distribución de Python para una lista parcial de contribuidores.

Es solamente con la aportación y contribuciones de la comunidad de Python que Python tiene tan fantástica documentación – Muchas gracias!

Lidiar con errores

Python es un lenguaje de programación maduro que ha establecido una reputación de estabilidad. Con el fin de mantener esta reputación, a los desarrolladores les gustaría conocer cualquier anomalía que encuentre en Python.

It can be sometimes faster to fix bugs yourself and contribute patches to Python as it streamlines the process and involves fewer people. Learn how to contribute.

Documentación de errores

If you find a bug in this documentation or would like to propose an improvement, please submit a bug report on the issue tracker. If you have a suggestion on how to fix it, include that as well.

If the bug or suggested improvement concerns the translation of this documentation, submit the report to the translation's repository [https://github.com/python/python-docs-es] instead.

También puede abrir una discusión en nuestro foro de Documentación de Discourse [https://discuss.python.org/c/documentation/26].

If you find a bug in the theme (HTML / CSS / JavaScript) of the documentation, please submit a bug report on the python-doc-theme issue tracker [https://github.com/python/python-docs-theme].

Ver también

Documentación de errores
Una lista de errores (*bugs*) que ha sido enviada al issue tracker (sistema de seguimiento de incidentes) de Python.

Seguimiento de incidencias [https://devguide.python.org/tracker/]

Resumen general del proceso necesario para reportar una mejora en el tracker.

Ayudar con la documentación [https://devguide.python.org/docquality/#helping-with-documentation]

Guía detallada para gente interesada en contribuir a la documentación de Python.

Documentation Translations [https://devguide.python.org/documentation/translating/]

Una lista de páginas de GitHub para la traducción de documentación y sus contactos principales.

Utilizar el issue tracker de Python

Los informes de incidencias de Python deben enviarse a través del gestor de incidencias de GitHub (https://github.com/python/cpython/issues). El gestor de incidencias de GitHub ofrece un formulario web que permite introducir la información pertinente y enviarla a los desarrolladores.

El primer paso para completar un informe es determinar si el problema ya ha sido descrito previamente. La ventaja de hacer esto, aparte de ahorrar tiempo a los desarrolladores, es que te permite aprender qué cambios se han realizado para repararlo; puede que el problema se haya reparado para la siguiente versión, o que sea necesaria información adicional (en ese caso, ¡te invitamos a incluirla si puedes!). Para hacerlo, busca en la base de datos de errores usando la zona de búsqueda al principio de esta página.

Si el problema del que informa no está ya en la lista, inicie sesión en GitHub. Si aún no tienes una cuenta en GitHub, crea una nueva a través del enlace «Regístrate». No es posible enviar un informe de error de forma anónima.

Ahora que has iniciado sesión, puedes enviar un informe de error. Haz clic en el botón «New issue» de la barra superior para notificar un nuevo informe de error.

El formulario de envío tiene dos campos: «Title» y «Comment».

En el campo «Title», introduzca una descripción *muy* breve del problema; menos de diez palabras es suficiente.

En el campo «Comment», describe el problema con detalle, incluyendo qué esperabas que ocurriera y que ha sucedido en realidad. Asegúrate de incluir si cualquier módulo de extensión está involucrado, y qué plataformas de hardware y software estás usando (incluyendo las versiones correspondientes).

Cada informe de error será asignado a un desarrollador que determinará qué es necesario hacer para corregir el problema. Recibirás una actualización cada vez que se tome una medida al respecto.

Ver también

Cómo informar de errores de manera efectiva [https://www.chiark.greenend.org.uk/~sgtatham/bugs.html]
Artículo que detalla cómo crear un informe de errores útil. Describe qué tipo de información es útil y por qué lo es.

Bug Writing Guidelines [https://bugzilla.mozilla.org/page.cgi?id=bug-writing.html]
Información sobre cómo escribir un buen informe de errores. Parte de esta información es específica al proyecto Mozilla, pero en general describe buenas prácticas.

Para empezar a contribuir en Python

Más allá de informar de los errores que puedas encontrar, te damos la bienvenida a enviar parches para corregirlos. Puedes encontrar más información en cómo empezar parcheando Python en la Python Developer's Guide [https://devguide.python.org/]. Si tienes preguntas, el core-mentorship mailing list [https://mail.python.org/mailman3/lists/core-mentorship.python.org/] es un agradable lugar para obtener respuestas a cualquiera y a todas las preguntas pertenecientes al proceso de corrección de problemas en Python.

Derechos de autor

Python y esta documentación es:

Copyright © 2001 Python Software Foundation. All rights reserved.

Derechos de autor © 2000 BeOpen.com. Todos los derechos reservados.

Derechos de autor © 1995-2000 Corporation for National Research Initiatives. Todos los derechos reservados.

Derechos de autor © 1991-1995 Stichting Mathematisch Centrum. Todos los derechos reservados.

Consulte Historia y Licencia para obtener información completa sobre licencias y permisos.

Historia y Licencia

Historia del software

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see https://www.cwi.nl) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see https://www.cnri.reston.va.us) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations, which became Zope Corporation. In 2001, the Python Software Foundation (PSF, see https://www.python.org/psf/) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation was a sponsoring member of the PSF.

All Python releases are Open Source (see https://opensource.org for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

Versión	**Derivado de**	**Año**	**Propietario**	**GPL compatible? (1)**
0.9.0 hasta 1.2	n/a	1991-1995	CWI	sí
1.3 hasta 1.5.2	1.2	1995-1999	CNRI	sí
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	sí (2)
2.1	2.0+1.6.1	2001	PSF	no

2.0.1	2.0+1.6.1	2001	PSF	sí
2.1.1	2.1+2.0.1	2001	PSF	sí
2.1.2	2.1.1	2002	PSF	sí
2.1.3	2.1.2	2002	PSF	sí
2.2 y superior	2.1.1	2001-ahora	PSF	sí

Nota

1. GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don't.
2. According to Richard Stallman, 1.6.1 is not GPL-compatible, because its license has a choice of law clause. According to CNRI, however, Stallman's lawyer has told CNRI's lawyer that 1.6.1 is «not incompatible» with the GPL.

Gracias a los muchos voluntarios externos que han trabajado bajo la dirección de Guido para hacer posibles estos lanzamientos.

Términos y condiciones para acceder o usar Python

Python software and documentation are licensed under the Python Software Foundation License Version 2.

Starting with Python 3.8.6, examples, recipes, and other code in the documentation are dual licensed under the PSF License Version 2 and the Zero-Clause BSD license.

Parte del software incorporado en Python está bajo diferentes licencias. Las licencias se enumeran con el código correspondiente a esa licencia. Consulte Licencias y reconocimientos para software incorporado para obtener una lista incompleta de estas licencias.

PYTHON SOFTWARE FOUNDATION LICENSE VERSION

2

```
1. This LICENSE AGREEMENT is between the Python Software
   the Individual or Organization ("Licensee") accessing
   software ("Python") in source or binary form and its

2. Subject to the terms and conditions of this License A
   grants Licensee a nonexclusive, royalty-free, world-w
   analyze, test, perform and/or display publicly, prepa
   distribute, and otherwise use Python alone or in any
   version, provided, however, that PSF's License Agreem
   copyright, i.e., "Copyright © 2001 Python Software Fo
   Reserved" are retained in Python alone or in any deri
   prepared by Licensee.

3. In the event Licensee prepares a derivative work that
   incorporates Python or any part thereof, and wants to
   derivative work available to others as provided herei
   agrees to include in any such work a brief summary of

4. PSF is making Python available to Licensee on an "AS
   PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS O
   EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAI
   WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTIC
   USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHT

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USER
   FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, O
   THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate u
   its terms and conditions.

7. Nothing in this License Agreement shall be deemed to
   of agency, partnership, or joint venture between PSF
   Agreement does not grant permission to use PSF tradem
   trademark sense to endorse or promote products or ser
   third party.

8. By copying, installing or otherwise using Python, Lic
   to be bound by the terms and conditions of this Licen
```

ACUERDO DE LICENCIA DE BEOPEN.COM PARA PYTHON 2.0

ACUERDO DE LICENCIA DE CÓDIGO ABIERTO DE BEOPEN PYTHON VERSIÓN 1

```
1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen
   160 Saratoga Avenue, Santa Clara, CA 95051, and the I
   ("Licensee") accessing and otherwise using this softw
   form and its associated documentation ("the Software"

2. Subject to the terms and conditions of this BeOpen Py
   BeOpen hereby grants Licensee a non-exclusive, royalt
   to reproduce, analyze, test, perform and/or display p
   works, distribute, and otherwise use the Software alo
   version, provided, however, that the BeOpen Python Li
   Software, alone or in any derivative version prepared

3. BeOpen is making the Software available to Licensee o
   BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRES
   EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISC
   WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTIC
   USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER U
   ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR
   MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVA
   ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate u
   its terms and conditions.

6. This License Agreement shall be governed by and inter
```

by the law of the State of California, excluding conf
Nothing in this License Agreement shall be deemed to
agency, partnership, or joint venture between BeOpen
Agreement does not grant permission to use BeOpen tra
trademark sense to endorse or promote products or ser
third party. As an exception, the "BeOpen Python" lo
http://www.pythonlabs.com/logos.html may be used acco
granted on that web page.

7. By copying, installing or otherwise using the softwar
bound by the terms and conditions of this License Agr

ACUERDO DE LICENCIA CNRI PARA PYTHON 1.6.1

1. This LICENSE AGREEMENT is between the Corporation for
Initiatives, having an office at 1895 Preston White D
("CNRI"), and the Individual or Organization ("Licens
otherwise using Python 1.6.1 software in source or bi
associated documentation.

2. Subject to the terms and conditions of this License A
grants Licensee a nonexclusive, royalty-free, world-w
analyze, test, perform and/or display publicly, prepa
distribute, and otherwise use Python 1.6.1 alone or i
provided, however, that CNRI's License Agreement and
i.e., "Copyright © 1995-2001 Corporation for National
Rights Reserved" are retained in Python 1.6.1 alone o
prepared by Licensee. Alternately, in lieu of CNRI's
Licensee may substitute the following text (omitting
is made available subject to the terms and conditions
Agreement. This Agreement together with Python 1.6.1
internet using the following unique, persistent ident
1895.22/1013. This Agreement may also be obtained fr
internet using the following URL: http://hdl.handle.n

3. In the event Licensee prepares a derivative work that
incorporates Python 1.6.1 or any part thereof, and wa
work available to others as provided herein, then Lic

```
   include in any such work a brief summary of the chang

4. CNRI is making Python 1.6.1 available to Licensee on
   MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IM
   BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY R
   OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURP
   PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USE
   ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.
   THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate u
   its terms and conditions.

7. This License Agreement shall be governed by the feder
   law of the United States, including without limitatio
   law, and, to the extent such U.S. federal law does no
   Commonwealth of Virginia, excluding Virginia's confli
   Notwithstanding the foregoing, with regard to derivat
   1.6.1 that incorporate non-separable material that wa
   under the GNU General Public License (GPL), the law o
   Virginia shall govern this License Agreement only as
   with respect to Paragraphs 4, 5, and 7 of this Licens
   this License Agreement shall be deemed to create any
   partnership, or joint venture between CNRI and Licens
   does not grant permission to use CNRI trademarks or t
   sense to endorse or promote products or services of L
   party.

8. By clicking on the "ACCEPT" button where indicated, o
   or otherwise using Python 1.6.1, Licensee agrees to b
   conditions of this License Agreement.
```

ACUERDO DE LICENCIA CWI PARA PYTHON 0.9.0 HASTA 1.2

```
Copyright © 1991 - 1995, Stichting Mathematisch Centrum
Netherlands.  All rights reserved.

Permission to use, copy, modify, and distribute this sof
documentation for any purpose and without fee is hereby
the above copyright notice appear in all copies and that
notice and this permission notice appear in supporting d
the name of Stichting Mathematisch Centrum or CWI not be
publicity pertaining to distribution of the software wit
prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTAB
EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR
OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESUL
DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLI
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
SOFTWARE.
```

ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON DOCUMENTATION

```
Permission to use, copy, modify, and/or distribute this
purpose with or without fee is hereby granted.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIM
REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES
AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSO
LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CO
OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION W
PERFORMANCE OF THIS SOFTWARE.
```

Licencias y reconocimientos para software incorporado

Esta sección es una lista incompleta, pero creciente, de licencias y reconocimientos para software de terceros incorporado en la distribución de Python.

Mersenne Twister

La extensión C **`_random`** subyacente al módulo **`random`** incluye código basado en una descarga de http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html. Los siguientes son los comentarios textuales del código original:

```
A C-program for MT19937, with initialization improved 2
Coded by Takuji Nishimura and Makoto Matsumoto.

Before using, initialize the state by using init_genran
or init_by_array(init_key, key_length).

Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji N
All rights reserved.

Redistribution and use in source and binary forms, with
modification, are permitted provided that the following
are met:

 1. Redistributions of source code must retain the abov
    notice, this list of conditions and the following di

 2. Redistributions in binary form must reproduce the ab
    notice, this list of conditions and the following di
    documentation and/or other materials provided with t

 3. The names of its contributors may not be used to en
    products derived from this software without specifi
    permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDIN
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AN
A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDEN
```

```
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TOR
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE U
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAM


Any feedback is very welcome.
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
email: m-mat @ math.sci.hiroshima-u.ac.jp (remove space)
```

Sockets

El módulo `socket` usa las funciones, **`getaddrinfo()`**, y **`getnameinfo()`**, que están codificadas en archivos fuente separados del Proyecto WIDE, http://www.wide.ad.jp /.

```
Copyright (C) 1995, 1996, 1997, and 1998 WIDE Project.
All rights reserved.

Redistribution and use in source and binary forms, with
modification, are permitted provided that the following
are met:
1. Redistributions of source code must retain the above
   notice, this list of conditions and the following dis
2. Redistributions in binary form must reproduce the abo
   notice, this list of conditions and the following dis
   documentation and/or other materials provided with th
3. Neither the name of the project nor the names of its
   may be used to endorse or promote products derived fr
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE PROJECT AND CONTRIBUTOR
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LI
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
ARE DISCLAIMED.  IN NO EVENT SHALL THE PROJECT OR CONTRI
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF S
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER I
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) A
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
SUCH DAMAGE.
```

Servicios de socket asincrónicos

Los módulos **`test.support.asynchat`** y **`test.support.asyncore`** contienen el siguiente aviso:

```
Copyright 1996 by Sam Rushing

                        All Rights Reserved

Permission to use, copy, modify, and distribute this sof
its documentation for any purpose and without fee is her
granted, provided that the above copyright notice appear
copies and that both that copyright notice and this perm
notice appear in supporting documentation, and that the
Rushing not be used in advertising or publicity pertaini
distribution of the software without specific, written p
permission.

SAM RUSHING DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
NO EVENT SHALL SAM RUSHING BE LIABLE FOR ANY SPECIAL, IN
CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTIN
OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRAC
NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR I
CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

Gestión de cookies

El módulo `http.cookies` contiene el siguiente aviso:

```
Copyright 2000 by Timothy O'Malley <timo@alum.mit.edu>

               All Rights Reserved
```

```
Permission to use, copy, modify, and distribute this sof
and its documentation for any purpose and without fee is
granted, provided that the above copyright notice appear
copies and that both that copyright notice and this perm
notice appear in supporting documentation, and that the
Timothy O'Malley  not be used in advertising or publicit
pertaining to distribution of the software without speci
prior permission.

Timothy O'Malley DISCLAIMS ALL WARRANTIES WITH REGARD TO
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTAB
AND FITNESS, IN NO EVENT SHALL Timothy O'Malley BE LIABL
ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DA
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TO
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

Seguimiento de ejecución

El módulo `trace` contiene el siguiente aviso:

```
portions copyright 2001, Autonomous Zones Industries, In
err...  reserved and offered to the public under the ter
Python 2.2 license.
Author: Zooko O'Whielacronx
http://zooko.com/
mailto:zooko@zooko.com

Copyright 2000, Mojam Media, Inc., all rights reserved.
Author: Skip Montanaro

Copyright 1999, Bioreason, Inc., all rights reserved.
Author: Andrew Dalke

Copyright 1995-1997, Automatrix, Inc., all rights reserv
Author: Skip Montanaro
```

```
Copyright 1991-1995, Stichting Mathematisch Centrum, all


Permission to use, copy, modify, and distribute this Pyt
its associated documentation for any purpose without fee
granted, provided that the above copyright notice appear
and that both that copyright notice and this permission
supporting documentation, and that the name of neither A
Bioreason or Mojam Media be used in advertising or publi
distribution of the software without specific, written p
```

funciones UUencode y UUdecode

The `uu` codec contains the following notice:

```
Copyright 1994 by Lance Ellinghouse
Cathedral City, California Republic, United States of Am
                        All Rights Reserved
Permission to use, copy, modify, and distribute this sof
documentation for any purpose and without fee is hereby
provided that the above copyright notice appear in all c
both that copyright notice and this permission notice ap
supporting documentation, and that the name of Lance Ell
not be used in advertising or publicity pertaining to di
of the software without specific, written prior permissi
LANCE ELLINGHOUSE DISCLAIMS ALL WARRANTIES WITH REGARD T
THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCH
FITNESS, IN NO EVENT SHALL LANCE ELLINGHOUSE CENTRUM BE
FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR AN
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,
OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS

Modified by Jack Jansen, CWI, July 1995:
- Use binascii module to do the actual line-by-line conv
  between ascii and binary. This results in a 1000-fold
  version is still 5 times faster, though.
- Arguments more compliant with Python standard
```

Llamadas a procedimientos remotos XML

El módulo `xmlrpc.client` contiene el siguiente aviso:

```
    The XML-RPC client interface is

Copyright (c) 1999-2002 by Secret Labs AB
Copyright (c) 1999-2002 by Fredrik Lundh

By obtaining, using, and/or copying this software and/or
associated documentation, you agree that you have read,
and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this sof
its associated documentation for any purpose and without
hereby granted, provided that the above copyright notice
all copies, and that both that copyright notice and this
notice appear in supporting documentation, and that the
Secret Labs AB or the author not be used in advertising
pertaining to distribution of the software without speci
prior permission.

SECRET LABS AB AND THE AUTHOR DISCLAIMS ALL WARRANTIES W
TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF ME
ABILITY AND FITNESS.  IN NO EVENT SHALL SECRET LABS AB O
BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAM
DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR P
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TO
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
OF THIS SOFTWARE.
```

test_epoll

El módulo **`test.test_epoll`** contiene el siguiente aviso:

```
Copyright (c) 2001-2006 Twisted Matrix Laboratories.

Permission is hereby granted, free of charge, to any per
a copy of this software and associated documentation fil
```

```
"Software"), to deal in the Software without restrictio
without limitation the rights to use, copy, modify, merg
distribute, sublicense, and/or sell copies of the Softwa
permit persons to whom the Software is furnished to do s
the following conditions:

The above copyright notice and this permission notice sh
included in all copies or substantial portions of the So

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF AN
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WAR
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRI
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHE
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SO
```

Seleccionar kqueue

El módulo `select` contiene el siguiente aviso para la interfaz kqueue:

```
Copyright (c) 2000 Doug White, 2006 James Knight, 2007 C
All rights reserved.

Redistribution and use in source and binary forms, with
modification, are permitted provided that the following
are met:
1. Redistributions of source code must retain the above
   notice, this list of conditions and the following dis
2. Redistributions in binary form must reproduce the abo
   notice, this list of conditions and the following dis
   documentation and/or other materials provided with th

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LI
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIB
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY
```

```
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF S
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER I
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) A
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
SUCH DAMAGE.
```

SipHash24

El archivo `Python/pyhash.c` contiene la implementación de Marek Majkowski del algoritmo SipHash24 de Dan Bernstein. Contiene la siguiente nota:

```
<MIT License>
Copyright (c) 2013  Marek Majkowski <marek@popcount.org>

Permission is hereby granted, free of charge, to any per
of this software and associated documentation files (the
in the Software without restriction, including without l
to use, copy, modify, merge, publish, distribute, sublic
copies of the Software, and to permit persons to whom th
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice sh
all copies or substantial portions of the Software.
</MIT License>

Original location:
   https://github.com/majek/csiphash/

Solution inspired by code from:
   Samuel Neves (supercop/crypto_auth/siphash24/little)
   djb (supercop/crypto_auth/siphash24/little2)
   Jean-Philippe Aumasson (https://131002.net/siphash/si
```

strtod y dtoa

El archivo `Python/dtoa.c`, que proporciona las funciones de C dtoa y strtod para la conversión de dobles C hacia y desde cadenas de caracteres, se deriva del archivo del mismo nombre por David M. Gay, actualmente disponible en https://web.archive.org/web/20220517033456/http://www.netlib.org/fp/dtoa.c. El archivo original, recuperado el 16 de marzo de 2009, contiene el siguiente aviso de licencia y derechos de autor:

```
/****************************************************************
 *
 * The author of this software is David M. Gay.
 *
 * Copyright (c) 1991, 2000, 2001 by Lucent Technologies
 *
 * Permission to use, copy, modify, and distribute this
 * purpose without fee is hereby granted, provided that
 * is included in all copies of any software which is or
 * or modification of this software and in all copies of
 * documentation for such software.
 *
 * THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY
 * WARRANTY.  IN PARTICULAR, NEITHER THE AUTHOR NOR LUCE
 * REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE
 * OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PU
 *
 ***************************************************************
```

OpenSSL

The modules `hashlib`, `posix` and `ssl` use the OpenSSL library for added performance if made available by the operating system. Additionally, the Windows and macOS installers for Python may include a copy of the OpenSSL libraries, so we include a copy of the OpenSSL license here. For the OpenSSL 3.0 release, and later releases derived from that, the Apache License v2 applies:

```
                              Apache License
                        Version 2.0, January 2004
                     https://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBU
```

1. Definitions.

"License" shall mean the terms and conditions for use
and distribution as defined by Sections 1 through 9 o

"Licensor" shall mean the copyright owner or entity a
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting ent
other entities that control, are controlled by, or ar
control with that entity. For the purposes of this de
"control" means (i) the power, direct or indirect, to
direction or management of such entity, whether by co
otherwise, or (ii) ownership of fifty percent (50%) o
outstanding shares, or (iii) beneficial ownership of

"You" (or "Your") shall mean an individual or Legal E
exercising permissions granted by this License.

"Source" form shall mean the preferred form for makin
including but not limited to software source code, do
source, and configuration files.

"Object" form shall mean any form resulting from mech
transformation or translation of a Source form, inclu
not limited to compiled object code, generated docume
and conversions to other media types.

"Work" shall mean the work of authorship, whether in
Object form, made available under the License, as ind
copyright notice that is included in or attached to t
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in So
form, that is based on (or derived from) the Work and
editorial revisions, annotations, elaborations, or ot
represent, as a whole, an original work of authorship

of this License, Derivative Works shall not include w
separable from, or merely link (or bind by name) to t
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, inc
the original version of the Work and any modification
to that Work or Derivative Works thereof, that is int
submitted to Licensor for inclusion in the Work by th
or by an individual or Legal Entity authorized to sub
the copyright owner. For the purposes of this definit
means any form of electronic, verbal, or written comm
to the Licensor or its representatives, including but
communication on electronic mailing lists, source coc
and issue tracking systems that are managed by, or or
Licensor for the purpose of discussing and improving
excluding communication that is conspicuously marked
designated in writing by the copyright owner as "Not

"Contributor" shall mean Licensor and any individual
on behalf of whom a Contribution has been received by
subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and
this License, each Contributor hereby grants to You a
worldwide, non-exclusive, no-charge, royalty-free, ir
copyright license to reproduce, prepare Derivative Wo
publicly display, publicly perform, sublicense, and c
Work and such Derivative Works in Source or Object fo

3. Grant of Patent License. Subject to the terms and cor
this License, each Contributor hereby grants to You a
worldwide, non-exclusive, no-charge, royalty-free, ir
(except as stated in this section) patent license to
use, offer to sell, sell, import, and otherwise trans
where such license applies only to those patent claim
by such Contributor that are necessarily infringed by
Contribution(s) alone or by combination of their Cont
with the Work to which such Contribution(s) was submi

institute patent litigation against any entity (inclu
cross-claim or counterclaim in a lawsuit) alleging th
or a Contribution incorporated within the Work consti
or contributory patent infringement, then any patent
granted to You under this License for that Work shall
as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copi
Work or Derivative Works thereof in any medium, with
modifications, and in Source or Object form, provided
meet the following conditions:

    (a) You must give any other recipients of the Work or
    Derivative Works a copy of this License; and

    (b) You must cause any modified files to carry promin
    stating that You changed the files; and

    (c) You must retain, in the Source form of any Deriva
    that You distribute, all copyright, patent, trade
    attribution notices from the Source form of the W
    excluding those notices that do not pertain to an
    the Derivative Works; and

    (d) If the Work includes a "NOTICE" text file as part
    distribution, then any Derivative Works that You
    include a readable copy of the attribution notice
    within such NOTICE file, excluding those notices
    pertain to any part of the Derivative Works, in a
    of the following places: within a NOTICE text fil
    as part of the Derivative Works; within the Sourc
    documentation, if provided along with the Derivat
    within a display generated by the Derivative Work
    wherever such third-party notices normally appear
    of the NOTICE file are for informational purposes
    do not modify the License. You may add Your own a
    notices within Derivative Works that You distribu
    or as an addendum to the NOTICE text from the Wor

that such additional attribution notices cannot b
as modifying the License.

You may add Your own copyright statement to Your modi
may provide additional or different license terms and
for use, reproduction, or distribution of Your modifi
for any such Derivative Works as a whole, provided Yo
reproduction, and distribution of the Work otherwise
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly st
any Contribution intentionally submitted for inclusio
by You to the Licensor shall be under the terms and c
this License, without any additional terms or conditi
Notwithstanding the above, nothing herein shall super
the terms of any separate license agreement you may h
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to
names, trademarks, service marks, or product names of
except as required for reasonable and customary use i
origin of the Work and reproducing the content of the

7. Disclaimer of Warranty. Unless required by applicable
agreed to in writing, Licensor provides the Work (and
Contributor provides its Contributions) on an "AS IS"
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
implied, including, without limitation, any warrantie
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNE
PARTICULAR PURPOSE. You are solely responsible for de
appropriateness of using or redistributing the Work a
risks associated with Your exercise of permissions un

8. Limitation of Liability. In no event and under no leg
whether in tort (including negligence), contract, or
unless required by applicable law (such as deliberate
negligent acts) or agreed to in writing, shall any Co
liable to You for damages, including any direct, indi

```
   incidental, or consequential damages of any character
   result of this License or out of the use or inability
   Work (including but not limited to damages for loss
   work stoppage, computer failure or malfunction, or a
   other commercial damages or losses), even if such Co
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While re
   the Work or Derivative Works thereof, You may choose
   and charge a fee for, acceptance of support, warranty
   or other liability obligations and/or rights consiste
   License. However, in accepting such obligations, You
   on Your own behalf and on Your sole responsibility,
   of any other Contributor, and only if You agree to i
   defend, and hold each Contributor harmless for any li
   incurred by, or claims asserted against, such Contrib
   of your accepting any such warranty or additional lia

END OF TERMS AND CONDITIONS
```

expat

La extensión **`pyexpat`** se construye usando una copia incluida de las fuentes de expatriados a menos que la construcción esté configurada `--with-system-expat`:

```
Copyright (c) 1998, 1999, 2000 Thai Open Source Software
                               and Clark Cooper

Permission is hereby granted, free of charge, to any pe
a copy of this software and associated documentation fil
"Software"), to deal in the Software without restrictio
without limitation the rights to use, copy, modify, merg
distribute, sublicense, and/or sell copies of the Softwa
permit persons to whom the Software is furnished to do
the following conditions:

The above copyright notice and this permission notice s
in all copies or substantial portions of the Software.
```

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF AN
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WAR
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NO
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LI
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

libffi

La extensión C **`_ctypes`** subyacente al módulo **`ctypes`** se construye usando una copia incluida de las fuentes de libffi a menos que la construcción esté configurada `--with-system-libffi`:

```
Copyright (c) 1996-2008  Red Hat, Inc and others.

Permission is hereby granted, free of charge, to any per
a copy of this software and associated documentation fil
"Software"), to deal in the Software without restrictio
without limitation the rights to use, copy, modify, merg
distribute, sublicense, and/or sell copies of the Softwa
permit persons to whom the Software is furnished to do s
the following conditions:

The above copyright notice and this permission notice sh
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF AN
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WAR
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT.  IN NO EVENT SHALL THE AUTHORS OR COPYR
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABIL
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARI
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
DEALINGS IN THE SOFTWARE.
```

zlib

La extensión `zlib` se crea utilizando una copia incluida de las fuentes de zlib si la versión de zlib encontrada en el sistema es demasiado antigua para ser utilizada para la compilación:

```
Copyright (C) 1995-2011 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express o
warranty.  In no event will the authors be held liable f
arising from the use of this software.

Permission is granted to anyone to use this software for
including commercial applications, and to alter it and r
freely, subject to the following restrictions:

1. The origin of this software must not be misrepresente
   claim that you wrote the original software. If you us
   in a product, an acknowledgment in the product docume
   appreciated but is not required.

2. Altered source versions must be plainly marked as suc
   misrepresented as being the original software.

3. This notice may not be removed or altered from any so

Jean-loup Gailly        Mark Adler
jloup@gzip.org          madler@alumni.caltech.edu
```

cfuhash

La implementación de la tabla hash utilizada por `tracemalloc` se basa en el proyecto cfuhash:

```
Copyright (c) 2005 Don Owens
All rights reserved.

This code is released under the BSD license:

Redistribution and use in source and binary forms, with
modification, are permitted provided that the following
```

```
are met:

  * Redistributions of source code must retain the above
    notice, this list of conditions and the following di

  * Redistributions in binary form must reproduce the ab
    copyright notice, this list of conditions and the fo
    disclaimer in the documentation and/or other materia
    with the distribution.

  * Neither the name of the author nor the names of its
    contributors may be used to endorse or promote produ
    from this software without specific prior written pe

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND C
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AN
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHA
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUT
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INT
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER I
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHER
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
OF THE POSSIBILITY OF SUCH DAMAGE.
```

libmpdec

La extension C **`_decimal`** subyacente al módulo **`decimal`** se construye usando una copia incluida de la biblioteca libmpdec a menos que la construcción esté configurada `--with-system-libmpdec`:

```
Copyright (c) 2008-2020 Stefan Krah. All rights reserved

Redistribution and use in source and binary forms, with
modification, are permitted provided that the following
are met:
```

```
1. Redistributions of source code must retain the above 
   notice, this list of conditions and the following dis

2. Redistributions in binary form must reproduce the abo
   notice, this list of conditions and the following dis
   documentation and/or other materials provided with th

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LI
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A 
ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIB
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF S
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS 
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER I
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) A
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE 
SUCH DAMAGE.
```

Conjunto de pruebas W3C C14N

El conjunto de pruebas C14N 2.0 en el paquete **`test`** (`Lib/test/xmltestdata/c14n-20/`) se recuperó del sitio web de W3C en https://www.w3.org/TR/xml-c14n2-testcases/ y se distribuye bajo la licencia BSD de 3 cláusulas:

```
Copyright (c) 2013 W3C(R) (MIT, ERCIM, Keio, Beihang),
All Rights Reserved.

Redistribution and use in source and binary forms, with 
modification, are permitted provided that the following 
are met:

* Redistributions of works must retain the original copy
  this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the orig
  notice, this list of conditions and the following disc
  documentation and/or other materials provided with the
```

```
* Neither the name of the W3C nor the names of its contr
  used to endorse or promote products derived from this
  specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND C
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AN
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL T
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUS
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILI
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY C
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
```

mimalloc

MIT License:

```
Copyright (c) 2018-2021 Microsoft Corporation, Daan Lei

Permission is hereby granted, free of charge, to any per
of this software and associated documentation files (the
in the Software without restriction, including without l
to use, copy, modify, merge, publish, distribute, sublic
copies of the Software, and to permit persons to whom th
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice sh
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF AN
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DA
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTH
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
SOFTWARE.
```

asyncio

Parts of the `asyncio` module are incorporated from uvloop 0.16 [https://github.com/MagicStack/uvloop/tree/v0.16.0], which is distributed under the MIT license:

```
Copyright (c) 2015-2021 MagicStack Inc.  http://magic.io

Permission is hereby granted, free of charge, to any per
a copy of this software and associated documentation fil
"Software"), to deal in the Software without restriction
without limitation the rights to use, copy, modify, merg
distribute, sublicense, and/or sell copies of the Softwa
permit persons to whom the Software is furnished to do s
the following conditions:

The above copyright notice and this permission notice sh
included in all copies or substantial portions of the So

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF AN
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WAR
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRI
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHE
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SO
```

Global Unbounded Sequences (GUS)

The file `Python/qsbr.c` is adapted from FreeBSD's «Global Unbounded Sequences» safe memory reclamation scheme in subr_smr.c [https://github.com/freebsd/freebsd-src/blob/main/sys/kern/subr_smr.c]. The file is distributed under the 2-Clause BSD License:

```
Copyright (c) 2019,2020 Jeffrey Roberson <jeff@FreeBSD.o

Redistribution and use in source and binary forms, with
modification, are permitted provided that the following
are met:
```

```
1. Redistributions of source code must retain the above
   notice unmodified, this list of conditions, and the f
   disclaimer.
2. Redistributions in binary form must reproduce the abo
   notice, this list of conditions and the following dis
   documentation and/or other materials provided with th

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE I
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, I
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVI
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUS
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILI
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY C
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUC
```

Zstandard bindings

Zstandard bindings in `Modules/_zstd` and `Lib/compression/zstd` are based on code from the pyzstd library [https://github.com/Rogdham/pyzstd/], copyright Ma Lin and contributors. The pyzstd code is distributed under the 3-Clause BSD License:

```
Copyright (c) 2020-present, Ma Lin and contributors.
All rights reserved.

Redistribution and use in source and binary forms, with
modification, are permitted provided that the following

1. Redistributions of source code must retain the above
   list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the abo
   this list of conditions and the following disclaimer
   and/or other materials provided with the distributio

3. Neither the name of the copyright holder nor the name
```

contributors may be used to endorse or promote produc
this software without specific prior written permissi

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND C
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NO
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CO
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF S
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INT
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRA
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN A
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF

Índice de Módulos Python

_ | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | z

_	
`__future__`	*Future statement definitions*
`__main__`	*The environment where top-level code is run. Covers command-line interfaces, import-time behavior, and ``_name_ = = '_main_'``.*
`_thread`	*Low-level threading API.*
`_tkinter`	*A binary module that contains the low-level interface to Tcl/Tk.*
a	
`abc`	*Abstract base classes according to :pep:`3119`.*
`aifc`	**Obsoleto:** *Removed in 3.13.*
`annotationlib`	*Functionality for introspecting annotations*
`argparse`	*Command-line option and argument parsing library.*
`array`	*Space efficient arrays of uniformly typed numeric values.*
`ast`	*Abstract Syntax Tree classes and*

	manipulation.
`asynchat`	**Obsoleto:** *Removed in 3.12.*
`asyncio`	*Asynchronous I/O.*
`asyncore`	**Obsoleto:** *Removed in 3.12.*
`atexit`	*Register and execute cleanup functions.*
`audioop`	**Obsoleto:** *Removed in 3.13.*
b	
`base64`	*RFC 4648: Base16, Base32, Base64 Data Encodings; Base85 and Ascii85*
`bdb`	*Debugger framework.*
`binascii`	*Tools for converting between binary and various ASCII-encoded binary representations.*
`bisect`	*Array bisection algorithms for binary searching.*
`builtins`	*The module that provides the built-in namespace.*
`bz2`	*Interfaces for bzip2 compression and decompression.*
c	
`calendar`	*Functions for working with calendars, including some emulation of the Unix cal program.*
`cgi`	**Obsoleto:** *Removed in 3.13.*
`cgitb`	**Obsoleto:** *Removed in*

	3.13.
`chunk`	**Obsoleto:** *Removed in 3.13.*
`cmath`	*Mathematical functions for complex numbers.*
`cmd`	*Build line-oriented command interpreters.*
`code`	*Facilities to implement read-eval-print loops.*
`codecs`	*Encode and decode data and streams.*
`codeop`	*Compile (possibly incomplete) Python code.*
`collections`	*Container datatypes*
`collections.abc`	*Abstract base classes for containers*
`colorsys`	*Conversion functions between RGB and other color systems.*
`compileall`	*Tools for byte-compiling all Python source files in a directory tree.*
`compression`	
`compression.zstd`	*Low-level interface to compression and decompression routines in the zstd library.*
`concurrent`	
`concurrent.futures`	*Execute computations concurrently using threads or processes.*
`concurrent.interpreters`	*Multiple interpreters in the same process*
`configparser`	*Configuration file parser.*
`contextlib`	*Utilities for with-statement contexts.*
`contextvars`	*Context Variables*
`copy`	*Shallow and deep copy*

	operations.
`copyreg`	*Register pickle support functions.*
`cProfile`	
`crypt`	**Obsoleto:** *Removed in 3.13.*
`csv`	*Write and read tabular data to and from delimited files.*
`ctypes`	*A foreign function library for Python.*
`curses`	*An interface to the curses library, providing portable terminal handling.*
`curses.ascii`	*Constants and set-membership functions for ASCII characters.*
`curses.panel`	*A panel stack extension that adds depth to curses windows.*
`curses.textpad`	*Emacs-like input editing in a curses window.*
d	
`dataclasses`	*Generate special methods on user-defined classes.*
`datetime`	*Basic date and time types.*
`dbm`	*Interfaces to various Unix "database" formats.*
`dbm.dumb`	*Portable implementation of the simple DBM interface.*
`dbm.gnu`	*GNU database manager*
`dbm.ndbm`	*The New Database Manager*
`dbm.sqlite3`	*SQLite backend for dbm*

`decimal`	*Implementation of the General Decimal Arithmetic Specification.*
`difflib`	*Helpers for computing differences between objects.*
`dis`	*Disassembler for Python bytecode.*
`distutils`	**Obsoleto:** *Removed in 3.12.*
`doctest`	*Test pieces of code within docstrings.*
e	
`email`	*Package supporting the parsing, manipulating, and generating email messages.*
`email.charset`	*Character Sets*
`email.contentmanager`	*Storing and Retrieving Content from MIME Parts*
`email.encoders`	*Encoders for email message payloads.*
`email.errors`	*The exception classes used by the email package.*
`email.generator`	*Generate flat text email messages from a message structure.*
`email.header`	*Representing non-ASCII headers*
`email.headerregistry`	*Automatic Parsing of headers based on the field name*
`email.iterators`	*Iterate over a message object tree.*
`email.message`	*The base class representing email*

	messages.
`email.mime`	*Build MIME messages.*
`email.mime.application`	
`email.mime.audio`	
`email.mime.base`	
`email.mime.image`	
`email.mime.message`	
`email.mime.multipart`	
`email.mime.nonmultipart`	
`email.mime.text`	
`email.parser`	*Parse flat text email messages to produce a message object structure.*
`email.policy`	*Controlling the parsing and generating of messages*
`email.utils`	*Miscellaneous email package utilities.*
`encodings`	*Encodings package*
`encodings.idna`	*Internationalized Domain Names implementation*
`encodings.mbcs`	*Windows ANSI codepage*
`encodings.utf_8_sig`	*UTF-8 codec with BOM signature*
`ensurepip`	*Bootstrapping the "pip" installer into an existing Python installation or virtual environment.*
`enum`	*Implementation of an*

		enumeration class.
	errno	*Standard errno system symbols.*
	f	
	faulthandler	*Dump the Python traceback.*
	fcntl	*The fcntl() and ioctl() system calls.*
	filecmp	*Compare files efficiently.*
	fileinput	*Loop over standard input or a list of files.*
	fnmatch	*Unix shell style filename pattern matching.*
	fractions	*Rational numbers.*
	ftplib	*FTP protocol client (requires sockets).*
	functools	*Higher-order functions and operations on callable objects.*
	g	
	gc	*Interface to the cycle-detecting garbage collector.*
	getopt	*Portable parser for command line options; support both short and long option names.*
	getpass	*Portable reading of passwords and retrieval of the userid.*
	gettext	*Multilingual internationalization services.*
	glob	*Unix shell style pathname pattern expansion.*

graphlib	*Functionality to operate with graph-like structures*
grp	*The group database (getgrnam() and friends).*
gzip	*Interfaces for gzip compression and decompression using file objects.*
h	
hashlib	*Secure hash and message digest algorithms.*
heapq	*Heap queue algorithm (a.k.a. priority queue).*
hmac	*Keyed-Hashing for Message Authentication (HMAC) implementation*
html	*Helpers for manipulating HTML.*
html.entities	*Definitions of HTML general entities.*
html.parser	*A simple parser that can handle HTML and XHTML.*
http	*HTTP status codes and messages*
http.client	*HTTP and HTTPS protocol client (requires sockets).*
http.cookiejar	*Classes for automatic handling of HTTP cookies.*
http.cookies	*Support for HTTP state management (cookies).*
http.server	*HTTP server and request handlers.*

i	
`idlelib`	*Implementation package for the IDLE shell/ editor.*
`imaplib`	*IMAP4 protocol client (requires sockets).*
`imghdr`	**Obsoleto:** *Removed in 3.13.*
`imp`	**Obsoleto:** *Removed in 3.12.*
`importlib`	*The implementation of the import machinery.*
`importlib.abc`	*Abstract base classes related to import*
`importlib.machinery`	*Importers and path hooks*
`importlib.metadata`	*Accessing package metadata*
`importlib.resources`	*Package resource reading, opening, and access*
`importlib.resources.abc`	*Abstract base classes for resources*
`importlib.util`	*Utility code for importers*
`inspect`	*Extract information and source code from live objects.*
`io`	*Core tools for working with streams.*
`ipaddress`	*IPv4/IPv6 manipulation library.*
`itertools`	*Functions creating iterators for efficient looping.*
j	
`json`	*Encode and decode the JSON format.*

`json.tool`	*A command-line interface to validate and pretty-print JSON.*
k	
`keyword`	*Test whether a string is a keyword in Python.*
l	
`linecache`	*Provides random access to individual lines from text files.*
`locale`	*Internationalization services.*
`logging`	*Flexible event logging system for applications.*
`logging.config`	*Configuration of the logging module.*
`logging.handlers`	*Handlers for the logging module.*
`lzma`	*A Python wrapper for the liblzma compression library.*
m	
`mailbox`	*Manipulate mailboxes in various formats*
`mailcap`	**Obsoleto:** *Removed in 3.13.*
`marshal`	*Convert Python objects to streams of bytes and back (with different constraints).*
`math`	*Mathematical functions (sin() etc.).*
`mimetypes`	*Mapping of filename extensions to MIME types.*
`mmap`	*Interface to memory-mapped files for Unix*

	and Windows.
`modulefinder`	*Find modules used by a script.*
`msilib`	**Obsoleto:** *Removed in 3.13.*
`msvcrt`	*Miscellaneous useful routines from the MS VC++ runtime.*
`multiprocessing`	*Process-based parallelism.*
`multiprocessing.connection`	*API for dealing with sockets.*
`multiprocessing.dummy`	*Dumb wrapper around threading.*
`multiprocessing.managers`	*Share data between process with shared objects.*
`multiprocessing.pool`	*Create pools of processes.*
`multiprocessing.shared_memory`	*Provides shared memory for direct access across processes.*
`multiprocessing.sharedctypes`	*Allocate ctypes objects from shared memory.*
n	
`netrc`	*Loading of .netrc files.*
`nis`	**Obsoleto:** *Removed in 3.13.*
`nntplib`	**Obsoleto:** *Removed in 3.13.*
`numbers`	*Numeric abstract base classes (Complex, Real, Integral, etc.).*
o	
`operator`	*Functions corresponding to the standard operators.*
`optparse`	*Command-line option*

	parsing library.
`os`	*Miscellaneous operating system interfaces.*
`os.path`	*Operations on pathnames.*
`ossaudiodev`	**Obsoleto:** *Removed in 3.13.*
p	
`pathlib`	*Object-oriented filesystem paths*
`pathlib.types`	*pathlib types for static type checking*
`pdb`	*The Python debugger for interactive interpreters.*
`pickle`	*Convert Python objects to streams of bytes and back.*
`pickletools`	*Contains extensive comments about the pickle protocols and pickle-machine opcodes, as well as some useful functions.*
`pipes`	**Obsoleto:** *Removed in 3.13.*
`pkgutil`	*Utilities for the import system.*
`platform`	*Retrieves as much platform identifying data as possible.*
`plistlib`	*Generate and parse Apple plist files.*
`poplib`	*POP3 protocol client (requires sockets).*
`posix`	*The most common POSIX system calls (normally used via module os).*

`pprint`	*Data pretty printer.*
`profile`	*Python source profiler.*
`pstats`	*Statistics object for use with the profiler.*
`pty`	*Pseudo-Terminal Handling for Unix.*
`pwd`	*The password database (getpwnam() and friends).*
`py_compile`	*Generate byte-code files from Python source files.*
`pyclbr`	*Supports information extraction for a Python module browser.*
`pydoc`	*Documentation generator and online help system.*
q	
`queue`	*A synchronized queue class.*
`quopri`	*Encode and decode files using the MIME quoted-printable encoding.*
r	
`random`	*Generate pseudo-random numbers with various common distributions.*
`re`	*Regular expression operations.*
`readline`	*GNU readline support for Python.*
`reprlib`	*Alternate repr() implementation with size limits.*
`resource`	*An interface to provide resource usage*

	information on the current process.
`rlcompleter`	*Python identifier completion, suitable for the GNU readline library.*
`runpy`	*Locate and run Python modules without importing them first.*
s	
`sched`	*General purpose event scheduler.*
`secrets`	*Generate secure random numbers for managing secrets.*
`select`	*Wait for I/O completion on multiple streams.*
`selectors`	*High-level I/O multiplexing.*
`shelve`	*Python object persistence.*
`shlex`	*Simple lexical analysis for Unix shell-like languages.*
`shutil`	*High-level file operations, including copying.*
`signal`	*Set handlers for asynchronous events.*
`site`	*Module responsible for site-specific configuration.*
`sitecustomize`	
`smtpd`	**Obsoleto:** *Removed in 3.12.*
`smtplib`	*SMTP protocol client (requires sockets).*
`sndhdr`	**Obsoleto:** *Removed in 3.13.*

`socket`	*Low-level networking interface.*
`socketserver`	*A framework for network servers.*
`spwd`	**Obsoleto:** *Removed in 3.13.*
`sqlite3`	*A DB-API 2.0 implementation using SQLite 3.x.*
`ssl`	*TLS/SSL wrapper for socket objects*
`stat`	*Utilities for interpreting the results of os.stat(), os.lstat() and os.fstat().*
`statistics`	*Mathematical statistics functions*
`string`	*Common string operations.*
`string.templatelib`	*Support for template string literals.*
`stringprep`	*String preparation, as per RFC 3453*
`struct`	*Interpret bytes as packed binary data.*
`subprocess`	*Subprocess management.*
`sunau`	**Obsoleto:** *Removed in 3.13.*
`symtable`	*Interface to the compiler's internal symbol tables.*
`sys`	*Access system-specific parameters and functions.*
`sys.monitoring`	*Access and control event monitoring*
`sysconfig`	*Python's configuration information*
`syslog`	*An interface to the Unix syslog library routines.*

t	
`tabnanny`	*Tool for detecting white space related problems in Python source files in a directory tree.*
`tarfile`	*Read and write tar-format archive files.*
`telnetlib`	**Obsoleto:** *Removed in 3.13.*
`tempfile`	*Generate temporary files and directories.*
`termios`	*POSIX style tty control.*
`test`	*Regression tests package containing the testing suite for Python.*
`test.regrtest`	*Drives the regression test suite.*
`test.support`	*Support for Python's regression test suite.*
`test.support.bytecode_helper`	*Support tools for testing correct bytecode generation.*
`test.support.import_helper`	*Support for import tests.*
`test.support.os_helper`	*Support for os tests.*
`test.support.script_helper`	*Support for Python's script execution tests.*
`test.support.socket_helper`	*Support for socket tests.*
`test.support.threading_helper`	*Support for threading tests.*
`test.support.warnings_helper`	*Support for warnings tests.*
`textwrap`	*Text wrapping and filling*
`threading`	*Thread-based parallelism.*
`time`	*Time access and*

	conversions.
`timeit`	*Measure the execution time of small code snippets.*
`tkinter`	*Interface to Tcl/Tk for graphical user interfaces*
`tkinter.colorchooser`	*Color choosing dialog*
`tkinter.commondialog`	*Tkinter base class for dialogs*
`tkinter.dnd`	*Tkinter drag-and-drop interface*
`tkinter.filedialog`	*Dialog classes for file selection*
`tkinter.font`	*Tkinter font-wrapping class*
`tkinter.messagebox`	*Various types of alert dialogs*
`tkinter.scrolledtext`	*Text widget with a vertical scroll bar.*
`tkinter.simpledialog`	*Simple dialog windows*
`tkinter.ttk`	*Tk themed widget set*
`token`	*Constants representing terminal nodes of the parse tree.*
`tokenize`	*Lexical scanner for Python source code.*
`tomllib`	*Parse TOML files.*
`trace`	*Trace or track Python statement execution.*
`traceback`	*Print or retrieve a stack traceback.*
`tracemalloc`	*Trace memory allocations.*
`tty`	*Utility functions that perform common terminal control operations.*
`turtle`	*An educational*

	framework for simple graphics applications
`turtledemo`	*A viewer for example turtle scripts*
`types`	*Names for built-in types.*
`typing`	*Support for type hints (see :pep:`484`).*
u	
`unicodedata`	*Access the Unicode Database.*
`unittest`	*Unit testing framework for Python.*
`unittest.mock`	*Mock object library.*
`urllib`	
`urllib.error`	*Exception classes raised by urllib.request.*
`urllib.parse`	*Parse URLs into or assemble them from components.*
`urllib.request`	*Extensible library for opening URLs.*
`urllib.response`	*Response classes used by urllib.*
`urllib.robotparser`	*Load a robots.txt file and answer questions about fetchability of other URLs.*
`usercustomize`	
`uu`	**Obsoleto:** *Removed in 3.13.*
`uuid`	*UUID objects (universally unique identifiers) according to RFC 9562*
v	
`venv`	*Creation of virtual environments.*

w	
`warnings`	*Issue warning messages and control their disposition.*
`wave`	*Provide an interface to the WAV sound format.*
`weakref`	*Support for weak references and weak dictionaries.*
`webbrowser`	*Easy-to-use controller for web browsers.*
`winreg`	*Routines and objects for manipulating the Windows registry.*
`winsound`	*Access to the sound-playing machinery for Windows.*
`wsgiref`	*WSGI Utilities and Reference Implementation.*
`wsgiref.handlers`	*WSGI server/gateway base classes.*
`wsgiref.headers`	*WSGI response header tools.*
`wsgiref.simple_server`	*A simple WSGI HTTP server.*
`wsgiref.types`	*WSGI types for static type checking*
`wsgiref.util`	*WSGI environment utilities.*
`wsgiref.validate`	*WSGI conformance checker.*
x	
`xdrlib`	**Obsoleto:** *Removed in 3.13.*
`xml`	*Package containing XML processing modules*

`xml.dom`	*Document Object Model API for Python.*
`xml.dom.minidom`	*Minimal Document Object Model (DOM) implementation.*
`xml.dom.pulldom`	*Support for building partial DOM trees from SAX events.*
`xml.etree.ElementInclude`	
`xml.etree.ElementTree`	*Implementation of the ElementTree API.*
`xml.parsers.expat`	*An interface to the Expat non-validating XML parser.*
`xml.parsers.expat.errors`	
`xml.parsers.expat.model`	
`xml.sax`	*Package containing SAX2 base classes and convenience functions.*
`xml.sax.handler`	*Base classes for SAX event handlers.*
`xml.sax.saxutils`	*Convenience functions and classes for use with SAX.*
`xml.sax.xmlreader`	*Interface which SAX-compliant XML parsers must implement.*
`xmlrpc`	*Server and client modules implementing XML-RPC.*
`xmlrpc.client`	*XML-RPC client access.*
`xmlrpc.server`	*Basic XML-RPC server implementations.*
z	
`zipapp`	*Manage executable Python zip archives*

`zipfile`	*Read and write ZIP-format archive files.*
`zipimport`	*Support for importing Python modules from ZIP archives.*
`zlib`	*Low-level interface to compression and decompression routines compatible with gzip.*
`zoneinfo`	*IANA time zone support*

Improve a documentation page

We are always interested to hear ideas about improvements to the documentation.

If the bug or suggested improvement concerns the translation of this documentation, open an issue or edit the page in translation's repository [https://github.com/python/python-docs-es] instead.

You have a few ways to ask questions or suggest changes:

- You can start a discussion about the page on the Python discussion forum. This link will start a topic in the Documentation category: New Documentation topic [https://discuss.python.org/new-topic?category=documentation].
- You can open an issue on the Python GitHub issue tracker. This link will create a new issue with the «docs» label: New docs issue [https://github.com/python/cpython/issues/new?template=documentation.yml].

Improve a documentation page

We are always interested to hear ideas about improvements to the documentation.

You were reading «PAGETITLE» at PAGEURL. The source for that page is on GitHub [https://github.com/python/cpython/blob/main/Doc/PAGESOURCE?plain=1].

If the bug or suggested improvement concerns the translation of this documentation, open an issue or edit the page in translation's repository [https://github.com/python/python-docs-es] instead.

You have a few ways to ask questions or suggest changes:

- You can start a discussion about the page on the Python discussion forum. This link will start a pre-populated topic: Question about page «PAGETITLE» [https://discuss.python.org/new-topic?category=documentation&title=Question+about+page+%22PAGETITLE%22&body=About+the+page+at+PAGEURL%3A].
- You can open an issue on the Python GitHub issue tracker. This link will create a new pre-populated issue: Docs: problem with page «PAGETITLE» [https://github.com/python/cpython/issues/new?template=documentation.yml&title=Docs%3A+problem+with+page+%22PAGETITLE%22&description=The+page+at+PAGEURL+has+a+problem%3A].
- You can edit the page on GitHub [https://github.com/python/cpython/blob/main/Doc/PAGESOURCE?plain=1] to open a pull request and begin the contribution process.

Initialization, finalization, and threads

This page has been split up into the following:

- Interpreter initialization and finalization
- Thread states and the global interpreter lock
- Synchronization primitives
- Thread-local storage support
- Multiple interpreters in a Python process
- Profiling and tracing

Pending removal in Python 3.14

- El campo `ma_version_tag` en **`PyDictObject`** para módulos de extensión (**PEP 699** [https://peps.python.org/pep-0699/]; gh-101193 [https://github.com/python/cpython/issues/101193]).
- Crear **`tipos inmutables`** con bases mutables (gh-95388 [https://github.com/python/cpython/issues/95388]).

Pending removal in Python 3.15

- La función `PyImport_ImportModuleNoBlock()`: use `PyImport_ImportModule()` en su lugar.
- `PyWeakref_GetObject()` and `PyWeakref_GET_OBJECT()`: Use `PyWeakref_GetRef()` instead. The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get `PyWeakref_GetRef()` on Python 3.12 and older.
- El tipo `Py_UNICODE` y macro `Py_UNICODE_WIDE`: use `wchar_t` en su lugar.
- `PyUnicode_AsDecodedObject()`: Use `PyCodec_Decode()` instead.
- `PyUnicode_AsDecodedUnicode()`: Use `PyCodec_Decode()` instead; Note that some codecs (for example, «base64») may return a type other than `str`, such as `bytes`.
- `PyUnicode_AsEncodedObject()`: Use `PyCodec_Encode()` instead.
- `PyUnicode_AsEncodedUnicode()`: Use `PyCodec_Encode()` instead; Note that some codecs (for example, «base64») may return a type other than `bytes`, such as `str`.
- Python initialization functions, deprecated in Python 3.13:
  - `Py_GetPath()`: Use `PyConfig_Get("module_search_paths")` (`sys.path`) instead.

  - `Py_GetPrefix()`: Use `PyConfig_Get("base_prefix")` (`sys.base_prefix`) instead. Use `PyConfig_Get("prefix")` (`sys.prefix`) if virtual environments need to be handled.
  - `Py_GetExecPrefix()`: Use `PyConfig_Get("base_exec_prefix")` (`sys.base_exec_prefix`) instead. Use `PyConfig_Get("exec_prefix")` (`sys.exec_prefix`) if virtual environments need to be handled.
  - `Py_GetProgramFullPath()`: Use `PyConfig_Get("executable")` (`sys.executable`) instead.
  - `Py_GetProgramName()`: Use `PyConfig_Get("executable")` (`sys.executable`) instead.
  - `Py_GetPythonHome()`: Use `PyConfig_Get("home")` or the `PYTHONHOME` environment variable instead.

  The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get `PyConfig_Get()` on Python 3.13 and older.

- Functions to configure Python's initialization, deprecated in Python 3.11:

  - `PySys_SetArgvEx()`: Set `PyConfig.argv` instead.
  - `PySys_SetArgv()`: Set `PyConfig.argv` instead.
  - `Py_SetProgramName()`: Set `PyConfig.program_name` instead.
  - `Py_SetPythonHome()`: Set `PyConfig.home` instead.
  - `PySys_ResetWarnOptions()`: Borre `sys.warnoptions` y `warnings.filters` en su lugar.

  The `Py_InitializeFromConfig()` API should be used with `PyConfig` instead.

- Global configuration variables:

- `Py_DebugFlag`: Use `PyConfig.parser_debug` or `PyConfig_Get("parser_debug")` instead.
- `Py_VerboseFlag`: Use `PyConfig.verbose` or `PyConfig_Get("verbose")` instead.
- `Py_QuietFlag`: Use `PyConfig.quiet` or `PyConfig_Get("quiet")` instead.
- `Py_InteractiveFlag`: Use `PyConfig.interactive` or `PyConfig_Get("interactive")` instead.
- `Py_InspectFlag`: Use `PyConfig.inspect` or `PyConfig_Get("inspect")` instead.
- `Py_OptimizeFlag`: Use `PyConfig.optimization_level` or `PyConfig_Get("optimization_level")` instead.
- `Py_NoSiteFlag`: Use `PyConfig.site_import` or `PyConfig_Get("site_import")` instead.
- `Py_BytesWarningFlag`: Use `PyConfig.bytes_warning` or `PyConfig_Get("bytes_warning")` instead.
- `Py_FrozenFlag`: Use `PyConfig.pathconfig_warnings` or `PyConfig_Get("pathconfig_warnings")` instead.
- `Py_IgnoreEnvironmentFlag`: Use `PyConfig.use_environment` or `PyConfig_Get("use_environment")` instead.
- `Py_DontWriteBytecodeFlag`: Use `PyConfig.write_bytecode` or `PyConfig_Get("write_bytecode")` instead.
- `Py_NoUserSiteDirectory`: Use `PyConfig.user_site_directory` or `PyConfig_Get("user_site_directory")` instead.
- `Py_UnbufferedStdioFlag`: Use `PyConfig.buffered_stdio` or `PyConfig_Get("buffered_stdio")` instead.
- `Py_HashRandomizationFlag`: Use `PyConfig.use_hash_seed` and `PyConfig.hash_seed` or `PyConfig_Get("hash_seed")` instead.
- `Py_IsolatedFlag`: Use `PyConfig.isolated` or `PyConfig_Get("isolated")` instead.

  - `Py_LegacyWindowsFSEncodingFlag`: Use `PyPreConfig.legacy_windows_fs_encoding` or `PyConfig_Get("legacy_windows_fs_encoding")` instead.
  - `Py_LegacyWindowsStdioFlag`: Use `PyConfig.legacy_windows_stdio` or `PyConfig_Get("legacy_windows_stdio")` instead.
  - **`Py_FileSystemDefaultEncoding`**, **`Py_HasFileSystemDefaultEncoding`**: Use `PyConfig.filesystem_encoding` or `PyConfig_Get("filesystem_encoding")` instead.
  - **`Py_FileSystemDefaultEncodeErrors`**: Use `PyConfig.filesystem_errors` or `PyConfig_Get("filesystem_errors")` instead.
  - **`Py_UTF8Mode`**: Use `PyPreConfig.utf8_mode` or `PyConfig_Get("utf8_mode")` instead. (see `Py_PreInitialize()`)

The `Py_InitializeFromConfig()` API should be used with `PyConfig` to set these options. Or `PyConfig_Get()` can be used to get these options at runtime.

Pending removal in Python 3.16

- The bundled copy of `libmpdec`.

Pending removal in Python 3.18

- The following private functions are deprecated and planned for removal in Python 3.18:
    - `_PyBytes_Join()`: use `PyBytes_Join()`.
    - `_PyDict_GetItemStringWithError()`: use `PyDict_GetItemStringRef()`.
    - `_PyDict_Pop()`: use `PyDict_Pop()`.
    - `_PyLong_Sign()`: use `PyLong_GetSign()`.
    - `_PyLong_FromDigits()` and `_PyLong_New()`: use `PyLongWriter_Create()`.
    - `_PyThreadState_UncheckedGet()`: use `PyThreadState_GetUnchecked()`.
    - `_PyUnicode_AsString()`: use `PyUnicode_AsUTF8()`.
    - `_PyUnicodeWriter_Init()`: replace `_PyUnicodeWriter_Init(&writer)` with `writer = PyUnicodeWriter_Create(0)`.
    - `_PyUnicodeWriter_Finish()`: replace `_PyUnicodeWriter_Finish(&writer)` with `PyUnicodeWriter_Finish(writer)`.
    - `_PyUnicodeWriter_Dealloc()`: replace `_PyUnicodeWriter_Dealloc(&writer)` with `PyUnicodeWriter_Discard(writer)`.
    - `_PyUnicodeWriter_WriteChar()`: replace `_PyUnicodeWriter_WriteChar(&writer, ch)` with `PyUnicodeWriter_WriteChar(writer, ch)`.
    - `_PyUnicodeWriter_WriteStr()`: replace `_PyUnicodeWriter_WriteStr(&writer, str)` with `PyUnicodeWriter_WriteStr(writer, str)`.
    - `_PyUnicodeWriter_WriteSubstring()`: replace

    `_PyUnicodeWriter_WriteSubstring(&writer, str, start, end)` with **`PyUnicodeWriter_WriteSubstring(writer, str, start, end)`**.
  - **`_PyUnicodeWriter_WriteASCIIString()`**: replace `_PyUnicodeWriter_WriteASCIIString(&writer, str)` with **`PyUnicodeWriter_WriteASCII(writer, str)`**.
  - **`_PyUnicodeWriter_WriteLatin1String()`**: replace `_PyUnicodeWriter_WriteLatin1String(&writer, str)` with **`PyUnicodeWriter_WriteUTF8(writer, str)`**.
  - **`_PyUnicodeWriter_Prepare()`**: (no replacement).
  - **`_PyUnicodeWriter_PrepareKind()`**: (no replacement).
  - **`_Py_HashPointer()`**: use **`Py_HashPointer()`**.
  - **`_Py_fopen_obj()`**: use **`Py_fopen()`**.

The pythoncapi-compat project [https://github.com/python/pythoncapi-compat/] can be used to get these new public functions on Python 3.13 and older. (Contributed by Victor Stinner in gh-128863 [https://github.com/python/cpython/issues/128863].)

Pending removal in future versions

Las siguientes APIs están obsoletas y serán eliminadas, aunque actualmente no hay fecha prevista para su eliminación.

- `Py_TPFLAGS_HAVE_FINALIZE`: Innecesaria desde Python 3.8.
- `PyErr_Fetch()`: Utilice `PyErr_GetRaisedException()` en su lugar.
- `PyErr_NormalizeException()`: Utilice `PyErr_GetRaisedException()` en su lugar.
- `PyErr_Restore()`: Utilice `PyErr_SetRaisedException()` en su lugar.
- `PyModule_GetFilename()`: Utilice `PyModule_GetFilenameObject()` en su lugar.
- `PyOS_AfterFork()`: Utilice `PyOS_AfterFork_Child()` en su lugar.
- `PySlice_GetIndicesEx()`: Utilice `PySlice_Unpack()` y `PySlice_AdjustIndices()` en su lugar.
- `PyUnicode_READY()`: Innecesaria desde Python 3.12
- `PyErr_Display()`: Utilice `PyErr_DisplayException()` en su lugar.
- `_PyErr_ChainExceptions()`: Utilice `_PyErr_ChainExceptions1()` en su lugar.
- Miembro `PyBytesObject.ob_shash`: Llame a `PyObject_Hash()` en su lugar.
- API de almacenamiento local de hilos (TLS):
    - `PyThread_create_key()`: Utilice `PyThread_tss_alloc()` en su lugar.
    - `PyThread_delete_key()`: Utilice `PyThread_tss_free()` en su lugar.
    - `PyThread_set_key_value()`: Utilice `PyThread_tss_set()` en su lugar.
    - `PyThread_get_key_value()`: Utilice

`PyThread_tss_get()` en su lugar.

  - `PyThread_delete_key_value()`: Utilice `PyThread_tss_delete()` en su lugar.
  - `PyThread_ReInitTLS()`: Innecesaria desde Python 3.7.

Pending removal in Python 3.13

Módulos (ver **PEP 594** [https://peps.python.org/pep-0594/]):

- **`aifc`**
- **`audioop`**
- **`cgi`**
- **`cgitb`**
- **`chunk`**
- **`crypt`**
- **`imghdr`**
- **`mailcap`**
- **`msilib`**
- **`nis`**
- **`nntplib`**
- **`ossaudiodev`**
- **`pipes`**
- **`sndhdr`**
- **`spwd`**
- **`sunau`**
- **`telnetlib`**
- **`uu`**
- **`xdrlib`**

Otros módulos:

- **`lib2to3`**, y el programa **2to3** (gh-84540 [https://github.com/python/cpython/issues/84540])

APIs:

- **`configparser.LegacyInterpolation`** (gh-90765 [https://github.com/python/cpython/issues/90765])
- `locale.resetlocale()` (gh-90817 [https://github.com/python/cpython/issues/90817])

- **`turtle.RawTurtle.settiltangle()`** (gh-50096 [https://github.com/python/cpython/issues/50096])
- **`unittest.findTestCases()`** (gh-50096 [https://github.com/python/cpython/issues/50096])
- **`unittest.getTestCaseNames()`** (gh-50096 [https://github.com/python/cpython/issues/50096])
- **`unittest.makeSuite()`** (gh-50096 [https://github.com/python/cpython/issues/50096])
- **`unittest.TestProgram.usageExit()`** (gh-67048 [https://github.com/python/cpython/issues/67048])
- **`webbrowser.MacOSX`** (gh-86421 [https://github.com/python/cpython/issues/86421])
- Encadenamiento de descriptores **`classmethod`** (gh-89519 [https://github.com/python/cpython/issues/89519])

Pending removal in Python 3.14

- **`argparse`**: Los parámetros de **`argparse.BooleanOptionalAction`**, *type*, *choices*, y *metavar*, son obsoletos y serán eliminados en la versión 3.14 (Contribución de Nikita Sobolev en gh-92248 [https://github.com/python/cpython/issues/92248].)
- **`ast`**: Las siguientes capacidades han quedado obsoletas en la documentación desde Python 3.8, ahora hacen que se emita una excepción **`DeprecationWarning`** en tiempo de ejecución cuando se accede a ellas o se utilizan, y se eliminarán en Python 3.14:

  - **`ast.Num`**
  - **`ast.Str`**
  - **`ast.Bytes`**
  - **`ast.NameConstant`**
  - **`ast.Ellipsis`**

  Use **`ast.Constant`** en su lugar. (Contribución de Serhiy Storchaka en gh-90953 [https://github.com/python/cpython/issues/90953].)
- **`asyncio`**:

  - The child watcher classes **`asyncio.MultiLoopChildWatcher`**, **`asyncio.FastChildWatcher`**, **`asyncio.AbstractChildWatcher`** and **`asyncio.SafeChildWatcher`** are deprecated and will be removed in Python 3.14. (Contributed by Kumar Aditya in gh-94597 [https://github.com/python/cpython/issues/94597].)
  - **`asyncio.set_child_watcher()`**,

    `asyncio.get_child_watcher()`,
    `asyncio.AbstractEventLoopPolicy.set_child_watch`
    and
    `asyncio.AbstractEventLoopPolicy.get_child_watch`
    are deprecated and will be removed in Python 3.14. (Contributed by Kumar Aditya in gh-94597 [https://github.com/python/cpython/issues/94597].)
  - ○ El método `get_event_loop()` de la política de bucle de eventos predeterminada ahora emite un `DeprecationWarning` si no hay ningún bucle de eventos definido actualmente y decide crear uno. (Contribución de Serhiy Storchaka y Guido van Rossum en gh-100160 [https://github.com/python/cpython/issues/100160].)

- `email`: Se ha descontinuado el parámetro *isdst* en `email.utils.localtime()`. (Contribución de Alan Williams en gh-72346 [https://github.com/python/cpython/issues/72346].)

- Clases obsoletas de `importlib.abc`:

  - ○ `importlib.abc.ResourceReader`
  - ○ `importlib.abc.Traversable`
  - ○ `importlib.abc.TraversableResources`

  Use las clases `importlib.resources.abc` en su lugar:

  - ○ `importlib.resources.abc.Traversable`
  - ○ `importlib.resources.abc.TraversableResources`

  (Contribución de Jason R. Coombs y Hugo van Kemenade en gh-93963 [https://github.com/python/cpython/issues/93963].)

- `itertools` tenía un soporte indocumentado, ineficiente, históricamente lleno de errores e inconsistente para operaciones de copia, copia profunda y *pickle*. Esto se eliminará en la versión 3.14 para lograr una reducción significativa en el volumen de código y la carga de mantenimiento. (Contribución de Raymond Hettinger en gh-101588 [https://github.com/python/cpython/issues/101588].)

- **`multiprocessing`**: El método de inicio predeterminado cambiará a uno más seguro en Linux, BSDs y otras plataformas POSIX que no sean macOS donde `'fork'` es actualmente el predeterminado (gh-84559 [https://github.com/python/cpython/issues/84559]). Agregar una advertencia de tiempo de ejecución sobre esto se consideró demasiado disruptivo, ya que no se espera que la mayoría del código tenga en cuenta esto. Use las APIs **`get_context()`** o **`set_start_method()`** para especificar explícitamente cuándo su código *requires* `'fork'`. Consulte Contextos y métodos de inicio.

- **`pathlib`**: **`is_relative_to()`** y **`relative_to()`**: pasar argumentos adicionales está obsoleto.

- **`pkgutil`**: **`pkgutil.find_loader()`** and **`pkgutil.get_loader()`** now raise **`DeprecationWarning`**; use **`importlib.util.find_spec()`** instead. (Contributed by Nikita Sobolev in gh-97850 [https://github.com/python/cpython/issues/97850].)

- **`pty`**:

    ○ `master_open()`: use **`pty.openpty()`**.
    ○ `slave_open()`: use **`pty.openpty()`**.

- **`sqlite3`**:

    ○ **`version`** and **`version_info`**.
    ○ **`execute()`** y **`executemany()`** si se utilizan marcadores de posición con nombre y *parameters* es una secuencia en lugar de **`dict`**.

- **`urllib`**: **`urllib.parse.Quoter`** está obsoleto: no estaba destinado a ser una API pública. (Contribución de Gregory P. Smith en gh-88168 [https://github.com/python/cpython/issues/88168].)

Pending removal in Python 3.15

- The import system:
    - Setting `__cached__` on a module while failing to set `__spec__.cached` is deprecated. In Python 3.15, `__cached__` will cease to be set or take into consideration by the import system or standard library. (gh-97879 [https://github.com/python/cpython/issues/97879])
    - Setting `__package__` on a module while failing to set `__spec__.parent` is deprecated. In Python 3.15, `__package__` will cease to be set or take into consideration by the import system or standard library. (gh-97879 [https://github.com/python/cpython/issues/97879])
- `ctypes`:
    - La función indocumentada `ctypes.SetPointerType()` ha quedado obsoleta desde Python 3.13.
- `http.server`:
    - La clase obsoleta y raramente usada `CGIHTTPRequestHandler` ha quedado obsoleta desde Python 3.13. No existe un reemplazo directo. *Cualquier cosa* es mejor que CGI para interconectar un servidor web con un controlador de solicitudes.
    - El indicador `--cgi` a la interfaz de línea de comandos **python -m http.server** ha quedado obsoleto desde Python 3.13.
- `importlib`:
    - `load_module()` method: use `exec_module()` instead.
- `locale`:
    - La función `getdefaultlocale()` ha quedado obsoleta desde Python 3.11. La eliminación se programó originalmente para Python 3.13 (gh-90817 [https://github.com/python/cpython/issues/90817]), pero ha

    sido pospuesto hasta Python 3.15. En su lugar, utilice `getlocale()`, `setlocale()`, y `getencoding()`. (Contribuido por Hugo van Kemenade in gh-111187 [https://github.com/python/cpython/issues/111187].)

- **`pathlib`**:
    - `PurePath.is_reserved()` ha quedado obsoleto desde Python 3.13. Utilice `os.path.isreserved()` para detectar rutas reservadas en Windows.
- **`platform`**:
    - `java_ver()` ha quedado obsoleta desde Python 3.13. Esta función es útil solo para la compatibilidad con Jython, tiene una API confusa y, en gran parte, falta pruebas.
- **`sysconfig`**:
    - The *check_home* argument of `sysconfig.is_python_build()` has been deprecated since Python 3.12.
- **`threading`**:
    - `RLock()` will take no arguments in Python 3.15. Passing any arguments has been deprecated since Python 3.14, as the Python version does not permit any arguments, but the C version allows any number of positional or keyword arguments, ignoring every argument.
- **`types`**:
    - `types.CodeType`: Accessing `co_lnotab` was deprecated in **PEP 626** [https://peps.python.org/pep-0626/] since 3.10 and was planned to be removed in 3.12, but it only got a proper `DeprecationWarning` in 3.12. May be removed in 3.15. (Contributed by Nikita Sobolev in gh-101866 [https://github.com/python/cpython/issues/101866].)
- **`typing`**:
    - The undocumented keyword argument syntax for creating `NamedTuple` classes (for example, `Point = NamedTuple("Point", x=int, y=int)`) has been deprecated since Python 3.13. Use the class-based syntax or the functional syntax instead.
    - When using the functional syntax of `TypedDict`s, failing to pass a value to the *fields* parameter (`TD =

    `TypedDict("TD")`) or passing `None` (`TD = TypedDict("TD", None)`) has been deprecated since Python 3.13. Use `class TD(TypedDict): pass` or `TD = TypedDict("TD", {})` to create a TypedDict with zero field.
  - La función decoradora **`typing.no_type_check_decorator()`** ha quedado obsoleta desde Python 3.13. Después de ocho años en el módulo **`typing`**, todavía no es compatible con ningún verificador de tipos importante.
- **`wave`**:
  - Los métodos **`getmark()`**, **`setmark()`**, y **`getmarkers()`** de las clases **`Wave_read`** y **`Wave_write`** han quedado obsoletos desde Python 3.13.
- **`zipimport`**:
  - **`load_module()`** has been deprecated since Python 3.10. Use **`exec_module()`** instead. (Contributed by Jiahao Li in gh-125746 [https://github.com/python/cpython/issues/125746].)

Pending removal in Python 3.16

- The import system:
  - Setting `__loader__` on a module while failing to set `__spec__.loader` is deprecated. In Python 3.16, `__loader__` will cease to be set or taken into consideration by the import system or the standard library.
- `array`:
  - El código de formato `'u'` (`wchar_t`) ha quedado obsoleto en la documentación desde Python 3.3 y en tiempo de ejecución desde Python 3.13. En su lugar, se utiliza el código de formato `'w'` (`Py_UCS4`) para caracteres Unicode.
- `asyncio`:
  - `asyncio.iscoroutinefunction()` is deprecated and will be removed in Python 3.16; use `inspect.iscoroutinefunction()` instead. (Contributed by Jiahao Li and Kumar Aditya in gh-122875 [https://github.com/python/cpython/issues/122875].)
  - `asyncio` policy system is deprecated and will be removed in Python 3.16. In particular, the following classes and functions are deprecated:
    - `asyncio.AbstractEventLoopPolicy`
    - `asyncio.DefaultEventLoopPolicy`
    - `asyncio.WindowsSelectorEventLoopPolicy`
    - `asyncio.WindowsProactorEventLoopPolicy`
    - `asyncio.get_event_loop_policy()`

        - `asyncio.set_event_loop_policy()`

          Users should use `asyncio.run()` or `asyncio.Runner` with *loop_factory* to use the desired event loop implementation.

          For example, to use `asyncio.SelectorEventLoop` on Windows:

          ```
          import asyncio

          async def main():
              ...

          asyncio.run(main(), loop_factory=asyncio.Select
          ```

          (Contributed by Kumar Aditya in gh-127949 [https://github.com/python/cpython/issues/127949].)

- `builtins`:

    - La inversión bit a bit en tipos booleanos, `~True` o `~False` ha quedado obsoleta desde Python 3.12, ya que produce resultados sorprendentes y poco intuitivos (`-2` y `-1`). En su lugar, se utiliza `not x` para la negación lógica de un booleano. En el caso poco frecuente de que se necesite la inversión bit a bit del entero subyacente, convertir a `int` explícitamente (`~int(x)`).

- `functools`:

    - Calling the Python implementation of `functools.reduce()` with *function* or *sequence* as keyword arguments has been deprecated since Python 3.14.

- `logging`:

  Support for custom logging handlers with the *strm* argument is deprecated and scheduled for removal in Python 3.16.

Define handlers with the *stream* argument instead. (Contributed by Mariusz Felisiak in gh-115032 [https://github.com/python/cpython/issues/115032].)

- **`mimetypes`**:
    - ○ Valid extensions start with a "." or are empty for **`mimetypes.MimeTypes.add_type()`**. Undotted extensions are deprecated and will raise a **`ValueError`** in Python 3.16. (Contributed by Hugo van Kemenade in gh-75223 [https://github.com/python/cpython/issues/75223].)
- **`shutil`**:
    - ○ La excepción **`ExecError`** ha quedado obsoleta desde Python 3.14. No se ha utilizado por ninguna función en **`shutil`** desde Python 3.4, y ahora es un alias de **`RuntimeError`**.
- **`symtable`**:
    - ○ El método **`Class.get_methods`** ha quedado obsoleto desde Python 3.14.
- **`sys`**:
    - ○ La función **`_enablelegacywindowsfsencoding()`** ha quedado obsoleta desde Python 3.13. En su lugar, se utiliza la variable de entorno **`PYTHONLEGACYWINDOWSFSENCODING`**.
- **`sysconfig`**:
    - ○ The **`sysconfig.expand_makefile_vars()`** function has been deprecated since Python 3.14. Use the `vars` argument of **`sysconfig.get_paths()`** instead.
- **`tarfile`**:
    - ○ El atributo **`TarFile.tarfile`** que no se ha

documentado ni utilizado ha quedado obsoleto desde Python 3.13.

Pending removal in Python 3.17

- **`collections.abc`**:
  - **`collections.abc.ByteString`** is scheduled for removal in Python 3.17.

    Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use **`Buffer`** or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).

    `ByteString` was originally intended to be an abstract class that would serve as a supertype of both **`bytes`** and **`bytearray`**. However, since the ABC never had any methods, knowing that an object was an instance of **`ByteString`** never actually told you anything useful about the object. Other common buffer types such as **`memoryview`** were also never understood as subtypes of **`ByteString`** (either at runtime or by static type checkers).

    See **PEP 688** [https://peps.python.org/pep-0688/#current-options] for more details. (Contributed by Shantanu Jain in gh-91896 [https://github.com/python/cpython/issues/91896].)

- **`typing`**:
  - Before Python 3.14, old-style unions were implemented using the private class `typing._UnionGenericAlias`. This class is no longer needed for the implementation, but it has been

retained for backward compatibility, with removal scheduled for Python 3.17. Users should use documented introspection helpers like **`typing.get_origin()`** and **`typing.get_args()`** instead of relying on private implementation details.

- **`typing.ByteString`**, deprecated since Python 3.9, is scheduled for removal in Python 3.17.

  Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use **`Buffer`** or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).

  `ByteString` was originally intended to be an abstract class that would serve as a supertype of both **`bytes`** and **`bytearray`**. However, since the ABC never had any methods, knowing that an object was an instance of **`ByteString`** never actually told you anything useful about the object. Other common buffer types such as **`memoryview`** were also never understood as subtypes of **`ByteString`** (either at runtime or by static type checkers).

  See **PEP 688** [https://peps.python.org/pep-0688/#current-options] for more details. (Contributed by Shantanu Jain in gh-91896 [https://github.com/python/cpython/issues/91896].)

Pending removal in Python 3.18

- `decimal`:
    - The non-standard and undocumented `Decimal` format specifier `'N'`, which is only supported in the `decimal` module's C implementation, has been deprecated since Python 3.13. (Contributed by Serhiy Storchaka in gh-89902 [https://github.com/python/cpython/issues/89902].)

Pending removal in Python 3.19

- `ctypes`:
    - Implicitly switching to the MSVC-compatible struct layout by setting `_pack_` but not `_layout_` on non-Windows platforms.

Pending removal in future versions

Las siguientes APIs serán eliminadas en el futuro, aunque no hay fecha de planificación exacta para ello.

- **`argparse`**:
  - Nesting argument groups and nesting mutually exclusive groups are deprecated.
  - Passing the undocumented keyword argument *prefix_chars* to **`add_argument_group()`** is now deprecated.
  - The **`argparse.FileType`** type converter is deprecated.
- **`builtins`**:
  - Generadores: las firmas `throw(type, exc, tb)` y `athrow(type, exc, tb)` están obsoletas: utilice `throw(exc)` y `athrow(exc)` en su lugar, la firma de argumento único.
  - Actualmente Python acepta literales numéricos seguidos inmediatamente de palabras clave, por ejemplo, `0in x`, `1or x`, `0if 1else 2`. Permite expresiones confusas y ambiguas como `[0x1for x in y]` (que se puede interpretar como `[0x1 for x in y]` o `[0x1f or x in y]`). Se genera una advertencia de sintaxis si el literal numérico va seguido inmediatamente de una de las palabras clave **`and`**, **`else`**, **`for`**, **`if`**, **`in`**, **`is`** y **`or`**. En una versión futura, se cambiará a un error de sintaxis. (gh-87999 [https://github.com/python/cpython/issues/87999])
  - Compatibilidad con los métodos `__index__()` e `__int__()` que retornan un tipo que no es int: estos métodos serán necesarios para retornar una instancia de una subclase estricta de **`int`**.
  - Compatibilidad con el método `__float__()` que

    retorna una subclase estricta de `float`: será necesario que estos métodos retornen una instancia de **`float`**.
  - Compatibilidad con el método `__complex__()` que retorna una subclase estricta de `complex`: será necesario que estos métodos retornen una instancia de **`complex`**.
  - Ahora está obsoleto el paso de un número complejo como argumento *real* o *imag* en el constructor `complex()`; solo debe pasarse como un único argumento posicional. (Contribución de Serhiy Storchaka en gh-109218 [https://github.com/python/cpython/issues/109218].)
- `calendar`: Las constantes `calendar.January` y `calendar.February` han quedado obsoletas y han sido reemplazadas por `calendar.JANUARY` y `calendar.FEBRUARY`. (Contribución de Prince Roshan en gh-103636 [https://github.com/python/cpython/issues/103636].)
- `codecs`: use `open()` instead of `codecs.open()`. (gh-133038 [https://github.com/python/cpython/issues/133038])
- `codeobject.co_lnotab`: use el método `codeobject.co_lines()` en su lugar.
- `datetime`:
  - `utcnow()`: use `datetime.datetime.now(tz=datetime.UTC)`.
  - `utcfromtimestamp()`: use `datetime.datetime.fromtimestamp(timestamp, tz=datetime.UTC)`.
- `gettext`: El valor plural tiene que ser un entero.
- `importlib`:
  - El parámetro *debug_override* de `cache_from_source()` queda obsoleto: use el parámetro *optimization* en su lugar.
- `importlib.metadata`:
  - Interfaz de tupla `EntryPoints`.
  - `None` implícito en valores retornados.
- `logging`: el método `warn()` ha quedado obsoleto desde Python 3.3, use `warning()` en su lugar.
- `mailbox`: El uso del modo de entrada y texto StringIO está obsoleto, use BytesIO y el modo binario en su lugar.
- `os`: Llamando a `os.register_at_fork()` en procesos

multi-hilos.

- **`pydoc.ErrorDuringImport`**: El valor de tupla para el parámetro *exc_info* queda obsoleto, use una excepción en su lugar.
- `re`: Ahora se aplican reglas más estrictas para las referencias numéricas de grupos y los nombres de grupos en expresiones regulares. Ahora solo se aceptan secuencias de dígitos ASCII como referencia numérica. El nombre de grupo en patrones de bytes y cadenas de reemplazo ahora solo puede contener letras y dígitos ASCII y guiones bajos. (Contribución de Serhiy Storchaka en gh-91760 [https://github.com/python/cpython/issues/91760].)
- Módulos **`sre_compile`**, **`sre_constants`** y **`sre_parse`**.
- `shutil`: el parámetro *onerror* de `rmtree()` queda obsoleto en Python 3.12; use el parámetro *onexc* en su lugar.
- Opciones y protocolos de `ssl`:
    - `ssl.SSLContext` sin argumento de protocolo queda obsoleto.
    - `ssl.SSLContext`: `set_npn_protocols()` y **`selected_npn_protocol()`** quedan obsoletas: use ALPN en su lugar.
    - Opciones `ssl.OP_NO_SSL*`
    - Opciones `ssl.OP_NO_TLS*`
    - `ssl.PROTOCOL_SSLv3`
    - `ssl.PROTOCOL_TLS`
    - `ssl.PROTOCOL_TLSv1`
    - `ssl.PROTOCOL_TLSv1_1`
    - `ssl.PROTOCOL_TLSv1_2`
    - `ssl.TLSVersion.SSLv3`
    - `ssl.TLSVersion.TLSv1`
    - `ssl.TLSVersion.TLSv1_1`
- Métodos de `threading`:
    - **`threading.Condition.notifyAll()`**: use `notify_all()`.
    - **`threading.Event.isSet()`**: use `is_set()`.
    - **`threading.Thread.isDaemon()`**, `threading.Thread.setDaemon()`: use el atributo `threading.Thread.daemon`.
    - **`threading.Thread.getName()`**, `threading.Thread.setName()`: use el atributo

    `threading.Thread.name`.
  - **`threading.currentThread()`**: use **`threading.current_thread()`**.
  - **`threading.activeCount()`**: use **`threading.active_count()`**.
- **`typing.Text`** (gh-92332 [https://github.com/python/cpython/issues/92332]).
- The internal class `typing._UnionGenericAlias` is no longer used to implement **`typing.Union`**. To preserve compatibility with users using this private class, a compatibility shim will be provided until at least Python 3.17. (Contributed by Jelle Zijlstra in gh-105499 [https://github.com/python/cpython/issues/105499].)
- **`unittest.IsolatedAsyncioTestCase`**: queda deprecado retornar un valor que no sea `None` en un caso de prueba.
- Funciones deprecadas de **`urllib.parse`**: use **`urlparse()`** en su lugar
  - `splitattr()`
  - `splithost()`
  - `splitnport()`
  - `splitpasswd()`
  - `splitport()`
  - `splitquery()`
  - `splittag()`
  - `splittype()`
  - `splituser()`
  - `splitvalue()`
  - `to_bytes()`
- **`wsgiref`**: `SimpleHandler.stdout.write()` no debería hacer escrituras parciales.
- **`xml.etree.ElementTree`**: La prueba del valor de verdad de un **`Element`** está obsoleta. En una versión futura, siempre retornará `True`. En su lugar, es preferible realizar pruebas explícitas `len(elem)` o `elem is not None`.
- **`sys._clear_type_cache()`** is deprecated: use **`sys._clear_internal_caches()`** instead.

Distribuir módulos de Python

Nota

La información y orientación sobre la distribución de módulos y paquetes de Python se trasladaron a Python Packaging User Guide [https://packaging.python.org/] y al tutorial sobre packaging Python projects [https://packaging.python.org/en/latest/tutorials/packaging-projects/].

Argument Clinic Cómo Hacerlo

Nota

La documentación de *Cómo usar Argument Clinic* ha sido trasladado a la Guía de Desarrolladores de Python [https://devguide.python.org/development-tools/clinic/].

How to port Python 2 Code to Python 3

autor:

Brett Cannon

Python 2 reached its official end-of-life at the start of 2020. This means that no new bug reports, fixes, or changes will be made to Python 2 - it's no longer supported: see **PEP 373** [https://peps.python.org/pep-0373/] and status of Python versions [https://devguide.python.org/versions].

Si está buscando portar un módulo de extensión en lugar de código Python puro, consulte Portar módulos de extensión a Python 3.

The archived python-porting [https://mail.python.org/pipermail/python-porting/] mailing list may contain some useful guidance.

Since Python 3.11 the original porting guide was discontinued. You can find the old guide in the archive [https://docs.python.org/3.10/howto/pyporting.html].

Third-party guides

There are also multiple third-party guides that might be useful:

- Guide by Fedora [https://portingguide.readthedocs.io]
- PyCon 2020 tutorial [https://www.youtube.com/watch?v=JgIgEjASOlk]
- Guide by DigitalOcean [https://www.digitalocean.com/community/tutorials/how-to-port-python-2-code-to-python-3]
- Guide by ActiveState [https://www.activestate.com/blog/how-to-migrate-python-2-applications-to-python-3]

`aifc` — Read and write AIFF and AIFC files

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

The last version of Python that provided the `aifc` module was Python 3.12 [https://docs.python.org/3.12/library/aifc.html].

`asynchat` — Asynchronous socket command/response handler

Deprecated since version 3.6, removed in version 3.12.

This module is no longer part of the Python standard library. It was removed in Python 3.12 after being deprecated in Python 3.6. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

Applications should use the **`asyncio`** module instead.

The last version of Python that provided the **`asynchat`** module was Python 3.11 [https://docs.python.org/3.11/library/asynchat.html].

`asyncore` — Asynchronous socket handler

Deprecated since version 3.6, removed in version 3.12.

This module is no longer part of the Python standard library. It was removed in Python 3.12 after being deprecated in Python 3.6. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

Applications should use the **`asyncio`** module instead.

The last version of Python that provided the **`asyncore`** module was Python 3.11 [https://docs.python.org/3.11/library/asyncore.html].

`audioop` — Manipulate raw audio data

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

The last version of Python that provided the **`audioop`** module was Python 3.12 [https://docs.python.org/3.12/library/audioop.html].

`cgi` — Common Gateway Interface support

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

A fork of the module on PyPI can be used instead: legacy-cgi [https://pypi.org/project/legacy-cgi/]. This is a copy of the cgi module, no longer maintained or supported by the core Python team.

The last version of Python that provided the `cgi` module was Python 3.12 [https://docs.python.org/3.12/library/cgi.html].

`cgitb` — Traceback manager for CGI scripts

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

A fork of the module on PyPI can now be used instead: legacy-cgi [https://pypi.org/project/legacy-cgi/]. This is a copy of the cgi module, no longer maintained or supported by the core Python team.

The last version of Python that provided the `cgitb` module was Python 3.12 [https://docs.python.org/3.12/library/cgitb.html].

`chunk` — Read IFF chunked data

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

The last version of Python that provided the **`chunk`** module was Python 3.12 [https://docs.python.org/3.12/library/chunk.html].

`crypt` — Function to check Unix passwords

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

Applications can use the **`hashlib`** module from the standard library. Other possible replacements are third-party libraries from PyPI: legacycrypt [https://pypi.org/project/legacycrypt/], bcrypt [https://pypi.org/project/bcrypt/], or argon2-cffi [https://pypi.org/project/argon2-cffi/]. These are not supported or maintained by the Python core team.

The last version of Python that provided the **`crypt`** module was Python 3.12 [https://docs.python.org/3.12/library/crypt.html].

`distutils` — Building and installing Python modules

Deprecated since version 3.10, removed in version 3.12.

This module is no longer part of the Python standard library. It was removed in Python 3.12 after being deprecated in Python 3.10. The removal was decided in **PEP 632** [https://peps.python.org/pep-0632/], which has migration advice [https://peps.python.org/pep-0632/#migration-advice].

The last version of Python that provided the **`distutils`** module was Python 3.11 [https://docs.python.org/3.11/library/distutils.html].

Program frameworks

This chapter is no longer maintained, and the modules it contained have been moved to their respective topical documentation.

- **`cmd`** — Command Line Interface Libraries
- **`shlex`** — Unix Specific Services
- **`turtle`** — Graphical User Interfaces with Tk

`imghdr` — Determine the type of an image

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

Possible replacements are third-party libraries from PyPI: filetype [https://pypi.org/project/filetype/], puremagic [https://pypi.org/project/puremagic/], or python-magic [https://pypi.org/project/python-magic/]. These are not supported or maintained by the Python core team.

The last version of Python that provided the **`imghdr`** module was Python 3.12 [https://docs.python.org/3.12/library/imghdr.html].

`imp` — Access the import internals

Deprecated since version 3.4, removed in version 3.12.

This module is no longer part of the Python standard library. It was removed in Python 3.12 after being deprecated in Python 3.4.

The removal notice includes guidance for migrating code from **`imp`** to **`importlib`**.

The last version of Python that provided the **`imp`** module was Python 3.11 [https://docs.python.org/3.11/library/imp.html].

`mailcap` — Mailcap file handling

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

The last version of Python that provided the `mailcap` module was Python 3.12 [https://docs.python.org/3.12/library/mailcap.html].

`msilib` — Read and write Microsoft Installer files

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

The last version of Python that provided the **`msilib`** module was Python 3.12 [https://docs.python.org/3.12/library/msilib.html].

`nis` — Interface to Sun’s NIS (Yellow Pages)

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

The last version of Python that provided the **`nis`** module was Python 3.12 [https://docs.python.org/3.12/library/nis.html].

`nntplib` — NNTP protocol client

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

The last version of Python that provided the **`nntplib`** module was Python 3.12 [https://docs.python.org/3.12/library/nntplib.html].

`ossaudiodev` — Access to OSS-compatible audio devices

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

The last version of Python that provided the **`ossaudiodev`** module was Python 3.12 [https://docs.python.org/3.12/library/ossaudiodev.html].

`pipes` — Interface to shell pipelines

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

Applications should use the `subprocess` module instead.

The last version of Python that provided the `pipes` module was Python 3.12 [https://docs.python.org/3.12/library/pipes.html].

`smtpd` — SMTP Server

Deprecated since version 3.6, removed in version 3.12.

This module is no longer part of the Python standard library. It was removed in Python 3.12 after being deprecated in Python 3.6. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

A possible replacement is the third-party aiosmtpd [https://pypi.org/project/aiosmtpd/] library. This library is not maintained or supported by the Python core team.

The last version of Python that provided the **`smtpd`** module was Python 3.11 [https://docs.python.org/3.11/library/smtpd.html].

`sndhdr` — Determine type of sound file

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

Possible replacements are third-party modules from PyPI: filetype [https://pypi.org/project/filetype/], puremagic [https://pypi.org/project/puremagic/], or python-magic [https://pypi.org/project/python-magic/]. These are not supported or maintained by the Python core team.

The last version of Python that provided the `sndhdr` module was Python 3.12 [https://docs.python.org/3.12/library/sndhdr.html].

`spwd` — The shadow password database

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

A possible replacement is the third-party library python-pam [https://pypi.org/project/python-pam/]. This library is not supported or maintained by the Python core team.

The last version of Python that provided the `spwd` module was Python 3.12 [https://docs.python.org/3.12/library/spwd.html].

`sunau` — Read and write Sun AU files

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

The last version of Python that provided the `sunau` module was Python 3.12 [https://docs.python.org/3.12/library/sunau.html].

`telnetlib` — Telnet client

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

Possible replacements are third-party libraries from PyPI: telnetlib3 [https://pypi.org/project/telnetlib3/] or Exscript [https://pypi.org/project/Exscript/]. These are not supported or maintained by the Python core team.

The last version of Python that provided the **`telnetlib`** module was Python 3.12 [https://docs.python.org/3.12/library/telnetlib.html].

`uu` — Encode and decode uuencode files

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

The last version of Python that provided the `uu` module was Python 3.12 [https://docs.python.org/3.12/library/uu.html].

`xdrlib` — Encode and decode XDR data

Deprecated since version 3.11, removed in version 3.13.

This module is no longer part of the Python standard library. It was removed in Python 3.13 after being deprecated in Python 3.11. The removal was decided in **PEP 594** [https://peps.python.org/pep-0594/].

The last version of Python that provided the `xdrlib` module was Python 3.12 [https://docs.python.org/3.12/library/xdrlib.html].